**✅ 1. OOPs & Design Principles**

**Q1. How does Java support Object-Oriented principles? Explain with examples.**  
**A:** Java supports **Encapsulation, Inheritance, Polymorphism, and Abstraction**.

* *Encapsulation:* Using private fields + public getters/setters
* *Polymorphism:* Overloading (compile-time), Overriding (runtime)
* *Abstraction:* Via interfaces or abstract classes
* *Inheritance:* class B extends A

**Q2. Explain SOLID principles in the context of Java design.**  
**A:**

* **S**: Single Responsibility – Each class should do one thing.
* **O**: Open/Closed – Classes should be open for extension, closed for modification. Use interfaces.
* **L**: Liskov – Derived classes should be substitutable for base classes.
* **I**: Interface Segregation – Prefer small interfaces over large ones.
* **D**: Dependency Inversion – Depend on abstractions, not concretions.

**✅ 2. Core Java Concepts**

**Q3. Difference between == and equals() in Java?**  
**A:**

* == checks **reference equality**,
* equals() checks **logical equality**.  
  For example:

java

CopyEdit

String a = new String("Hello");

String b = new String("Hello");

a == b; // false

a.equals(b); // true

**Q4. What is the difference between HashMap, LinkedHashMap, and TreeMap?**

| **Map Type** | **Order Maintained** | **Performance** | **Null Keys** |
| --- | --- | --- | --- |
| HashMap | No | Fast | One null key |
| LinkedHashMap | Insertion Order | Slightly slower | One null key |
| TreeMap | Sorted by keys (natural/comparator) | Slower | No null key |

**✅ 3. Exception Handling**

**Q5. How does exception handling work in Java? How do you design custom exceptions?**  
**A:** Java uses try-catch-finally for handling exceptions.

java

CopyEdit

class MyCustomException extends Exception {

public MyCustomException(String msg) {

super(msg);

}

}

* Use **checked exceptions** for recoverable errors,
* **unchecked exceptions** for programming errors (e.g., NullPointerException).

**✅ 4. Java Memory Model & GC**

**Q6. Describe Java memory management and the role of the Garbage Collector.**  
**A:**

* **Heap:** Stores objects
* **Stack:** Stores method call frames, local variables
* **GC:** Automatically deallocates unreachable objects.
* GC types: **Serial, Parallel, CMS, G1GC**
* JVM tuning via -Xmx, -Xms, -XX:+UseG1GC

**Cross Q:** How do you detect and troubleshoot memory leaks in Java?  
Use tools like **VisualVM**, **JProfiler**, or **Heap dumps** + jmap.

**✅ 5. Multithreading & Concurrency**

**Q7. Explain the Java Memory Model (JMM).**  
**A:** JMM defines how threads interact through memory. Guarantees **happens-before relationships**, visibility, and ordering.

**Q8. What are differences between synchronized, volatile, Lock, and Atomic?**

| **Feature** | **synchronized** | **volatile** | **ReentrantLock** | **Atomic** |
| --- | --- | --- | --- | --- |
| Mutual Excl. | Yes | No | Yes | No |
| Visibility | Yes | Yes | Yes | Yes |
| Blocking | Yes | No | Yes | No |

**✅ 6. Java 8+ Features**

**Q9. What are key Java 8 features you’ve used in production?**

* **Streams API** – for functional-style data processing
* **Lambdas & Method References**
* **Optional<T>** – avoid NullPointerException
* **Date/Time API** (java.time.\*)
* **Default and static methods in interfaces**

**Q10. Difference between map() and flatMap() in Java Streams?**

* map() transforms elements individually
* flatMap() flattens nested structures

java

CopyEdit

List<List<String>> -> flatMap -> List<String>

**✅ 7. Serialization & I/O**

**Q11. What are the risks of using Java Serialization?**

* Security vulnerabilities (e.g., RCE)
* Version compatibility issues
* Consider alternatives like JSON or Protobuf

