

Object Oriented Programming (23CSE111)

Assignment

|  |  |
| --- | --- |
| **Submitted by** | |
| Name | M.Pranavi |
| Roll No | AV.SC.U4CSE24302 |
| 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 | 4 |
| 2 | 2 | 6 |
| 3 | 3 | 8 |
| 4 | 4 | 12 |
| 5 | 5 | 14 |
| 6 | 6 | 16 |
| 7 | 7 | 18 |
| 8 | 8 | 20 |
| 9 | 9 | 22 |
| 10 | 10 | 24 |
| 11 | 11 | 22 |
| 12 | 12 | 26 |
| 13 | 13 | 28 |
| 14 | 14 | 31 |
| 15 | 15 | 33 |
| 16 | 16 | 35 |
| 17 | 17 | 38 |
| 18 | 18 | 39 |
| 19 | 19 | 42 |
| 20 | 20 | 45 |
| 21 | 21 | 47 |
| 22 | 22 | 52 |
| 23 | 23 | 54 |
| 24 | 24 | 57 |
| 25 | 25 | 59 |
| 26 | 26 | 61 |
| 27 | 27 | 66 |
| 28 | 28 | 70 |
| 29 | 29 | 73 |
| 30 | 30 | 76 |

**PROGRAM-1**

**AIM:**

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 year\_of\_publication;

public Book(String title, String author, int year\_of\_publication ) {

this.title = title;

this.author = author;

this.year\_of\_publication = year\_of\_publication;

}

public void displayDetails() {

System.out.println(this.title);

System.out.println(this.author);

System.out.println(this.year\_of\_publication);

System.out.println();

}

}

public class Main {

public static void main(String[]args) {

System.out.println("M.Pranavi");

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

System.out.println("AV.SC.U4CSE24302");

Book book\_one = new Book("Golden Hour", "JVKE", 2007);

Book book\_two = new Book("Car’s Outside", "JAMES ARTHUR", 2001);

System.out.println("Book one details: ");

book\_one.displayDetails();

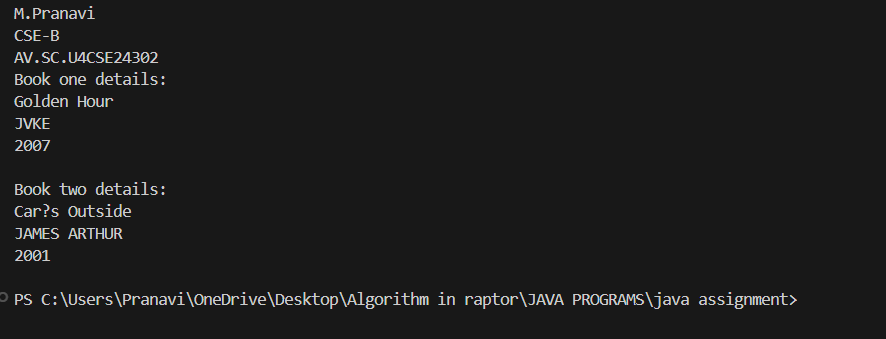
System.out.println("Book two details: ");

book\_two.displayDetails();

}

}

**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) {

myclass obj1 = new myclass();

myclass obj2 = new myclass();

myclass obj3 = new myclass();

System.out.println("M.Pranavi");

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

System.out.println("AV.SC.U4CSE24302");

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

System.out.println("Value of pi: " + obj1.pi);

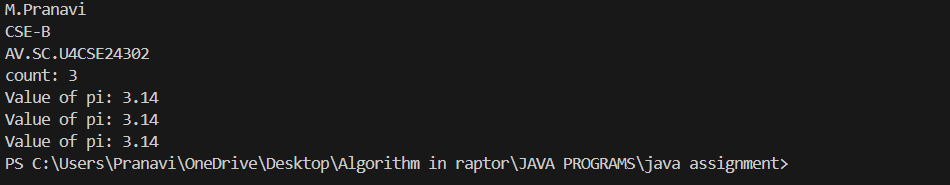
System.out.println("Value of pi: "+ obj2.pi);

System.out.println("Value of pi: " + obj3.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().
* Attempt to access the private variable privateVariable and call the private method privateMethod() in tne Main class.
* Note: attempting to do so will result in a compilation error.

**CODE:**public class VisibilityExample {

public int publicVariable = 10;

private int privateVariable = 20;

public int getprivateVariable() {

return privateVariable;

}

public void setprivateVariable(int privateVariable) {

this.privateVariable = privateVariable;

}

public int PublicVariable() {

return publicVariable;

}

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("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

VisibilityExample obj = new VisibilityExample();

System.out.println("public Variable: " + obj.PublicVariable());

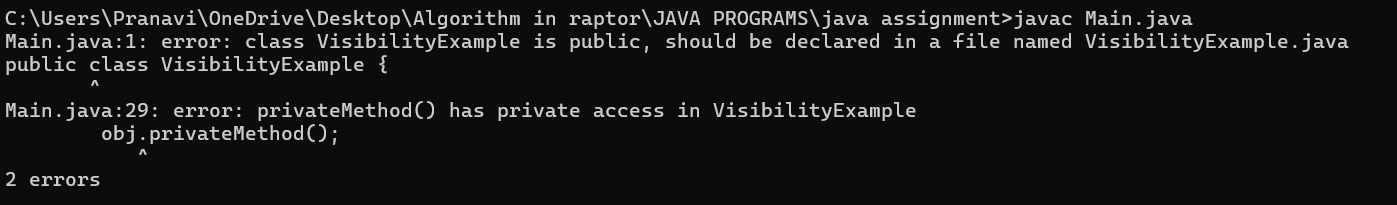
System.out.println("private Variable: " + obj.getprivateVariable());

obj.publicMethod();

obj.privateMethod();

}

}

**OUTPUT:**

**RECTIFIED CODE:**

public class VisibilityExample {

public int publicVariable = 10;

private int privateVariable = 20;

public int getprivateVariable() {

return privateVariable;

}

public void setprivateVariable(int privateVariable) {

this.privateVariable = privateVariable;

}

public int PublicVariable() {

return publicVariable;

}

public void publicMethod() {

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

privateMethod();

}

private void privateMethod() {

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

}

}

public class Main {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

VisibilityExample obj = new VisibilityExample();

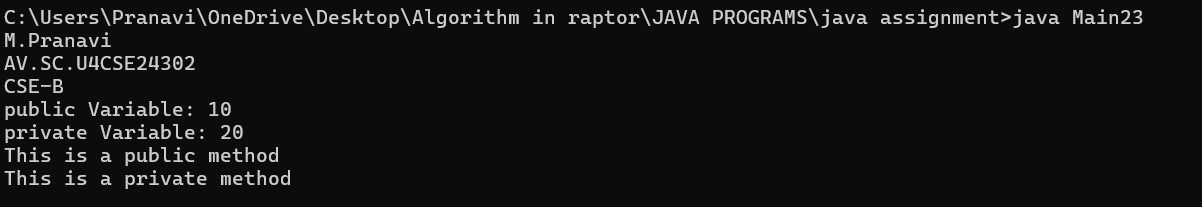
System.out.println("public Variable: " + obj.PublicVariable());

System.out.println("private Variable: " + obj.getprivateVariable());

obj.publicMethod();

}}

**RECTIFIED OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1. Private method is not visible. 2. Privatevariable not found. | 1. Create another public method or insert it in another public method to access. 2. Give proper getter and setter methods to access it. |

**IMPORTANT POINTS:**

1. Private methods are only accessible within the class where they are defined.
2. We can call a private method by declaring a public method and then accessing it by creating an object.

