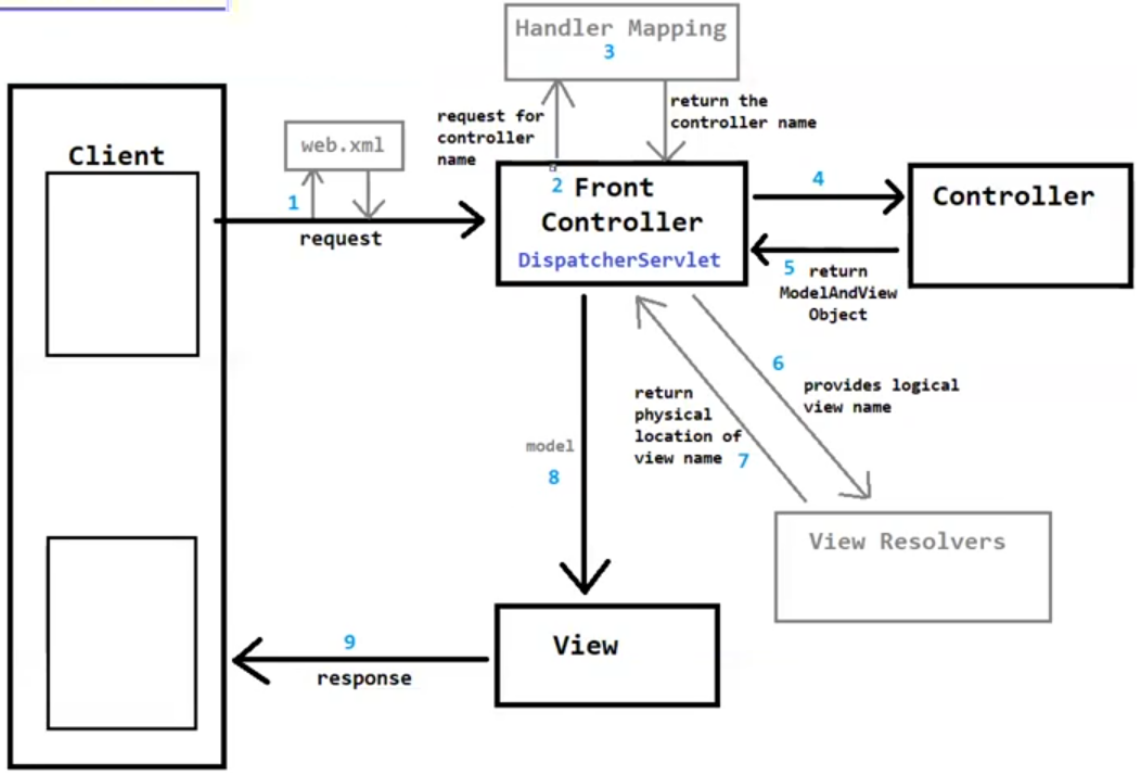
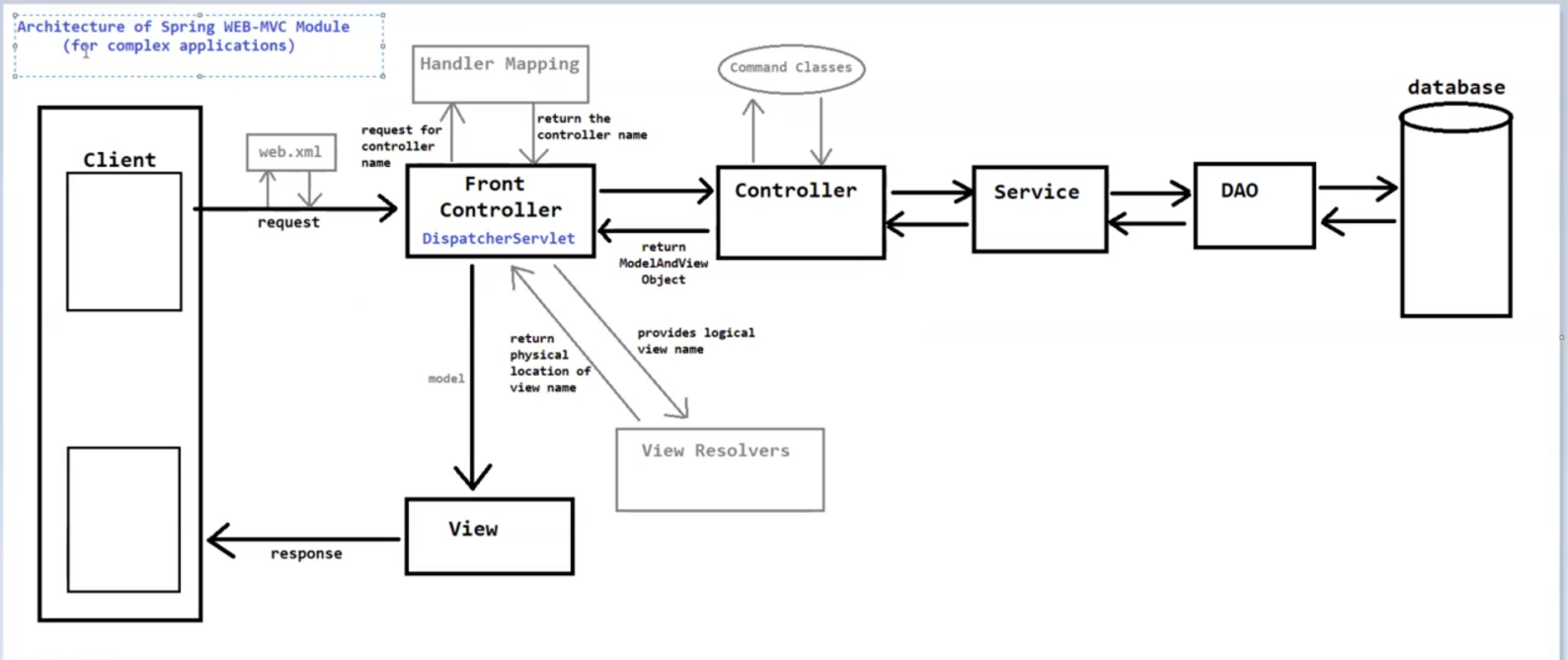
Architecture of Spring WEB-MVC Module




Architecture of Spring WEB-MVC Module
(for complex applications)
Handler Mapping
Command classes
database
Client
web.xml
request for
controller
name
return the
controller name
Front
Controller
DispatcherServlet
Controller
Service
DAO
request
return
ModelAndView
Object
return
physical
location of
view name
provides logical
view name
model
View Resolvers
View
response

=> Spring WEB Module :-
-> The Spring WEB Module is a part of the Sring Framework, designed for building flexible and loosly coupled web application
-> It provides a clean and organized way to develop web applications using Spring MVC or Spring WebFlux etc

-> NOTE :-
= Spring WEB-MVC Module is the part of Spring WEB Module. The Spring WEB Module provides the basic web-oriented integration features that the Spring WEB-MVC Module needs to operate

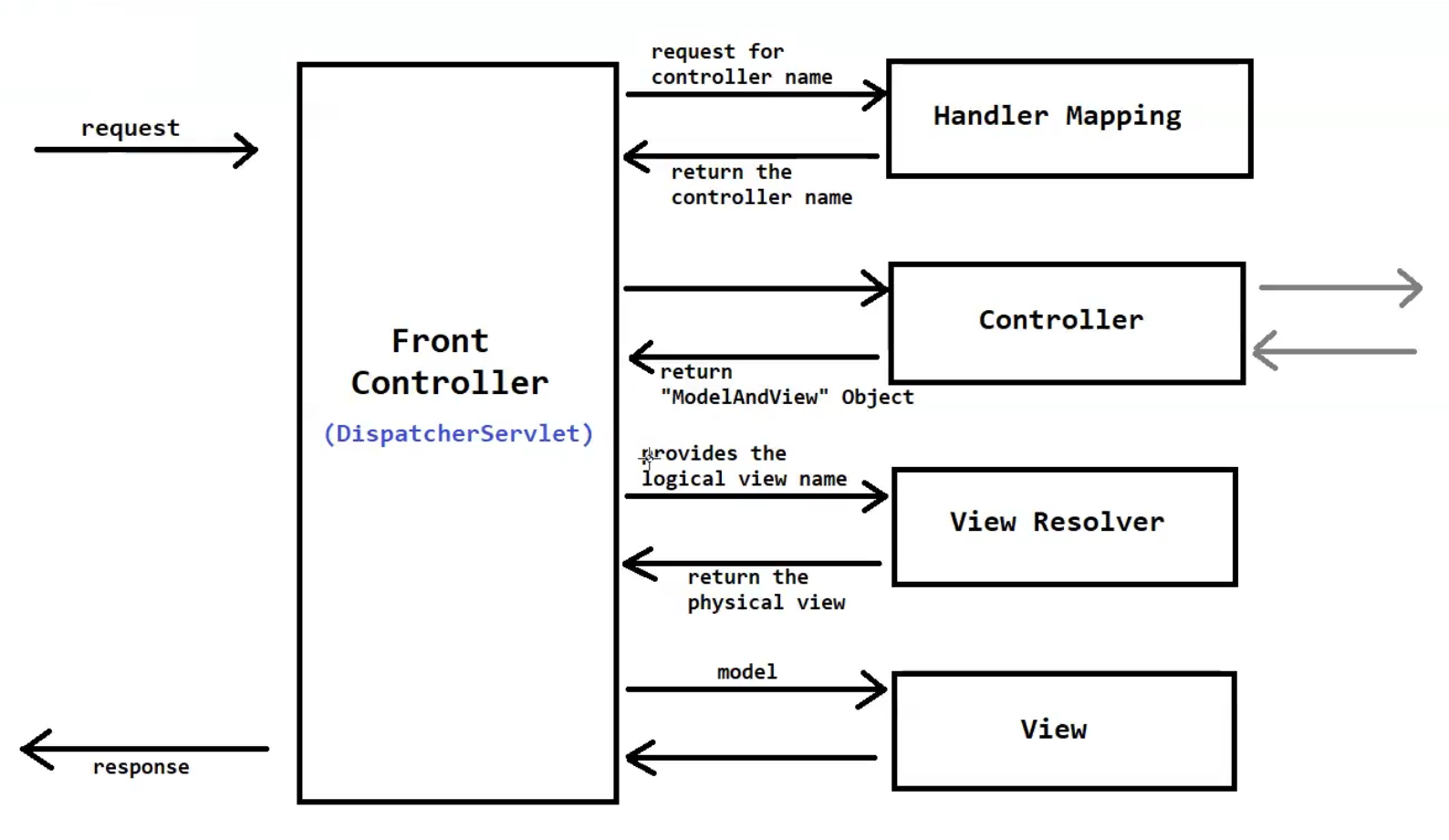
==========

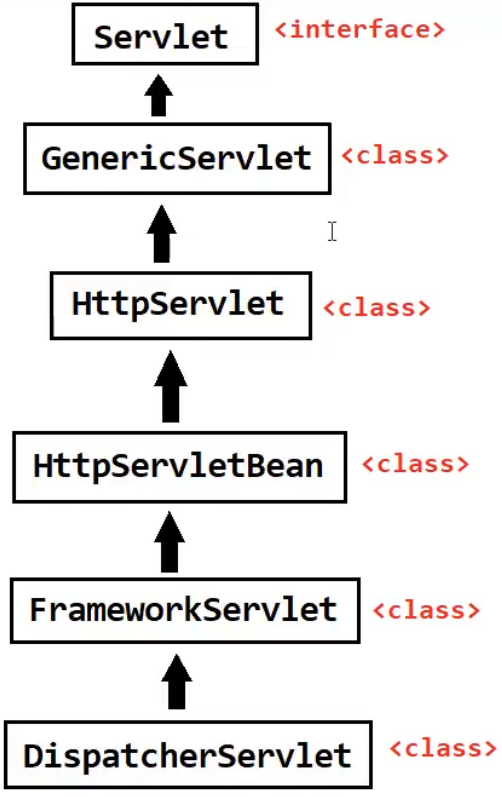
=> Spring WEB-MVC Module :-
-> The Spring WEB-MVC Module is used to build web applications following the Model-View-Controller (MVC) architectural/design pattern
-> It is used to develop flexible web applications in an organized manner

