Assignment_Java

Vaishnavi Shashikant Suryawanshi.

Java Basics

1.What is Java? Explain its features.

→ Java is a **high-level, object-oriented, and platform-independent** programming language developed by Sun Microsystems in 1995. It allows you to write code once and run it anywhere using the **Java Virtual Machine (JVM)**.

Features of Java:

1. **Simple** – Easy to learn and use.
2. **Object-Oriented** – Everything is based on objects and classes.
3. **Platform Independent** – Write once, run anywhere (via JVM).
4. **Secure** – Built-in security features.
5. **Robust** – Strong memory management and error handling.
6. **Multithreaded** – Can perform many tasks at once.
7. **Portable** – Code runs on any device with JVM.
8. **High Performance** – Uses Just-In-Time compiler.
9. **Distributed** – Supports network-based applications.
10. **Dynamic** – Loads classes at runtime as needed.

1. Explain the Java program execution process.

→ **Program Execution Process (Using Notepad, CMD, and JDK)**

Step 1: Install JDK

- Download and install **JDK (Java Development Kit)** from Oracle's website.
- After installation, **set the environment variable** (Path) so that you can run Java from CMD.

Step 2: Write the Java Program using Notepad

1. Open **Notepad**.

2. Write a simple Java program:
3. Save the file.

Step 3: Compile the Program using CMD

1. Open **Command Prompt (CMD)**.
2. Navigate to the folder where your `.java` file is saved using `cd` command.
3. Compile the Java file:

Step 4: Run the Program using CMD

Run the compiled code with the Java interpreter:

Java Program Execution Using VS Code

Step 1: Install JDK

- Download and install the **Java Development Kit (JDK)** from:
  Set the **environment variable (Path)** if not automatically set.

Step 2: Install VS Code & Java Extensions

1. Download and install **Visual Studio Code**.
2. Open VS Code and install these extensions:

   Java Extension Pack

Step 3: Create and Write Java Program

1. Open VS Code.
2. Create a new folder for your Java project and open it in VS Code.
3. Create a new file named:

Step 4: Run the Java Program

Open **Terminal** in VS Code (`Ctrl + ~`)

Compile the program

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

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Word!");
    }
}
[Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\Assignment\" && javac
Hello.java && java Hello
Hello Word!
```

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

In Java, data types define the type of data a variable can store. Java is a strongly-typed language, which means you must declare the type of variable before using it. Data types are mainly **divided into** two categories: primitive data types and non-primitive (reference) data types.

1. Primitive Data Types

- **byte** – used to store small whole numbers
- **short** – used for slightly larger whole numbers
- **int** – commonly used for integers
- **long** – used to store very large whole numbers
- **float** – used for decimal numbers with less precision
- **double** – used for decimal numbers with more precision
- **char** – used to store a single character like 'A'
- **boolean** – used to store true or false values

2. Non-Primitive (Reference) Data Types

- **String** – used to store a sequence of characters (like words or sentences)
- **Arrays** – used to store multiple values of the same type
- **Classes** – used to create objects (user-defined types)
- **Interfaces** – used for abstraction and multiple inheritance
- **Objects** – instances of classes used to store and manage data

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

JVM (Java Virtual Machine)

JVM stands for Java Virtual Machine**.**
It is responsible for running Java programs.
When you compile a Java file, it is converted into bytecode**.**
The JVM reads this bytecode and executes it on your machine.

JRE (Java Runtime Environment)

JRE stands for Java Runtime Environment.
It includes the **JVM** and the libraries needed to run Java programs.

JDK (Java Development Kit)

JDK stands for Java Development Kit**.**
It includes JRE + JVM + development tools like the Java compiler (`javac`), debugger, etc.

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

→ A variable in Java is a container that holds a value which can change during program execution.
It has a name, a data type, and a value.

Before using a variable, we must declare it using a data type.

Syntax:

datatype variableName = value;

Integer Variable: int age = 20;

i`nt` is the data type

`age` is the variable name

`20` is the value assigned to the variable

Boolean Variable: boolean isStudent = true;

String Variable: String name = "Vaishnavi";

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

Operators in Java are **symbols** used to perform operations on variables and values, such as arithmetic, comparison, or logical checks.

Types of Operators in Java:

1. **Arithmetic Operators**

Used for mathematical operations:

- `+` (Addition)
- `-` (Subtraction)
- `*` (Multiplication)
- `/` (Division)
- `%` (Modulus / Remainder)

2. **Relational (Comparison) Operators**

Used to compare two values:

- `==` (Equal to)
- `!=` (Not equal to)
- `>` (Greater than)
- `<` (Less than)
- `>=` (Greater than or equal to)
- `<=` (Less than or equal to)

3. **Logical Operators**

Used to combine multiple conditions:

- `&&` (Logical AND)
- `||` (Logical OR)
- `!` (Logical NOT)

4. **Unary Operators**

Work with only one operand:

- `+`, `-`
- `++` (Increment)
- `--` (Decrement)
- `!` (Logical NOT)

6. **Bitwise Operators**

- `&, |, ^, ~, <<, >>`

7. **Ternary Operator**

