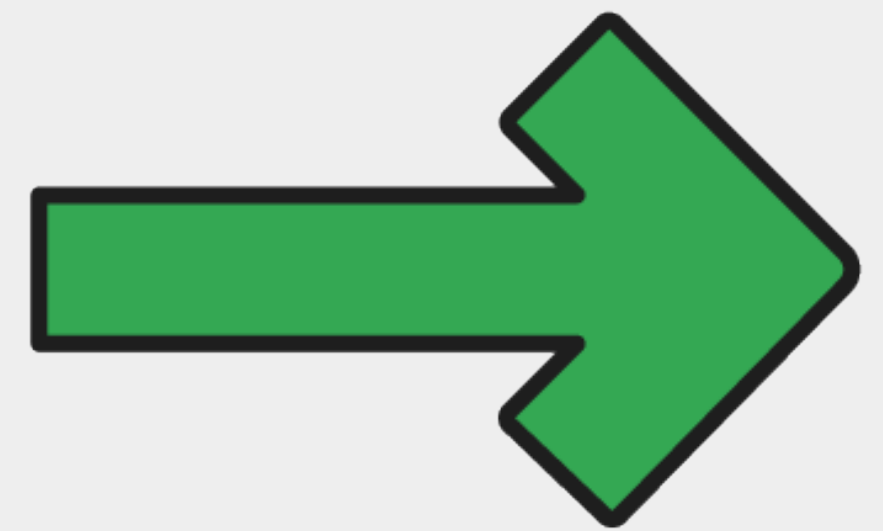


Google Developer Group
Incheon National University

예외 처리

발표자 : 남규리



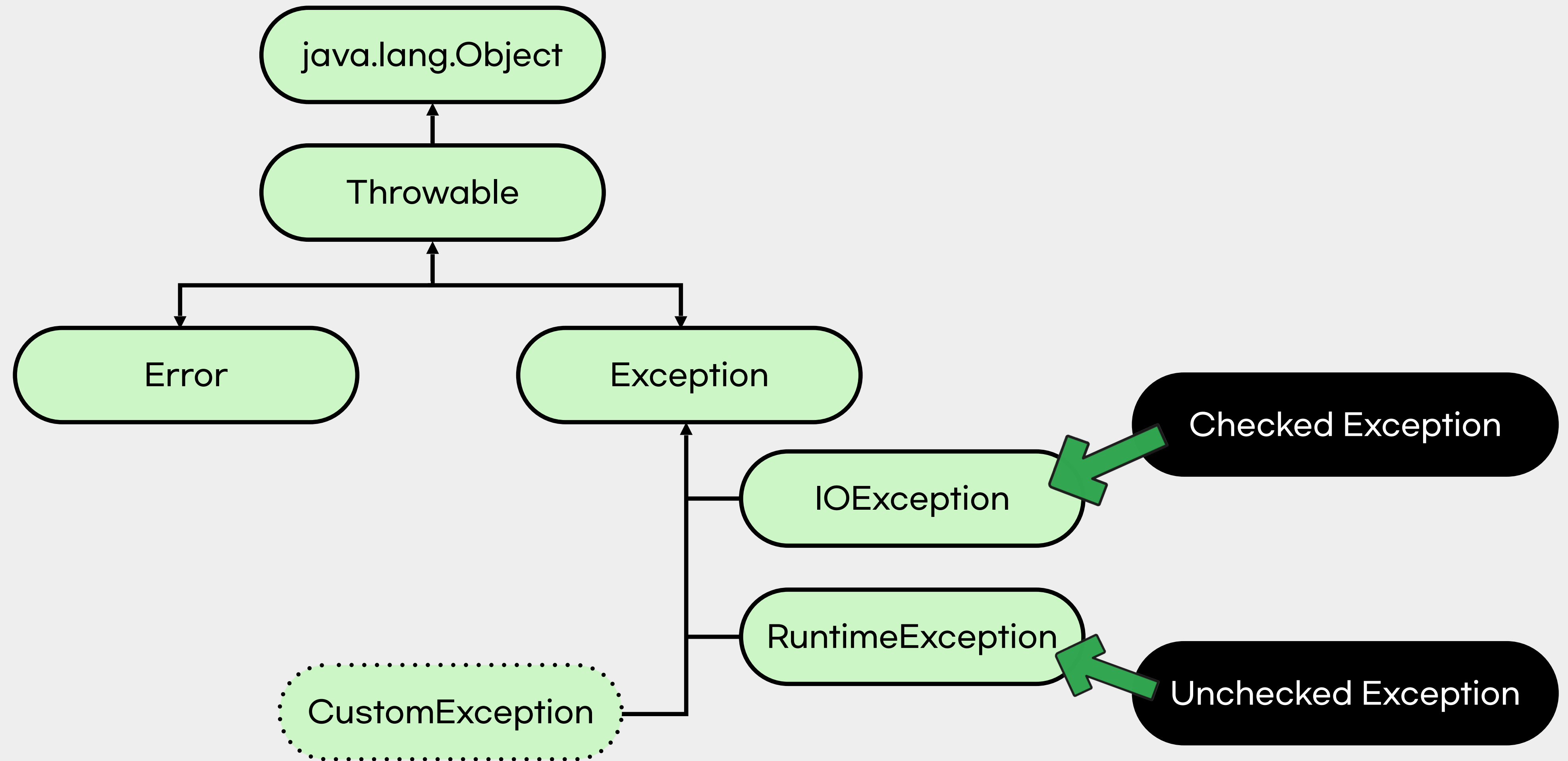
1. 예외 처리란?

