**SET - 1**

**1. What are the main features of Java?**

**Answer:**  
Java's main features include:

* **Platform Independence**: Java programs are compiled to bytecode that runs on any JVM, making them platform-independent ("Write Once, Run Anywhere").
* **Object-Oriented**: Java follows OOP principles like encapsulation, inheritance, and polymorphism.
* **Robust**: Strong memory management, exception handling, and type checking make Java robust.
* **Secure**: As we can directly share an application with the user without sharing the actual program makes Java a secure language. .
* **Multithreading**: Built-in support for multithreaded programming.
* **Automatic Memory Management**: Garbage collection automatically reclaims unused memory.

**Follow-up Answers:**

**Why is Java platform-independent?**  
Compiler converts source code to byte code and then the JVM executes the bytecode generated by the

compiler. This byte code can run on any platform be it Windows,Linux, or macOS which means if we compile a program on Windows, then we can run it on Linux and vice versa. Each operating system has a different JVM, but the output produced by all the OS is the same after the execution of the byte code. That is why we call java a platform-independent language. Give a real-time example like Minecraft or candy crush saga

### 1. Is Java Platform Independent if then how?

Yes, Java is a Platform Independent language. Unlike many programming languages javac compiles the program to form a bytecode or .class file. This file is independent of the software or hardware running but needs a JVM(Java Virtual Machine) file preinstalled in the operating system for further execution of the bytecode.

Although **JVM is platform dependent**, the bytecode can be created on any System and can be executed in any other system despite hardware or software being used which makes Java platform independent.

**How Java achieves memory management and what makes it robust?**   
Java uses automatic garbage collection to manage memory. The JVM tracks object references and automatically reclaims memory from objects no longer referenced. This prevents memory leaks and makes Java robust compared to languages requiring manual memory management.

Garbage collection in Java is an automatic memory management process that reclaims memory occupied by objects that are no longer in use.

**Real-life example:** Android apps are primarily written in Java, allowing them to run on diverse hardware while maintaining security and performance.

**2. What are the different class loaders in Java?**

**Answer:**

Classloader is the part of JRE(Java Runtime Environment), during the execution of the bytecode or created .class file classloader is responsible for dynamically loading the java classes and interfaces to JVM(Java Virtual Machine). Because of classloaders Java run time system does not need to know about files and file systems.

Java has three main class loaders:

1. **Bootstrap ClassLoader**: Loads core Java classes (rt.jar and other core libraries)
2. **Extension ClassLoader**: Loads classes from the extension directories (jre/lib/ext)
3. **Application/System ClassLoader**: Loads classes from the application classpath

**Real-life example:** When running a web application in Tomcat, Tomcat uses its own class loaders (that extend Java's class loaders) to load web application classes separately from other applications.

**3. What is bytecode in Java, and how is it generated?**

**Answer:**  
Bytecode is the intermediate representation of Java code that is executed by the JVM. It's generated when the Java compiler (javac) compiles .java source files into .class files.

**Follow-up Answers:**

**How does the JVM interpret bytecode?**  
The JVM can interpret bytecode directly or use Just-In-Time (JIT) compilation to convert frequently executed bytecode into native machine code for better performance.

**JVM**: JVM also known as Java Virtual Machine is a part of JRE. JVM is a type of interpreter responsible for converting bytecode into machine-readable code. JVM itself is platform dependent but it interprets the bytecode which is the platform-independent reason why Java is platform-independent.

**JRE**: JRE stands for Java Runtime Environment, it is an installation package that provides an environment to run the Java program or application on any machine.

**JDK**: JDK stands for Java Development Kit which provides the environment to develop and execute Java programs. JDK is a package that includes two things Development Tools to provide an environment to develop your Java programs and, JRE to execute Java programs or applications.

