Elmer S. Chuquiyauri Saldivar

Ho Nexus
EDI TORIAL

DESARROLLO DE SOLUCIONES CON JAVA Y BASE DE DATOS SQL Y NOSQL

PRIMERA EDICIÓN DIGITAL

Elmer S. Chuquiyauri Saldivar

Ho Nexus
EDITORIAL

PRIMERA EDICIÓN DIGITAL

DESARROLLO DE SOLUCIONES CON JAVA Y BASE DE DATOS SQL Y NOSQL

DESARROLLO DE SOLUCIONES CON JAVA Y BASE DE DATOS SQL Y NOSQL

Editor de contenido:
Diseño de cubierta: Ho Nexus

1ª edición digital, febrero 2026

Editado por:

© HO NEXUS E.I.R.L.
Dirección legal: Urb. Paseo del Mar Mz L4, Lt 33
Nuevo Chimbote, Santa, Ancash - Perú
Correo electrónico; ed.honexus@gmail.com
teléfono: 978 653 152
https://books.honexus.org
DOI: https://doi.org//10.70504/978-612-99293-1-6

Depósito Legal: 2026-00396
ISBN: 978-612-99293-1-6

Revisión por pares:
Este libro (o monografía) fue sometido a evaluación de pares mediante el sistema de doble ciego (doubleblinded review), garantizando la calidad, pertinencia, ética y rigor académico de la obra, conforme a los estándares internacionales de revisión científica y las políticas editoriales de Ho Nexus.

ÍNDICE

PRÓLOGO

En el mundo de la ingeniería y carreras similares, aprender a desarrollar software se ha vuelto algo fundamental. No es solo porque la industria tecnológica crece sin parar, sino, sobre todo, porque hoy es indispensable entender cómo se crean esas herramientas digitales que resuelven problemas de la vida real.

En la universidad, enseñar programación tiene una dificultad que se repite: cómo hacer que lo práctico y lo teórico vayan de la mano. Muchas veces, los alumnos logran que un programa funcione, pero les cuesta entender por qué detrás del código. Y es que programar va mucho más allá de escribir líneas de comando; se trata de cultivar un tipo de razonamiento especial, uno que permita desarmar un problema, diseñar una solución paso a paso y analizar sus consecuencias.

Precisamente de esa inquietud nació este libro. No es un simple recetario de comandos, sino un intento por sentar bases conceptuales sólidas para quienes aprenden Java y los enfoques clave del desarrollo de software. Las explicaciones de fondo se mezclan con ejemplos concretos, para que el lector pueda ver claramente cómo se conectan la teoría y la práctica.

Los temas están ordenados para avanzar poco a poco. Se arranca desde lo básico de la programación estructurada y se va subiendo hacia la programación orientada a objetos, el diseño de interfaces gráficas y cómo conectar todo con bases de datos. La idea es que el aprendizaje sea natural, sin saturar con tecnicismos desde el primer capítulo.

Por la manera en que está escrito y el nivel de detalle que maneja, este libro se dirige, en primer lugar, a quienes forman parte del ámbito universitario: estudiantes que están dando sus primeros pasos y profesores que buscan material para sus clases. Sin embargo, el texto no

se dirige únicamente a ese perfil de lector. También puede ser una herramienta útil para desarrolladores que ya están trabajando en proyectos reales, pero que sienten la necesidad de volver sobre los fundamentos para consolidar su comprensión. Quizás para refrescar conceptos olvidados, quizás para afianzar desde los cimientos lo que saben sobre Java y el desarrollo de software, con el fin de construir sobre una base más sólida.

RESUMEN

Esta obra plantea un aprendizaje progresivo para construir aplicaciones digitales. Arranca desde la base misma: la lógica y el razonamiento que dan forma a cualquier programa. Una vez asentados esos fundamentos, da el salto a la acción, tomando a Java como el lenguaje clave para transformar las ideas en código funcional. El proceso también incluye una parte clave en cualquier aplicación real: cómo manejar la información, explorando tanto las bases de datos tradicionales (relacionales) como las alternativas más modernas (no relacionales).

El contenido está organizado para que el lector vaya escalando de nivel poco a poco. La idea es que se pueda ver claramente cómo encajan todas las piezas del desarrollo de software. No son solo explicaciones; a lo largo del texto, se incluyen ejemplos concretos y casos de estudio que cumplen una función muy práctica. Su objetivo es tender un puente entre la teoría que se estudia y el momento en que uno se sienta frente al código para resolver un problema real.

Pensado principalmente como un recurso para cursos universitarios de programación y sistemas de información, este libro apunta a algo más que enseñar una tecnología. Busca ayudar a construir una mentalidad, un marco de pensamiento que permita, más adelante, analizar cualquier desafío y diseñar soluciones informáticas con una base sólida y bien fundamentada.

Palabras clave: Programación, Java, algoritmos, programación orientada a objetos, bases de datos, ingeniería de software.

INTRODUCCIÓN

Hoy en día, saber desarrollar software es prácticamente una habilidad clave, tanto en el mundo profesional como dentro de las aulas. Como cada vez dependemos más de los sistemas digitales, hay una demanda constante de personas que no solo entiendan estas herramientas, sino que también sean capaces de idearlas y construirlas de manera eficiente. En este panorama, Java sigue siendo una de las herramientas favoritas, tanto para enseñar a programar como para levantar proyectos de envergadura. Su solidez y esa particularidad de poder funcionar en distintos sistemas han hecho que se mantenga vigente con los años. Pero claro, aprender Java no se reduce a memorizar su sintaxis; el verdadero reto está en captar la lógica que hay detrás, los principios que guían su diseño y su aplicación.

Este manuscrito plantea un método que no separa el “porqué” del “cómo”. A medida que se avanza en los capítulos, se van desgranando temas como la programación estructurada, la orientada a objetos, cómo dar forma a interfaces gráficas y la manera de gestionar datos. La meta no es hacer un repaso enciclopédico de cada tecnología, sino más bien ofrecer una mirada unificada, que muestre cómo todo se entrelaza en el proceso de crear software. Con ese fin, la obra aspira a ser un recurso de estudio que ayude al lector a construir una comprensión más sólida de los conceptos esenciales que entran en juego cada vez que se desarrolla una aplicación.

CAPÍTULO 1. ALGORITMOS Y FUNDAMENTOS DE PROGRAMACIÓN EN JAVA

1.1 ALGORITMOS

Un algoritmo es una secuencia lógica de pasos para resolver un problema. Y esto es independiente de cualquier lenguaje de programación, es algo universal.

EJEMPLO N°01-001

Convertir un número real, que representa horas, a su equivalente en horas, minutos, segundos y decimos de segundo

DESARROLLO

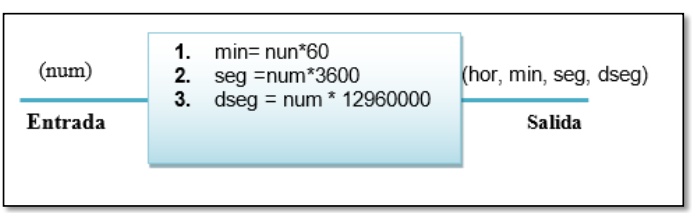
1. ANALISIS:

1.1. Entrada : Ingrese un valor de un número real: num

1.2. Salida : Imprime horas: hor
Imprime minutos: min
Imprime segundos: seg
Imprime decimos de segundo: dseg

1.3. Formula : min=num*60; seg=num*3600; dseg=num*12960000

1.4 Grafica :



2. ALGORITMO:

Inicio

1. Ingresamos un numero: num
2. Convertimos a las unidades pedidas:

min= nunl*60

seg =num*3600

dseg = num * 12960000

3. Escribir los resultados:

El resultado en horas es: hor

El resultado en minutos es: min

El resultado en segundos es: seg

El resultado en decimos de segundos es: dseg

Fin

EJEMPLO N°01-002

Estimar el número de páginas de un texto que puede almacenarse en la memoria de un computador, considerando un promedio de 300 palabras por página y 10 caracteres por palabra.

Asumir que un carácter ocupa un (1) byte. El tamaño de la memoria del computador debe ingresarse expresado en Kbyte.

DESARROLLO

1. ANALISIS:

1.1. Entrada **:** tamaño de la memoria: tammen

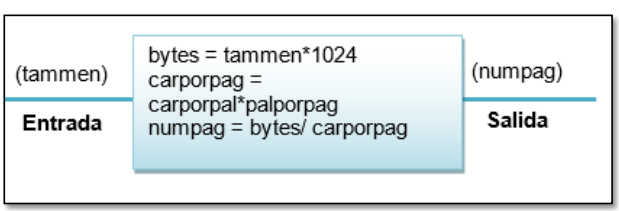
1.2. Salida : número de página: numpag

1.3. Formula : bytes = tammen*1024

carporpag = carporpal*palporpag

numpag = bytes/ carporpag

1.4 Grafica :



2. ALGORITMO:

Inicio

1 Definimos constantes:

carporpal = 10

palporpag = 300

2 Ingresamos tamaño de la memoria en Kbyte: tammen

3 Se convierte tammen en Kbytes a bytes:

bytes = tammen*1024

4 Se calcula el número de caracteres:

carporpag = carporpal*palporpag

5 Se calcula el número de páginas:

numpag = bytes/ carporpag

6 El número de páginas es: numpag

Fin

EJEMPLO N°01-003

Convertir grados centígrados a grados Fahrenheit a partir de la siguiente fórmula:

GradosFar = 1.8 * GradosCent + 32.

DESARROLLO

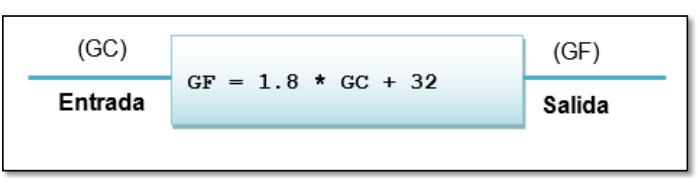
1. ANALISIS:

1.1. Entrada: valor en grados centígrados: GC

1.2. Salida: valor en grados farenheit es: GF

1.3. Formula: GradosFar = 1.8 * GradosCent + 32

1.4 Grafica:



2 ALGORITMO:

Inicio

1. Ingresar valor en grados centígrados: GC
2. Lo resolvemos por la formula:

   GradosFar = 1.8 * GradosCent + 32
3. Imprima en grados farenheir es: GF

Fin

1.2 CODIFICACION

Teniendo el algoritmo para solucionar un problema y si esto a su vez necesita utilizar la computadora para procesar los datos, entonces es necesario escribir un conjunto de comandos para que la computadora entienda el algoritmo. A este conjunto de comandos se le llama programa. Un programa entonces es una secuencia de comandos que obedece a un algoritmo para procesar datos utilizando la computadora, por lo que se necesita de un lenguaje de programación en lo cual se tienen una sintaxis definido y sentencias que permite hacer lo plasmado en un algoritmo.

1.2.1 LENGUAJE DE PROGRAMACION JAVA

Si se busca una definición de un lenguaje, en internet se encontrará muchas:

“Se llama lenguaje (del provenzal lenguatge y este del latín lingua) a cualquier sistema de comunicación estructurado, para el que existe un contexto de uso y ciertos principios combinatorios formales. Existen contextos tanto naturales como artificiales.

- ✓ *El lenguaje humano se basa en la capacidad de los seres humanos para comunicarse por medio de signos (usualmente secuencias sonoras, pero también gestos y señas, así como signos gráficos). Principalmente lo hacemos utilizando el signo lingüístico. Aun así, hay diversos tipos de lenguaje. El lenguaje humano puede estudiarse en cuanto a su desarrollo desde dos puntos de vista*

complementarios: la ontogenia, que remite al proceso de adquisición del lenguaje por el ser humano, y la filogenia.

- ✓ *El lenguaje animal se basa en el uso de señales sonoras, visuales, y olfativas, a modo de signos, para señalar a un referente o un significado diferente de dichas señales. Dentro del lenguaje animal están los gritos de alarma, el lenguaje de las abejas, etc.*
- ✓ *Los lenguajes formales son construcciones artificiales humanas, que se usan en matemática y otras disciplinas formales, incluyendo lenguajes de programación. Estas construcciones tienen estructuras internas que comparten con el lenguaje humano natural, por lo que pueden ser en parte analizados con los mismos conceptos que éste"1*.

Entonces un lenguaje de programación en un lenguaje formal, ya que es creado por el hombre con el propósito de que permite la comunicación con una máquina de obtención, almacenamiento y procesamiento de datos.
Un lenguaje de programación tiene un conjunto de alfabetos, palabras, sintaxis. Que permite escribir el código de un programa, para que luego se encargue de transformarlo en 0s y 1s, que los componentes físicos de una máquina que procesa datos entienda.
Algunos lenguajes de programación:

- ✓ Java
- ✓ Python
- ✓ C++
- ✓ C#
- ✓ PHP
- ✓ JasvaScript
- ✓ Visual Fox Pro
- ✓ Visual Basic

JAVA

Java es un lenguaje de programación orientado a objetos, de alto nivel, de propósito general, multiplataforma, creado por Sun Microsystems en 1995. Es un lenguaje portable que se puede ejecutar en cualquier sistema operativo.

El código se compila en un formato especial llamado bytecode y esto se ejecuta en una máquina virtual java(JVM).

Los archivos que contiene el código en java tienen la extensión .java, estos archivos se compila utilizando el compilador javac, generando archivos con extensión .class que contiene el bytecode. Se puede ver en la IMAGEN N° 01-0001, el proceso de complicación y ejecución de un programa en Java.

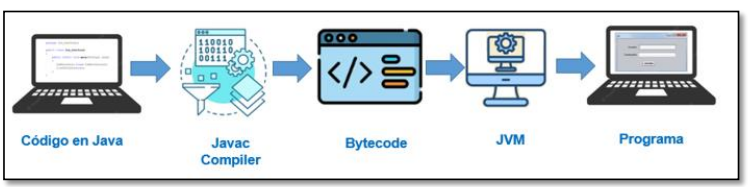


IMAGEN N° 01-0001: Proceso de complicación y ejecución de un programa en Java

INSTALACION

Para su instalación se descarga desde la página oficial: https://www.java.com/

Se mostrará la página como en la IMAGEN N° 01-0002.

IMAGEN N° 01-0002: Página oficial de descarga de Java

Descargar el JDK de la página:

https://www.oracle.com/pe/java/technologies/downloads/

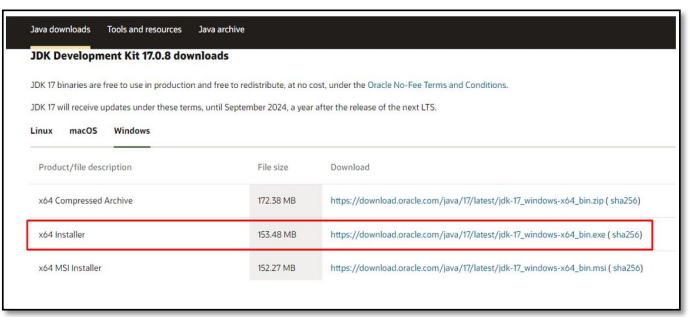


IMAGEN N° 01-0003: Pagina para descargar JDK

1.3 ENTORNO DE DESARROLLO NetBeans

Netbeans es un IDE (Integrated Development Environment) o entorno de desarrollo integrado, gratuito y de código abierto, que permite editar código fuente, junto con recursos de construcción automáticos y depurador (compilador y un intérprete). Facilita el proceso de diseño de aplicaciones de escritorio, web o móviles.

Página oficial: https://netbeans.apache.org

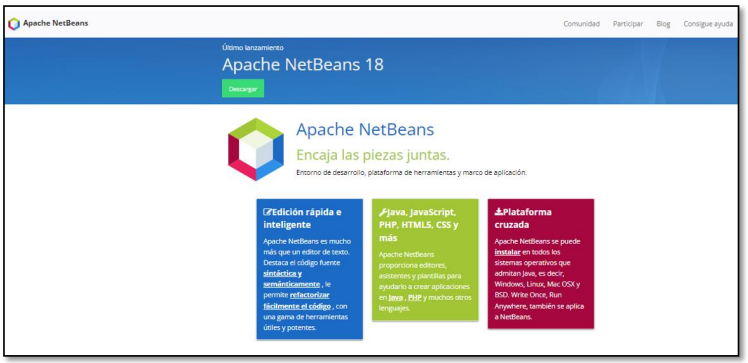


IMAGEN N° 01-0004: Página oficial de NetBeans

Para su instalación descargar de la página oficial y escoger la versión, en este caso se descarga la versión 16 tal como se muestra en la IMAGEN N° 01-0005

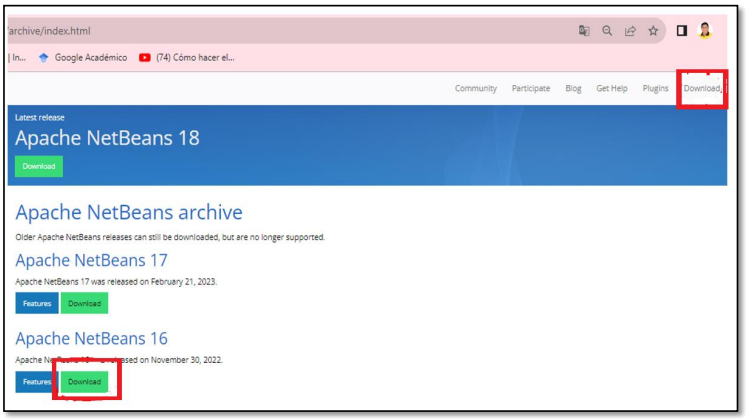


IMAGEN N° 01-0005: Versiones de descarga del NetBeans

En la IMAGEN N° 01-0006 se muestra el entorno de desarrollo NetBeans IDE 16

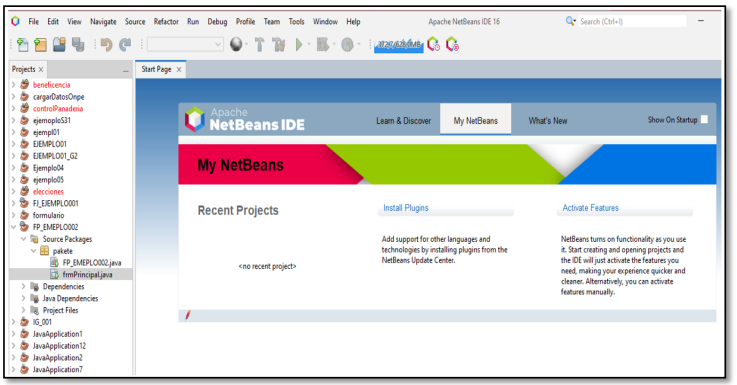


IMAGEN N° 01-0006: Entorno de desarrollo de NetBeans IDE 16

Para iniciar a programar del menú principal seleccionar la opción File > New Project tal como se muestra en la IMAGEN N° 01-0007 e IMAGEN N° 01-0008

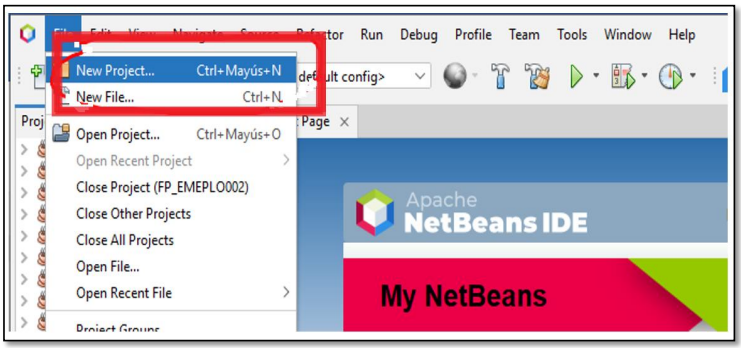


IMAGEN N° 01-0007: Creación de un nuevo proyecto en NetBeans IDE 16

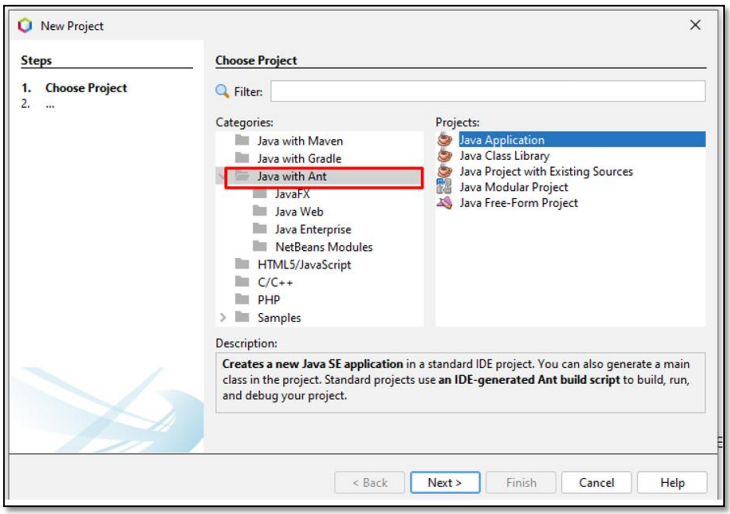


IMAGEN N° 01-0008: Selección del tipo de proyecto

En la IMAGEN N° 01-0009 se muestra la ventana para asignar un nombre al proyecto luego y cambiar la ubicación de este, click en Finish.

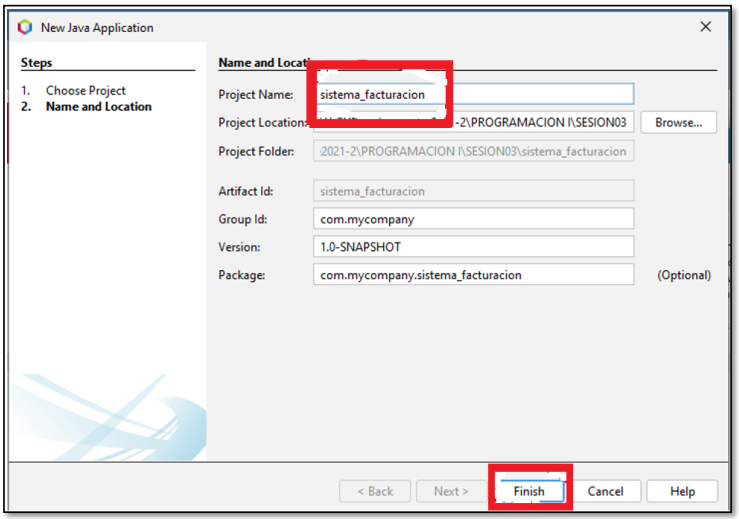


IMAGEN N° 01-0009: Ventana para asignar nombre, ubicación entre otros del proyecto

Una vez realizado todos los pasos anteriores, se crea los componentes del nuevo proyecto tal como se muestra en la IMAGEN N° 01-0010, creándose un paquete con el nombre que se le dio al proyecto y se crear una clase por defecto, en este caso la clase con el mismo nombre del proyecto, que se podría cambiar de acuerdo como se disponga. Para programas simples todo el código se podría insertar dentro del método ***main***:

```
public static void main(String[] args) {
    // TODO code application logic here
}
```

En todo proyecto la clase principal deben tener obligatoriamente este método, porque por ahí es en donde se comienza a ejecutar el programa.

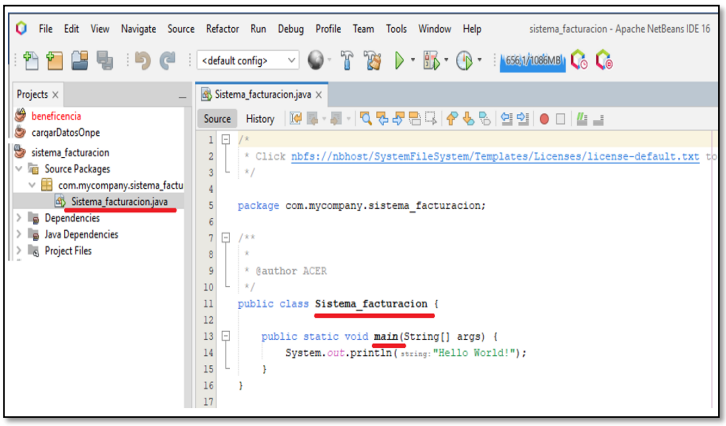


IMAGEN N° 01-0010: Al crear un proyecto se crea una clase con un método **main**

Codificando el EJEMPLO N°01-003, programa que calcula la suma de n números naturales, para lo cual se necesita dos variables: cantidad de números(***n***) y suma de los n números(***s***), así mismo para la lectura de datos desde el teclado se utiliza **Java Scanner** importando desde: ***import java.util.Scanner;***

```
import java.util.Scanner;
public class Sistema_facturacion {
public static void main(String[] args) {
int n;//variable n para almacenar cantidad de numeros
int s;//variable s para almacenar la suma de los n numeros
Scanner leer=new Scanner(System.in);//leer datos desde el teclado
System.out.print("INRESAR LA CANTIDAD DE NUMEROS: ");
n=leer.nextInt();//se lee el dato ingresado por el teclado
//se calcula la suma de n numeros por formula
s=n*(n+1)/2;
```

```
        //se imprime en pantalla s
        System.out.print("La suma de los<<"+n+">> numeros es: "+s);
        }
}
```

1.4 JAVA SCANNER PARA LECTURA DE DATOS

Para leer datos desde algún dispositivo que podría ser el teclado se utiliza el Scanner, para lo cual se tiene que importar en la parte en donde se utilizara:

```
import java.util.Scanner;
```

Así mismo para crear el flujo de lectura de datos se instancia la clase Scanner:

```
Scanner leer=new Scanner(System.in);//leer datos desde el teclado
```

En la sentencia anterior se crea el objeto "***leer***" asociado al teclado representado por System.in, luego de ello se puede leer datos desde el teclado utilizando los métodos:

- ***leer*** **.nextByte()** para leer un dato de tipo byte.
- ***leer*** **.nextShort()** para leer un dato de tipo short.
- ***leer*** **.nextInt()** para leer un dato de tipo int.
- ***leer*** **.nextLong()** para leer un dato de tipo long.
- ***leer*** **.nextFloat()** para leer un dato de tipo float.
- ***leer*** **.nextDouble()** para leer un dato de tipo double.
- ***leer*** **.nextBoolean()** para leer un dato de tipo boolean.
- ***leer*** **.nextLine()** para leer un String hasta encontrar un salto de línea.
- ***leer*** **.next()** para leer un String hasta el primer delimitador, generalmente hasta un espacio en blanco o hasta un salto de línea.

Para ejecutar el programa hacer clic en el icono del triángulo verde tal como se muestra en la IMAGEN N° 01-0011

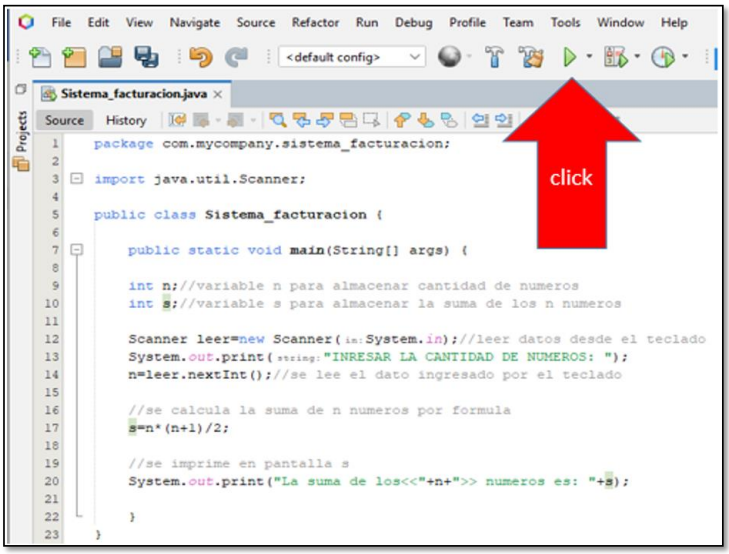


IMAGEN N° 01-0011: Para ejecutar el programa, clic en el icono del triángulo verde

Al ejecutar el programa nos solicita que se ingrese un número, luego muestra la suma de estos, IMAGEN N° 01-0012:

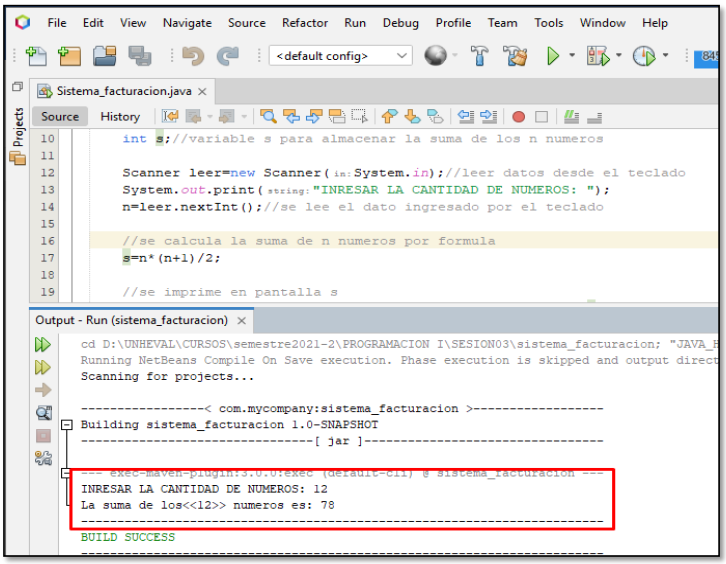


IMAGEN N° 01-0012: Programa en ejecución

1.5 VARIABLES EN JAVA

VARIABLES

Las variables en Java son un espacio de memoria en el computador en el que se almacena datos. La estructura para definir una variable es el siguiente:

```
[privacidad] tipo_variable identificador;
```

Java es un lenguaje de tipado estático, por lo que las variables tienen que tener un tipo de dato primitivo (int, long, float, double, boolean, etc.) o de clase y un nombre de identificador.

En el caso de que las variables sean propiedades de objetos tendrán un alcance o privacidad.

TIPOS DE DATOS PRIMITIVOS EN JAVA

En java se tiene los siguientes tipos de datos primitivos: byte, short, int, long, float, double, boolean, char.

byte: Representa un tipo de dato de 8 bits con signo. De tal manera que puede almacenar los valores numéricos de -128 a 127 (ambos inclusive).

short: Representa un tipo de dato de 16 bits con signo. De esta manera almacena valores numéricos de -32.768 a 32.767.

int: Es un tipo de dato de 32 bits con signo para almacenar valores numéricos. Cuyo valor mínimo es -2^{31} y el valor máximo $2^{31}-1$.

long: Es un tipo de dato de 64 bits con signo que almacena valores numéricos entre -2^{63} a $2^{63}-1$

float: Es un tipo dato para almacenar números en coma flotante con precisión simple de 32 bits.

double: Es un tipo de dato para almacenar números en coma flotante con doble precisión de 64 bits.

boolean: Sirve para definir tipos de datos booleanos. Es decir, aquellos que tienen un valor de true o false. Ocupa 1 bit de información.

char: Es un tipo de datos que representa a un carácter Unicode sencillo de 16 bits.

EJEMPLOS DE DECLARACION DE VARIABLES PRIMITIVOS

```
int n;
int s;
float nota01 =10 ;
float nota02=15;
float promedio=(nota01+nota02)/2;
String cadena = "Hola";//no es una variable primitivo
long decimal = 2.4;
boolean encendido = true;
```

Las variables son utilizadas como propiedades dentro de los objetos.

```
class Circulo {
    private long radio;
    private long area;
}
```

1.6 SENTENCIAS SIMPLES

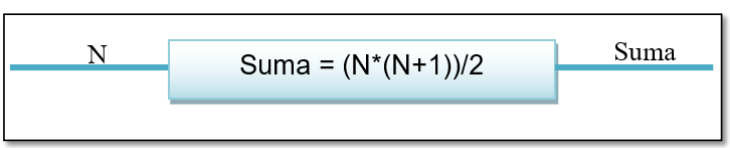
Las sentencias simples son código en donde se aplican directamente formulas o salidas inmediatas, para esto se analiza el problema y se investiga si es que se requiere de alguna fórmula matemática u otro proceso y/o procedimiento.

EJEMPLO N°01-004

Determinar la suma de los N primeros números enteros de acuerdo a la siguiente fórmula: Suma = (N*(N+1))/2

DESARROLLO

1. ANALISIS:
1.1. Entrada: Ingrese un número entero: N
1.2. Salida: imprime la suma de los n números: Suma
1.3. Formula: Suma = (N*(N+1))/2
1.4 Grafica:



2. ALGORITMO:

Inicio
1. Ingrese un numero entero: N
2. Calculamos la suma:
   Suma = (N*(N+1))/2
3. Imprime la suma por pantalla: Suma

Fin

3. CODIGO EN JAVA:

Lectura de datos utilizando el ***BufferedReader*** que se encuentra en: java.io.*; y que solo tiene el método ***.readLine()*** para leer todo tipo de datos:

```
//-----------------------------------------------------------------------------------------------------------
  //PROGRAMA QUE CALCULA LAS SUMA DE LOS N PRIMEROS NUMEROS
//-----------------------------------------------------------------------------------------------------------
    package javaapplication1;
    import java.io.*;
    class Main
    {
      public static void main(String args[])
     {
      int N=0; //variable para guardar el numero ingresado por teclado
            int Suma;//variable de tipo entero para guardar la suma
            //se crea el buffer para el ingreso de datos
      BufferedReader leer=new BufferedReader(new
    InputStreamReader(System.in));

    try
     {
            System.out.println("Ingresar un numero:");
            N=Integer.parseInt(leer.readLine());//se ingresa la cantidad de
    numeros
                                                    // y se convierte a
    entero
     }catch(IOException ex)
    {
            System.err.println("error: se ");
     }
                    //se calcula la suma
```

```
                Suma=N*(N+1)/2;
                //Se imprime por pantalla la suma de los N numeros
                System.out.println("La suma de los<<"+N+">>es:"+Suma);

    }
}
```

En el código de este ejemplo se utiliza el **BufferedReader**, que nos permite crear un flujo de entrada de datos. Para la entrada de datos se puede utilizar el **Scanner**, que se encuentra en el paquete: java.util.*; que es mucho más sencillo de usar, el ejemplo anterior seria así:

```
//-------------------------------------------------------------------------------------
//PROGRAMA QUE CALCULA LA SUMA DE n NUMEROS ENTEROS
//-------------------------------------------------------------------------------------
import java.util.*;
class Main
{
public static void main(String args[])
 {
                Scanner s=new Scanner(System.in);
                int n=0,suma=0;
                System.out.println("Ingresar un numero");
                n=s.nextInt();
                suma=n*(n+1)/2;
                System.out.println(suma);
    }
}
```

EJEMPLO N°01-005

Calcular el interés generado por un capital depositado durante cierta cantidad de periodos a una tasa de interés determinada y expresada en porcentaje.

Aplicar la siguiente fórmula:

Monto = Capital * (1 + tasa/100) $^{numero\ de\ periodos}$

Interés = Monto – Capital

Donde, tasa es el porcentaje de interés por periodo. Un periodo puede ser un día, un año, etc.

DESARROLLO

1. ANALISIS:

1.1. Entrada : Ingresar capital: C

Ingresamos tasa de interés: T

Ingrese el número de periodos: np

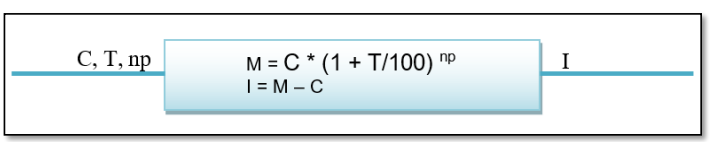
1.2. Salida : interés ganado: I

1.3. Formula: aplicamos la siguiente fórmula:

$M = C * (1 + T/100)^{np}$

$I = M – C$

1.4 Grafica :



2. ALGORITMO

Inicio

1. Ingresar el capital: C
2. Ingrese la tasa de interés: T
3. Ingrese el número de periodos: np
4. Calculamos el monto:

$M = C * (1 + T/100)^{np}$

5. Calculamos el interés:

I = M – C

6. Se imprime el interés ganado por pantalla: I

Fin

3. CODIGO EN JAVA:

```
//-----------------------------------------------------------------------------------------------------
//PROGRAMA QUE CALCULA EL INTERES GENERADO  POR UN
//CAPITAL CON UNA TASA DE INTERES EN UN PERIODO DETERMINADO
//-----------------------------------------------------------------------------------------------------
package javaapplication1;
import java.io.*;
class Main
{
        public static void main(String args[])
        {
                double C=0, T=0, np=0,I=0,M=0;//se declaran variables
               //se crea el buffer para el ingreso de datos
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
                try
                {

                System.out.println("Ingresar el capital");
                C=Double.parseDouble(in.readLine());//se ingresa el capital

        System.out.println("Ingresar la tasa de interes");
                 T=Double.parseDouble(in.readLine());//se ingresa la tasa

        System.out.println("Ingresar el numero de periodos");
                 np=Double.parseDouble(in.readLine());//num. de periodos
```

```
            }
            catch(IOException ex)
            {
                System.err.println("error: se ");
            }
            //se calcula el monto
            M=C*Math.pow((1+T/100), np);

            // se calcula el interes
            I=M-C;

            //Se imprime por pantalla el interes
            System.out.println("El interes generado del capital<< "+C+" >>
es:"+I);

        }
}
```

En este ejemplo se ha utilizado el **Math.pow(base, exponente)** que permite realizar potencias con una base definida y un exponente dado, por ejemplo si se quiere realizar la siguiente potencia 2^3, seria así: **Math.pow(2, 3) .**
En la clase **Math,** se encuentran operaciones, funciones y constantes:

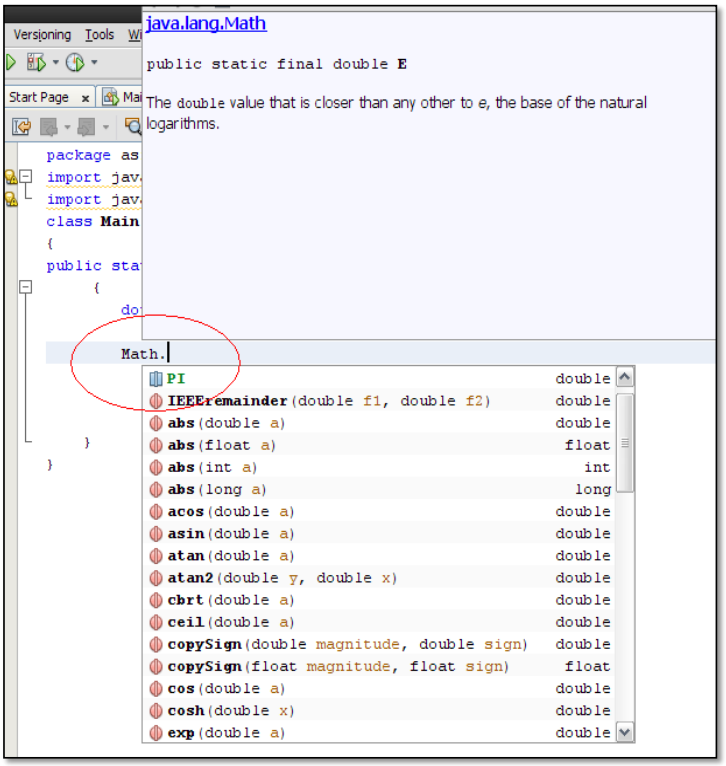


IMAGEN N° 01-0013: Métodos que contienen la clase Math

1.7 SENTENCIAS DE CONTROL DE FLUJO SIMPLE

Las sentencias de control de flujo simple son los puntos en donde el programa decide tomar un camino u otro dependiendo del valor de verdad de la proposición. Esto responde a:

si(proposición)

inicio

```
        //sentencia si la proposición es verdadera
    final
De lo contrario
    inicio
        //sentencia si la proposición es falsa
            final
```

En java:

```
if(proposición)
{
    //sentencia si la proposición es verdadera
}
else
{
    //sentencia si la proposición es falsa
}
```

EJEMPLO N°01-006

Realizar el algoritmo y programa en java para encontrar el mayor de 5 números ingresados por el teclado

DESARROLLO

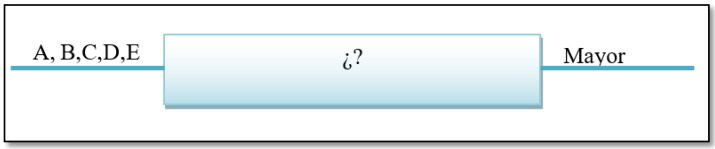
1. ANALISIS:

1.1. Entrada : Ingresar 5 números reales: A, B, C, D, E

1.2. Salida : el mayor de 5 números: Mayor

1.3. Formula: no requiere formula:

1.4 Grafica :



2. ALGORITMO

1. Ingresar 5 números: A, B C, D, E
2. Asignar a la variable Mayor el primer numero
   Mayor=A
3. Si Mayor<B
   Mayor=B
   Si Mayor<C
   Mayor=C
   Si Mayor<D
   Mayor=D
   Si Mayor<E
   Mayor=E

4. Se imprime el número mayor por pantalla: Mayor

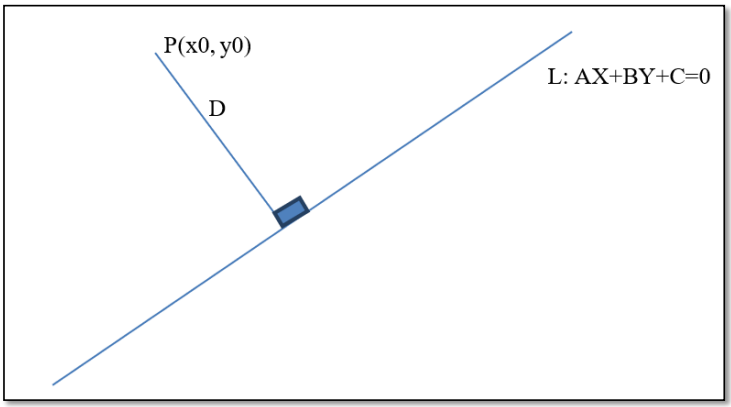
3. CODIGO EN JAVA:

```
//-------------------------------------------------------------------------------------------------
 * PROGRAMA QUE CALCULA EL MAYOR DE 5 NUMEROS
//-------------------------------------------------------------------------------------------------
package fundamentosjava;

import java.util.Scanner;

public class FundamentosJava {


   public static void main(String[] args) {

      float A=0, B=0, C=0, D=0, E=0, Mayor=0;

      Scanner leer=new Scanner(System.in);


      System.out.println("Ingresar 5 numeros: ");

      A=leer.nextFloat();

      B=leer.nextFloat();

      C=leer.nextFloat();
```

```
        D=leer.nextFloat();
        E=leer.nextFloat();

       Mayor=A;//se asigna el primer número a mayor
       //se saca el mayor valor
       if(Mayor<B)
          Mayor=B;
       if(Mayor<C)
          Mayor=C;
       if(Mayor<D)
          Mayor=D;
       if(Mayor<E)
      Mayor=E;
       //se imprime por pantalla el mayor valor
       System.out.println("El mayor valor es:"+Mayor);
    }
}
```

EJEMPLO N°01-007

Realizar el algoritmo y programa en java para encontrar la distancia de un punto a una recta, tal como se muestra en la siguiente figura.



DESARROLLO

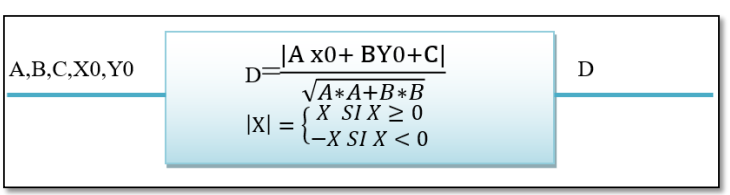
1. ANALISIS:

1.1. Entrada : Coef. de la recta: A, B, C
Coordenadas del punto: X0, Y0
1.2. Salida : La distancia del punto a la recta: D
1.3. Formulas: aplicamos la siguiente fórmula:

$$D=\frac{|A\,x0+\,BY0+C|}{\sqrt{A*A+B*B}}$$

$$|X| = \begin{cases} X \;\; SI\; X \geq 0 \\ -X \; SI\; X < 0 \end{cases}$$

1.4 Grafica :



2. ALGORITMO

Inicio

1. Ingresar los coef. De la ecuación: A,B,C
2. Ingresar el punto:X0, Y0
3. Calculo de distancia

E1= $A\,X0 + BY0 + C$

Si E1>=0

$$D=\frac{E1}{\sqrt{A*A+B*B}}$$

De lo contrario

$$D=\frac{-1*E1}{\sqrt{A*A+B*B}}$$

4. Imprimir por pantalla la distancia: D

Final

3. CODIGO EN JAVA:

```
//-----------------------------------------------------------------------------------------------------------
* PROGRAMA QUE CALCULA LA DISTANCIA DE UN PUNTO A UNA RECTA
//-----------------------------------------------------------------------------------------------------------
package fundamentosjava;

import java.util.Scanner;

public class FundamentosJava {


   public static void main(String[] args) {
      float A=0,B=0,C=0;//variables para los coef de la recta
      float X0=0,Y0=0;//variables para las coordenadas del punto
      double D=0;//variable para almacenar la distancia
      double E1=0;//variable temporal


      Scanner leer =new Scanner(System.in);
      System.out.println("Ingresar los coef. de la recta  AX+BY+C=0 ");


      //se lee los coeficientes
      A=leer.nextFloat();
      B=leer.nextFloat();
```

```
        C=leer.nextFloat();

        //se lee el punto
        System.out.println("Ingresar las coordenadas del punto");
        X0=leer.nextFloat();
        Y0=leer.nextFloat();

        //se evalua el punto en la ecuacion
        E1=A*X0+B*Y0+C;
        //se verifica el signo del valor abs |A x0+ BY0+C|
        if(E1>=0)
           D=E1/(Math.sqrt(A*A+B*B));
        else
           D=-1*E1/(Math.sqrt(A*A+B*B));

        System.out.println("La distancia del punto<<"+X0+","+Y0+">> a la recta:
"+A+"X+"+B+"Y+"+C+"=0  es:"+D);
        }
}
```

EJEMPLO N°01-008

Realizar el algoritmo y programa en java que permita encontrar las raíces de una ecuación cuadrática: $ax^2+bx+c=0$

DESARROLLO

1. ANALISIS:

1.1 Entrada:

Coeficientes de las variables: a, b, c

1.2 Salida:

Raíces de la ecuación: x1, x2

1.3 Formula:

$$x1,2 = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

2. ALGORITMO

Inicio

1. Ingresar los coeficientes de las variables: a, b, c
2. Si a=0

Imprimir: "la ecuación no es cuadrática"

3 De lo contrario

Inicio

3.1 Calcular la discriminante de la ecuación cuadrática
d=b^2-4ac

3.2 Calcular las raíces:
Si (d<0)
Inicio
d=-1*(d)
imprimir: -b/2a +sqrt(d)/2a+"I"
imprimir: -b/2a -sqrt(d)/2a+"i"

Final
De lo contrario
Inicio

imprimir: -b/2a +sqrt(d)/2a
imprimir: -b/2a -sqrt(d)/2a
Final

Final

Final

3. CODIGO EN JAVA:

```
//-------------------------------------------------------------------------------------------------
PROGRAMA QUE CALCULA LAS RAICES DE UNA ECUACION CUADRATICA
//-------------------------------------------------------------------------------------------------
package fundamentosjava;

import java.util.Scanner;

public class FundamentosJava {


    public static void main(String[] args) {
```

```
double a=0,b=0,c=0,d=0;
double x1=0,x2=0;
Scanner leer=new Scanner(System.in);

System.out.println("PROGRMA QUE CALCULA LAS");
System.out.println("RAICES DE UNA ECUACION CUADRATICA:");
System.out.println("ax2+bx+c=0");
System.out.println("Ingresar los coeficientes ");

//se lee los coeficientes
//y se almacena en las variables
a=leer.nextDouble();
b=leer.nextDouble();
c=leer.nextDouble();

//se valida el valor de a
if(a==0)
  System.out.println("No es una ecuacion cuadratica");
else{
  //se calcula la discriminante
  d=b*b-4*a*c;
  //calculo de las raices
  if(d<0){//si la discriminante es menor que 0, raices complejas
   d=-d;
   System.out.println(-b/(2*a)+"+"+Math.sqrt(d)/(2*a)+"i");
   System.out.println(-b/(2*a)+"-"+Math.sqrt(d)/(2*a)+"i");
  }
  else{ //si la discriminante es mayoro igual a 0, raíces reales
   System.out.println(-b/(2*a)+Math.sqrt(d)/(2*a));
```

```
            System.out.println(-b/(2*a)+Math.sqrt(d)/(2*a));
        }
       }
    }
}
```

1.8 SENTENCIAS DE CONTROL MULTIPLE

Las sentencias de control simple nos permite poner puntos de control en nuestros programas, pero que pasa si por ejemplo queremos realizar un programa de tal manera que al ingresar un número entre 1 y 12 nos imprime por pantalla el mes correspondiente en letras, esto se podría realizar utilizando varios **ifs(si),** el asunto es que por cada if el programa estaría comparando, esto requiere recursos de computación, por lo que nuestro algoritmo no sería tan eficaz y eficiente, también se podría mejorar utilizando **if**s anidados dentro de los **elses:**

```
if(mes==1)
        //imprimir mes:ENERO
else
        if(mes==2)
                //imprimir mes:FEBRERO
        else
                if(mes==3)
                        //imprimir mes:MARZO
                else
                        . . .
```

Pero como se puede ver, es bien engorroso realizar este tipo de controles. Para mejorar esto se utiliza sentencias de control múltiple respondiendo a; "**en caso que:**", para lo cual en java se ha implementado la **instrucción switch** que

permite ejecutar unos u otros bloques de instrucciones según sea el valor de una cierta expresión. Su estructura es:

switch (<expresión>**)**

{

case *<valor1>**:*** *<bloque1>*
<siguienteAcción>
case *<valor2>**:*** *<bloque2>*
<siguienteAcción>
...
default: *<bloqueDefault>*
<siguienteAcción>

}

El significado de esta instrucción es el siguiente: se evalúa **<expresión>**. Si su valor es **<valor1>** se ejecuta el **<bloque1>**, si es **<valor2>** se ejecuta **<bloque2>**, y así para el resto de valores especificados. Si no es igual a ninguno de esos valores y se incluye la rama **default**, se ejecuta el **<bloqueDefault>**; pero si no se incluye se pasa directamente a ejecutar la instrucción siguiente al **switch**.

Los valores indicados en cada rama del **switch** han de ser expresiones constantes que produzcan valores de algún tipo básico entero, de una enumeración, de tipo **char** o de tipo **string**. Además, no puede haber más de una rama con el mismo valor.

En realidad, aunque todas las ramas de un **switch** son opcionales siempre se tiene que incluir al menos una. Además, la rama **default** no tiene porqué aparecer la última si se usa, aunque es recomendable que lo haga para facilitar la legibilidad del código.

El elemento marcado como <siguienteAcción> colocado tras cada bloque de instrucciones indica qué es lo que ha de hacerse tras ejecutar las instrucciones del bloque que lo preceden.

EJEMPLO N°01-009

Determinar el nombre correspondiente a un número de mes y además la estación a la que pertenece, considerando 3 meses completos por estación, asi ismo considerar que:

Verano: diciembre, enero, febrero.
Otoño: marzo, abril, mayo.
Invierno: junio, julio, agosto.
Primavera: setiembre, octubre, noviembre.

DESARROLLO

1. ANALISIS:

1.1 Entrada:

Ingresar el número de mes: n

1.2 Salida:

Mes en letras y la estación a la que pertenece: mesl, estación

1.3 Formula:

Sin formula

2. ALGORITMO

Inicio

1. Ingresar número de mes: n
2. .En **CASO** que mes **SEA**

   1.
   Imprimir “ENERO”, estación: “VERANO”
   2.
   Imprimir “FEBRERO”, estación: “VERANO”
   3.
   Imprimir “MARZO” , estación: “OTOÑO”
   4.
   Imprimir “ABRIL” , estación: “OTOÑO”
   5.

Imprimir “MAYO” , estación: “OTOÑO”

6.

Imprimir “JUNIO” , estación: “INVIERNO”

7.

Imprimir “JULIO” , estación: “INVIERNO”

8.

Imprimir “AGOSTO” , estación: “INVIERNO”

9.

Imprimir “SETIEMBRE” , estación: “PRIMAVERA”

10.

Imprimir “OCTUBRE” , estación: “PRIMAVERA”

11.

Imprimir “NOVIEMBRE” , estación: “PRIMAVERA”

12.

Imprimir “DICIEMBRE” , estación: “VERANO”

FIN del **CASO**

Fin

3. CODIGO EN JAVA:

```
//-----------------------------------------------------------------------------------------------------
PROGRAMA QUE VISUALIZA UN MES INGRESADO EN NUMEROS A  LETRAS
//-----------------------------------------------------------------------------------------------------
package fundamentosjava;
import java.util.Scanner;
public class FundamentosJava {

    public static void main(String[] args) {
     int n;
     Scanner leer=new Scanner(System.in);
     System.out.println("Ingresar un valor:");
     n=leer.nextInt();
     switch(n){
```

```
case 1:
   System.out.println("ENERO");
   System.out.println("ESTACION: VERANO");
   break;
case 2:
   System.out.println("FEBRERO");
   System.out.println("ESTACION: VERANO");
   break;
case 3:
   System.out.println("MARZO");
   System.out.println("ESTACION: OTOÑO");
   break;
case 4:
   System.out.println("ABRIL");
   System.out.println("ESTACION: OTOÑO");
   break;
case 5:
   System.out.println("MAYO");
   System.out.println("ESTACION: OTOÑO");
   break;
case 6:
   System.out.println("JUNIO");
   System.out.println("ESTACION: INVIERNO");
   break;
case 7:
   System.out.println("JULIO");
   System.out.println("ESTACION: INVIERNO");
   break;
case 8:
```

```
            System.out.println("AGOSTO");
            System.out.println("ESTACION: INVIERNO");
            break;
        case 9:
            System.out.println("SETIEMBRE");
            System.out.println("ESTACION: PRIMAVERA");
            break;
        case 10:
            System.out.println("OCTUBRE");
            System.out.println("ESTACION: PRIMAVERA");
            break;
        case 11:
            System.out.println("NOVIEMBRE");
            System.out.println("ESTACION: PRIMAVERA");
            break;
        case 12:
            System.out.println("DICIEMBRE");
            System.out.println("ESTACION: VERANO");
            break;
        default:
            System.out.println("Dato no valido");
            break;
    }
  }
}
```

EJEMPLO N°01-010

Se desea diseñar un programa para controlar el motor eléctrico de la estación de bombeo representada en la IMAGEN N° 01-0014. Las entradas de este sistema vienen representadas mediante las variables lógicas:

NM: es la salida de un sensor que está en el depósito; vale "1" si el agua supera el Nivel Máximo, " 0" en el caso contrario.

NS: es la salida de otro sensor que se encuentra situado por debajo del anterior; vale "1" si el agua supera el Nivel de Seguridad, "0" en caso contrario.

IN: señal que vale "1" durante la noche y "0" durante el día.

CM: es una señal de Control Manual; cuando valga "0" no tendrá ningún efecto, "1" significará que se ha de poner en marcha la bomba si el depósito está por debajo del Nivel Máximo.

Especificaciones del diseño:

a) La bomba funcionará durante la noche si el depósito está por debajo del Nivel máximo.

b) La bomba funcionará de día cuando el agua esté por debajo del Nivel de Seguridad.

c) El funcionamiento de la bomba se produce cuando la señal B toma el valor "1" y se para cuándo toma el valor "0".

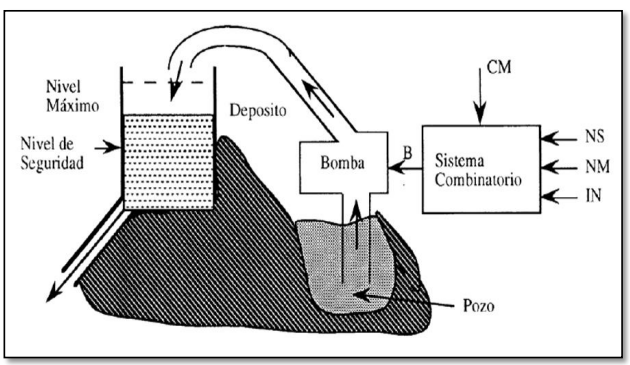


IMAGEN N° 01-0014: Sistema de bombeo automático

DESARROLLO

Para desarrollo del programa tener en cuenta que, las entradas desde el sensor son binario, ósea **0s y 1s**, entonces se analiza todas las posibles combinaciones que podría tener o transmitir los sensores y en respuesta a ello tenemos que generar una salida binaria, para hacer funcionar a la bomba.

1. ANALISIS:

1.1 Entrada:

Señal del sensor de nivel de seguridad: NS

Señal del sensor de nivel Máximo: NM

Señal del sensor que nos indica el día o la noche: IN

Señal de control manual: CM

1.2 Salida:

Señal para hacer funcionar la bomba: B

1.3 Formula:

ENTRADA		ENTRADA				SALIDA
CONCATENADO	DECIMAL	CM	IN	NM	NS	B
0	0	0	0	0	0	1
1	1	0	0	0	1	1
10	2	0	0	1	0	0
11	3	0	0	1	1	0
100	4	0	1	0	0	1
101	5	0	1	0	1	0
110	6	0	1	1	0	0
111	7	0	1	1	1	0
1000	8	1	0	0	0	1
1001	9	1	0	0	1	1
1010	10	1	0	1	0	0
1011	11	1	0	1	1	0
1100	12	1	1	0	0	1
1101	13	1	1	0	1	1
1110	14	1	1	1	0	0
1111	15	1	1	1	1	0

TABLA N° 01-0001: Posibles entradas

2. ALGORITMO

Inicio

1. Escanear las entradas de los sensores: CN, IN, NM, NS
2. Procesamiento de las entradas
3. entrada= CN+" "+IN+" "+NM+" "+NS
   entradaI=int(entrada)

En **CASO** que entradaI **SEA**

0: B=1
1: B=1
10: B=0
11: B=0
100: B=1
101: B=0
110: B=0
111: B=0
1000: B=1
1001: B=1
1010: B=0
1011: B=0
1100: B=1
1101: B=1
1110: B=0
1111: B=0

FIN del **CASO**

4. Enviar la señal de salida: B

Fin

3. CODIGO EN JAVA:

```
//-----------------------------------------------------------------------------------------------
PROGRAMA QUE CONTROLA EL SITEMA DE BOMBEO
//-----------------------------------------------------------------------------------------------
package fundamentosjava;
import java.util.Scanner;
public class FundamentosJava {
   public static void main(String[] args) {
        //señal de entrada de los sensores
      short CN, IN, NM, NS;
      short B;//señal de control de la bomba
      String cad;
      Scanner entrada=new Scanner(System.in);
      System.out.println("Scaneando las entradas:");

      //se Escanea los sensores
      CN=entrada.nextShort();
      IN=entrada.nextShort();
      NM=entrada.nextShort();
      NS=entrada.nextShort();

      //concatenamos las entradas
      cad=CN+""+IN+""+NM+""+NS;
          switch(Integer.parseInt(cad))
      {
         case 0:
            B=1;
```

```
    break;
case 1:
    B=1;
    break;
case 10:
    B=0;
    break;
case 11:
    B=0;
    break;
case 100:
    B=1;
    break;
case 101:
    B=0;
    break;
case 110:
    B=0;
    break;
case 111:
    B=0;
    break;
case 1000:
    B=1;
    break;
case 1001:
    B=1;
    break;
case 1010:
```

```
            B=0;
            break;
        case 1011:
            B=0;
            break;
        case 1100:
            B=1;
            break;
        case 1101:
            B=1;
            break;
        case 1110:
            B=0;
            break;
        case 1111:
            B=0;
            break;
        default:
            B=0;
            break;
      }
    //se envia la señal para el funcionamiento de la bomba
    if(B==0)
      System.out.println("Bomba apagado");
    else
      System.out.println("Bomba funcionando");
  }
}
```

EJEMPLO N°01-011

Realizar un programa para calcular el promedio ponderado de un alumno, considerando que lleva 4 cursos, cada curso tiene 4 notas (PP (promedio de prácticas), PT (promedio de trabajos académicas), EM (examen de medio curso peso 2), EF (Examen final de peso 2), ES (Examen sustitutorio que remplaza a la nota mínima de EM o EF)); en cada curso se tiene 3 practicas, 2 trabajos. El código del alumno tiene el siguiente formato: **XXXXYYZZZZ**; en donde los cuatro primeros caracteres representan el año de ingreso a la universidad; YY representa la modalidad de ingreso, siendo: 01(Examen de Admisión), 02(CEPRE), 03(Deportista calificado), 04(Primeros puestos), 05(Traslado Externo), 06(Violencia del Terrorismo); ZZZZ representa el puesto en que ha ingresado.

El código del curso debería ser autogenerado, teniendo en cuenta el siguiente formato: **XYYZZZ**; en donde X representa la primera letra de la escuela; YY representa el código de la escuela y ZZZ es un número secuencial.

Cada curso pertenece a una línea de formación: FB (Formación Básica); C(Carrera); H(Humanidades); S(Sistemas).

El reporte que debería generar tiene el siguiente formato:

ESCUELA: INGENIERIA DE SISTEMAS (13)
CODIGO: 1988010001
ALUMNO: GARCIA CALDERON, MARIA
SEMESTRE: 2024-2
PROMEDIO PONDERADO: 10(DIEZ)

CURSO	DESCRIPCION	CILCO	CRED.	NOTA	NOTA LETRA
I13001	MATEMATICA DISCRETAS (FB)	III	3	15	QUINCE
I13005	ECUACIONES DIFERENCIALES (FB)	IV	4	05	CINCO
I13006	PROGRAMACION I(C)	II	5	10	DIEZ
I13018	LENGUAJE (H)	I	2	15	QUINCE

DESARROLLO

1. ANALISIS:

Para desarrollar el programa, necesitamos de algunos datos que nos servirán para dar formato al código del alumno y los códigos de los cursos; como el código del alumno tiene un formato especial **XXXXYYZZZZ**, entonces primero tenemos

que pedir los datos del alumno: apellidos y nombres del alumno, año de ingreso, modalidad de ingreso, y el puesto en que ingreso a la universidad, además se tienen que pedir la escuela académico a la que ha ingresado con su respectivo código de escuela. Con respecto al curso tenemos que pedir: nombre del curso, ciclo, créditos y línea de formación.

Para calcular el promedio ponderado, primero tenemos que calcular los promedios de cada curso, para ello pedimos las notas correspondientes de las prácticas, trabajos, examen de medio curso, examen final y examen sustitutorio.

1.1. Entrada :

- Datos del alumno: Apellidos y nombres (apno), año de ingreso (ai), Modalidad de ingreso(mi), Puesto ingreso(pi), EAP a la que ingreso(eap), Código de la EAP(ceap).
- Datos de los Cursos:
  - ✓ Nombre del curso 1(nc1), ciclo(ci1), créditos (c1), línea de formación(lf1), nota practica 1(np11), nota practica 2(notap12), nota practica 3(notap13), nota trabajo 1(nt11), nota trabajo 2(nt12), examen medio curso(em1) examen final (ef1), examen sustitutorio (es1)
  - ✓ Nombre del curso 2(nc2), ciclo(ci2), créditos (c2), línea de formación(lf2), nota practica 1(np21), nota practica 2(notap22), nota practica 3(notap23), nota trabajo 1(nt21), nota trabajo 2(nt22), examen medio curso(em2) examen final (ef2), examen sustitutorio (es2)
  - ✓ Nombre del curso 3(nc3), ciclo(ci3), créditos (c3), línea de formación(lf3), nota practica 1(np31), nota practica 2(notap32), nota practica 3(notap33), nota trabajo 1(nt31), nota trabajo 2(nt32), examen medio curso(em3) examen final (ef3), examen sustitutorio (es3)
  - ✓ Nombre del curso 4(nc4), ciclo(ci4), créditos (c4), línea de formación(lf4), nota practica 1(np41), nota practica 2(notap42), nota practica 3(notap43), nota trabajo 1(nt41), nota trabajo 2(nt42), examen medio curso(em4) examen final (ef4), exame

n sustitutorio (es4)

1.2. Salida :

ESCUELA: INGENIERIA DE SISTEMAS (13)
CODIGO: 1988010001
ALUMNO: GARCIA CALDERON, MARIA
SEMESTRE: 2012-1
PROMEDIO PONDERADO: 10(DIEZ)

CURSO	DESCRIPCION	CILCO	CRED.	NOTA	NOTA LETRA
I13001	MATEMATICA DISCRETAS (FB)	III	3	15	QUINCE
I13005	ECUACIONES DIFERENCIALES (FB)	IV	4	05	CINCO
I13006	PROGRAMACION I(C)	II	5	10	DIEZ
I13018	LENGUAJE (H)	I	2	15	QUINCE

1.3. Formulas:

```
//para el cálculo del promedio del curso 1
pp1= (np11+np12+ np13)/3
pt1= (nt11+nt12)/2
pc1= (pp1+pt1+2*em1+2*ef1)/6
//para el cálculo del promedio del curso 2
pp2= (np21+np22+ np23)/3
pt2= (nt21+nt22)/2
pc2= (pp2+pt2+2*em2+2*ef2)/6
//para el cálculo del promedio del curso 3
pp3= (np31+np32+ np33)/3
pt3= (nt31+nt32)/2
pc3= (pp3+pt3+2*em3+2*ef3)/6
//para el cálculo del promedio del curso 4
pp4= (np41+np42+ np43)/3
pt4= (nt41+nt42)/2
pc4= (pp4+pt4+2*em4+2*ef4)/6
```

En todos los casos se supone que el examen sustitutorio remplaza a la nota mínima entre el examen de medio curso o examen final

```
//promedio ponderado
pp = (c1*pc1+c2*pc2+c3*pc3+c4*pc4)/(c1+ c2+ c3+
```


c4)

1.4 Grafica :

Sin grafico

2.- ALGORITMO

1. Ingresar los datos del alumno:

Apellidos y nombres: apno
Año de ingreso: ai
Modalidad de ingreso: mi
Puesto de ingreso: pi
EAP a la que ingreso: eap
Código de la EAP: ceap

2. Generar el código del alumno:

codigo= ai+" "+ mi+" "+pi

3. Ingresar datos de los cursos:

Nombre del curso: nc1
Ciclo: ci1
Créditos: c1
Línea de Formación: lf1
Nota de práctica 1:np11
Nota de práctica 2:np12
Nota de práctica 3:np13
Nota de trabajo 1:nt11
Nota de trabajo 2:nt12
Examen de medio curso: em1
Examen final: ef1
Examen Sustitutorio: es1

3.1 Generar código del curso 1

codigoCurso1=subString(eap,0,1)+ ceap+"001"

Nombre del curso: nc2
Ciclo: ci2
Créditos: c2

Línea de Formación: lf2

Nota de práctica 1:np21

Nota de práctica 2:np22

Nota de práctica 3:np23

Nota de trabajo 1: nt21

Nota de trabajo 2: nt22

Examen de medio curso: em2

Examen final: ef2

Examen Sustitutorio: es2

3.2 Generar código del curso 2

codigoCurso2=subString(eap,0,1) + ceap+"002"

Nombre del curso: nc3

Ciclo: ci3

Créditos: c3

Línea de Formación: lf3

Nota de práctica 1:np31

Nota de práctica 2:np32

Nota de práctica 3:np33

Nota de trabajo 1: nt31

Nota de trabajo 2: nt32

Examen de medio curso: em3

Examen final: ef3

Examen Sustitutorio: es3

3.3 Generar código del curso 3

codigoCurso3=subString(eap,0,1) + ceap+"003"

Nombre del curso: nc4

Ciclo: ci4

Créditos: c4

Línea de Formación: lf4

Nota de práctica 1:np41

Nota de práctica 2:np42
Nota de práctica 3:np43
Nota de trabajo 1:nt41
Nota de trabajo 2:nt42
Examen de medio curso: em4
Examen final: ef4
Examen Sustitutorio: es4

3.4 Generar código del curso 4

codigoCurso4=subString(eap,0,1) + ceap+"004"

4. Calculo de los promedios de los cursos

```
//para el cálculo del promedio del curso 1
                pp1= (np11+np12+ np13)/3
                pt1= (nt11+nt12)/2
                //-------------------------------------------------------------------------------
                //se sustituye el examen sustitutorio a la menor nota de:em o ef
                si em1< ef1
                  si es1>em1
                        em1= es1
                de lo contrario
                  si es1>ef1
                        ef1= es1
                //-------------------------------------------------------------------------------
                pc1= (pp1+pt1+2*em1+2*ef1)/6

                //para el cálculo del promedio del curso 2
                pp2= (np21+np22+ np23)/3
                pt2= (nt21+nt22)/2
//-------------------------------------------------------------------------------
                //se sustituye el examen sustitutorio a la menor nota de:em o ef
                si em2< ef2
```

```
                si es2>em2
                    em2= es2
                de lo contrario
                si es2>ef2
                    ef2= es2
                //-------------------------------------------------------------------------------
                pc2= (pp2+pt2+2*em2+2*ef2)/6

//para el cálculo del promedio del curso 3
                pp3= (np31+np32+ np33)/3
                pt3= (nt31+nt32)/2
//-------------------------------------------------------------------------
                //se sustituye el examen sustitutorio a la menor nota de:em o ef
                si em3< ef3
                si es3>em3
                    em3= es3
                de lo contrario
                si es3>ef3
                    ef3= es3
                //-------------------------------------------------------------------------------
                pc3= (pp3+pt3+2*em3+2*ef3)/6
//para el cálculo del promedio del curso 4
                pp4= (np41+np42+ np43)/3
                pt4= (nt41+nt42)/2
//-------------------------------------------------------------------------
                //se sustituye el examen sustitutorio a la menor nota de:em o ef
                si em4< ef4
                si es4>em4
                    em4= es4
                de lo contrario
                si es4>ef4
                    ef4= es4
                //-------------------------------------------------------------------------------
```

pc4= (pp4+pt4+2*em4+2*ef4)/6

5. Imprimir los promedios de los cursos según formato

3. CODIGO EN JAVA:

```
//-----------------------------------------------------------------------------------------
PROGRAMA QUE CALCULA EL PROMEDIO PONDERADO DE CURSOS
Y LOS VISUALIZA DE ACUERDO A UN FORMATO
//-----------------------------------------------------------------------------------------
package fundamentosjava;
import java.util.Scanner;
public class FundamentosJava {
 //METODO QUE CAMBIA UN NUMERO DEL 1 AL 20 EN LETRAS
 public static String numeroaLetra(int ppi)
 {
   String ppl="";
   switch(ppi)
   {
    case 0:
              ppl="CERO";
              break;

    case 1:
              ppl="UNO";
              break;
    case 2:
               ppl="DOS";
                break;
    case 3:
               ppl="TRES";
               break;
```

```
case 4:
        ppl="CUATRO";
        break;
case 5:
        ppl="CINCO";
        break;
case 6:
        ppl="SEIS";
        break;
case 7:
        ppl="SIETE";
        break;
case 8:
        ppl="OCHO";
        break;
case 9:
        ppl="NUEVE";
        break;
case 10:
        ppl="DIEZ";
        break;
case 11:
        ppl="ONCE";
        break;
case 12:
        ppl="DOCE";
        break;
case 13:
        ppl="TRECE";
```

```
            break;
case 14:
            ppl="CATORCE";
            break;
case 15:
            ppl="QUINCE";
            break;
case 16:
           ppl="DIEZ Y SEIS";
           break;
 case 17:
           ppl="DIEZ Y SIETE";
           break;
 case 18:
           ppl="DIEZ Y OCHO";
           break;
 case 19:
            ppl="DIEZ Y NUEVE";
            break;
 case 20:
            ppl="VEINTE";
            break;

 }

   return ppl;
}

public static void main(String[] args) {
```

```
//-----------------------------------------
        //variables para datos de alumno
//--------------------------------------
        String codigo="",apno="",ai,mi,pi,eap="",ceap="",se="";
//-----------------------------------------------
        //variables para datos de curso
//-----------------------------------------------
        String nc1="",nc2="",nc3="",nc4="";
        String ci1="",ci2="",ci3="",ci4="";
        float c1=0,c2=0,c3=0,c4=0;
        String lf1="",lf2="",lf3="",lf4="";

        float np11=0,np12=0,np13=0;
        float np21=0,np22=0,np23=0;
        float np31=0,np32=0,np33=0;
        float np41=0,np42=0,np43=0;

        float nt11=0,nt12=0;
        float nt21=0,nt22=0;
        float nt31=0,nt32=0;
        float nt41=0,nt42=0;

        float em1=0,em2=0,em3=0,em4=0;
        float ef1=0,ef2=0,ef3=0,ef4=0;
        float es1=0,es2=0,es3=0,es4=0;
        String codigoCurso1="";
        String codigoCurso2="";
        String codigoCurso3="";
        String codigoCurso4="";
```

```
float pc1,pc2,pc3,pc4;

Scanner entrada=new Scanner(System.in);
try{
  //Ingreso de datos de alumno;
  System.out.println("Apellidos y nombres:");
  apno =entrada.nextLine();
  System.out.println("Año de Ingreso:");
  ai =entrada.next();
  System.out.println("Modalidad de Ingreso:");
  mi =entrada.nextLine();
  System.out.println("Puesto de Ingreso:");
  pi =entrada.next();
  System.out.println("EAP:");
  eap =entrada.nextLine();
  System.out.println("Codigo de la EAP:");
  ceap =entrada.next();
  System.out.println("Semestre de estudio:");
  se =entrada.nextLine();
  //se genera el codigo del alumno
  codigo=ai+""+ mi+""+pi;

  //-----------------------------------------------------
  //Ingreso de datos de curso
  //-----------------------------------------------------
  //curso 1
  System.out.println("Nombre del curso 1:");
  nc1 =entrada.nextLine();
```

```
System.out.println("Ciclo");
ci1 =entrada.nextLine();
System.out.println("Creditos:");
c1 = entrada.nextInt();
System.out.println("linea de formacion:");
lf1 =entrada.nextLine();
System.out.println("Nota practica 1:");
np11 =entrada.nextInt();
System.out.println("Nota practica 2:");
np12 =entrada.nextInt();
System.out.println("Nota practica 3:");
np13 =entrada.nextFloat();
System.out.println("Nota trabajo 1:");
nt11 =entrada.nextFloat();
System.out.println("Nota trabajo 2:");
nt12 =entrada.nextFloat();
System.out.println("Nota de Examen de Medio curso:");
em1 =entrada.nextFloat();
System.out.println("Nota de Examen final:");
ef1 =entrada.nextFloat();
System.out.println("Nota Examen sustitutorio:");
es1 =entrada.nextFloat();

//se genera el codigo del curso
codigoCurso1=eap.substring(0,1)+ceap+"0001";
//-----------------------------------------------------
//-----------------------------------------------------
//curso 2
System.out.println("Nombre del curso 2:");
```

```
nc2 =entrada.nextLine();
System.out.println("Ciclo");
ci2 =entrada.nextLine();
System.out.println("Creditos:");
c2 = Float.parseFloat(entrada.nextLine());
System.out.println("linea de formacion:");
lf2 =entrada.nextLine();
System.out.println("Nota practica 1:");
np21 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota practica 2:");
np22 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota practica 3:");
np23 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota trabajo 1:");
nt21 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota trabajo 2:");
nt22 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota de Examen de Medio curso:");
em2 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota de Examen final:");
ef2 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota Examen sustitutorio:");
es2 =Float.parseFloat(entrada.nextLine());

//se genera el codigo del curso
codigoCurso2=eap.substring(0,1)+ceap+"0002";
//-----------------------------------------------------
//-----------------------------------------------------
//curso 3
```

```
System.out.println("Nombre del curso 3:");
nc3 =entrada.nextLine();
System.out.println("Ciclo");
ci3 =entrada.nextLine();
System.out.println("Creditos:");
c3 = Float.parseFloat(entrada.nextLine());
System.out.println("linea de formacion:");
lf3 =entrada.nextLine();
System.out.println("Nota practica 1:");
np31 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota practica 2:");
np32 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota practica 3:");
np33 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota trabajo 1:");
nt31 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota trabajo 2:");
nt32 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota de Examen de Medio curso:");
em3 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota de Examen final:");
ef3 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota Examen sustitutorio:");
es3 =Float.parseFloat(entrada.nextLine());

//se genera el codigo del curso
codigoCurso3=eap.substring(0,1)+ceap+"0003";
//-------------------------------------------------
//-------------------------------------------------
```

```
//curso 4
System.out.println("Nombre del curso 4:");
nc4 =entrada.nextLine();
System.out.println("Ciclo");
ci4 =entrada.nextLine();
System.out.println("Creditos:");
c4 = Float.parseFloat(entrada.nextLine());
System.out.println("linea de formacion:");
lf4 =entrada.nextLine();
System.out.println("Nota practica 1:");
np41 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota practica 2:");
np42 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota practica 3:");
np43 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota trabajo 1:");
nt41 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota trabajo 2:");
nt42 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota de Examen de Medio curso:");
em4 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota de Examen final:");
ef4 =Float.parseFloat(entrada.nextLine());
System.out.println("Nota Examen sustitutorio:");
es4 =Float.parseFloat(entrada.nextLine());

//se genera el codigo del curso
codigoCurso4=eap.substring(0,1)+ceap+"0004";
//-------------------------------------------------
```

```
}
catch(NumberFormatException e)
{
  System.err.println();
}
//-----------------------------------------------------
//para el cálculo del promedio del curso 1
float pp1,pt1;
pp1= (np11+np12+ np13)/3;
pt1= (nt11+nt12)/2;
  //-----------------------------------------------------
  //se sustituye el examen sustitutorio
//a la menor nota de:em o ef
  if( em1< ef1)
      {
           if( es1>em1)
                em1= es1;
      }
     else
      {
           if(es1>ef1)
                ef1= es1;
      }
  //-----------------------------------------------------
          pc1= (pp1+pt1+2*em1+2*ef1)/6;
//-----------------------------------------------------
//-----------------------------------------------------
 //para el cálculo del promedio del curso 2
```

```
float pp2,pt2;
pp2= (np21+np22+ np23)/3;
pt2= (nt21+nt22)/2;
//-----------------------------------------
//se sustituye el examen sustitutorio
//a la menor nota de:em o ef
if( em2< ef2)
  {
            if( es2>em2)
                em2= es2;
   }
   else
  {
            if(es2>ef2)
                ef2= es2;
   }
  //-----------------------------------------
  pc2= (pp2+pt2+2*em2+2*ef2)/6;
//-----------------------------------------
//-----------------------------------------
//para el cálculo del promedio del curso 3
float pp3,pt3;
pp3= (np31+np32+ np33)/3;
pt3= (nt31+nt32)/2;
//---------------------------------------
//se sustituye el examen sustitutorio
//a la menor nota de:em o ef
if( em3< ef3)
  {
```

```
            if( es3>em3)
                em3= es3;
  }
  else
  {
            if(es3>ef3)
                ef3= es3;
  }
  //---------------------------------
          pc3= (pp3+pt3+2*em3+2*ef3)/6;
  //-------------------------------------
 //-------------------------------------
  //para el cálculo del promedio del curso 4
float pp4,pt4;
pp4= (np41+np42+ np43)/3;
pt4= (nt41+nt42)/2;
  //-----------------------------------------------------
  //se sustituye el examen sustitutorio
//a la menor nota de: em o ef
if( em4< ef4)
  {
            if( es4>em4)
                em4= es4;
   }
   else
   {
            if(es4>ef4)
                ef4= es4;
   }
```

```
        //-----------------------------------------------
        pc4= (pp4+pt4+2*em4+2*ef4)/6;
      //-----------------------------------------------
      //calculo del promedio ponderado
      float pp=0;
      int ppi=0;
      String ppl="";
      pp=(c1*pc1+c2*pc2+c3*pc3+c4*pc4)/(c1+c2+c3+c4);

      //se redondea el promedio ponderado
      ppi=(int)Math.round(pp);

      //el promedio ponderado en letras
      ppl=numeroaLetra(ppi);
      //-----------------------------------------------
      //SE IMPRIME POR PANTALLA EL PROMEDIO DE CADA CURSO
      // Y EL PROMEDIO PONDERADO DEL SEMESTRE
      //-----------------------------------------------
      System.out.println("ESCUELA: "+eap);
      System.out.println("APELLIDOS Y NOMBRES: "+apno);
      System.out.println("SEMESTRE: "+se);
      System.out.println("PROMEDIO PONDERADO: "+ppi+"("+ppl+")");

System.out.println("=========================================");
      System.out.println("CURSO    DESCRIPCION            CICLO  CRED.
NOTA    LETRA");

System.out.println("=========================================");
      System.out.println(codigoCurso1+" "+nc1+" "+ci1+" "+c1+"
"+pc1+""+numeroaLetra((int)Math.round(pc1)));
```

```
        System.out.println(codigoCurso1+" "+nc1+" "+ci1+" "+c1+"
"+pc1+""+numeroaLetra((int)Math.round(pc2)));

        System.out.println(codigoCurso1+" "+nc1+" "+ci1+" "+c1+"
"+pc1+""+numeroaLetra((int)Math.round(pc3)));

        System.out.println(codigoCurso1+" "+nc1+" "+ci1+" "+c1+"
"+pc1+""+numeroaLetra((int)Math.round(pc4)));


    }
}
```

1.9 BUCLES

Los bucles son sentencias que hacen repetir una porción de código n veces, dependiendo del valor de verdad de la proposición establecida en el punto de término.

Todo bucle tiene un comienzo, un incremento y un final. Los bucles más usados en programación son: for(; ;), while(), do{ }while()

1.9.1 Bucle PARA-for(; ;)

Este bucle repite la ejecución de los códigos que se encuentran dentro del for, hasta que la proposición de salida sea falsa.

En algoritmo esto responde a:

PARA contador=valor inicial **HASTA** valor final, **HACER**

// bloque de codigo

Fin del **PARA**

Sintaxis en java

```
for(inicio ; proposición ;incremento/decremento)
{

}
```

EJEMPLO N°01-012

```
//sentencia que repite la impresión de números del 1 al 10
for(int i=1;i<=10;i++)
{
  System.out.println(i);
}
```

EJEMPLO N°01-013

Realizar un programa que calcule el factorial de un numero

DESARROLLO

El factorial de un número, se calcula multiplicando el mismo número por los números menores a dicho número, por ejemplo

5!=5x4x3x2x1=1x2x3x4x5=120

5!=5x4!

n!=nx(n-1)!

1. ANALISIS:

1.1 Entrada:

Ingresar un número: n

1.2 Salida:

Factorial del número ingresado:f

2.3 Formula:

n!=nx(n-1)!

2. ALGORITMO

Inicio

1. Ingresar número de mes:n
2. f=1
3. Para i=1 hasta n

f=fxi

4. Imprimir por pantalla el factorial del numero:f

Fin

3. CODIGO EN JAVA:

```
//-----------------------------------------------------------------------------------------------------
*PROGRAMA QUE IMPRIME EL FACTORIAL DE UN
*NUMERO INGRESADO POR TECLADO
//-----------------------------------------------------------------------------------------------------
package javaapplication1;
import java.io.*;
public class Main
{
  public static void main(String[] args)
   {
     double n=0,f;
BufferedReader entrada=new BufferedReader(new
InputStreamReader(System.in));
      try
      {
        System.out.println("Ingresar un numero");
        n=Double.parseDouble(entrada.readLine());
      }
      catch(Exception e)
      {
       System.err.println("Error");
      }

      //se calcula el factorial del numero
      f=1;
      for(int i=1;i<=n;i++)
         f=f*i;

      //se imprime por pantalla el factorial calculado
      System.out.println(n+"!="+f);
   }
}
```

EJEMPLO N°01-014

Realizar un algoritmo y programa para calcular la combinatoria de un numero ***n*** tomados de ***r*** en ***r***

DESARROLLO

Para calcular la combinatoria de un número ***n*** tomas de ***r*** en ***r*** se utiliza la siguiente fórmula:

$$C_r^n = \frac{n!}{(n-r)!r!}$$

Por lo tanto, esto requiere el cálculo del factorial de un numero, en este caso sería tres veces que calcularíamos el factorial

1. ANALISIS:

1.1 Entrada:

Ingresar un número entero: n

Ingresar de cuanto en cuanto: r

1.2 Salida:

Combinatoria de ***n*** tomas de ***r*** en ***r:*** *c*

Formula:

$$C_r^n = \frac{n!}{(n-r)!\,r!}$$

2. ALGORITMO

Inicio

1. Ingresar un numero entero: n
2. Ingresar de cuanto en cuanto se realiza la combinatoria: r
3. Calculo de los factoriales:

3.1 f1=1

3.2 **Para** i=1 **hasta** n

f1=f1xi

3.3 f2=1

3.4 **Para** i=1 **hasta** n-r

f2=f2xi

3.5 f3=1

3.6 **Para** i=1 **hasta** r

f3=f3xi

4. Se calcula la combinatoria

c=f1/(f2xf3)

5. Se imprime por pantalla la combinatoria: c

Fin

3. CODIGO EN JAVA:

¿?

EJEMPLO N°01-015

Realizar un algoritmo y programa para generar la siguiente secuencia: 0, 1, 1,2, 3, 5, 8, 13, … e imprima en pantalla los n términos

DESARROLLO

Esta secuencia de números, son los famosos números de Fibonacci, cada número empezando desde el tercer número, es la suma de los dos anteriores.

Código que genera en java seria:

```
import java.util.*;
class Main
{
public static void main(String args[])
{
    double ant1=0,ant2=1,siguiente=0;//se declaran variables
     int n;//numero de terminos
```

```
        Scanner s=new Scanner(System.in);//Flujo de entrada de datos
        //se pide a que se ingrese el numero de terminos
        System.out.println("Ingresar la cantidad de terminos");
        n=s.nextInt();
       for(int i=1;i<=n;i++)
        {
            System.out.println(siguiente);
            siguiente=ant1+ant2;
           //se hace el intercambio
           ant1=ant2;
           ant2=siguiente;
        }
    }
}
```

1.9.2 BUCLE MIENTRAS-while (*proposición*)

Cuando no tenemos exactamente cuántas veces queramos que se repita nuestro código, es un poco dificultoso realizar el bucle con **for,** para ello tenemos el while, que responde en lenguaje natural a:

MIENTRAS (***proposición***) **HACER**
INICIO
    //sentencias mientras proposición sea verdadero
FINAL

Las sentencias que se encuentran dentro del MIENTRAS se ejecutara, siempre en cuando la ***proposición*** sea verdadera, hasta que en algún momento esa proposición se convierta en falsa, por ejemplo.

