Java Interview Questions

Basic Interview Questions

Q1. What are the data types in Java? Java has two types of data types:

1. Primitive Data Types: These include byte, short, int, long, float, double, char, and boolean. They store simple values and are not objects.
2. Non-Primitive Data Types: These include String, Array, Class, and Interface. They are derived from primitive data types and provide more functionalities.

Q2. What are wrapper classes?

Wrapper classes provide an object representation of primitive data types, such as Integers, Doubles, and Booleans. These classes allow primitives to be used in collections and provide useful utility methods.

Q3. Are there dynamic arrays in Java?

Java arrays are fixed indynamically. Q4 size. However, ArrayList (from the Java.util package) provides a dynamic array implementation where elements can be added or removed dynamically.

Q4. What is JVM?

The Java Virtual Machine (JVM) is a part of the Java Runtime Environment (JRE). It is responsible for executing Java bytecode by converting it into machine code specific to the operating system.

Q5. Why is Java platform-independent?

Java achieves platform independence through bytecode. The Java compiler converts code into bytecode, which the JVM interprets for the underlying OS, making Java write-once, run-anywhere.

Q6. What are local and global variables?

- Local variables are declared inside methods or blocks and are accessible only within their scope.
- Global variables (also called instance variables) are declared within a class but outside any method and have a wider scope.

Q7. What is data encapsulation?

Encapsulation is an OOP principle where data (variables) and code (methods) are bundled into a single unit (class). It restricts direct access to data using access modifiers (private, protected).

Q8. What is function overloading?

Function overloading allows multiple methods to have the same name but different parameter lists. The compiler differentiates them based on the number or type of parameters.

Example:

```
public class Figure
{
     public int area(int a, int b)
     {
     int rectangleArea = a*b;
     return rectangleArea;
     }
     public  int area(int a)
     {
     int squareArea = a*a;
     return squareArea;
     }
public static void main(String[] args ){
     Figure f = new Figure();
     System.out.println("Area of square " + f.area(5));
     System.out.println("Area of Rectangle " + f.area(5,3));
     }
}
```

Q9. What is function overriding?

Overriding allows a subclass to provide a specific implementation of a method defined in its superclass. It enables dynamic method dispatch (runtime polymorphism).

Q10. Why is the main method static in Java?

The main method is static so that it can be called without creating an instance of the class, allowing the program to start execution without object instantiation.

Q11. What is the difference between the throw and throws keywords in Java?

Feature	throw	throws
Purpose	`Used to explicitly throw an exception`	`Declares that a method may throw an exception`
Usage	`throw new Exception("Error")`	`public void myMethod() throws IOException`
Number of Exceptions	`Can throw one exception at a time`	`Can declare multiple exceptions using commas`

Q12. What do you mean by singleton class?

A singleton class ensures that only one instance of the class exists throughout the application's lifecycle. It is implemented using a private constructor, a static instance variable, and a public static method that returns the single instance. The most common way to create a singleton class is using the lazy initialization or eager initialization approach.

Q13. Does every try block need a catch block?

No, a try block does not necessarily need a catch block. It can be followed by either a catch block, a final block, or both. A catch block handles exceptions that may arise in the try block, while a final block ensures that certain code

(such as resource cleanup) is executed regardless of whether an exception occurs.

Q14. What is the usage of the super keyword in Java?

The super keyword in Java is used to refer to the parent class. It can be used to:

1. Call the constructor of the parent class.
2. Access the parent class's methods and variables when they are overridden in a subclass.
3. Differentiate between methods and attributes of the parent and child class when they have the same name.

Q15. What do you mean by the final keyword?

The final keyword is used to restrict modifications in Java. It can be applied in three contexts:

1. Final variable: Its value cannot be changed once assigned.
2. Final method: Prevents method overriding in subclasses.
3. Final class: Prevents inheritance by other classes.

Q16. How is an exception handled in Java?

Java handles exceptions using the try-catch-finally mechanism:

1. Try block: Contains the code that might generate an exception.
2. Catch block: Handles the exception and defines what should be done when an error occurs.
3. Finally block: Executes regardless of whether an exception occurs or not, often used for resource cleanup (e.g., closing files or database connections).

Q17. How can objects in a Java class be prevented from serialization?

Serialization converts an object into a byte stream for storage or transmission. To prevent serialization:

1. Declare fields as transient to exclude them from serialization.
2. Implement writeObject() and readObject() methods to control serialization.
3. Extend NotSerializableException to explicitly prevent serialization.

Q18. What is the difference between a constructor and a method in Java?

Constructor	Method
It has no return type.	It always has a return type. It has a return type void when not returning anything.
It always has the same name as the class name.	It can have any name of its choice.

Q19. Why is reflection used in Java?

Reflection in Java allows a running program to inspect and manipulate its methods, fields, and constructors at runtime. It is commonly used in frameworks, debugging tools, and JavaBeans to dynamically access class properties.

Q20. What are the different types of ClassLoaders in Java?

Java provides three main types of ClassLoaders:

1. Bootstrap ClassLoader: Loads core Java classes from rt.jar and other essential libraries. It is implemented in native code and does not have a Java class representation.
2. Extension ClassLoader: Loads classes from the JRE/lib/ext directory or any other specified extension directories. It is implemented as sun.misc.Launcher$ExtClassLoader.
3. System (Application) ClassLoader: Loads application classes from the classpath (defined by CLASSPATH, -cp, or -classpath options). It is a child of the Extension ClassLoader.

Q21. What is a copy constructor in Java?

A copy constructor creates a new object by copying the properties of an existing object. It takes an instance of the same class as an argument and initializes the new object with the same values.

Q22. What is object cloning in Java?

Object cloning is a way to create an exact copy of an object. Java provides the clone() method from the Cloneable interface to perform shallow copies. A shallow copy copies field values but does not duplicate referenced objects, while a deep copy creates new instances of referenced objects.

Q23. Is Java a purely object-oriented language?

No, Java is not purely object-oriented because it supports primitive data types like int, char, boolean, and double, which are not objects. A purely object-oriented language would require every entity to be an object.

Q24. What is a package in Java?

A package in Java is a collection of related classes and interfaces grouped to organize code and prevent naming conflicts.

- Built-in packages: java.lang, java.util, etc.
- User-defined packages: Created by developers for organizing custom classes.

Q25. What is coercion in Java?

Coercion in Java refers to the automatic or explicit conversion of one data type into another.

- Implicit coercion: Automatically converts smaller data types to larger ones (e.g., int to double).
- Explicit coercion (casting): Converts larger data types to smaller ones using type casting (e.g., (int) 3.14).

Q26. Can a private method be overridden in Java?

No, private methods cannot be overridden because they are not accessible outside their class. If a subclass defines a method with the same name, it is

treated as a separate method rather than an override.

Q27. What are the phases in the lifecycle of a thread in Java?

A Java thread goes through the following states:

1. New: The thread is created but has not started executing.
2. Runnable: The thread is ready to run and waiting for CPU allocation.
3. Blocked: The thread is waiting for a resource or lock to be available.
4. Waiting: The thread is indefinitely waiting for another thread to notify it.
5. Timed Waiting: The thread waits for a specified time (e.g., using Thread.sleep()).
6. Terminated: The thread has completed execution or stopped due to an error.

