Comprehensive Java Interview Preparation Guide

This guide is organized by topic with questions at Basic, Intermediate, Advanced levels, and Scenario-based questions for each category.

1. JVM Architecture and Garbage Collection

Basic Level
1. What is JVM and how does it enable "write once, run anywhere"?
2. Explain the Java execution process from source code to execution.
3. What are the different memory areas in the JVM?
4. Explain the difference between stack memory and heap memory.
5. What is a classloader and what are its types?
6. What is garbage collection and why is it important?
7. What is the difference between JDK, JRE, and JVM?
8. What happens during class loading?
9. What are the different types of references in Java?
10. What is bytecode and why is it important?

Intermediate Level
1. Explain the different generations in Java garbage collection.
2. What is JIT compilation and how does it improve performance?
3. What is the "Stop the world" pause in garbage collection?
4. Explain the different types of garbage collectors (Serial, Parallel, CMS, G1).
5. What is metaspace in JVM and how is it different from PermGen?
6. What is the execution engine in JVM?
7. How does the JVM handle method calls?
8. What are JVM tuning parameters? Give examples of important ones.
9. How does class loading work in a hierarchical manner?
10. What is a memory leak in Java and how can it be prevented?

Advanced Level
1. Explain how to implement a custom classloader and when you would need one.
2. Deep dive into G1 garbage collector - how does it work and when should it be used?
3. What is the Z Garbage Collector (ZGC) and what advantages does it offer?
4. How would you tune the JVM for a high-throughput vs. low-latency application?
5. Explain the concept of Escape Analysis in JVM optimization.
6. How does the JVM implement thread synchronization at the hardware level?
7. What's the difference between ClassNotFoundException and NoClassDefFoundError?
8. How does the JVM handle OutOfMemoryError? What strategies can you use to diagnose it?
9. Explain the JVM's Just-In-Time compilation strategies in detail.
10. Compare and contrast different JVM implementations (HotSpot, OpenJ9, etc.)

Scenario-based Questions
1. Your application is experiencing frequent full GC pauses. How would you diagnose and fix this issue?
2. You notice your application is throwing OutOfMemoryError despite having sufficient heap

configured. What could be causing this and how would you troubleshoot?
3. How would you analyze a heap dump to identify memory leaks in a production application?
4. You observe that application performance degrades over time. How would you determine if this is JVM-related?
5. Design a system that needs to load classes dynamically at runtime based on configuration. How would you approach it?
6. Your microservices application has services with different resource needs. How would you configure JVM parameters differently for each?
7. You need to implement class versioning in a long-running application. How would you handle class evolution with custom classloaders?
8. How would you optimize JVM settings for a containerized application with limited resources?
9. Your application needs to maintain high throughput during GC. Which collector would you choose and how would you configure it?
10. Your application creates many short-lived objects. How would you tune the JVM to optimize for this pattern?

2. OOPs Concepts

Basic Level
1. What are the four main principles of OOP?
2. Explain encapsulation with an example. How is it achieved in Java?
3. What is inheritance? What are its advantages and types?
4. What is polymorphism and what are its types?
5. Explain the difference between an abstract class and an interface.
6. What are access modifiers in Java and what does each one do?
7. What is the difference between method overloading and method overriding?
8. Can abstract classes have constructors? Why or why not?
9. What is the super keyword used for?
10. What is the this keyword used for?

Intermediate Level
1. What are default methods in interfaces and why were they introduced in Java 8?
2. Can an interface have private methods? Since which Java version?
3. What is the diamond problem in multiple inheritance? How does Java handle it?
4. What happens if a method has the same signature but different return types in parent and child classes?
5. Explain the contract between equals() and hashCode() methods.
6. What is the purpose of the final keyword when applied to classes, methods, and variables?
7. What is the difference between composition and inheritance? When should you use each?
8. What is covariant return type in method overriding?
9. Can static methods be overridden? Explain.
10. What are marker interfaces? Give examples and explain their purpose.

