Spring Validation & Interceptor

Instructor:

Table Content

Learning Goals

After the course, attendees will be able to:

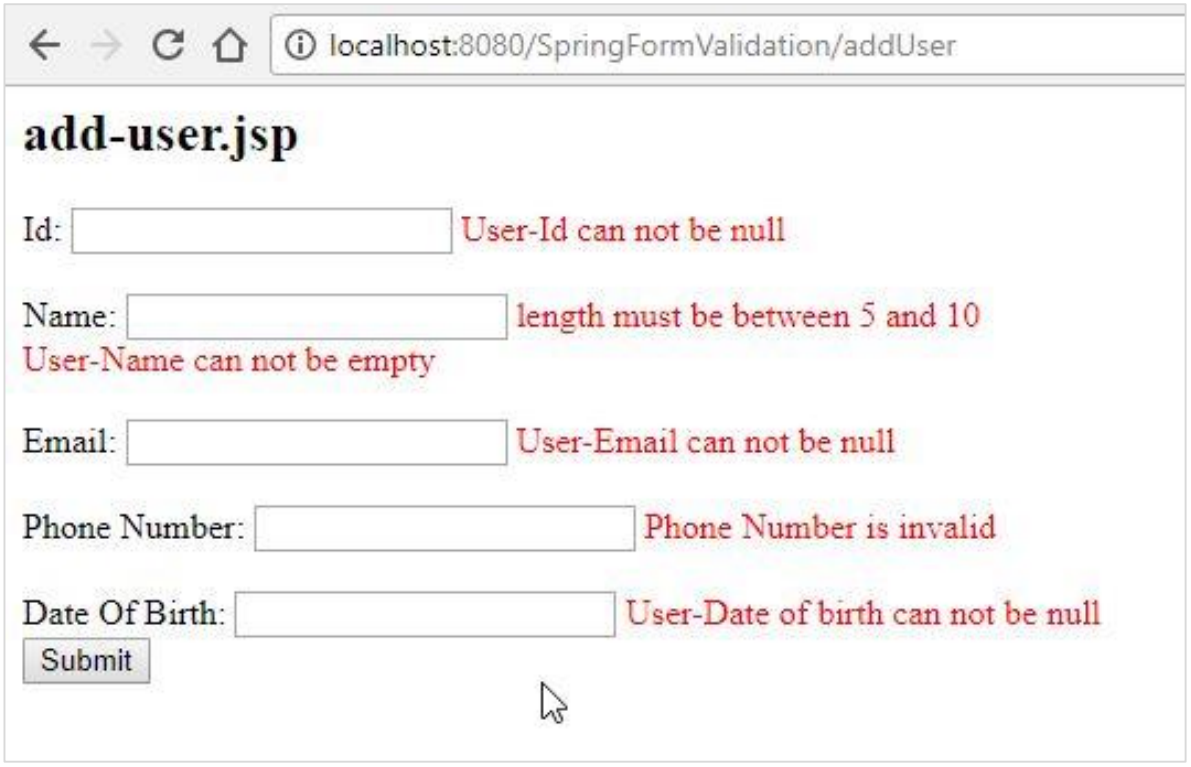
- Understand the importance of validation
- Implement validation in Spring MVC
- Understand how Interceptor works
- Create and use Interceptor
- Protect private resources interceptor application to

Section 1

FORM VALIDATION USING JAVA BEAN VALIDATION API

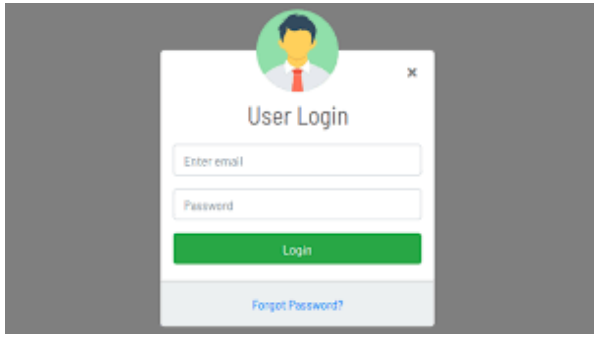
FORM VALIDATION

- To make sure the data inputted from users, the software must check inputted data is valid or not. Spring provides vary and easy way to validation user input.
- Form Validation using JSR-303 validation annotations, hibernate-validators, providing internationalization support using MessageSource.



What is validation in web aplicat

- ✓ Is pre-processing to make sure the application recived corrected data



User input data

Java script validator

Spring form-validation

Controller

```
index.htm*
Code  Split  Design  Title:
<script type="text/javascript">
  function verify(){
    // set some vars
    var user = document.getElementById("user");
    var pass = document.getElementById("pass");
    var msg = "";

    if(user.value == ""){
      msg+= "Please enter a username ! \n";
      user.className = "inpBoxError";
    }

    if(pass.value == ""){
      msg+= "Please enter a password ! \n";
      pass.className = "inpBoxError";
    }

    //
  }
```