**JDK is an acronym for Java Development Kit. It is a software development environment which is used to develop Java applications and applets. It physically exists. It includes the JRE, development tools (compiler, debugger), and libraries needed for Java development. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:**

* **Standard Edition Java Platform**
* **Enterprise Edition Java Platform**
* **Micro Edition Java Platform**

**4. What is the Just-In-Time (JIT) compiler, and how does it optimize performance?**

**Answer:**

JIT stands for (Just-in-Time) compiler is a part of JRE(Java Runtime Environment), it is used for better performance of the Java applications during run-time. The use of JIT is mentioned in step by step process mentioned below:

1. Source code is compiled with **javac** to form bytecode
2. Bytecode is further passed on to JVM
3. JIT is a part of JVM, JIT is responsible for compiling bytecode into native machine code at run time.
4. The JIT compiler is enabled throughout, while it gets activated when a method is invoked. For a compiled method, the JVM directly calls the compiled code, instead of interpreting it.
5. As JVM calls the compiled code that increases the performance and speed of the execution

The JIT compiler is part of the JVM that compiles bytecode into native machine code at runtime. It improves performance by:

**Follow-up Answers:**

**How does JIT compilation differ from interpretation?**  
Interpretation executes bytecode line by line, while JIT compiles entire methods to native code once and then executes the native code, which is faster.

**5. How does Java handle memory management for loaded classes?**

**Answer:**  
Java stores loaded class metadata in the Method Area (part of JVM memory). When a class is no longer referenced:

1. All instances are garbage collected
2. The Class object becomes unreachable
3. The class metadata may be unloaded (though this is JVM implementation dependent)

**Follow-up Answers:**

**What happens to classes that are no longer referenced?**  
They become eligible for unloading, but this depends on the JVM implementation. Most JVMs are conservative about unloading classes.

**6. What is the difference between stack memory and heap memory in Java?**

**Answer:**

* **Stack Memory**: Used for method execution, stores local variables and method calls. Each thread has its own stack. Faster access but limited in size.
* **Heap Memory**: Used for dynamic memory allocation (objects). Shared among all threads. Larger in size but slower access.

**Follow-up Answers:**

**Which data types are stored in stack and heap memory?**  
Primitive local variables and object references are stored on the stack. Objects themselves and their instance variables are stored on the heap.

**Real-life example:** In a web application, each request thread has its own stack for method execution, while objects like user sessions are stored in the heap shared by all threads.

**7. How does the stack memory work during method calls?**

**Answer:**  
For each method call:

1. A stack frame is created containing local variables, parameters, and return address
2. The frame is pushed onto the stack
3. When the method completes, the frame is popped
4. Control returns to the calling method

**Follow-up Answers:**

**What happens to local variables when a method exits?**  
Their stack frame is popped, and the memory is immediately reclaimed. For object references, the reference is lost but the object itself remains in heap until garbage collected.

**8. What are the implications of memory allocation in the heap for object creation?**

**Answer:**  
Heap allocation implications:

* Objects can be shared across threads
* Allocation is slower than stack allocation
* Memory must be managed (via garbage collection)
* OutOfMemoryError can occur if heap is exhausted

**Follow-up Answers:**

**How does garbage collection affect heap memory?**  
GC automatically reclaims memory from unreachable objects, preventing memory leaks but potentially causing pauses during collection.

**9. How does the JVM manage memory for objects in the heap?**

**Answer:**  
The JVM divides heap into generations:

1. **Young Generation**: New objects are allocated here. Minor GC collects dead objects here.
2. **Old Generation**: Long-lived objects are promoted here. Major GC collects here.
3. **Permanent Generation** (pre-Java 8)/Metaspace (Java 8+): Stores class metadata.

**Follow-up Answers:**

**What is the role of the garbage collector in this process?**  
The GC identifies unreachable objects, reclaims their memory, and may compact memory to reduce fragmentation.

**10. What are stack overflow and heap overflow errors?**

**Answer:**

* **StackOverflowError**: Occurs when the call stack exceeds its limit (usually due to deep recursion)
* **OutOfMemoryError (Heap)**: Occurs when the heap is full and cannot allocate more objects

**Follow-up Answers:**

**How can you prevent these errors in your applications?**  
For stack overflow: Limit recursion depth, use iteration instead of deep recursion.  
For heap overflow: Increase heap size (-Xmx), optimize memory usage, fix memory leaks.

