Spring and springboot Mock qns

1. What are the key features of the Spring Framework?

The Spring Framework is a powerful Java-based framework that simplifies enterprise application development. Key features include:

1. **Dependency Injection (DI)** - Helps manage object dependencies efficiently.
2. **Aspect-Oriented Programming (AOP)** - Separates cross-cutting concerns like logging, security, and transactions.
3. **Spring MVC** - A web framework for building scalable web applications.
4. **Spring Boot** - A module that simplifies microservices and standalone applications.
5. **Transaction Management** - Manages database transactions declaratively.
6. **Integration Support** - Works well with Hibernate, JPA, JDBC, JMS, and other frameworks.
7. **Security** - Provides authentication and authorization features.

2. What is a Spring Container?

The **Spring Container** is responsible for managing the lifecycle of Spring beans. It creates, configures, and manages beans defined in the configuration file or via annotations.

Types of Containers:

- **BeanFactory** - Lightweight container, useful for small applications.
- **ApplicationContext** - More advanced, supports internationalization, event propagation, and AOP.

Example:

java

CopyEdit

```
// Bean class

public class HelloWorld {

    public void sayHello() {

        System.out.println("Hello, Spring!");

    }

}
```

```
// Configuration and execution

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;



public class MainApp {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

        obj.sayHello();

    }

}
```

3. What is Dependency Injection (DI), and how does Spring implement it?

Dependency Injection (DI) is a design pattern where dependencies are injected into a class instead of being created inside the class. Spring implements DI through:

1. **Constructor Injection**
2. **Setter Injection**
3. **Field Injection (via @Autowired)**

Example of Constructor Injection:

java

CopyEdit

```
@Component

public class Car {

    private Engine engine;



    @Autowired

    public Car(Engine engine) {

        this.engine = engine;
    }

}
```

Example of Setter Injection:

java

CopyEdit

```
@Component

public class Car {

    private Engine engine;



    @Autowired

    public void setEngine(Engine engine) {

        this.engine = engine;

    }

}
```

4. Explain the difference between BeanFactory and ApplicationContext

Feature	BeanFactory	ApplicationContext
Type	Basic container	Advanced container
Lazy Loading	Yes (creates beans only when needed)	No (creates beans at startup)
Event Handling	No	Yes
AOP Support	No	Yes

5. What are the different scopes of a Spring bean?

Spring beans can have different scopes:

1. **Singleton** - A single instance per Spring container (default).

2. **Prototype** - New instance each time a bean is requested.
3. **Request** - New instance per HTTP request (Spring Web).
4. **Session** - New instance per HTTP session.
5. **Global-session** - Shared across all sessions.

Example of defining a prototype scope:

java

CopyEdit

```
@Component

@Scope("prototype")

public class MyBean {

    // Bean code

}
```

6. What is the difference between constructor injection and setter injection?

Type	Constructor Injection	Setter Injection
Use case	Required dependencies	Optional dependencies
Flexibility	Less flexible	More flexible
Circular Dependency	May cause issues	Avoids issues

Constructor Injection Example:

java

CopyEdit

```
public class Student {

    private Address address;



    @Autowired

    public Student(Address address) {
        this.address = address;
    }
}
```

Setter Injection Example:

java

CopyEdit

```
public class Student {
    private Address address;


    @Autowired
    public void setAddress(Address address) {
        this.address = address;
    }
}
```

7. What is Spring Boot, and how is it different from the Spring Framework?

Spring Boot is a module of the Spring Framework that simplifies application development by providing:

- Auto-configuration
- Embedded servers (Tomcat, Jetty)
- Opinionated defaults

Feature	Spring Framework	Spring Boot
Configuration	Requires XML/Java config	Auto-configured
Dependency Mgmt	Manual	Starter dependencies
Server	External setup needed	Embedded Tomcat

8. What is @Bean annotation, and how is it different from @Component?

Feature	@Bean (used in @Configuration class)	@Component (used on classes)
Scope	Defined manually in Java config	Scanned automatically
Use case	Used for third-party libraries	Used for user-defined beans

Example of @Bean:

java

CopyEdit

```
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

9. What is @Qualifier, and how is it used?

The @Qualifier annotation is used to resolve multiple bean conflicts when there are multiple candidates.

Example:

java

CopyEdit

