B.Tech - Even Sem : Semester in Exam-II
Academic Year:2020-2021
19CS2107S - Enterprise Programming
Set No: 6
KEY

1.	**Explain about Spring IOC ?**	**4.5Marks**

Scheme of Evaluation:
Spring Definition: 2M
Spring IOC : 2.5M

Spring:-Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc.

IOC Container:

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets information from the XML file and works accordingly. The main 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:

1. **BeanFactory**
2. **ApplicationContext**

Difference between BeanFactory and the ApplicationContext
The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

Using BeanFactory

The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

1. Resource resource=**new** ClassPathResource("applicationContext.xml");
2. BeanFactory factory=**new** XmlBeanFactory(resource);

The constructor of XmlBeanFactory class receives the Resource object so we need to pass the resource object to create the object of BeanFactory.

Using ApplicationContext

The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface. We need to instantiate the ClassPathXmlApplicationContext class to use the ApplicationContext as given below:

```
1. ApplicationContext context =
2.     new ClassPathXmlApplicationContext("applicationContext.xml");
```

The constructor of ClassPathXmlApplicationContext class receives string, so we can pass the name of the xml file to create the instance of ApplicationContext.

(OR)

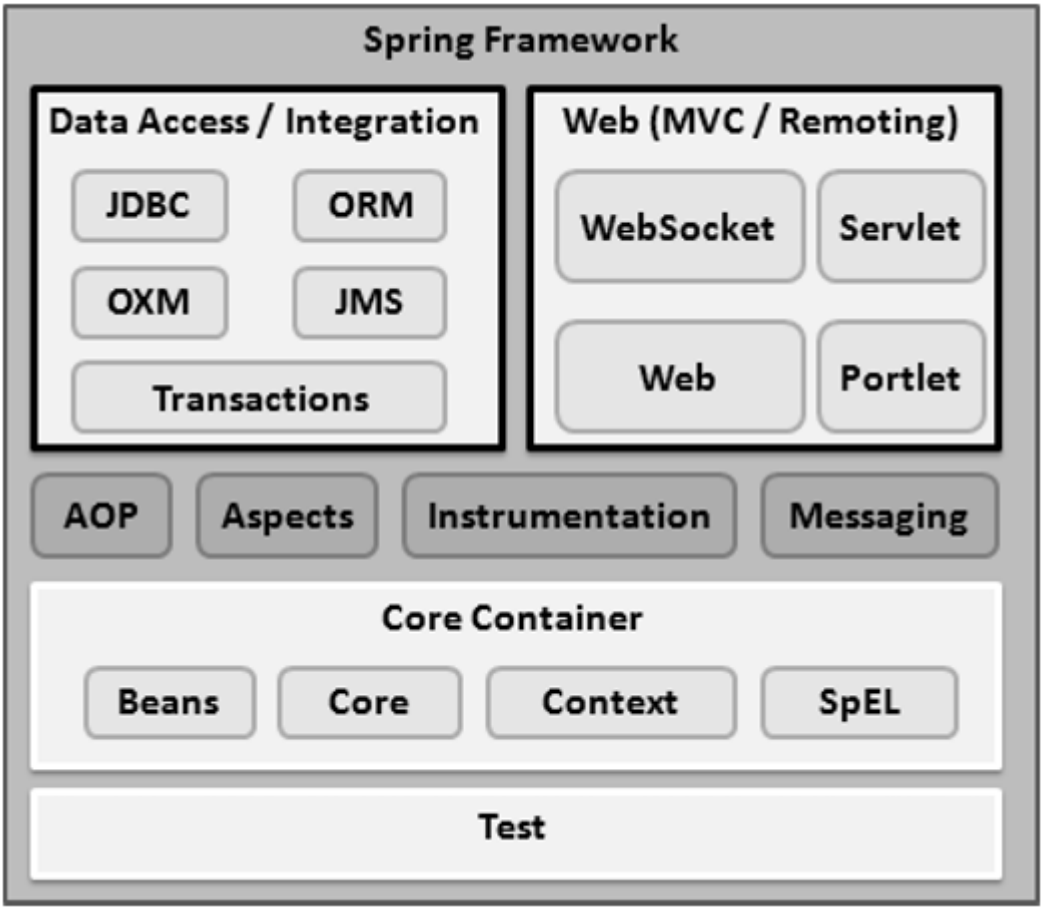
2.	**Outline the Spring Frame work architecture and explain about any two modules ?**	4.5Marks