Advanced Level
1. How does Java implement runtime polymorphism internally (method dispatch table)?
2. What is the Liskov Substitution Principle? Provide an example of a violation.
3. Explain deep vs. shallow copying of objects. How would you implement a deep copy?
4. How would you design immutable classes? What are the benefits?
5. What is the difference between aggregation and composition?

6. Explain method hiding in Java and how it differs from overriding.
7. What are the design considerations when creating an abstract class versus an interface?
8. How does the concept of bounded type parameters in generics relate to inheritance?
9. How does Java handle multiple inheritance through interfaces? What issues does this solve?
10. How would you implement dependency injection without using a framework?

Scenario-based Questions
1. Design a payment processing system that can handle multiple payment methods (credit card, PayPal, cryptocurrency) using OOP principles.
2. You need to model a shape hierarchy. How would you implement area and perimeter calculations using polymorphism?
3. Design a logging system that can log to different destinations (file, console, network) using OOP principles.
4. You're refactoring a codebase where inheritance is overused, causing maintenance issues. How would you apply composition instead?
5. Design a notification system that supports email, SMS, push notifications while making it easy to add new notification types.
6. Implement a plugin architecture for an application that allows third-party extensions without modifying core code.
7. Design a cache system that can work with different eviction policies (LRU, FIFO, etc.) using OOP principles.
8. You have a class with complex initialization requiring many parameters. How would you refactor it using the Builder pattern?
9. Design a validation system for different types of documents (invoices, contracts, applications) with different validation rules.
10. Implement a system for handling different file formats (CSV, JSON, XML) with parsing and writing capabilities.

3. Core Java

Basic Level
1. What are primitive data types in Java? List them with their sizes.
2. Explain the difference between == and equals() when comparing objects.
3. What is the difference between String, StringBuilder, and StringBuffer?
4. What is autoboxing and unboxing?
5. What is the static keyword and where can it be used?
6. What is the difference between an instance variable and a class variable?
7. What are the different ways to create objects in Java?
8. Explain the concept of pass-by-value in Java.
9. What happens when the main method is declared as private?
10. What are wrapper classes in Java?

Intermediate Level
1. What is the String constant pool? How does the intern() method work?
2. What is the difference between final, finally, and finalize()?
3. Explain the concept of variable scope in Java.
4. What is a static block? When is it executed?
5. What is the purpose of the transient keyword?
6. Explain the concept of initialization blocks (static and non-static).

7. What is the difference between checked and unchecked exceptions?
8. What is the try-with-resources statement?
9. What is the Integer cache in Java? What range of values does it cache?
10. What is the difference between shallow copy and deep copy?

Advanced Level
1. Explain the concept of classloading in Java. What are the different phases?
2. How does the hashCode() method work internally?
3. What is reflection in Java? Give examples of its usage.
4. What are weak references, soft references, and phantom references?
5. Explain the double-checked locking pattern. What are its pitfalls?
6. What is the Initialization-on-demand holder idiom?
7. How would you implement object cloning in Java? What are the various approaches?
8. What is the concept of Defensive Copying? When would you use it?
9. What is the difference between a static inner class and a non-static inner class?
10. How would you create a custom annotation in Java? What are retention policies?

Scenario-based Questions
1. You need to implement a cache with automatic expiration of entries. How would you design it?
2. You're working with a String that needs many modifications in a multi-threaded environment. Which class would you use and why?
3. You need to implement a memory-sensitive cache. How would you use reference types to allow GC to reclaim memory?
4. You need to design a class that must be immutable. What steps would you take?
5. You need to implement a singleton pattern that is thread-safe but also lazy-initialized. How would you do it?
6. You have a method that performs an expensive operation. How would you implement caching for this method?
7. You need to design a configuration class that ensures type safety. How would you use generics for this?
8. You need to create a deep copy of a complex object graph. How would you implement this?
9. You have code with many null checks. How would you refactor it using Optional?
10. You need to read sensitive information (like passwords) that shouldn't remain in memory. How would you handle this in Java?

