Deloitte.

Spring MVC

USI CBO CR LAUNCHPAD TRAINING PROGRAM

Spring MVC

Context, Objectives, Agenda

Context

The Spring Framework has grown over years from just being an Inversion of Control container, and currently includes several modules that provide range of services like

- Aspect-oriented programming
- Data access
- Transaction management
- Model–View–Controller
- Authentication and Authorization
- Messaging and Testing

Objectives

To get an understanding of

- What Spring MVC offers
- Develop a Spring MVC Application
- Role of Handler Mappings, View Resolvers
- Spring App with – CRUD Operations

Agenda

Topic	Content
Spring MVC	• Process flow, Components, Architecture • Spring Framework Stereotype Annotations
Spring Handler - Adapters and Mappings	• Handler adapter, Handler mappings • Types of Handler mappings
View Resolvers	• XmlViewResolver, InternalResourceViewResolver
Spring Validation	• Spring MVC Validation • Server side validation, bean validation API
Spring Application	• Use case-CRUD operations

Spring MVC

Spring MVC

Objectives

01 **Introduction**

02 **Process Flow**

03 **MVC Components**

04 **Advantages**

05 **Architecture**

06 **Stereotype Annotations**

07 **Dispatcher Servlet**

08 **Simple Spring MVC Application**

Spring MVC

Introduction

Spring :

- Is an open-source framework created to address the complexity of an enterprise application development.

Spring MVC :

- Is a Java framework to build Web Applications.
- Follows the Model-View-Controller Design Pattern.
- It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

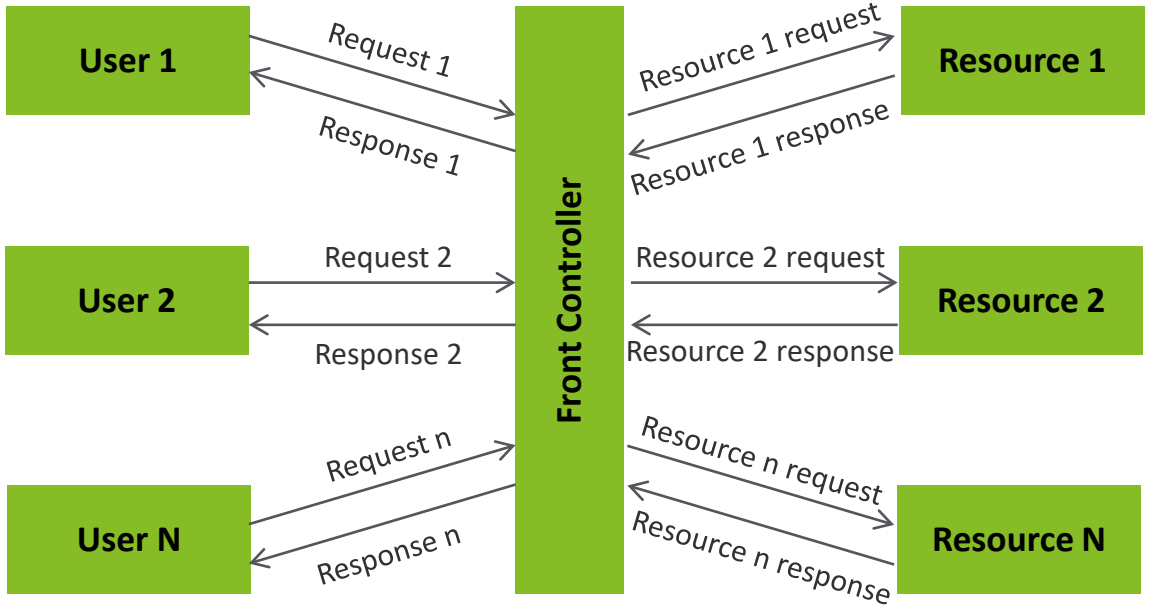
Overview

- Provides MVC architecture and ready components to develop flexible and loosely coupled Web Applications.
- MVC pattern separates aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between.
- It provides an elegant solution by the help of DispatcherServlet.

Spring MVC

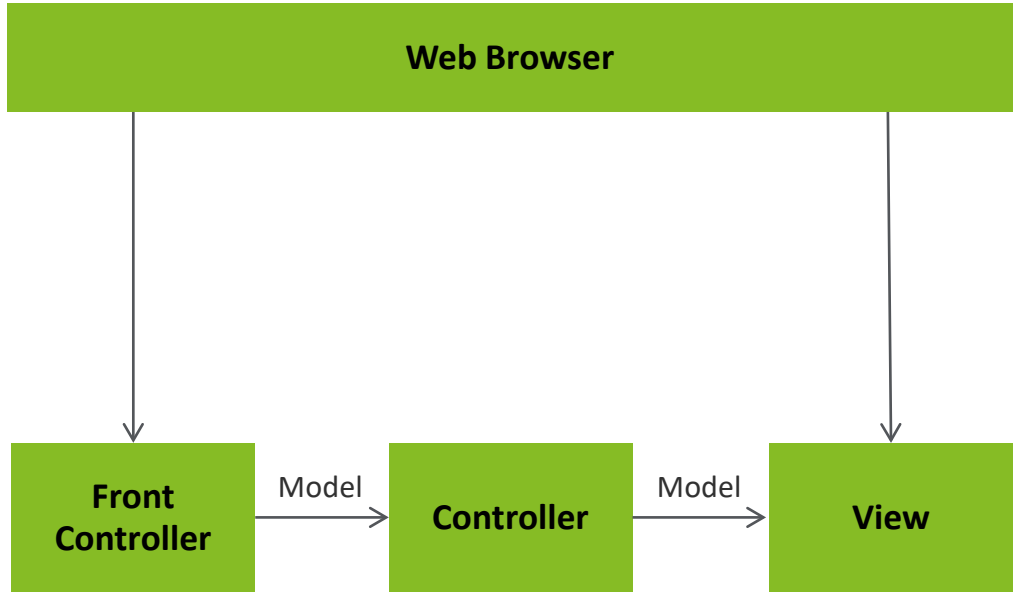
Process Flow

Front Controller Design Pattern



- Enforces a **single point of entry** for all the incoming requests.
- All the requests are handled by a single piece of code
- Further **delegate** the responsibility of processing the request to further application objects.

Spring MVC Design Pattern



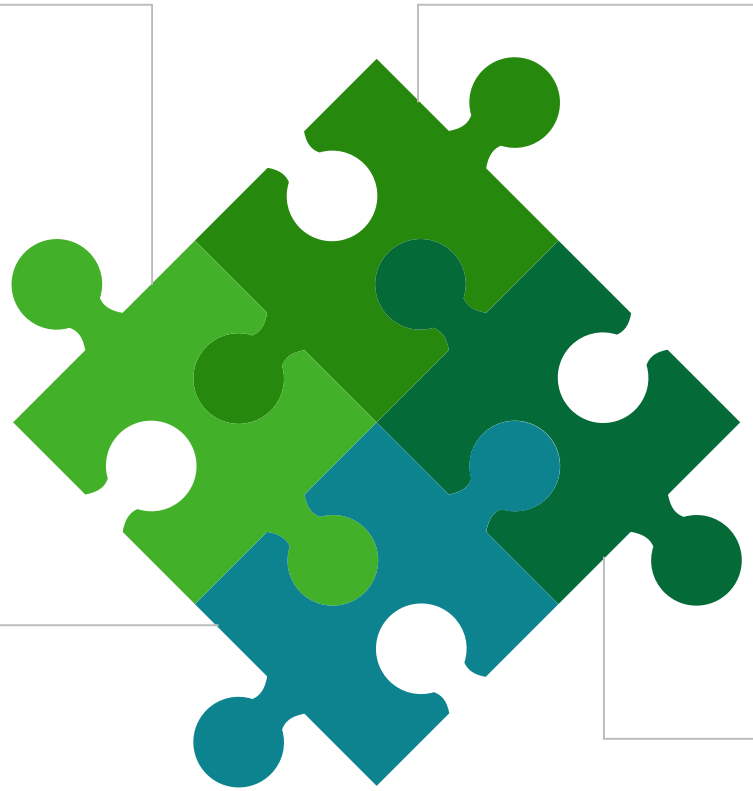
- The flow depicts the **sequence of events** as how a user request is handled by the Spring MVC
- There can be only **one Front Controller**, any number of Models, Controllers, Views

Spring MVC

MVC Components

Front Controller :

- In Spring Web MVC, DispatcherServlet class works as the **front controller.**
- It is responsible to **manage** the flow of the Spring MVC application.



Model :

- A model **contains** the **data** of the application.
- A data can be a single **object** or a collection of objects.

View :

- **Represents** the provided information in a particular format.
- Generally, **JSP+JSTL** is used to create a view page.

Controller :

- Controller contains the **business logic** of an application.
- Here, the **@Controller** annotation is used to mark the class as the controller.

Spring MVC

Advantages

Separate roles –
Separates each role, where Model object, controller, command object, ViewResolver, DispatcherServlet etc. can be fulfilled by a specialized object.

Light-weight –
It uses light-weight servlet container to develop and deploy your application.

Rapid development –
The Spring MVC facilitates fast and parallel development.

Reusable business code –
The Spring MVC facilitates fast and parallel development.

Powerful Configuration –
Provides robust config. for FW & Classes with easy referencing across contexts (Controllers to BO's).

Easy to test –
Facilitates JavaBeans classes that enable you to inject test data using the setter methods.

