AD

Práctica 2

PERSISTENCIA EN BBDD RELACIONALES

Álvaro Manuel Navarro Cruz
2º DAM
22/11/2024

Índice

Introducción (↑)

En esta práctica, se pretende desarrollar una aplicación para gestionar un hotel que guarde la información en una base de datos.

Se pide realizar el código de persistencia necesario para almacenar los datos de los clientes del hotel. Para ello, necesitaremos la clase ClienteANC de tipo POJO, la clase ClientePersistenciaANC, la clase TestHotelANC y un gestor de conexiones.

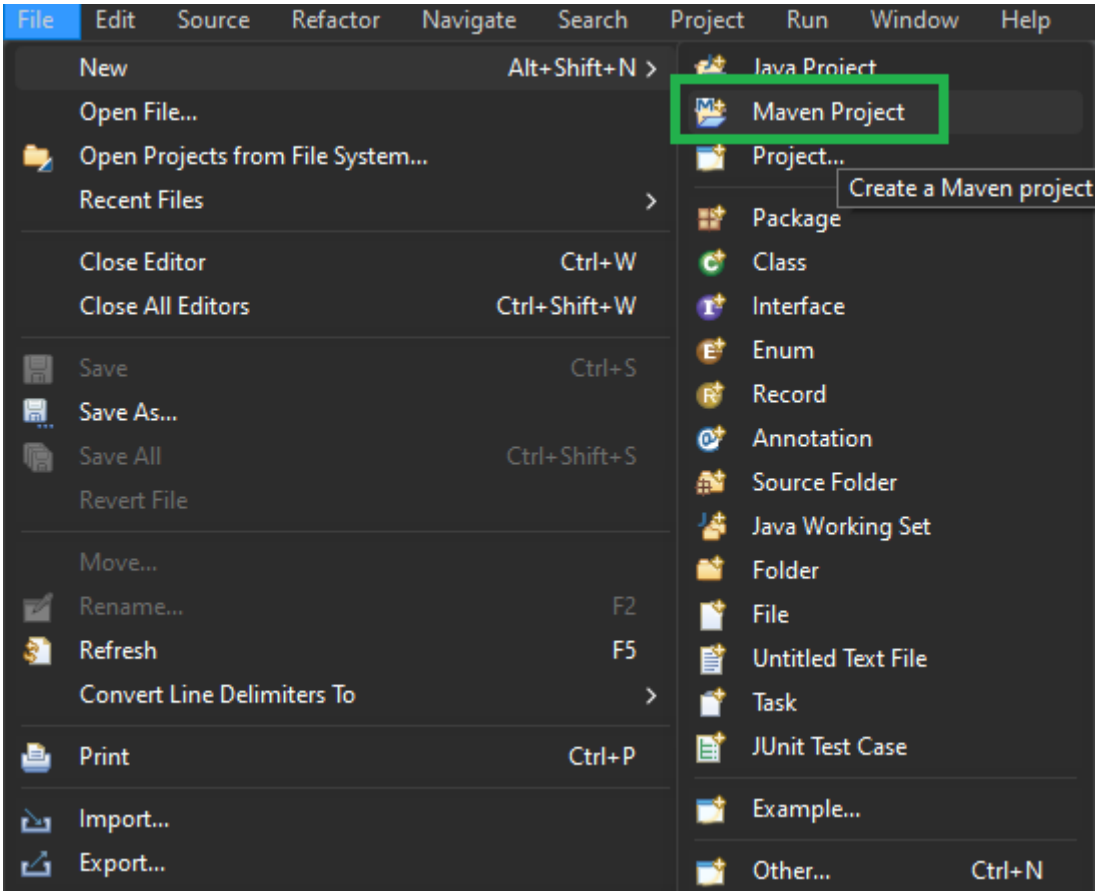
Se debe crear la Base de Datos correspondiente llamada HotelANC, con la tabla ClienteANC, que deberá llevar los campos idCliente, nombreCliente, apellidosCliente, emailCliente, dniCliente y claveCliente.

Enlace a Repositorio en GitHub

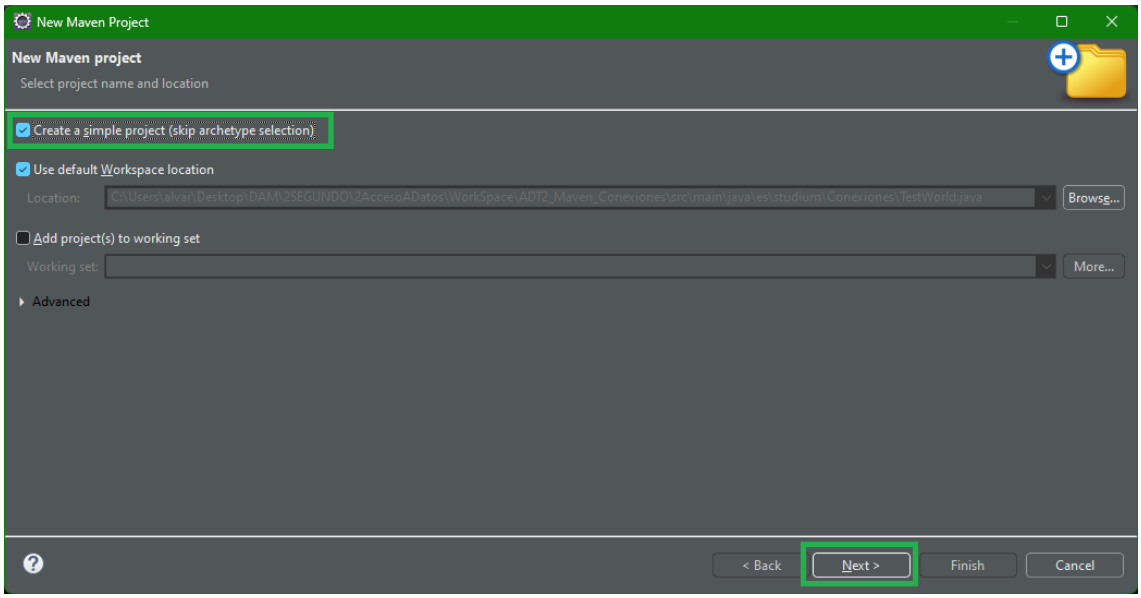
https://github.com/AlvaroMfco/AD_Practica2.git

