Name : Aditya Suryawanshi

**Java Basics**

**1. What is Java? Explain its features.**

**Java** is a high-level, object-oriented, platform-independent programming language. It was developed by Sun Microsystems (now owned by Oracle).

**Features:**

* **Platform Independent**: Compile once, run anywhere (WORA).
* **Object-Oriented**: Everything is treated as an object.
* **Secure**: Runs in a virtual machine sandbox.
* **Robust**: Strong memory management.
* **Multithreaded**: Supports multithreaded programming.
* **High Performance**: Just-In-Time (JIT) compiler improves performance.

**2. Explain the Java program execution process.**

1. Write Java code (.java file)
2. Compile using javac → generates .class bytecode
3. Execute using JVM (java command) → runs on any platform

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

java

CopyEdit

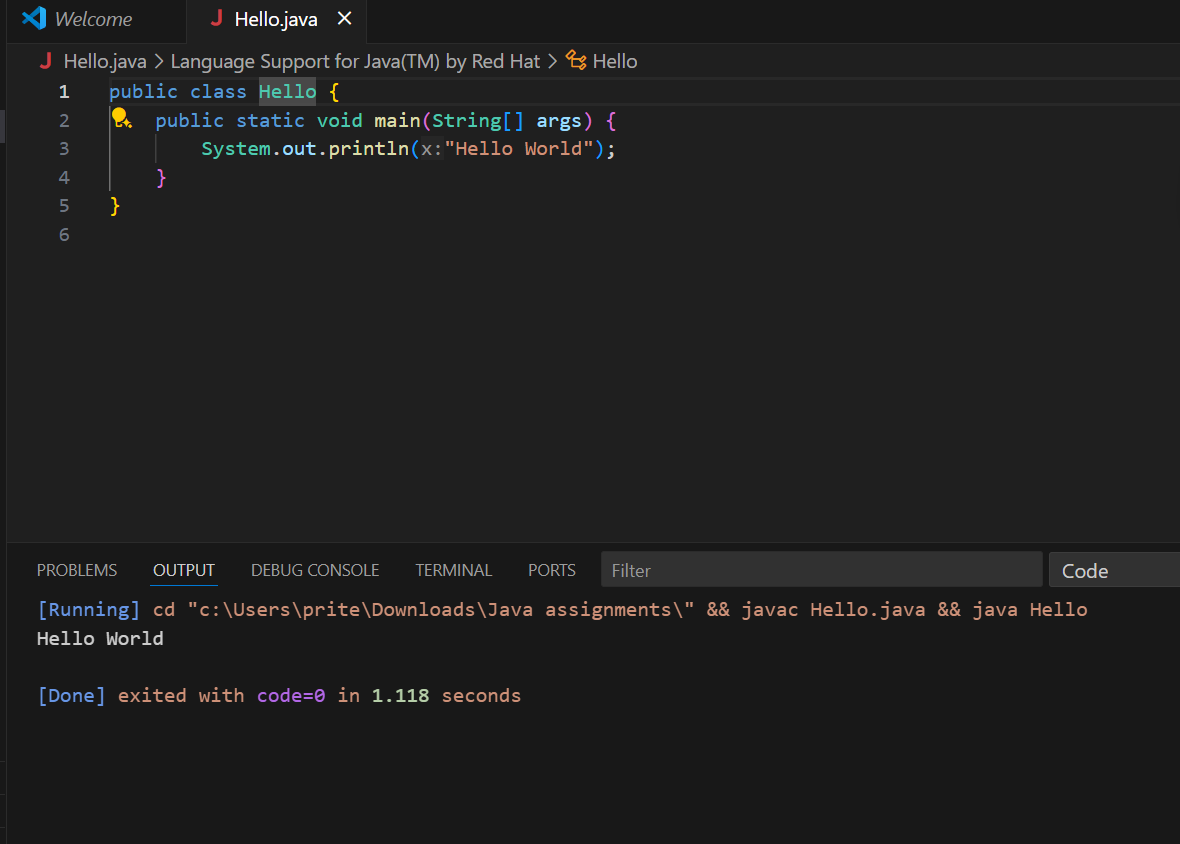
public class HelloWorld {

public static void main(String[] args) {

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

}

}



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

Java has two types:

* **Primitive**: int, float, double, char, boolean, byte, short, long
* **Non-primitive**: String, Array, Class, Interface

Example:

java

CopyEdit

int age = 25;

String name = "Sonal";

**5. Difference between JDK, JRE, and JVM**

| **Term** | **Description** |
| --- | --- |
| JVM | Runs Java bytecode |
| JRE | JVM + libraries (for running Java apps) |
| JDK | JRE + compiler and tools (for developing Java apps) |

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

A **variable** is a container for storing data values.

java

CopyEdit

int x = 10; // integer variable

String name = "Harsh"; // string variable

**7. Different types of operators in Java**

* **Arithmetic**: +, -, \*, /, %
* **Relational**: ==, !=, >, <, >=, <=
* **Logical**: &&, ||, !
* **Assignment**: =, +=, -=, etc.
* **Unary**: ++, --
* **Bitwise**: &, |, ^

**8. Control statements in Java (if, if-else, switch)**

java

CopyEdit

int x = 10;

if (x > 5) {

System.out.println("Greater than 5");

} else {

System.out.println("5 or less");

}

switch (x) {

case 10: System.out.println("Ten"); break;

default: System.out.println("Other");

}

**9. Java program to find even or odd number**

java

CopyEdit

import java.util.Scanner;

public class EvenOdd {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int num = sc.nextInt();

if (num % 2 == 0)

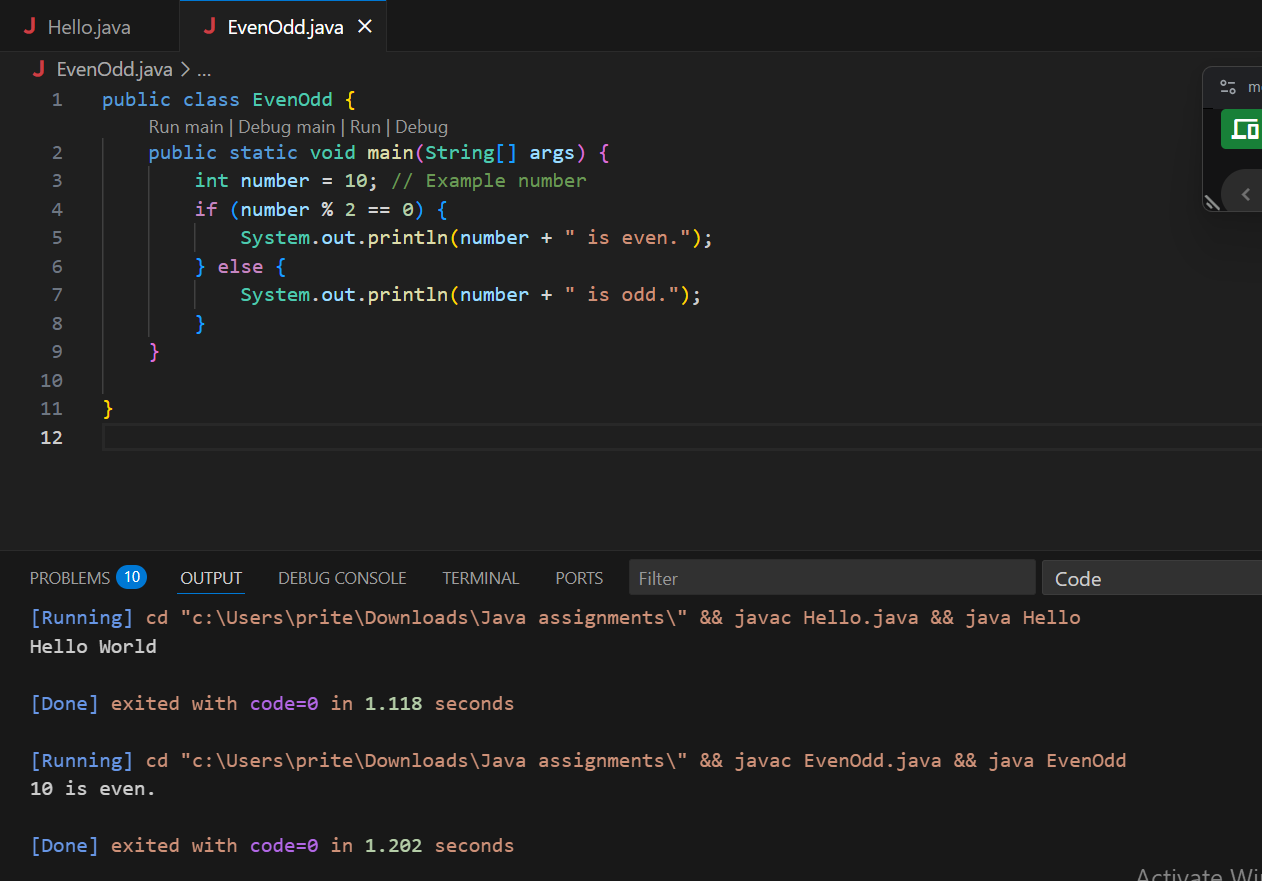
System.out.println("Even");

else

System.out.println("Odd");

}

}



**10. Difference between while and do-while loop**

| **While Loop** | **Do-While Loop** |
| --- | --- |
| Condition checked first | Condition checked after execution |
| May never execute | Executes at least once |

**Object-Oriented Programming (OOPs)**

**1. Principles of OOPs in Java**

* **Encapsulation**: Data hiding using classes
* **Abstraction**: Hiding implementation details
* **Inheritance**: Code reuse through subclasses
* **Polymorphism**: Many forms of methods/objects

**2. What is a class and object in Java?**

java

CopyEdit

class Car {

String color;

void drive() {

System.out.println("Driving...");