Scheme of Evaluation:
Spring Architecture →2.5M
Modules Description →2M
(1M for Each module)

Answer:

Spring could potentially be a one-stop shop for all your enterprise applications. However, Spring is modular, allowing you to pick and choose which modules are applicable to you, without having to bring in the rest. The following section provides details about all the modules available in Spring Framework.

The Spring Framework provides about 20 modules which can be used based on an application requirement.



Core Container

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

- The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
- The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
- The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows −

- The **JDBC** module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
- The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
- The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- The Java Messaging Service **JMS** module contains features for producing and consuming messages.
- The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules the details of which are as follows −

- The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- The **Web-MVC** module contains Spring's Model-View-Controller (MVC) implementation for web applications.
- The **Web-Socket** module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
- The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

3.	**Develop a Spring Application which must contains Student POJO class with Student Id, Student Name and Student Marks as properties and displayinfo() as method which must display the values of the properties. Create Client Controller class to get the Object of Student class and call displayinfo() method.**	8 Marks

Scheme of Evaluation:
Pojo class →3M
Client class →3M
Pojo Object Creation →1M
Displayinfo() method call →1M

//Student bean

==========

```
package com.org.ep;

public class Student{
private int id;
private String name;
private int marks;

public Student(int id,String name,int marks) {this.id = id;
this.name=name;
this.marks=marks;}

void displayInfo(){
   System.out.println(id+" "+name+" "+marks);
}

}
```

//Test or Client Class

==========

```
package com.org.ep;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.*;

public class Test {
   public static void main(String[] args) {

      Resource r=new ClassPathResource("resources/applicationContext.xml");
      BeanFactory factory=new XmlBeanFactory(r);

      Student s=(Student)factory.getBean("stu");
      s.displayInfo();

   }
}
```

4.	**Develop a Spring application to implement Object as Non Primitive Type. Consider Employee and Address as POJO Classes?**	8 Marks

Scheme of Evaluation:
Employee POJO Class →2.5M
Address POJO Class →2.5M
Client Class →2M
Show() method call →1M

Address.java

```
package com.org.ep;
public class Address {
private String addressLine1,city,state,country;

public String getAddressLine1() {
	return addressLine1;
}

public void setAddressLine1(String addressLine1) {
	this.addressLine1 = addressLine1;
}

public String getCity() {
	return city;
}

public void setCity(String city) {
	this.city = city;
}

public String getState() {
	return state;
}

public void setState(String state) {
	this.state = state;
}

public String getCountry() {
	return country;
```

```
}

public void setCountry(String country) {
        this.country = country;
}

public String toString(){
        return addressLine1+" "+city+" "+state+" "+country;
}

}
```

Employee.java

```
public class Employee {
private int id;
private String name;
private Address address;//Aggregation

public Employee() {System.out.println("def cons");}

public Employee(int id, String name, Address address) {
   super();
   this.id = id;
   this.name = name;
   this.address = address;
}

void show(){
   System.out.println(id+" "+name);
   System.out.println(address.toString());
}

}
```