spring form-validation

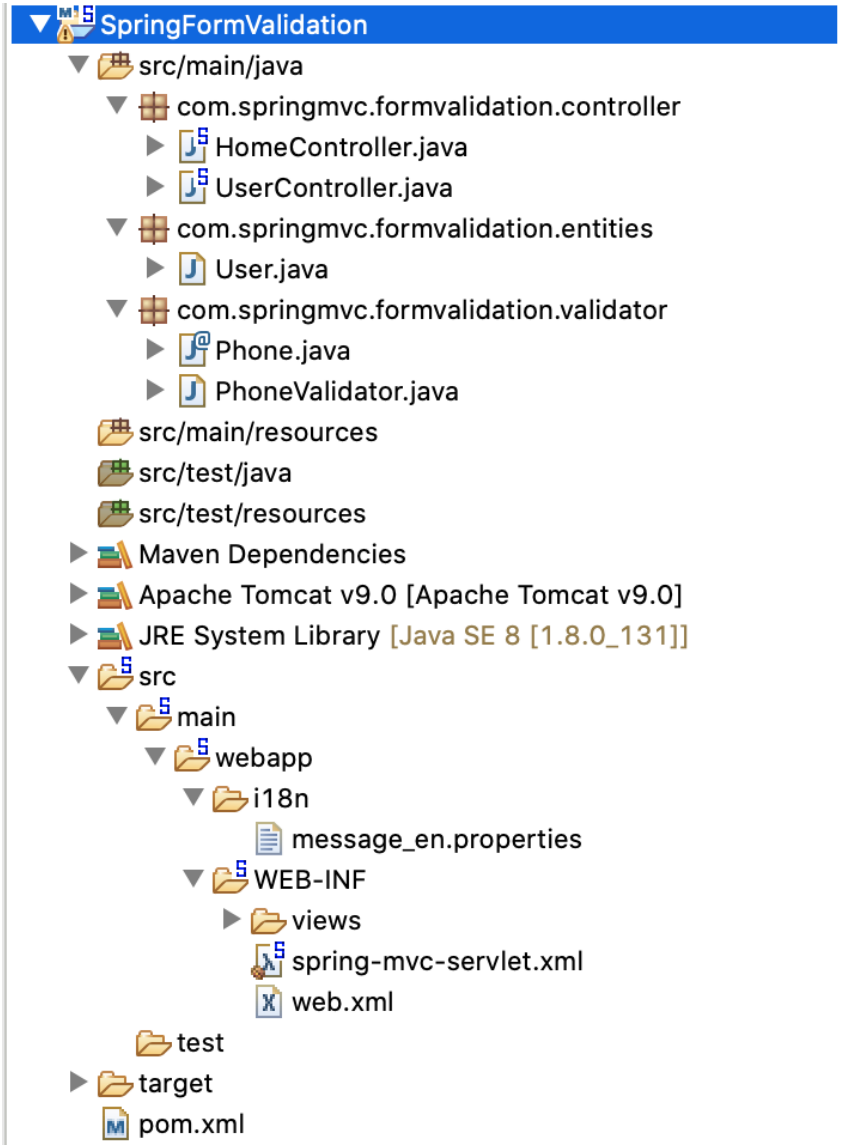
FORM VALIDATION

- Invalid input will cause unpredictable errors. Therefore, control of input data always plays an important role in the application.
- Common errors
  - ✓ Leave the input box blank ...
  - ✓ Incorrect email format, creditcard, url ...
  - ✓ Incorrect type of integer, real number, date and time ...
  - ✓ Minimum value, maximum value, within ...
  - ✓ Unlike passwords, correct captcha, identical code
  - ✓ Not as expected of some calculations ...

Form Validation Advantage

Security	• Prevent user enter wrong value to system. For example: user not allowed to wrong email look like **info@fsoft.com.vn**
Friendly to user	• User will be suggested the correct value

How to do?


SpringFormValidation
src/main/java
com.springmvc.formvalidation.controller
HomeController.java
UserController.java
com.springmvc.formvalidation.entities
User.java
com.springmvc.formvalidation.validator
Phone.java
PhoneValidator.java
src/main/resources
src/test/java
src/test/resources
Maven Dependencies
Apache Tomcat v9.0 [Apache Tomcat v9.0]
JRE System Library [Java SE 8 [1.8.0_131]]
src
main
webapp
i18n
message_en.properties
WEB-INF
views
spring-mvc-servlet.xml
web.xml
test
target
pom.xml

(1) Required Maven Dependency

- **Add maven dependency in pom.xml file.**

spring-mvc-servlet.xml | web.xml | SpringFormValidation/pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>springmvc</groupId>
    <artifactId>SpringFormValidation</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <spring.version>5.0.2.RELEASE</spring.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>1.1.0.Final</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>5.4.2.Final</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
    </dependencies>
</project>
```

(2) XML Config

❖ spring-mvc-servlet.xml

spring-mvc-servlet.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
4      xmlns:mvc="http://www.springframework.org/schema/mvc"
5      xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
6          http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
7          http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd">
8
9      <!-- Enables the Spring MVC @Controller programming model -->
10     <mvc:annotation-driven />
11     <context:component-scan base-package="com.springmvc.formvalidation" />
12
13     <bean
14         class="org.springframework.web.servlet.view.InternalResourceViewResolver">
15         <property name="prefix">
16             <value>/WEB-INF/views/jsp/</value>
17         </property>
18         <property name="suffix">
19             <value>.jsp</value>
20         </property>
21     </bean>
22
23     <bean id="messageSource"
24         class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
25         <property name="basename" value="/i18n/message" />
26     </bean>
27
28 </beans>
29
```

