Usando el controlador JDBC (Java Database Connectivity)

Para usar un controlador JDBC (Java Database Connectivity) en Java, debes seguir una serie de pasos para establecer una conexión a una base de datos, ejecutar consultas y manejar resultados. A continuación, te guiaré a través del proceso paso a paso de una manera clara y práctica.

Paso 1: Agregar el Controlador JDBC a tu Proyecto

Primero, necesitas el controlador JDBC adecuado para la base de datos con la que estás trabajando (por ejemplo, MySQL, PostgreSQL, Oracle, etc.). Cada proveedor de base de datos proporciona su propio controlador.

- **Para Proyectos Maven**: Agrega la dependencia a tu `pom.xml`. Por ejemplo, para MySQL:

  ```
  <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.33</version> <!-- Verifica la última versión -->
  </dependency>
  ```

  Para PostgreSQL:

  ```
  <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.7.3</version> <!-- Verifica la última versión -->
  </dependency>
  ```

- **Para Proyectos No Maven**: Descarga el archivo JAR del controlador JDBC del sitio web del proveedor (por ejemplo, MySQL Connector/J desde mysql.com) y agrégalo manualmente al classpath de tu proyecto.

Paso 2: Cargar el Controlador JDBC

En versiones anteriores de Java (pre-JDBC 4.0), tenías que cargar el controlador explícitamente usando `Class.forName()`. Con los controladores JDBC modernos (JDBC 4.0+), este paso es opcional ya que el controlador se registra automáticamente a través del mecanismo de Proveedor de Servicios. Sin embargo, aquí está cómo funciona:

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver"); // Ejemplo de MySQL
    // Para PostgreSQL: Class.forName("org.postgresql.Driver");
```

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

Paso 3: Establecer una Conexión

Usa la clase `DriverManager` para conectarte a la base de datos proporcionando una URL de conexión, un nombre de usuario y una contraseña. El formato de la URL depende de la base de datos.

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

public class JdbcExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase"; // Ejemplo de MySQL
        String username = "your_username";
        String password = "your_password";

        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            System.out.println("Conectado a la base de datos!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

- **Ejemplos de URL**:
  - MySQL: `jdbc:mysql://localhost:3306/mydatabase?useSSL=false`
  - PostgreSQL: `jdbc:postgresql://localhost:5432/mydatabase`
  - Oracle: `jdbc:oracle:thin:@localhost:1521:xe`

Paso 4: Crear una Declaración y Ejecutar Consultas

Una vez conectado, puedes ejecutar consultas SQL usando `Statement`, `PreparedStatement` (preferido para consultas parametrizadas) o `CallableStatement` (para procedimientos almacenados).

Ejemplo con una Consulta Simple:

```java
import java.sql.*;

public class JdbcExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "your_username";
        String password = "your_password";

        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT * FROM users");

            while (resultSet.next()) {
                System.out.println("ID: " + resultSet.getInt("id") + ", Name: " + resultSet.getString("name"))
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Ejemplo con un PreparedStatement (Más Seguro para Entrada del Usuario):

```java
try (Connection connection = DriverManager.getConnection(url, username, password)) {
    String sql = "SELECT * FROM users WHERE id = ?";
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setInt(1, 5); // Establece el valor del parámetro
    ResultSet resultSet = preparedStatement.executeQuery();

    while (resultSet.next()) {
        System.out.println("ID: " + resultSet.getInt("id") + ", Name: " + resultSet.getString("name"));
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

Paso 5: Cerrar Recursos

Siempre cierra tus objetos `ResultSet`, `Statement` y `Connection` para liberar recursos. Usar try-with-resources (como se muestra arriba) maneja esto automáticamente para ti.

Ejemplo Completo Funcional (MySQL)

Aquí tienes un ejemplo completo asumiendo una tabla `users` con columnas `id` (int) y `name` (varchar):

```java
import java.sql.*;

public class JdbcExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase?useSSL=false";
        String username = "root";
        String password = "your_password";

        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            String sql = "SELECT * FROM users WHERE id = ?";
            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setInt(1, 1);
            ResultSet resultSet = preparedStatement.executeQuery();

            while (resultSet.next()) {
                System.out.println("ID: " + resultSet.getInt("id") + ", Name: " + resultSet.getString("name"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Consejos Clave

- **Manejo de Errores**: Siempre captura `SQLException` para manejar errores relacionados con la base de datos.
- **Dependencias**: Asegúrate de que el controlador JDBC coincida con la versión de tu base de datos.
- **Seguridad**: Usa `PreparedStatement` para consultas con entrada del usuario para prevenir inyección SQL.
- **Pool de Conexiones**: Para aplicaciones de producción, considera usar un pool de conexiones (por ejemplo, HikariCP) en lugar de `DriverManager`.

¡Hazme saber si necesitas ayuda con una base de datos específica o casos de uso más avanzados!