**23CSE111**

**OBJECT-ORIENTED PROGRAMMING**

**LAB REPORT**

****

**Department of Computer Science Engineering**

**Amrita School of Computing**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**VERIFIED BY: NAME: P.PRANATHI**

**ROLL NO: AV.SC.U4CSE24239**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNo | Title | Date | Page No | Signature |
| WEEK 1 |  |  |  |  |
| 1 | Download and Install Java Software |  |  |  |
| 2 | Write a Java program to print the message “Welcome to Java Programming.” |  |  |  |
| WEEK 2 |  |  |  |  |
| 1 | Write a Java program to calculate area of rectangle. |  |  |  |
| 2 | Write a java program to calculate the area of the triangle. |  |  |  |
| 3 | Write a Java program to convert temperature from Fahrenheit to Celsius |  |  |  |
| 4 | Write a Java program to convert temperature from Celsius to Fahrenheit. |  |  |  |
| 5 | Write a Java program to calculate factorial of a number. |  |  |  |
| 6 | Write a Java program to calculate fibonacci of a number. |  |  |  |
| 7 | Write a Java program to calculate Simple interest. |  |  |  |
| WEEK 3 |  |  |  |  |
| 1 | Write a java program with  1. A class with name Car.  2.Create 4 attributes named car\_color , car\_brand ,fuel\_type ,mileage. 3.Create 3 methods named start() ,stop() ,service(). 4.Create 3 objects named car1 ,car2 ,car3. |  |  |  |
| 2 | Write a java program with 1.Create a class named Bankaccount. 2.Create a constructor. 3.Create 2 methods which are withdrawl() and deposit(). |  |  |  |
| WEEK 4 |  |  |  |  |
| 1 | Write a java program with class named book. This class should contain various attributes such as title, author ,year of publication. It should also contain a constructor with parameters which initializers title ,author ,year of publication. Create a method which displays the details of the book .Display the details of 2 books. |  |  |  |
| 2 | Create a java program with class named myclass with a static variable count of int type ,initial value to zero and a constant variable "pi" of type double initialize 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 values of count and pi variables . Create 3 objects. |  |  |  |
| WEEK 5 |  |  |  |  |
| 1 | Create a calculator using the operations including addition, subtraction, multiplication, and division using multi-level inheritance and display the desired output. |  |  |  |
| 2 | 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?  a.Truck should include an additional property capacity (in tons)  b.Create a showTruckdetails() method to display the truck’s capacity.  c.Write a constructor for Truck that initializes all properties 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. |  |  |  |
| WEEK 6 |  |  |  |  |
| 1 | Write a java program to create a vehicle class with a method displayinfo(). Override this method in the car subclass to provide specific information about a car. |  |  |  |
| 2 | A college is developing an automated admissions systems that verifies students eligibility for undergraduate(UG) and postgraduate(PG) programs. Each program has different eligibility. Criteria based on the students percentage in their previous qualification.  1. UG admission require min of 60%  2. PG admission require min of 70% |  |  |  |
| 3 | Create a calculator class with overloaded methods to perform addition.  A. Add two integers  B. Add two double  C. Add three integer |  |  |  |
| 4 | Create a shape class with a method CalculateArea() that is overloaded for different shapes (e.g square, rectangle) then, create a subclass circle that overrides the calculatearea() method for a circle. |  |  |  |

**WEEK 01**

**PROGRAM-1:**

**AIM:** Download and Install Java Software

**PROCEDURE:**

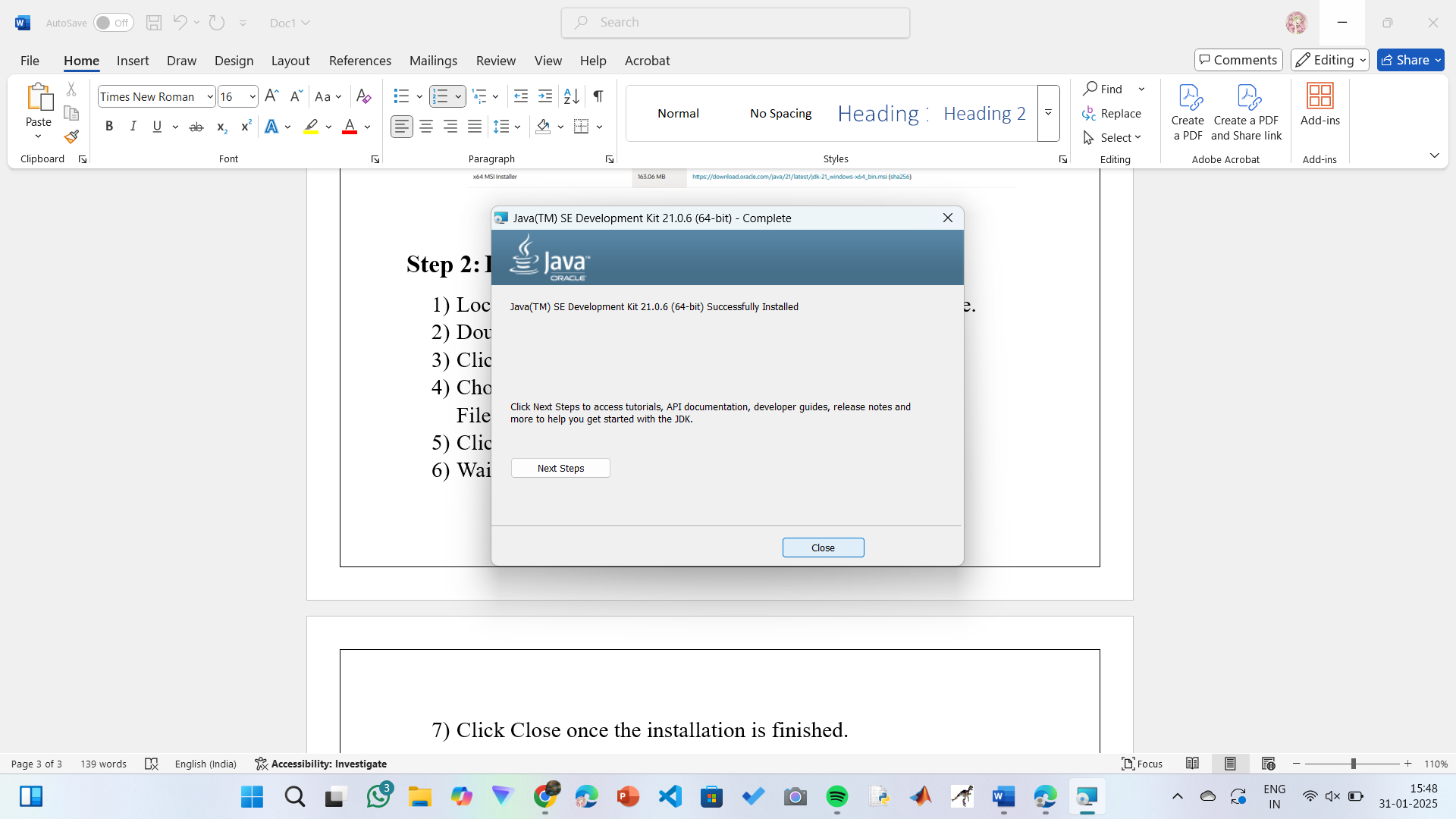
**Step 1: Download JDK 21**

1. Open your web browser and go to the Oracle JDK Downloads page
2. Scroll down to the Java SE Development Kit 21 section.
3. Choose the Windows x64 Installer version.
4. Click on Download, then Wait for the download to complete**.**



**Step 2:** **Install JDK 21**

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

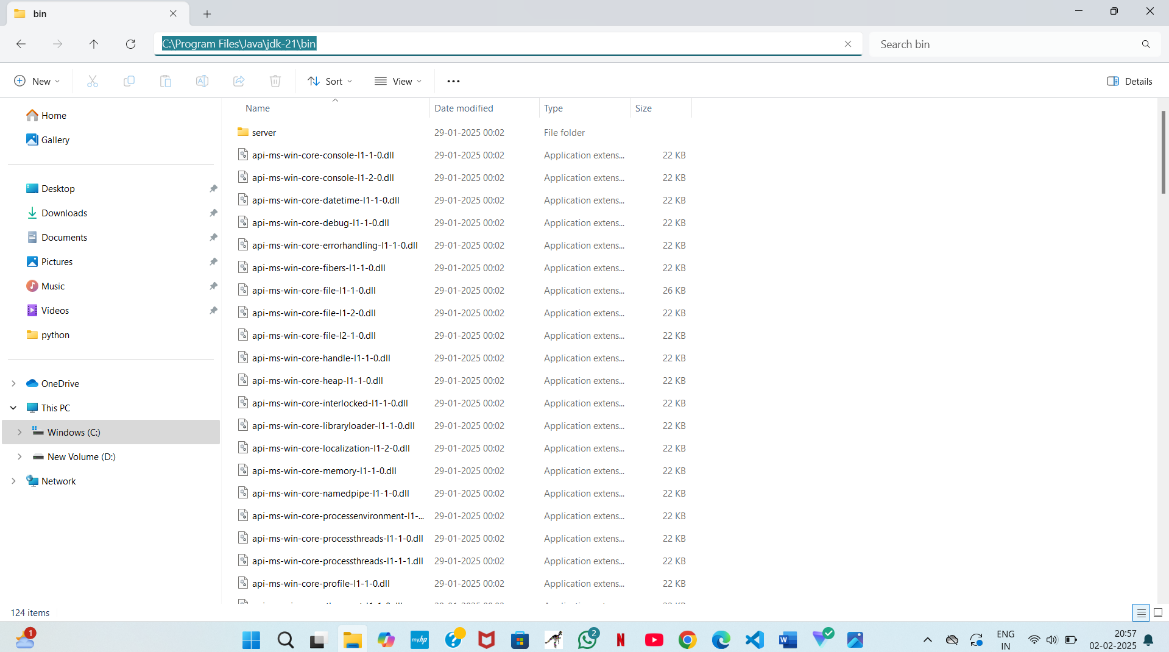


**Step 3: Setting up the path**

1) Go to “Windows C” Drive on Desktop

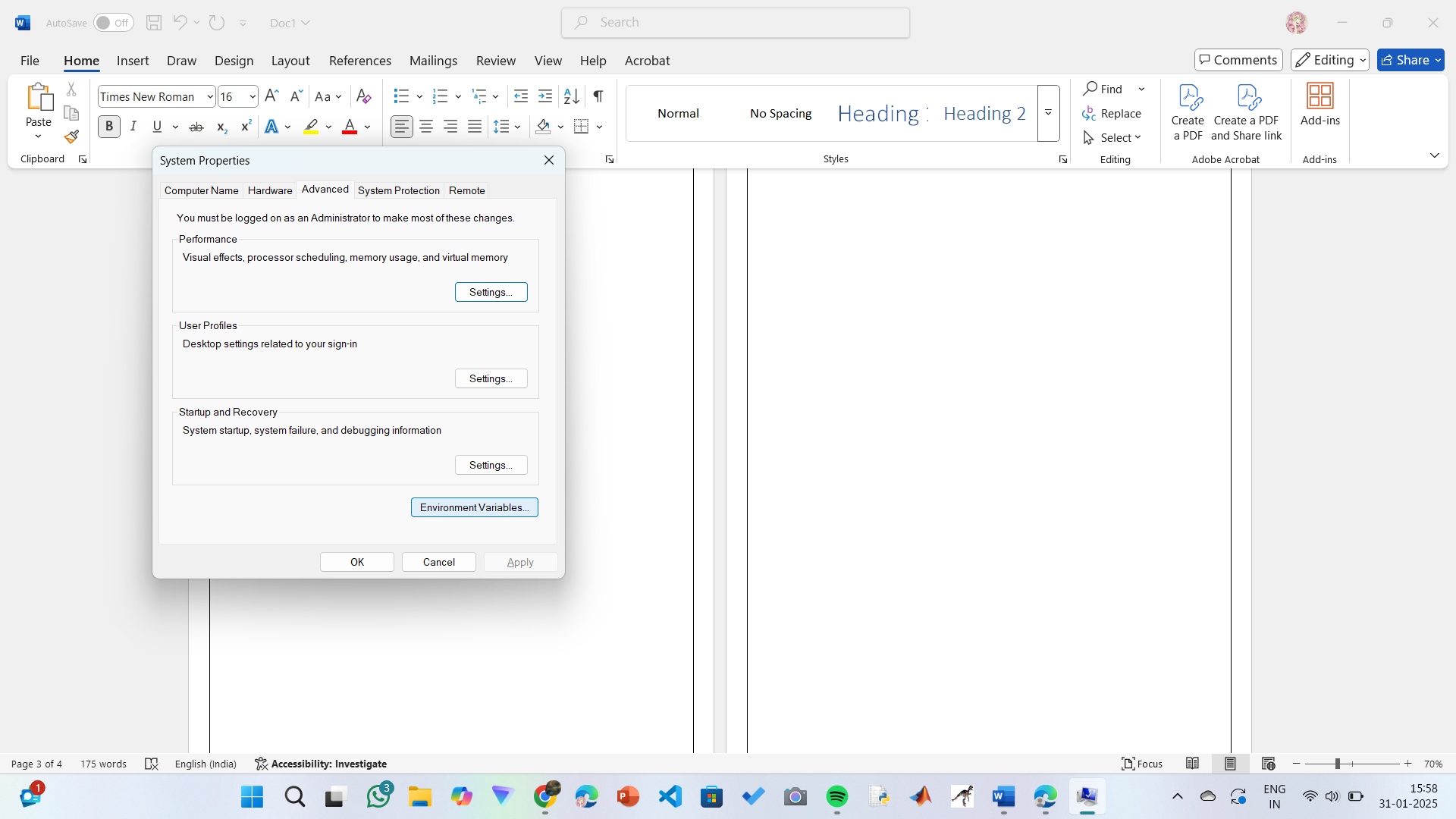
2) Choose Program Files, select Java, then JDK 21, then select Bin.

3) Select and copy the path at the address bar.



**Step 4: Open System Properties**

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



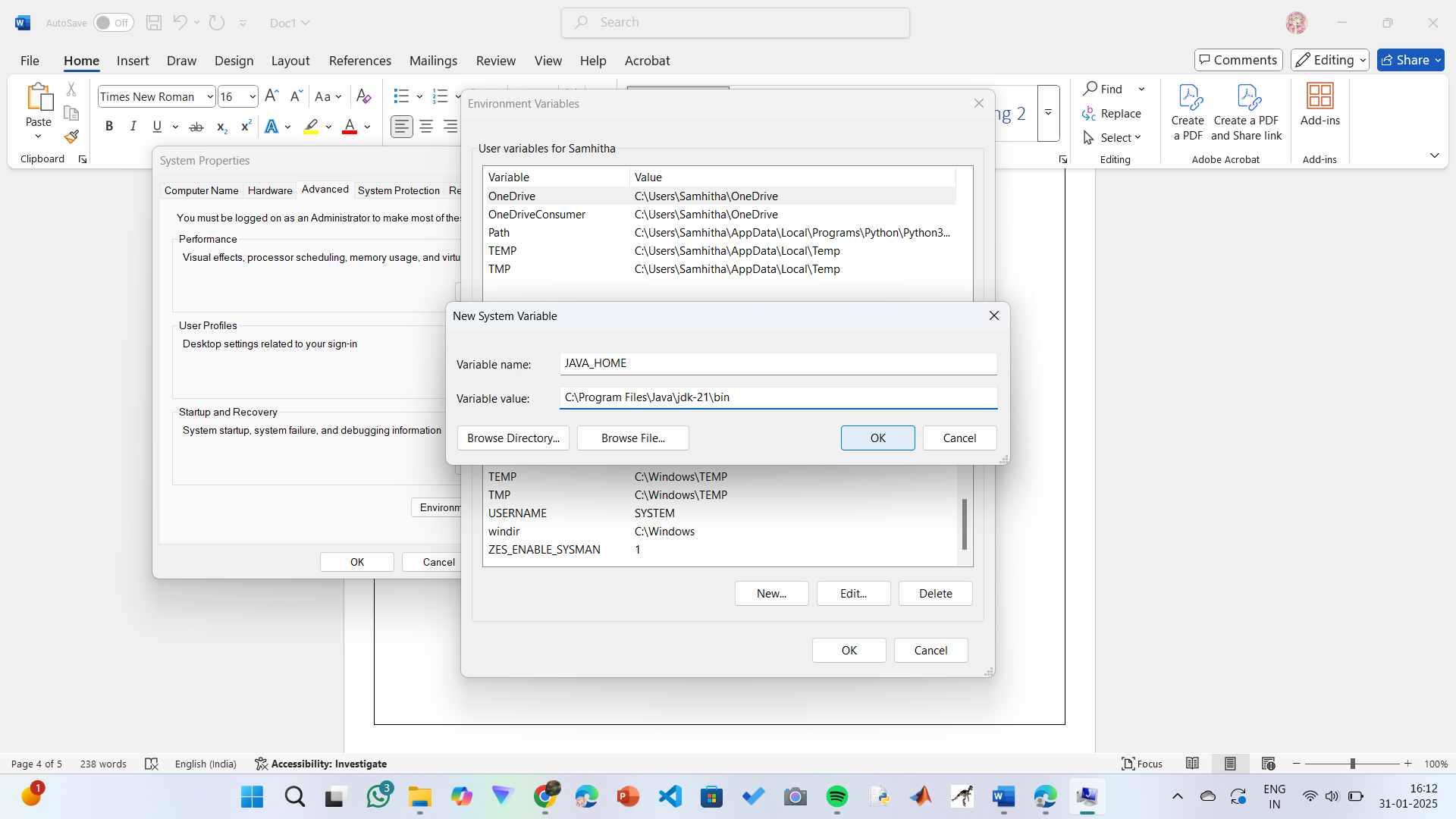
**Step 5: Set JAVA\_HOME**

1)Under System Variables, click New.

2)Set the Variable name as JAVA\_HOME.

3)Set Variable value as C:\Program Files\Java\jdk-21 (or your installation path).

4)Click OK.

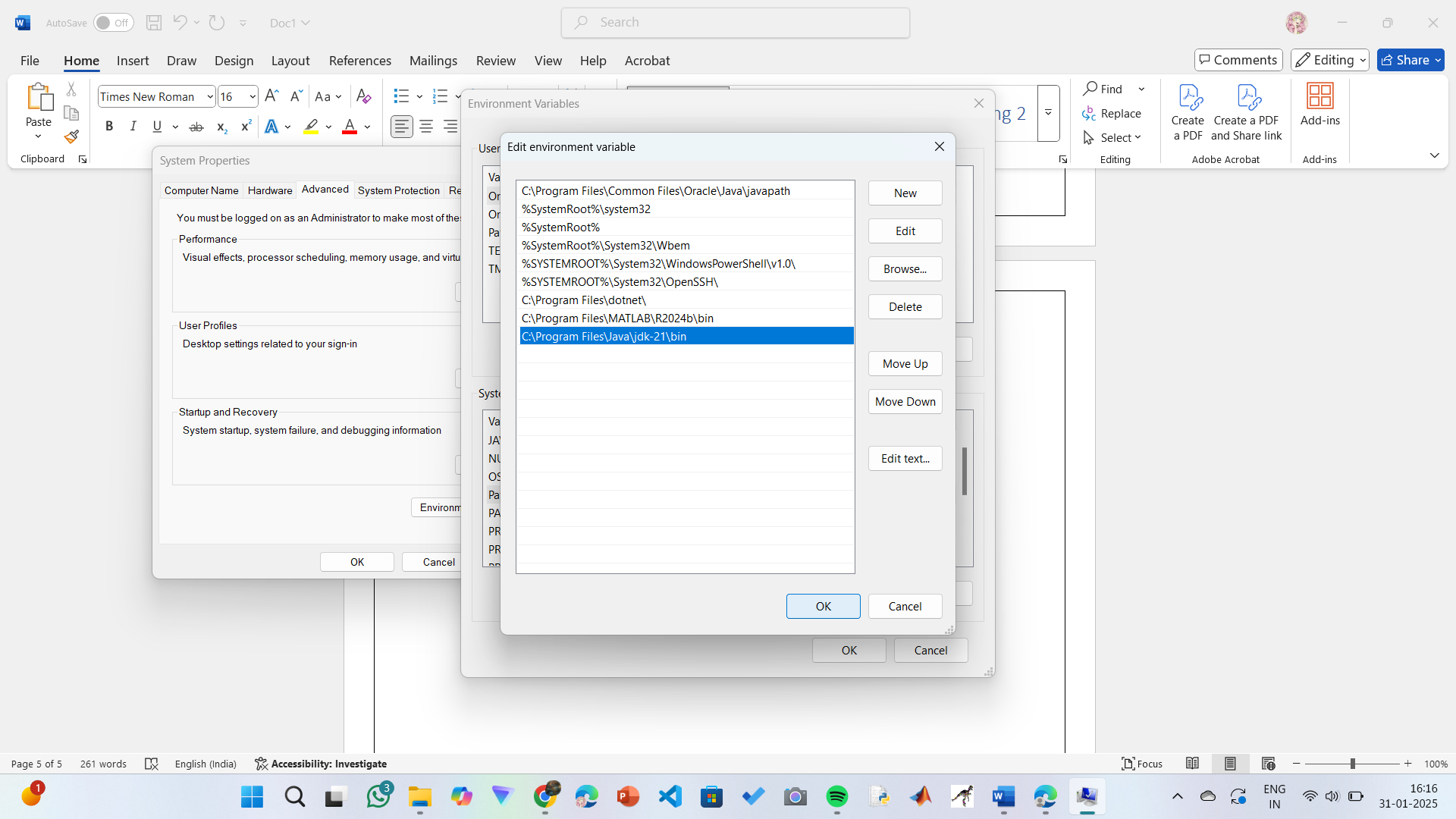


**Step 6: Update PATH Variable**

1)In System Variables, find Path and click Edit.

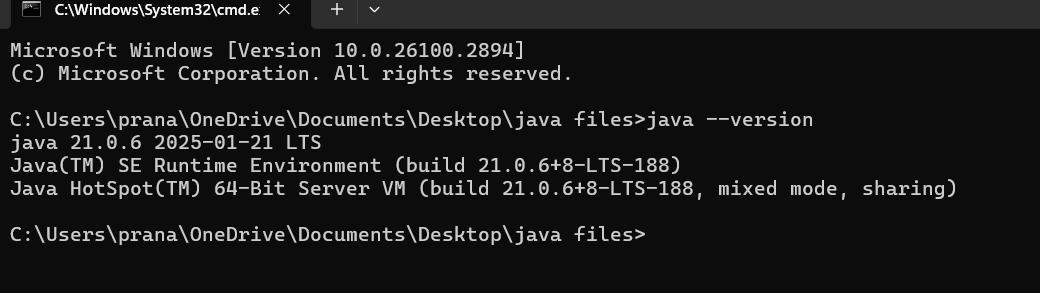
2)Click New and add: C:\Program Files\Java\jdk-21\bin

3)Click OK to save.



