**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**LAB MANUAL**



**Department of Computer Science Engineering**

**Amrita School of Computing**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Verified By: Name : U. Nischitha**

**Roll No: AV.SC.U4CSE24339**

**Section:CSE-B[CSE-D]**

|  |  |  |
| --- | --- | --- |
| S.No. | Week | Program |
| 1. | Week1 | a.Installationof java  b.printing student details |
| 2. | Week2 | a.Finding simple interest  b.Finding factorial  c.Conversion of Celsius to Fahrenheit and vice versa  d.Fibonacci  e.Finding area of rectangle  f.Finding area of triangleusing heron’s formula |
| 3. | Week3 |  |
| 4. | Week4 |  |
| 5. | Week5 |  |

**WEEK-01**

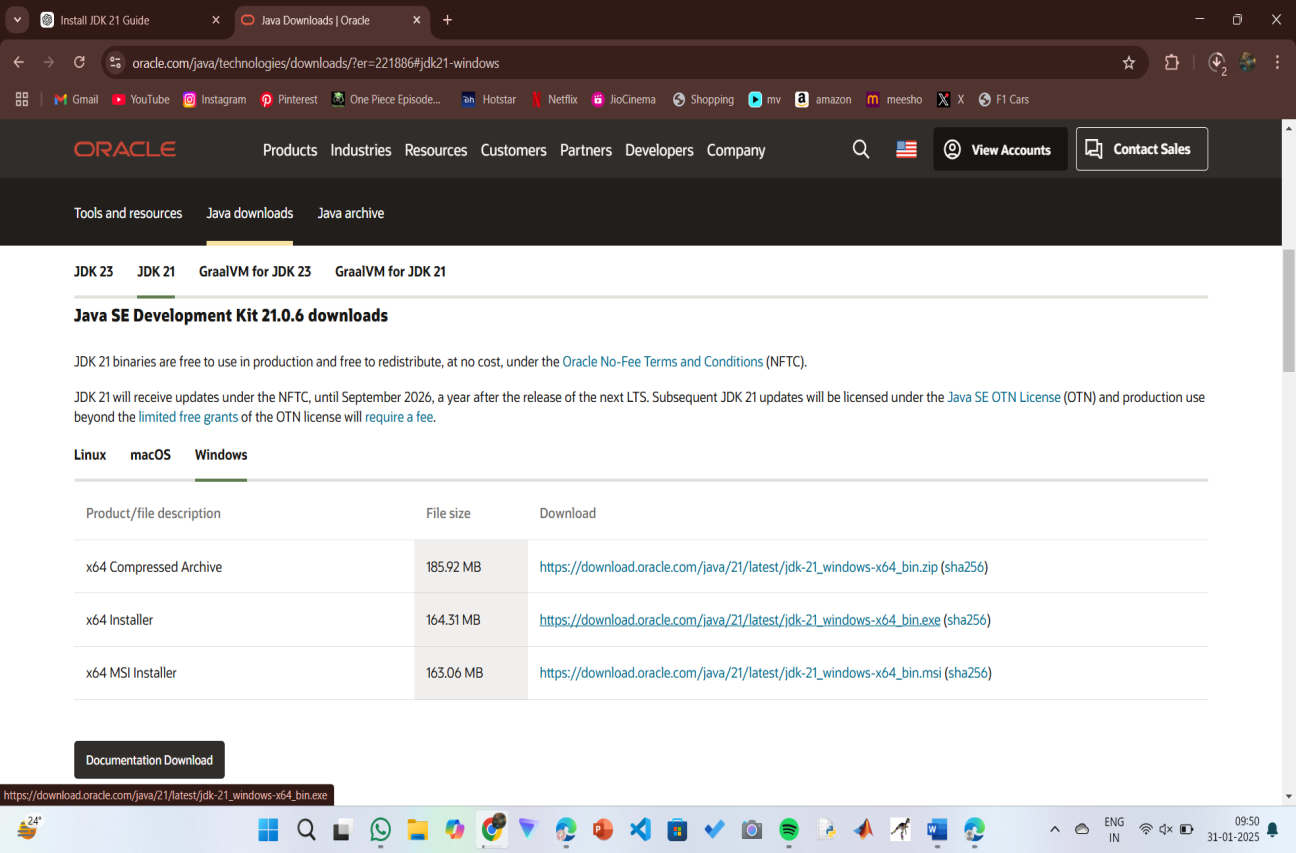
**PROGRAM-1:**

**Aim**: Download and install java software.

**Procedure:**

Step1. Download JDK21:

* Open the web browser and go to Oracle JDK Downloads page.
* ii .Scroll down to the Java SE Development Kit 21 section.
* Select` the Windows x64 Installer version.
* Click on Download, then Wait for the download to complete**.**



Step 2:Installation of JDK 21:

* Locate the downloaded jdk-21\_windows-x64\_bin.exe file.
* Double-click to launch the installer.
* Click Next on the setup wizard.
* Choose the installation path (default is C:\Program Files\Java\jdk-21).
* Click Next, then click Install.
* Wait for the installation to complete.
* vii.Click Close once the installation is finished.



**Step 3: Setting up the path:**

* Go to “Windows C” Drive in This PC.
* Choose Program Files, select Java, then JDK 21, then select Bin.
* Select and copy the path at the address bar.



**Step 4: Open System Properties:**

* Press Windows + R, type sysdm.cpl , and click Ok-
* The System Properties window will open.
* Navigate to the Advanced tab.
* Click on Environment Variables at the bottom.



**Step 5:Setting up of JAVA\_HOME:**

* Under System Variables, click New.
* Set the Variable name as JAVA\_HOME.
* Set Variable value as C:\Program Files\Java\jdk-21 (or your installation path).
* Click on OK.



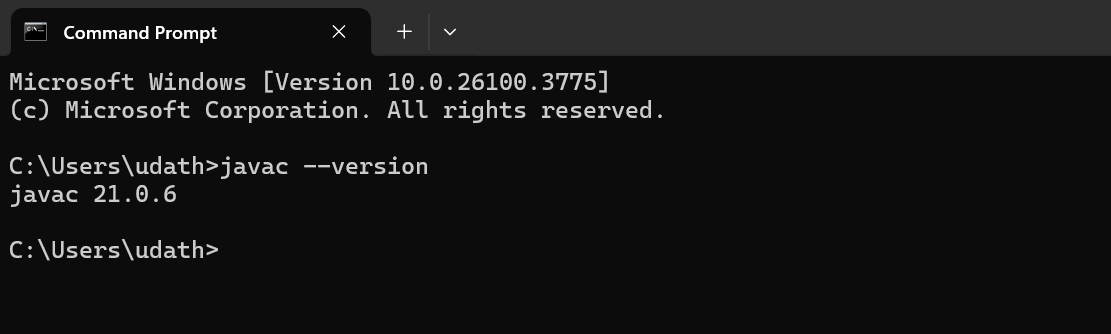
**Step 6: Update the PATH Variable:**

* In System Variables, find Path and click Edit.
* Click New and add: C:\Program Files\Java\jdk-21\bin
* Click OK to save.



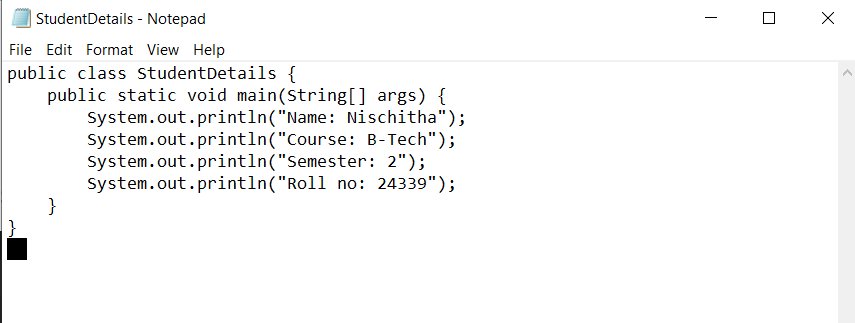
**Step 7:Verify Installation1:**

* Open Command Prompt.
* Type the following command: **java --version** and press Enter.
* To check the java compiler type: **javac –version.**

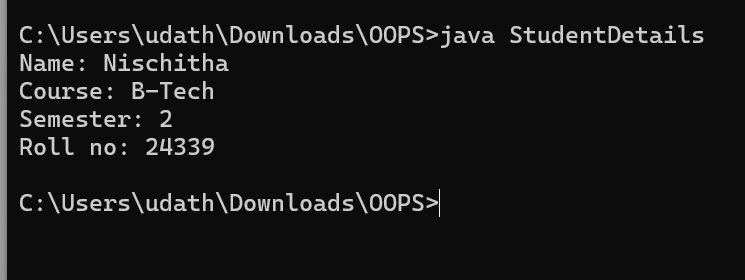


**PROGRAM-2:**

**Aim:** Write a Java Program that prints Name, Roll No, Section of a student.

****

**OUTPUT:**



ERROR TABLE:

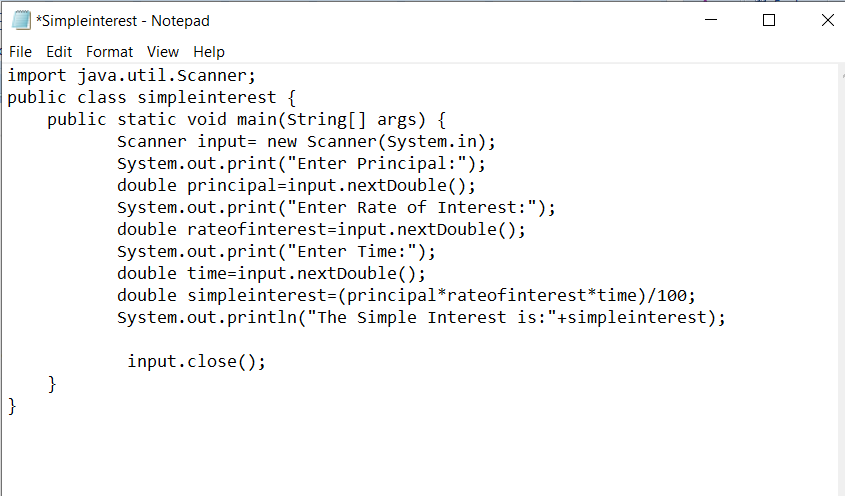
|  |  |
| --- | --- |
| **ERROR IN CODE** | **ERROR RECTIFICATION** |
| 1. ; is missing | 1. code is rectified by keeping a (;) in every place where it is missing. |

**WEEK-2:**

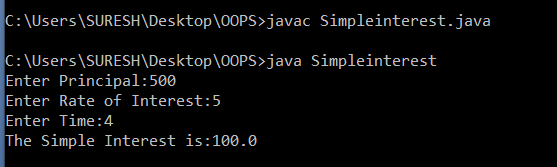
**PROGRAM-1:**

**AIM**: Write a java program to find the simple interest where all the inputs are given by the user.

CODE:



**OUTPUT:-**



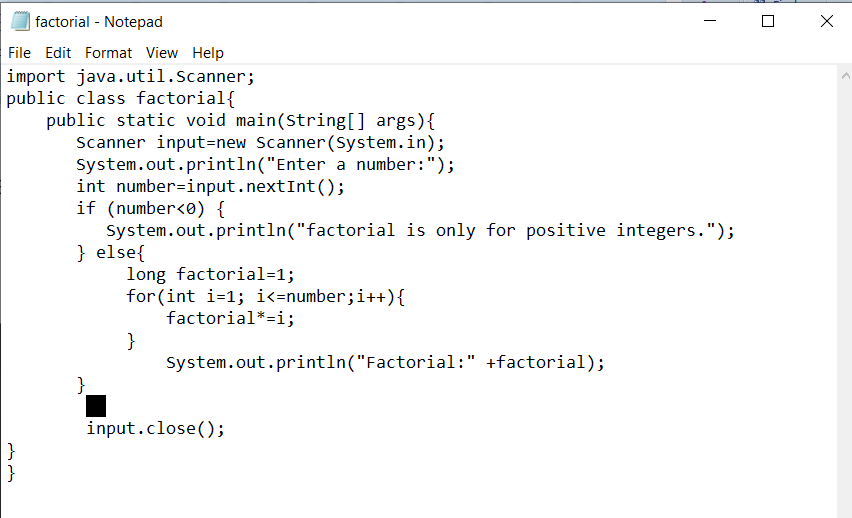
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
|  | ERROR | ERROR RECTIFICATION |
| 1) | ; is expected after input.close() | Corrected by writing a ; after input.close() |
| 2) | Error in the line double principal=input.next.double(); | Corrected by removing “. “after the next. |

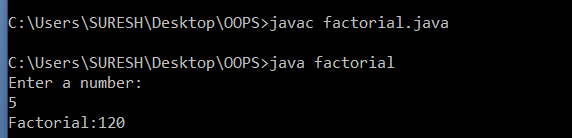
**PROGRAM-2:**

**AIM**: Write a java program to find the factorial of a number where all the inputs are given by the user.

**CODE:**



**OUTPUT:**

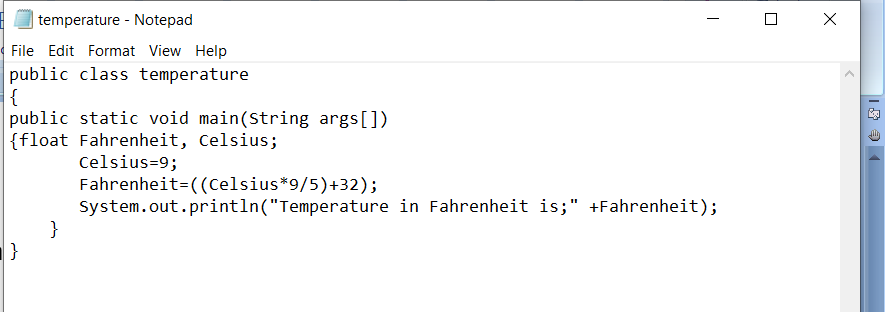


**ERROR TABLE:**

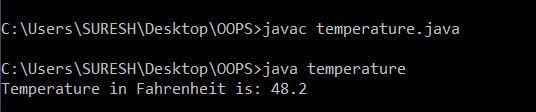
|  |  |  |
| --- | --- | --- |
|  | ERROR | ERROR RECTIFICATION |
| 1) | writing small “S” in place of ”S” in system.out.println() | code is rectified by keeping capital “S” |
| 2) | “ is missed in the print statement | Corrected by keeping “ in the end of print statement. |

**PROGRAM-3:**

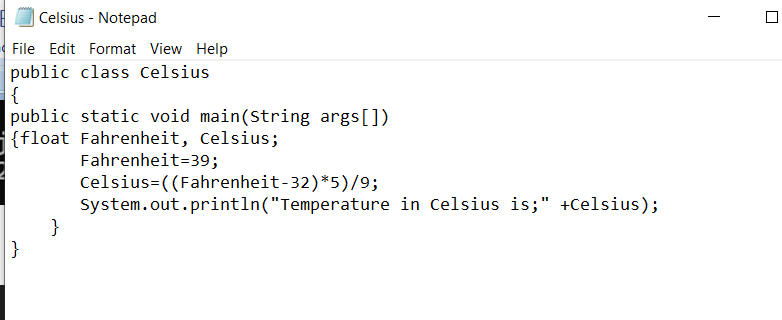
**AIM**: Write a java program to convert the temperature from Celsius to Fahrenheit and Celsius to Fahrenheit.



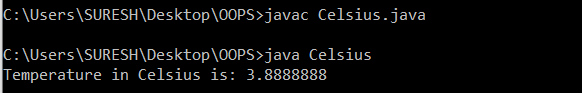
**OUTPUT:-**



**CODE2:**



**OUTPUT:**

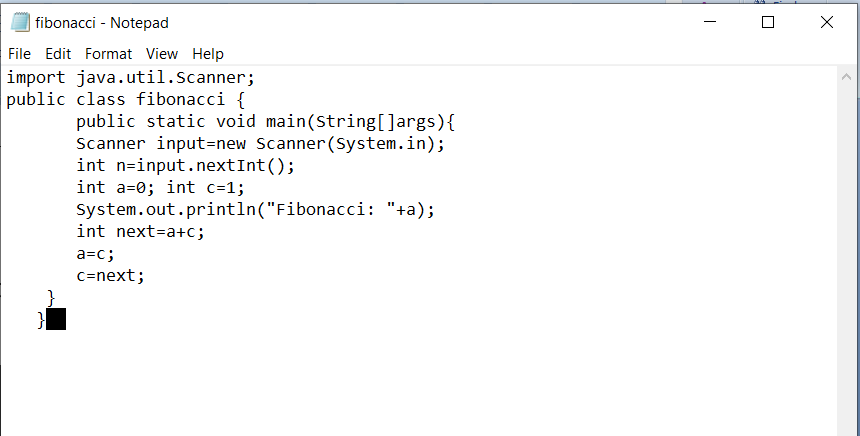
****

**ERROR TABLE:**

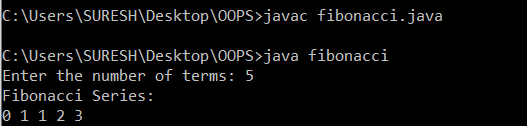
|  |  |  |
| --- | --- | --- |
| S.NO | ERROR | ERROR RECTIFICATION |
| 1. | Logical error  Missed \* after Celsius in code 1 | Corrected by keeping a \* operator in the appropriate place |

**PROGRAM – 4:**

AIM: Write a java program to find the Fibonacci series of a given number where all the inputs are taken from the user.

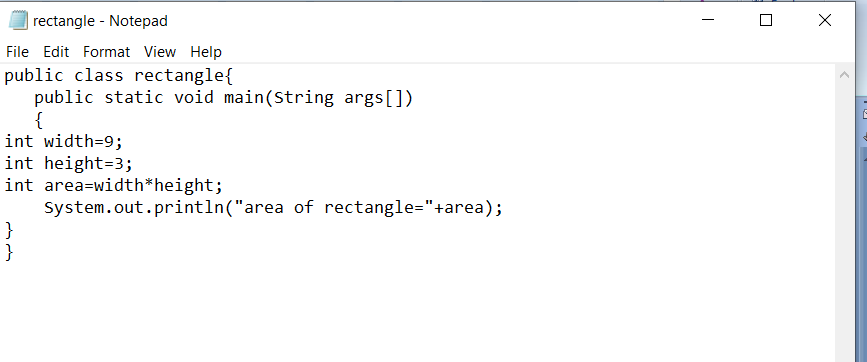


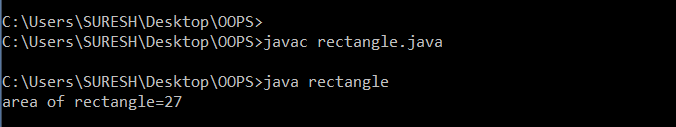
**OUTPUT:**

****

PROGRAM-5:

AIM: Area of a rectangle:

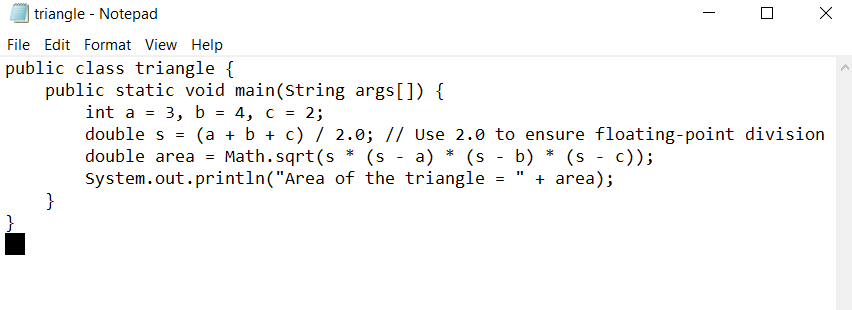
CODE:OUTPUT:



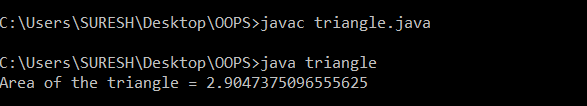
PROGRAM-6

AIM:Area of triangle using Heron’s formula

CODE:



OUTPUT:



ERROR TABLE:

|  |  |  |
| --- | --- | --- |
|  | ERROR | ERROR RECTIFICATION |
| 1. | Missed ) and ; in the formula line in code. | Rectified by placing )and ; at end of code. |

**WEEK-03**

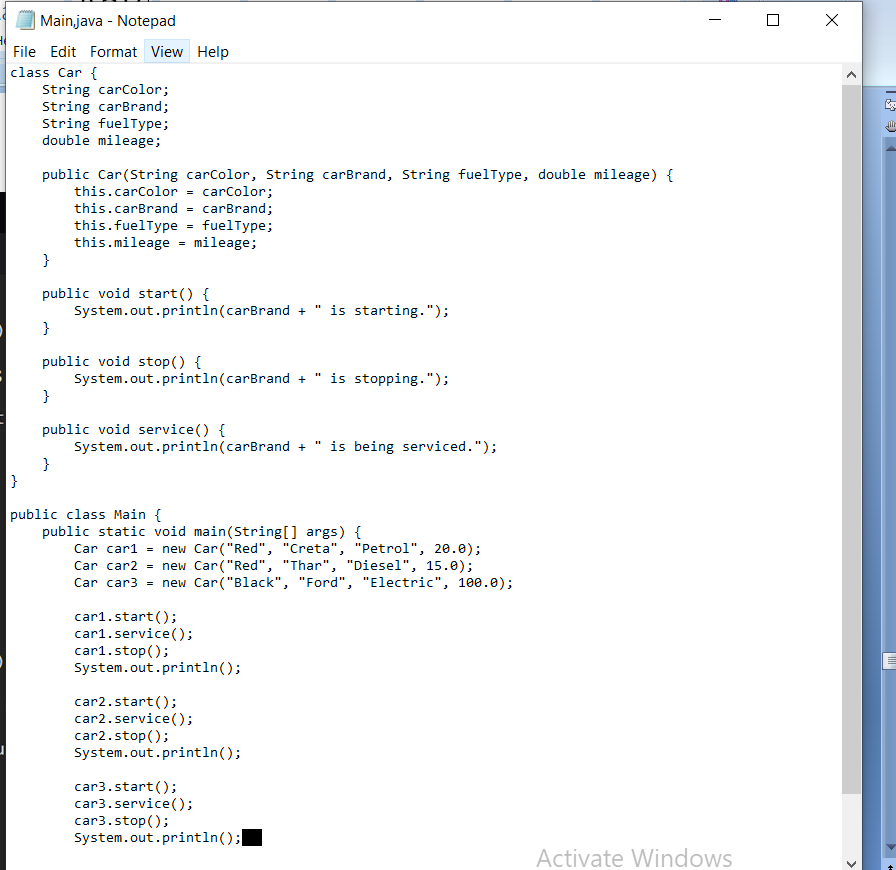
**PROGRAM-1:**

**AIM:**

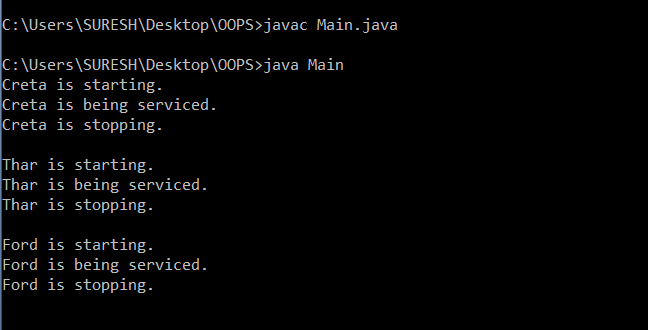
To write a java program with the following instructions:

* create a class with name car.
* Create four attributes named(car colour ,car brand ,fuel type ,milage).
* Create three methods named (start ,stop .service).
* Create three objects named (c1,c2,c3).
* Create a constructer with parameters(car colour ,car brand , fuel type ,milage).

**CODE:**

****

**OUTPUT:**

****

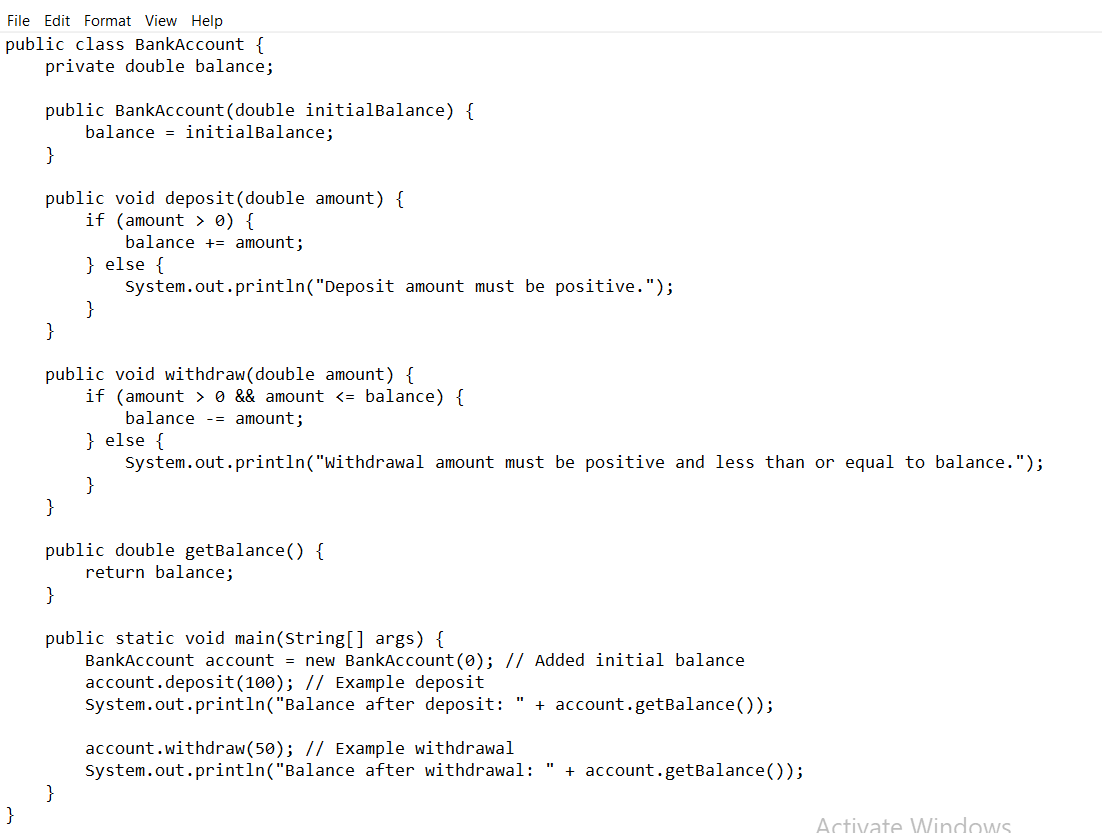
**CLASS DIAGRAM:**

|  |
| --- |
| Class car |
| +Car\_color: string  +Car\_brand: string  + fuel\_type:string  +mileage:int |
| +void start()  + void service()  +void stop() |

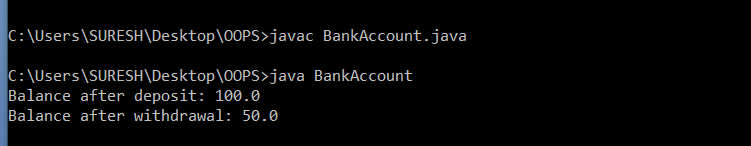
**PROGRAM-2:**

**AIM:** Create a class named bank account with methods deposit and withdraw where the deposit method should accept the parameter and when the method is called the deposit amount should be added to current balance**.** In addition to that when a withdraw method is called it has to verify whether the withdraw amount is less than current balance. If not display a message saying that “there are insufficient funds”. Use the constructer to display details of costumer(costumer name, account number, IFSC ,branch).Also create 2 costumer objects(c1,c2).

**CODE:**

****

**OUTPUT:**

****

**CLASS DIAGRAM:**

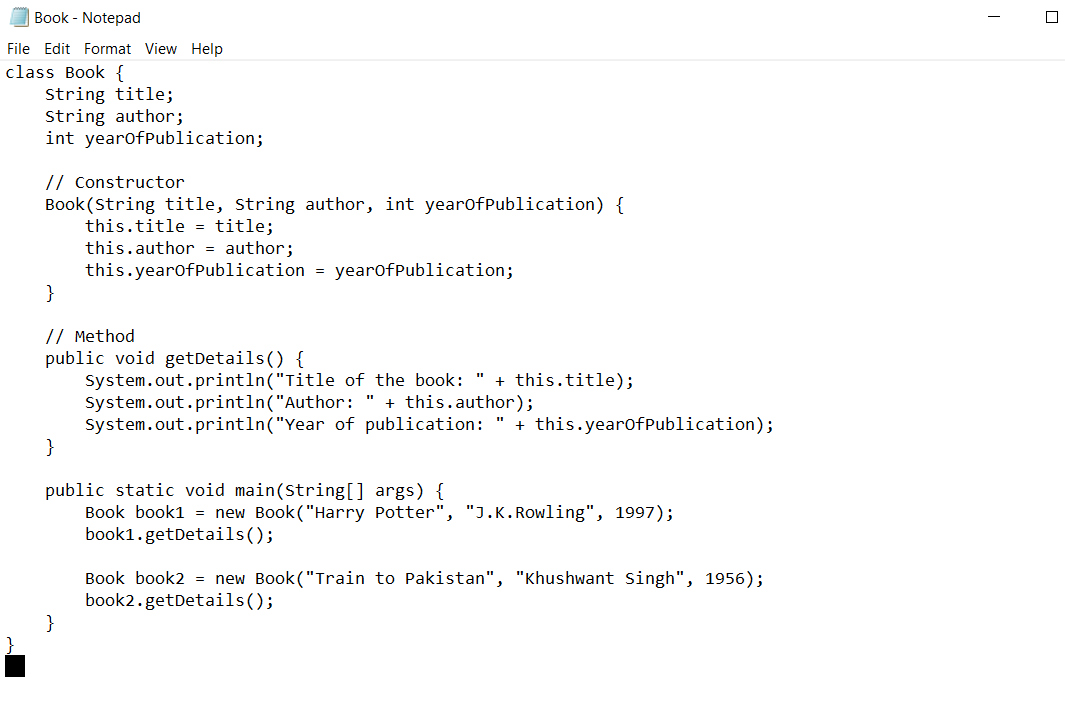
|  |
| --- |
| Class BankAccount |
| -balance:double |
| +void deposit()  +void withdrawal() |

**WEEK-04**

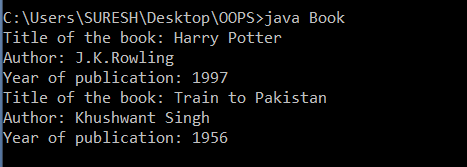
**PROGRAM-1:**

**AIM:** Write a java program with class named” book”. The class should contain various attributes such as “Title of the book”, “Author”, Year of publications”, it should also contain a constructor with parameter which initializes “Title of the book”, “Author”, “Year of publication”. Create a method which displays the details of the book. Title of the book (), Author (), Year of publication ().Display the details of two book, by creating 2 objects.

**CODE:**

****

**OUTPUT:**

****

**ERROR:**

|  |  |
| --- | --- |
| ERROR | RECTIFICTION |
| Not using capital ‘s’in string in constructor | By using capital s |
| Not keeping ‘.’ after this in constuctor | by making changes required |

While defining a method we should also define a function to call that method

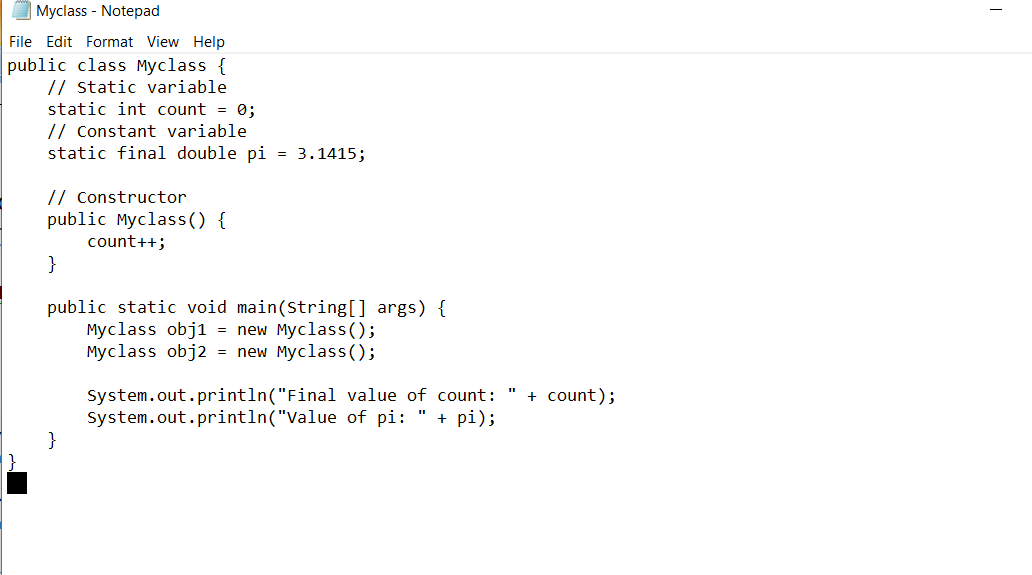
**CLASS DIAGRAM:**

|  |
| --- |
| class Book |
| Title: String  Author: String  Year of publication: int |
| + Book(title: String,                    Author: String;                    Year of publication: int )     + getDetails( ): void |

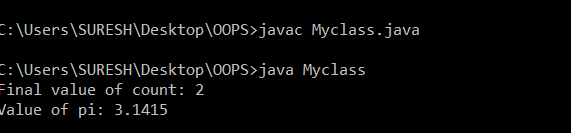
**PROGRAM-2:**

**AIM:**create a java program with class named “Myclass” with a static variable count of “int” type, initialised to “zero” and a constant variable “pi” of type “double” initialized to 3.1415 as attributes of that class . define a constructor for “Myclass” is created finally print the final values “count” and “pi” variables**.**

**CODE:**

****

**OUTPUT:**

****

**ERROR:**

|  |  |
| --- | --- |
| ERROR | RECTIFICATION |
| Not using; after creating new object | Using; after creating new object |

Static variable-belongs to class, can be change during runtime we use static as key word

Static final(constant variable)- value of variable(static) cannot be changed.

**CLASS DIAGRAM:**

|  |
| --- |
| Class Myclass |
| Count: int  Pi: double |
| + Myclass( )    +main(args:String[]):void |

Week – 5

PROGRAM-1

Aim: To create a Java Program of a calculator using the operations including addition Subtraction, multiplication and division using multilevel inheritance & dis Play the desired out Put

CODE:

class SimpleCalculator{

//attributes, objects

int a;

int b;

//initialization

public static void add(int a, int b) {

System.out.println(a+ " + " + b + " = " + (a+b));

}

public static void diff(int a, int b) {

System.out.println(a+ " - " + b +" = " + (a-b));

}

}

class AdvCalculator extends SimpleCalculator {

// Initialization

public static void mul(int a, int b) {

System.out.println(a + " \* " + b + " = " + (a \* b));

}

}

class UltiCalculator extends AdvCalculator{

//initialization

public static void div(int a, int b) {

if(b == 0) {

System.out.println("Denominator should not be a zero ");

}

else{

System.out.println(a+ "/" + b +"=" + (a/b));

}

}

}

class Calc{

public static void main(String[] args) {

UltiCalculator u = new UltiCalculator();

System.out.println("The calculated values are: ");

u.add(5,5);

u.diff(6,5);

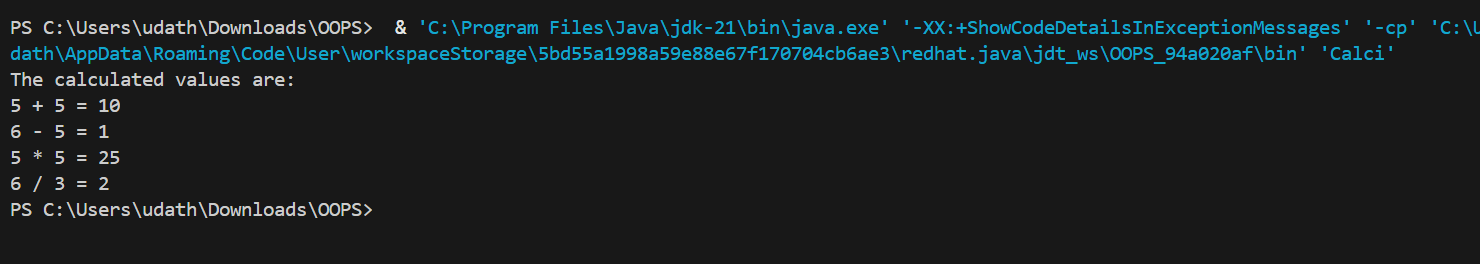
u.mul(5,5);

u.div(6,3);

} // end of the main function

} // end of the class

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| 1. | error: unclosed string literal System.out.println(a+ " - " + b +" = + (a-b)); | Add a '”' after the ‘=’. |
| 2. | error: '(' or '[' expected UltiCalculator u = new UltiCalculator; | Add a '()' after class name. |

**Class Diagram:**

|  |
| --- |
| **SimpleCalculator** |
| a : double  b : double |
| + add (a,b) : void  + diff (a,b) : void |

|  |
| --- |
| **AdvCalculator** |
| + mul (a,b) : void |

|  |
| --- |
| **UltiCalculator** |
| + div (a,b) : void |

|  |
| --- |
| **Calc** |
| + main(String[]) |

**Concepts to be known:**

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-2

**Aim:** A vehicle rental rental company wants to develop a system that maintains information about different types of vechicles available for rent the company rents out cars and bikes, and they need a program to store details about each vehicle, such as brand and speed( should be in super class)

1. cars should have an additional property: no.of doors
2. Bikes should have a property indicating whether they have gears or not.
3. The system should also include a function to display details about each vehicle and indicate when a vehicle is starting.
4. Every class should have a constructor

Question:

1. Which oops concept is used in the above program
2. If the company decides to add a new type of vehicle, Truck, how would you modify the program?
3. Truck should include an additional property capacity (in tons)
4. Create a showTruckdetails() method to display the truck’s capacity.
5. Write a constructor for Truck that initializes all properties
6. Implement the truck class and update the main method to create a Truck object and also create an object for car and bike sub classes Finally, display the details.

**CODE:**

class Vehicle {

    String brand;

    double speed;

    public Vehicle(String brand, double speed) {

        this.brand = brand;

        this.speed = speed;

    }

    public void displayDetails() {

        System.out.println("Brand: " + brand);

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

    }

    public void start() {

        System.out.println(brand + " vehicle is starting...");

}

}

class Car extends Vehicle {

    int numberOfDoors;

    int seatingCapacity;

    public Car(String brand, double speed, int numberOfDoors, int seatingCapacity) {

        super(brand, speed);

        this.numberOfDoors = numberOfDoors;

        this.seatingCapacity = seatingCapacity;

    }

    public void displayCarDetails() {

        super.displayDetails();

        System.out.println("Number of doors: " + numberOfDoors);

        System.out.println("Seating capacity: " + seatingCapacity);

    }

    public void startCar() {

        super.start();

        System.out.println("Car is ready to go!");

    }

}

class Bike extends Vehicle {

    boolean hasGears;

    public Bike(String brand, double speed, boolean hasGears) {

        super(brand, speed);

        this.hasGears = hasGears;

    }

    public void displayBikeDetails() {

        super.displayDetails();

        System.out.println("Has gears: " + (hasGears ? "Yes" : "No"));

    }

    public void startBike() {

        super.start();

        System.out.println("Bike is ready to go!");

    }

}

class Truck extends Vehicle {

    double cargoCapacity;

    public Truck(String brand, double speed, double cargoCapacity) {

        super(brand, speed);

        this.cargoCapacity = cargoCapacity;

    }

    public void displayTruckDetails() {

        super.displayDetails();

        System.out.println("Cargo capacity: " + cargoCapacity + " tons");

    }

    public void startTruck() {

        super.start();

        System.out.println("Truck is ready to go!");

    }

}

public class VehicleRentalSystem {

    public static void main(String[] args) {

        Car car = new Car("Toyota", 150, 4, 5);

        Bike bike = new Bike("Yamaha", 120, true);

        Truck truck = new Truck("Volvo", 90, 10);

        System.out.println("Car Details:");

        car.displayCarDetails();

        car.startCar();

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

        bike.displayBikeDetails();

        bike.startBike();

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

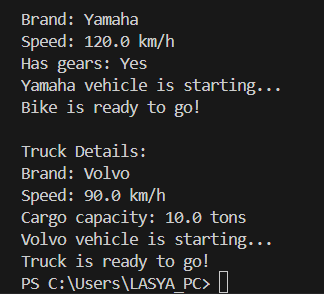
        truck.displayTruckDetails();

        truck.startTruck();

    }

}

**OUTPUT:**

****

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| **1.** | **error: Declaring two super classes inside the same file.** | **Make two separate files to save the two super classes** |
| **2.** | **error: Not declaring the variable using ‘this’ keyword inside the constructor.** | **Declare the variable using this keyword to run the program.** |

**Concepts to be known:**

**1. a constructor helps in initializing an object that doesn't exist.**

**2. a method performs functions on pre-constructed or already developed objects.**

**3. a double method can represent more decimal point numbers than float method.**

**Class Diagram:**

|  |
| --- |
| **Vehicle** |
| **- brand: string**  **- speed: double** |
| **+Vehicle()**  **+ displayDetails() : void**  **+ start() : void** |

|  |
| --- |
| **Car** |
| **- noOfDoors: int**  **- seatingCapacity: int** |
| **+Car()**  **+ displayCarDetails() : void**  **+ startCar() : void** |

|  |
| --- |
| **Bike** |
| **- hasGears: Boolean** |
| **+Bike()**  **+ displayBikeDetails() : void**  **+ startBike() : void** |

|  |
| --- |
| **Truck** |
| **- cargoCapacity: Boolean** |
| **+ Truck()**  **+displayTruckDetails() : void**  **+ startTruck() : void** |

**Answer:**

**The oops concepts used in the above program are:**

**Inheritance, encapsulation, polymorphism, abstraction.**

**To add a new vehicle type truck we need to create a truck class that will:**

* **Include an additional property capacity (in tons).**
* **Implement a showTruckdetials() method to display the truck's capacity.**
* **Implement a constructor for the truck class to initialize all its properties.**

**Week – 6**

**Aim: Write a Java Program to create a Vehicle class with a method display(). Override this method in the Car subclass. Print car model, brand, petrol type, car color and provide the information about the car.**

**Code :**

**class Vehicle {**

**String brand;**

**String petrolType;**

**String color;**

**public void display() {System.out.println("This is a vehicle.");**

**}**

**}**

**class Car extends Vehicle {**

**Car(String brand, String petrolType, String color) {**

**this.brand = brand;**

**this.petrolType = petrolType;**

**this.color = color;**

**}**

**public void display() {**

**System.out.println("Car Details:");**

**System.out.println("Brand: " + brand);**

**System.out.println("Petrol Type: " + petrolType);**

**System.out.println("Color: " + color);**

**}**

**}**

**public class Info {**

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

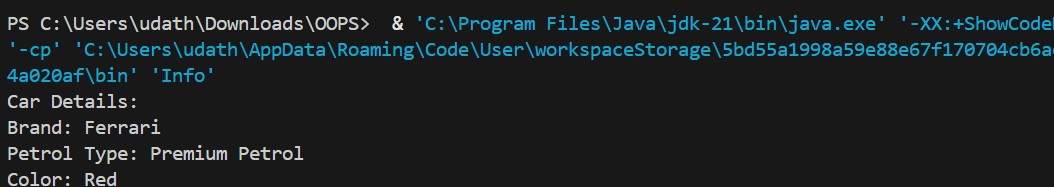
**Vehicle myCar = new Car("Ferrari", "Premium Petrol", "Red");**

**myCar.display();**

**}**

**}**

**Output :**

****

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| **1.** | **error: class Main is public, should be declared in a file named Main.java**  **public class Main {** | **Save the file name as the name of the main class** |

**Class Diagram:**

|  |
| --- |
| **Vehicle** |
| **+ brand : String**  **+ petrolType : String**  **+ color : String** |
| **+ display() : void** |

|  |
| --- |
| **Car** |
| **Car(String brand, String petrolType, String color)** |

**Concepts to be known:**

**1. We use the concept of method overriding where the names of the methods in the different classes. The method of the parent class is overridden by the method of the child class**

**Aim: A college is developing an automated admission system that verifies students’ eligibility for under-graduation and post-graduation. Each program has different eligibility criteria base on the percentage of students in their provided qualifications**

* **Ug requires 60%**
* **PG requires 70%**

**Code :**

**class Student{**

**String name;**

**double percentage;**

**Student(String name, double percentage){**

**this.name = name;**

**this.percentage = percentage;**

**}**

**public void Eligibility(){**

**System.out.println(name + " must meet the general admission criteria");**

**}**

**}**

**class UG extends Student{**

**UG(String name, double percentage){**

**super(name, percentage);**

**}**

**public void Eligibility(){**

**if (percentage>59){**

**System.out.println(name + " is elligible for UG admission");**

**}**

**else {**

**System.out.println(name + " is Not elligible for UG admission");**

**}**

**}**

**}**

**class PG extends Student{**

**PG(String name, double percentage){**

**super(name, percentage);**

**}**

**public void Eligibility(){**

**if (percentage>69){**

**System.out.println(name + " is elligible for PG admission");**

**}**

**else {**

**System.out.println(name + " is Not elligible for PG admission");**

**}**

**}**

**}**

**public class Admission {**

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

**UG ug = new UG("Hari", 69);**

**PG pg = new PG("Geetha", 59);**

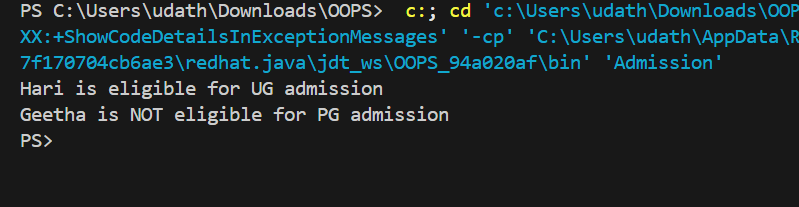
**ug.Eligibility();**

**pg.Eligibility();**

**}**

**}**

**Output :**

****

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| **1.** | **error: ';' expected System.out.println(name + " is elligible for UG admission")** | **Add a ‘;’ after the print statement.** |

**Class Diagram:**

|  |
| --- |
| **UG** |
| **UG(String name, double percentage)** |

|  |
| --- |
| **Student** |
| **+ name : String**  **+ percentage : double** |
| **+ Student(String name, double percentage): void**  **+ Eligibility(): void** |

|  |
| --- |
| **PG** |
| **PG(String name, double percentage)** |

**Concepts to be known:**

**1. The variables once declared in the super class need not be declared twice in any of the sub classes.**

**2. super keyword is used in sub classes to access the methods of super classes, they are basically the reverse of overriding.**

**Aim: To create a Java Program with class named “my class” with a Static Variable Count int type and initialize to 0 and A Constant Variable "pi" of type double initialized to 3.1415 has attributes of that class. Now defi a Constructor for my class that increments the Count Variable each time an object of my class is created. Finaly Print the final values of count.**

**CODE:**

**class AddCalculator{**

**AddCalculator(){**

**System.out.println("This is a calculator");**

**}**

**public int add(int a, int b){**

**return a+b;**

**}**

**public double add(double a, double b){**

**return a+b;**

**}**

**public int add(int a, int b, int c){**

**return a+b+c;**

**}**

**}**

**public class AddCalc{**

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

**AddCalculator calc = new AddCalculator();**

**System.out.println("The sum of the 2 numbers: 7 and 18 is "+ calc.add(7, 18));**

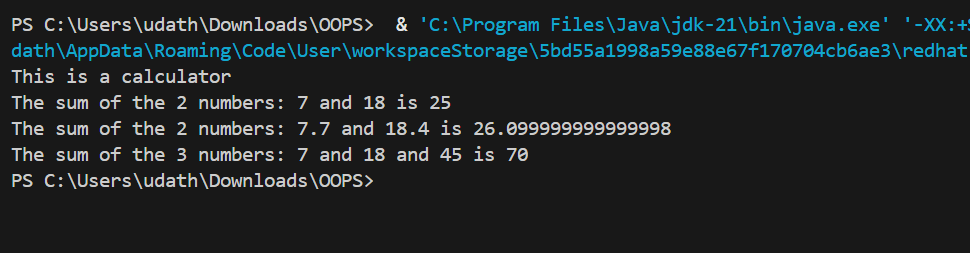
**System.out.println("The sum of the 2 numbers: 7.7 and 18.4 is "+ calc.add(7.7, 18.4));**

**System.out.println("The sum of the 3 numbers: 7 and 18 and 45 is "+ calc.add(7, 18, 45));**

**}**

**}**

**OUTPUT:**

****

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| **1.** | **error: Main method not defined** | **Add public static void main (String [] args){** |
| **2.** | **error: ';' return type is not expected** | **Change return type from int to double in its case.** |

**Class Diagram:**

|  |
| --- |
| **AddCalculator** |
| **+ AddCalculator ()**  **+ add(int a, int b) : int**  **+ add(double a, double b) : double**  **+ add(int a, int b, int c) : int** |

**Concepts to be known:**

**1. We use the concept of method overloading where the names of the methods in the same class are same but the parameters are given different.**

**Aim: Write a Java Program and create a Shape class with a method calcArea(). That is overloaded for different shapes like square and rectangle. Create a sub class circle that overrides the calcArea() for a circle.**

**Code :**

**class Shape{**

**int calcArea(int a){**

**return a\*a;**

**}**

**int calcArea(int b, int h){**

**return b\*h;**

**}**

**}**

**class Circle extends Shape{**

**double r;**

**double pi = 3.141592653589793;**

**Circle(double r){**

**this.r = r;**

**}**

**double calcArea(double r){**

**return pi\*r\*r;**

**}**

**}**

**public class AreaCalc {**

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

**Circle c = new Circle(7);**

**System.out.println("The area of circle is " + c.calcArea(7.7));**

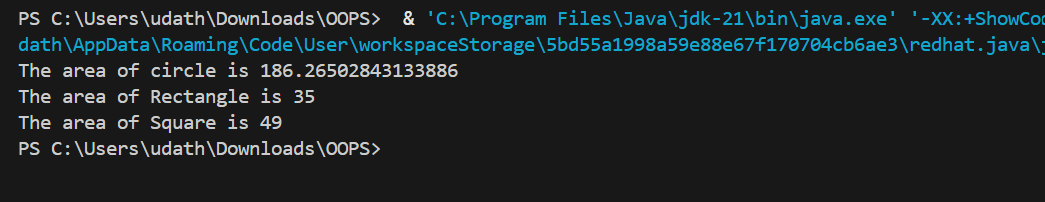
**System.out.println("The area of Rectangle is " + c.calcArea(5, 7));**

**System.out.println("The area of Square is " + c.calcArea(7));**

**}**

**}**

**Output :**

****

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| **1.** | **error: invalid method declaration; return type required**  **calcArea(int a){** | **Enter the return type as per required. Here it is int** |
| **2.** | **error: incompatible types: possible lossy conversion from double to int**  **return pi\*r\*r;** | **For calculating area of circle, we need to give return type double.** |

**Class Diagram:**

|  |
| --- |
| **Circle** |
| **+ r : double**  **+ pi : double** |
| **+ calcArea(int r) : double** |

|  |
| --- |
| **Shape** |
| **+ calcArea(int a) : int**  **+ calcArea(int b, int h) : int** |

**Concepts to be known:**

**1. We use the concept of method overloading to calculate the area of square and rectangle in the parent class Shape.**

**2 we use method overriding in the child class Circle to calculate it’s area.**

**Week-7**

**Aim:** Write a java program to create an abstract class Animal with an abstract method called sound( ).create a 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**

**abstract class Animal {**

**// Abstract method**

**public abstract void sound();**

**// You can also have regular methods in abstract classes**

**public void info() {**

**System.out.println("All animals make different sounds.");**

**}**

**}**

**// Subclass Lion**

**class Lion extends Animal {**

**@Override**

**public void sound() {**

**System.out.println("Lion roars!");**

**}**

**}**

**// Subclass Tiger**

**class Tiger extends Animal {**

**@Override**

**public void sound() {**

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

**}**

**}**

**public class AnimalSoundTest {**

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

**// Create objects of Lion and Tiger**

**Animal lion = new Lion();**

**Animal tiger = new Tiger();**

**// Call methods**

**lion.info();**

**lion.sound();**

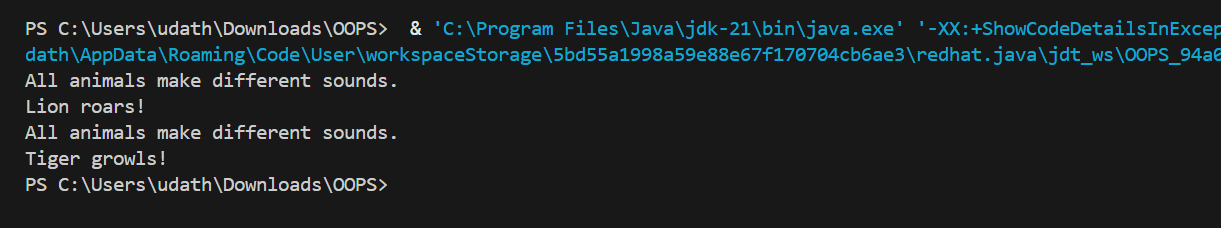
**tiger.info();**

**tiger.sound();**

**}**

**}**

**OUTPUT:**

****

**CLASS DIAGRAM:**

|  |
| --- |
| **Animal**  **+ sound(): void** |

|  |
| --- |
| **Lion**  **+ sound (): void** |

|  |
| --- |
| **Tiger**    **+ sound(): void** |

**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.** |

**Concepts to be known:**

**1.**We use the concept of abstraction and writing abstract classes.

2.we also use method overrriding concept.

3.Here we are using the heirarchial inheritance.

**Aim:**

Write a java program to create an abstract class shape 3D with abstract methods. Calculate Volume( )and calculate Surface area

( ).Create Subclasses Sphere and cube that extend the Shape 3D class and implement the respective method and to calculate the volume and surface area of each shape.

**CODE:**

**// Abstract base class**

**abstract class Shape3D {**

**// Abstract methods**

**public abstract double calculateVolume();**

**public abstract double calculateSurfaceArea();**

**}**

**// Subclass for Sphere**

**class Sphere extends Shape3D {**

**double radius;**

**Sphere(double 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);**

**}**

**}**

**// Subclass for Cube**

**class Cube extends Shape3D {**

**double side;**

**Cube(double side) {**

**this.side = side;**

**}**

**@Override**

**public double calculateVolume() {**

**return Math.pow(side, 3);**

**}**

**@Override**

**public double calculateSurfaceArea() {**

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

**}**

**}**

**// Main class**

**public class Shape3DTest {**

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

**Shape3D sphere = new Sphere(5); // radius = 5**

**Shape3D cube = new Cube(4); // side = 4**

**System.out.println("Sphere:");**

**System.out.printf("Volume: %.2f\n", sphere.calculateVolume());**

**System.out.printf("Surface Area: %.2f\n", sphere.calculateSurfaceArea());**

**System.out.println("\nCube:");**

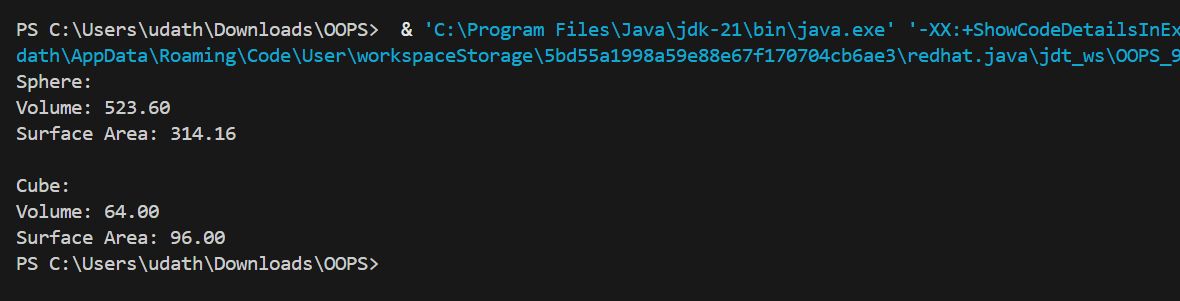
**System.out.printf("Volume: %.2f\n", cube.calculateVolume());**

**System.out.printf("Surface Area: %.2f\n", cube.calculateSurfaceArea());**

**}**

**}**

**OUTPUT:**

****

**CLASS DIAGRAM:**

|  |
| --- |
| **Shape3D**  **+calculatevolume(): double**  **+calculatesurfacearea(): double** |

**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.** |

**Concepts to be known:**

**1.**We use the concept of abstraction and writing abstract classes.

2.we also use method overrriding concept.

3. Abstract classes and methods help us to declare the methods without declaring the return type in them.

**Aim:**

**Write a java program using an abstract class to define a method for pattern printing.**

* **Create an abstract class named PatternPrinter with:an abstract method printPattern(int n)and a concrete method to display the pattern title**
* **Implement two subclasses:**
* **StarPattern – prints a right-angled triangle of stars**
* **NumberPattern – prints a right-angled triangle of increasing numbers**
* **In the main() method, create objects of both subclasses and print the patterns for a given number of rows.**

**Example Output for n = 5:**

**Star Pattern**

**\***

**\* \***

**\* \* \***

**\* \* \* \***

**\* \* \* \* \***

**Number Pattern**

**1**

**1 2**

**1 2 3**

**1 2 3 4**

**1 2 3 4 5**

**CODE:**

**abstract class PatternPrinter {**

**int rows;**

**PatternPrinter(int rows) {**

**this.rows = rows;**

**}**

**abstract void printPattern();**

**void displayTitle(String title) {**

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

**}**

**}**

**class StarPattern extends PatternPrinter {**

**StarPattern(int rows) {**

**super(rows);**

**}**

**void printPattern() {**

**for (int i = 1; i <= rows; i++) {**

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

**System.out.print("\* ");**

**}**

**System.out.println();**

**}**

**}**

**}**

**class NumberPattern extends PatternPrinter {**

**NumberPattern(int rows) {**

**super(rows);**

**}**

**void printPattern() {**

**for (int i = 1; i <= rows; i++) {**

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

**System.out.print(j + " ");**

**}**

**System.out.println();**

**}**

**}**

**}**

**public class Teststar {**

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

**System.out.println("Name:U.Nischitha,CSE-B[CSE-D] , AV.SC.U4CSE24339");**

**int numberOfRows = 5;**

**PatternPrinter star = new StarPattern(numberOfRows);**

**star.displayTitle("Star Pattern");**

**star.printPattern();**

**PatternPrinter number = new NumberPattern(numberOfRows);**

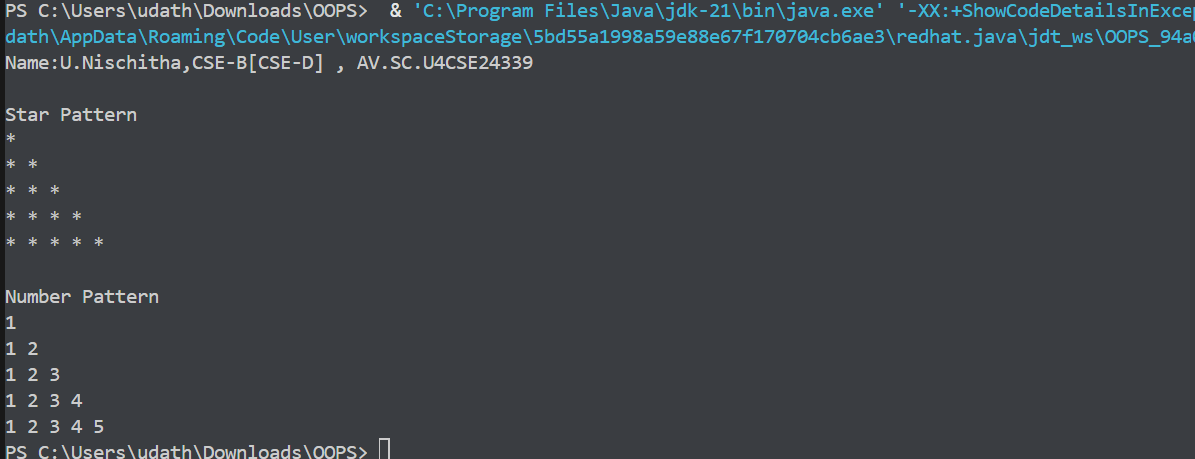
**number.displayTitle("Number Pattern");**

**number.printPattern();**

**}**

**}**

**OUTPUT:**

****

**CLASS DAIGRAM:**

**PatternPrinter**

**- rows: int**

**+displayTitle()**

**+printPattern()**

**StarPattern**

**+printPattern()**

**NumberPattern**

**+printPattern()**

**ERROR TABLE:**

|  |  |
| --- | --- |
| **CODE ERROR:**   1. **Class name and file name should match** 2. **Subclass doesn’t override abstract method** | **ERROR RECTIFICATION**   1. **Save file as main.java**   **2)implement printpattern()in all subclasses** |

**Concepts to be known:**

1. Use abstract classes to enforce a common structure for pattern printing.
2. PatternPrinter is the abstract class defining the common template.
3. Subclasses (StarPattern, NumberPattern) provide specific implementations.
4. displayTitle() is a concrete method shared by all subclasses.

