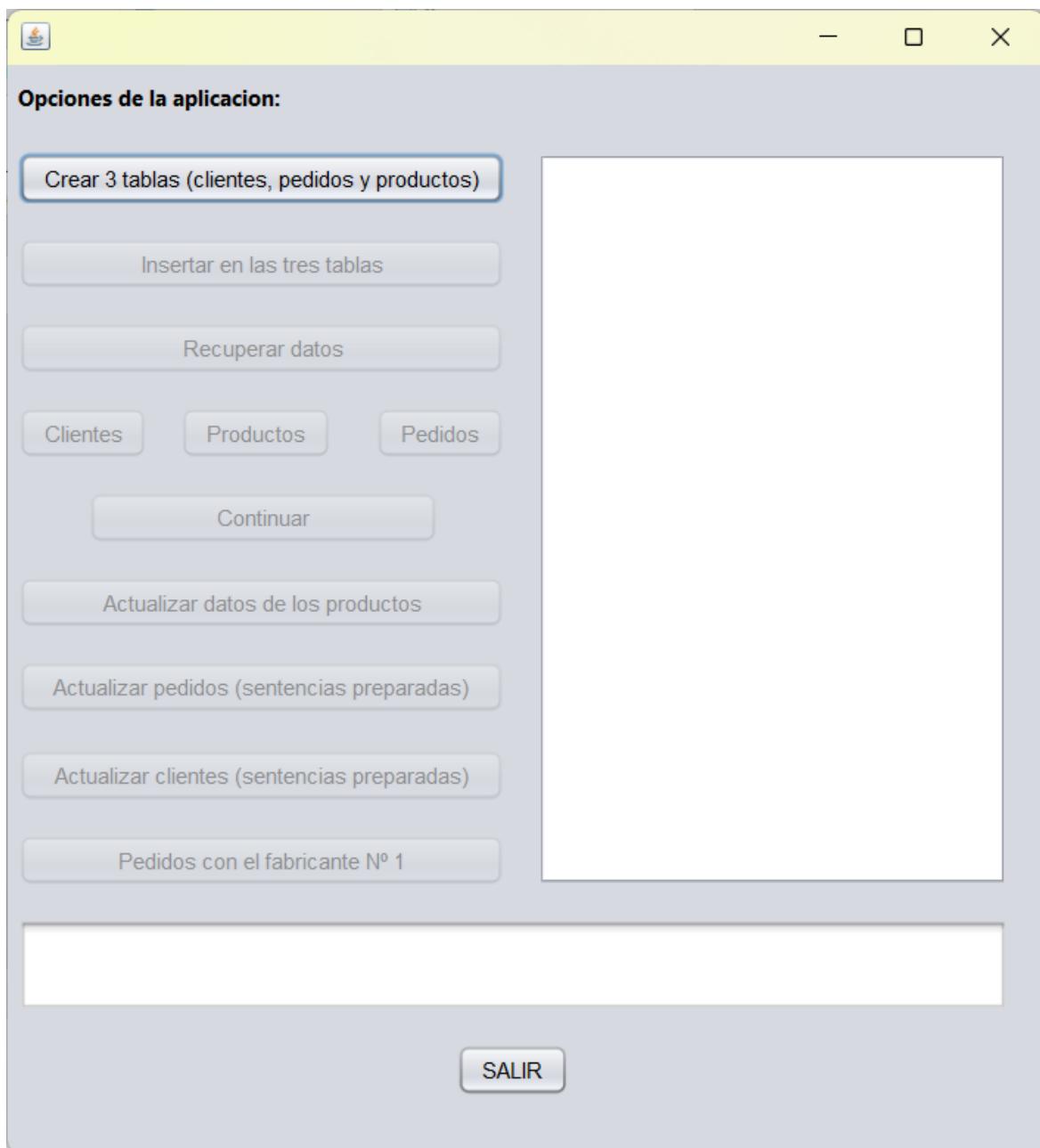


Tarea 3 para Acceso a Datos



Diego Manuel Carrasco Castañares

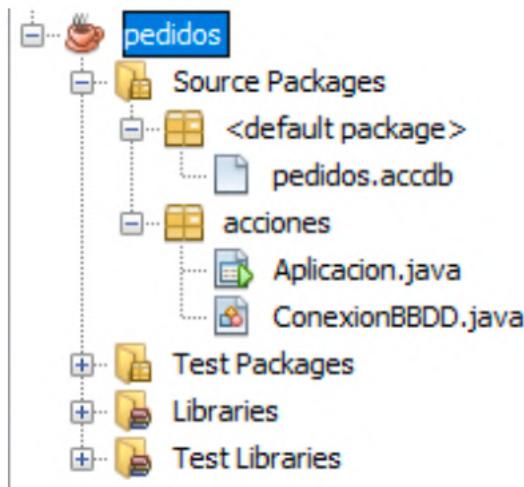
Indice:

Creación del proyecto en NetBeans	7
Creación de la base de datos en access	7
Creación de paquetes y clases	9
Instalación libreria ucanaccess	10
Instalación resto de librerías	10
Creación de interface	11
Código clase "ConexionBBDD"	12
Código clase "Aplicación"	13
Código botón para crear las tablas	14
Código botón para insertar en las tablas	15
Código botones para recuperar datos de las tablas	17
Código botón continuar	19
Código para actualizar productos	20
Código para actualizar pedidos	21
Código par actualizar clientes	23
Código para mostrar pedidos del proveedor 1	24
Código botón salir	26
Método para comprobar si una tabla existe	26
Metodo para recuperar datos de las tablas	26

Elabora un documento .pdf con la explicación de la tarea. Si no lo entregas la puntuación de la tarea se verá reducida en 2 puntos.

Hay que crear una aplicación o proyecto en java con Netbeans, que tendrá la siguiente estructura y debe hacer lo siguiente:

1.- Crea el proyecto en Netbeans con las dos clases del paquete acciones (0,25 puntos) con la librerías para la conexión con ucanaccess. Los jar deben ser librerías del proyecto (5 jar en Libraries). (0,25 puntos)



Las dos clases una para la conexión (ejemplo: ConexionBBDD.java) y otra para manejar la base de datos (Aplicación.java).

3.- Crea una base de datos en Access llamada pedidos, dentro del proyecto. (0,25 puntos)

Debe verse como la imagen en Netbeans

El siguiente texto es más o menos lo que escucháis en los vídeos.

4.- El proyecto hay que prepararlo de tal manera que nos permita, utilizando una interfaz , acceder a la base de datos pedidos creada en Access.

Recuerda la librería ucanaccess.

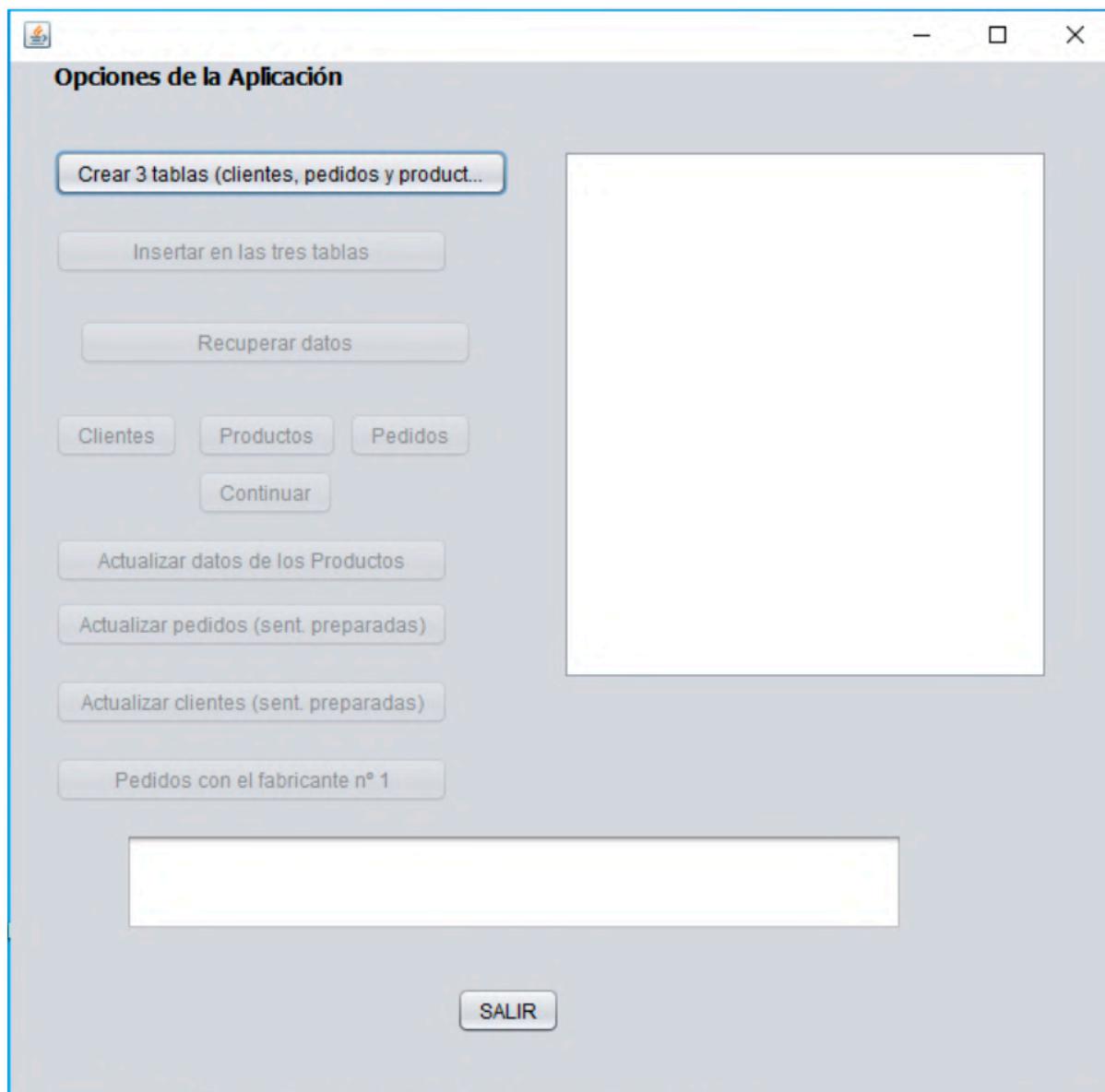
El vídeo 1 te cuenta en general el proyecto o aplicación.

Hay que controlar constantemente las posibles excepciones en cada opción, por ejemplo, si hemos seleccionado la opción de crear las tres tablas y ya están creadas en la base de datos pues que controle esa excepción y muestre el mensaje.

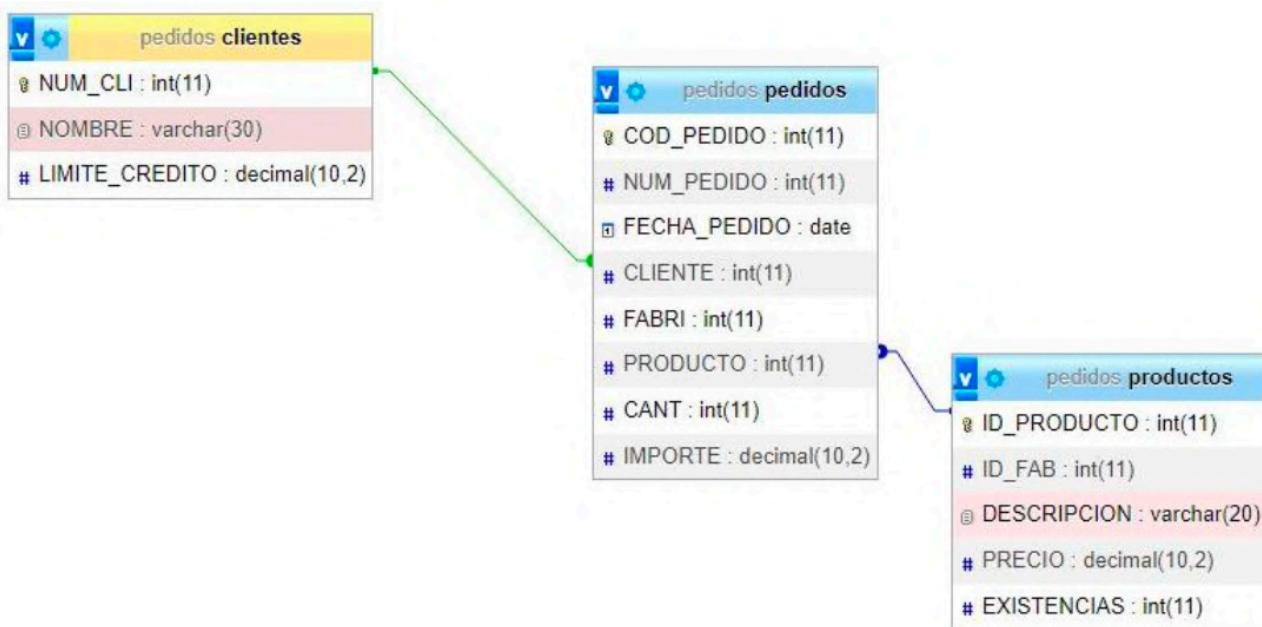
En el vídeo 2 se explica cómo va a funcionar esa interfaz.

La interfaz sólo tendrá una opción visible y se irán ejecutando de forma secuencial, ve quedando no visible la opción al pulsarla y pon visible la nueva opción junto con la muestra de la información correspondiente (el vídeo te lo muestra). Antes de nada en la clase conexión tenéis que programar simplemente para que haga la conexión a la base de datos pedidos creadas en Access (0,75 puntos).

Luego nos pasamos ya a la clase aplicación desde donde se instanciará a la clase conexión una vez por cada acción que se ejecute y se creará la interfaz donde tendremos las posibles opciones.



5.- La primera opción crear tres tablas consiste en crear la tabla Clientes, pedidos y productos. (1 punto)



6.- La segunda opción es para insertar cuatro registros en cada una de las tres tablas. (1 punto)

Los datos a insertar son:

En clientes (2101,'LUIS GARCIA','3000'), (2102,'ANGEL LUIS GARCIA','5000'), (2103,'MARIA MARIT','3500'), (2104,'EVA LOPEZ','3000')

En pedidos (1,0001,'1/1/2012',2101,1,001,20,0),
 (2,0002,'1/1/2012',2102,1,002,2,110),
 (3,0003,'2/1/2012',2102,2,002,200,NULL),
 (4,0004,'2/1/2012',2101,1,002,50,NULL)

En productos (001, 1 , 'ARANDELA',0.58,100), (002,
 1 , 'BISAGRA',1.58,10),(003,1,'JUNTA',0.58,1),
 (004,2,'RODAMIENTO',0.58,25),

7.- La siguiente opción es para recuperar datos de clientes de productos o de pedidos y también el poder continuar a la siguiente opción. (1 punto)

8.- La opción de actualizar datos de los productos es para actualizar el producto cuyo identificador es 002 a un precio de 2. (1 punto)

9.- La opción de actualizar pedidos hay que realizarla con sentencias preparadas preparedStatement y debe actualizar el importe el pedido 3 a 200. (1 punto)

10.- La opción de Actualizar clientes (1 punto) que también debe hacerse con sentencias preparadas preparedStatement lo que hace es modificar el nombre y el límite de crédito del cliente 2101, el nombre va a ser ahora Charo Holgado y el límite de crédito 1000.

