CENTRO SAFA NUESTRA SEÑORA DE LOS REYES.
DEPARTAMENTO DE INFORMÁTICA.

AAD

2ºDAM

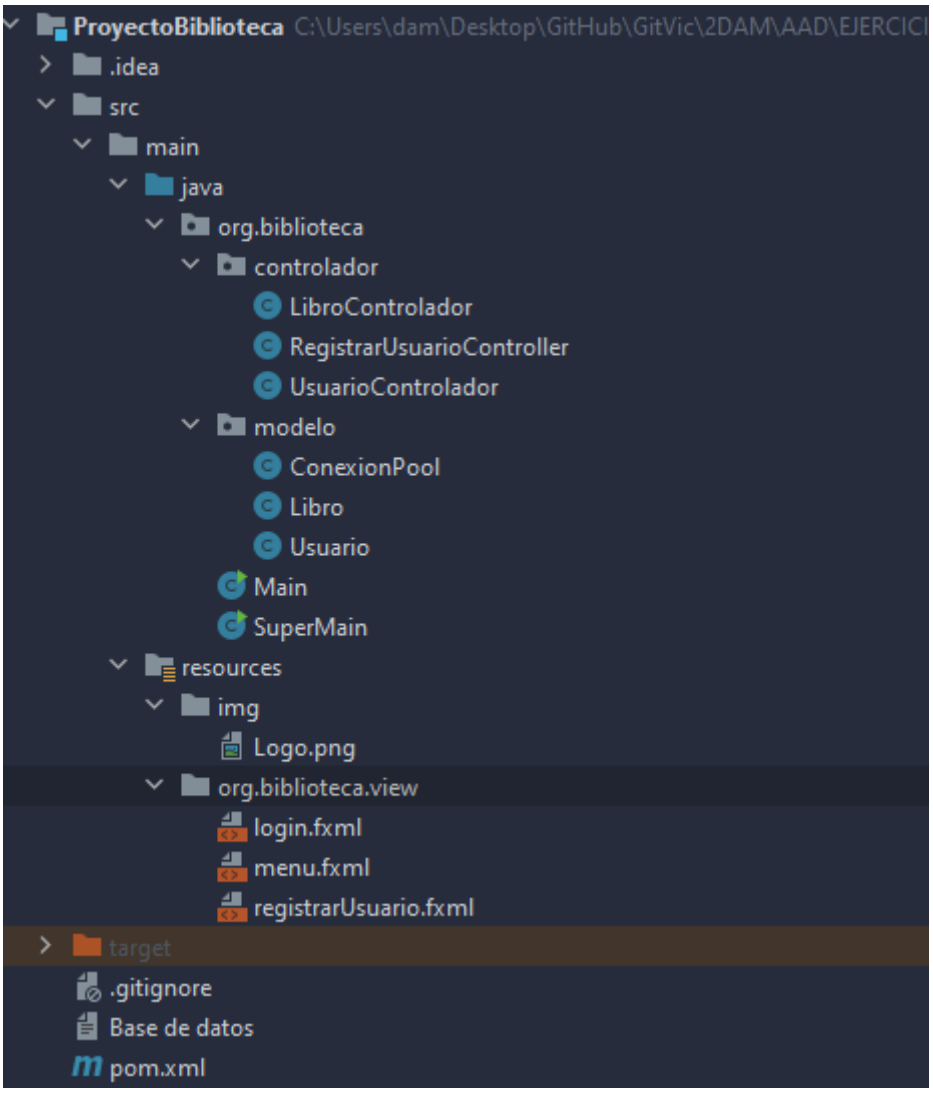
Curso 2022/23

VÍCTOR ADONAY GALIANO POMBO

library

Índice

Estructura del proyecto



Métodos de los modelos

Vamos a comentar el modelo Libro, ya que es el más completo de todos los métodos.
Esta es la clase que representa un libros de la biblioteca

```
public class Libro {
```

Propiedades del libro, utilizando JavaFX properties para facilitar el enlace de datos

```
3 usages
private final IntegerProperty idLibro;
3 usages
private final StringProperty titulo;
3 usages
private final StringProperty autor;
3 usages
private final IntegerProperty anioPublicacion;
3 usages
private final IntegerProperty cantidadDisponible;
```