}

}

public class Main {

public static void main(String[] args) {

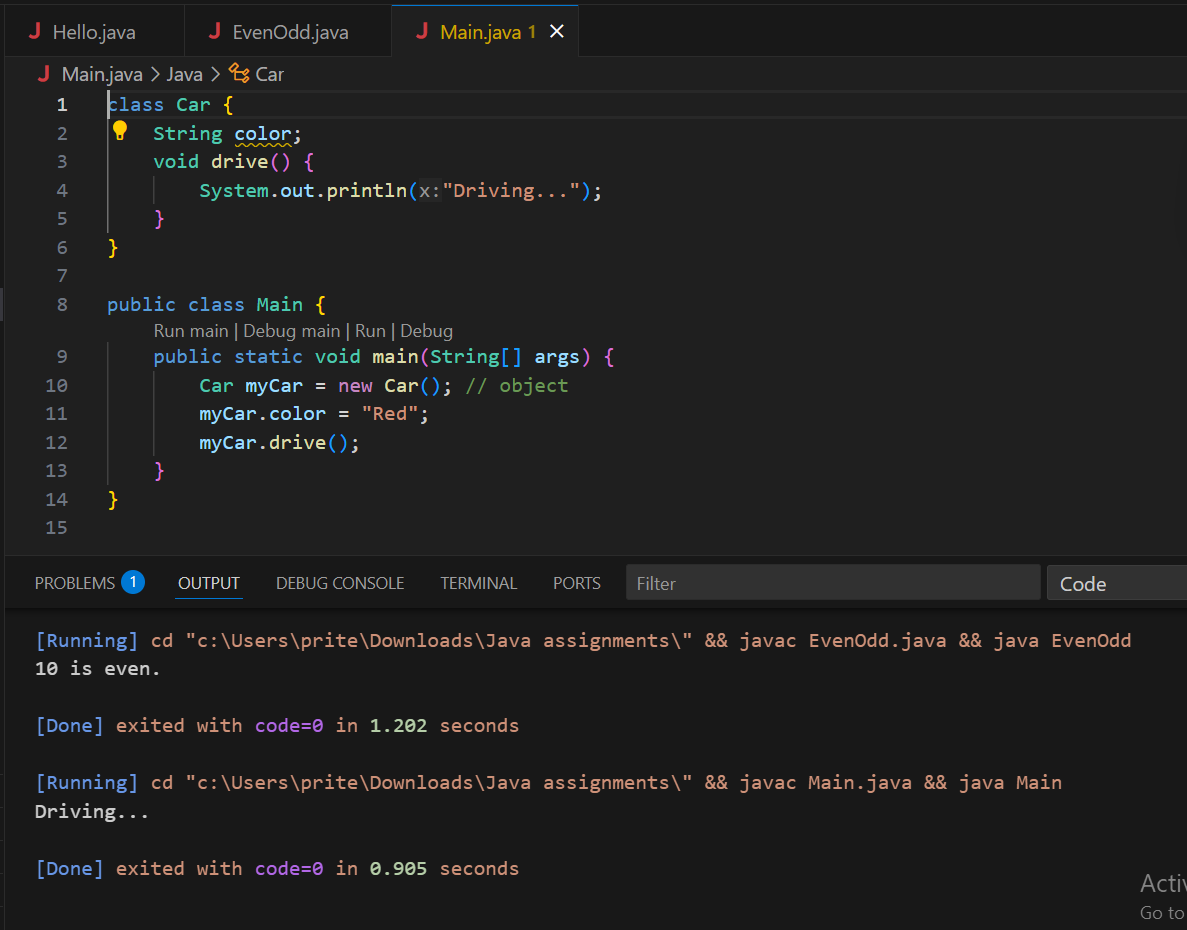
Car myCar = new Car(); // object

myCar.color = "Red";

myCar.drive();

}

}



**3. Program to calculate area of rectangle**

java

CopyEdit

class Rectangle {

int length, breadth;

int calculateArea() {

return length \* breadth;

}

}

public class Main {

public static void main(String[] args) {

Rectangle r = new Rectangle();

