Rest controller

Student.

```
package com.example.demo24;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Student{
   @RequestMapping("/welcome")
   public String Greet(){
      return "welcome to sdm college";
   }

   @RequestMapping("/bye")
   public String Bye(){
      return "thank you for visiting SDM college";
   }

}
```

field dependency

student.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Student {
   @Autowired
   Teacher teacher;
   void msg()
   {
      teacher.Hello();
      System.out.println("Hello from student class");
   }
}
```

teacher.

```
package com.example.demo100;

import org.springframework.stereotype.Component;

@Component
public class Teacher {
   void Hello()
   {
      System.out.println("hello from teacher class");
   }
}
```

setter dependency.

```
public class Student{
   private Teacher teacher;
   @Autowired
   public void setTeacher(Teacher teacher)
   {
      this.teacher=teacher;
   }
   public void Greet(){
      teacher.Hello();
      System.out.println("Hello from Student class");
   }
}
```

construct.

```
public class Student {
   public Teacher teacher;

   public Student(Teacher teacher) {
      this.teacher = teacher;
   }

   public void Greet() {
      teacher.Hello();
      System.out.println("hello from student");
   }
}
```

1. **With an example Explain pom.xml file.**

- **POM stands for Project Object Model. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.**
- **The POM contains information about the project and various configuration detail used by Maven to build the project(s).**
- **POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal.**
- **Some of the configuration that can be specified in the POM are following**
- **project dependencies**
- **plugins**
- **goals**
- **build profiles**
- **project version**
- **developers**
- **mailing list**

Before creating a POM, we should first decide the project group (groupId), its name (artifactId) and its version as these attributes help in uniquely identifying the project in repository.

POM Example

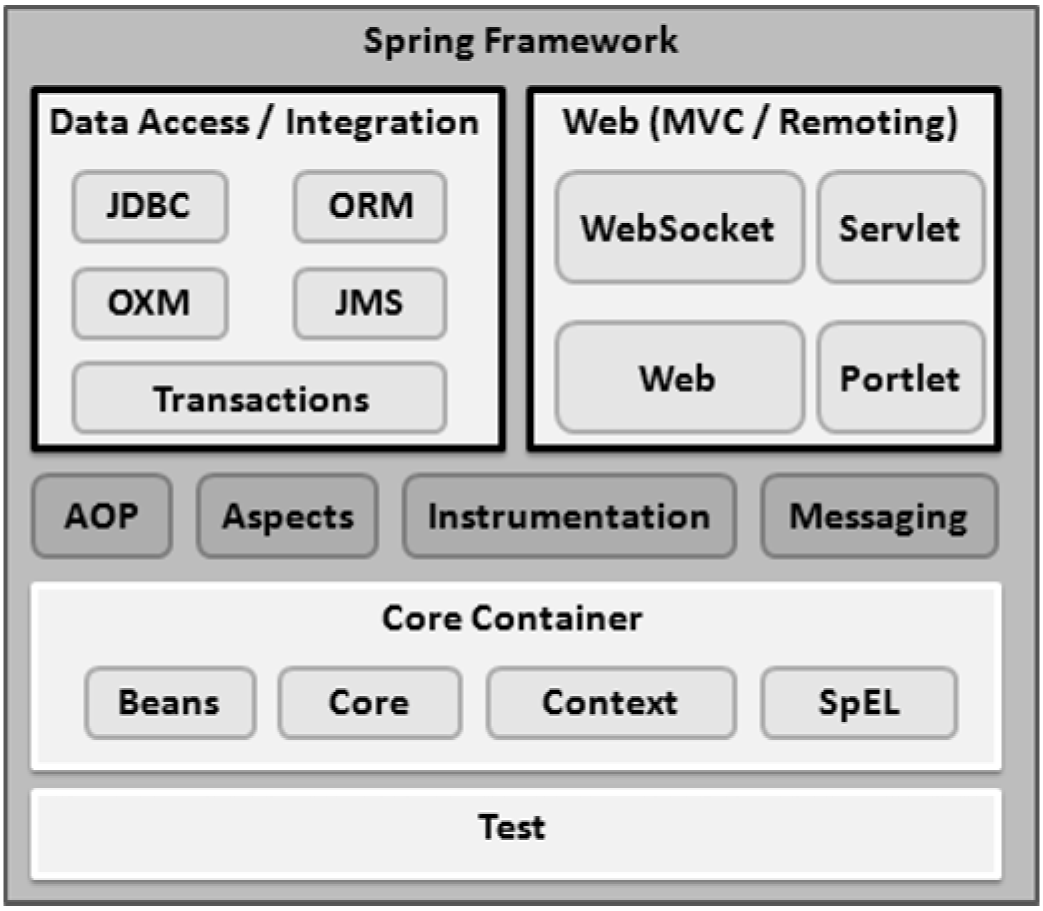
```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.companyname.project-group</groupId>
  <artifactId>project</artifactId>
  <version>1.0</version>
```

</project>

The **heart of a Maven project is the Project Object Model (POM) file, named `pom.xml`. It defines the project's configuration, dependencies, and build settings. This XML file is used to manage the project's lifecycle, including building, testing, packaging, and deploying.**

1. **Construct spring framework Architecture and explain the components.**



Core Container

The Core Container consists of the Core, Beans, Context, and Expression Language modules

- 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

- 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 **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.

Miscellaneous

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules

- The **AOP module** provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
- The **Aspects module** provides integration with AspectJ, which is again a powerful and mature AOP framework.
- The **Instrumentation module** provides class instrumentation support and class loader implementations to be used in certain application servers.
- The **Messaging module** provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.

The **Test module** supports the testing of Spring components with JUnit or TestNG frameworks.

1. **Explain The Key Components Of Spring framework**

Key components of Spring Framework

- Spring Core
- Spring AOP
- Spring Web MVC
- Spring DAO
- Spring ORM
- Spring context
- Spring Web flow

1.Compare spring and spring boot.

Compare Spring and SpringBoot

Spring	Spring Boot
Spring Framework is a widely used Java EE framework for building applications.	**Spring Boot Framework** is widely used to develop **REST APIs**.

It aims to simplify Java EE development that makes developers more productive.	It aims to shorten the code length and provide the easiest way to develop **Web Applications**.
The primary feature of the Spring Framework is **dependency injection**.	The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement.
It helps to make things simpler by allowing us to develop **loosely coupled** applications.	It helps to create a **stand-alone** application with less configuration.
The developer writes a lot of code (**boilerplate code**) to do the minimal task.	It **reduces** boilerplate code.
To test the Spring project, we need to set up the sever explicitly.	Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc.
It does not provide support for an in-memory database.	It offers several plugins for working with an embedded and **in-memory** database such as **H2**.
Developers manually define dependencies for the Spring project in **pom.xml**.	Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement.

1. **Explain different types of Dependency Injection**