

# Java Basics & OOPs Assignment Questions

---

**Name:** Arya Milind Dilliwale

**AF code:** AF04954002

---

## 1. Java Basics

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

Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle). It is platform-independent, secure, and widely used for building web and mobile applications.

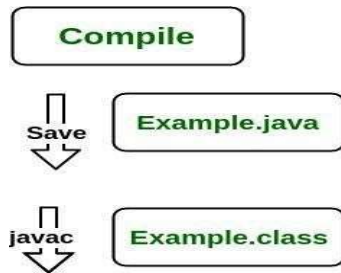
#### KEY FEATURES:

1. **Object-Oriented:** Everything in Java is treated as an object, allowing for modular programs and reusable code.
2. **Platform-Independent:** Java code is compiled into bytecode, which can run on any platform with a JVM.
3. **Simple and Familiar:** Java's syntax is clean and easy to understand, especially for those familiar with C or C++.
4. **Secure:** Java has built-in security features like bytecode verification, sandboxing, and runtime security checks.
5. **Robust:** It handles errors gracefully with strong memory management and exception handling.
6. **Multithread:** Java allows the development of programs that can perform multiple tasks at once.

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

The JDK enables the development and execution of Java programs. Consider the following process:

- **Java Source File (e.g., Example.java):** You write the Java program in a source file.
- **Compilation:** The source file is compiled by the Java Compiler (part of JDK) into bytecode, which is stored in a .class file (e.g., Example.class).
- **Execution:** The bytecode is executed by the JVM (Java Virtual Machine), which interprets the bytecode and runs the Java program.



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

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

```
1 // A Java program to print "Hello World"  
2 public class HelloWorld {  
3     public static void main(String args[])  
4     {  
5         System.out.println("Hello World");  
6     }  
7 }
```

Output:

```
Hello World
```

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

In Java, data types define the type of data a variable can hold. They are essential because Java is a strongly typed language, meaning every variable must be declared with a data type.

Java data types are broadly divided into two categories:

◆ 1. Primitive Data Types (8 types)			
These are the <b>basic built-in types</b> provided by Java.			
Type	Size	Description	Example
<code>byte</code>	1 byte	Stores whole numbers from -128 to 127	<code>byte a = 10;</code>
<code>short</code>	2 bytes	Stores whole numbers from -32,768 to 32,767	<code>short b = 2000;</code>
<code>int</code>	4 bytes	Default type for integers	<code>int c = 100000;</code>
<code>long</code>	8 bytes	Stores large whole numbers	<code>long d = 9999999999L;</code>
<code>float</code>	4 bytes	Stores decimal values (single precision)	<code>float e = 3.14f;</code>
<code>double</code>	8 bytes	Stores decimal values (double precision)	<code>double f = 3.14159;</code>
<code>char</code>	2 bytes	Stores a single character (Unicode)	<code>char g = 'A';</code>
<code>boolean</code>	1 bit	Stores true or false values	<code>boolean h = true;</code>