To access a private variable, first we initialized it, and then declared getter and setter methods to access it.

**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).

**CODE:**

import java.util.Scanner;

public class days {

public static void main(String[]args) {

Scanner input = new Scanner(System.in);

System.out.println("Enter a number(1-7): " );

int n = input.nextInt();

String day = null;

if (n==1) {

day = "Monday";

} else if (n==2) {

day = "Tuesday";

} else if (n==3) {

day = "Wednesday";

} else if (n==4) {

day = "Thursday";

} else if (n==5) {

day = "Friday";

} else if (n==6) {

day = "Saturday";

} else if (n==7) {

day = "Sunday";

} else {

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

}

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

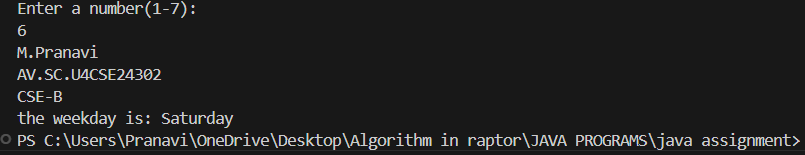
System.out.println("the weekday is: " + day );

input.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) {

Scanner input = new Scanner(System.in);

System.out.println("Table to be calculated: " );

int x = input.nextInt();

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

int n = input.nextInt();

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

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

}

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

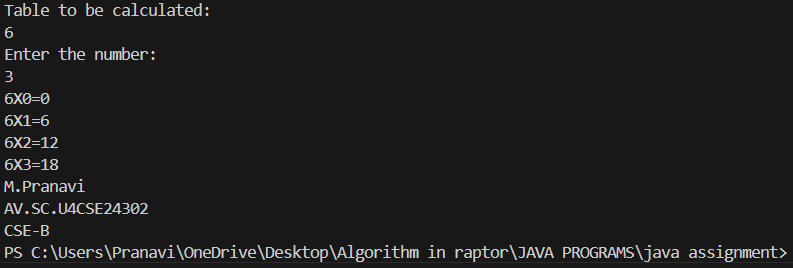
System.out.println("CSE-B");

input.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).

**CODE:**

import java.util.Scanner;

public class decimal {

public static void main(String[]args) {

Scanner input = new Scanner(System.in);

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

double a = input.nextDouble();

System.out.println("Enter b number: ");

double b = input.nextDouble() ;

if (a == b) {

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

} else {

System.out.println("They are not the same upto three decimal places.");

}

System.out.println("M.Pranavi");

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

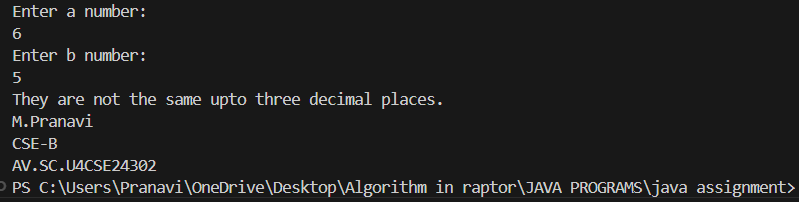
System.out.println("AV.SC.U4CSE24302");

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

**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).

**CODE:**

import java.util.Scanner;

public class order{

public static void main(String[]args) {

Scanner input = new Scanner(System.in);

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

int a = input.nextInt();

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

int b = input.nextInt();

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

int c = input.nextInt();

System.out.println("M.Pranavi");

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

System.out.println("AV.SC.U4CSE24302");

if (a<b) {

if (b<c) {

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

} else {

System.out.println("Neither inreasing nor decreasing");

}

} else {

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

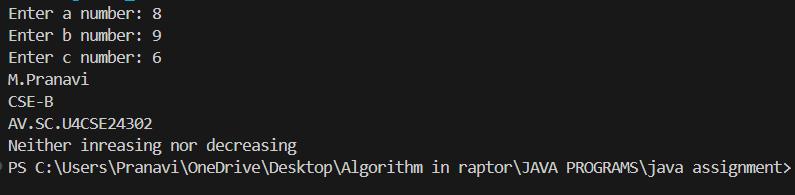
}

input.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 billion {

public static void main(String[]args) {

Scanner input = new Scanner(System.in);

System.out.print("Enter a number less than ten billion: ");

int a = input.nextInt();

int b = a;

int i = 0;

while (a>0) {

a = a/10;

i++;

}

if ( b == 0) {

i = 1;

}

System.out.println("the number of digits in the number is: " +i );

System.out.println("M.Pranavi");

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

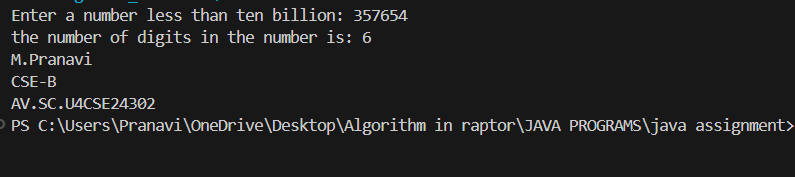
System.out.println("AV.SC.U4CSE24302");

input.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:**

public class PascalTriangle {

public static void main(String[]args) {

int r = 5;

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

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

System.out.print(" ");

}

int n = 1;

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

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

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

}

System.out.println( );

}

System.out.println("M.Pranavi");

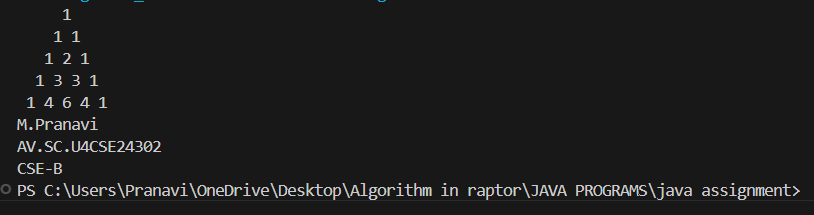
System.out.println("AV.SC.U4CSE24302");

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

}

}

**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) {

Scanner input = new Scanner(System.in);

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

int n = input.nextInt();

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

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

System.out.print(" ");

}

for (char c = 'A'; c < 'A' + i; c++) {

System.out.print(c);

}

for (char c = (char) ('A' + i - 2); c >= 'A'; c--) {

System.out.print(c);