En una sola sentencia

11.- La opción de listar pedidos de un cliente (1 punto) hay que hacer primero mostrar lo que vende el fabricante 1 que está en la tabla Productos.

A continuación con otra sentencia y utilizando las tres tablas, seleccionar los pedidos al fabricante 1, mostrando por cada pedido: el código y número del pedido, nombre del cliente, la descripción del producto y la cantidad.

12.- Y por último en la opción Salir cerramos la conexión a la base de datos con su correspondiente try-catch-finally. (1 punto)

13.- Si el proyecto anterior lo hubieses hecho con el JDBC de mysql (mira la imagen del apartado 3 de la unidad) ¿ Cómo hubiesen sido las líneas para la conexión a la base de datos pedidos ? Supón que tienes esta base de datos almacenada en la carpeta de mysql (0,5 puntos)

Criterios de puntuación. Total 10 puntos.

La tarea consta de 10 actividades a desarrollar que, en conjunto, sumarán 10 puntos.

Indicaciones de entrega.

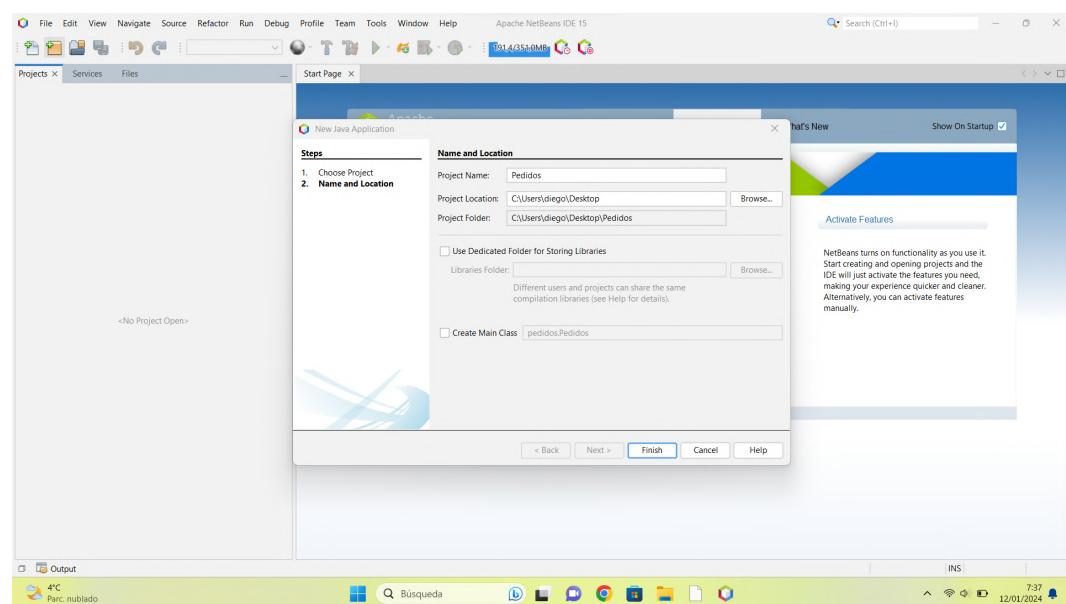
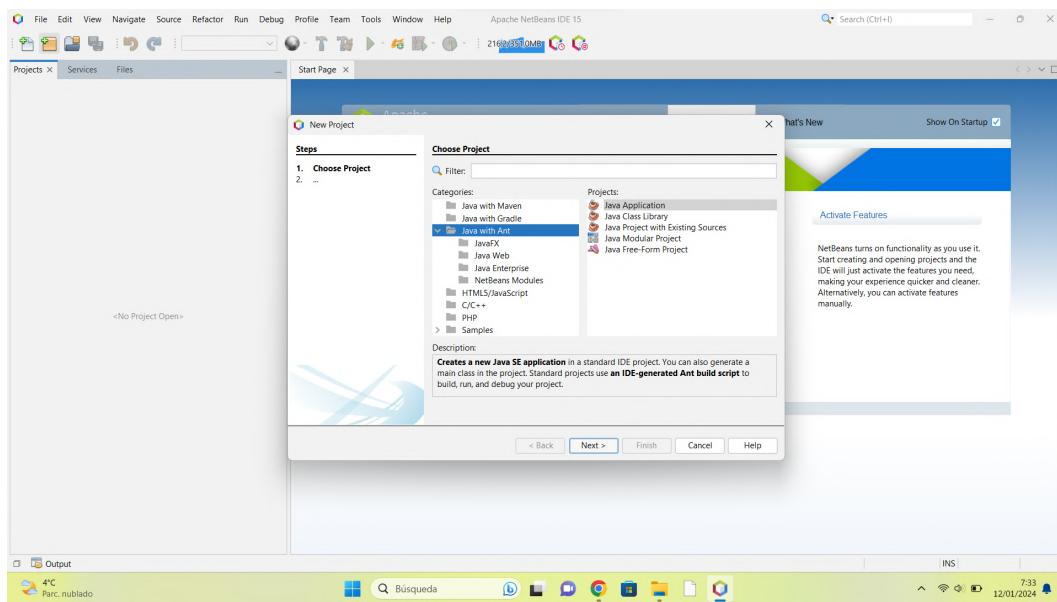
Una vez realizada la tarea entregarás un fichero comprimido con el proyecto y con el documento donde explicas el desarrollo de la tarea (pantallazos de ejecución y código de la ejecución de cada botón (en formato pdf) donde figuren las respuestas correspondientes. El envío se realizará a través de la plataforma de la forma establecida para ello, y el archivo se nombrará siguiendo las siguientes pautas:

apellido1_apellido2_nombre_ADxx_Tarea

Asegúrate que el nombre no contenga la letra ñ, tildes ni caracteres especiales extraños. Así por ejemplo la alumna Begoña Sánchez Mañas para la primera unidad del MP de AD, debería nombrar esta tarea como...

sanchez_manas_begona_AD02_Tarea

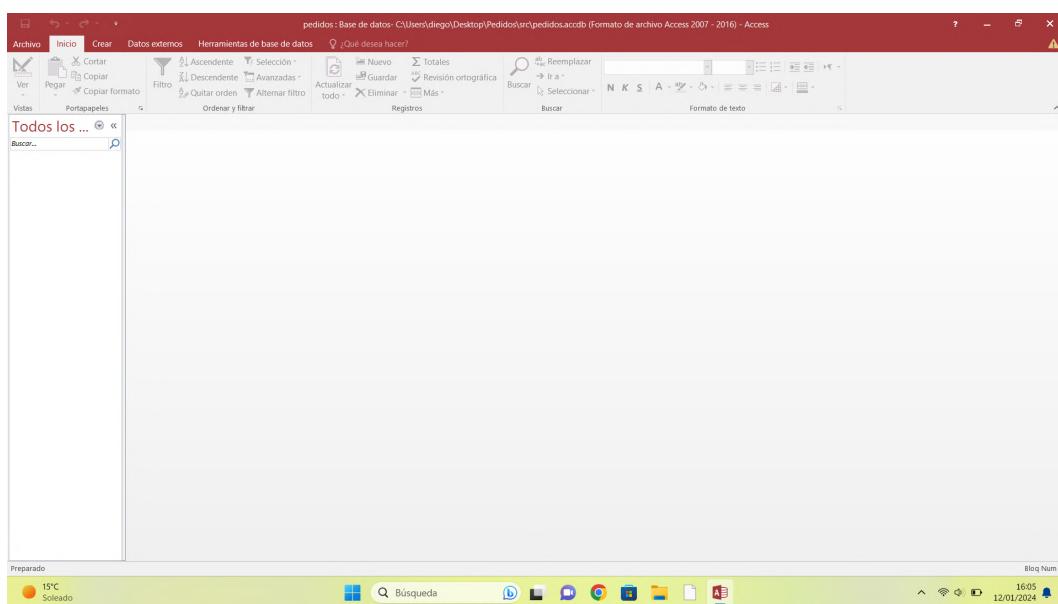
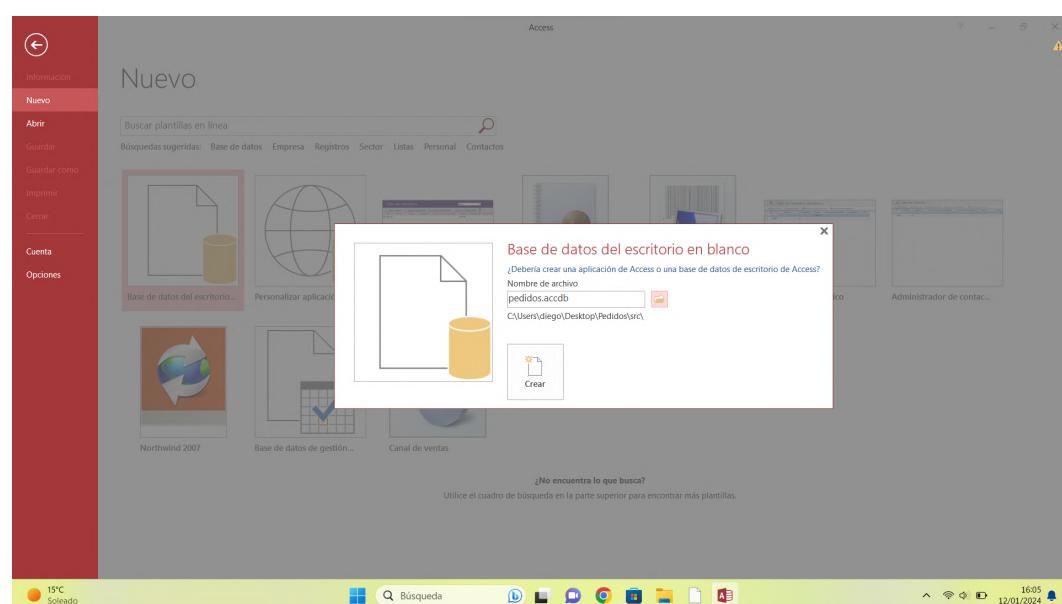
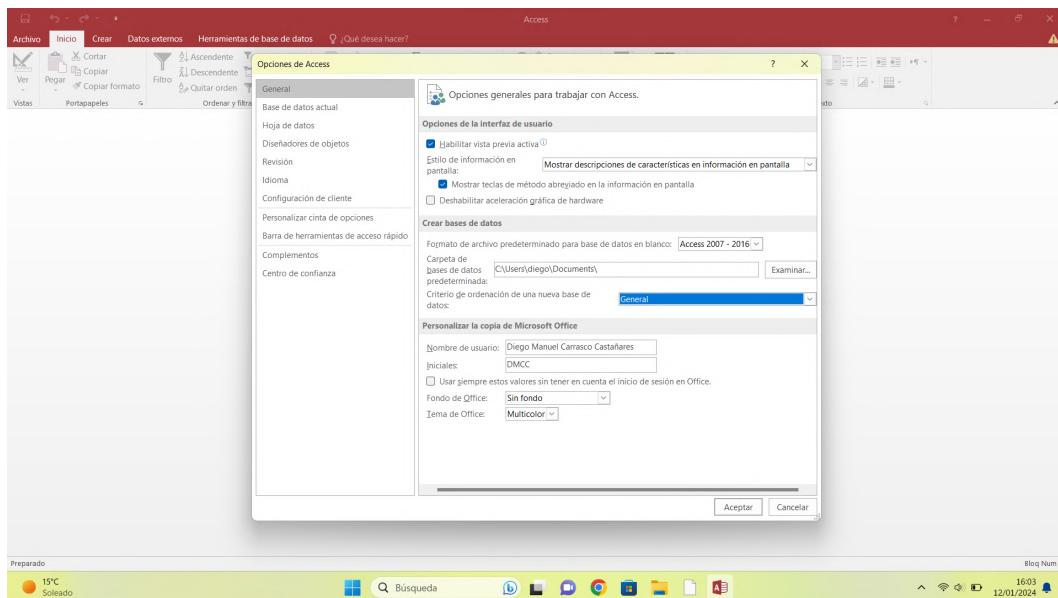
Lo primero que haremos será crear el proyecto en NetBeans, al cual llamaremos "Pedidos".



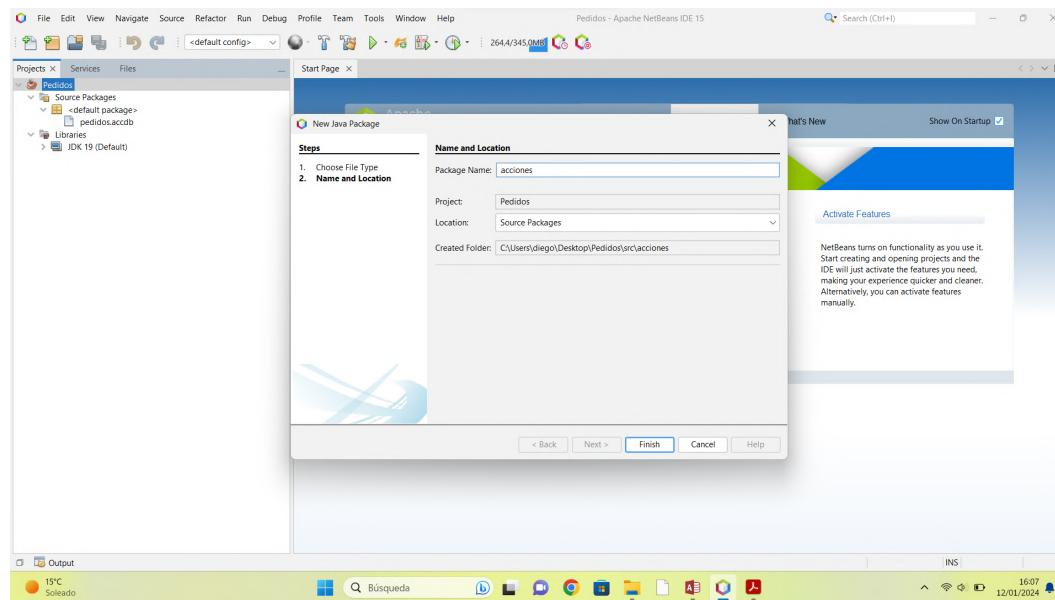
Posteriormente abriremos Access, nos dirigiremos a "ajustes - general" y en "criterios de ordenación de una nueva base de datos" seleccionaremos "General" en el desplegable, aceptando posteriormente.

Tras ello clicaremos en "Base de datos de escritorio en blanco" a la cual llamaremos "pedidos" y Access le dará la extensión ".accdb".

Buscaremos la carpeta "src" de nuestro proyecto y la seleccionaremos para guardarla en ella y clicaremos en "crear".

