03-03-2021

Spring Platform

IT BOARDING

BOOTCAMP

WIP

Índice

IT BOARDING

BOOTCAMP

TEMA

// Excepciones

IT BOARDING

BOOTCAMP

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

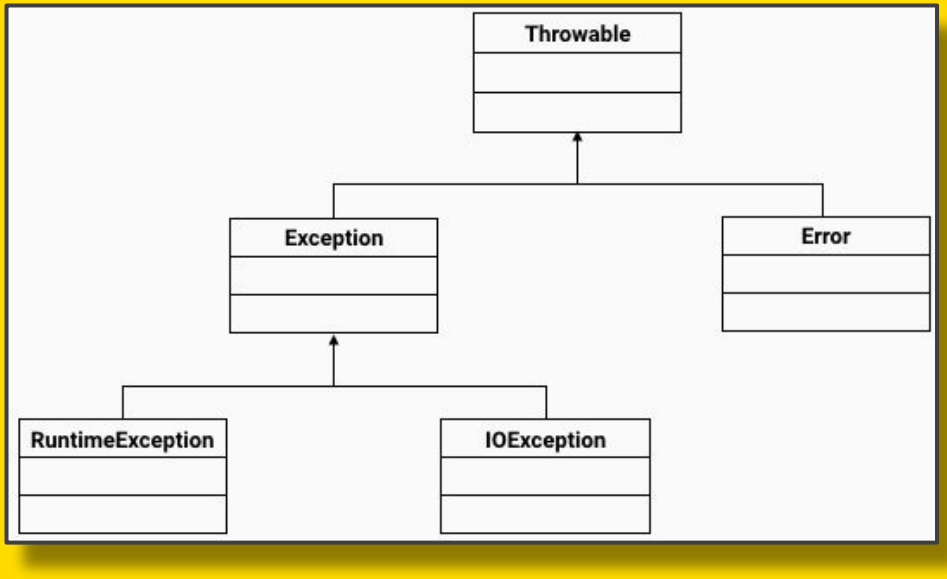
- Propias de Java
- Personalizadas

Exception

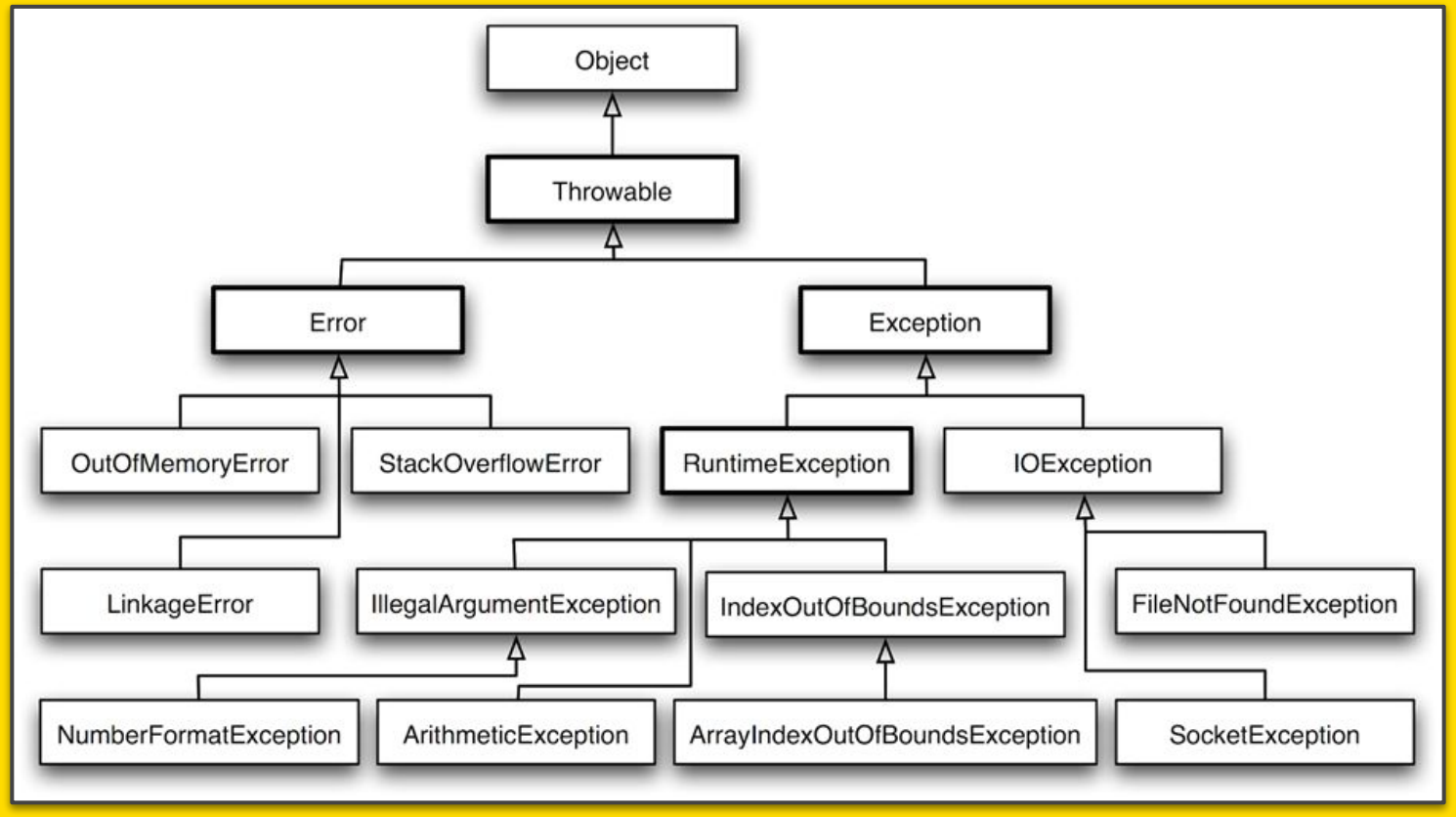
Excepciones Propias de Java

- **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.



Algunos tipos de excepciones



Captura de Excepciones

- En Java para capturar las excepciones que se hayan podido producir utilizamos las expresiones “try” y “catch” para delimitar el código que queremos controlar.

- Cuando se produce una excepción, el bloque “try” termina y “catch” muestra la excepción.

```
try {
    total = 53/0;
}
catch (Exception e) {
    System.out.println("¡No se puede dividir por cero!");
}
```

“Otras notaciones”

