**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**LAB MANUAL**



**Department of computer science engineering**

**Amrita school of computing**

**Amrita Vishwa Vidyapeetham,**

**Amaravati campus**

**Name: N Jayarudhramurthy**

**Rollno :24234**

**Verified by:**

**1.AIM: To download**

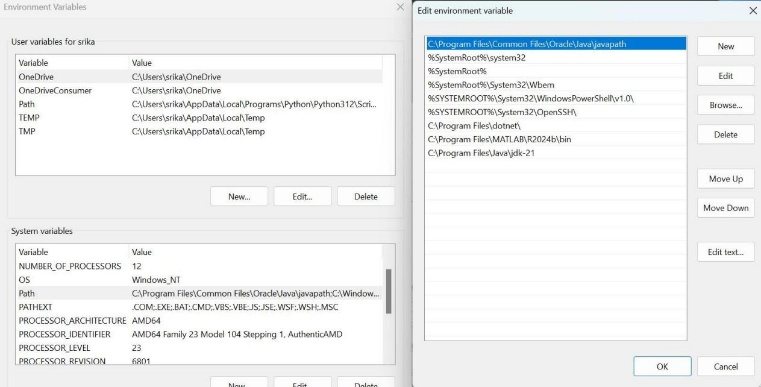
**Step – 1: Download JDK-21 from oracle website**



**Step-2: Install the JDK-21 with accepting terms and conditions according to the respective windows**

**Step-3: Setting up environmental variables***.*





1. Open Environment Variables:

Right-click This PC → Properties → Advanced system settings → Environment Variables.

2. Update JAVA\_HOME:

In System variables, find JAVA\_HOME.

Click Edit and set it to the path of the desired JDK (e.g., C:\Program Files\Java\jdk-21).

If it doesn’t exist, click New → Variable Name: JAVA\_HOME, Variable Value: C:\Program Files\Java\jdk-21.

3. Update Path:

Under System variables, select Path → Edit.

Add a new entry: %JAVA\_HOME%\bin.

3. Verify the Version

Open a new Command Prompt window and run:

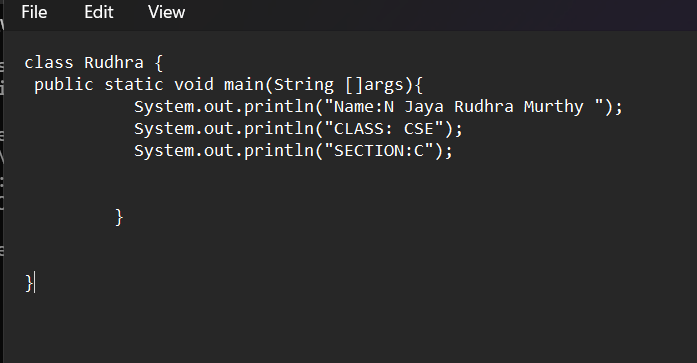
java -version

javac -version

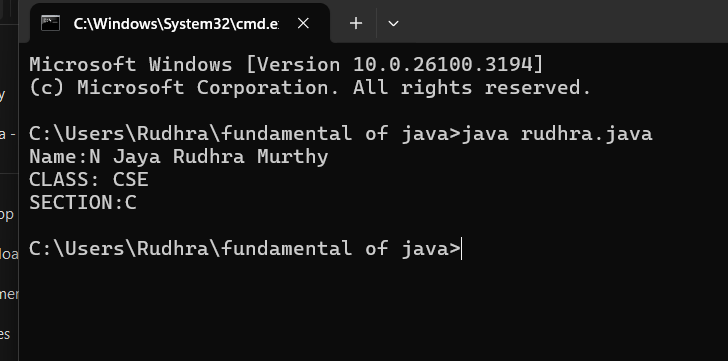
They should now display the updated Java version.

**1.AIM:**Simple Java Program for printing Name, Class, Roll No, of a Student

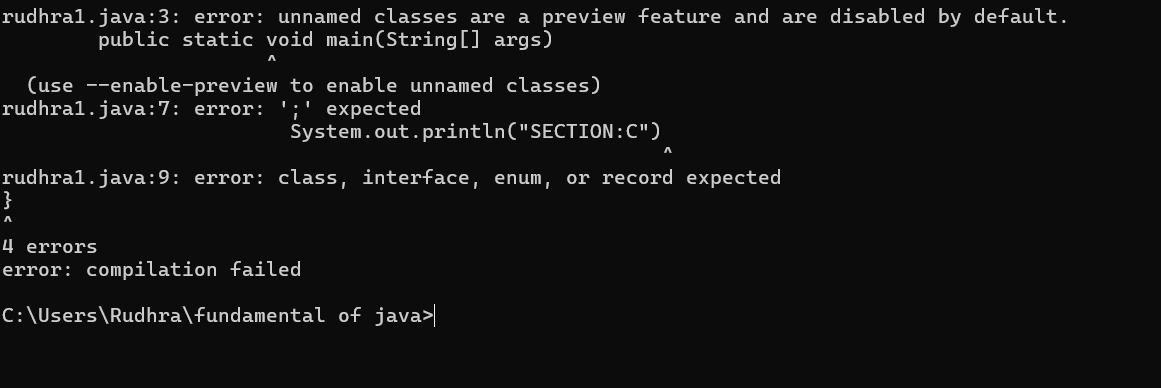
**CODE: -**

****

**Output: -**



Negative case:



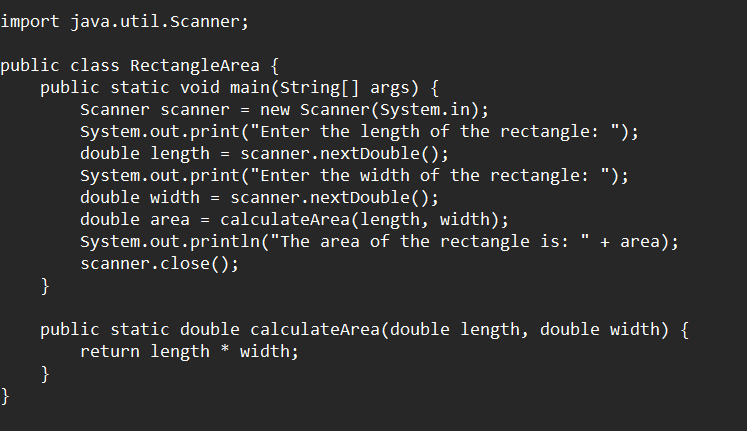
**Errors :**

|  |  |  |
| --- | --- | --- |
| **1** | **Syntax error** | **Semicolon added** |
| 2 | Runtime error | Copied correct path |
| 3 | Name error | rectified |

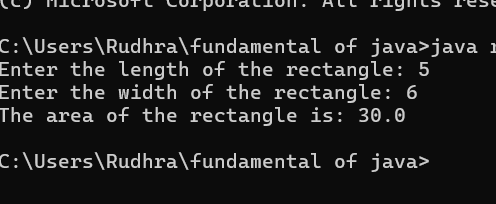
**Week -2 : SIMPLE JAVA PROGRAM**

1. **AIM:JAVA PROGRAM AREA OF RECTANGLE**

**Code:**

****

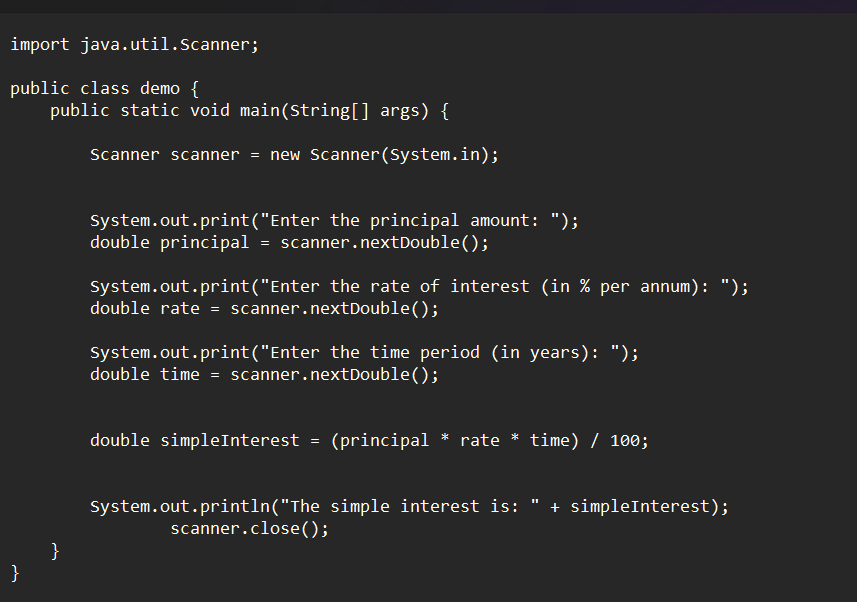
**Output:**

****

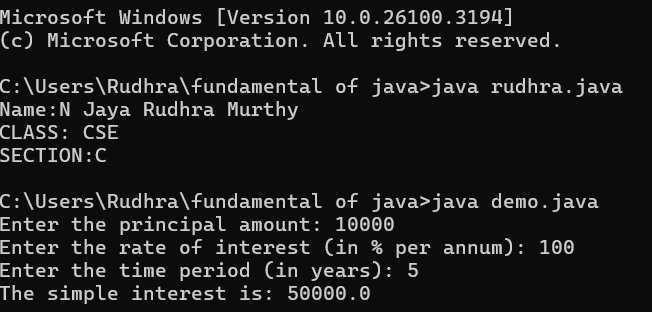
**Negative case:**

1. **AIM:Java program from calculate the simple interest**

**CODE:**

****

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **rectification** |
| 1 | Runtime error | Incorrect path | Copied correct path |
| 2 | Syntax error | { missing | { added |
| 3 | Logical error | Wrong formula | Formula rectified |

1. **AIM: Java program for Fibonacci sieries**

Syntax: import java.util.\*;

class fibo

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

int num;

int f3;

int f1 = 0;

int f2 = 1;

int i = 2;

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

num = sc.nextInt();

System.out.println(f1);

System.out.println(f2);

while(i<num)

{

}

f3 = f1+f2;

f1 = f2;

f2 = f3;

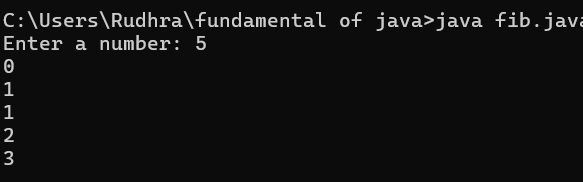
System.out.println(f3);

i = i+1;

