Shreya Dilip Babde

1. Java Basics

1. What is Java? Explain its features.

Java is a high-level, object-oriented, and platform-independent programming language used to build applications that can run on any device with a JVM.

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.

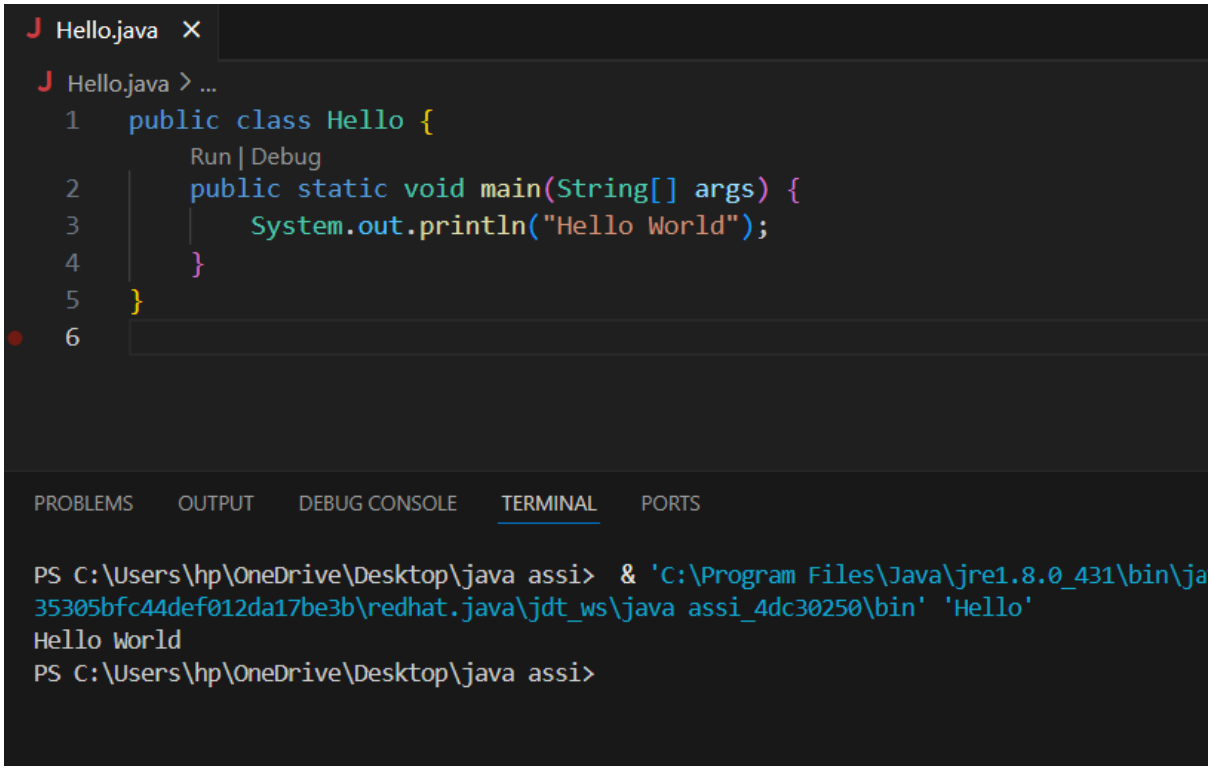
2. Explain the Java program execution process,

1. Code is written in .java file.

2. Compiled using javac → generates .class (bytecode).

3. Bytecode is run by the JVM.

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



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

1.Primitive Data Types :

1. int – Integer (e.g., 10)
2. float – Decimal (e.g., 10.5f)
3. char – Single character (e.g., 'A')
4. Boolean – true or false
5. byte, short, long, double – Other numeric types

2.Non-Primitive Data Types:

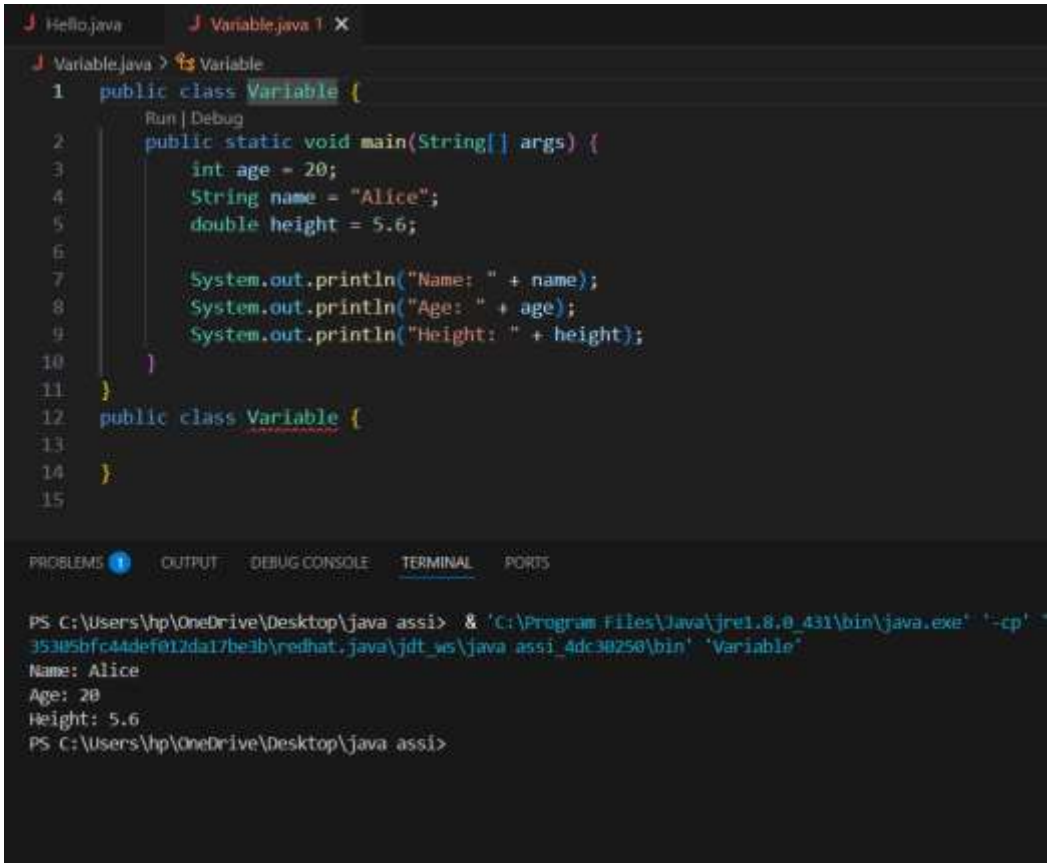
String, Arrays, Classes, Interfaces – User-defined or built-in objects.

