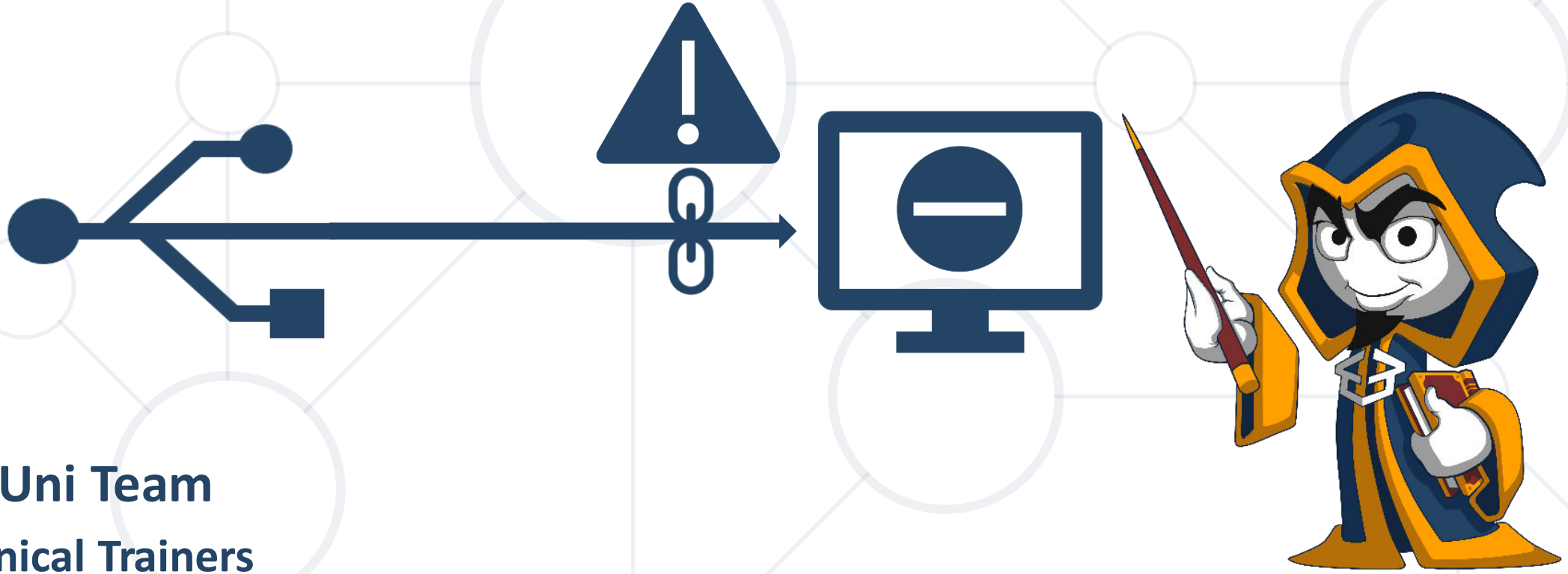
Error Handling

Exception Responses, Exception Handlers, Global Handlers



SoftUni Team

Technical Trainers

Software University

SoftUni Foundation

Software University

http://softuni.bg

Table of Contents

Questions

Software University

sli.do

#java-web

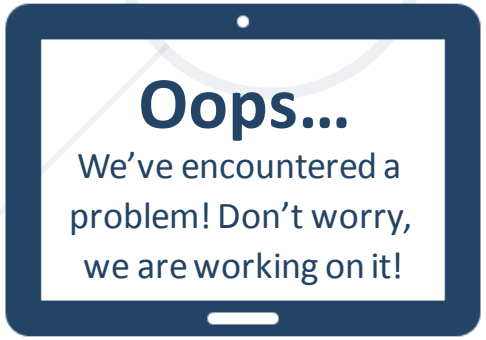
Error Handling

Anticipate! Detect! Resolve!

