1. What is the difference between JDK, JRE, and JVM?

- **JDK (Java Development Kit)**: Contains tools to develop Java applications (includes compiler, debugger, JRE).
- **JRE (Java Runtime Environment)**: Provides environment to run Java programs (includes JVM + libraries).
- **JVM (Java Virtual Machine)**: Executes Java bytecode on any platform.

Example:

- Writing and compiling code: needs **JDK**
- Running `.class` files: needs **JRE**
- Executing bytecode: done by **JVM**

---

2. Explain the concept of OOPs in Java.

OOPs = Object-Oriented Programming includes:

- **Encapsulation**: Wrapping data with methods (class)
- **Abstraction**: Hiding complex details
- **Inheritance**: Reusing code from a parent class
- **Polymorphism**: One function, many forms

Example:

```
java
CopyEdit
class Animal {
    void sound() { System.out.println("Animal sound"); }
}
class Dog extends Animal {
    void sound() { System.out.println("Bark"); }
}
```

---

3. What are the main features of Java 8?

- Lambda Expressions
- Stream API
- Functional Interfaces
- Default/Static Methods in Interfaces
- Optional Class
- New Date/Time API

Example:

```
java
CopyEdit
List<String> names = Arrays.asList("A", "B", "C");
names.forEach(name -> System.out.println(name)); // Lambda
```

4. What is the difference between == and `equals()`?

- `==`: Compares **reference/address**
- `equals()`: Compares **content**

Example:

```
java
CopyEdit
String a = new String("Hello");
String b = new String("Hello");
System.out.println(a == b);       // false
System.out.println(a.equals(b));  // true
```

5. What are access modifiers in Java?

- **public**: Accessible everywhere
- **private**: Only in the same class
- **protected**: Same package + subclass
- **default**: Same package

Example:

```
java
CopyEdit
public class A {
    private int x = 10;
    protected void show() { System.out.println("Hi"); }
}
```

6. Explain method overloading and overriding.

- **Overloading**: Same method name, different parameters (compile-time)
- **Overriding**: Subclass redefines parent's method (runtime)

Example (Overloading):

```
java
CopyEdit
void sum(int a, int b) {}
void sum(double a, double b) {}
```

Example (Overriding):

```
java
CopyEdit
class Parent {
    void show() { System.out.println("Parent"); }
}
```

```
class Child extends Parent {
    void show() { System.out.println("Child"); }
}
```

---

7. What is the difference between abstract class and interface?

- **Abstract class**: Can have both abstract and concrete methods
- **Interface**: Only abstract methods (Java 8 allows default methods)

Example:

```
java
CopyEdit
abstract class Animal {
    abstract void makeSound();
}

interface Flyable {
    void fly();
}
```

---

8. What is a constructor in Java?

Constructor is a special method that initializes an object.

Example:

```
java
CopyEdit
class Person {
    String name;
    Person(String n) {
        name = n;
    }
}
```

---

9. Explain the concept of inheritance.

Inheritance allows one class to acquire properties and methods of another.

Example:

```
java
CopyEdit
class Vehicle {
    void run() { System.out.println("Running"); }
}
class Bike extends Vehicle {}
```

---

10. What is the final keyword in Java?

- **final variable**: Value can’t change
- **final method**: Can’t override
- **final class**: Can’t extend

Example:

```
java
CopyEdit
final int speed = 90;
final class Bike {}
```

Would you like me to continue with the next section (**Exception Handling Q11–Q15**, etc.) in the same format?

You said:

Yes

ChatGPT said:

Great! Let’s continue with the next section:

Exception Handling (Q11–Q15)

11. What is the difference between checked and unchecked exceptions?

- **Checked exceptions**: Must be handled using try-catch or declared with `throws`. Checked at **compile-time**.
- **Unchecked exceptions**: Don't require explicit handling. Occur at **runtime**.

Examples:

```
java
CopyEdit
// Checked Exception
FileReader file = new FileReader("data.txt");  // IOException

// Unchecked Exception
int a = 5 / 0;  // ArithmeticException
```

---

12. How does try-with-resources work in Java?

Used to **automatically close resources** like files, sockets, etc. after use.

Example:

```
java
CopyEdit
```

```
try (BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {
    System.out.println(br.readLine());
} catch (IOException e) {
    e.printStackTrace();
}
```

Benefit: No need to explicitly call `close()`.

---

13. What is the use of finally block?

The `finally` block **always executes**, whether an exception occurs or not. Used for cleanup tasks.

Example:

```
java
CopyEdit
try {
    int x = 10 / 0;
} catch (Exception e) {
    System.out.println("Error");
} finally {
    System.out.println("Always runs");
}
```

---

14. What is custom exception in Java?

A custom exception is a **user-defined exception** class that extends `Exception` or `RuntimeException`.

Example:

```
java
CopyEdit
class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}
```

---

15. How do you create your own exception class?

You extend the `Exception` class and provide a constructor.