Sea el algoritmo que imprima 3 números por pantalla:

INICIO

1. *n*=1
2. MIENTRAS(*n*<4) HACER
   INICIO
     2.1 IMPRIMIR POR PANTALLA: *n*
     2.2 *n*=*n*+1
   FINAL

FINAL

En el algoritmo del ejemplo podemos observar que el valor de *n* empieza en 1 y luego se evalúa la proposición: ***n*<=4**

- n=1 es menor que 4(**verdadero**) por lo tanto realiza lo que está dentro del MIENTRAS.

Osea se imprime por pantalla 1(en el punto 2.1), luego el n se incrementa en 1(en el punto 2.2)

- n=2 es menor que 4(**verdadero**) por lo tanto realiza lo que está dentro del MIENTRAS.

Osea se imprime por pantalla 2(en el punto 2.1), luego el n se incrementa en 1(en el punto 2.2)

- n=3 es menor que 4(**verdadero**) por lo tanto realiza lo que está dentro del MIENTRAS.

Ósea se imprime por pantalla 3(en el punto 2.1), luego el n se incrementa en 1(en el punto 2.2)

- n=4 es menor que 4(**falso**) por lo tanto ya no entra al MIENTRAS.

SINTAXIS EN JAVA

```
while(proposición)
{
//sentencias mientras proposición sea verdadero
}
```

El código en java del algoritmo anterior seria:

```
int n:
n=1;
while(n<4)
{
   System.out.println(n);// se imprime por pantalla el valor de n
   n=n+1;
}
```

EJEMPLO N°01-016

Realizar el algoritmo y programa para hallar el máximo común divisor de 2 números enteros ingresados por el teclado.

DESARROLLO

Para realizar este programa, existe el algoritmo de Euclides que permite calcular el máximo común divisor de dos números, lo cual consisten en realizar divisiones sucesivas hasta encontrar un residuo igual a “0”.

Por ejemplo:

- MCD(12,8)

1	1	2
12	8	**4**
	4	0

TABLA N° 01-0003: Codificación de las salidas

Por lo tanto el MCD (12,8)=4

1. ANALISIS:

1.1 Entrada:

Ingresar dos números enteros: *A, B*

1.2 Salida:

Máximo común divisor de *A* y *B: D*

1.3 Formula:

Ninguno

2. ALGORITMO

INICIO

1. Ingresar dos numero enteros*: A, B*
2. Se calcula el MCD por divisiones sucesivas

r =*A* **mod** B

MIENTRAS(*r*!=0) HACER

INICIO

A=*B*

B=*r*

r =*A* **mod** B

FINAL

a. Imprimir por pantalla la combinatoria el MCD:*B*

FIN

3. CODIGO EN JAVA:

```
//-----------------------------------------------------------------------------------------------
*PROGRAMA QUE CALCULA EL MCD DE DOS NUMEROS
//-----------------------------------------------------------------------------------------------
package javaapplication4;

import java.util.*;

import java.io.*;
```

```
public class Main
{
        public static void main(String[] args)
        {
        int A,B,r;
        Scanner entrada=new Scanner(System.in);
        A=entrada.nextInt();
        B=entrada.nextInt();
        r=A%B;
        while(r!=0)
         {
                A=B;
                B=r;
                r=A%B;
         }
         System.out.println(B);
        }
}
```

EJEMPLO N°01-017

Realizar el algoritmo y programa para simplificar una fracción, por ejemplo 18/20=9/10

DESARROLLO

Para simplificar una fracción, tenemos que encontrar el MCD del numerador y denominador y luego lo dividimos a cada uno de ellos por el MCD de dichos números y la fracción ya estará simplificada.

Por ejemplo:

Simplificar: 18/20

Lo que hacemos es calcular el MCD (18, 20) =2

Luego al 18 se divide entre 2=9(Numerador)

Al 20 se divide entre 2=10(Denominador)

Entonces 18/20=9/10

1. ANALISIS:

1.1 Entrada:

Ingresar fracción: *A/B*

1.2 Salida:

Fracción simplificada*: A1/B1*

1.3 Formula:

Ninguno

2. ALGORITMO

INICIO

1. Ingresar la fracción: f
2. Capturar el numerador: num
3. Capturar el denominador: denom
4. Se calcula el MCD de num y denom:d
5. Se divide del num entre d:c1
6. Se divide el dem entre d:c2
7. Se compone la nueva fracción simplificada:c1/c2

FINAL

3. CODIGO EN JAVA:

```
package javaapplication4;
import java.util.*;
import java.io.*;
public class Main
{
    public static void main(String[] args)
```

```
        {
        int A,B,r,A1,B1;
        String cad;
        Scanner entrada=new Scanner(System.in);
        cad=entrada.next();
        String[] cad1=cad.split("/");

A1=Integer.parseInt(trim(cad1[0].toString()));
        B1=Integer.parseInt(trim(cad1[0].toString()));
         int aux=0;

        //se intercambia valores
        //el A siempre tiene que ser mayor
          if(A1<B1)
         {
           A=B1;
           B=A1;
         }
        else
        {
          B=B1;
         A=A1;
        }
        //se intercambia
        //se calcula el MCD del numerdor y denominador
         r=A%B;
        while(r!=0)
         {
                A=B;
```

```
            B=r;
            r=A%B;
        }
        System.out.println(cad+" = "+A1/B+"/"+B1/B);
    }
}
```

1.9.3 BUCLE HACER MIENTRAS- do{ }while (*proposición*)

Este bucle es parecido al while, con la diferencia de que al menos la sentencia que se encuentra dentro del HACER MIENTRAS, se ejecuten una vez.

En lenguaje natura, para realizar los algoritmos seria así:

HACER

INICIO

//sentencias mientras proposición sea verdadero

FINAL

MIENTRAS (***proposición***)

SINTAXIS EN JAVA

```
do{
//sentencias mientras proposición sea verdadero
    } while(proposición)
```

EJEMPLO N°01-018

Realizar un programa para que imprima los n números primos, por pantalla.

DESARROLLO

Para realizar este programa, se pone un contador en un bucle do while para que cada vez que encuentre un siguiente numero primo, se incremente en 1, hasta llegar a los n números.

1. ANALISIS:

1.1 Entrada:

Cantidad de números primos a imprimir: *n*

1.2 Salida:

Lista de *n* números primos*: p1, p2, p3,…*

1.3 Formula:

Ninguno

2. ALGORITMO

INICIO

1. Ingresar la cantidad de números primos a visualizar: n
2. Se calcula la cantidad secuencia de números primos

```
Cont=0
i=1
HACER
    Div=0
    PARA J=1 HASTA i
     INICIO
        Si (i mod j ==0)
          Div=Div+1
     FINAL
    Si(div==2)

    INICIO
        Se imprime por pantalla: i
        Se incrementa la cantidad de números primos
        Cont=cont+1
```

FINAL

I=i+1

MIENTRAS(cont<=n)

FINAL

3. CODIGO EN JAVA

```
package ejemploboocles;
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
     int cont=1,i=1,n;
     int div=0;
     Scanner s=new Scanner(System.in);
     System.out.println("Ingresar la cantidad de numeros primos a mostrar:");
     n=s.nextInt();
     System.out.println("Los primos son:");
     do{
          //--------------------------------------------
          //se verifica si el i es primo
          div=0;
          for(int j=1;j<=i;j++)
          {
            if(i%j==0)
                div++;
          }
          //si la cantidad de divisores es 2, entonces el i es primo
          if(div==2)
          {
             System.out.println(i);
             cont++;// se incrementa la cantidad de
```

```
                    // primos impresos
            }
            //-----------------------------------------
           i++;//se incrementa el siguiente numero a generar

        }while(cont<=n);
    }
}
```

1.10 MATRICES-ARREGLOS

Una variable en programación, solo puede almacenar un valor a la vez, por ejemplo, se tiene una variable ***n*** de tipo entero: *n*=5; si ahora a *n* le asigno otro valor: *n*=7, el valor que antes tenía de 5 ya se pierde. Este inconveniente se puede solucionar creando varias variables de tipo entero: *n1, n2, n3,…;* y asignándoles los valores que les corresponde, pero cuando se quiere almacenar muchos datos del mismo tipo, por ejemplo nombres de 100 alumnos, ya se tendría inconvenientes. Por esa necesidad nace la utilización del concepto matemático de matrices en programación.

Matemáticamente una matriz se representa como un conjunto de variables ordenados en filas y columnas, tal como se muestra en la IMAGEN N° 01-0015

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}_{3x3}$$

IMAGEN N° 01-0015: Matriz de 3x3

La identificación de cada elemento de la matriz se realiza por la fila y columna que ocupa en la matriz dicho elemento.

Estos conceptos se aplican en programación cuando se quiere almacenar muchos datos del mismo tipo.

En programación una matriz es un arreglo de datos del mismo tipo, representados en filas y columnas, y cada elemento se identifica por el índice que viene a ser la fila y columna que ocupa en la matriz dicha elemento.

1.10.1 ARREGLO DE UNA SOLA DIMENSIÓN

La sintaxis para declarar un arreglo de una sola dimisión en java seria de la siguiente manera:

```
{Tipo dato} nombre_arreglo [ ]=new {Tipo de dato}[n]
```

Para el recorrido de todos los casilleros generalmente se utiliza el for, que es más fácil conociendo la cantidad de elementos que tenga el arreglo

La sintaxis para declarar arreglo de dos dimisiones en java seria de la siguiente manera:

```
{Tipo dato} nombre_arreglo[ ][ ]=new {Tipo de dato}[n][m]
```

Para el recorrido de todos los casilleros generalmente se utiliza fors anidados, que es más fácil conociendo la cantidad de filas y columnas que tenga el arreglo.

EJEMPLO N°01-019

Arreglo que almacena 20 números enteros:

```
//se declara un arreglo de una sola columna

int numero[]=new int[20];
```

En este arreglo que es de una sola columna tal como se muestra en la FIGURA N° 01-017, se han habilitado 20 casilleros en donde podemos almacenar datos de tipo entero:

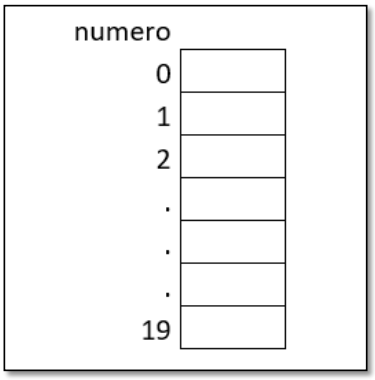


FIGURA N° 01-017: Arreglo de 20x1

Para exceder a cada casillero tenemos que hacerlo por su índice, por ejemplo:

Si se quiere almacenar el 5 en el tercer casillero, sería así: numero[2]=5, ahora se quiere almacenar en el casillero 10 el doble del valor del dato del casillero 3, entonces sería así: numero[9]= 2*numero[2]

Entonces cada casillero diferenciado por su índice representa a una variable del mismo tipo.

1.10.2 ARREGLO DE DOS DIMENSIONES

Un arreglo de datos de dos dimensiones está estructurado en filas por columnas. Y para su acceso a cada elemento se tiene que conocer la fila y columna del dato a acceder.

EJEMPLO N°01-020

Arreglo de dos dimensiones que almacena 400 números enteros:

```
//-------------------------------------------------------------------------------------------
se declara un matriz de 20 filas por  20 columnas, cuyos elementos
// seran datos de tipo entero
        int numero[][]=new int[20][20];

         for(int i=0;i<20;i++)

                for(int j=0;j<20;j++)

        numero[i][j]=i;


//-------------------------------------------------------------------------------------------
```

Estas líneas de código lo que hace es habilitar 20 filas por 20 columnas de casilleros, en los cuales se almacenan 20x20 datos, tal como se muestra en la TABLA N°01-0004.

numero	0	1	2	.	.	.	19
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
.							
.							
.							
19							

TABLA N° 01-0004: Arreglo de 20x20

Si se quiere acceder a cualquier elemento de la matriz de la TABLA N° 01-0004, se realiza por su ubicación en: fila y columna del elemento:
Numero[1][5]=1

EJEMPLO N°01-021

Realizar el análisis y programa para generar ***n*** números aleatorios, de uno o dos cifras cada uno y ordenarlos de manera ascendente.

DESARROLLO

1. ANALISIS:

Se genera con un bucle los ***n*** números aleatorios con la clase Random de java, para lo cual se importa: **java.util.Random** y **java.math.*** en los cuales se encuentran operaciones que permite generar números aleatorios y redondear respectivamente y debido a que estos números son decimales se multiplica por 100 y se redondea, así generar números enteros de uno o dos cifras, luego se almacena en un arreglo de una sola columna, para luego ser ordenado: **"números".**

Para ordenar el arreglo, se procede primero a encontrar el menor valor de la lista y lego se intercambia con el dato de la posición 0; luego se toma como la nueva lista a los datos desde la posición 1 y se busca el menor valor para intercambiar con el dato de la posición 1; así sucesivamente hasta que la sublista esté constituido por un solo elemento y cuando suceda eso los elementos del arreglo ya estarán ordenados de menor a mayor.

Por ejemplo:

Sea la lista que se muestra en la TABLA N° 01-0005

0	7
1	5
2	10
3	1
4	2
5	9
6	8

TABLA N° 01-0005: Arreglo de nx1

Aplicando el algoritmo, las listas en todo el proceso quedarían, así como se muestra en la IMAGEN N° 01-0018, en cada iteración del bucle.

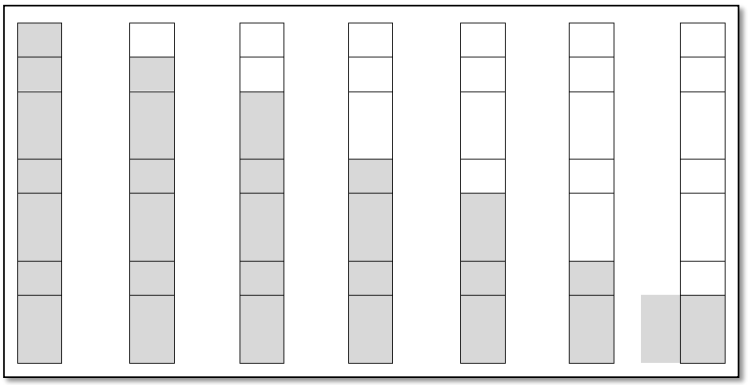


IMAGEN N° 01-0018: Proceso de ordenamiento de los elementos del arreglo

1.1 Entrada:

Cantidad de números a generar: n

1.2 Salida:

Lista de *n* números ordenados de forma ascendente

1.3 Formula:

Ninguno

2. ALGORITMO

El ordenamiento burbuja

3. CODIGO EN JAVA:

```
//-----------------------------------------------------------------------------------------------
*PROGRAMA QUE GENERA n NUMEROS ALEATORIOS Y
*LOS ORDENA DE MENOR A MAYOR */
//-----------------------------------------------------------------------------------------------
package javaapplication3;

import java.util.Random;

import java.util.*;

import java.math.*;

public class Main
```

```
{
   public static void main(String[] args)
   {
     //se declara un matriz de una sola columna
     int numero[]=new int[20];
     int n;
     Scanner s=new Scanner(System.in);
     Random r=new Random(10);//se instancia la clase Random
     System.out.println("Ingresar la cantidad de numeros a generar");
     n=s.nextInt();
     //-------------------------------------------------------------
     //se genera los numeros aleatorios
     // y se almacena en una matriz
     for(int i=0;i<n;i++)
     {
           numero[i]=(int)Math.round(r.nextDouble()*100);
           System.out.println(numero[i]);
     }
    //-------------------------------------------------------------
    //se ordena a los elementos de la matriz
     int indMenor;
     int aux;
     for(int i=0;i<n;i++)
     {
        //for que busca el menor valor en la sub lista
        indMenor=i;
        for(int j=i;j<n;j++)
        {
          if(numero[j]<numero[indMenor])
```

```
                indMenor=j;
            }
            //-----------------------------------
            //se intercambia los valores
            aux=numero[i];
            numero[i]=numero[indMenor];
            numero[indMenor]=aux;
            //-----------------------------------
        }

        //se imprime la lista ordenada
        System.out.println("La lista ordenada es:");
        for(int i=0;i<n;i++)
        {
            System.out.println(numero[i]);
        }
      //--------------------------------------------------
    }
}
```

EJEMPLO N°01-021

Implementar un sistema para controlar las compras y ventas que realiza una tienda de abarrotes, los datos de los artículos que se consideran pertinentes: código, descripción, precio unitario, stock actual, el sistema controlara las compras y ventas diarias que realiza la tienda, reportando las compras y ventas diarias discriminado por artículo, venta total, un reporte especial llamado KARDEX de acuerdo al siguiente formato:

```
Código: xxx             Descripción    : xxxxxxxx
Precio : 999.00         Stock Anterior: 999

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

```
Operación          Fecha       Cantidad          Importe
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
xxxxxxxx           99/99/99    9999999           9999.99
xxxxxxxx           99/99/99    9999999           9999.99
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
Total Compras:     9999999     Total Ventas:     999999
Stock Actual:      9999999
```

DESARROLLO

En el desarrollo del programa no se tomará en cuenta los datos del cliente ni del proveedor, solo se considerará datos de artículos y de las compras y ventas.

Para el almacenamiento de los artículos se necesita una matriz de 4 columnas por n filas, en donde la columna 0: serviría para almacenar el código, la columna 1 para almacenar la descripción del artículo, la columna 2 para almacenar el precio unitario, la columna 4 para almacenar el stock del articulo; Esto estaría representado como se muestra en la TABLA N° 01-0006.

	0	1	2	3
	Código	Descripción	Precio unitario	Stock actual
0	A001	PRODUCTO 1	20.00	100
1				
2				
3				
4				
5				
6				
.				
.				
.				

TABLA N° 01-0006: Arreglo para almacenar datos de artículos

Representación en java

```
String articulo[][] = new String[1000][4];
```

Manejo:

- Si se quiere almacenar el codigo del articulo seria asi:

```
articulo[0,0]= "A001";
```

- Si se quiere almacenar la descripcio del articulo seria asi:

  articulo[0,1]= "PRODUCTO 1";

- Si se quiere almacenar el precio unitario del articulo seria asi:

  articulo[0,2]= "20.00";

- Si se quiere almacenar el stock actual del articulo seria asi:

  articulo[0,3]= "100";

Para el almacenamiento de datos en cada registro, se tiene que llevar el control de la fila en donde se quiere almacenar, para lo cual se tiene que tener la posición del siguiente artículo a almacenar en una variable “n”,
Si se quiere modificar el valor de una columna de alguna fila, también se tienen que tener el índice de la fila. Para el recorrido de todas las filas lo más recomendable es utilizar bucles con ***for***, ya que se tienen exactamente el número de interacciones a ejecutar.
Para el almacenamiento de las compras y ventas se tendría que tener un arreglo con 6 columnas como mínimo que se representaría de la siguiente manera: **operacion.**

	0	1	2	3	4	5
	código	cantidad	precio unitario	stock actual	subtotal	tipo
0	A001	10	20	20/10/2011	200	v
1	A001	100	19	20/10/2011	1900	c
2						
3						
4						
5						
.						
.						
.						

TABLA N° 01-0007: Arreglo para almacenar datos de las operaciones

Representación en java

```
String operacion[] []  = new String[1000] [ 6];
```

Manejo:

- Si se quiere almacenar el codigo del articulo en el arreglo operacion seria asi:

```
operacion [0,0]= "A001";
```

Para el manejo del sistema se tienen que crear un menú que tendría las opciones siguientes

```
**********<<MENU>>**********

1. Gestión  de artículos
2. Compra/venta
3. Listar artículos
4. Mostrar  Kardex
0. Salir
Ingresar opción:
   ********************************
```

CODIGO JAVA

```
/*////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 * Programa que gestiona la compra y venta
 * de una tienda de artículos
/*////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package javaapplication4;
import java.util.*;
public class Main {
  public static void main(String[] args)
    {
      int op,op1;
      int numero_articulos=0;//Cantidad de articulos
      int numero_operaciones=0;//cantidad de operaciones
      Scanner s=new Scanner(System.in);
      String articulo[][] = new String[1000][4];//se declara la matriz para
                                        //almacenar datos de articulos
      String operacion[][]  = new String[1000] [ 6];
```

```
do{ //do 1
  //-------------------------------------------------------
  //SE CREA EL MENU DEL SISTEMA
  System.out.println(".....MENU.....");
  System.out.println("<1>.....Gestion de articulos");
  System.out.println("<2>.....Compra/Venta");
  System.out.println("<3>.....Listar articulos");
  System.out.println("<4>.....Mostrar Kardex");
  System.out.println("<0>.....Salir");
  System.out.print("Ingresar opcion");
  op=s.nextInt();
  //-------------------------------------------------------
  switch(op)
  {
    case 1:
       do//do 2
       {
       //-------------------------------------------------------
       //se genera el sub menu de gestion de  articulos
       System.out.println("Gestion de articulos");
       System.out.println("<1>.....Ingreso de nuevos articulos");
       System.out.println("<2>.....Modificar datos de articulos");
       System.out.println("<3>.....Eliminar datos de articulos");
       System.out.println("<4>.....Listar articulos");
       System.out.println("<0>.....RETORNAR AL MENU ANTERIOR");
       System.out.println("Gestion de articulos");
       System.out.println("Ingresar opcion");
       op1=s.nextInt();
```

```
switch(op1)
{
   case 1:
        System.out.print("Codigo:");
        articulo[numero_articulos][0]=s.next();
        System.out.print("Articulo:");
        articulo[numero_articulos][1]=s.next();
        System.out.print("Precio:");
        articulo[numero_articulos][2]=s.next();
        System.out.print("Stock actual:");
        articulo[numero_articulos][3]=s.next();
        numero_articulos++;//se incrementa el contador
                           //del numero de articulos

      break;
    case 2:
       String cod;
       //se modifica datos de articulo
       //para lo cual el usuario
       //tienen que ingresar el codigo del articulo
       //para buscar el dato que se modificara
        System.out.print("Codigo:");
        cod=s.next();

        //se realiza la busqueda secuencial
        //comparando el codigo ingresado  con cada
        //codigo existente en la matriz
        //hasta encontrar el primero que sea igual
        for(int i=0;i<numero_articulos;i++)
```

```
    {
      if(articulo[i][0].compareTo(cod)==0)
      {
       //si se encuentra
       //-----------------------------------------------------
       //se presenta por pantalla los datos del articulo
       System.out.println("Articulo:"+articulo[i][1]);
       System.out.println("Precio Unitario:"+articulo[i][2]);
       System.out.println("Stock:"+articulo[i][3]);
       //-----------------------------------------------------
       //se actualiza los datos
       System.out.print("Articulo:");
       articulo[i][1]=s.next();
       System.out.print("Precio:");
       articulo[i][2]=s.next();
       System.out.print("Stock actual:");
       articulo[i][3]=s.next();
       break;
      }
    }

  break;
 case 3:
   String cod1;
   int elim=0;
   //se elimina datos de articulo
   //para lo cual el usuario
   //tienen que ingresar el codigo del articulo
   //para buscar el dato que se eliminara
```

```
System.out.print("Codigo:");
cod1=s.next();

//se realiza la busqueda secuencial
//comparando el codigo ingresado  con cada
//codigo existente en la matriz
//hasta encontrar el primero que sea igual
for(int i=0;i<numero_articulos;i++)
{
  if(articulo[i][0].compareTo(cod1)==0)
  {
   //si se encuentra
   //-------------------------------------------------
   //se presenta por pantalla los datos del articulo
   System.out.println("Articulo:"+articulo[i][1]);
   System.out.println("Precio Unitario:"+articulo[i][2]);
   System.out.println("Stock:"+articulo[i][3]);
   //-------------------------------------------------
   System.out.print("Eliminar de todas maneras Si(1)/No(0)");
   elim=s.nextInt();
   if(elim==1)//si escogio eliminar
   {
    //desde la posicion en que  se encuentra ubicado
    // el dato se hace un desplazamiento con un
    //casillero menos, osea chancando al dato a
    //eliminar
    for(int j=i;j<numero_articulos-1;j++)
    {
     articulo[j][0]=articulo[j+1][0];
```

```
            articulo[j][1]=articulo[j+1][1];
            articulo[j][2]=articulo[j+1][2];
            articulo[j][3]=articulo[j+1][3];
          }
          numero_articulos--;//disminuimos la cantidd de articulos
         }

        break;
       }
     }


   break;

 case 4:
    //se lista todo los articulos existentes
    System.out.println("================================");
   System.out.println("Lista de articulos");
    System.out.println("================================");
          System.out.println("= Codigo   Nombre     Precio U. Stock");
    System.out.println("================================");
    for(int i=0;i<numero_articulos;i++)
    {
      System.out.print(articulo[i][0]);
      System.out.print("      ");
      System.out.print(articulo[i][1]);
      System.out.print("      ");
      System.out.print(articulo[i][2]);
      System.out.print("      ");
```

```
                System.out.println(articulo[i][3]);

            }
             System.out.println("================================");

           break;
    }
    //-------------------------------------------------------
    }while(op1!=0);//do 2

    break;
case 2:
    int compra_venta=0;
    String cod2;
    int existe=0;
    java.util.Date fecha = new Date();//se captura la fecha del CPU


    System.out.println(">>>>>>>>>>COMPRA/VENTA<<<<<<<<<<");
    System.out.println("<1>.....COMPRA");
    System.out.println("<2>.....VENTA");
    compra_venta=s.nextInt();
    if (compra_venta==1)
       operacion[numero_operaciones][5]="C";
    else
       operacion[numero_operaciones][5]="V";

    //se pide el codigo del arciculo a comprar o vender
    //y se realiza un busqueda en nuestra matriz
```

```
System.out.println("Codigo de Articulo");
cod2=s.next();
//se realiza la busqueda secuencial
//comparando el codigo ingresado  con cada
//codigo existente en la matriz
//hasta encontrar el primero que sea igual
for(int i=0;i<numero_articulos;i++)
 {
  if(articulo[i][0].compareTo(cod2)==0)
   {
    //si se encuentra
    existe=1;
    //-----------------------------------------------------
    //se presenta por pantalla los datos del articulo
    System.out.println("Articulo:"+articulo[i][1]);
    System.out.println("Precio Unitario:"+articulo[i][2]);
    System.out.println("Stock:"+articulo[i][3]);
    //-----------------------------------------------------
    //pedimos la cantidad a comprar o vender
    System.out.print("Ingresar cantidad:");
    double cant;
    cant=Double.parseDouble(s.next());
    //comparamos con el stock actual
    if(cant>Double.parseDouble(articulo[i][3]))
     System.out.print("La cantidad ingresado es mayor que el stock");
    else
    {
    //almacenamos el codigo del producto en la matriz  operacion
    operacion[numero_operaciones][0]=articulo[i][0];
```

```
                //almacenamos la cantidad
                operacion[numero_operaciones][1]=cant+"";
                //almacenamos el precio unitario
                operacion[numero_operaciones][1]=articulo[i][2];
                //almacenamos la fecha de compra o venta
                operacion[numero_operaciones][3]=fecha.toString();

                //almacenamos el sub total
                double sub_total;

sub_total=Double.parseDouble(operacion[numero_operaciones][3])*Double.par
seDouble(operacion[numero_operaciones][3]);
                operacion[numero_operaciones][4]=sub_total+"";

                //actualizamos el stock en la matriz articulo
                double stock_actualizado;
                if (compra_venta==1)

stock_actualizado=Double.parseDouble(articulo[i][2])+Double.parseDouble(ope
racion[numero_operaciones][1]);
                else
stock_actualizado=Double.parseDouble(articulo[i][2])-
Double.parseDouble(operacion[numero_operaciones][1]);

                articulo[i][3]=stock_actualizado+"";

                //incrementamos la cantidad de operaciones
                numero_operaciones++;

                }
```

```
            break;//salimos de for
            }
          }
      if(existe==0)
          System.out.println("Codigo no existe!!!!!!!.....");
      break;
  case 3:
      String cod3;
      //---------------------------------------------------------------
       System.out.println("Codigo de Articulo");
       cod3=s.next();
       //se imprime el kardex por producto
       //recorriendo toda la matriz operacion
       // y discriminando solo las operaciones
       // hechas con el articulo cuyo codigo
       //se ingreso
       System.out.println("================================");
       System.out.println("=OPERACION  FECHA   CANTIDAD  IMPORTE");
        System.out.println("================================");

        for(int i=0;i<numero_operaciones;i++)
          {
            if(operacion[i][0].compareTo(cod3)==0)
             {
             System.out.print(operacion[i][5]);
             System.out.print("      ");
             System.out.print(operacion[i][3]);
             System.out.print("      ");
             System.out.print(operacion[i][1]);
```

```
            System.out.print("      ");
            System.out.println(operacion[i][4]);
           }
          }
          System.out.println("==================================");
    //-----------------------------------------------------------
    break;

 case 4:
     //se lista todo los articulos existentes
          System.out.println("================================");
          System.out.println("Lista de articulos");
          System.out.println("================================");
          System.out.println("= Codigo   Nombre     Precio U. Stock");
          System.out.println("================================");
            for(int i=0;i<numero_articulos;i++)
            {
              System.out.print(articulo[i][0]);
              System.out.print("      ");
              System.out.print(articulo[i][1]);
              System.out.print("      ");
              System.out.print(articulo[i][2]);
              System.out.print("      ");
              System.out.println(articulo[i][3]);

            }
             System.out.println("================================");
    break;
case 0:
```

```
            System.out.println("A DIOS");
        default:
            System.out.println("Opcion no valido");
            break;
      }
    }while(op!=0);//do 1


  }
}
```

1.11 EXCEPCIONES(EXCEPTION)

Los eventos inusuales que pueden surgir durante la ejecución de un programa y que modifican su flujo normal de ejecución se conocen como excepciones. Esto sucede por lo general cuando alguna instrucción de nuestro código genera un error, como, por ejemplo: al intentar dividir por cero, cuando un objeto es 'null' y no debería serlo, o cuando un archivo no se abre correctamente, entre otros casos. Cuando se desencadena una excepción, se muestra un mensaje de error en pantalla y el programa termina su ejecución. Estos eventos se pueden gestionar de manera correcta y hacer que el programe siga su ejecución.

Tipos de Excepciones en Java

1. **Checked Exceptions** (Excepciones Verificadas)

Son excepciones que se verifican en tiempo de compilación. Esto significa que el compilador exige que estas excepciones sean manejadas de alguna manera, ya sea usando bloques **try-catch** o mediante la declaración throws en la firma del método. Ejemplos comunes de excepciones verificadas son IOException, SQLException, y FileNotFoundException.

2. **Unchecked Exceptions** (Excepciones No Verificadas)

Estas excepciones no son verificadas en tiempo de compilación. Suelen ser el resultado de errores de lógica del programa que el programador debería evitar. Ejemplos de excepciones no verificadas incluyen NullPointerException, ArrayIndexOutOfBoundsException, y ArithmeticException.

3. **Errors** (Errores):

Son problemas más graves que normalmente no se pueden manejar en tiempo de ejecución. Los errores representan problemas del entorno de ejecución, como OutOfMemoryError o StackOverflowError. En general, estos no deben ser capturados por bloques try-catch, ya que son el resultado de condiciones que el programa no puede manejar normalmente.

Manejo de excepciones en Java

Las excepciones se manejan principalmente utilizando bloques **try**, **catch** y **finally**. Un bloque **try** encapsula el código que puede lanzar una excepción, mientras que uno o varios bloques **catch** manejan las excepciones que puedan ocurrir.

Es posible utilizar varios bloques catch para gestionar distintos tipos de excepciones de forma específica.

El bloque **finally** se emplea para ejecutar código que debe ejecutarse siempre, sin importar si se generó una excepción o no. Es comúnmente utilizado para liberar recursos que necesitan ser cerrados, como conexiones a bases de datos o archivos.

Sintaxis:

```
try {
    int numerador, denominador=1;
    System.out.print("Ingresar numerador:");
    numerador=entrada.nextInt();

    System.out.print("Ingresar denominador:");
    numerador=entrada.nextInt();

    //codigo que podria generar una excepcion
    //cuando el denominador sea 0
    //de división por cero (ArithmeticException)
    int resultado = numerador / denominador;

    System.out.print(resultado);

    }catch (ArithmeticException e) {
    // Captura de la excepción de división por cero
```

```
        System.out.println("Excepción: " + e.getMessage());

    } catch (Exception e) {
        // Captura de cualquiera otra excepción
        System.out.println("Excepción: " + e.getMessage());
    } finally{

        // bloque opcional que siempre se ejecuta,
        // con o sin  exepcion
          System.out.println("El bloque se ejecutara si o si");

        }
```

Algunos ejemplos con clases de excepciones

- ✓ **Excepción Checked Exceptions**

En el siguiente código, intenta abrir un archivo llamado “persona.txt” para lectura utilizando un FileReader. Si el archivo no se encuentra, se captura la excepción FileNotFoundException y se imprime un mensaje indicando que el archivo no se encontró.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CheckedExample {
    public static void main(String[] args) {
```

```
        try {
            File file = new File("persona.txt");
            FileReader fileReader = new FileReader(file);
            BufferedReader br = new BufferedReader(fileReader);
            System.out.println(br.readLine());
        } catch (FileNotFoundException e) {
            System.out.println("El archivo no se encontró.");
        } catch (IOException ex) {

    Logger.getLogger(CheckedExample.class.getName()).log(Level.SEVERE,
    null, ex);
        }
    }
}
```

- ✓ **Excepcion Unchecked Exceptions**

En el siguiente código, intenta imprimir el valor ubicado en la posición 5 de un array. Sin embargo, el array solo contiene los índices 0, 1, 2 y 3, por lo que tratar de acceder al índice 5 provocará una excepción ArrayIndexOutOfBoundsException, ya que está fuera de los límites permitidos para ese array.

```
public class FJ_EJEMPLO001 {

    public static void main(String[] args)
    {
        String[] arr = {"Verano", "Invierno", "Otoño","Primavera"};
        System.out.println(arr[5]);
    }
```

```
}
```

La excepción ArrayIndexOutOfBoundsException es una excepción no verificada en Java. Esto significa que no es necesario gestionarla explícitamente con un bloque **try-catch**. La responsabilidad recae en el programador de asegurarse de que el acceso a los elementos del array esté dentro de sus límites válidos.

Excepciones personalizadas

Las excepciones se pueden personalizar y hacerlo más entendible para el usuario, atreves de una clase que hereda de la clase Exception. Se implementa un constructor que acepta un mensaje como argumento y lo pasa al constructor de la clase base Exception usando super(mensaje), a continuación se muestra el codigo:

```
public class MiExcepcionPersonalizada extends Exception {

    public MiExcepcionPersonalizada(String mensaje) {

        super(mensaje);

    }
}
```

Luego en el método **main**, se instancia la clase MiExcepcionPersonalizada utilizando throw. Se captura esta excepción usando un bloque **try-catch** y se imprime el mensaje asociado a la excepción. Como se muestra en el siguiente código:

```
public class Ejemplo {
    public static void main(String[] args) {
        try {
            throw new MiExcepcionPersonalizada("Este es un mensaje de
excepción personalizada.");
        } catch (MiExcepcionPersonalizada e) {
```

```
            System.out.println("excepción personalizada: " + e.getMessage());
        }
    }
}
```

1.12 LECTURA Y ESCRITURA DE FICHEROS

En Java, la lectura y escritura de ficheros es una tarea común que se puede realizar utilizando las clases de la biblioteca estándar, como File, FileReader, FileWriter, BufferedReader, BufferedWriter, FileInputStream, FileOutputStream, Scanner, entre otras.

Lectura de un fichero de texto

Para la lectura de fichero se utiliza el FileReader y BufferedReader, tal como se muestra el siguiente código:

```
File archivo = new File ("D:/archivo.txt");
FileReader fr = new FileReader (archivo);
BufferedReader br = new BufferedReader(fr);
...
String linea = br.readLine();
```

La apertura de un archivo y su lectura posterior pueden generar excepciones que es necesario manejar. Por esta razón, tanto la apertura como la lectura del archivo deben realizarse dentro de un bloque **try-catch**.

```
try (BufferedReader br = new BufferedReader(...)) {
  ...
} catch (Exception e) {
  ...
```

```
}
// El BufferedReader se cierra automáticamente, sin hacerlo
explícitamente.
```

Asimismo, es fundamental cerrar el archivo una vez hayamos terminado con él, independientemente de si la operación se realizó con éxito o si ocurrió un error durante la lectura después de abrirlo. Para gestionar esto, es común utilizar la estructura **try-with-resources**. En este enfoque, el archivo se abre dentro de los paréntesis del bloque try y se asigna a una variable. Si dicha variable implementa la interfaz Closeable, el archivo se cerrará automáticamente al finalizar el bloque try, sin importar si se produce o no una excepción.

Código completo:

```
import java.io.*;


class LeeFichero {
  public static void main(String [] arg) {


    try (FileReader fr = new FileReader("C:/archivo.txt")) {
      BufferedReader br = new BufferedReader(fr);
      // Lectura del fichero
      String linea;
      while((linea=br.readLine())!=null)
        System.out.println(linea);
    }
    catch(Exception e){
      e.printStackTrace();
    }
  }
}
```

Lectura de fichero utilizando Scanner

En las siguientes líneas de código se utiliza el **import java.util.Scanner**

```
import java.io.File;
import java.util.Scanner;
public class LecturaFicheros {
public static void main(String[] args) {

        // Fichero del que queremos leer
        File fichero = new File("D:/dato.txt");
        Scanner s = null;

        try {
            // Leemos el contenido del fichero
            System.out.println("Se lee contenido del fichero ...");
            s = new Scanner(fichero);

            // Leemos linea a linea el fichero
            while (s.hasNextLine()) {

                // Guardamos la linea en un String
                    String linea = s.nextLine();

                System.out.println(linea);// Imprimimos la linea
            }

        } catch (Exception ex) {
            System.out.println("Mensaje: " + ex.getMessage());
```

```
        } finally {
                // Cerramos el fichero
                try {
                        if (s != null)
                                s.close();
                } catch (Exception ex2) {

                        System.out.println("Mensaje 2: " + ex2.getMessage());

                        }
        }
    }
}
```

En el código, podemos observar que el objeto 's' de la clase "Scanner" avanza línea por línea en el archivo, y dentro del bucle "while" se solicita la siguiente línea si está disponible.

Además, el código incluye la definición de varias excepciones para asegurar que, en caso de que ocurra alguna, el programa no detenga su ejecución. Por ejemplo, al final del código, en la sección "finally", se asegura que, haya ocurrido o no un error durante la lectura del archivo, este se cierre correctamente.

Escritura de un fichero de texto

Para la escritura de ficheros se utiliza el **import java.io.FileWriter**

Así mismo se ve dos formas de escribir ficheros: para escribir fichero sin codificación determinada y para escribir con una determinada codificación (con UTF-8)

En el ejemplo se escribirá el siguiente arreglo en un fichero “alumnos.txt”

```
String[] alumnos={"Elmer","Jose","Maria","Juana","Socrates"};
```

El arreglo se recorre con un for y se escribe en el fichero línea por línea

PRIMERA FORMA:

```
import java.io.FileWriter;

public class EscrituraFicheros {

  public static void main(String[] args) {

        String[] alumnos={"Elmer","Jose","Maria","Juana","Socrates"};

        FileWriter fichero = null;
        try {

              fichero = new FileWriter("D:/alumnos.txt");

              // se escribe linea a linea en el fichero
              for (String fila : alumnos) {
                    fichero.write(fila + "\n");
              }
              fichero.close();

        } catch (Exception ex) {
        System.out.println("Msg exc: " + ex.getMessage());
        }
  }
}
```

Para seguir agregando al contenido del fichero más filas y no borrar el contenido existente, al segundo argumento de FileWriter se le pasa “true”:

```
FileWriter("D:/alumnos.txt", true);
```

En el código se utiliza el “**FileWriter**" en el que se le pasa el nombre del fichero. Si el fichero no existe se crea.

SEGUNDA FORMA:

```
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.io.Writer;

public class EscrituraFicheros {
public static void main(String[] args) throws FileNotFoundException {
String[] alumnos={"Elmer","Jose","Maria","Juana","Socrates"};
/** ESCRITURA Con el fichero codificado en UTF-8 **/
Writer out = null;
try {
   out = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream("D:/alumnos.txt"), "UTF-8"));

// Escribimos linea a linea en el fichero
   for (String fila : alumnos) {
        try {
             out.write(fila+"\n");
```

```
            } catch (IOException ex) {
            System.out.println("Msj excepcion escritura: " + ex.getMessage());
            }
    }

} catch (UnsupportedEncodingException | FileNotFoundException ex2) {
        System.out.println("Mensaje error 2: " + ex2.getMessage());
} finally {
    try {
            out.close();
    } catch (IOException ex3) {
            System.out.println("Msg erro cierre fichero: " + ex3.getMessage());
    }
}
}
}
```

REQUERIMIENTOS PROPUESTOS PARA DESARROLLAR ALGORITMO Y PROGRAMACION EN JAVA

1. Programa para determinar el menor y mayor valor de 8 números.
2. Realizar el algoritmo y programa para determinar el punto de intersección de dos rectas en el plano cartesiano.
3. Realizar un algoritmo y programa para que valide la existencia de un triángulo a partir de sus tres lados.
4. Realizar un algoritmo y programa para que calcule el área de un triángulo conociendo sus tres vértices en el plano cartesiano.
5. Realizar un programa para saber si tres puntos en el plano cartesiano pertenecen a una misma:
   a. recta
   b. circunferencia
   c. parábola
   d. hipérbola
   e. elipse
6. Se adquiere más de 100 unidades de un mismo artículo, nos hacen un descuento de un 40%, entre 25 y 100 unidades un 20%; entre 10 y 24 un 10% y no hay descuento para la adquisición de 10 unidades. Realizar un programa para calcular el importe a pagar.
7. De acuerdo a un principio aritmetico si un numero es multiplo de 5 termina en 0 ó 5. Aplicando este principio, determinar si un numero es multiplo de 5.
8. Realizar un programa para construir la ecuacion de una circunferencia a partir de 3 puntos.
9. La tribuna occidente de un estadio esta enumerada del 1 al 500 en la parte superior horizontal que corresponde al numero de asientos por columna; y del 1 al 50 en la parte izquierda vertical que corresponde al numero de asientos por fila. Los boletos tienen una numeracion del 1 al 25000. Para cualquier numero de boleto determinar el numero de fila y columna que le corresponderia a la ubicación de un asiento.
10. Hacer una calculadora en letras de tal manera que te realiza las 4 operaciones básicas (+,-,*,/).

El Ingreso de los números y los operadores se debe realizar en letras, Por ejemplo.

INGRESE EL PRIMER NUMERO…..>>**DOS**

INGRESE EL SEGUNDO NUMERO..>>**DOCE**

INGRESE EL OPERADOR……..>>**SUMAR**

LA SUMA ES……>>CATORCE

11. Realizar un programa para que factorice un número entero, mayor que 99 ingresado por el teclado. Por ejemplo.

    INGRESE EL NUMERO …..>>100

    EL NUMERO EN SUS FACTORES PRIMOS ES>>2^2x5^2

12. Un Técnico de luminotecnia debe encargarse de encender los focos de iluminación del escenario de un teatro. Para ello dispone de un microcontrolador con 4 entradas I1, I2, I3, I4, y cuatro salidas R, V, A, M que encienden los correspondientes focos de colores respectivos Rojo, Verde, Azul, y Marrón. Las especificaciones de funcionamiento del microcontrolador son las siguientes:

- ✓ Cuando esté el actor principal solo en el escenario debe activar únicamente dos interruptores cualesquiera que no sean consecutivos, de forma que se enciendan todas las luces mientras este actor se encuentre solo en el escenario.
- ✓ Cuando suban los restantes actores debe haber tres interruptores cualesquiera activados para que se encienda las luces Rojo, Verde, Azul.
- ✓ Si todos los interruptores están apagados o si se activan sólo dos interruptores consecutivos, todos focos están apagados.
- ✓ Cuando el escenario se quede sin actores, entre actos o antes de empezar la obra, debe encenderse sólo la luz Roja, para lo cual es necesario tener sólo un interruptor activado, el que sea.
- ✓ Cuando finalice la obra y salga todos los actores a saludar al público, debe encender todos los focos excepto el rojo. Para ello deberán estar todos los interruptores activados.

  Hacer el programa del microcontrolador que hacen prender y apagar a las luces del escenario.

13. Se considera un ascensor dotado de un **dispositivo de seguridad**, para que no puedan viajar niños menores solos ni pesos excesivos. Queremos que el ascensor se ponga en marcha cuando esté vacío o con pesos entre 25 y 300 kilos. Dotamos al ascensor de tres sensores: A sensible a cualquier peso, B sensible a pesos mayores de 25 kilos y C sensible a pesos superiores a 300 kilos. Desarrollar el programa del dispositivo de seguridad.
14. En una reunión celebrada entre 12 países de la Comunidad Europea se acuerda aceptar las resoluciones aprobadas por la mayoría de los miembros. España, Italia, Portugal y Grecia votan en bloque. Situación similar es la de Francia y Alemania. También hacen lo mismo Reino Unido e Irlanda por un lado y Bélgica, Holanda y Luxemburgo por otro. Dinamarca siempre vota lo contrario que Alemania y los tres países Bélgica, Holanda y Luxemburgo lo contrario que Irlanda.

    Desarrollar un programa para encontrar a los países que tienen mayor poder de decisión.
15. Un embalse de una presa, que se está llenando de agua, como se muestra en la IMAGEN N° 01-0018, dispone de 4 sensores de nivel designados como n_0, n_1, n_2, n_3.

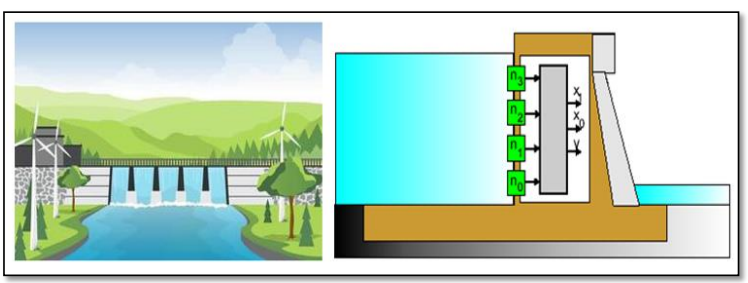


IMAGEN N° 01-0018: Sistema de embalse de una presa

La salida de cada sensor vale uno o cero según el sensor que esté cubierto por el agua o no. Por tanto, una vez que un sensor esté cubierto por el agua, todos los que están más abajo también lo estarán. Las salidas de los sensores están conectadas a un circuito integrado programable, que codifica el nivel del agua

mediante un número de 2 bits formados por variables lógicas x_1 y x_0. Existe una salida más, denominada V, que solamente vale 1 cuando la presa está vacía (ningún sensor cubierto), en cuyo caso las salidas x_0 y x_1 no importan. La codificación que realiza este bloque se muestra en la siguiente tabla:

Nivel del agua	Salida x1	Salida x2	Salida V
Ningún sensor cubierto	No importa	No importa	1
Agua hasta n_0	0	0	0
Agua hasta n_1	0	1	0
Agua hasta n_2	1	0	0
Agua hasta n_3	1	1	0

TABLA N° 01-0008: Codificación de las salidas

Realizar el programa que simule el circuito integrado, para que controle todo el sistema.

16. El grafico de la IMAGEN N° 01-0019 representa un sistema de obtención de agua caliente por energía solar, consta de un depósito, un panel solar para calentar agua y una caldera. Además, dispone de unas conducciones por donde circula el agua cuando funciona la bomba (B), hacia los paneles si está abierta la válvula Vp, o hacia la caldera si está abierta la válvula Vc. El sistema dispone de 3 sensores que a partir de las temperaturas Tp, Td y Tf producen las siguientes señales digitales:

$$Tdf = 1 \Leftrightarrow \quad Td \langle Tf \qquad Tpf = 1 \Leftrightarrow \quad Tp \langle Tf \qquad Tdp = 1 \Leftrightarrow \quad Td \langle Tp$$

Realizar un programa para el tratamiento de información que controle el funcionamiento de esta instalación, de forma que:

a) Siempre que la temperatura del agua en el depósito Td sea menor que la de los paneles Tp se pondrá en marcha la bomba y se abrirá la válvula Vp.
b) Si Tp y Td son menores que una temperatura prefijada Tf (por ejemplo. Tf = 35°
c) se debe poner en marcha la bomba y abrir Vc para que el agua sea calentada por la caldera.

d) En cualquier otro caso, la bomba debe estar parada y las válvulas cerradas.

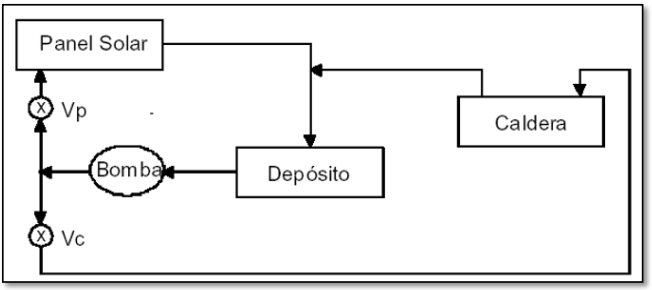


IMAGEN N° 01-0019: Sistema de bombeo automático

17. Dos sistemas digitales A y B utilizan códigos no convencionales para representar naturales entre el 0 y el 15. El sistema A representa el natural n como 4n ≅ p (mod 16) y p es mínimo y se representa en binario. El sistema B representa n como 7n ≅ q(mod 16) y también q es mínimo y se representa en binario.

Ejemplos: Sist A: n = 3, 12 ≅ 12(mod 16) entonces p =12 = 1100(binario)

Sist B: n = 6, 42 ≅10 (mod 16) entonces q=10 = 1010(binario)

Para conectar ambos sistemas es necesario desarrollar un **programa** que convierta un natural [0...15] del código A al código B.

Nota. - La representación A ≅ B (mod n), es una representación y notación de la aritmética modular.

18. Se desea desarrollar un sistema de detección de incendios para un edificio de 2 plantas. Cada planta dispone de 1 pulsador de alarma (P1 para la planta 1 y P2 para la segunda). Además, desde el centro de control puede activarse una señal de inhibición para cada planta (I1 e I2) una vez que se tiene conocimiento de la alarma. Como respuesta a la pulsación, el sistema generará 3 salidas: dos para indicar en qué piso se activó un pulsador (A_1 y A_2) y una tercera de alarma general (A). El mecanismo detallado de funcionamiento del sistema será el siguiente:

- ✓ Si se activa el pulsador de un piso se activará la señal de alarma de dicho piso.
- ✓ Si la señal de inhibición de un piso determinado está a '1' no se debe activar la alarma de ese piso.
- ✓ Si se produce alarma en cualquier piso debe activarse la señal A de alarma general.

  Desarrollar el software del sistema

19. El sistema nervioso humano, incluyendo el cerebro, está hecho de células especializadas llamadas neuronas. Cada neurona tiene sinapsis (puntos de interconexión, como se muestra en la figura adjunta) de excitación y sinapsis de inhibición. Una neurona produce una salida 1 si el número de sinapsis de excitación con pulsos 1 excede el número de sinapsis de inhibición con pulsos 1 por al menos el valor de umbral de la neurona.

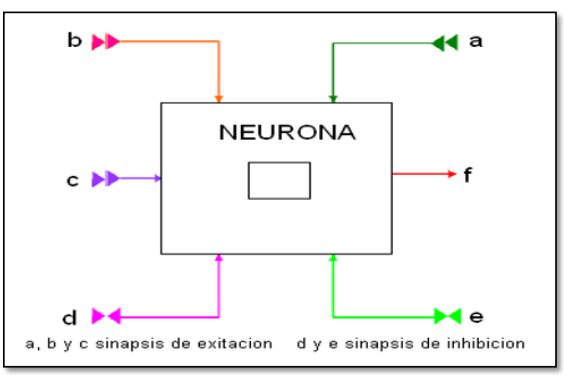


IMAGEN N° 01-0020: Neurona

Desarrollar un programa, de emisión de pulsos a través del canal de salida (axón) en el modelo de la figura, bajo las siguientes condiciones:
(C1) Valor del umbral = 1 (es decir, se produce una salida 1 si el número de sinapsis de excitación con pulsos 1, excede por al menos uno el número de sinapsis de inhibición con pulsos 1), y (C2) Siempre que haya al menos un

pulso 1 en alguna sinapsis del puerto de excitación, habrá al menos un pulso 1 en alguna sinapsis del puerto de inhibición (es decir, no es posible -en este modelo restringido- que existan pulsos 1 en el puerto de excitación si no existe al menos un pulso 1 en el puerto de inhibición).

20. Realizar un programa que permita evaluar una función ingresado por el teclado

    Por ejemplo.

    INGRESE LA FUNCION>>sen(x)+x
    INGRESE UN VALOR PARA "x">>0
        EL VALOR DE:f(0)=0
    DESEA SEGUIR EVALUANDO LA FUNCION SI(1)/NO(0)>>1
    INGRESE UN VALOR PARA "x">>3.1416
        EL VALOR DE:f(3.1416)=3.1416
    DESEA SEGUIR EVALUANDO LA FUNCION SI(1)/NO(0)>>0
        DESEA INGRESAR OTRA FUNCION SI(1)/NO(0)>>0
    "HASTA PRONTO"

21. Realizar un programa para controlar la planilla de pago de personal de una empresa considerando los siguientes datos por empleado: Código, Área, Sueldo Básico y Horas Extras. El reporte de boleta de pago para cada empleado debe tener la siguiente forma:

Empleado: xxxxxxxxxxxxx

Área: xxxx

Ingresos:	*Deducciones*		*Neto*
Sueldo Básico:	*9999.99*	*Ipss:*	*999.99*
Horas Extras:	*9999.99*	*Snp:*	*999.99*
		Fonavi:	*999.99*
	9999.99 +	*999.99*	*9999.99*

El monto de las horas extras se obtiene multiplicando el número de las horas extras trabajadas por el sueldo básico dividido por 240. El Ipss, Snp, y Fonavi representan el 3% del sueldo básico cada uno. El ingreso de datos se termina cuando al ingresar el código se presiona ENTER.

22. En el proceso de "**Bienes y Suministros**" de la empresa de turismo "**Pata Amarilla**" que pertenece al área de Administración, se realizan las actividades de manera manual, por lo que se requiere sistematizar todo ello con la ayuda de un software, el dueño del proceso de "**Bienes y Suministros**" detalla el procedimiento y actividades de cada área:

- ✓ El proceso involucra cinco (5) áreas: Administración, Logística, Almacén, Patrimonio, Áreas usuarias.
  - ✓ El área de Logística funciona de la siguiente forma:
- ➢ Recibe las solicitudes de requerimientos de bienes o suministros de las diferentes áreas de la empresa.
- ➢ Cada solicitud tiene un responsable que está adscrito a un área.
- ➢ Cada solicitud es autorizada por el jefe del área y posteriormente por el área de Administración.
- ➢ De la solicitud se requiere la siguiente información: Número de la solicitud (consecutivo), fecha, responsable, área, meta presupuestal. En cada solicitud se pueden contener uno o muchos ítems con la siguiente información: código, nombre del ítem, cantidad solicitada, unidad de medida, valor unitario.
- ➢ Cada ítem es identificado por un código único y es de carácter devolutivo (suministro) (útiles de escritorio, útiles de limpieza, etc.) o un bien (computadora, escritorio, etc.).
- ➢ La solicitud es enviada al área de Logística para atender si es que hay en stock o realizar la compra a uno o varios proveedores. Para realizar la compra se juntan todas las solicitudes para tener la cantidad total de ítems de cada rubro a comprar.
- ➢ Las cotizaciones son realizadas con uno o varios proveedores de los bienes solicitados.

- Para comprar bienes primero se realiza la cotización dos o tres proveedores para determinar la mejor oferta en calidad y precio que un proveedor ofrece, una vez elegido el proveedor se genera la orden de compra, con los siguientes datos: número de la **orden de compra**, nombre del proveedor al cual se le va a realizar la compra, fecha de la orden, monto total de la orden, fecha de entrega. Cada orden puede tener asociado uno o varios ítems, cada ítem debe tener los siguientes datos: código del ítem, nombre del ítem, cantidad de compra, unidad de medida del ítem, valor unitario y sub total.
- La orden de compra es aprobada por el área de Administración luego el área de Logística envía o hace entrega al proveedor elegido.
  - ✓ El área de Almacén funciona de la siguiente forma:
- Recepciona los bienes que llegan de los proveedores y distribuye con una pecosa a las áreas que realizaron las solicitudes.
- El proveedor hace entrega a Almacén los ítems detallado en la Orden de Compra, para ello adjunta la guía de remisión y factura para su pago correspondiente si todo está conforme, se registra una entrada de almacén por cada factura relacionada, con los siguientes datos: número de entrada, fecha, número de factura, Proveedor, total bienes, valor total, los ítems recibidos con la cantidad respectiva.
- El área de Almacén hace entrega de los bienes a las diferentes áreas solicitantes atreves de una pecosa detallando cada ítem entregado y otros datos más como el número de salida, empleado solicitante del ítem o los ítems, cantidad, fecha de salida, fecha de entrega.
  - ✓ El área Patrimonio es responsable de:
- Administrar y controlar la ubicación de los bienes dentro de la empresa, por esto antes de que el bien salga del Almacén es codificado y se genera un sticker que va pegado al bien.
- Cada trabajador tiene a su cargo una o varios bienes, esto permite controlar su ubicación.

Realizar el análisis, algoritmo y programa en java que permita sistematizar todo lo detallado en el proceso de “**Bienes y Suministros**” de la empresa de turismo “**Pata Amarilla**”. Los datos de deberán almacenar en archivos planos.

CAPÍTULO 2. PROGRAMACIÓN ORIENTADA A OBJETOS

2.1 CLASES Y POO

2.1.1 Programación Orientado a Objetos

En el contexto de la implementación de un sistema en Java, la filosofía central es que todo son clases y objetos. Esta perspectiva define la POO como una forma de abstraer la realidad (los clientes, los pagos, los reportes) dentro de una estructura fundamental: la **clase**. La clase actúa como un plano o molde donde se encapsulan los datos (atributos) y se define el comportamiento (métodos) para interactuar con ellos. Cuando esta estructura de clase adquiere valores específicos en tiempo de ejecución (por ejemplo, el nombre, la dirección y la deuda actual de un cliente concreto), es entonces cuando se crea el **objeto**.

2.1.2 Clases

Una clase en programación es una estructura o armazón de un conjunto de objetos ya sea del mundo real o abstracto, que se simplifica a dos cosas: **atributos** o características y los **mensajes** o métodos. Por ejemplo, si se quiere tener un directorio de los libros que se tienen en casa y estoy armando una tabla en Excel, las columnas que consideraría serían: código, nombre libro, año de publicación, edición, número de páginas, área, tomo, etc., tal como se muestra en la IMAGEN N°02-0001; podríamos darle nombre a esa estructura: **Libro**. Así mismo, si quiero tener un registro de los datos de mis contactos y diseño la estructura en Excel, las columnas serían: dni, apellidos, nombres, correo, teléfono, dirección, etc., tal como se muestra en la IMAGEN N°02-0002; podríamos darle nombre a esta estructura: **Contacto**.

Lo que faltaría a estas dos estructuras es la definición de los **métodos** de cada una de ellas para poder manipular los datos y realizar otras operaciones (ej. Libro.prestar(), Contacto.enviarNotificacion()), y así tendríamos la representación completa como **clases**. Es fundamental entender que la clase es el **molde estático** que define cómo debe lucir la información. Solo cuando

asignamos valores concretos a todos esos atributos (ej. "Código: L-001", "Nombre: Cien años de soledad") es que se crea la **instancia** real en la memoria, lo que denominamos **objeto**.

Las clases podrían tener un método que tiene el mismo nombre de la clase, a dicho método se le llama el **constructor**, en donde podría inicializarse o dar valores iniciales a los atributos.

LIBROS						
CODIGO	NOMBRE	AÑO	EDICION	NUMERO PAGINAS	AREA	TOMO

IMAGEN N°02-0001: Estructura o tabla “libro” para llenar datos de libros

CONTACTO					
DNI	APELLIDOS	NOMBRES	CORREO	TELEFONO	DIRECCION

IMAGEN N°02-0002: Estructura o tabla “contaco” para llenar datos de contactos

2.1.3 Representación gráfica de una clase: notación UML

La clase se representa gráficamente mediante un rectángulo dividido en tres compartimientos horizontales, lo que facilita la comprensión visual de su composición tal como se puede apreciar en la IMAGEN N°02-0003:

Primer Compartimiento (Superior): Se coloca el Nombre de la Clase (el identificador único de la plantilla, por ejemplo: Libro, UsuarioServicio).

Segundo Compartimiento (Medio): Se detallan todos los atributos o características de la clase, incluyendo su visibilidad (público, privado) y su tipo de dato (String, int, Decimal, etc.).

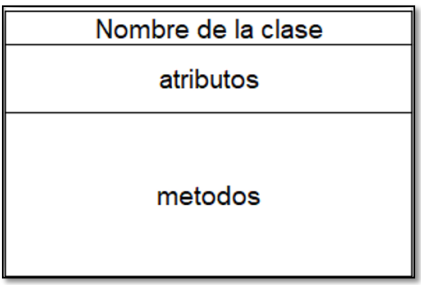


IMAGEN N°02-0003: Representación de una clase en UML

Tercer Compartimiento (Inferior): Se detallan todos los métodos de la clase, especificando su visibilidad, el nombre del método y el tipo de dato que devuelve (retorna).

Como se puede ver la clase Libro en la IMAGEN N°02-0004, esta notación establece claramente la diferencia entre los datos y el comportamiento de la entidad.

IMAGEN N°02-0004: la clase Libro

2.1.4 Diagrama de clases

El **Diagrama de Clases** es un componente fundamental en el desarrollo de sistemas de información. Su función principal es proporcionar un **esquema estático** de las clases que componen el sistema y sus **interrelaciones** con otras clases.
No se limita a detallar los atributos y métodos de una sola clase (como se hizo en la sección anterior), sino que representa la **arquitectura lógica** completa del *software*.

Componentes de un diagrama de clases

- ✓ **Esquema de las Clases:** Muestra cada clase del sistema (ej., UsuarioServicio, Boleta, Deuda) con sus respectivos atributos y métodos.
- ✓ **Interrelaciones:** El objetivo central es visualizar cómo las clases están conectadas, ya que las operaciones en la vida real (como registrar un pago) involucran múltiples entidades. Estas conexiones se representan mediante **relaciones**:
- **Asociación:** Una conexión general entre clases (ej., un Cajero usa el sistema ModuloPagos).
- **Agregación/Composición:** Representa relaciones de "contiene a" o "es parte de" (ej., una Boleta está compuesta por DetalleBoleta).
- **Herencia:** Muestra cómo una clase hereda propiedades de otra (ej., PiletaPublica hereda de UsuarioServicio).

La IMAGEN N°02-0005 presenta un Diagrama de Clases que modela las interacciones de un sistema de préstamo de libros, destacando dos estructuras relacionales esenciales. Por un lado, la operación principal del préstamo se gestiona mediante una Asociación binaria donde la Clase Prestamo actúa como el nexo que relaciona las clases principales Libro y Persona, registrando el evento con sus atributos transaccionales específicos.

Por otro lado, el diagrama también ilustra una Jerarquía Geográfica clara para el modelado de la ubicación de la persona: una Persona reside en un Distrito, el cual está contenido en una Provincia, y a su vez, esta pertenece a una Región.

Cada una de las seis clases (Libro, Persona, Prestamo, Distrito, Provincia, Region) posee sus propios atributos y métodos, conformando un esquema relacional que facilita tanto el registro transaccional del préstamo como la localización precisa del usuario.

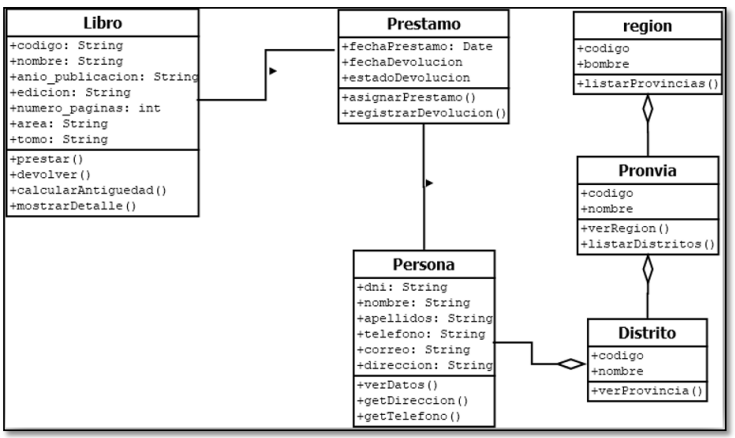


IMAGEN N°02-0005: Diagrama de clases de préstamo de libros

2.2 CLASES EN JAVA

Para definir una clase en java se utiliza la siguiente sintaxis:

```
class claseNombre
{
  [Visibilidad]tipoDato atributo1;
  [Visibilidad]tipoDato atributo1;
  [Visibilidad]tipoDato atributo1;
  ...
  [Visibilidad] [tipoDatoRetorno] operacion1()
  {
  }
  [Visibilidad] [tipoDatoRetorno] operacion2()
```

```
    {
    }
    ...
}
```

EJEMPLO N°02-001

De la IMAGEN N°02-0004, representar el código en java

DESARROLLO

En NetBenas para agregar una clase, clic derecho, New y seleccionar Java Class, tal como se muestra en la IMAGEN N°02-0006

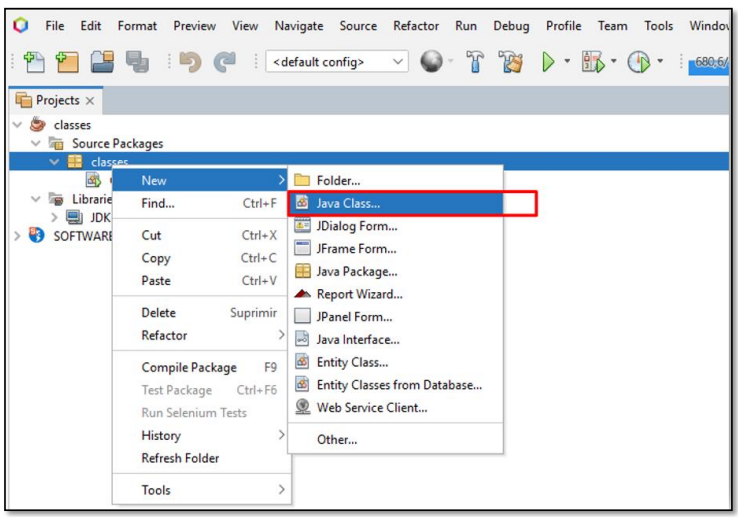


IMAGEN N°02-0006: Creando clases en NetBeans

Luego sale la ventana que se muestra en la IMAGEN N°02-0007, en donde se pone el nombre de la clase “Libro”.

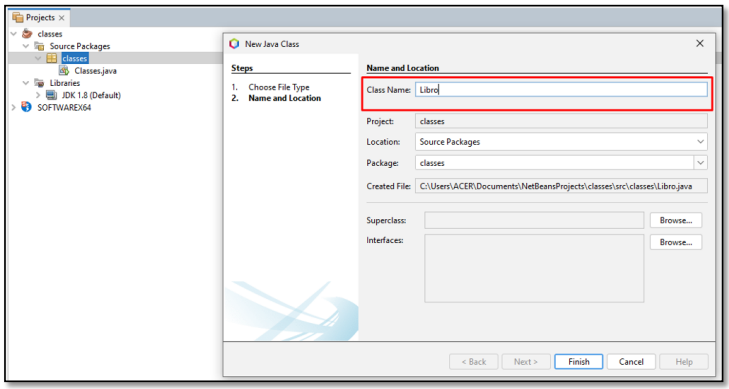


IMAGEN N°02-0007: Creando la clase Libro en NetBeans

El código en java de la clase libro quedaría así:

```
public class Libro {
  String codigo;
  String nombre;
  String anio;
  String edicion;
  int numeroPaginas;
  String area;
  String tomo;
  public void prestar(){
   }
  public void devolver(){
   }
  public int calcularAntiguedad(){
  int antiguedad=0;
  return antiguedad;
  }
}
```

EJEMPLO N°02-002

En la IMAGEN N°02-0008 se tiene la representación de la clase Ordenador, representar el código en java, también instanciar dicha clase y asignar valores al objeto creado.

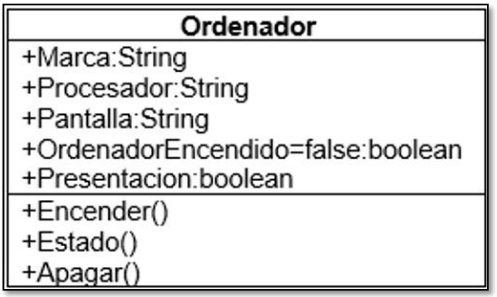


IMAGEN N°02-0008: Clase Ordenador

DESARROLLO

La clase Ordenador tiene 5 atributos, 3 métodos, el código en java seria así:

CODIGO EN JAVA

```
//-------------------------------------------------------------------------------------------------
 * Programa que implementa la clase Ordenador
 //-------------------------------------------------------------------------------------------------
package cordenador;

import java.io.*;//librería para la entrada y salida de datos

//se define la clase ordenador

class Ordenador

{

  //se declaran los atributos

  public String Marca;

  public String Procesador;
```

```
 public String Pantalla;
 public boolean OrdenadorEncendido=false;
 public boolean Presentacion;
  //-----------------------------------------------------------------------------------------------
//se declaran y definen las operaciones
//-----------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------
//método que me permite encender al ordenador, ósea se cambia de estado de apagado a
//encendido
//-----------------------------------------------------------------------------------------------
 public void Encender()
 {
     if(OrdenadorEncendido)
     System.out.println("Ordenador ya esta encendido!");
   else
   {
     OrdenadorEncendido=true;
     System.out.println("Ordenador encendido");
   }
 }
//-----------------------------------------------------------------------------------------------
//método que me permite apagar al ordenador, ósea se cambia de estado de
//encendido ha apagado.
//-----------------------------------------------------------------------------------------------
 public void Apagar()
 {
     if(OrdenadorEncendido)
   {
     OrdenadorEncendido=false;
     System.out.println("Ordenador se apago exitosamente!");
```

```
    }
    else
        System.out.println("Ordenador ya esta Apagado");
  }
//-----------------------------------------------------------------------------------------------
//método que me permite ver el estado del ordenador
//-----------------------------------------------------------------------------------------------
  public void Estado()
  {
     System.out.println("El ordenador de marca: "+Marca);
     System.out.println("con procesador: "+Procesador);
     System.out.println("de pantalla: "+Pantalla);
     if(OrdenadorEncendido)
       System.out.println("esta: ENCENDIDO ");
     else
       System.out.println("esta: APAGADO ");
    }
//-----------------------------------------------------------------------------------------------
}//fin de la classe Ordenador
```

Se crea la clase Main en donde se podrá instanciar la clase Ordenador:

```
public class Main
{
   public static void main(String[] args)
   {
      //se  instancia la clase Ordenador
      Ordenador o=new Ordenador();
      Scanner e=new Scanner(System.in);
       System.out.println("Ingresar la marca:");
       o.Marca=e.nextLine();
```

```
            System.out.println("Ingresar procesador:");
            o.Procesador=e.nextLine();

            System.out.println("Ingresar pantalla:");
            o.Pantalla=e.nextLine();

           o.Encender();
           o.Apagar();
           o.Estado();
     }
}
```

EJEMPLO N°02-003

Desarrollar un programa para realizar distintas operaciones con números complejos.

Un numero complejo tiene la siguiente forma: $Z = a + bi$, algunas de las operaciones que se puede realizar:

- Modulo del número complejo
- Argumentos del número complejo
- Conjugada del número complejo
- Cuatro operaciones de dos números complejos
- Etc.

DESARROLLO

Lo descrito sobre un número complejo y algunas operaciones, el modelado utilizando clases con UML, cuyos atributos seria: parteReal, parteImaginario, así mismo las operaciones vendrían a ser los métodos implementado en la clase, tal como se muestra en la IMAGEN N°02-0009:

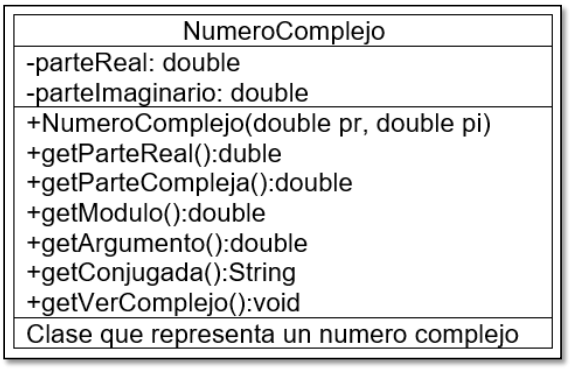


IMAGEN N°02-0009: Clase NumeroComplejo

Así mismo se crearía otra clase que represente las operaciones entre dos números complejos, tal como se muestra en la IMAGEN N°02-0010.

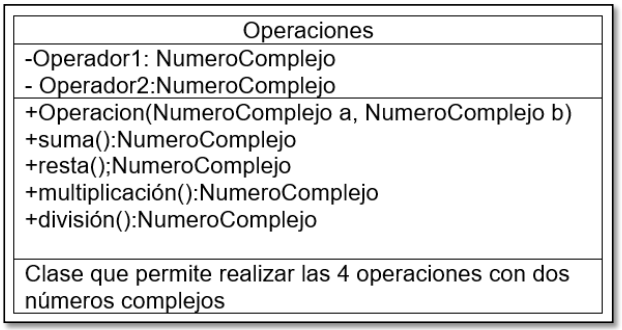


IMAGEN N°02-0010: Clase Operaciones

CODIGO EN JAVA

En NeatBeans, clic derecho sobre el paquete, del menú desplegable seleccionar New tal como se muestra en la IMAGEN N°02-0011.

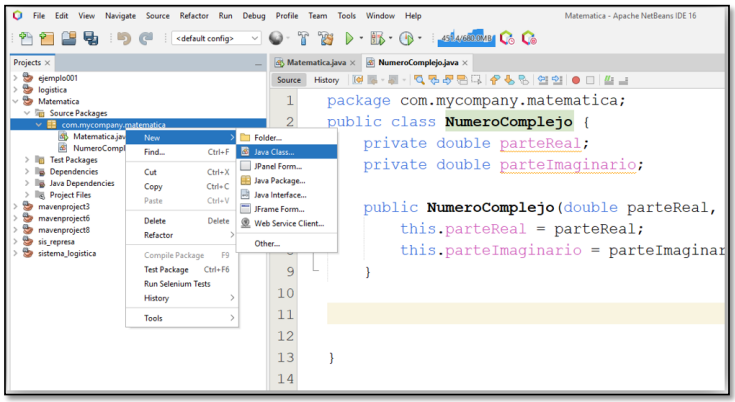


IMAGEN N°02-0011: Adición de una nueva clase al proyecto

Se da un nombre de la clase luego se agrega los dos atributos y se crea la estructura que se muestra en la IMAGEN N°02-0012.

```
public class NumeroComplejo1 {
    double parteReal;
    double parteImaginaria;
}
```

IMAGEN N°02-0012: Estructura de la clase con dos atributos

Para crear algunos métodos conocidos de manera automática con la tecla **Alt+Insert** se visualiza un menú desplegable (IMAGEN N°02-0013) en lo que se selecciona lo que se requiere:

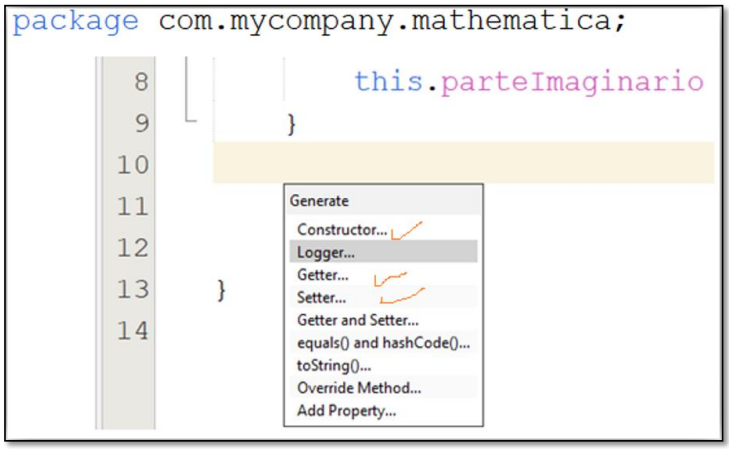


IMAGEN N°02-0013: Menú desplegable para crear algunos métodos de la clase

Clase con algunos métodos en java (IMAGEN N°02-0014):

```
package com.mycompany.matematica;
public class NumeroComplejo {
    private double parteReal;
    private double parteImaginario;
    public NumeroComplejo(double parteReal, double parteImaginario) {
        this.parteReal = parteReal;
        this.parteImaginario = parteImaginario;
    }
    public double getParteReal() {
        return parteReal;
    }
    public double getParteImaginario() {
        return parteImaginario;
    }
    public void setParteReal(double parteReal) {
        this.parteReal = parteReal;
    }

    public void setParteImaginario(double parteImaginario) {
        this.parteImaginario = parteImaginario;
    }


}
```

IMAGEN N°02-0014: Estructura de la clase NumeroComplejo con sus atributos y métodos

```
//-----------------------------------------------------------------------------------------------
// Código completo de la clase NumeroComplejo
package com.mycompany.matematica;
public class NumeroComplejo {
   private double parteReal;
   private double parteImaginario;
   public NumeroComplejo(double parteReal, double parteImaginario) {
      this.parteReal = parteReal;
      this.parteImaginario = parteImaginario;
   }
   public double getParteReal() {
      return parteReal;
```

```
}
public double getParteImaginario() {
   return parteImaginario;
}
public void setParteReal(double parteReal) {
   this.parteReal = parteReal;
}

public void setParteImaginario(double parteImaginario) {
   this.parteImaginario = parteImaginario;
}

public String verComplejo() {
   String nC;
   if(parteImaginario<0)
      {
         nC=parteReal+"-"+parteImaginario+"i";
      }
   else
         nC=parteReal+"+"+parteImaginario+"i";

   return nC;
}

public double modulo(){
  return Math.sqrt(parteReal*parteReal+parteImaginario*parteImaginario);

   }
```

```
    public double argumento(){
      return Math.atan2(parteImaginario, parteReal);
    }

    public String conjugada(){
    String nC;
       if(parteImaginario<0)
          {
             nC=parteReal+"+"+parteImaginario+"i";
          }
       else
             nC=parteReal+"-"+parteImaginario+"i";
        return nC;
    }
}
```

```
//-----------------------------------------------------------------------------------------------
```

// Código con los métodos suma y resta de la clase Operaciones

```
package com.mycompany.mathematica;
public  class Operaciones {
    numeroComplejo o1;
    numeroComplejo o2;

    //constructor de la clase
    public Operaciones(numeroComplejo o1, numeroComplejo o2) {
       this.o1 = o1;
       this.o2 = o2;
    }
```

```
    //metodo sumara que no necesita instanciar la clase
    public static numeroComplejo suma(numeroComplejo nc1,numeroComplejo
nc2){
    numeroComplejo n=new numeroComplejo(1,1);
    n.setParteReal(nc1.getParteReal()+nc2.getParteReal());
    n.setParteImaginario(nc1.getParteImaginario()+nc2.getParteImaginario());
    return n;
    }

    //metodo suma que necesariamente se tiene que instanciar la clase
    public  numeroComplejo suma(){
    numeroComplejo n=new numeroComplejo(1,1);
    n.setParteReal(o1.getParteReal()+o2.getParteReal());
    n.setParteImaginario(o1.getParteImaginario()+o2.getParteImaginario());
    return n;
    }

    //metodo resta que necesariamente se tiene que instanciar la clase
    public  numeroComplejo resta(){
    numeroComplejo n=new numeroComplejo(1,1);
    n.setParteReal(o1.getParteReal()-o2.getParteReal());
    n.setParteImaginario(o1.getParteImaginario()-o2.getParteImaginario());
    return n;
    }

}

//-----------------------------------------------------------------------------------------------
```

```
//Instanciando las clases creadas desde una clase principal Matemática:
package com.mycompany.matematica;
public class Matematica {
    public static void main(String[] args) {
     NumeroComplejo a=new NumeroComplejo(2,4);
     NumeroComplejo b=new NumeroComplejo(-2,-3);

     System.out.println(a.verComplejo());
     System.out.println(b.verComplejo());
  }
}
```

2.3 COLECCIONES EN JAVA

En Java, las colecciones son un marco de trabajo (framework) que proporciona una arquitectura para almacenar y manipular un grupo de objetos. Existen varios tipos de colecciones que permiten diferentes estructuras de datos, como listas, conjuntos, mapas, colas, etc. Estas colecciones se encuentran en el paquete java.util y forman parte de la Java Collections Framework (JCF).

Principales interfaces de la Java Collections Framework

- ✓ Collection: La interfaz raíz que representa un grupo de objetos.
- ✓ List: Permite almacenar elementos ordenados y duplicados.
- ✓ Set: No permite duplicados y no garantiza un orden específico.
- ✓ Queue: Representa una estructura de datos de tipo cola (FIFO).
- ✓ Map: Almacena pares clave-valor, donde cada clave es única.

Clases comunes de colecciones en Java

En Java, las colecciones son estructuras que permiten almacenar y manipular grupos de objetos de manera eficiente. Las interfaces principales son **List**, **Set** y **Map**, y algunas de las clases más utilizadas que las implementan son:

- **ArrayList**: implementa List.

- **LinkedList**: implementa List y Queue.
- **HashSet**: implementa Set.
- **TreeSet**: implementa Set.
- **HashMap**: implementa Map.
- **TreeMap**: implementa Map.

Métodos comunes de las colecciones

Las colecciones en Java proporcionan varios métodos para manipular sus elementos. Los más frecuentes son:

1. **add(E e)**: Agrega un elemento a la colección.
2. **remove(Object o)**: Elimina un elemento de la colección.
3. **contains(Object o)**: Verifica si un elemento existe en la colección.
4. **size()**: Devuelve el número de elementos de la colección.
5. **clear()**: Elimina todos los elementos de la colección.
6. **isEmpty()**: Verifica si la colección está vacía.
7. **get(int index)**: Obtiene el elemento en una posición específica (solo para List).
8. **set(int index, E element)**: Modifica el elemento en una posición específica (solo para List).
9. **addAll(Collection<? extends E> c)**: Agrega todos los elementos de otra colección.

Métodos para recorrer colecciones

Existen varias formas de recorrer los elementos de una colección:

- ✓ **For-each** Sintaxis:

```
for (String item : lista) {
   System.out.println(item);
}
```

- ✓ **Bucle while:**

```
Iterator<String> iterador = lista.iterator();
while (iterador.hasNext()) {
        String item = iterador.next();
        System.out.println(item);
    }
```

- ✓ **For clásico** (solo para List):

```
for (int i = 0; i < lista.size(); i++) {
   System.out.println(lista.get(i));
}
```

Métodos para ordenar colecciones

Las colecciones que implementan List se pueden ordenar utilizando la clase **Collections**:

- ✓ **sort(List<T> list)**: Ordena la lista en orden ascendente.
- ✓ **sort(List<T> list, Comparator<? super T> c)**: Ordena la lista según un comparador personalizado.

EJEMPLO N°02-004

A partir de la clase **Libro** presentada en la Imagen N.° 02-0004, se solicita desarrollar un programa en Java que permita gestionar de manera integral una colección de libros pertenecientes a una biblioteca. El sistema deberá implementar un menú interactivo en consola y utilizar una colección dinámica, como **ArrayList**, para almacenar objetos de tipo **Libro**. Todas las operaciones deberán ejecutarse sobre los objetos existentes en la colección, aplicando los principios fundamentales de la **Programación Orientada a Objetos**.

El programa deberá implementar, como mínimo, las siguientes funcionalidades:

1. **Ingresar nuevos libros:**

Permitir registrar libros ingresando todos sus datos (código, nombre, año, edición, número de páginas, área y tomo). Cada registro debe almacenarse como un objeto Libro dentro de la colección.

2. **Modificar los datos de un libro existente:**
   A partir del código del libro, el sistema deberá permitir al usuario localizar un registro específico y actualizar cualquiera de sus atributos.
3. **Eliminar libros:**
   Facilitar la búsqueda de un libro mediante su código y permitir su eliminación segura de la colección.
4. **Listar libros:**
   Mostrar en pantalla todos los libros registrados, utilizando el método verDatos() de la clase Libro para presentar su información de manera clara y ordenada.
5. **Validación y mensajes de confirmación:**
   El sistema debe informar cuando un libro ha sido registrado, modificado o eliminado correctamente, así como advertir cuando el código ingresado no coincide con ningún libro existente.

Estos requerimientos permiten practicar el uso de clases, objetos, colecciones dinámicas, métodos de búsqueda y modificación, promoviendo el dominio de conceptos fundamentales de la Programación Orientada a Objetos.
El programa deberá desarrollarse en **tres versiones**:

A) Versión que funciona únicamente en consola.
B) Versión que almacena los libros en un archivo .txt o .csv.
C) Versión completa que combine consola, almacenamiento en archivo y una clase adicional Biblioteca para implementar una arquitectura modular y limpia

DESARROLLO

A) Versión que funciona únicamente en consola.

La primera versión del programa permite gestionar una colección de libros aplicando los principios de la Programación Orientada a Objetos (POO) y utilizando colecciones dinámicas de Java. Para ello, se emplea la clase Libro, la cual modela las características esenciales de un libro dentro de una biblioteca mediante atributos como código, nombre, año, edición, número de páginas, área y tomo. Esta clase incorpora constructores, métodos de acceso (getters y setters)

y un método verDatos() que devuelve la información completa del libro en un formato legible. Gracias a esta estructura orientada a objetos, cada libro se gestiona como una entidad independiente, lo que facilita su almacenamiento, modificación y recuperación dentro del sistema.

En el método principal se utiliza una colección ArrayList<Libro> para almacenar de manera dinámica todos los libros registrados por el usuario. A través de un menú interactivo en consola, el programa ofrece las operaciones básicas de gestión: registrar nuevos libros, modificar los datos de libros existentes, eliminar registros y listar todos los libros almacenados. Cada opción del menú opera directamente sobre la colección utilizando estructuras de recorrido como for-each, lo que permite realizar búsquedas, actualizaciones y eliminaciones de forma sencilla y eficiente. Esta primera versión constituye un ejemplo práctico del uso combinado de POO y colecciones dinámicas en Java, mostrando cómo diseñar aplicaciones básicas de administración de objetos mediante interacción por consola.

CODIGO EN JAVA

Libro.java

```
//-----------------------------------------------------------------------------------------------
 * Programa que implementa la clase Libro
 //-----------------------------------------------------------------------------------------------
package classes;


import java.time.Year;
/**Clase que representa un libro dentro del sistema.
public class Libro {
    // Atributos básicos del libro
    private String codigo;          // Código único del libro
    private String nombre;          // Título del libro
    private String anio;            // Año de publicación
    private String edicion;         // descripción de la edición
    private int numeroPaginas;      // Total de páginas
```

```
private String area;          // Área o categoría del libro
private String tomo;          // Tomo al que pertenece

// Nuevo estado del libro
private boolean prestado;     // true = prestado, false = disponible

/**Constructor con parámetros para inicializar un libro completo*/
public Libro(String codigo, String nombre,
             String anio, String edicion,
             int numeroPaginas, String area,
             String tomo) {
    this.codigo = codigo;
    this.nombre = nombre;
    this.anio = anio;
    this.edicion = edicion;
    this.numeroPaginas = numeroPaginas;
    this.area = area;
    this.tomo = tomo;
    this.prestado = false;   // Por defecto disponible
}

/**Constructor vacío para crear objetos sin valores iniciales. */
public Libro() {
    this.prestado = false;
}

/**
 * Devuelve los datos del libro en un solo texto formateado.
 */
```

```
    public String verDatos() {
        return codigo + "|" + nombre + "|" +
                anio + "|" + edicion + "|" +
                numeroPaginas + "|" + area + "|" +
                tomo + "|" + (prestado ? "Prestado" : "Disponible");
    }

    /** Convierte el objeto Libro en texto legible automáticamente. */
    @Override
    public String toString() {
        return "Libro{" +
                "codigo='" + codigo + '\'' +
                ", nombre='" + nombre + '\'' +
                ", anio='" + anio + '\'' +
                ", edicion='" + edicion + '\'' +
                ", paginas=" + numeroPaginas +
                ", area='" + area + '\'' +
                ", tomo='" + tomo + '\'' +
                ", estado='" + (prestado ?
                        "Prestado" : "Disponible") + '\'' +
                '}';
    }
    // ========================
    //     MÉTODOS FUNCIONALES
    // ========================
    /** Marca el libro como prestado si está disponible.*/
public void prestar() {
        if (!prestado) {
            prestado = true;
        } else {
```

```
        System.out.println("El libro ya está prestado.");
    }
}

/**Marca el libro como devuelto. */
public void devolver() {
    if (prestado) {
        prestado = false;
    } else {
        System.out.println("El libro ya está disponible.");
    }
}

/**Calcula la antigüedad del libro según el año actual.
 * @return años transcurridos desde su publicación  */
public int calcularAntiguedad() {
    try {
        int anioPublicacion = Integer.parseInt(anio);
        int anioActual = Year.now().getValue();
        return anioActual - anioPublicacion;
    } catch (NumberFormatException e) {
        System.out.println("Error: el año '" + anio + "' no es válido.");
        return -1; // error
    }
}
// ========================
//       GETTERS
// ========================
public String getCodigo() { return codigo; }
public String getNombre() { return nombre; }
```

```
    public String getAnio() { return anio; }
    public String getEdicion() { return edicion; }
    public int getNumeroPaginas() { return numeroPaginas; }
    public String getArea() { return area; }
    public String getTomo() { return tomo; }
    public boolean isPrestado() { return prestado; }

    // ========================
    //       SETTERS
    // ========================
    public void setCodigo(String codigo) { this.codigo = codigo; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    public void setAnio(String anio) { this.anio = anio; }
    public void setEdicion(String edicion) { this.edicion = edicion; }
    public void setNumeroPaginas(int numeroPaginas)
{ this.numeroPaginas = numeroPaginas; }
    public void setArea(String area) { this.area = area; }
    public void setTomo(String tomo) { this.tomo = tomo; }
}
```

Biblioteca.java

```
//-----------------------------------------------------------------------------------------
 * Programa que implementa la clase Biblioteca
//-----------------------------------------------------------------------------------------

package classes;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Optional;

/**Clase que encapsula la lógica de gestión de libros.
```

```
 * Responsable de mantener la lista y realizar operaciones CRUD. */
public class Biblioteca {
    private final List<Libro> libros;
    /** Inicializa la biblioteca con una lista vacía.  */
    public Biblioteca() {
        this.libros = new ArrayList<>();
    }

    /** Agrega un libro si su código no existe aún.
     * @param libro libro a agregar
     * @return true si se agregó, false si ya existe un libro con el mismo
código   */
    public boolean agregar(Libro libro) {
        if (libro == null || libro.getCodigo() == null) return false;
        if (buscarPorCodigo(libro.getCodigo()).isPresent()) {
            return false; // ya existe
        }
        return libros.add(libro);
    }

    /** Busca un libro por su código (case-sensitive).
     * @param codigo código a buscar
     * @return Optional con el libro si existe  */
    public Optional<Libro> buscarPorCodigo(String codigo) {
        return libros.stream()
                .filter(l -> l.getCodigo() != null && l.getCodigo().equals(codigo))
                .findFirst();
    }
```

```
    /** Modifica los datos de un libro identificado por código.
     * Si el libro no existe retorna false.
     * @param codigo código del libro a modificar
     * @param nuevosDatos objeto Libro con los nuevos valores (pueden
ser parciales)
     * @return true si se modificó, false si no se encontró   */
    public boolean modificar(String codigo, Libro nuevosDatos) {
        Optional<Libro> opt = buscarPorCodigo(codigo);
        if (opt.isEmpty()) return false;

        Libro l = opt.get();
        // Aplicamos cambios solo si el valor nuevo no es nulo / válido
        if (nuevosDatos.getNombre() != null)
l.setNombre(nuevosDatos.getNombre());
        if (nuevosDatos.getAnio() != null) l.setAnio(nuevosDatos.getAnio());
        if (nuevosDatos.getEdicion() != null)
l.setEdicion(nuevosDatos.getEdicion());
        if (nuevosDatos.getArea() != null) l.setArea(nuevosDatos.getArea());
        if (nuevosDatos.getTomo() != null)
l.setTomo(nuevosDatos.getTomo());
        if (nuevosDatos.getNumeroPaginas() > 0)
l.setNumeroPaginas(nuevosDatos.getNumeroPaginas());
        // no permitimos cambiar código desde aquí (para mantener
unicidad)
        return true;
    }

    /** Elimina un libro por código de forma segura usando Iterator.
     * @param codigo código del libro a eliminar
     * @return true si se eliminó, false si no se encontró  */
    public boolean eliminar(String codigo) {
```

```
        Iterator<Libro> it = libros.iterator();
        while (it.hasNext()) {
            Libro l = it.next();
            if (l.getCodigo() != null && l.getCodigo().equals(codigo)) {
                it.remove(); // seguro contra ConcurrentModificationException
                return true;
            }
        }
        return false;
    }

    /** Lista todos los libros (copia para evitar exposiciones directas).
     * @return lista inmutable de libros (copia)  */
    public List<Libro> listar() {
        return new ArrayList<>(libros);
    }

    /** Comprueba si la biblioteca está vacía. */
    public boolean estaVacia() {
        return libros.isEmpty();
    }

    /** Número de libros registrados. */
    public int contar() {
        return libros.size();
    }
}
```

AppConsole.java

```
//-----------------------------------------------------------------------------------------
 * Programa que implementa la clase AppConsole
 //----------------------------------------------------------------------------------------

package classes;
import java.util.List;
import java.util.Scanner;

/** Aplicación de consola para gestionar la Biblioteca.
 * Maneja Scanner y entradas inválidas.*/
public class AppConsole {
    private final Biblioteca biblioteca;
    private final Scanner sc;

    public AppConsole() {
        biblioteca = new Biblioteca();
        sc = new Scanner(System.in);
    }

    public static void main(String[] args) {
        AppConsole app = new AppConsole();
        app.run();
    }

    /** Bucle principal del programa con el menú. */
    public void run() {
        int opcion;
        do {
            mostrarMenu();
            opcion = leerEntero("Seleccione una opción: ");
```

```
            switch (opcion) {
                case 1 -> opcionAgregar();
                case 2 -> opcionModificar();
                case 3 -> opcionEliminar();
                case 4 -> opcionListar();
                case 0 -> System.out.println("Salir");
                default -> System.out.println("Opción no válida.");
            }
        } while (opcion != 0);
    }
    private void mostrarMenu() {
        System.out.println("\n--- GESTIÓN DE LIBROS ---");
        System.out.println("1. Nuevo Libro");
        System.out.println("2. Modificar Libro");
        System.out.println("3. Eliminar Libro");
        System.out.println("4. Listar Libros");
        System.out.println("0. Salir");
    }


    // ============================
    // Opciones del menú
    // ============================
    private void opcionAgregar() {
        System.out.println("\n-- NUEVO LIBRO --");
        String codigo = leerCadenaNoVacia("Código (único): ");
        if (biblioteca.buscarPorCodigo(codigo).isPresent()) {
            System.out.println("Ya existe un libro con ese código. Operación
cancelada.");
            return;
        }
```

```
        String nombre = leerCadenaNoVacia("Nombre: ");
        String anio = leerCadena("Año (ej. 2020): ");
        String edicion = leerCadena("Edición: ");
        int paginas = leerEnteroPositivo("Número de páginas: ");
        String area = leerCadena("Área: ");
        String tomo = leerCadena("Tomo: ");

        Libro l = new Libro(codigo, nombre, anio, edicion, paginas, area,
tomo);
        biblioteca.agregar(l);
        System.out.println("Libro agregado correctamente.");
    }

    private void opcionModificar() {
        System.out.println("\n-- MODIFICAR LIBRO --");
        String codigo = leerCadenaNoVacia("Código del libro a modificar: ");
        if (biblioteca.buscarPorCodigo(codigo).isEmpty()) {
            System.out.println("No se encontró un libro con ese código.");
            return;
        }
        System.out.println("Ingrese los nuevos valores. Enter ->sin cambio.");
        // Leemos valores opcionales: si presiona Enter se mantiene el
original
        String nombre = leerCadenaOpcional("Nombre: ");
        String anio = leerCadenaOpcional("Año: ");
        String edicion = leerCadenaOpcional("Edición: ");
        Integer paginas = leerEnteroOpcional("Número de páginas: ");
        String area = leerCadenaOpcional("Área: ");
        String tomo = leerCadenaOpcional("Tomo: ");
```

```
        // Construimos objeto con los cambios (campos nulos o 0 = sin
cambio)
        Libro cambios = new Libro();
        if (!isNullOrEmpty(nombre)) cambios.setNombre(nombre);
        if (!isNullOrEmpty(anio)) cambios.setAnio(anio);
        if (!isNullOrEmpty(edicion)) cambios.setEdicion(edicion);
        if (paginas != null && paginas > 0)
cambios.setNumeroPaginas(paginas);
        if (!isNullOrEmpty(area)) cambios.setArea(area);
        if (!isNullOrEmpty(tomo)) cambios.setTomo(tomo);
        boolean modificado = biblioteca.modificar(codigo, cambios);
        System.out.println(modificado ? "Modificado " : "No se pudo
modificar");
    }
    private void opcionEliminar() {
        System.out.println("\n-- ELIMINAR LIBRO --");
        String codigo = leerCadenaNoVacia("Código del libro a eliminar: ");
        boolean eliminado = biblioteca.eliminar(codigo);
        System.out.println(eliminado ? "Eliminado." : "No se encontró el
libro.");
    }

    private void opcionListar() {
        System.out.println("\n-- LISTA DE LIBROS --");
        if (biblioteca.estaVacia()) {
            System.out.println("No hay libros registrados.");
            return;
        }
        List<Libro> todos = biblioteca.listar();
```

```
        for (Libro l : todos) {
            System.out.println(l.verDatos());
        }
        System.out.println("Total libros: " + biblioteca.contar());
    }
    // ============================
    // Métodos de lectura / utilidades
    // ============================
    private String leerCadena(String mensaje) {
        System.out.print(mensaje);
        return sc.nextLine().trim();
    }

    private String leerCadenaNoVacia(String mensaje) {
        String s;
        do {
            System.out.print(mensaje);
            s = sc.nextLine().trim();
            if (s.isEmpty()) System.out.println("El valor no puede estar vacío. Intente de nuevo.");
        } while (s.isEmpty());
        return s;
    }
    /** Lee una línea y si está vacía devuelve null (útil para lecturas opcionales). */
    private String leerCadenaOpcional(String mensaje) {
        System.out.print(mensaje);
        String s = sc.nextLine().trim();
        return s.isEmpty() ? null : s;
    }
    private int leerEntero(String mensaje) {
```

```
        while (true) {
            System.out.print(mensaje);
            String linea = sc.nextLine().trim();
            try {
                return Integer.parseInt(linea);
            } catch (NumberFormatException e) {
                System.out.println("Entrada inválida. Ingrese un número
entero.");
            }
        }
    }
    private int leerEnteroPositivo(String mensaje) {
        int val;
        do {
            val = leerEntero(mensaje);
            if (val <= 0) System.out.println("Debe ser un número positivo.");
        } while (val <= 0);
        return val;
    }
    /** Lee un entero opcional: si el usuario presiona Enter devuelve null. */
    private Integer leerEnteroOpcional(String mensaje) {
        System.out.print(mensaje);
        String linea = sc.nextLine().trim();
        if (linea.isEmpty()) return null;
        try {
            return Integer.parseInt(linea);
        } catch (NumberFormatException e) {
            System.out.println("Valor inválido; se ignorará este campo.");
            return null;
```

```
        }
    }
    private boolean isNullOrEmpty(String s) {
        return s == null || s.trim().isEmpty();
    }
}
```

B) Versión que almacena los libros en un archivo .txt o .csv

En esta versión del programa amplía la funcionalidad de la primera versión incorporando **persistencia de datos mediante un archivo externo** (.csv). Esto permite que la colección de libros se mantenga incluso después de cerrar el programa, evitando la pérdida de información. Al iniciar, el sistema carga automáticamente todos los libros almacenados en el archivo, y cada operación de registro, modificación o eliminación actualiza inmediatamente el archivo, garantizando que los datos siempre reflejen el estado actual de la colección.

Al igual que la versión anterior, se utiliza la clase Libro para representar cada libro como un objeto independiente, con atributos como código, nombre, año, edición, número de páginas, área y tomo. La interacción con el usuario se realiza a través de un **menú en consola**, que permite registrar nuevos libros, modificar datos existentes, eliminar registros y listar todos los libros. La diferencia principal es que ahora los cambios se reflejan en un archivo .csv, utilizando métodos específicos para **guardar y cargar** los libros, combinando así los conceptos de Programación Orientada a Objetos, colecciones dinámicas y manejo de archivos en Java. Esta versión enseña cómo implementar aplicaciones más robustas que mantienen persistencia de datos de manera eficiente y profesional.

En esta versión no se requiere modificar la clase Libro, solo realizaremos cambios en la clase que contienen al **main()**

CODIGO EN JAVA

Classes.java

```
//-----------------------------------------------------------------------------------------
 * Programa que implementa la clase Classes
//-----------------------------------------------------------------------------------------
 package classes;

 import java.io.*;

 import java.util.ArrayList;

 import java.util.Scanner;


 public class Classes {

    private static final String ARCHIVO = "libros.csv";
```

```
public static void main(String[] args) {

    ArrayList<Libro> libros = new ArrayList<>();
    Scanner leer = new Scanner(System.in);
    int opcion;

    // Cargar libros desde archivo al iniciar
    cargarLibros(libros);

    do {
        System.out.println("\n--- GESTIÓN DE LIBROS V0.2");
        System.out.println("1. Registrar nuevo libro");
        System.out.println("2. Modificar libro");
        System.out.println("3. Eliminar libro");
        System.out.println("4. Listar libros");
        System.out.println("0. Salir");
        System.out.print("Ingrese opción: ");
        opcion = leer.nextInt();
        leer.nextLine();

        switch (opcion) {
            case 1:
                Libro l = new Libro();
                System.out.println("REGISTRO DE LIBRO");
                System.out.print("Código: ");
                l.setCodigo(leer.nextLine());

                System.out.print("Nombre: ");
                l.setNombre(leer.nextLine());
```

```
            System.out.print("Año: ");
            l.setAnio(leer.nextLine());

            System.out.print("Edición: ");
            l.setEdicion(leer.nextLine());

            System.out.print("Páginas: ");
            l.setNumeroPaginas(Integer.parseInt(leer.nextLine()));

            System.out.print("Área: ");
            l.setArea(leer.nextLine());

            System.out.print("Tomo: ");
            l.setTomo(leer.nextLine());

            libros.add(l);
            guardarLibros(libros);
            System.out.println("✓ Libro registrado y guardado.");
            break;

        case 2:
            System.out.print("Ingrese código del libro a modificar: ");
            String codMod = leer.nextLine();
            boolean encontrado = false;

            for (Libro lib : libros) {
                if (lib.getCodigo().equals(codMod)) {
                    System.out.println("Ingrese nuevos datos:");
```

```
                System.out.print("Nombre: ");
                lib.setNombre(leer.nextLine());

                System.out.print("Año: ");
                lib.setAnio(leer.nextLine());

                System.out.print("Edición: ");
                lib.setEdicion(leer.nextLine());

                System.out.print("Páginas: ");
lib.setNumeroPaginas(Integer.parseInt(leer.nextLine()));

                System.out.print("Área: ");
                lib.setArea(leer.nextLine());

                System.out.print("Tomo: ");
                lib.setTomo(leer.nextLine());

                guardarLibros(libros);
                System.out.println("✓ Libro modificado y guardado.");
                encontrado = true;
                break;
            }
        }
        if (!encontrado)
            System.out.println("✗ Libro no encontrado.");
        break;
```

```
            case 3:
                System.out.print("Ingrese código del libro a eliminar: ");
                String codDel = leer.nextLine();
                boolean eliminado = libros.removeIf(x ->
x.getCodigo().equals(codDel));
                if (eliminado) {
                    guardarLibros(libros);
                    System.out.println("✓ Libro eliminado.");
                } else {
                    System.out.println("✗ Código no encontrado.");
                }
                break;

            case 4:
                System.out.println("\n--- LISTA DE LIBROS ---");
                for (Libro lib : libros) {
                    System.out.println(lib.verDatos());
                }
                break;
        }
    } while (opcion != 0);

    System.out.println("Programa finalizado.");
  }

  // ---------------------------------------------
  // MÉTODO PARA GUARDAR LIBROS EN ARCHIVO CSV
  // ---------------------------------------------
  public static void guardarLibros(ArrayList<Libro> libros) {
    try (PrintWriter pw = new PrintWriter(new FileWriter(ARCHIVO))) {
```

```
            for (Libro l : libros) {
                pw.println(
                        l.getCodigo() + ";" +
                        l.getNombre() + ";" +
                        l.getAnio() + ";" +
                        l.getEdicion() + ";" +
                        l.getNumeroPaginas() + ";" +
                        l.getArea() + ";" +
                        l.getTomo()
                );
            }
        } catch (Exception e) {
            System.out.println("Error al guardar el archivo: " +
e.getMessage());
        }
    }


    // ------------------------------------------------
    // MÉTODO PARA CARGAR LIBROS DESDE ARCHIVO CSV
    // ------------------------------------------------
    public static void cargarLibros(ArrayList<Libro> libros) {
        File archivo = new File(ARCHIVO);


        if (!archivo.exists()) {
            return; // No hay archivo todavía
        }


        try (BufferedReader br = new BufferedReader(new
FileReader(archivo))) {


            String linea;
```

```
            while ((linea = br.readLine()) != null) {
                String[] datos = linea.split(";");
                Libro l = new Libro(
                        datos[0],
                        datos[1],
                        datos[2],
                        datos[3],
                        Integer.parseInt(datos[4]),
                        datos[5],
                        datos[6]
                );
                libros.add(l);
            }

        } catch (Exception e) {
            System.out.println("Error al leer el archivo: " + e.getMessage());
        }
    }
}
```

C) Versión completa que combine consola, almacenamiento en archivo y una clase adicional Biblioteca para implementar una arquitectura modular y limpia

En esta versión el programa representa la implementación más completa del sistema de gestión de libros, integrando los principios de la Programación Orientada a Objetos (POO), colecciones dinámicas y persistencia de datos mediante archivos. En esta versión, se separan claramente las responsabilidades en tres componentes: la clase Libro, que modela cada libro como un objeto con atributos como código, nombre, año, edición, número de páginas, área y tomo; la clase Biblioteca, que centraliza la gestión de la colección

y la persistencia en un archivo .csv; y la clase AppConsole, que se encarga de la interacción con el usuario a través de un menú en consola.

El sistema permite realizar todas las operaciones fundamentales de gestión de libros: registrar nuevos libros, modificar datos existentes, eliminar registros y listar todos los libros almacenados. Cada acción se ejecuta sobre los objetos contenidos en la clase Biblioteca, garantizando que los cambios se reflejen automáticamente en el archivo externo y en la memoria del programa. Esta versión demuestra cómo combinar los conceptos de POO, colecciones dinámicas y manejo de archivos en Java de manera modular y profesional, ofreciendo una estructura limpia, escalable y de fácil mantenimiento. Además, la persistencia asegura que la información de los libros se conserve entre ejecuciones, proporcionando un ejemplo práctico de desarrollo de aplicaciones robustas de gestión bibliográfica.

En esta versión no requiere hacer cambios en la clase Libro, pero si en la clase Biblioteca y AppConsole.

CODIGO EN JAVA

Biblioteca.java

```
//-------------------------------------------------------------------------------------------
 * Programa que implementa la clase Biblioteca
* Esta clase centraliza toda la lógica de gestión y persistencia:
 //-------------------------------------------------------------------------------------------
package classes;

import java.io.*;

import java.util.ArrayList;


public class Biblioteca {

    private ArrayList<Libro> libros;

    private final String archivo;


    // Constructor

    public Biblioteca(String archivo) {

        this.archivo = archivo;
```

```
        libros = new ArrayList<>();
        cargarLibros();
    }

    // ---------------------------------
    // Agregar libro
    // ---------------------------------
    public void agregarLibro(Libro l) {
        libros.add(l);
        guardarLibros();
    }

    // ---------------------------------
    // Modificar libro
    // ---------------------------------
    public boolean modificarLibro(String codigo, Libro nuevo) {
        for (int i = 0; i < libros.size(); i++) {
            if (libros.get(i).getCodigo().equals(codigo)) {
                libros.set(i, nuevo);
                guardarLibros();
                return true;
            }
        }
        return false;
    }

    // ---------------------------------
    // Eliminar libro
    // ---------------------------------
    public boolean eliminarLibro(String codigo) {
        boolean eliminado = libros.removeIf(l ->
l.getCodigo().equals(codigo));
        if (eliminado) {
            guardarLibros();
```

```
    }
    return eliminado;
}


// --------------------------------
// Listar libros
// --------------------------------
public void listarLibros() {
    if (libros.isEmpty()) {
        System.out.println("No hay libros registrados.");
    } else {
        for (Libro l : libros) {
            System.out.println(l.verDatos());
        }
    }
}


// --------------------------------
// Guardar libros en archivo CSV
// --------------------------------
private void guardarLibros() {
    try (PrintWriter pw = new PrintWriter(new FileWriter(archivo))) {
        for (Libro l : libros) {
            pw.println(
                    l.getCodigo() + ";" +
                    l.getNombre() + ";" +
                    l.getAnio() + ";" +
                    l.getEdicion() + ";" +
                    l.getNumeroPaginas() + ";" +
                    l.getArea() + ";" +
                    l.getTomo()
            );
```

```
            }
        } catch (IOException e) {
            System.out.println("Error al guardar el archivo: " +
e.getMessage());
        }
    }

    // ------------------------------
    // Cargar libros desde archivo CSV
    // ------------------------------
    private void cargarLibros() {
        File f = new File(archivo);
        if (!f.exists()) return;
        try (BufferedReader br = new BufferedReader(new FileReader(f))) {
            String linea;
            while ((linea = br.readLine()) != null) {
                String[] datos = linea.split(";");
                if (datos.length == 7) {
                    Libro l = new Libro(
                            datos[0], datos[1], datos[2], datos[3],
                            Integer.parseInt(datos[4]), datos[5], datos[6]
                    );
                    libros.add(l);
                }
            }
        } catch (IOException e) {
            System.out.println("Error al cargar el archivo: " + e.getMessage());
        }
    }
}
```

AppConsole.java

```
//-----------------------------------------------------------------------------------------------
* Programa que implementa la clase AppConsole
* Esta clase interactúa con el usuario mediante un menú en consola y
 * delega toda la gestión a la clase Biblioteca
 //-----------------------------------------------------------------------------------------------

package classes;

import java.util.Scanner;

public class AppConsole {

    public static void main(String[] args) {

        Biblioteca biblioteca = new Biblioteca("libros.csv");

        Scanner leer = new Scanner(System.in);

        int opcion;

        do {

            System.out.println("\n--- GESTIÓN DE LIBROS ---");

            System.out.println("1. Registrar nuevo libro");

            System.out.println("2. Modificar libro");

            System.out.println("3. Eliminar libro");

            System.out.println("4. Listar libros");

            System.out.println("0. Salir");

            System.out.print("Ingrese opción: ");

            opcion = leer.nextInt();

            leer.nextLine(); // Limpiar buffer

            switch (opcion) {

                case 1:

                    Libro l = crearLibro(leer);

                    biblioteca.agregarLibro(l);

                    System.out.println("✓ Libro registrado y guardado.");

                    break;
```

```
            case 2:
                System.out.print("Ingrese código del libro a modificar: ");
                String codMod = leer.nextLine();
                Libro nuevo = crearLibro(leer);
                if (biblioteca.modificarLibro(codMod, nuevo)) {
                    System.out.println("✓ Libro modificado y guardado.");
                } else {
                    System.out.println("✗ Libro no encontrado.");
                }
                break;
            case 3:
                System.out.print("Ingrese código del libro a eliminar: ");
                String codDel = leer.nextLine();
                if (biblioteca.eliminarLibro(codDel)) {
                    System.out.println("✓ Libro eliminado.");
                } else {
                    System.out.println("✗ Código no encontrado.");
                }
                break;
            case 4:
                System.out.println("\n--- LISTA DE LIBROS ---");
                biblioteca.listarLibros();
                break;
        }
    } while (opcion != 0);
    System.out.println("Programa finalizado.");
}

// --------------------------------
// Método auxiliar para crear libro desde consola
```

```
        // --------------------------------
        private static Libro crearLibro(Scanner leer) {
            Libro l = new Libro();
            System.out.print("Código: ");
            l.setCodigo(leer.nextLine());

            System.out.print("Nombre: ");
            l.setNombre(leer.nextLine());

            System.out.print("Año: ");
            l.setAnio(leer.nextLine());

            System.out.print("Edición: ");
            l.setEdicion(leer.nextLine());

            System.out.print("Páginas: ");
            l.setNumeroPaginas(Integer.parseInt(leer.nextLine()));

            System.out.print("Área: ");
            l.setArea(leer.nextLine());

            System.out.print("Tomo: ");
            l.setTomo(leer.nextLine());

            return l;
        }
}
```

EJEMPLO N°02-005

Con la siguiente descripción de requerimientos desarrollar el programa en java para la gestión de bienes y suministros, así mismo el almacenamiento de los datos realizarlo en archivo planos:

En el proceso de "**Bienes y Suministros**" de la empresa de turismo "**Pata Amarilla**" que pertenece al área de Administración, se realizan las actividades de manera manual, por lo que se requiere sistematizar todo ello con la ayuda de un software, el dueño del proceso de "**Bienes y Suministros**" detalla el procedimiento y actividades de cada área:

- ✓ El proceso involucra cinco (5) áreas: Administración, Logística, Almacén, Patrimonio, Áreas usuarias.
- ✓ El área de Logística funciona de la siguiente forma:
- ➢ Recibe las solicitudes de requerimientos de bienes o suministros de las diferentes áreas de la empresa.
- ➢ Cada solicitud tiene un responsable que está adscrito a un área.
- ➢ Cada solicitud es autorizada por el jefe del área y posteriormente por el área de Administración.
- ➢ De la solicitud se requiere la siguiente información: Número de la solicitud (consecutivo), fecha, responsable, área, meta presupuestal. En cada solicitud se pueden contener uno o muchos ítems con la siguiente información: código, nombre del ítem, cantidad solicitada, unidad de medida, valor unitario.
- ➢ Cada ítem es identificado por un código único y es de carácter devolutivo (suministro) (útiles de escritorio, útiles de limpieza, etc.) o un bien (computadora, escritorio, etc.).
- ➢ La solicitud es enviada al área de Logística para atender si es que hay en stock o realizar la compra a uno o varios proveedores. Para realizar la compra se juntan todas las solicitudes para tener la cantidad total de ítems de cada rubro a comprar.
- ➢ Las cotizaciones son realizadas con uno o varios proveedores de los bienes solicitados.
- ➢ Para comprar bienes primero se realiza la cotización dos o tres proveedores para determinar la mejor oferta en calidad y precio que un

proveedor ofrece, una vez elegido el proveedor se genera la orden de compra, con los siguientes datos: número de la **orden de compra**, nombre del proveedor al cual se le va a realizar la compra, fecha de la orden, monto total de la orden, fecha de entrega. Cada orden puede tener asociado uno o varios ítems, cada ítem debe tener los siguientes datos: código del ítem, nombre del ítem, cantidad de compra, unidad de medida del ítem, valor unitario y sub total.

- La orden de compra es aprobada por el área de Administración luego el área de Logística envía o hace entrega al proveedor elegido.

- ✓ El área de Almacén funciona de la siguiente forma:
  - Recepciona los bienes que llegan de los proveedores y distribuye con una pecosa a las áreas que realizaron las solicitudes.
  - El proveedor hace entrega a Almacén los ítems detallado en la Orden de Compra, para ello adjunta la guía de remisión y factura para su pago correspondiente si todo está conforme, se registra una entrada de almacén por cada factura relacionada, con los siguientes datos: número de entrada, fecha, número de factura, Proveedor, total bienes, valor total, los ítems recibidos con la cantidad respectiva.
  - El área de Almacén hace entrega de los bienes a las diferentes áreas solicitantes atreves de una pecosa detallando cada ítem entregado y otros datos más como el número de salida, empleado solicitante del ítem o los ítems, cantidad, fecha de salida, fecha de entrega.
- ✓ El área Patrimonio es responsable de:
  - Administrar y controlar la ubicación de los bienes dentro de la empresa, por esto antes de que el bien salga del Almacén es codificado y se genera un sticker que va pegado al bien.
  - Cada trabajador tiene a su cargo una o varios bienes, esto permite controlar su ubicación.

DESARROLLO

La solución implementa un sistema de gestión de bienes y suministros para la empresa “Pata Amarilla” utilizando Programación Orientada a Objetos y colecciones dinámicas en Java, con persistencia de datos en archivos JSON mediante Gson. Cada entidad del proceso se modela con clases específicas: Item para bienes y suministros, Solicitud para las solicitudes de las áreas, OrdenCompra para las compras a proveedores y Almacen para el control de stock y distribución.

El programa permite registrar solicitudes con múltiples ítems, calcular totales y listar la información de manera ordenada. La persistencia en archivos asegura que los datos se mantengan entre ejecuciones, facilitando la sistematización de un proceso que anteriormente era manual. Esta implementación modular y escalable sirve como ejemplo práctico de cómo organizar, almacenar y manipular información de forma eficiente en un sistema de gestión empresarial.

CODIGO EN JAVA

Itrem.java

```
//-----------------------------------------------------------------------------------------------
* Programa que implementa la clase Item
* Representa cada bien o suministro.:
//-----------------------------------------------------------------------------------------------

package models;
public class Item {
    private String codigo;
    private String nombre;
    private double cantidad;
    private String unidadMedida;
    private double valorUnitario;
    private String tipo; // "Bien" o "Suministro"

    public Item() {}

    public Item(String codigo, String nombre, double cantidad, String
unidadMedida, double valorUnitario, String tipo) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.cantidad = cantidad;
        this.unidadMedida = unidadMedida;
        this.valorUnitario = valorUnitario;
        this.tipo = tipo;
    }
    public double getSubTotal() {
        return cantidad * valorUnitario;
    }
```

```
    @Override
    public String toString() {
        return codigo + " | " + nombre + " | " + cantidad + " " + unidadMedida
+ " | " + valorUnitario + " | " + tipo;
    }

    // Getters y setters
    public String getCodigo() { return codigo; }
    public void setCodigo(String codigo) { this.codigo = codigo; }
    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    public double getCantidad() { return cantidad; }
    public void setCantidad(double cantidad) { this.cantidad = cantidad; }
    public String getUnidadMedida() { return unidadMedida; }
    public void setUnidadMedida(String unidadMedida) { this.unidadMedida
= unidadMedida; }
    public double getValorUnitario() { return valorUnitario; }
    public void setValorUnitario(double valorUnitario) { this.valorUnitario =
valorUnitario; }
    public String getTipo() { return tipo; }
    public void setTipo(String tipo) { this.tipo = tipo; }
}
```

Solicitud.java

```
//-----------------------------------------------------------------------------------------
* Programa que implementa la clase Solicitud
* Representa la solicitud de bienes o suministros por un área.
//-----------------------------------------------------------------------------------------
package models;

import java.util.ArrayList;

import java.util.List;
```

```
public class Solicitud {
    private int numeroSolicitud;
    private String fecha;
    private String responsable;
    private String area;
    private double metaPresupuestal;
    private List<Item> items;

    public Solicitud() {
        items = new ArrayList<>();
    }

    public Solicitud(int numeroSolicitud, String fecha, String responsable,
String area, double metaPresupuestal) {
        this.numeroSolicitud = numeroSolicitud;
        this.fecha = fecha;
        this.responsable = responsable;
        this.area = area;
        this.metaPresupuestal = metaPresupuestal;
        this.items = new ArrayList<>();
    }

    public void agregarItem(Item item) {
        items.add(item);
    }

    public double calcularTotalSolicitud() {
        return items.stream().mapToDouble(Item::getSubTotal).sum();
    }
```

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("Solicitud N°").append(numeroSolicitud)
      .append(" | Fecha: ").append(fecha)
      .append(" | Responsable: ").append(responsable)
      .append(" | Área: ").append(area)
      .append(" | Meta: ").append(metaPresupuestal).append("\n");
    for (Item i : items) {
        sb.append("  ").append(i.toString()).append("\n");
    }
    sb.append("Total solicitud: ").append(calcularTotalSolicitud());
    return sb.toString();
}
```

```
    // Getters y setters
    public int getNumeroSolicitud() { return numeroSolicitud; }
    public void setNumeroSolicitud(int numeroSolicitud)
 { this.numeroSolicitud = numeroSolicitud; }
    public String getFecha() { return fecha; }
    public void setFecha(String fecha) { this.fecha = fecha; }
    public String getResponsable() { return responsable; }
    public void setResponsable(String responsable) { this.responsable =
 responsable; }
    public String getArea() { return area; }
    public void setArea(String area) { this.area = area; }
    public double getMetaPresupuestal() { return metaPresupuestal; }
    public void setMetaPresupuestal(double metaPresupuestal)
 { this.metaPresupuestal = metaPresupuestal; }
    public List<Item> getItems() { return items; }
    public void setItems(List<Item> items) { this.items = items; }
}
```

OrdenCompra.java

```
//-------------------------------------------------------------------------------------------
* Programa que implementa la clase OrdenCompra
* Representa la orden generada hacia un proveedor.
//-------------------------------------------------------------------------------------------
package models;
import java.util.ArrayList;
import java.util.List;
public class OrdenCompra {
    private int numeroOrden;
    private String proveedor;
    private String fechaOrden;
    private double montoTotal;
```

```
    private String fechaEntrega;
    private List<Item> items;

    public OrdenCompra() {
        items = new ArrayList<>();
    }

    public OrdenCompra(int numeroOrden, String proveedor, String
fechaOrden, String fechaEntrega) {
        this.numeroOrden = numeroOrden;
        this.proveedor = proveedor;
        this.fechaOrden = fechaOrden;
        this.fechaEntrega = fechaEntrega;
        this.items = new ArrayList<>();
        this.montoTotal = 0;
    }

    public void agregarItem(Item item) {
        items.add(item);
        montoTotal += item.getSubTotal();
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Orden N°").append(numeroOrden)
          .append(" | Proveedor: ").append(proveedor)
          .append(" | Fecha orden: ").append(fechaOrden)
          .append(" | Fecha entrega: ").append(fechaEntrega)
```

```
            .append(" | Monto total: ").append(montoTotal).append("\n");
        for (Item i : items) {
            sb.append("  ").append(i.toString()).append("\n");
        }
        return sb.toString();
    }

    // Getters y setters
    public int getNumeroOrden() { return numeroOrden; }
    public void setNumeroOrden(int numeroOrden) { this.numeroOrden =
numeroOrden; }
    public String getProveedor() { return proveedor; }
    public void setProveedor(String proveedor) { this.proveedor =
proveedor; }
    public String getFechaOrden() { return fechaOrden; }
    public void setFechaOrden(String fechaOrden) { this.fechaOrden =
fechaOrden; }
    public double getMontoTotal() { return montoTotal; }
    public void setMontoTotal(double montoTotal) { this.montoTotal =
montoTotal; }
    public String getFechaEntrega() { return fechaEntrega; }
    public void setFechaEntrega(String fechaEntrega) { this.fechaEntrega =
fechaEntrega; }
    public List<Item> getItems() { return items; }
    public void setItems(List<Item> items) { this.items = items; }
}
```

Almacen.java

```
//-----------------------------------------------------------------------------------------------
* Programa que implementa la clase Almacen
* Encargada de recepcionar y distribuir los bienes.
//-----------------------------------------------------------------------------------------------
package models;

import java.util.ArrayList;

import java.util.List;


public class Almacen {


   private List<Item> stock;


   public Almacen() {
      stock = new ArrayList<>();
   }


   public void agregarStock(Item item) {
      stock.add(item);
   }


   public boolean entregarItem(String codigo, double cantidad) {
      for (Item i : stock) {
         if (i.getCodigo().equals(codigo) && i.getCantidad() >= cantidad) {
            i.setCantidad(i.getCantidad() - cantidad);
            return true;
         }
      }
      return false;
   }
```

```
    public void listarStock() {
        System.out.println("--- STOCK DEL ALMACÉN ---");
        for (Item i : stock) {
            System.out.println(i);
        }
    }

    public List<Item> getStock() {
        return stock;
    }
}
```

GestorJSON.java

```
//-----------------------------------------------------------------------------------------
* Programa que implementa la clase GestorJSON
* Clase auxiliar para persistencia en archivos JSON usando Gson.
//-----------------------------------------------------------------------------------------
package utils;

import com.google.gson.Gson;

import com.google.gson.reflect.TypeToken;

import java.io.*;

import java.util.List;

public class GestorJSON {

    private static Gson gson = new Gson();

    public static <T> void guardar(String archivo, List<T> lista) {

        try (Writer writer = new FileWriter(archivo)) {

            gson.toJson(lista, writer);

        } catch (IOException e) {

            System.out.println("Error al guardar archivo JSON: " +
e.getMessage());
```

```
        }
    }

    public static <T> List<T> cargar(String archivo, TypeToken<List<T>>
typeToken) {
        try (Reader reader = new FileReader(archivo)) {
            return gson.fromJson(reader, typeToken.getType());
        } catch (FileNotFoundException e) {
            return null; // Archivo no existe
        } catch (IOException e) {
            System.out.println("Error al leer archivo JSON: " +
e.getMessage());
            return null;
        }
    }
}
```

AppeConsole.java

```
//-----------------------------------------------------------------------------------------
* Programa que implementa la clase AppeConsole
* Ejemplo de interacción con el usuario, registro y listado:.
//-----------------------------------------------------------------------------------------
package app;

import models.*;

import utils.GestorJSON;

import com.google.gson.reflect.TypeToken;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;


public class AppConsole {
```

```
    private static List<Solicitud> solicitudes = new ArrayList<>();
    private static Scanner leer = new Scanner(System.in);

    public static void main(String[] args) {
        // Cargar solicitudes desde JSON
        List<Solicitud> cargadas = GestorJSON.cargar("solicitudes.json",
new TypeToken<List<Solicitud>>() {});
        if (cargadas != null) solicitudes = cargadas;

        int opcion;
        do {
            System.out.println("\n--- BIENES Y SUMINISTROS ---");
            System.out.println("1. Registrar solicitud");
            System.out.println("2. Listar solicitudes");
            System.out.println("0. Salir");
            System.out.print("Opción: ");
            opcion = Integer.parseInt(leer.nextLine());

            switch (opcion) {
                case 1:
                    Solicitud s = crearSolicitud();
                    solicitudes.add(s);
                    GestorJSON.guardar("solicitudes.json", solicitudes);
                    System.out.println("✓ Solicitud registrada.");
                    break;
                case 2:
                    listarSolicitudes();
                    break;
            }
```

```
    } while (opcion != 0);
    System.out.println("Programa finalizado.");
}

private static Solicitud crearSolicitud() {
    System.out.print("Número solicitud: ");
    int numero = Integer.parseInt(leer.nextLine());
    System.out.print("Fecha: ");
    String fecha = leer.nextLine();
    System.out.print("Responsable: ");
    String responsable = leer.nextLine();
    System.out.print("Área: ");
    String area = leer.nextLine();
    System.out.print("Meta presupuestal: ");
    double meta = Double.parseDouble(leer.nextLine());

    Solicitud s = new Solicitud(numero, fecha, responsable, area, meta);

    System.out.print("Cantidad de ítems: ");
    int n = Integer.parseInt(leer.nextLine());
    for (int i = 0; i < n; i++) {
        System.out.println("Ítem " + (i + 1));
        System.out.print("Código: ");
        String cod = leer.nextLine();
        System.out.print("Nombre: ");
        String nombre = leer.nextLine();
        System.out.print("Cantidad: ");
        double cant = Double.parseDouble(leer.nextLine());
```

```
            System.out.print("Unidad de medida: ");
            String unidad = leer.nextLine();
            System.out.print("Valor unitario: ");
            double valor = Double.parseDouble(leer.nextLine());
            System.out.print("Tipo (Bien/Suministro): ");
            String tipo = leer.nextLine();
            s.agregarItem(new Item(cod, nombre, cant, unidad, valor, tipo));
        }
        return s;
    }

    private static void listarSolicitudes() {
        System.out.println("\n--- LISTA DE SOLICITUDES ---");
        for (Solicitud s : solicitudes) {
            System.out.println(s);
            System.out.println("----------------------------------");
        }
    }
}
```

CAPÍTULO 3. INTERFAZ GRÁFICA

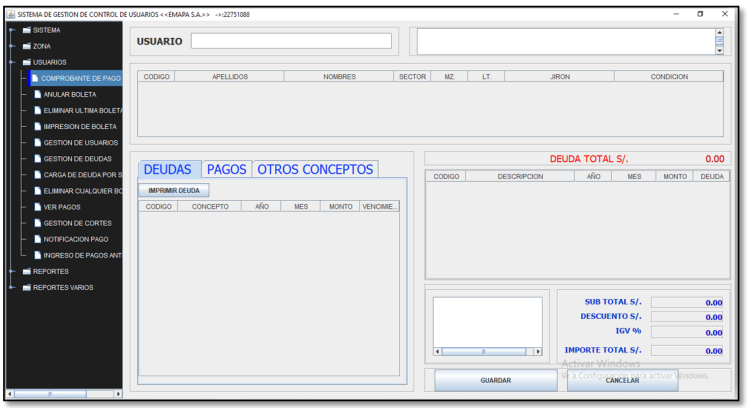


IMAGEN N° 03-0001: Interfaz gráfica de un aplicativo

3.1 INTERFAZ GRÁFICA-GUI

La interfaz gráfica en un aplicativo permite aumentar la usabilidad, la interacción programa – usuario, se hace más fácil, ya que intervienen el uso del mouse y de ventanas gráficas para la manipulación y procesamiento de los datos, una interfaz gráfica se puede observar en la IMAGEN N° 03-001.

EJEMPLO N°03-001

Código en java que genera una ventana con tres botones, en este caso utilizando el java.awt (que proporciona una serie de clases e interfaces para realizar programas que se ejecuten en un entorno gráfico).

```
package ejemplo01;
import java.awt.*;
public class EJEMPLO01 extends Frame{
private Button b1,b2,b3;
public EJEMPLO01()
```

```
    {
        //el constructor del padre recibe el título de la
        //ventana
          super("Ventana principal");

        //definiendo el layout que tendra la ventana:FlowLayout
        setLayout(new FlowLayout());
        //se instancian los botones
        b1=new Button("BOTON A");
        add(b1);
        b2=new Button("BOTON C");
        add(b2);

        b3=new Button("BOTON D");
        add(b3);
        //Se define el tamaño de la ventana
        setSize(400,400);
        setVisible(true);

        //para que la ventana este al centro
        setLocationRelativeTo(null);
    }
    public static void main(String[] args)
    {
        EJEMPLO01 f=new EJEMPLO01();
    }
}
```

Al ejecutar el programa se visualiza una ventana como se muestra en la IMAGEN N° 03-0002.

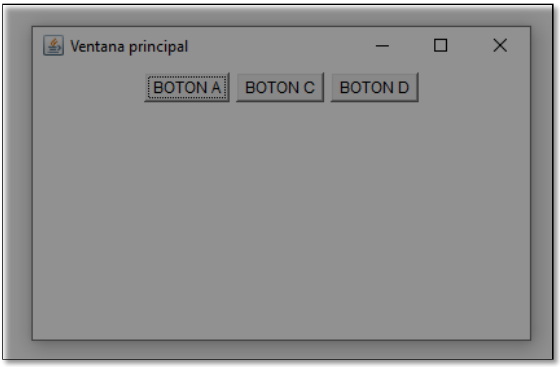


IMAGEN N° 03-0002: Ventana con tres botones

Los componentes que se utilizan en Java para crear interfaces graficas de usuario se agrupan en dos paquetes

- ✓ **java.awt**

  Los componentes AWT dependen de las facilidades graficas ofrecidas por cada sistema operativo.

- ✓ **Java.swing**

  100% java y, por tanto, completamente independiente de la plataforma.
  Los componentes gráficos se pintan en tiempo de ejecución.

En este libro nos ocuparemos del segundo paquete: Java.swing.*, ya que es 100% java y completamente independiente de la plataforma.

3.2 JAVA.SWING

Un entorno grafico utiliza componentes que aumenta usabilidad de un software, un componente es un elemento gráfico, que sirve de interfaz al usuario con un programa; también llamado control.

Cada uno de los elementos de un interfaz gráfico de usuario en Java se representa por una instancia de una determinada clase. Por ejemplo, si queremos que nuestro interfaz tenga un botón, tendremos que, entre otras cosas, instanciar un objeto de la clase **JButton**. Para configurar los aspectos gráficos de los componentes usaremos métodos en la clase determinada. Por ejemplo, podremos cambiar el texto del botón usando el método **void setText(String texto)** de la clase **JButton**.

En el siguiente cuadro se muestran los distintos tipos de elementos que se pueden encontrar en Swing y el nombre de la clase que los representa:

Componentes Atómicos	**JLabel** – Etiqueta, muestra imágenes y texto. El texto se puede formatear con etiquetas HTML. **JButton** – Botón **JCheckBox** – Casilla de verificación **JRadioButton** – Botón de radio, usado para seleccionar una opción entre varias **JToggleButton** – Botón que se queda presionado al pulsarle **JComboBox** – Control que muestra un elemento y pulsando en una flecha se pueden ver otros elementos **JScrollBar** – Barra de desplazamiento, usada en los contenedores que permiten que su contenido sea más grande que ellos. Nunca usaremos este componente directamente. **JSeparator** – Usado en los menus y barras de herramientas para separar opciones. **JSlider** - Deslizador **JSpinner** – Campo de texto con botones para elegir el elemento siguiente o anterior. Se puede usar para números, fechas o elementos propios. **JProgressBar** – Barra de progreso
Componentes complejos	**JTable** – Tabla **JTree** - Árbol **JList** – Lista de elementos **JFileChooser** – Selector de ficheros **JColorChooser** – Selector de color **JOptionPane** – Cuadro de diálogo personalizable
Componentes de texto	**JTextField** – Campo de texto **JFormattedTextField** – Campo de texto formateado **JPasswordField** – Campo de texto para contraseñas **JTextArea** – Área de texto **JTextPane** – Área de texto formateado y con imágenes **JEditorPane** – Área de texto formateado y con imágenes que permite la edición del contenido

Contenedores	**JPanel** – Contenedor **JScrollPane** – Contenedor con barras de desplazamiento **JSplitPane** – Contenedor dividido en dos partes **JTabbedPane** – Contenedor con pestañas **JDesktopPane** – Contenedor para incluir ventanas dentro **JToolBar** – Barra de herramientas
Contenedores de alto nivel	**JFrame** – Ventana de aplicación **JDialog** – Cuadro de diálogo **JWindow** – Ventana sin marco **JInternalFrame** – Ventana interna
Menus	**JMenu** – Un botón que al ser pulsado despliega un menú. **JCheckBoxMenuItem** – Elemento del menú como botón de chequeo. **JRadioButtonMenuItem** – Elemento del menú como botón de selección. **JPopupMenu** – Menú de elementos. **JMenuItem** – Un botón que se encuentra en un menú. **JMenuBar** – Barra de menús.
Otros	**JToolBar** – Barra de herramientas **JToolTip** – Tooltip

Para visualizar los controles en NetBeans en modo grafico en el menú Window – IDE Tools- Palette, tal como se muestra en la IMAGEN N° 03-003

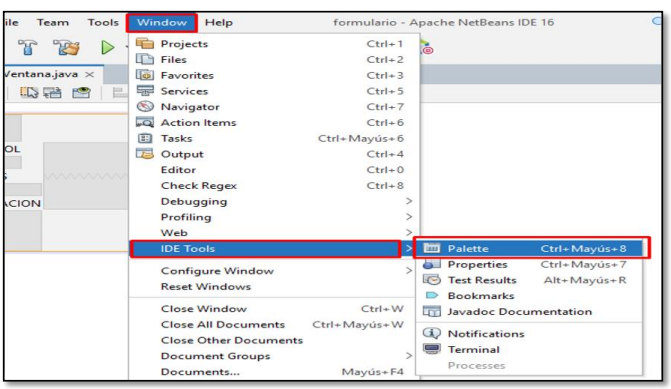


IMAGEN N° 03-0003: Opciones para visualizar la paleta de controles en NetBeans

En la IMAGEN N° 03-0004 se muestra los controles en modo gráfico en NetBeans, que para utilizarlo solo basta con arrastrar con el mouse al contenedor en donde se requiera.

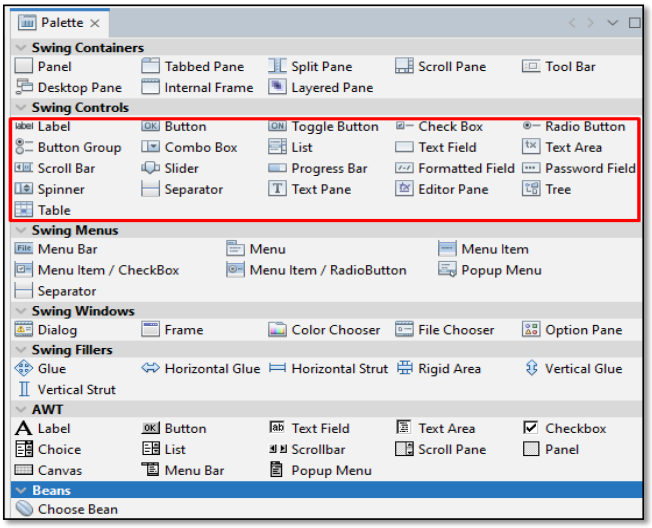


IMAGEN N° 03-0004: Paleta de controles en modo grafico en NetBeans

3.3 CARACTERÍSTICAS

- ✓ Todos los componentes heredan de *javax.swing.JComponent*
- ✓ JFrame será la base para la aplicación principal.
- ✓ JDialog construirá los dialógos (ventanas).
- ✓ El resto de clases serán componentes simples.
- ✓ Usar en todas las clases *import javax.swing.*;* y *import java.awt.*;*
- ✓ Todas las componentes permiten fijar un mnemotécnico: *componente.setMnemonic(KeyEvent.VK_letra);*
- ✓ Todas las componentes permiten fijar “tooltips”.

3.4 CONTROLES SWING

Empezaremos describiendo al JFrame, contenedor de alto nivel que utilizaremos en todos los ejemplos.

3.4.1 Clase JFrame

Esto es uno de los componentes básicos que se requiere para implementar software visual con el paquete **Java.swing** y sirve como contenedor de otros controles. Para agregar un formulario en NetBeans, se podría primero agregar un paquete **Java Package**. "Formularios" luego sobre ello agregar los formularios que se requiera, con clic derecho y se tendrá el menú desplegable, tal como se muestra en la IMAGEN N° 03-0005.

EJEMPLO N°03-002

El formulario también se puede agregar con código en java:

```
package javaapplication10;
import javax.swing.*;
public class Main
{
    public static void main(String[] ar)
    {
      JFrame formulario=new JFrame("Ventana");
      formulario.setBounds(10, 10,250,260);
      formulario.setVisible(true);
    }
}
```

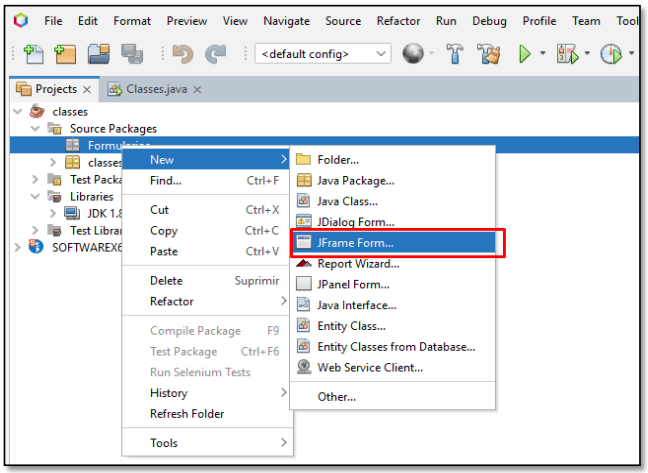


IMAGEN N° 03-0005: Agregando un JFrame en NetBeans

En el proyecto o paquete clic derecho y seleccionar JFrame tal como se muestra en la IMAGEN N° 03-0006 y se tendrá en modo diseño donde se puede cambiar el tamaño, modificar algunas propiedades, agregar controles, entre otros.

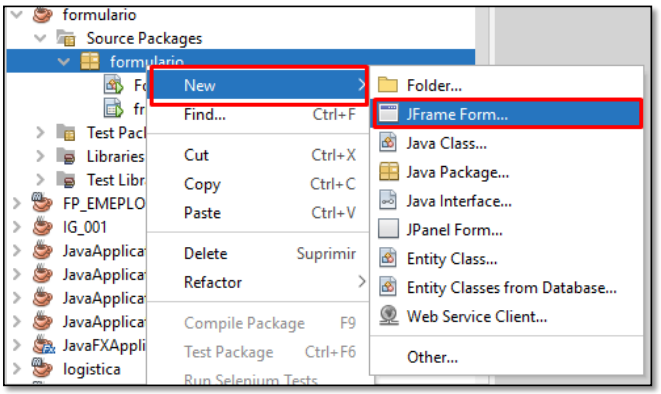


IMAGEN N° 03-0006: Menú desplegable para agregar un JFrame en NetBeans

En la IMAGEN N° 03-0007 se tiene un JFrame en modo diseño en donde se puede agregar controles que se requiera desde la paleta de controles.

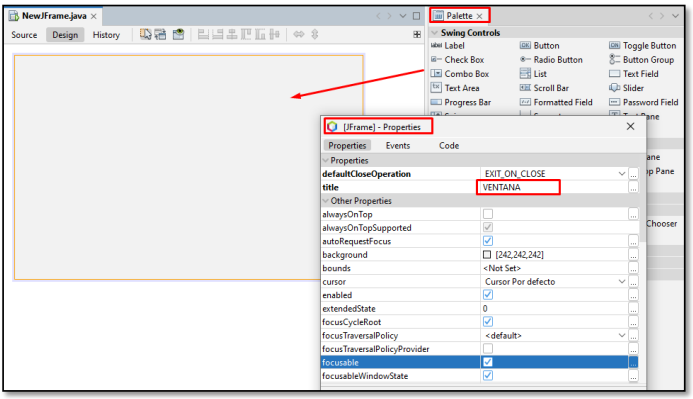


IMAGEN N° 03-0007: JFrame en modo grafico donde se puede agregar los controles que se requiera de la paleta de controles

EJEMPLO N°03-003

Aquí se tiene otra ventana que hereda todos los componentes de la clase JFrame:

```
package javaapplication11;
import javax.swing.*;
public class Main extends JFrame
{
    public Main()
    {
      setLayout(null);
    }
    public static void main(String[] ar)
    {
      JFrame formulario=new JFrame("Ventana");
```

```
        formulario.setBounds(10, 10,250,260);
        formulario.setVisible(true);
    }
}
```

La clase Main hereda de la clase JFrame, y en el constructor indicamos que ubicaremos los controles visuales con coordenadas absolutas mediante la desactivación del Layout heredado

3.4.2 Clase JLabel

La clase JLabel permite mostrar textos, para lo cual se importa del paquete java.swing.*;

EJEMPLO N°03-004

En el siguiente ejemplo mostraremos un formulario con dos controles JLabels:

```
package javaapplication11;
import javax.swing.*;
public class Main extends JFrame
{
   private JLabel lb1,lb2;
    public Main()
    {
      setLayout(null);
      lb1=new JLabel("Uso de label JLabel 1");
      add(lb1);//se adiciona el contro al contenedor
      lb1.setBounds(15,15,250,50);
      lb2=new JLabel("Uso de label JLabel 2");
      lb2.setBounds(15,20,250,100);
      add(lb2);//se adiciona el contro al contenedor
```

```
    }
    public static void main(String[] ar)
    {
      Main m=new Main();
      m.setBounds(10,10,250,260);
      m.setVisible(true);
    }
}
```

Al ejecutar el programa se mostraría como se ven en la IMAGEN N° 03-0008. Con la línea de código: lb1=new JLabel("Uso de label JLabel 1"); se instancia a la clase Jlabel con un parámetro que en este caso es el texto que se muestra.

El método setBounds(), establece las dimisiones que tendrá el rectángulo que contendrá el JLabel:

```
lb1.setBounds(15,15,250,50);
```

```
package javaapplication11;
import javax.swing.*;
public class Main extends JFrame
{  private JLabel lb1,lb2;
     public Main()
     {
       setLayout(null);
       lb1=new JLabel("Uso de label JLabel 1");
       add(lb1);//se adiciona el contro al contenedor
       lb1.setBounds(15,15,250,50);
       lb2=new JLabel("Uso de label JLabel 2");
       lb2.setBounds(15,20,250,100);
       add(lb2);//se adiciona el contro al contenedor

     }
    public static void main(String[] ar)
     {
       Main m=new Main();
       m.setBounds(10,10,250,260);
       m.setVisible(true);
     }
}
```

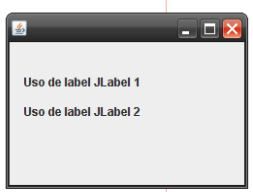


IMAGEN N° 03-0008: Ventana con dos controles JLabels

En modo gráfico con NetBeans se agrega un JFrame y el control jLabel, tal como se muestra en la IMAGEN N° 03-0009.

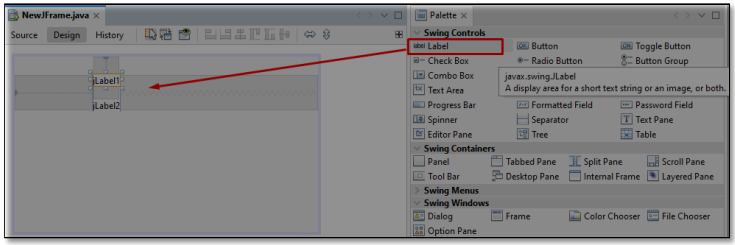


IMAGEN N° 03-0009: Se agrega al formulario el control jLabel

Para cambiar el texto clic derecho sobre el control y se desplegara el menú y en Edit Text cambiar por lo que se requiera, tal como se muestra en la IMAGEN N° 03-0010.

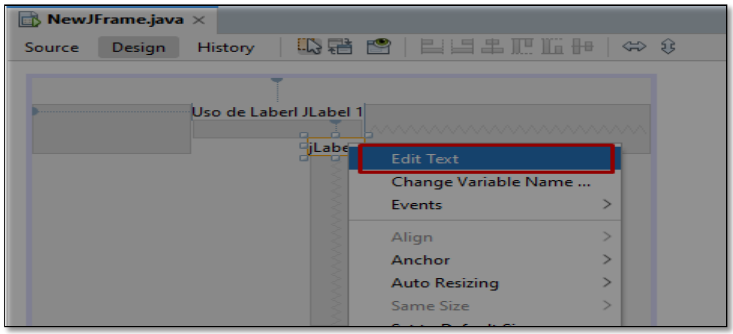


IMAGEN N° 03-0010: Menú desplegable para cambiar el texto del jLabel

3.4.2 Clase JTextField

La clase JTextField es un control que permite ingresar datos a nuestro programa, los métodos más usados:

- ✓ setBounds(int x, int y, int width, int height)
- ✓ setFont(Font f)

- ✓ setText(String t)
- ✓ setToolTipText(String text)
- ✓ setBackground(Color bg)
- ✓ public String getText()
- ✓ public void **setEditable**(boolean b)
- ✓ public void **setEnabled**(boolean enabled)

3.4.2.1 El método getText(): permite capturar la cadena de texto ingresado en el control, la cual es de tipo String.

3.4.2.2 El método setEditable(boolean b): Si esta instrucción, está establecida en *true,* permite que se pueda escribir sobre el *JTextField*. Si está establecida en *false*, impide que el usuario pueda modificar el contenido del *JTextField*.

3.4.2.3 El método setEnabled(boolean enabled): Este método hace que no se pueda modificar el contenido del JTextField, pero también lo pone de color gris, por lo que quizás el texto no se vea bien.

3.4.2.4 Eventos

3.4.2.4.1 ActionEvent: Este evento se dispara cuándo en el cuadro de texto se presiona la tecla Enter.

En el Ejemplo 3.5 se muestra dos controles de la clase JLabel y dos controles de la clase JTextField.

EJEMPLO N°03-005

Programa que muestra dos controles JTextField

```
package javaapplication11;
import javax.swing.*;
public class Main extends JFrame
{
  private JLabel lb1,lb2;
  private JTextField txt1;
```

```
  private JTextField txt2;
   public Main()
   {
     setLayout(null);
     lb1=new JLabel("DNI:");
     lb1.setBounds(15,15,100,30);

     lb2=new JLabel("Nombre: ");
     lb2.setBounds(15,50,100,35);
     txt1=new JTextField();
     txt2=new JTextField();

     txt1.setBounds(101,15,300,30);
     txt2.setBounds(101,50,300,35);
     add(lb1);//se adiciona el contro al contenedor
     add(lb2);//se adiciona el contro al contenedor
     add(txt1);//se adiciona el contro al contenedor
     add(txt2);//se adiciona el contro al contenedor
     setTitle("Formulario de ingreso de datos");
   }
   public static void main(String[] ar)
   {
     Main m=new Main();
     m.setBounds(10,10,450,160);
     m.setVisible(true);
   }
}
```

El programa del EJEMPLO N°03-005, muestra una un formulario con dos controles JTextField, que permitirá ingresar datos desde el teclado, al ejecutar dicho programa se verá como en la IMAGEN N° 03-0011.

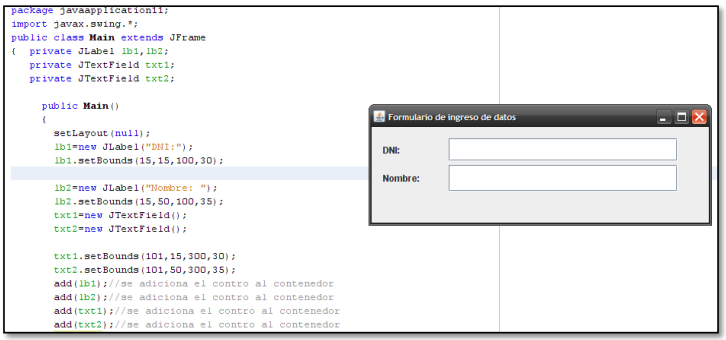


IMAGEN N° 03-00011: Ventana con dos controles JTextField

Para tener el formulario de la IMAGEN N° 03-0011, en NetBeans en modo diseño se agrega al formulario dos controles JLabel y dos controles JTextField, tal como se muestra en la IMAGEN N° 03-0012 y se cambian valores a los controles.

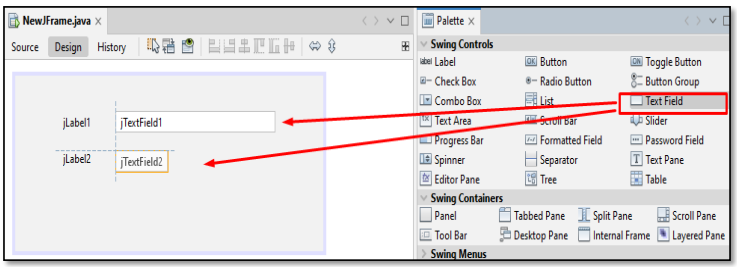


IMAGEN N° 03-0012: Ventana con dos controles jLabel y dos controles JTextField

Para cambiar el nombre de cualquier control con clic derecho desplegar el menú y seleccionar ***Change Variable Name***, tal como se muestra en la IMAGEN N° 03-0013.

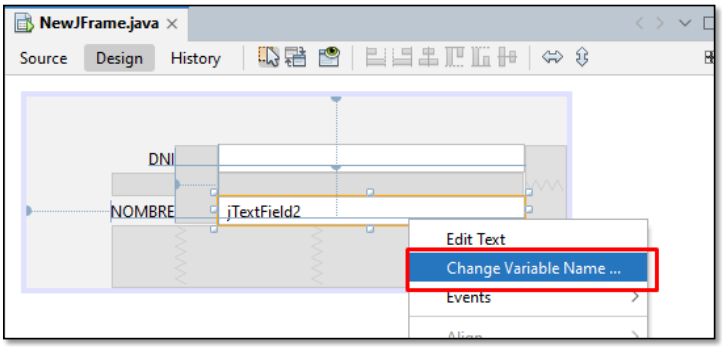


IMAGEN N° 03-0013: Opción Change Variable Name que permite cambiar nombre a cualquier control

3.4.3 Clase JButton

Este control es un botón que puede contener texto, gráficos, o ambos. Permite ejecutar código al realizar algún cambio en el estado del objeto instanciado de la clase JButton, su apariencia básica se muestra en la IMAGEN N° 03-0014

IMAGEN N° 03-0014: Botón con el texto “Sumar”

Métodos importantes:

- ✓ setText(“Texto”);
- ✓ setTooltipText(“Tooltip”);
- ✓ setBackground(new Color(R, G, B));
- ✓ setForeground(Color.color);
- ✓ setIcon(new ImageIcon(“ruta”));
- ✓ setFont(new Font(“tipo”, estilio, tamaño));
- ✓ setBounds(new Rectangle(posX,posY,tamX,tamY));

Y todos los métodos correspondientes a get.

EJEMPLO N°03-006

El siguiente programa genera un formulario con un botón instanciado de la clase JButton

```
package javaapplication10;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Main
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("FORMULARIO");
      frame.setBounds(new Rectangle(300,400, 600, 1000));
      frame.setBackground(new Color(255,0,0));
      JButton boton1 = new JButton();
      boton1.setBounds(new Rectangle(107, 50, 130, 60));
      boton1.setBackground(new Color(255,255,0));
      boton1.setForeground(Color.GREEN);
      boton1.setToolTipText("Prueba");
      boton1.setFont(new Font("Comic Sans MS",Font.BOLD, 14));
      boton1.setText("Boton");
      boton1.setMnemonic(KeyEvent.VK_B);
      frame.getContentPane().add(boton1);
      frame.pack();
      frame.setVisible(true);
}
}
```

En el ejemplo tenemos un contenedor de alto nivel JFrame. Cuando se ejecuta el botón se verá, así como en la IMAGEN N° 03-0015.

IMAGEN N° 03-0015: Ventana al correr el programa

Para tener el formulario de la IMAGEN N° 03-0015, en NetBeans en modo diseño se agrega al formulario un control JButton, tal como se muestra en la IMAGEN N° 03-0016 y se cambian valores a los controles.

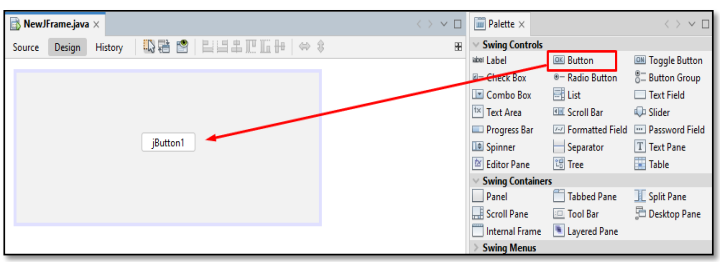


IMAGEN N° 03-0016: Formulario con un control JButton

Para cambiar de color al formulario, en el constructor agregar el siguiente código getContentPane().setBackground(new java.awt.Color(0,139,139));

```
public class NewJFrame extends javax.swing.JFrame {
    public NewJFrame() {
        initComponents();
        getContentPane().setBackground(new java.awt.Color( i: 255, i1: 255, i2: 0));
    }
```

IMAGEN N° 03-0017: Código que se agrega en el constructor del formulario para cambiar de color

3.4.4 Clase JCheckBox o caja de verificación

Es un control que representa dos estados (On y Off), son las cajas que permiten marcarlas o no para verificar alguna cosa en un formulario. Podemos ver una caja JCheckBox en la IMAGEN N° 03-0018.

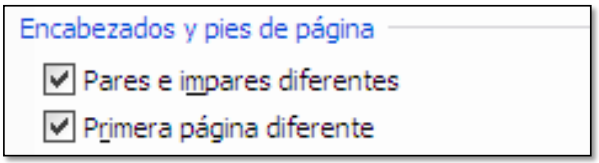


IMAGEN N° 03-0018: Control JCheckBox en ejecución

Métodos importantes:

- ✓ setText("Texto");
- ✓ setTooltipText("Tooltip");
- ✓ setBackground(new Color(R, G, B));
- ✓ setForeground(Color.color);
- ✓ setIcon(new ImageIcon("ruta"));
- ✓ setFont(new Font("tipo", estilio, tamaño));
- ✓ setBounds(new Rectangle(posX,posY,tamX,tamY));
- ✓ *isSelected()* y *setSelected(boolean)*

Y todos los métodos correspondientes a get.

EJEMPLO N°03-007

El siguiente programa muestre 3 objetos de la clase JCheckBox con etiquetas de tres deportes. Cuando se lo selecciona muestra el título del JFrame de todos los JCheckBox seleccionados hasta el momento. Al ejecutarse el programa debería verse como en la IMAGEN N° 03-0019.

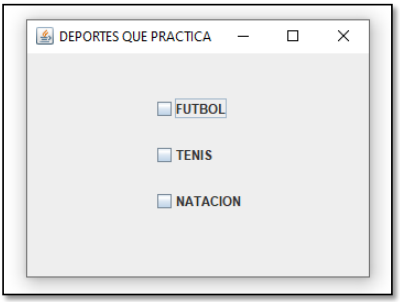


IMAGEN N° 03-0019: Ventana con 3 JCheckBox

DESARROLLO

Creando todo el diseño del formulario y los controles con código:

```
import javax.swing.*;
import javax.swing.event.*;
public class Main extends JFrame implements ChangeListener
{
   private JCheckBox check1,check2,check3;
   public Main()
   {
    setLayout(null);
    //----------------------------------------
    //se instancia los  checks y se adicional al control
    check1=new JCheckBox("FUTBOL");
    check1.setBounds(10,10,160,30);
    check1.addChangeListener(this);
    add(check1);
    //-----------------------------------
    check2=new JCheckBox("TENIS");
    check2.setBounds(10,50,160,30);
```

```
        check2.addChangeListener(this);
        add(check2);
        //----------------------------------
        check3=new JCheckBox("NATACION");
        check3.setBounds(10,90,160,30);
        check3.addChangeListener(this);
        add(check3);
    }
    public void stateChanged(ChangeEvent e)
    {
        String titulo="";
        if (check1.isSelected()==true) {
            titulo=titulo+"FUTBOL-";
        }
        if (check2.isSelected()==true) {
            titulo=titulo+"TENIS-";
        }
        if (check3.isSelected()==true) {
            titulo=titulo+"NATACION-";
        }
        setTitle(titulo);
    }
    public static void main(String[] args)
    {
        Main formulario1=new Main();
        formulario1.setBounds(0,0,300,200);
        formulario1.setVisible(true);
    }
}
```

En la IMAGEN N° 03-0020 se muestra la ejecución del programa, utilizando tres objetos de la clase JCheckBox

```
public void stateChanged(ChangeEvent e)
{
    String titulo="";
    if (check1.isSelected()==true) {
        titulo=titulo+"FUTBOL-";
    }
    if (check2.isSelected()==true) {
        titulo=titulo+"TENIS-";
    }
    if (check3.isSelected()==true) {
        titulo=titulo+"NATACION-";
    }
    setTitle( string:titulo);
}
```

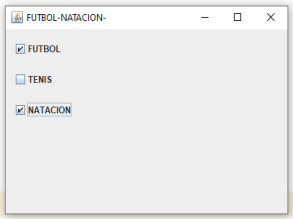


IMAGEN N° 03-0020: Ventana con 3 JCheckBox en ejecucion

Para capturar el cambio de estado del JCheckBox se implementa la interface ***ChangeListener*** que se encuentra en el paquete:

```
Import javax.swing.event.*;
```

Y cuando se declara la clase JFrame se indica que se implementara la interface ChangeListener:

```
public class Main extends JFrame implements ChangeListener
```

Se define 3 objetos de la clase JCheckBox:

```
private JCheckBox check1,check2,check3;
```

En la clase **Main** se crea un constructor en donde se instancia los tres objetos checks de la clase JCheckBox y se llama al método addChangeListener indicando quien procesará el evento de cambio de estado:

```
check1=new JCheckBox("FUTBOL");
check1.setBounds(10,10,160,30);
check1.addChangeListener(this);
```

```
add(check1);
```

Se implementa el método **stateChanged**(ChangeEvent e){}

Se verifica el estado de cada JCheckBox, si es verdadero concatenamos los deportes seleccionados:

```
String titulo="";
if (check1.isSelected()==true) {
 titulo=titulo+"FUTBOL-";
}
if (check2.isSelected()==true) {
 titulo=titulo+"TENIS-";
}
if (check3.isSelected()==true) {
 titulo=titulo+"NATACION-";
}
setTitle(titulo);
```

Diseño en modo gráfico:

En Netbeans con clic derecho agregamos JFrame, tal como se muestra en la IMAGEN N° 03-0021:

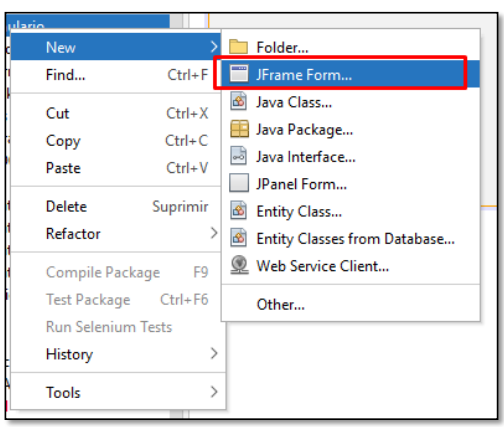


IMAGEN N° 03-0021: Se agrega un JFrame al proyecto

Se tendrá el formulario con diseño en modo grafico mostrado en la IMAGEN N° 03-0022, el tamaño se puede varias con el mouse, así mismo agregar los controles correspondientes de la paleta de controles:

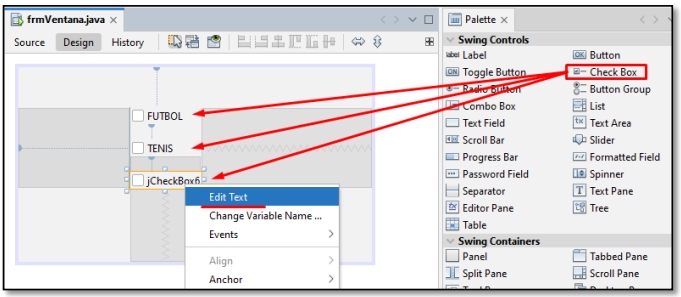


IMAGEN N° 03-0022: JFrame con diseño en modo gráfico y tres controles Check Box

Para cambiar algunas propiedades del formulario y asignar valores como el caso del título, con clic derecho seleccionar propiedades, tal como se muestra en la IMAGEN N° 03-0023, así mismo para cambiar de valores y nombre a los controles clic derecho en dichos controles y se mostrara el menú desplegable de la IMAGEN N° 03-0022(“***Edit Text”*** para cambiar etiqueta, “***Change Variable Name”*** para cambiar nombre de la variable: jCheckBox1->chkFutbol, jCheckBox2->chkTenis, jCheckBox3->chkNatacion)

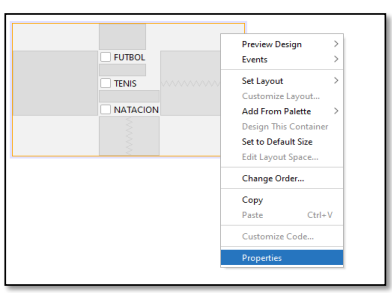


IMAGEN N° 03-0023: Menú desplegable que se muestra al hacer clic en algún control, en este caso el formulario.

En la IMAGEN N° 03-0024 se muestra las propiedades del formulario, cambiaremos el título "**DEPORTES QUE PRACTICA**". Al correr el programa tendremos el formulario de la IMAGEN N° 03-0019.

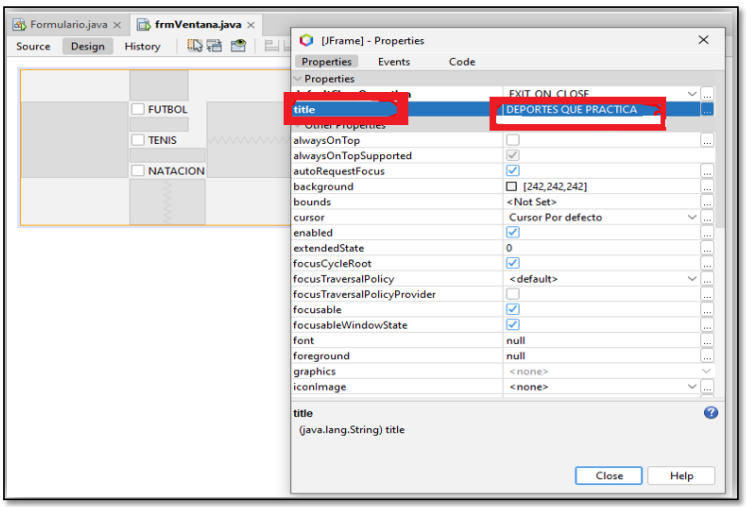


IMAGEN N° 03-0024: Propiedades del formulario, en este caso se cambia el title

Para agregar la lógica de que, al hacer click en cada check se visualice en el título del formulario los deportes seleccionados, se agrega los eventos mouseClicked, a cada control check, tal como se muestra en la IMAGEN N° 03-0025.

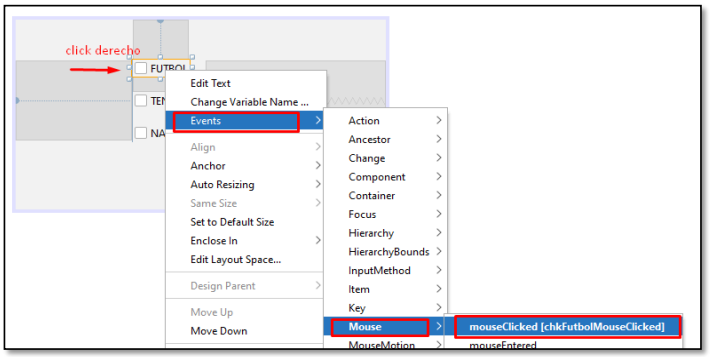


IMAGEN N° 03-0025: Menú desplegable para agregar el evento mouseClicked al check

Se agregara el método ***chkFutbolMouseClicked(ava.awt.event.MouseEvent evt)*** tal como se muestra en la IMAGEN N° 03-0026, en el cual se programara la lógica, y será lo mismo para los tres controles Check.

```
private void chkFutbolMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
}
```

IMAGEN N° 03-0026: Menú desplegable para agregar el evento mouseClicked al check

Código del método ***chkFutbolMouseClicked():***

```
private void chkFutbolMouseClicked(java.awt.event.MouseEvent evt) {
    String titulo="";
    if (chkFutbol.isSelected()==true) {
       titulo=titulo+"FUTBOL-";
    }
    if (chkTenis.isSelected()==true) {
       titulo=titulo+"TENIS-";
    }
    if (chkNatacion.isSelected()==true) {
```

```
            titulo=titulo+"NATACION-";
        }
        setTitle(titulo);
    }
```

Código del método ***chkTenisMouseClicked():***

```
    private void chkTenisMouseClicked(java.awt.event.MouseEvent evt) {
        String titulo="";
        if (chkFutbol.isSelected()==true) {
            titulo=titulo+"FUTBOL-";
        }
        if (chkTenis.isSelected()==true) {
            titulo=titulo+"TENIS-";
        }
        if (chkNatacion.isSelected()==true) {
            titulo=titulo+"NATACION-";
        }
        setTitle(titulo);
    }
```

Código del método ***chkNatacionMouseClicked():***

```
    private void chkTenisMouseClicked(java.awt.event.MouseEvent evt) {
        String titulo="";
        if (chkFutbol.isSelected()==true) {
            titulo=titulo+"FUTBOL-";
        }
        if (chkTenis.isSelected()==true) {
            titulo=titulo+"TENIS-";
        }
```

```
        if (chkNatacion.isSelected()==true) {
            titulo=titulo+"NATACION-";
        }
        setTitle(titulo);
    }
```

Para instanciar y visualizar el formulario en la clase principal en el método ***main***() se agrega el siguiente código:

```
package formulario;
public class Formulario {
    public static void main(String[] args) {
        frmVentana v=new frmVentana();
        v.setVisible(true);
        v.setLocationRelativeTo(null);
    }
```

En la IMAGEN N° 03-0027 se muestra la ejecución del programa, utilizando tres objetos de la clase JCheckBox

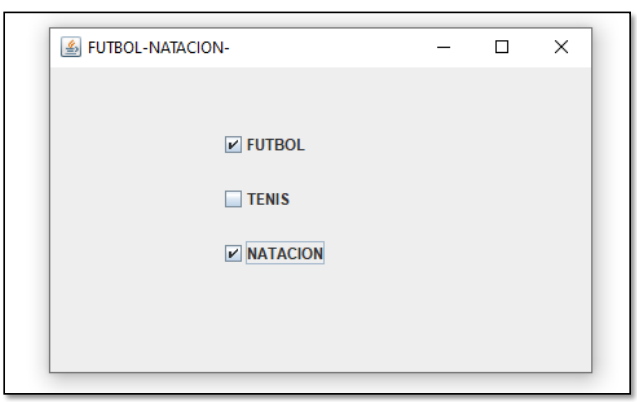


IMAGEN N° 03-0027: Formulario en ejecución con tres controles Check

EJEMPLO N°03-008

Diseñar el formulario que se muestra en la IMAGEN N° 03-0028. Cuando se clickea en el JCheckBox se debe activar el botón “Grabar y Salir”.

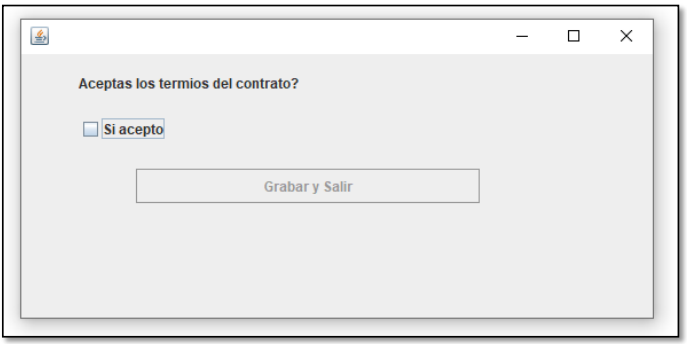


IMAGEN N° 03-0028: Formulario con un control JLabel, JCheckBox y JButton

DESARROLLO

Creando todo el diseño del formulario y los controles con código:

```
package javaapplication10;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
public class Main extends JFrame implements ActionListener,
ChangeListener{
   private JLabel label1;
   private JCheckBox check1;
   private JButton boton1;
   public Main() {
      setLayout(null);
      label1=new JLabel("Aceptas los términos del contrato?");
      label1.setBounds(50,10,400,30);
      add(label1);
```

```
    check1=new JCheckBox("Si acepto");
    check1.setBounds(50,50,100,30);
    check1.addChangeListener(this);
    add(check1);
    boton1=new JButton("Grabar y Salir");
    boton1.setBounds(100,100,300,30);
    add(boton1);
    boton1.addActionListener(this);
    boton1.setEnabled(false);
}

public void stateChanged(ChangeEvent e) {
    if (check1.isSelected()==true) {
        boton1.setEnabled(true);
    } else {
        boton1.setEnabled(false);
    }
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource()==boton1) {
        System.exit(0);
    }
}

public static void main(String[] ar) {
    Main formulario1=new Main();
    formulario1.setBounds(300,300,650,320);
    formulario1.setVisible(true);
```

```
    }
}
```

En este programa se ha implementado el método:

stateChanged(ChangeEvent e) , para que al hacer clic en el control JCheckBox, se active y desactive el botón grabar

actionPerformed(ActionEvent e), para que al hacer clic se salga del programa

Al ejecutar el programa se verá, así como en la IMAGEN N° 03-0029

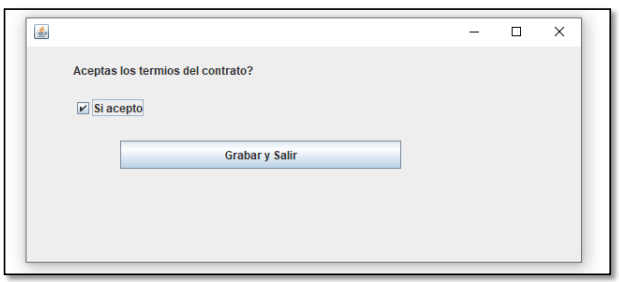


IMAGEN N° 03-0029: Formulario con un control JLabel, JCheckBox y JButton en ejecución

Como se tiene que implementar dos interfaces se enumera después de la palabra **implements** separadas por coma:

```
public class Main extends JFrame implements ActionListener,
ChangeListener{
```

Se define tres objetos:

```
private JLabel label1;
private JCheckBox check1;
private JButton boton1;
```

En el constructor se crea los objetos de tipo: JLabel, JCheckBox, JButton:

```
public Main() {
        setLayout(null);
        label1=new JLabel("Aceptas los términos del contrato?");
        label1.setBounds(50,10,400,30);
        add(label1);
        check1=new JCheckBox("Si acepto");
        check1.setBounds(50,50,100,30);
        check1.addChangeListener(this);
        add(check1);
        boton1=new JButton("Grabar y Salir");
        boton1.setBounds(100,100,300,30);
        add(boton1);
        boton1.addActionListener(this);
        boton1.setEnabled(false);
    }
```

Cuando se cambia el estado del control JCheckBox se ejecuta el método stateChanged donde se verifica si está seleccionado procediendo a activar el botón en caso negativo se desactiva:

```
public void stateChanged(ChangeEvent e) {
        if (check1.isSelected()==true) {
            boton1.setEnabled(true);
        } else {
            boton1.setEnabled(false);
        }
    }
```

El método actionPerformed se ejecuta cuando se presiona el objeto de tipo JButton (debe estar activo para poder presionarlo):

```
public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            System.exit(0);
```

```
    }
}
```

Diseño en modo gráfico:

En Netbeans agregar el JFrame, los controles necesarios según se requiere y hacer los cambios en los valores y propiedades, tal como se muestra en la IMAGEN N° 03-0030:

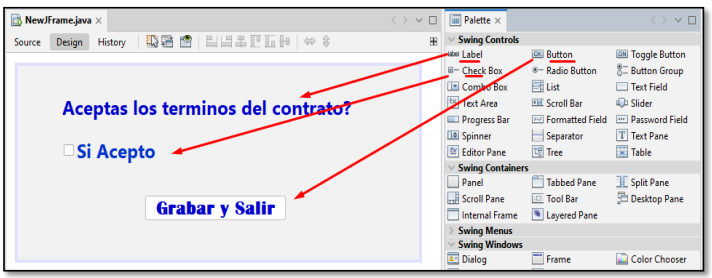


IMAGEN N° 03-0030: Formulario con un control JLabel, JCheckBox y JButton

Para agregar lógica, agregamos el evento **mouseClicked()** al control Check y el evento **actionPerformed**(ActionEvent e) al control JButton, tal como se muestra en la IMAGEN N° 03-0031, en el cual se programara la lógica.

