|  |  |
| --- | --- |
| Description: Logo, company name  Description automatically generated | **NAGARJUNA COLLEGE OF ENGINEERING AND TECHNOLOGY**  **(An autonomous college under VTU) DEPARTMENT OF CSE (DATA SCIENCE)** |

BE 3rd Sem

Object Oriented Programming Using Java (IC) Lab Programs

1)Program Name: Advanced Arithemetic calculator

import java.util.Scanner;

public class SimpleCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String history = "";

while (true) {

System.out.print("Enter expression (operand1 operator operand2): ");

int operand1 = scanner.nextInt();

char operator = scanner.next().charAt(0);

int operand2 = scanner.nextInt();

int result = calculate(operand1, operator, operand2);

if (result != Integer.MIN\_VALUE) {

history += operand1 + " " + operator + " " + operand2 + " = " + result + "**\**n";

System.out.println("Result: " + result);

}

System.out.print("Do you want to perform another calculation? (yes/no): ");

if (!scanner.next().equalsIgnoreCase("yes")) {

break;

}

}

if (!history.isEmpty()) {

System.out.println("\nCalculation History:");

System.out.println(history);

}

System.out.println("Calculator terminated.");

scanner.close();

}

public static int calculate(int operand1, char operator, int operand2) {

switch (operator) {

case '+': return operand1 + operand2;

case '-': return operand1 - operand2;

case '\*': return operand1 \* operand2;

case '/': return operand2 != 0 ? operand1 / operand2 : Integer.MIN\_VALUE;

default:

System.out.println("Error: Invalid operator.");

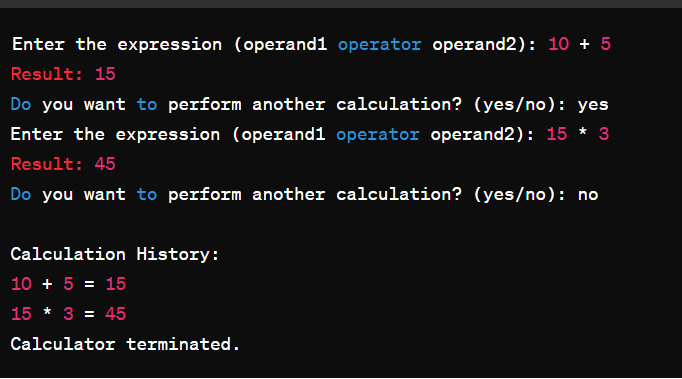
return Integer.MIN\_VALUE;

}

}

}

Output:



1 B) Program Name:Fibonaaci Generation

import java.util.Scanner;

public class FibonacciGenerator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of terms in the Fibonacci series: ");

int n = scanner.nextInt();

int a = 0, b = 1;

System.out.println("Fibonacci series up to " + n + " terms:");

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

System.out.print(a + " ");

int temp = b;

b = a + b;

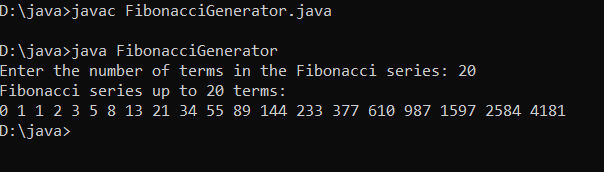
a = temp;

}

scanner.close();

}

}



1. (A) Develop a Java program showcasing method overloading with a base class "Phone" containing the dial() method, and two subclasses "CameraPhone" and "SmartPhone"

class Phone {

public void dial() {

System.out.println("Dialing a call...");

}

}

class CameraPhone extends Phone {

public void dial(String contact) {

System.out.println("Dialing " + contact + " and capturing a photo...");

}

}

class SmartPhone extends Phone {

public void dial(String contact) {

System.out.println("Dialing " + contact + " with smart features...");

}

}

public class PhoneDemo {

public static void main(String[] args) {

Phone basicPhone = new Phone();

CameraPhone cameraPhone = new CameraPhone();

SmartPhone smartPhone = new SmartPhone();

System.out.println("Basic Phone:");

basicPhone.dial();

System.out.println("\nCamera Phone:");

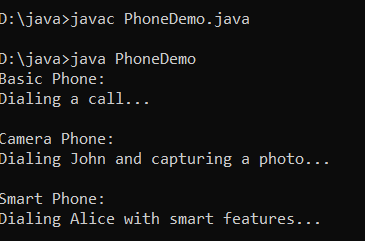
cameraPhone.dial("John");

System.out.println("\nSmart Phone:");

smartPhone.dial("Alice");

}

}



1. B)Develop a Java program illustrating constructor overloading for calculating the area of a rectangle and a circle using appropriate constructors.

class Shape {

protected double area;

public Shape() {

area = 0;

}

public Shape(double area) {

this.area = area;

}

}

class Rectangle extends Shape {

private double length, width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

area = calculateArea();

}

private double calculateArea() {

return length \* width;

}

public void display() {

System.out.println("Rectangle - Length: " + length + ", Width: " + width + ", Area: " + area);

}

}

class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

area = calculateArea();

}

private double calculateArea() {

return Math.PI \* radius \* radius;

}

public void display() {

System.out.println("Circle - Radius: " + radius + ", Area: " + area);

}

}

public class ShapeDemo {

public static void main(String[] args) {

Rectangle rectangle = new Rectangle(5.0, 4.0);

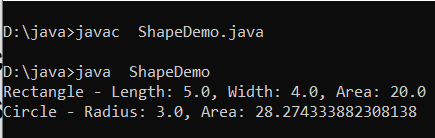
Circle circle = new Circle(3.0);

rectangle.display();

circle.display();

}

}



1. A)Create a Java program with a vehicle hierarchy, including Vehicle, Car, SportsCar, and Truck classes..

**// Parent class**

**class Vehicle {**

**void start() {**

**System.out.println("Vehicle started.");**

**}**

**void stop() {**

**System.out.println("Vehicle stopped.");**

**}**

**}**

**// Child class inheriting from Vehicle**

**class Car extends Vehicle {**

**void accelerate() {**

**System.out.println("Car is accelerating.");**

**}**

**void brake() {**

**System.out.println("Car is braking.");**

**}**

**}**

**// Grandchild class inheriting from Car**

**class SportsCar extends Car {**

**void boost() {**

**System.out.println("Sports car is boosting.");**

**}**

**}**

**// Main class to demonstrate multilevel inheritance**

**public class VehicleDemo {**

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

**SportsCar sportsCar = new SportsCar();**

**// Accessing methods from the Vehicle class**

**sportsCar.start(); // Inherited from Vehicle**

**sportsCar.stop(); // Inherited from Vehicle**

**// Accessing methods from the Car class**

**sportsCar.accelerate(); // Inherited from Car**

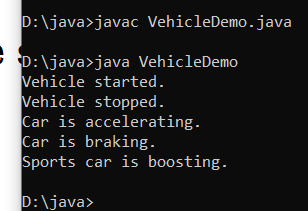
**sportsCar.brake(); // Inherited from Car**

**// Accessing methods from the SportsCar class**

**sportsCar.boost(); // Defined in SportsCar**

**}**

**}**



3 B)Program Name:Implement Bank Interest Rate of ICCI,SBI,AXIS banks etc.. using Abstract class

**abstract class Bank{**

**abstract int getRateOfInterest();**

**}**

**class SBI extends Bank{**

**int getRateOfInterest(){return 7;}**

**}**

**class PNB extends Bank{**

**int getRateOfInterest(){return 8;}**

**}**

**class TestBank{**

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

**Bank b;**

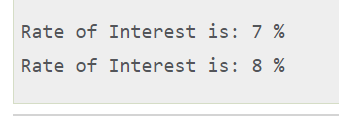
**b=new SBI();**

**System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");**

**b=new PNB();**

**System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");**

**}}**



1. Develop a Java lab program that handles exceptions for division by zero and invalid input. Use `try-catch` blocks to catch `ArithmeticException` for division by zero and

`InputMismatchException` for non-integer input and provide user-friendly error messages.

import java.util.InputMismatchException; import java.util.Scanner;

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

Scanner scanner = new Scanner(System.in);

try {

System.out.print("Enter an integer numerator: "); int numerator = scanner.nextInt();

System.out.print("Enter an integer denominator: "); int denominator = scanner.nextInt();

int result = divide(numerator, denominator); System.out.println("Result of division: " + result);

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero is not allowed.");

} catch (InputMismatchException e) { System.out.println("Error: Please enter valid integers.");

} finally { scanner.close();

}

}

private static int divide(int numerator, int denominator) { if (denominator == 0) {

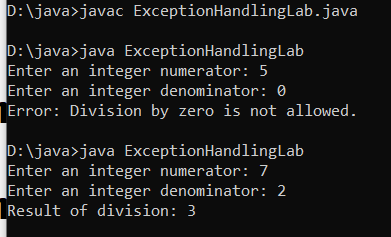
throw new ArithmeticException("Division by zero");

}

return numerator / denominator;

}

}



5(A)Program Name:Java program to implement When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

class Table{

synchronized void printTable(int n){//synchronized method

for(int i=1;i<=5;i++){

System.out.println(n\*i);

try{

Thread.sleep(400);

}catch(Exception e){System.out.println(e);}

}

}

}

class MyThread1 extends Thread{

Table t;

MyThread1(Table t){

this.t=t;

}

public void run(){

t.printTable(5);

}

}

class MyThread2 extends Thread{

Table t;

MyThread2(Table t){

this.t=t;

}

public void run(){

t.printTable(100);

}

}

public class TestSynchronization2{

public static void main(String args[]){

Table obj = new Table();//only one object

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

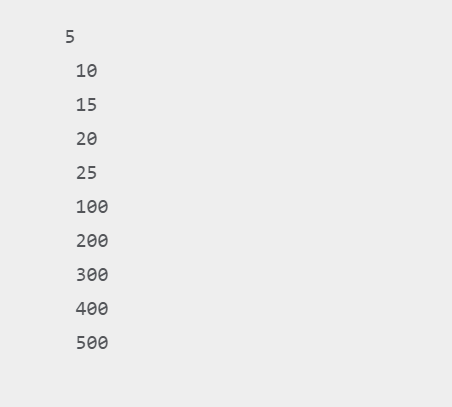
t1.start();

t2.start();

}

}

Output:



5B)Design a Java lab program to demonstrate string handling, including creating strings using constructors and literals, concatenating strings, extracting characters at a specified index, and comparing strings for equality.

public class StringHandlingLab {

public static void main(String[] args) {

// Creating strings using literals

String literalString1 = "Hello, ";

String literalString2 = "world!";

// Concatenating strings

String result = literalString1 + literalString2;

// Extracting characters at a specified index

char charAtIndex = result.charAt(7);

// Comparing strings for equality

boolean areEqual = literalString1.equals("Hello, ");

// Displaying the results

System.out.println("Using literals:");

System.out.println(literalString1);

System.out.println(literalString2);

System.out.println("\nConcatenated String:");

System.out.println(result);

System.out.println("\nCharacter at index 7: " + charAtIndex);

System.out.println("\nAre constructorString1 and literalString1 equal? " + areEqual);

}

}

OUTPUT:

