1. What is the difference between JDK, JRE, and JVM?

The JVM is the core component responsible for executing Java bytecode and providing platform independence.

The JRE builds on top of the JVM by including the standard class libraries and other runtime components required for running Java applications.

The JDK further extends the JRE by providing development tools such as the compiler and debugging utilities, which are essential for building, packaging, and testing Java programs.

---

2. What are the main differences between primitive types and reference types?

Primitive types store raw values directly and typically reside in stack memory, offering faster access and fixed memory usage.

Reference types, on the other hand, store references that point to objects located in heap memory.

Primitives hold simple, predefined data values, whereas reference types represent complex data structures and can grow dynamically.

Additionally, primitive types have predefined default values, while reference types default to null.

---

3. Is Java pass-by-value or pass-by-reference? Explain.

Java employs a strictly pass-by-value mechanism.

For primitive variables, the passed value is the actual numeric or boolean value.

For reference variables, the value passed is the reference itself, not the underlying object.

This means that while a method may modify the object that the reference points to, it cannot replace the caller’s original reference with a different object.

---

4. Why are Strings immutable in Java?

Strings are immutable to enhance security, optimize memory usage, and ensure thread safety.

Immutability prevents external modification of sensitive data and enables the JVM to store and reuse String objects efficiently through the String Constant Pool.

Since immutable objects cannot change state, they can be safely shared among multiple threads without requiring synchronization.

---

5. What is the String Constant Pool and how does it work?

The String Constant Pool is a dedicated memory region within the JVM that stores unique String literals.

When a String literal is created, the JVM checks this pool to determine whether an identical literal already exists.

If it does, the existing instance is reused; if not, a new one is created and placed into the pool.

This mechanism reduces memory consumption and improves performance by avoiding redundant String allocations.

---

6. What is the purpose of the final keyword in Java?

The `final` keyword enforces non-modifiability at different levels.

When applied to variables, it prevents reassignment.

When applied to methods, it prohibits overriding.

When applied to classes, it disallows inheritance.

Collectively, it enhances safety, stability, and in some cases performance by enabling certain compiler optimizations.

---

7. What does the static keyword mean for variables or methods?

The `static` keyword indicates that a variable or method belongs to the class itself rather than to any individual instance.

Static members are initialized once at class loading time and are shared across all objects of that class.

This is particularly useful for representing class-level state or behavior that does not depend on specific instances.

---

8. What is a static block and when does it run?

A static block is a class-level initialization construct executed once when the class is first loaded by the JVM.

It is typically used to perform complex or multi-step initialization of static variables.

Static blocks run prior to the creation of any instances and before any constructor is invoked.

---

9. Can a static method access non-static variables? Why or why not?

A static method cannot directly access non-static variables because non-static variables are tied to specific object instances, whereas static methods are associated with the class itself.

Without an instance context, the JVM has no reference to determine which object's state should be accessed.

---

10. Describe the JVM loading order: static block, static variables, and constructor.

When a class is loaded, the JVM initializes static variables first according to their declaration order.

Immediately afterward, static blocks execute to complete class-level initialization.

When an object is created, instance variables are initialized next, followed by the execution of the constructor.

Thus, class-level initialization always precedes object-level initialization.

---

11. What is the difference between a static variable and a constant defined as public static final?

A static variable is shared among all instances and may be reassigned.

A constant defined as public static final is both class-level and immutable, meaning its value is fixed at compile time and cannot be altered at runtime.

Such constants typically represent universally applicable values that must remain consistent throughout program execution.

---

12. Explain why immutable objects (like String) are thread-safe by design. Provide an example scenario.

Immutable objects are inherently thread-safe because their state cannot be changed after construction.

As there are no mutating operations, concurrent threads cannot introduce inconsistent states or race conditions.

Multiple threads may safely read and share the same immutable instance without requiring synchronization mechanisms.

---

13. What problem does making a class final solve in terms of immutability and inheritance security?

Declaring a class as final ensures that it cannot be subclassed, which protects the integrity of its behavior and prevents subclasses from overriding methods in ways that could compromise immutability.

This approach eliminates risks associated with inheritance-based state modification and reinforces the security and predictability of the class’s design.