

1. Write down the steps for creating a simple Spring Boot application and how we can implement the same.

## Steps for Creating a Simple Spring Boot Application

1. Set Up the Project

- Use **Spring Initializr** (<https://start.spring.io/>) or create a new Maven/Gradle project.
- Select dependencies like **Spring Web**, **Spring Boot DevTools**, and **Spring Boot Starter**.

2. Configure Application

- Ensure `application.properties` or `application.yml` is properly set up.

```
server.port=8080
spring.application.name=DemoApp
```

3. Create the Main Class

```
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

4. Create a REST Controller

```
@RestController
@RequestMapping("/api")
public class HelloController {
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, Spring Boot!";
    }
}
```

5. Run the Application

Use `mvn spring-boot:run` or run the `main` method in `DemoApplication.java`.

6. Access the API

Open a browser or Postman and visit `http://localhost:8080/api/hello`.

## 2. Explain the Spring MVC framework with its components.

Spring MVC is a framework used for developing web applications following the **Model-View-Controller (MVC)** pattern.

Components of Spring MVC:

1. **DispatcherServlet**
  - Acts as the front controller, routing requests to the appropriate handlers.
2. **Handler Mapping**
  - Maps incoming requests to the corresponding controllers.
3. **Controller**
  - Contains the business logic and handles client requests.

```
@Controller
public class MyController {
    @RequestMapping("/welcome")
    public String welcome(Model model) {
        model.addAttribute("message", "Welcome to Spring MVC");
        return "welcome";
    }
}
```

## 4. View Resolver

- Selects the appropriate view (e.g., JSP, Thymeleaf) to render the response.

## 5. Model and View

- Model holds data, and View presents the UI.

```
model.addAttribute("message", "Hello Spring MVC");
return "index"; // View (index.jsp or index.html)
```

## 6. View (JSP/Thymeleaf/HTML)

```
<h1>${message}</h1>
```

## Spring MVC Workflow:

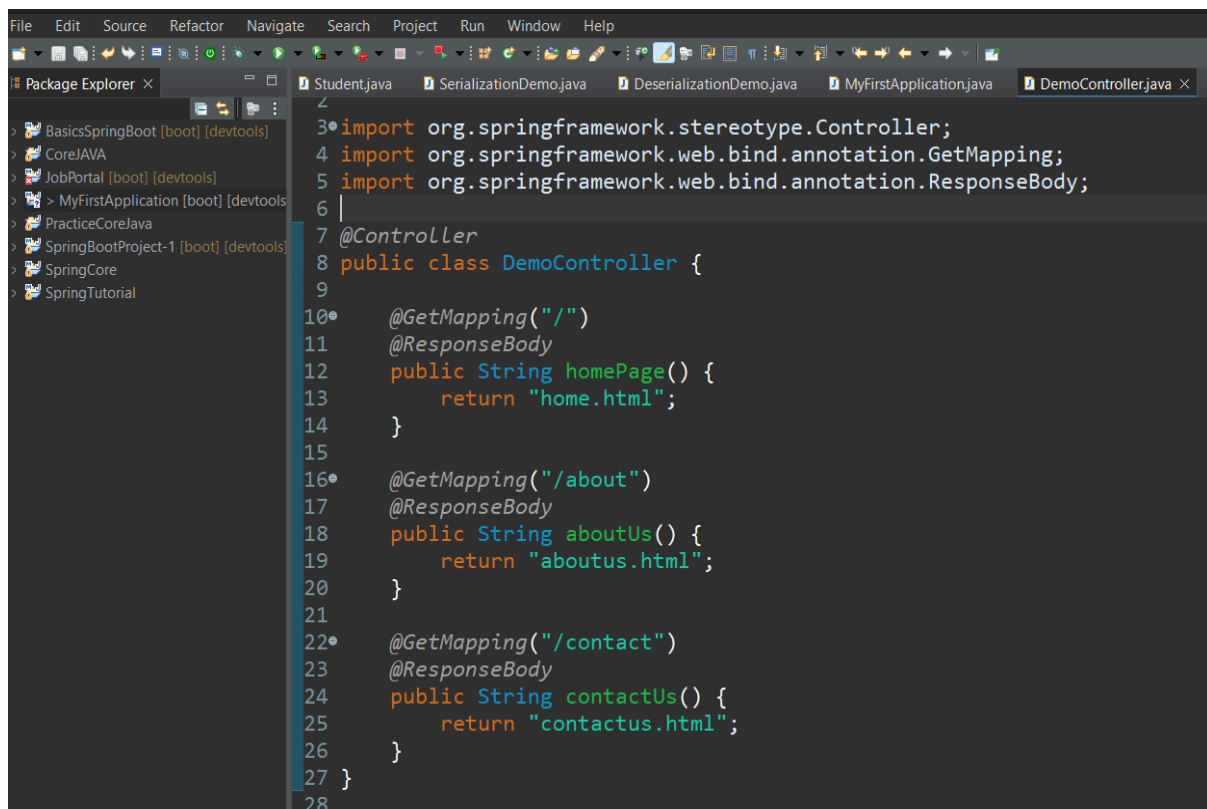
1. Client sends a request →
2. DispatcherServlet intercepts →
3. HandlerMapping finds the correct controller →
4. Controller processes the request and returns data →
5. ViewResolver selects the appropriate view →
6. Response is rendered and sent to the client.

Q.3 Create a Spring Boot application with **DemoController** having three methods

Application Name: **MyFirstApplication**

Method Name	HTML Page	URL Mapping
homePage	home.html	"/"
aboutUs	aboutus.html	"/about"
contactUs	contactus.html	"/contact"

Each Controller in the application must return only the page's name as content to be viewed by the User.