“

프로그램 실행 시 발생할 수 있는 예기치 못한 문제들을
예상하여 이를 대비한 코드를 작성하는 것

”

2. 예외 클래스



예외 클래스의 상속 구조

2. 예외 클래스

Checked Exception

IOException

SQLException

컴파일 단계에 발생

Unchecked Exception

RuntimeException

NullPointerException

IllegalArgumentException

IndexOutOfBoundsException

런타임 단계에 발생

```
// 토큰 정보 검증
public boolean validateToken(String token) { 1 usage  👤 gyuri00
    try {
        Jwts.parserBuilder() JwtParserBuilder
            .setSigningKey(key)
            .build() JwtParser
            .parseClaimsJws(token);
        return true;
    } catch (SecurityException | MalformedJwtException e) {
        log.info("Invalid JWT Token", e);
    } catch (ExpiredJwtException e) {
        log.info("Expired JWT Token", e);
    } catch (UnsupportedJwtException e) {
        log.info("Unsupported JWT Token", e);|
    } catch (IllegalArgumentException e) {
        log.info("JWT claims string is empty", e);
    }
    return false;
}
```



try-catch 구문이
너무 많아짐.

3. 스프링에서의 기본 예외 처리


```
@Slf4j
@RequiredArgsConstructor
@RestController
@RequestMapping("/api/user")
public class UserController {
    private final UserService userService;

    @GetMapping("/{id}")
    public String getUser(@PathVariable Long id) {
        // 예외 강제로 발생시키기
        if (id == 0) {
            throw new IllegalArgumentException("잘못된 사용자 ID입니다.");
        }

        // 테스트용 응답
        return "사용자 ID: " + id;
    }
}
```

{ } JSON Preview Visualize

```
1 {
2   "timestamp": "2025-04-07T13:47:56.138+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "message": "잘못된 사용자 ID입니다.",
6   "path": "/api/user/0"
}
```



스프링 부트는 예외 발생 시
/error 경로로 요청을 넘기고,
BasicErrorController가
JSON 또는 HTML로 자동 응답을 생성합니다.

BasicErrorController.java

```
@Controller
@RequestMapping("${server.error.path:${error.path:/error}}")
public class BasicErrorController extends AbstractErrorController {

    private final ErrorProperties errorProperties;
    ...

    @RequestMapping(produces = MediaType.TEXT_HTML_VALUE)
    // 브라우저 요청 시 HTML 에러 페이지 반환
    public ModelAndView errorHtml(HttpServletRequest request, HttpServletResponse response) {
        ...
    }

    @RequestMapping
    // API 요청 시 JSON 형태로 에러 응답 반환
    public ResponseEntity<Map<String, Object>> error(HttpServletRequest request) {
        ...
        return new ResponseEntity<>(body, status);
    }
    ...
}
```


BasicErrorController.java

```
@Controller
@RequestMapping("${server.error.path:${error.path:/error}}")
public class BasicErrorController {
```

```
    private
    ...
```

```
    @RequestMapping
    // 브
    public
    }
    }
```

```
    @RequestMapping
    // AF
    public
    .
    re
    }
```

```
    ...
```

```
}
```

BasicErrorController의 한계

1. 예외 처리 흐름이 복잡하다.

WAS -> filter -> DispatcherServlet -> interceptor -> controller (요청 처리)

🔴예외 발생🔴 -> controller -> interceptor -> DispatcherServlet -> filter -> WAS

WAS -> filter -> DispatcherServlet -> interceptor -> BasicErrorController

2. 클라이언트에게 전달하는 예외의 내용이 충분 하지 않거나 너무 많은 정보를 넘겨 줄 수 있다.

```
response) {
```


4. 예외 처리 전략

✅ HandlerExceptionResolver

- 예외 처리를 **메인 로직에서 분리**하기 위해 사용
- 대부분의 HandlerExceptionResolver는 발생한 예외를 catch 하고 HTTP 상태나 응답 메시지 등을 설정

4. 예외 처리 전략

✓ HandlerExceptionResolver

- 예외 처리를 **메인 로직에서 분리**하기 위해 사용
- 대부분의 HandlerExceptionResolver는 발생한 예외를 catch 하고 HTTP 상태나 응답 메시지 등을 설정

따라서, **WAS 입장에서는 해당 요청이 정상적인 응답인 것으로 인식되며,** 앞에서 설명한 복잡한 WAS의 에러 전달이 진행되지 않는다.

4. 예외 처리 전략

✅ HandlerExceptionResolver 종류 (우선순위 순)

1.ExceptionHandlerExceptionResolver

: 에러 응답을 위한 Controller나 ControllerAdvice에 있는 ExceptionHandler를 처리함.

2.ResponseStatusExceptionResolver

: Http 상태 코드를 지정하는 @ResponseStatus 또는 ResponseStatusException을 처리함.

3. DefaultHandlerExceptionResolver

: 스프링 내부의 기본 예외들을 처리함.

5. 실습

@RestControllerAdvice + @ExceptionHandler를 이용

```
@GetMapping("/{id}") new *
public String getUser(@PathVariable Long id) {
    // 예외 강제로 발생시키기
    if (id == 0) {
        throw new IllegalArgumentException("잘못된 사용자 ID입니다.");
    }

    // 테스트용 응답
    return "사용자 ID: " + id;
}
```

예외 처리 전

```
@GetMapping("/{id}") new *
public String getUser(@PathVariable Long id) {
    // 예외 강제로 발생시키기
    if (id == 0) {
        throw new CustomException(ErrorCode.USER_NOT_FOUND);
    }

    // 테스트용 응답
    return "사용자 ID: " + id;
}
```

예외 처리 후

5. 실습

1. 에러 코드 정의하기

```
@Getter 5 usages new *
@AllArgsConstructor
public enum ErrorCode {
    USER_NOT_FOUND(HttpStatus.NOT_FOUND, message: "사용자를 찾을 수 없습니다."), 1 usage
    INVALID_INPUT(HttpStatus.BAD_REQUEST, message: "잘못된 입력입니다."), no usages
    INTERNAL_ERROR(HttpStatus.INTERNAL_SERVER_ERROR, message: "서버 내부 오류입니다."); no usages

    private final HttpStatus status;
    private final String message;
}
```

5. 실습

2. CustomException 클래스 추가

```
@Getter 4 usages new *  
@RequiredArgsConstructor  
public class CustomException extends RuntimeException{  
    private final ErrorCode errorCode;  
}
```

Unchecked Exception(RuntimeException)을 상속받는 예외 클래스 추가

5. 실습

3. 에러 응답 클래스 추가

```
@Getter 5 usages new *
@Builder
public class ErrorResponse {
    private final int status;
    private final String message;
    private final LocalDateTime timestamp;

    public static ErrorResponse of(ErrorCode errorCode) { 1 usage new *
        return ErrorResponse.builder()
            .status(errorCode.getStatus().value())
            .message(errorCode.getMessage())
            .timestamp(LocalDateTime.now())
            .build();
    }
}
```


5. 실습

4. @RestControllerAdvice 클래스 추가

```
@RestControllerAdvice new *
public class GlobalExceptionHandler {

    @ExceptionHandler(CustomException.class) new *
    public ResponseEntity<ErrorResponse> handleCustomException(CustomException ex) {
        ErrorCode errorCode = ex.getErrorCode();
        ErrorResponse response = ErrorResponse.of(errorCode);
        return new ResponseEntity<>(response, errorCode.getStatus());
    }
}
```

@ExceptionHandler

- 매우 유연하게 에러 처리를 할 수 있는 방법 제공
- 에러 응답을 자유롭게 다룰 수 있음
- 전역으로 처리하기 위해서는 @(Rest)ControllerAdvice를 사용해야 함.