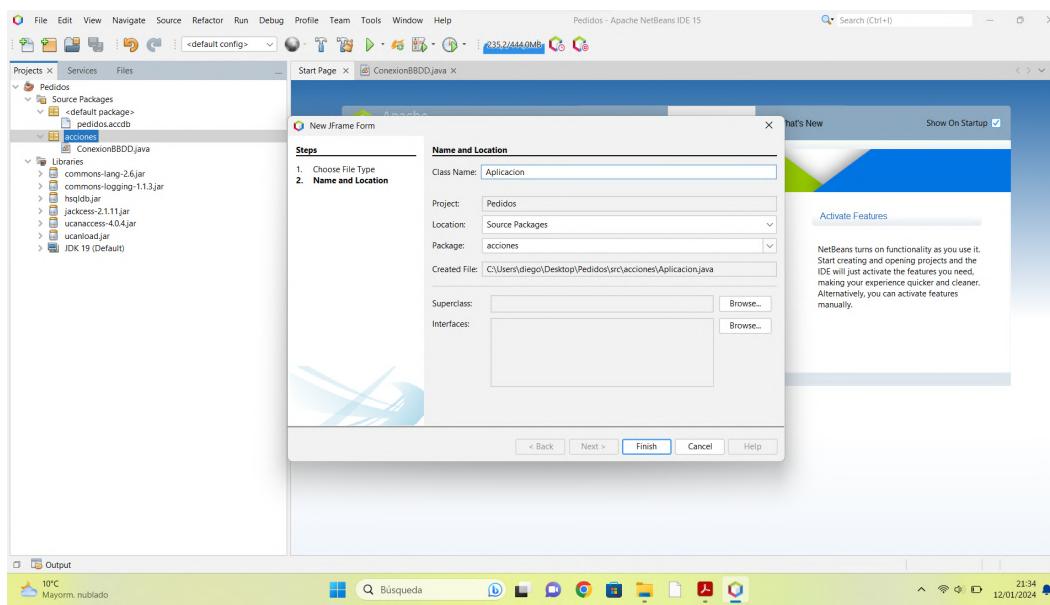


A continuación volveremos a nuestro proyecto de NetBeans (donde ya podemos apreciar como esta guardada nuestra base de datos de Access) y crearemos un nuevo paquete llamado "acciones" dentro del paquete pro defecto.



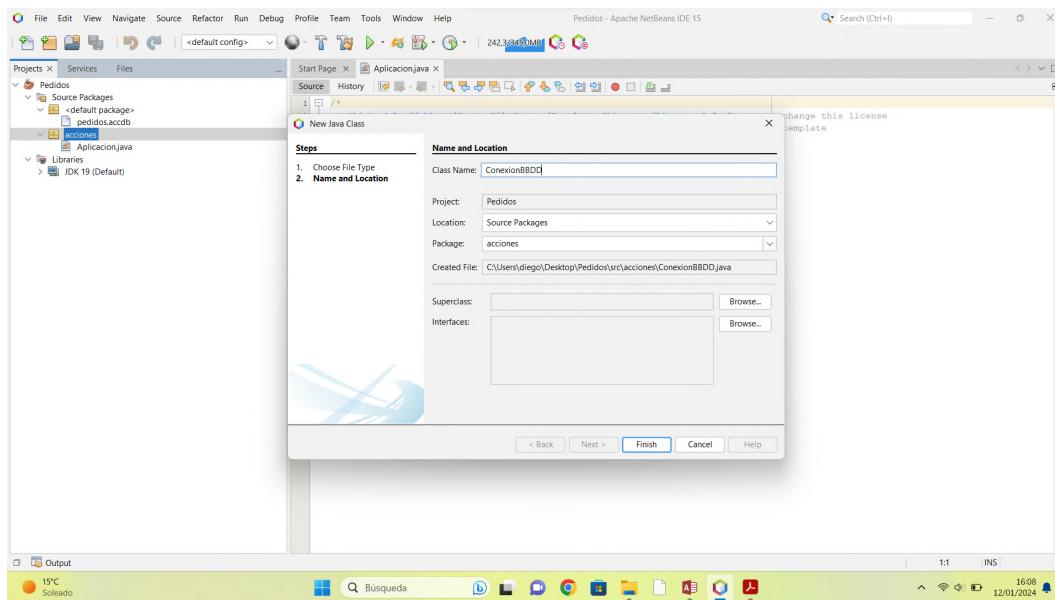
Posteriormente crearemos una nueva clase "jFrame form" llamada "Aplicación" dentro del paquete "acciones" creado anteriormente.

Esta clase será la encargada de alojar la interfaz y el código correspondiente.

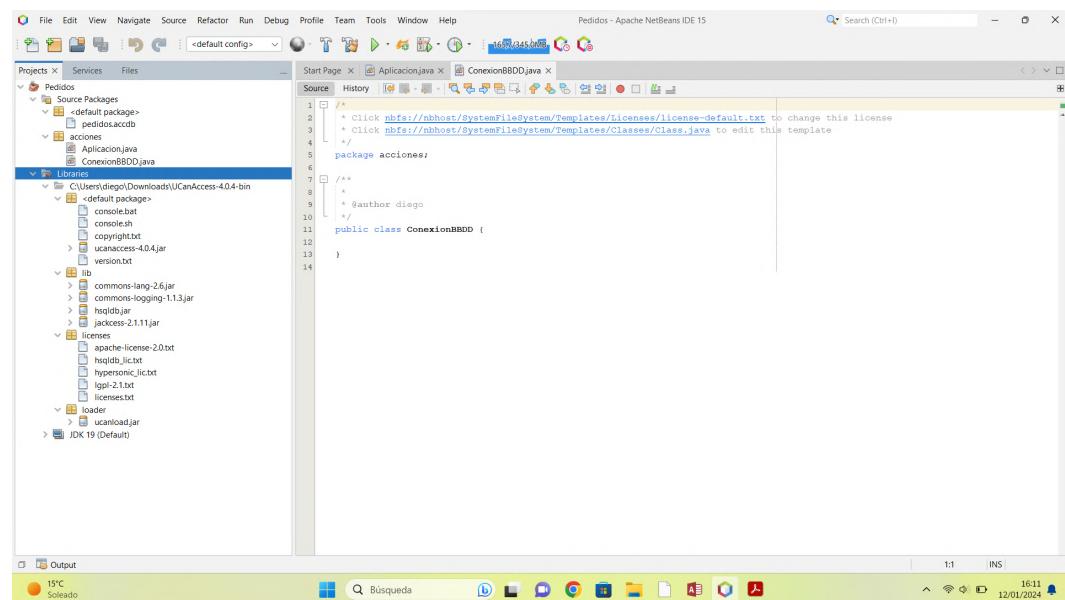


Tras ello crearemos otra clase java llamada "ConexionBBDD" dentro del paquete "acciones".

Esta clase albergará la conexión a nuestra base datos.

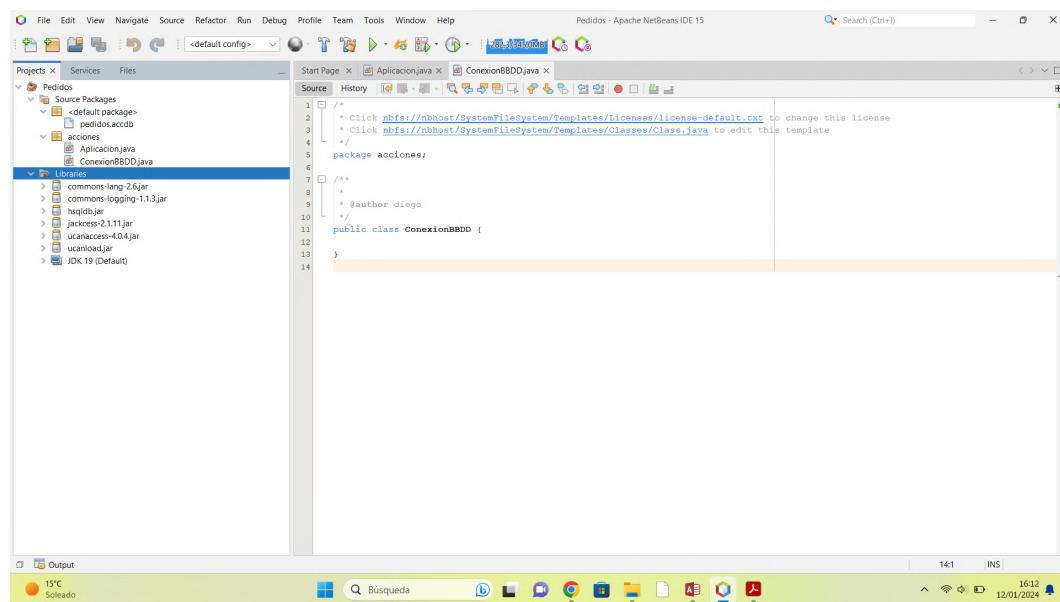


A continuación añadiremos la librería UCanAccess haciendo clic con el botón derecho en "librerías" – "Add .jar" y buscando el ".jar" de dicha librería.



Tras ello haremos lo mismo con las demás librerías necesarias tal como indica la tarea.

En la siguiente imagen podemos verlo.



Volviendo a "Aplicación" crearemos la interfaz.

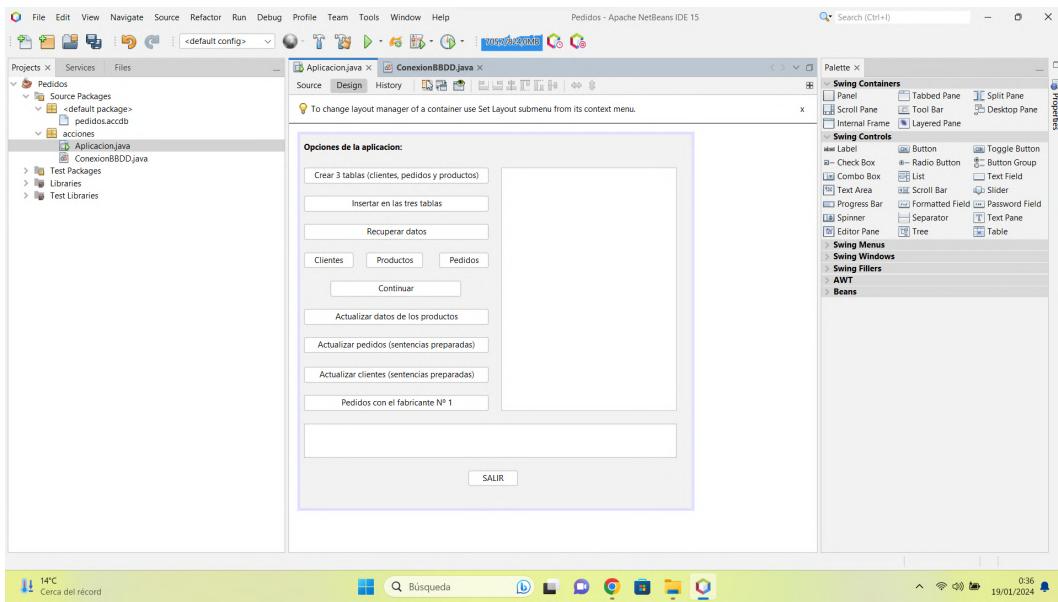
Dicha interface contendrá:

Un "label" con el texto "Opciones de la aplicación.

- Un botón que tendrá la función de crear las tres tablas. Contendrá el texto "Crear 3 tablas (clientes, pedidos y productos)".
- Un botón que tendrá la función de insertar los datos en las tablas. Contendrá el texto "Insertar en las tres tablas".
- Un botón que tendrá la función de activar los botones "Clientes, Productos y Pedidos". Contendrá el texto "Recuperar datos".
- Un botón que tendrá la función de recuperar todos los datos de la tabla clientes y mostrarlos en un text área. Contendrá el texto "Clientes".
- Un botón que tendrá la función de recuperar todos los datos de la tabla productos y mostrarlos en un text área. Contendrá el texto "Productos".
- Un botón que tendrá la función de recuperar todos los datos de la tabla pedidos y mostrarlos en un text área. Contendrá el texto "Pedidos".
- Un botón que tendrá la función de activar el botón siguiente con el texto "Continuar".
- Un botón que tendrá la función de actualizar, en la tabla productos, el precio del producto cuyo identificador es "002" al valor 2 y mostrar los datos de ese producto en el

text área. Contendrá el texto "Actualizar datos de los productos".

- Un botón que tendrá la función de actualizar, en la tabla pedidos, el importe del pedido 3 al valor 200 y mostrar los datos de ese pedido en en text área. Contendrá el texto "Actualizar pedidos (sentencias preparadas)".
- Un botón que tendrá la función de actualizar, en la tabla clientes, el nombre y el límite de crédito del cliente 2101 a los valores Charo Holgado para el nombre y 1000 para el límite de crédito en una sola sentencia. Contendrá el texto "Actualizar clientes (sentencias preparadas)".
- Un botón que tendrá la función de mostrar todos los pedidos realizados que contengan productos del fabricante con id 1 mostrando en el text área, para cada pedido, el código y número de pedido, nombre del cliente, la descripción del producto y la cantidad. Contendrá el texto "Pedidos con el fabricante nº1".
- Un text área que mostrara los resultados que necesitemos.
- Un jTextField para mostrar los mensajes que necesitemos.
- Un botón que será el encargado de cerrar la conexión con la base de datos y la aplicación. Contendrá el texto "CERRAR".



Volviendo a la clase en la clase "ConexionBBDD", vamos a crear su código.

Crearemos un objeto de clase Connection llamado "coon" y otro de clase Statement llamado "st". Ambos los instanciaremos a null.

Tras ello crearemos el método constructor de la clase que constará de un string llamado "ruta" y otro llamado "url".

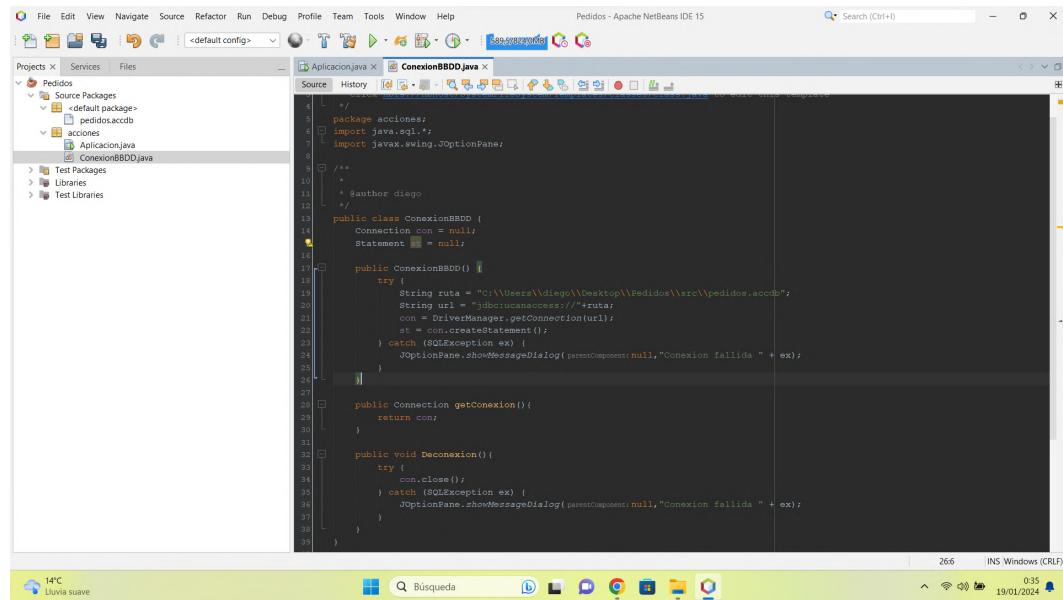
En el primero guardaremos la ruta hasta la base de datos Access y el otro almacenará en nombre del conector mas la ruta.

Instanciaremos el objeto "con", que mediante el método DriverManager.getConnection, al cual le pasaremos el objeto "url" creara la conexión.

Posteriormente instanciaremos el objeto "st", al cual le pasaremos el objeto "coon" aplicándole el método createStatement.

Todo ello envuelto en try - catch para controlar las excepciones.

