

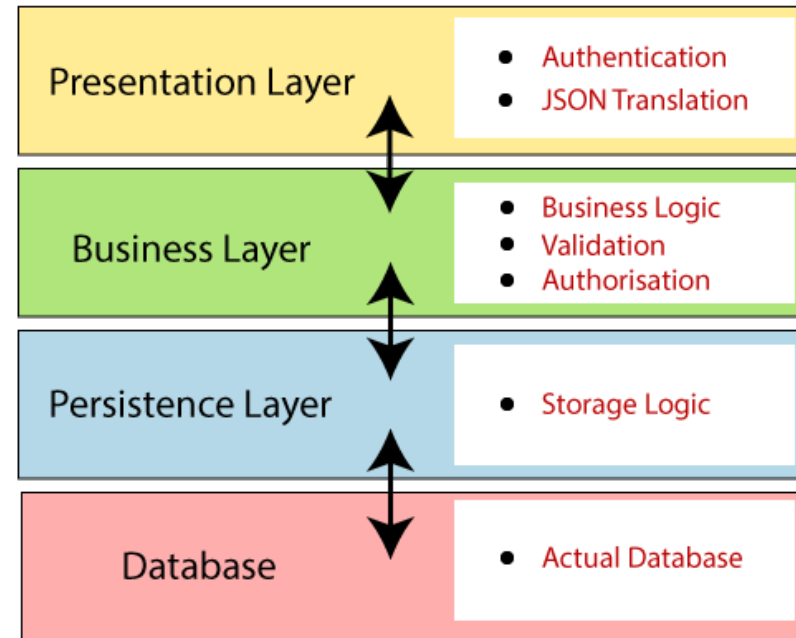
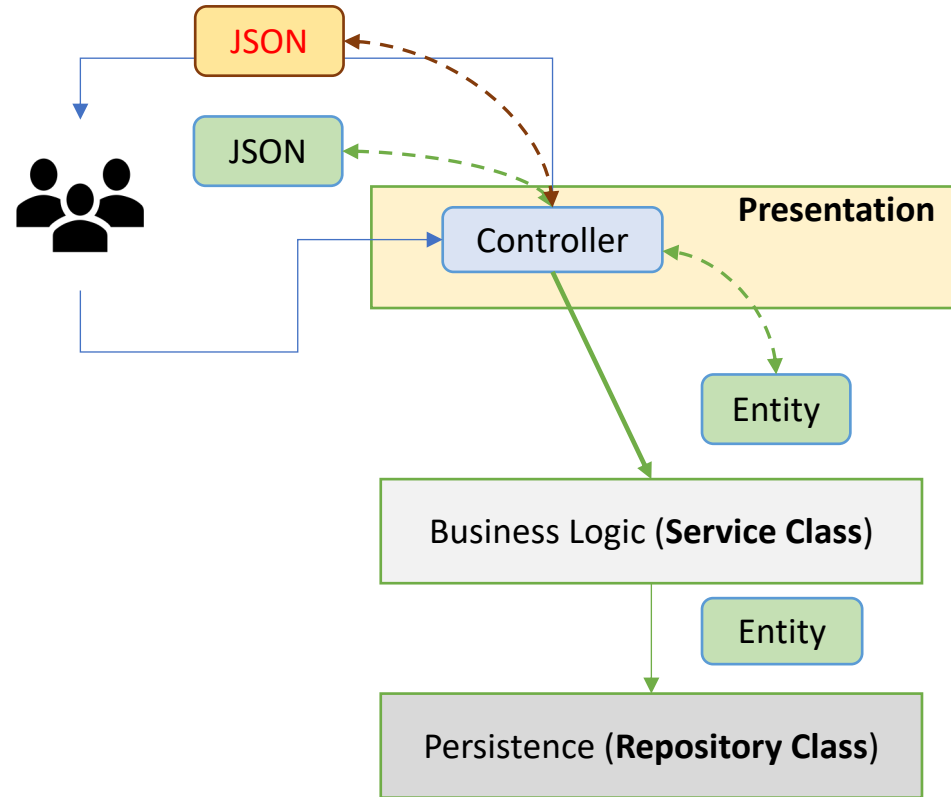


