**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**ASSIGNMENT**



**Department of Computer Science Engineering**

**Amrita School of Computing**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Name:M.SAI MOKSHITHA**

**Verified By Roll No: 24151**

|  |  |
| --- | --- |
| **Submitted by** | |
| Name | M.SAI MOKSHITHA |
| Roll No | AV.SC.U4CSE24151 |
| Year/Sem/Section | 1st YEAR/SEM-2/CSE-B |
| Date of Submission |  |
| **Submitted to** | |
| Name | Dr. B Raj Kumar |
| Department | CSE |
| Designation | Asst. Professor |

|  |  |
| --- | --- |
| Marks |  |

|  |  |  |
| --- | --- | --- |
| S.NO | Q.No | PAGE No. |
| 1. | 1 |  |
| 2. | 2 |  |
| 3. | 3,19,23,26,29,30 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**PROGRAM-1:**

**AIM:** Write a java program with class named “book”, the class should contain various attributes such as title, author, year of publication it should also contain a constructor with parameters which initializes, title, author, and year of publication.

Create a method which displays the details of the book and display the details of two books.

**CODE:**

public class Book{

public String Title;

public String Author;

public String Year\_of\_publication;

public Book(String Title, String Author, String Year\_of\_publication){

this.Title=Title;

this.Author=Author;

this.Year\_of\_publication=Year\_of\_publication;

}

public void Bookdetails()

{

System.out.println("Title:"+Title);

System.out.println("Author:"+Author);

System.out.println("Year\_of\_publication:"+Year\_of\_publication);

}

public static void main (String[] args){

Book book1=new Book("Mahabaratam","Vyasa","3rd century");

Book book2=new Book("Ramayanam","Valmiki","5th century");

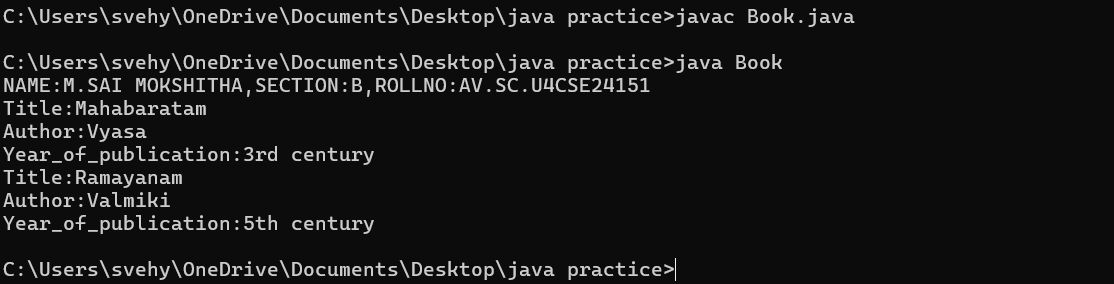
book1.Bookdetails();

book2.Bookdetails();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Not putting the semi-colon; after calling the function. 2. After Start, Stop, Service not giving the parenthesis ( ). | 1. Put the semi-colon after the writing the code. 2. After every method, put the parenthesis ( ). |

**IMPORTANT POINTS**

1. While defining two classes for a code, we must be sure that we save both the classes in separate files.
2. While defining a method we should also define a function to call that method.

**PROGRAM-2:**

**AIM:**Write a java program with class named “MyClass”, with a static variable “count” of “int” type, initialized to “0” and a constant variable “PI” of type “double” initialized to 3.14159 as attributes of that class. Now define a constructor for “MyClass” that increments the “count” variable each time an object of “MyClass” is created.Finally print the final values of “count” and “PI” variables.

**CODE**:

public class myclass {

static int count = 0;

final double pi = 3.14;

public myclass() {

count++;

}

public static void main(String[] args) {

System.out.println(NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151");

myclass a = new myclass();

myclass b = new myclass();

myclass c = new myclass();

myclass d = new myclass();

myclass e = new myclass();

System.out.println("count: " + count);

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

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

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

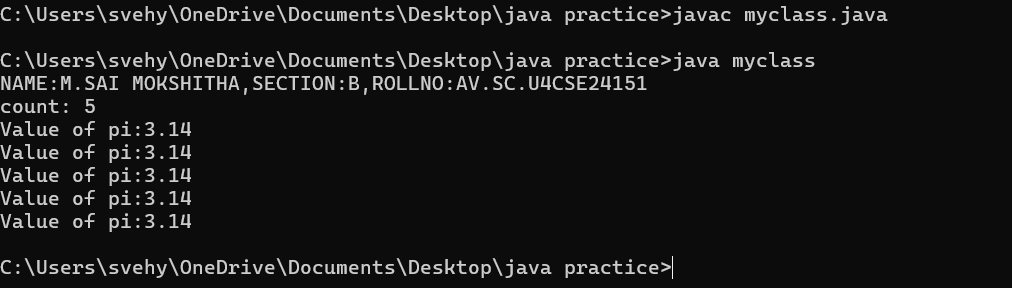
System.out.println("Value of pi:"+d.pi);

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

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Incorrect datatype entered. 2. Not giving the indentation properly. | 1. Enter the correct input during the run-time. 2. All the indentation must be correct to run the code correct. |

**IMPORTANT POINTS:**

1. We must declare the initial value of the variable before declaring the final one.
2. Here the main objective is to increase the count according to the number of objects we make, i.e the count increases when the no.of objects are increasing.

**PROGRAM-3:**

**AIM:**Define a Java class named **VisibilityExample** with the following attributes and methods:

Attributes:

* A public integer variable named publicVariable, initialized to 10.
* A private integer variable named privateVariable, initialized to 20.

Methods:

* A public method named publicMethod() that prints "This is a public method."
* A private method named privateMethod() that prints "This is a private method."
* In a separate Java class named **Main**, write the main method to demonstrate accessing the members of the VisibilityExample class:
* Create an object of the VisibilityExample class.
* Access and print the value of the public variable publicVariable.
* Call the public method publicMethod().

**PROGRAM-4:**

**AIM:**.Write a Java program that takes a number from the user and generates an integer between 1 and 7. It displays the weekday name (Use Conditional Statements).

import java.util.Scanner;

class WeekdayConditional {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Input number (1-7): ");

int numb = sc.nextInt();

String day;

if (numb == 1) {

day = "Sunday";

} else if (numb == 2) {

day = "Monday";

} else if (numb == 3) {

day = "Tuesday";

} else if (numb == 4) {

day = "Wednesday";

} else if (numb == 5) {

day = "Thursday";

} else if (numb == 6) {

day = "Friday";

} else if (numb == 7) {

day = "Saturday";

} else {

day = "Invalid input! Please enter a number between 1 and 7.";

}

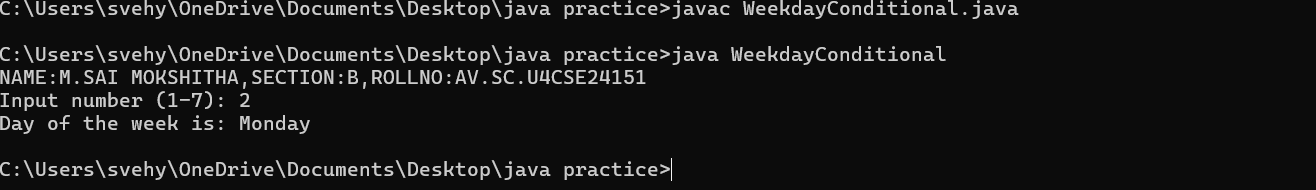
System.out.println("Day of the week is: " + day);

sc.close();

}

}

**Output:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. While printing the variable not giving + sign. 2. Not closing the scanner. | 1. We should give correct indentation. 2. Closing the scanner is must. |

**IMPORTANT POINTS:**

1. While declaring a variable, first we must initialize the variable,
2. Here, we used the Scanner package to take the inputs from the user instead of declaring it beforehand.
3. After using the package we close the scanner package.

**PROGRAM-5:**

**AIM:**Write a Java program to display the multiplication table of a given integer.

Input the number (Table to be calculated) : Input number of terms : 5

**Code:**

import java.util.Scanner;

public class MultiplicationTable {

public static void main(String[] args) {

System.out.println(

“NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151”);

Scanner scanner = new Scanner(System.in);

System.out.print("Input the number (Table to be calculated): ");

int number = scanner.nextInt();

System.out.print("Input number of terms: ");

int terms = scanner.nextInt();

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

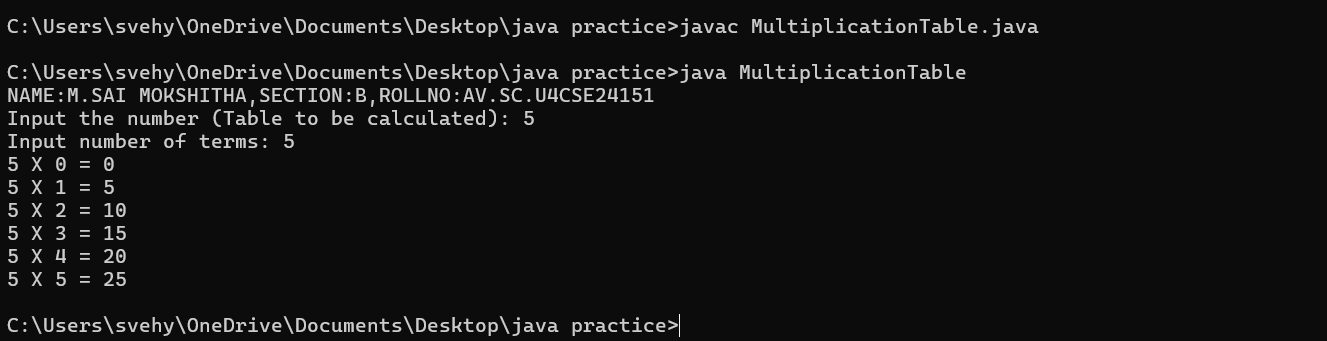
System.out.println(number + " X " + i + " = " + (number \* i));

}

scanner.close();

}

}

**OUTPUT:**  


**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Error while printing the variables. 2. Incorrect declaration of integer. | 1. Give the plus sign while printing. 2. Give input.nextInt(), where I should be capital. |

**IMPORTANT POINTS:**

1. Here, we used the Scanner package to take the inputs from the user instead of declaring it beforehand.
2. After using the package we close the scanner package.

**PROGRAM-6:**

**AIM:**Write a Java program that reads two floating-point numbers and tests whether they are the same up to three decimal places (Use Conditional Statements).

import java.util.Scanner;

public class CompareFloatNumbers {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151");

Scanner scanner = new Scanner(System.in);

System.out.print("Input floating-point number: ");

double num1 = scanner.nextDouble();

System.out.print("Input floating-point another number: ");

double num2 = scanner.nextDouble();

num1 = Math.round(num1 \* 1000) / 1000.0;

num2 = Math.round(num2 \* 1000) / 1000.0;

if (num1 == num2) {

System.out.println("They are the same up to three decimal places.");

} else {

System.out.println("They are different.");

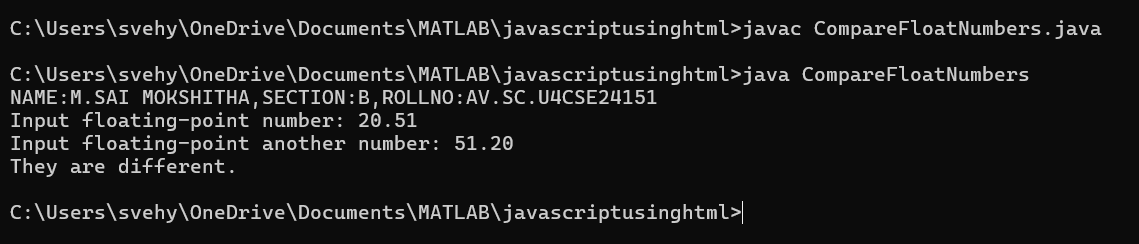
}

scanner.close();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Invalid datatype. 2. The print statement should be closed at last. | 1. Decalre double instead of int. 2. Add the ‘;’ after each and every statement. |

**IMPORTANT POINTS:**

1. The ‘double’ data type in java states that it can calculate upto any decimal places.

Here, the if and else statement is used as if both the floating-point numbers are equal it prints the if statement or else print

**PROGRAM-7:**

**AIM:**Write a program that accepts three numbers from the user and prints "increasing" if the numbers are in increasing order, "decreasing" if the numbers are in decreasing order, and "Neither increasing or decreasing order" otherwise (Use Conditional Statements).

import java.util.Scanner;

public class NumberOrder {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151");

Scanner scanner = new Scanner(System.in);

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

int num1 = scanner.nextInt();

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

int num2 = scanner.nextInt();

System.out.print("Input third number: ");

int num3 = scanner.nextInt();

if (num1 < num2 && num2 < num3) {

System.out.println("Increasing order");

} else if (num1 > num2 && num2 > num3) {

System.out.println("Decreasing order");

} else {

System.out.println("Neither increasing nor decreasing order");

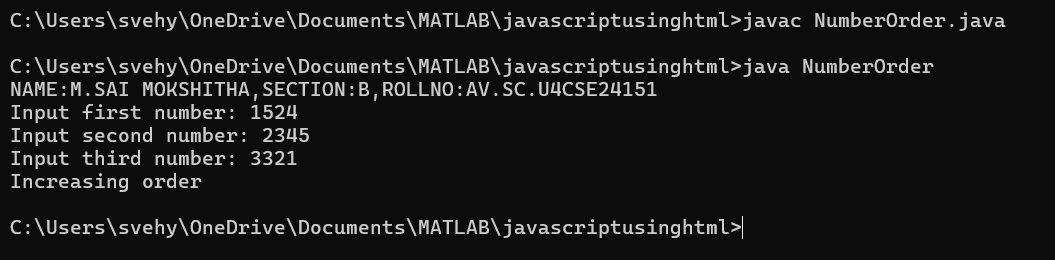
}

scanner.close();

}

}

OUTPUT:



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Incorrect data type. 2. Didn’t close the input. | 1. Declare int instead of Stirng. 2. Close the input at last like ‘input.close();’ |

**IMPORTANT POINTS**:

1. Here, we are finding whether the given numbers are in the increasing order or decreasing order.
2. We do it by using the if and else statements,.
3. According to the given condition in the question, if a<b and b<c, then it prints in increasing order or it prints in decreasing order or it prints neither increasing nor decreasing.

**PROGRAM-8:**

**AIM:**Write a Java program that reads a positive integer and count the number of digits the number (less than ten billion) has (Use Conditional Statements).

**CODE:**

import java.util.Scanner;

public class DigitCounter {

public static void main(String[] args) {

System.out.println("NAME:M.SAMOKSHITHA,SECTION:B,ROLL NO:AV.SCU4CSE24151");

Scanner scanner = new Scanner(System.in);

System.out.print("Input an integer number less than ten billion: ");

long number = scanner.nextLong();

if (number < 0) {

System.out.println("Please enter a positive number.");

} else if (number >= 10\_000\_000\_000L) {

System.out.println("Number should be less than ten billion.");

} else {

int digitCount = 0;

if (number == 0) {

digitCount = 1;

} else {

long temp = number;

while (temp != 0) {

temp /= 10;

digitCount++;

}

}

System.out.println("Number of digits in the number: " + digitCount);

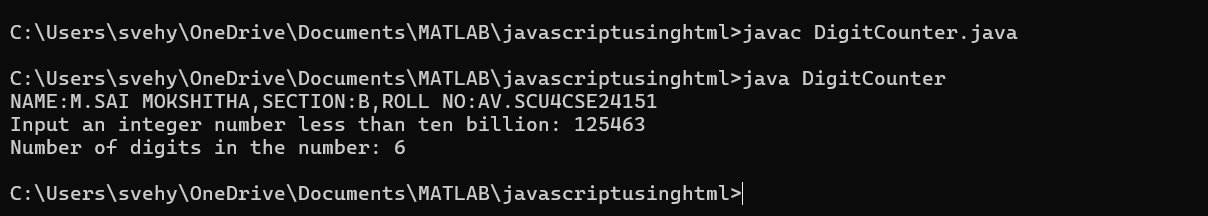
}

scanner.close();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Variable I not found. 2. Not giving the value of a to b. | 1. Initialize variable I to 0. 2. Giving the value of a to b prints the correct output. |

**IMPORTANT POINTS:**

1. Here, we are using the while loop to execute our program.
2. In the while loop, we are dividing the number by 10 and then increasing the count number.
3. The loop continues until the value of a becomes zero.

**PROGRAM-9:**

**AIM:**Write a Java program to display Pascal's triangle.

CODE:

import java.util.Scanner;

public class PascalTriangle {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLL NO:AV.SC.U4CSE24151");

Scanner scanner = new Scanner(System.in);

System.out.print("Input number of rows: ");

int rows = scanner.nextInt();

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

int number = 1;

System.out.format("%" + (rows - i) \* 2 + "s", "");

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

System.out.format("%4d", number);

number = number \* (i - j) / (j + 1);

}

System.out.println();

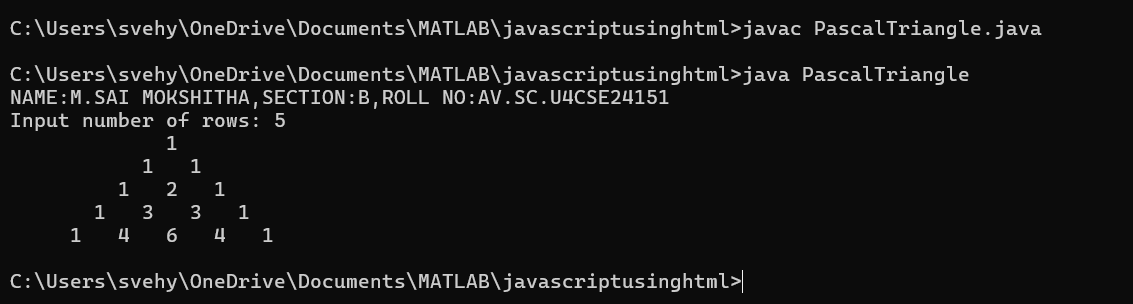
}

scanner.close();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Wrong comma in the for loop. 2. Using println instead of using print. | 1. Put the ‘;’ istead of ‘,’ in the for loop. 2. Thought it runs the code it doesn’t give the right output. |

**IMPORTANT POINTS:**

1. The first inner loop controls the spaces before the numbers in each row to ensure the triangle is properly aligned.
2. R – I calculates the number of spaces needed for the current row.
3. The formula n = n\*(i-j)/(j+1) is a way of calculating the binomial coefficient c(i,j) which represents the value at the jth position in the ith row.

**PROGRAM-10:**

**AIM:**Write a Java program to display the following character rhombus structure.

**Code:**

import java.util.Scanner;

public class DiamondPattern {

public static void main(String[] args) {

System.out.println("NAME:M.SAMOKSHITHA,SECTION:B,ROLL NO:AV.SC.U4CSE24151");

Scanner sc = new Scanner(System.in);

System.out.print("Input the number: ");

int n = sc.nextInt();

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

for (int j = n - i; j > 0; j--) {

System.out.print(" ");

}

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

System.out.print((char)('A' + j));

}

for (int j = i - 2; j >= 0; j--) {

System.out.print((char)('A' + j));

}

System.out.println();

}

for (int i = n - 1; i >= 1; i--) {

for (int j = n - i; j > 0; j--) {

System.out.print(" ");

}

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

System.out.print((char)('A' + j));

}

for (int j = i - 2; j >= 0; j--) {

System.out.print((char)('A' + j));

}

System.out.println();

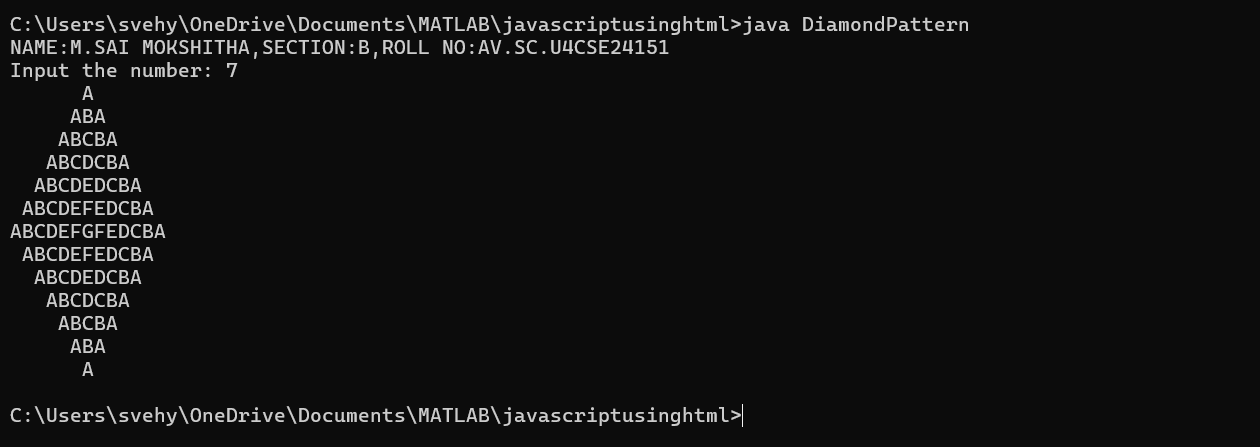
}

sc.close();

}

}

OUTPUT:



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Declaring the int type instead of char. 2. Sometimes printing ‘println’ instead of print. | 1. Declare char instead of int type. 2. Though it gives the output, it gives the wrong answer. |

**IMPORTANT POINTS:**

1. The program starts by taking an integer n as input, which represents the size of the rhombus.
2. The first for loop iterates from 1 to n. It prints the upper half of the rhombus.
3. Second and third loops handle the printing of the alphabet pattern. The second loop prints from A to the i-th letter, and the third loop prints the decreasing sequence of letters to completer the row.

**PROGRAM-11:**

**AIM:**Write a Java program to create a vehicle class hierarchy. The base class should be Vehicle, with subclasses Truck, Car and Motorcycle. Each subclass should have properties such as make, model, year, and fuel type. Implement methods for calculating fuel efficiency, distance travelled, and maximum speed.

import java.util.Scanner;

class Vehicle {

public String make;

public String model;

public int year;

public String fuelType;

public Vehicle(String make, String model, int year, String fuelType) {

this.make = make;

this.model = model;

this.year = year;

this.fuelType = fuelType;

}

public void displayInfo() {

System.out.println("Make: " + make + ", Model: " + model + ", Year: " + year + ", Fuel Type: " + fuelType);

}

}

class Truck extends Vehicle {

public Truck(String make, String model, int year, String fuelType) {

super(make, model, year, fuelType);

}

public void fuelEfficiency() {

System.out.println("Trucks generally have lower fuel efficiency.");

}

}

class Car extends Vehicle {

public Car(String make, String model, int year, String fuelType) {

super(make, model, year, fuelType);

}

public void distanceTravelled(int speed, int time) {

int distance = speed \* time;

System.out.println("Distance traveled: " + distance + " km");

}

}

class Motorcycle extends Vehicle {

public Motorcycle(String make, String model, int year, String fuelType) {

super(make, model, year, fuelType);

}

public void maximumSpeed(int maxSpeed) {

System.out.println("Maximum speed: " + maxSpeed + " km/h");

}

}

public class VehicleTest {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151");

Truck truck = new Truck("Ford", "F-150", 2022, "Diesel");

Car car = new Car("Toyota", "Corolla", 2023, "Petrol");

Motorcycle motorcycle = new Motorcycle("Yamaha", "R1", 2021, "Petrol");

truck.displayInfo();

truck.fuelEfficiency();

car.displayInfo();

car.distanceTravelled(80, 2);

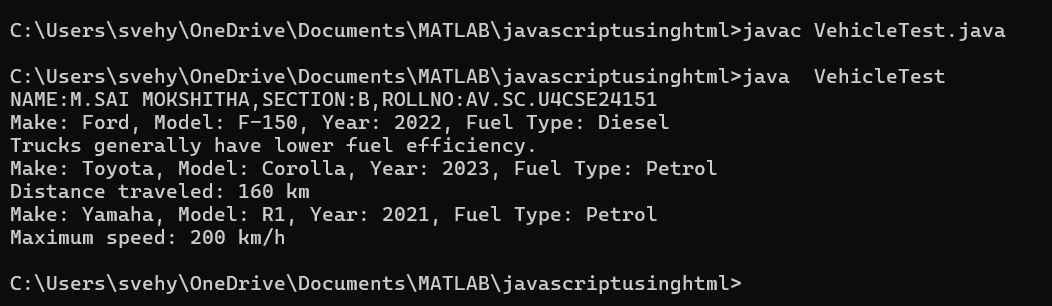
motorcycle.displayInfo();

motorcycle.maximumSpeed(200);

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Error while printing the print statement. 2. no values in the constructor. | 1. Give + sign when you want to print methods in the print statement. 2. Initialize the value of variables in the constructor. |

**IMPORTANT POINTS:**

1. Here, we declared the class abstract because there are some methods which do not have any return method.
2. Generally, abstract classes are used to define a common structure and behavior for a group of related classes.
3. We created a constructor in the super class and linked it with the subclasses so that the details of the subclasses can also be printed.
4. For that, we use the ‘super’ keyword to access the constructor of the super class.
5. We also override certain methods to access the information.
6. Here, even after initializing values in the constructor, we still declared gettermethods and give a return type for it to access them individually.

**PROGRAM-12:**

**AIM:**Write a Java program to create a class called Employee with methods called work () and getSalary(). Create a subclass called HRManager that overrides the work () method and adds a new method called addEmployee().

**CODE:**

class Employee {

String name;

int salary;

public Employee(String name, int salary) {

this.name = name;

this.salary = salary;

}

public String getname() {

return name;

}

public int getsalary() {

return salary;

}

public void work() {

System.out.println(name + " is working under ABC technologies.");

}

}

class HRmanager extends Employee {

public HRmanager(String name, int salary) {

super(name,salary);

}

public void work() {

System.out.println(name + " is working as a HRmanager at ABC technologies.");

}

public void addEmployee(String employee) {

System.out.println(name + " added " + employee + " to the Employee list.");

}

}

public class employeetest {

public static void main(String[]args) {

System.out.println("Name:M.sai mokshitha,Rollno:AV.SC.U4CSE2451,Section:B");

Employee e = new Employee("Ava Ayala", 2400000);

System.out.println(e.getname() + "'s basic salary is: " + e.getsalary());

e.work();

HRmanager hr = new HRmanager("Sam", 100000);

System.out.println(hr.getname() + "'s basic salary is: " + hr.getsalary());

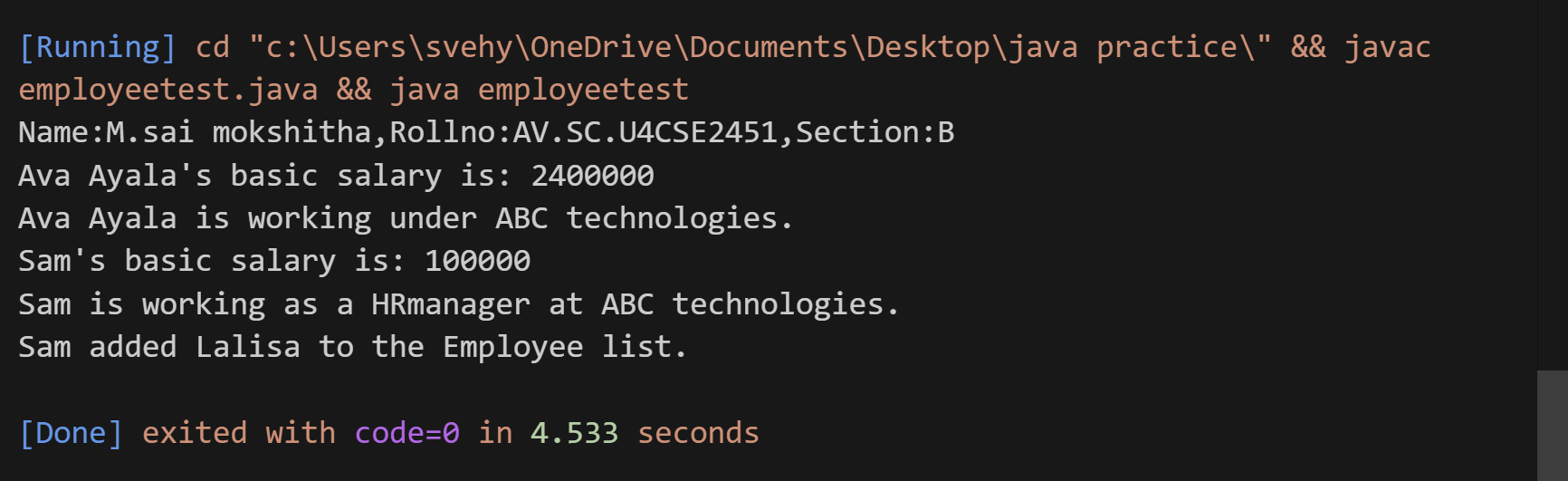
hr.work();

hr.addEmployee("Lalisa");

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Error while printing the print statement. 2. no values in the constructor. | 1. Give + sign when you want to print methods in the print statement. 2. Initialize the value of variables in the constructor. |

**IMPORTANT POINTS:**

1. Here, we declared two variables and then initialized them through a constructor.
2. Then created two methods to access them separately.
3. To create a new method in the subclass we declared a new argument within the parenthesis in the subclass.

**PROGRAM-13:**

**AIM:**Create a calculator using the operations including addition, subtraction, multiplication and division using multi-level inheritance and display the desired output.

**CODE:**

class Calculator {

protected double num1;

protected double num2;

public Calculator(double num1, double num2) {

this.num1 = num1;

this.num2 = num2;

}

public void displayNumbers() {

System.out.println("Numbers: " + num1 + " and " + num2);

}

}

class AdditionCalculator extends Calculator {

public AdditionCalculator(double num1, double num2) {

super(num1, num2);

}

public double add() {

return num1 + num2;

}

}

class SubtractionCalculator extends AdditionCalculator {

public SubtractionCalculator(double num1, double num2) {

super(num1, num2);

}

public double subtract() {

return num1 - num2;

}

}

class MultiplicationCalculator extends SubtractionCalculator {

public MultiplicationCalculator(double num1, double num2) {

super(num1, num2);

}

public double multiply() {

return num1 \* num2;

}

}

class DivisionCalculator extends MultiplicationCalculator {

public DivisionCalculator(double num1, double num2) {

super(num1, num2);

}

public double divide() {

if(num2 == 0) {

throw new ArithmeticException("Cannot divide by zero");

}

return num1 / num2;

}

}

public class CalculatorDemo {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151");

DivisionCalculator calc = new DivisionCalculator(10, 5);

calc.displayNumbers();

System.out.println("Addition: " + calc.add());

System.out.println("Subtraction: " + calc.subtract());

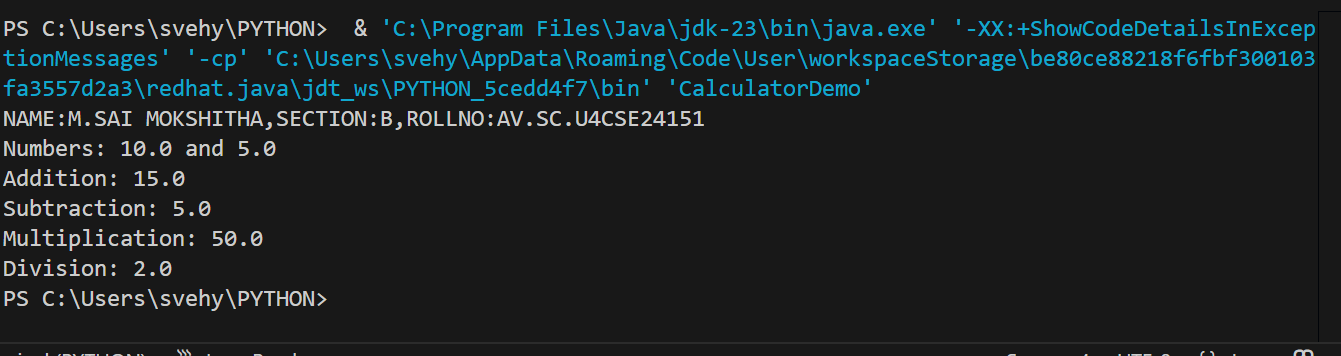
System.out.println("Multiplication: " + calc.multiply());

System.out.println("Division: " + calc.divide());

}

}

**OUTPUT**:



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Error, is showing zero. 2. Incorrect datatype entered. | 1. Declare the Nan after the double. 2. Declare int instead of double. |

**IMPORTANT POINTS:**

1. To get the inputs from the user we use import java.util.Scanner; this is a package.
2. Scanner class is used to get the user input.
3. in java.util.Scanner, the java.util is a package while Scanner is a class of the java.util package.
4. to import a whole package, end the sentence with an asterisk sign(\*).

Here, Nan stands for ‘Not a Number’ which mostly used while dividing two numbers as if it equals to zero, instead of printing an error it just passes declaring it as not a number

**PROGRAM-14:**

**AIM:**Consider a software system for a company that manages its employees. The company categorizes its employees into two primary types: RegularEmployee and Manager. Both types of employees share common attributes such as name and employee ID, but managers have attributes such as a bonus. You are tasked with designing the Java classes for this scenario and add up the salary for each type.

**CODE:**

class Software {

public String name;

public int employeeId;

public Software(String name, int employeeId) {

this.name = name;

this.employeeId = employeeId;

}

public String getName() {

return name;

}

public int getEmployeeId() {

return employeeId;

}

}

class RegularEmployee extends Software {

public RegularEmployee(String name, int employeeId) {

super(name, employeeId);

}

}

class Manager extends Software {

public Manager(String name, int employeeId) {

super(name, employeeId);

}

public void giveBonus(int bonusAmount) {

System.out.println("A manager gets bonus up to: " + bonusAmount);

}

}

public class SoftwareTest {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151");

Software s = new Software("Jennie", 101526);

RegularEmployee re = new RegularEmployee("Lalisa", 145926);

Manager m = new Manager("Rosie", 246789);

System.out.println("A software company consists of a Boss named " + s.getName());

System.out.println(re.getName() + " is a regular employee.");

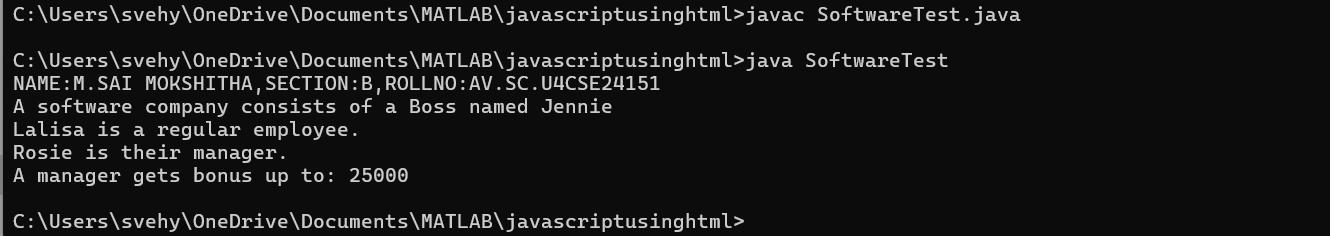
System.out.println(m.getName() + " is their manager.");

m.giveBonus(25000);

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Subclass doesn’t have a super() constructor. | 1. Every subclass must implement a super constructor. |

**IMPORTANT POINTS:**

1. Here, we declared two variables and then initialized them through a constructor.
2. Then created two methods to access them separately.
3. To create a new method in the subclass we declared a new argument within the parenthesis in the subclass.

**PROGRAM-15:**

**AIM:**A superclass named “Shapes” has a method called “area()”. Subclasses of “Shapes” can be “Triangle”, “circle”, “Rectangle”, etc. Each subclass has its own way of calculating area. Using base class as Shapes with subclasses triangle, circle and rectangle, use overriding polymorphism and find the area for each shape.

**CODE:**

class Shapes {

public double area() {

return 0;

}

}

class Triangle extends Shapes {

private double base;

private double height;

public Triangle(double base, double height) {

this.base = base;

this.height = height;

}

@Override

public double area() {

return 0.5 \* base \* height;

}

}

class Circle extends Shapes {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double area() {

return Math.PI \* radius \* radius;

}

}

class Rectangle extends Shapes {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

@Override

public double area() {

return length \* width;

}

}

public class ShapeArea {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151);

Shapes triangle = new Triangle(2, 5);

Shapes circle = new Circle(4);

Shapes rectangle = new Rectangle(6, 9);

System.out.println("Area of Triangle: " + triangle.area());

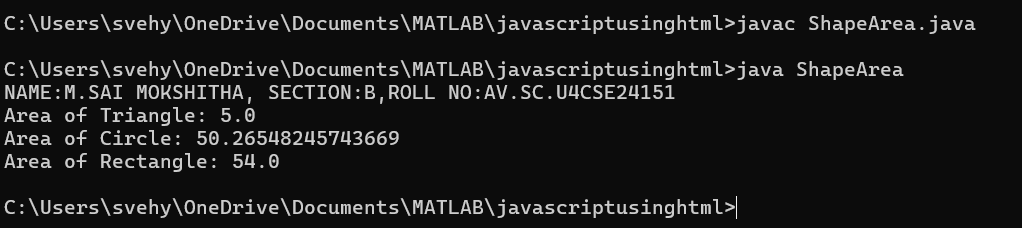
System.out.println("Area of Circle: " + circle.area());

System.out.println("Area of Rectangle: " + rectangle.area());

}

}

OUTPUT:



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Error while printing the variables. 2. Incorrect declaration of integer. | 1. Give the plus sign while printing. 2. Give input.nextInt(), where I should be capital. |

**IMPORTANT POINTS:**

1. Here we used the abstract to declare an abstract class.
2. Abstract classes and methods help us to declare the methods without declaring the return type in them.

To get the values, we declared a constructor for each subclass and initialized values for them

**PROGRAM-16:**

**AIM:**creating one superclass Animal and three subclasses, Herbivores, Carnivores, and Omnivores. Subclasses extend the superclass and override its eat() method. Returning the method for the required type of animals.

CODE:

class Animal{

void eat(){

System.out.println("iam king");

}

}

class Herbivores extends Animal{

void eat(){

System.out.println("cow and deer are herbivores");

}

}

class carnivores extends Animal{

void eat(){

System.out.println("bears and pigs are carnivores");

}

}

class omnivores extends Animal{

void eat(){

System.out.println("human and rats are omnivores");

}

}

class TestAnimal{

public static void main(String[] args){

System.out.println("Name:M.sai mokshitha,Rollno:AV.SC.U4CSE2451,Section:B");

Animal a=new Animal();

a.eat();

Animal b=new Herbivores();

b.eat();

Animal c=new carnivores();

c.eat();

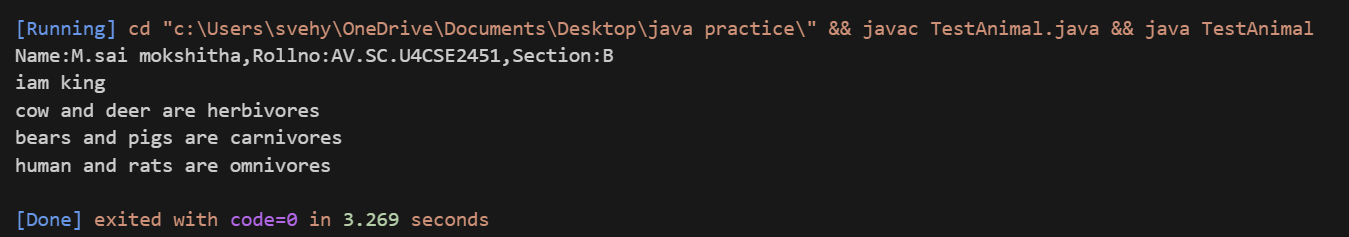
Animal d=new omnivores();

d.eat();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. No return method after declaring a variable. 2. Void doesn’t support return method. | 1. After declaring a variable in the method,we must return that variable.   2.Void itself says that it doesn’t return any method, it just prints the statement. |

**IMPORTANT POINTS:**

1. Here, we initialized a variable inside a method in the form of a parameter.
2. At last, after creating the object, while calling the method, if the parameter is passed inside the method then It prints the value.

**PROGRAM-17:**

**AIM:**Write a Java program to create an abstract class Animal with an abstract method called sound(). Create subclasses Lion and Tiger that extend the Animal class and implement the sound() method to make a specific sound for each animal.

**CODE:**

abstract class Animal {

abstract void sound();

}

class Lion extends Animal {

@Override

void sound() {

System.out.println("Roar");

}

}

class Tiger extends Animal {

@Override

void sound() {

System.out.println("Tiger growls");

}

}

class Testsound {

public static void main(String[] args) {

System.out.println("Name: M.sai mokshitha, Rollno: AV.SC.U4CSE2451, Section: B");

Lion l = new Lion();

Tiger t = new Tiger();

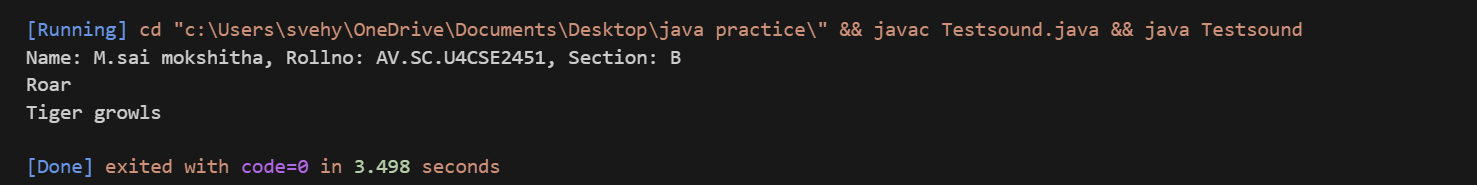
l.sound();

t.sound();

}

}

OUTPUT:



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Error while printing the variables. 2. Incorrect declaration of integer. | 1. Give the plus sign while printing. 2. Give input.nextInt(), where I should be capital. |

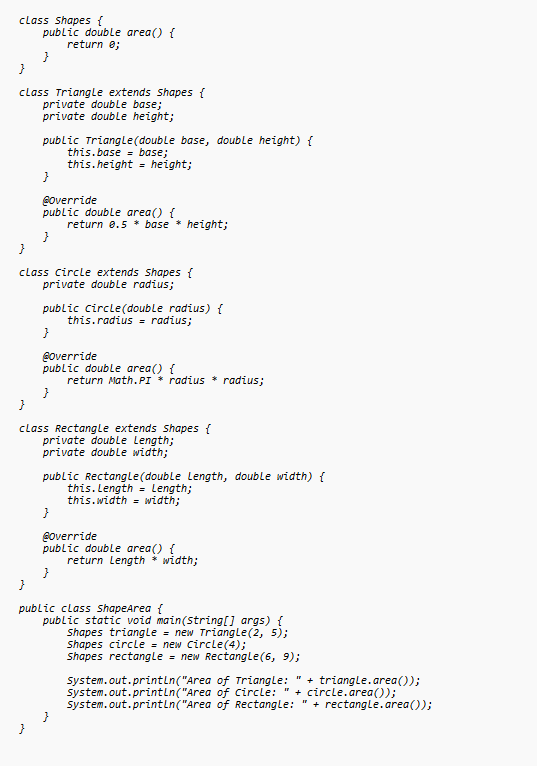
**IMPORTANT POINTS:**

1. We override the methods in the superclass.
2. Here we are using the heirarchial inheritance.

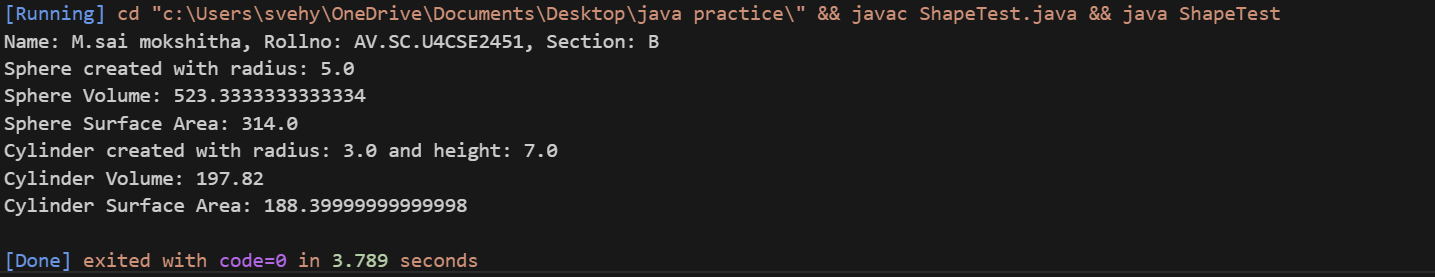
**PROGRAM-18:**

**AIM:**Write a Java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extend the Shape3D class and implement the respective methods to calculate the volume and surface area of each shape.

**CODE:**



**OUTPUT:**



**PROGRAM-19:**

**AIM:**What will be the output of the following program?

**interface** A

{

**void** Method ();

}

**class** B

{

**public** **void** Method ()

    {

        System. out.println ("My Method");

    }

}

**class** C **extends** B **implements** A

{

}

**class** Main

{

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

    {

        A a = **new** C ();

        a. Method ();

    }

}

interface A {

void method();

}

class B {

public void method() {

System.out.println("My Method");

}

}

class C extends B implements A {

}

class Main {

public static void main(String[] args) {

System.out.println("NAME: M.SAI MOKSHITHA, SECTION: B, ROLLNO: AV.SC.U4CSE24151");

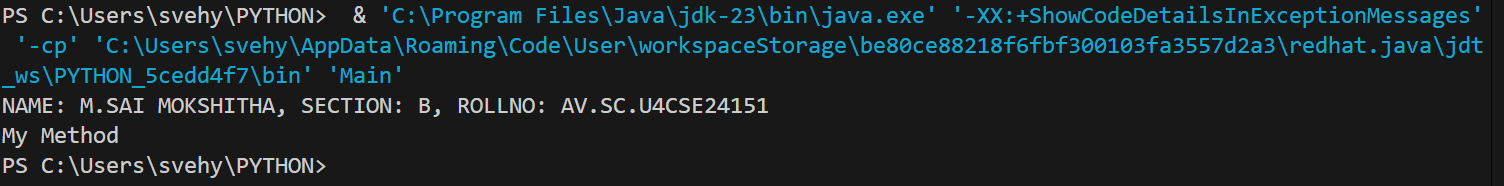
A a = new C();

a.method();

}

}

Output:



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Giving class name not as main 2. Not giving input I as capital | 1. Give the plus sign while printing. 2. Give input.nextInt(), where I should be capital. |

**IMPORTANT POINTS:**

1. Here we used the abstract to declare an abstract class.
2. Abstract classes and methods help us to declare the methods without declaring the return type in them.
3. To get the values, we declared a constructor for each subclass and initialized values for them.

**PROGRAM-19:**

**AIM: Does below code compile successfully? If not, why?**

**interface** A

{

**int** i = 111;

}

**class** B **implements** A

{

**void** methodB()

    {

        i = 222;

    }

}

**PROGRAM-20:**

**AIM:**Write a Java program to create an interface Shape with the getPerimeter() method. Create three classes Rectangle, Circle, and Triangle that implement the Shape interface. Implement the getPerimeter() method for each of the three classes.

Code:

abstract class Shape3D {

abstract double volume();

abstract double surfaceArea();

}

class Sphere extends Shape3D {

double radius;

Sphere(double radius) {

this.radius = radius;

System.out.println("Sphere created with radius: " + radius);

}

@Override

double volume() {

return (4.0/3.0) \* 3.14 \* radius \* radius \* radius;

}

@Override

double surfaceArea() {

return 4 \* 3.14 \* radius \* radius;

}

}

class Cylinder extends Shape3D {

double radius;

double height;

Cylinder(double radius, double height) {

this.radius = radius;

this.height = height;

System.out.println("Cylinder created with radius: " + radius + " and height: " + height);

}

@Override

double volume() {

return 3.14 \* radius \* radius \* height;

}

@Override

double surfaceArea() {

return (2 \* 3.14 \* radius \* height) + (2 \* 3.14 \* radius \* radius);

}

}

class ShapeTest {

public static void main(String[] args) {

System.out.println("Name: M.sai mokshitha, Rollno: AV.SC.U4CSE2451, Section: B");

Sphere s1 = new Sphere(5.0);

System.out.println("Sphere Volume: " + s1.volume());

System.out.println("Sphere Surface Area: " + s1.surfaceArea());

Cylinder c1 = new Cylinder(3.0, 7.0);

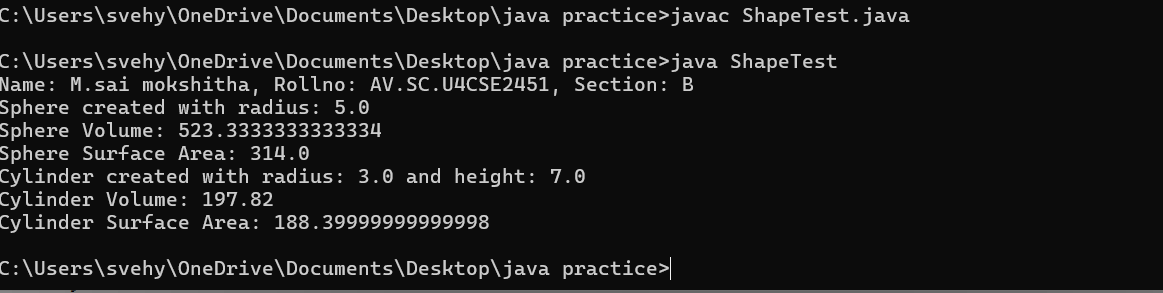
System.out.println("Cylinder Volume: " + c1.volume());

System.out.println("Cylinder Surface Area: " + c1.surfaceArea());

}

}

OUTPUT:



**PROGRAM-21:**

**AIM:**Write a Java program that creates a class hierarchy for employees of a company. The base class should be Employee, with subclasses Manager, Developer, and Programmer. Each subclass should have properties such as name, address, salary, and job title. Implement methods for calculating bonuses, generating performance reports, and managing projects.

**Code**:

public class Employee1 {

public String name;

public String address;

public int salary;

public String jobtitle;

public Employee1(String name, String address, int salary, String jobtitle) {

this.name = name;

this.address = address;

this.salary = salary;

this.jobtitle = jobtitle;

}

public String name() {

return name;

}

public String address() {

return address;

}

public int salary() {

return salary;

}

public String jobtitle() {

return jobtitle; // Fixed: was calling jobtitle() recursively

}

public double bonus() {

return (salary \* 0.10);

}

public void performancereport() {

System.out.println("Employee name: " + name);

System.out.println("Salary: " + salary);

System.out.println("Job Title: " + jobtitle);

}

}

class Manager extends Employee1 {

public int team;

public Manager(String name, String address, int salary, String jobtitle, int team) {

super(name, address, salary, jobtitle);

this.team = team;

}

@Override

public double bonus() {

return (salary \* 0.15);

}

public void manageproject() {

System.out.println(name + " is managing a project with " + team + " members.");

}

}

class Developer extends Employee1 {

public String develop;

public Developer(String name, String address, int salary, String jobtitle, String develop) {

super(name, address, salary, jobtitle);

this.develop = develop;

}

@Override

public double bonus() {

return (salary \* 0.13);

}

public void developcode() {

System.out.println(name + " is developing a website using " + develop);

}

}

class Programmer extends Employee1 {

public String program;

public Programmer(String name, String address, int salary, String jobtitle, String program) {

super(name, address, salary, jobtitle);

this.program = program;

}

@Override

public double bonus() {

return (salary \* 0.12);

}

public void codeproject() {

System.out.println(name + " is working on a project using " + program);

}

}

class TestEmployee {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA, SECTION:B,ROLL NO:AV.SC.U4CSE24151");

Manager m = new Manager("Lalisa", "St.avenue", 2500000, "Healthcare", 5);

m.performancereport();

System.out.println("Bonus given to Lalisa is: " + m.bonus());

m.manageproject();

Developer d = new Developer("Jennie", "westline", 2300000, "IT", "Javascript");

d.performancereport();

System.out.println("Bonus given to Jennie is: " + d.bonus());

d.developcode();

Programmer p = new Programmer("Ariana", "southeast", 3000000, "Cybersecurity", "Java");

p.performancereport();

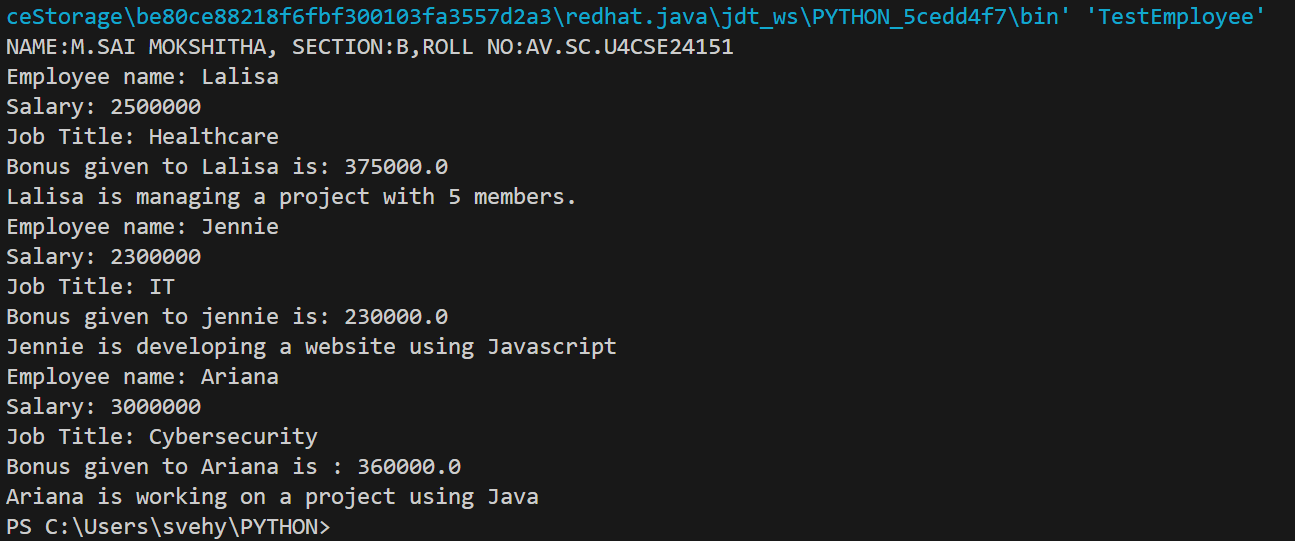
System.out.println("Bonus given to Ariana is: " + p.bonus());

p.codeproject();

}

}

OUTPUT:



IMPORTANT POINTS:

1. Here, we initialized values through a constructor, and then separately called them with methods.
2. We created a method in superclass which we didn’t override i.e the performance report.

**PROGRAM-22:**

**AIM:**Write a Java program to create a class called Student with private instance variables student\_id, student\_name, and grades. Provide public getter and setter methods to access and modify the student\_id and student\_name variables. However, provide a method called addGrade() that allows adding a grade to the grades variable while performing additional validation.

**CODE:**

import java.util.ArrayList;

import java.util.List;

public class Student {

private int studentid;

private String name;

private List<Integer> grades;

public Student(int studentid, String name) {

this.studentid = studentid;

this.name = name;

this.grades = new ArrayList<>();

}

public int getstudentid() {

return studentid;

}

public void setstudentid(int studentid) {

this.studentid = studentid;

}

public String getname() {

return name;

}

public void setname(String name) {

this.name = name;

}

public void addgrade(int grade) {

if (grade>=0 && grade<=100) {

grades.add(grade);

System.out.println("grade added: " + grade);

} else {

System.out.println("Invalid grade entered.");

}

}

public void studentinfo() {

System.out.println("Student Name: " + name);

System.out.println("Student ID: " + studentid);

System.out.println("Grade: " + grades);

}

public static void main(String[]args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151");

Student s = new Student(241325, "Lalisa" );

s.addgrade(84);

s.addgrade(98);

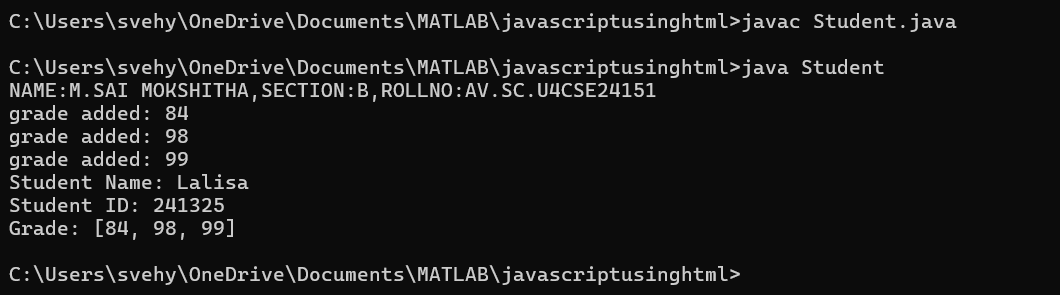
s.addgrade(99);

s.studentinfo();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Arraylist cannot be resolved to a type. 2. Cannot infer type arguments for arraylist(). | 1. While declaring a list we must declare Arraylist also. 2. Rectify the case sensitive letters. |

**IMPORTANT POINTS:**

1. As the programming languages are case sensitive, we must make sure that the List part is defined correctly.
2. First, we defined a constructor for the private variables and initialized them, then, we access the private variables using getter and setter methods.
3. In the addgrade method we first check if the integer is suitable for given conditions and then add them to the list.

**PROGRAM-23:**

**AIM:**Write a Java program to create a base class BankAccount with methods deposit() and withdraw(). Create two subclasses SavingsAccount and CheckingAccount. Override the withdraw() method in each subclass to impose different withdrawal limits and fees.

**CODE:**

class BankAccount {

void deposit() {

System.out.println("Deposit in BankAccount");

}

void withdraw() {

System.out.println("Withdraw from BankAccount");

}

}

class SavingsAccount extends BankAccount {

@Override

void withdraw() {

System.out.println("Withdraw from SavingsAccount (limit:₹5000000)");

}

}

class CheckingAccount extends BankAccount {

@Override

void withdraw() {

System.out.println("Withdraw from CheckingAccount (fee:₹10000)");

}

}

class TestAccount {

public static void main(String[] args) {

System.out.println("NAME:M.SAIMOKSHITHA, SECTION:B,ROLL NO:AV.SC.U4CSE24151");

SavingsAccount s1 = new SavingsAccount();

s1.deposit();

s1.withdraw();

CheckingAccount c1 = new CheckingAccount();

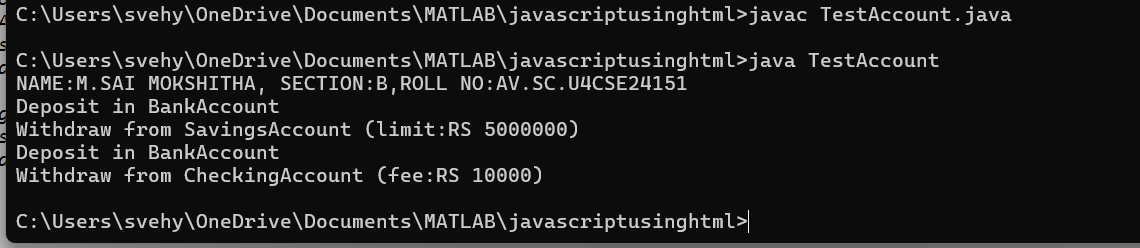
c1.deposit();

c1.withdraw();

}

}

**OUTPUT:**



**PROGRAM-24:**

**AIM:**Write a Java program to create an abstract class Bird with abstract methods fly() and makeSound(). Create subclasses Eagle and Hawk that extend the Bird class and implement the respective methods to describe how each bird flies and makes a sound.

**CODE:**

abstract class Bird {

abstract void fly();

abstract void makesound();

}

class Eagle extends Bird {

void fly() {

System.out.println("Eagle flies high in the sky");

}

void makesound() {

System.out.println("Eagle screeches");

}

}

class Hawk extends Bird {

void fly() {

System.out.println("Hawk flies high in the sky");

}

void makesound() {

System.out.println("Hawk screeches");

}

}

class Testfly {

public static void main(String args[]) {

Hawk h1 = new Hawk();

Eagle e1 = new Eagle();

h1.fly();

h1.makesound();

e1.fly();

e1.makesound();

}

}

**OUTPUT**:



**ERROR TABLE**:

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Declaring an abstract class instead of interface class. 2. Not declaring public in each class. | 1. Declare an interface class instead of abstract class. 2. Declare public infront of each class. |

**IMPORTANT POINTS:**

1. We declared an interface class without declaring it an abstract because to keep all the abstract methods in one place.
2. Then we override the methods using sub classes.
3. While calling the methods first, declare the super class and then the sub class name as shown in the above code.

**PROGRAM-25:**

**AIM:**Write a Java program to create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play() method to play the respective sports.

**CODE:**

interface Playable {

void play();

}

class Football implements Playable {

@Override

public void play() {

System.out.println("Playing Football: Kicking the ball towards the goal");

}

}

class Volleyball implements Playable {

@Override

public void play() {

System.out.println("Playing Volleyball: Bumping, setting, and spiking the ball");

}

}

class Basketball implements Playable {

@Override

public void play() {

System.out.println("Playing Basketball: Dribbling and shooting the ball");

}

}

public class TestSports {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLL NO:AV.SC.U4CSE24151");

Playable football = new Football();

Playable volleyball = new Volleyball();

Playable basketball = new Basketball();

football.play();

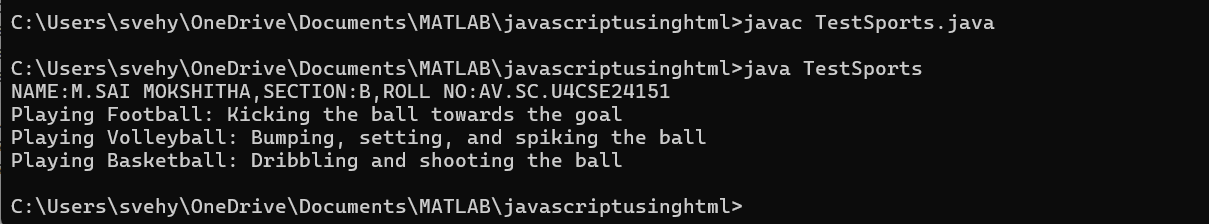
volleyball.play();

basketball.play();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Declaring an abstract class instead of interface class. 2. Not declaring public in each class. | 1. Declare an interface class instead of abstract class. 2. Declare public infront of each class. |

**IMPORTANT POINTS:**

1. The playable interface abstracts the play() method, ensuring different classes implement it differently
2. The play() method behaves differently based on the object type football, volleyball, basketball.

Each class encapsulates its own implementation of how the sport is played, hiding the details from the user

**PROGRAM-26:**

**AIM:**Write a Java programming to create a banking system with three classes - Bank, Account, SavingsAccount, and CurrentAccount. The bank should have a list of accounts and methods for adding them. Accounts should be an interface with methods to deposit, withdraw, calculate interest, and view balances. SavingsAccount and CurrentAccount should implement the **Account** **interface** and have their own unique methods.

**Code:**

import java.util.ArrayList;

interface Account {

void deposit(double amount);

void withdraw(double amount);

double getBalance();

}

class Bank {

private ArrayList<Account> accounts = new ArrayList<>();

public void addAccount(Account account) {

accounts.add(account);

}

}

class SavingsAccount implements Account {

private double balance;

private double interestRate;

public SavingsAccount(double initialBalance, double rate) {

balance = initialBalance;

interestRate = rate;

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

}

}

public double getBalance() {

return balance;

}

public void addInterest() {

balance += balance \* interestRate / 100;

}

}

class CurrentAccount implements Account {

private double balance;

private double overdraftLimit;

public CurrentAccount(double initialBalance, double limit) {

balance = initialBalance;

overdraftLimit = limit;

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

if (amount <= balance + overdraftLimit) {

balance -= amount;

}

}

public double getBalance() {

return balance;

}

}

public class SimpleBankingSystem {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLLNO:AV.SC.U4CSE24151");

Bank bank = new Bank();

SavingsAccount savings = new SavingsAccount(1000, 2.5);

CurrentAccount current = new CurrentAccount(2000, 500);

bank.addAccount(savings);

bank.addAccount(current);

savings.deposit(500);

savings.withdraw(200);

savings.addInterest();

current.deposit(1000);

current.withdraw(2500);

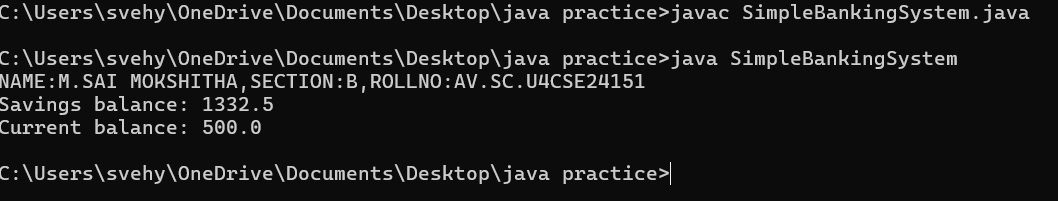
System.out.println("Savings balance: " + savings.getBalance());

System.out.println("Current balance: " + current.getBalance());

}

}

OUTPUT:



**PROGRAM-27:**

**AIM:**.How would you demonstrate the initialization and usage of arrays in Java? Discuss the various methods of declaring, initializing, and populating arrays. Using the arrays concept write a java program to initialize a matrix, addition of two matrices, multiplication of two matrices and display the output.

CODE:

import java.util.Scanner;

public class MatrixOperations {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLL NO:AV.SC.U4CSE24151");

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of rows: ");

int rows = scanner.nextInt();

System.out.print("Enter number of columns: ");

int cols = scanner.nextInt();

int[][] matrix1 = new int[rows][cols];

int[][] matrix2 = new int[rows][cols];

System.out.println("Enter elements of first matrix:");

readMatrix(matrix1, scanner);

System.out.println("Enter elements of second matrix:");

readMatrix(matrix2, scanner);

System.out.println("\nMatrix 1:");

printMatrix(matrix1);

System.out.println("\nMatrix 2:");

printMatrix(matrix2);

System.out.println("\nMatrix Addition:");

int[][] sum = addMatrices(matrix1, matrix2);

printMatrix(sum);

System.out.println("\nMatrix Multiplication:");

if (matrix1[0].length == matrix2.length) {

int[][] product = multiplyMatrices(matrix1, matrix2);

printMatrix(product);

} else {

System.out.println("Cannot multiply - column count of first matrix must match row count of second matrix");

}

}

private static void readMatrix(int[][] matrix, Scanner scanner) {

for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix[i].length; j++) {

matrix[i][j] = scanner.nextInt();

}

}

}

private static void printMatrix(int[][] matrix) {

for (int[] row : matrix) {

for (int element : row) {

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

}

System.out.println();

}

}

private static int[][] addMatrices(int[][] a, int[][] b) {

int[][] result = new int[a.length][a[0].length];

for (int i = 0; i < a.length; i++) {

for (int j = 0; j < a[i].length; j++) {

result[i][j] = a[i][j] + b[i][j];

}

}

return result;

}

private static int[][] multiplyMatrices(int[][] a, int[][] b) {

int[][] result = new int[a.length][b[0].length];

for (int i = 0; i < a.length; i++) {

for (int j = 0; j < b[0].length; j++) {

for (int k = 0; k < a[0].length; k++) {

result[i][j] += a[i][k] \* b[k][j];

}

}

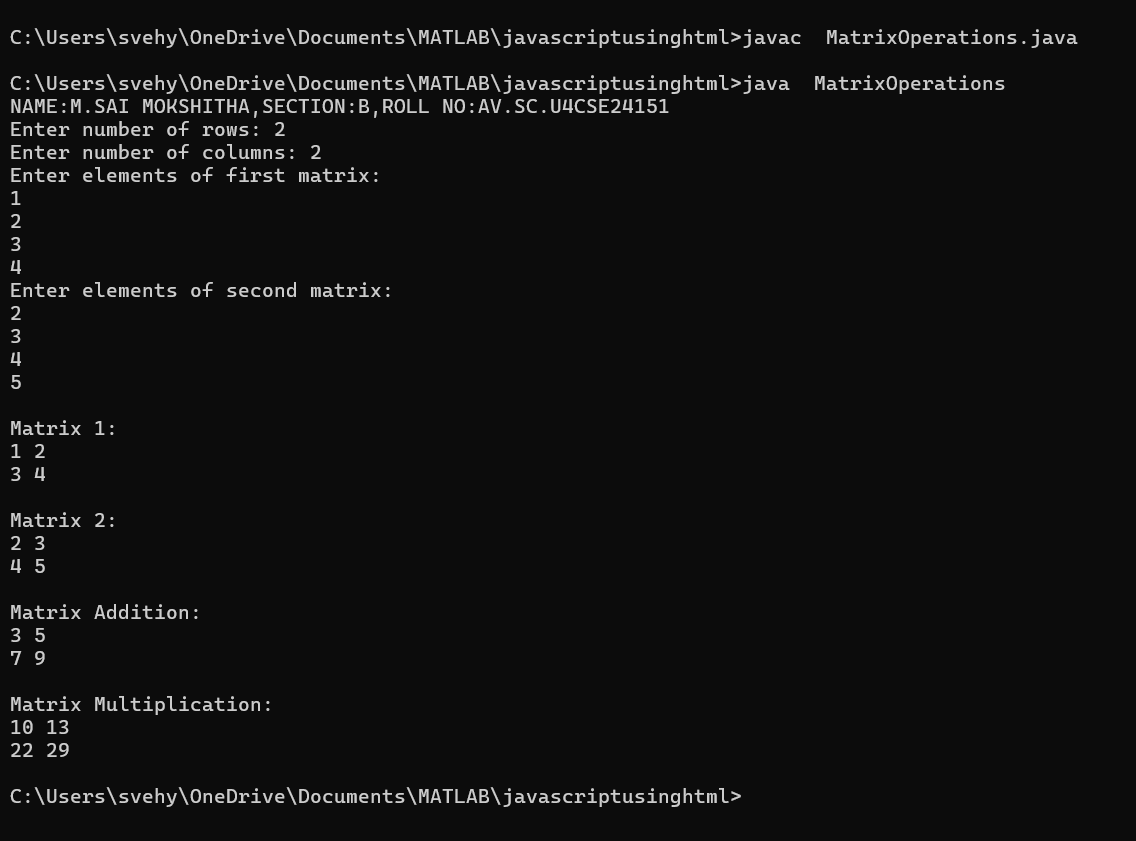
}

return result;

}

}

OUTPUT:



ERROR TABLE:

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Not closing the input. 2. Not declaring the arrays correctly. | 1. Close the input. 2. Though it gives the output, it doesn’t give give what we like. |

IMPORTANT POINTS:

1. Here the two 2D arrays mat1 and mat2 are initialized based on the user input for rows and columns
2. The user then enters the elements for each matrix.
3. The addition of two matrices is done by adding elements from mat1 and mat2 and storing the result in smat.
4. For matrix multiplication the program checks if the matrices are square (i.e the number of rows equals the number of columns), as matrix multiplication requires the number of columns in the first matrix to match the number.
5. If the condition is met, the product of the two matrices is calculated.

**PROGRAM-28:**

**AIM:**a. Discuss the difference between the Interfaces vs. Abstract Classes in detail.

|  |  |
| --- | --- |
| **Interface** | **Abstract** |
| An interface is a contract that defines a set of methods without providing any implementation for them.  A class that implements an interface must provide the actual implementation for all methods declared in the interface.  Interfaces are used to represent abilities or capabilities that can be shared across multiple classes, regardless of the class hierarchy.  All the methods in an interface do not have a method body.  The implementing class must provide the implementation of these methods.  Multiple interfaces can be implemented by a single class, each interface can define its own contract without conflict.  An interface cannot have constructors because interfaces cannot be instantiated. They only provide method declarations.  Interfaces cannot have instance fields(variables), but they can have constant fields.  Interfaces are are ideal for defining roles that can be implemented by any class.  e.g comparable, runnable, serializable etc….  You can define common behaviour across unrelated classes. | An abstract class is a class that cannot be instantiated directly.It may contain both abstract methods(without implementation) and concrete methods(with implementation)  An abstract class is often used to model shared properties and behaviour between related classes.  Abstract classes are used to model shared properties and behaviour between related classes.  Abstract classes can have both abstract methods and concrete methods.  Subclass of an abstract class must implement the abstract methods, but it can inherit the concrete methods from the abstract class.  A class can inherit from only one abstract class, meaning that java does not support multiple inheritance through abstract classes.  Abstract classes can have constructors. These constructors are used by the subclasses when creating instances.  Abstract classes can have instance fields(non-static fields) that can be accessed or modified by both the abstract class and its subclasses.  An abstract class can serve as a blueprint for other classes and allow some default behaviour.  You can use this when you want to share code(implementation) among related classes. |

b. Discuss the difference between the Overriding vs. Overloading in detail.

|  |  |
| --- | --- |
| **Overriding** | **Overloading** |
| It provides a specific implementation of a method already defined in the superclass.  Must involve inheritance(Subclass and superclass).  Overriding contains same method and same parameters as the parent class.  Return type must be the same as the parent class method.  It is achieved at run time, so it’s called run-time polymorphism.  It cannot have a more restrictive access modifier than the parent class method.  Here constructors cannot be overridden.  Overriding uses @override annotation. | It defines multiple methods with the same name but different parameter lists.  Does not require inheritance. Methods can exist in the same class.  Overloading conatins a method name, but different number or type of parameters.  It’s return type can be different, but it is not considered for overloading.  It is achieved at compile time, so it’s called compile-time polymorphism.  It can have different access modifiers.  Here constructors can be overloaded by varying parameters.  Overloading does not use any annotation. |

**PROGRAM-29:**

**AIM:**.(Triangle class) Design a new Triangle class that extends the abstract GeometricObject class. Draw the UML diagram for the classes Triangle and GeometricObject and then implement the Triangle class. Write a test program that prompts the user to enter three sides of the triangle, a color, and a Boolean value to indicate whether the triangle is filled. The program should create a Triangle object with these sides and set the color and filled properties using the input. The program should display the area, perimeter, color, and true or false to indicate whether it is filled or not.

**PROGRAM-30:**

**AIM:**.Rewrite the PrintCalendar class in Listing 6.12 to display a calendar for a specified month using the Calendar and GregorianCalendar classes. Your program receives the month and year from the command line. For

example:

java Exercise13\_04 5 2016

This displays the calendar shown in Figure.