****

**Object Oriented Programming (23CSE111)**

**Assignment**

|  |  |
| --- | --- |
| **Submitted by** | |
| **Name** | **N.ABHISHEK** |
| **Roll No** | **AV.SC.U4CSE24232** |
| **Year/Sem/Section** | **1st Year/ 2nd SEM/CSE-C** |
| **Date of Submission** |  |
| **Submitted to** | |
| **Name** | **Dr. B Raj Kumar** |
| **Department** | **CSE** |
| **Designation** | **Asst. Professor** |

|  |  |
| --- | --- |
| **Marks** |  |

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Q.No** | **PAGE No.** |
| **1.** | **1** | **4** |
| **2.** | **2** | **6** |
| **3.** | **3** | **8** |
| **4.** | **4** | **10** |
| **5.** | **5** | **13** |
| **6.** | **6** | **15** |
| **7.** | **7** | **16** |
| **8.** | **8** | **19** |
| **9.** | **9** | **21** |
| **10.** | **10** | **23** |
| **11.** | **11** | **25** |
| **12.** | **12** | **28** |
| **13.** | **13** | **30** |
| **14.** | **14** | **32** |
| **15.** | **15** | **35** |
| **16.** | **16** | **37** |
| **17.** | **17** | **39** |
| **18.** | **18** | **41** |
| **19.** | **19** | **43** |
| **20.** | **20** | **45** |
| **21.** | **21** | **48** |
| **22.** | **22** | **52** |
| **23.** | **23** | **54** |
| **24.** | **24** | **57** |
| **25.** | **25** | **59** |
| **26.** | **26** | **61** |
| **27.** | **27** | **65** |
| **28.** | **28** | **68** |
| **29.** | **29** | **71** |
| **30.** | **30** | **74** |

**1) Write a java program with class named “book”. The class should contain various attributes such as “title, author, yearofpublication”. It should also contain a “constructor” with parameters which initializes “title”, ”author”, and “yearofpublication”.Create a method which displays the details of the book i.e. “author, title, yearofpublication”.(Display the details of two books i.e. create 2 objects** **and display their details).  
  
CODE:**

public class book {

    String title;

    String author;

    int yearOfPublication;

    book(String title,String author,int yearOfPublication){

    this.title = title;

    this.author = author;

    this.yearOfPublication = yearOfPublication;

    }

    // error : wrote static in the method

    public void detailsBook(){

        System.out.println("the title of book is " + this.title);

        System.out.println("the author of the book is " + this.author);

        System.out.println("the year of publication of the book is " + this.yearOfPublication);

    }

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        book b1 = new book("topper","manoj",2025);

        b1.detailsBook();

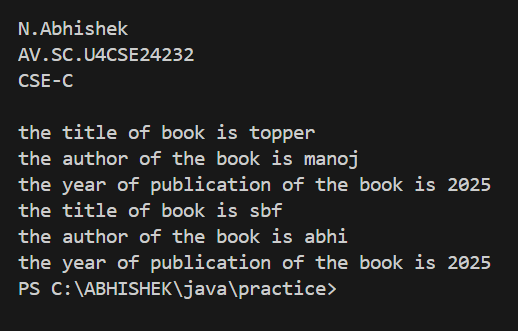
        book b2 = new book("sbf","abhi",2025);

        b2.detailsBook();

        //did not put double quotes for strings

    }

}

**OUTPUT:  
  
  
  
Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Constructor Error | Constructor parameters not matching class attributes |
| 2 | Static Method Error | Removed static from detailsBook() to allow object access |
| 3 | String Literal Error | Added double quotes for String values in object creation |

**IMPORTANT\_POINTS**

* Class Structure: Define a Book class with attributes title (String), author (String), and yearOfPublication (int) to store book details.
* 2)Constructor: Implement a parameterized constructor to initialize title, author, and yearOfPublication when a Book object is created.
* 3)Display Method: Create a method (e.g., displayDetails()) to print the book’s details in a formatted manner, including author, title, and year of publication.
* 4)Object Creation: Instantiate two Book objects with different values and call the display method for each to demonstrate functionality.

**2) 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;

    static double PI =3.14159 ;

    public MyClass() {

        count++;

    }

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        MyClass cl1 = new MyClass();

        MyClass cl2 = new MyClass();

        MyClass cl3 = new MyClass();

        System.out.println("the number of objects created are :" +count);

        //error did not put + symbol

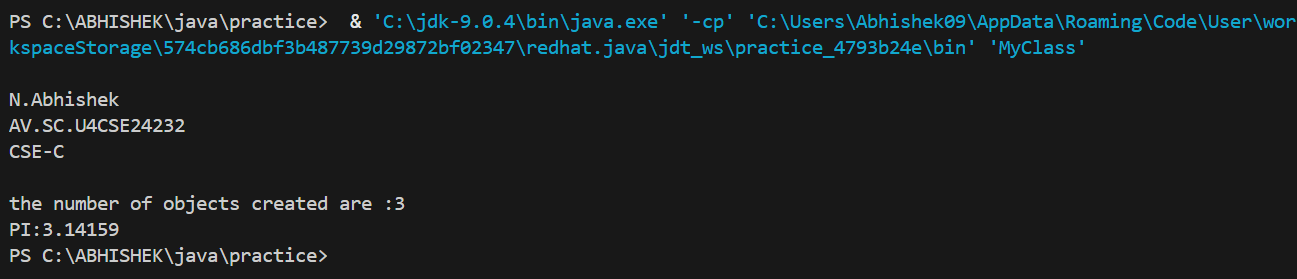
        // did not mention static for PI

        System.out.println("PI:"+PI);

}

}

}**OUTPUT:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Static Declaration Missing | Declared count as static to maintain object count |
| 2 | Final Constant Error | Declared PI as final to mark it as a constant |
| 3 | String Concatenation Error | Added + for string concatenation in print statement |

**IMPORTANT POINTS:**

* Static Variable: Declare a static int variable count initialized to 0 to track the number of MyClass objects created.
* Constant Variable: Define a final double variable PI initialized to 3.14159, ensuring it cannot be modified.
* 3)Constructor Logic: Implement a constructor that increments the count variable each time a new MyClass object is instantiated.

**3) 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().   
• Attempt to access the private variable privateVariable and call the private method privateMethod() in the Main class.   
• Note: attempting to do so will result in a compilation error.**

**CODE:**

class visibilityExample {

    public int publicVariable = 10;

    private int privateVariable = 20;

public void publicMethod(){

    System.out.println("this is a public method");

}

private void privateMethod(){

    System.out.println("this is a private method");

}

}

public class Main{

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        visibilityExample var1 = new visibilityExample();

        System.out.println(var1.publicVariable);

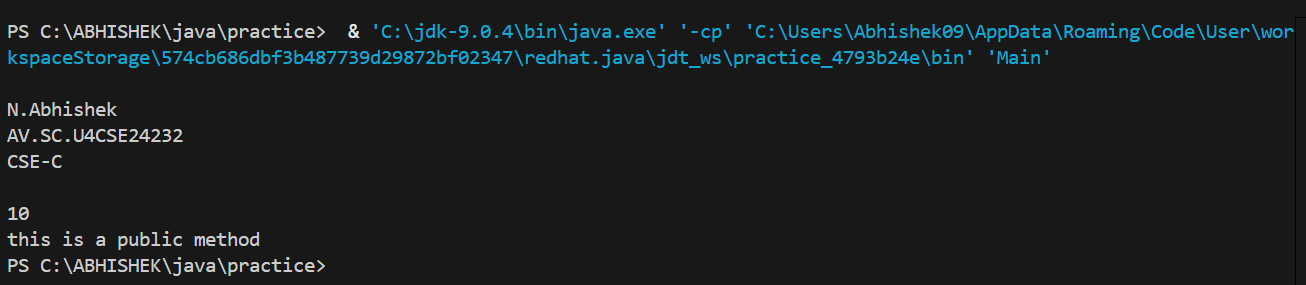
        var1.publicMethod();

        //System.out.println(var1.privateVariable);

        //var1.privateMethod();

    }

}

**OUTPUT:  
  
**

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Access Modifier Error | Tried to access private variable/method outside class |
| 2 | Encapsulation Violation | Access to private members is not allowed directly |
| 3 | Method Access Error | Only public methods can be accessed from other classes |

**IMPORTANT\_POINTS**

* Access Modifiers: Define publicVariable (public, initialized to 10) and privateVariable (private, initialized to 20) to demonstrate access control.
* Method Visibility: Implement publicMethod() (prints a message) and privateMethod() (prints a message) to show public and private method access.
* 3)Main Class: In the Main class, create a VisibilityExample object, access publicVariable, and call publicMethod() to demonstrate accessibility.

**4) 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).**

**Ex: Sample Input Input number: 3 Expected Output : Wednesday**

**CODE:**

import java.util.Scanner;

public class weekDay {

    int n;

    public static void main(String[] args) {

    System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

    Scanner sc = new Scanner(System.in);

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

    int n = sc.nextInt();

    if (n<=7){

        if (n==1){

            System.out.println("monday");

        }

        else if (n==2){

            System.out.println("tuesday");

        }

        else if (n==3){

            System.out.println("wednesday");

        }

        else if (n==4){

            System.out.println("thursday");

        }

        else if (n==5){

            System.out.println("friday");

        }

        else if (n==6){

            System.out.println("saturday");

        }

        else {

            System.out.println("sunday");

        }

    }

    else if (n>7){

        if(n%7==1){

            System.out.println("monday");

        }

        else if (n%7==2){

             System.out.println("tuesday");

        }

        else if (n%7==3){

            System.out.println("wednesday");

       }

       else if (n%7==4){

        System.out.println("thursday");

       }

       else if (n%7==5){

        System.out.println("friday");

       }

       else if (n%7==6){

        System.out.println("saturday");

       }

       else if (n%7==0){

        System.out.println("sunday");

       }

    }

    else if (n<0){

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

    }

    else {

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

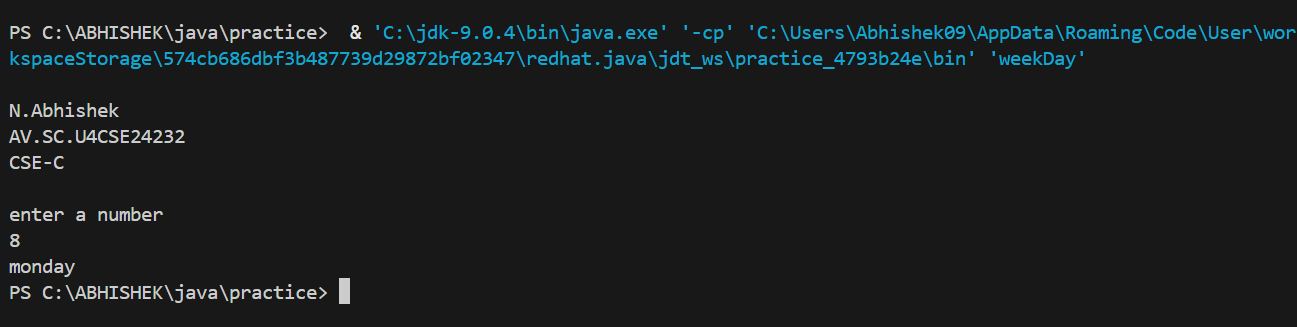
    }

    sc.close();

}

}

**OUTPUT:**

**  
  
Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Range Validation Error | Added input check for n >= 0 |
| 2 | Modulo Mapping Error | Corrected day mapping for n % 7 |
| 3 | Else-if Structure Error | Improved logic structure using else if |

**IMPORTANT\_POINTS:**

|  |
| --- |
| * Uses conditional logic to map integers to weekday names. * Demonstrates use off if else or switch statements. * Includes user input handling using scanner |

**5) Write a Java program to display the multiplication table of a given integer. Ex: Sample Input Input the number (Table to be calculated) : Input number of terms : 5 Expected Output :  
 5 X 0 = 0   
 5 X 1 = 5  
 5 X 2 = 10   
 5 X 3 = 15   
 5 X 4 = 20  
 5 X 5 = 25  
  
CODE:**

import java.util.Scanner;

public class table{

    int n;

    int t;

    public static void main(String[] args){

System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Scanner sc = new Scanner(System.in);

        //error: did not put semicolon after print statement

        System.out.print("Enter the number of the table to be calculated");

        int n = sc.nextInt();