Flexible Mapping –
It provides the specific annotations that easily redirect the page.

Spring MVC

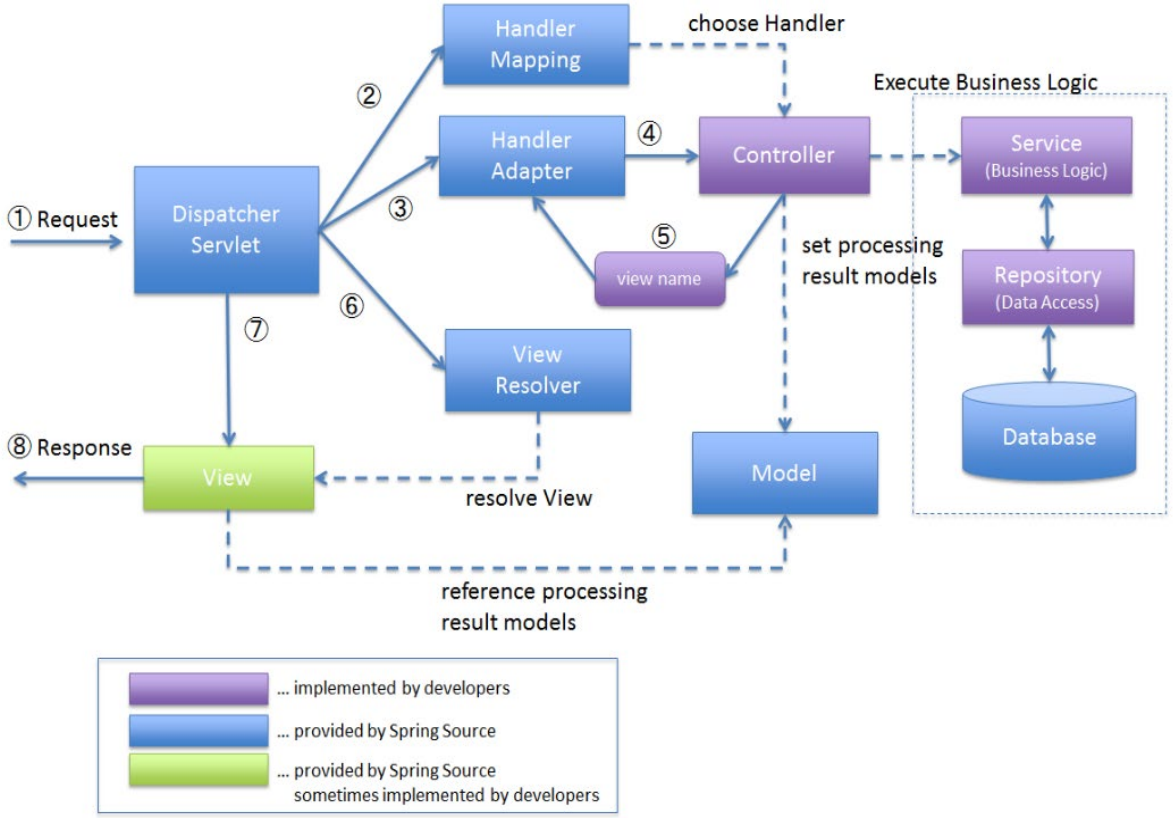
Architecture

Spring's MVC :

It is **request-driven**, designed around a **Central Servlet** that dispatches requests to controllers and offers other functionality that facilitates the development of web applications.

Spring's DispatcherServlet :

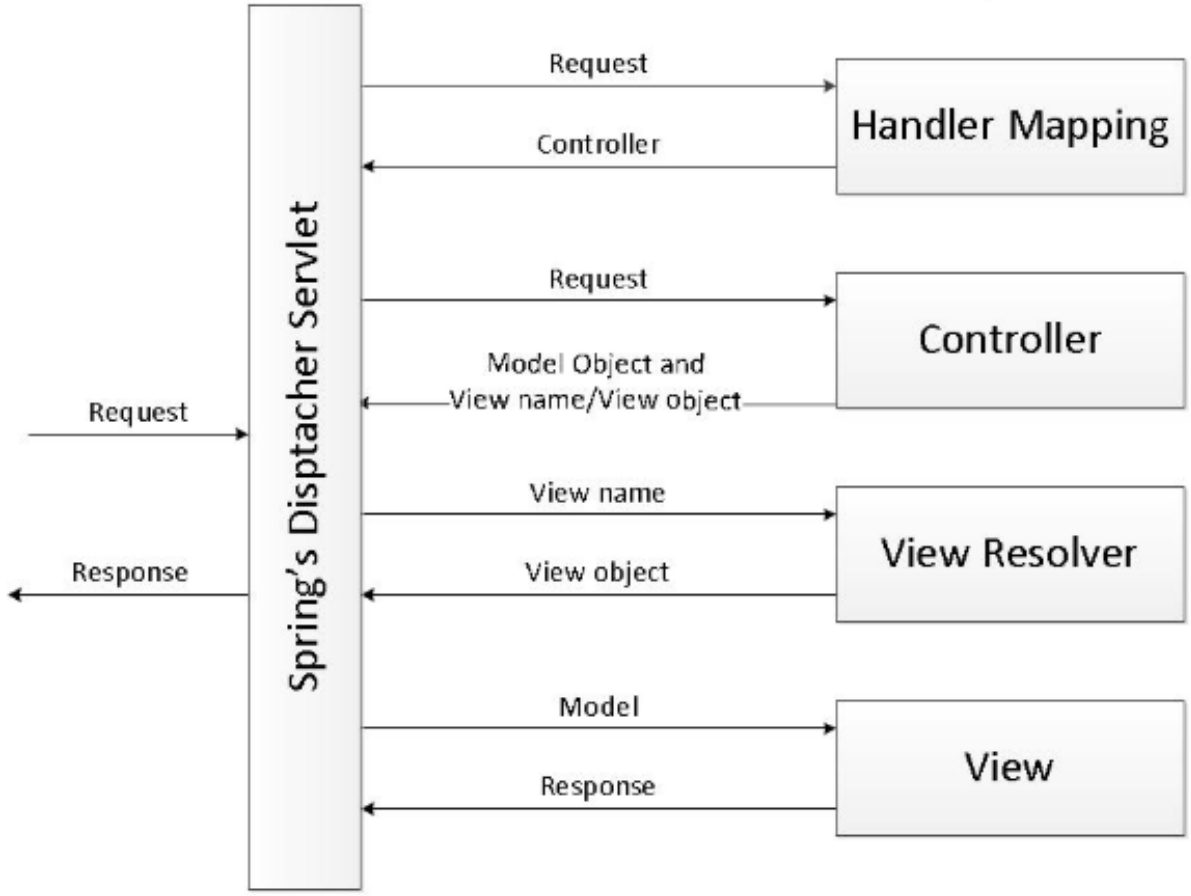
Completely **integrated with the Spring IoC container** and allows you to use every other feature that Spring has.



1. **DispatcherServlet :** receives the request.
2. **DispatcherServlet :** dispatches the task of selecting an appropriate controller to HandlerMapping.
3. **HandlerMapping :** selects the controller which is mapped to the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet
4. **DispatcherServlet :** dispatches the task of executing of business logic of Controller to HandlerAdapter.
5. **HandlerAdapter :** calls the business logic process of Controller.
6. **Controller :** executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter.
7. **DispatcherServlet :** dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolver returns the View mapped to View name.
8. **DispatcherServlet :** dispatches the rendering process to returned View.
9. **View :** renders Model data and returns the response.

Spring MVC

Architecture (Contd..)



- Spring's MVC module is based on **front controller design pattern** followed by **MVC design pattern.**
- All the incoming requests are handled by the single servlet named **DispatcherServlet** which acts as the *front controller* in Spring's MVC module.
- The DispatcherServlet then refers to the **HandlerMapping** (from the XML file) to find a controller object which can handle the request and forwards the request to the controller.
- DispatcherServlet then **dispatches** the request to the **controller** object so that it can actually perform the business logic to fulfil the user request.
- The controller returns an encapsulated object containing the model object and the view object (or a logical name of the view).
  In Spring's MVC, this encapsulated object is represented by class **ModelAndView.**
- In case ModelAndView contains the **logical name** of the view, the DispatcherServlet refers the **ViewResolver** to find the actual View object based on the logical name.
- The DispatcherServlet checks the entry of view resolver in the XML file and **invokes** the **specified view** component.
- DispatcherServlet then passes the **model object to the view** object which is then **rendered** to the end user.

Spring MVC

Spring Framework Stereotype Annotations

- Stereotype annotations are markers for any class that fulfills a role within an application.
- This helps remove, or at least greatly reduce, the Spring XML configuration required for these components.
  - @Component
  - @Controller
  - @Service
  - @Repository

@Controller

- Is used to indicate the class is a **Spring controller.**
- To identify controllers for Spring MVC or Spring WebFlux
- Indicates that a particular class serves the role of a controller. Spring **does not** require you to extend any controller base class or reference the **Servlet API.**
- However, you can **still reference** Servlet-specific features if you need to.
- In Spring MVC you can make controller class very easily by **prefixing** @Controller before any class declaration

@Component

- Is used on classes to indicate a Spring component.
- The **@Component** annotation marks the Java class as a bean or component so that the component-scanning mechanism of Spring can add it into the application context.
- @Component is a **generic stereotype** for any Spring-managed component.
- **@Repository, @Service**, and **@Controller** are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

@Service

- Is used on a class
- @Service marks a Java class that performs some service, such as **executing business logic**, performing calculations, and calling external APIs.
- This annotation is a specialized form of the @Component annotation intended to be **used** in the **service layer**.
- **Annotate** all your service classes with @Service. All your business logic should be in Service classes.

Spring MVC

Spring Framework Stereotype Annotations

(Contd…)

@Repository

- Used on Java classes that directly access the **database**.
- Works as a marker for any class that fulfills the role of repository **or Data Access Object.**
- This annotation has **an automatic translation** feature. For example, when an exception occurs in the @Repository, there is a handler for that exception and there is no need to add a try-catch block.
- A class that serves in the persistence layer of the application as a **Data Access Object (DAO),** otherwise known as a repository in some other technologies.
- Annotate all your DAO classes with @Repository. All your database access logic should be in DAO classes.

@RestController

- The **@RestController** annotation marks the class as a controller where every method returns a domain object instead of a view.
- By annotating a class with this annotation, you **no longer need** to add @ResponseBody to all the RequestMapping methods.
- It means that you **no long use** view-resolvers or send HTML in response. You just send the domain object as an HTTP response in the format that is understood by the consumers, like JSON.
- **@RestController = @Controller + @ResponseBody**
- Spring RestController annotation is used to create **RESTful web services** using Spring MVC. Spring RestController takes care of mapping request data to the defined request handler method.
- Once response body is generated from the handler method, it **converts it to JSON or XML response.**

Spring MVC

Spring Framework Stereotype Annotations

(Contd…)

@Configuration

- Is used on classes that define beans.
- @Configuration is an analog for an XML configuration file – it is configured using Java classes.
- A Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.

```
@Configuartion
public class DataConfig {
    @Bean
    public DataSource source() {
        DataSource source = new OracleDataSource();
        source.setURL();
        source.setUser();
        return source;
    }
    @Bean
    public PlatformTransactionManager manager() {
        PlatformTransactionManager manager = new BasicDataSourceTransactionManager();
        manager.setDataSource(source());
        return manager;
    }
}
```

Interceptor

- Spring Interceptor are used to - **intercept** client requests and process them.
- Sometimes we want to intercept the HTTP Request and do some processing before handing it over to the controller handler methods. That's where Spring MVC Interceptor come handy.
- We can create our own Spring interceptor by either implementing org.springframework.web.servlet.**HandlerInterceptor** interface or by overriding abstract class org.springframework.web.servlet.handler.**HandlerInterceptorAdapter** that provides the base implementation of **HandlerInterceptor interface.**

Spring MVC

Dispatcher Servlet - Configuration

Example 1 : Custom DispatcherServlet - Config

```
<web-app id = "WebApp_ID" version = "2.4"
   xmlns = "http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

   <display-name>Spring MVC Application</display-name>

   <servlet>
      <servlet-name>HelloWeb</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>HelloWeb</servlet-name>
      <url-pattern>*.jsp</url-pattern>
   </servlet-mapping>
</web-app>
```

- **Map requests** that you want the DispatcherServlet to handle, by using a URL mapping in the **web.xml** file.
- **Example :** Declaration & Mapping for **HelloWeb** DispatcherServlet

✓ **web.xml location**	• WebContent/WEB-INF directory
✓ **Upon initialization of HelloWeb**	• framework will load the application context from a file named **[servlet-name]-servlet.xml** located in the WebContent/WEB-INF directory.
✓ **In this case**	• Our file will be **HelloWeb-servlet.xml.**
✓ **<servlet-mapping> tag**	• indicates what URLs will be handled by which DispatcherServlet.
✓ **HTTP requests ending with .jsp**	• will be handled by the **HelloWeb** DispatcherServlet.

- Configuration for **HelloWeb-servlet.xml** file, placed in web application's WebContent/WEB-INF directory –

✓ **[servlet-name]-servlet.xml**	• used to create the beans defined, overriding the definitions of any beans defined with the same name in the global scope.
✓ **<context:component-scan...> tag**	• use **to activate Spring MVC annotation scanning capability** which allows to make use of annotations like @Controller and @RequestMapping etc.
✓ **InternalResourceViewResolver**	• has rules defined to **resolve the view names**. As per the above defined rule, a logical view named hello is delegated to a view implementation located at /WEB-INF/jsp/hello.jsp

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:context = "http://www.springframework.org/schema/context"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package = "com.example" />

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

Spring MVC

Dispatcher Servlet - Configuration (contd..)

Custom DispatcherServlet – Config

If you do not want to go with default filename as [servlet-name]-servlet.xml and default location as WebContent/WEB-INF,

you can customize this file name and location by adding the servlet listener ContextLoaderListener in your web.xml file as follows –

```
<web-app...>

   <!-------- DispatcherServlet definition goes here----->
   ....
   <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/HelloWeb-servlet.xml</param-value>
   </context-param>

   <listener>
      <listener-class>
         org.springframework.web.context.ContextLoaderListener
      </listener-class>
   </listener>

</web-app>
```

Example 2 : Default DispatcherServlet - Config

```
<web-app xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID"
version="2.5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
   <display-name>Library</display-name>
   <servlet>
      <servlet-name>myLibraryAppFrontController</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>myLibraryAppFrontController</servlet-name>
      <url-pattern>*.htm</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>welcome.htm</welcome-file>
   </welcome-file-list>
</web-app>
```

- **DispatcherServlet** acts as the front controller in the Spring's MVC module.
- All the user requests are **handled** by this servlet.
- Since this is like any other servlet, it **must be configured** in the application's web deployment descriptor file i.e. **web.xml.**
- We have named the servlet as "**myLibraryAppFrontController**".
- The **URI pattern** in the servlet mapping section is "*.htm".
- Thus all the requests matching the URI pattern will be handled by **myLibraryAppFrontController.**

Spring MVC

Dispatcher Servlet - Configuration (contd..)

Defining a Controller

✓ **DispatcherServlet**	• delegates web request to Controllers to execute functionality specific to it
✓ **@Controller**	• indicates that a particular class serves the role of a controller.
✓ **@RequestMapping**	• is used to map a URL to either an entire class or a particular handler method.

@RequestMapping *is one of the most common annotation used in Spring Web applications.*

- **Maps** HTTP requests to handler methods of MVC and REST controllers.
- When configuring Spring MVC, specify the mappings between requests and handler methods.
- To **configure** mapping of web requests, you use the @RequestMapping annotation.

✓ **@Controller**	• defines the class as a Spring MVC controller.
✓ **@RequestMapping**	• indicates that all handling methods on this controller are relative to the /hello path.
✓ **@RequestMapping(method = RequestMethod.GET)**	• is used to declare theprintHello() method as the controller's default service method to handle HTTP GET request.

- You can define another method to handle any POST request at the same URL.

- The value attribute indicates the URL to which the handler method is mapped and the method attribute defines the service method to handle HTTP GET request.
- You will define required business logic inside a service method.
- Based on the business logic defined, you will create a model within this method. You can use setter different model attributes and these attributes will be accessed by the view to present the final result. This example creates a model with its attribute "message".
- A defined service method can return a String, which contains the name of the view to be used to render the model. This example returns "hello" as logical view name.

```
@Controller
@RequestMapping("/hello")
public class HelloController {
   @RequestMapping(method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "hello";
   }
}
```

You can write the previous controller in another form where you can add additional attributes in **@RequestMapping** as follows :

```
@Controller
public class HelloController {
   @RequestMapping(value = "/hello", method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "hello";
   }
}
```

Spring MVC

Steps to develop a simple Spring MVC Application

UseCase

Develop a Simple Spring MVC Application that displays output in browser

Approach

- Create project in workspace
- Load the spring jar files manually/add dependencies in case of Maven
- Create the controller class
- Provide the entry of controller in the web.xml file
- Define the bean in the separate XML file
- Display the message in the JSP page
- Start the server and deploy the project

Required JAR files(add manually)/Maven dependencies:

- Spring Core jar files
- Spring Web jar files
- JSP + JSTL jar files (depends on view technology)

Download Link: Download all the jar files for spring including JSP and JSTL

1. Configure pom.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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>SpringMVC</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>SpringMVC Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>

  </dependencies>
  <build>
    <finalName>SpringMVC</finalName>
  </build>
</project>
```

Spring MVC

Steps to develop a simple Spring MVC Application (Contd..)

2. Create Controller Class

To create the controller class, we are using two annotations:

- **@Controller** : Used to mark this class as Controller.
- **@RequestMapping :** Used to map the class with the specified URL name.

