**✅ 1. What is JDK, JRE, and JVM?**

**Answer:**

| **Component** | **Description** |
| --- | --- |
| **JDK (Java Development Kit)** | It's a software development kit used to develop Java applications. It includes JRE + development tools (javac, javadoc, etc.). |
| **JRE (Java Runtime Environment)** | It's a runtime environment that includes JVM + libraries for running Java applications. |
| **JVM (Java Virtual Machine)** | It's a virtual engine that runs the bytecode and provides abstraction from underlying OS/hardware. |

**Follow-up:**  
👉 *Is JVM platform-dependent?*  
**Answer:** JVM is platform-dependent because its implementation differs across OS. But Java is platform-independent due to the JVM.

**✅ 2. Is JVM part of JRE or JDK?**

**Answer:** JVM is part of JRE.  
JRE is part of JDK.  
Thus, JDK ⊃ JRE ⊃ JVM.

**✅ 3. Can you run Java code without JDK?**

**Answer:** Yes, if the code is already compiled into bytecode (.class), you can run it using **JRE**, which contains the JVM.  
To compile Java source code (.java), **JDK** is required.

**✅ 4. What is the role of JVM in Java?**

**Answer:**

* Loads .class files (classloader)
* Verifies bytecode
* Executes bytecode using Just-In-Time (JIT) compiler
* Manages memory (via Garbage Collector)
* Handles runtime exceptions and multithreading

**✅ 5. What are the main components of JVM architecture?**

**Answer:**

1. **ClassLoader**
2. **Method Area**
3. **Heap**
4. **Stack**
5. **Program Counter Register**
6. **Native Method Stack**
7. **Execution Engine**
8. **Garbage Collector**

**Follow-up:**  
👉 *Which area is used for storing class metadata?*  
**Answer:** Method Area.

**✅ 6. What is the difference between JVM and a compiler?**

**Answer:**

* **Compiler (javac):** Translates Java code into bytecode (.class)
* **JVM:** Executes the bytecode.

**✅ 7. What is JIT compiler in JVM?**

**Answer:**  
JIT (Just-In-Time) compiler improves performance by compiling bytecode into native machine code **at runtime**, reducing interpretation overhead.

**Follow-up:**  
👉 *Where is JIT used in the JVM?*  
**Answer:** Inside the **Execution Engine**.

**✅ 8. Explain JVM memory areas.**

**Answer:**

* **Heap:** Objects and class instances
* **Stack:** Method-local variables and call frames
* **Method Area:** Class metadata and static variables
* **PC Register:** Keeps track of instruction being executed
* **Native Method Stack:** Native (non-Java) methods

**✅ 9. What is the use of ‘java’ and ‘javac’ commands?**

**Answer:**

* javac: Java Compiler – compiles .java → .class
* java: Java Application Launcher – runs .class using JVM

**✅ 10. Can JVM execute .java files directly?**

**Answer:**  
No. JVM only executes **bytecode** (.class files).  
.java files must be compiled using javac first.

**✅ 11. How does Java ensure platform independence?**

**Answer:**  
Java code is compiled to **bytecode**, which is interpreted or compiled by **JVM** specific to each OS.

**✅ 12. What is the difference between JDK 8 and JDK 11+?**

**Answer:**

* JDK 8: LTS version, Oracle-supported, includes javac, tools.jar
* JDK 11+: More modular (JEP-220), tools.jar removed, java command supports running .java files directly

**✅ 13. How is memory managed in JVM?**

**Answer:**  
Through **Automatic Garbage Collection**, performed in the Heap. JVM frees memory used by unreachable objects.

**Follow-up:**  
👉 *Which GC algorithms do you know?*  
**Answer:** Serial GC, Parallel GC, CMS (deprecated), G1 GC, ZGC (from Java 11+), Shenandoah.

**✅ 14. Can multiple JVMs run on a single machine?**

**Answer:**  
Yes, multiple JVM instances can run simultaneously on the same OS as separate processes.

**✅ 15. What happens when you type java HelloWorld in terminal?**

**Answer:**

1. JVM is launched.
2. ClassLoader loads HelloWorld.class
3. Bytecode is verified.
4. main() method is executed inside the JVM Execution Engine.

