Object Oriented Programming using Java
19CST102

2 Marks

1. **Define OOPS.**

   Object-Oriented Programming System(OOPs) is a programming technique based on the concept of “objects” that contain data and methods.
   The six Main Principles of OOPS are Class, Object, Abstraction, Encapsulation, Inheritance, Polymorphism.

2. **Differentiate between encapsulation and abstraction.**

ABSTRACTION	ENCAPSULATION
Refers to showing only the necessary details to the intended user.	Means to hide(Data Hiding) wrapping, just hiding properties and methods.
Used in programming languages to make abstract class	Used to hide the code and data in a single unit to protect the data from the outside world.
Abstraction is implemented using interface and abstract class.	Encapsulation is implemented using private and protected access modifiers.

3. **What is an Object, give an example?**
   - ★ Object is a Real world entity, which is also known as an instance of a class. An Object consists of State(Attribute), Behaviour(Method) and Identity(Name).
   - ★ Example : Dog, Pen.

4. **What are the various types of Looping?**

   There are 4 types of looping in java, they are
   - For-Loop
   - While Loop
   - Do While Loop
   - For-each Loop

5. **Enumerate the importance of Bytecode in Java.**

Bytecode can be defined as an intermediate code generated by the compiler after the compilation of the source code(Java Program). This Intermediate code makes Java a platform-independent language.

13 Marks

6. **a) Write a detailed note on Concepts of OOPS.**
   - Object-Oriented Programming System(OOPs) is a programming technique based on the concept of "objects" that contain data and methods.
   - The six Main Principles of OOPS are Class, Object, Abstraction, Encapsulation, Inheritance, Polymorphism.

1) Class

A class is a blueprint or template that defines the properties (attributes) and behaviors (methods) that objects of that class can have. It serves as a fundamental building block for creating objects, encapsulating data and functionality, and enabling code reusability and organization.

2) Object

In Java, an object is an instance of a class that encapsulates data (attributes) and behavior (methods). It represents a specific entity or concept based on the class blueprint. Objects are created using the `new` keyword and can interact with other objects through method invocations, enabling complex program structures and interactions.

3) Abstraction

Abstraction in Java is a concept that focuses on providing a simplified, high-level view of complex systems. It involves hiding unnecessary details and exposing only essential features through abstract classes and interfaces. Abstraction allows for code modularization, promotes code reuse, and facilitates the implementation of complex systems with clear boundaries and interactions.

4) Encapsulation

Encapsulation in Java is a mechanism that combines data (attributes) and methods within a class, protecting data from direct access from outside. It allows for data hiding and access control, ensuring that data is only accessed and modified through defined methods. Encapsulation promotes code organization, security, and maintainability.

5) Inheritance

Inheritance in Java is a mechanism that allows classes to inherit properties and behaviors from other classes. It establishes a parent-child relationship between classes, where the child class (subclass) inherits the attributes and methods of the parent class (superclass). Inheritance promotes code reuse, hierarchical organization, and supports specialization and generalization of objects.

6) Polymorphism

- **"Polymorphism"** means **"Many Forms".** In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- Real-life Illustration Polymorphism: A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, and an employee. So the same person possesses different behavior in different situations. This is called polymorphism.

6. B) Define static members, datatype, variables using Java.

In Java, static members, data types, and variables can be defined as follows:

1. Static Members: Static members are class-level elements that are shared among all instances of the class. They are associated with the class itself rather than individual objects. Static members include static variables (also known as class variables) and static methods. They are accessed using the class name followed by the member name (e.g., ClassName.staticMember).

2. Data Types: Data types in Java define the kind of values that variables can hold. Java has two main categories of data types: primitive data types and reference data types. Primitive data types include int, double, boolean, char, etc., which hold simple values. Reference data types include classes, interfaces, arrays, and enumerated types.

3. Variables: Variables in Java are used to store and manipulate data. They have a specific data type, a unique name, and a value associated with them. Variables can be classified as local variables, instance variables (non-static member variables), and static variables (static member variables). Local variables are defined within methods or blocks, instance variables are declared in a class but outside methods, and static variables are associated with the class and shared among instances.

Here's an example demonstrating the usage of static members, data types, and variables in Java:

```
public class MyClass {
    // Static variable
    static int staticVar = 10;

    // Instance variable
    int instanceVar = 20;

    // Static method
    static void staticMethod() {
        System.out.println("This is a static method.");
    }

    public static void main(String[] args) {
        // Local variable
        int localVar = 30;

        // Accessing static variable
        System.out.println("Static Variable: " + staticVar);

        // Creating object of MyClass
        MyClass obj = new MyClass();

        // Accessing instance variable
        System.out.println("Instance Variable: " + obj.instanceVar);

        // Accessing local variable
        System.out.println("Local Variable: " + localVar);

        // Calling static method
        staticMethod();
    }
}
```

In this example, `staticVar` is a static variable, `instanceVar` is an instance variable, `localVar` is a local variable, and `staticMethod()` is a static method. The main method demonstrates accessing these variables and invoking the static method.

7. **A) Define Constructor and its types.**

Constructors in Java are special methods with the same name as the class. They initialize objects by assigning initial values and can be overloaded to create objects with different parameter sets.

There are three types of constructor in java, they are

1. Default constructor
2. Parameterized constructor
3. Copy constructor

1. Default Constructor

A constructor that has no parameters is known as default the constructor. A default constructor is invisible. And if we write a constructor with no arguments, the compiler does not create a default constructor. It is taken out. It is being overloaded and called a parameterized constructor. The default constructor changed into the parameterized constructor. But Parameterized constructor can't change the default constructor.

```
// Java Program to demonstrate Default Constructor

import java.io.*;


class DefaultConstructor{

        // Default Constructor
        DefaultConstructor() {
                        System.out.println("Default constructor");
        }


        public static void main(String[] args)
        {
                DefaultConstructor obj = new DefaultConstructor();
        }
}
```

2. Parameterized Constructor

A constructor that has parameters is known as a parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

```
// Java Program for Parameterized Constructor
import java.io.*;
class Geek {
	// data members of the class.
	String name;
	int id;
	Geek(String name, int id)
	{
		this.name = name;
		this.id = id;
	}
}
class GFG {
	public static void main(String[] args)
	{
		// This would invoke the parameterized constructor.
		Geek geek1 = new Geek("avinash", 68);
		System.out.println("GeekName :" + geek1.name
						+ " and GeekId :" + geek1.id);
	}
}
```

3. Copy Constructor

Unlike other constructors, a copy constructor is passed with another object which copies the data available from the passed object to the newly created object.

```
// Java Program for Copy Constructor
import java.io.*;
```

```
class Geek {
	// data members of the class.
	String name;
	int id;

	// Parameterized Constructor
	Geek(String name, int id)
	{
		this.name = name;
		this.id = id;
	}

	// Copy Constructor
	Geek(Geek obj2)
	{
		this.name = obj2.name;
		this.id = obj2.id;
	}
}
class GFG {
	public static void main(String[] args)
	{
		// This would invoke the parameterized constructor.
		System.out.println("First Object");
		Geek geek1 = new Geek("avinash", 68);
		System.out.println("GeekName :" + geek1.name
						+ " and GeekId :" + geek1.id);

		System.out.println();

		// This would invoke the copy constructor.
		Geek geek2 = new Geek(geek1);
		System.out.println(
			"Copy Constructor used Second Object");
		System.out.println("GeekName :" + geek2.name
						+ " and GeekId :" + geek2.id);
	}
}
```

7. B) Define JDK and JVM in Detail.

Parameter	JDK	JVM
Full-Form	The JDK is an abbreviation for Java Development Kit.	The JVM is an abbreviation for Java Virtual Machine.
Definition	The JDK (Java Development Kit) is a software development kit that develops applications in Java. Along with JRE, the JDK also consists of various development tools (Java Debugger, JavaDoc, compilers, etc.)	The Java Virtual Machine (JVM) is a platform-independent abstract machine that has three notions in the form of specifications. This document describes the requirement of JVM implementation.
Functionality	The JDK primarily assists in executing codes. It primarily functions in development.	JVM specifies all of the implementations. It is responsible for providing all of these implementations to the JRE.
Platform Dependency	The JDK is platform-dependent. It means that for every different platform, you require a different JDK.	The JVM is platform-independent. It means that you won’t require a different JVM for every different platform.
Tools	Since JDK is primarily responsible for the development, it consists of various tools for debugging, monitoring, and developing java applications.	JVM does not consist of any tools for software development.

Implementation	**JDK = Development Tools + JRE (Java Runtime Environment)**	**JVM = Only the runtime environment that helps in executing the Java bytecode.**
Why Use It?	**Why use JDK?** **Some crucial reasons to use JDK are:** ● **It consists of various tools required for writing Java programs.** ● **JDK also contains JRE for executing Java programs.** ● **It includes an Appletviewer, Java application launcher, compiler, etc.** ● **The compiler helps in converting the code written in Java into bytecodes.** ● **The Java application launcher helps in opening a JRE. It then loads all of the necessary details and then executes all of its main methods.**	**Why use JVM?** **Some crucial reasons to use JVM are:** ● **It provides its users with a platform-independent way for executing the Java source code.** ● **JVM consists of various tools, libraries, and multiple frameworks.** ● **The JVM also comes with a Just-in-Time (JIT) compiler for converting the Java source code into a low-level machine language. Thus, it ultimately runs faster than any regular application.** ● **Once you run the Java program, you can run JVM on any given platform to save your time.**

Features	Features of JDK	Features of JVM
	• Here are a few crucial features of JDK: • It has all the features that JRE does. • JDK enables a user to handle multiple extensions in only one catch block. • It basically provides an environment for developing and executing the Java source code. • It has various development tools like the debugger, compiler, etc. • One can use the Diamond operator to specify a generic interface in place of writing the exact one. • Any user can easily install JDK on Unix, Mac, and Windows OS (Operating Systems).	Here are a few crucial features of JVM: • The JVM enables a user to run applications on their device or in a cloud environment. • It helps in converting the bytecode into machine-specific code. • JVM also provides some basic Java functions, such as garbage collection, security, memory management, and many more. • It uses a library along with the files given by JRE (Java Runtime Environment) for running the program. • Both JRE and JDK contain JVM. • It is easily customizable. For instance, a user can feasibly allocate a maximum and minimum memory to it. • JVM can also execute a Java program line by line. It is thus also known as an interpreter.

14 Marks

8. **write a Java Program to create a calculator using Switch Statement**

```
//Java Program to create a Calculator using Switch Statement
import java.util.Scanner;

public class SimpleCalculator{

public static void main(String[] args){

   double num1,num2;
   Scanner scanner = new Scanner(System.in);
   System.out.print("Enter First number:");
   num1 = scanner.nextDouble();
   System.out.println("Enter Second number:");
   num2 = scanner.nextDouble();
   System.out.println("Enter an Operator ( + , - , * , / ):");
   char operator = scanner.next().charAt(0);
   scanner.close();

   double output;
      switch(operator){
         case '+':
         output = num1+num2;
         break;
         case '-':
         output = num1-num2;
         break;
         case'*':
         output = num1 * num2;
         break;
         case '/':
         output = num1/num2;
         break;
```

```
            default:
            System.out.printf("You have entered wrong operator");
            return;
    }
            System.out.println(num1+" "+operator+" "+num2+": "+output);
    }

    }
```

9. **Java program to Print all the Departments of SNS College of Technology using array and for-loop.**

```
            public class SNScollege {
               public static void main(String[] args) {
                  String[] Department = {"AIML", "IT", "CSE", "ECE", "EEE",
    "MECH", "CIVIL", "FT", "AGRI", "AUTO", "AEROSPACE", "MCT", "BME"};

                  // Display the department names
                  System.out.println("Departments in SNS College:");
                  for (int i = 0 ; i<=12 ; i++) {
                     System.out.println(Department[i]);
                  }
               }
            }
```