Example:

```
java
CopyEdit
class AgeException extends Exception {
    public AgeException(String msg) {
        super(msg);
    }
}

public class Test {
    public static void main(String[] args) {
        try {
            int age = 15;
            if (age < 18)
                throw new AgeException("Underage not allowed");
        } catch (AgeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Next section coming up: **Collections (Q16–Q20)**. Want me to continue now?

You said:

Yes

ChatGPT said:

Perfect! Let's move on to:

Collections (Q16–Q20)

16. What is the difference between ArrayList and LinkedList?

- **ArrayList**:
    - Uses a **dynamic array** internally
    - Fast for **searching**
    - Slow for **inserting/deleting** in the middle
- **LinkedList**:
    - Uses **doubly linked list**
    - Fast for **inserting/deleting**
    - Slow for **searching**

Example:

```
java
CopyEdit
List<String> list = new ArrayList<>();
list.add("Java");

List<String> linked = new LinkedList<>();
linked.add("Python");
```

---

17. How does HashMap work internally?

- HashMap stores key-value pairs.
- It uses the **hashCode()** method to calculate the **bucket index**.
- It resolves collisions using a **linked list** or **tree** (after Java 8).

Example:

```
java
CopyEdit
Map<String, String> map = new HashMap<>();
map.put("name", "John");  // "name".hashCode() used
```

---

18. What are the differences between HashSet and TreeSet?

Feature	HashSet	TreeSet
Order	No order	Sorted order
Performance	Faster (O(1))	Slower (O(log n))
Null	Allows one null element	Doesn't allow null

Example:

```
java
CopyEdit
Set<String> set1 = new HashSet<>();
Set<String> set2 = new TreeSet<>();
```

---

19. What is the difference between List and Set?

Feature	List	Set
Duplicates	Allows duplicates	No duplicates
Order	Maintains insertion order	Not guaranteed (except LinkedHashSet)
Implementations	ArrayList, LinkedList	HashSet, TreeSet

Example:

```
java
CopyEdit
List<String> names = new ArrayList<>();
Set<String> uniqueNames = new HashSet<>();
```

---

20. What is the use of Iterator in Java?

Iterator is used to **traverse elements** in a collection.

Methods:

- `hasNext()`: checks if there's a next element
- `next()`: returns next element

- `remove()`: removes current element

Example:

```
java
CopyEdit
List<String> list = Arrays.asList("A", "B", "C");
Iterator<String> it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

Next up: **Multithreading (Q21–Q25)**. Ready to continue?

You said:

Yes

ChatGPT said:

Awesome! Let’s continue with:

Multithreading (Q21–Q25)

21. What is the difference between Runnable and Callable?

Feature	Runnable	Callable
Return Value	No	Yes (uses `Future`)
Exceptions	Cannot throw checked exception	Can throw checked exception
Method	`run()`	`call()`

Example (Callable):

```
java
CopyEdit
Callable<Integer> task = () -> 123;
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<Integer> future = executor.submit(task);
System.out.println(future.get());  // Output: 123
```

---

22. What is the Executor framework in Java?

It provides a **thread pool** to manage multiple threads efficiently.

Key Classes:

- `ExecutorService`

- `Executors`
- `Future`

Example:

```
java
CopyEdit
ExecutorService executor = Executors.newFixedThreadPool(2);
executor.execute(() -> System.out.println("Task running"));
executor.shutdown();
```

---

23. What are synchronized methods and blocks?

They prevent **race conditions** by allowing only one thread to access a block of code at a time.

Example (synchronized block):

```
java
CopyEdit
public void printData() {
    synchronized(this) {
        System.out.println("Thread-safe code");
    }
}
```

---

24. What is the difference between wait() and sleep()?

Feature	`wait()`	`sleep()`
Lock	Releases lock	Doesn't release lock
Belongs to	`Object` class	`Thread` class
Usage	Inter-thread communication	Pause execution

Example:

```
java
CopyEdit
synchronized(obj) {
    obj.wait();  // Wait until notified
}
Thread.sleep(1000);  // Sleep for 1 second
```

---

25. Explain deadlock and how to avoid it.

- **Deadlock**: Two or more threads are waiting forever for each other to release locks.

Example:

```
java
```

```
CopyEdit
// Thread 1 locks A then waits for B
// Thread 2 locks B then waits for A
```

Avoidance Tips:

- Acquire locks in the same order.
- Use timeout with `tryLock()`.
- Use fewer synchronized blocks.

Next up: **Spring Boot Basics (Q26–Q30)**. Shall we continue?

You said:

Yes

ChatGPT said:

Great! Now let's move to:

Spring Boot Basics (Q26–Q30)

26. What is Spring Boot?

Spring Boot is a framework that simplifies building Java applications using the Spring Framework by providing:

- **Auto-configuration**
- **Embedded servers**
- **Starter dependencies**

Goal: Minimize boilerplate and configuration.

---

27. How does Spring Boot differ from Spring?

Feature	Spring Framework	Spring Boot
Setup	Manual	Auto-configured
Server	External (Tomcat, etc.)	Embedded (Tomcat/Jetty)
Configuration	XML or Java-based	Mostly `application.properties`
Deployment	WAR	JAR (self-contained)

---

28. What are starters in Spring Boot?

Starters are pre-configured dependency bundles.

Examples:

- `spring-boot-starter-web` → for building web apps and REST APIs
- `spring-boot-starter-data-jpa` → for working with databases using JPA
- `spring-boot-starter-test` → for unit and integration testing

---

29. What is `@SpringBootApplication` annotation?

This annotation is a shortcut for:

- `@Configuration` → marks class as configuration class
- `@EnableAutoConfiguration` → enables auto-configuration
- `@ComponentScan` → enables component scanning

Example:

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

---

30. How does auto-configuration work in Spring Boot?

Spring Boot checks:

- **Classpath** (which libraries are present)
- **Configuration files** (`application.properties` or `.yml`)
- Then it automatically configures beans using `@Conditional` annotations.

