



Handling Java Exceptions - Spring Way

Azat Satklichov

azats@seznam.cz,

<http://sahet.net/htm/java.html>,

<https://github.com/azatsatklichov/vatinfo>

Agenda

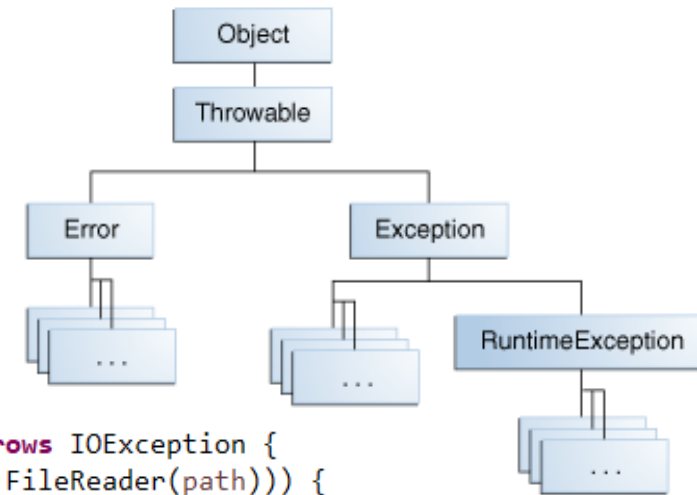
- ❑ Java Exceptions
- ❑ HTTP Request Methods, Response Status Codes
- ❑ Handling exceptions in Spring Framework
- ❑ Mapping exceptions to HTTP status code
- ❑ Writing exception-handling methods
- ❑ Advising Controllers - @ControllerAdvice
- ❑ References

Java Exceptions

- **Checked exceptions** - exceptional conditions should anticipate and recover from, and subject to (compiler forces) Handle or Declare(Specify) [*]. E.g. Path (wrong-file-name)
- **Unchecked exceptions (Errors, Runtime Exceptions)** *are not subject to [*]*

Errors - external to the app., recovery not possible. E.g. Hardware issue

Runtime exception - internal to the app., recovery not possible. E.g. Program err.



Enhanced Features:

Java 7: try-with-resource (AutoCloseable), multi-catch examples, Suppressed,

Java 9: Thread's | Throwables's **getStackTrace(),[]** was costly so new lazy-access to stack-trace **java.lang.StackWalker**

Java 10 (`try (var fis = new FileInputStream("abc.txt"))`)

```
static String readFirstLineFromFile(String path) throws IOException {
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

Suppressed Exceptions - demoSuppressException

method throws the exception thrown from the **finally block**; the exception thrown from the **catch block (or even in try block)** is **suppressed**.

- 1) Use Java 7 **Throwable addSuppressed** and **getSuppressed** methods to get lost exceptions
- 2) Or use try-with-resource Java approach, then you see original Exception will be thrown
- 3) Or safe code – validation check

```
public static void demoSuppressedException(String filePath) throws IOException {
    FileInputStream fileIn = null;
    try {
        fileIn = new FileInputStream(filePath);
    } catch (FileNotFoundException e) {
        throw new IOException(e);
    } finally {
        fileIn.close();
    }
}
```

Unchecked Exceptions — The Controversy

Java programming language does not require methods to **catch** or to **specify** unchecked exceptions (compiler not forces)

Wrap the Exception Without Consuming it – your

Custom Exceptions to add additional business spec-info. E.g. Contactless payment Provisioning err. Or Spring exceptions.

