

# SPRING FRAMEWORK

## Manejo de Excepciones en los Servicios

El **manejo de excepciones** en los servicios es crucial para garantizar la estabilidad y confiabilidad de una aplicación. Anticipar y gestionar errores desde el inicio del desarrollo ayuda a evitar fallos que podrían afectar la experiencia del usuario.

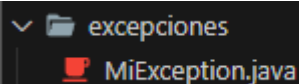
Uno de los problemas más comunes es el manejo de **datos incorrectos o nulos** que pueden surgir durante la ejecución del programa. Para abordar estos casos, es recomendable implementar **excepciones personalizadas** que reflejen errores específicos de la lógica de negocio.

### 1. Creación de una excepción personalizada

Es recomendable definir una **clase de excepción** específica dentro de un paquete dedicado a excepciones dentro del proyecto.

Ejemplo de una excepción personalizada:

```
public class MiException extends Exception{  
    public MiException(String msg) {  
        super(msg);  
    }  
}
```



Esta clase extiende `Exception`, lo que permite lanzar errores controlados con mensajes descriptivos.

### 2. Uso de la excepción personalizada en los servicios

Dentro de los métodos del servicio, puedes lanzar esta excepción cuando detectes situaciones que requieran una validación especial.

Ejemplo de validación de datos:

```
private void validar(String nombre) throws MiException {  
    if (nombre.isEmpty() || nombre == null) {  
        throw new MiException("el nombre de la editorial no puede ser nulo  
o estar vacío");  
    }  
}
```

En este caso, si el nombre proporcionado es `null` o está vacío, se lanza una `MiException` con un mensaje descriptivo.

### 3. Aplicación de la validación en un método del servicio

El siguiente método se encarga de **crear una nueva editorial**, asegurando que los datos sean válidos antes de proceder con la persistencia en la base de datos.

```
@Transactional
public void crearEditorial(String nombre) throws MiException{

    validar(nombre);
    Editorial editorial = new Editorial();
    editorial.setNombre(nombre);
    editorialRepositorio.save(editorial);
}
```

#### Beneficios de este enfoque

- ✓ **Diferenciación de errores:** Se pueden distinguir los errores específicos de la lógica de negocio de los errores generales del sistema.
- ✓ **Mensajes de error descriptivos:** Facilitan la identificación y resolución de problemas.
- ✓ **Código más limpio y organizado:** La validación se separa de la lógica de negocio, mejorando la mantenibilidad.

Este enfoque permite un manejo estructurado de errores en los servicios, lo que mejora la **robustez y estabilidad** de la aplicación.