UNIT 5 INTRODUCTION TO J2EE FRAMEWORKS

Structure

5.0 INTRODUCTION

In Units 1 to 4, we have already discussed J2EE and design patterns like MVC. In this unit, we shall cover an introduction to various open-source J2EE frameworks.

Spring MVC framework is used for creating Web applications that utilize MVC architecture. Apache Struts 2.x is a free open source framework that utilizes MVC (Model-View-Controller) architecture to develop web applications. Spring Framework offers an extensive programming model with the configuration for developing J2EE applications. Spring Boot made on top of the spring framework and set stand-alone Spring-based application much faster. Spring Boot applications require minimal Spring configuration, so the development is much more comfortable.

Java Persistent API (JPA) is a specification that offers object-relational mapping standards for managing java application relational database. It is used to persist, access and collect data between java objects and relational database. JPA is a specification, so it requires an implementation to perform any operation. JPA is considered as a link between ORM and relational databases. ORM tools like Hibernate, iBatis and TopLink implement JPA specifications for data persistence.

Annotations allow additional information about a program. They do not directly affect the code behaviour they annotate. Annotations start with '@'. They are mainly used for instructions during Compile time, Build-time and Runtime.

5.1 OBJECTIVES

This unit covers various J2EE open-source frameworks, and after going through this unit, you will be able to:

- explain Struts 2.x framework,
- describe the concept of Spring, Spring MVC and Spring Boot,
- use Java Annotation in programming, and
- describe Java Persistent API using Hibernate.

5.2 STRUTS 2.x FRAMEWORK INTRODUCTION

Struts1 was an open-source framework to develop Java EE web applications. Struts1 was created by Craig McClanahan and given to the Apache Foundation. The WebWork framework started developing independently by keeping base as the Strut framework to provide more features to the developers. Still, later on, they reunited and offered more robust Apache Struts 2. It extended the Java Servlet API and employed a Model, View, Controller (MVC) architecture.

Apache Struts 2.x is a web application framework based on OpenSymphony. It supports the complete application development cycle from the build to maintenance phase built on the MVC design pattern. In a typical Java web application, a client calls the server using a web form. Subsequently, the information pass to a Java Servlet that produces an HTML-formatted response after interaction with the database.

Suppose the information passed to a JavaServer Pages (JSP) document offers similar results after combining the HTML and Java code. But as these approaches mix the application logic with the presentation so they are considered insufficient for larger projects' maintenance. Strut2 is an MVC based framework where the application logic is isolated from the user interface layer. The application logic interacts with the database in the Model and is the lowest level of the pattern. The View is responsible for exhibiting data in the form of HTML pages to the client. The Controller holds the instances or the software code that passes the information between the Model and View.

5.2.1 Struts 2 features

Various new features were added in Struts2, making it a more robust and enterprise-ready framework for application development. Some of the features are as follows:

- POJO Forms and POJO Actions: Struts2 allows using any Plain Old Java Object (POJO) to get the form input or Action class. The POJO prevents the developers from interface implementation or inheritance of any class.
- Tag Support: Struts2 allows developers to use new tags where they have to write less code and so they are code efficient form tags
- AJAX support: Struts2 supports the asynchronous requests using the AJAX tags. AJAX is beneficial in improving the performance by only sending the required field data and avoiding unnecessary information.
- Easy integration: Struts2 supports easy integration with other frameworks like Spring, Tiles, hibernate etc.

- Template and Plugin support: Struts2 supports creating views with the help of templates and supports plugins to improve performance.
- Profiling: Struts2 supports debugging through integrated debugging and profiling and also has inbuild debugging tools.
- Easy to modify tags: Struts2 supports easy tag modification, requiring only basic HTML, XML and CSS knowledge.
- Promote less configuration: Struts2 supports default values for the various settings, which provide ease of access.
- View Technologies: Struts2 supports multiple view options like JSP, Freemarker, Velocity etc.

The client sends the request in terms of "Actions" to the Controller as per the specifications. The Controller demands the corresponding Action class to interact with the respective model code. Finally, returns an "ActionForward" string containing output page information to refer to the client.

5.2.2 Struts2 core components

The Model here is the business class where we write calls to business logic that implement with actions. The Model is executed with interceptors to intercepts specific purpose requests, and the dispatch servlet filter acts between the framework and the client as the Front Controller.

The JSP/HTML represents the View, and it is a combination of result types and results. The presentable MVC pattern expresses through JSP pages, Velocity templates, or some other presentation-layer technology. The action class characterizes the Controller. The Controller's job is to route the incoming HTTP request to the appropriate action. It maps requests to actions. The value stack and OGNL are linking with the other components through a common thread. Also, there are few other components to store configuration information of web application, actions, interceptors, results etc. Primarily, we need to create the following components for the Struts2 project-

- Configuration Files: This component creates the configuration files to couple the Action, View and Controllers like struts.xml, web.xml, struts properties. There are two main configuration files i.e. struts.xml and struts.properties file, where struts.xml file is used to override the applications default settings and struts.proerties file is used to modify the behaviour of the framework.

  The struts.xml file is created within the WEB_INF/classes directory and holds the configuration information to be modified as actions. The struts.properties file allows us to change the framework properties as per the requirements.

- Action: This component contains an action class that controls the user's interaction, the Model, and the View and holds the complete business logic and helps in the data processing. The Action class also responds to a user request and allows the framework to return the result back to the user based on the configuration file.

  We can create an Action class using a simple action class, where we may use any java class mandatorily containing an execute() method with the return type of string. The second method to create Strut 2 Action class is implementing action interface, and it also contains a execute() method implemented by the implementing class. The third method is by extending the

Action Support class and is the default implementation to provide various web application functionalities.

- Interceptors: This component is a part of the Controller, and it creates interceptors or uses existing interceptors as required. These are like servlet filters and execute before and after the processing of the request. Generally, they perform common actions like session logging, validation etc., for different actions.

  The interceptors are pluggable, so we can remove them from the application whenever we don't require any specific action. The removal doesn't require redeploying but simply editing the struts.xml file by removing the entry.

- View: This component creates JSPs to control the user's interaction to take input and display the final message.

- VALUESTACK: This component is a storage area used to store associated data during processing, i.e. Temporary, Model, Action and Named objects.

- Object Graph Navigation Language (OGNL): This component provides the functionality of retrieving VALUESTACK data and helps in converting data types.

StrutsPrepareAndExecuteFilter works as the Front Controller in Struts 2, and it implies that the Controller component acts first in the processing. Strut2 is Pull-MVC based architecture where data storage in ValueStack and render by view layer.

The Servlet Filter Object scans and regulates each incoming request and tell the framework which requests URLs map to which actions. XML-based configuration files or, generally, the Java annotations used to perform this operation.

5.2.3 Working and flow of Struts 2

Let us discuss the various steps of flow as shown in figure 5.1

1 The client/user sends an HTTP resource request to the server. It reaches to ServletContainer.

2 Web Container loads the web.xml and verifies the URL pattern if it matches; web Container forwards the request to ***StrutsPrepareAndExecuteFilter*** (Front Controller).

3 Based on the request URL mapping in struts.xml, the Filter dispatcher identifies the appropriate action class to execute.

4 Before the Action class is executed, the request is passed through the stack of interceptors. Interceptor allows common tasks defined in clean, reusable components such as workflow, validation, file upload etc., that you can keep separate from your action code.

5 The action class calls business logic. Action executed through the action methods performing database operations of storing or retrieving data.

6 Processed data from business logic sent back to the Action class.

7 Based on the result, the Controller identifies the View rendered. Before the response is generated, the stack of interceptors is executed again in the reverse order performing clean-up etc

8 View rendered to the user through the servlet controller.

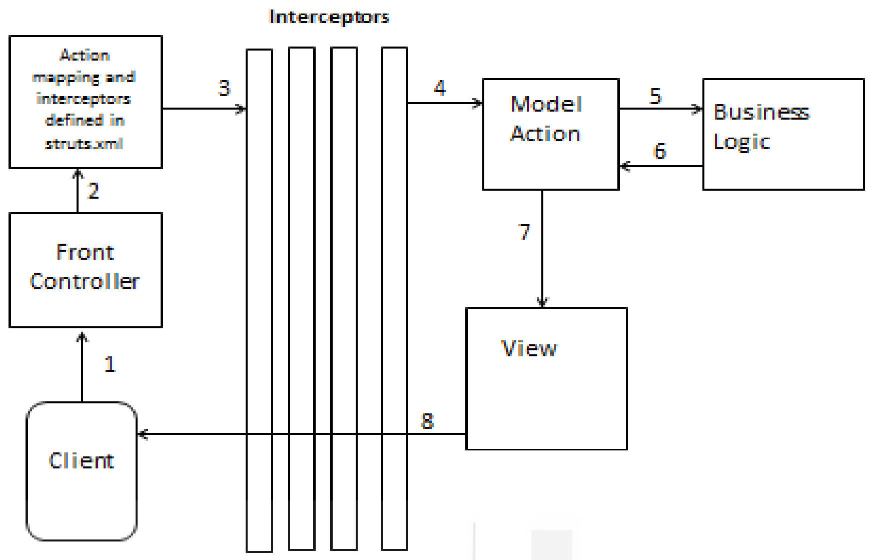


Figure 5.1 Strut 2 flow

Struts2 offers various methods to create Action classes, own interceptors for various common tasks and converters for rendering result pages. These can be configured using struts.xml or annotations and support many tags and OGNL expression language to help build web applications in Java.

☞ Check Your Progress 1:

1. What are the Struts2 core components?

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

2. Explain the operational flow of struts2.

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

3. What is the role of Action?

………………………………………………………………………………………………

………………………………………………………………………………………………

4. How are ValueStack and OGNL related to each other?

5. What is the role of interceptors in Struts2?

5.3 INTRODUCTION TO SPRING FRAMEWORK

The spring framework is a lightweight and open-source Java platform that provides solutions to various technical problems through J2EE applications. Initially, the Spring framework was released under the Apache 2.0 license, and Rod Johnson wrote it in June 2003. Spring is assumed as a framework of frameworks that supports other frameworks like struts, hibernate etc., through extensive programming and configuration model.

Spring is lightweight, with the basic version of only around 2MB. The spring framework is used to develop simple, reliable, and scalable Java applications. It builds web applications on top of the Java EE platform with different extensions.

It uses various new techniques to make easier J2EE development and eliminate the complications of developing enterprise applications. Different methods like Dependency Injection, Inversion of Control, Plain Old Java Object (POJO) and Aspect-Oriented Programming (AOP) are used to develop enterprise applications. It mainly focuses on business logic and offers better options and easy development for the Web applications compared to classic Java frameworks and Application Programming Interfaces (APIs) like servlets, JSP, JDBC etc.

Spring framework offers many advantages, and the following are the list of benefits:

- **POJO** - It allows developers to use POJOs for enterprise-class applications development, making them evade using an EJB container like an application server instead of using a servlet container like Tomcat.
- **Modular** - It supports a large number of modules, and due to the modular nature of Spring, the developer considers only the one they require for the development.
- **Integration** - It offers good integration with existing frameworks like ORM, logging frameworks and other view technologies etc.

- **Testability -** It offers simple application testing due to the use of POJOs. The environment-dependent code moves directly into the framework for injecting test data that supports dependency injection.
- **Web MVC -** It offers an elegant web MVC framework that delivers better options than other web frameworks such as Struts etc.
- **Central Exception Handling** – It offers an API that translates technology-specific exceptions into consistent, unchecked exceptions.
- **Lightweight** – It is lightweight compared to EJB containers and is very efficient with low resources like memory and CPU to develop and deploy applications.
- **Transaction management -** It provides proper scalability for the consistent transaction management interface. It supports a global transaction to local transactions by scaling up and down as required.

5.3.1 Spring Framework

The fundamental design principle of the Spring framework is "Open for extension, closed for modification". To better understand the Spring Framework, let us first discuss Dependency Injection (DI), Inversion of Control (IoC) and Aspect-oriented programming (AOP).

In any java-based project, we use objects as a unified one to act as a working application. For complex applications, we generally try to keep classes independent from other java classes. *Spring Dependency Injection* helps in sticking together these classes while keeping them independent, and this principle helps in reusability and facilitates unit testing.

The dependency injection facilitates creating an object for someone else and allows the direct use of dependency. Let us understand the vital concept of Dependency Injection using an example.

Consider a car class containing various objects such as wheels, fuel, battery, engine etc., and the car class is responsible for creating all dependent objects. If we decide to change the TATA battery to an AMARON battery at any stage, we need to recreate the car object with a new AMARON dependency.

We can change or inject the battery's dependencies at runtime rather than compile time using Dependency injection. Dependency injection is acting as a middleware for creating required objects and providing them to the car class.

There are typically three types of DI viz. constructor injection, setter injection, interface injection, and they are responsible for creating and providing the objects to the required classes. For any change in the object requirement, the DI will handle it automatically without bothering the concerned class. The inversion of the control principle behind DI states that class dependencies should be configured from outside and not statically, i.e., should not be hard-coded.

Let us summarize the inversion of control and dependency injection as

- **Inversion of Control** – Inversion of Control is based on the abstraction principle of object-oriented programming, where program objects depending not on implementations but interfaces of other objects for the interaction.
- **Dependency Injection** – It is a structural design pattern and is a vital aspect of the spring framework implementation of Inversion of Control. In this technique, Spring

containers inject an object into other dependencies and dependencies are implemented at the object creation time, i.e. the **dependent objects are created outside of a class and then provide these objects** through a class reference with other classes.

Constructor or setter method is used to implement Dependency injection through parameter passing, and the Libraries or the Inversion of Control containers are used to implement these approaches.

Another important concept of the spring framework is Aspect-oriented programming:

- **Aspect-oriented programming** – It is a programming style where instead of OOPS objects, the aspects are used. An aspect is a class that spans multiple application nodes distributed across methods, classes, object hierarchies and object models, such as logging, security, transaction etc. AOP defines specific policies and offers modularity, and in this paradigm, the program logic is broken down into distinct parts called concerns, enabling the application to adapt to changes. The **Spring Aspect-oriented programming** module intercepts an application through interceptors to add extra functionality during method executing before or after method execution.

The application classes are separated into different modules through Dependency injection and cross-cutting concerns separate from the affected objects through the Aspect-oriented programming.

Spring is modular and provides around 20 modules, and we can use them based on application requirements. Let us discuss a few essential modules as depicted in the figure 5.2:

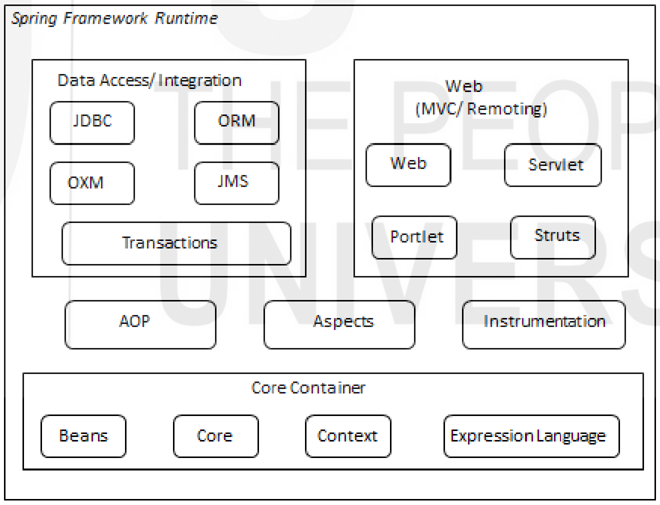


Figure 5.2: Spring Framework

Core container

It comprises Bean module, Core module, Context module, and Expression Language module.

- The Core module offers the framework's critical parts, like Dependency Injection (DI) or Inversion of Control (IoC) features.
- The Beans module offers the BeanFactory for the factory pattern implementation and is responsible for creating and managing the context structure unit.

- The Context module offers an ApplicationContext interface to access any object. It is built on a core and beans module base and inherits features from the Bean modules to facilitate internationalization.
- The Expression Language module offers object graph querying and manipulating at runtime through a powerful expression language.

Data Access/Integration Layer

It comprises the JDBC module, object-relational mapping module, Object/XML mapping module, Java Messaging Service module and Transaction modules.

- The JDBC module offers an abstraction layer that eases access by reducing tedious JDBC coding of manually connecting the database.
- The ORM module offers an integration layer supporting object-relational mapping APIs, like JPA, JDO, Hibernate, iBatis etc.
- The OXM module offers an abstraction layer for linking Object/XML mapping implementations for JAXB, XMLBeans, XStream etc.
- The Java Messaging Service (JMS) module offers features for creating, sending receiving messages.
- The Transaction module offers transaction management for programmatic and declarative classes with POJOs interfaces (Plain Old Java Objects).

Web Layer

The Web layer comprises the Web, Web-MVC, Web-Socket, and Web-Portlet modules.

- The **Web** module offers elementary features like uploading/ downloading files, initializing the Inversion of Control container using servlet listeners, creating a web application, etc.
- The **Web-MVC** module offers implementation for web applications through Spring MVC.
- The **Web-Socket** module provides client-server communication using WebSocket-based support in web applications.
- The **Web-Portlet** module offers the Model-View-Controller implementation with a portlet environment and mirrors the Web-Servlet module functionality.

Miscellaneous Modules

Following are a few other essential modules, viz. AOP, Aspects, Instrumentation etc.

- AOP module offers aspect-oriented programming capabilities.
- The Aspects module offers robust AOP framework integration AspectJ.
- The Instrumentation module offers provision to the class instrumentation and loader in the server applications.

5.3.2 Spring MVC

Spring MVC is a Java-based framework that provides an MVC design pattern and ready components to build a web application. The MVC pattern loosely coupled the important application aspects like input logic, business logic, and UI logic, resulting in developing flexible web applications. A spring MVC implements Inversion of Control, Dependency Injection etc., features and provide optimized solutions using DispatcherServlet class.

The application logic interacts with the database in the Model and is the lowest level of the pattern. The View is responsible for exhibiting data in the form of HTML pages to the client. The Controller holds the instances or the software code that passes the information between the Model and View.

The **Model** encapsulates the application data; the **View** renders that data and generating output display on the client. The **Controller** does the processing by building the correct model based on the specifications that are subsequently sent for rendering. A class DispatcherServlet plays a vital role in terms of mapping the request to the corresponding resource viz. the controllers, models, and views.

The DispatcherServlet

The Spring MVC *DispatcherServlet* request processing workflow is depicted in figure 5.3.

1. DispatcherServlet plays a vital role in the Spring MVC framework design. It handles all the HTTP requests and HTTP responses.
2. On receiving an HTTP request, the DispatcherServlet working as the Front Controller gets handler mapping information from the XML.
3. The HandlerMapping calls the appropriate Controller and based on the Get or Post methods, and the Controller appropriately calls the service method. The business logic based service method based sets the model data and subsequently returns the View name to the DispatcherServlet.
4. DispatcherServlet appropriately chooses the defined View by checking the ViewResolver entry in the XML file for the request.
5. After finalizing the View, DispatcherServlet passes the Model data and invoke specified View to render on the browser.

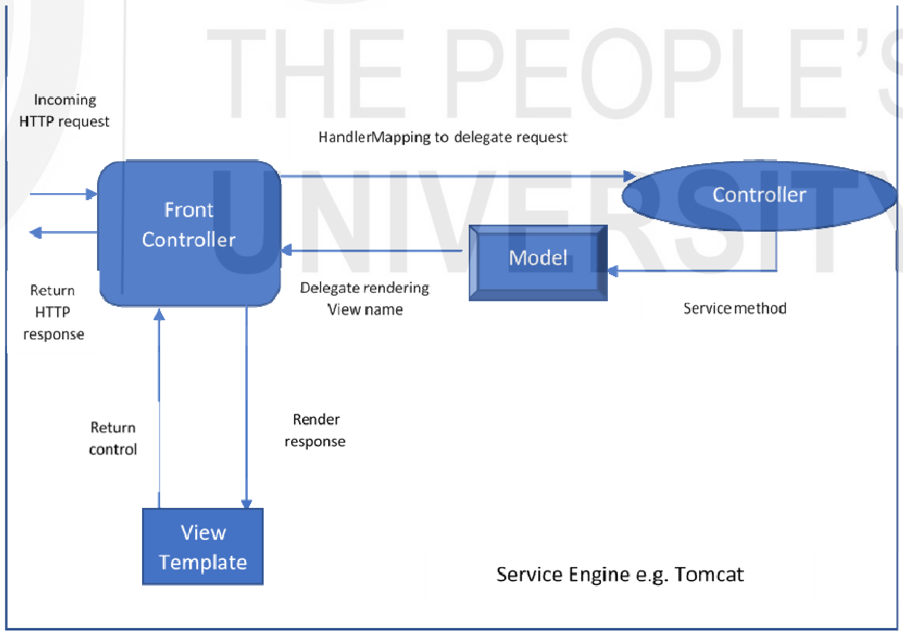


Figure 5.3: Spring Web MVC *DispatcherServlet* request processing workflow

Spring MVC Configuration

To create a simple Spring MVC application, we need to perform the following steps:

1. We need to add the spring context and spring web MVC dependencies to use the Spring MVC framework in the java project. We add spring-mvc.jar in the

application classpath for the java project and add the jar file in the /WEB-INF/lib folder for the web application.
2. Configure the DispatcherServlet through the web.xml file for handling requests through the spring container. For the application deployment, the Servlet container creates an instance for the ContextLoaderListner, which facilitates the loading of the webApplicationContext.
3. Configure the spring file to define beans for annotations usage and configure the view resolver for view pages.
4. Configure the web request mappings to handle the client requests.

The Spring also provides a built-in interface, the MultipartResolver interface, to upload file in Spring MVC Application. The implementation is easy and only requires a little configuration change with the controller handler method to handle the incoming file for processing. The Spring also supports annotation-based validations through bean variable and custom-based validators through controller class for data validation in Spring Web MVC Framework.

Spring MVC Framework advantages

- Separate roles - The different components of the flow are parts of the applicationContext that extends the plainApplicationContext. A specialized object fulfils the position of each element.
- Lightweight – To develop and deploy an application, the Spring MVC Framework uses a lightweight servlet container.
- Robust Configuration - Spring MVC offers a strong configuration for the framework and application classes.
- Reusable business code – Spring MVC Framework offers reusable code that permits usage of existing objects instead of creating a new one.

5.3.3 Spring Boot

Spring is commonly used to create scalable applications, but there are configuration overheads that are time-consuming and take some time. Sprint boot framework provides an efficient solution, with minimal efforts to develop the production-ready stand-alone spring-based application. Spring Boot module is built on top of the spring framework that offers a Rapid Application Development(RAD) to the new Spring-based web-based or straightforward applications. In spring boot, developers need not worry much about the configuration. It provides default configuration "Opinionated Defaults Configuration" for code and annotations and can quickly start new developments. It also includes functionality like code-reusability through Spring Boot Batch, which is very effective in transaction management, maintaining logs, job processing statistics etc.

Spring Framework and **Embedded Servers** like Tomcat comprises the spring boot, and the XML configuration part is not required, which reduces the cost and development time. As we have discussed, Spring integrates various modules during application development. For instance, we need to configure DispatcherServlet, View Resolver, Web Jar, XMLs, etc., but spring boot is an auto-configuration tool. Spring boot autoconfigures automatically using a web jar that helps us choose and set up the required number of modules fast and allows us to change as needed. Spring Boot framework bundles all the dependencies and provides stand-alone JAR file with embedded servers for the web application. The spring STS IDE or Spring Initializr may be used for application development.

Spring Boot components

The essential Spring Boot components are:

1. Spring Boot Starter: It includes a set of convenient descriptors to make application development much more manageable. The spring framework offers many dependencies, and the Spring Boot Starter aggregates them together to enhance productivity. The Spring Boot ensures the required libraries are added to the build.

2. Spring Boot autoconfiguration: It automatically configures the Spring application based on classpath parameter dependencies and makes development fast and easy.

3. Spring Initializer: It is a web application that helps create an internal project structure, i.e. a skeleton project, automatically reducing development time. It helps in quick start a new project using the web interface "Spring Initializr'.

4. Spring Boot Actuator: It allows us to monitor and manage the application while pushing it for production and helps us in debugging. It controls the application using HTTP endpoints. We may enable the actuator simply by adding the dependency to the starter, i.e. spring-boot-starter-actuator, and it is disabled if we don't add the dependency. It provides complete insight into the running spring boot application.

5. Spring Boot CLI: It allows us to write Groovy Spring Boot application with concise code without requiring traditional project build.

Advantages of Spring Boot

- **Stand-alone**: It can create **stand-alone** applications.
- **Embedded** HTTP servers: Deployment of the WAR files is not required as the web applications' testing may quickly be done using different **Embedded** HTTP servers.
- CLI tools: For developing and testing an application, the CLI tools are available
- **Better productivity:** For application development, there is no requirement of XML configuration that reduces the development time and improve productivity
- Plugins: A lot of plugins are available for the development and testing of an application. It offers several **plugins**.
- **Autoconfiguration**: The Spring boot configures the classes automatically based on the project requirements.
- **Starter**: Based on the application requirements, the Starter concept available in the pom.xml file confirms all the required JARs dependency downloads.

☞ Check Your Progress 2:

1. What is Dependency Injection?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2. What is Aspect-Oriented Programming?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

3. What is the difference between Spring Boot and Spring MVC?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

4. What are the essential components of Spring Boot?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

5.4 INTRODUCTION OF ANNOTATION

Java Annotations is a form of syntactic metadata that allows adding supplement info in the source code. We can add an annotation to packages, classes, interfaces, methods, and fields, but they do not change the program's execution, i.e. they are not a part of the program itself. The annotation representation is like @ followed by annotation name; for example, @Entity contains the annotation name following @, i.e. Entity and compiler will act accordingly.

Java Annotation applications

Java annotations can be used in various instructions, such as Compiler, Build-time, Runtime etc.

- Compiler instructions: The compiler uses annotations to detect errors or suppress warnings. @Deprecated, @Override & @SuppressWarnings are the three built-in annotations used to provide specific instructions to the compiler.
- Compile-time instructors: Software tools process the metadata information and subsequently pass the compile-time instructions to the compiler. The software tools generate required code, XML files etc.
- Runtime instructions: The Java reflections is used to access the Runtime annotations that provide instructions to the program at runtime.

5.4.1 Built-in Java Annotations

The @Override, @SuppressWarnings and @Deprecated are the Built-in Java Annotations used in Java code, whereas @Target, @Retention, @Inherited and @Documented are Built-in Java Annotations used in other annotations.

- **@Override** annotation ensures subclass overriding the parent class method, otherwise giving a compile-time error. For example, any spelling error can be handled using @Override annotation that provides the method overridden.

Let us understand the concept taking an example:

```
class ClassParent
{
   public void display()
   {
      System.out.println("Method of Parent class");
   }
}
class ClassChild extends ClassParent
{

   @Override
   public void display()
   {
      System.out.println("Method of Child class");
   }
}
class Main
{
  public static void main(String[] args)
  {
     ClassChild c1 = new ClassChild ();
     c1.display();
   }
}
```

```
OUTPUT:  Method of Child class
```

From example, we observe that the display() method is present in both the ClassParent superclass and ClassChild subclass. The display method is called from the main program actually called the subclass method instead of the method in the superclass.