}

}

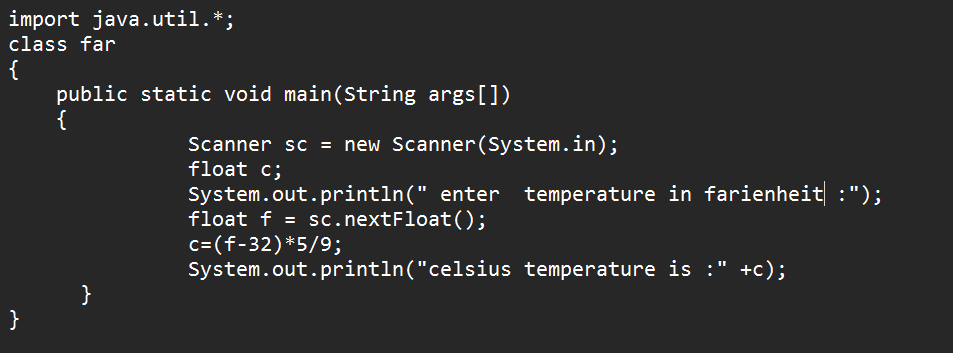
**Output:**



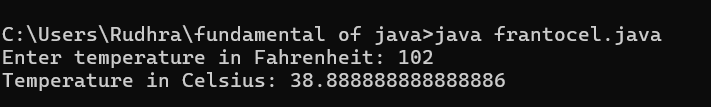
|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| 1 | Name error | Incorrect usage of function | Correcting by using correct formula |
| 2 | Syntax error | No semicolon | Acolnidded sem |
| 3 | Runtime error | Incorrect path | Copied correct path |

1. **AIM:Java program for calculating temperature in Celsius**

**CODE:**

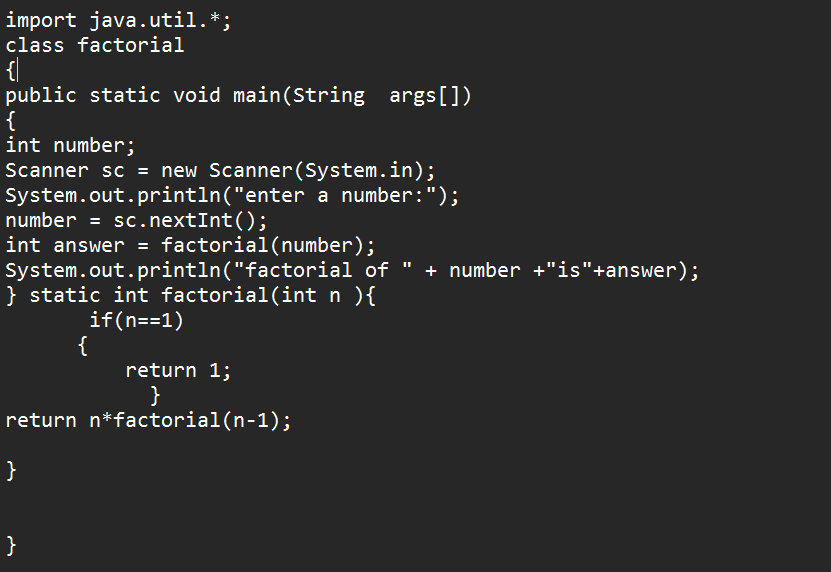
****

Output:

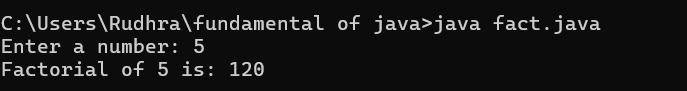


|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **rectification** |
| 1. | Syntax | No semicoln | Added semicoln |
| 2. | Logical error | Due to incorrect input | Corrected by giving correct input |
| 3. | Runtime error | Incorrect path | Using correct path |

**5.AIM:Java program factorial of a given number**

****

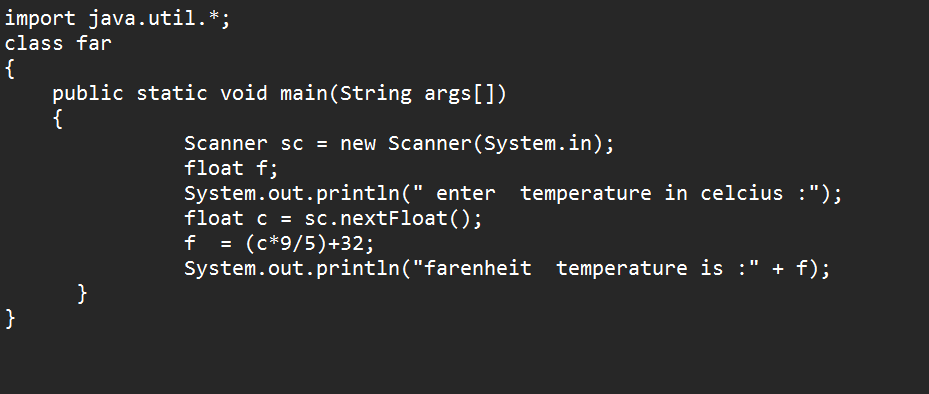
Output:



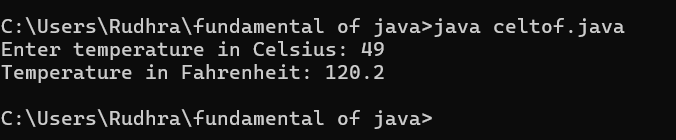
|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **rectification** |
|  | Logical error | Incorrect input | Correcting input |
|  | Runtime error | Incorrect path | Using correct path |
|  | syntax | No semicoln | Using semicoln |

**6.AIM:Java program for Celsius to farenheit:**

**CODE:**



Output:



**7.AIM:Write a program to find the area of triangle by using heron’s formula take the input from the user**

Code:



Output:



**Week -3:**

**1.AIM:**

**To create java program with following instructions**

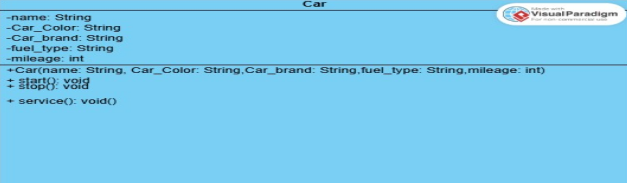
**1.Create a class with name car**

**2. Create four attributes named car\_color Car\_brand,fuel\_type,mileage**

**3. Create three methods named start(), stop(). Service()**

**4. Create three objects named car1,car2 and car3**

**Class diagram:**



**Code:**

import java.util.\*;

class car

{

public String Car\_color;

public String Car\_brand;

public String fuel\_type;

public int mileage;

public void start()

{

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

System.out.println("Car color is :"+Car\_color);

System.out.println("Car Brand is:"+Car\_brand);

System.out.println("Car fuel type is:"+fuel\_type);

System.out.println("Car mileage is:"+mileage);

}

public void service()

{

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

System.out.println("Car color is :"+Car\_color);

System.out.println("Car Brand is:"+Car\_brand);

System.out.println("Car fuel type is:"+fuel\_type);

System.out.println("Car mileage is:"+mileage);

}

public void stop()

{

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

System.out.println("Car color is :"+Car\_color);

System.out.println("Car Brand is:"+Car\_brand);

System.out.println("Car fuel type is:"+fuel\_type);

System.out.println("Car mileage is:"+mileage);

}

public static void main(String args[])

{ System.out.println("\nBHANU TEJA\n\n");

car car1 = new car();

car1.Car\_color = "Blue";

car1.Car\_brand = "Audi";

car1.fuel\_type = "Deisel";

car1.mileage = 100;

car1.start();

car car2 = new car();

car2.Car\_color = "Red";

car2.Car\_brand = "Tesla";

car2.fuel\_type = "EV";

car2.mileage = 200;

car2.stop();

car car3 = new car();

car3.Car\_color = "Yellow";

car3.Car\_brand = "BMW";

car3.fuel\_type = "Petrol";

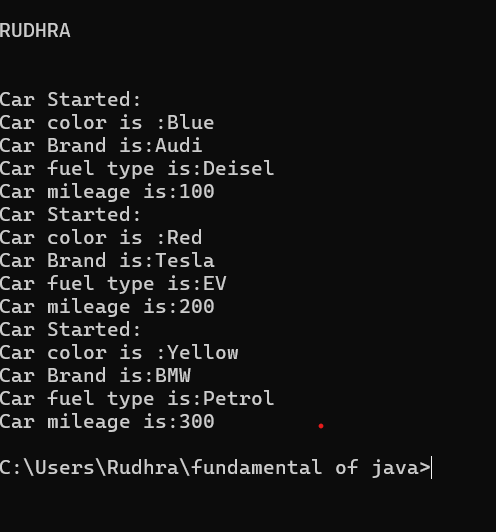
car3.mileage = 300;

car3.service();

}

**}**

**Output:**

****

**Important points:**

 The car class has four attributes: Car\_color, Car\_brand, fuel\_type, and mileage.

 It also has three methods: start(), service(), and stop().

 The start(), service(), and stop() methods all print the same details about the car.

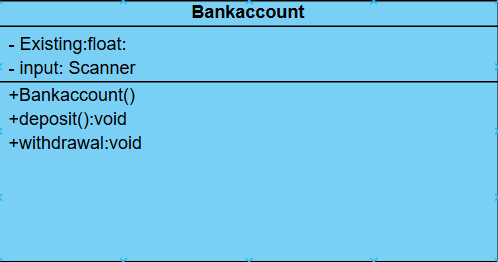
 Each method prints the car's color, brand, fuel type, and mileage to the console

 The main method creates three instances of the car class: car1, car2, and car3.

 Each car object is assigned specific values for Car\_color, Car\_brand, fuel\_type, and mileage.

**2. AIM:To create a class bankAccount with methods deposit() and withdrawl**

**Class diagram:**



**Code:**

class BankAccount

{

private double balance;

public BankAccount(double initialBalance)

{

if(initialBalance > 0)

{

this.balance = initialBalance;

}

else

{

this.balance = 0;

}

}

public void deposit(double amount)

{

if(amount>0)

{

balance = balance+amount;

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

}

else

{

System.out.println("Deposited amount must be positive");

}

}

public double getBalance()

{

return balance;

}

}

public class Main1

{

public static void main(String args[])

{

BankAccount account = new BankAccount(1000);

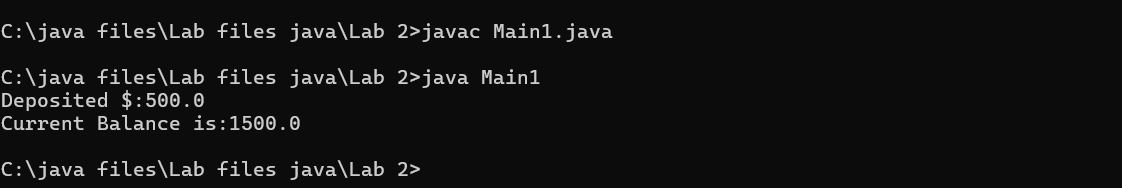
account.deposit(500);

System.out.println("Current Balance is:"+account.getBalance());

}

}

**Output:**



**Important points:**

The BankAccount class has a private attribute balance to store the account balance.

* The class has a constructor, BankAccount(double initialBalance), which initializes the balance. If the initial balance is not positive, it sets the balance to 0.
* The deposit(double amount) method adds a positive amount to the balance and prints a message. If the deposit amount is not positive, it prints an error message.
* The getBalance() method returns the current balance of the account.
* The Main1 class contains the main method, which serves as the entry point of the program.
* In the main method, an instance of BankAccount is created with an initial balance of 1000.