```
public void wrapException(String input) throws MyBusinessException {  
    try {  
        // do something  
    } catch (NumberFormatException e) {  
        throw new MyBusinessException("A message that describes the error.", e);  
    }  
}
```

Avoid unnecessary use of Checked Exceptions

Checked exceptions can increase the reliability of programs [;) see swallowed exception] - when overused, they make APIs painful to use. Java 8, as methods throwing **checked exceptions can't be used directly in streams**

- Instead of throwing a checked exception, the method simply returns an empty optional. (can't return additional info)
- Turn Checked Exception into Unchecked

```
// Invocation with checked exception  
try {  
    obj.action(args);  
} catch (TheCheckedException e) {  
    ... // Handle exceptional condition  
}
```

into this:

```
// Invocation with state-testing method and unchecked exception  
if (obj.actionPermitted(args)) {  
    obj.action(args);  
} else {  
    ... // Handle exceptional condition  
}
```

Antipatterns

- **Swallowing Exceptions:** try { // ... } catch (Exception e) {} // <== catch and swallow
- **Using return in a finally Block:** see [JavaDemoExceptionsAntipatterns](#) and According to the [Java Language Specification](#):
- **Using throw in a finally Block**
- **Using throw as Go To**
- **Log and Throw** - try { // ... } catch (Exception e) { log.error(e); throw e; } //It will write multiple err-msg for same ex.
- **Not documenting Exceptioning in Java doc** - @throws

HTTP Request Methods

HTTP works as a request-response protocol between a client and server.

HTTP Methods: GET, POST, PUT, HEAD, DELETE, PATCH, CONNECT, OPTIONS, TRACE

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

HTTP Response Status Codes

https://www.w3schools.com/tags/ref_httpmessages.asp

When a browser requests a service from a web server, an error might occur, and the server might return an error code (**status code**) like "404 Not Found". Below is a list of HTTP status messages that might be returned. XX (00-99)

1xx: Information	100 Continue	101 Switching Protocols	102 Processing(WebDAV), 103 Checkpoint (early hints, Link Header)
	200 OK (GET, HEAD, PUT, POST, TRACE)	201 Created (POST, PUT)	202 Accepted (noncommittal)
2xx: Successful	203 Non-Authoritative Information	204 No Content	205 Reset Content, ...
	300 Multiple Choices	301 Moved Permanently	302 Found
3xx: Redirection	303 See Other (GET)	304 Not Modified (forcaching)	306 Switch Proxy, ...
	400 Bad Request	401 Unauthorized	402 Payment Required (future)
4xx: Client Error	403 Forbidden	404 Not Found	405 Method Not Allowed (security)
	406 Not Acceptable	415 Unsupported Media Type	...
5xx: Server Error	500 Internal Server Error	501 Not Implemented (Server must support GET, HEAD)	502 Bad Gateway
	503 Service Unavailable	504 Gateway Timeout	505 HTTP Version Not Supported
	511 Network Authentication Required	...	

Handling exceptions

No matter what happens (positive or negative output), the outcome of a servlet request is a servlet response !!!

Spring offers a handful of ways to translate exceptions to responses:

- Certain Spring exceptions are automatically mapped to specific HTTP status codes.
- An exception can be annotated with **@ResponseStatus** to map it to an HTTP status code.
- A method can be annotated with **@ExceptionHandler** to handle the exception.

Mapping exceptions to HTTP status codes

Spring automatically (built-in) maps a dozen of its own exceptions (result of something going wrong in DispatcherServlet or while performing validation) to appropriate status codes.

Although these built-in mappings are helpful, they do no good for any **application exceptions** that may be thrown.

