Core Java Interviews questions and answers

Q1. Can we define static variables inside the static method or not?

Answer: No, static variables cannot be defined inside a static method. Static variables are associated with the class itself rather than instances of the class, and must be declared at the class level.

Example:

```
public class Example {
    static int classVar = 10; // Static variable

    public static void staticMethod() {
        // int localVar = 20; // Local variable (cannot be static)
    }
}
```

Q2. In project when we will instance & static variables give the scenario?

Answer: Use instance variables when each object of the class needs its own copy of a variable. Use static variables when the variable is shared across all instances of the class.

Example:

```
public class Counter {
    private int instanceCount = 0; // Instance variable
```

```
    private static int staticCount = 0; // Static variable

    public void increment() {
        instanceCount++;
        staticCount++;
    }

    public int getInstanceCount() {
        return instanceCount;
    }

    public static int getStaticCount() {
        return staticCount;
    }
}
```

Q3. When we will use for vs. for-each?

Answer: Use a `for` loop when you need to iterate over a range of values or need an index. Use the `for-each` loop when you need to iterate over elements of a collection or array without needing to modify the elements.

Example:

```
// Using for loop
for (int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
}
```

```
// Using for-each loop
for (int element : array) {
    System.out.println(element);
}
```

Q4. Define JDK vs. JRE vs. JVM?

Answer:

- JVM (Java Virtual Machine): It is the engine that provides the runtime environment to execute Java bytecode. It is platform-dependent but allows Java programs to be platform-independent.

- JRE (Java Runtime Environment): It includes the JVM along with libraries and other components needed to run Java applications. It does not include development tools.

- JDK (Java Development Kit): It includes the JRE and additional tools needed for Java development, such as the compiler (javac), debugger, and documentation generator.

Example:

- JVM runs Java programs.

- JRE provides the environment to run Java programs.

- JDK provides tools to develop Java programs.

Q5. How many parts of Java?

Answer: Java is generally divided into three main parts:

1. Java SE (Standard Edition): Core Java platform including basic libraries and APIs.

2. Java EE (Enterprise Edition): Provides additional libraries for enterprise-level applications, including Servlets, JSPs, and EJBs.

3. Java ME (Micro Edition): Designed for embedded and mobile devices with a subset of Java SE functionalities.

Q6. What is the purpose of methods?

Answer: Methods are used to perform operations, execute code, and manage data. They help to organize code into reusable blocks, enhancing modularity and readability.

Example:

```
public class Example {
    public void greet() {
        System.out.println("Hello, World!");
    }
}
```

Q7. Define class vs. Object?

Answer:

- Class: A blueprint for creating objects, defining properties and behaviors.
- Object: An instance of a class that contains data and methods to operate on the data.

Example:

```
public class Car {
```

```
    String color;
    void drive() {
        System.out.println("Driving");
    }
}

Car myCar = new Car(); // Object
myCar.color = "Red";
myCar.drive();
```

Q8. Define and (&), or (|)?

Answer:

- & (Bitwise AND): Performs a bitwise AND operation. It operates on individual bits of integers.
- | (Bitwise OR): Performs a bitwise OR operation. It also operates on individual bits.

Example:

```
int a = 5; // 0101 in binary
int b = 3; // 0011 in binary

int andResult = a & b; // 0001 in binary (1 in decimal)
int orResult = a | b;  // 0111 in binary (7 in decimal)
```

Q9. Define the constructor?

Answer: A constructor is a special method that is called when an object is instantiated. It initializes the newly created object.

Example:

```
public class Example {
    int value;

    public Example(int value) {
        this.value = value;
    }
}
```

Q10. Define break vs. continue?

Answer:

- Break: Exits from the nearest enclosing loop or switch statement.

- Continue: Skips the current iteration of the nearest enclosing loop and proceeds to the next iteration.

Example:

```
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        break; // Exit loop when i is 2
```

```
    }
    System.out.println(i);
}

for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue; // Skip printing when i is 2
    }
    System.out.println(i);
}
```

Q11. Define the package in Java?

Answer: A package is a namespace that organizes classes and interfaces into a folder structure, providing modularity and preventing name conflicts.