WEEK-4

1.AIM:

WRITE A JAVA PROGRAM WITH CLASS NAMED “Book”. THE CLASS SHOUKD CONTAIN VARIOUS ATTRIBUTES SUCH AS TITLE, AUTHOR, YEAR zOF

PUBLICATION. IT SHOULD ALSO CONTAIN A CONSTRUCTOR WITH

PARAMETERS WHICH INITIALIZES TITLE, AUTHOR, YEAR OF PUBLICATION

AND CREATE A METHOD WHICH DISPLAYS THE DETAILS OF 2 BOOKS.

PROGRAM:

class book{

public String title\_of\_book;

public String author;

public int year\_of\_publication;

public void start() {

System.out.println("Title of the book is :"+title\_of\_book);

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

System.out.println("Year of publication of the book is :"+year\_of\_publication);

}

public void service() {

System.out.println("Title of the book is :"+title\_of\_book);

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

System.out.println("Year of publication of the book is :"+year\_of\_publication);

}

public static void main(String[] args){

book book1=new book();

book1.title\_of\_book="Harry Potter-The Goblet Of Fire";

book1.author="JK Rowling";

book1.year\_of\_publication=1992;

book1.start();

book book2=new book();

book2.title\_of\_book="Harry Potter-Deathy Hallows";

book2.author="JK Rowling";

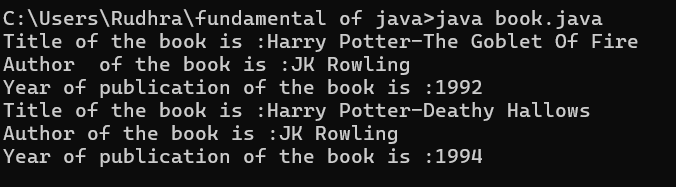
book2.year\_of\_publication=1994;

book2.service();

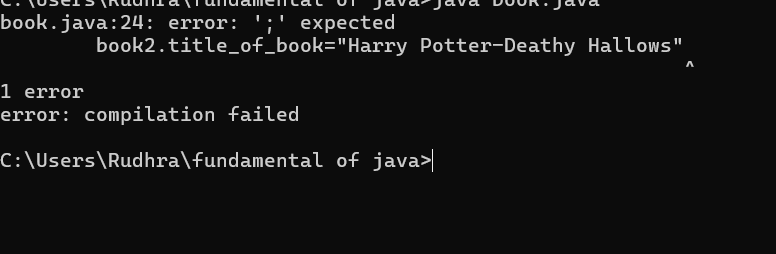
}

}

**Output:**

****

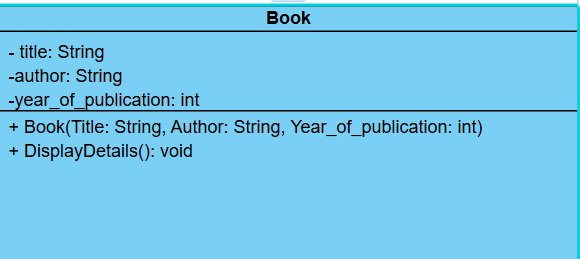
**Negative case:**



**ERROR:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **ERROR TYPE** | **Reason for error** | **Rectification** |
| **1.** | Syntax error | No semicolon | Semicolon added |
| **2.** | Runtime error | Incorrect path | Copied correct path |

**Class diagram:**



**IMPORTANT POINTS:**

1. **Constructor**:

* The constructor Book(String, String, int) is used to initialize the object when it is created.
* The keyword this is used to differentiate between class attributes and constructor parameters.

2.**Method**:

* The method displayDetails() is used to display the book details.
* The System.out.println() method prints the details to the console.

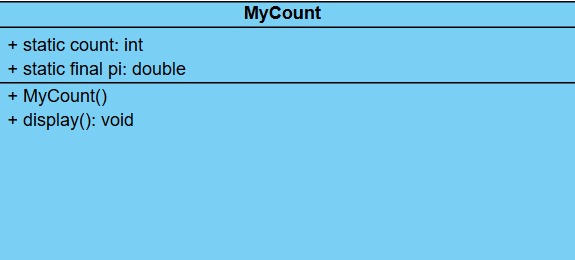
3. **Object Creation**:

* Two objects b1 and b2 are created using the constructor.

**2.AIM:**

**WRITE A JAVA PROGRAM WITH CLASS NAMED “MyClass” WITH A STATIC VARIABLE COUNT OF INT TYPE. INTIALIZE IT TO ZERO AND A CONSTANT VARIABLE “Pi” OF TYPE DOUBLE INITIALIZED TO “3.14” 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 AND CREATE 3 OBJECTS.**

**Class diagram:**



**Code:**

class myclass{

static int count = 0;

static final double pi = 3.14;

public myclass() {

count++;

}

public void start() {

System.out.println("Value of count : " + myclass.count);

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

}

public void service() {

System.out.println("Value of count : " + myclass.count);

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

}

public void stop(){

System.out.println("Value of count : " + myclass.count);

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

}

public static void main(String[] args) {

myclass obj1 = new myclass();

obj1.start();

myclass obj2 = new myclass();

obj2.service();

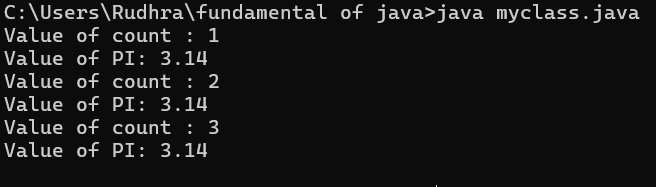
myclass obj3 = new myclass();

obj3.stop();

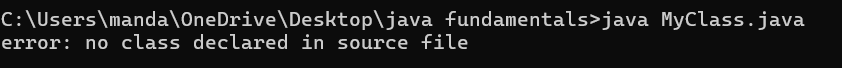
}

}

**Output:**

****

Negative case:



**ERROR:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error Type** | **Reason for error** | **Rectification** |
| **1.** | No class | No class name declared | Created class named ‘MyClass’ |
| **2.** | Syntax error | Not added keyword | Added keyword named ‘new’ |

**IMPORTANT POINTS:**

1.Static Keyword

* Static members belong to the class, not to individual objects.
* Only one copy of the static variable is maintained for all objects.