◆ 2. Non-Primitive (Reference/Object) Data Types		
These refer to <b>objects</b> and <b>classes</b> in Java. They don't store the actual data but a reference to it.		
Type	Description	Example
<code>String</code>	Stores a sequence of characters	<code>String name = "Vinay";</code>
Arrays	Collection of fixed number of same type elements	<code>int[] arr = {1, 2, 3};</code>
Classes	Custom data types created using <code>class</code> keyword	<code>MyClass obj = new MyClass();</code>
Interfaces	Reference to an object that implements the interface	<code>Runnable r = new MyThread();</code>

```

1 public class DataTypesExample {
    Run | Debug
2     public static void main(String[] args) {
3
4         // boolean: 1 bit (but JVM uses 1 byte for practical reasons)
5         boolean isJavaFun = true;
6         System.out.println("Boolean value: " + isJavaFun);
7
8         // char: 2 bytes (16 bits), Unicode character
9         char letter = 'A';
10        System.out.println("Char value: " + letter);
11
12        // byte: 1 byte (8 bits), range: -128 to 127
13        byte smallNumber = 100;
14        System.out.println("Byte value: " + smallNumber);
15
16        // short: 2 bytes (16 bits), range: -32,768 to 32,767
17        short shortNumber = 32000;
18        System.out.println("Short value: " + shortNumber);
19
20        // int: 4 bytes (32 bits), range: -2^31 to 2^31-1
21        int number = 100000;
22        System.out.println("Int value: " + number);
23
24        // long: 8 bytes (64 bits), range: -2^63 to 2^63-1
25        long bigNumber = 100000000000L;
26        System.out.println("Long value: " + bigNumber);
27
28        // float: 4 bytes (32 bits), single-precision 32-bit IEEE 754
29        float pi = 3.14f;
30        System.out.println("Float value: " + pi);
31
32        // double: 8 bytes (64 bits), double-precision 64-bit IEEE 754
33        double largeDecimal = 12345.6789;
34        System.out.println("Double value: " + largeDecimal);
35    }
36 }

```

```

Boolean value: true
Char value: A
Byte value: 100
Short value: 32000
Int value: 100000
Long value: 100000000000
Float value: 3.14
Double value: 12345.6789

```

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

JDK: Java Development Kit is a software development environment used for developing Java applications and applets.

JRE: JRE stands for Java Runtime Environment, and it provides an environment to run only the Java program onto the system.

JVM: JVM stands for Java Virtual Machine and is responsible for executing the Java program.

## JDK vs JRE vs JVM

Aspect	JDK	JRE	JVM
Purpose	Used to develop Java applications	Used to run Java applications	Executes Java bytecode
Platform Dependency	Platform-dependent (OS specific)	Platform-dependent (OS specific)	JVM is OS-specific, but bytecode is platform-independent
Includes	JRE + Development tools (javac, debugger, etc.)	JVM + Libraries (e.g., rt.jar)	ClassLoader, JIT Compiler, Garbage Collector
Use Case	Writing and compiling Java code	Running a Java application on a system	Convert bytecode into native machine code

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

In Java, variables are containers that store data in memory, it defines how data is stored, accessed, and manipulated.

```
3 class variables {  
    Run | Debug  
4     public static void main(String[] args) {  
5         // Declaring and initializing variables  
6  
7         // Integer variable  
8         int age = 25;  
9  
10        // String variable  
11        String name = "Vinay-The Boss!";  
12  
13        // Double variable  
14        double salary = 50000.50;  
15  
16        // Displaying the values of variables  
17        System.out.println("Age: " + age);  
18        System.out.println("Name: " + name);  
19        System.out.println("Salary: " + salary);  
20    }  
21 }
```

Output:

```
Age: 25  
Name: Vinay-The Boss!  
Salary: 50000.5
```

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

ARITHMETIC OPERATORS:

```
Lecture 1 > ArithmeticExample.java > ArithmeticExample > main(String[])
1 public class ArithmeticExample {
    Run | Debug
2     public static void main(String[] args) {
3         int a = 15;
4         int b = 4;
5
6         System.out.println(x:"Arithmetic Operators Example:");
7         System.out.println("a + b = " + (a + b)); // Addition
8         System.out.println("a - b = " + (a - b)); // Subtraction
9         System.out.println("a * b = " + (a * b)); // Multiplication
10        System.out.println("a / b = " + (a / b)); // Division (integer division)
11        System.out.println("a % b = " + (a % b)); // Modulus (remainder)
12    }
13 }
14
```