```
private void chkAceptarMouseClicked(java.awt.event.MouseEvent evt) {

}

private void btnGrabarActionPerformed(java.awt.event.ActionEvent evt) {

}
```

IMAGEN N° 03-0031: Metodos MouseClicked() del control Check y ActionPerformed() del control JButton

Código del método ***chkAceptarMouseClicked():***

```
private void chkAceptarMouseClicked(java.awt.event.MouseEvent evt) {
```

```
if (chkAceptar.isSelected()==true) {
        btnGrabar.setEnabled(true);
    } else {
        btnGrabar.setEnabled(false);}
}
```

Código del método ***btnGrabarActionPerformed():***

```
private void btnGrabarActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
  }
```

Para instanciar y visualizar el formulario, en la clase principal en el método ***main***() se agrega el siguiente código:

```
package formulario;
public class Formulario {
  public static void main(String[] args) {

    NewJFrame v=new NewJFrame();
    v.setVisible(true);
    v.setLocationRelativeTo(null);
  }
```

En la IMAGEN N° 03-0032 se muestra la ejecución del programa.

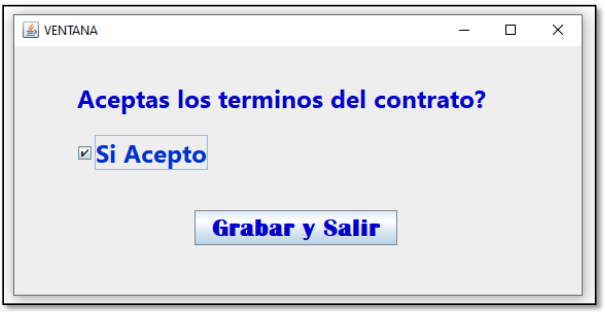


IMAGEN N° 03-0032: Formulario en ejecución, al hacer clic en Si Acepto se habilita el botón

3.4.4 Clase JTextArea

Esta clase es parecida a la clase JTextField, con la diferencia de que en este control se permite ingresar datos con múltiples filas.

Para crear un objeto *JTextArea* se pueden emplear varios constructores:

```
JTextArea texto = new JTextArea(); //sin parametros
JTextArea texto = new JTextArea(100,50); //con filas y columnas
JTextArea texto = new JTextArea( "Texto por defecto" ); //con texto por defecto
```

Para que JTextArea tenga barras de deslizamiento (scroll), debe estar envuelta en un objeto JScrollPane. El primer parámetro del JScrollPane es el objeto que se va a contener(JTextArea), y los otros dos parámetros se relacionan con las barras de desplazamiento:

```
JScrollPane barraXY = new JScrollPane(texto,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
```

El componente JScrollPane es el que se debe incluir en el panel contenedor en lugar de la JTextArea en sí, ya que de lo contrario, las barras de desplazamiento no funcionarán correctamente.

EJEMPLO N°03-009

Diseñar el formulario de la IMAGEN N° 03-0033, con los controles: JTextField, JTextArea y JButton, al hacer clic en el botón “Agregar”, agregara el contenido del JTextField al JTextArea y se habilitara el botón “Grabar y Salir”.

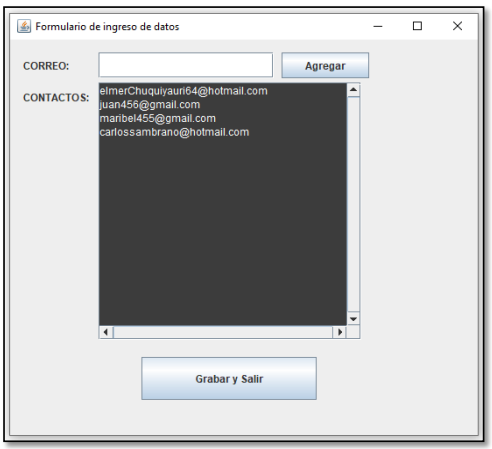


IMAGEN N° 03-0033: Formulario con un control JTextField y otro control JText Area

DESARROLLO

Creando todo el diseño del formulario y los controles con código, las barras de desplazamiento horizontal y vertical se agregarán con JScrollPane.

```
package javaapplication11;
import javax.swing.*;
import java.awt.Color*;
import java.awt.event.*;

public class Main extends JFrame implements ActionListener
{
  private JLabel lb1,lb2;
  private JTextField txt1;
```

```
private JTextArea txtarea1;
private JButton boton1;
private JButton boton2;


 public Main()
 {
        setLayout(null);
  lb1=new JLabel("CORREO:");
  lb1.setBounds(15,15,70,30);
  add(lb1);//se adiciona el control al contenedor


  lb2=new JLabel("CONTACTOS: ");
  lb2.setBounds(15,50,100,35);
  add(lb2);//se adiciona el control al contenedor


  txt1=new JTextField();
  txt1.setBounds(101,15,200,30);
  add(txt1);//se adiciona el control al contenedor


  //-----------------------------------------------
  //control textoArea
  txtarea1=new JTextArea();
  Color colorFondo = new Color(60,60,60);
  txtarea1.setBackground(colorFondo);
  Color colorLetra=new Color(255,255,255);
  txtarea1.setForeground(colorLetra);
  //-----------------------------------------------
   //txtarea1 = new JTextArea(4,40); //numero de filas y columnas
  //txtarea1.setBounds(101,50,300,300);
  //add(txtarea1);//se adiciona el control al contenedor
```

```
        //Barra horizontal y vertical para el textoArea
        JScrollPane barraXY = new
JScrollPane(txtarea1,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        barraXY.setBounds(101,50,300,300);
        add(barraXY);
        //txtarea1.add(barraXY);
       //boton agregar
        boton1=new JButton("Agregar");
        boton1.setBounds(310,15,100,30);
        add(boton1);
        boton1.addActionListener(this);

        boton2=new JButton("Grabar y Salir");
        boton2.setBounds(150,370,200,50);
        add(boton2);
        boton2.addActionListener(this);
        boton2.setEnabled(false);

        setTitle("Formulario de ingreso de datos");
}

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton2) {
            System.exit(0);
        }
        if (e.getSource()==boton1) {
            txtarea1.append(txt1.getText()+" \n");
            txt1.setText("");//se limpia el txt1
```

```
            txt1.requestFocus();//se coloca el enfoque en txt1
            if(txtarea1.getText().length()>0)
               boton2.setEnabled(true);}
      }
      public static void main(String[] ar)
       {
        Main m=new Main();
        m.setBounds(300,100,550,500);
        m.setVisible(true);
       }
   }
```

Diseño en modo gráfico:

En Netbeans agregar el JFrame, los controles necesarios según se requiere y hacer los cambios en los valores y propiedades, tal como se muestra en la IMAGEN N° 03-0034:

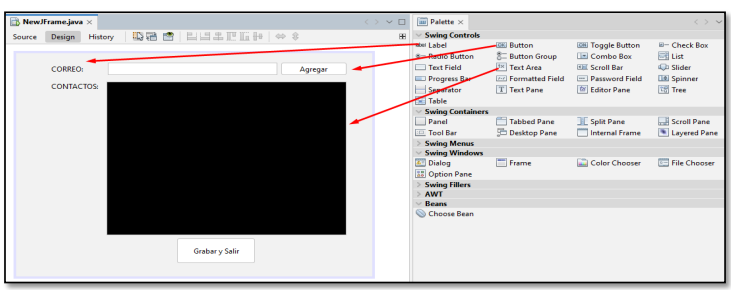


IMAGEN N° 03-0034: Diseño en modo gráfico con el control JTextArea

Para agregar lógica, agregamos el evento **actionPerformed**(ActionEvent e) al botón Agregar, tal como se muestra en la IMAGEN N° 03-0035, en el cual se obtendrá el valor del cuadro de texto y se agregara al textArea.

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
    txtaContactos.append(txtCorreo.getText()+"\n");
    txtCorreo.setText( string:"");
    txtCorreo.requestFocus();
}
```

IMAGEN N° 03-0035: Metodo ActionPerformed() del botón Agregar

Código del método ***jButton2ActionPerformed():***

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    txtaContactos.append(txtCorreo.getText()+"\n");
    txtCorreo.setText("");
    txtCorreo.requestFocus();
}
```

3.4.5 Clase JComboBox

Es un componente que se asemeja mucho a un elemento HTML llamado 'select'. El JComboBox nos permite crear una lista desplegable en la que podemos agregar elementos y manipularla completamente. Es muy sencillo de utilizar, e incluso podemos cargar datos desde una base de datos para crear una lista más profesional. Este objeto es una clase que nos proporciona una lista desplegable. Para capturar el item seleccionado se tienen que implementar la interface ***ItemListener*** que contienen un método llamado ***itemStateChanged()***.

3.4.5.1 métodos más usados

Los métodos más utilizados:

void : addItem(Object item).

Agrega un elemento al combo

Object: getSelectedItem().

Retorna el valor seleccionado.

int: getSelectedIndex().

Obtiene el índice seleccionado.

void: setEditable(boolean tipo).

Determina si sólo mostrara los valores (false) o si se pueden escribir nuevos valores (true).

EJEMPLO N°03-010

El formulario mostrado en la IMAGEN N° 03-0036, contienen los meces del año, en este ejemplo se utiliza un control JComboBox

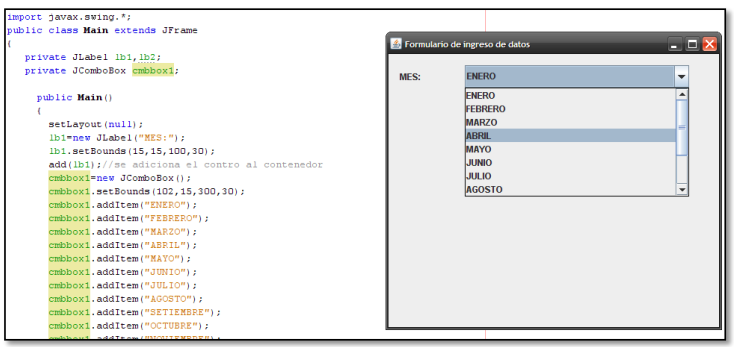


IMAGEN N° 03-0036: el control JComboBox que despliega los meces del año

Para este ejemplos se adiciona al formulario un objeto de la clase JCombobox, y luego se adiciona los item con los meces del año.

CODIGO

```
package javaapplication11;
import javax.swing.*;
public class Main extends JFrame
{
  private JLabel lb1,lb2;
  private JComboBox cmbbox1;

   public Main()
   {
```

```
    setLayout(null);
    lb1=new JLabel("MES:");
    lb1.setBounds(15,15,100,30);
    add(lb1);//se adiciona el contro al contenedor
    cmbbox1=new JComboBox();
    cmbbox1.setBounds(102,15,300,30);
    cmbbox1.addItem("ENERO");
    cmbbox1.addItem("FEBRERO");
    cmbbox1.addItem("MARZO");
    cmbbox1.addItem("ABRIL");
    cmbbox1.addItem("MAYO");
    cmbbox1.addItem("JUNIO");
    cmbbox1.addItem("JULIO");
    cmbbox1.addItem("AGOSTO");
    cmbbox1.addItem("SETIEMBRE");
    cmbbox1.addItem("OCTUBRE");
    cmbbox1.addItem("NOVIEMBRE");
    cmbbox1.addItem("DICIEMBRE");

    add(cmbbox1);
    setTitle("Formulario de ingreso de datos");
  }
  public static void main(String[] ar)
  {
    Main m=new Main();
    m.setBounds(10,10,450,400);
    m.setVisible(true);
  }
}
```

EJEMPLO N°03-011

El siguiente programa muestra una lista de correos en un JComboBox de contactos y cada vez que se selecciona un correo se adiciona en otro JComboBox.

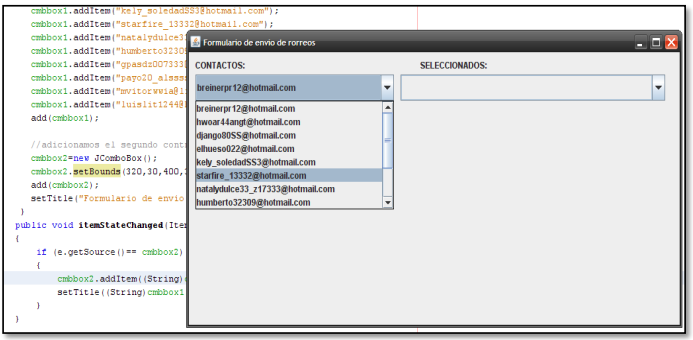


IMAGEN N° 03-0037: Formulario con dos controles JComboBox

CODIGO

```
package javaapplication11;
import javax.swing.*;
import java.awt.event.*;

public class Main extends JFrame implements ItemListener
{
  private JLabel lb1,lb2;
  private JComboBox cmbbox1;
  private JComboBox cmbbox2;

    public Main()
    {
      setLayout(null);
```

```
lb1=new JLabel("CONTACTOS:");
lb1.setBounds(10,10,100,15);
add(lb1);//se adiciona el contro al contenedor

lb2=new JLabel("SELECCIONADOS:");
lb2.setBounds(350,10,520,15);
add(lb2);//se adiciona el contro al contenedor

cmbbox1=new JComboBox();
cmbbox1.setBounds(10,30,300,35);
cmbbox1.addItem("breinerpr12@hotmail.com ");
cmbbox1.addItem("hwoar44angt@hotmail.com ");
cmbbox1.addItem("django80SS@hotmail.com");
cmbbox1.addItem("elhueso022@hotmail.com");
cmbbox1.addItem("kely_soledadSS3@hotmail.com");
cmbbox1.addItem("starfire_13332@hotmail.com");
cmbbox1.addItem("natalydulce33_z17333@hotmail.com");
cmbbox1.addItem("humberto32309@hotmail.com");
cmbbox1.addItem("gpasdz007333@hotmail.com");
cmbbox1.addItem("payo20_alssss@hotmail.com");
cmbbox1.addItem("mvitorwwia@live.com");
cmbbox1.addItem("luislit1244@hotmail.com");
add(cmbbox1);

//se adicional el segundo control JComboBox
cmbbox2=new JComboBox();
cmbbox2.setBounds(320,30,400,35);
add(cmbbox2);
setTitle("Formulario de envio de rorreos");
```

```
    }
public void itemStateChanged(ItemEvent e)
  {
    if (e.getSource()== cmbbox1)
    {
      //se verifica para evitar que se cargue dos veces
       if (e.getStateChange() == ItemEvent.SELECTED){
            JOptionPane.showMessageDialog(null, "Se agrego");
            cmbbox2.addItem((String)cmbbox1.getSelectedItem());
       }
    }
  }

  public static void main(String[] ar)
   {
     Main m=new Main();
     m.setBounds(10,10,750,400);
     m.setVisible(true);
  }
}
```

3.4.5 Clase JRadioButon

La clase JRadioButton se emplea para crear opciones de selección que permite elegir una de entre varias disponibles. Para ello se le agrupa para que actúen en conjunto, esto para que cuando se selecciona uno, automáticamente se debe deseleccionar los otros. En la IMAGEN N° 03-0038 se puede ver varios controles Radio Button que se agregaron al formulario.

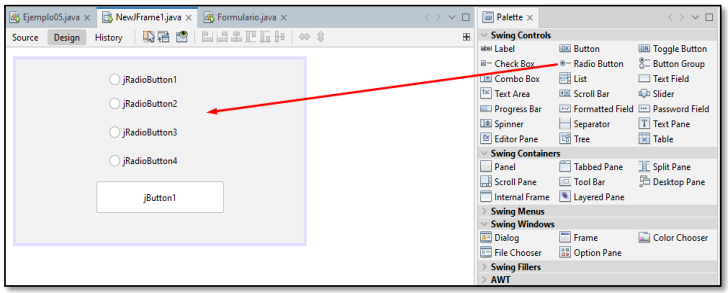


IMAGEN N° 03-0038: Formulario con controles Radio Button

EJEMPLO N°03-012

El siguiente programa permite seleccionar una respuesta a una pregunta de una encuesta electoral, para lo cual se muestra un conjunto de partidos.

CODIGO

```
package javaapplication11;
import javax.swing.*;
public class Main extends JFrame
{
  private JLabel lb1;
  private JRadioButton rb1,rb2,rb3,rb4;
  private ButtonGroup bg;

    public Main()
    {
      setLayout(null);
      bg=new ButtonGroup();
      lb1=new JLabel("¿Por que partido votaria en las proximas
elecciones? ");
      lb1.setBounds(10,5,400,15);
```

```
  add(lb1);//se adiciona el contro al contenedor

  rb1=new JRadioButton("El partido de las galaxias");
  rb1.setBounds(15,20,300,25);
  add(rb1);
  bg.add(rb1);
  rb2=new JRadioButton("Corre Perú");
  rb2.setBounds(15,45,300,50);
  add(rb2);
  bg.add(rb2);

  rb3=new JRadioButton("Cambia Perú");
  rb3.setBounds(15,70,300,75);
  add(rb3);
  bg.add(rb3);

  rb4=new JRadioButton("Te Amo Perú");
  rb4.setBounds(15,100,300,105);
  add(rb4);
  bg.add(rb4);
  setTitle("Encuesta");
 }

public static void main(String[] ar)
 {
  Main m=new Main();
  m.setBounds(10,10,400,200);
  m.setVisible(true);
}
```

```
}
```

Al ejecutar el programa se vería tal como se muestra en la IMAGEN N° 03-0039

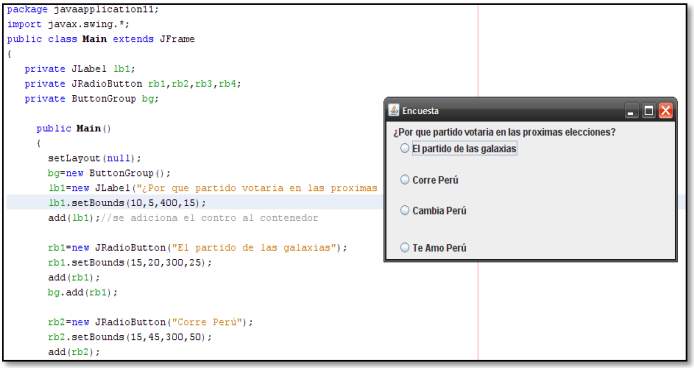


IMAGEN N° 03-0039: Formulario con tres controles Radio Button

EJEMPLO N°03-012

El siguiente programa muestra 3 objetos de la clase JRadioButton, que permite configurar el ancho y alto del JFrame:

```
import javax.swing.*;
import javax.swing.event.*;
public class Main extends JFrame implements ChangeListener
{
    private JRadioButton rb1,rb2,rb3;
    private ButtonGroup bg;
    public Main()
    {
      setLayout(null);
      bg=new ButtonGroup();
      rb1=new JRadioButton("640x480");
      rb1.setBounds(15,20,300,25);
```

```
        add(rb1);
        bg.add(rb1);//se adiciona al grupo
        rb1.addChangeListener(this);
        rb2=new JRadioButton("800x600");
        rb2.setBounds(15,45,300,50);
        add(rb2);
        bg.add(rb2);//se adiciona al grupo
        rb2.addChangeListener(this);
        rb3=new JRadioButton("1024x768");
        rb3.setBounds(15,70,300,75);
        add(rb3);

        bg.add(rb3);//se adiciona al grupo
        rb3.addChangeListener(this);
        setTitle("Configuración de Ventana");
      }
public void stateChanged(ChangeEvent e)
      {
        if (rb1.isSelected()) {
            setSize(640,480);
        }
        if (rb2.isSelected()) {
            setSize(800,600);
        }
        if (rb3.isSelected()) {
            setSize(1024,768);
        }
    }
    public static void main(String[] ar)
```

```
    {
      Main m=new Main();
      m.setBounds(10,10,400,200);
      m.setVisible(true);
    }
}
```

Al ejecutar el programa se vería tal como se muestra en la IMAGEN N° 03-0040

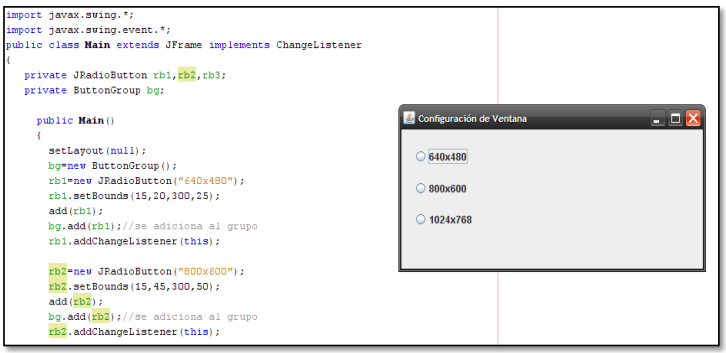


IMAGEN N° 03-0040: Formulario con tres controles Radio Button en ejecución

EJEMPLO N°03-013

Realizar una calculadora que ejecute las 4 operaciones básicas, ver IMAGEN N° 03-0041:

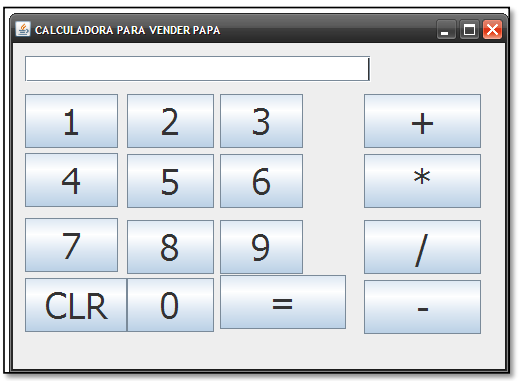


IMAGEN N° 03-0041: Calculadora que realiza las cuatro operaciones básicas

DESARROLLO

Para realizar dicho programa agregamos una caja de texto(jTextField) y 16 botones(JButton), todo dentro de un Frame, mostrado en la IMAGEN N° 03-0042

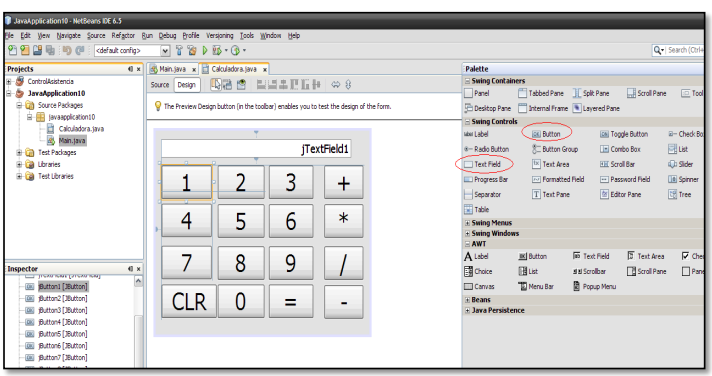


IMAGEN N° 03-0042: Diseño de la Calculadora que realiza las cuatro operaciones basicas

De cada botón que será como numero cambiamos su text por el número correspondiente, diseño mostrado en la IMAGEN N° 03-0043:

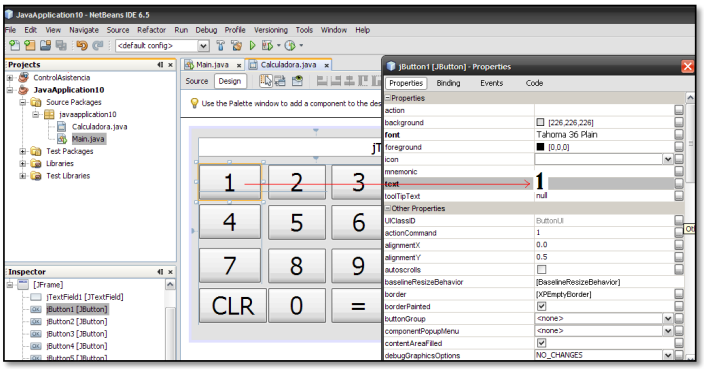


IMAGEN N° 03-0043: Cambio del text de cada boton, colocándolos los números

Para diseñar la calculadora mostrado en la IMAGEN N° 03-0041, en NetBeans en modo grafico se agrega todos los controles de la paleta de controles, ósea arrastramos con el mouse todos los controles necesarios al contenedor y se cambia sus propiedades de cada uno de ellos de acuerdo a los requerimientos. Para que realice las cuatro operaciones básicas en el botón " = " se programa en el evento cuando el usuario realiza un clic, por ejemplo cuando se quiere hacer una suma.

```
if (e.getSource()==jButton16)//boton igual
 {
   operador2=jTextField1.getText();
   switch(operacion)
   {
      case 1://suma

            resultado=Double.parseDouble(operador1)+Double.parseD
            ouble(operador2);
         break;
    .
        .
```

.

El código completo de la calculadora se muestra a continuación.

```
package javaapplication10;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;

public class Calculadora extends JFrame  implements ActionListener
{
   private String cadena;
   private String operador1;
   private String operador2;
   private int operacion;
   private double resultado;
   private int op;

   public Calculadora()
    {
      super("CALCULADORA PARA VENDER PAPA");
      cadena="";
       initComponents();
       jTextField1.setText("");
            jTextField1.setText("");
      jButton1.addActionListener(this);
      jButton2.addActionListener(this);
      jButton3.addActionListener(this);
      jButton4.addActionListener(this);
      jButton5.addActionListener(this);
```

```
    jButton6.addActionListener(this);
    jButton7.addActionListener(this);
    jButton8.addActionListener(this);
    jButton9.addActionListener(this);
    jButton13.addActionListener(this);
    jButton14.addActionListener(this);
    jButton15.addActionListener(this);
    jButton16.addActionListener(this);

    jButton10.addActionListener(this);
    jButton11.addActionListener(this);
    jButton12.addActionListener(this);

    operacion=0;
    operador1="";
    operador2="";
    resultado=0;
    op=0;
  }

public void actionPerformed(ActionEvent e)
  {
    if (e.getSource()==jButton1)
     {
       if(op==1)
       {
          jTextField1.setText(" ");

          op=0;
```

```
        }
        cadena=jTextField1.getText()+"1";
        jTextField1.setText(cadena);

      }
     if (e.getSource()==jButton2)

{
       if(op==1)
       {
          jTextField1.setText(" ");
          op=0;
       }

  cadena=jTextField1.getText()+"2";

       jTextField1.setText(cadena);

      }

     if (e.getSource()==jButton3)
      {

       if(op==1)
       {
          jTextField1.setText(" ");
          op=0;
       }
       cadena=jTextField1.getText()+"3";
```

```
            jTextField1.setText(cadena);
        }
        if (e.getSource()==jButton4)
        {
            if(op==1)
            {
                jTextField1.setText(" ");
                op=0;
            }

cadena=jTextField1.getText()+"4";
            jTextField1.setText(cadena);
        }

        if (e.getSource()==jButton5)
        {
            if(op==1)
            {
                jTextField1.setText(" ");
                op=0;
            }
            cadena=jTextField1.getText()+"5";
            jTextField1.setText(cadena);
        }

        if (e.getSource()==jButton6)
        {
            if(op==1)
            {
```

```
        jTextField1.setText(" ");
        op=0;
    }

    cadena=jTextField1.getText()+"6";
    jTextField1.setText(cadena);
}

if (e.getSource()==jButton7)
{
    if(op==1)
    {
        jTextField1.setText(" ");
        op=0;
    }

    cadena=jTextField1.getText()+"7";
    jTextField1.setText(cadena);
}


if (e.getSource()==jButton8)
{
    if(op==1)
    {
        jTextField1.setText(" ");
        op=0;
    }
    cadena=jTextField1.getText()+"8";
```

```
            jTextField1.setText(cadena);
        }
    if (e.getSource()==jButton9)
        {
            if(op==1)
            {
                jTextField1.setText(" ");
                op=0;
            }
            cadena=jTextField1.getText()+"9";
            jTextField1.setText(cadena);
        }

        if (e.getSource()==jButton13)
        {
            if(op==1)
            {
                jTextField1.setText(" ");
                op=0;
            }
            cadena=jTextField1.getText()+"0";
            jTextField1.setText(cadena);
        }

        if (e.getSource()==jButton14)
        {
            cadena=" ";
            jTextField1.setText(cadena);
        }
```

```
        if (e.getSource()==jButton10)//boton suma
        {
          operador1=jTextField1.getText();
          operacion=1;  //suma
          cadena=" ";
          op=1;
        }
        if (e.getSource()==jButton11)//boton multiplicacion
        {
          operador1=jTextField1.getText();
          operacion=2;  //multiplicacion
          cadena=" ";
          op=1;
        }
    if (e.getSource()==jButton12)//boton divicion
        {
          operador1=jTextField1.getText();
          operacion=3;  //divicion
          cadena=" ";
          op=1;
        }
       if (e.getSource()==jButton15)//boton resta
        {
          operador1=jTextField1.getText();
          operacion=4;  //divicion
          cadena=" ";
          op=1;
        }
```

```
if (e.getSource()==jButton16)//boton igual
 {
   operador2=jTextField1.getText();
   switch(operacion)
   {
      case 1://suma

resultado=Double.parseDouble(operador1)+Double.parseDouble(operado
r2);
         break;
      case 2:

resultado=Double.parseDouble(operador1)*Double.parseDouble(operador
2);
         break;
      case 3:

resultado=Double.parseDouble(operador1)/Double.parseDouble(operador
2);
         break;
      case 4:
         resultado=Double.parseDouble(operador1)-
Double.parseDouble(operador2);
         break;
   }
   String Res;
   Res=resultado+" ";
   jTextField1.setText(Res);
   operador1="";
   operador2="";
   op=1;
```

```
    }
}
```

Programa en plena ejecución, visto en la IMAGEN N° 03-0044

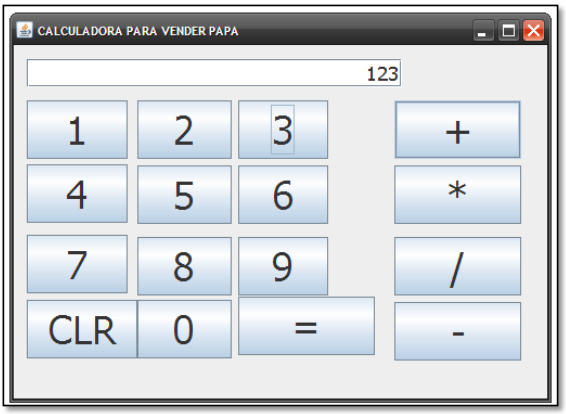


IMAGEN N° 03-0044: Calculadora que realiza las 4 operaciones básicas

3.4.6 Clase JTable

El JTable es un control de javax.swing que sirve para poder dibujar tablas, con sus respectivas filas y columnas, tal como se muestra en la IMAGEN N°003-0045.

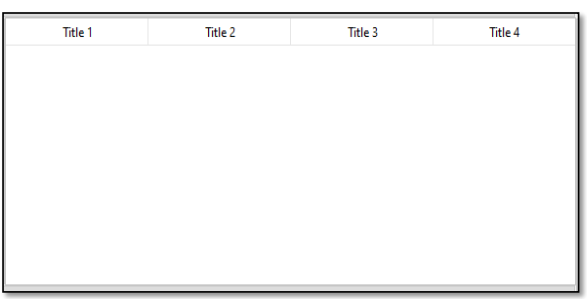


IMAGEN N° 03-0045: JTable

Se pueden cargar datos en tiempo de diseño y en tiempo de ejecución.
Para poder visualizar datos en la tabla, modificar valores, entre otras operaciones.
La tabla se relaciona con un objeto tipo **DefaultTableModel**, que es una clase que implementa TableModel y que contiene métodos necesarios para hacer diversas operaciones.

Pasos para trabajar con JTable y gestionar datos

- ✓ Se le asigna un nombre al control instanciado: JTable1 → **nuevoNombre**
- ✓ Se instancia el **DefaultTableModel:** DefaultTableModel ***modeloTabla***
- ✓ Se relaciona el control con el modelo**: uevoNombre.setModel(*modeloTabla*)**
- ✓ Se podría cambiar la cantidad de columnas y sus nombres:

```
String []columnaNombre={“nombreColumna1”, “nombreColumna2”,…};
String [][] dato={};
modeloTabla =new DefaultTableModel(dato,columnaNombre);
```

- ✓ Se podría cambiar las dimisiones de las columnas de la tabla con el siguiente código:

```
TableColumnModel columnModel = nuevoNombre.getColumnModel();
    columnModel.getColumn(0).setPreferredWidth(15);
    columnModel.getColumn(1).setPreferredWidth(60);
    columnModel.getColumn(2).setPreferredWidth(60);
    columnModel.getColumn(3).setPreferredWidth(30);
```

- ✓ Para agregar filas a las tablas utilizar el método:

```
modeloTabla.addRow(dato)
```

Donde dato es un arreglo con la misma cantidad de columnas que la tabla

- ✓ Para remover elementos se utiliza el método:

```
modeloTabla.removeRow(f);
```

Donde f es la fila seleccionado que se captura con:

```
int f= nuevoNombre.getSelectedRow();
```

EJEMPLO N°03-014

Realizar un programa para gestionar contactos (datos de personas: DNI, apellidos y nombres, correo, teléfono, lugar de residencia). El formulario principal podría tener el diseño presentado en la IMAGEN N° 03-0046.

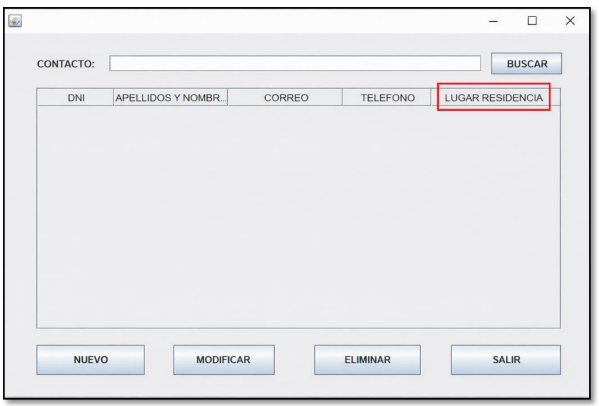


IMAGEN N° 03-0046: Diseño de la Ventana Principal del programa

Donde lugar de residencia tenga el siguiente formato:
REGION/PROVINCIA/DISTRITO/dirección

El formulario es el ingreso y modificación de datos de un contacto podría tener el diseño de la IMAGEN N° 03-0047:

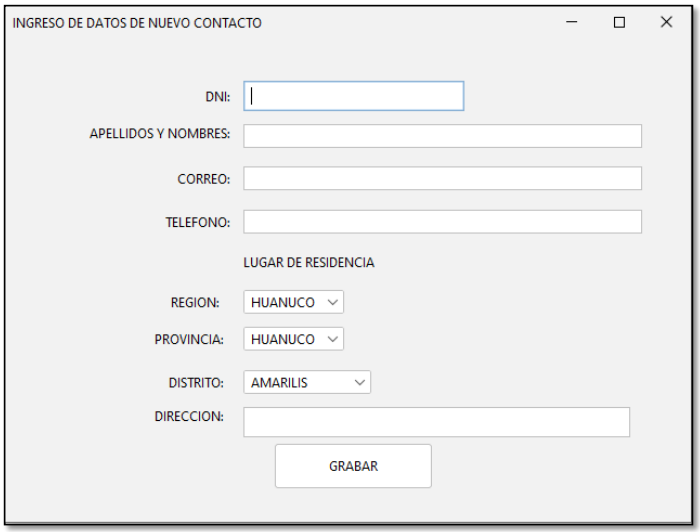


IMAGEN N° 03-0047: Diseño de la Ventana de ingreso de nuevos datos

DESARROLLO

Para desarrollar el programa para la ventana principal se agregará los controles necesarios, entre ellos el JTable, para la gestión de los datos se almacenará en archivos .txt(contactos.txt) para hacer persistente los datos. Así mismo las regiones, provincias y distritos estarán almacenado en archivos txt.

Para el manejo de los datos se crearán las siguientes clases: Contacto, Región, Provincia, Distrito, con la siguiente estructura:

La clase Region:

```
public class Region {
public int id_region;
public String nombre;
public Region(int i, String nombre1){
   id_region=i;
   nombre=nombre1;
```

```
    }
    public int getId() {
        return id_region;
    }
    public String getNombre() {
        return nombre;
    }
   // @Override
    @Override
    public String toString() {
        return nombre;
    }
}
```

La clase Provincia:

```
public class Provincia {
    int id_provincia;
    String nombre;
    Region r;
    public Provincia(int i,String nombre1,Region r1){
        id_provincia=i;
        nombre=nombre1;
        r=r1;
    }
    public int getIdProvincia() {
        return id_provincia;
    }
    public int getIdRegion() {
        return r.id_region;
```

```
    }
    public String getNombre() {
        return nombre;
    }

    // @Override
    @Override
    public String toString() {
        return nombre;
    }
}
```

La clase Distrito:

```
public class Distrito {
    int id_distrito;
    String nombre;
    Provincia p;
    public Distrito(int id_distrito, String nombre, Provincia p) {
        this.id_distrito = id_distrito;
        this.nombre = nombre;
        this.p = p;
    }

    public int getId_distrito() {
        return id_distrito;
    }
    public String getNombre() {
        return nombre;
    }
```

```
    public Provincia getP() {
        return p;
    }
    public void setId_distrito(int id_distrito) {
        this.id_distrito = id_distrito;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public void setP(Provincia p) {
        this.p = p;
    }
    @Override
    public String toString() {
        return "Distrito{" + '}';
    }
}
```

La clase Contacto:

```
public class Contacto {
    private String DNI;
    private String apellido_nombre;
    private String correo;
    private String telefono;
    private Distrito d;
    private String direccion;
    public Contacto(String DNI, String apellido_nombre, String correo,
String telefono, Distrito d, String direccion) {
        this.DNI = DNI;
```

```
        this.apellido_nombre = apellido_nombre;
        this.correo = correo;
        this.telefono = telefono;
        this.d = d;
        this.direccion = direccion;
    }

public String getDNI() {
        return DNI;
    }
    public String getApellido_nombre() {
        return apellido_nombre;
    }
    public String getCorreo() {
        return correo;
    }

    public String getTelefono() {
        return telefono;
    }
    public Distrito getD() {
        return d;
    }

    public String getDireccion() {
        return direccion;
    }
    public void setDNI(String DNI) {
        this.DNI = DNI;
```

```
    }
    public void setApellido_nombre(String apellido_nombre) {
        this.apellido_nombre = apellido_nombre;
    }

    public void setCorreo(String correo) {
        this.correo = correo;
    }
    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }
    public void setD(Distrito d) {
        this.d = d;
    }
    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    @Override
    public String toString() {
        return "Contacto{" + "apellido_nombre=" + apellido_nombre + '}';
    }
}
```

Se diseña el formulario de la IMAGEN N° 03-0046, para lo cual se agrega los controles necesarios, tal como se muestra en la IMAGEN N° 03-0048

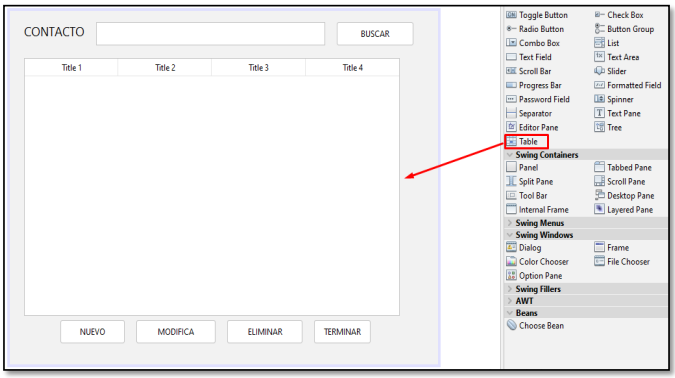


IMAGEN N° 03-0048: Diseño de la Ventana principal

Luego se diseña el formulario de la IMAGEN N° 03-0047, tal como se muestra en la IMAGEN N° 03-0049.

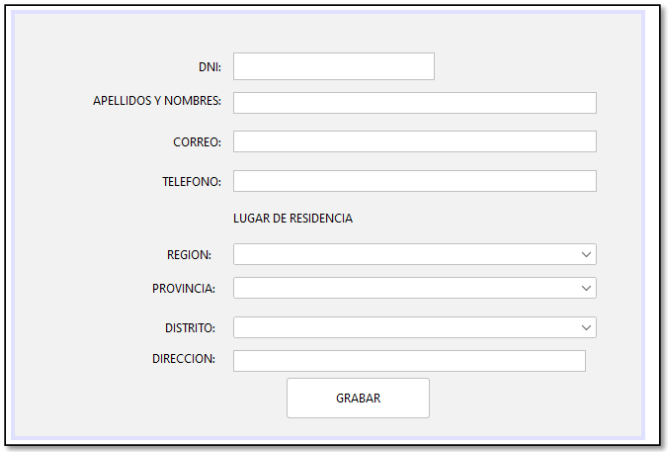


IMAGEN N° 03-0049: Diseño de la Ventana ingreso de nuevo contacto

CAPÍTULO 4. DISEÑO DE BASE DE DATOS

4.1 BASE DE DATOS

En el mundo de la programación, las bases de datos constituyen uno de los pilares más importantes para el desarrollo de aplicaciones modernas. Son el medio a través del cual los programas almacenan información de manera persistente, es decir, que los datos permanecen disponibles incluso después de que la aplicación se cierre. En el contexto de Java, la conexión con bases de datos permite construir aplicaciones empresariales, sistemas web y móviles con almacenamiento estructurado y confiable.

Una base de datos puede definirse como un conjunto de datos organizados de manera que se puedan manipular y consultar fácilmente. Para gestionar estos datos, se utiliza un Sistema de Gestión de Bases de Datos (SGBD), que actúa como intermediario entre el usuario o la aplicación y los datos almacenados. Entre los SGBD más conocidos se encuentran MySQL, PostgreSQL, Oracle, SQL Server y SQLite, todos compatibles con Java mediante tecnologías como JDBC (Java Database Connectivity) o ORM (Object-Relational Mapping).

El uso de bases de datos no se limita al almacenamiento; también garantiza la integridad, consistencia y seguridad de la información. A través de reglas, restricciones y relaciones, se evita la redundancia de datos y se facilita la escalabilidad de los sistemas. Esto permite que una aplicación Java pueda crecer sin perder control sobre su estructura interna.

El aprendizaje de bases de datos debe comenzar con una comprensión conceptual del problema que se desea resolver. Antes de escribir una sola línea de código SQL o de conectar Java a un servidor, se debe diseñar correctamente la base de datos. Este proceso de diseño se divide en tres etapas: **diseño conceptual, diseño lógico y diseño físico**, que se explicarán a continuación.

4.2 DISEÑO DE BASE DE DATOS

El diseño de base de datos es un proceso sistemático que tiene como objetivo estructurar la información de manera lógica y eficiente para satisfacer las necesidades de una organización o aplicación. En el contexto de la programación con Java, un diseño adecuado es esencial para garantizar que la aplicación pueda acceder, almacenar y manipular los datos correctamente, sin redundancia ni inconsistencias. Una base de datos mal diseñada puede generar errores, pérdida de rendimiento e incluso corrupción de la información.

El diseño de una base de datos se basa en el principio de que los datos deben representar fielmente la realidad del entorno que se quiere modelar. Esto implica analizar el problema, identificar las entidades relevantes, establecer las relaciones entre ellas y definir cómo se almacenarán y accederán los datos. Todo este proceso se conoce como el proceso de diseño de base de datos, el cual se desarrolla en varias etapas bien definidas.

Proceso de diseño de base de datos

El proceso de diseño de base de datos generalmente comprende seis etapas fundamentales, que permiten pasar desde una comprensión abstracta del problema hasta una implementación física lista para usarse en un sistema gestor (como MySQL, PostgreSQL o SQL Server). A continuación, se describen cada una de ellas:

- ✓ **Recolección y análisis de requisitos:**
  En esta primera etapa, se recopila toda la información necesaria sobre el sistema que se desea desarrollar. Esto incluye entrevistas con usuarios, revisión de procesos existentes, identificación de los datos que se manejarán y definición de los informes o consultas que se necesitarán. Por ejemplo, si se diseña un sistema de ventas, se deben identificar los clientes, productos, facturas, pagos, etc. Esta etapa tiene un carácter analítico, y su objetivo es comprender el dominio del problema.

- ✓ **Diseño conceptual:**
  En esta fase se construye una representación abstracta de los datos, independiente del SGBD. El resultado es el modelo entidad-relación (E-R), que muestra las entidades principales, sus atributos y las relaciones entre ellas. Este modelo sirve como base para discutir con los usuarios y validar que la estructura planteada refleje correctamente la realidad del negocio.

- ✓ **Diseño lógico:**
  Una vez validado el modelo conceptual, se transforma en un modelo relacional, compuesto por tablas, columnas, claves primarias y claves foráneas. Aquí se determinan los tipos de relaciones (uno a uno, uno a muchos o muchos a muchos) y se aplican las reglas de normalización para evitar redundancias y asegurar la consistencia de los datos.

- ✓ **Diseño físico:**
  En esta etapa se define cómo se almacenarán realmente los datos dentro del SGBD. Se seleccionan los tipos de datos, los índices, las configuraciones de rendimiento, las copias de seguridad y los esquemas de particionamiento. Este diseño depende directamente del gestor de base de datos que se utilizará junto con Java, pues cada uno tiene características específicas de optimización.

- ✓ **Implementación:**
  Una vez completado el diseño físico, se crea la base de datos en el sistema gestor mediante el lenguaje SQL. En el contexto de Java, esta fase implica preparar la estructura de tablas que luego se conectarán mediante JDBC, JPA o Hibernate. También se crean las restricciones, vistas y procedimientos almacenados necesarios para el funcionamiento de la aplicación.

- ✓ **Pruebas y mantenimiento:**
  Finalmente, se realizan pruebas para verificar que la base de datos funciona según lo esperado. Se evalúa el rendimiento, la integridad y la seguridad. Durante la vida útil del sistema, la base de datos puede requerir

mantenimiento, como la adición de nuevas tablas o la modificación de estructuras existentes para adaptarse a cambios en los requisitos del usuario.

Cada una de estas etapas se retroalimenta de las demás. Por ejemplo, durante la implementación pueden detectarse errores de diseño lógico que requieran volver al modelo conceptual. Este enfoque iterativo garantiza una mayor calidad y estabilidad del sistema.

El diseño de base de datos también implica la normalización de datos, que consiste en aplicar un conjunto de reglas para estructurar las tablas de manera que se elimine la redundancia y se garantice la coherencia. Existen varias formas normales (1FN, 2FN, 3FN, BCNF, entre otras), y aplicarlas correctamente permite que las actualizaciones y consultas se realicen sin errores. En la práctica, los diseñadores deben equilibrar la normalización con el rendimiento, ya que un exceso de división en tablas puede ralentizar las operaciones.

En el caso de las aplicaciones Java, un diseño bien planificado permite reflejar las tablas de la base de datos como clases del lenguaje. Por ejemplo, una tabla “Cliente” se puede representar como una clase Cliente, con atributos como idCliente, nombre, direccion y telefono. Este enfoque se conoce como mapeo objeto-relacional, y se implementa fácilmente con frameworks como Hibernate o Jakarta Persistence API (JPA).

Por último, el diseño de una base de datos debe tener en cuenta aspectos de seguridad, integridad y rendimiento. Las reglas de acceso, las copias de respaldo y los mecanismos de autenticación son tan importantes como el modelo de datos mismo. Una base de datos puede estar correctamente diseñada en lo lógico, pero si no se protege adecuadamente, puede ser vulnerable a ataques o pérdida de información.

4.3 DISEÑO CONCEPTUAL BASE DE DATOS

El diseño conceptual de base de datos es la primera fase formal del proceso de diseño y representa el punto de partida para la construcción de un sistema de información sólido. Su objetivo principal es modelar la realidad del negocio o del problema que se desea resolver, utilizando una representación abstracta de los datos, sin depender todavía del tipo de gestor de base de datos (SGBD) ni del lenguaje SQL. Este modelo conceptual permite comprender qué datos se deben almacenar y cómo se relacionan entre sí.

En esta fase, el diseñador no se preocupa aún por aspectos técnicos, sino por la estructura lógica del conocimiento. Por ejemplo, si se va a desarrollar un sistema de ventas, el diseñador identifica entidades como Cliente, Producto, Factura y Vendedor, y define cómo se relacionan entre ellas. Estas entidades se representan gráficamente mediante un modelo entidad-relación (E-R), propuesto por Peter Chen en 1976, que se ha convertido en el estándar más utilizado en el diseño conceptual de bases de datos.

El modelo entidad-relación describe los datos en términos de tres componentes básicos: entidades, atributos y relaciones.

Entidades

Las entidades representan objetos del mundo real o conceptos relevantes para el sistema y su representación en el esquema conceptual es con un rectangulo. Por ejemplo, en un sistema universitario, las entidades podrían ser Estudiante, Docente y Curso, personas tal como se puede representar el la IMAGEN N°04-0001.

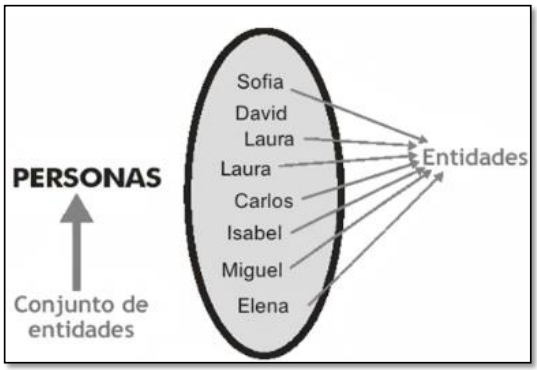


IMAGEN N°04-0001: Conjunto de personas

Su representación de la entidad persona como entidad en el esquema conceptual es con un rectángulo, tal como se muestra en la IMAGEN N°04-0002

IMAGEN N°04-0002: Entidad persona representado con un rectángulo

Tipos de Entidades (según su existencia):

Entidades Regulares (Fuertes):

Existencia: Independiente. Tienen su propia clave primaria (identificador único).

Representación: Un rectángulo simple, tal como se muestra en la IMAGEN N°04-0003.

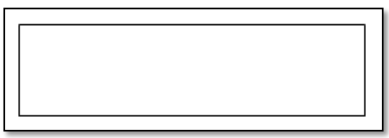


IMAGEN N°04-0003: Representación de una entidad regular

Entidades Débiles:

Existencia: Dependiente de otra entidad (la entidad fuerte o propietaria). No tienen una clave primaria completa por sí mismas; usan parte de la clave de la entidad fuerte.

Representación: Un rectángulo doble, tal como se muestra en la IMAGEN N°04-0004.

IMAGEN N°04-0004: Representación de una entidad débil

Relación con la Propietaria: Se unen a la entidad fuerte a través de una relación identificadora (o de dependencia), que se representa con un rombo doble.

Atributos

Los atributos son las características o propiedades de las entidades. Por ejemplo, la entidad Estudiante podría tener los atributos Código, Nombres, Apellidos y Carrera. Para la entidad "Persona", los atributos podrían ser: Nombre, Fecha de Nacimiento, Dirección, id_persona (Clave Primaria), su representación en el esquema lógico sería como se muestra en la IMAGEN N°04-0005

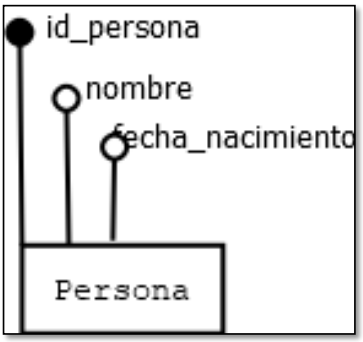


IMAGEN N°04-0005: Algunos Atributos de la entidad Persona

Tipos de Atributos Importantes:

- ✓ **Clave Primaria (Key Attribute):** El atributo (o conjunto de atributos) que identifica de forma única a cada instancia de la entidad. Se subraya en la representación gráfica.
- ✓ **Compuestos:** Atributos que se pueden dividir en subpartes (ej. Dirección se divide en Calle, Ciudad, Código Postal).
- ✓ **Multivaluados:** Atributos que pueden tener varios valores para una sola entidad (ej. Número de Teléfono).

La representación de los atributos en un Modelo Entidad-Relación (MER) se hace típicamente utilizando óvalos conectados a la entidad a la que pertenecen, tal como se muestra en la IMAGEN N°04-0005.

Relaciones

Las relaciones representan los vínculos entre entidades y es representado en el esquema conceptual con un rombo. Por ejemplo, un Estudiante “matricula” un Curso, lo que constituye una relación entre ambas entidades.

Una persona puede tener una o varios oficios o especialidad, esto se podría tomar como conjuntos el cual se mostraría en la IMAGEN N°04-0006, su

representación en el esquema conceptual de la relación de la entidad persona con especialidad seria, así como se muestra en la IMAGEN N°04-0007.

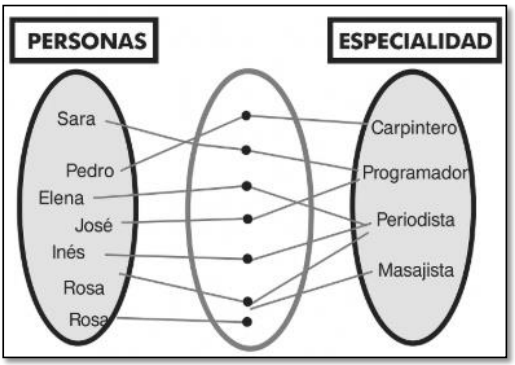


IMAGEN N°04-0006: Entidad como conjunto

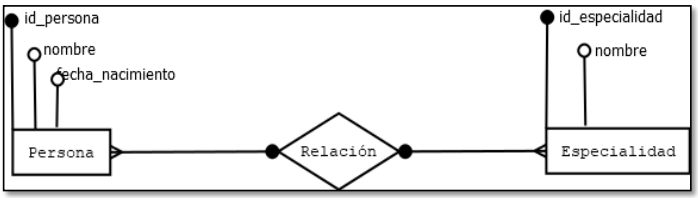


IMAGEN N°04-0007: Relación entre la entidad Persona y Especialidad, representado por un rombo

Un aspecto importante en el diseño conceptual es la cardinalidad, que indica el número de ocurrencias de una entidad que pueden asociarse con otra. Las cardinalidades más comunes son: uno a uno (1:1), uno a muchos (1:N) y muchos a muchos (N:M).

Por ejemplo, un cliente puede realizar muchos pedidos (1:N), pero cada pedido pertenece a un solo cliente; o un estudiante puede matricularse en varios cursos y cada curso puede tener varios estudiantes (N:M). Este tipo de reglas ayuda a

definir correctamente las relaciones y a evitar errores en el modelo lógico posterior.

El diseño conceptual también diferencia entre entidades fuertes y entidades débiles.

Una entidad fuerte puede existir por sí misma, como Empleado o Departamento, mientras que una entidad débil depende de otra para existir. Por ejemplo, DetalleFactura es una entidad débil, ya que no puede existir sin una Factura asociada. Estas dependencias se representan en el diagrama con claves primarias y claves foráneas en etapas posteriores.

En esta fase, el diseñador también debe identificar las restricciones de integridad, que son reglas que los datos deben cumplir. Existen tres tipos principales:

- ✓ **Integridad de entidad:** garantiza que cada registro tenga un identificador único.
- ✓ **Integridad referencial:** asegura que las relaciones entre tablas sean válidas (por ejemplo, no puede existir un pedido asociado a un cliente que no exista).
- ✓ **Restricciones semánticas:** reflejan reglas propias del negocio, como que el sueldo de un empleado no puede ser negativo o que una matrícula solo puede realizarse en periodos activos.

El diseño conceptual puede realizarse de forma manual o mediante herramientas especializadas de modelado, como MySQL Workbench, Lucidchart, Draw.io, Dia, ERDPlus o DBeaver, que permiten crear diagramas entidad-relación de manera visual. Estas herramientas facilitan la detección de errores y ofrecen la opción de transformar el modelo conceptual en un esquema lógico automáticamente.

Un ejemplo práctico de diseño conceptual sería el siguiente:

Supongamos que se desea desarrollar una aplicación en Java para gestionar una biblioteca. Las entidades principales serían Libro, Autor, Usuario y Préstamo.

- ✓ Cada Libro tiene atributos como CódigoLibro, Título, AñoPublicación y Género.
- ✓ Cada Autor tiene CódigoAutor, Nombre y Nacionalidad.
- ✓ La relación entre Libro y Autor es de muchos a muchos (N:M), ya que un autor puede escribir varios libros y un libro puede tener varios autores.
- ✓ Usuario se relaciona con Préstamo (1:N), y Préstamo se relaciona con Libro (N:M), porque un usuario puede pedir varios libros y un libro puede ser prestado varias veces.
- ✓ Este modelo conceptual servirá como base para definir las tablas y sus relaciones en la fase lógica.

En el contexto de la programación con Java, el modelo conceptual tiene una relación directa con el diseño orientado a objetos. Cada entidad del modelo puede representarse como una clase Java, con sus atributos y métodos. Por ejemplo, la entidad Libro puede representarse como la clase Libro, con atributos privados (codigoLibro, titulo, autor) y métodos públicos (getTitulo(), setTitulo(), etc.). Esta correspondencia facilita el uso de frameworks como Hibernate o Jakarta Persistence (JPA), que permiten mapear las clases Java directamente a las tablas de la base de datos.

4.4 DISEÑO LÓGICO Y FÍSICO

El diseño lógico y físico de una base de datos constituye la etapa final en el proceso de modelado de datos. Mientras que el diseño conceptual se centra en la representación abstracta de la información, el diseño lógico traduce este modelo a estructuras que pueden implementarse en un sistema de gestión de bases de datos (SGBD). En esta fase se definen las tablas, atributos, claves primarias y foráneas, relaciones y restricciones que garantizan la coherencia y la integridad de los datos.

El diseño lógico tiene como propósito principal optimizar la organización de la información para reducir la redundancia y mantener la consistencia entre los registros. Para ello, se aplican las reglas de normalización, un proceso que permite descomponer las tablas en estructuras más pequeñas y relacionadas, eliminando duplicidades. Este proceso no solo mejora el almacenamiento, sino también el rendimiento de las consultas y actualizaciones de datos.

Durante el diseño lógico, los diagramas entidad-relación (E-R) creados en la fase conceptual se transforman en esquemas relacionales. Las entidades se convierten en tablas, los atributos en columnas, y las relaciones en claves foráneas. Además, se definen los tipos de datos (como VARCHAR, INT, DATE, etc.) y las restricciones de integridad (NOT NULL, UNIQUE, CHECK, entre otras) que aseguran la validez de la información. En este punto, la base de datos ya tiene una estructura definida que podrá implementarse directamente en el gestor de base de datos.

Para realizar estas tareas de manera profesional y eficiente, una herramienta ampliamente utilizada es Erwin Data Modeler. Este software permite desarrollar tanto el diseño lógico como el diseño físico de la base de datos, generando automáticamente los esquemas y diagramas a partir del modelo conceptual. Con Erwin, es posible visualizar gráficamente las entidades, relaciones y atributos, aplicar normalización, y exportar el modelo directamente al código SQL del sistema gestor elegido (como MySQL, Oracle, SQL Server o PostgreSQL). Además, permite sincronizar el modelo con la base de datos real, detectar inconsistencias y documentar el diseño de manera automatizada.

El diseño físico, por su parte, traduce el modelo lógico a un entorno de ejecución real dentro del SGBD. En esta etapa se toman decisiones técnicas orientadas a mejorar el rendimiento, la seguridad y la disponibilidad del sistema. Esto incluye la creación de índices, la definición de particiones, la configuración de espacios de almacenamiento, la planificación de respaldo y recuperación, y la definición de políticas de acceso y permisos. Con herramientas como Erwin, estos aspectos pueden planificarse visualmente y ajustarse de acuerdo con las características específicas del servidor y del volumen de datos esperado.

Un aspecto esencial del diseño físico es la optimización del acceso a los datos. A través de índices, por ejemplo, se puede acelerar la búsqueda de registros, aunque su uso excesivo puede afectar el rendimiento en operaciones de inserción y actualización. También es importante determinar la ubicación física de los archivos de datos y de registro (logs) para mejorar la eficiencia en entornos con alta concurrencia de usuarios.

De igual manera, el diseño físico debe contemplar la seguridad y recuperación ante fallos. Se establecen mecanismos de respaldo, políticas de replicación y procedimientos de auditoría para garantizar la continuidad del sistema ante errores o pérdidas de información. En entornos empresariales, estas consideraciones son fundamentales para mantener la integridad y disponibilidad de los datos.

El diseño lógico y físico consolidan todo el proceso de modelado y preparación de la base de datos. A partir de esta fase, el sistema se encuentra listo para ser implementado y utilizado por las aplicaciones. En el caso de la programación con Java, esta estructura es aprovechada mediante tecnologías como JDBC, Hibernate o JPA, que permiten conectar el programa con la base de datos, ejecutar consultas, y gestionar transacciones de forma eficiente. De este modo, un diseño lógico y físico bien elaborado —apoyado en herramientas profesionales como Erwin Data Modeler— garantiza la estabilidad y escalabilidad del sistema desarrollado

4.5 CASO DE ESTUDIO 001: SISTEMA DE LOGISTICA

Se tiene el siguiente sistema para desarrollar:

En el proceso de “**Bienes y Suministros**” de la empresa de turismo “**Pata Amarilla**” que pertenece al área de Administración, se realizan las actividades de manera manual, por lo que se requiere sistematizar todo ello con la ayuda de un software, el dueño del proceso de "**Bienes y Suministros**" detalla el procedimiento y actividades de cada área:

- ✓ El proceso involucra cinco (5) áreas: Administración, Logística, Almacén, Patrimonio, Áreas usuarias.

- ✓ El área de Logística funciona de la siguiente forma:
  - ➢ Recibe las solicitudes de requerimientos de bienes o suministros de las diferentes áreas de la empresa.
  - ➢ Cada solicitud tiene un responsable que está adscrito a un área.
  - ➢ Cada solicitud es autorizada por el jefe del área y posteriormente por el área de Administración.
  - ➢ De la solicitud se requiere la siguiente información: Número de la solicitud (consecutivo), fecha, responsable, área, meta presupuestal. En cada solicitud se pueden contener uno o muchos ítems con la siguiente información: código, nombre del ítem, cantidad solicitada, unidad de medida, valor unitario.
  - ➢ Cada ítem es identificado por un código único y es de carácter devolutivo (suministro) (útiles de escritorio, útiles de limpieza, etc.) o un bien (computadora, escritorio, etc.).
  - ➢ La solicitud es enviada al área de Logística para atender si es que hay en stock o realizar la compra a uno o varios proveedores. Para realizar la compra se juntan todas las solicitudes para tener la cantidad total de ítems de cada rubro a comprar.
  - ➢ Las cotizaciones son realizadas con uno o varios proveedores de los bienes solicitados.
  - ➢ Para la compra de un producto, primero se realiza la cotización a dos o tres proveedores para determinar la mejor oferta en calidad y precio que ofrecen, una vez elegido el proveedor se genera la orden de compra, con los siguientes datos: número de **orden de compra**, nombre del proveedor al cual se le va a realizar la compra, fecha de la orden, monto total de la orden, fecha de entrega. Cada orden puede tener asociado uno o varios ítems, cada ítem debe tener los siguientes datos: código del ítem, nombre del ítem, cantidad de compra, unidad de medida del ítem, valor unitario y sub total.
  - ➢ La orden de compra es aprobada por el área de Administración luego el área de Logística envía o hace entrega al proveedor elegido.
- ✓ El área de Almacén funciona de la siguiente forma:

- Recepciona los bienes y/o suministros que llegan de los proveedores y distribuye con una pecosa a las áreas y trabajadores que realizaron las solicitudes.
- El proveedor hace entrega a Almacén los ítems detallado en la Orden de Compra, para ello adjunta la guía de remisión y factura para su pago correspondiente si todo está conforme, se registra una entrada de almacén por cada factura relacionada, con los siguientes datos: número de entrada, fecha, número de factura, Proveedor, total bienes, valor total, los ítems recibidos con la cantidad respectiva.
- El área de Almacén hace entrega de los bienes a las diferentes áreas solicitantes atreves de una pecosa, detallando cada ítem entregado y otros datos más como el numero de salida, empleado solicitante del ítem o los ítems, cantidad, fecha de salida, fecha de entrega.
- ✓ El área Patrimonio es responsable de:
- Administrar y controlar la ubicación de los bienes dentro de la empresa, por esto antes de que el bien salga del Almacén es codificado y se genera su estiker que se adhiere al bien.
- Cada trabajador tiene a su cargo una o varios bienes, esto permite controlar su ubicación.

Diseñar el esquema conceptual para el sistema de gestión de "**Bienes y Suministros**" de la empresa de turismo "**Pata Amarilla**".

001-01 DISEÑO CONCEPTUAL

En la IMAGEN N°04-0008 se tiene un diagrama de flujo para entender el camino de una solicitud.

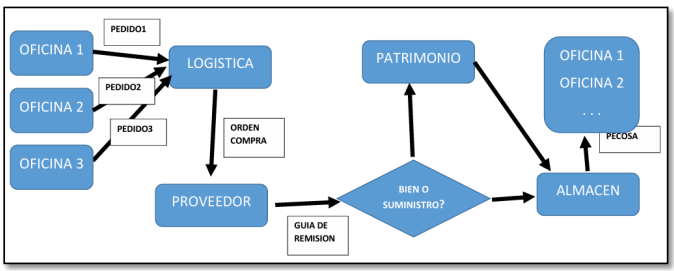


IMAGEN N°04-0008: Flujograma de solicitud de productos por las áreas usuarias

De los requerimientos se identifica a las posibles entidades marcando con amarillo y las posibles relaciones marcando con verde:

En el proceso de "**Bienes y Suministros**" de la empresa de turismo "**Pata Amarilla**" que pertenece al área de Administración, se realizan las actividades de manera manual, por lo que se requiere sistematizar todo ello con la ayuda de un software, el dueño del proceso de "**Bienes y Suministros**" detalla el procedimiento y actividades de cada área:

- ✓ El proceso involucra cinco (5) áreas: Administración, Logística, Almacén, Patrimonio, Áreas usuarias.
- ✓ El área de Logística funciona de la siguiente forma:
- ➢ Recibe las solicitudes de requerimientos de bienes o suministros de las diferentes áreas de la empresa.
- ➢ Cada solicitud tiene un responsable **que está adscrito** a un área.
- ➢ Cada solicitud es **autorizada** por el jefe del área y **posteriormente por el área de Administración**.

- De la solicitud se requiere la siguiente información: Número de la solicitud (consecutivo), fecha, responsable, área, meta presupuestal. En cada solicitud se pueden contener uno o muchos ítems con la siguiente información: código, nombre del ítem, cantidad solicitada, unidad de medida, valor unitario.
- Cada ítem es identificado por un código único y es de carácter devolutivo (suministro) (útiles de escritorio, útiles de limpieza, etc.) o un bien (computadora, escritorio, etc.).
- La solicitud es enviada al área de Logística para atender si es que hay en stock o realizar la compra a uno o varios proveedores. Para realizar la compra se juntan todas las solicitudes para tener la cantidad total de ítems de cada rubro a comprar.
- Las cotizaciones son realizadas con uno o varios proveedores de los bienes solicitados.
- Para la compra de un producto, primero se realiza la cotización a dos o tres proveedores para determinar la mejor oferta en calidad y precio que ofrecen, una vez elegido el proveedor se genera la orden de compra, con los siguientes datos: **número de orden de compra, nombre del proveedor al cual se le va a realizar la compra, fecha de la orden, monto total de la orden, fecha de entrega**. Cada orden puede tener **asociado uno o varios ítems**, cada ítem debe tener los siguientes datos: **código del ítem, nombre del ítem, cantidad de compra, unidad de medida del ítem, valor unitario y sub total.**
- La orden de compra es **aprobada por el área de Administración** luego el **área de Logística** envía o hace entrega al proveedor elegido.

✓ El área de Almacén funciona de la siguiente forma:

- Recepcióna los bienes y/o suministros que llegan de los proveedores y distribuye con una pecosa a las áreas y trabajadores que realizaron las solicitudes.
- El proveedor hace entrega a Almacén los ítems detallado en la Orden de Compra, para ello adjunta la guía de remisión y factura para su pago correspondiente si todo está conforme, se registra una entrada de almacén por cada factura relacionada, con los siguientes datos: **número de entrada,**

fecha, número de factura, Proveedor, total bienes, valor total, los ítems recibidos con la cantidad respectiva.

- El área de Almacén hace entrega de los bienes a las diferentes áreas solicitantes atreves de una pecosa, detallando cada ítem entregado y **otros datos más como el número de salida, empleado solicitante del ítem o los ítems, cantidad, fecha de salida, fecha de entrega**.
- ✓ El área Patrimonio es responsable de:
- Administrar y controlar la **ubicación** de los bienes dentro de la empresa, por esto antes de que el bien salga del Almacén es codificado y se genera su estiker que se adhiere al bien.
- Cada trabajador tiene a su cargo una o varios bienes, esto permite controlar su ubicación.

LISTA DE POSIBLES ENTIDADES Y SUS ATRIBUTOS

- Area
  - ✓ código
  - ✓ nombre
  - ✓ dependencia (de otra area)
- Usuarios(trabajadores)
  - ✓ DNI
  - ✓ apellidos
  - ✓ nombre
  - ✓ teléfono
  - ✓ correo
  - ✓ dirección
  - ✓ fecha_nacimiento
  - ✓ dependencia (área donde trabaja)
  - ✓ bienes[bien,código,estado]
- Bienes o suministros(item)
  - ✓ código
  - ✓ nombre
  - ✓ stock
  - ✓ unidad medida
  - ✓ valor unitario
  - ✓ bien o suministro
- Unidad de medida
  - ✓ codigo
  - ✓ nombre

    - ✓ abreviatura
- Categoria_item
    - ✓ Id_categoria_item
    - ✓ descripcion
- Jefe de área(cargo)
    - ✓ código
    - ✓ descripcion
- Meta presupuestal
    - ✓ código
    - ✓ descripción
    - ✓ presupuesto
- Proveedor
    - ✓ RUC
    - ✓ nombre
    - ✓ dirección
    - ✓ teléfono
    - ✓ correo
- Bienes
    - ✓ código_patrimonial
    - ✓ ubicación
    - ✓ trabajador
    - ✓ estado

LISTA DE POSIBLES RELACIONES Y SUS ATRIBUTOS

- Solicitud
    - ✓ numero_solicitud
    - ✓ fecha_solicitud
    - ✓ responsable
    - ✓ área
    - ✓ meta_presupuestal
    - ✓ autorización_jefe_area
    - ✓ autorización_administracion
    - ✓ ítems[ítem, cantidad_solicitud, unidad_medida,valor_unitario]
- Cotización
    - ✓ numero_cotizacion
    - ✓ fecha_cotizacion
    - ✓ ítems[ítem, unidad_medida, cantidad]
    - ✓ proveedores[proveedor, dirección, teléfono, correo]

- Orden_compra
    - ✓ numero_orden_compra
    - ✓ proveedor
    - ✓ fecha_orden_compra

  - ✓ monto_total
  - ✓ fecha_entrega
  - ✓ ítems[ítem,cantidad,unidad,valor_unitario]:
  - ✓ aprobado
- Guía_remision
  - ✓ numero_guia_remision
  - ✓ numero_orden_compra
  - ✓ numero_factura
  - ✓ proveedor
  - ✓ fecha_entrega
  - ✓ ítems[ítem, unidad_medida, cantidad, valor_unitario]
- Pecosa
  - ✓ numero_pecosa
  - ✓ trabajador
  - ✓ numero_solicitud
  - ✓ fecha_pecosa
  - ✓ fecha_entrega
  - ✓ ítems[ítem,unidad_medida, cantidad, valor_unitario]

Para considerar otras entidades que no se encuentran explicitas se busca más información y se hace un análisis más detallado del caso.

EN EL CASO DE ÁREA

En el diseño de las instituciones la estructura orgánica y funcional agrupa las competencias y funciones de las entidades en **unidades de organización** y establece las líneas de autoridad y mecanismos de coordinación para el logro de sus objetivos, a esto se acostumbra a denominarlos **áreas**, por lo que para el diseño conceptual considerar como **Unidad_organica**.

En el **Reglamento de Organización y Funciones** se desarrolla la estructura orgánica de las entidades y se representa en el organigrama.

En el organigrama cada unidad de organización es representado por rectángulos y las dependencias unidas por líneas.

En la IMAGEN N°04-0009, se tiene el organigrama de una municipalidad, se puede apreciar que la unidad orgánica de alcaldía depende de la unidad de organización de concejo municipal, además las dependencias de alcaldía son:

PROCURADURIA PUBLICA MUNICIPAL, SECRETARIA GENERAL, GERENCIA MUNICIPAL

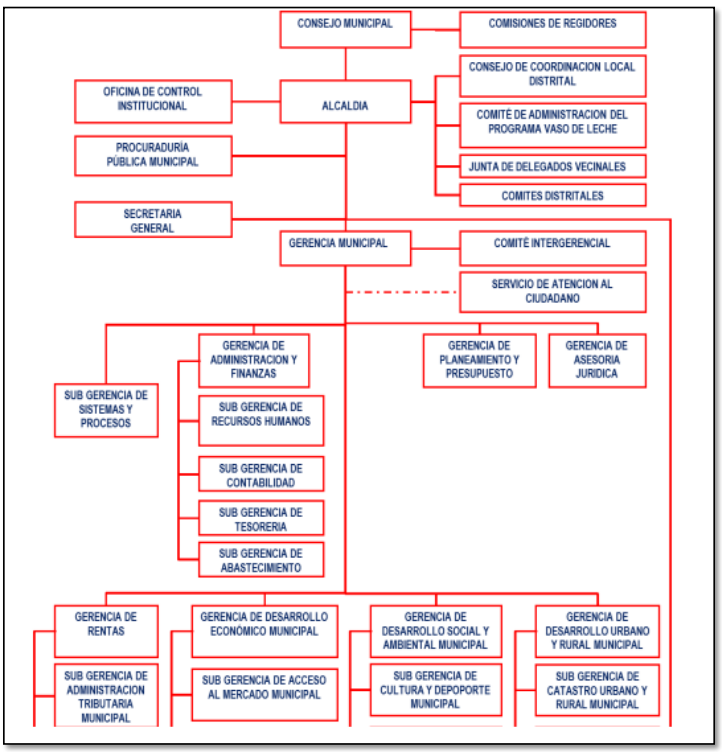


IMAGEN N°04-0009: Organigrama de una municipalidad

En la IMAGEN N°04-0009 se puede apreciar que las unidades orgánicas representado por rectángulos dependen de otras unidades orgánicas y tienen a su cargo también unidades orgánicas, por lo que se tiene que considerar una relación recursiva con la misma entidad:

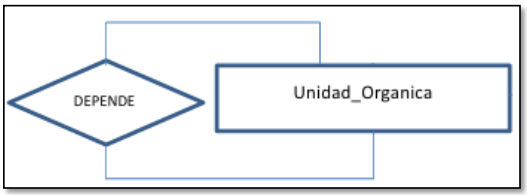
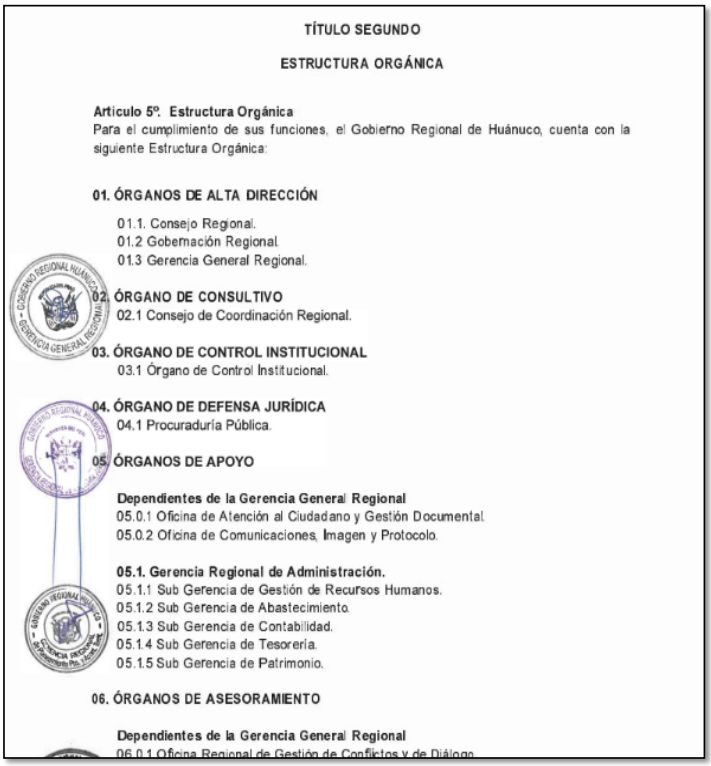


IMAGEN N°04-0010: Relación recursiva entre unidad_organica

La función de las áreas se puede encontrar en el Reglamento de Organizaciones y funciones y los cargos de cada trabajador relacionado a los cargos en el Manual de Organizaciones y funciones.

IMAGEN N°04-0011: Reglamento de Organizaciones y funciones Gobierno Regional Huánuco



TÍTULO SEGUNDO

ESTRUCTURA ORGÁNICA

Artículo 5º. Estructura Orgánica
Para el cumplimiento de sus funciones, el Gobierno Regional de Huánuco, cuenta con la siguiente Estructura Orgánica:

01. ÓRGANOS DE ALTA DIRECCIÓN

01.1. Consejo Regional.
01.2 Gobernación Regional.
01.3 Gerencia General Regional.

02. ÓRGANO DE CONSULTIVO
02.1 Consejo de Coordinación Regional.

03. ÓRGANO DE CONTROL INSTITUCIONAL
03.1 Órgano de Control Institucional.

04. ÓRGANO DE DEFENSA JURÍDICA
04.1 Procuraduría Pública.

05. ÓRGANOS DE APOYO

Dependientes de la Gerencia General Regional
05.0.1 Oficina de Atención al Ciudadano y Gestión Documental.
05.0.2 Oficina de Comunicaciones, Imagen y Protocolo.

05.1. Gerencia Regional de Administración.
05.1.1 Sub Gerencia de Gestión de Recursos Humanos.
05.1.2 Sub Gerencia de Abastecimiento.
05.1.3 Sub Gerencia de Contabilidad.
05.1.4 Sub Gerencia de Tesorería.
05.1.5 Sub Gerencia de Patrimonio.

06. ÓRGANOS DE ASESORAMIENTO

Dependientes de la Gerencia General Regional
06.0.1 Oficina Regional de Gestión de Conflictos y de Diálogo

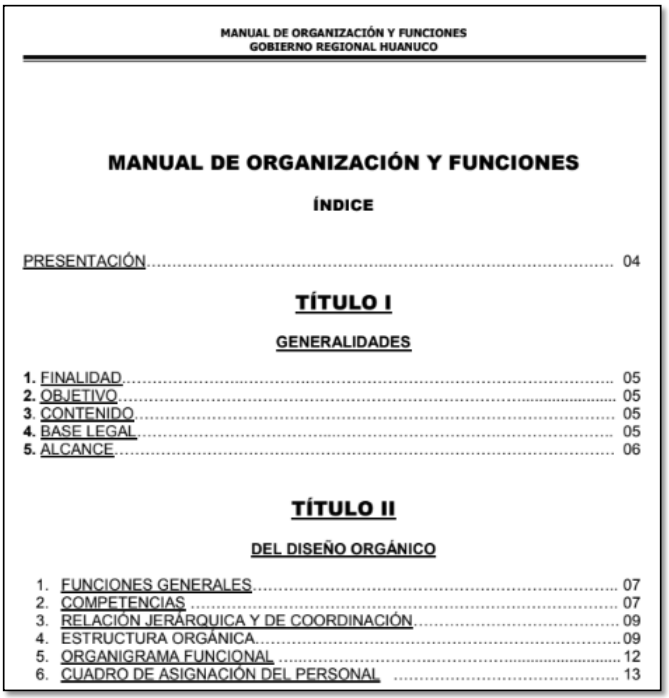
IMAGEN N°04-0012: ROF estructura orgánica

De la estructura orgánica se puede apreciar la dependencia y niveles entre unidades orgánicas:

Por lo que se considera cono entidad a **Tipo_unidad_organica**

IMAGEN N°04-0013: entidad Tipo_unidad_organica

En la IMAGEN N°04-0014 se muestra el MOF del Gobierno regional Huánuco, en la cual se puede apreciar el contenido de dicho instrumento de gestión.



MANUAL DE ORGANIZACIÓN Y FUNCIONES
GOBIERNO REGIONAL HUANUCO

MANUAL DE ORGANIZACIÓN Y FUNCIONES

ÍNDICE

IMAGEN N°04-0014: MOF del Gobierno Regional Huánuco

MANUAL DE ORGANIZACIÓN Y FUNCIONES
GOBIERNO REGIONAL HUANUCO

6. CUADRO DE ASIGNACIÓN DE PERSONAL

ÓRGANO NORMATIVO Y FISCALIZADOR

SECRETARIA DEL CONSEJO REGIONAL

Nº DE ORDEN	DENOMINACIÓN DE LA UNIDAD ORGÁNICA Y CARGOS CLASIFICADOS	TOTAL CARGOS	CLASIFICACIÓN	CÓDIGO
001	Director	1	Directivo Superior	448-1-1-0-D-0
002	Asesor II	1	Especialista	448-1-1-0-E-2
003	Técnico Administrativo II	1	De apoyo	448-1-1-0-A-2

ÓRGANOS DE GOBIERNO

PRESIDENCIA REGIONAL

Nº DE ORDEN	DENOMINACIÓN DE LA UNIDAD ORGÁNICA Y CARGOS CLASIFICADOS	TOTAL CARGOS	CLASIFICACIÓN	CÓDIGO
004	Presidente Regional	1	FP (ELEGIDO)	448-2-1-0-F-0
005	Vicepresidente	1	FP (ELEGIDO)	448-2-1-0-F-0
006	Asesor II	1	Especialista	448-2-1-0-E-2
007	Asistente Administrativo II	1	Especialista	448-2-1-0-E-2
008	Chofer III	1	De apoyo	448-2-1-0-A-3
009	Trabajador de Servicios III	1	De apoyo	448-2-1-0-A-3

ÓRGANO DE CONTROL

IMAGEN N°04-0015: Cuadro de asignación de personal detallado en el MOF

En el MOF se puede apreciar que existen los cargos y que estos a su vez pertenecen a una clasificación, por lo que se considera las siguientes entidades: Cargo y Categoría.

IMAGEN N°04-0016: Entidad Cargo y Categoría

En la IMAGEN N°04-0018 se puede apreciar las funciones asignado a cada cargo, por lo que se considera una entidad Función.

IMAGEN N°04-0017: Entidad Función

DEL MANUAL DE ORGANIZACIÓN Y FUNCIONES (MOF)

CAPÍTULO I

ÓRGANO NORMATIVO Y FISCALIZADOR

SECRETARIA DEL CONSEJO REGIONAL

1. **FINALIDAD**
   La Secretaría del Consejo Regional es el órgano responsable de programar, ejecutar y coordinar el apoyo administrativo, para un mejor cumplimiento de las funciones del Consejo Regional, está a cargo de un Director de nivel profesional Letrado, a propuesta del Consejo Regional, designado por el Presidente Regional.

2. **UBICACIÓN**
   La Secretaría del Consejo Regional es un órgano de apoyo del Consejo Regional, dentro de la estructura orgánica del Gobierno Regional Huánuco.

3. **FUNCIONES GENERALES**
   a) Recepcionar, clasificar y procesar las diversas comunicaciones dirigidas al Consejo Regional.
   b) Certificar y Llevar las actas de las Sesiones de Consejo y de las Comisiones Permanentes de Consejo Regional.
   c) Grabar el desarrollo de las sesiones de consejo regional y redactar el acta de cada una de estas, entregando una copia de la misma a cada miembro del consejo regional, con 48 horas de anticipación para su revisión antes de ser aprobada.
   d) Emitir citaciones a las sesiones ordinarias, solemnes y extraordinarias, con la anticipación establecida en el Reglamento Interno y con agenda documentada.
   e) Proyectar las ordenanzas, enviarlo para la promulgación por el Presidente Regional, llevando el registro de los mismos

IMAGEN N°04-0018: funciones detallado en el MOF

Para realizar el esquema conceptual, con las **entidades** encontradas se busca las **relaciones** entre ellas, tal como se muestra en la IMAGEN N°04-0019. Una unidad orgánica(Unidad_organica) pertenece a un tipo de unidad orgánica(Tipo_unidad_organica), esto viendo en cardinalidad seria: UNO A UNO

(1:1) y un tipo de unidad orgánica es parte de muchas unidades orgánicas, en este caso la cardinalidad es de UNO A MUCHOS (1: n).

Para que el esquema conceptual sea más legible y por el espacio, no se considera el **ROMBO** en las relaciones que no tengan atributos, esto sucede generalmente en las relaciones de **UNO a UNO**, **UNO a MUCHOS**, de **MUCHOS a UNO**.

En la relación entre Trabajador y Unidad_organica, un trabajador está contratado para que labore en una unidad orgánica y en un puesto o cargo asignado según los instrumentos de gestión, por lo que a esta relación se denominara **Contrato** y se representa con un **ROMBO (**IMAGEN N°04-0019**)** en lo cual se consideran sus propios atributos (numero_contrato, fecha_contrto, fecha_inico, fecha_termino, sueldo, horario,etc.)

En la IMAGEN N°04-0019 se presenta el esquema conceptual de la parte de unidades orgánicas y trabajador, diseñado con el **software Dia.**

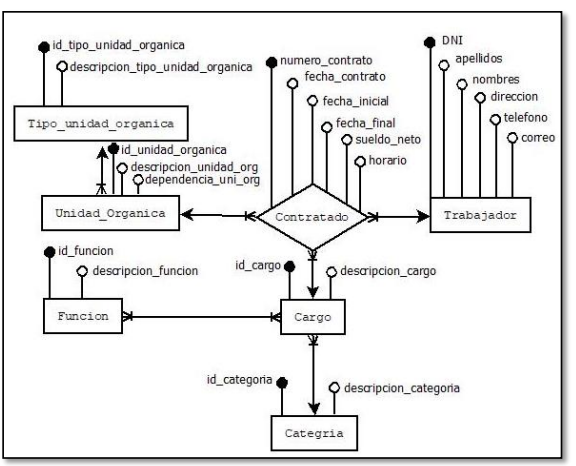


IMAGEN N°04-0019: Esquema conceptual de la parte de unidades orgánicas

Para completar el diagrama conceptual considerando las solicitudes, las órdenes de compras a los proveedores, etc. se buscan las relaciones entre las entidades,

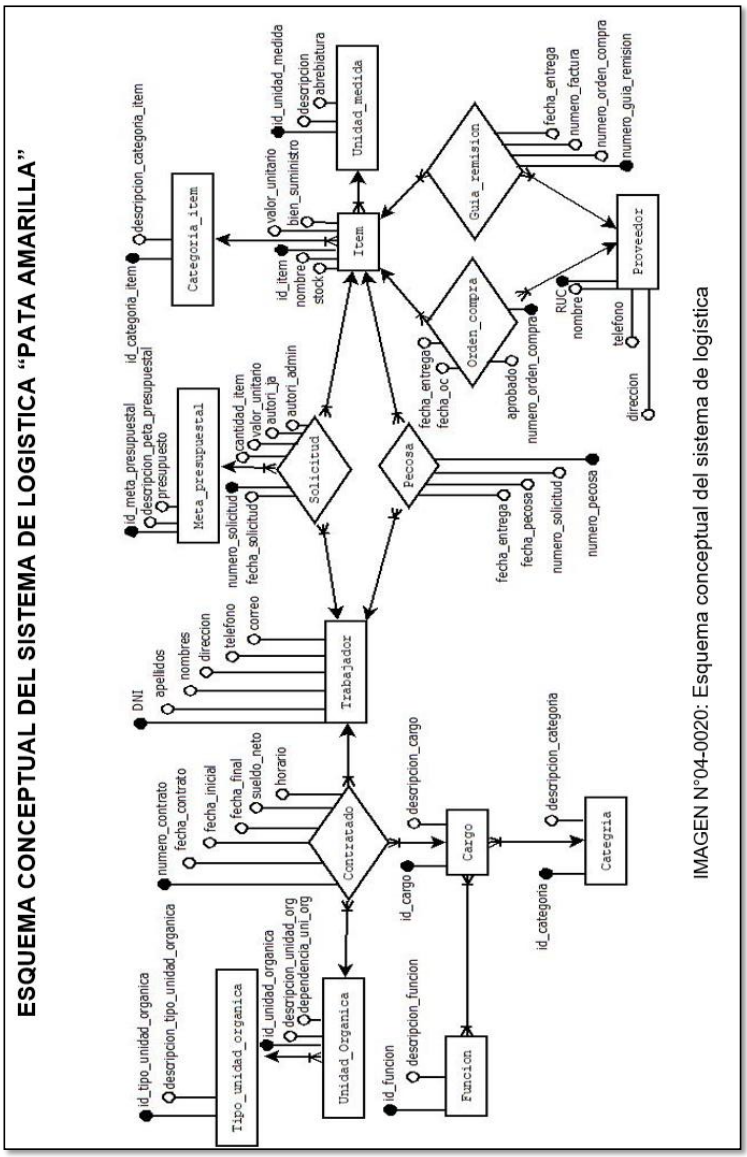
todo esto se puede apreciar en el esquema conceptual diseñado en el **software Dia** tal como se muestra en la IMAGEN N°04-0020.

El trabajador solicita uno o muchos bienes o suministros(items), los ítems están en muchas solicitudes de Trabajadores, por lo que la relación entre la entidad **Trabajador** e **Item** es de **MUCHOS a MUCHOS** (m: n), a esta relación se le llama **Solicitud** y es representado por un **ROMBO** con sus propios atributos (numero_solicitud, fecha_solicitud, autorización_jefe, autorización_administracion, ítems [ítem, cantidad_item, unidad_medida, valor_unitario]). Así mismo en esta misma relación participa la entidad **Meta_presupuestal,** en una solicitud participa sola una meta presupuestal **UNO a UNO** (1:1) y una meta presupuestal puede estar en una o muchas solicitudes **UNA a MUCHAS** (1: n) tal como se ve en la Figura N°001-14.

Entre la entidad **Trabajador** y la entidad **Item** hay otra relación que corresponde a la entrega de los productos solicitados por parte de la oficina de almacén, las salidas están plasmadas en un documento llamado pecosa, por lo que a la relación se le denomina **Pecosa** y es representado con un **ROMBO** con sus propios atributos (numero_pecosa, trabajador, numero_solicitud, fecha_pecosa, fecha_entrega, ítems [ítem,unidad_medida, cantidad, valor_unitario]).

La entidad **Item** y **Proveedor** se relacionan a través de la orden de compra, por lo que a esta relación se le denomina **Orden_compra** y es representado con un **ROMBO** con sus propios atributos (numero_orden_compra, proveedor, fecha_orden_compra, monto_total, fecha_entrega, aprobado, ítems[ítem,cantidad,unidad,valor_unitario].

Entre la entidad **Item** y **Proveedor** hay otra relación denominado **Guia_remision** en la cual se registra las entradas de los productos por parte del proveedor a la oficina de almacén, esta relación es representado también con un ROMBO con sus propios atributos (numero_guia_remision, numero_orden_compra, numero_factura, proveedor, fecha_entrega, ítems[ítem, unidad_medida, cantidad, valor_unitario]).


ESQUEMA CONCEPTUAL DEL SISTEMA DE LOGISTICA "PATA AMARILLA"
Trabajador
DNI
apellidos
nombres
direccion
telefono
correo
Contratado
numero_contrato
fecha_contrato
fecha_inicial
fecha_final
sueldo_neto
horario
Cargo
id_cargo
descripcion_cargo
Funcion
id_funcion
descripcion_funcion
Unidad_Organica
id_unidad_organica
descripcion_unidad_org
dependencia_uni_org
Tipo_unidad_organica
id_tipo_unidad_organica
descripcion_tipo_unidad_organica
Categoria
id_categoria
descripcion_categoria
Solicitud
numero_solicitud
fecha_solicitud
cantidad_item
valor_unitario
autori_ja
autori_admin
Meta_presupuestal
id_meta_presupuestal
descripcion_peta_presupuestal
presupuesto
Pecosa
fecha_entrega
fecha_oc
fecha_pecosa
numero_solicitud
numero_pecosa
Item
id_item
nombre
stock
valor_unitario
bien_suministro
Categoria_item
id_categoria_item
descripcion_categoria_item
Unidad_medida
id_unidad_medida
descripcion
abreviatura
Orden_compra
aprobado
numero_orden_compra
Guia_remision
fecha_entrega
numero_factura
numero_orden_compra
numero_guia_remision
Proveedor
RUC
nombre
telefono
direccion
IMAGEN N°04-0020: Esquema conceptual del sistema de logística

001-02 DISEÑO LOGICO

Para el diseño lógico se utilizará el **modelo relacional**, donde: las entidades, relaciones (con atributos) y algunos atributos se representan con tablas(relaciones), con una representación gráfica de rectángulo, con dos compartimientos, en el primer compartimiento los atributos que hacen de llave primaria PK y el segundo compartimiento otros atributos tal como se puede ver en la Figura N°001-15:

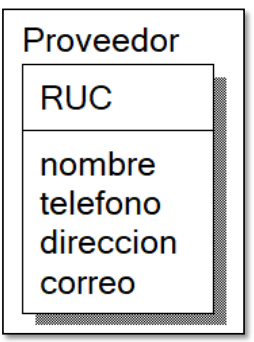


IMAGEN N°04-0021: Representación de una relación(tabla) en el esquema lógico

En el esquema conceptual de la IMAGEN N°04-0020, se muestran 11 entidades y 5 relaciones quienes pasaran a ser tablas(relaciones) en el esquema lógico.

En el esquema conceptual hay entidades que son hojas, los cuales se coloca primero en el esquema lógico:

- ✓ Tipo_unidad_organica(id_tipo_unidad_organica, descripcion_tipo_unidad_organica)
- ✓ Funcion(id_funcion, descripción_funcion)
- ✓ Categoria(id_categoria, descripción_categoria)
- ✓ Categoria_item(id_categoria_item, descripción_categoria_item)

- ✓ Unidad_medida(id_unidad_medida, descripción, abreviatura)
- ✓ Proveedor (RUC, nombre, dirección, telefono)
- ✓ Meta_presupuestal(id_meta_presupuestal, descripción_meta_presupuestal, presupuesto)
- ✓ Trabajador (DNI, apellidos, nombres, direccion, teléfono, correo)

En las relaciones entre las entidades de UNO a MUCHOS como en el caso de la entidad Tipo_unidad_organica que puede estar presente en MUCHAS de las instancias de la entidad Unidad_organica, en el esquema lógico la tabla Unidad_organica contendrá como una de sus columnas a la llave primaria(PK) de la entidad Tipo_unidad_organica en este caso el ***id_tipo_unidad_organica*** (como llave foránea FK en la taba Unidad_organica), así seria para las demás entidades con las mismas cardinalidades:

- ✓ Unidad_organica(id_unidad_organica, id_tipo_unidad_organica, descripcion_unidad_organica, dependencia_unidad_organica)
- ✓ Cargo (id_cargo, id_funcion, id_categoria, descripción_cargo)
- ✓ Item(id_item, id_categoria_item, id_unidad_medida, nombre, stock, valor_unitario, bien_suministro).

En el caso de las relaciones del esquema conceptual: Contrato, Solicitud, Pecosa, Orden_compra, Guia_remision; pasan a ser tablas(relaciones) con sus respectivos atributos, las llaves primarias de las entidades que relacionan pasan como llaves foranes a la tabla:

- ✓ Contrato (numero_contrato, id_unidad_organica, id_cargo, DNI_trabajador, fecha_contrato, fecha_inicial, fecha_final, sueldo_neto, horario).
- ✓ Solicitud (numero_solicitud, DNI_trabajador, id_meta_presupuestal, decha_solicitud, ítems [id_item, cantidad_item, valor_unitario, unidad], autorización_jefe, autorización_administracion).
- ✓ Pecosa (numero_pecosa, DNI_trabajador, fecha_pecosa, fecha_entrega, numero_solicitud, ítems[id_item, cantidad_item, valor_unitario, unidad])

- ✓ Orden_compra(numero_orden_compra,RUC_proveedor, fecha_orden_compra, fecha_entrega, aprobado, ítems[id_item, cantidad_item, valor_unitario, unidad])
- ✓ Guia_remision(numero_guia_remision, RUC_proveedor, numero_orden_compra, numero_factura, fecha_entrega, ítems[id_item, cantidad_item, valor_unitario, unidad])

El esquema lógico se diseña con el Software Erwin, seleccionar File →New y se tendrá la pantalla que se muestra en la IMAGEN N°04-0022.

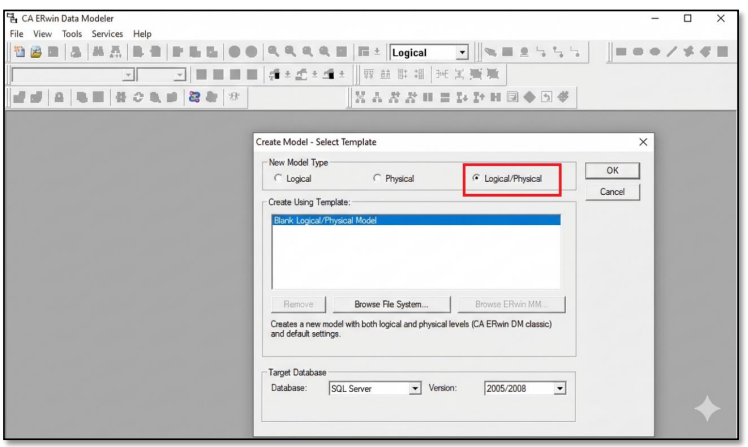


IMAGEN N°04-0022: Creación de un esquema lógico y físico con Erwin

Se selecciona la opción Logical/Physical para poder diseñar los esquemas lógico y físico de la base de datos.

Como en estos esquemas las entidades, relaciones y algunos atributos, en el modelo relacional son tablas, en la IMAGEN N°04-0023 se muestra los iconos para agregar las tablas y las relaciones según el tipo.

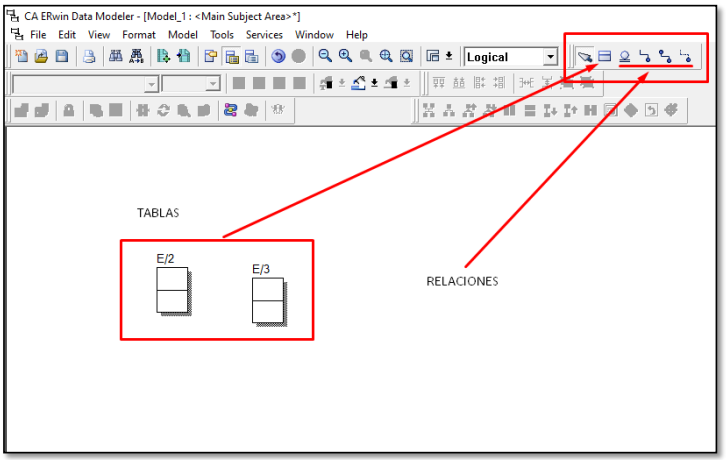


IMAGEN N°04-0023: Creación de un esquema lógico y físico con Erwin

Para empezar a diseñar el esquema lógico se agrega todas las tablas con sus atributos detallado en los párrafos anteriores, empezando por las tablas hoja, luego las tablas que tienen como llave foránea de otras tablas, también agregar tablas en las relaciones de muchos a muchos, luego las tablas que se originan de las relaciones con atributos, así como se muestra en la IMAGEN N°04-0024.

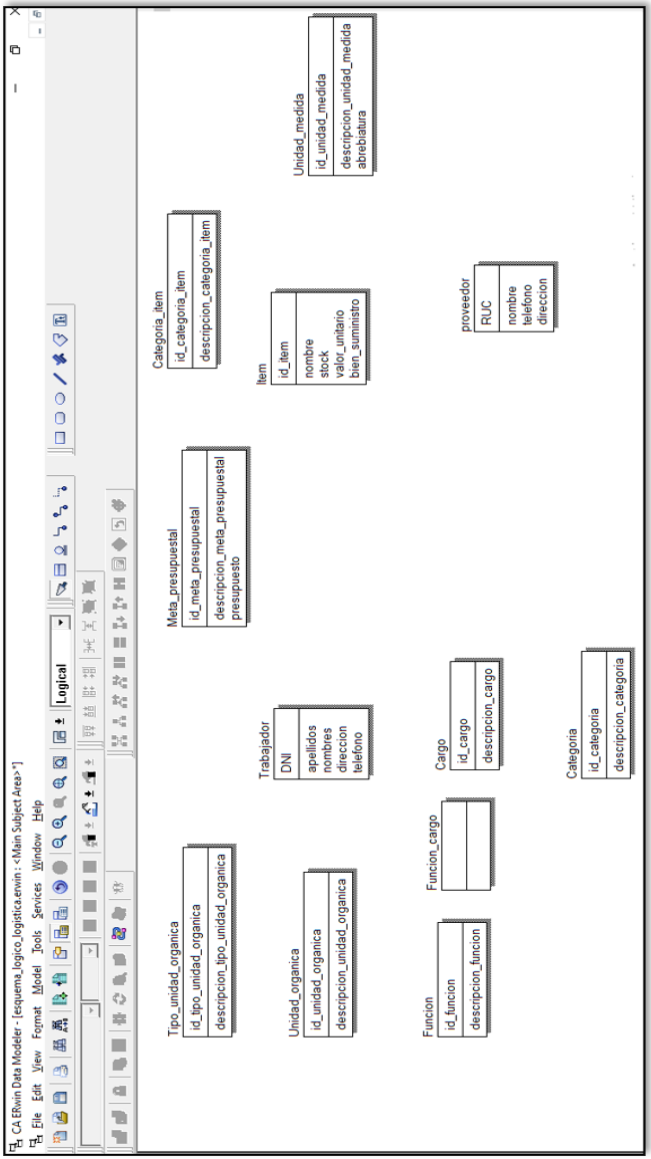


IMAGEN N°04-0024: Primera versión del esquema lógico del sistema de logística en Erwin

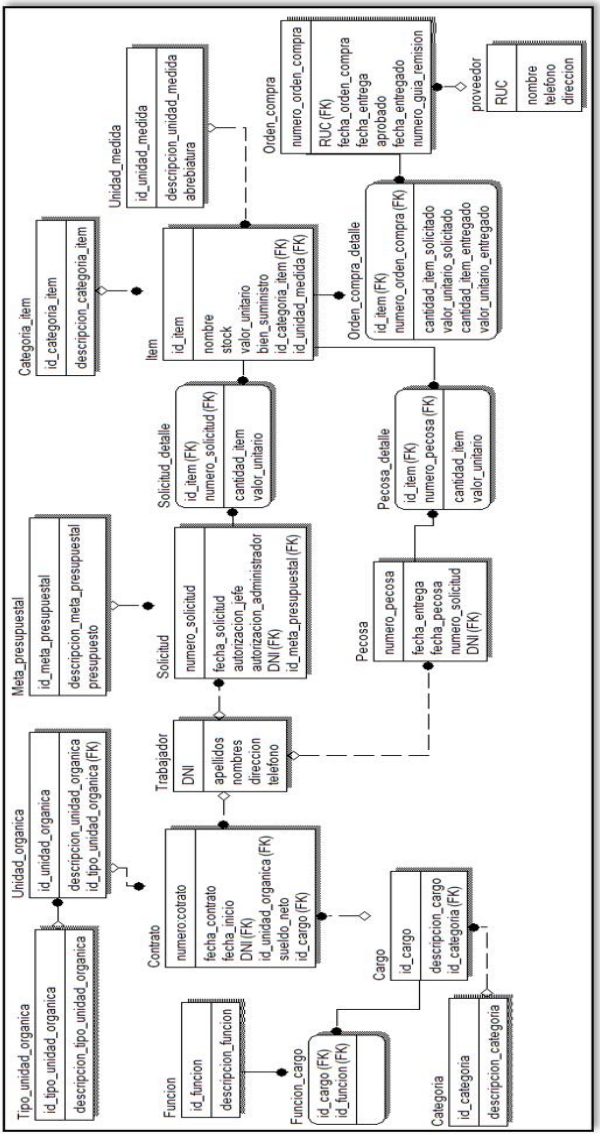


IMAGEN N°04-0025: Primera versión del esquema lógico del sistema de logística en Erwin

CAPÍTULO 5. SQL Y SISTEMAS GESTORES DE BASE DE DATOS: SQL SERVER, MYSQL, MARIADB

5.1 SQL(STRUCTURED QUERY LANGUAGE)

Lenguaje de consulta estructurado, estándar que permite interactuar con bases de datos relacional en: Definición (Lenguaje de definición de datos-LDD), consulta y Manipulación de datos (Lenguaje interactivo de manipulación de datos-LMD) y control de datos y transacciones.

5.2 DDL (LENGUAJE DE DEFINICIÓN DE DATOS)

Es el encargado de la gestión de la estructura de los objetos de la base de datos con cuatro operaciones: ***CREATE, ALTER, DROP y TRUNCATE***.

CREATE

Esta sentencia permite crear objetos de base de datos: base de datos, tablas, vistas, procedimientos almacenados, funciones, entre otros.

Por ejemplo, para crear una base de datos, una tabla:

- ***CREATE DATABASE*** *sis_academico;*
- ***CREATE TABLE*** *table_name*(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  ....
);

Sintaxis en MySQL

