**Manejo de excepciones**

El manejo de excepciones es una parte fundamental en el desarrollo de aplicaciones robustas con Spring Boot. En este framework hay varias formas de manejar las excepciones, dependiendo de las necesidades específicas de la aplicación.

Spring Boot posee un **manejo global de excepciones**. Eso quiere decir que el programa nunca se va a romper por más que a un número lo divida por cero o el usuario ingrese un char cuando se solicita un INT. Pero, lo ideal es manejar las excepciones de forma personalizada para que el mensaje de error que le llegue al usuario sea más aclarativo respecto al error que cometió. Pues, si nos quedamos solo con el manejo global de excepciones al usuario solo le llegará un mensaje que diga “error”.

**Tipos de excepciones**

Recordemos que las excepciones pueden clasificarse en dos categorías:

* Excepciones verificadas. Son excepciones que se verifican en tiempo de compilación. Estas excepciones deben ser manejadas explícitamente en el código porque sino no compila directamente. Por ejemplo, cuando querés utilizar un método de la clase Connection sí o sí estás obligado a manejar las posibles excepciones.
* Excepciones no verificadas. Son excepciones que se verifican en tiempo de ejecución. Estas excepciones no necesitan ser manejadas explícitamente en el código. Por ejemplo, aunque no emplees un método de excepción para actuar ante los casos en que el usuario intente dividir por cero, el programa compila y arranca igual.

**Formas de manejar las excepciones en Spring Boot**

* Manejo global de excepciones que tiene Spring Boot. Nunca una aplicación Spring Boot se va a romper por una excepción.
* Try-Catch. Se puede utilizar el bloque try-catch para manejar excepciones en un método específico. NO USAR ESTA FORMA EN LA MATERIA.
* **@ExceptionHandler**. Se puede utilizar la anotación @ExceptionHandler para manejar excepciones en un controlador específico. Es una forma efectiva de manejar errores ya que permite capturar y gestionar excepciones sin necesidad de atraparlas donde surjan con el típico bloque try-catch, lo que genera un código mucho menos verboso y más legible. Se puede crear una clase llamada **Objeto de error** para personalizar bien el mensaje o ruta http enviada al usuario ante la excepción.
* **GlobalExceptionHandler**.

**@ExceptionHandler**

Podemos definir un método dentro de un **controlador** para manejar una excepción específica en ese controlador utilizando la anotación @ExceptionHandler, la cual viene de por sí en el corazón del framework Spring Boot. Este método se ejecutará automáticamente cuando ocurra la excepción indicada.

**Sintaxis**

*@ExceptionHandler (TipoDeExcepcion.class)*

*Public ResponseEntity<ObjetoDeError> nombreMetodo (Tipo de Excepcion e)*

*{*

*//Lógica para manejar la excepción*

*}*

**Funcionamiento**

Cuando se produce una excepción en un controlador, Spring Boot busca un método anotado con @ExceptionHandler que pueda manejar esa excepción específica. Si encuentra un método que coincide con el tipo de excepción, lo invoca y devuelve la respuesta adecuada; sino, Spring Boot usa el manejo global de excepciones y simplemente tira un mensaje que dice “error”. Acordate que Spring Boot, gracias al manejo global de excepciones, impide que una aplicación se rompa por una excepción.

**Ventajas**

El uso de @ExceptionHandler ofrece varias ventajas, incluyendo:

* Manejo específico de excepciones. Permite manejar excepciones específicas de manera diferente, lo que puede ser útil para proporcionar respuestas más informativas y útiles.
* Centralización del manejo de excepciones. Permite centralizar el manejo de excepciones en un solo lugar, lo que puede hacer que el código sea más fácil de mantener y depurar.
* Flexibilidad. Permite devolver diferentes tipos de respuestas, como objetos de error personalizados o códigos de estado HTTP específicos.

**Clase ObjetoDeError**

La clase ObjetoDeError es una clase personalizada que se utiliza para representar errores de manera personalizada. Al utilizar esta clase, se puede proporcionar información detallada sobre los errores y estandarizar las respuestas de error en toda la aplicación.

El propósito de la clase ObjetoDeError es un proporcionar una forma estándar de representar errores en la aplicación, lo que puede ser útil para:

* Proporcionar información sobre detallada sobre el error. La clase ObjetoDeError puede contener propiedades como el mensaje de error, el código de estado HTTP, la fecha y la hora del error, etc.
* Estandarizar la respuesta de error. Al utilizar una clase personalizada para representar errores, se puede asegurar que las respuestas de error sean consistentes en toda la aplicación.
* Facilitar la depuración y registros de errores. La clase ObjetoDeError puede contener información adicional que sea útil para depurar y registrar errores.

**Ejemplo**

A continuación, se muestra un ejemplo de cómo utilizar @ExceptionHandler para manejar una excepción específica en un controlador empleando la clase ObjetoDeError.