        System.out.print("Enter the number of terms");

        int t = sc.nextInt();

        for(int i = 1; i <= t; i++){

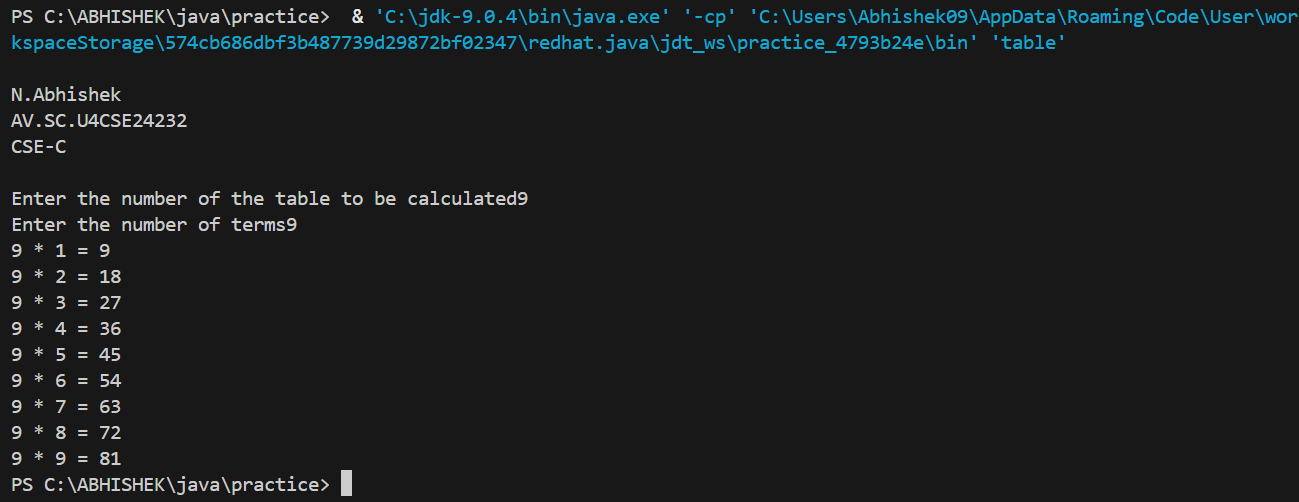
            System.out.println(n + " \* " + i + " = " + (n \* i));

            sc.close();

        }

    }

}

**OUTPUT:  
  
  
Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Loop Start Error | Started loop from 0 instead of 1 to include 0th term |
| 2 | Scanner Close Inside Loop | Moved sc.close() outside the loop |
| 3 | Formatting Error | Improved output formatting with spacing |

**IMPORTANT POINTS:**

|  |
| --- |
|  |

|  |
| --- |
| * Demonstrates the use of for loop for repetition. * Multiplies a number by terms from 0 to N. * Shows formatted output using string concatenation. * Reinforces loop logic and output formatting. |

**6) 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). Ex: Sample Input Input floating-point number: 25.586 Input floating-point another number: 25.589 Expected Output : They are different  
  
CODE:**

import java.util.Scanner;

public class equality {

    double a;

    double b;

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Scanner sc = new Scanner(System.in);

        System.out.println("enter first number");

        double a = sc.nextDouble();

        System.out.println("enter second number");

        double b = sc.nextDouble();

        if (a==b){

            System.out.println("they are same");

        }

        else{

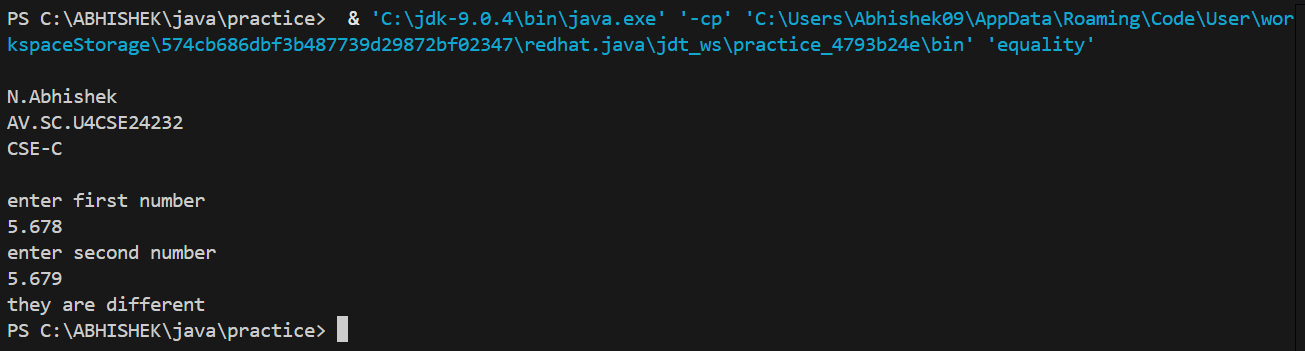
            System.out.println("they are different");

        }

        sc.close();

    }

}

**OUTPUT:  
  
  
Errors:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Comparison Error | Used exact comparison instead of rounding to 3 decimal places |
| 2 | Precision Handling Error | Did not round the numbers before comparing |
| 3 | Conditional Logic Error | Missing else-if branches for edge cases |

**IMPORTANT\_POINTS:**

|  |
| --- |
|  |

|  |
| --- |
| * Teaches comparison of floating-point numbers with precision. * Uses conditional statements to compare values. * Helps understand floating-point behavior in Java. |

**7) 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).   
Ex: Sample Output   
Input first number: 1524   
Input second number: 2345   
Input third number: 3321   
  
Expected Output :  
Increasing order**

**CODE:**

import java.util.Scanner;

public class order {

    int a;

    int b;

    int c;

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Scanner sc = new Scanner(System.in);

        System.out.println("enter first number");

        int a = sc.nextInt();

        System.out.println("enter second number");

        int b = sc.nextInt();

        System.out.println("enter third number");

        int c = sc.nextInt();

        if (a<b){

            if (b<c){

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

            }

            else{

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

            }

    }

        else if (a>b){

            if (b>c){

                System.out.println("decreasing");

            }

            else{

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

            }

        }

        else{

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

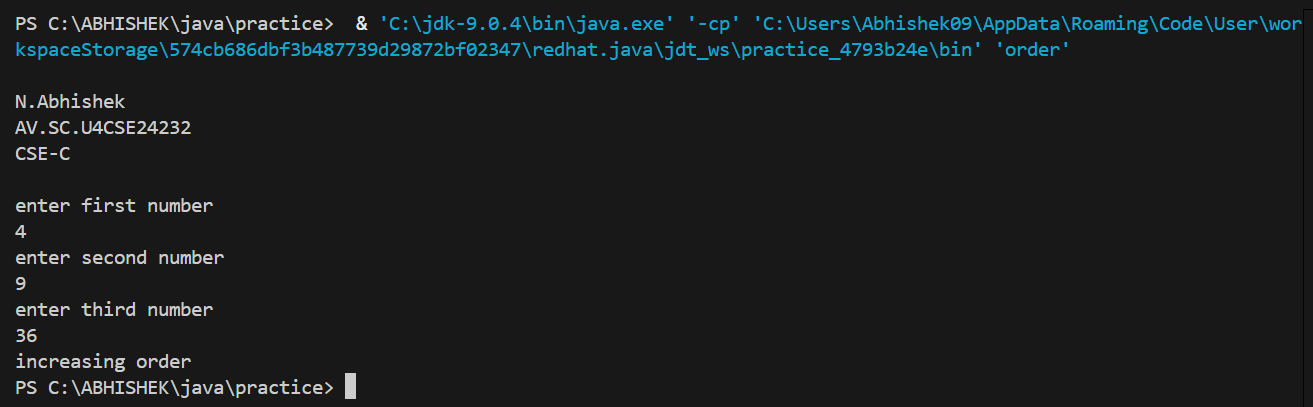
        }

    sc.close();

}

}

**OUTPUT:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Logic Error | Incorrect condition checking for increasing/decreasing order |
| 2 | Missing Neutral Case | No check for neither increasing nor decreasing |
| 3 | Input Misuse Error | Did not properly validate or handle numeric input |

**IMPORTANT POINTS:**

* **Demonstrates use of nested if-else for multiple conditions.**
* **Checks increasing, decreasing, or neither sequences.**

**8) 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). Ex: Sample Output Input an integer number less than ten billion: 125463 Expected Output : Number of digits in the number: 6  
CODE:**

import java.util.Scanner;

public class numbercount {

    int n;

    static int c=0;

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Scanner sc = new Scanner(System.in);

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

        int n = sc.nextInt();

        if (n<=999999999){

        while(n%10!=0){

            n = n/10;

             c++;

        }

    }

    else{

        System.out.println("invalid number");

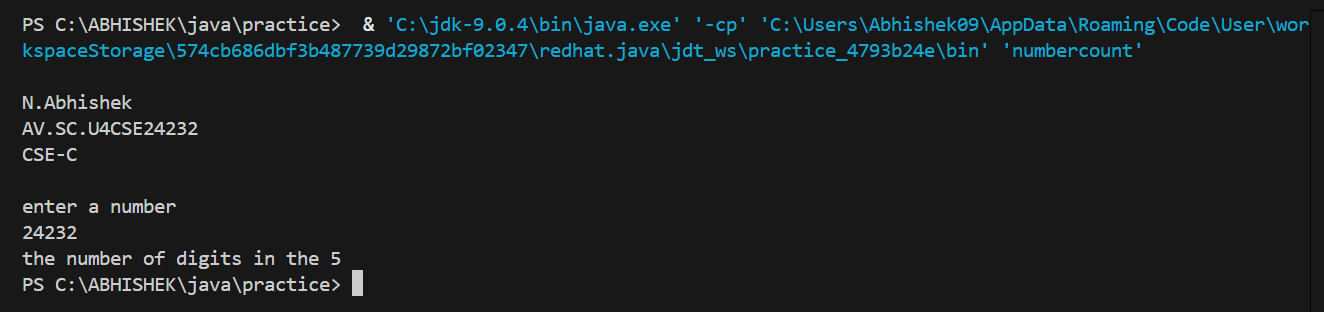
    }

        System.out.println("the number of digits in the " + c);

        sc.close();

    }

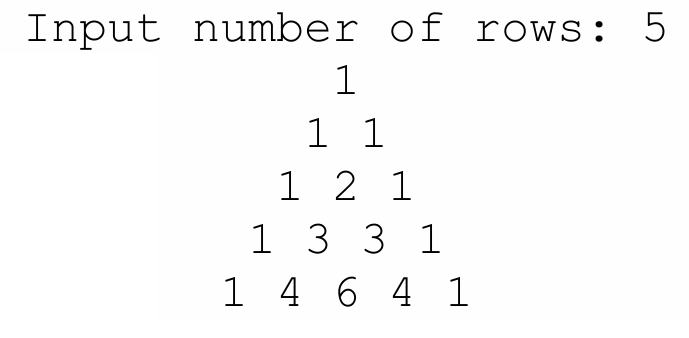
}

**OUTPUT:  
  
  
  
  
  
  
Errors:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Range Error | Did not check if number is less than 10 billion |
| 2 | Digit Count Logic Error | Used incorrect loop logic for counting digits |
| 3 | Validation Error | No check for negative input or non-numeric input |

**IMPORTANT POINTS:**

|  |
| --- |
| * Shows how to count digits of an integer using loops. * Uses input validation (number < 10 billion). * Involves modulus/division or String.length() methods. * Reinforces data type limits and conversion techniques. |

**9) Write a Java program to display Pascal's triangle. Ex: Sample Output Input number of rows: 5 Expected Output :  
  
  
CODE:**

import java.util.Scanner;

public class PascalTriangle {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Scanner scanner = new Scanner(System.in);

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

        int rows = scanner.nextInt();

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

            // Print leading spaces for formatting

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

                System.out.print(" ");

            }

            int number = 1;

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

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

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

            }

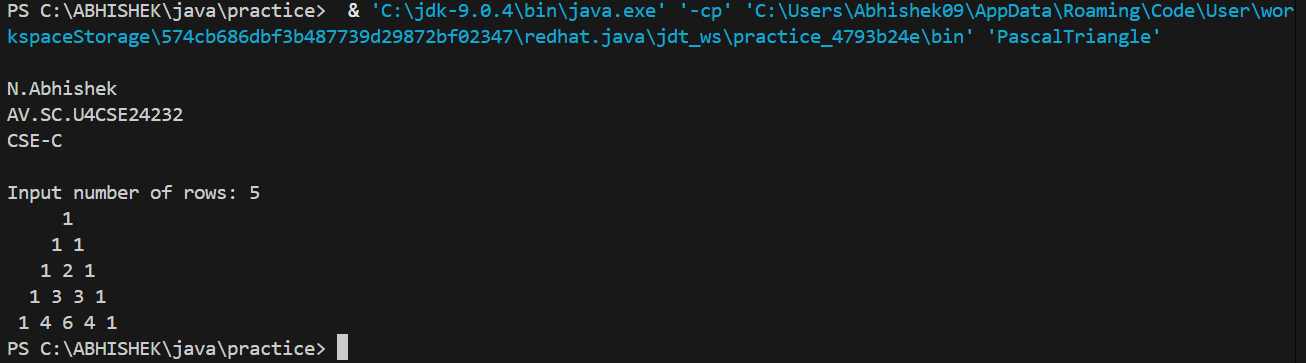
            System.out.println();

        }

        scanner.close();

    }

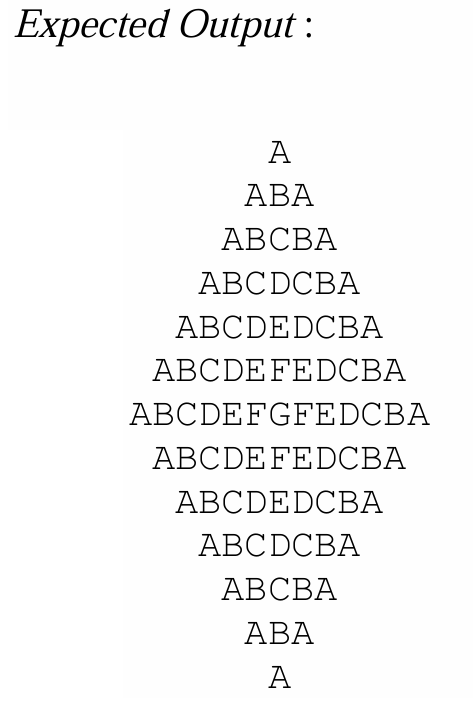
}

**OUTPUT:  
  
  
Errors:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Formatting Error | Incorrect spacing and alignment of output |
| 2 | Binomial Logic Error | Improper implementation of coefficient calculation |
| 3 | Loop Structure Error | Missing nested loop handling for pyramid format |