```
@Component
public class PetrolEngine implements Engine { }
```

```
@Component
public class DieselEngine implements Engine { }


@Component
public class Car {
    private Engine engine;


    @Autowired
    public Car(@Qualifier("dieselEngine") Engine engine) {
        this.engine = engine;
    }
}
```

10. What is JUnit, and why is it used?

JUnit is a popular testing framework for Java. It helps in writing unit tests.

Example:

java

CopyEdit

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;


public class MathTest {
    @Test
    public void testAddition() {
        assertEquals(5, 2 + 3);
    }
}
```

11. How do you create a REST API in Spring Boot?

1. Add **Spring Web** dependency.
2. Create a **Controller** with @RestController.
3. Define **routes** using @GetMapping, @PostMapping, etc.

Example:

java

CopyEdit

```
@RestController
@RequestMapping("/api")
public class MyController {

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

12. What is the difference between @Controller and @RestController?

Feature	@Controller	@RestController
Use case	Returns views (JSP/Thymeleaf)	Returns JSON/XML
Annotation	Used with @ResponseBody	Inherits @ResponseBody

Example:

java

CopyEdit

```
@Controller
```

```
public class WebController {

    @GetMapping("/page")

    public String showPage() {

        return "index"; // Returns view

    }

}
```

java

CopyEdit

```
@RestController

public class APIController {

    @GetMapping("/json")

    public String getJson() {

        return "{\"message\": \"Hello\"}";

    }

}
```

LEFT A FEW QNS, SO:

1. What are the key features of the Spring Framework?

Spring is a popular Java framework that simplifies enterprise application development. Key features:

- **Dependency Injection (DI)** – Manages object dependencies.
- **Aspect-Oriented Programming (AOP)** – Handles cross-cutting concerns (logging, security).
- **Spring MVC** – Web framework for creating scalable applications.
- **Spring Boot** – Simplifies Spring applications with auto-configuration.
- **Data Access** – Supports JDBC, JPA, Hibernate.
- **Security** – Built-in authentication and authorization.

2. What is a Spring Container?

A **Spring Container** is responsible for creating, managing, and configuring Spring beans.
Types of Containers:

- **BeanFactory** – Lightweight, lazy loading.
- **ApplicationContext** – Advanced features like event handling, AOP.

Example:

java

CopyEdit

```
ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

MyBean obj = context.getBean(MyBean.class);
```

3. What is Dependency Injection (DI), and how does Spring implement it?

Dependency Injection (DI) is a design pattern where dependencies are injected instead of being created inside the class.

Spring DI supports:

1. **Constructor Injection**
2. **Setter Injection**
3. **Field Injection (@Autowired)**

Example of Constructor Injection:

java

CopyEdit

```
@Component

public class Car {

    private Engine engine;



    @Autowired

    public Car(Engine engine) {

        this.engine = engine;

    }

}
```

4. Difference between BeanFactory and ApplicationContext

Feature	BeanFactory	ApplicationContext
Loading	Lazy	Eager
Event Handling	No	Yes
AOP Support	No	Yes

5. What are the different scopes of a Spring bean?

1. **Singleton** – Single instance per container (default).
2. **Prototype** – New instance per request.
3. **Request** – New instance per HTTP request (Spring Web).
4. **Session** – New instance per session.

Example:

java

CopyEdit

```
@Scope("prototype")

@Component

public class MyBean { }
```

6. Difference between Constructor Injection and Setter Injection

Feature	Constructor Injection	Setter Injection
Mandatory Dependency	Yes	No
Flexibility	Less flexible	More flexible
Circular Dependency	Problematic	Avoids it

7. How does Spring manage bean lifecycle?

1. **Instantiation**
2. **Dependency Injection**
3. **Initialization (@PostConstruct)**
4. **Destruction (@PreDestroy)**

Example:

java

CopyEdit

```
@Component
public class MyBean {
    @PostConstruct
    public void init() { System.out.println("Bean initialized"); }

    @PreDestroy
    public void destroy() { System.out.println("Bean destroyed"); }
}
```

8. What is Spring Boot, and how is it different from Spring Framework?

Spring Boot is a module that simplifies Spring applications by providing:

- **Auto-configuration**
- **Embedded servers (Tomcat, Jetty)**
- **No XML configuration needed**

9. Difference between Eager Loading and Lazy Loading in Hibernate/Spring

Feature	**Eager Loading**	**Lazy Loading**
Fetch Timing	**At the start**	**When accessed**
Performance	**Slower**	**Faster**

Example in Hibernate:

java

CopyEdit

@OneToMany(fetch = FetchType.LAZY)

private List<Order> orders;

10. What is @Value and @PropertySource annotation?

- @Value injects property values into beans.
- @PropertySource loads properties files.

Example:

java

CopyEdit

```
@PropertySource("classpath:application.properties")

public class AppConfig {

   @Value("${app.name}")

   private String appName;

}
```

11. Purpose of @Component, @Service, @Repository, @Controller

Annotation	Purpose
@Component	**Generic bean**
@Service	**Business logic layer**
@Repository	**Data access layer**
@Controller	**Web controller**

12. What is @Bean and how is it different from @Component?

Feature	@Bean	@Component
Definition	In @Configuration class	Directly on a class
Use case	Third-party beans	Spring-managed classes

Example:

java

CopyEdit

```
@Configuration
public class MyConfig {
   @Bean
   public MyBean myBean() {
      return new MyBean();
   }
}
```

13. What is @Qualifier in Spring?

Used to resolve multiple bean conflicts.

Example:

java

CopyEdit

```
@Autowired
@Qualifier("dieselEngine")
private Engine engine;
```

14. What is JUnit and why is it used?

JUnit is a testing framework for Java used for unit testing.

Example:

java

CopyEdit

```java
@Test

public void testAddition() {

   assertEquals(5, 2 + 3);

}
```

15. Difference between @Before, @BeforeEach, @After, @AfterEach in JUnit

Annotation	Purpose
@BeforeEach	**Runs before each test**
@AfterEach	**Runs after each test**

16. How do you test Spring beans using JUnit?

Use @SpringBootTest and @MockBean.

Example:

java

CopyEdit

```java
@SpringBootTest

public class MyServiceTest {

   @MockBean

   private MyRepository myRepository;



   @Autowired
```

```
    private MyService myService;


    @Test

    public void testService() {

        when(myRepository.findById(1)).thenReturn(Optional.of(new MyEntity()));

        assertNotNull(myService.getEntity(1));

    }

}
```

17. How does Spring Boot auto-configure?

Spring Boot scans dependencies and applies default configurations automatically.

18. What are Spring Boot Starters?

Spring Boot starters are dependency bundles that simplify project setup.
Example:

- spring-boot-starter-web → Web apps
- spring-boot-starter-data-jpa → JPA & Hibernate

19. How do you create a REST API in Spring Boot?

java

CopyEdit

```
@RestController

@RequestMapping("/api")

public class MyController {

    @GetMapping("/hello")

    public String sayHello() {

        return "Hello, World!";

    }

}
```

20. Difference between application.properties and application.yml

- application.properties → Key-value pairs
- application.yml → Hierarchical structure

Example:

yaml

CopyEdit

```
server:

  port: 8080
```

21. How does Spring Boot handle exceptions globally?

Use @ControllerAdvice and @ExceptionHandler.

Example:

java

CopyEdit

```
@ControllerAdvice

public class GlobalExceptionHandler {

    @ExceptionHandler(Exception.class)

    public ResponseEntity<String> handleException(Exception e) {

        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(e.getMessage());

    }

}
```

22. What is Spring Boot Actuator?

Spring Boot Actuator provides production-ready monitoring.

Enable by adding:

properties

CopyEdit

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

23. What is Spring MVC, and how does it work?

Spring MVC is a web framework that follows the **Model-View-Controller** (MVC) pattern.

24. Difference between @Controller and @RestController

@RestController is equivalent to @Controller + @ResponseBody.

25. What is @RequestMapping and how does it work?

Maps HTTP requests to controller methods.

Example:

java

CopyEdit

```
@RequestMapping("/hello")

public String hello() {

   return "Hello!";

}
```