Benefit: You don’t need to write configuration code manually.

Example:
If `spring-boot-starter-web` is present, Spring Boot auto-configures:

- DispatcherServlet
- Jackson for JSON
- Embedded Tomcat

Next up: **Spring Boot Configuration and REST (Q31–Q35)**. Want to proceed?

You said:

Yes

ChatGPT said:

Perfect! Let's move on to:

Spring Boot Configuration & REST (Q31–Q35)

31. What are Spring Profiles?

Spring Profiles allow you to create multiple sets of configurations for different environments like **dev**, **test**, and **prod**.

How to use:

- Define in `application-dev.properties`, `application-prod.properties`
- Activate with:
  `spring.profiles.active=dev`

Example:

```
properties
CopyEdit
# application-dev.properties
server.port=8081

# application-prod.properties
server.port=8080
```

---

32. How do you handle exceptions in Spring Boot?

Use:

- `@ControllerAdvice` → for global exception handling
- `@ExceptionHandler` → to handle specific exceptions

Example:

```
java
CopyEdit
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handle(Exception e) {
        return new ResponseEntity<>("Error: " + e.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

---

33. What is the use of `application.properties` file?

Used to define configuration settings such as:

- Server port
- Database credentials
- Logging level
- Custom application variables

Example:

```
properties
CopyEdit
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.jpa.show-sql=true
```

---

34. How to create a REST API using Spring Boot?

Steps:

1. Use `@RestController`
2. Create mapping methods with `@GetMapping`, `@PostMapping`, etc.
3. Run application

Example:

```
java
CopyEdit
@RestController
@RequestMapping("/api")
public class HelloController {
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello World";
    }
}
```

---

35. What is the use of `@RestController` and `@RequestMapping`?

- `@RestController`: Combines `@Controller` and `@ResponseBody`. It tells Spring to return data directly (like JSON).
- `@RequestMapping`: Maps HTTP URLs to controller methods.

Example:

```
java
CopyEdit
@RestController
```

```
@RequestMapping("/user")
public class UserController {
    @GetMapping("/greet")
    public String greet() {
        return "Hello User";
    }
}
```

Next: **Spring Core & DI (Q36–Q40)**. Shall we continue?

You said:

Yes

ChatGPT said:

Awesome! Let's move ahead with:

Spring Core & Dependency Injection (Q36–Q40)

36. How does dependency injection work in Spring?

Dependency Injection (DI) is when Spring provides required objects (beans) to a class instead of creating them manually.

Types of DI:

- **Constructor Injection**
- **Setter Injection**
- **Field Injection (not recommended for testing)**

Example (Constructor Injection):

```
java
CopyEdit
@Component
public class StudentService {
    private final StudentRepository repo;

    @Autowired
    public StudentService(StudentRepository repo) {
        this.repo = repo;
    }
}
```

---

37. What is the difference between `@Component`, `@Service`, and `@Repository`?

Annotation	Used For	Extra Features
`@Component`	Generic Spring bean	None
`@Service`	Business logic layer	Semantic label
`@Repository`	DAO layer (DB access)	Converts DB exceptions to Spring exceptions

Example:

```
java
CopyEdit
@Component
class Helper {}

@Service
class UserService {}

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

---

38. Explain the Spring Bean lifecycle.

1. **Instantiation**
2. **Dependency Injection**
3. **`@PostConstruct` called** (custom init)
4. **Bean is ready for use**
5. **Context is closed**
6. **`@PreDestroy` called** (custom destroy)

Example:

```
java
CopyEdit
@Component
public class MyBean {
    @PostConstruct
    public void init() { System.out.println("Init called"); }

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

---

39. What is the use of `@Autowired` annotation?

- Used to **inject dependencies** automatically.
- Can be used on:
    - Constructor
    - Setter method
    - Field (not ideal for testing)

Example:

```
java
CopyEdit
@Autowired
private UserRepository repo;
```

---

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

Feature	Constructor Injection	Setter Injection
Use for	Required dependencies	Optional dependencies
Immutability	Yes	No
Object consistency	Guaranteed at construction time	Might be incomplete

Example (Setter):

```
java
CopyEdit
@Component
public class MyService {
    private MyRepo repo;

    @Autowired
    public void setRepo(MyRepo repo) {
        this.repo = repo;
    }
}
```

Next up: **Spring Data JPA & Hibernate (Q41–Q45)**. Want to go ahead?

You said:

Yes

ChatGPT said:

Great! Let's continue with:

Spring Data JPA & Hibernate (Q41–Q45)

41. What is the difference between JPA and Hibernate?

Feature	JPA (Java Persistence API)	Hibernate
Type	Specification/Interface	Implementation of JPA
Usage	Standard API	Framework that implements JPA
Dependency	Needs an implementation	Is an implementation

Example:
You use **JPA annotations** (`@Entity`, `@Table`) in your app, and Hibernate does the actual ORM work under the hood.

---

42. How does `@OneToMany` and `@ManyToOne` relationship work in JPA?

- **@OneToMany**: One entity has many of another entity
- **@ManyToOne**: Many entities refer to one parent entity

Example:

```
java
CopyEdit
@Entity
public class Department {
    @OneToMany(mappedBy = "department")
    List<Employee> employees;
}

@Entity
public class Employee {
    @ManyToOne
    @JoinColumn(name = "dept_id")
    Department department;
}
```

---

43. What is the use of `@Entity` and `@Table` annotations?

- `@Entity`: Marks a class as a JPA entity (mapped to a database table)
- `@Table`: (Optional) Specifies the actual table name

Example:

```
java
CopyEdit
@Entity
@Table(name = "students")
public class Student {
    @Id
    private Long id;
}
```

---

44. How does lazy and eager loading work in JPA?

- **Lazy loading**: Loads related entities **only when accessed**
- **Eager loading**: Loads related entities **immediately**

Example:

```
java
```

```
CopyEdit
@OneToMany(fetch = FetchType.LAZY)   // Load later
private List<Order> orders;

@OneToMany(fetch = FetchType.EAGER)  // Load immediately
private List<Order> orders;
```

---

45. What is the purpose of `EntityManager`?

- `EntityManager` is the **main interface** for interacting with persistence context.
- It's used for:
    - Persisting, removing, finding entities
    - Managing transactions

Example:

```
java
CopyEdit
entityManager.persist(user);
entityManager.find(User.class, 1L);
```

Note: In Spring Boot, you usually don't need to use `EntityManager` directly because of **Spring Data JPA** abstractions.

Next section: **Spring JPA Advanced & Pagination (Q46–Q50)**. Shall we continue?

You said:

Meaning of persisting

ChatGPT said:

Great question!

What is "Persisting" in JPA (Java Persistence API)?

Persisting means **saving a Java object to the database** so that it becomes a record (row) in a table.

In simple terms:

You create a Java object → Then **persist** it → It becomes a row in a database table.

Example:

```
java
CopyEdit
```

```
@Entity
public class Student {
    @Id
    private Long id;
    private String name;
}
java
CopyEdit
Student student = new Student();
student.setId(1L);
student.setName("Alice");

entityManager.persist(student); // This saves the student to the DB
```

After `persist()`, the object is in a **managed** state, and any changes to it will be synchronized with the database.

Summary for Beginners:

Term	Meaning
Persist	Save the object into the database
Persisted Object	A record in the database

Would you like a simple diagram or visual explanation as well?

You said:

Next section: Spring JPA Advanced & Pagination (Q46–Q50). Shall we continue?

ChatGPT said:

Perfect! Let's move on to:

Spring JPA Advanced & Pagination (Q46–Q50)

46. How do you write custom queries using `@Query` in Spring Data JPA?

- You can write custom **JPQL** or **native SQL** queries using `@Query` annotation in repository interfaces.

Example (JPQL):

```
java
CopyEdit
@Query("SELECT u FROM User u WHERE u.email = ?1")
User findByEmail(String email);
```

Example (Native SQL):

```
java
CopyEdit
@Query(value = "SELECT * FROM users WHERE email = ?1", nativeQuery = true)
User findByEmailNative(String email);
```

---

47. What are transactions and how are they managed in Spring?

- A **transaction** ensures that a set of database operations is executed **completely or not at all**.
- Managed using `@Transactional`.

Example:

```
java
CopyEdit
@Transactional
public void transferMoney(Account from, Account to, double amount) {
    from.withdraw(amount);
    to.deposit(amount);
}
```

If any exception occurs, the entire transaction is **rolled back** automatically.

---

48. What is cascading in JPA?

Cascading lets you apply operations (like persist, delete) **to related entities** automatically.

Example:

```
java
CopyEdit
@OneToMany(cascade = CascadeType.ALL)
private List<Order> orders;
```

When you **save or delete** the parent entity, it will automatically do the same for the child entities.

---

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

Use `Pageable` and `Sort` objects in your repository methods.

Example:

```
java
CopyEdit
Page<User> findAll(Pageable pageable);
```

Usage:

```
java
CopyEdit
PageRequest.of(pageNumber, pageSize, Sort.by("name").ascending());
```

Controller Example:

```
java
CopyEdit
@GetMapping("/users")
public Page<User> getUsers(@RequestParam int page, @RequestParam int size)
{
    return userRepository.findAll(PageRequest.of(page, size));
}
```

---

50. Explain the use of `CrudRepository` and `JpaRepository`.

Interface	Features
`CrudRepository`	Basic CRUD operations
`JpaRepository`	Extends CrudRepo + pagination, sorting

Example:

```
java
CopyEdit
public interface UserRepository extends JpaRepository<User, Long> {}
```

Use `JpaRepository` in most cases as it includes all `CrudRepository` methods plus more.

Next up: **Microservices with Spring Boot (Q51–Q55)**. Ready to go on?

You said:

Yes

ChatGPT said:

Awesome! Let's dive into:

Microservices with Spring Boot (Q51–Q55)

51. What are microservices?

Microservices are an architecture style where an application is built as a **collection of small, independent services**, each focusing on a single business capability.

Benefits:

- Easy to scale
- Independent deployment
- Better fault isolation

Example:

- `User Service`, `Order Service`, and `Payment Service` as separate Spring Boot apps.

---

52. How is Spring Boot used to create microservices?

Spring Boot makes it easy to:

- Build REST APIs
- Handle configurations with profiles
- Integrate with **Eureka** (discovery), **Config Server**, **Gateway**, etc.

Example:

```
java
CopyEdit
@RestController
@RequestMapping("/users")
public class UserController {
    @GetMapping("/{id}")
    public String getUser(@PathVariable int id) {
        return "User with id: " + id;
    }
}
```

Deploy as a separate service with its own port.

---

53. What is Spring Cloud?

Spring Cloud provides tools for:

- **Service Discovery (Eureka)**
- **API Gateway (Spring Cloud Gateway)**
- **Distributed Config (Config Server)**
- **Circuit Breaker (Resilience4j/Hystrix)**

It makes building cloud-native microservices easier.

---

54. How does service discovery work with Eureka?

Eureka Server acts as a **service registry**.

- Services (clients) **register** themselves with Eureka.
- Other services **discover** them via Eureka.

Key Annotations:

```
java
CopyEdit
@EnableEurekaServer      // On Eureka server app
@EnableEurekaClient      // On microservices
```

---

55. What is API Gateway and how does it work?

API Gateway is the **entry point** to all microservices.

- Routes incoming requests to the correct service
- Handles cross-cutting concerns: security, logging, throttling

Spring Cloud Gateway is used in Spring Boot.

Example Route Configuration:

```
yaml
CopyEdit
spring:
  cloud:
    gateway:
      routes:
        - id: user-service
          uri: http://localhost:8081
          predicates:
            - Path=/users/**
```

Next up: **Microservices Advanced Topics (Q56–Q60)**. Want to proceed?

You said:

Yes

ChatGPT said:

Great! Let's continue with:

Microservices Advanced Topics (Q56–Q60)

56. What is the Circuit Breaker pattern?

- It prevents your system from **trying to access a failing service repeatedly**.
- If a service is down, the circuit breaker **opens** and returns a fallback response.

Spring Boot Tools:

- **Resilience4j** (preferred)
- **Hystrix** (older, now deprecated)

Example (Resilience4j):

```
java
CopyEdit
@CircuitBreaker(name = "orderService", fallbackMethod = "fallbackOrder")
public String getOrders() {
    // Call external service
}

public String fallbackOrder(Throwable t) {
    return "Order Service is temporarily unavailable.";
}
```

---

57. What are Feign Clients?

Feign is a declarative REST client that lets you **call other microservices using interfaces**—no `RestTemplate` or `WebClient` needed.

Example:

```
java
CopyEdit
@FeignClient(name = "order-service")
public interface OrderClient {
    @GetMapping("/orders/{id}")
    Order getOrder(@PathVariable("id") Long id);
}
```

---

58. What is the role of Config Server?

Spring Cloud Config Server is used for **centralized configuration management** across all microservices.

- Stores properties in **Git**
- Microservices fetch configurations from the server

Example:

```
properties
CopyEdit
# in application.properties of the microservice
spring.config.import=optional:configserver:http://localhost:8888
```

---

59. How do you secure microservices?

Common ways:

- **JWT (JSON Web Token)**
- **OAuth2**
- **Spring Security**
- **API Gateway authentication**

JWT Example Flow:

1. Login → Generate token
2. Send token in `Authorization` header
3. Microservices validate token before processing

---

60. What is distributed tracing?

Used to **trace requests across microservices**, especially useful for debugging.

Tools:

- **Zipkin**
- **Jaeger**
- **Sleuth** (adds trace IDs to logs)

Spring Cloud Sleuth + Zipkin setup helps track request flow from service A → B → C.

Next section: **Advanced Topics (Q61–Q65)**. Would you like to continue?

You said:

Yes

ChatGPT said:

Excellent! Let's move forward with:

Advanced Topics (Q61–Q65)

61. What are design patterns used in Java and Spring?

Some commonly used design patterns are:

- **Singleton** – Ensures only one instance (e.g., Spring Beans by default)
- **Factory** – Creates objects without exposing creation logic
- **Builder** – Builds complex objects step by step
- **Proxy** – Used in AOP (e.g., method interception)
- **Observer** – Event-driven systems (e.g., ApplicationListeners)

---

62. Explain the Singleton pattern with an example.

Singleton ensures a class has **only one instance** and provides a global point of access.

Example (Plain Java):

```
java
CopyEdit
public class MySingleton {
    private static MySingleton instance = new MySingleton();

    private MySingleton() {}

    public static MySingleton getInstance() {
        return instance;
    }
}
```

In Spring, beans are Singleton by default:

```
java
CopyEdit
@Component  // By default, singleton scope
public class MyService {}
```

---

63. What is Dependency Injection and Inversion of Control?

- **Inversion of Control (IoC)**: The control of creating and managing objects is given to the Spring container.
- **Dependency Injection (DI)**: Dependencies (beans) are injected into classes instead of being created manually.

Example:

```
java
CopyEdit
@Component
public class ServiceA {
    private final ServiceB serviceB;

    @Autowired
    public ServiceA(ServiceB serviceB) {
        this.serviceB = serviceB;
    }
}
```

Spring creates `ServiceB` and injects it into `ServiceA`.

---

64. What is AOP in Spring?

AOP (Aspect-Oriented Programming) helps separate **cross-cutting concerns** (like logging, security, transactions) from business logic.

Core Concepts:

- **Aspect**: A class with cross-cutting logic
- **Advice**: Action taken at a join point (e.g., before, after)
- **Join point**: A point in execution (like method call)

Example:

```
java
CopyEdit
@Aspect
@Component
public class LoggingAspect {
    @Before("execution(* com.example.service.*.*(..))")
    public void logBefore() {
        System.out.println("Method is about to be called");
    }
}
```

---

65. Explain the use of Kafka or RabbitMQ in microservices.

- Both are **message brokers** used for **asynchronous communication** between services.

Feature	Kafka	RabbitMQ
Use case	High-throughput streaming	General-purpose messaging
Model	Pub-Sub (Log-based)	Message Queue (AMQP)

Benefits:

- Decouples services
- Handles retries and failures

Use Case Example:

- **Order Service** sends message → **Inventory Service** receives and updates stock

---

Next up: **Docker, Swagger & Deployment (Q66–Q70)**. Ready to continue?

You said:

Yes

ChatGPT said:

Great! Let’s continue with:

Docker, Swagger & Deployment (Q66–Q70)

66. What is Docker and how do you use it with Spring Boot?

Docker is a platform to package applications in lightweight, portable containers.

Why use Docker?

- Consistent environments (runs anywhere)
- Easy deployment

Dockerizing a Spring Boot App:

1. Create a `Dockerfile`:

```
Dockerfile
CopyEdit
FROM openjdk:17
COPY target/app.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

2. Build image:

```
bash
CopyEdit
docker build -t my-spring-app .
```

3. Run container:

```
bash
CopyEdit
docker run -p 8080:8080 my-spring-app
```

---

67. What is the difference between monolithic and microservices architecture?

Feature	Monolithic	Microservices
Structure	Single application	Multiple independent services
Deployment	One WAR/JAR	Each service deployed independently
Scalability	Scale whole app	Scale individual services
Fault isolation	Hard	Easier

---

68. What is Swagger and how do you integrate it?

Swagger (OpenAPI) is used to document and test REST APIs interactively.

Integration in Spring Boot:

- Add dependency:

```
xml
CopyEdit
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.15</version>
</dependency>
```

- Access API docs at:
  `http://localhost:8080/swagger-ui.html`

---

69. How do you monitor Spring Boot applications?

Common tools:

- **Spring Boot Actuator**: Health, metrics, info
- **Prometheus + Grafana**: Visual dashboards
- **ELK stack (Elasticsearch, Logstash, Kibana)**

Actuator Example:

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

Then visit:

```
bash
CopyEdit
http://localhost:8080/actuator/health
```

---

70. Explain the deployment of Spring Boot apps in cloud platforms.

Deployment options:

- **AWS EC2**: Package JAR → SSH → Run with `java -jar`
- **Heroku**: Git-based deployment
- **Docker on cloud**: Push Docker image to AWS ECS / GCP / Azure

- **Kubernetes (K8s)**: Use pods and services for orchestration

Steps (Generic):

1. Build `.jar`
2. Upload/deploy to cloud server
3. Set environment configs
4. Run the app

Next section: **Additional Questions (Q71–Q75)**. Shall we proceed?

You said:

Yes

ChatGPT said:

Perfect! Let's go through:

Additional Questions (Q71–Q75)

71. What is the difference between HashMap and ConcurrentHashMap?

Feature	`HashMap`	`ConcurrentHashMap`
Thread-safe	No	Yes
Synchronization needed	Manually (if needed)	Internally synchronized
Performance in threads	Poor	Excellent

Use Case: Use `ConcurrentHashMap` in multi-threaded environments.

Example:

```
java
CopyEdit
Map<String, String> map = new ConcurrentHashMap<>();
map.put("name", "Alice");
```

---

72. What are the new features in Java 17?

Key Java 17 features:

- Sealed Classes
- Pattern Matching for `switch` (preview)
- Enhanced Pseudo-Random Number Generators

- Removal of outdated APIs (like Applet)

Example (Sealed Class):

```
java
CopyEdit
public sealed class Vehicle permits Car, Bike {}

final class Car extends Vehicle {}
final class Bike extends Vehicle {}
```

---

73. What is a functional interface?

- An interface with **only one abstract method**.
- Used in **lambdas** and **method references**.

Example:

```
java
CopyEdit
@FunctionalInterface
interface Greeting {
    void sayHello();
}
```

Usage:

```
java
CopyEdit
Greeting g = () -> System.out.println("Hello");
g.sayHello();
```

---

74. What is the Stream API in Java?

Stream API is used for processing data in a functional style.

Features:

- Filter, map, reduce, collect
- Lazy evaluation
- Parallel processing

Example:

```
java
CopyEdit
List<String> names = List.of("Tom", "Sam", "John");
names.stream()
     .filter(n -> n.startsWith("J"))
     .forEach(System.out::println);
```

---

75. How do you handle CORS in Spring Boot?

CORS (Cross-Origin Resource Sharing) is used to allow requests from different domains.

Enable CORS (Simple Way):

```
java
CopyEdit
@CrossOrigin(origins = "http://localhost:3000")
@GetMapping("/data")
public String getData() {
    return "CORS Enabled";
}
```

Global CORS Configuration:

```
java
CopyEdit
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("*")
                .allowedMethods("*");
    }
}
```

Next: **REST API Design, Validation & Lombok (Q76–Q80)**. Shall we proceed?