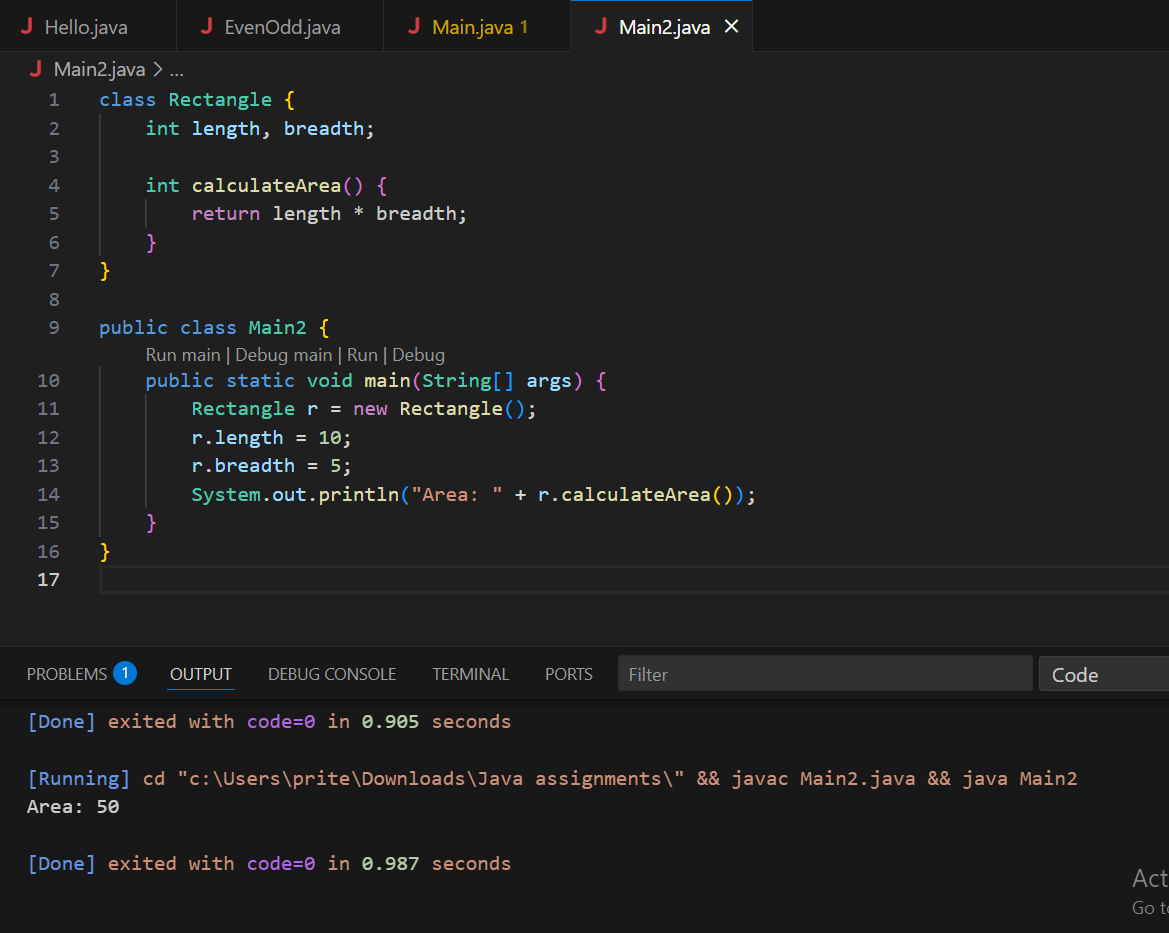
r.length = 10;

r.breadth = 5;

System.out.println("Area: " + r.calculateArea());

}

}



**4. Inheritance with real-life example**

java

CopyEdit

class Animal {

void eat() {

System.out.println("This animal eats food.");

}

}

class Dog extends Animal {

void bark() {

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

}

}

public class Main {

public static void main(String[] args) {

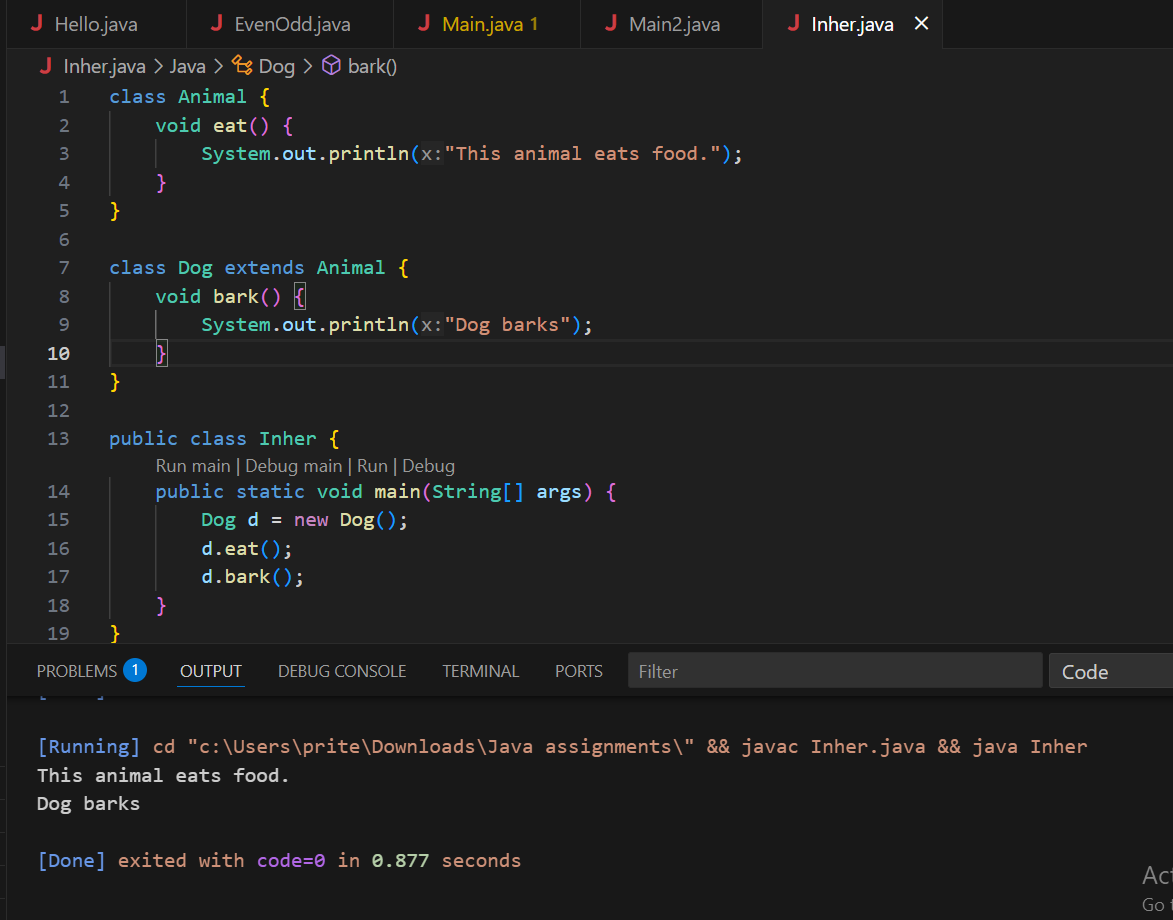
Dog d = new Dog();

d.eat();

d.bark();

}

}



**5. What is polymorphism?**

**Runtime (method overriding):**

class Animal {

    void sound() {

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

    }

}

class Dog extends Animal {

    @Override

    void sound() {

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

    }

}

class Cat extends Animal {

    @Override

    void sound() {

        System.out.println("Cat meows");

    }

}

public class Runtime {

    public static void main(String[] args) {

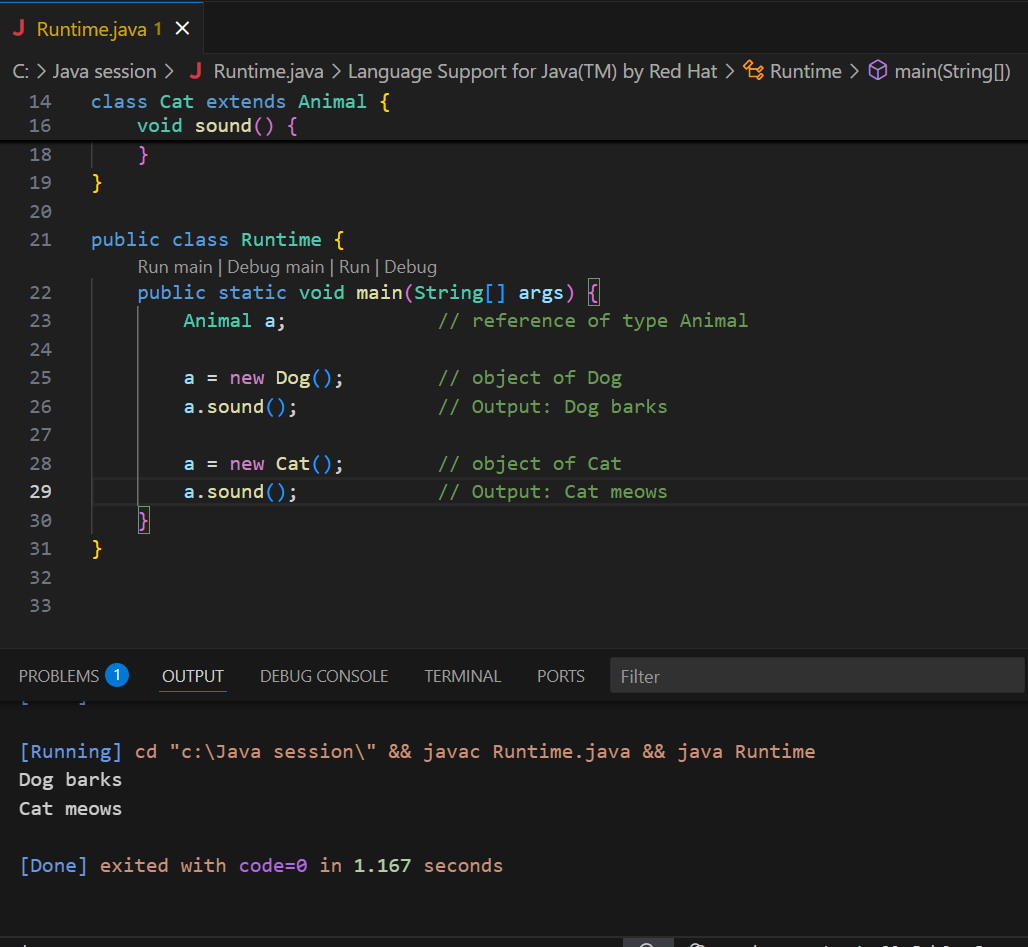
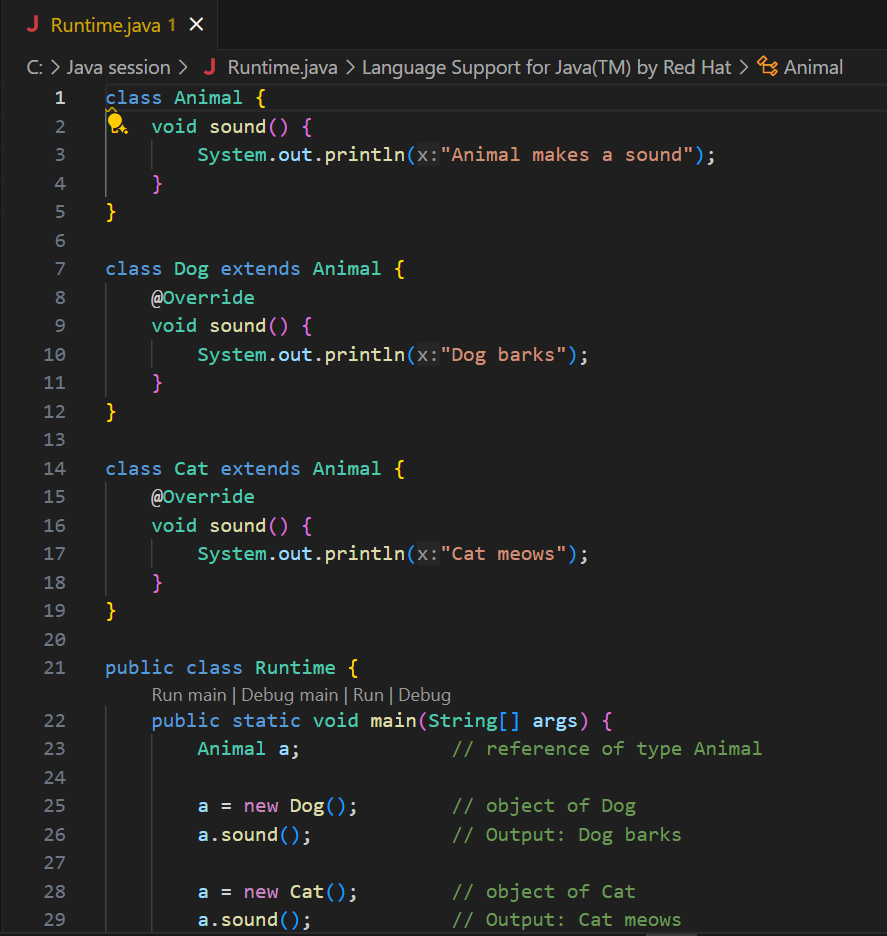
        Animal a;             // reference of type Animal