**✅ 16. What is the difference between HotSpot JVM and OpenJ9?**

**Answer:**

* **HotSpot:** Oracle’s default JVM, widely used, optimized with JIT and adaptive optimizations.
* **OpenJ9:** IBM’s open-source JVM with low memory footprint and fast startup (used in microservices).

**✅ 17. Can Java code be run without installing JDK or JRE?**

**Answer:**  
Yes, using a **packaged native image** via tools like **GraalVM Native Image** or using **jlink** in modular Java to create custom runtime.

**✅ 18. Can you list Java commands and their usage?**

| **Command** | **Usage** |
| --- | --- |
| javac | Compile .java files |
| java | Run .class files |
| jps | JVM Process Status Tool |
| jstack | Prints thread stack trace |
| jmap | Memory usage info |
| jconsole, jvisualvm | Monitoring JVM |

**✅ 19. What is the role of JAVA\_HOME?**

**Answer:**  
JAVA\_HOME is an environment variable that points to the root directory of the installed JDK. It's required by many tools (Maven, Gradle, IDEs) to compile/run Java.

**✅ 20. How do you debug JVM issues in production?**

**Answer:**

* Use jstack, jmap, jconsole, jvisualvm, or Java Flight Recorder
* Analyze GC logs
* Enable verbose GC: -verbose:gc
* Enable heap dumps on OOM: -XX:+HeapDumpOnOutOfMemoryError

Would you like this as a **PDF**, or should I add **20 more advanced questions** (JVM tuning, GC tuning, native methods, class loading, etc.) for senior-level interviews?

**You said:**

for 12+ experience

**ChatGPT said:**

Updated saved memory

Here are **advanced JVM, JRE, and JDK interview questions and answers** specifically curated for **12+ years of experience**. These are the kinds of questions you'd face in **architect-level, senior developer, or Java performance tuning interviews**.

**🔥 1. Explain the complete JVM architecture with internal components.**

**Answer:**  
JVM components:

* **ClassLoader Subsystem** (Bootstrap, Extension, Application)
* **Runtime Data Areas**:
  + **Method Area**: class-level data (static vars, method code)
  + **Heap**: object instances (young, old, metaspace in Java 8+)
  + **JVM Stack**: each thread has its own stack frame
  + **Program Counter (PC)**: per-thread, tracks bytecode execution
  + **Native Method Stack**
* **Execution Engine**:
  + **Interpreter**
  + **JIT Compiler**
  + **Garbage Collector (GC)**
* **Native Interface (JNI)**
* **Native Method Libraries**

**Cross-question:**  
👉 What changes happened in Metaspace vs PermGen?  
**Answer:** PermGen was removed in Java 8. Metaspace grows dynamically and is OS-dependent.

**🔥 2. What are common JVM tuning parameters used in production?**

**Answer:**

bash

CopyEdit

-Xms1G # Initial heap size

-Xmx4G # Max heap size

-Xss512k # Thread stack size

-XX:MetaspaceSize=128M # Initial metaspace size

-XX:+UseG1GC # Use G1 Garbage Collector

-XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=/path/to/dump.hprof

-XX:+PrintGCDetails -Xloggc:gc.log

**🔥 3. How does JVM manage memory across threads and GC?**

**Answer:**

* Each thread has its own **JVM stack** and **PC Register**.
* **Heap** is shared.
* GC pauses impact all threads during **stop-the-world (STW)** events.
* G1/ZGC/Shenandoah try to reduce pause times by region-based GC or concurrent processing.

**🔥 4. Explain the lifecycle of a Java class in the JVM.**

**Answer:**

1. **Loading** – ClassLoader reads .class file.
2. **Linking**
   * Verification
   * Preparation (static vars default values)
   * Resolution (symbolic references → actual)
3. **Initialization** – Static initializers and static blocks.
4. **Using**
5. **Unloading** – Done by GC only if class is no longer referenced.

**Cross-question:**  
👉 When is a class unloaded?  
**Answer:** Only when the classloader that loaded it is garbage collected.

**🔥 5. What are different types of classloaders?**

**Answer:**

* **Bootstrap** – loads core Java classes (rt.jar)
* **Extension (Platform)** – loads from lib/ext or Java modules
* **Application** – loads from classpath
* **Custom/User-defined** – for special isolation or instrumentation (e.g., Tomcat, OSGi)

**Follow-up:**  
👉 Can two classes with the same name exist in JVM?  
**Answer:** Yes, if loaded by different classloaders.

