**✅ 1. Why Use @ControllerAdvice?**

In Spring Boot, exception handling can be centralized using @ControllerAdvice, which allows you to handle exceptions thrown from any controller in one place, improving code reuse and readability.



**✅ 2. Key Annotations**

| **Annotation** | **Purpose** |
| --- | --- |
| @ControllerAdvice | Marks a global exception handler class |
| @ExceptionHandler | Declares method to handle a specific exception type |
| ResponseEntity | Returns HTTP response with custom body and status |

**✅ 3. Create a Custom Exception**

// File: ResourceNotFoundException.java

package com.example.exception;

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

**✅ 4. Global Exception Handler with @ControllerAdvice**

// File: GlobalExceptionHandler.java

package com.example.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDateTime;

import java.util.HashMap;

import java.util.Map;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<?> handleResourceNotFound(ResourceNotFoundException ex) {

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

error.put("timestamp", LocalDateTime.now());

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

error.put("status", HttpStatus.NOT\_FOUND.value());

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

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

}

// Example: handle any other unhandled exception

@ExceptionHandler(Exception.class)

public ResponseEntity<?> handleGlobalException(Exception ex) {

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

error.put("timestamp", LocalDateTime.now());

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

error.put("status", HttpStatus.INTERNAL\_SERVER\_ERROR.value());

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

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

}

}

**✅ 5. Sample Controller Throwing Custom Exception**

// File: UserController.java

package com.example.controller;

import com.example.exception.ResourceNotFoundException;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

public class UserController {

@GetMapping("/{id}")

public String getUser(@PathVariable("id") int id) {

if (id != 1) {

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

}

return "User found!";

}

}

**✅ 6. Sample Response**

**Request:**

GET /users/10

**Response:**

{

"timestamp": "2025-08-05T14:30:00",

"message": "User with ID 10 not found",

"status": 404,

"error": "Resource Not Found"

}

**✅ 7. Tips**

* Use @ResponseStatus in custom exceptions if you don’t want to return ResponseEntity.
* For validation errors, use MethodArgumentNotValidException in your handler.
* Avoid putting business logic in exception handler methods.