4. Exception Handling

Basic Level
1. What is an exception in Java?
2. What is the difference between checked and unchecked exceptions?
3. Explain the exception hierarchy in Java.
4. What is the purpose of the try-catch-finally block?
5. What is the difference between throw and throws keywords?
6. What happens if an exception occurs in a finally block?
7. What are some common unchecked exceptions?
8. What are some common checked exceptions?
9. Can you have a try block without a catch block?
10. What is the purpose of the try-with-resources statement?

Intermediate Level
1. What is exception chaining? How is it implemented?
2. How would you create a custom exception?
3. Explain multi-catch block introduced in Java 7.
4. What happens when a method throws an exception that is not caught?
5. How does exception handling affect method overriding?
6. What is the AutoCloseable interface and how is it used?
7. What is the difference between Error and Exception?
8. How are exceptions handled in a multi-threaded environment?
9. What are the guidelines for creating custom exceptions?
10. Explain the concept of exception suppression in try-with-resources.

Advanced Level
1. What is the performance impact of exceptions?
2. How would you design a proper exception hierarchy for your application?
3. What are best practices for logging exceptions?
4. What is the concept of exception tunneling?
5. How do you handle exceptions in lambda expressions?
6. How do you unit test exception handling code?
7. How would you implement a retry mechanism for operations that might throw exceptions?
8. What is the concept of try-finally vs. try-with-resources in terms of performance?
9. How do you handle errors versus exceptions in a robust application?
10. What strategies can you use to prevent exceptions rather than handling them?

Scenario-based Questions
1. You're designing a REST API. How would you translate exceptions into appropriate HTTP responses?
2. You have a legacy codebase with poor exception handling. How would you refactor it?
3. You need to implement a method that reads from a file and writes to a database. How would you handle potential exceptions?
4. You're experiencing performance issues in your application due to excessive exception throwing. How would you fix it?
5. You need to implement a validation framework. How would you effectively use exceptions?
6. You're implementing a library that will be used by other developers. How would you design your exception architecture?
7. Your application needs to recover from certain types of failures automatically. How would you implement this?
8. You need to implement a circuit breaker pattern to handle external service failures. How would you use exceptions?
9. You have operations that need to be rolled back if an exception occurs. How would you implement this?
10. You need to handle exceptions across multiple layers of your application. What strategy would you use?

5. Collections Framework

Basic Level
1. What is the Java Collections Framework? Explain its hierarchy.
2. What is the difference between Collection and Collections?

3. What is the difference between ArrayList and LinkedList?
4. What is the difference between HashSet and TreeSet?
5. What is the difference between HashMap and Hashtable?
6. What is an Iterator and how does it differ from Enumeration?
7. What is the Comparable interface and how is it used?
8. What is the difference between List, Set, and Queue interfaces?
9. How does LinkedList work internally?
10. How does ArrayList work internally?

Intermediate Level
1. What is the internal structure of HashMap? How does it handle collisions?
2. What is the load factor in HashMap?
3. What is the difference between fail-fast and fail-safe iterators?
4. What is the time complexity of common operations on ArrayList, LinkedList, HashSet, and HashMap?
5. What is the Comparator interface? How is it different from Comparable?
6. What happens if you try to add elements to a Collection while iterating over it?
7. What is the ConcurrentModificationException and when does it occur?
8. What are synchronized collections in Java? Give examples.
9. What is the difference between Vector and ArrayList?
10. What is the LinkedHashMap class and when would you use it?

Advanced Level
1. Explain how HashMap handles collisions before and after Java 8.
2. What is the difference between HashMap and ConcurrentHashMap?
3. How would you implement a thread-safe collection?
4. What is the TreeMap class and when would you use it?
5. How would you implement a custom collection class?
6. What are concurrent collections in Java? Give examples.
7. What are the different ways to iterate over a collection?
8. What is the difference between Iterator and ListIterator?
9. How would you design a LRU cache using Java collections?
10. What is the difference between Deque and Queue interfaces?