Q28. What is a marker interface in Java?

A marker interface is an interface with no methods or fields, used to provide metadata to the JVM or compiler. Examples include Serializable and Cloneable. Modern Java prefers annotations over marker interfaces.

Q29. What is a memory leak in Java?

A memory leak occurs when objects that are no longer needed are not garbage collected because they are still referenced somewhere. This can cause excessive memory consumption and slow down the application.

Q30. What is the difference between new and newInstance() in Java?

- New is a keyword that creates a new object of a known class at compile time.
- newInstance() (from Class) creates an object dynamically at runtime, requiring reflection, and is slower because it involves additional security and access checks.

Q31. What is the difference between JDK, JRE, and JVM?

- **JDK (Java Development Kit)** – Contains **JRE + development tools** (compiler, debugger) for **developing and running** Java applications.
- **JRE (Java Runtime Environment)** – Includes **JVM + libraries** needed to **run** Java applications but lacks development tools.
- **JVM (Java Virtual Machine)** – Executes Java bytecode, providing **platform independence** and **memory management (GC)**.

JDK > JRE > JVM – JDK includes JRE, and JRE includes JVM.
JDK is for developers, while **JRE is for users running Java applications**

Q32. What is the difference between abstraction and encapsulation?

- Abstraction hides implementation details and exposes only essential functionalities (e.g., using interfaces and abstract classes).
- Encapsulation bundles data and methods within a class and restricts direct access using access modifiers.

Q33. What is inheritance in Java?

Inheritance in Java is a mechanism where a **child class acquires properties and behaviors** of a **parent class**, promoting **code reusability** and **hierarchical relationships**.

- Achieved using the `extends` **keyword**.
- Supports **single and multilevel inheritance** (not multiple inheritance with classes).
- Allows method **overriding** for polymorphism.
- **The super (`super`) keyword** is used to access parent class members.

Q34. What are functional interfaces in Java 8?

Functional interfaces have exactly one abstract method and are used with lambda expressions.

Examples include Runnable, Callable, and Comparator.

Q35. What is polymorphism in Java?

Polymorphism allows the same method to behave differently based on the context.

- Compile-time polymorphism (Method Overloading): Methods with the same name but different parameters.
- Runtime polymorphism (Method Overriding): A subclass provides a specific implementation of a parent method.

Q36. What is the purpose of the default keyword in interfaces?

The default keyword allows methods in interfaces to have default implementations, enabling backward compatibility without forcing all implementing classes to override them.

Q37. What is an interface in Java?

An **interface** in Java is a **blueprint** for classes that defines a **contract** without implementation.

- Declared using the `interface` **keyword**.
- Contains **only abstract methods** (until Java 7).
- **Java 8+** allows **default and static methods** with implementations.
- Supports **multiple inheritance**.
- Implemented by classes using the `implements` **keyword**.

Q38. What is the difference between ArrayList and Vector?

- ArrayList is not synchronized (faster), while Vector is synchronized (thread-safe).
- ArrayList increases its size by 50% when full, while Vector doubles its size.

Q39. What is an abstract class?

An **abstract class** in Java is a class that **cannot be instantiated** and is meant to be **extended by subclasses**.

- Declared using the `abstract` **keyword**.
- Can have **both abstract (without implementation) and concrete methods**.
- Used for **partial implementation and code reusability**.
- **Must be extended** by a subclass that provides implementations for abstract methods.

Q40. What is the difference between HashMap and ConcurrentHashMap?

- HashMap is not thread-safe, while ConcurrentHashMap is thread-safe.
- ConcurrentHashMap locks only portions of the map, improving performance.
- HashMap allows one null key, but ConcurrentHashMap does not.

Q41. What is the difference between an abstract class and an interface?

- Abstract class: Can have both abstract and concrete methods.
- Interface: Contains only abstract methods (before Java 8) and supports multiple inheritance.

Q42. What is the Java Memory Model (JMM)?

The **Java Memory Model (JMM)** defines how **threads interact with memory** and ensures **visibility, ordering, and atomicity** of shared data in a **multi-threaded environment**.

- Controls how **variables are read/written across threads**.
- Ensures **happens-before relationships** to prevent race conditions.
- Uses **volatile, synchronized, and locks** for thread safety.

- Helps in **optimizing CPU caching and instruction reordering**.
- Ensures **safe and predictable concurrency behavior**.

Q43. What is this keyword in Java?

This refers to the current instance of a class, distinguishing between instance variables and parameters with the same name.

Q44. What are Java Generics?

Generics provide compile-time type safety by allowing a class, method, or interface to work with different types while avoiding runtime errors.

Q45. What are access modifiers in Java?

- Private: Accessible only within the same class.
- Default: Accessible within the same package.
- Protected: Accessible within the same package and subclasses.
- Public: Accessible from anywhere.

Q46. What is the purpose of the synchronized keyword?

Synchronized ensures that only one thread can execute a block of code or method at a time, preventing race conditions.

Q47. What is a static method in Java?

A **static method** in Java belongs to the **class**, not instances. It can be called using the **class name** without creating an object.

- Declared using the `static` **keyword**.
- Can **access only static variables and methods** directly.
- Cannot use `this` **or** `super`.
- Commonly used for **utility methods** (e.g., `Math.pow()`).

```
class Example {
   static void display() {
```

```
    System.out.println("Static Method");
  }
}
                Example.display(); // Call without creating an object
```

Q48. What are Java 8 Streams?

Java 8 **Streams** provides a **functional programming** approach to processing data efficiently. They allow operations like **filtering, mapping, and reducing** collections in a **declarative and parallelizable** way.

- Supports **sequential (`stream()`) and parallel (`parallelStream()`) processing**.
- Uses **lazy evaluation** for optimized execution.
- Common methods: `filter()`, `map()`, `reduce()`, `collect()`, `forEach()`.

Q49. What is garbage collection in Java?

1. **Automatic Memory Management**: Java's Garbage Collector (GC) automatically reclaims memory by removing unused objects.
2. **Heap Memory Cleanup**: GC works in the heap, where objects are dynamically allocated.
3. **Identifies Unreachable Objects**: Objects with no active references are eligible for garbage collection.
4. **Prevents Memory Leaks**: Helps manage memory efficiently and avoids out-of-memory errors.
5. **No Manual Deallocation**: Unlike languages like C/C++, Java does not require explicit `free()` or `delete()`.
6. **Uses Mark and Sweep Algorithm**: Identifies live objects (mark) and removes dead ones (sweep).
7. **JVM Optimization for GC**: JVM parameters like `-Xms`, `-Xmx`, `-XX:+UseG1GC` help tune GC performance.

Q50. What is the difference between implements and extends?

- Implements are used for interfaces.
- Extends are used for class inheritance.

Intermediate Interview Questions:

Q51. What is the purpose of the "assert" statement in Java?