Constructor de la clase que inicializa las propiedades con los valores proporcionados

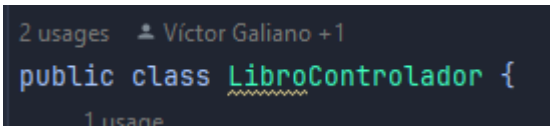
```
7 usages  Víctor Galiano
public Libro(int idLibro, String titulo, String autor, int anioPublicacion, int cantidadDisponible) {
    this.idLibro = new SimpleIntegerProperty(idLibro);
    this.titulo = new SimpleStringProperty(titulo);
    this.autor = new SimpleStringProperty(autor);
    this.anioPublicacion = new SimpleIntegerProperty(anioPublicacion);
    this.cantidadDisponible = new SimpleIntegerProperty(cantidadDisponible);
}
```

Métodos de acceso y modificación para la propiedades.

```
1 usage  Víctor Galiano
public int getIdLibro() { return idLibro.get(); }

no usages  Víctor Galiano
public IntegerProperty idLibroProperty() { return idLibro; }

3 usages  Víctor Galiano
public String getTitulo() { return titulo.get(); }

1 usage  Víctor Galiano
public StringProperty tituloProperty() { return titulo; }

3 usages  Víctor Galiano
public String getAutor() { return autor.get(); }

1 usage  Víctor Galiano
public StringProperty autorProperty() { return autor; }

2 usages  Víctor Galiano
public int getAnioPublicacion() { return anioPublicacion.get(); }

no usages  Víctor Galiano
public IntegerProperty anioPublicacionProperty() { return anioPublicacion; }

3 usages  Víctor Galiano
public int getCantidadDisponible() { return cantidadDisponible.get(); }

1 usage  Víctor Galiano
public IntegerProperty cantidadDisponibleProperty() { return cantidadDisponible; }
```

Métodos de los controladores

Ahora comentaremos el controlador para la interfaz del menú principal.



Elementos de la interfaz para búsqueda, generación de informes HTML, mostrar fecha, hora, clientes, productos, proveedor, ventas, botones y registros.

```
1 usage
public Button actLibro;
2 usages
private String nombreUsuario;
@FXML
private Label fechaActual, horaActual, usuarioField;
@FXML
private TableView<Libro> librosTableView;
@FXML
private TextField tituloField, autorField, anioField, nuevoTituloField, nuevoAutorField, nuevoAnioField, actualiz
no usages
@FXML
private ChoiceBox<String> seccionChoiceBox, seccionNuevoChoiceBox, seccionActualizarChoiceBox;
@FXML
private TableColumn<Libro, Integer> idColumn;
@FXML
private TableColumn<Libro, String> tituloColumn, autorColumn;
@FXML
private TableColumn<Libro, Integer> anoColumn, cantidadColumn;
5 usages
private boolean automaticUpdatesEnabled = true;
```

Inicialización del controlador, configuración del reloj para mostrar fecha y hora actuales

```java
@FXML
private void initialize() {


    Timeline timeline = new Timeline(
            new KeyFrame ( Duration.seconds( s: 1), event -> {...})
    );
    timeline.setCycleCount(Timeline.INDEFINITE);
    timeline.play();


    configurarColumnas();
    configurarTabla();


    usuarioField.setText("Bienvenido de nuevo "+ nombreUsuario);
}
```

Configuración de las columnas de las tablas

```java
1 usage   Víctor Galiano
private void configurarColumnas() {
    idColumn.setCellValueFactory(new PropertyValueFactory<>("idLibro"));
    tituloColumn.setCellValueFactory(new PropertyValueFactory<>("titulo"));
    autorColumn.setCellValueFactory(new PropertyValueFactory<>("autor"));
    anoColumn.setCellValueFactory(new PropertyValueFactory<>("anioPublicacion"));
    cantidadColumn.setCellValueFactory(new PropertyValueFactory<>("cantidadDisponible"));
}
```

Método para consultar libros

```java
Víctor Galiano + 1
@FXML
void consultarLibros(ActionEvent actionEvent) {
    automaticUpdatesEnabled = !automaticUpdatesEnabled;

    String titulo = tituloField.getText();
    String autor = autorField.getText();
    int anio = anioField.getText().isEmpty() ? 0 : Integer.parseInt(anioField.getText());

    List<Libro> librosEncontrados = consultarLibros(titulo, autor, anio);
    mostrarLibrosEnTableView(librosEncontrados);

    // Update the table only if automatic updates are enabled
    if (automaticUpdatesEnabled) {
        llenarTablaConLibros();
    }
}
```