Error Handling

Software University

- **Error handling** refers to:
  - The **anticipation**, **detection** and **resolution** of programming errors
  - The response & recovery procedures from error conditions
- Error handling is necessary!
  - Improves **user experience**
  - **Optimizes** debugging
  - Facilitates **code maintenance**
  - Ensures **product quality**



Error Handling

- Spring MVC provides several approaches to error handling
  - Per exception, Per controller, Globally
- Depending on the option you are ought to choose
  - Because each option has its own **use cases** and **circumstances**
  - You can use
    - **Response-annotated** custom exceptions
    - **Controller-based** handlers on specified actions
    - **@ControllerAdvise** annotated classes for global handlers

HTTP Status Codes

Annotated Custom Exceptions

HTTP Status Codes

- Unhandled exceptions during a request produce HTTP 500 response
- Any custom exception can be annotated with **@ResponseStatus**
  - Supports all HTTP status codes
  - When thrown and unhandled – produces error page with appropriate response

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Product was not found.")
public class ProductNotFoundException extends RuntimeException {
   // Exception definition
}
```

HTTP Status Codes

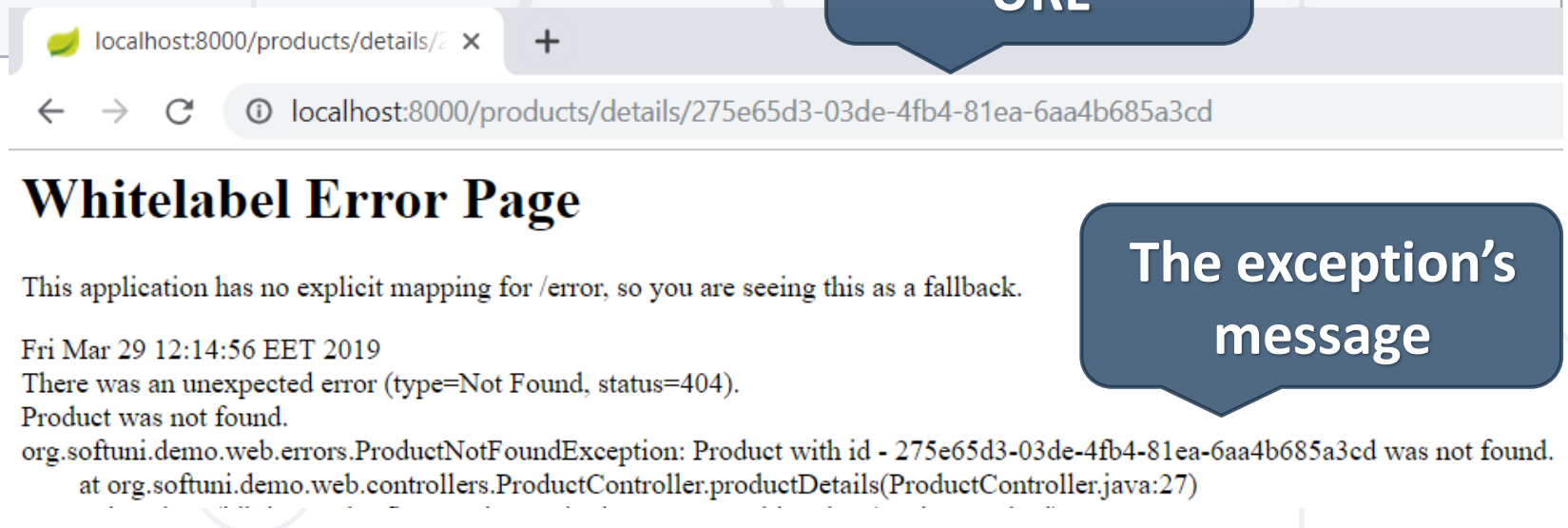
- And the controller action, throwing the exception

```
@GetMapping("/products/details/{id}")
public ModelAndView productDetails(@PathVariable String id, ModelAndView modelAndView) {
    Product product = this.productRepository.findProductById(id);

    if(product == null) throw new ProductNotFoundException(id);

    modelAndView.addObject("product", product);
    return this.view("product/details", modelAndView);
}
```

The requested URL



The produced HTTP Status & Message

The exception's message

Controller-Based Error Handling

Exceptions & Views, @ExceptionHandler

Controller-Based Error Handling

Software University

- You can defined Controller-specific Exception Handlers
  - Casual non-action methods in a Controller
  - Annotated with **@ExceptionHandler** annotation
  - They work **only** for the **Controller** they are defined in
  - Can be annotated with **@ResponseStatus** to convert HTTP status
  - Can accept the **caught exception** as a **parameter**
  - Can return **ModelAndView** or **String** (view name)
  - Can catch **multiple** exception types

Controller-Based Error Handling

Software University

```
@ExceptionHandler({PersistenceException.class, TransactionException.class})
public ModelAndView handleDbExceptions(DatabaseException e) {
   ModelAndView modelAndView = new ModelAndView("error");
   modelAndView.addObject("message", e.getMessage());

   return modelAndView;
}
```

Parent Exception

An error occurred while processing your request!

Error: Database server is down.