-> Architecture of Spring WEB-MVC Application :-

-> Components of Spring WEB-MVC Module :-
1. Front Controller (DispatcherServlet)
2. Handler Mapping
3. Controller
4. Model & ModelAndView
5. Command Classes
6. View Resolvers
7. View

==========


request
Front Controller
(DispatcherServlet)
request for controller name
Handler Mapping
return the controller name
Controller
return "ModelAndView" Object
provides the logical view name
View Resolver
return the physical view
model
View
response


Servlet
<interface>
GenericServlet
<class>
HttpServlet
<class>
HttpServletBean
<class>
FrameworkServlet
<class>
DispatcherServlet
<class>

=> Front Controller (DispatcherServlet) :-
    -> It is the controller that manages or handles all the client requests and delegates them to other components
    -> It acts as the single entry point for spring web application

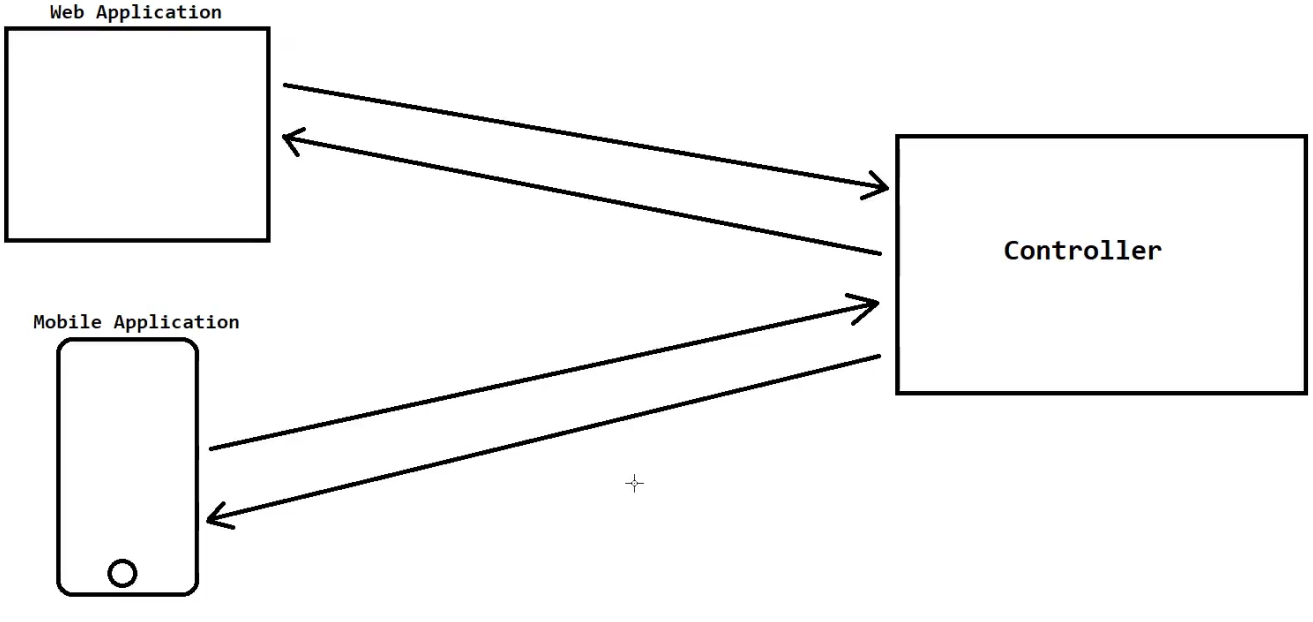
    -> Advantages of Front Controller :-
        1. Centralized Control :-
            = It is the single entry point for centralized control over request processing and common features like security, internationalization etc
        2. Flexibility :-
            = It is highly flexible and customizable request processing flow to meet specific application needs
        3. Separation of Concerns :-
            = It separates request processing logic for more modular and maintanable applications

    -> In Spring WEB-MVC, "DispatcherServlet" acts as the front controller
    -> Flow of DispatcherServlet in Spring WEB-MVC :-
        = Diagram
    -> Hierarchy of DispatcherServlet (class) :-
        = Diagram

=> Handler Mapping :-
-> It is used to map the requests to proper controller and returns that controller name to the front controller

-> For this, Spring WEB-MVC has provided "HandlerMapping" interface and its implemented classes
-> Some commonly used HandlerMapping implementations are :-
1. RequestMappingHandlerMapping (default)
2. SimpleUrlHandlerMapping
3. BeanNameUrlHandlerMapping
etc

-> How to provide handler mapping configurations :-
1. XML Configurations
2. Java Configurations
3. Annotations
4. Component Scanning
5. Default URL Mapping
-> NOTE :-
= The most common ways to provide handler mapping in Spring are using "annotations" and "component scanning". because they are simple and flexible


Web Application
Controller
Mobile Application

=> Controller :-
-> Controllers are the heart of Spring WEB-MVC Applications.
-> They are responsible for handling the incoming requests, executing business logic and returning response

-> Advantages of Controllers :-
1. Controllers separates the presentation layer from the business layer
2. Controllers can be reused acress different applications
3. Controllers are easy to test and maintain

-> We can create the controllers by 3 ways :-
1. By inheriting "Controller" interface or its implemented classes
2. By "@Controller" annotation (to create general purpose controller or for traditional web applications)
3. By "@RestController" annotation (to create RESTful controller)
-> NOTE :-
= Controller and its implemented classes has been deprecated from Spring 3.x version. So it is recommended to use annotations to create controllers
= For mapping we use annotations i.e. @RequestMapping, @GetMapping, @PostMapping etc (for 2 & 3rd point)

-> There are mainly 3 types of controllers :-
1. Simple Controllers
= These are the controllers that handle basic requests such as returning a static HTML page or displaying a list of data. They typically do not interact with other spring components i.e. services or respository
2. Form-Handling Controllers
= These are the controllers that handle the requests from HTML forms. They typically validate the form data and then perform some action such as saving the data into database or sending an email
3. RESTful Controllers
= These are the controllers that implement the RESTful API design pattern. Thet typically handle the requests for specific resources such as products, users, orders etc. RESTful controllers typically return the JSON data.

=> Model :-
-> It is an interface that represents a map of attributes that a controller cab use to pass data to the view
-> Model is typically used to send the dynamic data that needs to be displayed on the view

-> How to add data in model object :-
= addAttribute(String name, Object value)
= addAllAttributes(Map<String, Object> attributes)
-> How to retrieve data from model object :-
= getAttribute(String name)
= containsAttribute(String name)
= We can also use Expression Language

=> ModelAndView :-
-> It is a class that combines both the "Model" and "view name" into a single object
-> It allows the controller to specify which view should be used to render the response and what data should be available in that view

---

=> Command Classes :-
    -> Command Classes are normal JavaBean class or POJO class
    -> It is used to store the form data which is submmited by the client and then this data is available for business logic

=> View Resolvers :-
-> It is used to resolve or translates the "logical view name" returned by the controller into the "actual physical view" that should be rendered to the user
-> For example :-
>> Logical View Name : home
Actual Physical View : WEB-INF/views/home.jsp
>> Logical View Name : productDetails
Actual Physical View : WEB-INF/templates/productDetails.html
-> NOTE :-
= To get Actual Physical Name, postfix and prefix are added to the Logical View Name

-> For this Spring WEB-MVC has provided "ViewResolver" interface and its implemented classes
-> Some implemented classes are :-
1. InternalResourceViewResolver (default) - (used for JSP view)
2. ResourceBundleViewResolver
3. XmlViewResolver
4. BeanNameViewResolver
5. URLBasesViewSolver
6. ThymeleafViewResolver (used for Thymeleaf view)
7. VelocityViewResolver (used for Velocity view)
8. FreeMarkerViewResolver (used for FreeMarker view)

=> View :-
    -> It is the presentation or UI which is sent to the client as a response
    -> Some common view technologies are HTML, JSP, Thymeleaf, Velocity, FreeMarker etc

---

=> web.xml file :-
    -> It is the "deployment descriptor" file which is the part of every JavaEE application
    -> Some of the responsibilites of web.xml file are :-
        1. welcome file configurations
        2. servlets configurations
        3. session timeout configurations
        4. filters configurations
        5. listeners configurations
        6. context parameters configurations
        7. error page configurations
        etc

    -> In Spring WEB-MVC, its main task is to configure the front controller i.e. DispatcherServlet
    -> We create web.xml file in WEB-INF folder

=> Spring Configuration XML File :-
    -> It is an XML file which is used to configure :-
        1. bean classes
        2. handler mapping classes
        3. controller classes
        4. view resolver classes
        etc

    -> The default name of Spring Configuration File is "[servlet-name]-servlet.xml"
    -> We create this file in WEB-INF folder

```
-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN (doctype with catalog)
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
    <servlet>
        <servlet-name>myds</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>myds</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/b
        http://www.springframework.org/schema/context http://www.springframework.org/schema/

</beans>
```

=> Steps to create String WEB-MVC Application 1 (Old Approach) :-

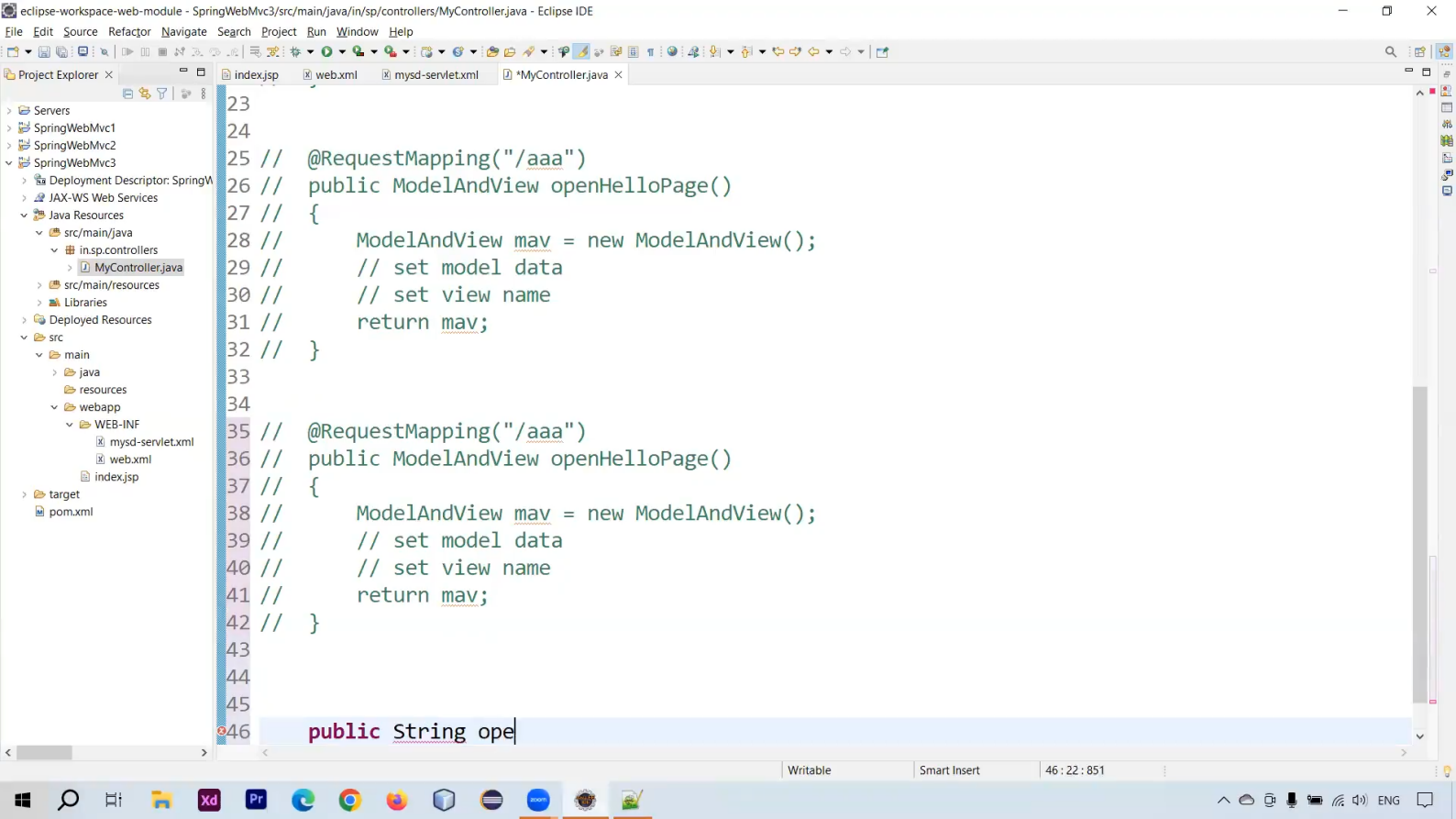
1. Provide dependencies:
   >> Include the necessary Spring MVC dependencies in the pom.xml file i.e.
      = spring-core
      = spring-context
      = spring-expression
      = spring-webmvc
2. Create index.jsp page:
   >> Create index.jsp page that contains a hyperlink to trigger the request
3. Configure DispatcherServlet in web.xml:
   >> Define a servlet mapping in web.xml file to map incoming requests to the DispatcherServlet
4. Create Spring Configuration XML File:
   >> Create an XML file (i.e. myds-servlet.xml) and configure the Spring WEB-MVC Application as follows:
      4.1 Define handler mapping:
         = Specify a BeanNameUrlHandlerMapping to map URL pattern to the specific controller.
      4.2 Configure URL with controller:
         = Map the URL pattern to a specific controller bean
      4.3 Configure View Resolver:
         = Use InternalResourceViewResolver to resolve view names to actual JSP pages
5. Create controller:
   >> Implements the Controller interface to handle the incoming requests and prepare the data for the view
6. Create JSP page for response:
   >> Create JSP page (i.e. hello.jsp) to display the data prepared by the controller