Tendrá los métodos getConexion, que retornara un objeto de la clase "coon", y Desconexion, que cerrara la conexión, envuelta en una try - catch, para controlar si tuviera alguna excepción.



```

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
default config > Search (Ctrl+I)
Pedidos - Apache NetBeans IDE 15
Projects Services Files
Pedidos
  Source Packages
    default package
      pedidos.accdt
    acciones
      Conexion.java
      ConexionBDD.java
Test Packages
Libraries
Test Libraries
Aplicacion.java X ConexionBDD.java
Source History ...
/*
 * @author diego
 */
public class ConexionBDD {
    Connection con = null;
    Statement st = null;
    public ConexionBDD() {
        try {
            String ruta = "C:\\Users\\diego\\Desktop\\Pedidos\\src\\pedidos.accdt";
            String url = "jdbc:access://" + ruta;
            con = DriverManager.getConnection(url);
            st = con.createStatement();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Conexion fallida " + ex);
        }
    }
    public Connection getConnection() {
        return con;
    }
    public void Desconexion() {
        try {
            con.close();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Conexion fallida " + ex);
        }
    }
}

```

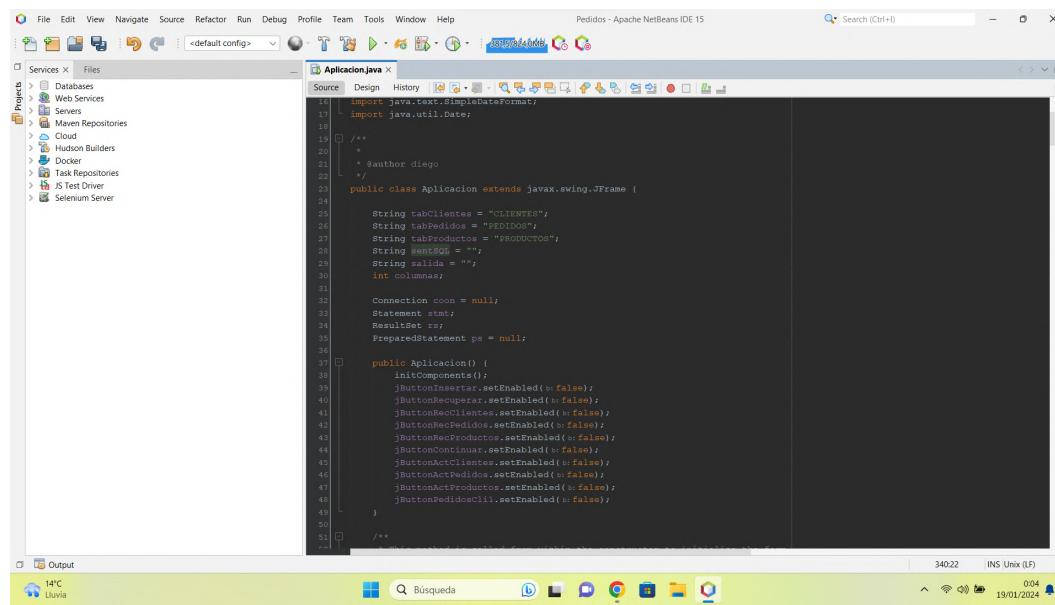
La clase aplicación dispondrá varios atributos tipo string llamados "tabClientes", que contendrá la cadena "CLIENTES", otro llamado "tabPedidos" que contendrá la cadena "PEDIDOS", otro llamado "tabProductos" que serán los nombres que tienen las tablas correspondientes en la base de datos.

Los otros atributos de tipo string que tendrán serán "sentSQL" y se encargará de guardar cada sentencia que queramos enviar a la BBDD y otro llamado "salida" que será el encargado de almacenar los datos recogidos de las tablas de la BBDD y mostrar posteriormente, en el test área dicha información.

También de un atributo tipo int llamado "columnas", que recogerá el número de columnas de cada tabla.

Dispondrá de un objeto tipo Connection llamado "coon", uno de tipo Statement llamado "stmt", uno de tipo ResultSet llamado "rs" y uno de tipo PreparedStatement llamado "ps".

Posteriormente, en el método constructor de la clase, declararemos, mediante el método setEnable, deshabilitados todos los botones a excepción del primero (Crear 3 tablas) y el botón de salir.



```

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Pedidos - Apache NetBeans IDE 15
Search (Ctrl+F)
Projects Services Files
Aplicacion.java
Source Design History
import java.text.SimpleDateFormat;
import java.util.Date;
/*
 * @author diego
 */
public class Aplicacion extends javax.swing.JFrame {
    ...
    String tablClientes = "CLIENTES";
    String tablPedidos = "PEDIDOS";
    String tablproductos = "PRODUCTOS";
    String sentSQL = "";
    String salida = "";
    int columnas;
    Connection coon = null;
    Statement stmt;
    ResultSet rs;
    PreparedStatement ps = null;
    ...
    public Aplicacion() {
        initComponents();
        jButtonInsertar.setEnabled(false);
        jButtonRecuperar.setEnabled(false);
        jButtonRecClientes.setEnabled(false);
        jButtonRecPedidos.setEnabled(false);
        jButtonActualizar.setEnabled(false);
        jButtonContinuar.setEnabled(false);
        jButtonActClientes.setEnabled(false);
        jButtonActPedidos.setEnabled(false);
        jButtonActProductos.setEnabled(false);
        jButtonPedidosCli.setEnabled(false);
    }
    ...
}

```

El botón de crear las tablas constará de un objeto "coon" que creará una nueva conexión a la base de datos a través del método getConexion., al cual le aplicamos el método toString se encargará de crear las tablas en la base de daos.

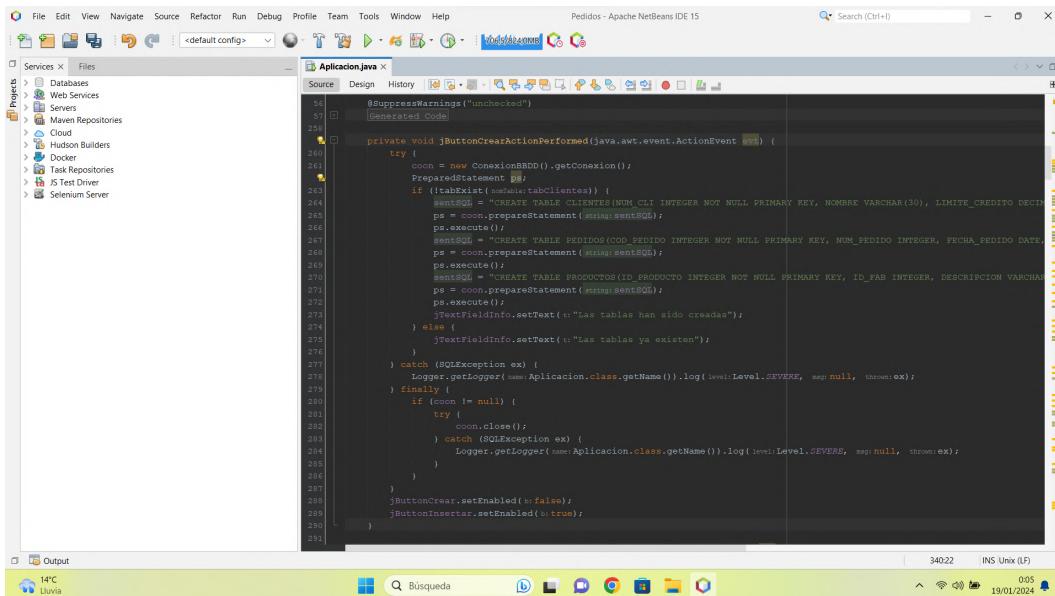
Posteriormente crearemos las sentencias y las guardaremos en el atributo creado para tal fin.

Esto último lo haremos mediante el "prepareStatement" de la clase "Conexión".

Todo el código esta envuelto en un try - catch - finally.

En el try estará envuelto todo el código, en el catch se controlarán las excepciones más habituales y en el finally se encargará, pase lo que pase en el código anterior, de cerrar la conexión a la base de datos mediante el método close.

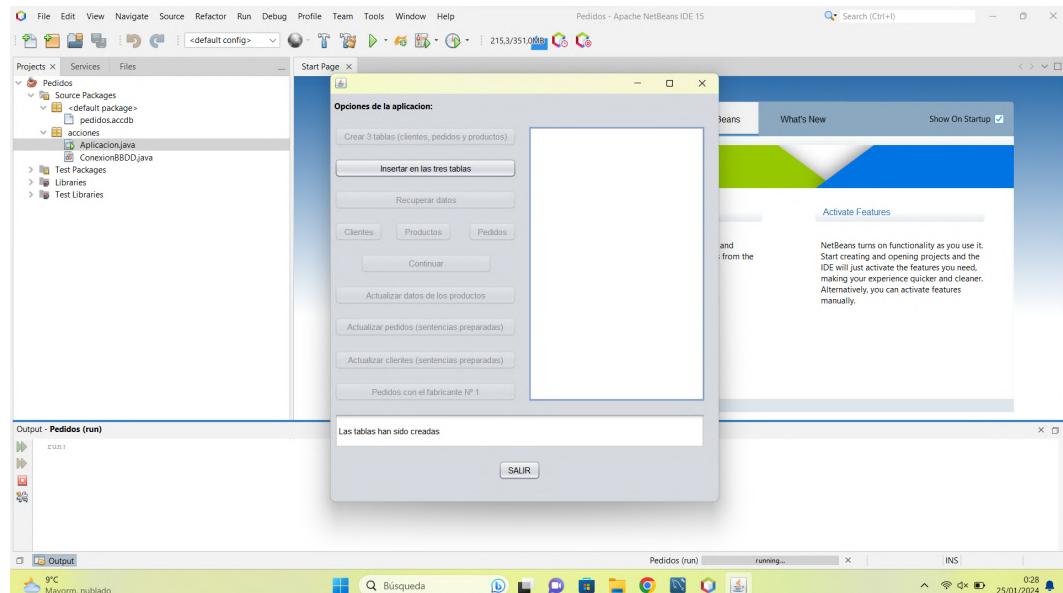
En la siguiente imagen podemos verlo.



```

private void jButtonCrearActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        con = new ConexionBD().getConexion();
        PreparedStatement ps;
        if (!tbExist("clientes")) {
            sentSQL = "CREATE TABLE CLIENTES(NUM_CLI INTEGER NOT NULL PRIMARY KEY, NOMBRE VARCHAR(30), LIMITE_CREDITO DECIMAL(10,2))";
            ps = con.prepareStatement(sentSQL);
            ps.execute();
            sentSQL = "CREATE TABLE PEDIDOS(COD_PEDIDO INTEGER NOT NULL PRIMARY KEY, NUM_PEDIDO INTEGER, FECHA_PEDIDO DATE, ID_FABRICANTE INTEGER, DESCRIPCION VARCHAR(100))";
            ps = con.prepareStatement(sentSQL);
            ps.execute();
            sentSQL = "CREATE TABLE PRODUCTOS(ID_PRODUCTO INTEGER NOT NULL PRIMARY KEY, ID_FAB INTEGER, DESCRIPCION VARCHAR(100))";
            ps = con.prepareStatement(sentSQL);
            ps.execute();
        } else {
            jTextFieldInfo.setText("Las tablas ya existen");
        }
    } catch (SQLException ex) {
        Logger.getLogger(Aplicacion.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        if (con != null) {
            try {
                con.close();
            } catch (SQLException ex) {
                Logger.getLogger(Aplicacion.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}

```



El botón “Insertar”, al hacer clic sobre el, establecerá una conexión con la base de datos e instanciará un statement.

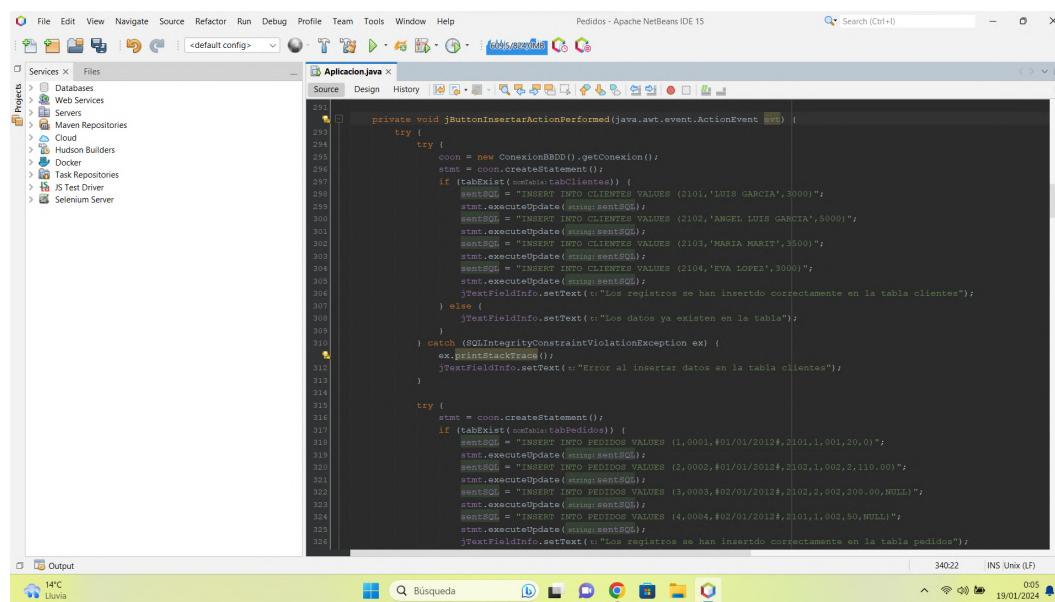
Posteriormente controlará si la tabla existe mediante un if, y si dicha tabla existe insertará los datos en ella.

En caso de existir los datos, como el código anterior está en un try - catch, mostrará un mensaje en el jtext creado para tal fin.

Estos pasos los seguiremos igual para inserción de los datos en las tres tablas.

Todo ello estará envuelto en un try - catch - finally, en el cual se controlará la excepción de que los datos ya existan en la tabla y cerrará la conexión y el statement en el finally.