**Step 7:Verify Installation**

1. Open Command Prompt.
2. Type the following command: **java --version** and press Enter.



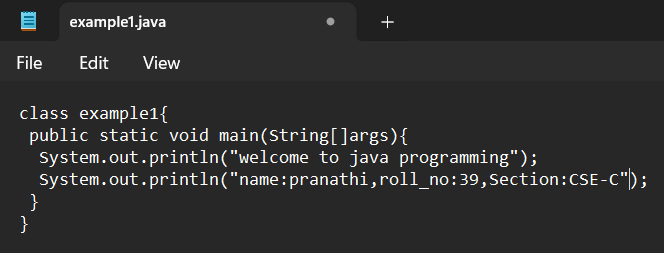
1. To check the java compiler type: **javac –version.**



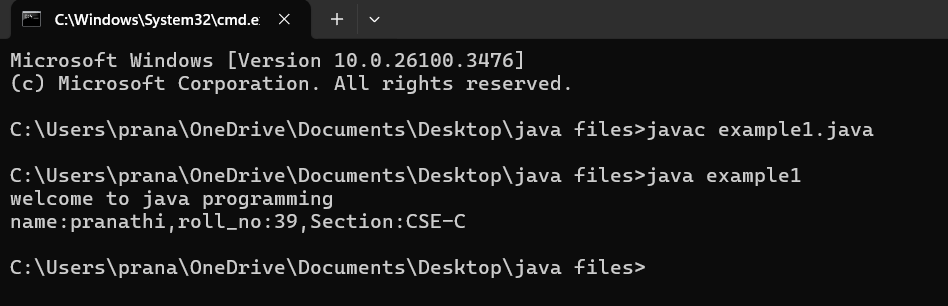
**PROGRAM-2:**

**AIM:** Write a Java program to print the message “Welcome to Java Programming.”

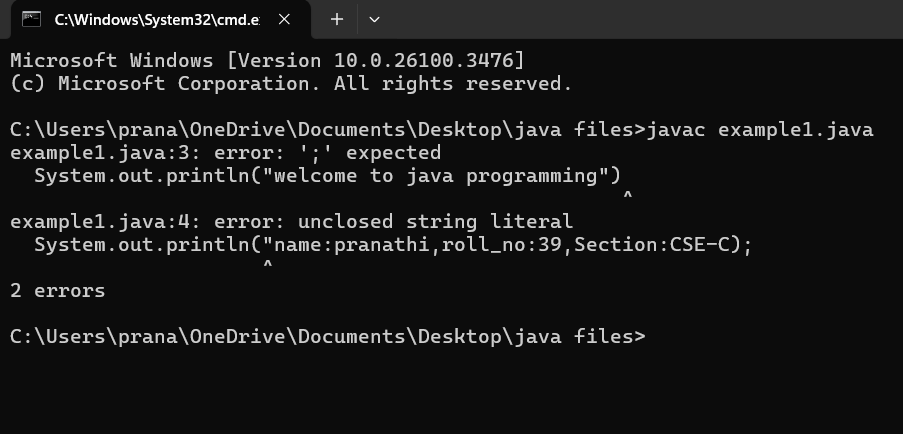
**CODE:**

****

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE**:

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1)writing small ‘s’ in place of ‘S’  In system.out.println()  2)not keeping the “(strings)” gives error .  3)java is case sensitive language . so verify those capital and small letters | 1)code is rectified by keeping capital ‘S’  2)keep the strings so that the output can be printed.  3)check the case sensitivity. Because string and String are different. Here we may get errors . |

**IMPORTANT POINTS:**

1)make sure that there were no syntax errors.

2)Check the case sensitivity.

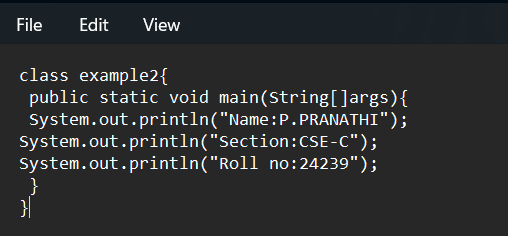
3)follow the syntaxs , in this code we learn that system.out.println()

Is used to print the statements line after line.

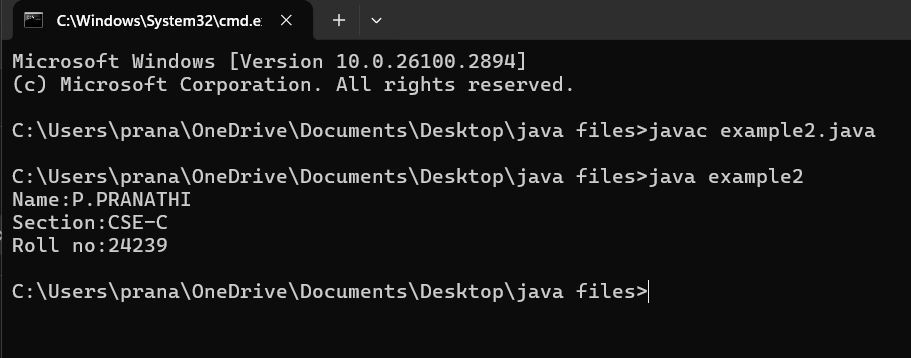
**PROGRAM-3:**

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

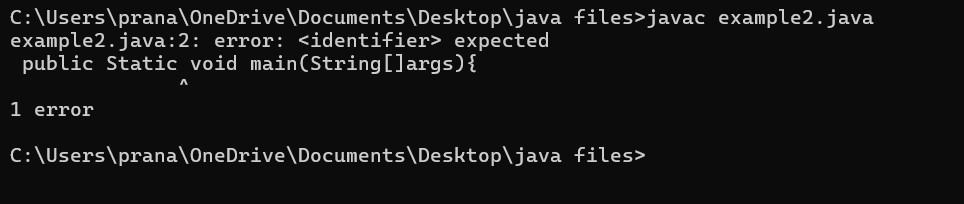
**CODE:**



**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1)writing small ‘s’in place of’S’ In system.out.println()  2)missing semicolon can give the error. | 1)code is corrected by keeping capital ‘S’  2)giving semicolon leads to give the correct the code. |

**IMPORTANT POINTS:**

1)make sure that there were no syntax errors.

2)Check the case sensitivity.

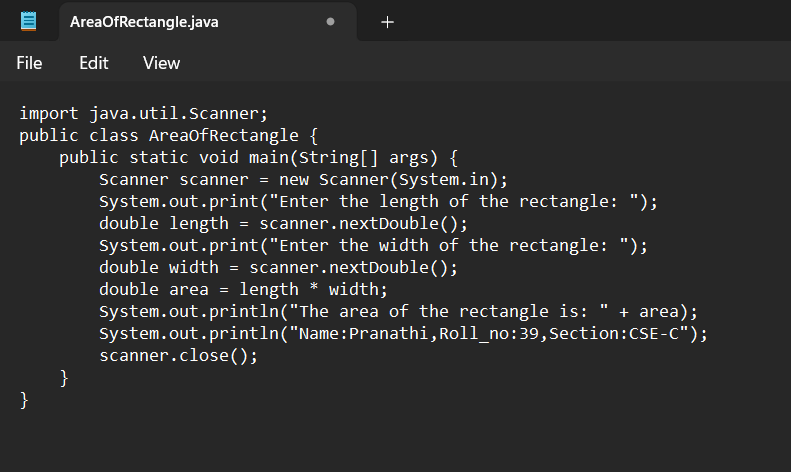
3)follow the syntaxs , in this code we learn that system.out.println()

Is used to print the statements line after line.

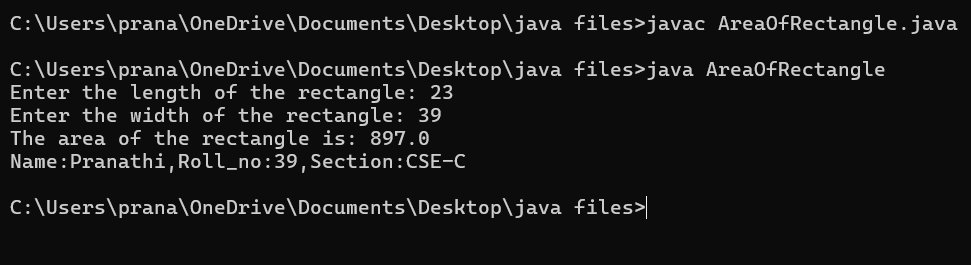
**PROGRAM-4:**

**AIM:** Write a java program to Calculate area of rectangle.

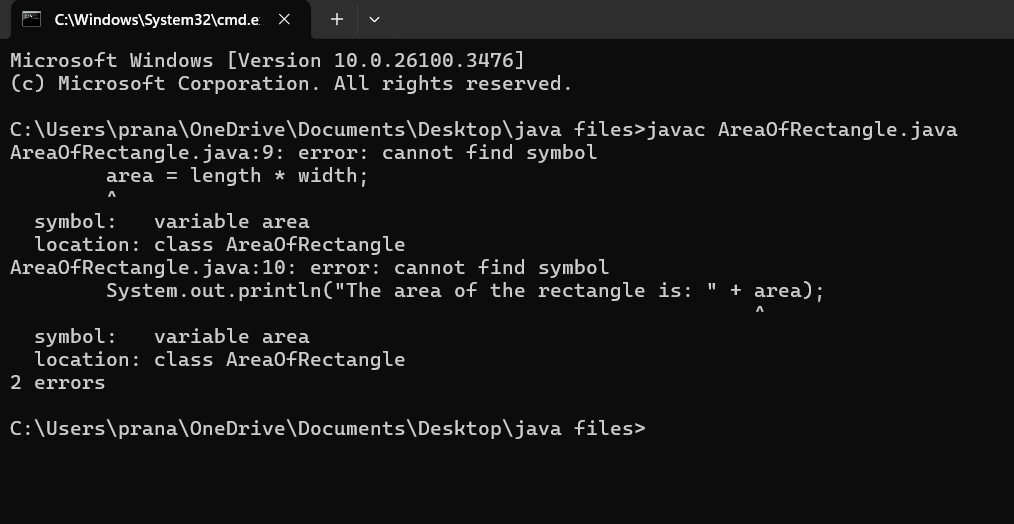
**CODE:**

****

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1)writing small “s”in place of”S”  In (system.out.println()).  2)not giving strings to the name and section  3)not specifying the data type of the variable clearly  Ex: String area is incorrect | 1)code is corrected by keeping capital “S”.  2)Giving strings to name and section  3)mention the data type of the variable clearly  Ex: double area |

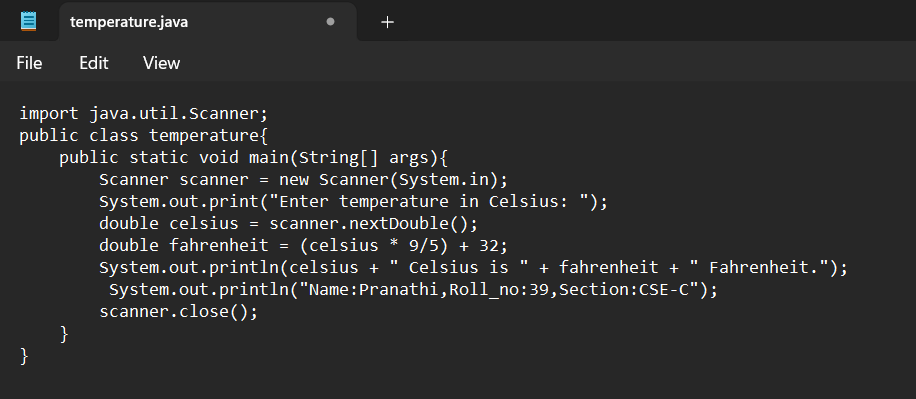
**IMPORTANT POINTS:**

1. The formula to calculate Simple Interest is: (P \* T \* R) / 100, where:
   * P: Principal amount
   * R: Rate of interest
   * T: Time period
2. The Scanner class allows for reading input from the keyboard.
3. import java.util.Scanner signifies:
   * Import: Informs the Java compiler that a specific class or package is being utilized in the code.
   * java.util: This is the package containing utility classes, including the Scanner class.
4. double is a data type used to represent floating-point numbers, which are numbers with decimals.

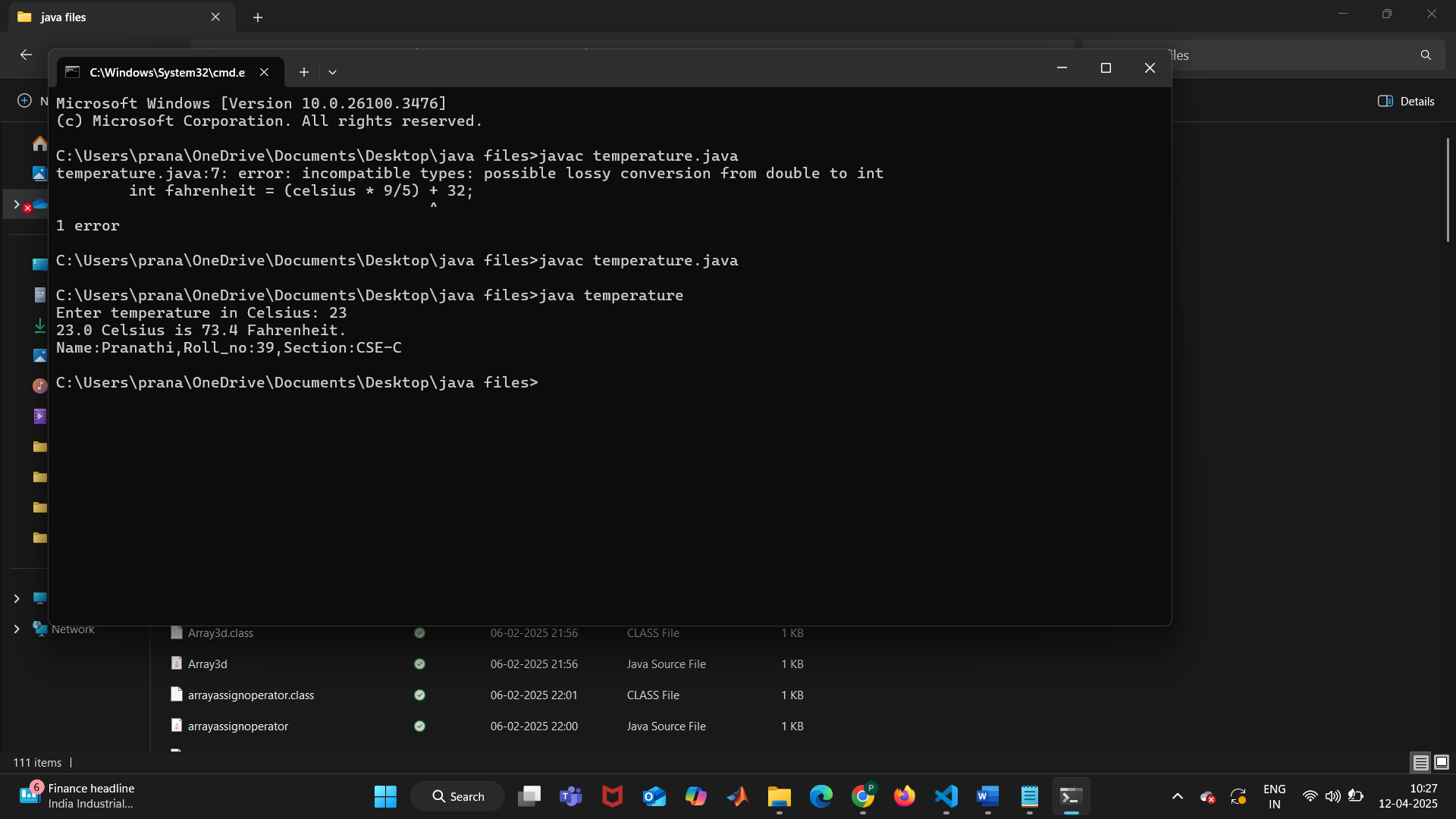
**PROGRAM-5:**

**AIM:** Write a java program to Convert temperature celsius into fahrenheit

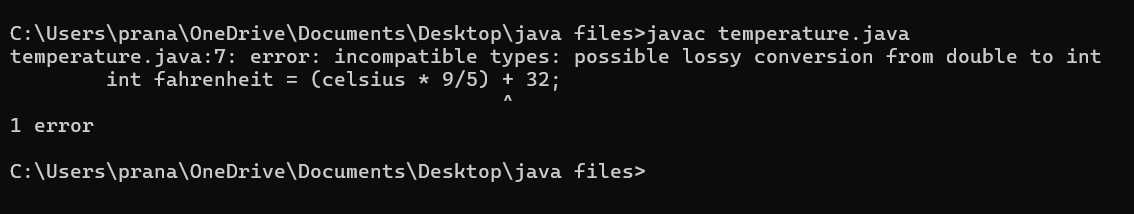
**CODE:**

****

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1)writing small “s”in place of”S”  In system.out.println()  2)not giving strings to the name and section  3)keeping int data type in place of double data type(giving incorrect data types). | 1)code is corrected by keeping capital “S”  2)Giving strings to name and section  3)give correct data type for the particular formulae. |

**IMPORTANT POINTS:**

1.The formula to convert a Celsius to Fahrenheit is

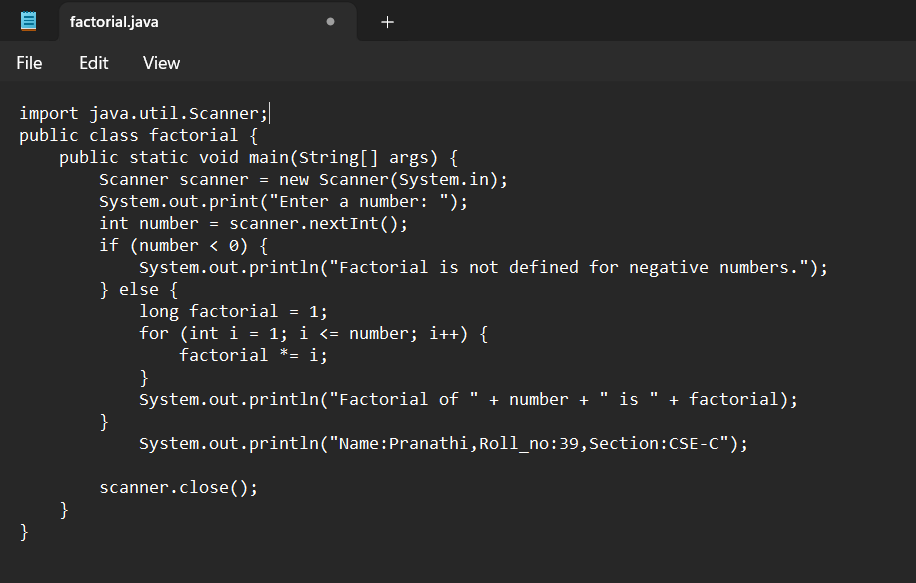
Fahrenheit = (Celsius\*9/5)+32.

2.The line “Scanner input = new Scanner(System.in),” tends to create a new Scanner object named “input” that reads input from the standard input stream (System.in), like keyboard.

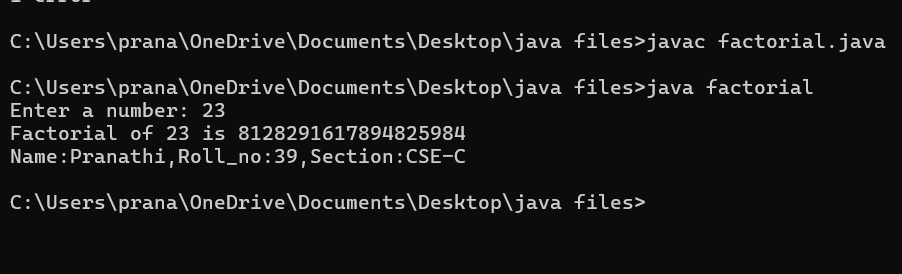
**PROGRAM-6:**

**AIM:**Write a java program for factorial of a number n.

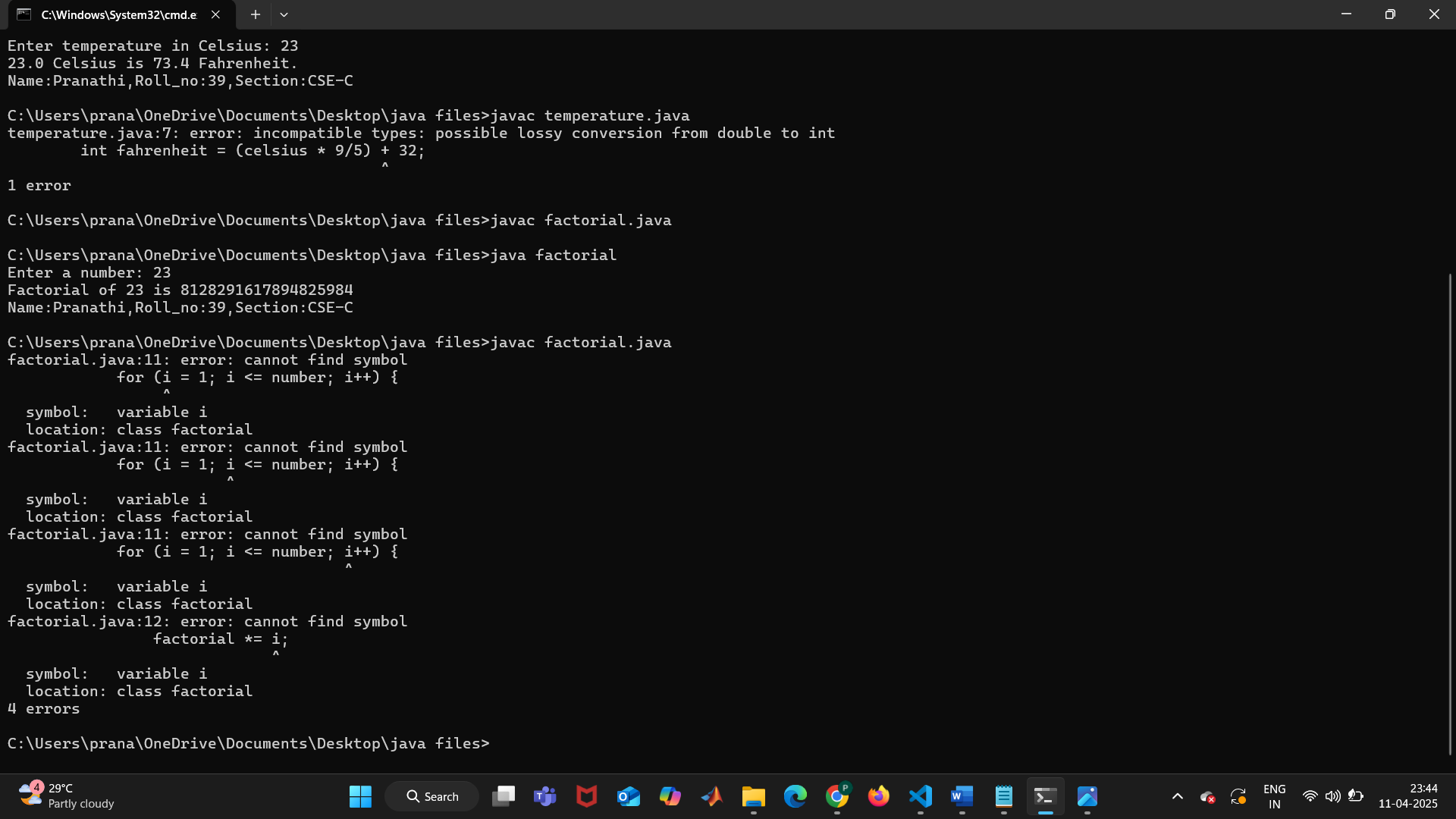
**CODE:**

****

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1)writing small ‘s’ in place of ’S’  In (system.out.println().)  2)not giving strings to the name and section.  3)not mentioning data type inside the for loop can gives the error | 1)code is corrected by keeping capital ‘S’.  2)Giving strings to name and section.  3)mention the data type inside the for loop. |

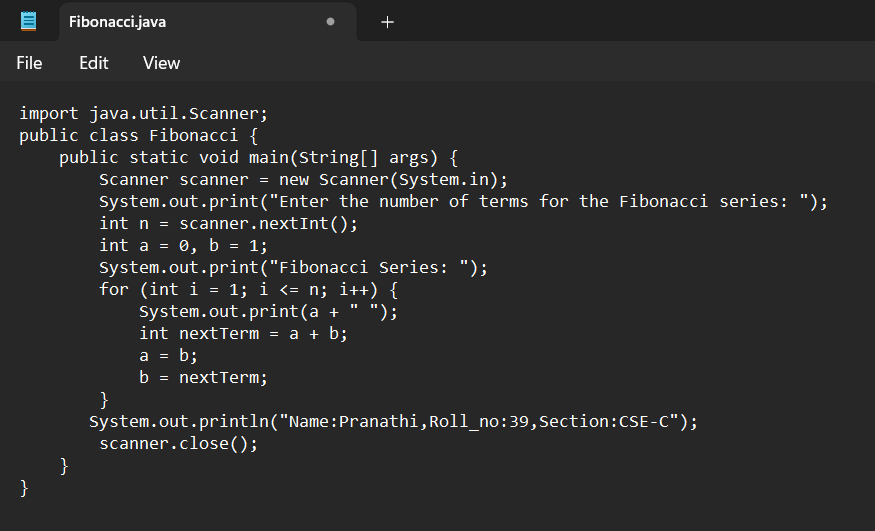
**IMPORTANT POINTS:**

1. int is a data type used to store integer values, and it does not support floating-point numbers.
2. Inside the parentheses of a for loop, the data refers to:
   * Initial expression
   * Test expression
   * Update expression
3. \**factorial = I* is shorthand for factorial = factorial \* I.