```
public class RateNotFoundException extends
RuntimeException {
    public RateNotFoundException(String msg) {
        super(msg);
    }
}
```

How?



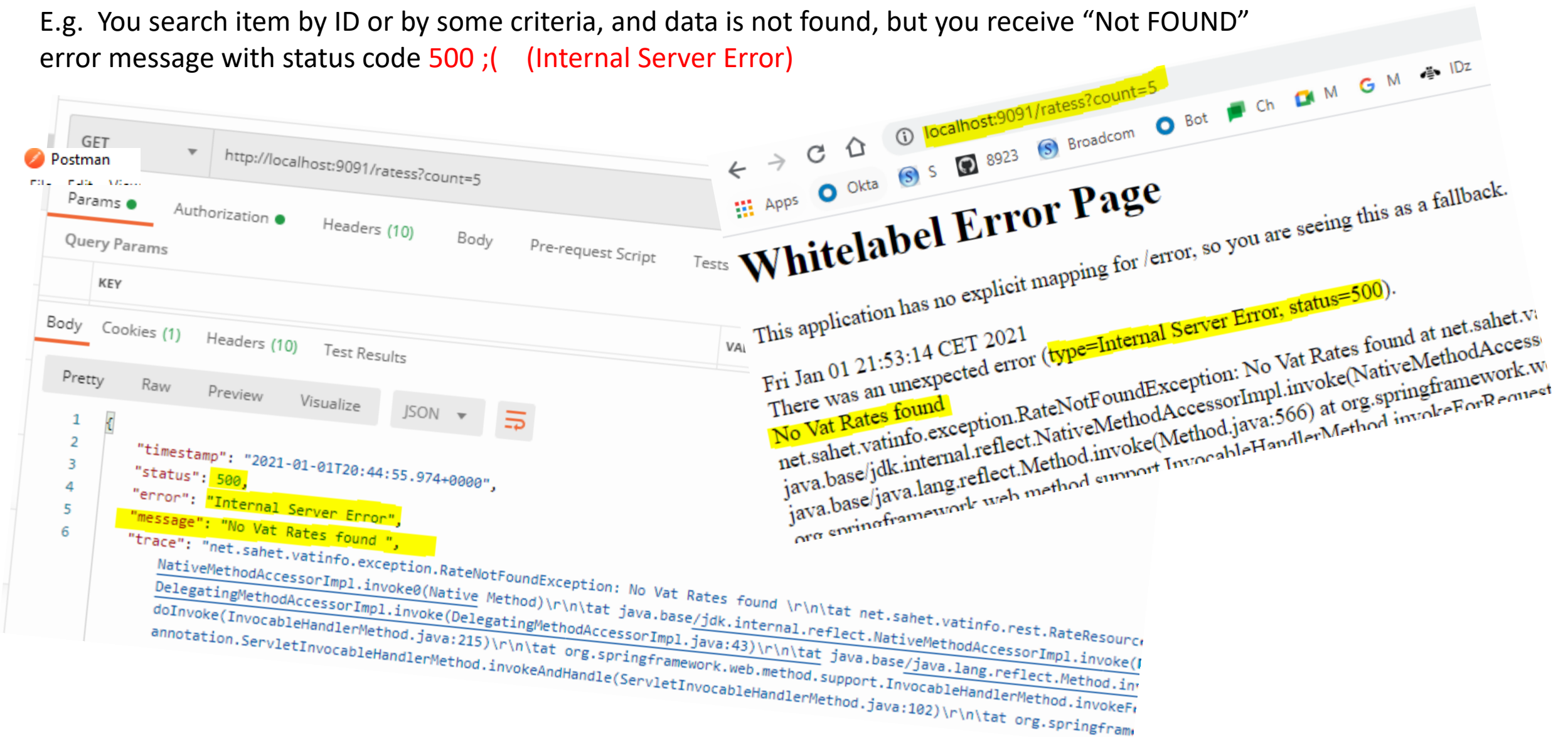
Table 7.1 Some Spring exceptions are mapped by default to HTTP status codes.

Spring exception	HTTP status code
BindException	400 - Bad Request
ConversionNotSupportedException	500 - Internal Server Error
HttpMediaTypeNotAcceptableException	406 - Not Acceptable
HttpMediaTypeNotSupportedException	415 - Unsupported Media Type
HttpMessageNotReadableException	400 - Bad Request
HttpMessageNotWritableException	500 - Internal Server Error
HttpRequestMethodNotSupportedException	405 - Method Not Allowed
MethodArgumentNotValidException	400 - Bad Request
MissingServletRequestParameterException	400 - Bad Request
MissingServletRequestPartException	400 - Bad Request
NoSuchRequestHandlingMethodException	404 - Not Found
TypeMismatchException	400 - Bad Request

What if some engine processes your data and makes a decision based on your STATUS Code? E.g. games, rates, ...

Mapping exceptions to HTTP status codes

E.g. You search item by ID or by some criteria, and data is not found, but you receive “Not FOUND” error message with status code **500 ;((Internal Server Error)**



The image shows a Postman client on the left and a web browser on the right, both displaying an error response from a server.

Postman Details:

- Method: GET
- URL: `http://localhost:9091/rates?count=5`
- Body (JSON):