=> Steps to create String WEB-MVC Application 2 (New Approach) :-

1. Provide dependencies:
   >> Include the necessary Spring MVC dependencies in the pom.xml file i.e.
      = spring-core
      = spring-context
      = spring-expression
      = spring-webmvc
2. Create index.jsp page:
   >> Create index.jsp page that contains a hyperlink to trigger the request
3. Configure DispatcherServlet in web.xml:
   >> Define a servlet mapping in web.xml file to map incoming requests to the DispatcherServlet
4. Create Spring Configuration XML File:
   >> Create an XML file (i.e. myds-servlet.xml) and configure the Spring WEB-MVC Application as follows:
      4.1 Define handler mapping:
         = Specify a RequestMappingHandlerMapping to map URL pattern to the specific controller.
      4.2 Configure View Resolver:
         = Use InternalResourceViewResolver to resolve view names to actual JSP pages
5. Create controller:
   >> Create a controller to handle incoming requests and prepare data for the view with following steps:
      5.1 Annotate the class with @Controller
      5.2 Define handler methods annotated with @RequestMapping to handle the specific URL pattern
      5.3 The handler method can return ModelAndView object to specify the view name and data to be passed to the view
6. Create JSP page for response:
   >> Create JSP page (i.e. hello.jsp) to display the data prepared by the controller

=> Difference between Old and New Approach :-

1. Handler Mapping :-
   - = Old Approach : BeanNameUrlHandlerMapping
   - = New Approach : RequestMappingHandlerMapping
2. Controller Defination :-
   - = Old Approach : Separate controller beans in Spring Configuration XML File
   - = New Approach : Methods annotated with @RequestMapping
3. Flexibility :-
   - = Old Approach : Less Flexible
   - = New Approach : More Flexible
4. Ease to use :-
   - = Old Approach : Less easy to use
   - = New Approach : More easy to use


eclipse-workspace-web-module - SpringWebMvc3/src/main/java/in/sp/controllers/MyController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Servers
SpringWebMvc1
SpringWebMvc2
SpringWebMvc3
Deployment Descriptor: SpringW
JAX-WS Web Services
Java Resources
src/main/java
in.sp.controllers
MyController.java
src/main/resources
Libraries
Deployed Resources
src
main
java
resources
webapp
WEB-INF
mysd-servlet.xml
web.xml
index.jsp
target
pom.xml
index.jsp
web.xml
mysd-servlet.xml
*MyController.java
23
24
25 // @RequestMapping("/aaa")
26 // public ModelAndView openHelloPage()
27 // {
28 // ModelAndView mav = new ModelAndView();
29 // // set model data
30 // // set view name
31 // return mav;
32 // }
33
34
35 // @RequestMapping("/aaa")
36 // public ModelAndView openHelloPage()
37 // {
38 // ModelAndView mav = new ModelAndView();
39 // // set model data
40 // // set view name
41 // return mav;
42 // }
43
44
45
46 public String ope
Writable
Smart Insert
46 : 22 : 851
ENG

```
45 //  }
46
47
48 //  @RequestMapping("/aaa")
49 //  public String openHelloPage(int a)
50 //  {
51 //      return "hello";
52 //  }
53
54
55 //  @RequestMapping("/aaa")
56 //  public String openHelloPage(String name)
57 //  {
58 //      return "hello";
59 //  }
60
61
62      @RequestMapping("/aaa")
63      public String openHelloPage(HttpServletRequest request, HttpServletResponse response)
64      {
65          return "hello";
66      }
67 }
68
```

=> @Controller :-

-> It is stereotype annotation that indicates that a class serves the role of controller

-> Syntax :-

```
@Controller
class MyController
{

}
```

-> It indicates that the annotated class is responsible for handling the HTTP requests

-> This annotation is used to the MVC architectural pattern for building web applications

=> Handler Methods :-
-> These methods are also known as URL Handler Methods
-> These methods are responsible for processing the incoming requests, performing business logic and preparing model data to be rendered by view page
-> In order to map the incoming rendered or URL, handler methods can be annotated with HTTP methods related annotations i.e. @RequestMapping, @PostMapping, @GetMapping etc

-> Case studies for handler methods :-
1. We can provide any name for handler method
2. We can have any return type for handler method such as String (representing a view name) or ModelAndView (a container for model data and view name) etc
3. We can use non-primitive data type as a parameter for handler methods. However, primitive data types can be also used but they are typically used for simple cases like receiving query parameters

-> NOTE : From above points it is clear that the handler method is very flexible and thus this approach is mostly used

```
//  //@RequestMapping(value = "/aaa")
//  //@RequestMapping(value = {"/aaa", "/bbb", "/ccc"})
//  //@RequestMapping(value = "/aaa", method = RequestMethod.GET)
//  @RequestMapping(value = "/aaa", method = {RequestMethod.GET, RequestMethod.POST})
//  public String openHelloPage()
//  {
//      return "hello";
//  }
```

=> HTTP Methods Related Annotations :-
-> Some annotations related to HTTP Methods are :-
1. @RequestMapping
= This is a versatile annotation that can be used for handling various HTTP methods i.e. GET, POST etc
= It can be applied at the method level or class level
= Syntax :-
>> @RequestMapping("/url")
>> @RequestMapping(value = {"/url1", "/url2", "/url3"})
>> @RequestMapping(value = "/url", method = RequestMethod.POST) <<==>> @PostMapping("/url")
>> @RequestMapping(value = "/url", method = {RequestMethod.POST, RequestMethod.GET})
2. @GetMapping
= It is used for handling HTTP GET requests
= It's a specialized form of @RequestMapping for GET method
3. @PostMapping
= It is used for handling HTTP POST requests
= It's a specialized form of @RequestMapping for POST method
etc

eclipse-workspace-web-module - SpringWebMvc6SessionAttributes/src/main/java/in/sp/controllers/MyController.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- Servers
- SpringWebMvc1
- SpringWebMvc2
- SpringWebMvc3
- SpringWebMvc4
- SpringWebMvc5HttpSession
- SpringWebMvc6SessionAttributes
  - Deployment Descriptor: SpringW
  - JAX-WS Web Services
  - Java Resources
    - src/main/java
      - in.sp.controllers
        - MyController.java
    - src/main/resources
    - Libraries
  - Deployed Resources
  - src
    - main
      - java
      - resources
      - webapp
        - WEB-INF
          - views
            - first.jsp
            - second.jsp
          - myds-servlet.xml
          - web.xml
        - index.jsp
  - target
  - pom.xml

index.jsp | MyController.java | first.jsp | second.jsp

```
6  import org.springframework.web.bind.annotation.SessionAttributes;
7
8  @Controller
9  @SessionAttributes("m_name")
10 public class MyController
11 {
12     @GetMapping("/aaa")
13     public String openFirstPage(Model model)
14     {
15         //100 lines of code
16         String name = "Deepak";
17
18         model.addAttribute("m_name", name);
19
20         return "first";
21     }
22
23     @GetMapping("/bbb")
24     public String openSecondPage()
25     {
26         return "second";
27     }
28 }
29
```

Writable | Smart Insert | 9 : 19 [18]

ENG

eclipse-workspace-web-module - SpringWebMvc7SessionAttributes/src/main/java/in/sp/controllers/MyController.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

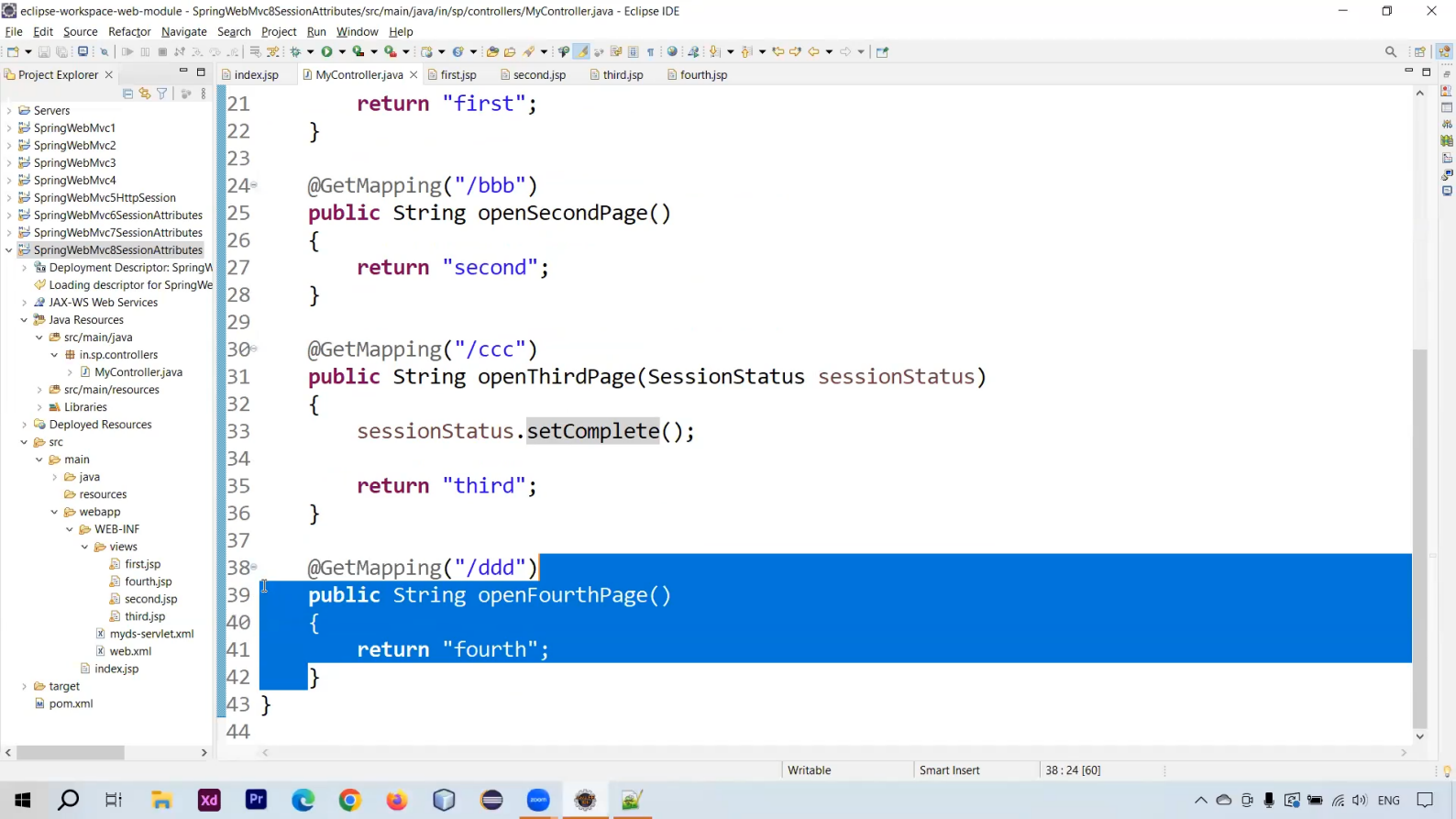
- Servers
- SpringWebMvc1
- SpringWebMvc2
- SpringWebMvc3
- SpringWebMvc4
- SpringWebMvc5HttpSession
- SpringWebMvc6SessionAttributes
- SpringWebMvc7SessionAttributes
  - Deployment Descriptor: SpringW
  - JAX-WS Web Services
  - Java Resources
    - src/main/java
      - in.sp.controllers
        - MyController.java
    - src/main/resources
    - Libraries
  - Deployed Resources
  - src
    - main
      - java
      - resources
      - webapp
        - WEB-INF
          - views
            - first.jsp
            - second.jsp
          - myds-servlet.xml
          - web.xml
        - index.jsp
  - target
  - pom.xml

index.jsp | MyController.java | first.jsp | second.jsp

```
3  import org.springframework.stereotype.Controller;
7
8  @Controller
9  @SessionAttributes({"m_name", "m_gender", "m_city"})
10 public class MyController
11 {
12     @GetMapping("/aaa")
13     public String openFirstPage(Model model)
14     {
15         //100 lines of code
16         String name = "Deepak";
17         String gender = "Male";
18         String city = "Chandigarh";
19
20         model.addAttribute("m_name", name);
21         model.addAttribute("m_gender", gender);
22         model.addAttribute("m_city", city);
23
24         return "first";
25     }
26
27     @GetMapping("/bbb")
28     public String openSecondPage()
29     {
```

