Java basics & OOPs Assignment

**NAME : DIKSHA GANGADHAR DHAKNE**

1)Java Basics

1. What is Java? Explain its features.

**Java** is a high-level, object-oriented, platform-independent programming language developed by Sun Microsystems (now owned by Oracle). It's widely used for building web, desktop, and mobile applications.

**Features of Java:**

* **Simple**: Easy to learn and use.
* **Object-Oriented**: Everything is an object.
* **Platform-Independent**: Write Once, Run Anywhere (WORA).
* **Secure**: Provides runtime security features.
* **Robust**: Handles errors through exception handling.
* **Multithreaded**: Supports multiple threads of execution.
* **High Performance**: Uses Just-In-Time (JIT) compiler.
* **Distributed**: Facilitates distributed computing.

1. Explain the Java program execution process.

**Writing Code**: Developer writes .java source code.

**Compilation**: The compiler converts .java to .class (bytecode).

**Class Loading**: Bytecode is loaded into JVM using ClassLoader.

**Bytecode Verification**: Verifier checks code for security.

**Execution**: The JVM executes the bytecode via the interpreter or JIT.

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

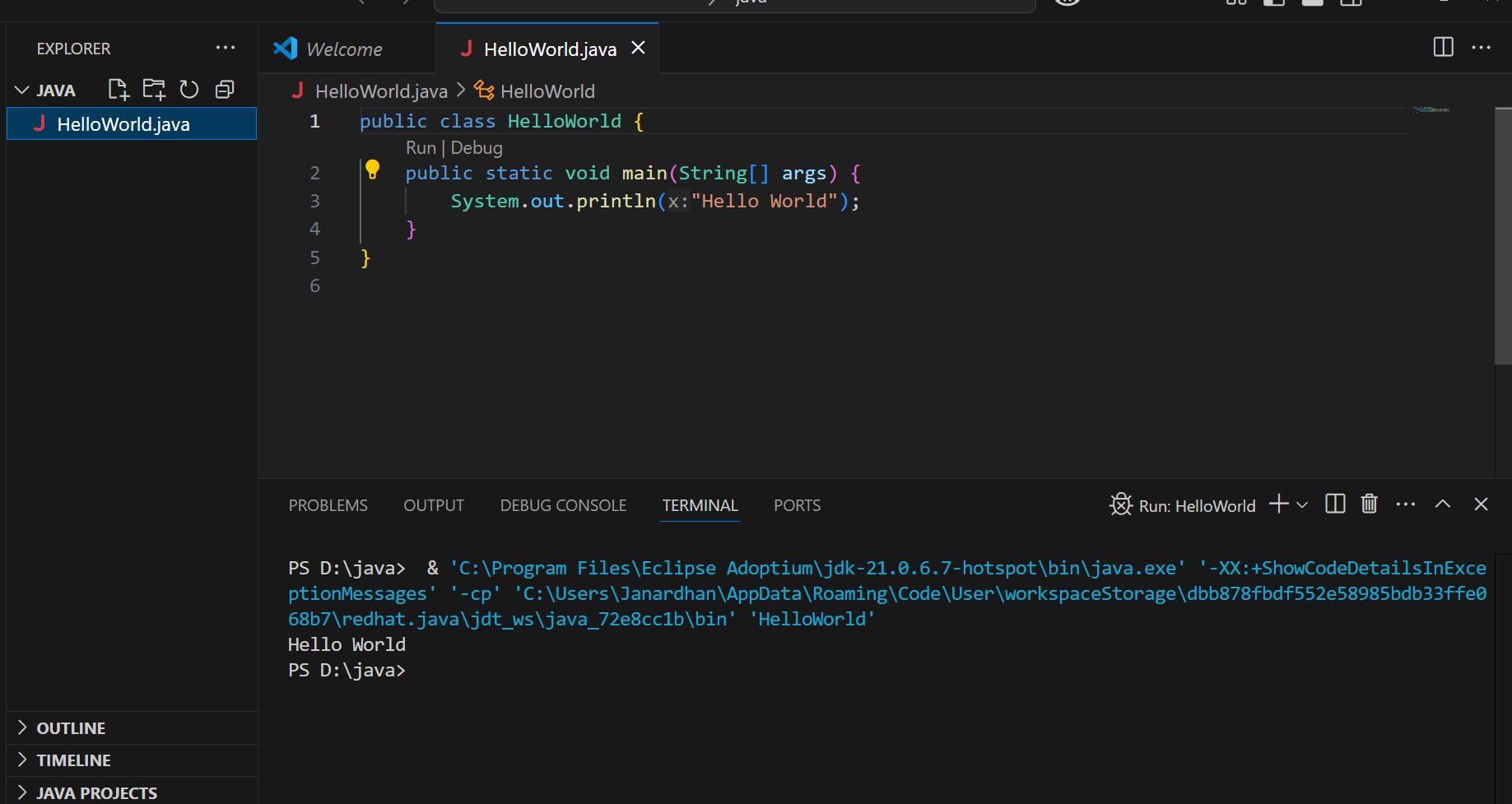
public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello World");

}

}



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

Java data types are classified into:

* **Primitive**:
  + Integer: byte, short, int, long
  + Floating: float, double
  + Character: char
  + Boolean: boolean
* **Non-Primitive**:
  + Arrays, Strings, Classes, Interfaces.

1. What is the difference between JDK, JRE, and JVM?
2. **JDK (Java Development Kit)**: Complete package to develop and run Java apps (includes JRE + tools).
3. **JRE (Java Runtime Environment)**: Runs Java applications (includes JVM + libraries).
4. **JVM (Java Virtual Machine)**: Executes bytecode.

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

Variables store data values.

int age = 21; // integer variable

String name = "Diksha"; // string variable

Types: local, instance, static.

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

1) **Arithmetic**: +, -, \*, /, %

2) **Relational**: ==, !=, >, <, >=, <=

3) **Logical**: &&, ||, !

4) **Assignment**: =, +=, -=, etc.

