JDBC

JDBC es una API en Java que permite a las aplicaciones Java interactuar con bases de datos relacionales. Proporciona un conjunto de interfaces y clases que permiten realizar operaciones como insertar, actualizar, eliminar y consultar datos en una base de datos.

1. Componentes principales de JDBC

- **DriverManager**: Administra los controladores de base de datos que se utilizan para establecer una conexión.
- **Connection**: Representa la conexión a la base de datos.
- **Statement**: Permite enviar consultas SQL a la base de datos.
- **PreparedStatement**: Extiende Statement y permite enviar consultas precompiladas y parametrizadas.
- **ResultSet**: Contiene los resultados de una consulta SQL.
- **SQLException**: Maneja las excepciones relacionadas con la base de datos.

2. Flujo básico de trabajo con JDBC

El proceso básico con JDBC implica los siguientes pasos:

1. **Cargar el driver**: Dependiendo del tipo de base de datos (MySQL, Oracle, etc.), se carga el driver correspondiente.

   `Class.forName("com.mysql.cj.jdbc.Driver");`
2. **Establecer la conexión**: Se utiliza `DriverManager` para obtener una conexión con la base de datos.

```
    Connection conn =

DriverManager.getConnection("jdbc:mysql://localhost:3306/mi_base_de_datos",
    "usuario", "contraseña");
```

1. **Crear un Statement**: Se crea un objeto Statement o PreparedStatement para enviar la consulta SQL.

   `Statement stmt = conn.createStatement();`
2. **Ejecutar una consulta**: Dependiendo del tipo de operación, se ejecuta una consulta SELECT, INSERT, UPDATE o DELETE.

   `ResultSet rs = stmt.executeQuery("SELECT * FROM empleados");`

   Para consultas de modificación (INSERT, UPDATE, DELETE):

   `int rowsAffected = stmt.executeUpdate("INSERT INTO empleados (nombre, salario) VALUES ('Juan', 50000)");`

3. **Procesar los resultados**: Si se realiza una consulta SELECT, se obtiene el `ResultSet`, que contiene los datos devueltos por la base de datos.

```
    while (rs.next()) {
        String nombre = rs.getString("nombre");
        int salario = rs.getInt("salario");
        System.out.println(nombre + " - " + salario);
    }
```

1. **Cerrar los recursos**: Es importante cerrar las conexiones, statements y result sets una vez que se termina de trabajar con ellos.

   `rs.close(); stmt.close(); conn.close();`

3. Uso de PreparedStatement

A diferencia de `Statement`, `PreparedStatement` es más seguro y eficiente, ya que permite usar parámetros en las consultas SQL, protegiendo contra ataques de **inyección SQL**.

- **Creación de un PreparedStatement**:

```
    String sql = "SELECT * FROM empleados WHERE salario > ?";
    // Establece el valor del parámetro
    PreparedStatement pstmt = conn.prepareStatement(sql); pstmt.setInt(1,
50000);
    ResultSet rs = pstmt.executeQuery();
```

- **Uso de parámetros en INSERT, UPDATE, DELETE**:

```
    String insertSql = "INSERT INTO empleados (nombre, salario) VALUES (?,
?)";
    PreparedStatement pstmt = conn.prepareStatement(insertSql);
    pstmt.setString(1, "Ana");
    pstmt.setInt(2, 60000);
    int rowsAffected = pstmt.executeUpdate();
```

4. Transacciones en JDBC

JDBC permite controlar transacciones. Esto es útil cuando necesitas asegurarte de que varias operaciones en la base de datos se realicen de manera atómica (todo o nada).

- **Deshabilitar el auto-commit** (por defecto cada sentencia se ejecuta como una transacción independiente): `conn.setAutoCommit(false);`
- **Realizar varias operaciones**:

```
    try {
        Statement stmt = conn.createStatement();
        stmt.executeUpdate("UPDATE empleados SET salario = salario + 1000
WHERE nombre = 'Juan'");
        stmt.executeUpdate("INSERT INTO log (mensaje) VALUES ('Incremento de
salario')");
        conn.commit();  // Confirmar la transacción
    } catch (SQLException e) {
        conn.rollback();  // Revertir si ocurre un error
    }
```

- **Habilitar el auto-commit nuevamente**: `conn.setAutoCommit(true);`

5. Manejo de Excepciones

JDBC usa `SQLException` para manejar errores relacionados con la base de datos. Es importante capturar y manejar esta excepción adecuadamente.

```
try {
// Código JDBC
} catch (SQLException e) {
    System.out.println("Error de base de datos: " + e.getMessage());
    e.printStackTrace();
}
```

6. Conexión a una base de datos MySQL (ejemplo completo)

```
import java.sql.*;
public class ConexionDB {
    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            // Cargar el driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            // Establecer la conexión
            conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mi_base_de_datos",
"usuario", "contraseña");
            // Crear el Statement
            stmt = conn.createStatement();
```

```
            // Ejecutar una consulta
            String query = "SELECT * FROM empleados";
            rs = stmt.executeQuery(query);
            // Procesar los resultados
            while (rs.next()) {
                String nombre = rs.getString("nombre");
                int salario = rs.getInt("salario");
                System.out.println(nombre + " - " + salario);
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            // Cerrar los recursos
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```