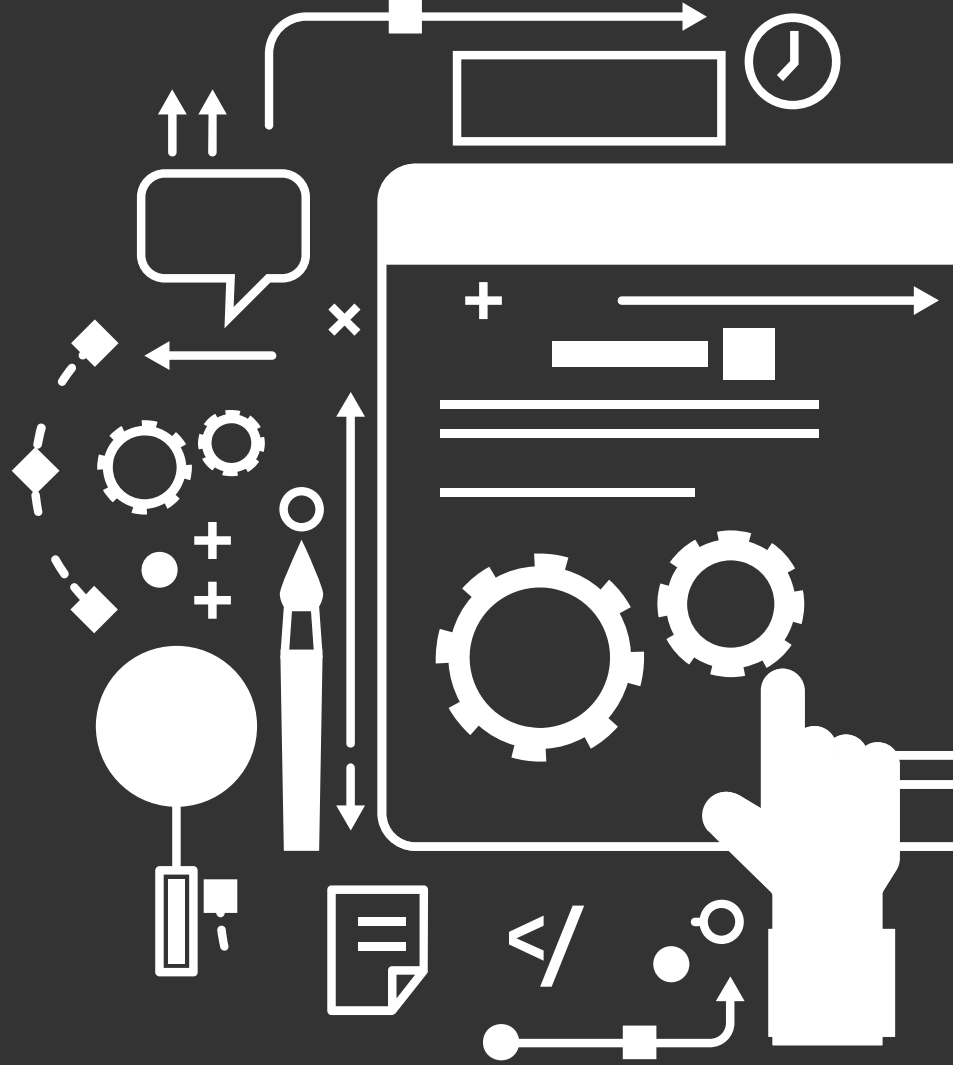
- **Finally:** Es un bloque de código que es opcional, sin embargo, en caso de que esté, se ejecuta siempre (sin importar si hubieron errores o no) con la finalidad de brindar un determinado mensaje luego de la declaración de la excepción.

- **Throw:** Nos permite lanzar una excepción (Cualquiera que tengamos).

- **Throws:** Nos permite decidir o determinar qué excepciones puede/debe lanzar un método en particular.