Spring RESTful API Exception Handling

By

Pichet Limvajiranan

Spring Boot Layer Architectures



HTTP Status Codes

- When a client makes a request to an HTTP server - and the server successfully receives the request - the server must notify the client if the request was successfully handled or not.
- HTTP accomplishes this with five categories of status codes:
 - 100-level (Informational) - server acknowledges a request
 - 200-level (Success) - server completed the request as expected
 - 300-level (Redirection) - client needs to perform further actions to complete the request
 - 400-level (Client error) - client sent an invalid request
 - 500-level (Server error) - server failed to fulfill a valid request due to an error with server
- Based on the response code, a client can surmise the result of a particular request.

Handling Errors

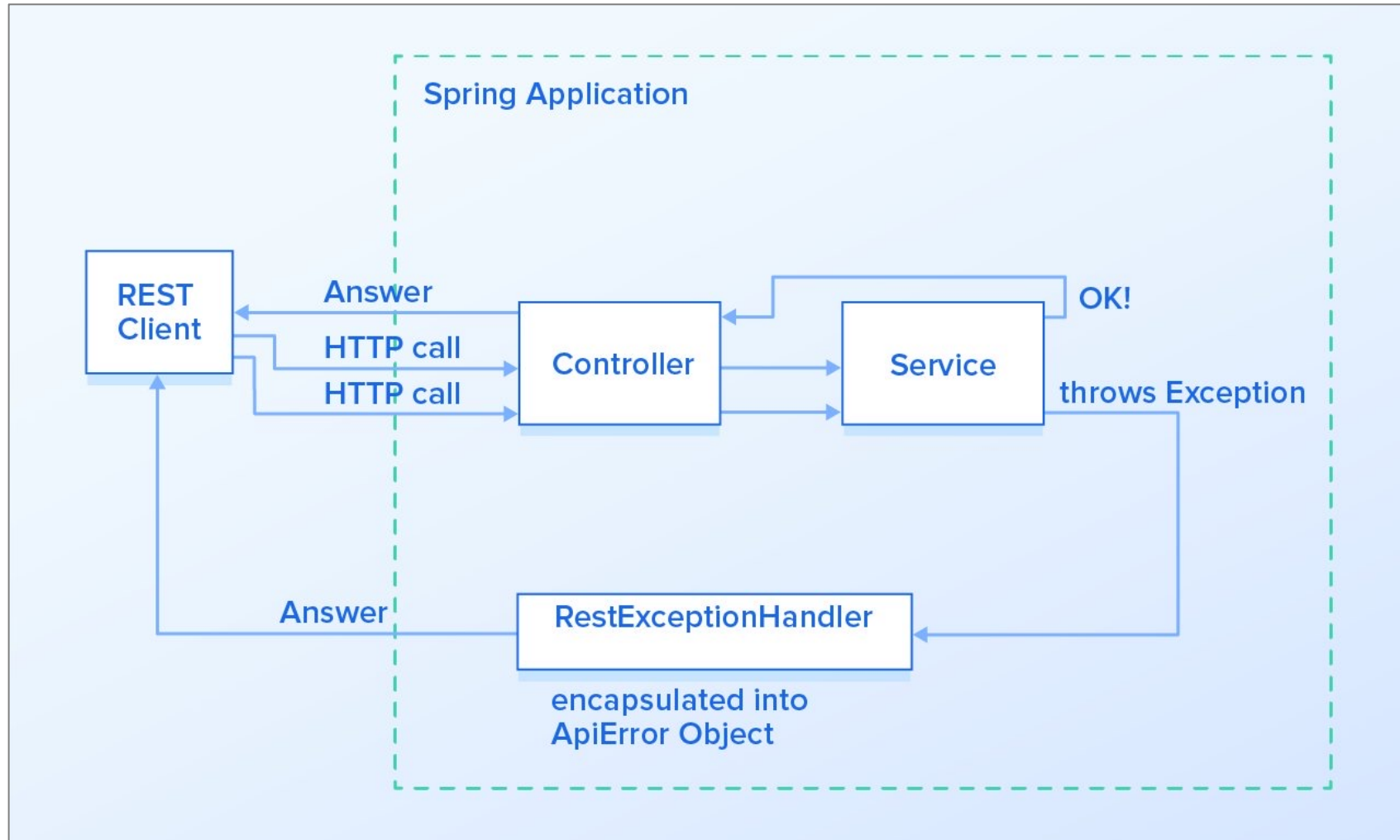
- The first step in handling errors is to provide a client with a proper **status code**. Additionally, we may need to provide more **information in the response body**.
- Basic Responses
 - The simplest way we handle errors is to respond with an appropriate status code.
 - Here are some common response codes:
 - 400 Bad Request - client sent an invalid request, such as lacking required request body or parameter
 - 401 Unauthorized - client failed to authenticate with the server
 - 403 Forbidden - client authenticated but does not have permission to access the requested resource
 - 404 Not Found - the requested resource does not exist
 - 412 Precondition Failed - one or more conditions in the request header fields evaluated to false
 - 500 Internal Server Error - a generic error occurred on the server
 - 503 Service Unavailable - the requested service is not available

Standardized Response Bodies

- In an effort to standardize REST API error handling, the IETF devised RFC 7807, which creates a generalized error-handling schema.
- This schema is composed of five parts:
 - type - a URI identifier that categorizes the error
 - title - a brief, human-readable message about the error
 - status - the HTTP response code (optional)
 - detail - a human-readable explanation of the error
 - instance - a URI that identifies the specific occurrence of the error

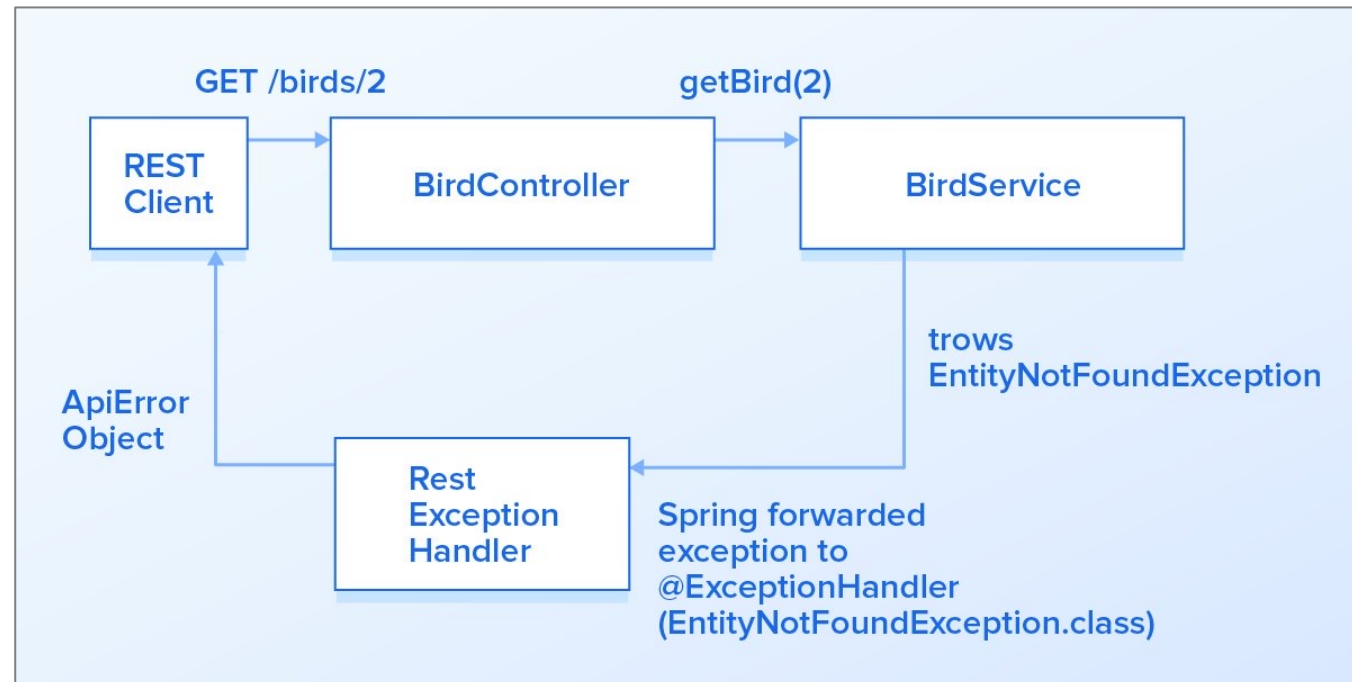
```
{  
  "type": "/errors/incorrect-user-pass",  
  "title": "Incorrect username or password.",  
  "status": 401,  
  "detail": "Authentication failed due to incorrect username or password.",  
  "instance": "/login/log/abc123"  
}
```