Writable | Smart Insert | 9 : 50 : 310

ENG


eclipse-workspace-web-module - SpringWebMvc8SessionAttributes/src/main/java/in/sp/controllers/MyController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Servers
SpringWebMvc1
SpringWebMvc2
SpringWebMvc3
SpringWebMvc4
SpringWebMvc5HttpSession
SpringWebMvc6SessionAttributes
SpringWebMvc7SessionAttributes
SpringWebMvc8SessionAttributes
Deployment Descriptor: SpringW
Loading descriptor for SpringWe
JAX-WS Web Services
Java Resources
src/main/java
in.sp.controllers
MyController.java
src/main/resources
Libraries
Deployed Resources
src
main
java
resources
webapp
WEB-INF
views
first.jsp
fourth.jsp
second.jsp
third.jsp
myds-servlet.xml
web.xml
index.jsp
target
pom.xml
index.jsp
MyController.java
first.jsp
second.jsp
third.jsp
fourth.jsp
21 return "first";
22 }
23
24 @GetMapping("/bbb")
25 public String openSecondPage()
26 {
27 return "second";
28 }
29
30 @GetMapping("/ccc")
31 public String openThirdPage(SessionStatus sessionStatus)
32 {
33 sessionStatus.setComplete();
34
35 return "third";
36 }
37
38 @GetMapping("/ddd")
39 public String openFourthPage()
40 {
41 return "fourth";
42 }
43 }
44
Writable
Smart Insert
38 : 24 [60]
ENG

=> HttpSession :-
    -> HttpSession is a server-side mechanism in Java Servlets and JSP that allows web applicaitons to store and retrieve user-specific information across multiple requests
    -> It enables session management, aiding in maintaning state and user data during user's visit to website

==========

=> @SessionAttributes :-
    -> It is used to store the attributes in the session for specific handler conversation.
    -> A conversational session is a sequence of requests that are related to each other. For example, a shopping cart conversation might consists of adding items to cart, viewing the cart and checking out.
    -> Attributes that are stored using @SessionAttributes will be removed from the session once the handler indicates that the conversational session is complete

    -> This annotation is used to manage session attributes, simplifying the process of injecting, storing and accessing data within controller methods

    -> This annotation is mostly used at "class level"

==========4

=> Difference between HttpSession and @SessionAttributes :-
    1. HttpSession scope is for entire session
       @SessionAttributes scope is till handler's conversational session

    2. HttpSession will store the data for longer period of time
       @SessionAttributes will store the data for temporary purpose

==========

```xml
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>5.8.0</version>
</dependency>
```

=> Spring Form Tag Library :-
-> It is a collection of JSP tags that simplify the process of creating HTML forms with Spring MVC
-> Why we should use Spring Form Tag Library :-
1. Data Binding Support:
= The form tags help with data binding between form fields and Java Objects. This means that values entered into the form fields can be automatically bound to corresponding properties of Java Object
2. Validation Support:
= The form tag library integrates with the Spring Validation Framework or others allowing us to perform server-side validations easily
3. Internationalization (i18n):
= The form tag library supports internationalization by providing tags that allow us to display messages in different langauges based on the user locale
4. Improved Code Reability:
= The tags make our code more readable and easier to maintain
5. Reduce Development Time:
= The tags help us to develop web applications more quickly and easily
-> NOTE :
= The Spring Form Tag Library is a powerful tool that can help us to develop web applications more quickly and easily. It is a standard part of the Spring MVC Framework and used by many developers
= To use Spring Form Tag Library we have to add one dependency i.e. spring-security-taglibs

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
```

```
<form:form action="regForm" method="POST">
    Name : <form:input path="name"/> <br/> <br/>
    Email : <form:input path="email"/> <br/> <br/>
    Password : <form:password path="password"/> <br/> <br/>
    Gender : <form:radiobutton path="gender" label="Male" value="Male"/> <form:radiobutton
    City : <form:select path="city">
                <form:option value="Select City" label="Select City" />
                <form:option value="Chandigarh" label="Chandigarh" />
                <form:option value="Mumbai" label="Mumbai" />
                <form:option value="Pune" label="Pune" />
            </form:select> <br/> <br/>

    <input type="submit" value="Register" />
</form:form>
```

<form:form action="regForm" method="POST" modelAttribute="modelStd">

```java
@Controller
public class MyController
{
    @GetMapping("/regPage")
    public String openRegPage(Model model)
    {
        model.addAttribute("modelStd", new Student());
        return "register";
    }

    @PostMapping("/regForm")
    public String handleRegForm(Student std, Model model)
    {
        // insert into database code

        model.addAttribute("model_std", std);

        return "profile";
    }
}
```

```
//  @PostMapping("/regForm")
//  public String handleRegForm(Student std, Model model)
//  {
//      // insert into database code
//
//      model.addAttribute("model_std", std);
//
//      return "profile";
//  }


    @PostMapping("/regForm")
    public String handleRegForm(@ModelAttribute("model_std") Student std)
    {
        // insert into database code

        return "profile";
    }
}
```

```
//  @PostMapping("/regForm")
//  public String handleRegForm(@ModelAttribute("model_std") Student std)
//  {
//      // insert into database code
//
//      return "profile";
//  }


    @ModelAttribute
    public void addStdObject(Model model, Student std)
    {
        model.addAttribute("model_std", std);
    }

    @PostMapping("/regForm")
    public String handleRegForm()
    {
        // insert into database code

        return "profile";
    }
}
```

=> @ModelAttribute :-
    -> It is an annotation that is used to bind a method parameters or return value to a model attribute, facilatating data preparation for view or form handling
    -> How to use @ModelAttribute:-
        1. Parameter-Level Annotation
        2. Method-Level Annotation

=> Validations in Spring :-

-> Validations are the restrictions provided to the client while filling the form so that the data entered by the client is valid

-> Tyere are 2 types of validations :-

1. Client-Side Validations
   = Technologies used for Client-Side Validations are JavaScript, VB Script etc
2. Server-Side Validations
   = Technologies used for Server-Side Validations are Servlets, JSP, Spring, Webservices etc

-> We can achieve validations by multiple ways but below are important ways :-

1. Using "Validator" interface
2. Using JSR-303 Bean Validation
3. Using "@Valid" annotation

   etc

=> JCP (Java Community Process) :-
-> It is simply a process which allows interested parties (organisation or individual) to develop standard technical specifications for java technology
-> It was established in 1998
-> The JCP is a membership-based community and anyone can join by filling out an application form on Oracle JCP website. JCP members have the right to propose JSR's, lead JSR's, vote for JSR's, participate in JCP discussion etc

=> JSR (Java Specification Request) :-
-> It is a formal, open standard document proposal that is submmited to the JCP by an organisation or individual
-> JSR contains proposed changes, additions or improvements to the Java Technology Specifications

-> Some JSR's for Spring are JSR-303 (Bean Validation), JSR-349 (Context and Dependency Injection), JSR-380 (Expression Langauge) etc

```
=> JSR-303 Bean Validation :-
    -> It is the formal specifications that defines the standard annotations for validating Java Beans

    -> Annotations in JSR-303 Bean Validation are :-
        = Basic Annotations : @NotNull, @Null, @NotEmpty, @NotBlank, @Size, @Min, @Max, @Pattern
        = Temporal Annotations : @Past, @Future, @Present
          etc

    -> NOTE :-
        = "Java Bean Validations API" implements the JSR-303 Bean Validation Annotations
        = "Hibernate Validator" extends Java Bean Validations API with additional annotations for hibernate entities
        and ORM scenerios


=================================================================================================================
```

=> @Valid Annotation :-

-> It is a standard annotation which is used to indicate that a method parameter or object field requires validation

-> It is placed directly before the parameter or object to be validated. It can also be applied for method parameter of type Object or any class that supports bean validation

-> Syntax :-

```
public void anyMethod(@Valid SomeClass/Object param)
{
    //implementation
}
```

-> @Valid can be integrated with the JSR-303 Bean Validation API for comprehensive and standardized validation in Spring Applications

```
//@NotEmpty
@NotEmpty(message = "Name should not be empty")
@Size(min = 3, max = 15, message = "Name should be of length 3 to 15 characters")
private String name;

@Email(message =  "Email id is not valid")
private String email;

@Min(value = 3)
@Max(value = 15)
@NotEmpty(message = "Password should not be empty")
@Size(min = 3, max = 15, message = "Password should be of length 3 to 15 characters")
@Pattern(regexp = "^[a-zA-Z0-9@]{3,15}$", message =  "Password is not properly formatted")
private String password;

@NotEmpty(message = "Please select the gender")
private String gender;
```

```
    <error-page>
        <error-code>404</error-code>
        <location>/pageNotFound</location>
    </error-page>
</web-app>
```

```
@GetMapping("/pageNotFound")
public String opepageNotFound()
{
    return "page-not-found";
}
```

```
<error-page>
    <exception-type>java.lang.ArithmeticException</exception-type>
    <location>/errorPage</location>
</error-page>
```

```
<error-page>
    <exception-type>java.lang.ArithmeticException</exception-type>
    <location>/errorPage</location>
</error-page>

<error-page>
    <exception-type>java.lang.NullPointerException</exception-type>
    <location>/errorPage</location>
</error-page>
```

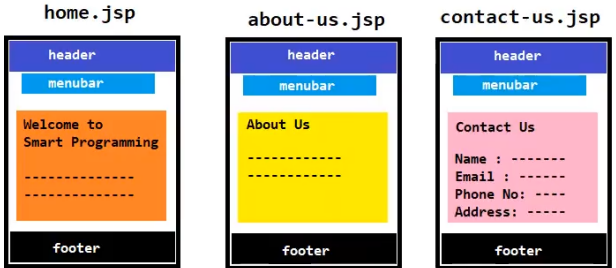
```
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang.ArithmeticException">error-page</prop>
        </props>
    </property>
</bean>
```

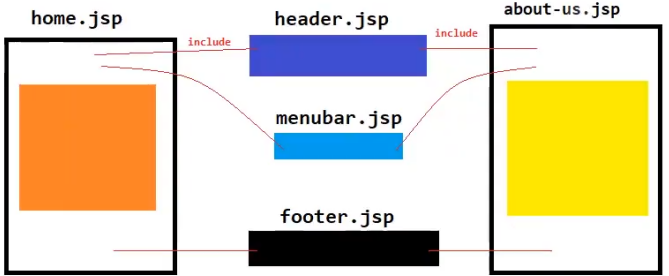
```
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang">error-page</prop>
        </props>
    </property>
</bean>
```

```java
@Controller
public class MyController
{
    @GetMapping("/helloPage")
    public String openHelloPage()
    {
        System.out.println(100/0);

//      String name = null;
//      System.out.println(name.length());

        return "hello";
    }

    @ExceptionHandler(ArithmeticException.class)
    public String openErrorPage()
    {
        return "error-page";
    }
}
```

=> Error & Exception in Spring WEB-MVC :-
    -> Error and Exception are the unwanted events in the program
    -> In Spring WEB-MVC, we can handle the error or exception by 3 ways :-
        1. By web.xml file
        2. By pre-defined class (SimpleMappingExceptionResolver)
        3. By annotations (@ExceptionHandler)

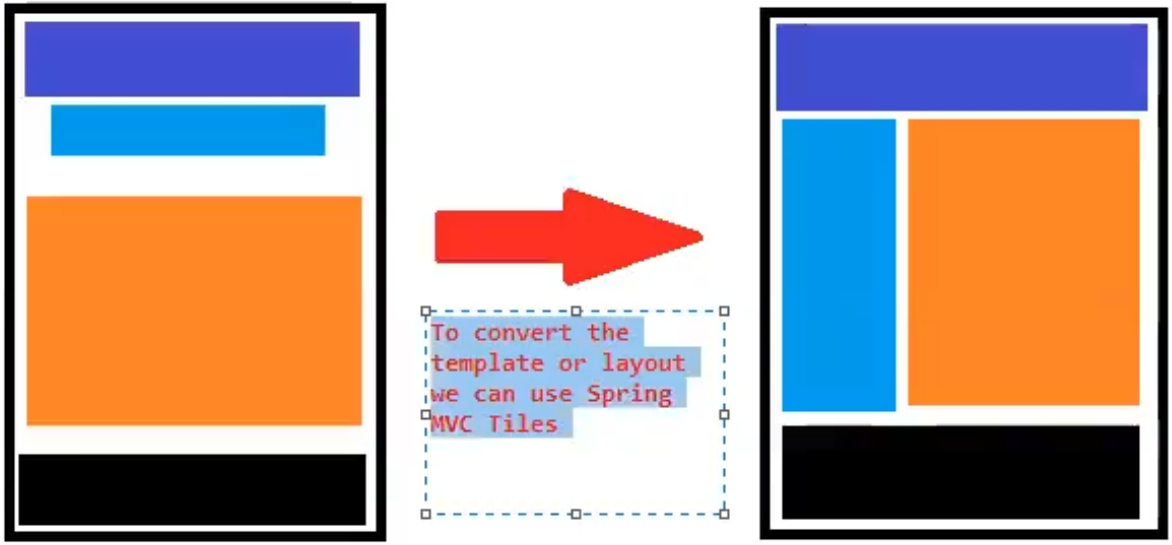
=> @ExceptionHandler :-
    -> It is used to define a method that handles exception thrown by the methods in the controller
    -> It provides a way to centralize the exception handling and present a consistent response to the client in case of errors


home.jsp
header
menubar
Welcome to
Smart Programming
footer
about-us.jsp
header
menubar
About Us
footer
contact-us.jsp
header
menubar
Contact Us
Name : -------
Email : ------
Phone No: ----
Address: -----
footer


home.jsp
include
header.jsp
include
about-us.jsp
menubar.jsp
footer.jsp

header.jsp | menubar.jsp | footer.jsp | home.jsp | about-us.jsp | contact-us.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

    <jsp:include page="header.jsp" />

    <jsp:include page="menubar.jsp" />

    <div style="height: 500px; width: 100%; background-color: orange;">
        <h1> Welcome to Smart Programming </h1>
        <p> This is home page </p>
    </div>

    <jsp:include page="footer.jsp" />

</body>
</html>
```


To convert the
template or layout
we can use Spring
MVC Tiles

```
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-api</artifactId>
    <version>3.0.8</version>
</dependency>

<!-- tiles-jsp dependency -->
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-jsp</artifactId>
    <version>3.0.8</version>
</dependency>
```

```
<!-- tiles-servlet dependency -->
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-servlet</artifactId>
    <version>3.0.8</version>
</dependency>
```

```
<%@taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
```

You can put everything you want in this pages, they are just a test.

Create a definition

By default, the definition file is `/WEB-INF/tiles.xml`. If you're using `CompleteAutoloadTilesListener`, tiles will use any file in the webapp that matches `/WEB-INF/tiles*.xml` or any file in the classpath that matches `/META-INF/tiles*.xml`; if several are found, it will merge them together.

But for now, let's stick to the default and create the `/WEB-INF/tiles.xml` file, with a definition as described in concepts:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
       "http://tiles.apache.org/dtds/tiles-config_3_0.dtd">
<tiles-definitions>
  <definition name="myapp.homepage" template="/layouts/classic.jsp">
    <put-attribute name="title" value="Tiles tutorial homepage" />
    <put-attribute name="header" value="/tiles/banner.jsp" />
    <put-attribute name="menu" value="/tiles/common_menu.jsp" />
    <put-attribute name="body" value="/tiles/home_body.jsp" />
    <put-attribute name="footer" value="/tiles/credits.jsp" />
  </definition>
</tiles-definitions>
```

Render the definition

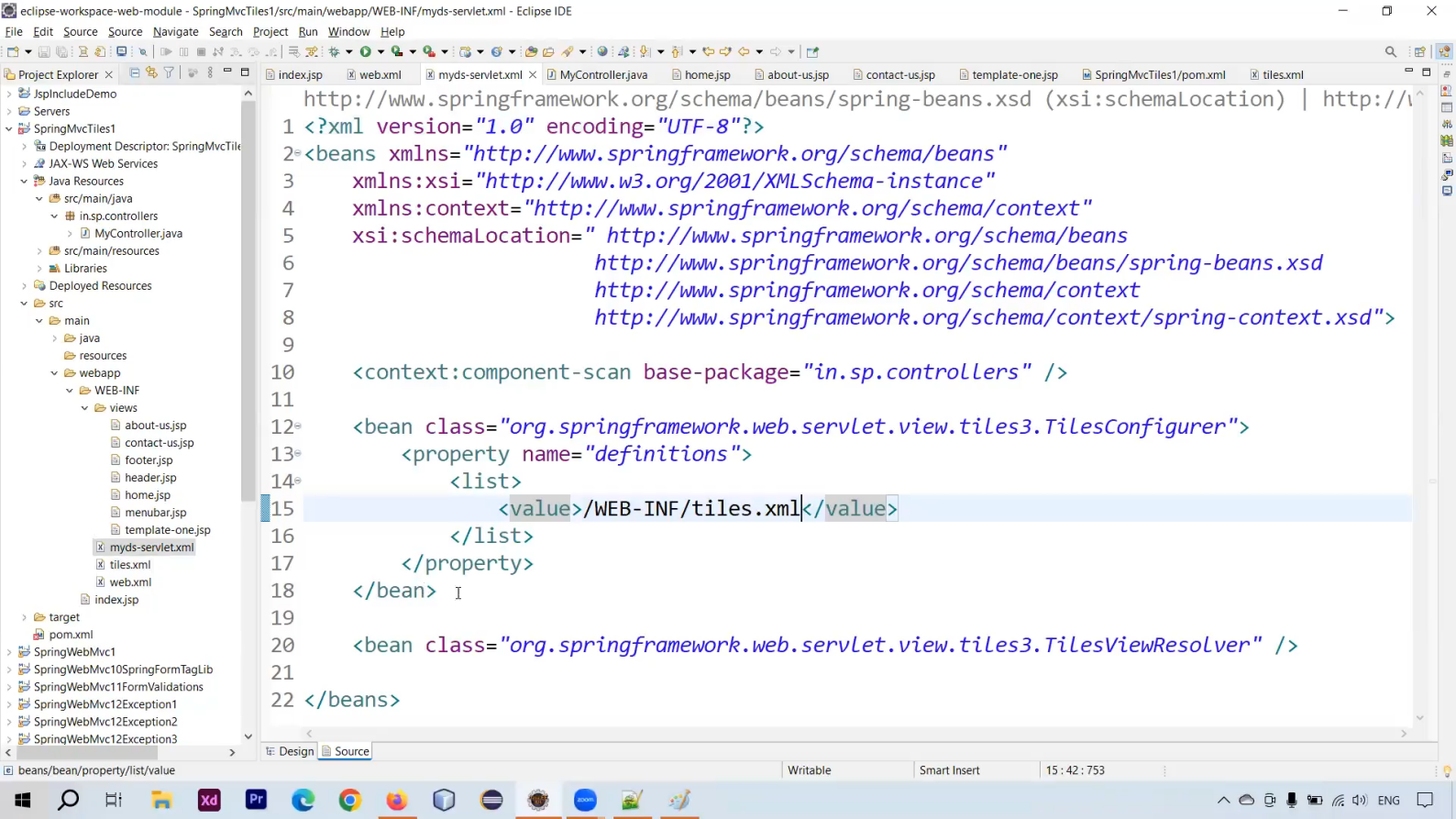
After creating the definition, you can render it:

- by using the `<tiles:insertDefinition />` tag, inserting it in a JSP page:

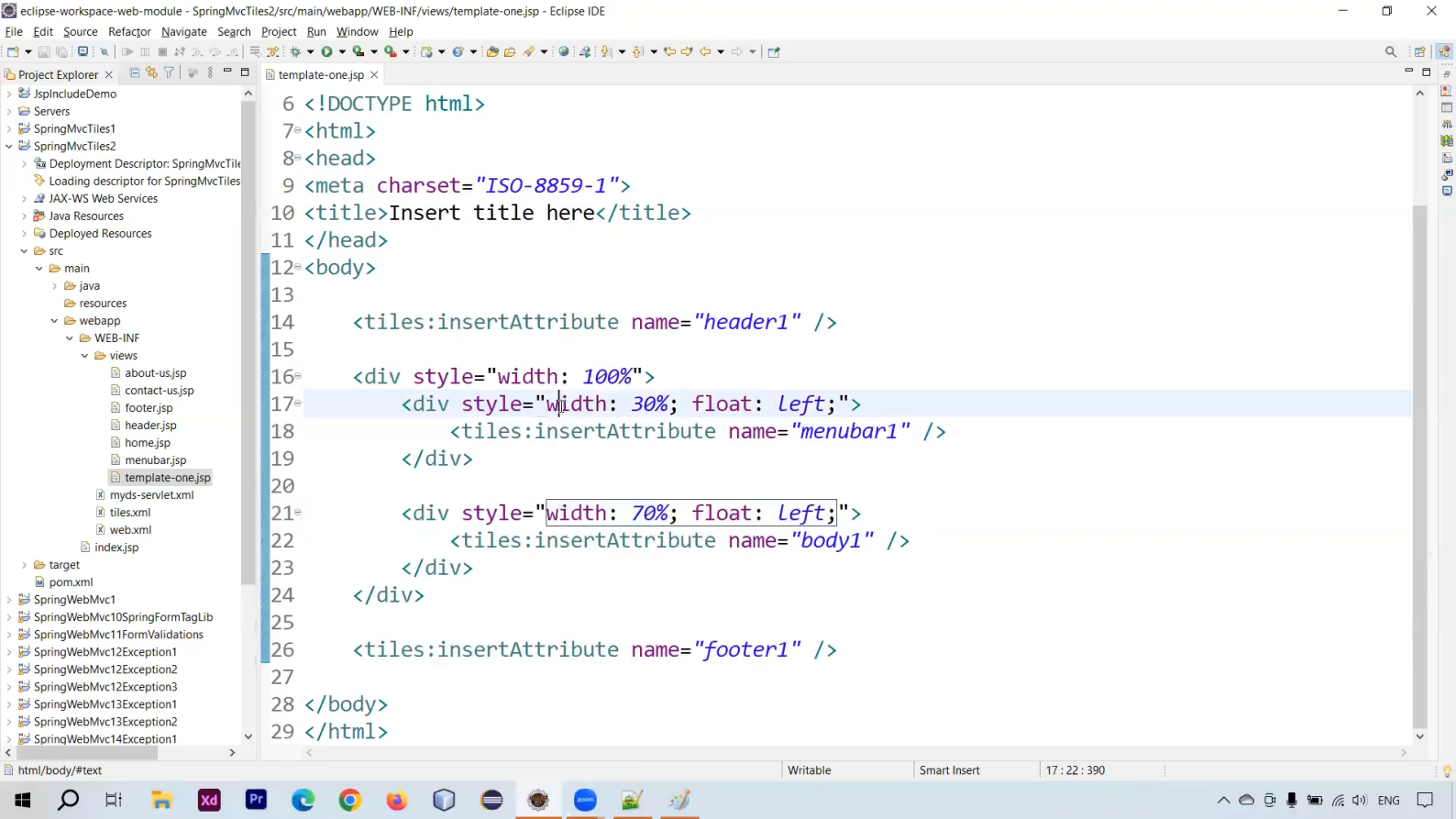
```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<tiles:insertDefinition name="myapp.homepage" />
```

- in other cases, you can render directly in the response, by using the Tiles container:

```
TilesContainer container = TilesAccess.getContainer(
```


eclipse-workspace-web-module - SpringMvcTiles1/src/main/webapp/WEB-INF/myds-servlet.xml - Eclipse IDE
File Edit Source Source Navigate Search Project Run Window Help
Project Explorer
JspIncludeDemo
Servers
SpringMvcTiles1
Deployment Descriptor: SpringMvcTile
JAX-WS Web Services
Java Resources
src/main/java
in.sp.controllers
MyController.java
src/main/resources
Libraries
Deployed Resources
src
main
java
resources
webapp
WEB-INF
views
about-us.jsp
contact-us.jsp
footer.jsp
header.jsp
home.jsp
menubar.jsp
template-one.jsp
myds-servlet.xml
tiles.xml
web.xml
index.jsp
target
pom.xml
SpringWebMvc1
SpringWebMvc10SpringFormTagLib
SpringWebMvc11FormValidations
SpringWebMvc12Exception1
SpringWebMvc12Exception2
SpringWebMvc12Exception3
index.jsp
web.xml
myds-servlet.xml
MyController.java
home.jsp
about-us.jsp
contact-us.jsp
template-one.jsp
SpringMvcTiles1/pom.xml
tiles.xml
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation) | http://
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"
4 xmlns:context="http://www.springframework.org/schema/context"
5 xsi:schemaLocation=" http://www.springframework.org/schema/beans
6 http://www.springframework.org/schema/beans/spring-beans.xsd
7 http://www.springframework.org/schema/context
8 http://www.springframework.org/schema/context/spring-context.xsd">
9
10 <context:component-scan base-package="in.sp.controllers" />
11
12 <bean class="org.springframework.web.servlet.view.tiles3.TilesConfigurer">
13 <property name="definitions">
14 <list>
15 <value>/WEB-INF/tiles.xml</value>
16 </list>
17 </property>
18 </bean>
19
20 <bean class="org.springframework.web.servlet.view.tiles3.TilesViewResolver" />
21
22 </beans>
Design
Source
beans/bean/property/list/value
Writable
Smart Insert
15 : 42 : 753
ENG

```
<!-- jstl dependency -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

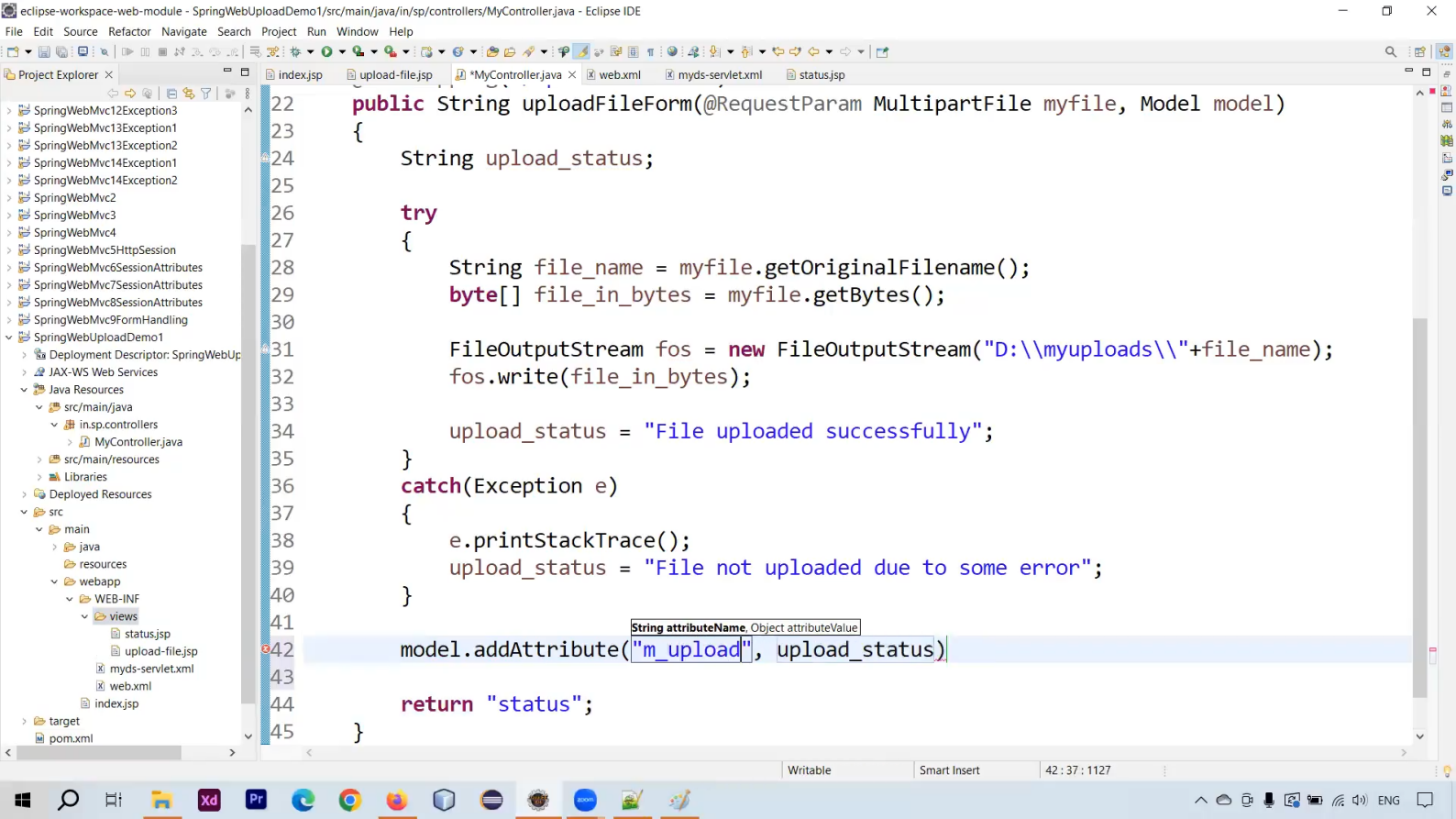

eclipse-workspace-web-module - SpringMvcTiles2/src/main/webapp/WEB-INF/views/template-one.jsp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
JspIncludeDemo
Servers
SpringMvcTiles1
SpringMvcTiles2
Deployment Descriptor: SpringMvcTile
Loading descriptor for SpringMvcTiles
JAX-WS Web Services
Java Resources
Deployed Resources
src
main
java
resources
webapp
WEB-INF
views
about-us.jsp
contact-us.jsp
footer.jsp
header.jsp
home.jsp
menubar.jsp
template-one.jsp
myds-servlet.xml
tiles.xml
web.xml
index.jsp
target
pom.xml
SpringWebMvc1
SpringWebMvc10SpringFormTagLib
SpringWebMvc11FormValidations
SpringWebMvc12Exception1
SpringWebMvc12Exception2
SpringWebMvc12Exception3
SpringWebMvc13Exception1
SpringWebMvc13Exception2
SpringWebMvc14Exception1
template-one.jsp
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="ISO-8859-1">
10 <title>Insert title here</title>
11 </head>
12 <body>
13
14 <tiles:insertAttribute name="header1" />
15
16
17
18 <tiles:insertAttribute name="menubar1" />
19
20
21
22 <tiles:insertAttribute name="body1" />
23
24
25
26 <tiles:insertAttribute name="footer1" />
27
28 </body>
29 </html>
html/body/#text
Writable
Smart Insert
17 : 22 : 390
ENG

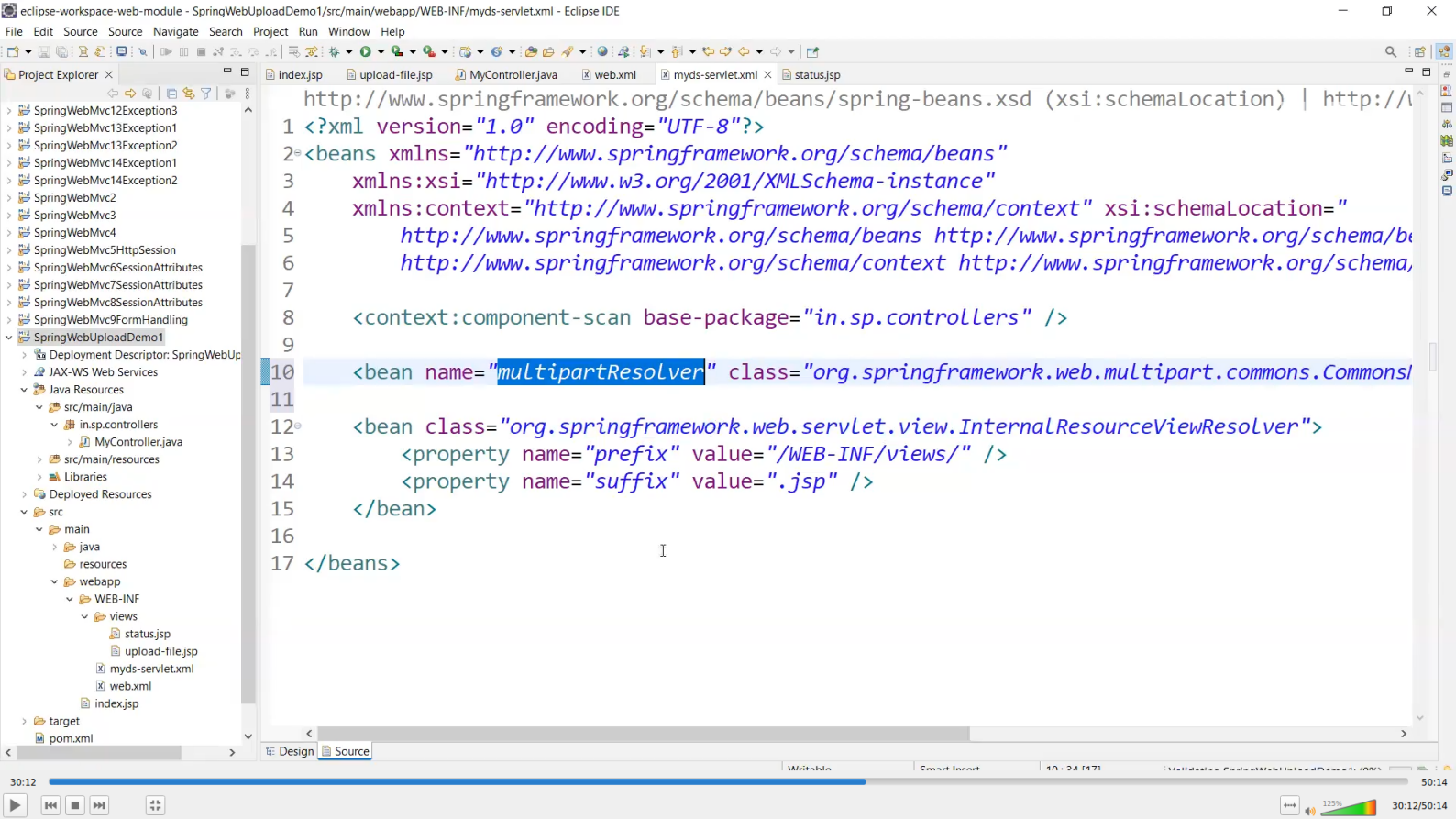
=> JSP Include :-
    -> It is used to include one page content into another page
    -> Syntax :
        <jsp:include page="page-name.jsp" />

=> Spring MVC Tiles :-
    -> It is used to manage the layout/template of Spring WEB-MVC Application
    -> For this Spring provides an integration with "Apache Tiles Framework"

    -> Advantages :-
        1. Reusability :-
            = We can reuse web components i.e. header, menubar, footer etc
        2. Easy to change the layout :-
            = We can easily change the layout from one template to another template
        3. Centralized Control :-
            = We can control the layout of the pages from a single template page

