**PRACTICAL 1**

**What is Java? Explain Components of Java Architecture. Write down the steps to compile and run a Java Program. Write a program to print Hello World.**

Java is a high-level, object-oriented programming language designed to be platform-independent, secure, and robust. It is widely used for developing web applications, mobile applications, and enterprise systems. Java programs run on the Java Virtual Machine (JVM), which ensures portability across platforms.

### 1. **Java Development Kit (JDK)**:

The JDK is a **software development environment** designed for Java programming. It contains everything needed to **develop** Java applications, including:

* **Compiler (javac)**: Converts Java source code (.java) into bytecode (.class) that can be executed by the Java Virtual Machine (JVM).
* **Interpreter (java)**: Executes the compiled bytecode on the JVM.
* **Libraries**: A set of pre-built Java classes (in .jar files) that provide essential functionalities like string manipulation, file I/O, networking, and more. These libraries help developers avoid writing common functionality from scratch.
* **Debugger**: Helps in troubleshooting code by allowing step-by-step execution and inspection of variables.
* **Documentation tools**: Tools like javadoc to generate API documentation for Java code.

The JDK provides **everything** needed for both **development** and **execution** of Java programs, making it essential for anyone who writes Java code.

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

The JRE is a **runtime environment** required to **run** Java applications. It contains the necessary libraries and the Java Virtual Machine (JVM) to execute Java bytecode but **does not include development tools like the compiler**.

* **JVM**: The part of the JRE responsible for executing Java bytecode.
* **Libraries**: The JRE includes a set of standard libraries necessary to run Java programs. These libraries provide classes and methods to interact with the system, handle basic I/O operations, and manage the Java environment.
* **JRE** is typically used for **running** Java applications, not for developing them. It’s enough for users who want to **run** Java programs but don’t need the tools to compile or write them.

### 3. **Java Virtual Machine (JVM)**:

The JVM is an essential part of the Java architecture, providing the foundation for **platform independence** and **execution of Java programs**. It does the following:

* **Execution of Bytecode**: After a Java program is compiled into bytecode by the javac compiler, the JVM is responsible for **executing that bytecode** on the host machine. The JVM makes sure that the Java application runs similarly across all platforms (Windows, macOS, Linux, etc.).
* **Platform Independence**: Because the JVM abstracts the underlying operating system, Java programs can run on any platform that has a compatible JVM, making Java a **write-once, run-anywhere** language.
* **Memory Management**: The JVM automatically handles memory allocation and garbage collection. It keeps track of memory used by the program, deallocating memory when it is no longer needed.
* **Garbage Collection**: It periodically identifies and removes objects that are no longer referenced or used, freeing up memory.
* **Exception Handling**: The JVM also handles runtime exceptions and ensures the program behaves correctly by managing exceptions thrown during execution.

**JVM** is the key to Java's **portability**, as it abstracts platform-specific details.

### 4. **Java Application Programming Interface (API)**:

The Java API is a massive collection of **pre-written classes and methods** organized into package.

It provides **standard functionality** that Java developers can use to avoid reinventing the wheel. Key features include:

* **Core Libraries**: Java provides fundamental libraries for common tasks such as file I/O, networking, string handling, and database connectivity.
* **Standard Packages**: The API is divided into packages like:
  + java.lang: Contains fundamental classes like String, Math, Object, and Thread.
  + java.util: Provides utility classes like ArrayList, HashMap, Date, and Collections.
  + java.io: Handles input and output operations with files, streams, and serialization.
  + java.net: Provides classes for networking (e.g., sockets, HTTP).
  + java.sql: Contains classes for database interaction using JDBC (Java Database Connectivity).
* **Frameworks and Utilities**: Java also includes APIs for more complex tasks like graphical user interfaces (GUI) via javax.swing, multithreading via java.util.concurrent, and much more.
* **JavaDocs**: The official documentation that describes the API, detailing classes, methods, and their functionalities, making it easier for developers to use the right classes and methods.

The **Java API** is designed to help developers focus on writing business logic and functionality rather than common tasks like file manipulation or network communication. It saves a lot of time and effort by offering reusable components.

**1. Install Java Development Kit (JDK)**

Before proceeding, ensure that you have the Java Development Kit (JDK) installed on your system. You can download it from the official [Oracle website](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html).

**2. Set the PATH Environment Variable**

To compile and run Java programs from any directory, you need to set the PATH environment variable to include the bin directory of your JDK installation.

* **On Windows:**
  1. Right-click on This PC or Computer and click Properties.
  2. Click Advanced system settings on the left.
  3. In the System Properties window, click on the Environment Variables button.
  4. Under the System variables section, find the Path variable and click Edit.
  5. Add the path to the bin directory of your JDK installation (e.g., C:\Program Files\Java\jdk-11.0.x\bin).