5. 실습

4. @RestControllerAdvice 클래스 추가

```
@RestControllerAdvice new *
public class GlobalExceptionHandler {

    @ExceptionHandler(CustomException.class) new *
    public ResponseEntity<ErrorResponse> handleCustomException(CustomException ex) {
        ErrorCode errorCode = ex.getErrorCode();
        ErrorResponse response = ErrorResponse.of(errorCode);
        return new ResponseEntity<>(response, errorCode.getStatus());
    }
}
```

@RestControllerAdvice

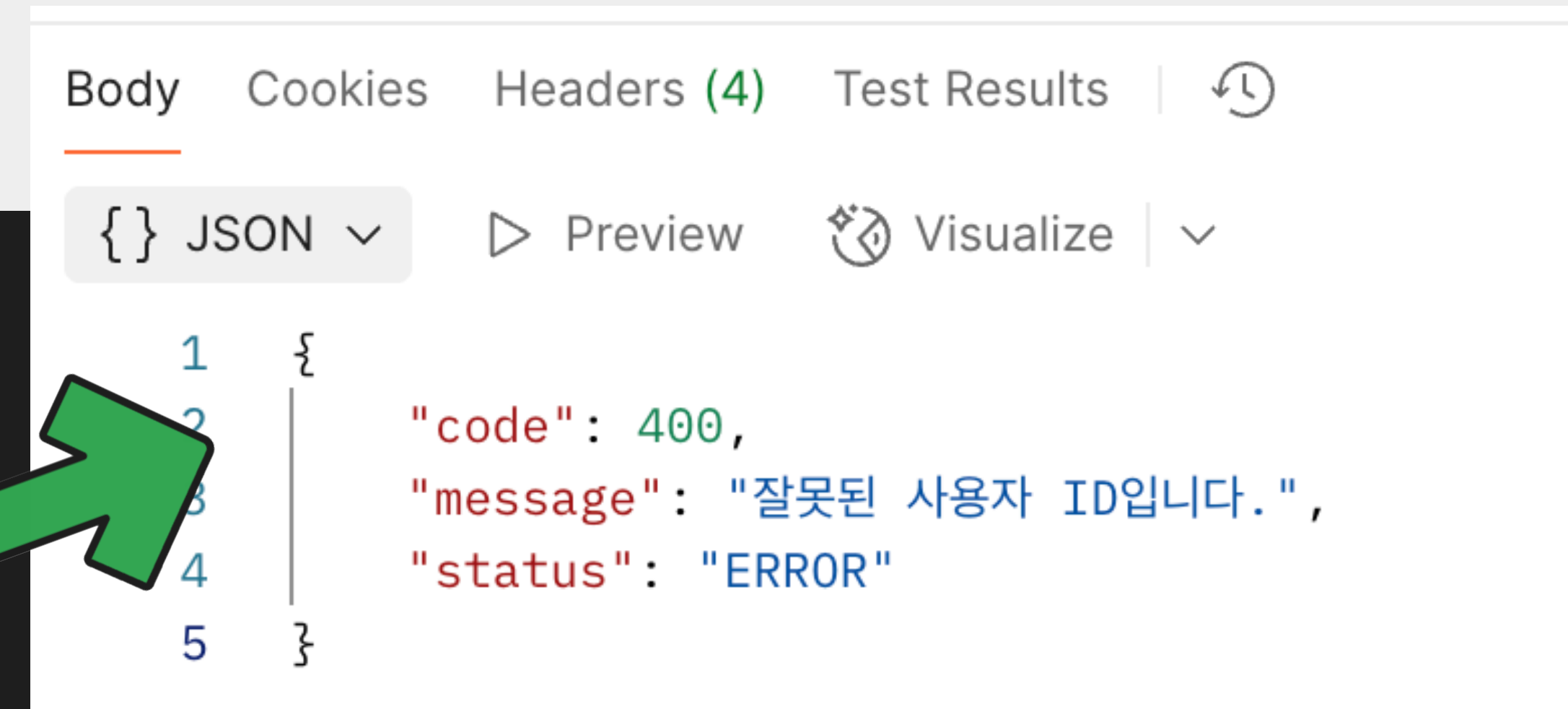
- Spring 4.3부터 제공
- @ControllerAdvice와의 차이점은 에러 응답을 JSON으로 준다는 것임.
- @RestControllerAdvice 선언 -> 빈 등록 됨
- 따라서 전역적으로 발생하는 예외를 감지하고 처리할 수 있음.

5. 실습

5. 에러 응답 확인

```
@GetMapping("/{id}") new *
public String getUser(@PathVariable Long id) {
    // 예외 강제로 발생시키기
    if (id == 0) {
        throw new CustomException(ErrorCode.USER_NOT_FOUND);
    }

    // 테스트용 응답
    return "사용자 ID: " + id;
}
```



정상적으로 예외 처리 완료!

6. ControllerAdvice 장점 및 주의점

장점)

- 하나의 클래스로 모든 컨트롤러에 대해 전역적으로 예외처리 가능
- 직접 정의한 예외 응답을 일관성 있게 클라이언트에게 줄 수 있음

주의점)

- 한 프로젝트당 하나의 ControllerAdvice만 관리하는 것이 좋음
- 만약 여러 개가 필요하다면 basePackages나 annotations 지정 필요

감사합니다!