Scenario-based Questions
1. You need to store elements in insertion order but also need fast access by key. Which collection would you use?
2. You're experiencing performance issues with HashMap due to poor hash function distribution. How would you fix it?
3. You need to implement a cache with a size limit where least recently used items are removed. Which collection would you use?
4. You need frequent insertions and deletions in the middle of a large list. Which collection type would be most efficient?
5. You need to maintain a collection of elements sorted by natural ordering. Which collection would you use?
6. You need to implement a queue where consumers process elements in priority order. Which collection would you use?
7. You need to implement a system that preserves the order of insertion but also needs fast lookup. Which collection would you use?
8. You need a thread-safe collection that doesn't block for reads. Which collection would you

choose?
9. You need to implement a custom Map with case-insensitive String keys. How would you approach this?
10. You need to store millions of elements and frequently check if elements exist. Which collection would be most memory-efficient?

6. Java 8 Features

Basic Level
1. What are the major features introduced in Java 8?
2. What is a lambda expression? Give a simple example.
3. What is a functional interface?
4. What is the purpose of the default methods in interfaces?
5. What is the purpose of the static methods in interfaces?
6. What is the Stream API?
7. What is the difference between intermediate and terminal operations in streams?
8. What is a method reference? Give an example.
9. What is the purpose of the Optional class?
10. What is the forEach method in Java 8?

Intermediate Level
1. What are the main functional interfaces in Java 8 (Function, Predicate, Consumer, Supplier)?
2. How do you handle exceptions in lambda expressions?
3. What is the difference between map and flatMap operations in Stream?
4. What is the difference between sequential and parallel streams?
5. How does the reduce operation work in streams?
6. What is the difference between collect and reduce in streams?
7. How does the groupingBy collector work?
8. What is the purpose of the @FunctionalInterface annotation?
9. What are the different types of method references?
10. What is the difference between orElse() and orElseGet() in Optional?

Advanced Level
1. How would you implement a custom collector for the Stream API?
2. What is the difference between stateful and stateless intermediate operations?
3. How does the parallel stream work internally? What is the ForkJoinPool?
4. What are the performance implications of using streams vs traditional loops?
5. How would you debug a stream pipeline?
6. What are the limitations of lambda expressions?
7. How would you implement a custom functional interface?
8. How would you test code that uses lambda expressions?
9. What is the spliterator in Java 8?
10. How does method reference resolution work under the hood?

Scenario-based Questions
1. You have a list of objects and need to transform them, filter some out, and collect the results. Write a stream pipeline for this.
2. You need to process a large dataset in parallel. How would you use parallel streams

effectively?
3. You have a complex object transformation logic. Would you use a lambda expression or a method reference? Why?
4. You need to implement a retry mechanism with exponential backoff. How would you use functional interfaces for this?
5. You have code with many null checks. How would you refactor it using Optional?
6. You need to group a collection of objects by multiple criteria. How would you implement this using collectors?
7. You need to implement a custom sorting logic for a collection. How would you use lambda expressions for this?
8. You have a computation-heavy operation that you want to execute asynchronously. How would you use CompletableFuture for this?
9. You need to process a stream of data while maintaining state. How would you implement this?
10. You need to implement a pipeline of operations where each step depends on the result of the previous step. How would you use functional interfaces for this?

7. Multithreading and Concurrency

Basic Level
1. What is a thread in Java? How is it different from a process?
2. What are the different ways to create a thread in Java?
3. What is the lifecycle of a thread?
4. What is the difference between start() and run() methods?
5. What is a synchronized block?
6. What is the volatile keyword and when would you use it?
7. What is thread safety?
8. What is a deadlock? How can it be prevented?
9. What is a race condition? Give an example.
10. What is thread priority? How does it affect thread scheduling?

Intermediate Level
1. What is the difference between the wait() and sleep() methods?
2. What is the Executor framework?
3. What is a ThreadPool? What are the different types in Java?
4. What is the purpose of the join() method?
5. What is a daemon thread?
6. What is the difference between synchronized methods and synchronized blocks?
7. What is the purpose of the ReentrantLock class?
8. What is a ThreadLocal and when would you use it?
9. What is the CyclicBarrier class used for?
10. What is the CountDownLatch class used for?