RELATIONAL OPERATORS:

```
Lecture 1 > RelationalExample.java > RelationalExample > main(String[])
1 public class RelationalExample {
    Run | Debug
2     public static void main(String[] args) {
3         int a = 10;
4         int b = 5;
5
6         System.out.println(x:"Relational (Comparison) Operators Example:");
7         System.out.println("a == b: " + (a == b)); // Checks if a is equal to b
8         System.out.println("a != b: " + (a != b)); // Checks if a is not equal to b
9         System.out.println("a > b: " + (a > b)); // Checks if a is greater than b
10        System.out.println("a < b: " + (a < b)); // Checks if a is less than b
11        System.out.println("a >= b: " + (a >= b)); // Checks if a is greater than or equal to b
12        System.out.println("a <= b: " + (a <= b)); // Checks if a is less than or equal to b
13    }
14 }
15
```

LOGICAL OPERATORS

```
Employee.java calculator.java EncapsulationDemo.java ThreadExample.java HelloWorld.java DataTypesExample.java variables.java
Lecture 1 > LogicalExample.java > ...
1 public class LogicalExample {
    Run | Debug
2     public static void main(String[] args) {
3         boolean x = true;
4         boolean y = false;
5
6         System.out.println(x:"Logical Operators Example:");
7         System.out.println("x && y: " + (x && y)); // Logical AND
8         System.out.println("x || y: " + (x || y)); // Logical OR
9         System.out.println("!x: " + (!x)); // Logical NOT
10        System.out.println("!y: " + (!y)); // Logical NOT
11    }
12 }
13
```

ASSIGNMENT OPERATORS:

```
Calculator.java EncapsulationDemo.java ThreadExample.java HelloWorld.java DataTypesExample.java variables.java
AssignmentExample.java > _
1 public class AssignmentExample {
    Run | Debug
2     public static void main(String[] args) {
3         int a = 10;
4
5         System.out.println(x:"\nAssignment Operators Example:");
6
7         a += 5; // a = a + 5
8         System.out.println("a += 5: " + a); // 15
9
10        a -= 3; // a = a - 3
11        System.out.println("a -= 3: " + a); // 12
12
13        a *= 2; // a = a * 2
14        System.out.println("a *= 2: " + a); // 24
15
16        a /= 4; // a = a / 4
17        System.out.println("a /= 4: " + a); // 6
18
19        a %= 4; // a = a % 4
20        System.out.println("a %= 4: " + a); // 2
21    }
22 }
23
PROBLEMS 47 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Vinay\OneDrive\Documents\ANP-D1544> cmd /C ""C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview -XX:+ShowCodeDetailsIn
\jdt_ws\ANP-D1544_68b56e37\bin AssignmentExample "

Assignment Operators Example:
a += 5: 15
a -= 3: 12
a *= 2: 24
a /= 4: 6
a %= 4: 2

C:\Users\Vinay\OneDrive\Documents\ANP-D1544>
```

UNARY OPERATORS:



```

1 public class UnaryExample {
    Run | Debug
2     public static void main(String[] args) {
3         int a = 5;
4
5         System.out.println(x:"Unary Operators Example:");
6         System.out.println("Initial a: " + a); // 5
7
8         System.out.println("++a: " + (++a)); // 6 (pre-increment)
9         System.out.println("a++: " + (a++)); // 6 (post-increment, then a becomes 7)
10        System.out.println("After a++: " + a); // 7
11
12        System.out.println("--a: " + (--a)); // 6 (pre-decrement)
13        System.out.println("a--: " + (a--)); // 6 (post-decrement, then a becomes 5)
14        System.out.println("After a--: " + a); // 5
15
16        boolean flag = true;
17        System.out.println("!flag: " + (!flag)); // false (logical NOT)
18    }
19 }

```

Output:

```

Initial a: 5
++a: 6
a++: 6
After a++: 7
--a: 6
a--: 6
After a--: 5
!flag: false

