



## **Actividad | 3 | Programa Banco**

### **Mexicano (parte 2)**

#### **Nombre del curso**

---

Ingeniería en Desarrollo de Software



TUTOR: Aarón Iván Salazar Macías.

---

ALUMNO: Uziel de Jesús López Ornelas.

---

FECHA: 5 de agosto del 2025.

---

## Tabla de Contenido

<b>Introducción .....</b>	<b>1</b>
<b>Descripción .....</b>	<b>2</b>
<b>Justificación .....</b>	<b>3</b>
<b>Desarrollo .....</b>	<b>4</b>
Interfaz.....	4
Codificación.....	5
<b>Conclusión .....</b>	<b>14</b>
<b>Referencias .....</b>	<b>15</b>

## Introducción

Tocaremos algunos puntos para repasar y dar paso a la última actividad de la materia. La interfaz, esta se define como una variable de una clase que es abstracta donde todas sus condiciones en todos los métodos deben ser abstractos. Una clase puede implementar una o más interfaces (separadas por comillas ""). La clase que implementa una interfaz tiene dos opciones:

- Implementan todos los métodos de la interfaz.
- Implementan solo algunos de los métodos de la interfaz, pero esa clase debe ser una clase abstracta (**abstract**).

Una interfaz es, simplemente, una lista de métodos no implementados; además, puede incluir la declaración de constantes. Una clase abstracta, por otro lado, puede incluir métodos implementados, y no implementados o abstractos.

- Una clase hereda con la palabra **extends**.
- Una clase implementa con la palabra **implements**.

Una colección, por su parte, representa un grupo de objetos. Estos objetos son conocidos como elementos. Para trabajar con un conjunto de elementos, es necesario tener un almacén donde se puedan guardar. En el lenguaje de Java se emplea la interfaz genérica **Collection**. Con esto se permite almacenar cualquier tipo de objeto y usar una serie de métodos comunes.

Existen distintos tipos de colecciones que destacan, como los siguientes:

- **Set**: Define una colección que no se puede contener elementos duplicados.

Dentro de la interfaz **Set**, existen varios tipos de implementaciones:

- **HashSet**: Almacena los elementos en una tabla **hash**. Es la que tiene mejor rendimiento de todos.
- **TreeSet**: Almacena los elementos ordenables en función a sus valores. Es mucho más lento que **HashSet**.
- **LinkedHashSet**: Esta implementación almacena los elementos en función del orden de inserción.

## Descripción

Con esto nos abrimos a otro horizonte de la interfaz, **List** define una sucesión de elementos. A diferencia de la interfaz **Set**. La interfaz **List** si admite elementos duplicados. Agregado a ello este ofrece métodos que permiten mejorar los siguientes puntos:

- Acceso posicional a elementos.
- Búsqueda de elementos.
- Iteración sobre elementos.
- Rango de operación.

Dentro de la interfaz, **List** existen varios tipos de implementaciones realizadas dentro de la plataforma Java:

- **ArrayList**: Aumenta su tamaño según la colección de elementos.
- **LinkedList**: Mejora el rendimiento en ciertas ocasiones. Se basa en una lista doblemente enlazada de los elementos. Cuando se usa una implementación de **List** variara en función de la situación que nos encontramos.

Generalmente **ArrayList** será la implementación que usemos en la mayoría de las situaciones. Los **drivers JDBC** (equivalente a ODBC), permiten efectuar conexiones con bases de datos. El lenguaje los incorpora como punto de extensibilidad en el sistema a la hora de conectarnos a una base de datos.

El concepto **driver** hace referencia al conjunto de clases necesarios que implementa de forma nativa, el protocolo de comunicación para la base de datos como lo pueden ser MySQL y SQLServer.