Advanced Level
1. What is the Fork/Join framework?
2. What are atomic variables and how do they work?
3. What is the Semaphore class used for?
4. What is the Phaser class?
5. What is the CompletableFuture class and how is it used?
6. What is the difference between Callable and Runnable?

7. What is StampedLock and how is it different from ReadWriteLock?
8. What is lock striping?
9. How do you implement non-blocking algorithms?
10. What is the happens-before relationship in Java Memory Model?

Scenario-based Questions
1. You need to implement a producer-consumer pattern. How would you do it using Java's concurrency utilities?
2. You have a resource that multiple threads need to access, but you want to limit concurrent accesses. How would you implement this?
3. You need to implement a task that waits for multiple other tasks to complete before it starts. How would you do this?
4. You have a system where multiple threads are causing deadlocks. How would you diagnose and fix this?
5. You need to implement a cache that can be safely accessed by multiple threads. How would you design it?
6. You need to perform a computation-intensive task that can be broken down into smaller tasks. How would you use the Fork/Join framework?
7. You have a service that makes HTTP calls to an external API. How would you implement rate limiting using concurrency utilities?
8. You need to implement a thread pool that prioritizes certain types of tasks. How would you approach this?
9. You have a UI application that needs to perform background tasks without freezing the UI. How would you implement this?
10. You need to implement a system where multiple readers can access a resource simultaneously, but writers need exclusive access. How would you implement this?

8. Design Patterns

Basic Level
1. What is a design pattern? What are the benefits of using design patterns?
2. What are the three categories of design patterns?
3. What is the Singleton pattern? How do you implement it?
4. What is the Factory pattern? When would you use it?
5. What is the Builder pattern? What problem does it solve?
6. What is the Decorator pattern? Give an example.
7. What is the Strategy pattern? When would you use it?
8. What is the Observer pattern? Give a real-world example.
9. What is the Adapter pattern? When would you use it?
10. What is the Composite pattern? When would you use it?

Intermediate Level
1. What is the difference between Factory Method and Abstract Factory patterns?
2. How do you implement a thread-safe Singleton pattern?
3. What are the use cases for the Builder pattern?
4. What is the Chain of Responsibility pattern?
5. What is the Command pattern? When would you use it?
6. What is the Iterator pattern? How does the Java Collections Framework use it?
7. What is the Mediator pattern? When would you use it?

8. What is the State pattern? How is it different from Strategy pattern?
9. What is the Template Method pattern?
10. What is dependency injection? How does it relate to design patterns?

Advanced Level
1. What is the Bridge pattern and when would you use it?
2. What is the Flyweight pattern? What problem does it solve?
3. What is the Visitor pattern? When would you use it?
4. What is the Interpreter pattern?
5. What is the Prototype pattern? How does it differ from cloning?
6. What are anti-patterns? Give examples.
7. How would you implement dependency injection without a framework?
8. What is the difference between the Proxy pattern and the Decorator pattern?
9. How does the MVC pattern relate to other design patterns?
10. What is the difference between the Strategy pattern and the State pattern?

Scenario-based Questions
1. You need to create instances of different classes based on some input parameter. Which design pattern would you use and how?
2. You need to add functionality to an object dynamically without affecting other instances. How would you implement this?
3. You have a complex object creation process. How would you use the Builder pattern to simplify it?
4. You need to ensure a class has only one instance with thread-safety and lazy initialization. How would you implement this?
5. You need to define a family of algorithms and make them interchangeable. How would you implement this?
6. You need to convert the interface of a class into another interface clients expect. Which pattern would you use?
7. You need to notify multiple objects when another object's state changes. How would you implement this?
8. You have a complex conditional logic that determines object behavior. How would you refactor it using patterns?
9. You need to provide a way to access elements sequentially without exposing the underlying structure. Which pattern would you use?
10. You need to allow objects to vary their behavior when their state changes. How would you implement this?