        a = new Dog();        // object of Dog

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

        a = new Cat();        // object of Cat

        a.sound();            // Output: Cat meows



**Compile-time (method overloading):**

class MathUtils {

    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 Compile {

    public static void main(String[] args) {

        MathUtils mu = new MathUtils();

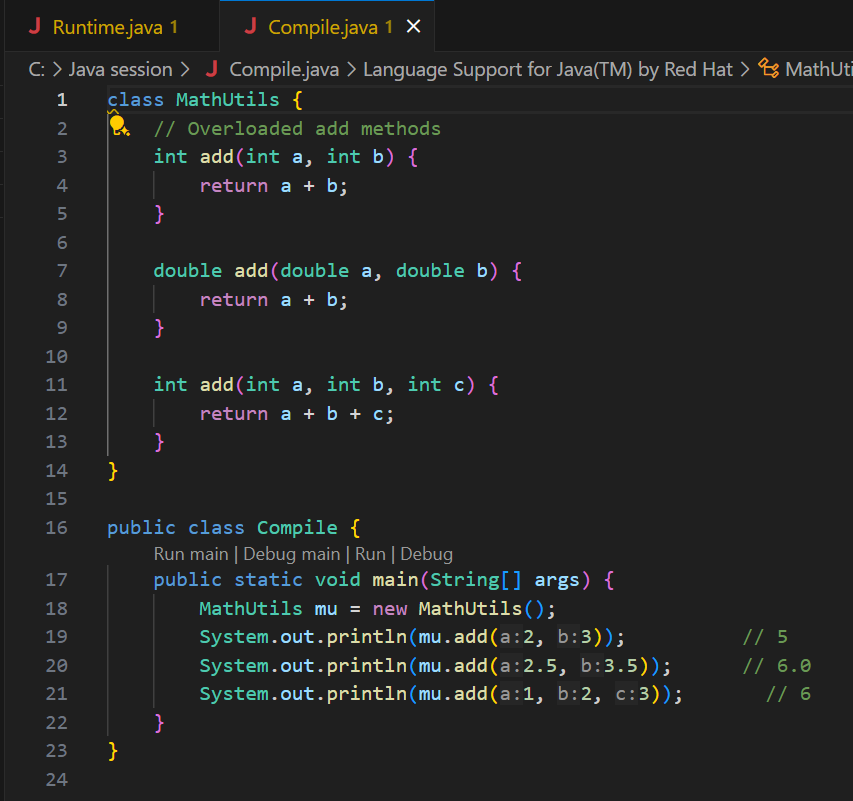
        System.out.println(mu.add(2, 3));          // 5