Applicationcontext.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:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="a1" class="Address">
<constructor-arg value="ghaziabad"></constructor-arg>
<constructor-arg value="UP"></constructor-arg>
<constructor-arg value="India"></constructor-arg>
</bean>
<bean id="e" class="Employee">
<constructor-arg value="12" type="int"></constructor-arg>
<constructor-arg value="Sonoo"></constructor-arg>
<constructor-arg>
<ref bean="a1"/>
</constructor-arg>
</bean>
</beans>
```

Test.java:

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.*;

public class Test {
   public static void main(String[] args) {

      Resource r=new ClassPathResource("applicationContext.xml");
      BeanFactory factory=new XmlBeanFactory(r);

      Employee s=(Employee)factory.getBean("e");
      s.show();

   }
}
```

5.A.	**Explain about Spring MVC Architecture with suitable diagram ?**	6 Marks

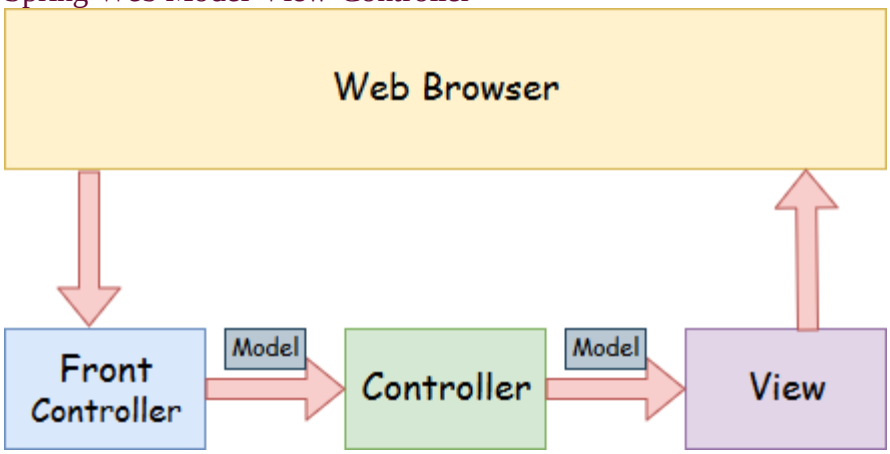
Scheme of Evaluation:
Spring MVC Diagram →2M
Description of MVC →2M
Dispatcher Servlet →2M

Spring MVC:

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

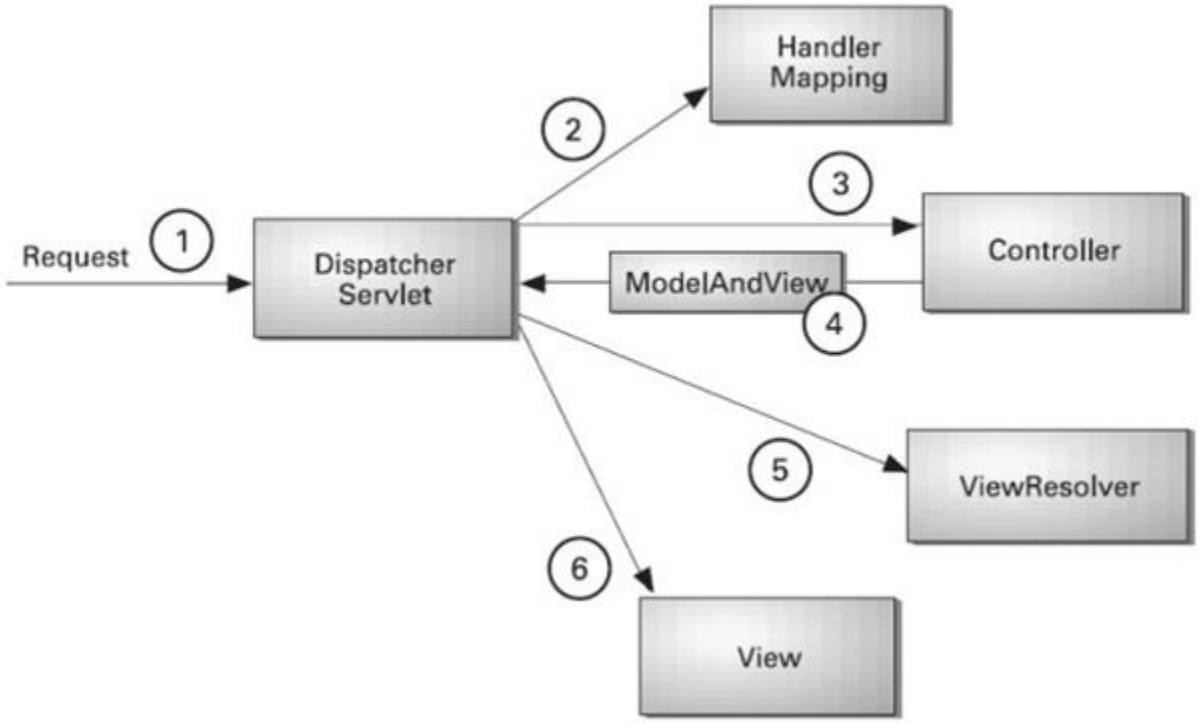
A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

Spring Web Model-View-Controller



- **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
- **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
- **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
- **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

Understanding the flow of Spring Web MVC



- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
- The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
- The controller returns an object of ModelAndView.
- The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

5.B.	**Build a Spring Application to implement Setter Dependency Injection using BeanFactory**.?	6.5Marks

Scheme of Evaluation:

POJO Class→2M
Client Class →2M

Pojo class Object creation by using BeanFactory in Client Class →2.5M

Employee.java

```
public class Employee {
private int id;
private String name;
private String city;

public int getId() {
   return id;
}
public void setId(int id) {
   this.id = id;
}
public String getName() {
   return name;
}
public void setName(String name) {
   this.name = name;
}

public String getCity() {
   return city;
}
public void setCity(String city) {
   this.city = city;
}
void display(){
   System.out.println(id+" "+name+" "+city);
}

}
```

applicationcontext.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:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="obj" class="Employee">
<property name="id" value=”5164”/>
<property name="name" value=”harika”/>
<property name="city" value=”vijayawada”/>
</bean>
</beans>
```

Test.java

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.*;

public class Test {
   public static void main(String[] args) {

      Resource r=new ClassPathResource("applicationContext.xml");
      BeanFactory factory=new XmlBeanFactory(r);

      Employee e=(Employee)factory.getBean("obj");
      s.display();

   }
}
```

6.A.	**Describe the steps to create Spring with JDBC application to perform CRUD Operations and explain JDBCTemplate class?**	6 Marks

Scheme of Evaluation:
Steps to create Spring With JDBC App →4M
JDBCTemplate Class description →2.5M

JDBCTemplate is a powerful mechanism to connect to SQL DB and to execute SQL Queries
It internally uses JDBC API
By using this we can avoid problems in JDBC API

Problems with JDBC API:
1.connection code,statements creation,closing the connection
2.Handling the exceptions for writing the database logic
3. To handle transaction management
4. Due to repetition of same code when migrate from db to db

Advantages:
we have predefined methods in order to perform crud operations

int---update(String Query)-for static insert,update,delete
execute(String Query,object)-to execute DDL commands(dynamic Queries)
Query(String QUery,object)-for select QUery

In java we have 2 techniques to connect to DB
1. DriverManager.getConnection()
2.DriverManagerDataSource

```
class JdbcTemplate{
   DriverManagerDataSource dataSource;
    setter method
}

class EmployeeDao
{
     JdbcTemplate template;
      setter
```

}

in JdbcTemplate we are going to work with DataSource Concept

How to configure DataSource in applicationcontext.xml file?

```
<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
  <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
  <property name="userName" value="system" />
  <property name="password" value="manager" />
</bean>
```

How to configure JdbcTemplate in spring configuration file?

```
<bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="ds"/>
</property>
```

How to configure our own DAO class in congiguration file

```
<bean id="eb" class="EmployeeDAO">
<property name="template" ref="jt" />
</bean>
```

Steps to create Spring With JDBC App

1. Create employee table in the database
2. Create a pojo class Employee with either setter injection or constructor injection.
3. Create a mapping file with the configuration specified above
4. Create a test client program in which we use BeanFactory/ApplicationContext
5. Create EmployeeDao class in which all the crud operations related methods should be implemented.