- @Deprecated annotation marks deprecated methods and informs the user not to use such methods and prints warning that it may be removed in the future version.

For example:

```
class Main
{

   // @deprecated
   // use of deprecated method which has been replaced by a newerMethod()

  @Deprecated
  public static void methodDeprecated()
  {
    System.out.println("Method is Deprecated");
  }

  public static void main(String args[])
  {
    methodDeprecated();
   }
}
```

Output: Method is Deprecated

We observe that method has been declared deprecated from the example, and the compiler generates a warning message.

- @SuppressWarnings annotation is used to suppress potential compiler warnings. The annotation allows us to ignore a specific warning. The deprecation and unchecked are the two most common warnings, Deprecation annotation used to ignore deprecated method and unchecked to ignore raw types. For example, @SuppressWarnings("unchecked", "rawtypes") annotation will ignore warning at compile time for using the non-generic collection.

```
class Main
{
  @Deprecated
  public static void methodDeprecated()
  {
    System.out.println("Method is Deprecated");
  }

  @SuppressWarnings("deprecated")
  public static void main(String args[])
  {
    Main d1 = new Main();
    d1. methodDeprecated ();
  }
}
```

Output: Method is Deprecated

From the example, we observe that methodDeprecated has been declared deprecated, and the compiler generates a warning message, but we can avoid the compiler warning by using @SuppressWarnings("deprecated") annotation.

5.4.2 Java Custom Annotations

User-defined or Java custom annotations are very useful in developing readable code and are declared using @interface with the annotation name. The method declaration prohibited having any parameters and restricted to have primitives, String, Class, enums, annotations, and array of the preceding types as a return type in user-defined annotations.

We require a retention policy and a target to create an annotation. A retention policy defines the time duration in the program's life-cycle, where we have to retain the annotation. The retention of the annotation may be compile-time or runtime as per the annotation's retention policy. The three standard retention policy are Source, Class and Runtime.
Source: compiler discard annotations so invisible for compiler and runtime.
Class: The class file records the annotations but not retained by Java Virtual machine, so only visible by compiler.
Runtime: class file records the annotations but retained by Java Virtual machine at the runtime so visible to both the compiler and runtime.

- **@Target annotation** tag defines the valid Java constructs among the methods, class, fields etc. An annotation may associate with one or more targets.

- **@Documented annotation indicates the inclusion of new annotation into java document.**

- **@Inherited annotation** allows inheritance where we can apply an annotation to other annotation. Annotation inheritance is not available by default and so not available to child class from the parent class.

- @Repeatable annotation facilitates annotation more than once as, by default, it is applied once on a java element.

5.5 INTRODUCTION OF HIBERNATE WITH JAVA PERSISTENCE API (JPA)

We may require storing the information during any application development, and several applications use relational databases for this purpose. Although the JDBC API offers the database connectivity to perform various database operations for Java applications, it requires a lot of code to manage.
Spring offers API to integrate with Object-Relational Mapping (ORM) frameworks such as Java Data Objects, Hibernate etc. These tools simplify the database operations like data creation, manipulation, and access required to implement a persistent storage application.
Hibernate is a lightweight, open-source Java framework that simplifies the database interaction during Java application development. Hibernate implements the Java Persistence API specifications and bridges the gap between the java object and the relational database to provide data persistence.
JPA specifications define a common abstraction that we can use in our program to interact with ORM products.

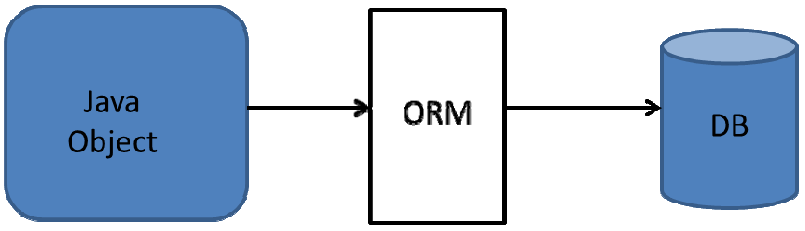


Figure 5.4: Java Persistence API specifications

5.5.1 Hibernate Architecture

The Hibernate framework is built on top of existing Java API, and the architecture is characterized into four layers viz. Database, Back-end API, Hibernate framework, and Java application layer.

The core components of Hibernate architecture are:

- Configuration object holds the configuration properties viz. database configuration file and class mapping files.
- SessionFactory is a factory of sessions that provides a factory method to get the session object. It is a heavyweight object, immutable and is available to all the sessions. It is created at the time of application startup for later use and persists till the Hibernate is running.
- Session objects provide an interface between the application and the database. It is a lightweight object and gets instantiated whenever an application requires a database interaction. The session object offers a method to create, read, update and delete operations.
- TransactionFactory is a factory of Transaction that offers a method for transaction management.
- ConnectionProvider is an optional factory of JDBC connections.
- Transient or Persistent objects are database objects. In the transient states the new objects created in the Java program are not associated with any hibernate session. Still, in the case of the persistent state the object is associated with a hibernate session.

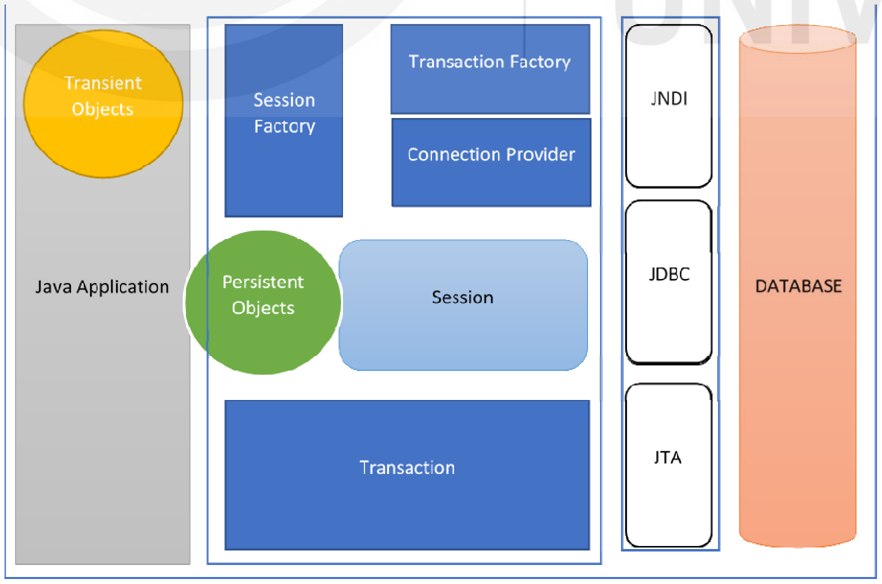


Figure 5.5: Hibernate architecture

Hibernate provides the support for persisting the Collections and depending on the type of interface; Hibernate injects the persistent collections. The Persistent object contains a persistent state saved to the database, whereas the transient object does not save or is associated with any session yet. A newly created entity is transient until it persisted. Detached state is when a previously persistent object is currently not associated with any session. We may switch instance from Transient to persistent by calling save(), persist(), or saveOrUpdate() and switching Persistent instances to transient by calling delete(). We may switch instance from Detached to persistent by calling update(), saveOrUpdate(), lock() or replicate(). The transient or detached instance state may convert to a new persistent instance by calling merge().

5.5.2 Hibernate Framework advantages

The Hibernate framework advantages are as follows:

- Open Source - It is an open-source and lightweight framework.
- Sound Performance - The hibernate framework offers better performance due to the internal cache mechanism. Out of the two caches, the first level cache is by default enabled for performance.
- Powerful query language – Hibernate provides the object-oriented version of SQL i.e. Hibernate Query Language (HQL) that creates database-independent queries. Any database change would not lead to a maintenance problem as the queries are not database-specific.
- Automatic Table Creation - It creates automatic database tables.
- Simplifies Complex Join - Hibernate framework provides ease in the data fetching from multiple tables.
- Query Statistics - It provides query statistics and database status.
- Transaction management - Hibernate offers transaction management to avoid any data inconsistency.

5.5.3 Java Persistence API

Java Persistence API is a Java specification that offers functionality and standard to Object Relational Mapping tools that manage relational data in Java applications. The JPA facilitates the mapping, storing, updating and retrieving data from the java objects to the relational database and vice versa. The **javax.persistence** package contains the Java Persistence API classes and interfaces to bridge the relational database and object-oriented programming gap. For the database access applications, the JPA automate the implementation as it requires only a repository interface and custom finder methods that reduce the code complexity and improves efficiency.

JPA Object Relational Mapping

Hibernate automatically manipulates domain model entities using ORM tools, and thus it doesn't require modifying all associated insert and update command upon adding a new column.

Object Relational Mapping (ORM) facilitates developing and maintaining a relation between an object state and a relational database column. Hibernate ORM automates the Object-relational mapping task process, where the ORM layer converts the java classes and objects to interact with the relational database. It provides various database operations smoothly and efficiently, like insert, update, deletes etc. The persisted object name becomes the table name, and the fields become the columns. There are different ORM mapping types but before discussing them, let us understand the persistent objects, also called entities.

Entities

An entity is a databases table is associated with a group of states, and each instance correspondingly represents a row in the table. JPA Entities is an application-defined object which persists in the database. The persistent entity state may represent using persistent fields or persistent properties. The object/relational mapping annotations are used by these persistent fields or persistent properties for mapping the entities and entity relationships to the relational database.

- EntityManager - The model class objects or instances form the persistence context and interact with the database, we need an EntityManager instance. Generally, the EntityManager manages the entity instances like creating, updating, removing or finding entities and managing their life cycle.
- EntityManagerFactory - EntityManagerFactory is linked with a persistence unit, and it creates an EntityManager.

An entity must follow the property of persistability with unique Persistent Identity, and it should support Transactionality. It means object can be accessed any time after storing in the database through the object identity and the changes in the database are all atomic following transactionality. Each Entity is associated with some metadata represented through annotation that form tags that persist inside the Class and XML persist outside the class in an XML file.

A Java class transform into an entity using No-argument Constructor and Annotation by adding @Entity and @Id annotation. @Entity is placed on the class name to indicate that this class is an entity. @Id is placed on a specific field treated as a primary key holding the persistent identifying properties.

The important characteristics requirement that an entity class must have:

- The class must annotate with the javax.persistence.Entity annotation
- The class, methods or even the persistent instance variables must not be declared final.
- The class must have a no-argument constructor that may be public or protected.
- The class must have a Serializable interface if an entity instance is passed by value through a remote interface.
- The abstract and concrete classes can be annotated with the Entity annotation, and the Entities and non-entity classes can extend each other.

Entity Primary Keys

An Entity must have a corresponding unique object identifier or primary key that enables locating a particular entity instance. Based on the persistent properties, an entity may have a simple denoted by javax.persistence.Id annotation or a composite key denoted by javax.persistence.EmbeddedId and javax.persistence.IdClass annotations.

Types of ORM Mapping

The various ORM mappings are as follows:

- One-to-one – The @OneToOne Annotation represents association of one entity instance to single instance of another entity.
- One-to-many – The @OneToMany Annotation represents association of one entity instance to many Entity instance on another entity.

- Many-to-one – The @ManyToOne Annotation represents association of many entity instance to one Entity instance of another entity.
- Many-to-many – The @ManyToMany Annotation represents association of many entity instance to many Entity instance of another entity.

The JPA simplify the database programming by importing interfaces and classes from the javax.persistence package. JPA defines object-oriented query language, Java Persistence Query Language (JPQL) is very similar to SQL. Unlike SQL, which operates directly with the database tables, the JPQL interacts through the java objects.

Let us take an example to understand a simple database program using Hibernate framework with JPA annotation.

Step1: create a database named empdb using MYSQL,

create database empdb;

Then create a table

CREATE TABLE EMP (id INTEGER not NULL, first VARCHAR(30), last VARCHAR(30), age INTEGER, city VARCHAR(30), salary INTEGER, PRIMARY KEY (id));

Step 2: We Simplify the process of project building using Maven project in Eclipse. The details about Maven will be discussed in the next unit.

In the Eclipse IDE, we will go to through the File->New->Project->Maven->Maven Project to open a New project and click next.

We only provide the following project information on the New Project screen

- Group Id: net.codejava.hibernate

- Artifact Id: HibernateJPADemo

After clicking next, we will add Hibernate, JPA and MySQL Connector Java dependencies in Maven's Project Object Model (pom.xml). By simply adding the dependencies the Maven automatically downloads the required jar files. In the pom.xml file, simply add the following XML before the </project> tag:

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.12.Final</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.18</version>
    </dependency>
    <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
</dependency>
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>eclipselink</artifactId>
    <version>2.7.0</version>
</dependency>
</dependencies>
```

Step 3: Now let us create a net.codejava.hibernate java package under the folder src/main/java to put our java classes.

First, we create a model class employee with some getter and setter methods. We add JPA annotations in this mode class to map it with the corresponding database table.

```
package net.codejava.hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table
public class employee
{

        private Integer userid;
        private String firstname;
        private String lastname;
        private Integer age;
        private String city;
        private Integer salary;
        @Column(name = "id")
        @Id
        public Integer getUserid()
        {
                return userid;
        }
        public void setUserid(Integer userid)
        {
                this.userid = userid;
        }
        @Column(name = "first")
        public String getFirstname()
        {
                return firstname;
        }
        public void setFirstname(String firstname)
        {
                this.firstname = firstname;
        }
        @Column(name="last")
        public String getLastname()
        {
                return lastname;
        }
        public void setLastname(String lastname)
        {
                this.lastname = lastname;
        }
        public Integer getAge()
        {
                return age;
        }
        public void setAge(Integer age)
        {
                this.age = age;
        }
        public String getCity()
        {
                return city;
        }
        public void setCity(String city)
        {
                this.city = city;
        }
        public Integer getSalary()
        {
                return salary;
        }
        public void setSalary(Integer salary)
        {
                this.salary = salary;
        }
}
```

The annotation @Entity is placed before the class definition and maps the class with the database table. The annotation @Table is also placed before the class definition and is used if the class name and the table name are different. The @Column annotation is placed before the getter method if the instance field of the class is different to the database column name. The @Id map the primary key column in the table.

Step 4: Next, create a persistence.xml configuration file for JPA, it will be placed in the new folder named META-INF that is created under src/main/resources folder. The Hibernate uses this configuration file to connect with the database. Let us add the following XML code in the persistence.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
        http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="empUnit">
        <properties>
            <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/empdb" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password" value="admin" />
            <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
        </properties>
    </persistence-unit>

</persistence>
```

The first property tag tells us about the JDBC URL value pointing to the database. The second and third tags provide the username and password; the subsequent tag specifies the JDBC driver and the last two property tags tell the Hibernate to show and format the SQL statements.

Step 5: Finally, we write a test program to check the overall working of the example by updating the employee entity instance using JPA. We create a new employeeManager.java class under the src/main/java folder with the main() method. We add the following code:

```
package net.codejava.hibernate;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class EmployeesManager
{
  public static void main(String[] args)
  {
    EntityManagerFactory factory =
Persistence.createEntityManagerFactory("empUnit");
    EntityManager entityManager = factory.createEntityManager();
    entityManager.getTransaction().begin();
    employee newEmployee = new employee();
    newEmployee.setUserid(702);
    newEmployee.setFirstname("Vikash");
    newEmployee.setLastname("Sharma");
    newEmployee.setAge(38);
    newEmployee.setCity("Vizag");
    newEmployee.setSalary(7000);
    entityManager.persist(newEmployee);
    entityManager.getTransaction().commit();
    entityManager.close();
    factory.close();
  }
}
```

In the main() method, we first create an EntityManager and begin the transaction; subsequently, we save newEmployee, a new employee object using the persistent (object) method. Finally, we close the EntityManager and EntityMangerFactory after completing the transaction.
The output will show that the Hibernate print the SQL statement and the program executed successfully and can be verified through MYSQL command line. Using Hibernate/JPA we can add a new row without explicitly writing any SQL query.

Output:

```
Hibernate:
  insert
  into
    employee
    (age, city, first, last, salary, id)
  values
    (?, ?, ?, ?, ?, ?)
Apr 24, 2021 11:24:01 PM
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/emp]
```

☞ Check Your Progress 3:

1. Explain the different entity bean states.

2. What do you understand by Java Persistence API?

3. How do you differentiate between Hibernate and JPA?

4. How do you Create an Annotation?

5. What are the differenty types of ORM mapping?

5.6 SUMMARY

In this unit, we have introduced various open-source J2EE frameworks. Apache Struts 2.x is a free open source framework that offers an extensive programming model. It supports the complete application development cycle from build to maintenance based on the MVC design pattern. We have discussed various Struts2 features, mapped them with the core components, and elaborated on the working flow of Struts2 from the clients' request to the View.

The spring framework is thought of as a framework of frameworks supporting other frameworks like struts, hibernate etc., through extensive programming and configuration model. The fundamental design principle of this framework is "Open for extension, closed for modification" and is achieved with the concept of Dependency Injection (DI), Inversion of Control (IoC) and Aspect-oriented programming (AOP). The *Spring Dependency Injection* facilitates the reusability and enables unit testing. The inversion of control principle behind DI states that class dependencies need not be hard-coded. The **Spring Aspect-oriented programming** module intercepts an application through interceptors and facilitates adding extra functionality during method executing before or after method execution.

Spring MVC and Spring Boot frameworks provide more flexibility to the developers. Spring MVC provides default configurations to build a Spring-powered framework. Sprint boot framework offers an efficient solution, with minimal efforts to develop the production-ready stand-alone spring-based application. Spring Boot module builds on top of the spring framework, and it gives the Rapid Application Development to the new Spring-based web-based or straightforward applications. Spring Boot reduces the code length in developing a web application using annotation configuration and default codes.

Java Annotations is a form of syntactic metadata that add additional information about a program. They do not have a direct effect on the behaviour of the code they annotate. Java annotations are used for various purposes, such as Compiler instructions, Build-time instructions, Runtime instructions, etc. Annotations start with '@'. They are mainly used for instructions during Compile time, Build-time and Runtime.

Java Persistent API (JPA) is a specification that provides object-relational mapping standards in java applications. JPA is considered as a link between ORM and relational databases. Object Relational Mapping (ORM) facilitates developing and maintaining a relation between an object state and a relational database column. ORM tools like Hibernate, iBatis and TopLink implements JPA specifications for data persistence. We will be discussing these topics in more details in the subsequent units.

5.7 SOLUTIONS/ ANSWER TO CHECK YOUR PROGRESS

Check Your Progress 1

Answer 1: The Struts2 core components are Action, Interceptors, View, VALUESTACK, Object Graph Navigation Language (OGNL) and Configuration Files.

Answer 2: The client request is sent to the server through the browser. After matching the request pattern, the server loads the web.xml and forwards it to the FilterDispatcher. The request passed through the interceptor before the appropriate action class is executed. Based on the business logic function, the action class or the Controller gets the processed data from the database. The Controller decides on the rendered View depending on the result, and only after the interceptor execution result is generated.

Answer 3: Action component contains an action class that controls the user's interaction, the Model, and the View. It holds the complete business logic and prepares the response based on the client request.

Answer 4: A ValueStack store action and all the data related to action where the OGNL facilitates manipulating the data available at ValueStack.

Answer 5: The Interceptor component is a crucial part of the Controller and is mostly responsible for framework processing. These are like servlet filters and executes before and after the processing of the request. Generally, perform common actions like session logging, validation etc., for different actions.

Check Your Progress 2

Answer 1: Dependency Injection implements the principle of Inversion of Control (IoC), allowing creating and binding the dependent objects outside of a class. It separates object creation from its usage and thus reduces the boilerplate code based on business logic.

Answer 2: Aspect-oriented programming – Aspect-oriented programming (AOP) is a programming style where aspects are used instead of OOPS objects. An aspect is a class that contains advice and joinpoints. It spans multiple application nodes distributed across methods, classes, object hierarchies, and object models, such as logging, security, transaction etc. AOP defines specific policies and offers modularity, and in this paradigm, the program logic is broken down into distinct parts called concerns, enabling the application to adapt to changes dynamically. The Spring Aspect-oriented programming module intercepts an application through interceptors to add extra functionality during method executing before or after method execution.

Answer 3: The spring framework aids in developing simple, reliable, and scalable Java applications. It follows the MVC design pattern implementing all the basic features of a core spring framework. Spring MVC provides default configurations to build a Spring-powered framework. Sprint boot framework offers an efficient solution, with minimal efforts to develop the production-ready stand-alone spring-based application. Spring Boot module builts on top of the spring framework, and it gives the Rapid Application Development to the new Spring-based web-based or straightforward applications. Spring Boot reduce the code length in developing a web application using annotation configuration and default codes.

Answer 4: The essential Spring Boot components are:

1. Spring Boot Starter: It includes a set of convenient descriptors to make application development much more manageable. The spring framework offers many dependencies, and the Spring Boot Starter aggregates them together to enhance productivity.
2. Spring Boot autoconfiguration: It automatically configures the Spring application based on classpath parameter dependencies and makes the fast and easy development.

3. Spring Initializer: It is a web application that helps create an internal project structure, i.e. a skeleton project, automatically reducing development time.
4. Spring Boot Actuator: It allows us to monitor and manage the application while pushing it for production and helps us in debugging. It controls the application using HTTP endpoints. We may enable the actuator simply by adding the dependency to the starter, i.e. spring-boot-starter-actuator, and it is disabled if we don't add the dependency.
5. Spring Boot CLI: It allows us to write Groovy Spring Boot application with concise code.

Check Your Progress 3

Answer 1: The three states where an entity bean instance exists are:

- Transient: It is a state when an object is not associated with any session or is not persisted.
- Persistent: It is a state when an object is linked with a unique session.
- Detached: It is a state when a previously persistent object is currently not associated with any session.

We may switch instance from Transient to persistent by calling save(), persist(), or saveOrUpdate() and switching Persistent instances to Transient by calling delete(). We may switch instance from Detached to persistent by calling update(), saveOrUpdate(), lock() or replicate(). The transient or detached instance state may convert to a new persistent instance by calling merge().

Answer 2: Java Persistence API is a Java specification that offers functionality and standard to Object Relational Mapping tools for managing relational data in Java applications. It acts as an interface to bridge the relational database and object-oriented programming gap. For the database access applications, the JPA automate the implementation as it requires only a repository interface and custom finder methods that reduce the code complexity and improving efficiency.

Answer 3: Java Persistence API (JPA) is the interface defined in javax.persistence package and the Hibernate is the implementation-defined in org.hibernate package. In the JPA, we use Entity Manager for handling the data persistence through the Java Persistence Query Language (JPQL). But in Hibernate, we use the Sessions for the persistence of data through the Hibernate Query Language (HQL).

Answer 4: Java Annotations is a form of syntactic metadata that adds supplement information into our source code. The annotation representation @Entity, where @ sign specifies to the compiler that following @ is an annotation.

Answer 5: The various ORM mappings are as follows:

- One-to-one – The @OneToOne Annotation represents one to one Entity instance association.
- One-to-many – The @OneToMany Annotation represents one to many Entity instance association.
- Many-to-one – The @ManyToOne Annotation represent many to one Entity instance association.
- Many-to-many – The @ManyToMany Annotation represent many to many Entity instance association.

5.8 REFERENCES/FURTHER READING

1. Craig Walls, “Spring Boot in action” Manning Publications, 2016. (https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf)
2. Paul Deck, “Spring MVC: a Tutorial”, Brainy Software, 2016.
3. Ian Roughley,”Practical Apache Struts 2 Web 2.0 Projects”, Dreamtech Press, 2008.
4. Cazzola, Walter, and Edoardo Vacchi, "@ Java: Bringing a richer annotation model to Java”, Computer Languages, Systems & Structures 40(1), pp.2-18,2014.
5. https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

UNIT 6 FRAMEWORKS AVAILABLE FOR J2EE DEVELOPMENT –STRUTS, SPRING BOOT AND HIBERNATE

6.0 INTRODUCTION

A conventional website mainly delivers only static pages, while a web application has the ability to create dynamic responses. A web application receives input from users, interacts with the database and executes the business logic to customize the view as a response to the users.

Web applications use ***Servlet and JSP*** technologies for dynamic response. The Servlet and JSP may contain page design code, control execution code and database code. Mixing design code, control execution, and database code together make a large application difficult or impossible to maintain.

The ***Model-View-Controller (MVC)*** architecture provides the separation of concerns and makes the application easy to maintain and develop. The model represents the business logic, view represents the page design, and controller represents the navigational code. The ***Spring MVC and Struts*** are based on MVC architecture which helps developers to create a maintainable web application efficiently.

Build process of a software project involves a series of processes such as downloading required dependencies, putting jars on classpath, generating source code (for auto-generated code), source code compilation, tests execution, packaging compiled code along with the dependencies into deployable artifacts such as WAR, EAR or JAR. All processes can be done manually by developers but it's time consuming and error prone.

Apache Maven simplifies the build process and automates all the involved processes. Maven is a popular open-source build tool developed by the Apache Group to build, publish, and deploy several projects at once for better project management.

6.1 OBJECTIVES

After going through this unit, you will be able to:

- explain Model1 and Model2(MVC) architecture,
- describe Struts architecture and execution flow,
- describe Spring Boot and Spring MVC feature,
- describe features of Hibernate/JPA,
- differentiate among Spring, Spring MVC and Spring Boot,
- use Maven tool to automate the build process, and
- describe Maven build life cycles, build phases and build goals

6.2 STRUTS: FEATURES

The Struts framework was developed by Craig McClanahan and he donated it to Apache foundation in May, 2000. In June, 2001 Apache released the first version of Struts as Struts 1.0. Struts 2.5.22 is the most current version of the Struts framework in April, 2021.

Apache Struts2 is an elegant, popular Java Model-View-Controller (MVC) framework to develop MVC based web applications. Struts2 is a complete rewrite of Struts architecture. Struts2 is completely different from Struts1, and it is not the next version of Struts1 framework. The **Webwork** framework was started with **Struts** as base in order to provide a simplified and enhanced framework based on Struts to develop the MVC based web application easily. After a while Webwork framework and Struts community joined hands together to work on this enhanced framework which became famous as Struts2 framework.

The Struts2 is enriched with many important features such as support to POJO based actions, Configurable MVC Components, Integration support to various frameworks such as Spring, Tiles, Hibernate etc., AJAX support, Validation support, support to various theme and Template such as JSP, Velocity, Freemarker etc.

6.2.1 Model 1 Vs Model 2 (MVC) Architecture

The knowledge about design models helps us to decide the architecture of a web application. There are two types of design models.

- Model 1 Architecture
- Model 2 (MVC) Architecture

6.2.1.1 Model 1 Architecture

In earlier days, web application development using Servlet had multiple issues like mixing Business and Presentation logic and recompilation of Servlet if any design code changed.

In model 1 architecture, JSP pages were the focal point for the entire web application. JSP provides the solutions to the problems of Servlet technology. JSP provides a better separation of concern not to mix business logic and presentation logic. Re-Deployment of web applications is not required if the JSP page is modified. JSP supports JavaBean, custom tags and JSTL to keep the Business logic separate from JSP. Model 1 architectural diagram is shown below.

