## Basic Questions:

1. **What is Spring Boot?**
    - **Answer:** Spring Boot is an open-source Java-based framework used to create stand-alone, production-ready Spring applications with minimal configuration. It simplifies the setup and development of new Spring applications by providing default configurations and features like embedded servers (Tomcat, Jetty, etc.), auto-configuration, and a range of built-in tools for rapid development.

---

2. **What are the main features of Spring Boot?**
    - **Answer:**
        - **Auto-Configuration:** Automatically configures Spring application based on the project's dependencies.
        - **Spring Boot Starters:** A set of dependency descriptors to include necessary libraries.
        - **Embedded Servers:** Provides built-in servers like Tomcat or Jetty, so you don't need to deploy WAR files.
        - **Spring Boot CLI:** A command-line tool for quickly prototyping Spring applications.
        - **Actuator:** Provides production-ready features like monitoring and managing applications through endpoints.
        - **Standalone Applications:** Allows the creation of Java applications that can run with `java -jar`.

---

3. **What is the difference between Spring and Spring Boot?**
    - **Answer:**
        - **Spring:** A comprehensive framework for building Java applications. It requires extensive configuration and is used for creating a wide range of applications (web, microservices, batch, etc.).

- **Spring Boot:** A sub-project of Spring that simplifies application development by offering pre-configured setups and defaults, reducing the need for boilerplate code. It allows developers to start coding with minimal setup and is ideal for microservices.

---

4. **What is a Spring Boot Starter?**
   - **Answer:** Spring Boot Starters are a set of convenient dependency descriptors that aggregate commonly used libraries for specific functionalities. For example:
     - **spring-boot-starter-web:** Includes dependencies for building web applications (e.g., Spring MVC, Tomcat).
     - **spring-boot-starter-data-jpa:** Includes dependencies for JPA and Spring Data.
     - **spring-boot-starter-security:** Includes dependencies for Spring Security.

---

5. **What is @SpringBootApplication annotation?**
   - **Answer:** `@SpringBootApplication` is a convenience annotation that combines three crucial annotations:
     - **@Configuration:** Marks the class as a source of bean definitions.
     - **@EnableAutoConfiguration:** Enables Spring Boot's auto-configuration mechanism, which automatically configures beans based on the classpath settings.
     - **@ComponentScan:** Enables component scanning, so Spring can find and register beans within the package of the annotated class and its sub-packages.

---

6. **How do you create a Spring Boot application?**
   - **Answer:**
     - **Using Spring Initializr:** Go to the Spring Initializr website, select your project settings (dependencies, Java version, etc.), and generate the project. Import it into your IDE and start coding.

- **Using Spring Boot CLI:** Install Spring Boot CLI and create a new project by running commands like `spring init --dependencies=web myproject`.
- **Manually:** Create a new Maven or Gradle project, add Spring Boot dependencies, and annotate the main class with `@SpringBootApplication`.

---

7. **What is Spring Boot Auto-Configuration?**
   - **Answer:** Spring Boot's auto-configuration feature automatically configures your Spring application based on the dependencies present in the classpath. For example, if `spring-boot-starter-web` is in the classpath, it automatically configures Spring MVC, a dispatcher servlet, and a default embedded server (Tomcat). This reduces the need for manual configuration, allowing you to focus on your application's business logic.

---

8. **What is the role of the application.properties file in Spring Boot?**
   - **Answer:** The `application.properties` (or `application.yml`) file is used to define application-level configurations in a Spring Boot project. You can configure settings like:
     - **Server properties:** `server.port=8081`
     - **Database configurations:** `spring.datasource.url=jdbc:mysql://localhost:3306/mydb`
     - **Logging levels:** `logging.level.org.springframework=DEBUG`
     - **Custom properties:** You can also define your own properties, e.g., `myapp.custom.property=value`.

---

9. **What is Spring Boot Actuator?**
   - **Answer:** Spring Boot Actuator provides a set of production-ready features to monitor and manage your application. It exposes various endpoints, such as:
     - **/actuator/health:** Displays the health status of the application.

    - **/actuator/metrics:** Provides metrics data like memory usage, CPU usage, and other application statistics.
    - **/actuator/env:** Displays environment properties. Actuator endpoints can be customized and secured, and it integrates well with monitoring tools like Prometheus and Grafana.