```

BITWISE OPERATOR:

```

1 public class BitwiseExample{
    Run | Debug
2     public static void main(String[] args) {
3         int a = 5; // 0101 in binary
4         int b = 3; // 0011 in binary
5
6         System.out.println(x:"Bitwise Operators Example:");
7         System.out.println("a & b: " + (a & b)); // 1 (0001)
8         System.out.println("a | b: " + (a | b)); // 7 (0111)
9         System.out.println("a ^ b: " + (a ^ b)); // 6 (0110)
10        System.out.println("~a: " + (~a)); // -6 (2's complement)
11        System.out.println("a << 1: " + (a << 1)); // 10 (1010)
12        System.out.println("a >> 1: " + (a >> 1)); // 2 (0010)
13    }
14 }

```

Output:

```

Bitwise Operators Example:
a & b: 1
a | b: 7
a ^ b: 6
~a: -6
a << 1: 10
a >> 1: 2

```



## **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. **3.**

### **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.**

```
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)

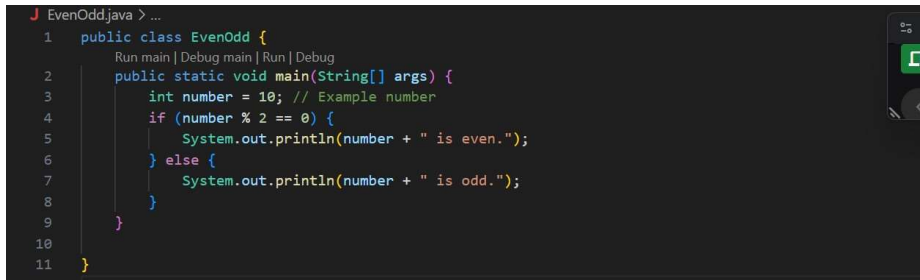
            System.out.println("Even");

        else

            System.out.println("Odd");

    }

}
```

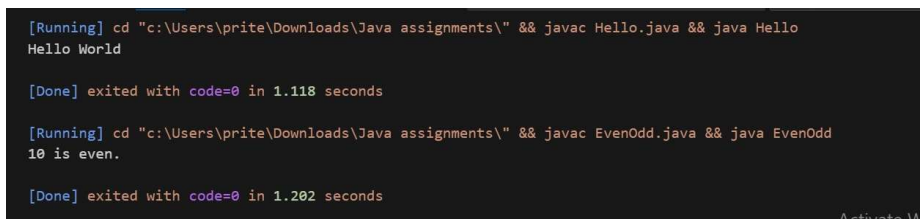


```

1  public class EvenOdd {
2      Run main | Debug main | Run | Debug
3      public static void main(String[] args) {
4          int number = 10; // Example number
5          if (number % 2 == 0) {
6              System.out.println(number + " is even.");
7          } else {
8              System.out.println(number + " is odd.");
9          }
10     }
11 }

```

Output:



```

[Running] cd "c:\Users\prite\Downloads\Java assignments\" && javac Hello.java && java Hello
Hello World

[Done] exited with code=0 in 1.118 seconds

[Running] cd "c:\Users\prite\Downloads\Java assignments\" && javac EvenOdd.java && java EvenOdd
10 is even.

[Done] exited with code=0 in 1.202 seconds

```

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

while: checks condition before executing, May never execute    do-while:  
executes at least once, Executes at least once

## 2. Object-Oriented Programming (OOPs)

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

Object-Oriented Programming (OOP) in Java is based on **four main principles**. These principles help design modular, reusable, and maintainable code. The following are the main principles of OOPs:

#### a. Encapsulation.

Wrapping data (variables) and code (methods) together into a single unit called a class and restricting direct access to some of the object's components.

Key Points:

- Achieved using private variables.
- Accessed via getter and setter methods.
- Helps in data hiding and security

#### b. Inheritance

One class (child/subclass) inherits the properties and behaviours (methods) of another class (parent/superclass).

Key Points:

- Promotes code reusability.
- Use of extends keyword.
- Base class methods can be overridden.
- 

### **c. Polymorphism**

The ability to take many forms. The same method or object behaves differently based on context.

Types:

- Compile-time Polymorphism (Method Overloading)
- Run-time Polymorphism (Method Overriding)

### **d. Abstraction**

Hiding internal implementation details and showing only essential features to the user.

