

Elaboro	Documento	Versión	Descripción	Fecha
Luis Sánchez Martínez	Práctica Manejo de excepciones: @ControllerAdvice y @ExceptionHandler en Spring boot.	1.0	Creación de documento	04/04/2024

Práctica Manejo de excepciones: @ControllerAdvice y @ExceptionHandler en Spring boot.

Tiempo estimado: 1 hora.

Uso de:

- Spring boot
- IntelliJ
- Junit
- Mockito

Manejo de excepciones con @ControllerAdvice y @ExceptionHandler

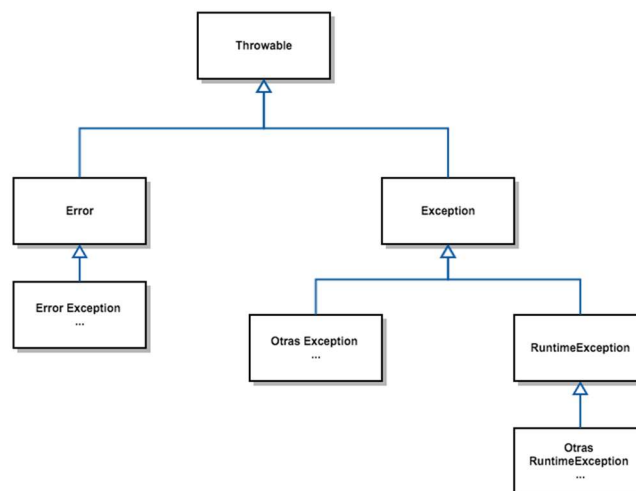
¿Qué es una excepción?

Java utiliza excepciones para proporcionar la posibilidad del manejo de errores para sus programas. Una excepción es un evento que se produce cuando se ejecuta el programa de forma que interrumpe el flujo normal de instrucciones.

Tipos de excepciones

Checked Exceptions: Objetos de la clase *Exception* o cualquier otra clase que herede de ella. Las excepciones Checked son aquellas que deben declararse en el método mediante la palabra *throws* y que obligan al que lo llama a hacer un tratamiento de dicha excepción. Ejemplos: *SocketTimeoutException*, *IOException*, *DataFormatException* y *SQLException*.

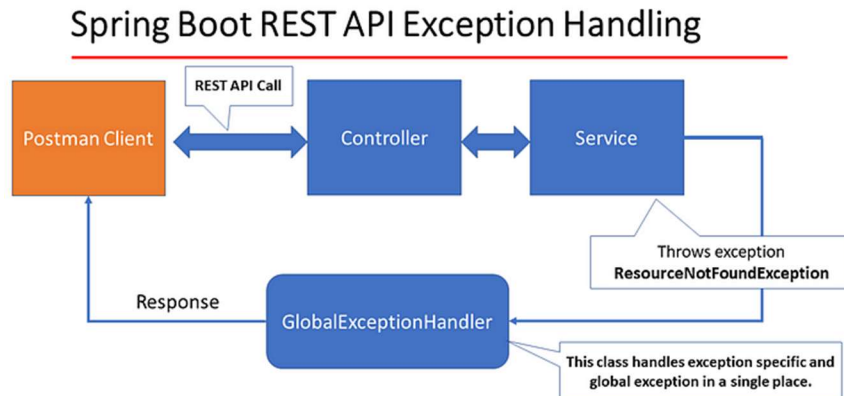
Unchecked Exceptions: Objetos de la clase *RuntimeException* o de cualquier otra clase que herede de ella. Ejemplos: *NullPointerException* y *IndexOutOfBoundsException*.



Controller Advice y Exception Handler

El manejo de excepciones es necesario en el desarrollo de aplicaciones Spring Boot, para el caso de servicios REST se utilizan el `@ControllerAdvice` y el `@ExceptionHandler`.

Las notaciones podrán ser utilizadas para llamar a un método específico cada vez que sea lanzada una excepción en el servicio.



La notación utilizada en la clase que manejará las excepciones es `@ControllerAdvice`, dentro de esta clase podremos tener varios métodos los cuales usarán la notación `@ExceptionHandler` y serán los encargados de manejar cada excepción.

Pasos a seguir

1. El primer cambio se realiza para generar un error 404 al realizar la validación de la lista consultada desde la función GET endpoint `/permiso` en caso de que la lista este vacía.

