**(23CSE111) Object Oriented Programming**

**LAB MANUAL**



  Department of CSE

                   Amrita School of Computing

        Amrita Vishwa Vidyapeetham, Amaravati Campus

Name: S. charan reddy

Roll no: AV.SC.U4CSE24321

Section: CSE-A

Verified by

**WEEK1:**

**Program1:** How to download and install java

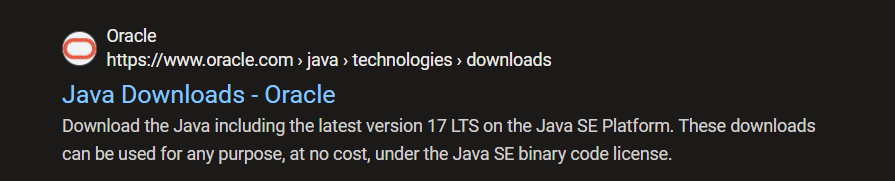
**Aim:** To download and install java

**Procedure:**

**Step1**: Go to browser and search “download java ”.

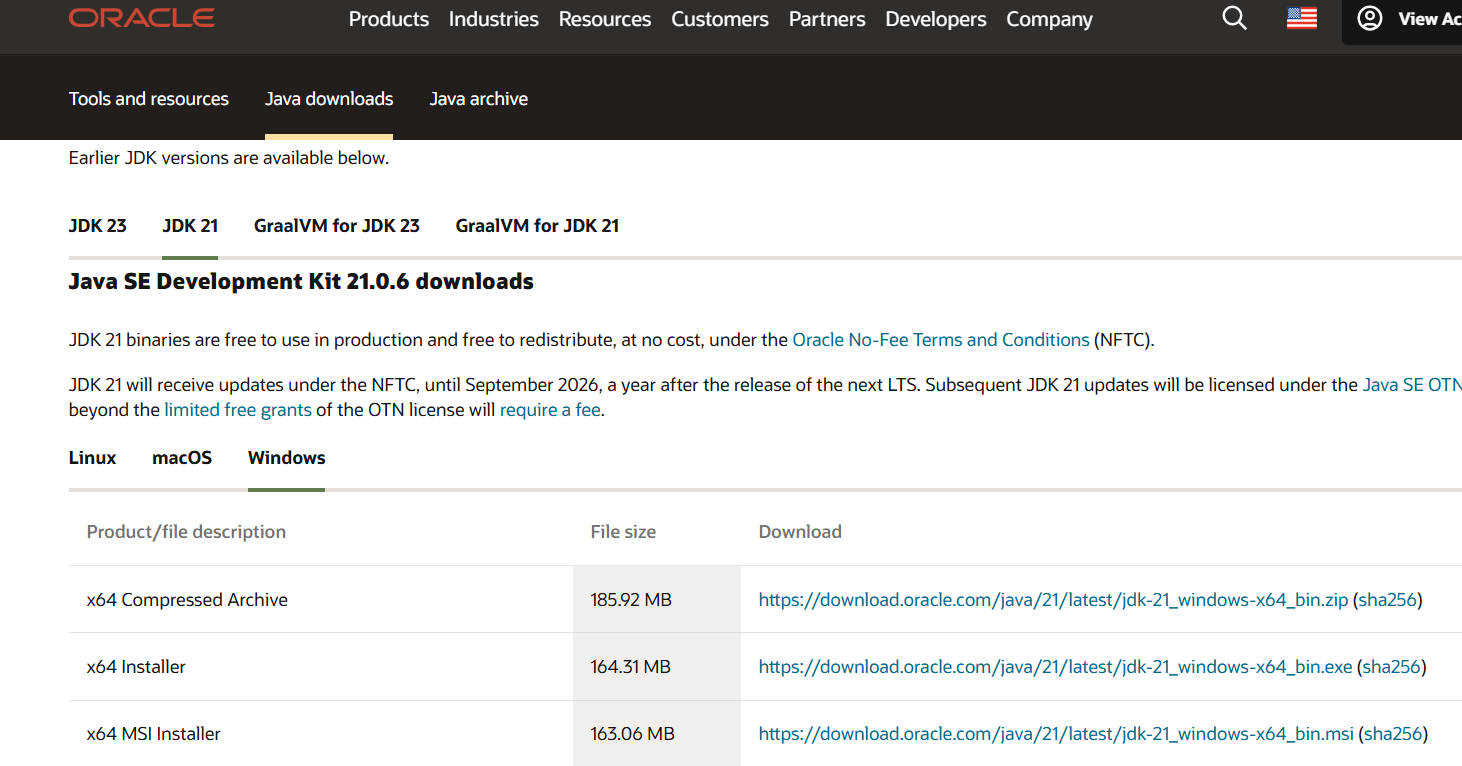
**Step2**: when you search you can see oracle website

to download java , click on it.



**Step3**: Download the java JDK21 version in

linux/macOS/windows in “64x installer”.



**Step4:** After downloading JDK 21 java version

download the installer of java



**Step5:** go to search the environmental variables

**Step6:** select system variables and add the selected Path.

A screenshot of a computer

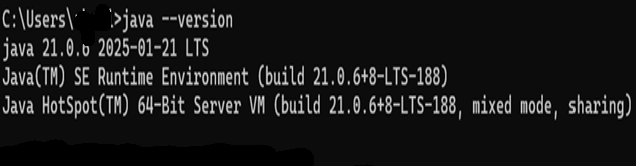
Description automatically generated

**Step7:** lastly do apply

**Step8:** for cross checking it was installed or not open

Command promt and type java --version

Then it will show like this

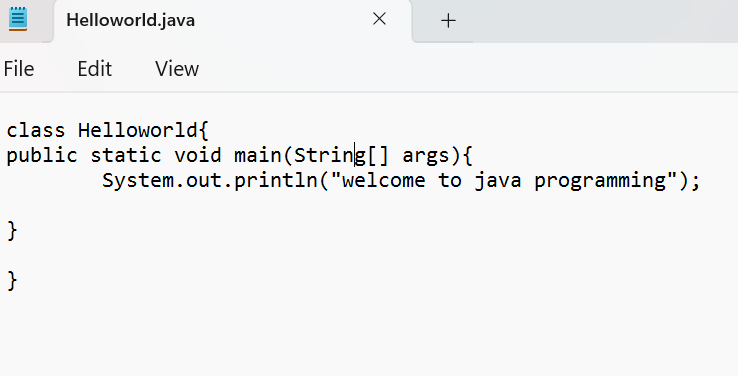


**Program2:** write a java program to print the message

‘welcome to java programming.’

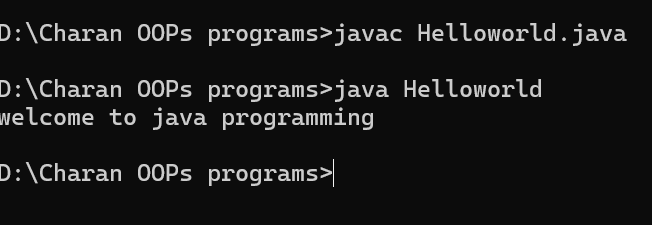
**Aim:** to print ‘welcome to java programming’

**Code:**

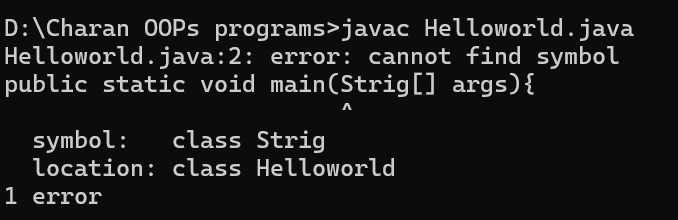


**Output:**

Positive case:



Negative case:



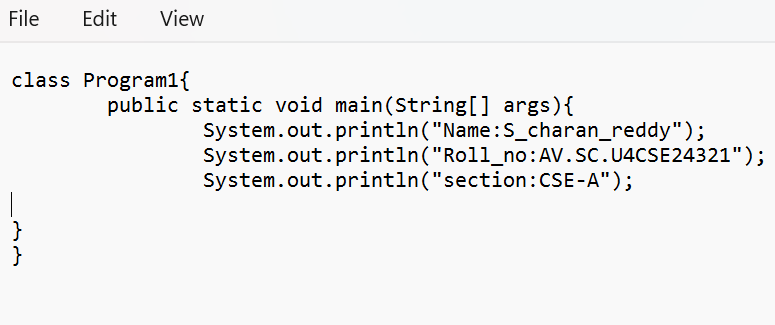
|  |  |  |
| --- | --- | --- |
| S.no | Error message | rectification |
| 1 | symbol: class Strig | symbol: class String |
| 2 | error: package system does not exist system.out.println("welcome to java programming"); | System.out.println(“welcome to java programming”); |

**Errors:**

**Program3:**

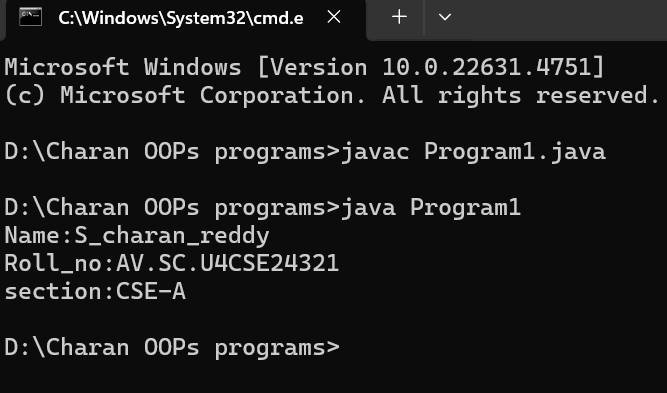
**Aim:** to print name, roll number and section

**Code:**

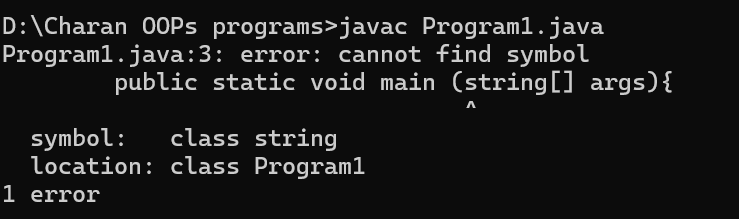


**Output:**

positive case:



Negative case:



**Error:**

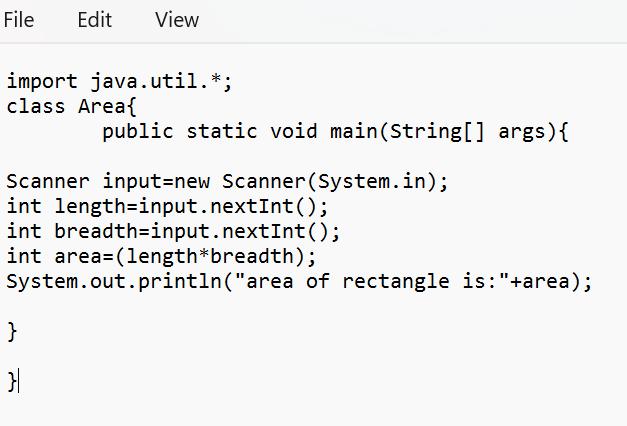
|  |  |  |
| --- | --- | --- |
| s.no | Error message | rectification |
| 1. | Symbol: class string | Symbol: class String |
| 2. | Error: Program1 (wrong name: program1) | class Program1 |