You make sure that <mvc:annotaion-drive/> is existed

Entity class

- User set validator annotaion to entity/bean

Define a validator

Or use existed one from hibernate

PhoneValidator.java

```
public class PhoneValidator implements ConstraintValidator<Phone, String> {

  public void initialize(Phone paramA) {
  }

  public boolean isValid(String phoneNo, ConstraintValidatorContext ctx) {
    if (phoneNo == null) {
      return false;
    }
    return phoneNo.matches("\\d{10}");
  }

}
```

Phone.java

```
@Documented
@Constraint(validatedBy = PhoneValidator.class)
@Retention(RUNTIME)
@Target({ FIELD, METHOD })
public @interface Phone {
  String message() default "{Phone}";

  Class<?>[] groups() default {};

  Class<? extends Payload>[] payload() default {};
}
```

Phone validator

```
public class User {

  @NotNull(message = "Id may not be null")
  private Integer id;

  @NotBlank
  @Length(min = 5, max = 10)
  private String name;

  @NotBlank
  @Email
  private String email;

  @NotNull
  @DateTimeFormat(pattern = "dd/MM/yyyy")
  @Past
  private Date dateOfBirth;

  @Phone(message = "Phone Number is invalid")
  private String phoneNumber;

  // getter/setter
}
```

Controller

UserController.java

```
@Controller
public class UserController {

  @RequestMapping(value = "/addUser", method = RequestMethod.GET)
  public String doGetAddUser(Model model) {
    if (!model.containsAttribute("user")) {
      model.addAttribute("user", new User());
    }
    return "add-user";
  }

  @RequestMapping(value = "/addUser", method = RequestMethod.POST)
  public String doPostAddUser(@ModelAttribute("user") @Valid User user, BindingResult result)
{
    if (result.hasErrors()) {
      return "add-user";
    }
    return "view-user";
  }

}
```

In view

add-user.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<html>
<head>
<style>
  .error {
    color: red;
  }
</style>
</head>

<body>
  <h2>add-user.jsp</h2>
  <form:form action="addUser" method="POST" modelAttribute="user">
    Id: <form:input path="id"/> <form:errors path="id" cssClass="error"/> <br/><br/>
    Name: <form:input path="name"/> <form:errors path="name" cssClass="error"/> <br/><br/>
    Email: <form:input path="email"/> <form:errors path="email" cssClass="error"/> <br/><br/>
    Phone Number: <form:input path="phoneNumber"/> <form:errors path="phoneNumber"
cssClass="error"/> <br/><br/>
    Date Of Birth: <form:input path="dateOfBirth"/> <form:errors path="dateOfBirth"
cssClass="error"/> <br/>
     <button type="submit">Submit</button>
  </form:form>
</body>
</html>
```

Section 2

SPRING MVC EXCEPTION HANDLING

Using XML configuration

- This approach uses XML to configure exceptions handling declaratively. Consider the following bean declaration in Spring's application context file:

```
<bean class="org.springframework.web.servlet.handler.
                                        SimpleMappingExceptionResolver">
   <property name="exceptionMappings">
     <props>
          <prop key="java.lang.ArithmeticException">MathError</prop>
     </props>
   </property>
</bean>
```

- That will map any exceptions of type **java.lang.ArithmeticException** (or *its sub types*) to the view named MathError.
- During execution of a Spring controller, if such an exception is thrown, the client will be redirected to the mapped view: **/views/MathError.jsp:**

```
<bean id="viewResolver"
     class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
</bean>
```

Using XML configuration

- **Controller**:

```
@Controller
@RequestMapping("/doMath")
public class MathController {
    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView calculateSum(@RequestParam int a, @RequestParam int b) {
        ModelAndView model = new ModelAndView("MathResult");

        model.addObject("sum", (a + b));
        model.addObject("subtract", (a - b));
        model.addObject("multiply", (a * b));
        model.addObject("divide", (a / b));

        return model;
    }
}
```

- In this code, there are two possible exceptions:
  - ✓ Either a or b is not a number.
  - ✓ b is zero, so the operation a / b will throw a java.lang.ArithmeticException exception.

Using XML configuration

- Following is code of the **MathResult.jsp** page:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Math Result</title>
</head>
<body>
    <center>
        <b>Sum: ${sum} </b><br/>
        <b>Subtract: ${subtract} </b><br/>
        <b>Multiply: ${multiply} </b><br/>
        <b>Divide: ${divide} </b><br/>
    </center>
</body>
</html>
```

