SOPORTE TEMÁTICO - MÓDULO 5 SEMANA 4

Título: Java Avanzado: Persistencia, Manejo de Archivos y Arquitectura por Capas

Introducción

¡Hola, Coder!

Has recorrido un gran camino dominando los fundamentos de Java, la programación orientada a objetos y la interacción con bases de datos a través de JDBC. Esta semana, llevarás tu conocimiento al siguiente nivel con tres pilares fundamentales en el desarrollo backend profesional: **persistencia avanzada con JDBC**, **manejo de archivos** y **arquitectura por capas**.

Estos temas son esenciales para construir aplicaciones robustas, modulares y mantenibles. Al dominar estas herramientas, estarás preparado para enfrentar desafíos reales en entornos empresariales y consolidarás tu lógica de programación.

Prepárate para escribir código más limpio, estructurado y profesional.

1. JDBC Avanzado

1.1 Reutilización de código con clases DAO

En aplicaciones reales, separar el acceso a datos en clases especializadas llamadas **DAO** (Data Access Object) mejora la organización y la reutilización del código.

¿Qué es una clase DAO?

Una clase DAO encapsula las operaciones de base de datos para una entidad específica. Por ejemplo, UsuarioDAO se encarga de insertar, leer, actualizar y eliminar objetos Usuario.

Ejemplo básico:

```
public class UsuarioDAO {
   private Connection conn;

   public UsuarioDAO(Connection conn) {
      this.conn = conn;
   }

   public void insertar(Usuario usuario) throws SQLException {
      String sql = "INSERT INTO usuarios(nombre, correo) VALUES (?, ?)";
      try (PreparedStatement stmt = conn.prepareStatement(sql)) {
         stmt.setString(1, usuario.getNombre());
         stmt.setString(2, usuario.getCorreo());
         stmt.executeUpdate();
      }
   }
}
```

1.2 Uso de JOIN y consultas complejas

Cuando se trabaja con múltiples tablas, es común realizar operaciones JOIN para combinar información relacionada.

Ejemplo: obtener usuarios junto con su rol:

```
SELECT u.nombre, r.nombre AS rol
FROM usuarios u
JOIN roles r ON u.rol_id = r.id;
```

Se ejecuta con PreparedStatement y ResultSet, igual que las consultas simples.

1.3 Transacciones en JDBC

Las transacciones permiten agrupar varias operaciones en una sola unidad que se ejecuta completamente o no se ejecuta.

```
try {
   conn.setAutoCommit(false);

   // múltiples operaciones
   usuarioDAO.insertar(usuario);
   logDAO.registrar("Usuario insertado");

   conn.commit(); // guardar cambios
} catch (Exception e) {
   conn.rollback(); // deshacer todo
}
```

1.4 Buenas prácticas con JDBC

- Usa try-with-resources para cerrar conexiones automáticamente.
- Separa la lógica de acceso a datos en DAOs.
- Nunca mezcles lógica de negocio con SQL directamente.

2. Manejo de Archivos en Java

2.1 Escritura de archivos

Guardar información en archivos de texto es útil para reportes, backups o sistemas sin base de datos.

```
PrintWriter writer = new PrintWriter("usuarios.txt");
writer.println("Nombre: Kevin Mejía");
```

writer.close();

2.2 Lectura de archivos

```
BufferedReader reader = new BufferedReader(new FileReader("usuarios.txt"));
String linea;
while ((linea = reader.readLine()) != null) {
   System.out.println(linea);
}
reader.close();
```

2.3 Serialización de objetos

Permite guardar objetos completos en archivos binarios.

```
ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("usuario.ser"));
out.writeObject(usuario);
out.close();
```

Y para leerlo:

```
ObjectInputStream in = new ObjectInputStream(new
FileInputStream("usuario.ser"));
Usuario usuario = (Usuario) in.readObject();
in.close();
```

2.4 Casos de uso comunes

- Backup temporal de datos
- Exportación de reportes
- Persistencia ligera en apps pequeñas

3. Arquitectura por Capas

3.1 ¿Qué es una arquitectura en capas?

Es una forma de organizar el código en módulos separados por responsabilidad, lo que permite trabajar de forma ordenada y desacoplada.

Las capas principales son:

3.1.1 Modelo

Clases que representan los datos (ej. Usuario, Producto).

3.1.2 DAO

Clases que interactúan con la base de datos.

3.1.3 Servicio (Service)

Contiene la lógica de negocio, validaciones y reglas.

3.1.4 Presentación (consola)

Recoge datos del usuario y muestra resultados.

3.2 Ejemplo de estructura:

```
src/
├── modelo/
│   └── Usuario.java
├── dao/
```

```
│   └── UsuarioDAO.java
├── servicio/
│   └── UsuarioService.java
├── presentacion/
│   └── App.java
```

3.3 Beneficios de esta arquitectura

- Reutilización de componentes
- Separación clara de responsabilidades
- Escalabilidad del proyecto

3.4 Caso práctico

Crear una aplicación que:

1. Solicite datos del usuario desde consola.
2. Guarde los datos en un archivo .txt.
3. Inserte los datos también en la base de datos.
4. Tenga validaciones de negocio en la capa servicio.

Conclusiones

Esta semana profundizaste en herramientas clave para el desarrollo backend con Java. Aprendiste a organizar tu aplicación en capas, a trabajar con archivos como alternativa o complemento a la base de datos, y a mejorar tus habilidades con JDBC de forma profesional.

Puntos clave:

- El patrón DAO permite encapsular el acceso a datos y mejorar la mantenibilidad.
- Las transacciones aseguran que varias operaciones se ejecuten de forma segura.
- Leer y escribir archivos permite persistir datos localmente o hacer reportes.
- La arquitectura por capas organiza tu aplicación y facilita el trabajo en equipo.

Bibliografía

Baeldung. (2023). *Guide to JDBC with MySQL*. https://www.baeldung.com/java-jdbc

Oracle. (n.d.). *Java Platform, Standard Edition Documentation*. https://docs.oracle.com/javase/8/docs/

Deitel, P. J., & Deitel, H. M. (2017). *Java: How to Program (11th ed.)*. Pearson Education.

Schildt, H. (2018). *Java: The Complete Reference (11th ed.)*. McGraw-Hill Education.