**23CSE111**

**Object Oriented Programming**



**Department of Computer Science Engineering Amrita School of Engineering**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Name: Mehuli Sarkar**

**Verified By Roll No:AV.SC.CSE24150**

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Questions | Page No | Signature |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |

**Week-1**

**JAVA:**

Java is a high level, class based, object oriented programming language that is widely used across various operating systems.

**Installation of JAVA in Computer:**

**Aim:**

Solving computational programming using JAVA.

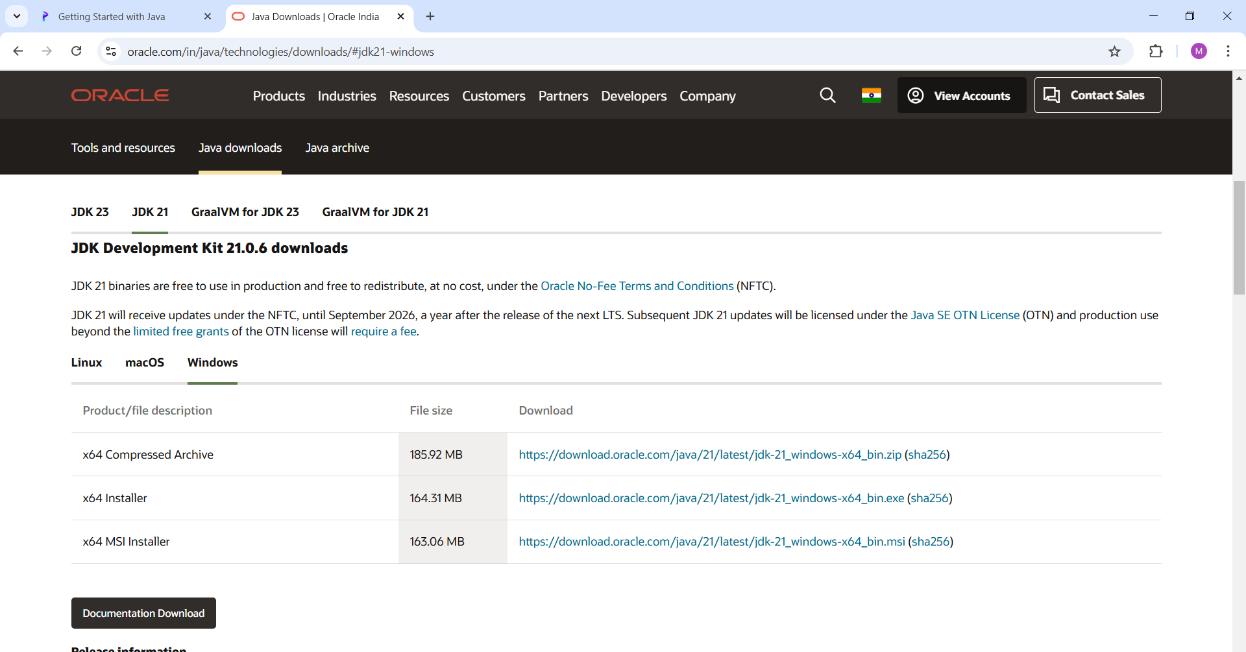
**Procedure:**

Follow the steps below to install Java on Windows:

1. Download JDK(Java Development Kit)
2. Run the Installer
3. Configure Environment Variables
4. Verify Installation

1.Download JDK:

Go to the official oracle website to download the JDK. Choose x64 MSI Installer on the windows tab and click on download link.

****

2. Run the Installer:

Now, go to your downloads folder and run the installer you just downloaded.

3. Configure Environment Variables:

After installation, you will need to tell your system where to find Java. This is done by setting environment variables.

Locate JDK Path:

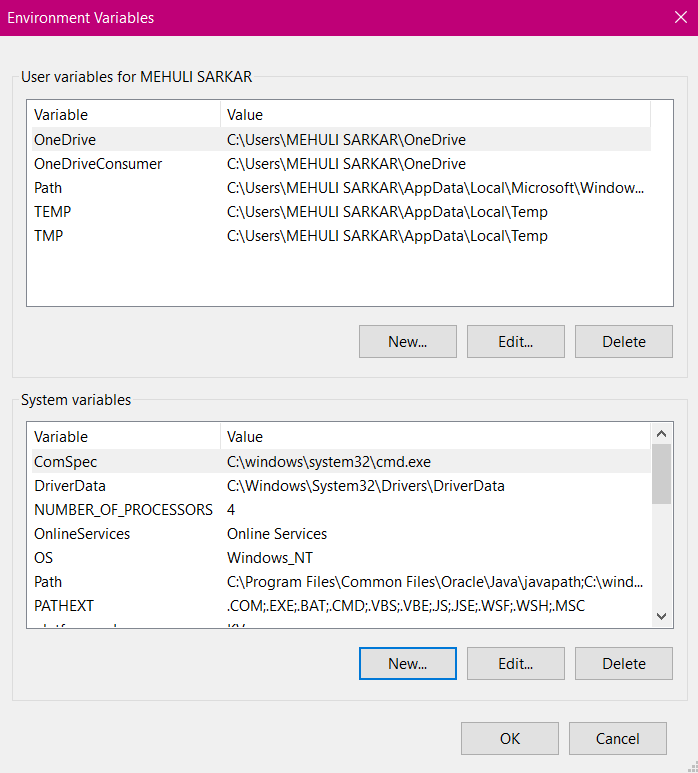
Navigate through your file explorer to reach the JDK installation directory. Normally, it is located at -

C:\Program Files\Java\jdk-22\bin

Copy this path.

Access Environment Variables:

Search environment variable on the terminal. In system properties, click on environment variables. You will be prompted to the screen below.



Update the Path Variable:

Find the Path variable in the System variables section and click on edit. Then, click New and paste your JDK bin path.

Finally, click Ok to close each window.

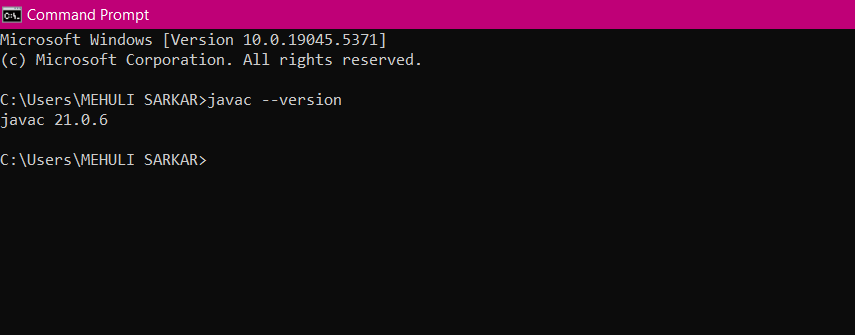
Set JAVA\_HOME Variable:

Back in the environment variables window, under the system variables section, click New to create a new variable.

Now, name the variable  and set its value to the path of the JDK folder directory. Close all the dialogues with the Ok button.

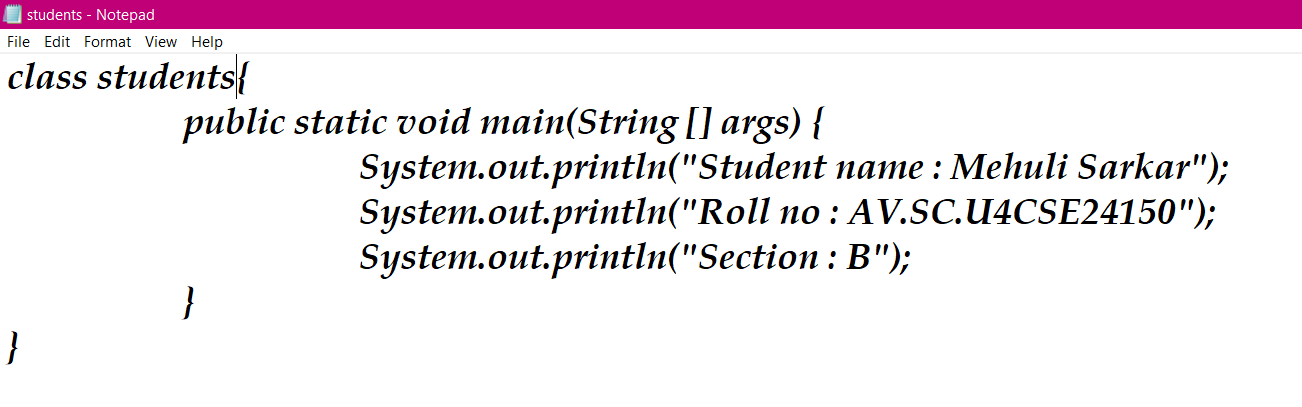
1. Verify Installation:

After the installation, you can verify whether Java is installed by using the following command in the command prompt.

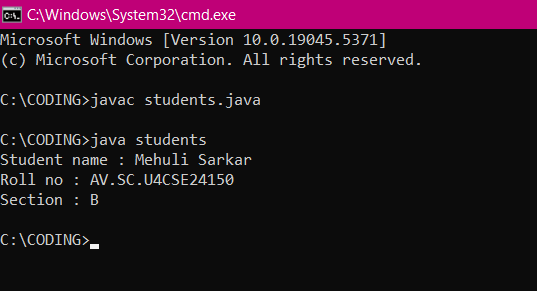


If Java is installed successfully, it will print the version information; otherwise, it will produce an error message indicating that the command is not recognized.