Example:

```
package com.example;

public class Example {
    // Class code here
}
```

Q12. What is the difference between bitwise & logical operator?

Answer:

- Bitwise Operators: Operate on bits and perform bit-level operations.

- Logical Operators: Operate on boolean values and perform logical operations.

Example:

```
// Bitwise AND
int a = 5; // 0101 in binary
int b = 3; // 0011 in binary
int result = a & b; // 0001 in binary (1 in decimal)

// Logical AND
boolean x = true;
boolean y = false;
boolean result = x && y; // false
```

Q13. What are the entry controlled loop & exit controlled loop?

Answer:

- Entry-Controlled Loop: The condition is checked before the loop body is executed (e.g., `for`, `while` loops).

- Exit-Controlled Loop: The condition is checked after the loop body has executed at least once (e.g., `do-while` loop).

Example:

```
// Entry-Controlled Loop
for (int i = 0; i < 5; i++) {
   System.out.println(i);
}

// Exit-Controlled Loop
int i = 0;
do {
   System.out.println(i);
   i++;
} while (i < 5);
```

Q14. When will we get StackOverflowError?

Answer: `StackOverflowError` occurs when a program recurses too deeply, causing the stack memory to overflow.

Example:

```
public class Example {
   public void recursiveMethod() {
      recursiveMethod(); // Infinite recursion
   }

   public static void main(String[] args) {
      new Example().recursiveMethod();
```

```
    }
}
```

Q15. Explain instance & static blocks? What is the purpose of blocks?

Answer:

- Instance Block: Executes when an instance of the class is created. Used for instance initialization.
- Static Block: Executes when the class is loaded. Used for static initialization.

Example:

```
public class Example {
    static {
        System.out.println("Static block executed");
    }

    {
        System.out.println("Instance block executed");
    }

    public Example() {
        System.out.println("Constructor executed");
    }

    public static void main(String[] args) {
        new Example();
    }
```

}

Q16. What are the different ways to load the .class file into memory?

Answer: The `.class` file can be loaded into memory via:

- Class.forName()
- ClassLoader.loadClass()
- Java Virtual Machine (JVM) automatically loading classes

Example:

```
Class<?> clazz = Class.forName("com.example.Example");
```

Q17. How to call the constructor in Java? Is it one constructor can call multiple constructors?

Answer: Constructors are called when creating an object using the `new` keyword. A constructor can call other constructors within the same class using `this()`.

Example:

```
public class Example {
    public Example() {
        this(10); // Calling another constructor
```

```
    }

    public Example(int value) {
        System.out.println("Value: " + value);
    }
}
```

Q18. What is method recursion?

Answer: Method recursion occurs when a method calls itself to solve a problem in smaller chunks, typically used for problems that can be divided into similar sub-problems.

Example:

```
public class Example {
    public int factorial(int n) {
        if (n == 0) {
            return 1;


 }
        return n * factorial(n - 1);
    }
}
```

Q19. What are the different ways to call the static members in Java?

Answer: Static members can be accessed via:

- Class Name: `ClassName.staticMethod()`
- Object Reference: `object.staticMethod()` (not recommended)

Example:

```
public class Example {
    public static void staticMethod() {
        System.out.println("Static method");
    }
}

Example.staticMethod();
```

Q20. When will we use switch & else-if statements in Application?

Answer:

- Switch: When dealing with a single variable that can take discrete values (e.g., enum, constant values).
- Else-if: When dealing with multiple conditions or ranges of values.

Example:

```
// Switch
switch (day) {
   case 1: System.out.println("Monday"); break;
   case 2: System.out.println("Tuesday"); break;
   default: System.out.println("Other");
}

// Else-if
if (day == 1) {
   System.out.println("Monday");
} else if (day == 2) {
   System.out.println("Tuesday");
} else {
   System.out.println("Other");
}
```

Q21. When will we use for vs. while loops in Application?

Answer:

- For Loop: When the number of iterations is known beforehand.
- While Loop: When the number of iterations is not known or depends on a condition.