El paquete “Java.sql” contiene los siguientes elementos:

- **Driver**: Permite conectarse a una base de datos.
- **DriverManager**: Permite gestionar todos los drivers instalados en el sistema.
- **DriverPropertyInfo**: Proporciona información sobre un driver.
- **Connection**: Representa una conexión con una base de datos.
- **DataBaseMetaData**: Proporciona la información acerca de una base de datos.
- **Statement**: Permite ejecutar sentencias SQL sin parámetros.
- **PreparedStatement**: Permite ejecutar sentencias SQL con parámetros de entrada.
- **CallableStatement**: Permite ejecutar sentencias SQL con parámetros de entrada y

salida.

- **ResultSet:** Contiene las filas o registros obtenidos al ejecutar un **SELECT**.
- **ResultSetMetaData:** Permite obtener información sobre un **ResultSet**.

### Justificación

Es importante mencionar que el manejo de excepciones agiliza el control de errores al permitir que tu programa defina un bloque de código, el cual se ejecuta automáticamente cuando ocurre un error.

Normalmente en Java, este tipo de excepción están representadas por clases. Todas las clases se derivan de una clase llamada **Throwable**.

Hay dos tipos de subclases directas de **Throwable**:

- **Exception.**
- **Error.**

La excepción de tipo **Error** están relacionadas con errores que ocurren en la máquina virtual de Java y no en el programa. El manejo de excepciones Java se gestiona a través de cinco palabras clave:

- **Try.**
- **Catch.**
- **Throw.**
- **Throws.**
- **Finally.**

Todos los anteriores forman un subsistema interrelacionado en el que el uso de una implica el uso de la otra. Las declaraciones del programa que se desean supervisar para excepciones están contenidas dentro de un bloque **try**, si se produce algo inesperado dentro de este bloque, se lanza una excepción. El código puede “atrapar” la excepción usando **Catch**.

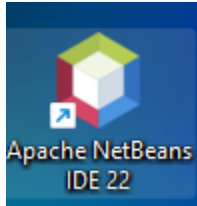
Para lanzar manualmente una excepción se usa la palabra clave **Throws**. Cualquier código que debe ejecutarse al salir de un bloque **try** se coloca en un bloque **finally**.

Con todo lo anterior tenemos las herramientas para vincular nuestra base de datos al lenguaje de programación Java y con este lograr darle vida y también acciones que permitan el funcionamiento total de la misma.

