MVC (Model-View-Controller)

Mi aplicación implementa el patrón **MVC** de manera clara y estructurada. He dividido el código en tres capas principales:

- **Modelo**: El Modelo lo forman las clases Empleado, NominaService, y EmpleadoDAO. El Modelo es donde se encuentra toda la lógica de negocio y acceso a los datos. Empleado es una representación del empleado, NominaService maneja la lógica del cálculo de salarios, y EmpleadoDAO se encarga de acceder a la base de datos para obtener y procesar datos de los empleados.
- **Vista**: Las vistas están representadas por las páginas JSP (como listar.jsp, salario.jsp, error.jsp, etc.).
- **Controlador**: El Controlador está representado por la clase EmpleadoController. Este controlador recibe las solicitudes del usuario (a través de los parámetros opción), decide qué acción tomar (listar empleados, mostrar salario, editar un empleado) y luego interactúa con el Modelo para obtener los datos necesarios. Finalmente, pasa esos datos a las Vistas para que se presenten al usuario.

DAO (Data Access Object)

Tengo la clase EmpleadoDAO, que se encarga de todas las interacciones con la base de datos. Los métodos como obtenerEmpleado(), buscarEmpleadosPorDNI(), buscarEmpleadosPorCriterios(), y actualizarEmpleado() encapsulan toda la lógica de acceso a los datos.

Singleton

Implementé el patrón Singleton en la clase MDBConection. Aquí se centraliza la creación de la base de datos BasicDataSource, que gestiona las conexiones a la base de datos. De esta manera, solo existe una instancia de la base de datos en toda la aplicación.

Factory

El patrón Factory lo implemento en MDBConection, que actúa como una fábrica para crear conexiones a la base de datos. El método getConnection() devuelve una conexión preconfigurada sin necesidad de que otras clases tengan que preocuparse por cómo se configura o se obtiene la conexión.

Strategy (Parcialmente Implementado)

Lo he implementado parcialmente con la forma en que gestiono las búsquedas de empleados. Por ejemplo, tengo métodos como buscarEmpleadosPorDNI() y buscarEmpleadosPorCriterios(), que son como diferentes "estrategias" de búsqueda, es decir, dependiendo del parámetro que reciba el controlador (dni, criterio),se ejecutará un método u otro.

Observer (Implícito en la gestión de vistas)

Aunque no es un Observer formal con un Subject y Observer, la aplicación sigue una forma base de este patrón en la manera en que las vistas reaccionan a las solicitudes del controlador. Cuando el controlador recibe una acción del usuario (por ejemplo, hacer clic en "listar empleados"), el controlador "notifica" a la vista de que debe mostrar la lista de empleados.

Command (Implícito en las acciones del controlador)

El patrón Command se ve de manera básica en la forma en que manejo las diferentes "acciones" en el controlador EmpleadoController. Dependiendo de la opción seleccionada (listar, salario, editar, etc.), el controlador ejecuta un bloque de código diferente.