**WEEK2:**

**Program1:**

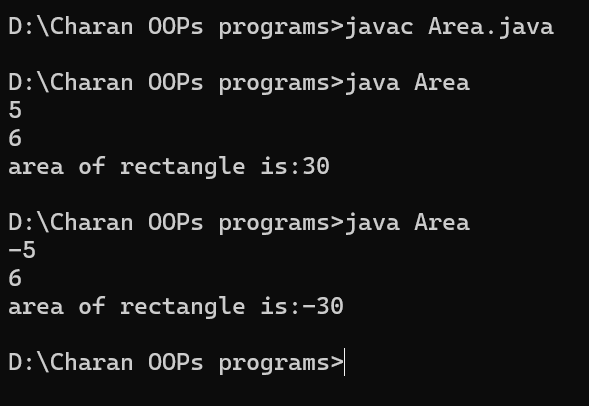
**Aim:** To find area of rectangle

**Code:**

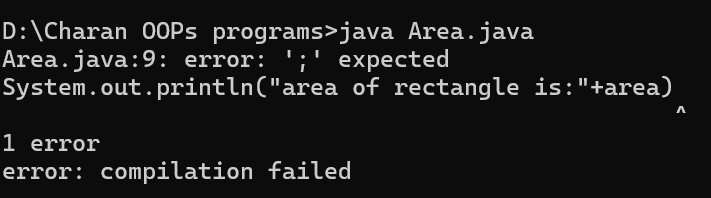


**Output:**

positive case:



Negative case:



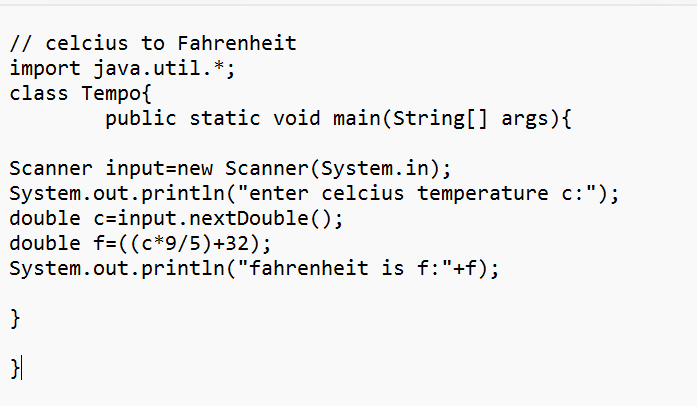
**Error:**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | Syntax error | ; |
| 2 | Symbol: class string | Symbol: class String |

**Program2:**

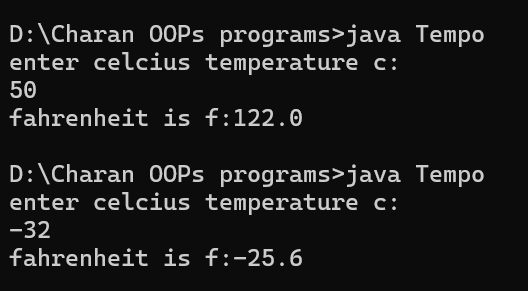
**Aim:** To convert Celsius to Fahrenheit

**Code:**

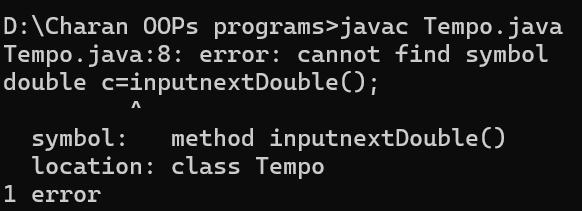


**Output:**

Positive case:



Negative case:



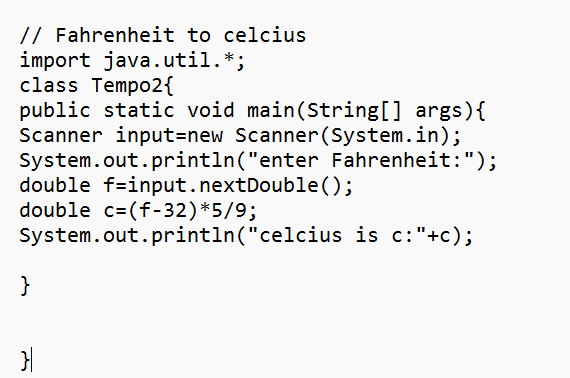
**Error**:

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | Symbol: method inputnextDouble() | Input.nextDouble() |
| 2 | Syntax missing | ; |

**Program3:** write a java program to convert Fahrenheit to Celsius

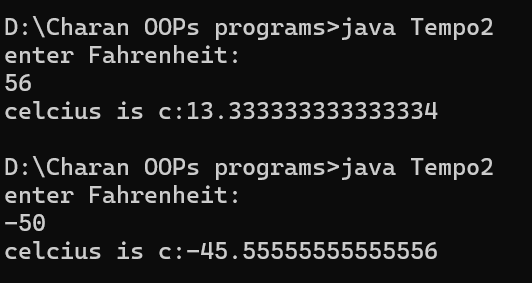
**Aim:** to convert Fahrenheit to Celsius

**Code:**

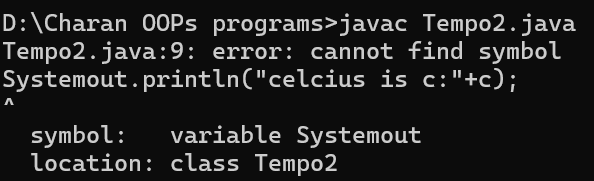


**Output:**

Positive case:



Negative case:

****

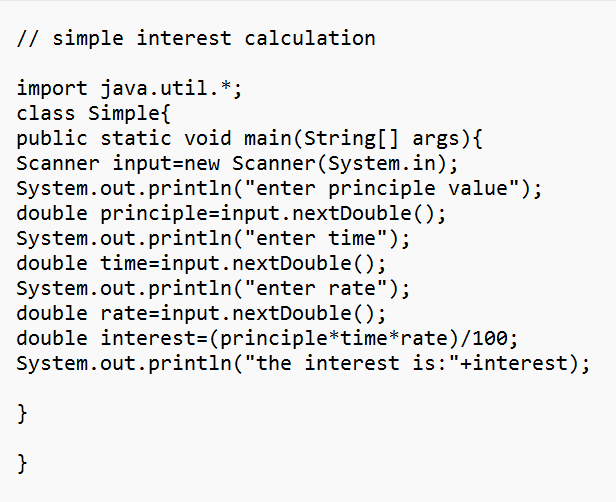
**Error:**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | Symbol: Systemout | System.out |
| 2 | package javautil does not exist  import javautil.\*; | Import java.util.\*; |

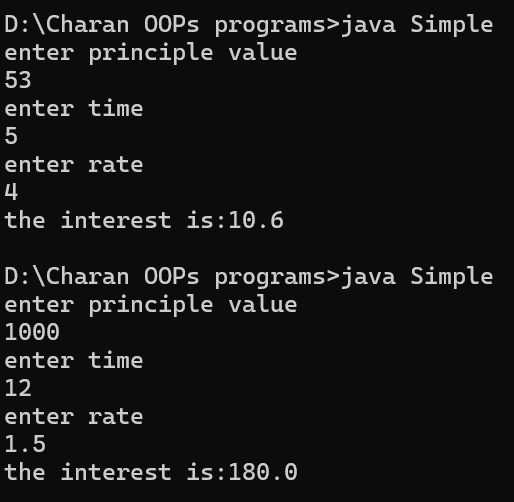
**Program4**: write a java program to calculate simple interest

**Aim:** to calculate simple interest

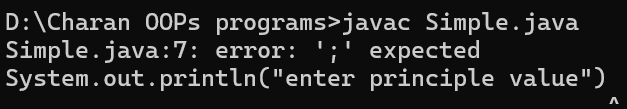
**Code:**



**Output: positive case:**



Negative case:



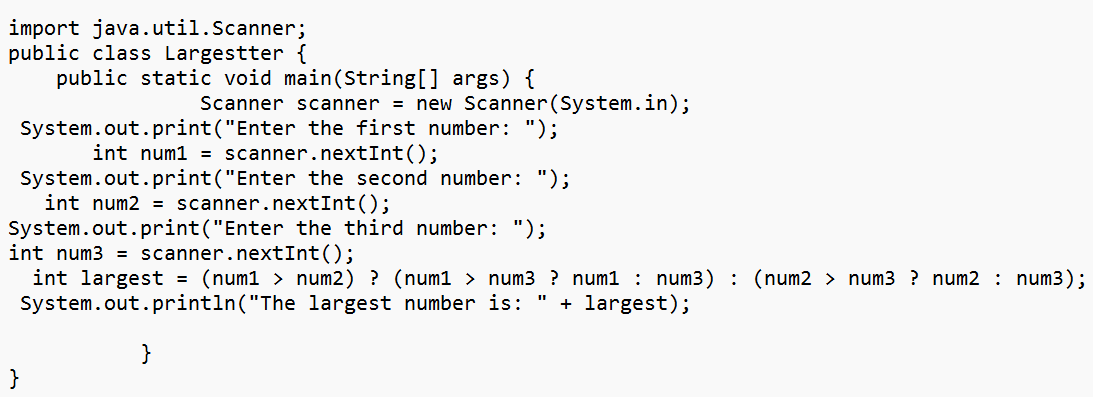
Error:

|  |  |  |
| --- | --- | --- |
| S.no | Error message | rectification |
| 1 | Syntax missing | ; |
| 2 | Symbol: principe | principle |

**Program5**:

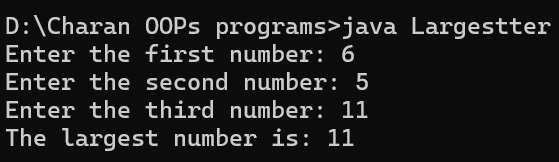
**Aim:** To find largest of three numbers using ternary operator

**Code:**

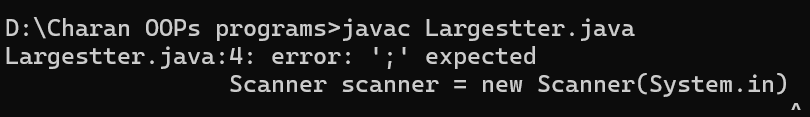
****

**Output:**

Positive case:



Negative case:



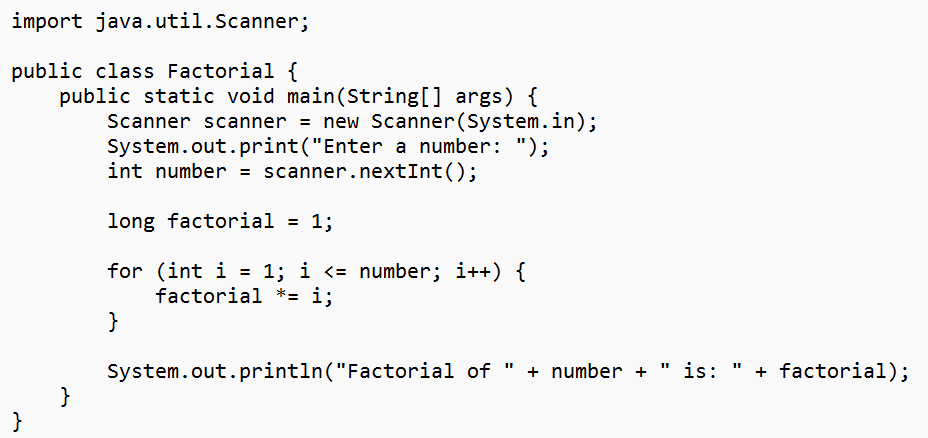
**Error:**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | Scanner scanner = new Scanner(System.in) | Scanner scanner = new Scanner(System.in); |
| 2 | Variable : num not found | num1 |

**Program 6:**

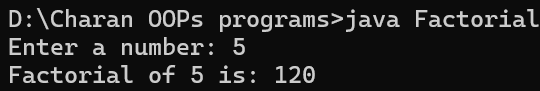
**Aim :** To find factorial of a number

**Code:**

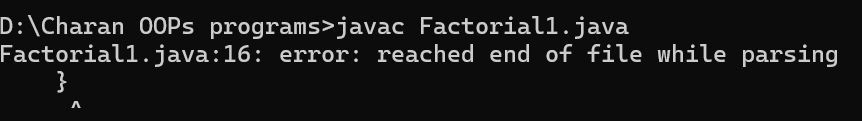
****

**Output :**

Positive case:



Negative case:



**Error :**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | error: reached end of file while parsing  } | } |
| 2 | Scanner not closed | Add read.close(); at the end of the program |

**Week 3:**

**Program 1:**

**Aim** : To create java program with following instructions

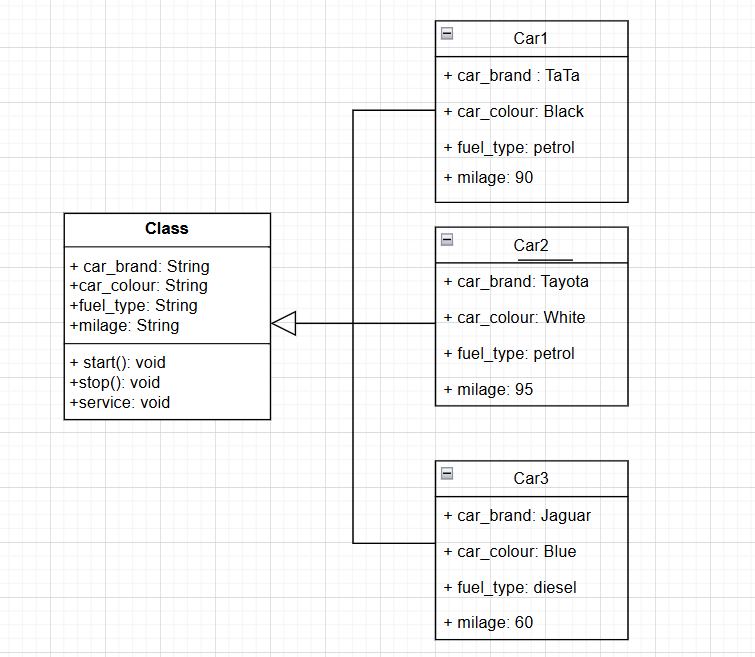
1.Create a class with name car

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

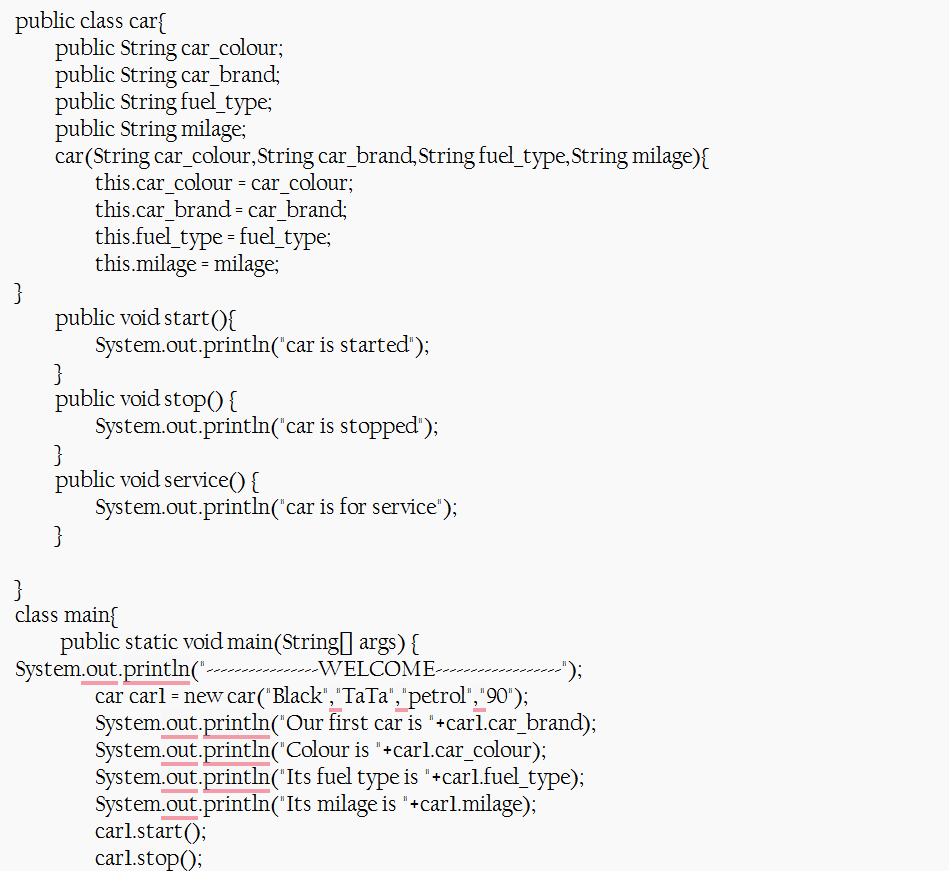
3. Create three methods named start(), stop(). Service()

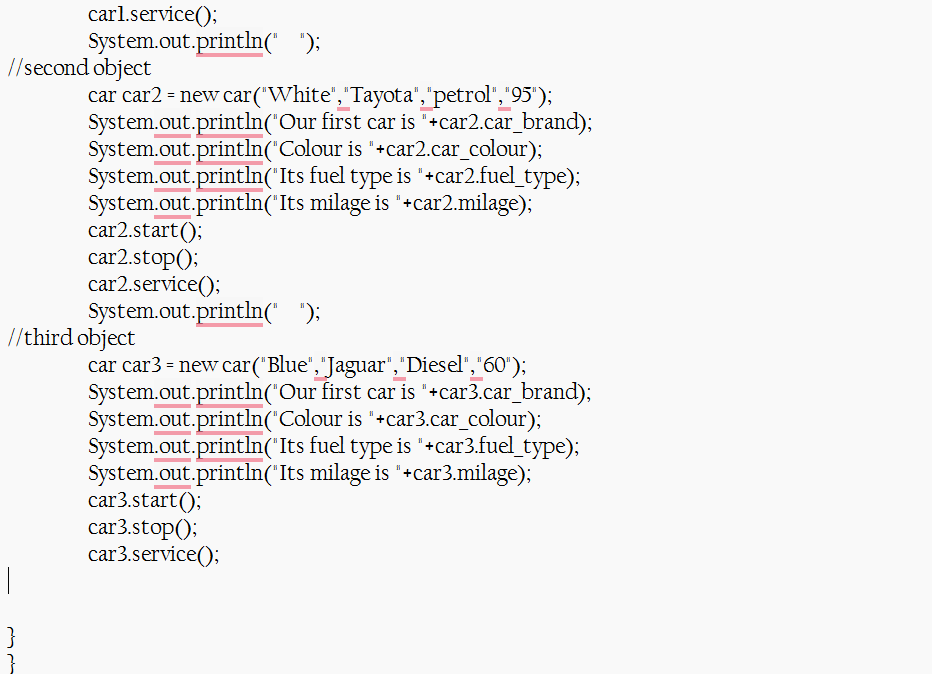
4. Create three objects named car1,car2 and car3

**Class diagram:**

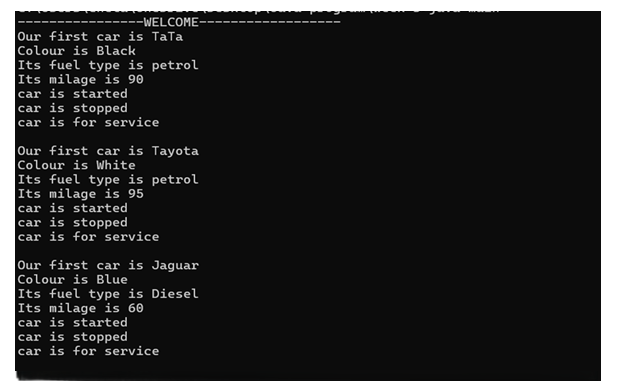


**Code:**





**Output:**

****

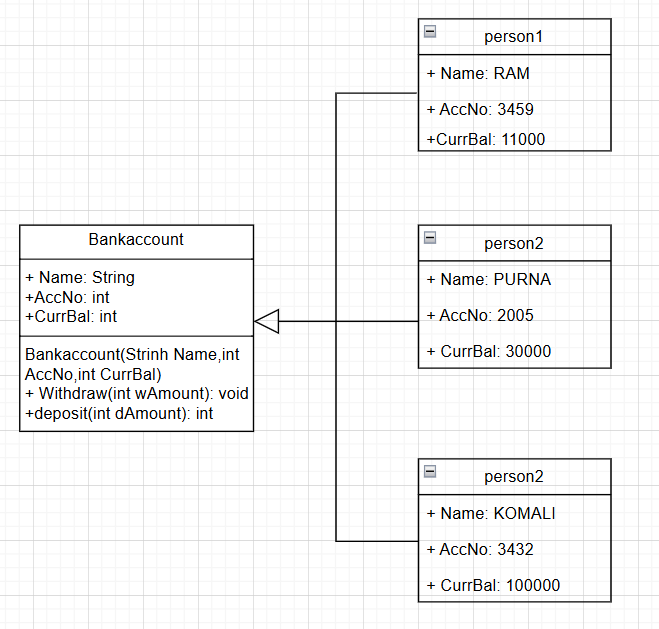
**Error:**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | class main{ | class Main{ |
| **2** | "Our first car is "+car2.car\_brand; | "Our first car is "+car2.car\_brand; |

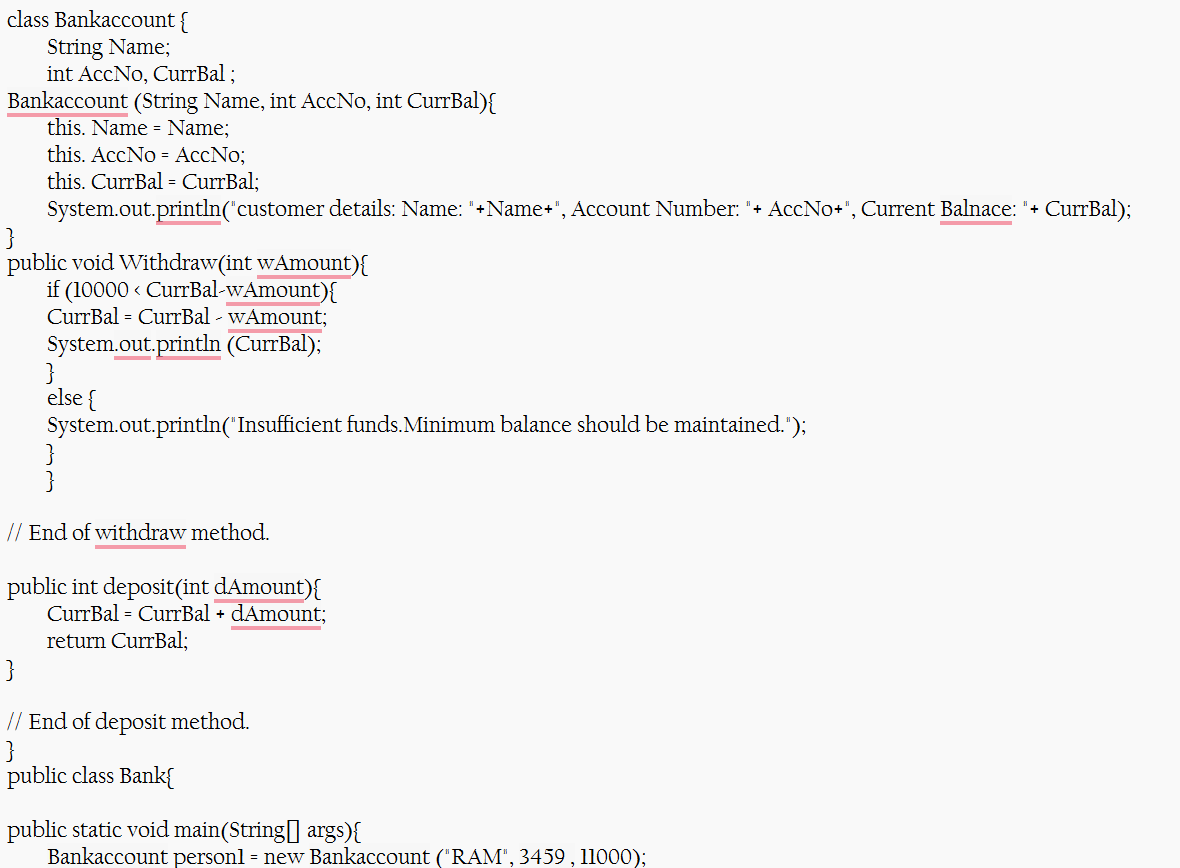
**Program 2:**

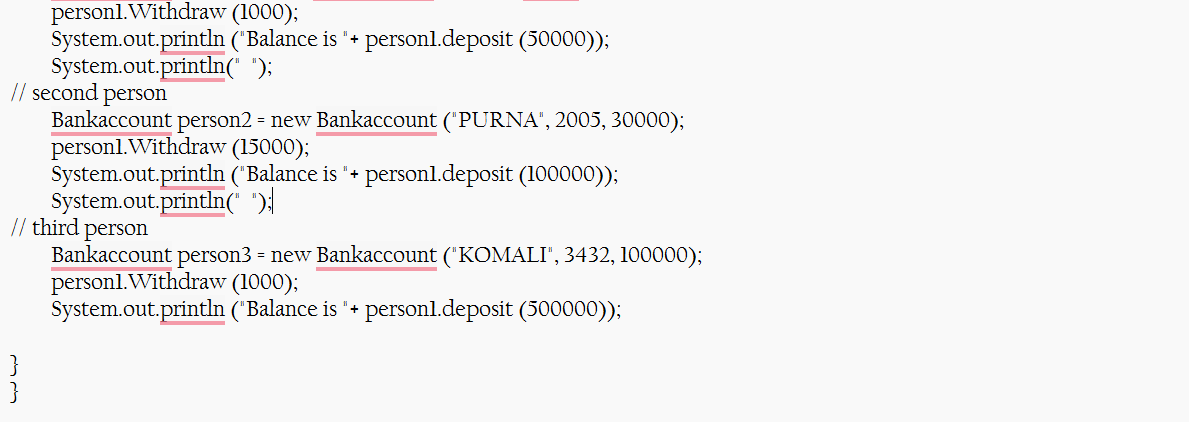
**Aim:** Create the java program to withdraw and deposit money in the bank account

**Class diagram:**

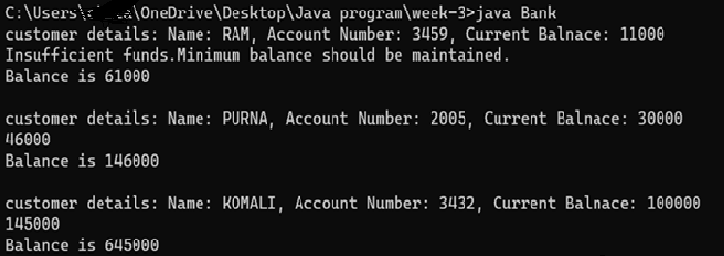


**Code:**





**Output:**

****

**Error:**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | class Bankaccount | class BankAccount (Java follows PascalCase for class names) |
| 2 | BankAccount person-1 (hyphen is not allowed) | BankAccount person1 |

**Week 4:**

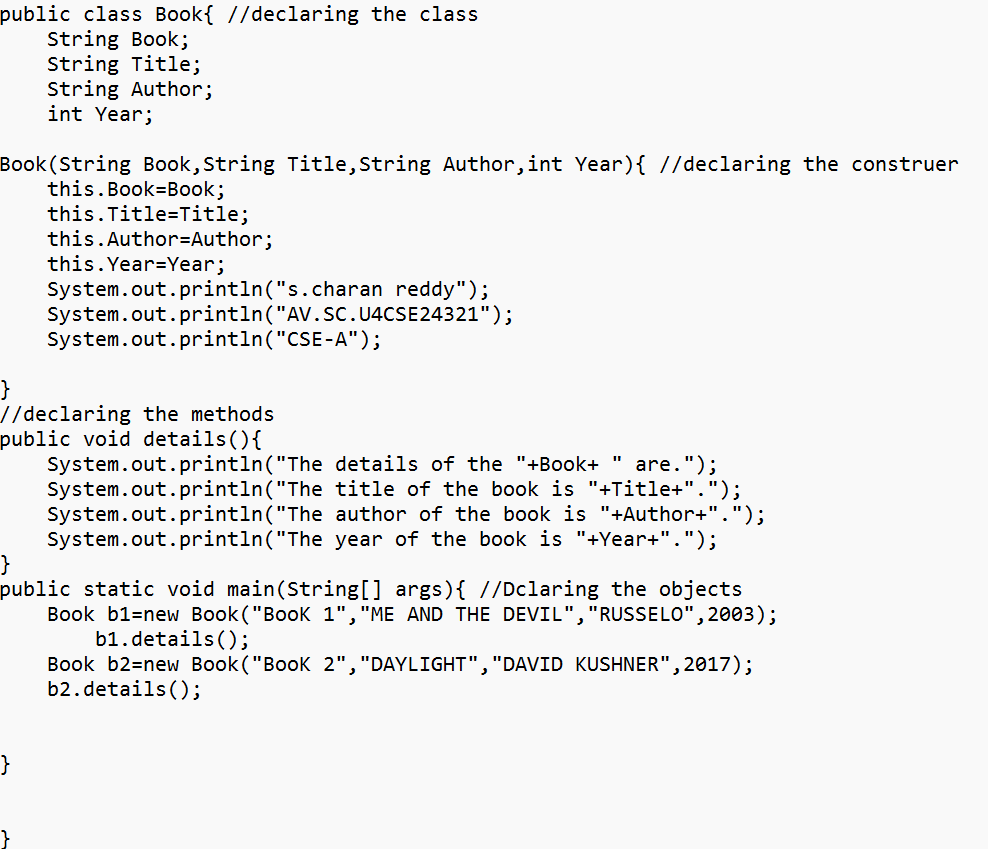
**Aim:** WRITE A JAVA PROGRAM WITH CLASS NAMED “Book”. THE CLASS SHOUKD CONTAIN VARIOUS ATTRIBUTES SUCH AS TITLE, AUTHOR, YEAR OF PUBLICATION. IT SHOULD ALSO CONTAIN A CONSTRUCTOR WITH PARAMETERS WHICH INITIALIZES TITLE, AUTHOR, YEAR OF PUBLICATION AND CREATE A METHOD WHICH DISPLAYS THE DETAILS OF 2 BOOKS.

PROGRAM:

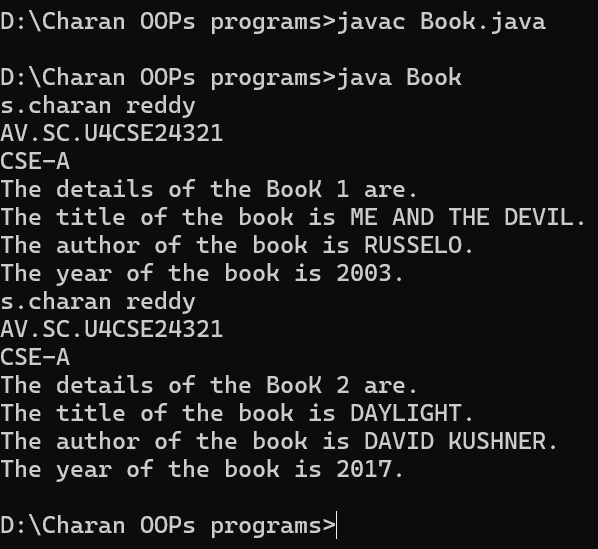
**CLASS DIAGRAM:**

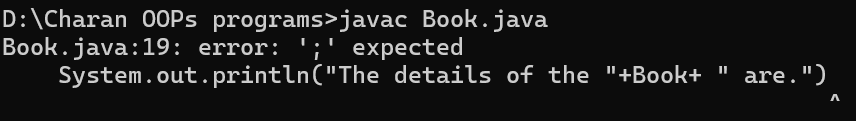
|  |
| --- |
| **Book** |
| **-title: String**  **-author: String**  **-year: int** |
| **+ Book(title: String, author:String, year: int) + displayDetails(): void** |

**Code :**

****

**Output:**

**Positive case:**

**Negative case:  
**

**Error :**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | System.out.println("The details of the "+Book+ " are.")  ^ | System.out.println("The details of the "+Book+ " are."); |
| 2 | Variable mistake: yea | year |

**Program 2:**

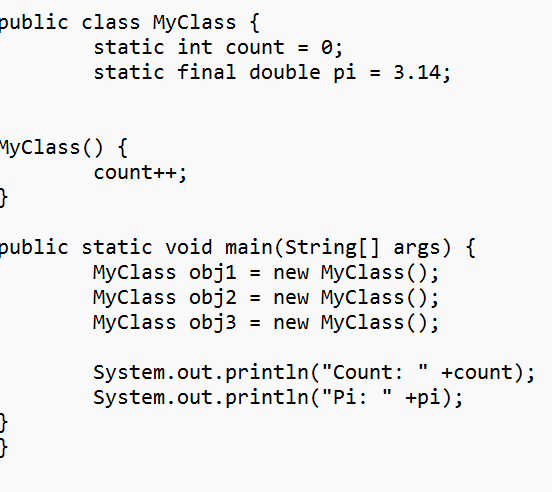
**Aim:**

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:**

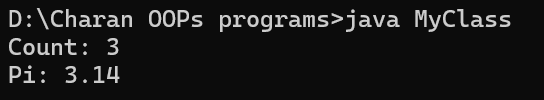
|  |
| --- |
| **MyClass** |
| **-count: int (static)**  **-pi: double (static, final)** |
| **+MyClass()**  **+main(args: String[]):void** |

**Code:**

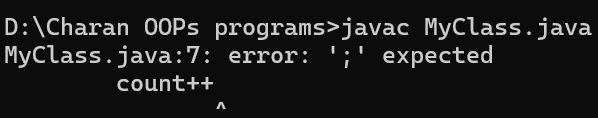
****

**Output:**

Positive case:

****

Negative case:



**Error:**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | rectification |
| 1 | error: ';' expected  count++ | Count++; |
| **2** | : error: cannot find symbol  System.out.println("Pi: " +pi);  ^  symbol: variable pi | pi |

**WEEK 5:**

**Program1 :**

**Aim:**

Create a calculator using the operations including addition, subtraction

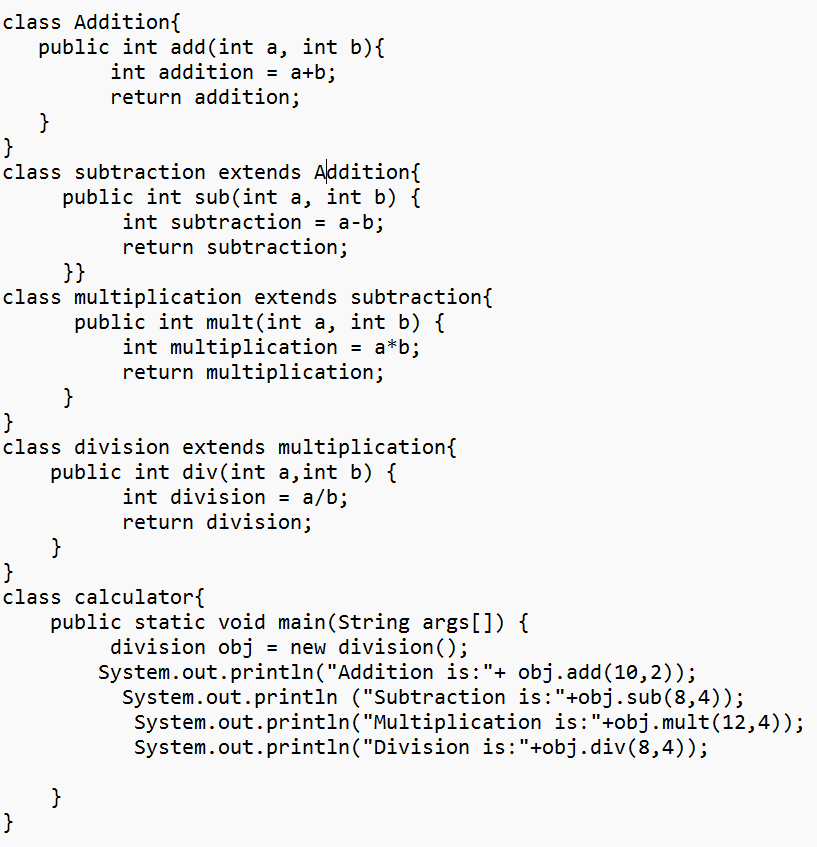
Multiplication and division using multilevel inheritance and display the desired

Output.

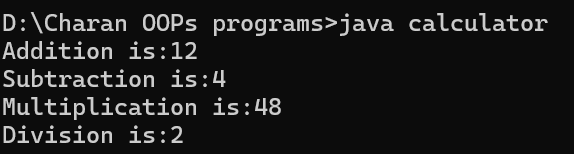
**Class diagram:**



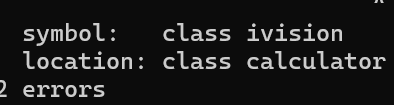
**Code:**



**Output :**

**Positive case:  
**

**Negative case:**

****

**Error :**

|  |  |  |  |
| --- | --- | --- | --- |
| 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:**

**Inheritance:**

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 Inheritance:**

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 inheritance.

**Extends:**

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

**Program 2:**

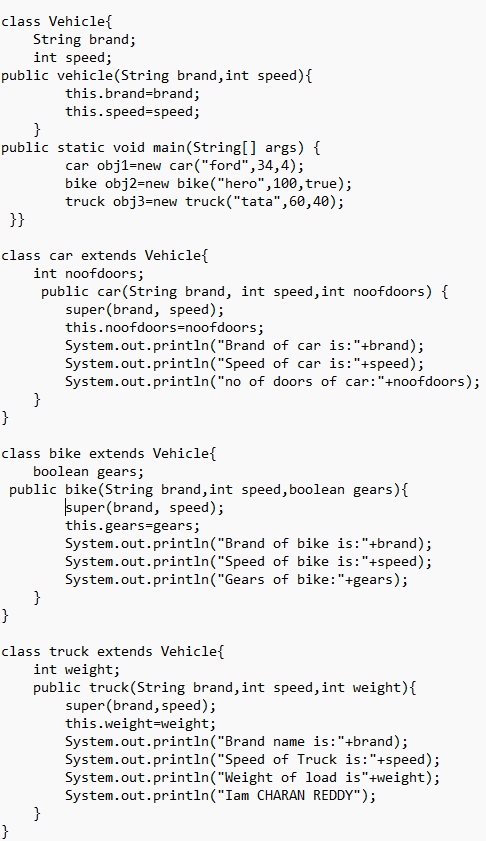
**Aim:** A vehicle rental company wants to develop a system that maintains Information about different types of vehicles available for rent. The Company rents out cars, bikes and truck and they need a program to Store details about each vehicle, such as brand and speed.

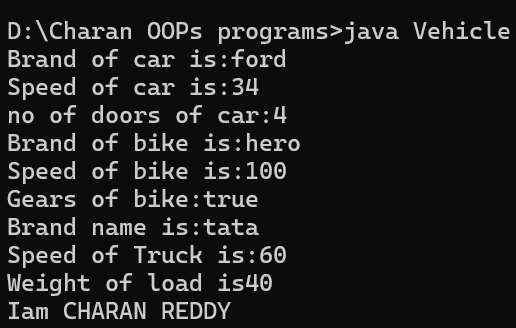
Cars should have an additional property: number of doors. Bikes should have a property indicating whether they have gears or not. the system should also include a function to display details about each vehicle and indicate when a vehicle is starting.

**Class diagram:**

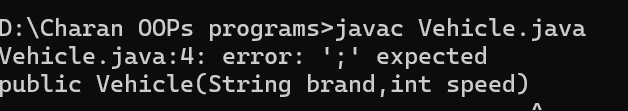


**Code:**

****

**Output:  
positive case:  
**

**Negative case:**

****

**Errors :**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | Rectification |
| 1 | Syntax missing | ; |
| 2 | Syntax missing | { |

**Important points:**

**Hierarchical Inheritance:**

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

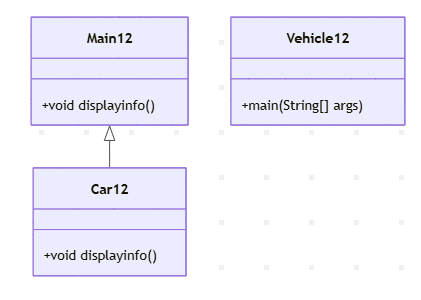
Single parent class

**WEEK-6:**

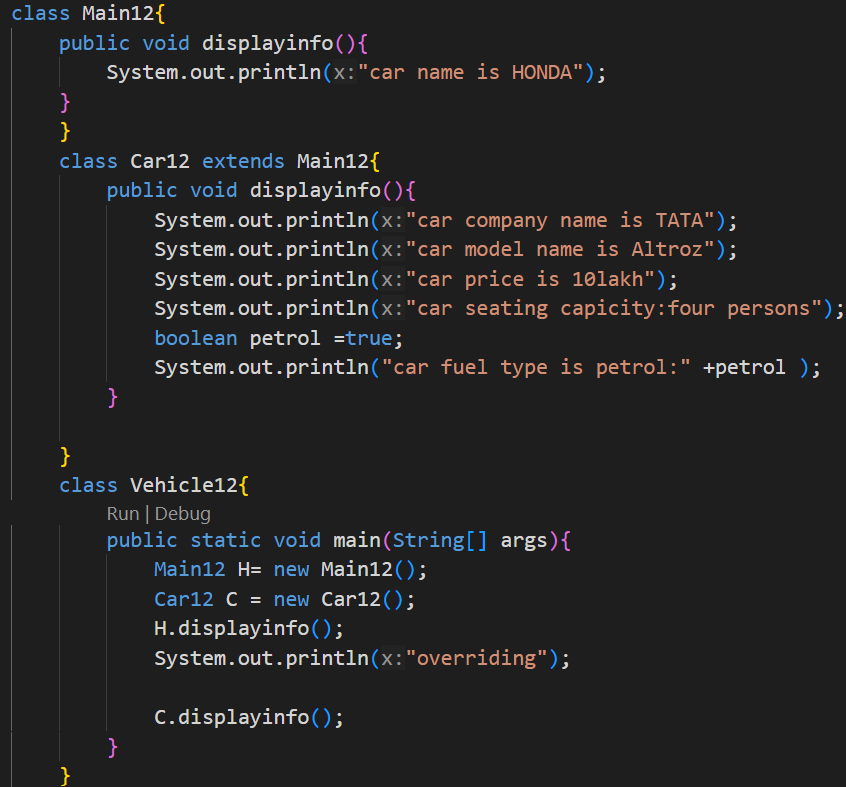
**Program 1:**

**Aim:** To create a vehicle class with a method display info. Override this method in the car subclass to provide specific information about car(car company, car model, car price, seating capacity , petrol or not [Boolean]).

**Class diagram:**

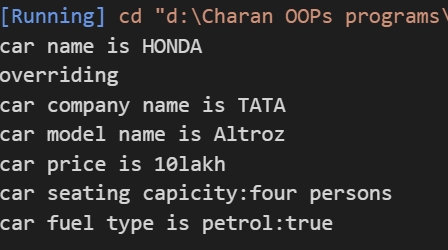
****

**Code:**

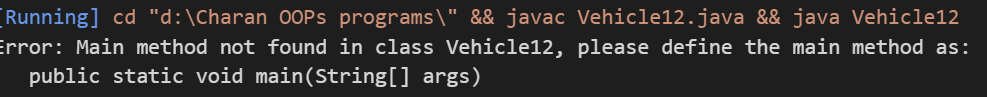


**Output:**

**Positive output:**



**Negative output:**



**Errors:**

|  |  |  |
| --- | --- | --- |
| S.no | Error | Rectification |
| 1 | Main method not found in class vehicle12 | Keep the correct one |
| 2 | displayinfo | Displayinfo() |

**Important points:**

**Override:**

In Java, overriding occurs when a subclass (child class) provides its own specific implementation of a method that is already defined in its superclass (parent class).

It is a run time polymorphism.

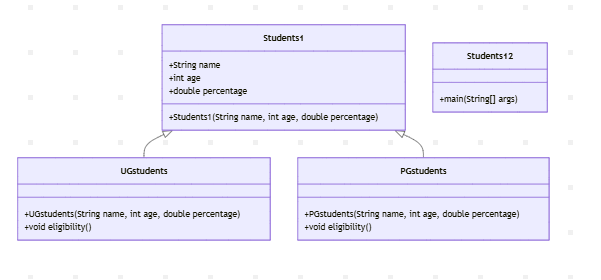
**Program 2:**

**Aim:** A college is developing an automated admission system that verifies students eligibility for under graduate(UG) and post graduate(PG) programs. Each program has different eligibility criteria based on these students percentage in their previous qualification.

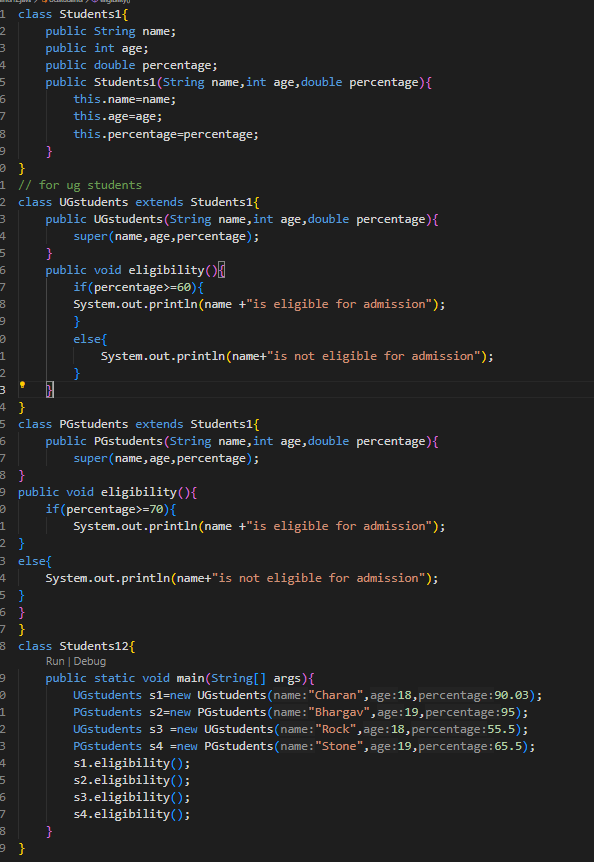
UG admissions require a minimum of 60%.

PG admissions require a minimum of 70%.

**Class diagram:**

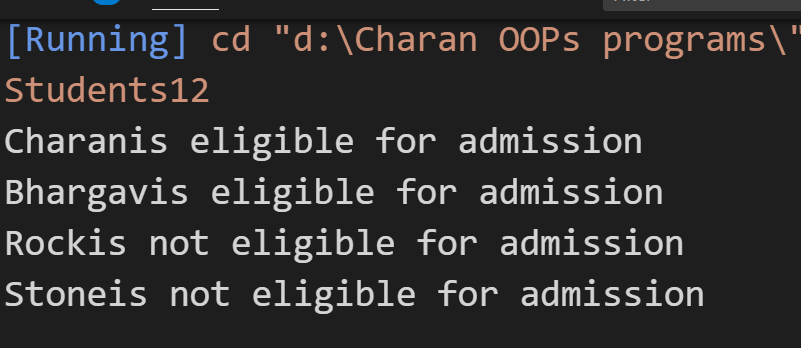
****

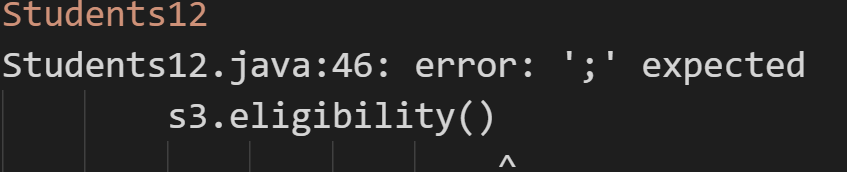
**Code:**

****

**Output:**

**Positive output:**

****

**Negative output:  
**

**Error :**

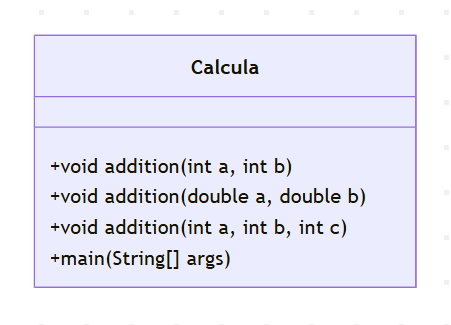
|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| **1** | Syntax missing | **;** |
| **2** | Construstor syntax error | **{}** |

**Program 3:**

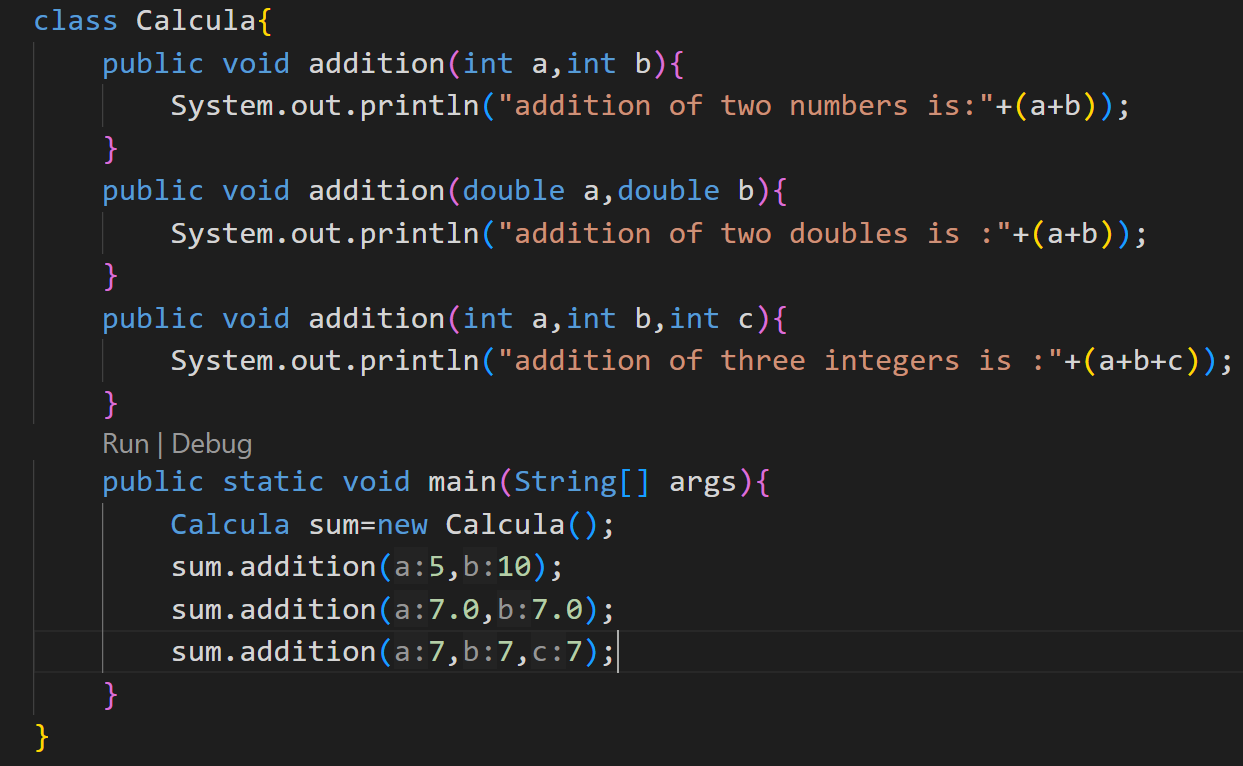
**Aim:** create a calculator class with overloaded methods to perform addition

1.add two integers 2.add two doubles 3.add three integers

**Class diagram:**

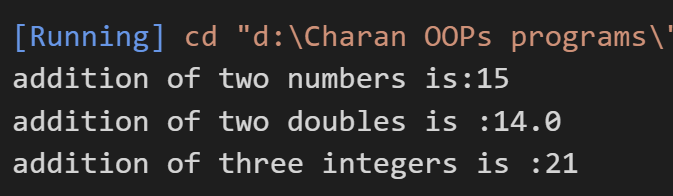
****

**Code:**

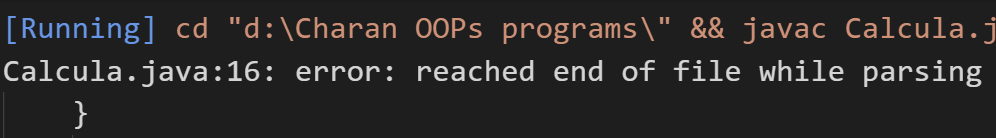
****

**Output:**

**Positive case:**

****

**Negative case:**

****

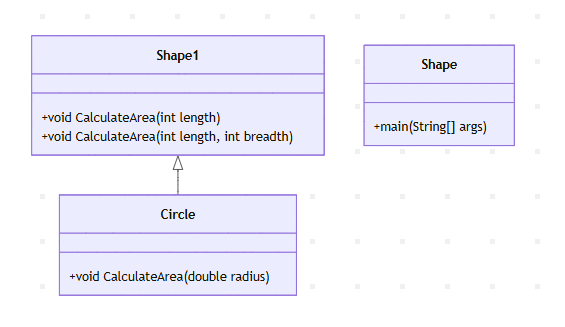
**Error :**

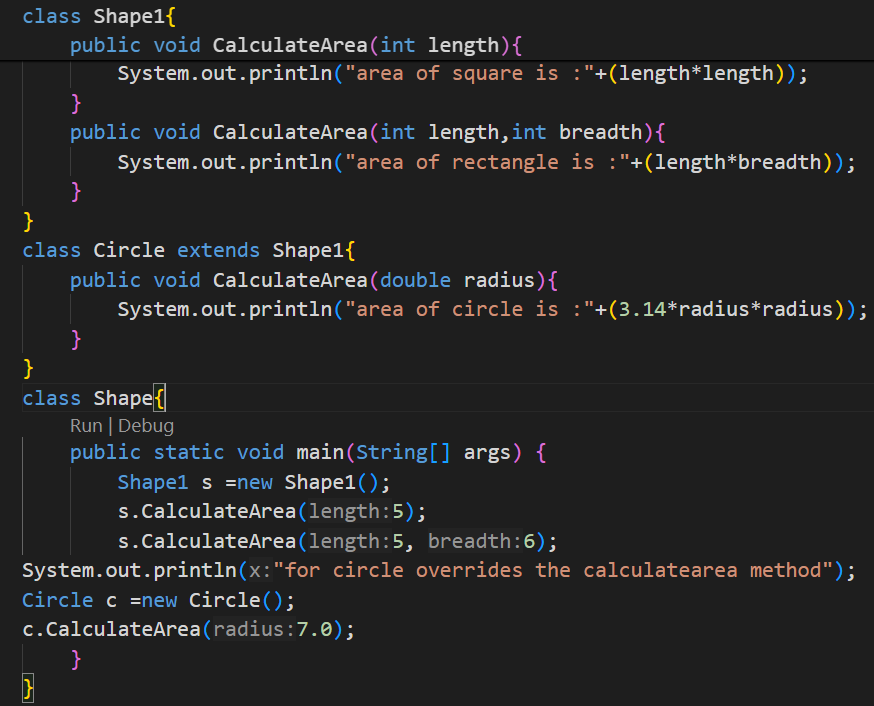
|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| **1** | Calcula.java:16: error: reached end of file while parsing      } | **}** |
| **2** | syntax missing | **;** |

**Program 4:**

**Aim:** create a shape class with a method calculatearea that is overloaded for different shapes like square and rectangle then create a subclass circle that extends calculatearea method for a circle

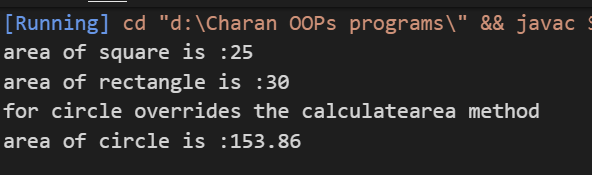
**Class diagram:**

**  
code:**

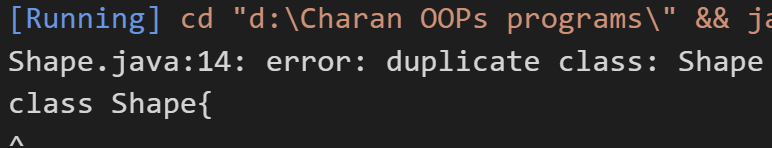
****

**Output:**

**Positive case:**

****

**Negative case:**

****

**Error :**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| **1** | Shape{ | Shape1{ |
| **2** | Syntax missing | **;** |

**Important points:**

**1.overloading :**

In Java, method overloading allows you to define multiple methods with the same name within the same class, as long as they have different parameter lists (either different number of parameters, different data types of parameters, or different order of parameters).

It is a compile time polymorphism.

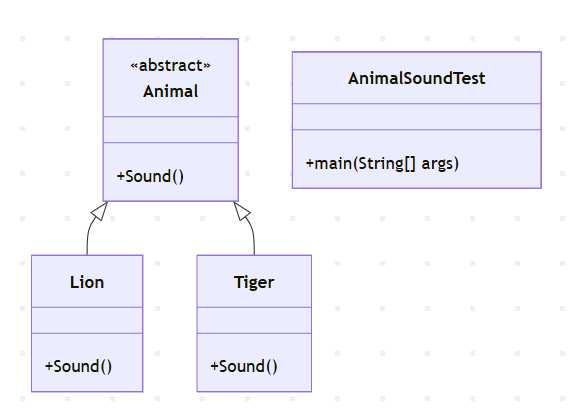
**Week 7:**

**Program 1:**

**Aim:**write a java program to create an abstract class Animal with an abstract method called

Sound. Create a subclass as lion and tiger that extends animal class and implement the sound method to make a specific sound for each animal

**Class diagram:**

****

**Code:**

abstract class Animal {

    abstract void Sound();

}

class Lion extends Animal {

    @Override

    void Sound() {

        System.out.println("Name:S.charan reddy,roll no:24321,Section:CSE-A");

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

    }

}

class Tiger extends Animal {

    @Override

    void Sound() {

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

    }

}

public class AnimalSoundTest {

    public static void main(String[] args) {

        Animal lion = new Lion();

        Animal tiger = new Tiger();

        lion.Sound();

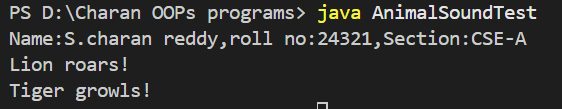
        tiger.Sound();

    }

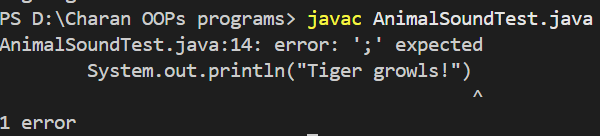
}

**Output:**

**Positive case:**

****

**Negative case:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| 1 | Syntax missing | ; |
| 2 | Class name error | Animal |

**Important points:**

**Abstract class:**

An abstract class in Java is a class that cannot be instantiated (no objects can be created from it directly).

**Abstract method:**

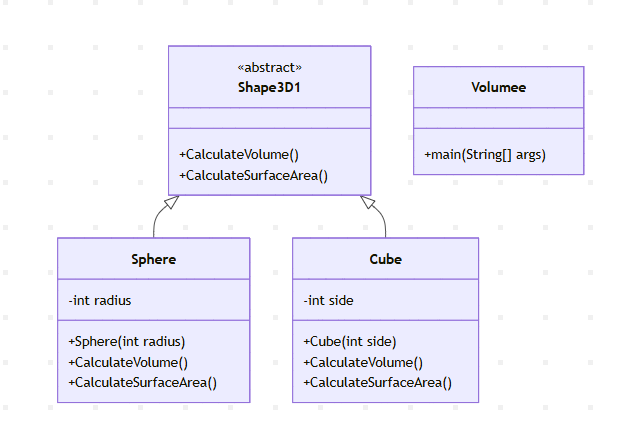
An abstract method in Java is a method declared without a body (implementation), meaning it doesn't have code within its curly braces {}.

**Program 2:**

**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.

**Class diagram:**

****

**Code:**

abstract class Shape3D1 {

    public abstract void CalculateVolume();

    public abstract void CalculateSurfaceArea();

}

class Sphere extends Shape3D1 {

    int radius;

    Sphere(int radius) {

        this.radius = radius;

    }

    public void CalculateVolume() {

        System.out.println("Volume of sphere is: " + (4.0 / 3.0) \* Math.PI \* radius \* radius \* radius);

    }

    public void CalculateSurfaceArea() {

        System.out.println("Surface area of sphere is: " + 4 \* Math.PI \* radius \* radius);

    }

}

class Cube extends Shape3D1 {

    int side;

    Cube(int side) {

        this.side = side;

    }

    public void CalculateVolume() {

        System.out.println("Volume of cube is: " + side \* side \* side);

    }

    public void CalculateSurfaceArea() {

        System.out.println("Surface area of cube is: " + 6 \* side \* side);

    }

}

class Volumee{

    public static void main(String[] args){

        System.out.println("Name:S.charan reddy,Roll no:24321,Section:CSE-A");

        Sphere sp = new Sphere(5);

        Cube c = new Cube(4);

        System.out.println("    ");

        sp.CalculateSurfaceArea();

        System.out.println("    ");

        sp.CalculateVolume();

        System.out.println("    ");

        c.CalculateSurfaceArea();

        System.out.println("    ");

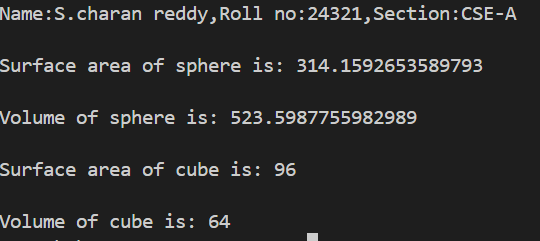
        c.CalculateVolume();

    }

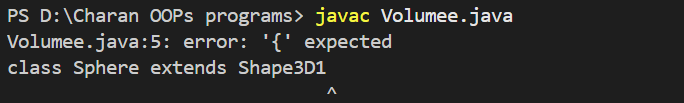
}

**Output:**

**Positive case:**

****

**Negative case:**

****

**Errors :**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| **1** | { missing | { |
| **2** | Syntax error | **;** |

**Program 3:**

**Aim:**

Write a java program using an abstract class to define a method for pattern printing Create an abstract class named pattern printer with an abstract method printpattern(int n) and a concrete method to display the pattern title.

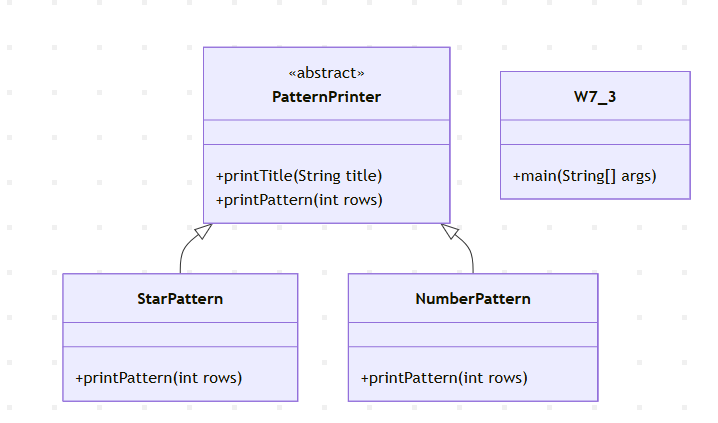
Implement two subclasses:

1) Star pattern - Prints a right-angled triangle of stars(\*).

2) Number pattern - Prints a right- angled triangles of increasing numbers.

In the main() method, create Objects of both subclass and print the patterns for given number of rows.

