MCQ Questions

1) A _abstraction __ represents the need of information in the program without the presenting the details.
   a) abstraction
   b) polymorphism
   c) implementation
   d) class

2) The process of forming a new class from an existing class. = Inheritance
   a) Abstraction
   b) Polymorphism
   c) Inheritance
   d) Implementation

3) The ability for programmers to use the same written and debugged existing class.=a) Reusability
   b) Design
   c) Debugging
   d) Implementation

4) A concept that combines data and functions into a single unit called class.
   = encapsulation a) inheritance
   b) encapsulation
   c) polymorphism
   d) abstraction

5) _object__ represents a particular instance of a class.
   a) module
   b) block
   c) object
   d) token

6) A basic unit of object-oriented programming is __object__
   a) module
   b) block
   c) object
   d) token

7) Object oriented programming approach follows _top-down _ approach.

a) top-down
b) bottom-up
c) left-right
d) right-left

8) Which of the following is not OOPS concept in Java? = Compilation
a) Inheritance
b) Encapsulation
c) Polymorphism
d) Compilation

9) Which concept of Java is a way of converting real world objects in terms of class? = Abstraction
a) Polymorphism
b) Encapsulation
c) Abstraction
d) Inheritance

10) Which of the following is the functionality of 'Data Abstraction'? = Reduce Complexity
a) Reduce Complexity
b) Binds together code and data
c) Parallelism
d) None of the mentioned

11) How will a class protect the members inside it? = Using Access Specifiers
a) Using Access specifiers
b) Abstraction
c) Use of Inheritance
d) All of the mentioned

12) Which keyword from the following is used to inherit properties from one class into another? = a) extends
a) extends
b) subclasses
c) native
d) all of the mentioned

13) Polymorphism that is resolved during compile time is known as-----= b) Static Polymorphism

a) Dynamic Polymorphism
b) Static Polymorphism
c) Special Polymorphism
d) None of the mentioned

14) Which of the following statements is correct?
a) Public method is accessible to all other classes in the hierarchy
b) Public method is accessible only to subclasses of its parent
class c) Public method can only be called by object of its class
d) Public method can be accessed by calling object of the public class

15) Which of the following concept means waiting until runtime to determine which function to call?
a) Data hiding
b) Dynamic loading
c) Dynamic Casting
d) Dynamic Binding

16)Break statement in switch case
a.prevents from fallthrough
b.causes an exit from innermost loop
c.both a and b
d.none

17) The keyword 'this' is used
a) As reference to current object
b) Explicit constructor invocation
c) In open recursion
d) All the above

18) When does method overloading is determined?
a) At run time
b) At compile time
c) At coding time
d) At execution time

19) Which of these keywords is used to make a class?
a) class
b) struct
c) int
d) none of the mentioned

20) Which of the following is a valid declaration of an object of class
Box? a) Box obj = new Box();
b) Box obj = new Box;
c) obj = new Box();
d) new Box obj;

21) Which of these operators is used to allocate memory for an
object? a) malloc

b) alloc
c) new
d) give
b)

22) Which of the following statements is correct?
a) Public method is accessible to all other classes in the hierarchy b) Public method is accessible only to subclasses of its parent class
c) Public method can only be called by object of its class
d) Public method can be accessed by calling object of the public class

23) . What will be the output of the following Java program?

```
    class box
{
    int width;
    int height;
    int length;
    public static void main(Stringargs[])
    {
                boxobj=new box();
                obj.width=10;
                obj.height=2;
                obj.length=10;
                int y =obj.width*obj.height*obj.length;
                System.out.print(y);}}
```

a) 12
b) 200
c) 400
d) 100

24) What is the process of defining two or more methods within same class that have same name but different parameters declaration?
a) method overloading
b) method overriding
c) method hiding
d) none of the mentioned

25) Which of these can be overloaded?
a) Methods
b) Constructors
c) a & b
d) None of the above

26) Which of these keywords is used in Java to prevent value of a variable from being modified?

a)final

b) void
c) constant
d) static

27) What will be the output of the following Java program?

```
class Output
{
Public static voidmain(Stringargs[])
{
            int a1[]=new int[10];
            int a2[]={1, 2, 3, 4, 5};
            System.out.println(a1.length+""+ a2.length);
}}
```