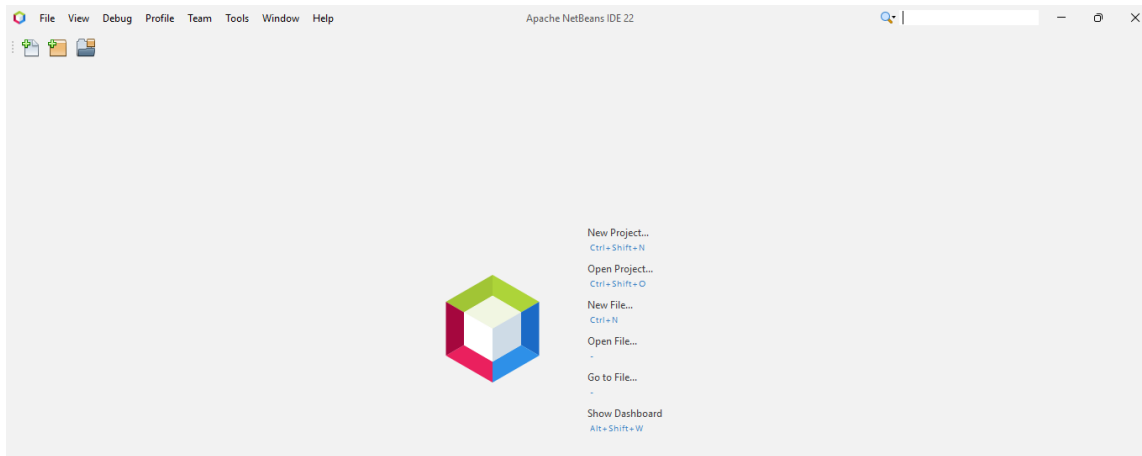
## Desarrollo

### Interfaz

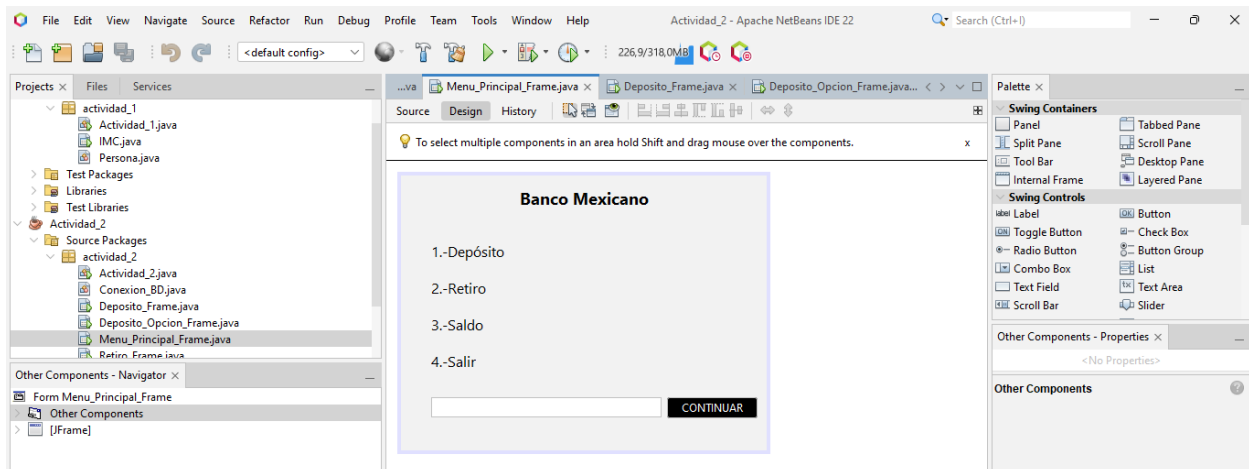
Para iniciar con esta etapa primero abriremos la aplicación de “NetBeans” en nuestro ordenador:



Ingresamos y esperamos a que cargue toda la configuración necesaria en donde se nos abrirá la ventana siguiente:

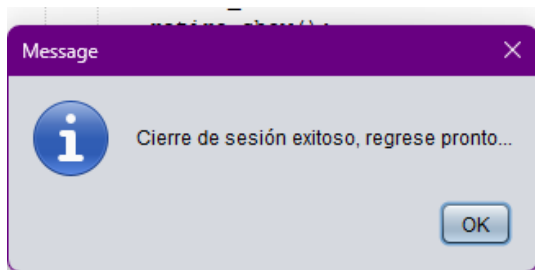


Trabajaremos con el proyecto anterior de la actividad número #2:



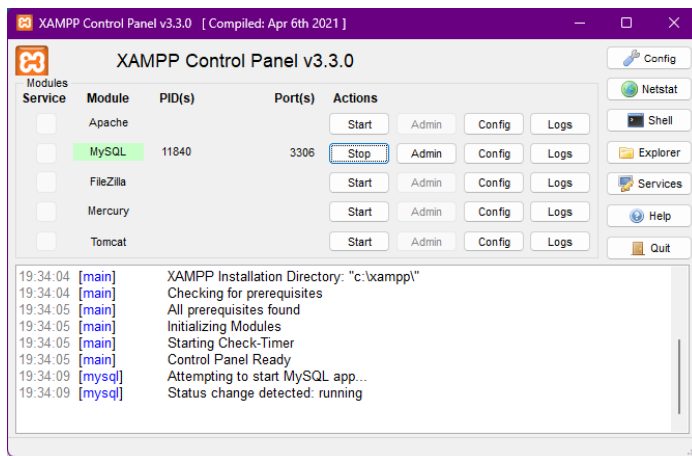
La actividad final nos indica que tenemos que diseñar la ventana de saldo:

La opción de “Salir” que está en el menú principal nos tiene que mostrar una ventana de cerrar sesión, como la siguiente:

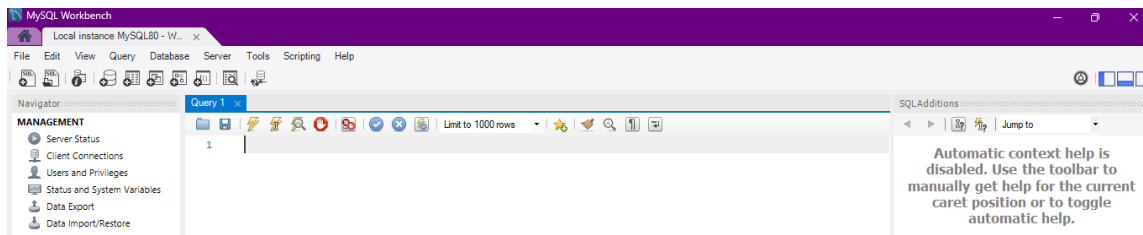


## Codificación

Es momento de enlazar la base de datos con Java, en el cual el daremos vida a las acciones e instrucciones que le demos a la aplicación, iremos a la aplicación “XAMPP”, y seleccionaremos la siguiente opción:



Abriremos MySQL para crear la base de datos:



Vamos a crear la base de datos con el mismo nombre que colocamos en la conexión de Java:

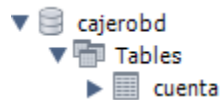
```
CREATE DATABASE cajeroibd;
```

Ahora es momento de crear la tabla con los siguientes valores:

```
USE cajeroibd;
```

```
CREATE TABLE cuenta
(
  id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
  saldo FLOAT
);
```

Revisamos que este correcto; como se puede observar si esta creada:



Vamos a insertar un valor en la tabla de “500”:

ID	Saldo
1	500
NULL	NULL

Nos dirigiremos a Java, en la ventana de “Deposito”:

### Depósito

Cantidad a depositar:



Seleccionaremos el botón de “DEPOSITAR” para darle la acción mediante código:

```
Connection con;
Conexion_BD conn = new Conexion_BD();

try
{
    con = conn.getConnection();

    PreparedStatement ps = con.prepareStatement("UPDATE cuenta SET saldo = saldo + ? WHERE id = 1");
    ps.setString(1, jtf_Deposito.getText());

    int res = ps.executeUpdate();

    if(res > 0)
    {
        JOptionPane.showMessageDialog(null, "Depósito exitoso");
    }
    else
    {
        JOptionPane.showMessageDialog(null, "Error en el depósito");
    }

    con.close();
}
catch(Exception e)
```

```
    }
    catch(Exception e)
    {
        System.err.println(e);
    }

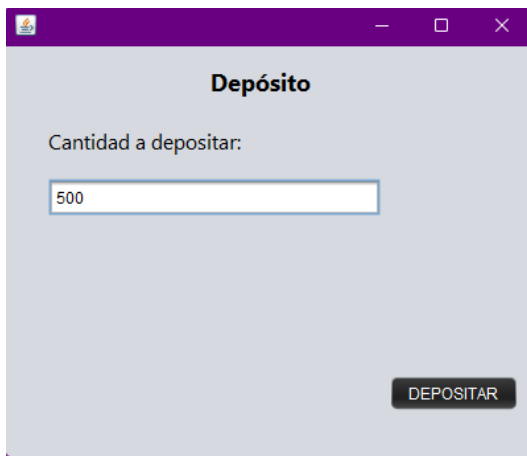
    Deposito_Opcion_Frame deposito_opcion = new Deposito_Opcion_Frame();
    deposito_opcion.show();
    this.hide();

    jtf_Deposito.setText("");
}
```

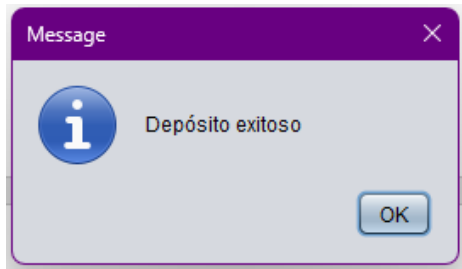
Probaremos con depositar una cantidad en la base de datos, tenemos “1000” de saldo en un inicio:

	id	saldo
▶	1	1000
*	NULL	NULL

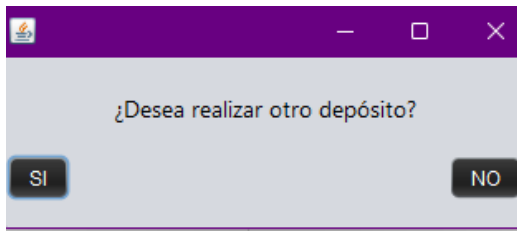
Ejecutamos el programa y vamos a la ventana de “Depósito”, vamos a depositar “500”:



Tenemos una ventana de confirmación de que el depósito se realizó correctamente:



Confirmamos y nos aparece otra ventana donde preguntan si queremos hacer otro depósito:



Seleccionaremos “NO” para visualizar la base de datos:

```
SELECT * FROM cajeroibd.cuenta;
```

Actualizamos la base de datos para visualizar el resultado; como se puede observar la base de datos reconoce el monto que se anexo:

	id	saldo
▶	1	1500
*	NULL	NULL

Nos dirigimos a la ventana de “Retiro” para codificar lo necesario, seleccionamos el botón de “RETIRAR”:



Una vez dentro el código será muy parecido, solo con ciertas diferencias, en vez de sumar vamos a restar:

```
...
```

La caja de texto es distinta ya que estamos en la ventana de “Retiro”:

```
, jtf Retiro
```

Y los mensajes tienen que coincidir con la ventana representada:

```
"Retiro exitoso");
```

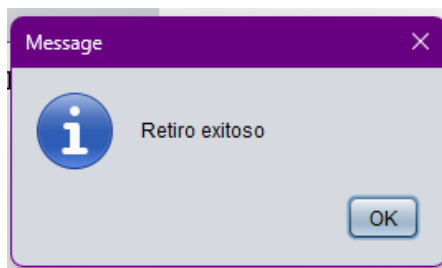
```
"Error en el Retiro");
```

Ejecutaremos el programa y nos dirigiremos a la ventana de “Retiro”:

Retiraremos “500” y seleccionaremos el botón de “RETIRAR”:



Como se observa el retiro se realizó correctamente:



Nos dirigimos a nuestra base de datos para revisar el movimiento, si se tenía “1500” menos los “500” que le restamos tendríamos “1000” de saldo:

```
SELECT * FROM cajeroibd.cuenta;
```

	id	saldo
▶	1	1000
*	NULL	NULL

El código completo sería el siguiente:

```

Connection con;
Conexion_BD conn = new Conexion_BD();

try
{
    con = conn.getConnection();

    PreparedStatement ps = con.prepareStatement("UPDATE cuenta SET saldo = saldo - ? WHERE id = 1");
    ps.setString(1, jtf_Retiro.getText());

    int res = ps.executeUpdate();

    if(res > 0)
    {
        JOptionPane.showMessageDialog(null, "Retiro exitoso");
    }
    else
    {
        JOptionPane.showMessageDialog(null, "Error en el Retiro");
    }

    con.close();
}

    con.close();
}
catch(Exception e)
{
    System.err.println(e);
}

this.hide();
Menu_Principal_Frame back = new Menu_Principal_Frame();
back.show();
}

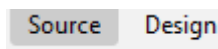
```

Nos dirigiremos ahora a la ventana de “Saldo”:



The image shows a Java Swing window titled "Saldo". It contains a label "Monto:" and a text input field for entering a value. A button labeled "CONTINUAR" is located at the bottom right of the window.

Y nos moveremos a la sección de “Source”:



Buscamos el siguiente apartado:

```
public Saldo_Frame() {
    initComponents();
}
```

Pegaremos el código que teníamos anteriormente, pero con ligeras diferencias, se agregan librerías para que no de error:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import javax.swing.JOptionPane;
import java.sql.ResultSet;
```

En este apartado, vamos a querer que nos muestre los datos de “saldo” de la tabla de “cuenta”:

```
PreparedStatement ps = con.prepareStatement("SELECT saldo FROM cuenta WHERE id = 1");
```

En este lugar si es correcto la instrucción nos mostrara el saldo en la caja de texto, si no es así nos mandara un error:

```
if (rs.next()) {
    jtf_Saldo.setText(rs.getString("saldo"));
} else {
    JOptionPane.showMessageDialog(null, "Error");
}
```

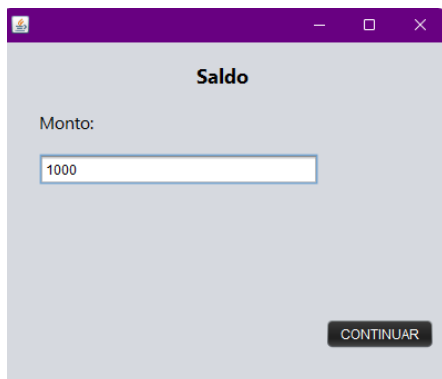
Vamos a programar el botón de continuar para que nos mande al menú principal:

CONTINUAR

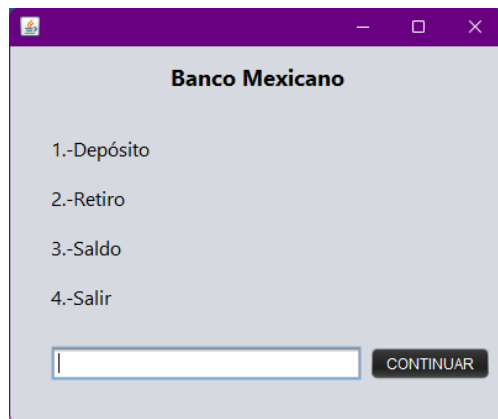
Dentro de este colocaremos la codificación necesaria:

```
this.hide();
Menu_Principal_Frame back = new Menu_Principal_Frame();
back.show();
}
```

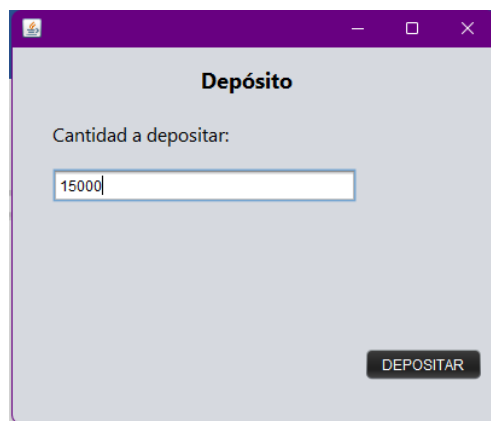
Ejecutaremos el programa para revisar el monto que se tiene en la cuenta, anteriormente se tenía era de “1000”, veremos si eso nos muestra la ventana:



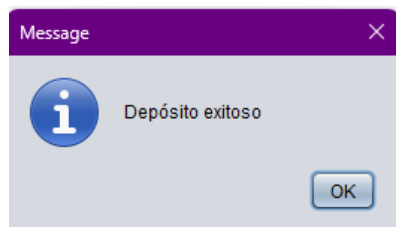
Si seleccionamos el botón de “CONTINUAR” este nos mandara al menú principal:



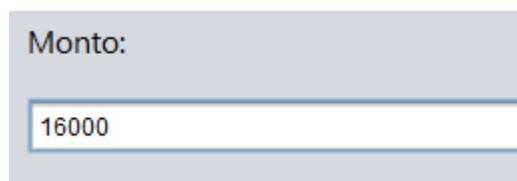
Si agregamos a la cuenta “15000” en el depósito:



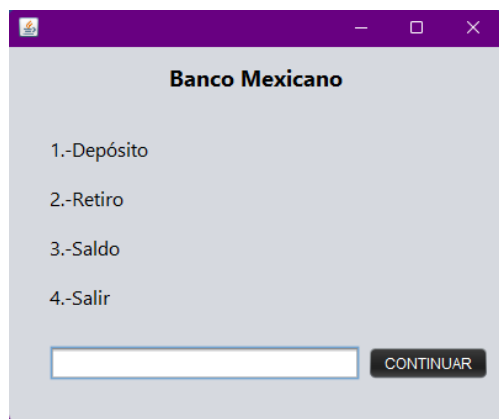
Como podemos observar el depósito se realizó con éxito:



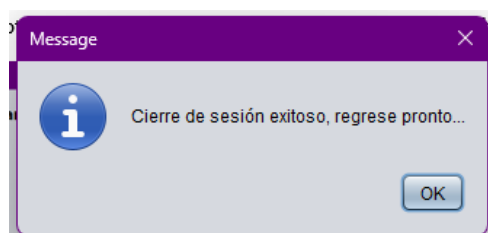
Vamos a la ventana de “Saldo”, ahora tenemos “16000” de saldo:



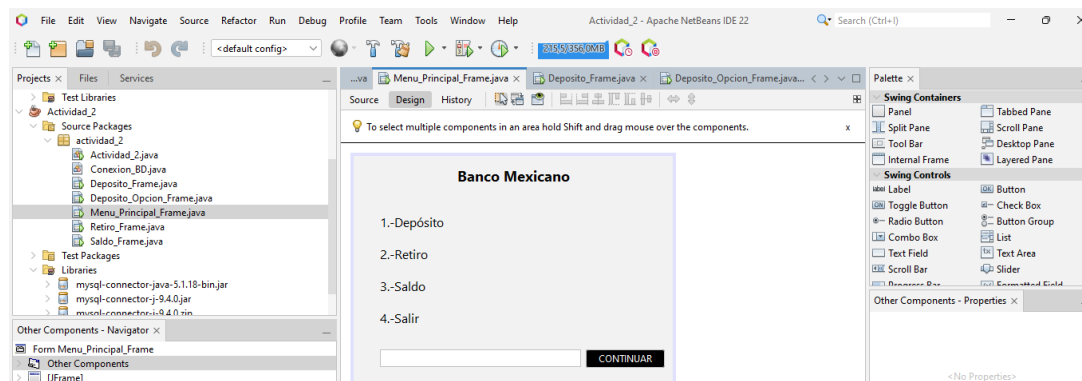
Si damos clic al botón de “CONTINUAR” este nos manda al menú principal:



Si colocamos el 4 en la caja de texto este nos mostrara el siguiente mensaje:



Este nos cerrara el programa por completo:



### **Conclusión**

De acuerdo a lo que se realizó en esta última actividad me he dado cuenta de la importancia de conocer todas las instancias y librerías que se necesitan para resolver o ejecutar alguna instrucción para que el ordenador la reciba. En todas las aplicaciones que se utilicen los lenguajes pueden variar, pero las instrucciones o sintaxis se mantienen para brindar de manera universal aquella herramienta tan prescindible y única en la programación, casos como el vincular una base de datos para mostrar información sencilla sin tantas complicaciones hasta tener servidores, contraseñas encriptadas y demás datos sensibles que ocupan un mayor grado de control y acceso en la autenticación así como las diferentes redes que se utilicen para trabajar será de vital importancia el conocimiento y la antelación ante posibles riesgos. En esta última actividad ubique puntos en los cuales cualquiera puede tropezar y que básicamente es tan normal que es necesario tener cuidado, recordar que cuando se pasan datos a una base estos tienen que coincidir con el nombre de nuestra tabla y elementos que la conforman puesto que estos al no ser reconocidos no se vincularan o se completara la instrucción de manera correcta y eficiente.

### **Link de GitHub**

<https://github.com/UZLOP984/Lenguajes-de-Programaci-n-IV.git>



## Referencias

*Java SE Technologies - Database.* (s. f.). <https://www.oracle.com/java/technologies/javase/javase-tech-database.html>

*What Is a Database?* (2020, 24 noviembre). <https://www.oracle.com/mx/database/what-is-database/>

González, J. D. M. (2021, 20 junio). *Librerías o bibliotecas.*  
<https://www.programarya.com/Cursos/Java/Librerias>

*Uso de Java Management Cloud service.* (s. f.). Oracle Help Center. <https://docs.oracle.com/es-ww/iaas/jms/doc/java-libraries.html>