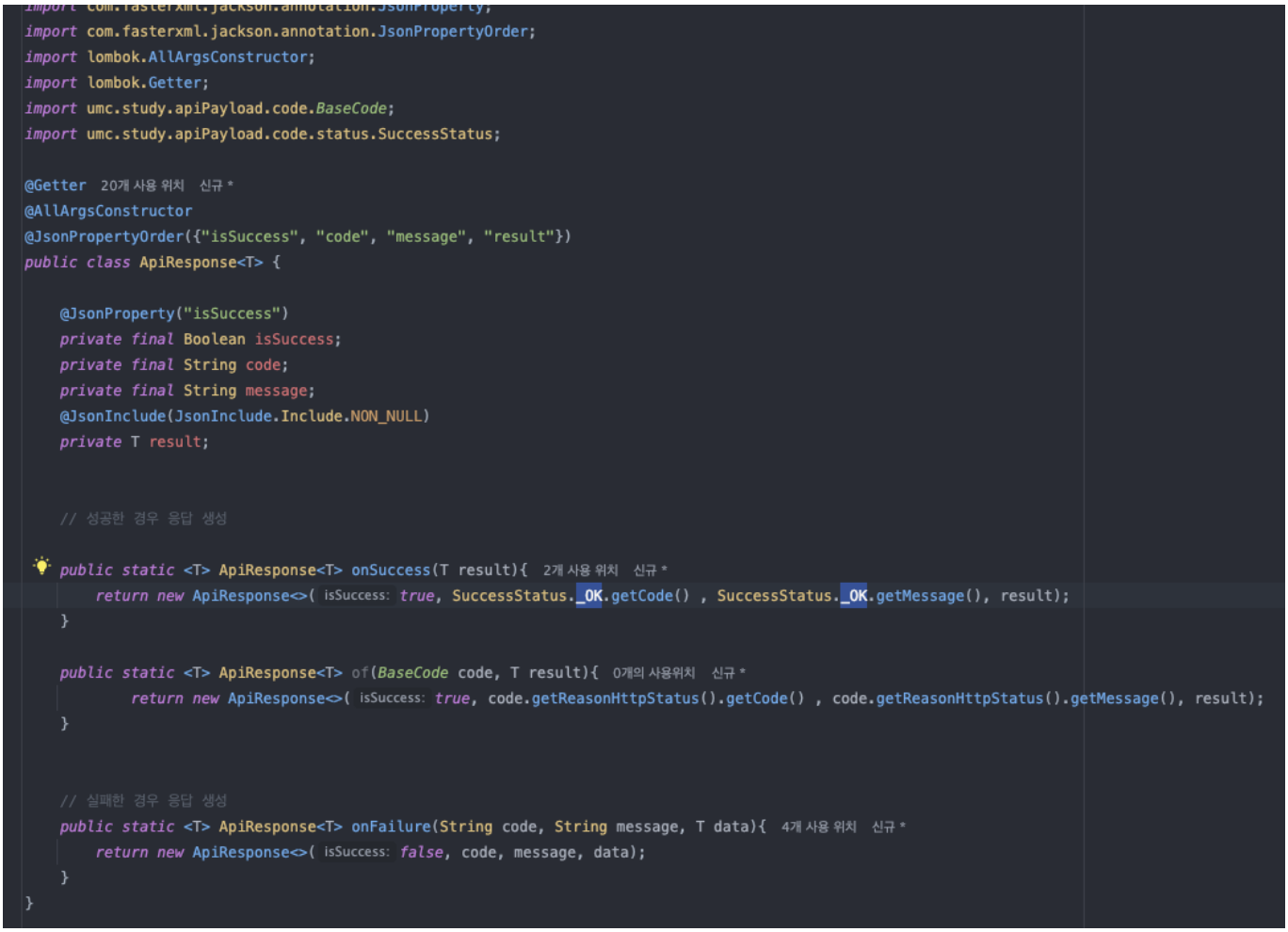
ApiResponse - 통일된 API 응답을 위한



```java
import com.fasterxml.jackson.annotation.JsonPropertyOrder;
import lombok.AllArgsConstructor;
import lombok.Getter;
import umc.study.apiPayload.code.BaseCode;
import umc.study.apiPayload.code.status.SuccessStatus;

@Getter  20개 사용 위치  신규 *
@AllArgsConstructor
@JsonPropertyOrder({"isSuccess", "code", "message", "result"})
public class ApiResponse<T> {

    @JsonProperty("isSuccess")
    private final Boolean isSuccess;
    private final String code;
    private final String message;
    @JsonInclude(JsonInclude.Include.NON_NULL)
    private T result;


    // 성공한 경우 응답 생성

    public static <T> ApiResponse<T> onSuccess(T result){  2개 사용 위치  신규 *
        return new ApiResponse<>( isSuccess: true, SuccessStatus._OK.getCode() , SuccessStatus._OK.getMessage(), result);
    }

    public static <T> ApiResponse<T> of(BaseCode code, T result){  0개의 사용위치  신규 *
            return new ApiResponse<>( isSuccess: true, code.getReasonHttpStatus().getCode() , code.getReasonHttpStatus().getMessage(), result);
    }


    // 실패한 경우 응답 생성
    public static <T> ApiResponse<T> onFailure(String code, String message, T data){  4개 사용 위치  신규 *
        return new ApiResponse<>( isSuccess: false, code, message, data);
    }
}
```