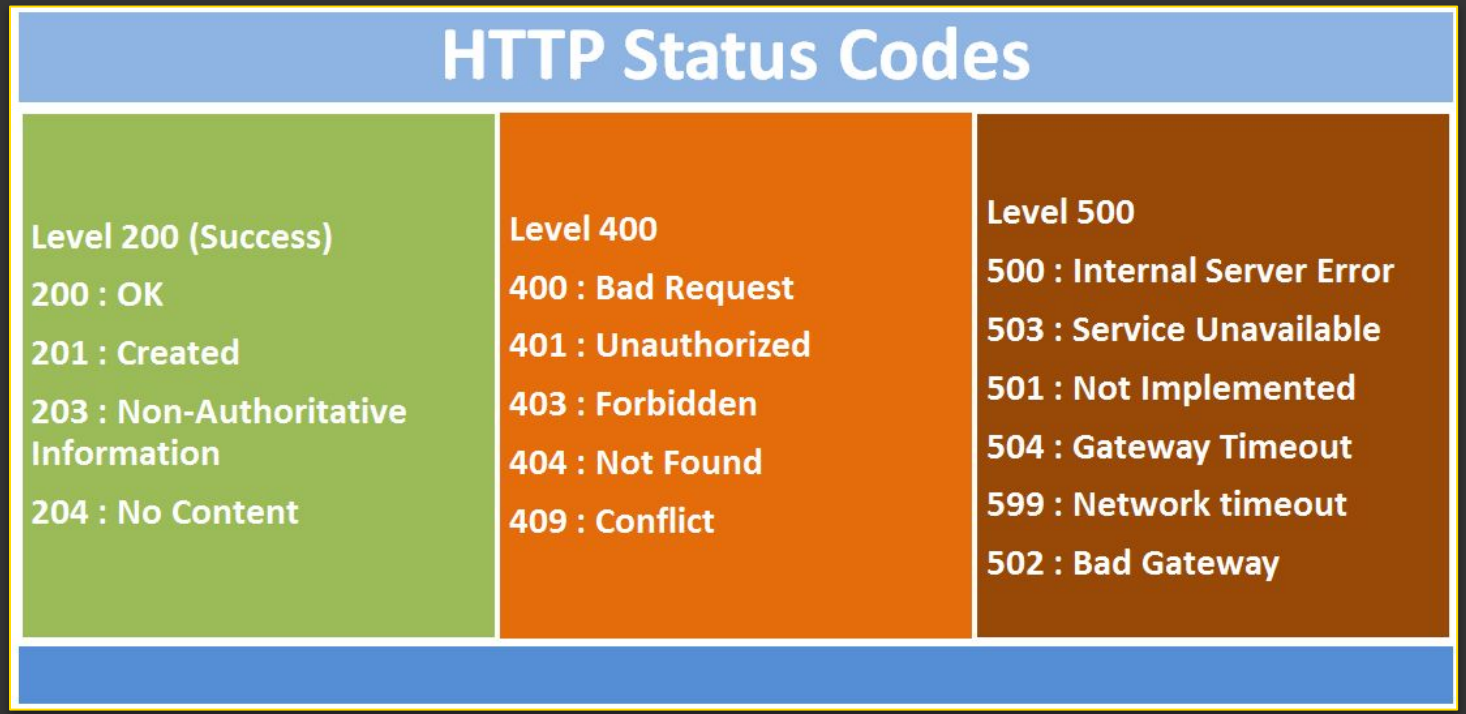
Status Code & Manejo de Errores

IT BOARDING
BOOTCAMP



Repaso

STATUS CODES más usados:



Manejo de errores en Spring Boot

- 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 nullPointerHandler(){
    logger.log(Level.ERROR, "NullPointerException!!!");
}
```

Ejemplo práctico

- Podemos modificar la calculadora de calorías para que cuando no encuentre un ingrediente retorne 400: Bad Request con un título y un mensaje de error.

- Para esto vamos a crear un tipo de excepción propia que sea “IngredientNotFound” y una clase DTO (ErrorDTO) para ingresar un nombre de error y un mensaje a mostrar.

- Luego mediante ResponseEntity, podemos devolver el mensaje de la excepción que queramos en conjunto con el status code correspondiente (en este caso por ejemplo, Bad Request).

```java
@ExceptionHandler(IngredientNotFound.class)
public ResponseEntity<ErrorDTO> handleException(IngredientNotFound errorException) {
  ErrorDTO errorDTO = new ErrorDTO();
  errorDTO.setName("Invalid Ingredient");
  errorDTO.setDescription("The ingredient " + errorException.getMessage() + " is invalid");
  return new ResponseEntity<>(errorDTO, HttpStatus.BAD_REQUEST);
}
```

Manejo de errores en Spring Boot

@ResponseStatus

- La anotación @ResponseStatus nos permite marcar un método o excepción con el código de estado http (code) y la razón (reason) que serán devueltos.

- 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 de vuelto. Este valor es un string que puede pasarse como segundo parámetro

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

Otros Ejemplos Prácticos

```java
@ResponseStatus(value=HttpStatus.NOT_FOUND)
@ExceptionHandler(NullPointerException.class)
public void nullPointerHandler(){
    logger.log(Level.ERROR, "NullPointerException!!!");
}

@ResponseStatus(value=HttpStatus.BAD_REQUEST)
@ExceptionHandler(NumberFormatException.class)
public void numberFormatHandler(){
    logger.log(Level.ERROR, "NumberFormatException!!!");
}
```

Práctica Integradora

IT BOARDING

BOOTCAMP