You said:

Yes

ChatGPT said:

Great! Let’s go ahead with:

REST API Design, Validation & Lombok (Q76–Q80)

76. What is the difference between PUT and PATCH in REST?

Method	Purpose	Behavior
PUT	Full update	Replaces the entire resource
PATCH	Partial update	Modifies only specified fields

Example:

```
http
```

```
CopyEdit
PUT /users/1
{
  "name": "Alice",
  "email": "alice@mail.com"
}
http
CopyEdit
PATCH /users/1
{
  "email": "newemail@mail.com"
}
```

---

77. How can you version REST APIs?

Common versioning strategies:

1. **URI Versioning**:
    - `/api/v1/users`
2. **Request Parameter**:
    - `/users?version=1`
3. **Header Versioning**:
    - `Accept: application/vnd.company.v1+json`

Best Practice: URI versioning for simplicity.

---

78. How do you validate request bodies in Spring Boot?

Use:

- `@Valid` for triggering validation
- Bean validation annotations like `@NotNull`, `@Size`, `@Email`

Example:

```
java
CopyEdit
public class User {
    @NotBlank
    private String name;

    @Email
    private String email;
}
```

Controller:

```
java
CopyEdit
@PostMapping("/user")
public String createUser(@Valid @RequestBody User user) {
    return "User created";
}
```

---

79. What is Lombok and how does it help in Spring Boot development?

Lombok reduces boilerplate code by generating methods automatically using annotations.

Example:

```
java
CopyEdit
@Data  // Generates getters, setters, toString, etc.
@AllArgsConstructor
@NoArgsConstructor
public class Employee {
    private Long id;
    private String name;
}
```

Benefit: Cleaner, shorter code.

---

80. How do you create and consume a REST API in Spring Boot using RestTemplate?

Creating a REST API:

```
java
CopyEdit
@RestController
public class HelloController {
    @GetMapping("/greet")
    public String greet() {
        return "Hello";
    }
}
```

Consuming using RestTemplate:

```
java
CopyEdit
RestTemplate restTemplate = new RestTemplate();
String response = restTemplate.getForObject("http://localhost:8080/greet",
String.class);
System.out.println(response);
```

Note: For reactive apps, use `WebClient` instead of `RestTemplate`.

Next up: **Spring WebFlux, Security & JWT (Q81–Q85)**. Want to continue?

You said:

Yes

ChatGPT said:

Awesome! Let's now go through:

Spring WebFlux, Security & JWT (Q81–Q85)

81. What is WebClient in Spring WebFlux?

- `WebClient` is a **non-blocking**, **reactive** HTTP client used in Spring WebFlux.
- Replaces the older `RestTemplate`.

Example:

```
java
CopyEdit
WebClient webClient = WebClient.create();
String response = webClient.get()
    .uri("http://localhost:8080/greet")
    .retrieve()
    .bodyToMono(String.class)
    .block();

System.out.println(response);
```

---

82. How does Spring Security work?

Spring Security handles:

- Authentication (who are you?)
- Authorization (what can you access?)

Key Concepts:

- Security Filter Chain
- AuthenticationManager
- UserDetailsService

Example Setup:

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

```
                .and()
                .formLogin();
    }
}
```

---

83. How to configure JWT in Spring Boot?

Steps:

1. Generate JWT on login
2. Send JWT in `Authorization` header
3. Validate JWT in a filter

Example Filter:

```
java
CopyEdit
public class JwtFilter extends OncePerRequestFilter {
    protected void doFilterInternal(...) {
        String token = request.getHeader("Authorization");
        // Validate token and set authentication
    }
}
```

Dependencies: Use `jjwt` or `java-jwt` libraries.

---

84. What is the role of AuthenticationManager in Spring Security?

- `AuthenticationManager` authenticates user credentials.
- It delegates authentication to providers like `DaoAuthenticationProvider`.

Usage:

```
java
CopyEdit
Authentication auth = authManager.authenticate(
    new UsernamePasswordAuthenticationToken("user", "pass")
);
```

---

85. What is the difference between OAuth2 and JWT?

Feature	**OAuth2**	**JWT**
Type	Protocol	Token format (can be used in OAuth2)
Purpose	Authorization framework	Token for data exchange & identity
Storage	Server/session-based or JWT	Self-contained

- **OAuth2**: Delegates login to providers (Google, GitHub)

- **JWT**: Carries user info (claims) in a signed token

Next: **Testing, DevTools & Configuration (Q86–Q90)**. Shall we go on?

You said:

Yes

ChatGPT said:

Perfect! Let's move on to:

Testing, DevTools & Configuration (Q86–Q90)

86. How do you test a Spring Boot application?

Spring Boot supports:

- **Unit Testing**: Using JUnit + Mockito
- **Integration Testing**: Using `@SpringBootTest`

Example:

```
java
CopyEdit
@SpringBootTest
public class UserServiceTest {
    @Autowired
    private UserService userService;

    @Test
    void testFindUser() {
        assertEquals("John", userService.getUser(1).getName());
    }
}
```

---

87. What is MockMvc in Spring Boot testing?

- `MockMvc` allows you to **test controllers** without starting the full server.