6.B.	**Outline the steps to create Spring with Hibernate application and explain HibernateTemplate class ?**	6.5Marks

Scheme of Evaluation:
Steps to create Spring With Hibernate App →4M
HibernateTemplate Class description →2.5M

HibernateTemplate--It is a class which overcomes the problems of creating configuration,sessionfactory,session,transactions.

Hibernate components:
1.POJO class
2. Test client
3. Mapping File
4. COnfiguration FIle

When we are implementing spring with hibernate no need to create hibernate configuration file

1.create a table employee with fields like empid,empName,empDesig,empSal(it is optional)
2.create a pojo class -Employee
3.create mapping file-Employee.hbm.xml
4.create EmployeeDAO class to perform CRUD operations
HibernateTemplate class as a dependency
||
LocalSessionFactoryBean---org.springframework.orm.hibernate3 package
mapping resources,hibernate properties(hbm2ddl,show_sql,dialect)
||
BasicDataSource--driver classname,url,username,password

applicationContext.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:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">


   <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
      <property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"></property>
      <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"></property>
      <property name="username" value="system"></property>
      <property name="password" value="manager"></property>
   </bean>

   <bean id="mysessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
      <property name="dataSource" ref="dataSource"></property>

      <property name="mappingResources">
      <list>
      <value>employee.hbm.xml</value>
      </list>
      </property>

      <property name="hibernateProperties">
         <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.OracleDialect</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop key="hibernate.show_sql">true</prop>
         </props>
      </property>
   </bean>
   <bean id="template" class="org.springframework.orm.hibernate3.HibernateTemplate">
   <property name="sessionFactory" ref="mysessionFactory"></property>
   </bean>
   <bean id="d" class="EmployeeDao">
   <property name="template" ref="template"></property>
   </bean>
   </beans>
```

5. you need to configure EmployeeDAO in configuration file
6. Test Client Application.

7.	**Discuss Spring and Spring Boot. Write the advantages of Spring Boot over spring?**	4.5Marks

Scheme of Evaluation:
Spring → 1M
SpringBoot →1M
Advantages of SpringBoot →2.5M

Spring:-

Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc.

Spring Boot:-

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the **RAD (*Rapid Application Development*)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.

In short, Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.

Advantages of Spring Boot

1. It creates stand-alone Spring applications that can be started using Java -jar.
2. It tests web applications easily with the help of different **Embedded HTTP servers such as Tomcat**, Jetty, etc. We don't need to deploy WAR files.
3. It provides opinionated 'starter' POMs to simplify our Maven configuration.
4. It provides production-ready features such as metrics, health checks, and externalized configuration.
5. **There is no requirement for XML configuration.**

8.	**Explain about Spring Boot MVC design pattern and write the flow of the application.**	4.5Marks

Scheme of Evaluation:

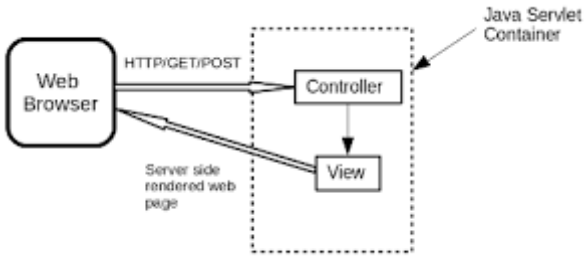
SpringBoot MVC Desription →2M

Description of Classes →2.5M

(like greeting Controller class and Client (SpringBootApplication) class

Spring MVC:-

The Model-View-Controller (MVC) architecture for building web applications is one of the oldest paradigms in computer science. The MVC architecture for web applications is supported by the Spring MVC component of the Spring framework.
An HTTP request is sent to the Java Servlet container (e.g., Tomcat, Jetty, etc...) The HTTP request is first intercepted by the controller written in Java. The controller returns a Java Server Page (JSP) which is rendered on the server and returned to the web browser as static HTML.
The diagram below shows how a Java MVC application interacts with the client side of the web application in a web browser.



greeting.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
   <title>Getting Started: Serving Web Content</title>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
   <p th:text="'Hello, ' + ${name} + '!'" />
</body>
</html>
```