Key Points:

- Achieved using abstract classes and interfaces.
- Helps in focusing on what an object does rather than how.

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

A class is a blueprint or template for creating objects.

It defines properties (fields/variables) and behaviors (methods) that the objects created from the class can have.

Example:

```

1  class Car {
2      String color;
3      int speed;
4
5      void drive() {
6          System.out.println(x:"The car is driving");
7      }
8
9      void brake() {
10         System.out.println(x:"The car is braking");
11     }
12 }
13 public class class_assign {
14     public static void main(String[] args) {
15         // First object
16         Car myCar = new Car();
17         myCar.color = "Red";
18         myCar.speed = 100;
19         myCar.drive();
20         System.out.println("Color: " + myCar.color);
21         System.out.println("Speed: " + myCar.speed + " km/h");
22         myCar.brake();
23
24         System.out.println(x:"-----");
25
26         // Second object
27         Car yourCar = new Car();
28         yourCar.color = "Blue";
29         yourCar.speed = 80;
30         yourCar.drive();
31         System.out.println("Color: " + yourCar.color);
32         System.out.println("Speed: " + yourCar.speed + " km/h");
33         yourCar.brake();
34     }
35 }

```

Output :

```

The car is driving
Color: Red
Speed: 100 km/h
The car is braking

```

```

-----
The car is driving
Color: Blue
Speed: 80 km/h
The car is driving
Color: Red
Speed: 100 km/h
The car is braking

```

```

The car is driving
Color: Blue
Speed: 80 km/h
Speed: 100 km/h
The car is braking

```

```

-----
The car is driving
Color: Blue
Speed: 80 km/h
The car is driving
Color: Blue
Speed: 80 km/h
The car is braking
The car is braking

```

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

```

1  import java.util.Scanner;
2
3  public class area_of_rec{
4
5      // Class to represent a rectangle
6      static class Rectangle {
7          double length;
8          double width;
9
10         // Method to calculate area
11         double calculateArea() {
12             return length * width;
13         }
14     }
15
16     Run | Debug
17     public static void main(String[] args) {
18         Scanner sc = new Scanner(System.in);
19
20         // Create object of Rectangle
21         Rectangle rect = new Rectangle();
22
23         // Take user input
24         System.out.print(s:"Enter the length of the rectangle: ");
25         rect.length = sc.nextDouble();
26
27         System.out.print(s:"Enter the width of the rectangle: ");
28         rect.width = sc.nextDouble();
29
30         // Calculate and display area
31         double area = rect.calculateArea();
32         System.out.println("Area of the rectangle = " + area);
33     }
34 }

```

#### OUTPUT:

```

Enter the length of the rectangle: 2.5
Enter the width of the rectangle: 4.1
Area of the rectangle = 10.25
PS C:\Users\icon\.vscode>

```

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

Inheritance in Java is an object-oriented programming concept where one class (child/subclass) can inherit properties and behaviors (fields and methods) from another class (parent/superclass).

It promotes code reusability, extensibility, and supports polymorphism.

#### Real-Life Example:

##### Scenario:

Think of a "Vehicle" as a general category.

There are many specific types of vehicles like Car, Bike, Truck, etc.

- All vehicles have common properties like speed, fuel, and methods like start(), stop().
- A Car is a Vehicle, but it also has unique features like airConditioner().

Thus, Car inherits from Vehicle.

```

1  class Vehicle {
2      int speed;
3      void start() {
4          System.out.println(x:"Vehicle has started.");
5      }
6      void stop() {
7          System.out.println(x:"Vehicle has stopped.");
8      }
9  }
10 class Car extends Vehicle {
11     String brand;
12
13     void airConditioner() {
14         System.out.println(x:"Air Conditioner is ON.");
15     }
16 }
17 public class inher_asnm {
18     Run | Debug
19     public static void main(String[] args) {
20         Car myCar = new Car();
21         myCar.speed = 80;
22         myCar.brand = "Toyota";
23
24         myCar.start();
25         System.out.println("Brand: " + myCar.brand);
26         System.out.println("Speed: " + myCar.speed + " km/h");
27         myCar.airConditioner();
28         myCar.stop();
29     }
30 }

```

Output:

```

Vehicle has started.
Brand: Toyota
Speed: 80 km/h
Air Conditioner is ON.
Vehicle has stopped.

```

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

**Polymorphism** is one of the four core principles of Object-Oriented Programming (OOP) in Java.

It means “**many forms**”, the ability of a method or object to behave differently based on the context.

Types of Polymorphism in Java:

Java supports two types of polymorphism:

1. Compile-time Polymorphism (Static Binding / Method Overloading)
2. Run-time Polymorphism (Dynamic Binding / Method Overriding)

### Compile-time Polymorphism (Method Overloading)

When multiple methods in the same class have the same name but different parameters, it's called method overloading.

The decision of which method to call is made at compile time.

```
1  class Calculator {
2      int add(int a, int b) {
3          return a + b;
4      }
5
6      double add(double a, double b) {
7          return a + b;
8      }
9
10     int add(int a, int b, int c) {
11         return a + b + c;
12     }
13 }
14
15 public class moloading {
16     Run|Debug
17     public static void main(String[] args) {
18         Calculator calc = new Calculator();
19
20         System.out.println("Add (int, int): " + calc.add(a:5, b:3));
21         System.out.println("Add (double, double): " + calc.add(a:2.5, b:3.5));
22         System.out.println("Add (int, int, int): " + calc.add(a:1, b:2, c:3));
23     }
24 }
```

Output:

```
C:\Users\icon\.vscode\JAVA>java moloading
Add (int, int): 8
Add (double, double): 6.0
Add (int, int, int): 6
```

## Run-time Polymorphism (Method Overriding)

When a subclass provides its own implementation of a method that is already defined in the parent class, it's called method overriding.

The method to be called is determined at runtime, based on the object's actual type.



```

1  class Animal {
2      void sound() {
3          System.out.println(x:"Animal makes a sound");
4      }
5  }
6  class Dog extends Animal {
7      @Override
8      void sound() {
9          System.out.println(x:"Dog barks");
10     }
11 }
12 class Cat extends Animal {
13     @Override
14     void sound() {
15         System.out.println(x:"Cat meows");
16     }
17 }
18 public class moriding{
19     Run | Debug
20     public static void main(String[] args) {
21         Animal a1 = new Dog();
22         Animal a2 = new Cat();
23         a1.sound();
24         a2.sound();
25     }
26 }

```

Output:

```

Animal makes a sound
Dogs barks.
Animal makes a sound
Cat meows.

```

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

Compile-time Polymorphism (Method Overloading)

```

1  class Calculator {
2      int add(int a, int b) {
3          return a + b;
4      }
5
6      double add(double a, double b) {
7          return a + b;
8      }
9
10     int add(int a, int b, int c) {
11         return a + b + c;
12     }
13 }
14
15 public class moloading {
16     Run | Debug
17     public static void main(String[] args) {
18         Calculator calc = new Calculator();
19
20         System.out.println("Add (int, int): " + calc.add(a:5, b:3));
21         System.out.println("Add (double, double): " + calc.add(a:2.5, b:3.5));
22         System.out.println("Add (int, int, int): " + calc.add(a:1, b:2, c:3));
23     }
24 }

```

Output:

```

C:\Users\icon\.vscode\JAVA>java moloading
Add (int, int): 8
Add (double, double): 6.0
Add (int, int, int): 6