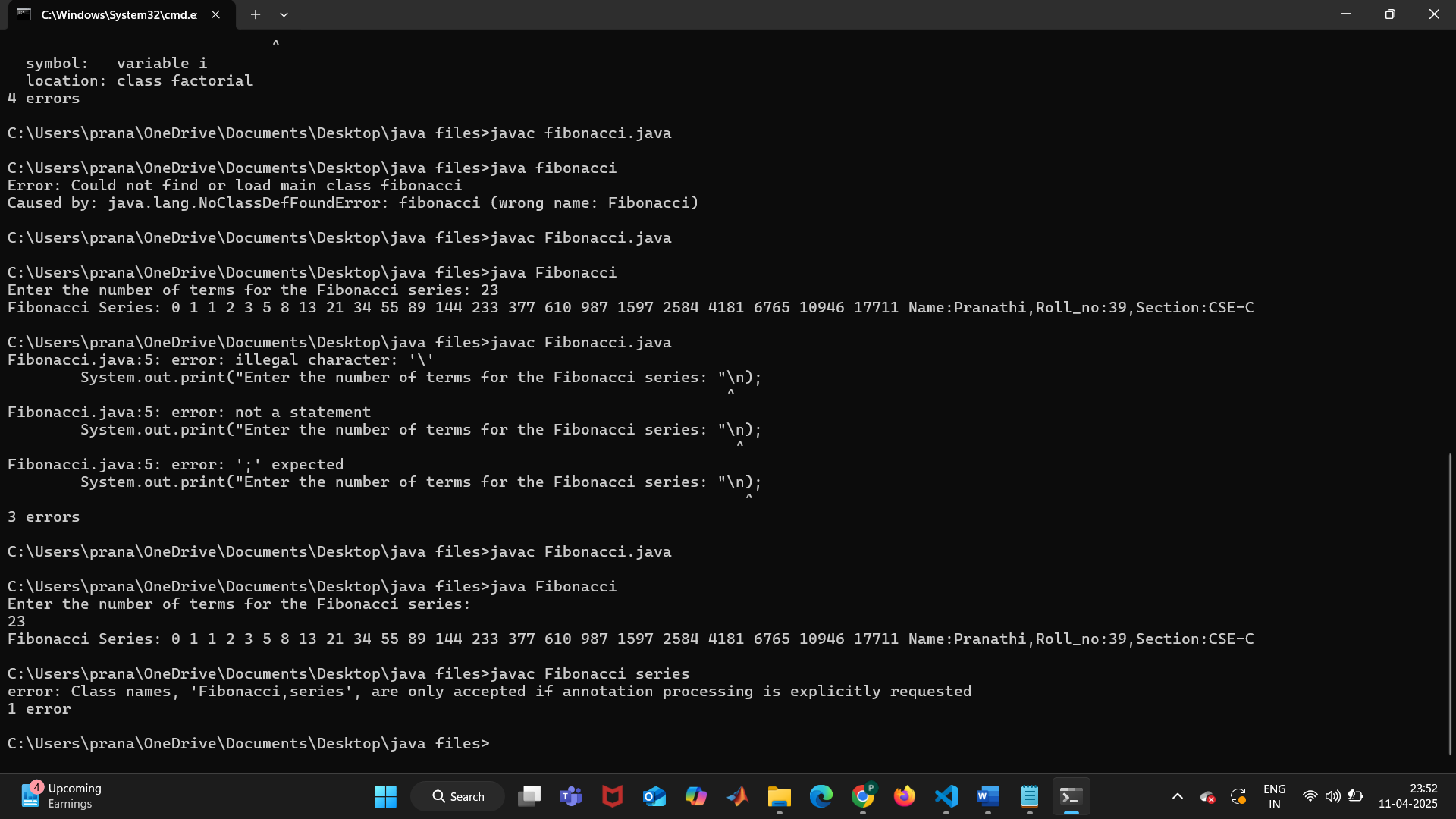
**PROGRAM-7:**

**AIM:** Write a java program for Fibonacci of a number

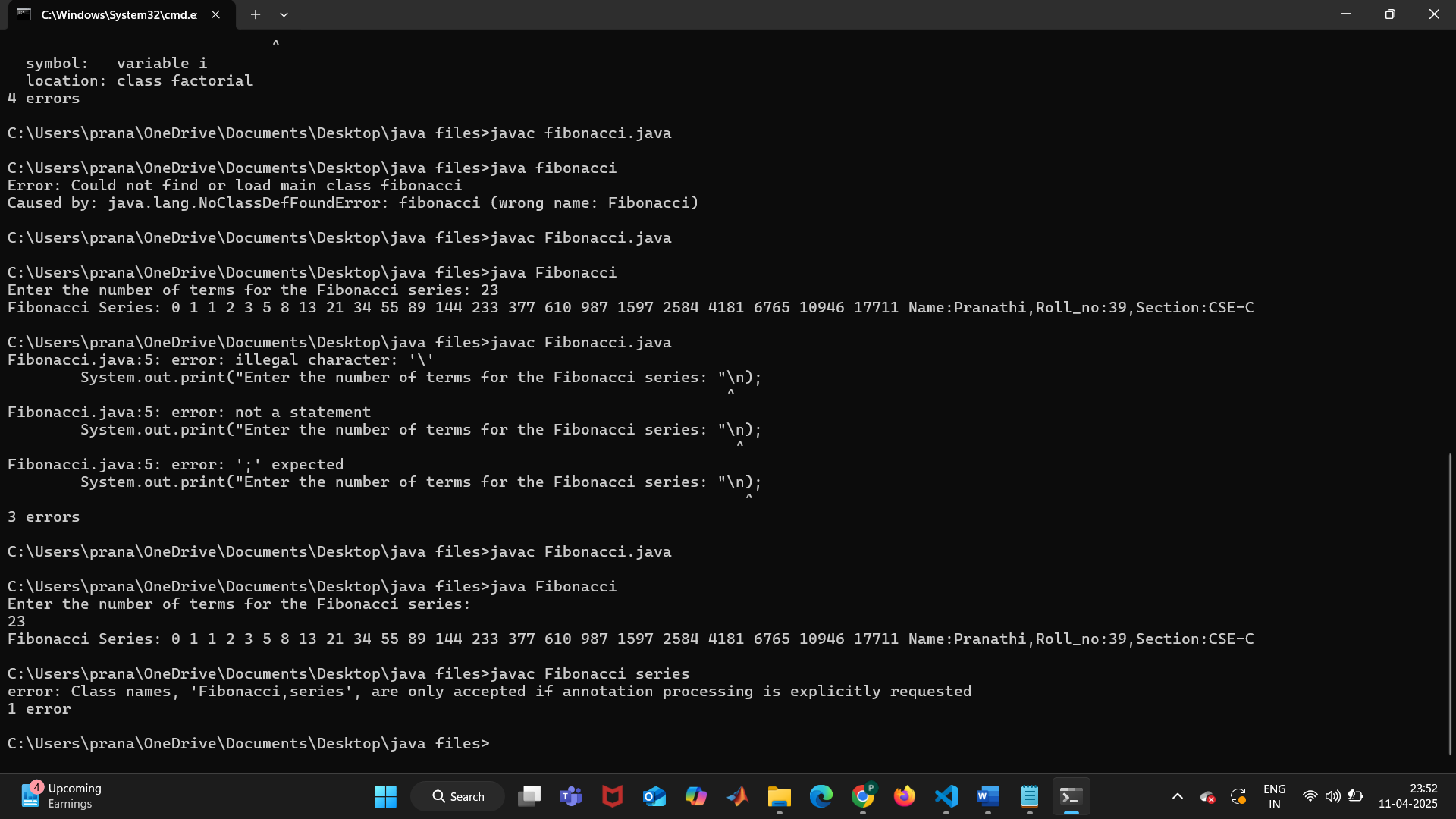
**CODE:**

****

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1)writing small “S”in place of”S”.In system.out.println()  2)not declaring the variables can give error  3)Make sure that you are mentioning the type of the data type .if we not mention data type can gives the error. | 1)code is rectified by keeping capital “S”  2)declaring variables before usage is important.  3)mention the data type of a particular variable ckearly in the code. |

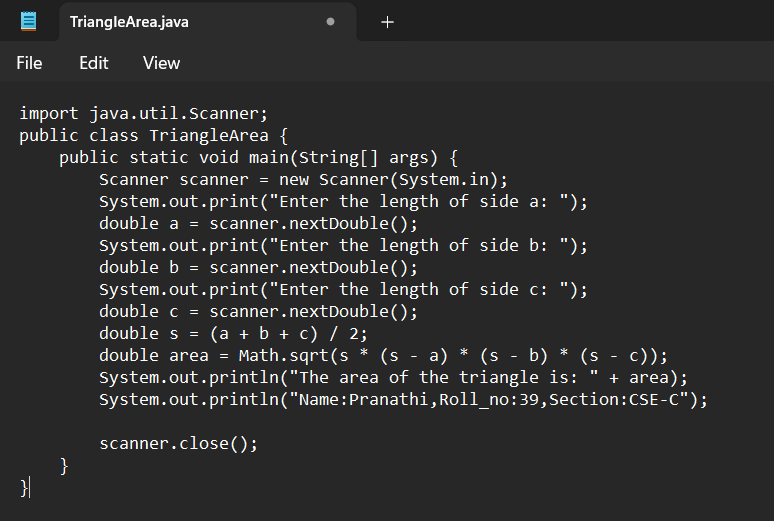
**IMPORTANT POINTS:**

1. The process continues until the specified conditions are satisfied, repeating a certain number of times.
2. In the Fibonacci sequence, the value of the second variable is assigned to the first, and the sum is stored in the second variable.
3. We must have to give first two elements like a=0,b=1. Otherwise we will get the errors.

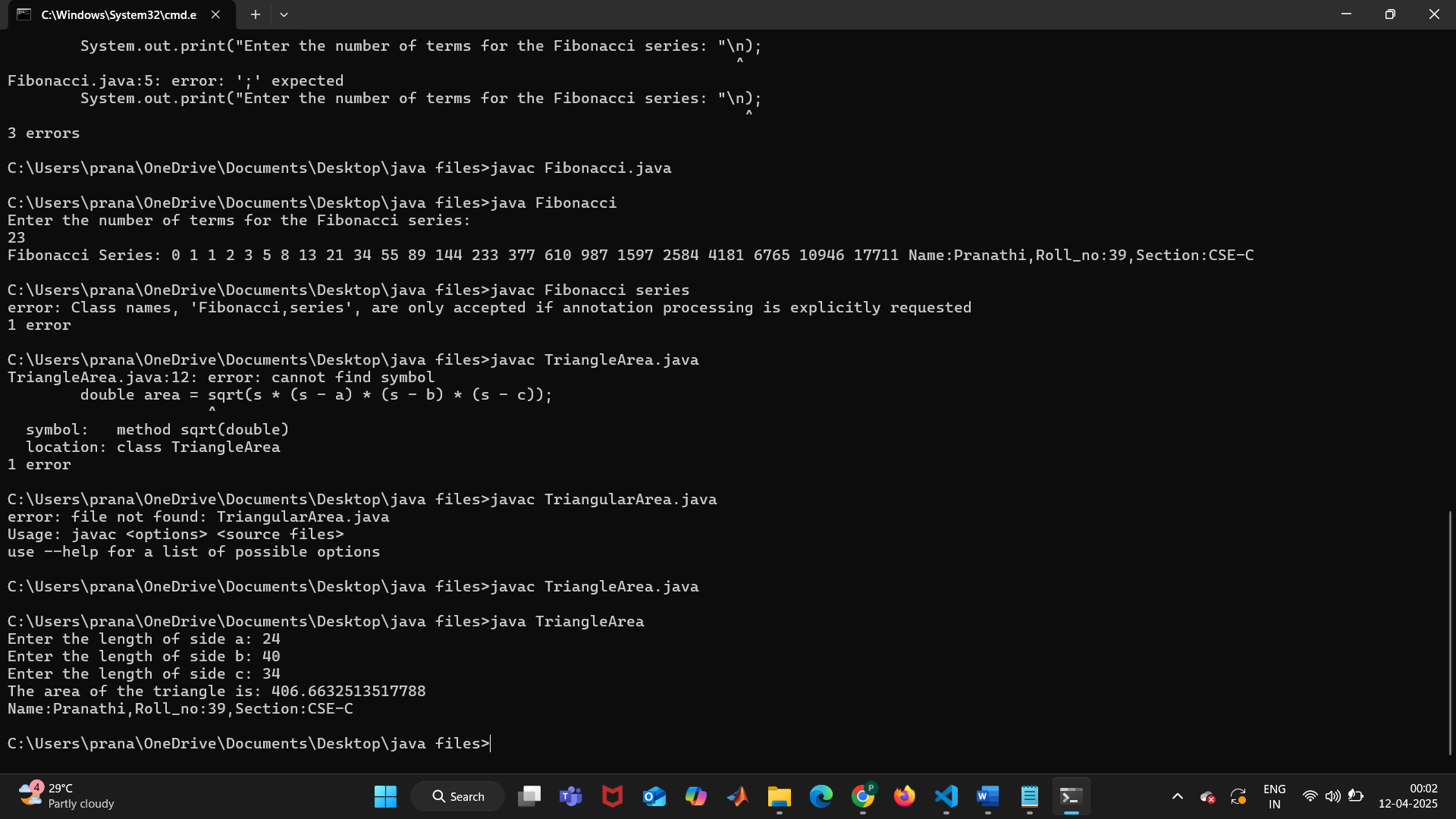
**PROGRAM-8:**

**AIM:** Write a java program for area of the triangle using Heron’s formulae.

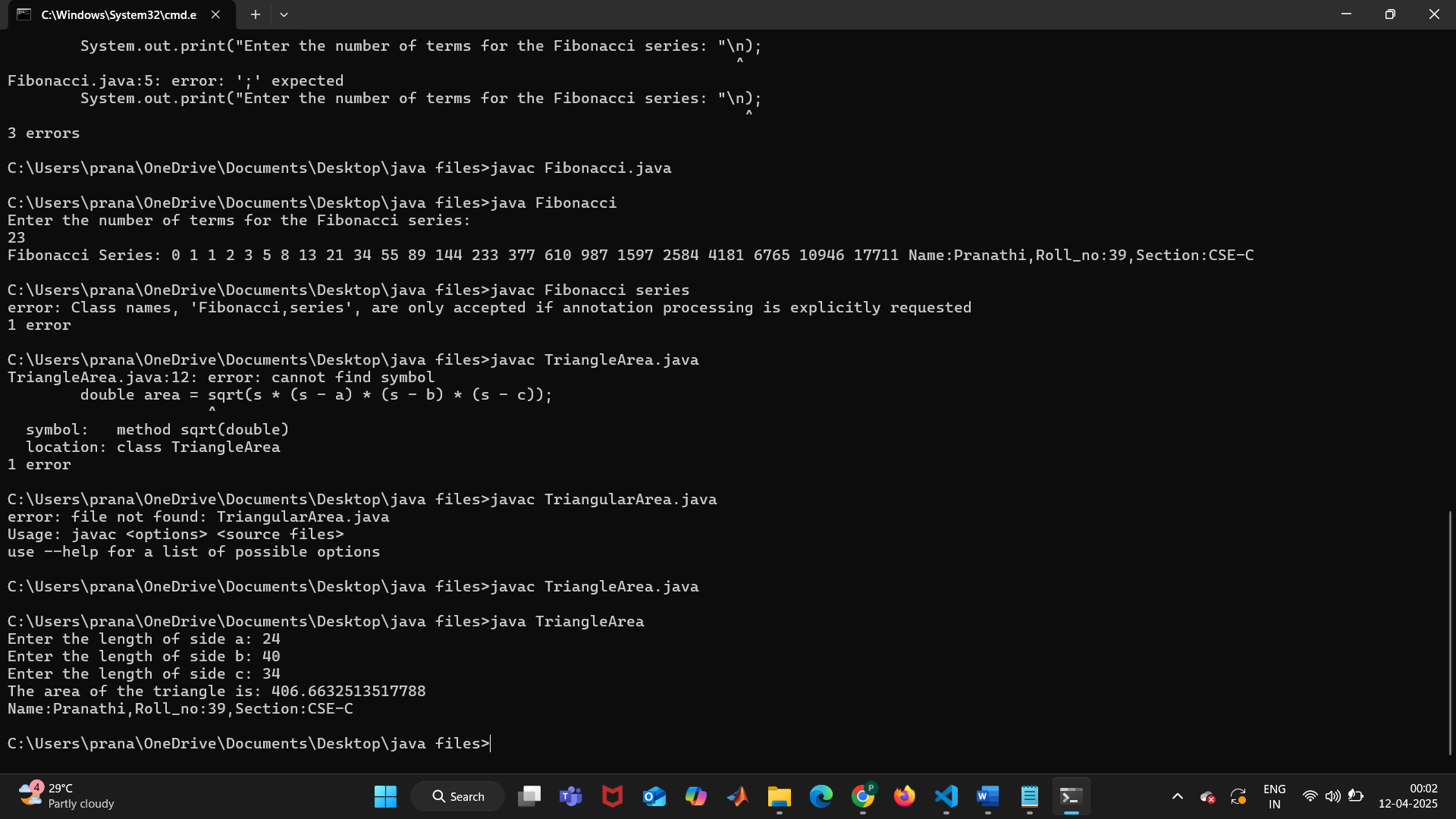
**CODE:**

****

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT;**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1)writing small “S”in place of”S”  In system.out.println()  2)incorrect formulae may leads to correct output but it is incorrect answer  3)if we are not mentioning the math.sqrt then we will get error. | 1)code is rectified by keeping capital “S”  2)check formulae correctly.  3)make sure that you are mentioning the method in the program/code. |

**IMPORTANT POINTS:**

1.Heron’s formula for finding area of the triangle is:

S = (a +b +c)/2

Where S is the semi-perimeter of the triangle.

Now the area formula is:Area = sqrt(s\*(s-a)\*(s-b)\*(s-c)).