**11. Explain the concept of OOP in Java.**

**Answer:**  
OOP in Java is based on four main principles:

1. **Encapsulation**: Bundling data with methods that operate on that data
2. **Inheritance**: Creating new classes from existing ones
3. **Polymorphism**: Ability of objects to take many forms
4. **Abstraction**: Hiding complex implementation details

**Follow-up Answers:**

**How does Java implement polymorphism?**  
Through method overriding (runtime polymorphism) and method overloading (compile-time polymorphism).

**Difference between method overloading and method overriding?**  
Overloading: Same method name, different parameters (compile-time)  
Overriding: Subclass provides specific implementation of superclass method (runtime)

**12. What is the difference between = and equals() in Java?**

**Answer:**

* = compares object references (memory addresses)
* equals() compares object contents (by default same as ==, but can be overridden)

**Follow-up Answers:**

**Scenario where using = can cause a logical error:**

java

Copy

String s1 = new String("hello");

String s2 = new String("hello");

if (s1 == s2) { ... } // False, different objects

if (s1.equals(s2)) { ... } // True, same content

**How equals() works with user-defined objects:**  
By default compares references, should be overridden to compare meaningful fields.

**13. What is the significance of the final keyword in Java?**

**Answer:**  
final can be applied to:

* Variables: Value cannot be changed (constant)
* Methods: Cannot be overridden
* Classes: Cannot be inherited

**Follow-up Answers:**

**Difference between final variable, method, and class:**

* Variable: Value set once and cannot change
* Method: Subclasses cannot override
* Class: Cannot be subclassed

**Can a final class be inherited?**  
No, final classes cannot be inherited. Useful for security (e.g., String class) or when class implementation should not be modified.

**14. Explain the difference between ArrayList and LinkedList.**

**Answer:**

* **ArrayList**: Backed by dynamic array. Fast random access (O(1)), slower insertions/deletions (O(n) except at end)
* **LinkedList**: Doubly-linked list. Slower random access (O(n)), faster insertions/deletions (O(1) if position known)

**Follow-up Answers:**

**When to prefer one over the other?**  
Use ArrayList for mostly read operations with random access. Use LinkedList for frequent insertions/deletions, especially in middle.

**Time complexity for adding an element:**  
ArrayList: O(1) amortized at end, O(n) elsewhere  
LinkedList: O(1) if position known, O(n) to find position

**15. What is the Java memory model?**

**Answer:**  
The Java Memory Model specifies how threads interact through memory, ensuring visibility of changes across threads. Key aspects:

* Happens-before relationship
* Memory barriers
* Volatile and synchronized semantics

**Follow-up Answers:**

**What happens during garbage collection?**  
Unreachable objects are identified, their memory is reclaimed, and memory may be compacted.

**16. Explain how the try-catch-finally block works in exception handling.**

**Answer:**

* try: Contains code that might throw exceptions
* catch: Handles specific exceptions
* finally: Always executes (for cleanup), whether exception occurs or not

**Follow-up Answers:**

**What if exception occurs in finally block?**  
The exception propagates up, and if there was an exception in try/catch, it's lost.

**Code where finally doesn't execute:**  
If JVM exits (System.exit()) in try or catch, or if the thread is killed.

**17. What are checked and unchecked exceptions in Java?**

**Answer:**

* **Checked**: Must be declared or caught (Exception and subclasses, except RuntimeException)
* **Unchecked**: Don't need declaration (RuntimeException and subclasses)

**Follow-up Answers:**

**Why is RuntimeException unchecked while IOException is checked?**  
RuntimeExceptions typically indicate programming errors (null pointer, array index), while IOExceptions represent recoverable external conditions.

**18. What is the difference between an interface and an abstract class in Java?**

**Answer:**

* **Interface**: All methods abstract (before Java 8), can't have state, multiple inheritance
* **Abstract Class**: Can have concrete methods and state, single inheritance

**Follow-up Answers:**

**Can an abstract class implement an interface?**  
Yes, and it can choose to implement none, some, or all interface methods.

