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.

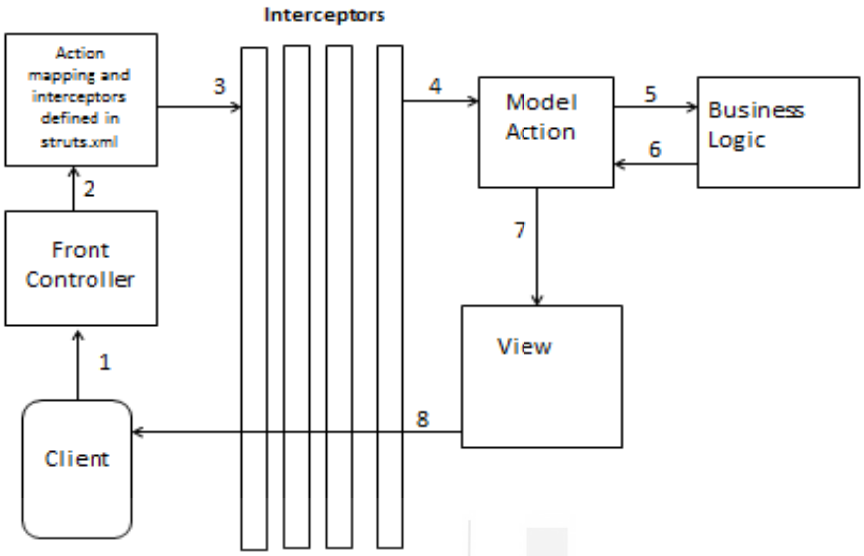

Figure 5.1 Strut 2 flow

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:

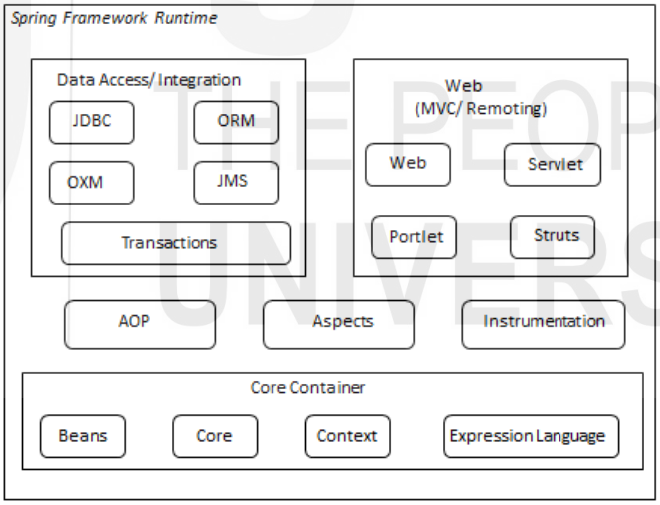

Figure 5.2: Spring Framework

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.

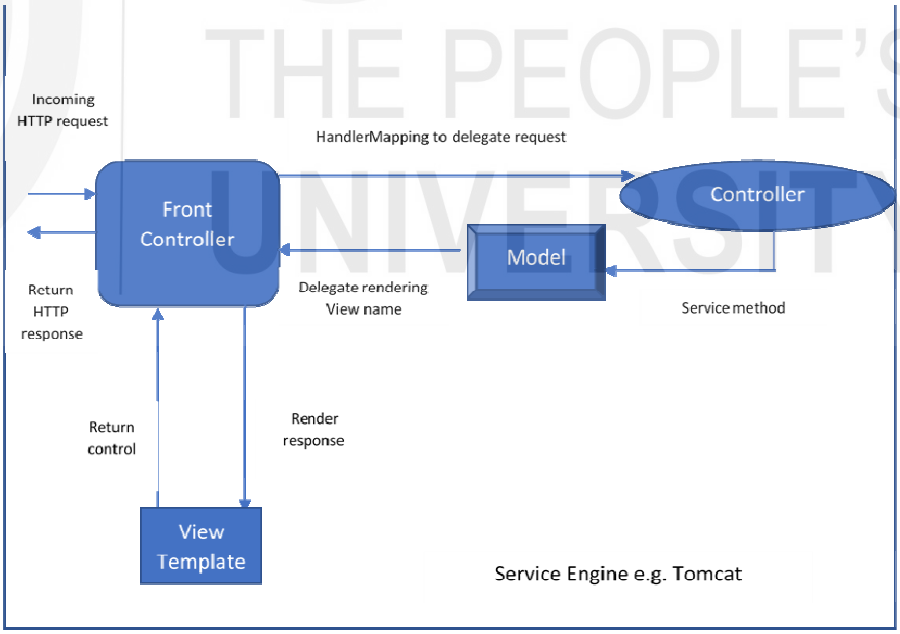

Figure 5.3: Spring Web MVC *DispatcherServlet* request processing workflow

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.

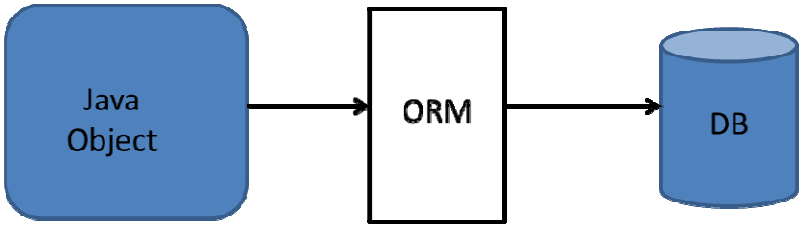

Figure 5.4: Java Persistence API specifications

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.

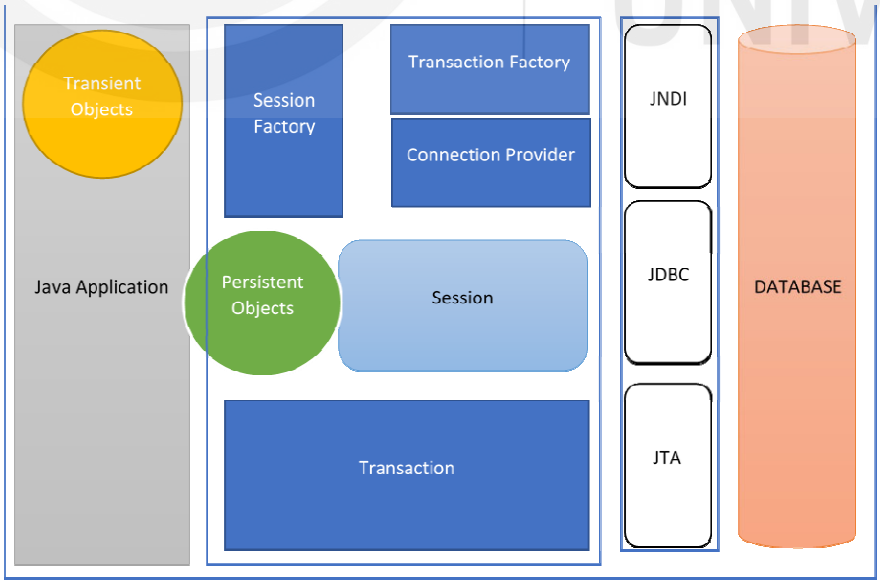

Figure 5.5: Hibernate architecture

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