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

- **JDK (Java Development Kit)**:
  - Contains tools to write, compile, and run Java programs (includes JRE + compiler).
- **JRE (Java Runtime Environment)**:
  - Provides environment to run Java programs (includes JVM + libraries).
- **JVM (Java Virtual Machine)**:
  - Executes Java bytecode and makes Java platform-independent.

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

Variables in Java are containers used to store data values.
Each variable has a type, a name, and can hold a value.



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

Types of Operators in Java:

1. **Arithmetic Operators**: +, -, *, /, %
2. **Relational Operators**: ==, !=, >, <, >=, <=
3. **Logical Operators**: &&, ||, !
4. **Assignment Operators**: =, +=, -=, *=, /=, %=
5. **Unary Operators**: +, -, ++, --, !
6. **Bitwise Operators**: &, |, ^, ~, <<, >>
7. **Ternary Operator**: condition ? expr1 : expr2
8. **Instanceof Operator**: Checks object type – obj instanceof Class

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

1. if Statement

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

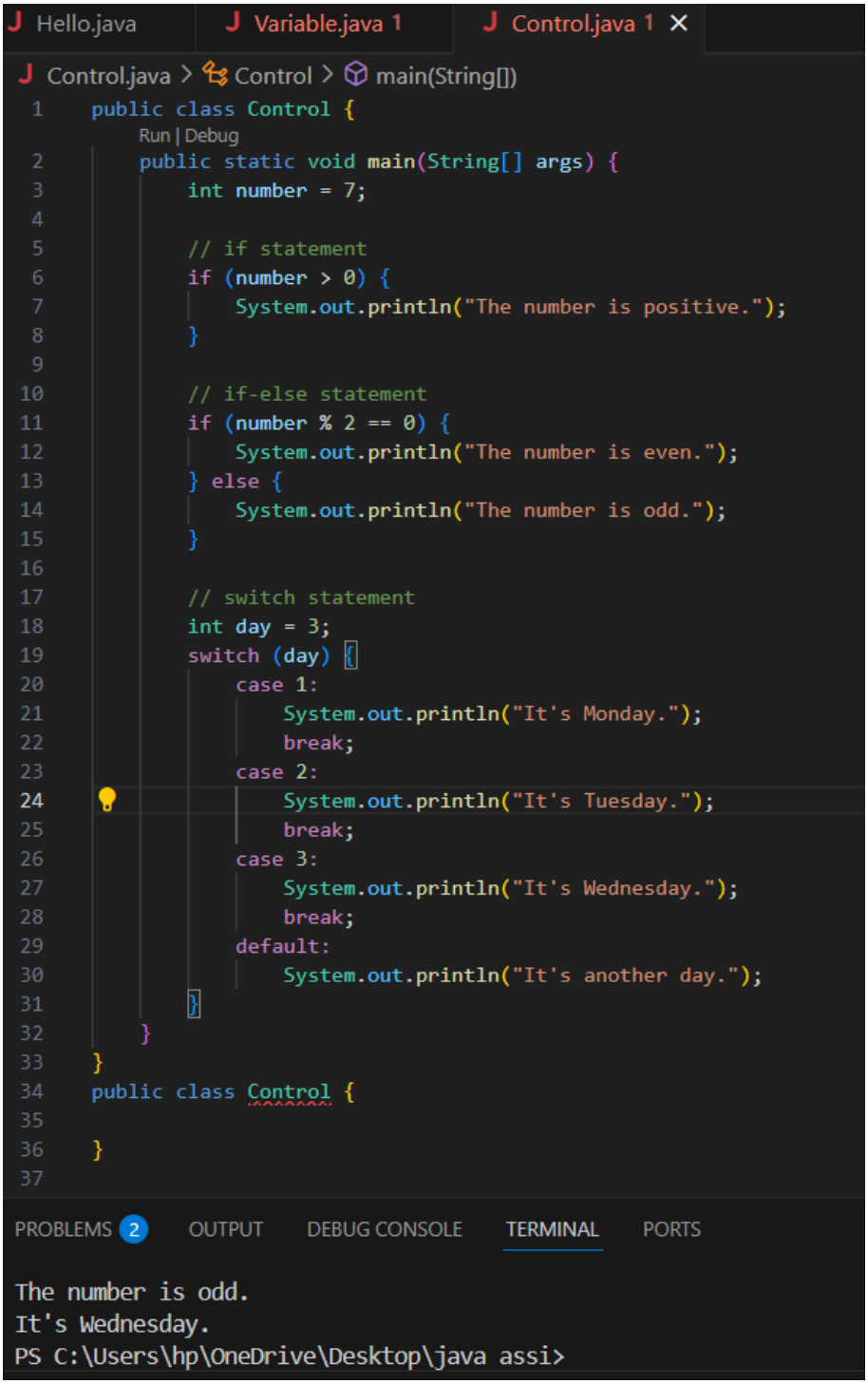
2. if-else Statement

Executes one block of code if the condition is true, otherwise executes another block.