에러 형식 DTO

```java
package umc.study.apiPayload.code;

import lombok.Builder;
import lombok.Getter;
import org.springframework.http.HttpStatus;

@Getter  13개 사용 위치  신규 *
@Builder
public class ErrorReasonDTO {
    private HttpStatus httpStatus;

    private final boolean isSuccess;
    private final String code;
    private final String message;

    public boolean getIsSuccess(){return isSuccess;}  0개의 사용위치  신규 *
}

```

에러 상태 enum

```java
package umc.study.apiPayload.code;

import lombok.Builder;
import lombok.Getter;
import org.springframework.http.HttpStatus;

@Getter  7개 사용 위치  신규 *
@Builder
public class ReasonDTO {
    private HttpStatus httpStatus;

    private final boolean isSuccess;
    private final String code;
    private final String message;

    public boolean getIsSuccess(){return isSuccess;}  0개의 사용위치  신규 *
}

```

```java
@Getter  10개 사용 위치  신규 *
@AllArgsConstructor
public enum ErrorStatus implements BaseErrorCode {
    // 가장 일반적인 응답
    _INTERNAL_SERVER_ERROR(HttpStatus.INTERNAL_SERVER_ERROR,  code: "COMMON500",  message: "서버 에러, 관리자에게 문의 바랍니다."),  2개 사
    _BAD_REQUEST(HttpStatus.BAD_REQUEST, code: "COMMON400", message: "잘못된 요청입니다."),  0개의 사용위치
    _UNAUTHORIZED(HttpStatus.UNAUTHORIZED, code: "COMMON401", message: "인증이 필요합니다."),  0개의 사용위치
    _FORBIDDEN(HttpStatus.FORBIDDEN,  code: "COMMON403",  message: "금지된 요청입니다."),  0개의 사용위치


    // 멤버 관려 에러
    MEMBER_NOT_FOUND(HttpStatus.BAD_REQUEST,  code: "MEMBER4001",  message: "사용자가 없습니다."),  0개의 사용위치
    NICKNAME_NOT_EXIST(HttpStatus.BAD_REQUEST,  code: "MEMBER4002",  message: "닉네임은 필수 입니다."),  0개의 사용위치

    // 예시,,,
    TEMP_EXCEPTION(HttpStatus.BAD_REQUEST,  code: "TEMP4001", message: "이거는 테스트"),  1개 사용 위치
    ARTICLE_NOT_FOUND(HttpStatus.NOT_FOUND,  code: "ARTICLE4001",  message: "게시글이 없습니다.");  0개의 사용위치

    private final HttpStatus httpStatus;
    private final String code;
    private final String message;

    @Override  1개 사용 위치  신규 *
    public ErrorReasonDTO getReason() {
        return ErrorReasonDTO.builder()
                .message(message)
                .code(code)
                .isSuccess(false)
                .build();
    }

    @Override  1개 사용 위치  신규 *
    public ErrorReasonDTO getReasonHttpStatus() {
        return ErrorReasonDTO.builder()
                .message(message)
                .code(code)
                .isSuccess(false)
                .httpStatus(httpStatus)
                .build();
    }
}
```