**✅ 8. Best Practices**

**Q12. How do you ensure high-quality code in Java applications?**

* **SOLID principles**, **Code Reviews**, **Unit Tests + Mocks**
* Use **SonarQube** for static analysis
* Apply **design patterns** like Factory, Singleton, Builder

**✅ 9. Real-World Scenario Questions**

**Q13. You notice a CPU spike in a production JVM process. How do you investigate?**

1. Use top, jstack to find high CPU thread
2. Use jmap, jconsole, VisualVM
3. Analyze deadlocks or infinite loops
4. Look for tight loops or excessive GC

**Q14. What is your strategy for handling thread-safety in multi-threaded environments?**

* Minimize shared mutable state
* Use concurrent collections
* Favor **immutable objects**, AtomicXXX, or Lock APIs
* Use ExecutorService for controlled thread management

**✅ PDF / GitHub / Diagram Options**

Would you like:  
✅ A **PDF version** of this Q&A?  
✅ A **GitHub repo** with real coding exercises on each topic?  
✅ Sequence diagrams or memory management illustrations?

**✅ Java 17 & Java 21 Key Features (LTS Versions)**

**🔷 Java 17 (LTS - Sept 2021)**

| **Feature** | **Description** | **Use Case / Syntax** |
| --- | --- | --- |
| ✅ Sealed Classes | Restrict which classes can extend a class | sealed class Vehicle permits Car, Truck |
| ✅ Pattern Matching for instanceof | Simplifies type checks and casting | if (obj instanceof String s) { ... } |
| ✅ Switch Expressions (from Java 14) | More powerful and concise | switch (day) { case MONDAY -> ...; } |
| ✅ Records | Immutable data carriers | record Person(String name, int age) {} |
| ✅ Text Blocks | Multi-line string literals | String sql = """SELECT \* FROM user"""; |
| ✅ JEP 356: Enhanced Pseudo-Random Number Generators | New RNG API | Better statistical properties |
| ✅ JEP 382: New macOS rendering pipeline | For macOS platforms | JavaFX/GUIs on macOS |

**🔷 Java 21 (LTS - Sept 2023)**

| **Feature** | **Description** | **Use Case / Syntax** |
| --- | --- | --- |
| ✅ Virtual Threads (Project Loom) | Lightweight concurrency model | Thread.startVirtualThread(() -> runTask()); |
| ✅ Structured Concurrency | Treat multiple tasks as a single unit | Java API under java.util.concurrent.StructuredTaskScope |
| ✅ Sequenced Collections | Ordered Collection Interface | SequencedCollection, Deque, NavigableMap |
| ✅ Pattern Matching for Switch (Preview) | Advanced type-safe matching | switch (obj) { case String s -> ... } |
| ✅ Record Patterns (Preview) | Deconstruct record objects in patterns | case Person(String name, int age) -> ... |
| ✅ Scoped Values (Incubator) | Thread-local alternatives for virtual threads | Thread-safe immutable values |
| ✅ Foreign Function & Memory API (FFM) | Replace JNI, safe native access | MemorySegment, Linker |

**⚠️ Interview Tip:**

**Virtual Threads** (Java 21) = Replaces traditional thread-per-request. Massive for scalability.

Use Case:

java

CopyEdit

try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {

Future<String> user = scope.fork(() -> fetchUser());

Future<String> order = scope.fork(() -> fetchOrders());

scope.join().throwIfFailed();

return user.result() + order.result();

}

**🚀 JVM Internals & JIT Compilation (Advanced)**

**🔧 JIT Compilation in JVM**

| **Type** | **Description** |
| --- | --- |
| Client Compiler (C1) | Fast, less-optimized, used during startup |
| Server Compiler (C2) | Optimized, used for long-running code |
| Graal JIT Compiler | Replaces C2, more efficient, polyglot (JavaScript, Python, etc.) |

Use -XX:+UnlockExperimentalVMOptions -XX:+UseJVMCICompiler to enable Graal.