The assert statement in Java is used to validate assumptions during development and debugging. It checks whether a given expression evaluates to true. If the condition is false, an AssertionError is thrown.

Assertions are mainly used for debugging and testing purposes to detect logical errors early in the development process.

Q52. What is the difference between ArrayList and LinkedList in Java?

- ArrayList: A resizable array-based data structure that provides fast random access (O(1)) but slower insertions and deletions (O(n)) due to shifting elements. It is preferred when frequent access operations are needed.
- LinkedList: A doubly linked list implementation that allows efficient insertions and deletions (O(1)) but slower random access (O(n)). It is more suitable for scenarios with frequent modifications and dynamic resizing.

Q53. What is the purpose of the hashCode() method in Java?

The hashCode() method generates a unique integer value (hash code) that represents an object's contents. It is primarily used in hash-based collections like HashMap, HashSet, and Hashtable for efficient storage and retrieval. A properly implemented hashCode() ensures better performance in hashing-based operations.

Q54. What is the purpose of the toString() method in Java?

The toString() method returns a string representation of an object, typically including its class name and key attributes. It enhances debugging, logging, and object readability. When an object is printed or concatenated with a string, the toString() method is implicitly called. Developers often override it to provide meaningful output.

Q55. How is encapsulation achieved in Java?

Encapsulation in Java is achieved using access modifiers (private, protected, public) to restrict direct access to class members. It ensures data hiding and maintains control over how data is accessed and modified. Getters and setters are commonly used to enforce controlled access to private variables.

Q56. What are method references in Java?

Method references provide a shorthand way to refer to existing methods without executing them immediately. They improve code readability by replacing lambda expressions with direct references to class or instance methods. They are used with functional interfaces and written as Class::methodName.

Example: System.out::println instead of x -> System.out.println(x).

Q57. What are annotations in Java?

Annotations in Java are metadata tags used to provide additional information about code elements such as classes, methods, and variables.

They start with @ (e.g., @Override, @Deprecated) and are used for configuration, code documentation, validation, and runtime processing. Annotations enhance code organization and facilitate frameworks like Spring and Hibernate.

Q58. What is the BitSet class used for in Java?

The BitSet class represents a sequence of bits that can grow dynamically. It provides efficient bitwise operations such as setting, clearing, flipping, and checking bits. It is used for memory-efficient handling of binary data, such as flags, filters, and permission settings.

Q59. What is a CyclicBarrier in Java?

A CyclicBarrier is a synchronization mechanism that allows multiple threads to wait at a common barrier point until all threads reach it. Once all participating threads arrive, they proceed together. It is useful in parallel programming scenarios where tasks must be synchronized before continuing execution.

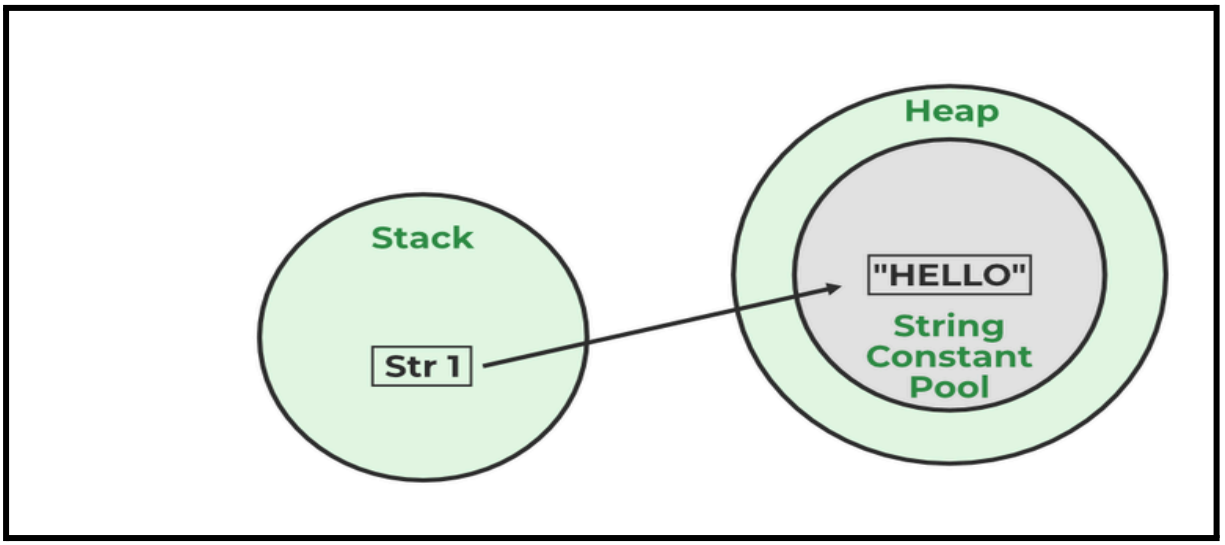
Q60. What are the types of JDBC statements in Java?

JDBC (Java Database Connectivity) provides three types of statements for database interaction:

- Statement: Executes simple SQL queries without parameters.
- PreparedStatement: Precompiled SQL statement with placeholders for parameters, improving performance and security.
- CallableStatement: Used for executing stored procedures in the database.

Q61. What is the Java String Pool?

The Java String Pool is a memory optimization technique where the JVM stores string literals in a common pool to avoid redundant allocations. When a new string literal is created, the JVM checks the pool first; if the string exists, it reuses the reference instead of creating a new object. This conserves memory and enhances performance.



Example:

```
String str1="Hello";
// "Hello" will be stored in String Pool
// str1 will be stored in stack memory
```

Q62. What is the difference between Path and Classpath?

1. Path specifies the location of **system executables** like `Java` and `javac`.

   Classpath specifies the location of **Java class files, JARs, and resources**.

2. Path is used by the **operating system**, while Classpath is used by the **JVM**.
3. The path is set via the `PATH` environment variable, whereas Classpath is set using `CLASSPATH` or `-cp` option.
4. Example: `set PATH=C:\Java\bin` (Path), `set CLASSPATH=C:\libs\myLib.jar` (Classpath).

Q63. What is the difference between Heap and Stack memory?

- Heap Memory: Stores objects and data that need to persist throughout the program. Managed by the JVM's garbage collector.
- Stack Memory: Holds method call frames and local variables. It manages method execution and automatically deallocates memory when a method exits.

Q64. Can we use String with a switch-case statement?

Yes, **String** can be used in a **switch-case** statement from **Java 7 onwards**. The switch works by **computing the hashcode** of the string and then comparing it.

```
String fruit = "Apple";
switch (fruit) {
   case "Apple": System.out.println("It's an Apple!"); break;
   case "Mango": System.out.println("It's a Mango!"); break;
   default: System.out.println("Unknown fruit");
```

```
}
```

Key Points:

- **Case labels are case-sensitive** (`"Apple" ≠ "apple"`).
- **Uses `String.hashCode()` for comparison** internally.
- **Less efficient than integer-based switches** due to hashing.

Q65. What are the different types of class loaders?

Java has three primary class loaders:

1. Bootstrap ClassLoader – Loads core Java classes (e.g., rt.jar).
2. Extension ClassLoader – Loads classes from the JDK's extensions directory ($JAVA_HOME/lib/ext).
3. System (Application) ClassLoader – Loads application classes from the classpath, configurable using -cp or -classpath.

Q66. What is the difference between fail-fast and fail-safe iterators?

- Fail-fast iterators: Immediately throw a ConcurrentModificationException if a collection is modified while iterating.
- Fail-safe iterators: Operate on a cloned copy of the collection, allowing modifications without exceptions.

Q67. What is a compile-time constant in Java?

A compile-time constant is a value assigned at the time of compilation and remains unchanged throughout execution. Example:

static final int MAX_VALUE = 100;

Q68. What is the difference between Map and Queue in Java?

- Map: Stores key-value pairs and allow fast retrieval based on keys (e.g., HashMap).
- Queue: Stores elements in a specific order (FIFO, Priority-based, etc.), designed for processing elements sequentially.

Q69. What is the difference between LinkedHashMap and PriorityQueue?

- LinkedHashMap: Maintains insertion order and maps keys to values.
- PriorityQueue: Orders elements based on priority (natural ordering or a custom comparator).

Q70. What is a memory-mapped buffer in Java?

A memory-mapped buffer allows a file to be directly mapped into memory, improving I/O efficiency by enabling direct access to file contents. Useful for handling large files.

Q71. What is the difference between notify() and notifyAll()?

- notify(): Wakes up one waiting thread.
- notifyAll(): Wakes up all waiting threads.

Q72. What are the types of exceptions in Java?

1. Checked Exceptions (e.g., IOException) – Must be handled at compile-time.
2. Unchecked Exceptions (e.g., NullPointerException) – Occurs at runtime and doesn't require explicit handling.

Q73. What is OutOfMemoryError?

- Error when **JVM runs out of memory**.
- Caused by **memory leaks, large objects, or infinite loops**.

Q74. What is the difference between == and equals()?

- == compares references (memory addresses).
- equals() compares object content (must be overridden in custom classes).

Q75. How can you concatenate multiple strings in Java?

1. **+ operator** – Simple but creates new objects.
2. **concat()** – Works with String.
3. **StringBuilder.append()** – **Efficient**, recommended for multiple concatenations.
4. **String.join()** – Introduced in **Java 8**.

Q76. What are the main differences between Array and Collection?

- Array: Fixed size, stores elements of the same type.
- Collection: Dynamic size, can store elements of different types (e.g., ArrayList, HashSet).

Q77. What is BlockingQueue in Java?

1. A **thread-safe queue** that blocks when empty/full.
2. Used in **producer-consumer** patterns.
3. Example: ArrayBlockingQueue, LinkedBlockingQueue.

Q78. What is the difference between a process and a thread?

- Process: Independent execution unit with its own memory space.
- Thread: Lightweight unit within a process, sharing memory with other threads.

Q79. What are the advantages of multithreading?

- Efficient CPU utilization
- Faster task execution
- Improved application responsiveness
- Better handling of I/O operations

Q80. What is context switching?

1. The process of saving a thread's state and switching to another.
2. Happens in **multi-threading and multitasking**.
3. Causes **CPU overhead**.

Q81. What is the difference between Array and ArrayList?

- Array: Fixed-size, stores primitive types and objects.
- ArrayList: Dynamic-size, stores only objects, provides built-in resizing and utility methods.

Q82. What is the purpose of the `volatile` keyword?

1. Ensures **visibility** of shared variable updates across threads.
2. Prevents **instruction reordering**.
3. Used in **multi-threading** scenarios.

Q83. Explain Java NIO.

Java NIO (New I/O) provides faster I/O operations using:

- Channels & Buffers for data transfer
- Non-blocking I/O for asynchronous processing
- Selectors for managing multiple channels with a single thread

---

Q84. Difference between String, StringBuilder, and StringBuffer?

- `String` – Immutable, slower for modifications.
- `StringBuilder` – Mutable, **fast**, not thread-safe.
- `StringBuffer` – Mutable, **thread-safe** (synchronized).

Q85. Difference between `Runnable` and `Callable`?

- `Runnable` – No return value, cannot throw checked exceptions.
- `Callable` – Returns a value (`Future<T>`), and can throw checked exceptions.

Q86. What is a checked exception in Java?

1. Exceptions checked at compile-time.
2. Must be handled using try-catch or throws.
3. Examples: IOException, SQLException.

Q87. Difference between static and final keywords?

- Static – Belongs to the class, shared across instances.
- Final – Makes variables constant, prevents method overriding & class inheritance.

Q88. What is an unchecked exception?

1. **Runtime exceptions** that do **not require handling** (try-catch).
2. Examples: NullPointerException, ArrayIndexOutOfBoundsException.
3. **Occurs due to programming errors**.

Q89. Explain Java Lambda Expressions.

1. Introduced in **Java 8** to enable **functional programming**.
2. Provides a **concise way** to write **anonymous functions**.
3. Used with **functional interfaces (`Runnable`, `Comparator`)**.

Example:

```
(a, b) -> a + b
```

Q90. Purpose of the try-catch block?

1. Handles runtime exceptions to prevent program crashes.
2. Try contains risky code, and catch handles exceptions.
3. Ensures graceful error handling.

Example:

```
try { int x = 10 / 0; } catch (ArithmeticException e) {
System.out.println("Error!"); }
```

Q91. Difference between Collection and Collections?

- Collection: Interface representing data structures (e.g., List, Set).
- Collections: A utility class for collection operations (e.g., sorting, searching).

Q92. Use of the `finalize()` method?

1. Called by a Garbage Collector before an object is reclaimed.
2. Used for cleanup operations (e.g., closing files, releasing resources).
3. Deprecated in Java 9 due to unpredictable behavior.

Q93. Explain Java 8 Optional.

1. A **container object** for handling **null values safely**.
2. Helps avoid **`NullPointerException`**.
3. Methods: `of()`, `ofNullable()`, `isPresent()`, `orElse()`, `map()`.

Q94. What is the Java Collection Framework?

1. A **set of classes & interfaces** for managing groups of objects.
2. Includes **List (ArrayList), Set (HashSet), Map (HashMap), Queue (PriorityQueue)**, etc.
3. Provides **efficient data structures & algorithms** for handling collections.

Q95. Explain Java 8 CompletableFuture.

1. A **non-blocking** way to handle **asynchronous programming**.
2. Supports **callbacks (`thenApply()`, `thenAccept()`)** and **chaining multiple tasks**.
3. Uses the **ForkJoinPool** for efficient execution.

Q96. What is multithreading in Java?

1. Allows **parallel execution** of multiple tasks (threads).
2. Improves **CPU utilization & performance**.
3. Created using **Thread class** or **Runnable interface**.
4. Managed using **Executors, Fork/Join, and CompletableFuture**.

Q97. Purpose of the `strictfp` keyword?

1. Ensures **consistent floating-point calculations** across platforms.

2. Enforces **IEEE 754 standard** for precision.
3. Used with **classes & methods** but not variables.

Q98. Difference between sleep() and wait()?

- sleep(): Pauses thread but retains lock.
- wait(): Releases lock and waits for notify().

Q99. Difference between LinkedHashSet and TreeSet?

- LinkedHashSet maintains **insertion order**, while TreeSet maintains **sorted order** (natural or custom comparator).
- LinkedHashSet uses a **HashMap with a linked list**, whereas TreeSet is based on a **Red-Black Tree**.
- LinkedHashSet has **O(1)** time complexity for add, remove, and contains, while TreeSet has **O(log n)** due to tree balancing.
- LinkedHashSet **allows null elements**, but TreeSet **does not allow null** (throws NullPointerException).
- LinkedHashSet is best for **maintaining insertion order with fast lookups**, while TreeSet is ideal for **sorting unique elements and range queries**.

Q100. What is synchronization in Java?

Synchronization in Java is a mechanism that ensures **thread safety** by controlling access to shared resources in a **multi-threaded environment**. It prevents **race conditions** and ensures **data consistency** when multiple threads access a shared object. Java provides synchronization through the **synchronized keyword**, which can be applied at the method or block level to allow only one thread to execute a critical section at a time.

Additionally, **explicit locks (ReentrantLock)**, **volatile variables**, and **atomic classes (AtomicInteger, ConcurrentHashMap)** offer alternative ways to handle concurrency efficiently. While synchronization ensures safe execution, excessive use can lead to **performance overhead** due to thread blocking

and context switching.

Advanced Interview Questions:

Q101. How does the JVM handle method overloading and overriding internally?

Method overloading is resolved at compile-time, where the compiler determines the method signature based on argument types. Method overriding, on the other hand, is resolved at runtime using dynamic method dispatch, where the JVM decides which method to invoke based on the actual object type.

Q102. Explain Java's Non-blocking Algorithms and how they differ from traditional blocking algorithms.

Non-blocking algorithms allow multiple threads to operate on shared data without using locks, improving performance and avoiding issues like deadlocks. In contrast, traditional blocking algorithms use locks to synchronize access, potentially leading to contention and reduced concurrency.

Q103. How can you create an immutable class in Java?

To create an immutable class:

- Declare the class as final.
- Make all fields private and final.
- Do not provide setters.
- Ensure deep copies of mutable fields in constructors and getters.
- Avoid exposing mutable references.

Q104. Describe the Contract of `hashCode()` and `equals()` methods.

- If two objects are equal according to `equals()`, they must have the same `hashCode()`.
- However, two objects with the same `hashCode()` may not necessarily be equal.
- Properly overriding both methods ensures correct behavior in hash-based collections like `HashMap` and `HashSet`.

Q105. What is Type Inference in Generics?

Inner Workings of `ConcurrentHashMap`

`ConcurrentHashMap` in Java is a **thread-safe, high-performance** alternative to `HashMap`, optimized for concurrent access.

How It Works:

1. **Segmented Locking (Java 7 and earlier)** – The map was divided into **segments**, reducing lock contention.
2. **CAS (Compare-And-Swap) & Fine-Grained Locking (Java 8+)** – Instead of segments, it uses **locks at bucket level** and **CAS operations** for updates.
3. **Bucket-Level Synchronization** – Write operations lock only the affected bucket, allowing **parallel updates**.
4. **No Lock for Reads** – Most read operations (`get()`) are **lock-free**, ensuring high performance.
5. **Improved Scalability** – Reduces contention, making it efficient for multi-threaded environments.

Key Benefits:

✅ **Faster than `Collections.synchronizedMap()`** due to reduced locking.
✅ **Supports high-concurrency scenarios** like caching and real-time analytics.
✅ **Atomic operations** (`putIfAbsent()`, `compute()`, etc.) ensure thread safety.

Q106. Explain the inner workings of `ConcurrentHashMap`.

Inner Workings of `ConcurrentHashMap`: `ConcurrentHashMap` in Java is a **thread-safe, high-performance** alternative to `HashMap`, optimized for concurrent access.

How It Works:

6. **Segmented Locking (Java 7 and earlier)** – The map was divided into **segments**, reducing lock contention.
7. **CAS (Compare-And-Swap) & Fine-Grained Locking (Java 8+)** – Instead of segments, it uses **locks at bucket level** and **CAS operations** for updates.
8. **Bucket-Level Synchronization** – Write operations lock only the affected bucket, allowing **parallel updates**.
9. **No Lock for Reads** – Most read operations (`get()`) are **lock-free**, ensuring high performance.
10. **Improved Scalability** – Reduces contention, making it efficient for multi-threaded environments.

Key Benefits:

✅ **Faster than `Collections.synchronizedMap()`** due to reduced locking.
✅ **Supports high-concurrency scenarios** like caching and real-time analytics.
✅ **Atomic operations** (`putIfAbsent()`, `compute()`, etc.) ensure thread safety.

Q107. Describe the Java Memory Fence and its importance.

Java Memory Fence & Its Importance

A **Memory Fence** (or **Memory Barrier**) in Java ensures **proper ordering of memory operations** in a multi-threaded environment, preventing **reordering by the CPU or compiler**.

How It Works:

1. **Prevents Instruction Reordering** – Ensures operations happen in the expected sequence.
2. **Ensures Visibility Across Threads** – Changes made by one thread become visible to others.
3. **Used in Concurrency Primitives** – Found in `volatile`, `synchronized`, and `Lock` implementations.

Why It's Important?

✅ **Prevents data races & inconsistencies** in multi-threaded applications.
✅ **Ensures correct execution order** in concurrent programming.
✅ **Optimizes performance while maintaining thread safety.**

Q108. Explain the concept of False Sharing in Java.

False sharing occurs when multiple threads modify variables that reside in the same CPU cache line, causing unnecessary cache invalidations and performance degradation.

Q109. How do you handle OutOfMemoryError in Java?

To handle OutOfMemoryError:

- Optimize memory usage and avoid memory leaks.
- Increase heap size using JVM options (-Xmx).
- Use efficient data structures.
- Employ garbage collection tuning strategies.
- Implement graceful degradation mechanisms.

Q110. Describe Escape Analysis in Java optimizations.

Escape analysis is a compiler optimization technique that determines whether an object can be allocated on the stack instead of the heap, reducing memory allocation overhead and improving performance.

Q111. Explain Thread Starvation and how to prevent it.

Thread starvation occurs when low-priority threads are indefinitely delayed due to high-priority threads monopolizing CPU time.

It can be prevented by:

- Using fair locks (ReentrantLock with fairness parameter).
- Adjusting thread priorities carefully.
- Ensuring resource allocation is balanced.

Q112. How does Class Data Sharing (CDS) work in JVM?

Class Data Sharing (CDS) is a JVM optimization technique that reduces startup time and memory usage by preloading and sharing class metadata across JVM instances.

How It Works:

1. CDS Archive Creation – During JVM execution, class metadata is stored in a shared archive file.
2. Faster Startup – On subsequent runs, JVM loads classes from the precompiled archive, avoiding costly parsing.
3. Memory Efficiency – Multiple JVM instances share the same metadata, reducing RAM usage.

Q113. Explain Java's Thread-Local Allocation Buffers (TLAB).

Java's Thread-Local Allocation Buffers (TLAB)

Thread-Local Allocation Buffers (TLAB) is a JVM optimization that improves object allocation speed by providing each thread a private memory buffer in the Young Generation heap.

How It Works:

1. Each thread gets a small, preallocated space in the Eden space.
2. Fast object allocation occurs within this space, avoiding synchronization.
3. When TLAB is full, objects spill over to the shared heap, triggering GC if needed.

Benefits:

✅ Faster Object Allocation – No locking required, reducing contention.
✅ Improved GC Performance – Most objects remain in Eden, reducing Old Gen promotion.
✅ Better Multi-Threading Efficiency – Each thread allocates independently.

Q114. How does the Fork/Join Framework work?

The **Fork/Join Framework** (introduced in Java 7) is designed for **parallel execution** of tasks using a **divide-and-conquer** approach.

How It Works:

1. **Task Splitting (`fork()`)** – A task is **split into smaller subtasks** recursively.
2. **Parallel Execution** – The subtasks run in a **ForkJoinPool**, utilizing multiple CPU cores.
3. **Task Joining (`join()`)** – The results of subtasks are **combined** once the computation is complete.
4. **Work Stealing** – Idle threads **steal tasks** from busy threads, improving efficiency.

Q115. What is the difference between Pessimistic Locking and Optimistic Locking?

- Pessimistic Locking: Locks resources throughout the transaction, preventing concurrent modifications.
- Optimistic Locking: Allows concurrent access but verifies data consistency before committing changes, reducing contention.

Q116. Explain the inner workings of a Java Agent.

Java Agents use the java.lang.instrument package to modify bytecode at runtime or load-time for profiling, monitoring, or altering application behavior dynamically.

Q117. Describe the uses of Java Compiler API.

Uses of Java Compiler API

The **Java Compiler API (`javax.tools.JavaCompiler`)** allows **programmatic compilation** of Java code within applications.

Key Uses:

1. **Dynamic Code Compilation** – Compile Java code at runtime (e.g., scripting engines).
2. **Custom IDEs & Tools** – Used in IDEs for **syntax checking and on-the-fly compilation**.
3. **Code Generation Frameworks** – Supports frameworks like **annotation processors**.

4. **Plugin Systems** – Enables plugins to compile and execute custom Java code.
5. **Testing & Automation** – Helps run dynamically generated test cases.

Q118. What is the Principle of Locality, and how does it apply to Java?

The Principle of Locality states that programs tend to reuse the same set of memory locations frequently. Java optimizes cache utilization by keeping frequently accessed data close to the CPU.

Q119. How does the G1 Garbage Collector work?

G1 GC divides the heap into regions and prioritizes collecting regions with the most garbage, improving performance and reducing pause times.

Q120. Explain Polymorphic Inline Caching (PIC).

Polymorphic Inline Caching (PIC) in Java

Polymorphic Inline Caching (PIC) is a JIT optimization technique that speeds up method dispatch in dynamically-typed and polymorphic code.

How Does It Work?

1. Caches multiple method targets instead of looking them up every time.
2. Reduces virtual method call overhead by optimizing frequently called methods.
3. Improves performance in polymorphic scenarios where multiple subclasses override a method.

Impact on Performance:

✅ **Faster method dispatch** than standard virtual calls.
✅ **Reduces CPU cache misses** by keeping method lookups efficient.
✅ **Common in JVM JIT compilers** like HotSpot.

Q121. What is the difference between StampedLock and ReentrantLock?

Difference Between StampedLock and ReentrantLock

1. **Locking Mechanism**
    - StampedLock: Provides **optimistic and pessimistic** locking, improving read performance.
    - ReentrantLock: Traditional **exclusive locking**, allowing multiple reentrant acquisitions by the same thread.
2. **Read Performance**
    - StampedLock: Supports **optimistic reads** without blocking, making it faster.
    - ReentrantLock: This does not support optimistic reads, and always requires acquiring a lock.
3. **Write Locking**
    - StampedLock: Provides an **exclusive write lock**, invalidating optimistic read stamps.
    - ReentrantLock: Standard exclusive lock with thread reentrancy.
4. **Interruptibility**
    - StampedLock: **Write locks are not interruptible**, making them less flexible in some cases.
    - ReentrantLock: Supports **interruptible locking**, useful for responsiveness.
5. **Use Case**
    - StampedLock: Best for **high-read, low-write scenarios** (e.g., caches, data structures).
    - ReentrantLock: Suitable for **general-purpose locking** where fairness and reentrancy are needed.

Q122. Explain the term "Busy Spin" in multi-threading.

Busy Spin in Multi-Threading

Busy Spin is a technique where a thread continuously loops (spins) while waiting for a condition to be met, instead of sleeping or blocking. It helps reduce context switching overhead but wastes CPU cycles.

Example:

```
while (!conditionMet) {
    // Busy-wait (spinning)
}
```

When to Use?

✅ Low-latency systems (e.g., high-frequency trading) where thread wake-up delays are costly.
✅ Short wait times where context switching overhead is higher than spinning cost.

Drawbacks:

❌ High CPU usage – Can degrade system performance.
❌ Inefficient for long waits – Better to use LockSupport.park() or blocking mechanisms.

Q123. How can you implement a custom ClassLoader?

Create a subclass of ClassLoader and override methods like findClass(), loadClass(), and defineClass() to customize class loading behavior.

Q124. Describe Java JIT Compiler optimizations like Loop Unrolling and Vectorization.

The Java Just-In-Time (JIT) Compiler applies various optimizations to improve runtime performance:

- Loop Unrolling: Reduces the overhead of loop control (e.g., incrementing indices, checking conditions) by increasing the number of operations in the loop body, and reducing the number of iterations.
- Vectorization: Converts scalar operations into vector operations, allowing multiple data elements to be processed simultaneously using SIMD (Single Instruction, Multiple Data) instructions, significantly improving performance on modern CPUs.

Q125. Explain the use of the jstack tool.

jstack is a Java utility that displays the stack traces of Java threads in a running JVM. It is useful for debugging, diagnosing deadlocks, and analyzing performance bottlenecks.

Q126. Describe Java's Project Loom and its impact on concurrency.

Java's Project Loom & Its Impact on Concurrency

Project Loom (introduced in Java 21) enhances concurrency by introducing **virtual threads**, which are lightweight, user-mode threads managed by the JVM.

Impact on Concurrency:

1. **Massive Scalability** – Supports millions of virtual threads vs. limited OS threads.
2. **Simplifies Concurrency** – Replaces complex ThreadPool management with lightweight threading.
3. **Efficient Resource Utilization** – Uses fewer system resources, reducing context switching overhead.
4. **Better Performance for I/O Tasks** – Ideal for high-throughput, I/O-bound applications like web servers.
5. **Easy Migration** – Works seamlessly with existing Java threading APIs.

Q127. What is a Java Decompiler, and how can it be used securely?

Java Decompiler & Secure Usage

A Java Decompiler converts compiled .class files back into readable Java source code, helping in debugging, reverse engineering, and learning from third-party libraries.