Rest Api - Exception Handling



Exception Handling

- Handling exceptions is an important part of building a robust application.
- Spring Boot provides tools to handle exceptions beyond simple 'try-catch' blocks.
 - `@ResponseStatus`
 - `@ExceptionHandler`
 - `@ControllerAdvice`



Spring Boot's Default Exception Handling Mechanism

`server.error.include-stacktrace=always`

```
Body  Cookies  Headers (4)  Test Results
Pretty  Raw  Preview  Visualize  JSON  ↗
1  {
2    "timestamp": "2023-03-19T03:20:25.325+00:00",
3    "status": 500,
4    "error": "Internal Server Error",
5    "trace": "org.springframework.web.client.HttpClientErrorException
        lambda$getProductById$0(ProductService.java:25)\n\tat java.l
        services.ProductService.getProductById(ProductService.java:2
        (ProductController.java:56)\n\tat java.base/jdk.internal.re
        java.base/java.lang.reflect.Method.invoke(Method.java:577)\n\t
        (InvocableHandlerMethod.java:207)\n\tat org.springframework
        java:152)\n\tat org.springframework.web.servlet.mvc.method.a
        (ServletInvocableHandlerMethod.java:117)\n\tat org.springfra
        invokeHandlerMethod(RequestMappingHandlerAdapter.java:884)\n\t"
```

`server.error.include-stacktrace=on_param`
`server.error.include-exception=true`

```
Body  Cookies  Headers (4)  Test Results
Pretty  Raw  Preview  Visualize  JSON  ↗
1  {
2    "timestamp": "2023-03-19T03:42:00.775+00:00",
3    "status": 500,
4    "error": "Internal Server Error",
5    "exception": "org.springframework.web.client.HttpClientError
6    "message": "404 1 does not exists !!!",
7    "path": "/api/products/dtos/1"
8  }
```

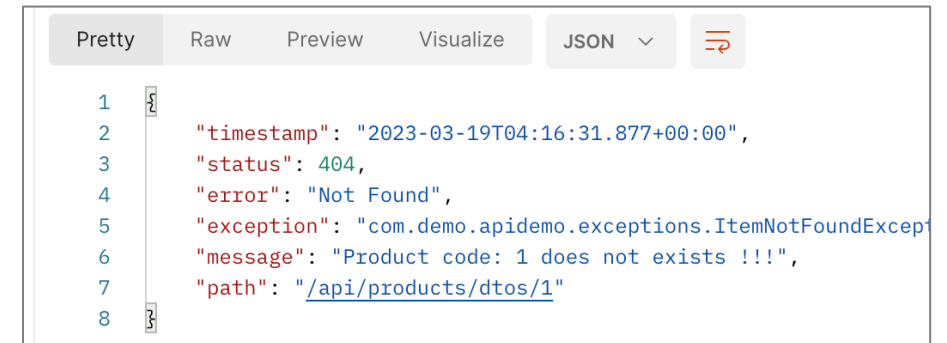
Enum Constants

Enum Constant	Description
ALWAYS	Always add stacktrace information.
NEVER	Never add stacktrace information.
ON_PARAM	Add stacktrace attribute when the appropriate request parameter is not "false".