9. SOLID Principles

Basic Level
1. What are the SOLID principles?
2. What is the Single Responsibility Principle?
3. What is the Open/Closed Principle?
4. What is the Liskov Substitution Principle?
5. What is the Interface Segregation Principle?
6. What is the Dependency Inversion Principle?
7. Why are SOLID principles important?
8. What are the benefits of following the Single Responsibility Principle?

9. What are the signs that a class is violating the Single Responsibility Principle?
10. How does SOLID improve code maintainability?

Intermediate Level
1. How do you identify if a class is violating the Open/Closed Principle?
2. What is the contract that the Liskov Substitution Principle enforces?
3. What are the signs that an interface is violating the Interface Segregation Principle?
4. How does dependency injection help in implementing the Dependency Inversion Principle?
5. What is the relationship between SOLID principles and design patterns?
6. How do SOLID principles improve testability?
7. What are the trade-offs of strictly following SOLID principles?
8. How do you refactor code to adhere to the Single Responsibility Principle?
9. How do you apply the Dependency Inversion Principle in a legacy codebase?
10. How do SOLID principles relate to object-oriented design?

Advanced Level
1. How do you balance the Open/Closed Principle with YAGNI (You Aren't Gonna Need It)?
2. What is the relationship between the Liskov Substitution Principle and behavioral subtyping?
3. How do SOLID principles relate to functional programming?
4. What is the impact of applying SOLID principles on the architecture of a system?
5. How do you apply SOLID principles in a microservices architecture?
6. What is the relationship between SOLID principles and other design principles like DRY and KISS?
7. How do you apply the Interface Segregation Principle in a language that doesn't support interfaces?
8. What are the common anti-patterns that violate SOLID principles?
9. How do you measure adherence to SOLID principles in a codebase?
10. How do you balance the application of SOLID principles with performance requirements?

Scenario-based Questions
1. You have a class that handles user authentication, data validation, and database operations. How would you refactor it to follow SRP?
2. You have a system that needs to support multiple types of reporting formats (PDF, Excel, HTML). How would you design it following OCP?
3. You have a parent class and a child class, but the child class behaves differently in some methods. Is this a violation of LSP? How would you fix it?
4. You have a large interface with many methods, but most implementing classes only use a subset. How would you refactor it to follow ISP?
5. You have a high-level module that depends directly on low-level modules. How would you refactor it to follow DIP?
6. You're designing a payment processing system that needs to support multiple payment methods. How would you apply SOLID principles?
7. You're reviewing code and notice that changes to one class often require changes to many other classes. Which SOLID principle is being violated?
8. You have a class hierarchy where child classes are not fully substitutable for their parent classes. How would you identify and fix LSP violations?
9. You're designing a plugin architecture. How would you use SOLID principles to ensure the system is extensible?
10. You have a system where objects are tightly coupled. How would you use DIP to improve testability and maintainability?

10. Spring Framework

Basic Level
1. What is Spring Framework? What are its core modules?
2. What is Inversion of Control (IoC) and Dependency Injection (DI)?
3. What are the different types of dependency injection?
4. What is a Spring Bean?
5. What is the Spring IoC container?
6. What is the difference between BeanFactory and ApplicationContext?
7. What is the lifecycle of a Spring Bean?
8. What is the purpose of the @Component annotation?
9. What is the purpose of the @Autowired annotation?
10. What is the difference between @Component, @Repository, @Service, and @Controller annotations?

Intermediate Level
1. What is Spring Boot auto-configuration?
2. What are Spring Boot starters?
3. What is Spring AOP?
4. What are the different types of advice in Spring AOP?
5. What is the purpose of the @Transactional annotation?
6. What are the bean scopes in Spring?
7. What is the purpose of the @Qualifier annotation?
8. What is the Spring MVC framework?
9. What is the purpose of the DispatcherServlet?
10. What is Spring Data JPA?