5) **Bitwise**: &, |, ^, ~, <<, >>

6) **Unary**: +, -, ++, --

7) **Ternary**: condition ? true : false

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

* **if**: Executes block if condition is true.

int age = 18;

if (age >= 18) {

System.out.println("Eligible to vote");

}

* **if-else**: Adds an alternative block.

int number = 5;

if (number % 2 == 0) {

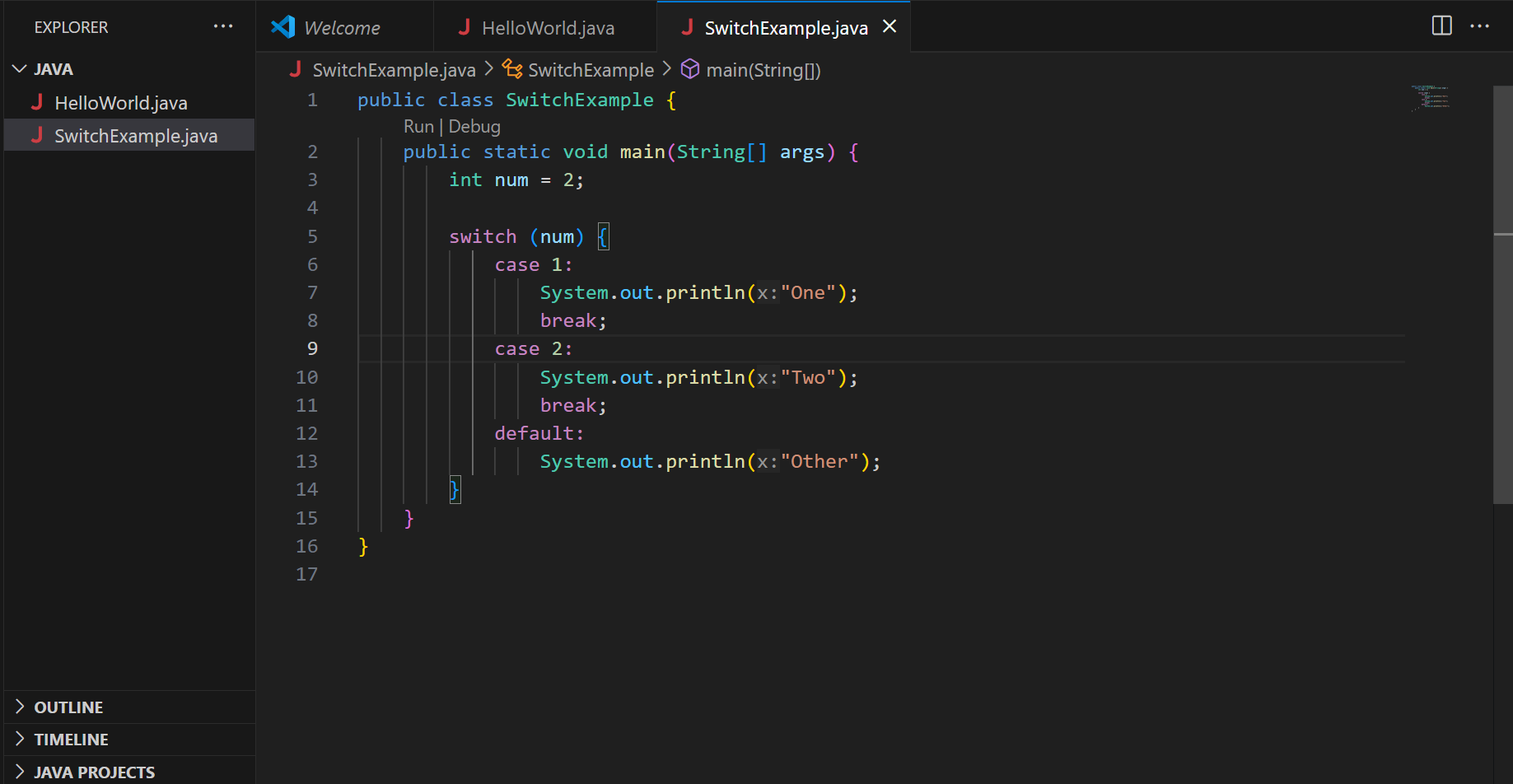
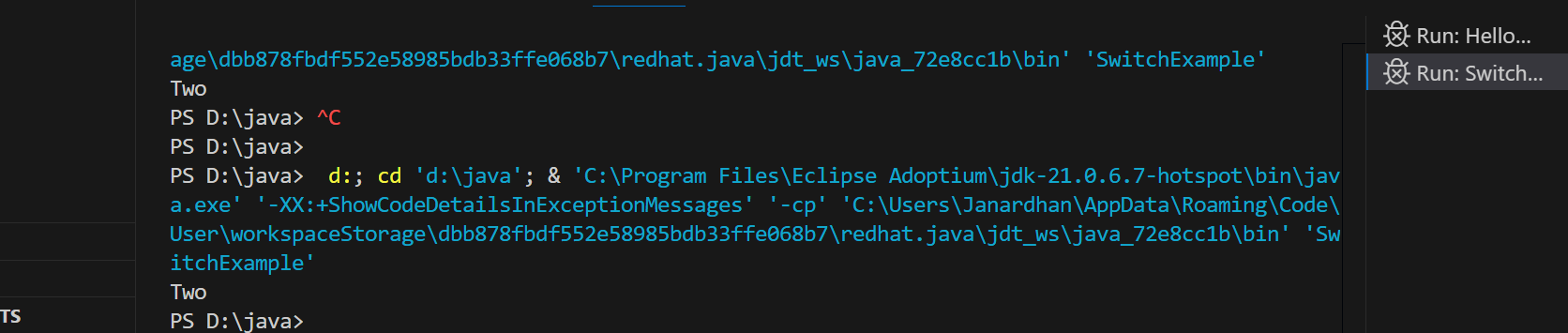
System.out.println("Even");

} else {

System.out.println("Odd");

}

* **switch**: Replaces multiple if-else for fixed values.



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

import java.util.Scanner;

public class EvenOdd {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in); // Create Scanner object

System.out.print("Enter a number: "); // Ask user for input

int num = sc.nextInt(); // Read the number

if (num % 2 == 0) {

System.out.println("Even");

} else {

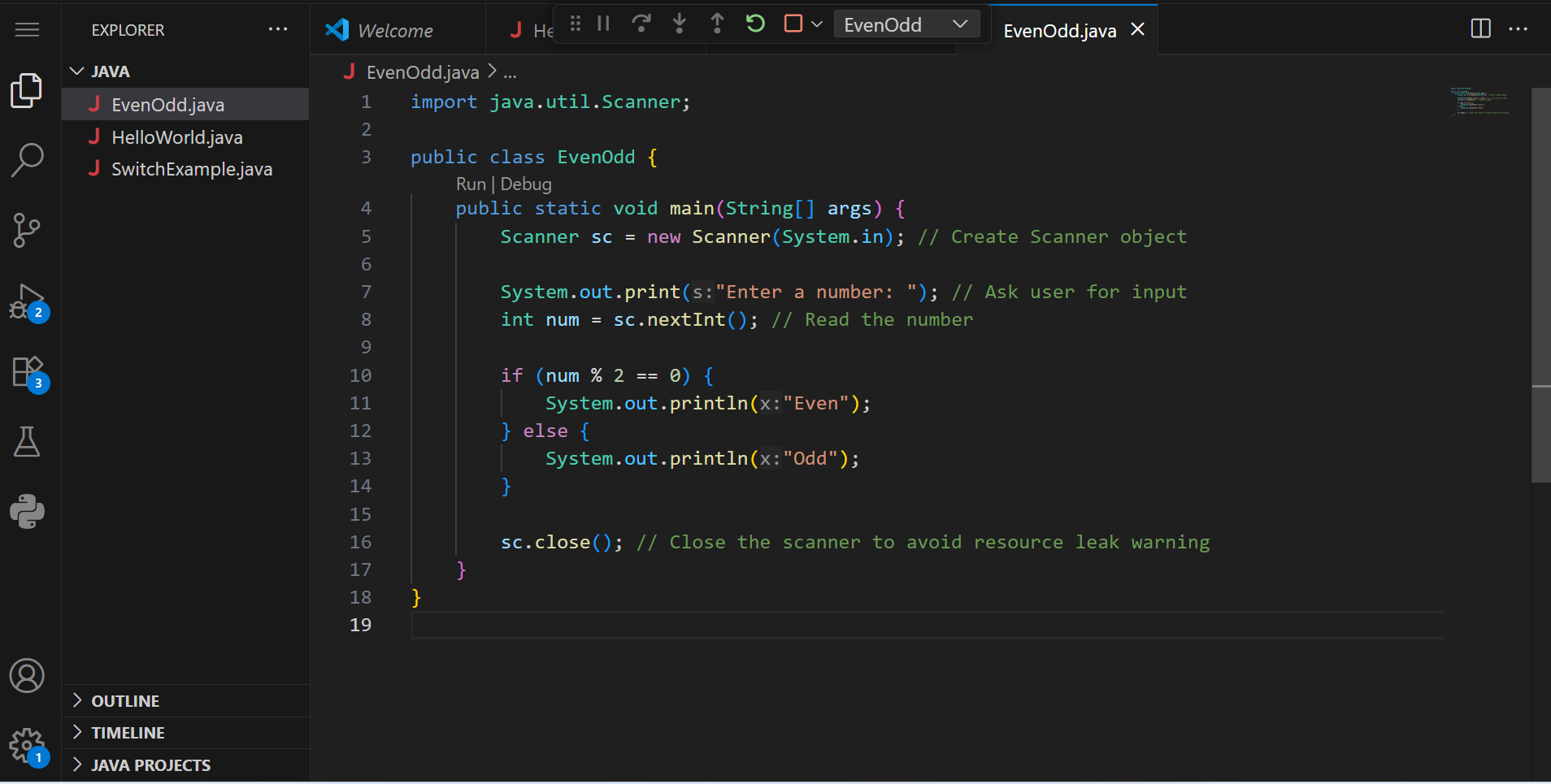
System.out.println("Odd");

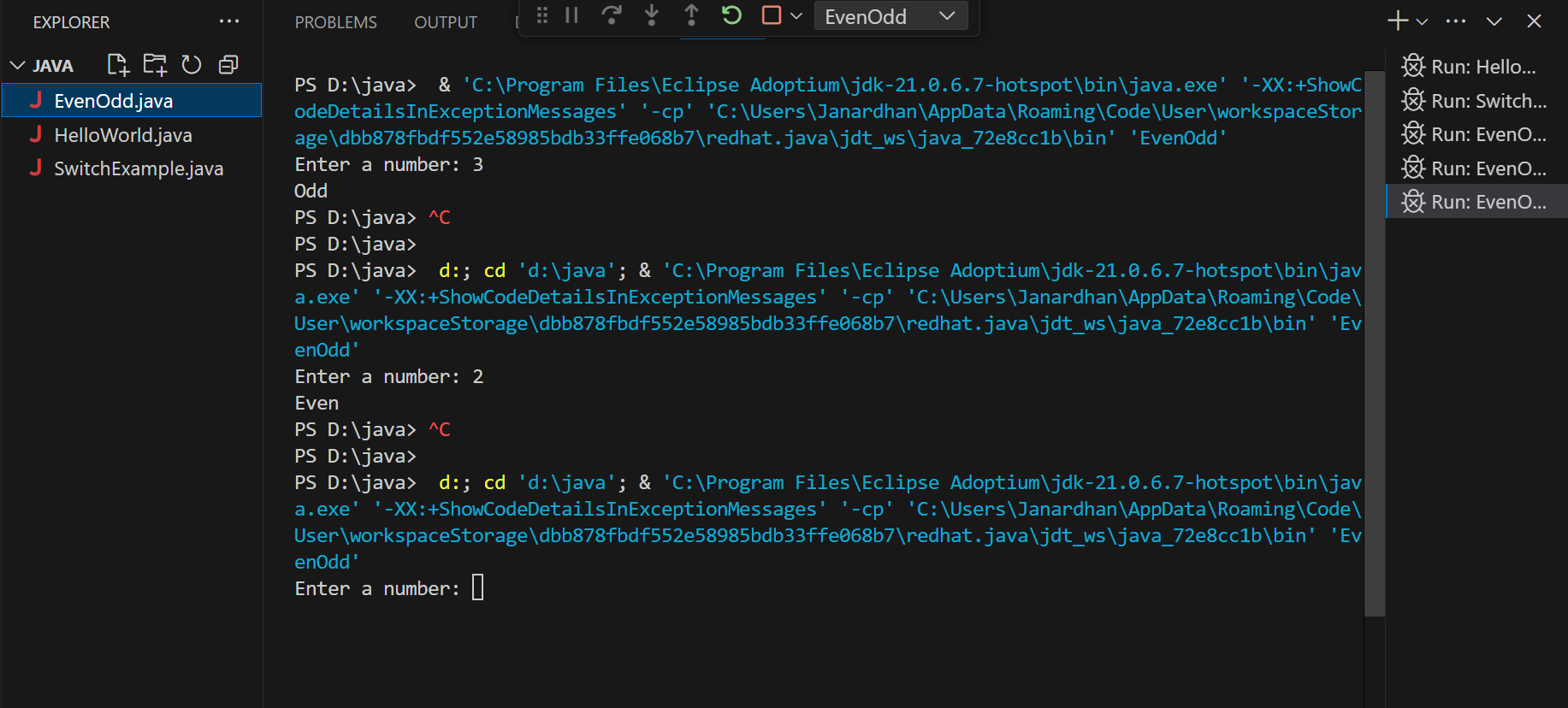
}

sc.close(); // Close the scanner to avoid resource leak warning

}

}





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

|  |  |
| --- | --- |
| **While Loop** | **Do-While Loop** |
| Condition checked first | Code runs first |
| May not execute at all | Executes at least once |

2) Object-Oriented Programming (OOPs)

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

* **Encapsulation**: Hiding internal data (e.g., using private variables).
* **Inheritance**: Acquiring properties from parent class.
* **Polymorphism**: One task, many forms (method overloading/overriding).
* **Abstraction**: Hiding complex implementation.

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

* **Class**: Blueprint/template.
* **Object**: Instance of a class

class Car {

String color = "Red";

}

public class Main {

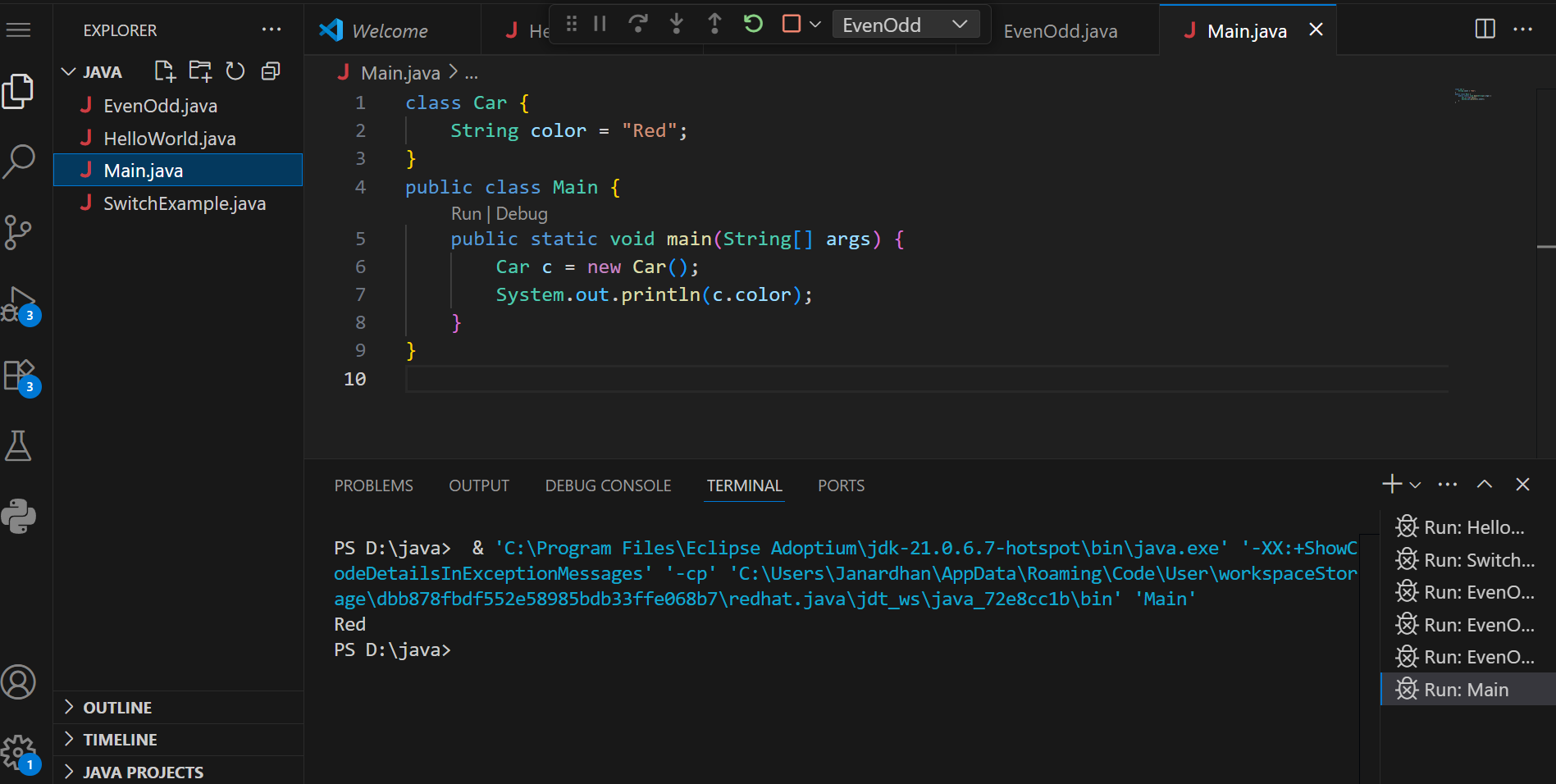
public static void main(String[] args) {

Car c = new Car();

System.out.println(c.color);

}

}



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

class Rectangle {

int length, width;

void insert(int l, int w) {

length = l;

width = w;

}

void calculateArea() {

System.out.println("Area: " + (length \* width));

}

}

public class TestRectangle {

public static void main(String[] args) {

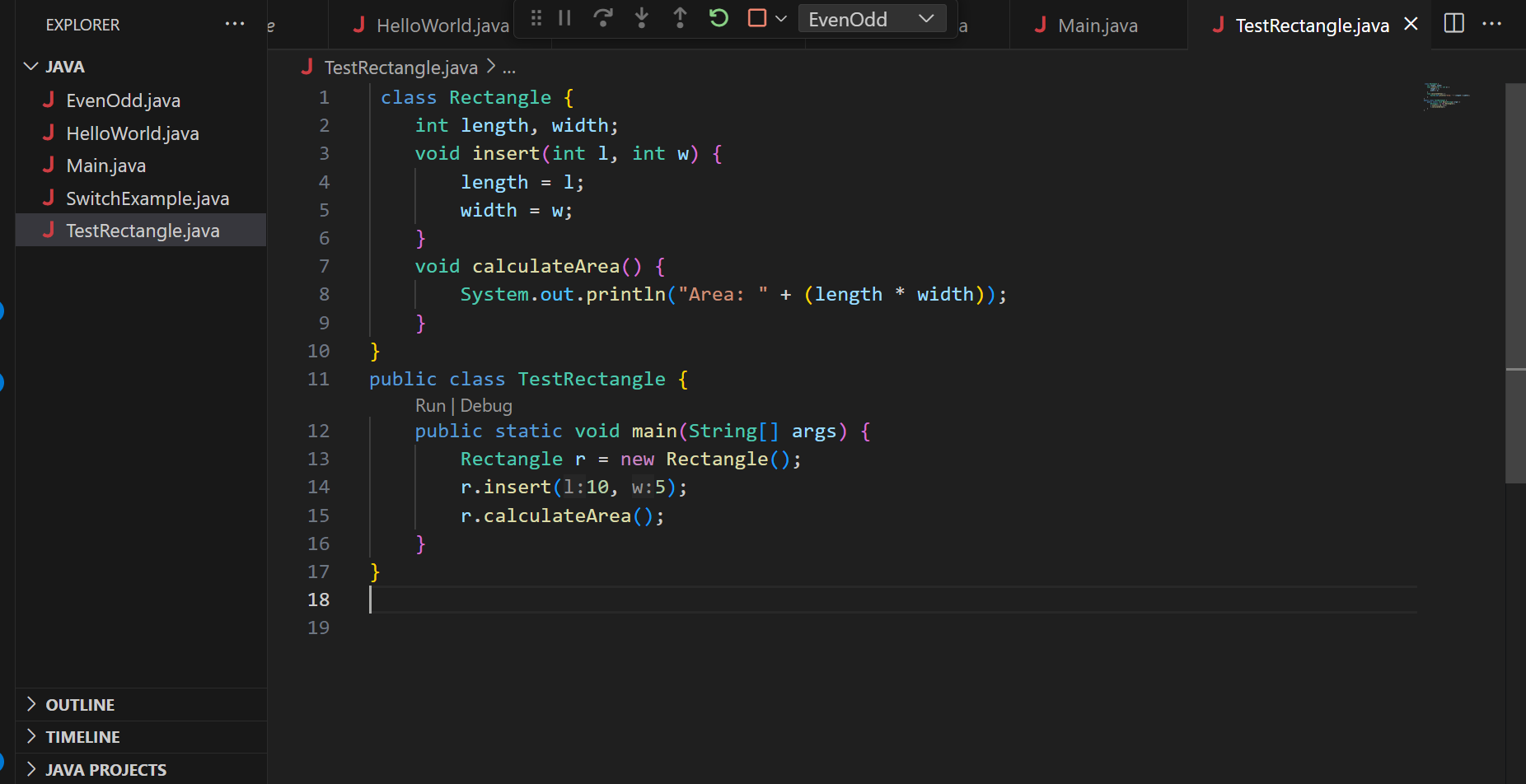
Rectangle r = new Rectangle();

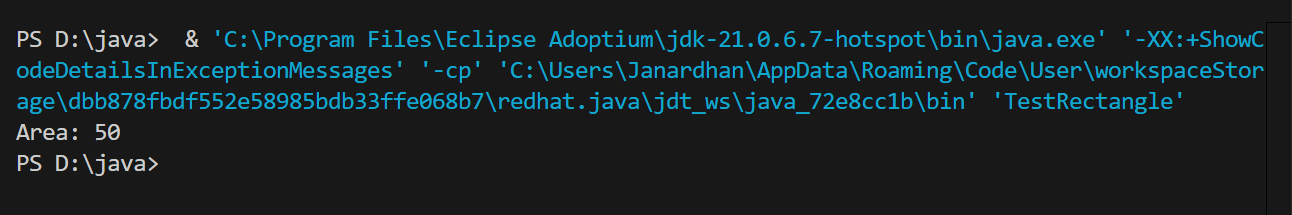
r.insert(10, 5);

r.calculateArea();

}

}





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

**Example**: A dog is an animal. Dog inherits from Animal.

class Animal {

void sound() {

System.out.println("Animal makes sound");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog barks");

