ESTRUCTURA Y CREACION DE CARPETAS Y CLASES E INTERFACES EN JAVA-SPRINGBOOT

Resumen del Orden

1. Modelo (Entidad): Define las tablas de la base de datos.
2. Repositorio: Define la interfaz para acceder a la base de datos.
3. DTOs: Define los objetos para transferir datos.
4. Servicio: Define la lógica de negocio.
5. Controlador: Expone los endpoints de la API.
6. Excepciones: Maneja errores específicos.
7. Configuración: Configura beans personalizados o seguridad.
8. Pruebas: Escribe pruebas para validar el funcionamiento.

1º MODEL (objeto como por ejemplo 1 pelicula):

Propósito: Representa una entidad del dominio de tu aplicación, normalmente mapeada a una tabla en la base de datos.

Uso común: En la capa de persistencia con JPA/Hibernate (@Entity).

- El modelo define la estructura de la tabla que iran los datos en su bbdd cada atributo es una columna de esa tabla en la base de datos
- Cada instancia del modelo representa una fila en la tabla.
- Las filas se añaden dinámicamente al insertar datos en la base de datos.

Ventajas:

⑩ Refleja directamente la estructura de tus datos.

- Incluye relaciones (@OneToMany, @ManyToOne, etc.).

```java
import jakarta.persistence.*;

@Entity
@Table(name = "peliculas") // Nombre de la tabla en la base de datos
public class Pelicula {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Autoincremental
    private Long id;

    @Column(nullable = false)
    private String titulo;

    @Column(nullable = false)
    private String genero;

    @Column(name = "duracion_min") // Se llama diferente en la base de datos
    private int duracion;

    @Column(name = "director", length = 100)
    private String director;

    // Getters y Setters
```

2º REPOSITORY : (Armario para guardar) Es el que hace la persistencia en la base de datos es decir :

⓾ Crea operaciones CRUD : guardar,listar, eliminar , actualizar datos (si realizar codigo SQL)

¿Como lo hace ?

- Definiendo una interfaz que extiende de JPARepository y Spring se encarga de generar automáticamente la implementación en tiempo de ejecución.

Ejemplo 1 Listar las peliculas por genero (METODO GET) :

1º Linea de codigo : Indica la anotacion que es un repositorio (OPERADOR CRUD)

2º Linea de codigo : Aqui el interfaz indica que vamos a trabajar con la entidad pelicula y su clave primaria es de tipo Long

3º Linea de codigo → El método findByGenero lo interpreta Spring Data JPA automáticamente y genera la consulta SQL equivalente para buscar películas por género

```
@Repository
public interface PeliculaRepository extends JpaRepository<Pelicula, Long> {
    // Puedes definir métodos personalizados con nombres especiales
    List<Pelicula> findByGenero(String genero);
}
```

Ejemplo 2 :

Aqui tenemos mas ejemplos por ejemplo de arriba a abajo

- Listar todas las peliculas → METODO GET
- Guardar pelicula o Actualizar melicula → METODO POST (GUARDAR) METODO PUT(ACTUALIZAR)
- Elimina una pelicula por su numero de id → METODO DELETE

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface PeliculaRepository extends JpaRepository<Pelicula, Long> {

    // Método para obtener todas las películas
    @Override
    List<Pelicula> findAll(); // SELECT * FROM pelicula
objeto generico
    // Método para guardar o actualizar una película
    @Override
    <S extends Pelicula> S save(S entity); // INSERT o UPDATE según si tiene ID

    // Método para eliminar una película por su ID
    @Override
    void deleteById(Long id); // DELETE FROM pelicula WHERE id = ?
```

SI USAS ESTOS METODOS NO ES NECESARIO QUE LOS ESCRIBAS

Método	¿Qué hace?	Para qué sirve
`save(entity)`	Guarda o actualiza una entidad	Crear/Actualizar
`findById(id)`	Busca por ID	Leer uno
`findAll()`	Devuelve todas las entidades	Leer todos
`delete(entity)`	Elimina una entidad	Borrar
`deleteById(id)`	Elimina por ID	Borrar
`existsById(id)`	Devuelve `true` si existe ese ID	Comprobación

- Estos metodos ya existen en JPA y los reconoce dejarias entonces el repository solo con la extension