```
condition ? value_if_true : value_if_false
```

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

Control statements are used to control the flow of execution in a Java program based on conditions. They help the program make **decisions**.

1. if Statement

The `if` statement checks a condition. If the condition is true, the block of code runs.

Syntax:

```
if (condition) {

   code to execute
}
```

Example:

```
int age = 18;
if (age >= 18) {
   System.out.println("You are eligible to vote.");
}
```

2. if-else Statement

The `if-else` statement runs one block of code if the condition is **true**, and another if it is **false**.

Syntax:

```
if (condition) {
   // code if condition
} else {
   // code
}
```

Example:

```
int number = 5;
if (number % 2 == 0) {
  System.out.println("Even number");
} else {
  System.out.println("Odd number");
}
```

3. switch Statement

The `switch` statement is used to select one value from **multiple options**.

Syntax:

```
switch (expression) {
  case value1:
    // code
    break;
  case value2:
    // code
    break;
  ...
  default:
    // code if no case matches
}
```

Example:

```
int day = 2;
switch (day) {
  case 1:
    System.out.println("Sunday");
    break;
  case 2:
    System.out.println("Monday");
    break;
  default:
    System.out.println("Other day");
}
```

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

```java
public class evenodd {
    public static void main(String[] args) {
        int num = 4;

        if (num % 2 == 0) {
            System.out.println(num + " is Even");
        } else {
            System.out.println(num + " is Odd");
        }
    }
}
[Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\Assignment\" && javac
evenodd.java && java evenodd
4 is Even

[Done] exited with code=0 in 0.611 seconds
```

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

while loop:

- The **condition is checked first**, then the loop body runs.
- If the condition is **false at the beginning**, the loop will **not run at all**.

Syntax:

```
while (condition) {
   // code to execute
}
```

Example:

```
int i = 5;
while (i < 5) {
   System.out.println(i);
   i++;
}
// Output: (No output because condition is false initially)
```

do-while loop

- The loop body is executed **first**, then the condition is checked.
- So, the loop **always runs at least once**, even if the condition is false.

Syntax:

```
do {
   // code to execute
} while (condition);
```

Example:

```
int i = 5;
```

```
do {
   System.out.println(i);
   i++;
} while (i < 5);
// Output: 5
```

2.Object-Oriented Programming (OOPs)

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

There are **4 main principles** of OOP (Object-Oriented Programming):

1. Encapsulation

Definition:
Encapsulation is the process of wrapping data and methods (functions) into a single unit called a class.

It also hides the internal details of the object from the outside world using private access modifiers, and exposes them using getter and setter methods.

2. Abstraction

Definition:
Abstraction means **s**howing only the essential details to the user and hiding the unnecessary parts.

It helps in reducing complexity and increasing code clarity.

3. Inheritance

Definition:
Inheritance allows one class to inherit properties and methods from another class.
It helps in code reusability and creating a parent-child relationship.

4. Polymorphism

Definition:
Polymorphism means one name, many forms. It allows the same method to perform different behaviors depending on the object.

There are two types:

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

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

Class in Java:

A **class** in Java is a blueprint or template used to create objects.
It defines properties (variables) **and** behaviors (methods**)** of an object but doesn't hold actual values.

Object in Java:

An object is an instance of a class**.**
It is the actual thing that exists in memory and can be used to access the class's variables and methods.

Continuing the example, an object is the real car made using the design.

```java
class Car {

    String color = "Red";


    void drive() {
        System.out.println("Car is driving");
    }
}


public class Main {
    public static void main(String[] args) {

        Car myCar = new Car();


        System.out.println(myCar.color);
        myCar.drive();
    }
}
```

- **Class**: A blueprint (e.g., `Car`)
- **Object**: A real instance of a class (e.g., `myCar`)

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

```
class Rectangle {

    int length;
    int width;

    void calculateArea() {
        int area = length * width;
        System.out.println("Area of Rectangle: " + area);
    }
}

public class Arearec {
    public static void main(String[] args) {

        Rectangle rect = new Rectangle();


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


        rect.calculateArea();
    }
}
```

```
[Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\Assignment\" && javac
Arearec.java && java Arearec
Area of Rectangle: 50
```

```
[Done] exited with code=0 in 0.587 seconds
```

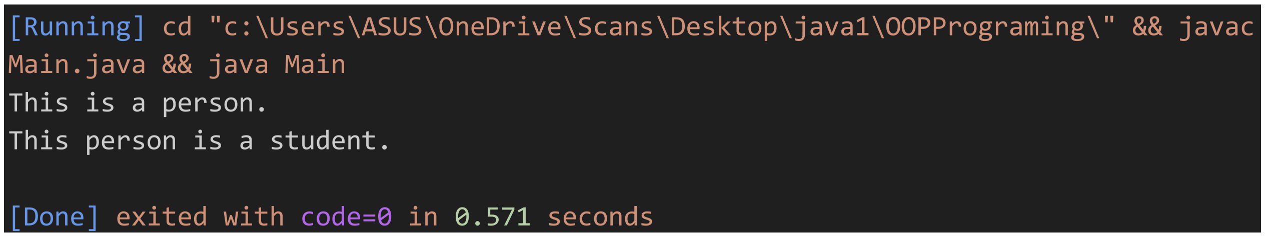
4.Explain inheritance with real-life example and Java code.

Inheritance is a feature in Java where one class (called the **child** or **subclass**) **inherits** the properties and behaviors (variables and methods) of another class (called the **parent** or **superclass**).

Real-Life Example: Vehicle and Car

Imagine a Vehicle is a general class that has common features like `start()` and `stop()`.
A **Car** is a type of Vehicle, so it can also `start()` and `stop()`, but it has its own method like `playMusic()`.

```java
class Person {
    void displayInfo() {
        System.out.println("This is a person.");
    }
}


class Student extends Person {
    void showDetails() {
        System.out.println("This person is a student.");
    }
}


public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        s.displayInfo();
        s.showDetails();
    }
}
```



```
[Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\java1\OOPPrograming\" && javac
Main.java && java Main
This is a person.
This person is a student.

[Done] exited with code=0 in 0.571 seconds
```

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

Polymorphism means "many forms".
In Java, it allows a method or object to behave in different ways based on the context.

Polymorphism increases flexibility, code reusability, and helps in writing clean and manageable code.

There are two types of polymorphism in Java:

1. Compile-Time Polymorphism (Method Overloading)

- Achieved by **defining multiple methods** with the same name but different **parameters** (number or type).
- The method that gets executed is decided **at compile time**.

Example:

```
public class Calculator {

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


  int add(int a, int b, int c) {
    return a + b + c;
  }

  public static void main(String[] args) {
    Calculator calc = new Calculator();
    System.out.println(calc.add(5, 10));     // Output: 15
    System.out.println(calc.add(2, 3, 4));   // Output: 9
  }
}
```

2. Runtime Polymorphism (Method Overriding)

- Achieved using **inheritance**.
- A **subclass overrides** a method of the **superclass**.
- The method that gets executed is decided **at runtime**, based on the object.

Example:

```java
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 Main {
    public static void main(String[] args) {
        Animal a;

        a = new Dog();
        a.sound();

        a = new Cat();
        a.sound();
    }
}
```

```
[Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\Assignment\" && javac
Main.java && java Main
Dog barks
Cat meows
```

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

→ Method Overloading (Compile-Time Polymorphism)

Definition:
Method overloading means defining multiple methods with the same name **but** different parameters (type, number, or order).
It happens within the same class and is resolved at compile time**.**

```
public class Main{


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


    int add(int a, int b, int c) {
        return a + b + c;
    }


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

    public static void main(String[] args) {
      Main obj = new Main();

        System.out.println(obj.add(10, 20));
        System.out.println(obj.add(5, 10, 15));
        System.out.println(obj.add(2.5, 3.5));
    }
}
[Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\Assignment\" && javac
Main.java && java Main
30
30
6.0
```

2. Method Overriding (Runtime Polymorphism)

Definition:
Method overriding means a subclass provides a specific implementation of a method that is already defined in its superclass**.**
It is resolved at runtime and requires inheritance.