}

}

public class Test {

public static void main(String[] args) {

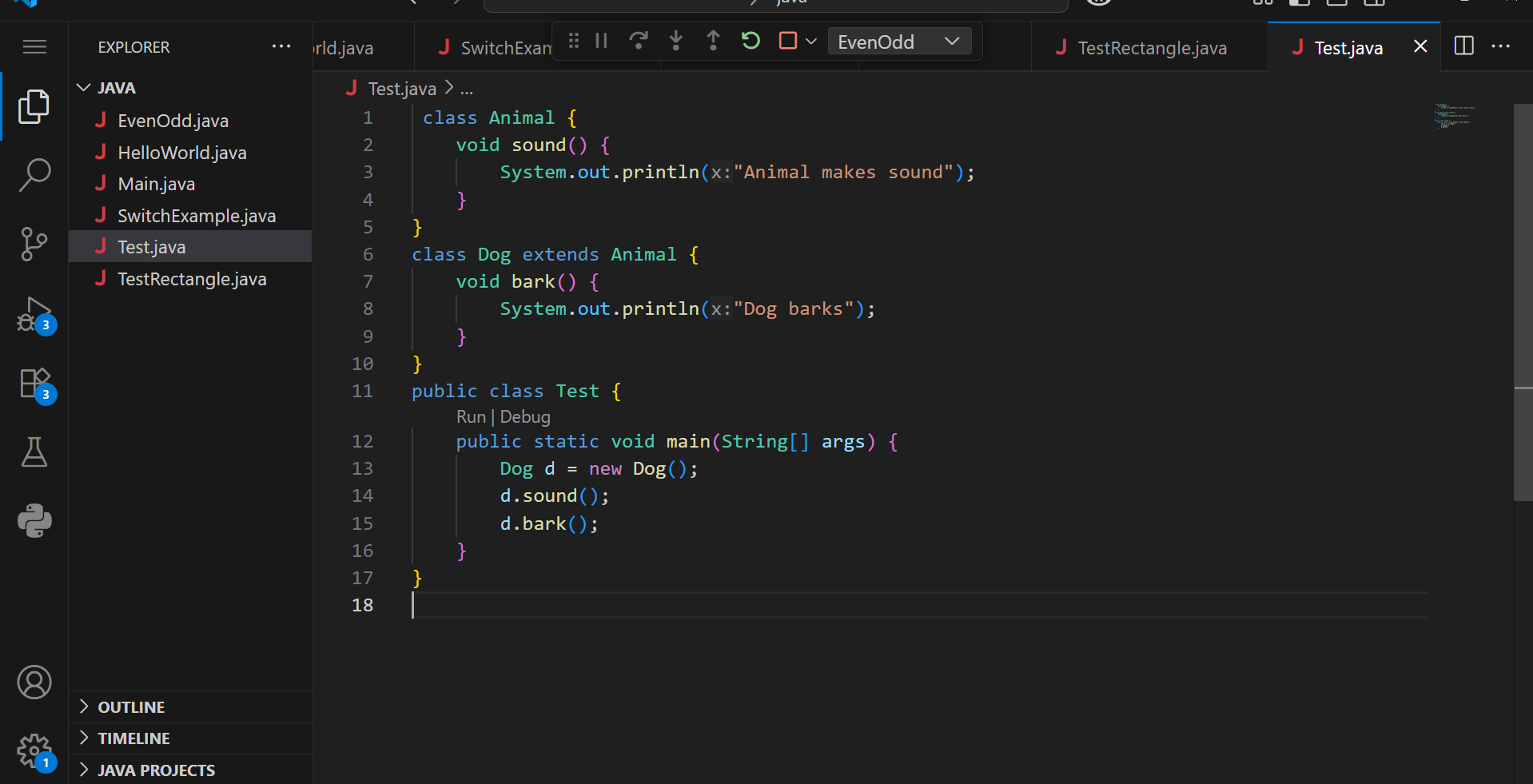
Dog d = new Dog();

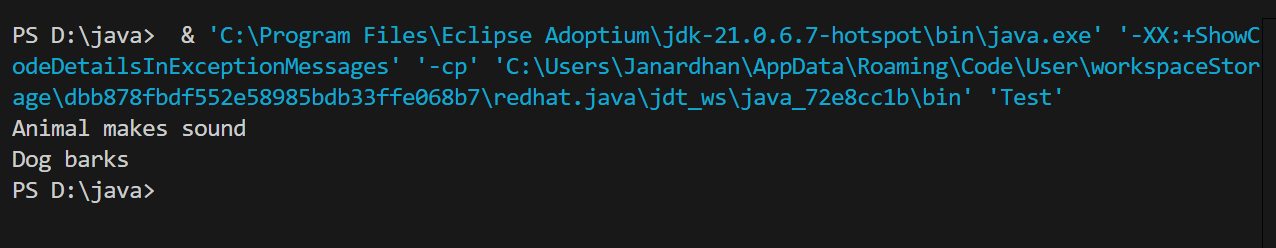
d.sound();

d.bark();

}

}





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

**Polymorphism** means "**many forms**".  
In Java, it allows the **same method name** to behave **differently** based on the context.

* **Compile-Time (Overloading)**
* **Runtime (Overriding)**

/**/ Compile-Time Polymorphism: Method Overloading**

class Calculator {

int add(int a, int b) {

return a + b;

}

int add(int a, int b, int c) {

return a + b + c;

}

}

**// Runtime Polymorphism: Method Overriding**

class Animal {

void sound() {

System.out.println("Animal makes sound");

}

}

class Dog extends Animal {

@Override

void sound() {

System.out.println("Dog barks");

}

}

public class PolymorphismDemo {

public static void main(String[] args) {

// Using Calculator class - Method Overloading

Calculator calc = new Calculator();

System.out.println("Sum of 10 + 20: " + calc.add(10, 20));

System.out.println("Sum of 10 + 20 + 30: " + calc.add(10, 20, 30));

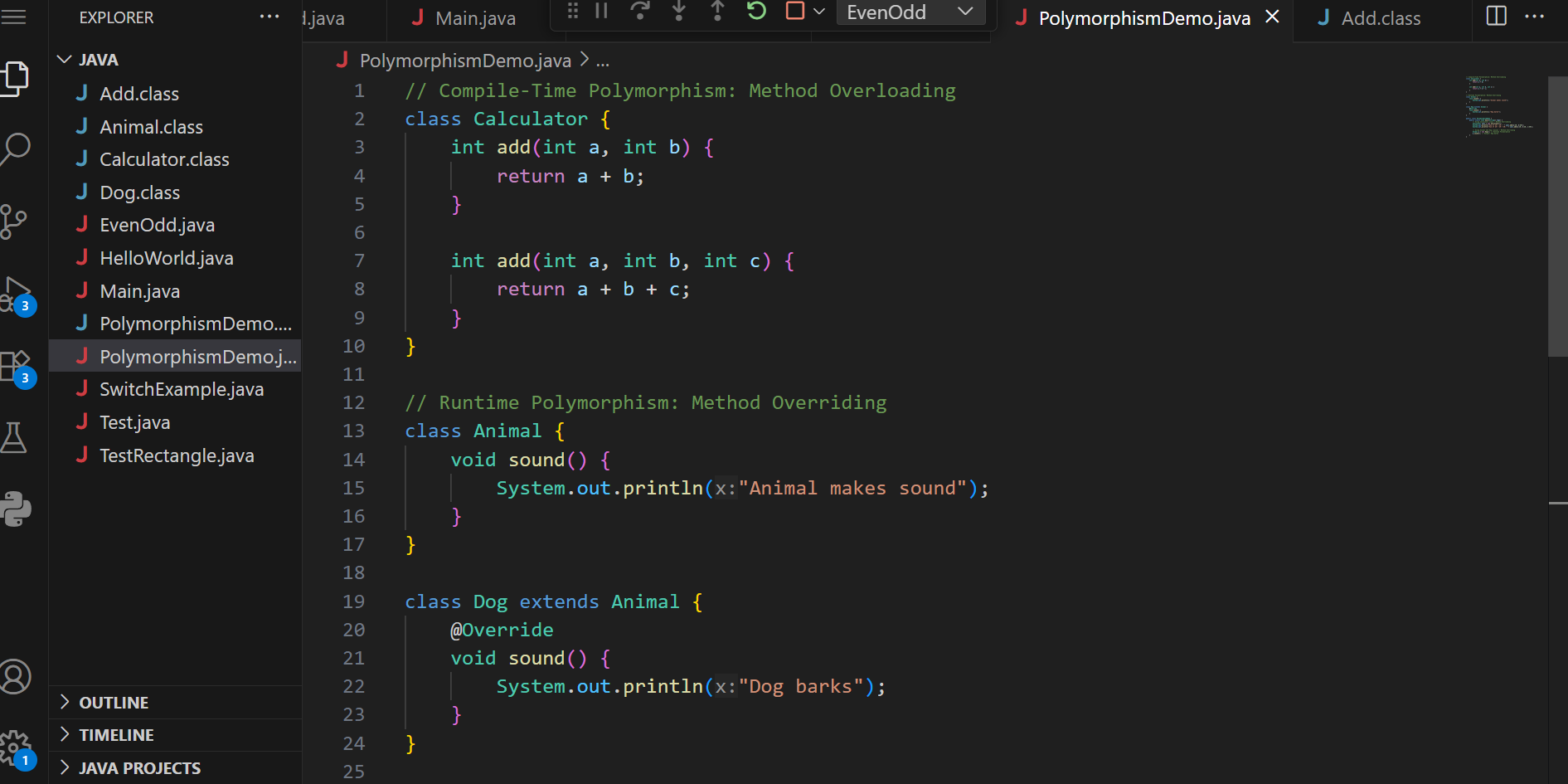
// Using Animal and Dog classes - Method Overriding

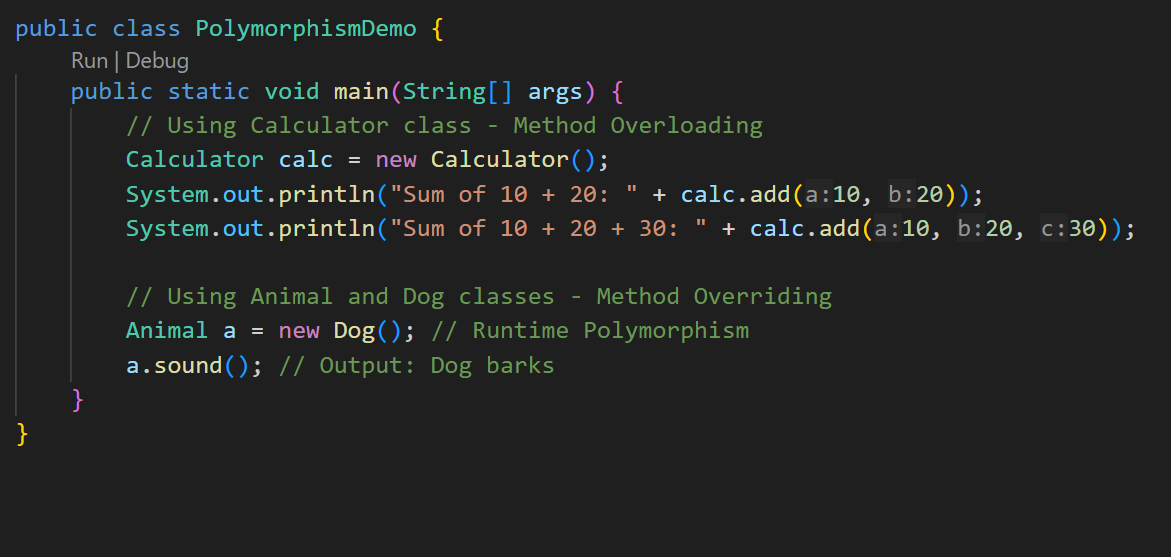
Animal a = new Dog(); // Runtime Polymorphism