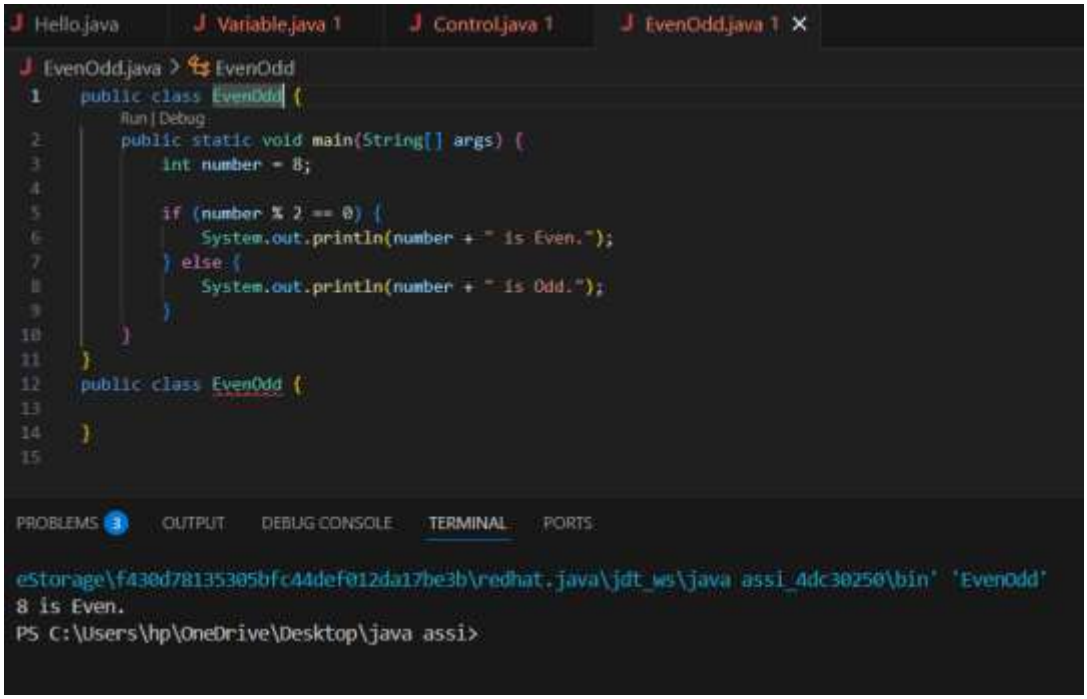
4. switch Statement

Used to select one option from multiple choices based on the value of a variable. Each

option is called a "case"



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



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

a. In a while loop, the condition is checked before the loop body is executed.

b. In a do-while loop, the condition is checked after the loop body is executed.

c. The while loop may not run at all if the condition is false initially.

d. The do-while loop will always run at least once, even if the condition is false.

e. while loop is used when you want to run the loop only if the condition is true. f. do-while loop is used when you want the loop to run once before checking the condition.

2.Object-OrientedProgramming

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

Main Principles of OOPs in Java:

1. **Encapsulation**
   - Wrapping data and methods into a single unit (class).
   - Achieved using private variables and public getters/setters.
2. **Abstraction**
   - Hiding internal details and showing only essential features.
   - Done using abstract classes and interfaces.
3. **Inheritance**
   - One class inherits properties of another class.
   - Promotes code reuse using the extends keyword.
4. **Polymorphism**
   - One task, many forms.
   - Compile-time polymorphism: method overloading.
   - Runtime polymorphism: method overriding.

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

Class: A class in Java is a blueprint for creating objects. It defines properties (variables) and behaviors (methods).

Object: An object is an instance of a class. It holds real values and can use class methods.

```java
// Class
class Student {
    String name;
    int age;

    void display() {
        System.out.println(name + " is " + age + " years old.");
    }
}

// Object
public class Main {
    Run | Debug
    public static void main(String[] args) {
        Student s1 = new Student(); // Creating object
        s1.name = "John";
        s1.age = 20;
        s1.display();
    }
}
public class Main {

}
```

```
PS C:\Users\hp\OneDrive\Desktop\java assi> & 'C:\Program Files\Java\jre1.8.0_431\bin\java.exe' '-cp
15305bfc44def012da17be3b\redhat.java\jdt_ws\java assi_4dc30250\bin' 'Main'
John is 20 years old.
PS C:\Users\hp\OneDrive\Desktop\java assi>
```