Métodos para registrar libros

```java
@FXML
void registrarLibro(ActionEvent actionEvent) {
    Libro nuevoLibro = new Libro( idLibro: 0, nuevoTituloField.getText(), nuevoAutorField.getText(), nuevoAnioField.getText().isEmpty() ? 0 : Integ
    registrarLibro(nuevoLibro);
    nuevoTituloField.clear ();
    nuevoAutorField.clear ();
    nuevoAnioField.clear ();
    mostrarAlerta( titulo: "Libro Registrado", contenido: "El nuevo libro se ha registrado exitosamente.");
    llenarTablaConLibros(); // Actualizar la tabla después de registrar un nuevo libro
}
```

```java
public void registrarLibro(Libro nuevoLibro) {
    try (Connection conn = ConexionPool.obtenerConexion();
         PreparedStatement stmt = conn.prepareStatement( sql: "INSERT INTO libros (titulo, autor, anioPublicacion, cantidadDisponible) VALUES (?, ?, ?, ?

        stmt.setString( parameterIndex: 1, nuevoLibro.getTitulo());
        stmt.setString( parameterIndex: 2, nuevoLibro.getAutor());
        stmt.setInt( parameterIndex: 3, nuevoLibro.getAnioPublicacion());
        stmt.setInt( parameterIndex: 4, nuevoLibro.getCantidadDisponible());
        stmt.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Métodos para actualizar libros

```java
@FXML
void actualizarLibro(ActionEvent actionEvent) {
    // Obtén el id del libro seleccionado desde la tabla
    Libro libroSeleccionado = librosTableView.getSelectionModel().getSelectedItem();

    if (libroSeleccionado != null) {
        libroSeleccionado.tituloProperty().set(actualizarTituloField.getText());
        libroSeleccionado.autorProperty().set(nuevoAutorActualizadoField.getText());
        libroSeleccionado.cantidadDisponibleProperty().set(nuevaCantidadField.getText().isEmpty() ? 0 : Integer.parseInt(nuevaCantidadField

        actualizarLibro(libroSeleccionado);
        mostrarAlerta( titulo: "Libro Actualizado", contenido: "La información del libro se ha actualizado exitosamente.");
        llenarTablaConLibros(); // Actualizar la tabla después de actualizar un libro
    } else {
        mostrarAlerta( titulo: "Libro no Seleccionado", contenido: "Por favor, selecciona un libro de la tabla para actualizar.");
    }
}
```

```java
1 usage  Víctor Galiano
public void actualizarLibro(Libro libroActualizado) {
    try (Connection conn = ConexionPool.obtenerConexion();
         PreparedStatement stmt = conn.prepareStatement( sql: "UPDATE libros SET titulo=?, autor=?, anioPublicacion=?, cantidadDisponible=? WHERE

        stmt.setString( parameterIndex: 1, libroActualizado.getTitulo());
        stmt.setString( parameterIndex: 2, libroActualizado.getAutor());
        stmt.setInt( parameterIndex: 3, libroActualizado.getAnioPublicacion());
        stmt.setInt( parameterIndex: 4, libroActualizado.getCantidadDisponible());
        stmt.setInt( parameterIndex: 5, libroActualizado.getIdLibro());
        stmt.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