**WEEK -3**

**PROGRAM 1:**

**1.AIM:** To create java program with following instructions **:**

1.Create a class with name Car

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

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

4.Create the objects named car, car1,car2

**CLASS DIAGRAM:**

**+---------------------------------+**

**| Cars |**

**+---------------------------------+**

**| - car\_color: String |**

**| - car\_brand: String |**

**| - fuel\_type: String |**

**| - mileage: double |**

**+---------------------------------+**

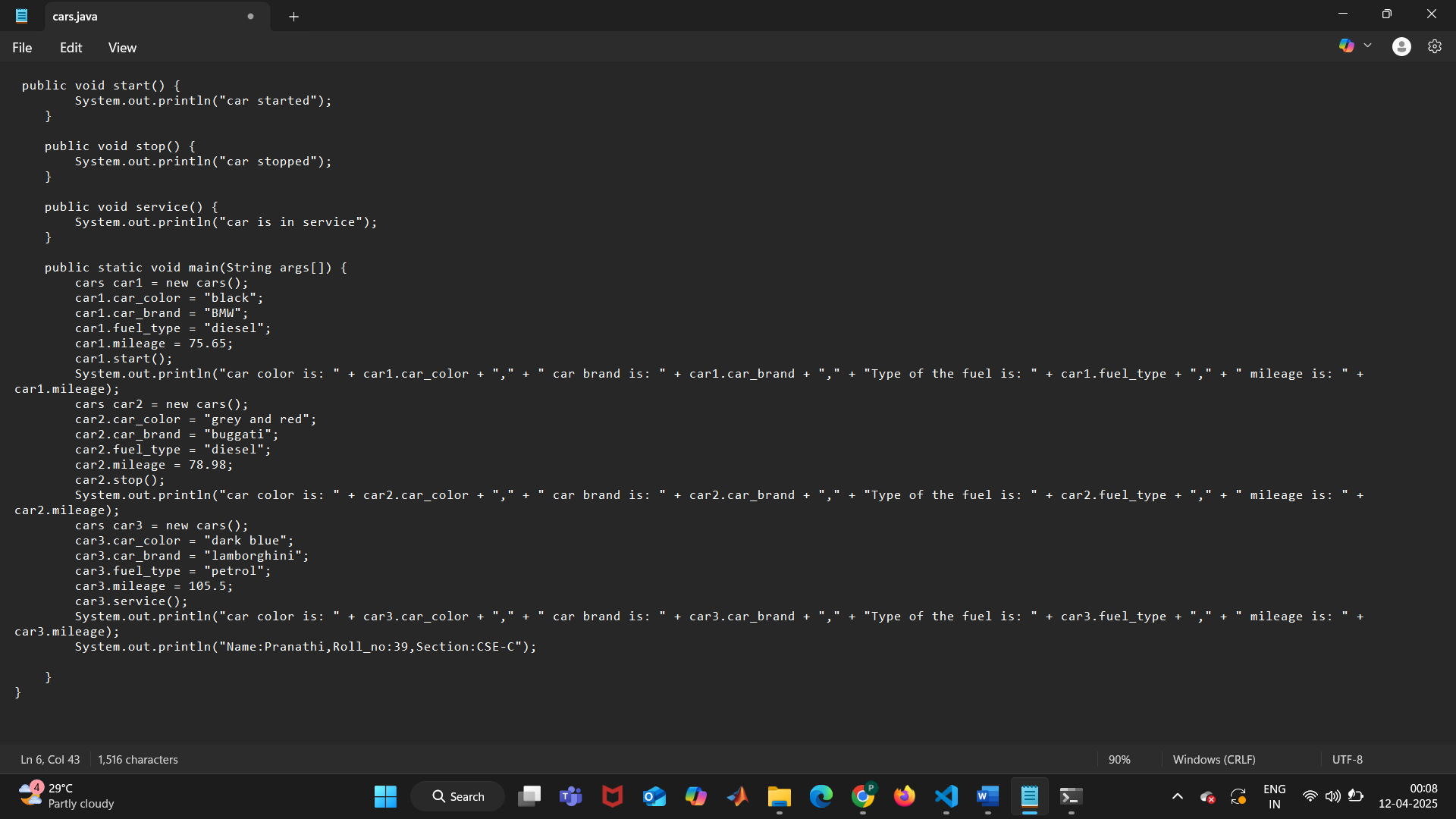
**| + start(): void |**

**| + stop(): void |**

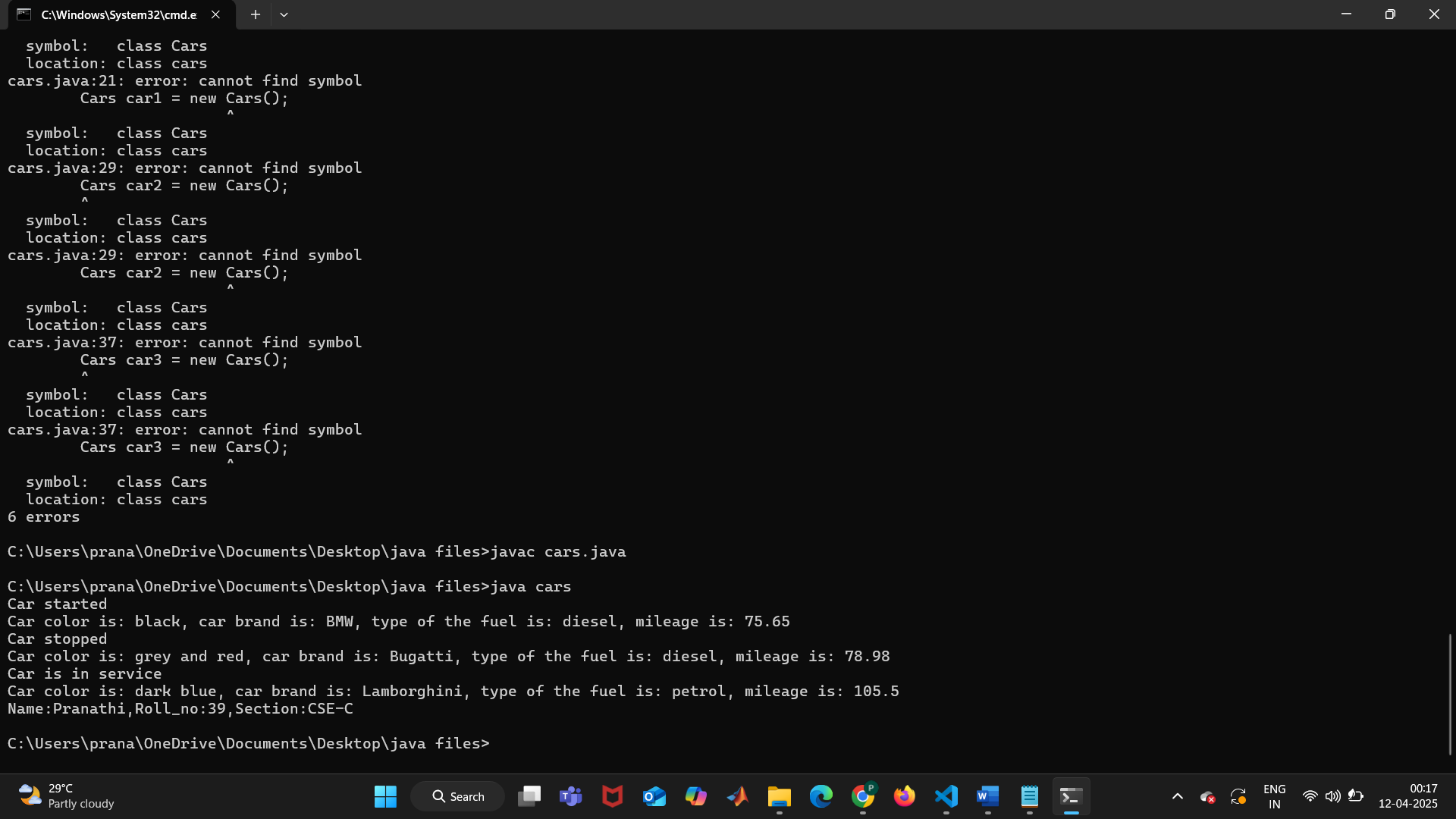
**| + service(): void |**

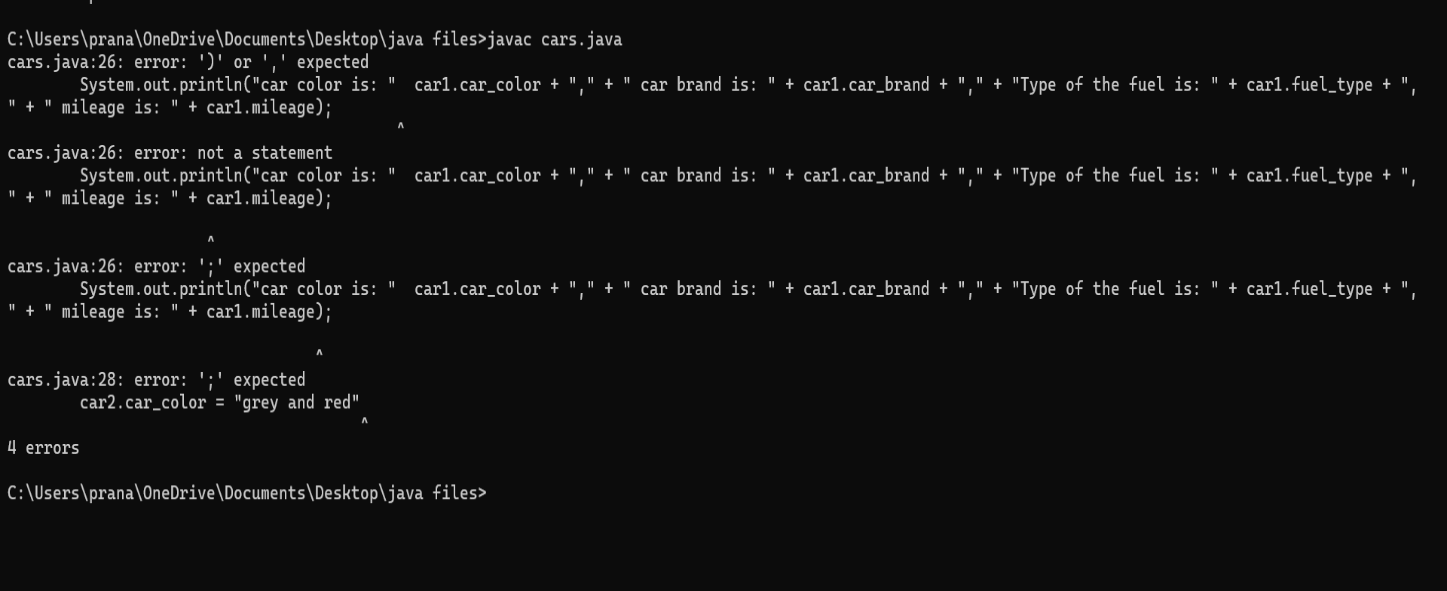
**+---------------------------------+**

**CODE:**



**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT; **

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1. Not putting the semi-colon; after calling the method . 2. After Start, Stop, Service not giving the parenthesis ( ). | 1. keep semi-colon after calling the method in the code. 2. After every method, put the parenthesis ( ). |

**IMPORTANT POINTS:**

1. We need to define the method correctly before using it.
2. The statement **"public void start()"** shows that we are creating a method to execute a function.
3. When we call a method, whatever is inside it gets executed and displayed as output.
4. The values inside the function are known as objects, and we can create multiple objects as needed.

**PROGRAM2:**

**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**

**CLASS DIAGRAM:**

+--------------------------------------------------------------------+

| CustomerAccount |

+--------------------------------------------------------------------+

| - customerName: String |

| - accountID: int |

| - accountBalance: double |

+--------------------------------------------------------------------+

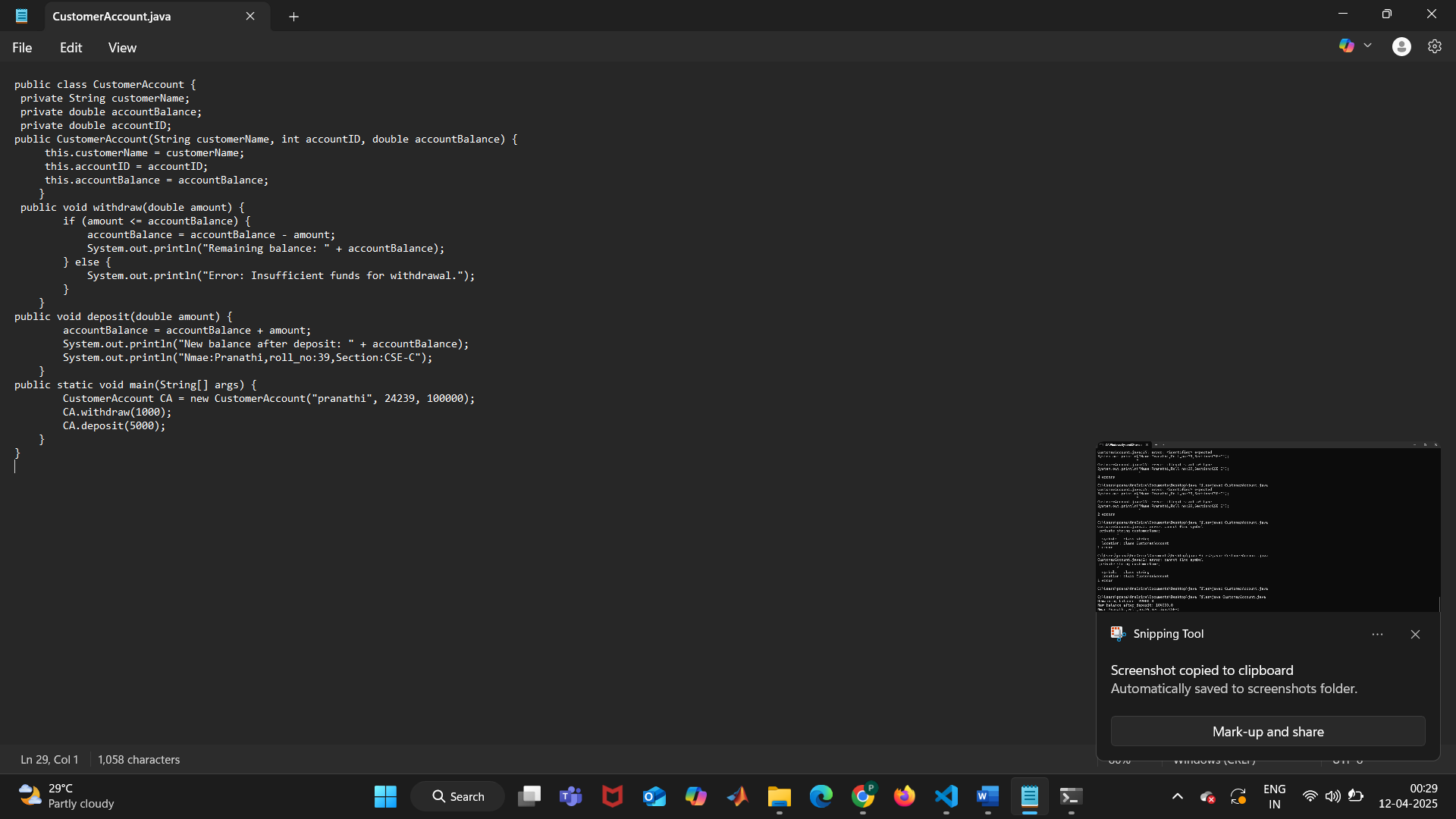
| + CustomerAccount(String, int, double) |

| + withdraw(double): void |

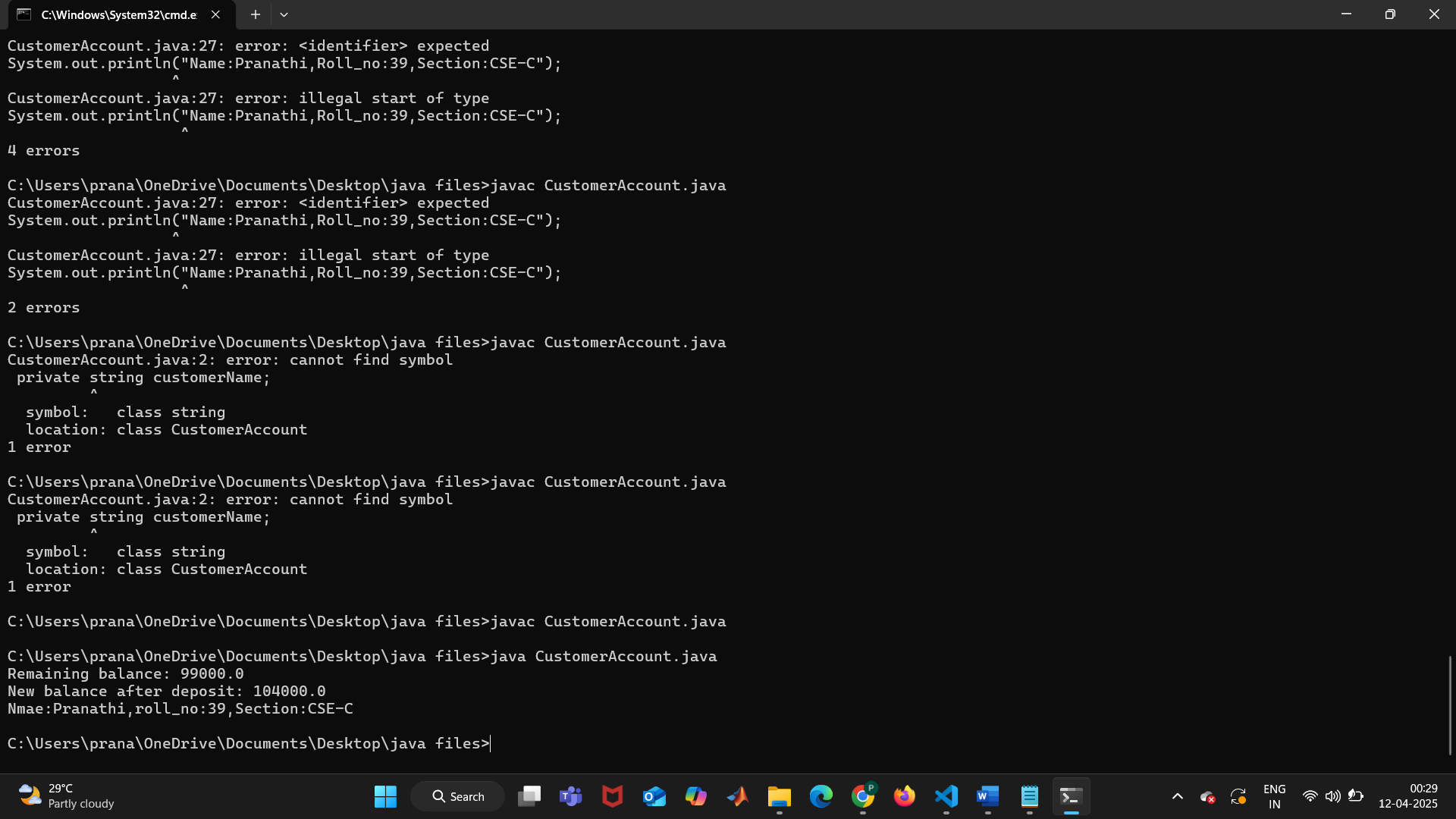
| + deposit(double): void |

+--------------------------------------------------------------------+

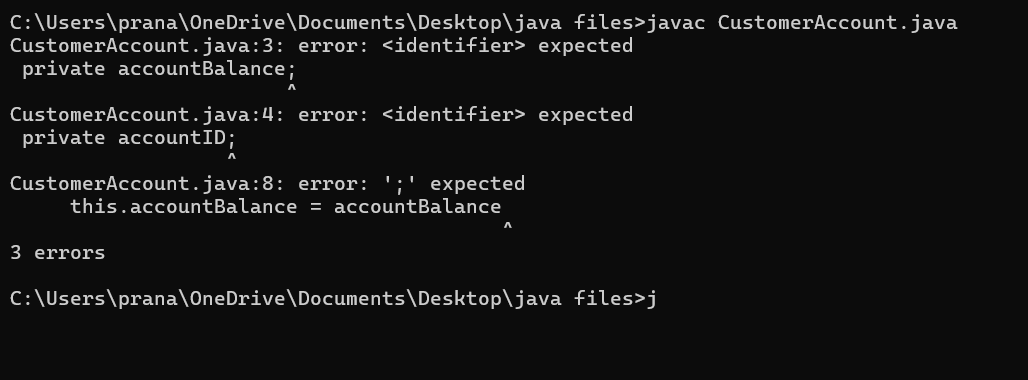
**CODE:**

****

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Not putting the semi-colon; after calling the function. 2. After Start, Stop, Service not giving the parenthesis ( ). | 1. Put the semi-colon after the writing the code. 2. After every method, put the parenthesis ( ). |

**IMPORTANT POINTS:**

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

**WEEK -4**

**PROGRAM – 1:**

**AIM:** Write a java program with class named “book”, the class should contain various attributes such as title, author, year of publication it should also contain a constructor with parameters which initializes, title, author, and year of publication.

Create a method which displays the details of the book and display the details of two books.

**CLASS DIAGRAM:**

+------------------------------------------+

| books |

+--------------------------------------------+

| - title: String |

| - author: String |

| - yearOfPublication: int |

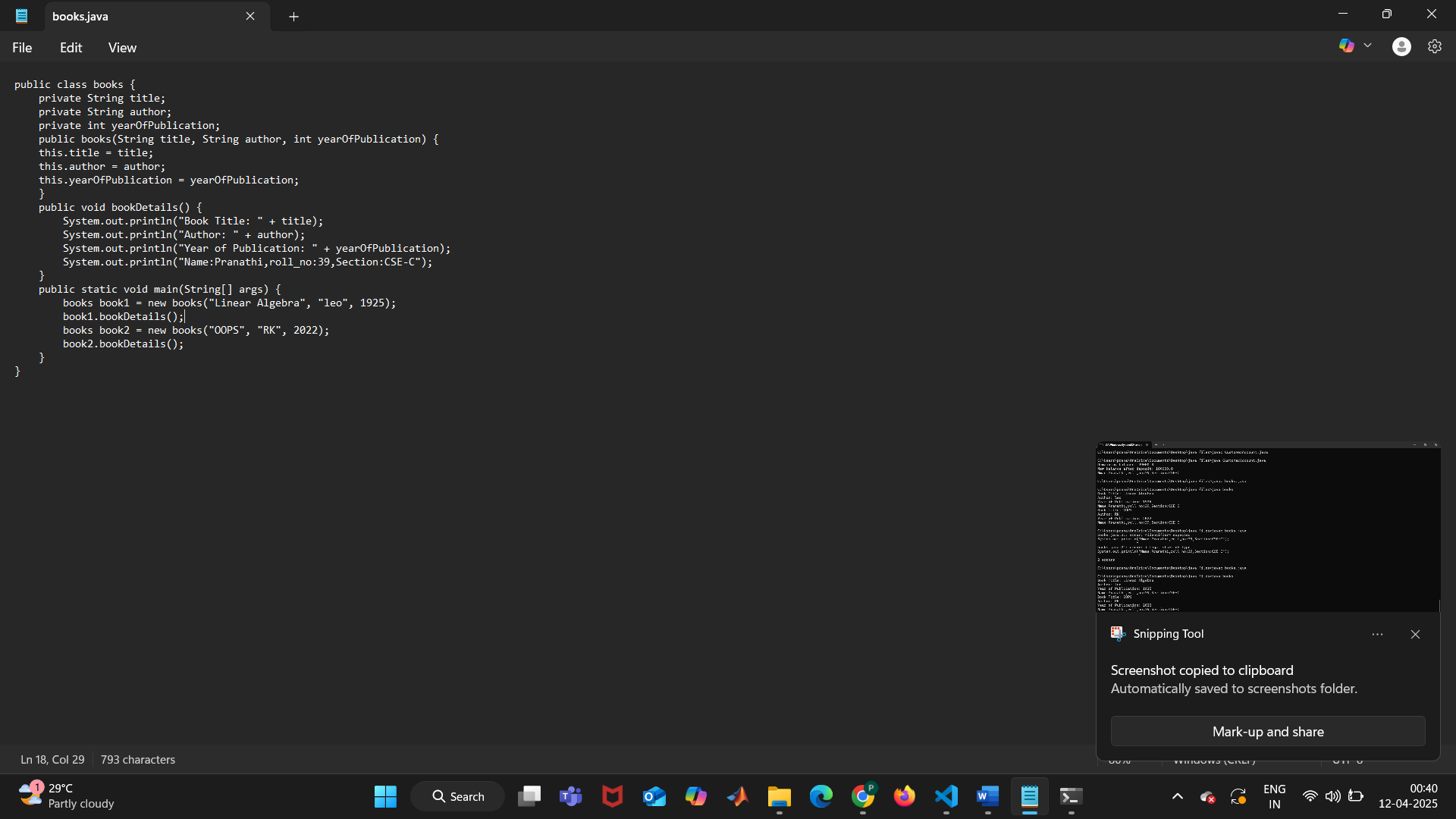
+-------------------------------------------+

| + books(String, String, int) |

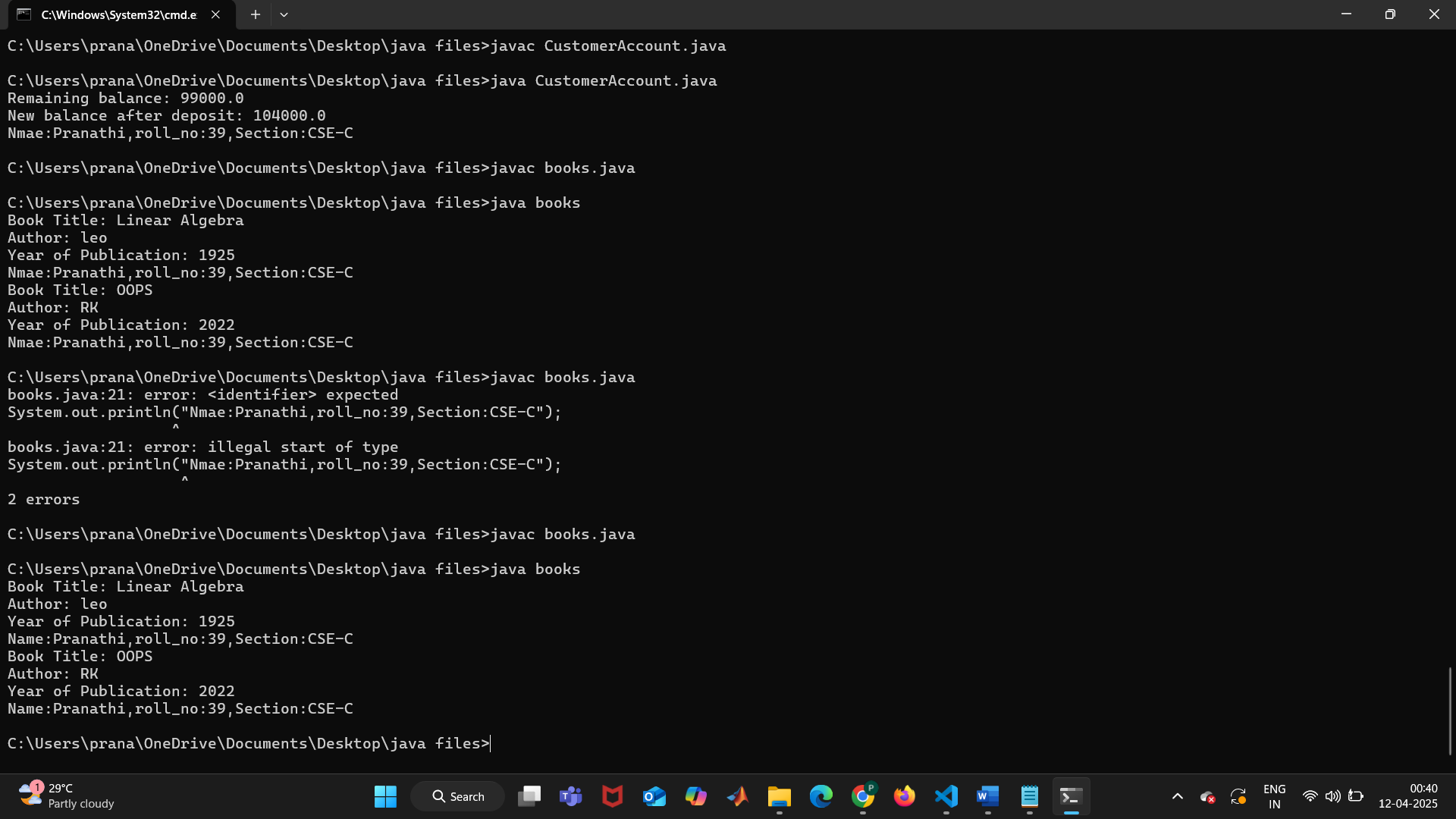
| + bookDetails(): void |

+--------------------------------------------+

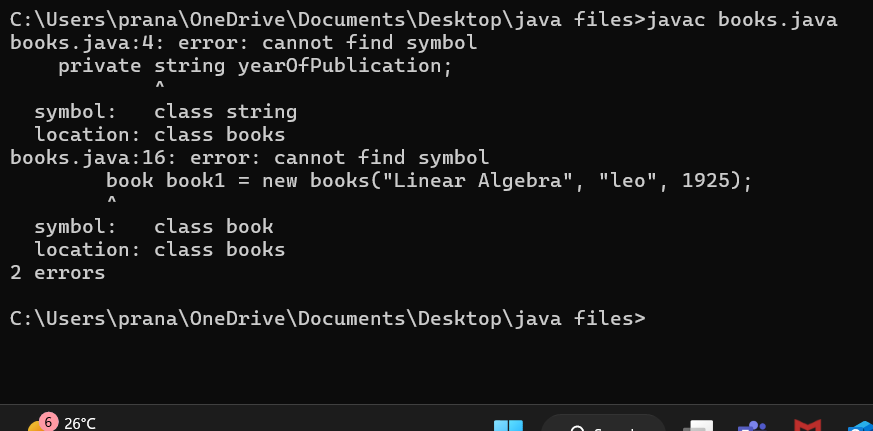
**CODE:**



**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Not defining the function in a file. 2. Two public class files should not be saved in the same file. | 1. To call the method we must define a function in a file. 2. Two public class files should be saved in different files. |

**IMPORTANT POINTS:**

1. While defining two classes for a code, we must be sure that we save both the classes in separate files.
2. While defining a method we should also define a function to call that method.

**PROGRAM – 2:**

**AIM:** Create a java Program with class named myclass with static variable count of int type, initialized to zero and a constant variable “pi” of type double initialized to 3.14 as attributes of the class, ow define a constructor for “myclass” that increments the count variable each time an object of my class is created (count++), finally print the final values of count and pi variables create three objects.

**CLASS DIAGRAM:**

+----------------------------------+

| myclass |

+----------------------------------+

| - count: int (static) |

| - pi: double (final) |

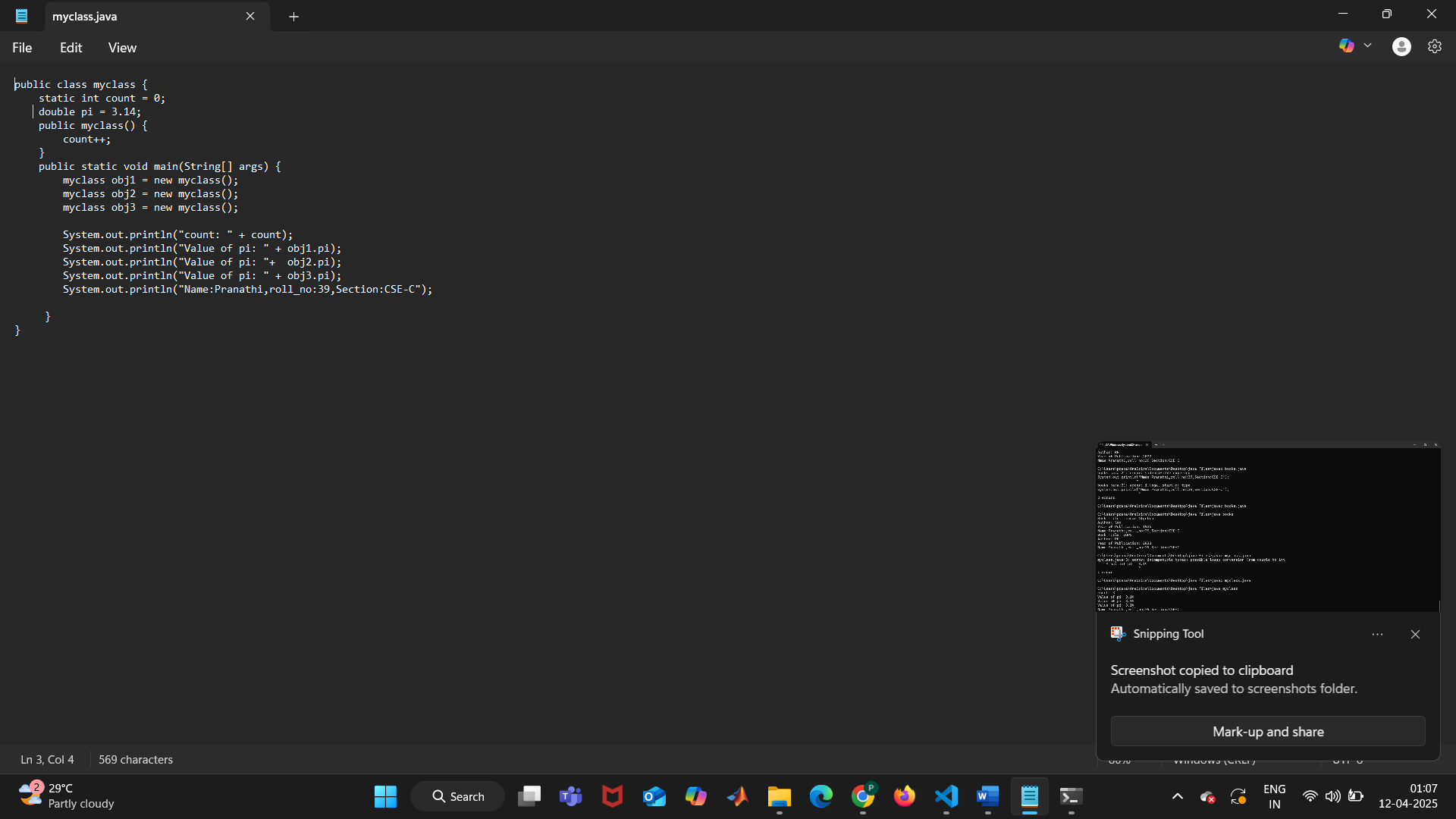
+----------------------------------+

| + myclass() |

| + main(String[]): void |

+-----------------------------------+

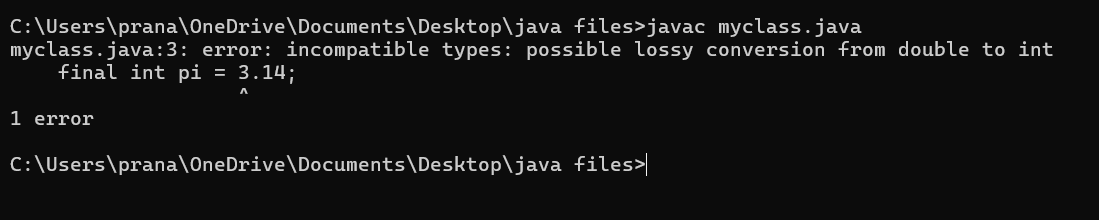
**CODE:**



**POSITIVE CASE OUTPUT:**



**NEGATIVE CASE OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Not Putting the semi-colon after calling a function, 2. Not giving the indentation properly. | 1. Put the semi-colon after calling a function. 2. All the indentation must be correct to run the code correct. |

**IMPORTANT POINTS:**

1) Before setting the final value, we should first assign an initial value to the variable.

2) The main goal is to increase the count based on the number of objects created, meaning the count goes up as more objects are added.

**WEEK-5**

**PROGRAM-1:**

**AIM:** Create a calculator using the operations including addition, subtraction, multiplication, and division using multi-level inheritance and display the desired output. Hint: collect required variables using super class, Create each class for a parameter and each class must contain a method.

**CLASS DIAGRAM:**

**+----------------+**

**| calculator |**

**+----------------+**

**| - a: double |**

**| - b: double |**

**+----------------+**

**| + calculator() |**

**+----------------+**

**▲**

**│**

**+-------------------+--------------------+**

**▲ ▲**

**| |**

**+---------+ +---------------+**

**| Sum | | Subtraction |**

**+---------+ +-----------------+**

**| + sum() | | + sub() |**

**+-----------+ +----------------+**

**▲ │**

**| ▼**

**+------------------+ +----------------+**

**| Multiplication | | Division |**

**+-------------------+ +----------------+**

**| + mult() | | + div() |**

**+--------------------+ +----------------+**

**▲**

**│**

**+----------------------+**

**| Final |**

**+-----------------------+**

**| + displayResults() |**

**+-----------------------+**

**CODE:**

class calculator {

protected double a, b;

public calculator(double a, double b) {

this.a = a;

this.b = b;

}

}

class Addition extends calculator {

public Addition (double a, double b) {super(a, b); }

public double Addition () {

return a + b;

}

}

class Subtraction extends Addition {

public Subtraction(double a, double b) { super(a, b); }

public double sub() {

return a - b;

}

}

class Multiplication extends Subtraction {

public Multiplication(double a, double b) { super(a, b); }

public double mult() {

return a \* b;

}

}

class Division extends Multiplication {

public Division(double a, double b) { super(a, b); }

public double div() {

if (b != 0) {

return a / b;

} else {

System.out.println("Error");

}

}

}

class Final extends Division {

public Final(double a, double b) { super(a, b); }

public void displayResults() {

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

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

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

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

System.out.println("Name:Pranathi,Rollno:39,Section:CSE-C");

}

}

import java.util.Scanner;

public class allcalculator {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

double a = input.nextDouble();

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

double b = input.nextDouble();

Final calc = new Final( a, b);

calc.displayResults();

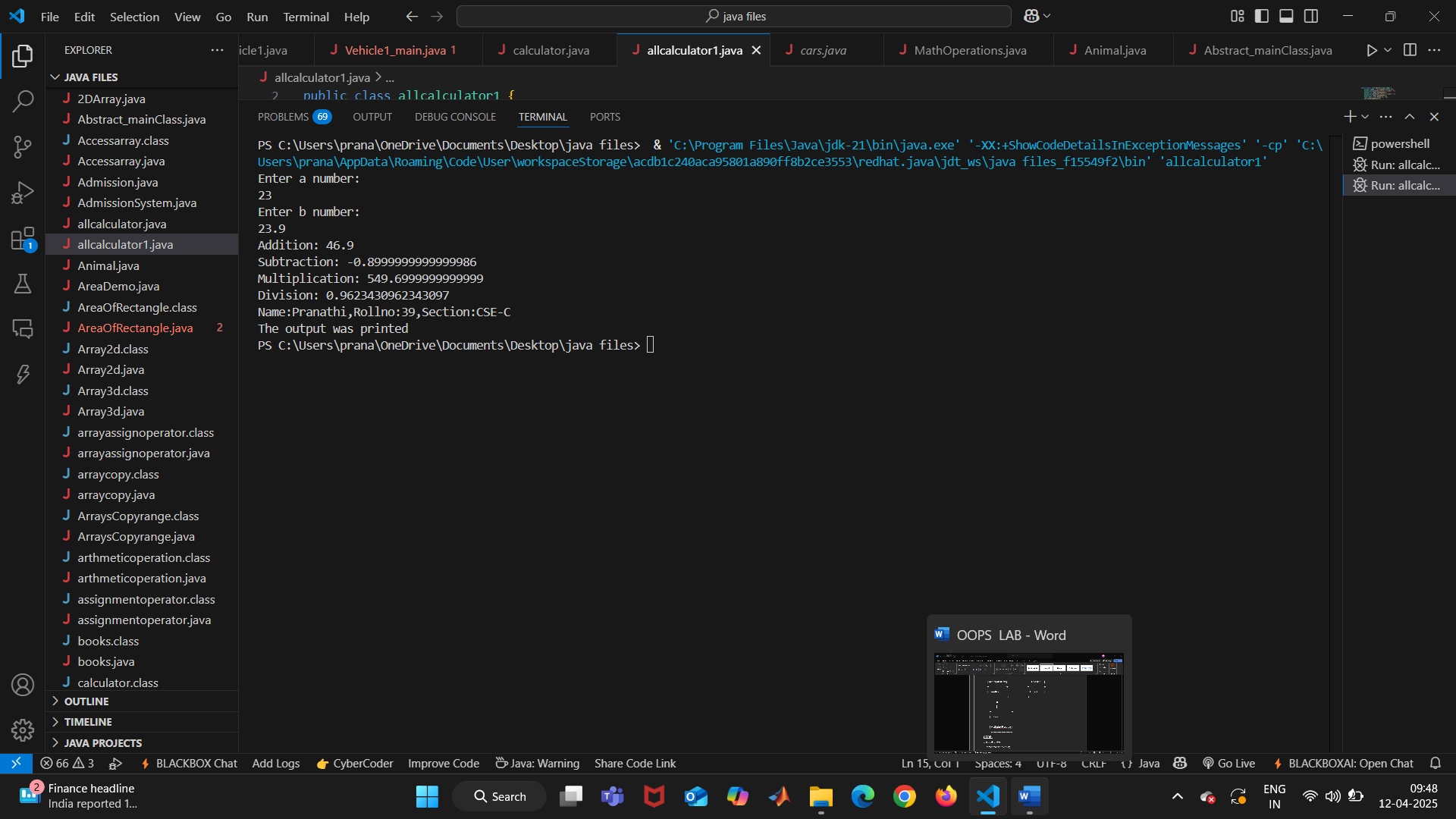
System.out.println("The output was printed");

input.close();

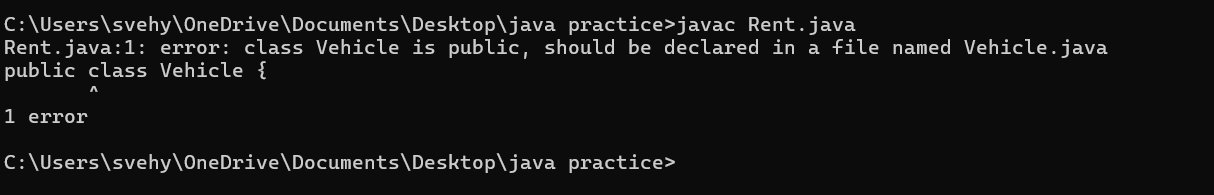
}

}

**POSITIVE CASE OUTPUT:**



**NEGATIVE CASE OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. not providing the return method. 2. Not mentioning super to obtain the super class constructor. 3. removing semicolon can give errors. | 1. After declaring methods, we must provide the return method correctly. 2. To obtain the super class we need to mention super. 3. 3.make sure, you are keeping semi colon. |

**IMPORTANT POINTS:**

1. To get the inputs from the user we use import java.util.Scanner; this is a package.
2. Scanner class is used to get the user input.
3. in java.util.Scanner, the java.util is a package while Scanner is a class of the java.util package.
4. to import a whole package, end the sentence with an asterisk sign(\*).

**PROGRAM-2**

**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
5. Question:
6. Which oops concept is used in the above program
7. If the company decides to add a new type of vehicle, Truck, how would you modify the program?
8. Truck should include an additional property capacity (in tons)
9. Create a showTruckdetails() method to display the truck’s capacity.
10. Write a constructor for Truck that initializes all properties
11. 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.

**CLASS DIAGRAM:**

**+---------------------+**

**| vehicle |**

**+----------------------+**

**| - brand: String |**

**| - speed: int |**

**+----------------------+**

**| + vehicle() |**

**| + start() |**

**| + showDetails() |**

**+-----------------------+**

**▲**

**│**

**+-------------------------+-----------------------------+**

**▲ ▲ ▲**

**| | |**

**+---------------+ +--------------+ +------------+**

**| Car | | Bike | | Truck |**

**+---------------+ +---------------+ +-------------+**

**| - noOfDoors: int | | - hasGears: bool | |- capacity: int |**

**+---------+ +-----------------+ +-------------------+**

**| + Car() | | + Bike() | | + Truck() |**

**| + showDetails() | | + showDetails() | | + showTruck() |**

**+----------------+ +----------------+ +-------------------+**

**▲**

**|**

**+----------------+**

**| rent |**

**+----------------+**

**| + main() |**

**+----------------+**

**CODE:**

public class vehicle {

public String brand;

public int speed;

public vehicle(String brand, int speed) {

this.brand = brand;

this.speed = speed;

}

public void start() {

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

}

public void showDetails() {

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

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

}

}

class Car extends vehicle {

private int noOfDoors;

public Car(String brand, int speed, int noOfDoors) {

super(brand, speed);

this.noOfDoors = noOfDoors;

}

public void showDetails() {

super.showDetails();

System.out.println("Number of Doors: " + noOfDoors);

}

}

class Bike extends vehicle {

private boolean hasGears;

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

super(brand, speed);

this.hasGears = hasGears;

}

public void showDetails() {

super.showDetails();

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

}

}

class Truck extends vehicle {

private int capacity;

public Truck(String brand, int speed, int capacity) {

super(brand, speed);

this.capacity = capacity;

}

public void showTruck() {

super.showDetails();

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

}

}

public class rent {

public static void main(String[] args) {

Car car = new Car("Buggati",140,8);

Bike bike = new Bike("Duke",250, true);

Truck truck = new Truck("TATA", 100, 2);

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

car.start();

car.showDetails();

System.out.println("Bike Details");

bike.start();

bike.showDetails();

System.out.println("Truck Details");

truck.start();

truck.showTruck();

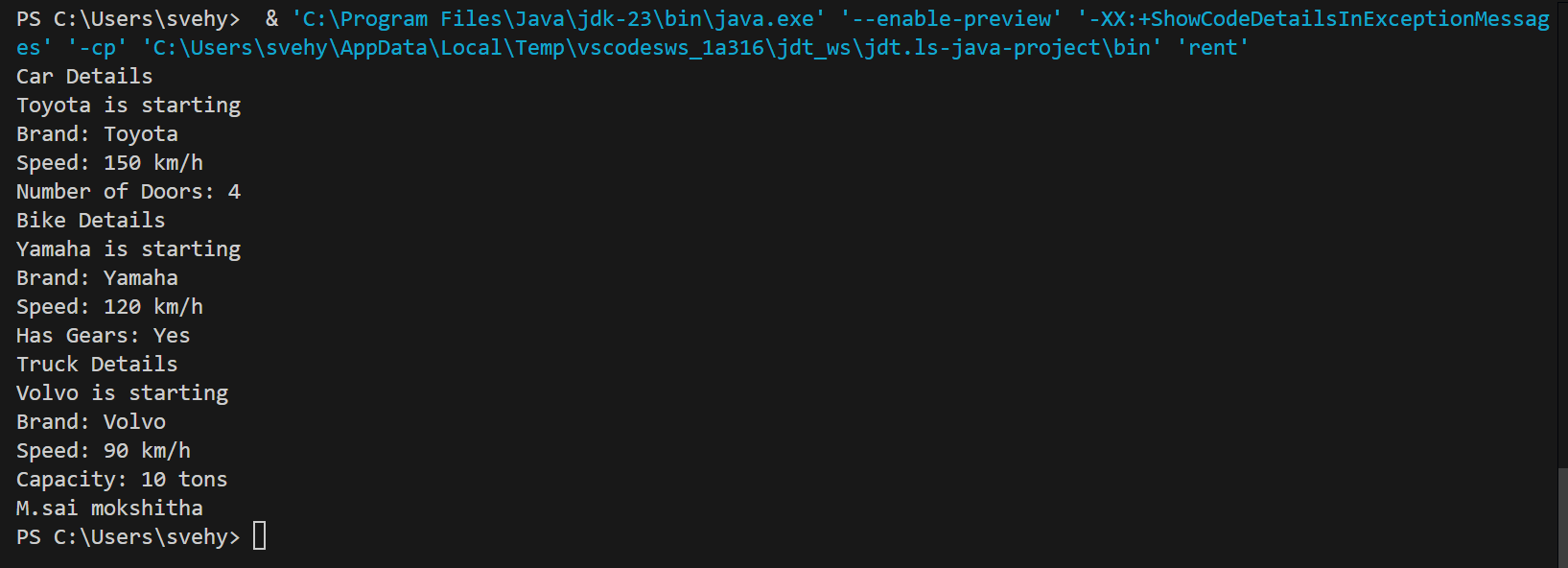
        System.out.println("Name:Pranathi,Rollno:39,Section:CSE-C");