@ResponseStatus

- As the name suggests, @ResponseStatus allows us to modify the HTTP status of our response. It can be applied in the following places:
 - On the exception class itself
 - Along with the @ExceptionHandler annotation on methods
 - Along with the @ControllerAdvice annotation on classes
- In this section, we'll be looking at the first case only.

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ItemNotFoundException extends RuntimeException {
    public ItemNotFoundException(String message) {
        super(message);
    }
}
```



A JSON viewer interface with tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize'. The 'JSON' tab is selected. The JSON content is as follows:

```
{
  "timestamp": "2023-03-19T04:16:31.877+00:00",
  "status": 404,
  "error": "Not Found",
  "exception": "com.demo.apidemo.exceptions.ItemNotFoundException",
  "message": "Product code: 1 does not exists !!!",
  "path": "/api/products/dtos/1"
}
```

```
public Product getProductById(String productCode) {
    return repo.findById(productCode).orElseThrow(
        ()->new ItemNotFoundException("Product code: " + productCode + " does not exists !!!"));
}
```

Another way to achieve the same is by extending the `ResponseStatusException` class

`@ResponseStatus`, in combination with the `server.error` configuration properties, allows us to manipulate almost all the fields in our Spring-defined error response payload.

```
import org.springframework.http.HttpStatus;
import org.springframework.web.server.ResponseStatusException;

public class ItemNotFoundException extends ResponseStatusException {

    public ItemNotFoundException(String message){
        super(HttpStatus.NOT_FOUND, message);
    }
}
```

@ExceptionHandler

- The @ExceptionHandler annotation gives us a lot of flexibility in terms of handling exceptions.
- For starters, to use it, we simply need to create a method either in the **controller itself** or in a @RestControllerAdvice class and annotate it with @ExceptionHandler:

```
@RestController
public class ProductController {
    :
    @ExceptionHandler(ItemNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public ItemNotFoundException handleItemNotFound (
        ItemNotFoundException exception) {
        return exception;
    }
}
```

```
public class ItemNotFoundException extends RuntimeException {
    public ItemNotFoundException(String message) {
        super(message);
    }
    @Override
    public synchronized Throwable fillInStackTrace() {
        return this;
    }
}
```



The screenshot shows a REST client interface with tabs for Body, Cookies, Headers (5), and Test Results (0/1). The 'Body' tab is selected, displaying a JSON response in 'Pretty' format. The response contains fields for 'cause' (null), 'stackTrace' (empty array), 'message' (Product id 'S18_3020' does not exist !!!), 'suppressed' (empty array), and 'localizedMessage' (Product id 'S18_3020' does not exist !!!).

```
1 {
2   "cause": null,
3   "stackTrace": [],
4   "message": "Product id 'S18_3020' does not exist !!!",
5   "suppressed": [],
6   "localizedMessage": "Product id 'S18_3020' does not exist !!!"
7 }
```

Exception Error Code

- Now, let's finalize an error response payload for our APIs. In case of any error, clients usually expect two things:
 - An error code that tells the client what kind of error it is. Error codes can be used by clients in their code to drive some business logic based on it.
 - Usually, error codes are standard HTTP status codes, but I have also seen APIs returning custom errors code likes E001.
- An additional human-readable message which gives more information on the error and even some hints on how to fix them or a link to API docs.
- We will also add an optional `stackTrace` field which will help us with debugging in the development environment.

Handling validation errors in the response.

