-:Spring Boot Notes:-

Important notes on Java Spring Boot:

1. **Basics of Spring Boot**

- **Spring Framework**: A popular Java framework for building enterprise-level applications. It simplifies Java development by providing features like dependency injection, aspect-oriented programming, and transaction management.
- **Spring Boot**: A project under the Spring Framework that makes it easier to set up, configure, and deploy Spring applications. It reduces the need for boilerplate code and configurations.
- **Key Features of Spring Boot**:
    - **Auto-Configuration**: Automatically configures your application based on the dependencies in the project (e.g., if you have spring-boot-starter-web, it will configure Tomcat as the embedded web server).
    - **Embedded Server**: Spring Boot includes an embedded server (like Tomcat, Jetty, or Undertow), so you don't need to deploy your application to an external server.
    - **Opinionated Defaults**: Spring Boot provides sensible defaults and configurations to get you up and running quickly.
    - **Spring Boot Starters**: Pre-configured sets of dependencies (like spring-bootstarter-web for web applications, spring-boot-starter-data-jpa for database access).
    - **Spring Boot Actuator**: Provides production-ready features like health checks, metrics, and application monitoring.

---

2. **Core Components in Spring Boot**

- **Application Class**: Every Spring Boot application has a main class annotated with @SpringBootApplication. This annotation is a combination of:
    - @EnableAutoConfiguration: Tells Spring Boot to configure beans automatically.
    - @ComponentScan: Tells Spring to scan for components (like @Controller, @Service, etc.) in the current package and its sub-packages.
    - @Configuration: Indicates that this class can define beans for the application context.

- **RestController**: A specialized @Controller that simplifies creating REST APIs. It combines @Controller and @ResponseBody so that returned objects are automatically serialized to JSON or XML.
- **Spring Boot Starter**: These are pre-configured templates for specific tasks (like spring-boot-starter-web for web apps, spring-boot-starter-data-jpa for databases).

---

3. Database and Persistence

- **Spring Data JPA**: Spring Boot integrates seamlessly with Spring Data JPA to work with databases.
    - **Entity**: Represents a table in the database, mapped using @Entity annotation.
    - **Repository**: An interface that extends JpaRepository or CrudRepository, providing CRUD operations.
- **H2 Database**: An in-memory database often used for testing and development in Spring Boot applications.
- **Application Properties**: Database configurations (like JDBC URL, username, password) can be set in application.properties or application.yml.

---

4. Dependency Injection (DI)

- **Dependency Injection** is the core principle of Spring Framework.
    - **Autowired**: The @Autowired annotation is used to inject beans automatically into the required places (e.g., constructor, field, setter).
    - **Beans**: In Spring Boot, beans are Java objects that are managed by the Spring container.

---

5. Profiles and Configuration

- **Application Profiles**: Spring Boot allows you to define different configurations for different environments (e.g., dev, prod, test) using application-{profile}.properties files.
    - Example: application-dev.properties, application-prod.properties
- **@Value Annotation**: Used to inject properties from the application.properties or application.yml file into Spring beans.

---

6. **Spring Boot Restful API**

- **REST API Basics**: Using Spring Boot, you can easily build RESTful web services with HTTP methods such as GET, POST, PUT, and DELETE. ○ **@RequestMapping**: Used to map HTTP requests to specific handler methods. ○ **@GetMapping**: Maps HTTP GET requests. ○ **@PostMapping**: Maps HTTP POST requests. ○ **@PutMapping**: Maps HTTP PUT requests.
  - **@DeleteMapping**: Maps HTTP DELETE requests.
- **Exception Handling**: Use @ControllerAdvice to handle exceptions globally and provide custom error responses.

---

7. **Security (Spring Security)**

- **Spring Security**: A framework for securing Java applications, providing authentication and authorization.
  - **Authentication**: Verifying the identity of a user.
  - **Authorization**: Determining if the authenticated user has permission to access certain resources.
- **Basic Authentication**: Allows users to authenticate using a username and password, typically used for simple security needs.
- **JWT (JSON Web Token)**: Often used in modern applications for stateless authentication.

---

8. **Advanced Concepts**

- **Spring Boot Actuator**: Provides built-in endpoints to monitor and manage applications in production. Some common endpoints are: ○ /actuator/health: Provides health check information.
  - /actuator/metrics: Provides metrics for monitoring the app's performance.
- **Spring Boot and Microservices**: Spring Boot is commonly used in microservices architecture, where multiple small services work together to fulfill an application's requirements. Spring Cloud provides additional tools for building microservices with features like service discovery, distributed configuration, and circuit breakers.
- **Spring Cloud**: Used for building cloud-native applications in a microservices architecture. Key components include:

    - **Spring Cloud Netflix**: For microservices patterns like service discovery (Eureka) and circuit breakers (Hystrix).
    - **Spring Cloud Config**: For managing application configurations across multiple environments.
    - **Spring Cloud Gateway**: For routing requests to microservices.
- **Caching**: Spring Boot supports caching mechanisms like @Cacheable to improve performance by storing frequently accessed data.

---

9. Testing in Spring Boot

- **Unit Testing**: Spring Boot supports writing tests using frameworks like JUnit and Mockito. It simplifies testing with @SpringBootTest annotation that loads the full application context.
- **MockMvc**: Used to perform tests on REST APIs by simulating HTTP requests and verifying responses.
- **Integration Testing**: Testing multiple components together in a Spring Boot application to verify their interactions.

---

10. Deployment and Packaging

- **Executable JAR**: Spring Boot can create an executable JAR file that includes an embedded server and all the necessary dependencies. You can run it with java -jar your-app.jar.
- **Docker**: Spring Boot applications can be containerized using Docker, making deployment easier and more consistent.
- **Cloud Deployment**: Spring Boot applications are easy to deploy to cloud platforms like AWS, Google Cloud, and Azure.