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.5</version>
</dependency>
```

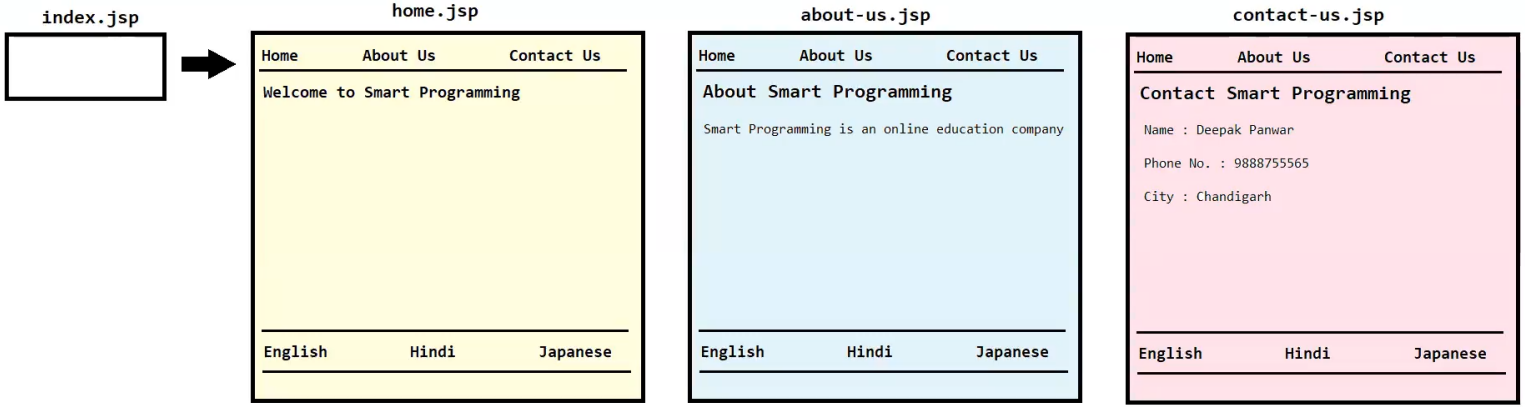

eclipse-workspace-web-module - SpringWebUploadDemo1/src/main/java/in/sp/controllers/MyController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
SpringWebMvc12Exception3
SpringWebMvc13Exception1
SpringWebMvc13Exception2
SpringWebMvc14Exception1
SpringWebMvc14Exception2
SpringWebMvc2
SpringWebMvc3
SpringWebMvc4
SpringWebMvc5HttpSession
SpringWebMvc6SessionAttributes
SpringWebMvc7SessionAttributes
SpringWebMvc8SessionAttributes
SpringWebMvc9FormHandling
SpringWebUploadDemo1
Deployment Descriptor: SpringWebUp
JAX-WS Web Services
Java Resources
src/main/java
in.sp.controllers
MyController.java
src/main/resources
Libraries
Deployed Resources
src
main
java
resources
webapp
WEB-INF
views
status.jsp
upload-file.jsp
myds-servlet.xml
web.xml
index.jsp
target
pom.xml
index.jsp
upload-file.jsp
*MyController.java
web.xml
myds-servlet.xml
status.jsp
22 public String uploadFileForm(@RequestParam MultipartFile myfile, Model model)
23 {
24 String upload_status;
25
26 try
27 {
28 String file_name = myfile.getOriginalFilename();
29 byte[] file_in_bytes = myfile.getBytes();
30
31 FileOutputStream fos = new FileOutputStream("D:\\myuploads\\"+file_name);
32 fos.write(file_in_bytes);
33
34 upload_status = "File uploaded successfully";
35 }
36 catch(Exception e)
37 {
38 e.printStackTrace();
39 upload_status = "File not uploaded due to some error";
40 }
41
String attributeName, Object attributeValue
42 model.addAttribute("m_upload", upload_status)
43
44 return "status";
45 }
Writable
Smart Insert
42 : 37 : 1127
ENG


eclipse-workspace-web-module - SpringWebUploadDemo1/src/main/webapp/WEB-INF/myds-servlet.xml - Eclipse IDE
File Edit Source Source Navigate Search Project Run Window Help
Project Explorer
SpringWebMvc12Exception3
SpringWebMvc13Exception1
SpringWebMvc13Exception2
SpringWebMvc14Exception1
SpringWebMvc14Exception2
SpringWebMvc2
SpringWebMvc3
SpringWebMvc4
SpringWebMvc5HttpSession
SpringWebMvc6SessionAttributes
SpringWebMvc7SessionAttributes
SpringWebMvc8SessionAttributes
SpringWebMvc9FormHandling
SpringWebUploadDemo1
Deployment Descriptor: SpringWebUp
JAX-WS Web Services
Java Resources
src/main/java
in.sp.controllers
MyController.java
src/main/resources
Libraries
Deployed Resources
src
main
java
resources
webapp
WEB-INF
views
status.jsp
upload-file.jsp
myds-servlet.xml
web.xml
index.jsp
target
pom.xml
index.jsp
upload-file.jsp
MyController.java
web.xml
myds-servlet.xml
status.jsp
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation) | http://
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"
4 xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="
5 http://www.springframework.org/schema/beans http://www.springframework.org/schema/be
6 http://www.springframework.org/schema/context http://www.springframework.org/schema/
7
8 <context:component-scan base-package="in.sp.controllers" />
9
10 <bean name="multipartResolver" class="org.springframework.web.multipart.commons.CommonsM
11
12 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
13 <property name="prefix" value="/WEB-INF/views/" />
14 <property name="suffix" value=".jsp" />
15 </bean>
16
17 </beans>
Design
Source
30:12
50:14
30:12/50:14

=> Upload Files (images, pdf, word, xml etc) in Spring WEB Module :-
    -> Steps to upload file :-
        1. Create form to upload file
            = To select file we will use <input type="file">
            = We have to provide enctype="multipart/form-data" in order to upload any file
        2. Provide dependency i.e. commons-fileupload
        3. Get the file name and file (in bytes) and write the file in the folder (in server)
        4. Configure CommonsMultipartResolver class in spring configuration file

==========

=> What is enctype ?
    -> It specifies how form-data should be encoded before sending it to the server
    -> It can be used only with post method, not with get method


index.jsp
home.jsp
Home
About Us
Contact Us
Welcome to Smart Programming
English
Hindi
Japanese
about-us.jsp
Home
About Us
Contact Us
About Smart Programming
Smart Programming is an online education company
English
Hindi
Japanese
contact-us.jsp
Home
About Us
Contact Us
Contact Smart Programming
Name : Deepak Panwar
Phone No. : 9888755565
City : Chandigarh
English
Hindi
Japanese

```
1 <html>
2 <body>
3     <jsp:forward page="home" />
4 </body>
5 </html>
6
```

web.xml | myds-servlet.xml | index.jsp | home.jsp | about-us.jsp | contact-us.jsp | MyController.java

```
<title>Insert title here</title>
</head>
<body>

    <a href="home"> Home </a>
    <a href="aboutUs"> About Us </a>
    <a href="contactUs"> Contact Us </a>

    <hr/>

    <h2> Welcome to Smart Programming </h2>

    <div style="height: 500px"></div>

    <hr/>

    <a href=""> English </a>
    <a href=""> Hindi </a>
    <a href=""> Japanese </a>

    <hr/>

</body>
</html>
```

web.xml | myds-servlet.xml | index.jsp | home.jsp | about-us.jsp | contact-us.jsp | MyController.java

```
10
11      <a href="home"> Home </a>
12          
13      <a href="aboutUs"> About Us </a>
14          
15      <a href="contactUs"> Contact Us </a>
16
17      <hr/>
18
19      <h2> About Smart Programming </h2>
20
21      <p> Smart Programming is an online education company. </p>
22
23      <div style="height: 450px"></div>
24
25      <hr/>
26
27      <a href=""> English </a>
28          
29      <a href=""> Hindi </a>
30          
31      <a href=""> Japanese </a>
32
33      <hr/>
```

```
4  import org.springframework.web.bind.annotation.GetMapping;
5
6  @Controller
7  public class MyController
8  {
9      @GetMapping("/home")
10     public String openHomePage()
11     {
12         return "home";
13     }
14
15     @GetMapping("/aboutUs")
16     public String openAboutUsPage()
17     {
18         return "about-us";
19     }
20
21     @GetMapping("/contactUs")
22     public String openContactUsPage()
23     {
24         return "contact-us";
25     }
26 }
27
```

```
menubar_home = Home
mennubar_aboutus = About Us
menubar_contactus = Contact Us

home_title = Welcome to Smart Programming

aboutus_title = About Smart Programming
aboutus_paragraph = Smart Programming is an online education company.

contactus_title = Contact Smart Programming
contactus_name = Name
contactus_namee = Deepak Panwar
contactus_phoneno = Phone No
contactus_city = City
contactus_cityy = Chandigarh
```

```
13
14      <a href="home"> <spring:message code="menubar_home" /> </a>
15          
16      <a href="aboutUs"> <spring:message code="mennubar_aboutus" /> </a>
17          
18      <a href="contactUs"> <spring:message code="menubar_contactus" /> </a>
19
20      <hr/>
21
22      <h2> <spring:message code="contactus_title" /> </h2>
23
24      <p> <spring:message code="contactus_name" /> : <spring:message code="contactus_namee" /> </p>
25      <p> <spring:message code="contactus_phoneno" /> : 9888755565 </p>
26      <p> <spring:message code="contactus_city" /> : <spring:message code="menubar_contactus" /> </p>
27
28      <div style="height: 400px"></div>
29
30      <hr/>
31
32      <a href=""> English </a>
33          
34      <a href=""> Hindi </a>
35          
36      <a href=""> Japanese </a>
```

```
<a href="?language=en"> English </a>
    
<a href="?language=hi"> Hindi </a>
    
<a href="?language=jp"> Japanese </a>
```

```
<context:component-scan base-package="in.sp.controllers" />

<bean name="messageSource" class="org.springframework.context.support.ResourceBundleMes
    <property name="basename" value="/in/sp/resources/message" />
</bean>

<bean name="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleRe
    <property name="defaultLocale" value="en" />
</bean>

<bean name="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.Locale
    <property name="paramName" value="language" />
</bean>
```

```
<bean name="handlerMapping" class="org.springframework.web.servlet.mvc.method.annotatio
    <property name="interceptors">
        <list>
            <ref bean="localeChangeInterceptor" />
        </list>
    </property>
</bean>
```

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

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

=> Internationalization (I18N) :-
    -> Internationalization means designing and creating the application (web or software) in such a way that makes them easy to adapt for the people of different countries or cultures
    -> It's like making the foundation that can later be customized to fit for the preferences and languages of different countries

==========

=> How our application works :-
    1. The "LocaleChangeInterceptor" intercepts the requests that include a parameter named "language" to change the locale
    2. The "HandlerMapping" determines the appropriate handler method based on the request
    3. The "ViewResolver" will resolves the name to an actual JSP file path based on the configured prefix and suffix
    4. In this process other beans are utilized :-
        -> "MessageSource" helps to resolve internationalized messages
        -> "LocaleResolver" helps to determine the locale to be used

```
=> ResourceBundleMessageSource Bean :-
    -> It is responsible for resolving messages from resource bundle
    -> It's configured to look up messages from "/in/sp/resources/message" location


=> SessionLocaleResolver Bean :-
    -> It is used to resolve the user locale by checking the session
    -> It's set with the default locale value i.e. "en" (English)


=> LocaleChangeInterceptor Bean :-
    -> It intercepts requests to change the locale based on the parameter named "language"


=> RequestMappingHandlerMapping Bean :-
    -> It maps the incoming requests to appropriate handler method
    -> It configures with interceptors (in this case LocaleChangeInterceptor)

=> InternalResourceViewResolver Bean :-
    -> It will resolves the view names to actual JSP page
    -> It configured to look for JSP page in the "/WEB-INF/views" directory
```

=> Spring Web App using Java Configuration :-
    -> Till now we have used xml file for spring configuration, but in modern applications we mostly use java configurations. So we can remove xml based spring configuration file (even web.xml file)

==========

=> @Configuration :-
    -> It is used to declare that the class provides configuration information and bean definations for the application context
    -> When it is applied with class, it indicates that the class contains one or more @Bean methods, which defines the beans that make up the application context
    -> @Configuration classes can be used to replace XML-based configuration

=> AnnotationConfigWebApplicationContext :-
    -> It is a class that is used in web applications where configuration is done through annotated classes, such as marked with @Configuration

=> WebMvcConfigurer :-
    -> It is an interface that provides a way to customize and extend the default configuration of the Spring WEB-MVC framework
    -> It allows us to register additional components, view resolvers, interceptors etc

