



# Manejo de Excepciones

## REPASO Excepciones

### Excepciones

- Una excepción es un evento que ocurre durante la ejecución de un programa que rompe el flujo normal de ejecución.
- Muchas cosas pueden generar excepciones: un error en algún elemento de hardware, operaciones (por ejemplo dividir por cero), errores generales de un programa (error por desbordamiento de un arreglo), apertura de archivo inexistente, etc.



Exception

### Tipos de excepciones:

- Propias de Java
- Personalizadas

### Excepciones propias de Java:

- RuntimeException: representa las excepciones que ocurren dentro de la máquina virtual Java (durante el tiempo de ejecución). Un ejemplo de estas excepciones es NullPointerException.

- **IOException:** Significa que se ha producido un error en la entrada/salida. Por ejemplo, cuando estamos leyendo de la consola, un fichero, etc. Es obligatorio tratar la excepción, ya sea en la cabeza del método con "throws IOException" o con un bloque try/catch.

#### “Otras notaciones”:

- **Finally:** Es un bloque de código que es opcional, sin embargo, en caso de que esté, se ejecuta siempre (sin importar si hubieron errores o no) con la finalidad de brindar un determinado mensaje luego de la declaración de la excepción.
- **Throw:** Nos permite lanzar una excepción (Cualquiera que tengamos).
- **Throws:** Nos permite decidir o determinar qué excepciones puede/debe lanzar un método en particular.

## Status Code

Status Code más usados		
HTTP Status Codes		
<b>Level 200 (Success)</b> 200 : OK 201 : Created 203 : Non-Authoritative Information 204 : No Content	<b>Level 400</b> 400 : Bad Request 401 : Unauthorized 403 : Forbidden 404 : Not Found 409 : Conflict	<b>Level 500</b> 500 : Internal Server Error 503 : Service Unavailable 501 : Not Implemented 504 : Gateway Timeout 599 : Network timeout 502 : Bad Gateway

# Manejo de excepciones con Spring Boot.

## @ResponseStatus:

- El manejo de excepciones es una parte importante de la construcción de una aplicación sólida
- Spring Boot ofrece más de una forma de hacerlo. Nos proporciona herramientas para manejar excepciones más allá de los bloques "try-catch":

## @ResponseStatus:

vincula una excepción particular a un estado de respuesta HTTP específico. Entonces, cuando Spring detecta esa excepción en particular, genera una respuesta HTTP con la configuración definida en @ResponseStatus.

```
@ResponseStatus (value = HttpStatus.NOT_FOUND, reason = "No existe tal libro")
public class BookNotFoundException extends RuntimeException{

    public BookNotFoundException (String message){
        super(message);
    }
}
```



## Manejo de errores con Spring Boot:

## @ResponseStatus

- En este ejemplo podemos ver una respuesta de Status 200 (OK), sin embargo, Spring nos ofrece un gran número de `HttpStatus` con todos los posibles códigos HTTP de retorno.

```
@ResponseStatus(value=HttpStatus.OK)
public String getInfo(){
    ...
}
```

- `@ResponseStatus` también permite establecer la razón del código devuelto. Este valor es un String que puede pasarse como segundo parámetro

```
@ResponseStatus(value=HttpStatus.OK, reason="Everything works fine.")
public String getInfo(){
    ...
}
```

## @ExceptionHandler

- Cuando hacemos operaciones **sin errores**, recibimos un **Status Code 200 (Success)**.
- Con Spring podemos realizar diferentes manejos de excepciones con la finalidad de poder detectar algún error o solicitud no deseada y responder el status que corresponda.
- Para ello Spring Boot se vale del `@ExceptionHandler` en el cual especificamos el tipo de excepción que vamos a tratar y de qué manera.
- Este tipo de excepción puede ser uno propio (creado por nosotros) o uno por defecto de Java.

```
@ExceptionHandler(NullPointerException.class)
public void nullPointerException(){
    logger.log(Level.ERROR, "NullPointerException!!!");
}
```

## @ControllerAdvice

- La anotación **@ControllerAdvice** nos permite consolidar nuestros múltiples **@ExceptionHandler**s dispersos de antes, en un solo componente global de manejo de errores. Nos da un control total sobre el cuerpo de la respuesta y sobre el código de estado.

**Ejemplo:** Podemos crear una clase anotada con **@ControllerAdvice** que maneje dos tipos de excepciones que nosotros hayamos creado anteriormente y que hayamos marcado con **@ExceptionHandler** (NotFoundException y BadRequestException).



```
@ControllerAdvice(annotations = RestController.class)
public class ExceptionConfig {

    @ExceptionHandler(NotFoundException.class)
    public ResponseEntity<?> notFoundException(Exception e){
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
    }

    @ExceptionHandler(BadRequestException.class)
    public ResponseEntity<?> badRequestException(Exception e){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
    }
}
```

## ¿Cómo sabemos qué usar y en qué momento?

- Para excepciones personalizadas, tenemos que considerar **@ResponseStatus**.
- Para el manejo de excepciones específicas del **controlador**, agregar los métodos **@ExceptionHandler** a la capa mencionada.
- Para todas las demás excepciones, implementar un método **@ExceptionHandler** en una clase **@ControllerAdvice**.