Important topics In Spring Boot:

1. Spring Boot Introduction

- What is Spring Boot?
- Benefits of using Spring Boot
- Difference between Spring and Spring Boot

2. Spring Boot Setup and Configuration

- How to create a Spring Boot application (using Spring Initializr, IDE, or command line)

- Spring Boot Starter Projects (starter dependencies)
- application.properties and application.yml configuration files

3. **Spring Boot Annotations**
   - **@SpringBootApplication**: Main entry point for Spring Boot applications.
   - **@RestController**: For creating RESTful APIs.
   - **@RequestMapping, @GetMapping, @PostMapping**: To map HTTP requests to methods.
   - **@Component, @Service, @Repository, @Controller**: Defining beans in Spring Boot.
4. **Spring Boot Auto Configuration**
   - What is auto-configuration in Spring Boot?
   - How Spring Boot auto-configures beans and resources based on dependencies.
5. **Spring Boot Dependency Injection**
   - What is Dependency Injection (DI)?
   - Types of DI: Constructor injection, setter injection, field injection
   - **@Autowired**: How Spring Boot injects beans automatically.
6. **Spring Boot Data Access (JPA and Hibernate)**
   - What is Spring Data JPA?
   - How to configure a database connection in Spring Boot
   - CRUD operations with **JpaRepository** or **CrudRepository**
   - How to use **@Entity** and **@Repository** annotations
   - Using **Spring Boot with MySQL, PostgreSQL, etc.**
7. **Spring Boot RESTful Web Services**
   - Building REST APIs with **@RestController**
   - Handling HTTP requests with **@RequestMapping**, **@GetMapping**, **@PostMapping**, **@PutMapping**, and **@DeleteMapping**
   - **@PathVariable**, **@RequestParam**, **@RequestBody**
   - ResponseEntity and status codes
8. **Spring Boot Validation**
   - Validating user input using **@Valid** and **@NotNull, @Size, @Email, etc.**
   - Custom validation using annotations

9. **Spring Boot Profiles**
    - What are profiles in Spring Boot?
    - Using multiple profiles for different environments (dev, prod, test)
    - **spring.profiles.active** configuration in application.properties
10. **Spring Boot Security**
    - Introduction to **Spring Security** in Spring Boot
    - Basic authentication and authorization
    - Using **@EnableWebSecurity** and HttpSecurity to configure security
    - Role-based access control and method security
11. **Spring Boot Exception Handling**
    - Handling exceptions in Spring Boot
    - Using **@ExceptionHandler** to handle specific exceptions
    - Global exception handling with **@ControllerAdvice**
12. **Spring Boot Actuator**
    - What is Spring Boot Actuator?
    - Exposing application metrics (e.g., health, info, metrics endpoints)
    - Monitoring application health and performance
13. **Spring Boot Logging**
    - Default logging with **SLF4J** and **Logback**
    - How to configure logging in application.properties
    - Custom log levels for different packages
14. **Spring Boot Testing**
    - **Unit Testing** with JUnit and Mockito
    - **Integration Testing** with **@SpringBootTest**
    - Writing tests for RESTful APIs using **MockMvc**
15. **Spring Boot Asynchronous Processing**
    - Using **@Async** for asynchronous method execution
    - **@EnableAsync** to enable async processing globally
16. **Spring Boot Scheduling**

- Scheduling tasks using **@Scheduled** annotation
- Configuring cron expressions for scheduled tasks

17. Spring Boot Messaging (MQ)

- Integrating with message queues like **RabbitMQ** and **Kafka** using Spring Boot
- Sending and receiving messages with **@RabbitListener** or **@KafkaListener**

18. Spring Boot Testing with MockMvc

- Writing tests for controller methods using **MockMvc**
- Verifying request/response and status codes in tests

19. Spring Boot File Handling

- Uploading and downloading files in Spring Boot
- Handling file storage and access

20. Spring Boot Caching

- Introduction to caching in Spring Boot
- Using **@Cacheable** to cache method results
- Different cache providers like **EhCache, Redis**

21. Spring Boot Integration with External APIs

- Calling external APIs using **RestTemplate** or **WebClient**
- Consuming third-party services in Spring Boot

22. Spring Boot WebSocket

- Introduction to WebSocket for real-time communication
- Building WebSocket applications in Spring Boot

23. Spring Boot with Thymeleaf

- Using Thymeleaf for rendering dynamic web pages
- Spring Boot with MVC architecture and Thymeleaf

24. Spring Boot Profile-Specific Configuration

- Managing different configurations for different environments (e.g., dev, prod, test)
- Externalized configuration using application-dev.properties, applicationprod.properties

25. Spring Boot Custom Starter

- How to create custom starters in Spring Boot for reusable configurations or libraries
- Creating reusable libraries with specific configurations and dependencies

26. Spring Boot Dependency Management

- Understanding **Spring Boot Starter Dependencies** and how they help manage versions.
- How Spring Boot simplifies dependency management.

27. Spring Boot with Docker

- Containerizing Spring Boot applications using Docker
- Running Spring Boot in Docker containers and Docker Compose

28. Spring Boot with Cloud (Spring Cloud)

- Introduction to **Spring Cloud** for building cloud-native applications
- Concepts like Service Discovery, Circuit Breakers, Configuration Management in the cloud

29. Spring Boot Data Binding

- Binding form inputs to Java objects
- Using **@ModelAttribute**, **@RequestBody**, and **@ResponseBody**

30. Spring Boot Performance Tuning

- Optimizing Spring Boot application for performance
- Using **JVM tuning**, **Thread Pool configuration**, etc.