```
public interface PeliculaRepository extends JpaRepository<Pelicula, Long> {
    // No hace falta añadir métodos si usas los básicos
}
```

3º @SERVICE : (REGLAS PARA PODER ACCEDER A EL ARMARIO)

Propósito: Pones las reglas para poder acceder a los metodos del armario (logica progamacion)

Ejemplo 1 Validar datos antes de guardar (Para guardar una pelicula tiene que tener titulo y duracion) y que se guarde la pelicula en minusculas

```java
@Service
public class PeliculaService {

    private final PeliculaRepository peliculaRepository;

    public PeliculaService(PeliculaRepository peliculaRepository) {
        this.peliculaRepository = peliculaRepository;
    }

    public Pelicula guardarPelicula(Pelicula pelicula) {
        // Validación del título
        if (!StringUtils.hasText(pelicula.getTitulo())) {
            throw new IllegalArgumentException("El título no puede estar vacío");
        }

        // Validación del tiempo
        if (pelicula.getTiempo() <= 0) {
            throw new IllegalArgumentException("El tiempo debe ser mayor que cero");
        }

        // Convertir el título a minúsculas antes de guardar
        pelicula.setTitulo(pelicula.getTitulo().toLowerCase());

        // Guardar la película
        return peliculaRepository.save(pelicula);
    }
```

- Usar un DTO si no quieres mostrar los datos de la entidad (haces 1º un dto con los campos de la entidad que quieres exponer y los mapeas despues en el servicio)
- Ejemplo de creacion de un DTO (Se muestra todo menos el id de entidad)

```java
package com.tuapp.dto;  // Asegúrate de que el paquete coincida con tu e

import java.time.LocalDate;

public class PeliculaDTO {
    // Campos expuestos (sin el ID)
    private String titulo;
    private int duracionMinutos;
    private String genero;



    // Constructor vacío (necesario para frameworks)
    public PeliculaDTO() {
    }

    // Constructor con campos
    public PeliculaDTO(String titulo, int duracionMinutos, String genero
o, boolean estaActiva) {
        this.titulo = titulo;
        this.duracionMinutos = duracionMinutos;
        this.genero = genero;


    }

    // Getters y Setters (sin ID)
```

1º DEBES CREAR LA CARPETA MAPPER Y EL ARCHIVO MAPPER

Ejemplo 1 : Si tienes un dto y tienes que mapear entonces harias

Aqui el DTO tiene los mismos campos que la entidad

```java
package com.tuapp.mapper;  // Asegúrate de que el paquete coincida con tu estructura

// Importaciones OBLIGATORIAS para MapStruct
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;

// Importaciones de tus clases
import com.tuapp.model.Pelicula;    // Tu entidad
import com.tuapp.dto.PeliculaDTO;  // Tu DTO
```

En este ejemplo el DTO no tiene id

2º

```java
// PeliculaMapper.java
package com.tuapp.mapper;

import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import com.tuapp.model.Pelicula;  // Entidad
import com.tuapp.dto.PeliculaDTO; // DTO sin ID

@Mapper(componentModel = "spring")
public interface PeliculaMapper {

    // Entity → DTO (ignora id y tiempo)
    @Mapping(target = "id", ignore = true)     // Ignoramos el ID
    @Mapping(target = "tiempo", ignore = true) // Ignoramos tiempo
    PeliculaDTO toDto(Pelicula pelicula);

    // DTO → Entity (asigna valores por defecto)
    @Mapping(target = "id", ignore = true)      // El ID lo genera la BD
    @Mapping(target = "tiempo", constant = "8") // Valor por defecto
    @Mapping(target = "fechaCreacion", expression = "java(java.time.LocalDate.now())")
    Pelicula toEntity(PeliculaDTO peliculaDTO);
}
```

VOLVERIAS AL SERVICIO Y HARIAS EL MAPPEO O DE DTO A MAPPER O AL REVES O AMBOS

