

Name : Sakshi Kapil Badave

PRN: 202401110019

1. What is Java? Explain its features.

Java is a high-level, object-oriented, class-based programming language. It is designed to have as few implementation dependencies as possible, making it ideal for cross-platform development. Java programs are compiled into bytecode which can run on any device that has a Java Virtual Machine (JVM) — making Java a "write once, run anywhere" language.

Features of java :

- 1.Simple: It is a very simple language. It is easy to learn.
- 2.Object Oriented: Everything in Java is treated as an object.
- 3.Platform independent: Java code is compiled into bytecode, which is platform-independent.
- 4.Robust: Java emphasizes early error checking, runtime checking, and garbage collection.
5. Dynamic: Java can dynamically link classes and methods at runtime, making it adaptable to evolving environments.

2. Explain the Java program execution process.

1.Write java program:

Write the java code and save the file with “.java” extension.

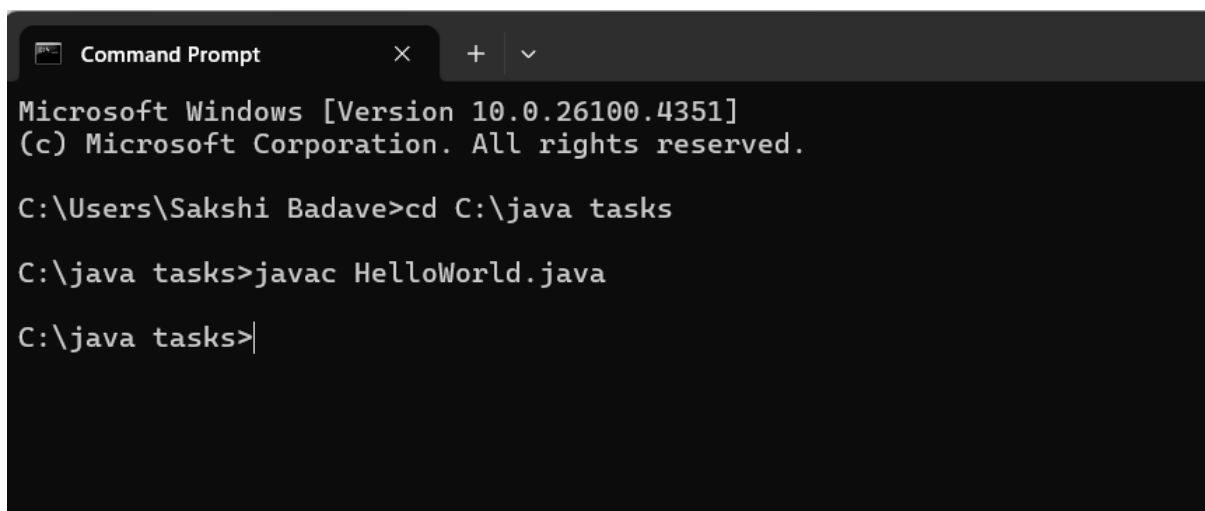


A screenshot of a code editor window. The title bar shows a tab for 'HelloWorld.java' with a plus sign to its right. Below the title bar is a menu bar with 'File', 'Edit', and 'View'. The main area contains the following Java code:

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

2.Compilation:

The source code is compiled by the Java Compiler (javac).

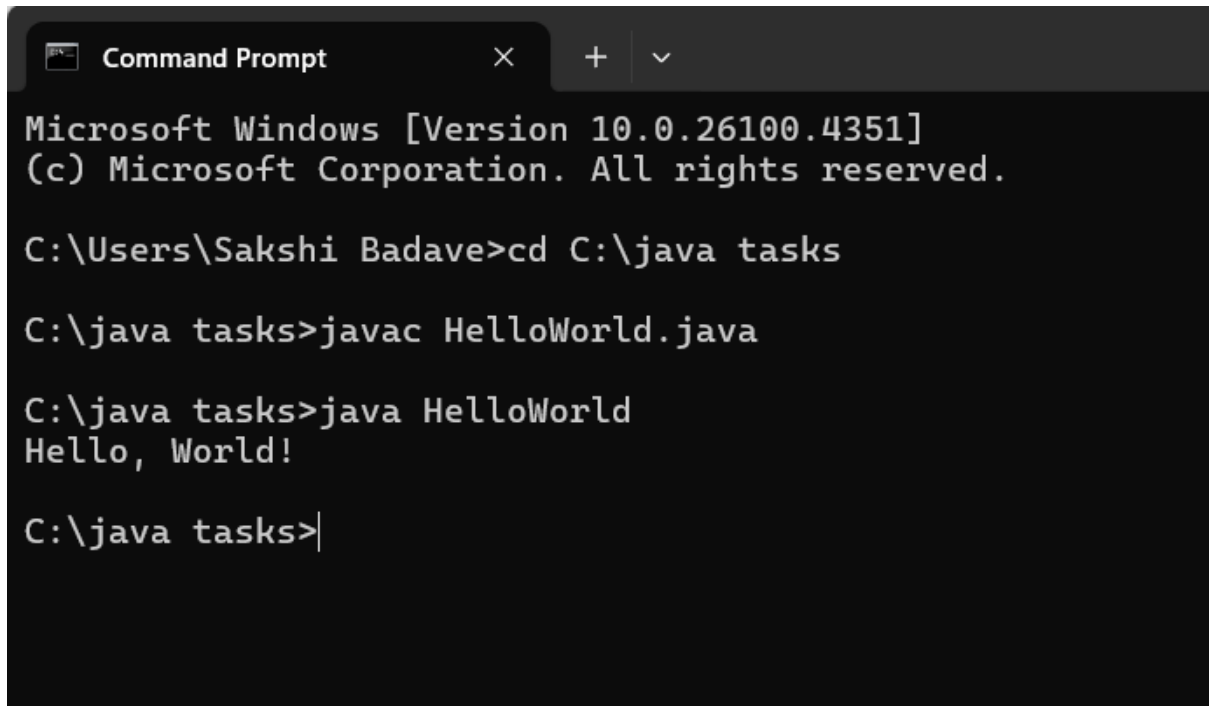


A screenshot of a Windows Command Prompt window. The title bar says 'Command Prompt'. The text in the window is as follows:

```
Microsoft Windows [Version 10.0.26100.4351]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\Sakshi Badave>cd C:\java tasks  
  
C:\java tasks>javac HelloWorld.java  
  
C:\java tasks>
```

3. Loading into JVM:

The Java Class Loader loads the .class file into the Java Virtual Machine (JVM) and execute the code.

A screenshot of a Windows Command Prompt window. The title bar shows 'Command Prompt' with standard window controls. The text inside the window shows the following sequence of commands and output:

```
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Sakshi Badave>cd C:\java tasks

C:\java tasks>javac HelloWorld.java

C:\java tasks>java HelloWorld
Hello, World!

C:\java tasks>|
```

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



```
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.

Data types in Java define the type of data a variable can hold. They tell the compiler what kind of value is expected, how much memory to allocate, and what operations can be performed on that data.

There are two types of data types:

- A. Primitive data types
- B. Non-primitive data types

Primitive data types:

These data types are predefined by java. These are stored directly in a memory. As these data types are predefined by java hence the size of these data types is fixed.

There are 8 primitive data types:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores large whole numbers
float	4 bytes	Stores fractional numbers, up to 7 decimal digits
double	8 bytes	Stores fractional numbers, up to 15 decimal digits
char	2 bytes	Stores a single character (like 'A' or '1')
boolean	1 bit	Stores true or false values

Non-primitive data types:

These data types are defined by users. As these data types are defined by users the size of these data types is not fixed it can be changed according to the need of the users.

Data types

String , Arrays , Class , interface

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

1.JDK:

The JDK is a Java Development Kit. This tool kit provides the tools to develop, compile and run java programs.

2.JRE:

The JRE is a Java Runtime Environment. This provides everything you need to run java programs. But not to develop them.

3.JVM:

The JVM is a Java Virtual Machine. This runs java bytecode on any platform.

Converts the compiled bytecode into machine code specific to the platform.

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

A variable in Java is a container that holds a value that can be changed or used later in the program.

Syntax of declaration of variables:

```
dataType variableName = value;
```

Example:

```
public class Example {  
    public void showAge() {  
        int age = 25;  
        System.out.println("Age: " + age);  
    }  
}
```

Here we declared the variable age and we assigned the value 25 to it hence we can say that we initialize the variable age by value at the time of declaration.

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

1. Arithmetic Operators

Used to perform basic mathematical operations like Addition(+), Subtraction(-), Multiplication(*), Modulo (%), Division(/).

2. Relational Operators

Used to compare two values like Equal to(==), Not equal to (!=), Greater than (>), Less than(<), Greater or equal to(>=), Less or equal to(<=).

3. Logical Operators

Used for combining multiple boolean expressions like Logical AND (&&), Logical OR (||), Logical NOT (!).

4. Assignment Operators

Used to assign values to variables like Assign(=), Add and assign(+=), Subtract and assign(-=), Multiply and assign(*=), Divide and assign(/=), Modulus and assign(%=).

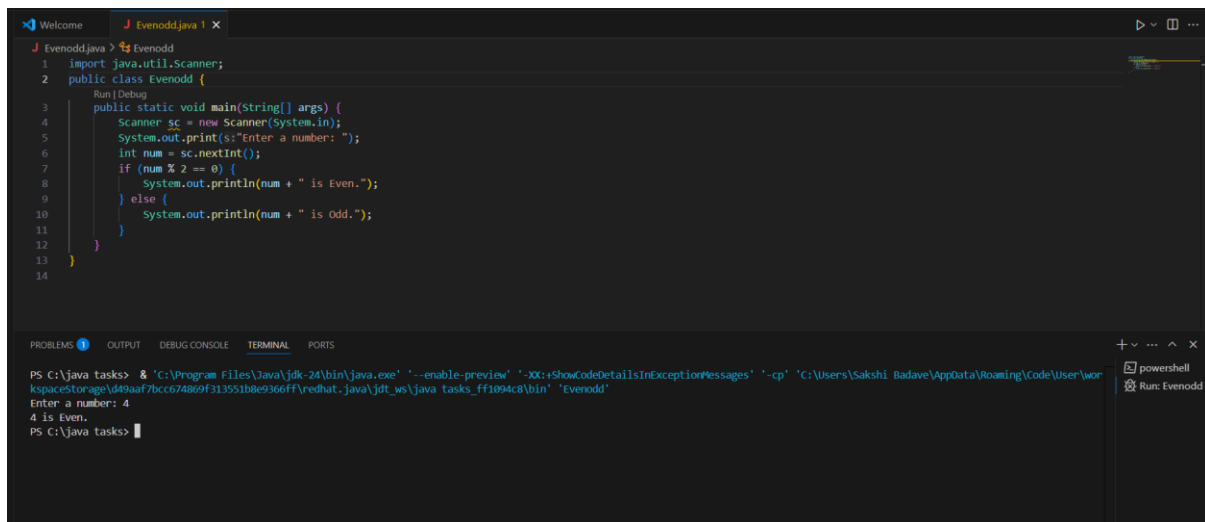
5. Unary Operators

Operate on a single operand like Unary plus(+a), Unary minus(-a), Increment(a++ or ++a), Decrement(a--or --a), Logical NOT(!).

6. Bitwise Operators

Used to perform operations on bits like Bitwise AND (&), Bitwise OR(|) Bitwise XOR (^), Bitwise Complement (~), Left shift (<<) , Right shift (>>)

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



```
1 import java.util.Scanner;
2 public class Evenodd {
3     public static void main(String[] args) {
4         Scanner sc = new Scanner(System.in);
5         System.out.print("Enter a number: ");
6         int num = sc.nextInt();
7         if (num % 2 == 0) {
8             System.out.println(num + " is Even.");
9         } else {
10            System.out.println(num + " is Odd.");
11        }
12    }
13 }
14
```

PS C:\java tasks> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Sakshi_Badave\AppData\Roaming\Code\User\workspaceStorage\d49aaf7bcc674869f313551b8e9366ff\redhat_java\jdt_ws\java_tasks_ff1894c8\bin' 'Evenodd'

Enter a number: 4

4 is Even.

PS C:\java tasks>

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

While loop:

Condition is checked before the loop body is executed.

Do-while loop:

Loop body is executed at least once before the condition is checked.

Object oriented programming

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

Encapsulation: Wrapping data and methods into one unit (class).

Inheritance: Acquiring properties from parent class.

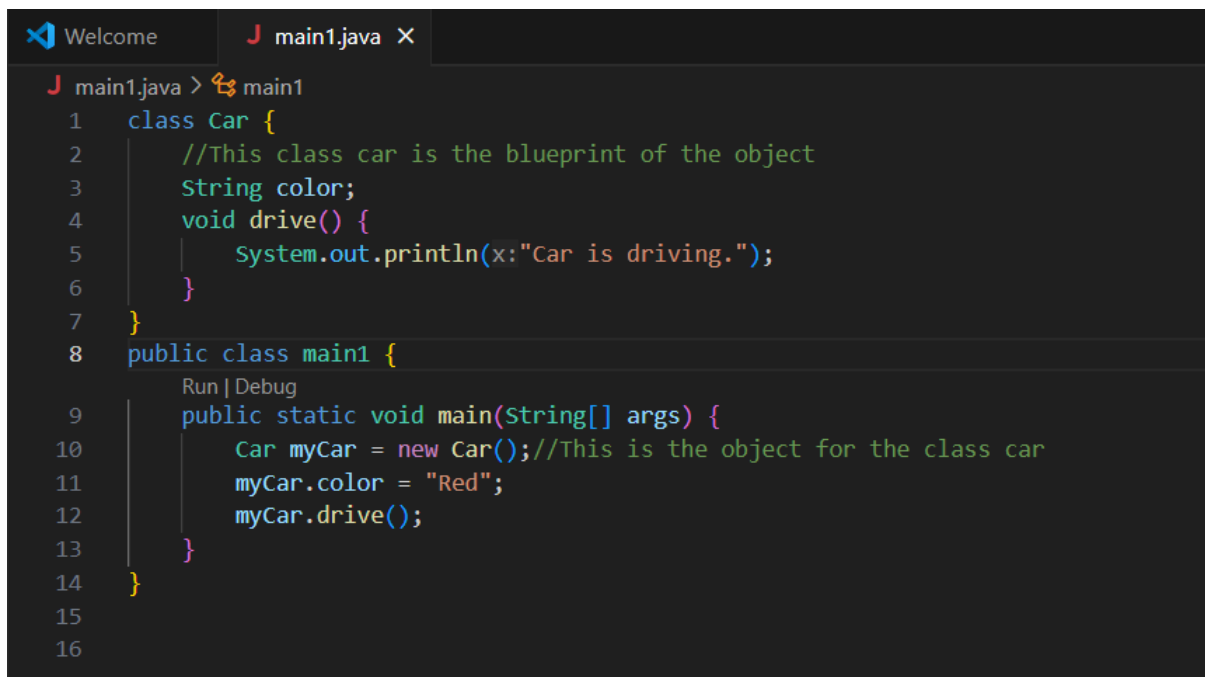
Polymorphism: One action in different forms (overloading/overriding).

Abstraction: Hiding internal details and showing only functionality.

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

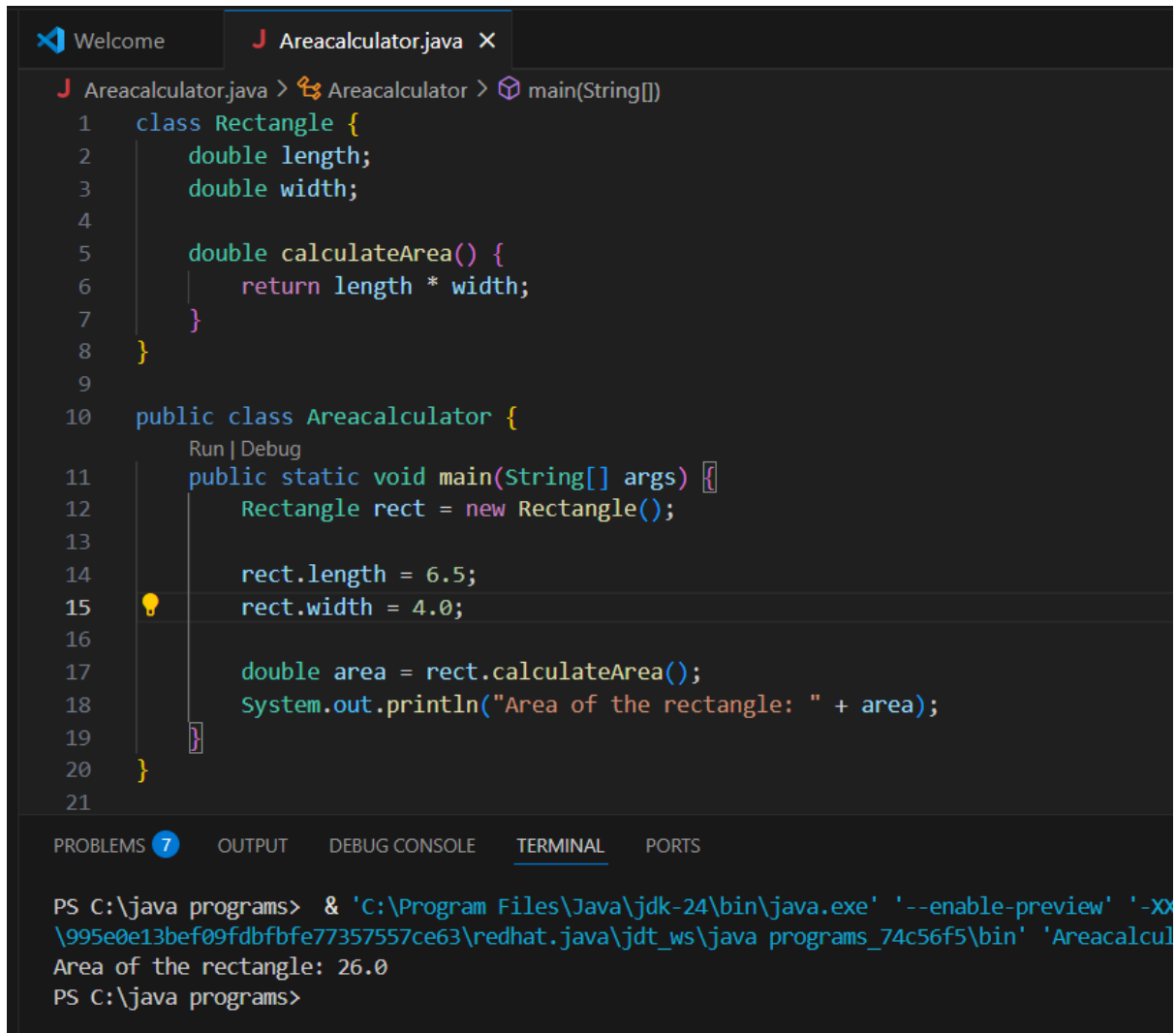
Class: A blueprint for objects.

Object: Instance of a class.



```
main1.java > main1
1  class Car {
2      //This class car is the blueprint of the object
3      String color;
4      void drive() {
5          System.out.println(x:"Car is driving.");
6      }
7  }
8  public class main1 {
9      public static void main(String[] args) {
10         Car myCar = new Car();//This is the object for the class car
11         myCar.color = "Red";
12         myCar.drive();
13     }
14 }
15
16
```

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



```
Areacalculator.java X
Areacalculator.java > Areacalculator > main(String[])
1  class Rectangle {
2      double length;
3      double width;
4
5      double calculateArea() {
6          return length * width;
7      }
8  }
9
10 public class Areacalculator {
11     Run | Debug
12     public static void main(String[] args) {
13         Rectangle rect = new Rectangle();
14
15         rect.length = 6.5;
16         rect.width = 4.0;
17
18         double area = rect.calculateArea();
19         System.out.println("Area of the rectangle: " + area);
20     }
21 }

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\java programs> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-preview' '-XX
\995e0e13bef09fdbfbfe77357557ce63\redhat.java\jdt_ws\java programs_74c56f5\bin' 'Areacalcul
Area of the rectangle: 26.0
PS C:\java programs>
```

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

Imagine a Vehicle class. All vehicles have common properties like speed, fuel, and methods like start() or stop(). A Car is a type of Vehicle, so instead of rewriting all that again, the Car class inherits the features of the Vehicle class.



The image shows a screenshot of an IDE with a dark theme. The top bar has a 'Welcome' tab and an active tab for 'InheritanceExample.java'. The editor displays Java code for a class hierarchy. The 'Vehicle' class is the parent, with a 'brand' attribute and a 'start()' method. The 'Car' class inherits from 'Vehicle', adding a 'model' attribute and a 'displayDetails()' method. The 'InheritanceExample' class contains a 'main()' method that creates a 'Car' object and calls its methods. The bottom panel shows the 'TERMINAL' output, which matches the code's execution: 'Vehicle is starting...', 'Brand: Generic Vehicle', and 'Model: Sedan'.

```
J InheritanceExample.java > ...
1  // Parent class
2  class Vehicle {
3      String brand = "Generic Vehicle";
4
5      void start() {
6          System.out.println("Vehicle is starting...");
7      }
8  }
9
10 // Child class that inherits from Vehicle
11 class Car extends Vehicle {
12     String model = "Sedan";
13
14     void displayDetails() {
15         System.out.println("Brand: " + brand); // inherited
16         System.out.println("Model: " + model); // own property
17     }
18 }
19
20 // Main class to run the code
21 public class InheritanceExample {
22     Run | Debug
23     public static void main(String[] args) {
24         Car myCar = new Car();
25         myCar.start(); // calling inherited method
26         myCar.displayDetails(); // calling child method
27     }
28 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

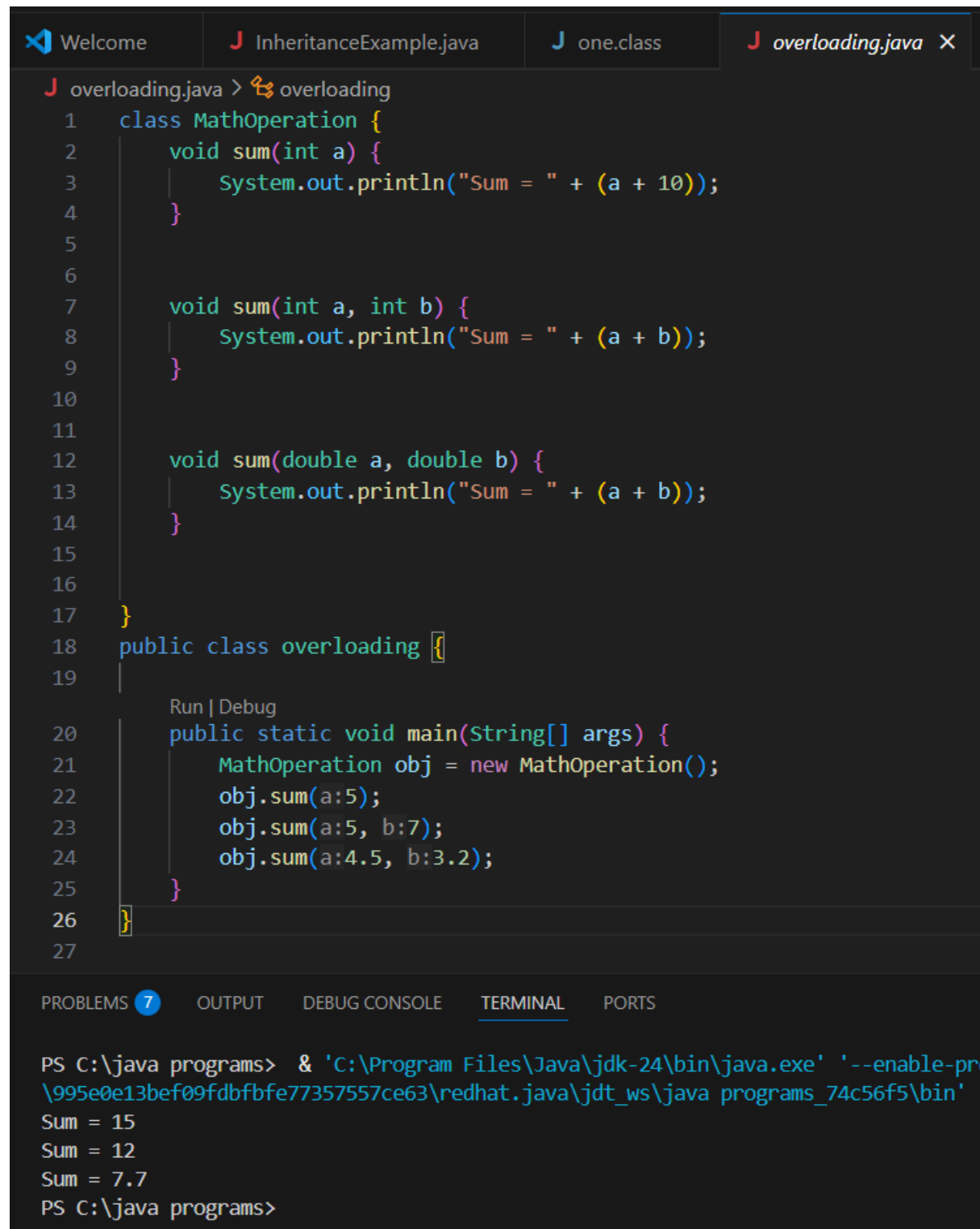
```
PS C:\java programs> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable
\995e0e13bef09fdbfbfe77357557ce63\redhat.java\jdt_ws\java programs_74c56f5\bi
Vehicle is starting...
Brand: Generic Vehicle
Model: Sedan
PS C:\java programs>
```

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

Polymorphism means “many forms” — the ability of an object to take on many behaviours depending on the context.

It allows the same method or operator to behave differently in different situations.

Compile-time Polymorphism (Method Overloading)



```

J overloading.java > overloading
1  class MathOperation {
2      void sum(int a) {
3          System.out.println("Sum = " + (a + 10));
4      }
5
6
7      void sum(int a, int b) {
8          System.out.println("Sum = " + (a + b));
9      }
10
11
12     void sum(double a, double b) {
13         System.out.println("Sum = " + (a + b));
14     }
15
16
17 }
18 public class overloading {
19
20     Run | Debug
21     public static void main(String[] args) {
22         MathOperation obj = new MathOperation();
23         obj.sum(a:5);
24         obj.sum(a:5, b:7);
25         obj.sum(a:4.5, b:3.2);
26     }
27 }

```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\java programs> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-pr
\995e0e13bef09fdbfbfe77357557ce63\redhat.java\jdt_ws\java programs_74c56f5\bin'
Sum = 15
Sum = 12
Sum = 7.7
PS C:\java programs>

```

Runtime Polymorphism (Method Overriding)



```

Welcome  RuntimeExample.java X

RuntimeExample.java > ...
1  class Animal {
2      void sound() {
3          System.out.println(x:"Animal makes a sound");
4      }
5  }
6
7  class Dog extends Animal {
8      void sound() {
9          System.out.println(x:"Dog barks");
10     }
11 }
12
13 class Cat extends Animal {
14     void sound() {
15         System.out.println(x:"Cat meows");
16     }
17 }
18
19 public class RuntimeExample {
20     Run | Debug
21     public static void main(String[] args) {
22         Animal a;
23
24         a = new Dog();    // Upcasting
25         a.sound();        // Output: Dog barks
26
27         a = new Cat();    // Upcasting
28         a.sound();        // Output: Cat meows
29     }
30 }

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

\995e0e13bef09fdbfbfe77357557ce63\redhat.java\jdt_ws\java programs_74c56f5\b
Dog barks
Cat meows
PS C:\java programs>
```

The image shows an IDE window with a Java file named `RuntimeExample.java`. The code defines three classes: `Animal`, `Dog`, and `Cat`. `Animal` has a `sound()` method that prints "Animal makes a sound". `Dog` and `Cat` both extend `Animal` and override the `sound()` method to print "Dog barks" and "Cat meows" respectively. The `RuntimeExample` class contains a `main` method that demonstrates runtime polymorphism by creating `Dog` and `Cat` objects and assigning them to an `Animal` reference variable `a`. This is referred to as "upcasting". When `a.sound()` is called, the output is "Dog barks" for the first instance and "Cat meows" for the second, showing that the method from the actual object's class is executed. The terminal at the bottom shows the execution output and the command prompt.

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

Method Overloading

Method Overloading means having multiple methods in the same class with the same name but different parameters (number, type, or order).



The screenshot shows an IDE with three tabs: 'Welcome', 'InheritanceExample.java', and 'one.class'. The active tab is 'overloading.java'. The code defines a `MathOperation` class with three `sum` methods: one for a single integer, one for two integers, and one for two doubles. A `main` method in the `overloading` class tests these methods with specific values. The terminal output shows the results of these calls: 'Sum = 15', 'Sum = 12', and 'Sum = 7.7'.

```
overloading.java > overloading
1  class MathOperation {
2      void sum(int a) {
3          System.out.println("Sum = " + (a + 10));
4      }
5
6
7      void sum(int a, int b) {
8          System.out.println("Sum = " + (a + b));
9      }
10
11
12     void sum(double a, double b) {
13         System.out.println("Sum = " + (a + b));
14     }
15
16
17 }
18 public class overloading {
19
20     Run | Debug
21     public static void main(String[] args) {
22         MathOperation obj = new MathOperation();
23         obj.sum(a:5);
24         obj.sum(a:5, b:7);
25         obj.sum(a:4.5, b:3.2);
26     }
27 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\java programs> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-pr
\995e0e13bef09fdbfbfe77357557ce63\redhat.java\jdt_ws\java programs_74c56f5\bin'
Sum = 15
Sum = 12
Sum = 7.7
PS C:\java programs>
```

Method Overriding

Method Overriding means a subclass provides a specific implementation of a method that is already defined in its superclass.

Welcome

RuntimeExample.java X

RuntimeExample.java > ...