```
1 {  
2   "timestamp": "2021-01-01T20:44:55.974+0000",  
3   "status": 500,  
4   "error": "Internal Server Error",  
5   "message": "No Vat Rates found",  
6   "trace": "net.sahet.vatinfo.exception.RateNotFoundException: No Vat Rates found \r\n\tat net.sahet.vatinfo.rest.RateResource.  
NativeMethodAccessorImpl.invoke0(Native Method)\r\n\tat java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(  
DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\r\n\tat java.base/java.lang.reflect.Method.in  
voke(InvocableHandlerMethod.java:215)\r\n\tat org.springframework.web.method.support.InvocableHandlerMethod.invoke(  
annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:102)\r\n\tat org.springfram
```

Browser Details:

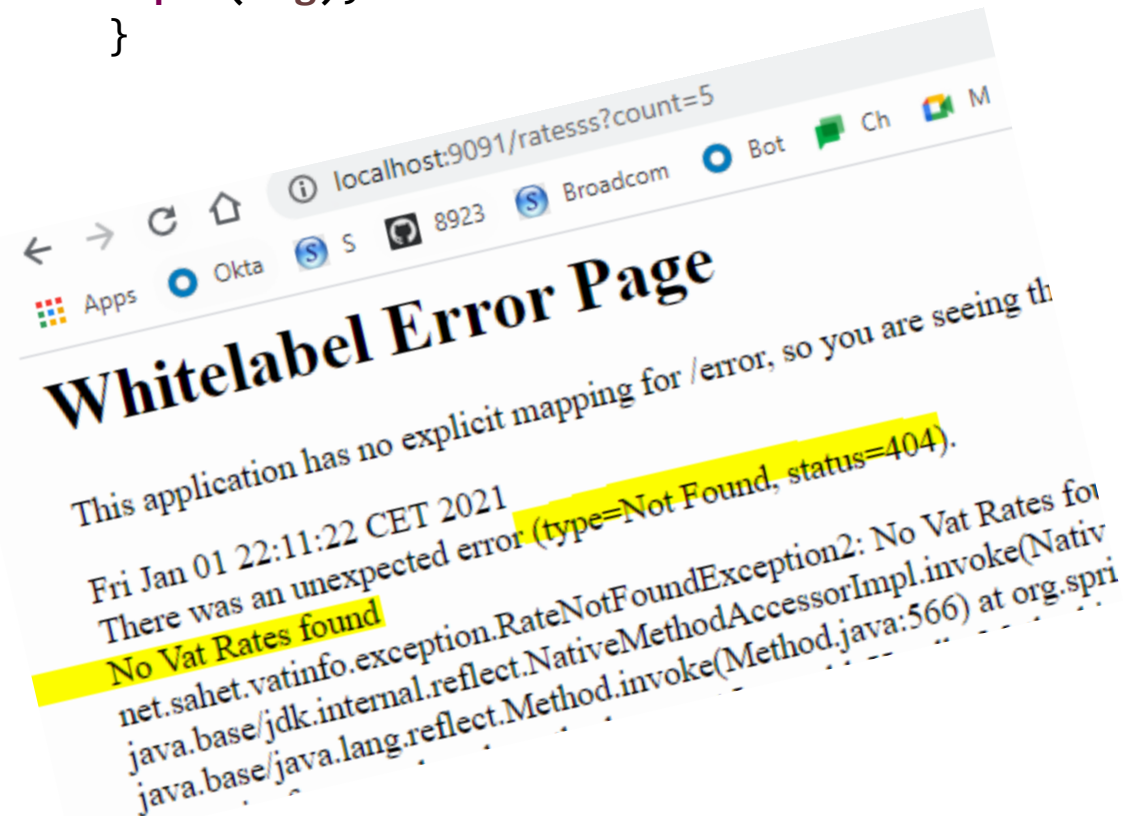
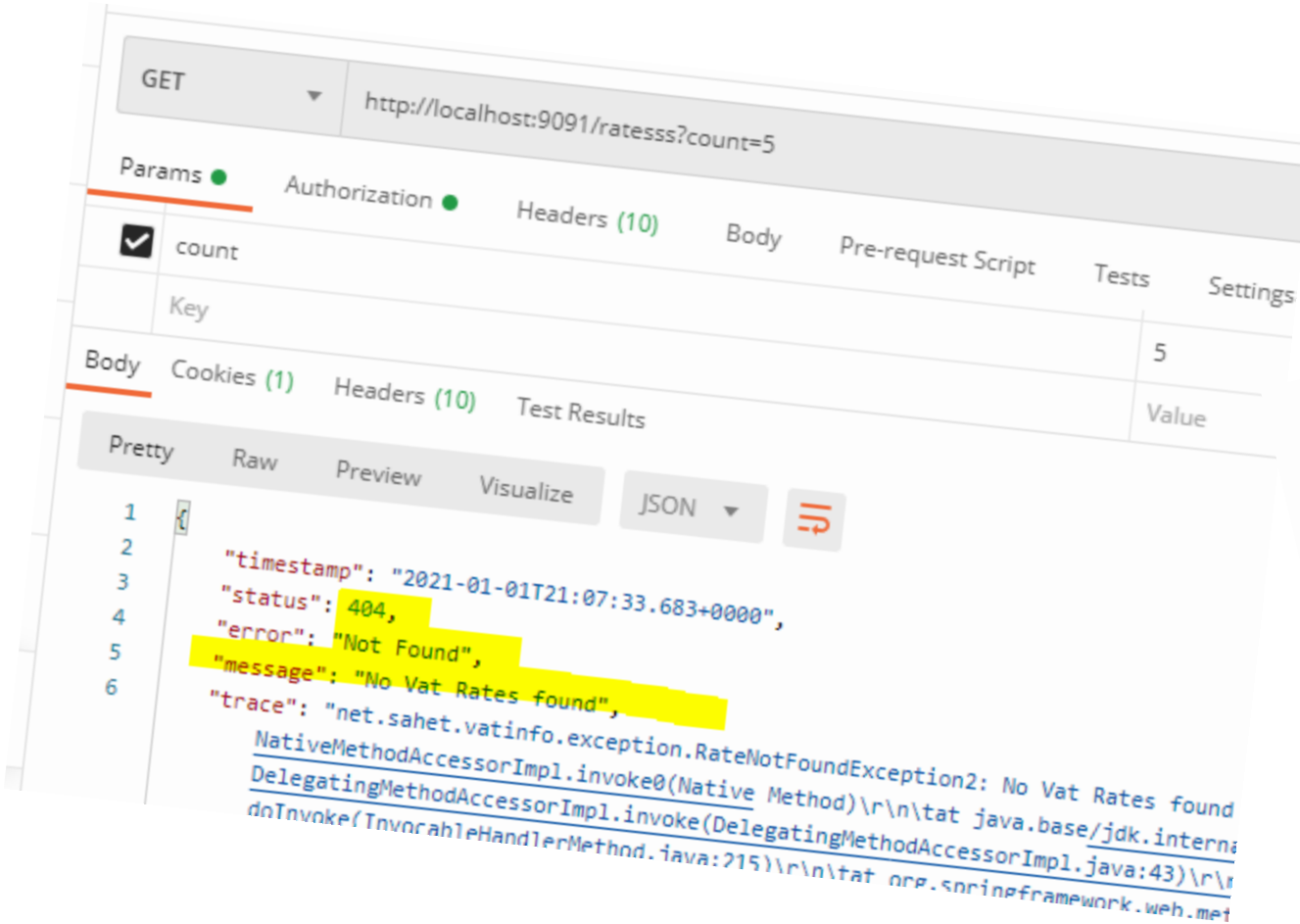
- Address bar: `localhost:9091/rates?count=5`
- Page Title: **Whitelabel Error Page**
- Message: **This application has no explicit mapping for /error, so you are seeing this as a fallback.**
- Timestamp: **Fri Jan 01 21:53:14 CET 2021**
- Error Description: **No Vat Rates found**
There was an unexpected error (type=Internal Server Error, status=500).
`net.sahet.vatinfo.exception.RateNotFoundException: No Vat Rates found at net.sahet.vatinfo.rest.RateResource.
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:43)
java.base/java.lang.reflect.Method.invoke(Method.java:566) at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest`

Mapping exceptions to HTTP status codes

The HTTP status code of 404 is precisely the appropriate response status code **when a resource isn't found**. So, let's use `@ResponseStatus` to map `RateNotFoundException` to HTTP status code 404