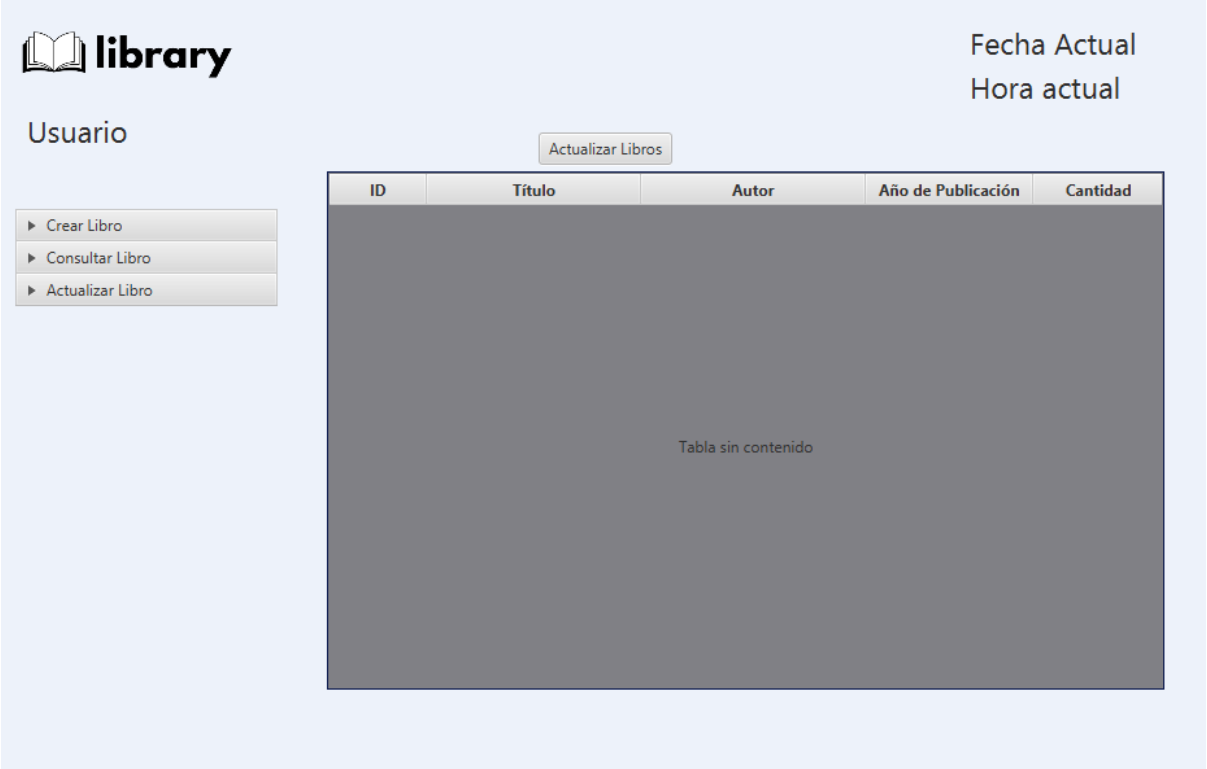
Métodos para cargar los datos de los libros

```
5 usages  Víctor Galiano
void llenarTablaConLibros() {
    List<Libro> todosLosLibros = consultarTodosLosLibros();
    ObservableList<Libro> observableLibros = FXCollections.observableArrayList(todosLosLibros);
    librosTableView.setItems(observableLibros);
}
```

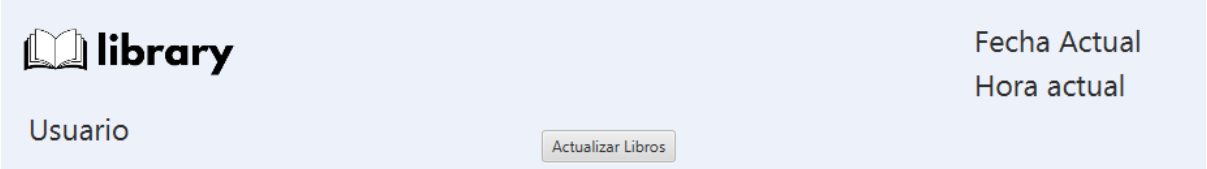
```
1 usage  Víctor Galiano
private List<Libro> consultarTodosLosLibros() {
    List<Libro> todosLosLibros = new ArrayList<>();
    try (Connection conn = ConexionPool.obtenerConexion();
         PreparedStatement stmt = conn.prepareStatement( sql: "SELECT * FROM libros");
         ResultSet rs = stmt.executeQuery()) {

        while (rs.next()) {
            Libro libro = new Libro(
                    rs.getInt( columnLabel: "idLibro"),
                    rs.getString( columnLabel: "titulo"),
                    rs.getString( columnLabel: "autor"),
                    rs.getInt( columnLabel: "anioPublicacion"),
                    rs.getInt( columnLabel: "cantidadDisponible")
            );
            todosLosLibros.add(libro);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return todosLosLibros;
}
```

Estructura de los archivos JavaFX

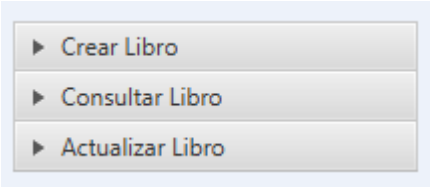
Ahora vamos a comentar el menú principal de la aplicación con todos sus componentes.
AnchorPane como contenedor principal:
Se utiliza un AnchorPane como contenedor principal con dimensiones preferidas especificadas.



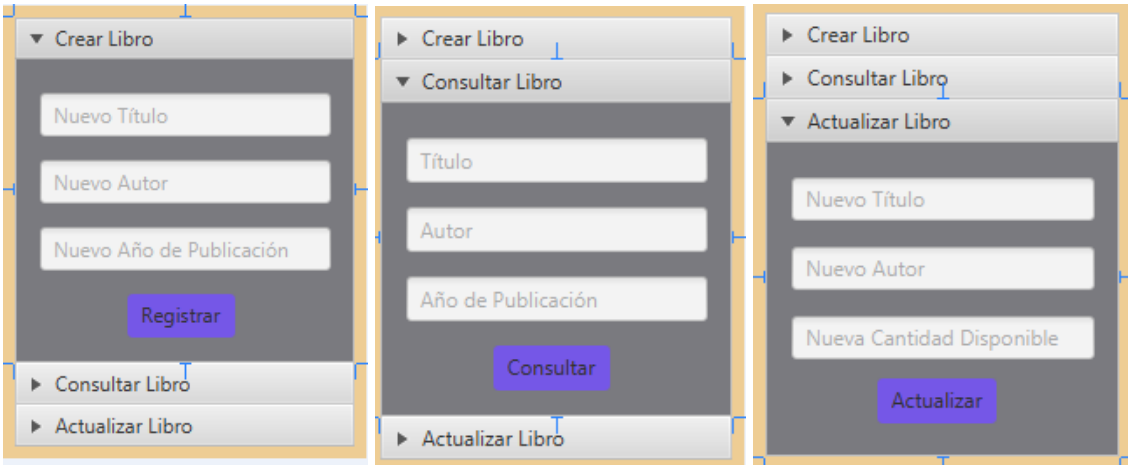
Barra superior:
Incluye un ImageView con una imagen de logo.
Tres Label para mostrar la fecha, la hora actual y el usuario conectado.
Button para actualizar la tabla de libros



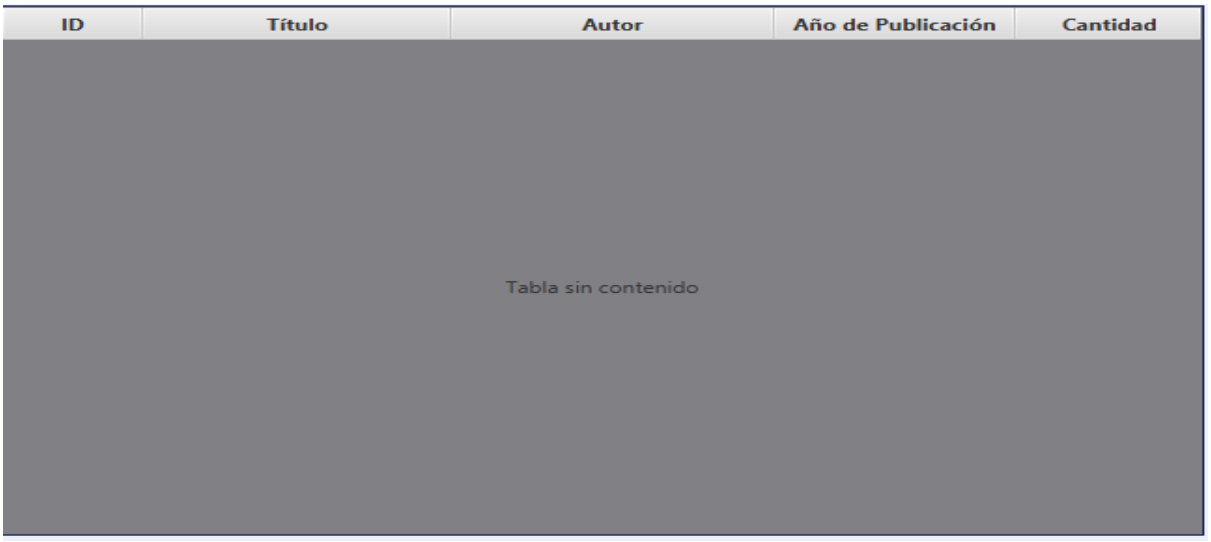
Pestañas (Accordion):
Contiene varias pestañas para diferentes secciones: "Crear libro", "Consultar Libro" y "Actualizar Libro"



