How to Handle Custom Exception in Spring Boot

1. Create a Custom Exception Class

This class usually extends RuntimeException.

Example:

```
public class ResourceNotFoundException extends RuntimeException {
public ResourceNotFoundException(String message) {
super(message);
}
}
```

2. Throw the Custom Exception

Throw this exception wherever needed (controller/service layers).

Example:

```
if (employee == null) {
throw new ResourceNotFoundException("Employee not found with ID: " + id);
}
```

3. Handle it Globally Using @ControllerAdvice

Create a global exception handler to ensure centralized response.

Example:

```
@ControllerAdvice
public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)
public ResponseEntity<String> handleResourceNotFound(ResourceNotFoundException ex) {
return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
}

@ExceptionHandler(Exception.class)
public ResponseEntity<String> handleAllOtherExceptions(Exception ex) {
return new ResponseEntity<>("An unexpected error occurred",
HttpStatus.INTERNAL_SERVER_ERROR);
}
}
```

Optional: Custom Error Response Object

Create a custom DTO for structured error response.

```
public class ErrorResponse {
private String message;
private LocalDateTime timestamp;

// Constructors, Getters, Setters
}
```

Update the handler:

```
@ExceptionHandler(ResourceNotFoundException.class)
public ResponseEntity<ErrorResponse> handleResourceNotFound(ResourceNotFoundException
ex) {
ErrorResponse error = new ErrorResponse(ex.getMessage(), LocalDateTime.now());
return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
}
```