}

System.out.println();

}

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

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

System.out.print(" ");

}

for (char c = 'A'; c < 'A' + i; c++) {

System.out.print(c);

}

for (char c = (char) ('A' + i - 2); c >= 'A'; c--) {

System.out.print(c);

}

System.out.println();

}

input.close();

System.out.println("M.Pranavi");

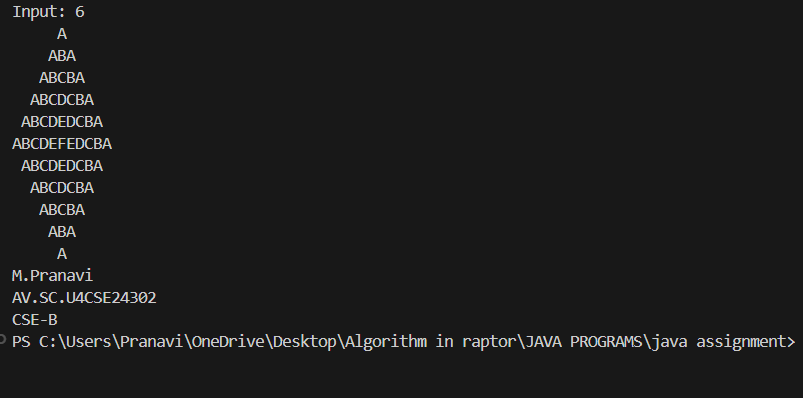
System.out.println("AV.SC.U4CSE24302");

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

}

}

**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.

**CODE:**

abstract 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;

}

abstract double fuelefficiency();

abstract double maxspeed();

public double distancetravelled(double fuelused) {

return fuelefficiency()\*fuelused;

}

public String getmake() {

return make;

}

public String getmodel() {

return model;

}

public int getyear() {

return year;

}

public String getfueltype() {

return fueltype;

}

}

class truck extends vehicle {

public truck(String make, String model, int year, String fueltype) {

super(make, model, year, fueltype);

}

public double fuelefficiency() {

return 5.5;

}

public double maxspeed() {

return 145.0;

}

}

class car extends vehicle {

public car(String make, String model, int year, String fueltype) {

super(make, model, year, fueltype);

}

public double fuelefficiency() {

return 6.0;

}

public double maxspeed() {

return 150.0;

}

}

class motorcycle extends vehicle {

public motorcycle(String make, String model, int year, String fueltype) {

super(make, model, year, fueltype);

}

public double fuelefficiency() {

return 7.0;

}

public double maxspeed() {

return 155.0;

}

}

public class vtest {

public static void main(String[] args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

vehicle truck = new truck("TATA", "R-430", 2007, "Diesel");

vehicle car = new car("HYUNDAI", "I 20 Sportz", 2022, "Petrol");

vehicle motorcycle = new motorcycle("Pleasure", "5G", 2024, "Petrol");

System.out.println("truck details:");

System.out.println("Make: " + truck.getmake() + ", Model: " + truck.getmodel());

System.out.println("Fuel Efficiency: " + truck.fuelefficiency() + " km/l");

System.out.println("Max Speed: " + truck.maxspeed() + " km/h");

System.out.println("Distance travelled:" + truck.distancetravelled(10) + " km");

System.out.println();

System.out.println("car details:");

System.out.println("Make: " + car.getmake() + ", Model: " + car.getmodel());

System.out.println("Fuel Efficiency: " + car.fuelefficiency() + " km/l");

System.out.println("Max Speed: " + car.maxspeed() + " km/h");

System.out.println("Distance traveled with 10 liters of fuel: " + car.distancetravelled(10) + " km");

System.out.println();

System.out.println("motorcycle details:");

System.out.println("Make: " + motorcycle.getmake() + ", Model: " + motorcycle.getmodel());

System.out.println("Fuel Efficiency: " + motorcycle.fuelefficiency() + " km/l");

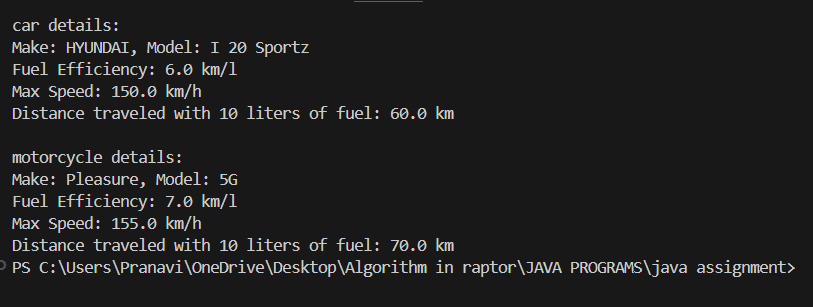
System.out.println("Max Speed: " + motorcycle.maxspeed() + " km/h");

System.out.println("Distance traveled with 10 liters of fuel: " + motorcycle.distancetravelled(10) + " km");

}

}

**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:**

public 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("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

Employee e = new Employee("Pooja", 2400000);

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

e.work();

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

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

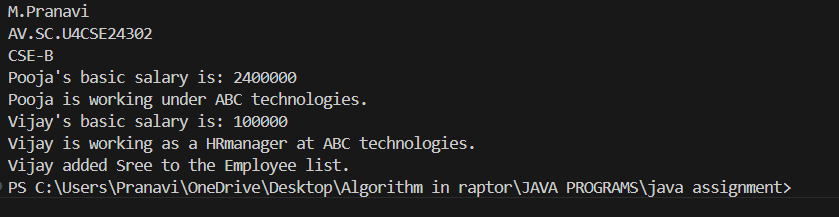
hr.work();

hr.addEmployee("Sree");

}

}

**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 {

public double a, b;

public calculator(double a, double b) {

this.a = a;

this.b = b;

}

}

class Addition extends calculator {

public Addition(double a, double b) {

super(a, b);

}

public double add() {

return a + b;

}

}

class Subtraction extends Addition {

public Subtraction(double a, double b) {

super(a, b);

}

public double subtract() {

return a - b;

}

}

class Multiplication extends Subtraction {

public Multiplication(double a, double b) {

super(a, b);

}

public double multiply() {

return a \* b;

}

}

class Division extends Multiplication {

public Division(double a, double b) {

super(a, b);

}

public double divide() {

if (b != 0) {

return a / b;

} else {

System.out.println("Error");

return Double.NaN;

}

}

}

class Final extends Division {

public Final(double a, double b) {

super(a, b);

}

public void displayResults() {

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

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

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

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

}

}

import java.util.Scanner;

public class allcalculator {

public static void main(String[] args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

Scanner input = new Scanner(System.in);

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

double a = input.nextDouble();

System.out.println("Enter b number: ");

double b = input.nextDouble();

Final calc = new Final( a, b);

