PROYECTO “exception-demo”

Excepciones Checked (Checked Exceptions)

Son verificadas en tiempo de compilación.

- El compilador obliga a manejarlas con try-catch o throws.
- **Ejemplo:** IOException, SQLException, FileNotFoundException.

Excepciones Unchecked (Unchecked Exceptions)

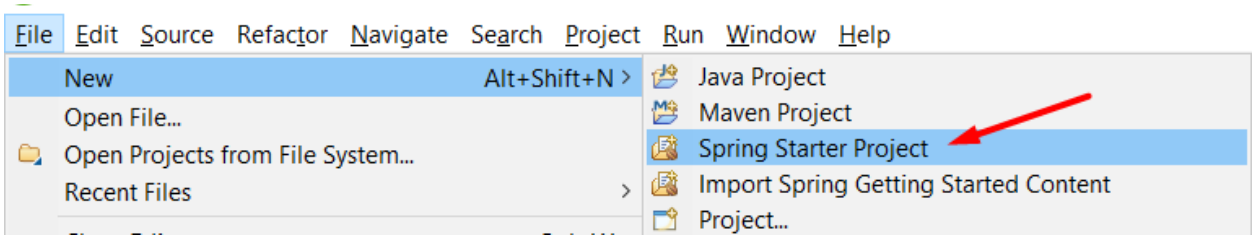
No son verificadas en tiempo de compilación.

- El compilador NO obliga a manejarlas con try-catch o throws.
- **Ejemplo:** NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException.

Paso 1: Crear el Proyecto en STS

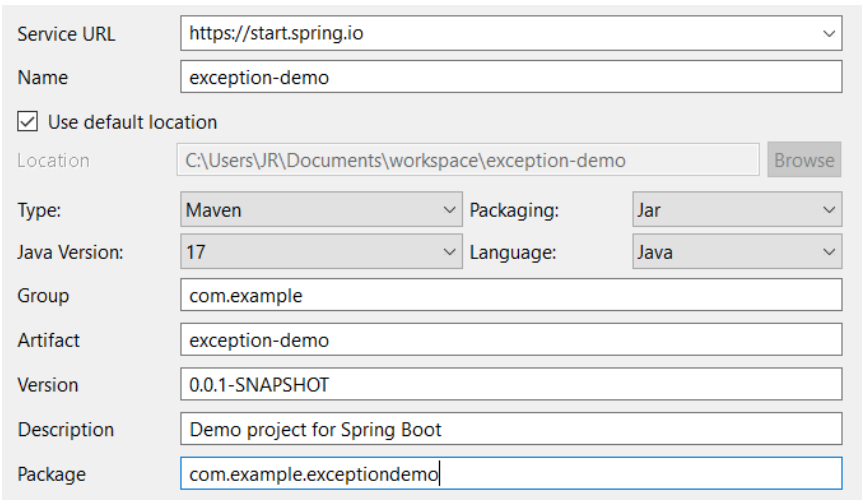
Abrir Spring Tool Suite (STS).

Ir a **File → New → Spring Starter Project**.

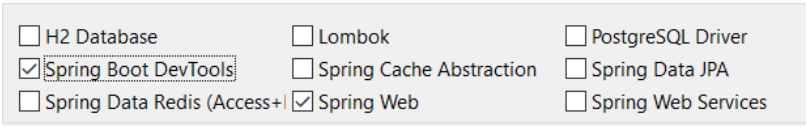


Completar los datos del proyecto:

- **Name:** redis-session-demo
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** redis-session-demo
- **Packaging:** Jar



- **Dependencies:**
  - **Spring Web (Para la API REST).**
  - **Spring Boot DevTools (Para recarga en vivo).**



Hacer clic en **Finish** para generar el proyecto.

Paso 2: Crear el service FileService

Maneja la lógica de negocio para leer archivos y procesar datos.

```java
package com.example.exceptiondemo.service;

import com.example.exceptiondemo.exception.FileNotFoundException;
import com.example.exceptiondemo.exception.InvalidDataException;
import org.springframework.stereotype.Service;

import java.util.HashMap;
import java.util.Map;

@Service
public class FileService {
    private final Map<String, String> files = new HashMap<>();

    public FileService() {
        files.put("document1.txt", "Contenido del documento 1");
        files.put("report.pdf", "Contenido del reporte PDF");
    }

    // Ejemplo de Checked Exception (Obliga a usar try-catch o throws)
    public String readFile(String filename) throws FileNotFoundException {
        if (!files.containsKey(filename)) {
            throw new FileNotFoundException("El archivo '" + filename + "' no fue encontrado.");
        }
        return files.get(filename);
    }

    // Ejemplo de Unchecked Exception (No obliga a usar try-catch)
    public String processData(Map<String, Object> data) {
        if (data.get("data") == null || ((String)data.get("data")).isEmpty()) {
            throw new InvalidDataException("Los datos proporcionados son inválidos.");
        }
        return "Datos procesados: " + data;
    }
}
```

Explicación:

- **files** : Un Map<String, String> que simula una "base de datos" en memoria con archivos disponibles.
- **readFile(filename):**
  - Usa una Checked Exception (FileNotFoundException).
  - Obliga a manejarse con try-catch o throws.
  - Si el archivo no existe en files, lanza FileNotFoundException.
- **processData(data):**
  - Usa una Unchecked Exception (InvalidDataException).
  - No obliga a manejarse con try-catch.
  - Si el dato es null o vacío, lanza InvalidDataException.