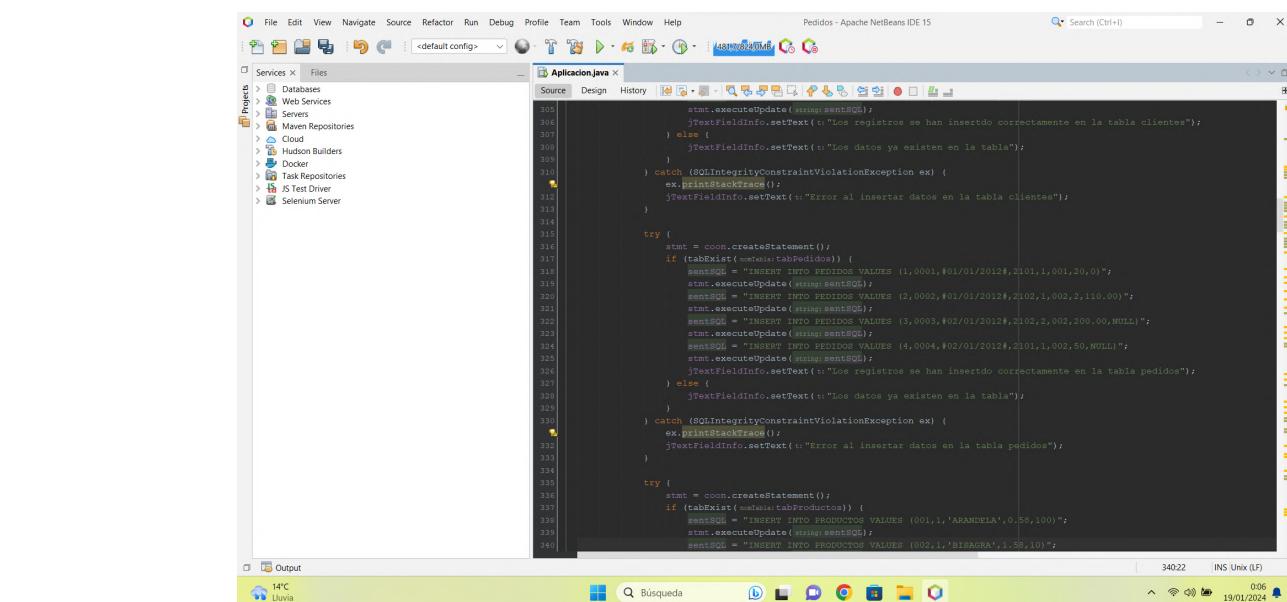
Por último, este botón, se encargará de desactivarse a sí mismo y de activar al siguiente.



```

private void jButtonInsertarActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        try {
            coon = new ConexionBD().getConexion();
            stmt = coon.createStatement();
            if (!tabExist(tabName:tabClientes)) {
                sentSQL = "INSERT INTO CLIENTES VALUES (2101,'LOUIS GARCIA',3000)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO CLIENTES VALUES (2102,'ANGEL LUIS GARCIA',5000)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO CLIENTES VALUES (2103,'MARIA MARITZ',3500)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO CLIENTES VALUES (2104,'EVA LOPEZ',3000)";
                stmt.executeUpdate(string:sentSQL);
                jTextFieldInfo.setText("Los registros se han insertado correctamente en la tabla clientes");
            } else {
                jTextFieldInfo.setText("Los datos ya existen en la tabla");
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            jTextFieldInfo.setText("Error al insertar datos en la tabla clientes");
        }
    } catch (SQLException ex) {
        try {
            stmt = coon.createStatement();
            if (!tabExist(tabName:tabPedidos)) {
                sentSQL = "INSERT INTO PEDIDOS VALUES (1,0001,#01/01/2012#,2101,1,001,20,0)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PEDIDOS VALUES (2,0002,#01/01/2012#,2102,1,002,2,110.00)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PEDIDOS VALUES (3,0003,#02/01/2012#,2102,2,002,200.00,NULL)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PEDIDOS VALUES (4,0004,#02/01/2012#,2101,1,002,50,NULL)";
                stmt.executeUpdate(string:sentSQL);
                jTextFieldInfo.setText("Los registros se han insertado correctamente en la tabla pedidos");
            } else {
                jTextFieldInfo.setText("Los datos ya existen en la tabla");
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            jTextFieldInfo.setText("Error al insertar datos en la tabla pedidos");
        }
    }
}

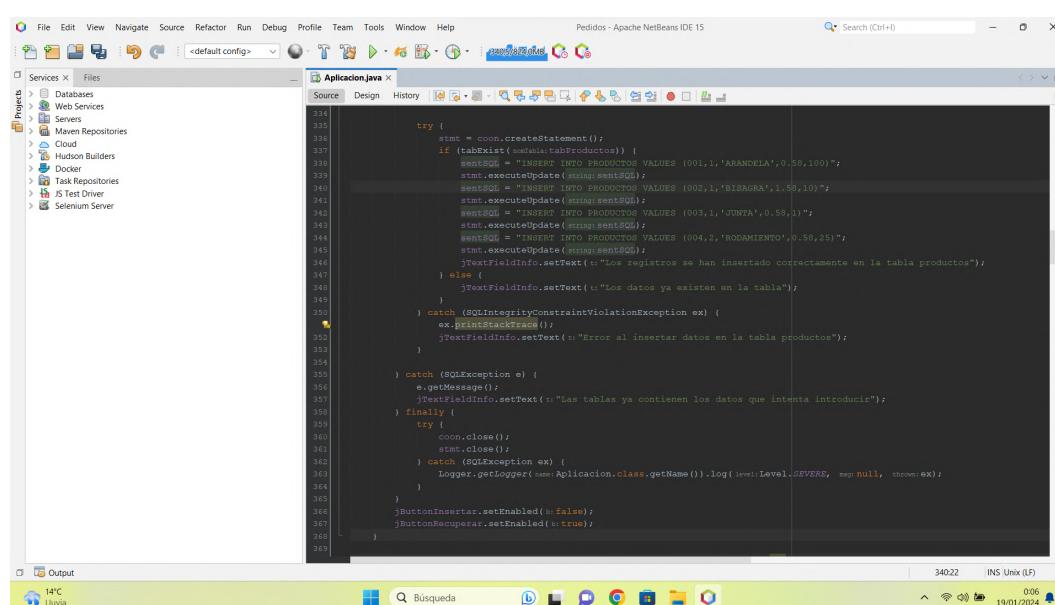
```



```

private void jButtonInsertarActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        try {
            coon = new ConexionBD().getConexion();
            stmt = coon.createStatement();
            if (!tabExist(tabName:tabPedidos)) {
                sentSQL = "INSERT INTO PEDIDOS VALUES (1,0001,#01/01/2012#,2101,1,001,20,0)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PEDIDOS VALUES (2,0002,#01/01/2012#,2102,1,002,2,110.00)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PEDIDOS VALUES (3,0003,#02/01/2012#,2102,2,002,200.00,NULL)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PEDIDOS VALUES (4,0004,#02/01/2012#,2101,1,002,50,NULL)";
                stmt.executeUpdate(string:sentSQL);
                jTextFieldInfo.setText("Los registros se han insertado correctamente en la tabla pedidos");
            } else {
                jTextFieldInfo.setText("Los datos ya existen en la tabla");
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            jTextFieldInfo.setText("Error al insertar datos en la tabla pedidos");
        }
    }
}

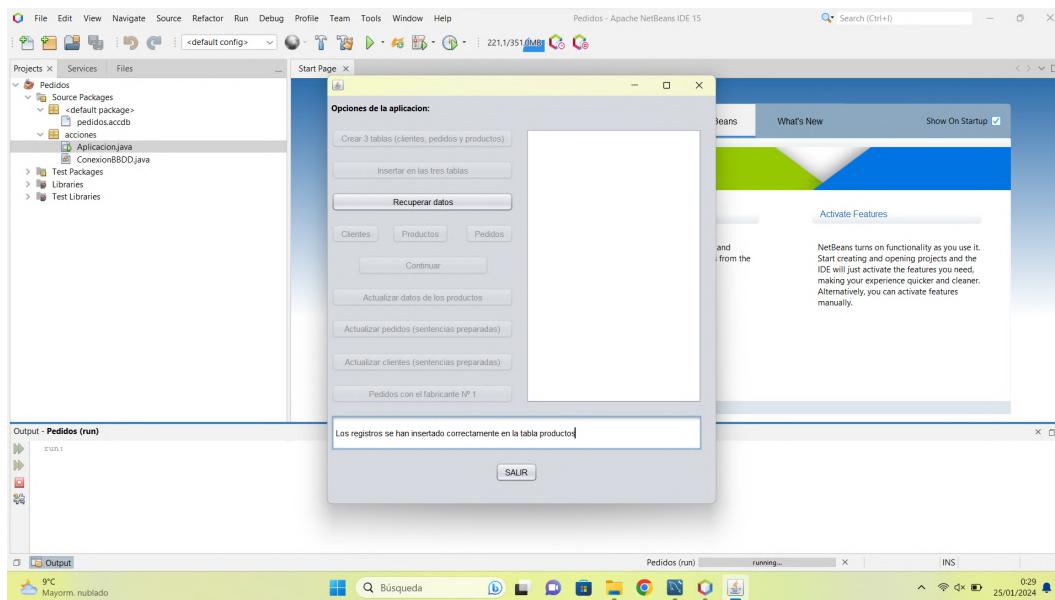
```



```

private void jButtonInsertarActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        try {
            coon = new ConexionBD().getConexion();
            stmt = coon.createStatement();
            if (!tabExist(tabName:tabProductos)) {
                sentSQL = "INSERT INTO PRODUCTOS VALUES (001,1,'ARANDELA',0.56,100)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PRODUCTOS VALUES (002,1,'BISNAGRA',1.56,10)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PRODUCTOS VALUES (003,1,'JUNTA',0.88,1)";
                stmt.executeUpdate(string:sentSQL);
                sentSQL = "INSERT INTO PRODUCTOS VALUES (004,2,'RCOMIENTO',0.58,25)";
                stmt.executeUpdate(string:sentSQL);
                jTextFieldInfo.setText("Los registros se han insertado correctamente en la tabla productos");
            } else {
                jTextFieldInfo.setText("Los datos ya existen en la tabla");
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            jTextFieldInfo.setText("Error al insertar datos en la tabla productos");
        }
    }
}

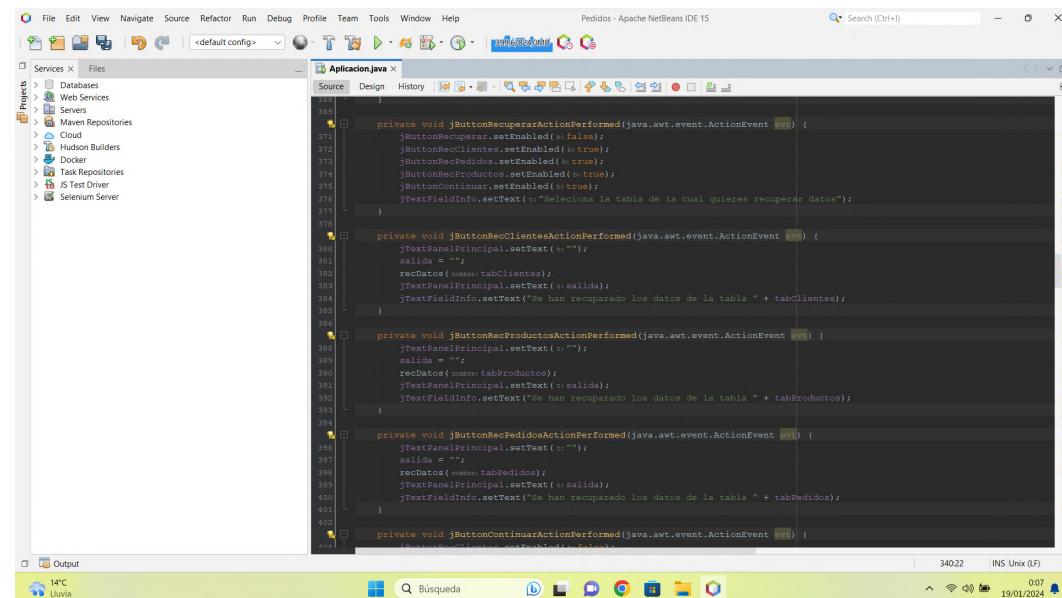
```

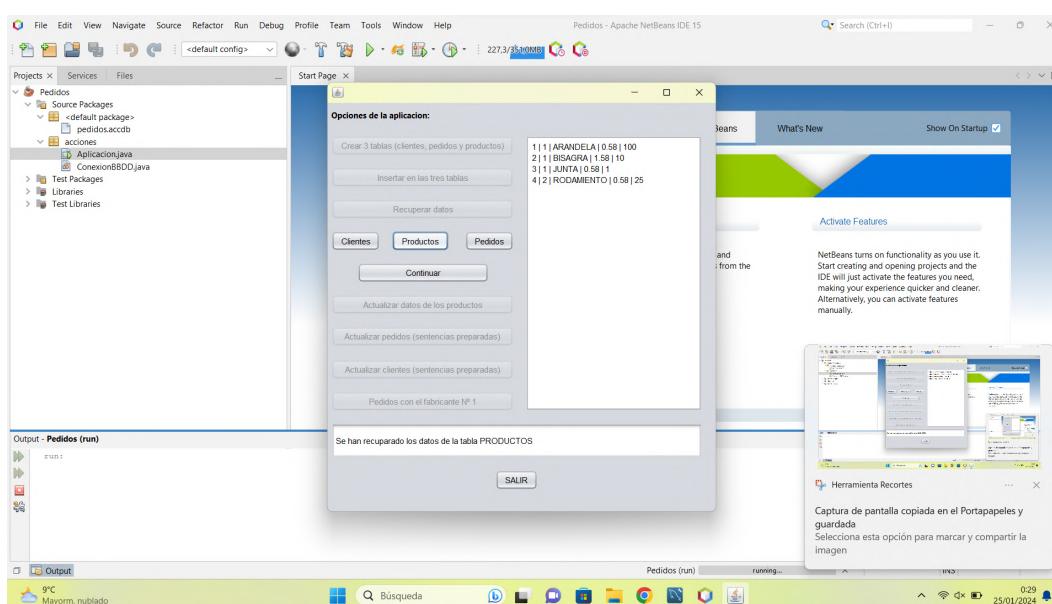
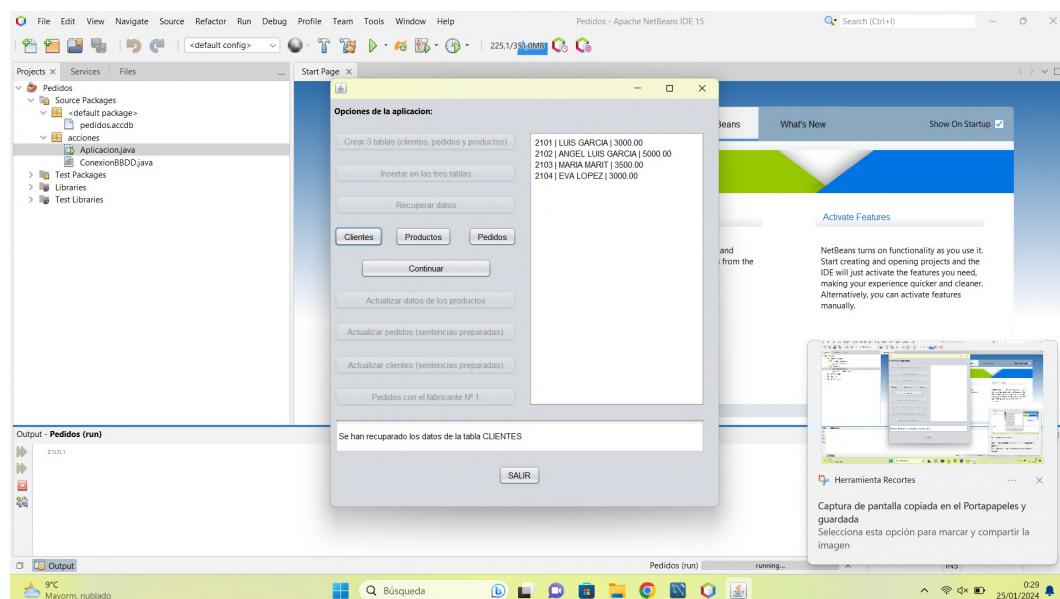
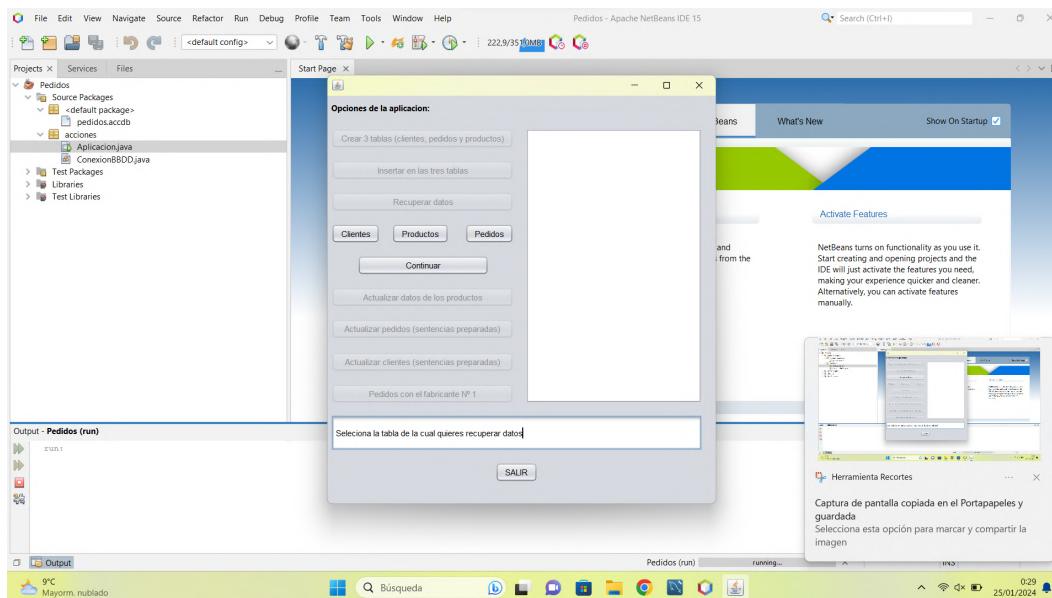