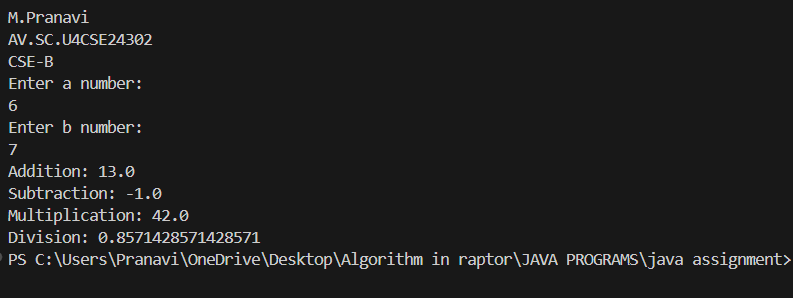
calc.displayResults();

input.close();

}

}

**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(\*).
5. 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:**

public 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 bonus(int bonus) {

System.out.println("a manager gets bonus upto: " + bonus);

}

}

public class softwaretest {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

software s = new software("Shiva", 101526);

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

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

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

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

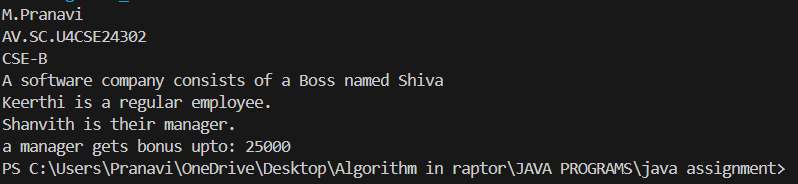
System.out.println(m.getname() + " is their manager.");

m.bonus(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:**

abstract class shapes {

abstract double area();

}

class triangle extends shapes {

public double base;

public double height;

public triangle(double base, double height) {

this.base = base;

this.height = height;

}

double area() {

return (0.5\*base\*height);

}

}

class circle extends shapes {

public double r;

public circle(double r) {

this.r = r;

}

double area() {

return (3.14\*r\*r);

}

}

class rectangle extends shapes {

public double l;

public double b;

public rectangle(double l, double b) {

this.l = l;

this.b = b;

}

double area() {

return(l\*b);

}

}

public class shapestest {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

triangle t = new triangle(2,5);

circle c = new circle(4);

rectangle r = new rectangle(2,5);

System.out.println("Area of a Triangle is: " + t.area());

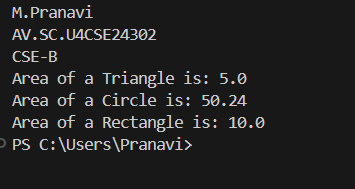
System.out.println("Area of a Circle is: " + c.area());

System.out.println("Area of a Rectangle is: " + r.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.
3. 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:**

public class Animal {

public String example(String name) {

System.out.println("Example of an Animal: " + name);

return name;

}

public void eat() {

System.out.println("All the animals follow the food chain.");

}

}

class Herbivores extends Animal {

public void eat() {

System.out.println("Herbivores eat grass.");

}

public String example(String name) {

System.out.println("Example of a Herbivore: " + name);

return name;

}

}

class Carnivores extends Animal {

public void eat() {

System.out.println("Carnivores eat Herbivores.");

}

public String example(String name) {

System.out.println("Example of a Carnivore: " + name);

return name;

}

}

class Omnivores extends Animal {

public void eat() {

System.out.println("Omnivores eat both Herbivores and Carnivores.");

}

public String example(String name) {

System.out.println("Example of an Omnivore: " + name);

return name;

}

}

public class Animaltest {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

Animal a = new Animal();

Herbivores h = new Herbivores();

Carnivores c = new Carnivores();

Omnivores o = new Omnivores();

a.eat();

h.eat();

h.example("Grasshopper");

c.eat();

c.example("Lion");

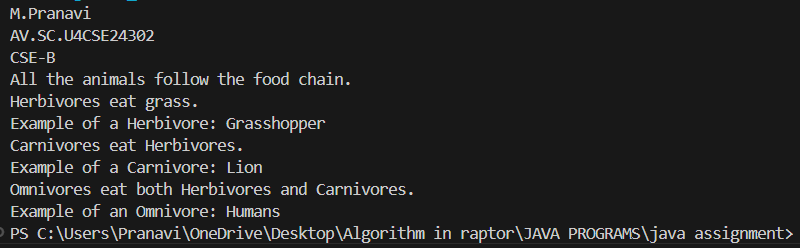
o.eat();

o.example("Humans");

}

}

**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 Animal2 {

abstract void sound();

}

class Lion extends Animal2 {

void sound() {

System.out.println("Lion roars in the jungle.");

}

}

class Tiger extends Animal2 {

void sound() {

System.out.println("Tigers roar and attack secretly.");

}

}

public class Animal2test {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302);

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

Lion l = new Lion();

l.sound();

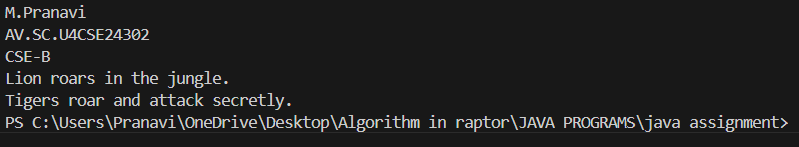
Tiger t = new Tiger();

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:**

abstract class Shape3D {

abstract double calculateVolume();

abstract double calculateSurfaceArea();

}

class Sphere extends Shape3D {

int radius;

Sphere(int radius) {

this.radius = radius;

}

@Override

public double calculateVolume() {

return (4.0 / 3.0) \* Math.PI \* Math.pow(radius, 3);

}

@Override

public double calculateSurfaceArea() {

return 4 \* Math.PI \* Math.pow(radius, 2);

}

}

class Cube extends Shape3D {

int edge;

Cube(int edge) {

this.edge = edge;

}

@Override

public double calculateVolume() {

return Math.pow(edge, 3);

}

@Override

public double calculateSurfaceArea() {

return 6 \* Math.pow(edge, 2);

}

}

public class Main {

public static void main(String[] args) {

Sphere s = new Sphere(4);

System.out.println("Sphere Volume: " + s.calculateVolume());

System.out.println("Sphere Surface Area: " + s.calculateSurfaceArea());

Cube c = new Cube(3);

System.out.println("Cube Volume: " + c.calculateVolume());

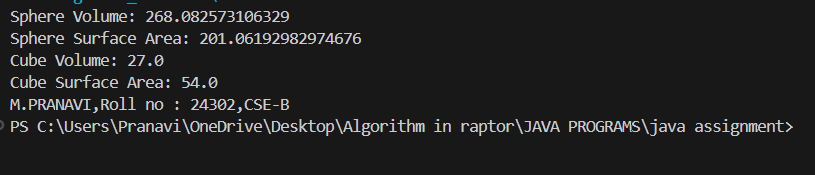
System.out.println("Cube Surface Area: " + c.calculateSurfaceArea());

System.out.println("M.PRANAVI,Roll no : 24302,CSE-B");

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Wrong datatype entered. 2. Object not defined. | 1. Enter the correct datatype i.e double instead of int. 2. Enter the correct object and if not create new one. |

**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:**

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 ();

    }

}

**CODE:**

interface A {

void Method();

}

class B {

public void Method() {

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

}

}

class C extends B implements A {

}

class Main2 {

public static void main(String[] args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

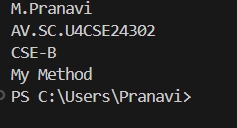
A a = new C();

a.Method();

}

}

**OUTPUT:**



**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.

19. Does below code compile successfully? If not, why?

interface A

{

    int i = 111;

}

class B implements A

{

    void methodB()

    {

        i = 222;

    }

}

**ANSWER:**

1. In interface A, the variable I is simply public, static and final. This means that I is is a constant and its value cannot be changed.
2. Class B implements the interface A, but it tries to modify the value of I insde the method method(). Since I is final (as part of the implicit behaviour of interface variables), it cannot be reassigned after it has been initialized.
3. The statement I = 222, inside method() is illegal because I is final, and final variables cannot be reassigned after they are initialized.

**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:**

interface Shape {

double getperimeter();

}

class rectangle implements Shape{

public int l;

public int b;

public rectangle(int l, int b) {

this.l = l;

this.b = b;

}

public double getperimeter(){

return (2\*(l+b));

}

}

class circle implements Shape {

public int r;

public circle(int r) {

this.r = r;

}

public double getperimeter() {

return (2\*3.14\*r);

}

}

class triangle implements Shape {

public int a;

public int b;

public int c;

public triangle(int a, int b, int c) {

this.a = a;

this.b = b;

this.c = c;

}

public double getperimeter() {

return (a + b + c);

}

}

public class Shapetest1 {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

Shape r = new rectangle(3,6);

System.out.println("The perimeter of the given Rectangle: " + r.getperimeter());

Shape c = new circle(4);

System.out.println("The perimeter of the given circle is: " + c.getperimeter());

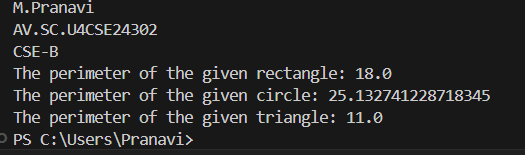
Shape t = new triangle(2,4,5);

System.out.println("The perimeter of the given triangle: " + t.getperimeter());

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Subclasses are not implementing the super class. | 1. Give class implements instead of extends because the class is interface. |

**IMPORTANT POINTS:**

1. While creating objects for the subclasses, we first declare the super class name and then give the subclass name like we did in the above example.
2. We declared method called get perimeter and override it for the three subclasses.
3. While declaring the superclass as interface, for extending the subclasses, we use the ‘interface’ keyword.

**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();

}

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;

}

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;

}

public double bounus() {

return (salary\*0.13);

}

public void manageproject() {

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;

}

public double bonus() {

return (salary\*0.12);

}

public void manageproject() {

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

}

}

public class Employee1test {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

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

m.performancereport();

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

m.manageproject();

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

d.performancereport();

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

d.manageproject();

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

p.performancereport();

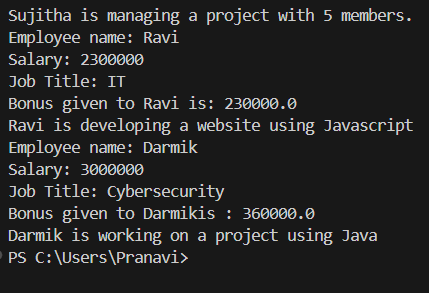
System.out.println("Bonus given to Darmikis : " + p.bonus());

p.manageproject();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Not declaring the variable. 2. Error while printing the statement. | 1. Declare the variable once the constructor is created. 2. Give the plus sign while printing a variable. |

**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("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

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

s.addgrade(84);

s.addgrade(98);

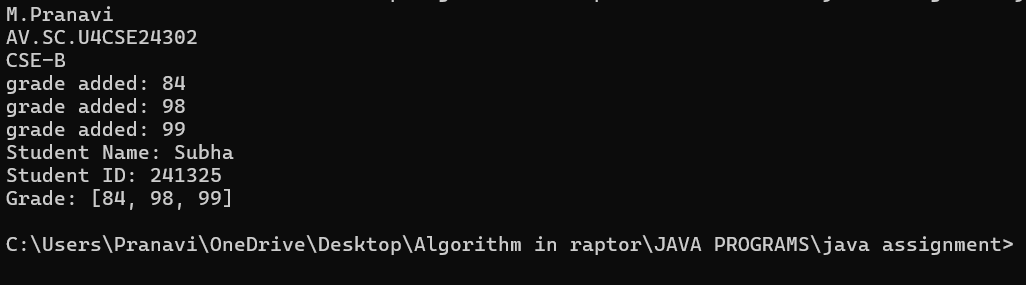
s.addgrade(99);

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.

**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:**

public class bankaccount {

public int balance;

public bankaccount(int balance) {

this.balance = balance;

}

public int deposit(int amount) {

balance = balance + amount;

return balance;

}

}

class savingsaccount extends bankaccount {

public savingsaccount(int balance) {

super(balance);

}

public void withdraw(int withdraw) {

int fee = 100;

int total = withdraw + fee;

if (balance >= total) {

balance = balance-total;

System.out.println("Withdrawing the amount: " + total);

} else {

System.out.println("Enter valid amount.");

}

}

}

class checkingaccount extends bankaccount {

checkingaccount(int balance) {

super(balance);

}

public void withdraw(int withdraw) {

int fee = 150;

int total = withdraw + fee;

if (balance >= total) {

balance = balance-total;

System.out.println("Withdrawing the amount: " + total);

} else {

System.out.println("Enter valid amount.");

}

}

}

public class banktest {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

bankaccount ba = new bankaccount(450000);

savingsaccount sa = new savingsaccount(450000);

checkingaccount ca = new checkingaccount(450000);

System.out.println("depositing the amount: " + ba.deposit(3000));

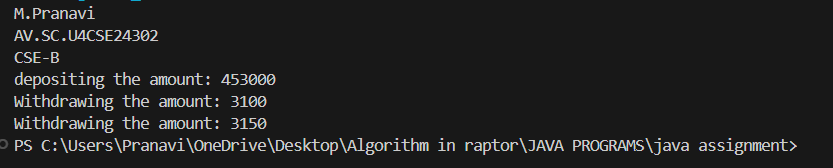
sa.withdraw(3000);

ca.withdraw(3000);

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Int is not defined for sa. 2. Variable amount is not used. | 1. Initialize the variable int in sa. 2. Declare the int amount in ca. |

**IMPORTANT POINTS:**

1. Here we have created instances, of bankaccount, savingsaccount, and checking account with an intial balance of 450000.
2. The deposit method adds 3000 to the balance, updating it to 453000.
3. The withdrawal method in savingsaccount and checkingaccount subtracts the withdrawal amount along with the respective fees(100 for savings and 150 for checking).
4. After each withdrawal, the balance is updated.