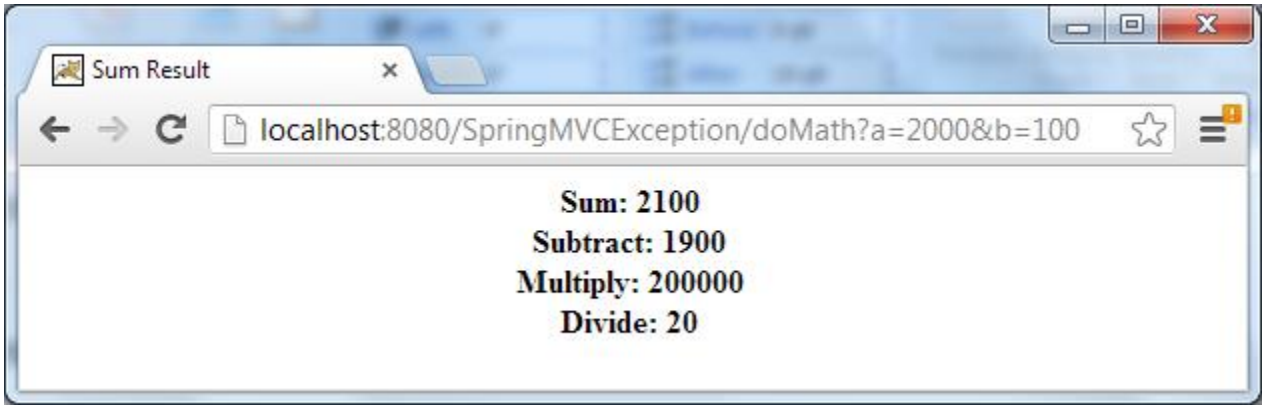
Using XML configuration

- Code of the **MathError.jsp** page:

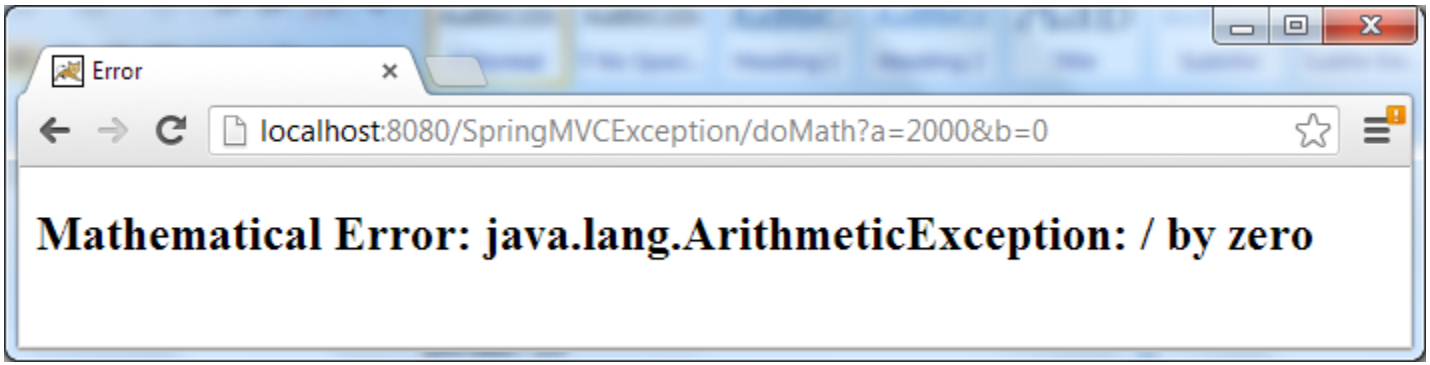
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Error</title>
</head>
<body>
<h2>
      Mathematical Error: ${exception} <br />
</h2>
</body>
</html>
```

Using XML configuration

- Output when testing the application with two numbers **a = 2000** and **b = 100**:



- If we pass b = 0, then the MathError.jsp page will be displayed:



Using exception handler method

- This approach uses the **@ExceptionHandler** annotation to annotate a method in controller class to handle exceptions raised during execution of the controller's methods.
- **Consider the following controller class:**

```
@Controller
public class FileUploadController {

    @RequestMapping(value = "/uploadFile", method = RequestMethod.GET)
    public String doFileUpload(@RequestParam int a) throws IOException, SQLException {

        // handles file upload stuff...
        if (a == 1) {
            throw new IOException("Could not read upload file.");
        } else if (a == 2) {
            throw new SQLException("Database exception!!!");
        }

        return "done";
    }

    @ExceptionHandler({IOException.class, java.sql.SQLException.class})
    public ModelAndView handleIOException(Exception ex) {
        ModelAndView model = new ModelAndView("IOError");

        model.addObject("exception", ex.getMessage());

        return model;
    }
}
```

Using exception handler method

- The method **doFileUpload()** may throw an **IOException**, and the handler method is declared as follows:

```
@ExceptionHandler(IOException.class)
public ModelAndView handleIOException(IOException ex) {
    ModelAndView model = new ModelAndView("IOError");

    model.addObject("exception", ex.getMessage());

    return model;
}
```

- This **handleIOException()** method will be invoked whenever an exception of type **java.io.IOException** (or *its sub types*) is raised within the controller class.
- Spring will pass the exception object into the method's argument.
- It's also possible to specify a list of exception classes in the **@ExceptionHanlder** annotation, for example:
  -

```
ExceptionHandler({IOException.class, java.sql.SQLException.class})
```

Spring @ExceptionHandler Global exception handli

- If we want to centralize the exception handling logic to one class which is capable to handle exceptions thrown from any handler class/ controller class – then we can use **@ControllerAdvice** annotation.
- **Example**:
  - ✓ (1) **Custom Exception Class:** Let's create a simple custom exception class. In this class, we have defined the "error code" and "error message" member variables for specifying the user-defined error messages

