JAVA PRACTICALS

Note:-First you verify through sir, then you write in the file (This is not officially created by sir)

1. Write down a program for creating a Calculator with its basic operations and display the appropriate output.

```
-> import java.util.Scanner;

public class Calculator {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      double num1, num2, result = 0;

      char operator;


      System.out.println("=== Basic Calculator ===");


      // Take first number

      System.out.print("Enter first number: ");

      num1 = sc.nextDouble();


      // Take operator

      System.out.print("Enter operator (+, -, *, /): ");

      operator = sc.next().charAt(0);


      // Take second number

      System.out.print("Enter second number: ");

      num2 = sc.nextDouble();


      // Perform calculation
```

```
switch (operator) {
    case '+':
        result = num1 + num2;
        System.out.println("Result = " + result);
        break;
    case '-':
        result = num1 - num2;
        System.out.println("Result = " + result);
        break;
    case '*':
        result = num1 * num2;
        System.out.println("Result = " + result);
        break;
    case '/':
        if (num2 == 0) {
            System.out.println("Error: Cannot divide by zero!");
        } else {
            result = num1 / num2;
            System.out.println("Result = " + result);
        }
        break;
    default:
        System.out.println("Invalid operator!");
}

sc.close();
```

```
    }
}
```

OUTPUT:-

```
=== Basic Calculator ===
Enter first number: 10
Enter operator (+, -, *, /): *
Enter second number: 5
Result = 50.0
```

Description:-

- Takes two numbers and an operator as input.
- Performs the selected operation using a switch statement.
- Handles divide-by-zero case properly.

2. Write a program in Java to find maximum of three numbers with appropriate output.

```
->public class MaxOfThree {
    public static void main(String[] args) {
        // Assign values directly
        int num1 = 45;
        int num2 = 62;
        int num3 = 17;

        // Find the maximum
        int max = num1;

        if (num2 > max) {
            max = num2;
        }
        if (num3 > max) {
            max = num3;
        }
        // Output the result
        System.out.println("The numbers are: " + num1 + ", " + num2 + ", " +
num3);
        System.out.println("The maximum of the three numbers is: " + max);
    }
}
```

OUTPUT:-
```
The numbers are: 45, 62, 17
The maximum of the three numbers is: 62
```

Description:- The program prints the three predefined numbers and then displays the maximum among them. It compares the numbers using simple if conditions and stores the largest value in the max variable.

3. Write a program in Java to multiply two matrix and display the appropriate output.

```
->public class MatrixMultiplication {
    public static void main(String[] args) {
        // Define first matrix (2x3)
        int[][] matrix1 = {
            {2, 4},
            {3, 5}
        };

        // Define second matrix (3x2)
        int[][] matrix2 = {
            {1, 3},
            {2, 4}
        };

        // Result matrix will be of size 2x2
        int[][] result = new int[2][2];

        // Matrix multiplication logic
        for (int i = 0; i < 2; i++) { // rows of matrix1
            for (int j = 0; j < 2; j++) { // columns of matrix2
                for (int k = 0; k < 2; k++) { // columns of matrix1 / rows of matrix2
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }

        // Display result
        System.out.println("Result of Matrix Multiplication:");
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                System.out.print(result[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

OUTPUT:- Result of Matrix Multiplication:

```
10      22
13      29
```

Description:- The program multiplies two predefined matrices:

- matrix1 (2 rows × 3 columns)
- matrix2 (3 rows × 2 columns)

The result is a new 2×2 matrix, where each element is computed by taking the dot product of rows from the first matrix and columns from the second.

4. Write a program in Java to demonstrate the use of operators with appropriate output.

4.1 Ternory 4.2 Left Shift 4.3 Right Shift 4.4 Increment & Decrement

-><u>4.1 ternory operator:-</u>

```
        public class TernaryOperator {
                public static void main(String[] args) {
                int a = 10, b = 20;
                int max = (a > b) ? a : b;
                System.out.println("Ternary Operator: The maximum of " + a + "
and " + b + " is " + max);
   }
}
```

OUTPUT:- Ternary Operator: The maximum of 10 and 20 is 20

<u>Description</u>:-This program compares two numbers (a and b) and uses the **ternary operator** ? : to determine and print the **maximum** of the two.

- **Use**: Short-form if-else condition.

<u>4.2 Left Shift:-</u>

```
public class LeftShiftOperator {
   public static void main(String[] args) {
      int num = 5; // Binary: 00000101
      int result = num << 2; // Multiply by 2^2 = 4
      System.out.println("Left Shift Operator: " + num + " << 2 = " + result);
   }
}
```

OUTPUT:- Left Shift Operator: 5 << 2 = 20

Description:- This program shifts the bits of a number **to the left** using <<. Shifting left by n multiplies the number by 2^n.

- **Example**: 5 << 2 = 20 (5 × 4)

4.3 Right Shift:-

```
public class RightShiftOperator {
    public static void main(String[] args) {
        int num = 20; // Binary: 00010100
        int result = num >> 2; // Divide by 2^2 = 4
        System.out.println("Right Shift Operator: " + num + " >> 2 = " + result);
    }
}
```

OUTPUT:- Right Shift Operator: 20 >> 2 = 5

Description:- This program shifts the bits of a number **to the right** using >>. Shifting right by n divides the number by 2^n.

- **Example**: 20 >> 2 = 5 (20 ÷ 4)

4.4 Increment & Decrement:-

```
public class IncDecOperator {
    public static void main(String[] args) {
        int x = 7;
        System.out.println("Original value of x: " + x);
        System.out.println("Post-increment (x++): " + (x++)); // Prints 7, then x becomes 8
        System.out.println("After post-increment, x = " + x);
        System.out.println("Pre-increment (++x): " + (++x)); // x becomes 9, then prints 9
        System.out.println("Post-decrement (x--): " + (x--)); // Prints 9, then x becomes 8
        System.out.println("After post-decrement, x = " + x);
        System.out.println("Pre-decrement (--x): " + (--x)); // x becomes 7, then prints 7
    }
}
```

OUTPUT:-

```
Original value of x: 7
Post-increment (x++): 7
After post-increment, x = 8
Pre-increment (++x): 9
Post-decrement (x--): 9
After post-decrement, x = 8
Pre-decrement (--x): 7
```

Description:- This program demonstrates **post** and **pre** increment (++) and decrement (--) on a variable.

- **Post-increment**: x++ — use first, then increment
- **Pre-increment**: ++x — increment first, then use
- Similarly for decrement

5. Write a program in Java to demonstrate the use of below features with their inbuilt methods and display the appropriate output.

5.1 String 5.2 StringBuffer 5.3 StringBuilder 5.4 Collection Framework 5.5 Type Casting

->**5.1 String:-**

```
public class StringDemo {
   public static void main(String[] args) {
      String str = "Hello World";


      System.out.println("Original String: " + str);
      System.out.println("Uppercase: " + str.toUpperCase());
      System.out.println("Lowercase: " + str.toLowerCase());
      System.out.println("Length: " + str.length());
      System.out.println("Substring (0 to 5): " + str.substring(0, 5));
      System.out.println("Character at index 1: " + str.charAt(1));
   }
}
```

OUTPUT:-
```
Original String: Hello World
Uppercase: HELLO WORLD
Lowercase: hello world
Length: 11
Substring (0 to 5): Hello
Character at index 1: e
```

Description:- Demonstrates the use of the immutable String class and its inbuilt methods like .toUpperCase(), .length(), .substring(), etc.

- Use: Common operations on text data.

5.2 StringBuffer:-

```
public class StringBufferDemo {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("Java");

        sb.append(" Programming");
        sb.insert(5, "is Fun ");
        sb.replace(0, 4, "JAVA");
        sb.reverse();

        System.out.println("Final StringBuffer: " + sb);
    }
}
```

OUTPUT:- Final StringBuffer: gninargorP nuF si AVAJ

Description:- Shows how to use StringBuffer, a mutable and thread-safe class for handling strings.
Methods used: .append(), .insert(), .replace(), .reverse()

- Use: When multiple threads need to safely modify strings.

5.3 StringBuilder:-

```
public class StringBuilderDemo {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Welcome");
        sb.append(" to Java");
        sb.delete(0, 3);
        sb.insert(0, "Say ");
        sb.replace(4, 11, "Hello");

        System.out.println("Final StringBuilder: " + sb);
    }
}
```

OUTPUT:- Final StringBuilder: Say Hello Java

Description:- Demonstrates StringBuilder, a mutable but non-thread-safe version of StringBuffer, offering better performance.
Methods used: .append(), .delete(), .insert(), .replace()

- Use: When performance is more important than thread safety.

5.4 Collection Framework:-

```
import java.util.*;

public class CollectionDemo {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();

        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");

        System.out.println("Fruits List: " + fruits);
        fruits.remove("Banana");
        fruits.add("Orange");

        System.out.println("Updated Fruits List: " + fruits);
    }
}
```

OUTPUT:- Fruits List: [Apple, Banana, Mango]

Updated Fruits List: [Apple, Mango, Orange]

Description:- Uses ArrayList from the Java Collection Framework to store and manage a dynamic list of strings.
Methods used: .add(), .remove()

- Use: Efficiently handle groups of objects like lists, sets, or maps.

5.5 Type Casting:-

```
public class TypeCastingDemo {
    public static void main(String[] args) {
        // Implicit casting (Widening)
        int a = 10;
        double b = a;
        System.out.println("Implicit Casting (int to double): " + b);

        // Explicit casting (Narrowing)
        double x = 9.78;
        int y = (int) x;
        System.out.println("Explicit Casting (double to int): " + y);
    }
}
```

OUTPUT:- Implicit Casting (int to double): 10.0

Explicit Casting (double to int): 9

Description:- Shows both implicit casting (e.g., int to double) and explicit casting (e.g., double to int).

- Use: Convert between different data types in Java.

6. Write a program in Java to demonstrate the use of single inheritance, multilevel inheritance, hierarchical inheritance and multiple inheritance.

->Single Inheritance:-

```
class Animal {
   void eat() {
      System.out.println("Animal eats food");
   }
}
class Dog extends Animal {
   void bark() {
      System.out.println("Dog barks");
   }
}
public class SingleInheritance {
   public static void main(String[] args) {
      Dog d = new Dog();
      d.eat();   // inherited from Animal
      d.bark();  // from Dog
   }
}
```

OUTPUT:-
```
Animal eats food
Dog barks
```

Description:- Dog inherits from Animal, so it can access both its own method bark() and the parent method eat().

The program creates a Dog object and calls both methods, demonstrating inheritance.

<u>multilevel inheritance:-</u>

```
class Animal {
   void eat() {
      System.out.println("Animal eats");
   }
}
class Dog extends Animal {
   void bark() {
      System.out.println("Dog barks");
   }
}
class Puppy extends Dog {
   void weep() {
      System.out.println("Puppy weeps");
   }
}
public class MultilevelInheritance {
   public static void main(String[] args) {
      Puppy p = new Puppy();
      p.eat();   // from Animal
      p.bark();  // from Dog
      p.weep();  // from Puppy
   }
}
```

OUTPUT:- Animal eats

```
Dog barks

Puppy weeps
```

<u>Description:-</u> Puppy inherits from Dog, and Dog inherits from Animal.

This creates a multilevel inheritance chain.

A Puppy object can access methods from all three classes, proving multilevel inheritance.

<u>Hierarchical Inheritance:-</u>

```
class Animal {
   void eat() {
      System.out.println("Animal eats");
   }
}
class Dog extends Animal {
   void bark() {
      System.out.println("Dog barks");
   }
}
class Cat extends Animal {
   void meow() {
      System.out.println("Cat meows");
   }
}
```

```
public class HierarchicalInheritance {
   public static void main(String[] args) {
       Dog d = new Dog();
       d.eat();
       d.bark();

       Cat c = new Cat();
       c.eat();
       c.meow();
   }
}
```

```
OUTPUT:- Animal eats
Dog barks
Animal eats
Cat meows
```

Description:- Both Dog and Cat inherit from Animal.

The Dog object can access eat() from Animal and its own bark().

The Cat object can access eat() from Animal and its own meow().

This shows **multiple child classes sharing a single parent class**.

Multiple Inheritance:-

```
interface Printable {
   void print();
}
```

```
interface Showable {
   void show();
}
class Document implements Printable, Showable {
   public void print() {
      System.out.println("Printing document");
   }
   public void show() {
      System.out.println("Showing document");
   }
}
public class MultipleInheritance {
   public static void main(String[] args) {
      Document doc = new Document();
      doc.print();
      doc.show();
   }
}
```

OUTPUT:- Printing document

Showing document

Description:- Document implements two interfaces: Printable and Showable.

It provides implementations for both print() and show().

This shows multiple inheritance through interfaces, as Java does not allow it through classes.

7. Write a program in Java to demonstrate the difference between abstract class and interface with appropriate output.

->

```
// Interface example
interface Vehicle {
   void start();
   void stop();
}

// Abstract class example
abstract class Machine {
   abstract void operate(); // abstract method
   void status() {
      System.out.println("Machine is functioning.");
   }
}

// Class implementing interface and extending abstract class
class Car extends Machine implements Vehicle {
   public void start() {
      System.out.println("Car is starting...");
   }

   public void stop() {
      System.out.println("Car is stopping...");
   }
```

```
    void operate() {
        System.out.println("Car is operating on road.");
    }
}
public class AbstractVsInterface {
    public static void main(String[] args) {
        Car myCar = new Car();

        System.out.println("=== Interface Methods ===");
        myCar.start();
        myCar.stop();

        System.out.println("\n=== Abstract Class Methods ===");
        myCar.operate();
        myCar.status();
    }
}
```

OUTPUT:-

```
=== Interface Methods ===
Car is starting...
Car is stopping...

=== Abstract Class Methods ===
Car is operating on road.
Machine is functioning.
```

<u>Description</u>:- Car is starting... & Car is stopping...
→ From Vehicle interface methods, showing how Car implements interface behavior.

Car is operating on road.
→ From abstract method operate() in Machine, overridden by Car.

Machine is functioning.
→ From concrete method in abstract class, showing abstract classes can have implemented methods.

8. Write a java program to demonstrate the use of below features with appropriate output.

8.1 Static Ploymorpishm 8.2 Dynamic Ploymorpishm 8.3 Constructor Overloading

->8.1 Static Ploymorpishm:-

```
public class StaticPolymorphism {
    void show() {
        System.out.println("No parameters");
    }

    void show(String name) {
        System.out.println("Name: " + name);
    }

    void show(int a, int b) {
        System.out.println("Sum: " + (a + b));
    }

    public static void main(String[] args) {
        StaticPolymorphism obj = new StaticPolymorphism();
        obj.show();
        obj.show("Java");
        obj.show(5, 10);
    }
}
```

```
OUTPUT:- No parameters
Name: Java
Sum: 15
```

Description:- Different versions of the show() method are called based on the number/type of arguments.

- Shows compile-time (static) method selection.

8.2 Dynamic Polymorphism:-

```
class Animal {
  void sound() {
    System.out.println("Animal makes sound");
  }
}
class Dog extends Animal {
  void sound() {
    System.out.println("Dog barks");
  }
}
public class DynamicPolymorphism {
  public static void main(String[] args) {
    Animal a = new Dog(); // Reference of parent, object of child
    a.sound(); // Calls Dog's version
  }
}
```

```
OUTPUT:- Dog barks
```

Description:- Even though the reference is of type Animal, the method in Dog is called at runtime.

Demonstrates run-time (dynamic) method resolution.

8.3 Constructor Overloading:-

```
public class ConstructorOverloading {
   ConstructorOverloading() {
      System.out.println("Default constructor");
   }
   ConstructorOverloading(String name) {
      System.out.println("Hello, " + name);
   }
   ConstructorOverloading(int a, int b) {
      System.out.println("Sum: " + (a + b));
   }
   public static void main(String[] args) {
      new ConstructorOverloading();
      new ConstructorOverloading("Java");
      new ConstructorOverloading(10, 20);
   }
}
```

```
OUTPUT:- Default constructor
Hello, Java
Sum: 30
```

Description:- Three constructors with different parameters are called.

Each one performs a different task, based on input.

Shows how constructors can be overloaded like methods.

9. Write a program in Java to demonstrate use of below keywords with appropriate output.

9.1 static 9.2 super 9.3 final 9.4 this

->9.1 static Keyword :-

```
public class StaticDemo {
   static int count = 0; // static variable


   static void displayCount() { // static method
      System.out.println("Static count: " + count);
   }
   public static void main(String[] args) {
      count = 5;
      StaticDemo.displayCount();
   }
}
```

OUTPUT:- Static count: 5

Description:- A static variable and method are accessed without creating an object — shared across all instances.

9.2 super Keyword:-

```
class Parent {
   void message() {
      System.out.println("Message from Parent");
   }
}
```

```
class Child extends Parent {
   void message() {
      super.message(); // calls Parent class method
      System.out.println("Message from Child");
   }
}

public class SuperDemo {
   public static void main(String[] args) {
      Child c = new Child();
      c.message();
   }
}
```

```
OUTPUT:- Message from Parent
Message from Child
```

<u>Description:-</u> super calls the parent class method before executing the child's method — shows inheritance and method overriding.

<u>9.3 final Keyword:-</u>

```
public class FinalDemo {
   final int speedLimit = 90; // final variable

   void showLimit() {
      System.out.println("Speed limit is: " + speedLimit);
   }
   public static void main(String[] args) {
      FinalDemo obj = new FinalDemo();
      obj.showLimit();
      // obj.speedLimit = 100; // ❌ would cause compile-time error
   }
}
```

OUTPUT:- Speed limit is: 90

Description:- The final variable cannot be changed once assigned — enforces constant values.

9.4 this Keyword:-

```
public class ThisDemo {
    int id;
    String name;

    ThisDemo(int id, String name) {
        this.id = id;        // refers to current object
        this.name = name;
    }

    void display() {
        System.out.println("ID: " + this.id + ", Name: " + this.name);
    }

    public static void main(String[] args) {
        ThisDemo obj = new ThisDemo(1, "Java");
        obj.display();
    }
}
```

OUTPUT:- ID: 1, Name: Java

Description:- this keyword refers to the current object, helping differentiate between instance and local variables.

10. Write a program in Java to demonstrate the use of Exception Handling mechanism with appropriate output.

->
```
public class ExceptionHandlingDemo {
    public static void main(String[] args) {
        try {
            int a = 10, b = 0;
            int result = a / b; // This will throw ArithmeticException
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        } finally {
            System.out.println("Finally block always executes.");
        }
    }
}
```

OUTPUT:-
```
Error: Cannot divide by zero!
Finally block always executes.
```

<u>Description:-</u> Error: Cannot divide by zero!
→ Caught ArithmeticException when dividing by zero.

Finally block always executes.
→ finally block runs regardless of exception — useful for cleanup.

11. Write a program in Java to demonstrate the use of Custom Exception with appropriate output.

```
->// Custom exception class
class InvalidAgeException extends Exception {
   public InvalidAgeException(String message) {
      super(message);
   }
}
public class CustomExceptionDemo {

   // Method to check age
   public static void checkAge(int age) throws InvalidAgeException {
      if (age < 18) {
         throw new InvalidAgeException("Age must be 18 or above. Given: " +
age);
      } else {
         System.out.println("Age is valid for registration.");
      }
   }
   // Main method
public static void main(String[] args) {
      int[] testAges = {16, 18, 21};

      for (int age : testAges) {
         try {
            System.out.println("Checking age: " + age);
            checkAge(age);
            } catch (InvalidAgeException e) {
                System.out.println("Exception caught: " + e.getMessage());
            }
            System.out.println(); // for spacing
        }
    }
}
```

OUTPUT:-

```
Checking age: 16
Exception caught: Age must be 18 or above. Given: 16

Checking age: 18
Age is valid for registration.

Checking age: 21
Age is valid for registration.
```

<u>Description:-</u>

The program tests multiple age values:

- For age 16, the custom exception InvalidAgeException is thrown and caught, displaying an error message.
- For ages 18 and 21, the age is considered valid, and a success message is printed.

This demonstrates how a custom exception can handle specific error conditions in a controlled way.

12. Write a program in Java to demonstrate the use of thread life cycle and display the appropriate output.

->
```
class MyThread extends Thread {
   public void run() {
      try {
         System.out.println("Thread is running...");
         Thread.sleep(1000); // TIMED_WAITING state
         System.out.println("Thread completed.");
      } catch (InterruptedException e) {
         System.out.println("Thread interrupted.");
      }
   }
}
public class ThreadLifeCycleDemo {
   public static void main(String[] args) {
      MyThread t = new MyThread();

      // Thread in NEW state
      System.out.println("Thread state after creation: " + t.getState());

      // Start the thread - it moves to RUNNABLE state
      t.start();
      System.out.println("Thread state after start(): " + t.getState());

      try {
         // Give the thread time to enter TIMED_WAITING due to sleep()
         Thread.sleep(100);
```

```
            System.out.println("Thread state during sleep(): " + t.getState());

            // Wait for thread to finish
            t.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }
        // After completion - TERMINATED state
        System.out.println("Thread state after completion: " + t.getState());
    }
}
```

OUTPUT:-

```
Thread state after creation: NEW
Thread is running...
Thread state after start(): RUNNABLE
Thread state during sleep(): TIMED_WAITING
Thread completed.
Thread state after completion: TERMINATED
```

Description:-

- NEW: Thread is created but not yet started.
- RUNNABLE: After calling start(), thread becomes eligible to run.
- TIMED_WAITING: Thread is sleeping using sleep().
- TERMINATED: Thread has finished execution.

This program prints the thread's state at each key phase, clearly showing the Thread Life Cycle in action.

}

13. Write down a program in Java to demonstrate the use of thread synchronization with appropriate output.

->
```
// Shared resource class
class Table {
   // synchronized method to prevent race conditions
   synchronized void printTable(int n) {
      for (int i = 1; i <= 5; i++) {
         System.out.println(n + " x " + i + " = " + (n * i));
         try {
            Thread.sleep(400); // simulate delay
         } catch (InterruptedException e) {
            System.out.println(e);
         }
      }
   }
}
// Thread class 1
class MyThread1 extends Thread {
   Table t;
   MyThread1(Table t) {
      this.t = t;
   }
   public void run() {
      t.printTable(5);
   }
}
```

```
// Thread class 2
class MyThread2 extends Thread {
    Table t;
    MyThread2(Table t) {
        this.t = t;
    }
    public void run() {
        t.printTable(10);
    }
}
// Main class
public class ThreadSyncDemo {
    public static void main(String[] args) {
        Table obj = new Table(); // shared object

        MyThread1 t1 = new MyThread1(obj);
        MyThread2 t2 = new MyThread2(obj);

        t1.start();
        t2.start();
    }
}
```

OUTPUT:-

```
5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

10 x 1 = 10

10 x 2 = 20

10 x 3 = 30

10 x 4 = 40

10 x 5 = 50
```

Description:-

- Two threads attempt to print multiplication tables (5 and 10) using the same shared object.

- The synchronized method ensures one thread executes at a time, preventing mixed or interleaved outputs.

- This demonstrates thread synchronization, ensuring data consistency and safe access to shared resources.

14. Write down a program to demonstrate the use of applet life cycle with appropriate output.

15. Create a desktop application in Java which contents the different GUI components with its appropriate event.