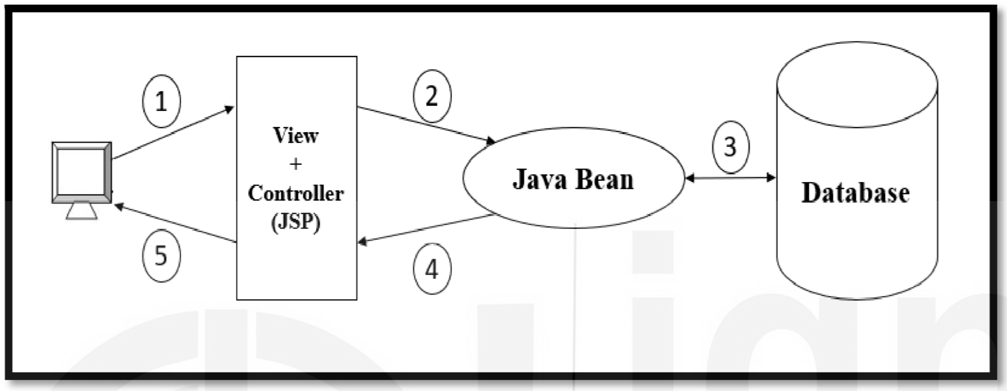


Figure 6.1 : Model 1 Architecture

Notice that there is no **Servlet** involved in the process. The client request is sent directly to a JSP page, which may communicate with JavaBeans or other services, but ultimately the JSP page selects the next page for the client.

Advantages

- Quick and easy way to develop a web application

Disadvantages

- **Decentralized navigation Control:** There is no central control to determine the navigation since every JSP page contains the logic to determine the next page.
- **Hard to maintain:** If a JSP is renamed, then every other JSP which refers to the renamed JSP, needs to be modified.
- **Time Consuming:** Reusable components like custom tags development requires more time.
- **Difficult to extend:** Model1 architecture is not suitable for large and complex applications.

6.2.1.2 Model 2 (MVC) Architecture

In Model 2 architecture contrast to the Model 1 architecture, a Servlet known as ***Controller Servlet*** intercepts all client requests. The Controller Servlet performs all the initial request processing and determines which JSP to display next.

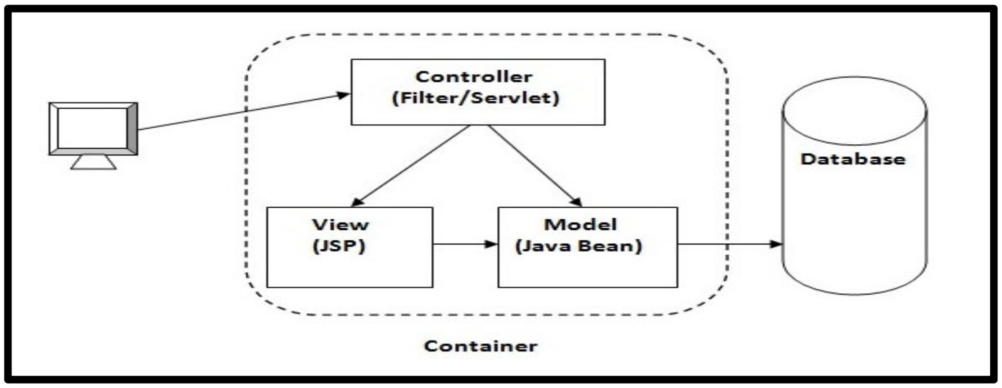


Figure 6.1: Model 2 (MVC) Architecture

In this architecture, the client never sends requests directly to the JSP. This architecture enables us to perform authentication and authorization, centralized logging etc. Once request processing is finished, the servlet directs the request to the appropriate JSP page.

As it can be observed that the key difference between the two architectures is the ***Controller Servlet*** introduced into Model 2, which provides a single point of entry and encourages more reuse and extensibility than the Model 1 architecture. The Model 2 architecture clearly provides the separation of business logic, presentation logic, and request processing. This separation is commonly known as ***Model-View-Controller*** **(MVC)** pattern.

Advantages

- Centralized navigation Control
- Easy to maintain
- Separation of concern
- Easy to extend
- Easy to test

Disadvantages

- Developer writes the Controller Servlet code and it needs to be compiled and redeployed if any logic changes for Controller Servlet.

Struts provides the configurable MVC support with declarative approach for defining view components, request mapping in order to resolve the drawbacks of Model2 architecture. In Struts 2, we define all the action classes and view components in ***struts.xml*** *file.*

6.2.2 Struts2 Architecture

Struts2 is a Pull-MVC (MVC2) based architecture, in which the view layer pulls data stored into Value Stack to render. Struts2 implements the Model-View-Controller architecture with the following five components.

- Interceptors
- Actions
- Results/ Result Types
- Value Stack / OGNL
- View Technologies

Struts2 architectural diagram is shown below.

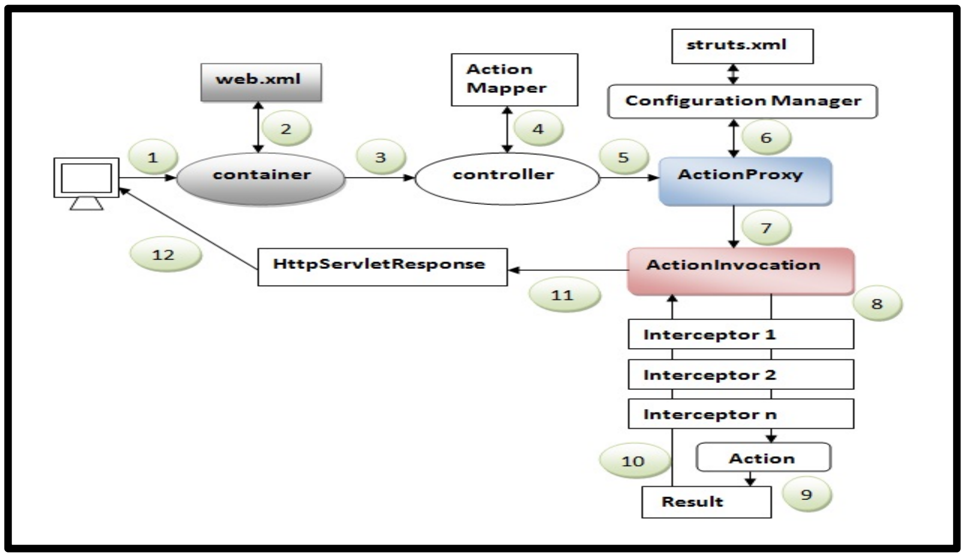


Figure 6.2: Struts2 Architecture

Interceptors: Interceptors are conceptually similar to servlet filters. The Interceptors are used to implement the cross-cutting functionality such as validation, exception handling, internationalization etc. The Struts2 provides many preconfigured and ready to use interceptors out-of-the-box.

Actions: Actions are the core of Struts2 framework. These are used to provide the business logic which executes to serve the client request. Each URL is mapped to a specific action.

In Struts2, the action class is simply POJO (Plane Old Java Object). The only prerequisites for a POJO to be an action is that there must be a no-argument method that returns either a String or Result Object. If the no-argument method is not specified, the execute() method is used to perform business logic processing.

Results / Result Types: The next step after execution of an action is to display a view. The **<results>** tag plays the role of a **view** in the Struts2 MVC framework.

Some navigation rules are associated with the results. For example, if the action method is to register a user, there are three possible results.

1. Successful registration
2. Unsuccessful registration
3. Already registered

For the above, the action method will be configured with three possible outcome strings and three different views to render the outcome. Struts2 does not force us to use JSP as view technology. Struts2 is based on the MVC paradigm, which makes the components configurable.

Struts2 has many predefined **result types,** and the **default** result type is **dispatcher** to dispatch to JSP pages. Struts2 defines different result types for different view technologies such as **Velocities, Tiles, XSLT, Freemarker**.

ValueStack / OGNL: A ValueStack represents a stack that contains application-specific objects such as action objects and other model objects. The objects in the ValueStack are available to the response on UI pages. There are various tags available for JSP, Velocity and Freemarker to access the ValueStack. The ValueStack allows us to put objects, query objects and delete objects using **OGNL**.

The **Object-Graph Navigation Language** (OGNL) is a very powerful expression language similar to the JSP Expression Language. Struts2 OGNL performs two important tasks – **data transfer** and **type conversion**. The OGNL in Struts2 takes the request parameters from servlet context, performs the type conversion using the in-built type converters to the corresponding type in bean and transfers it to corresponding java variable.

6.2.3 Struts2 Features

The Struts2 is enriched with many important features, and some of them are listed as follows.

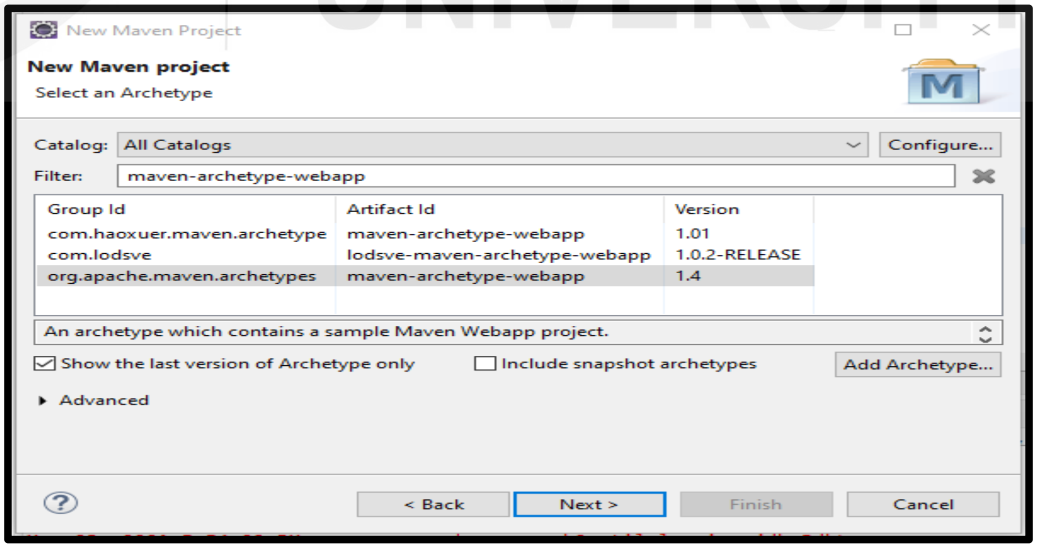
1. **Decoupling of view from action classes:** Struts2 allows us to associate a view with multiple action classes.
2. **Ready to use view components:** Struts2 provides many ready to use view components such as stylesheet driven tags which reduces the lines of code into the application for form design and form validation.
3. **Intelligent defaults in all configuration files:** One of the important features of the Struts2 is the default value for the configurable properties. Hence, developers do not need to specify the obvious default values into configuration files.
4. **Use of OGNL:** Struts2 uses the expression language OGNL, which is more powerful and flexible than JSTL.
5. **Type conversion mechanism:** Struts2 supports any type of properties with the help of OGNL, while in Struts1 all the properties are mostly of string type.
6. **Strong validation support:** Struts2 provides a strong validation mechanism with the support of Xwork validation framework, which provides both client-side and server-side validation. It also supports custom validation with the support of Validate() method.

6.2.3 Struts2 Example

The concepts and the components of Struts have been explained in previous sections. This section explains step by step to create a simple Struts2 web application. The required tools and software are as follows.

- Eclipse IDE
- Maven
- Java 8 or above
- Struts 2.x

Step 1: Create a maven project in eclipse with the following group id and artifact id.



Catalog: All Catalogs

Filter: maven-archetype-webapp

Group Id	Artifact Id	Version
com.haoxuer.maven.archetype	maven-archetype-webapp	1.01
com.lodsve	lodsve-maven-archetype-webapp	1.0.2-RELEASE
org.apache.maven.archetypes	maven-archetype-webapp	1.4

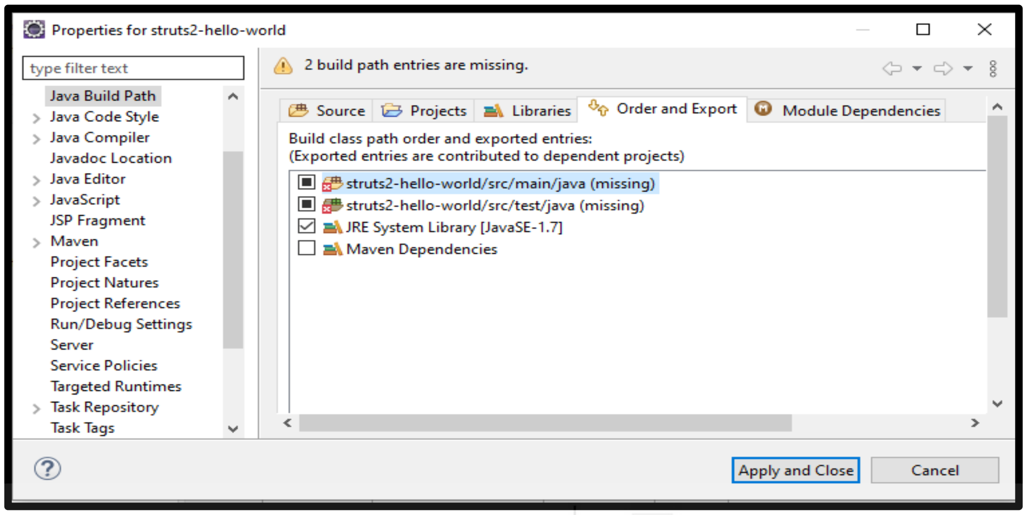
An archetype which contains a sample Maven Webapp project.

[✓] Show the last version of Archetype only

[ ] Include snapshot archetypes

Figure 6.3: Create Maven Project Into Eclipse

Step 2: Right click on the project and go to properties of the project. Check the Java Build Path. If some error is there, select the JRE System Library as shown in the screenshot.


Properties for struts2-hello-world

2 build path entries are missing.

Build class path order and exported entries:
(Exported entries are contributed to dependent projects)

- [■] struts2-hello-world/src/main/java (missing)
- [■] struts2-hello-world/src/test/java (missing)
- [✓] JRE System Library [JavaSE-1.7]
- [ ] Maven Dependencies

Figure 6.4: Fix Build Path Error

Step 3: Right click on Project folder and click on New option to add a Source Folder as src/main/resources into the project. Final folder structure of the project is as follows.

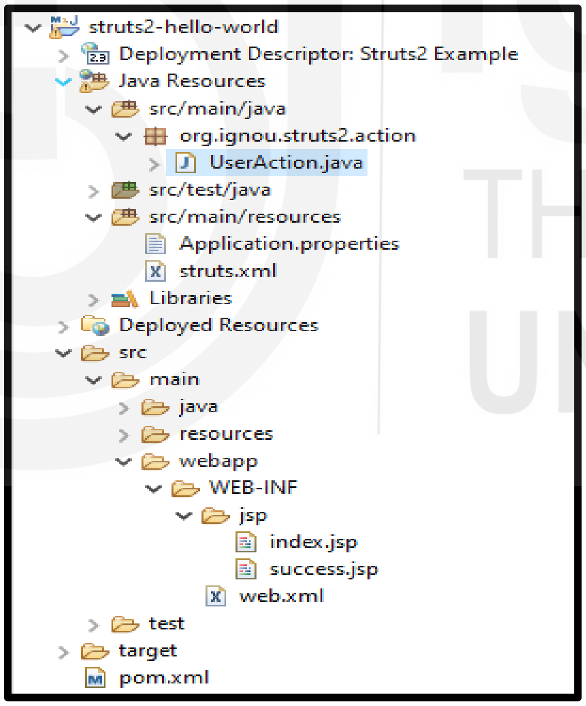

```
struts2-hello-world
  Deployment Descriptor: Struts2 Example
  Java Resources
    src/main/java
      org.ignou.struts2.action
        UserAction.java
    src/test/java
    src/main/resources
      Application.properties
      struts.xml
    Libraries
  Deployed Resources
  src
    main
      java
      resources
      webapp
        WEB-INF
          jsp
            index.jsp
            success.jsp
          web.xml
    test
  target
  pom.xml
```

Figure 6.5: Project Structure into Eclipse

Step 4: Add the following maven dependency into pom.xml

```
<!-- https://mvnrepository.com/artifact/javax.servlet/servlet-
api -->
            <dependency>
                  <groupId>javax.servlet</groupId>
                  <artifactId>servlet-api</artifactId>
                  <version>2.5</version>
                  <scope>provided</scope>
            </dependency>
<!--
https://mvnrepository.com/artifact/org.apache.struts/struts2-
core -->
            <dependency>
                  <groupId>org.apache.struts</groupId>
                  <artifactId>struts2-core</artifactId>
                  <version>2.5.26</version>
            </dependency>
```

Step 5: Configure the Controller Servlet into 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="apache-struts-config-example" version="3.0">
      <display-name>Struts2 Example</display-name>

      <filter>
            <filter-name>struts2</filter-name>
            <filter-class>

      org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteF
ilter
            </filter-class>
      </filter>
      <filter-mapping>s
            <filter-name>struts2</filter-name>
            <url-pattern>/*</url-pattern>
      </filter-mapping>

</web-app>
```

Step 6: Create an Action class. UserAction class extends ActionSupport class in order to implement the validation with Validate() method.

```
public class UserAction extends ActionSupport
{
  private static final long serialVersionUID = 1L;
  private String firstName;
  private String lastName;
  private String message;
  @Override
  public String execute() throws Exception
  {
    int timeOfDay =
Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
    if(timeOfDay >= 0 && timeOfDay < 12)
    {
      message = "Good Morning";
    }
    else if(timeOfDay >= 12 && timeOfDay < 16)
    {
     message = "Good Afternoon";
    }
    else if(timeOfDay >= 16 && timeOfDay < 21)
    {
      message = "Good Evening";
    }
    else if(timeOfDay >= 21 && timeOfDay < 24)
    {
      message = "Good Morning";
    }
    return ActionSupport.SUCCESS;
  }
  @Override
  public void validate()
  {
    if (null == firstName || firstName.length() == 0)

addActionError(getText("error.firstName.required"));
    if (null == lastName || lastName.length() == 0)

addActionError(getText("error.lastName.required"));
  }

  // getter setter...
}
```

Step 7: Configure the request mapping and view in **struts.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <include file="struts-default.xml"/>

    <constant name="struts.enable.DynamicMethodInvocation"
value="false" />
    <constant name="struts.devMode" value="true" />
    <constant name="struts.custom.i18n.resources"
value="Application" />

    <package name="default" extends="struts-default">
        <action name="">
            <result>/WEB-INF/jsp/index.jsp</result>
        </action>
       <action name="user"
class="org.ignou.struts2.action.UserAction">
          <result name="error">/WEB-INF/jsp/index.jsp</result>
          <result name="input">/WEB-INF/jsp/index.jsp</result>
          <result name="success">/WEB-INF/jsp/success.jsp</result>
       </action>
    </package>

</struts>
```

Step 8: Add the required properties into Application.properties.

```
label.welcome = Struts 2 Hello World!!!
label.firstName =  First Name
label.lastName =  Last Name
error.firstName.required = First Name is required!
error.lastName.required = Last Name is required!

submit = Go!!
```

Step 9: Add the jsp file inside folder /WEB-INF/jsp.

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-
1" pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head>
     <title>Struts 2 Maven Hello world!!!</title>
     <s:head/>
 </head>

 <body>
        <h1 style="color: green;text-align: center;"><s:text
name="label.welcome" /></h1>
        <s:if test="hasActionErrors()">
            <div id="fieldErrors">
                <s:actionerror/>
            </div>
        </s:if>

        <s:form action="user" namespace="/" method="post"
name="strutsForm">
            <s:textfield name="firstName" size="30" maxlength="50"
key="label.firstName"/>
            <s:textfield name="lastName" size="30" maxlength="50"
key="label.lastName"/>
            <s:submit key="submit" align="right"/>
        </s:form>
    </body>

</html>
```

success.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title>Struts 2 Maven Welcome page!</title>
    </head>
    <body>
        <h2 style="color: green"><s:property
value="message"/>,</h2>
        <h3 style="color: green">&emsp; &emsp; Mr./Mrs.
<s:property value="lastName" /> <s:property value="firstName"
/></h3>
    </body>

</html>
```

Step 10: Run the application in eclipse by right clicking on project > **Run As** > **Run on Server.** Configure tomcat server into eclipse in order to execute the web application.

Step 11: Access the URL http://localhost:8080/struts2-hello-world/. Outputs of the execution are as followings.

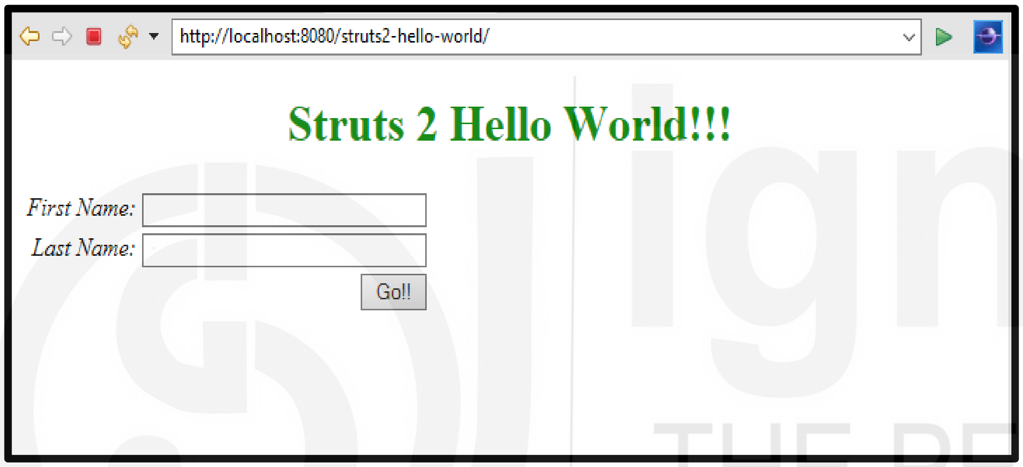
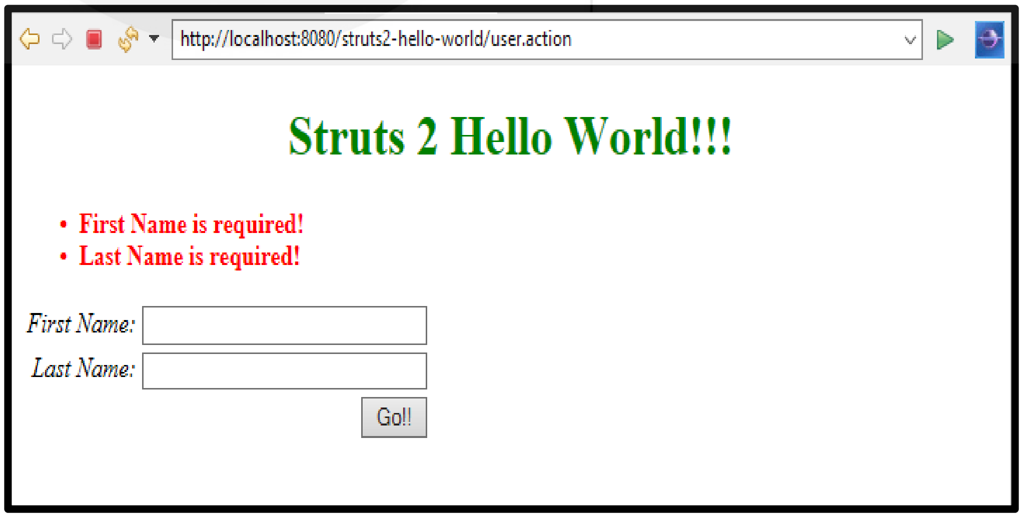


Figure6. 6: Execution Result 1

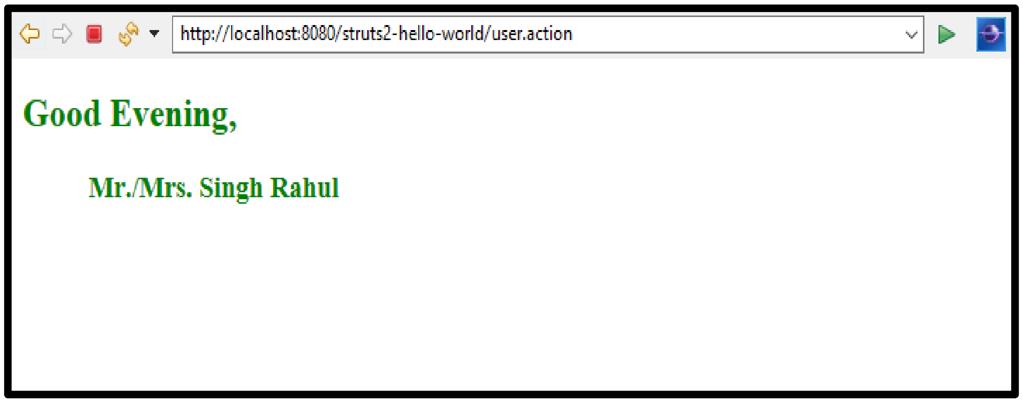
Validation has been implemented for both the fields. Hence, if any field validation fails, the user will be notified with error messages.



- First Name is required!
- Last Name is required!

Figure 6.7: Execution Result 2

Success response after valid form submission is as follows.



```
http://localhost:8080/struts2-hello-world/user.action
```

```
Good Evening,
    Mr./Mrs. Singh Rahul
```

Figure 6.8: Execution Result 3

☞ Check Your Progress 1:

1) Explain Model 1 architecture with its advantages and disadvantages.

----

2) Explain Model 2 architecture with its merits and demerits.

----

3) What are Actions in Struts2?

----

4) List down the features of Struts.

6.3 SPRING BOOT AND MVC: FEATURES

Spring, Spring MVC and Spring Boot do not compete for the same space; instead, they solve the different problems very well. This section briefs about Spring, Spring MVC and Spring Boot. Later it explains the features of Spring MVC and Spring Boot.

6.3.1 Spring MVC features

The **dependency injection (DI)** is the core concept of the **Spring framework,** which enables the development of loosely coupled and easily unit testable applications. Based on the DI, Spring framework created various modules such as ***Spring MVC, Spring JDBC, Spring Test, Spring ORM, Spring AOP*** etc. These modules do not bring any new functionality since we can do all these with J2EE. **These modules simply bring in abstraction** and the abstraction aims-

- Reduce Duplication/ Reduce Boilerplate Code
- Promote Decoupling/ Increase Unit testability

Spring Web MVC framework enables us to develop **decoupled web applications**. It introduces the concepts of Dispatcher Servlet, ModelAndView and View Resolvers to develop loosely coupled web applications. Spring MVC holds many unique features listed below.

- **Clear separation of responsibility:** The DispatcherServlet, controller, handler mapping, view resolver, form object, model object, command object, validation and so on are specialized to perform a single responsibility.
- **Adaptability and flexibility:** Spring MVC is very flexible and adaptable. One of the features to define the controller method signature as per need, possibly using one of the parameter annotations such as @PathVaribale, @RequestParam, @RequestHeader etc.
- **Reusable business code:** It allows us to reuse the existing business object as a form object or command object instead of creating a duplicate business object.
- **Flexible model transfer:** It supports model transfer as name/ value map, which can be easily integrated with any view technology.
- **Customizable handler mapping and view resolution:** Spring Web MVC supports various Handler mapping and View resolution strategies ranging from simple URL-based configuration to sophisticated, purpose-built resolution strategies.
- **Robust JSP form tag library:** Spring 2.0 introduced a powerful JSP form tag library that simplifies the form design in JSP.

6.3.2 Spring Boot features

Spring MVC requires lots of configurations such as dispatcher servlet, component scan, handler mapping, view resolver and many more. When an application uses Hibernate/JPA, it requires configurations such as data source, entity manager, transaction manager etc.

Spring Boot project aims at simplifying and make it easier to develop a Spring based web application. Spring Boot brings intelligence and provides auto configuration features. Spring Boot looks at the framework / jars available in the classpath and existing configuration of the application in order to provide the basic configuration needed to configure the application. Features of Spring Boot are as follows.

- The guiding principle of Spring Boot is **convention over configuration.**
- Spring Boot starter simplifies dependency management.
- It also supports the development of stand-alone Spring applications.
- Supports auto-configuration wherever possible.
- It supports production-ready features such as metrics, health checks, and externalized configuration.
- XML configuration is not required any more.
- It supports embedded servers such as Jetty, Tomcat or Undertow.

6.4 HIBERNATE WITH JPA: FEATURES

Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages. ORM makes it possible to perform ***CRUD*** operations without considering how those objects relate to their data source. It manages the mapping details between a set of objects and the underlying database. It hides and encapsulates the changes in the data source itself. Thus, when data sources or their APIs change, only ORM change is required rather than the application that uses ORM.

Hibernate is a pure Java object-relational mapping (ORM) and persistence framework which maps the POJOs to relational database tables. The usage of Hibernate as a persistence framework enables the developers to concentrate more on the business logic instead of making efforts on SQLs and writing boilerplate code. Hibernate is having many features as listed below –

- Open Source
- High Performance
- Light Weight
- Database independent
- Caching
- Scalability
- Lazy loading
- Hibernate Query Language (HQL)

Java Persistence API (JPA) is a specification that defines APIs for object relational mappings and management of persistent objects. JPA is a set of interfaces that can be used to implement the persistence layer. JPA itself doesn't provide any implementation classes. In order to use the JPA, a provider is required which implements the specifications. Hibernate and EclipseLink are popular JPA providers.

Spring Data JPA is one of the many sub-projects of Spring Data that simplifies the data access for relational data stores. Spring Data JPA is not a JPA provider; instead it wraps the JPA provider and adds its own features like a no-code implementation of the repository pattern. Spring Data JPA uses Hibernate as the default JPA provider. JPA provider is configurable, and other providers can also be used with Spring Data JPA. Spring Data JPA provides a complete abstraction over the DAO layer into the project.

6.5 COMPARISON AMONG THESE FRAMEWORKS

Spring, Struts and Hibernate are parts of MVC architecture. **These frameworks serve different purposes and can exist independently or together**. It depends on the project requirement to choose the combination of frameworks. This section gives an overview of comparison among the various frameworks such as Spring Vs Struts, Spring Boot Vs Spring MVC and Spring Vs Hibernate. In the subsequent units Spring, Spring Boot and Hibernate have been explained in detail.

6.5.1 Struts Vs Spring

Struts	Spring
An open source framework which enables to extend Java Servlet API and MVC framework.	An open source framework to implement inversion of Control (IOC) and dependency Injection (DI).
It is a Heavyweight framework.	It is a Lightweight framework.
It is less flexible than Spring.	It is more flexible than Struts.
It has non layered architecture.	It has layered architecture.
It is tightly coupled.	It is loosely coupled.
It also provides integration with ORM and JDBC but manual coding is required.	It provides an easy integration with ORM and JDBC technologies.

6.5.2 Spring Boot Vs Spring MVC

Spring Boot	Spring MVC
Spring Boot removes all boilerplate code and supports the auto configuration based on jars available into classpath.	Spring MVC requires a lot of configuration and it contains a lot of boilerplate code.
Spring Boot provides many spring boot starters. Spring boot starter is a unit of related dependencies wrapped together. It solves the problem of dependency management.	Dependency management is a tough process since every dependency with the correct version needs to be declared.
Spring Boot makes the application development easier and faster.	Spring MVC requires a lot of configurations and dependency management is time consuming. Hence, development is slow.
Spring Boot supports embedded servers to be packaged along with applications to run the jar stand alone.	A lot of manual configuration is required to attain the same level of packaging.
Spring Boot supports many production ready features such as Actuators, Process monitoring out-of-box.	Spring MVC does not support any production ready feature.
Spring Boot is very flexible and provides many modules which can be used to build many different other applications.	Spring MVC is designed only for the development of dynamic web pages and RESTful web services.

6.5.3 Spring Vs Hibernate

Spring	**Hibernate**
Spring is an open source, complete and modular application framework to develop enterprise applications in java.	Hibernate is an ORM framework specialized in data persisting and data retrieval from DB.
Spring provides many useful features such as transaction management, aspect-oriented programming and dependency injection (DI).	Hibernate provides Object-Relational Persistence and Query service for applications.
Spring framework has many modules such as Spring-Core, Spring-MVC, Spring-Rest, Spring-Security and many more.	Hibernate does not have modules like Spring framework.
Spring framework has support for connection pooling by changing the configuration in the spring configuration file.	Hibernate supports robust connection pooling features. Hibernate supports two levels of cache which improves the application performance.

☞ **Check Your Progress 2:**

1) What are the advantages of Spring Web MVC?

----

2) Discuss the features of Spring Boot.

----

3) Explain Hibernate with its advantages.

----

4) Compare the Struts and Spring frameworks.

....

....

....

....

5) Compare Spring Boot and Spring MVC.

....

....

....

....

6.6 MAVEN: INTRODUCTION, OVERVIEW AND CONFIGURATION

Building the process of a software project may involve a series of tasks among downloading required dependencies, putting jars on classpath, generating source code (for auto generated code), generating documentation from source code, source code compilation, tests execution, packaging compiled code along with the dependencies into deployable artifacts such as WAR, EAR or JAR and deploying the generated artifacts to the application server or repository.

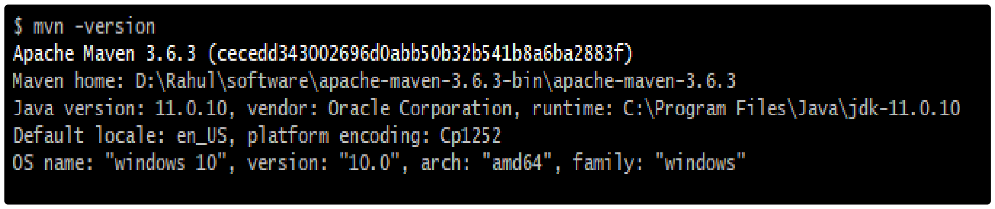
Maven is a simple and powerful build automation and management tool used primarily for Java Projects. It can also be used to build and manage other language projects such as C#, Scala, Ruby, etc. A manual build process is error prone and time consuming. Maven minimizes the risk of humans making errors and speeds up the build process of a software project.

6.6.1 Maven Installation

JDK installation is the prerequisite to execute maven into the system. Visit http://maven.apache.org/download.cgi to download the maven. Extract the downloaded file and perform the following configurations in order to execute maven.

- Set JAVA_HOME environment variable pointing to valid JDK installation path.
- Set M2_HOME environment variable pointing to the directory of extracted maven.
- Add the executable maven path into PATH variable as **%M2_HOME%\bin** on Windows and **%M2_HOME%/bin** on Unix.

Open a command prompt and execute the command ***mvn –version*** to check the maven installation and configuration. You will get the output similar to below.



```
$ mvn -version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: D:\Rahul\software\apache-maven-3.6.3-bin\apache-maven-3.6.3
Java version: 11.0.10, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-11.0.10
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Figure 6.9: Maven Version Verification

6.6.2 Maven Core Concepts

POM file (Project Object Model) is the focal point of a maven project. A maven project is configured using a POM file known as pom.xml. A POM file is an XML file which describes the project and contains all the references of project resources like source code, test code, dependencies etc. Maven execution and the main content of the POM file has been illustrated below.

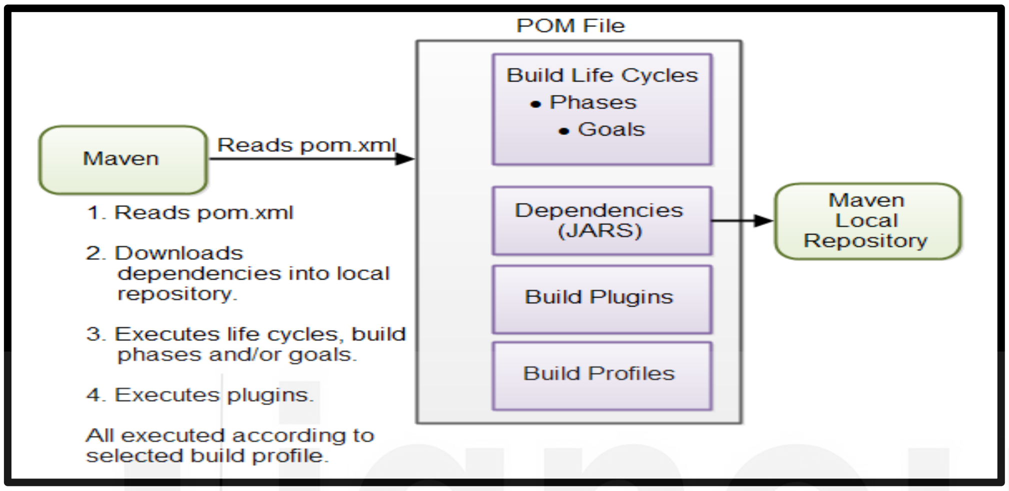


Figure 6.10: Maven Execution Flow

6.6.2.1 Basic Structure of the POM File

Basic structure of a typical POM file contains project identifiers, dependencies, properties, build etc. POM file also supports the concept of profile. Profiles are used to customize the build configuration for different environments such as dev, test and prod. A sample POM file example is shown below.

```
<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>org.test</groupId>
      <artifactId>DemoWebApp</artifactId>
      <version>1.0-SNAPSHOT</version>
       <packaging>war</packaging>
      <name>DemoWebApp Maven Webapp</name>
      <url>http://maven.apache.org</url>
      <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
       </properties>
      <dependencies>
            <dependency>
                  <groupId>javax.servlet</groupId>
                  <artifactId>servlet-api</artifactId>
                  <version>2.5</version>
                  <scope>provided</scope>
            </dependency>
            <dependency>
```

```
				<groupId>junit</groupId>
				<artifactId>junit</artifactId>
				<version>3.8.1</version>
				<scope>test</scope>
			</dependency>
		</dependencies>
		<build>
			<finalName>Demo WebApp</finalName>
		</build>
</project>
```

- **Project Identifiers:** Maven combines groupId:artifactId:version (GAV) to form the unique identifier to identify a project uniquely.
  - groupId – a unique base name of the group or company of the project creator
  - artifactId – a unique name for the project
  - version – a version of the project
  - packaging – a packaging format of the project such as JAR, WAR, EAR, ZIP. If the packaging type is ***pom***, Maven does not create anything for this project since it is just meta-data.
- **Dependencies:** A mid-scale or large scale project uses external Java APIs or frameworks which are packaged in their own JAR files. These Jar files for Java APIs or frameworks are known as dependencies. A dependency JAR may again depend on other dependencies. Keeping projects up-to-date with the correct version of external dependencies a comprehensive task. Downloading all these external dependencies (JAR files) recursively and making sure that the right versions are downloaded is cumbersome.

  Maven has a built-in powerful dependency management feature. To make the project dependencies available into classpath, we just need to specify all the required dependencies with GAV (groupId, artifactId, version) as shown in sample pom.xml. Maven downloads all the dependencies recursively and puts them into the local **maven repository**.
- **Maven Repositories:** Maven repositories are directories of packaged JAR files with extra Meta data. There are three types of repository in Maven.
  - **Local Repository –** As the name indicates, it's a repository located on the developer's machine itself. Maven downloads the dependencies from Central repository, Remote repositories and stores it into the Local Repository. By default, the Local Repository location is **user-home/.m2.**
  - **Central Repository –** The Central Repository is maintained and provided by the maven community. The Central Repository access does not require any specific configuration. It can be accessed at ***https://mvnrepository.com/*** and maven dependencies can be searched.

- **Remote Repository – A** Remote Repository is like the Central Repository from where maven can download the dependencies and store it into Local Repository. It may be located on any web server on the internet or the local network**.** It can be configured into pom.xml by putting the following contents just after **<dependencies>** element**.**

```
<repositories>
            <repository>
               <id>test.code</id>
               <url>http://myremote.maven.com/maven2/lib</url>
            </repository>
</repositories>
```

- **Properties:** Custom properties make the pom file readable and maintainable. Example to use custom properties is shown below.

```
<properties>
   <spring.version>4.3.5.RELEASE</spring.version>
</properties>
<dependencies>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
   </dependency>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
   </dependency>
</dependencies>
```

If we want to upgrade the spring to a newer version, we just need to change the version at one place only for custom property <spring.version> to update all required dependencies versions for spring.

- **Build:** The build section provides information about the default maven **goal,** the final name of the artifact and the directory for the compiled project. The default build section is like shown below.

```
<build>
     <defaultGoal>install</defaultGoal>
     <finalName>${artifactId}-${version}</finalName>
     <directory>${basedir}/target</directory>
    //...
</build>
```

  - Default goal is install
  - Final name of the artifact contains artifactId and version. It can be modified at any time into the build section.
  - The default output folder for the generated artifact is target folder.

6.6.2.2 Maven Build Life Cycles, Phases and Goals

Maven follows a **build life cycle** while building an application. The build life cycle consists of phases and phase consists of build goals.

- **Build Life Cycle:** Maven has 3 built-in build life cycles which are responsible for different aspects of building a software project. These build life cycles execute independent of one another. For two different build life cycles, you have to use two different maven commands and these build life cycles will execute sequentially. The build life cycles are as followings-
  - **default -** The *default* build lifecycle is of most interest since this is what builds the code. It is responsible to handle everything related to compiling and packaging the software project. ***This build life cycle can't be executed directly***. You need to execute a build **phase** or **goal** from the default build life cycle.
  - **clean -** The *clean* build life cycle performs the tasks related to cleaning such as deleting compiled classes, removing previous jar files, deleting the generated source codes, removing temporary files from the output directory etc. The command **mvn clean** cleans the project.
  - **site -** The *site* build life cycle performs the task related to generating documents for the project.
- **Build Phases:** A Maven phase represents a stage in the Maven build life cycle. Each phase performs a specific task. The most commonly used build phases defined for default build life cycle are as follows.

Build Phase	Description
validate	Verifies the correctness of the project and make sure that all required dependencies are downloaded.
Compile	Compiles the source code to produce the binary artifacts
Test	Executes the unit test cases using a suitable unit testing framework without packaging the binary artifacts
Package	Packs the binary artifacts into distributable format such as JAR,WAR,EAR
intergration-test	Executes integration test cases which require packaging.
Verify	Verifies the correctness of package
Install	Installs the package into local maven repository
Deploy	Copies the final package to the remote repository for sharing with other developers and projects.

Maven allows us to execute either a whole build life cycle such as ***clean or site***, a build phase like ***deploy*** which is a phase of default build life cycle, or a build goal like ***sunfire:test***.

*Note: The execution of a phase using command **mvn <phase>** does not execute only the specified phase. Instead it executes all the preceding phases as well. The execution of **mvn install** will execute all other phases in the above table.*

- **Build Goals:** The finest step in the maven build process is a Goal. A goal can be bound to zero or more build phases. A phase helps to determine the order in which the *goals* are executed.
  - Command **mvn <goal>** executes a goal which is not bound to any phase.
  - Command **mvn <phase>:<goal>** executes a goal which is bound to a phase.

6.6.2.3 Maven Build Profiles

Different build configurations are required for different environments such as production, test, dev. The concepts of Maven Build Profiles enable us to keep the multiple environments build configuration into a single pom.xml file. A sample example to create the configuration based on profiles is shown below.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

<groupId>com.test</groupId>
<artifactId>maven-demo</artifactId>
<version>1.0.0</version>
<profiles>
    <profile>
        <id>production</id>
        <build>
            <plugins>
                <plugin>
                //...
                </plugin>
            </plugins>
        </build>
    </profile>
    <profile>
        <id>development</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <build>
            <plugins>
                <plugin>
                //...
                </plugin>
            </plugins>
        </build>
     </profile>
 </profiles>
</project>
```

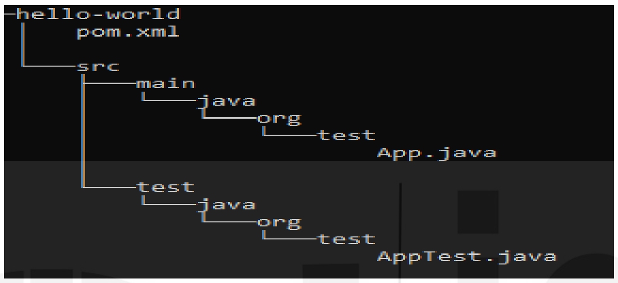
In the above configuration development profile is set as default. If we want to execute production profile build, it can be executed as **mvn clean install –Pproduction**. The flag –P is used to provide the profile to be used for the build process.

6.7 CREATE FIRST PROJECT USING MAVEN

This section describes how to create a simple Hello World java application using Maven. The Maven project will be created using the command line, and pom.xml will be modified to execute the jar file using maven plug-in. Perform the following steps and execute the application using maven command.

Step 1: Create a simple Hello World Java project using the below command. The command will generate the directory structure and pom.xml file. The generated directory structure is shown in the screenshot.

mvn archetype:generate -DgroupId=org.test -DartifactId=hello-world -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false



```
-hello-world
│     pom.xml
│
└─────src
        ├──────main
        │      └─────java
        │               └─────org
        │                       └─────test
        │                               App.java
        │
        └──────test
               └─────java
                        └─────org
                                └─────test
                                        AppTest.java
```

Figure 6.11: Maven Project Folder Structure

Step 2: Simple Hello World application is ready to be compiled and packaged. Update the pom.xml as below.

```
<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>org.test</groupId>
  <artifactId>hello-world</artifactId>
  <name>hello-world</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
    </dependency>
  </dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.6.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

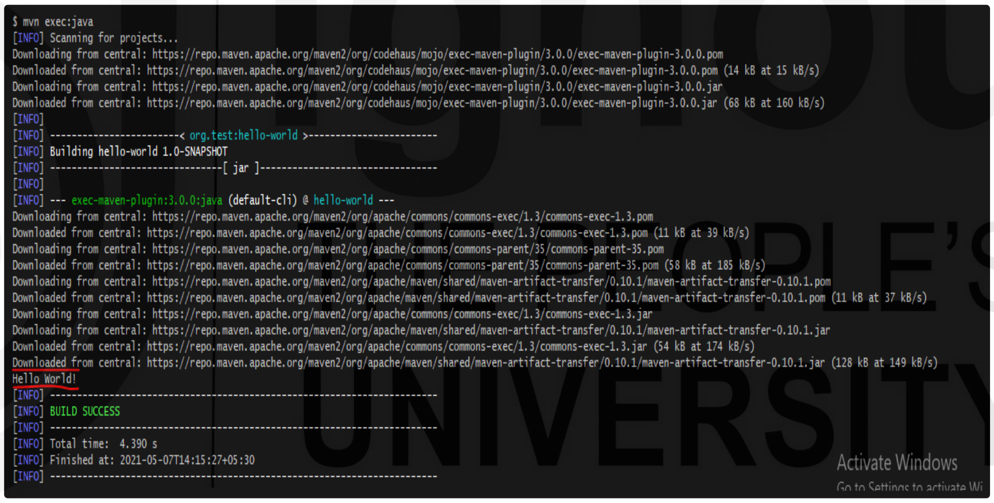
Execute the following commands and check the output on the console.

- mvn compile : Maven will execute all the required phases for compile phase and build the binary artifact.
- mvn test: Only test phase can be executed with this command.
- mvn package : It packages the binary artifact with the specified package type such as jar, war, ear etc. Default packaging type is jar and the generated jar file is located into the target folder.

Step 3: Add the following plug-in into the build section after compiling the plug-in. The plugin requires the main class into configuration. Update the main class. This added plug-in will enable the jar file execution from maven command.

```
<plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>3.0.0</version>
        <configuration>
           <mainClass> org.test.App</mainClass>
        </configuration>
  </plugin>
```

The command **mvn exec:java** will execute the jar file. The output for the command execution is shown below:



```
$ mvn exec:java
[INFO] Scanning for projects...
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/exec-maven-plugin/3.0.0/exec-maven-plugin-3.0.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/exec-maven-plugin/3.0.0/exec-maven-plugin-3.0.0.pom (14 kB at 15 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/exec-maven-plugin/3.0.0/exec-maven-plugin-3.0.0.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/exec-maven-plugin/3.0.0/exec-maven-plugin-3.0.0.jar (68 kB at 160 kB/s)
[INFO]
[INFO] -----------------------< org.test:hello-world >------------------------
[INFO] Building hello-world 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ hello-world ---
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.3/commons-exec-1.3.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.3/commons-exec-1.3.pom (11 kB at 39 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-parent/35/commons-parent-35.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-parent/35/commons-parent-35.pom (58 kB at 185 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-artifact-transfer/0.10.1/maven-artifact-transfer-0.10.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-artifact-transfer/0.10.1/maven-artifact-transfer-0.10.1.pom (11 kB at 37 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.3/commons-exec-1.3.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-artifact-transfer/0.10.1/maven-artifact-transfer-0.10.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.3/commons-exec-1.3.jar (54 kB at 174 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-artifact-transfer/0.10.1/maven-artifact-transfer-0.10.1.jar (128 kB at 149 kB/s)
Hello World!
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.390 s
[INFO] Finished at: 2021-05-07T14:15:27+05:30
[INFO] ------------------------------------------------------------------------
```

Figure 6.12: Maven Execution Result

☞ Check Your Progress 3:

1) What is Maven? Discuss its usage.

----

2) Explain about Maven repositories.

....

....

3) Explain the different build life cycles available in the Maven tool.

....

....

....

....

4) Write down the command to execute a build goal into Maven.

....

....

....

....

6.8 SUMMARY

The unit has explained the Model 1 and Model 2 architecture of a web application with its advantages and disadvantages. Model 2 fixes the concerns into Model 1 architecture and Struts fixes the issues into Model 2 architecture and provides Pull-MVC (MVC2) based framework to develop enterprise web applications. The features of different frameworks such as Struts, Spring Boot, Spring MVC and Hibernate have been explained and these frameworks have been compared. The last section of this unit has explained Maven as a simple and powerful build and deployment management tool. The highlights of this unit are as follows.

- The Model-View-Controller (MVC) architecture provides the separation of concerns and makes the application easy to maintain and easy to develop.
- The Struts2 is enriched with many important features such as support to POJO based actions, Configurable MVC Components, Integration support to various frameworks such as Spring, Tiles, Hibernate etc, AJAX support, Validation support, support to various themes and Template such as JSP, Velocity, Freemarker etc.
- In model 1 architecture, JSP pages were the focal point for the entire web application.
- In Model 2 architecture contrast to the Model 1 architecture, a Servlet known as ***Controller Servlet*** intercepts all client requests. The Controller Servlet performs all the initial request processing and determines which JSP to display next.
- Spring, Spring MVC and Spring Boot do not compete for the same space instead, they solve the different problems very well.
- Spring Web MVC framework enables us to develop **decoupled web applications**. It introduces the concepts of Dispatcher Servlet, ModelAndView and View Resolvers to develop loosely coupled web applications.
- **Spring Boot** project aims at simplifying and making it easier to develop a Spring based web application. Spring Boot brings intelligence and provides

auto-configuration features. Spring Boot looks at the framework / jars available in the classpath and existing configuration of the application in order to provide the basic configuration needed to configure the application.
- Maven is a simple and powerful build automation and management tool used primarily for Java Projects. Maven minimizes the risk of humans making errors and speeds up the build process of a software project.
- The concepts of Maven Build Profiles enable us to keep the multiple environments build configuration into a single pom.xml file.

6.9 SOLUTIONS/ ANSWER TO CHECK YOUR PROGRESS

Check Your Progress 1

1) In model 1 architecture, JSP pages are the focal point for entire web applications. JSP provides the solutions to the problems of Servlet technology. JSP provides better separation of concern not to mix business logic and presentation logic. Re-Deployment of web applications is not required if the JSP page is modified. JSP supports JavaBean, custom tags and JSTL to keep the Business logic separate from JSP. **Refer section 6.2.1.1 for architecture diagram and advantages and disadvantages.**
2) In Model 2 architecture contrast to the Model 1 architecture, a Servlet known as ***Controller Servlet*** intercepts all client requests. The Controller Servlet performs all the initial request processing and determines which JSP to display next. **Refer section 6.2.1.2 for architecture diagram and advantages and disadvantages.**
3) Actions are the core of Struts2 framework. These are used to provide the business logic which executes to serve the client request. Each URL is mapped to specific action. In Struts2, the action class is simply POJO (Plane Old Java Object). The only prerequisites for a POJO to be an action is that there must be a no-argument method that returns either a String or Result Object. If the no-argument method is not specified, the execute() method is used to perform business logic processing.
Refer section 6.2.3

Check Your Progress 2

1) Spring Web MVC framework enables us to develop **decoupled web applications**. It introduces the concepts of Dispatcher Servlet, ModelAndView and View Resolvers to develop loosely coupled web applications. Spring MVC holds many unique features listed below.
   - Clear separation of responsibility
   - Adaptability and flexibility
   - Reusable business code
   - Flexible model transfer
   - Customizable handler mapping and view resolution
   - Robust JSP form tag library
2) **Spring Boot** project aims at simplifying and making it easier to develop a Spring based web application. Spring Boot brings intelligence and provides auto configuration features. Spring Boot looks at the framework / jars available into the classpath and existing configuration of the application in order to provide the basic configuration needed to configure the application. Features of Spring Boot are as follows.
   - The guiding principle of Spring Boot is **convention over configuration.**
   - Spring Boot starter simplifies dependency management.
   - It also supports the development of stand-alone Spring applications.

- o Supports auto-configuration wherever possible.
- o It supports production-ready features such as metrics, health checks, and externalized configuration.
- o XML configuration is not required anymore.
- o It supports embedded servers such as Jetty, Tomcat or Undertow.

3) Hibernate is a pure Java object-relational mapping (ORM) and persistence framework which maps the POJOs to relational database tables. The usage of Hibernate as a persistence framework enables the developers to concentrate more on the business logic instead of making efforts on SQLs and writing boilerplate code. Hibernate is having many features as listed below –

- o Open Source
- o High Performance
- o Light Weight
- o Database independent
- o Caching
- o Scalability
- o Lazy loading
- o Hibernate Query Language (HQL)

4) Refer the section 6.5.1
5) Refer the section 6.5.2

Check Your Progress 3

1) Maven is a simple and powerful build automation and management tool used primarily for Java Projects. It can also be used to build and manage other language projects such as C#, Scala, Ruby, etc. Manual build process is error prone and time consuming. Maven minimizes the risk of human making errors and speeds up the build process of a software project. Maven is used to perform many tasks like:
   - o We can easily build a project using maven.
   - o We can add jars and other dependencies of the project easily using the help of maven.
   - o Maven provides project information (log document, dependency list, unit test reports etc.)
   - o Maven is very helpful for a project while updating the central repository of JARs and other dependencies.
   - o With the help of Maven, we can build any number of projects into output types like the JAR, WAR etc, without doing any scripting.
   - o Using maven, we can easily integrate our project with source control systems (such as Subversion or Git).

2) Maven repositories are directories of packaged JAR files with extra Meta data. There are three types of repositories in Maven.
   - o Local Repository
   - o Central Repository
   - o Remote Repository

   Refer the section 6.6.2.1

3) Refer Build life cycle in **section 6.6.2.2**

4) The finest step in the maven build process is a Goal. A goal can be bound to zero or more build phases. A phase helps to determine the order in which the goals are executed.
   - o Command mvn <goal> executes a goal which is not bound to any phase.
   - o Command mvn <phase>:<goal> executes a goal which is bound to a phase.

6.10 REFERENCES/ FURTHER READING

- Craig Walls, “Spring Boot in action” Manning Publications, 2016. (https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf)
- Christian Bauer, Gavin King, and Gary Gregory, “Java Persistence with Hibernate”, Manning Publications, 2015.
- Ethan Marcotte, “Responsive Web Design”, Jeffrey Zeldman Publication, 2011(http://nadin.miem.edu.ru/images_2015/responsive-web-design-2nd-edition.pdf)
- Tomcy John, “Hands-On Spring Security 5 for Reactive Applications”, Packt Publishing,2018

- https://mkyong.com/tutorials/struts-2-tutorials/
- https://www.tutorialspoint.com/struts_2/struts_overview.htm
- https://www.javatpoint.com/struts-2-tutorial
- https://www.codejava.net/frameworks/struts/introduction-to-struts-2-framework#vsStruts1
- https://stackoverflow.com/questions/17441926/push-vs-pull-model-in-mvc
- https://www.baeldung.com/maven
- http://tutorials.jenkov.com/maven/maven-tutorial.html#maven-pom-files
- https://howtodoinjava.com/maven/maven-dependency-management/
- https://www.quora.com/Whats-the-difference-between-J2EE-Struts-with-Hibernate-and-J2EE-Hibernate-with-Spring-framework

UNIT 7 SPRING MVC CONCEPTS

Structure **Page no.**

7.0 INTODUCTION

The Spring MVC framework is an open source Java platform. This framework was developed by Rod Johnson and it was first released in June 2003 under the Apache 2.0 license. In web based applications, the size of code plays a vital role.This framework is lightweight when it comes to size. This leads this framework to reach its heights. The basic version of this framework is of around 2MB. It follows the MVC (Model-View-Controller) design pattern. Spring Framework gears all the basic features of a core spring framework like Inversion of Control, and Dependency Injection.

MVC is a software structural design - the building of the system - that separates the business logic (domain/application/business) (whatever the way business prefers) from the rest of the user interface. The MVC does it by separating the whole application into three parts: the model, the view, and the controller.

The Spring MVC Dispatcher servlet acts as glue and this act makes it the most important component. It finds the correct methods and views for received incoming URL. This can be considered as a middleware as it receives the URL via HTTP request and it communicates with two ends – the sender of HTTP request and the Spring application. The Dispatcher servlet allows the use of all the features of Spring and it is completely integrated in the IoC container.

A the Spring Framework is modular in nature, it can be used in parts instead of using the whole of it. Java & Web applications can be built by using Spring Framework.

7.1 OBJECTIVES

After study this unit you should be able to:

- explain Spring MVC Framework,
- setup Spring MVC Development Environment,
- develop simple projects using Spring MVC,
- create control and views using MVC,and
- perform forms validation.

7.2 SETTING UP DEVELOPMENT ENVIRONMENT FOR SPRING MVC

Spring Framework is used widely in the market and there are many Solution that have been developed by keeping Spring Framework as core architecture. It is widely used in the market as well and SAP Hybris is one such example.

Spring Framework is used to build almost anything, it's make coding in Java much easy as it takes care of most of boilerplate (Boilerplate term refers to standardized text, methods, or procedures that can be used again without making any major changes to the original. It is mainly used for efficiency and to increase standardization in the structure and language of written or digital documents) code and let developer work on business logic.
Spring can be used to build any kind of applications (web application, enterprise application, REST APIs and distributed systems) with spring cloud and Angular framework which makes it more useful/popular.

As we discussed above Spring Framework is modular in nature which suggests that Spring Framework it's not a single entity. It has been divided into several modules which serve their own functionality:

- Spring Framework allows application to integrate with various other backend technologies.
- Spring Framework allows the development of standalone applications.
- Spring Framework becomes very useful while using distributed architecture.
- Due to lightweight in nature it reduces the complexity while using EJBs.
- Spring Security make the applications more secure.
- Spring Batch competently manage long running tasks.
- IoC and DI adds up to the flexibility and modularity of the application.

Spring offers a one-stop shop for all the business needs to develop applications. Due to its nature of modularity, it allows to pick and choose the modules which are more suitable to the business and technical demand without resorting to the whole stack. The following diagram provides details about all the modules available in Spring Framework.
As all the below modules are the part of Spring Framework libraries hence we need to download all these libraries and need setup to take all the below modules' advantage.

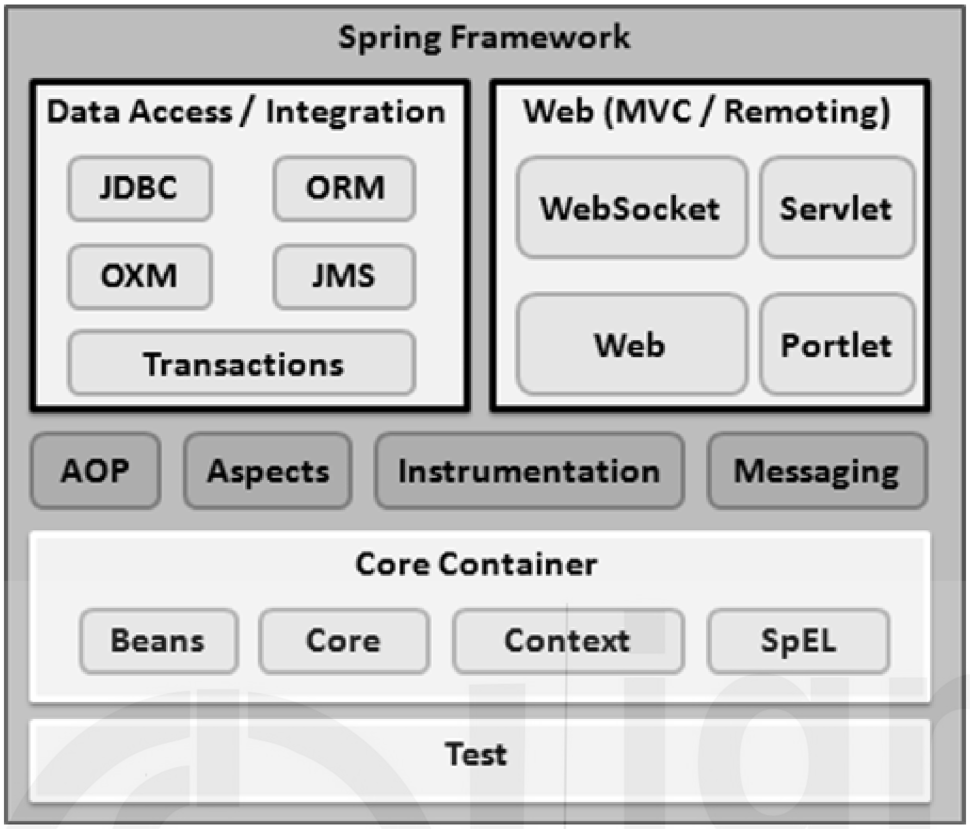


Figure 7.1: Spring Framework

Core Container: The Core Container contains the Core, Beans, Context, and Expression Language modules, the details of which are as follows −

- IoC and DI features are the main features of the Framework which comes under the core container as it provides the fundamental parts of the framework.
- BeanFactory comes under Bean module, it helps in the implementation of the factory pattern.
- Core and Beans module provides the strong base to build the Context module and it becomes the middleman to access any defined and configured object. The Context module works as a focal point of Application Context.
- A powerful expression language for querying and manipulation an object graph at runtime is provided by SpEL module,

The Data Access/Integration:This layer involves the JMS, OXM, ORM, & JDBC and Transaction modules and below are the details −

- It eliminates the need of religious JDBC related coding efforts of the developer as this JDBC module provides a JDBC-abstraction layer.
- The ORM Module takes care integration layer for some popular object-relational mapping APIs like iBatis, Hibernate, JDO and JPA.
- Object/XML mapping implementations for some APIs like XStream, JiBX, XMLBeans, Castor and JAXB are supported by the OXM Module.
- The Java Messaging Service JMS module holds features for producing and consuming messages.

... The Programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs are done by the Transaction Module.

Web: The Web layer holds of the following modules (Web, Web-MVC, Web-Socket, and Web-Portlet) and the details of these are as follows:

... Basic featurs for multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context provided by the Web Module.

... Spring's Model-View-Controller (MVC) implementation for web applications is covered under Web-MVC.

... The Web-Socket module takes responsibility of communication (support for WebSocket-based, two-way communication between the client and the server in web applications).

... The Web-Portlet module covers the MVC implementation to be used in a portlet setup and emulates the functionality of Web-Servlet module.

Miscellaneous:AOP, Aspects, Instrumentation, Web and Test modules are also some other important modules and the details of these are as follows −

... To decouple code, there is a need to define method-interceptors and pointcuts and this is achieved by AOP Module as it provides an aspect-oriented programming implementation.

... AspectJ integration is achieved by the Aspects Module and AspectJ is mature and powerful AOP Framework.

... This module is basically based on application servers as class loader and class instrumentation is used and supported by The Instrumentation Module.

... Support of Streaming Text Oriented Messaging Protocol (STOMP) as the WebSocket sub-protocol to use in applications is provided by the Messaging Module.

... JUnit or TestNG frameworks are used for Spring Components Testing and all this is done by Test Module

Java IDE Setup

It is always good to use integrated development environment (IDE) for any kind of java development as it increases the efficiency of the developer. An IDE is used for programming in Java or any other language provided by any IDE. IDE basically provides the functionality like debugging of the code, compilation of the code, interpreter for java like languages with code editor. Developer perform all the above mentioned activities with the help of graphical interface (GUI).

We would like to suggest you to use latest version of Eclipse or Netbean for your programming exercises.

Suppose you install Eclipse or other IDE on your syste. For example once IDE downloaded the installation, unpack the binary distribution into a convenient location. For example, in C:\eclipse. Eclipse can be started by double-click on eclipse.exe

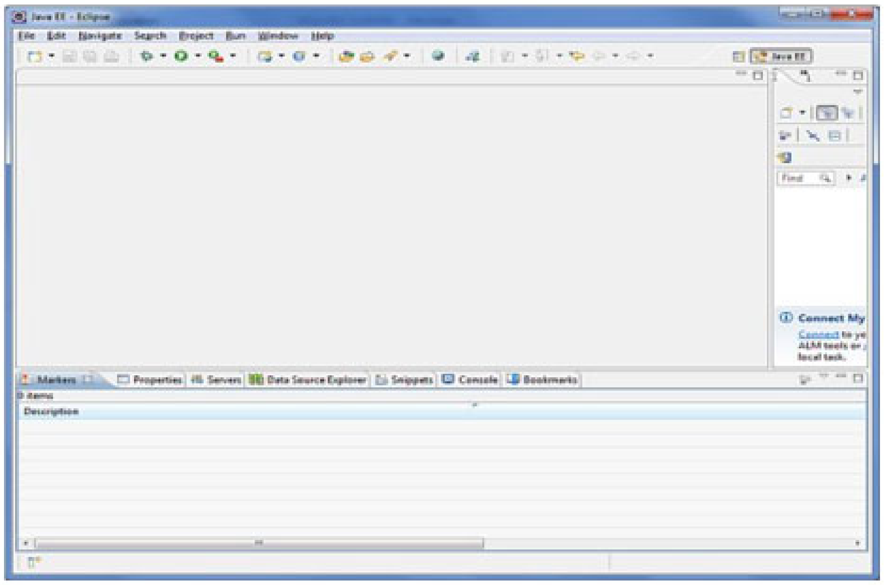
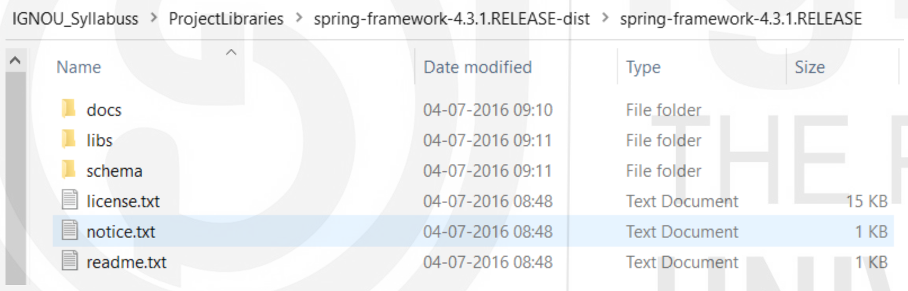


Figure 7.2: Eclipse IDE Setup

Setup Spring Framework Libraries

Once IDE setup is done then download the latest version of Spring framework binaries from https://repo.spring.io/release/org/springframework/spring.



IGNOU_Syllabuss › ProjectLibraries › spring-framework-4.3.1.RELEASE-dist › spring-framework-4.3.1.RELEASE

Name	Date modified	Type	Size
docs	04-07-2016 09:10	File folder	
libs	04-07-2016 09:11	File folder	
schema	04-07-2016 09:11	File folder	
license.txt	04-07-2016 08:48	Text Document	15 KB
notice.txt	04-07-2016 08:48	Text Document	1 KB
readme.txt	04-07-2016 08:48	Text Document	1 KB

Figure7.3: Downloading Spring IDE

7.3 FIRST HELLO WORLD PROJECT USING SPRING MVC

We will see in the example given below how to write a simple Hello World application in Spring MVC Framework. First thing to notice is to have Eclipse or some other IDE in place and follow the steps to develop a Dynamic Web Application. Here Eclip IDE is used for explanations.

Step 1:

Create a Dynamic Web Project with a name HelloIGNOU and create a package com.ignou the src folder in the created project.

Step 2:

Drag and drop below mentioned Spring and other libraries into the folder WebContent/WEB-INF/lib.

Step 3:

Create a Java class HelloController under the com.ignou.springmvc package.

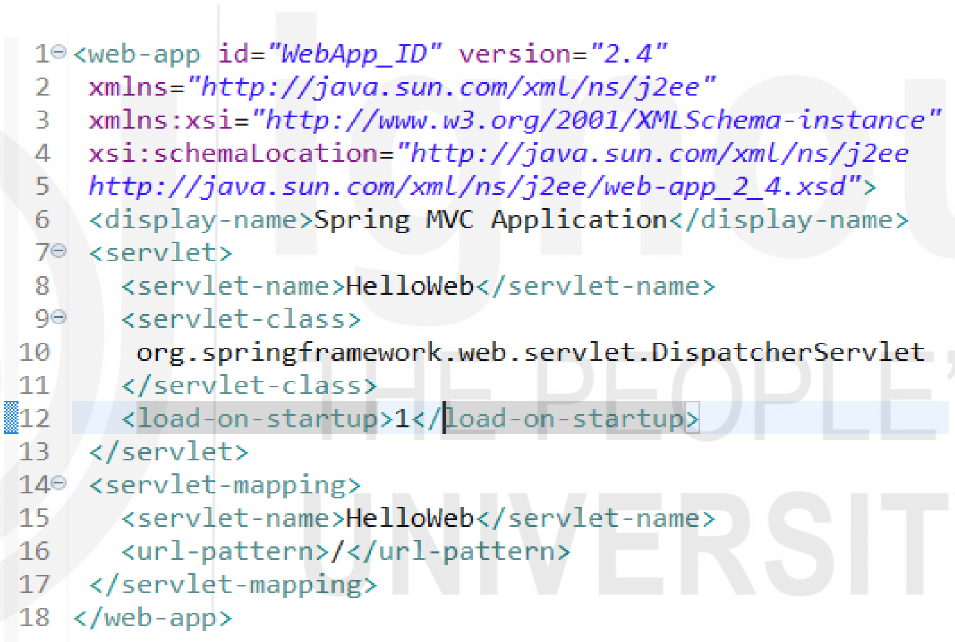
Step 4:

Create Spring configuration files web.xml and HelloWeb -servlet.xml under the WebContent/WEB-INF folder.

Step 5:
Create a sub-folder with a name jsp under the WebContent/WEB-INF folder. Create a view file hello.jsp under this sub-folder.

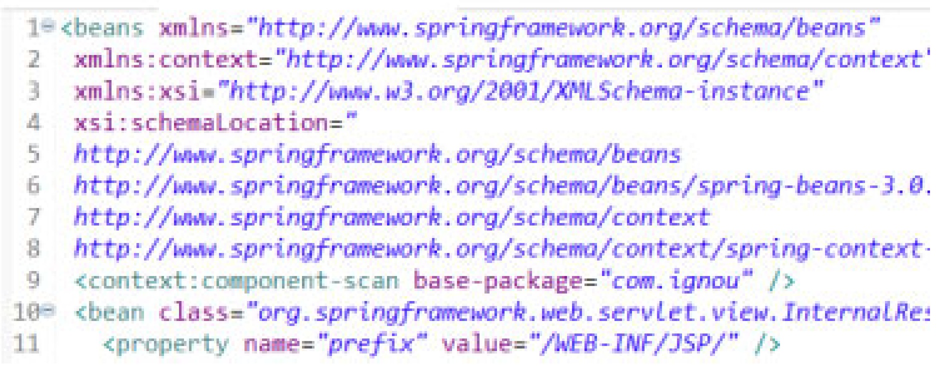
Step 6:
The final step is to create the content of all the source and configuration files. Once it is done then you have to export the application as explained below.

Web.xml



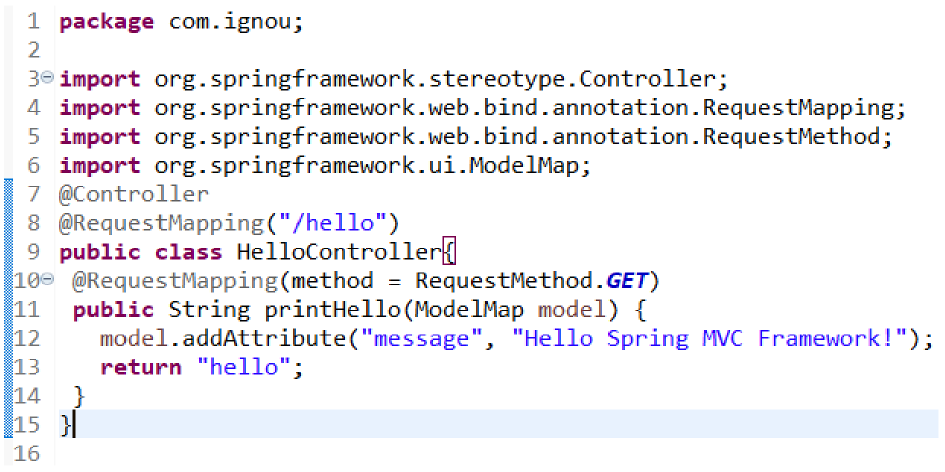
```
<web-app id="WebApp_ID" version="2.4"
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <display-name>Spring MVC Application</display-name>
 <servlet>
   <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>/</url-pattern>
 </servlet-mapping>
</web-app>
```

HelloWeb -servlet.xml



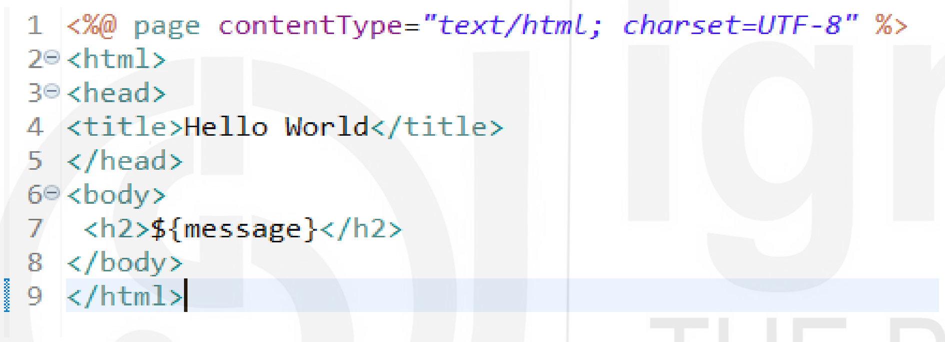
```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:context="http://www.springframework.org/schema/context'
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
 http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-3.0.
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-
 <context:component-scan base-package="com.ignou" />
 <bean class="org.springframework.web.servlet.view.InternalRe:
   <property name="prefix" value="/WEB-INF/JSP/" />
```

HelloController.java



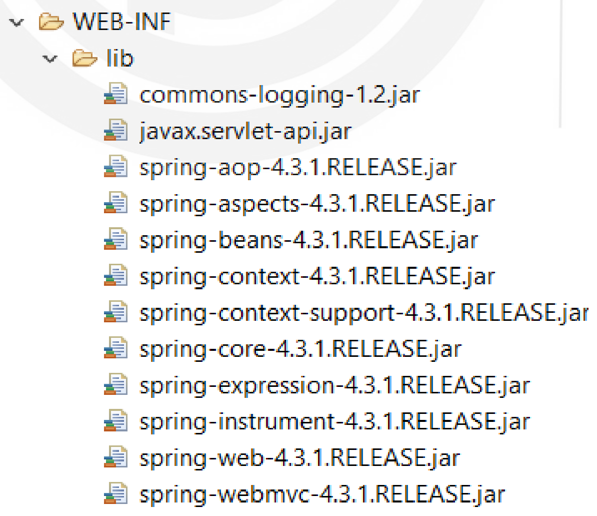
```java
package com.ignou;

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

Hello.jsp



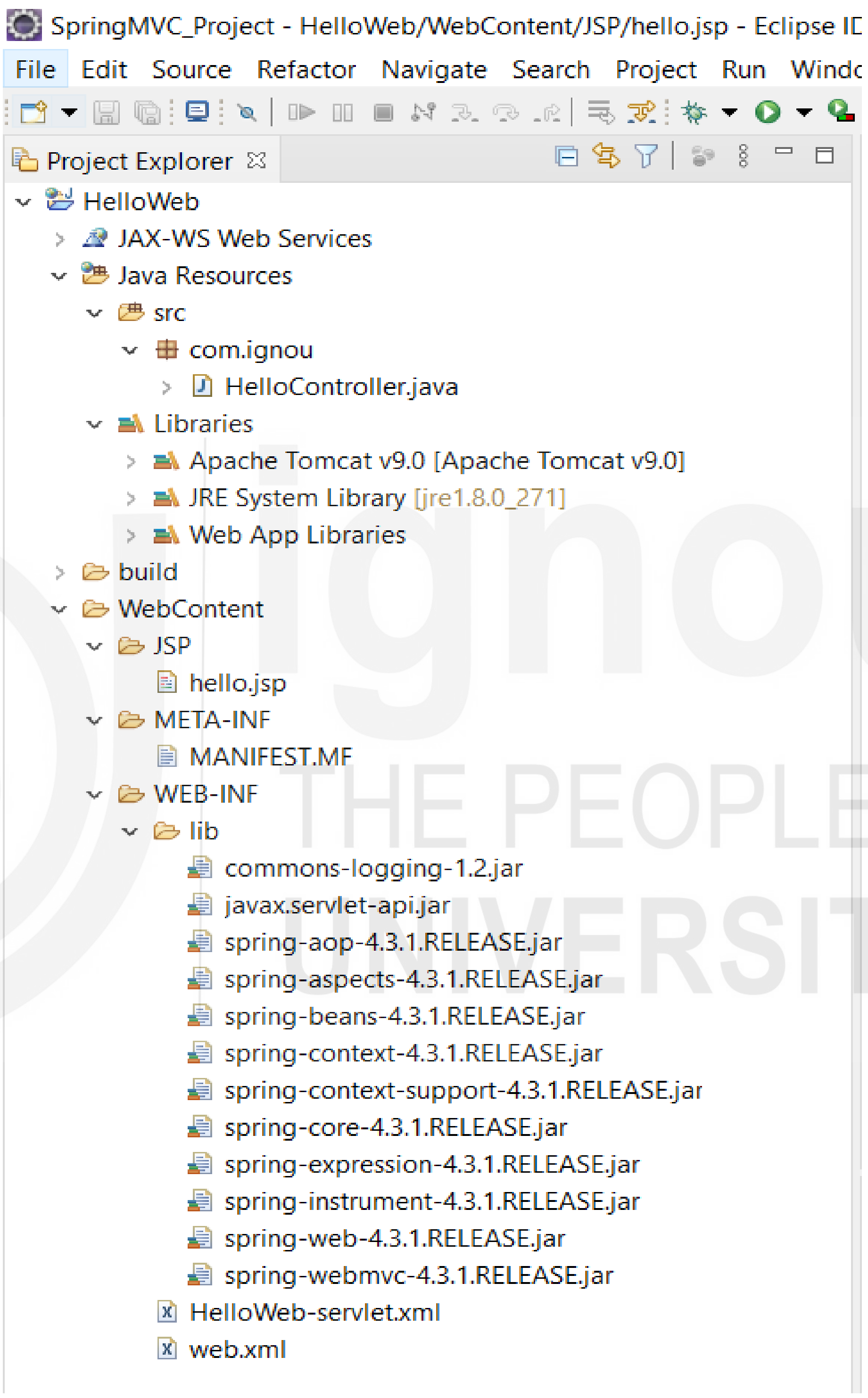
```jsp
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
 <h2>${message}</h2>
</body>
</html>
```

Need to include the below list of Spring and other libraries in our web application. We can do this by drag and drop them in in WebContent/WEB-INF/lib folder.



```
WEB-INF
  lib
    commons-logging-1.2.jar
    javax.servlet-api.jar
    spring-aop-4.3.1.RELEASE.jar
    spring-aspects-4.3.1.RELEASE.jar
    spring-beans-4.3.1.RELEASE.jar
    spring-context-4.3.1.RELEASE.jar
    spring-context-support-4.3.1.RELEASE.jar
    spring-core-4.3.1.RELEASE.jar
    spring-expression-4.3.1.RELEASE.jar
    spring-instrument-4.3.1.RELEASE.jar
    spring-web-4.3.1.RELEASE.jar
    spring-webmvc-4.3.1.RELEASE.jar
```

Finally, Project Explorer looks as below


SpringMVC_Project - HelloWeb/WebContent/JSP/hello.jsp - Eclipse IC
File Edit Source Refactor Navigate Search Project Run Windc
Project Explorer

```
HelloWeb
  JAX-WS Web Services
  Java Resources
    src
      com.ignou
        HelloController.java
    Libraries
      Apache Tomcat v9.0 [Apache Tomcat v9.0]
      JRE System Library [jre1.8.0_271]
      Web App Libraries
  build
  WebContent
    JSP
      hello.jsp
    META-INF
      MANIFEST.MF
    WEB-INF
      lib
        commons-logging-1.2.jar
        javax.servlet-api.jar
        spring-aop-4.3.1.RELEASE.jar
        spring-aspects-4.3.1.RELEASE.jar
        spring-beans-4.3.1.RELEASE.jar
        spring-context-4.3.1.RELEASE.jar
        spring-context-support-4.3.1.RELEASE.jar
        spring-core-4.3.1.RELEASE.jar
        spring-expression-4.3.1.RELEASE.jar
        spring-instrument-4.3.1.RELEASE.jar
        spring-web-4.3.1.RELEASE.jar
        spring-webmvc-4.3.1.RELEASE.jar
      HelloWeb-servlet.xml
      web.xml
```

Once the source and configuration files are created then you have to export the application. Do right click on the application and use Export > WAR File option for saving HelloWeb.war file in Tomcat's webapps folder or deploy this war file to any web/app server via admin role.

