JAVA Assignment

Name: Sakshi Ramesh Mogali

AF Code: AF04954116

Batch Code: ANP – D1544

1. Java Basics

1. What is Java? Explain its features.

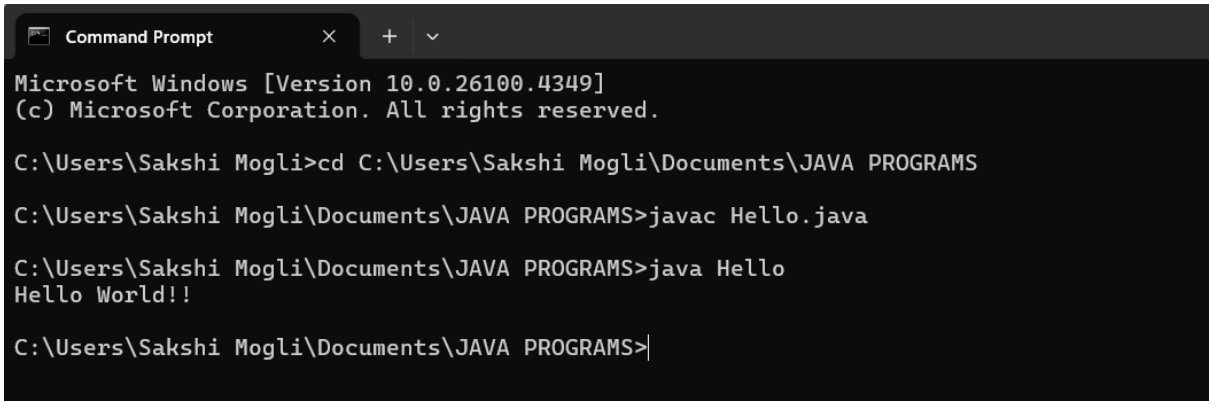
Ans: Java is a high-level, object-oriented programming language developed by **Sun Microsystems** in 1995 and now maintained by **Oracle Corporation**. It is widely used for building platform-independent applications, thanks to its **"Write Once, Run Anywhere"** capability. Java applications are compiled into bytecode, which runs on the **Java Virtual Machine (JVM)**, allowing the same code to run on different operating systems.

1. Features:
   Simple – Easy to learn and understand.
2. **Object-Oriented** – Everything is written using objects and classes.
3. **Platform Independent** – Java programs can run on any system (Windows, Mac, Linux) using JVM.
4. **Secure** – Java helps protect data and prevents unauthorized access.
5. **Robust** – Java handles errors well and avoids crashes.
6. **Multithreaded** – Java can do many tasks at the same time.
7. **High Performance** – Java is fast because of its special compiler called JIT.
8. **Distributed** – Java programs can work on networks and communicate with other systems.
9. **Dynamic** – Java can load and use new code while the program is running.

2. Explain the Java program execution process.

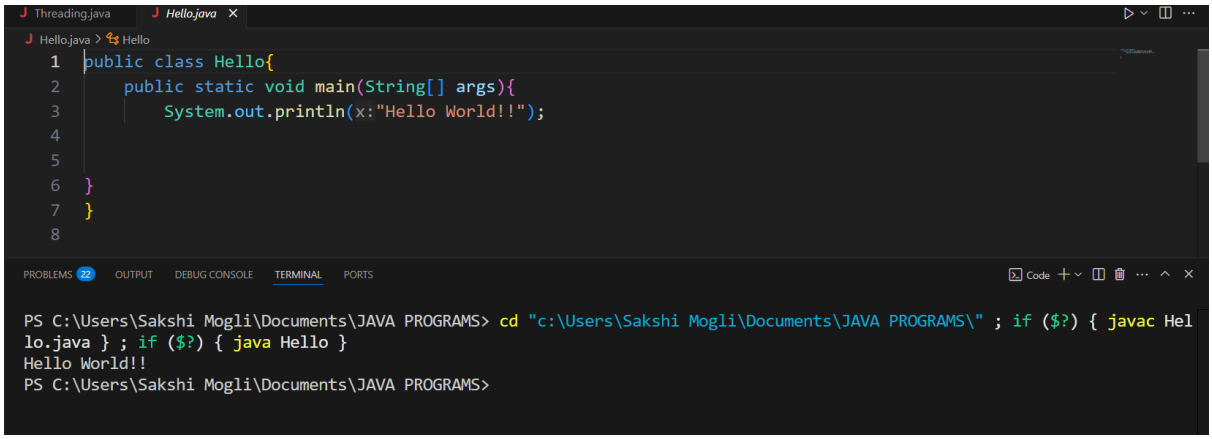
Ans: First we have to install JDK for executing java programs and after that we also need to set the environment variables for the system. Java program can be executed with the help of Notepad by saving the file with the extension filename.java. Then we have to open the command prompt and paste the path of the folder using **cd** command, next we need to compile the program file using the command **javac filename.java,** after the compilation of the file execute the code using the command **java filename.**

Java Source Code (.java)
↓
Java Compiler
↓
Bytecode (.class file)
↓
Java Virtual Machine (JVM)
↓
Machine Code (Output on screen)



3. Write a simple Java program to display 'Hello World'.

Ans:



4. What are data types in Java? List and explain them.

Ans: In Java, data types tell the compiler what type of data a variable will store. They help in declaring variables, allocating memory, and ensuring type safety. There are two types of Datatypes in Java:

Primitive and Nonprimitive

In primitive datatypes we have:

- **Numeric:** byte, short, int, long, float (for single precision decimal nos.), double (for double precision decimal nos.)
- **Non-numeric:** char (to store a single character or letter), boolean (true/false)

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

Ans: In Java, **JDK**, **JRE**, and **JVM** are essential components for writing, running, and executing Java programs. Here's the difference between them:

1. 🔹 **JVM (Java Virtual Machine):**
   - It is the **engine** that runs Java bytecode.
   - JVM reads .class files (bytecode) and converts them into machine code for the specific operating system.
   - It is **platform-dependent**, but the bytecode is platform-independent.
   - **Role:** Executes Java programs
2. 🔹 **JRE (Java Runtime Environment):**
   - It includes the **JVM** + **libraries** and **class files** required to run Java applications.
   - It **does not contain tools to develop** Java programs.
   - **Role:** Allows running Java programs, but not developing them.
3. 🔹 **JDK (Java Development Kit):**
   - It includes the **JRE** + **development tools** like the compiler (javac), debugger, etc.
   - Required for **writing, compiling, and running** Java programs.
   - **Role:** Complete package for Java developers.

6. What are variables in Java? Explain with examples.

Ans: In Java, a **variable** is a name used to store a **value** in memory. It helps in storing data that can be used and changed during program execution. Every variable has a **data type** that defines the kind of value it can hold.

Types of Variables:

1. **Local Variable** – Declared inside a method.
2. **Instance Variable** – Declared inside a class but outside methods.
3. **Static Variable** – Declared with static keyword and shared by all objects.

Example:

```
static String collegeName = "ABC College";
String name = "Sakshi";
int number = 5;
float num = 4.5f;
char ch = ‘s’;
```

7. What are the different types of operators in Java?

Ans: In Java, **operators** are symbols used to perform **operations on variables and values** (e.g., addition, comparison).

Types of Operators in Java:

- **Arithmetic Operators** – Used for mathematical operations.
  +, -, *, /, %
  Example: a + b, x * y

- **Relational (Comparison) Operators** – Compare two values.
  ==, !=, >, <, >=, <=
  Example: a > b, x == y
- **Logical Operators** – Used for logical (boolean) operations.
  && (AND), || (OR), ! (NOT)
  Example: a > b && b > c
- **Assignment Operators** – Assign values to variables.
  =, +=, -=, *=, /=, %=
  Example: a = 10, x += 5 (x = x+5)
- **Unary Operators** – Work with a single operand.
  +, -, ++ (increment) , -- (decrement), !
  Example: ++a, --b, !flag
- **Bitwise Operators** – Perform bit-level operations.
  & (and), |(or), ^ (ex-or), ~ (complement/negation), << (left shift), >> (right shift)
  Example: a & b, x << 2
- **Ternary Operator** – Short form of if-else.
  condition ? value_if_true : value_if_false
  Example: int max = (a > b) ? a : b;
- **Instanceof Operator** – Checks if an object is an instance of a class.
  Example: String obj = “Sakshi”;
  obj instanceof String;

8. Explain control statements in Java (if, if-else, switch).

Ans: **Control statements** in Java are used to **control the flow** of a program based on certain conditions. The most common control statements are if, if-else, and switch.

1. if Statement

The if statement checks a condition. If it is **true**, the code inside runs.

2. if-else Statement

It checks a condition. If it's **true**, one block runs; if **false**, another block runs.

3. switch Statement

The switch statement is used when we have **multiple conditions** to check based on one variable

Ifelse.java

```java
public class Ifelse{
    public static void main(String[] args){
        int age = 17;
        if(age>=18){
            System.out.println("Eligible to vote");
        }
        else{
            System.out.println("Not eligible to vote");
        }
    }
}
```

```
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS\"
lse.java } ; if ($?) { java Ifelse }
Not eligible to vote
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS>
```

Switch.java

```java
public class Switch{
    Run | Debug
    public static void main(String[] args){
        int month = 4;
        switch (month){
            case 1: System.out.println(x:"Jan");
            break;
            case 2: System.out.println(x:"Feb");
            break;
            case 3: System.out.println(x:"Mar");
            break;
            case 4: System.out.println(x:"Apr");
```

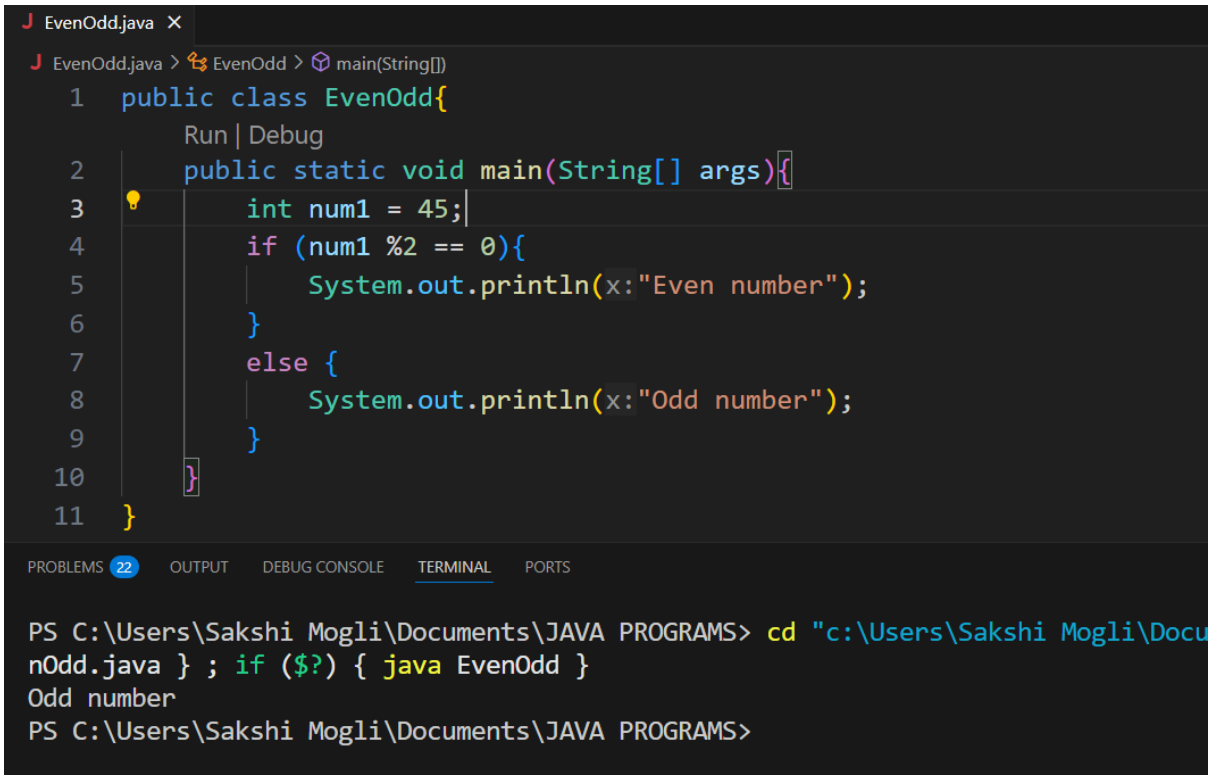
```
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli
tch.java } ; if ($?) { java Switch }
Apr
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS>
```

Switch.java

```java
public class Switch{
    public static void main(String[] args){
            case 4: System.out.println(x:"Apr");
            break;
            case 5: System.out.println(x:"May");
            break;
            case 6: System.out.println(x:"Jun");
            break;
            case 7: System.out.println(x:"Jul");
            break;
            case 8: System.out.println(x:"Aug");
            break;
            case 9: System.out.println(x:"Sep");
            break;
            case 10: System.out.println(x:"Oct");
            break;
            case 11: System.out.println(x:"Nov");
            break;
            case 12: System.out.println(x:"Dec");
            break;
            default: System.out.println(x:"Not a valid month");
            break;
        }
    }
}
```

9.Write a Java program to find whether a number is even or odd.

Ans:



10.What is the difference between while and do-while loop?

Ans: Both while and do-while are **looping statements** in Java that repeat a block of code, but there is a key difference in how they check the condition.

while Loop:

- The condition is **checked before** the loop body runs.
- If the condition is **false at the beginning**, the loop **may not run at all**.

do-while Loop:

- The loop body runs **at least once**, and then the condition is checked.
- If the condition is **false**, the loop stops after the first run.

EvenOdd.java | Whileloop.java

Whileloop.java > Whileloop > main(String[])

```java
public class Whileloop {
    Run | Debug
    public static void main(String[] args){
        int i = 1;
        while(i <= 30){
            if(i %2 != 0){
                System.out.println(i);
            }
            i++;
        }
    }
}
```

PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli\Documents\JAVA PROGRAM
leloop.java } ; if ($?) { java Whileloop }
1
3
5
7
9
11
13
15
17
19
21
23
```

Dowhileloop.java

Dowhileloop.java > Dowhileloop > main(String[])

```java
public class Dowhileloop {
    Run | Debug
    public static void main(String[] args){
        int i = 1;
        do{
            System.out.println("Count: "+i);
            i++;
        }while(i <= 10);
    }
}
```

PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli\Documents\JA
hileloop.java } ; if ($?) { java Dowhileloop }
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
Count: 6
Count: 7
Count: 8
Count: 9
Count: 10
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS>
```

2. Object-Oriented Programming (OOPs)

1. What are the main principles of OOPs in Java? Explain each.

Ans: **OOP (Object-Oriented Programming)** is a programming approach that organizes code into **objects**, which are created from **classes**. Java follows the OOP model completely, which helps in writing modular, reusable, and efficient programs.

- **Class:** A blueprint or template for creating objects. It defines properties (variables) and behaviors (methods).
  Example: class Car { String color; void drive() {
  } }
- **Object:** An instance of a class. It represents a real-world entity and uses the class's variables and methods.
  Example: Car myCar = new Car();

Encapsulation

- **Meaning:** Wrapping data (variables) and methods into a single unit (class).
- **Purpose:** To protect data from direct access using private access modifier, and provide access using get and set methods.

Inheritance

- **Meaning:** A class (child/subclass) can inherit fields and methods from another class (parent/superclass).
- **Purpose:** Reuse existing code and add more features.

Polymorphism

- **Meaning:** One thing has **many forms**.
- **Types:**
  - **Compile-time polymorphism:** Method overloading (same method name, different parameters).
  - **Runtime polymorphism:** Method overriding (child class provides its own version of the method).

Abstraction

- **Meaning:** Hiding unnecessary details and showing only the important parts.
- **How:** Using abstract classes or interfaces.

2. What is a class and an object in Java? Give examples.

Ans: **Class:**

- A **class** is a **blueprint** or **template** for creating objects.
- It defines **variables (attributes)** and **methods (functions)** that describe the behavior of the object.

Example:

```
class Car {
String color;
  void drive() {
  System.out.println("Car is driving");
 }
}
```

4. **Object:**
   - An **object** is an **instance of a class**.
   - It is a real-world entity created from a class and can access its methods and variables.

   Example:

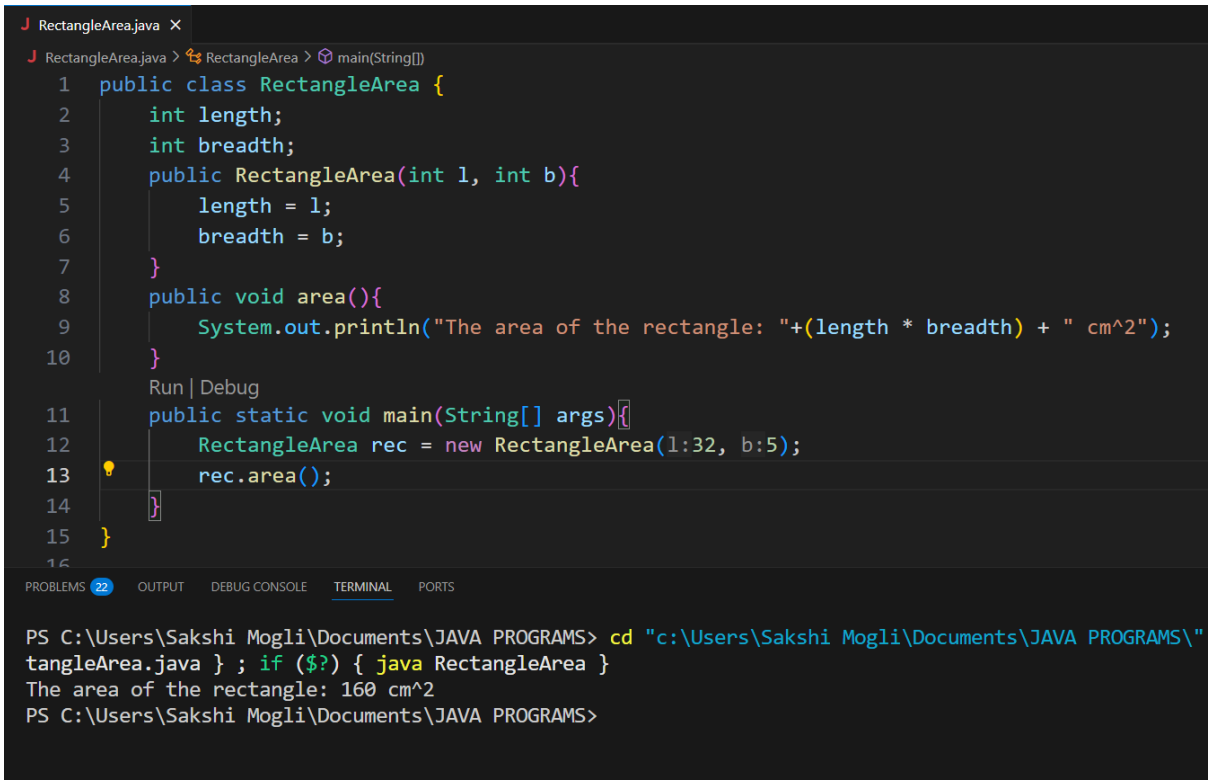
```
public class Main {
   public static void main(String[] args) {
      Car myCar = new Car();      // Creating an object of class Car
      myCar.color = "Red";
      myCar.drive();              // Calling method
   }
}
```

   Output:

   Car is driving

3. Write a program using class and object to calculate area of a rectangle.

Ans:



4. Explain inheritance with real-life example and Java code.

Ans: **Inheritance** in Java is a feature that allows one class (called the **child** or **subclass**) to **inherit** the properties and behaviors (variables and methods) of another class (called the **parent** or **superclass**).

It supports **code reusability**, reduces redundancy, and promotes a hierarchical relationship.

Real-life example:

- An **Appliance** is a general machine that has basic features like turning **on** and **off**.
- A **WashingMachine** is a **type of Appliance**, but it also has specific features like **wash clothes**.

So, WashingMachine can **inherit** the common features of Appliance and also have its own feature.

RectangleArea.java | InhExample.java

InhExample.java > InhExample > main(String[])

```java
// Parent class
class Appliance {
    void turnOn() {
        System.out.println(x:"Appliance is now ON");
    }

    void turnOff() {
        System.out.println(x:"Appliance is now OFF");
    }
}

// Child class
class WashingMachine extends Appliance {
    void washClothes() {
        System.out.println(x:"Washing clothes...");
    }
}

public class InhExample {
    Run | Debug
    public static void main(String[] args) {
        WashingMachine wm = new WashingMachine();
        wm.turnOn();        // Inherited from Appliance
        wm.washClothes();   // Specific to WashingMachine
        wm.turnOff();       // Inherited from Appliance
    }
```

PROBLEMS 22 | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS

```
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS\"
Example.java } ; if ($?) { java InhExample }
Appliance is now ON
Washing clothes...
Appliance is now OFF
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS>
```

5. What is polymorphism? Explain with compile-time and runtime examples.

Ans: **Polymorphism** means "**many forms**". In Java, polymorphism allows one action or method to behave **differently** based on the context. It improves **flexibility and reusability** in code.

Java supports **two types** of polymorphism:

1. Compile-Time Polymorphism (Method Overloading)

- Happens **during compilation**.
- Multiple methods have the **same name** but **different parameters** (type, number, or order).

2. Runtime Polymorphism (Method Overriding)

- Happens **during program execution**.
- A child class **overrides** a method from the parent class.
- The method that runs depends on the **object type** at runtime.

6. What is method overloading and method overriding? Show with examples.

Ans: **1. Method Overloading (Compile-Time Polymorphism)**

- **Definition:** Two or more methods in the **same class** with the **same name but different parameters** (number, type, or order).
- It is resolved **at compile time**.

MethodOverloading.java

```java
public class MethodOverloading {
    public int pro(int a, int b){
        return a * b;
    }
    public float pro(float a,float b){
        return a * b;
    }
    public int pro(int a,int b,int c){
        return a * b * c;
    }
    Run | Debug
    public static void main(String[] args){
        MethodOverloading myobj = new MethodOverloading();
        System.out.println("Multiplication using 2 int:"+(myobj.pro(a:4,b:5)));
        System.out.println("Multiplication using 2 float:"+(myobj.pro(a:23.98f,b:9.7f)));
        System.out.println("Multiplication using 3 int:"+(myobj.pro(a:7,b:8,c:13)));
    }
}
```

```
PROBLEMS 22  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS\"
hodOverloading.java } ; if ($?) { java MethodOverloading }
Multiplication using 2 int:20
Multiplication using 2 float:232.60599
Multiplication using 3 int:728
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS>
```

2. Method Overriding (Runtime Polymorphism)

- **Definition:** A method in a **child class** has the **same name, return type, and parameters** as a method in the **parent class**.
- It is resolved **at runtime**.

PolyDemo.java

```java
class Mydemo{
    public void display(){
        System.out.println(x:"This is mydemo method");
    }
}
class NextDemo extends Mydemo{
    public void display(){
        System.out.println(x:"This is nextdemo method");
    }
}
public class PolyDemo {
    Run | Debug
    public static void main(String[] args){
        Mydemo obj1 = new NextDemo(); //Overriding of method in child class
        obj1.display();
        Mydemo obj2 = new Mydemo();
        obj2.display();
    }
}

```

PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS\"
yDemo.java } ; if ($?) { java PolyDemo }
This is nextdemo method
This is mydemo method
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS>
```

7.What is encapsulation? Write a program demonstrating encapsulation.

Ans: **Encapsulation** is one of the main principles of **Object-Oriented Programming (OOP)**. It means **binding data (variables) and code (methods)** into a single unit, usually a **class**, and **restricting direct access** to the data from outside the class.

Encapsulation is implemented using:

- **Private** variables (cannot be accessed directly from outside).
- **Public** getter and setter methods to read/write those variables.

Advantages of Encapsulation:

- Protects data from unauthorized access.
- Makes the class easier to maintain and modify.
- Improves security and control.

Encapsulation.java

```java
public class Encapsulation {
    private String name;
    private int age;
    public String getname(){
        return name;
    }
    public void setname(String name){
        this.name = name;
    }
    public int getage(){
        return age;
    }
    public void setage(int age){
        this.age = age;
    }
    Run | Debug
    public static void main(String[] args){
        Encapsulation s1 = new Encapsulation();
        s1.setname(name:"Sakshi");
        s1.setage(age:19);
        System.out.println("Name: "+(s1.getname()));
        System.out.println("Age: "+(s1.getage()));
    }
}
```

```
PROBLEMS 22  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS\"
apsulation.java } ; if ($?) { java Encapsulation }
Name: Sakshi
Age: 19
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS>
```

8. What is abstraction in Java? How is it achieved?

Ans: **Abstraction** is a concept in Java where we **hide complex implementation details** and **show only the necessary features** of an object.

It helps to reduce complexity and focus only on **what an object does**, not **how it does it**.

In Java, abstraction is achieved using:

1. **Abstract Classes**
2. **Interfaces**

1. Using Abstract Class:

- An **abstract class** cannot be instantiated.
- It can have **abstract methods** (without body) and **regular methods** (with body).
- The subclass must provide implementations for the abstract methods.

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

        void eat() {
           System.out.println("Animal eats food");
        }
     }

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

  public class Main {
    public static void main(String[] args) {
           Dog d = new Dog();
           d.sound();   // Output: Dog barks
           d.eat();     // Output: Animal eats food
        }
}
```

2. Using Interface:

- An interface is a fully abstract class (only method declarations, no body).
- A class uses implements keyword to provide method definitions.

```
     interface Vehicle {
        void start();   // abstract method
     }

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

9. Explain the difference between abstract class and interface.

Ans: **Abstract Class:**

- Declared using the abstract keyword.
- Can have **abstract methods** (no body) and **concrete methods** (with body).
- Can declare **instance variables** (non-static, non-final).
- Supports **single inheritance** only.
- Can have **constructors**.
- Can use various **access modifiers** (public, private, protected).
- A class **extends** an abstract class to inherit from it.
- Suitable when you need **partial abstraction** and **shared code**.

Interface:

- Declared using the interface keyword.
- All methods are **abstract by default** (until Java 7). From Java 8+, can have default and static methods.
- Can only have **public static final** (constant) variables.
- Supports **multiple inheritance** (a class can implement multiple interfaces).
- **Cannot have constructors**.
- All methods are **public by default**.
- A class **implements** an interface to follow it.
- Suitable when you need **full abstraction** and a **common contract** for multiple classes.

10. Create a Java program to demonstrate the use of interface.

Ans:

HybridInheritance.java | Interface.java

Interface.java > Interface > main(String[])

```java
interface Printable {
    void print();  // abstract method
}

// Class implementing the interface
class Document implements Printable {
    public void print() {
        System.out.println(x:"Printing a document...");
    }
}

// Another class implementing the same interface
class Photo implements Printable {
    public void print() {
        System.out.println(x:"Printing a photo...");
    }
}

public class Interface {
    Run | Debug
    public static void main(String[] args) {
        Printable doc = new Document();  // Interface reference
        Printable pic = new Photo();     // Interface reference

        doc.print();   // Output: Printing a document...
        pic.print();   // Output: Printing a photo...
```

PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS> cd "c:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS\"
erface.java } ; if ($?) { java Interface }
Printing a document...
Printing a photo...
PS C:\Users\Sakshi Mogli\Documents\JAVA PROGRAMS>
```