Java Spring Boot – Important Points

1. What is Spring Boot?

- Spring Boot is a framework used to build **Java-based web applications and microservices**
- It is built on top of the **Spring Framework**
- It simplifies Spring application development

2. Features of Spring Boot

- **Auto Configuration** – Automatically configures Spring and third-party libraries
- **Standalone Applications** – Run using java -jar
- **Embedded Server** – Comes with Tomcat, Jetty, or Undertow
- **Production Ready** – Includes monitoring, health checks, and metrics
- **No XML Configuration** – Uses annotations and Java config

3. Advantages of Spring Boot

- Faster development
- Less boilerplate code
- Easy to deploy
- Easy integration with databases
- Microservices support

4. Spring Boot Architecture

- Presentation Layer (Controllers)
- Business Layer (Services)
- Persistence Layer (Repositories)
- Database Layer

5. Important Annotations

- @SpringBootApplication
- @RestController
- @Controller
- @RequestMapping

- @GetMapping, @PostMapping
- @Autowired
- @Service
- @Repository
- @Entity

6. Spring Boot Starters

- Predefined dependency descriptors
- Examples:
    - spring-boot-starter-web
    - spring-boot-starter-data-jpa
    - spring-boot-starter-security
    - spring-boot-starter-test

Spring Boot Dependencies

- Dependencies are **libraries** required for the application to work
- Managed using **Maven** or **Gradle**
- Defined in pom.xml (Maven)
- Spring Boot automatically manages **versions** of dependencies

Example (Maven Dependency)

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

---

5. Important Annotations (In Detail)

@SpringBootApplication

- Main annotation of Spring Boot

- Used on the **main class**
- Combination of:
    - @Configuration
    - @EnableAutoConfiguration
    - @ComponentScan
- Starts the Spring Boot application

Example:

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

@RestController

- Used to create **RESTful web services**
- Returns **JSON or XML data**
- Combination of:
    - @Controller + @ResponseBody

Example:

```
@RestController
public class DemoController {
  @GetMapping("/hello")
  public String hello() {
    return "Hello World";
  }
}
```

@Controller

- Used in **MVC applications**
- Returns **view pages** (HTML, JSP, Thymeleaf)
- Does NOT return JSON by default

Example:

```
@Controller
public class PageController {
   @RequestMapping("/home")
   public String home() {
      return "home";
   }
}
```

@RequestMapping

- Maps **HTTP requests** to controller methods
- Can be used at **class level or method level**
- Supports GET, POST, PUT, DELETE

Example:

```
@RequestMapping("/api")
public class ApiController {
   @RequestMapping("/users")
   public String users() {
      return "User List";
   }
}
```

@GetMapping & @PostMapping

- Shortcut annotations for HTTP methods

@GetMapping

- Used to **fetch data**

@PostMapping

- Used to **send data**

Example:

```
@GetMapping("/getData")
public String getData() {
   return "Data";
}


@PostMapping("/saveData")
public String saveData() {
   return "Saved";
}
```

@Autowired

- Used for **dependency injection**
- Automatically injects required beans
- Reduces object creation using new

Example:

```
@Autowired
private UserService userService;
```

@Service

- Used in **business logic layer**
- Marks a class as a **service component**
- Improves code readability and structure

Example:

```
@Service
public class UserService {
   public String getUser() {
      return "User";
   }
}
```

@Repository

- Used in **data access layer**
- Communicates with database
- Handles **exception translation**

Example:

```
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {
}
```

@Entity

- Used to map a class to a **database table**
- Part of **JPA**
- Each object represents a row in the table

Example:

```
@Entity
public class User {
   @Id
   @GeneratedValue
   private int id;
   private String name;
}
```

---

6. Spring Boot Starters (In Detail)

What are Spring Boot Starters?