```
CREATE TABLE alumno (
  codigo varchar(10),
  apellidos varchar(50),
  nombre varchar(50),
  direccion varchar(50),
  fecha_nacimiento date
)
  CREATE TABLE curso(
codigo char(10),
nombre varchar(50),
credito int
)
```

ALTER

Esta sentencia permite modifica la estructura de un objeto de base de datos. Se puede modificar, quitar campos de una tabla, vistas, entre otros.

Por ejemplo, se agrega una columna a la tabla alumno:

- ***ALTER TABLE*** *alumno* ***ADD*** *correo varchar(50);*

DROP

Esta sentencia permite eliminar un objeto de la base de datos, puede ser base de datos, tabla, vista, procedimiento almacenado, entre otros.

Por ejemplo, para eliminar una tabla “curso” la sintaxis seria así:

- ***DROP TABLE*** *curso;*

TRUNCATE

Esta sentencia permite borrar el contenido de una tabla, por ejemplo, para borrar todo el contenido de la tabla alumnos escribiríamos así:

TRUNCATE TABLE *alumno;*

5.3 DML (LENGUAJE DE MANIPULACIÓN DE DATOS)

Permite alterar el contenido de los registros de una tabla, realizar consultas, las cuatro operaciones fundamentales de la gestión de toda base de datos: ***INSERT, UPDATE, DELETE, SELECT***.

INSERT

Esta sentencia permite agregar una o más registros a una tabla relacional, la cantidad de columnas y valores tienen que ser iguales y del mismo tipo asignado en su definición.

Sintaxis:

```
INSERT INTO
     nombre_tabla(column1,column2,…)
VALUES
     ('valor1', 'valor2', …)
```

Insertando múltiples registros o filas:

```
INSERT INTO
     nombre_tabla(column1,column2,…)
VALUES
     ('valor11', 'valor21', …),
     ('valor12', 'valor22', …),
     ('valor13', 'valor23', …),
          .
          .
          .
```

Insertando todas las columnas:

```
INSERT INTO nombre_tabla VALUES('valor1', 'valor2',…)
```

Insertando desde una consulta:

```
INSERT INTO nombre_tabla SELECT 'valor1','valor2',… FROM
```

Por ejemplo, insertando un registro a la tabla alumno:

```
INSERT INTO
     alumno(codigo,apellido,nombre,dirección,
     fecha_nacimiento,correo)
VALUES
     ('2023012354', 'CHUQUIYAURI SALDIVAR','ELMER', 'JR.
     ENRIQUE LOPEZ ALBUJAR MZ H LT. 2 SET 4', '10-07-
     1980',elmerchuquiyauri@gmail.com)
```

Insertando varios registros a la tabla alumno:

```
INSERT INTO
     alumno(codigo,apellidos,nombre,direccion,
     fecha_nacimiento,correo)
VALUES
     ('2023012888', 'GARCIA CHAVEZ','MIRANDA', 'JR. 28 DE
     JULIO #125', '01-06-1990', 'miranda@gmail.com'),

     ('2023012889', 'MARTINEZ SANDOVAL','JUAN', 'JR. DOS DE
     MAYO #658', '06-30-1998', 'juan@gmail.com'),

     ('2023012890', 'COTRINA CONDOR','ANA', 'JR. 28 DE
     JULIO #100', '15-07-1994', 'miranda@gmail.com')
```

UPDATE

Esta sentencia permite modificar valores de los campos que se requiera.

Sintaxis:

```
UPDATE nombre_tabla
SET campoa='valor1',campo2='valor2',…
WHERE proposicion
```

Por ejemplo, para modificar la dirección del alumno "*COTRINA CONDOR*" cuyo código es "***2023012890***":

```
UPDATE alumno
SET direccion='JR. PUNO #4587',correo='ana@gmail.com'
WHERE codigo='2023012890'
```

DELETE

Esta sentencia borra una o más registro de una tabla.

Sintaxis:

```
DELETE FROM nombre_tabla WHERE proposicion
```

Para borrar al alumno "*COTRINA CONDOR*" cuyo código es "***2023012890***".

```
DELETE FROM alumno WHERE codigo='2023012890'
```

SELECT

Esta sentencia permite realizar consultas a los datos de las tablas de una base de datos.

Sintaxis básica:

```
SELECT [{ALL|DISTINCT}]
    column1, column2,…

FROM {<nombre_tabla1>|<nombre_vista1>}[,
    {<nombre_tabla2>|<nombre_vista2>}...]

[WHERE <proposicion1> [{AND|OR} <proposicion2>...]]

[GROUP BY <column1>[, <column2>...]]

[HAVING <proposicion1> [{AND|OR} <proposicion2>...]]

[ORDER BY {<column1>|<indice1>} [{ASC|DESC}][,
    {<column2>|<indice2>} [{ASC|DESC}]]];
```

SELECT	Con esta sentencia se seleccionan registros con columnas de uno o más tablas dependiendo de la condición del WHERE. el ALL anteponiendo a una columna indica que se seleccionaran todos los registros. Anteponer el DISTINCT a una columna seleccionara solo los valores distintos.
FROM	Aquí se indica la tabla o tablas de donde se obtendrán los datos
WHERE	Aquí se establece la condición o proposición según el cual se escogerán los registros, para generar proposiciones compuestas se utiliza los conectores lógicos AND o OR
GROUP BY	Con esto se agrupan registros de una tabla dependiendo de lo que se requiera con la columna establecida.
HAVING	Esto es similar al de WHERE, pero aplicado a los registros devueltos por la consulta. Se aplica junto a GROUP BY, la condición debe estar referida a las columnas contenidos en ella.
ORDER BY	Esto permite ordenar la consulta de forma ascendente(ASC) o descendente(DESC) por la columna que se indique.

EJEMPLO N°05-0001.

En la IMAGEN N°05-0001 se muestra parte de un modelo lógico de una base de datos de facturación con algunas tablas y sus relaciones entre ellos:

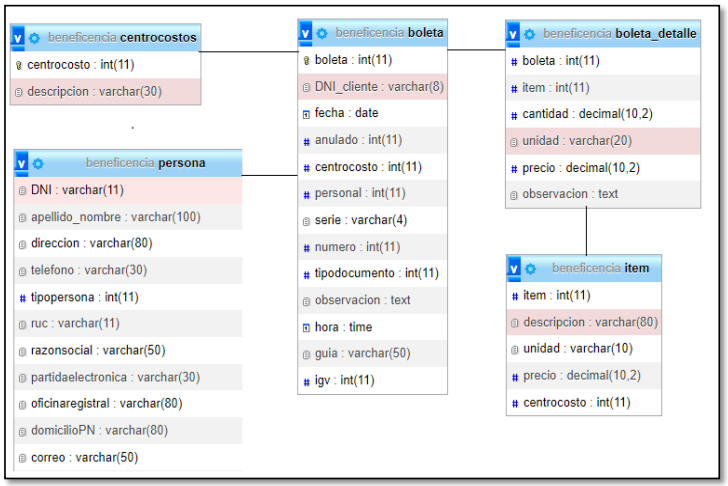


IMAGEN N°05-0001: Diagrama que muestra algunas tablas de una base de datos de facturación

a) Del diagrama mostrado en la IMAGEN N°05-0001, realizar una consulta para que me muestre los apellidos y nombres de los clientes con sus respectivas cantidades de boletas y ordenado por apellido.

b) Del diagrama mostrado en la IMAGEN N°05-0001, realizar una consulta para que me muestre los apellidos y nombres de los clientes con sus respectivas cantidades de ítems por boleta.

DESARROLLO

a) Para realizar la consulta requerida se consultará a la tabla ***bolete*** y a la tabla ***persona*** que están relacionado por la columna DNI, así mismo se agrupara por la columna DNI para contar la cantidad de boletas por persona.

Consulta:

```
SELECT p.apellido_nombre, COUNT(b.boleta) AS cantidad
FROM boleta AS b, persona AS p
WHERE b.DNI_cliente=p.DNI
GROUP BY b.DNI_cliente ORDER BY cantidad DESC
```

Resultado consulta:

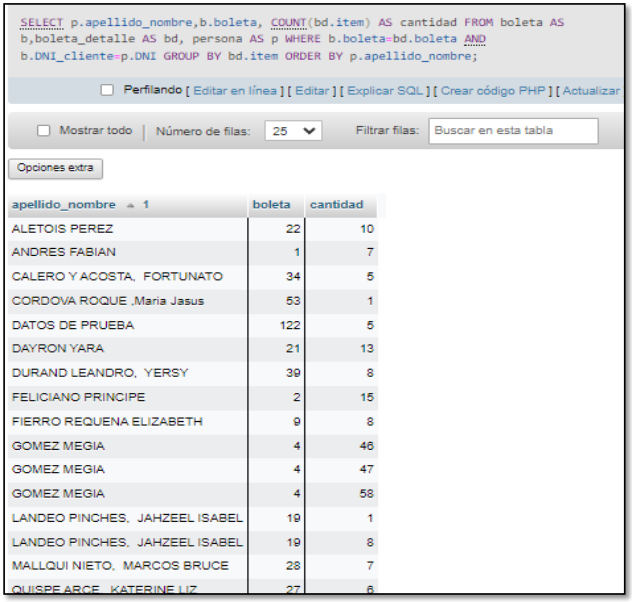
IMAGEN N°05-0002: Resultado de la consulta realizada

b) Para realizar la consulta requerida se consultará a la tabla ***bolete***, ***boleta_detalle*** y la tabla ***persona***, la tabla ***boleta*** está relacionado con la tabla ***persona*** por la columna **DNI**, la tabla ***boleta*** está relacionado con la tabla ***boeta_detalle*** por la columna ***boleta***, así mismo se agrupa por la columna ***boleta*** de la tabla ***boleta_detalle*** de la tabla ***boleta_detalle*** para contar la cantidad de ítems por boleta y se rodena por apellido

Consulta:

```
SELECT p.apellido_nombre,b.boleta, COUNT(bd.item) AS cantidad
FROM boleta AS b, boleta_detalle AS bd, persona AS p
WHERE b.boleta=bd.boleta AND b.DNI_cliente=p.DNI
GROUP BY bd.boleta ORDER BY p.apellido_nombre
```

Resultado consulta:



```
SELECT p.apellido_nombre,b.boleta, COUNT(bd.item) AS cantidad FROM boleta AS
b,boleta_detalle AS bd, persona AS p WHERE b.boleta=bd.boleta AND
b.DNI_cliente=p.DNI GROUP BY bd.item ORDER BY p.apellido_nombre;
```

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar

Mostrar todo | Número de filas: 25 Filtrar filas: Buscar en esta tabla

Opciones extra

apellido_nombre ▲ 1	boleta	cantidad
ALETOIS PEREZ	22	10
ANDRES FABIAN	1	7
CALERO Y ACOSTA, FORTUNATO	34	5
CORDOVA ROQUE ,Maria Jasus	53	1
DATOS DE PRUEBA	122	5
DAYRON YARA	21	13
DURAND LEANDRO, YERSY	39	8
FELICIANO PRINCIPE	2	15
FIERRO REQUENA ELIZABETH	9	8
GOMEZ MEGIA	4	46
GOMEZ MEGIA	4	47
GOMEZ MEGIA	4	58
LANDEO PINCHES, JAHZEEL ISABEL	19	1
LANDEO PINCHES, JAHZEEL ISABEL	19	8
MALLQUI NIETO, MARCOS BRUCE	28	7
QUISPE ARCE, KATERINE LIZ	27	6

IMAGEN N°05-0003: Resultado de la consulta realizada

5.4 DCL (LENGUAJE DE CONTROL DE DATOS)

Estas sentencias permiten controlar el acceso a los objetos de una base de datos, otorgando o denegando permisos a uno o más usuarios para realizar determinadas tareas.

Los comandos que se utilizan para ello son: **GRANT y REVOKE**

GRANT

Con este comando se otorga permisos

REVOKE

Con este comando se quita permisos que previamente se han concedido con el comando GRANT.

5.5 TCL(TRANSACTION CONTROL LANGUAGE)

Lenguaje de control de transacción en una base de datos, esto es; agrupar operaciones en un solo concepto, para indicar a la base de datos que haga todas las operaciones o ninguno. Para esto se utiliza estos comandos: COMMIT, ROLLBACK y SAVEPOINT.

COMMIT

Con este comando se graba todos los cambios en la base de datos, completando así la transacción. Es importante indicar el inicio de cada transacción.

ROLLBACK

Si hay algún inconveniente en la transacción se puede deshacer las operaciones o cambios realizados en la base de datos, para ello se utiliza este comando ROLLBACK.

SAVEPOINT

Este comando permite definir un punto de salvaguarda dentro de una transacción.

5.6 GESTOR DE BASE DE DATOS SQL SERVER

SQL Server es un sistema de gestión de base de datos relacional cuya función principal es la de almacenar y recuperar datos según lo requiera otras aplicaciones. Fue desarrollado por **Microsoft** por lo que su uso depende de una licencia, pero también se cuenta con versión gratuita SQLServer Express y SQLServer Developer, este último tiene todas las características que se puede usar como servidor de base de datos de desarrollo y pruebas en los entornos de no productivos.

Algunas características:

- ✓ Soporte de transacciones.
- ✓ Escalabilidad, estabilidad y seguridad.
- ✓ Soporte de procedimientos almacenados.
- ✓ Incluye también un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- ✓ Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y las terminales o clientes de la red solo acceden a la información.
- ✓ Permite administrar información de otros servidores de datos.
- ✓ Versiones de SQL Server.

Últimas versiones del SQLServer:

- ✓ Microsoft SQL Server 2022
- ✓ Microsoft SQL Server 2019 (64 bits)
- ✓ Microsoft SQL Server 2019 en Linux (64 bits)
- ✓ Microsoft SQL Server 2017 (64 bits)
- ✓ Microsoft SQL Server 2017 en Linux (64 bits)
- ✓ Microsoft SQL Server 2016 (64 bits)

✓ Microsoft SQL Server 2014 SP3 (64 bits)

Instalación y configuración de SQL Server 2022 Express

Para su instalación se descarga desde la página oficial de Microsoft:

https://www.microsoft.com/es-mx/sql-server/sql-server-downloads

Seleccionar la versión requerida:

IMAGEN N°05-0004: Descarga del instalador del SQLServer2022 Express

Una vez que se descargue se ejecuta el archivo de instalación.

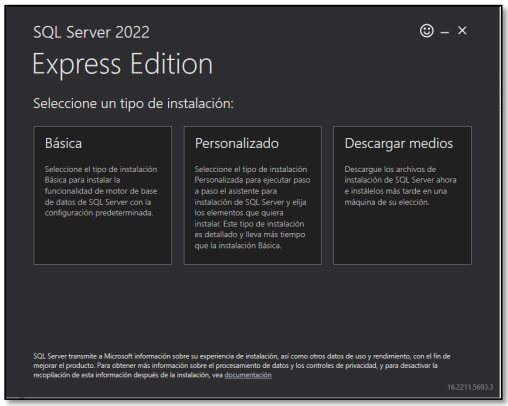


IMAGEN N°05-0005: Instalación de SQLServer2022 Express

Se elige la **Personalizado**. La instalación predeterminada está en inglés. Para la instalación en español, el Windows del sistema operativo se tiene que configurar a español.

IMAGEN N°05-0006: Instalación Personalizado de SQLServer2022 Express

Clic en si para continuar:

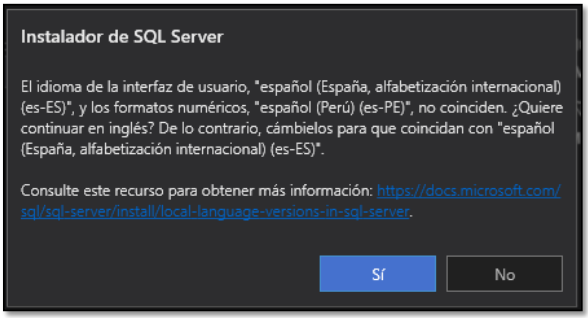


IMAGEN N°05-0007: Clic en Si para continuar

Clic en instalar:

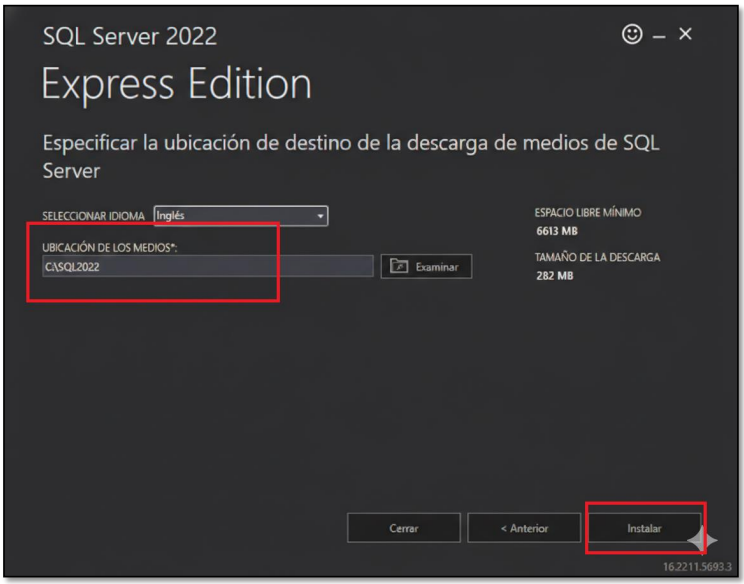


IMAGEN N°05-0008: Clic en instalar para continuar

El proceso de descarga de paquetes de instalación:

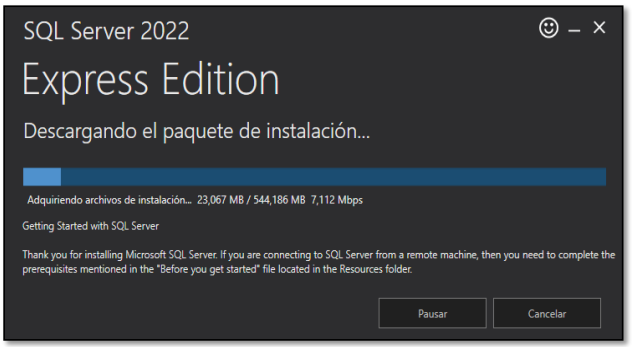


IMAGEN N°05-0009: En proceso de descarga paquetes de instalación

Una vez terminado la descarga de paquetes, saldrá la ventana mostrado en la FIGURA Nº 04-010, para poder instalar SQLServer2022, del cual seleccionamos New SQL Server, para instalar una nueva instancia.

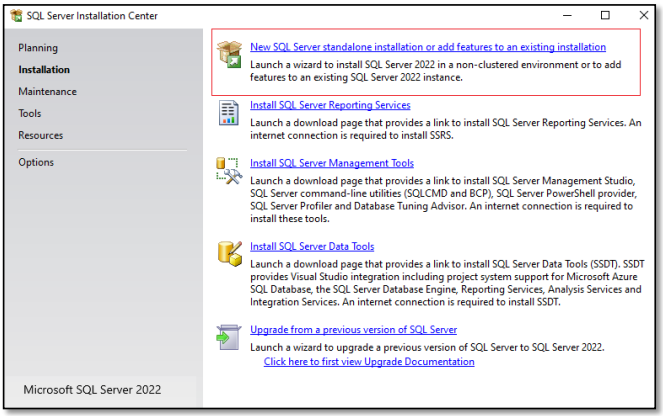


IMAGEN N°05-0010: instalación de una nueva instancia

Chec en aceptar los términos de la licencia y Next:

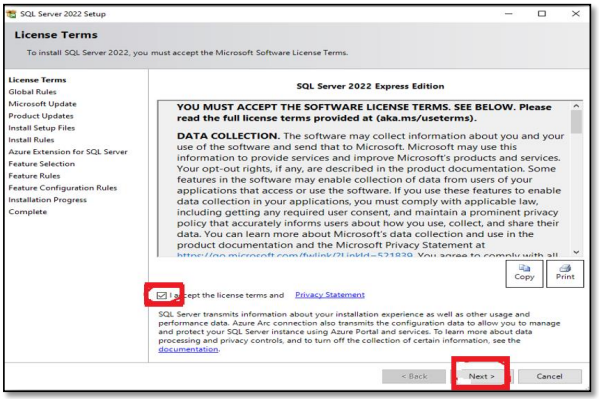


IMAGEN N°05-0011: Chec en aceptar licencia y Next

Deseleccionar Azure Extension for SQL Server:

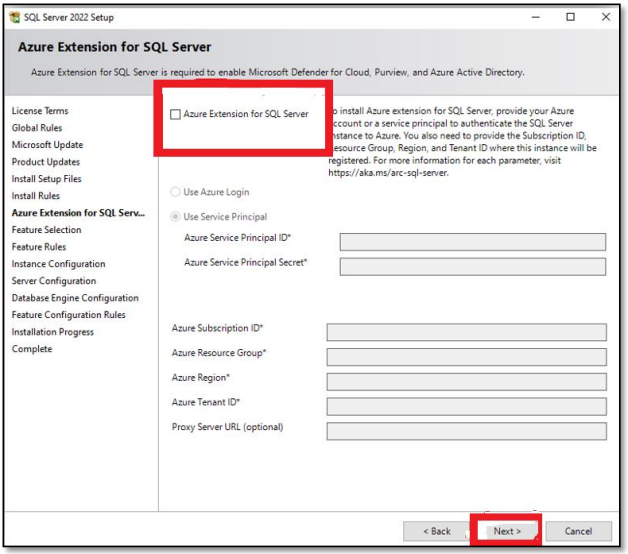


IMAGEN N°05-0012: Deseleccionar Azure Extension for SQL Server y Next

Clic en Next para proseguir con la instalación:

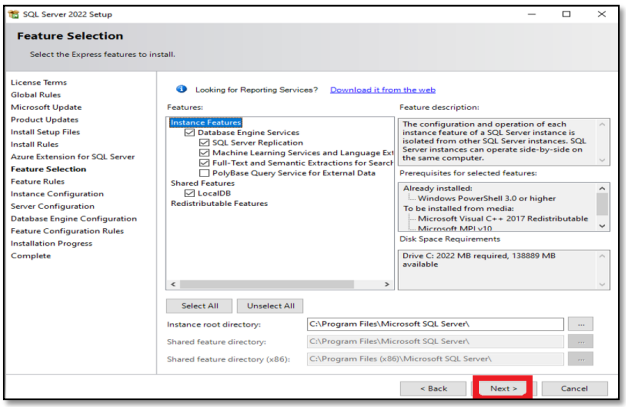


IMAGEN N°05-0013: Next para proseguir la instalación

Si se desea crear una instancia nueva seleccionar ***Named instance*** e ingresar el nombre, tal como se muestra en la IMAGEN N°05-0014:

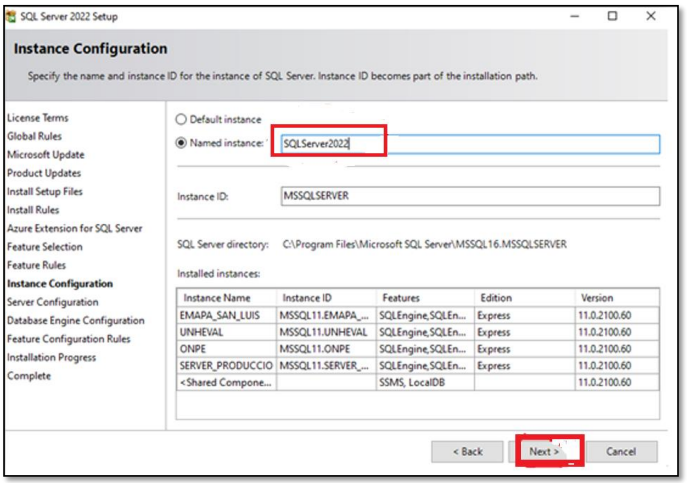


IMAGEN N°05-0014: Nombre de la nueva instancia

En la IMAGEN N°05-0014 se muestra la forma de Authentication, en este caso Mixed y se ingresa el password del súper usuario “sa”:

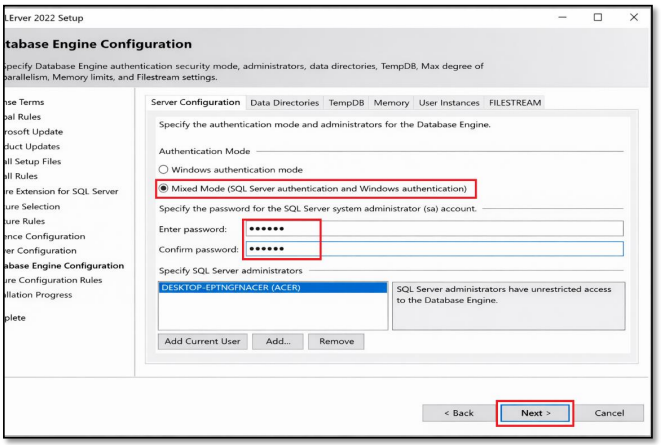


IMAGEN N°05-0015: Authentication Mixed y se ingresa clave del '***sa***'

Una vez terminado la instalación si todo está ok saldrá la ventana sugiriendo el reinicio de la computadora, así como se muestra en la IMAGEN N°05-0016:

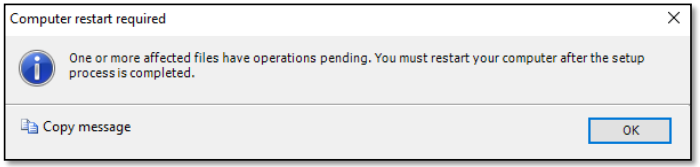


IMAGEN N°05-0016: Authentication Mixed y se ingresa clave del '***sa***'

Para ver la instancia de SQL Server 2022, en el inicio de Windows seleccionamos:

IMAGEN N°05-0017: En el menú de inicio de Window ya figura el SQL Server

En la figura IMAGEN N°05-0018 se puede apreciar que la nueva instancia instalada está corriendo: SQL Server (SQLSERVER2022)

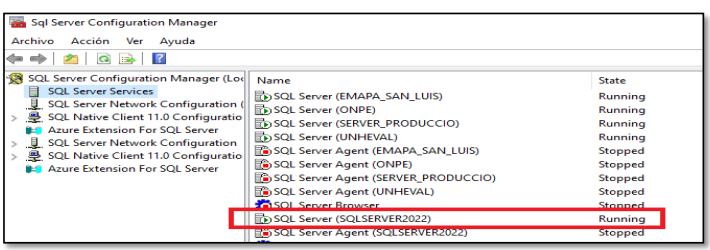


IMAGEN N°05-0018: La instancia SQLSERVER2022 está corriendo

Para poder tener un entorno de trabajo amigable y poder gestionar nuestra base de datos se necesita instalar el SQL Server Management Studio(SSMS)

"SQL Server Management Studio (SSMS) es un entorno integrado para administrar cualquier infraestructura de SQL, desde SQL Server hasta Azure SQL Database. SSMS proporciona herramientas para configurar, monitorear y administrar instancias de SQL Server y bases de datos. Use SSMS para implementar, monitorear y actualizar los componentes del nivel de datos que usan sus aplicaciones y crear consultas y scripts.

Use SSMS para consultar, diseñar y administrar sus bases de datos y almacenes de datos, donde sea que estén, en su computadora local o en la nube".

https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

Instalación de SQL Server Management Studio (SSMS)

Para su instalación se descarga desde la página de Microsoft:

https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

En la IMAGEN N°05-0019, se muestra el entorno de trabajo del SQL Server Management Studio, con las bases de datos creado por defecto:

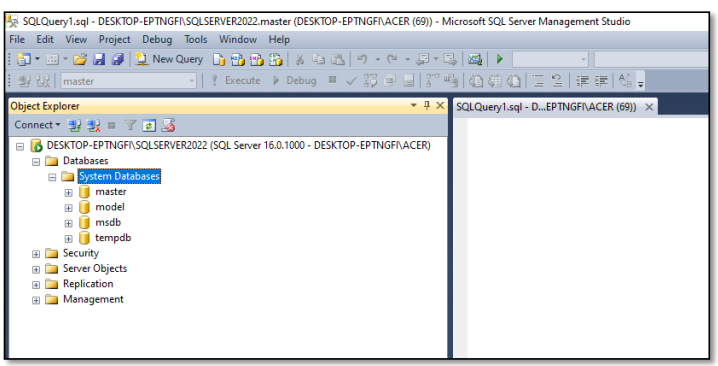


IMAGEN N°05-0019: Entorno de trabajo del SQL Server Management Studio

Para crear una base de datos, clic derecho sobre Databases, tal como se muestra en la IMAGEN N°05-0020:

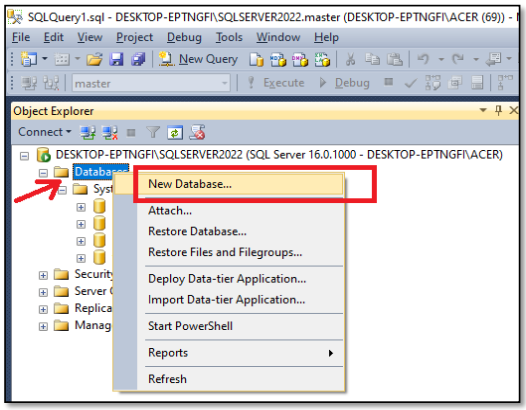


IMAGEN N°05-0020: Creación de una base de datos

En la ventana que aparece en la IMAGEN N°05-0021, se ingresa el nombre de la base de datos, en este caso: ***sis_logistica***

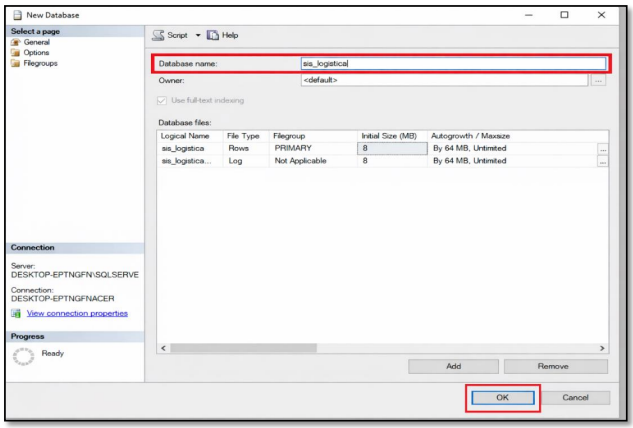


IMAGEN N°05-0021: Creación de la base de datos “sis_logistica

También se puede crear la base de datos utilizando sentencia SQL:
CREATE DATABASE sis_logistica, así como se muestra en IMAGEN N°05-0022

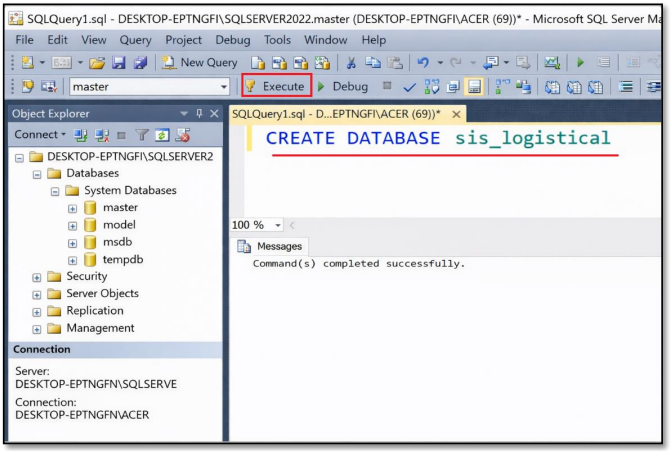


IMAGEN N°05-0022: Creación de base de datos, con sentencia SQL

EJEMPLO N°04-002

Se tiene el siguiente sistema para desarrollar:

En el proceso de "**Bienes y Suministros**" de la empresa de turismo "**Pata Amarilla**" que pertenece al área de Administración, se realizan las actividades de manera manual, por lo que se requiere sistematizar todo ello con la ayuda de un software, el dueño del proceso de **"Bienes y Suministros"** detalla el procedimiento y actividades de cada área:

- ✓ El proceso involucra cinco (5) áreas: Administración, Logística, Almacén, Patrimonio, Áreas usuarias.
- ✓ El área de Logística funciona de la siguiente forma:
- ➢ Recibe las solicitudes de requerimientos de bienes o suministros de las diferentes áreas de la empresa.
- ➢ Cada solicitud tiene un responsable que está adscrito a un área.

- Cada solicitud es autorizada por el jefe del área y posteriormente por el área de Administración.
- De la solicitud se requiere la siguiente información: Número de la solicitud (consecutivo), fecha, responsable, área, meta presupuestal. En cada solicitud se pueden contener uno o muchos ítems con la siguiente información: código, nombre del ítem, cantidad solicitada, unidad de medida, valor unitario.
- Cada ítem es identificado por un código único y es de carácter devolutivo (suministro) (útiles de escritorio, útiles de limpieza, etc.) o un bien (computadora, escritorio, etc.).
- La solicitud es enviada al área de Logística para atender si es que hay en stock o realizar la compra a uno o varios proveedores. Para realizar la compra se juntan todas las solicitudes para tener la cantidad total de ítems de cada rubro a comprar.
- Las cotizaciones son realizadas con uno o varios proveedores de los bienes solicitados.
- Para de un producto, primero se realiza la cotización a dos o tres proveedores para determinar la mejor oferta en calidad y precio que ofrecen, una vez elegido el proveedor se genera la orden de compra, con los siguientes datos: número de **orden de compra**, nombre del proveedor al cual se le va a realizar la compra, fecha de la orden, monto total de la orden, fecha de entrega. Cada orden puede tener asociado uno o varios ítems, cada ítem debe tener los siguientes datos: código del ítem, nombre del ítem, cantidad de compra, unidad de medida del ítem, valor unitario y sub total.
- La orden de compra es aprobada por el área de Administración luego el área de Logística envía o hace entrega al proveedor elegido.

✓ El área de Almacén funciona de la siguiente forma:

- Recepciona los bienes que llegan de los proveedores y distribuye con una pecosa a las áreas que realizaron las solicitudes.
- El proveedor hace entrega a Almacén los ítems detallado en la Orden de Compra, para ello adjunta la guía de remisión y factura para su pago correspondiente si todo está conforme, se registra una entrada de almacén por cada factura relacionada, con los siguientes datos: número de entrada,

fecha, número de factura, Proveedor, total bienes, valor total, los ítems recibidos con la cantidad respectiva.

- El área de Almacén hace entrega de los bienes a las diferentes áreas solicitantes atreves de una pecosa detallando cada ítem entregado y otros datos más como el numero de salida, empleado solicitante del ítem o los ítems, cantidad, fecha de salida, fecha de entrega.

✓ El área Patrimonio es responsable de:

- Administrar y controlar la ubicación de los bienes dentro de la empresa, por esto antes de que el bien salga del Almacén es codificado y se genera su estiker que se pega al bien.
- Cada trabajador tiene a su cargo una o varios bienes, esto permite controlar su ubicación.

Diseñar las tablas en SQL Server, para poder almacenar los datos de que responda al sistema de "**Bienes y Suministros**" de la empresa de turismo "**Pata Amarilla**".

DESARROLLO

Para el sistema de acuerdo a los requerimientos y descripción de datos se puede considerar las siguientes tablas:

- Área
- Trabajador
- cargo
- Contrato
- Ítem
- Categoría
- Unidad_medida
- Solicitud
- Metas_presupuestal
- Orden_compra
- proveedor

Relacionando las tablas se diseña el modelo lógico mostrado en la IMAGEN N° 05-023:

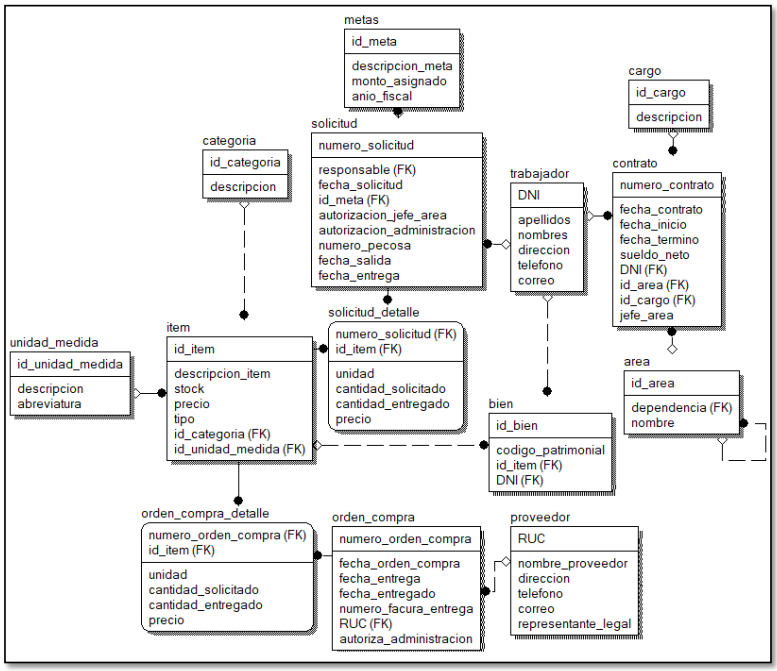


IMAGEN N°05-0023: Modelo lógico de la base de datos logística

Script de creación de las tablas en SQL Server:

```
CREATE TABLE area
(
  id_area          int  NOT NULL ,
  dependencia        int  NULL ,
  nombre           varchar(80)  NULL
)
go
```

```
ALTER TABLE area
  ADD CONSTRAINT XPKarea PRIMARY KEY  NONCLUSTERED (id_area
ASC)
go


CREATE TABLE bien
(
  id_bien             char(18)  NOT NULL ,
  codigo_patrimonial   char(18)  NULL ,
  id_item             char(10)  NULL ,
  DNI                 char(8)  NULL
)
go


ALTER TABLE bien
  ADD CONSTRAINT XPKbien PRIMARY KEY  NONCLUSTERED (id_bien
ASC)
go


CREATE TABLE cargo
(
  id_cargo            int  NOT NULL ,
  descripcion          varchar(80)  NULL
)
go


ALTER TABLE cargo
  ADD CONSTRAINT XPKcargo PRIMARY KEY  NONCLUSTERED (id_cargo
ASC)
go


CREATE TABLE categoria
```

```
(
  id_categoria        int  NOT NULL ,
  descripcion         varchar(100)  NULL
)
go


ALTER TABLE categoria
  ADD CONSTRAINT XPKcategoria PRIMARY KEY  NONCLUSTERED
(id_categoria ASC)
go


CREATE TABLE contrato
(
  numero_contrato      char(10)  NOT NULL ,
  fecha_contrato       datetime  NULL ,
  fecha_inicio         datetime  NULL ,
  fecha_termino        datetime  NULL ,
  sueldo_neto          FLOAT  NULL ,
  DNI                  char(8)  NULL ,
  id_area              int  NULL ,
  id_cargo             int  NULL ,
  jefe_area            int  NULL
)
go




ALTER TABLE contrato
  ADD CONSTRAINT XPKcontrato PRIMARY KEY  NONCLUSTERED
(numero_contrato ASC)
```

```
go

CREATE TABLE item
(
 id_item            char(10)  NOT NULL ,
 descripcion_item    varchar(80)  NULL ,
 stock              FLOAT  NULL ,
 precio             FLOAT  NULL ,
 tipo               CHAR(1)  NULL ,
 id_categoria       int  NULL ,
 id_unidad_medida    int  NULL
)
go

ALTER TABLE item
 ADD CONSTRAINT XPKitem PRIMARY KEY  NONCLUSTERED (id_item
ASC)
go

CREATE TABLE metas
(
 id_meta            int  NOT NULL ,
 descripcion_meta    varchar(80)  NULL ,
 monto_asignado      FLOAT  NULL ,
 anio_fiscal        int  NULL
)
go

ALTER TABLE metas
 ADD CONSTRAINT XPKpetas PRIMARY KEY  NONCLUSTERED (id_meta
ASC)
```

```
go

CREATE TABLE orden_compra
(
  numero_orden_compra  char(10)  NOT NULL ,
  fecha_orden_compra   datetime  NULL ,
  fecha_entrega        datetime  NULL ,
  RUC                  char(11)  NULL ,
  autoriza_administracion int  NULL ,
  fecha_entregado      char(18)  NULL ,
  numero_facura_entrega char(18)  NULL
)
go

ALTER TABLE orden_compra
  ADD CONSTRAINT XPKorden_compra PRIMARY KEY  NONCLUSTERED
(numero_orden_compra ASC)
go

CREATE TABLE orden_compra_detalle
(
  unidad               char(18)  NULL ,
  cantidad_solicitado  char(18)  NULL ,
  precio               char(18)  NULL ,
  numero_orden_compra  char(10)  NOT NULL ,
  id_item              char(10)  NOT NULL ,
  cantidad_entregado   char(18)  NULL
)
go

ALTER TABLE orden_compra_detalle
```

```
  ADD CONSTRAINT XPKorden_compra_detalle PRIMARY KEY
NONCLUSTERED (numero_orden_compra ASC,id_item ASC)
go


CREATE TABLE proveedor
(
  RUC                 char(11)  NOT NULL ,
  nombre_proveedor    varchar(80)  NULL ,
  direccion           varchar(80)  NULL ,
  telefono            varchar(30)  NULL ,
  correo              varchar(30)  NULL ,
  representante_legal  varchar(100)  NULL
)
go


ALTER TABLE proveedor
  ADD CONSTRAINT XPKproveedor PRIMARY KEY  NONCLUSTERED (RUC
ASC)
go


CREATE TABLE solicitud
(
  numero_solicitud    char(18)  NOT NULL ,
  responsable         char(8)  NULL ,
  fecha_solicitud     char(18)  NULL ,
  id_meta             int  NULL ,
  autorizacion_jefe_area int  NULL ,
  autorizacion_administracion int  NULL ,
  numero_pecosa       char(18)  NULL ,
  fecha_salida        char(18)  NULL ,
  fecha_entrega       char(18)  NULL
```

```
)
go

ALTER TABLE solicitud
  ADD CONSTRAINT XPKsolicitud PRIMARY KEY  NONCLUSTERED
(numero_solicitud ASC)
go

CREATE TABLE solicitud_detalle
(
  unidad              char(18)  NULL ,
  cantidad_solicitado  char(18)  NULL ,
  numero_solicitud     char(18)  NOT NULL ,
  id_item             char(10)  NOT NULL ,
  precio              char(18)  NULL ,
  cantidad_entregado   char(18)  NULL
)
go

ALTER TABLE solicitud_detalle
  ADD CONSTRAINT XPKsolicitud_detalle PRIMARY KEY  NONCLUSTERED
(numero_solicitud ASC,id_item ASC)
go

CREATE TABLE trabajador
(
  DNI                char(8)  NOT NULL ,
  apellidos          varchar(50)  NULL ,
  nombres             varchar(50)  NULL ,
  direccion          varchar(50)  NULL ,
  telefono           varchar(30)  NULL ,
```

```
  correo              varchar(30)  NULL
)
go


ALTER TABLE trabajador
  ADD CONSTRAINT XPKtrabajador PRIMARY KEY  NONCLUSTERED (DNI
ASC)
go


CREATE TABLE unidad_medida
(
  id_unidad_medida     int  NOT NULL ,
  descripcion          VARCHAR(80)  NULL ,
  abreviatura          VARCHAR(50)  NULL
)
go


ALTER TABLE unidad_medida
  ADD CONSTRAINT XPKunidad_medida PRIMARY KEY  NONCLUSTERED
(id_unidad_medida ASC)
go


ALTER TABLE area
  ADD CONSTRAINT id_area_dependencia FOREIGN KEY (dependencia)
REFERENCES area(id_area)
          ON DELETE NO ACTION
          ON UPDATE NO ACTION
go


ALTER TABLE bien
  ADD CONSTRAINT R_16 FOREIGN KEY (id_item) REFERENCES
item(id_item)
```

```
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE bien
  ADD CONSTRAINT R_17 FOREIGN KEY (DNI) REFERENCES
trabajador(DNI)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE contrato
  ADD CONSTRAINT R_3 FOREIGN KEY (DNI) REFERENCES trabajador(DNI)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE contrato
  ADD CONSTRAINT R_4 FOREIGN KEY (id_area) REFERENCES
area(id_area)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE contrato
  ADD CONSTRAINT R_5 FOREIGN KEY (id_cargo) REFERENCES
cargo(id_cargo)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE item
```

```
  ADD CONSTRAINT R_6 FOREIGN KEY (id_categoria) REFERENCES
categoria(id_categoria)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE item
  ADD CONSTRAINT R_18 FOREIGN KEY (id_unidad_medida) REFERENCES
unidad_medida(id_unidad_medida)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE orden_compra
  ADD CONSTRAINT R_15 FOREIGN KEY (RUC) REFERENCES
proveedor(RUC)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE orden_compra_detalle
  ADD CONSTRAINT R_13 FOREIGN KEY (numero_orden_compra)
REFERENCES orden_compra(numero_orden_compra)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE orden_compra_detalle
  ADD CONSTRAINT R_14 FOREIGN KEY (id_item) REFERENCES
item(id_item)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
```

```
go

ALTER TABLE solicitud
  ADD CONSTRAINT R_7 FOREIGN KEY (responsable) REFERENCES
trabajador(DNI)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE solicitud
  ADD CONSTRAINT R_8 FOREIGN KEY (id_meta) REFERENCES
metas(id_meta)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE solicitud_detalle
  ADD CONSTRAINT R_11 FOREIGN KEY (numero_solicitud) REFERENCES
solicitud(numero_solicitud)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go

ALTER TABLE solicitud_detalle
  ADD CONSTRAINT R_12 FOREIGN KEY (id_item) REFERENCES
item(id_item)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
go
```

Se ejecuta en la ventana de consultas del SQL Server Management Studio, y se tendría todas las tablas creadas en SQL Server, tal como se muestra en la IMAGEN N°05-0024:

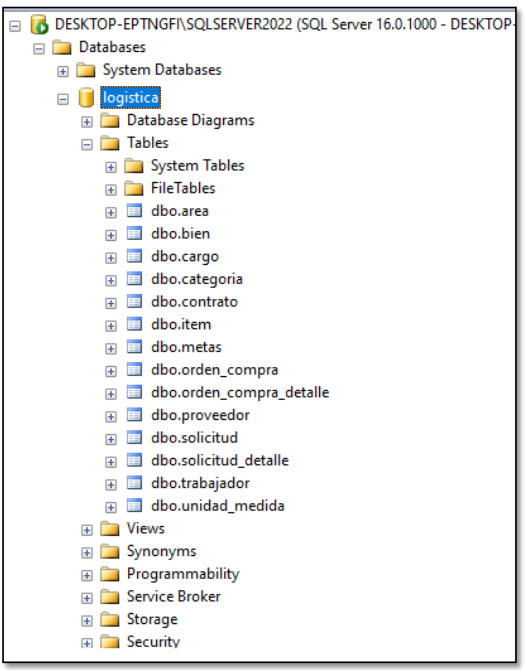


IMAGEN N°05-0024: Tablas de la base de datos **logistica** en SQL Server

5.7 GESTOR DE BASE DE DATOS MYSQL Y MARIADB

MySQL y MariaDB, son dos gestores de base de datos muy parecido de código abierto. Funcionan con un modelo cliente-servidor, son multiplataforma, por lo que se puede instalar en Windows, Linux entre otros sistemas operativos.

Algunas características de MySQL:

✓ Sirve para crear y manejar de bases de datos relacionales

- ✓ Implementa varios motores de almacenamiento con características y velocidades distintas.
- ✓ Software libre y gratuito para el uso en su versión de la comunidad
- ✓ Dispone de una arquitectura cliente / servidor y la instalación proporciona tanto el programa de cliente (por línea de comandos) como el del servidor (sistema gestor de la base de datos en sí)
- ✓ Sistema ligero, fácil de usar, fácil de mantener.
- ✓ Es robusto y seguro.

Instalacion de MySQL

Para la instalación de MySQL, se descarga desde su página oficial:

https://www.mysql.com/

IMAGEN N°05-0025: Página oficial de MySQL

Otra alternativa para instalar MariaDB o MySQL con: Xampp, AppServ, WampServer que también a su vez instala Apache + PHP + MySQL o MariaDB. En la IMAGEN N°05-0025 se muestra la página oficial de descarga del instalador del XAMPP:

https://www.apachefriends.org/es/index.html

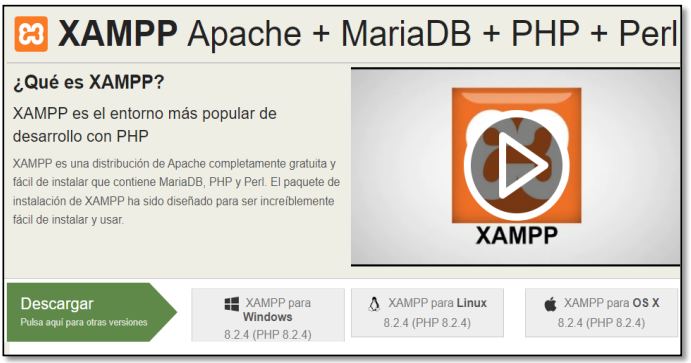


IMAGEN N°05-0026: Página oficial para descargar el instalador de XAMPP

Una vez instalado el XAMPP con su panel de control se pone en marcha los servidores Apache y MySQL tal como se muestra en la IMAGEN N°05-0027:

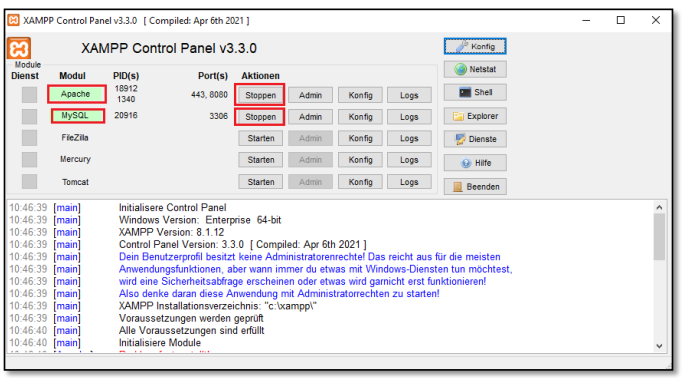


IMAGEN N°05-0027: Panel de control del XAMPP

Para tener un entorno de trabajo más amigable en modo gráfico con MySQL o MariaDB, se podría utilizar el **MySQL Workbench** o el **phpMyAdmin** en entorno web. En la FIGURA Nº 04-027 tenemos la interfaz del phpMyAdmin, que se accede poniendo en la barra de direcciones de cualquier navegador web: localhost:8080/phpmyadmin (el 8080 depende del puerto del servidor web)

IMAGEN N°05-0028: la Interfaz del phpMyAdmin

EJEMPLO N°04-003

Del EJEMPLO N°04-002, crear la base de datos logística y todas sus tablas en MySQL.

DESARROLLO

El modelo lógico vendría a ser lo mismo mostrado en la FIGURA Nº 04-022

Script de creación de las tablas en MySQL:

```
CREATE TABLE area
(
  id_area            INTEGER NOT NULL,
  dependencia        INTEGER NULL,
  nombre             varchar(80) NULL
);

ALTER TABLE area
ADD PRIMARY KEY (id_area);

CREATE TABLE bien
(
  id_bien            CHAR(18) NOT NULL,
  codigo_patrimonial   char(18) NULL,
```

```
  id_item          char(10) NULL,
  DNI              char(8) NULL
);

ALTER TABLE bien
ADD PRIMARY KEY (id_bien);

CREATE TABLE cargo
(
  id_cargo         INTEGER NOT NULL,
  descripcion      varchar(80) NULL
);

ALTER TABLE cargo
ADD PRIMARY KEY (id_cargo);

CREATE TABLE categoria
(
  id_categoria     INTEGER NOT NULL,
  descripcion      varchar(100) NULL
);

ALTER TABLE categoria
ADD PRIMARY KEY (id_categoria);

CREATE TABLE contrato
(
  numero_contrato    char(10) NOT NULL,
  fecha_contrato     datetime NULL,
  fecha_inicio       datetime NULL,
  fecha_termino      datetime NULL,
```

```
  sueldo_neto          FLOAT NULL,
  DNI                  char(8) NULL,
  id_area              INTEGER NULL,
  id_cargo             INTEGER NULL,
  jefe_area            INTEGER NULL
);

ALTER TABLE contrato
ADD PRIMARY KEY (numero_contrato);


CREATE TABLE item
(
  id_item              char(10) NOT NULL,
  descripcion_item     varchar(80) NULL,
  stock                FLOAT NULL,
  precio               FLOAT NULL,
  tipo                 CHAR(1) NULL,
  id_categoria         INTEGER NULL,
  id_unidad_medida     INTEGER NULL
);

ALTER TABLE item
ADD PRIMARY KEY (id_item);

CREATE TABLE metas
(
  id_meta              INTEGER NOT NULL,
  descripcion_meta     varchar(80) NULL,
  monto_asignado       FLOAT NULL,
  anio_fiscal          INTEGER NULL
```

```
);

ALTER TABLE metas
ADD PRIMARY KEY (id_meta);

CREATE TABLE orden_compra
(
  numero_orden_compra  char(10) NOT NULL,
  fecha_orden_compra   datetime NULL,
  fecha_entrega        datetime NULL,
  RUC                  char(11) NULL,
  autoriza_administracion INTEGER NULL,
  fecha_entregado      CHAR(18) NULL,
  numero_facura_entrega CHAR(18) NULL
);

ALTER TABLE orden_compra
ADD PRIMARY KEY (numero_orden_compra);

CREATE TABLE orden_compra_detalle
(
  unidad               CHAR(18) NULL,
  cantidad_solicitado  CHAR(18) NULL,
  precio               CHAR(18) NULL,
  numero_orden_compra  char(10) NOT NULL,
  id_item              char(10) NOT NULL,
  cantidad_entregado   CHAR(18) NULL
);

ALTER TABLE orden_compra_detalle
ADD PRIMARY KEY (numero_orden_compra,id_item);
```

```
CREATE TABLE proveedor
(
  RUC                 char(11) NOT NULL,
  nombre_proveedor    varchar(80) NULL,
  direccion           varchar(80) NULL,
  telefono            varchar(30) NULL,
  correo              varchar(30) NULL,
  representante_legal varchar(100) NULL
);

ALTER TABLE proveedor
ADD PRIMARY KEY (RUC);

CREATE TABLE solicitud
(
  numero_solicitud    CHAR(18) NOT NULL,
  responsable         char(8) NULL,
  fecha_solicitud     CHAR(18) NULL,
  id_meta             INTEGER NULL,
  autorizacion_jefe_area INTEGER NULL,
  autorizacion_administracion INTEGER NULL,
  numero_pecosa       CHAR(18) NULL,
  fecha_salida        CHAR(18) NULL,
  fecha_entrega       CHAR(18) NULL
);

ALTER TABLE solicitud
ADD PRIMARY KEY (numero_solicitud);

CREATE TABLE solicitud_detalle
```

```
(
  unidad              CHAR(18) NULL,
  cantidad_solicitado  CHAR(18) NULL,
  numero_solicitud     CHAR(18) NOT NULL,
  id_item             char(10) NOT NULL,
  precio              CHAR(18) NULL,
  cantidad_entregado   CHAR(18) NULL
);

ALTER TABLE solicitud_detalle
ADD PRIMARY KEY (numero_solicitud,id_item);

CREATE TABLE trabajador
(
  DNI                char(8) NOT NULL,
  apellidos           varchar(50) NULL,
  nombres             varchar(50) NULL,
  direccion           varchar(50) NULL,
  telefono            varchar(30) NULL,
  correo              varchar(30) NULL
);

ALTER TABLE trabajador
ADD PRIMARY KEY (DNI);


CREATE TABLE unidad_medida
(
  id_unidad_medida     INTEGER NOT NULL,
  descripcion          VARCHAR(80) NULL,
  abreviatura          VARCHAR(50) NULL
```

```
);

ALTER TABLE unidad_medida
ADD PRIMARY KEY (id_unidad_medida);

ALTER TABLE area
ADD FOREIGN KEY id_area_dependencia (dependencia) REFERENCES area
(id_area);

ALTER TABLE bien
ADD FOREIGN KEY R_16 (id_item) REFERENCES item (id_item);

ALTER TABLE bien
ADD FOREIGN KEY R_17 (DNI) REFERENCES trabajador (DNI);

ALTER TABLE contrato
ADD FOREIGN KEY R_3 (DNI) REFERENCES trabajador (DNI);

ALTER TABLE contrato
ADD FOREIGN KEY R_4 (id_area) REFERENCES area (id_area);

ALTER TABLE contrato
ADD FOREIGN KEY R_5 (id_cargo) REFERENCES cargo (id_cargo);

ALTER TABLE item
ADD FOREIGN KEY R_6 (id_categoria) REFERENCES categoria (id_categoria);

ALTER TABLE item
ADD FOREIGN KEY R_18 (id_unidad_medida) REFERENCES unidad_medida
(id_unidad_medida);
```

```
ALTER TABLE orden_compra
ADD FOREIGN KEY R_15 (RUC) REFERENCES proveedor (RUC);

ALTER TABLE orden_compra_detalle
ADD FOREIGN KEY R_13 (numero_orden_compra) REFERENCES
orden_compra (numero_orden_compra);

ALTER TABLE orden_compra_detalle
ADD FOREIGN KEY R_14 (id_item) REFERENCES item (id_item);

ALTER TABLE solicitud
ADD FOREIGN KEY R_7 (responsable) REFERENCES trabajador (DNI);

ALTER TABLE solicitud
ADD FOREIGN KEY R_8 (id_meta) REFERENCES metas (id_meta);

ALTER TABLE solicitud_detalle
ADD FOREIGN KEY R_11 (numero_solicitud) REFERENCES solicitud
(numero_solicitud);

ALTER TABLE solicitud_detalle
ADD FOREIGN KEY R_12 (id_item) REFERENCES item (id_item);
```

Se ejecuta en la ventana de consultas del SQL phpMyAdmin, y se tendría todas las tablas creadas en MySQL, tal como se muestra en la IMAGEN N°05-0029:

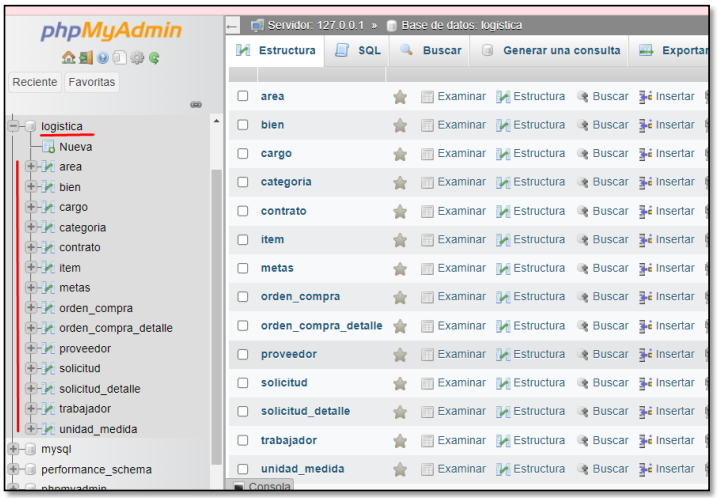


IMAGEN N°05-0029: Tablas de la base de datos logistica creadas en MySQL

Poblado y consultas SQL a la Base de datos

Insertando datos a algunas tablas:

Tabla **unidad_medida**

```
INSERT INTO unidad_medida (id_unidad_medida, descripcion, abreviatura)
VALUES
(1, 'ACCION', 'ACCION'),
(2, 'ACERVO', 'ACERVO'),
(52, 'EXPEDIENTE PROCESADO', 'EXPEDIEN'),
(53, 'EXPEDIENTE RESUELTO', 'EXPEDIEN'),
(54, 'EXPEDIENTE TECNICO', 'EXPEDIEN'),
(55, 'EXPEDIENTE TRAMITADO', 'EXPEDIEN'),
(56, 'FAMILIA', 'FAMILIA'),
(57, 'FOLLETO', 'FOLLETO'),
(67, 'KILOMETRO', 'KILOMETR'),
```

(68, 'LICENCIA', 'LICENCIA'),

(69, 'M2', 'M2'),

(70, 'M3', 'M3'),

(72, 'MAPA', 'MAPA'),

(73, 'METRO', 'METRO'),

(74, 'METRO LUZ', 'METRO LU'),

(126, 'PAQUETE ESCOLAR', 'PAQUETE '),

(127, 'PLANTAS', 'PLANTAS'),

(128, 'KILOGRAMO', 'KLG'),

(129, 'DIQUE', 'DIQUE'),

(162, 'NUEVOS SOLES', 'NUEVOS SOLES'),

(163, 'DOLARES', 'DOLARES'),

(164, 'LT./SEG.', 'LT./SEG.'),

(172, 'GLOBAL', 'GLOBAL'),

(173, 'CASOS RESUELTOS', 'CASOS RESUELTOS'),

(174, 'NORMAS APROBADAS', 'NORMAS APROBADA'),

(175, 'APELACIONES RESUELTAS', 'APELACIONES RES'),

(176, 'PASAJEROS/DIA', 'PASAJEROS/DIA'),

(177, 'TRANSFERENCIA', 'TRANSFERENCIA'),

(178, 'MEDIDAS CAUTELARES', 'MEDIDAS CAUTELA'),

(181, 'HORAS', 'HORAS'),

(182, 'LOCAL', 'LOCAL'),

(183, 'ACCIONES DE AUDITORIA', 'ACCIONES DE AUD'),

(185, 'CRIADERO', 'CRIADERO'),

(186, 'KW', 'KW'),

(187, 'ASPERSORES', 'ASPERSORES'),

(188, 'AUTORIDADES', 'AUTORIDADES'),

(189, 'TELEFONOS EN ZONAS RURALES', 'TELEFONOS EN Z'),

(190, 'NUMERO', 'NUMERO'),

(191, 'PERSONA PROTEGIDA', 'PERSONA PROTEGI'),

(192, 'CATALOGO', 'CATALOGO'),

(193, 'QUINTAL', 'QUINTAL');

Tabla **categoría:**

```
INSERT INTO categoria (id_categoria, descripcion) VALUES
(1, 'MATERIAL DE GUERRA'),
(2, 'OBRAS RELACIONADAS CON LA AGRICULTURA'),
(3, 'ACONDICIONAMIENTO'),
(4, 'ABRASIVOS'),
(5, 'OBRAS RELACIONADAS CON EL TRANSPORTE'),
(6, 'SERVICIOS RELACIONADOS CON  LA ENERGIA NUCLEARES'),
(7, 'OBRAS RELACIONADAS CON LA ENERG?A Y MINER?A'),
(8, 'ALAMBRES, BARRAS, PLANCHAS, PERFILES Y SIMILARES DE METAL'),
(9, 'ALIMENTACI?N'),
(10, 'AGR?COLA Y PESQUERO'),
(11, 'OBRAS RELACIONADAS CON EL TURISMO'),
(12, 'OBRAS RELACIONADAS CON LA VIVIENDA Y CONSTRUCCI?N'),
(13, 'SERVICIOS RELACIONADOS CON ALOJAMIENTOS Y ASISTENCIA
PARA VIAJES'),
(14, 'AGROPECUARIA, GANADER?A  Y JARDINER?A  :  REPUESTOS,
ACCESORIOS Y MATERIALES'),
(15, 'AIRE ACONDICIONADO Y REFRIGERACI?N  :  REPUESTOS Y
ACCESORIOS'),
(16, 'SERVICIO DE ASEO Y LIMPIEZA'),
(17, 'OBRAS RELACIONADAS CON LAS COMUNICACIONES'),
(18, 'OBRAS RELACIONADAS CON LA ZOOLOGIA'),
(19, 'ASESOR?A, CONSULTOR?A E INTERVENCIONES ESPECIALIZADAS'),
(20, 'AISLANTES T?RMICOS Y AC?STICOS, REFRACTARIOS, FILTROS Y
PURIFICADORES'),
(21, 'CONTRATACI?N ADMINISTRATIVA DE SERVICIOS - CAS'),
(22, 'SERVICIOS NO PERSONALES'),
(23, 'ALIMENTOS PARA ANIMALES'),
(24, 'SERVICIOS PRESTADOS POR TERCEROS'),
```

```
(25, 'ALIMENTOS Y BEBIDAS PARA PERSONAS'),
(26, 'ATENCIONES Y CELEBRACIONES'),
(27, 'ARMAMENTO, MUNIC.. Y EXPLOS. : RPTOS, ACCESORIOS Y MATERIALES'),
(28, 'AIRE ACONDICIONADO Y REFRIGERACI?N'),
(29, 'SERVICIOS RELACIONADOS CON LAS CONSTRUCCIONES.'),
(30, 'SERVICIOS RELACIONADOS CON EL MEDIO AMBIENTE Y SANEAMIENTO'),
(31, 'EQUIPOS PARA ARTES GR?FICAS E IMPRESIONES : REPUESTOS Y ACCESORIOS'),
(32, 'ASEO, LIMPIEZA Y TOCADOR : REPUESTOS, ACCESORIOS, ?TILES Y MATERIALES'),
(33, 'SERVICIOS RELACIONADOS CON FINANZAS E INVERSIONES Y SIMILARES'),
(34, 'BIENES DE ACTIVO FIJO NO CATALOGADOS POR SBN');
```

Tabla **ítem:**

```
INSERT INTO item (id_item, descripcion_item, stock, precio, tipo, id_categoria, id_unidad_medida) VALUES
('1', 'PIEDRA ESMERIL CIRCULAR 2 in X 10 in GRANO GRUESO', 0, 0, 'B', 2, 112),
('10', 'LIJA CIRCULAR PARA FIERRO N? 24', 0, 0, 'B', 2, 112),
('100', 'ACERO REDONDO ST37 ? 5/8 in', 0, 0, 'B', 3, 73),
('1000', 'CAFE INSTANTANEO X 7 G', 0, 0, 'B', 9, 358),
('1001', 'AVENA X 200 G X 24 UNI', 0, 0, 'B', 9, 335),
('1002', 'LENTEJA X 20 KG', 0, 0, 'B', 9, 378),
('1003', 'ARROZ CORRIENTE X 48 Kg', 0, 0, 'B', 9, 378),
('1004', 'FRIJOL OJO DE PALOMA', 0, 0, 'B', 9, 128),
('1005', 'FREJOL DE SOYA', 0, 0, 'B', 9, 128),
('1006', 'KIWICHA ENTERA TOSTADA', 0, 0, 'B', 9, 128),
('1007', 'CEBADA TOSTADA', 0, 0, 'B', 9, 128),
('1008', 'MAIZENA', 0, 0, 'B', 9, 128),
```

('1009', 'CREMA DE CHOCLO X 5 kg', 0, 0, 'B', 9, 112),

('101', 'ACERO REDONDO TREFILADO SAE 1020 ? 1/4', 0, 0, 'B', 3, 73),

('1010', 'CEBADA', 0, 0, 'B', 9, 128),

('1011', 'CREMA DE HABAS CALIDAD SUPERIOR', 0, 0, 'B', 9, 128),

('1012', 'HARINA DE TRIGO X 50 KG', 0, 0, 'B', 9, 378),

('1013', 'SEMOLA X 20 UNIDADES DE 250 GRS C/U', 0, 0, 'B', 9, 352),

('1014', 'HARINA PREPARADA X KILO X 12 UNIDADES', 0, 0, 'B', 9, 352),

('1015', 'HOJUELA DE KIWICHA ', 0, 0, 'B', 9, 128),

('1016', 'CREMA DE ALVEJA', 0, 0, 'B', 9, 128),

('1017', 'HOJUELA DE TRIGO ', 0, 0, 'B', 9, 128),

('1018', 'HOJUELA DE QUINUA', 0, 0, 'B', 9, 128),

('1019', 'MAIZENA IMPORTADA DE 25 KG', 0, 0, 'B', 9, 335),

('102', 'ACERO REDONDO SAE 4340 (VCN 150) ? 30MM', 0, 0, 'B', 3, 73),

('1020', 'MAIZENA DE 500 G', 0, 0, 'B', 9, 337),

('1021', 'HOJUELA DE CEBADA', 0, 0, 'B', 9, 128),

('1022', 'HARINA DE HABAS 1 KG X 12 UNI', 0, 0, 'B', 9, 352),

('1023', 'HOJUELA DE QUINUA 1 KG X 12 UNI', 0, 0, 'B', 9, 352),

('1024', 'CREMA DE ESPARRAGOS X 70 g', 0, 0, 'B', 9, 112),

('1025', 'CREMA DE CHAMPI?ONES', 0, 0, 'B', 9, 128),

('1026', 'CREMA DE ESPARRAGOS', 0, 0, 'B', 9, 128),

('1027', 'QUINUA PERLADA x 50 kg', 0, 0, 'B', 9, 112),

('1028', 'TRIGO MORON x 25 kg', 0, 0, 'B', 9, 112),

('1029', 'HARINA PASTELERA', 0, 0, 'B', 9, 128),

('103', 'ACERO SAE 4140 O CVL 140 REDONDO 81/2 in X 2 MT', 0, 0, 'B', 3, 112),

('1030', 'MAIZENA X 100 G X 96 UNI', 0, 0, 'B', 9, 352),

('1031', 'HOJUELA DE AVENA', 0, 0, 'B', 9, 128),

('1032', 'HOJUELA DE CEREALES', 0, 0, 'B', 9, 128),

('1033', 'HOJUELA DE AVENA X 250 g', 0, 0, 'B', 9, 112),

('1034', 'HARINA DE PLATANO', 0, 0, 'B', 9, 128),

('1035', 'HARINA DE MAIZ', 0, 0, 'B', 9, 128),

('1036', 'CREMA DE HABAS', 0, 0, 'B', 9, 128),

('1037', 'HOJUELA DE QUINUA AVENA', 0, 0, 'B', 9, 128),

('1038', 'HARINA DE TRIGO FORTIFICADO', 0, 0, 'B', 9, 128),

('1039', 'CREMA DE HABAS LACTEADA', 0, 0, 'B', 9, 128),

('104', 'ACERO SAE 4140 O VCL 140 REDONDO 11 in X 2 MTS', 0, 0, 'B', 3, 112),

('1040', 'HOJUELA DE AVENA X 225 g', 0, 0, 'B', 9, 112),

('1041', 'HOJUELA DE AVENA X 450 g', 0, 0, 'B', 9, 112),

('1042', 'HOJUELA DE AVENA ENRIQUECIDA', 0, 0, 'B', 9, 128),

('1043', 'HOJUELA DE AVENA X 10 kg', 0, 0, 'B', 9, 112),

('1044', 'HOJUELA DE AVENA ENRIQUECIDA X 500 g', 0, 0, 'B', 9, 112),

('1045', 'HOJUELA DE CEBADA ENRIQUECIDA', 0, 0, 'B', 9, 128),

('1046', 'HOJUELA DE CEREAL ENRIQUECIDA', 0, 0, 'B', 9, 128),

('1190', 'AGUA MINERAL ENVASE NO RETORNABLE CON GAS X 500 ML', 0, 0, 'B', 9, 112),

('1191', 'AGUA MINERAL DE PLASTICO NO RETORNABLE C/GAS X 2 L', 0, 0, 'B', 9, 336),

('1192', 'AGUA MINERAL DE PLASTICO RETORNABLE X 20 L', 0, 0, 'B', 9, 333),

('1193', 'AGUA MINERAL ENVASE NO RETORNABLE S/GAS X 500 ML', 0, 0, 'B', 9, 112),

('1194', 'AGUA MINERAL VIDRIO RETORNABLE C/GAS X 600 ML', 0, 0, 'B', 9, 336),

('1195', 'AGUA MINERAL VIDRIO RETORNABLE S/GAS X 600 ML', 0, 0, 'B', 9, 336),

('1196', 'AGUA MINERAL DE PLASTICO RETORNABLE X 19 L', 0, 0, 'B', 9, 333),

('1197', 'AGUA MINERAL CAJA X 20 L', 0, 0, 'B', 9, 337),

('1198', 'AGUA MINERAL BOTELLA SIN GAS X 02 L X 06 UNI', 0, 0, 'B', 9, 352),

('1199', 'AGUA MINERAL S/GAS EN VASO X 500 ML', 0, 0, 'B', 9, 112),

('12', 'LIJA PARA METAL N? 100 X 50 UNI', 0, 0, 'B', 2, 112),

('120', 'PLANCHA DE FIERRO DE 90 cm X 30 cm X 1.5 mm', 0, 0, 'B', 3, 112),

('1200', 'AGUA MINERAL S/GAS X 500 ML', 0, 0, 'B', 9, 112),

```
('1201', 'AGUA MINERAL C/GAS X 500 ML', 0, 0, 'B', 9, 112),
('1202', 'AGUA MINERAL BOTELLA CON GAS X 02 L X 06 UNI', 0, 0, 'B', 9,
352),
('1203', 'AGUA MINERAL CON GAS EN ENVASE NO RETORNABLE X 625 ML', 0, 0,
'B', 9, 336);
```

EJEMPLO N°04-004

Del base de datos de "logistica" diseñando en el EJEMPLO N°04-003 hacer las siguientes consultas:

1. Lista de trabajadores con su respectiva área de trabajo
2. Cantidad de trabajadores por área
3. Lista de trabajadores por área con su respectivo cargo
4. Cantidad de solicitudes por área
5. Cantidad de órdenes de compra por proveedor
6. Lista de categoría con la cantidad de ítem que pertenecen a dicha categoría

DESARROLLO

1. Lista de trabajadores con su respectiva área de trabajo
Para esta consulta se:

Para esta consulta, del modelo lógico mostrado en la FIGURA Nº 04-022, se tiene que consultar a las tablas "***contrato***", "***trabajador***", "***area***"; como la tabla "***contrato***" está relacionado con "***trabajador***" por la columna común ***DNI*** entonces en la cláusula ***WHERE*** se tiene que poner esa relación. Así mismo la tabla "***contrato***" está relacionado con "***area***" por la columna común "***id_area***" también en la cláusula ***WHERE*** se tiene poner esa relación.