Paso 3: Crear el Controlador FileController

Expone los endpoints REST para interactuar con FileService.

```java
package com.example.exceptiondemo.controller;

import com.example.exceptiondemo.exception.FileNotFoundException;
import com.example.exceptiondemo.service.FileService;

import java.util.Map;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/files")
public class FileController {
    private final FileService fileService;

    public FileController(FileService fileService) {
        this.fileService = fileService;
    }

    // Endpoint que maneja una Checked Exception
    @GetMapping("/{filename}")
    public ResponseEntity<String> getFile(@PathVariable String filename) {
        try {
            return ResponseEntity.ok(fileService.readFile(filename));
        } catch (FileNotFoundException e) {
            return ResponseEntity.status(404).body(e.getMessage());
        }
    }

    // Endpoint que lanza una Unchecked Exception
    @PostMapping("/process")
    public ResponseEntity<String> processFile(@RequestBody Map<String, Object> data) {
        return ResponseEntity.ok(fileService.processData(data));
    }
}
```

Explicación:

- **GET /files/{filename}**:
  - Maneja una **Checked Exception** con try-catch.
  - Si el archivo no existe, devuelve **404 Not Found** con el mensaje "El archivo '...' no fue encontrado".
- **POST /files/process**:
  - Lanza una Unchecked Exception (InvalidDataException).
  - Si los datos son inválidos (null o ""), lanza un **400 Bad Request** automáticamente.

Paso 4: Crear el Exception FileNotFoundException

Representa una excepción para archivos no encontrados.

```java
package com.example.exceptiondemo.exception;

// Checked Exception (obliga a usar try-catch o throws)
public class FileNotFoundException extends Exception {
    public FileNotFoundException(String message) {
        super(message);
    }
}
```

Explicación:

- Extiende **Exception:** Checked Exception.
- Obliga al programador a manejarla con try-catch o throws.

Paso 5: Crear el Exception InvalidDataException

Representa un error cuando los datos ingresados son inválidos.

```java
package com.example.exceptiondemo.exception;

// Unchecked Exception (no obliga a manejarse)
public class InvalidDataException extends RuntimeException {
    public InvalidDataException(String message) {
        super(message);
    }
}
```

Explicación:

- Extiende **RuntimeException:** Unchecked Exception.
- No obliga a manejarse con try-catch.

Paso 6: Crear el Handler GlobalExceptionHandler

Captura excepciones globalmente y devuelve respuestas HTTP personalizadas.

```java
package com.example.exceptiondemo.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class GlobalExceptionHandler {

    // Manejo de Unchecked Exception
    @ExceptionHandler(InvalidDataException.class)
    public ResponseEntity<String> handleInvalidData(InvalidDataException e) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("ERROR: " + e.getMessage());
    }
}
```

Explicación:

- **@RestControllerAdvice:** Permite capturar excepciones globalmente.
- **@ExceptionHandler(InvalidDataException.class):** Captura automáticamente **todas las excepciones InvalidDataException** y responde con 400 Bad Request.

Nota:

- FileNotFoundException **NO se maneja aquí** porque ya es capturado con try-catch en FileController.

Paso 7: ¿Cómo funciona el proyecto?

Caso: Un archivo no existe (Checked Exception)

- GET /files/inexistente.txt
- FileService.readFile() lanza FileNotFoundException.
- FileController captura la excepción y responde 404 Not Found.

Caso: Datos inválidos (Unchecked Exception)

