JAVA BASICS AND OOPs ASSIGNMENT QUESTIONS

NAME: SHRAVANI DESAI

AF ID: AF04953349

DATE: 26 JUNE 2025

1.JAVA BASICS

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

ANSWER:

**Java** is a **high-level, object-oriented, and platform-independent** programming language developed by **Sun Microsystems** in **1995**. It follows the principle of **"Write Once, Run Anywhere"**, meaning compiled Java code can run on any system with a **Java Virtual Machine (JVM)**.

Key Features:

**Platform Independent** – Java code runs on any device using the JVM.

**Object-Oriented** – Follows OOP principles like inheritance and polymorphism.

**Simple** – Easy to learn with a clean and readable syntax.

**Secure** – Provides built-in security features and avoids unsafe operations.

**Robust** – Strong memory management and exception handling.

**Multithreaded** – Supports multiple threads for concurrent execution.

**High Performance** – Uses JIT compiler to improve execution speed.

**Distributed** – Supports network-based programming (e.g., RMI).

**Dynamic** – Loads classes at runtime and supports reflection.

Q.2. Explain the Java program execution process.

ANSWER:

The execution of a Java program involves several steps from writing code to running it. Here's a clear explanation of the **Java program execution process**:

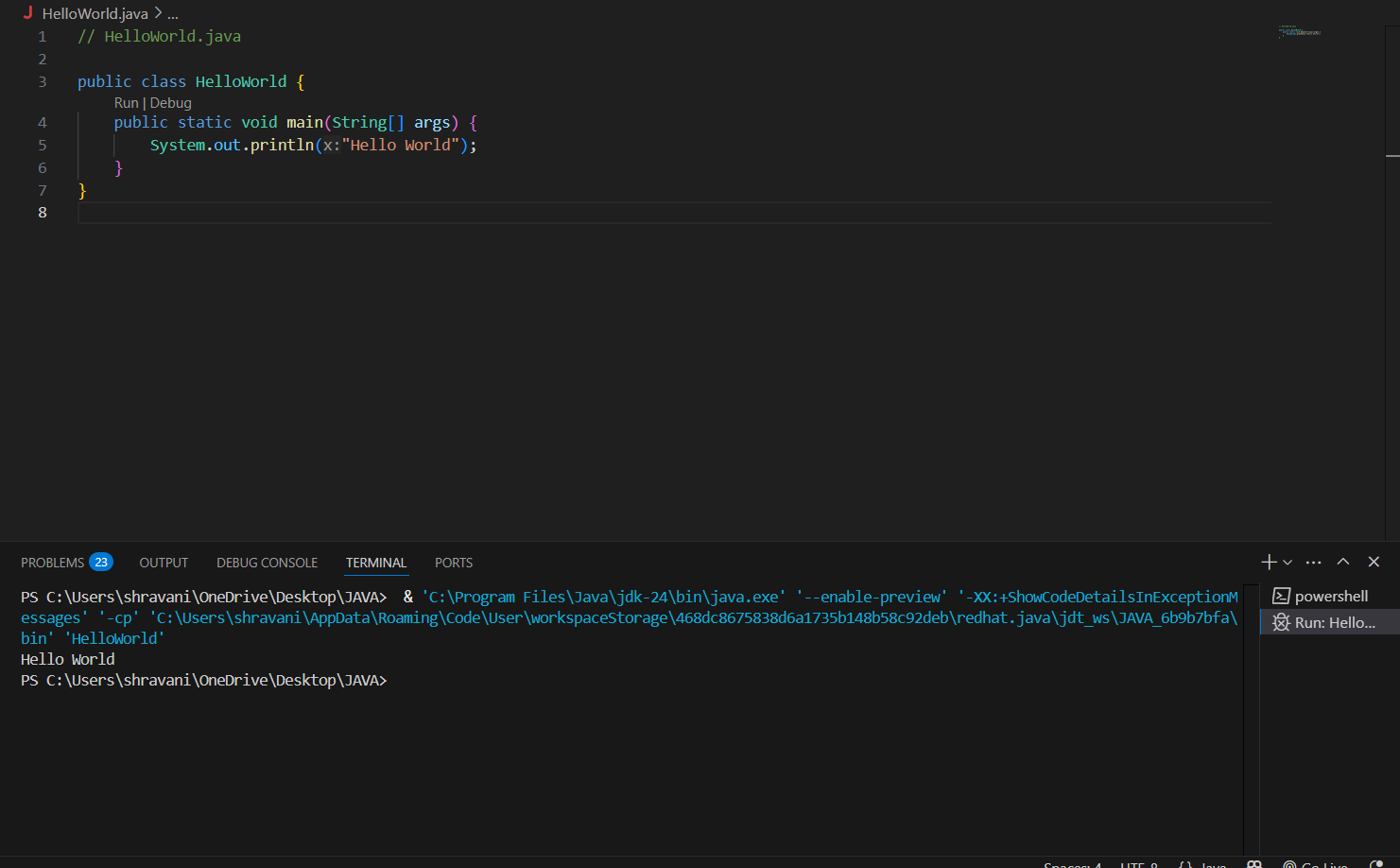
**🔄 Step-by-Step Execution Process:**

1. **Writing the Code**
   * Java code is written in a .java file using a text editor or IDE like IntelliJ or Eclipse.
2. **Compilation**
   * The .java file is compiled using the **Java Compiler (javac)**, which converts the source code into **bytecode** and stores it in a .class file.
   * Example: javac HelloWorld.java → generates HelloWorld.class
3. **Bytecode**
   * This .class file contains **platform-independent bytecode**, which is not readable by the machine directly.
4. **Class Loader**
   * The **Class Loader** loads the .class file into memory when you run the program.
5. **Bytecode Verification**
   * The **Bytecode Verifier** checks the bytecode for security and correctness before execution.
6. **Java Virtual Machine (JVM)**
   * The JVM interprets or compiles the bytecode into **machine code** specific to the operating system and hardware.
7. **Execution**
   * Finally, the machine code is executed by the **CPU**, and the program runs with output shown in the console.

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

ANSWER:

Code & Output given



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

ANSWER:

In Java, **data types** define the type of data a variable can hold. They determine the size, kind of values, and operations allowed on the data. Java has two main types of data types:

**1. Primitive Data Types:**

Java provides 8 built-in primitive data types:

| **Data Type** | **Description** | **Size** | **Example** |
| --- | --- | --- | --- |
| byte | Small integer value | 1 byte | byte a = 10; |
| short | Short-range integer | 2 bytes | short s = 1000; |
| int | Default integer type | 4 bytes | int i = 50000; |
| long | Large integer values | 8 bytes | long l = 100000L; |
| float | Decimal number (single precision) | 4 bytes | float f = 5.6f; |
| double | Decimal number (double precision) | 8 bytes | double d = 99.99; |
| char | Single Unicode character | 2 bytes | char c = 'A'; |
| boolean | Logical values (true/false) | 1 bit | boolean b = true; |

**2. Non-Primitive (Reference) Data Types:**

These refer to objects and can store multiple values or complex data. They include:

* **String** – Stores a sequence of characters  
  String name = "Java";
* **Array** – Stores multiple values of the same type  
  int[] numbers = {1, 2, 3};
* **Class** – User-defined blueprint for objects  
  Student s = new Student();
* **Interface** – Defines abstract methods for classes to implement

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

ANSWER:

| **Component** | **Full Form** | **Description** | **Contains** |
| --- | --- | --- | --- |
| **JVM** | Java Virtual Machine | JVM is a runtime environment that executes Java bytecode (.class files). | Only the **engine** to run Java code |
| **JRE** | Java Runtime Environment | JRE provides libraries and JVM to run Java applications (but not develop). | **JVM + Core Libraries + Tools** |
| **JDK** | Java Development Kit | JDK is a complete package for Java development including JRE and compilers. | **JRE + javac + debugger + tools** |

**Simple Explanation:**

* **JVM**: Executes Java programs. It is **platform-dependent** but provides **platform independence** to Java.
* **JRE**: Allows you to **run** Java applications but not write or compile them.
* **JDK**: Needed to **develop** Java programs. It includes everything in JRE plus development tools.

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

ANSWER:

In Java, a **variable** is a **name given to a memory location** that stores a value. It is used to store data that can be used and modified during program execution.

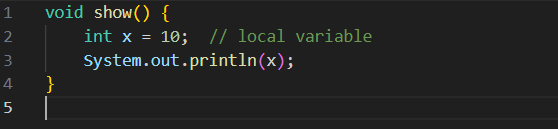
**Types of Variables in Java:**

**2.Local Variable**

Declared inside a method or block.

Scope is limited to that method or block.

Example:

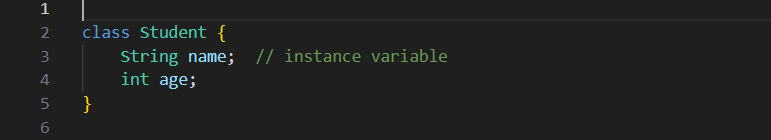


**2**.**Instance Variable**

Declared inside a class but outside any method.

Each object has its own copy.

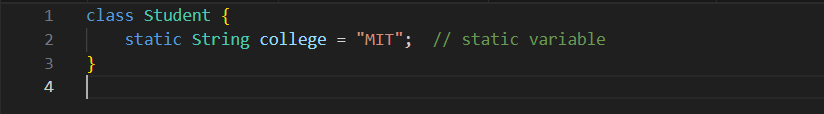
Example:

  
**3.Static Variable**

Declared using the static keyword.

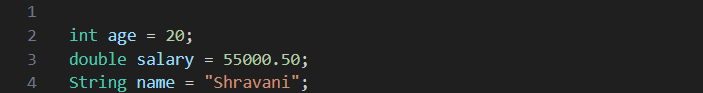
Shared among all objects of the class.

Example:



Variable Declaration Syntax:





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

ANSWER:

In Java, **operators** are special symbols used to perform operations on variables and values. Java supports several types of operators:

**1. Arithmetic Operators**

Used to perform basic mathematical operations.

* + : Addition (a + b)
* - : Subtraction (a - b)
* \* : Multiplication (a \* b)
* / : Division (a / b)
* % : Modulus (remainder) (a % b)

**2. Relational (Comparison) Operators**

Used to compare two values.

* == : Equal to (a == b)
* != : Not equal to (a != b)
* > : Greater than (a > b)
* < : Less than (a < b)
* >= : Greater than or equal to (a >= b)
* <= : Less than or equal to (a <= b)

**3. Logical Operators**

Used to combine multiple conditions.

* && : Logical AND (a > 5 && b < 10)
* || : Logical OR (a > 5 || b < 10)
* ! : Logical NOT (!(a == b))

**4. Assignment Operators**

Used to assign values to variables.

* = : Assign (a = b)
* += : Add and assign (a += b → a = a + b)
* -= : Subtract and assign (a -= b)
* \*= : Multiply and assign (a \*= b)
* /= : Divide and assign (a /= b)
* %= : Modulus and assign (a %= b)

**5. Unary Operators**

Operate on a single operand.

* + : Unary plus (+a)
* - : Unary minus (-a)
* ++ : Increment (a++ or ++a)
* -- : Decrement (a-- or --a)
* ! : Logical NOT (!true)

**6. Bitwise Operators**

Operate at the bit level.

* & : Bitwise AND (a & b)
* | : Bitwise OR (a | b)
* ^ : Bitwise XOR (a ^ b)
* ~ : Bitwise Complement (~a)
* << : Left shift (a << 2)
* >> : Right shift (a >> 2)

**7.Ternary Operators**

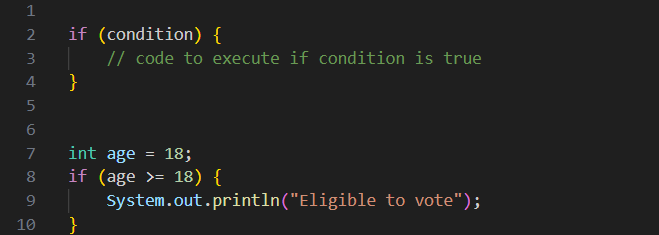
* Used as a shortcut for if-else conditions.
* Syntax: condition ? value\_if\_true : value\_if\_false;

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

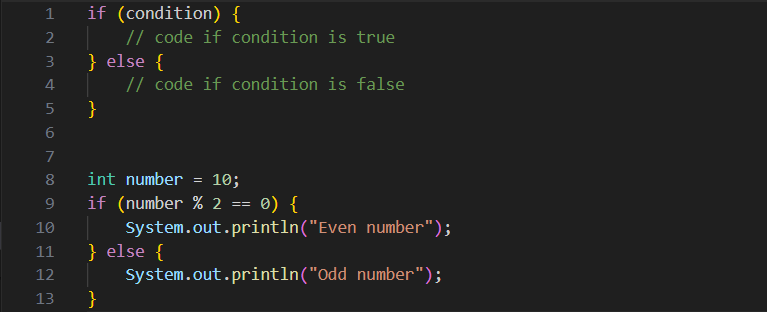
ANSWER:

In Java, **control statements** are used to control the flow of execution based on certain conditions. The most commonly used decision-making control statements are:

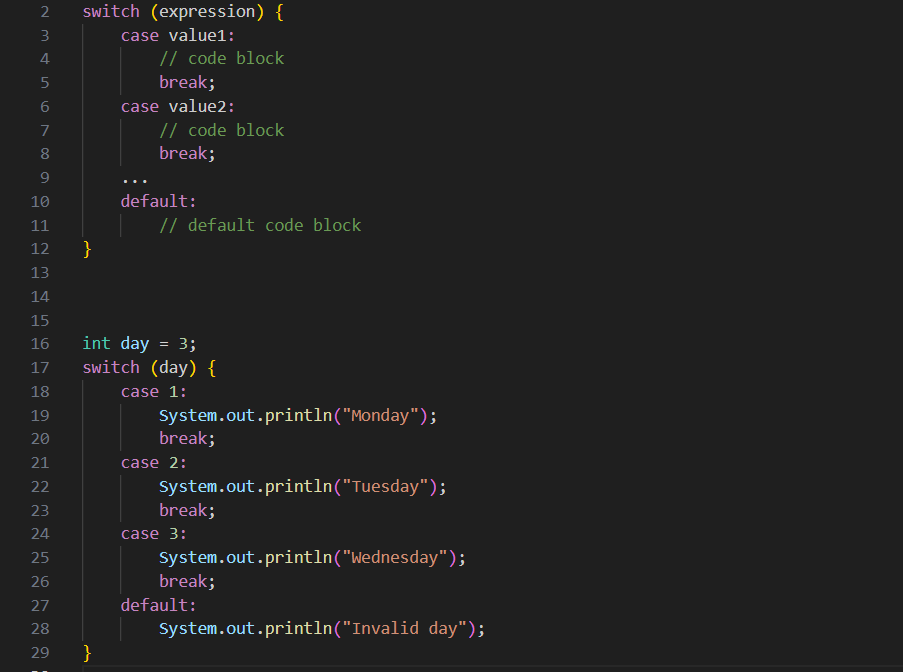
**1.if Statement**

****

**2. if-else Statement**

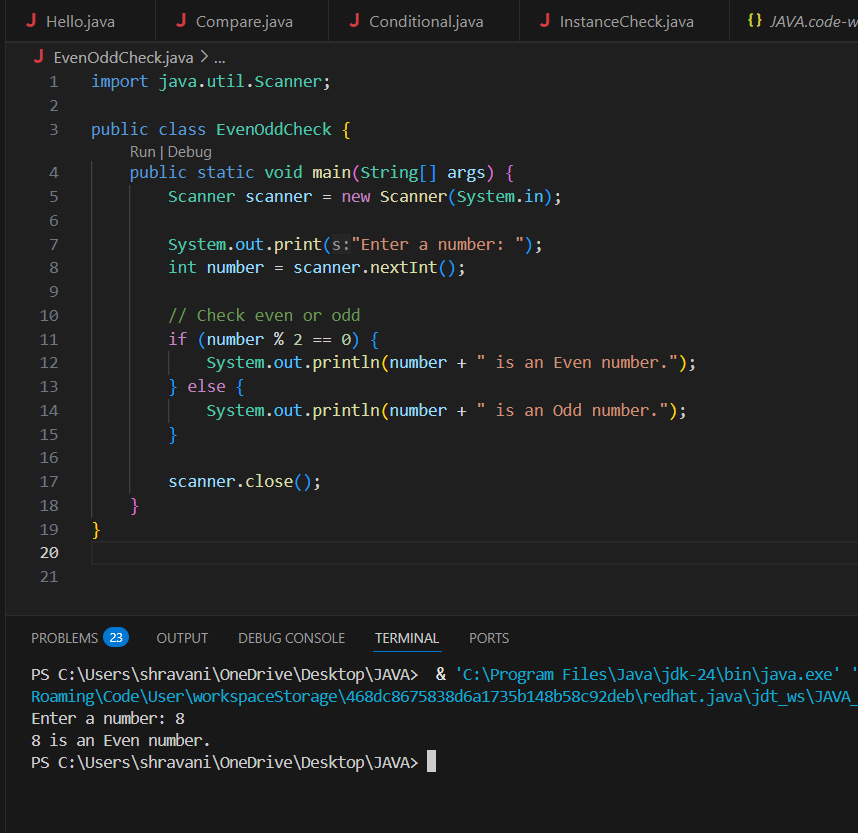
****

**3. switch Statement**

****

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

ANSWER:

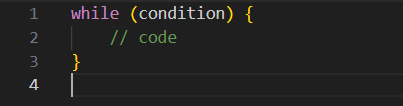


Q.10 What is the difference between while and do-while loop?

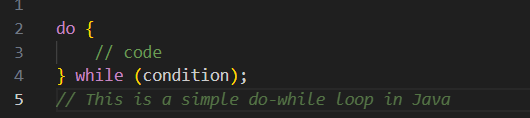
ANSWER:

| **Feature** | **while Loop** | **do-while Loop** |
| --- | --- | --- |
| **Condition Check** | Condition is checked **before** the loop runs | Condition is checked **after** the loop runs |
| **Minimum Execution** | May **not execute** even once if condition is false | Executes **at least once** even if condition is false |

SYNTAX FOR WHILE LOOP:



SYNTAX FOR DO WHILE LOOP:



**2. Object-Oriented Programming (OOPs)**

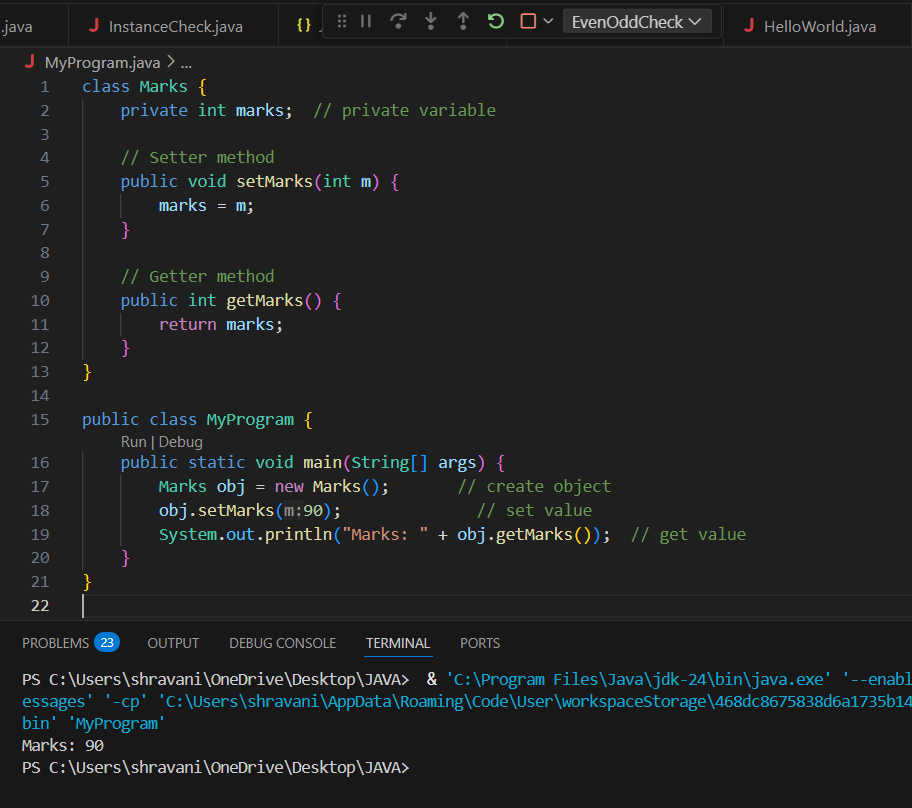
Q.11 . What are the main principles of OOPs in Java? Explain each.

ANSWER:

Java is an **Object-Oriented Programming (OOP)** language. The four main principles of OOP in Java are:

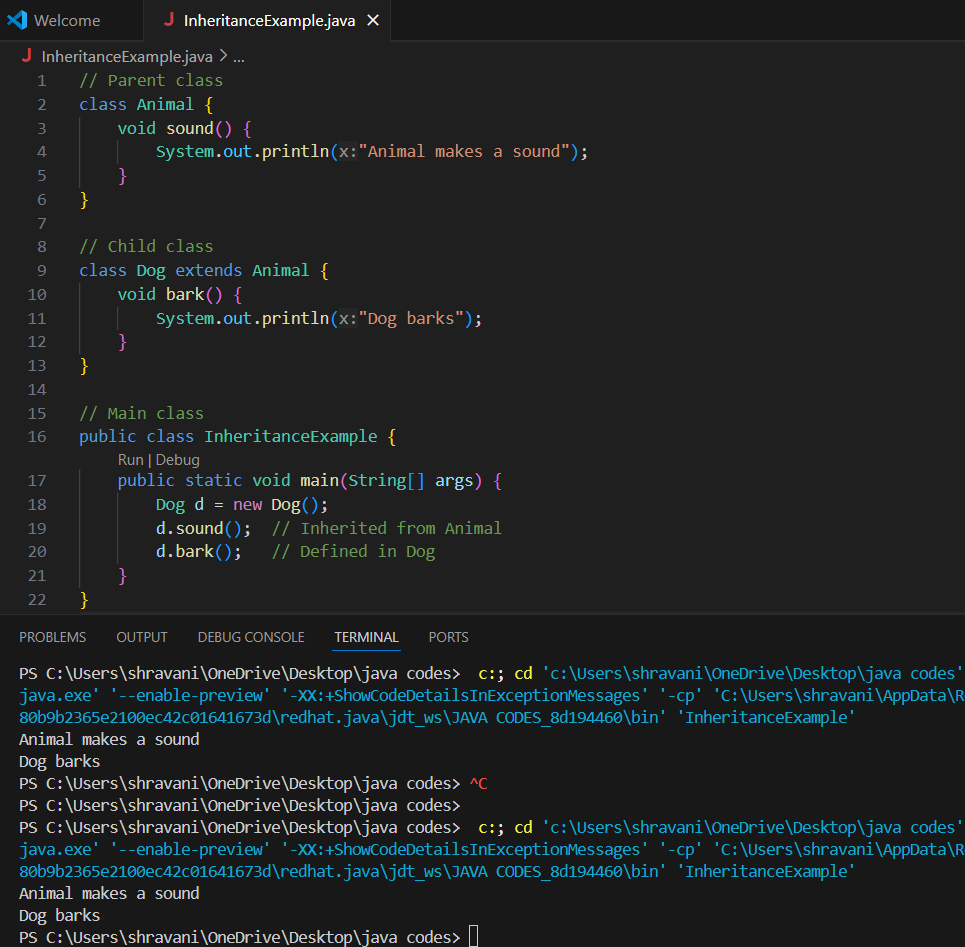
**1. Encapsulation**

Encapsulation is the process of **wrapping data (variables) and code (methods)** into a single unit, called a **class**. It helps to protect data from unauthorized access using **private access modifiers** and provides public methods (getters/setters) to access or modify the data.



**2. Inheritance**

Inheritance allows a class (**subclass**) to inherit properties and behaviors (fields and methods) from another class (**superclass**). It promotes **code reuse** and represents **IS-A** relationships.

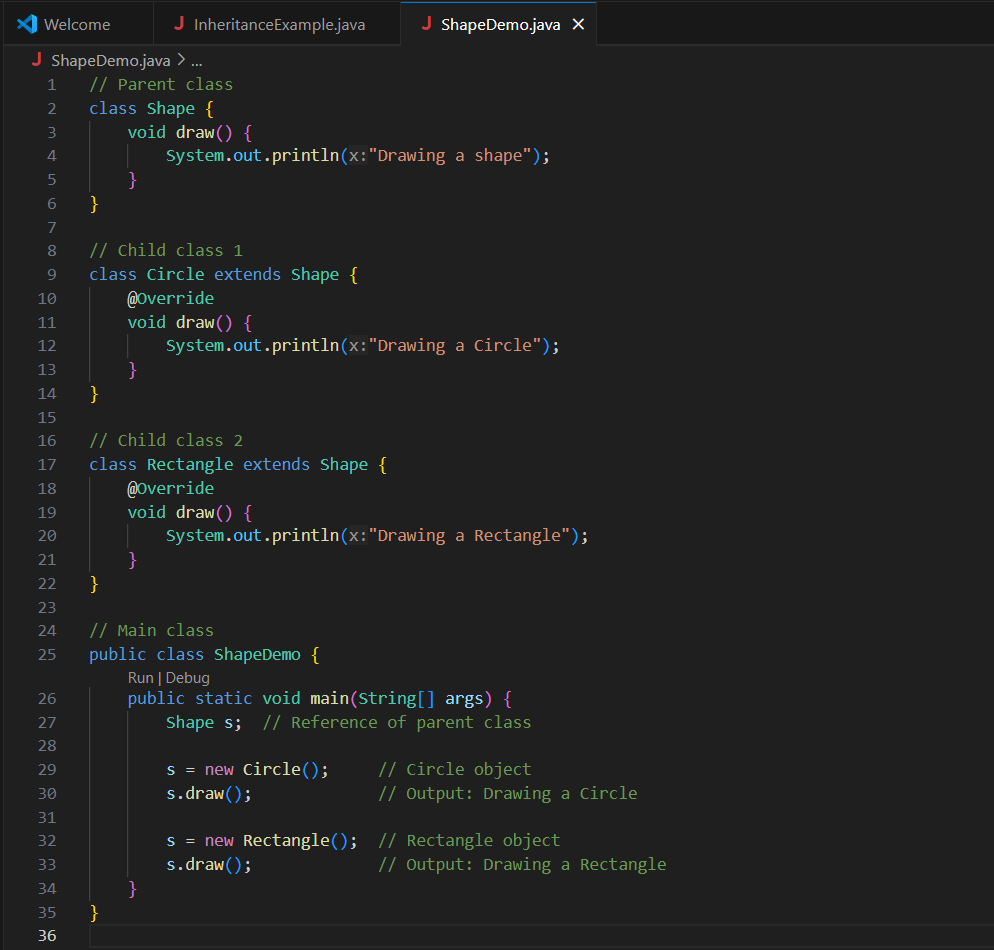


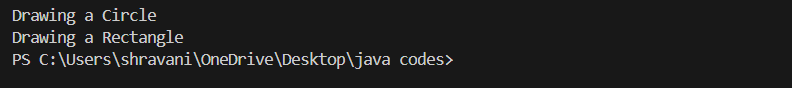
**3. Polymorphism**

Polymorphism means **one name, many forms**. It allows the same method or function to behave differently based on the object that calls it. It is of two types:

**Compile-time Polymorphism** (Method Overloading)

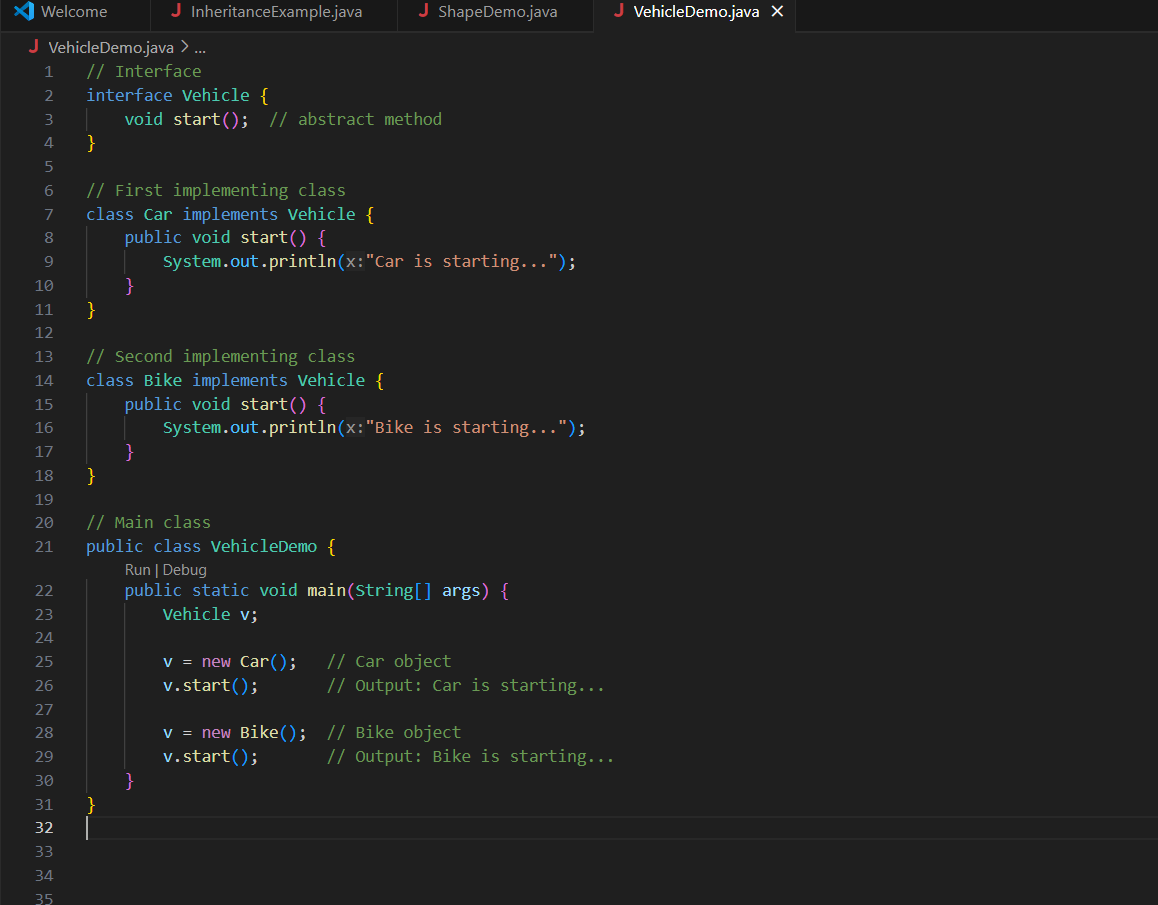
**Runtime Polymorphism** (Method Overriding)

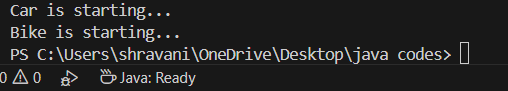




**4. Abstraction**

Abstraction is the process of **hiding internal details** and showing only essential features. It can be achieved using **abstract classes** or **interfaces** in Java.





Q.12 What is a class and an object in Java? Give examples.

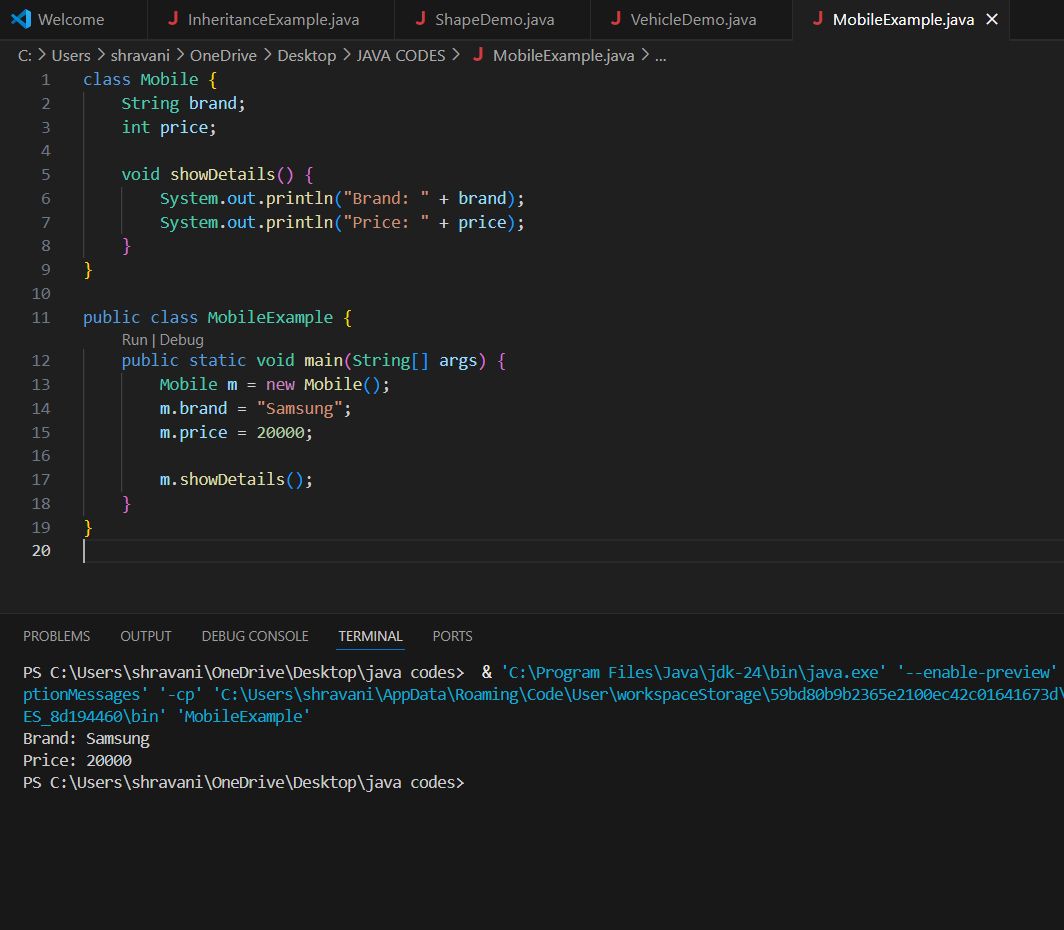
ANSWER:

**🔹 Class**

A **class** is a blueprint or template for creating objects. It defines the structure (data/variables) and behavior (methods) of objects.

**🔹 Object**

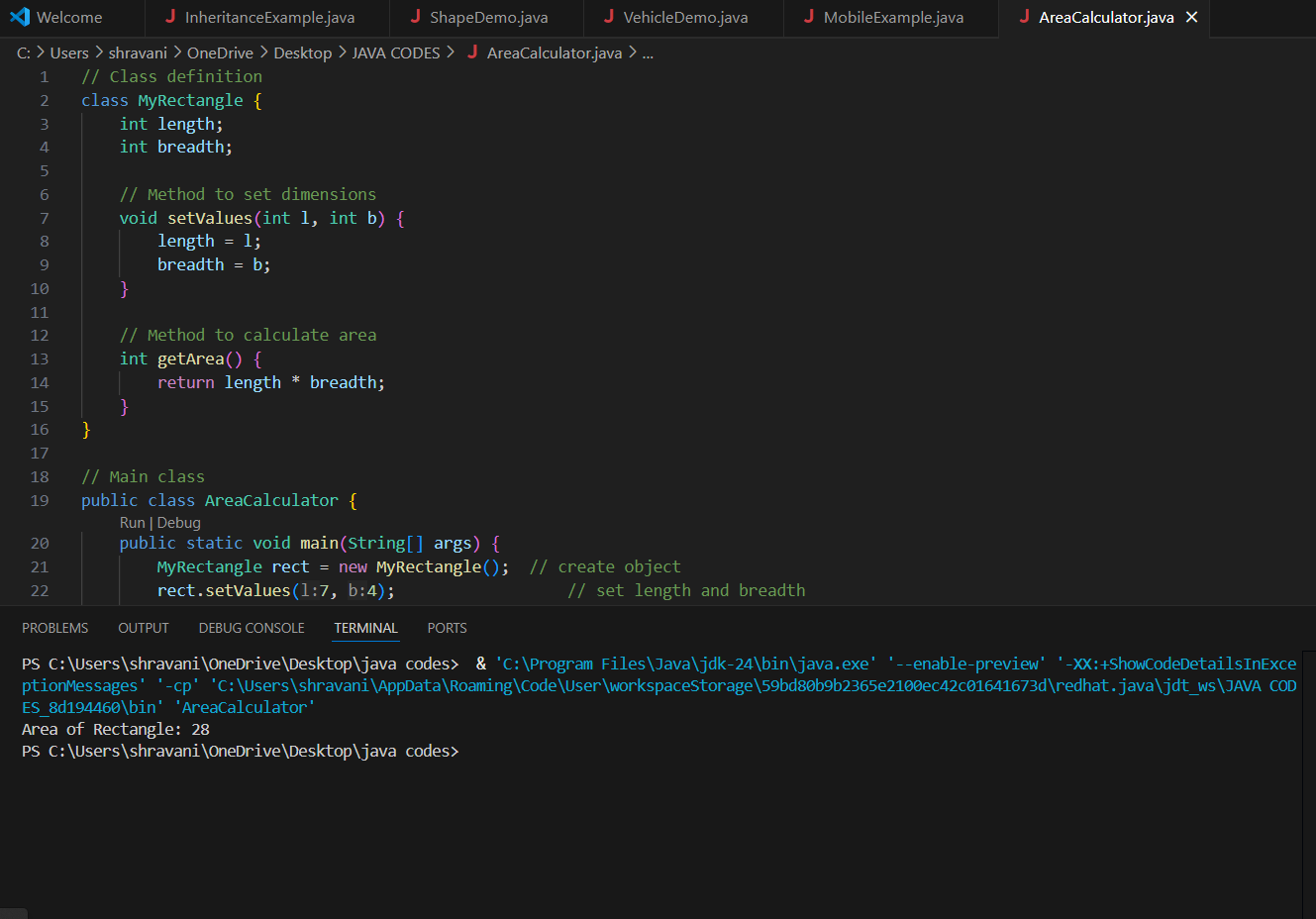
An **object** is an instance of a class. It has **state** (data) and **behavior** (methods). Multiple objects can be created from one class.



| **Term** | **Meaning** |
| --- | --- |
| Class | Blueprint for objects |
| Object | Real-world instance of the class |

Q.13. Write a program using class and object to calculate area of a rectangle.

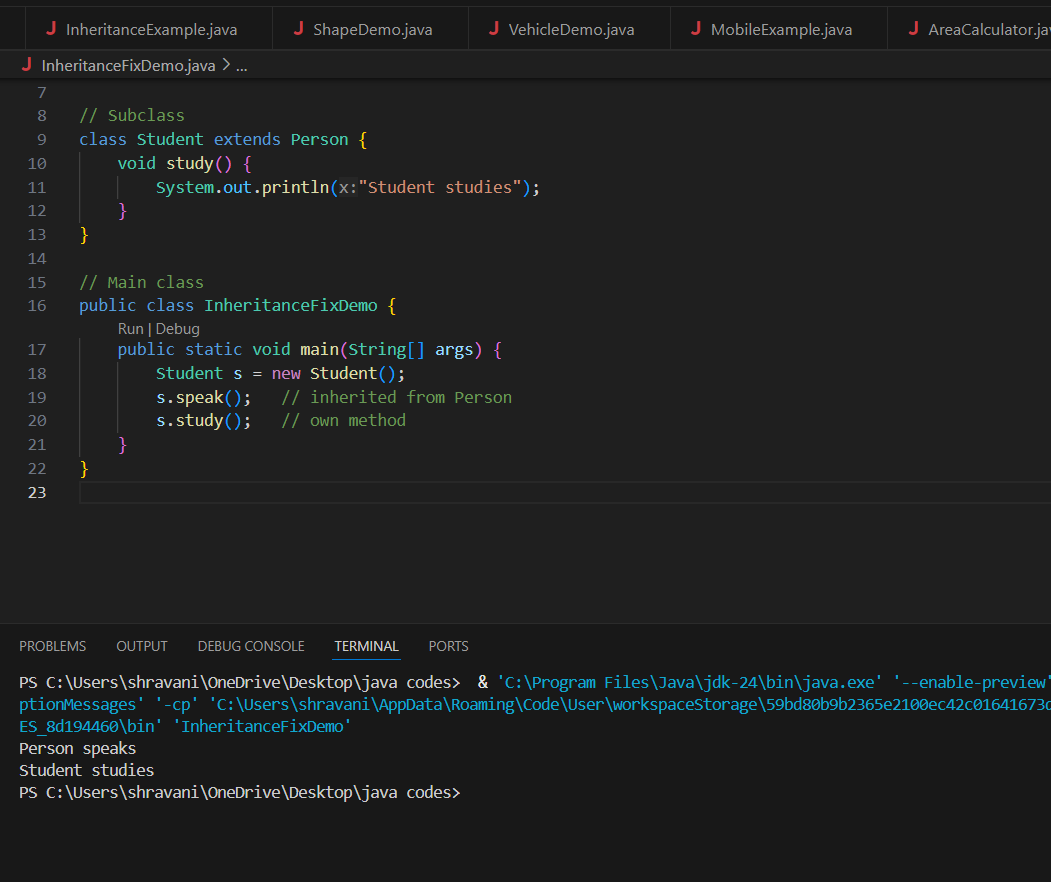
ANSWER:



Q.14 . Explain inheritance with real-life example and Java code.

ANSWER:

Inheritance is a concept in Java where **one class (child)** acquires the **properties and behaviors** (fields and methods) of **another class (parent)**.



Q.15 What is polymorphism? Explain with compile-time and runtime examples.

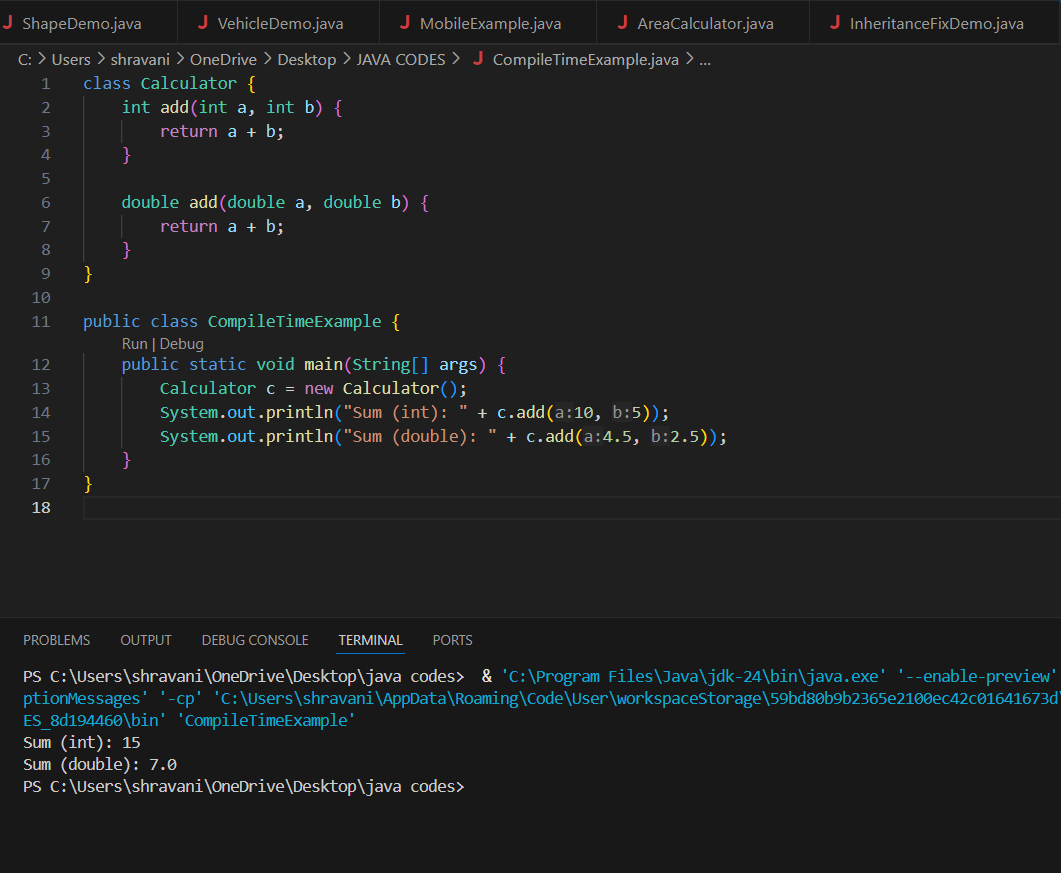
ANSWER:

**Polymorphism** in Java means **"many forms"**. It allows the same method or operation to behave differently based on the object or context.

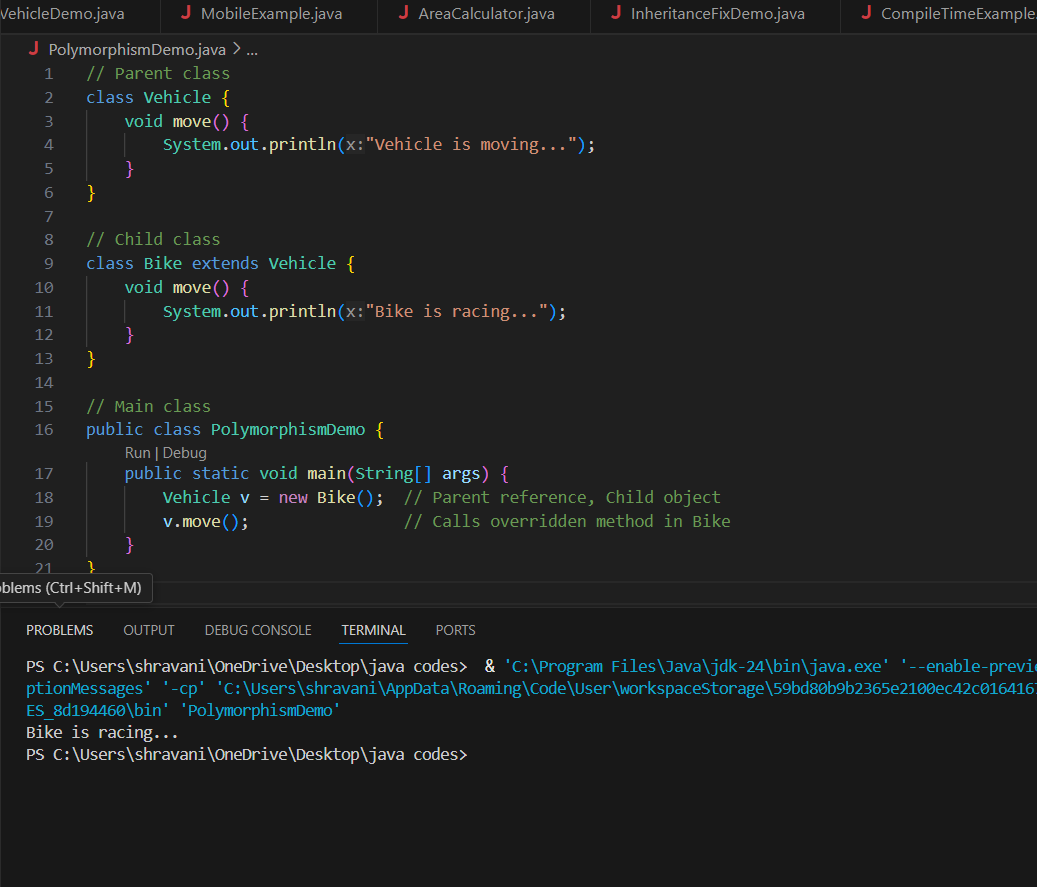
**🔸 Types of Polymorphism:**

| **Type** | **Also Called** | **Achieved By** |
| --- | --- | --- |
| Compile-time | Static Polymorphism | Method Overloading |
| Runtime | Dynamic Polymorphism | Method Overriding |

**Compile-Time Polymorphism (Method Overloading)**



**Runtime Polymorphism (Method Overriding)**



**Summary:**

| **Type** | **Polymorphism** | **How?** | **Decided At** |
| --- | --- | --- | --- |
| Compile-time | Overloading | Same method, diff args | Compile time |
| Runtime | Overriding | Inherited method, redefined | Run time |

Q.16 What is method overloading and method overriding? Show with examples

ANSWER:

**1. Method Overloading (Compile-Time Polymorphism)**

**Definition:**  
Multiple methods with the **same name** but **different parameters** (number or type) in the **same class**.

**Method Overriding (Runtime Polymorphism)**

**Definition:**  
When a **subclass provides its own version** of a method that is already defined in the **superclass**.

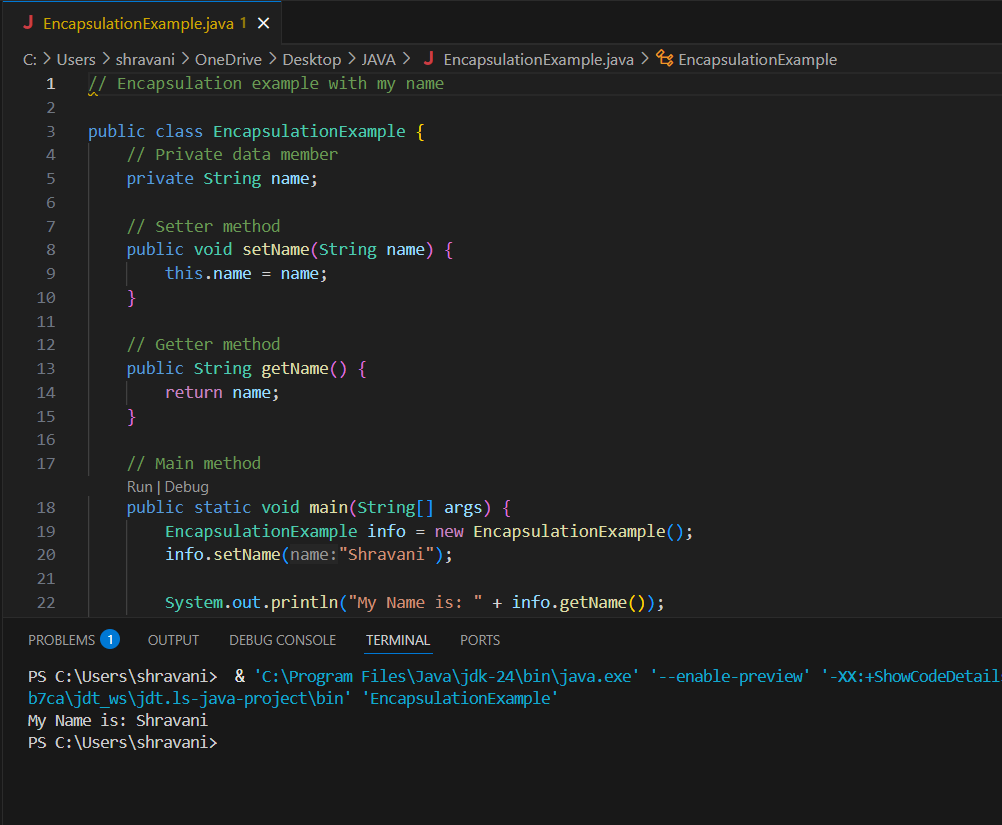
**Key Point:** Resolved at **runtime** using object type.