There is another options to run the app directly on the server from Eclipse.
You can check the working by using URL http://localhost:8080/HelloWeb/hello in your browser. For this you have to first start the Tomcat Server and using a standard browser check if you have access to other web pages from webapps folder.

7.4 INVERSION OF CONTROL (IOC) DEPENDENCY INJECTION

Inversion of Control (IoC): Inversion of Control (IoC) is considered as a design principle but some of the developer society consider it as a pattern. This is designed to achieve loose coupling as its name suggests that is used to invert controls on Object-Oriented design. It also controls the flow of an application and also controls over the flow of dependent object or an object creation and binding.

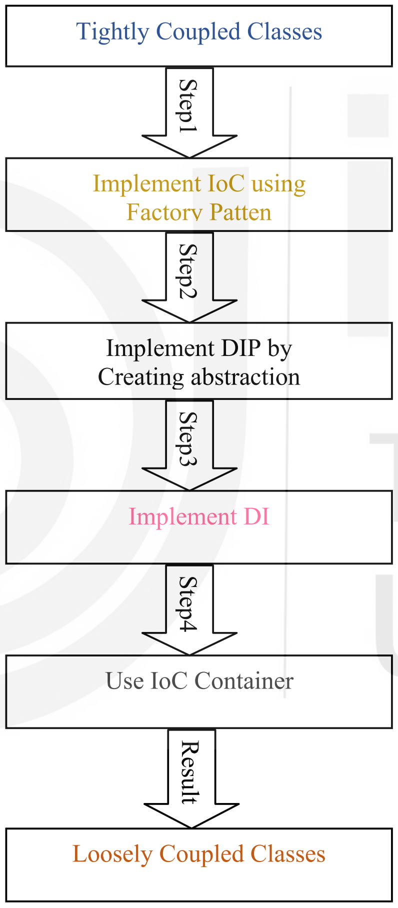


Figure 7.4: Inversion of Control Flow

IoC is all about overturning the control. In other words you can understand this by this example. Let us assume that you drive your car daily to your work place. This means you are controling the car. The IoC principle suggests to overturn the control, meaning that instead of you drive your car yourself, you hire a cab, where driver will be driving the car. Thus, this is called inversion of the control from you to the cab driver. Now you don't need to drive a car yourself but you can let the driver do the driving so that you can focus on your office work.

Classes of Spring IoC container are part of org.springframework.beans and org.springframework.context packages. Spring IoC container provides us different ways to decouple the object dependencies.

BeanFactory as the root Interface of Spring IoC Container. ApplicationContext is the child interface of BeanFactory and it provides Spring AOP features, i18n etc.

Dependency Injection:

The Dependency Injection is a design pattern that removes the dependency of the programs. To removes the dependency of the programs, Dependency Injection design pattern do this job. In that kind of scenario, information is provided by the external sources such as XML file. Dependency Injection plays a vital role to make the code loosely coupled and easier for testing. In such a case we write the code as:

```
class Ignou
{
  Student student;
  Ignou(Student student)
  {
    this.student =student;
  }
  public void setStudent(Student student)
  {
    this.student =student;
  }
}
```

We can perform Dependency Injection via following two ways in Spring framework

By Constructor – In the above code, the below snapshot of the code uses the constructor

```
Ignou(Student student)
{
  this.student =student;
}
```

By Setter method – In the above code, the below snapshot of the code uses the Setter method

```
public void setStudent(Student student)
{
  this.student =student;
}
```

☞ Check you progress 1:

Q1: What do mean by IDE and its benefits?

....

....

....

....

Q2: Is Spring Framework open source?

....

....

....

....

Q3: What is the use of IoC?

....

....

....

....

Q4: What is DI?

....

....

....

....

7.5 CREATING CONTROLLERS AND VIEWS

In Spring MVC, controller classes are used to handle the request which are coming from the client. After getting the request from client then controller starts its task by invoking a business class to process business-related tasks, and once it gets fulfilled then it redirects the client request to a logical view name, and now it is resolved by Spring's dispatcher servlet in order to render results or output.
Below are the main responsibilities:

- Controllers main task is to intercept the incoming requests.
- Controller Converts the payload of the request to the internal structure of the data (mapping basically).
- After getting response from above two steps then it sends the data to Model for further processing,
- Once it gets the processed data from the Model then that data is referred to the View for rendering/display.

Below diagram for the high level flow in Spring MVC:

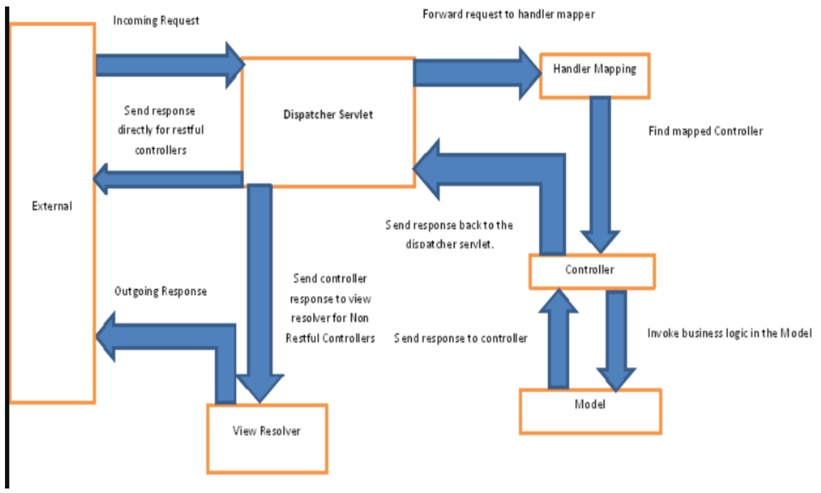

Incoming Request
Forward request to handler mapper
Handler Mapping
Send response directly for restful controllers
Dispatcher Servlet
Find mapped Controller
External
Send response back to the dispatcher servlet.
Controller
Outgoing Response
Send controller response to view resolver for Non Restful Controllers
Send response to controller
Invoke business logic in the Model
Model
View Resolver

Figure 7.5: High Level Flow in Spring MVC

There are basically two kind of constructors as typical MVC controllers as well as RESTful controllers and the above diagram caters to both with some small differences.

In the traditional approach, MVC applications are not service-oriented hence these applications rely on View technology as data is finally redirected to the views received from a Controller.
RESTful applications are designed as service-oriented and return raw/response data in the form of JSON/XML typically.
As data is returned in the form of JSON/XML hence these applications do not have Views, then the Controller is expected to send data directly via the HTTP response as per the designed structure.

Below is the example of controller from the above given.

```
packagecom.ignou;

importorg.springframework.stereotype.Controller;
importorg.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestMethod;
importorg.springframework.ui.ModelMap;
@Controller
@RequestMapping("/hello")
publicclassHelloIngnouController
{
  @RequestMapping(method = RequestMethod.GET)
  public String printHello(ModelMapmodel)
  {
    model.addAttribute("message", "Hello Spring MVC Framework!");
    return"hello";
  }
}
```

Views: FrontController design pattern is a fundamental design pattern to any MVC design and Spring MVC architecture also uses the “FrontController” design pattern. The DispatcherServlet is considered as the heart of this design as the process of dispatching the request to the right cpontroller is achieved by configuring the DispatcherServlet. This class handles the complete request handling process. It is a regular servlet as others and can be configured along with any custom handler mappings.

Spring MVC framework empowers parting of modules namely Model, View, and Controller and seamlessly handles the application integration. This permits the developer to develop complex applications also using plain java classes. The model object is passed between controller and view using Maps objects. Moreover, it also provides types mismatch validations and two-way data-binding support in the user interface. Data-binding and form-submission in the user interface becomes possible with spring form tags, model objects, and annotations.

In this article. We are discussing the steps involved in developing a Spring web MVC application, it explains the initial project setup for an MVC application in Spring. We have multiple options to use view technology but in this example we are using JSP(Java Server Pages) is used as a view technology.

Please find the below example which explains this whole concept of Controllers and Views.

DispatcherServlet is the root Servlet, or we can say that it is the front controller class to handle all requests and then to start processing. This needs to configure first in web.xml file. DispatcherServlet's primary duty is to pass the request to appropriate controller class and send the response back.

First step towards updating the web.xml.

```
<?xml version="1.0" encoding="UTF-8"?><web-app
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns="https://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="https://java.sun.com/xml/ns/javaee
https://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd"id="WebApp_ID"version="3.0">
<display-name>IGNOU_Example</display-name>
<!-- Adding DispatcherServlet as front controller -->
<servlet>
  <servlet-name>ignou-serv</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet
  </servlet-class><init-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring-servlet.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>ignou-serv</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

Second step is to create spring bean configuration file as ignou-serv-servlet.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beansxmlns="https://www.springframework.org/schema/mvc"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns:beans="https://www.springframework.org/schema/beans"
xmlns:context="https://www.springframework.org/schema/context"
xsi:schemaLocation="https://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd
https://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
https://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd">
<!-- DispatcherServlet Context: defines processing infrastructure -->
<!-- Enables @Controller programming model -->
<annotation-driven />
<context:component-scanbase-package="com.ignou" />
<!-- Resolves views -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<beans:property name="prefix" value="/WEB-INF/views/" />
<beans:propertyname="suffix"value=".jsp" />
</beans:bean>
</beans:beans>
```

There are basically three important configurations.

1. @Controller annotation tells DispatcherServlet to look for controller.

2. Where to look for controller classes, it is updated by context:component-scan to DispatcherServlet.

3. Location of view pages is handled in the configuration of InternalResourceViewResolver bean.

Third step is to write the controller class.

```
packagecom.ignou.controller;
importjava.text.DateFormat;
importjava.util.Date;
importjava.util.Locale;
importorg.springframework.stereotype.Controller;
importorg.springframework.ui.Model;
importorg.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestMethod;
importcom.journaldev.spring.model.User;
@ControllerpublicclassIgnouController
{
/** * Simply selects the home view to render by returning its name. */
@RequestMapping(value = "/", method = RequestMethod.GET) public String ignou(Locale
locale, Model model)
{
System.out.println("LandingIGNOU Page Requested = " + locale); Date date = newDate();
DateFormatdateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG,
DateFormat.LONG, locale);
String formattedDate = dateFormat.format(date); model.addAttribute("serverTime",
formattedDate);
return"ignou";
}
@RequestMapping(value = "/ignoustudent", method = RequestMethod.POST) public String
user(@Validated User user, Model model)
{
System.out.println("IGNOUStudent Page Requested"); model.addAttribute("studentName",
ignoustudent.getStudentName());
return"ignoustudent";
}
}
```

Fourth step is to define Model. This class can be represented by many names like Bean Class, POJO Class or Model Class.

```
packagecom.ignou.model;
publicclassIgnoustudent
{
private String studentName;
public String getStudentName()
{
returnstudentName;
}
publicvoidsetStudentName(String studentName)
{
this.studentName = studentName;
}
}
```

Now the last step is to create the View Pages.

```
Ignou.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ tagliburi="https://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page session="false"%>
<html>
<head>
<title>IGNOU Landing Page</title>
</head>
<body>
        <h1>Spring MVC Example For IGNOU</h1>

        <P>The time on the server is ${serverTime}.</p>

        <form action="user" method="post">
                <input type="text" name="studentName"><br> <input
                        type="submit" value="Login">
        </form>
</body>
</html>

Student.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"https://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>IGNOU Student Home Page</title>
</head>
<body>
<h3>Hello ${studentName}</h3>
</body>
</html>
```

This above example basically explains that what configurations are needed to do in web.xml and how do we define root servlet (DispatcherServlet) and defining of controllers, models, and views.

Lastly, we have to always remember below three important configurations.

1. @Controller annotation tells DispatcherServlet to look controller.

2. Where to look for controller classes, it is updated by context:component-scan to DispatcherServlet.

3. Location of view pages is handled in the configuration of InternalResourceViewResolver bean.

☞ Check you progress 2:

Q1. What is an Init Binder?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

Q2. What is the front controller class of Spring MVC?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

Q3. What is the difference between @Component, @Controller, @Repository & @Service annotations?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

Q4. What is ~~does~~ the ViewResolver class?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

7.6 REQUEST PARAMS AND REQUEST MAPPING

Request parameters can be read in controller class with the help of RequestParam annotation. Or we can say that views are sending some parameters with keys.

We can explain this with the help of student details which we want to add into database.

```
Student_ID
Student_FirstName
Student_LastName
Student_DOB

Just imagine that we have filled the below details.

Student_ID=1001 (depends upon requirement as it may be system generated)
Student_FirstName=Dharmendra
Student_LastName=Singh
Student_DOB =21.01.1979

So, we have the below code for this

@Controller
@RequestMapping("/ignoustudent/*")
public class IgnouStudentController
{
  @GetMapping("/fill")
  public String fill()
  {
    return "studentForm";
  }
  @PostMapping("/fill/process")
  public String processForm(ModelMap model, @RequestParam("Student_ID")
  intstud_id, @RequestParam("Student_FirstName") String stud_firstName,
  @RequestParam("Student_LastNAme") String stud_lastName,
  @RequestParam("Student_DOB") Date stud_dob)
  {
    model.addAttribute("Student_ID ", stud_id);
    model.addAttribute("Student_FirstName", stud_firstName);
    model.addAttribute("Student_LastName", stud_lastName);
    model.addAttribute("Student_DOB", stud_dob);
    return "index";
  }
}
```

We just binded web request parameters to method parameters (Student_id, Student_firstName, Student_lastName, Student _DOB).

In the above example, values are passed to request params, as @RequestParam("Student_id"). Though this could have not worked if the targent variable name is same as the param, we could have done it in the way given below:

```
@Controller
@RequestMapping("/ignoustudent/*")
public class IgnouStudentController
{
  @GetMapping("/fill")
  public String fill()
  {
    return "studentForm";
  }
  @PostMapping("/fill/process")
  public String processForm(ModelMap model, @RequestParamint_id,
  @RequestParam("FirstName") String firstName, @RequestParam("LastName")
  String lastName)
  {
    model.addAttribute("Student_id", id);
    model.addAttribute("Student_firstName", firstName);
    model.addAttribute("Student_lastName", lastName);
    model.addAttribute("Student_DOB", dob);
    return "index";
  }
}
```

Request Mapping:

Spring Controller supports the three levels of request mapping and it depends on the different @RequestMapping annotations declared in its handler methods and controller class.
Please find below the different mapping types:

By path
@RequestMapping(“path”)
By HTTP method
@RequestMapping(“path”, method=RequestMethod.GET).
Other Http methods such as POST,
PUT, DELETE, OPTIONS, and TRACE are also supported.
By query parameter
@RequestMapping(“path”, method=RequestMethod.GET,
params=”param1”)
By the presence of request header
@RequestMapping(“path”, header=”content-type=text/*”)

7.7 FORM TAGS AND DATA BINDING

For the Web Pages, Spring MVC has provided the form tags which are configurable and reusable building blocks. These Tags are very helpful for easy development, easy to read and maintain. These Tags can be used as data binding tags where these tags can set data to java objects (Bean or POJO).

These Tags provide support for corresponding HTML tags.

The form tag library comes under the spring-webmvc.jar. To get the support from the form tag library,you requires to reference some configuration So, you have to add the following directive at the beginning of the JSP page:

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

Some of the below frequently used Spring MVC form tags.

Form Tag	**Description**
form:form	It is a container tag that contains all other form tags.
form:input	This tag is used to generate the text field.
form:radiobutton	This tag is used to generate the radio buttons.
form:checkbox	This tag is used to generate the checkboxes.
form:password	This tag is used to generate the password input field.
form:select	This tag is used to generate the drop-down list.
form:textarea	This tag is used to generate the multi-line text field.
form:hidden	This tag is used to generate the hidden input field.

Spring data binding mechanism provides the functionality to bound the user inputs dynamically to the beans. It can explained as it allows the setting property values into a target object and this functionality is provided by DataBinder class though BeanWrapper also provides the same functionality but DataBinder additionally supports field formatting, binding result analysis and validation.

Where DataBinder works on high level and BeanWrapper works on low level. Both Binders are used in PropertyEditors to parse and format the property values. We should use the DataBinder as it works on high level and internally it uses the BeanWrapper only.

Data binding from web request parameters to JavaBean/POJO objects are done by WebDataBinder.We use it for customizing request param bindings.

We can understand better these topics with the help of the below example:

First, we are going to create a request page (Welcome.jsp)

```
<html>
<body>
<h2> Spring MVC Form Tag with Data Binding Example for IGNOU Students </h2>
<a href = "home_page">Home page | </a>
<a href = "about_us"> About Us | </a>
<a href = "student_form"> Student Detail Form</a>
</body>
</html>
```

Create model (getter/setter) class (Student.java)

```
package com.ignou.studentdetails;
public class Student
{
  private String fname;
  private String lname;
  public Student()
   {
   }
  public String getFname()
   {
     return fname;
   }
  public void setFname(String fname)
   {
      this.fname = fname;
   }
   public String getLname()
   {
      return lname;
   }
   public void setLname(String lname)
   {
      this.lname = lname;
   }
}
```

Create the controller (StudentDetailController.java) now

```
package com.ignou.studentdetails;
 import org.springframework.stereotype.Controller;
 import org.springframework.ui.Model;
 import org.springframework.web.bind.annotation.ModelAttribute;
 import org.springframework.web.bind.annotation.RequestMapping;
 import org.springframework.web.bind.annotation.RequestParam;
 @Controller
 public class StudentDetailController
{
   @RequestMapping("/student_form")
   public String showStudentForm( Model m)
   {
       Student student = new Student();
       m.addAttribute("student", student);
       return "studentform" ;
   }
   @RequestMapping("/studentregis")
   public String showStudentData(@ModelAttribute("student") Student student)
   {
       System.out.println("student:" + student.getFname() +" "+ student.getLname());
       return "student-data" ;
   }
}
```

Add the entry of controller in Web.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns = "http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation                    =                    "http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        id = "WebApp_ID" version = "3.1">
<display-name>spring-mvc-demo</display-name>
<absolute-ordering />
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/spring-servlet.xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
```

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

Define model in 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">
<!-- Step 3: Add support for component scanning -->
<context:component-scan base-package = "com.app.SpringMVC5" />
<!-- Step 4: Add support for conversion, formatting and validation support -->
<mvc:annotation-driven/>
<!-- Step 5: Define Spring MVC view resolver -->
<bean

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

Create the view pages

StudentDetailForm.jsp

```
<%@ taglib prefix = "form" uri = "http://www.springframework.org/tags/form" %>
<html>
<head>
<title>Student Detail Form</title>
</head>
<body>
<form:form action = "studentdetail" modelAttribute = "student" >
 FIRST NAME: <form:input path = "fname" />
<br></br>
 LAST NAME: <form:input path = "lname" />
<br></br>
<input type = "submit" value = "Submit"/>
</form:form>
</body>
</html>
```

StudentDetailData.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>Student Detail Data</title>
</head>
<body>
 The student name is ${student.fname} ${student.lname}
</body>
</html>
```

7.8 FORM VALIDATION

Spring Validation is very important as it is used for validating the @Controller inputs. Using Spring validation, the user input on the server-side can be validated. It is used for checking and subsequently accepting the user input in any web application. From Spring framework version 4 and above, the Spring framework supports Bean Validation 1.0 (JSR-303) and Bean Validation 1.1 (JSR-349).

BindingResult is an an interface that represents the binding results. It extends an interface for error registration for allowing a validator to be applied, and adds binding-specific analysis.

In Spring MVC form Validation, a variety of annotations are used. These annotations are available in javax.validation.constraints package. Below is a list of commonly used Validation annotations:

Annotations	**Description**
@Valid	This annotation is used on a model object that we want to validate
@Size	It specifies that the size of the annotated element must be between thespecified boundaries.
@Pattern	It specifies that the annotated CharSequence must match the regularexpression.
@Past	It specifies that the annotated element must be a date in the past.
@Null	It specifies that the annotated element must be null.
@NotNull	It specifies that the annotated element should not be null.
@Min	It specifies that the annotated element must be a number whose value mustbe equal or greater than the specified minimum number.
@Max	It specifies that the annotated element must be a number whose value mustbe equal or smaller than the specified maximum.
@Future	It specifies that the annotated element must be a date in the future.
@Digits	It specifies that the annotated element must be a digit or number within thespecified range. The supported types are short, int, long, byte, etc.
@DecimalMin	It specifies that the annotated element must be a number whose value must be equal or greater than the specified minimum. It is very similar to @Minannotation, but the only difference is it supports CharSequence type.
@DecimalMax	It specifies that the annotated element must be a number whose value should be equal or smaller than the specified maximum. It is very similarto @Max annotation, but the only difference is it supports CharSequence type.

@AssertTrue	It determines that the annotated element must be true.
@AssertFalse	It determines that the annotated element must be false

We can use the model (Student.java) from previous example and put the validations inside form class as below

```
package com.ignou.studentdetails;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
public class Student
{
@NotNull
@Size(min =1, message ="You can't leave this empty.")

  private String fname;
@NotNull
@Size(min =1, message ="You can't leave this empty.")
  private String lname;
  public Student()
  {
  }
  public String getFname()
  {
    return fname;
  }
  public void setFname(String fname)
  {
    this.fname = fname;
  }
  public String getLname()
  {
    return lname;
  }
  public void setLname(String lname)
  {
    this.lname = lname;
  }
}
```

☞ Check you progress 3:

Q1: What do you understand by Tag libraries?

....

....

....

....

Q2: What are the @RequestBody and the @ResponseBody?

....

....

....

....

Q3: State the difference between Request Param & Request Mapping

....

....

....

....

Q4: What is the Role of the @Autowired Annotation?

....

....

....

....

7.9 SUMMARY

Spring's Web MVC Framework is very popular and used framework due to its request-driven, designed around a central Servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications. It also facilitates fast and parallel development. It Reuses business code - instead of creating new objects. It allows us to use the existing business objects. It provides easy to test functionality as we create JavaBeans classes that enable us to inject test data using the setter methods. This framework is used in many products development like Hybris and banking domain products.

7.10 ANSWER TO CHECK YOUR PROGRESS

Check you progress 1:

1. An IDE stands for integrated development environment for programming in Java or any language. It is typically an application software which provide a code editor, a compiler or interpreter and a debugger that can be accessed by the application developer using unified graphical user interface (GUI). Also many Java IDEs includes language-specific elements such as Ant and Maven build tools and TestNG and JUnit testing.

2. Spring Framework is an open-source framework used for developing web applications with Java as a programming language. It is a powerful lightweight application development framework used for Java Enterprise Edition (JEE). We can also say that it is a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc.

3. The IoC container is responsible to instantiate, configure and assemble the objects. The IoC contains gets informations from the XML file and works accordingly and its primary tasks performed by IoC container are:

To instantiate the application class
To configure the object
To assemble the dependencies between the objects

There are two types of IoC containers. They are:

BeanFactory
ApplicationContext

4. Dependency Injection (DI) is a design pattern which removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

Check you progress 2:

1. The Init Binder is an annotation that is used to customize the request being sent to the controller. It is used to to control and format those requests that come to the controller.

2. A controller is used to handl all requests for a Web Application is called as Front Controller. DispatcherServlet (actually a servlet) is the front controller in Spring MVC that not only intercepts every request but also dispatches/forwards requests to an appropriate controller.

3. One of the major difference between these stereotypes is that these are used for different classification. You may have different layers like presentation, service, business, data access etc. in a multitier application. When you try to annotate~~d~~ a class for auto-detection by Spring, then you should have to use the respective stereotype as below:

@Component – generic and can be used across application.
@Service – annotate classes at service layer level.
@Controller – annotate classes at presentation layers level, mainly used in Spring MVC.
@Repository – annotate classes at persistence layer, which will act as database repository.

4. ViewResolver is an interface wich is to be implemented by objects. This is used to resolve views by name. There are many other ways also, using which you can resolve view names. These ways are supported by various in-built implementations of ViewResolve.

Check you progress 3:

1. Custom tags are implemented and distributed using taglib. It is a structure which is known as a tag library. A tag library is nothing but a a collection of classes and meta-information that includes:

- … Tag Handlers Java classes. This class implement the functionality of custom tags.
- … Tag Extra Information Classes. These classes are used to supply the JSP container with logic for validating tag attributes and creating scripting variables.
- … A Tag Library Descriptor (TLD). It is an XML document, used to describe the properties of the individual tags and the tag library as a whole.

2. @RequestBody and @ResponseBody annotations are used to convert the body of the HTTP request and response with java class objects. Both these annotations will use registered HTTP message converters in the process of converting/mapping HTTP request/response body with java objects.

3. @RequestMapping is a class or method annotation is used to map the request url to the java method.
@RequestParam is a (Method) field annotation is used to bind a request parameter to a method parameter

4. The @Autowired annotation can be used to autowire (placing an instance of one bean into the desired field of another bean) bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names or multiple arguments.

7.11 FURTHER READINGS/ REFERENCES

1. Craig Walls, "Spring Boot in action" Manning Publications, 2016.
(https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf)
2. Paul Deck, "Spring MVC: a Tutorial", Brainy Software, 2016.
3. Ian Roughley,"Practical Apache Struts 2 Web 2.0 Projects", Dreamtech Press, 2008.
4. Ethan Marcotte, "Responsive Web Design", Jeffrey Zeldman Publication, 2011
5. https://www.oracle.com/java/technologies/java-technology-
6. reference.html#documentation
7. https://spring.io/projects
8. https://www.tutorialandexample.com/spring-mvc-form-validation/

UNIT 8 SPRING MVC WITH BOOTSTRAP CSS

8.0 INTRODUCTION

Nowadays, huge traffic comes from mobile devices, and most of the internet searches like google are done on mobile devices, whether smartphones or tablets. The internet users who search a company to buy food or items, visit the company's website from the google link on the mobile devicesthemselves. If a company's website is not flexible across all screen resolutions and devices, it risks missing out on a large group of buyers. A responsive website prevents this loss and leads to substantial revenue gains.

Responsive web design improves user experience and creates a positive perception for a company or business. If your customers can access your website easily on all platforms, they're more likely to return to you for more business in the future.

It is a big challenge for website developers to keep the design responsive as the web evolves more and more. Bootstrap is the solution to this big challenge and makes things a whole lot easier. Bootstrap takes care of everything related to responsiveness without the user need to do the responsive bit. Bootstrap automatically scales in and out among the devices such as mobile phones, tablets, laptops, desktop computers, screen readers, etc.

Spring is the most popular and widely used Java EE framework, while Hibernate is the most popular ORM framework to interact with databases. Spring supports integrating Hibernate very easily, which makes the Hibernate thefirst choice as an ORM with Spring.

CRUD is an acronym for CREATE, READ, UPDATE and DELETE operations which are used in relational database applications and executed by mapping to a SQL statement or by standard HTTP verbs such as POST, GET, PUT, DELETE. CRUD is thus data-oriented and uses standardized HTTP action verbs.

8.1 OBJECTIVES

After going through this unit, you will be able to:

- Describe responsive web design and show its advantages,

- use Bootstrap integration with Spring MVC,
- apply different approaches to include Bootstrap into Spring MVC,
- include custom CSS into Spring applications,
- do Spring and Hibernate integration,
- do CRUD execution mapping to SQL and Rest Web Service, and
- develop a complete Spring MVC web application with Bootstrap, custom CSS and Hibernate.

8.2 CONFIGURATION OF BOOTSTRAP IN SPRING APPLICATION

Bootstrap was originally created by a designer and developer at Twitter in mid 2010. Prior to being an open-sourced framework, Bootstrap was known as *Twitter Blueprint*. **Bootstrap** is a free, popular and open-source CSS framework directed at responsive, mobile-first front-end web development. Unlike many web frameworks, it concerns itself with front-end development only. It contains HTML and CSS based design templates for various interface components such as typography, forms, buttons, modals, image carousels, navigation etc., as well as optional JavaScript extensions. Bootstrap provides the ability to create responsive designs faster and easily. **Responsive Web Design** (**RWD**) is an approach to web design that makes web pages adjust automatically to look good and render well on a variety of devices and window or screen sizes from minimum to maximum display size. The various factors which support learning Bootstrap are as follows.

- **Browser Support:** Bootstrap supports all popular web browsers such as Chrome, Safari, Firefox, Opera, and Internet edge.
- **Responsive Design:** Bootstrap enables us to write responsive websites very easily. Its responsive CSS adjusts automatically to Desktops, Tablets and Mobiles.

- **Uniform solution to build interface:** Bootstrap provides a clean and uniform solution to build an interface for the developers.
- **Customization:** Bootstrap provides a simple and easy way to customize the CSS.
- **Functional built-in components:** Bootstrap contains beautiful and functional built-in components which can be customized very easily.

8.2.1 Different ways to configure Bootstrap

The Bootstrap 5 is the most recent version of the Bootstrap framework.There are two ways to include Bootstrap into JSP or HTML.The required configurations for Bootstrap 4 and Bootstrap 5 has been described as follows.

1. **Bootstrap through Content Delivery Network (CDN)**

   Bootstrap CDN is an efficient and faster way to deliver the content from your website to the users. Bootstrap CDN speeds up the websites as CDN servers are intelligent enough to serve the resources based on proximity.

Including Bootstrap 4 into JSP/HTML.

For some of the functionalities Bootstrap 4 JavaScript file requires jQuery and Popper JavaScript, and these two must be loaded before loading of bootstrap.min.js file.

- CSS: Include the below link to the <head> tag of your desired HTML/JSP file.

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/boots
trap.min.css" integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
```

- JS: Place the following <script>s near the end of your pages, right before the closing </body> tag, to enable them. jQuery must come first, then Popper.js, and then our JavaScript plugins.

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/p
opper.min.js" integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>

<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstr
ap.min.js" integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
```

Including Bootstrap 5 into JSP/HTML

Bootstrap 5 no longer needs jQuery as a dependency since JavaScript can provide the same functionality. Add the following CSS and JS through Bootstrap 5 CDN to JSP/HTML:

- CSS: Copy-paste the below Stylesheet link to the <head> tag of your desired HTML/JSP file.

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootst
rap.min.css" rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
```

- JS: Place the following <script>s near the end of your pages, right before the closing </body> tag.

```
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/po
pper.min.js" integrity="sha384-
IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstra
p.min.js" integrity="sha384-
cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"
crossorigin="anonymous"></script>
```

2. Custom configuration of Bootstrap with Spring MVC (Offline by downloading files locally)

Another approach to include Bootstrap is to directly download the Bootstrap CSS and JS files locally to the project folder and then include the local copy into JSP/HTML. The following links can be used to download the Bootstrap.

- Bootstrap 4: https://getbootstrap.com/docs/4.3/getting-started/download/
- Bootstrap 5: https://v5.getbootstrap.com/docs/5.0/getting-started/download/

Download the Bootstrap and perform the following steps to configure Bootstrap in the Spring application.

i. Put static resources CSS and JS files into webapp/resources/static directory. To segregate CSS and JS, keep them into respective folders webapp/resources/static/js and webapp/resources/static/css into the eclipse project.

ii. Create the resource mapping into Spring using either XML configuration or Java configuration.

XML configuration

```
<mvc:resources mapping="/js/**" location="/resources/static/js/" />
<mvc:resources mapping="/css/**" location="/resources/static/j=css/" />
```

Java configuration

```
@Configuration
@EnableWebMvc
public class MyApplication implements WebMvcConfigurer
{
   public void addResourceHandlers(final ResourceHandlerRegistry registry)
   {
      registry.addResourceHandler("/js/**").addResourceLocations("/resources/static/js/");
      registry.addResourceHandler("/css/**").addResourceLocations("/resources/static/css/");
   }
}
```

iii. Include the Bootstrap CSS and JS into JSP page via JSTL tag **c:url** or Spring tag **spring:url**.
JSTL tag **c:url**

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html lang="en">
<head>
<link href="<c:url value="/css/bootstrap.css" />" rel="stylesheet">
<script src="<c:url value="/js/jquery.1.10.2.min.js" />"></script>
<script src="<c:url value="/js/bootstrap.js" />"></script>
</head>
<body>
</body>
</html>
```

Spring tag **spring:url**

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<spring:url value="/css/bootstrap.css" var="bootstrapCss" />
<spring:url value="/js/jquery.1.10.2.min.js" var="jqueryJs" />
<spring:url value="/js/bootstrap.js" var="bootstrapJs" />

<link href="${bootstrapCss}" rel="stylesheet" />
<script src="${jqueryJs}"></script>
<script src="${bootstrapJs}"></script>
</head>
<body><body>
</body>
</html>
```

☞Check Your Progress 1

1) What is Bootstrap and what factors make Bootstrap a popular choice to use in web applications?

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

2) What are the different ways to configure Bootstrap in Spring MVC?

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

3) What are the obvious reasons to use Bootstrap CDN to configure it with Spring?

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

8.3 APPLY CUSTOM CSS IN PAGES

CSS or Cascading Style Sheets is a method of adding stylistic instructions for your website to its back-end code. The HTML (hypertext markup language) is the most common coding language for a website. CSS is an extension of HTML that outlines specific stylistic instructions to style websites. It allows you to customize the look of your website and apply stylistic decisions across its entirety. It also allows you to separate the style from the structure of a web page. CSS is used to specify the color of a heading or what font your content should be written in. CSS can be embedded into HTML elements, as shown below.

```
<p style="color: green;">
    This is Green Color Text.
</p>
```

The above changes the style of the text for a particular <p> tag. This style can be separated from <p> tag with following instructions.

```
<head>
<style type="text/css">
  p
 {
    color: green;
 }
</style>
</head>
<body>
<p>
 This is Green Color Text.
</p>
<p>
This is another paragraph with a green color.
</p>
</body>
```

The above separated embedded style instructions change all the text within <p> tags **on a page** to green. A website generally consists of multiple pages, and the pages should be uniform and consistent in terms of font-size, font-color, font-family etc. To keep the style uniform across the pages, a separate CSS file with any name e.g. custom.css can be created, and custom styles can be added to this file. The required configuration for custom css, in Spring MVC is like using the Bootstrap downloading the files locally described into section 8.2.1. The custom css can be included at the top of each webpage inside the <head> tag using the below instruction.

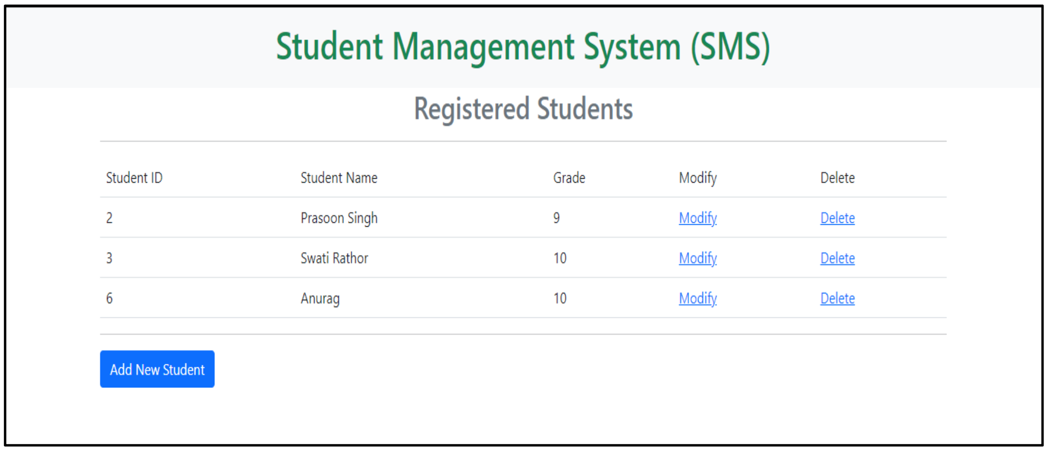
```
<link rel="stylesheet" type="text/css" href="css/custom.css">
```

There are three main components that must be understood clearly to proceed with adding customized CSS language to the website. The components are **selectors, properties and values**.

The selector uses HTML to identify the part of your website that you want to style. For example, the HTML code for a paragraph is “p” and if you want to use CSS to change the style of the paragraph, “p” becomes your selector.

Properties and values are then used to apply stylistic instructions to the selectors. If you want your paragraphs to be written in red text, the property will come first and will indicate the specific attribute that you’re trying to change (**color in this case**). The value is what you want to change that attribute to, which is **red** in our example.

The section 8.6 describes the CRUD application using Spring MVC and Hibernate. Custom CSS and configuration code have been described there. Check the impact of custom CSSon the following screens.

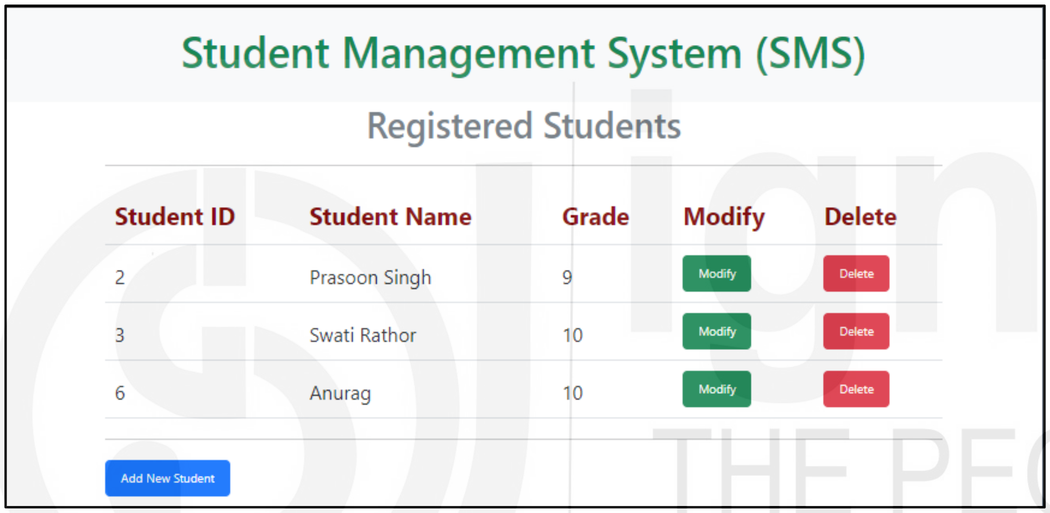


Student Management System (SMS)

Registered Students

Student ID	Student Name	Grade	Modify	Delete
2	Prasoon Singh	9	Modify	Delete
3	Swati Rathor	10	Modify	Delete
6	Anurag	10	Modify	Delete

Figure 8.1: CRUD Application Page Using Bootstrap



Student Management System (SMS)

Registered Students

Student ID	Student Name	Grade	Modify	Delete
2	Prasoon Singh	9	Modify	Delete
3	Swati Rathor	10	Modify	Delete
6	Anurag	10	Modify	Delete

Figure 8.2: CRUD Application Page Using Bootstrap And Custom CSS

8.4 SETTING UP DATABASE USING HIBERNATE(Database configuration using Hibernate)

Spring supports bootstrapping the Hibernate SessionFactory in order to set up the database using Hibernate. The database setup can be done with a few lines of Java code or XML configuration. Spring provides two key beans to support the Hibernate integration, available in the org.springframework.orm.hibernate5 package:

- **LocalSessionFactoryBean**: creates a Hibernate's **SessionFactory** which is injected into Hibernate-based DAO classes.
- **PlatformTransactionManager**: provides transaction support code for a **SessionFactory**. Programmers can use **@Transactional** annotation in DAO methods to avoid writing boiler-plate transaction code explicitly.

The following section describes how these concepts are implemented in a real project with the required maven dependencies and Java configuration for Hibernate.

Maven Dependencies

The necessary dependencies for pom.xml to integrate Hibernate with Spring are as follows.

```
<dependency>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-orm</artifactId>
                    <version>5.1.0.RELEASE</version>
             </dependency>

             <!--Hibernate -->
             <dependency>
                    <groupId>org.hibernate</groupId>
                    <artifactId>hibernate-core</artifactId>
                    <version>5.3.5.Final</version>
             </dependency>

             <!-- MySQL -->
             <dependency>
                    <groupId>mysql</groupId>
                    <artifactId>mysql-connector-java</artifactId>
                    <version>8.0.16</version>
             </dependency>
```

8.4.1 Java based configuration for Spring and Hibernate Integration

The required beans such as LocalSessionFactoryBean, DataSource, and PlatformTransactionManager, as well as some Hibernate-specific properties have been defined into HibernateConfig.

```
packagecom.ignou.mvcapp;
importjava.util.Properties;
importjavax.sql.DataSource;
importorg.springframework.beans.factory.annotation.Autowired;
importorg.springframework.context.annotation.Bean;
importorg.springframework.context.annotation.Configuration;
importorg.springframework.context.annotation.PropertySource;
importorg.springframework.core.env.Environment;
importorg.springframework.jdbc.datasource.DriverManagerDataSource;
importorg.springframework.orm.hibernate5.HibernateTransactionManager;
importorg.springframework.orm.hibernate5.LocalSessionFactoryBean;
importorg.springframework.transaction.PlatformTransactionManager;
importorg.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@EnableTransactionManagement
@PropertySource(value = { "classpath:application.properties" })
publicclassHibernateConfig
{
  @Autowired
  private Environment environment;

  @Bean
  publicLocalSessionFactoryBeansessionFactory()
  {
    LocalSessionFactoryBeansessionFactory = newLocalSessionFactoryBean();
    sessionFactory.setDataSource(dataSource());
    sessionFactory.setPackagesToScan(newString[] { "com.ignou.mvcapp.model" });
    sessionFactory.setHibernateProperties(hibernateProperties());
    returnsessionFactory;
  }

  @Bean
  publicDataSourcedataSource()
  {
    DriverManagerDataSourcedataSource = newDriverManagerDataSource();
    dataSource.setDriverClassName(environment.getRequiredProperty("jdbc.driverClassName"));
    dataSource.setUrl(environment.getRequiredProperty("jdbc.url"));
    dataSource.setUsername(environment.getRequiredProperty("jdbc.username"));
    dataSource.setPassword(environment.getRequiredProperty("jdbc.password"));
    returndataSource;
  }

  private Properties hibernateProperties()
  {
    Properties properties = newProperties();
    properties.put("hibernate.dialect", environment.getProperty("hibernate.dialect"));
    properties.put("hibernate.show_sql", environment.getProperty("hibernate.show_sql"));
    properties.put("hibernate.format_sql", environment.getProperty("hibernate.format_sql"));
    properties.put("hibernate.hbm2ddl.auto", environment.getProperty("hibernate.hbm2ddl.auto"));
    returnproperties;
  }
  @Bean
  publicPlatformTransactionManagergetTransactionManager()
  {
    HibernateTransactionManagerhtm = newHibernateTransactionManager();
    htm.setSessionFactory(sessionFactory().getObject());
    return htm;
  }

}
```

Property file inside src/main/resources. File name can be any with extension as properties.

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/schoolDB
jdbc.username = root
jdbc.password = root

hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
hibernate.format_sql = true
hibernate.hbm2ddl.auto = update
```

☞Check Your Progress 2

1) What is custom CSS and how can these be included into a JSP?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2) Write the custom CSS to make all the paragraphs of a page to red color. Write the sample JSP page for it.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

3) What are the key beans provided by Spring to integrate with Hibernate?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

4) Write the skeleton code for Hibernate Configuration code into Spring.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

8.5 CREATE, READ, UPDATE, AND DELETE (CRUD)

Within computer programming, the acronym CRUD stands for CREATE, READ, UPDATE and DELETE. The CRUD paradigm is common in constructing web applications. While constructing the APIs, the model should provide four basic types of functionality such as Create, Read, Update and Delete the resources. CRUD operations are also often used with SQL.

It can also describe user-interface conventions that allow viewing, searching and modifying information through computer-based forms and reports. Most applications have some form of CRUD functionality. Let's consider that we have a student database that stores information for students and grades. When programmers provide interactions with this database (often through stored procedures), the steps of CRUD are carried out as follows:

- Create: A new student is entered into the database.
- Read: The student's information is displayed to the users.
- Update: Already, existing student's attributes are being updated with new values.
- Delete: If the student is not part of the institute, the student can be deleted from the records.

8.5.1 Mapping CRUD to SQL Statements

The previous section explained the concept of **CRUD** and defined it based on different contexts. The following table maps **CRUD to database sql** operation with an example.

Function	SQL Statement	Example
C(reate)	Insert	Insert into student (name,grade) values('Prasoon',10);
R(ead)	Select	Select * from student
U(pdate)	Update	Update student set grade=10 where id=1;
D(elete)	Delete	Delete from student where id=1;

8.5.2 Mapping CRUD to REST

In **REST context**, CRUD corresponds to Rest web service **POST, GET, PUT and DELETE** respectively. The following table maps CRUD to REST with example.

Function	Rest web service	Example
C(reate)	POST	POST http://teststudent.com/students/
R(ead)	GET	GET http://teststudent.com/students/
U(pdate)	PUT	PUT http://teststudent.com/students/1
D(elete)	DELETE	DELETE http://teststudent.com/students/1

The following sections explain the CRUD operations in the REST environment for the system, Student Management System (SMS), to keep the student records with their grade.

- **Create**

  To create a resource in a REST environment, HTTP Post method is used. Post method creates a new resource of the specified resource type. To create a new student record for the Student Management System, HTTP POST request and response are shown below.

  - **Request:** POST http://teststudent.com/students/
  - **Request Body:**

    ```
    {
      "firstName": "Anurag",
      "lastName": "Singh",
      "grade": 10
    }
    ```
  - **Response:** Status Code – 201(Created)

    ```
    {
      "id": 1,
      "firstName": "Anurag",
      "lastName": "Singh",
      "grade": 10
    }
    ```

- **Read**

  To read resources in a REST environment, HTTP GET method is used. Read operation should not change the resource. The Get method returns the same response irrespective of number of times of execution. HTTP GET method is **idempotent**. A HTTP method is called idempotent if an identical request can be made once or several times in a row with the same effect while leaving the server in the same state. To retrieve all student records from the Student Management System, HTTP GET method request and response is shown below.

  - **Request:** GET http://teststudent.com/students/
  - **Response:** Status Code – 200 (OK)

    ```
    [
    {"id": 1, "firstName": "Anurag", "lastName": "Singh", "grade": 10},
    {"id": 1, "firstName": "Dinesh", "lastName": "Chaurasia", "grade": 10}
    ]
    ```

- **Update**

  To update a resource in a REST environment, HTTP **PUT or PATCH** method is used. This operation is idempotent. Request and response to update the grade of student id 1, is shown below.

  - **Request:** PUT http://teststudent.com/students/1
  - **Request Body:**

    ```
    {
      "id": 1,
      "firstName": "Anurag",
      "lastName": "Singh",
      "grade": 8
    }
    ```
  - **Response:** Status Code – **200 (OK)**. The response includes a Status Code of **200 (OK)** to signify that the operation was successful, but it need not return a response body.

… **Delete**

To delete a resource in a REST environment, **HTTP DELETE** method is used. It is used to remove a resource from the system. This operation is also **idempotent**. Request and response to delete a student with id 1, is shown below.

- Request: DELETE http://teststudent.com/students/1
- **Response:** Status Code – **204 (No Content)**. Successful delete operation returns a response code of **204 (NO CONTENT)**, with no response body.

8.6 CRUD EXAMPLES IN SPRING MVC AND HIBERNATE

The previous section has explained the CRUD operation for the REST environment. This section is full of code to integrate **Spring MVC** and **Hibernate** to implement the **CRUD** operation at the database level using**Mysql** database. Givenbelow is described the Student Management System application implements which were explained in previous sections.These include the following:

- BootStrap configuration into Spring MVC using CDN
- Custom CSS
- Spring and Hibernate integration
- CRUD operation implementation with Mysql

The required tools and software to complete the implementations of Student Management System application are as follows:

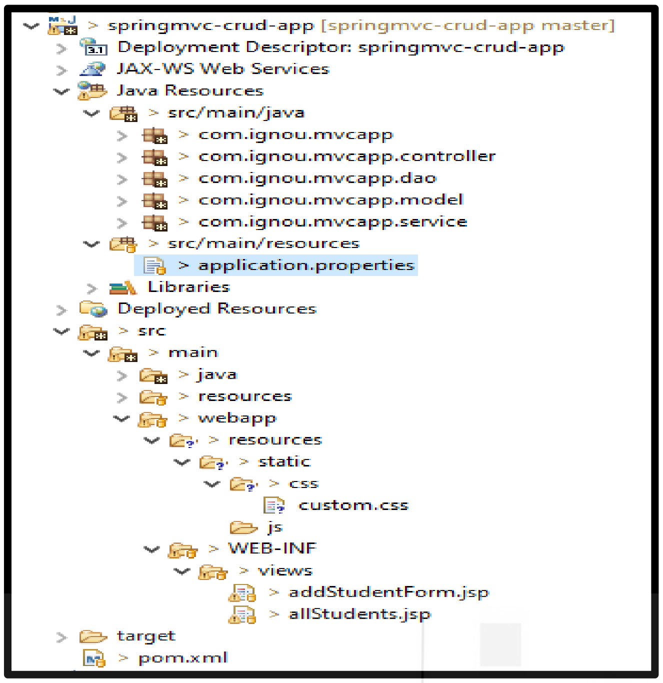
- Eclipse IDE
- Maven
- Tomcat 9
- MySql 8
- Java 8 or Higher

Perform the following steps to get a good hands-on in **Spring MVC and Hibernate integration along with Bootstrap and custom CSS.**

Step 1: Create a maven project for a web application using the below maven command and import the project into eclipse. Maven project creation is described in unit 6 section 6.7.

```
mvnarchetype:generate -DgroupId=com.ignou.springcrud -DarchetypeGroupId=org.apache.maven.archetypes -
DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion=1.4 -DartifactId=springmvc-crud-app -
DinteractiveMode=false
```

If you get any error like missing folder into eclipse project, go to Java Build Path into project properties. Select the JRE and Maven Dependencies into Order and export tab. Project folder structure is shown below.



```
springmvc-crud-app [springmvc-crud-app master]
  Deployment Descriptor: springmvc-crud-app
  JAX-WS Web Services
  Java Resources
    src/main/java
      com.ignou.mvcapp
      com.ignou.mvcapp.controller
      com.ignou.mvcapp.dao
      com.ignou.mvcapp.model
      com.ignou.mvcapp.service
    src/main/resources
      application.properties
    Libraries
  Deployed Resources
  src
    main
      java
      resources
      webapp
        resources
          static
            css
              custom.css
            js
        WEB-INF
          views
            addStudentForm.jsp
            allStudents.jsp
  target
  pom.xml
```

Figure 8.3: CRUD Application Folder Structure In Eclipse

Add the maven dependencies into pom.xml, which is given in **section 8.4**

Step2: Create the configuration classes into the package **com.ignou.mvcapp**
WebMvcConfig.java

```
packagecom.ignou.mvcapp;
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.ignou.mvcapp")
publicclassWebMvcConfigimplementsWebMvcConfigurer
{
  @Bean
  publicViewResolverviewResolver()
  {
    InternalResourceViewResolverviewResolver = newInternalResourceViewResolver();
    viewResolver.setViewClass(JstlView.class);
    viewResolver.setPrefix("/WEB-INF/views/");
    viewResolver.setSuffix(".jsp");
    returnviewResolver;
  }

  @Bean
  publicMessageSourcemessageSource()
  {
    ResourceBundleMessageSourcemessageSource = newResourceBundleMessageSource();
    messageSource.setBasename("messages");
    returnmessageSource;
  }

  @Override
  publicvoidaddResourceHandlers(ResourceHandlerRegistryregistry)
  {
    registry.addResourceHandler("/js/**").addResourceLocations("/resources/static/js/");
    registry.addResourceHandler("/css/**").addResourceLocations("/resources/static/css/");
    registry.addResourceHandler("/images/**").addResourceLocations("/resources/static/images/");
  }
}
```

HibernateConfig.java

The Hibernate integration with Spring MVC configuration code has been given into section **8.4.1**.