**🔬 HotSpot JVM – Optimization Techniques**

| **Optimization** | **Description** |
| --- | --- |
| Inlining | Method calls are replaced with method body |
| Escape Analysis | Determines if objects can be stack-allocated |
| Dead Code Elimination | Unused branches are removed |
| Loop Unrolling | Expands loops for performance |
| Tiered Compilation | Mix of C1 & C2 based on profiling data |

Use:

sh

CopyEdit

-XX:+PrintCompilation

-XX:+UnlockDiagnosticVMOptions -XX:+PrintInlining

**🔧 JVM Tuning Parameters**

**Heap Settings:**

bash

CopyEdit

-Xms2G -Xmx4G # Min/Max Heap size

-Xss512k # Thread stack size

-XX:NewRatio=3 # Ratio between Young:Old gen

**GC Tuning (for G1GC, default in Java 17+):**

bash

CopyEdit

-XX:+UseG1GC

-XX:MaxGCPauseMillis=200

-XX:InitiatingHeapOccupancyPercent=45

**Diagnostic Tools:**

| **Tool** | **Use** |
| --- | --- |
| jmap | Heap dump |
| jstack | Thread dump |
| jstat | JVM statistics |
| jcmd | JVM diagnostics |
| VisualVM / JMC | GUI monitoring & profiling |

**🧠 Common Interview Cross Questions**

**Q: How does Virtual Thread differ from Traditional Thread?**  
**A:** Virtual threads are **lightweight**, managed by the JVM scheduler—not OS. Can spawn millions vs ~1000s with platform threads.

**Q: How do you analyze GC issues?**  
**A:**

1. Enable GC logs: -Xlog:gc\*
2. Use tools: GCViewer, GCEasy
3. Identify Full GCs, memory leaks (via jmap, heap dumps)

**Q: Can you control JIT thresholds?**  
Yes:

bash

CopyEdit

-XX:CompileThreshold=1000

Use JIT logs: -XX:+UnlockDiagnosticVMOptions -XX:+PrintCompilation

**✅ Java 17 & 21 Features – MCQs**

**Q1. Which of the following is a feature introduced in Java 17?**

A. Pattern Matching for switch  
B. Virtual Threads  
C. Sealed Classes  
D. Structured Concurrency  
**✅ Answer: C**  
*(Virtual Threads and Structured Concurrency are Java 21 features)*

**Q2. What does the record keyword do in Java 17?**

A. Defines a mutable class  
B. Defines an immutable data carrier  
C. Acts like a Serializable class  
D. Supports checked exceptions  
**✅ Answer: B**

**Q3. Which feature enables using multi-line string literals in Java 17?**

A. Multiline String  
B. Text Templates  
C. Text Blocks  
D. StringBuilder  
**✅ Answer: C**

**Q4. What is the purpose of Sealed Classes?**

A. To improve serialization  
B. To restrict inheritance  
C. To remove abstract classes  
D. To allow multiple inheritance  
**✅ Answer: B**

**Q5. What is the main benefit of Virtual Threads (Java 21)?**

A. Improved file I/O  
B. Native OS thread integration  
C. Support for Reactive Streams  
D. Lightweight threads for massive scalability  
**✅ Answer: D**

**Q6. What package contains the StructuredTaskScope API introduced in Java 21?**

A. java.util.threads  
B. java.concurrent.virtual  
C. java.util.concurrent  
D. java.task.scope  
**✅ Answer: C**

**Q7. What does pattern matching in switch enable in Java 21?**

A. Runtime overloading  
B. Enum simplification  
C. Type-safe condition branching  
D. Partial class loading  
**✅ Answer: C**

**Q8. Which of the following is not allowed in a Java record?**

A. Final fields  
B. Inheritance from another class  
C. Compact constructor  
D. Implementing an interface  
**✅ Answer: B**