```
SELECT t.apellidos,t.nombres, a.nombre AS descripcion_area
FROM contrato AS c,trabajador AS t, area AS a
WHERE c.DNI=t.DNI and c.id_area=a.id_area
ORDER BY t.apellidos ASC
```

2. Cantidad de trabajadores por área
Para esta consulta, del modelo lógico mostrado en la FIGURA Nº 04-022, se tiene que consultar a las tablas "***contrato***" y "***area***"; como la tabla "***contrato***" está relacionado con la tabla "***area***" por la columna común "***id_area***" en la cláusula ***WHERE*** se tiene poner esa relación.

```
SELECT a.nombre AS descripcion_area,COUNT(c.DNI) AS cantidad
FROM contrato AS c, area AS a
WHERE c.id_area=a.id_area
ORDER BY a.nombre ASC
```

3. Lista de trabajadores por área con su respectivo cargo

Para esta consulta, del modelo lógico mostrado en la FIGURA Nº 04-022, se tiene que consultar a las tablas "***contrato***", "***trabajador***", "***area***" y "**cargo**"; como la tabla "***contrato***" está relacionado con "***trabajador***" por la columna común ***DNI*** entonces en la cláusula ***WHERE*** se tiene que poner esa relación. Así mismo la tabla "***contrato***" está relacionado con "***area***" por la columna común "***id_area***" también en la cláusula ***WHERE*** se tiene que poner esa relación. También la tabla "**cargo**" está relacionado con "***contrato***" por la columna "***id_cargo***", por lo que se tiene que poner en la cláusula **WHERE** esa relación**.**

```
SELECT t.apellidos,t.nombres,
  a.nombre AS descripcion_area,
  ca.descripcion
FROM contrato AS c,trabajador AS t, area AS a,cargo AS ca
WHERE c.DNI=t.DNI
    AND c.id_area=a.id_area
    AND c.id_cargo=ca.id_cargo
ORDER BY t.apellidos ASC
```

4. Cantidad de solicitudes por área

Para esta consulta, del modelo lógico mostrado en la FIGURA Nº 04-022, se tiene que consultar a las tablas "***solicitud***", "***trabajador***", "***contrato***" y "**area**"; como la tabla "***solicitud***" está relacionado con "***trabajador***" por la columna responsable y ***DNI*** respectivamente entonces en la cláusula ***WHERE*** se pone esa relación. Así mismo la tabla "***trabajador***" está relacionado con "***contrato***" por la columna común "***DNI***" en la cláusula ***WHERE*** se pone esa relación. También la tabla "**contrato**" está relacionado con "***area***" por la columna "***id_area***", por lo que se pone en la cláusula **WHERE** esa relación**.**

```
SELECT a.nombre, COUNT(s.numero_solicitud) AS cant
FROM solicitud AS s,trabajador AS t,
   contrato AS c,area AS a
```

```
WHERE s.responsable=t.DNI AND t.DNI=c.DNI AND c.id_area=a.id_area
GROUP BY c.id_area
ORDER BY a.nombre
```

5. Cantidad de órdenes de compra por proveedor

```
SELECT p.nombre_proveedor,
COUNT(oc.numero_orden_compra) AS cant
FROM proveedor AS p, orden_compra AS oc
WHERE p.RUC=oc.RUC
GROUP BY p.RUC
ORDER BY p.nombre_proveedor
```

6. Lista de categoría con la cantidad de ítem que pertenecen a dicha categoría

```
SELECT c.descripcion, COUNT(i.id_categoria)
FROM item AS i, categoria AS c
WHERE i.id_categoria=c.id_categoria
GROUP BY i.id_categoria;
```

El resultado de la consulta se muestra en la figura:

	descripcion	count(i.id_categoria)
Editar Copiar Borrar	OBRAS RELACIONADAS CON LA AGRICULTURA	21
Editar Copiar Borrar	ACONDICIONAMIENTO	351
Editar Copiar Borrar	ABRASIVOS	17
Editar Copiar Borrar	OBRAS RELACIONADAS CON EL TRANSPORTE	36
Editar Copiar Borrar	SERVICIOS RELACIONADOS CON LA ENERGIA NUCLEARES	32
Editar Copiar Borrar	OBRAS RELACIONADAS CON LA ENERG?A Y MINER?A	294
Editar Copiar Borrar	ALAMBRES, BARRAS, PLANCHAS, PERFILES Y SIMILARES D...	35
Editar Copiar Borrar	ALIMENTACI?N	1625
Editar Copiar Borrar	AGR?COLA Y PESQUERO	4
Editar Copiar Borrar	OBRAS RELACIONADAS CON EL TURISMO	175
Editar Copiar Borrar	OBRAS RELACIONADAS CON LA VIVIENDA Y CONSTRUCCI?N	8
Editar Copiar Borrar	SERVICIOS RELACIONADOS CON ALOJAMIENTOS Y ASISTENC...	1056

IMAGEN N°05-0030: Tablas de la base de datos logistica creadas en MySQL

CAPÍTULO 6. INTRODUCCIÓN A CONSULTAS SQL, PROCEDIMIENTOS ALMACENADOS, FUNCIONES Y VISTAS EN MYSQL Y SQL SERVER

6.1 CONSULTAS SQL AVANZADAS Y OPTIMIZACIÓN

Esta sección es fundamental para cualquier desarrollador que trabaje con grandes volúmenes de datos, ya que se enfoca en técnicas para manejar lógica compleja, mejorar el rendimiento de las consultas y obtener información analítica detallada directamente en la capa SQL.

6.1.1 Cláusulas Fundamentales de Agregación y Ordenamiento (Integración)

Aunque son cláusulas básicas, su correcta aplicación es crucial para las consultas avanzadas.

Agrupamiento de Resultados (GROUP BY)

- ✓ Se utiliza para agrupar filas con valores idénticos en una o más columnas, permitiendo aplicar funciones de agregación (**como** SUM, AVG, COUNT) a cada grupo.
- ✓ **Filtrado de Grupos (HAVING):** Mientras que **WHERE** filtra filas individuales *antes* de agrupar, HAVING filtra los grupos *después* de que se han aplicado las funciones de agregación.

EJEMPLO N°06-001

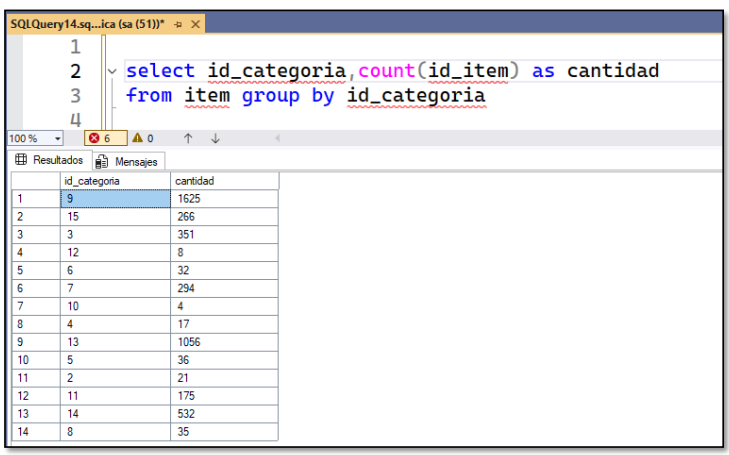
Del modelo y base de datos del EJEMPLO N°04-002 del Capítulo V, base de datos de logística, listar la cantidad de ítems por categoría.

DESARROLLO

En el ejemplo y diseño se puede visualizar la relación entre ítem y categoría, un ítem pertenece a una categoría específica, se consulta a la tabla ítem agrupando por categoría, la consulta en SQLServer seria así:

```
select id_categoria,count(id_item) as cantidad from item group by
id_categoria
```

Resultado consulta:



	id_categoria	cantidad
1	9	1625
2	15	266
3	3	351
4	12	8
5	6	32
6	7	294
7	10	4
8	4	17
9	13	1056
10	5	36
11	2	21
12	11	175
13	14	532
14	8	35

IMAGEN N°06-0001: Resultado de la consulta realizada en SQLServer

Ordenamiento de Resultados (ORDER BY)

Determina la secuencia final en la que se presentan las filas (ascendente ASC por defecto, o descendente DESC). Es esencial en reportes para mostrar el "Top N" o la secuencia temporal correcta.

EJEMPLO N°06-002

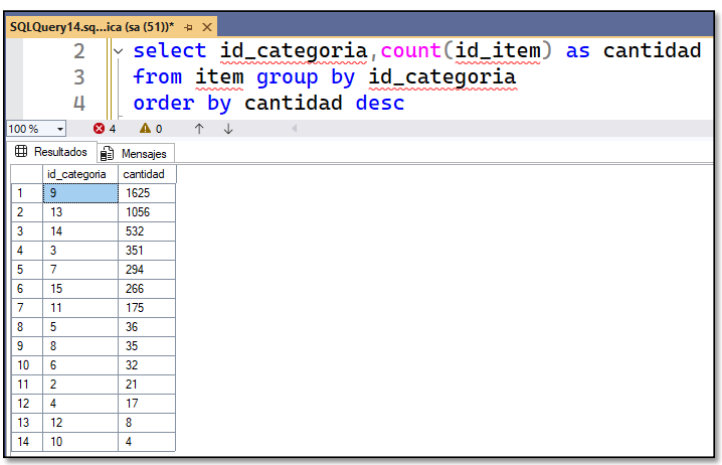
Del modelo y base de datos del EJEMPLO N°04-002 del Capítulo V, base de datos de logística, listar la cantidad de ítems por categoría, además que se ordene en forma descendente las categorías que tengan mayor cantidad de ítems a menor cantidad.

DESARROLLO

En el ejemplo y diseño se puede visualizar la relación entre ítem y categoría, un ítem pertenece a una categoría específica, se consulta a la tabla ítem agrupando por categoría y ordenando con ORDER BY, la consulta en SQLServer seria así:

```
select id_categoria,count(id_item) as cantidad
from item group by id_categoria
order by cantidad desc
```

Resultado consulta:



	id_categoria	cantidad
1	9	1625
2	13	1056
3	14	532
4	3	351
5	7	294
6	15	266
7	11	175
8	5	36
9	8	35
10	6	32
11	2	21
12	4	17
13	12	8
14	10	4

IMAGEN N°06-0002: Resultado de la consulta realizada en SQLServer

6.1.2 Subconsultas y Tablas Derivadas

Las subconsultas (o consultas anidadas) permiten construir consultas modulares y complejas, donde el resultado de una consulta se utiliza como entrada para otra.

Subconsultas en SELECT, FROM y WHERE

- ✓ **En SELECT:** Se usan para recuperar un único valor escalar que se aplica a cada fila del resultado principal (ej. obtener el promedio de ventas total junto a cada venta individual).
- ✓ **En FROM (Tablas Derivadas):** La subconsulta actúa como una tabla temporal durante la ejecución. Esto es útil para aplicar lógica de negocio compleja o cálculos intermedios antes de unir los resultados.
- ✓ **En WHERE:** Compara un valor con el conjunto de resultados devuelto por la subconsulta.

EJEMPLO N°06-003

En los ejemplos anteriores se observa que en los resultados solo sale la columna id_categoria en código de categoría, para entender mejor el resultado se requiere que salga descripción de la categoría: id_categoria, nombre_categoria, cantidad_item

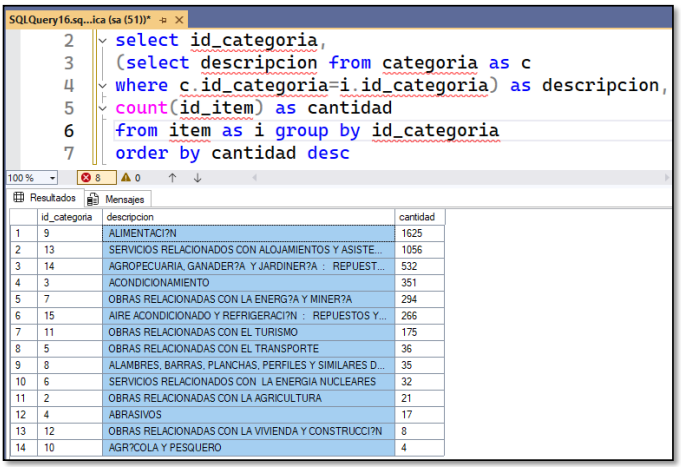
DESARROLLO

En el ejemplo y diseño se puede visualizar la relación entre ítem y categoría, un ítem pertenece a una categoría específica, se consulta a la tabla ***ítem*** agrupando por categoría y ordenando con ORDER BY, adicional a ello se realiza una subconsulta a la tabla categoría, para extraer la descripción, la consulta en SQLServer seria así:

```
select id_categoria,
(select descripcion from categoria as c
where c.id_categoria=i.id_categoria) as descripcion,
count(id_item) as cantidad
```

```
from item as i group by id_categoria
order by cantidad desc
```

Resultado consulta se muestra en la IMAGEN N°06-0003



	id_categoria	descripcion	cantidad
1	9	ALIMENTACI?N	1625
2	13	SERVICIOS RELACIONADOS CON ALOJAMIENTOS Y ASISTE...	1056
3	14	AGROPECUARIA, GANADER?A Y JARDINER?A : REPUEST...	532
4	3	ACONDICIONAMIENTO	351
5	7	OBRAS RELACIONADAS CON LA ENERG?A Y MINER?A	294
6	15	AIRE ACONDICIONADO Y REFRIGERACI?N : REPUESTOS Y...	266
7	11	OBRAS RELACIONADAS CON EL TURISMO	175
8	5	OBRAS RELACIONADAS CON EL TRANSPORTE	36
9	8	ALAMBRES, BARRAS, PLANCHAS, PERFILES Y SIMILARES D...	35
10	6	SERVICIOS RELACIONADOS CON LA ENERGIA NUCLEARES	32
11	2	OBRAS RELACIONADAS CON LA AGRICULTURA	21
12	4	ABRASIVOS	17
13	12	OBRAS RELACIONADAS CON LA VIVIENDA Y CONSTRUCCI?N	8
14	10	AGR?COLA Y PESQUERO	4

IMAGEN N°06-0003: Resultado de la consulta realizada en SQLServer

Operadores de existencia y pertenencia: IN, EXISTS, ANY **y** ALL

- ✓ **IN:** Verifica si un valor está presente en la lista de valores devuelta por la subconsulta.
- ✓ **EXISTS:** Es un operador booleano que verifica si la subconsulta devuelve **alguna** fila. Es a menudo más eficiente que IN para grandes conjuntos de datos.

EJEMPLO N°06-004

Del modelo y base de datos del EJEMPLO N°04-002 del Capítulo V, base de datos de logística, listar los ítems que pertenecen a las categorías ' ALIMENTACIÓN' o ' ANIMALES'.

DESARROLLO

En el ejemplo y diseño se puede visualizar la relación entre ítem y categoría, un ítem pertenece a una categoría específica, se consulta a la tabla ***ítem*** agrupando por categoría y ordenando con **ORDER BY**, para unir las tablas, en **FROM** mencionamos todas las tablas involucradas, y en **WHERE** se relaciona las dos tablas, para extraer la descripción de la tabla “categoria”, la consulta en SQLServer seria así:

```
SELECT
   i.descripcion_item,
   c.descripcion AS Categoria
FROM
   item as i,categoria as c
WHERE i.id_categoria=c.id_categoria
   and
   c.descripcion IN ('ALIMENTACIÓN', 'ANIMALES');
```

Resultado consulta se muestra en la IMAGEN N°06-0004:

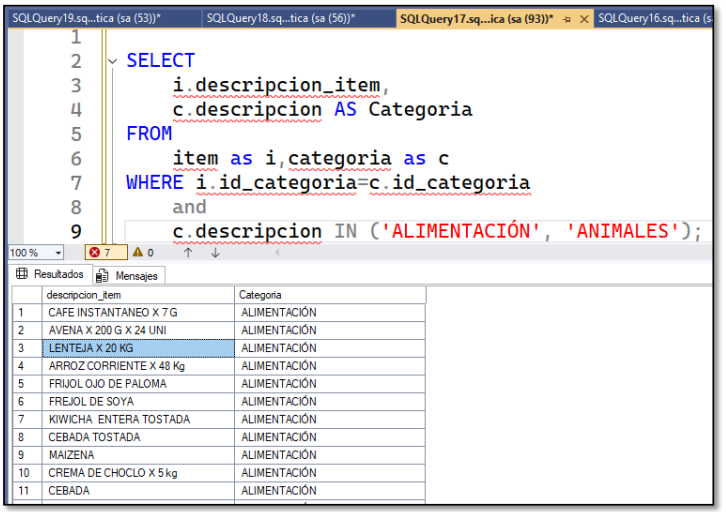


IMAGEN N°06-0004: Resultado de la consulta realizada en SQLServer

EJEMPLO N°06-005

Del modelo y base de datos del EJEMPLO N°04-002 del Capítulo V, base de datos de logística, Listar todas las unidades de medida que están actualmente en uso por algún ítem.

DESARROLLO

En el ejemplo y diseño se puede visualizar la relación entre ítem y unidad_medida, un ítem pertenece a una unidad_medida específica, se consulta a la tabla ***ítem*** y en WHERE se utiliza **EXISTS**, la consulta seria así:

```
SELECT
    um.descripcion,
    um.abreviatura
```

```
FROM
    unidad_medida um
WHERE
    EXISTS (
        SELECT 1 -- Solo necesitamos verificar la existencia, no el
valor
        FROM item i
        WHERE i.id_unidad_medida = um.id_unidad_medida
    );
```

Resultado consulta se muestra en la IMAGEN N°06-0006:

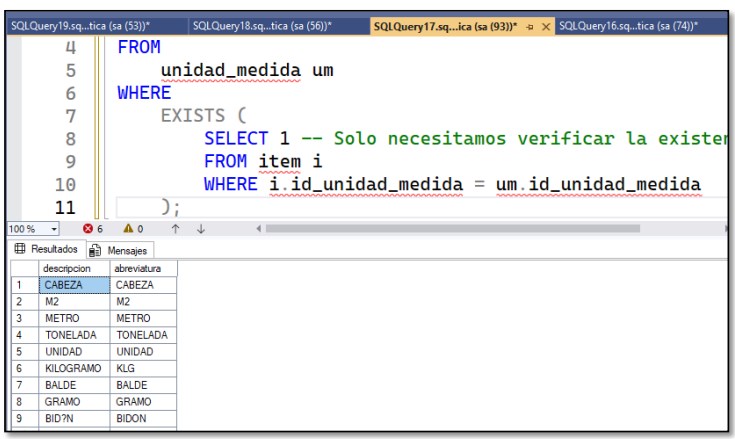


IMAGEN N°06-0006: Resultado de la consulta realizada en SQLServer

EJEMPLO N°06-006

Del modelo y base de datos del EJEMPLO N°04-002 del Capítulo V, base de datos de logística, listar todos los ítems cuyo precio sea **mayor que el precio más bajo** de cualquier ítem en la categoria con id_categoria = 2.

DESARROLLO

La consulta seria así:

```
SELECT
   descripcion_item,
   precio
FROM
   item
WHERE
   precio > ANY (
      SELECT precio
      FROM item
      WHERE id_categoria = 2
   )
ORDER BY precio;
```

6.1.3 Joins Especializados

Más allá de los INNER y LEFT JOIN básicos, estos *joins* resuelven problemas de relación de datos más sofisticados.

Self-Join (Auto-unión) para manejo de jerarquías

Consiste en unir una tabla consigo misma, utilizando alias de tabla. Es la técnica estándar para modelar relaciones jerárquicas (ej. encontrar el nombre del jefe de un empleado en la misma tabla de empleados).

Full Outer Join y técnicas de simulación en MySQL

- ✓ Devuelve todas las filas de **ambas** tablas unidas, emparejando donde es posible y usando NULL donde no hay coincidencia.
- ✓ **SQL Server** lo implementa directamente. **MySQL** generalmente requiere simularlo combinando un LEFT JOIN y un RIGHT JOIN con el operador UNION.

EJEMPLO N°06-007

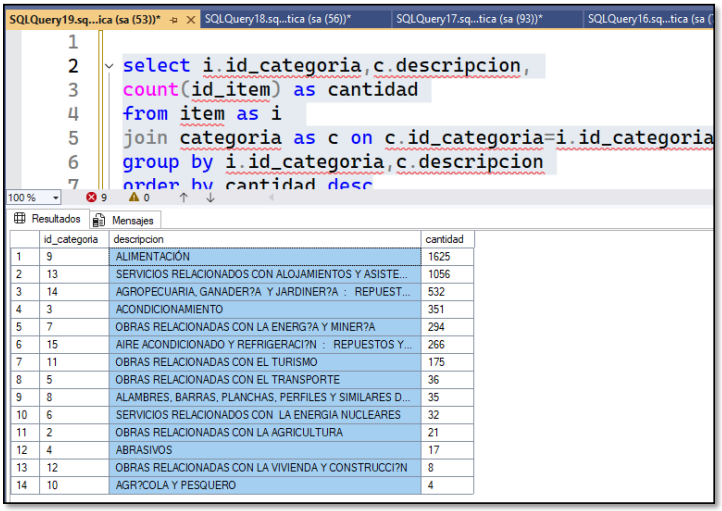
Del esquema de los ejemplos anteriores se observa que en los resultados solo sale la columna id_categoria en código de categoría, para entender mejor el resultado se requiere que salga descripción de la categoría: id_categoria, nombre_categoria, cantidad_item. Se unira las tablas con **Join.**

DESARROLLO

En el ejemplo y diseño se puede visualizar la relación entre ítem y categoría, un ítem pertenece a una categoría específica, se consulta a la tabla ***ítem*** agrupando por categoría y ordenando con **ORDER BY**, adicional a ello se relaciona a la tabla categoría con **JOIN**, para extraer la descripción, la consulta en SQLServer seria así:

```
select i.id_categoria,c.descripcion,
count(id_item) as cantidad
from item as i
join categoria as c on c.id_categoria=i.id_categoria
group by i.id_categoria,c.descripcion
order by cantidad desc
```

Resultado consulta de la consulta se muestra en la IMAGEN N°06-0007:



	id_categoria	descripcion	cantidad
1	9	ALIMENTACIÓN	1625
2	13	SERVICIOS RELACIONADOS CON ALOJAMIENTOS Y ASISTE...	1056
3	14	AGROPECUARIA, GANADER?A Y JARDINER?A : REPUEST...	532
4	3	ACONDICIONAMIENTO	351
5	7	OBRAS RELACIONADAS CON LA ENERG?A Y MINER?A	294
6	15	AIRE ACONDICIONADO Y REFRIGERACI?N : REPUESTOS Y...	266
7	11	OBRAS RELACIONADAS CON EL TURISMO	175
8	5	OBRAS RELACIONADAS CON EL TRANSPORTE	36
9	8	ALAMBRES, BARRAS, PLANCHAS, PERFILES Y SIMILARES D...	35
10	6	SERVICIOS RELACIONADOS CON LA ENERGIA NUCLEARES	32
11	2	OBRAS RELACIONADAS CON LA AGRICULTURA	21
12	4	ABRASIVOS	17
13	12	OBRAS RELACIONADAS CON LA VIVIENDA Y CONSTRUCCI?N	8
14	10	AGR?COLA Y PESQUERO	4

IMAGEN N°06-0007: Resultado de la consulta realizada en SQLServer

6.1.4 Funciones de Ventana (Window Functions)

Estas son las herramientas analíticas más potentes de SQL. Realizan cálculos sobre un conjunto de filas relacionadas (la ventana), pero **sin colapsar las filas de resultado**, permitiendo que el detalle de la fila original se mantenga junto al resultado analítico.

Concepto, sintaxis y la cláusula OVER()

- ✓ **OVER():** Define la "ventana" o grupo de filas. Es lo que diferencia estas funciones de las funciones de agregación normales.
- ✓ **PARTITION BY:** Divide las filas en grupos lógicos (ej. por departamento o por año), y la función se aplica independientemente en cada grupo.

Funciones de Ranking

- ✓ **ROW_NUMBER():** Asigna un número secuencial único a cada fila dentro de la partición.
- ✓ **RANK():** Asigna el mismo rango a valores empatados, saltando el siguiente número.
- ✓ **DENSE_RANK():** Asigna el mismo rango a valores empatados, **sin saltar** el siguiente número.

Funciones de Desplazamiento

- ✓ **LAG():** Accede al valor de una fila anterior (ej. ventas del mes pasado).
- ✓ **LEAD():** Accede al valor de una fila posterior (ej. el siguiente precio de cotización).

EJEMPLO N°06-008

Del esquema de los ejemplos anteriores, se quiere ver cada ítem individualmente, pero junto a él, queremos saber:

- ✓ El Precio Promedio de todos los ítems en su misma Categoría.
- ✓ El Ranking de Precio del ítem dentro de su Categoría.

DESARROLLO

1. AVG(i.precio) OVER (PARTITION BY i.id_categoria)

- ✓ **Función:** AVG(i.precio) (Función de Agregación).
- ✓ **OVER():** Le dice a la función que no debe colapsar las filas.
- ✓ **PARTITION BY i.id_categoria:** Define la ventana de cálculo. El motor de SQL agrupa lógicamente los datos por id_categoria, calcula el promedio para ese grupo, y luego repite ese promedio en cada fila que pertenece a esa categoría.

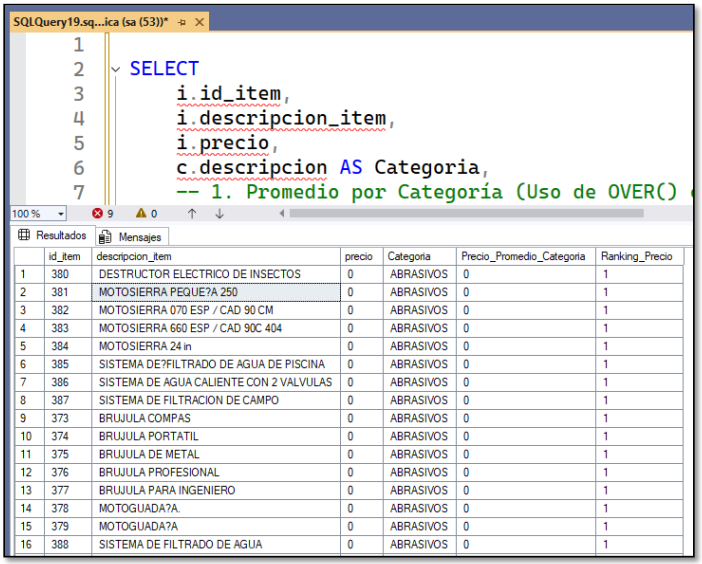
2. RANK() OVER (PARTITION BY i.id_categoria ORDER BY i.precio DESC)

- ✓ **Función:** RANK() (Función de Ventana Específica).
- ✓ **PARTITION BY i.id_categoria:** Reinicia el ranking cada vez que cambia la categoría.
- ✓ **ORDER BY i.precio DESC:** Especifica que el ranking debe ordenarse por precio de forma descendente (el ítem más caro es el Rango 1).

Consulta:

```
SELECT
    i.id_item,
    i.descripcion_item,
    i.precio,
    c.descripcion AS Categoria,
    -- 1. Promedio por Categoría (Uso de OVER() con PARTITION BY)
    AVG(i.precio) OVER (PARTITION BY i.id_categoria) AS
Precio_Promedio_Categoria,
    -- 2. Ranking de Precio (Uso de OVER() con PARTITION BY y ORDER
BY)
    RANK() OVER (PARTITION BY i.id_categoria ORDER BY i.precio
DESC) AS Ranking_Precio
FROM
    item i
JOIN
    categoria c ON i.id_categoria = c.id_categoria
ORDER BY
    Categoria, Ranking_Precio;
```

Resultado consulta se muestra en la IMAGEN N°06-0008:



	id_item	descripcion_item	precio	Categoria	Precio_Promedio_Categoria	Ranking_Precio
1	380	DESTRUCTOR ELECTRICO DE INSECTOS	0	ABRASIVOS	0	1
2	381	MOTOSIERRA PEQUE?A 250	0	ABRASIVOS	0	1
3	382	MOTOSIERRA 070 ESP / CAD 90 CM	0	ABRASIVOS	0	1
4	383	MOTOSIERRA 660 ESP / CAD 90C 404	0	ABRASIVOS	0	1
5	384	MOTOSIERRA 24 in	0	ABRASIVOS	0	1
6	385	SISTEMA DE?FILTRADO DE AGUA DE PISCINA	0	ABRASIVOS	0	1
7	386	SISTEMA DE AGUA CALIENTE CON 2 VALVULAS	0	ABRASIVOS	0	1
8	387	SISTEMA DE FILTRACION DE CAMPO	0	ABRASIVOS	0	1
9	373	BRUJULA COMPAS	0	ABRASIVOS	0	1
10	374	BRUJULA PORTATIL	0	ABRASIVOS	0	1
11	375	BRUJULA DE METAL	0	ABRASIVOS	0	1
12	376	BRUJULA PROFESIONAL	0	ABRASIVOS	0	1
13	377	BRUJULA PARA INGENIERO	0	ABRASIVOS	0	1
14	378	MOTOGUADA?A.	0	ABRASIVOS	0	1
15	379	MOTOGUADA?A	0	ABRASIVOS	0	1
16	388	SISTEMA DE FILTRADO DE AGUA	0	ABRASIVOS	0	1

IMAGEN N°06-0008: Resultado de la consulta realizada en SQLServer

6.2 PROCEDIMIENTOS ALMACENADOS (STORED PROCEDURES)

Los **Procedimientos Almacenados (Stored Procedures - SP)** son conjuntos de instrucciones SQL, precompiladas y almacenadas en el servidor de la base de datos. Funcionan como subrutinas o funciones que pueden ser invocadas por aplicaciones, *scripts* o incluso por otros procedimientos. Encapsulan la lógica de negocio y la ejecutan de manera más eficiente que el envío de múltiples consultas SQL ad-hoc.

6. 2.1. Definición y Arquitectura

Ventajas de Rendimiento y Seguridad

La adopción de procedimientos almacenados ofrece beneficios críticos en un entorno de producción:

- ✓ **Rendimiento (Planes Precompilados):** Cuando un procedimiento se ejecuta por primera vez, el motor de la base de datos crea un **plan de**

ejecución óptimo. Este plan se guarda en caché, permitiendo que las ejecuciones posteriores sean mucho más rápidas al saltarse el proceso de análisis y compilación de la consulta (o *parsing*).

- ✓ **Seguridad:** Permiten a los desarrolladores y usuarios interactuar con los datos sin tener acceso directo a las tablas subyacentes. Se pueden conceder permisos de ejecución sobre el procedimiento, limitando la superficie de ataque y previniendo la manipulación directa de datos o inyección SQL.
- ✓ **Modularidad y Mantenimiento:** Centralizan la lógica de negocio. Si una regla cambia (ej. un cálculo de impuestos), solo es necesario modificar el procedimiento en la base de datos, sin tener que actualizar, recompilar y desplegar toda la aplicación Java.
- ✓ **Reducción del Tráfico de Red:** En lugar de enviar múltiples sentencias SQL a través de la red, la aplicación Java solo envía una sentencia simple para llamar al procedimiento, reduciendo la latencia.

Diferencias Clave entre MySQL y SQL Server

Aunque ambos motores soportan SPs, existen diferencias significativas en la sintaxis, especialmente en el manejo de flujos y delimitadores:

Característica	MySQL	SQL Server
Delimitador	Requiere el comando DELIMITER (ej. DELIMITER //) para cambiar el delimitador temporalmente a algo diferente de ;.	No requiere cambio de delimitador; el ; estándar funciona.
Creación	CREATE PROCEDURE nombre_sp (...)	CREATE PROCEDURE dbo.nombre_sp (...)
Ejecución	CALL nombre_sp(...)	EXEC nombre_sp... o EXECUTE nombre_sp...
Esquema	No es obligatorio referenciar el esquema, aunque se usa en versiones recientes.	La convención es usar dbo. (Database Owner) o el esquema correspondiente.

6. 2.2. Sintaxis y Control de Flujo

El cuerpo del procedimiento se compone de comandos SQL y estructuras de programación que permiten la lógica condicional y repetitiva.

Declaración de Variables y Estructuras de Control

Ambos motores soportan la declaración de variables locales y la manipulación de flujos de ejecución:

Estructura	MySQL	SQL Server
Declaración de Var.	DECLARE variable_name TIPO;	DECLARE @variable_name TIPO;
Asignación	SET variable_name = valor;	SET @variable_name = valor; (o SELECT @variable_name = valor)
Condicional	IF condicion THEN ... ELSEIF ... END IF;	IF condicion ... ELSE ... (sin END IF;)
Bucle	WHILE condicion DO ... END WHILE;	WHILE condicion BEGIN ... END

Manejo de Errores y Transacciones

El manejo de transacciones asegura la atomicidad (propiedad ACID) de las operaciones.

✓ **Transacciones:**

- Ambos usan BEGIN TRANSACTION o START TRANSACTION, COMMIT, y ROLLBACK.
- SQL Server: BEGIN TRAN, COMMIT TRAN, ROLLBACK TRAN. Permite savepoints (SAVE TRANSACTION).
- MySQL: START TRANSACTION, COMMIT, ROLLBACK.

✓ **Manejo de Errores (Diferencias Cruciales):**

Motor	Estrctura de Manejo	Descripción
SQL Server	BEGIN TRY ... END TRY / BEGIN CATCH ... END CATCH	Permite una gestión de errores estructurada y moderna. El código fallido se ejecuta en TRY; si hay un error, el control pasa al bloque CATCH.
MySQL	DECLARE HANDLER	Permite declarar un *handler* (manejador) para una condición específica (ej. SQLSTATE, NOT FOUND, SQLEXCEPTION). La lógica del *handler* se activa automáticamente cuando ocurre la condición.
SQL Server	BEGIN TRY ... END TRY / BEGIN CATCH ... END CATCH	Permite una gestión de errores estructurada y moderna. El código fallido se ejecuta en TRY; si hay un error, el control pasa al bloque CATCH.

6. 2.3. Parámetros de Intercambio de Datos

Los parámetros son el mecanismo para comunicar el procedimiento almacenado con el entorno externo (generalmente, la aplicación Java).

Parámetros de Entrada (IN)

- ✓ Permiten a la aplicación pasar valores **al procedimiento** para ser utilizados en la lógica o en las cláusulas WHERE.
- ✓ Son obligatorios para la ejecución del procedimiento (a menos que tengan un valor por defecto).
- ✓ **Ejemplo:** CREATE PROCEDURE ObtenerVentas (IN p_cliente_id INT)

Parámetros de Salida (OUT) y Bidireccionales (INOUT)

- ✓ OUT **(Salida):** Permiten al procedimiento devolver valores **a la aplicación** (ej. un mensaje de estado, el ID de un registro recién insertado o el resultado de un cálculo). El valor del parámetro se establece *dentro* del procedimiento.
- ✓ INOUT (**Entrada/Salida):** Permite que la aplicación envíe un valor inicial al procedimiento, y este valor puede ser modificado y devuelto a la aplicación.

Ejemplo:

Sintaxis general para crear, modificar y eliminar procedimientos almacenados en SQLServer

```
--crear procedimientos almacenados
CREATE PROCEDURENombreProcedimiento
@parametro1 tipo_dato[=valor1],
@parametro2 tipo_dato[=valor2],
@parametro3 tipo_dato[=valor3],…
 AS
  --sentenciasSQL
----------------------------------------------------------------------
```

```
--modificar procedimientos
ALTER PROCEDURE [dbo].[carga_deuda_alumno]
   @cod char(10),@semsem char(6)
 AS
--sentenciasSQL
-----------------------------------------------------------------------
--eliminar procedimiento
drop procedure NombreProcedimiento
```

Sintaxis general para crear, modificar y eliminar procedimientos almacenados en SQLServer

Sintaxis para crear Procedimiento Almacenado

```
CREATE PROCEDURE NombreProcedimiento (
   IN parametro1 TIPO_DATO,
   IN parametro2 TIPO_DATO,
   IN parametro3 TIPO_DATO
   )
BEGIN
   -- Declarar variables locales usando DECLARE (opcional)
   DECLARE var_local TIPO_DATO DEFAULT 0;

   -- Cuerpo del procedimiento (sentencias SQL)
   SELECT 'Ejecutando procedimiento...';
   END

DELIMITER ; -- Restaura el delimitador a ';'
```

Sintaxis para modificar Procedimiento Almacenado

MySQL **no tiene** la sentencia ALTER PROCEDURE para cambiar el cuerpo del procedimiento. Para modificarlo, primero debes eliminarlo y luego volver a crearlo con la nueva definición.

```
-- 1. Eliminar el procedimiento existente (Si existe)
DROP PROCEDURE IF EXISTS NombreProcedimiento;

-- 2. Volver a crearlo con la nueva lógica (Usando la sintaxis de CREATE)

DELIMITER

CREATE PROCEDURE NombreProcedimiento (
    IN parametro1 TIPO_DATO
)
BEGIN
    -- Nuevo cuerpo del procedimiento
    UPDATE tabla SET columna = 1 WHERE condicion = 1;
END

DELIMITER ;
```

EJEMPLO N°06-009

En la IMAGEN N°06-0009 se tiene el esquema lógico de una base de datos de una empresa prestadora de servicios de dotación de agua. El cual está diseñada para gestionar de forma integral la información relacionada con los usuarios del servicio de agua potable y alcantarillado, así como sus pagos, boletas y estados de conexión. Su estructura central se basa en la tabla usuarios, que almacena datos personales y de ubicación, vinculándose con entidades auxiliares como usuarios_estado, usuarios_tipo y usuarios_categoria para clasificar a cada cliente según su condición, tipo y categoría de servicio. Además, se incluye la relación con la tabla jiron y zona para identificar la ubicación geográfica de cada usuario, y con pileta y pileta_detalle para controlar los puntos de acceso de agua y su historial de consumo o mantenimiento.

En el ámbito de la facturación y control financiero, la base de datos cuenta con las tablas boletasemtes, bdetboltes, bdeudastes y bcuentastes, que permiten registrar boletas, detallar conceptos facturados, controlar deudas y administrar cuentas de los usuarios. Complementariamente, la tabla usuarios_corte gestiona los procesos de suspensión y reconexión del servicio, integrando información de responsables, fechas y estados. El modelo se complementa con módulos de gestión interna como personal, menu y menupersonal para la administración de usuarios del sistema, y variables_globales para la configuración centralizada.

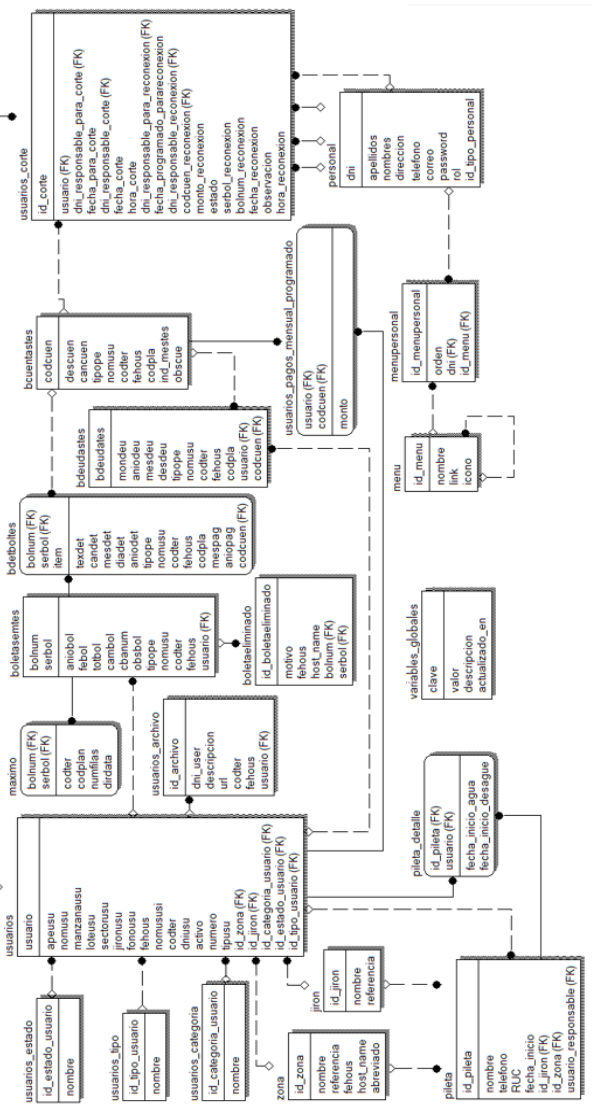


IMAGEN N°06-0009: Esquema físico de la base de datos de control del servicio de agua

Programar un procedimiento almacenado en SQLServer para que reporte las deudas de servicio de dotación de agua de un determinado sector, discriminado por mes y año.

DESARROLLO

El procedimiento almacenado tendrá 4 parámetros de entrada: sector, mes, año, cuenta del ítem. Se utilizará cursores para obtener el monto total que se debe pagar por el concepto (@cuenta) de una tabla (BCuentasTes) y luego itera a través de cada usuario del sector (@sector) para calcular la suma de los pagos realizados por ese usuario en el período indicado (boletasemtes y bdetboltes), finalmente insertando en una tabla temporal (#listaPagos) la diferencia entre el monto total a pagar y el pago real, mostrando así el saldo de la deuda de cada usuario.

A continuación, se tiene el procedimiento almacenado:

```
CREATE procedure [dbo].[repDeuda]
  @sector char(5),
  @mes int,
  @anio int,
  @cuenta  char(8)
as
begin

   --- se declaran variables locales
  declare @usuario char(10)
  declare @apeusu char(50)
  declare @nomusu char(50)
  declare @pago float
  declare @monto_por_pagar float
  set @pago=0.00
```

```
-- se crea un cursor temporal #listaPagos
select usuario,apeusu,nomusu,monto_pago into #listaPagos
from usuarios where sectorusu='44'
-----------------------------------------------------
-- se saca el monto a pagar de la tabla cuentas
declare cCuenta cursor  for select  cancuen
from BCuentasTes where codcuen=@cuenta

open cCuenta

-- se captura el monto en la variable @monto_por_pagar
fetch next from cCuenta into  @monto_por_pagar

-- se cierra la tabla temporal
close cCuenta

-- se elimina la tabla temporal
deallocate cCuenta
-----------------------------------------------------

-- se saca todos los uauarios del sector solicitado
declare cUsuarios cursor  for select usuario,apeusu,nomusu
from usuarios where sectorusu=@sector order by apeusu

open cUsuarios
fetch next from cUsuarios into  @usuario,@apeusu,@nomusu

-- se hace el recorrido por todo el cursor cUsuarios
while(@@fetch_status = 0)
begin
```

```
                -- se saca los pagos que realizo el usuario
                -- por el concepto indicado
                declare cPago cursor  for select sum(d.candet) as pago
                from boletasemtes as b join bdetboltes as d on
                b.bolnum=d.bolnum and b.serbol=d.serbol
                where usuario=@usuario and d.codcuen=@cuenta
                and d.aniopag=@anio and d.mespag=@mes
                group by d.aniopag,d.mespag
                order by mespag

                ------------------------------------------------------

                open cPago
                fetch next from cPago into  @pago
                close cPago
                deallocate cPago
                ------------------------------------------------------
                insert into #listaPagos(usuario,apeusu,nomusu,monto_pago)
                values(@usuario,@apeusu,@nomusu,@monto_por_pagar-
@pago)

                fetch next from cUsuarios into  @usuario,@apeusu,@nomusu

  end

  close cUsuarios
  deallocate cUsuarios

  select *from #listaPagos
end
```

Para probar su ejecución se llama al procedimiento almacenado utilizando:

```
execute repDeuda '1',10,2025,'00000001'
```

Resultado de la ejecución:

	usuario	apeusu	nomusu	monto_pago
1	0000001665	AGUIRRE ISLA	ALEJANDRINA	0.00
2	0000003003	AGUIRRE SOLORZANO	NELSON	0.00
3	0000003013	AGUIRRE VENTURA	MARIA PILAR	0.00
4	0000001660	ALBORNOZ TALENAS	LENIN ADLER	0.00
5	0000001759	ALVARADO DE TIBURCIO	JUANA	0.00
6	0000002279	ALVARADO FUENTES RIVERA	AUGUSTO ABDEL	0.00
7	0000001722	ALVAREZ TINEO	HUMBERTA	0.00
8	0000002747	ARRATEA RODRIGUEZ	LUIS ALBERTO	0.00
9	0000003270	ARRATEA SALVADOR	CARLOS ALBERTO	0.00
10	0000001638	ATAPOMA TARAZONA	MAMERTO	0.00
11	0000003026	ATAVILLOS ARANDA	ETHEL	0.00
12	0000001671	ATAVILLOS HUAMAN	PAULINO	0.00
13	0000001742	ATENCIA ESTEBAN	EDER	0.00
14	0000001731	ATENCIA MARTEL	EUSEBIO	0.00
15	0000001634	ATENCIO ESTRADA	HILARIO	0.00
16	0000002277	AYALA DURAN	FIDENCIA	0.00
17	0000001631	BALDEON SANTACRUZ	DOMINGO JOSÉ	0.00
18	0000002695	BASILIO TOLENTINO	TEODOLINDA	0.00
19	0000003107	BAZÁN ALARCON	ABEL	0.00
20	0000001732	BAZAN ARZAPALO	ASCENCIONA MARISIA	0.00
21	0000001741	BERRIOS MATIAS	GONZALO	0.00
22	0000001740	BERRIOS MATIAS	ARMANDO	0.00

IMAGEN N°06-0010: Resultado de la ejecución del procedimiento almacenado

6.3 FUNCIONES DEL SISTEMA Y DEFINIDAS POR EL USUARIO

Esta sección distingue las funciones que vienen integradas en el motor de la base de datos de aquellas que el desarrollador crea para encapsular lógica y utilizar dentro de las consultas SQL.

6.3.1. Funciones del Sistema (Built-in Functions)

Las funciones del sistema son comandos pre programados que ejecutan cálculos o manipulación de datos sobre una expresión. Son esenciales para el manejo diario de datos.

- **Funciones de Agregación (**SUM, AVG, COUNT**)**
- Ya cubiertas en el contexto de GROUP BY, estas funciones operan sobre un conjunto de filas (un grupo o la tabla completa) y devuelven un único valor resumen.
- **Ejemplos:** Calcular el AVG (promedio) de salarios, o el COUNT (conteo) de registros.
- **Funciones de Fecha y Manipulación de Cadenas**
- Estas funciones son vitales para el *parsing* y formateo de datos.
- **Fechas:** Funciones para obtener la fecha/hora actual (NOW() en MySQL, GETDATE() en SQL Server), extraer partes (YEAR, MONTH), o realizar cálculos (DATEDIFF).
- **Cadenas:** Funciones para modificar texto (UPPER, LOWER), obtener subcadenas (SUBSTRING), o buscar y reemplazar texto.

EJEMPLO N°06-010

Del modelo y base de datos del EJEMPLO N°04-002 del Capítulo V, base de datos de logística, modificar los precios de cada ítem de manera aleatoria.

DESARROLLO

La consulta lo solicitamos que genere IA(Gemini), con este prompt: "*en la tabla item la columna precio esta con 0, quiero remplazar de manera aleatoria un precio a cada item en sqlserver*"

Consulta generada con IA:

```
-- 1. Actualiza solo las filas donde el precio es 0.
UPDATE item
SET precio = (
-- La función RAND() se ejecuta una sola vez por lote en SQL Server.
   -- Para obtener un valor diferente por fila, se usa una técnica que
   -- incluye el chequeo de una columna de fila única (como NEWID() o el ID).
   -- Aquí usamos el cálculo más sencillo para propósitos de ejemplo:
   (ABS(CHECKSUM(NEWID())) % 190) + 10.00
   -- Explicación del cálculo:
   -- NEWID(): Genera un GUID único por fila.
   -- CHECKSUM(): Convierte el GUID en un valor entero.
   -- ABS(): Asegura que el número sea positivo.
   -- % 190: Resto de la división, garantiza un rango de 0 a 189.
   -- + 10.00: Asegura que el precio mínimo sea 10.00.
   -- Rango de Precios Resultante: $10.00 a $199.00
)
WHERE
   precio = 0
   OR precio IS NULL; -- Opcional: también actualiza nulos si los hubiera.
```

6.3.2. Funciones Definidas por el Usuario (UDFs)

Las UDFs permiten a los desarrolladores extender el lenguaje SQL creando su propia lógica para ser utilizada *dentro* de las consultas. A diferencia de los procedimientos almacenados, una UDF está diseñada para devolver un valor y mantener la integridad de la consulta donde se invoca.

Diferencia Principal con los Procedimientos Almacenados

Característica	Función Definida por el Usuario (UDF)	Procedimiento Almacenado (SP)
Valor de Retorno	**Siempre** devuelve un valor (escalar o tabla).	Puede devolver cero, uno o múltiples conjuntos de resultados y/o parámetros OUT.
Uso en SQL	Puede usarse en cualquier cláusula que acepte una expresión (ej. SELECT, WHERE, HAVING).	Se llama con EXEC o CALL, y no puede ser usado directamente en una consulta.
Manipulación de Datos	**Restringida:** Generalmente no puede ejecutar comandos DML (INSERT, UPDATE, DELETE).	**Permitida:** Se usa para modificar datos.

Funciones Escalares: Creación y Sintaxis

- ✓ Devuelven un **único valor** (entero, cadena, fecha, etc.). Son la forma más común de UDF.
- ✓ **Sintaxis Clave:** La cláusula RETURNS es obligatoria para especificar el tipo de dato que será devuelto. En MySQL, el bloque debe usar BEGIN...END y se requiere la sentencia RETURN. En SQL Server, también se usa BEGIN...END.
- ✓ **Uso:** SELECT Nombre, dbo.CalcularImpuesto(Salario) AS Impuesto FROM Empleados;

Funciones con Valores de Tabla (TVFs) en SQL Server y Alternativas

- ✓ Estas funciones devuelven un **conjunto de resultados** (una tabla), lo que les permite ser usadas en la cláusula FROM de una consulta, como si fueran una tabla o vista parametrizada.

- ✓ **SQL Server:** Implementa las TVFs de forma nativa. Existen las TVFs **Inline** (más eficientes, funcionan como vistas parametrizadas) y **Multi-Statement** (para lógica compleja).
- ✓ **MySQL: No tiene un concepto nativo** de TVFs. El desarrollador debe recurrir a la ejecución de un **procedimiento almacenado** que devuelva un *ResultSet* y manejar el resultado en el código Java.

Sintaxis general para crear funciones en SQLServer

```
CREATE FUNCTION [esquema].[NombreFuncion]
(
   -- Parámetros de entrada, opcionales
   @parametro1 tipo_dato,
   @parametro2 tipo_dato
)
RETURNS tipo_dato_retorno -- Define el tipo de dato único que devolverá
AS
BEGIN
   -- Declaración de variables locales
   DECLARE @resultado tipo_dato_retorno;

   -- Lógica de la función
   SET @resultado = @parametro1 * @parametro2; -- Ejemplo de cálculo

   -- La sentencia RETURN es obligatoria y devuelve el valor.
   RETURN @resultado;
END
GO
```

Sintaxis general para crear funciones en MySQL

```
DELIMITER //
CREATE FUNCTION NombreFuncion (
    parametro1 TIPO_DATO,
    parametro2 TIPO_DATO -- Nota: Los parámetros no llevan modo (IN, OUT, INOUT) aquí.
)
RETURNS TIPO_DATO_RETORNO -- Obligatorio: Define el tipo de dato que devolverá
DETERMINISTIC -- Opcional: Indica que siempre devuelve el mismo resultado para las mismas entradas
BEGIN
    -- Declaración de variables locales
    DECLARE variable_local TIPO_DATO_RETORNO;

    -- Lógica de la función
    SET variable_local = parametro1 * parametro2;

    -- La sentencia RETURN es obligatoria y debe devolver un valor único
    RETURN variable_local;
END //

DELIMITER ; -- Restaura el delimitador estándar
```

EJEMPLO N°06-011

Del esquema de la base de datos del EJEMPLO N°04-002 del Capítulo V, base de datos de logística, programar una función que devuelva el valor total del inventario para ese producto (stock * precio).

DESARROLLO

Se crea una funcion CalcularValorStock en SQL Server como una **función escalar** que acepta un parámetro de entrada, @ItemId (de tipo CHAR(10)), el cual representa el identificador único del ítem. Dentro del bloque BEGIN...END, la función utiliza una sentencia SELECT para consultar la tabla **item**, multiplicar el valor de la columna **stock** por la columna **precio** (usando ISNULL para manejar posibles valores nulos en el cálculo) y asignar el resultado a una variable local llamada @ValorTotal. Finalmente, la función devolverá este valor calculado, que representa el valor monetario total del inventario para ese ítem, con un tipo de retorno definido como DECIMAL(10, 2)

Codigo SQLServer:

```
CREATE FUNCTION dbo.CalcularValorStock
(
    -- Parámetro de entrada: el ID del ítem que queremos evaluar
    @ItemId CHAR(10)
)
RETURNS DECIMAL(10, 2) -- El valor del stock es monetario, usamos DECIMAL
AS
BEGIN
    -- Declaramos una variable para almacenar el resultado del cálculo
    DECLARE @ValorTotal DECIMAL(10, 2);


    -- 1. Buscamos el stock y el precio, y calculamos el valor total.
    --    Se usa ISNULL para asegurar que si stock o precio son NULL, el
resultado sea 0.
    SELECT
        @ValorTotal = ISNULL(i.stock, 0) * ISNULL(i.precio, 0)
    FROM
        item i
    WHERE
```

```
        i.id_item = @ItemId;

    -- 2. Devolvemos el valor total del stock del ítem
    RETURN @ValorTotal;
END
GO

-- ----------------------------------------------------
-- EJEMPLO DE USO DE LA FUNCIÓN
-- ----------------------------------------------------
-- Asumiendo que tenemos un ítem con id_item 'A100'
SELECT
    i.descripcion_item,
    i.stock,
    i.precio,
    -- Se llama a la función para obtener el valor calculado
    dbo.CalcularValorStock(i.id_item) AS Valor_Stock_Total
FROM
    item i
WHERE
    i.id_item = '1002';
GO
```

Resultado de la ejecución de la consulta se muestra en la IMAGEN N°06-0011:

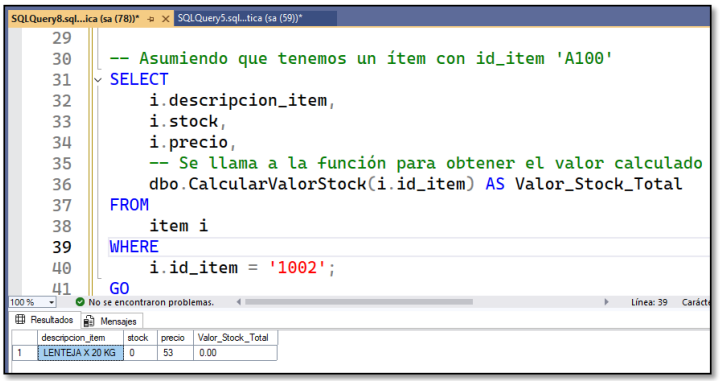


IMAGEN N°06-0011: Resultado uso de la función

6.4 VISTAS (VIEWS)

Una **Vista** en SQL es una **tabla virtual** basada en el conjunto de resultados de una consulta SQL. La vista contiene filas y columnas, al igual que una tabla real, pero sus datos no se almacenan físicamente en la base de datos; la vista simplemente almacena la consulta que define los datos. Cada vez que se consulta la vista, se ejecuta la consulta subyacente.

6. 4.1. Creación y Propósito

Las vistas se crean con la sentencia CREATE VIEW y se consultan con SELECT como si fueran tablas normales.

Uso de Vistas para Seguridad y Abstracción de Datos

El propósito principal de las vistas recae en la gestión de la complejidad y el control de acceso:

- ✓ **Abstracción de Datos:** Permiten ocultar la complejidad de los *joins* y las subconsultas a los usuarios y a la aplicación Java. En lugar de escribir una consulta compleja de cinco uniones, la aplicación simplemente consulta una vista llamada, por ejemplo, Vista_PedidosCompletos.

- ✓ **Seguridad y Restricción de Acceso:** Una de las funciones más importantes. Permiten a los administradores restringir el acceso a columnas o filas específicas de una tabla base. Por ejemplo, se puede crear una vista de la tabla Empleados que omita la columna Salario y dar acceso solo a esa vista, garantizando que los datos sensibles permanezcan ocultos.
- ✓ **Lógica de Negocio Consistente:** La lógica de cálculo o filtrado definida en la vista se aplica de manera consistente en cualquier aplicación que la use, asegurando que todos vean el mismo conjunto de datos transformados.

```
SQL
-- Ejemplo de creación de una Vista que oculta información sensible
CREATE VIEW Empleados_Publico AS
SELECT
   ID,
   Nombre,
   Puesto,
   Departamento
FROM
   Empleados;

-- La aplicación Java solo consultará la vista:
SELECT Nombre, Puesto FROM Empleados_Publico WHERE Departamento =
'Ventas';
```

6. 4.2. Vistas Materializadas (Materialized Views)

Concepto y Uso para Mejorar el Rendimiento de Consultas Complejas

Mientras que una vista estándar es una **tabla virtual** que se calcula *en tiempo real* cada vez que se llama, una Vista Materializada es una **tabla real** cuyos resultados se almacenan físicamente en el disco.

Característica	Vista Estándar	Vista Materializada
Almacenamiento	Virtual (Solo guarda la definición SQL).	**Físico** (Guarda los datos resultantes).
Tiempo de Consulta	Lento (Ejecuta la consulta subyacente cada vez).	**Rápido** (Solo lee la tabla pre calculada).
Datos Actuales	Siempre al día.	Se actualizan periódicamente (se deben "refrescar").

- ✓ **Propósito:** Las vistas materializadas están diseñadas específicamente para **mejorar el rendimiento** de consultas analíticas o de *reporting* que involucran grandes agregaciones (GROUP BY, SUM, AVG) o *joins* complejos.
- ✓ **Uso:** El motor de la base de datos (principalmente **SQL Server** bajo el nombre de "Vistas Indexadas" u **Oracle/PostgreSQL** como "Materialized Views") tiene la tarea de **refrescar** estos datos de forma periódica (por ejemplo, cada noche o cada hora), lo que significa que el rendimiento de lectura se dispara a cambio de una ligera demora en la actualización de los datos.

EJEMPLO N°06-012

Del esquema de la base de datos del EJEMPLO N°04-002 del Capítulo V, base de datos de logística, crear una vista en SQLServer que unifica la información de los ítems con las descripciones legibles de sus categorías y unidades de medida mediante JOINs, simplificando futuras consultas de inventario.

DESARROLLO

Se crea la vista **Vista_InventarioDetallado** con **CREATE VIEW**. Esta vista actúa como una **tabla virtual** que consolida datos de las tablas **item**, **categoria** y **unidad_medida**. Su definición incluye una consulta base (SELECT) que utiliza operaciones **JOIN** para relacionar item con categoria usando id_categoria y con unidad_medida usando id_unidad_medida.

Codigo SQLServer:

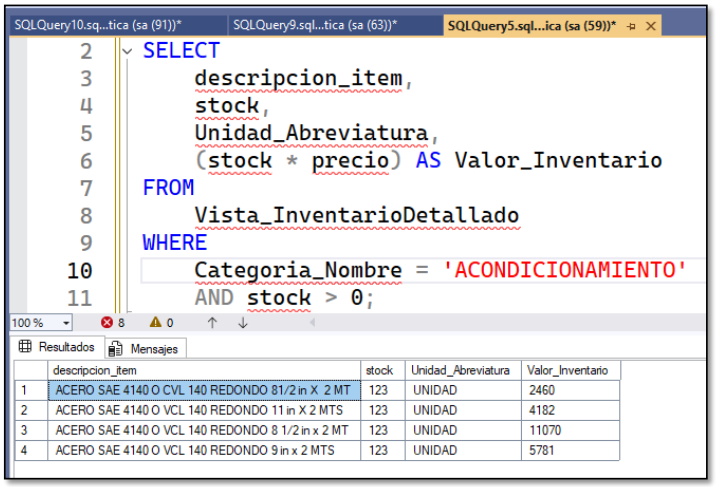
```
CREATE VIEW Vista_InventarioDetallado
```

```
AS
SELECT
    i.id_item,
    i.descripcion_item,
    i.stock,
    i.precio,
    i.tipo,
    c.descripcion AS Categoria_Nombre, -- Nombre de la Categoría
    um.descripcion AS Unidad_Medida_Nombre, -- Nombre de la Unidad de
Medida
    um.abreviatura AS Unidad_Abreviatura
FROM
    item i
JOIN
    categoria c ON i.id_categoria = c.id_categoria
JOIN
    unidad_medida um ON i.id_unidad_medida = um.id_unidad_medida;
GO
```

Uso de la vista

```
-- Consultar todos los ítems de la categoría 'Oficina'
SELECT
    descripcion_item,
    stock,
    Unidad_Abreviatura,
    (stock * precio) AS Valor_Inventario
FROM
    Vista_InventarioDetallado
WHERE
    Categoria_Nombre = 'ACONDICIONAMIENTO'
    AND stock > 0;
```

Resultado de la ejecución:



	descripcion_item	stock	Unidad_Abreviatura	Valor_Inventario
1	ACERO SAE 4140 O CVL 140 REDONDO 81/2 in X 2 MT	123	UNIDAD	2460
2	ACERO SAE 4140 O VCL 140 REDONDO 11 in X 2 MTS	123	UNIDAD	4182
3	ACERO SAE 4140 O VCL 140 REDONDO 8 1/2 in x 2 MT	123	UNIDAD	11070
4	ACERO SAE 4140 O VCL 140 REDONDO 9 in x 2 MTS	123	UNIDAD	5781

IMAGEN N°06-0012: Resultado uso de la consulta a una vista

CAPÍTULO 7. INTEGRACIÓN DE JAVA CON BASE DE DATOS

7.1 PERSISTENCIA DE DATOS

En el desarrollo de software, la **persistencia de datos** es un concepto fundamental que se refiere a la capacidad de almacenar información de manera permanente para su uso futuro. Tradicionalmente, esta persistencia se ha logrado utilizando archivos de texto, archivos binarios o formatos estructurados como XML o JSON. Sin embargo, aunque este enfoque es relativamente sencillo de implementar, conlleva una serie de desafíos y limitaciones que pueden volverse complejos a medida que la aplicación crece.

Desafíos de la Persistencia en Archivos

Al utilizar archivos para la persistencia de datos, surgen varios problemas comunes:

- ✓ **Escalabilidad**: A medida que el volumen de datos crece, manejar archivos grandes se vuelve ineficiente. Leer y escribir en grandes cantidades de datos puede afectar negativamente el rendimiento.
- ✓ **Concurrencia**: En aplicaciones donde múltiples usuarios o procesos acceden a los mismos datos, el control de acceso a archivos se vuelve complicado. Los archivos no proporcionan un mecanismo eficiente para manejar accesos concurrentes.
- ✓ **Integridad de datos**: La posibilidad de corrupción de archivos es alta, especialmente si no se manejan correctamente las operaciones de lectura/escritura. Además, no hay un sistema robusto que garantice la integridad de los datos cuando ocurren fallos en la aplicación.
- ✓ **Estructura y relaciones complejas**: Aunque los archivos pueden almacenar datos estructurados, gestionar relaciones complejas entre datos, como en el caso de un sistema de inventario o un sistema bancario, se vuelve laborioso y propenso a errores.
- ✓ **Búsqueda y manipulación**: Encontrar información específica dentro de archivos grandes requiere técnicas de búsqueda manual o herramientas

adicionales, lo que puede hacer que la manipulación de los datos sea lenta y tediosa.

Transición a una Base de Datos

Para superar estas dificultades, se introduce el uso de **bases de datos** como medio de persistencia. Las bases de datos proporcionan una solución más escalable, segura y eficiente para almacenar y gestionar grandes volúmenes de información. En lugar de gestionar los datos a nivel de archivo, las bases de datos permiten almacenar los datos en **tablas** con un sistema relacional o en documentos estructurados (como en bases de datos NoSQL), donde las operaciones de manipulación de datos son rápidas y optimizadas.

Al utilizar **Java** para interactuar con bases de datos, se hace uso de tecnologías como **JDBC (Java Database Connectivity)** o **ORMs (Object Relational Mappers)** como **Hibernate**, lo que simplifica la integración de la lógica de negocio con los datos almacenados.

Las principales ventajas de utilizar una base de datos incluyen:

- ✓ **Escalabilidad y eficiencia**: Las bases de datos están diseñadas para manejar grandes volúmenes de datos de forma eficiente, permitiendo la lectura y escritura rápida y organizada.
- ✓ **Concurrencia**: Los sistemas de bases de datos permiten múltiples accesos concurrentes con mecanismos de bloqueo y control transaccional.
- ✓ **Integridad de datos**: Con el uso de transacciones y restricciones, se puede garantizar que los datos sean consistentes y válidos.
- ✓ **Consultas complejas**: Mediante el uso de SQL o lenguajes de consulta equivalentes, se pueden realizar búsquedas y manipulaciones complejas de los datos con facilidad.

Implementación de la Persistencia en una Base de Datos desde Java

Para implementar la persistencia de datos en una base de datos desde Java, el primer paso es conectar la aplicación a una base de datos mediante **JDBC**. Este proceso involucra:

- ✓ **Configurar la conexión**: Definir los parámetros de conexión, como la URL de la base de datos, el usuario y la contraseña.
- ✓ **Ejecutar consultas SQL**: Utilizar sentencias SQL para realizar operaciones de **inserción**, **actualización**, **eliminación** y **consulta** de los datos.
- ✓ **Manejo de transacciones**: Garantizar la integridad de los datos mediante el uso de transacciones.
- ✓ **Cerrar recursos**: Asegurarse de que todos los recursos, como conexiones y sentencias, se cierren adecuadamente para evitar problemas de rendimiento y seguridad.

Una alternativa más avanzada a JDBC es el uso de un framework ORM como **Hibernate**, que permite mapear los objetos Java a las tablas de la base de datos de manera automática, reduciendo la necesidad de escribir SQL manualmente y permitiendo trabajar directamente con objetos.

Por todo esto, la transición de archivos a bases de datos para la persistencia de datos en Java ofrece un enfoque más robusto, escalable y eficiente. Aunque conlleva una mayor complejidad inicial, las ventajas en cuanto a rendimiento, manejo de concurrencia, y la integridad de los datos hacen que sea una elección preferida para sistemas de información modernos y de mayor envergadura.

7.2 CONEXIÓN DE JAVA CON MYSQL

Instalación de MySQL

MySQL es un gestor de bases de datos relacionales esencial para la persistencia de datos. Permite almacenar de forma duradera la información generada desde una interfaz, como la desarrollada en Java. Para lograr esta comunicación, es necesario seguir ciertos pasos e integrar librerías específicas

(drivers) que establecen el vínculo entre el código Java y la base de datos. Para su instalación descargar:

- ✓ XAMPP(https://www.apachefriends.org/es/index.html) o
- ✓ LARAGON(https://laragon.org/download), los cuales tienen el Apache, PHP, MySQL. phpMyAdmin entre otros.

Una vez instalado, corriendo LARAGON:

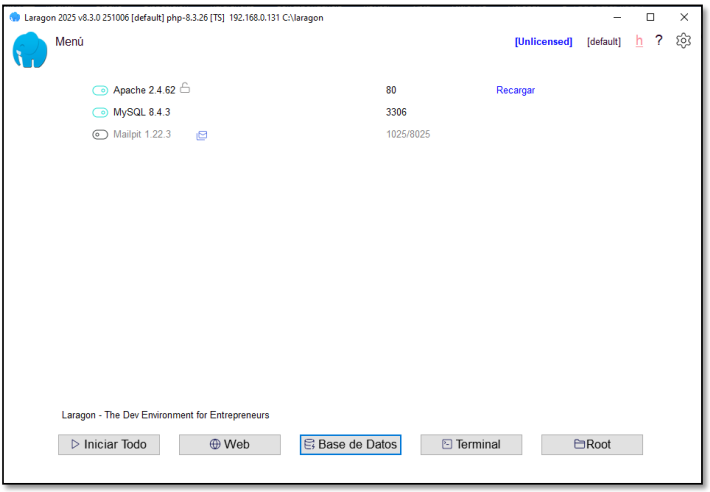


IMAGEN Nº07-0001: Pantalla del administrador de los servicios de LARAGON

Para acceder al administrador de base de datos en MySQL clic en el botón “Base de Datos” y se tendrá la interfaz de la IMAGEN Nº07-0002.

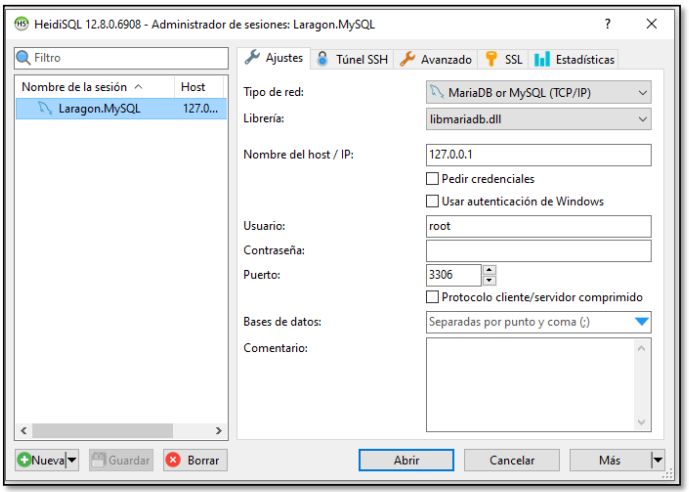


IMAGEN Nº07-0002: Pantalla para acceder a MySQL

En la IMAGEN Nº07-0003 se muestra la interfaz para administrar bases de datos en MySQL.

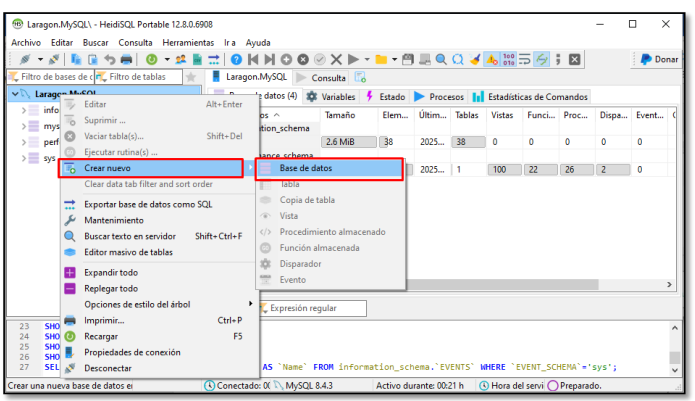


Figura Nº07-003: Interfaz para administrar bases de datos MySQL

Para administrar base de datos MySQL se tiene Workbench, phpMyAdmin, entre otros.

Conexión

La conexión de una aplicación desarrollada en Java a una base de datos MySQL es un proceso que funciona como un "puente de comunicación", haciendo que ambos sistemas (que hablan lenguajes distintos) se entiendan.

Aquí están las tres etapas principales:

1. El Servidor y la Base de Datos (MySQL)

- ✓ **Función:** MySQL actúa como el **servidor de datos**. Es el motor que gestiona, almacena y organiza la información de manera persistente (duradera) en tablas relacionales.
- ✓ **Requisito:** El servidor MySQL debe estar **instalado** y **activo**. Además, la base de datos específica (ej. biblioteca) y las tablas deben estar **previamente creadas** y esperando peticiones.
- ✓ **Identificación:** El servidor se localiza mediante una **dirección IP** (usualmente localhost o 127.0.0.1) y un **puerto** (por defecto, 3306).

2. El Conector (JDBC Driver)

- ✓ **Función:** Este es el elemento clave que actúa como un **traductor** o intermediario. En el contexto de Java, se le llama **JDBC Driver** (Java Database Connectivity), específicamente el **MySQL Connector/J**.
- ✓ **Proceso:** El conector es un archivo **.jar** que debe ser **descargado** e **incluido** como una librería dentro del proyecto Java.
- ✓ **Resultado:** Al incluir el driver, la Máquina Virtual de Java (**JVM**) obtiene el conjunto de clases e instrucciones necesarias para formular las peticiones en un formato que MySQL pueda entender.

3. El Cliente y el Código Java (La Aplicación)

- ✓ **Función:** El código Java actúa como el **cliente** que inicia la comunicación y envía las órdenes (**consultas SQL**).
- ✓ **Pasos de Código:**

1. Descargar el conector de MySQL: MySQL Community Downloads, de la página oficial, como se muestra en la IMAGEN Nº07-0004: https://dev.mysql.com/downloads/connector/j/

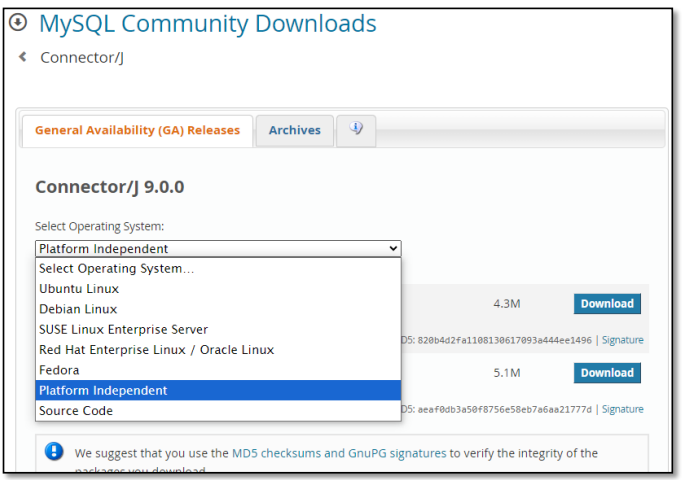


IMAGEN Nº07-0004: Página oficial de descarga del conector de MySQL

Teniendo descargado la librería en NetBeans se adiciona la librería al proyecto, tal como se muestra en la IMAGEN Nº07-0005 e IMAGEN Nº07-0006

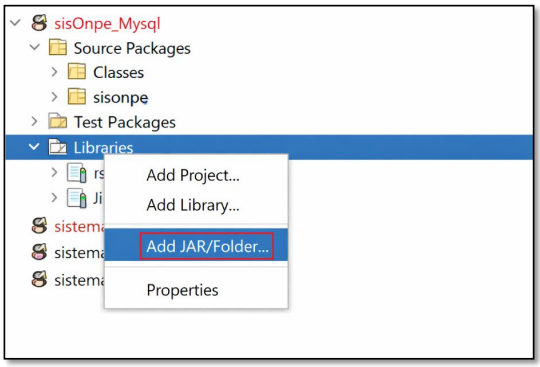


IMAGEN Nº07-0005: Add del JAR de conexión con MySQL

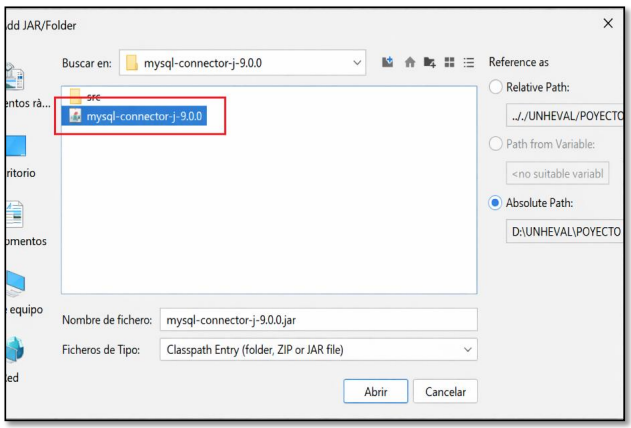


IMAGEN Nº07-0006: Se ubica el jar para agregar

Teniendo agregado se visualiza en la parte de Librería en el proyecto tal como se muestra IMAGEN Nº07

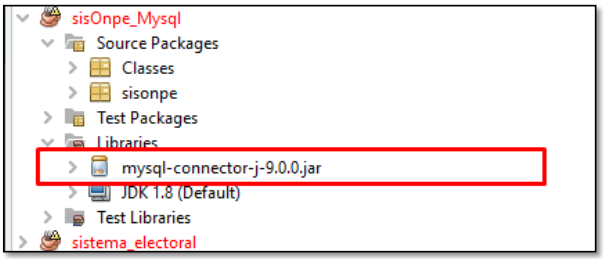


IMAGEN Nº07-0007: jar agregado en NetBenas

2. **Carga del Driver:** El código intenta cargar la clase principal del driver (aunque es opcional en versiones modernas de Java, es una buena práctica conceptual).
3. **Establecimiento de la Conexión:** Se utiliza la clase DriverManager de Java para intentar establecer la conexión usando la **URL de la base de datos**, el **nombre de usuario** y la **contraseña**.
   - ✓ La URL sigue un formato estándar: jdbc:mysql://[dirección]:[puerto]/[nombre_db].
4. **Ejecución de Peticiones:** Una vez que la conexión es exitosa (obteniendo un objeto Connection), el código puede crear objetos (Statement o PreparedStatement) para enviar comandos SQL como SELECT, INSERT, UPDATE o DELETE.
5. **Cierre:** Finalmente, se cierran todos los recursos (ResultSet, Statement y Connection) para liberar la conexión del servidor.

Clase de conexión para MySQL

Clase que permite conectarse a la base de datos Mysql desde Java, importar los paquetes correspondientes:

```
import java.sql.Connection;

import java.sql.DriverManager;
```

Tambien se podría importar asi:

```
import java.sql.*;
```

Para la conexión se necesita como mínimo tres atributos:

- ✓ La URL de conexión que incluye la base de datos
- ✓ El usuario de la base de datos
- ✓ La clave de acceso a la base de datos

El usuario y clave son datos creados en MySQL como usuarios de la base de datos, con todo el permiso necesario para la gestión de los datos (poblar, modificar, eliminar, realizar consultas, etc.)

Se necesitará un método que retorne un valor de tipo Connection

```
public Connection conexion(){}
```

Lo primero que se hace para poder conectarse a una base de datos es cargar el driver encargado de ésta función. Para ello es utilizada la llamada ***Class.forName***.