**🔥 6. How do you troubleshoot memory leaks in Java?**

**Answer:**

* Analyze heap dumps with **MAT** or **VisualVM**
* Enable GC logging
* Use **reference leak detection** tools like **YourKit**, **jProfiler**
* Watch for **static references**, unclosed resources, listeners, caches

**🔥 7. What is the difference between G1, CMS, ZGC, and Shenandoah?**

| **GC Algorithm** | **Strength** | **Weakness** |
| --- | --- | --- |
| **G1** | Low pause time, parallelism, region-based | May still cause STW |
| **CMS** | Concurrent, low pause (deprecated) | Fragmentation |
| **ZGC** | Sub-millisecond pause times, scales to TBs | Java 11+ only |
| **Shenandoah** | Pause-time independent of heap size | Available in OpenJDK |

**🔥 8. What is bytecode verification in JVM?**

**Answer:**  
Before execution, the JVM verifies that bytecode:

* Has valid structure
* Follows access rules
* Doesn’t overflow the operand stack  
  This prevents **malicious code** execution.

**🔥 9. How does the JIT compiler optimize Java code?**

**Answer:**

* Inlines hot methods
* Eliminates dead code
* Performs **loop unrolling**
* Uses **escape analysis** to allocate objects on stack instead of heap

**Cross-question:**  
👉 What is escape analysis?  
**Answer:** Determines if an object can be **safely allocated on the stack** instead of the heap.

**🔥 10. What is Code Cache in JVM?**

**Answer:**

* Memory area where JIT-compiled native code is stored.
* Controlled by ReservedCodeCacheSize
* Too small code cache can cause CodeCacheFull warnings and degraded performance.

**🔥 11. How do you monitor and diagnose JVM at runtime?**

**Answer:**

* **JMX**, **jconsole**, **VisualVM**
* jstat, jstack, jmap, jcmd
* Application Performance Monitoring tools: **New Relic**, **Dynatrace**, **AppDynamics**

**🔥 12. Explain metaspace tuning in Java 8+.**

**Answer:**  
Metaspace holds class metadata (replaced PermGen).  
Tuned using:

* -XX:MetaspaceSize
* -XX:MaxMetaspaceSize

**🔥 13. What happens during a Stop-the-World (STW) event?**

**Answer:**  
All application threads are paused so JVM can safely:

* Perform GC
* Resize heap or compact memory
* Collect diagnostic data

**Tuning tip:** Use concurrent collectors to minimize STW.

**🔥 14. How does JVM handle thread scheduling?**

**Answer:**  
Delegated to the **OS thread scheduler**. Java threads map to **native threads** (1:1). JVM only ensures **synchronization** and **lock semantics**, not strict thread priority enforcement.

**🔥 15. Can JVM memory be fragmented?**

**Answer:**  
Yes. Especially in **CMS**, where heap fragmentation can cause OOM even if memory is available. G1 and ZGC reduce fragmentation via **compaction**.

**🔥 16. What is the difference between interpreter and JIT in JVM?**

**Answer:**

* **Interpreter** executes bytecode line-by-line (slow startup).
* **JIT** compiles hot bytecode paths to native code (faster execution after warm-up).

**🔥 17. What is the default GC in Java 8 and Java 11+?**

| **Version** | **Default GC** |
| --- | --- |
| Java 8 | Parallel GC |
| Java 11 | G1 GC |
| Java 17+ | G1 (or ZGC as opt-in) |

**🔥 18. What is jlink and how does it relate to JDK?**

**Answer:**  
jlink allows creating a **custom runtime image** from JDK modules. Reduces JDK footprint, useful for containers and microservices.

**🔥 19. What is GraalVM and how is it different from JVM?**

**Answer:**  
GraalVM is a high-performance runtime supporting **Java, JS, Python**, etc.

* Supports **AOT (native-image)** compilation.
* Faster startup, low memory footprint
* Replaces HotSpot’s JIT

**🔥 20. Can you create your own JVM?**

**Answer:**  
Yes, by implementing the **Java Virtual Machine Specification**. Examples: HotSpot, OpenJ9, GraalVM. You must support class loading, bytecode execution, GC, and threading.