GreetingController

```
package com.example.servingwebcontent;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class GreetingController {

	@GetMapping("/greeting")
	public String greeting(@RequestParam(name="name", required=false,
defaultValue="World") String name, Model model) {
		model.addAttribute("name", name);
		return "greeting";
	}

}
```

ServingWebContentApplication

```
package com.example.servingwebcontent;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ServingWebContentApplication {

   public static void main(String[] args) {
      SpringApplication.run(ServingWebContentApplication.class, args);
   }

}
```

Steps to create a SpringBoot Project(optional)

1. Navigate to https://start.spring.io. This service pulls in all the dependencies you need for an application and does most of the setup for you.
2. Choose either Gradle or Maven and the language you want to use. This guide assumes that you chose Java.
3. Click **Dependencies** and select **Spring Web**, **Thymeleaf**, and **Spring Boot DevTools**.
4. Click **Generate**.
5. Download the resulting ZIP file, which is an archive of a web application that is configured with your choices.

1.Create a Web Controller

2. Create a SpringBoot Client class (ServingWebContentApplication)

3.Run the Application

4.Test the Application

Now that the web site is running, visit http://localhost:8080/greeting, where you should see "Hello, World!"

Provide a `name` query string parameter by visiting http://localhost:8080/greeting?name=User. Notice how the message changes from "Hello, World!" to "Hello, User!":

This change demonstrates that the `@RequestParam` arrangement in `GreetingController` is working as expected. The `name` parameter has been given a default value of `World`, but it can be explicitly overridden through the query string

Flow of the application:

1.Request to http://localhost:8080/greeting,

2.it calls to greetingController class

3.SpringBoot Application Client class

Display the output

9.	**Outline the Steps to create SpringBoot application ?**	8Marks

Scheme of Evaluation
Steps to create a SpringBoot Application→4M
Classes Syntax →2M
Description →2M

Creating a Spring Boot Project
Following are the steps to create a simple Spring Boot Project.

Step 1: Open the Spring initializr https://start.spring.io.

Step 2: Provide the Group and Artifact name.
We have provided Group name com.javatpoint and Artifact spring-boot-example.

Step 3: Now click on the Generate button.

Creating Spring Boot Project
When we click on the Generate button, it starts packing the project in a .rar file and downloads the project.

Step 4: Extract the RAR file.

Step 5: Import the folder.

File -> Import -> Existing Maven Project -> Next -> Browse -> Select the project -> Finish

SpringBootExampleApplication.java

```
package com.org.ep.springbootexample;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringBootExampleApplication
{
public static void main(String[] args)
{
SpringApplication.run(SpringBootExampleApplication.class, args);
}
}
```

Step 6: Run the SpringBootExampleApplication.java file.