성공 상태 enum

```java
package umc.study.apiPayload.code.status;

import lombok.AllArgsConstructor;
import lombok.Getter;
import org.springframework.http.HttpStatus;
import umc.study.apiPayload.code.BaseCode;
import umc.study.apiPayload.code.ReasonDTO;

@Getter  3개 사용 위치  신규 *
@AllArgsConstructor
public enum SuccessStatus implements BaseCode {

    // 일반적인 응답
    _OK(HttpStatus.OK,  code: "COMMON200",  message: "성공입니다.");  2개 사용 위치

    private final HttpStatus httpStatus;
    private final String code;
    private final String message;

    @Override  0개의 사용위치  신규 *
    public ReasonDTO getReason() {
        return ReasonDTO.builder()
                .message(message)
                .code(code)
                .isSuccess(true)
                .build();
    }

    @Override  2개 사용 위치  신규 *
    public ReasonDTO getReasonHttpStatus() {
        return ReasonDTO.builder()
                .message(message)
                .code(code)
                .isSuccess(true)
                .httpStatus(httpStatus)
                .build()
                ;
    }
}
```

서비스

```java
package umc.study.service.TempService;

public interface TempQueryService {  0개의 사용위치  1개 구현  신규 *
    void CheckFlag(Integer flag);  0개의 사용위치  1개 구현  신규 *
}
```

TempQueryServiceImpl.java ×

```java
package umc.study.service.TempService;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import umc.study.apiPayload.code.status.ErrorStatus;
import umc.study.apiPayload.exception.handler.TempHandler;

@Service  신규 *
@RequiredArgsConstructor
public class TempQueryServiceImpl implements  TempQueryService{
    // 컨트롤러는 인터페이스를 의존하며 실제 인터페이스에 대한 구체화 클래스는 스프링부트의 의존성 주입을 이용한다
    @Override  1개 사용 위치  신규 *
    public void CheckFlag(Integer flag){
        if(flag == 1) //if문 내부로 들어오면 Service 이후 controller로 가지 않고, 바로 Exception handler에 의해 응답이 보내진다.
            throw new TempHandler(ErrorStatus.TEMP_EXCEPTION);
    }
}
```


```java
package umc.study.service.TempService;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import umc.study.apiPayload.code.status.ErrorStatus;
import umc.study.apiPayload.exception.handler.TempHandler;

@Service  신규 *
@RequiredArgsConstructor
public class TempQueryServiceImpl implements  TempQueryService{
    // 컨트롤러는 인터페이스를 의존하며 실제 인터페이스에 대한 구체화 클래스는 스프링부트의 의존성 주입을 이용한다
    @Override  1개 사용 위치  신규 *
    public void CheckFlag(Integer flag){
        if(flag == 1) //if문 내부로 들어오면 Service 이후 controller로 가지 않고, 바로 Exception handler에 의해 응답이 보내진다.
            throw new TempHandler(ErrorStatus.TEMP_EXCEPTION);
    }
}
```


컨트롤러

TempRestController.java

```java
package umc.study.web.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import umc.study.apiPayload.ApiResponse;
import umc.study.converter.TempConverter;
import umc.study.service.TempService.TempQueryService;
import umc.study.web.dto.TempResponse;

@RestController 신규 *
@RequestMapping("/temp")
@RequiredArgsConstructor
public class TempRestController {
    private final TempQueryService tempQueryService;

    @GetMapping("/test") 신규 *
    public ApiResponse<TempResponse.TempTestDTO> testAPI() {

        return ApiResponse.onSuccess(TempConverter.toTempTestDTO());
    }
    @GetMapping("/exception") 신규 *
    public ApiResponse<TempResponse.TempExceptionDTO> exceptionAPI(@RequestParam Integer flag) {
        tempQueryService.CheckFlag(flag);
        return ApiResponse.onSuccess(TempConverter.toTempExceptionDTO(flag));
    }
}
```

컨버터

```java
package umc.study.converter;

import umc.study.web.dto.TempResponse;

public class TempConverter { 3개 사용 위치 신규 *
    public static TempResponse.TempTestDTO toTempTestDTO() { 1개 사용 위치 신규 *
        return TempResponse.TempTestDTO.builder()
                .testString("This is Test!")
                .build();
    }
    public static TempResponse.TempExceptionDTO toTempExceptionDTO(Integer flag) { 1개 사용 위치 신규 *
        return TempResponse.TempExceptionDTO.builder()
                .flag(flag)
                .build();
    }
}
```