Example:

```
// For Loop
for (int i = 0; i < 5; i++) {
   System.out.println(i);
```

```
}

// While Loop
int i = 0;
while (i < 5) {
   System.out.println(i);
   i++;
}
```

Q22. Can we access the other packages classes without import statements?

Answer: No, you need import statements to access classes from other packages, unless they are in the same package or are part of the `java.lang` package which is imported by default.

Example:

```
import com.example.OtherClass;

public class Example {
   OtherClass obj = new OtherClass();
}
```

Q23. What is the difference between normal import & static import?

Answer:

- Normal Import: Imports classes or interfaces.

- Static Import: Imports static members of a class.

Example:

```
// Normal Import
import java.util.ArrayList;

// Static Import
import static java.lang.Math.PI;
```

Q24. What are the modifiers applicable to constructors?

Answer: Constructors can have access modifiers such as `public`, `private`, `protected`, and default access. They cannot be `static`, `final`, or `abstract`.

Example:

```
public class Example {
    public Example() { }
    private Example(int value) { }
}
```

Q25. Explain System.out.println()?

Answer: `System.out.println()` is a method in Java used to print text to the console. It appends a new line after printing the text.

Example:

```
System.out.println("Hello, World!");
```

Q26. Can we access sub-packages data when we import the main package with *?

Answer: No, importing a package with `*` only imports the classes in that package, not sub-packages.

Example:

```
import com.example.*; // Imports classes in com.example but not in sub-packages
```

Q27. What are the Permission/Scoping modifiers in Java?

Answer: The permission modifiers are:

- public: Accessible from anywhere.
- protected: Accessible within the same package and subclasses.

- default: Accessible within the same package (no modifier).
- private: Accessible only within the same class.

Example:

```
public class Example {
    public int publicVar;
    protected int protectedVar;
    int defaultVar; // package-private
    private int privateVar;
}
```

Q28. Define inheritance? How many types of inheritance in Java?

Answer: Inheritance is a mechanism where one class (subclass) acquires the properties and behaviors of another class (superclass). Types of inheritance in Java:

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Interface Inheritance (via interfaces)

Example:

```
// Single Inheritance
class Parent { }
class Child extends Parent { }
```

Q29. Explain about Object class in Java?

Answer: The `Object` class is the root class of all classes in Java. It provides basic methods like `toString()`, `equals()`, and `hashCode()`.

Example:

```
public class Example {
    public static void main(String[] args) {
        Object obj = new Object();
        System.out.println(obj.toString());
    }
}
```

Q30. Define Polymorphism? How many types of Polymorphisms in Java?

Answer: Polymorphism allows objects to be treated as instances of their parent class rather than their actual class. Types of polymorphism:

- Compile-Time Polymorphism (Method Overloading)
- Runtime Polymorphism (Method Overriding)

Example:

```
class Parent {
    void display() { System.out.println("Parent"); }
}

class Child extends Parent {
    @Override
    void display() { System.out.println("Child"); }
}

public class Example {
    public static void main(String[] args) {
        Parent obj = new Child(); // Runtime Polymorphism
        obj.display(); // Calls Child's display()
    }
}
```

Q31. What are the different types of overloading in Java? What is the advantage?

Answer: Overloading involves defining multiple methods with the same name but different parameters. Types include:

- Constructor Overloading
- Method Overloading

Advantage: Increases code readability and reusability.

Example:

```
class Example {
   void show(int a) { }
   void show(double a) { }
   void show(int a, double b) { }
}
```

Q32. What is method Overriding? What is the advantage?

Answer: Method overriding occurs when a subclass provides a specific implementation for a method already defined in its superclass.

Advantage: Allows a subclass to customize or extend the behavior of methods inherited from a superclass.

Example:

```
class Parent {
   void display() { System.out.println("Parent"); }
}

class Child extends Parent {
   @Override
   void display() { System.out.println("Child"); }
}
```

Q33. What are the rules to follow while overriding the method?

Answer:

- The method name, return type, and parameters must be the same as in the superclass.
- The overriding method cannot have a more restrictive access modifier.
- It can throw fewer or the same exceptions as the overridden method.

