# *ItsRunTym*

# JAVA Spring Boot

## 50 interview questions/answers

### Spring Boot Basics

1. What is Spring Boot and what are its key features?

   Answer: Spring Boot is a framework that simplifies the development of new Spring applications. It provides defaults for code and annotation configuration to quickly start new Spring projects. Key features include:
   - Auto-configuration: Automatically configures your application based on the dependencies you have added.
   - Starter POMs: Simplifies dependency management.
   - Spring Boot CLI: Allows you to quickly prototype with Spring.
   - Embedded Servers: Comes with embedded servers like Tomcat, Jetty, or Undertow.

2. Explain the difference between Spring and Spring Boot.

   Answer: Spring is a comprehensive framework for enterprise Java development, while Spring Boot is an extension that helps in rapid application development. Spring Boot simplifies the setup and development by providing auto-configuration and embedded servers, whereas Spring requires more manual setup.

3. How do you create a Spring Boot application?

   Answer: You can create a Spring Boot application using Spring Initializr (https://start.spring.io/) or by using the Spring Boot CLI.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

4. What is the purpose of the @SpringBootApplication annotation?

   Answer: @SpringBootApplication is a convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
   It enables auto-configuration and component scanning, making it easier to configure and bootstrap the application.

5. How do you configure a Spring Boot application?

   Answer: Configuration can be done using application.properties or application.yml files located in the src/main/resources directory.

   ```
   server.port=8081
   spring.datasource.url=jdbc:mysql://localhost:3306/mydb
   spring.datasource.username=root
   spring.datasource.password=pass
   ```

## Dependency Injection and Beans

6. Explain dependency injection in Spring Boot.

   Answer: Dependency injection is a design pattern used to implement IoC (Inversion of Control), allowing the creation of dependent objects outside of a class and providing those objects to a class through different ways.

7. What are the different types of dependency injection in Spring?

   Answer: The types include:
   - Constructor Injection
   - Setter Injection
   - Field Injection

8. How do you define a Spring Bean?

   Answer: Beans can be defined using annotations like @Component, @Service, @Repository, or by using the @Bean annotation within a @Configuration class.

```
@Component
public class MyComponent {
        // Bean logic
}

@Configuration
public class AppConfig {
        @Bean
        public MyComponent myComponent() {
                return new MyComponent();

        }

}
```

9. What is the difference between @Component, @Service, @Repository, and @Controller?
   Answer:
   - @Component: General-purpose stereotype annotation for any Spring-managed component.
   - @Service: Specialization of @Component for service layer.
   - @Repository: Specialization of @Component for DAO (Data Access Object) layer.
   - @Controller: Specialization of @Component for web controller.

10. Explain the difference between @Autowired and @Inject.

    Answer:
    @Autowired is a Spring-specific annotation for dependency injection,
    while @Inject is a standard annotation from JSR-330 (Java Dependency Injection).
    Both can be used interchangeably in Spring applications.

## Spring Boot Annotations

11. What is the use of @Configuration?

    Answer: @Configuration indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

12. How does @ComponentScan work?

Answer: @ComponentScan is used to specify the packages to scan for annotated components (e.g., @Component, @Service, @Repository, and @Controller). It enables Spring to automatically detect and register beans.

```
@Configuration
@ComponentScan(basePackages = "com.example.myapp")
public class AppConfig {
}
```

13. What is the purpose of @Conditional annotations in Spring Boot?

    Answer: @Conditional annotations are used to conditionally register beans based on certain conditions, such as the presence of a class, property, or method. For example, @ConditionalOnProperty, @ConditionalOnClass.

14. Explain the use of @RestController vs @Controller.

    Answer: @RestController is a specialized version of @Controller that combines @Controller and @ResponseBody. It is used to create RESTful web services, where each method returns a domain object instead of a view.

```
@RestController
public class MyRestController {
        @GetMapping("/hello")
        public String sayHello() {
                return "Hello, World!";

        }

}
```

15. What is @PathVariable and how is it different from @RequestParam?

    Answer: @PathVariable is used to extract values from the URI path, whereas @RequestParam is used to extract query parameters from the URL.

```
@GetMapping("/users/{id}")
public User getUser(@PathVariable("id") Long id) {
        // logic to get user by id
}

@GetMapping("/users")
public List<User> getUsers(@RequestParam("name") String name) {
        // logic to get users by name
}
```

## Configuration and Properties

16. How do you externalize configuration in Spring Boot?

    Answer: Configuration can be externalized using application.properties, application.yml, or environment variables.

17. Explain the use of application.properties and application.yml files.

    Answer: application.properties and application.yml are used to define configuration properties for Spring Boot applications. The application.yml file allows hierarchical data and is more readable for complex configurations.

```
server:
	port: 8081
spring:
	datasource:
		url: jdbc:mysql://localhost:3306/mydb
		username: root
		password: pass
```

18. How do you handle different environments in Spring Boot?

    Answer: Use Spring Profiles to handle different environments. Profiles can be specified in application.properties or application.yml and activated via the spring.profiles.active property.

```
spring.profiles.active=dev
```

```
spring:
	profiles: dev
	datasource:
		url: jdbc:mysql://localhost:3306/devdb
		username: devuser
		password: devpass
```

19. What is the purpose of @Value annotation in Spring Boot?

    Answer: @Value is used to inject values from properties files or environment variables into fields.

```
@Value("${my.custom.property}")
private String myProperty;
```

20. Explain how profiles work in Spring Boot.

    Answer: Profiles allow you to segregate parts of your application configuration and make them available only in certain environments. Beans can be annotated with @Profile to indicate that they should only be created in specific profiles.

```
@Profile("dev")
@Configuration
public class DevConfig {
        // Development-specific beans

}
```

## Spring Boot Starters

21. What are Spring Boot Starters?

    Answer: Spring Boot Starters are a set of convenient dependency descriptors you can include in your application. They provide a ready-to-use set of dependencies for various features (e.g., spring-boot-starter-web).

22. Name some commonly used Spring Boot Starters.
    Answer:
    - spring-boot-starter-web: For building web, including RESTful, applications.
    - spring-boot-starter-data-jpa: For JPA and Hibernate.
    - spring-boot-starter-security: For security and authentication.
    - spring-boot-starter-test: For testing Spring Boot applications.

23. How do you create a custom Spring Boot Starter?

    Answer: To create a custom starter, you need to create a new module, add required dependencies in pom.xml or build.gradle, and provide auto-configuration classes.

```
@Configuration
public class MyCustomAutoConfiguration {
        @Bean
        public MyService myService() {
                return new MyService();

        }
}
```

## Spring Boot Actuator

24. What is Spring Boot Actuator?

    Answer: Spring Boot Actuator provides production-ready features like monitoring and managing your application. It includes several endpoints to help you understand the current state of your application.

25. How do you enable Spring Boot Actuator in an application?

    Answer: Add the ‘spring-boot-starter-actuator’ dependency to your ‘pom.xml’ or ‘build.gradle’.

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

</dependency>

management.endpoints.web.exposure.include=*
```

26. What are some of the key metrics and endpoints provided by Actuator?

    Answer: Key endpoints include:
    - /actuator/health: Shows application health information.
    - /actuator/info: Displays arbitrary application info.
    - /actuator/metrics: Shows various application metrics.

## Spring Boot Security

27. How do you secure a Spring Boot application?

    Answer: By adding the spring-boot-starter-security dependency and configuring security settings using WebSecurityConfigurerAdapter.

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http
                        .authorizeRequests()
```

```
                    .antMatchers("/public/**").permitAll()
                    .anyRequest().authenticated()
                    .and()
                    .formLogin();
        }
}
```

28. What is Spring Security and how does it integrate with Spring Boot?

    Answer: Spring Security is a framework for securing Java applications. It integrates with Spring Boot by providing default security configurations and easy-to-use annotations for securing endpoints.

29. Explain the purpose of @PreAuthorize and @Secured annotations.
    Answer:
    - @PreAuthorize: Checks authorization before executing the method.
    - @Secured: Specifies a list of roles required for the method invocation.

```
@PreAuthorize("hasRole('ROLE_USER')")
public void secureMethod() {
        // method logic
}
```

30. How do you implement OAuth2 authentication in Spring Boot?

    Answer: By adding the spring-boot-starter-oauth2-client dependency and configuring OAuth2 clients in application.properties or application.yml.

```
spring:
        security:
                oauth2:
                        client:
                                registration:
                                        google:
                                                client-id: your-client-id
                                                client-secret: your-client-secret
                                                scope: profile, email
                                                redirect-uri:
                                        "{baseUrl}/login/oauth2/code/{registrationId}"
```

## Spring Boot Data

31. What is Spring Data JPA?

    Answer: Spring Data JPA is part of the larger Spring Data family, making it easier to implement JPA-based repositories. It reduces boilerplate code and provides repository support.

32. Explain the purpose of @Entity and @Table annotations.
    Answer: @Entity marks a class as a JPA entity. @Table specifies the table in the database that maps to the entity.

```
@Entity @Table(name = "users")
public class User {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(name = "name")
        private String name;

}
```

33. What is a Repository in Spring Data JPA?

    Answer: A repository is a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects. Spring Data JPA provides CrudRepository, JpaRepository, and PagingAndSortingRepository.

34. How do you implement pagination and sorting in Spring Data JPA?

    Answer: Use Pageable and Sort in repository methods.

```
public interface UserRepository extends JpaRepository<User, Long> {
        Page<User> findAll(Pageable pageable);

 }


Pageable pageable = PageRequest.of(0, 10, Sort.by("name"));
Page<User> users = userRepository.findAll(pageable);
```

35. Explain the use of @Query annotation.

Answer: @Query is used to define custom queries using JPQL or SQL in repository interfaces.

```
@Query("SELECT u FROM User u WHERE u.name = :name")
List<User> findUsersByName(@Param("name") String name);
```

## Spring Boot Testing

36. How do you test Spring Boot applications?

Answer: Spring Boot applications can be tested using @SpringBootTest for integration tests and @WebMvcTest, @DataJpaTest for slicing tests.

```
@SpringBootTest
public class MyApplicationTests {
        @Test
        void contextLoads() {
        }

}
```

37. What is the use of @SpringBootTest annotation?

Answer: @SpringBootTest is used to create an application context for integration tests. It loads the full application context.