- **Predefined dependency descriptors**
- Simplify dependency management
- One starter includes **multiple related libraries**

---

spring-boot-starter-web

Used for building **web & REST applications**

Includes:

- Spring MVC
- Embedded Tomcat
- Jackson (JSON processing)
- Validation APIs

Use case:

- REST APIs
- Web applications

---

spring-boot-starter-data-jpa

Used for **database operations**

Includes:

- Spring Data JPA
- Hibernate ORM
- JPA APIs

Use case:

- CRUD operations
- Database integration

---

spring-boot-starter-security

Used for **application security**

Includes:

- Spring Security
- Authentication & Authorization
- Password encoding

Use case:

- Login systems
- Role-based access

---

spring-boot-starter-test

Used for **testing applications**

Includes:

- JUnit
- Mockito
- Spring Test
- AssertJ

Use case:

- Unit testing
- Integration testing

Summary (Exam Ready Points)

- Spring Boot uses **starters** to simplify dependency management
- Annotations reduce configuration
- Auto-configuration makes development faster
- Supports MVC, REST, database, and security

7. Embedded Servers

- No need to install external Tomcat
- Default server is **Tomcat**
- Makes deployment simple

8. Configuration in Spring Boot

- application.properties
- application.yml
- Used to configure:
    - Database
    - Server port
    - Logging
    - Security

9. Database Integration

- Supports:
    - MySQL
    - PostgreSQL
    - Oracle
    - MongoDB
- Uses Spring Data JPA

10. RESTful Web Services

- Easy to create REST APIs
- Supports JSON and XML
- Uses @RestController

11. Spring Boot Security

- Provides authentication and authorization
- Integrates with Spring Security

12. Spring Boot Actuator

- Used for monitoring and management
- Provides endpoints like:

- /actuator/health
- /actuator/info
- /actuator/metrics

13. Microservices with Spring Boot

- Each service is independent
- Easy to scale
- Supports REST communication

14. DevTools

- Automatic restart
- Live reload
- Faster development

15. Deployment

- JAR deployment
- WAR deployment
- Cloud deployment (AWS, Azure, GCP)

Java Server Pages (JSP) – Important Points

1. What is JSP?

- JSP stands for **Java Server Pages**
- It is a **server-side technology**
- Used to create **dynamic web pages**
- Part of **Java EE**

---

2. Features of JSP

- Easy to develop dynamic pages
- Supports **HTML + Java code**
- Platform independent
- Faster than servlets (after compilation)
- Supports MVC architecture

---

3. JSP Architecture

- Client sends request to server
- JSP page is converted into **Servlet**
- Servlet is compiled and executed
- Response sent back to client

---

4. JSP Life Cycle

1. Translation (JSP → Servlet)
2. Compilation
3. Loading
4. Initialization (jspInit())
5. Request Processing (_jspService())
6. Destruction (jspDestroy())

---

5. JSP Tags (In Detail)

1. Scriptlet Tag

- Used to write Java code

<% int a = 10; %>

2. Expression Tag

- Used to display output

<%= a %>

3. Declaration Tag

- Used to declare variables/methods

<%! int x = 5; %>

---

6. JSP Directives

Directives provide instructions to the JSP container.

Page Directive

<%@ page language="java" %>

Include Directive

<%@ include file="header.jsp" %>

Taglib Directive

<%@ taglib uri="..." prefix="c" %>

---

7. JSP Implicit Objects

Automatically available objects in JSP:

- request
- response
- session
- application
- out

- config
- page
- pageContext
- exception

---

8. JSP Actions

Used to control JSP behavior.

Examples:

- <jsp:include>
- <jsp:forward>
- <jsp:useBean>
- <jsp:setProperty>
- <jsp:getProperty>

Example:

<jsp:include page="menu.jsp" />

---

9. JSP Expression Language (EL)

- Simplifies data access
- Avoids Java code in JSP

Example:

${user.name}

---

10. JSTL (JSP Standard Tag Library)

- Reduces Java code
- Makes JSP clean and readable

Common tags:

- <c:out>
- <c:if>

- <c:forEach>

Example:

```
<c:forEach var="u" items="${users}">
  ${u.name}
</c:forEach>
```

---

11. JSP with MVC Architecture

- **Model** → Java Beans
- **View** → JSP
- **Controller** → Servlet / Spring Controller

---

12. Advantages of JSP

- Easy to learn
- Good separation of view logic
- Reusable components
- Better readability than servlets

---

13. Disadvantages of JSP

- Mixing Java and HTML can be messy
- Slower initial load
- Not suitable for large applications alone

---

14. JSP vs Servlet

JSP	**Servlet**
View based	Controller based
HTML + Java	Pure Java
Easy UI design	Complex UI

---

15. JSP with Spring Boot

- JSP used as **view layer**
- Requires:
    - Embedded Tomcat
    - View resolver configuration
- Mostly replaced by **Thymeleaf** today

Short Exam Note (2–3 Marks)

- JSP is a server-side technology used to create dynamic web pages
- It is compiled into a servlet
- Supports EL and JSTL