```
package com.example;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HelloController {
@RequestMapping("/")
    public String display()
    {
        return "index";
    }
}
```

3. Controller Entry in web.xml

- Specify the servlet class DispatcherServlet that acts as the front controller
- All the incoming request for the html file will be forwarded to the DispatcherServlet.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         id="WebApp_ID" version="3.0">
  <display-name>SpringMVC</display-name>
   <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>
          org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

Spring MVC

Steps to develop a simple Spring MVC Application (Contd..)

4. Define Bean in xml file

- This is the important configuration file where we need to specify the View components.
- The **context:component-scan** element defines where DispatcherServlet searches Controllers.

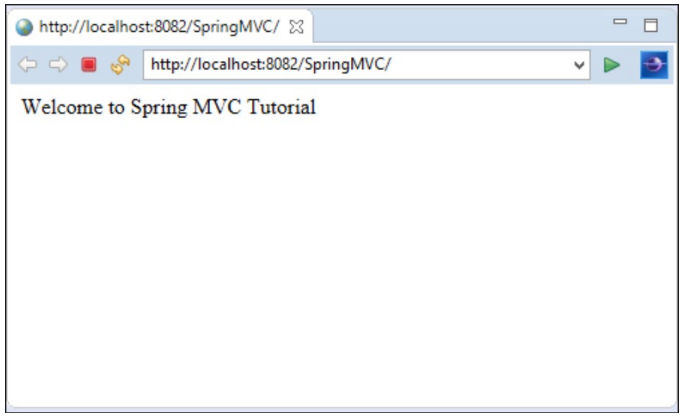
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="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

    <!-- Provide support for component scanning -->
    <context:component-scan base-package="com.example" />

    <!--Provide support for conversion, formatting and validation -->
    <mvc:annotation-driven/>

</beans>
```

5. Create JSP page

This is the simple JSP page, displaying the message returned by the Controller.

```
<html>
    <body>
        <p>
            Welcome to Spring MVC Tutorial
        </p>
    </body>
</html>
```

6. Output



Knowledge Check

1. To configure the mapping of web requests, you use which annotation

- ○ @Mapping
- ○ @RequestMapping
- ○ @ControllerMapping
- ○ None of the above

2. Which Component holds the data of the application

- ○ Component
- ○ RequestDispatcher
- ○ Model
- ○ Controller

3. Spring MVC is used to develop applications that are

- ○ Tightly coupled
- ○ Loosely coupled
- ○ Decoupled
- ○ All of the above

4. Which following is used to execute pre-processing logic of an HTTP request

- ○ HandlerMapping
- ○ ViewResolver
- ○ RequestMapping
- ○ Interceptor

Break – 15 min.

Spring Handler - Adapters and Mappings

Spring Handler- Adapters and Mappings

Objectives

01 **Handler Adapters**

02 **Types of Handler Adapters**

03 **Handler Mappings**

04 **Types of Handler Mappings**

Handler Adapters

Introduction

Handler Adapter:

- Is basically an interface which facilitates the handling of HTTP requests in a very flexible manner in Spring MVC.
- It's used in conjunction with the *HandlerMapping*, which maps a method to a specific URL.
- The *DispatcherServlet* then uses a HandlerAdapter to invoke this method.
- The servlet doesn't invoke the method directly – it basically serves as a bridge between itself and the handler objects, leading to a loosely coupling design.

Let's take a look at various methods available in this HandlerAdapter interface

- handle()
- getLastModified()

```
1  public interface HandlerAdapter {
2      boolean supports(Object handler);
3
4      ModelAndView handle(
5        HttpServletRequest request,
6        HttpServletResponse response,
7        Object handler) throws Exception;
8
9      long getLastModified(HttpServletRequest request, Object handler);
10 }
```

- The **supports** API is used to check if a particular handler instance is supported or not.
- This method should be called first before calling the **handle()** method of this interface, in order to make sure whether the handler instance is supported or not.
- The handle API is used to handle a particular HTTP request. This method is responsible for invoking the handler by passing the **HttpServletRequest** and **HttpServletResponse** object as the parameter.
- The handler then executes the application logic and returns a **ModelAndView** object, which is then processed by the **DispatcherServlet.**

Maven Dependency

- Maven dependency that needs to be added to pom.xml

```
1  <dependency>
2      <groupId>org.springframework</groupId>
3      <artifactId>spring-webmvc</artifactId>
4      <version>4.3.4.RELEASE</version>
5  </dependency>
```

Types of Handler Adapters

Following are the different types of Handler Adapters associated with Spring

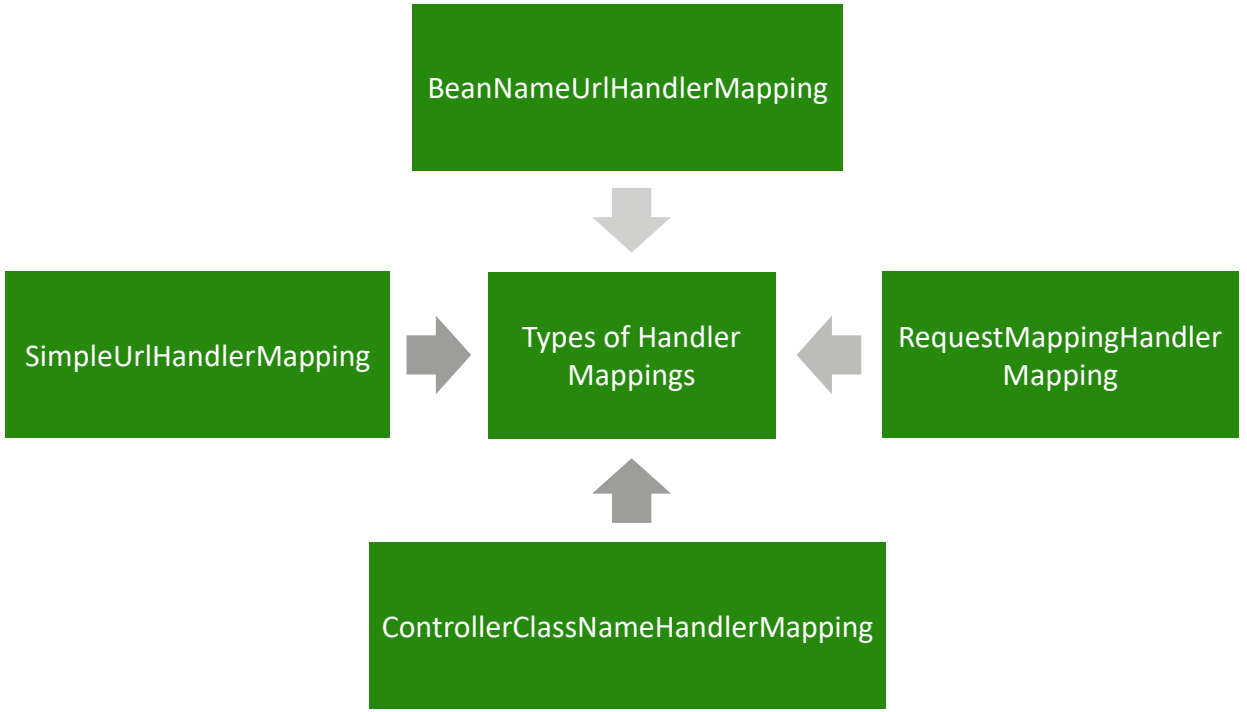
- SimpleControllerHandlerAdapter
- SimpleServletHandlerAdapter
- AnnotationMethodHandlerAdapter
- RequestMappingHandlerAdapter

Handler Mappings

Introduction

Handler Mappings:

- As the name specifies, the handler mapping **maps the request with the corresponding request handler** (in fact handler execution chain).
- When a request comes to Spring's dispatcher servlet, it hands over the request to the handler mapping.
- Handler mapping then inspects the request and identifies the appropriate handler execution chain and delivers it to dispatcher servlet.
- The handler execution chain contains handler that matches the incoming request and optionally contains the list of interceptors that are applied for the request.
- Dispatcher servlet then executes the handlers and any associated handler interceptor.
- All the handler mappings classes implement the interface : **org.springframework.web.servlet.HandlerMapping**



Handler Mappings

Types of Handler Mappings

1. BeanNameUrlHandlerMapping

a) By default DispatcherServlet uses :

- **BeanNameUrlHandlerMapping** and **DefaultAnnotationHandlerMapping**

b) How it works:

- This implementation of handler mapping **matches the URL** of the incoming request with the name of the controller beans.
- **The matching bean** is then used as the controller for the request.
- This is the default handler mapping used by the Spring's MVC module i.e. in case the dispatcher servlet does not find any handler mapping bean defined in Spring's application context then the dispatcher servlet uses **BeanNameUrlHandlerMapping**.

c) Let us assume we have three web pages in our application. The URL of the pages:

- http://servername:portnumber/ApplicationContext/welcome.htm
- http://servername:portnumber/ApplicationContext/listBooks.htm
- http://servername:portnumber/ApplicationContext/displayBookContent.htm

d) Below controllers perform the business logic to fulfil request made to pages above:

- net.example.frameworks.spring.mvc.controller.WelcomeController
- net.example.frameworks.spring.mvc.controller.ListBooksController
- net.example.frameworks.spring.mvc.controller.DisplayBookTOCController

e) Define controllers in Spring's application context such that the name of the controller matches the URL of the request. XML configuration file

```
<bean
    name="/welcome.htm"
    class="net.example.frameworks.spring.mvc.controller.WelcomeController" />
<bean
    name="/listBooks.htm"
    class="net.example.frameworks.spring.mvc.controller.ListBooksController"/>
<bean
    name="/displayBookTOC.htm"
    class="net.example.frameworks.spring.mvc.controller.DisplayBookTOCController"/>
```

Handler Mappings

Types of Handler Mappings (Contd..)

2. SimpleUrlHandlerMapping

- The BeanNameUrlHandlerMapping puts a **restriction** on the name of the controller beans that they should match the URL of the incoming request.

- SimpleUrlHandlerMapping removes this restriction and maps the controller beans to request URL using a property "*mappings*".
  - key of the *<prop>* element is the URL pattern of the incoming request.
  - value of the *<prop>* element is the name of the controller bean which will perform the business logic to fulfil the request.

- SimpleUrlHandlerMapping is one of the simplest and most commonly used handler mapping which allows you **specify URL pattern** and **handler** *explicitly*

- There are two ways of defining **SimpleUrlHandlerMapping**
  - using **<value>** tag and **<props>** tag.