```java
@Service
public class PeliculaService {

    @Autowired
    private PeliculaRepository repo;

    @Autowired
    private PeliculaMapper mapper; // El "traductor"

    // GUARDAR: Convierte DTO → Entity y guarda
    public PeliculaDTO guardarPelicula(PeliculaDTO dto) {
        // Paso 1: Conversión (DTO → Entity)
        Pelicula pelicula = mapper.toEntity(dto);
        // Aquí MapStruct automáticamente pone tiempo=8

        // Validaciones
        if(pelicula.getTitulo() == null) {
            throw new RuntimeException("El título es obligatorio");
        }

        // Paso 2: Guardar (Repository trabaja con Entity)
        Pelicula guardada = repo.save(pelicula);

        // Paso 3: Conversión (Entity → DTO para respuesta)
        return mapper.toDto(guardada);
        // Aquí MapStruct omite el campo tiempo
    }

    // MOSTRAR: Convierte Entity → DTO
    public PeliculaDTO obtenerPelicula(Long id) {
        Pelicula pelicula = repo.findById(id).orElseThrow();
        return mapper.toDto(pelicula); // Entity → DTO
    }
}
```

4º @RESTCONTROLLER (el mesero , el chofer, el trabajador , el que lleva los endpoints)

⑩ El que toma pedidos y lleva los platos

```java
@RestController
@RequestMapping("/peliculas") // Ruta como "/peliculas"
public class PeliculaController {

    // El mesero necesita al chef (Service)
    private final PeliculaService peliculaService;

    public PeliculaController(PeliculaService peliculaService) {
        this.peliculaService = peliculaService;
    }

    // POST → Como hacer un pedido
    @PostMapping
    public ResponseEntity<String> crearPelicula(@RequestBody PeliculaDTO peliculaDTO) {
        try {
            PeliculaDTO nueva = peliculaService.guardarPelicula(peliculaDTO);
            return ResponseEntity.ok("Película creada: " + nueva.getTitulo());
        } catch (Exception e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    // GET → Como pedir una película
    @GetMapping("/{id}")
    public ResponseEntity<PeliculaDTO> obtenerPelicula(@PathVariable Long id) {
        try {
            PeliculaDTO pelicula = peliculaService.obtenerPelicula(id);
            return ResponseEntity.ok(pelicula);
        } catch (Exception e) {
            return ResponseEntity.notFound().build();
        }
    }
}
```

USAREMOS POSTMAN PARA VER SI FUNCIONAN LAS PETICIONES Y QUE RESPONDE

Pruebas con Postman (Como pedir a domicilio)

. Crear Película (POST)

- **URL:** `http://localhost:8080/peliculas`
- **Método:** POST
- **Headers:**

```
Content-Type: application/json
```

- **Body (raw JSON):**

```json
{
    "titulo": "Toy Story",
    "duracion": 81
}
```

2. Ver Película (GET)

- **URL**: `http://localhost:8080/peliculas/1`
- **Método**: GET

🔄 Flujo Completo (Ejemplo Real)

1. **Tú envías** (desde Postman o app):

```json
{"titulo": "Toy Story", "duracion": 81}
```

2. **El Controller (mesero)** recibe el pedido y llama al Service (chef)
3. **El Service (chef)**:
   - Convierte el menú (DTO) a receta (Entity)
   - Guarda en la cocina (base de datos)
   - Prepara la respuesta (convierte Entity→DTO)
4. **Respuesta** que recibes:

```json
{
    "titulo": "toy story",  // Minúsculas (por tu regla)
    "duracion": 81
}
```

📌 Errores Comunes (y cómo evitarlos)

1. **Olvidar el** `@RestController` → Sin esto, no es un controller
2. **No poner** `@PostMapping` **o** `@GetMapping` → No sabrá qué método usar
3. **Enviar JSON mal formado** → Postman te ayuda con el formato