**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:**

interface Bird {

abstract void fly();

abstract void makesound();

}

class Eagle implements Bird {

public void fly() {

System.out.println("The Eagle flies up so high that no other bird can reach its height.");

}

public void makesound() {

System.out.println("Eagles' sound ranges upto 2kHz.");

}

}

class Hawk implements Bird {

public void fly() {

System.out.println("Hawks are the most violent predators and fly very fast.");

}

public void makesound() {

System.out.println("Hawks sound like kee-eeee-arr");

}

}

public class Birdtest {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

Bird e = new Eagle();

Bird h = new Hawk();

e.fly();

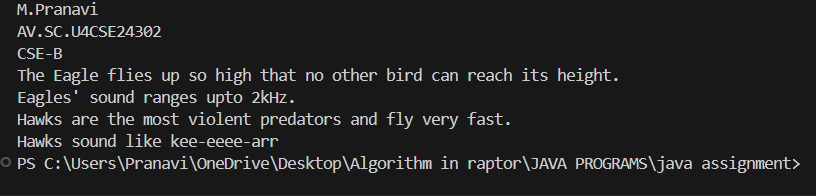
e.makesound();

h.fly();

h.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.

**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 {

abstract void play();

}

class Football implements Playable {

public void play() {

System.out.println("some people play football in the Ground.");

}

}

class Volleyball implements Playable {

public void play() {

System.out.println("some people play volleyball in the Stadium.");

}

}

class Basketball implements Playable {

public void play() {

System.out.println("some people play basket ball in the Rain.");

}

}

public class playtest {

public static void main(String[]args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

Playable f = new Football();

Playable v = new Volleyball();

Playable b = new Basketball();

f.play();

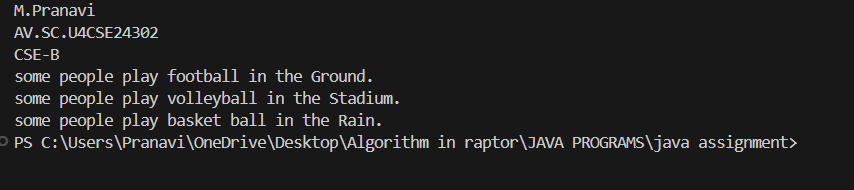
v.play();

b.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.
3. 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;

import java.util.List;

interface Account {

void deposit(int amount);

void withdraw(int amount);

double calculateintrest();

void balance();

}

class savingsaccount implements Account {

public double balance;

public double intrestrate;

public savingsaccount(double balance, double intrestrate) {

this.balance = balance;

this.intrestrate = intrestrate;

}

public void deposit(int amount) {

if (amount > 0) {

balance += amount;

System.out.println("Deposited: " + amount);

} else {

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

}

}

public void withdraw(int amount) {

if (amount > 0 && amount <= balance) {

balance -=amount;

System.out.println("Withdrew: " + amount);

} else {

System.out.println("Invalid withdrawal amount.");

}

}

public double calculateintrest() {

return balance\*(intrestrate/100);

}

public void balance() {

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

}

public double getintrestrate() {

return intrestrate;

}

}

class currentaccount implements Account {

public double balance;

public currentaccount(double balance) {

this.balance = balance;

}

public void deposit(int amount) {

if (amount > 0) {

balance += amount;

System.out.println("Deposited: " + amount);

} else {

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

}

}

public void withdraw(int amount) {

if (amount > 0 && amount <= balance) {

balance -=amount;

System.out.println("Withdrew: " + amount);

} else {

System.out.println("Invalid withdrawal amount.");

}

}

public double calculateintrest() {

return 0;

}

public void balance() {

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

}

}

class Bank {

public List<Account> accounts;

public Bank() {

accounts = new ArrayList<>();

}

public void addAccount(Account account) {

accounts.add(account);

}

public void allaccounts() {

for (Account account: accounts) {

account.balance();

}

}

}

public class Accounttest {

public static void main(String[] args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

Bank b = new Bank();

Account sa = new savingsaccount(5000, 5);

Account ca = new currentaccount(2000);

b.addAccount(sa);

b.addAccount(ca);

sa.deposit(1000);

sa.withdraw(200);

ca.deposit(500);

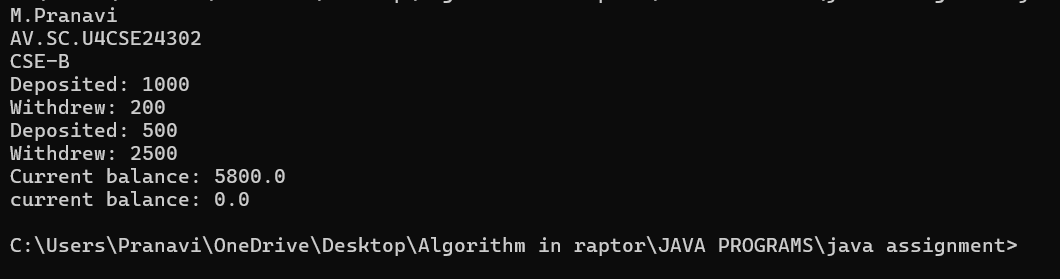
ca.withdraw(2500);

b.allaccounts();

}

}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. List not defined. 2. Integer not defined. | 1. Import‘Import java.util.List’ 2. Give the integer a value. |

**IMPORTANT POINTS:**

1. Here, the account interface is used to define common methods for both savings account and currentaccount classes, ensuring that both types of accounts follow the same contract
2. The bank class manages the overall bank operations and maintains a collection of account objects.
3. Here different types of accounts implement specific behavior.

. savings account calculates intrest based on its balance and intrest rate.

**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.

1. An array is a collection of similar types of data.
2. If we want to store the names of 100 people then we can create an array of the string type that can store 100 names.
3. e.g code: String[] array = new String[100];
4. we can declare an array by writing (dataType[] arrayName;)
5. Here dataType – it can be of any primitive data types like int, char, double, byte, etc… and arrayName is an identifier.
6. E.g double[] data; here data is an array that can hold values of type double.

Array Initialization:

1. Array initialization can be done in the following ways:
2. Static Initialization: (Array size is implicitly defined by the number of elements we provide.
3. E.g int[] arr = {1,2,3,4,5 };
4. Dynamic initialization: (Array size is defined explicitly)
5. E.g int [] arr = new int[5];

Multidimensional Arrays:

1. A multidimensional array is an array of arrays. Each element of a multidimensional array is an array itself.
2. E.g int[][] a = new int[3][4];
3. Here, we have created a multidimensional array named a. it is a 2-dimensional array, that can hold a maximum of 12 elements.
4. An example for 3-dimensional array looks like:
5. String[][][] data = new String[3][4][2];
6. Here, data is a 3d array that can hold a maximum of 24 (3\*4\*2) elements of type ‘String’.