Example:

```
class Parent {
    protected void display() { }
}

class Child extends Parent {
    @Override
    public void display() { } // Allowed
}
```

Q34. What is Co-variant return type in Java?

Answer: Co-variant return type allows a method in a subclass to return a type that is a subclass of the type returned by the overridden method in the superclass.

Example:

```
class Parent {
```

```
    Parent get() { return this; }
}

class Child extends Parent {
    @Override
    Child get() { return this; } // Co-variant return type
}
```

Q35. What is Method overriding & method hiding in Java?

Answer:

- Method Overriding: A subclass provides a specific implementation of a method already defined in its superclass.

- Method Hiding: Occurs with static methods; a subclass defines a static method with the same name as in the superclass.

Example:

```
class Parent {
    static void staticMethod() { System.out.println("Static Parent"); }
    void instanceMethod() { System.out.println("Instance Parent"); }
}

class Child extends Parent {
    static void staticMethod() { System.out.println("Static Child"); } // Method hiding
    @Override
    void instanceMethod() { System.out.println("Instance Child"); } // Method overriding
}
```

Q36. What is the advantage of parent class reference holding the child class object? Or What is runtime polymorphism?

Answer: Allows for dynamic method binding, where the method that gets executed is determined at runtime based on the object type. It provides flexibility and extensibility.

Example:

```
Parent obj = new Child(); // Parent class reference holding Child class object
obj.display(); // Calls Child's display() method
```

Q37. Explain about final modifier/prevention modifier in Java?

Answer:

- `final` Variable: Constant value that cannot be changed once assigned.
- `final` Method: Cannot be overridden by subclasses.
- `final` Class: Cannot be subclassed.

Example:

```
final class Example { }
```

Q38. Define abstract methods & normal methods in Java?

Answer:

- Abstract Method: Declared with `abstract` keyword, no implementation provided. Must be implemented by subclasses.

- Normal Method: Has a body with implementation.

Example:

```
abstract class AbstractClass {
   abstract void abstractMethod(); // Abstract method
   void normal

Method() { } // Normal method
}
```

Q39. Explain normal classes & abstract classes in Java?

Answer:

- Normal Class: Can be instantiated and may contain methods with or without implementations.

- Abstract Class: Cannot be instantiated, may contain abstract methods and concrete methods.

Example:

```
class NormalClass {
    void normalMethod() { }
}

abstract class AbstractClass {
    abstract void abstractMethod();
}
```

Q40. What is the abstraction concept in Java? What is the advantage of abstraction?

Answer: Abstraction is the concept of hiding implementation details and showing only functionality. It is achieved using abstract classes and interfaces. It helps in reducing complexity and increasing efficiency.

Example:

```
abstract class Shape {
    abstract void draw(); // Abstract method
}

class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing Circle");
    }
}
```

Q41. Explain about interfaces in Java?

Answer: An interface is a reference type in Java that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields or constructors.

Example:

```
interface Drawable {
   void draw(); // Abstract method
}
```

Q42. What is the difference between interfaces & abstract classes & normal classes & client code?

Answer:

- Interface: Defines a contract with abstract methods.
- Abstract Class: Can have both abstract and concrete methods.
- Normal Class: Can be instantiated and contain concrete methods.

Client Code: Code that uses classes and interfaces.

Example:

```
interface Drawable { void draw(); }
abstract class Shape implements Drawable { }
```

```
class Circle extends Shape {
    @Override
    public void draw() { System.out.println("Drawing Circle"); }
}
```

Q43. How to clone objects in Java? What is the advantage?

Answer: Objects can be cloned using the `clone()` method from the `Cloneable` interface.

Advantage: Allows for creating a copy of an object with the same state.