- **SimpleUrlHandlerMapping** has a property called **mappings** we will be passing the URL pattern to it.

```
<bean
 id="myHandlerMapping"
 class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
   <property name="mappings">
      <props>
         <prop key="/welcome.htm">welcomeController</prop>
         <prop key="/listBooks.htm">listBooksController</prop>
         <prop key="/displayBookTOC.htm">displayBookTOCController</prop>
      </props>
   </property>
</bean>
<bean name="welcomeController"
 class="net.codejava.frameorks.spring.mvc.controller.WelcomeController"/>
<bean name="listBooksController"
 class="net.codejava.frameorks.spring.mvc.controller.ListBooksController"/>
<bean name="displayBookTOCController"
 class="net.codejava.frameorks.spring.mvc.controller.DisplayBookTOCController"/>
```

Handler Mappings

Types of Handler Mappings (Contd..)

3. RequestMappingHandlerMapping

- The RequestMappingHandlerMapping is used to **maintain the mapping** of the request URI to the handler.
- Once the handler is obtained, the DispatcherServlet dispatches the request to the appropriate handler adapter, which then invokes the handlerMethod().

1 Let's define a simple controller class:

```
@Controller
public class RequestMappingHandler {

    @RequestMapping("/requestName")
    public ModelAndView getEmployeeName() {
        ModelAndView model = new ModelAndView("Greeting");
        model.addObject("message", "Madhwal");
        return model;
    }
}
```

2 There are 2 different ways of configuring this adapter depending on whether the application uses Java-based configuration or XML based configuration.

Let's look at the first way using Java configuration:

```
@ComponentScan("com.baeldung.spring.controller")
@Configuration
@EnableWebMvc
public class ServletConfig implements WebMvcConfigurer {
    @Bean
    public InternalResourceViewResolver jspViewResolver() {
        InternalResourceViewResolver bean = new InternalResourceViewResolver();
        bean.setPrefix("/WEB-INF/");
        bean.setSuffix(".jsp");
        return bean;
    }
}
```

3 If the application uses XML configuration, then there are two different approaches for configuring this handler adapter in web application context XML. Let us take a look at the first approach defined in the file *spring-servlet_RequestMappingHandlerAdapter.xml*:

```
<beans ...>
    <context:component-scan base-package="com.baeldung.spring.controller" />

    <bean
      class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>

    <bean
      class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>

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

Handler Mappings

Types of Handler Mappings

(Contd..)

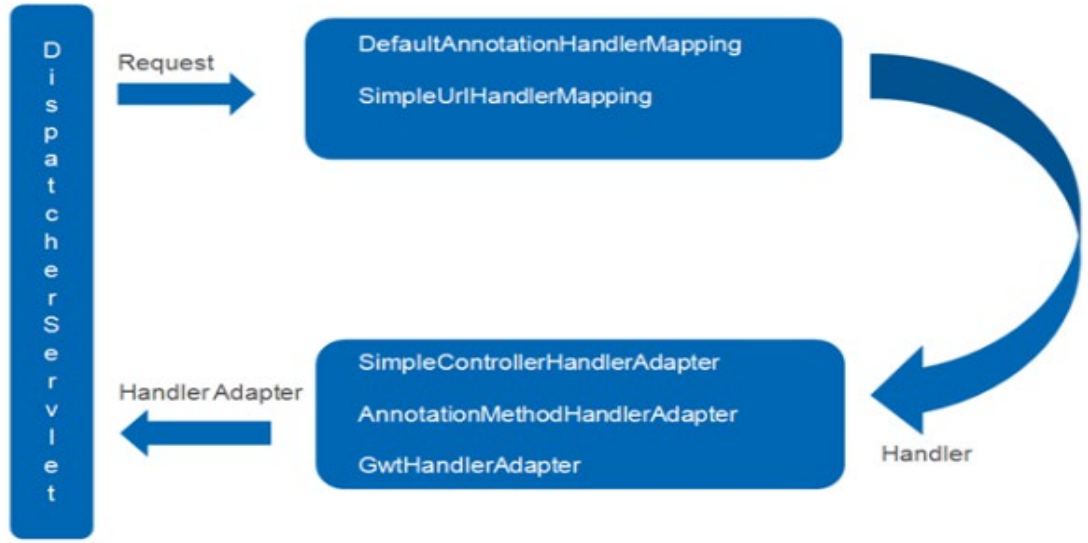
4. ControllerClassNameHandlerMapping

- This type of **HandlerMapping** uses a convention to map the requested URL to the Controller. (but deprecated in Spring 5)
- When using **ControllerClassNameHandlerMapping**, there is no need for **bean name**

Using ControllerClassNameHandlerMapping

```
<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" />
<bean class="com.javainterviewpoint.HelloWorldController"></bean>
<bean class="com.javainterviewpoint.WelcomeController"></bean>
```

HandlerMapping & HandlerAdapter



Knowledge Check

1. Which one dispatches the request to the appropriate handler adapter?

- HandlerMapping
- HandlerAdapter
- DispatcherServlet
- None of the above

2. Which component invokes the handler method?

- RequestDispatcher
- HandlerAdapter
- DispatcherServlet
- Handler Mapping

3. Handler Adapter returns response in the form of which object?

- Model
- ModelAndView
- View
- None

4. DispatcherServlet uses which HandlerMapping by default?

- BeanNameUrlHandlerMapping
- SimpleUrlHandlerMapping
- RequestMappingHandlerMapping
- Interceptor

Lunch Break – 45 min.

View Resolvers

View Resolvers

Objectives

01 **View Resolver Overview**

02 **View Resolver (Types) Classes**

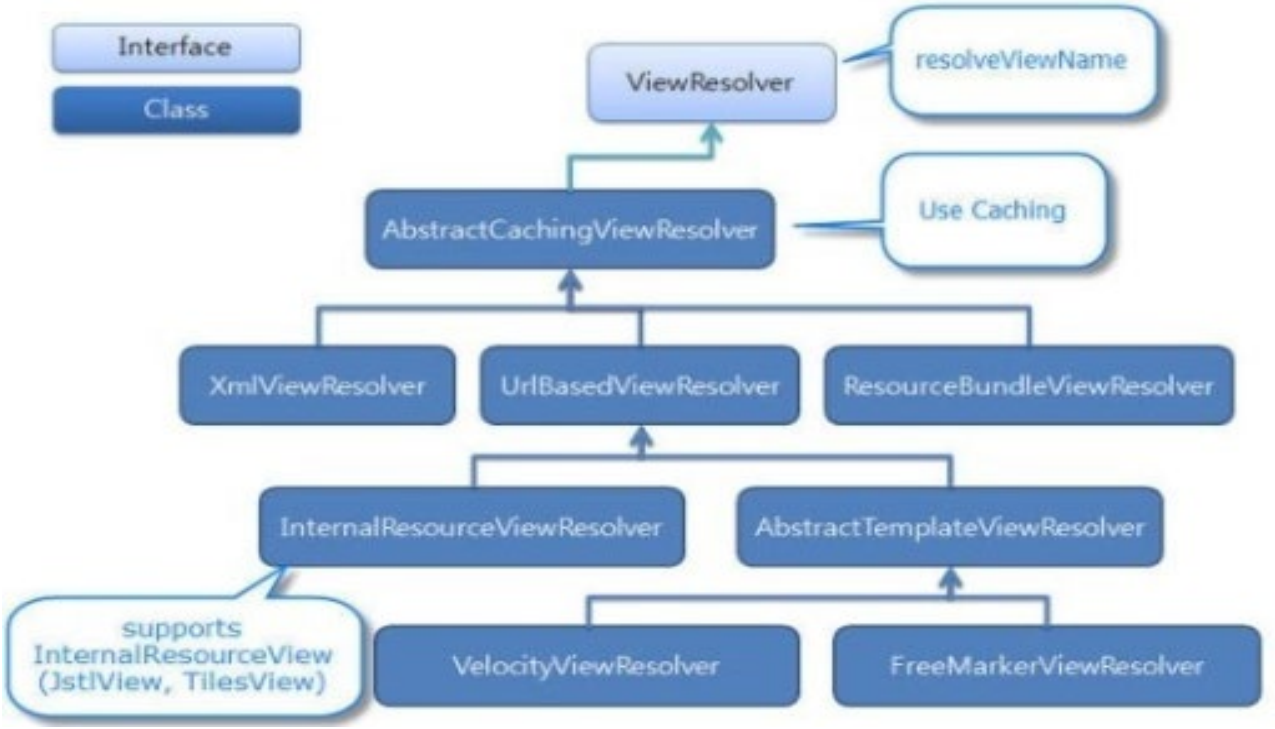
03 **Hands on**

View Resolvers

Overview

1. In Spring MVC, view resolvers enable you to render models in a browser without tying you to a specific view technology like JSP, Velocity, XML…etc.
2. There are two interfaces that are important to the way Spring handles views are **ViewResolver** and **View**.
3. The ViewResolver provides a mapping between view names and actual views.
4. The View interface addresses the preparation of the request and hands the request over to one of the view technologies.

View Resolver Hierarchy



View Resolvers

View Resolver Hierarchy

Classes	Description
✓ AbstractCachingViewResolver	• Abstract view resolver that caches views. Often views need preparation before they can be used; extending this view resolver provides caching
✓ XmlViewResolver	• Implementation of ViewResolver that accepts a configuration file written in XML with the same DTD as Spring's XML bean factories. The default configuration file is /WEB-INF/views.xml.
✓ ResourceBundleViewResolver	• Implementation of ViewResolver that uses bean definitions in a ResourceBundle, specified by the bundle base name. Typically you define the bundle in a properties file, located in the classpath. The default file name is views.properties.
✓ UrlBasedViewResolver	• Simple implementation of the ViewResolver interface that effects the direct resolution of logical view names to URLs, without an explicit mapping definition. This is appropriate if your logical names match the names of your view resources in a straightforward manner, without the need for arbitrary mappings.
✓ InternalResourceViewResolver	• Convenient subclass of UrlBasedViewResolver that supports InternalResourceView (in effect, Servlets and JSPs) and subclasses such as JstlView and TilesView. You can specify the view class for all views generated by this resolver by using setViewClass(..).
✓ VelocityViewResolver	• Convenient subclass of UrlBasedViewResolver that supports VelocityView (in effect, Velocity templates) or FreeMarkerView, respectively, and custom subclasses of them.
✓ ContentNegotiatingViewResolver	• Implementation of the ViewResolver interface that resolves a view based on the request file name or Accept header.

View Resolvers

View Resolver Hierarchy (contd..)

InternalResourceViewResolver

- The **InternalResourceViewResolver** is
  - Implementation of **ViewResolver** interface
  - Extends **UrlBasedViewResolver** class.
- Maps the jsp, servlet and jstl. It uses prefix and suffix to prepare the final view page url, configured in *-servlet.xml file.
- **How to use :**
  Place the below entry inside *-servlet.xml file

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

- Now all the views name return from your controller class will **maps** inside WEB-INF/pages with **suffix as .jsp.**
- It's good practice to put all your view component inside WEB-INF folder for **security purpose.**
- Putting viewing components inside WEB-INF folder will hide them to direct access from the URL, and allow **only controller to access** the viewing components.

XmlViewResolver

- The **XmlViewResolver** is also an
  - Implementation of **ViewResolver** interface
  - Uses bean definitions from the dedicated **xml file.**
- XML file
  - Default Name : **views.xml**
  - Default location : **class path**
- You can map your view name with views inside views.xml file, but XmlViewResolver will not support internationalization.
- **How to use :**
  Create **views.xml** file inside class path with below mapping entries

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="home"
        class="org.springframework.web.servlet.view.JstlView">
        <property name="url" value="/WEB-INF/pages/xml-home.jsp" />
    </bean>
</beans>
```