```
Class.forName("DriverXYZ");
```

Donde *DriverXYZ* corresponde al driver a cargar, por ejemplo, el driver JDBC-ODBC es ***sun.jdbc.odbc.JdbcOdbcDriver***; el driver JDBC-MySQL es ***com.mysql.cj.jdbc.Driver***.

Una vez cargado el driver es necesario crear un objeto del tipo *Connection*, para administrar la conexión. Una aplicación puede utilizar ***DriverManager*** para obtener un objeto de tipo conexión, ***Connection***, con una base de datos. La conexión se especifica siguiendo una sintaxis basada en la especificación más amplia de los URL, de la forma:

jdbc:subprotocolo//servidor:puerto/base de datos

La siguiente línea de código ilustra ésta idea:

Connection cn = DriverManager.getConnection(url, "myLogin", "myPassword");

En la clase: cn = DriverManager.getConnection(url, usuario, clave);

Si uno de los drivers que hemos cargado reconoce la URL suministrada por el método ***DriverManager.getConnection***, dicho driver establecerá una conexión con el controlador de base de datos especificado en la URL del JDBC. La clase ***DriverManager*** , como su nombre indica, maneja todos los detalles del establecimiento de la conexión detrás de la escena. A menos que estemos escribiendo un driver, posiblemente nunca utilizaremos ningún método de la interface Driver.

La conexión devuelta por el método ***DriverManager.getConnection*** es una conexión abierta que se puede utilizar para crear sentencias JDBC que pasen nuestras sentencias SQL al controlador de la base de datos.

Después, dentro de un try, se llama a la clase que maneja el Driver para la base de datos y se realiza la conexión.

Código completo de la Clase Conexión

```
package Classes;
import java.sql.*;
public class Conexion
{

    String usuario="root";//usuario de mysql
    String clave="";    //clave del usuario
    String url="jdbc:mysql://localhost/sistema_elecciones"; //base de datos

    public  Connection conexion()
    {
      Connection cn=null;
      try {
            Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
            cn = DriverManager.getConnection(url, usuario, clave);

           }
```

```
            catch (Exception ex)
                {
                 System.out.println("Error: "+ex.getMessage());
                }

             return cn;
           }
}
```

EJEMPLO N°06-001

“Sistema de proceso electoral”

El Perú es un país democrático cuya estructura de gobierno es: Gobierno Nacional, Gobiernos Regionales y Gobiernos Locales, en estos diferentes niveles de gobierno sus autoridades se les elige en voto popular. Al presidente de República y Congresistas se les eligen para un periodo de 5 años. Para autoridades de los Gobiernos Regionales y Municipales el periodo es de 4 años. La población para participar en los procesos electorales lo realizan a través de una Organización Política normada por la LEY Nº 28094 Ley de Organizaciones Política y que está inscrito en el Registro de Organizaciones Políticas(https://sroppublico.jne.gob.pe/) del Jurado Nacional de Elecciones, para ellos los órganos encargados de llevar el proceso son: El Jurado Nacional de Elecciones - JNE, La Oficina Nacional de Procesos Electorales-ONPE, Registro Nacional de Identificación y Estado Civil – RENIEC, entre otros, quienes cada uno de ellos tienen sus competencias de acuerdo a la Ley N° 26859 Ley Orgánica de Elecciones.

Se desea desarrollar un sistema para realizar todo el proceso de inscripción de candidatos a un proceso electoral y a su vez el proceso de elección, conteo de votos, resultados, para ello diseñe la base de datos que responda a todos los requerimientos.

Se propone desarrollar un Sistema que gestione de manera eficiente y segura los distintos procesos electorales en Perú, que pueden ser de ámbito **nacional**, **regional** o **local**, que permita realizar el proceso de inscripción de candidatos a

un proceso electoral y a su vez el proceso de elección, conteo de votos, resultados, para ello diseñe la base de datos que responda a todos los requerimientos.

El sistema debe estar respaldado por una base de datos que permita almacenar toda la información necesaria para la correcta organización y ejecución de los comicios.

Página de ayuda: https://www.onpe.gob.pe/elecciones/historico-elecciones/

Estructura y Procedimientos:

1. Base de Datos del Sistema:

- Se creará una base de datos que registre los datos de los **partidos políticos** participantes. Cada partido político deberá contar con la siguiente información:

  - ✓ **Siglas** del partido.
  - ✓ **Nombre completo** del partido.
  - ✓ **Dirección de la sede principal**.

- Los partidos políticos podrán participar en los tres tipos de procesos electorales:

  - ✓ **Elecciones Nacionales**: Para elegir Presidente y congresistas de la República.

  - ✓ **Elecciones Regionales**: Para elegir gobernadores y consejeros regionales.
  - ✓ **Elecciones Locales**: Para elegir alcaldes provinciales y distritales, así como regidores.

- Para este caso específico, el enfoque estará en el **proceso electoral local**, destinado a la elección de alcaldes de municipalidades **provinciales** o **distritales**.

2. **Organización de Locales de Votación y Mesas de Sufragio:**

- El sistema organizará a los **electores** en **mesas de sufragio**, las cuales estarán distribuidas en **locales de votación**. Cada local de votación está asociado a un **distrito**, y en cada distrito puede haber múltiples locales de votación.
- Cada local puede albergar varias mesas de sufragio, y cada **mesa** tiene asignados a un número específico de electores.

3. **Composición de las Mesas de Sufragio:**

- Cada mesa de sufragio estará compuesta por **tres miembros** designados: un **presidente**, un **secretario** y un **vocal**. Estos miembros serán responsables de la supervisión y correcto desarrollo del proceso de votación en su respectiva mesa.

4. **Asignación de Electores:**

- Los **electores** estarán asignados a una mesa de sufragio en un local de votación determinado. El sistema permitirá el control de los electores que acuden a votar, garantizando que solo voten en la mesa y el local que les corresponden.
- La información de los electores será gestionada de manera segura y se verificará durante el proceso de votación.

5. **Tipos de Elecciones:**

- El sistema debe ser capaz de adaptarse a los distintos niveles de elecciones:

✓ **Nacionales**: Votaciones para elegir autoridades nacionales como Presidente, Vicepresidentes y congresistas.
✓ **Regionales**: Para elegir autoridades de gobiernos regionales, como gobernadores y consejeros.
✓ **Locales**: Elecciones para alcaldes provinciales y distritales, así como regidores.

6. **Registro de Votación y Resultados:**

- Durante el proceso electoral, los electores emiten su voto en la mesa de sufragio asignada.
- Al finalizar la votación, los miembros de mesa registran los votos y los resultados son transmitidos al sistema.
- El sistema consolidará los resultados y generará **reportes detallados** por:

✓ **Mesa de sufragio**.
✓ **Local de votación**.
✓ **Distrito**.
✓ **Provincia** (en caso de una elección provincial).

7. **Reporte de Resultados:**

- Los resultados electorales serán reportados de manera jerárquica a las autoridades correspondientes:

✓ En elecciones **locales**, los resultados se envían a la alcaldía distrital o provincial.
✓ En elecciones **regionales**, los resultados se envían a la gobernación regional.
✓ En elecciones **nacionales**, los resultados son consolidados a nivel nacional y enviados a las autoridades competentes.

8. **Flujo del Proceso Electoral:**

- Los electores acuden al local de votación asignado para emitir su voto en la mesa correspondiente.
- Los miembros de mesa supervisan el proceso de votación y validan los votos emitidos.
- Al cierre del proceso, los votos de cada mesa son contabilizados y enviados al sistema central.
- El sistema agrupa los resultados y genera reportes detallados a nivel local, regional o nacional según el tipo de elección, garantizando un control y transparencia en cada fase del proceso.

DESARROLLO

Para el caso a estudiar necesitamos una serie de tablas que permita almacenar datos de: ***electores, partidos políticos, las regiones, provincias, distritos, centros de votación, mesas, los candidatos que participan para una alcaldía provincial y distrital con sus respectivos regidores, el padrón electoral, los miembros de mesa, entre otros***. En la IMAGEN Nº07-0007 se tiene el esquema lógico de la base de datos

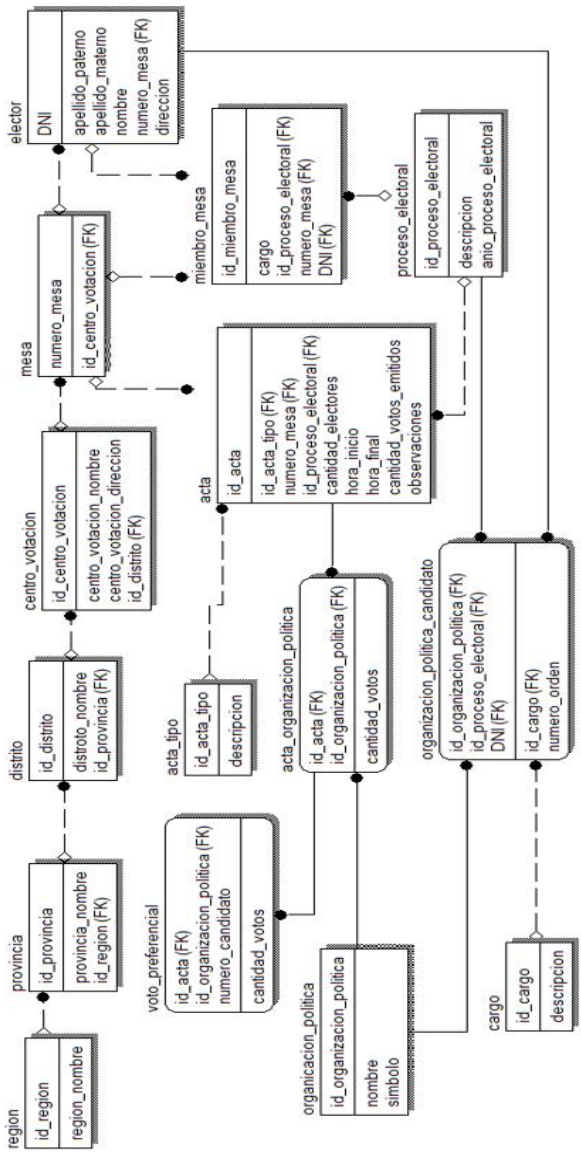


IMAGEN N°07-0007: Esquema lógico de la base de datos

Se crea una base de datos "*sistema_electoral*", en MyQSL:

IMAGEN Nº07-0008: Creación de la base de datos "*sistema_electoral*"

Creación de las tablas

Para la creación de las tablas se puede hacer de dos maneras, desde la ventana del MySQL o con código

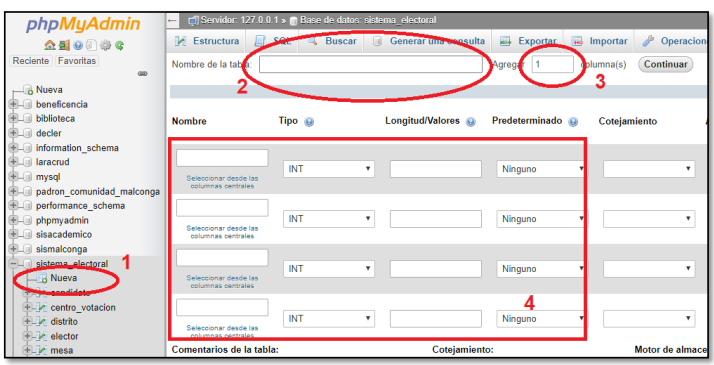


IMAGEN Nº07-0009: Proceso para crear una tabla con sus columnas

Seleccionar la base de datos, seleccionar nueva (1), poner el nombre de la tabla (2), agregar el número de columnas (3), y agregar los nombres de las columnas con sus respectivos tipos de datos, tamaño entre otros (4).

Script de creación de algunas tablas en código SQL para MySQL:

```
CREATE TABLE candidato (
                    DNI_candidato char(8) NOT NULL,
                    partido_politico_codigo int(11) NOT NULL,
                    distrito_provincia int(11) NOT NULL,
tipo_eleccion int(11) NOT NULL DEFAULT 1
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE centro_votacion (
                    centro_votacion_codigo int(11) NOT NULL,
                    centro_votacion_nombre varchar(80) NOT NULL,
                     centro_votacion_direccion varchar(80) NOT NULL,
                     distrito_codigo int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE distrito (
                    distrito_codigo int(11) NOT NULL,
                    distrito_nombre varchar(80) NOT NULL,
                    provincia_codigo int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE elector (
                    DNI char(8) NOT NULL,
```

```
                    Apellidos varchar(80) NOT NULL,
                    Nombres varchar(80) NOT NULL,
                    direccion varchar(50) NOT NULL,
                    mesa int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE mesa (
                    mesa int(11) NOT NULL,
                    centro_votacion int(11) NOT NULL,
                    DNI_presidente char(8) NOT NULL,
                    DNI_secretario char(8) NOT NULL,
                    DNI_vocal char(8) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE partido_politico (
                    partido_politico_codigo int(11) NOT NULL,
                    partido_politico_nombre varchar(80) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE provincia (
  provincia_codigo int(11) NOT NULL,
  provincia_nombre varchar(80) NOT NULL,
  region_codigo int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE region (
                    region_codigo int(11) NOT NULL,
```

```
                region_nombre varchar(80) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

La base de datos “sistema_electoral” con las tablas, quedara así:

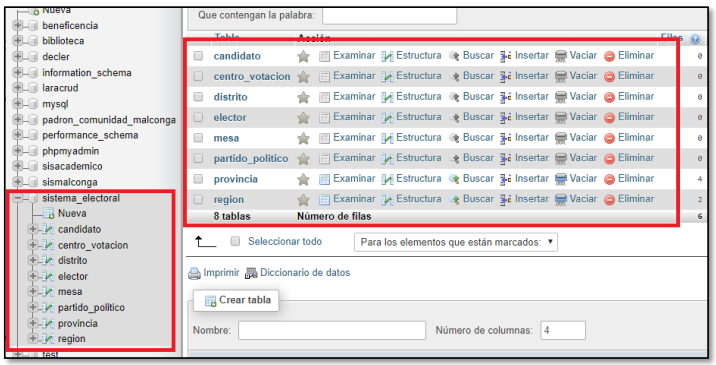


IMAGEN Nº07-0010: Base de datos “sistema_electoral” con todas las tablas creadas

Para hacer autoincrementado cualquier campo de tipo numerico, de cuaquier tabla despues de creado se puede hacer con codigo utilizando ALTER TABLE, pero para ello el campo tiene que ser llave primaria, por ejemplo en la tabla partido_politico se hace que la columna “*partido_politico_codigo*” se autoincremente de la siguiente manera:

```
ALTER TABLE partido_politico CHANGE partido_politico_codigo
partido_politico_codigo INT UNSIGNED NOT NULL AUTO_INCREMENT
```

SENTENCIAS SQL

En toda base de datos con modelo relacional, para poblarla, modificar, eliminar registros de una tabla, se utiliza sentencias **SQL** (Structured Query Language, en castellano: Lenguaje de Consulta Estructurado), que es un lenguaje declarativo estándar internacional de comunicación dentro de una base de datos. Existen sentencias **SQL** básicas que alteran los registros de una tabla en una base de datos y una sentencia que permite recuperar datos y generar información, estas sentencias son: **INSERT, UPDATE, DELETE Y SELECT**

Sentencia INSERT:

Sentencia UPDATE:

Sentencia DELETE:

Sentencia SELECT:

Insertando registros a la tabla: "region"

```
INSERT INTO `region` (`region_codigo`, `region_nombre`) VALUES

(1, 'AMAZONAS'),
(2, 'ANCASH'),
(3, 'APURIMAC'),
(4, 'AREQUIPA'),
(5, 'AYACUCHO'),
(6, 'CAJAMARCA'),
(7, 'PROV. CONST. DEL CALLAO'),
(8, 'CUSCO'),
(9, 'HUANCAVELICA'),
(10, 'HUANUCO'),
(11, 'ICA'),
(12, 'JUNIN'),
(13, 'LA LIBERTAD'),
(14, 'LAMBAYEQUE'),
(15, 'LIMA'),
(16, 'LORETO'),
(17, 'MADRE DE DIOS'),
(18, 'MOQUEGUA'),
(19, 'PASCO'),
(20, 'PIURA'),
(21, 'PUNO'),
(22, 'SAN MARTIN'),
```

(23, 'TACNA'),
(24, 'TUMBES'),
(25, 'UCAYALI');

Insertando registros a la tabla "provincia"

INSERT INTO `provincia` (`provincia_codigo`, `provincia_nombre`, `region_codigo`) VALUES

(101, 'CHACHAPOYAS', 1),
(102, 'BAGUA', 1),
(103, 'BONGARA', 1),
(104, 'CONDORCANQUI', 1),
(105, 'LUYA', 1),
(106, 'RODRÍGUEZ DE MENDOZA', 1),
(107, 'UTCUBAMBA', 1),
(201, 'HUARAZ', 2),
(202, 'AIJA', 2),
(203, 'ANTONIO RAYMONDI', 2),
(204, 'ASUNCIÓN', 2),
(205, 'BOLOGNESI', 2),
(206, 'CARHUAZ', 2),
(207, 'CARLOS F. FITZCARRALD', 2),
(208, 'CASMA', 2),
(209, 'CORONGO', 2),
(210, 'HUARI', 2),
(211, 'HUARMEY', 2),
(212, 'HUAYLAS', 2),
(213, 'MARISCAL LUZURIAGA', 2),
(214, 'OCROS', 2),
(215, 'PALLASCA', 2),
(216, 'POMABAMBA', 2),
.
.
.
(2503, 'PADRE ABAD', 25),
(2504, 'PURUS', 25);

Insertando los distritos del Perú:

INSERT INTO `distrito` (`distrito_codigo`, `distrito_nombre`, `provincia_codigo`) VALUES

(10101, 'CHACHAPOYAS', 101),
(10102, 'ASUNCIÓN', 101),
(10103, 'BALSAS', 101),
(10104, 'CHETO', 101),

```
(10105, 'CHILIQUIN', 101),
(10106, 'CHUQUIBAMBA', 101),
(10107, 'GRANADA', 101),
(10108, 'HUANCAS', 101),
(10109, 'LA JALCA', 101),
(10110, 'LEIMEBAMBA', 101),
(10111, 'LEVANTO', 101),
(10112, 'MAGDALENA', 101),
(10113, 'MARISCAL CASTILLA', 101),
(10114, 'MOLINOPAMPA', 101),
(10115, 'MONTEVIDEO', 101),
(10116, 'OLLEROS', 101),
(10117, 'QUINJALCA', 101),
(10118, 'SAN FRANCISCO DE DAG', 101),
(10119, 'SAN ISIDRO DE MAINO', 101),
(10120, 'SOLOCO', 101),
(10121, 'SONCHE', 101),
(10201, 'LA PECA', 102),
(10202, 'ARAMANGO', 102),
(10203, 'COPALLIN', 102),
(10204, 'EL PARCO', 102),
(10205, 'IMAZA', 102),
(10301, 'JUMBILLA', 103),
(10302, 'CHISQUILLA', 103),
(10303, 'CHURUJA', 103),
(10304, 'COROSHA', 103),
(10305, 'CUISPES', 103),
(10306, 'FLORIDA', 103),
(10307, 'JAZAN', 103),
(10308, 'RECTA', 103),
(10309, 'SAN CARLOS', 103),
(10310, 'SHIPASBAMBA', 103),
(10311, 'VALERA', 103),
(10312, 'YAMBRASBAMBA', 103),
(10401, 'NIEVA', 104),
(10402, 'EL CENEPA', 104),
(10403, 'RIO SANTIAGO', 104),
(10501, 'LAMUD', 105),
(10502, 'CAMPORREDONDO', 105),
(10503, 'COCABAMBA', 105),
(10504, 'COLCAMAR', 105),
(10505, 'CONILA', 105),
(10506, 'INGUILPATA', 105),
(10507, 'LONGUITA', 105),
(10508, 'LONYA CHICO', 105),
(10509, 'LUYA', 105),
(10510, 'LUYA VIEJO', 105),
(10511, 'MARIA', 105),
(10512, 'OCALLI', 105),
.
```

```
.
.
(250204, 'YURUA', 2502),
(250301, 'PADRE ABAD', 2503),
(250302, 'IRAZOLA', 2503),
(250303, 'CURIMANA', 2503),
(250401, 'PURUS', 2504);
```

Para probar la conexión se crea el formulario que se muestra en la IMAGEN Nº07-0011, el cual permitirá listar las provincias en una **jTable**, ingresar nuevas provincias, modificar datos de provincias, eliminar provincias, etc.

IMAGEN Nº07-0011: Diseño del formulario de gestion de provincias

En NetBeans IDE 16, en un proyecto creado se agrega el formulario “JFrame Form” con los siguientes controles: jTextField(1), jButton(2,4), jTable(4)

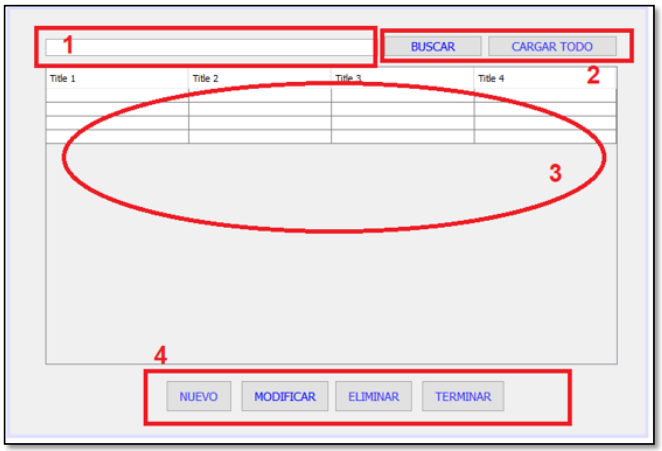


IMAGEN Nº07-0011: Diseño del formulario de gestion de provincias

Para que se cargue las provincias en el jTable desde la tabla “provincia” de la base de datos en MySQL se hace la conexión a la base de datos “sistema_electoral” y luego hacer consultas a la tabla “provincia” unido con la tabla “region”, para ello se crea el cargarProvincia() en la clase del formulario. Para hacer la conexión se utiliza la Clase “Conexion”.

Método para cargar las provincias de la tabla “provincias”

En el **executeQuery**() lo pasamos como cadena la consulta para sacar todas las provincias con sus respectivas regiones:

select provincia_codigo, provincia_nombre, region_nombre from provincia as p left join region as r on p.region_codigo=r.region_codigo

```
public  void cargarProvincia()
{
    Conexion cn1=new Conexion();
    Connection cn= cn1.conexion();

    try {
```

```
        Statement   s=cn.createStatement();
ResultSet     r=s.executeQuery("select  provincia_codigo,  provincia_nombre,
region_nombre  from  provincia  as  p  left  join  region  as  r  on
p.region_codigo=r.region_codigo");

        while(r.next())
        {
          Object
bien[]={r.getString("provincia_Codigo"),r.getString("provincia_nombre"),
                        r.getString("region_nombre")};

          modeloProvincia.addRow(bien);
        }


        }
    catch (Exception ex)
        {
         System.out.println("Error,  no  se  ha  podido  cargar  SQL  Server  JDBC
Driver");
        }

   }
```

El cual debe ser llamado al momento de iniciar todos los controles del formulario.

```
package sisonpe;
import Classes.Conexion;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumnModel;

public class frmProvincia extends javax.swing.JFrame
{
    public DefaultTableModel modeloProvincia;
    public frmProvincia()
    {
        initComponents();

        String []columnaNombre={"CODIGO","NOMBRE","REGION"};
        String [][] dato={};
        modeloProvincia=new DefaultTableModel(dato,columnaNombre);
        grdProvincia.setModel(modeloProvincia);

        TableColumnModel columnModel = grdProvincia.getColumnModel();
        columnModel.getColumn(0).setPreferredWidth(10);
        columnModel.getColumn(1).setPreferredWidth(200);
        columnModel.getColumn(2).setPreferredWidth(200);

        cargarProvincia();
```

IMAGEN Nº07-0012: Diseño del formulario de gestion de provincias

Configuración del jTable para que se cargue las provincias, para lo cual se instancia la clase DefaulTableModel y se asigna al jTable con su método setModel():

```
public final class frmProvincia extends javax.swing.JFrame
{
    //se define el modeloProvincia de tipo DefaultTableModel
   //para relacionar con el jTable
public DefaultTableModel modeloProvincia;

   public frmProvincia()
   {
      initComponents();
      String []columnaNombre={"CODIGO","NOMBRE","REGION","REG
CODIGO"};
      String [][] dato={};

  //se instancia la clase DefaultTableModel
  //pasándolo como parámetros el arreglo vacio y los nombres
  //de las columnas
      modeloProvincia=new DefaultTableModel(dato,columnaNombre);

  //al jTable se le asigna el modeloProvincia
      grdProvincia.setModel(modeloProvincia);

  //Se define el tamaño de las columnas
      TableColumnModel columnModel = grdProvincia.getColumnModel();
      columnModel.getColumn(0).setPreferredWidth(50);
      columnModel.getColumn(1).setPreferredWidth(200);
      columnModel.getColumn(2).setPreferredWidth(200);
      columnModel.getColumn(3).setPreferredWidth(50);
```

```
        //se llama al método cargarProvincia()
            cargarProvincia();
        }
    .
    .
    .
    }
```

Código del btnNuevo:

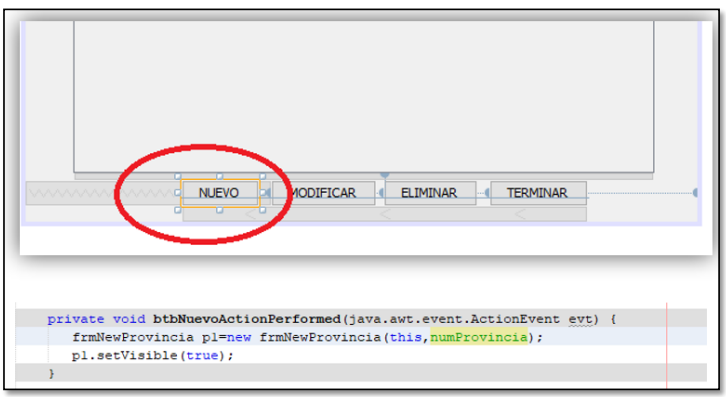


IMAGEN Nº07-0013: el evento clic del boton “nuevo”

7.3 CONEXIÓN DE JAVA CON SQL SERVER

La comunicación entre una aplicación Java y una base de datos **Microsoft SQL Server** se logra mediante la tecnología **JDBC** (Java Database Connectivity), que requiere un *driver* específico.

1. El Conector: JDBC para SQL Server

Para que Java pueda hablar con SQL Server, se necesita el *driver* oficial de Microsoft: el **Microsoft JDBC Driver para SQL Server**.

- ✓ **Nombre del Driver:** Se conoce formalmente como **mssql-jdbc**.
- ✓ **Función:** Este archivo .jar se encarga de traducir las llamadas a la API de Java (JDBC) en el protocolo de comunicación de SQL Server (TDS - Tabular Data Stream).
- ✓ **Descarga e Inclusión:** Al igual que con MySQL, el archivo .jar debe descargarse desde el sitio web de Microsoft e **incluirse como una librería** en las dependencias de tu proyecto Java (en tu IDE: NetBeans, Eclipse, IntelliJ). Descarga de Microsoft JDBC Driver para SQL Server desde: https://learn.microsoft.com/es-es/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver17

Luego agregar al proyecto en la parte de librería tal como se muestra en la IMAGEN Nº07-0014

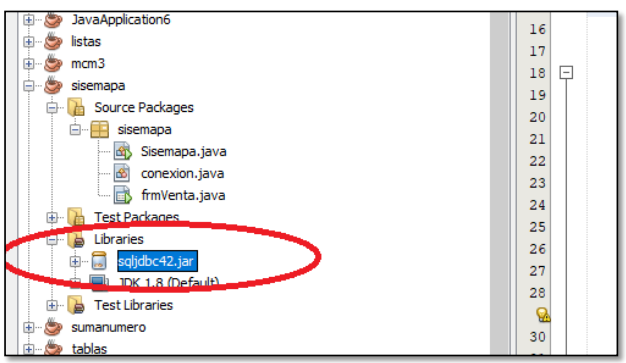


IMAGEN Nº07-0014: el evento clic del boton “nuevo”

2. Parámetros de Conexión

Al momento de escribir el código Java, debes especificar la ubicación y las credenciales del servidor de SQL Server.

Parámetro	Descripción	Valor Típico
Driver Class	La clase Java que se carga para iniciar la comunicación.	com.microsoft.sqlserver.jdbc.SQLServerDriver
URL de Conexión	La ruta al servidor, el puerto y el nombre de la base de datos.	jdbc:sqlserver://[nombre_servidor]:1433;databaseName=[nombre_db]
Usuario	Credenciales de usuario de SQL Server.	sa (o un usuario personalizado)
Contraseña	Contraseña del usuario.	******

Clase de conexión para SQLServer "Conexion"

La clase **conexion**, ubicada en el paquete sisemapa, tiene como objetivo principal encapsular y centralizar la lógica para **establecer una conexión con una base de datos SQL Server** utilizando el *driver* JDBC de Microsoft.

Lo más notable de esta clase es su método para obtener las credenciales:

1. **Lectura de Configuración Externa:** En lugar de codificar el usuario y la contraseña directamente en el código fuente (lo cual es inseguro), la clase lee estos datos (usuario, clave y nombre de la base de datos) desde un archivo de texto llamado **conf.txt** ubicado en la ruta **C://Windows/conf.txt**.
2. **Formulación de la URL:** Una vez que obtiene las credenciales, construye la **URL de conexión JDBC** completa, incluyendo los parámetros de conexión necesarios para SQL Server.
3. **Establecimiento de la Conexión:** Finalmente, utiliza el método DriverManager.getConnection() para abrir la conexión real con la base de datos.
4. **Devolución:** El método conexion() devuelve un objeto de tipo **Connection**, que es el recurso que la aplicación principal usará para ejecutar todas las consultas SQL (SELECT, INSERT, etc.).

Esta estructura permite actualizar fácilmente las credenciales del servidor sin necesidad de recompilar el código Java.

Código completo de la Clase "conexión"

```
//=========================================================
      // Clase conexión
//=========================================================
      package siselectoral;

      // Importa clases para leer datos de archivos (para el archivo de configuración)
      import java.io.BufferedReader;
      import java.io.FileReader;
      import java.io.IOException;


      // Importa todas las clases de JDBC para la conexión a la base de datos
      import java.sql.*;
      // Aunque se importó, el JOptionPane no se usa en el código actual
      import javax.swing.JOptionPane;


      public class conexion
      {
         // Variables de instancia para almacenar los datos leídos
         // del archivo de configuración
         String usuario; // Almacenará el nombre de usuario de la base de datos
         String clave;   // Almacenará la contraseña de la base de datos
         String bd;      // Almacenará el nombre de la base de datos (Database Name)
         String url;     // Almacenará la URL de conexión completa para JDBC


         // Método que intenta establecer la conexión y
        //  devuelve un objeto Connection
         public Connection conexion() //throws IOException
         {
            Connection cn = null; // Inicializa el objeto de conexión como nulo


            String cadena; // (Variable no utilizada en el código, pero declarada)
```

```
// Ruta absoluta del archivo que contiene las credenciales de conexión
String dir = "C://Windows/conf.txt";

/* ---------------------------------------------------------------- */
/* PRIMER BLOQUE try-catch: Lectura del Archivo de Configuración */
/* ---------------------------------------------------------------- */
try {
   // Abre el archivo para lectura
   FileReader f = new FileReader(dir);
   // Crea un buffer para leer el archivo línea por línea de manera eficiente
   BufferedReader b = new BufferedReader(f);

   // Lee la primera línea y la asigna como usuario
   usuario = b.readLine();
   // Lee la segunda línea y la asigna como clave (contraseña)
   clave = b.readLine();
   // Lee la tercera línea y la asigna como nombre de la BD
   bd = b.readLine();

   // Lee la cuarta línea (debe contener la parte inicial de la URL:
    // jdbc:sqlserver://servidor:puerto;)
   // y construye la URL de conexión completa
    // concatenando los parámetros
   url = b.readLine() + "databaseName=" + bd + ";user=" + usuario +
";password=" + clave + ";";

   // Cierra el buffer y libera los recursos del archivo
   b.close();
}
// Captura excepciones si hay problemas de lectura o
// si el archivo no existe
```

```
        catch (IOException ex)
        {
            System.out.println("no se pudo leer archivo");
        }

        /* ---------------------------------------------------------- */
        /* SEGUNDO BLOQUE try-catch: Establecimiento de la Conexión */
        /* ---------------------------------------------------------- */
        try {
            // Carga dinámicamente el driver JDBC de SQL Server en memoria

            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newIns
            tance();

            // Establece la conexión utilizando la URL completa que se construyó
            // con las credenciales leídas del archivo.
            cn = DriverManager.getConnection(url);
        }
        // Captura cualquier error que ocurra al cargar el driver o
        // al intentar conectar
        catch (Exception ex)
        {
            System.out.println("Error, no se ha podido cargar SQL Server JDBC
Driver o conectar.");
            // También se podría usar ex.printStackTrace() para ver
            // el detalle del error
        }

        // Devuelve el objeto Connection establecido (o null si falló)
        return cn;
    }
}
```

Configuración de Conexión Remota en SQL Server

Para permitir que aplicaciones externas (como tu programa Java) o computadoras diferentes se conecten a la instancia de SQL Server, es fundamental configurar el protocolo **TCP/IP** y asegurar que el puerto esté fijo.

1. Habilitar el Protocolo TCP/IP

El protocolo TCP/IP es el estándar de Internet que permite la comunicación a través de redes. Por defecto, muchas instalaciones de SQL Server solo permiten conexiones locales.

- ✓ **Herramienta:** Utiliza el **SQL Server Configuration Manager**.
- ✓ **Ruta:** Ve a **SQL Server Network Configuration** y luego a **Protocols for [Nombre de la Instancia]** (ej. SQLEXPRESS).
- ✓ **Acción:** Asegúrate de que el protocolo **TCP/IP** esté **Habilitado**.

En Sql Server Configuration Manager, tal como se muestra en la IMAGEN Nº07-0015 y Nº07-0015.

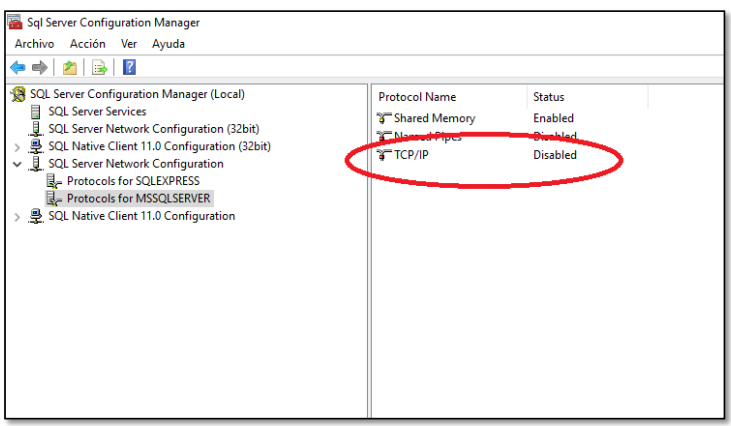


IMAGEN Nº07-0015: Configuracion del TCP/IP

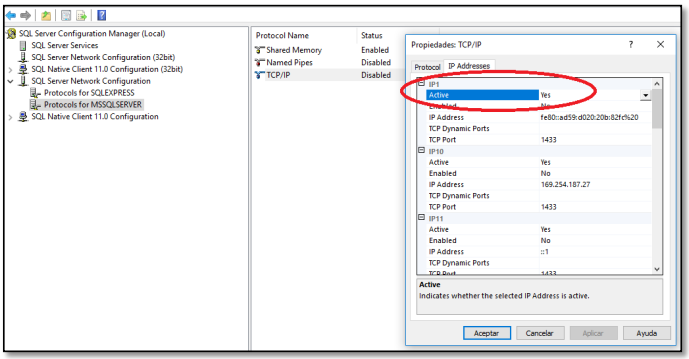


IMAGEN Nº07-0016: Configuracion del TCP/IP, activar

Debería quedar el servicio en “Enabled”, tal como se muestra en la IMAGEN Nº07-0017.

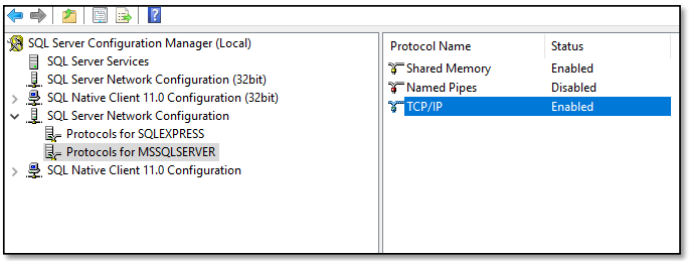


IMAGEN Nº07-0017: Configuracion del TCP/IP, activado

2. Reiniciar el Servicio

Tras habilitar el protocolo, el servicio de SQL Server debe ser reiniciado para que los cambios surtan efecto, IMAGEN Nº07-0018.

- ✓ **Herramienta:** En el mismo **SQL Server Configuration Manager**.
- ✓ **Ruta:** Ve a **SQL Server Services**.

Acción: Clic derecho sobre el **SQL Server ([Nombre de la Instancia])** y selecciona **Reiniciar (Restart)**.

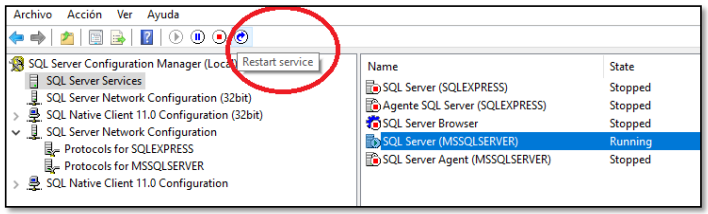


IMAGEN Nº07-0018: Configuracion del TCP/IP, activado

3. Configurar el Puerto Fijo (Estático)

Es crucial asignar un puerto estático (fijo) en lugar de uno dinámico. El puerto estándar para SQL Server es el **1433**, IMAGEN Nº07-0019 y 07-0020.

- **Herramienta:** Dentro del **SQL Server Configuration Manager**, en las propiedades del protocolo **TCP/IP**.
- **Acción:** Ve a la pestaña **IP Addresses**. Busca las secciones de las interfaces de red relevantes (generalmente **IPAll**):

✓**Puertos Dinámicos TCP (TCP Dynamic Ports):** Debe dejarse **vacío** o en **0**.

✓**Puerto TCP (TCP Port):** Configúralo con el valor **1433**.

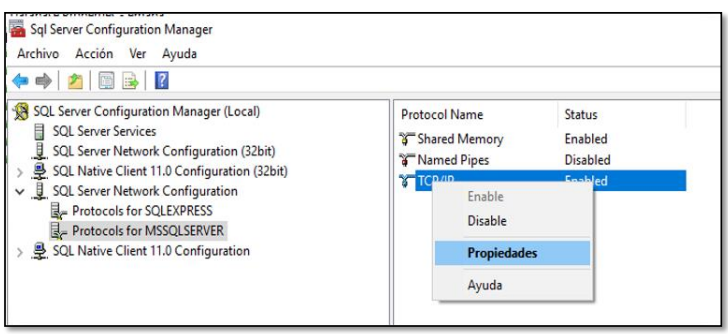


IMAGEN Nº07-0019: Configuracion del puerto TCP/IP, 1433

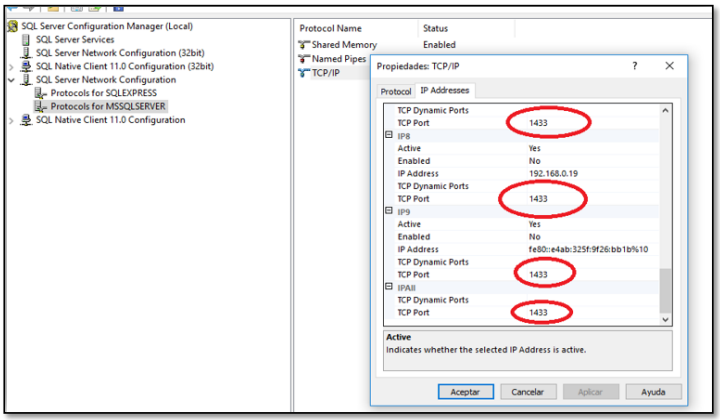


IMAGEN Nº07-0020: Configuracion del puerto TCP/IP, 1433

Nota sobre Conexión Remota y Nombre de Instancia

Cuando la conexión se establece a través del puerto estándar **1433** (o cualquier puerto estático asignado) y el servidor de SQL Server está a la escucha en ese puerto, la conexión se simplifica:

PARA CONECTAR DESDE UNA COMPUTADORA A LA BASE DE DATOS QUE ESTÁ EN OTRA COMPUTADORA EN SQL SERVER, A VECES SE UTILIZA ESTA CONFIGURACIÓN.

YA NO SE PONE EL NOMBRE DE LA INSTANCIA (ej. SQLEXPRESS), SOLO EL PUERTO POR DEFECTO (1433) Y AHÍ YA SE CONECTA.

Esto se debe a que, si se utiliza el puerto estándar 1433, el **SQL Server Browser Service** (que se usa para resolver nombres de instancias) no es necesario, y la aplicación (Java en este caso) puede dirigirse directamente a la IP del servidor más el puerto, IMAGEN Nº07-0021 y Nº07-0022

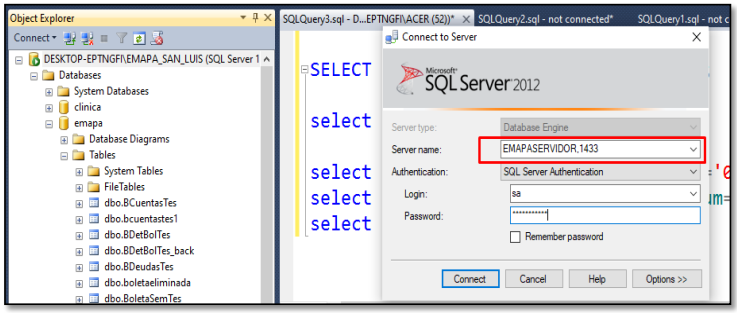


IMAGEN Nº07-0021: Conexión entre SQLServer de distintas computadoras

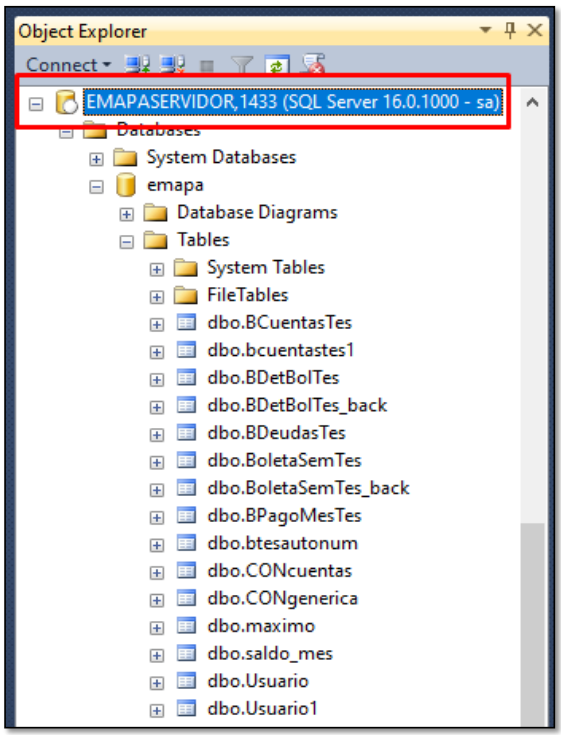


IMAGEN Nº07-0021: Conexión entre SQLServer de distintas computadoras

EJEMPLO N°06-002

Del modelo del EJEMPLO N°06-001, realizar el formulario que se muestra en la IMAGEN Nº07-0022, para gestionar jirones.

IMAGEN Nº07-0022: Formulario de gestion de jirones

Al hacer clic en el boton “NUEVO” sale el formulario para agregar datos de un nuevo jiron, tal como se muestra en la IMAGEN Nº07-0023.

Al hacer clic en el botón “MODIFICAR”, si se seleccionó en la tabla un registro saldrá el formulario con los dos campos del registro seleccionado, tal como se muestra en la IMAGEN Nº07-0024.

En la parte de la etiqueta JIRON, se ingresa una palabra o nombre de jirón y al presionar la tecla espaciadora se filtra y se muestra los jirones que tengan dicha palabra.

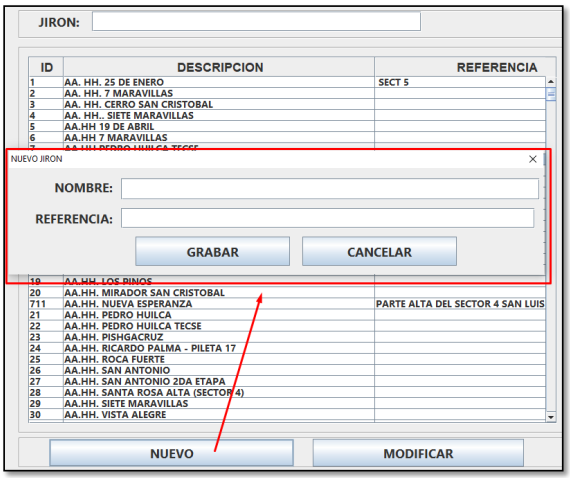


IMAGEN Nº07-0023: Formulario al hacer clic en el boton “NUEVO”

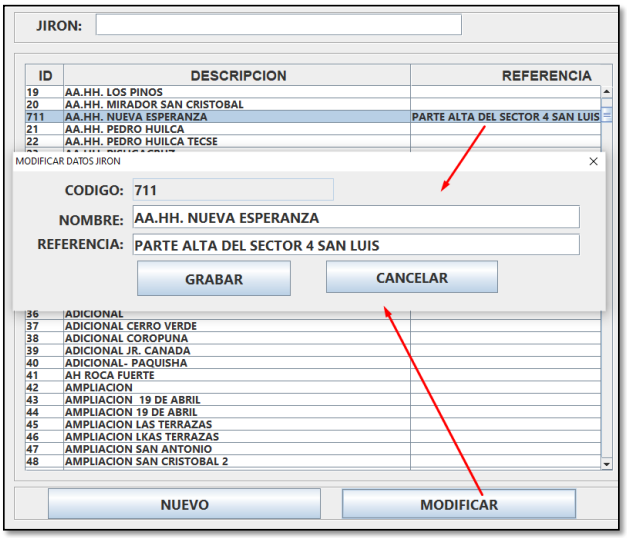


IMAGEN Nº07-0024: Formulario al hacer clic en el boton “MODIFICAR”

DESARROLLO

En modo diseño en NetBeans se agrega un JPanel, tal como se muestra en IMAGEN Nº07-0025. Se utiliza este contenedor porque será parte de un sistema en donde el formulario se cargará sobre un JFrame.

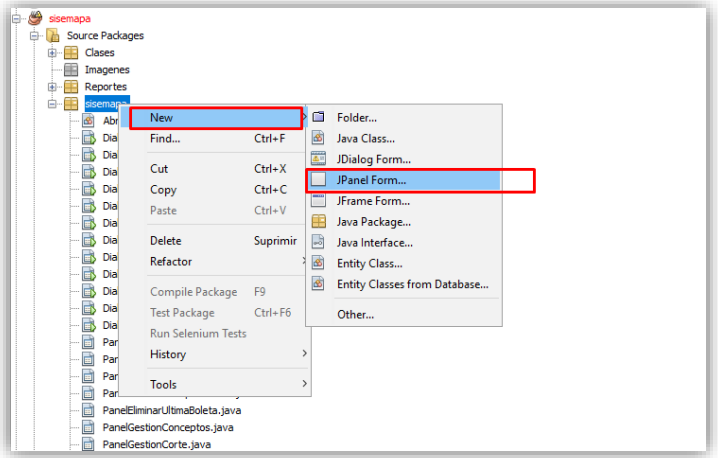


IMAGEN Nº07-0025: Clic derecho sobre el proyecto para agregar un JPanel

Agregando todos los controles necesarios al contenedor se tendrá el diseño que se muestra en la IMAGEN Nº07-0026.

Controles agregados:

- ✓ Label
- ✓ Text Field
- ✓ Button
- ✓ JTable
- ✓ Panel

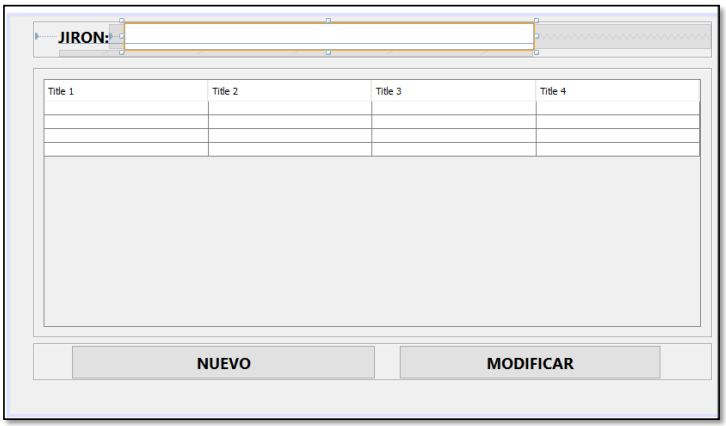


IMAGEN Nº07-0026: Formulario en modo diseño de gestion de jirones

Clase "PanelJiron"

La clase **PanelJiron** es un componente de interfaz gráfica (**javax.swing.JPanel**) que muestra una lista de jirones en una tabla y proporciona funcionalidad básica para interactuar con esos datos.

Funcionalidad Clave:

1. **Modelo de Tabla Personalizado (DefaultTableModel):** Utiliza un modelo de tabla (modeloJiron) para estructurar los datos y, lo más importante, **evitar que el usuario edite las celdas directamente** en la tabla (isCellEditable siempre devuelve false).
2. **Diseño y Configuración (initDiseño()):** Configura visualmente la tabla (**grdJiron**), estableciendo los nombres de las columnas (ID, DESCRIPCION, REFERENCIA), fijando el ancho de cada columna para asegurar una visualización consistente y dando formato al encabezado.
3. **Carga Inicial de Datos (cargarJiron()):** Conecta con la base de datos (a través de la clase conexion), limpia la tabla actual y ejecuta una consulta SQL para traer todos los registros de la tabla jiron, poblando la tabla grdJiron.

4. **Búsqueda Rápida (txtCadenaKeyTyped y filtrarZona()):** Permite al usuario buscar jirones en tiempo real. Cuando el usuario teclea en el campo de texto (txtCadena) y pulsa **espacio** (código ASCII 32), se activa la función filtrarZona, que ejecuta una consulta SQL con una cláusula WHERE para mostrar solo los jirones que coincidan.
5. **Operaciones CRUD Básicas:** Implementa botones para **NUEVO** (jButton2ActionPerformed) y **MODIFICAR** (jButton3ActionPerformed), los cuales abren formularios modales específicos (frmJironNew y frmJironUpdate) para realizar las operaciones.

Código completo "PanelJiron"

```
package siselectoral;
// Importaciones de clases para manejo de interfaz gráfica y base de datos
import java.awt.Font;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumnModel;
import javax.swing.table.*;

// La clase principal es un Panel (JPanel) que contendrá la interfaz
public class PanelJiron extends javax.swing.JPanel {

    // Modelo estático que se usará para llenar la tabla JTable (grdJiron)
    public static DefaultTableModel modeloJiron;

    // Constructor de la clase
    public PanelJiron(String DNI) {
        initComponents(); // Método generado automáticamente
```

```
                                    // por el IDE para inicializar componentes visuales
        initDiseño();     // Llama a la función para configurar el diseño de la tabla
    }


// ----------------------------------------------------------------------------------------------------
// MÉTODO: initDiseño()
// Configura el modelo de la tabla, los tamaños de columna y el diseño del
encabezado.
// ----------------------------------------------------------------------------------------------------
public void initDiseño(){
    String []columnaNombre={"ID","DESCRIPCION","REFERENCIA"};
    String [][] dato={}; // Matriz de datos inicial vacía


    // Crea el modelo de la tabla, personalizando el método isCellEditable.
    modeloJiron = new DefaultTableModel(dato,columnaNombre)
    {
        @Override
        public boolean isCellEditable (int fila, int columna) {
            // Este método personalizado garantiza que ninguna
             // celda de la tabla pueda ser editada por el usuario.
            return false;
        }
    };


    // Asigna el modelo personalizado a la JTable (grdJiron)
    grdJiron.setModel(modeloJiron);


    // --- Configuración de Columnas ---
    TableColumnModel columnModel = grdJiron.getColumnModel();


    // Columna 0 (ID)
    columnModel.getColumn(0).setPreferredWidth(50);
```

```
    columnModel.getColumn(0).setMinWidth(50);
    columnModel.getColumn(0).setMaxWidth(50);
    columnModel.getColumn(0).setResizable(false); // No permite
                                                  //redimensionar al usuario

    // Columna 1 (DESCRIPCION/NOMBRE)
    columnModel.getColumn(1).setPreferredWidth(450);
    columnModel.getColumn(1).setMinWidth(450);
    columnModel.getColumn(1).setMaxWidth(450);
    columnModel.getColumn(1).setResizable(false);

    // Columna 2 (REFERENCIA)
    columnModel.getColumn(2).setPreferredWidth(350);
    columnModel.getColumn(2).setMinWidth(350);
    columnModel.getColumn(2).setMaxWidth(350);
    columnModel.getColumn(2).setResizable(false);

    // --- Personalización del Encabezado ---
    JTableHeader header = grdJiron.getTableHeader();
    // Establece la fuente y estilo del encabezado (negrita, tamaño 18)
    header.setFont(new Font("Bold", Font.BOLD, 18));

    // Llama al método estático para cargar los datos
   // iniciales de la base de datos
    cargarJiron();
}

// ------------------------------------------------------------------------------------------------
// MÉTODO: cargarJiron() (Estático)
// Consulta la tabla 'jiron' de la BD y llena la JTable.
// ------------------------------------------------------------------------------------------------
public static void cargarJiron(){
```

```
// Establece la conexión a la base de datos utilizando la clase de conexión
conexion cn1 = new conexion();
Connection cn = cn1.conexion();

// --- Limpieza de la Tabla ---
// Recorre el modelo desde el final y elimina todas las filas existentes
for (int i = modeloJiron.getRowCount() -1; i >= 0; i--)
    modeloJiron.removeRow(i);

try {
    Statement s = cn.createStatement();
    // Consulta SQL para seleccionar ID, nombre y referencia,
        // ordenando por nombre
    String CadSQL = "SELECT id_jiron,nombre,referencia FROM jiron order by
nombre";
    ResultSet r = s.executeQuery(CadSQL);

    // Itera sobre los resultados de la consulta
    while (r.next())
    {
        // Crea un array de Strings con los datos de la fila actual del ResultSet
        String
dataJiron[]={r.getString("id_jiron"),r.getString("nombre"),r.getString("referencia")};
        // Agrega la nueva fila al modelo de la tabla
        modeloJiron.addRow(dataJiron);
    }

    // --- Cierre de Conexiones ---
    // Es crucial cerrar los recursos en orden inverso al que se abrieron
    r.close();
    s.close();
```

```
            cn.close();
            cn1.conexion().close(); // Cierra cualquier otra conexión
                        //residual (posiblemente redundante)

        }
        catch (SQLException ex)
        {
            // Imprime cualquier error de SQL que pueda ocurrir
            System.out.println(ex);
        }
    }
    // ----------------------------------------------------------------------------------------------------
    // Métodos generados automáticamente por el IDE (initComponents, Handlers de eventos, etc.)
    // ----------------------------------------------------------------------------------------------------
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        // ... Código de inicialización de
            // componentes (no comentado por ser autogenerado) ...
    }
    // </editor-fold>

    private void txtCadenaActionPerformed(java.awt.event.ActionEvent evt) {
        // Manejador del evento ActionPerformed del campo
            // de texto (ejecutado al presionar Enter)
    }

    private void txtCadenaKeyReleased(java.awt.event.KeyEvent evt) {
        // Se ejecuta cada vez que una tecla es liberada en el campo txtCadena
        // Convierte el texto ingresado a mayúsculas para la búsqueda
```

```
    txtCadena.setText(txtCadena.getText().toUpperCase());
}

private void txtCadenaKeyTyped(java.awt.event.KeyEvent evt) {
    // Se ejecuta cada vez que una tecla es presionada y
        // consumida por el componente
    char caracter = evt.getKeyChar();
    String where_proposicion = "";
    String cadena = txtCadena.getText();

    // Prepara la cláusula WHERE para buscar coincidencias
        //  parciales en el nombre
    where_proposicion = " nombre like '%" + cadena + "%' ";

    int a = caracter;
    // Si el caracter ingresado es un espacio (código ASCII 32)
    if(a == 32)
    {
       // Llama a la función de filtrado con la proposición WHERE construida
       filtrarZona(where_proposicion);
    }
    // Nota: Este filtrado solo ocurre al presionar ESPACIO.
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // Manejador del botón NUEVO
    // Crea y muestra un nuevo formulario modal para insertar un Jirón
    frmJironNew n = new frmJironNew(null,true);
    n.setLocationRelativeTo(this);
    n.setVisible(true);
}
```

```
    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        // Manejador del botón MODIFICAR
        int f = grdJiron.getSelectedRow(); // Obtiene el índice de la
                                                    // fila seleccionada

        // Obtiene los datos de la fila seleccionada para
            // pasarlos al formulario de edición
        String id_jiron = grdJiron.getValueAt(f, 0).toString();
        String nombre = grdJiron.getValueAt(f, 1).toString();
        String referencia = grdJiron.getValueAt(f, 2).toString();

        // Crea y muestra un nuevo formulario modal
            // para modificar el Jirón seleccionado
        frmJironUpdate n = new
frmJironUpdate(null,true,f,id_jiron,nombre,referencia);
        n.setLocationRelativeTo(this);
        n.setVisible(true);
    }

// ----------------------------------------------------------------------------------------------------
// MÉTODO: filtrarZona(String where_proposicion)
// Realiza una consulta SQL filtrada según la proposición WHERE proporcionada.
// ----------------------------------------------------------------------------------------------------
public void filtrarZona(String where_proposicion){
    // Conexión a base de datos
    conexion cn1 = new conexion();
    Connection cn = cn1.conexion();

    // Limpia la tabla antes de cargar los nuevos resultados del filtro
    for (int i = modeloJiron.getRowCount() -1; i >= 0; i--)
        modeloJiron.removeRow(i);
```

```
    try {
        Statement s = cn.createStatement();

        // Construye la consulta SQL completa usando la proposición WHERE
        String CadSQL = "select id_jiron,nombre, referencia " +
                        " from jiron where " + where_proposicion +
                        " order by nombre ";

        ResultSet r = s.executeQuery(CadSQL);

        // Itera sobre los resultados de la consulta filtrada y agrega filas al modelo
        while (r.next())
        {
            String data_jiron[] =
{r.getString("id_jiron"),r.getString("nombre"),r.getString("referencia")};
            modeloJiron.addRow(data_jiron);
        }

        // Cierra las conexiones
        r.close();
        s.close();
        cn.close();
        cn1.conexion().close();

    }
    catch (SQLException ex)
    {
        System.out.println(ex);
    }
}
}
```

Código completo clase "frmJironNew"

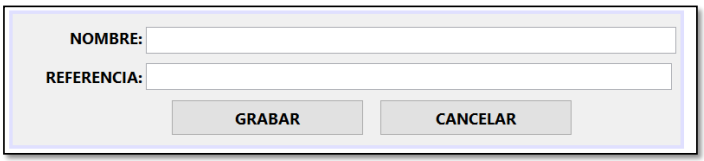


IMAGEN Nº07-0027: Formulario en modo diseño para ingresar nuevo jiron

La clase **frmJironNew** es un cuadro de diálogo modal (**JDialog**) que cumple la función de ingresar nuevos datos de "Jirón" (nombre de una calle) en el sistema.

Funcionalidad Clave:

1. **Formulario Modal:** Al ser un JDialog modal, este formulario bloquea la interacción con la ventana principal (PanelJiron) mientras está abierto, forzando al usuario a completar la acción (grabar o cancelar).
2. **Validación Básica:** Antes de intentar la inserción en la base de datos, el método jButton2ActionPerformed (botón **GRABAR**) verifica que los campos txtNombre y txtReferencia no estén vacíos. Si lo están, muestra una alerta.
3. **Inserción de Datos (GRABAR):**

✓ Si los campos son válidos, establece una conexión a la base de datos a través de la clase **conexion**.

✓ Ejecuta una sentencia SQL de tipo **INSERT** que toma los valores directamente de los campos de texto (txtNombre y txtReferencia).

✓ Cierra la conexión y el formulario (this.dispose()).

4. **Actualización de la Vista Padre:** Después de grabar exitosamente, llama al método estático **PanelJiron.cargarJiron()**. Esta es una acción fundamental, ya que garantiza que el PanelJiron (la tabla principal que muestra los datos) se actualice automáticamente con el nuevo registro insertado sin que el usuario tenga que recargar manualmente.
5. **Formato de Entrada:** Los eventos KeyReleased de los campos de texto fuerzan el contenido a **MAYÚSCULAS** (toUpperCase()), estandarizando el formato de los datos que se guardan en la base de datos.

Código completo "frmJironNew"

```
package siselectoral;
// Importaciones para manejo de base de datos SQL
import java.sql.Connection;
import java.sql.ResultSet; // No se usa en este formulario, pero está importado
import java.sql.SQLException;
import java.sql.Statement;
// Importación para mostrar mensajes de alerta
import javax.swing.JOptionPane;


// La clase hereda de JDialog, creando un formulario modal
public class frmJironNew extends javax.swing.JDialog {


    // Constructor que define el formulario como modal
    public frmJironNew(java.awt.Frame parent, boolean modal) {
        super(parent, modal); // Llama al constructor padre JDialog, estableciendo
la modalidad
        initComponents(); // Inicializa los componentes visuales
    }


    @SuppressWarnings("unchecked")


    private void initComponents() {
        // ... Código de inicialización de componentes
            // visuales (etiquetas, campos, botones) ...

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE
);
        setTitle("NUEVO JIRON"); // Título de la ventana
        setResizable(false);    // Impide que el usuario redimensione la ventana


        // ... Configuración de listeners (manejadores de eventos) para
            // txtNombre, txtReferencia, jButton1 y jButton2 ...
```

```
    // ... Código de layout y empaquetamiento ...
    pack();
}

/* ------------------------------------------------------------ */
/* MANEJADORES DE EVENTOS DE BOTONES */
/* ------------------------------------------------------------ */

// Método asociado al botón CANCELAR (jButton1)
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose(); // Cierra y libera los recursos del formulario modal
}

// Método asociado al botón GRABAR (jButton2)
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

    // 1. Validación de campos: Verifica que los campos de
        //texto no estén vacíos
    if(txtNombre.getText().length() == 0 || txtReferencia.getText().length() == 0){
        JOptionPane.showMessageDialog(null,"Ingrese nombre o referencia",
        "Alerta",JOptionPane.WARNING_MESSAGE );
    } else {
        // --- 2. Bloque de Inserción ---

        // Establece la conexión a la base de datos
        conexion cn1 = new conexion();
        Connection cn = cn1.conexion();

        try {
            Statement s = cn.createStatement();
```

```
            // Sentencia SQL de INSERCIÓN (INSERT INTO)
            // Toma los valores de los JTextField
            String cadSQL = "insert into jiron(nombre,referencia) " +
                            "values('" + txtNombre.getText() + "','" +
txtReferencia.getText() + "')";

            s.execute(cadSQL); // Ejecuta la consulta SQL (INSERT)

            s.close(); // Cierra el Statement
            cn.close(); // Cierra la conexión

        } catch (SQLException ex) {
            System.out.println(ex);
        }

        // --- 3. Finalización y Actualización ---

        this.dispose(); // Cierra el formulario modal

        // Llama al método estático en la clase PanelJiron para refrescar la tabla
        PanelJiron.cargarJiron();
    }
}

/* ------------------------------------------------------------ */
/* MANEJADORES DE EVENTOS DE TECLADO */
/* ------------------------------------------------------------ */

// Se ejecuta al liberar una tecla en el campo Nombre
private void txtNombreKeyReleased(java.awt.event.KeyEvent evt) {
    // Convierte el texto ingresado a mayúsculas para estandarizar el dato
    txtNombre.setText(txtNombre.getText().toUpperCase());
```

```
    }

    // Se ejecuta al liberar una tecla en el campo Referencia
    private void txtReferenciaKeyReleased(java.awt.event.KeyEvent evt) {
        // Convierte el texto ingresado a mayúsculas
        txtReferencia.setText(txtReferencia.getText().toUpperCase());
    }
}
```

Código completo clase "frmJironUpdate"

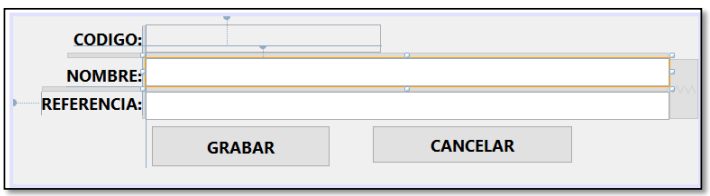


IMAGEN Nº07-0028: Formulario en modo diseño para ingresar nuevo jiron

La clase **frmJironUpdate** es un cuadro de diálogo modal (JDialog) utilizado para editar los campos nombre y referencia de un registro de Jirón específico.

Funcionalidad Clave:

1. **Recepción de Datos:** El constructor recibe múltiples parámetros (fila1, id_jiron, nombre, referencia), que son los datos del registro seleccionado en la tabla principal (PanelJiron). Inmediatamente, estos valores se asignan a sus respectivos campos de texto, pre-cargando el formulario para la edición.

✓ El campo **txtId_jiron** (código) está configurado como **no editable** (setEditable(false)) para asegurar que la clave primaria no sea modificada, garantizando la integridad de la base de datos.

2. **Lógica de Modificación (GRABAR):** El método jButton1ActionPerformed implementa la lógica de **UPDATE**:

- ✓ Verifica que los campos nombre y referencia no estén vacíos.
- ✓ Establece la conexión a la base de datos.
- ✓ Construye y ejecuta una sentencia SQL de tipo **UPDATE**, utilizando el valor inmutable de txtId_jiron en la cláusula WHERE para dirigirse al registro correcto.

3. **Refresco de Vista:** Tras una modificación exitosa, el formulario se cierra (this.dispose()) y llama al método estático **PanelJiron.cargarJiron()** para actualizar inmediatamente la tabla en la ventana principal, mostrando los cambios al usuario.
4. **Formato de Entrada:** Al igual que en el formulario de inserción, los eventos KeyReleased convierten el texto de txtNombre y txtReferencia a **MAYÚSCULAS**.

Código completo "frmJironNew"

```
package siselectoral;

// Importaciones para manejo de base de datos SQL
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
// Importación para mostrar mensajes de alerta
import javax.swing.JOptionPane;


// La clase hereda de JDialog, creando un formulario modal para la modificación
public class frmJironUpdate extends javax.swing.JDialog {


    // Variable de instancia para almacenar el índice de la fila seleccionada
   //(no usada en la lógica SQL, pero recibida)
    int fila;


    // Constructor con múltiples parámetros para recibir el registro seleccionado
    public frmJironUpdate(java.awt.Frame parent, boolean modal, int fila1, String
id_jiron, String nombre, String referencia) {
```

```
    super(parent, modal); // Llama al constructor padre,
                          //estableciendo la modalidad
    initComponents();     // Inicializa los componentes visuales

    // Asigna los valores recibidos a los campos del formulario
    fila = fila1;
    txtId_jiron.setText(id_jiron);     // Asigna el ID del jirón (clave primaria)
    txtNombre.setText(nombre);         // Asigna el nombre para edición
    txtReferencia.setText(referencia); // Asigna la referencia para edición
}

/* ------------------------------------------------------------------ */
/* MANEJADORES DE EVENTOS DE BOTONES */
/* ------------------------------------------------------------------ */

// Método asociado al botón GRABAR (jButton1) para actualizar el registro
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    // 1. Validación de campos: Verifica que los campos de texto
        // no estén vacíos
    if(txtNombre.getText().length() == 0 || txtReferencia.getText().length() == 0){
       JOptionPane.showMessageDialog(null,"Ingrese nombre o referencia",
       "Alerta",JOptionPane.WARNING_MESSAGE );
    } else {
       // --- 2. Bloque de Actualización (UPDATE) ---

       // Establece la conexión a la base de datos
       conexion cn1 = new conexion();
       Connection cn = cn1.conexion();

       try {
          Statement s = cn.createStatement();
```

```
            // Sentencia SQL de ACTUALIZACIÓN (UPDATE)
            // Modifica el nombre y referencia, usando el ID del jirón en el
                        // WHERE para apuntar al registro correcto
            String cadSQL = "update jiron set nombre='" + txtNombre.getText() +
                        "',referencia='" + txtReferencia.getText() + "' " +
                        "where id_jiron=" + txtId_jiron.getText();

            s.execute(cadSQL); // Ejecuta la sentencia SQL de actualización

            s.close(); // Cierra el Statement
            cn.close(); // Cierra la conexión

        } catch (SQLException ex) {
            // Captura y muestra errores de SQL
            System.out.println(ex);
        }

        // --- 3. Finalización y Actualización ---

        this.dispose(); // Cierra el formulario modal

        // Llama al método estático en la clase PanelJiron para
                // refrescar la tabla principal
        PanelJiron.cargarJiron();
    }

    // Se repite this.dispose() aquí, lo cual es redundante pero no causa error
    this.dispose();
}
```

```
    // Método asociado al botón CANCELAR (jButton2)
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose(); // Cierra el formulario modal
    }

    /* ------------------------------------------------------------ */
    /* MANEJADORES DE EVENTOS DE TECLADO (Formato) */
    /* ------------------------------------------------------------ */

    // Convierte el texto del Nombre a mayúsculas al liberar la tecla
    private void txtNombreKeyReleased(java.awt.event.KeyEvent evt) {
        txtNombre.setText(txtNombre.getText().toUpperCase());
    }

    // Convierte el texto de la Referencia a mayúsculas al liberar la tecla
    private void txtReferenciaKeyReleased(java.awt.event.KeyEvent evt) {
        txtReferencia.setText(txtReferencia.getText().toUpperCase());
    }

}
```

CAPÍTULO 8. CASO PRÁCTICO

Este capítulo aplica los conocimientos adquiridos en la gestión de bases de datos relacionales, ilustrando cómo la lógica de negocio se implementa eficientemente en la capa de datos (SQL) y cómo la interfaz de usuario gestiona estas operaciones desde la capa de aplicación (Java).

8.1 CASO 01: SISTEMA DE CONTROL DE PAGOS MENSUALES DE SERVICIO DE DOTACIÓN DE AGUA DE LA EMPRESA “PATA AMARILLA”.

El presente caso práctico consiste en el diseño e implementación del **Sistema de Gestión de Deudas y Control de Pagos Mensuales** para la empresa privada **"Pata Amarilla"**. La empresa administra los servicios de dotación de agua y control de alcantarillado para **3,500 usuarios** en diversos sectores populares, manejando la complejidad de **tarifas planas** y **tarifas especiales para piletas públicas**.

La solución debe ser robusta, multiusuario, y funcionalmente completa, abordando las necesidades operativas de sus diferentes áreas (Gerencia, Comercialización, Caja y Contabilidad).

Requerimientos Técnicos y de Infraestructura

1. **Motor de Base de Datos: SQL Server** (versión 2022 mínimo) instalado en un servidor centralizado.
2. **Capa de Aplicación:** Desarrollada en **Java (JDBC)**, diseñada para ser accedida por múltiples terminales conectadas en red local (15 puntos de acceso como minimo).
3. **Seguridad y Lógica:** El sistema debe implementar un **menú dinámico** basado en **roles y permisos** para garantizar la seguridad. Toda la **lógica de negocio crítica** (cálculos de deuda, registro de pagos, fraccionamientos y cortes) debe residir en la base de datos a través de **Procedimientos Almacenados**, asegurando la integridad transaccional.

Algunos Requerimientos Funcionales por Rol

Área de la Empresa	Rol Principal	Responsabilidades y Funcionalidades Clave
Caja	Cajero	Realizar **cobros**, **generar las deudas mensuales** de los usuarios, y acceso a reportes operativos (**ingreso diario, mensual, anual, registro de ventas**).
Comercialización	Administrador	**Ingreso de nuevos usuarios**, **gestión de deudas** (fraccionamientos, multas, notificaciones de recibos), y **gestión de cortes** por deuda.
Gerencia	Gerente	**Visualización de todos los reportes** consolidados (ingresos, pagos, listados de deuda, fraccionamientos) para la toma de decisiones.
Soporte/TI	Administrador del Sistema	**Gestión de conceptos de pago**, **gestión de usuarios del sistema**, y herramientas de **copia de seguridad**.

Módulos Clave a Implementar

El sistema debe cubrir todos los procesos administrativos, incluyendo:

- ✓ **Acceso al Sistema:** Pantalla de Logueo, Roles de Usuario y Menú Principal Dinámico.
- ✓ **Módulo ZONA:** Gestión de Zonas y Jirones.
- ✓ **Módulo USUARIOS:** Gestión completa de Usuarios, **Gestión de Deudas** y **Cortes**.
- ✓ **Módulo de Pagos:** Emisión, Anulación y Eliminación de Boletas, Ingreso de Pagos Anteriores.
- ✓ **Módulo REPORTES:** Reportes de Ingreso (acumulado, por concepto), Reporte de Venta, Reporte de Deuda por Sector y Listado de Usuarios al Día.

DESARROLLO

El diseño de la base de datos se basa en una arquitectura relacional desarrollada en **SQL Server**, estructurada para dar soporte a las complejas operaciones de gestión de pagos y seguridad de la empresa "Pata Amarilla". Inicialmente, el diseño se enfoca en tres áreas clave: la **seguridad** (gestionando **Roles** y **UsuarioSistema**), la **geografía** (con **Zona** y **Jiron** para organizar a los más de 3,500 clientes) y los **datos maestros** del servicio (**UsuarioServicio**,

diferenciando instalaciones domiciliarias de **piletas públicas**, y **ConceptoPago** para registrar las tarifas de agua, alcantarillado o multas). Esta fase sienta las bases para las transacciones financieras, asegurando la correcta identificación y tarificación de cada cliente.

La esencia del diseño reside en la gestión de las transacciones de deuda y pago, modeladas a través de tres entidades vinculadas: **Deuda**, **Boleta** y **DetalleBoleta**. La tabla **Deuda** es la fuente de verdad del inventario financiero de la empresa, donde se registra cada cargo pendiente (monto_original, fecha_vencimiento) asociado a un **Usuarios** y un **ConceptoPago**. Cuando un cliente realiza un pago, se genera un registro en la tabla **Boleta** (el comprobante de pago) y múltiples registros en **DetalleBoleta** que vinculan la **Boleta** con las filas de **Deuda** que se están cubriendo, permitiendo el control preciso de pagos parciales o deudas fraccionadas.

En el diseño de la base de datos tendremos:

1. El esquema conceptual
2. El esquema lógico
3. El esquema físico
4. Script de creación de las tablas en SQLServer generado por la herramienta Erwin

En el desarrollo de la interfaz tendremos:

2. Acceso al Sistema

- ✓ 2.1 Pantalla de Logueo
- ✓ 2.2 Roles de Usuario

3. Funcionalidades

- ✓ 3.1 Menú Principal
- ✓ 3.2 Modulo SISTEMA

- 3.2.1 GESTION DE CONCEPTOS DE PAGO
- 3.2.2 COPIA DE SEGURIDAD DEL LA BD
- 3.2.3 MIGRAR PAGOS A SERVIDOR LOCAL
- 3.2.4 GESTIÓN DE USUARIOS DEL SISTEMA

- 3.2.5 CAMBIO DE CLAVE

✓ 3.3 Modulo ZONA

- 3.3.1 GESTION DE ZONAS
- 3.3.2 GESTION DE JIRONES

✓ 3.4 Modulo USUARIOS

- 3.4.1 COMPROBANTE DE PAGO
- 3.4.2 ANULAR BOLETA
- 3.4.3 ELIMINAR ULTIMA BOLETA GENERADA
- 3.4.4 IMPRESIÓN DE BOLETA
- 3.4.5 GESTION DE USUARIOS
- 3.4.6 GESTION DE DEUDAS
- 3.4.7 CARGA DE DEUDA POR ZONA O SECTOR
- 3.4.8 ELIMINAR CUALQUIER BOLETA
- 3.4.9 VER PAGOS
- 3.4.10 GESTION DE CORTES
- 3.4.11 INGRESO DE PAGOS ANTERIORES

✓ 3.5 Modulo REPORTES

- 3.5.1 RESUMEN ACUMULADO POR DIA DEL MES
- 3.5.2 REGISTRO DE VENTA POR DIA
- 3.5.3 REGISTRO DE VENTA POR MES
- 3.5.4 REPORTE DE INGRESO DIARIO POR CONCEPTO
- 3.5.5 REPORTE DE INGRESO MENSUAL POR CONCEPTO
- 3.5.6 REPORTE DE INGRESO ANUAL POR CONCEPTO
- 3.5.7 REPORTE DE DEUDA POR SECTOR

✓ 3.6 Modulo REPORTES VARIOS

- 3.6.1 CANTIDAD DE USUARIOS POR SECTOR
- 3.6.2 REPORTE DE USUARIOS POR SECTOR
- 3.6.3 REPORTE DE PAGOS DE USUARIOS POR ZONA O SECTOR

3.6.4 LISTA DE USUARIOS AL DIA EN SUS PAGOS

Esquema conceptual de la base de datos

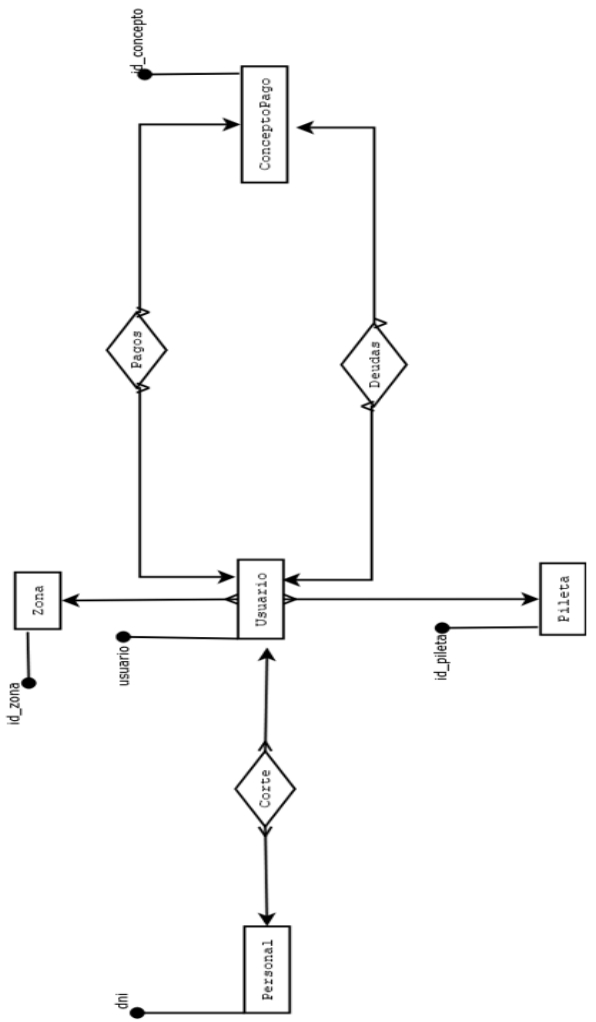


IMAGEN N°08-0001: Parte del esquema conceptual de la base de datos

Descripción esquema conceptual de la base de datos

El **esquema conceptual de la base de datos** representa de forma abstracta las entidades principales y las relaciones que organizan la información del sistema de gestión de agua potable. En el centro se encuentra la entidad **Usuarios**, que almacena los datos de los clientes del servicio. Cada usuario está asociado a una única **Zona**, mientras que una zona puede agrupar a varios usuarios, reflejando la distribución geográfica de los puntos de consumo. La entidad **Pileta** se conecta directamente con los usuarios, ya que un cliente puede tener una o varias piletas registradas como puntos de abastecimiento de agua.

La operación y control del servicio se representan a través de la relación entre **Usuarios** y **Personal** mediante el proceso de cortes y reconexiones. Dado que un trabajador puede realizar cortes a múltiples usuarios y un usuario puede ser cortado en diferentes fechas por distintos trabajadores, la relación se maneja como una conexión de tipo **muchos a muchos**, lo que permite registrar eventos históricos de forma precisa. Esto refleja la trazabilidad de las operaciones internas y la gestión de incidencias en la red de suministro.

En la parte financiera, la entidad **ConceptoPago** se vincula con los **Usuarios** para registrar tanto los pagos como las deudas. Un usuario puede pagar múltiples conceptos, y cada concepto puede ser abonado por distintos usuarios en varias boletas de pago, creando una relación **muchos a muchos** que también se replica en el registro de deudas. Esta estructura conceptual define las bases de la gestión administrativa, operativa y económica del sistema, asegurando que la información de clientes, zonas, personal y conceptos de pago esté conectada de manera coherente antes de pasar a los modelos lógico y físico.

En el diseño solo se muestra las entidades relaciones y los identificadores únicos de cada entidad. Los atributos no se consideraron para que sea más legible el esquema, los atributos están considerado en el esquema lógico como columnas de las tablas.

Esquema lógico de la base de datos

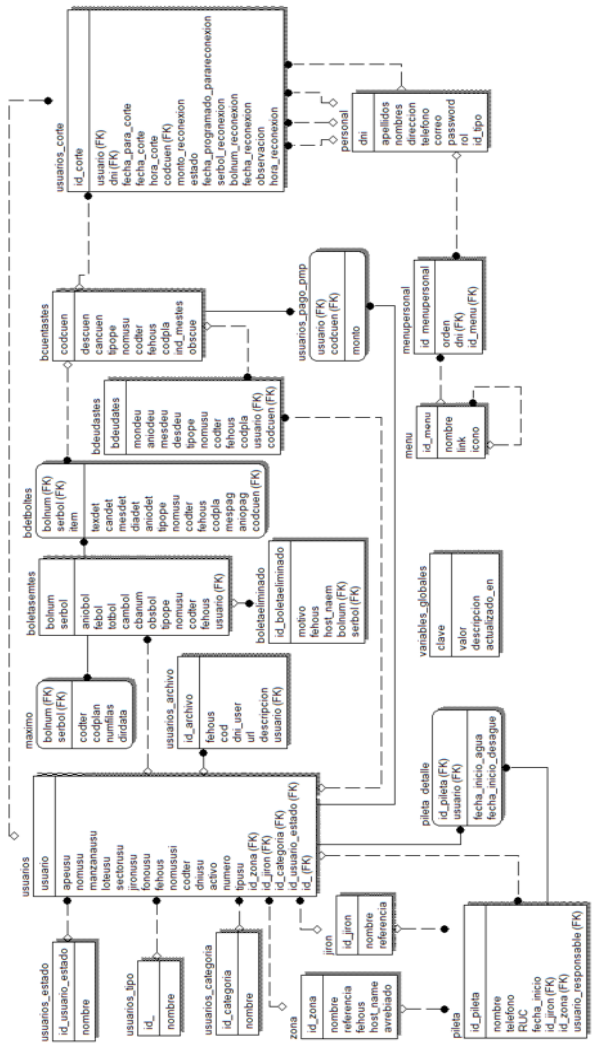


IMAGEN N°08-0002: Parte del esquema lógico de la base de datos

Descripción del esquema lógico de la base de datos

La base de datos está diseñada para gestionar de forma integral la información relacionada con los usuarios del servicio de agua potable y alcantarillado, así como sus pagos, boletas y estados de conexión. Su estructura central se basa en la tabla usuarios, que almacena datos personales y de ubicación, vinculándose con entidades auxiliares como usuarios_estado, usuarios_tipo y usuarios_categoria para clasificar a cada cliente según su condición, tipo y categoría de servicio. Además, se incluye la relación con la tabla jiron y zona para identificar la ubicación geográfica de cada usuario, y con pileta y pileta_detalle para controlar los puntos de acceso de agua y su historial de consumo o mantenimiento.