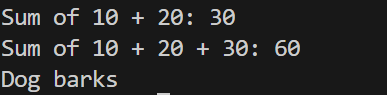
a.sound(); // Output: Dog barks

}

}







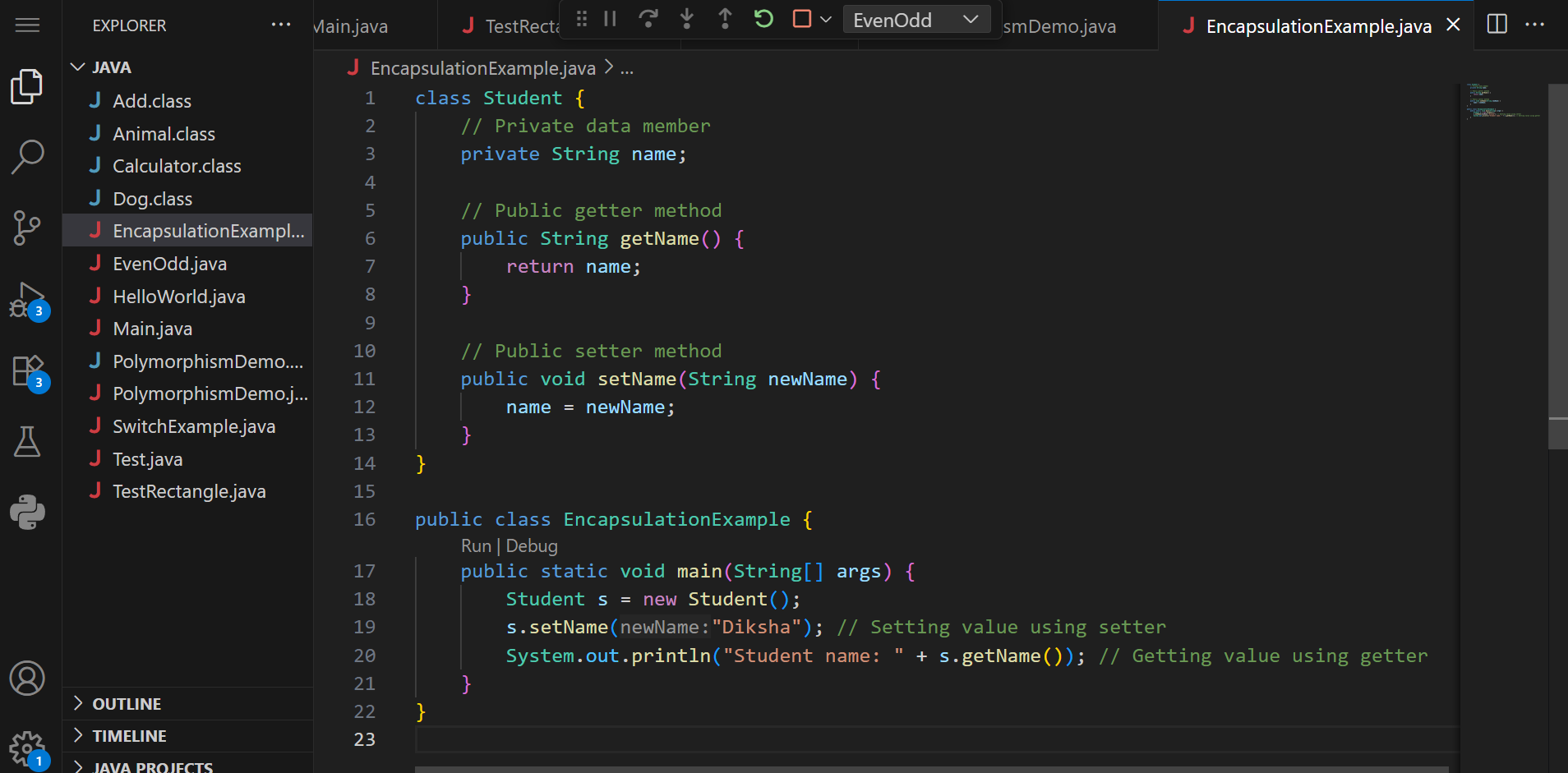
6. What is method overloading and method overriding? Show with examples.

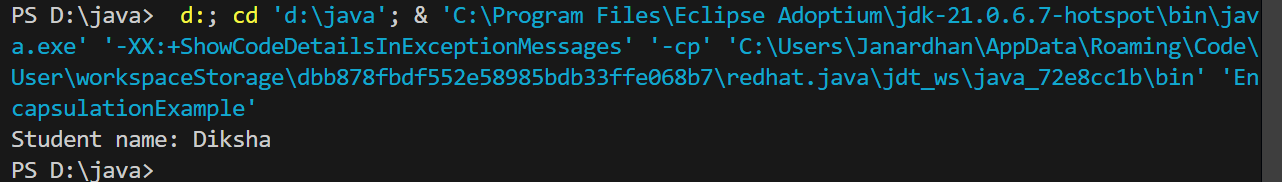
* **Overloading**: Same method name, different parameters.
* **Overriding**: Same method in child class with different behavior.

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

**Encapsulation** is the process of **hiding internal details** of a class and **protecting the data** by bundling it with related methods.

It means **wrapping data (variables)** and **code (methods)** into a single unit — a **class** — and **restricting direct access** to some of the object’s components.





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

**abstraction** is the concept of hiding internal implementation details and exposing only the essential functionality to the user.  
It is achieved using **abstract classes** and **interfaces**.

* Abstract classes can contain both abstract and non-abstract methods.
* Interfaces provide complete abstraction by allowing only abstract methods (in Java 7) or abstract + default/static methods (in Java 8+).  
  Abstraction helps in reducing complexity, improving security, and enhancing code flexibility.

9.Explain the difference between abstract class and interface.

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| Keywords | abstract | interface |
| Methods | Can have both abstract & concrete | Only abstract (Java 7), default/static (Java 8+) |
| Inheritance | Single | Multiple |
| Use | Partial abstraction | Full abstraction |

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