**Advantages of interfaces:**  
Enable multiple inheritance, define contracts without implementation, support lambda expressions.

**19. What is the significance of the volatile keyword in Java?**

**Answer:**  
volatile ensures:

* Visibility: Changes are immediately visible to other threads
* Prevents instruction reordering  
  But doesn't provide atomicity for compound operations.

**Follow-up Answers:**

**How it relates to Java Memory Model:**  
Establishes happens-before relationship for reads/writes to the variable.

**Difference from synchronized:**  
synchronized provides mutual exclusion and visibility for entire blocks, while volatile only ensures visibility for single variable.

**20. Explain the static keyword in Java.**

**Answer:**  
static indicates class-level (rather than instance-level) members:

* Variables: Shared among all instances
* Methods: Can be called without instance
* Blocks: Executed when class is loaded

**Follow-up Answers:**

**Call non-static from static?**  
No, because non-static methods require an instance (would get "non-static method cannot be referenced from static context").

**Static block example:**

java

Copy

static {

// Initialize static resources

}

**21. What is the use of the transient keyword in Java?**

**Answer:**  
transient marks fields that shouldn't be serialized. Useful for sensitive data or derived fields.

**Follow-up Answers:**

**Why transient isn't inherited?**  
Because serialization works on fields, not inheritance hierarchy. Each class controls its own serialization.

**What happens during serialization?**  
Transient fields are ignored and set to default values during deserialization.

**22. What are wrapper classes in Java?**

**Answer:**  
Object representations of primitives (e.g., Integer for int). Used when objects are required (collections, generics).

**Follow-up Answers:**

**Autoboxing/unboxing:**  
Automatic conversion between primitives and wrappers (e.g., Integer i = 5;).

**Performance overhead:**  
Each conversion creates a new object, which can impact performance in tight loops.

**23. What is the significance of the this keyword in Java?**

**Answer:**  
this refers to the current object instance. Used to:

* Distinguish instance variables from parameters
* Pass current object as parameter
* Call other constructors (this())

**Follow-up Answers:**

**Assign to this?**  
No, this is a final reference.

**Pass this to static method?**  
Allowed but unusual, as static methods don't operate on instances.

**24. Explain multithreading in Java.**

**Answer:**  
Java supports multithreading via:

* Thread class (extend)
* Runnable interface (implement)
* Executor framework
* Synchronization (synchronized, volatile, atomic classes)

**Follow-up Answers:**

**Difference between Thread and Runnable:**  
Runnable is preferred as it separates task from execution and allows multiple inheritance.

**Implement thread safety:**  
Use synchronization, volatile, concurrent collections, immutable objects.

**25. What is the difference between String, StringBuilder, and StringBuffer?**

**Answer:**

* **String**: Immutable, thread-safe
* **StringBuilder**: Mutable, not thread-safe, faster
* **StringBuffer**: Mutable, thread-safe (synchronized), slower

**Follow-up Answers:**

**Which for multithreaded environment?**  
StringBuffer if synchronization needed, otherwise StringBuilder with external synchronization.

**How immutability affects performance:**  
String operations create new objects, which can be inefficient for frequent modifications.

**26. What is the purpose of the synchronized keyword in Java?**

**Answer:**  
synchronized provides thread safety by:

* Ensuring only one thread can execute a method/block at a time
* Establishing happens-before relationships for memory visibility

**Follow-up Answers:**

**Synchronize static method?**  
Yes, it locks on the class object (ClassName.class).

**Drawbacks of excessive synchronization:**  
Can cause contention, reduce performance, and lead to deadlocks.

**27. What is the Java Collections Framework?**

**Answer:**  
A unified architecture for representing and manipulating collections, including:

* Interfaces (List, Set, Map, etc.)
* Implementations (ArrayList, HashSet, HashMap, etc.)
* Algorithms (sorting, searching)