Advanced Level
1. How does Spring handle bean circular dependencies?
2. What is Spring Boot Actuator?
3. How would you handle exceptions in a Spring MVC application?
4. What is the difference between @RequestParam, @PathVariable, and @RequestBody?
5. What is the purpose of the @EnableAutoConfiguration annotation?
6. How would you implement custom validation in Spring?
7. What is Spring Security and how does it work?
8. What are the best practices for designing RESTful APIs with Spring?
9. What is the Richardson Maturity Model?
10. How would you implement caching in a Spring application?

Scenario-based Questions
1. You need to implement a service that depends on another service with multiple implementations. How would you handle this?
2. You need to implement transactional behavior for certain methods. How would you do this?
3. You need to implement cross-cutting concerns like logging and security. How would you use Spring AOP?
4. You have a Spring Boot application and need to customize its configuration. What approaches would you take?
5. You need to implement pagination for a REST API returning large datasets. How would you

implement this?
6. You need to configure different behaviors based on the environment (dev, test, prod). How would you do this?
7. You need to implement authentication and authorization. How would you use Spring Security?
8. You need to implement CRUD operations for a domain entity. How would you use Spring Data JPA?
9. You need to implement a REST API that handles file uploads. How would you approach this?
10. You have a Spring Boot application that needs to communicate with another microservice. How would you implement this?

11. Microservices with Spring Boot

Basic Level
1. What is a microservice architecture?
2. What are the advantages and disadvantages of microservices?
3. What is Spring Boot and how does it simplify microservice development?
4. What is service discovery? Why is it important?
5. What is Netflix Eureka?
6. What is an API Gateway?
7. What is the Circuit Breaker pattern?
8. What is the Feign client?
9. What is Spring Cloud Config?
10. What is Spring Boot Actuator?

Intermediate Level
1. How do microservices communicate with each other?
2. What is the difference between synchronous and asynchronous communication in microservices?
3. What is the role of Netflix Ribbon?
4. What is Hystrix? How does it implement the Circuit Breaker pattern?
5. What is Spring Cloud Gateway?
6. How do you handle distributed transactions in microservices?
7. What is the Saga pattern?
8. How do you implement service discovery with Eureka?
9. What is Spring Cloud Sleuth?
10. What is the purpose of Spring Cloud Config Server?

Advanced Level
1. How would you implement distributed tracing in a microservice architecture?
2. How would you handle data consistency in a microservice architecture?
3. What are the different deployment strategies for microservices?
4. What is a service mesh? How does it benefit microservices?
5. What is the role of event-driven architecture in microservices?
6. How would you implement authentication and authorization across microservices?
7. What are the challenges in testing microservices?
8. How would you handle rate limiting in a microservice architecture?
9. How would you implement fault tolerance in microservices?
10. What are the monitoring and observability considerations for microservices?

Scenario-based Questions
1. You need to deploy microservices across multiple regions. How would you handle service discovery?
2. You have a microservice that depends on several other services. How would you implement resilience?
3. You need to implement asynchronous communication between microservices. What approach would you take?
4. You need microservices to scale independently based on load. How would you design this?
5. You need to implement authentication across multiple microservices. How would you approach this?
6. You're migrating a monolith to microservices. What strategy would you follow?
7. You need to implement API versioning for your microservices. How would you handle this?
8. You have microservices that need to share common data. How would you implement this?
9. You need to implement a deployment pipeline for your microservices. What would it look like?
10. You need to implement a feature that spans multiple microservices. How would you manage this?

12. Maven and Git

Basic Level - Maven
1. What is Maven and what problems does it solve?
2. What is the purpose of pom.xml?
3. What is a Maven repository?
4. What are the different types of Maven repositories?
5. What is the Maven build lifecycle?
6. What are the Maven phases?
7. What is a Maven artifact?
8. What is dependency management in Maven?
9. What is a Maven plugin?
10. What is the structure of a Maven project?