Example:

```
class Example implements Cloneable {
    int value;

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

Q44. What are the new features about interfaces in Java?

Answer: New features include:

- Default Methods: Methods with a default implementation.

- Static Methods: Methods that can be called on the interface itself.

- Private Methods: Methods that can be used within the interface.

Example:

```
interface Example {
    default void defaultMethod() { }
    static void staticMethod() { }
    private void privateMethod() { }
}
```

Q45. Explain about functional interfaces in Java?

Answer: A functional interface has exactly one abstract method and can have multiple default or static methods. It is used as the target for lambda expressions.

Example:

```
@FunctionalInterface
interface MyFunctionalInterface {
    void abstractMethod(); // Single abstract method
}
```

Q46. What is the meaning of Automatic garbage collection?

Answer: Automatic garbage collection refers to the JVM's ability to automatically reclaim memory by deleting objects that are no longer in use, freeing up resources without explicit intervention by the programmer.

Q47. What are the different ways to call the Garbage Collector?

Answer: The garbage collector can be requested using:

- System.gc(): Suggests that the JVM performs garbage collection.
- Runtime.getRuntime().gc(): Similar to `System.gc()`.

Example:

```
System.gc();
```

Q48. What are the different ways to make the object un-referenced?

Answer: To make an object un-referenced:

- Set the reference variable to null
- Reassign the reference variable to another object

Example:

```
Example obj = new Example();
obj = null; // Object is now un-referenced
```

Q49. Explain Encapsulation mechanism in Java?

Answer: Encapsulation is the concept of wrapping data (variables) and methods into a single unit (class) and restricting access to some of the object's components. It is achieved using access modifiers.

Example:

```
public class Example {
    private int value; // Private variable

    public void setValue(int value) { this.value = value; }
    public int getValue() { return value; }
}
```

Q50. What are the different ways to initialize the data in Java? Explain.

Answer:

- Initialization at Declaration: Assigning values directly during declaration.
- Initialization in Constructor: Using constructor initialization.
- Initialization Blocks: Using instance and static blocks.

Example:

```
public class Example {
   int value = 10; // Initialization at declaration

   public Example() {
      this.value = 20; // Initialization in constructor
   }
}
```

Q51. Define type conversion? What are the different types of type conversions in Java?

Answer: Type conversion is the process of converting one data type into another. Types include:

- Implicit Conversion (Widening): Automatic conversion by the compiler.
- Explicit Conversion (Narrowing): Manual conversion using casting.

Example:

```
int i = 10;
double d = i; // Implicit conversion (widening)

double d = 10.5;
int i = (int) d; // Explicit conversion (narrowing)
```

Q52. Why is the method signature of the main() method always public static void main(String[] args)?

Answer: The `main()` method must be `public` to be accessible from outside the class, `static` so that it can be called without creating an instance of the class, `void` because it does not return a value, and it must accept a `String[]` parameter to handle command-line arguments.

Q53. Though an interface is a pure abstract class, why are interfaces needed?

Answer: Interfaces allow for multiple inheritance and provide a way to achieve abstraction and polymorphism. They also define a contract that implementing classes must follow.

Q54. Define the exception? What is the main objective of exception handling?

Answer: An exception is an event that disrupts the normal flow of execution. Exception handling provides a way to manage runtime errors, allowing the program to continue executing or to fail gracefully.

Q55. What are the types of exceptions in Java?

Answer:

- Checked Exceptions: Must be either caught or declared in the method signature (e.g., IOException).

- Unchecked Exceptions: Do not need to be explicitly handled (e.g., NullPointerException, ArithmeticException).

Q56. What are the different ways to handle exceptions in Java?

Answer:

- Try-Catch Block: Catches and handles exceptions.

- Try-Catch-Finally Block: Ensures code in the `finally` block is executed regardless of whether an exception occurs.

- Throw Statement: Explicitly throws an exception.

- Throws Keyword: Declares exceptions that a method may throw.

Example:

```
try {
   // Code that may throw an exception
} catch (Exception e) {
   // Code to handle the exception
} finally {
   // Code to execute regardless of exception
}
```

Q57. What are the possible ways to handle multiple exceptions in Java?

Answer:

- Multiple Catch Blocks: Handling each exception type separately.

- Multi-catch Block: Handling multiple exception types in a single catch block.

Example:

```
try {
   // Code that may throw exceptions
} catch (IOException | SQLException e) {
   // Handle both IOException and SQLException
}
```

Q58. What is the purpose of try-with-resources concept?

Answer: Try-with-resources ensures that resources are automatically closed after the try block completes, reducing the risk of resource leaks.

Example:

```
try (BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {
   // Use the resource
} catch (IOException e) {
   e.printStackTrace();
}
```

Q59. What is the purpose of the finally block?

Answer: The `finally` block is used to execute code that must run regardless of whether an exception was thrown or not, such as closing resources.

Q60. What is the purpose of the throw keyword? How to handle user-defined exceptions in Java?

Answer: The `throw` keyword is used to explicitly throw an exception. User-defined exceptions are handled by creating custom exception classes that extend `Exception` or `RuntimeException`.

Example:

```
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}

public class Example {
    public void method() throws CustomException {
        throw new CustomException("Custom Exception");
    }
}
```

Q81. Explain Daemon & non-Daemon threads in Java?

Answer:

- Daemon Threads: Background threads that do not prevent the JVM from exiting. They are used for tasks such as garbage collection.

- Non-Daemon Threads: User threads that keep the JVM running until they complete.

Example:

```
Thread daemonThread = new Thread();
daemonThread.setDaemon(true); // Daemon thread
daemonThread.start();
```

Q82. Explain thread names & priority in Java?

Answer:

- Thread Names: Used

for identifying threads.

- Priority: Determines the order in which threads are scheduled. Ranges from `Thread.MIN_PRIORITY` (1) to `Thread.MAX_PRIORITY` (10).

Example:

```
Thread thread = new Thread();
thread.setName("MyThread");
thread.setPriority(Thread.MAX_PRIORITY);
thread.start();
```

Q83. Explain Thread Scheduling in Java?

Answer: Thread scheduling is managed by the JVM and OS. Threads are scheduled based on priority, time slicing, and various scheduling algorithms.

Q84. Explain synchronization in Java. Why is it used?

Answer: Synchronization is used to control access to shared resources by multiple threads, preventing data inconsistency and race conditions.

Example:

```
synchronized (this) {
    // Critical section
}
```

Q85. What is the difference between synchronized method & synchronized block in Java?

Answer:

- Synchronized Method: Locks the entire method.
- Synchronized Block: Locks only a specific block of code, providing more granular control.

Example:

```
// Synchronized Method
```

```
public synchronized void method() { }

// Synchronized Block
public void method() {
   synchronized (this) {
      // Critical section
   }
}
```

Q86. What are the states of a thread in Java?

Answer:

- New: Thread is created but not started.
- Runnable: Thread is ready to run and waiting for CPU time.
- Blocked: Thread is blocked waiting for a monitor lock.
- Waiting: Thread is waiting indefinitely for another thread to perform a particular action.
- Timed Waiting: Thread is waiting for a specified period.
- Terminated: Thread has exited.

Q87. What is the difference between wait() & sleep() methods in Java?

Answer:

- wait(): Used in synchronization to wait until notified. Must be called from within a synchronized block.
- sleep(): Pauses the thread for a specified period without releasing locks.

Example:

```
synchronized (this) {
   wait(); // Waits until notified
}
```

```
Thread.sleep(1000); // Pauses for 1 second
```

Q88. What is the use of join() method in Java?

Answer: The `join()` method allows one thread to wait for the completion of another thread.

Example:

```
Thread thread = new Thread();
thread.start();
thread.join(); // Main thread waits for thread to complete
```

Q89. What is the use of yield() method in Java?

Answer: The `yield()` method suggests to the thread scheduler that the current thread is willing to yield its current use of the CPU.

Example:

```
Thread.yield(); // Suggests to the thread scheduler to give other threads a chance
```

Q90. What is the difference between the Runnable interface and the Thread class in Java?

Answer:

- Runnable Interface: Provides a `run()` method to be executed by a thread. It allows you to separate the task from the thread.

- Thread Class: Represents a thread and contains methods to manage the thread's lifecycle.

Example:

```
// Runnable
class MyRunnable implements Runnable {
    public void run() { }
}

Thread thread = new Thread(new MyRunnable());
thread.start();