Now add below entry into your *-servlet.xml file

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.XmlViewResolver">
  <property name="location">
 <value>/WEB-INF/views.xml</value>
  </property>
</bean>
```

View Resolvers

View Resolver Hierarchy (contd..)

ResourceBundleViewResolver

- The **ResourceBundleViewResolver** is an
  - Implementation of **ViewResolver** interface
  - Uses bean definition from **ResourceBundle** specified by the bundle basename.

- Bundle defined in a properties file,
  - Default bundle basename: **views.properties.**
  - Default location is in **class path**

- Will also
  - helps to achieve **Internationalization**
  - to **return excel/pdf** file as a view instead of jsp, jstl

- **How to use :**
  Create a **views.properties** file in your class path with below entries to return views as excel file instead of jsp, jstl.

```
excel.(class)=com.javamakeuse.springmvc.util.ExcelBuilderForResourceBundleViewResolver
```

Now add below entry in your *-servlet.xml file

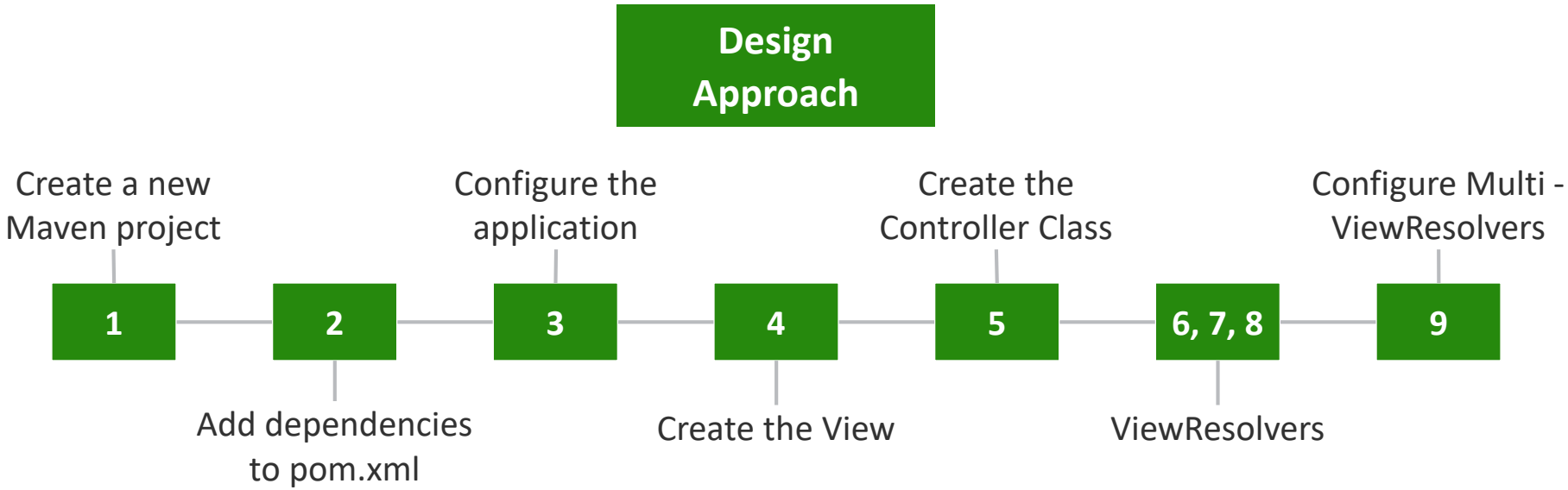
```
<bean id="viewResolver"
  class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
 <property name="basename" value="views" />
</bean>
```

View Resolvers

Demo

USE CASE

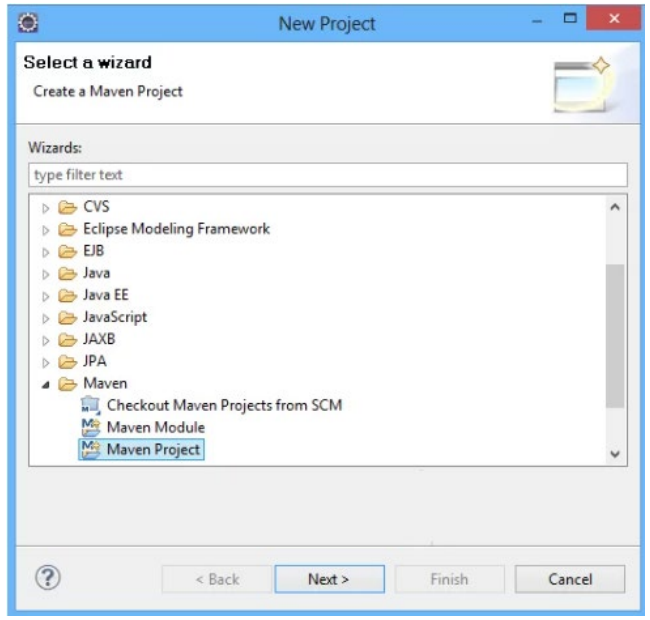
Create a Spring MVC application which shows implementation of ViewResolver



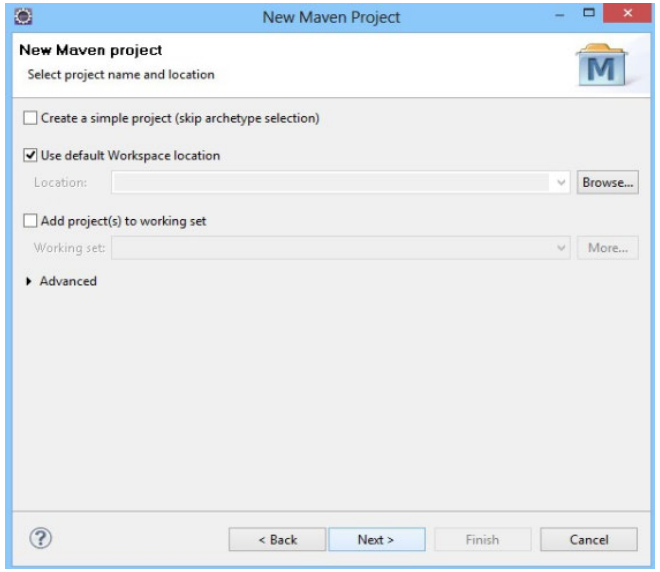
View Resolvers

Demo (contd..)

1. Create a new Maven project



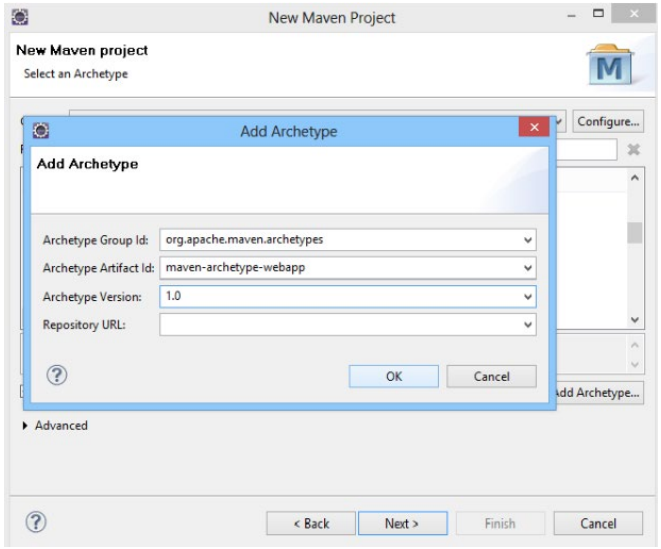
Go to File -> Project -> Maven -> Maven Project.



In the “Select project name and location” page of the wizard, make sure that “Create a simple project (skip archetype selection)” option is **unchecked**, hit “Next” to continue with default values.

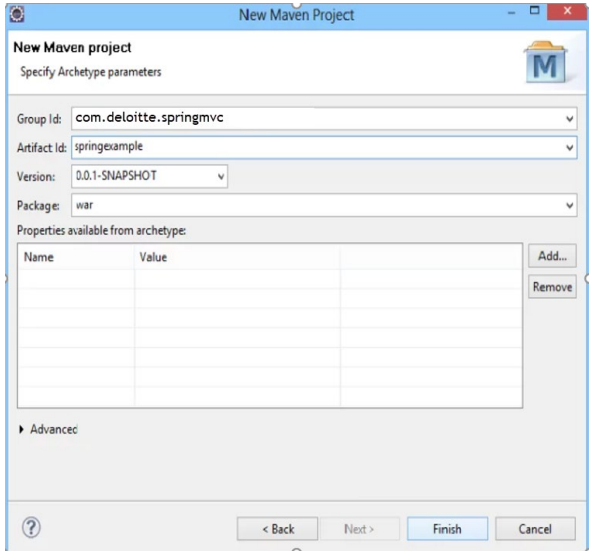
View Resolvers

Demo (contd..)



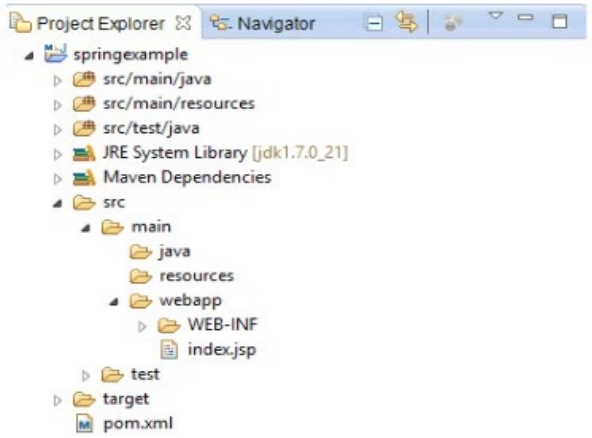
- Define the name and main package of your project.
- Set the “Group Id” & “Artifact Id” values.
- Project package will be **<Group Id value>.<Artifact Id value>** and the project name **<Artifact Id value>**
- “Package” variable set to "war", so that a war file will be created to be deployed onto server (tomcat).
- Hit “Finish” to exit the wizard and to create your project.

- Define the name and main package of your project.
- Set the “Group Id” & “Artifact Id” values.
- Project package will be **<Group Id value>.<Artifact Id value>** and the project name **<Artifact Id value>**
- “Package” variable set to "war", so that a war file will be created to be deployed onto server (tomcat).
- Hit “Finish” to exit the wizard and to create your project.



View Resolvers

Demo (contd..)



✓ **/src/main/java folder**	• Contains source files for the dynamic content of the application
✓ **/src/test/java**	• Contains configurations files
✓ **/src/main/resources**	• Folder contains all source files for unit tests
✓ **/target**	• Folder contains the compiled and packaged deliverables
✓ **/src/main/resources/webapp/WEB-INF**	• Contains the deployment descriptors for the Web application
✓ **pom.xml - project object model (POM)**	• File contains all project related configuration.

2. Add Spring-MVC dependencies

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLScher
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.deloitte.springmvc </groupId>
  <artifactId>springexample</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>springexample Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-webmvc</artifactId>
          <version>${spring.version}</version>
      </dependency>
      <dependency>
          <groupId>javax.servlet</groupId>
          <artifactId>servlet-api</artifactId>
          <version>2.5</version>
      </dependency>
  </dependencies>
  <build>
    <finalName>springexample</finalName>
  </build>
    <properties>
        <spring.version>3.2.3.RELEASE</spring.version>
    </properties>
</project>
```