**3. Steps to Set the PATH** **with CMD**

1. Press Win + X and select System (or right-click This PC and click Properties).
2. Click Advanced system settings on the left-hand side.
3. In the System Properties window, click Environment Variables.
4. In the Environment Variables window, under System variables, scroll down and select the Path variable, then click Edit.
5. In the Edit Environment Variable window, click New and then paste the path to the bin directory of your JDK installation. For example:

**cd C:\Users\YourUsername\Documents\**

**javac HelloWorld.java**

**java HelloWorld**

**Code:**

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Vansh Phogat 03814202023!");

System.out.println("Hello, World!");

}

}

**PRACTICAL 2**

**What WAP in Java To study the basics of programming:**

**What is the import Keyword in Java?**

In Java, the import keyword is used to bring external classes or entire packages into the program's namespace. This allows us to use predefined classes from the Java Standard Library, or custom classes that may be in separate files or packages. By importing classes, we can access their methods and functionality without needing to rewrite them.

For example:

**import java.util.\*;**

This imports all the classes from the java.util package, which is a commonly used package in Java.

**What is java.util?**

The java.util package is one of the core packages in the Java Standard Library. It provides a wide variety of utility classes that simplify common tasks in programming, such as handling collections (lists, sets, maps), manipulating dates and times, and working with random numbers.

Some common classes in java.util include:

* **Scanner (for reading input),**
* **Arrays (for working with arrays),**
* **ArrayList (for dynamic arrays),**
* **HashMap (for key-value storage).**

For example, to use Scanner, we can import it as:

**import java.util.Scanner;**

Key Classes in java.util

1. **Scanner Class:** The Scanner class is used to obtain input of primitive types (like int, double) and strings. It's commonly used to read user input from the console, files, or other input streams. You can create a Scanner object using:
2. **Scanner sc = new Scanner(System.in);**

It’s is a good practice to close it to release resources:

**sc.close();**

1. **Arrays Class:** The Arrays class provides methods for manipulating arrays. It allows for sorting, searching, and converting arrays. For instance, you can sort an array with the Arrays.sort() method.

Example:

**import java.util.Arrays;**

**int[] numbers = {3, 1, 4, 1, 5, 9};**

**Arrays.sort(numbers);**

**System.out.println(Arrays.toString(numbers));**

**Common Methods of Scanner**

* **nextInt():** Reads the next integer input from the user.
* int num = sc.nextInt();
* **nextLine()**: Reads the next complete line of text input.
* String name = sc.nextLine();
* **nextDouble()**: Reads the next double (floating-point number).
* double value = sc.nextDouble();

**What is public static void main(String[] args)?**

The main method is the entry point of any Java application. It is always required in Java applications that you run from the command line or an IDE. Let's break down the components:

* **public**: This is an access modifier. It means that the main method can be accessed from anywhere, meaning that the Java Virtual Machine (JVM) can call it to start the program.
* **static**: This indicates that the method belongs to the class itself, not to instances (objects) of the class. The JVM calls the main method before any objects of the class are created.
* void: The method doesn't return any value. It's used to execute the code and perform operations.
* **main**: This is the name of the method, and it's a special method recognized by the JVM as the starting point of the program.
* **String[] args**: This is an array of strings that can store command-line arguments. When you run the program from the command line, you can pass parameters to the main method. These are stored in the args array.

Example:

**public class HelloWorld {**

**public static void main(String[] args) {**

**System.out.println("Hello, World!");**

**}**

**}**

**What is System.out.println()?**

The System.out.println() statement is used to print messages or values to the console (standard output). It is commonly used for displaying output from a Java program.

* **System**: A built-in class in Java, part of the java.lang package, which provides access to system resources like input/output (I/O) streams, environment properties, and more.
* **out**: A static member of the System class, which is an instance of the PrintStream class. It represents the standard output stream (typically the console).
* **println()**: This method prints the specified message or value to the console, followed by a newline, which means the next output will appear on a new line.

Example:

**System.out.println("Hello, World!");**

This prints "Hello, World!" to the console and moves the cursor to the next line.

**PRACTICAL 2**

1. **Program to swap two numbers.**

**What is Swapping?**

Swapping is the process of exchanging the values of two variables. In Java, swapping can be done using various methods. The most common and simplest way is using a temporary variable, as shown in the following program.

**Code:**

import java.util.Scanner;

public class Swap {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

Scanner s = new Scanner(System.in);

System.out.println("Enter the value of a : ");

int a = s.nextInt();

System.out.println("Enter the value of b : ");

int b = s.nextInt();

System.out.println("\nBefore Swapping values:");

System.out.println("Value of a: " + a);

System.out.println("Value of b: " + b);

int temp = a;

a = b;

b = temp;

System.out.println("\nAfter Swapping values:");

System.out.println("Value of a: " + a);

System.out.println("Value of b: " + b);

}

}

**Explanation:**

The Java program begins with the class declaration Swap, which encapsulates all the code. In Java, every program must be inside a class. The main method, public static void main(String[] args), serves as the entry point and is executed automatically when the program runs. Within the main method, the program first prompts the user to enter values for two integer variables, a and b. These values are input by the user using the Scanner class. Next, System.out.println() is used to display the initial values of a and b, along with the user’s name and roll number.

The swapping logic is implemented using a temporary variable temp. First, temp stores the value of a, then a is assigned the value of b. Finally, b is assigned the value stored in temp, completing the swap.

After the swap, the program prints the new values of a and b to confirm that the swap has been successfully performed. This output verifies that the values of a and b were exchanged.

**PRACTICAL 2**

1. **Program to take input as an integer from the user and display the sum of two numbers.**

**Addition Program in Java**

Addition is one of the basic arithmetic operations where two numbers are added to produce a sum. The following Java program demonstrates how to take input from the user and perform addition.

**Code:**

import java.util.Scanner;

public class Sum {

public static void main(String[] args) {

int num1, num2;

Scanner scanner = new Scanner(System.in);

System.out.println("Name: Vansh Phogat\n03814202023");

System.out.println("Enter the number 1 :");

num1 = scanner.nextInt();

System.out.println("Enter the number 2 :");

num2 = scanner.nextInt();

int sum = num1 + num2;

System.out.println("Sum : " + sum);

scanner.close();

}

}

**Explanation:**

The Java program begins by importing the Scanner class using import java.util.Scanner; to enable the program to take input from the user. The Sum class is then declared to contain the entire program's logic.

Within the main method, two integer variables, num1 and num2, are declared to store the user-provided numbers.

The Scanner object scanner is created to read input from the console.

The program prompts the user to input two numbers, which are then stored in num1 and num2 using scanner.nextInt().

The sum of these two numbers is calculated by adding num1 and num2, and the result is stored in the variable sum.

Finally, the program displays the sum of the two numbers using System.out.println(). The Scanner object scanner is then closed to release system resources.

**PRACTICAL 2**

1. **Program to check whether the number is even or odd.**

**Even or Odd Program in Java**

This Java program checks whether a given number is even or odd. The program takes an integer input from the user and performs a check to determine if the number is divisible by 2 (even) or not (odd). The result is then displayed to the user.

**Code:**

import java.util.Scanner;

public class Check\_Num {

public static void main(String[] arg) {

int num;

System.out.println("Name: Vansh Phogat\n03814202023");

Scanner s = new Scanner(System.in);

System.out.println("Enter Num:");

num = s.nextInt();

if (num % 2 == 0) {

System.out.println(num + " is a even number ") ;

} else {

System.out.println(num + " is a odd number");

}

}

}

**Explanation:**

The Java program begins by importing the Scanner class from the java.util package, which allows the program to take input from the user through the console.

A class named **Check\_Num** is defined, and the program execution begins with the main method.

This is the entry point of the program, where execution starts when the program is run.

Within the main method, a Scanner object named s is created using new Scanner(System.in). This object is used to read input from the console (keyboard).

Next, the program prompts the user to input a number using System.out.println("Enter Num:") and reads the integer entered by the user using s.nextInt(). The value is stored in the variable num.

The core of the program is a conditional statement: if (num % 2 == 0). The modulus operator % checks the remainder when num is divided by 2. If the remainder is zero, the number is even; otherwise, it is odd. Based on this check, the program prints either "num is a even number" or "num is a odd number".

Finally, the program completes without explicitly closing the Scanner object, as it is not a strict requirement in this specific context, but it is a good practice to close resources after use.

**PRACTICAL 2**

1. **Program to print factorial of a number.**

**Factorial Program in Java**

This Java program calculates the factorial of a given number. The factorial of a number n is the product of all positive integers from 1 to n.

Example, 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120.

**Code:**

import java.util.Scanner;

public class Factorial {

public static void main(String[] args){

System.out.println("Name: Vansh Phogat\n03814202023");

Scanner s = new Scanner(System.in);

System.out.println("Enter a Number : ");

int num = s.nextInt();

int fact=1;

for(int i=num;i>0;i--){

fact=fact\*i;

}

System.out.println("Factorial of "+num+" is "+fact);

}

}

**Explanation:**

The Java program begins by importing the Scanner class from the java.util package, which allows the program to take input from the user through the console.

A class named Factorial is defined, and the program execution begins with the main method.

This is the entry point of the program, where execution starts when the program is run.

Within the main method, a Scanner object named s is created using new Scanner(System.in). This object is used to read input from the console (keyboard).

Next, the program prompts the user to input a number using System.out.println("Enter a Number : ") and reads the integer entered by the user using s.nextInt(). The value is stored in the variable num.

The core of the program is a for loop that calculates the factorial of the given number. The loop starts from the number num and iterates down to 1. In each iteration, the variable fact is multiplied by the loop counter i. Initially, fact is set to 1, and after the loop completes, it holds the calculated factorial value.

Finally, the program displays the factorial result using System.out.println(). The program does not explicitly close the Scanner object, which is a good practice to release system resources, though it is not included in the code.

**PRACTICAL 2**

1. **Program to check whether a given number is Armstrong or not.**

**Armstrong Program in Java**

This Java program checks whether a given number is an Armstrong number or not. An Armstrong number (or narcissistic number) for a given number of digits is a number that is equal to the sum of its own digits raised to the power of the number of digits.

Example : ++=153

**Code:**

import java.util.Scanner;  
  
public class ArmStrong {  
 public static void main(String[] args){  
 System.*out*.println("Name: Vansh Phogat\n03814202023");  
 Scanner s = new Scanner(System.*in*);  
 System.*out*.println("Enter a number");  
 int num = s.nextInt();  
 int temp\_num = num;  
 int armStrong = 0;  
 int digits = 0;  
 while (temp\_num > 0) {  
 temp\_num = temp\_num / 10;  
 digits++;  
 }  
 temp\_num = num;  
 while (temp\_num > 0) {  
 int remainder = temp\_num % 10;  
 temp\_num = temp\_num / 10;  
 armStrong += Math.*pow*(remainder, digits);  
 }  
 System.*out*.println("Armstrong sum: " + armStrong);  
 if (armStrong == num) {  
 System.*out*.println(num + " is an Armstrong number.");  
 } else {  
 System.*out*.println(num + " is not an Armstrong number.");  
 }  
 }  
}

**Explanation:**

The Java program begins by importing the Scanner class from the java.util package, which allows the program to take input from the user. It defines a class named ArmStrong, where execution starts with the main method.

1. **Taking Input**: Inside the method, a Scanner object is created to read an integer from the user, storing it in the variable num. The program prints the user’s name and roll number as well.
2. **Counting the Number of Digits**: The program calculates how many digits the input number has. This is done by using a while loop where temp\_num (a temporary copy of num) is divided by 10 until temp\_num becomes zero. Each division reduces the number by a factor of 10, and the digits counter is incremented. The final value of digits will be the number of digits in num.
3. **Calculating the Armstrong Sum**: After determining the number of digits, the program resets temp\_num to the original num value and processes each digit in the second while loop:
   * The last digit is obtained by taking the remainder of temp\_num divided by 10 (temp\_num % 10).
   * This digit is then raised to the power of the total number of digits using Math.pow(remainder, digits).
   * The result of this operation is added to the armStrong variable, which accumulates the sum of each digit raised to the power of the total number of digits.
   * The last digit is removed by performing integer division (temp\_num = temp\_num / 10).
4. **Comparison and Output**: Once all digits have been processed, the program compares the accumulated sum (armStrong) with the original number (num).
   * If they are equal, the program prints that the number is an Armstrong number.
   * Otherwise, it prints that the number is not an Armstrong number.

**PRACTICAL 3**

**Write a program declaring a class Rectangle with data member’s length and breadth and member functions Input, Output and CalcArea.**

**Rectangle Program in Java**

This Java program calculates the area of a rectangle by taking the length and breadth as inputs from the user.

**Code :**

import java.util.Scanner;  
  
public class Rectangle {  
 int length;  
 int breadth;  
 int area;  
 public void Input() {  
 Scanner s = new Scanner(System.*in*);  
 System.*out*.println("Enter length: ");  
 length = s.nextInt();  
 System.*out*.println("Enter breadth: ");  
 breadth = s.nextInt();  
 }  
 public void CalcArea() {  
 area = length \* breadth;  
 }  
 public void Output() {  
 System.*out*.println("Area of Rectangle: " + area);  
 }  
 public static void main(String[] args) {  
 System.*out*.println("Name: Vansh Phogat\n03814202023");  
 Rectangle r = new Rectangle();  
 r.Input();  
 r.CalcArea();  
 r.Output();  
 }  
}

**Explanation:**

1. **Class Declaration**: The program defines a class Rectangle that contains three data members: length, breadth, and area.
2. **Input Method**:
   * The Input() method uses the Scanner class to take the length and breadth of the rectangle from the user.
3. **Calculation Method**:
   * The CalcArea() method calculates the area of the rectangle by multiplying the length and breadth and stores it in the area variable.
4. **Output Method**:
   * The Output() method displays the area of the rectangle.
5. **Main Method**:
   * The main() method creates an object r of the Rectangle class and calls its methods in sequence: Input(), CalcArea(), and Output().

**PRACTICAL 4**

**Write a program to demonstrate use of method overloading to calculate area of square, rectangle and triangle.**

**Method Overloading to Calculate Area of Square, Rectangle, and Triangle**

Method overloading allows you to define multiple methods with the same name but with different parameter lists. In this case, we can create overloaded methods to calculate the area of a square, rectangle, and triangle based on the number and type of arguments passed.

**Code:**

public class AreaCalculator {  
 public int calculateArea(int side) {  
 return side \* side;  
 }  
 public int calculateArea(int length, int breadth) {  
 return length \* breadth;  
 }  
 public double calculateArea(double base, double height) {  
 return 0.5 \* base \* height;  
 }  
 public static void main(String[] args) {  
 System.*out*.println("Name: Vansh Phogat\n03814202023");  
 AreaCalculator calculator = new AreaCalculator();  
  
 int squareArea = calculator.calculateArea(4);  
 System.*out*.println("Area of Square: " + squareArea);  
  
 int rectangleArea = calculator.calculateArea(5, 10);  
 System.*out*.println("Area of Rectangle: " + rectangleArea);  
  
 double triangleArea = calculator.calculateArea(6, 8);  
 System.*out*.println("Area of Triangle: " + triangleArea);  
 }  
}

**Explanation:**

1. **Class Declaration**:
   * The program defines a class AreaCalculator which contains three overloaded methods named calculateArea.
2. **Method Overloading**:
   * The first method calculates the **area of a square** by taking one parameter (side), and the formula is side \* side.
   * The second method calculates the **area of a rectangle** by taking two parameters (length and breadth), and the formula is length \* breadth.
   * The third method calculates the **area of a triangle** by taking two parameters (base and height), and the formula is 0.5 \* base \* height.
3. **Main Method**:
   * In the main method, an object calculator of the AreaCalculator class is created.
   * The area of a square, rectangle, and triangle is calculated by calling the respective overloaded method with the appropriate parameters.

**PRACTICAL 5**

**Create a class employee which have name, age and address of employee, include methods getdata() and showdata(), getdata() takes the input from the user, showdata() display the data in following format:**

**Name:**

**Age:**

**Address:**

**Employee Class Program**

This program defines a class Employee with the attributes name, age, and address. The class includes methods getdata() to input the employee details and showdata() to display the details in the specified format.

**Code:**

import java.util.Scanner;  
  
public class Employee {  
  
 String name;  
 int age;  
 String address;  
  
 public void getdata() {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println("Enter Name: ");  
 name = scanner.nextLine();  
 System.*out*.println("Enter Age: ");  
 age = scanner.nextInt();  
 scanner.nextLine();  
 System.*out*.println("Enter Address: ");  
 address = scanner.nextLine();  
 }  
 public void showdata() {  
 System.*out*.println("Name: " + name);  
 System.*out*.println("Age: " + age);  
 System.*out*.println("Address: " + address);  
 }  
 public static void main(String[] args) {  
 System.*out*.println("Name: Vansh Phogat\n03814202023");  
 Employee emp = new Employee();  
 emp.getdata();  
 emp.showdata();  
 }  
}

**Explanation:**

1. **Class Declaration**:
   * The class Employee contains three data members: name, age, and address.
2. **Method getdata()**:
   * This method is used to input the employee's details.
   * It uses a Scanner object to read input for name, age, and address from the user.
   * scanner.nextLine() is used to capture strings, and scanner.nextInt() is used for integer input.
3. **Method showdata()**:
   * This method displays the employee's details (name, age, address) in the specified format.
4. **Main Method**:
   * The main() method creates an object emp of the Employee class.
   * The getdata() method is called to take input from the user, and the showdata() method is called to display the data.

**PRACTICAL 6**

**Write a program to demonstrate concept of “this”.**

The program demonstrates the concept of "this" keyword in Java. The this keyword refers to the current instance of the class and is often used to distinguish between instance variables and parameters with the same name. In the code, this.name is used to refer to the instance variable name, while name in the method welcome refers to the local parameter.

**Code:**

class game {

String name;

void welcome(String name) {

this.name = name;

System.out.println("Welcome to " + name);

}

}

public class Main {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

game g1 = new game();

g1.welcome("BattelGround India - BGMI");

}

}

**Explanation:**

1. **Class Declaration:**

* The game class contains a member variable name which stores the name of the game.

1. **Method welcome():**

* The method welcome(String name) takes a parameter named name and assigns it to the instance variable this.name. This distinguishes the instance variable from the local parameter name.
* It prints a message to the console welcoming the game.

1. **Main Method:**

* The Main class creates an object g1 of type game.
* It then calls the welcome() method with the argument "BattelGround India - BGMI", which sets the name and prints the welcome message.

**PRACTICAL 7**

**WAP to read the array dynamically, sort the array and display the sorted array.**

This program demonstrates how to read an array dynamically from the user, sort the array, and display the sorted array. It uses the Scanner class to input the number of elements and the array elements, then it uses the Arrays.sort() method to sort the array, and finally, the sorted array is displayed using Arrays.toString().

**Code:**

import java.util.Scanner;

import java.util.Arrays;

public class SortArray {

public static void main(String args[]){

System.out.println("Name: Vansh Phogat\n03814202023");

Scanner scan = new Scanner(System.in);

System.out.println("Enter the no of Elements : ");

int n = scan.nextInt();

int arr[] = new int[n];

for(int i=0;i<n;i++){

System.out.println("Enter " +(i+1)+" Elements : ");

arr[i] = scan.nextInt();

}

Arrays.sort(arr);

System.out.println("Sorted array : "+Arrays.toString(arr));

}

}

**Explanation:**

1. **Array Declaration and User Input:**

* The program first prompts the user to enter the number of elements in the array.
* It creates an array arr with the size n and then uses a loop to fill the array with user input values.

1. **Sorting the Array:**

* The Arrays.sort() method is used to sort the array in ascending order.

1. **Displaying the Sorted Array:**

* Finally, the sorted array is printed using Arrays.toString(arr) to display the contents in a readable format.

**PRACTICAL 8**

**Write a program to implement all string operations.**

This program demonstrates various string operations in Java. It takes user input for a string and then performs several operations such as calculating its length, concatenation, character access, substring extraction, comparison, search, replacement, conversion to uppercase and lowercase, trimming, checking if it's empty, and checking if it's a palindrome.

**Code:**

import java.util.Scanner;

public class StringOperations {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string: ");

String input = scanner.nextLine();

System.out.println("Length of the string: " + input.length());

System.out.print("Enter another string to concatenate: ");

String anotherString = scanner.nextLine();

System.out.println("Concatenated string: " + input + anotherString);

System.out.print("Enter index to access character: ");

int index = scanner.nextInt();

if (index >= 0 && index < input.length()) {

System.out.println("Character at index " + index + ": " + input.charAt(index));

} else {

System.out.println("Invalid index!");

}

System.out.print("Enter start index for substring: ");

int start = scanner.nextInt();

System.out.print("Enter end index for substring: ");

int end = scanner.nextInt();

if (start >= 0 && end <= input.length() && start < end) {

System.out.println("Substring: " + input.substring(start, end));

} else {

System.out.println("Invalid indices for substring!");

}

scanner.nextLine();

System.out.print("Enter another string to compare: ");

String compareString = scanner.nextLine();

if (input.equals(compareString)) {

System.out.println("The strings are equal.");

} else {

System.out.println("The strings are not equal.");

}

System.out.print("Enter a character or substring to search: ");

String searchString = scanner.nextLine();

if (input.contains(searchString)) {

System.out.println("Found '" + searchString + "' in the string.");

} else {

System.out.println("Substring not found.");

}

System.out.print("Enter character or substring to replace: ");

String toReplace = scanner.nextLine();

System.out.print("Enter replacement string: ");

String replacement = scanner.nextLine();

System.out.println("String after replacement: " + input.replace(toReplace, replacement));

System.out.println("Uppercase: " + input.toUpperCase());

System.out.println("Lowercase: " + input.toLowerCase());

System.out.println("Trimmed string: '" + input.trim() + "'");

if (input.isEmpty()) {

System.out.println("The string is empty.");

} else {

System.out.println("The string is not empty.");

}

String reversed = new StringBuilder(input).reverse().toString();

if (input.equals(reversed)) {

System.out.println("The string is a palindrome.");

} else {

System.out.println("The string is not a palindrome.");

}

scanner.close();

}

}

**Explanation:**

1. **String Operations:** The program handles various string operations like calculating length, concatenating, comparing, checking for substring, replacing characters, changing case, trimming spaces, and checking for palindromes.
2. **User Interaction:** It prompts the user to input strings for each operation and outputs the result accordingly.

**PRACTICAL 9**

**Write a program to demonstrate multi-level and hierarchical inheritance.**

This program demonstrates both **multi-level** and **hierarchical inheritance** in Java. In multi-level inheritance, a class is derived from another class, which in turn is derived from another class. In hierarchical inheritance, multiple classes inherit from a single base class. The program shows how a Dog and Cat class inherit from the Animal class, and how the Puppy class extends the Dog class, illustrating both inheritance types.

**Code:**

class Animal {

void eat() {

System.out.println("This animal eats food.");

}

}

class Dog extends Animal {

void bark() {

System.out.println("The dog barks.");

}

}

class Puppy extends Dog {

void play() {

System.out.println("The puppy is playing.");

}

}

class Cat extends Animal {

void meow() {

System.out.println("The cat meows.");

}

}

// Main class to demonstrate the inheritance

public class AnimalCreatures {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

// Multi-level

Puppy puppy = new Puppy();

puppy.eat();

puppy.bark();

puppy.play();

System.out.println();

// Hierarchical

Dog dog = new Dog();

dog.eat();

dog.bark();

Cat cat = new Cat();

cat.eat();

cat.meow();

}

}

**Explanation:**

1. **Multi-level Inheritance:**

* The Puppy class extends the Dog class, which itself extends the Animal class. This creates a chain of inheritance: Puppy → Dog → Animal.
* The Puppy class inherits the eat() method from Animal and bark() from Dog, in addition to its own play() method.

1. **Hierarchical Inheritance:**

* The Dog and Cat classes both inherit from the Animal class. This means both classes share the eat() method from the Animal class, but each has its own additional functionality: bark() for Dog and meow() for Cat.

**PRACTICAL 10**

**Write a program to demonstrate run-time polymorphism.**

This program demonstrates **run-time polymorphism** (also known as dynamic method dispatch) in Java. Run-time polymorphism allows a method to behave differently based on the object that invokes it, even if the object is referred to by a reference of the base class. In this example, the sound() method is overridden in the Dog1 and Cat1 classes, but the appropriate version of the method is invoked based on the actual object type (either Dog1 or Cat1), not the reference type (Animal1).

**Code:**

class Animal1 {

// Method to be overridden

public void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog1 extends Animal1 {

@Override

public void sound() {

System.out.println("Dog barks");

}

}

class Cat1 extends Animal1 {

@Override

public void sound() {

System.out.println("Cat meows");

}

}

public class AnimalPloy {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

Animal1 myAnimal = new Animal1();

Animal1 myDog = new Dog1();

Animal1 myCat = new Cat1();

myAnimal.sound();

myDog.sound();

myCat.sound();

}

}

**Explanation:**

1. **Base Class (Animal1):**

* The sound() method is defined in the base class Animal1, and it prints a generic message: "Animal makes a sound."

1. **Overridden Methods:**

* The Dog1 and Cat1 classes extend Animal1 and override the sound() method. In Dog1, it prints "Dog barks", and in Cat1, it prints "Cat meows".

1. **Run-time Polymorphism:**

* In the main() method, objects of Animal1, Dog1, and Cat1 are created, but they are all referred to by the Animal1 reference type.
* The actual method that gets called is determined at run-time based on the object's actual type (Dog1 or Cat1), not the reference type (Animal1). This is the core concept of run-time polymorphism.

**PRACTICAL 11**

**Write a program to use super() to invoke base class constructor.**

This program demonstrates the use of super() to invoke the constructor of the base class (Person) from the subclass (Student). The super() keyword is used to call the parent class's constructor before executing the subclass's constructor code. This ensures that the parent class is properly initialized before the subclass is initialized.

**Code:**

class Person {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

System.out.println("Person constructor called: " + name + ", Age: " + age);

}

}

class Student extends Person {

String studentId;

public Student(String name, int age, String studentId) {

super(name, age);

this.studentId = studentId;

System.out.println("Student constructor called: Student ID: " + studentId);

}

}

public class College {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

Student student = new Student("Alice", 20, "S12345");

}

}

**Explanation:**

1. **Base Class (Person):**

* The Person class has a constructor that takes name and age as parameters and initializes the respective fields. It then prints a message containing the name and age.

1. **Subclass (Student):**

* The Student class extends Person and adds a new field studentId. Its constructor takes name, age, and studentId as parameters.
* The super(name, age) call invokes the constructor of the Person class, initializing the fields inherited from Person. After that, the studentId field is initialized, and a message specific to the student is printed.

1. **Using super():**

* The super(name, age) call ensures that the Person class is properly initialized before the Student class adds its own initialization. This is necessary because the Student class inherits from Person, and the parent class must be initialized first.

1. **Output:**

* When a Student object is created in the main method, the program first calls the Person constructor via super(), then the Student constructor.

**PRACTICAL 12**

**Write a program to demonstrate the use of static variable, static method and static block.**

This program demonstrates the use of static variables, static methods, and static blocks in Java. A static variable is shared among all instances of a class, a static method belongs to the class rather than an instance, and a static block is used for static initialization that runs once when the class is loaded.

**Code:**

class MyClass {

static int staticVariable = 0;

static {

System.out.println("Static block is executed.");

staticVariable = 10;

}

public static void staticMethod() {

System.out.println("This is a static method. Static variable value: " + staticVariable);

}

public MyClass() {

System.out.println("Constructor is called.");

staticVariable++;

}

}

public class bca {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

System.out.println("Static variable value: " + MyClass.staticVariable);

MyClass.staticMethod();

MyClass obj = new MyClass();

System.out.println("After creating object, static variable value: " + MyClass.staticVariable);

MyClass.staticMethod();

}

}

**Explanation:**

1. **Static Variable (staticVariable):**

* staticVariable is a class variable shared by all instances of the class. It is initialized to 0 but modified later by a static block and the constructor.

1. **Static Block:**

* The static block is executed once when the class is first loaded. In this case, it initializes the staticVariable to 10 and prints "Static block is executed."

1. **Static Method (staticMethod()):**

* The staticMethod() is a class method that can be called without creating an object of the class. It prints the value of the staticVariable.

1. **Constructor:**

* The constructor of MyClass increments the staticVariable by 1 each time an object is created. This shows that static variables are shared across instances.

1. **Main Method (bca Class):**

* The main method demonstrates the usage of static elements:
* It prints the value of the static variable before and after creating an object of MyClass.
* It calls the static method before and after object creation.

**PRACTICAL 13**

**Write a program to demonstrate the concept of abstract class**

This program demonstrates the concept of **abstract classes** in Java. An abstract class is a class that cannot be instantiated on its own and may contain abstract methods (methods without a body) that must be implemented by subclasses. In this example, we have an abstract Car class with an abstract method startEngine() and a concrete method displayDetails(). The subclasses Sedan and SUV extend the Car class and provide their own implementations of the startEngine() method.

**Code:**

abstract class Car {

String brand;

int year;

public Car(String brand, int year) {

this.brand = brand;

this.year = year;

}

public abstract void startEngine();

public void displayDetails() {

System.out.println("Brand: " + brand);

System.out.println("Year: " + year);

}

}

class Sedan extends Car {

public Sedan(String brand, int year) {

super(brand, year);

}

public void startEngine() {

System.out.println("The " + brand + " sedan's engine starts with a smooth hum.");

}

}

class SUV extends Car {

public SUV(String brand, int year) {

super(brand, year);

}

public void startEngine() {

System.out.println("The " + brand + " SUV's engine roars to life.");

}

}

public class MetalBox {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

Car sedan = new Sedan("Toyota", 2023);

Car suv = new SUV("Ford", 2022);

sedan.displayDetails();

sedan.startEngine();

System.out.println();

suv.displayDetails();

suv.startEngine();

}

}

**Explanation:**

1. **Abstract Method:** startEngine() is declared abstract in the Car class and must be implemented in the subclasses Sedan and SUV.
2. **Abstract Class:** The Car class cannot be instantiated directly, but its subclasses (Sedan and SUV) can be.
3. **Polymorphism:** The startEngine() method is polymorphic, as it behaves differently depending on the type of car (either Sedan or SUV).

**PRACTICAL 14**

**Write a program to demonstrate the concept of interface**

This program demonstrates the concept of **interfaces** in Java. Interfaces define a contract of methods that a class must implement. In this example, we have three interfaces: Drivable, Fuelable, and ElectricVehicle. The Car1 class implements Drivable and Fuelable, while the ElectricScooter class implements Drivable and ElectricVehicle. This illustrates how multiple interfaces can be implemented by a single class.

**Code:**

interface Drivable {

void drive();

}

interface Fuelable {

void refuel();

}

interface ElectricVehicle {

void charge();

}

class Car1 implements Drivable, Fuelable {

String brand;

public Car1(String brand) {

this.brand = brand;

}

@Override

public void drive() {

System.out.println(brand + " car is driving on the road.");

}

@Override

public void refuel() {

System.out.println(brand + " car is refueling with gasoline.");

}

}

class ElectricScooter implements Drivable, ElectricVehicle {

String brand;

public ElectricScooter(String brand) {

this.brand = brand;

}

@Override

public void drive() {

System.out.println(brand + " electric scooter is riding on the road.");

}

@Override

public void charge() {

System.out.println(brand + " electric scooter is charging.");

}

}

public class Fuel {

public static void main(String[] args) {

System.out.println("Name: Vansh Phogat\n03814202023");

Car1 car = new Car1("Toyota");

ElectricScooter scooter = new ElectricScooter("Xiaomi");

car.drive();

scooter.drive();

car.refuel();

scooter.charge();

}

}

**Explanation:**

1. **Interfaces (Drivable, Fuelable, ElectricVehicle):**

* Drivable: Defines the method drive() that any class implementing this interface must provide.
* Fuelable: Defines the method refuel() for refueling a fuel-powered vehicle.
* ElectricVehicle: Defines the method charge() for charging an electric vehicle.

1. **Classes Implementing Interfaces:**

* Car1: Implements both the Drivable and Fuelable interfaces. It provides the implementation for the drive() method (indicating that a car drives) and the refuel() method (indicating that a car refuels with gasoline).
* ElectricScooter: Implements the Drivable and ElectricVehicle interfaces. It provides the implementation for the drive() method (indicating that an electric scooter rides) and the charge() method (indicating that an electric scooter charges).

1. **Main Method (Fuel Class):**

* In the main() method, objects of Car1 and ElectricScooter are created and the appropriate methods are called for each object:
* car.drive(): Drives the car.
* scooter.drive(): Rides the electric scooter.
* car.refuel(): Refuels the car.
* scooter.charge(): Charges the electric scooter.