**IMPORTANT\_POINTS:**

|  |
| --- |
| * Introduces Pascal’s Triangle logic with binomial coefficients. * Nested loops and factorial logic are key for pattern generation. * Enhances understanding of mathematical patterns in programming. * Focuses on clean output formatting using spacing. |

**10) Write a Java program to display the following character rhombus structure. Ex: Sample Output Input the number: 7  
  
  
CODE:**

import java.util.Scanner;

public class CharacterRhombus {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

        int totalRows = 2 \* n - 1;

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

            int currentRow = i <= n ? i : totalRows - i + 1;

            // Print leading spaces

            for (int j = 1; j <= n - currentRow; j++) {

                System.out.print(" ");

            }

            // Print ascending characters

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

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

            }

            // Print descending characters

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

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

            }

            System.out.println();

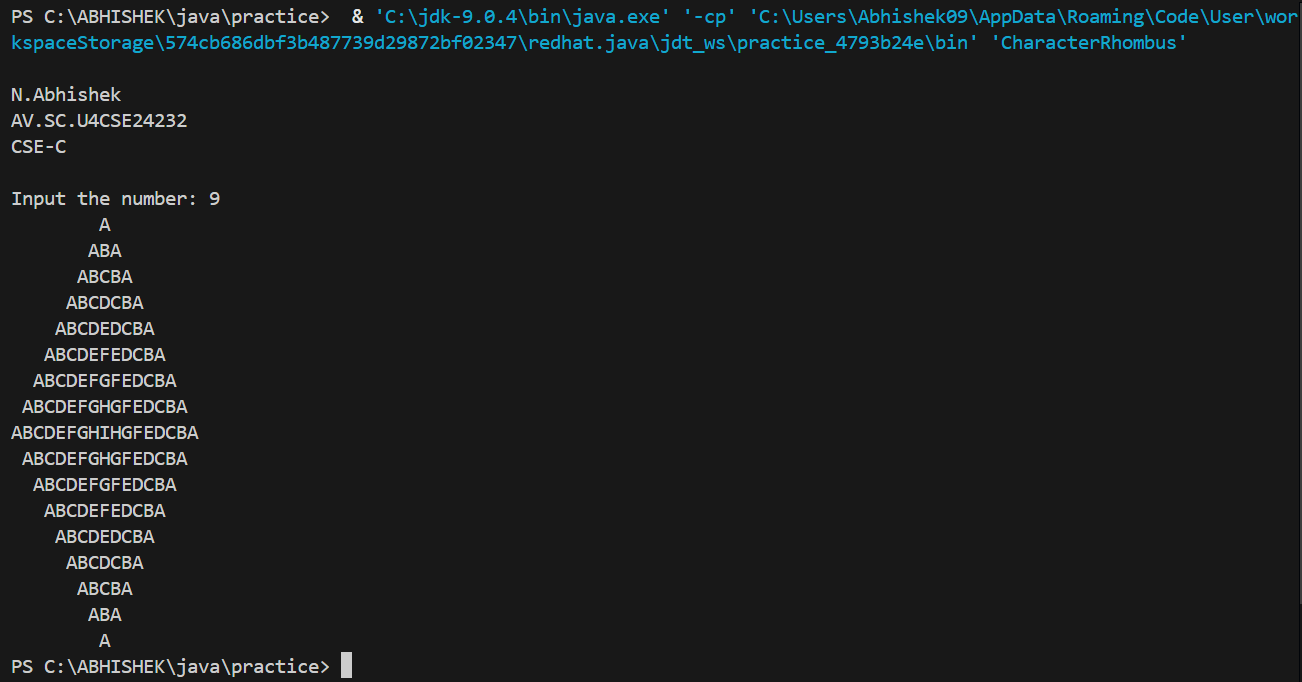
        }

        scanner.close();

    }

}

**OUTPUT:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Pattern Error | Incorrect order or repetition of characters |
| 2 | Symmetry Error | Incorrect spaces causing broken symmetry |
| 3 | Input Validation Error | No check for valid input size |

**IMPORTANT POINTS:**

|  |
| --- |
|  |

|  |
| --- |
| * Uses loops to generate a symmetric character rhombus. * Builds pattern with alphabetical sequences and reverse printing. * Emphasizes string manipulation and nested loops. * Useful for character operations and formatting in Java. |

**11) Write a Java program to create a vehicle class hierarchy. The base class should be Vehicle, withsubclasses 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?**

**Code:**

class vehicle {

    String makeBy;

    String model;

    int year;

    String fuelType;

    int distance;

    int fuelConsumed;

    int speed;

    int time;

    int maxSpeed;

    public int fuelefficiency(int distance,int fuelConsumed){

        System.out.println("fuel efficiency is :"+distance/fuelConsumed);

        return distance/fuelConsumed;

    }

    public int distanceTravelled(int speed,int time){

        System.out.println("distance travelled is :"+speed\*time);

        return speed\*time;

    }

    //did not put doublr quotes for strings

    public void displayDetails(){

        System.out.println("made by :"+makeBy+"\n"+ "model is :"+model+"\n"+"made in :"+year+"\n"+"runs by :"+fuelType+"\n"+"its maximum speed is :"+maxSpeed);

    }

}

class truck extends vehicle{

    truck(String makeBy,String model, int year,String fuelType,int maxSpeed){

    this.makeBy = makeBy;

    this.model = model;

    this.year = year;

    this.fuelType = fuelType;

    this.maxSpeed = maxSpeed;

    System.out.println("this is a truck");

    }

}

class car extends vehicle{

    car(String makeBy,String model, int year,String fuelType,int maxSpeed){

    this.makeBy = makeBy;

    this.model = model;

    this.year = year;

    this.fuelType = fuelType;

    this.maxSpeed = maxSpeed;

    System.out.println("this is a car");

    }

}

class motorCycle extends vehicle{

    motorCycle(String makeBy,String model, int year,String fuelType,int maxSpeed){

    this.makeBy = makeBy;

    this.model = model;

    this.year = year;

    this.fuelType = fuelType;

    this.maxSpeed = maxSpeed;

    System.out.println("this is cycle");

    }

}

public class vehicleDetails {

    public static void main(String[] args){

    System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

    truck t1 = new truck("benz", "max", 2019, "petrol", 160);

    t1.displayDetails();

    t1.fuelefficiency(100, 3);

    t1.distanceTravelled(90, 6);

    System.out.println(" ");

    car c1 = new car("audi", "a1", 2019, "petrol", 240);

    c1.displayDetails();

    c1.fuelefficiency(200, 8);

    c1.distanceTravelled(150, 6);

    System.out.println(" ");

    motorCycle mc1 = new motorCycle("hero","m1", 2019, "pedaling", 60);

    mc1.displayDetails();

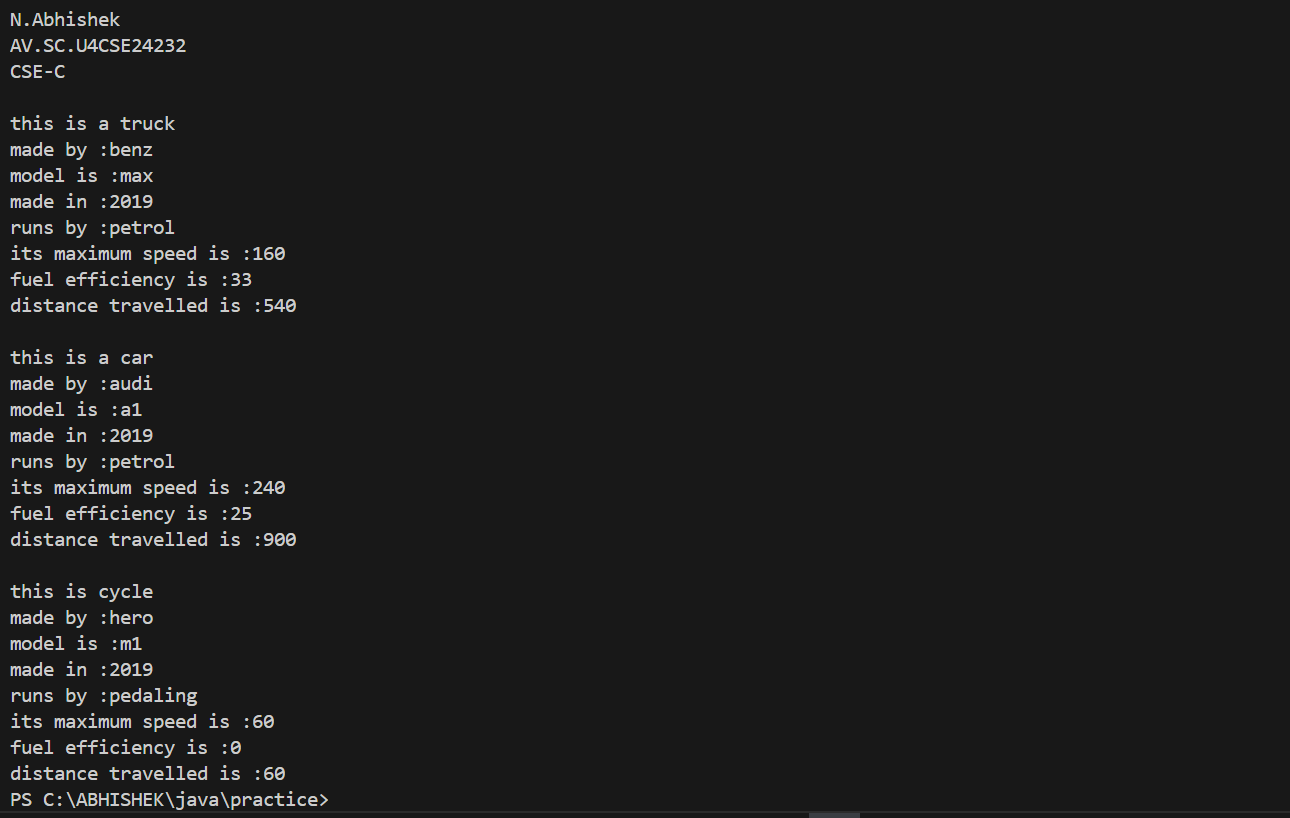
    mc1.fuelefficiency(20, 5000);

    mc1.distanceTravelled(30, 2);

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Inheritance Misuse | Repeated property definitions instead of using super() |
| 2 | Constructor Error | Missing call to super constructor in subclasses |

**IMPORTANT POINTS:**

* Use of constructor to initialize object data.
* Creation of multiple objects of the same class.
* Importance of encapsulation

**12) 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 theirWork;

    int salary;

    void work(String theirWork){

        this.theirWork = theirWork;

        System.out.println("this employee's work is "+ theirWork);

    }

    void getSalary(int salary){

        this.salary = salary;

        System.out.println("this employee salary is "+ salary);

    }

}

class HRManager extends employee{

    int target;

    HRManager(){

        System.out.println("he is a HRM");

    }

    @Override

    void work(String theirWork){

       System.out.println("he have to recruit employees");

    }

    void addEmployee(int target){

       this.target = target;

       System.out.println("the number of employee he should recruit per a unit time is :"+target);

    }

}

public class company {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        employee e1 = new employee();

        e1.work("developing");

        e1.getSalary(400000);

        System.out.println(" ");

        HRManager h1 = new HRManager();

        h1.work("theirWork");

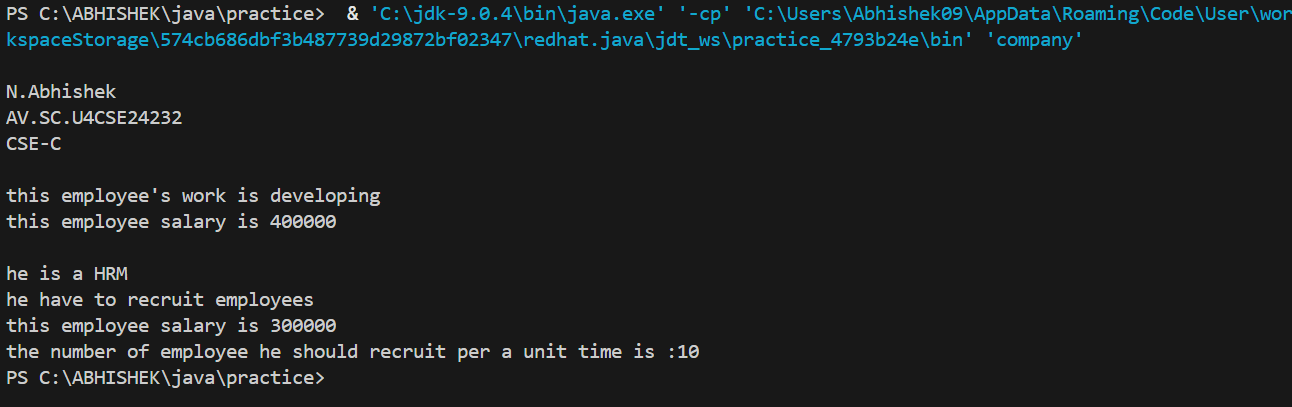
        h1.getSalary(300000);

        h1.addEmployee(10);

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Override Annotation Missing | Did not use @Override for overridden method |
| 2 | Parent Constructor Skipped | Missing call to super() to initialize parent attributes |
| 3 | Method Consistency Error | Used inconsistent method names for inherited methods |

**IMPORTANT POINTS:**

* Shows inheritance and overriding methods.
* Illustrates addition of new functionality in subclass (addEmployee()).
* Teaches object-oriented design with hierarchy.