**CODE:**

import java.util.Scanner;

public class Matrix {

public static void main(String[]args) {

Scanner input = new Scanner(System.in);

System.out.print("Enter no.of rows: ");

int r = input.nextInt();

System.out.print("Enter no.of columns: ");

int c = input.nextInt();

int[][] mat1 = new int[r][c];

int[][] mat2 = new int[r][c];

int[][] smat = new int[r][c];

int[][] pmat = new int[r][c];

System.out.print("Enter elements for mat1: ");

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

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

mat1[i][j] = input.nextInt();

}

}

System.out.println("Enter elements for mat2: ");

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

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

mat2[i][j] = input.nextInt();

}

}

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

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

smat[i][j] = mat1[i][j] + mat2[i][j];

}

}

if (r == c) {

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

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

pmat[i][j] = 0;

for (int k = 0; k<c; k++) {

pmat[i][j] += mat1[i][j] \* mat2[k][j];

} } } }

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

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

displayMatrix(mat1);

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

displayMatrix(mat2);

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

displayMatrix(smat);

if (r == c) {

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

displayMatrix(pmat);

} else {

System.out.println("not possible.");

}

input.close();

}

public static void displayMatrix(int[][] matrix) {

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

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

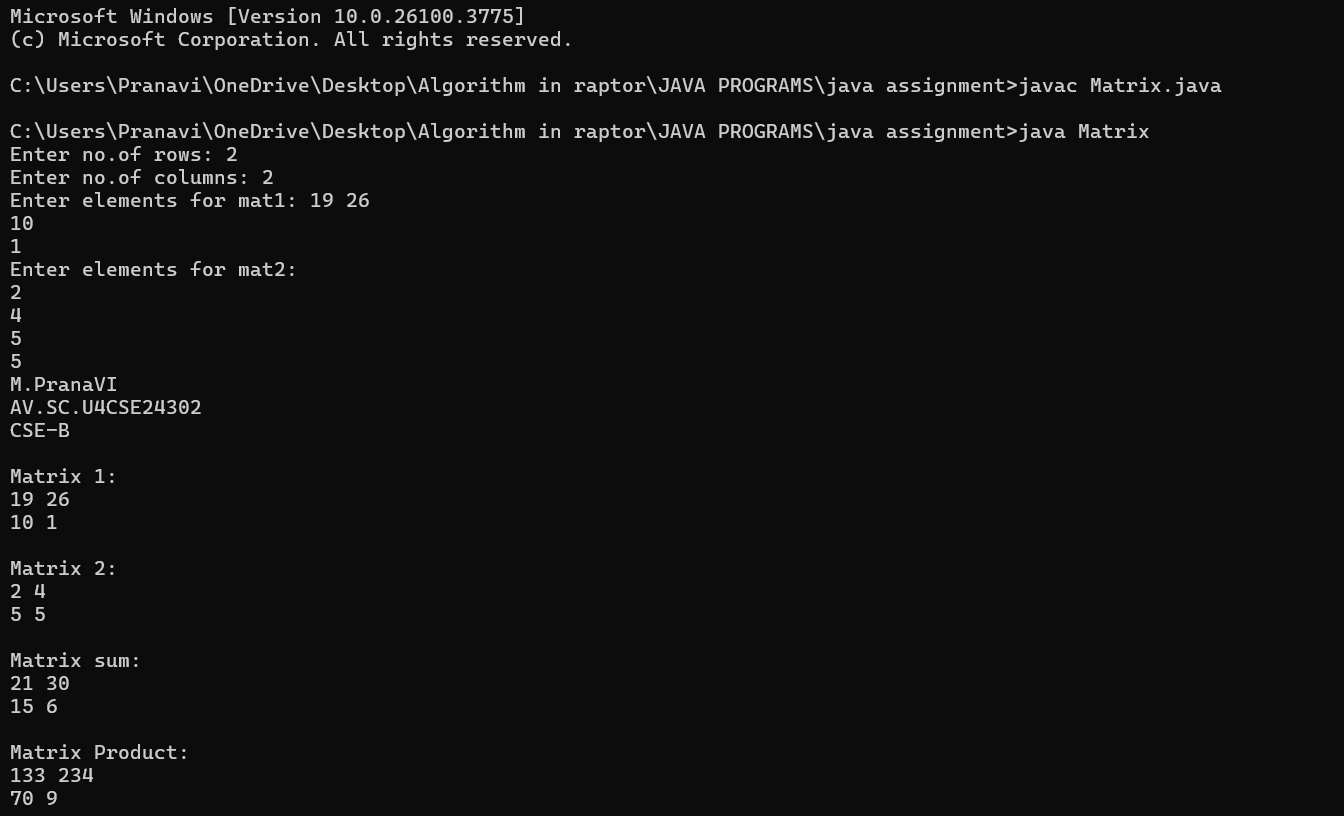
System.out.print(matrix[i][j] + " ");

}

System.out.println(); }

}}

**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.

**QUESTION – 28**

1. **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.

**CODE:**

import java.util.Scanner;

abstract class GeometricObject {

public String color = "white";

public boolean filled;

public GeometricObject(String color, boolean filled) {

this.color = color;

this.filled = filled;

}

public String getColor() {

return color;

}

public boolean getFilled() {

return filled;

}

public String details() {

return "Color: " + color + ", Filled: " + filled;

}

}

class Triangle extends GeometricObject {

public int s1;

public int s2;

public int s3;

public Triangle(int s1, int s2, int s3, String color, boolean filled) {

super(color, filled);

this.s1 = s1;

this.s2 = s2;

this.s3 = s3;

}

public int getS1() {

return s1;

}

public int getS2() {

return s2;

}

public int getS3() {

return s3;

}

public int perimeter() {

return s1 + s2 + s3;

}

public double area() {

double s = (s1 + s2 + s3) / 2.0;

return Math.sqrt(s \* (s - s1) \* (s - s2) \* (s - s3));

}

}

public class GeometricObject2 {

public static void main(String[] args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

Scanner input = new Scanner(System.in);

System.out.print("Enter s1: ");

int s1 = input.nextInt();

System.out.print("Enter s2: ");

int s2 = input.nextInt();

System.out.print("Enter s3: ");

int s3 = input.nextInt();

input.nextLine();

System.out.print("Enter color: ");

String color = input.nextLine();

System.out.print("Is the triangle filled (true/false): ");

boolean filled = input.nextBoolean();

Triangle t = new Triangle(s1, s2, s3, color, filled);

System.out.println("\nTriangle Details:");

System.out.println("Area: " + t.area());

System.out.println("Perimeter: " + t.perimeter());