Example:

```
java
CopyEdit
@WebMvcTest(UserController.class)
public class UserControllerTest {
    @Autowired
    private MockMvc mockMvc;
    @Test
    void testHello() throws Exception {
        mockMvc.perform(get("/hello"))
               .andExpect(status().isOk())
               .andExpect(content().string("Hello"));
    }
}
```

---

88. What are annotations used for testing in Spring Boot?

Annotation	Purpose
`@SpringBootTest`	Load full application context
`@WebMvcTest`	Test only controller layer
`@DataJpaTest`	Test repository layer
`@MockBean`	Add mock objects to the Spring context
`@Test`	Marks a test method (JUnit)

---

89. What is the role of DevTools in Spring Boot?

Spring Boot DevTools:

- Enables **automatic restart** when code changes
- Enables **LiveReload** in browser
- Provides **fast feedback loop** during development

Add dependency:

```
xml
CopyEdit
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
</dependency>
```

---

90. How do you externalize configuration in Spring Boot?

You can externalize configuration using:

- `application.properties` or `.yml`
- Command-line arguments
- Environment variables
- Spring Cloud Config Server

Example (properties file):

```
properties
```

```
CopyEdit
app.message=Welcome
```

Usage in code:

```
java
CopyEdit
@Value("${app.message}")
private String message;
```

Next up: **Async, Logging, Scopes, Interceptors, and More (Q91–Q95)**. Continue?

You said:

Yes

ChatGPT said:

Great! Let's continue with:

Async, Logging, Scopes & Interceptors (Q91–Q95)

91. What is the difference between synchronous and asynchronous communication?

Type	Synchronous	Asynchronous
Waits for reply	Yes	No
Use case	REST APIs, immediate responses	Messaging (Kafka, RabbitMQ)
Speed	Slower (waits)	Faster for non-blocking operations

Example:

- Sync: HTTP call to another service
- Async: Kafka event sent and forget

---

92. How do you use `@Async` in Spring Boot?

- `@Async` makes a method run **in a separate thread**.

Steps:

1. Enable async support:
   `@EnableAsync` in configuration
2. Use `@Async` on methods

Example:

```
java
CopyEdit
@Async
public void sendEmail() {
    // runs in background
}
```

---

93. What is the difference between `ResponseEntity` and `@ResponseBody`?

Annotation	Purpose
`@ResponseBody`	Converts return object to JSON
`ResponseEntity`	Full control over response (body + status + headers)

Examples:

Using `@ResponseBody`:

```
java
CopyEdit
@ResponseBody
@GetMapping("/hello")
public String hello() {
    return "Hello";
}
```

Using `ResponseEntity`:

```
java
CopyEdit
@GetMapping("/user")
public ResponseEntity<User> getUser() {
    return ResponseEntity.ok(new User("John"));
}
```

---

94. How do you log in a Spring Boot application?

Use `SLF4J` with `LoggerFactory`.

Example:

```
java
CopyEdit
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@RestController
public class MyController {
    private static final Logger logger =
LoggerFactory.getLogger(MyController.class);
```

```
    @GetMapping("/log")
    public String logExample() {
        logger.info("Info log message");
        logger.error("Error log message");
        return "Logged";
    }
}
```

---

95. What logging frameworks are commonly used in Spring Boot?

- **SLF4J** (API layer)
- **Logback** (default backend)
- **Log4j2** (alternative backend)
- **Commons Logging** (older)

Spring Boot default = SLF4J + Logback

Last stretch coming up: **Bean Scopes, Interceptors, Embedded Server & WebFlux (Q96–Q100)**. Shall we finish it up?

You said:

Yes

ChatGPT said:

Fantastic! Let's wrap up with the final set:

Bean Scopes, Interceptors, Server & WebFlux (Q96–Q100)

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

Scope	Description
`singleton`	One instance per Spring container (default)
`prototype`	New instance every time it's requested
`request`	One per HTTP request (web apps)
`session`	One per HTTP session
`application`	One per servlet context

Example:

```
java
CopyEdit
@Component
```

```
@Scope("prototype")
public class MyBean {}
```

---

97. What are interceptors in Spring Boot?

Interceptors are used to **intercept HTTP requests** before reaching the controller.

Use Cases:

- Logging
- Authentication
- Request modification

Example:

```
java
CopyEdit
public class MyInterceptor implements HandlerInterceptor {
    public boolean preHandle(...) {
        System.out.println("Intercepted!");
        return true;
    }
}

@Configuration
public class Config implements WebMvcConfigurer {
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new MyInterceptor());
    }
}
```

---

98. What is an embedded server in Spring Boot?

Spring Boot includes **embedded servers** like:

- **Tomcat** (default)
- **Jetty**
- **Undertow**

Benefit: No need to deploy WAR files to external servers. Just run the `.jar` file.

Run:

```
bash
CopyEdit
java -jar myapp.jar
```

---

99. What are advantages of using Spring Boot?

- Auto-configuration

- Embedded server
- Simplified dependency management (starters)
- Production-ready with **Actuator**
- Easy integration with Spring ecosystem
- Less boilerplate code

---

100. What is the difference between Spring MVC and Spring WebFlux?

Feature	Spring MVC	Spring WebFlux
Type	Synchronous (Servlet API)	Asynchronous, Non-blocking
Performance	Good for typical apps	Better under high concurrency
HTTP Client	`RestTemplate`	`WebClient`
Annotation API	Same (e.g., `@GetMapping`)	Same