Dependencies

- Add the dependencies in Maven's pom.xml file, by editing it at the "Pom.xml" page of the POM editor.
- The dependency needed for MVC is the **spring-webmvc** package

View Resolvers

Demo (contd..)

3. Configure the application

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml

    <servlet>
        <servlet-name>mvc-dispatcher</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>mvc-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

- The files that we must configure in the application are the
  - web.xml file
  - mvc-dispatcher-servlet.xml file.
- The web.xml file –
  - **defines** : everything about the application that a server needs to know.
  - **placed in** : the /WEB-INF/ directory of the application.
- The <servlet> element declares the DispatcherServlet.
- When the DispatcherServlet is initialized, the framework will try to load the application context from a file named *[servlet-name]-servlet.xml* located in /WEB-INF/ directory.
- So, we have created the *mvc-dispatcher-servlet.xml* file, that will be explained below. The <servlet-mapping> element of web.xml file specifies what URLs will be handled by the DispatcherServlet.

mvc-dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" http://www.springframework.org/schema/beans
                     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                     http://www.springframework.org/schema/context
                     http://www.springframework.org/schema/context/spring-context-3.0.xsd
                     http://www.springframework.org/schema/mvc
                     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

  <context:component-scan base-package="com.deloitte.springmvc" />

 <bean class="com.deloitte.springmvc.controller.HelloWorldController" />

 </beans>
```

- The mvc-dispatcher-servlet.xml file is also placed in WebContent/WEB-INF directory.
- This is the file where all beans created, such as Controllers, will be placed and defined.
- So, the HelloWorldController, that is the controller of our application is defined here, and will be shown in next steps.
- The <context:component-scan> tag is used so that the container will know where to search for the classes.

View Resolvers

Demo (contd..)

4. Create the View

helloWorld.jsp

```
<html>
<body>
    <h1>Spring 3.2.3 MVC view resolvers example</h1>

    <h3> ${msg}</h3>
</body>
</html>
```

- The view is a simple jsp page, placed in **/WEB-INF/ folder.**
- It shows the value of the attribute that was **set to the Controller.**

5. Create the Controller

HelloWorldController.java

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;

public class HelloWorldController extends AbstractController{

    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest request,
            HttpServletResponse response) throws Exception {

        ModelAndView model = new ModelAndView("helloWorld");
        model.addObject("msg", "hello world!");

        return model;
    }

}
```

- The **HelloWorldController** extends the **AbstractController** provided by Spring, and overrides the **handleRequestInternal(HttpServletRequest request, HttpServletResponse response)** method, where a **org.springframework.web.servlet.ModelAndView** is created by a handler and returned to be resolved by the DispatcherServlet.

View Resolvers

Demo (contd..)

6. InternalResourceViewResolver

mvc-dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" http://www.springframework.org/schema/beans
                     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                     http://www.springframework.org/schema/context
                     http://www.springframework.org/schema/context/spring-context-3.0.xsd
                     http://www.springframework.org/schema/mvc
                     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

   <context:component-scan base-package="com.deloitte.springmvc" />

   <bean class="com.deloitte.springmvc.controller.HelloWorldController" />

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

- The InternalResourceViewResolver maps the jsp and html files in the WebContent/WEB-INF/ folder.
- It allows us to set properties such as prefix or suffix to the view name to generate the final view page URL.
- It is configured as shown below in mvc-dispatcher-servlet.xml.
- When the Controller returns the "helloworld" view, the InternalResourceViewResolver will create the url of the view making use of the prefix and suffix properties that are set to it, and will map the "helloworld" view name to the correct "helloworld" view.

7. XmlViewResolver

views.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="helloWorld"
        class="org.springframework.web.servlet.view.JstlView">
        <property name="url" value="/WEB-INF/helloWorld.jsp" />
    </bean>

</beans>
```

- XmlViewResolver is an implementation of ViewResolver that accepts a configuration file written in XML, where the view implementation and the url of the jsp file are set. Above is the configuration file, views.xml.
- The resolver is defined in mvc-dispatcher-servlet.xml. It provides a property to configure, which is the location property, and there the path of the configuration file is set.

View Resolvers

Demo (contd..)

7. XmlViewResolver (Contd..)

mvc-dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" http://www.springframework.org/schema/beans
                     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                     http://www.springframework.org/schema/context
                     http://www.springframework.org/schema/context/spring-context-3.0.xsd
                     http://www.springframework.org/schema/mvc
                     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

   <context:component-scan base-package="com.deloitte.springmvc" />

   <bean class="com.deloitte.springmvc.controller.HelloWorldController" />

   <bean class="org.springframework.web.servlet.view.XmlViewResolver">
       <property name="location">
           <value>/WEB-INF/views.xml</value>
       </property>
       <property name="order" value="1" />
   </bean>
</beans>
```

- Now, when the Controller returns the "helloworld" view, the XmlViewResolver will make use of the views.xml file to get the view class and the url of the view that will be mapped to the name "helloworld".