1.- Documentación (↑)

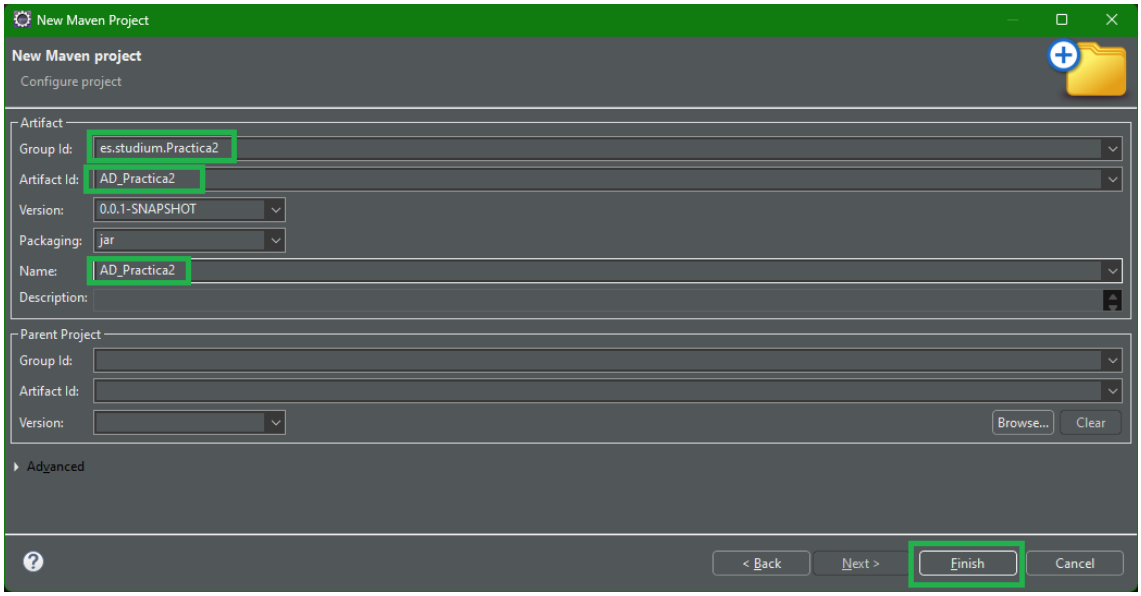
1.1.- Creación Proyecto Maven



Para crear un proyecto tipo Maven debemos clicar e “File -> New -> Maven Project”.

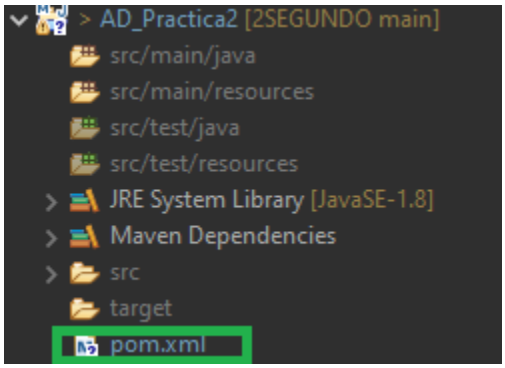


Se nos abrirá esta ventana, en la cual debemos seleccionar la opción marcada en la imagen superior.



A continuación, se nos pedirán una serie de datos, a saber, “Group Id”, en el cual debemos establecer el nombre del paquete que vamos a utilizar, “Artifact Id”, donde debemos proporcionar el nombre del proyecto, que será el mismo que en “Name”.