```
@Component
public class MyException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    private String errCode;
    private String errMsg;

    public MyException() {
    }
    // getter and setter methods
}
```

(2) Global Exception Handler Class:

```
@ControllerAdvice
public class ExceptionControllerAdvice {

    @ExceptionHandler(MyException.class)
    public ModelAndView handleMyException(MyException mex) {

        ModelAndView model = new ModelAndView();
        model.addObject("errCode", mex.getErrCode());
        model.addObject("errMsg", mex.getErrMsg());
        model.setViewName("error/generic_error");
        return model;
    }

    @ExceptionHandler(Exception.class)
    public ModelAndView handleException(Exception ex) {

        ModelAndView model = new ModelAndView();
        model.addObject("errMsg", "This is a 'Exception.class' message.");
        model.setViewName("error/generic_error");
        return model;

    }
}
```

- ✓ ***Notes***: generic_error is a view name. ***Ex***: *error/generic_error.jsp*.

❖ (3) Controller Class:

```
@Controller
public class ExceptionCtrl {

    @RequestMapping(value = "/exception/{type}", method = RequestMethod.GET)
    public String exception(@PathVariable(name = "type") String exception)
            throws IOException {

        if (exception.equalsIgnoreCase("error")) {
            throw new MyException("A1001",
                    "This is a custom exception message.");
        } else if (exception.equalsIgnoreCase("io-error")) {
            throw new IOException();
        } else {
            return "success";
        }
    }
}
```

- ✓ If the user provides a **/error** request, the controller method will throw the **MyException**, and the ExceptionControllerAdvice.handleMyException() method will be invoked to handle the exception.
- ✓ If the user provides a **/io-error** request, the controller method throw the **IOException**, and the ExceptionControllerAdvice.handleException() method will be invoked to handle the exception.

@ResponseStatus

- You also can throw an Exception declared with @ResponseStatus annotation:
  - ✓ (1) I created a class named **ResourceNotFoundException** that extends from **RuntimeException** and I putted this annotation over the class definition:

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {

}
```

  - ✓ (2) I created this method in my exception's controller class:

```
@ControllerAdvice
public class ExceptionControllerAdvice {

@ExceptionHandler(ResourceNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public String handleResourceNotFoundException() {

        return "notFoundJSPPage";
    }
}
```

@ResponseStatus

❖ Controller class example:

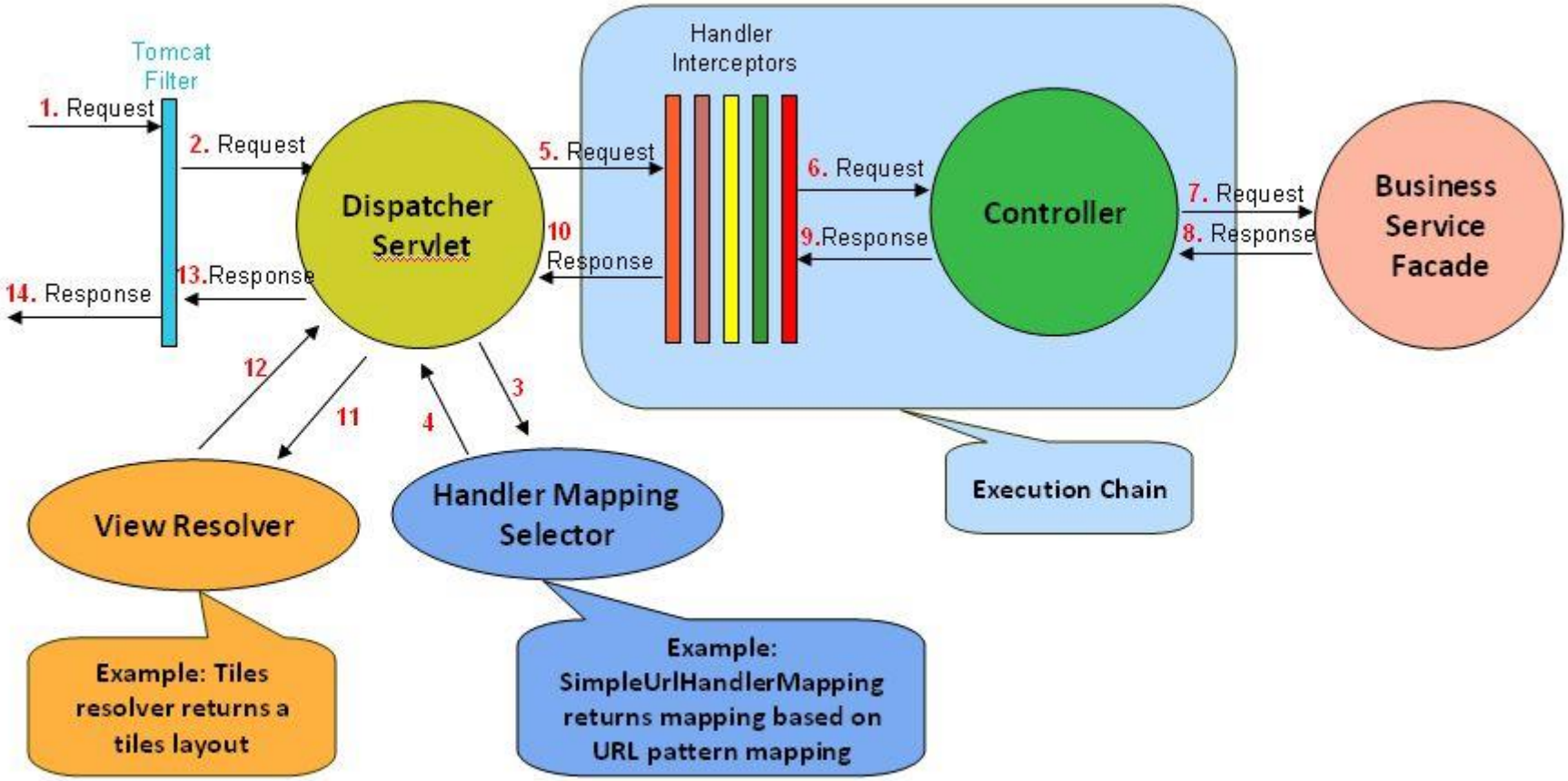
```
@Controller
public class ExceptionCtrl {

    @RequestMapping(value = "/exception/{type}", method = RequestMethod.GET)
    public String exception(@PathVariable(name = "type") String exception)
            throws IOException {

        if (exception.equalsIgnoreCase("error")) {
            throw new MyException("A1001", "This is a custom exception message.");
        } else if (exception.equalsIgnoreCase("io-error")) {
            throw new IOException();
        } else if (exception.equalsIgnoreCase("404")) {
            throw new ResourceNotFoundException();
        } else {
            return "success";
        }
    }
}
```