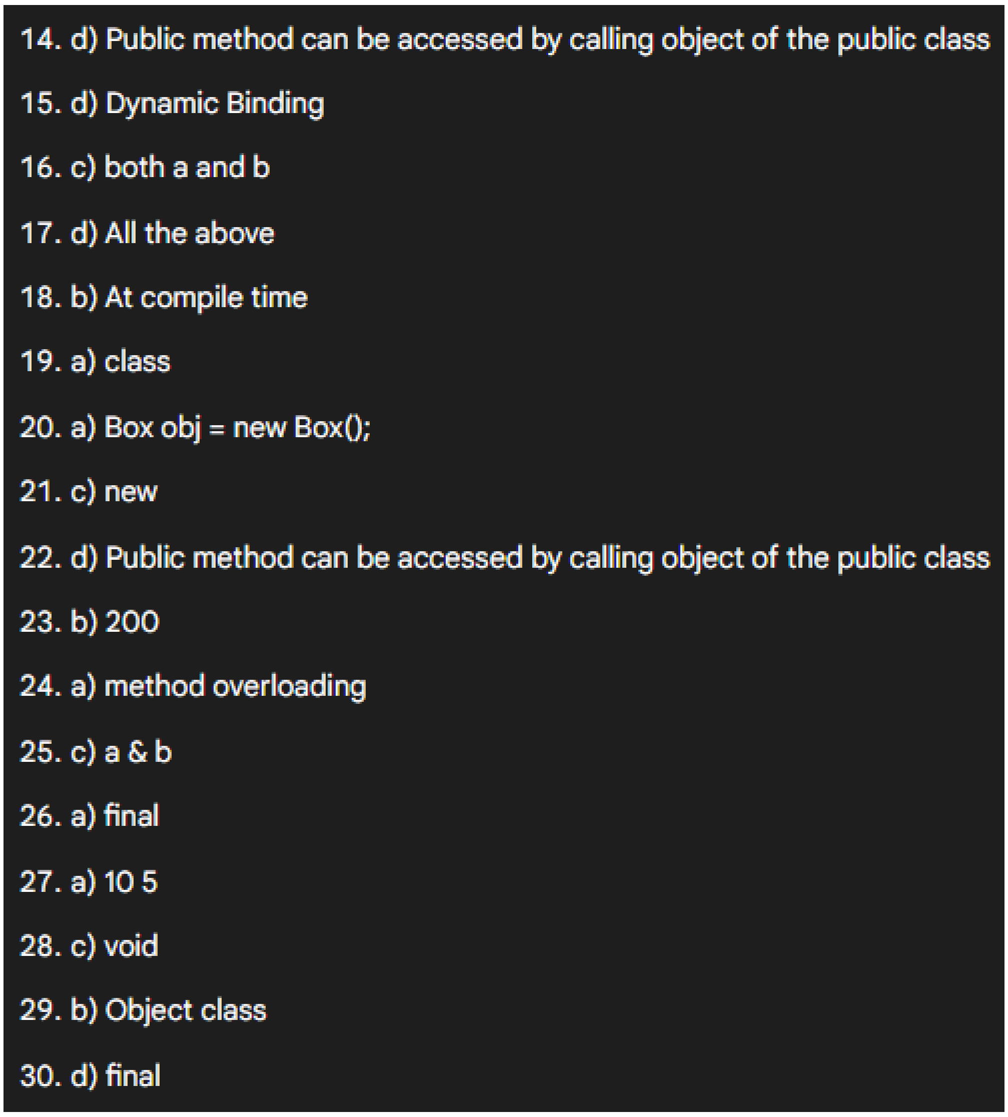
a) 10 5
b) 5 10
c) 0 10
d) 0 5

28) What is the return type of a method that does not return any value?

a) int
b) float
c) void
d) double

29) Which of these classes is superclass of every class in Java?
a) String class
b) Object class
c) Abstract class
d) ArrayList class

30) Which of these keywords can be used to prevent inheritance of a class? a) super
b) constant
c) class
d) final


14. d) Public method can be accessed by calling object of the public class

15. d) Dynamic Binding

16. c) both a and b

17. d) All the above

18. b) At compile time

19. a) class

20. a) Box obj = new Box();

21. c) new

22. d) Public method can be accessed by calling object of the public class

23. b) 200

24. a) method overloading

25. c) a & b

26. a) final

27. a) 10 5

28. c) void

29. b) Object class

30. d) final

Question bank :

1) Discuss the characteristics of Object Oriented Programming.
2) Write a note on Java Virtual Machine.
3) JVM is platform dependent. Justify.
4) How is for-each loop is used in Java? Illustrate with a suitable example.
5) Explain Data Types in detail with example.
6) Explain any three methods of StringBuffer class with appropriate example.
7) Create a two dimensional array. Instantiate and Initialize it.
8) What is a Jagged Array in Java ? How is a Jagged array declared and defined in Ja
9) Define following. 1) Byte code 2) Java Virtual Machine
10) Explain type-conversion in java.
11) Compare different types of variables in Java with a suitable example.
12) What is the difference between while and do-while statement
13) Discuss any three features of Java Programming Language
14) Differentiate between constructor and method of a class
15) How is "this" keyword used to access the data members in a class? Explain it with asuitable Java program.

16)Explain the words “static” and “this” with the help of example
17) Declare a class “Product” which has private data members such as “id”, “name” And“price”. Write constructors for “Product” class, display() method to display values of theobjects.
18) Write a program to create circle class with area function to find area of circle.
19) Enlist types of constructors. Explain any two with example.
20) Explain static variable and static method with example
21) What are the different access control/specifiers in Java. Explain “default” access controlusing the Java code.
22) How is “this” keyword used to access the data members in a class? Explain it with Asuitable Java program.
23) Illustrate the concept of polymorphism/method overloading in Java with a suitable example.
24) What is wrapper class? What is the use of wrapper class in Java? 25) Define a class Box with length and breadth as its data members. Write down withoutargument and parameterized constructor for the class.
26) Explain the uses of “static “ keyword in Java.
27) Write a program to implement the Fibonacci series using for loop control statement. 28) Explain features of JAVA.
29) Explain instance of operator.
30) List features of Java. Briefly explain any two.
31) Compare Object oriented programming with procedural programming.
32) Explain type-conversion in java.
33) Explain JRE, JDK and JIT.
34) Why is main defined as a public static method. Justify the answer. 35)
What are syntax errors (compile errors), runtime errors, and logical errors? 36)
Explain Data Types in Java detail with suitable examples.
37) Write a program to implement the fibonacci series using for loop control statement. 38) Compare while and do-while loop with proper example.
39) Write a program that creates and initializes a four integer element array. Calculate and display the average of its values
40) Create a two dimensional array. Instantiate and Initialize it.
41) Explain method overriding with an example Write a Java program which shows an example of method overriding.
42) Differentiate between Method overloading and Method overriding
43) Explain passing argument by values with example.
44) What do you mean by constructor overloading? Explain constructor overloading with a suitable example.
45) Differentiate between constructor and method of a class.
46) What do you mean by method overloading? Explain method overloading with suitableexample.
47) Explain the words “static” and “this” with the help of example
48) When will you declare a method as a static? Explain static method with suitable Example
49) Write a program to create circle class with area function to find area of circle
50) Explain class and object with respect to java
51) List and explain available type of constructors in java with example.
52) How to access object via reference variable? Explain with example.
53) Explain static keyword with example

54) Define constructor. How objects are constructed?
55) Write short notes on access specifiers and modifiers in java.
56) Explain static variable and static method with example
57) Explain the concept of multilevel inheritance with a suitable example. 58)
Explain the conceptof multiple inheritance with a suitable example. 59)
Describe the following method related to String i)replace(), ii)compareTo(). 60)
Differentiate String class and StringBuffer class
61) Describe Inheritance and its type with suitable example.
62) Discuss Following with example: (i) super Keyword (ii) final Keyword (iii) this keyword
63) What is dynamic method dispatch? Explain with suitable example. 64)
Write a java program to explain runtime polymorphism using interface 65)
What do you mean by Interface? Explain with an example.
66) Compare interface and abstract class with suitable example
67) What do you mean by Interface? Explain it with suitableexample.
68) Explain super keyword with example.
69) Differentiate between final, finally and finalize. What will happen if we make class &method as final?
70) Explain abstract class with example.
71) How can we protect sub class to override the method of super class? Explain with anexample
72) Explain Primitive data type and wrapper class data types.
73) Explain about Encapsulation, Abstraction.
74) Explain Exception handling in JAVA.
75) Discuss exception and error in Java
76) Explain use of throw in exception handling with example.
77) When do we use throw statement? Explain it by giving an example 78)
What is the keyword throws used for? Illustrate with a suitable example 79)
Demonstrate how you can handle different types of exception separately. 80)
Explain use of finally in exception handling with an example.
81) Explain use of following methods with suitable example isAlive(), join(),
getPriority() 82) List out ibuilt-in Java exceptions.
83) Write a program using BufferedInputStream, FileInputStream,BufferedOutputStream, FileOutputStream to copy content of one file Test1.txt intoanother file Test2.txt
84) What is a Java Package and which package is imported by default?
85) How we can create a package?
86) Explain usage of class FileInputStream and FileOutputStream by giving an example
87) Create a class called Student. Write a student manager program to read and write
the student information from and to files by using the
BufferedReader&BufferedWriter
88) The abstract vegetable class has three subclasses named Potato, Brinjal Declare one instance variable of type String that indicates the color ofa vegetable. Create and display instances of these objects. Override the toString()method of object to return a string with the name of vegetable & its colour
89) Write a program to rise and handle divide by zero exception.
90) Explain the exception hierarchy. Explain how to throw, catch and handle
Exceptions. 91) What is a stream? List down the types of steam.
92) Explain usage of class FileInputStream and FileOutputStream by giving an example.

93) Write short notes about I/O stream classes.
94) Explain File class with its methods.Explain multithreading in java using example
95) What are the ways in which you can create a thread? Explain with example 96)
Explain Thread Life Cycle in detail
97) Explain the use of join() with an example.
98) Explain the use of sleep() with an example.
99) List down the thread methods & explain any two.
100) Explain Thread Synchronization with example.
101) What is constraint? Explain constraints on generalization sets

1) Characteristics of Object Oriented Programming
Encapsulation: This is the bundling of data (attributes) and methods (functions) that operate on the data into a single unit or class. It restricts direct access to some of the object's components, which can prevent the accidental modification of data.

Abstraction: Abstraction is the concept of hiding the complex reality while exposing only the necessary parts. In Java, abstraction can be achieved through abstract classes and interfaces.

Inheritance: This allows a new class (subclass) to inherit attributes and methods from an existing class (superclass). This promotes code reusability. MAKE A PDF FOR THE ABOVE

Polymorphism: This allows methods to do different things based on the object it is acting upon. It can be achieved through method overloading and method overriding.

2) Java Virtual Machine (JVM)
The JVM is an abstract machine that enables computers to run Java programs. It converts Java bytecode into machine language. This makes Java platform-independent, as the same bytecode can run on any machine that has a compatible JVM.

3) JVM is Platform Dependent. Justify.
While Java bytecode is platform-independent, the JVM itself is platform-dependent because each operating system requires a specific implementation of the JVM to convert bytecode into machine code. Thus, while Java programs can run on any system with a JVM, the JVM must be tailored to each operating system.

4) For-each Loop in Java
The for-each loop is used to iterate through arrays or collections in Java. It simplifies the code and makes it easier to read.

java

Verify

Open In Editor
Edit
Copy code

```
int[] numbers = {1, 2, 3, 4, 5};
```

```
for (int number : numbers) {
    System.out.println(number);
}
```
5) Data Types in Java
Java has two categories of data types:

Primitive Data Types: These include byte, short, int, long, float, double, char, and boolean. They hold their values directly.

Reference Data Types: These include objects and arrays. They hold references to the actual data. Example:

java

Verify

Open In Editor
Edit
Copy code
```
int num = 10; // Primitive type
String name = "Java"; // Reference type
```
6) Methods of StringBuffer Class
append(): Adds a string to the end of the StringBuffer.
java

Verify

Open In Editor
Edit
Copy code
```
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");
System.out.println(sb); // Output: Hello World
```
insert(): Inserts a string at the specified index.
java

Verify

Open In Editor
Edit
Copy code
```
sb.insert(5, " Java");
System.out.println(sb); // Output: Hello Java World
```
reverse(): Reverses the characters in the StringBuffer.
java

Verify

Open In Editor

Edit
Copy code

```
sb.reverse();
System.out.println(sb); // Output: dlroW avaJ olleH
```

7) Two Dimensional Array
java

Verify

Open In Editor
Edit
Copy code

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

8) Jagged Array in Java
A jagged array is an array of arrays where each inner array can have different lengths.

java

Verify

Open In Editor
Edit
Copy code

```
int[][] jaggedArray = new int[3][];
jaggedArray[0] = new int[2]; // 2 elements
jaggedArray[1] = new int[3]; // 3 elements
jaggedArray[2] = new int[1]; // 1 element
```

9) Definitions
Bytecode: Bytecode is the intermediate representation of Java code, generated by the Java compiler, which can be executed by the JVM.

Java Virtual Machine: The JVM is an engine that provides a runtime environment to execute Java bytecode. It converts bytecode into machine language.

10) Type Conversion in Java
Type conversion in Java refers to converting a variable from one data type to another. It can be:

Widening Conversion: Automatic conversion (e.g., int to float).
Narrowing Conversion: Manual conversion (e.g., float to int).
java

Verify

Open In Editor

Edit
Copy code
```
int a = 10;
float b = a; // Widening
float c = 10.5f;
int d = (int) c; // Narrowing
```
11) Types of Variables in Java
Local Variables: Declared within a method or block and are accessible only within that method/block.
Instance Variables: Declared in a class but outside any method and are accessible to all methods in the class.
Static Variables: Declared with the static keyword and shared among all instances of a class.
java

Verify

Open In Editor
Edit
Copy code
```
class Example {
   static int staticVar; // Static Variable
   int instanceVar; // Instance Variable

   void method() {
      int localVar = 5; // Local Variable
   }
}
```
12) While vs. Do-While Statement
While: Checks the condition before executing the loop body.
java

Verify

Open In Editor
Edit
Copy code
```
int i = 0;
while (i < 5) {
   System.out.println(i);
   i++;
}
```
Do-While: Executes the loop body at least once before checking the condition.
java

Verify

Open In Editor
Edit

Copy code
```
int j = 0;
do {
   System.out.println(j);
   j++;
} while (j < 5);
```
13) Features of Java Programming Language
Platform Independence: Java code is compiled into bytecode, which can run on any platform with a JVM.
Object-Oriented: Java is based on the principles of OOP, which promotes code reusability and modularity.
Automatic Memory Management: Java has a garbage collector that automatically manages memory, reducing memory leaks.
14) Constructor vs. Method
Constructor: A special method invoked when an object is created. It has the same name as the class and does not have a return type.
Method: A block of code that performs a specific task and can be called multiple times. It can have any name and a return type.
java

Verify

Open In Editor
Edit
Copy code
```
class MyClass {
   MyClass() { // Constructor
      System.out.println("Constructor called");
   }

   void myMethod() { // Method
      System.out.println("Method called");
   }
}
```
15) Using "this" Keyword
The this keyword refers to the current object. It is used to access class members and differentiate between instance variables and parameters.

java

Verify

Open In Editor
Edit
Copy code
```
class Product {
   private int id;
   private String name;
```

```
    Product(int id, String name) {
        this.id = id; // 'this' differentiates instance variable from parameter
        this.name = name;
    }

    void display() {
        System.out.println("ID: " + id + ", Name: " + name);
    }
}
```

16) "static" and "this" Keywords
static: Indicates that a variable or method belongs to the class rather than instances of the class. It can be accessed without creating an instance of the class.
this: Refers to the current instance of the class.
java

Verify

Open In Editor
Edit
Copy code

```
class Example {
    static int staticVar = 10; // Static variable
    int instanceVar;

    Example(int instanceVar) {
        this.instanceVar = instanceVar; // 'this' refers to the instance variable
    }

    static void staticMethod() {
        System.out.println("Static variable: " + staticVar);
    }
}
```

17) Class "Product"
java

Verify

Open In Editor
Edit
Copy code

```
class Product {
    private int id;
    private String name;
    private double price;

    Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    void display() {
        System.out.println("ID: " + id + ", Name: " + name + ", Price: " + price);
    }
}
```
18) Circle Class with Area Function
java

Verify

Open In Editor
Edit
Copy code
```
class Circle {
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    double area() {
        return Math.PI * radius * radius;
    }
}
```
19) Types of Constructors
Default Constructor: A constructor with no parameters.
Parameterized Constructor: A constructor that takes parameters to initialize an object.
java

Verify

Open In Editor
Edit
Copy code
```
class Box {
    int length, breadth;

    Box() { // Default Constructor
        length = 1;
        breadth = 1;
    }

    Box(int l, int b) { // Parameterized Constructor
        length = l;
        breadth = b;
    }
}
```
20) Static Variable and Static Method

Static Variable: A variable that is shared among all instances of a class.
Static Method: A method that can be called without creating an instance of the class.
java

Verify

Open In Editor
Edit
Copy code

```java
class Counter {
   static int count = 0; Counter() {
      count++;
   }

   static void displayCount() {
      System.out.println("Count: " + count);
   }
}
```

21) Access Control Specifiers in Java
Java has four access specifiers:

public: Accessible from any other class.
protected: Accessible within the same package and subclasses.
default: Accessible only within the same package (no keyword).
private: Accessible only within the same class.
java

Verify

Open In Editor
Edit
Copy code

```java
class Example {
   public int publicVar;
   protected int protectedVar;
   int defaultVar; // default access
   private int privateVar;
}
```

22) Using "this" Keyword to Access Data Members
java

Verify

Open In Editor
Edit
Copy code

```java
class Employee {
   private String name;
   private int age;
```

```
    Employee(String name, int age) {
        this.name = name; // 'this' refers to the instance variable
        this.age = age;
    }

    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```
23) Polymorphism/Method Overloading
Polymorphism allows methods to perform different tasks based on the object it is acting upon. Method overloading is a type of polymorphism where multiple methods have the same name but different parameters.

java

Verify

Open In Editor
Edit
Copy code
```
class MathOperations {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}
```
24) Wrapper Class
A wrapper class in Java is a class that encapsulates a primitive data type into an object. It provides methods to convert between types and manipulate the data.

Use: To use primitive types in collections (like ArrayList) or to perform operations that require objects.
java

Verify

Open In Editor
Edit
Copy code
```
Integer intObj = Integer.valueOf(10); // Integer is a wrapper class for int
```
25) Class Box with Constructors
java

Verify

Open In Editor
Edit
Copy code
```
class Box {
    double length;
    double breadth;

    Box() { // Default Constructor
        length = 1.0;
        breadth = 1.0;
    }

    Box(double length, double breadth) { // Parameterized Constructor
        this.length = length;
        this.breadth = breadth;
    }
}
```
26) Uses of "static" Keyword
The static keyword is used for:

Declaring static variables that are shared among all instances of a class.
Declaring static methods that can be called without creating an instance of the class.
java

Verify

Open In Editor
Edit
Copy code
```
class Example {
    static int staticVar = 0;

    static void staticMethod() {
        System.out.println("Static variable: " + staticVar);
    }
}
```
27) Fibonacci Series Using For Loop
java

Verify

Open In Editor
Edit
Copy code
```
class Fibonacci {
    public static void main(String[] args) {
        int n = 10, firstTerm = 0, secondTerm = 1;
        System.out.println("Fibonacci Series up to " + n + ":");
        for (int i = 1; i <= n; ++i) {
            System.out.print(firstTerm + ", ");
            int nextTerm = firstTerm + secondTerm;
            firstTerm = secondTerm;
            secondTerm = nextTerm;
        }
    }
}
```
28) Features of Java
Simple: Java is easy to learn and use.
Object-Oriented: Supports OOP principles.
Secure: Provides a secure environment for developing applications.
Robust: Strong memory management and exception handling.
29) Instance of Operator
The instanceof operator is used to test whether an object is an instance of a specific class or subclass.

java

Verify

Open In Editor
Edit
Copy code
```
String str = "Hello";
boolean result = str instanceof String; // true
```
30) Features of Java
Multithreading: Java supports concurrent execution of two or more threads.
Dynamic: Java is designed to adapt to an evolving environment.
31) OOP vs. Procedural Programming
OOP: Focuses on objects and data encapsulation. Promotes code reusability through inheritance and polymorphism.
Procedural: Focuses on procedures or routines. Code is organized into functions.
32) Type Conversion in Java
Type conversion can be:

Implicit: Automatic conversion (e.g., int to double).
Explicit: Manual conversion (e.g., double to int).
java

Verify

Open In Editor
Edit
Copy code
```
double d = 10.5;
int i = (int) d; // Explicit conversion
```
33) JRE, JDK, and JIT
JRE (Java Runtime Environment): Provides the libraries, Java Virtual Machine (JVM), and

other components to run applications written in Java.

JDK (Java Development Kit): A software development kit that includes JRE, development tools (like the Java compiler), and libraries for developing Java applications.

JIT (Just-In-Time Compiler): A component of the JVM that improves the performance of Java applications by compiling bytecode into native machine code at runtime.

34) Why is main defined as a public static method?
The main method is defined as public so that it can be called by the Java Virtual Machine (JVM) from outside the class. It is static because it can be invoked without creating an instance of the class. The signature of the main method is:

java

Verify

Open In Editor
Edit
Copy code
```
public static void main(String[] args) {
   // Code
}
```
35) Syntax Errors, Runtime Errors, and Logical Errors
Syntax Errors: Errors that occur when the code violates the syntax rules of the programming language. These are caught at compile time.
Runtime Errors: Errors that occur during the execution of the program, such as division by zero or accessing an out-of-bounds array index.
Logical Errors: Errors that occur when the program runs without crashing but produces incorrect results due to a flaw in the logic.
36) Data Types in Java
Java has two main categories of data types:

Primitive Data Types: These include byte, short, int, long, float, double, char, and boolean. They hold their values directly.
Reference Data Types: These include objects and arrays. They hold references to the actual data. Example:
java

Verify

Open In Editor
Edit
Copy code
```
int num = 10; // Primitive type
String name = "Java"; // Reference type
```
37) Fibonacci Series Using For Loop
java

Verify

Open In Editor
Edit
Copy code