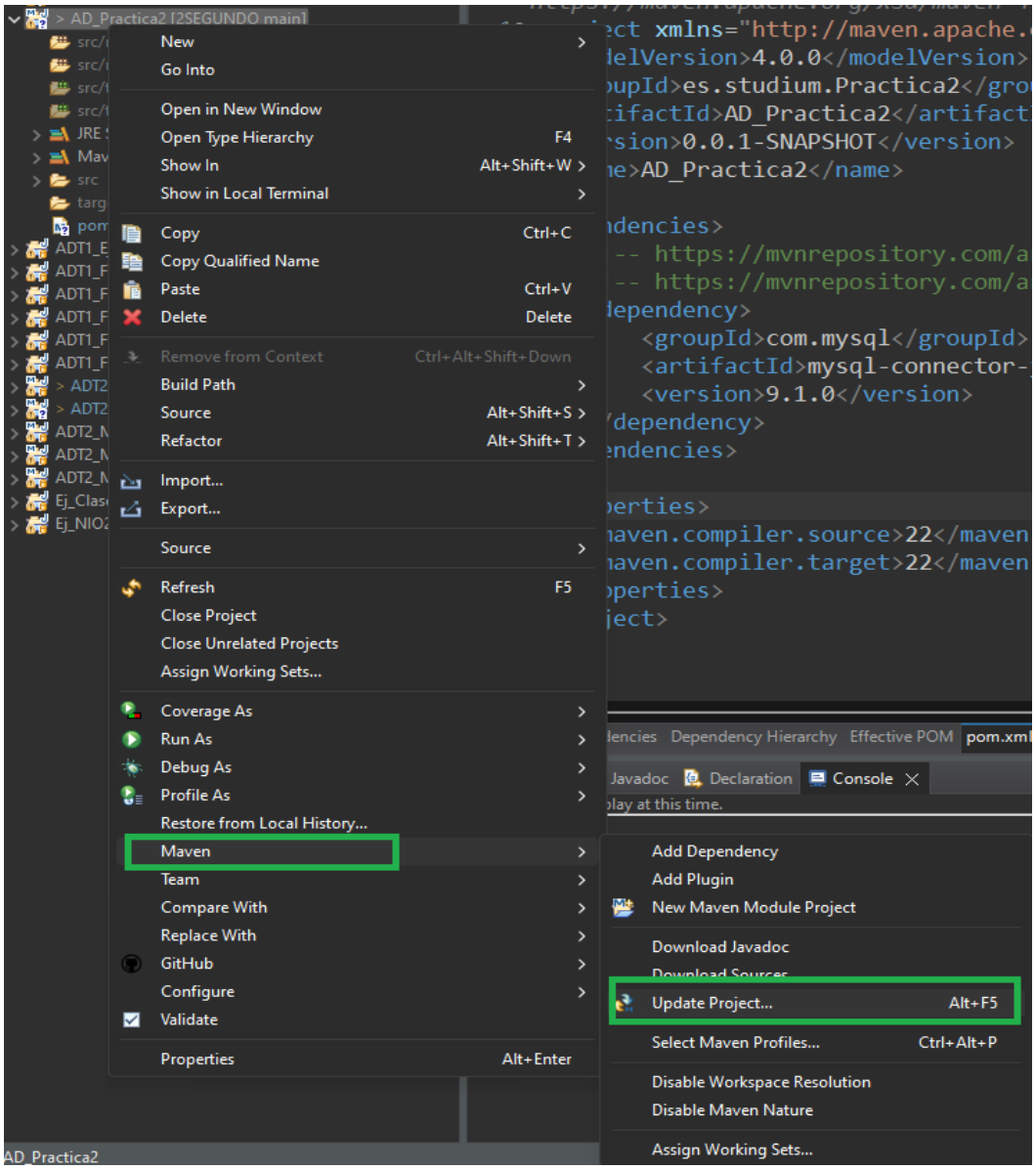
Para finalizar clicamos en “Finish”.



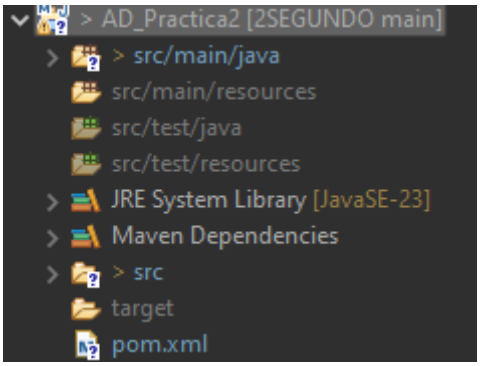
Una vez creado el proyecto, debemos acceder al fichero “pom.xml”, en el que debemos establecer las dependencias de MySQL connector para indicar su versión, además de las propiedades del compilador de Java, indicando la versión del JDK.

```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  <modelVersion>4.0.0</modelVersion>
  <groupId>es.studium.Practica2</groupId>
  <artifactId>AD_Practica2</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>AD_Practica2</name>

<dependencies>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>9.1.0</version>
    </dependency>
</dependencies>

<properties>
    <maven.compiler.source>23</maven.compiler.source>
    <maven.compiler.target>23</maven.compiler.target>
</properties>
</project>
```

Para esta práctica, se utilizarán las últimas versiones disponibles tanto de MySQL como de JDK, quedando como en la imagen superior.

