Created by: Lamhot Siagian

111 Java Interview Prep Questions — Angel Cheatsheet

🔰 Core Java Fundamentals (10)

Question	Hint answer
Primitive vs reference types?	8 primitives (byte…double, boolean, char) stored by value vs. objects/arrays by reference.
var vs explicit types (Java 10)?	Local type inference for readability; still statically typed; avoid overuse in public APIs.
== vs .equals()?	== compares references (primitives by value); equals compares value semantics if overridden.
String, StringBuilder, StringBuffer?	String immutable; Builder mutable & non-thread-safe; Buffer mutable & synchronized.
switch expressions (Java 14+)?	Return a value, use arrow labels, yield; less boilerplate than classic switch.
pass-by-value clarification?	Java passes copies of references for objects and copies of primitive values.
Control statements basics?	if/else, switch, loops (for/while/do), break/continue, try-with-resources.
Integer caching?	-128..127 boxed cache; == may appear equal in range; don't rely on it.
Enums advantages?	Type safety, singleton per constant, can hold fields/behaviors, switch-friendly.
Records (Java 16+)?	Compact immutable data carriers with auto equals/hashCode/toString.

🫘 OOP Principles (10)

Question	Hint answer
Encapsulation?	Hide state via private fields; expose behavior via methods; invariants protected.
Abstraction vs interface?	Abstract class = partial implementation; interface = contract; multiple interfaces allowed.
Inheritance vs composition?	"Is-a" vs "has-a"; prefer composition for flexibility and testability.
Polymorphism example?	Overriding methods resolved at runtime (dynamic dispatch).
Overloading vs overriding?	Overload = same name, diff params/arity; override = same signature, subclass behavior.
final, finally, finalize()?	final keyword, finally block after try, legacy finalize (deprecated).
LSP/SOLID quick recap?	Liskov substitution; SOLID: SRP, OCP, LSP, ISP, DIP.
Sealed classes (Java 17)?	Restrict allowed subclasses; control type hierarchies.
Equals/HashCode contract?	Consistent, reflexive, symmetric, transitive; equal objects share hash.
Designing immutable classes?	Private final fields, no setters, defensive copies, builders for construction.

🧺 Collections Framework (10)

Question	Hint answer
List vs Set vs Map?	List ordered & dupes; Set unique; Map key→value associations.
ArrayList vs LinkedList?	ArrayList random access fast; LinkedList fast insert/delete at ends.
HashMap vs LinkedHashMap vs TreeMap?	HashMap unordered; Linked preserves insertion; TreeMap sorted/Navigable.
ConcurrentHashMap traits?	Segmented/bucket concurrency, no nulls, safe concurrent ops.
Fail-fast vs fail-safe iterators?	Fail-fast detect mod (ConcurrentModificationException); fail-safe iterate over snapshot.
Comparator vs Comparable?	Comparable defines natural order; Comparator external custom order.
Immutable collections?	List.of/Set.of/Map.of (Java 9+); Collections.unmodifiable* wrappers.
Stream vs Collection?	Stream is pipeline for processing; Collection stores data.
Queue vs Deque?	FIFO vs double-ended operations (stack/queue behaviors).
Big-O hints?	Hash O(1) avg; Tree O(log n); array random access O(1), insert mid O(n).

🧯 Exception Handling (9)

Question	Hint answer
Checked vs unchecked?	Checked extends Exception; unchecked extends RuntimeException; API design tradeoffs.
try-with-resources?	Auto-close AutoCloseable; suppresses exceptions recorded.
Custom exceptions design?	Meaningful names, context fields, preserve cause, avoid overuse.
Best practices in catch blocks?	Log with context, avoid swallow, rethrow as needed, map to error codes.
finally vs try-with-resources?	finally for generic cleanup; TWR safer for resources.
Exception translation?	Wrap low-level exceptions into domain-specific ones.
Common pitfalls?	Catch Exception broadly, empty catch, using exceptions for flow control.
Stack trace management?	Include root cause; avoid log spam; sanitize in prod logs.
Assertions vs exceptions?	assert for dev invariants; exceptions for runtime error handling.

🗂️ Java I/O & NIO (9)

Question	Hint answer
Stream vs Reader/Writer?	Stream for bytes; Reader/Writer for chars (with encodings).
Buffered I/O benefits?	Reduces system calls; improves throughput for small reads/writes.
Path, Files, and NIO.2?	java.nio.file APIs for modern file ops, walk, watch service.
Channels & ByteBuffer?	Non-blocking I/O; direct vs heap buffers; scatter/gather.
Memory-mapped files?	Map file regions to memory for fast random access; be cautious with size.
Serialization options?	Java native (legacy), JSON (Jackson/Gson), binary (Kryo/ProtoBuf).
Character encodings?	UTF-8 default (Java 18); always specify when interoperating.
WatchService use case?	Directory changes monitoring for sync/hot-reload pipelines.
Try-with-resources in I/O?	Auto close streams/readers/writers to avoid leaks.

⚙️ Multithreading & Concurrency (10)

Question	Hint answer
Thread vs Runnable vs Callable?	Thread = execution unit; Runnable no result; Callable returns & can throw.
synchronized vs ReentrantLock?	Intrinsic vs explicit lock; tryLock, fairness, conditions in ReentrantLock.
volatile keyword?	Visibility & ordering guarantees; not atomicity for compound ops.
Atomic classes?	AtomicInteger, AtomicReference for lock-free atomic updates.
Executors framework?	Thread pools, submit tasks, schedule; use newFixedThreadPool/commonPool.
ForkJoinPool & parallel streams?	Work-stealing; use for CPU-bound tasks; beware blocking ops.
Concurrent collections?	BlockingQueue, ConcurrentHashMap, CopyOnWriteArrayList.
Deadlock & avoidance?	Lock ordering, timeouts, min critical sections, detect via thread dumps.
CompletableFuture basics?	thenApply/thenCompose, allOf/anyOf, async combinators, timeouts.
Structured concurrency (Project Loom)?	Virtual threads simplify concurrency; structured scoping of tasks.