Contenido de las Pestañas:
Cada pestaña tiene un AnchorPane que contiene TextFields, para rellenar datos y botones asociados a cada sección.



Columnas de las Tablas (TableView):
El TableView tiene columnas (TableColumn) que representan diferentes atributos de los elementos (por ejemplo, id, título, autor, año de publicación, cantidad).



Pool de conexiones y gestión de concurrencias

Implementamos un pool de conexiones utilizando Apache Commons DBCP (Database Connection Pooling). La arquitectura del pool de conexiones y la gestión de concurrencia se describen a continuación:

Arquitectura del Pool de Conexiones:
DataSource:
BasicDataSource es una implementación de la interfaz DataSource. Esta interfaz es parte del estándar JDBC (Java Database Connectivity) y proporciona una forma estándar de obtener conexiones a la base de datos.
La clase BasicDataSource es específica de Apache Commons DBCP y se utiliza para configurar y gestionar el pool de conexiones.

Configuración del DataSource:
Se establecen las propiedades esenciales del BasicDataSource, como la URL de la base de datos, el nombre de usuario y la contraseña.
MinIdle indica el número mínimo de conexiones inactivas que el pool debe mantener disponible.
MaxIdle indica el número máximo de conexiones inactivas que el pool puede contener.

MaxOpenPreparedStatements establece el número máximo de declaraciones preparadas (prepared statements) que pueden mantenerse abiertas simultáneamente.

Inicialización Estática:
El bloque static se ejecuta cuando la clase es cargada por primera vez.
En este bloque, se crea una instancia de BasicDataSource y se configura con los valores especificados.

Obtención de Conexiones:
El método obtenerConexion() se utiliza para obtener una conexión de la pool.
Cuando se llama a este método, se devuelve una conexión activa del pool.

Gestión de Concurrencia:

Singleton:
La clase ConexionPool sigue un patrón de diseño Singleton, ya que la instancia del BasicDataSource se crea y configura solo una vez durante la inicialización estática.

Concurrencia en la Obtención de Conexiones:
El pool de conexiones maneja automáticamente la concurrencia al proporcionar conexiones desde el pool de manera segura para su uso simultáneo por múltiples hilos.
La implementación subyacente del pool de conexiones (en este caso, BasicDataSource) se encarga de gestionar la concurrencia de manera eficiente.

Configuración de Propiedades:
Las propiedades como MinIdle, MaxIdle, y MaxOpenPreparedStatements ayudan a controlar el rendimiento y la concurrencia del pool.

En resumen, este código implementa un pool de conexiones que aprovecha Apache Commons DBCP para gestionar conexiones a una base de datos de manera eficiente y segura. La configuración del pool y la gestión de concurrencia están encapsuladas en la clase ConexionPool.

```java
package org.biblioteca.modelo;

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

import javax.sql.DataSource;
import org.apache.commons.dbcp2.BasicDataSource;

11 usages  Víctor Galiano
public class ConexionPool {
    1 usage
    private static final String URL = "jdbc:mysql://localhost:3306/biblioteca";
    1 usage
    private static final String USER = "root";
    1 usage
    private static final String PASSWORD = "root";

    8 usages
    private static BasicDataSource dataSource;

    static {
        dataSource = new BasicDataSource();
        dataSource.setUrl(URL);
        dataSource.setUsername(USER);
        dataSource.setPassword(PASSWORD);
        dataSource.setMinIdle(5);
        dataSource.setMaxIdle(10);
        dataSource.setMaxOpenPreparedStatements(100);
    }

    8 usages  Víctor Galiano
    public static Connection obtenerConexion() throws SQLException {
        return dataSource.getConnection();
    }
}
```