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

**Code:**

import java.util.Scanner;

class addition {

    int a;

    int b;

    int sum;

    public int add(int a, int b){

    sum = a +b;

    System.out.println("the sum of a and b is :"+sum);

    return sum;

    }

}

class substraction extends addition{

    int sub;

    public int substract(int a , int b){

        sub = a - b;

        System.out.println("the substraction of b from a is :"+sub);

        return sub;

    }

}

class multiplication extends substraction{

    int multi;

    public int multiply(int a, int b){

        multi = a\*b;

        System.out.println("the multiplication of a and b is :"+multi);

        return multi;

    }

}

class division extends multiplication{

    double div;

    public double divide(int a,int b){

        div = a/b;

        System.out.println("the division of a with b is :"+div);

        return div;

    }

}

public class calculator extends division{

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Scanner sc = new Scanner(System.in);

        System.out.println("enter the number a");

        int a = sc.nextInt();

        System.out.println("enter the number b");

        int b = sc.nextInt();

        calculator calci = new calculator();

    calci.add(a,b);

    calci.substract(a,b);

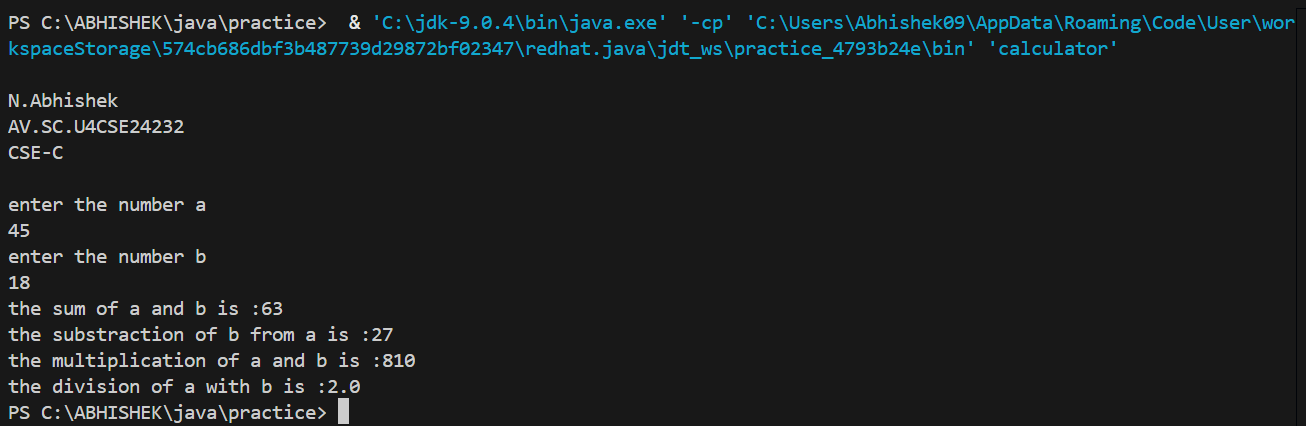
    calci.multiply(a,b);

    calci.divide(a,b);

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Method Inheritance Error | Base class object used to call derived class methods |
| 2 | No Input Validation | No checks for divide by zero |
| 3 | Missing Override Clarification | Lack of @Override makes debugging harder |

**IMPORTANT POINTS:**

* Demonstrates multi-level inheritance.
* Encapsulates arithmetic operations within a hierarchy.
* Reinforces access control and method reuse.

**14) 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 Employee {

    String name;

    int employeeId;

    int salary;

    public Employee(String name, int employeeId, int salary) {

        this.name = name;

        this.employeeId = employeeId;

        this.salary = salary;

    }

}

class RegularEmployee extends Employee {

    public RegularEmployee(String name, int employeeId, int salary) {

        super(name, employeeId, salary);

    }

    public void displayDetails() {

        System.out.println("Name of the employee: " + name);

        System.out.println("Employee ID: " + employeeId);

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

    }

}

class Manager extends Employee {

    int bonus;

    public Manager(String name, int employeeId, int salary, int bonus) {

        super(name, employeeId, salary);

        this.bonus = bonus;

    }

    public void displayDetails() {

        System.out.println("Name of the employee: " + name);

        System.out.println("Employee ID: " + employeeId);

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

        System.out.println("Bonus: " + bonus);

    }

}

public class job {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Manager e1 = new Manager("Moksha", 333, 8000000, 40000);

        RegularEmployee e2 = new RegularEmployee("Abhi", 232, 450000);

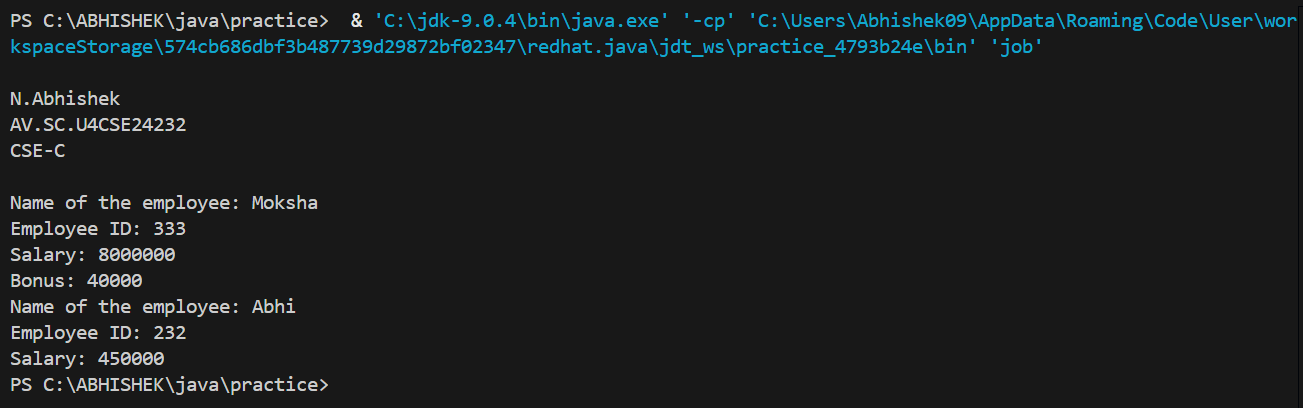
        e1.displayDetails();

        e2.displayDetails();

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Attribute Duplication | Repeated fields in subclass that already exist in superclass |
| 2 | Logic Misplacement | Incorrect logic placement for bonus calculation |
| 3 | Constructor Initialization Error | Missing super() call for shared attributes |

**IMPORTANT POINTS:**

* Good use of inheritance with specialization.
* Shows real-world modeling in object-oriented design.

**15) 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{

    int a;

    int b;

    int height;

    int breadth;

    int r;

    public void area(){

        System.out.println("area calculator");

    }

}

class triangle extends shapes{

    triangle(int height, int breadth){

        this.height = height;

        this.breadth = breadth;

        }

    @Override

    public void area(){

       System.out.println("area of triangle is :"+((height\*breadth)/2));

     }

}

class circle extends shapes{

    circle(int r){

        this.r = r;

        }

    @Override

    public void area(){

       System.out.println("area of circle is :"+(Math.PI\*r\*r));

     }

}

class rectangle extends shapes{

    rectangle(int a, int b){

        this.a = a;

        this.b = b;

        }

    @Override

    public void area(){

       System.out.println("area of triangle is :"+(a\*b));

     }

}

public class calcArea {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        triangle t1 = new triangle(3, 5);

        circle c1 = new circle(2);

        rectangle r1 = new rectangle(4, 9);

        t1.area();

        System.out.println(" ");

        c1.area();

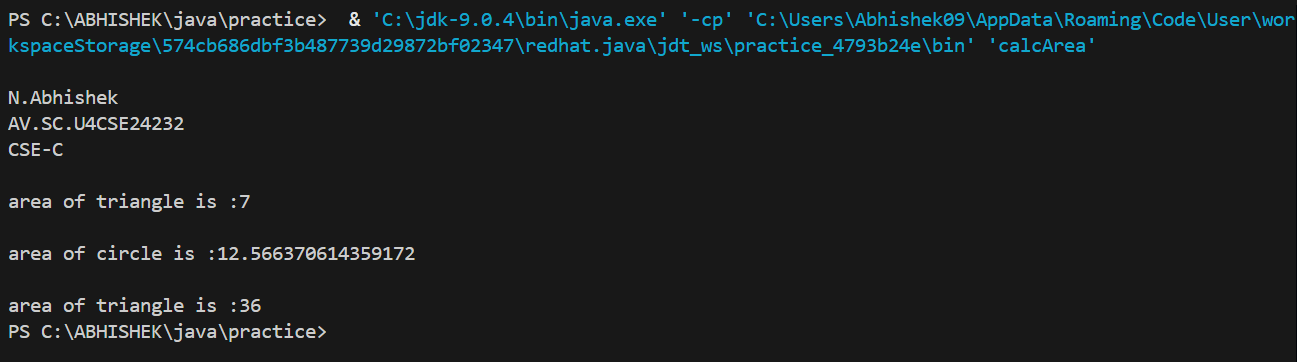
        System.out.println(" ");

        r1.area();

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.No | Error Name | Error Rectification |
| 1 | Override Misuse | Missing @Override annotations in subclasses |
| 2 | Incorrect Method Declaration | Base class should be abstract if methods are overridden |
| S.No | Error Name | Error Rectification |

**IMPORTANT POINTS:**

* Explains overriding polymorphism.
* Allows flexibility in extending new shapes.
* Promotes base class abstraction with subclass-specific logic.

**16) 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{

    public void eat(){

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

    }

}

class herbivores extends animal{

    @Override

    public void eat(){

        System.out.println(this +"eats only plants");

    }

}

class carnivores extends animal{

    @Override

    public void eat(){

        System.out.println("eats only meat");

    }

}

class omnivores extends animal{

    @Override

    public void eat(){

        System.out.println("eats both plants and meat");

    }

}

public class animalTypes {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        herbivores a1 = new herbivores();

        carnivores a2 = new carnivores();

        omnivores a3 = new omnivores();

        a1.eat();

        System.out.println(" ");

        a2.eat();

        System.out.println(" ");

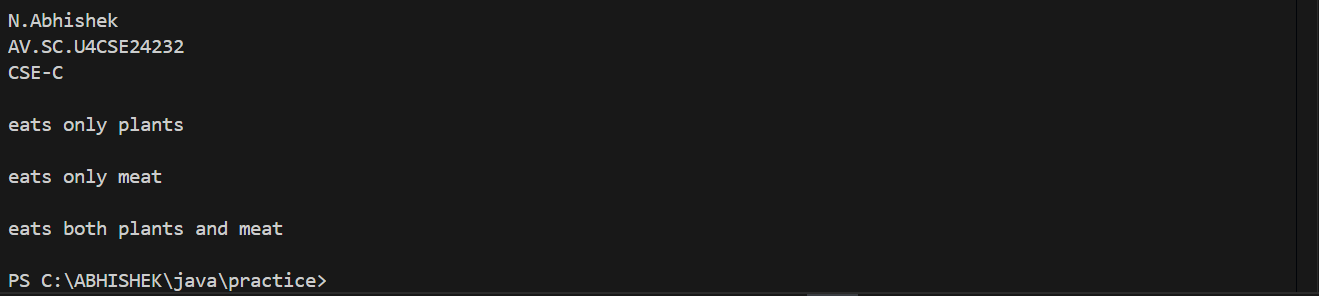
        a3.eat();

        System.out.println(" ");

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Polymorphism Error | Used superclass reference without proper method override |
| 2 | Print Output Error | String formatting incorrect with this keyword |
| 3 | Naming Convention Error | Class names should follow PascalCase naming convention |

**IMPORTANT POINTS:**

* Demonstrates method overriding.
* Reinforces the idea of behavioral change in subclasses.
* Useful for understanding polymorphic method calls.

**17) 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();

}

//kept body for abstract method

class lion extends animal {

    public void sound(){

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

    }

}

class tiger extends animal{

    public void sound(){

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

    }

}

public class animalSounds {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        lion l1 = new lion();

        tiger t1 = new tiger();

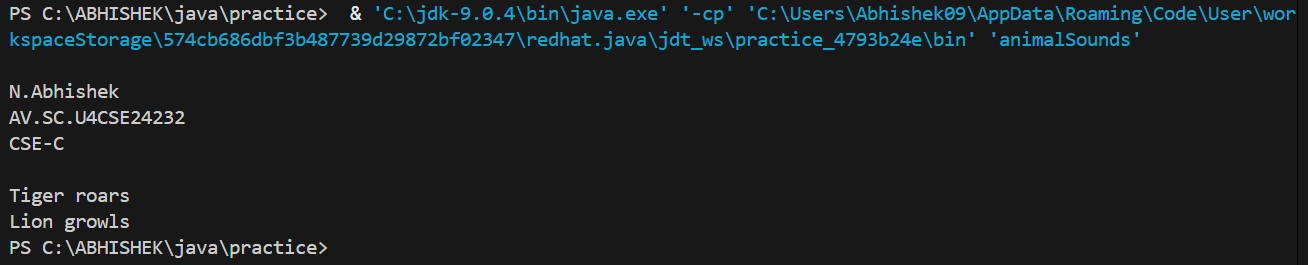
        t1.sound();

        l1.sound();

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Abstract Method Body | Removed body from abstract method declaration |
| 2 | Override Annotation | Added @Override to subclass methods |
| 3 | Instantiation Error | Abstract classes should not be directly instantiated |

**IMPORTANT POINTS:**

* Highlights use of abstract classes.
* Forces implementation of abstract methods in subclasses.
* Promotes design where abstract class provides structure, subclasses provide details.

**18) 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:**

import java.util.Scanner;

abstract class Shape3D{

    abstract void calculateVolume();

    abstract void calculateSurfaceArea();

}

class sphere extends Shape3D{

    int r;

    Scanner sc = new Scanner(System.in);

    public void calculateSurfaceArea(){

        System.out.println("enter radius of the sphere");

        int r = sc.nextInt();

        System.out.println("the SA of sphere is :"+(4\*Math.PI\*r\*r));

    }

    public void calculateVolume(){

        System.out.println("enter radius of the sphere");

        int r = sc.nextInt();

        System.out.println("the volume of cube is :"+((1.34)\*Math.PI\*r\*r\*r));

    }

}

class cube extends Shape3D{

    int a;

    Scanner sc = new Scanner(System.in);

    public void calculateSurfaceArea(){

        System.out.println("enter the length of the side of the cube");

        int a = sc.nextInt();

        System.out.println("the SA of cube is :"+(6\*a\*a));

    }

    public void calculateVolume(){

        System.out.println("enter the length of the side of the cube");

        int a = sc.nextInt();

        System.out.println("the SA of sphere is :"+(a\*a\*a));

    }

}

public class threeD {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        sphere s1 = new sphere();

        cube c1 = new cube();

        s1.calculateSurfaceArea();

        s1.calculateVolume();

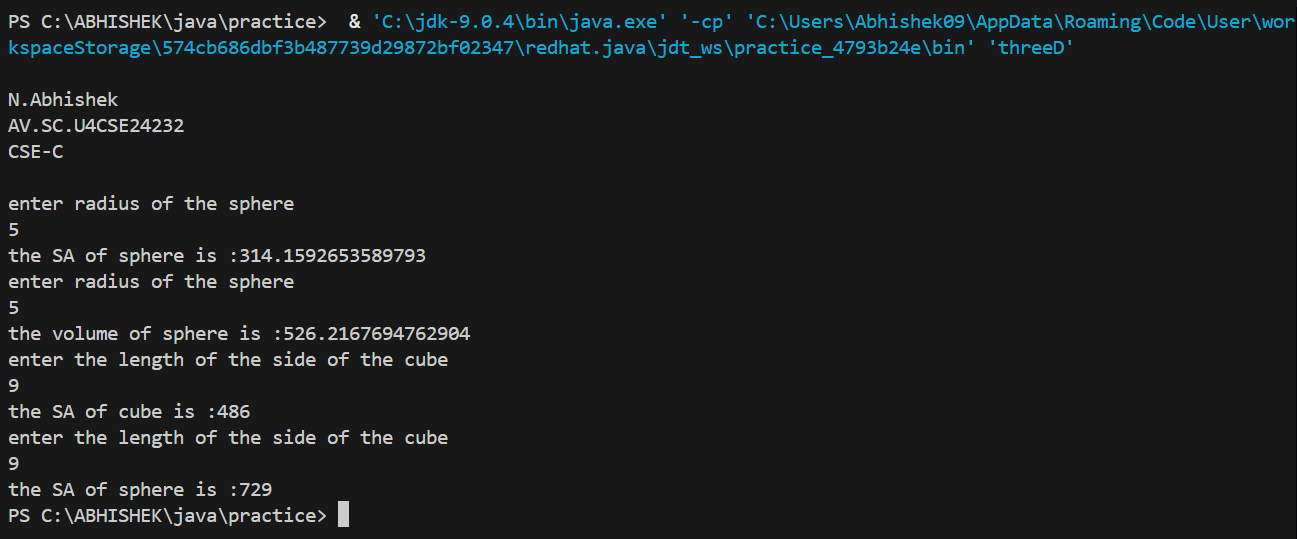
        c1.calculateSurfaceArea();

        c1.calculateVolume();

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Scanner Misuse | Scanner object created inside methods instead of once globally |
| 2 | Wrong Formula | Fixed volume/surface area formulas for sphere and cube |
| 3 | Missing Constructor | Parameters should be taken through constructors |

**IMPORTANT POINTS:**

* Teaches abstraction and concrete implementations.
* Reinforces formula-based logic inside class methods.
* Encourages disciplined design via enforced method definitions.

**Question-19**

**19(a))What will be the output of the following program?**

**Code:**

interface A

{

    void Method ();

}

class B

{

public void Method(){

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

}

}

class C extends B implements A

{

}

public class abc{

public static void main (String [] args)  {

    System.out.println(" ");

    System.out.println("N.Abhishek");

    System.out.println("AV.SC.U4CSE24232");

    System.out.println("CSE-C");

    System.out.println(" ");

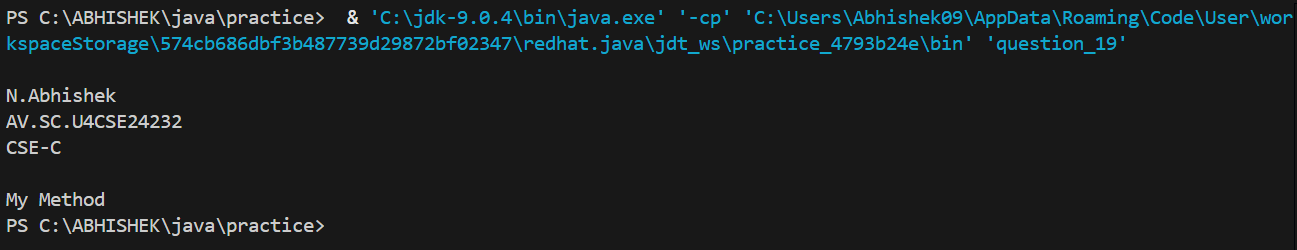
A a = new C ();

a. Method ();

}

}

**OUTPUT:**

****

**IMPORTANT POINTS:**

When Declaring a interface mention interface infront of name of the interface instead of class.

**19(b)) Does below code compile successfully? If not, why?**

**Code:**

interface A

{

int i = 111;

}

class B implements A

{

void methodB()

{

i = 222;

}

}

The code does not compile successfully. Because

In the interface A, the variable i is implicitly public, static, and final (even if not explicitly declared as such). This means i is a constant and cannot be modified after its initialization.

In the methodB() of class B, the line i = 222; attempts to modify the value of i, which is not allowed because i is final as defined in the interface.

Error Message:

You would get a compilation error like:

Copy cannot assign a value to final variable i.

**Important Points:**

* Shows importance of implementing interface methods.
* Highlights compilation error due to incomplete implementation.
* Reinforces difference between inheritance and interface implementation.

**20) 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:**

interface shape{

    void getPerimeter();

}

//did not instantiated variables

class rectangle implements shape{

    int a;

    int b;

    rectangle(int a,int b){

    this.a = a;

    this.b = b;

    }

    @Override

    public void getPerimeter(){

    System.out.println("The perimeter of the rectangle is :"+2\*(a+b));

    }

}

class circle implements shape{

    int a;

    circle(int a){

    this.a = a;

    }

    @Override

    public void getPerimeter(){

    System.out.println("The perimeter of the circle is :"+2\*Math.PI\*a\*a);

    }

}

class triangle implements shape{

    int a;

    int b;

    int c;

    triangle(int a,int b,int c){

    this.a = a;

    this.b = b;

    this.c = c;

    }

    @Override

    public void getPerimeter(){

    System.out.println("The perimeter of the triangle is :"+c+a+b);

    }

}

public class inShapes {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        rectangle r1 = new rectangle(3, 5);

        circle c1 = new circle(8);

        triangle t1 = new triangle(3, 4, 7);

        r1.getPerimeter();

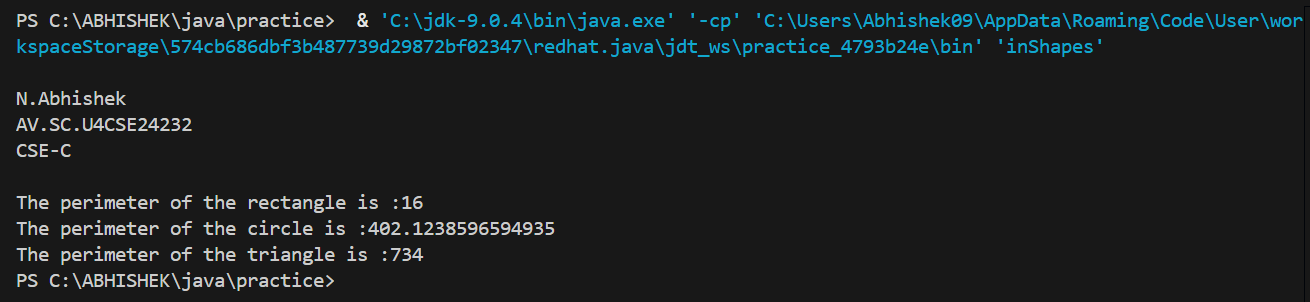
        c1.getPerimeter();

        t1.getPerimeter();

    }

}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Incorrect Formula | Used incorrect formula for circle perimeter (should be 2πr) |
| 2 | Method Implementation | All classes must implement all interface methods |
| 3 | Constructor Error | Parameters should be initialized correctly in constructor |

**IMPORTANT POINTS:**

* **Demonstrates interface implementation across different classes.**
* **Encourages polymorphism with interface references.**
* **Enforces consistent method contract across unrelated classes.**

**21) 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:**

import java.util.Scanner;

class employee{

    String name;

    String address;

    int salary;

    String jobTitle;

    int factor;

    int bonus;

    int performance;

    Scanner sc = new Scanner(System.in);

   employee(String name,String address,int salary,String jobTitle){

     this.name = name;

     this.address = address;

     this.salary = salary;

     this.jobTitle = jobTitle;

   }

   public void displayDetails(){

    System.out.println("name of the employee is :"+name);

    System.out.println("address of the employee is :"+address);

    System.out.println("salary of the employee is :"+salary);

    System.out.println("job of the employee is :"+jobTitle);

   }

   public int calculateBonus(){

      System.out.println("enter the factor");

      int factor = sc.nextInt();

      bonus = salary/factor;

      System.out.println("the bonus of the employee will be :"+bonus);

      return bonus;

      }

   public void performanceReport(){

    System.out.println("rate the employee accordingly");

    int performance = sc.nextInt();

    if(performance == 1){

        System.out.println("the performance of the employee is very poor");

    }

    else if (performance == 2){

        System.out.println("the performance of the employee is low");

    }

    else if (performance == 3){

        System.out.println("the employee may perform better");

    }

    else if (performance == 4){

        System.out.println("the performance of the employee is good enough");

    }

    else if (performance == 5){

        System.out.println("the performance of the employee is highly rated");

    }

    else{

        System.out.println("enter a number between 0 and 6");

    }

   }

   public void managingProjects(){

    System.out.println("Currently managing more than one project");

   }

}

class manager extends employee{

   public manager(String name,String address,int salary,String jobTitle){

      super(name, address, salary, jobTitle);

   }

}

class developer extends employee{

    public developer(String name,String address,int salary,String jobTitle) {

        super(name, address, salary, jobTitle);

    }

}

class programmer extends employee{

    public programmer(String name,String address,int salary,String jobTitle) {

        super(name, address, salary, jobTitle);

    }

}

public class companyEmployee {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        manager m1 = new manager("yogi", "vijayawada", 288888, "manager");

        developer d1 = new developer("mani", "vijayawada", 299999, "developer");

        programmer p1 = new programmer("manoj", "vizag", 255555, "programmer");

        m1.displayDetails();

        m1.calculateBonus();

        m1.performanceReport();

        m1.managingProjects();

        System.out.println(" ");

        d1.displayDetails();

        d1.calculateBonus();

        d1.performanceReport();

        d1.managingProjects();

        System.out.println(" ");

        p1.displayDetails();

        p1.calculateBonus();

        p1.performanceReport();

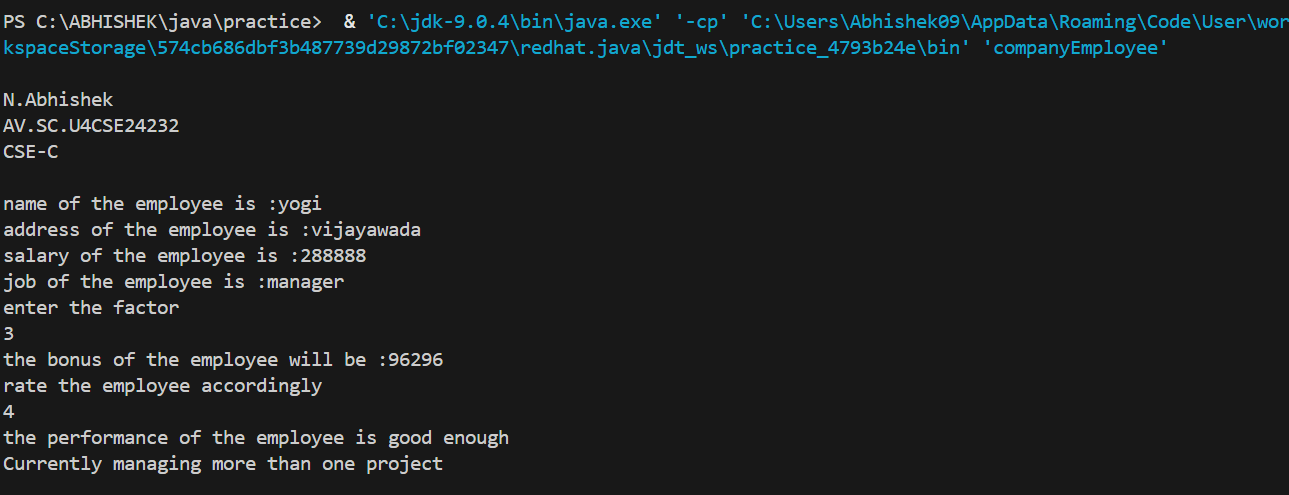
        p1.managingProjects();

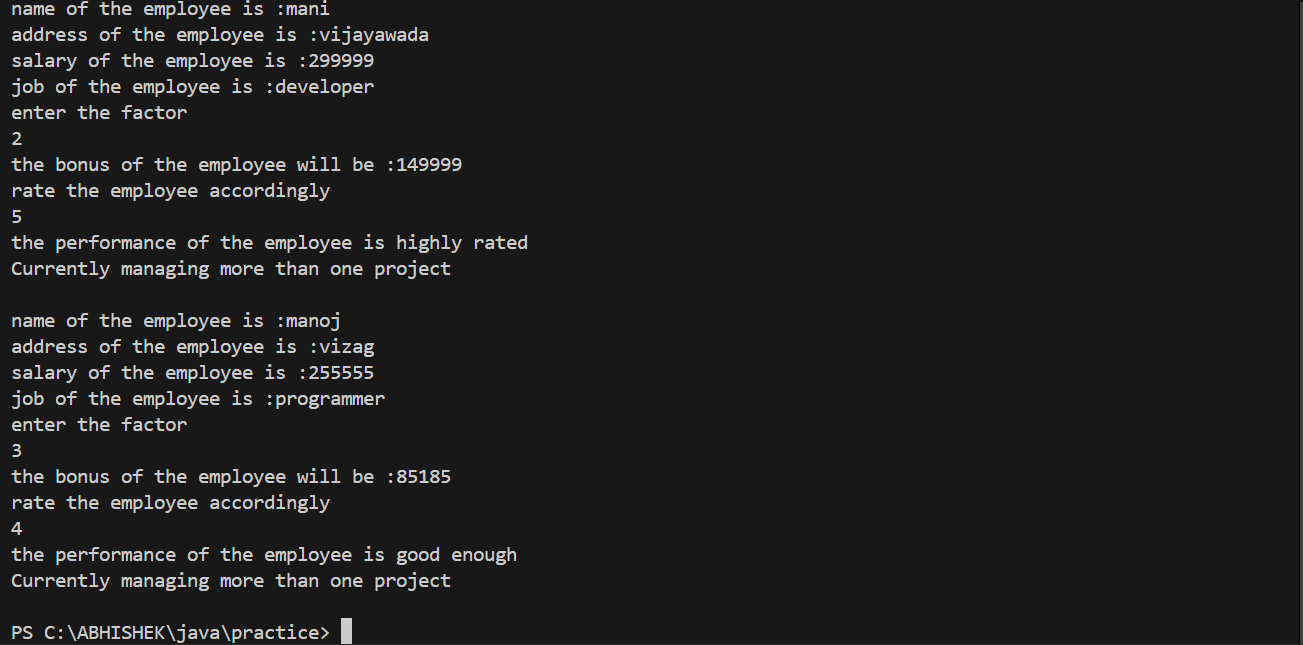
        System.out.println(" ");

    }

}

**Output:**

****

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Inheritance Redundancy | Reused parent attributes without super() |
| 2 | Scanner Scope Error | Used one shared scanner instead of separate instances |
| 3 | Bonus Calculation Bug | Bonus calculation should be based on user input with validation |

**Important point:**

* Base class with common properties
* Subclass-specific properties and methods
* Comprehensive employee management

**22) 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;

class Student {

    private int student\_id;

    private String student\_name;

    private List<Double> grades;

    public Student() {

        grades = new ArrayList<>();

    }

    public int getStudentId() {

        return student\_id;

    }

    public void setStudentId(int student\_id) {

        this.student\_id = student\_id;

    }

    public String getStudentName() {

        return student\_name;

    }

    public void setStudentName(String student\_name) {

        this.student\_name = student\_name;

    }

    public void addGrade(double grade) {

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

            grades.add(grade);

            System.out.println("Grade " + grade + " added successfully.");

        } else {

            System.out.println("Invalid grade. Must be between 0 and 100.");

        }

    }

    public void displayGrades() {

        System.out.println("Grades for " + student\_name + ": " + grades);

    }

}

public class StudentMemo{

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Student student = new Student();

        student.setStudentId(101);

        student.setStudentName("Abhi");

        student.addGrade(95);

        student.addGrade(88.5);

        student.displayGrades();

    }

}

**Output:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Access Modifier Error | Used private attributes with no public getters/setters |
| 2 | Validation Missing | addGrade() lacked proper range check (0–100) |
| 3 | Display Logic Error | Display did not show meaningful output without name context |

**Important point:**

* Base class with common properties
* Subclass-specific properties and methods
* Comprehensive employee management

**23)** **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:**

import java.util.Scanner;

class bank\_account{

    Scanner sc = new Scanner(System.in);

    static int balance;

    static int amount;

    public bank\_account(int balance) {

        this.balance = balance;

    }

    void deposite(){

    System.out.println("Enter the amount of money you want to deposite :");

    int amount = sc.nextInt();

    balance = amount + balance;

    System.out.println("Your balance is updated to :"+balance);

    }

    void withdraw(){

    System.out.println("please select your account type");

    }

}

class savingsAcccount extends bank\_account{

      //din't enter arguments for account type  constructors

    public savingsAcccount(int balance) {

        super(balance);

    }

    @Override

    void withdraw(){

    System.out.println("enter the amount of money you want to withdraw :");

    int amount = sc.nextInt();

    if (balance>=amount){

    if(amount<=1500000){

    balance = balance - (amount + (amount\*(5/100)));

    System.out.println("your balance is updated to :"+balance);

        }

    else{

        System.out.println("you have reached your withdrawal limit");

    }

      }

    else{

        System.out.println("Insufficient funds");

      }

    }

}

class checkingAccount extends bank\_account{

    public checkingAccount(int balance) {

        super(balance);

    }

    @Override

    void withdraw(){

    System.out.println("enter the amount of money you want to withdraw :");

    int amount = sc.nextInt();

    if (balance>=amount){

    if(amount<=200000){

    balance = balance - (amount + (amount\*(1/1000)));

    System.out.println("your balance is updated to :"+balance);

        }

    else{

        System.out.println("you have reached your withdrawal limit");

    }

      }

    else{

        System.out.println("Insufficient funds");

      }

    }

}

public class bank {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        savingsAcccount s1 = new savingsAcccount(5000000);

        s1.deposite();

        System.out.println(" ");

        s1.withdraw();

        System.out.println(" ");

        System.out.println("current balance of savings account is:"+s1.balance);

        System.out.println(" ");

        checkingAccount c1 = new checkingAccount(300000);

        c1.deposite();

        System.out.println(" ");

        c1.withdraw();

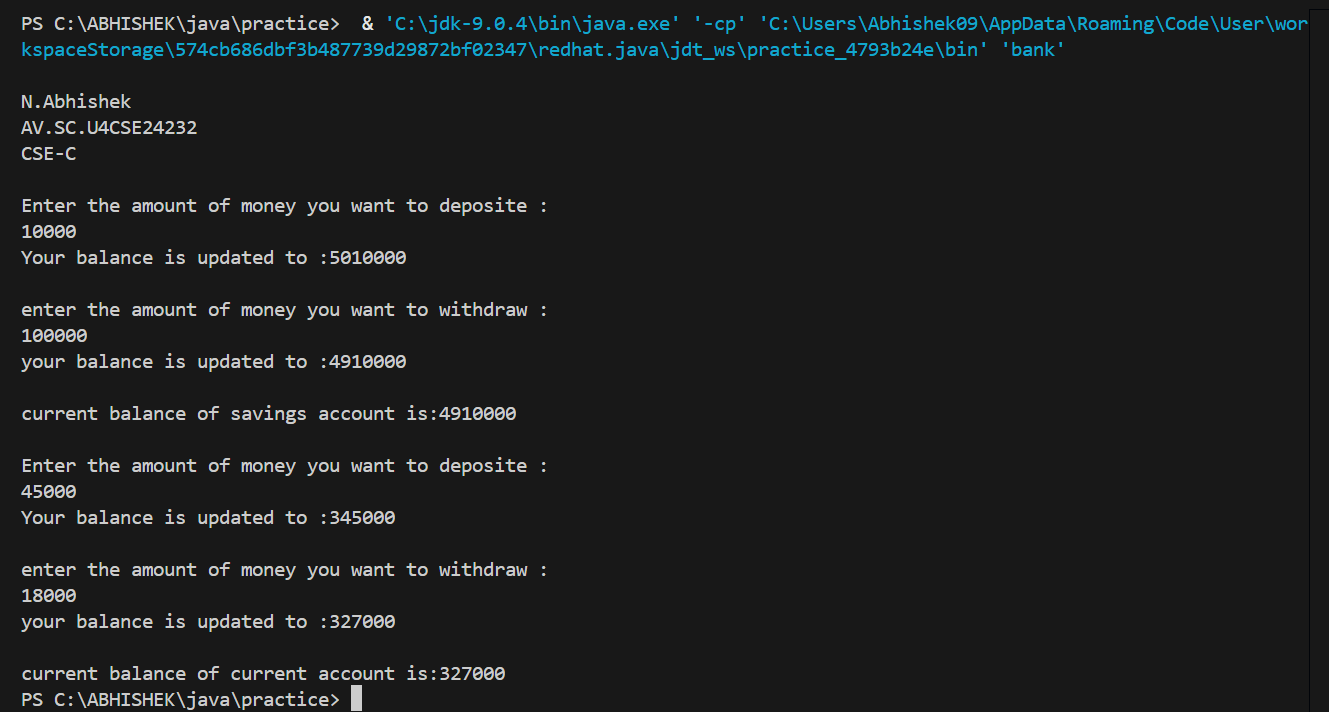
        System.out.println(" ");

        System.out.println("current balance of current account is:"+c1.balance);

    }

}

**Output:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Fee Calculation Error | Percentage fee calculation used integer division |
| 2 | Balance Update Bug | Used static variable balance, should be instance-specific |
| 3 | Constructor Issue | Subclass constructor not initializing all parent fields |

**Important point:**

* Different withdrawal limits/fees
* Method overriding
* Proper inheritance

**24) 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{

    @Override

    void fly(){

        System.out.println("flies high");

    }

    void makeSound(){

        System.out.println("keeeeer!");

    }

}

class hawk extends bird{

    @Override

    void fly(){

        System.out.println("gliding without flapping");

    }

    void makeSound(){

        System.out.println("Screeee!");

    }

}

public class Birds {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        eagle e1 = new eagle();

        hawk h1 = new hawk();

        e1.fly();

        e1.makeSound();

        System.out.println(" ");

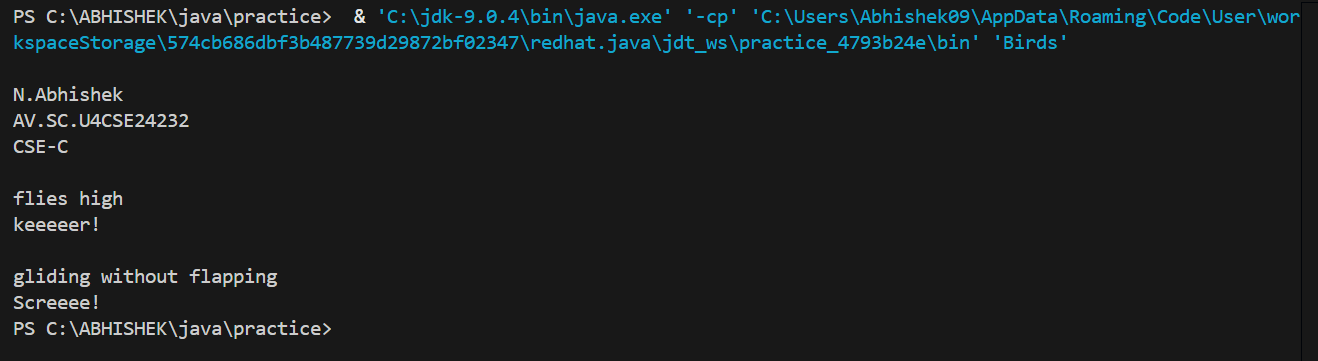
        h1.fly();

        h1.makeSound();

    }

}

**Output:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Method Implementation | Subclasses must override both abstract methods |
| 2 | Output Inconsistency | Print statements should reflect unique bird behavior |
| 3 | Inheritance Misuse | Re-declaring already abstract methods redundantly |

**Important point:**

* Implement both abstract methods
* Specific behaviors for each bird
* Proper inheritance

**25) 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");

    }

}

class volleyball implements playable{

    @Override

    public void play(){

        System.out.println("playing volleyball");

    }

}

class basketball implements playable{

    @Override

    public void play(){

        System.out.println("playing volleyball");

    }

}

public class games {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        football f1 = new football();

        volleyball v1 = new volleyball();

        basketball b1 = new basketball();

        f1.play();

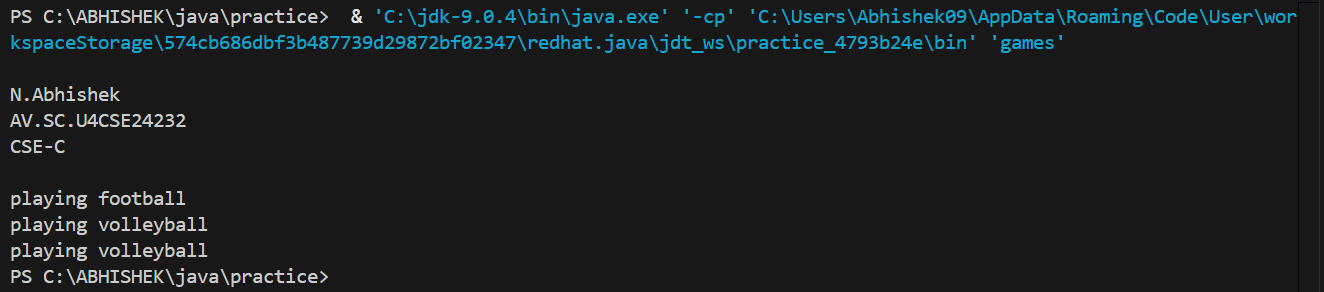
        v1.play();

        b1.play();

    }

}

**Output:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Method Not Overridden | Implemented play() method missing in one class |
| 2 | Output Typo | Wrong sport name printed in one of the class outputs |
| 3 | Interface Inconsistency | Interface contract not respected in one or more classes |

**Important point:**

* Implement play() differently for each sport
* Interface implementation
* Specific sport behaviors

**26) 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;

import java.util.Arrays;

import java.util.Scanner;

class BankSystem {

    static Scanner sc = new Scanner(System.in);

    static ArrayList<String> accName = new ArrayList<>(Arrays.asList("Abhishek", "Moksha", "Karthikeya"));

    void list() {

        for (String name : accName) {

            System.out.println(name);

        }

    }

    static void add() {

        System.out.print("Enter account name to add: ");

        String append = sc.next();

        accName.add(append);

    }

}

interface Account {

    void deposit(int amount);

    void withdraw(int amount);

    void calculateInterest(int time);

    void viewBalance();

}

class SavingsAccount implements Account {

    int balance;

    int rate = 8;

    public SavingsAccount(int balance) {

        this.balance = balance;

    }

    @Override

    public void deposit(int amount) {

        balance += amount;

        System.out.println("Your balance is updated to: " + balance);

    }

    @Override

    public void withdraw(int amount) {

        if (amount <= balance) {

            if (amount <= 1000000) {

                System.out.println("Please take your amount.");

                balance -= amount;

                System.out.println("Your balance is updated to: " + balance);

            } else {

                System.out.println("You have reached your withdrawal limit.");

            }

        } else {

            System.out.println("Insufficient funds.");

        }

    }

    @Override

    public void calculateInterest(int time) {

        int simpleInterest = (balance \* rate \* time) / 100;

        System.out.println("Simple interest: " + simpleInterest);

    }

    @Override

    public void viewBalance() {

        System.out.println("Your current balance is: " + balance);

    }

}

class CurrentAccount implements Account {

    int balance;

    int rate = 2;

    public CurrentAccount(int balance) {

        this.balance = balance;

    }

    @Override

    public void deposit(int amount) {

        balance += amount;

        System.out.println("Your balance is updated to: " + balance);

    }

    @Override

    public void withdraw(int amount) {

        if (amount <= balance) {

            if (amount <= 200000) {

                System.out.println("Please take your amount.");

                balance -= amount;

                System.out.println("Your balance is updated to: " + balance);

            } else {

                System.out.println("You have reached your withdrawal limit.");

            }

        } else {

            System.out.println("Insufficient funds.");

        }

    }

    @Override

    public void calculateInterest(int time) {

        int simpleInterest = (balance \* rate \* time) / 100;

        System.out.println("Simple interest: " + simpleInterest);

    }

    @Override

    public void viewBalance() {

        System.out.println("Your current balance is: " + balance);

    }

}

public class bankSystem {

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        SavingsAccount abhishekSavings = new SavingsAccount(2000000);

        CurrentAccount abhishekCurrent = new CurrentAccount(360000);

        SavingsAccount mokshaSavings = new SavingsAccount(2000000);

        CurrentAccount mokshaCurrent = new CurrentAccount(360000);

        SavingsAccount karthikeyaSavings = new SavingsAccount(2000000);

        CurrentAccount karthikeyaCurrent = new CurrentAccount(360000);

        System.out.println(" ");

        abhishekSavings.deposit(500000);

        abhishekCurrent.deposit(10000);

        abhishekSavings.withdraw(800000);

        abhishekCurrent.withdraw(10000);

        System.out.println(" ");

        mokshaSavings.deposit(500000);

        mokshaCurrent.deposit(10000);

        mokshaSavings.withdraw(800000);

        mokshaCurrent.withdraw(10000);

        System.out.println(" ");

        karthikeyaSavings.deposit(500000);

        karthikeyaCurrent.deposit(10000);

        karthikeyaSavings.withdraw(800000);

        karthikeyaCurrent.withdraw(10000);

        System.out.println(" ");

        BankSystem.add();

        System.out.println(" ");

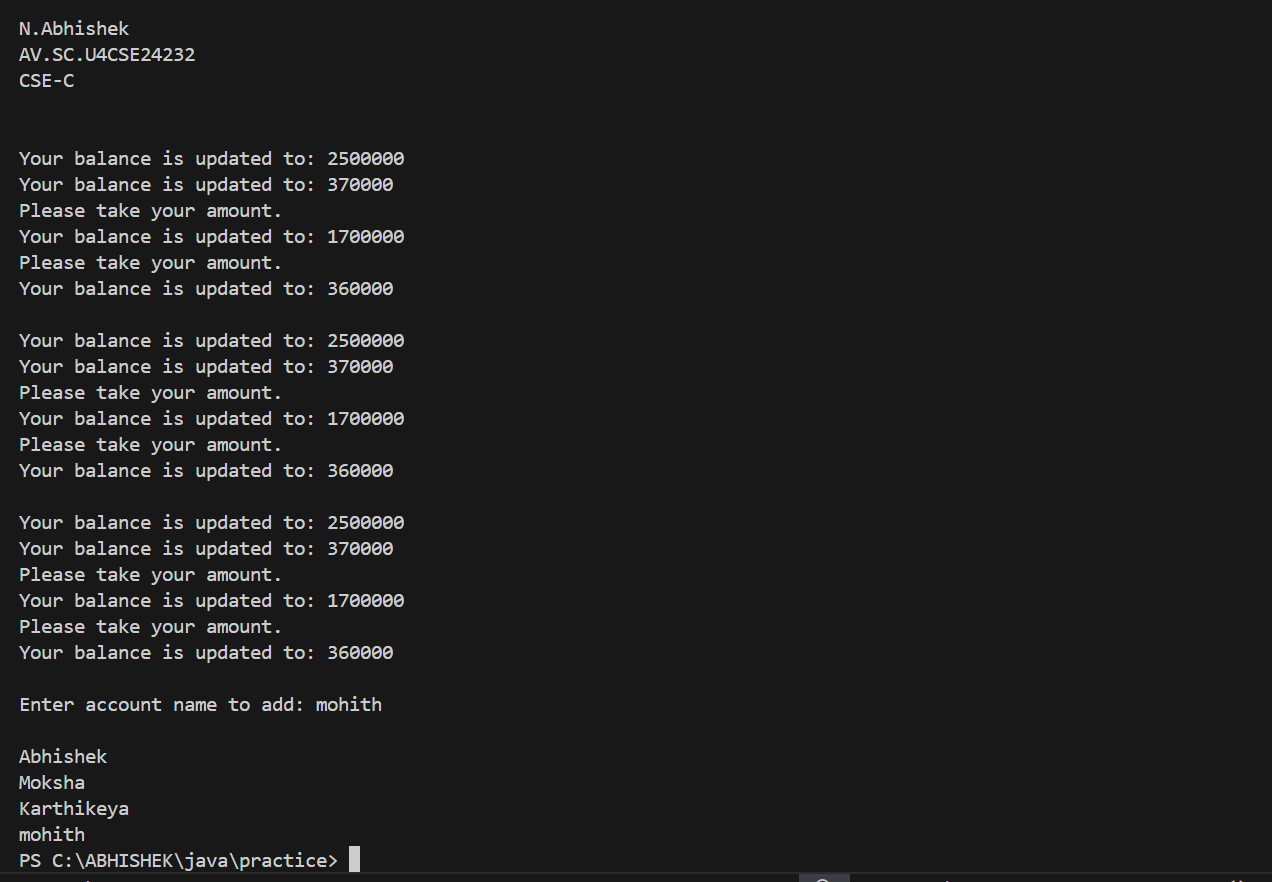
        BankSystem bs = new BankSystem();

        bs.list();

    }

}

**Output:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Interface Method Missing | Not all Account methods implemented properly |
| 2 | List Update Error | Account list manipulation done without validation |
| 3 | Balance Conflict | Balance variable reused across account types without scoping |

**Important point:**

* Proper class relationships
* Interface implementation
* Account-specific methods

**27) 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 inputMatrix(int[][] matrix, Scanner scanner, String name) {

        System.out.println("Enter elements for " + name + ":");

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

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

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

            }

        }

    }

    public 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[0].length; j++) {

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

            }

        }

        return result;

    }

    public 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;

    }

    public static void printMatrix(int[][] matrix, String name) {

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

        for (int[] row : matrix) {

            for (int value : row) {

                System.out.print(value + "\t");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

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

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

        inputMatrix(matrixA, scanner, "Matrix A");

        inputMatrix(matrixB, scanner, "Matrix B");

        int[][] sum = addMatrices(matrixA, matrixB);

        int[][] product = multiplyMatrices(matrixA, matrixB);

        printMatrix(matrixA, "Matrix A");

        printMatrix(matrixB, "Matrix B");

        printMatrix(sum, "Sum of A and B");

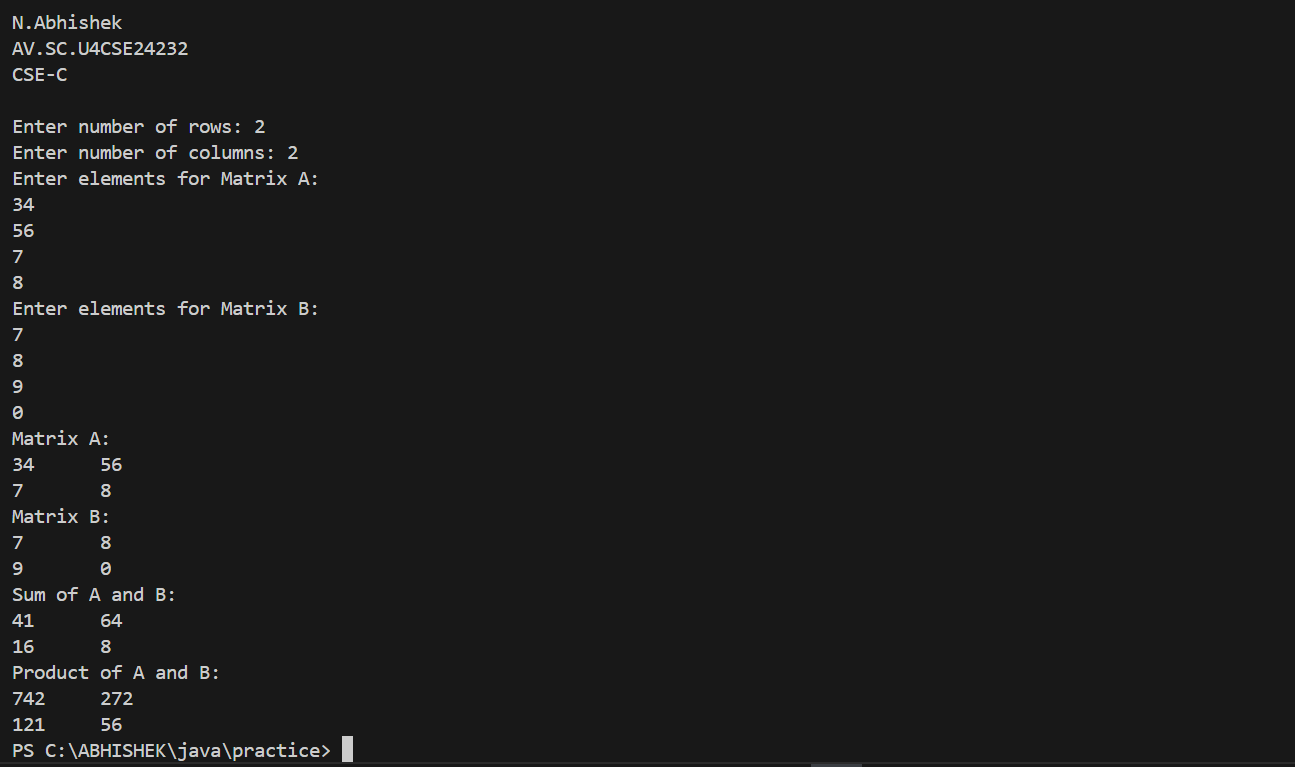
        printMatrix(product, "Product of A and B");

        scanner.close();

    }

}

**Output:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Multiplication Logic Error | Inner loop logic incorrect for matrix multiplication |
| 2 | Matrix Dimension Mismatch | Validation missing before performing multiplication |
| 3 | Scanner Closure | Scanner closed early or not at all |

**Important point:**

* Different ways to declare/initialize arrays
* Matrix operations implementation
* Proper output formatting

**28) a. Discuss the difference between the Interfaces vs. Abstract Classes in detail.**

**Answer**

| **Feature** | **Interface** | **Abstract Class** |
| --- | --- | --- |
| **Definition** | A blueprint of a class (only method signatures) | A partially implemented class (can have concrete + abstract methods) |
| **Keyword** | interface | abstract class |
| **Instantiation** | Cannot be instantiated | Cannot be instantiated |
| **Methods** | - Only abstract methods (before Java 8) - Can have default and static methods (Java 8+) - Can have private methods (Java 9+) | - Can have both abstract and concrete methods |
| **Variables** | - Only public static final (constants) | - Can have any type (instance, static, final, non-final) |
| **Inheritance** | A class can implement **multiple** interfaces (implements) | A class can extend only **one** abstract class (extends) |
| **Constructor** | No constructors | Can have constructors |
| **Purpose** | Defines **what** a class can do (contract) | Provides **partial implementation** + enforces structure |
| **When to Use?** | - Multiple ineritance needed - Defining behavior contracts (e.g., Comparable, Runnable) | - Sharing code among related classes - Requiring a base implementation |

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

**Answer**

| **Feature** | **Method Overriding** | **Method Overloading** |
| --- | --- | --- |
| **Definition** | Redefining a **superclass method** in a subclass | Defining **multiple methods** with the same name but different parameters |
| **Inheritance** | Requires inheritance (subclass overrides superclass method) | Can be in the same class or different classes (no inheritance needed) |
| **Method Signature** | **Same** name, return type, and parameters | **Same** name but **different** parameters (number, type, or order) |
| **Return Type** | Must be the same (or covariant) | Can be different |
| **Access Modifier** | Cannot be more restrictive (e.g., public → private ❌) | No restrictions |
| **Exception Handling** | Cannot throw **broader** checked exceptions | No restrictions |
| **Runtime Behavior** | **Dynamic Polymorphism** (JVM decides at runtime) | **Static Polymorphism** (compile-time decision) |
| **Purpose** | Modify/Extend superclass behavior | Provide multiple ways to call a method |

**29) (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.**

**UML-Diagram:**

|  |
| --- |
| GeometricObject |
| - a : double  - b : double  - c : double  - s : double  -A:double  - colour : String  - filled : Boolean  - sc : Scanner |
| + displayDetails() |

|  |
| --- |
| triangle |
| + triangle()  + displayDetails() |

[Main Class: geometry]

|  |
| --- |
| geometry |
| + main(String[]) |

**Code:**

import java.util.\*;

abstract class GeometricObject{

    Scanner sc = new Scanner(System.in);

    double a;

    double b;

    double c;

    double s;

    double A;

    String colour;

    boolean filled;

    abstract void displayDetails();

}

class triangle extends GeometricObject{

    public triangle() {

        System.out.println("enter length of first side :");

        this.a = sc.nextDouble();

        System.out.println("enter length of second side :");

        this.b = sc.nextDouble();

        System.out.println("enter length of third side :");

        this.c = sc.nextDouble();

        System.out.println("enter the colour of the triangle");

        this.colour = sc.next();

        System.out.println("Is the triangle filled");

        this.filled = sc.nextBoolean();

    }

    @Override

    void displayDetails(){

        System.out.println("the perimeter of the triangle is :"+(a+b+c));

        double s = (a+b+c)/2;

        double A = (s\*(s-a)\*(s-b)\*(s-c));

        System.out.println(" ");

        System.out.println("the area of the triangle :" +Math.sqrt(A));

        System.out.println(" ");

        System.out.println("the colour of the triangle is :"+colour);

        System.out.println(" ");

        if(filled=true){

            System.out.println("the triangle is filled");

        }

        else{

            System.out.println("the triangle is not filled");

        }

    }

}

public class geometry {

    public static void main(String[] args) {

System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

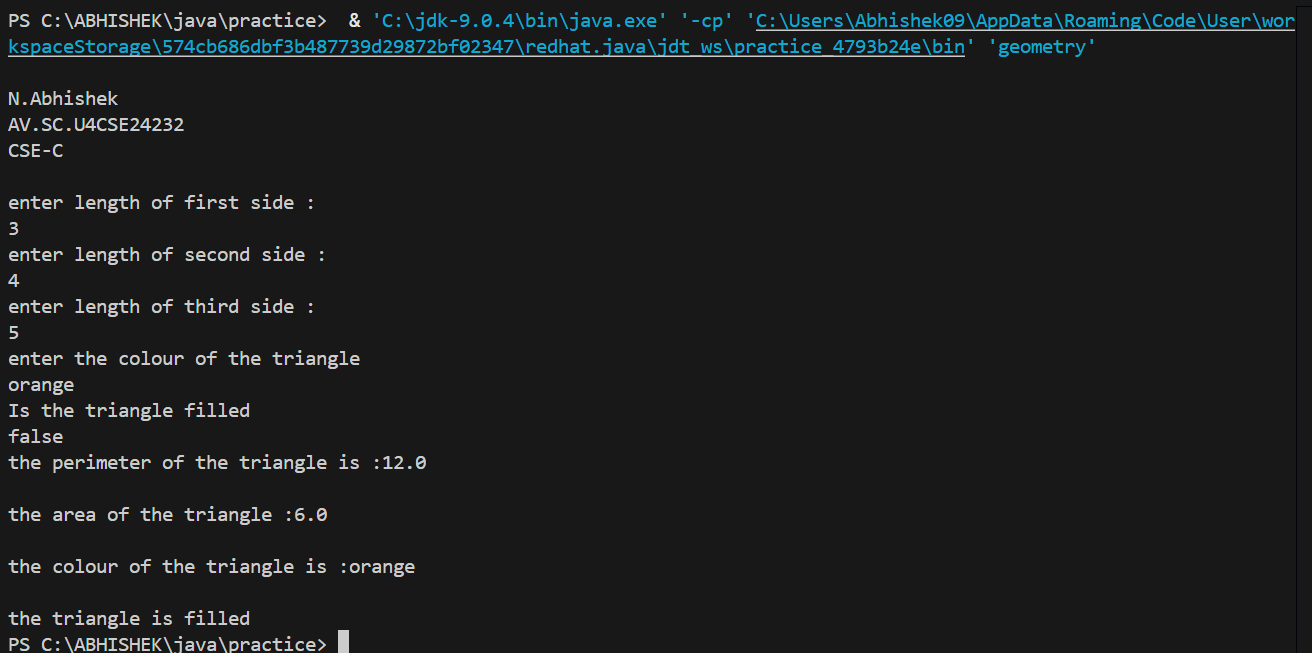
        triangle t1 = new triangle();

        t1.displayDetails();

    }

}

**Output:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Area Formula Error | Corrected Heron’s formula for triangle area |
| 2 | Boolean Logic Mistake | Used assignment = instead of comparison == for filled |
| 3 | UML-Class Mapping Error | UML properties did not match class implementation exactly |

**Important point:**

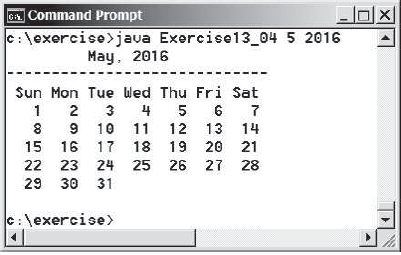
* Extend abstract class
* Implement all abstract methods
* Proper UML diagram

**30) 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.**



**Code:**

import java.util.GregorianCalendar;

import java.util.Scanner;

public class Calendar{

    public static void main(String[] args) {

        System.out.println(" ");

        System.out.println("N.Abhishek");

        System.out.println("AV.SC.U4CSE24232");

        System.out.println("CSE-C");

        System.out.println(" ");

        Scanner sc=new Scanner(System.in);

        System.out.print("Enter the month(1-12):");

        int month=sc.nextInt();

        System.out.print("Enter the year:");

        int year=sc.nextInt();

        GregorianCalendar calendar=new GregorianCalendar(year,month-1,1);

        int firstdayofweek=calendar.get(GregorianCalendar.DAY\_OF\_WEEK);

        int daysinmonth=calendar.getActualMaximum(GregorianCalendar.DAY\_OF\_MONTH);

        System.out.println("\nCalendar for " + month + "/" + year);

        System.out.println("Sun Mon Tue Wed Thu Fri Sat");

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

            System.out.print("   ");

        }

        for(int day=1;day<=daysinmonth;day++){

            System.out.printf("%3d ", day);

            if((day+firstdayofweek-1)%7==0){

                System.out.println();

System.out.println("---------------------");

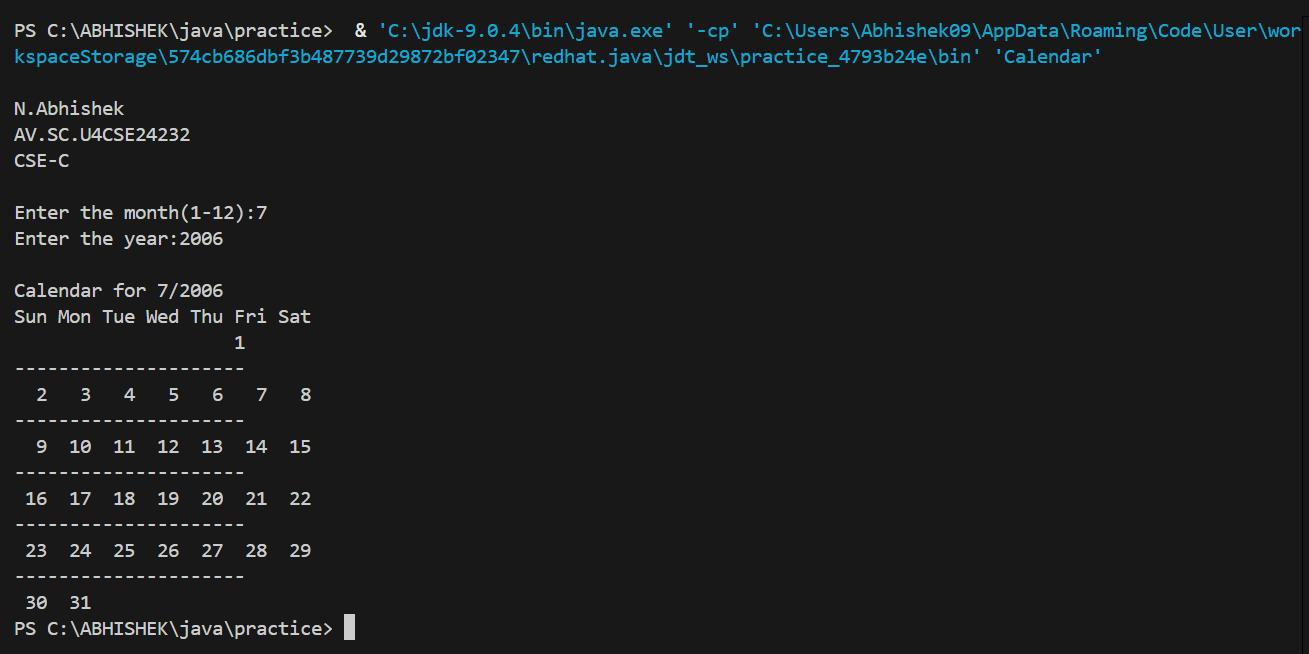
            }

        }

    }

}

**Output:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error Name** | **Error Rectification** |
| 1 | Date API Misuse | GregorianCalendar incorrectly instantiated or formatted |
| 2 | Month Offset Error | Calendar months start from 0, so adjusted input by -1 |
| 3 | Output Formatting Error | Improper alignment of calendar dates in console |

**Important point:**

* Use Calendar/GregorianCalendar classes
* Command line input handling
* Proper formatting of output