2.Static Variable

* static int count:
  + Shared among all objects of the class.
  + It is initialized only once and not for every object.
  + It increments every time the constructor is called.

3.Final Variable

* static final double pi:
  + The final keyword makes the variable constant.
  + Its value cannot be changed once assigned.
  + It must be initialized at the time of declaration.

**Week-5:**

**AIM:** Create a calculator using the operations including addition, subtraction Multiplication and division using multilevel inheritance and display the desired Output

Class Diagram**:**



**Code:**

class addition

{

public int add(int a, int b)

{

int addition = a+b;

return addition;

}

}

class subtraction extends addition

{

public int sub(int a, int b)

{

int subtraction = a-b;

return subtraction;

}

}

class multiplication extends subtraction

{

public int mult(int a, int b)

{

int multiplication = a\*b;

return multiplication;

}

}

class division extends multiplication

{

public int div(int a,int b)

{

int division = a/b;

return division;

}

}

class calculator

{

public static void main(String args[])

{

division obj = new division();

System.out.println("Addition is:"+ obj.add(10,2));

System.out.println ("Subtraction is:"+obj.sub(8,4));

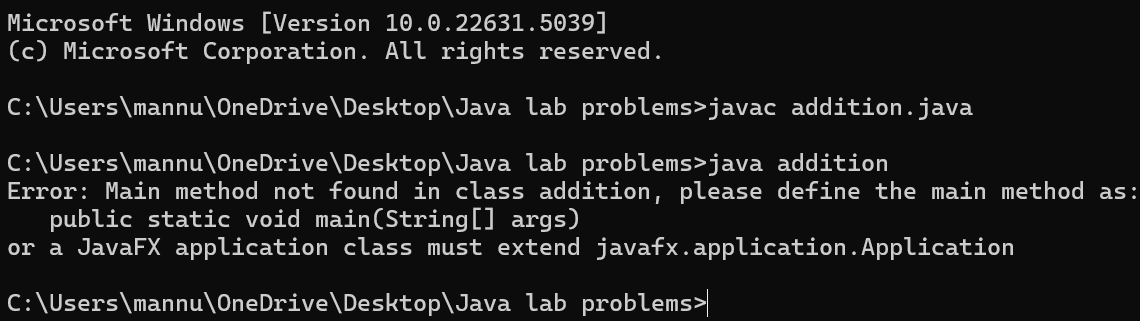
System.out.println("Multiplication is:"+obj.mult(12,4));

System.out.println("Division is:"+obj.div(8,4));

  }

}

**Negative Case:**



**Error Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Error Type | Cause | Rectification |
| 1 | Constructor error | Invalid name to method | Defined class name |
| 2 | Syntax error | Expected ‘(‘ | Added parenthesis |
| 3 | Logical error | Incorrect arithmetic  operation | Correct operation  rectified |

**Important Points:**

**Inheritence:**

The concept of OOP where a class inherits the properties and behaviours from

Another class (parent class) which promotes code reusability and hieratchical relationships

**Multilevel Inheritence:**

This is a type of inheritance in which a class inherited from another class, and

That superclass, in turn, inherits from yet another class, creating a chain of

inheritence

**extends:**

The extends keyword defines the relation of child class with the parent class

**2.Aim:** A vehicle rental company wants to develop a system that maintains Information about different types of vehicles available for rent

The Company rents out cars, bikes and truck and they need a program to

Store details about each vehicle, such as brand and speed

Cars should have an additional property: number of doors

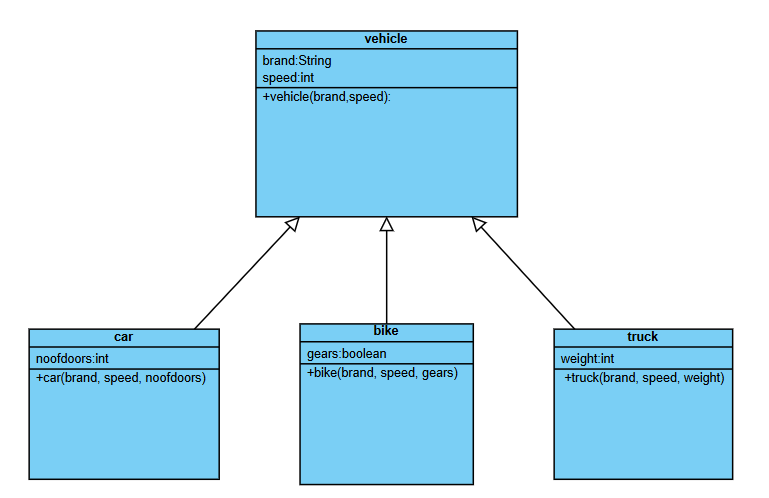
Bikes should have a property indicating whether they have gears or not

The system should also include a function to display details about each vehicle

And indicate when a vehicle is starting

Class diagram

**CLASS DIAGRAM:**



**0**

**Code:**

class vehicle{

String brand;

int speed;

public vehicle(String brand,int speed){

this.brand=brand;

this.speed=speed;

}

public static void main(String[] args) {

car obj1=new car("ford",34,4);

bike obj2=new bike("hero",100,true);

truck obj3=new truck("tata",60,40);

}

}

class car extends vehicle{

int noofdoors;

public car(String brand, int speed,int noofdoors) {

super(brand, speed);

this.noofdoors=noofdoors;

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

System.out.println("Speed of car is:"+speed);

System.out.println("no of doors of car:"+noofdoors);

}

}

class bike extends vehicle{

boolean gears;

public bike(String brand,int speed,boolean gears){

super(brand, speed);

this.gears=gears;

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

System.out.println("Speed of bike is:"+speed);

System.out.println("Gears of bike:"+gears);

}

}

class truck extends vehicle{

int weight;

public truck(String brand,int speed,int weight){

super(brand,speed);

this.weight=weight;

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

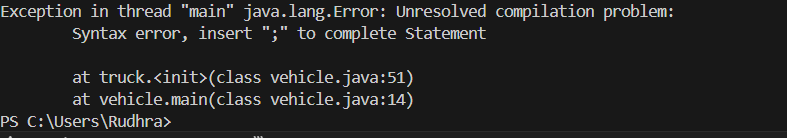
System.out.println("Speed of Truck is:"+speed);

System.out.println("Weight of load is"+weight);

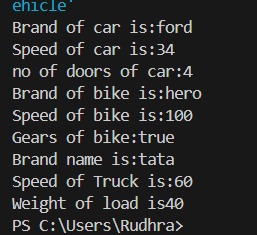
}

}

**NEGATIVE CASE:**

****

**OUTPUT:**

****

**Error Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| S No | Error Type | Cause | Rectification |
| 1 | Syntax Error | Semicolon missing | Added ; |
|  |  |  |  |

**Important Points:**

**Hierarchical Inheritence:**

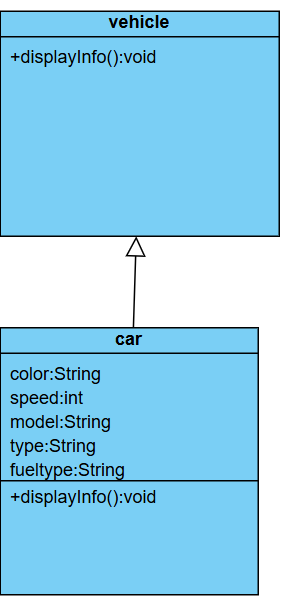
This is a type of inheritance occurs when multiple subclasses inherit from a

Single parent class

**WEEK-6:**

**AIM:**Write a java program to create a Vehicle class with a method displayInfo().Override this method in the car subclass to provide specific information about a car.

**Class diagram:**



**Code:**

class vehicle{

public void displayInfo(){

System.out.println("This is a vehicle.");

}

}

class car extends vehicle{

String color = "red";

int speed = 120;

String model = "Toyota";

String type = "SUV";

String fuelType = "Petrol";

public void displayInfo(){

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

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

33

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

System.out.println("Model: " + model);

System.out.println("Type: " + type);

}

}

class Override{

public static void main(String[] args) {

  System.out.println("N.Jaya Rudhra Murthy");

vehicle v = new vehicle();

v.displayInfo(); // Calls the method from the vehicle(super) class

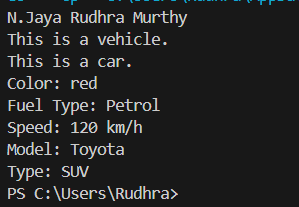
car c = new car();

c.displayInfo(); // Calls the overridden method from the car(sub) class

}

}

Output:



**Important points:**

\* Here class vehicle and car share a single level inheritance between them.

\*Here in the above code is an example of method overriding in java where the method names and their signature are same in both super class and sub class.

34

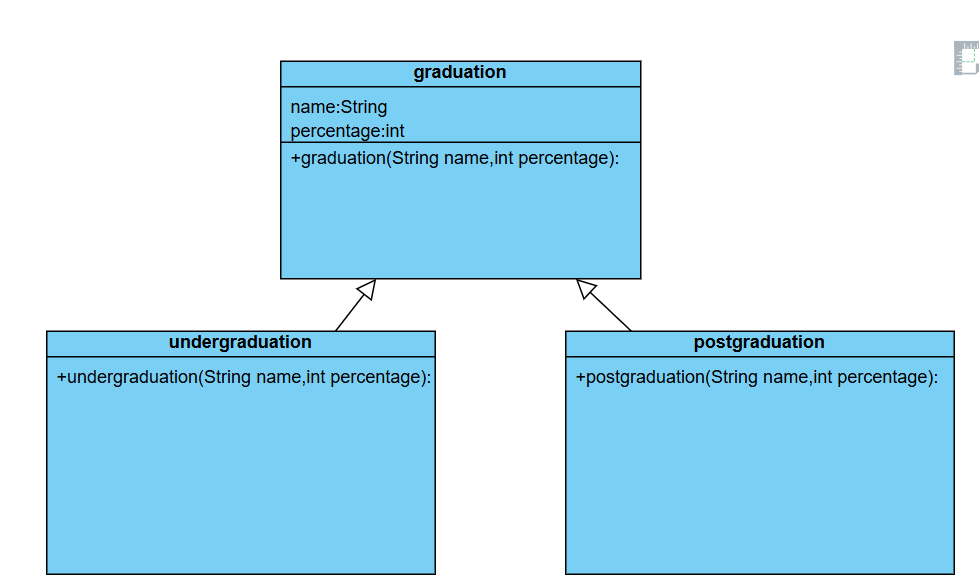
**AIM:** A college is developing an automated admission system that verifies student’s eligibility for Undergraduate(UG) and Postgraduate(PG) programs. Each program has different eligibility criteria based on the student’s percentage in their previous qualification.

CONDITION:

1.UG admission require minimum of 60%.

2.PG admission require minimum of 70%.

**Class diagram:**



**Code:**

class student{

String name;

double percentage;

public student(String name,double percentage){

this.name=name;

this.percentage=percentage;

}

void eligibility(){

System.out.println(name+"this student must have the foolowing criteria");

}

}

class UG extends student{

UG(String name,double percentage){

super(name,percentage);

}

void eligibility(){

if(percentage>=60){

System.out.println(name+"this student is eligible for ug");

}

else{

System.out.println(name+"this student is not eligible for ug");

}

}

}

class PG extends student{

PG(String name,double percentage){

super(name,percentage);

}

void eligibility(){

if(percentage>=70.0){

System.out.println(name+"this student is not eligible for pg");

}

else{

System.out.println(name+"this student is eligible for pg");

}

}

}

class collegeadmission{

public static void main(String[] args){

student s1=new student("Rahul",80.0);

s1.eligibility();

student s2=new student("Rahul",50.0);

s2.eligibility();

PG s=new PG("rahul",70.0);

s.eligibility();

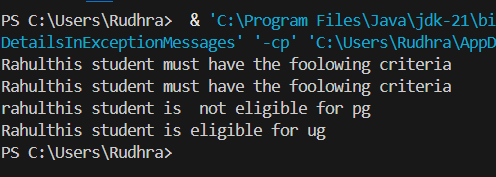
UG u=new UG("Rahul",60.0);

u.eligibility();

}

}

Output:



**Important points:**

\*Here classes undergraduation and postgraduation share a Hierarchical inheritance with graduation class.

\*For each class there is a respective constructor used for the values name and percentage

\*super(name, percentage); Call the constructor of the superclass (graduation).

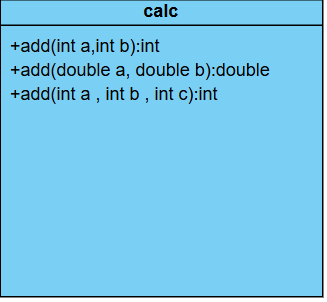
**AIM:** Create a calculator class with overloaded methods to perform addition.

1.Add two integers

2.Add three integers

3.Add two doubles

**Class diagram:**



**Code:**

public class calc{

    //method overloading

    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 static void main(String[] args){

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

        calc c=new calc();

38

System.out.println(c.add(4,5));//as there are two int values in the parameter calls the first add method

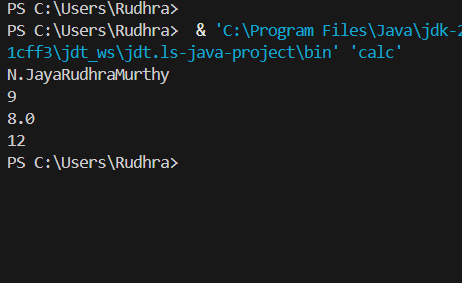
        System.out.println(c.add(3.5,4.5));//as there are two double values in the parameter calls the second add method

        System.out.println(c.add(3,4,5));//as there are three int values in the parameter calls the third add method

        }

}

Output:



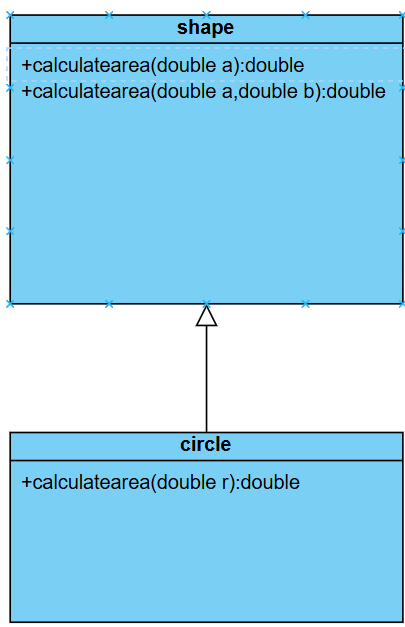
**Important points:**

\*Here all the methods have same name add but each and every method have different signature.

\*Method overloading is taking place.

**AIM:** Create a Shape class with a method calculateArea() that is overloaded for different Shapes (Ex: square,triangle). Then create a subclass circle that overrides the calculatrArea() method for a circle.

**Class diagram:**



**Code:**

public class shape {

//method overloading

public double calculatearea(double a){

return a\*a;

}

40

public double calculatearea(double a,double b){

return a\*b;

}

}

class circle extends shape{

//method overriding

public double calculatearea(double r){

return Math.PI\*r\*r;

}

}

class Main4{

public static void main(String[] args) {

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

shape s=new shape();

circle c=new circle();

System.out.println(s.calculatearea(3.5));//as there are one double value in the parameter calls the first calculatearea method

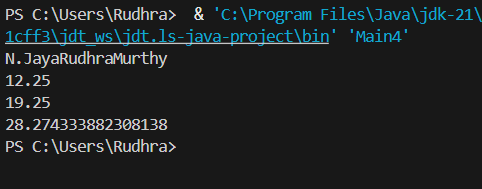
System.out.println(s.calculatearea(3.5, 5.5));//as there are two double values in the parameter calls the second calculatearea method

System.out.println(c.calculatearea(3));//as there are one double value in the parameter calls the overridden calculatearea method from the circle class

}

}

Output:



**Important points:**

\*Here in this code both method overloading and method overriding takes place.