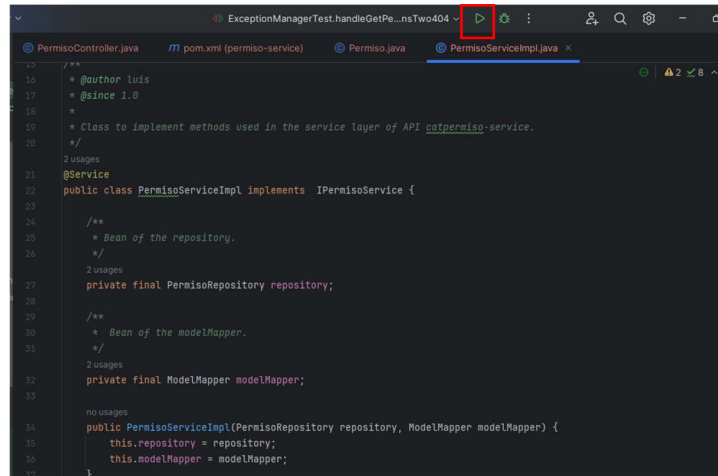
```
no usages
public PermisoServiceImpl(PermisoRepository repository, ModelMapper modelMapper) {
    this.repository = repository;
    this.modelMapper = modelMapper;
}

/**
 * Implements the method to get all records of Permiso entity.
 * @return a response entity of the list of the Permiso object.
 */
3 usages
@Override
public ResponseEntity<List<PermisoDTO>> listAll() {

    if(getAllPermiso().isEmpty()){
        return ResponseEntity.status(404).body(null);
    }
    return ResponseEntity.ok(getAllPermiso());
}

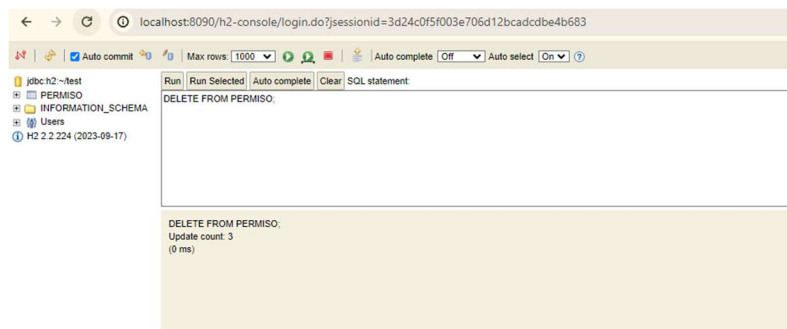
/**
 * Method to query from Permiso entity.
 * @return a list of Permiso object.
 */
2 usages
private List<PermisoDTO> getAllPermiso() {
```

2. Ejecutar el proyecto.

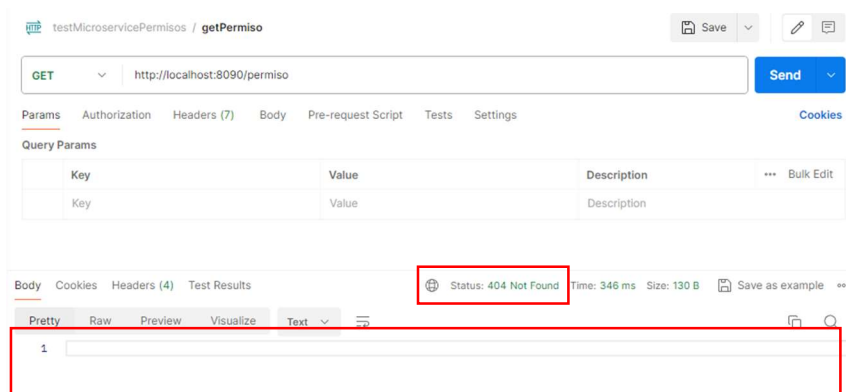


```
16  /**
17  * @author Luis
18  * @since 1.0
19  *
20  * Class to implement methods used in the service layer of API catpermiso-service.
21  */
22  @Service
23  public class PermisServiceImpl implements IPermisService {
24
25      /**
26       * Bean of the repository.
27       */
28      @2 usages
29      private final PermisRepository repository;
30
31      /**
32       * Bean of the modelMapper.
33       */
34      @2 usages
35      private final ModelMapper modelMapper;
36
37      no usages
38      public PermisServiceImpl(PermisRepository repository, ModelMapper modelMapper) {
39          this.repository = repository;
40          this.modelMapper = modelMapper;
41      }
42  }
```

3. Eliminar los registros de la tabla permiso y eliminar los permisos.



4. Ejecutar prueba del endpoint GET de /permiso para validar código de respuesta 404 cuando la lista esta vacía.



5. Ahora para genera correctamente la gestión de excepciones con *ControllerAdvice* se crea la clase *ItemNotFoundException*, crear también el paquete *exception*, clase usada para generar los errores *404 Not Found* del endpoint */permiso* cuando no se encuentren coincidencias en la consulta y la lista este vacía.

```
src
├── main
│   └── java
│       ├── mx.com.tcs.permiso
│       │   ├── configuration
│       │   ├── controller
│       │   └── exception
│       └── ...
└── ...
```

```
/**
 * @author Luis
 * @since 1.0
 *
 * Class to define the not found exception in the manage of exceptions of API of permiso.
 */
1 usage new *
public class ItemNotFoundException extends RuntimeException{

    /**
     * Constructor of ItemNotFoundException using the message of exception.
     *
     * @param message The message of the error.
     */
    1 usage new *
    public ItemNotFoundException(String message){super(message);}

    /**
     * Constructor of ItemNotFoundException using the message and throw cause of exception.
     *
     * @param message The message of the error.
     * @param cause The throw cause of the error.
     */
    1 usage new *
    public ItemNotFoundException(String message, Throwable cause){super(message, cause);}
}
```

