Java Basics & OOPs Assignment Questions

NAME : *REVATI RAJENDRA BHOSALE*

BATCH : ANP-D1544

STUDENT CODE : AF04953344

COLLEGE : MIT ACADEMY OF ENGINEERING , ALANDI

1. Java Basics

1. What is Java? Explain its features.

Ans :

Java is a high-level, object-oriented programming language known for its platform independence and security.

Key Features (Short Version):

- **Platform Independent:** Runs on any OS via JVM
- **Object-Oriented:** Uses classes and objects
- **Simple & Secure:** Easy syntax and built-in security
- **Robust:** Strong error handling
- **Multithreaded:** Supports parallel tasks
- **High Performance:** Uses JIT compiler
- **Distributed & Dynamic:** Supports networking and runtime class loading

2. Explain the Java program execution process.

Ans:

1. **Write the Code**

- Open Notepad and write your Java code.
- Save the file as hello.java

2. **Compile the Code**

- Open **Command Prompt (cmd)**.
- Navigate to the folder where Hello.java is saved using cd command.
- USE Command as Javac hello.java
- This creates `Hello.class` (bytecode file).

3. **Run the Program**

- Run the compiled class using the Java interpreter:
- Use command as java Hello

```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Documents\java Practice>javac hello.java

C:\Users\ASUS\OneDrive\Documents\java Practice>java hello
Hello Guyzzz !! , I am Revati
```

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

```
public class Greeting {

    Run | Debug
    public static void main(String[] args) {
        System.out.println(x:"Hello World !!!!");
    }
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\java Practice>  & 'C:\Pro
eDetailsInExceptionMessages' '-cp' 'C:\Users\ASUS\AppData\Roa
at.java\jdt_ws\java Practice_f1b49777\bin' 'Greeting'
Hello World !!!!
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

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

   In Java, **data types** define the type of data a variable can hold. They ensure proper use of memory and help the compiler catch errors early.

1. Primitive Data Types

These are the basic, built-in types. Java has 8 primitives:

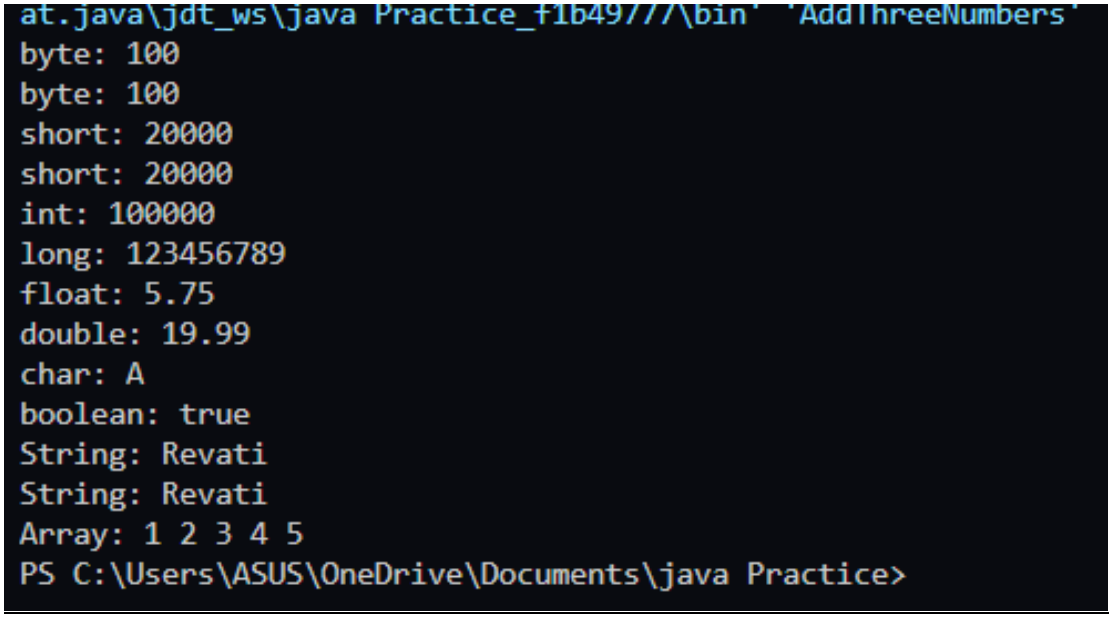
- **byte**: Small integers, saves memory. ……Range: -128 to 127.
- **short**: Slightly larger integers. ……Range: -32,768 to 32,767.
- **int**: Default for integers……..Range: about ±2 billion.
- **long**: Used for very large integers…….. Ends with L.
- **float**: Decimal numbers (less precision)…….. Ends with f.
- **double**: Default for decimal numbers (more precision).
- **char**: Stores a single character (e.g., 'A', '5').
- **boolean**: Stores true or false values.

2. Non-Primitive (Reference) Data Types

These are more complex types that refer to objects in memory:

- **String** – Stores a sequence of characters.
- **Arrays** – Stores multiple values of the same type.
- **Classes/Objects** – User-defined data types with methods and properties.
- **Interfaces** – Used to achieve abstraction and multiple inheritance.