38. Explain the purpose of @MockBean and @Autowired in testing.
Answer:
- @MockBean: Replaces a bean in the application context with a mock.
- @Autowired: Injects dependencies into your test class.

```
@SpringBootTest
public class MyServiceTests {
        @MockBean
        private MyRepository myRepository;

        @Autowired
        private MyService myService;

        @Test
        void testServiceMethod() {
                // testing logic
                }
}
```

39. How do you test RESTful services in Spring Boot?

    Answer: Use MockMvc to test RESTful services.

```
@WebMvcTest(MyRestController.class)
public class MyRestControllerTests {
        @Autowired
        private MockMvc mockMvc;

        @Test
        void testGetHello() throws Exception {
                mockMvc.perform(get("/hello"))
                        .andExpect(status().isOk())
                        .andExpect(content().string("Hello, World!"));

        }

}
```

## Spring Boot Best Practices

40. What are some best practices for designing Spring Boot applications?
    Answer:
    - Follow SOLID principles.
    - Use profiles for different environments.
    - Externalize configuration.
    - Use meaningful logging.
    - Write tests for critical parts of the application.

41. How do you handle exceptions in Spring Boot?

    Answer: Use @ControllerAdvice and @ExceptionHandler to handle exceptions globally.

```
@ControllerAdvice
public class GlobalExceptionHandler {
        @ExceptionHandler(UserNotFoundException.class)
        public ResponseEntity<String> handleUserNotFound(UserNotFoundException
        ex) {
            return
        ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
            }

}
```

42. What are some ways to improve performance in Spring Boot applications?
    Answer:
    - Use caching.
    - Optimize database queries.
    - Enable asynchronous processing.
    - Profile and monitor the application.
    - Use appropriate JVM settings.

## Advanced Spring Boot Topics

43. What is Spring Cloud and how does it integrate with Spring Boot?

    Answer: Spring Cloud provides tools for developers to quickly build common patterns in distributed systems (e.g., configuration management, service discovery). It integrates with Spring Boot to simplify microservice architecture.

44. Explain the purpose of Spring Boot DevTools.

    Answer: Spring Boot DevTools provides features to improve the development experience, such as automatic restart, live reload, and configurations for development.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <optional>true</optional>

</dependency>
```

45. How do you handle transactions in Spring Boot?

    Answer: Transactions in Spring Boot can be handled using @Transactional annotation on methods that require transactional behavior.

```
@Service
public class MyService {
        @Transactional
        public void performTransaction() {
```

```
        // Transactional logic
    }
}
```

46. What is the use of @Transactional annotation?

    Answer: @Transactional is used to mark methods or classes to be executed within a transaction context. It manages the transaction lifecycle (begin, commit, rollback).

47. How do you implement caching in Spring Boot?

    Answer: By using Spring's caching abstraction with annotations like @EnableCaching, @Cacheable, @CachePut, and @CacheEvict.

```
@EnableCaching
@Configuration
public class CacheConfig {
}

@Service
public class MyService {
    @Cacheable("items")
    public Item getItemById(Long id) {
        // Retrieve item
    }
}
```

## Miscellaneous

48. How do you create a RESTful web service using Spring Boot?

    Answer: Create a controller class annotated with @RestController and define request mappings using @GetMapping, @PostMapping, etc.

```
@RestController
@RequestMapping("/api")
public class MyRestController {
    @GetMapping("/items")
    public List<Item> getAllItems() {
        // Logic to get items
    }

    @PostMapping("/items")
    public Item createItem(@RequestBody Item item) {
        // Logic to create item

    }

}
```

49. What is the role of a Servlet Filter in Spring Boot?

    Answer: A Servlet Filter is used to perform filtering tasks on request and response objects before they reach the servlet or after the servlet has processed the request. It can be used for logging, authentication, etc.

```
@Component
public class MyFilter implements Filter {
        @Override
        public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
                // Filtering logic
                chain.doFilter(request, response);

        }

}
```

50. Explain the concept of microservices and how Spring Boot is used to build microservices.

    Answer: Microservices architecture structures an application as a collection of loosely coupled services, each implementing a business capability. Spring Boot simplifies microservices development with features like embedded servers, actuator for monitoring, and Spring Cloud for managing infrastructure concerns.

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

        }

}
```

```
server:
    port: 8080
spring:
    application:
        name: microservice-name
eureka:
    client:
        service-url:
            defaultZone: http://localhost:8761/eureka/
```