**NAME: Rushikesh Shinde**

**AF CODE: AF04953330**

**Batch Code: ANP-D1544**

**# Java Basics & OOPs Assignment Questions**

1. **Java Basics**

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

**ANS:**

#Definition

Java is a general-purpose, secure, and robust programming language known for its "Write Once, Run Anywhere" philosophy.

#Key Features of Java

1. Simple

Easy to learn and use.

Removes complex features like pointers, operator overloading, etc.

2. Object-Oriented

Everything is treated as an object.

Supports concepts like inheritance, encapsulation, polymorphism, and abstraction.

3. Platform Independent

Java code is compiled into bytecode which runs on any system with a JVM.

"Write Once, Run Anywhere" (WORA).

4. Secure

Java provides a secure environment using features like bytecode verification, sandboxing, and security APIs.

No direct access to memory (no pointers).

5. Robust

Strong memory management.

Exception handling and type checking reduce runtime errors.

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

**ANS:**

**1. Writing the Code (Source Code)**

You write the program using the Java programming language and save it in a file with the .java extension.

**2. Compilation**

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

* Command: javac HelloWorld.java
* This converts the human-readable Java code into **bytecode**, which is stored in a file with the .class extension.
* Bytecode is platform-independent.

**3. Class Loading**

The **Class Loader** loads the .class file (bytecode) into memory.

* It is part of the Java Runtime Environment (JRE).
* It loads classes as needed during execution.

**4. Bytecode Verification**

The **Bytecode Verifier** checks the bytecode for security and correctness.

* Ensures the code does not violate access rights.
* Prevents dangerous operations like illegal memory access.

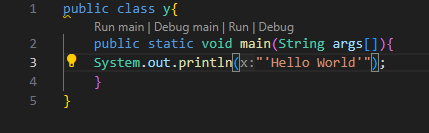
**5. Execution by JVM (Java Virtual Machine)**

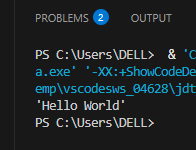
The **JVM** interprets or compiles the bytecode into **machine code** using one of two methods:

* **Interpreter**: Reads and executes bytecode line by line.
* **JIT (Just-In-Time) Compiler**: Converts bytecode into native machine code at runtime for faster execution.
* **6. Runtime Environment**
* The **Java Runtime Environment (JRE)** provides the libraries, Java Virtual Machine, and other components to run applications.

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

**ANS:**





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

**ANS:**

**What are Data Types in Java?**

**Data types** in Java define the type of data a variable can hold. They are used to allocate memory and specify the kind of operations that can be performed on the data.

**Types of Data Types in Java**

Java data types are mainly divided into two categories:

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

**1. Primitive Data Types**

These are the most basic data types in Java. Java has **8**

**(1) primitive data types**:

| **Data Type** | **Size** | **Description** | **Example** |
| --- | --- | --- | --- |
| byte | 1 byte | Stores small integers (-128 to 127) | byte a = 100; |
| short | 2 bytes | Stores medium integers | short b = 10000; |
| int | 4 bytes | Stores large integers (default) | int c = 50000; |
| long | 8 bytes | Stores very large integers | long d = 100000L; |
| float | 4 bytes | Stores decimal numbers (single precision) | float e = 10.5f; |
| double | 8 bytes | Stores decimal numbers (double precision) | double f = 20.99; |
| char | 2 bytes | Stores a single character | char g = 'A'; |
| boolean | 1 bit | Stores true or false values | boolean h = true; |

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

These are data types that refer to objects and include:

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| String | Sequence of characters | String s = "Hello"; |
| Arrays | Collection of similar data types | int[] arr = {1, 2, 3}; |
| Classes | Custom types created using class keyword | class Car { ... } |
| Interfaces | Abstract types for abstraction and polymorphism | interface Vehicle {} |

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

**ANS:**

**1. JVM (Java Virtual Machine)**

* **Definition**: JVM is a part of the JRE that runs Java bytecode. It is responsible for executing Java programs.
* **Function**: Converts **.class** files (bytecode) into machine code for the host operating system.
* **Platform Dependent**: The JVM is platform-specific, meaning different OSes have different JVM implementations.
* **Key Features**: Garbage collection, memory management, security, and execution of bytecode.