```
/*
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "in.sp.controllers")
public class WebConfig implements WebMvcConfigurer
{
    @Bean
    public InternalResourceViewResolver viewResolver()
    {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");

        return resolver;
    }
}
```

eclipse-workspace-web-module - SpringWebAppJavaConfig2/src/main/java/in/sp/configurations/WebInitializer.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

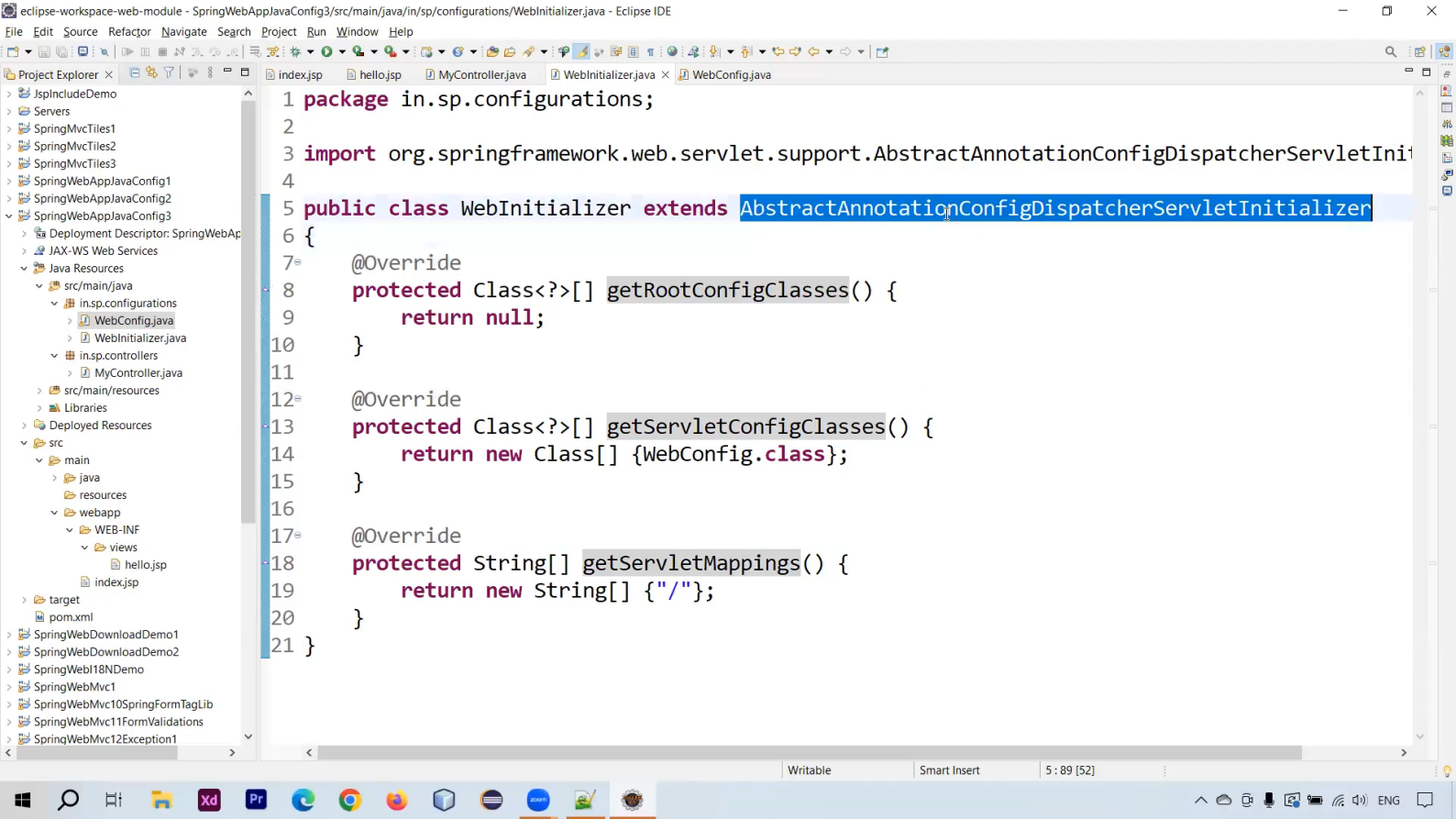
index.jsp | hello.jsp | MyController.java | WebConfig.java | WebInitializer.java

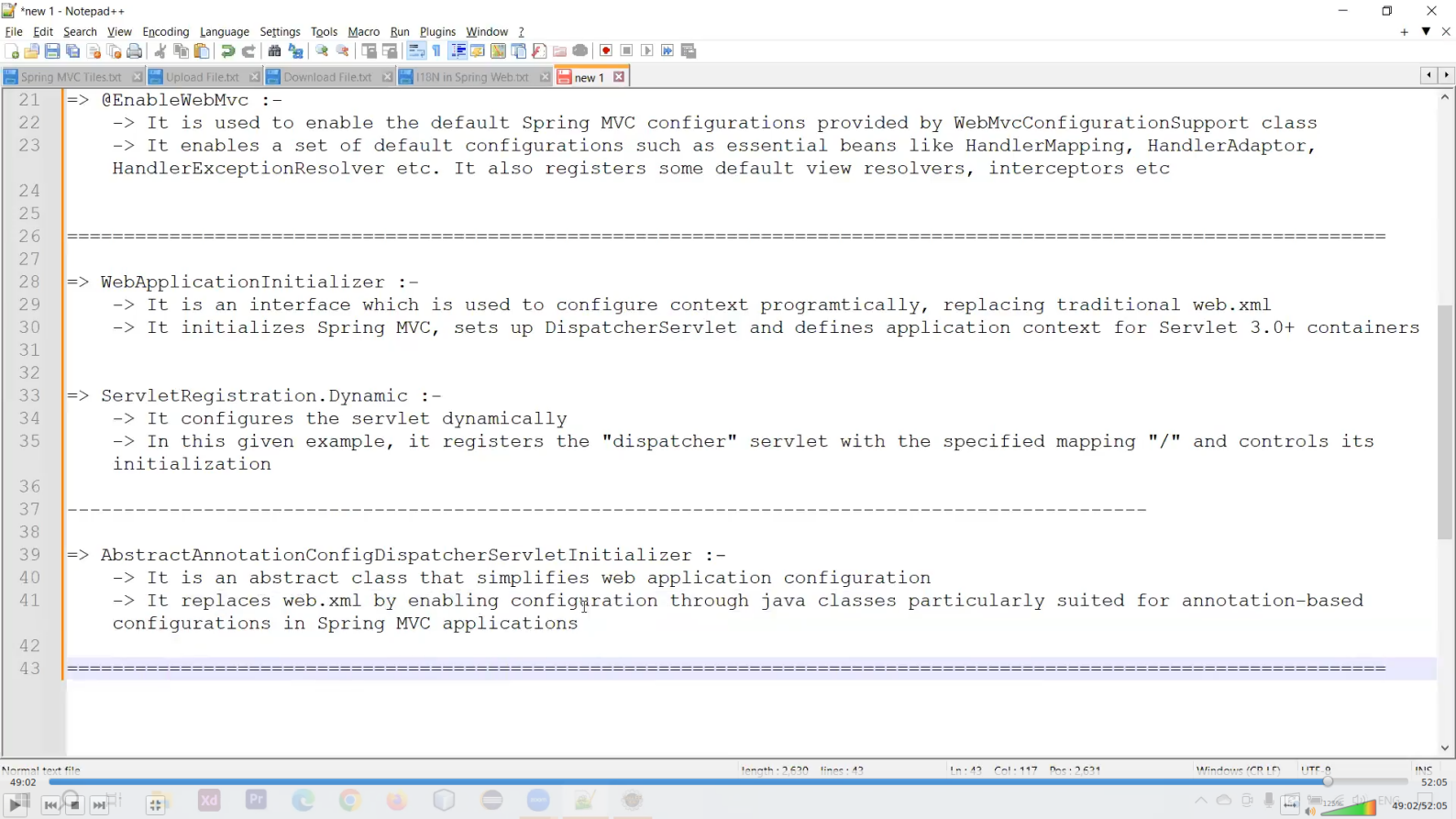
```
1  package in.sp.configurations;
2
3  import javax.servlet.ServletContext;
4  import javax.servlet.ServletException;
5  import javax.servlet.ServletRegistration;
6
7  import org.springframework.web.WebApplicationInitializer;
8  import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
9  import org.springframework.web.servlet.DispatcherServlet;
10
11 public class WebInitializer implements WebApplicationInitializer
12 {
13     @Override
14     public void onStartup(ServletContext servletContext) throws ServletException
15     {
16         AnnotationConfigWebApplicationContext webAppContext = new AnnotationConfigWebApplicationContext();
17         webAppContext.register(WebConfig.class);
18         webAppContext.setServletContext(servletContext);
19
20         ServletRegistration.Dynamic servlet = servletContext.addServlet("dispatcher", new DispatcherServlet(webA
21         servlet.addMapping("/");
22     }
23 }
24
```

Writable | Smart Insert | 20 : 108 [17]

ENG

```
etRegistration.Dynamic servlet = servletContext.addServlet("dispatcher", new DispatcherServlet(webAppContext));
```


eclipse-workspace-web-module - SpringWebAppJavaConfig3/src/main/java/in/sp/configurations/WebInitializer.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
JspIncludeDemo
Servers
SpringMvcTiles1
SpringMvcTiles2
SpringMvcTiles3
SpringWebAppJavaConfig1
SpringWebAppJavaConfig2
SpringWebAppJavaConfig3
Deployment Descriptor: SpringWebAp
JAX-WS Web Services
Java Resources
src/main/java
in.sp.configurations
WebConfig.java
WebInitializer.java
in.sp.controllers
MyController.java
src/main/resources
Libraries
Deployed Resources
src
main
java
resources
webapp
WEB-INF
views
hello.jsp
index.jsp
target
pom.xml
SpringWebDownloadDemo1
SpringWebDownloadDemo2
SpringWebI18NDemo
SpringWebMvc1
SpringWebMvc10SpringFormTagLib
SpringWebMvc11FormValidations
SpringWebMvc12Exception1
index.jsp
hello.jsp
MyController.java
WebInitializer.java
WebConfig.java
1 package in.sp.configurations;
2
3 import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletIni
4
5 public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer
6 {
7 @Override
8 protected Class<?>[] getRootConfigClasses() {
9 return null;
10 }
11
12 @Override
13 protected Class<?>[] getServletConfigClasses() {
14 return new Class[] {WebConfig.class};
15 }
16
17 @Override
18 protected String[] getServletMappings() {
19 return new String[] {"/"};
20 }
21 }
Writable
Smart Insert
5 : 89 [52]
ENG


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Spring MVC Tiles.txt
Upload File.txt
Download File.txt
I18N in Spring Web.txt
new 1
21 => @EnableWebMvc :-
22 -> It is used to enable the default Spring MVC configurations provided by WebMvcConfigurationSupport class
23 -> It enables a set of default configurations such as essential beans like HandlerMapping, HandlerAdaptor, HandlerExceptionResolver etc. It also registers some default view resolvers, interceptors etc
24
25
26 ==========
27
28 => WebApplicationInitializer :-
29 -> It is an interface which is used to configure context programtically, replacing traditional web.xml
30 -> It initializes Spring MVC, sets up DispatcherServlet and defines application context for Servlet 3.0+ containers
31
32
33 => ServletRegistration.Dynamic :-
34 -> It configures the servlet dynamically
35 -> In this given example, it registers the "dispatcher" servlet with the specified mapping "/" and controls its initialization
36
37 ----------
38
39 => AbstractAnnotationConfigDispatcherServletInitializer :-
40 -> It is an abstract class that simplifies web application configuration
41 -> It replaces web.xml by enabling configuration through java classes particularly suited for annotation-based configurations in Spring MVC applications
42
43 ==========
Normal text file
length : 2,630 lines : 43
Ln : 43 Col : 117 Pos : 2,631
Windows (CR LF)
UTF-8
INS
49:02
52:05
49:02/52:05