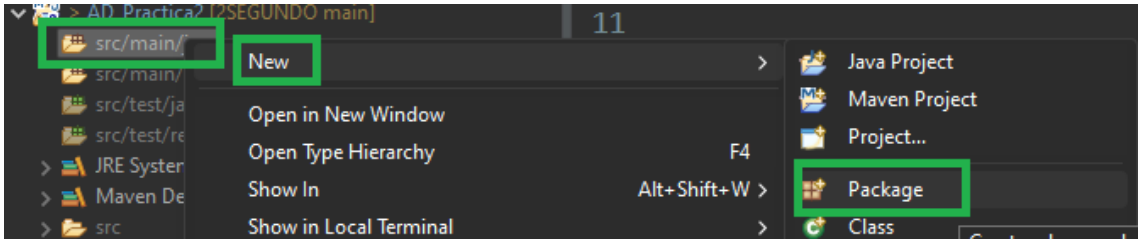


Una vez guardado el fichero pom.xml con los datos correctos, debemos actualizar el proyecto haciendo clic derecho sobre la carpeta del mismo y posteriormente clicar en “Maven -> Update Project…”.



Una vez actualizado el proyecto Maven, en el explorador de paquetes podemos observar que la versión de Java se ha actualizado correctamente, indicando la versión del JDK 23.

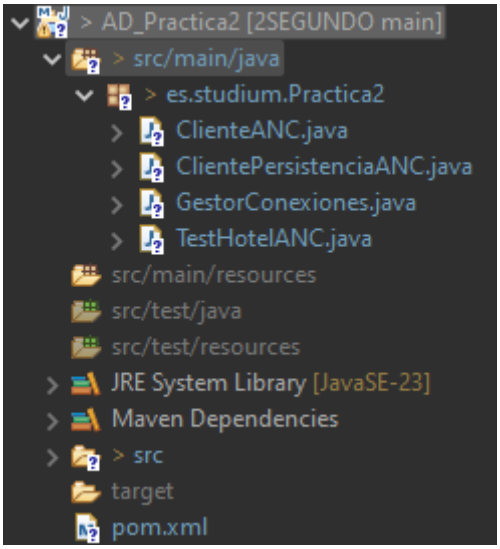
1.2.- Creación de Paquetes y Clases



Para crear un nuevo paquete, debemos hacerlo en la carpeta “src/Main/java”. Para ello, debemos hacer clic derecho sobre la carpeta y clicar en “New -> Package”, al que daremos nombre de “es.studium.Practica2”.

Para crear una clase, debemos hacer clic derecho sobre el paquete que acabamos de crear y navegar hasta “New -> Class”.

Una vez creadas todas las clases necesarias para la práctica, el explorador de paquetes se verá de la siguiente manera.



2.- Creación de la BBDD (↑)

Nuestra Base de Datos, llamada **hotelanc**, tendrá una tabla llamada **clienteanc**, que a su vez estará compuesta por las columnas **idCliente, nombreCliente, apellidosCliente, emailCliente, dniCliente y claveCliente**.

```
1  CREATE DATABASE hotelANC CHARSET utf8mb4 COLLATE utf8mb4_spanish2_ci;
2  USE hotelANC;
3
4  CREATE TABLE clienteanc (idCliente INT AUTO_INCREMENT, nombreCliente VARCHAR(45) NOT NULL,
5  apellidosCliente VARCHAR(45) NOT NULL, emailCliente VARCHAR(200) NOT NULL,
6  dniCliente VARCHAR(10) NOT NULL,
7  claveCliente VARCHAR(45) NOT NULL,
8  PRIMARY KEY (idCliente)
9  );
```

3.- Capturas del código

3.1.- Clase ClienteANC

```java
package es.studium.Practica2;

public class ClienteANC {
    private int idCliente;
    private String nombre;
    private String apellidos;
    private String email;
    private String dni;
    private String clave;

    //Constructor vacío
    public ClienteANC() {
        idCliente = 0;
        nombre = "";
        apellidos = "";
        email = "";
        dni = "";
        clave = "";
    }

    //Constructor por parámetros
    public ClienteANC(String nombre, String apellidos, String email, String dni, String clave) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.email = email;
        this.dni = dni;
        this.clave = clave;
    }

    public int getID() {
        return idCliente;
    }

    public void setID(int idCliente){
        this.idCliente = idCliente;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getDni() {
        return dni;
    }

    public void setDni(String dni) {
        this.dni = dni;
    }

    public String getClave() {
        return clave;
    }

    public void setClave(String clave) {
        this.clave = clave;
    }
}
```