**2. JRE (Java Runtime Environment)**

* **Definition**: JRE provides the runtime environment required to run Java applications.
* **Components**: It includes the JVM, core libraries, and other files needed to run Java programs, **but not for developing them**.
* **Use Case**: Ideal for users who want to run Java applications but not develop them.

**3. JDK (Java Development Kit)**

* **Definition**: JDK is a full-featured software development kit required to develop Java applications.
* **Components**: It includes:
  + JRE (and thus also includes the JVM),
  + Compiler (javac),
  + Debugger,
  + Development tools and libraries.
* **Use Case**: Required by developers to write, compile, and debug Java applications.

| **Feature** | **JVM** | **JRE** | **JDK** |
| --- | --- | --- | --- |
| Runs Java programs | ✅ | ✅ | ✅ |
| Contains JVM | ✅ | ✅ | ✅ |
| Contains development tools | ❌ | ❌ | ✅ |
| Used to develop Java programs | ❌ | ❌ | ✅ |
| Used to run Java programs | ✅ | ✅ | ✅ |

In short:

* **JVM**: Runs bytecode.
* **JRE**: Contains JVM + libraries to **run** Java programs.
* **JDK**: Contains JRE + tools to **develop** Java programs.

Top of Form

Bottom of Form

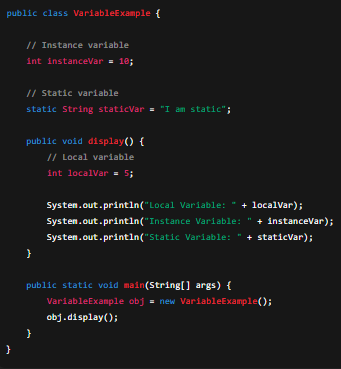
**6. What are variables in Java? Explain with examples.**

**ANS:**

In Java, **variables** are containers used to store data that can be used and manipulated within a program. Each variable has a **type**, a **name**, and a **value**.

**Types of Variables in Java**

1. **Local Variables** – Declared inside a method or block and accessible only within it.
2. **Instance Variables** – Declared inside a class but outside any method; each object has its own copy.
3. **Static Variables** – Declared with the static keyword; shared among all instances of a class.





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

**ANS:**

In Java, **operators** are special symbols used to perform operations on variables and values. They are categorized based on the kind of operation they perform.

**1. Arithmetic Operators**

Used to perform basic mathematical operations.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| + | 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

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| == | 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.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| && | Logical AND | a > 5 && b < 10 |
| ` |  | ` |
| ! | Logical NOT | !(a > 5) |

**4. Assignment Operators**

Used to assign values to variables.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Assign | a = 10 |
| += | Add and assign | a += 5 (same as a = a + 5) |
| -= | Subtract and assign | a -= 3 |
| \*= | Multiply and assign | a \*= 2 |
| /= | Divide and assign | a /= 4 |
| %= | Modulus and assign | a %= 3 |

**5. Unary Operators**

Operate on a single operand.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| + | Unary plus (positive) | +a |
| - | Unary minus (negate) | -a |
| ++ | Increment | a++ or ++a |
| -- | Decrement | a-- or --a |
| ! | Logical complement | !true → false |

**6. Bitwise Operators**

Operate on individual bits.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| & | Bitwise AND | a & b |
| ` | ` | Bitwise OR |
| ^ | Bitwise XOR | a ^ b |
| ~ | Bitwise complement | ~a |
| << | Left shift | a << 2 |
| >> | Right shift | a >> 2 |

**7. Ternary Operator**

Short-hand for if-else.

int result = (a > b) ? a : b;

**8. instanceof Operator**

Checks if an object is an instance of a specific class or subclass.

if (obj instanceof String) {

// do something

}

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

**ANS:**

**1. if Statement**

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

**Syntax:**

if (condition) {

// code to execute if condition is true

}

**Example:**

int age = 20;

if (age >= 18) {

System.out.println("You are an adult.");

}

**2. if-else Statement**

Provides an **alternative block** of code if the condition is false.