---

10. **What is an embedded server in Spring Boot?**
    - **Answer:** An embedded server is a server that is packaged with your application, allowing it to run as a stand-alone Java application without needing to deploy WAR files to an external server. Spring Boot supports embedded servers like Tomcat, Jetty, and Undertow. This simplifies the deployment process and is especially useful for microservices and containerized applications.

---

## Advanced Questions:

11. **What is Spring Boot DevTools?**
    - **Answer:** Spring Boot DevTools is a module that enhances the development experience by providing features like automatic restart, live reload, and configuration properties that are more suitable for development environments. It helps speed up the development process by reducing the need for manual application restarts after code changes.

---

12. **How does Spring Boot handle security?**
    - **Answer:** Spring Boot provides security out-of-the-box through Spring Security. By adding `spring-boot-starter-security` to your project, you automatically enable basic authentication. You can customize security configurations using `@EnableWebSecurity` and configure specific authentication and authorization rules through the `WebSecurityConfigurerAdapter`.

---

13. **What is the purpose of @RestController in Spring Boot?**
    - **Answer:** `@RestController` is a convenience annotation that combines `@Controller` and `@ResponseBody`. It marks a class as a controller where every method returns a domain object instead of a view. The response is automatically converted to JSON or XML, depending on the client request.

---

14. **What is the difference between @Controller and @RestController?**
    - **Answer:**
        1. **@Controller:** Used to define a controller class that handles web requests and typically returns a view (e.g., HTML).
        2. **@RestController:** Used to define a controller class that handles RESTful web services. It returns data directly (like JSON) instead of views, eliminating the need for `@ResponseBody` on each method.

---

15. **What is a @Bean in Spring Boot?**
    - **Answer:** `@Bean` is an annotation used to declare a bean within a Spring configuration class. Beans are objects that are managed by the Spring container. Declaring a method with `@Bean` ensures that the method's return value is registered as a bean in the Spring application context.

---

16. **How do you define a custom exception in Spring Boot?**
    - **Answer:**
        1. Create a custom exception class, e.g., `public class ResourceNotFoundException extends RuntimeException {}`.
        2. Use `@ResponseStatus` to associate an HTTP status code with the exception, e.g., `@ResponseStatus(HttpStatus.NOT_FOUND)`.
        3. Optionally, create a global exception handler using `@ControllerAdvice` and handle custom exceptions using `@ExceptionHandler`.

---

17. **How do you handle application configuration in Spring Boot?**
    - **Answer:** Configuration in Spring Boot is managed through properties files (`application.properties` or `application.yml`), environment variables, and command-line arguments. You can also use profiles (e.g., `application-dev.properties`) for environment-specific configurations. Configuration properties can be injected into beans using `@Value` or
    - `@ConfigurationProperties`.

---

    -

18. **What is @ConfigurationProperties in Spring Boot?**
    - **Answer:** `@ConfigurationProperties` is used to bind external configurations from properties or YAML files to a Java object. For example, you can map properties with a common prefix (e.g., `app.datasource`) to a POJO by annotating the class with
    - `@ConfigurationProperties(prefix="app.datasource")`.

---

**19. What is a Spring Boot Profile?**

- **Answer:** Profiles in Spring Boot allow you to create different configurations for different environments, such as development, testing, and production. You can define environment-specific properties in separate files like `application-dev.properties`, `application-test.properties`, etc. You can activate a profile by setting the `spring.profiles.active` property in the `application.properties` file, passing it as a command-line argument, or setting it in the environment variables.

---

**20. How do you handle exceptions in Spring Boot?**

- **Answer:** Spring Boot provides several ways to handle exceptions in a web application:

- **Global Exception Handling:** Use `@ControllerAdvice` and `@ExceptionHandler` annotations to create a global exception handler that can catch and handle exceptions across the entire application.
- **Custom Error Pages:** Define custom error pages by creating `ErrorController` or by configuring error views in the `application.properties` file.
- **ResponseEntityExceptionHandler:** Extend this class to create a centralized exception handling mechanism with specific HTTP status codes and response bodies.