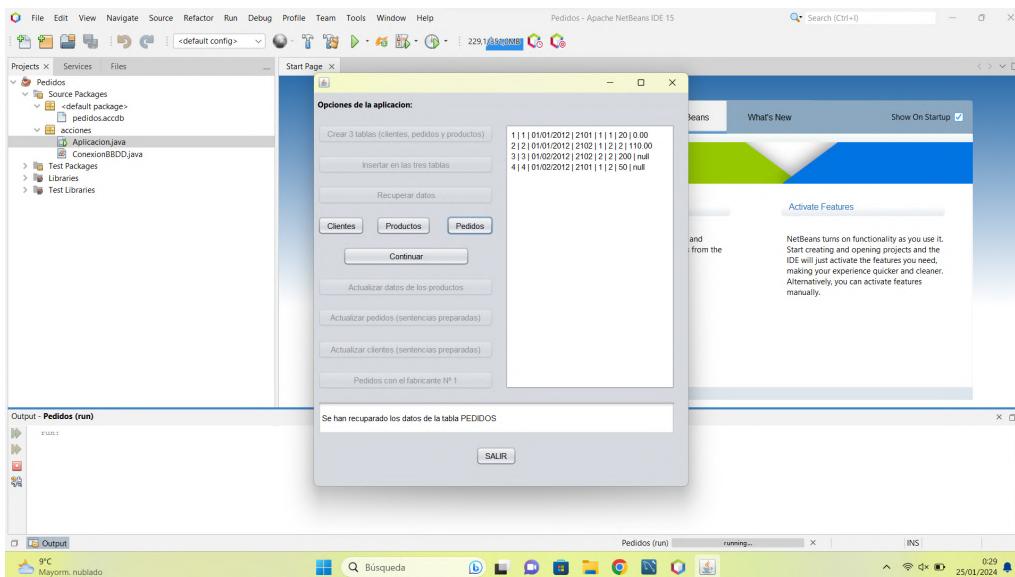
El botón recuperar, el siguiente en activarse, solo se encargará de desactivarse a si mismo y activar los botones que dispondrán los métodos de recuperar los datos de tablas correspondientes y activar el botón continuar.

El botón que se encarga de recuperar los datos de la tabla clientes se encargará de vaciar el jtextpanel, vaciar la variable donde almacenamos los datos recogidos de la tabla, llamar al método "recDatos" al cual le pasaremos el nombre de la tabla clientes, mostrar en el panel los datos recogidos de la tabla clientes y mostrar en el jtext el mensaje correspondiente.

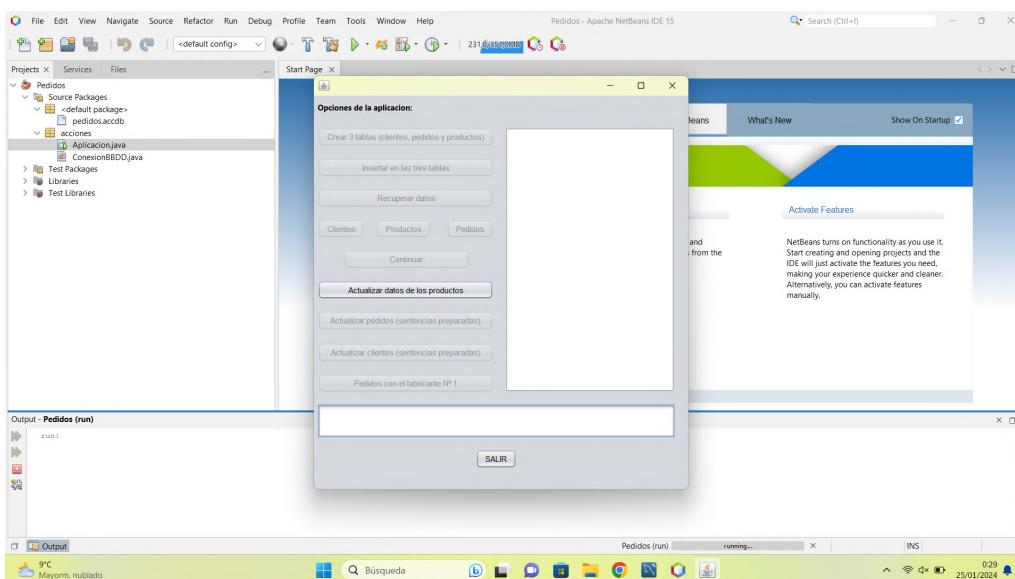
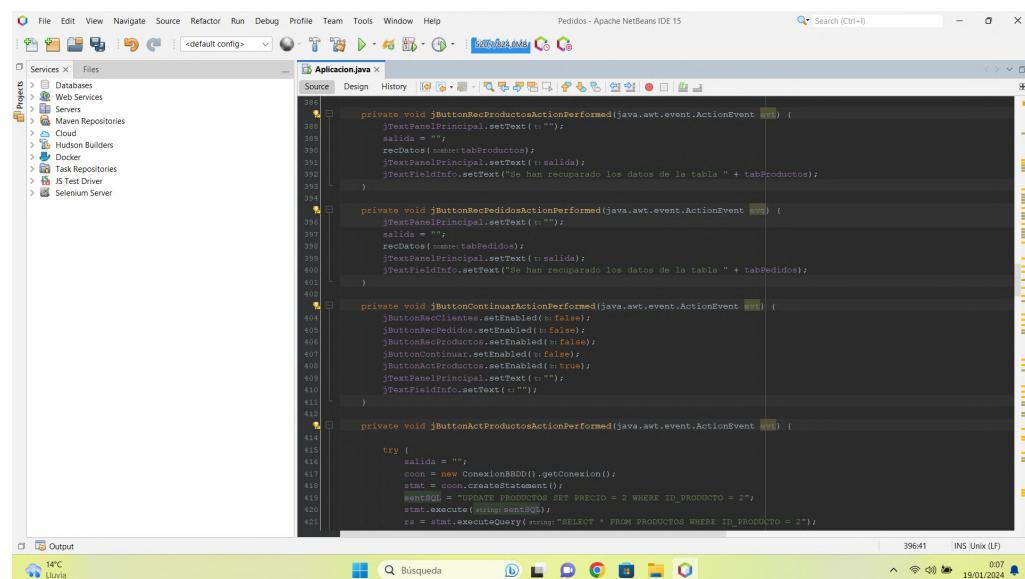
Los métodos de recuperar productos y pedidos harán lo mismo que el método anterior pero pasándole el nombre de la tabla correspondiente.







El botón continuar será el encargado de deshabilitar los botones "recClientes", "recPedidos", "recProductos", "continuar", habilitar el de "actProductos", y vaciar el textPanel y el textField.



El botón de "actProductos" tendrá las siguientes funciones.

Vaciara la variable "salida", establecerá una nueva conexión con la base de datos a través le la clase "ConexionBBDD" y su método "getConexion" y creará un nuevo statement pasándole la conexión a través del método "createStatement".

Posteriormente instanciaremos en la variable sentSQL la sentencia para actualizar a 2 el precio del producto con "id_producto" igual a 2.

A continuación, ejecutaremos la sentencia pasándosela al statement mediante el método execute.

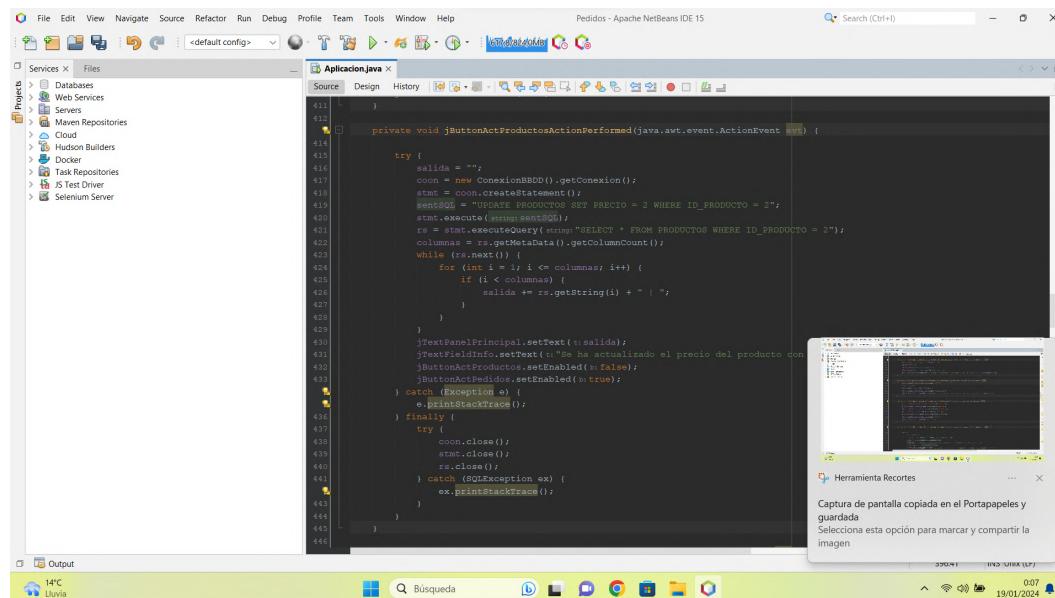
Tras ello, mediante el resultSet pasándole el statement al método executeQuery, y extraeremos el número de columnas de la tabla mediante el resultSet y sus métodos getMetaData y getColumnCount y lo guardaremos en la variable "columnas".

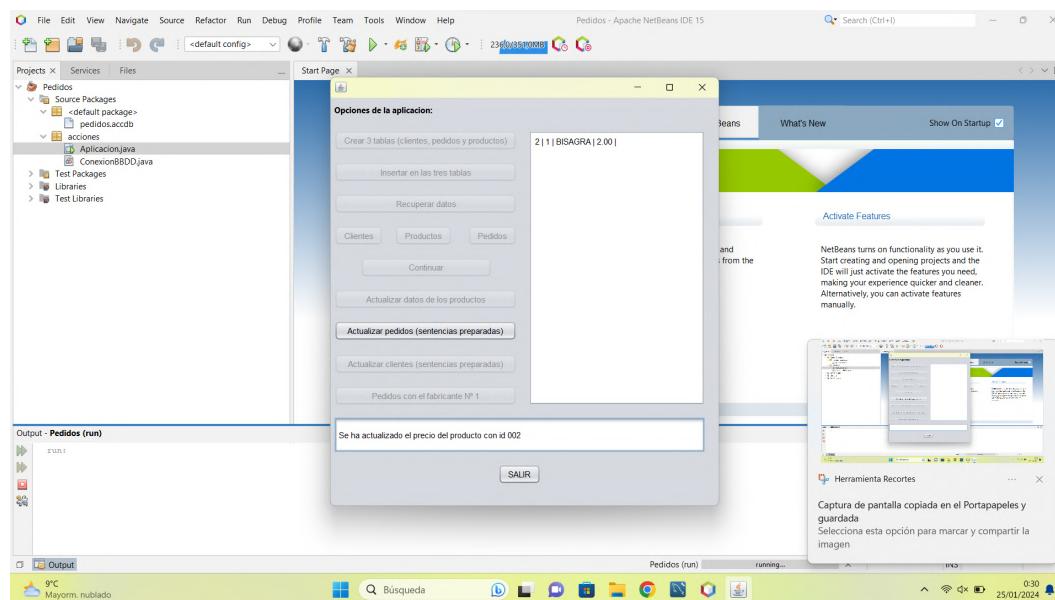
En un while recuperaremos cada columna y la guardaremos en la variable salida.

Posteriormente mostraremos el contenido de dicha variable en el textPanel y mostraremos un mensaje en el textField.

Por último deshabilitaremos el botón de este método y activaremos el siguiente botón que será "actPedidos".

Todo ello estará dentro de un try - catch que capturara las excepciones y un finally que cerrara la conexión, el statement y el "resultset".





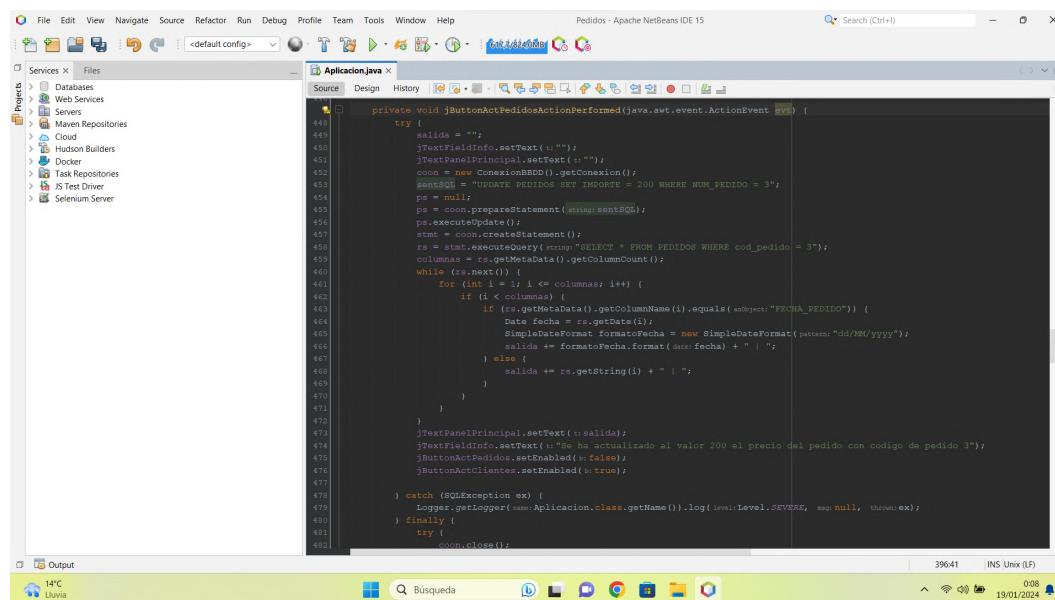
El botón "ActPedidos" tendrá los siguientes métodos.

Lo primero que hará será vaciar la variable salida y los textField y textPanel, crearemos una nueva conexión con la base de datos a través del objeto coon, instanciaremos en la variable "sentsQL" la sentencia que utilizaremos para actualizar a 200 el importe del pedido con numero de pedido 3, instanciaremos a null el preparedStatement para, posteriormente, pasarle la sentencia sql, y la ejecutaremos mediante el método executeUpdate.

A continuación instanciaremos nuevo statement, un resulset al cual le pasaremos la sentencia para recuperar de la base de datos los datos del pedido que hemos actualizado, instanciaremos en la variable columnas el número de columnas del resultado, y recorreremos, mediante un while, cada columna, guardando en la variable salida todo lo recogido mediante dicho while, dándole formato a la fecha en el caso de la columna fecha mediante otro while.

Tras ello mostraremos en el textPanel el contenido almacenado en la variable salida, el mensaje correspondiente en el textField, deshabilitaremos el botón "actPedidos" y habilitaremos el botón "actClientes".

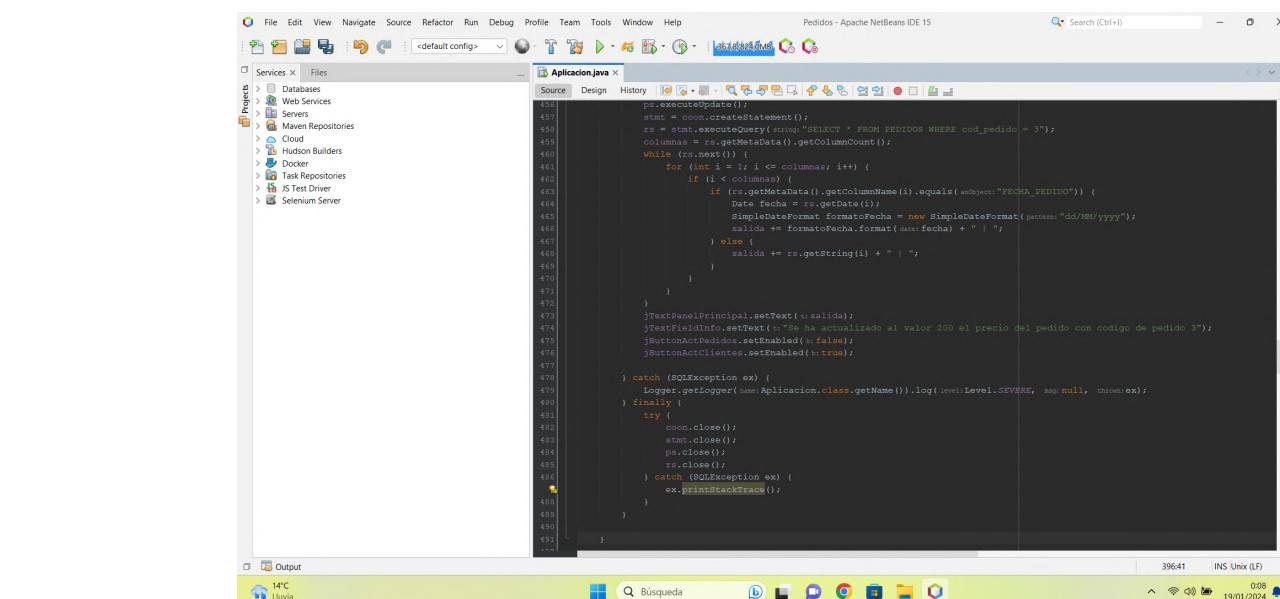
Todo este código estará envuelto en un try - catch - finally para capturar las excepciones que se puedan producir y cerrar, en el finally, la conexión, el statement, el preparedStatement y el resulSet.



```

private void jButtonActPedidosActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        salida = "";
        jTextFieldInfo.setText("");
        jPanelPrincipal.setText("");
        coon = new ConexionBDD().getConexion();
        sentSQL = "UPDATE PEDIDOS SET IMPORTE = 200 WHERE NUM_PEDIDO = 3";
        ps = coon.prepareStatement(sentSQL);
        ps.executeUpdate();
        stmt = coon.createStatement();
        rs = stmt.executeQuery("SELECT * FROM PEDIDOS WHERE cod_pedido = 3");
        columnas = rs.getMetaData().getColumnCount();
        while (rs.next()) {
            for (int i = 1; i < columnas; i++) {
                if (i < columnas)
                    if (rs.getMetaData().getColumnName(i).equals("objeto: FECHA_PEDIDO")) {
                        Date fecha = rs.getDate(i);
                        SimpleDateFormat formatoFecha = new SimpleDateFormat(pattern:"dd/MM/yyyy");
                        salida += formatoFecha.format(date:fecha) + " | ";
                    } else {
                        salida += rs.getString(i) + " | ";
                    }
                }
            }
        jTextFieldPrincipal.setText(salida);
        jTextFieldInfo.setText("Se ha actualizado al valor 200 el precio del pedido con codigo de pedido 3");
        jButtonActPedidos.setEnabled(false);
        jButtonActClientes.setEnabled(true);
    } catch (SQLException ex) {
        Logger.getLogger(name:Aplicacion.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            coon.close();
        }
    }
}

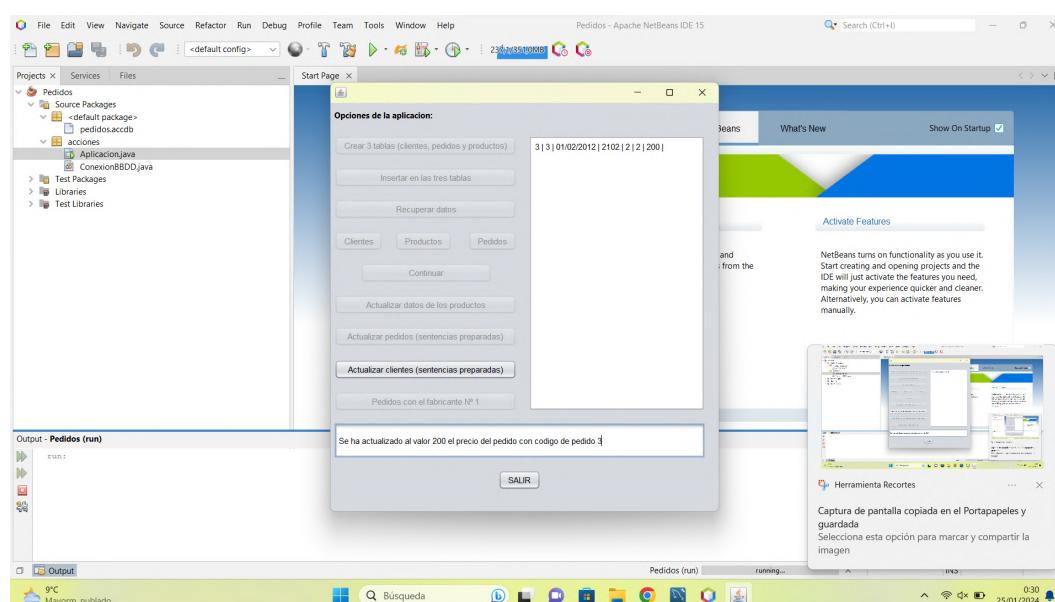
```



```

private void jButtonActPedidosActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        ps.executeUpdate();
        stmt = coon.createStatement();
        rs = stmt.executeQuery("SELECT * FROM PEDIDOS WHERE cod_pedido = 3");
        columnas = rs.getMetaData().getColumnCount();
        while (rs.next()) {
            for (int i = 1; i < columnas; i++) {
                if (i < columnas)
                    if (rs.getMetaData().getColumnName(i).equals("objeto: FECHA_PEDIDO")) {
                        Date fecha = rs.getDate(i);
                        SimpleDateFormat formatoFecha = new SimpleDateFormat(pattern:"dd/MM/yyyy");
                        salida += formatoFecha.format(date:fecha) + " | ";
                    } else {
                        salida += rs.getString(i) + " | ";
                    }
                }
            }
        jTextFieldPrincipal.setText(salida);
        jTextFieldInfo.setText("Se ha actualizado al valor 200 el precio del pedido con codigo de pedido 3");
        jButtonActPedidos.setEnabled(false);
        jButtonActClientes.setEnabled(true);
    } catch (SQLException ex) {
        Logger.getLogger(name:Aplicacion.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            coon.close();
            stmt.close();
            ps.close();
            rs.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}

```



Output - Pedidos (run)

```

run:

```

Se ha actualizado al valor 200 el precio del pedido con codigo de pedido 3

SALIR

9°C Mayorm, nublado

El botón "actClientes" será el encargado de actualizar el cliente 2101, cambiándole el nombre y el límite de crédito.

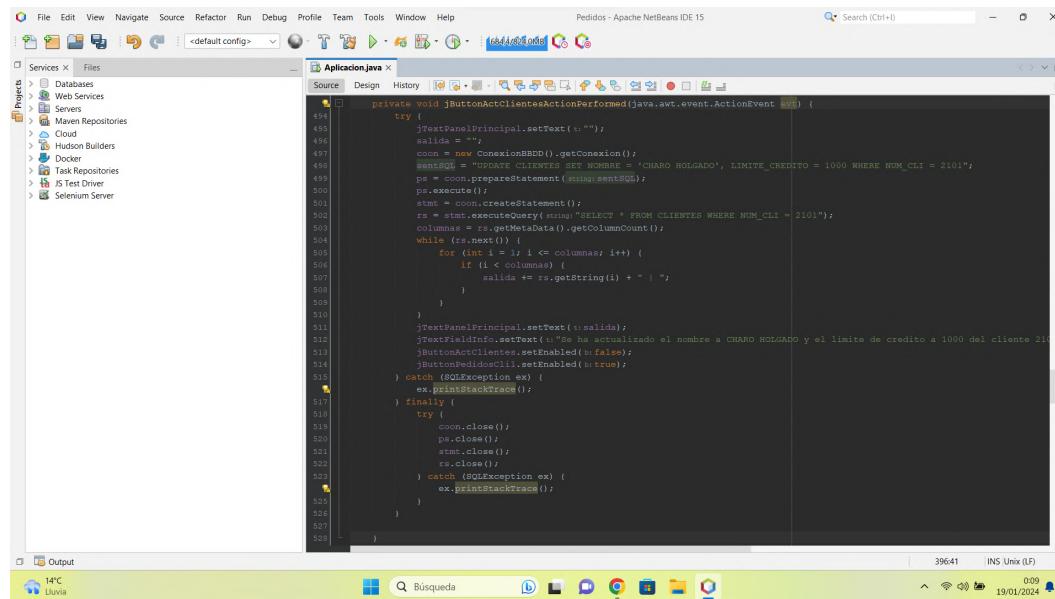
Para ello vaciaremos el JPanel y la variable salida, instanciaremos una nueva conexión a la base de datos, instanciaremos en la variable sentSQL la sentencia para actualizar dicho cliente, le pasaremos dicha sentencia al preparedStatement, lo ejecutaremos e instanciaremos un nuevo statement .

Tras ello instanciaremos un nuevo resultSet con la sentencia para recuperar de la base de datos el cliente que hemos actualizado, obtendremos el número de columnas, mediante un while recorreremos el resultSet y guardaremos en la variable salida el resultado de cada columna.

A continuación mostraremos en el JPanel el contenido almacenado en la variable salida, mostraremos en el textField el mensaje correspondiente, desactivaremos el botón "actClientes" y activaremos el siguiente botón, "pedidoCli1", en nuestro caso.

Todo este código estará envuelto en un try - catch - finally para capturar las excepciones y cerrar, en el finally, la conexión, el preparedStatement, el statement y el resultSet.

En la siguiente imagen podemos verlo.



The screenshot shows the Apache NetBeans IDE interface with the following details:

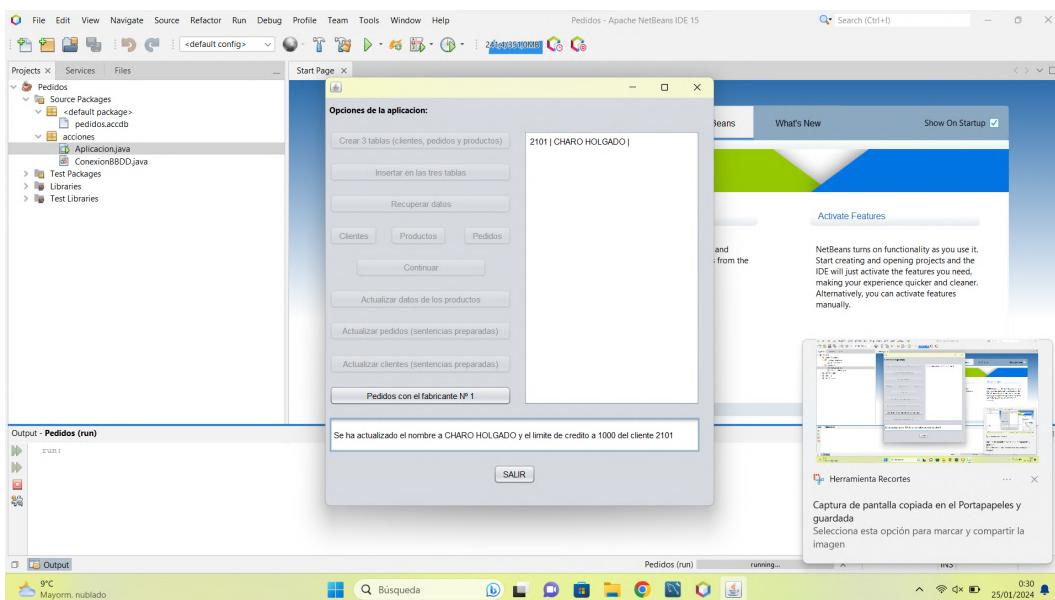
- Menu Bar:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help.
- Title Bar:** Pedidos - Apache NetBeans IDE 15.
- Toolbar:** Standard Java development tools.
- Left Sidebar:** Services X Files, Projects (with Databases, Web Services, Servers, etc.), Cloud, Hudson Builders, Docker, Task Repositories, JS Test Driver, Selenium Server.
- Central Editor:** Aplicacion.java X. The code is as follows:

```

private void jButtonActClientesActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        jTextFieldPrincipal.setText("");
        salida = "";
        coon = new ConexionBDD().getConexion();
        sentSQL = "UPDATE CLIENTES SET NOMBRE = 'CHARO HOLGADO', LIMITE_CREDITO = 1000 WHERE NUM_CLI = 2101";
        ps = coon.prepareStatement(sentSQL);
        rs = ps.executeQuery();
        stat = coon.createStatement();
        rs = stat.executeQuery("SELECT * FROM CLIENTES WHERE NUM_CLI = 2101");
        columnas = rs.getMetaData().getColumnCount();
        while (rs.next()) {
            for (int i = 1; i <= columnas; i++) {
                if (i < columnas) {
                    salida += rs.getString(i) + " | ";
                }
            }
        }
        jTextFieldPrincipal.setText(salida);
        jTextFieldInfo.setText("Se ha actualizado el nombre a CHARO HOLGADO y el límite de crédito a 1000 del cliente 2101");
        jButtonActClientes.setEnabled(false);
        jButtonPedidoCli1.setEnabled(true);
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally {
        try {
            coon.close();
            ps.close();
            stat.close();
            rs.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}

```

- Bottom Status Bar:** 396:41, INS Unix (LF), 099 notifications, 19/01/2024.



El botón "pedidosCli1" será el encargado de mostrar primeramente los artículos que vende el fabricante con id de fabricante 1 y posteriormente de recuperar los pedidos de los clientes que han pedido productos de ese fabricante.

Lo primero será vaciar el `textPanel`, las variables `salida` y `sentSQL`, de instanciar una nueva conexión a la base de datos y de crear un `statement`.

Tras ello instanciara en la variable `sentSQL` la sentencia para mostrar los productos que ofrece el fabricante 1, le pasaremos esa variable al `resultSet` para que ejecute la sentencia, obtendremos el número de columnas, las recorremos, mediante un `while`, el cual, contendrá un `for` y este a su vez un `if - else` para darle formato al resultado que iremos guardando en la variable `salida`.

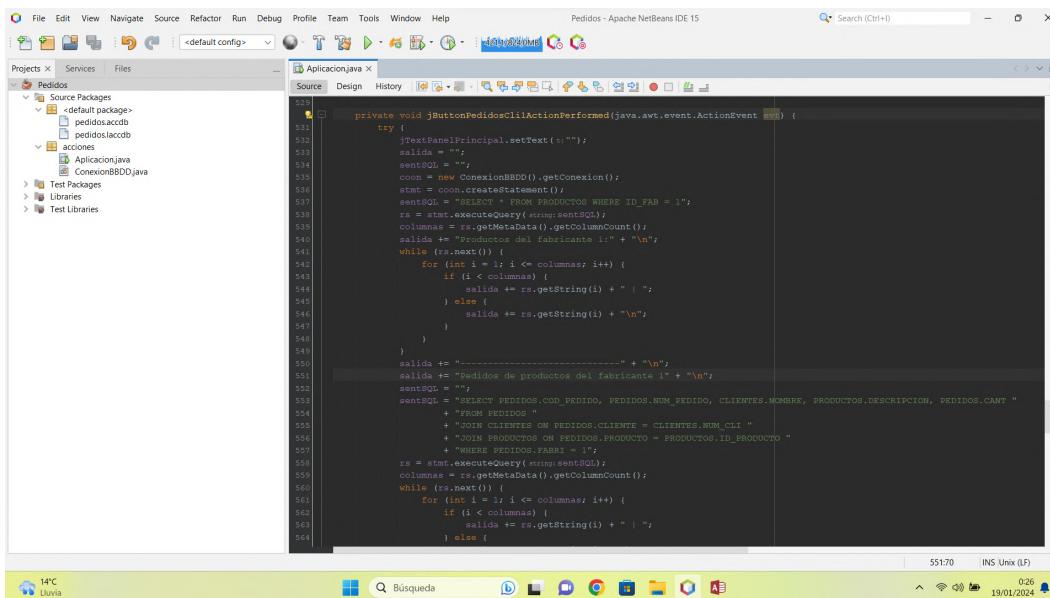
Posteriormente instanciaremos en la variable `sentSQL` la sentencia para mostrar el resultado mediante dos `join` ya que debemos unir tres tablas.

A continuación le pasaremos la variable al `resultSet`, para que ejecute la sentencia mediante el `statement`, obtendremos el número de columnas, las recorreremos con un `while`, un `for` y un `if` como en el caso anterior para darle formato al resultado y guardaremos en la variable `salida` el resultado.

Tras ello le mostraremos en el `textPanel` el contenido de la variable `salida` y en el mensaje correspondiente en el `textField`.

Como en los casos anteriores, todo el código estará envuelto en un `try - catch -finally` para capturar las excepciones y cerrar, pase lo que pase, el `statement`, la conexión y el `resultSet`.

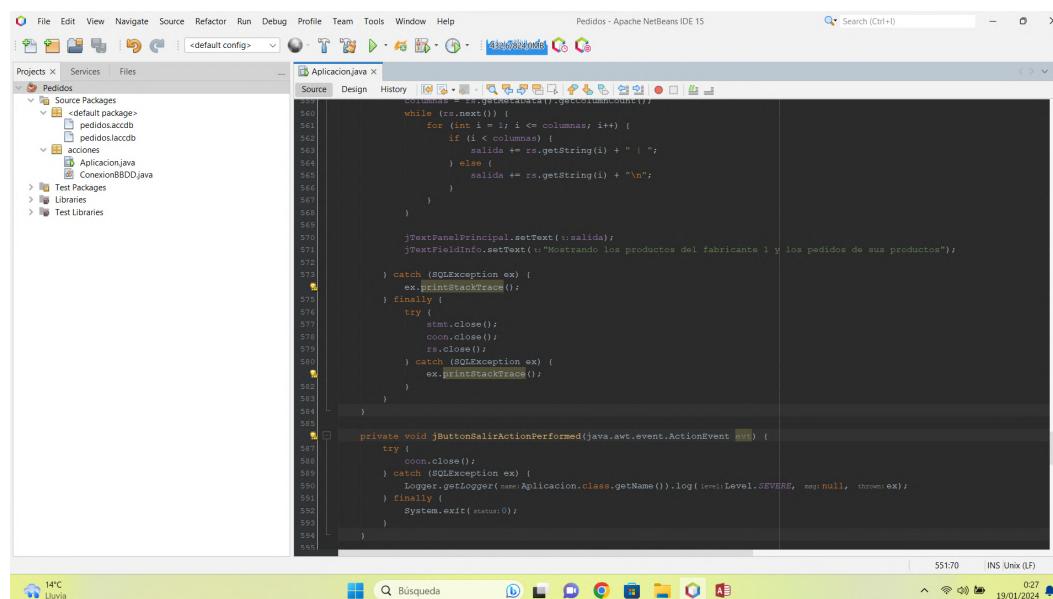
Podemos verlo en la siguientes imágenes.



```

private void jButtonPedidosActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        jTextFieldPrincipal.setText("");
        String salida = "";
        sentSQL = "";
        coon = new ConexionBDD().getConexion();
        stat = coon.createStatement();
        sentSQL = "SELECT * FROM PRODUCTOS WHERE ID_FAB = 1";
        rs = stat.executeQuery(sentSQL);
        columnas = rs.getMetaData().getColumnCount();
        salida += "Productos del fabricante 1" + "\n";
        while (rs.next()) {
            for (int i = 1; i <= columnas; i++) {
                if (i < columnas) {
                    salida += rs.getString(i) + " | ";
                } else {
                    salida += rs.getString(i) + "\n";
                }
            }
        }
        salida += "-----" + "\n";
        salida += "Pedidos de productos del fabricante 1" + "\n";
        sentSQL = "SELECT PEDIDOS.COD_PEDIDO, PEDIDOS.NUM_PEDIDO, CLIENTES.NOMBRE, PRODUCTOS.DESCRIPCION, PEDIDOS.CANT "
                + "FROM PEDIDOS "
                + "JOIN CLIENTES ON PEDIDOS.CLIENTE = CLIENTES.NUM_CLI "
                + "JOIN PRODUCTOS ON PEDIDOS.PRODUCTO = PRODUCTOS.ID_PRODUCTO "
                + "WHERE PEDIDOS.FABRI = 1";
        rs = stat.executeQuery(sentSQL);
        columnas = rs.getMetaData().getColumnCount();
        while (rs.next()) {
            for (int i = 1; i <= columnas; i++) {
                if (i < columnas) {
                    salida += rs.getString(i) + " | ";
                } else {
                    salida += rs.getString(i) + "\n";
                }
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally {
        try {
            stat.close();
            coon.close();
            rs.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}

```

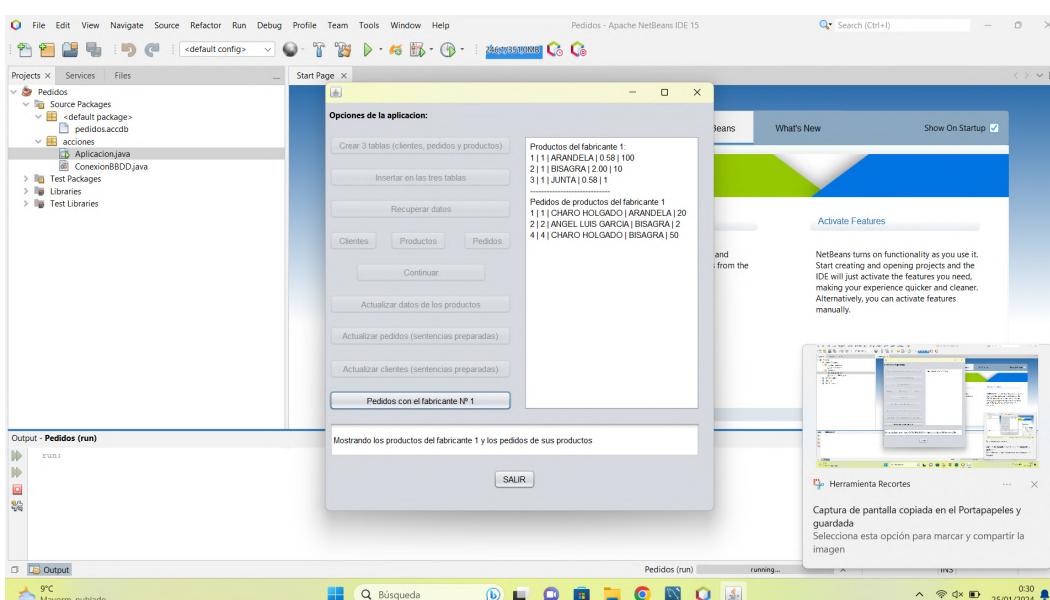


```

private void jButtonPedidosActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        jTextFieldPrincipal.setText("");
        jTextFieldInfo.setText("Mostrando los productos del fabricante 1 y los pedidos de sus productos");
        while (rs.next()) {
            for (int i = 1; i <= columnas; i++) {
                if (i < columnas) {
                    salida += rs.getString(i) + " | ";
                } else {
                    salida += rs.getString(i) + "\n";
                }
            }
        }
        jTextFieldPrincipal.setText(salida);
        jTextFieldInfo.setText("Mostrando los productos del fabricante 1 y los pedidos de sus productos");
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally {
        try {
            stat.close();
            coon.close();
            rs.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}

private void jButtonSalirActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        coon.close();
    } catch (SQLException ex) {
        Logger.getLogger(Aplicacion.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        System.exit( status );
    }
}

```



El botón salir tendrá la función de cerrar la conexión (si estuviese abierta) y cerrar el programa mediante el método System.exit pasandole el status 0, envolviendo el cierre de la conexión en un try - catch y metiendo en el finally el cierre de la aplicación.

El método "tabExist" contendrá un método para verificar si la tabla que el pasemos existe o no en la base de datos.

Para ello crearemos un DatabaseMetaData al cual le pasaremos la conexión a la base de datos, un resultSet, al cual le pasaremos, en el tercer parámetro, el nombre de la tabla y retornará si existe o mediante un boolean.

Podemos verlo en la siguiente imagen.

```

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Pedidos - Apache NetBeans IDE 15
Search (Ctrl+F)
Projects Services Files
Aplicacion.java
Source Design History
177     rs.close();
178   } catch (SQLException ex) {
179     ex.printStackTrace();
180   }
181 }
182 }
183 }
184 }
185 }
186 }
187 private void jButtonSalirActionPerformed(java.awt.event.ActionEvent evt) {
188   try {
189     con.close();
190   } catch (SQLException ex) {
191     Logger.getLogger(Aplicacion.class.getName()).log(Level.SEVERE, null, ex);
192   } finally {
193     System.exit(vesnor0);
194   }
195 }
196 }
197 private boolean tabExist(String nomTabla) throws SQLException {
198   DatabaseMetaData dmt = coon.getMetaData();
199   ResultSet rs = dmt.getTables(null, null, nomTabla, null);
200   return rs.next();
201 }
202 }
203 private void recDatos(String nombre) {
204   try {
205     coon = new ConexionBDD().getConexion();
206     statmt = coon.createStatement();
207     sentsQL = "SELECT * FROM " + nombre;
208     rs = statmt.executeQuery(sentsQL);
209     columnas = rs.getMetaData().getColumnCount();
210     while (rs.next()) {
211       for (int i = 1; i < columnas; i++) {
212         if (!i < columnas) {
213           if (rs.getMetaData().getColumnName(i).equals("FECHA_PEDIDO")) {
214             Date fecha = rs.getDate(i);
215             ...
216           }
217         }
218       }
219     }
220   } catch (SQLException ex) {
221     ex.printStackTrace();
222   }
223 }
224 }
225 
```

El método "recDatos" será el encargado de recuperar todos los datos de las tablas.

En primer lugar instanciaremos una nueva conexión a la base de datos, instanciaremos un statement e instanciaremos en la variable sentsQL la sentencia para recuperar dichos datos.

Tras ello le pasaremos dicha sentencia al resultSel para que ejecute la sentencia y obtendremos el número de columnas para almacenarlo en la variable columnas.

Mediante un while que contendrá un for y dos if anidados recorreremos el resultSet para obtener los resultados y darle formato a la información obtenida, incluida la fecha.

Todo el código estará envuelto en un try - catch -finally que controlara las excepciones y cerrará, en el finally, la conexión, el statement y el resultSet.

```

private void recetas(String nombre) {
    try {
        coon = new ConexionBBDD().getConexion();
        stmt = coon.createStatement();
        sentSQL = "SELECT * FROM " + nombre;
        rs = stmt.executeQuery(sentSQL);
        columnas = rs.getMetaData().getColumnCount();
        while (rs.next()) {
            for (int i = 1; i < columnas; i++) {
                if (rs.getMetaData().getColumnName(i).equals("FECHA_PEDIDO")) {
                    Date fecha = rs.getDate(i);
                    SimpleDateFormat formatoFecha = new SimpleDateFormat(pattern:"dd/MM/YYYY");
                    salida += formatoFecha.format(fecha) + " | ";
                } else {
                    salida += rs.getString(i) + " | ";
                }
            }
            salida += "\n";
        }
    } catch (SQLException ex) {
        Logger.getLogger(Aplicacion.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            coon.close();
            stmt.close();
            rs.close();
        } catch (SQLException ex) {
            Logger.getLogger(Aplicacion.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

```

```

public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    [Look and feel setting code (optional)]

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Aplicacion().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton jButtonActualizarClientes;
private javax.swing.JButton jButtonActualizarPedidos;
private javax.swing.JButton jButtonActualizarProductos;
private javax.swing.JButton jButtonContinuar;
private javax.swing.JButton jButtonCrear;
private javax.swing.JButton jButtonInsertar;
private javax.swing.JButton jButtonPedidoscli;
private javax.swing.JButton jButtonRecClientes;
private javax.swing.JButton jButtonRecPedidos;
private javax.swing.JButton jButtonRecProductos;
private javax.swing.JButton jButtonRecuperarff;
private javax.swing.JLabel jLabelTitulo;
private javax.swing.JPanel jPanelPrincipal;
private javax.swing.JScrollPane jScrollPane;
private javax.swing.JTextField jTextFieldInfo;
private javax.swing.JTextPane jTextPanelPrincipal;
// End of variables declaration
}

```

Y con esto damos por finalizada la tarea.