En el ámbito de la facturación y control financiero, la base de datos cuenta con las tablas boletasemtes, bdetboltes, bdeudastes y bcuentastes, que permiten registrar boletas, detallar conceptos facturados, controlar deudas y administrar cuentas de los usuarios. Complementariamente, la tabla usuarios_corte gestiona los procesos de suspensión y reconexión del servicio, integrando información de responsables, fechas y estados. El modelo se complementa con módulos de gestión interna como personal, menu y menupersonal para la administración de usuarios del sistema, y variables_globales para la configuración centralizada. En conjunto, el diseño refleja una estructura orientada a la trazabilidad de datos, la eficiencia en la gestión de cobros y el control operativo de la red de agua potable y alcantarillado.

Además, la base de datos integra un sistema de relaciones que garantiza la consistencia referencial entre sus entidades, permitiendo mantener la integridad de la información a lo largo de los procesos administrativos y operativos. La normalización aplicada reduce la redundancia de datos y facilita la generación de reportes confiables para la toma de decisiones.

Esquema físico de la base de datos “emapa”

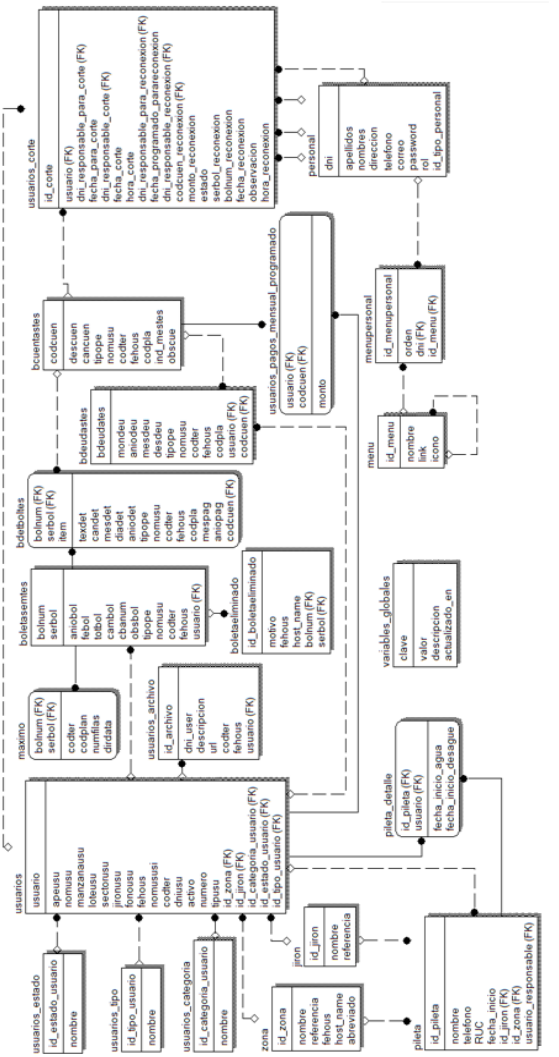


IMAGEN N°08-0003: Parte del esquema físico de la base de datos

Descripción del esquema físico de la base de datos

El **esquema físico de la base de datos en SQL Server** define la estructura exacta implementada en el motor relacional, con tablas, claves primarias, claves foráneas y relaciones diseñadas para garantizar integridad y rendimiento. El núcleo del modelo lo conforma la tabla usuarios, que almacena los datos principales de cada cliente y se enlaza mediante claves foráneas a tablas de clasificación (usuarios_estado, usuarios_tipo, usuarios_categoria) y de localización (jiron, zona). La tabla pileta y su detalle permiten gestionar los puntos de abastecimiento y su historial operativo, vinculados directamente con los usuarios.

En la parte operativa y financiera, las tablas boletasemtes, bdetboltes, bdeudastes y bcuentastes manejan la facturación, el detalle de consumos y la administración de cuentas de usuarios, mientras que usuarios_corte controla los procesos de corte y reconexión del servicio, integrando registros de fechas, responsables y estados. El esquema también incorpora módulos de administración interna (personal, menu, menupersonal) y configuración centralizada (variables_globales).

4. Script de Creación de las tablas estándar en SQL

Para la generación del script se usó el mismo Erwin que permite de manera automática a partir del modelo lógico/físico generar.

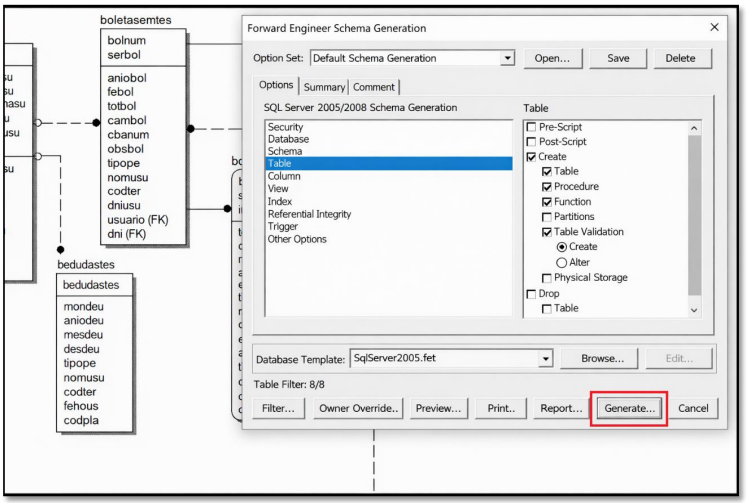


IMAGEN N°08-0004: Generación del Script de creación de las tablas en Erwin

Se desarrollará solo algunos módulos del sistema

2. Acceso al Sistema

El acceso al Sistema de Gestión de Usuarios y Control de Pagos Mensuales está restringido a usuarios debidamente registrados y autorizados por la administración de EMAPA San Luis S.A. Cada usuario tiene un nombre de usuario y contraseña únicos, y los permisos asignados determinan las funciones que puede visualizar y utilizar dentro del sistema.

2.1 Pantalla de Logueo

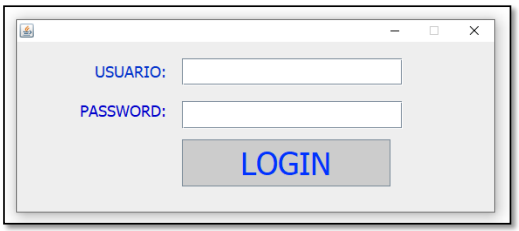


IMAGEN N°08-0005: Formulario de logeo para la autenticación y acceso al sistema

Al iniciar la aplicación, se mostrará una ventana de autenticación. Esta interfaz solicita dos datos:

- ✓ **Nombre de usuario**
- ✓ **Contraseña**

Una vez ingresadas las credenciales, el sistema validará la información con la base de datos. Si son correctas, se mostrará el menú principal con las funcionalidades correspondientes al rol del usuario.

En caso de error en las credenciales, el sistema mostrará un mensaje indicando que el usuario o la contraseña son incorrectos.

Se ingresa el usuario y contraseña y clic en LOGIN para acceder al sistema, así como se muestra en la IMAGEN N°08-0006:

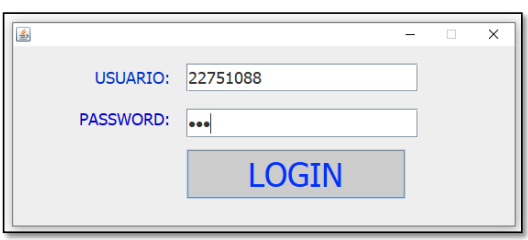


IMAGEN N°08-0006: Formulario de logeo para la autenticación y acceso al sistema

2.2 Roles de Usuario

El sistema de gestión de usuarios y control de pagos mensuales de la empresa "Pata Amarilla" permite el acceso controlado de acuerdo con el **rol asignado** a cada usuario. Esta asignación determina qué funciones y módulos del sistema están habilitados, presentando un menú dinámico según los permisos correspondientes.

A continuación, se describen los principales roles definidos en el sistema:

- ✓ **Caja**:
  Accede a las funciones de atención al cliente, registro de pagos, emisión de comprobantes y consulta de deuda por usuario. Su interfaz está optimizada para operaciones rápidas en ventanilla.
- ✓ **Comercialización**:
  Gestiona el registro y actualización de usuarios, servicios activos, lectura de medidores y facturación, gestión de cortes, gestión de deudas. También puede generar reportes comerciales y dar seguimiento a cuentas pendientes.
- ✓ **Administración del Sistema**:
  Encargado de la configuración técnica del sistema, gestión de usuarios, asignación de permisos, respaldo de base de datos y mantenimiento general del sistema.
- ✓ **Gerencia**:
  Cuenta con acceso a reportes ejecutivos y consolidados de los módulos operativos. Este rol tiene vista general del sistema, pero sin funciones de edición o modificación operativa.
- ✓ **Consulta**:
  Permite visualizar información general del sistema (como registros de pagos, usuarios y reportes) sin realizar modificaciones. Ideal para personal de supervisión o auditoría.
- ✓ **Administrador**:
  Tiene acceso total al sistema, incluyendo todas las funciones de los demás roles. Este rol se asigna únicamente a personal autorizado con competencias en gestión de sistemas.

El administrador del sistema es responsable de asignar y modificar los roles conforme a la estructura organizativa y funciones del personal de la empresa "Pata Amarilla".

3. Funcionalidades

El sistema de gestión de la empresa "Pata Amarilla" está diseñado para facilitar y automatizar los procesos relacionados con la administración de usuarios y el control de pagos mensuales por servicios de agua potable y alcantarillado. A continuación, se describen las principales funcionalidades disponibles según los módulos habilitados por rol de usuario:

3.1 Menú Principal

Al iniciar sesión, el sistema muestra un menú dinámico generado en función del rol del usuario tal como se puede apreciar en la IMAGEN N°08-0007 y resaltado en la IMAGEN N°08-0008. Este menú organiza las funcionalidades por categorías y permite un acceso rápido a las operaciones más frecuentes. A la izquierda de la pantalla se tiene las categorías:

- ✓ SISTEMA
- ✓ ZONA
- ✓ USUARIOS
- ✓ REPORTES
- ✓ REPORTES VARIOS

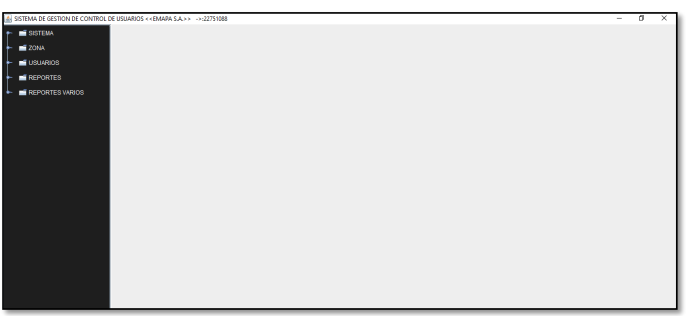


IMAGEN N°08-0007: Menú principal dinámico por rol, agrupado por categoría

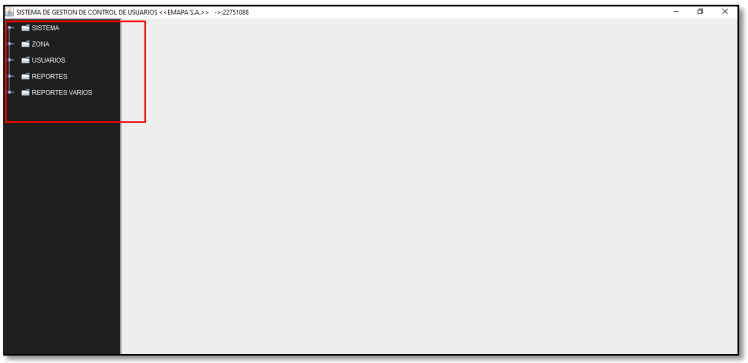


IMAGEN N°08-0008: Menú principal dinámico por rol, agrupado por categoría

Al hacer clic en cada categoría se tiene el menú lateral extendido con todas las opciones, tal como se puede apreciar en la IMAGEN N°08-0009

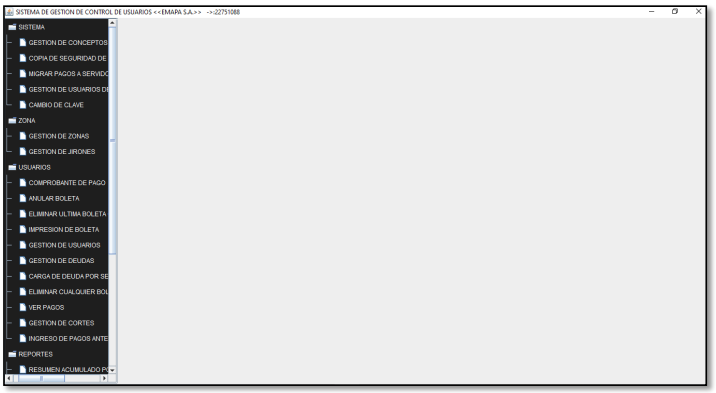


IMAGEN N°08-0009: Menú principal dinámico por rol, extendido

3.2 Modulo SISTEMA

Este módulo permite configurar y mantener el entorno general del sistema.

En esta categoría se tiene las opciones:

- ✓ GESTION DE CONCEPTOS DE PAGO
- ✓ COPIA DE SEGURIDAD DE LA BASE DE DATOS
- ✓ MIGRAR PAGOS A SERVIDOR LOCAL
- ✓ GESTION DE USUARIOS DEL SISTEMA
- ✓ CAMBIO DE CLAVE

Tal como se muestra en la IMAGEN N°08-0010.

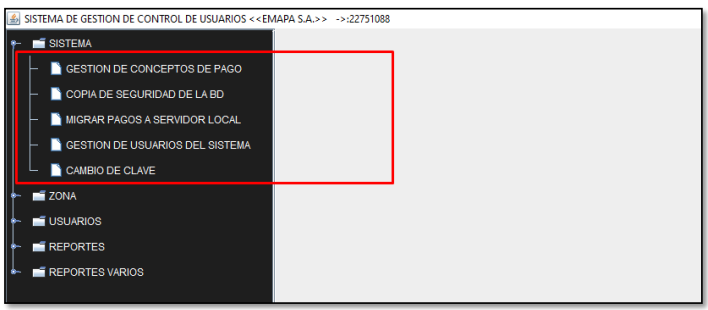


IMAGEN N°08-0010: Opciones del Módulo SISTEMA

3.2.1 GESTION DE CONCEPTOS DE PAGO

Esta opción permite ingresar nuevos conceptos de pago, modificar los conceptos existentes y eliminar aquellos que ya no se utilizan.

Al hacer clic en el ítem **"GESTIÓN DE CONCEPTOS DE PAGO"**, se mostrará la pantalla correspondiente (ver IMAGEN N°08-0011).

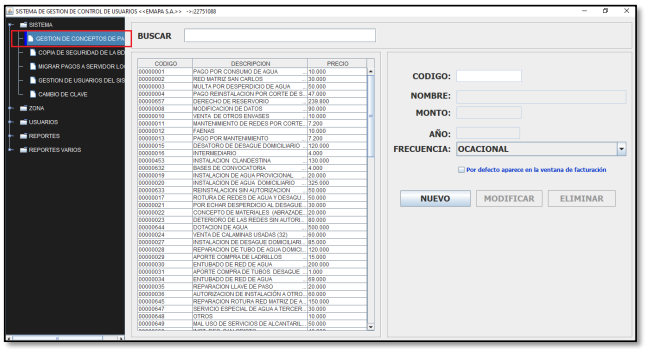


IMAGEN N°08-0011: Pantalla para la gestión de conceptos de pago

Registrar un nuevo concepto:

1. Hacer clic en el botón **"NUEVO"**.
2. Se habilitarán los campos del formulario para su llenado, así como se muestra en la IMAGEN N°08-0012.
3. El campo **CÓDIGO** se autogenera automáticamente.
4. En el campo **FRECUENCIA**, seleccionar si el concepto corresponde a:
   - **Pagos mensuales** (por ejemplo: consumo de agua), o
   - **Pagos esporádicos** (por ejemplo: instalación de servicio).
5. Marcar la opción para que el concepto **aparezca por defecto** en el módulo de pagos, si se desea.

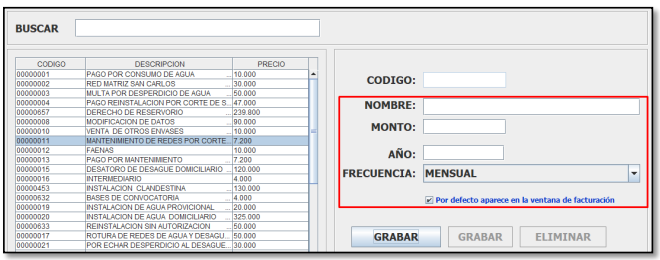


IMAGEN N°08-0012: Pantalla para registrar nuevo concepto de pago

Buscar y visualizar conceptos:

- ✓ Para buscar un concepto, utilizar el campo **BUSCAR**.
- ✓ Escriba la palabra clave y presione la tecla **espaciadora**
- ✓ El sistema filtrará automáticamente los registros que coincidan con la palabra ingresada y los mostrará en la tabla, **tal como se muestra en la** IMAGEN N°08-0013.

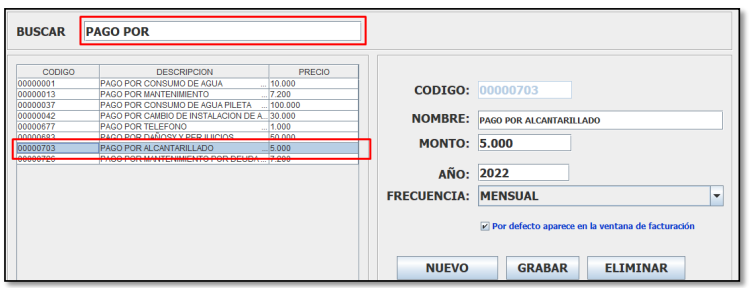


IMAGEN N°08-0013: Pantalla de buscador de conceptos

Modificar un concepto:

1. Seleccionar el concepto desde la tabla de resultados.
2. Hacer clic en el botón **"MODIFICAR",** así como se muestra en la IMAGEN N°08-0014
3. Realizar los cambios necesarios.
4. Hacer clic en **"GRABAR"** para guardar las modificaciones.

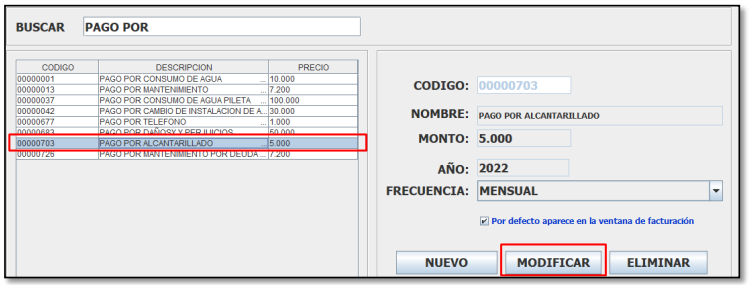


IMAGEN N°08-0014: Pantalla de buscador de conceptos

3.2.2 COPIA DE SEGURIDAD DEL LA BD

Esta opción permite generar manualmente una copia de respaldo de la base de datos del sistema, con el fin de proteger la información ante posibles fallos o pérdidas de datos.

Al hacer clic en el ítem “COPIA DE SEGURIDAD DE LA BD”, se muestra una interfaz sencilla con un único botón, como se puede visualizar en la Imagen N° 011.

COPIA DE SEGURIDAD DE LA BD: al presionar este botón, el sistema genera automáticamente una copia de la base de datos.

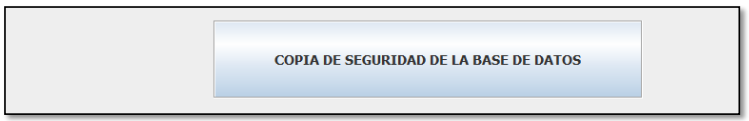


Imagen N° 011: Pantalla para realizar copia de seguridad de la base de datos en el servidor

Detalles técnicos:

- ✓ El archivo de respaldo se guarda por defecto en la carpeta: \\sisemapa\bakup_bd del servidor.
- ✓ El nombre del archivo generado incluye la fecha y hora de la copia, lo que permite identificar fácilmente el respaldo más reciente.

Recomendaciones:

- ✓ Realizar esta acción de forma periódica, especialmente antes de realizar actualizaciones o cambios importantes en el sistema.
- ✓ Copiar manualmente los archivos de respaldo a dispositivos externos o almacenamiento en la nube para mayor seguridad.

3.2.3 MIGRAR PAGOS A SERVIDOR LOCAL

Esta opción permite migrar todos los pagos realizados por los usuarios desde la base de datos del servidor hacia la base de datos local instalada en la computadora de caja, el formulario se muestra en la Imagen N° 012.

> *Esta migración es necesaria debido a que en la computadora de caja está instalado un programa especial encargado de* ***enviar las boletas electrónicas a SUNAT****, el cual está configurado para* ***tomar los datos directamente desde la base de datos local de caja*** *y no desde el servidor central.*

Pasos para realizar la migración:

1. Seleccionar la **fecha de los pagos** que se desea migrar.
2. Hacer clic en el botón **"MIGRAR DATOS AL SERVIDOR LOCAL"**.
3. El sistema realizará la búsqueda de los pagos correspondientes a esa fecha y los mostrará en una **tabla en pantalla**.
4. Una vez visualizados, los datos serán transferidos automáticamente a la base de datos local (computadora de caja).

Consideraciones importantes:

- ✓ Solo se migrarán los pagos **registrados en la fecha seleccionada**.
- ✓ Los pagos ya migrados no se duplicarán en futuras migraciones.
- ✓ Es recomendable realizar esta migración **antes del envío diario** de boletas electrónicas a SUNAT.

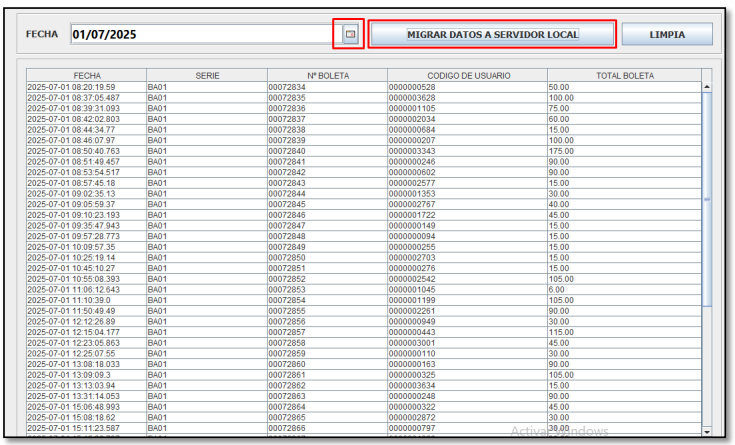


Imagen N° 012: Pantalla que se muestra el formulario para migrar las boletas de pago a la computadora local

3.2.4 GESTIÓN DE USUARIOS DEL SISTEMA

Este formulario permite **asignar o modificar los permisos de acceso** al sistema para cada usuario registrado. Desde esta pantalla se puede vincular a cada persona con las funcionalidades que podrá utilizar, de acuerdo a su rol o área (como Caja, Comercialización, Gerencia, etc.). Esta pantalla se muestra en la Imagen N° 013.

Interfaz del formulario:

- ➢ **Panel izquierdo** – *PERSONAL*:

  Se muestra la lista de usuarios registrados en el sistema con sus respectivos **DNI, apellidos y nombres**.

  Los botones disponibles son:

- ✓ NUEVO: Permite registrar un nuevo usuario.
- ✓ MODIFICAR: Permite editar los datos o permisos del usuario seleccionado.
- ✓ CANCELAR: Cierra o limpia la selección actual.

Para poder buscar a un personal escribir la palabra clave en PERSONAL y presionar la tecla **ESPACIADORA** y filtrara al personal.

- **Panel derecho** – *MENU*:

  Se listan todas las opciones del sistema, cada una con un **ID, descripción** y una columna para **asignar permisos** (checkbox).

  Al marcar un ítem, se habilita el acceso del usuario seleccionado a dicha función del sistema.

  Para poder buscar un menú especifico escribir la palabra clave en MENU y presionar la tecla **ESPACIADORA** y filtrara al menú.

- **Botón GRABAR**:

  Al hacer clic, se guardan todos los cambios realizados a los permisos del usuario seleccionado.

Ejemplo de uso:

1. Seleccionar un usuario en la tabla de personal.
2. Marcar o desmarcar los permisos del menú según lo requerido.
3. Hacer clic en el botón **GRABAR** para guardar los cambios.

Esta funcionalidad garantiza que cada usuario solo acceda a las partes del sistema que le corresponden, asegurando el control y la confidencialidad de la información

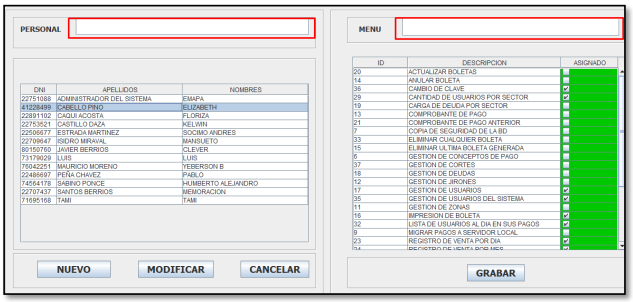


Imagen N° 013: Pantalla que se muestra el formulario de gestión de usuarios del sistema

3.2.5 CAMBIO DE CLAVE

Esta opción permite a los usuarios del sistema actualizar su contraseña de acceso de manera segura.

Objetivo:

Brindar al usuario la posibilidad de modificar su clave personal para garantizar la confidencialidad y el acceso seguro al sistema.

Interfaz:

La interfaz se muestra en la Imagen N° 014

- ✓ **DNI**: Campo visible y bloqueado, muestra el número de documento del usuario que inicio sesión el cual está realizando el cambio.
- ✓ **Nueva contraseña**: Campo para ingresar la nueva clave.
- ✓ **Repetir nueva contraseña**: Campo de confirmación para evitar errores en la escritura de la nueva clave.

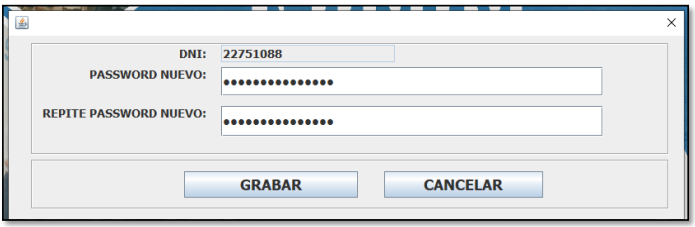


Imagen N° 014: Interfaz que permite modificar el password de acceso al sistema

Funcionamiento:

1. El usuario debe ingresar su nueva contraseña dos veces.
2. El sistema verifica que ambas coincidan.
3. Si coinciden, se actualiza la contraseña del usuario.
4. Si no coinciden, se muestra un mensaje de advertencia indicando que las claves no coinciden.
5. Una vez registrada la nueva clave, el formulario se cierra y se actualiza la información correspondiente en el sistema.

Botones:

- **Grabar**: Valida la información ingresada y guarda la nueva contraseña.
- **Cancelar**: Cierra el formulario sin realizar cambios.

Seguridad:

La contraseña es almacenada en la base de datos utilizando mecanismos seguros que evitan su visualización directa o suplantación. El formulario también oculta los caracteres mientras el usuario escribe la nueva clave.

3.3 Modulo ZONA

En esta categoría se tiene las opciones:

- ✓ GESTION DE ZONAS
- ✓ GESTION DE JIRONES

Tal como se muestra en la Imagen N° 015

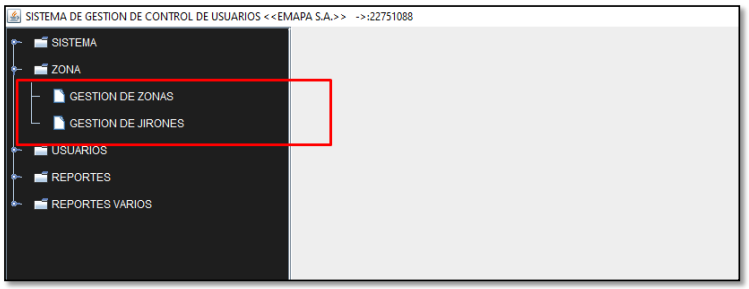


Imagen N° 015: Menú ZONA

3.3.1 GESTION DE ZONAS

En esta opción se permite **agregar, modificar y eliminar zonas**, sectores o asentamientos humanos donde residen los usuarios del servicio de agua y alcantarillado. Esta funcionalidad es fundamental para mantener actualizada la información geográfica que facilita la administración y distribución del servicio.
La pantalla correspondiente a esta opción se muestra en la **Imagen N.º 016**, donde se visualiza una tabla con los siguientes campos:

- **ID**: Código numérico de la zona.
- **Descripción**: Nombre del sector o asentamiento.
- **Referencia**: Ubicación o punto de referencia para identificar la zona.
- **Abreviado**: Código corto utilizado para identificar rápidamente la zona.

Además, se cuenta con los botones funcionales:

- **NUEVO**: Permite registrar una nueva zona.
- **MODIFICAR**: Permite editar los datos de una zona seleccionada.
- **ELIMINAR**: Permite eliminar una zona registrada (solo habilitado si hay una selección previa).

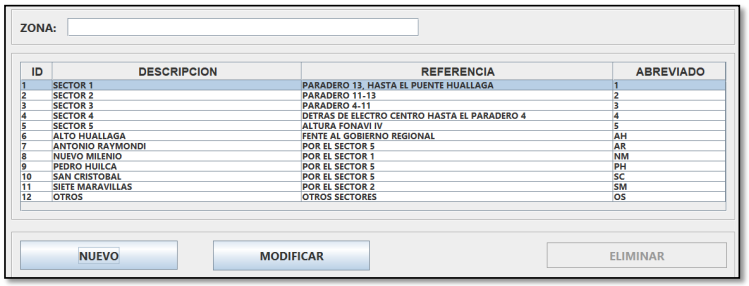


Imagen N° 016: Interfaz para gestión de zonas

Para agregar nuevas zonas clic en el botón “NUEVO” y se tendrá el formulario que se muestra en la Imagen N° 017. Rellenar los campos y clic en “GRABAR”

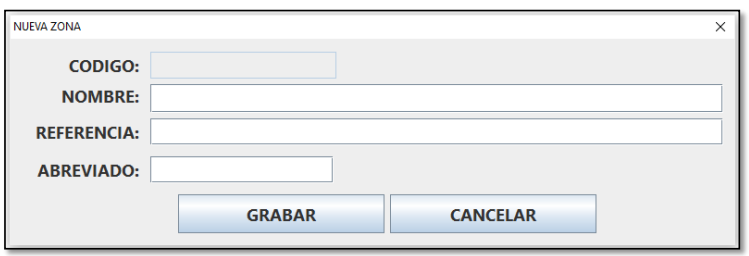


Imagen N° 017: Interfaz para ingresar datos de nueva zona

Para modificar datos de zonas, seleccionar o buscar ingresando en la caja de texto **ZONA** la palabra clave y presionar la tecla **espaciadora que filtrará las zonas y se cargará en la tabla, seleccionar registro y clic en “MODIFICAR”** y se tendrá el formulario que se muestra en la Imagen N° 018, modificar los campos requeridos y clic en “GRABAR”

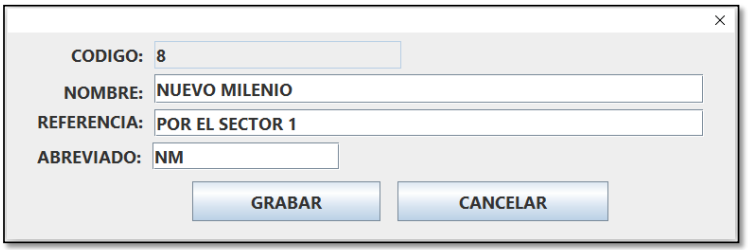


Imagen N° 018: Interfaz para modificar datos de zona

3.3.2 GESTION DE JIRONES

En esta opción, el usuario puede **registrar nuevos jirones**, así como **modificar** o **eliminar** aquellos que ya se encuentran registrados en el sistema tal como se muestra en la Imagen N° 019. Los jirones representan las calles o vías donde residen los usuarios del servicio de agua y alcantarillado.

La pantalla presenta una lista de jirones ya registrados, mostrando sus principales características como el identificador, nombre o descripción del jirón, y una referencia adicional. Asimismo, se incluye un campo de texto para ingresar o editar el nombre del jirón.

Las principales funcionalidades disponibles son:

- **Nuevo**: Permite registrar un nuevo jirón en el sistema.
- **Modificar**: Permite editar la información de un jirón previamente registrado.
- **Eliminar** (si está habilitado): Permite quitar un jirón del registro, siempre que no esté vinculado a usuarios activos del servicio.

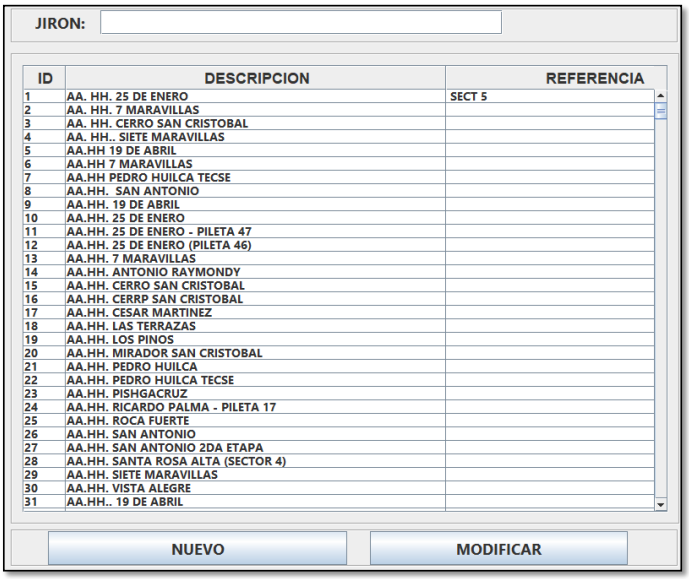


Imagen N° 019: Interfaz de gestión de jirones

Para agregar nuevos jirones, clic en el botón “NUEVO” y se tendrá el formulario que se muestra en la Imagen N° 020. Rellenar los campos y clic en “GRABAR”

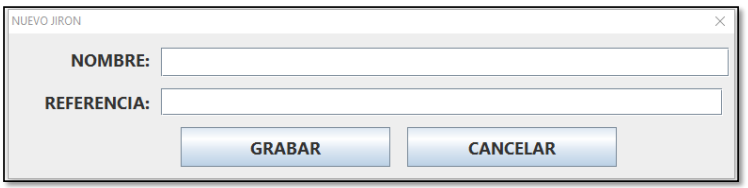


Imagen N° 020: Interfaz para agregar un nuevo jiron

Para modificar jirones, clic en el botón “MODIFICAR” y se tendrá el formulario que se muestra en la Imagen N° 021. Rellenar los campos y clic en “GRABAR”

Imagen N° 021: Interfaz para agregar un nuevo jirón

3.4 Modulo USUARIOS

El módulo de **Usuarios** agrupa las funciones principales relacionadas con la gestión de datos de los usuarios del servicio de agua y alcantarillado. Permite generar comprobantes de pago, administrar boletas, gestionar deudas, registrar pagos, y controlar procesos de corte del servicio, entre otros.

En esta categoría se tiene las opciones:

- ✓ **COMPROBANTE DE PAGO**
- ✓ **ANULAR BOLETA**
- ✓ **ELIMINAR ULTIMA BOLETA GENERADA**
- ✓ **IMPRESIÓN DE BOLETA**
- ✓ **GESTION DE USUARIOS**
- ✓ **GESTION DE DEUDAS**
- ✓ **CARGA DE DEUDA POR SECTOR**
- ✓ **ELIMINAR CUALQUIER BOLETA**
- ✓ **VER PAGOS**
- ✓ **GESTION DE CORTES**
- ✓ **INGRESO DE PAGOS ANTERIORES**

Tal como se muestra en la Imagen N° 022

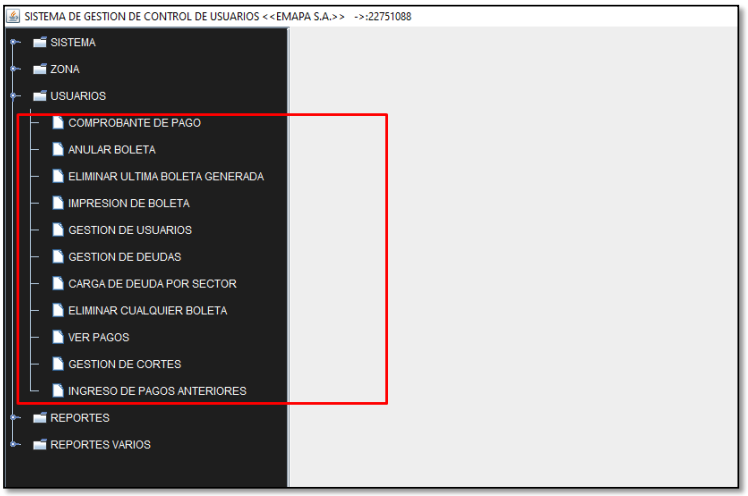


Imagen N° 022: Opciones del módulo de USUARIOS

3.4.1 COMPROBANTE DE PAGO

Esta opción le permite generar las boletas de pago para los usuarios por diversos conceptos, como el pago por consumo de agua y el pago por alcantarillado.
Interfaz de "COMPROBANTE DE PAGO" es la que se muestra en la Imagen N° 023**:**

Pasos para Generar una Boleta de Pago:

1. **Búsqueda y Selección de Usuario:**

- ✓ En la parte superior de la pantalla, ubique la caja de texto **"USUARIO"**.
- ✓ Ingrese el nombre o parte del nombre del usuario que desea buscar (ej. "LOPEZ T" como se muestra en la imagen).
- ✓ Presione la tecla **[ESPACIO]** para filtrar a los usuarios que coincidan con la palabra clave ingresada. Los resultados se mostrarán en la tabla debajo de la caja de texto.
- ✓ **Seleccione el usuario** haciendo clic en la fila correspondiente en la tabla. Por ejemplo, en la Imagen N° 023, "LOPEZ TUCTO VICENTE HUBERTO" está seleccionado.

2. **Visualización de Información del Usuario:**

✓ Una vez seleccionado el usuario, el sistema cargará automáticamente su información en las pestañas inferiores: **"DEUDAS"**, **"PAGOS"** y **"OTROS CONCEPTOS"**.

✓ La sección **"DEUDA TOTAL S/."** en la parte superior derecha mostrará el monto total adeudado por el usuario. En el ejemplo, es "45.0".

3. **Selección de Deudas para Pagar:**

✓ Asegúrese de estar en la pestaña **"DEUDAS"** (resaltada en azul en la imagen).

✓ La tabla de "DEUDAS" mostrará los conceptos pendientes de pago (ej., "PAGO POR CONSUMO DE AGUA", "PAGO POR ALCANTARILLADO") con su respectivo año, mes y monto, tal como se muestra en la Imagen N° 024.

✓ Para agregar una deuda al comprobante de pago, haga **doble clic** en la fila correspondiente en la tabla de "DEUDAS".

✓ Cada deuda seleccionada se agregará a la tabla de ítems del comprobante de pago, ubicada en la parte inferior derecha de la pantalla (actualmente vacía en la imagen, pero es donde aparecerían los ítems seleccionados para el pago).

4. **Verificación y Confirmación:**

✓ Revise cuidadosamente los ítems que se han añadido al comprobante de pago.

✓ Verifique el "SUB TOTAL S/.", "DESCUENTO S/.", "IGV %", e "IMPORTE TOTAL S/." en la sección inferior derecha.

5. **Generación de la Boleta de Pago:**

✓ Cuando toda la información esté correcta, haga clic en el botón **"GUARDAR"** (ubicado en la parte inferior central de la pantalla), tal como se muestra en la Imagen N° 025

✓ El sistema generará la boleta de pago correspondiente, así como se muestra en la Imagen N° 026

Nota: La opción "IMPRIMIR DEUDA" está disponible en la pestaña "DEUDAS" si necesita generar un reporte de las deudas pendientes antes de procesar el pago.

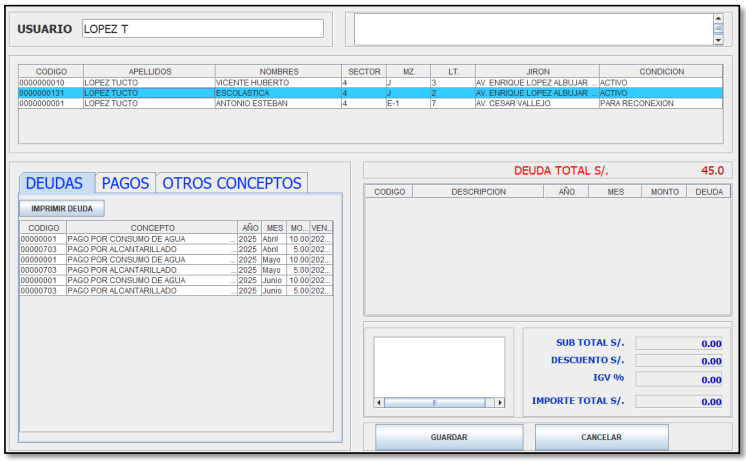


Imagen N° 023: Interfaz para generar boletas de pago de los usuarios

Así mismo en la pestaña “PAGOS” se puede visualizar lo pagos realizados por el usuario, tal como se muestra en la Imagen N° 024:

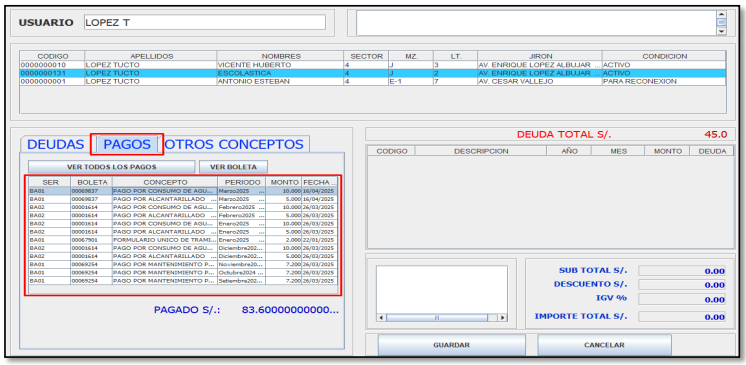


Imagen N° 024: Interfaz para generar boletas de pago de los usuarios donde se visualiza los pagos realizados por el usuario

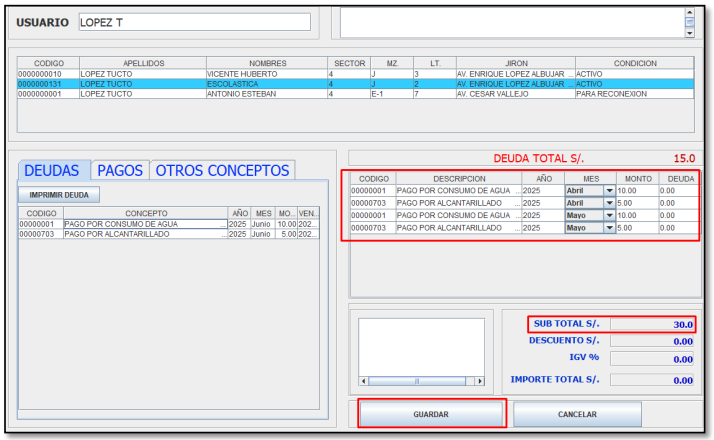
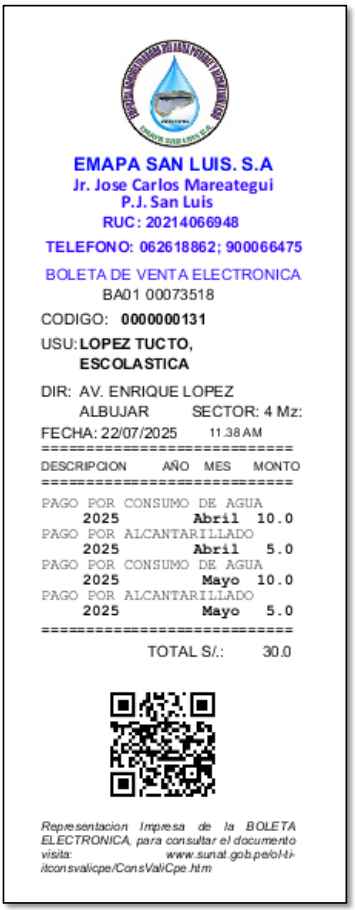


Imagen N° 025: Interfaz para generar boletas de pago



EMAPA SAN LUIS. S.A
Jr. Jose Carlos Mareategui
P.J. San Luis
RUC: 20214066948
TELEFONO: 062618862; 900066475
BOLETA DE VENTA ELECTRONICA
BA01 00073518
CODIGO: 0000000131
USU: LOPEZ TUCTO,
ESCOLASTICA
DIR: AV. ENRIQUE LOPEZ
ALBUJAR SECTOR: 4 Mz:
FECHA: 22/07/2025 11.38 AM
==========
DESCRIPCION AÑO MES MONTO
==========
PAGO POR CONSUMO DE AGUA
2025 Abril 10.0
PAGO POR ALCANTARILLADO
2025 Abril 5.0
PAGO POR CONSUMO DE AGUA
2025 Mayo 10.0
PAGO POR ALCANTARILLADO
2025 Mayo 5.0
==========
TOTAL S/.: 30.0

Representacion Impresa de la BOLETA ELECTRONICA, para consultar el documento visita: www.sunat.gob.pe/ol-ti-itconsvalicpe/ConsValiCpe.htm

Imagen N° 026: Boleta generada por los pagos realizados

3.4.2 ANULAR BOLETA

Esta opción le permite anular una boleta de pago previamente generada. Es crucial que solo se anulen boletas cuando sea estrictamente necesario y con un motivo válido.

Interfaz de "ANULAR BOLETA":

Como se muestra en la **Imagen N°027: Interfaz de Anulación de Boletas**, esta pantalla le presenta los campos y controles necesarios para realizar el proceso.

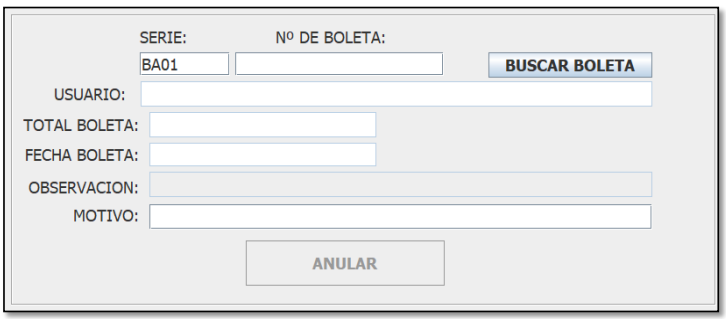


Imagen N° 027: Interfaz para anular boleta

Pasos para Anular una Boleta:

1. **Ingreso de datos de la Boleta:**
   - ✓ En la sección superior de la interfaz (referida en la Imagen N°027), localice los campos etiquetados como **"SERIE"** y **"N° DE BOLETA"**.
   - ✓ Ingrese el número de serie y el número correlativo exactos de la boleta que desea anular.
2. **Búsqueda de la Boleta:**
   - ✓ Haga clic en el botón **"BUSCAR BOLETA"** (como se observa en la Imagen N°027).
   - ✓ Si la boleta existe y es válida para anulación, el sistema cargará y mostrará automáticamente la siguiente información en la pantalla:
     - **Usuario:** El nombre completo del usuario al que se emitió la boleta.
     - **Total de la Boleta:** El importe monetario total registrado en la boleta.
     - **Fecha de la Boleta:** La fecha exacta en que se generó la boleta.

3. **Ingreso del Motivo de Anulación:**
   - ✓ Una vez que los datos de la boleta encontrada se visualicen, diríjase al área de texto designada para el **"MOTIVO"** (mostrado en la Imagen N°027).
   - ✓ Escriba de forma clara y concisa la razón por la cual se procede con la anulación de la boleta. Este campo es de carácter obligatorio.
4. **Confirmación de Anulación:**
   - ✓ Después de haber verificado que la información mostrada es la correcta para la boleta a anular y de haber ingresado el motivo, haga clic en el botón **"ANULAR"** (ubicado, como se ve en la Imagen N°028, que está en la parte inferior de la interfaz.
   - ✓ El sistema procesará la solicitud de anulación mostrando antes un mensaje de confirmación, así como se muestra en la Imagen N°029. Recibirá un mensaje de confirmación en pantalla si la operación fue exitosa, o un mensaje de error en caso contrario, tal como se muestra en la Imagen N°030.

Consideraciones Importantes:

- ✓ Una boleta anulada no es válida para ningún fin contable o de cobro posterior.
- ✓ Todas las anulaciones quedan registradas en el sistema, incluyendo la fecha, hora y el usuario que realizó la operación, para fines de auditoría y control interno.

Imagen N° 028: Interfaz para anular boleta

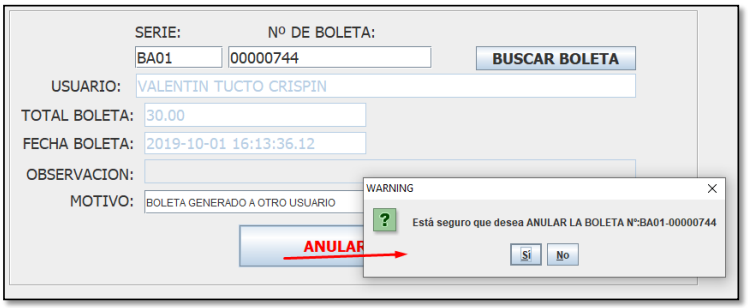


Imagen N° 029: Mensaje solicitando confirmación para la anulación de la boleta

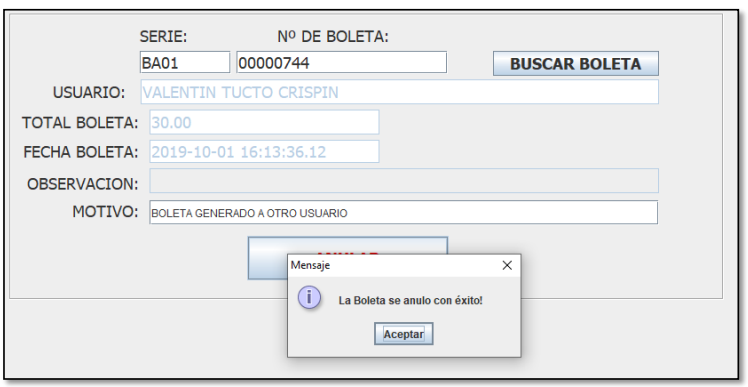


Imagen N° 030: Mensaje de confirmación de anulación de la boleta

3.4.3 ELIMINAR ULTIMA BOLETA GENERADA

Esta opción le permite eliminar la boleta más reciente que ha sido generada por el sistema. Una característica importante de esta función es que, al eliminar la última boleta, se **libera su número** para que pueda ser reutilizado en una nueva generación de boleta. Esto es útil para corregir errores inmediatos sin alterar la secuencia numérica.

Interfaz de "ELIMINAR ÚLTIMA BOLETA GENERADA":

La interfaz principal para esta opción se presenta como se muestra en la **Imagen N° 031: Interfaz de Carga de Última Boleta**.

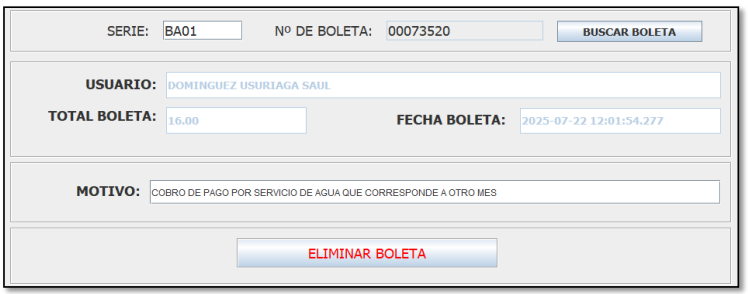


Imagen N° 031: Interfaz para eliminar la última boleta generado

Pasos para Eliminar la Última Boleta Generada:

1. **Carga de la Última Boleta:**

✓ Al acceder a esta opción, el campo **"Serie"** se cargará por defecto con el valor "BA01" (o la serie predeterminada configurada para la última boleta).

✓ El sistema intentará cargar automáticamente los detalles de la última boleta generada con esa serie en la interfaz (como se ve en la Imagen N° 031).

✓ Si la información de la última boleta no se carga automáticamente al abrir el formulario, haga clic en el botón **"BUSCAR BOLETA"**.

2. **Visualización de la Boleta a Eliminar:**

✓ Una vez que la boleta es encontrada, la interfaz mostrará los datos de la última boleta generada, incluyendo:

➢ **Serie:** La serie de la boleta (ej. "BA01").

➢ **Número de Boleta:** El número correlativo de la última boleta.

➢ **Datos del Usuario:** Información del usuario asociado a la boleta.

➢ **Total Boleta:** El monto total de la boleta.

➢ **Fecha Boleta:** La fecha en que se generó la boleta.

- ✓ Asegúrese de que la información mostrada en la Imagen N° 031 corresponde a la boleta que desea eliminar.

3. **Ingreso del Motivo de Eliminación:**

- ✓ En el cuadro de texto etiquetado como **"MOTIVO"** (visible en la Imagen N° 031), ingrese la razón por la cual se está eliminando esta última boleta. Este detalle es importante para el registro y auditoría.

4. **Confirmación de Eliminación:**

- ✓ Revise cuidadosamente los datos de la boleta mostrados para asegurarse de que es la boleta correcta que desea eliminar.
- ✓ Haga clic en el botón **"ELIMINAR BOLETA"** (como se observa en la Imagen N° 031).
- ✓ El sistema le presentará un mensaje de confirmación pidiéndole que confirme la acción, como se ilustra en la **Imagen N° 032: Mensaje de Confirmación de Eliminación de Boleta**.

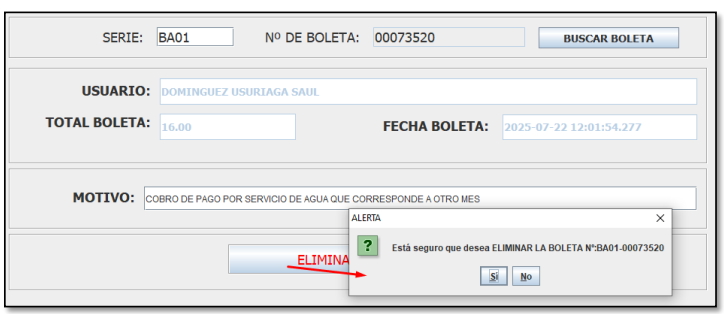


Imagen N° 032: Mensaje solicitando que confirme la eliminación de la boleta

- ✓ Para proceder con la eliminación, haga clic en **"Sí"** en el cuadro de diálogo.

5. **Proceso y Confirmación de Eliminación Exitosa:**

- ✓ Una vez confirmada la acción, la boleta será eliminada de la base de datos.
- ✓ Posteriormente, el sistema mostrará un mensaje final de confirmación en pantalla, indicando que la boleta ha sido eliminada exitosamente, tal como se ve en la **Imagen N° 033: Mensaje de Confirmación de Eliminación Exitosa**.

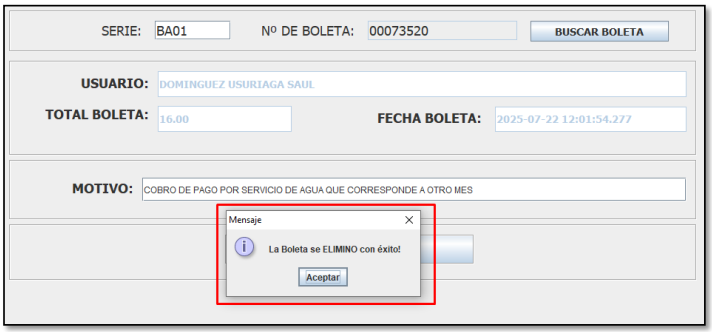


Imagen N° 033: Mensaje confirmando la eliminación de la boleta

Consideraciones Importantes:

- ✓ Esta función debe usarse con precaución, ya que elimina permanentemente la boleta del sistema.
- ✓ El número de boleta eliminado queda disponible para ser utilizado nuevamente, manteniendo así la secuencia de numeración.
- ✓ Asegúrese de tener un motivo válido para la eliminación, ya que esta acción queda registrada.

3.4.4 IMPRESIÓN DE BOLETA

Esta opción le permite reimprimir cualquier boleta de pago que haya sido generada previamente en el sistema. Es útil para obtener copias físicas de las boletas cuando sea necesario.

Interfaz de "IMPRESIÓN DE BOLETA":

Pasos para Imprimir una Boleta:

1. **Ingreso de Datos de la Boleta:**

- ✓ Localice el campo para la **"Serie"** de la boleta. Por defecto, este campo estará precargado con el valor **"BA01"**. Si la boleta que desea imprimir pertenece a otra serie, deberá modificar este valor.
- ✓ Ingrese el **Número de Boleta** exacto de la boleta que desea imprimir en el campo correspondiente, tal como se ilustra en la **Imagen N° 034: Ingreso de Datos para Impresión de Boleta**.

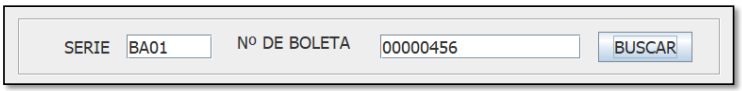


Imagen N° 034: Interfaz para visualizar e imprimir boleta

2. **Búsqueda y Vista Preliminar:**

- Una vez ingresados la serie y el número de boleta, haga clic en el botón **"BUSCAR"** (visible en la Imagen N° 034).
- El sistema procesará la búsqueda y, si la boleta existe, le presentará una **vista preliminar** de la boleta en pantalla, como se muestra en la **Imagen N° 035: Vista Preliminar de Boleta**. Esta vista le permite revisar el contenido y el formato de la boleta antes de enviarla a la impresora.


EMAPA SAN LUIS. S.A
Jr. Jose Carlos Mareategui
P.J. San Luis
RUC: 20214066948
TELEFONO: 062618862; 900066475
BOLETA DE VENTA ELECTRONICA
BA01 00000456
CODIGO: null
USU: SANTOS ROJAS, TELESFORO
DIR: JR. PASTAZA Sect:3 Mz:S Lt.12
FECHA: 23/09/2019 7.22 PM
DESCRIPCION AÑO MES MONTO
PAGO POR CONSUMO DE AGUA
2019 Ago 4.0
PAGO POR MANTENIMIENTO
2019 Ago 4.0
PAGO REINSTALACION POR
2019 Jul 10.0
TOTAL S/.: 18.0
Representacion Impresa de la BOLETA ELECTRONICA, para consultar el documento visita: www.sunat.gob.pe/ol-ti-itconsvalicpe/ConsValiCpe.htm

Imagen N° 034: Vista previa de la boleta a imprimir

3. **Impresión de la Boleta:**

✓ Desde la vista preliminar (Imagen N° 035), encontrará la opción para enviar la boleta a su impresora configurada. Siga las instrucciones en pantalla o los botones de la barra de herramientas de la vista preliminar para completar el proceso de impresión.

Consideraciones Importantes:

- Asegúrese de que su impresora esté correctamente conectada y configurada antes de intentar imprimir.
- La vista preliminar es una herramienta útil para verificar que la boleta es la correcta y que se imprimirá de la manera deseada.

3.4.5 GESTION DE USUARIOS

Esta opción es el centro para la administración de los usuarios del servicio de agua. Permite realizar las siguientes operaciones principales: **agregar nuevos usuarios, modificar datos existentes y eliminar registros de usuarios.**

La interfaz general de "GESTIÓN DE USUARIOS" se presenta como se muestra en la **Imagen N° 035: Interfaz de Gestión de Usuarios**.

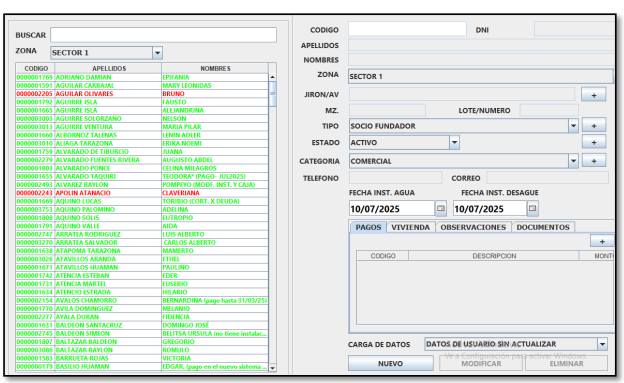


Imagen N° 035: Interfaz de Gestión de Usuarios

Agregar un Nuevo Usuario

Para incorporar un nuevo usuario al sistema, siga los siguientes pasos:

1. **Iniciar Nuevo Registro:**

✓ Haga clic en el botón **"NUEVO"** (visible en la Imagen N° 035). Al hacerlo, los campos de entrada de datos en la interfaz se habilitarán, permitiéndole ingresar la información del nuevo usuario.

2. **Relleno de Campos Principales:**

✓ Rellene todos los campos solicitados con la información personal y de contacto del usuario.

3. **Selección de Zona:**

✓ En el campo de selección de zona, elija la zona a la que pertenece el nuevo usuario, como se ilustra en la **Imagen N° 036: Selección de Zona**.

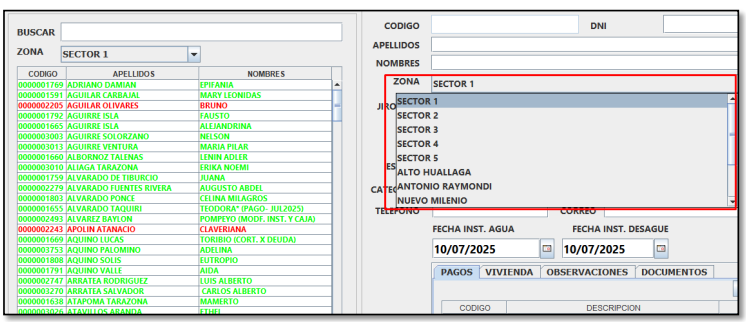


Imagen N° 036: Lista de zonas que se tiene disponible

4. Búsqueda y Selección de Jirón:

✓ Utilice la función de búsqueda para encontrar y seleccionar el jirón correspondiente a la dirección del usuario. Esta acción se muestra en la **Imagen N° 037: Búsqueda y Selección de Jirón**.

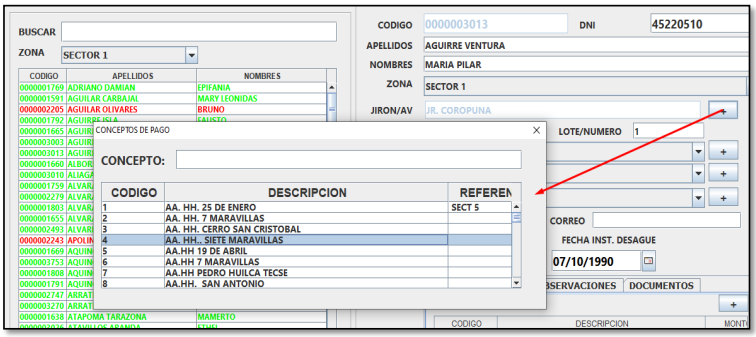


Imagen N° 037: Interfaz para seleccionar la dirección del usuario

5. Selección de Tipo de Usuario:

- ✓ Seleccione el tipo de usuario de la lista desplegable, según lo que se observa en la **Imagen N° 038: Selección de Tipo de Usuario**.

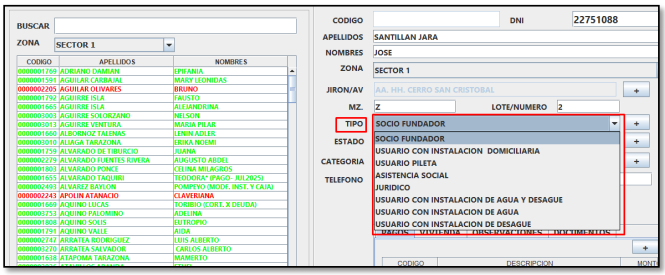


Imagen N° 038: Lista para seleccionar el tipo de usuario

- ✓ **Para Agregar un Nuevo Tipo de Usuario (Opcional):** Si el tipo de usuario requerido no está en la lista, haga clic en el botón **"+"** ubicado al lado derecho de la lista desplegable (referencia **Imagen N° 039: Botón para Agregar Tipo de Usuario**).

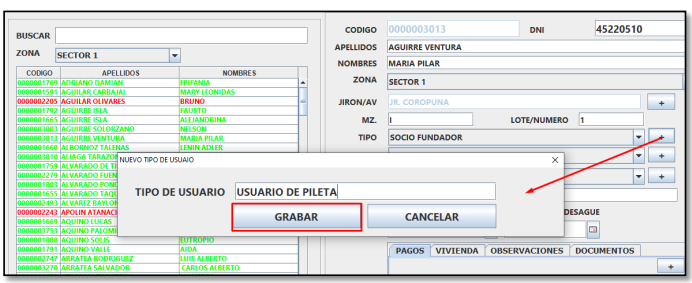


Imagen N° 039: Interfaz para agregar un nuevo tipo de usuario

- ✓ Rellene la información para el nuevo tipo de usuario y haga clic en el botón **"GRABAR"** para guardarlo.

6. Selección de Estado del Usuario:

- ✓ Seleccione el estado del usuario. Por defecto, se recomienda seleccionar **"Activo"**, como se muestra en la **Imagen N° 040: Selección de Estado del Usuario**.

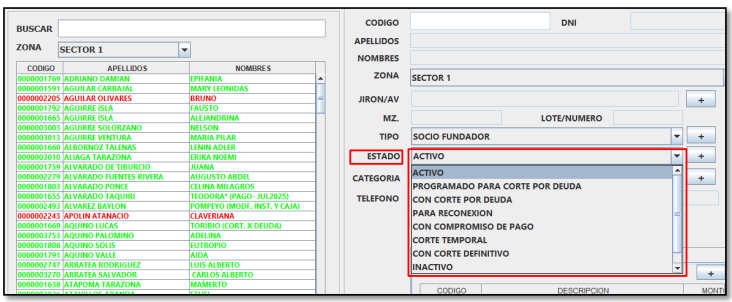


Imagen N° 040: Lista para seleccionar el estado del usuario

- ✓ **Para Agregar un Nuevo Estado (Opcional):** Si necesita añadir un nuevo estado, haga clic en el botón **"+"** al lado derecho de la lista desplegable (referencia **Imagen N° 041: Botón para Agregar Estado**).

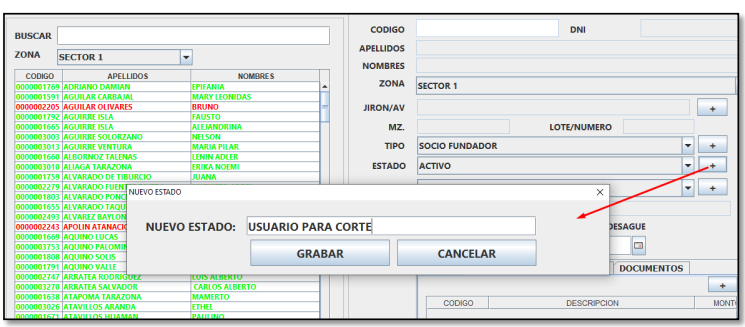


Imagen N° 041: Interfaz para agregar un nuevo estado al sistema

- ✓ Rellene la información para el nuevo estado y haga clic en el botón **"GRABAR"** para guardarlo.

7. Selección de Categoría del Usuario:

- ✓ Seleccione la categoría a la que pertenece el usuario, como se indica en la **Imagen N° 042: Selección de Categoría del Usuario**.

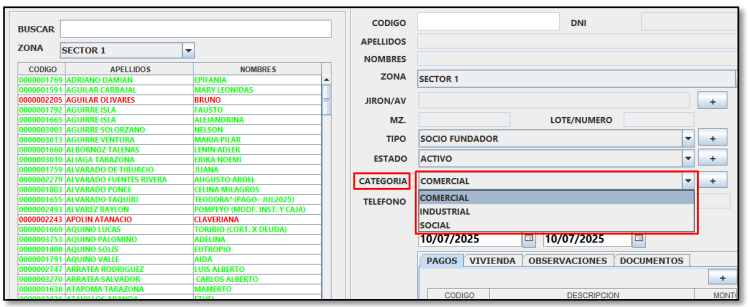


Imagen N° 042: Lista para seleccionar la categoría de usuario

- ✓ **Para Agregar una Nueva Categoría (Opcional):** Si la categoría deseada no está disponible, haga clic en el botón **"+"** ubicado al lado derecho de la lista desplegable (referencia **Imagen N° 043: Botón para Agregar Categoría**).

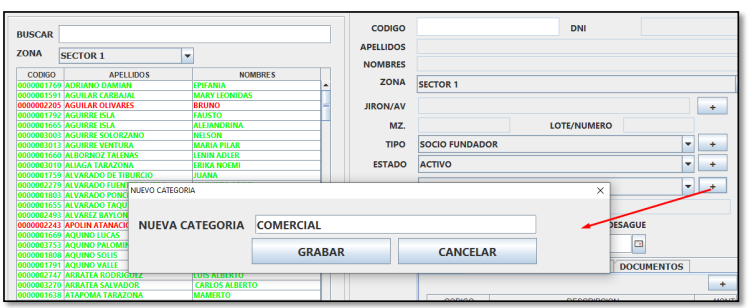


Imagen N° 043: Interfaz para agregar una nueva categoría al sistema

- ✓ Complete los datos para la nueva categoría y haga clic en el botón **"GRABAR"**.

8. Relleno de Campos Adicionales y Pestañas:

- ✓ Rellene los demás campos de información que sean necesarios.
- ✓ Explore las pestañas adicionales para completar la información detallada del usuario:

➢ **Pestaña "PAGOS":**

- ✓ Haga clic en el botón **"+"** para agregar los pagos mensuales que realizara el usuario y sus respectivos costos, como se muestra en la **Imagen N° 044: Pestaña Pagos y Agregado de Pagos Mensuales**.

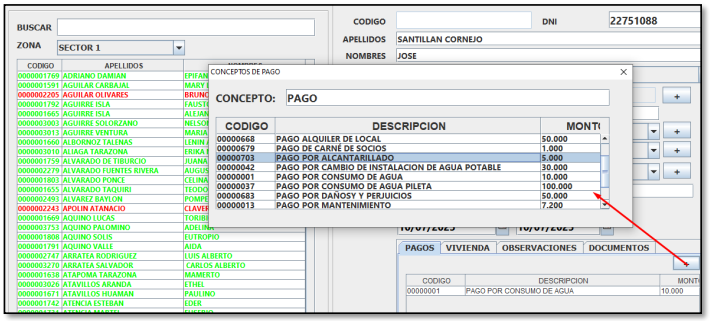


Imagen N° 044: Interfaz que permite buscar los conceptos de pago y agregar a la tabla

➢ **Pestaña "VIVIENDA":**

- ✓ Ingrese las características relevantes de la vivienda del usuario, según lo que se indica en la **Imagen N° 045: Pestaña Vivienda y Características de la Vivienda**.

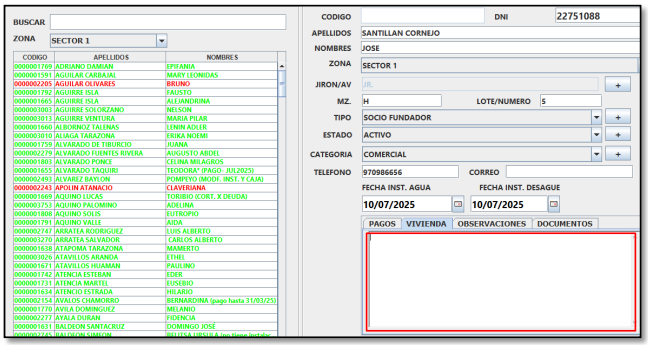


Imagen N° 045: Cuadro que permite ingresar datos de la vivienda

- **Pestaña "OBSERVACIONES":**
  - ✓ Utilice este espacio para añadir cualquier observación o nota importante relacionada con el usuario o su servicio, como se presenta en la **Imagen N° 046: Pestaña Observaciones**.

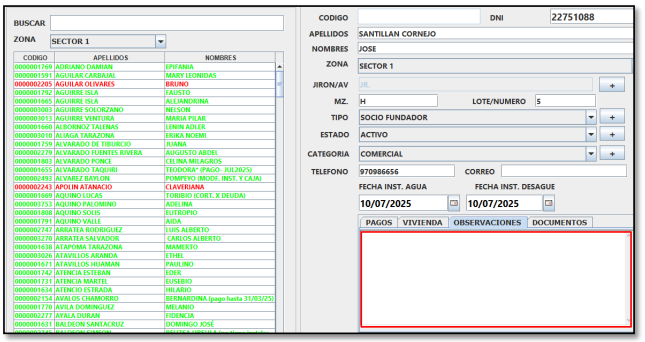


Imagen N° 046: Cuadro que permite ingresar algunas otras observaciones respecto al usuario

- **Pestaña "DOCUMENTOS":**
  - ✓ Esta sección permite cargar y gestionar archivos digitales asociados al usuario (contratos, comprobantes de pago, fotos, etc.), como se observa en la **Imagen N° 047: Pestaña Documentos**.

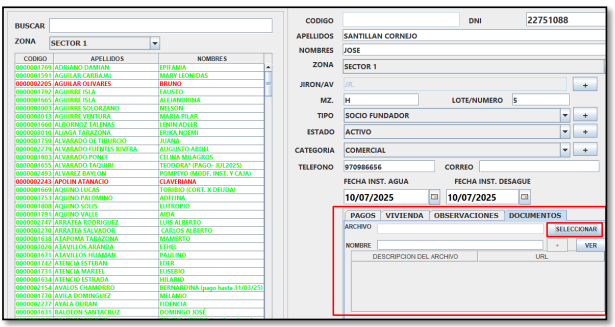


Imagen N° 047: Parte donde se cargará los archivos relacionado al usuario (contratos, fotos, etc.)

- **Para Cargar un Archivo:**
  - ✓ Haga clic en el botón **"SELECCIONAR"** para abrir el explorador de archivos y buscar el documento digital que desea cargar (referencia **Imagen N° 048: Selección de Archivo Digital**).

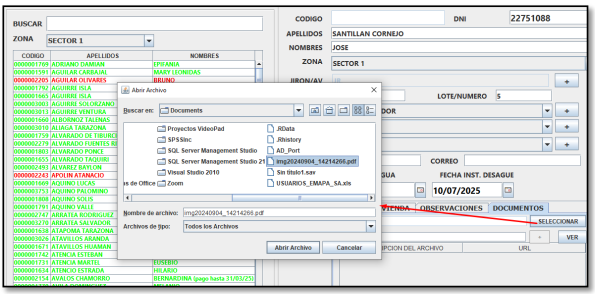


Imagen N° 048: En esta ventana se selecciona el archivo a cargar en cualquier formato

- ✓ Una vez seleccionado el archivo, haga clic en el botón **"+"** para agregarlo a la tabla de documentos y subirlo al servidor del sistema (referencia **Imagen N° 049: Agregado de Archivo a la Tabla y Servidor**).

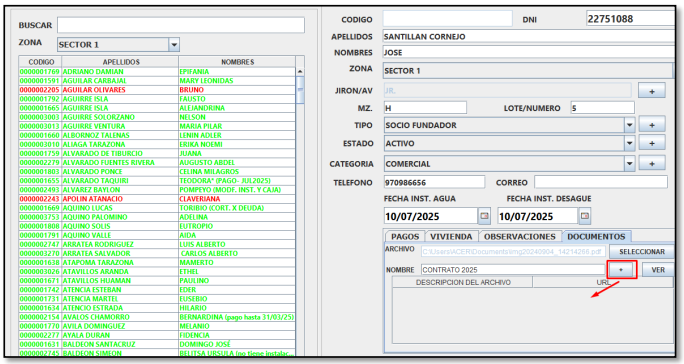


Imagen N° 049: Botón que permite agregar el archivo seleccionado a la tabla y al servidor

- **Para Visualizar un Archivo Cargado:**
  - ✓ Seleccione el archivo en la tabla y haga clic en el botón **"VER"** (referencia **Imagen N° 050: Botón para Ver Archivo Cargado**).

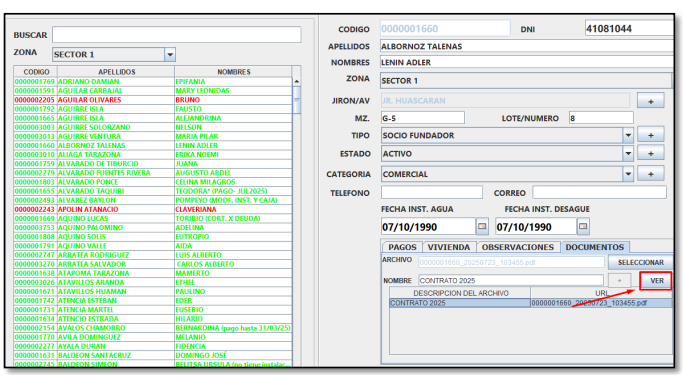


Imagen N° 050: Botón que permite visualizar el archivo cargado

- ✓ El sistema abrirá y visualizará el archivo digital, como se muestra en la **Imagen N° 051: Visualización del Archivo Digital**.

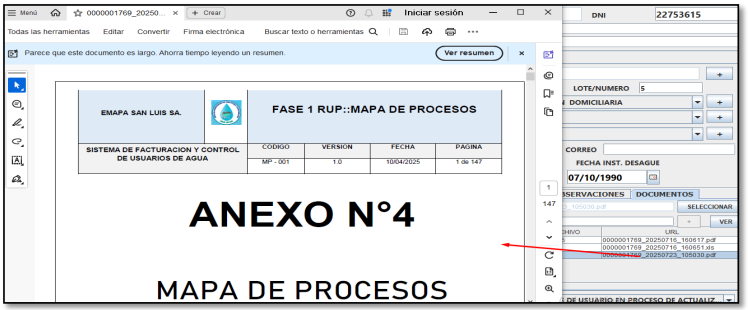


Imagen N° 051: Vista previa de del archivo cargado

9. Estado de Actualización de Datos:

- ✓ Seleccione el estado de la actualización de datos, según corresponda (referencia **Imagen N° 052: Selección de Estado de Actualización de Datos**).

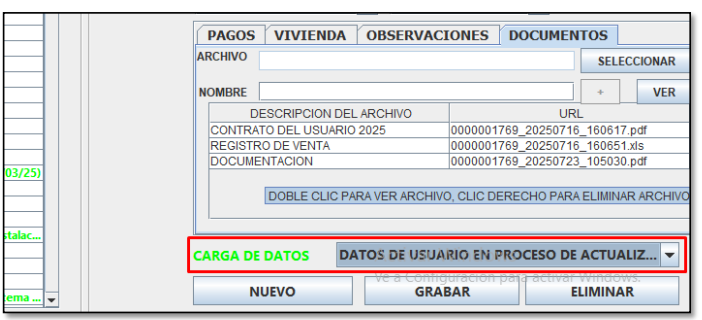


Imagen N° 052: Lista de selección del estado de actualización de datos

10. Guardar el Nuevo Usuario:

- ✓ Una vez que todos los datos estén completos y verificados, haga clic en el botón **"GRABAR"** para guardar la información del nuevo usuario en la base de datos (referencia **Imagen N° 053: Botón para Grabar el Usuario**).

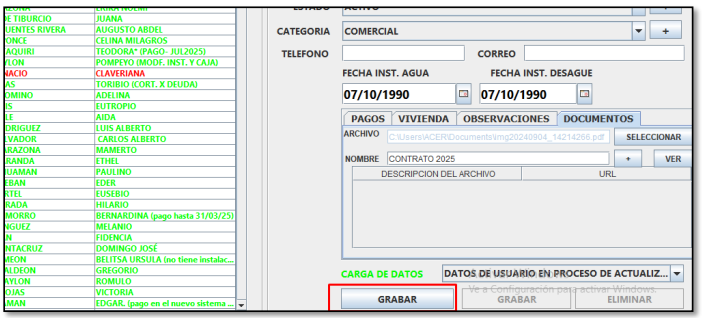


Imagen N° 053: Vista previa de la boleta a imprimir

Modificar Datos de un Usuario Existente

Para actualizar la información de un usuario ya registrado:

1. **Buscar el Usuario:**

- ✓ En la parte superior izquierda de la interfaz de Gestión de Usuarios (Imagen N° 035), ingrese una palabra clave (nombre, apellido, etc.) en la caja de texto para buscar al usuario. Este proceso se detalla en la **Imagen N° 054: Búsqueda de Usuario por Palabra Clave**.

Imagen N° 054: Caja de texto que permite filtrar usuarios

- ✓ Presione la tecla **[ESPACIO]** (o **[ENTER]**) para filtrar la lista de usuarios. Los resultados que contengan la palabra clave ingresada se cargarán en la tabla de la interfaz.

2. **Seleccionar el Usuario:**

- ✓ En la tabla de resultados, haga clic en la fila correspondiente al usuario cuyos datos desea modificar para seleccionarlo.
- ✓ Al seleccionar el usuario, todos sus datos actuales se cargarán automáticamente en los campos de entrada y las pestañas a la derecha de la tabla.

3. **Habilitar Edición:**

- Haga clic en el botón **"MODIFICAR"** (referencia **Imagen N° 055: Botón Modificar Usuario**).

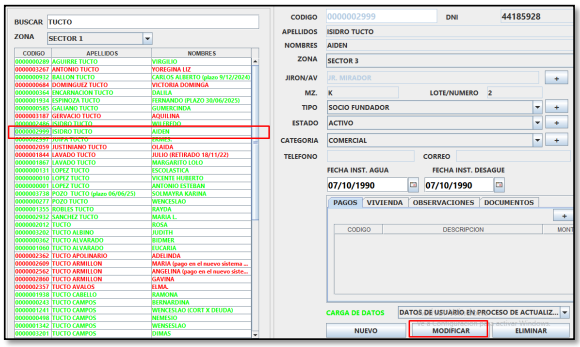


Imagen N° 055: El botón para modificar datos de un usuario

- ✓ Esto habilitará todos los campos de entrada, permitiéndole realizar los cambios necesarios en la información del usuario (nombre, dirección, tipo de usuario, etc., y también en las pestañas como Pagos, Vivienda, Observaciones, Documentos).

4. **Guardar los Cambios:**

- ✓ Una vez que haya realizado todas las modificaciones deseadas, haga clic en el botón **"GRABAR"** (referencia **Imagen N° 056: Botón Grabar Cambios**).

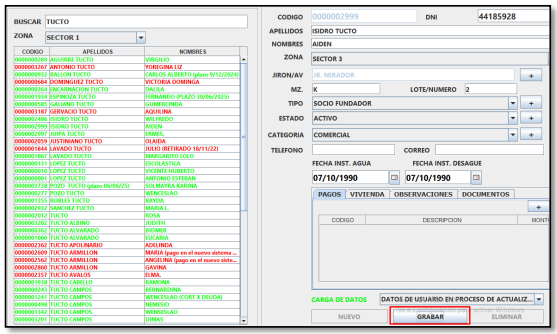


Imagen N° 056: El botón para guardar datos actualizados de un usuario

- ✓ Los cambios se guardarán en la base de datos.

Eliminar datos de un Usuario

Para eliminar un registro de usuario del sistema:

1. **Buscar y Seleccionar el Usuario:**
   - ✓ Siga los mismos pasos de búsqueda descritos en el apartado de "Modificar Datos" (Paso 1 y 2). Ingrese una palabra clave en la parte superior izquierda y seleccione el usuario deseado en la tabla de resultados.
2. **Iniciar Eliminación:**
   - Con el usuario seleccionado en la tabla, haga clic en el botón **"ELIMINAR"** (referencia **Imagen N° 057: Botón Eliminar Usuario**).

   (***Aquí es donde usted insertaría la Imagen N° 057 en su manual***)

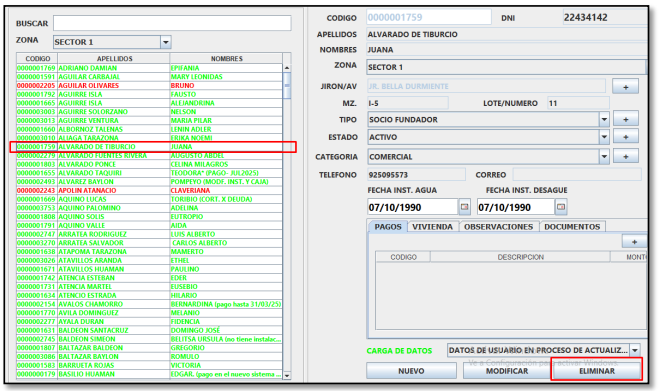


Imagen N° 057: El botón para eliminar datos de un usuario

Dirección legal: Urb. Paseo del Mar
Nuevo Chimbote, Santa, Ancash
Correo electrónico:
ed.honexus@gmail.com**Teléfono**:
978653152

ISBN: 978-612-99293-1-6