**✅ JIT & JVM Tuning – MCQs**

**Q9. What does JIT stand for in Java?**

A. Java Internal Threads  
B. Just-in-Time Compiler  
C. Java Interactive Toolkit  
D. Java Injection Tuning  
**✅ Answer: B**

**Q10. Which compiler is used for advanced JIT compilation in Java?**

A. C1  
B. C2  
C. Graal  
D. JRE  
**✅ Answer: C**

**Q11. What does escape analysis help with in JIT optimization?**

A. Removing unreachable code  
B. Deciding GC pause times  
C. Allocating objects on the stack  
D. Inlining method calls  
**✅ Answer: C**

**Q12. Which flag enables detailed JIT compilation logs?**

A. -Xlog:jit\*  
B. -XX:+PrintInlining  
C. -XX:+TraceGC  
D. -XX:+EnableC2  
**✅ Answer: B**

**Q13. What is the purpose of Tiered Compilation?**

A. Mix of static and dynamic compilation  
B. Runs C1 and C2 JITs in cooperation  
C. Compiles JVM and JRE together  
D. Allows hot code to run in interpreter mode  
**✅ Answer: B**

**Q14. Which of these is a default Garbage Collector in Java 17+?**

A. CMS  
B. G1GC  
C. SerialGC  
D. Shenandoah  
**✅ Answer: B**

**Q15. How do you limit GC pause times?**

A. -XX:+UseSerialGC  
B. -Xms4G  
C. -XX:MaxGCPauseMillis=200  
D. -XX:+AlwaysPreTouch  
**✅ Answer: C**

**Q16. Which tool helps take thread dumps in real-time?**

A. jmap  
B. jstat  
C. jstack  
D. jlink  
**✅ Answer: C**

**Q17. What is the primary advantage of Graal JIT?**

A. Faster interpreter  
B. Ahead-of-Time only support  
C. Support for polyglot languages & better performance  
D. Smaller class size  
**✅ Answer: C**

**Q18. What does -Xmx4G do?**

A. Set thread stack size  
B. Set max heap size to 4GB  
C. Set metaspace limit  
D. Limits GC logs  
**✅ Answer: B**

**Q19. What does -Xss256k configure in JVM?**

A. Heap size  
B. PermGen size  
C. Metaspace size  
D. Thread Stack Size  
**✅ Answer: D**

**Q20. Which flag is used to enable Graal JIT compiler in Java?**

A. -XX:+UseGraalVM  
B. -XX:+EnableJIT2  
C. -XX:+UseJVMCICompiler  
D. -XX:+JITGraal  
**✅ Answer: C**

**📎 Do you want?**

* ✅ All these questions in **PDF format**
* ✅ Add **20 more scenario-based MCQs**
* ✅ A **mock quiz app (HTML/JS)** for self-practice
* ✅ GitHub repo with code samples and tuning configs

Let me know what you'd like next!

**You said:**

Add 20 more scenario-based MCQs

**ChatGPT said:**

Here are **20 scenario-based MCQ questions** covering **Java 17/21 features and JVM/JIT tuning**, suitable for **12+ years experienced professionals** targeting architect/Sr. dev roles.

**✅ Java 17/21 Feature-Based Scenarios**

**Q1. You want to model an immutable value object Person with two fields: name and age. What's the most concise way in Java 17?**

A. Use a POJO with private fields and getters  
B. Use Lombok @Data  
C. Use record Person(String name, int age)  
D. Use an abstract class with final fields

**✅ Answer: C**

**Q2. You are designing a framework and want to restrict inheritance to specific subclasses only. What should you use?**

A. abstract class  
B. final class  
C. sealed class with permits  
D. interface with default methods

**✅ Answer: C**

**Q3. You are writing a parser and need to switch based on the actual type of tokens (String, Integer, Double). What feature from Java 21 helps?**

A. Sealed Interfaces  
B. Enhanced instanceof  
C. Pattern matching for switch  
D. Method overloading

**✅ Answer: C**

**Q4. You are building a reactive API, but want to retain the thread-per-request model with high scalability. What's your Java 21 solution?**