Intermediate Level - Maven
1. What is the difference between Maven and Gradle?
2. What is the purpose of Maven profiles?
3. How do you handle dependency conflicts in Maven?
4. What is the purpose of Maven dependency scopes?
5. What is the difference between compile, provided, and runtime scopes?
6. What is the purpose of the Maven parent POM?
7. What is the purpose of the Maven BOM (Bill of Materials)?
8. How do you skip tests in Maven?
9. What is the difference between a plugin and a goal in Maven?
10. How does Maven handle transitive dependencies?

Basic Level - Git
1. What is Git?
2. What is a Git repository?
3. What is the difference between Git and GitHub?
4. What is a Git commit?
5. What is a Git branch?

6. What is the difference between pull and fetch?
7. What is a Git merge conflict?
8. What is the purpose of .gitignore?
9. What is the Git staging area?
10. What is a remote in Git?

Intermediate Level - Git
1. What is the Git flow branching model?
2. What is the difference between merge and rebase?
3. What is Git cherry-pick?
4. What is Git stash?
5. What is a Git tag?
6. How do you undo the last commit in Git?
7. What is Git bisect?
8. What is a Git submodule?
9. What is Git blame?
10. What is the HEAD in Git?

Scenario-based Questions - Maven and Git
1. You need to manage different configurations for different environments in your Maven project. How would you approach this?
2. You have a Maven project with a long build time. How would you optimize it?
3. You need to set up a continuous integration pipeline for your Maven project. What steps would you take?
4. You need to collaborate with a team on a Git project. What branching strategy would you recommend?
5. You've made changes but need to switch branches before committing. How would you handle this?
6. You accidentally committed sensitive information to a Git repository. How would you remove it from history?
7. You need to manage dependencies for a large Java project. What approaches would you take using Maven?
8. You need to implement a custom Maven plugin. How would you approach this?
9. You have a long-running feature branch that has diverged significantly from main. How would you integrate it?
10. You need to debug a build failure in a Maven project. What steps would you take?

13. Testing with JUnit and Mockito

Basic Level
1. What is unit testing?
2. What is JUnit?
3. What are the annotations used in JUnit 5?
4. What is the purpose of assertions in JUnit?
5. How do you handle exceptions in JUnit tests?
6. What is Mockito?
7. What is the difference between mock and spy in Mockito?
8. What is the purpose of the @Mock annotation?
9. What is the purpose of the @InjectMocks annotation?

10. How do you verify method calls with Mockito?

Intermediate Level
1. What are the differences between JUnit 4 and JUnit 5?
2. What are parameterized tests in JUnit?
3. What are test lifecycle methods in JUnit?
4. How do you implement test fixtures in JUnit?
5. What is test-driven development (TDD)?
6. How do you stub method calls with Mockito?
7. How do you handle void methods with Mockito?
8. What is the difference between when-then and do-return syntax in Mockito?
9. What is an ArgumentCaptor and when would you use it?
10. How do you mock static methods with Mockito?

Advanced Level
1. How do you implement custom assertions in JUnit?
2. What are JUnit extensions? How do you create a custom extension?
3. What is behavior-driven development (BDD) style in Mockito?
4. How do you handle complex mock interactions in Mockito?
5. How do you test multi-threaded code with JUnit?
6. How do you mock final classes and methods?
7. What is the Spring Test framework? How does it integrate with JUnit?
8. How do you test Spring Boot applications?
9. What is test coverage? How do you measure it?
10. What are the best practices for writing maintainable tests?

Scenario-based Questions
1. You need to test a method that depends on an external service. How would you use Mockito to mock the dependency?
2. You need to test a method that should throw an exception under certain conditions. How would you write this test?
3. You have a method that processes a large dataset. How would you test it efficiently?
4. You need to test a Spring MVC controller. What approach would you take?
5. You need to test a Spring Data JPA

repository. How would you do this?
6. You need to test a method that has complex dependencies. How would you use Mockito's stubbing feature?
7. You need to test a method that has different behaviors based on different inputs. How would you use parameterized tests?
8. You need to test a legacy code with poor testability. What strategies would you use?
9. You need to implement integration tests for a microservice. What approach would you take?
10. You want to improve your team's test coverage. What steps would you take?