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

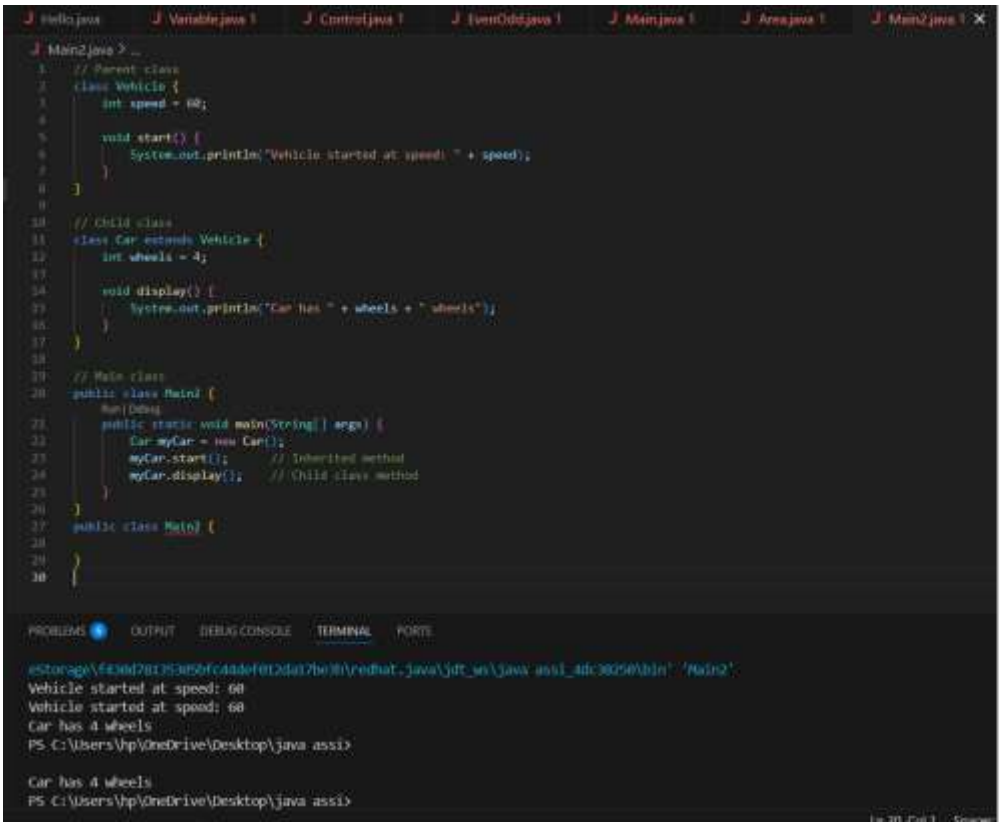
```java
// Class to represent a Rectangle
class Rectangle {
    int length;
    int width;

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

// Main class to create object and use it
public class Area {
    Run | Debug
    public static void main(String[] args) {
        Rectangle rect = new Rectangle(); // Creating object

        rect.length = 10;
        rect.width = 5;

        rect.calculateArea(); // Calling method
    }
}
public class Area {

}
```

```
PS C:\Users\hp\OneDrive\Desktop\java assi> & 'C:\Program Files\Java\jre1.8.0_431\bin\java.exe' '-cp' 'C:\Users\hp\App
eStorage\f430d7815305bfc44def012da17be3b\redhat.java\jdt_ws\java assi_4dc30250\bin' 'Area'
Area of Rectangle: 50
PS C:\Users\hp\OneDrive\Desktop\java assi>
```

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

Inheritance is an OOP principle where one class (child) inherits the properties and behaviors of another class (parent). It promotes code reusability and supports method overriding.
Real-life Example:

- Parent Class: Vehicle – has general features like speed and start().
- Child Class: Car – inherits from Vehicle and adds wheels.

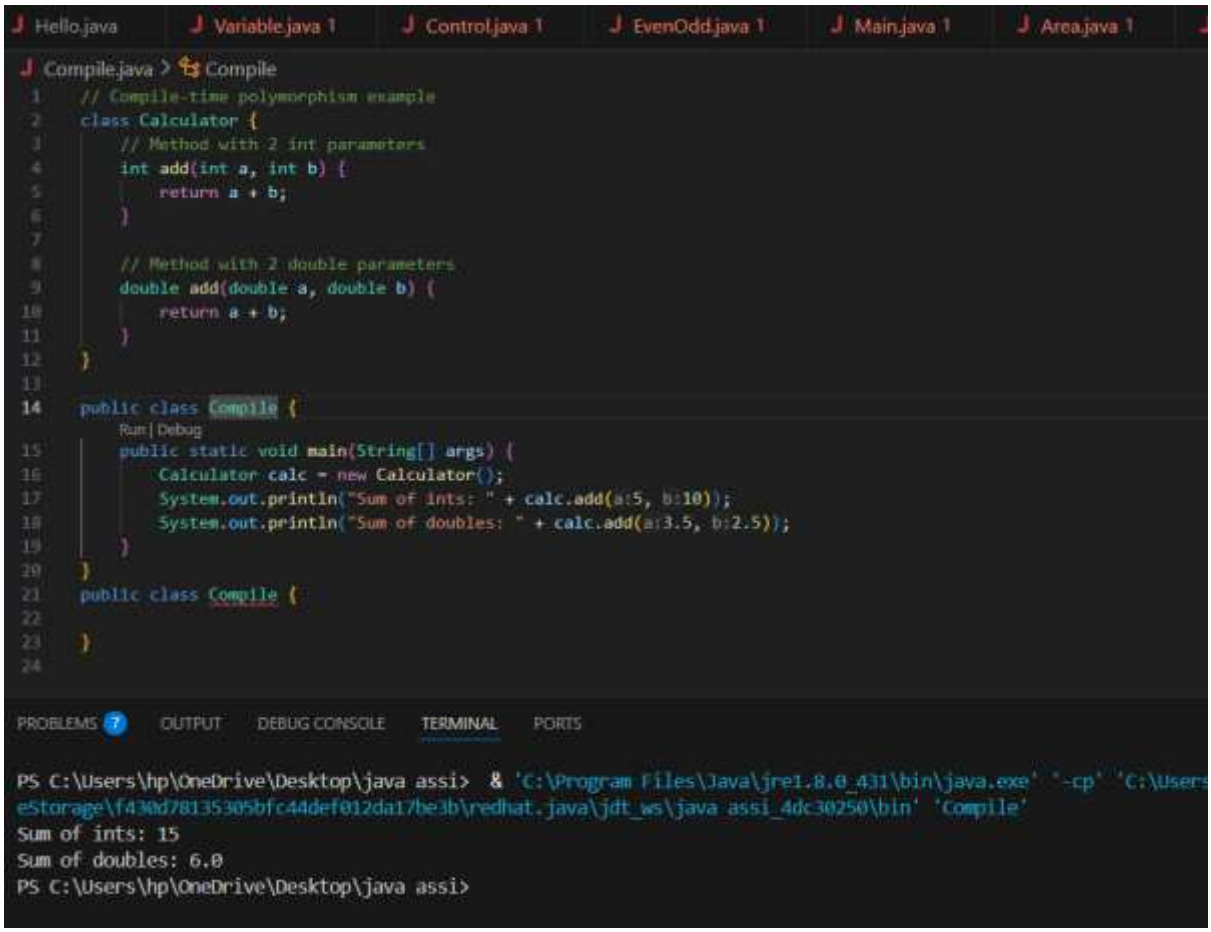


5.What is polymorphism? Explain with compile-time and runtime examples.

Polymorphism in Java means one name, many forms. It allows the same method name to perform different tasks.

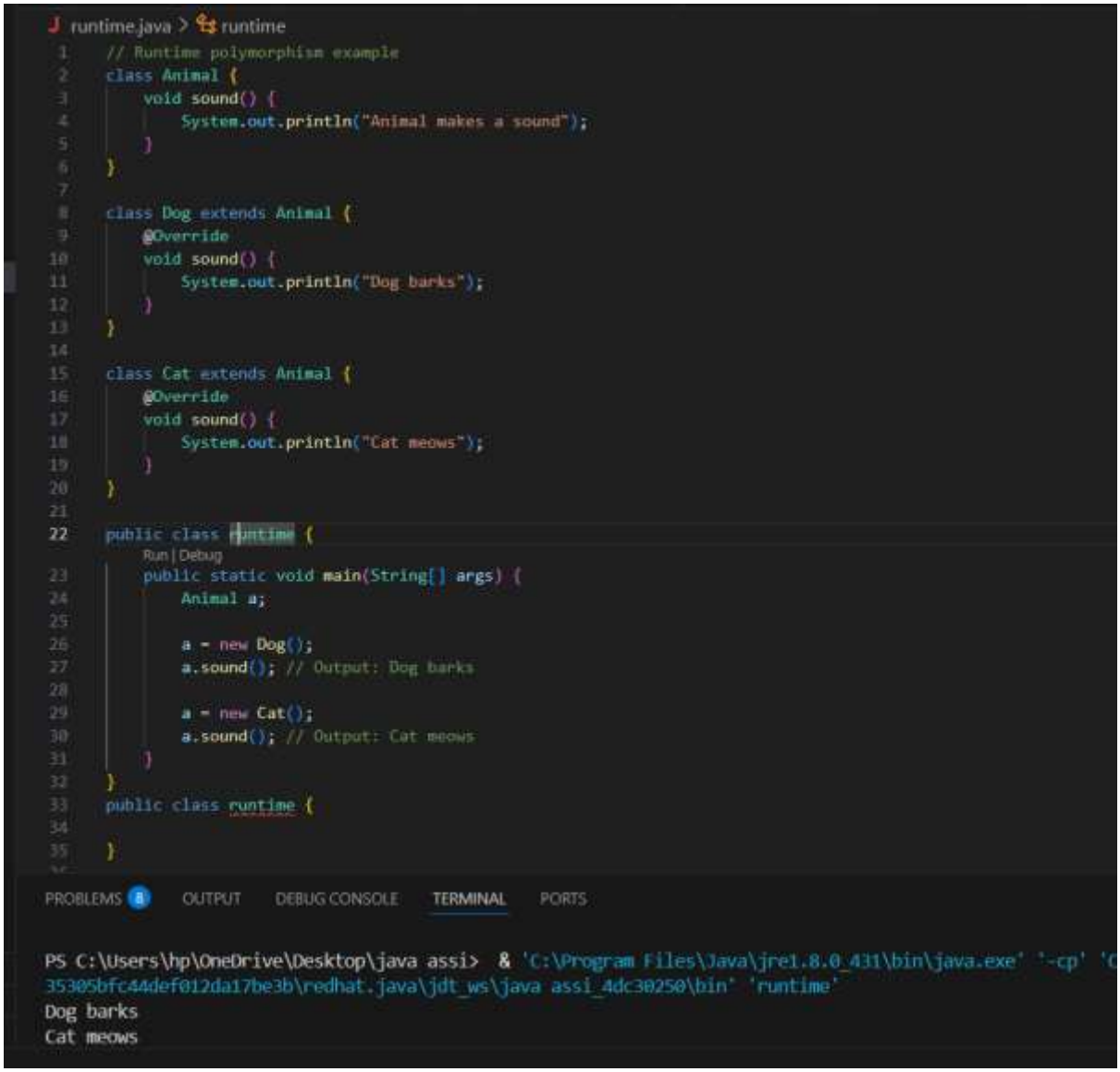
Types of Polymorphism:

1. Compile-Time Polymorphism (Method Overloading) Same method name, different parameters.



2.Runtime Polymorphism (Method Overriding)

- Subclass provides specific implementation of a method already defined in its parent.



```java
// Runtime polymorphism example
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;

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

        a = new Cat();
        a.sound(); // Output: Cat meows
    }
}
public class runtime {

}
```

```
PS C:\Users\hp\OneDrive\Desktop\java assi> & 'C:\Program Files\Java\jre1.8.0_431\bin\java.exe' '-cp' 'C
35305bfc44def012da17be3b\redhat.java\jdt_ws\java assi_4dc30250\bin' 'runtime'
Dog barks
Cat meows
```

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

- Method Overloading (Compile-Time Polymorphism)
  Same method name, but different parameters in the same class.

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

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

public class Overloading {
    Run | Debug
    public static void main(String[] args) {
        MathOperations obj = new MathOperations();
        System.out.println("Int Sum: " + obj.add(a:5, b:3));
        System.out.println("Double Sum: " + obj.add(a:2.5, b:1.5));
    }
}
public class Overloading {

}
```

```
eStorage\f430d78135305bfc44def012da17be3b\redhat.java\jdt_ws\java assi_4dc30250\bin' 'Overloading'
Int Sum: 8
Double Sum: 4.0
PS C:\Users\hp\OneDrive\Desktop\java assi>
```

- Method Overriding (Runtime Polymorphism)
  Same method name and parameters, but defined in child class to override parent method.

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

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

public class overriding {
    Run | Debug
    public static void main(String[] args) {
        Animal a = new Dog(); // Parent reference, child object
        a.sound(); // Calls Dog's version
    }
}
public class overriding {

}
```

```
PS C:\Users\hp\OneDrive\Desktop\java assi> & 'C:\Program Files\Java\jre1.8.0_431\bin\java.exe' '-c
eStorage\f430d78135305bfc44def012da17be3b\redhat.java\jdt_ws\java assi_4dc30250\bin' 'overriding'
Dog barks
PS C:\Users\hp\OneDrive\Desktop\java assi>
```

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

Encapsulation is the concept of wrapping data (variables) and methods into a single unit (class), and restricting direct access using private access modifiers. Access is provided through getters and setters.

Main.java 1 | Area.java 1 | Main2.java 1 | Compile.java 1

main3.java > main3 > main(String[])

```java
class Person {
    // Private data members
    private String name;
    private int age;

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

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

    public int getAge() {
        return age;
    }

    public void setAge(int newAge) {
        age = newAge;
    }
}

public class main3 {
    Run | Debug
    public static void main(String[] args) {
        Person p = new Person();
        p.setName(newName:"Alice");
        p.setAge(newAge:25);

        System.out.println("Name: " + p.getName());
        System.out.println("Age: " + p.getAge());
    }
}
public class main3 {

}
```

PROBLEMS 11 | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS

```
PS C:\Users\hp\OneDrive\Desktop\java assi>  & 'C:\Program Files\Java\jre1
eStorage\f430d78135305bfc44def012da17be3b\redhat.java\jdt_ws\java assi_4d
Name: Alice
Age: 25
PS C:\Users\hp\OneDrive\Desktop\java assi>
```

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

Abstraction is the process of hiding internal details and showing only essential features to the user.
It helps reduce complexity and increase code clarity.

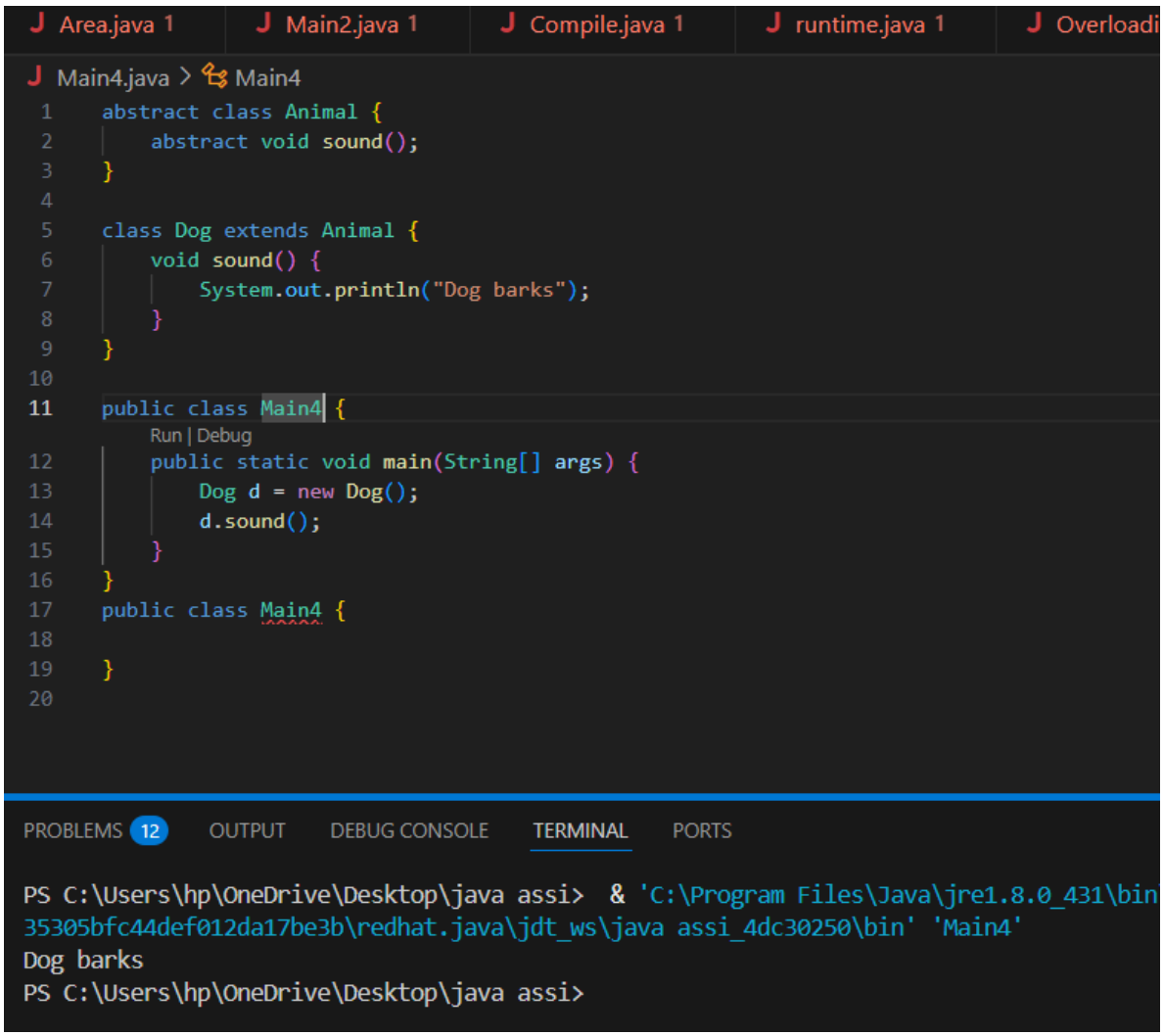
Abstraction is achieved in java :

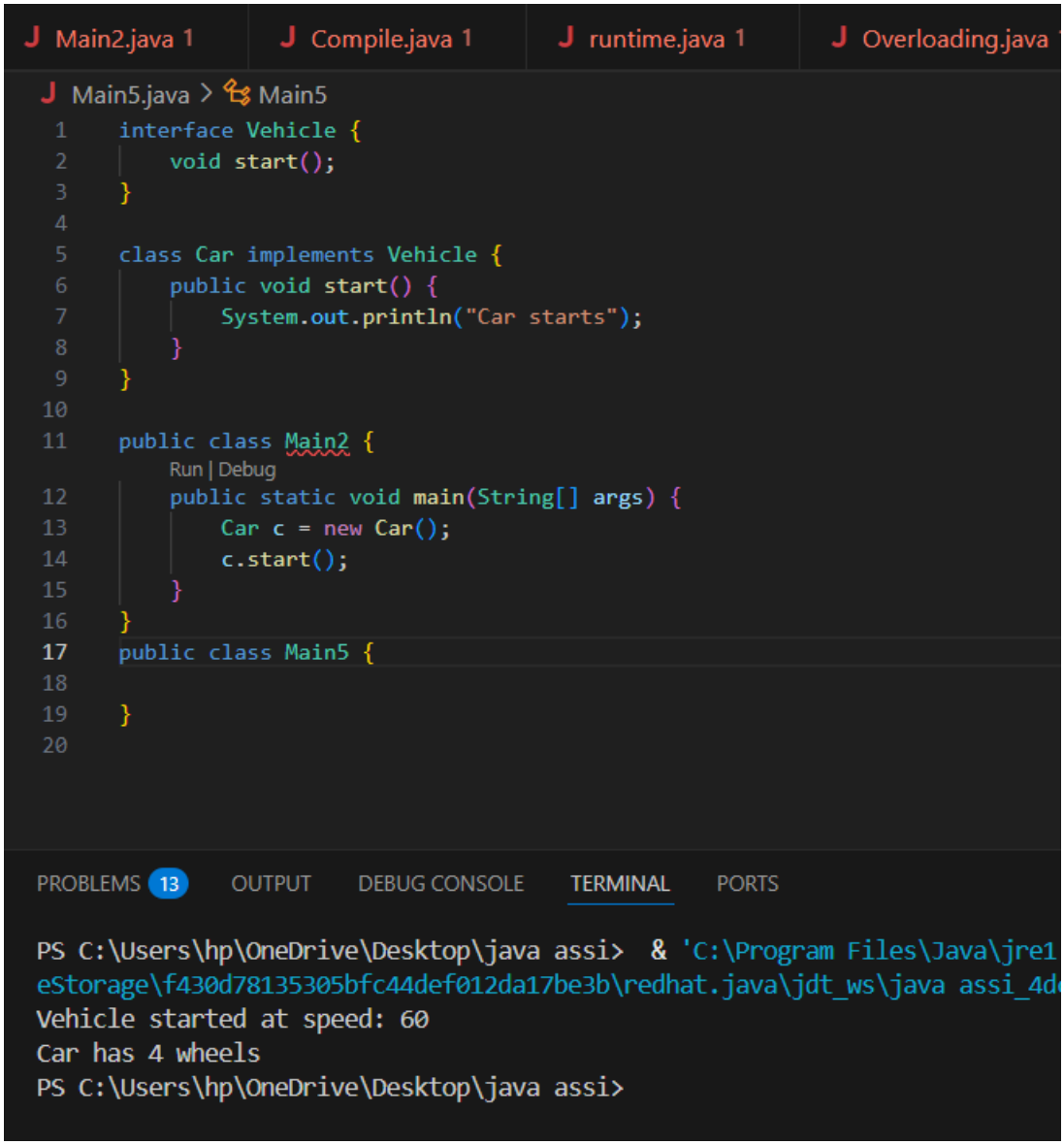
1.Using Abstract Classes

- A class with abstract keyword
- Can have both abstract (no body) and concrete methods

2.Using Interfaces

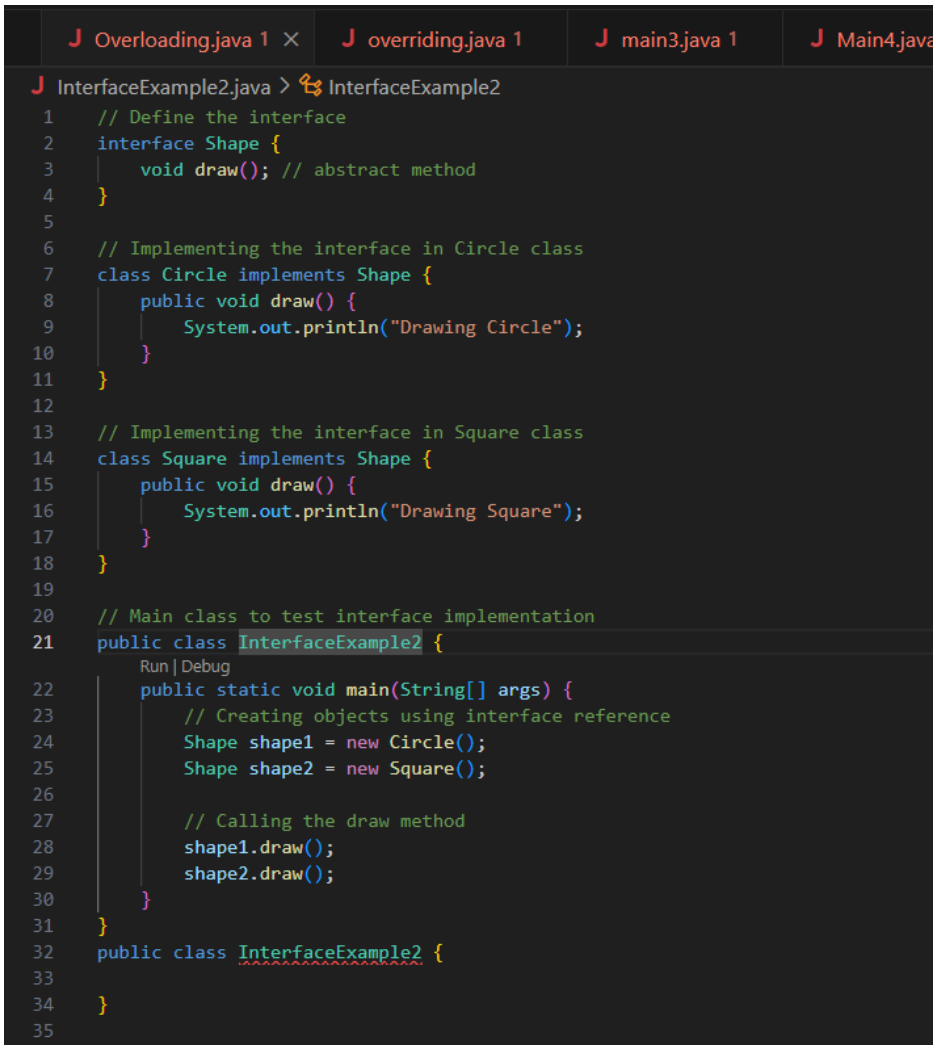
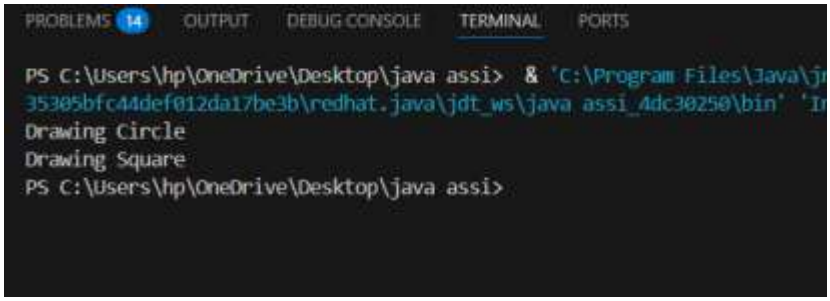
- Contains only abstract methods (Java 8+ can have default/static too)





9. Explain the difference between abstract class and interface.

1. Abstract class can have both abstract and concrete methods; interface has only abstract methods (until Java 7).
2. Abstract class can have constructors and instance variables; interface cannot have constructors, only constants.
3. A class can extend only one abstract class, but can implement multiple interfaces.
4. Abstract class members can have any access modifier; interface members are always public.
5. Use abstract class for shared code; use interface to define a common behavior (contract).

6. From Java 8, interfaces can have default and static methods.
7. Interfaces can be used as functional interfaces; abstract classes cannot.

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