```
/** An exception can be annotated with @ ResponseStatus
to map it to an HTTP status code*/
```

```
@ResponseStatus(value = HttpStatus.NOT_FOUND,  
reason = "No Vat Rates found")  
public class RateNotFoundException2 extends  
RuntimeException {  
    public RateNotFoundException2(String msg) {  
        super(msg);  
    }  
}
```



Writing exception-handling methods

Mapping exceptions to status codes is simple and sufficient for many cases. But what if you want the response to carry more than just a status code that represents the error that occurred?

Rather than treat the exception generically as some HTTP error, maybe you'd like to handle the exception the same way you might handle the request itself ??? E.g. create user with existing ID and save.. duplication ex. ..

Listing 7.9 Handling an exception directly in a request-handling method

```
@RequestMapping(method=RequestMethod.POST)
public String saveSpittle(SpittleForm form, Model model) {
    try {
        spittleRepository.save(
            new Spittle(null, form.getMessage(), new Date(),
                form.getLongitude(), form.getLatitude()));
        return "redirect:/spittles";
    } catch (DuplicateSpittleException e) {
        return "error/duplicate";
    }
}
```

← Catch the exception

Here it works fine, just only adds a bit complexity. As we see **two paths** (happy[success] and negative case[fail]), each with a different outcome.

BUT, much **BETTER approach** is, your method should focus only on HAPPY scenario and let the SPRING take care of negative cases (Exception Handling) for you.

Writing exception-handling methods

Let's divide code into happy scenario (much simpler) and exception-handling code, both contains only one path 😊

```
@RequestMapping(method=RequestMethod.POST)
public String saveSpittle(SpittleForm form, Model model) {
    spittleRepository.save(
        new Spittle(null, form.getMessage(), new Date(),
            form.getLongitude(), form.getLatitude()));
    return "redirect:/spittles";
}
```

//methods annotated with `@ExceptionHandler` will handle a `DuplicateSpittleException` thrown from any method in same controller

```
@ExceptionHandler(DuplicateSpittleException.class)
public String handleDuplicateSpittle() {
    return "error/duplicate";
}
```

Perfect, so negative-scenario can be even re-used by other methods.

If `@ExceptionHandler` methods can handle exceptions thrown from any handler is it possible to make it global?

Which means, simply handle exceptions thrown from handler methods in *any* controller.

Advising controllers

Certain aspects of controller classes might be handier if they could be **applied broadly across all controllers** in an app. One option to avoid code duplication is use base Controller class to keep @ExceptionHandler methods.

BUT there is **more elegant option** which Spring introduces. A **controller advice** is any class that's annotated with **@ControllerAdvice** and has one or more of the following kinds of methods:

@ExceptionHandler-annotated (1), @InitBinder-annotated, @ModelAttribute-annotated

Those methods in an @ControllerAdvice-annotated class **are applied globally** across all (1) annotated methods on all controllers in an application. One of the **most practical uses** is to gather all **@ExceptionHandler** methods in a single class so that exceptions from all controllers are handled consistently in one place.

```
@ControllerAdvice
public class VatRateExceptionAdvice {

    @ExceptionHandler(value = VatRateNotFoundException.class)
    public ResponseEntity<Object> exception(VatRateNotFoundException exception) {
        return new ResponseEntity<>("VatRate not found", HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(value = DuplicateRateException.class)
    public ResponseEntity<Object> exception2(DuplicateRateException exception) {
        return new ResponseEntity<>("Rate Duplication", HttpStatus.CONFLICT);
    }
}
```


Advising controllers

Both **HAPPY** and **FAIL** (**VatRate not found**) scenarios are with pretty OBJECT form 😊

GET http://localhost:9091/rates3?count=5

Params Authorization Headers (10) Body Pre-request Script

Query Params

KEY	VALUE
-----	-------

Body Cookies (1) Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "CountriesWithHighestStandardVATRates": [  
3     "Luxembourg",  
4     "Malta",  
5     "Cyprus",  
6     "Netherlands",  
7     "Germany"
```

localhost:9091/rates3?count=5

Apps Oka S 8923 Broadcom Bot Ch M M IDz DD G

```
{"CountriesWithHighestStandardVATRates":["Luxembourg","Malta","Cyprus","Netherlands","Germany"]}
```

GET http://localhost:9091/rates3?count=6

Params Authorization Headers (10) Body Pre-request Script

Body Cookies (1) Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
1 VatRate not found
```

localhost:9091/rates3?count=6

Apps Oka S 8923 Broadcom Bot Ch M M IDz DD G

VatRate not found

Spring 5 and above

Spring 5 introduces the *ResponseStatusException* class — a fast way for basic error handling in our REST APIs.

```
@GetMapping(value =("/{id}")
public Foo findById(@PathVariable("id") Long id, HttpServletResponse response) {
    try {
        Foo resourceById = RestPreconditions.checkNotNull(service.findOne(id));

        eventPublisher.publishEvent(new SingleResourceRetrievedEvent(this, response));
        return resourceById;
    }
    catch (MyResourceNotFoundException exc) {
        throw new ResponseStatusException(
            HttpStatus.NOT_FOUND, "Foo Not Found", exc);
    }
}
```

- + Excellent for prototyping: We can implement a basic solution quite fast.
- + One type, multiple status codes: One exception type can lead to multiple different responses.
- + We won't have to create as many custom exception classes, exceptions can be created programmatically.
- There's no unified way of exception handling: as opposed to *@ControllerAdvice*, which provides a global approach.
- Code duplication: We may find ourselves replicating code in multiple controllers.

Spring Boot Support

A fallback error page for browsers (a.k.a. the Whitelabel Error Page) and a JSON response for RESTful, non-HTML requests:

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Jan 28 12:29:38 CET 2021

There was an unexpected error (type=Internal Server Error, status=500).

No Vat Rates found

As usual, Spring Boot allows configuring these features with properties:

- **`server.error.whitelabel.enabled`**: can be used to disable the Whitelabel Error Page and rely on the servlet container to provide an HTML error message
- **`server.error.include-stacktrace`**: with an *always* value; includes the stacktrace in both the HTML and the JSON default response

Apart from these properties, **we can provide our own view-resolver mapping for `/error`, overriding the Whitelabel Page.**

```
@Component public class MyCustomErrorAttributes extends  
DefaultErrorAttributes { @Override public Map<String, Object>  
getErrorAttributes(  
...
```

```
@Component public class MyErrorController extends  
BasicErrorController { public MyErrorController(ErrorAttributes  
errorAttributes) { super(errorAttributes, new ErrorProperties()); }
```




THANK YOU

References

<https://github.com/azatsatklichov/vatinfo>

<https://www.manning.com/books/spring-in-action-fourth-edition>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

<https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc>

<https://www.baeldung.com/exception-handling-for-rest-with-spring>