```
1  class Animal {
2      void sound() {
3          System.out.println(x:"Animal makes a sound");
4      }
5  }
6
7  class Dog extends Animal {
8      void sound() {
9          System.out.println(x:"Dog barks");
10     }
11 }
12
13 class Cat extends Animal {
14     void sound() {
15         System.out.println(x:"Cat meows");
16     }
17 }
18
19 public class RuntimeExample {
20     Run | Debug
21     public static void main(String[] args) {
22         Animal a;
23
24         a = new Dog();    // Upcasting
25         a.sound();        // Output: Dog barks
26
27         a = new Cat();    // Upcasting
28         a.sound();        // Output: Cat meows
29     }
30 }
```

PROBLEMS 7

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
\995e0e13bef09fdbfbfe77357557ce63\redhat.java\jdt_ws\java programs_74c56f5\b
Dog barks
Cat meows
PS C:\java programs>
```


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

```

Welcome | EncapsulationExample.java X
EncapsulationExample.java > Employee
1  class Employee {
2      private String name;
3      private int age;
4
5      public String getName() {
6          return name;
7      }
8
9      public void setName(String newName) {
10         name = newName;
11     }
12
13     public int getAge() {
14         return age;
15     }
16
17     public void setAge(int newAge) {
18         if (newAge > 0) {
19             age = newAge;
20         } else {
21             System.out.println("Invalid age!");
22         }
23     }
24 }
25
26 public class EncapsulationExample {
27     Run | Debug
28     public static void main(String[] args) {
29         Employee emp = new Employee();
30
31         emp.setName(newName:"Sakshi");
32         emp.setAge(newAge:19); // Age updated here
33
34         System.out.println("Employee Name: " + emp.getName());
35         System.out.println("Employee Age: " + emp.getAge());
36     }
37 }
```

```
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\java programs> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-preview'
\995e0e13bef09fdbfbfe77357557ce63\redhat.java\jdt_ws\java programs_74c56f5\bin' 'Encaps
Employee Name: Sakshi
Employee Age: 19
PS C:\java programs>
```

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

Abstraction is the process of hiding the internal implementation details and showing only the essential features of an object.

- To simplify complex systems by exposing only the necessary parts.
- To reduce code complexity and improve security.
- Helps in focusing on what an object does, not how it does it.

Java supports abstraction using:

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

9. Explain the difference between abstract class and interface.

1. Keywords

- **Abstract Class:** Declared using abstract class
- **Interface:** Declared using interface

2. Methods

- **Abstract Class:** Can have both abstract and non-abstract (concrete) methods

- **Interface:** Only abstract methods by default (till Java 7); from Java 8+, can have default, static, and private methods

3. Variables

- **Abstract Class:** Can have variables with any access modifier (private, protected, public)
- **Interface:** Variables are public static final by default (constants)

4. Constructors

- **Abstract Class:** Can have constructors
- **Interface:** Cannot have constructors

5. Inheritance

- **Abstract Class:** Supports single inheritance only
- **Interface:** Supports multiple inheritance (a class can implement multiple interfaces)

6. Accessibility Modifiers

- **Abstract Class:** Methods and variables can use any access modifier
- **Interface:** All methods are public by default

7. Use Case

- **Abstract Class:** Use when classes are closely related or share common base functionality
- **Interface:** Use when different classes need to follow a common contract

8. Level of Abstraction

- **Abstract Class:** Provides partial abstraction
- **Interface:** Provides full abstraction

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



```
InterfaceDemo.java > ...
1  // Define an interface
2  interface Drawable {
3      void draw(); // abstract method
4  }
5
6  // Implement the interface in a class
7  class Circle implements Drawable {
8      public void draw() {
9          System.out.println(x:"Drawing a Circle");
10     }
11 }
12
13 // Another class implementing the interface
14 class Rectangle implements Drawable {
15     public void draw() {
16         System.out.println(x:"Drawing a Rectangle");
17     }
18 }
19
20 // Main class to test interface
21 public class InterfaceDemo {
22     Run | Debug
23     public static void main(String[] args) {
24         Drawable d1 = new Circle(); // Using interface reference
25         Drawable d2 = new Rectangle(); // Polymorphism
26
27         d1.draw(); // Output: Drawing a Circle
28         d2.draw(); // Output: Drawing a Rectangle
29     }
30 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
\995e0e13bef09fdbfbfe77357557ce63\redhat.java\jdt_ws\java programs_74c56f5\bin' 'Inte
Drawing a Circle
Drawing a Rectangle
PS C:\java programs>
```