CODE EXAMPLES SHOWN IN ABOVE QUE

Q.17 What is encapsulation? Write a program demonstrating encapsulation.

ANSWER:

**Encapsulation** is the process of **binding data (variables)** and **methods (functions)** that operate on the data into a single unit, called a **class**.  
It also means **restricting direct access** to some components, usually by using **private** variables and **public** getter/setter methods.



Q.18 What is abstraction in Java? How is it achieved?

ANSWER:

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

It helps in focusing on **what an object does**, instead of **how it does it**.

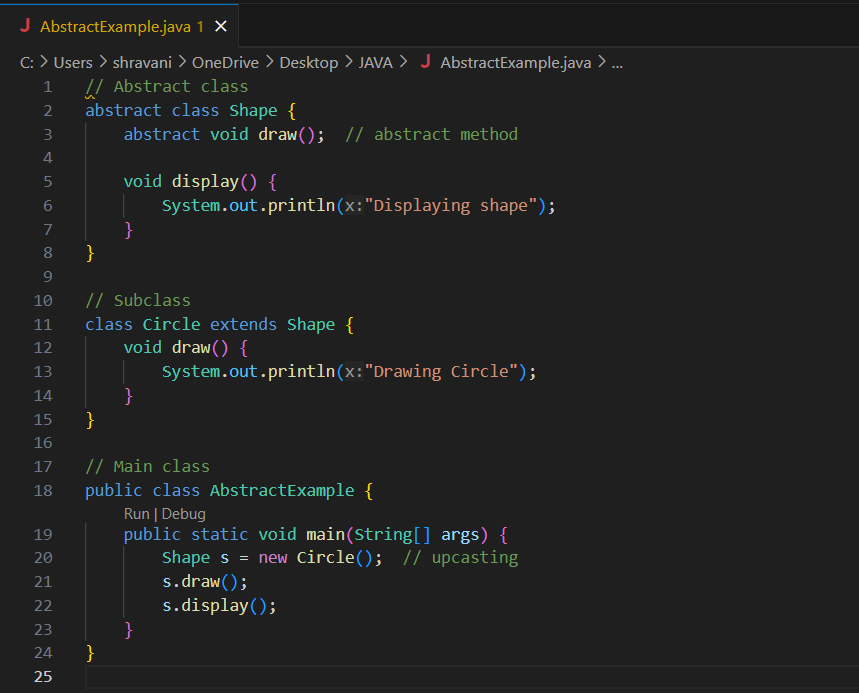
**How is Abstraction Achieved in Java?**

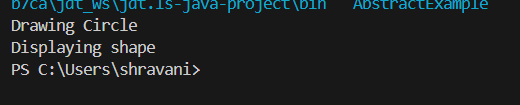
Abstraction in Java is achieved using:

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

**1. Using Abstract Class**

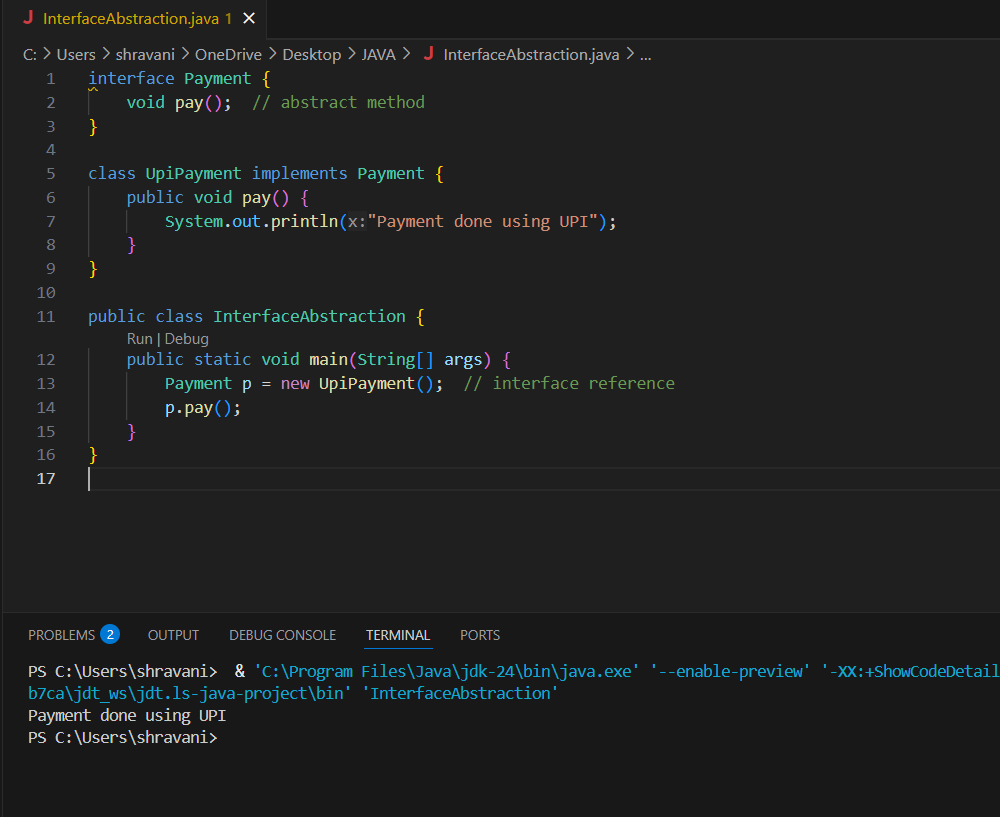
* Contains one or more abstract methods (methods without body).
* Cannot be instantiated directly.
* Subclasses must provide implementations.





**2. Using Interface**

* All methods are **implicitly abstract and public** (Java 8+ can also have default/static methods).
* A class implements the interface and provides method definitions.



Q.19 Explain the difference between abstract class and interface.

ANSWER:

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| **Purpose** | To provide partial abstraction | To provide full abstraction |
| **Keyword Used** | abstract | interface |
| **Method Type** | Can have abstract **and** non-abstract methods | Only abstract methods (Java 7), can have default, static (Java 8+) |
| **Constructor** | Yes, can have constructors | ❌ No constructors |
| **Multiple Inheritance** | ❌ Not supported (single inheritance only) | ✅ Supported (a class can implement multiple interfaces) |
| **Access Modifiers** | Can use private, protected, public | All methods are public and abstract by default (Java 7) |
| **Variables** | Can have instance variables | Only public static final constants |
| **Usage Example** | Use when classes are closely related | Use to define common behavior across classes |
|  |  |  |

Q.20 Create a Java program to demonstrate the use of interface.

ANSWER:

