✅ Core Java – Complex Interview Questions

🧵 Multithreading & Concurrency

1. **What’s the difference between** synchronized**,** ReentrantLock**, and** StampedLock**?**
2. How does ThreadLocal work internally? When should it be used and avoided?
3. What is the difference between synchronized, ReentrantLock, and StampedLock?
4. How does wait(), notify(), and notifyAll() work internally? Why are they in Object class?
5. What are potential deadlock conditions, and how do you prevent or detect them in production?
6. Explain Java Memory Model (JMM) and happens-before relationships.
7. Difference between volatile and AtomicInteger? When to use what?
8. What is the difference between ForkJoinPool and ExecutorService?
9. What is CompletableFuture and how does it compare to Future and RxJava?

🧠 OOPs & Design Principles

1. Explain the SOLID principles with real Java code examples.
2. How does method resolution work in Java for interface default methods in case of multiple inheritance?
3. What is method hiding vs overriding? Explain with static and private methods.
4. Can abstract class have a constructor? When would you use one?
5. What is the Liskov Substitution Principle, and how does it apply to Java collections?
6. Explain the diamond problem and how Java 8 resolves it with interfaces.

🚀 JVM Internals & Performance

1. What happens in memory when you write new String("abc")?
2. Explain different memory areas in JVM and how GC handles each.
3. What is the difference between stack memory and heap memory in Java?
4. What is escape analysis and how does it affect object allocation?
5. Explain class loading process and class loaders in Java.
6. What are soft, weak, and phantom references? When to use them?

💥 Exception Handling & Serialization

1. Can you override a method and throw a broader exception than the parent?
2. How does exception handling impact performance in Java?
3. What is the role of serialVersionUID?
4. Can a class be Serializable but not Serializable at runtime? Why and how?
5. How do you securely deserialize objects in Java to avoid vulnerabilities (e.g., RCE)?

🔐 Immutability & Memory Management

1. How do you create an immutable class in Java?
2. Why is String immutable in Java? Can you make your own immutable version of StringBuilder?
3. What is the impact of string interning on memory and performance?
4. Explain shallow vs deep cloning and how to implement them properly.
5. What is the role of finalize() and why is it deprecated?

🔄 Collections & Generics

1. What is the internal structure of HashMap in Java 8 and Java 16+?
2. Explain fail-fast vs fail-safe iterators with examples.
3. What are the differences between ConcurrentHashMap, Hashtable, and Collections.synchronizedMap()?
4. What is type erasure in generics? What are the consequences?
5. How would you implement your own generic LRU cache?

👨‍💻 Coding Challenges / Puzzles

1. Implement a thread-safe singleton with lazy loading.
2. Write a function to detect a deadlock in a system.
3. Design a rate limiter using Java concurrency utilities.
4. Implement a custom ThreadPoolExecutor or simplified version.
5. Implement a producer-consumer model using BlockingQueue.

🎯 Bonus – Real-world Scenario Questions

1. How do you troubleshoot memory leaks in a Spring Boot application?
2. How would you debug a java.lang.OutOfMemoryError: GC Overhead limit exceeded?
3. How do you tune a garbage collector for high-throughput vs low-latency systems?
4. How would you architect a Java system that processes 10,000 transactions/sec in real-time?
5. What changes would you make if a HashMap has performance degradation due to key collisions?