```java
public class AddThreeNumbers {

    Run | Debug
    public static void main(String[] args) {

        // Primitive data types
        byte byteValue = 100;
        short shortValue = 20000;
        int intValue = 100000;
        long longValue = 123456789L;
        float floatValue = 5.75f;
        double doubleValue = 19.99;
        char charValue = 'A';
        boolean booleanValue = true;

        // Non-primitive data types
        String stringValue = "Revati";
        int[] numbers = {1, 2, 3, 4, 5};

        // Print all values
        System.out.println("byte: " + byteValue);
        System.out.println("short: " + shortValue);
        System.out.println("int: " + intValue);
        System.out.println("long: " + longValue);
        System.out.println("float: " + floatValue);
        System.out.println("double: " + doubleValue);
        System.out.println("char: " + charValue);
        System.out.println("boolean: " + booleanValue);
        System.out.println("String: " + stringValue);
        System.out.print(s:"Array: ");
        for (int num : numbers) {
            System.out.print(num + " ");
        }
    }
}
```



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

Ans:-

1. JVM (Java Virtual Machine)

- It runs the **compiled Java bytecode** (.class files).
- JVM makes Java **platform-independent** (same code runs on Windows, Linux, etc.).
- It handles memory, garbage collection, and execution.

Java program runner.

2. JRE (Java Runtime Environment)

- It contains the **JVM + required libraries** to run Java programs.
- You **can run** Java applications but **cannot develop or compile** code with it.

Ready-to-run Java setup.

3. JDK (Java Development Kit)

- It includes **JRE + development tools** like javac (compiler), debugger, etc.
- You need JDK to **write, compile, and run** Java programs.

Full Java software development kit.

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

Ans:-

In Java, a **variable** is used to store data that your program can use and modify. Each variable has:

- A **name**
- A **data type**
- A **value**

Types of Variables:

1. **Local Variable**
   - Declared inside a method or block.
   - Exists only during method execution.
   - Must be initialized before use.
2. **Instance Variable**
   - Declared inside a class, but outside methods.
   - Each object gets its own copy.
3. **Static Variable .**
   - Declared with `static` keyword.
   - Shared across all objects of the class.

```java
public class AddThreeNumbers {
    int age = 25; // Instance variable
    static String company = "Apollo"; // Static variable

    public void display() {
        String name = "Revati"; // Local variable
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Company: " + company);
    }

    Run | Debug
    public static void main(String[] args) {
        AddThreeNumbers obj = new AddThreeNumbers();
        obj.display();
    }
}
```

```
er\workspaceStorage\f10234dea73844daFc93bce2eb7dbeb5\redha
Name: Revati
Age: 25
Company: Apollo
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

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

Ans:-

1. Arithmetic Operators

Used to perform basic math operations like addition, subtraction, multiplication, division, and modulus.

- Examples: +, -, *, /, %
- Also includes increment (++) and decrement (--).
- **Example:** a + b, a++

2. Relational Operators

Used to compare two values and return true or false.

- Examples: == (equal), != (not equal), >, <, >=, <=
- **Example:** a > b returns true if a is greater than b.

3. Logical Operators

Used to combine multiple boolean expressions.

- Examples: && (AND), || (OR), ! (NOT)
- **Example:** (a > b) && (a < c) checks if both conditions are true.

4. Assignment Operators

- Used to assign values to variables and modify them.
- Examples: =, +=, -=, *=, /=, %=
- **Example:** c += 5 means c = c + 5.

5. Bitwise Operators

Operate on bits of integers. Useful for low-level programming.

- Examples: & (AND), | (OR), ^ (XOR), ~ (NOT), << (left shift), >> (right shift), >>> (unsigned right shift)
- **Example:** d & e performs AND on each bit of d and e

6. Ternary Operator

A shorthand for if-else statements.

- Syntax: condition ? valueIfTrue : valueIfFalse
- **Example:** int max = (a > b) ? a : b;

7. Instanceof Operator

Checks whether an object is an instance of a specific class or interface.

- Returns true or false.
- **Example:** str instanceof String returns true if str is a String object.

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

Ans:-

1. if Statement

Executes a block of code **only if** a specified condition is true.

```
if (condition) {
   // code to execute if condition is true
}
```

Example:

```
int num = 10;
if (num > 5) {
   System.out.println("Number is greater than 5");
}
```

2. if-else Statement

Chooses between two blocks: executes one if the condition is true, otherwise the other.

```
if (condition) {
  // if true
} else {
   // if false
}
```

Example:

```
int num = 3;
if (num % 2 == 0) {
   System.out.println("Even number");
} else {
   System.out.println("Odd number");
}
```

3. switch Statement

Selects one of many blocks to execute based on the value of a variable.

```
switch (variable) {
 case value1:
      // code block
      break;
   case value2:
      // code block
      break;
   default:
      // default code block
}
```

Example:

```
int day = 3;
switch (day) {
   case 1: System.out.println("Monday"); break;
```

```
   case 2: System.out.println("Tuesday"); break;
   case 3: System.out.println("Wednesday"); break;
   default: System.out.println("Other day");
}
```

```
System.out.println(x:"Enter The Number : ");
int day = sc.nextInt();
switch (day) {
    case 1:
        System.out.println(x:"Day is Monday");
        break;

    case 2:
        System.out.println(x:"Day is Tuesday");
        break;

    case 3:
        System.out.println(x:"Day is Wednesday");
        break;

    case 4:
        System.out.println(x:"Day is Thursday");
        break;

    case 5:
        System.out.println(x:"Day is Friday");
        break;

    case 6:
        System.out.println(x:"Day is Saturday");
        break;

    case 7:
        System.out.println(x:"Day is Sunday");
        break;

    default:
        System.out.println(x:"You Have Entered a wrong input ");
        break;
}
```

```
float tax = 0.0f;
System.out.println(x:"Enter your Income : ");
float income = sc.nextFloat();
if(income<=2.5f){
    tax = tax + 0;
    System.out.println(x:"No tax !!!");

}
else if(income >2.5f && income <=5.0f ){
    tax = tax + 0.05f * (income - 2.5f);
}
else if(income > 0.5f && income <=10.0f){
    tax = tax + 0.05f * (5.0f   - 2.5f);
    tax = tax + 0.2f * ( income - 5.0f);
}
else if(income > 10.0f){
     tax = tax + 0.05f * (10.0f   - 5.0f);
    tax = tax + 0.2f * (5.0f - 2.5f);
    tax = tax + 0.3f * (income - 10.0f);
}

System.out.println("Your Tax is " + tax);
```

5. Write a Java programm to find whether a number is even or odd.

   Ans:-

```
import java.util.Scanner;

public class AddThreeNumbers{
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print(s:"Enter a number: ");
        int num = scanner.nextInt();

        if (num % 2 == 0) {
            System.out.println(num + " is Even.");
        } else {
            System.out.println(num + " is Odd.");
        }

        scanner.close();
    }
}
```

```
er\workspaceStorage\f10234dea73844dafc93bce2eb7dbeb5\r
Enter a number: 5
5 is Odd.
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

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

Feature	While loop	Do-While
Condition Check	At the beginning of the loop	At the end of the loop
Execution	May not execute at all if condition is false	Executes at least once, even if condition is false

Syntax	while (condition) {
	...//code
	}
		do {
		...//code
		} while (condition);

EXAMPLE	int i = 5;
	while (i < 5) {
	
	System.out.println("Hello");
	// Won't run
	i++;
	}
		int i=5;
		do{
		System.out.println("Hello");
		i++;
		}while(i<5);

2. Object-Oriented Programming (OOPs)

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

Ans:-

Main Principles of OOPs in Java

Java is an Object-Oriented Programming (OOP) language, which is based on **4 main principles**:

1. Encapsulation

- Wrapping data (variables) and methods into a single unit (class).
- Helps protect data from unauthorized access.
- Achieved using **private** variables and **public** getters/setters.

Example :-

```
class Person {
  private String name;

  public void setName(String n) { name = n; }
  public String getName() { return name; }
}
```

2. Inheritance

- One class (child) inherits properties and methods from another class (parent).
- Promotes **code reusability**.

Example :-

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

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

3. Polymorphism

- One action behaves differently in different situations.
- Two types:
  - **Compile-time** (method overloading)
  - **Runtime** (method overriding)

Example :-

```
class Shape {
  void draw() { System.out.println("Drawing shape"); }
}

class Circle extends Shape {
  void draw() { System.out.println("Drawing circle"); } // Overriding
}
```

4. Abstraction

- Hiding internal details and showing only functionality.
- Achieved using **abstract classes** or **interfaces**.

Example :-

```
abstract class Vehicle {
  abstract void start();
}

class Car extends Vehicle {
  void start() { System.out.println("Car starts"); }
}
```

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 **properties (variables)** and **methods (functions)**.

Example :-

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

Object

An **object** is an instance of a class. It represents a real-world entity and uses the class's variables and methods.

Example :-

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

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

Ans:-

```
// Class to represent a Rectangle
class Rectangle {
    int length;
    int width;

    // Method to calculate area
    void calculateArea() {
        int area = length * width;
        System.out.println("Area of Rectangle: " + area);
    }
}

// Main class to run the program
public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Rectangle rect = new Rectangle();  // Creating object
        rect.length = 10;                  // Assigning values
        rect.width = 5;

        rect.calculateArea();              // Calling method
    }
}
```

```
er\workspaceStorage\f10234dea73844dafc93bce2eb7dbeb5\red
Area of Rectangle: 50
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

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

Ans : -

Inheritance in Java (with Real-Life Example)

So… What is Inheritance?

Inheritance allows one class (child/subclass) to inherit properties and methods of another class (parent/superclass).

It promotes code reuse and represents "is-a" relationship.

So…. Real-Life Example:

Class: Vehicle

- Common features: start(), stop()

Class: Car (inherits Vehicle)

- Additional feature: playMusic()

Here, a Car is a Vehicle → Car inherits Vehicle.

```java
// Parent class
class Vehicle {
    void start() {
        System.out.println(x:"Vehicle started");
    }

    void stop() {
        System.out.println(x:"Vehicle stopped");
    }
}

// Child class
class Car extends Vehicle {
    void playMusic() {
        System.out.println(x:"Playing music in the car");
    }
}

// Main class
public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Car myCar = new Car();

        myCar.start();       // Inherited method
        myCar.playMusic();   // Child class method
        myCar.stop();        // Inherited method
    }
}
```

```
e_f1b49777\bin' 'AddThreeNumbers'
Vehicle started
Playing music in the car
Vehicle stopped
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

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

Ans:-

Polymorphism means "many forms." It allows one interface (method name) to behave differently based on context.

Java supports two types:

1. Compile-Time Polymorphism (Method Overloading)

- Same method name, different parameters
- Decided during compilation

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}

public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Calculator c = new Calculator();
        System.out.println(c.add(a:2, b:3));
        System.out.println(c.add(a:2.5, b:3.5));
    }
}
```

```
5
6.0
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

2. Runtime Polymorphism (Method Overriding)

- Same method in parent and child class

- Decided during runtime using inheritance

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

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

public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Animal a = new Dog();  // Upcasting
        a.sound();             // Calls Dog's sound() at runtime
    }
}
```

```
dThreeNumbers'
Dog barks
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

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

Ans :-

Feature	Overloading	Overriding
Occurs in	Same class	Parent-child classes
Parameters	Must differ	Must be same
Return Type	Can be same or different	Must be same or covariant
Access Modifier	Can be anything	Cannot reduce visibility
Timing	Compile-time	Runtime

Method Overloading (Compile-Time Polymorphism)

- **Same method name**, but **different parameters** (type, number, or order)
- Happens **within the same class**
- Resolved at **compile-time**

```java
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}

public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Calculator c = new Calculator();
        System.out.println(c.add(a:2, b:3));
        System.out.println(c.add(a:2.5, b:3.5));
        System.out.println(c.add(a:1, b:2, c:3));
    }
}
```

```
dThreeNumbers'
5
6.0
6
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

Method Overriding (Runtime Polymorphism)

- Same method name and parameters, but **defined in child class**
- Must be in **inheritance (parent-child)** relationship
- Resolved at **runtime**

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

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

public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Animal a = new Dog();  // Upcasting
        a.sound();             // Calls Dog's version at runtime
    }
}
```

```
dThreeNumbers'
Dog barks
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

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

Ans:-

Encapsulation is an Object-Oriented Programming principle that bundles data (variables) and methods that operate on the data into a single unit (class). It also restricts direct access to some of an object's components, which helps protect the data from unauthorized access and modification.

- Achieved by using private variables and providing public getter and setter methods.
- Enhances data security and maintains integrity.

```java
class Student {
    // Private variables (data hidden)
    private String name;
    private int age;

    // Public getter method
    public String getName() {
        return name;
    }

    // Public setter method
    public void setName(String name) {
        this.name = name;
    }

    // Public getter method
    public int getAge() {
        return age;
    }

    // Public setter method with validation
    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        } else {
            System.out.println(x:"Age must be positive.");
        }
    }
}
```

```java
public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Student student = new Student();

        // Setting values using setters
        student.setName(name:"Revati");
        student.setAge(age:22);

        // Getting values using getters
        System.out.println("Name: " + student.getName());
        System.out.println("Age: " + student.getAge());

        // Trying to set invalid age
        student.setAge(-5);  // Will show error message
    }
}
```

```
Name: Revati
Age: 22
Age must be positive.
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

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

Ans :-

Abstraction is an OOP concept that hides complex implementation details and shows only the essential features of an object. It helps reduce programming complexity and increases efficiency by focusing on what an object does, not how it does it.

How is Abstraction Achieved in Java?

1. **Abstract Classes:**
    - Can have abstract methods (without body) and concrete methods (with body).
    - A subclass must provide implementation for abstract methods.

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

    void sleep() {
        System.out.println(x:"Animal is sleeping");
    }
}

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

public class AddThreeNumbers{
    Run | Debug
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.sound();  // Implemented method
        dog.sleep();  // Inherited concrete method
    }
}
```

```
dThreeNumbers'
Dog barks
Animal is sleeping
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

2. **Interfaces:**
    - Define method signatures without implementation.
    - A class implementing an interface must provide method bodies.

```java
interface Vehicle {
    void start();
    void stop();
}

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

    public void stop() {
        System.out.println(x:"Car stopped");
    }
}

public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Car car = new Car();
        car.start();
        car.stop();
    }
}
```

```
dThreeNumbers'
Car started
Car stopped
PS C:\Users\ASUS\OneDrive\Documents\java Practice>
```

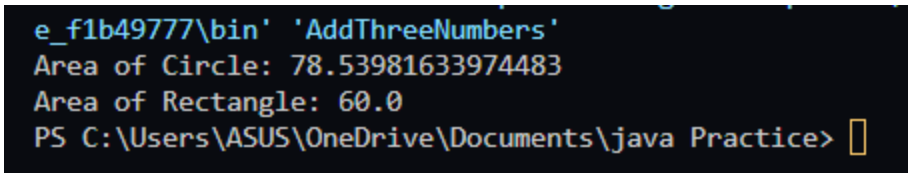
9. Explain the difference between abstract class and interface.
   Ans :-

Feature	Abstract Class	Interface
Purpose	To provide common base with some implementation	To define a contract with no/limited implementation
Method	Can have abstract and concrete methods	All methods are abstract (Java 8+ can have default and static methods)
Multiple Inheritance	Not supported (a class can extend only one abstract class)	Supported (a class can implement multiple interfaces)
Variables	Can have instance variables	Only static final (constant) variables
Access Modifiers	Can have any access modifier for methods and variables	Methods are implicitly public; variables are public static final
Constuctor	Can have constructors	Cannot have constructors
Use	Use when classes share code or common state	Use to define capabilities or contracts

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

Ans :-

```java
// Implementing the interface in Circle class
class Circle implements Shape {
    double radius = 5;

    public void calculateArea() {
        double area = Math.PI * radius * radius;
        System.out.println("Area of Circle: " + area);
    }
}

// Implementing the interface in Rectangle class
class Rectangle implements Shape {
    double length = 10;
    double width = 6;

    public void calculateArea() {
        double area = length * width;
        System.out.println("Area of Rectangle: " + area);
    }
}

// Main class
public class AddThreeNumbers {
    Run | Debug
    public static void main(String[] args) {
        Shape circle = new Circle();
        circle.calculateArea();

        Shape rectangle = new Rectangle();
        rectangle.calculateArea();
    }
}
```


e_f1b49777\bin' 'AddThreeNumbers'
Area of Circle: 78.53981633974483
Area of Rectangle: 60.0
PS C:\Users\ASUS\OneDrive\Documents\java Practice>