```
@Getter
@Setter
@RequiredArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class ErrorResponse {
    private final int status;
    private final String message;
    private final String instance;
    private String stackTrace;
    private List<ValidationError> errors;

    @Getter
    @Setter
    @RequiredArgsConstructor
    private static class ValidationError {
        private final String field;
        private final String message;
    }

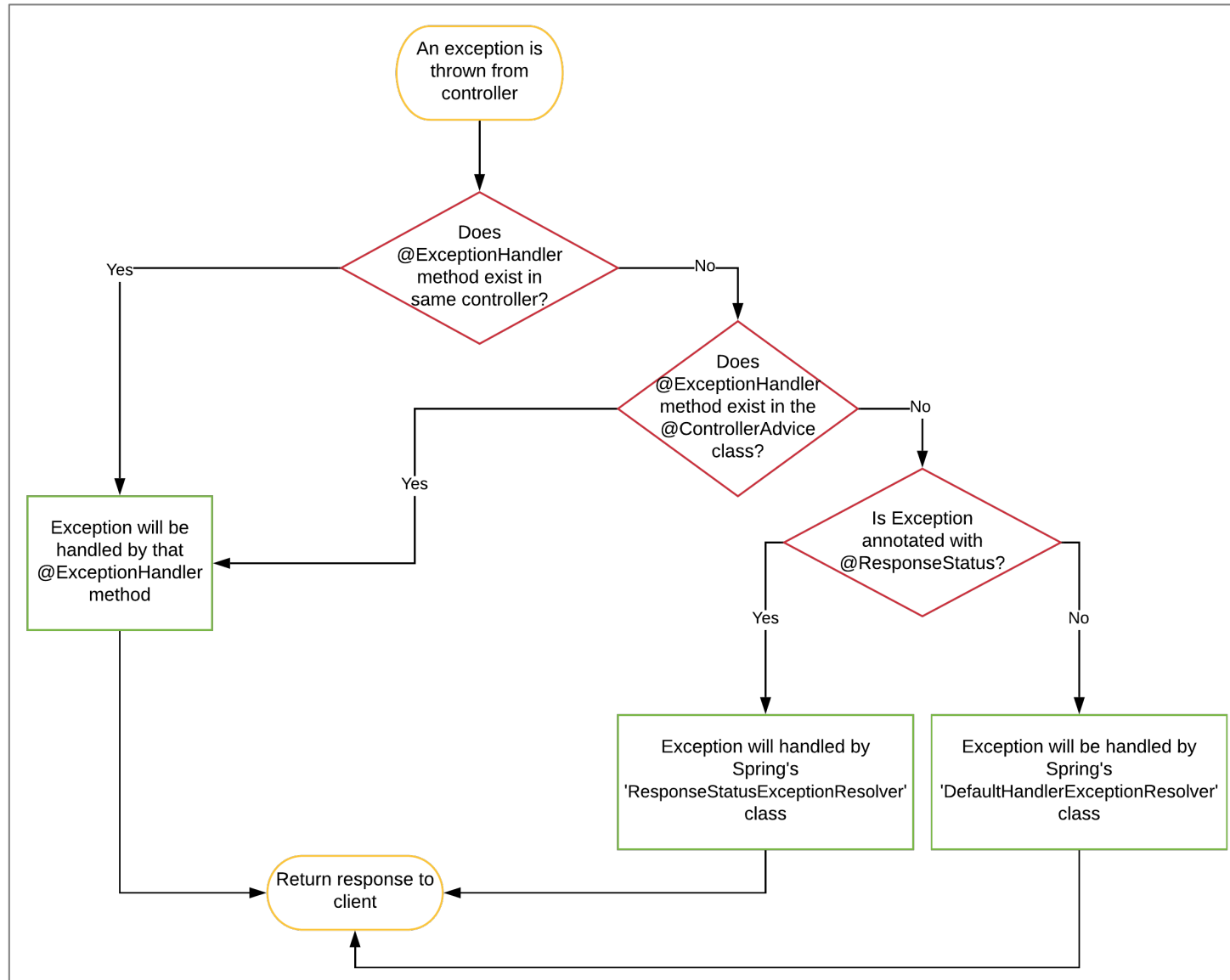
    public void addValidationError(String field, String message){
        if(Objects.isNull(errors)){
            errors = new ArrayList<>();
        }
        errors.add(new ValidationError(field, message));
    }
}
```