3.2.- Clase ClientePersistenciaANC

```java
package es.studium.Practica2;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class ClientePersistenciaANC {
    static int idCliente;
    static String campo;

    //Método para crear clientes
    /*1-Instanciamos la conexión, ejecutamos la sentencia SQL para insertar mediante un Statement
      y posteriormente usamos un resultset para almacenar el id del cliente recien creado mediante su dni,
      ya que debe ser único*/
    public static int crearCliente(String nombre, String apellidos,
            String email, String dni, String clave) throws SQLException{

        Connection conexion = GestorConexiones.getMySQL_Connection("hotelanc");
        String sentencia = "INSERT INTO clienteanc VALUES (null, '" + nombre + "','" + apellidos +
                "','" + email + "','" + dni + "','" + clave + "')";

            Statement st = conexion.createStatement();

            // Ejecutar el INSERT
            st.execute(sentencia);

            // Consulta para obtener el ID del cliente recién insertado
            String queryId = "SELECT idCliente FROM clienteanc WHERE dniCliente = '" + dni +
                    "' ORDER BY idCliente DESC LIMIT 1";
            ResultSet rs = st.executeQuery(queryId);
            while (rs.next()) {
                idCliente = rs.getInt("idCliente");  // Obtiene el ID del cliente recién insertado
            }
            // Cerrar ResultSet y Statement
            rs.close();
            st.close();
            conexion.close();

        /* Devuelve el id del nuevo cliente */
        return idCliente;
    }

    //Método para leer clientes.
    /*Creamos la conexión y ejecutamos una sentencia sql de consulta para
     * mostrar el campo pasado como parámetro.*/
    public static String leerCliente(int idCliente, String campo) throws SQLException{
        String nombre = "";
        Connection conexion = GestorConexiones.getMySQL_Connection("hotelanc");
        String sentencia = "SELECT " + campo + " FROM clienteanc WHERE idCliente = " + idCliente+";";
            Statement st = conexion.createStatement();
            ResultSet rs = st.executeQuery(sentencia);
            while(rs.next()) {
                nombre = rs.getString(campo);
            }
            rs.close();
            st.close();
            conexion.close();

        /* Devuelve el valor de la columna "campo" del cliente identificado por
        "idCliente" */
        return "El nombre del cliente es " + nombre + "\n\n";

    }
```

```java
    //Método para actualizar clientes.
    /*Creamos la sentencia Update para actualizar el campo pasado como parámetro
      con el nuevo valor.*/
    public static boolean actualizarCliente(int idCliente, String campo,
            String nuevoValor) throws SQLException{

        Connection conexion = GestorConexiones.getMySQL_Connection("hotelanc");
        String sentencia = "UPDATE hotelanc.clienteanc SET " + campo + " = '" + nuevoValor +
                "' WHERE idCliente = " + idCliente+";";
        Statement st = conexion.createStatement();
        //Si aplica a más de 0 líneas, es decir, si hay cambios.
        if(st.executeUpdate(sentencia) > 0) {
            return true;
        }
        /* Actualiza el valor de la columna "campo" del cliente identificado por
        "idCliente". Devuelve true si se ha logrado actualizar. */
        return false;

    }

    //Método para eliminar clientes.
    /*Utilizamos la sentencia Delete para eliminar el cliente con id
      pasado como parámetro*/
    public static boolean eliminarCliente(int idCliente) throws SQLException{
        Connection conexion = GestorConexiones.getMySQL_Connection("hotelanc");
        String sentencia = "DELETE FROM hotelanc.clienteanc WHERE idCliente = " + idCliente+";";
        Statement st = conexion.createStatement();

        if(st.executeUpdate(sentencia) > 0) {
            return true;
        }
        /* Elimina el cliente identificado por "idCliente". Devuelve true si se ha logrado
        eliminar. */
        return false;
    }

    //Método para mostrar clientes por consola
    /*Ejecutamos la sentencia SELECT para obtener en el ResultSet
      todos los campos de la tabla*/
    public static void mostrarClientes() throws SQLException {
        //Mostrar los datos de los clientes creados por consola
        Connection conexion = GestorConexiones.getMySQL_Connection("hotelanc");
        Statement st = conexion.createStatement();
        String queryDatos = "SELECT * FROM hotelanc.clienteanc;";
        ResultSet rs = st.executeQuery(queryDatos);
        while(rs.next()) {
            System.out.println(rs.getInt("idCliente") + " - " + rs.getString("nombreCliente") +
                    " - " + rs.getString("apellidosCliente") +
                    " - " + rs.getString("emailCliente") + " - " + rs.getString("dniCliente") +
                    " - " + rs.getString("claveCliente"));
        }
    }
}
```

3.3.- Clase TestHotelANC

```java
package es.studium.Practica2;

import java.sql.SQLException;
import java.util.Scanner;

public class TestHotelANC {
    static int opcion;
    static ClienteANC cliente1;
    static ClienteANC cliente2;
    static ClienteANC cliente3;
    static int idCliente1;
    static int idCliente2;
    static int idCliente3;
    static boolean sigue = true;
    static boolean clientesCreados = false;

    public static void main(String[] args) throws SQLException {
        Scanner teclado = new Scanner(System.in);
        // Mostramos el menú y ejecutamos la operación según el número introducido.
        do {
            mostrarMenu();
            opcion = teclado.nextInt();
            try {
                operaciones(opcion, teclado);
            } catch (SQLException e) {
                e.printStackTrace();
                sigue = false;
            }
        }
        // Lo anterior se repite hasta que sigue sea False.
        while (sigue);
        teclado.close();
    }


    public static void mostrarMenu() {
        System.out.println("*****************************************");
        System.out.println("------ ¿Qué operación desea realizar? ------");
        System.out.println(" 1 - Crear clientes\n 2 - Leer cliente\n 3 - Actualizar cliente\n 4 - "
                + "Eliminar cliente\n 5 - Salir del programa");
        System.out.println("*****************************************");
    }

    public static void operaciones(int opcion, Scanner teclado) throws SQLException {
        // Si la opción es 1.
        if (opcion == 1) {
            // Creamos los clientes mediante el constructor por parámetros
            cliente1 = new ClienteANC("Álvaro", "Navarro Cruz", "alvaromfco@gmail.com",
                    "49545195K", "Studium2023;");
            cliente2 = new ClienteANC("Patricia", "Velasco Reverendo", "patricia@gmail.com",
                    "12345678A", "Contraseña1");
            cliente3 = new ClienteANC("Juan", "Martínez Sánchez", "juan@gmail.com",
                    "87654321B", "Contraseña2");
            // Almacenamos los IDs de los 3 clientes en variables.
            idCliente1 = ClientePersistenciaANC.crearCliente(cliente1.getNombre(), cliente1.getApellidos(),
                    cliente1.getEmail(), cliente1.getDni(), cliente1.getClave());
            idCliente2 = ClientePersistenciaANC.crearCliente(cliente2.getNombre(), cliente2.getApellidos(),
                    cliente2.getEmail(), cliente2.getDni(), cliente2.getClave());
            idCliente3 = ClientePersistenciaANC.crearCliente(cliente3.getNombre(), cliente3.getApellidos(),
                    cliente3.getEmail(), cliente3.getDni(), cliente3.getClave());
            // Seteamos los IDs a cada cliente.
            cliente1.setID(idCliente1);
            cliente2.setID(idCliente2);
            cliente3.setID(idCliente3);
            // Mostramos Feedback por consola
            System.out.println("Clientes creados correctamente.\n");
            // Mostramos los clientes creados por consola
            ClientePersistenciaANC.mostrarClientes();
            // Establecemos que los clientes han sido creados
            clientesCreados = true;
        }
```

```
72
73          // Si la opción es 2
74          if (opcion == 2) {
75              // Comprobamos si los clientes han sido creados
76              if (clientesCreados == true) {
77                  System.out.println("-- Accediendo al nombre del cliente " + cliente2.getID() + " --\n");
78                  // Leemos el campo nombreCliente del cliente con ID 2.
79                  System.out.println(ClientePersistenciaANC.leerCliente(cliente2.getID(), "nombreCliente"));
80              }
81              // Si los clientes no están creados
82              else
83                  System.out.println("Primero tienes que crear los clientes");
84          }
85
86          // Si la opción es 3
87          if (opcion == 3) {
88              String campo = "nombreCliente";
89              System.out.println("-- Introduce un nuevo nombre para el cliente -- ");
90              String nombre = teclado.next();
91              // Comprobamos que la actualización devuelve true y mostramos el cambio por
92              // consola
93              if (ClientePersistenciaANC.actualizarCliente(idCliente2, campo, nombre)) {
94                  System.out.println(
95                          "\nCliente " + cliente2.getID() + " Actualizado correctamente con nombre " + nombre + "\n");
96              }
97
98          }
99          // Si la opción es 4
100         if (opcion == 4) {
101             // Confirmamos la eliminación
102             System.out.println("\n-- Desea eliminar el registro con id " + cliente2.getID() + "? --\n");
103             System.out.println("Escribe si/no\n");
104             String respuesta = teclado.next();
105             // Si se confirma, comprobamos si la eliminación se ha realizado correctamente
106             if (respuesta.equals("si")) {
107                 if (ClientePersistenciaANC.eliminarCliente(idCliente2)) {
108                     System.out.println("\nCliente eliminado correctamente\n");
109                     System.out.println("Los clientes actuales son: \n");
110                     // Mostramos los clientes para confirmar que se ha eliminado.
111                     ClientePersistenciaANC.mostrarClientes();
112                 }
113             }
114         }
115         // Si la opción es 5, finaliza el bucle.
116         if (opcion == 5) {
117             System.out.println("**** Fin del programa ****");
118             sigue = false;
119         }
120     }
121 }
```

3.4.- Clase GestorConexiones

```java
package es.studium.Practica2;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class GestorConexiones {
    private final static String MySQL_DB_USUARIO = "root";
    private final static String MySQL_DB_PASSWORD = "Studium2023;";

    private final static String MySQL_DB_DRIVER = "com.mysql.cj.jdbc.Driver";
    private final static String MySQL_DB_URL = "jdbc:mysql://localhost/";

    public static Connection getMySQL_Connection(String database) {

        Connection connMySQL = null;
        try {
            Class.forName(MySQL_DB_DRIVER);
            connMySQL = DriverManager.getConnection(MySQL_DB_URL + database,
                    MySQL_DB_USUARIO, MySQL_DB_PASSWORD);

        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException eq) {
            eq.printStackTrace();
        }
        return connMySQL;
    }
}
```

4.- Ejecución (↑)

Para la ejecución de nuestro programa, en primer lugar, comprobamos que la tabla de la BBDD está vacía.

```
1 •   SELECT * FROM hotelanc.clienteanc;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idCliente	nombreCliente	apellidosCliente	emailCliente	dniCliente	claveCliente
NULL	NULL	NULL	NULL	NULL	NULL

Problems | Javadoc | Declaration | Console

TestHotelANC [Java Application] C:\Users\alvar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspc

```
**********************************************
------ ¿Qué operación desea realizar? ------
 1 - Crear clientes
 2 - Leer cliente
 3 - Actualizar cliente
 4 - Eliminar cliente
 5 - Salir del programa
**********************************************
```

En la ejecución de nuestro programa, aparecerá un menú por consola para elegir la acción que queremos realizar. Debemos elegir la opción introduciendo el número correspondiente.

4.1.- Creación de Clientes

```
1
Clientes creados correctamente.