**Week-8**

**PROGRAM-1**

**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 {

    double length;

    double width;

    public Rectangle(double length, double width) {

        this.length = length;

        this.width = width;

    }

    public double getPerimeter() {

        return 2 \* (length + width);

    }

}

class Circle implements Shape {

    double radius;

    public Circle(double radius) {

        this.radius = radius;

    }

    public double getPerimeter() {

        return 2 \* Math.PI \* radius;

    }

}

class Triangle implements Shape {

    double a;

    double b;

    double c;

   public Triangle(double a, double b, double c) {

        this.a = a;

        this.b = b;

        this.c = c;

    }

    public double getPerimeter() {

        return a + b + c;

    } }

public class Perimeter {

    public static void main(String[] args) {

        Shape rectangle = new Rectangle(5.7, 10.3);

        Shape circle = new Circle(7);

        Shape triangle = new Triangle(3.09, 4.57, 5.87);

                System.out.println("U.Nischitha\nAV.SC.U4CSE24339\nCSE-B");

        System.out.println("Rectangle Perimeter: " + rectangle.getPerimeter());

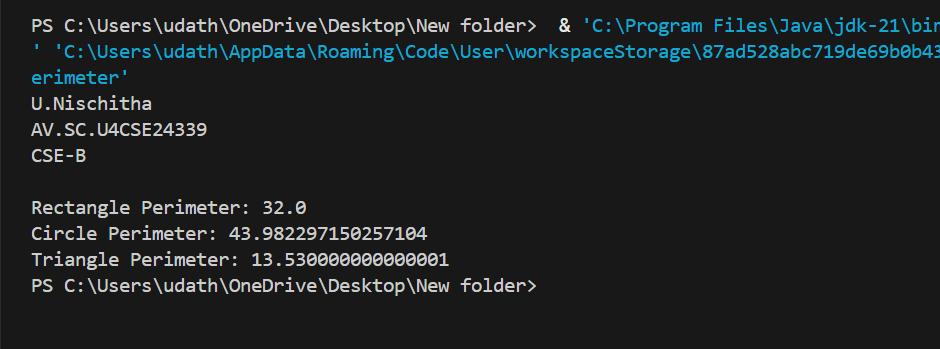
        System.out.println("Circle Perimeter: " + circle.getPerimeter());

        System.out.println("Triangle Perimeter: " + triangle.getPerimeter());

    }

}

**OUTPUT:**

****

**CLASS DIAGRAM:**

**Shapes**

**+ area(): double**

**Triangle**

**- base**

**- height**

**+ area():double**

**Circle**

**-radius**

**Rectangle**

**-length**

**-width**

**+ area(): double**

**ERROR TABLE:**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| System.out.println("The perimeter of the circle is: "+perimeter) | System.out.println("The perimeter of the circle is: "+perimeter); |

**IMPORTANT POINTS:**

Key Points on Interface

1. No Implementation: Interfaces cannot contain method implementations (in many languages), only method signatures.

2. Multiple Inheritance: A class can implement multiple interfaces, providing a way to achieve multiple inheritance.

3. Abstract Methods: All methods in an interface are implicitly abstract (in many languages) and must be implemented by the implementing class.

4. No Constructors: Interfaces cannot have constructors because they cannot be instantiated directly.

**PROGRAM-2**

**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{

    public void play();

}

class FootBall implements Playable {

    public void play(){

        System.out.println("The player is playing Football.");

    }

}

class VolleyBall implements  Playable {

    public void play(){

        System.out.println("The player is playing Volleyball.");

    }

}

class BasketBall implements  Playable {

    public void play(){

        System.out.println("The player is playing Basketball.");

    }

}

class Sports{

    public static void main(String[] args){

        System.out.println("U.Nischitha\nAV.SC.U4CSE24339\nCSE-B");

        FootBall f1 = new FootBall();

        f1.play();

        VolleyBall v1 = new VolleyBall();

        v1.play();

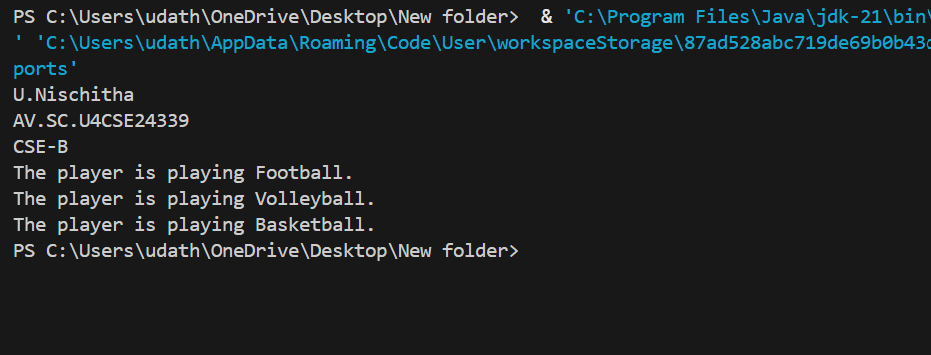
        BasketBall b1 = new BasketBall();

        b1.play();

    }

}

**OUTPUT:**

****

**CLASS DAIGRAM:**

**Interface: Playable**

**+ play(): void**

**FOOTBALL**

**+ play(): void**

**Volleyball**

**+ play(): void**

**Basketball**

**+ play(): void**

**ERROR TABLE:**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| VolleyBall = new VolleyBall();  ^ 1 error | VolleyBall v1 = new VolleyBall(); |

**IMPORTANT POINTS:**

* The program demonstrates the use of interfaces in Java through the Playable interface, which defines a single method play() that must be implemented by any class that uses it.
* Three different sports classes (FootBall, VolleyBall, and BasketBall) implement the Playable interface, each providing its own version of the play() method, showcasing polymorphism.

**PROGRAM-3:**

**AIM:**Write a java program to implements login System using interfaces.

**CODE:**

interface LoginSystem {

boolean login(String id, String pass);

}

class University\_portal implements LoginSystem {

@Override

public boolean login(String id, String pass) {

if (id.equals("Student123") && pass.equals("pass02")) {

System.out.println("Login successful");

return true;

} else {

System.out.println("Invalid credentials");

return false;

}

}

public static void main(String[] args) {

System.out.println("U.Nischitha/n CSE-B/nAV.SC.U4CSE24339");

University\_portal p1 = new University\_portal();

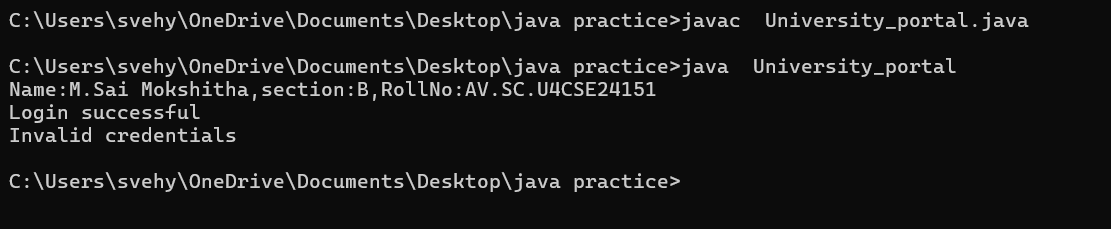
p1.login("Student123", "pass02");

p1.login("Student123", "wrongpass");

}

}

**OUTPUT:**

****

**CLASS DAIGRAM:**

**interface :Login system**

**+ login(String,String): boolean**

**University\_portal**

**+ login(StringString): boolean**

**+ main(String[]): void**

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| **CODE ERROR**   1. **Saving file with the interface name is error** 2. **Removing public will leads to error**  |  | | --- | |  | | **ERROR RECTIFICATION**   1. **save with implement name.** 2. **add public static void main**   **(String[]args)** |

**IMPORTANT POINTS:**

**Interface Implementation:**

University\_potral correctly implements Login System interface

Uses Override annotation for the login() method

**Authentication Logic:**

Hardcoded credentials: id="Student123", password="pass02"

Returns boolean and prints appropriate message.

**Main Method:**

Demonstrates both successful and failed login attempts

Includes student information print statement

**Polymorphism:**

Could create Login System Portal=new University\_portal();

Demonstrates interface-based programming