Right-click on the file -> Run As -> Java Applications

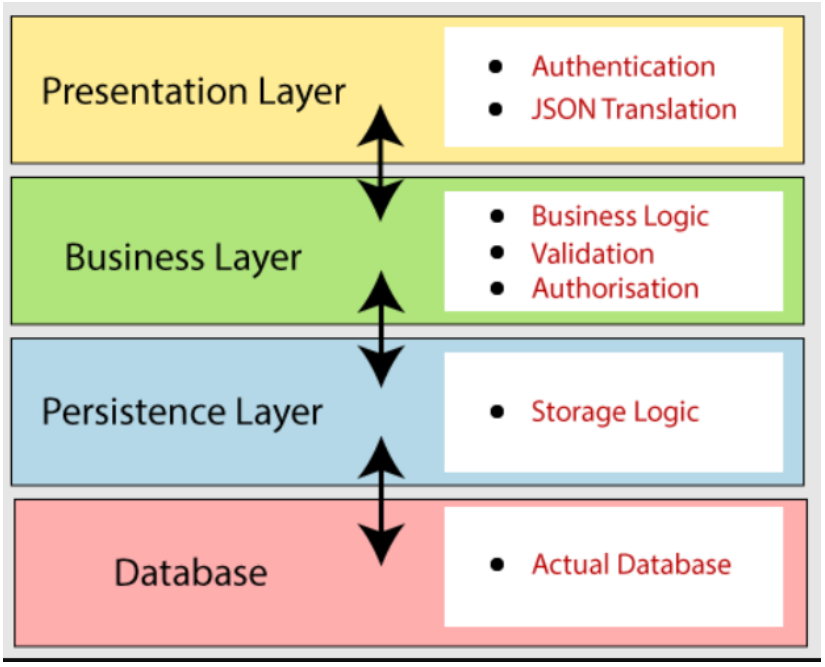
10.	**Demonstrate the Spring Boot Architecture and explain any two layers ?**	8Marks

Scheme of Evaluation **SpringBoot Architeture →4M** **Description of Any two layers →4M**

SpringBoot Architecture:

- 'Spring Boot is a module of the Spring Framework.
- It is used to create stand-alone, production-grade Spring Based Applications with minimum efforts.
- It is developed on top of the core Spring Framework.
- Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above (hierarchical structure) it.
- Before understanding the Spring Boot Architecture, we must know the different layers and classes present in it.
-

There are four layers in Spring Boot are as follows:



- **Presentation Layer:** The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of views i.e., frontend part.
- **Business Layer:** The business layer handles all the business logic. It consists of service classes and uses services provided by data access layers. It also performs authorization and validation.
- **Persistence Layer:** The persistence layer contains all the storage logic and translates business objects from and to database rows.
- **Database Layer:** In the database layer, CRUD (create, retrieve, update, delete) operations are performed.

11.A.	**Describe the steps to create SpringBoot Dependency Injection Application ?**	6Marks

Scheme of Evaluation:
Steps to create SpringBoot Application →3M
SpringBoot DI classes →3M

Steps to create SpringBoot Application

Creating a Spring Boot Project

Following are the steps to create a simple Spring Boot Project.

Step 1: Open the Spring initializr https://start.spring.io.

Step 2: Provide the Group and Artifact name.

We have provided Group name com.javatpoint and Artifact spring-boot-example.

Step 3: Now click on the Generate button.

Creating Spring Boot Project

When we click on the Generate button, it starts packing the project in a .rar file and downloads the project.

Step 4: Extract the RAR file.

Step 5: Import the folder.

File -> Import -> Existing Maven Project -> Next -> Browse -> Select the project -> Finish

SpringBoot DI Classes:-

Bean Class uses @Component

SPRINGBOOT CLIENT CLASS uses @SpringBootApplication annotations.

11.B.	**Build an application for SpringBoot Dependency Injection ?**	6.5Marks

Scheme of Evaluation:
Customer Class →3M
SPRINGBOOT CLIENT CLASS →2M
DI Syntax in SPRINGBOOT CLIENT CLASS →1.5 M

CUSTOMER CLASS

```
import org.springframework.stereotype.Component;
@Component
public class Customers {
private int custid;
private String custname;
private String coursename;

public int getCustid() {
return custid;
}
public void setCustid(int custid) {
this.custid = custid;
}
public String getCustname() {
return custname;
}
public void setCustname(String custname) {
this.custname = custname;
}
public String getCoursename() {
return coursename;
}
public void setCoursename(String coursename) {
this.coursename = coursename;
```

```
}
public void display()

{
System.out.println("Object Returned Successfully");

}
}
```