Section 3

SPRING INTERCEPTOR

What is SPRING interceptor?

- Spring Interceptor are used to intercept client requests and process them



Interceptor structure

derived class from **HandlerInterceptorAdapter**

```
public class MyInterceptor extends HandlerInterceptorAdapter{

        @Override
        public boolean preHandle(HttpServletRequest request,
                        HttpServletResponse response, Object handler) throws Exception {
                return true;
        }

        @Override
        public void postHandle(HttpServletRequest request,
                        HttpServletResponse response, Object handler,
                        ModelAndView modelAndView) throws Exception {
        }

        @Override
        public void afterCompletion(HttpServletRequest request,
                        HttpServletResponse response, Object handler, Exception ex)
                        throws Exception {
        }
}
```

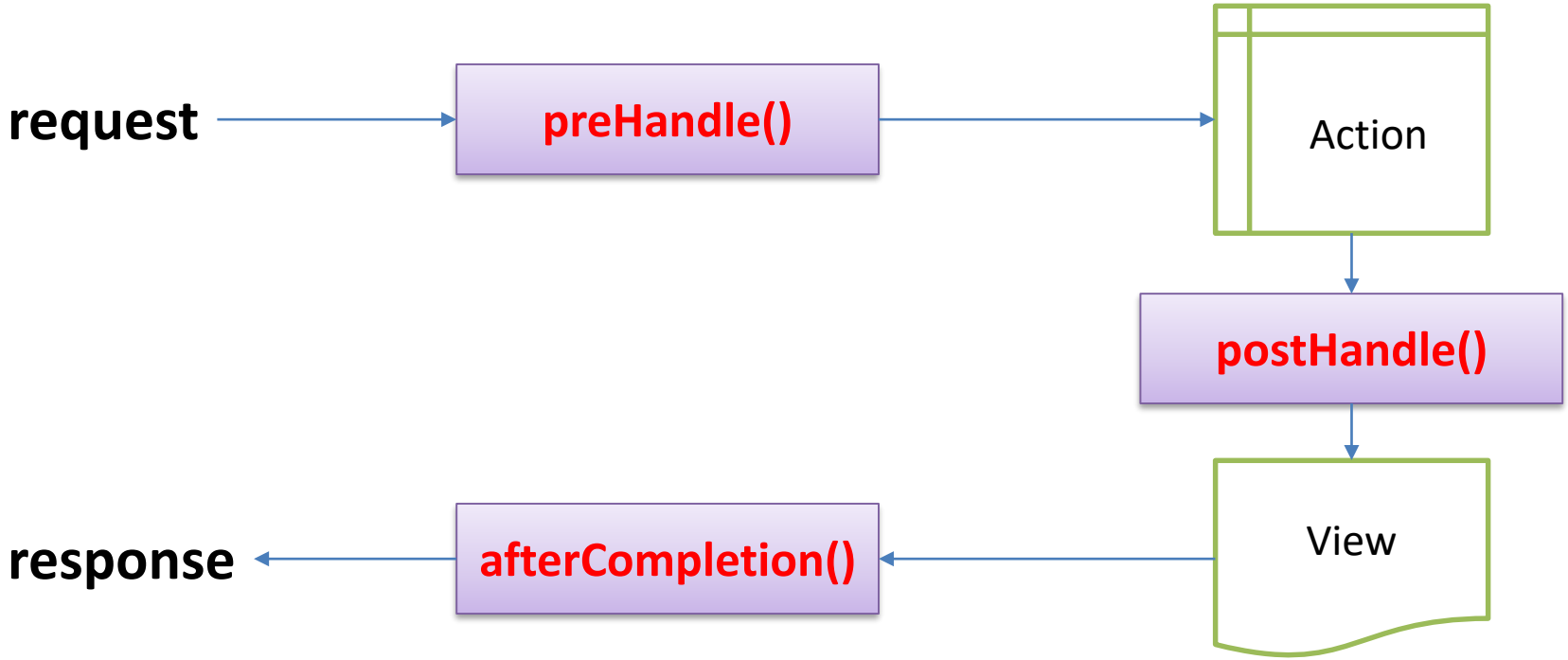
Run **BEFORE** action method

false will not run action method

Run **AFTER** action method and **BEFORE** view

Run **AFTER** view

Interceptor process



- If we want to handle a task before the action executed, we have to write code in **preHandle**.
- If we want to prepare something for View, we write code in **postHandle** method.

An Example of interceptor

- Create class and extend HandlerInterceptorAddaptor

```
public class LoggerInterceptor extends HandlerInterceptorAdapter{
	@Override
	public boolean preHandle(HttpServletRequest request,
			HttpServletResponse response, Object handler) throws Exception {
		System.out.println("LoggerInterceptor.preHandle()");
		return true;
	}

	@Override
	public void postHandle(HttpServletRequest request, HttpServletResponse response,
			Object handler,  ModelAndView modelAndView) throws Exception {
		System.out.println("LoggerInterceptor.postHandle()");
	}

	@Override
	public void afterCompletion(HttpServletRequest request,
			HttpServletResponse response, Object handler, Exception ex) throws Exception {
		System.out.println("LoggerInterceptor.afterCompletion()");
	}
}
```

Configure interceptor

- Interceptors, after being built, need to config in Spring configuration file (.xml)
- The following declaration **LoggerInterceptor** will filter all actions

```
<mvc:interceptors>
   <mvc:interceptor>
     <mvc:mapping path="/**" />
     <bean class="com.springmvc.LoggerInterceptor" />
   </mvc:interceptor>
 </mvc:interceptors>
```

Implementing controller and view

❖ Controller class:

```java
@Controller
@RequestMapping("/home/")
public class HomeController{
	@RequestMapping("index")
	public String index() {
		System.out.println("HomeController.index()");
		return "home/index";
	}
	@RequestMapping("about")
	public String about() {
		System.out.println("HomeController.about()");
		return "home/index";
	}
	@RequestMapping("contact")
	public String contact() {
		System.out.println("HomeController.contact()");