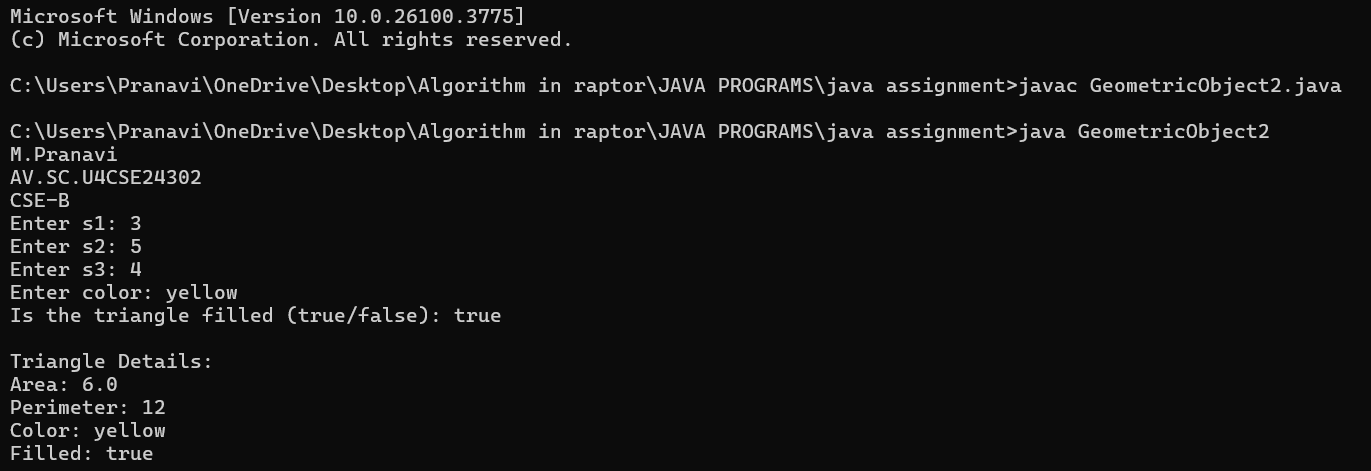
System.out.println("Color: " + t.getColor());

System.out.println("Filled: " + t.getFilled());

input.close();

}

}



**ERROR TABLE:**

|  |  |
| --- | --- |
| CCode Error | CCode rectification |
| 1. Method filled is not defined. 2. Error in giving input to color. | 1. Define the method through an object. 2. Use input.nextLine() instead of input.toString(). |

**IMPORTANT POINTS:**

1. Here, we declared the main class as an abstract method geometric method.
2. The perimeter is calculated using a method called getperimeter().
3. In the area method, we are using math.sqrt() method where it defines that it is going to calculate the area of the triangle.
4. We are using the Boolean data type to see whether the triangle is filled with color or not.

**PROGRAM – 30**

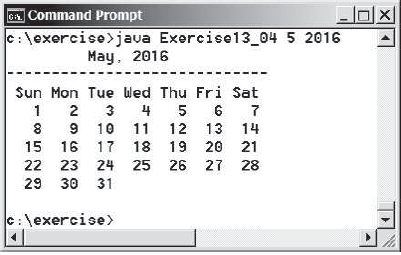
**AIM:**

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.



**CODE:**

import java.util.Calendar;

import java.util.GregorianCalendar;

public class Exercise {

public static void main(String[] args) {

System.out.println("M.Pranavi");

System.out.println("AV.SC.U4CSE24302");

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

if (args.length != 2) {

System.out.println("Usage: java Exercise13\_04 <month> <year>");

return;

}

int month = Integer.parseInt(args[0]);

int year = Integer.parseInt(args[1]);

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

int startDay = calendar.get(Calendar.DAY\_OF\_WEEK);

int numberOfDaysInMonth = calendar.getActualMaximum(Calendar.DAY\_OF\_MONTH);

printMonthTitle(year, month);

printMonthBody(startDay, numberOfDaysInMonth);

}

public static void printMonthTitle(int year, int month) {

String[] monthNames = {

"",

"January", "February", "March", "April", "May", "June",

"July", "August", "September", "October", "November", "December"

};

System.out.println(" " + monthNames[month] + " " + year);

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

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

}

public static void printMonthBody(int startDay, int numberOfDaysInMonth) {

int currentDay = 1;

for (int i = Calendar.SUNDAY; i < startDay; i++) {

System.out.print(" ");

}

for (int i = startDay; currentDay <= numberOfDaysInMonth; i++) {

System.out.printf("%4d", currentDay);

if (i % 7 == 0) {

System.out.println();

}

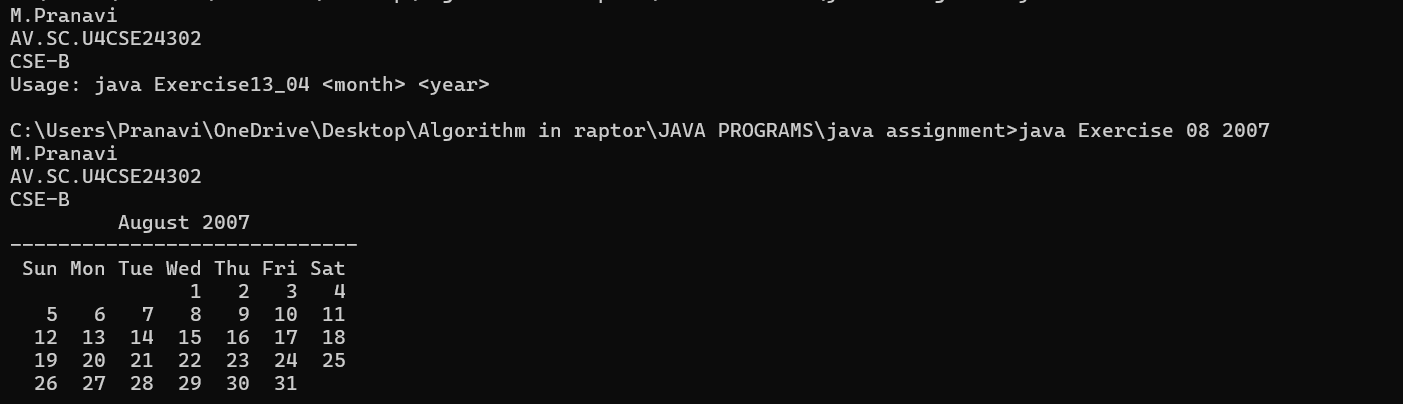
currentDay++;

}

System.out.println();

}}

**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| CCode Error | CCode rectification |
| 1. Method filled is not defined. 2. Error in giving input to color. | 1. Define the method through an object. 2. Use input.nextLine() instead of input.toString(). |

**IMPORTANT POINTS:**

1. These values are passed as strings in the args[] array and must be parsed to integers.
2. Here, the Gregorian calendar is a subclass of calendar.
3. The get(Calendar.DAY\_OF\_WEEK) returns the day of the week for the 1st of the month.
4. getActualMaximum(Calendar.DAY\_OF\_MONTH) gives the number of days in the month(handles leap years automatically).
5. To display the name of the month correctly, a String[]array is used.
6. A duplicate “ ” at index 0 is added to align month numbers(1-12) with array index.