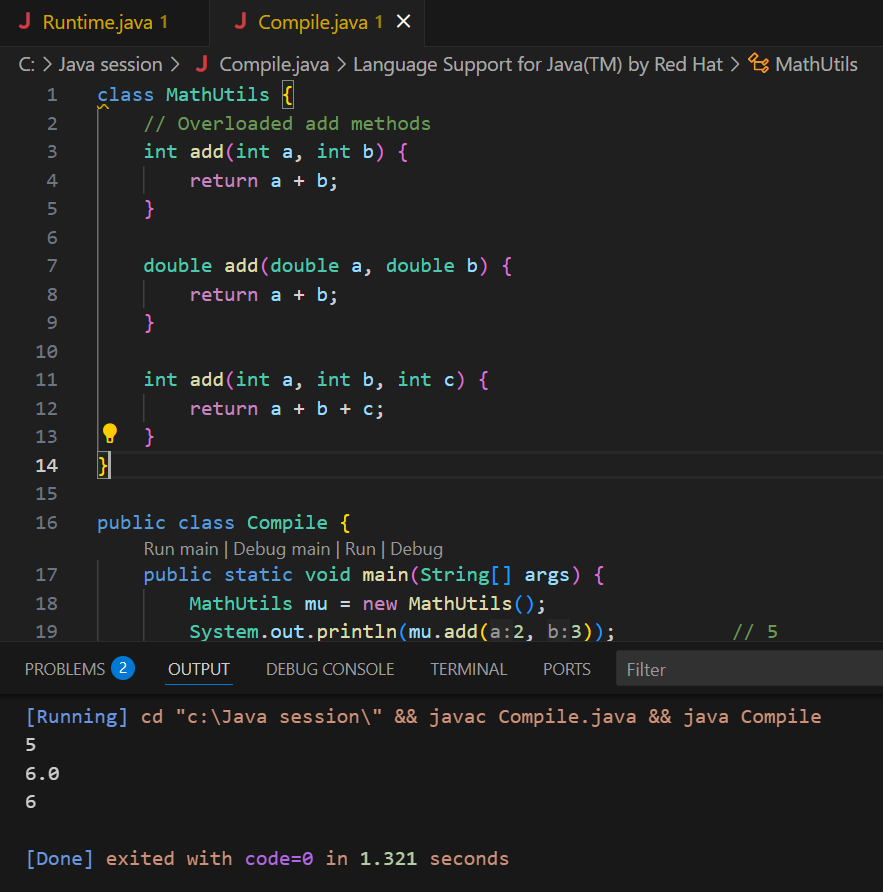
        System.out.println(mu.add(2.5, 3.5));      // 6.0

        System.out.println(mu.add(1, 2, 3));       // 6

    }

}





**6. Method Overloading vs Overriding**

**Overloading**: Same method name, different parameters (same class)

**Overriding**: Same method name and parameters in subclass

**7. Program for encapsulation**

public class person {

    private String name;

    private int age;

    public String getName() {

        return name;

    }

    public void setName(String newName) {

        name = newName;

    }

    public int getAge() {

        return age;

    }

    public void setAge(int newAge) {

        if (newAge > 0) {

            age = newAge;

        } else {

            System.out.println("Age must be positive.");

        }

    }

    public static void main(String[] args) {

        person p1 = new person();

        p1.setName("Sonal");