6. Se genera la clase ExceptionManager usando `@ControllerAdvice` en la cual se agregue el método `manageNotFoundException` usando `@ExceptionHandler`.

```
1 usage
private final String MESSAGE_NOT_FOUND = "Item(s) not found: ";

/**
 * Method used to manage the no found exception when the list of objects not contains coincidence.
 *
 * @param ex The Exception throws in the service
 * @return The ResponseEntity of ErrorDTO with information about the exception.
 */
2 usage new *
@ControllerAdvice
public class ExceptionManager {

    @ExceptionHandler
    public ResponseEntity<ErrorDTO> manageNotFoundException(ItemNotFoundException ex) {
        ErrorDTO error = buildErrorMessage(ex.getMessage());

        return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
    }

    1 usage new *
    private ErrorDTO buildErrorMessage(String message) {
        ErrorDTO errorDTO = new ErrorDTO();
        errorDTO.setStatus(HttpStatus.NOT_FOUND.value());
        errorDTO.setTimestamp(LocalDate.now());
        errorDTO.setPath("/permiso");
        errorDTO.setError(MESSAGE_NOT_FOUND+message);
    }
}
```

7. Se modifica la clase PermisoServiceImpl levantando una excepción `ItemNotFoundException` cuando la lista consultada está vacía.

```

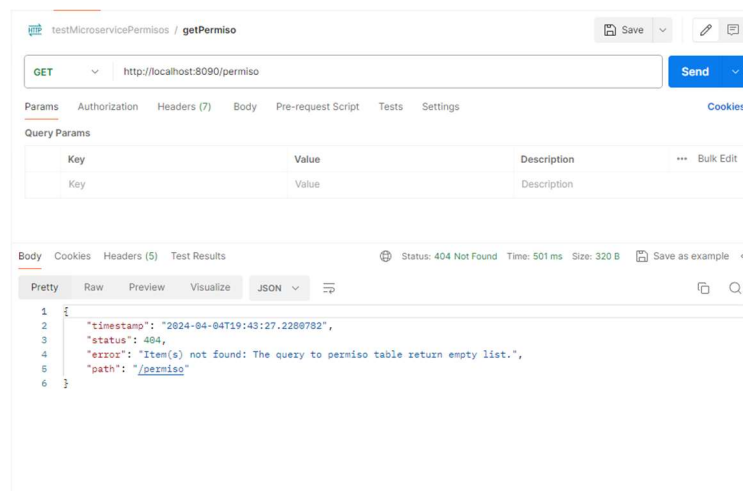
@Override
public ResponseEntity<List<PermisoDTO>> listAll() {
    return ResponseEntity.ok(getAllPermiso());
}

/**
 * Method to query from Permiso entity.
 * @return a list of Permiso object.
 */
1 usage
private List<PermisoDTO> getAllPermiso() {
    List<Permiso> permisoList = (List<Permiso>) repository.findAll();

    if(permisoList.isEmpty()){
        throw new ItemNotFoundException("The query to permiso table return empty list.");
    }
    return permisoList.
        stream().
        map(permiso -> modelMapper.map(permiso, PermisoDTO.class)).
        toList();
}

```

8. Ejecutar prueba del endpoint GET de /permiso para validar código de respuesta 404 cuando la lista está vacía.



9. Realizar la refactorización para volver a tener la cobertura total de las pruebas unitarias después del cambio.

permiso-service									
permiso-service									
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes	
mx.com.tcs.permiso.exception	100%	100%	n/a	100%	0	6	0	6	0
mx.com.tcs.permiso.service	100%	100%	100%	100%	0	5	0	4	0
mx.com.tcs.permiso.controller	100%	100%	n/a	100%	0	2	0	2	0
Total	0 of 101	100%	0 of 2	100%	0	13	0	12	0

Referencias

<https://www.baeldung.com/rest-api-error-handling-best-practices>

<https://docs.spring.io/spring-framework/reference/web/webflux/controller/ann-advice.html>

<https://www.baeldung.com/exception-handling-for-rest-with-spring>

<https://nikhilsukhani.medium.com/mastering-exception-handling-in-spring-boot-using-controlleradvice-and-exceptionhandler-e676b5dd62ed#:~:text=The%20%40ControllerAdvice%20annotation%20is%20used,for%20handling%20a%20specific%20exception.>

<https://www.ibm.com/docs/es/i/7.3?topic=driver-java-exceptions>

<https://blog.pcollaog.cl/2015/08/02/cheked-y-unchecked-exception/>

<https://dip-mazumder.medium.com/spring-boot-exception-handling-best-practices-e8ebe97e8ce>