//SPRINGBOOT CLIENT CLASS

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringBootExampleApplication
{
public static void main(String[] args)
{
ConfigurableApplicationContext context =
SpringApplication.run(SpringBootExampleApplication.class, args);

//DEPENDENCY INJECTION IN SPRING BOOT
customers c = context.getBean(customers.class);
c. display();



}
}
```

12.	Answer the following Questions	12.5Marks
12.A.	**List out the Steps to create Springboot Application by using Spring Tool Suite (STS) IDE?**	6Marks

Scheme of Evaluation:

Steps to create STS IDE SpringBoot App →6M

Download and Install STS IDE

Spring Tool Suite (STS) IDE

Spring Tool Suite is an IDE to develop Spring applications. It is an Eclipse-based development environment. It provides a ready-to-use environment to implement, run, deploy, and debug the application. It validates our application and provides quick fixes for the applications.

Installing STS

Step 1: Download Spring Tool Suite from https://spring.io/tools3/sts/all. Click on the platform which you are using. In this tutorial, we are using the Windows platform.

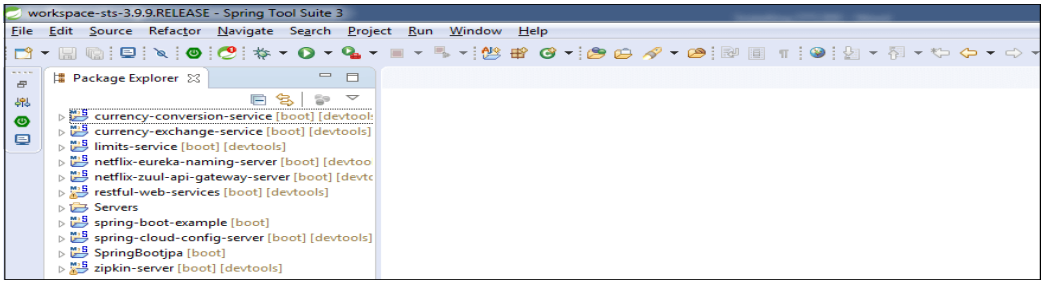
Step 2: Extract the **zip** file and install the STS.

sts-bundle -> sts-3.9.9.RELEASE -> Double-click on the **STS.exe**.

Step 3: Spring Tool Suite 3 Launcher dialog box appears on the screen. Click on the **Launch** button. You can change the Workspace if you want.

Step 4: It starts launching the STS.

The STS user interface looks like the following:



12.B.	**Develop an application for Spring Boot "Hello World! " Example ?**	6.5Marks

Scheme of Evaluation:

Steps to Creating a SpringBoot Project →3M

SpringBootExample Application →2.5M

Display message →1 M

Creating a Spring Boot Project

Following are the steps to create a simple Spring Boot Project.

Step 1: Open the Spring initializr https://start.spring.io.

Step 2: Provide the Group and Artifact name. We have provided Group name com.org.ep and Artifact spring-boot-example

Step 3: Now click on the Generate button.

When we click on the Generate button, it starts packing the project in a **.rar** file and downloads the project.

Step 4: Extract the **RAR** file.

Step 5: Import the folder.

File -> Import -> Existing Maven Project -> Next -> Browse -> Select the project -> Finish

SpringBootExampleApplication.java

```
package com.javatpoint.springbootexample;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringBootExampleApplication
{
public static void main(String[] args)
{
SpringApplication.run(SpringBootExampleApplication.class, args);
System.out.println("Hello World!");
}
}
```

NOTE: The Enterprise Programming Test-2 Key with Scheme of Evaluation.
The above sample code represents one of the solutions for the given problem. You may attempt the problem in a different way to obtain the solution.

COURSE COORDINATOR HOD-CSE

SIGNATURE