```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 Main {
    public static void main(String[] args) {
        Animal a = new Dog();
        a.sound();
    }
}
```

```
Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\Assignment\" && javac Main.java
&& java Main
Dog barks

[Done] exited with code=0 in 0.56 seconds
```

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

Encapsulation is one of the main principles of Object-Oriented Programming (OOP).
It means hiding the internal details of a class and only exposing necessary parts through public methods.

In Java, encapsulation is achieved by:

1. Declaring data members as **`private`.**
2. Providing `public` **getter and setter** methods to access and update the data.

```java
class Student {

    private String name;


    public String getName() {
        return name;
    }


    public void setName(String newName) {
        name = newName;
    }
}

public class Main {
    public static void main(String[] args) {

        Student s1 = new Student();


        s1.setName("Vaishnavi");


        System.out.println("Student name: " + s1.getName());
    }
}
```

```
Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\Assignment\" && javac Main.java
&& java Main
Student name: Vaishnavi

[Done] exited with code=0 in 0.588 seconds
```

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

→ Abstraction in Java is the process of hiding the complex internal details and showing only the essential **features** to the user.

It helps reduce complexity and improves code clarity.

How is Abstraction Achieved in Java?

In Java, abstraction is achieved in two ways:

1. Using Abstract Classes
2. Using Interfaces

1. Abstract Class

- Declared using the `abstract` keyword.
- It can have abstract methods (without body) and concrete methods (with body).
- Cannot be instantiated directly.

8. Explain the difference between abstract class and interface.

→ Abstract Class in Java:

An abstract class is a class that is declared with the abstract keyword**.**
It can have:

- Abstract methods (without a body)
- Concrete methods (with a body)

It is used when you want to provide a base class with **s**ome shared code and some methods that must be implemented by child classes.

- Supports partial abstraction
- Can have constructors
- Can have instance variables
- Can be inherited using **`extends`**
- A class can extend only one abstract class

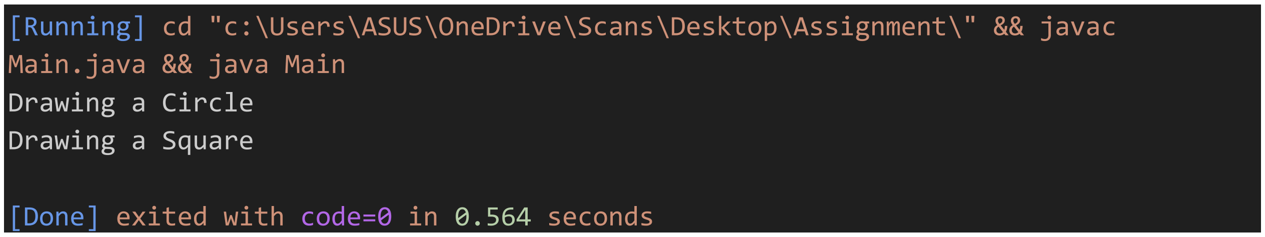
Interface in Java:

An **i**nterface is a blueprint of a class that contains only abstract methods (by default) and constants**.**
It is used to define a set of rules or behaviors that multiple unrelated classes can follow.

- Supports full abstraction (100%)
- All methods are abstract and public by default
- Variables are public, static, and final
- No constructors allowed
- A class can implement multiple interfaces

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

```java
interface Shape {
    void draw();
}

class Circle implements Shape {
    public void draw() {
        System.out.println("Drawing a Circle");
    }
}

class Square implements Shape {
    public void draw() {
        System.out.println("Drawing a Square");
    }
}

public class Main {
    public static void main(String[] args) {

        Shape s;

        s = new Circle();
        s.draw();

        s = new Square();
        s.draw();
    }
}
```



```
[Running] cd "c:\Users\ASUS\OneDrive\Scans\Desktop\Assignment\" && javac
Main.java && java Main
Drawing a Circle
Drawing a Square

[Done] exited with code=0 in 0.564 seconds
```

`Shape` is an interface with the method `draw()`.

`Circle` and `Square` are classes that implement the `Shape` interface.

The `draw()` method is overridden in each class to provide specific behavior.

In `main()`, the same interface reference is used to call methods of different classes — showing runtime polymorphism.