Clase ObjetoDeError

public class ObjetoDeError

{

    private String mensaje;

    private String detalles;

    //builders

    public ObjetoDeError(String mensaje, String detalles)

    {

        this.mensaje = mensaje;

        this.detalles = detalles

    }

    //getters and setters

}

@ExceptionHandler

@RestController

@RequestMapping("/api")

public class MiControlador

{

    @AutoWeird //Inyecto la dependencia

    private MiServicio servicio;

    @GetMapping("/datos")

    public ResponseEntity<Dato> obtenerDato(@PathVariable Long id)

    {

        Dato dato = servicio.obtenerDato(id);

        if(dato == null){

            throw new DatoNoEncontradoException("Dato no encontrado");

        }

        return ResponseEntity.ok(dato);

    }

    @ExceptionHandler(DatoNoEncontradoException.class)

    public ResponseEntity<ObjetoDeError> manejarDatoNoEncontrado(DatoNoEncontradoException e)

    {

        ObjetoDeError error = new ObjetoDeError("Dato no encontrado", e.getMessage());

        return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(error);

    }

}

Explicación del código:

1. Inyecto la dependencia de la clase servicio para poder utilizar sus métodos.
2. Solicitud http con el método GET. Se realiza una solicitud GET al endpoint /api/datos/{id} para obtener un dato específico.
3. Excepción. Si el dato no se encuentra, Spring Boot va a buscar si hay un método en el controlador con la anotación @ExceptionHandler que pueda manejar la excepción lanzada (DatoNoEncotradoException). Lo encuentra:
4. Manejo de excepción. El método manejarDatoNoEncontrado anotado con @ExceptionHandler(DatoNoEncontrado.class) se activa automáticamente cuando se lanza la excepción DatoNoEncontradoException.
5. Creación del objeto de error. En el método manejarDatoNoEncontrado, se crea un objeto ObjetoDeError que contiene información sobre el error, como el mensaje de error y el código de estado http.
6. Respuesta. Se devuelve una respuesta http con un código de estado 404(No encontrado) y el objeto ObjetoDeError como cuerpo de la respuesta.

**Conclusión**

@ExceptionHandler es una herramienta poderosa en Spring Boot que permite manejar excepciones de manera específica y centralizada. Al utilizar @ExceptionHandler, se puede proporcionar respuestas más informativas y útiles a los usuarios y hacer que el código sea más fácil de mantener y depurar.

**GlobalExceptionHandler**

La clase **GlobalExceptionHandler** es una clase que se utiliza para manejar excepciones de manera global en una aplicación Sping Boot. Esta clase se anota con la anotación **@RestControllerAdvice**, la cual viene en el corazón del framework, y se utiliza para manejar excepciones que se producen en cualquier parte de la aplicación. Cada método que maneja una excepción en esta clase global lleva anotado la anotación **@ExceptionHandler**. Suele guardarse el archivo clase en una carpeta llamada Exceptions.

**Ventajas**

* Manejo centralizado de excepciones. Permite manejar excepciones de manera centralizada, lo que facilita la gestión y el mantenimiento del código.
* Flexibilidad. Permite manejar diferentes tipos de excepciones y devolver respuestas personalizadas según sea necesario.
* Reutilización de código. Permite reutilizar el código de manejo de excepciones en toda la aplicación. Es común que una excepción se use en más de un controlador. Al tener esta clase global, no tengo que andar copiando y pegando la misma excepción en varias clases-controladoras.

**Funcionamiento**

Al producirse una excepción, Spring Boot busca en la clase con la anotación @RestControllerAdvice (clase GlobalExceptionHandler) si hay algún método, con la anotación @ExceptionHandler, que pueda gestionar dicha excepción. Si existe, el método se activa automáticamente. De no existir, busca en el controlador del método donde se produce la excepción si hay algún método con la etiqueta @ExceptionHandler que pueda manejar dicha excepción. De no existir allí tampoco, Spring Boot activa el manejo global de excepciones y, simplemente, lanza un mensaje de error.

**Ejemplo:**

@RestControllerAdvice

public class GlobalExceptionHandler

{

    @ExceptionHandler (DatoNoEncontradoException.class)

    public ResponseEntity<ObjetoDeError> manejarDatoNoEncontrado(DatoNoEncontradoException e)

    {

        ObjetoDeError error = new ObjetoDeError("Dato no encontrado", e.getMessage, Http.NOT\_FOUND.value());

        return ResponseEntity.status

    }

    @ExceptionHandler

    public ResponseEntity<ObjetoDeError> manejarExcepcionGeneral(Exception e)

    {

        ObjetoDeError error = new ObjetoDeError("Error interno del servidor", e.getMessage(), Http.INTERNAL\_SERVER\_ERROR.value());

        return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(error);

    }

}

En este ejemplo, la clase GlobalExceptionHandler maneja dos tipos de excepciones:

* DatoNoEncontradoException. Se maneja específicamente y se devuelve una respuesta con un código de estado 404(No encontrado).
* Exception. Se maneja de manera general y se devuelve una respuesta con un código de estadop 500 (error interno del servidor).

**Conclusión**

En resumen, la clase GlobalExceptionHandler es una herramienta útil para manejar excepciones de manera global en una aplicación Spring Boot. Al utilizar esta clase, se puede centralizar el manejo de excepciones y proporcionar respuestas personalizadas según sea necesario.