		return "home/index";
	}
}
```

```jsp
<%@ page pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<title>Interceptor</title>
</head>
<body>
	<h1>Interceptor</h1>
	<%
		System.out.println("index.jsp");
	%>
</body>
</html>
```

Run the project and see the result

- **Run home / index.htm and see the output from Console:**

```
LoggerInterceptor.preHandle()
HomeController.index()
LoggerInterceptor.postHandle()
index.jsp
LoggerInterceptor.afterCompletion()
```

- **Through the results we see the order of execution**

preHandle()=>**index**()=>**postHandle**()=>**index.jsp**=>**afterCompletion**()

More: Interceptor configuration

- Sometimes Interceptor is built just to filter some actions, not all actions.
- The following configuration allows only LoggerInterceptor to filter **home/index.htm** and **home/about.htm** actions:

```
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/home/index.htm" />
     <mvc:mapping path="/home/about.htm" />
    <bean class="com.springmvc.LoggerInterceptor" />
  </mvc:interceptor>
</mvc:interceptors>
```

More: Interceptor configuration

- Another situation is that we want to filter all actions in the HomeController except for **home/ index.htm**:

```
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/home/**" />
     <mvc-exclude:mapping path="/home/about.htm" />
    <bean class="com.springmvc.LoggerInterceptor" />
  </mvc:interceptor>
</mvc:interceptors>
```

- Here we see that **<mvc: exclude-mapping>** is used to exclude unfiltered actions and ** is a symbol for any group of characters.

An issue when implement security

- The yellow actions of the following two controllers are only accessible after logging in

```
@Controller
@RequestMapping("/user/")
public class UserController{
    @RequestMapping("login")
    public String login() {...}
    @RequestMapping("logoff")
    public String logoff() {...}
    @RequestMapping("register")
    public String register() {...}
    @RequestMapping("activate")
    public String activate() {...}
    @RequestMapping("forgot-password")
    public String forgot() {...}
    @RequestMapping("change-password")
    public String change() {...}
    @RequestMapping("edit-profile")
    public String edit() {...}
}
```

```
@Controller
@RequestMapping("/order/")
public class OrderController{
    @RequestMapping("checkout")
    public String checkout() {...}
    @RequestMapping("list")
    public String list() {...}
    @RequestMapping("detail")
    public String detail() {...}
}
```

Resolve the issue

- Implement **SecurityInterceptor** to filters all actions of the two controllers except for those that do not fill in yellow.
- The **SecurityInterceptor** must run before requesting the action and will do the following:
  - ✓ Check that the session has an attribute named user or not? If not, redirect to **user/ login.htm**.
  - ✓ On user/login.htm, after successful login, you need to create a user attribute in the session.

Resolve the issue

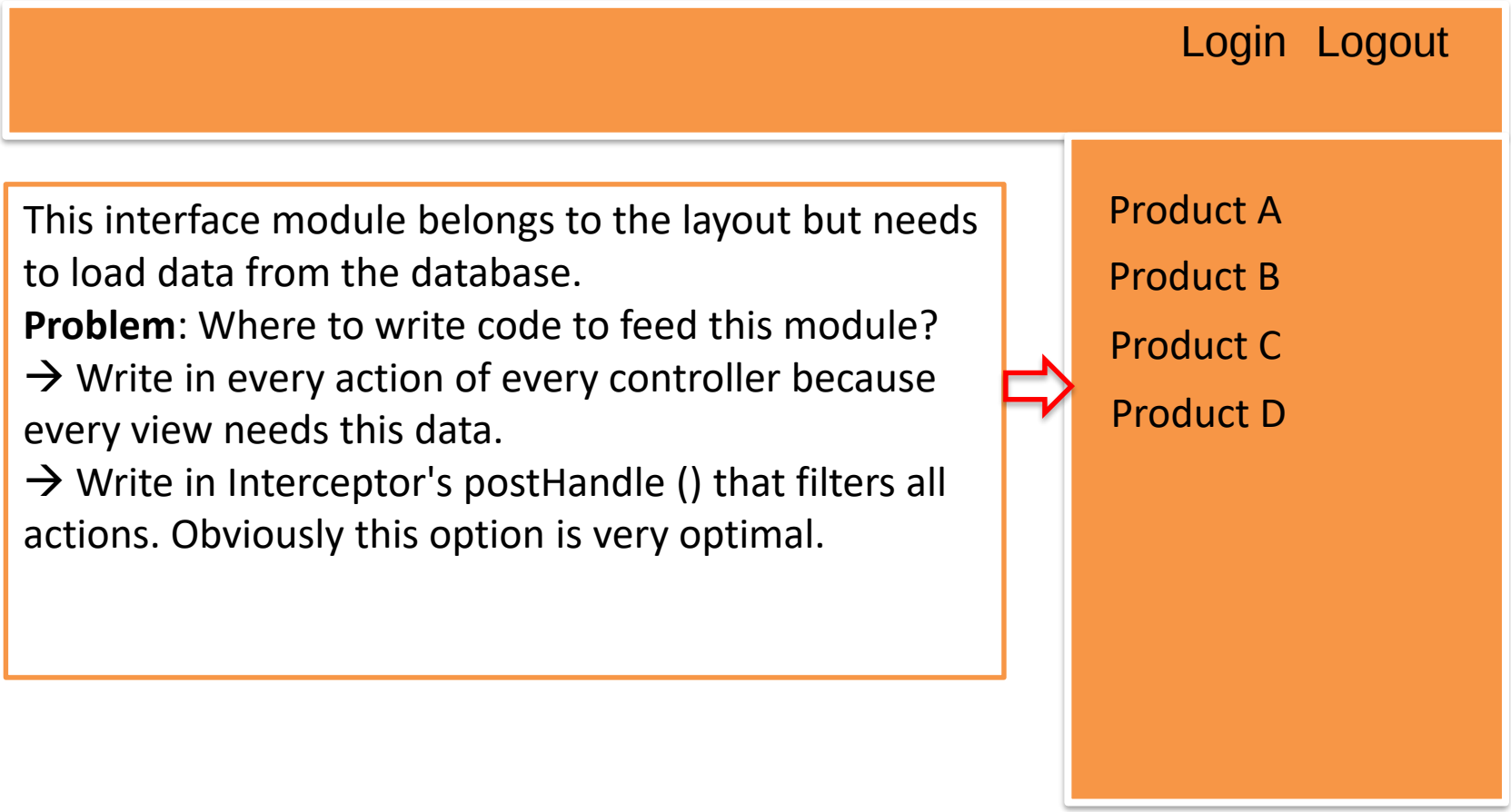
```java
public class SecurityInterceptor extends HandlerInterceptorAdapter{
    @Override
    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws Exception {
        HttpSession session = request.getSession();
        if(session.getAttribute("user") == null){
            response.sendRedirect(request.getContextPath() + "/user/login.htm");
            return false;
        }
        return true;
    }
}
```

```xml
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/user/**" />
    <mvc:mapping path="/order/**" />
     <mvc-exclude:mapping path="/user/login.htm" />
     <mvc-exclude:mapping path="/user/register.htm" />
     <mvc-exclude:mapping path="/user/forgot-passowrd.htm" />
     <mvc-exclude:mapping path="/user/active.htm" />
    <bean class="com.springmvc.LoggerInterceptor" />
  </mvc:interceptor>
</mvc:interceptors>
```

Resolve the issue

- This interface module belongs to the layout but needs to load data from the database.
- **Problem**: Where to write code to feed this module?
  - ✓ Write in **every action of every controller** because every view needs this data.
  - ✓ Write in Interceptor's **postHandle()** that *filters all actions*. **Obviously this option is very optimal**.

Load shared data issue



Summary

- Study the importance of Validation
- Implement validator
- Handling Exception in Spring Web MVC
- Learn Interceptor
- Implement Interceptor
- Configure Interceptor to filter actions
- Interceptor application to protect privacy functions.

Thank you