```

## Run-time Polymorphism (Method Overriding)

```
1  class Animal {
2      void sound() {
3          System.out.println(x:"Animal makes a sound");
4      }
5  }
6  class Dog extends Animal {
7      @Override
8      void sound() {
9          System.out.println(x:"Dog barks");
10     }
11 }
12 class Cat extends Animal {
13     @Override
14     void sound() {
15         System.out.println(x:"Cat meows");
16     }
17 }
18 public class moriding{
19     Run|Debug
20     public static void main(String[] args) {
21         Animal a1 = new Dog();
22         Animal a2 = new Cat();
23         a1.sound();
24         a2.sound();
25     }
26 }
```

Output:

```
Animal makes a sound
Dogs barks.
Animal makes a sound
Cat meows.
```

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

### Encapsulation:

Encapsulation is one of the fundamental principles of Object-Oriented Programming (OOP) in Java.

It refers to the binding of data (variables) and code (methods) together into a single unit, typically a class, and restricting direct access to some of the object's internal components.

### **Key Features of Encapsulation:**

- Fields (variables) are made private.
- Access is provided through public getter and setter methods.
- Helps in data hiding, security, and controlled access.

```

2  class Student {
3
4      private String name;
5      private int age;
6
7      public void setName(String name) {
8          this.name = name;
9      }
10
11     public String getName() {
12         return name;
13     }
14
15     public void setAge(int age) {
16         if (age > 0) {
17             this.age = age;
18         } else {
19             System.out.println(x:"Invalid age!");
20         }
21     }
22
23     public int getAge() {
24         return age;
25     }
26 }
27
28 public class encaps {
29     public static void main(String[] args) {
30
31         Student s1 = new Student();
32
33         s1.setName(name:"Jay");
34         s1.setAge(age:19);
35
36         System.out.println("Name: " + s1.getName());
37         System.out.println("Age: " + s1.getAge());
38     }
39 }

```

Output:

```

Name: Jay
Age: 19

```

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

Abstraction is an Object-Oriented Programming (OOP) concept in Java that means hiding the internal implementation details and showing only the essential features of an object to the user.

It helps in reducing complexity and increases code readability and maintainability.

Real-life example:

When you drive a car, you just use the steering, accelerator, and brakes. You don't need to know how the engine works internally.

Similarly, in Java, abstraction hides the complex code and shows only what's necessary

## 9. Explain the difference between abstract class and interface.

Feature	Abstract Class	Interface
Keyword	<b>abstract</b>	<b>interface</b>
Methods	Can have abstract and non-abstract methods	Can have only abstract methods ; default, static, and private methods
Variables	Can have instance variables (with any access modifier)	Only public static final (constants)
Constructor	Can have constructors	Cannot have constructors
Access Modifiers	Can use private, protected, public	Methods are public by default
Inheritance Type	Supports single inheritance	Supports multiple inheritance
When to Use	When you want to share code and common logic	When you want to define a contract (what to do)
Instantiation	Cannot be instantiated directly	Cannot be instantiated directly

#### Abstract Class Example:

```

1  abstract class Animal {
2      abstract void sound();
3
4      void sleep() {
5          System.out.println(x:"Sleeping...");
6      }
7  }
8
9  class Dog extends Animal {
10     void sound() {
11         System.out.println(x:"Dog barks");
12     }
13 }

```

#### Interface Example:

```

1  interface Flyable {
2      void fly();
3  }
4
5  class Bird implements Flyable {
6      public void fly() {
7          System.out.println(x:"Bird is flying");
8      }
9  }

```

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

```
1 interface Shape {
2     void draw();
3 }
4 class Circle implements Shape {
5     public void draw() {
6         System.out.println(x:"Drawing a Circle");
7     }
8 }
9 class Rectangle implements Shape {
10     public void draw() {
11         System.out.println(x:"Drawing a Rectangle");
12     }
13 }
14 public class interf_eg {
15     Run | Debug
16     public static void main(String[] args) {
17         Shape circle = new Circle();
18         Shape rectangle = new Rectangle();
19         circle.draw();
20         rectangle.draw();
21     }
22 }
```

### Output:

```
Drawing a Circle
Drawing a Rectangle
```