**Syntax:**

if (condition) {

// executes if condition is true

} else {

// executes if condition is false

}

**Example:**

int number = 5;

if (number % 2 == 0) {

System.out.println("Even number");

} else {

System.out.println("Odd number");

}

**3. if-else if-else Ladder**

Used when there are **multiple conditions** to check.

**Syntax:**

if (condition1) {

// block1

} else if (condition2) {

// block2

} else {

// default block

}

**Example:**

int marks = 85;

if (marks >= 90) {

System.out.println("Grade A");

} else if (marks >= 75) {

System.out.println("Grade B");

} else {

System.out.println("Grade C");

}

**4. switch Statement**

Used to execute **one block out of many**, based on the value of a variable.

**Syntax:**

switch (expression) {

case value1:

// code block

break;

case value2:

// code block

break;

...

default:

// default block

}

**Example:**

int day = 3;

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

default:

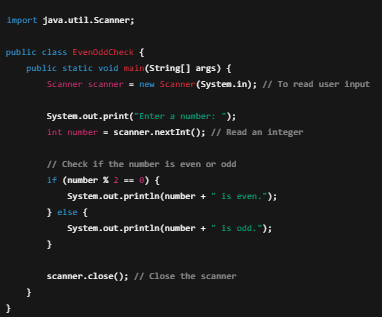
System.out.println("Other day");

}

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

**ANS:**

**Java Program to Check Even or Odd**





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

**ANS:**

Both while and do-while loops are used to repeat a block of code **as long as a condition is true**, but they differ in **when** the condition is checked.

**1. while Loop**

* **Condition is checked first**, before executing the loop body.
* If the condition is false initially, the loop body **may not execute at all**.

**Syntax:**

while (condition) {

// loop body

}

**Example:**

int i = 5;

while (i < 5) {

System.out.println("This won't print");

i++;

}

**2. do-while Loop**

* **Condition is checked after** the loop body is executed.
* The loop body will execute **at least once**, even if the condition is false.

**Syntax:**

do {

// loop body

} while (condition);

**Example:**

java

CopyEdit

int i = 5;

do {

System.out.println("This will print once");

i++;

} while (i < 5);

#Key Differences

| **Feature** | **while Loop** | **do-while Loop** |
| --- | --- | --- |
| Condition check | Before loop body (pre-check) | After loop body (post-check) |
| Minimum execution | May not execute at all | Executes at least once |
| Use case | When the loop may not run | When loop must run 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 {

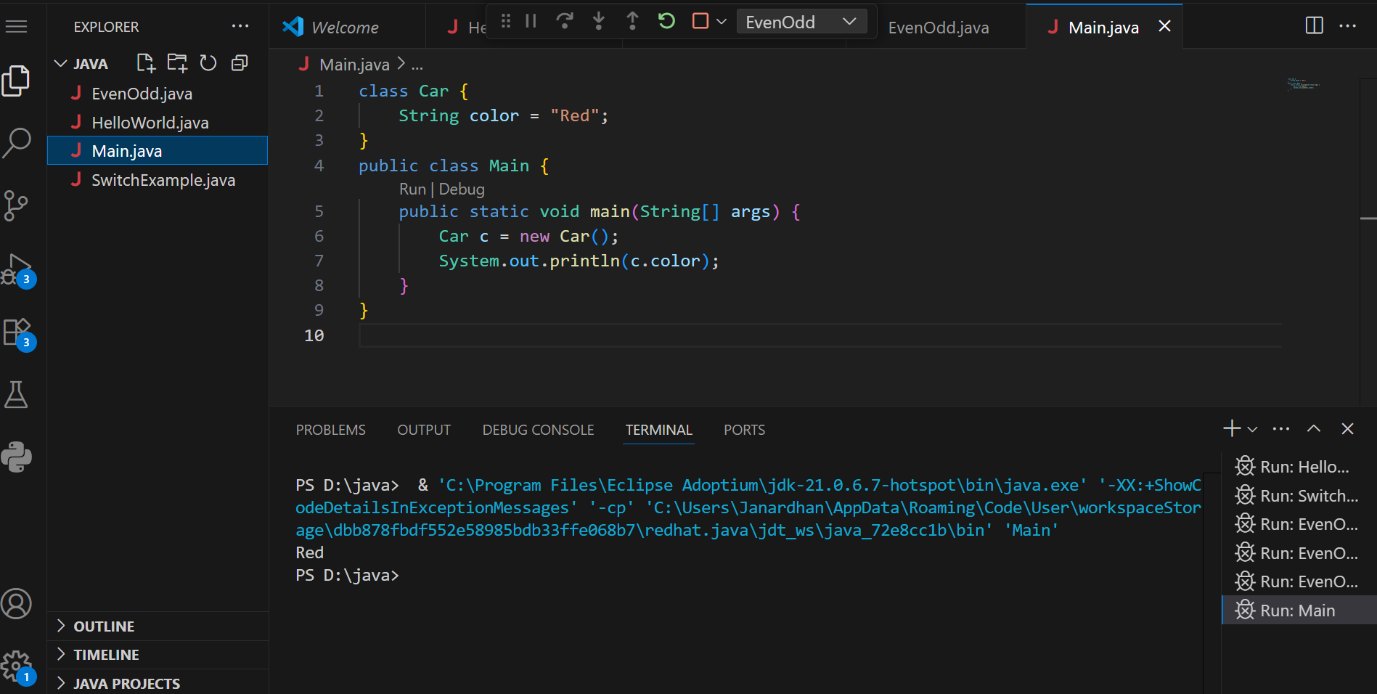
public static void main(String[] args) {

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) {

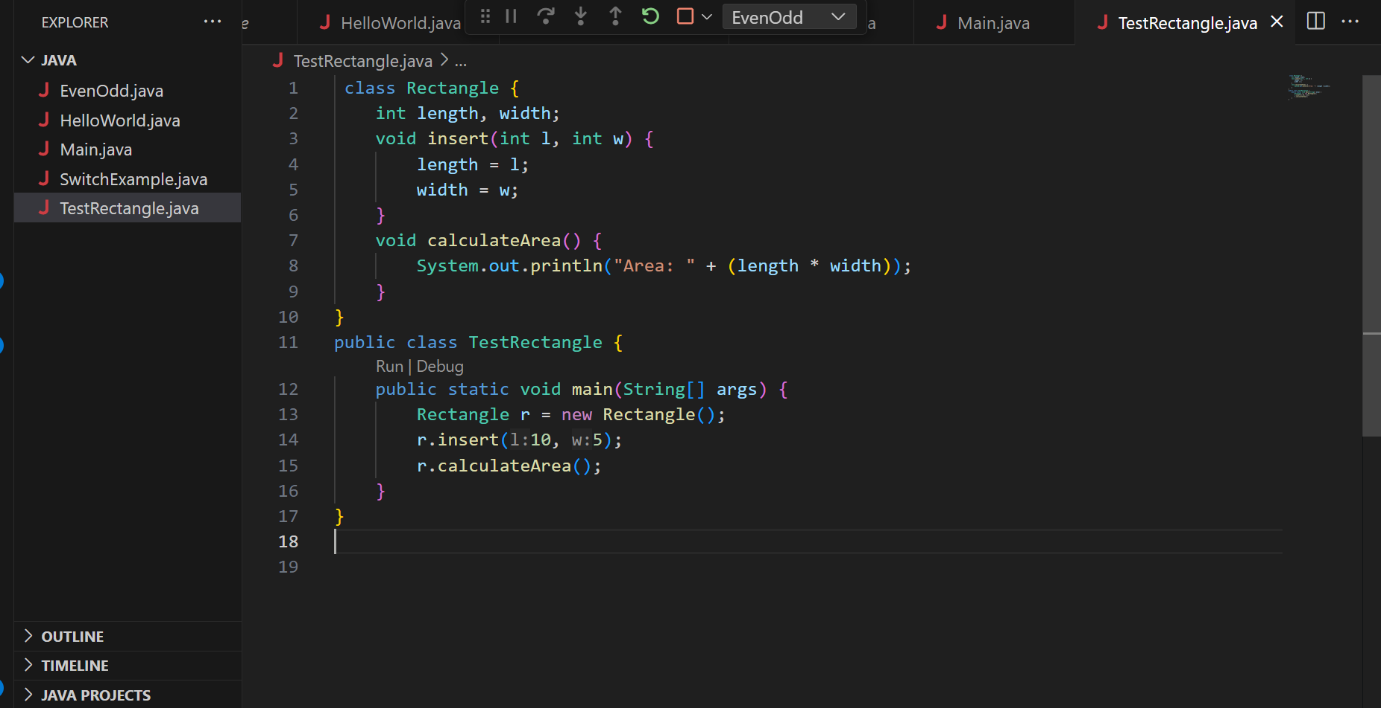
Rectangle r = new Rectangle();

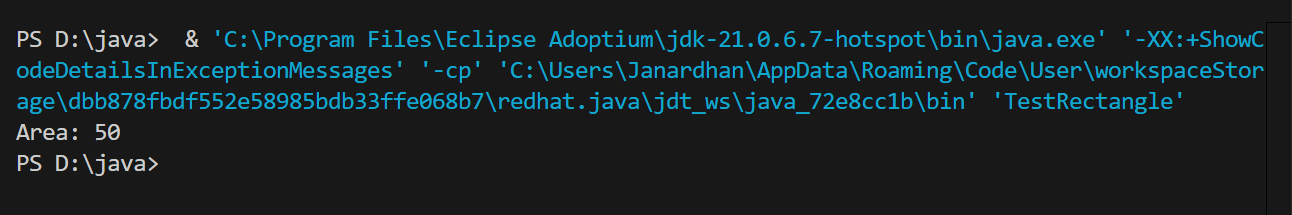
r.insert(10, 5);

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) {

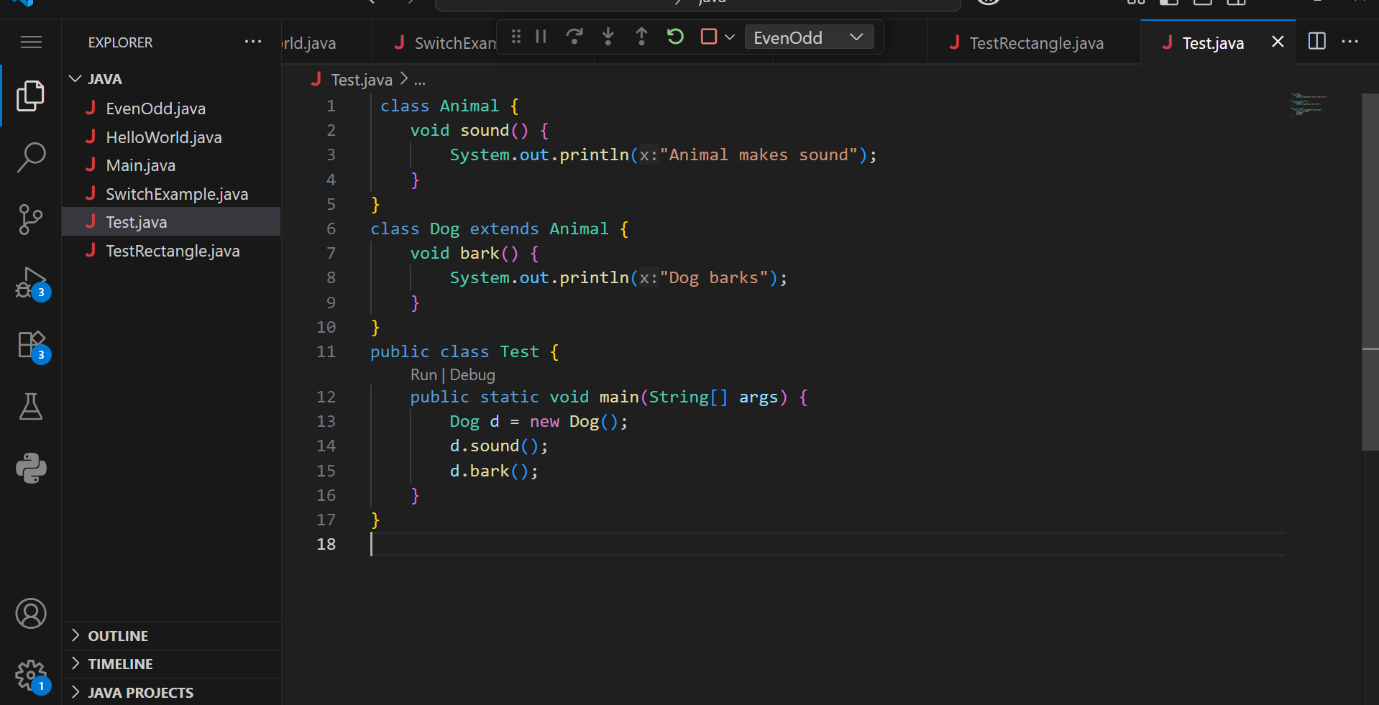
Dog d = new Dog();

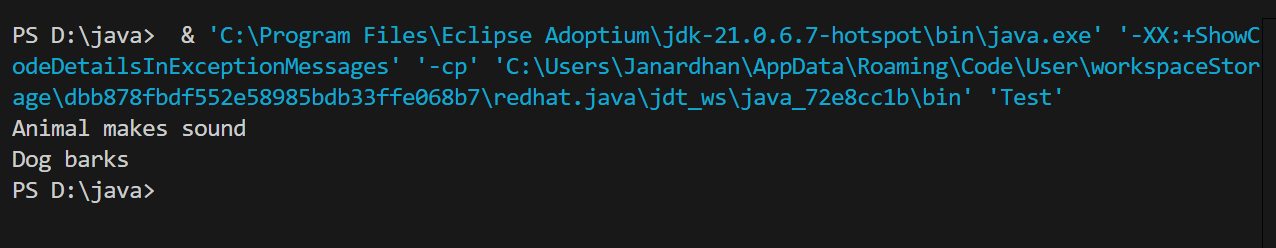
d.sound();

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));

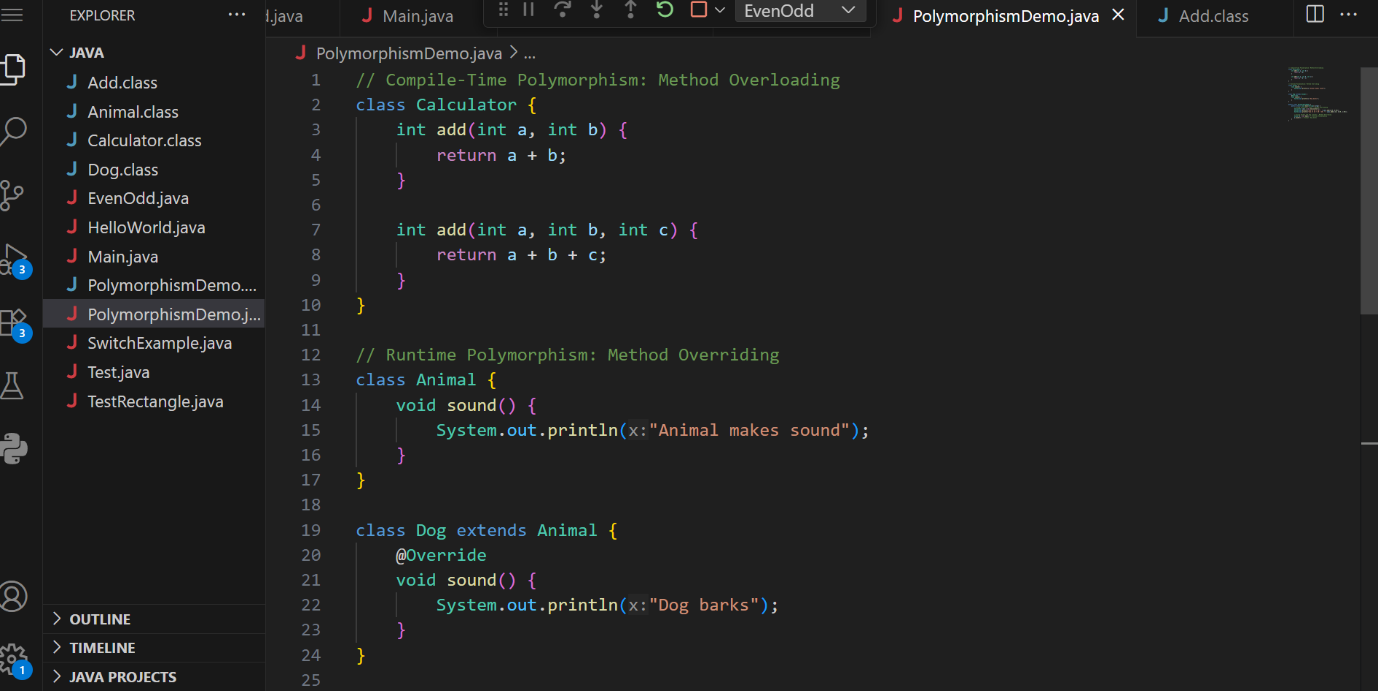
// Using Animal and Dog classes - Method Overriding

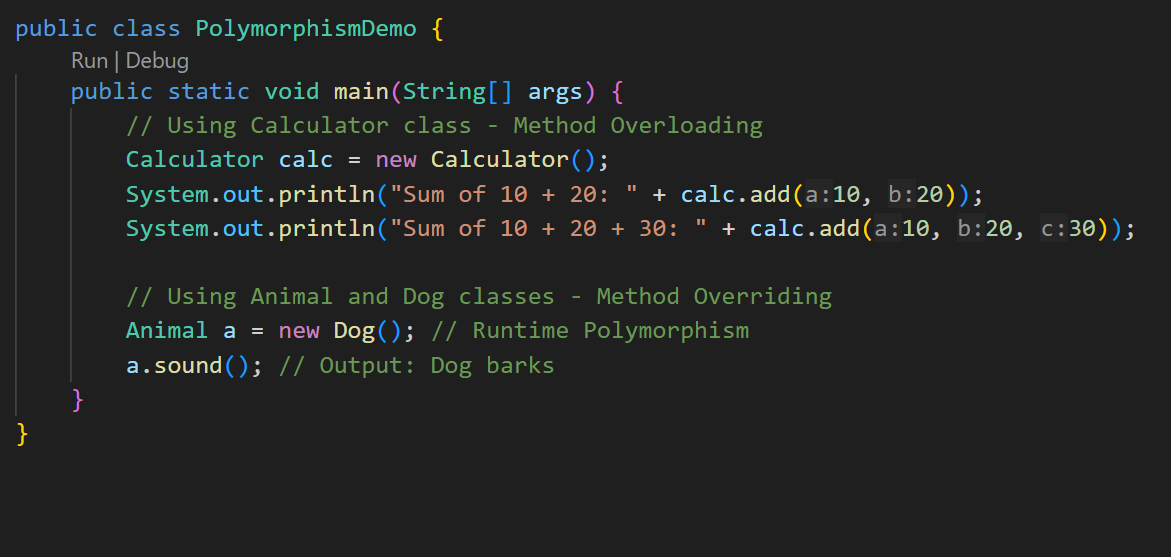
Animal a = new Dog(); // Runtime Polymorphism