Secure Usage of Java Decompilers:

1. Avoid Decompiling Proprietary Code – Respect copyright and licensing laws.
2. Use for Debugging & Recovery – Retrieve lost source code from compiled classes.
3. Analyze Security Risks – Identify vulnerabilities in compiled applications.
4. Prevent Reverse Engineering – Use obfuscation tools (e.g., ProGuard) to protect code.
5. Trusted Decompilers – Use reliable tools like JD-GUI, CFR, or Fernflower to avoid malware risks.

Q128. What are the challenges in implementing Distributed Garbage Collection (DGC)?

DGC is complex because it must manage memory across multiple JVMs. Challenges include:

- Tracking object references across distributed systems
- Handling network latency and failures
- Avoiding memory leaks and orphaned objects
- Minimizing performance overhead

Q129. Describe the impact of JVM flags on performance tuning.

Impact of JVM Flags on Performance Tuning

JVM flags optimize **memory, garbage collection (GC), and execution speed**, directly affecting application performance.

1. **Heap Size Management**
   - -Xms<size> & -Xmx<size>: Control **initial & max heap size**, preventing excessive GC.
2. **Garbage Collection (GC) Optimization**
   - -XX:+UseG1GC, -XX:+UseZGC: Tune GC for **low-latency or high-throughput** apps.
3. **JIT Compilation**
   - -XX:+TieredCompilation: Speeds up execution by **optimizing method compilation**.
4. **Thread & CPU Usage**

    - -XX:ActiveProcessorCount=<N>: Controls CPU usage for **better multi-threading**.
5. **Logging & Debugging**
    - -XX:+PrintGCDetails: **Monitors GC activity**, aiding performance tuning.

Q130. What are Java Modules, and how do they enhance security and maintainability?

Java Modules (Introduced in Java 9)

Java Modules (module-info.java) help organize code into self-contained, reusable units, improving security, maintainability, and performance.

How Modules Enhance Security & Maintainability?

1. **Encapsulation & Restricted Access**
    - Unlike traditional public classes, **only explicitly exported packages** are accessible, preventing unintended usage.
2. **Explicit Dependencies**
    - require keywords to ensure a module **only uses necessary dependencies**, avoiding **classpath conflicts**.
3. **Improved Maintainability**
    - Large applications can be split into **small, independent modules**, making it **easier to manage and scale**.
4. **Reduced Attack Surface**
    - Since internal classes are **not exposed** by default, security vulnerabilities due to unintended access are minimized.
5. **Faster Startup & Performance**
    - **JVM loads only required modules**, improving startup time and reducing memory usage.

Q131. Explain the concept of Java 8 parallelStream().

Java 8 `parallelStream()` Concept

The parallelStream() method in Java 8 allows **parallel processing** of collections using the **ForkJoin framework**, improving performance on multi-core processors.

Key Features:

1. **Splits data into multiple sub-tasks** and processes them in parallel.
2. **Uses ForkJoinPool internally** for task execution.
3. **Ideal for large data sets** where parallelism improves efficiency.
4. **Order of execution is not guaranteed**, unlike stream()

When to Use parallelStream()?

✅ CPU-intensive tasks (e.g., large computations, data processing).
✅ Processing large collections for better performance.
✅ Multi-core processors where parallel execution speeds up work.

When to Avoid?

❌ Small datasets (overhead may reduce performance).
❌ Dependent tasks (race conditions can occur).
❌ Modifying shared resources (risk of inconsistent state).

Q132. What is a deadlock in Java?

A deadlock occurs when two or more threads are indefinitely blocked, each waiting for a resource held by another.

Example:

- Thread A locks Resource 1 and waits for Resource 2
- Thread B locks Resource 2 and waits for Resource 1
  Proper synchronization strategies, such as lock ordering or timeouts, help prevent deadlocks.

Q133. What is the difference between Executor and ExecutorService?

Difference Between Executor and ExecutorService in Java

1. **Definition**
   - Executor: A simple interface for executing tasks asynchronously.
   - ExecutorService: A more advanced interface that extends Executor and provides additional control over thread execution.
2. **Functionality**

    - Executor: Only has **one method**: execute(Runnable command), which runs a task asynchronously.
    - ExecutorService: Supports **more methods** like submit(), invokeAll(), shutdown(), and awaitTermination().
3. **Task Handling**
    - Executor: Just fires and forgets tasks without managing their completion.
    - ExecutorService: Returns Future<T> objects, allowing **task tracking and results retrieval**.
4. **Shutdown Control**
    - Executor: Does **not** provide shutdown control.
    - ExecutorService: Provides methods like shutdown(), and shutdownNow() to **gracefully stop** tasks.

Q134. What is the difference between HashMap and Hashtable?

Feature	HashMap	Hashtable
Synchronization	Not synchronized(not thread-safe)	Synchronized(thread-safe)
Null keys/values	Allows one null key and multiple null values.	Does not allow null keys or values.
Performance	Faster	Slower due to synchronization

Q135. Explain Java 8 default methods in interfaces.

Java 8 introduced default methods in interfaces using the default keyword, allowing interfaces to have concrete method implementations without breaking existing classes.

Key Features:

1. Provides method implementation inside interfaces using default.
2. Maintains backward compatibility without forcing all implementing classes to override.
3. Supports multiple inheritance of behavior, but avoids diamond problem with explicit method resolution.

Q136. What is the difference between wait() and notify()?

Difference Between wait() and notify() in Java

1. **Purpose**
   - wait(): Causes the current thread to **release the lock** and enter a waiting state.
   - notify(): Wakes up **one** waiting thread from the waiting state.
2. **Usage**
   - wait(): Called when a thread **needs to wait** for a condition.
   - notify(): Called when a thread **wants to signal** a waiting thread to resume execution.
3. **Lock Requirement**
   - Both wait() and notify() must be called inside a **synchronized block** on the same object monitor.
4. **Thread Behavior**
   - wait(): The thread **releases the lock** and goes to the waiting queue.
   - notify(): The awakened thread must **reacquire the lock** before proceeding.

Q137. What is the difference between Comparable and Comparator?

Comparable vs. Comparator in Java

1. **Purpose**
   - Comparable: Used for **natural ordering** of objects (e.g., sorting by ID).
   - Comparator: Used for **custom ordering** (e.g., sorting by name, age, etc.).
2. **Implementation**
   - Comparable: Implemented by the class itself (implements Comparable<T>).

    - Comparator: Implemented as a separate class (implements Comparator<T>).
3. **Method Used**
    - Comparable: Overrides compareTo(T o).
    - Comparator: Overrides compare(T o1, T o2).
4. **Usage**
    - Comparable: Used when a **single sorting order** is needed.
    - Comparator: Used when **multiple sorting criteria** are required.

Q138. What is the transient keyword in Java?

Transient prevents a field from being serialized, meaning it will not be saved when an object is written to a file or transmitted over a network.

Q139. What is the purpose of the instanceof operator?

instanceof checks if an object is an instance of a specific class or subclass.

Example:

```
if (object instanceof MyClass) {
    // Do something
}
```

Q140. Explain the concept of lock-free programming in Java.

Lock-free programming in Java ensures thread-safe concurrency without using traditional locks (synchronized, ReentrantLock). Instead, it relies on atomic operations and Compare-And-Swap (CAS) to manage shared resources efficiently.

How It Works:

1. **Uses Atomic Classes** – Java provides AtomicInteger, AtomicLong, AtomicReference, etc., from java.util.concurrent.atomic.
2. **Relies on CAS (Compare-And-Swap)** – Instead of locking, CAS updates a value only if it hasn't changed, ensuring consistency.
3. **Avoids Blocking & Deadlocks** – Threads keep retrying operations until they succeed, improving performance.

4. **Optimized by Hardware** – CAS operations leverage CPU-level instructions for fast, low-overhead execution.
5. **Used in Concurrent Data Structures** – ConcurrentHashMap, ConcurrentLinkedQueue, and AtomicStampedReference use lock-free techniques.

Q141. What is java.lang.reflect.Proxy used for?

Proxy dynamically creates objects implementing interfaces at runtime. It is widely used in AOP (Aspect-Oriented Programming) and frameworks like Spring.

Q142. What are some advanced features of CompletableFuture in Java?

- Chaining asynchronous tasks
- Combining multiple async computations
- Exception handling
- Timeouts and default values
- Custom executor control

Q143. What is ForkJoinPool, and how is it different from a regular thread pool?

ForkJoinPool (introduced in Java 7) is a specialized thread pool designed for parallel processing of recursive tasks using the work-stealing algorithm. It is part of the java.util.concurrent package.

ForkJoinPool vs. Regular Thread Pool

1. **Task Type**
    - ForkJoinPool: Designed for divide-and-conquer tasks (RecursiveTask, RecursiveAction).
    - Regular Thread Pool: Executes independent Runnable/Callable tasks.
2. **Work Stealing**

    - ForkJoinPool: Uses work-stealing (idle threads take tasks from busy threads).
    - Regular Thread Pool: No work-stealing; tasks are assigned to specific threads.
3. **Best Use Case**
    - ForkJoinPool: Ideal for CPU-intensive, recursive parallel processing (e.g., sorting, tree traversal).
    - Regular Thread Pool: Suitable for I/O tasks, web servers, and background tasks.
4. **Task Splitting & Execution**
    - ForkJoinPool: Splits large tasks into subtasks (fork() & join()).
    - Regular Thread Pool: No built-in task splitting, executes tasks independently.
5. **Performance Optimization**
    - ForkJoinPool: Optimized for parallel execution using all available CPU cores.
    - Regular Thread Pool: Works well for asynchronous task execution but lacks parallel optimization.

Q144. How does the Unsafe class work, and what are its use cases?

Unsafe Class in Java (sun.misc.Unsafe)

The Unsafe class provides low-level, unsafe operations such as direct memory access, off-heap allocation, and atomic operations. It bypasses standard Java safety checks, making it powerful but dangerous.

Key Features:

- Memory manipulation (allocateMemory(), freeMemory(), putInt(), etc.).
- Bypass constructor calls (allocateInstance() to create objects without calling a constructor).
- CAS (Compare-And-Swap) operations for atomic variables (used in java.util.concurrent).
- Thread operations (park(), unpark(), getAndSetInt(), etc.).
- Class loading manipulation (defineClass(), defineAnonymousClass()).

Use Cases:

- High-performance libraries (e.g., Netty, Cassandra).
- Off-heap memory storage (avoids GC overhead).

- Atomic operations (used in AtomicInteger, ConcurrentHashMap).
- Class and method hacking (modifying final fields, method handles).

Q145. What is PhantomReference in Java?

PhantomReference in Java

PhantomReference (from java.lang.ref) is a special type of reference that allows post-mortem cleanup after an object is garbage collected. Unlike SoftReference and WeakReference, a PhantomReference is always null when accessed via get().

Key Features:

- Used for resource cleanup (e.g., deallocating native memory).
- Not automatically removed by GC; it is added to a ReferenceQueue instead.
- Ensures proper cleanup before memory is reclaimed.

Q146. Explain the concept of MethodHandle in Java.

MethodHandle in Java

MethodHandle (introduced in Java 7, java.lang.invoke) is a lightweight, faster alternative to reflection for dynamically invoking methods, constructors, and fields. It offers better performance and type safety compared to java.lang.reflect.Method.

Key Points:

- More efficient than reflection (Method).
- Supports dynamic invocation of private, public, static, and instance methods.
- Type-safe (checked at compile time).
- Used inter**nally in lambdas & `invokedynamic` for JVM optimizations.**

When to Use?

- When better performance than reflection is needed.
- For dynamic method invocation (e.g., lambda metafactories).

- In JVM internals and advanced frameworks.

Q147. What is the role of java.lang.instrument?

The java.lang.instrument package enables bytecode modification at runtime, used in Java agents for profiling, monitoring, and aspect-oriented programming.

Q148. How does the Java Memory Model handle out-of-order execution and memory visibility?

- Defines happens-before relationships for correct memory visibility
- Uses volatile and synchronized to enforce ordering
- Prevents compiler and CPU optimizations that may break concurrency semantics

---

Q149. What is the difference between ReentrantLock and synchronized?

1. Synchronized is a built-in Java monitor lock, while ReentrantLock is part of java.util.concurrent.locks with more flexibility.
2. Synchronized acquires and releases the lock automatically, whereas ReentrantLock requires explicit lock() and unlock().
3. synchronized does not support try-locking, but ReentrantLock allows tryLock() and tryLock(timeout, TimeUnit).
4. Synchronized is unfair (no guaranteed thread order), whereas ReentrantLock can be fair (new ReentrantLock(true)) or unfair.
5. Synchronized does not allow lock interruption, but ReentrantLock supports lockInterruptibly() for better thread management.
6. Synchronized uses wait() and notify(), while ReentrantLock provides Condition variables for finer control over waiting threads.
7. Synchronized is simpler and easier to use, while ReentrantLock is more powerful but requires careful handling to avoid deadlocks.

Q150. What is the difference between WeakReference and SoftReference?

Both WeakReference and SoftReference are part of Java's reference types in the java.lang.ref package, used to manage memory more efficiently by allowing objects to be garbage collected under specific conditions. Here's the key difference between them:

Garbage Collection Timing

- WeakReference: Reclaimed immediately when no strong references exist.
- SoftReference: Reclaimed only under memory pressure.

Retention Policy

- WeakReference: Short-lived, discarded aggressively.
- SoftReference: Longer-lived, retained until JVM needs memory.

Use Case

- WeakReference: Used for lookup tables, avoiding memory leaks (e.g., caches, listeners).
- SoftReference: Used for caching data that is expensive to reload (e.g., image caching, object pools).

Garbage Collector Behavior

- WeakReference: GC clears weak references at the earliest opportunity.
- SoftReference: GC clears soft references only when memory is low.

Likelihood of Being Collected

- WeakReference: Very high (almost immediate collection).
- SoftReference: Lower (kept until necessary to free memory)