**Follow-up Answers:**

**Key interfaces:**  
Collection, List, Set, Map, Queue, etc.

**How HashMap ensures unique keys:**  
Uses hashCode() and equals() to identify duplicates.

**28. What is garbage collection in Java?**

**Answer:**  
Automatic memory management that reclaims memory from objects no longer in use. Types:

* Minor GC (young generation)
* Major GC (old generation)
* Full GC (entire heap)

**Follow-up Answers:**

**Types of garbage collectors:**  
Serial, Parallel, CMS, G1, ZGC, Shenandoah.

**Force garbage collection?**  
System.gc() suggests GC but doesn't guarantee it.

**29. What are lambda expressions in Java?**

**Answer:**  
Anonymous functions that enable functional programming. Syntax: (parameters) -> expression

**Follow-up Answers:**

**Problem they solve:**  
Verbosity of anonymous classes for single-method interfaces.

**Example:**

java

Copy

List<String> names = Arrays.asList("a", "b");

names.forEach(name -> System.out.println(name));

**30. What is the use of Optional in Java 8?**

**Answer:**  
Optional is a container that may or may not contain a value, helping avoid NullPointerException.

**Follow-up Answers:**

**Avoid NullPointerException:**

java

Copy

Optional<String> opt = Optional.ofNullable(getString());

String value = opt.orElse("default");

**Difference between of() and ofNullable():**  
of() throws NPE for null, ofNullable() allows null.

**31. How does a HashMap work internally?**

**Answer:**  
HashMap uses an array of buckets (nodes):

1. Computes hashCode() of key
2. Maps hash to bucket index
3. Stores key-value pair in bucket (as linked list or tree)

**Follow-up Answers:**

**Collision handling:**  
In Java 8+, buckets use linked lists that convert to trees when large.

**Load factor significance:**  
Threshold (default 0.75) when resize occurs (capacity doubles).

**32. What is a thread pool, and why is it used?**

**Answer:**  
A managed pool of reusable threads. Benefits:

* Reduces thread creation overhead
* Controls resource usage
* Manages thread lifecycle

**Follow-up Answers:**

**newCachedThreadPool() vs newFixedThreadPool():**  
Cached: Grows as needed, shrinks when idle  
Fixed: Fixed size, better for resource control

**33. What are generics in Java?**

**Answer:**  
Generics enable types (classes/interfaces) to be parameters. Benefits:

* Type safety
* Eliminate casting
* Enable algorithms to work on different types

**Follow-up Answers:**

**Why generics exist:**  
Make code more type-safe and reusable.

**Primitive types in generics:**  
No, must use wrapper classes (autoboxing helps).

**34. Explain the concept of class loading in Java.**

**Answer:**  
Process of loading class bytecode into JVM memory. Steps:

1. Loading: Find binary representation
2. Linking: Verify, prepare, resolve
3. Initialization: Execute static initializers

**Follow-up Answers:**

**Class loaders:**  
Bootstrap, Extension, Application.

**Two classes with same name:**  
Can coexist if loaded by different class loaders.

**35. What are the differences between Comparator and Comparable?**

**Answer:**

* **Comparable**: Defines natural ordering (compareTo())
* **Comparator**: External ordering (compare())

**Follow-up Answers:**

**When to use Comparator:**  
When you need multiple orderings or can't modify the class.

**36. What is the purpose of super in Java?**

**Answer:**  
super refers to superclass, used to:

* Call superclass methods
* Access superclass fields
* Invoke superclass constructors

**Follow-up Answers:**

**Call superclass constructor:**

java

Copy

super(params);

**37. Explain the Singleton design pattern in Java.**

**Answer:**  
Ensures a class has only one instance with global access. Implementation:

java

Copy

public class Singleton {

private static final Singleton INSTANCE = new Singleton();

private Singleton() {}

public static Singleton getInstance() { return INSTANCE; }

}

**Follow-up Answers:**

**Thread-safe Singleton:**  
Use enum, static final field, or double-checked locking.

**Issues in distributed systems:**  
Multiple JVMs mean multiple instances.

**38. What are method references in Java 8?**

**Answer:**  
Shorthand for lambdas calling existing methods. Types:

* Static: Class::staticMethod
* Instance: instance::method
* Constructor: Class::new

**Follow-up Answers:**

**Difference from lambdas:**  
More concise when calling existing methods.

**39. What is the default keyword in Java interfaces?**

**Answer:**  
default allows interface methods to have implementations. Added in Java 8 for backward compatibility.

**Follow-up Answers:**

**Override default method:**  
Yes, implementing classes can override.

**Why introduced:**  
Allow interface evolution without breaking existing implementations.

**40. What is a ConcurrentHashMap, and how is it different from HashMap?**

**Answer:**  
ConcurrentHashMap is a thread-safe HashMap that:

* Uses fine-grained locking (not whole map)
* Allows concurrent reads and limited concurrent writes
* Doesn't throw ConcurrentModificationException

**Follow-up Answers:**

**Thread safety:**  
Achieved via segmented locking in Java 7, CAS in Java 8+.

**41. What is the difference between wait() and sleep() in Java?**

**Answer:**

* wait(): Releases lock, must be in synchronized block
* sleep(): Keeps lock, can be called anywhere

**Follow-up Answers:**

**Why wait() in synchronized block:**  
To prevent lost wake-up and race conditions.

**Interrupt sleeping thread:**  
Yes, throws InterruptedException.

**42. Explain how notify() and notifyAll() work in Java.**

**Answer:**

* notify(): Wakes one waiting thread (arbitrary)
* notifyAll(): Wakes all waiting threads

**Follow-up Answers:**

**Difference:**  
Use notify() when all threads can handle the wakeup, notifyAll() when only one can proceed.

**43. What is reflection in Java?**

**Answer:**  
Runtime inspection and modification of classes, methods, fields. Used by frameworks for:

* Dependency injection
* Serialization
* ORM mapping

**Follow-up Answers:**

**Drawbacks:**  
Performance overhead, security issues, breaks encapsulation.

**44. Explain the difference between Serializable and Externalizable.**

**Answer:**

* Serializable: Automatic serialization (uses reflection)
* Externalizable: Manual control (writeExternal/readExternal)

**Follow-up Answers:**

**When to use Externalizable:**  
When you need precise control over serialization format/process.

**45. What is the purpose of the enum keyword in Java?**

**Answer:**  
Defines a fixed set of constants. More powerful than C enums:

* Can have methods, fields
* Implement interfaces
* Type-safe

**Follow-up Answers:**

**Extend enum:**  
No, enums are implicitly final.

**46. What is a deadlock in Java?**

**Answer:**  
When two or more threads wait forever for locks held by each other.

**Follow-up Answers:**

**Example:**  
Thread 1 locks A, wants B  
Thread 2 locks B, wants A

**Prevention:**  
Lock ordering, timeouts, deadlock detection.

**47. Explain the concept of immutability in Java with an example.**

**Answer:**  
Immutable objects cannot be modified after creation. Example: String.

**Follow-up Answers:**

**Why String immutable:**  
Security (e.g., network connections), thread safety, caching hashcode.

**Create immutable class:**

1. Final class
2. Private final fields
3. No setters
4. Defensive copies

**48. What is the difference between throw and throws in Java?**

**Answer:**

* throw: Actually throws an exception
* throws: Declares that a method might throw exceptions

**Follow-up Answers:**

**Throw multiple exceptions:**  
Yes, declare in throws clause, throw based on conditions.

**49. What are functional interfaces in Java?**

**Answer:**  
Interfaces with exactly one abstract method. Enable lambda expressions.

**Follow-up Answers:**

**Built-in examples:**  
Predicate, Function, Consumer, Supplier.

**50. What is the difference between String and char[] for storing passwords in Java?**

**Answer:**

* char[]: Preferred as contents can be wiped (set to zero)
* String: Immutable, stays in memory until garbage collected

**Follow-up Answers:**

**Securely clear char[]:**