8. ResourceBundleViewResolver

mvc-dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" http://www.springframework.org/schema/beans
                     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                     http://www.springframework.org/schema/context
                     http://www.springframework.org/schema/context/spring-context-3.0.xsd
                     http://www.springframework.org/schema/mvc
                     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
   <context:component-scan base-package="com.deloitte.springmvc" />

   <bean class="com.deloitte.sprinamvc.controller.HelloWorldController" />

    <bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
        <property name="basename" value="views" />
        <property name="order" value="0" />
    </bean>
</beans>
```

- The ResourceBundleViewResolver uses bean definitions in a ResourceBundle, that is specified by the bundle basename.
- The bundle is typically defined in a properties file, located in the classpath.
- Below is the views.properties file:

  views.properties

  1. helloworld.(class)=org.springframework.web.servlet.view.JstlView
  2. helloworld.url=/WEB-INF/helloworld.jsp

- The ResourceBundleViewResolver is defined in mvc-dispatcher-servlet.xml, and in its definition the basename property is set to view.properties file.
- So, in this case, when the Controller returns the "helloworld" view, the ResourceBundleViewResolver will make use of the views.properties file to get the view class and the url of the view that will be mapped to the name "helloworld".

View Resolvers

Demo (contd..)

9. Multiple View Resolvers together

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" http://www.springframework.org/schema/beans
                     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                     http://www.springframework.org/schema/context
                     http://www.springframework.org/schema/context/spring-context-3.0.xsd
                     http://www.springframework.org/schema/mvc
                     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

   <context:component-scan base-package="com.deloitte.springmvc" />

   <bean class="com.deloitte.springmvc.controller.HelloWorldController" />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
        <property name="order" value="2" />
    </bean>

    <bean class="org.springframework.web.servlet.view.XmlViewResolver">
        <property name="location">
            <value>/WEB-INF/views.xml</value>
        </property>
        <property name="order" value="1" />
    </bean>

    <bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
        <property name="basename" value="views" />
        <property name="order" value="0" />
    </bean>
</beans>
```

- In order to set multiple Resolvers together in the same configuration file, you can set the order property in all definitions, so that the order that they are used will be defined, as shown beside.
- Note that the InternalResourceViewResolver has the lowest priority, because it can map any request to the correct view, so if set before other resolvers the other resolvers will never be used.

Output

You can run your application, using a tomcat server and the result will be as shown in the screenshot beside

Knowledge Check

1. Which of the following ViewResolver extends **UrlBasedViewResolver?**

- InternalResourceViewResolver
- XmlViewResolver
- ResourceBundleViewResolver
- VelocityViewResolver

2.What is the default bundle basename in ResourcebunderViewResolver?

- views
- views.properties
- views.xml
- None of the above

3. /target folder contains

- Configuration files coupled
- Dynamic Content
- Compiled and Packaged deliverables
- All of the above

4. What is the dependency needed for Spring MVC?

- spring-mvc
- spring-webmvc
- spring-coremvc
- spring-viewmvc

Spring Validation

Spring Validation

Objectives

01 **Server Side Validation**

02 **Bean Validation API**

03 **Spring MVC Validation**

04 **Use Case**

Spring Validation

Overview

Server Side Validation

- When we accept user inputs in any web application, it become necessary to validate them. We can validate the user input at client side using JavaScript but it's also necessary to validate them at server side to make sure we are processing valid data incase user has javascript disabled.
- Spring MVC Framework supports JSR-303 specs by default and all we need is to add JSR-303 and it's implementation dependencies in Spring MVC application. Spring also provides @Validator annotation and BindingResult class through which we can get the errors raised by Validator implementation in the controller request handler method.
- We can create our custom validator implementations by two ways – first one is to create an annotation that confirms to the JSR-303 specs and implement it's Validator class. Second approach is to implement the org.springframework.validation.Validator interface and add set it as validator in the Controller class using @InitBinder annotation.

Bean Validation API

- The Bean Validation API is a Java specification which is used to apply constraints on object model via annotations.
- Here, we can validate a length, number, regular expression, etc. Apart from that, we can also provide custom validations.
- As Bean Validation API is just a specification, it requires an implementation.
- So, for that, it uses Hibernate Validator. The Hibernate Validator is a fully compliant JSR-303/309 implementation that allows to express and validate application constraints.

Spring Validation

Spring MVC Validation

Spring MVC Validation

- The Spring MVC Validation is used to restrict the input provided by the user.

- To validate the user's input, the Spring 4 or higher version supports and use Bean Validation API. It can validate both server-side as well as client-side applications.

Annotation	Description
✓ @NotNull	• It determines that the value can't be null.
✓ @Min	• It determines that the number must be equal or greater than the specified value.
✓ @Max	• It determines that the number must be equal or less than the specified value.
✓ @Size	• It determines that the size must be equal to the specified value.
✓ @Pattern	• It determines that the sequence follows the specified regular expression.

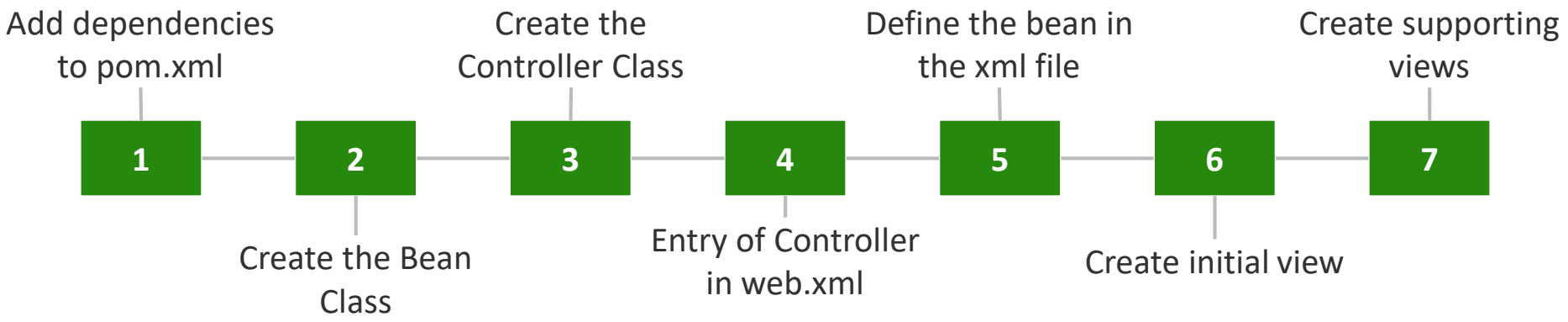
Spring MVC Validation

Demo

USE CASE

Create a Spring MVC application with Login form and implement **Spring MVC validation** for the form fields

Design Approach



Spring MVC Validation

Demo

(contd..)

1. Add dependencies to pom.xml

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
 <!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
```

Highlighted the dependencies to be included - in below code snippet

2. Create the Bean Class

```
package com.example;
import javax.validation.constraints.Size;

public class Employee {

    private String name;
    @Size(min=1,message="required")
    private String pass;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
}
```

An Employee class with two properties for capturing Username and Password

Spring MVC Validation

Demo (contd..)

3. Create the Controller Class

```
package com.javatpoint;

import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class EmployeeController {

    @RequestMapping("/hello")
    public String display(Model m)
    {
        m.addAttribute("emp", new Employee());
        return "viewpage";
    }
    @RequestMapping("/helloagain")
    public String submitForm( @Valid @ModelAttribute("emp") Employee e, BindingResult br)
    {
        if(br.hasErrors())
        {
            return "viewpage";
        }
        else
        {
        return "final";
        }
    }
}
```

In controller class:

The **@Valid** annotation applies validation rules on the provided object.

- The BindingResult interface contains the result of validation.

4. Entry of Controller in web.xml

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         id="WebApp_ID" version="3.0">
  <display-name>SpringMVC</display-name>
   <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

- When the DispatcherServlet is initialized, the framework will try to load the application context from a file named [servlet-name]-servlet.xml

- The <servlet-mapping> element of web.xml file specifies what URLs will be handled by the DispatcherServlet.

Spring MVC Validation

Demo

(Contd..)

5. Define the bean in the xml file

spring-servlet.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"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- Provide support for component scanning -->
    <context:component-scan base-package="com.javatpoint" />
    <!--Provide support for conversion, formatting and validation -->
    <mvc:annotation-driven/>
    <!-- Define Spring MVC view resolver -->
     <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
     </bean>
</beans>
```

6. Create initial view

index.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<body>
<a href="hello">Click here...</a>
</body>
</html>
```

7. Create supporting views

viewpage.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
<style>
.error{color:red}
</style>
</head>
<body>
<form:form action="helloagain" modelAttribute="emp">
Username: <form:input path="name"/> <br><br>
Password(*): <form:password path="pass"/>
<form:errors path="pass" cssClass="error"/><br><br>
<input type="submit" value="submit">
</form:form>
</body>
</html>
```

final.jsp

```
<html>
<body>
Username: ${emp.name} <br><br>
Password: ${emp.pass}
</body>
</html>
```

Knowledge Check

1. Spring MVC Validation is used for

- ○ securing the application data
- ○ restrict data input by user
- ○ storing data in application
- ○ None

2. Bean Validation API is just

- ○ an implementation
- ○ a specification
- ○ a documentation
- ○ All of the above

3. Which of the following is used to apply constraints on object models

- ○ Spring Validation API
- ○ Bean Validation API
- ○ View Validation API
- ○ None

4. Client validation by

- ○ Javascript
- ○ Bean Validation API
- ○ Spring MVC Validation
- ○ Client Validation API

Break – 15 min.

SPRING MVC
CRUD APPLICATION
HANDS ON

Any Questions?

Thank You

Deloitte.