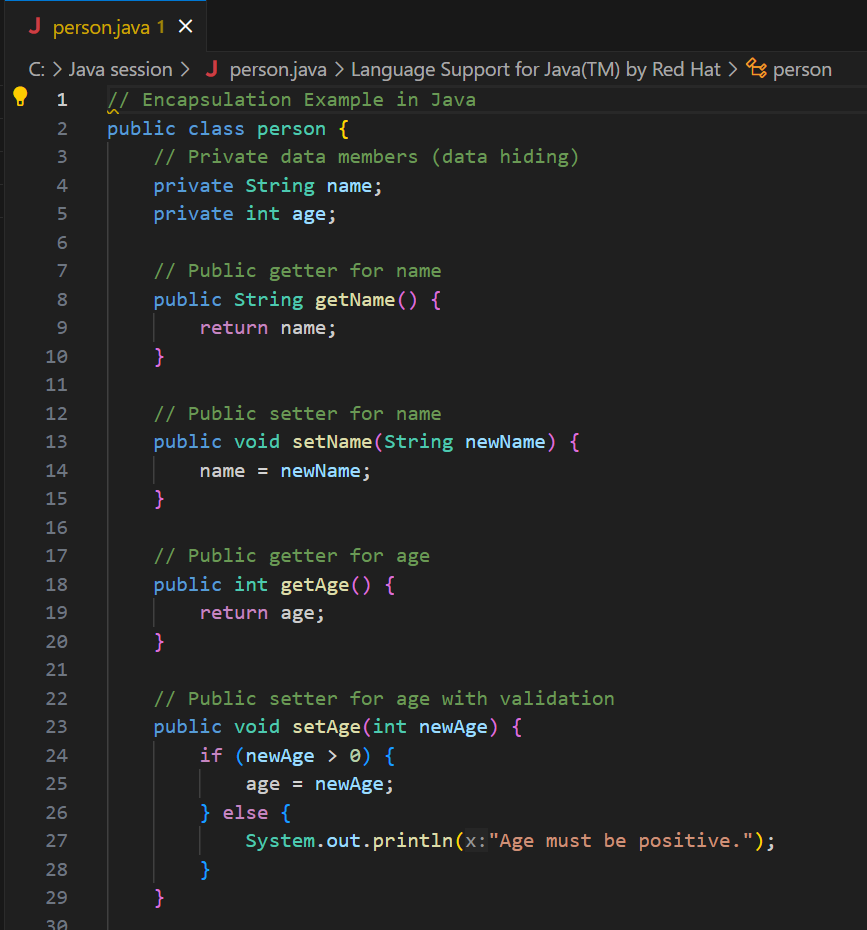
        p1.setAge(18);

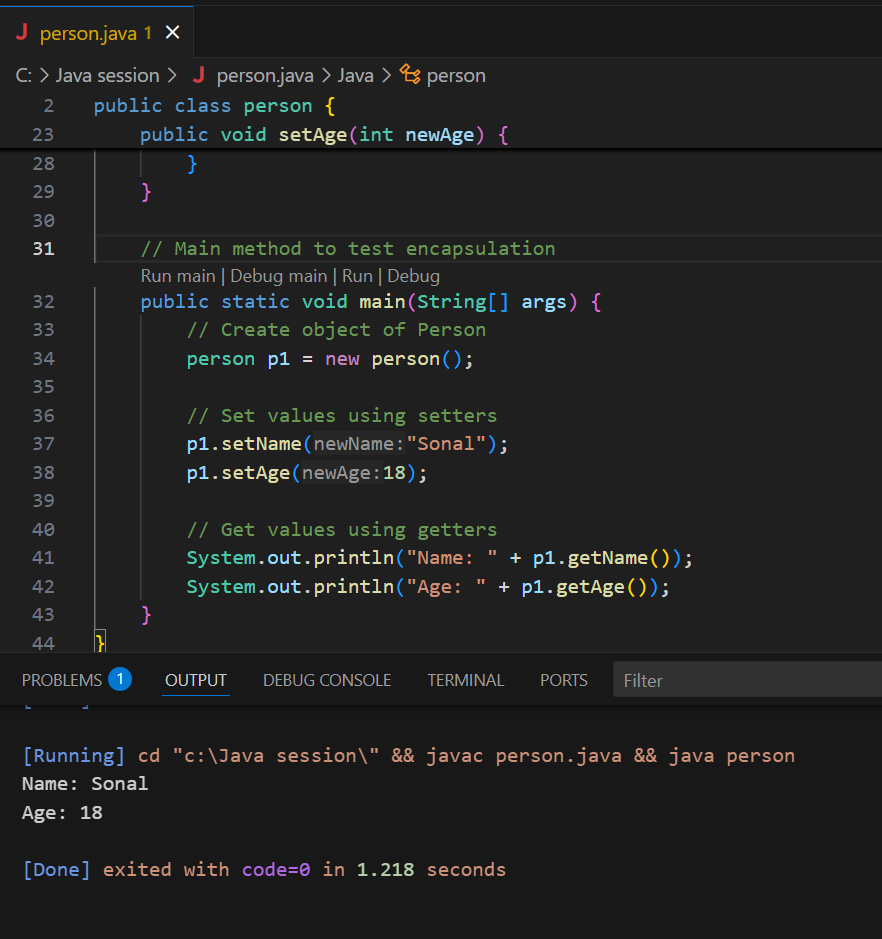
        System.out.println("Name: " + p1.getName());

        System.out.println("Age: " + p1.getAge());

    }

}





**8. What is abstraction?**

Abstraction means hiding details and showing only essential features. Achieved using:

* **Abstract class**
* **Interface**

**9. Abstract class vs Interface**

| **Abstract Class** | **Interface** |
| --- | --- |
| Can have constructors | Cannot have constructors |
| Can have both abstract and concrete methods | All methods abstract (Java 7) |
| Supports inheritance | Supports multiple inheritance |

**10. Program using Interface**

java

CopyEdit

interface Vehicle {

void start();

}

class Bike implements Vehicle {

public void start() {

System.out.println("Bike started");

}

}

public class Main {

public static void main(String[] args) {

Vehicle v = new Bike();

v.start();

}

}