A. Use ExecutorService  
B. Use Virtual Threads  
C. Use NIO  
D. Use ThreadPoolExecutor with 1000 threads

**✅ Answer: B**

**Q5. You are performing concurrent tasks that must be managed as a single unit (fail/succeed together). What Java 21 API do you use?**

A. CompletableFuture  
B. ExecutorService  
C. StructuredTaskScope  
D. ForkJoinPool

**✅ Answer: C**

**Q6. You are accessing a native C library from Java. What new API from Java 21 replaces JNI?**

A. Unsafe  
B. MemoryMappedFile  
C. Foreign Function & Memory API  
D. JNA

**✅ Answer: C**

**Q7. You want to enforce a read/write order in collections. What should you use from Java 21?**

A. List  
B. Set  
C. SequencedCollection  
D. ConcurrentMap

**✅ Answer: C**

**Q8. What is a correct use case of record pattern matching in Java 21?**

A. Deconstructing JSON  
B. Validating REST input  
C. Extracting data from record in switch  
D. Replacing reflection

**✅ Answer: C**

**✅ JVM/JIT Tuning Scenarios**

**Q9. Your application is experiencing long GC pause times. What is your first tuning step?**

A. Increase heap size  
B. Use -XX:+UseParallelGC  
C. Tune -XX:MaxGCPauseMillis with G1GC  
D. Switch to CMS

**✅ Answer: C**

**Q10. Your application creates many short-lived threads, causing performance issues. How do you resolve it in Java 21?**

A. Use ThreadPoolExecutor  
B. Use Virtual Threads  
C. Use ForkJoinPool  
D. Increase -Xss

**✅ Answer: B**

**Q11. You observe that frequently invoked methods are not being optimized. What JIT flag helps debug this?**

A. -XX:+PrintGCDetails  
B. -XX:+PrintInlining  
C. -XX:+TraceClassLoading  
D. -Xmx1G

**✅ Answer: B**

**Q12. You're debugging a memory leak. What tool provides a heap dump?**

A. jstack  
B. jmap  
C. jcmd GC.class\_histogram  
D. jstat

**✅ Answer: B**

**Q13. You notice too many Full GCs. Which option most likely helps reduce them in Java 17?**

A. Decrease -Xmx  
B. Use SerialGC  
C. Use G1GC and tune region size  
D. Increase thread count

**✅ Answer: C**

**Q14. You're profiling a long-running production app and want low overhead live monitoring. What should you use?**

A. jstack  
B. VisualVM  
C. Java Mission Control (JMC)  
D. NetBeans Debugger

**✅ Answer: C**

**Q15. You're seeing high CPU usage from one thread. How do you identify the method causing this?**

A. Take a heap dump  
B. Use top + jstack with thread ID  
C. Use jmap -hist  
D. Use -Xss to increase stack

**✅ Answer: B**

**Q16. A method is being called 10,000 times/sec. How can JIT optimize it faster?**

A. Use -XX:CompileThreshold=100  
B. Disable GC  
C. Use G1GC  
D. Use final class

**✅ Answer: A**

**Q17. You're seeing frequent Safepoints. What could be the cause?**

A. High thread contention  
B. Code cache is full  
C. GC events  
D. Large stack frames

**✅ Answer: C**

**Q18. You need fine-grained control over memory access for high-performance native ops. What Java 21 API helps?**

A. ByteBuffer  
B. Unsafe  
C. MemorySegment from FFM  
D. MappedByteBuffer

**✅ Answer: C**

**Q19. You want to eliminate method call overhead in performance-critical code. What JIT optimization helps?**

A. Devirtualization  
B. Loop Unrolling  
C. Inlining  
D. Code caching

**✅ Answer: C**

**Q20. You want to use a compiler that supports multiple JVM languages and advanced JIT. Which do you use?**

A. C2  
B. C1  
C. GraalVM  
D. CMS

**✅ Answer: C**