# Interview Questions for 8+ Years Experience (Core Java, Spring Boot, Micr

## Core Java

Q: 1. What is the difference between an abstract class and an interface in Java?

A: An abstract class can have method implementations and member variables, while an interface can only have abstract methods (until Java 8, after which default and static methods are allowed). Abstract classes are used when classes share a common base, whereas interfaces are best for multiple inheritance.

Follow-up Q: Can a class implement multiple interfaces and extend an abstract class simultaneously?

A: Yes, a class can implement multiple interfaces and extend only one abstract class due to Java's single inheritance rule.

Q: 2. Explain the concept of Java Memory Model (JMM).

A: JMM defines how threads interact through memory and what behaviors are allowed in concurrent execution. It ensures visibility, atomicity, and ordering of variables.

Follow-up Q: How does the `volatile` keyword help in JMM?

A: `volatile` ensures that changes to a variable are visible to all threads immediately, and it prevents instruction reordering.

Q: 3. What is the difference between == and equals() in Java?

A: `==` compares object references while `equals()` compares the values or states of objects.

Follow-up Q: What happens if you use equals() on a null object?

A: It will throw a NullPointerException if the object is null.

Q: 4. How does Java achieve platform independence?

A: Java programs are compiled to bytecode which is executed by the Java Virtual Machine (JVM), making them platform-independent.

Follow-up Q: Does JVM provide complete abstraction from the underlying OS?

A: Not completely, JVM still interacts with OS for I/O operations, memory, and process management.

Q: 5. What is the role of the ClassLoader in Java?

A: ClassLoader loads classes into memory during runtime. There are three main types: Bootstrap, Extension, and Application ClassLoader.

Follow-up Q: Can we write our own custom ClassLoader?

A: Yes, we can extend ClassLoader to implement custom class-loading behavior.

## Spring Boot

Q: 1. What is Spring Boot and how is it different from the Spring Framework?

A: Spring Boot is a project built on the top of the Spring Framework. It simplifies the setup of new Spring applications by providing defaults and auto-configuration to reduce boilerplate code.

Follow-up Q: What is auto-configuration in Spring Boot?

A: It attempts to automatically configure your Spring application based on the jar dependencies you have added.

Q: 2. How does Spring Boot handle dependency management?

A: Spring Boot uses a parent POM that includes default versions of common dependencies, reducing conflicts and simplifying builds.

Follow-up Q: What is the role of spring-boot-starter-parent?

A: It provides default configurations and dependency versions, so you don't have to declare them explicitly.

Q: 3. What are Spring Boot starters?

A: Starters are a set of convenient dependency descriptors you can include in your application. For example, spring-boot-starter-web includes Tomcat, Spring MVC, etc.

Follow-up Q: Can you create a custom starter?

A: Yes, by creating a module with necessary dependencies and configurations, and publishing it as a reusable artifact.

Q: 4. What is Spring Boot Actuator?

A: Spring Boot Actuator provides production-ready features like monitoring, metrics, health checks, and more.

Follow-up Q: How do you expose custom health indicators?

A: By implementing the HealthIndicator interface and registering it as a bean.

Q: 5. How does Spring Boot manage embedded servers?

A: Spring Boot can run applications with embedded servers like Tomcat, Jetty, or Undertow, removing the need for external deployment.

Follow-up Q: How can you change the default embedded server?

A: Exclude the default server dependency and add your preferred server dependency in the POM or build.gradle.

## Microservices

Q: 1. What are microservices and how do they differ from monolithic architecture?

A: Microservices is an architectural style where applications are composed of small, independent services. Monoliths are single deployable units.

Follow-up Q: What challenges are faced in microservices that arent present in monoliths?

A: Challenges include distributed transactions, service discovery, inter-service communication, and deployment complexities.

Q: 2. How do you ensure communication between microservices?

A: Through REST APIs, messaging queues (RabbitMQ, Kafka), and service meshes.

Follow-up Q: What is the difference between synchronous and asynchronous communication?

A: Synchronous waits for response; asynchronous does not block and usually uses messaging systems.

Q: 3. What is the role of an API Gateway in microservices?

A: It handles request routing, authentication, rate limiting, and aggregating results from multiple services.

Follow-up Q: Name some tools for implementing API Gateway.

A: Zuul, Spring Cloud Gateway, Kong, NGINX.

Q: 4. What strategies do you use for service discovery?

A: Eureka, Consul, or Kubernetes DNS are commonly used for service discovery in microservices.

Follow-up Q: How does client-side discovery differ from server-side discovery?

A: Client-side discovery involves the client querying the registry. Server-side discovery delegates this to a router/load balancer.

Q: 5. How do you handle failure in microservices architecture?

A: Use of circuit breakers, retries, timeouts, and fallbacks (e.g., Hystrix or Resilience4j).

Follow-up Q: How does a circuit breaker help in failure handling?

A: It prevents calls to a failing service, allowing it time to recover and protecting the system.

## Multithreading

Q: 1. What is the difference between a process and a thread?

A: A process is an independent executing program, while a thread is a lightweight subprocess sharing memory space with other threads.

Follow-up Q: Can threads from different processes communicate directly?

A: No, threads can only directly communicate within the same process.

Q: 2. Explain the life cycle of a thread in Java.

A: New -> Runnable -> Running -> Blocked/Waiting -> Terminated.

Follow-up Q: What happens if we call start() twice on the same thread?

A: It throws IllegalThreadStateException.

Q: 3. How does synchronized keyword work in Java?

A: It restricts access to a block/method to one thread at a time, preventing race conditions.

Follow-up Q: What is the difference between class-level and object-level lock?

A: Class-level lock synchronizes on the class object; object-level lock synchronizes on the instance.

Q: 4. What are the differences between ExecutorService and creating threads manually?

A: ExecutorService manages a pool of threads efficiently, reducing overhead and improving resource usage.

Follow-up Q: How do you shut down an ExecutorService properly?

A: Use shutdown() or shutdownNow() methods and awaitTermination() to block until all tasks complete.

Q: 5. What are deadlock and how can you prevent it?

A: Deadlock occurs when two or more threads wait indefinitely for resources locked by each other. Prevent by avoiding nested locks, using lock ordering, or tryLock.

Follow-up Q: How does tryLock help in preventing deadlocks?

A: tryLock times out instead of waiting indefinitely, so threads can avoid deadlock by backing off and retrying.