🌿 Java 8+ Features (10)

Question	Hint answer
Lambda syntax & targets?	(args) → body; targets functional interfaces (SAM types).
Method references?	Class::static, instance::method, Class::new (constructor ref).
Stream pipeline phases?	Source → intermediate ops (map/filter) → terminal (collect/forEach).
Stateless vs stateful ops?	Stateless map/filter; stateful sorted/distinct/limit.
Collectors you use?	toList, groupingBy, mapping, joining, collectingAndThen.
Optional best practices?	Avoid fields/params; use for return types; orElseGet vs orElse.
Default & static methods in interfaces?	Provide behavior & utilities; resolve diamond via explicit override.
LocalDate/Time API perks?	Immutable, time-zones, Duration/Period, formatting/parsing.
Records & sealed types synergy?	Model ADTs with sealed interfaces + record implementations.
Pattern matching (instanceof, switch)?	Simplify type casts and destructuring in newer Java.

🧠 Memory & GC (9)

Question	Hint answer
Heap vs stack?	Heap for objects; stack for frames/locals; stack per thread.
GC algorithms?	Mark-sweep, generational, G1, ZGC, Shenandoah—tradeoffs latency/throughput.
Young vs old generation?	Eden/survivor spaces vs tenured; promotion thresholds.
Stop-the-world events?	Pauses for safepoints/GC; tune via GC choice & heap sizing.
Soft/weak/phantom references?	Soft for caches; weak for maps; phantom for cleanup notifications.
Memory leaks in Java?	Unbounded caches, listeners not removed, static references, ThreadLocal misuse.
JVM tuning basics?	-Xms/-Xmx, GC flags, profiling with JFR/VisualVM, heap dumps.
Escape analysis?	JIT allocs on stack, scalar replacement; reduces GC pressure.
OutOfMemoryError handling?	Capture heap dump, analyze dominators, fix root cause not just heap size.

🛠️ JVM Internals (9)

Question	Hint answer
Class loading phases?	Loading → linking (verify/prepare/resolve) → initialization.
Bootstrap vs app class loader?	Bootstrap loads core; platform/app loaders for libs/apps; delegation model.
JIT vs AOT?	JIT compiles hot methods at runtime; AOT precompiles; tiered compilation.
Bytecode & JVM stack model?	Operand stack architecture; frames, local vars, constant pool.
Reflection pros/cons?	Dynamic access vs performance/security risks; limit in prod paths.
Modules (JPMS)?	module-info.java boundaries; exports/requires; strong encapsulation.
Invokedynamic use?	Dynamic languages, lambdas implementation; call site linking.
Class redefinition/instrumentation?	javaagent, ASM/ByteBuddy for runtime weaving/profiling.
JFR & diagnostics?	Low-overhead profiling/events; flight recordings for prod issues.

🧩 Design Patterns in Java (10)

Question	Hint answer
Singleton pitfalls & fixes?	Lazy vs eager; double-checked locking; enum singleton safest.
Factory vs Abstract Factory?	Factory creates one product; Abstract Factory families of related products.
Builder benefits?	Readable immutable construction; telescoping constructors avoided.
Strategy vs State?	Strategy swaps algorithms; State changes behavior by internal state.
Observer use case?	Event subscription model; decoupled publishers/subscribers.
Decorator vs Proxy?	Decorator adds behavior; Proxy controls access/caching/lazy load.
Adapter vs Facade?	Adapter converts interface; Facade simplifies a complex subsystem.
Template Method vs Hook?	Skeleton algorithm in base; subclasses fill steps; hooks optional.
Composite pattern?	Treat part-whole trees uniformly; leaf vs composite nodes.
Command pattern?	Encapsulate requests as objects; undo/redo, queues, logging.

🫙 Java Frameworks & Libraries (8)

Question	Hint answer
Spring Boot auto-configuration?	Conditionals on classpath/props; starters for rapid setup.
DI/IoC in Spring?	Beans, @Configuration, @Component, @Autowired/@Inject, scopes.
Spring MVC vs WebFlux?	Servlet blocking vs reactive non-blocking; choose per workload.
Hibernate/JPA basics?	Entity mapping, persistence context, lazy vs eager, JPQL/Criteria.
Transactions?	@Transactional, propagation, isolation levels, rollback rules.
Testing with JUnit & Mockito?	JUnit 5 Jupiter; @ExtendWith; mocks/stubs; parameterized tests.
Logging options?	SLF4J API + Logback/Log4j2; structured JSON logs for services.
Build tools & dependency mgmt?	Maven/Gradle basics; BOMs; version alignment; reproducible builds.

🤖 Java for Automation & APIs (7)

Question	Hint answer
REST Assured basics?	given-when-then; request spec, response extraction, JSON schema checks.
HTTP client choices?	java.net.http (Java 11), Apache HttpClient, OkHttp; async vs blocking.
JSON/XML parsing?	Jackson/Gson, JAXB; POJOs with annotations; streaming vs tree model.
Selenium WebDriver integration?	Page objects, waits, headless CI, grid/cloud providers.
CI integration for tests?	JUnit XML/Allure reports, artifacts (traces/screenshots), environment variables.
Contract & schema tests?	OpenAPI/Swagger validation, consumer-driven contracts (Pact).
Security considerations?	Secrets, OAuth2 flows, CSRF, input validation, rate limiting tests.