1 - Álvaro - Navarro Cruz - alvaromfco@gmail.com - 49545195K - Studium2023;
2 - Patricia - Velasco Reverendo - patricia@gmail.com - 12345678A - Contraseña1
3 - Juan - Martínez Sánchez - juan@gmail.com - 87654321B - Contraseña2
*****************************************
------ ¿Qué operación desea realizar? ------
 1 - Crear clientes
 2 - Leer cliente
 3 - Actualizar cliente
 4 - Eliminar cliente
 5 - Salir del programa
*****************************************
```

Si elegimos la opción 1 – Crear Clientes, aparecerá por consola un mensaje indicando si se han creado correctamente o si, por el contrario, ha ocurrido algún error.

Si los clientes se crean correctamente, se mostrarán por consola.

```
1 •  SELECT * FROM hotelanc.clienteanc;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wra

idCliente	nombreCliente	apellidosCliente	emailCliente	dniCliente	claveCliente
1	Álvaro	Navarro Cruz	alvaromfco@gmail.com	49545195K	Studium2023;
2	Patricia	Velasco Reverendo	patricia@gmail.com	12345678A	Contraseña1
3	Juan	Martínez Sánchez	juan@gmail.com	87654321B	Contraseña2
NULL	NULL	NULL	NULL	NULL	NULL

Como podemos ver, la tabla de la BBDD se ha actualizado con los datos correctos.

4.2.- Leer Cliente

Para esta ejecución, vamos a leer el campo “nombreCliente”, del cliente con ID = 2.

```
2
-- Accediendo al nombre del cliente 2 --

El nombre del cliente es Patricia


**********************************************
------ ¿Qué operación desea realizar? ------
 1 - Crear clientes
 2 - Leer cliente
 3 - Actualizar cliente
 4 - Eliminar cliente
 5 - Salir del programa
**********************************************
```

Vamos a cambiar el código para acceder ahora al nombre del cliente 3:

```
//Si la opción es 2
if (opcion == 2) {
    //Comprobamos si los clientes han sido creados
    if (clientesCreados == true) {
        System.out.println("-- Accediendo al nombre del cliente " + cliente3.getID() + " --\n");
        //Leemos el campo nombreCliente del cliente con ID 2.
        System.out.println(ClientePersistenciaANC.leerCliente(cliente3.getID(), "nombreCliente"));
    }
    //Si los clientes no están creados
    else
        System.out.println("Primero tienes que crear los clientes");
}
```

Al ejecutar el programa y elegir la opción 2, se mostrará el nombre del cliente con Id 3.

```
2
-- Accediendo al nombre del cliente 3 --

El nombre del cliente es Juan
```

4.3.- Actualizar Cliente

Si elegimos la opción 3, se nos pedirá un nuevo nombre para editar el cliente con Id 2.

Al introducirlo, se mostrará un mensaje que confirma si se ha actualizado correctamente el nombre.

```
3
-- Introduce un nuevo nombre para el cliente --
Pepa

Cliente 2 Actualizado correctamente con nombre Pepa

**********************************************
------ ¿Qué operación desea realizar? ------
 1 - Crear clientes
 2 - Leer cliente
 3 - Actualizar cliente
 4 - Eliminar cliente
 5 - Salir del programa
**********************************************
```

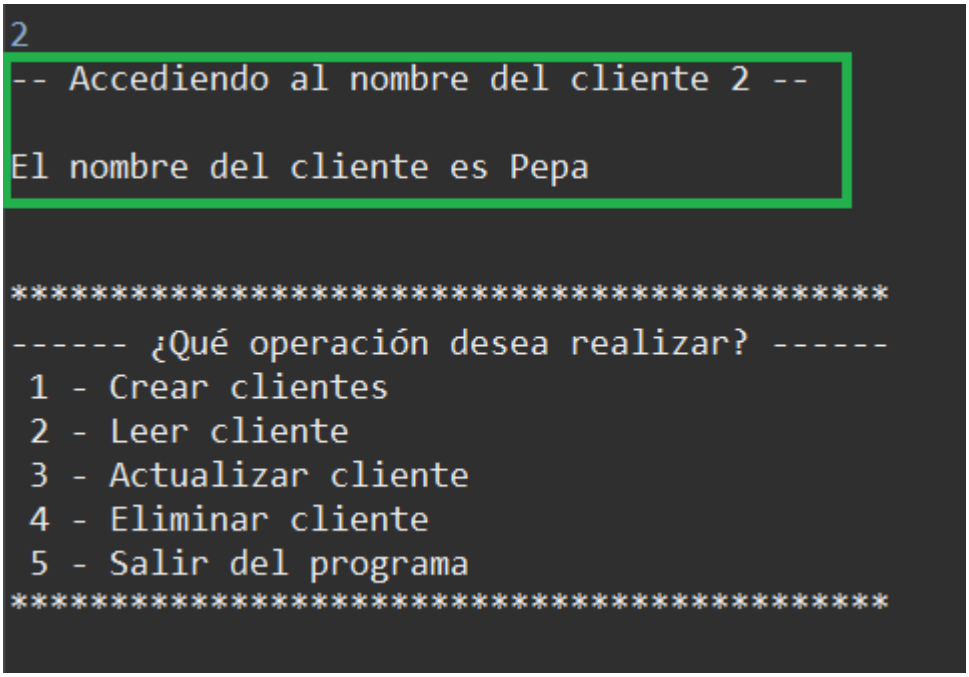
Si comprobamos la tabla de la BBDD, podemos observar que se ha actualizado el nombre correctamente.

```
1 •  SELECT * FROM hotelanc.clienteanc;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wra

idCliente	nombreCliente	apellidosCliente	emailCliente	dniCliente	claveCliente
1	Álvaro	Navarro Cruz	alvaromfco@gmail.com	49545195K	Studium2023;
2	Pepa	Velasco Reverendo	patricia@gmail.com	12345678A	Contraseña1
3	Juan	Martínez Sánchez	juan@gmail.com	87654321B	Contraseña2
NULL	NULL	NULL	NULL	NULL	NULL

Si, por otra parte, volvemos a elegir la opción 2 – Leer Cliente, con el Id del cliente 2, aparecerá el nombre cambiado correctamente:



4.4.- Eliminar Cliente

Si elegimos la opción 4 – Eliminar Cliente, aparecerá un mensaje para confirmar si deseamos continuar con la operación.

Si escribimos “si”, el cliente se eliminará de nuestra Base de Datos y se mostrará por consola los clientes que quedan.

```
4

-- Desea eliminar el registro con id 2? --

Escribe si/no

si

Cliente eliminado correctamente

Los clientes actuales son:

1 - Álvaro - Navarro Cruz - alvaromfco@gmail.com - 49545195K - Studium2023;
3 - Juan - Martínez Sánchez - juan@gmail.com - 87654321B - Contraseña2
*****************************************
------ ¿Qué operación desea realizar? ------
 1 - Crear clientes
 2 - Leer cliente
 3 - Actualizar cliente
 4 - Eliminar cliente
 5 - Salir del programa
*****************************************
```

Si hacemos un SELECT en la Base de Datos, podremos ver que el cliente con Id 2 ha sido eliminado.

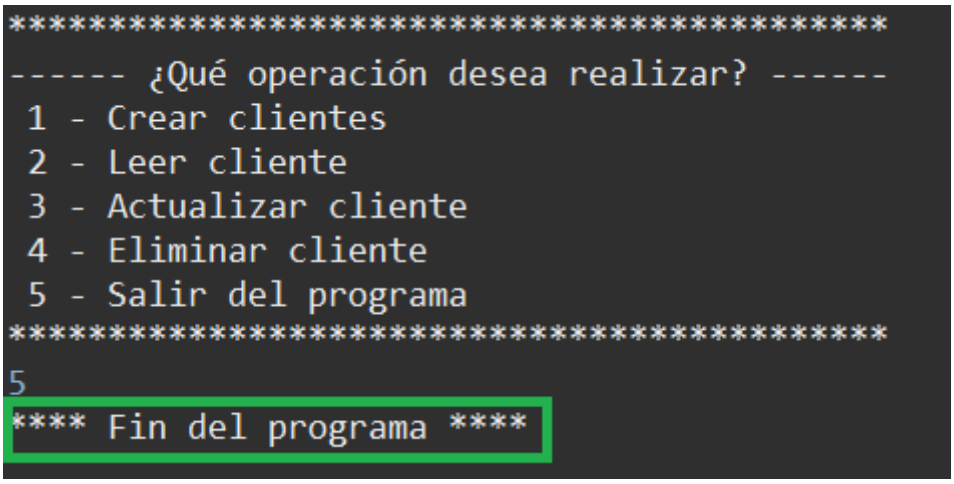
```
1 •   SELECT * FROM hotelanc.clienteanc;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

idCliente	nombreCliente	apellidosCliente	emailCliente	dniCliente	claveCliente
1	Álvaro	Navarro Cruz	alvaromfco@gmail.com	49545195K	Studium2023;
3	Juan	Martínez Sánchez	juan@gmail.com	87654321B	Contraseña2
NULL	NULL	NULL	NULL	NULL	NULL

4.5.- Fin del Programa

Para finalizar el bucle de ejecución, debemos introducir el número 5 por consola, lo que hará que se muestre por pantalla que el programa se ha detenido.



5.- Valoración Personal (↑)

La realización de esta práctica me ha parecido muy interesante, a la vez que entretenida, teniendo que utilizar conexiones a una Base de Datos y la creación de métodos para las operaciones CRUD de una manera mucho más simple que en el primer curso.

Además, ha servido para afianzar los conocimientos vistos en las asignaturas tanto de Bases de Datos, como en Acceso a Datos.

Tengo mucho interés en esta asignatura y me parece que el método de conexión con Hibernate, que veremos en el próximo tema, será mucho más sencillo de implementar, facilitando y agilizando el proceso de creación de este tipo de programas.

6.-Bibliografía

Grupo Studium (↑)

1. *María José Martínez Navas, Acceso a Datos (Práctica Tema 2)*. Publicado en **Grupo Studium.**
Recuperado:
https://campustudium.com/pluginfile.php/15090/mod_resource/content/6/Tema%202%20-%20Persistencia%20en%20BD%20Relacionales%20-%20Pr%C3%A1ctica_v2.pdf
Último acceso (18/11/2024).