**Class diagram:**

****

**Code:**

abstract class PatternPrinter {

    public void printTitle(String title) {

        System.out.println("=== " + title + " ===");

    }

    public abstract void printPattern(int rows);

}

class StarPattern extends PatternPrinter {

    @Override

    public void printPattern(int rows) {

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

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

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

            }

            System.out.println();

        }

    }

}

class NumberPattern extends PatternPrinter {

    @Override

    public void printPattern(int rows) {

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

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

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

            }

            System.out.println();

        }

    }

}

public class W7\_3 {

    public static void main(String[] args) {

        System.out.println("Name:S.charan reddy,Roll no:24321,Section:CSE-A");

        int rows = 5;

        PatternPrinter star = new StarPattern();

        star.printTitle("Star Pattern");

        star.printPattern(rows);

        System.out.println();

        PatternPrinter number = new NumberPattern();

        number.printTitle("Number Pattern");

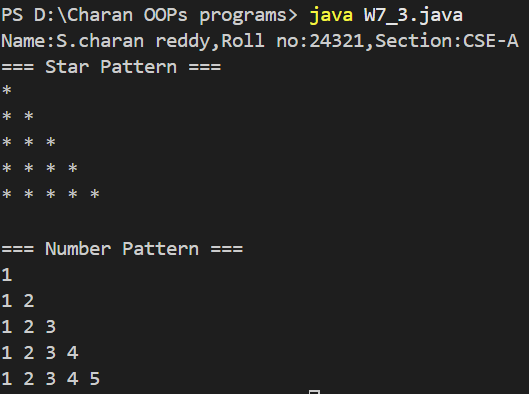
        number.printPattern(rows);

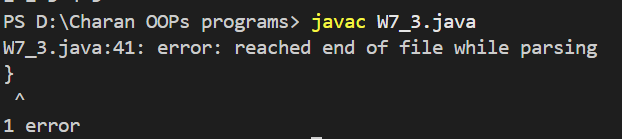
    }

}

**Output:**

**Positive case:**

****

**Negative case:  
**

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| **1** | } missing | } |
| **2** | Semi column missing | ; |

**Important points:**

**Abstract class:**

An abstract class in Java is a class that cannot be instantiated (no objects can be created from it directly).

**Abstract method:**

An abstract method in Java is a method declared without a body (implementation), meaning it doesn't have code within its curly braces {}.

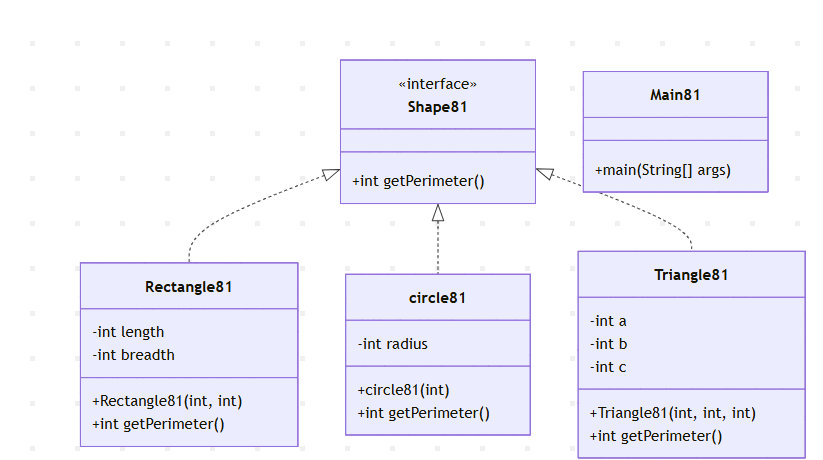
**Week 8 :**

**Program 1:**

**Aim:** write a java program to create an interface shape with the getperimeter() method. Create three classes rectangle, circle and triangle that implements the shape interface.

Implements the getperimeter() method of three classes.

**Class diagram:**

****

**Code:**

import java.util.\*;

interface Shape81{

    abstract int getPerimeter();

}

class Rectangle81 implements Shape81{

    int length, breadth;

    Rectangle81(int length, int breadth){

        this.length = length;

        this.breadth = breadth;

    }

    @Override

    public int getPerimeter(){

        return 2\*(length + breadth);

    }

}

class circle81 implements Shape81{

    int radius;

    circle81(int radius){

        this.radius = radius;

    }

    @Override

    public int getPerimeter(){

        return (int) (2\*3.14\*radius);

    }

}

class Triangle81 implements Shape81{

    int a, b, c;

    Triangle81(int a, int b, int c){

        this.a = a;

        this.b = b;

        this.c = c;

    }

    @Override

    public int getPerimeter(){

        return a+b+c;

    }

}

class Main81{

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter length of Rectangle:");

        int length = scanner.nextInt();

        System.out.println("Enter breadth of Rectangle:");

        int breadth = scanner.nextInt();

        Rectangle81 r = new Rectangle81(length, breadth);

        System.out.println("Enter radius of Circle:");

        int radius = scanner.nextInt();

        circle81 c = new circle81(radius);

        System.out.println("Enter side a of Triangle:");

        int a = scanner.nextInt();

        System.out.println("Enter side b of Triangle:");

        int b = scanner.nextInt();

        System.out.println("Enter side c of Triangle:");

        int c1 = scanner.nextInt();

        Triangle81 t = new Triangle81(a, b, c1);

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

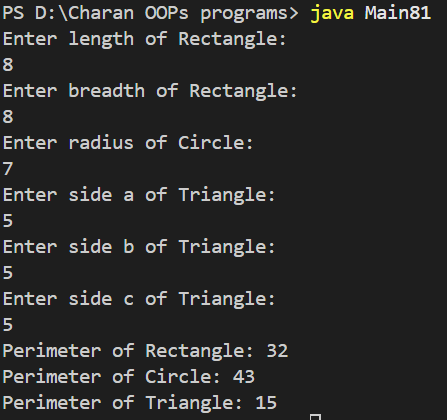
        System.out.println("Perimeter of Circle: " + c.getPerimeter());

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

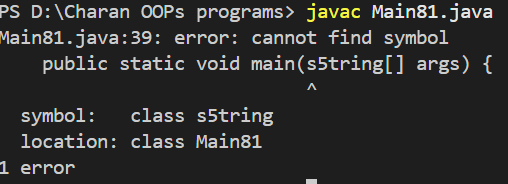
    }

}

**Output:**

**Positive case:  
**

**Negative case:**

****

**Errors:**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| **1** | S5tring | String |
| **2** | Syntax error | ; |

**Important points:**

**Interface:**

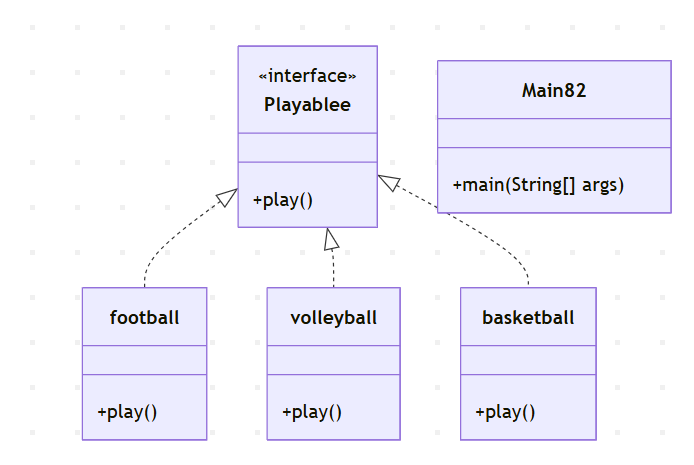
In Java, an interface is a blueprint of a class that specifies a contract for what a class should do. It defines a set of abstract methods (methods without a body) that any class implementing the interface must provide concrete implementations for. Interfaces are declared using the interface keyword.

**Program 2:**

**Aim:**

Write a java program to create an interface playable with a method play() that takes no arguments and return void. Create three classes football, volleyball and basketball that implement playable interface and override the paly() method to play respective sports.

**Class diagram:**

****

**Code:**

interface Playablee {

void play();

}

class football implements Playablee {

public void play() {

System.out.println("Playing the football");

}

}

class volleyball implements Playablee {

public void play() {

System.out.println("Playing the volleyball");

}

}

class basketball implements Playablee {

public void play() {

System.out.println("Playing the basketball");

}

}

class Main82 {

public static void main(String[] args) {

Playablee football = new football();

Playablee volleyball = new volleyball();

Playablee basketball = new basketball();

football.play();

volleyball.play();

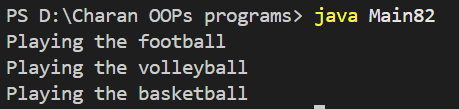
basketball.play();

}

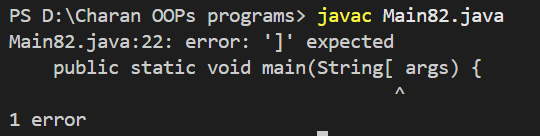
}

**Output:**

**Positive case:**

****

**Negative case:**

****

**Errors :**

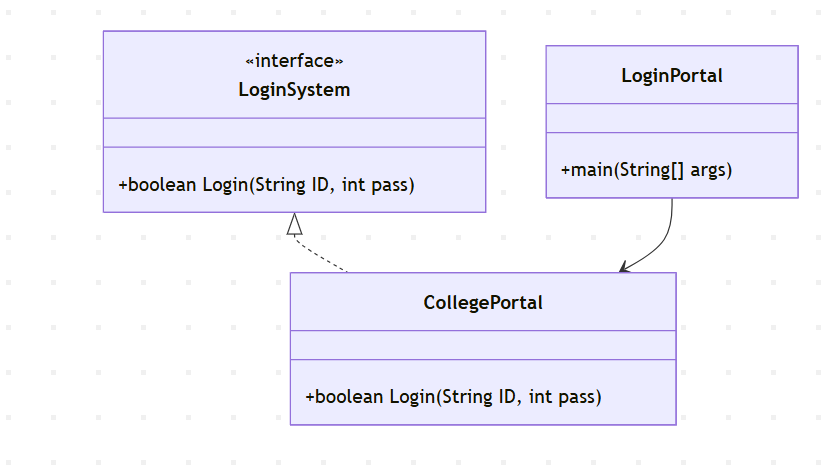
|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| **1** | String[ | String[] |
| **2** | Syntax missing | ; |

**Program 3:**

**Aim:**

Write a java program to implement a login system using interfaces

**Class diagram:**

****

**Code:**

interface LoginSystem {

    boolean Login(String ID, int pass);

}

class CollegePortal implements LoginSystem {

    public boolean Login(String ID, int pass) {

        if ((ID=="charan") && (pass==24321)){

            System.out.println("Login Successful..!");

            return true;

        }else {

            System.out.println("Invalid ID or Password");

            return false;

        }

    }

}

class LoginPortal {

    public static void main(String[] args) {

        CollegePortal CP = new CollegePortal();

        System.out.println("Name:S.charan reddy,Roll no:24321,Section:CSE-A");

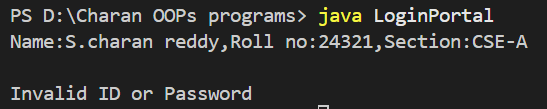
        System.out.println("    ");

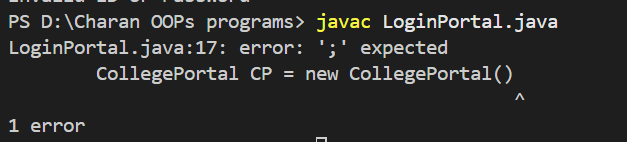
        CP.Login("charan", 24321);

    }

}

**Output:**

**Positive case:  
**

**Negative case:  
**

**Errors :**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Error message** | **Rectification** |
| **1** | Error: ‘;’ expected | ; |

**Important points :**

**Interface**: LoginSystem defines a method contract Login(String, int).

**Interface Implementation**: CollegePortal implements LoginSystem and provides the logic for the Login method.

**Conditional Logic**: Checks if provided credentials match hardcoded values.

**Main Method**: LoginPortal contains main to run the program.