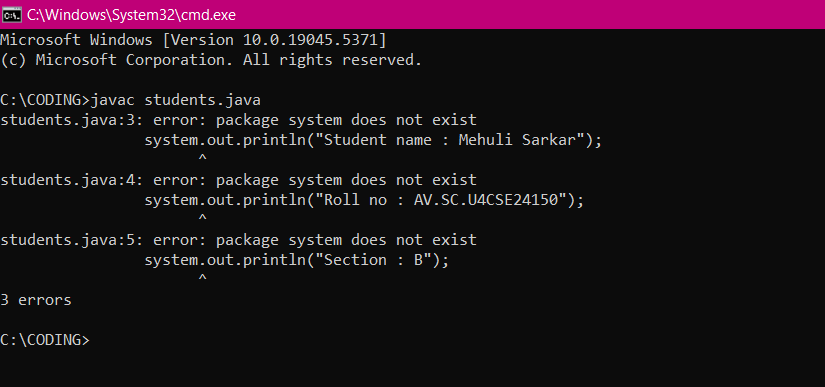
**JAVA Program:**

****

Output:

****

Error:

****:

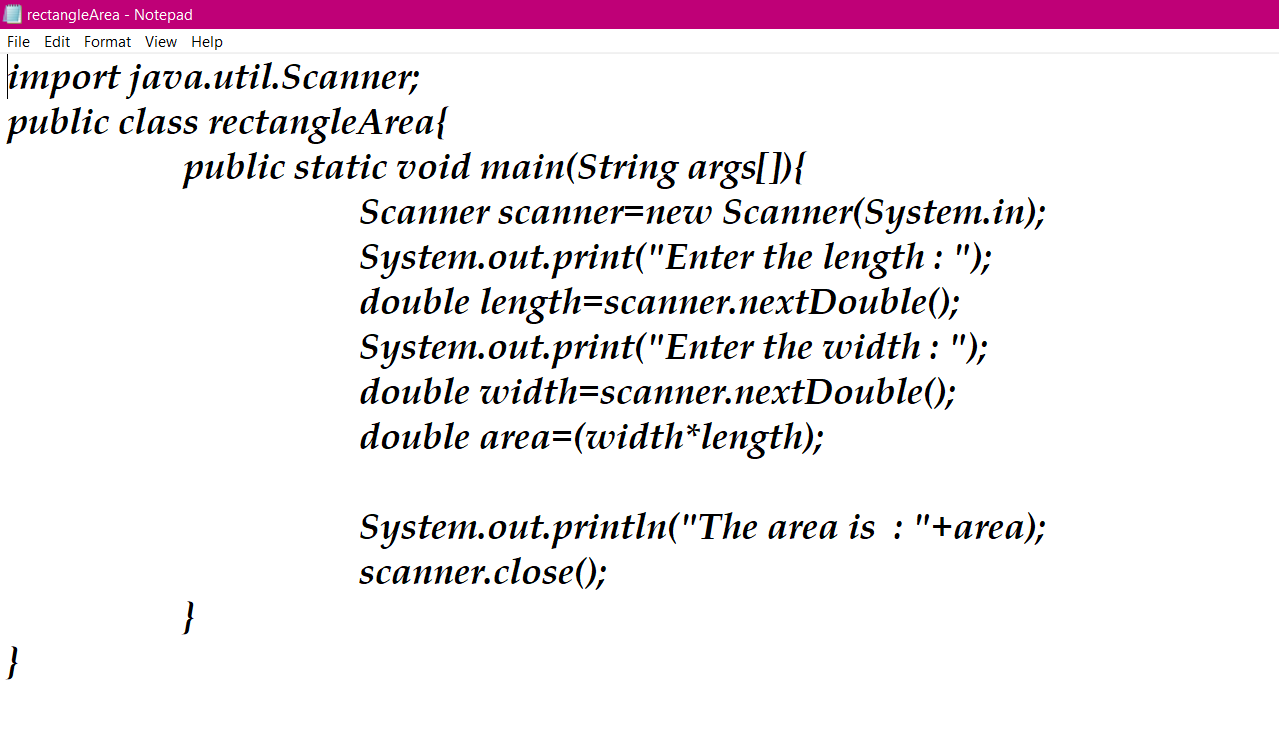
**Week-2**

**1.****Program 1:**

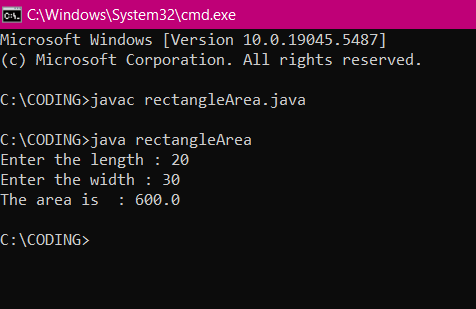
**AIM:**

To find area of rectangle input is given by user

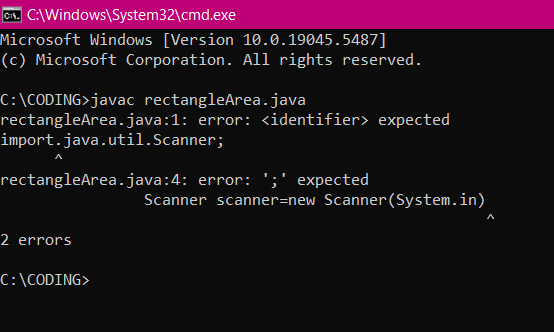
**CODE:**

****

**OUTPUT:**

****

**ERROR TABLE:**

****

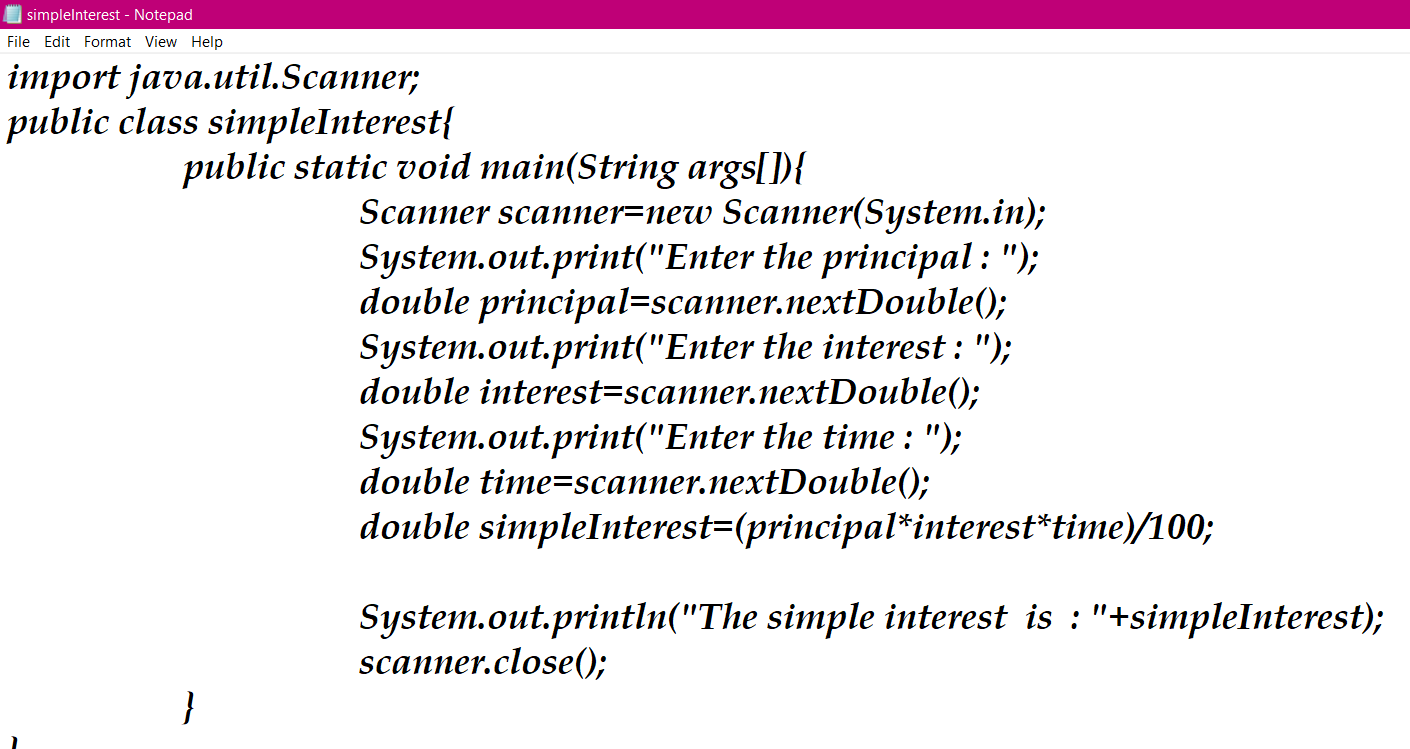
|  |  |
| --- | --- |
| **ERROR** | **RECTIFICATION** |
| Missed semicolon | Added semicolon |

**2. Program 2:**

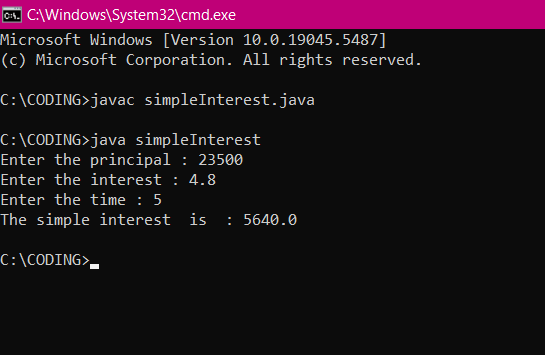
**AIM:**

To find simple interest input given by user

**CODE:**

****

**OUTPUT:**

****

**ERROR TABLE:**

-----

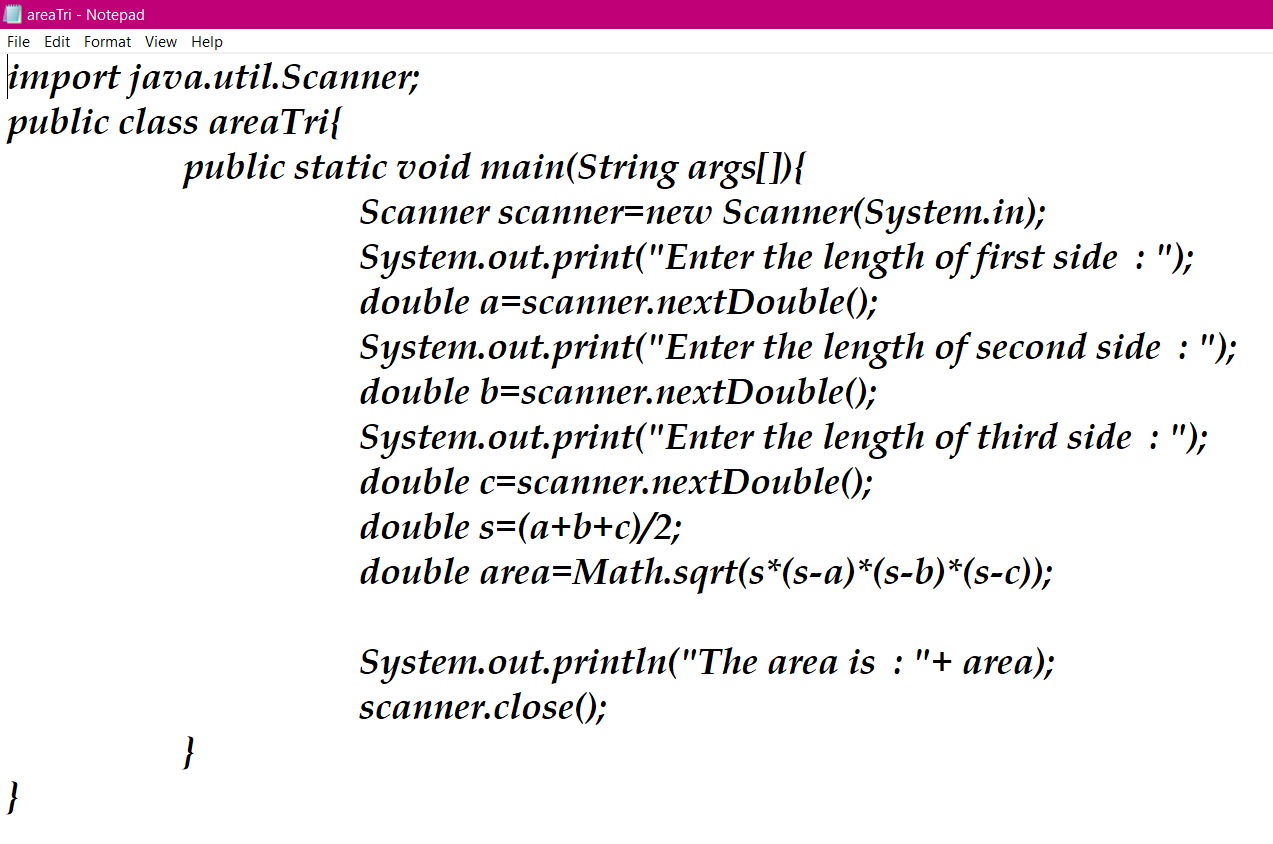
|  |  |
| --- | --- |
| **ERROR** | **RECTICATION** |
| No error | ---- |

**3. Program 3:**

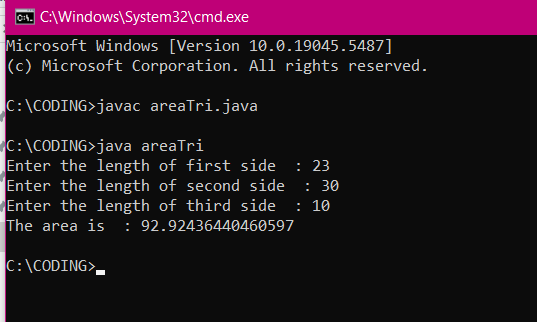
**AIM:**

To find area of triangle input is given by user

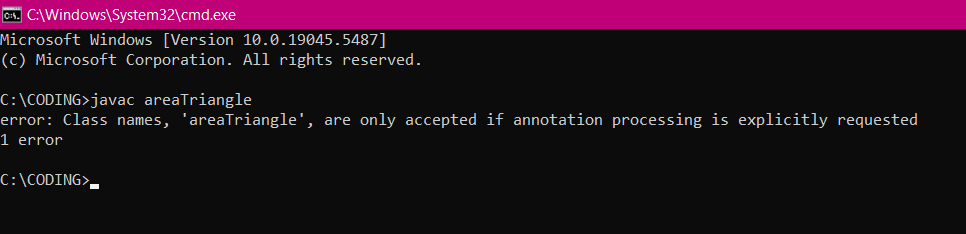
**CODE:**

****

**OUTPUT:**

****

**ERROR TABLE:**

****

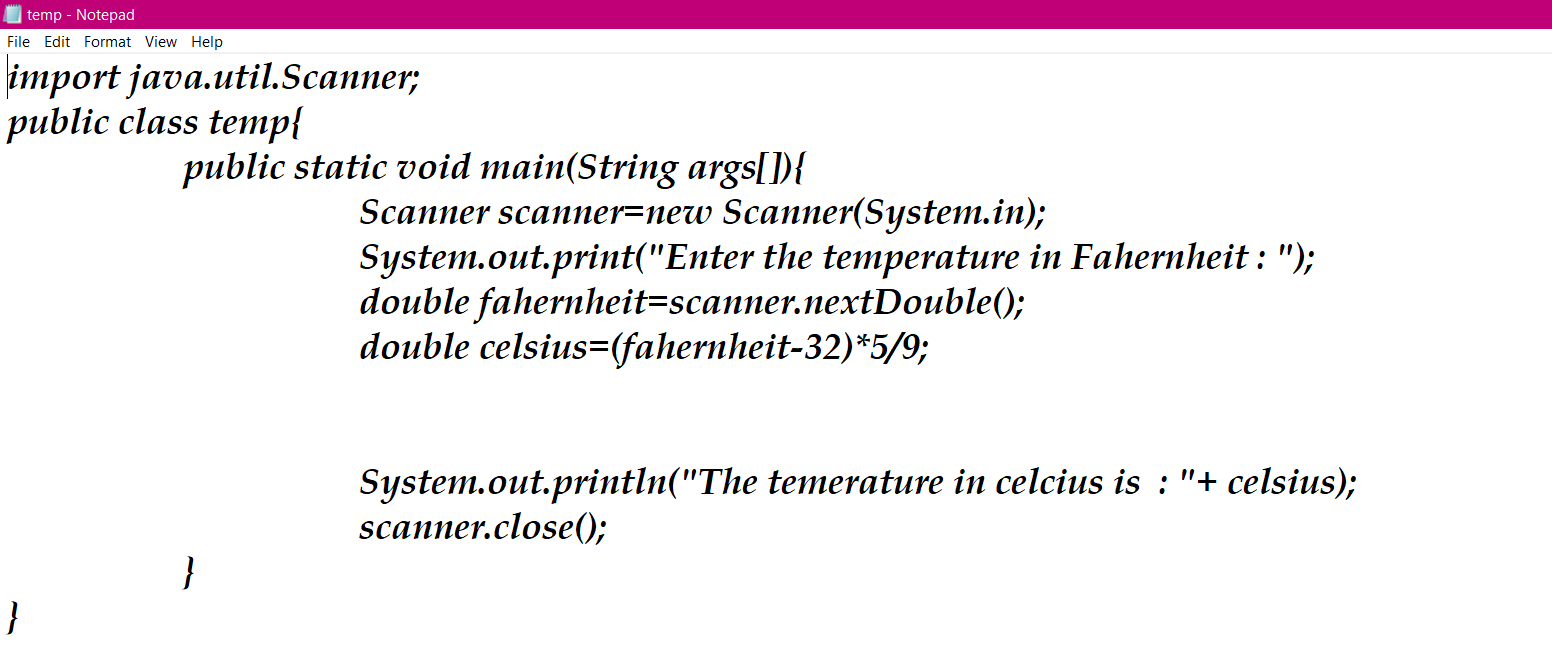
|  |  |
| --- | --- |
| **ERROR** | **RECTIFICATION** |
| Wrong class name | Changed the class name |

**4. Program 4:**

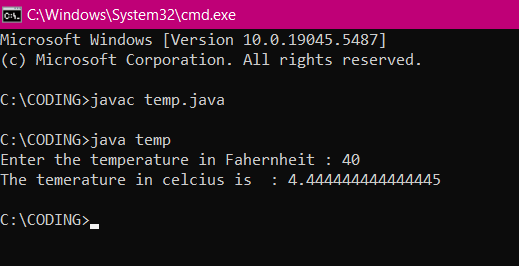
**AIM:**

To convert Fahrenheit to Celsius input is given by user

**CODE:**

****

**OUTPUT:**

****

**ERROR TABLE:**

----

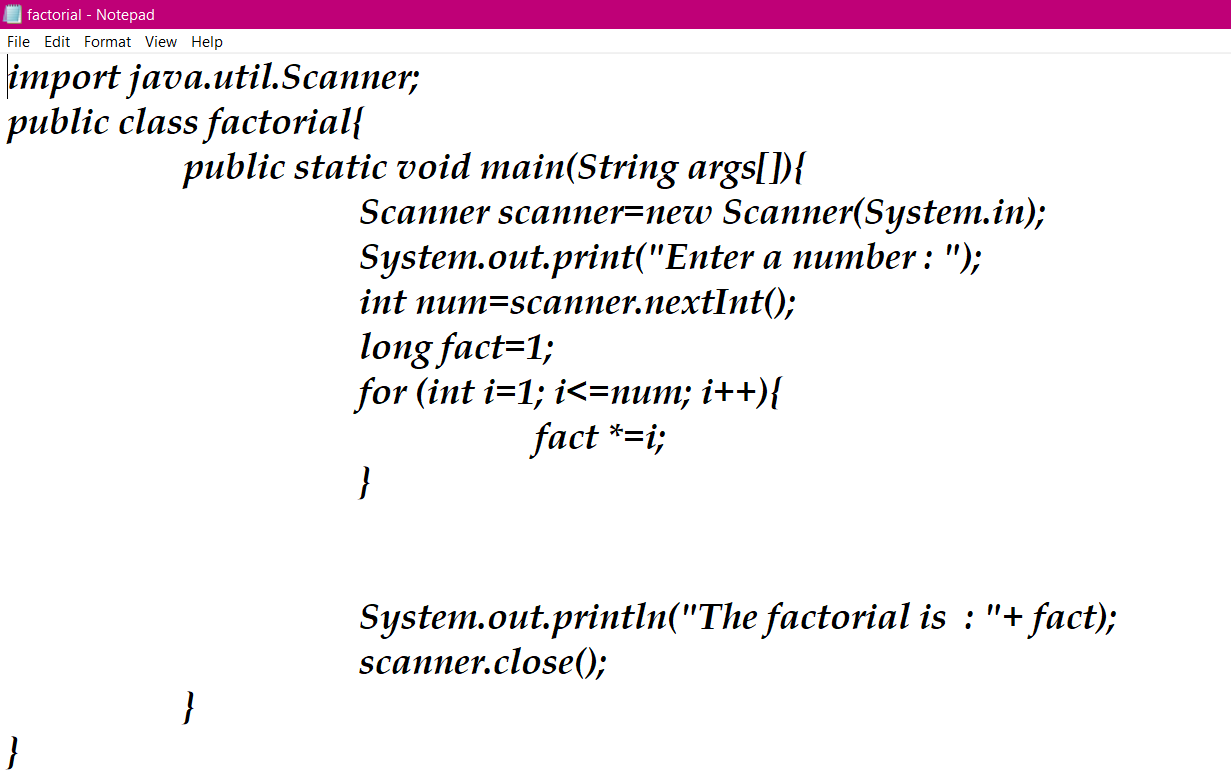
|  |  |
| --- | --- |
| **ERROR** | **RECTIFICATION** |
| No error | ---- |

**5. Program 5:**

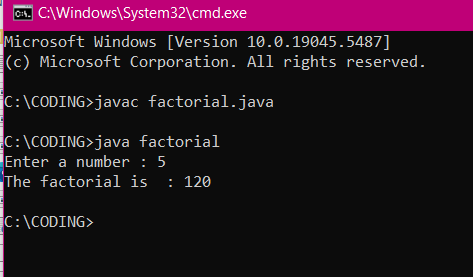
**AIM:**

To find factorial of n

**CODE:**

****

**OUTPUT:**

****

**ERROR TABLE:**

-----

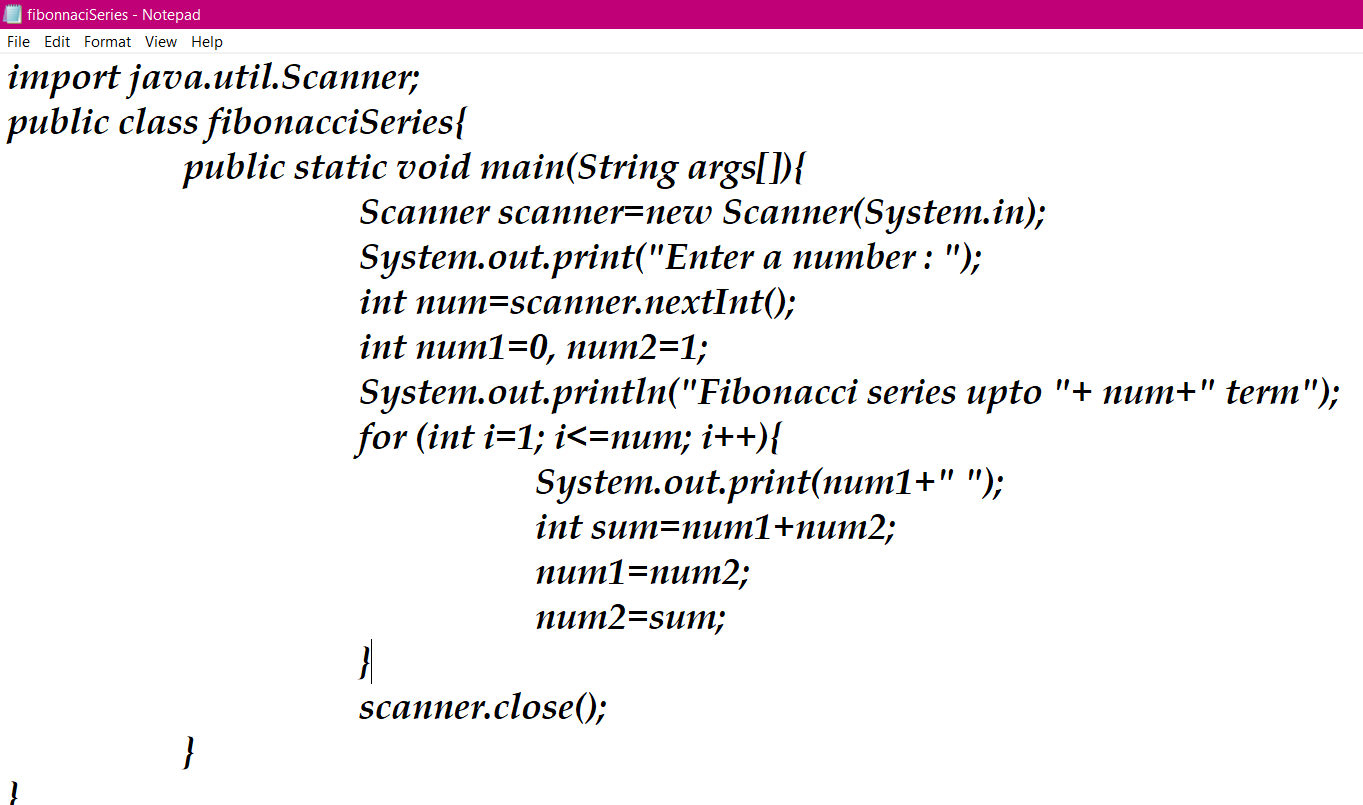
|  |  |
| --- | --- |
| **ERROR** | **RECTIFICATION** |
| No error | ---- |

**6. Program 6:**

**AIM:**

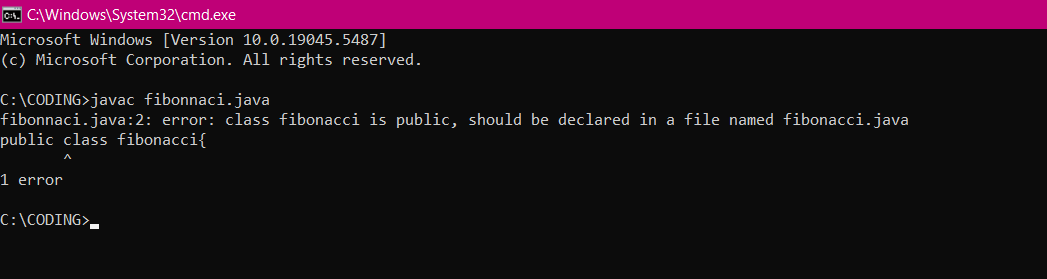
To find Fibonacci sequence

**CODE:**

****

**OUTPUT:**

**ERROR TABLE:**

****

|  |  |
| --- | --- |
| **ERROR** | **RECTIFICATION** |
| Class is public, should be declared in a file named fibonacci.java |  |

**WEEK – 3:**

**PROGRAM- 1:**

**Q. Write a java program through the following instruction:**

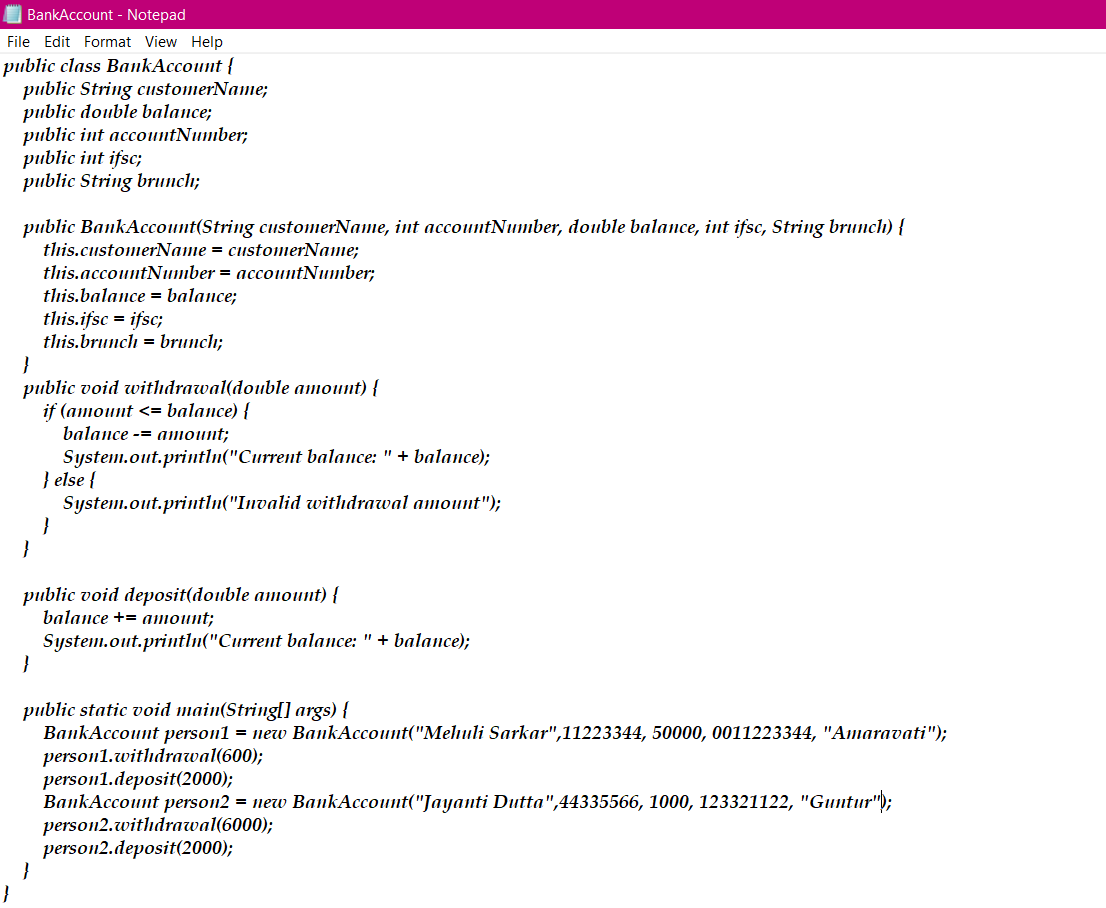
**Create a class named bankAccount with methods deposit and withdraw, where deposit method should accept a parameter and when this method is called the deposited amount should be added to current balance. In addition to that when a withdraw method is called it has to verify if the withdraw amount is less than the current balance, if not then display a message saying insufficient funds.**

**Use the constructor to display the details of the customer.**

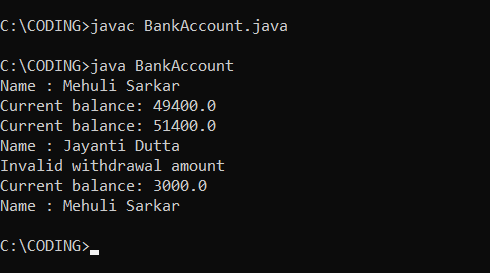
**AIM:**

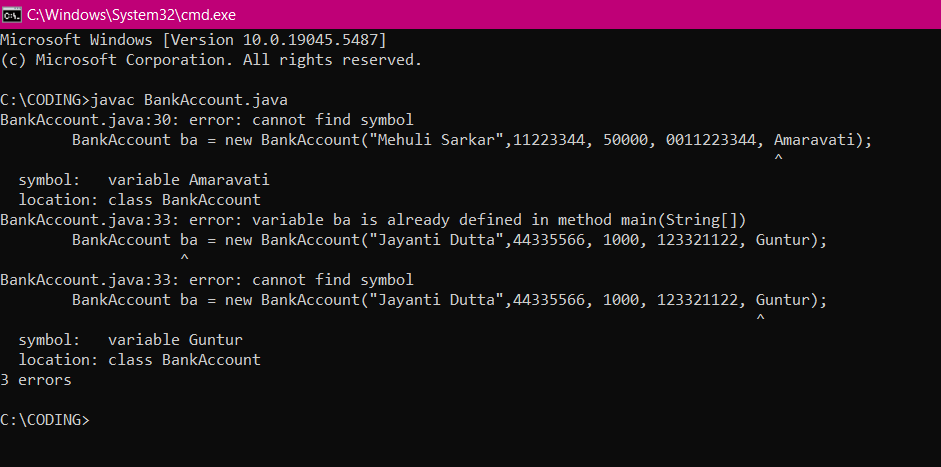
To create a class bankAccount with methods deposit() and withdraw() . create two subclasses savingsaccount and checkingaccount override the withdraw () method in each subclass to impose different withdrawal limits and fees.

**CODE:**



**Output:**



**ERROR:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Wrong data type of string variable. | 1. Write correct data type. |

**ERROR TABLE:**

**IMPORTANT POINTS:**

1. The condition inside the if statement must be correct.
2. It explains that if the withdrawal money is less than the money in the bank account, then we can withdraw the amount.

**Class diagram:**

|  |
| --- |
| BankAccount |
| - name: String  - Accno: int  - CurrBal: int |
| BankAccount: void  + withdraw(int WAmt): void  + deposit(int DAmt): int |

**PROGRAM- 2:**

**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 these methods named start(),stop(),service()**

**4. Create the objects named c1, c2,c3.**

**AIM:**

To create java program with following instructions :

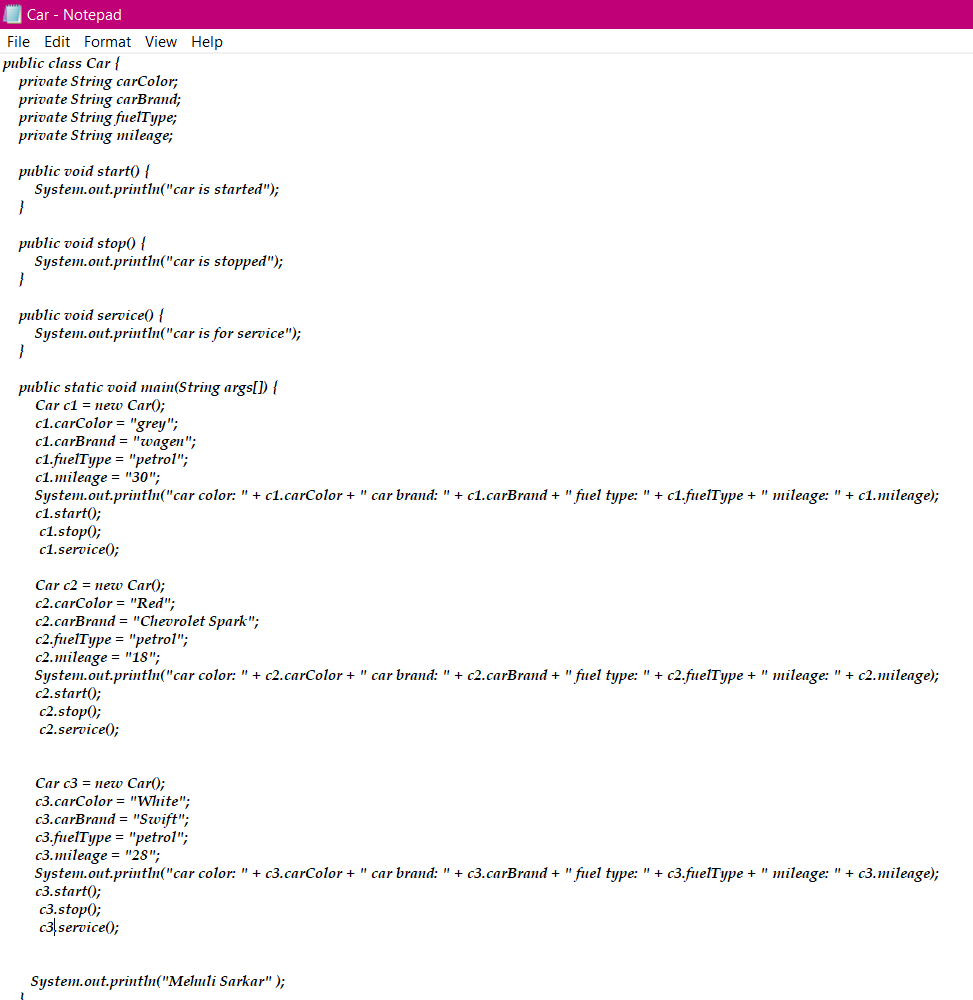
1. Create a class with name Car

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

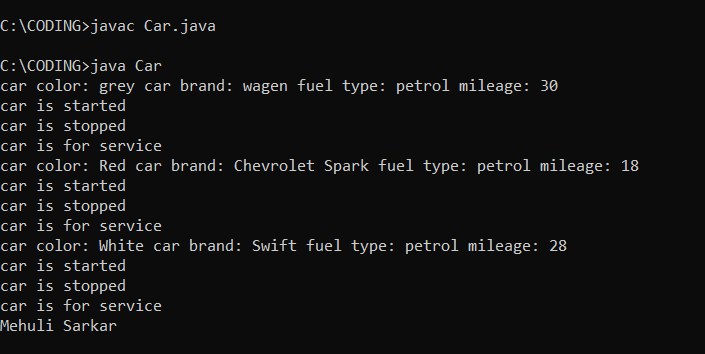
3. Create these methods named start(),stop(),service()

4. Create the objects named c1, c2,c3.

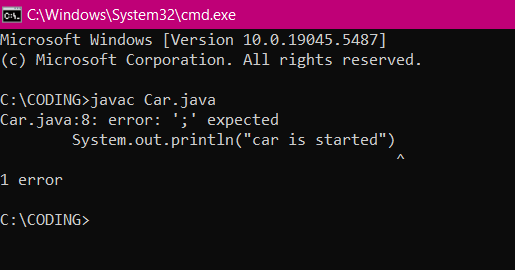
**CODE:**



**Output:**



**ERROR:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| Not putting the semi-colon | Use semi-colon |

**IMPORTANT POINTS:**

1. Before calling the function we should write the method properly.
2. Here, the “public void start( )” indicates that we are writing a method to call the function.
3. When we call a certain method, the process inside it will be printed as an output of the code.
4. Here the details inside the function are called objects, we can give any objects

**Class diagram:**

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

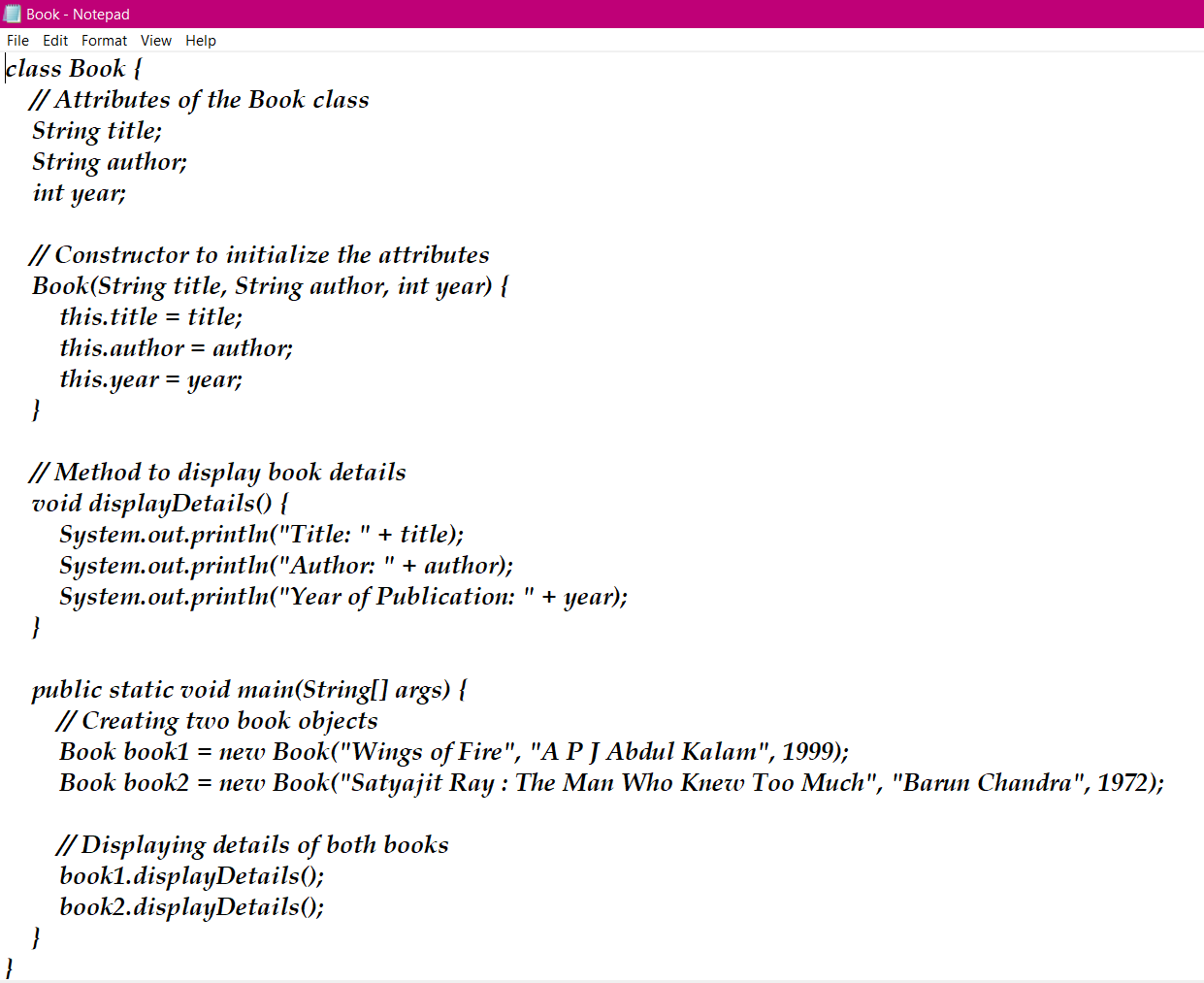
**WEEK 4**

**Program 1**

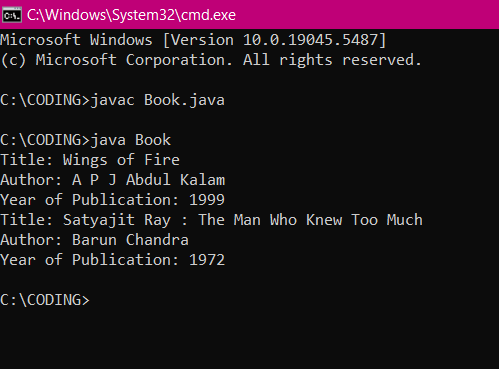
**Aim:**

Write a java program with class named “book” the class should contain various attributes such as the title of the book, author, year of publication. It should also contain a constructor with parameter which initializes the title of the book, author, year of publication. Create a method which displays the details of the book ie title , author, year. Display the details of 2 books by creating 2 objects.

**Code:**



**Output:**

****

**Error:**

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Error message** | **Error rectification** |
| **1.** | **Accessing count in System.out.println without a class reference** | **count is a static variable, meaning it belongs to the class, not an instance. It should be accessed using MyClass.count instead of count inside System.out.println.** |
| **2.** | **Accessing PI through an object** | **Since PI is a final instance variable (not static), it must be accessed through an object, which is correct.** |

**Class Diagram:**

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

**Concepts to be known:**

1)A class is a blueprint, and an object is an instance of that class. (Book is the class, book1 and book2 are objects).

2)Special methods used to initialize objects when they are created (Book(String title, String author, int year)).

3) This keyword refers to the current object’s attributes and prevents naming conflicts.

4) A function inside a class that defines behavior (displayDetails() method prints book details).

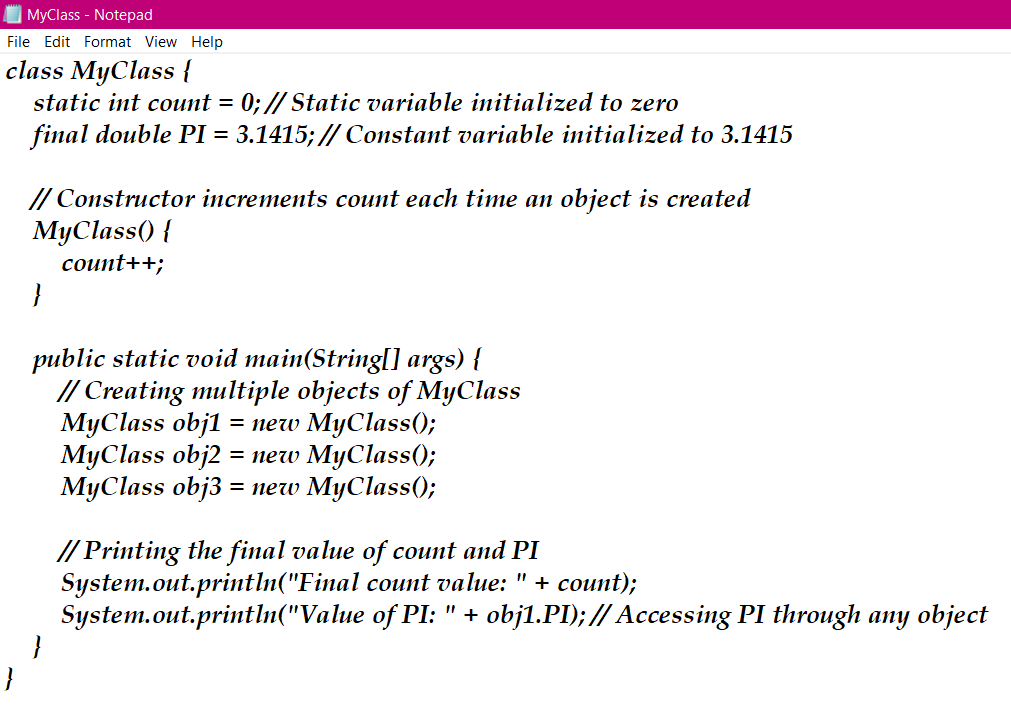
5) Keeping data organized by defining attributes and accessing them through methods instead of direct manipulation.

**Program 2**

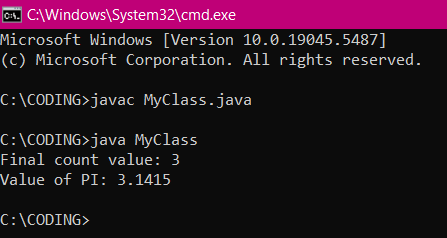
**Aim:**

To create a java program with class named “my class” with a static variable “count” of int type,initialized to zero and a constant variable “pi” of type “double” initialized to 3.1415 as attributes of that class. Now define a constructor for my class that increments the count variable each time and object of my class is created. Finally print the final value of “count” and “pi”.

**Code:**

****

**Output:**

****

**Error:**

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Error message** | **Error rectification** |
| **1.** | **Accessing count in System.out.println without a class reference** | **count is a static variable, meaning it belongs to the class, not an instance. It should be accessed using MyClass.count instead of count inside System.out.println.** |
| **2.** | **Accessing PI through an object** | **Since PI is a final instance variable (not static), it must be accessed through an object, which is correct.** |

**Class Diagram:**

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

**Concepts to be known:**

1. Shared among all instances of a class and accessed using the class name (e.g., MyClass.count).

2) Constants that cannot be modified after initialization (e.g., final double PI = 3.1415).

3) Special methods that execute when an object is created, often used for initializing values or modifying static variables.

4) Instance variables belong to objects, while static variables belong to the class. Static members are accessed using the class name, while instance members require an object.

5) Keeping data safe by defining variables as private (though not used here, it’s a key OOP principle for better control and security).

**WEEK 5**

**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);

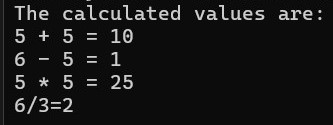
u.mul(5,5);

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.

**Aim:**

A vehicle 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:");

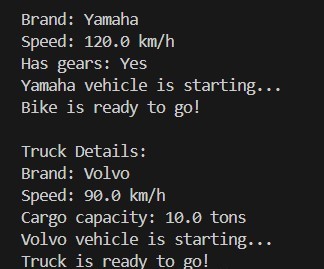
        truck.displayTruckDetails();

        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) {

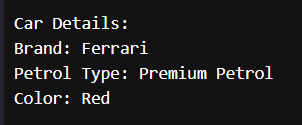
Vehicle myCar = new Car("Ferrari", "Premium Petrol", "Red");

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("Laila", 69);

PG pg = new PG("Majnu", 59);

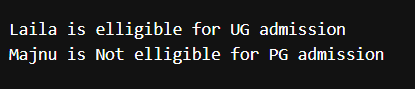
ug.Eligibility();

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();

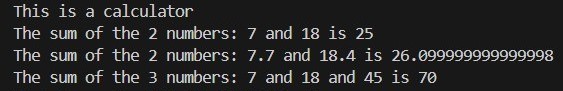
    System.out.println("The sum of the 2 numbers: 7 and 18 is "+ calc.add(7, 18));

    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));

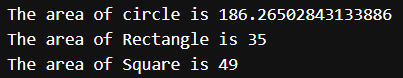
System.out.println("The area of Rectangle is " + c.calcArea(5, 7));

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**

**PROGRAM-1**

**AIM**

**Write a Java program to create an abstract class Animal with an abstract method called sound(). Create subclasses Lion and Tiger that extend the Animal class and implement the sound() method to make a specific sound for each animal.**

**CODE:**

abstract class Animal {

abstract void sound();

}

class Lion extends Animal {

@Override

void sound() {

System.out.println("Roar");

}

}

class Tiger extends Animal {

@Override

void sound() {

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

}

}

class Testsound {

public static void main(String[] args) {

System.out.println("Name: Mehuli Sarkar, Rollno: AV.SC.U4CSE2450, Section: B");

Lion l = new Lion();

Tiger t = new Tiger();

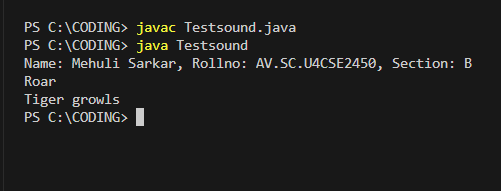
l.sound();

t.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.** |

**IMPORTANT POINTS:**

1. We override the methods in the superclass.

Here we are using the heirarchial inheritance**.**

**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.**

**CODE:**

abstract class Shape3D {

    abstract double volume();

    abstract double surfaceArea();

}

class Sphere extends Shape3D {

    double radius;

    Sphere(double radius) {

        this.radius = radius;

        System.out.println("Sphere created with radius: " + radius);

    }

    @Override

    double volume() {

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

    }

    @Override

    double surfaceArea() {

        return 4 \* Math.PI \* radius \* radius;

    }

}

class Cylinder extends Shape3D {

    double radius;

    double height;

    Cylinder(double radius, double height) {

        this.radius = radius;

        this.height = height;

        System.out.println("Cylinder created with radius: " + radius + " and height: " + height);

    }

    @Override

    double volume() {

        return Math.PI \* radius \* radius \* height;

    }

    @Override

    double surfaceArea() {

        return 2 \* Math.PI \* radius \* (height + radius);

    }

}

public class Shapetest {

    public static void main(String[] args) {

        System.out.println("Name: Mehuli Sarkar, Rollno: AV.SC.U4CSE2450, Section: B");

        Sphere s1 = new Sphere(5.0);

        System.out.printf("Sphere Volume: %.2f%n", s1.volume());

        System.out.printf("Sphere Surface Area: %.2f%n", s1.surfaceArea());

        Cylinder c1 = new Cylinder(3.0, 7.0);

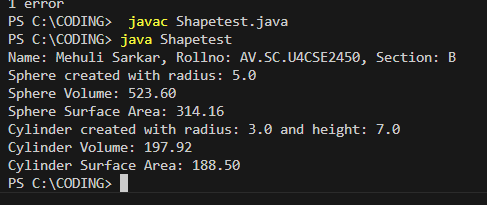
        System.out.printf("Cylinder Volume: %.2f%n", c1.volume());

        System.out.printf("Cylinder Surface Area: %.2f%n", c1.surfaceArea());

    }

}

**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.** |

1. Here we used the abstract to declare an abstract class.
2. Abstract classes and methods help us to declare the methods without declaring the return type in them.

To get the values, we declared a constructor for each subclass and initialized values for them

**PROGRAM-3**

**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:**

**1. StarPattern – prints a right-angled triangle of stars**

**2. 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: Mehuli Sarkar, Section: B, Roll No: AV.SC.U4CSE24150");

        int numberOfRows = 5;

        PatternPrinter star = new StarPattern(numberOfRows);

        star.displayTitle("Star Pattern");

        star.printPattern();

        PatternPrinter number = new NumberPattern(numberOfRows);

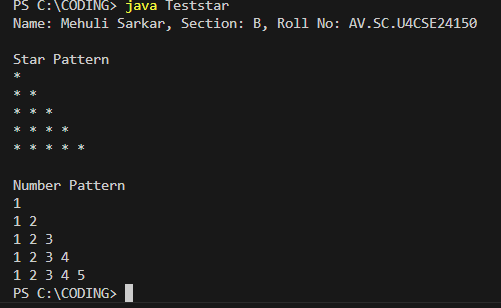
        number.displayTitle("Number Pattern");

        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** |

**Important Points:**

Use abstract classes to enforce a common structure for pattern printing.

PatternPrinter is the abstract class defining the common template.

Subclasses (StarPattern, NumberPattern) provide specific implementations.

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:**

class Shapes {

    public double area() {

        return 0;

    }

}

class Triangle extends Shapes {

    private double base;

    private double height;

    public Triangle(double base, double height) {

        this.base = base;

        this.height = height;

    }

    @Override

    public double area() {

        return 0.5 \* base \* height;

    }

}

class Circle extends Shapes {

    private double radius;

    public Circle(double radius) {

        this.radius = radius;

    }

    @Override

    public double area() {

        return Math.PI \* radius \* radius;

    }

}

class Rectangle extends Shapes {

    private double length;

    private double width;

    public Rectangle(double length, double width) {

        this.length = length;

        this.width = width;

    }

    @Override

    public double area() {

        return length \* width;

    }

}

public class ShapeArea {

    public static void main(String[] args) {

        Shapes triangle = new Triangle(2, 5);

        Shapes circle = new Circle(4);

        Shapes rectangle = new Rectangle(6, 9);

        System.out.println("Mehuli Sarkar CSE-B 24150 " );

        System.out.println("Area of Triangle: " + triangle.area());

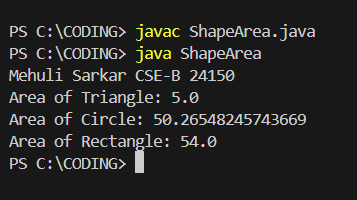
        System.out.println("Area of Circle: " + circle.area());

        System.out.println("Area of Rectangle: " + rectangle.area());

    }

}

**OUTPUT:**

****

**CLASS DIAGRAM:**

**Shapes**

**+ area(): double**

**Triangle**

**- base**

**- height**

**+ area():double**

**Circle**

**radius**

**Rectangle**

**-length**

**-width**

**+ area(): double**

**ERROR TABLE:**

|  |  |
| --- | --- |
| CODE ERROR  1)Class name "Shapes" is inconsistently used (should be consistent capitalization)    2)Base class method area() returns 0 by default - better to make it abstract | ERROR RECTIFICATION  1)Change to consistent capitalization (either all "Shapes" or all "Shapes")  2)Consider making Shapes abstract with abstract area() method |

**IMPORTANT POINTS:**

Inheritance Hierarchy: The Traingle, Circle and Rectangle classes all inherit from the base Shapes class (note: class name is misspelled as "Shapes" in some places and "Shapes" in others).

Polymorphism: Each subclass overrides the area() method to provide its own implementation, demonstrating polymorphic behavior.

Encapsulation: All shape classes properly encapsulate their attributes (base, height, radius, length, width) as private fields.

Method Overriding: The area() method is overridden in each subclass with the appropriate calculation formula for that shape.

Main Class: The ShapeArea class demonstrates the use of these shapes by creating instances and calling their area() methods.

**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 {

    void play();

}

class Football implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Football: Kicking the ball towards the goal");

    }

}

class Volleyball implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Volleyball: Bumping, setting, and spiking the ball");

    }

}

class Basketball implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Basketball: Dribbling and shooting the ball");

    }

}

public class TestSport {

    public static void main(String[] args) {

      System.out.println("Mehuli Sarkar, CSE-B, 24150");

        Playable football = new Football();

        Playable volleyball = new Volleyball();

        Playable basketball = new Basketball();

        football.play();

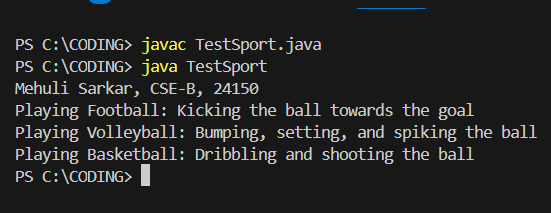
        volleyball.play();

        basketball.play();

    }

}

**OUTPUT:**

****

**CLASS DAIGRAM:**

**Interface: Playable**

**+ play(): void**

**FOOTBALL**

**+ play(): void**

**Volleyball**

**+ play(): void**

**Basketball**

**+ play(): void**

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1. Declaring an abstract class instead of interface class. 2. Not declaring public in each class. | 1. Declare an interface class instead of abstract class. 2. Declare public infront of each class. |

**IMPORTANT POINTS:**

1. The playable interface abstracts the play() method, ensuring different classes implement it differently
2. The play() method behaves differently based on the object type football, volleyball, basketball.

Each class encapsulates its own implementation of how the sport is played, hiding the details from the user

**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(" Mehuli Sarkar, CSE-B, 24150");

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

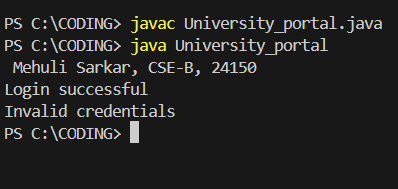
        p1.login("Student123", "pass02");

        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

**WEEK-9**

**PROGRAM-1:**

**AIM:**

write a java program to create a method that takes an integer as parameter and throws an exception of the number is even.

**CODE:**

public class EvenNumberExceptionExample{

    public static void checkNumber(int number) throws Exception{

        if (number%2==0){

            throw new Exception ("Even number is not allowed " + number);

        } else {

            System.out.println(" Valid odd number "+ number);

        }

    }

    public static void main(String[] args) {

        try {

            checkNumber(100);

        } catch (Exception e) {

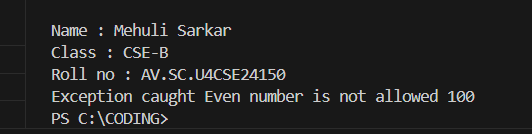
            System.out.println("Name : Mehuli Sarkar\nClass : CSE-B\nRoll no : AV.SC.U4CSE24150");

            System.out.println("Exception caught " + e.getMessage());

        }    }

}

**Output:**

****

**Class Diagram:**

|  |
| --- |
| EvenNumberExceptionDemo |
| + checkOddNumber(int number) : void  + main(String[] args) : void |

|  |
| --- |
| EvenNumberException  (extends Exception) |
| + EvenNumberException(String msg) |

**ERROR TABLE:**

|  |  |
| --- | --- |
| CODE ERROR  1)Unhandled exception type EvenNumberException  2) Syntax Error  3) Compilation Error | ERROR RECTIFICATION  1) If you call checkOddNumber() without using try-catch or without declaring throws.  2)If missing curly braces {} or wrong method syntax  3)If constructor of EvenNumberException is missing or incorrectly defined. |

**IMPORTANT POINTS:**

1) Created a custom exception by extending the Exception class.

2) Used throw keyword to manually throw the custom exception if the number is even.

3)Handled the exception using a try-catch block inside main() method.

4) Demonstrates user-defined exception handling.

5) Shows clear separation of concerns: checking number and exception message.

**PROGRAM-2:**

**AIM:**

write a java program to create a method that takes an integer as parameter and throws an exception of the number is even

**CODE:**

import java.io.\*;

public class FileReadExample{

    public static void main(String[]args) throws IOException{

        try{

            BufferedReader reader = new BufferedReader(new FileReader("C://Users//CSE-Lab1//Desktop//cse24141//Games.java/"));

            String line;

            while ((line = reader.readLine()) != null) {

                System.out.println(line);

            }

            reader.close();

        } catch (FileNotFoundException e){

            System.out.println("File not found" +e.getMessage());

        } catch (IOException e) {

            System.out.println("Error reading file: " + e.getMessage());

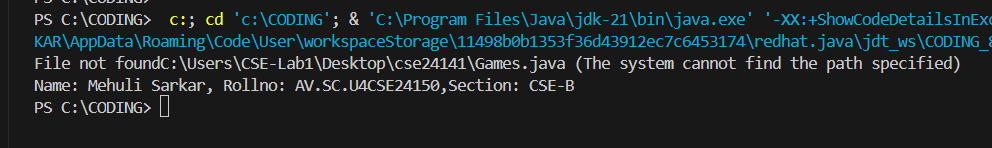
        }

        System.out.println("Name: Mehuli Sarkar, Rollno: AV.SC.U4CSE24150,Section: CSE-B");

    }

}

**Output:**

****

**CLASS DIAGRAM:**

|  |
| --- |
| **FileReadExample** |
| **+ main(String[] args) : void** |

|  |
| --- |
| **Uses:** |
| **- BufferedReader**  **- FileReader**  **- FileNotFoundException**  **- IOException** |

**ERROR TABLE:**

|  |  |
| --- | --- |
| Error | Error Rectification |
| 1. FileNotFoundException 2. IOException 3. Syntax Error | 1. Occurs if the specified file path is wrong or file does not exist. 2. Occurs while reading file if an input/output error happens. 3. If missing semicolon ;, wrong try-catch block syntax. |

**IMPORTANT POINTS:**

1) Used BufferedReader and FileReader to read text files.

2) FileNotFoundException occurs if the file is missing.

3) IOException occurs for input/output errors during file reading.

4) try-catch block is used for proper exception handling.

5) Always close the reader after reading the file (reader.close()).

**PROGRAM-3:**

**AIM:**

write a java program to handle an arthematic exception using try catch finally

**CODE:**

public class ArithmeticExceptionDemo {

public static void main(String[] args) {

int numerator = 10;

int denominator = 0; // This will cause an ArithmeticException

try {

int result = numerator / denominator;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("ArithmeticException caught: " + e.getMessage());

} finally {

System.out.println("Finally block executed. Cleaning up resources if any.");

}

System.out.println("Program continues after try-catch-finally block.");

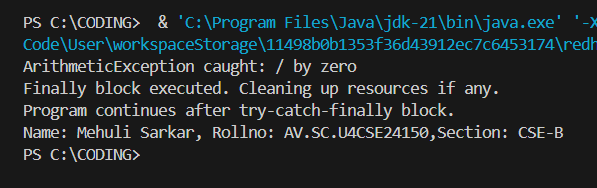
System.out.println("Name: Mehuli Sarkar, Rollno: AV.SC.U4CSE24150,

Section: CSE-B");

}

}

**Output:**

****

**CLASS DIAGRAM:**

|  |
| --- |
| ArithmeticExceptionDemo |
| + main(String[] args) : void |

|  |
| --- |
| Uses: |
| - try-catch-finally block  - ArithmeticException |

**ERROR TABLE:**

|  |  |
| --- | --- |
| Error | Error Rectification |
| 1. ArithmeticException 2. Syntax Error 3. Compilation Error | 1. Division by zero operation 2. Missing try-catch or braces 3. Catch block written incorrectly |

**IMPORTANT POINTS:**

1)Division by zero causes ArithmeticException.

2)finally block always executes.

3)Safe program termination even after exceptions.

**PROGRAM-4:**

**AIM:**

Write a Java program to simulate a University system using inner classes.

1) Create an outer class named University with a variable universityName. Inside it, define two non-static inner classes:

2) Department with variables like deptName and deptCode, and a method to display department details.

3) Student with variables like studentName and rollNumber, and a method to display student details.

4) Create an object for each class, their methods to display their details along with the university name.

**CODE:**

public class University {

String universityName;

University(String universityName) {

this.universityName = universityName;

}

class Department {

String deptName;

String deptCode;

Department(String deptName, String deptCode) {

this.deptName = deptName;

this.deptCode = deptCode;

}

void displayDepartmentDetails() {

System.out.println("University Name: " + universityName);

System.out.println("Department Name: " + deptName);

System.out.println("Department Code: " + deptCode);

}

}

class Student {

String studentName;

int rollNumber;

Student(String studentName, int rollNumber) {

this.studentName = studentName;

this.rollNumber = rollNumber;

}

void displayStudentDetails() {

System.out.println("University Name: " + universityName);

System.out.println("Student Name: " + studentName);

System.out.println("Roll Number: " + rollNumber);

}

}

public static void main(String[] args) {

University university = new University("AMRITA VISWA VIDYAPEETHAM");

University.Department department = university.new

Department("Computer Science", "CSE01");

University.Student student = university.new Student("Mehuli Sarkar",

24150);

department.displayDepartmentDetails();

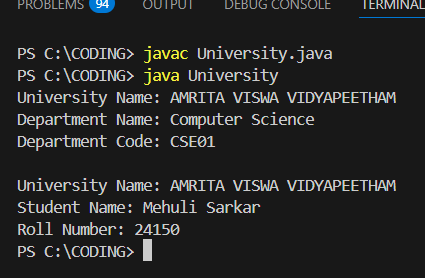
System.out.println();

student.displayStudentDetails();

}

}

**Output:**

****

**CLASS DIAGRAM:**

|  |
| --- |
| University |
| -universityName: String  + University(String name) |

|  |
| --- |
| Inner Classes:  - Department  - deptName: String  - deptCode: String  + displayDepartment(): void |
| - Student  - studentName: String  - rollNumber: int  + displayStudent(): void |

|  |
| --- |
| + main(String[] args): void |

**ERROR TABLE:**

|  |  |
| --- | --- |
| Error | Error Rectification |
| 1. Syntax Error 2. Compilation Error 3. Runtime Error | 1. Wrong object creation for inner class 2. Accessing outer class members wrongly 3. NullPointerException if outer object missing |

**IMPORTANT POINTS:**

1. Demonstrates inner class usage.
2. Inner classes access outer class members easily.
3. Separate objects for Department and Student.
4. Good example of encapsulation.