```
class Fibonacci {
    public static void main(String[] args) {
        int n = 10, firstTerm = 0, secondTerm = 1;
        System.out.println("Fibonacci Series up to " + n + ":");
        for (int i = 1; i <= n; ++i) {
            System.out.print(firstTerm + ", ");
            int nextTerm = firstTerm + secondTerm;
            firstTerm = secondTerm;
            secondTerm = nextTerm;
        }
    }
}
```

38) Compare While and Do-While Loop

While Loop: The condition is checked before executing the loop body. If the condition is false initially, the loop body may not execute at all.

java

Verify

Open In Editor
Edit
Copy code

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

Do-While Loop: The loop body is executed at least once before the condition is checked.

java

Verify

Open In Editor
Edit
Copy code

```
int j = 0;
do {
    System.out.println(j);
    j++;
} while (j < 5);
```

39) Average of Integer Array

java

Verify

Open In Editor
Edit
Copy code

```
class Average {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40};
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        double average = sum / (double) numbers.length;
        System.out.println("Average: " + average);
    }
}
```

40) Two Dimensional Array
java

Verify

Open In Editor
Edit
Copy code

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

41) Method Overriding Example
java

Verify

Open In Editor
Edit
Copy code

```
class Animal {
    void sound() {
        System.out.println("Animal makes sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}

class Test {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        myDog.sound(); // Output: Dog barks
    }
}
```
42) Method Overloading vs. Method Overriding
Method Overloading: Same method name with different parameters in the same class.
Method Overriding: Same method name and parameters in a subclass.
43) Passing Arguments by Value
In Java, arguments are passed by value, meaning a copy of the variable is passed to the method.

java

Verify

Open In Editor
Edit
Copy code
```
class Test {
    void increment(int num) {
        num++;
    }

    public static void main(String[] args) {
        Test obj = new Test();
        int number = 5;
        obj.increment(number);
        System.out.println("Number: " + number ); // Output: Number: 5
    }
}
```
44) Constructor Overloading
Constructor overloading occurs when a class has multiple constructors with different parameters.

java

Verify

Open In Editor
Edit
Copy code
```
class Rectangle {
    int length, breadth;

    Rectangle() { // Default constructor
        length = 1;
        breadth = 1;
    }
    Rectangle(int l, int b) { // Parameterized constructor
        length = l;
        breadth = b;
    }
}
```
45) Constructor vs. Method

Constructor: A special method used to initialize objects. It has the same name as the class and no return type.

Method: A block of code that performs a specific task and can be called multiple times. It can have any name and a return type.

java

Verify

Open In Editor
Edit
Copy code
```
class MyClass {
    MyClass() { // Constructor
        System.out.println("Constructor called");
    }

    void myMethod() { // Method
        System.out.println("Method called");
    }
}
```
46) Method Overloading

Method overloading allows a class to have more than one method with the same name, as long as their parameter lists are different.

java

Verify

Open In Editor
Edit
Copy code
```
class MathOperations {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}
```
47) "static" and "this" Keywords

static: Indicates that a variable or method belongs to the class rather than instances of

the class. It can be accessed without creating an instance of the class.
this: Refers to the current instance of the class.
java

Verify

Open In Editor
Edit
Copy code

```java
class Example {
   static int staticVar = 10; // Static variable
   int instanceVar;

   Example(int instanceVar) {
      this.instanceVar = instanceVar; // 'this' refers to the instance variable
   }

   static void staticMethod() {
      System.out.println("Static variable: " + staticVar);
   }
}
```

48) Declaring a Method as Static
A method is declared as static when it does not require an instance of the class to be invoked. Static methods can access static variables and methods directly.

java

Verify

Open In Editor
Edit
Copy code

```java
class Counter {
   static int count = 0;

   Counter() {
      count++;
   }

   static void displayCount() {
      System.out.println("Count: " + count);
   }
}
```

49) Circle Class with Area Function
java

Verify

Open In Editor

Edit
Copy code
```
class Circle {
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    double area() {
        return Math.PI * radius * radius;
    }
}
```
50) Class and Object in Java
Class: A blueprint for creating objects. It defines properties (attributes) and behaviors (methods).
Object: An instance of a class. It represents a specific implementation of the class.
java

Verify

Open In Editor
Edit
Copy code
```
class Car {
    String model;
    int year;

    Car(String model, int year) {
        this.model = model;
        this.year = year;
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", 2020); // Object creation
    }
}
```
51) Types of Constructors in Java
Default Constructor: A constructor with no parameters.
Parameterized Constructor: A constructor that takes parameters to initialize an object.
java

Verify

Open In Editor
Edit
Copy code

```
class Box {
   int length, breadth;

   Box() { // Default Constructor
      length = 1;
      breadth = 1;
   }

   Box(int l, int b) { // Parameterized Constructor
      length = l;
      breadth = b;
   }
}
```
52) Accessing Object via Reference Variable
An object can be accessed using a reference variable, which holds the memory address of the object.

java

Verify

Open In Editor
Edit
Copy code
```
class Student {
   String name;

   Student(String name) {
      this.name = name;
   }
}

public class Main {
   public static void main(String[] args) {
      Student student = new Student("Alice"); // Reference variable
      System.out.println(student.name);
   }
}
```
53) Static Keyword with Example
The static keyword is used to declare class-level variables and methods that can be accessed without creating an instance of the class.

java

Verify

Open In Editor
Edit
Copy code

```
class Example {
    static int staticVar = 10;

    static void staticMethod() {
        System.out.println("Static variable: " + staticVar);
    }
}
```

54) Constructor Definition
A constructor is a special method used to initialize objects. It is called when an object of a class is created. Constructors can have parameters (parameterized constructors) or no parameters (default constructors). The constructor has the same name as the class and does not have a return type.

55) Access Specifiers and Modifiers in Java
Access specifiers determine the visibility of classes, methods, and variables. The main access specifiers in Java are:

public: Accessible from any other class.
protected: Accessible within the same package and subclasses.
default: Accessible only within the same package (no keyword).
private: Accessible only within the same class.
Modifiers like final, static, and abstract further define the behavior of classes and methods.

56) Static Variable and Static Method
Static Variable: A variable that is shared among all instances of a class. It is declared with the static keyword.
Static Method: A method that can be called without creating an instance of the class. It can access static variables and methods directly.
java

Verify

Open In Editor
Edit
Copy code

```
class Example {
    static int staticVar = 0;

    static void staticMethod() {
        System.out.println("Static variable: " + staticVar);
    }
}
```

57) Multilevel Inheritance
Multilevel inheritance occurs when a class is derived from another derived class. This creates a hierarchy of classes.

java

Verify

Open In Editor
Edit
Copy code
```
class Animal {
   void eat() {
      System.out.println("Eating...");
   }
}

class Dog extends Animal {
   void bark() {
      System.out.println("Barking...");
   }
}

class Puppy extends Dog {
   void weep() {
      System.out.println("Weeping...");
   }
}
```
58) Multiple Inheritance
Java does not support multiple inheritance with classes to avoid ambiguity. However, it can be achieved through interfaces.

java

Verify

Open In Editor
Edit
Copy code
```
interface CanRun {
   void run();
}

interface CanBark {
   void bark();
}

class Dog implements CanRun, CanBark {
   public void run() {
      System.out.println("Dog is running");
   }

   public void bark() {
      System.out.println("Dog is barking");
   }
```

}
59) String Methods
replace(): Replaces all occurrences of a specified character or substring with another character or substring.
java

Verify

Open In Editor
Edit
Copy code
```
String str = "Hello World";
String newStr = str.replace("World", "Java");
System.out.println(newStr); // Output: Hello Java
```
compareTo(): Compares two strings lexicographically.
java

Verify

Open In Editor
Edit
Copy code
```
String str1 = "apple";
String str2 = "banana";
int result = str1.compareTo(str2);
System.out.println(result); // Output: negative value (since "apple" < "banana")
```
60) String Class vs. StringBuffer Class
String: Immutable; once created, its value cannot be changed. Any modification creates a new String object.
StringBuffer: Mutable; allows modification of the string without creating a new object.
java

Verify

Open In Editor
Edit
Copy code
```
String str = "Hello";
str = str + " World"; // Creates a new String object

StringBuffer sb = new StringBuffer("Hello");
sb.append(" World"); // Modifies the existing StringBuffer object
```
61) Inheritance and Its Types
Inheritance allows a class to inherit properties and methods from another class. Types of inheritance include:

Single Inheritance: One class inherits from another.
Multilevel Inheritance: A class inherits from a derived class.
Hierarchical Inheritance: Multiple classes inherit from a single superclass.

java

Verify

Open In Editor
Edit
Copy code

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

62) Keywords: super, final, this
super: Refers to the superclass and is used to access superclass methods and constructors.
java

Verify

Open In Editor
Edit
Copy code

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

class Child extends Parent {
   void display() {
      super.display(); // Calls Parent's display method
      System.out.println("Child class");
   }
}
```

final: Used to declare constants, prevent method overriding, and inheritance.
java

Verify

Open In Editor
Edit
Copy code

```
final int CONSTANT = 10; // Constant
final void display() {} // Cannot be overridden
```

this: Refers to the current object instance.
java

Verify

Open In Editor
Edit

```
Copy code
class Example {
   int value;

   Example(int value) {
      this.value = value; // 'this' differentiates instance variable from parameter
   }
}
```

63) Dynamic Method Dispatch

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime rather than compile-time. It is a key feature of polymorphism.

java

Verify

Open In Editor
Edit
Copy code

```
class ```java
Animal {
   void sound() {
      System.out.println("Animal makes sound");
   }
}

class Dog extends Animal {
   void sound() {
      System.out.println("Dog barks");
   }
}

class Test {
   public static void main(String[] args) {
      Animal myDog = new Dog(); // Upcasting
      myDog.sound(); // Output: Dog barks (dynamic method dispatch)
   }
}
```

64) Runtime Polymorphism Using Interface

Runtime polymorphism can also be achieved using interfaces. The method that gets called is determined at runtime based on the object type.

java

Verify

Open In Editor
Edit
Copy code

```
interface Animal {
   void sound();
}

class Dog implements Animal {
   public void sound() {
      System.out.println("Dog barks");
   }
}

class Cat implements Animal {
   public void sound() {
      System.out.println("Cat meows");
   }
}

class Test {
   public static void main(String[] args) {
      Animal myAnimal = new Dog(); // Upcasting
      myAnimal.sound(); // Output: Dog barks
      myAnimal = new Cat(); // Upcasting to Cat
      myAnimal.sound(); // Output: Cat meows
   }
}
```
65) What is an Interface?
An interface in Java is a reference type that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields. The methods in interfaces are abstract by default.

java

Verify

Open In Editor
Edit
Copy code
```
interface Animal {
   void sound(); // Abstract method
}

class Dog implements Animal {
   public void sound() {
      System.out.println("Dog barks");
   }
}
```
66) Compare Interface and Abstract Class
Interface: Cannot have any method implementations (until Java 8 with default methods). A class can implement multiple interfaces.
Abstract Class: Can have both abstract methods (without implementation) and concrete

methods (with implementation). A class can extend only one abstract class.
java

Verify

Open In Editor
Edit
Copy code
```java
abstract class Animal {
   abstract void sound(); // Abstract method
   void eat() { // Concrete method
      System.out.println("Eating...");
   }
}

interface CanRun {
   void run(); // Abstract method
}
```
67) What is an Interface? (Reiteration)
An interface is a contract that a class can implement. It defines methods that must be implemented by any class that chooses to implement the interface.

java

Verify

Open In Editor
Edit
Copy code
```java
interface Vehicle {
   void start();
   void stop();
}

class Car implements Vehicle {
   public void start() {
      System.out.println("Car started");
   }

   public void stop() {
      System.out.println("Car stopped");
   }
}
```
68) Explain Super Keyword
The super keyword is used to refer to the immediate parent class object. It can be used to access parent class methods and constructors.

java

Verify

Open In Editor
Edit
Copy code

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

class Child extends Parent {
   void display() {
      super.display(); // Calls Parent's display method
      System.out.println("Child class");
   }
}
```

69) Differentiate Between final, finally, and finalize
final: Used to declare constants, prevent method overriding, and inheritance.
finally: A block that follows a try-catch block and is executed regardless of whether an exception is thrown or not.
finalize(): A method called by the garbage collector before an object is removed from memory.
java

Verify

Open In Editor
Edit
Copy code

```
final int CONSTANT = 10; // Constant
try {
   // Code that may throw an exception
} catch (Exception e) {
   // Handle exception
} finally {
   // Cleanup code
}
```

70) Explain Abstract Class with Example
An abstract class is a class that cannot be instantiated and may contain abstract methods (without implementation) and concrete methods (with implementation).

java

Verify

Open In Editor
Edit
Copy code

```
abstract class Animal {
   abstract void sound(); // Abstract method

   void eat() { // Concrete method
      System.out.println("Eating...");
   }
}

class Dog extends Animal {
   void sound() {
      System.out.println("Dog barks");
   }
}
```
71) Protecting Subclass from Overriding
To prevent a subclass from overriding a method, declare the method as final.

java

Verify

Open In Editor
Edit
Copy code
```
class Parent {
   final void display() {
      System.out.println("This method cannot be overridden");
   }
}

class Child extends Parent {
   // void display() { // This will cause a compile-time error
   //     System.out.println("Trying to override");
   // }
}
```
72) Primitive Data Type and Wrapper Class Data Types
Primitive Data Types: Basic data types provided by Java (e.g., int, char, boolean).
Wrapper Class Data Types: Classes that encapsulate primitive data types into objects (e.g., Integer, Character, Boolean). They provide methods to convert between types and manipulate the data.
java

Verify

Open In Editor
Edit
Copy code
```
int primitiveInt = 10; // Primitive data type
Integer wrapperInt = Integer.valueOf(primitiveInt); // Wrapper class
```
73) Encapsulation and Abstraction

Encapsulation: The bundling of data (attributes) and methods (functions) that operate on the data into a single unit or class. It restricts direct access to some of the object's components, which can prevent the accidental modification of data.
java

Verify

Open In Editor
Edit
Copy code
```
class Account {
   private double balance; // Private variable

   public void deposit(double amount) {
      balance += amount; // Method to modify balance
   }

   public double getBalance() {
 {
      return balance; // Method to access balance
   }
}
```
Abstraction: The concept of hiding the complex reality while exposing only the necessary parts. In Java, abstraction can be achieved through abstract classes and interfaces.
java

Verify

Open In Editor
Edit
Copy code
```
abstract class Shape {
   abstract void draw(); // Abstract method
}

class Circle extends Shape {
   void draw() {
      System.out.println("Drawing Circle");
   }
}
```
74) Exception Handling in Java
Exception handling in Java is a powerful mechanism that allows developers to handle runtime errors, ensuring the normal flow of the application. It uses try, catch, finally, throw, and throws keywords.

java

Verify

Open In Editor
Edit
Copy code

```
try {
   int result = 10 / 0; // This will throw an ArithmeticException
} catch (ArithmeticException e) {
   System.out.println("Cannot divide by zero");
} finally {
   System.out.println("This block always executes");
}
```

75) Exception and Error in Java
Exception: An event that disrupts the normal flow of the program. It can be handled using try-catch blocks.
Error: A serious problem that a reasonable application should not try to catch. Errors are usually external to the application (e.g., OutOfMemoryError).
java

Verify

Open In Editor
Edit
Copy code

```
try {
   // Code that may throw an exception
} catch (Exception e) {
   // Handle exception
}
// Errors are not typically caught
```

76) Use of throw in Exception Handling
The throw keyword is used to explicitly throw an exception.

java

Verify

Open In Editor
Edit
Copy code

```
void checkAge(int age) {
   if (age < 18) {
      throw new IllegalArgumentException("Age must be 18 or older");
   }
}
```

77) When to Use throw Statement
The throw statement is used when you want to create a custom exception or when a certain condition is met that requires an exception to be thrown.

java

Verify

Open In Editor
Edit
Copy code

```
void validate(int age) {
   if (age < 18) {
      throw new IllegalArgumentException("Age must be at least 18");
   }
}
```

78) Keyword throws
The throws keyword is used in method signatures to declare that a method can throw exceptions. It allows the caller of the method to handle the exception.

java

Verify

Open In Editor
Edit
Copy code

```
void readFile() throws IOException {
   // Code that may throw IOException
}
```

79) Handling Different Types of Exceptions Separately
You can catch multiple exceptions separately using multiple catch blocks.

java

Verify

Open In Editor
Edit
Copy code

```
try {
   // Code that may throw exceptions
} catch (ArithmeticException e) {
   System.out.println("Arithmetic Exception occurred");
} catch (IOException e) {
   System.out.println("IO Exception occurred");
}
```

80) Use of finally in Exception Handling
The finally block is used to execute important code such as closing resources, regardless of whether an exception is thrown or not.

java

Verify

Open In Editor
Edit
Copy code

```
try {
   // Code that may throw an exception
} catch (Exception e) {
   System.out.println("Exception caught");
} finally {
   System.out.println("This block always executes");
}
```

81) Methods: isAlive(), join(), getPriority()
isAlive(): Checks if a thread is alive.
java

Verify

Open In Editor
Edit
Copy code

```
Thread t = new Thread();
t.start();
System.out.println(t.isAlive()); // true if the thread is running
```

join(): Waits for a thread to die.
java

Verify

Open In Editor
Edit
Copy code

```
t.join(); // Main thread waits for t to finish
```

getPriority(): Returns the priority of a thread.
java

Verify

Open In Editor
Edit
Copy code

```
int priority = t.getPriority(); // Gets the thread's priority
```

82) Built-in Java Exceptions
Java has several built-in exceptions, including:

NullPointerException
ArrayIndexOutOfBoundsException
ArithmeticException
ClassCastException
IOException
83) Copying File Content Using Streams

java

Verify

Open In Editor
Edit
Copy code
```
import java.io.*;

public class FileCopy {
   public static void main(String[] args) {
       try (BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("Test1.txt"));
            BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("Test2.txt"))) {
           byte[] buffer = new byte[1024];
           int bytesRead;
           while ((bytesRead = bis.read(buffer)) != -1) {
               bos.write(buffer, 0, bytesRead);
           }
           System.out.println("File copied successfully.");
       } catch (IOException e) {
           e.printStackTrace();
       }
   }
}
```
84) Java Package
A Java package is a namespace that organizes a set of related classes and interfaces. The default package imported by Java is java.lang, which contains fundamental classes.

85) Creating a Package
To create a package, use the package keyword at the top of your Java file.

java

Verify

Open In Editor
Edit
Copy code
```
package com.example.myapp;

public class MyClass {
   // Class code
}
```
86) FileInputStream and FileOutputStream Usage
FileInputStream is used to read bytes from a file, while FileOutputStream is used to write bytes to a file.

java

Verify

Open In Editor
Edit
Copy code
```
import java.io.*;

public class FileExample {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("input.txt");
             FileOutputStream fos = new FileOutputStream("output.txt")) {
            int data;
            while ((data = fis.read()) != -1) {
                fos.write(data);
            }
            System.out.println("File copied successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```
87) Student Manager Program
java