응답 DTO

```java
package umc.study.web.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;

public class TempResponse {  8개 사용 위치  신규 *

    @Builder  3개 사용 위치  신규 *
    @Getter
    @NoArgsConstructor
    @AllArgsConstructor
    public static class TempTestDTO{
        String testString;
    }

    @Builder  3개 사용 위치  신규 *
    @Getter
    @NoArgsConstructor
    @AllArgsConstructor
    public static class TempExceptionDTO{
        Integer flag;
    }
}
```

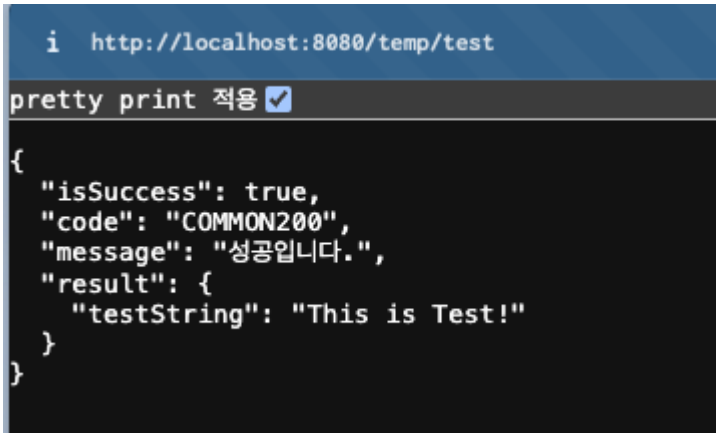
핸들러

```java
package umc.study.apiPayload.exception.handler;

import umc.study.apiPayload.code.BaseErrorCode;
import umc.study.apiPayload.exception.GeneralException;

public class TempHandler extends GeneralException {  2개 사용 위치  신규 *
    public TempHandler(BaseErrorCode errorCode){  1개 사용 위치  신규 *
        super(errorCode); // 상위클래스의 생성자를 호출하여, GeneralException의 로직대로 errorCode를 초기화
    }
}
```

ExceptionAdvice - `@RestControllerAdvice`를 예외처리에 적용

```java
@Slf4j 신규 *
@RestControllerAdvice(annotations = {RestController.class})
public class ExceptionAdvice extends ResponseEntityExceptionHandler {


    @ExceptionHandler 신규 *
    public ResponseEntity<Object> validation(ConstraintViolationException e, WebRequest request) {
        String errorMessage = e.getConstraintViolations().stream() Stream<ConstraintViolation<...>>
                .map(constraintViolation -> constraintViolation.getMessage()) Stream<String>
                .findFirst() Optional<String>
                .orElseThrow(() -> new RuntimeException("ConstraintViolationException 추출 도중 에러 발생"));

        return handleExceptionInternalConstraint(e, ErrorStatus.valueOf(errorMessage), HttpHeaders.EMPTY,request);
    }
    @Override 0개의 사용위치 신규 *
    public ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException e, HttpHeaders headers, HttpStatusCode status, WebRequest request) {

        Map<String, String> errors = new LinkedHashMap<>();

        e.getBindingResult().getFieldErrors().stream()
                .forEach(fieldError -> {
                    String fieldName = fieldError.getField();
                    String errorMessage = Optional.ofNullable(fieldError.getDefaultMessage()).orElse( other: "");
                    errors.merge(fieldName, errorMessage, (existingErrorMessage, newErrorMessage) -> existingErrorMessage + ", " + newErrorMessage);
                });

        return handleExceptionInternalArgs(e,HttpHeaders.EMPTY,ErrorStatus.valueOf( name: "_BAD_REQUEST"),request,errors);
    }

    @ExceptionHandler 신규 *
    public ResponseEntity<Object> exception(Exception e, WebRequest request) {
        e.printStackTrace();

        return handleExceptionInternalFalse(e, ErrorStatus._INTERNAL_SERVER_ERROR, HttpHeaders.EMPTY, ErrorStatus._INTERNAL_SERVER_ERROR.getHttpStatus(),request, e.getMes
    }

    @ExceptionHandler(value = GeneralException.class) 신규 *
    public ResponseEntity onThrowException(GeneralException generalException, HttpServletRequest request) {
        // GeneralException에 대해 다시 한번 오버로딩 된 함수를 호출
        ErrorReasonDTO errorReasonHttpStatus = generalException.getErrorReasonHttpStatus();
        return handleExceptionInternal(generalException,errorReasonHttpStatus, headers: null,request);
```

`/temp/test` 경로 실행 결과:



`/temp/exception?flag` 실행결과: