**Exception Handling with Controlleradvice**

controlleradvice for exceptions in spring boot with example

###### **ChatGPT said:**

In **Spring Boot**, @ControllerAdvice is used for **global exception handling**. It allows you to centralize exception handling logic instead of writing try-catch blocks in every controller.

## **Example of @ControllerAdvice for Exception Handling**

### **Step 1: Create a Custom Exception**

java

CopyEdit

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

### **Step 2: Create a Global Exception Handler**

java

CopyEdit

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.context.request.WebRequest;

import java.util.HashMap;

import java.util.Map;

@ControllerAdvice

public class GlobalExceptionHandler {

// Handle specific exception

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<Object> handleResourceNotFoundException(ResourceNotFoundException ex, WebRequest request) {

Map<String, Object> response = new HashMap<>();

response.put("error", "Resource Not Found");

response.put("message", ex.getMessage());

return new ResponseEntity<>(response, HttpStatus.NOT\_FOUND);

}

// Handle global exception

@ExceptionHandler(Exception.class)

public ResponseEntity<Object> handleGlobalException(Exception ex, WebRequest request) {

Map<String, Object> response = new HashMap<>();

response.put("error", "Internal Server Error");

response.put("message", ex.getMessage());

return new ResponseEntity<>(response, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

### **Step 3: Use the Exception in a Controller**

java

CopyEdit

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/api")

public class SampleController {

@GetMapping("/find")

public String findResource(@RequestParam String id) {

if ("123".equals(id)) {

return "Resource Found";

} else {

throw new ResourceNotFoundException("Resource with ID " + id + " not found.");

}

}

}

## **Testing the Exception Handling**

### **Request:**

bash

CopyEdit

GET /api/find?id=999

### **Response:**

json

CopyEdit

{

"error": "Resource Not Found",

"message": "Resource with ID 999 not found."

}

## **Key Benefits of @ControllerAdvice**

1. **Centralized Exception Handling** – Keeps controllers clean.
2. **Handles Multiple Exceptions** – Can handle specific and global exceptions.
3. **More Readable Code** – No need for try-catch in each controller.

Let me know if you need more details! 🚀

##### **You said:**

how controlleradvice is beneficial from normal exception handling

###### **ChatGPT said:**

Using @ControllerAdvice for **global exception handling** in Spring Boot provides several advantages over **normal exception handling** within individual controllers. Here’s a detailed comparison:

## **1. Centralized Exception Handling**

### **🔹 With @ControllerAdvice (Global Handling)**

* All exception-handling logic is in a single place.
* No need to repeat exception-handling code in multiple controllers.

java

CopyEdit

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<Object> handleResourceNotFound(ResourceNotFoundException ex) {

return new ResponseEntity<>(Map.of("error", ex.getMessage()), HttpStatus.NOT\_FOUND);

}

}

### **🔹 Without @ControllerAdvice (Local Handling)**

* Each controller must handle exceptions separately, leading to **code duplication**.

java

CopyEdit

@RestController

@RequestMapping("/api")

public class SampleController {

@GetMapping("/find")

public ResponseEntity<Object> findResource(@RequestParam String id) {

try {

if (!id.equals("123")) {

throw new ResourceNotFoundException("Resource not found");

}

return ResponseEntity.ok("Resource Found");

} catch (ResourceNotFoundException ex) {

return new ResponseEntity<>(Map.of("error", ex.getMessage()), HttpStatus.NOT\_FOUND);

}

}

}

### **✅ Benefit of @ControllerAdvice**

* No need to **repeat** try-catch in every controller.
* **Cleaner & maintainable code.**

## **2. Separation of Concerns**

* **Without @ControllerAdvice**: Exception handling is mixed with business logic in controllers.
* **With @ControllerAdvice**: Business logic and error handling are separated, making the code more modular.

## **3. Better Maintainability**

* If exception-handling logic needs to be updated, it only needs to be modified in **one place** (GlobalExceptionHandler).
* Without @ControllerAdvice, you would have to update exception-handling logic in multiple controllers.

## **4. Improved Reusability**

* @ControllerAdvice can handle multiple exceptions across different controllers.
* No need to rewrite handling logic for each controller.

## **5. Handles Multiple Exception Types Efficiently**

Instead of handling different exceptions separately in each controller, @ControllerAdvice allows handling all exceptions in one class.

java

CopyEdit

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<Object> handleResourceNotFound(ResourceNotFoundException ex) {

return new ResponseEntity<>(Map.of("error", ex.getMessage()), HttpStatus.NOT\_FOUND);

}

@ExceptionHandler(IllegalArgumentException.class)

public ResponseEntity<Object> handleBadRequest(IllegalArgumentException ex) {

return new ResponseEntity<>(Map.of("error", "Invalid input"), HttpStatus.BAD\_REQUEST);

}

@ExceptionHandler(Exception.class)

public ResponseEntity<Object> handleGlobalException(Exception ex) {

return new ResponseEntity<>(Map.of("error", "Something went wrong"), HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

### **✅ Benefit of @ControllerAdvice**

* One place to manage **multiple exceptions**.
* Reduces **boilerplate code** in controllers.

## **6. Provides Consistent Error Responses**

Using @ControllerAdvice, you can ensure that all exceptions return a **uniform response format**.

Example of a **consistent error response**:

json

CopyEdit

{

"error": "Resource Not Found",

"message": "Resource with ID 999 not found."

}

Without @ControllerAdvice, different controllers might return different error structures, making it harder for clients to handle errors consistently.

## **7. More Readable & Testable Code**

* Makes **unit testing easier** because the exception-handling logic is separate from controllers.
* Improves **readability** by removing redundant try-catch blocks.

## **Conclusion: Why Use @ControllerAdvice?**

| **Feature** | **With @ControllerAdvice** | **Without @ControllerAdvice** |
| --- | --- | --- |
| **Code Duplication** | ❌ Avoided | ✅ Present |
| **Separation of Concerns** | ✅ Yes | ❌ No |
| **Maintainability** | ✅ High | ❌ Low |
| **Error Consistency** | ✅ Uniform error responses | ❌ Inconsistent |
| **Centralized Exception Handling** | ✅ Yes | ❌ No |
| **Readability & Testability** | ✅ Better | ❌ Worse |

### **Final Verdict:**

✅ @ControllerAdvice is the **best practice** for handling exceptions in Spring Boot because it simplifies code, improves maintainability, and ensures consistent error responses across controllers.

Would you like an example with logging or a custom error response structure? 🚀