AppInitializer.java

```
packagecom.ignou.mvcapp;
publicclassAppInitializerextendsAbstractAnnotationConfigDispatcherServletInitializer
{

  @Override
  protected Class<?>[] getRootConfigClasses()
  {
    returnnewClass[] { HibernateConfig.class };
  }

  @Override
  protected Class<?>[] getServletConfigClasses()
  {
    returnnewClass[] { WebMvcConfig.class };
  }

  @Override
  protectedString[] getServletMappings()
  {
    returnnewString[] { "/" };
  }
}
```

Step 3: Create the controller into package **com.ignou.mvcapp.controller**

AppController.java

```
packagecom.ignou.mvcapp.controller;
@Controller
@RequestMapping("/")
publicclassAppController
{

  @Autowired
  StudentServicestudentService;

  @Autowired
  MessageSourcemessageSource;

  /**
   * This Method will list All existing Students
   */
  @RequestMapping(value = { "/", "/list" }, method = RequestMethod.GET)
  publicModelAndViewwelcome(ModelMapmodel)
  {
    ModelAndViewmv = newModelAndView();
    List<Student>list = studentService.listAllStudents();
    model.addAttribute("allstudents", list);
    mv.setViewName("allStudents");
    returnmv;
  }

  /**
   * This Method will provide way to Add New Student
   */
  @RequestMapping(value = { "/new" }, method = RequestMethod.GET)
  public String newStudentForm(ModelMapmodel)
  {
    Student newStudent = newStudent();
    model.addAttribute("student", newStudent);
    model.addAttribute("edit", false);
    return"addStudentForm";
  }

  @RequestMapping(value = { "/new" }, method = RequestMethod.POST)
  publicModelAndViewsaveStudent(@Valid Student student, BindingResultresult, ModelMapmodel)
  {
    ModelAndViewmv = newModelAndView();
    Integer grade = student.getGrade();
    if (grade == null)
    {
      model.addAttribute("student", student);
      model.addAttribute("error", "Grade must be Numeric.");
      mv.setViewName("addStudentForm");
    }
    else
    {
      studentService.saveStudent(student);
      List<Student>list = studentService.listAllStudents();
      model.addAttribute("allstudents", list);
      model.addAttribute("success", "Student " + student.getName() + " added successfully.");
      mv.setViewName("allStudents");
    }
    returnmv;
  }

  /**
   * This Method will provide the way to update an existing Student
   *
   * @param id
   * @param model
   * @return
   */
  @RequestMapping(value = { "/edit-{id}" }, method = RequestMethod.GET)
  public String editStudent(@PathVariable Long id, ModelMapmodel)
  {
    Student student = studentService.getStudent(id);
    model.addAttribute("student", student);
    model.addAttribute("edit", true);
    return"addStudentForm";
  }

  /**
   * This method will be called on form submission, handling POST request for
   * updating Student in database. It also validates the user input
   *
   * @param Student
   * @param result
   * @param model
   * @param id
   * @return
   */
  @RequestMapping(value = { "/edit-{id}" }, method = RequestMethod.POST)
  publicModelAndViewupdateStudent(@Valid Student student, BindingResultresult, ModelMapmodel,
  @PathVariable Long id)
  {
    ModelAndViewmv = newModelAndView();
    Integer grade = student.getGrade();
    if (grade == null)
    {
      model.addAttribute("student", student);
      model.addAttribute("edit", true);
      model.addAttribute("error", "Grade must be Numeric.");
      mv.setViewName("addStudentForm");
    }
    else
    {
      studentService.update(id, student);
      List<Student>list = studentService.listAllStudents();
      model.addAttribute("allstudents", list);
      model.addAttribute("success", "Student " + student.getName() + " updated successfully.");
      mv.setViewName("allStudents");
    }
    returnmv;
  }

  /**
   * This method will Delete a Student by Id
   *
   * @param id
   * @return
   */
  @RequestMapping(value = { "/delete-{id}" }, method = RequestMethod.GET)
  publicModelAndViewdeleteStudent(@PathVariable Long id, ModelMapmodel)
  {
    ModelAndViewmv = newModelAndView();
    Student student = studentService.getStudent(id);
    studentService.delete(id);
    List<Student>list = studentService.listAllStudents();
    model.addAttribute("allstudents", list);
    model.addAttribute("success", "Student " + student.getName() + " deleted successfully.");
```

```
    mv.setViewName("allStudents");
    returnmv;
  }

}
```

Step 4: Create the service layer into package com.ignou.mvcapp.service package with the following classes.

StudentService.java

```
packagecom.ignou.mvcapp.service;
publicinterfaceStudentService
{
    Student getStudent(Long id);
    Long saveStudent(Student st);
    List<Student>listAllStudents();
    voidupdate(Long id, Student st);
    voiddelete(Long id);
    booleanisStudentUnique(Long id);
}
```

StudentServiceImpl.java

```
packagecom.ignou.mvcapp.service;
@Service("studentService")
publicclassStudentServiceImplimplementsStudentService
{

  @Autowired
  privateStudentDaostudentDao;

  @Override
  public Student getStudent(Long id)
  {
    returnstudentDao.getStudent(id);
  }

  @Override
  public Long saveStudent(Student st)
  {
    returnstudentDao.saveStudent(st);
  }

  @Override
  public List<Student>listAllStudents()
  {
    returnstudentDao.listAllStudents();
  }

  @Override
  publicvoidupdate(Long id, Student st)
  {
    Student stEntity = studentDao.getStudent(id);
    if (stEntity != null)
    {
      stEntity.setFirstName(st.getFirstName());
      stEntity.setLastName(st.getLastName());
      stEntity.setGrade(st.getGrade());
      studentDao.updateStudent(stEntity);
    }
  }

  @Override
  publicvoiddelete(Long id)
  {
    Student stEntity = studentDao.getStudent(id);
    if (stEntity != null)
    {
      studentDao.deleteStudent(stEntity);
    }
  }
  @Override
  publicbooleanisStudentUnique(Long id)
  {
    Student student = studentDao.getStudent(id);
    return (student == null || (id != null& !id.equals(student.getId())));
  }

}
```

Step 5: Create the model into package **com.ignou.mvcapp.model**.

```
packagecom.ignou.mvcapp.model;
@Entity
@Table(name = "students")
publicclass Student
{
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  privateLongid;

  @Column(name = "first_name", nullable = false)
  private String firstName;

  @Column(name = "last_name", nullable = false)
  private String lastName;

  @Column(name = "grade", nullable = false)
  private Integer grade;

  public Long getId()
  {
    returnid;
  }

  publicvoidsetId(Long id)
  {
    this.id = id;
  }

  public String getFirstName()
  {
    returnfirstName;
  }

  publicvoidsetFirstName(String firstName)
  {
    this.firstName = firstName;
  }

  public String getLastName()
  {
    returnlastName;
  }

  publicvoidsetLastName(String lastName)
  {
    this.lastName = lastName;
  }

  public Integer getGrade()
  {
    returngrade;
  }

  publicvoidsetGrade(Integer grade)
  {
    this.grade = grade;
  }

  public String getName()
  {
    returnthis.firstName + " " + this.lastName;
  }

}
```

Step 6: Create the DAO layer into the package com.ignou.mvcapp.dao. DAO layer is responsible for interacting with the database.

StudentDao.java

```
packagecom.ignou.mvcapp.model;

publicinterfaceStudentDao
{

        Student getStudent(Long id);

        Long saveStudent(Student st);

        List<Student>listAllStudents();

        voidupdateStudent(Student st);

        voiddeleteStudent(Student st);

}
```

StudentDaoImpl.java

```
packagecom.ignou.mvcapp.model;

@Repository("studentDao")
@Transactional
publicclassStudentDaoImplimplementsStudentDao
{
  @Autowired
  privateSessionFactorysessionFactory;

  @Override
  public Student getStudent(Long id)
  {
    Session session = sessionFactory.getCurrentSession();
    Student s  =session.get(Student.class, id);
    returns;
  }

  @Override
  public Long saveStudent(Student st)
  {
    Session session = sessionFactory.getCurrentSession();
    session.save(st);
    returnst.getId();
  }

  @Override
  public List<Student>listAllStudents()
  {
    Session session = sessionFactory.getCurrentSession();
    CriteriaBuildercb = session.getCriteriaBuilder();
    CriteriaQuery<Student>cq = cb.createQuery(Student.class);
    Root<Student>root = cq.from(Student.class);
    cq.select(root);
    Query query = session.createQuery(cq);
    @SuppressWarnings("unchecked")
    List<Student>students = query.getResultList();
    returnstudents;
  }

  @Override
  publicvoidupdateStudent(Student st)
  {
    Session session = sessionFactory.getCurrentSession();
    session.update(st);
  }

  @Override
  publicvoiddeleteStudent(Student st)
  {
    Session session = sessionFactory.getCurrentSession();
    session.delete(st);
  }

}
```

Step 7: Create the hibernate properties into src/main/resources/application.properties file. Change the username and password for the Mysql installed into your environment.

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/schoolDB
jdbc.username = root
jdbc.password = root

hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
hibernate.format_sql = true

hibernate.hbm2ddl.auto = update
```

Step 8: Create the views (JSP) inside src/main/webapp/WEB-INF/views.

addStudentForm.jsp

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<%@taglibprefix="form"uri="http://www.springframework.org/tags/form"%>
<%@taglibprefix="c"uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<head>
<metahttp-equiv="Content-Type"content="text/html; charset=ISO-8859-1">
<title>Add New Student Form</title>

<link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
        rel="stylesheet">
<scriptsrc="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"></
script>
<styletype="text/css">
<
style>.error
{
color: #ff0000;
}
</style>

</head>

<body>
        <navclass="navbar navbar-light bg-light">
                <divclass="container">
                        <divclass="navbar-brand mx-auto text-success"href="#">
                                <h1>Student Management System (SMS)</h1>
                        </div>
                </div>
        </nav>
        <divclass="container">
                <divclass="text-secondary text-center">
                <h2>Student Form</h2>

                </div>

                <c:iftest="${not empty error}">
                        <divclass="alert alert-danger alert-dismissible fade show"
                                role="alert">${error}
                                <buttontype="button"class="btn-close"data-bs-dismiss="alert"
                                aria-label="Close"></button>
                        </div>
                </c:if>
                <hr/>
                <form:formmethod="POST"modelAttribute="student"
```

```
                                        cssClass="form-horizontal">
                                        <form:inputtype="hidden"path="id"id="id"/>

                                        <divclass="form-group">
                                                <labelfor="firstName">First Name</label>
                                <form:inputpath="firstName"id="firstName"cssClass="form-control"/>
                                        </div>

                                        <divclass="form-group">
                                                <labelfor="lastName">Last Name</label>
                                        <form:inputpath="lastName"id="lastName"cssClass="form-control"/>
                                        </div>

                                        <divclass="form-group">
                                                <labelfor="grade">Grade</label>
                                        <form:inputpath="grade"id="grade"cssClass="form-control"/>
                                        </div>
                                        <hr/>
                                        <c:choose>
                                                <c:whentest="${edit}">
                                                        <buttontype="submit"class="btnbtn-
primary">Update</button>
                                                </c:when>
                                                <c:otherwise>
                                                        <buttontype="submit"class="btnbtn-
primary">Save</button>
                                                </c:otherwise>
                                        </c:choose>

                                        <aclass="btnbtn-secondary"href="<c:urlvalue='/list'/>">List
                                                of All Students</a>

                                </form:form>
                        </div>
</body>

</html>
```

allStudents.jsp

```
<%@taglibprefix="form"uri="http://www.springframework.org/tags/form"%>
<%@taglibprefix="c"uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPEhtml>
<htmllang="en">
<head>
<metahttp-equiv="Content-Type"content="text/html; charset=ISO-8859-1">
<title>All Students</title>

<link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
        rel="stylesheet">
<scriptsrc="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"></
script>
<linkhref="<c:urlvalue="/css/custom.css"/>"rel="stylesheet">

</head>
<body>
        <navclass="navbar navbar-light bg-light">
                <divclass="container">
                        <divclass="navbar-brand mx-auto text-success"href="#">
                        <h1>Student Management System (SMS)</h1>
                        </div>
                </div>
        </nav>
        <divclass="container">
                <divclass="text-secondary text-center">
                        <h2>Registered Students</h2>
                </div>

                <c:iftest="${not empty success}">
                        <divclass="alert alert-success alert-dismissible fade show"
                                role="alert">${success}
                                <buttontype="button"class="btn-close"data-bs-
dismiss="alert"
                                aria-label="Close"></button>
```

```
                    </div>
            </c:if>

            <hr/>
            <tableclass="table">
                    <theadclass="thead-dark">
                            <tr>
                                    <td>Student ID</td>
                                    <td>Student Name</td>
                                    <td>Grade</td>
                                    <td>Modify</td>
                                    <td>Delete</td>
                            </tr>
                            <c:forEachitems="${allstudents}"var="student">
                                    <tr>
<td>${student.id}</td><td>${student.firstName}&nbsp;${student.lastName}</td>
<td>${student.grade}</td>
<td><ahref="<c:urlvalue='/edit-${student.id}'/>"class="btnbtn-success">Modify</a></td>
<td><ahref="<c:urlvalue='/delete-${student.id}' />"class="btnbtn-danger">Delete</a></td>
                                    </tr>
</c:forEach>
            </table>
            <hr/>
            <divclass="form-group">
                    <aclass="btnbtn-primary"href="<c:urlvalue='/new'/>">Add
                            New Student</a>
            </div>
        </div>
</body>
</html>
```

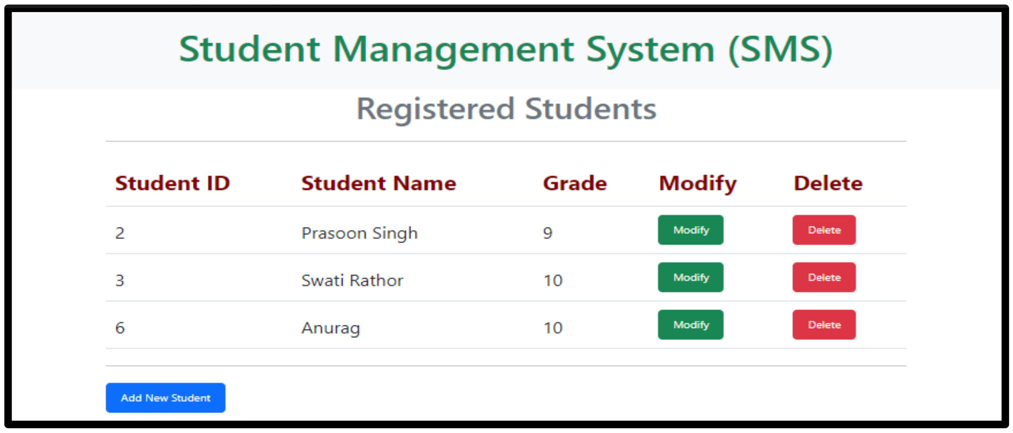
Step 9: Create the custom css inside folder src/main/webapp/resources/static/css.

custom.css

```
.btn{font-size:10px }

tr:first-child{font-size:20px; color:maroon; font-weight:bold;}
```

Step 10: Build the project using maven with goal clean install and then deploy the generated war file inside the target directory to the external tomcat. You can directly execute the project into the eclipse by using the Run on Server option. Once the application is up and running, access the URL http://localhost:8080/spring-mvc-crud-app/ and perform the CRUD operation provided by the application.

Output:
Home screen renders all existing students by executing read operation into the database.


Student Management System (SMS)

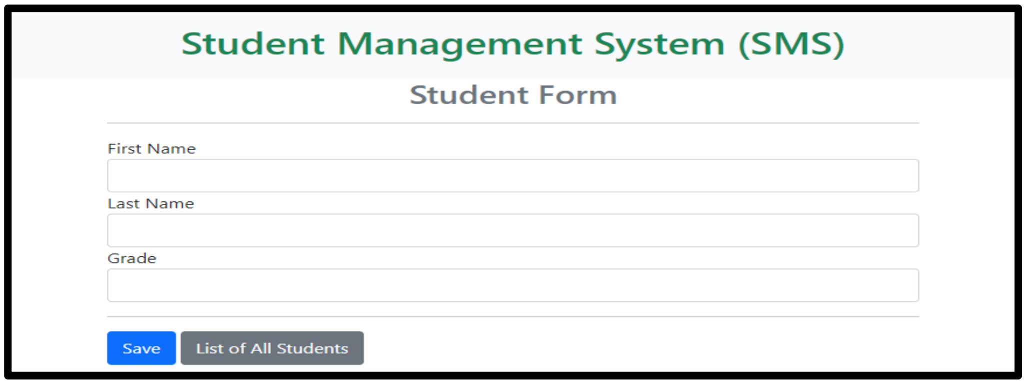
Registered Students

Student ID	Student Name	Grade	Modify	Delete
2	Prasoon Singh	9	Modify	Delete
3	Swati Rathor	10	Modify	Delete
6	Anurag	10	Modify	Delete

Add New Student

Figure 8.4: CRUD Application Home Screen

Click on the ***Add New Student*** button, and a form will be opened to create a new student record. Fill the form and click the save button. This operation performs create operation into the database, and a new student record will be created.



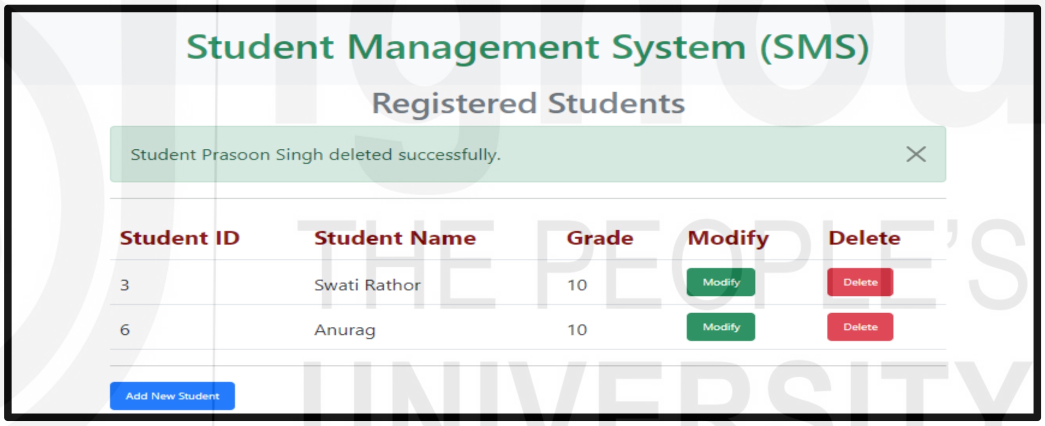
Student Management System (SMS)

Student Form

First Name	
Last Name	
Grade	

Figure 8.5: CRUD Application Create Form

A student record can be either modified or deleted from the home screen. The click of modifying button will render the student record in editable mode. Any attribute can be updated. This operation will perform an update operation into the database. While on click of the delete button will delete the student record from the database.



Student Management System (SMS)

Registered Students

Student Prasoon Singh deleted successfully.

Student ID	Student Name	Grade	Modify	Delete
3	Swati Rathor	10	Modify	Delete
6	Anurag	10	Modify	Delete

Figure 8.6: CRUD Application Delete Screen

☞Check Your Progress 3

1) Define CRUD within computer programming.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2) Define the mapping of the CRUD operations with SQL statements with suitable examples.

……………………………………………………………………………………

……………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

3) Define the mapping of CRUD in the Rest context.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

8.7 SUMMARY

The unit has briefly explained the Bootstrap CSS framework. Bootstrap is a feasible solution to tackle the challenges of making a responsive website as it evolves more and more. The Bootstrap makes things a lot easier to make the website responsive. Bootstrap CSS framework configuration into Spring MVC web application has been described in detail. Custom CSS can be used to override the Bootstrap styles in order to customize the Bootstrap and match the theme of a website. Hibernate is a very popular ORM framework to interact with databases, and Spring provides the required implementation to integrate Hibernate and Spring. The highlights of this unit are as follows.

- Bootstrap is a free, popular and open-source CSS framework directed at responsive, mobile-first front-end web development.
- There are two ways to include Bootstrap into JSP or HTML.
  - Bootstrap through Content Delivery Network (CDN)
  - Including by downloading files
- CSS allows you to customize the style of your website and apply stylistic decisions across its entirety.
- CSS allows you to separate the style from the structure of a web page.
- There are three main components for adding customized CSS language to the website. The components are **selectors, properties and values**.
- Spring supports bootstrapping the Hibernate SessionFactory in order to set up the database using Hibernate.
- Spring provides two key beans in order to support the hibernate integration.
  - **LocalSessionFactoryBean**
  - **PlatformTransactionManager**
- Within computer programming, the acronym CRUD stands for CREATE, READ, UPDATE and DELETE.
- In SQL context, CRUD corresponds to Insert, Select, Update and Delete, respectively.
- In REST context, CRUD corresponds to Rest web service POST, GET, PUT and DELETE, respectively.

8.7 SOLUTIONS/ ANSWER TO CHECK YOUR PROGRESS

Check Your Progress 1

1) Bootstrap is a free, popular and open-source CSS framework directed at responsive, mobile-first front-end web development. Unlike many web frameworks, it concerns itself with front-end development only. It contains HTML and CSS based design templates for various interface

components such as typography, forms, buttons, modals, image carousels, navigation etc, as well as optional JavaScript extensions. Bootstrap provides the ability to create responsive designs faster and easier. The various factors which make Bootstrap a popular choice for CSS framework are as follows.

(i) All popular web browser support
(ii) Responsive design
(iii) Uniform Solution to build an interface
(iv) Functional built-in components

2) The Bootstrap 5 is the most recent version of the Bootstrap framework. There are two ways to include Bootstrap into JSP or HTML.

(i) **Bootstrap through Content Delivery Network (CDN)**: Bootstrap CDN is an efficient and faster way to deliver the content from your website to the users. Bootstrap CDN speeds up the websites as CDN servers are intelligent enough to serve the resources based on proximity.
(ii) **Offline or Downloading files locally**: Another approach to include Bootstrap is to directly download the Bootstrap CSS and JS files locally to the project folder and then include the local copy into JSP/HTML.

3) For the development environment, offline usage of Bootstrap is suitable. For the production environment, CDN is better, and the following obvious reasons support to prefer CDN configuration for Bootstrap-

- CDN servers are intelligent enough to serve the resources based on proximity
- They are super-fast at serving the content
- The CDN network is spread across the world.
- If you think that your server will challenge the above, then go for it

Check Your Progress 2

1) CSS is an extension of HTML that outlines specific stylistic instructions to style websites. CSS allows you to customize the look of your website and apply stylistic decisions across its entirety. CSS allows you to separate the style from the structure of a web page. CSS is used to specify the color of a heading or what font your content should be written in. To keep the style uniformly across the pages, a separate CSS file with any name e.g. custom.css can be created and custom styles can be added to this file. The custom CSScan be included at the top of each webpage inside the <head> tag using the below instruction.

```
<link rel="stylesheet" type="text/css" href="css/custom.css">
```

2) The below style instructions change all of the text within <p> tags on a page to red.

```
<head>
<style type="text/css">
  p
  {
     color: red;
  }
</style>
</head>
<body>
<p>
 This is Red Color Text.
```

```
</p>
<p>
This is another paragraph with red color.
</p>
</body>
```

3) Spring supports bootstrapping the Hibernate SessionFactory in order to set up the database using Hibernate. The database setup can be done with a few lines of Java code or XML configuration. Spring provides two key beans, in order to support the hibernate integration, available in the org.springframework.orm.hibernate5 package:
   - **LocalSessionFactoryBean**: creates a Hibernate's **SessionFactory** ,which is injected into Hibernate-based DAO classes.
   - **PlatformTransactionManager**: provides transaction support code for a **SessionFactory**. Programmers can use **@Transactional** annotation in DAO methods to avoid writing boiler-plate transaction code explicitly.

4) The Java configuration to integrate Hibernate with Spring is as follows.

```
packagecom.ignou.mvcapp;

@Configuration
@EnableTransactionManagement
@PropertySource(value = { "classpath:application.properties"
})
publicclassHibernateConfig
{

  @Autowired
  private Environment environment;

  @Bean
  publicLocalSessionFactoryBeansessionFactory()
  {
    LocalSessionFactoryBeansessionFactory =
newLocalSessionFactoryBean();
    sessionFactory.setDataSource(dataSource());
    sessionFactory.setPackagesToScan(newString[] {
"com.ignou.mvcapp.model" });
    sessionFactory.setHibernateProperties(hibernateProperties());
    returnsessionFactory;
  }

  @Bean
  publicDataSourcedataSource()
  {
    DriverManagerDataSourcedataSource =
newDriverManagerDataSource();
    // set datasource properties
    returndataSource;
  }

  private Properties hibernateProperties()
  {
    Properties properties = newProperties();
    //set Hibernate properties
    returnproperties;
  }
  @Bean
  publicPlatformTransactionManagergetTransactionManager()
  {
    HibernateTransactionManagerhtm =
newHibernateTransactionManager();
    htm.setSessionFactory(sessionFactory().getObject());
    return htm;
  }
}
```

Check Your Progress 3

1) Within computer programming, the acronym CRUD stands for CREATE, READ, UPDATE and DELETE. The CRUD paradigm is common in constructing web applications. While constructing the APIs, the model should provide four basic types of functionality such as Create, Read, Update and Delete the resources. CRUD operations are also often used with SQL.
It can also describe user-interface conventions that allow viewing, searching and modifying information through computer-based forms and reports. Most applications have some form of CRUD functionality. When programmers provide interactions with this database (often through stored procedures), the steps of CRUD, for a Student Management System, are carried out as follows:
   - Create: A new student is entered into the database.
   - Read: The student's information is displayed to the users.
   - Update: Already existing student's attributes are being updated with new values.
   - Delete: If the student is not part of the institute, the student can be deleted from the records.
2) CRUD to database SQL operation mapping with examples are as follows.

Function	**SQL Statement**	**Example**
C(reate)	Insert	Insert into student (name,grade) values('Prasoon',10);
R(ead)	Select	Select * from student
U(pdate)	Update	Update student set grade=10 where id=1;
D(elete)	Delete	Delete from student where id=1;

3) In REST context, CRUD corresponds to Rest web service POST, GET, PUT and DELETE respectively. The following table maps CRUD to REST with an example.

Function	**Rest web service**	**Example**
C(reate)	POST	POST http://teststudent.com/students/
R(ead)	GET	GET http://teststudent.com/students/
U(pdate)	PUT	PUT http://teststudent.com/students/1
D(elete)	DELETE	DELETE http://teststudent.com/students/1

8.9 REFERENCES/FURTHER READING

… Craig Walls, "Spring Boot in action" Manning Publications, 2016. (https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf)

- Christian Bauer, Gavin King, and Gary Gregory, "Java Persistence with Hibernate",Manning Publications, 2015.
- Ethan Marcotte, "Responsive Web Design", Jeffrey Zeldman Publication, 2011.(http://nadin.miem.edu.ru/images_2015/responsive-web-design-2nd-edition.pdf)
- Tomcy John, "Hands-On Spring Security 5 for Reactive Applications",Packt Publishing,2018.
- https://www.creative-tim.com/blog/web-design/add-bootstrap-html-guide/
- https://getbootstrap.com/docs/5.0/getting-started/introduction/
- https://getbootstrap.com/docs/5.0/examples/
- https://wordpress.com/go/web-design/what-is-css/
- https://getbootstrap.com/docs/5.0/customize/overview/
- https://www.baeldung.com/hibernate-5-spring
- https://stackoverflow.com/questions/22693452/bootstrap-with-spring-mvc/22765275
- https://www.quora.com/Is-it-better-to-use-Bootstrap-offline-or-just-include-a-link-to-the-CDN