// Thread Class
class MyThread extends Thread {
    public void run() { }
}
```

```
MyThread thread = new MyThread();
thread.start();
```

Q91. What are the different ways to create threads in Java?

Answer:

- Extend Thread Class: Subclass `Thread` and override `run()` method.
- Implement Runnable Interface: Implement `Runnable` interface and pass it to a `Thread` object.

Example:

```
// Extend Thread Class
class MyThread extends Thread {
    public void run() { }
}

MyThread thread = new MyThread();
thread.start();

// Implement Runnable Interface
class MyRunnable implements Runnable {
    public void run() { }
}

Thread thread = new Thread(new MyRunnable());
thread.start();
```

Q92. Explain Concurrent Collections in Java?

Answer: Concurrent collections are designed for safe usage by multiple threads concurrently, without needing explicit synchronization.

Example:

```
ConcurrentHashMap<String, String> map = new ConcurrentHashMap<>();
```

Q93. What is the use of Callable interface? How it is different from Runnable interface?

Answer: The `Callable` interface is similar to `Runnable` but can return a result and throw checked exceptions. It is used with the `ExecutorService` to handle tasks that return a result.

Example:

```
Callable<Integer> callable = () -> {
   return 123;
};

Future<Integer> future = executorService.submit(callable);
Integer result = future.get();
```

Q94. Explain about Executor framework in Java?

Answer: The Executor framework provides a higher-level replacement for the traditional way of managing threads. It includes classes like `ThreadPoolExecutor` and `ScheduledThreadPoolExecutor` for managing and scheduling tasks.

Example:

```
ExecutorService executorService = Executors.newFixedThreadPool(10);
executorService.submit(() -> {
    // Task code
});
executorService.shutdown();
```

Q95. What is the difference between ExecutorService and ScheduledExecutorService?

Answer:

- ExecutorService: Manages the execution of tasks asynchronously.
- ScheduledExecutorService: Extends `ExecutorService` to provide scheduling capabilities.

Example:

```
ScheduledExecutorService scheduledExecutorService = Executors.newScheduledThreadPool(10);
scheduledExecutorService.schedule(() -> {
    // Task code
```

}, 1, TimeUnit.SECONDS);

Q96. Explain about the Fork/Join framework in Java?

Answer: The Fork/Join framework is designed for parallel processing of tasks by dividing a task into smaller subtasks, processing them in parallel, and then combining the results.

Example:

```
ForkJoinPool pool = new ForkJoinPool();
RecursiveTask<Integer> task = new RecursiveTask<>() {
   @Override
   protected Integer compute() {
      // Task code
   }
};
Integer result = pool.invoke(task);
```

Q97. Explain about CopyOnWriteArrayList & ConcurrentHashMap?

Answer:

- CopyOnWriteArrayList: A thread-safe variant of `ArrayList` where modifications are made by copying the underlying array.

- ConcurrentHashMap: A thread-safe variant of `HashMap` that supports concurrent access and modifications.

Example:

```
CopyOnWriteArrayList<String> list = new CopyOnWriteArrayList<>();
ConcurrentHashMap<String, String> map = new ConcurrentHashMap<>();
```

Q98. Explain about ReentrantLock in Java?

Answer: `ReentrantLock` is an explicit lock that allows threads to acquire a lock on a resource and provides more control over synchronization compared to `synchronized` blocks.

Example:

```
ReentrantLock lock = new ReentrantLock();
lock.lock();
try {
   // Critical section
} finally {
   lock.unlock();
}
```

Q99. What is the difference between synchronized keyword and ReentrantLock?

Answer:

- Synchronized Keyword: Implicitly locks a block or method, providing automatic lock release.

- ReentrantLock: Provides more flexibility and control, such as try-lock and timed lock features.

Example:

```
// Synchronized
synchronized (this) {
    // Critical section
}

// ReentrantLock
ReentrantLock lock = new ReentrantLock();
lock.lock();
try {
    // Critical section
} finally {
    lock.unlock();
}
```

Q100. What are the advantages of using Concurrent Collections?

Answer: Concurrent collections provide thread-safe operations, reducing the need for explicit synchronization and improving performance and scalability in concurrent applications.