```
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer x Student.java SerializationDemo.java DeserializationDemo.java MyFirstApplication.java DemoController.java x
> BasicsSpringBoot [boot] [devtools]
> CoreJAVA
> JobPortal [boot] [devtools]
> MyFirstApplication [boot] [devtools]
> PracticeCoreJava
> SpringBootProject-1 [boot] [devtools]
> SpringCore
> SpringTutorial
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.ResponseBody;
6
7 @Controller
8 public class DemoController {
9
10     @GetMapping("/")
11     @ResponseBody
12     public String homePage() {
13         return "home.html";
14     }
15
16     @GetMapping("/about")
17     @ResponseBody
18     public String aboutUs() {
19         return "aboutus.html";
20     }
21
22     @GetMapping("/contact")
23     @ResponseBody
24     public String contactUs() {
25         return "contactus.html";
26     }
27 }
28
```

#### 4. Explain the @Controller and @RequestMapping annotations with examples.

##### @Controller:

Marks a class as a Spring MVC controller that can handle web requests. It enables the class to return views (like HTML pages) or data directly to the client.

##### Example:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

    // Handles HTTP GET requests for /home
    @RequestMapping("/home")
    public String home() {
        return "home"; // This refers to a view named "home" (e.g.,
home.html)
    }
}
```

##### @RequestMapping:

Used to map HTTP requests to handler methods in the controller. It can be applied at the class level (to define a base path) and method level (to specify endpoint paths).

##### Example:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/api") // Base path for all methods in this controller
public class ApiController {

    // Handles requests for /api/greeting
    @RequestMapping("/greeting")
    public String greeting() {
        return "greeting"; // Returns a view named "greeting"
    }
}
```

## 5. Explain the use of the Spring DevTools dependency.

### Spring DevTools Dependency

- **Purpose:**

The Spring DevTools dependency enhances the development experience by enabling features such as automatic application restarts, live reload, and configuration caching. This speeds up development by reducing the need for manual restarts when code changes are made.

- **Key Benefits:**

- **Automatic Restart:** Automatically restarts the application whenever code changes are detected.
- **Live Reload:** Integrates with browsers to refresh the page as soon as resources change.
- **Enhanced Debugging:** Provides additional debugging information and improved error messages.