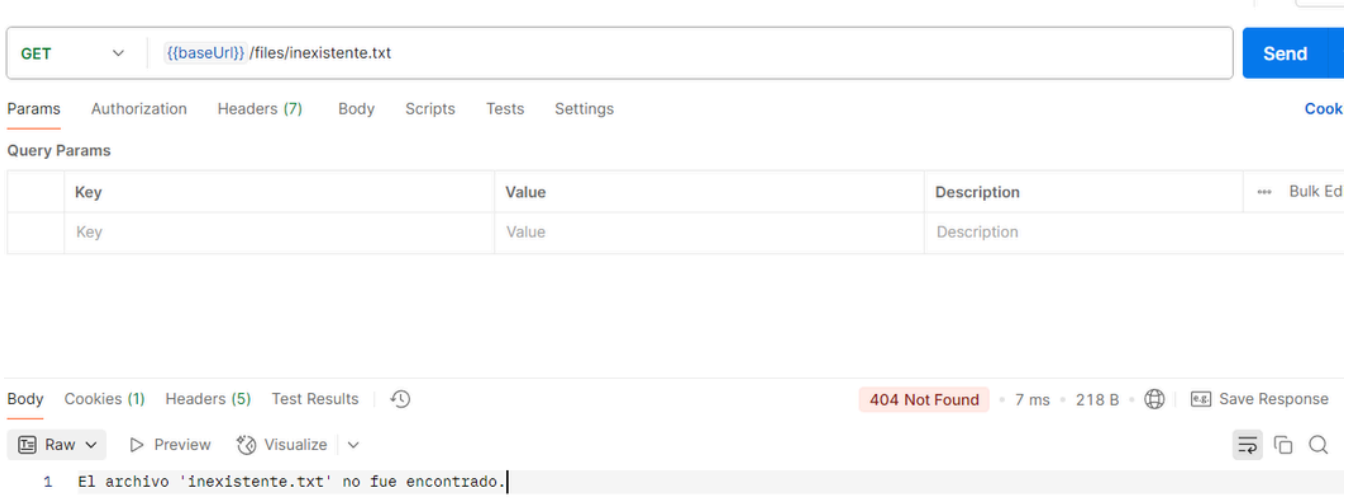
- POST /files/process con {"data": ""}
- FileService.processData() lanza InvalidDataException.
- GlobalExceptionHandler la captura automáticamente y responde 400 Bad Request.

Paso 8: Ejecutar y Probar

- **GET** http://localhost:8080/files/inexistente.txt

Respuesta esperada (404 Not Found):

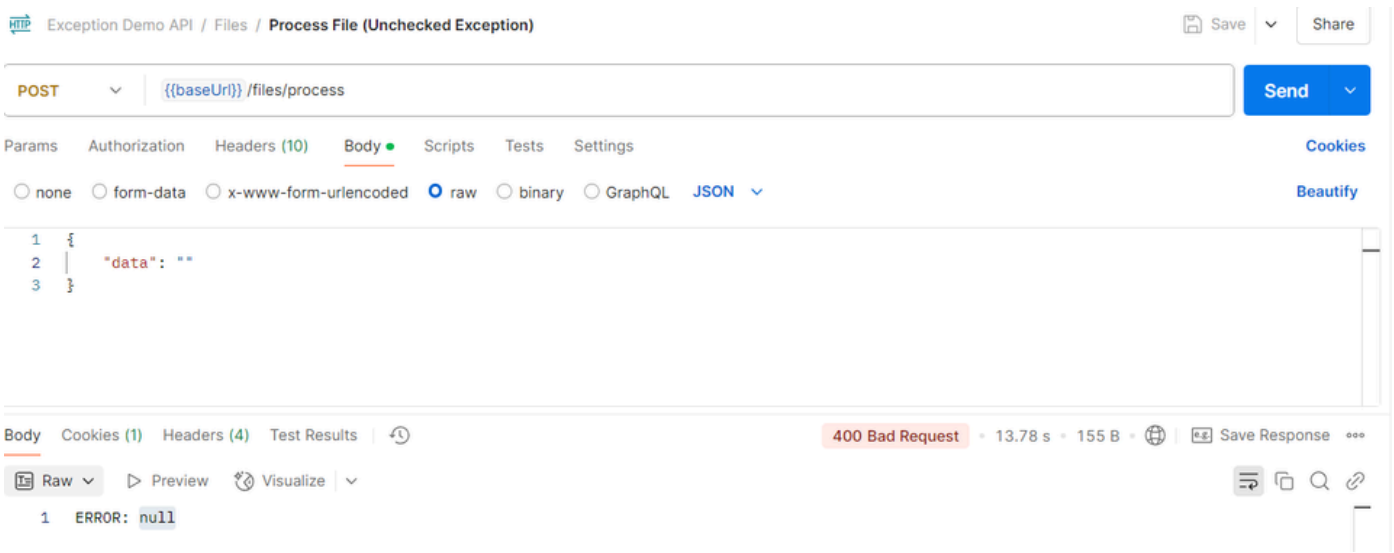
```
El archivo 'inexistente.txt' no fue encontrado.
```



- **POST** http://localhost:8080/files/process

Respuesta esperada (400 Bad Request):

"message": "ERROR: Los datos proporcionados son inválidos."



Paso 9: Conclusión

Checked Exception (FileNotFoundException)

- Manejada **con try-catch** en FileController.
- Obliga al programador a manejarla.
- Se usa cuando se espera que el error ocurra y puede recuperarse.

Unchecked Exception (InvalidDataException)

- Manejada globalmente con @ExceptionHandler.
- No obliga a try-catch.
- Se usa cuando el error es debido a lógica de programación.