```
<html>
<head>...</head>
<body>
   <h1>An error occurred while processing your request!</h1>
   <p th:text="|Error: ${message}|"></p>
</body>
</html>
```

Controller-Based Error Handling

- Handler methods have flexible signatures
  - You can pass in servlet-related objects as parameters
    - **HttpServletRequest**, **HttpServletResponse**
    - **HttpSession**, **Principal**
- The **Model** or **ModelAndView** cannot be a parameter though
  - Instead of passing it, you have to setup it inside the method
  - Nevertheless, this is not an issue because the **IoC container** would have done the same (pass an **empty instance**)

Controller-Based Error Handling

Software University

- It is not a good practice for full error **stacktraces** to be exposed
  - Your users don't need to see ugly exception web-pages
  - You may even have security policies which **strictly forbid** any public exception info
  - Hide as **much information** as **possible** and present **User-friendly** error pages
  - For **testing** purposes you may view details
    - This may need an **environment** setup

Global Exception Handling

@ControllerAdvice classes

Global Exception Handling

Software University

- There is a way to achieve Global exception handling in Spring
  - This is done through the **@ControllerAdvise** annotation
- Any class annotated with **@ControllerAdvise** turns into an **interceptor-like** controller:
  - Enables **global exception handling**
  - Enables **model enhancement** methods
  - Enables **model-binding customization**

Global Exception Handling

Software University

- In **@ControllerAdvice** classes you still use **@ExceptionHandler**
  - However, this time it refers to the whole application
  - The error handling is not limited only to a specific controller

```
@ControllerAdvice
public class GlobalExceptionHandler {
   @ExceptionHandler({TransactionException.class, PersistenceException.class})
   public ModelAndView handleDatabaseErrors(DatabaseException e) {
      ModelAndView modelAndView = new ModelAndView("index");
      modelAndView.addObject("message", e.getMessage());
      modelAndView.addObject("stack", {...} /* Formatted Stack Trace */);

      return modelAndView;
   }
}
```

Global Exception Handling (REST)

- **RESTful requests** may also generate unexpected exceptions
  - HTTP Error response codes are a good choice
  - However sometimes you might need more than just a status
    - **Customized Error Object**, which can be presented on the Client
    - **Limited Information** returned to the Client
- You can customize the **Error Response** by introducing a class
  - The **Error Handler** itself remains the same as in casual web apps

Global Exception Handling (REST)

Software University

```
public class ErrorInfo {
    public final String url;
    public final String ex;
    public ErrorInfo(String url, Exception ex) {
        this.url = url;
        this.ex = ex.getLocalizedMessage();
    }
}
```

```
@ControllerAdvice
public class GlobalRESTExceptionHandler {
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler({TransactionException.class, PersistenceException.class})
    public @ResponseBody ErrorInfo handleRESTErrors(HttpServletRequest req, DbException e) {
        return new ErrorInfo(req.getRequestURL(), ex);
    }
}
```

Exception Handling

Live Demonstration

What to use When?

- Spring offers many choices, when it comes to error handling
  - There are some semantics, that should be followed, though
- Here are some rules for good error handling structure:
  - For custom exceptions, **@ResponseStatus** is the ideal choice
  - For all other exceptions, **@ExceptionHandler** methods in **@ControllerAdvice** classes should be implemented
  - For Controller-specific exceptions, **@ExceptionHandler** methods should be added alongside the actions

Summary

- **Error Handling** is essential
  - Improves **User Experience**
  - Improves **Application maintenance**
- **Error Handling** in **Spring**
  - **HTTP Error Response Status Codes**
  - **@ExceptionHandler** methods
  - **@ControllerAdvice** global handlers
- When to use What?
- Additional Error Handling information here

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

Software University

SoftUni Organizational Partners

Software University

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license