```
@ExceptionHandler(ItemNotFoundException.class)
@ResponseStatus(code = HttpStatus.NOT_FOUND)
public ResponseEntity<ErrorResponse>
handleItemNotFound(ItemNotFoundException ex, WebRequest request) {
    ErrorResponse er = new
    ErrorResponse(HttpStatus.NOT_FOUND.value(), ex.getMessage(),
        request.getDescription(false));
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(er);
}
```

@RestControllerAdvice

- The term 'Advice' comes from Aspect-Oriented Programming (AOP) which allows us to inject cross-cutting code (called "advice") around existing methods. A controller advice allows us to intercept and modify the return values of controller methods, in our case to handle exceptions.
- Controller advice classes allow us to apply exception handlers to more than one or all controllers in our application:
- If we want to selectively apply or limit the scope of the controller advice to a particular controller, or a package, we can use the properties provided by the annotation:
 - `@RestControllerAdvice("com.reflectoring.controller")`: we can pass a package name or list of package names in the annotation's value or `basePackages` parameter. With this, the controller advice will only handle exceptions of this package's controllers.
 - `@RestControllerAdvice(assignableTypes={Controller.class})`: only controllers specify by `assignableType` will be handled by the controller advice.

How Does Spring Process The Exceptions?



@RestControllerAdvice (1)

@RestControllerAdvice

```
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {  
    public static final String TRACE = "trace";
```

```
    @ExceptionHandler(ItemNotFoundException.class)
```

```
    @ResponseStatus(HttpStatus.NOT_FOUND)
```

```
    public ResponseEntity<ErrorResponse> handleItemNotFoundException(  
        ItemNotFoundException exception, WebRequest request) {  
        return buildErrorResponse(exception, HttpStatus.NOT_FOUND, request);  
    }
```


@ControllerAdvice (2)

```
@ExceptionHandler(MethodArgumentNotValidException.class)
@ResponseStatus(HttpStatus.UNPROCESSABLE_ENTITY)
public ResponseEntity<ErrorResponse> handleMethodArgumentNotValid(
    MethodArgumentNotValidException ex,
    WebRequest request
){
    ErrorResponse errorResponse = new ErrorResponse(
        HttpStatus.UNPROCESSABLE_ENTITY.value(),
        "Validation error. Check 'errors' field for details.", request.getDescription(false)
    );

    for (FieldError fieldError : ex.getBindingResult().getFieldErrors()) {
        errorResponse.addValidationError(fieldError.getField(),
            fieldError.getDefaultMessage());
    }
    return ResponseEntity.unprocessableEntity().body(errorResponse);
}
```

@ControllerAdvice (3)

```
@ExceptionHandler(Exception.class)
```

```
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
```

```
public ResponseEntity<ErrorResponse> handleAllUncaughtException(  
    Exception exception, WebRequest request) {
```

```
    return buildErrorResponse(  
        exception, "Unknown error occurred", HttpStatus.INTERNAL_SERVER_ERROR, request  
    );  
}
```

@ControllerAdvice (4)

```
private ResponseEntity<ErrorResponse> buildErrorResponse(  
    Exception exception, HttpStatus httpStatus, WebRequest request) {  
  
    return buildErrorResponse( exception, exception.getMessage(), httpStatus, request);  
}  
  
private ResponseEntity<ErrorResponse> buildErrorResponse(  
    Exception exception, String message, HttpStatus httpStatus, WebRequest request) {  
  
    ErrorResponse errorResponse = new ErrorResponse(httpStatus.value(), exception.getMessage(),  
        request.getDescription(false)  
    );  
  
    return ResponseEntity.status(httpStatus).body(errorResponse);  
}
```

Modify ErrorResponse

- ```
public class ErrorResponse {
 private final String title = "Item does not exist !!!";
 private final int status;
 private final String message;
 private final String instance;
 private String stackTrace;
 private List<ValidationError> errors;
 private final LocalDateTime timestamp = LocalDateTime.now();
}
```