Verify

Open In Editor
Edit
Copy code
```
import java.io.*;

class Student {
    String name;
    int age;

    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age;
    }
}

public class StudentManager {
    public static void main(String[] args) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter("students.txt"))) {
            Student student1 = new Student("Alice", 20);
            Student student2 = new Student("Bob", 22);
            writer.write(student1.toString());
            writer.newLine();
            writer.write(student2.toString());
            System.out.println("Student information written to file.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

88) Abstract Vegetable Class
java

Verify

Open In Editor
Edit
Copy code

```
abstract class Vegetable {
    String color;

    Vegetable(String color) {
        this.color = color;
    }

    @Override
    public String toString() {
        return "Color: " + color;
    }
}

class Potato extends Vegetable {
    Potato(String color) {
        super(color);
    }

    @Override
    public String toString() {
        return "Potato - " + super.toString();
    }
}

class Brinjal extends Vegetable {
    Brinjal(String color) {
        super(color);
    }
    @Override
    public String toString() {
        return "Brinjal - " + super.toString();
    }
}

public class VegetableTest {
    public static void main(String[] args) {
        Vegetable potato = new Potato("Brown");
        Vegetable brinjal = new Brinjal("Purple");
        System.out.println(potato);
        System.out.println(brinjal);
    }
}
```

89) Divide by Zero Exception Handling
java

Verify

Open In Editor
Edit
Copy code

```
public class DivideByZero {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // This will throw ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero.");
        }
    }
}
```

90) Exception Hierarchy
Java exceptions are divided into two main categories: checked exceptions and unchecked exceptions. Checked exceptions must be declared in a method or constructor's throws clause, while unchecked exceptions do not need to be declared.

java

Verify

Open In Editor
Edit
Copy code

```
try {
    // Code that may throw an exception
} catch (IOException e) {
    // Handle checked exception
} catch (RuntimeException e) {
    // Handle unchecked exception
}
```
91) What is a Stream?
A stream is a sequence of data elements made available over time. In Java, streams are used for input and output operations. Types of streams include:

Byte Streams: Handle raw binary data (e.g., FileInputStream, FileOutputStream).
Character Streams: Handle character data (e.g., FileReader, FileWriter).
92) FileInputStream and FileOutputStream Usage
java

Verify

Open In Editor
Edit
Copy code
```
import java.io.*;

public class FileStreamExample {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("input.txt");
             FileOutputStream fos = new FileOutputStream("output.txt")) {
            int data;
            while ((data = fis.read()) != -1) {
                fos.write(data);
            }
            System.out.println("File copied successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```
93) I/O Stream Classes
Java provides several classes for input and output operations, including:

InputStream: Abstract class for reading byte streams.
OutputStream: Abstract class for writing byte streams.
Reader: Abstract class for reading character streams.
Writer: Abstract class for writing character streams.
94) File Class and Multithreading
The File class in Java represents a file or directory path. It provides methods to create, delete, and manipulate files and directories.

java

Verify

Open In Editor
Edit

Copy code

```
import java.io.File;

public class FileExample {
    public static void main(String[] args) {
        File file = new File("example.txt");
        try {
            if (file.createNewFile()) {
                System.out.println("File created: " + file.getName());
            } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

95) Ways to Create a Thread
There are two main ways to create a thread in Java:

By extending the Thread class:
java

Verify

Open In Editor
Edit
Copy code

```
class MyThread extends Thread {
    public void run() {
        System.out ```java
.println("Thread is running");
    }
}

public class ThreadExample {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start(); // Start the thread
    }
}
```

By implementing the Runnable interface:
java

Verify

Open In Editor
Edit
Copy code

```
class MyRunnable implements Runnable {
   public void run() {
      System.out.println("Thread is running");
   }
}

public class ThreadExample {
   public static void main(String[] args) {
      Thread thread = new Thread(new MyRunnable());
      thread.start(); // Start the thread
   }
}
```
96) Thread Life Cycle
The life cycle of a thread in Java includes several states:

New: The thread is created but not yet started.
Runnable: The thread is ready to run and waiting for CPU time.
Blocked: The thread is blocked waiting for a monitor lock.
Waiting: The thread is waiting indefinitely for another thread to perform a particular action.
Timed Waiting: The thread is waiting for another thread to perform an action for a specified waiting time.
Terminated: The thread has completed its execution.
97) Use of join()
The join() method allows one thread to wait for the completion of another thread.

java

Verify

Open In Editor
Edit
Copy code
```
class MyThread extends Thread {
   public void run() {
      System.out.println("Thread is running");
   }
}

public class ThreadExample {
   public static void main(String[] args) throws InterruptedException {
      MyThread thread = new MyThread();
      thread.start();
      thread.join(); // Main thread waits for thread to finish
      System.out.println("Thread has finished execution");
   }
}
```
98) Use of sleep()
The sleep() method pauses the execution of the current thread for a specified period.

java

Verify

Open In Editor
Edit
Copy code
```
class MyThread extends Thread {
    public void run() {
        try {
            System.out.println("Thread is sleeping");
            Thread.sleep(2000); // Sleep for 2 seconds
            System.out.println("Thread woke up");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class ThreadExample {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start();
    }
}
```
99) Thread Methods
start(): Starts the execution of the thread.
run(): Contains the code that defines the thread's behavior.
sleep(long millis): Causes the currently executing thread to sleep for the specified number of milliseconds.
join(): Waits for the thread to die.
isAlive(): Tests if the thread is alive.
100) Thread Synchronization
Thread synchronization is a mechanism that ensures that two or more concurrent threads do not simultaneously execute some particular program segment. It is used to prevent thread interference and memory consistency errors.

java

Verify

Open In Editor
Edit
Copy code
```
class Counter {
    private int count = 0;

    public synchronized void increment() { // Synchronized method
```

```
        count++;
    }

    public int getCount() {
        return count;
    }
}

public class ThreadExample {
    public static void main(String[] args) {
        Counter counter = new Counter();
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });
        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Final count: " + counter.getCount());
    }
}
```

101) Constraints on Generalization Sets
Constraints in generalization sets define the relationships between classes in an inheritance hierarchy. They can include:

Disjoint: A subclass can only be a member of one superclass.
Overlapping: A subclass can be a member of multiple superclasses.
Complete: All instances of a superclass must be represented in its subclasses.
Incomplete: Not all instances of a superclass need to be represented in its subclasses.