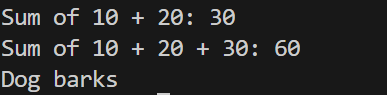
a.sound(); // Output: Dog barks

}

}







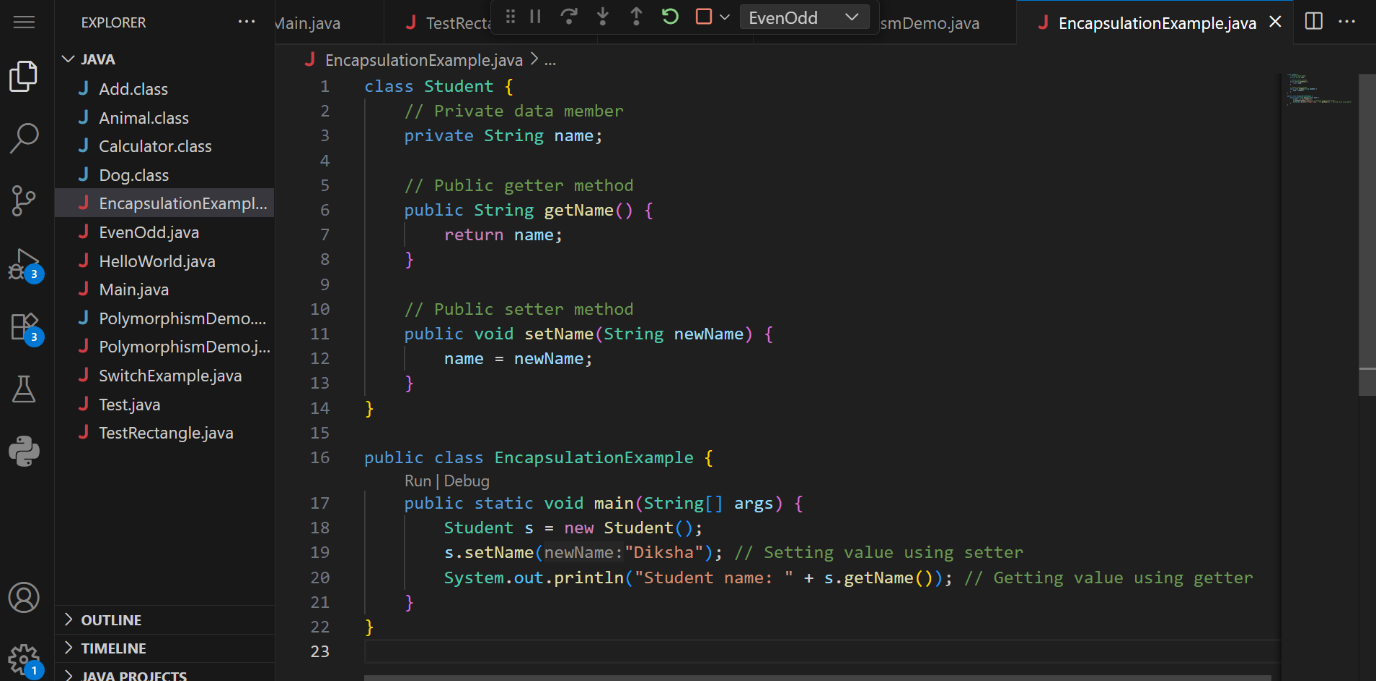
6. What is method overloading and method overriding? Show with examples.

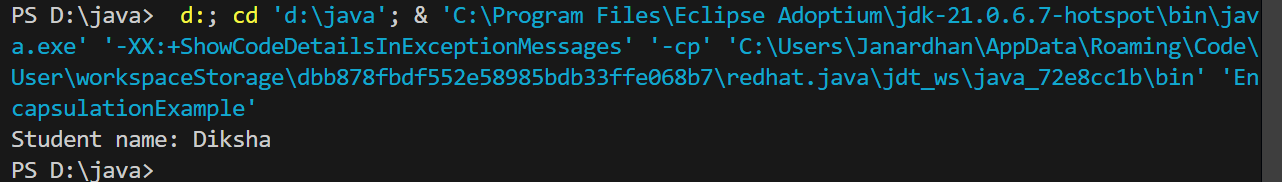
* **Overloading**: Same method name, different parameters.
* **Overriding**: Same method in child class with different behavior.

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.