System.out.println("the output was printed");

}

}

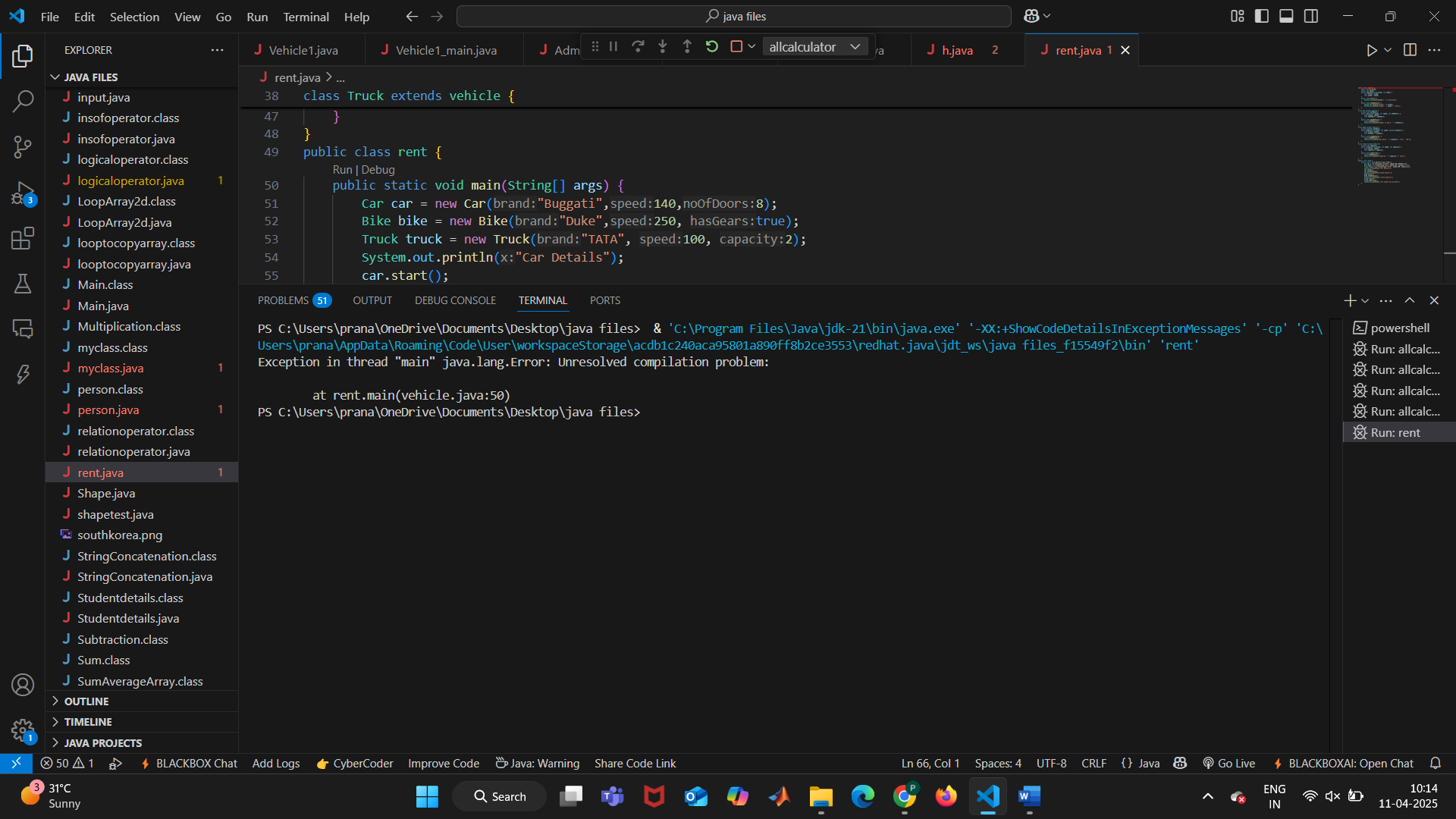
**OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Declaring two superclasses inside the same file. 2. Not declaring the variable using ‘this’ keyword inside the constructor. | 1. Make two separate files to save the two super classes. 2. Declare the variable using this keyword to run the program. |

**NEGATIVE CASE:**



**IMPORTANT POINTS:**

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.
4. the void keyword in java is used to specify that a method does not return any value. it is a return type that indicates the method performs a function and doesn't produce a result.

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**

**PROGRAM-1:**

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

**CLASS DIAGRAM:**

+------------------+

| Vehicle1 |

+------------------+

| - car\_name: String|

| - car\_color: String|

| - fuel\_type: String|

+------------------+

| + Vehicle1(car\_name: String, car\_color: String, fuel\_type: String)|

| + displayinfo(): void|

+------------------+

▲

|

+------------------+

| Car |

+------------------+

| (inherits Vehicle1)|

+------------------+

| + Car(car\_name: String, car\_color: String, fuel\_type: String)|

| + displayinfo(): void|

+------------------+

**CODE:**

public class Vehicle1 {

    public String car\_name;

    public String car\_color;

    public String fuel\_type;

    public Vehicle1(String car\_name, String car\_color, String fuel\_type){

    this.car\_name = car\_name;

    this.car\_color = car\_color;

    this.fuel\_type = fuel\_type;

    }

    public void displayinfo() {

        System.out.println("Car Model: " + car\_name);

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

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

        System.out.println("these are the  details about the car-buggati .");

    }

}

class Car extends Vehicle1 {

    public Car(String car\_name, String car\_color, String fuel\_type) {

        super(car\_name, car\_color, fuel\_type);

    }

     public void displayinfo() {

        super.displayinfo();

        System.out.println("these are the  details about the car-lamborghini.");

    }

}

   public class Vehicle1\_main{

    public static void main(String[]args) {

        System.out.println("the model ,color,fuel and type of a car");

        Vehicle1 v = new  Vehicle1("buggati", "White", "Diesel");

        v.displayinfo();

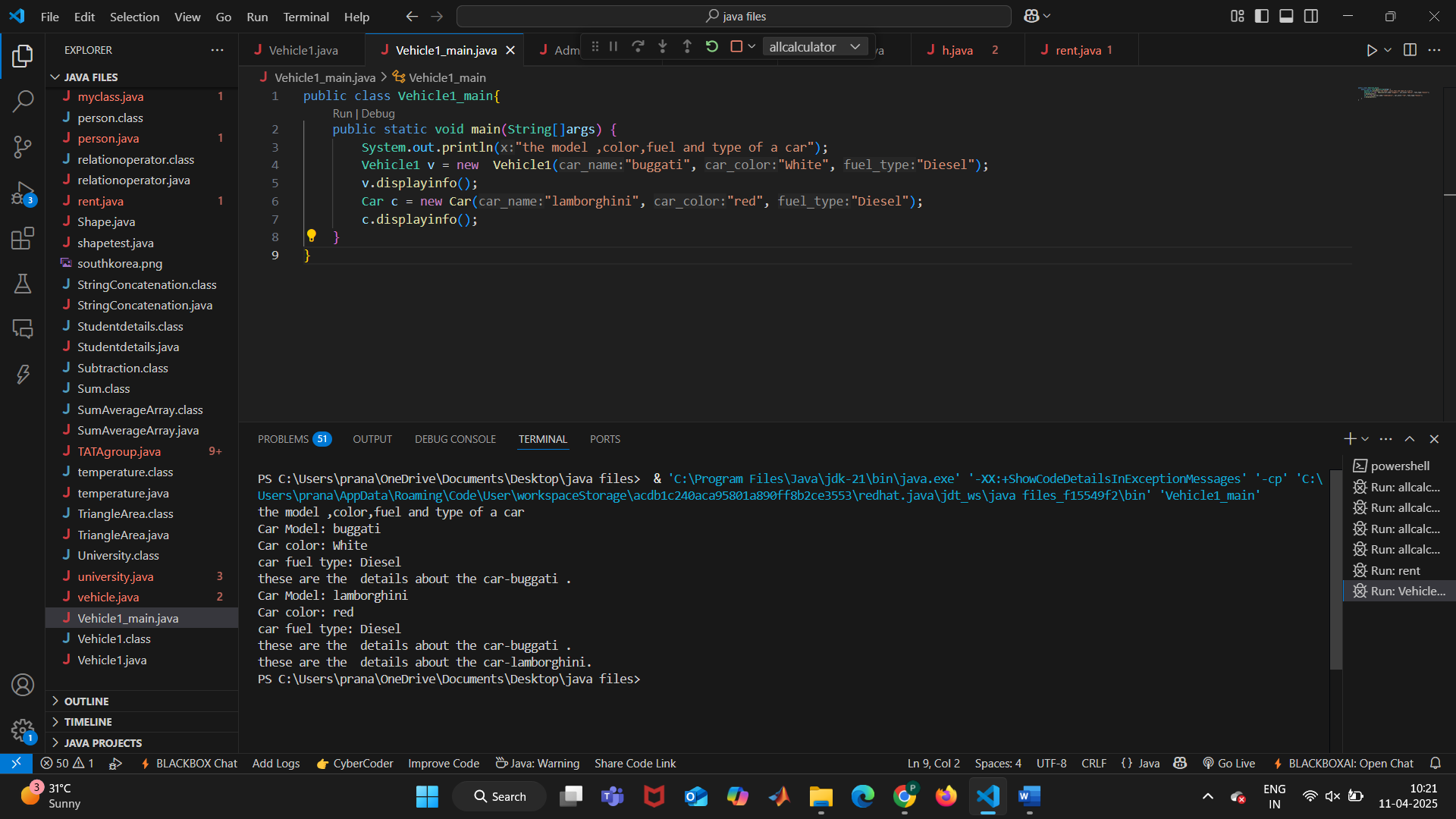
        Car c = new Car("lamborghini", "red", "Diesel");

        c.displayinfo();

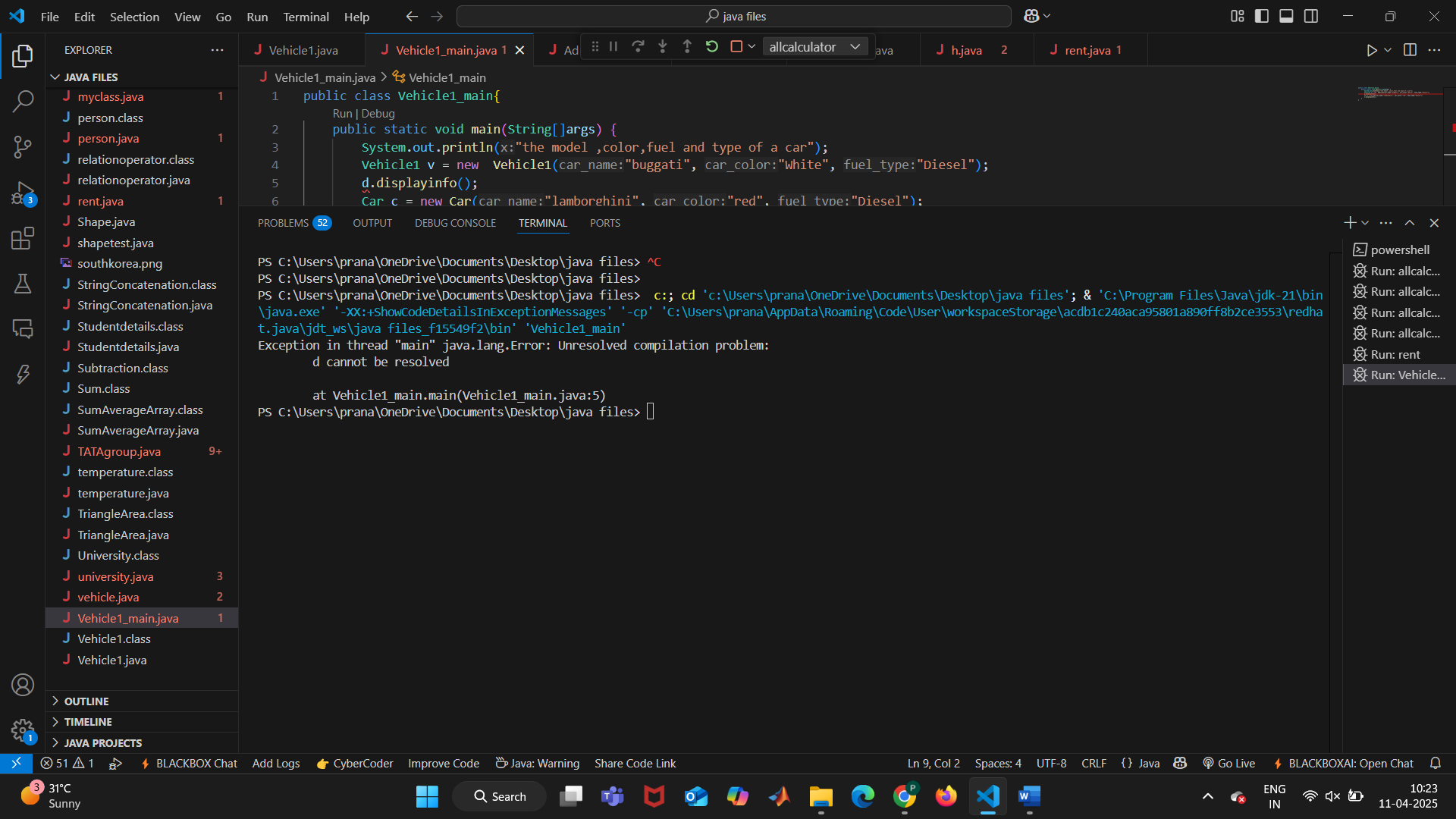
    }

}

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| |  | | --- | | **1.Incorrect class name for** main **method (**Truck**)** | | **2.Inconsistent car model output in** displayInfo() | | |  | | --- | | 1.Rename Truck to Main or place main inside Car or Vehicle. | | 2.Ensure Car correctlypasses  Benz" to super(car\_model,  color, fuel\_type);. | |

**IMPORTANT POINTS:**

**Inheritance:** The Car class extends the Vehicle class, demonstrating **inheritance** in Java.

**Constructor Chaining:**The Car class calls the parent constructor using super(car\_model, color, fuel\_type); to initialize inherited attributes.

**Method Overriding:**The Car class overrides the displayInfo() method from Vehicle and calls super.displayInfo() to reuse the parent method before adding its own output.

**Incorrect** main **Class Name:**The main method is inside Truck, which is unrelated to Vehicle and Car. The class should be renamed for clarity.

**PROGRAM-2:**

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

1. Add two integers
2. Add two double

C.Add three integer

**CLASS DIAGRAM:**

+--------------------+

| MathOperations |

+--------------------+

| +sum(int, int): int |

| +sum(double, double): double |

| +sum(int, int, int): int |

+--------------------+

**CODE:**

public class MathOperations {

    public int sum(int x, int y) {

        return x + y;

    }

    public double sum(double x, double y) {

        return x + y;

    }

    public int sum(int x, int y, int z) {

        return x + y + z;

    }

}

public class RunCalculator {

    public static void main(String[] args) {

        MathOperations calc = new MathOperations();

        System.out.println("Sum of two integers: " + calc.sum(10, 20));

        System.out.println("Sum of two doubles: " + calc.sum(5.5, 4.5));

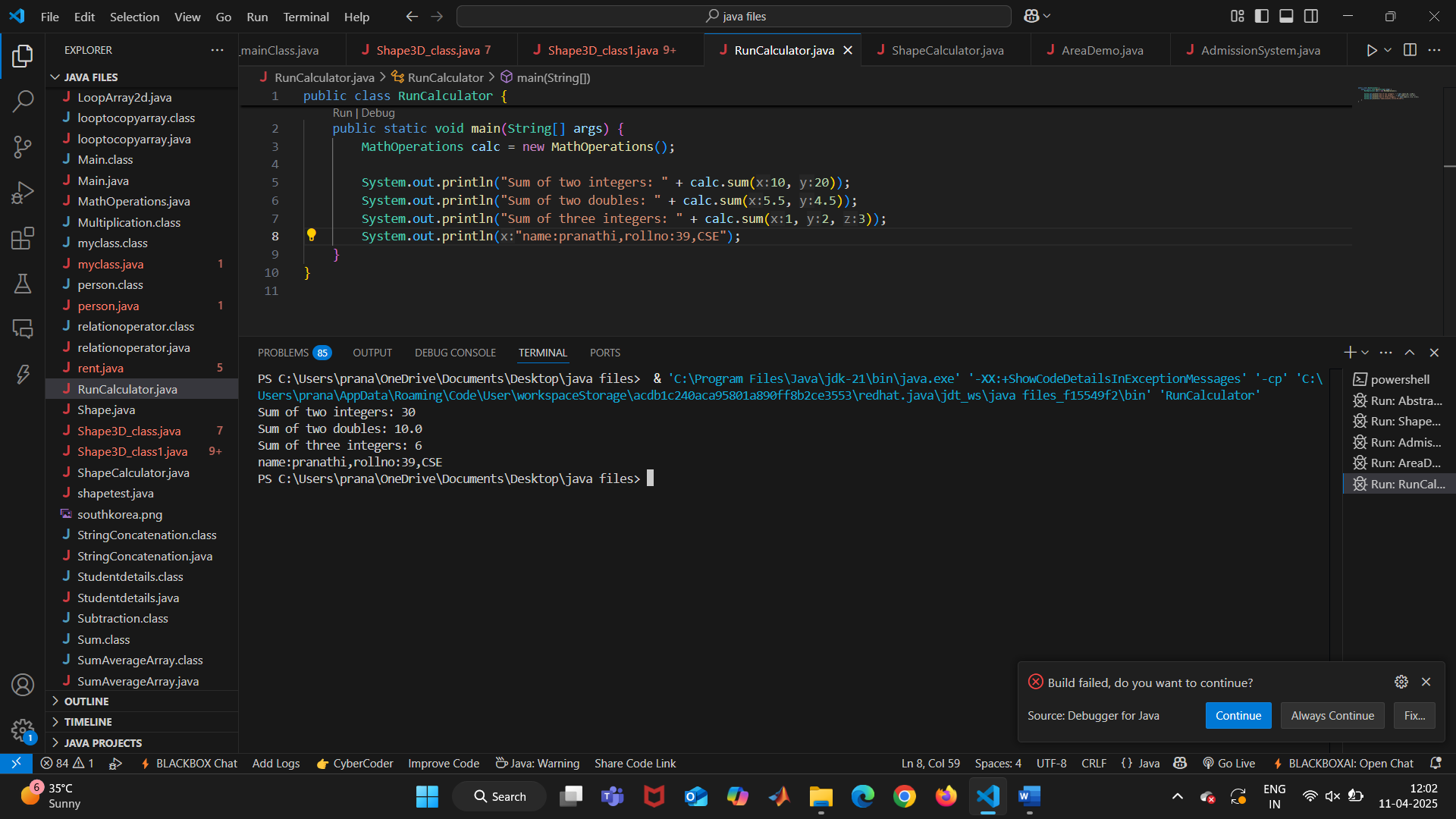
        System.out.println("Sum of three integers: " + calc.sum(1, 2, 3));

 System.out.println("name:pranathi,rollno:39,CSE");

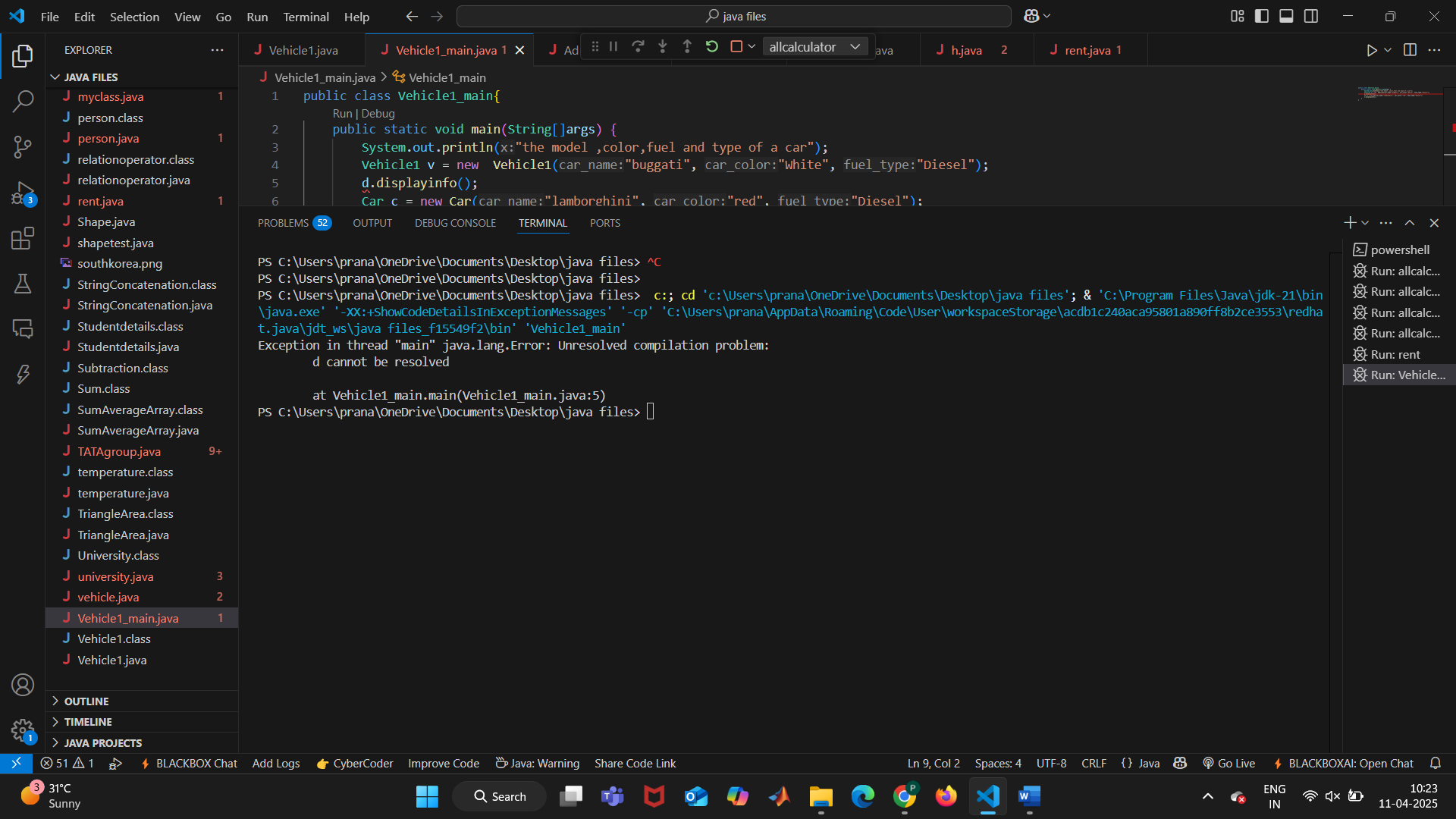
    }

}

**POSITIVE CASE OUTPUT:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Method parameters missing spaces   2. Inconsistent indentation in method bodies | 1. Add proper spacing between 2.Fix indentation |

**IMPORTANT POINTS:**

**1.Method Overloading:** The add method is overloaded with different parameter types and counts, demonstrating compile-time polymorphism.

**2.Automatic Method Selection:** Java selects the appropriate add method based on the argument types during compilation.

**PROGRAM-3:**

**AIM:** Create a shape class with a method CalculateArea() that is overloaded for different shpaes (e.g square, rectangle) then, create a subclass circle that overrides the calculatearea() method for a circle.

**CLASS DIAGRAM:**

**+-----------------------------------+**

**| ShapeCalculator |**

**+-----------------------------------+**

**| +findArea(double): double |**

**| +findArea(int, int): int |**

**+------------------------------------+**

**▲**

**|**

**|**

**+------------------------------------------------+**

**| CircleArea |**

**+------------------------------------------------+**

| +findArea(double): double [overridden] |

**+------------------------------------------------+**

**CODE:**

public class ShapeCalculator {

public double findArea(double side) {

return side \* side;

}

public int findArea(int length, int width) {

return length \* width;

}

}

class CircleArea extends ShapeCalculator {

@Override

public double findArea(double radius) {

return 3.1416 \* radius \* radius;

}

}

public class AreaDemo {

public static void main(String[] args) {

ShapeCalculator shape = new ShapeCalculator();

System.out.println("Square area: " + shape.findArea(4.2));

System.out.println("Rectangle area: " + shape.findArea(7, 5));

CircleArea circle = new CircleArea();

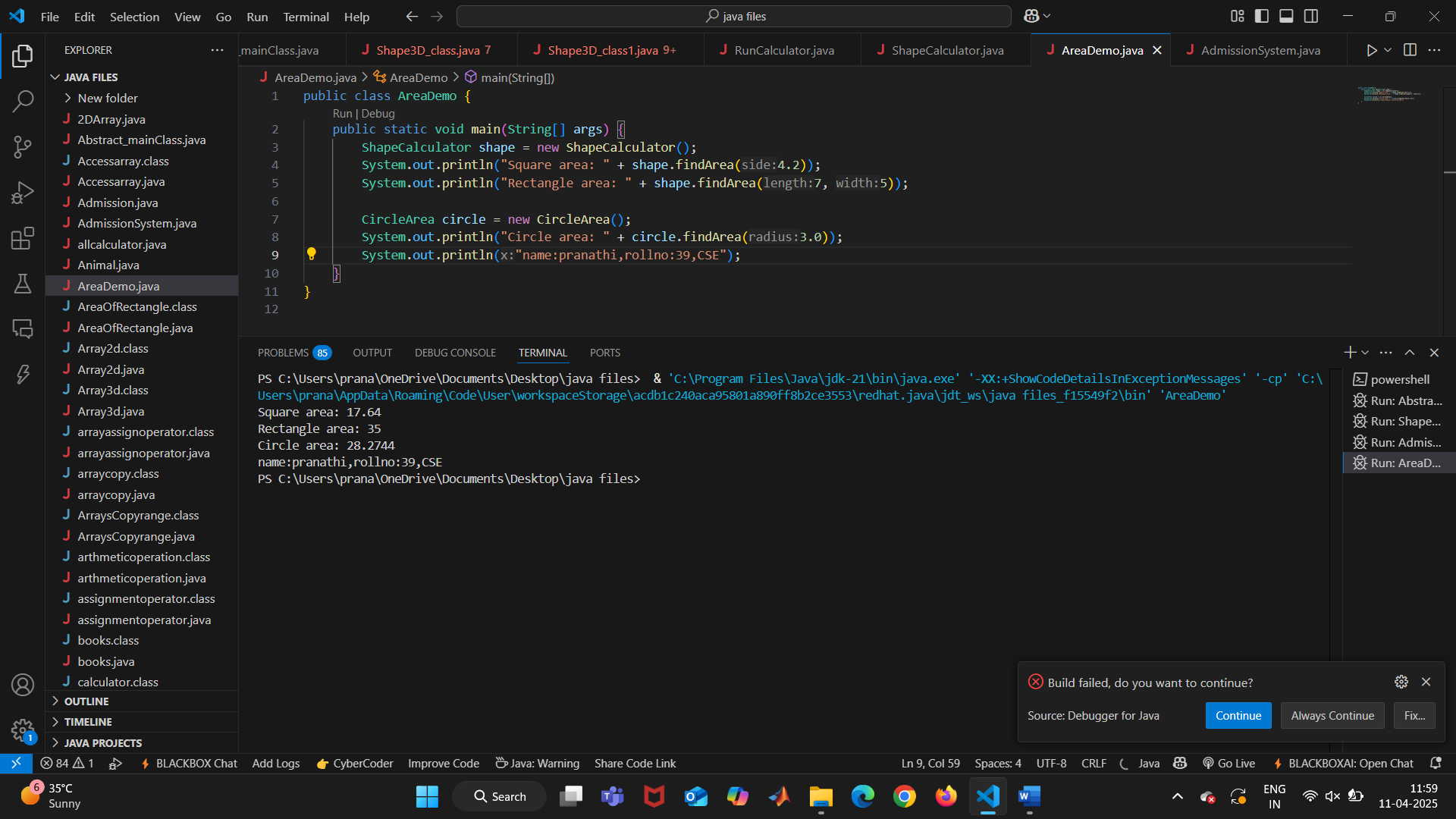
System.out.println("Circle area: " + circle.findArea(3.0));

 System.out.println("name:pranathi,rollno:39,CSE");

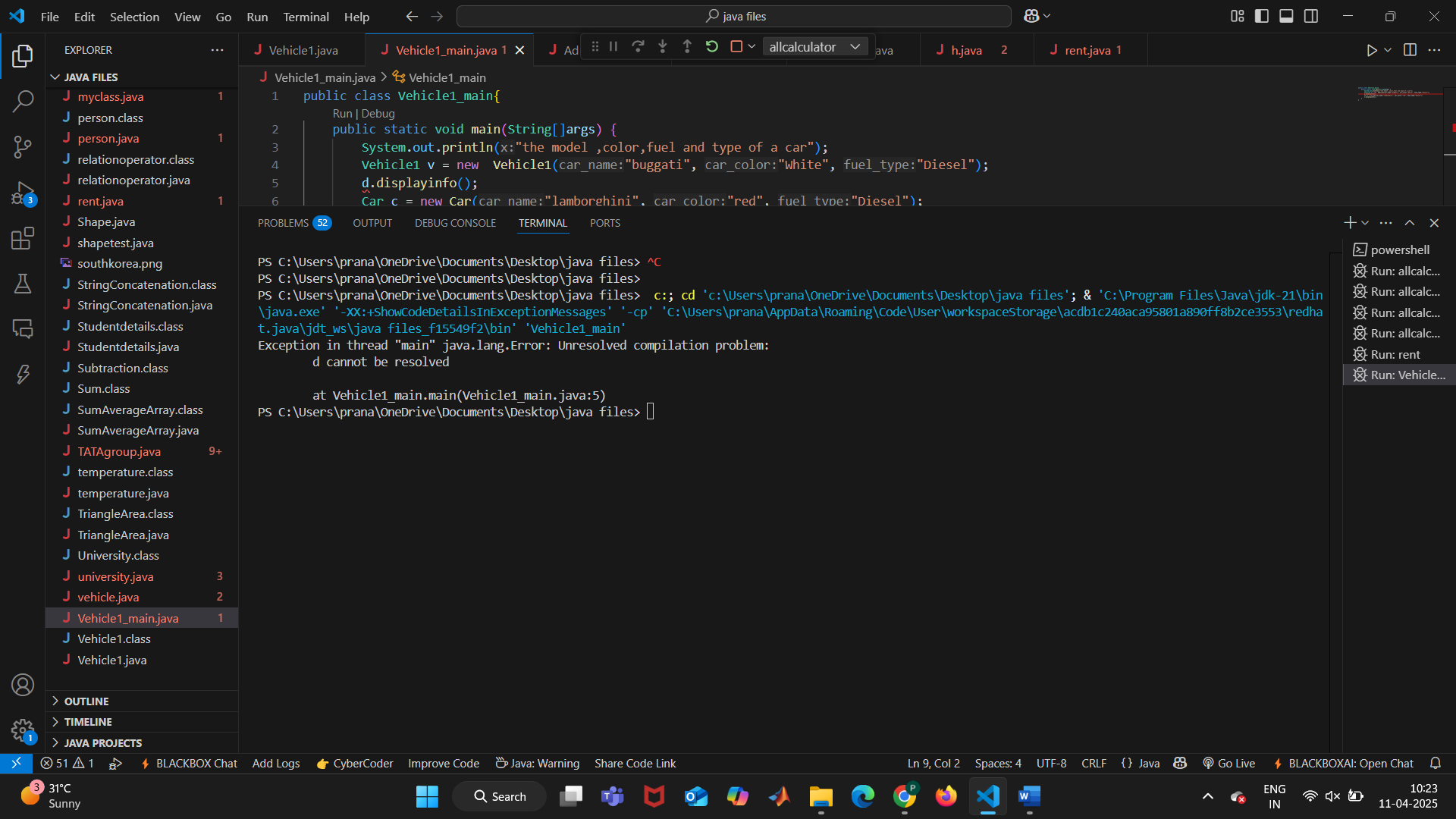
}

}}

**POSITVE CASE OUTPUT:**



**NEGATIVE CASE OUTPUT:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| |  | | --- | | **1.Method calls in main are missing an object reference** | | **2.Circle class method does not override theparent class method properly.** | | |  | | --- | | **1.Use s.calculateArea(4) and c.calculateArea(2) to call the method correctly.** | | **2.Ensure @Override is used, and**  **the method signature should match correctly.** | |

**IMPORTANT POINTS:**

**1.Inheritance**: Circle class extends Shape, inheriting its methods.

**2.Method Overloading**: Shape has multiple calculateArea methods with different parameters.

**3.Method Overriding**: Circle overrides calculateArea from Shape to implement its own formula.

**4.Polymorphism**: The overridden method in Circle demonstrates runtime polymorphism.

**5.Proper Object Reference**: Methods should be called using an object (s.calculateArea(4), c.calculateArea(2)).

**PROGRAM-4:**

**AIM:** A college is developing an automated admissions systems that verifies students eligibility for undergraduate(UG) and postgraduate(PG) programs. Each program has different eligibility. Criteria based on the students percentage in their previous qualification.

1. UG admission require min of 60%
2. PG admission require min of 70%

**CLASS DIAGRAM:**

**+---------------------------+**

**| AdmissionSystem |**

**+---------------------------+**

**| - input: Scanner |**

**| - name: String |**

**|- percentage: double |**

**|- program: String |**

**+---------------------------+**

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

**|+ takeInput(): void |**

**| + checkEligibility(): void |**

**|+ closeScanner(): void |**

**+---------------------------+**

**CODE:**

import java.util.Scanner;

public class AdmissionSystem {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.println("Enter program type (UG/PG):");

String program = input.nextLine();

System.out.println("Enter your percentage:");

double percentage = input.nextDouble();

System.out.println("name:pranathi,rollno:39,CSE");

if (program.equalsIgnoreCase("UG")) {

if (percentage >= 60) {

System.out.println("Eligible for UG admission");

} else {

System.out.println("Not eligible for UG admission");

}

} else if (program.equalsIgnoreCase("PG")) {

if (percentage >= 70) {

System.out.println("Eligible for PG admission");

} else {

System.out.println("Not eligible for PG admission");

}

} else {

System.out.println("Invalid program type");

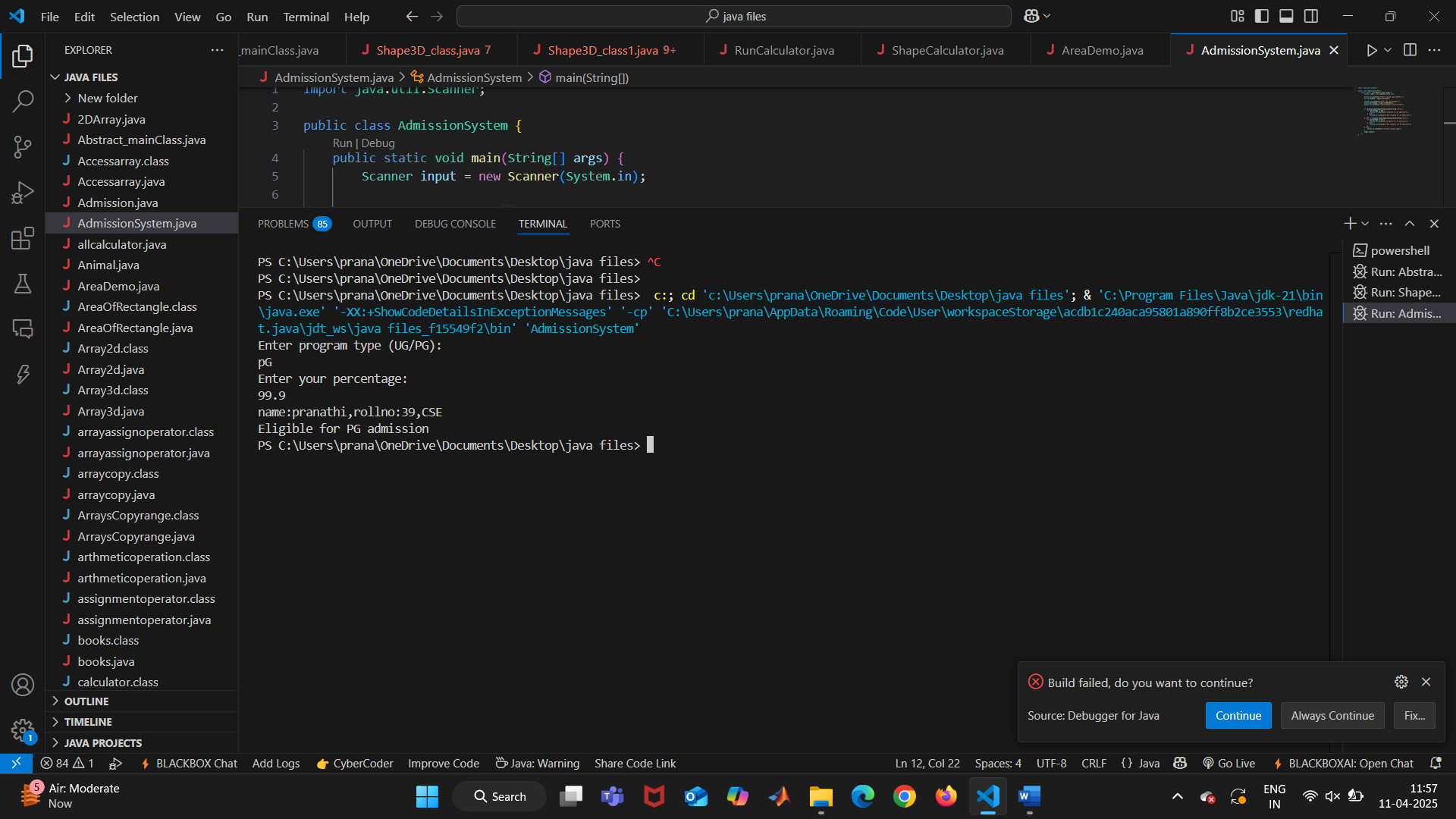
}

input.close();

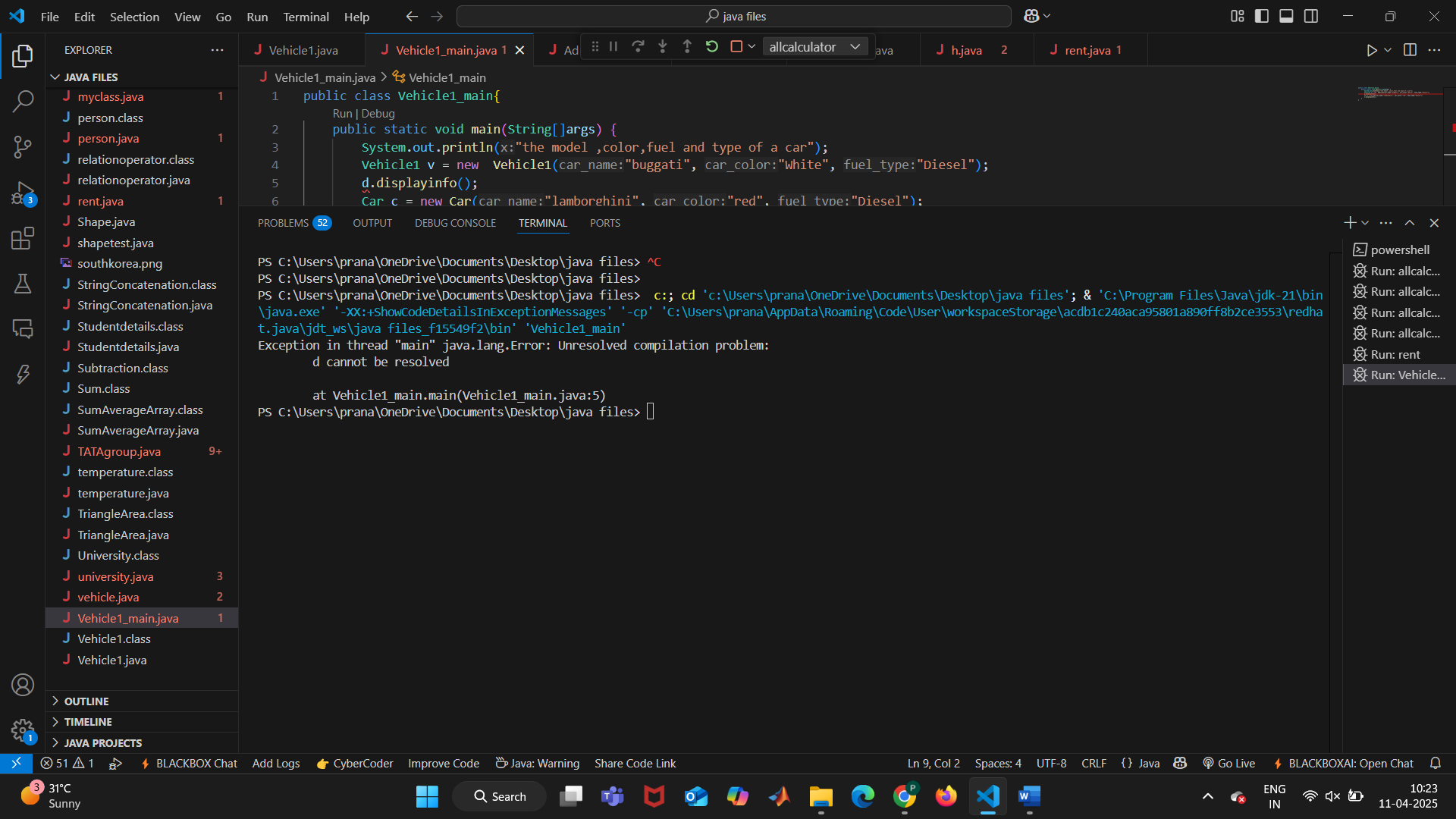
}

}

**POSITIVE CASE OUTPUT:**



**NEGATIVE CASE OUTPUT::**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| |  | | --- | | **1.Scanner nextLine() issue after nextDouble()**: After scanner.nextDouble(), the newline character remains in the buffer, causing nextLine() to be skipped. | | **2.Program type input case sensitivity issue**: If the user enters ug or pg in lowercase, it may cause incorrect comparisons. | | |  | | --- | | 1.Add scanner.nextLine(); after nextDouble(); to consume the leftover newline. | | 2.Use program.toUpperCase() to ensure case-insensitive comparison. | |

**IMPORTANT POINTS:**

**User Input Handling:** Uses Scanner to take user input for name, percentage, and program type.

Decision Making with Conditions**:** Uses if-else statements to check eligibility criteria.

**String Handling:** Converts program input to uppercase (toUpperCase()) to handle case variations.

**Closing Scanner:** Properly closes scanner using scanner.close(); to prevent resource leaks.

\

**WEEK-7**

**PROGRAM-1:**

**AIM:** write a java program to create an abstract class Animal with an abstract method called sound(). Create subclass Lion and Tiger that extend the Animal class and implent the sound() method to make a specific sound for each animal.

**CLASS DIAGRAM:**

**+------------------+**

**| <<abstract>> |**

**| Animal |**

**+------------------+**

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

**+------------------+**

**▲**

**|**

**+------------------------------------+**

**| |**

**+---------------+ +---------------+**

**| Tiger | | Lion |**

**+---------------+ +---------------+**

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

**+---------------+ +---------------+**

**CODE:**

abstract class Animal{

    abstract void sound();

}

class Tiger extends Animal{

    @Override

    public void sound(){

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

    }

}

class Lion extends Animal{

    @Override

    public void sound(){

        System.out.println("tiger roars");

    }

}

public class Abstract\_mainClass{

    public static void main(String[] args) {

        Animal animal = new Tiger();

        animal.sound();

        Animal animal2 = new Lion();

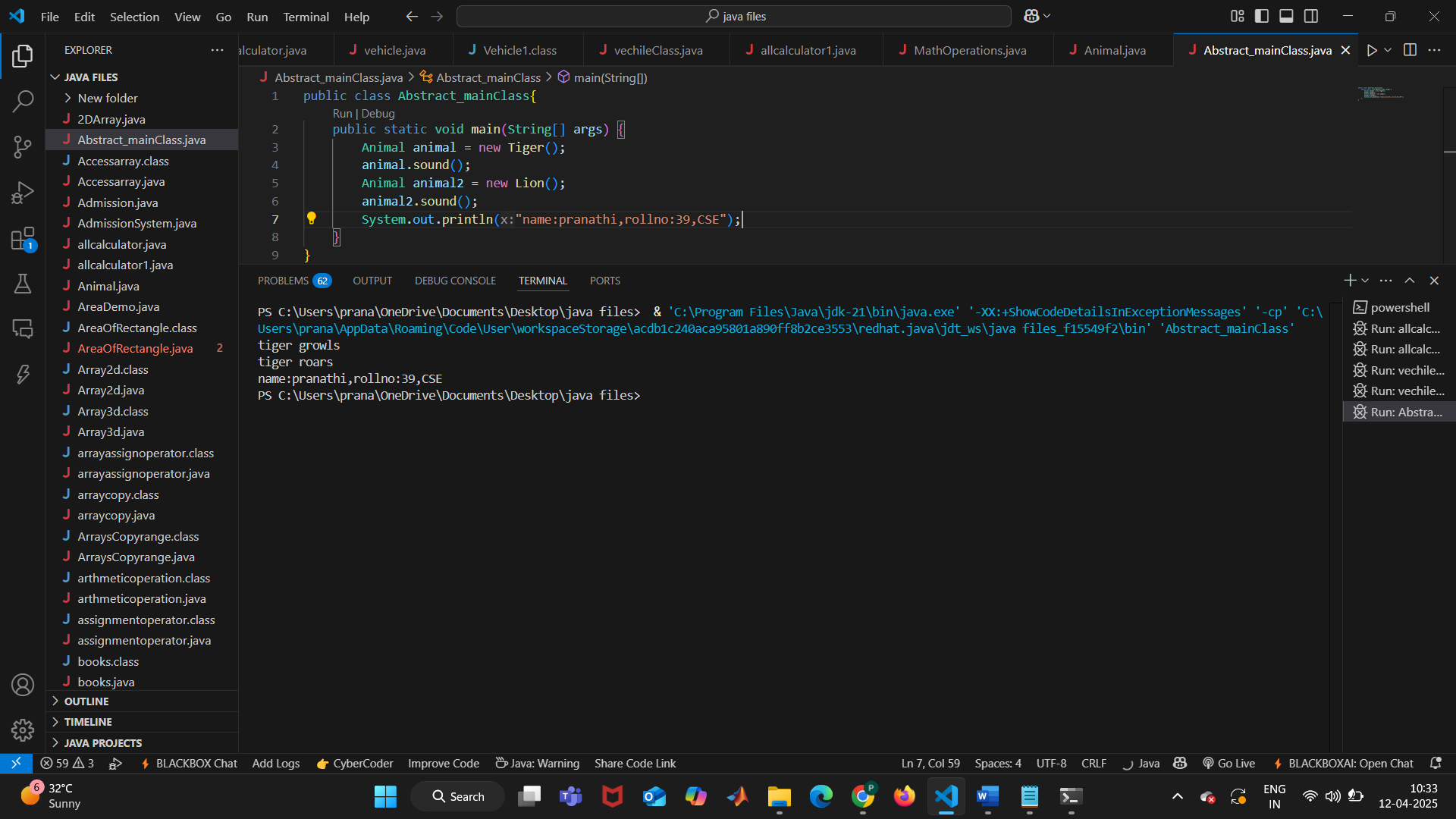
        animal2.sound();

System.out.println("name:pranathi,rollno:39,CSE");

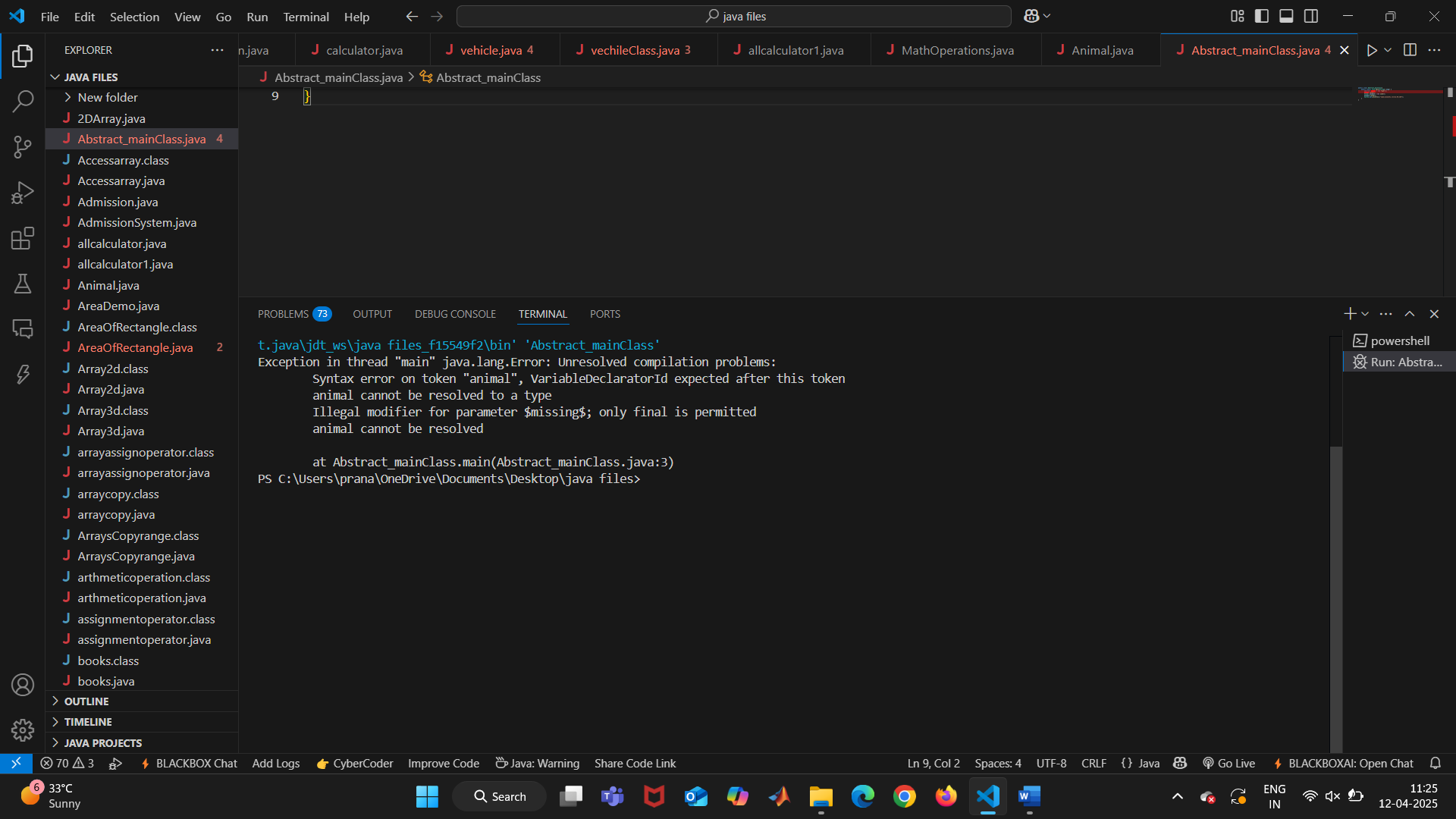
    }

}

**POSITIVE CASE OUTPUT:**



**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE:**

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

**IMPORTANT POINTS:**

Here we used the abstract to declare an abstract class.

1. 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-2:**

**AIM:** write a java program to create an abstract class shape3d with abstract class Shape3D with abstract methods calculate volume() and calculateSurfaceArea(). Create subclasses sphere and cube that extend the shape3D class and implemt the respective methods to calculate the volume and surface area of each shape.

**CLASS DIAGRAM:**

**CODE:**

**POSITIVE CASE OUTPUT:**

**NEGATIVE CASE OUTPUT:**

**ERROR TALE:**

**IMPORTANT POINTS:**

**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 ith an abstract method print pattern (int n)and concrete method to display the pattern title.
* Implement 2 subclasses :

1. Star pattern-prints a right -angled triangle stars(\*).
2. Number pattern-prints a right-angled triangle of increasing numbers.

* In the main() method, create objects of both subclasses and print the pattern for a given number of rows.

**CLASS DIAGRAM:**

**CODE:**

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 pattern\_class {

    public static void main(String[] args) {

    System.out.println("Name:P.Pranathi,Section:,CSE-CRoll NO:AV.SC.U4CSE24239");

        int numberOfRows = 5;

        PatternPrinter star = new StarPattern(numberOfRows);

        star.displayTitle("Star Pattern");

        star.printPattern();

        PatternPrinter number = new NumberPattern(numberOfRows);

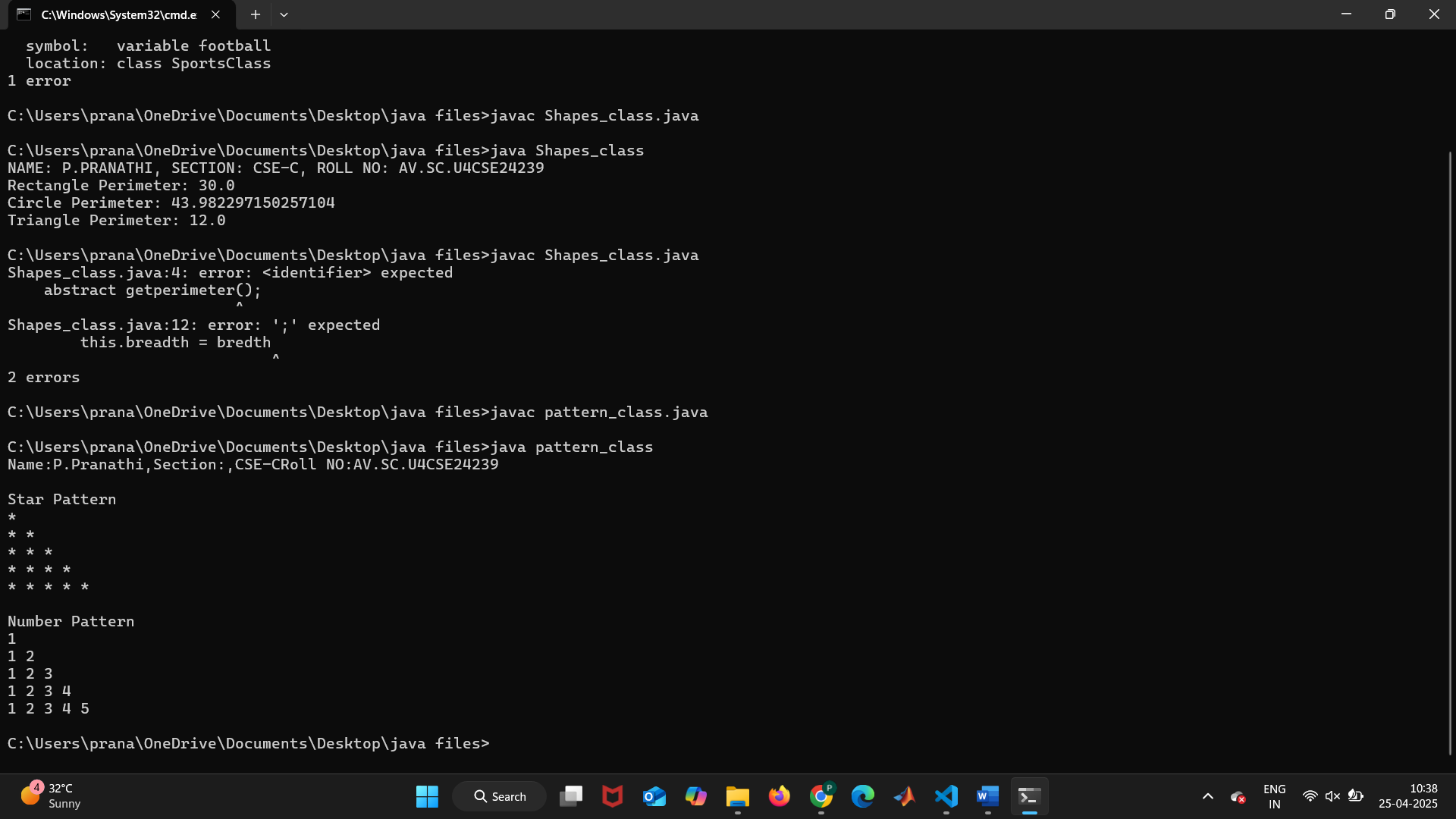
        number.displayTitle("Number Pattern");

        number.printPattern();

    }

}

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

**ERROR TABLE:**

**IMPORTANT POINTS:**

**WEEK-8**

**PROGRAM-1:**

**AIM:** Write a java program to create an interface shape with the get perimeter() method. Create three classes Rectangle,Circle andTriangle that implement the shape interface. Implent the get perimeter() method for each of the three classes.

**CLASS DIAGRAM:**

+------------------+

| <<interface>>|

| shape |

+-------------------+

| +getperimeter(): double |

+--------------------+

▲

|

-------------------------------------

| | |

+-------------+ +-------------+ +-------------+

| rectangle | | circle | | triangle |

+-------------+ +-------------+ +-------------+

| -length:int | | -radius:double | | -side1:int |

| -breadth:int | | -side2:int |

+-------------+ +-------------+ | -side3:int |

| +getperimeter(): double | +-------------+

+-----------------------------+ | +getperimeter(): double |

+--------------------------+

**CODE:**

import java.util.\*;

interface shape {

    abstract double getperimeter();

}

class rectangle implements shape {

    int length, breadth;

    rectangle(int len, int bredth) {

        this.length = len;

        this.breadth = bredth;

    }

    public double getperimeter() {

        return 2 \* (length + breadth);

    }

}

class circle implements shape {

    double radius;

    circle(double r) {

        this.radius = r;

    }

    public double getperimeter() {

        return 2 \* Math.PI \* radius;

    }

}

class triangle implements shape {

    int side1, side2, side3;

    triangle(int a, int b, int c) {

        this.side1 = a;

        this.side2 = b;

        this.side3 = c;

    }

    public double getperimeter() {

        return side1 + side2 + side3;

    }}

class Shapes\_class {

    public static void main(String args[]) {

        System.out.println("NAME: P.PRANATHI, SECTION: CSE-C, ROLL NO: AV.SC.U4CSE24239");

        rectangle r = new rectangle(10, 5);

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

        circle c = new circle(7);

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

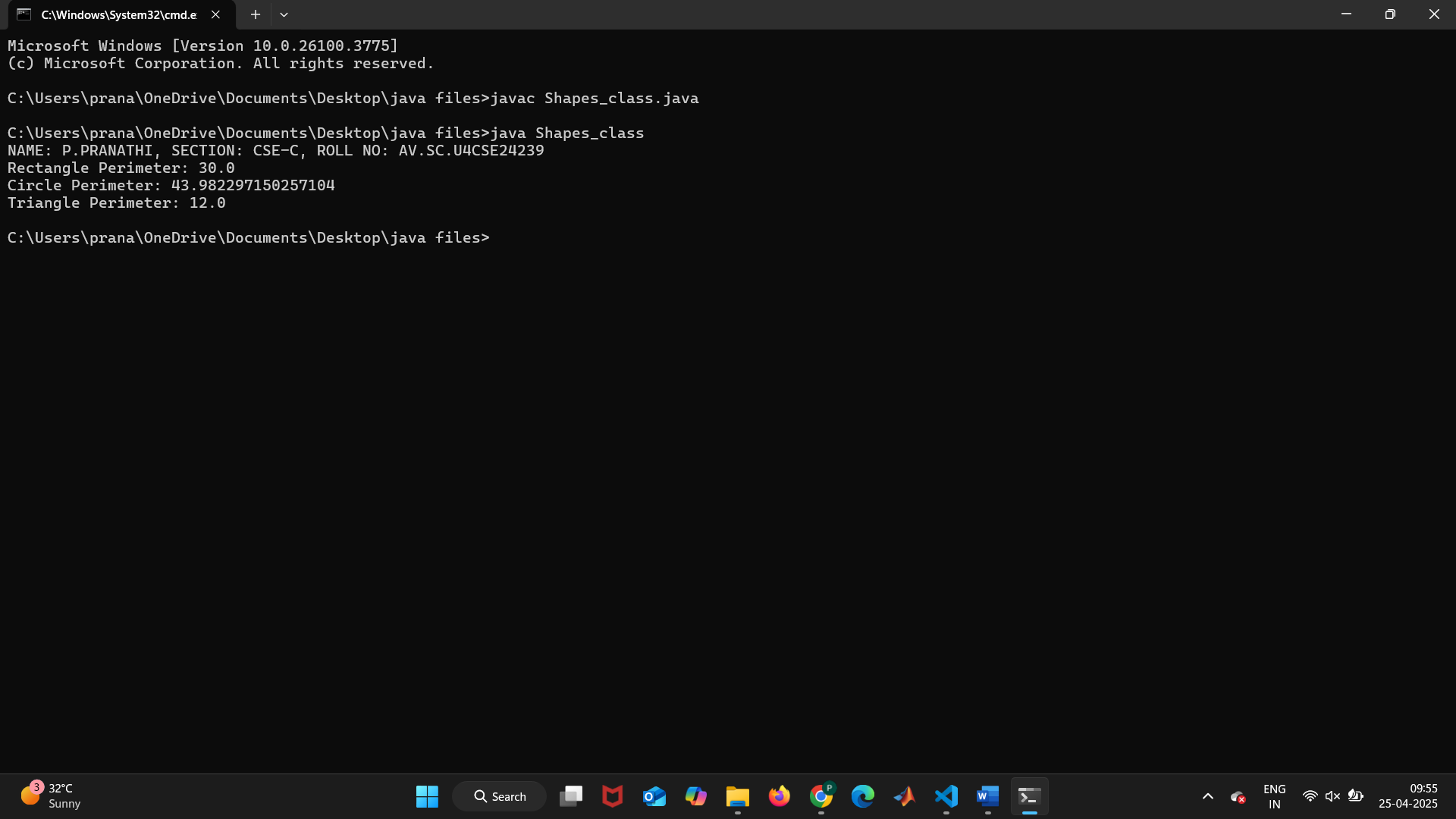
        triangle t = new triangle(3, 4, 5);

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

    }

}

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

**ERROR TABLE:**

|  |  |
| --- | --- |
| **CODE ERROR:**  1) Class names like rectangle, circle, and triangle start with lowercase letters.  2) Unused import: import java.util.\*; is included but not used.  3) Interface method has abstract keyword (optional and redundant in modern Java). | **ERROR RECTIFICATION**  1) Rename them to Rectangle, Circle, and Triangle to follow standard Java class naming conventions.  2) Remove import java.util.\*; as there’s no usage of any class from java.util in the code.  3) Remove abstract from double getperimeter(); inside the interface, as it's implicit for interface methods. |

**IMPORTANT POINTS:**

1. The Shape interface defines the getperimeter() method, ensuring that every shape class provides its own implementation for calculating the perimeter, promoting polymorphism.
2. Each shape class (Rectangle, Circle, Triangle) implements Shape and encapsulates its unique logic for perimeter calculation, hiding internal details from the user and demonstrating encapsulation.
3. The code is modular and scalable, allowing easy addition of new shape types in the future by simply implementing the Shape interface, which follows the Open/Closed Principle.
4. Polymorphism is showcased as the same method (getperimeter()) behaves differently based on the object type (e.g., Rectangle, Circle, or Triangle), while the main() method handles the objects in a uniform way.
5. Adherence to Java conventions: Proper naming (e.g., capitalized class names) and the use of @Override annotations enhance code readability, maintainability, and correctness.

**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 implements the palyable interface and override the player method to play the respective sports.

**CLASS DIAGRAM:**

+-----------------+

| Playable |

+-----------------+

| + play(): void |

+-----------------+

^

|

+----------------------+---------------------+

| | |

+-----------+ +--------------+ +--------------+

| Football | | Volleyball | | Basketball |

+-----------+ +--------------+ +--------------+

| + play(): void | | + play(): void | | + play(): void |

+-----------+ +--------------+ +--------------+

**CODE:**

interface Playable {

    void play();

}

class Football implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Football! Kick the ball and score goals.");

    }}

class Volleyball implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Volleyball! Hit the ball over the net.");

    }}

class Basketball implements Playable {

    @Override

    public void play() {

        System.out.println("Playing Basketball! Dribble, pass, and shoot the ball.");

    }}

public class SportsClass {

    public static void main(String[] args) {

        System.out.println("NAME: P.PRANATHI, SECTION: CSE-C, ROLL NO: AV.SC.U4CSE24239");

        Playable football = new Football();

        Playable volleyball = new Volleyball();

        Playable basketball = new Basketball();

        football.play();

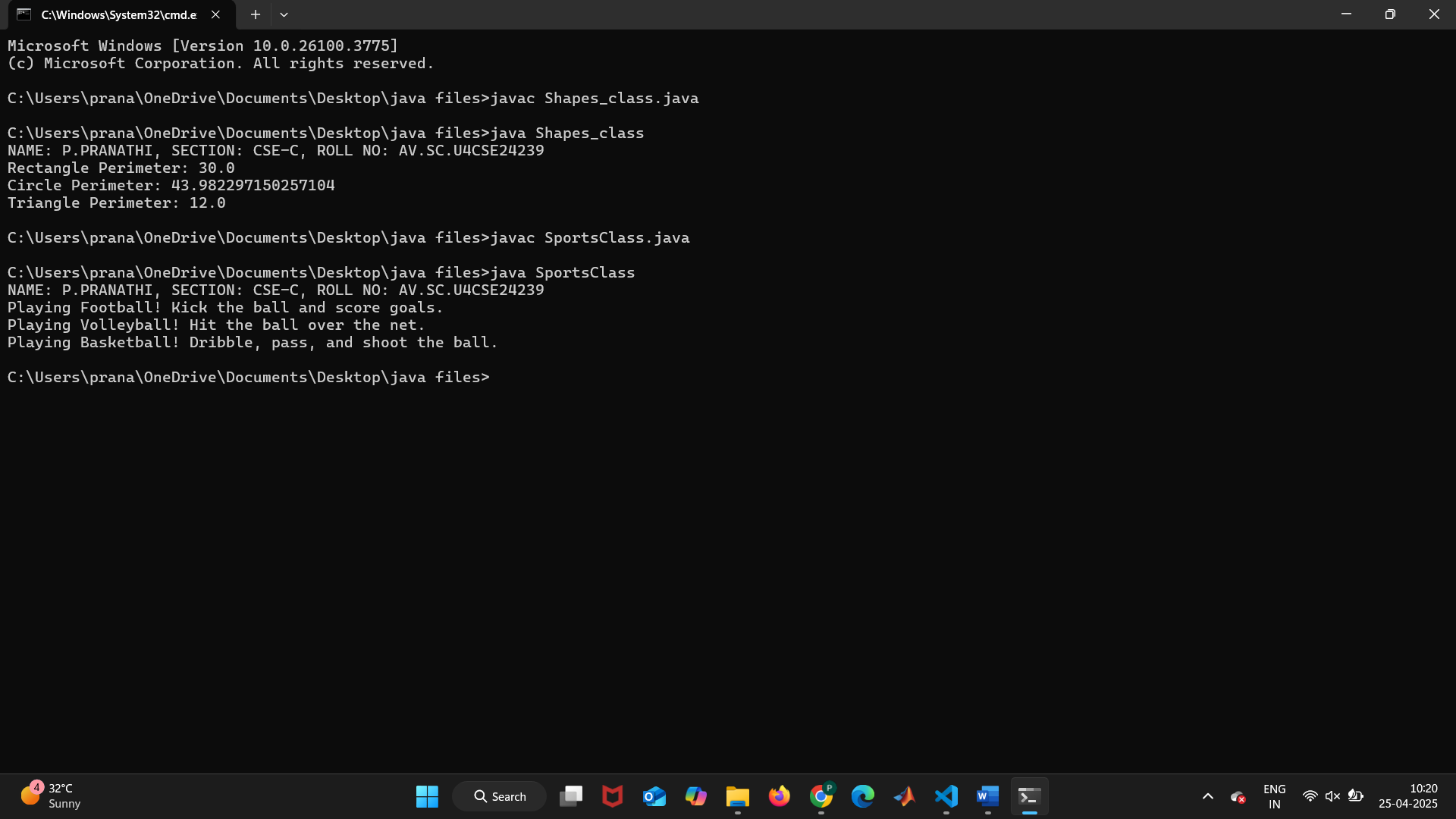
        volleyball.play();

        basketball.play();

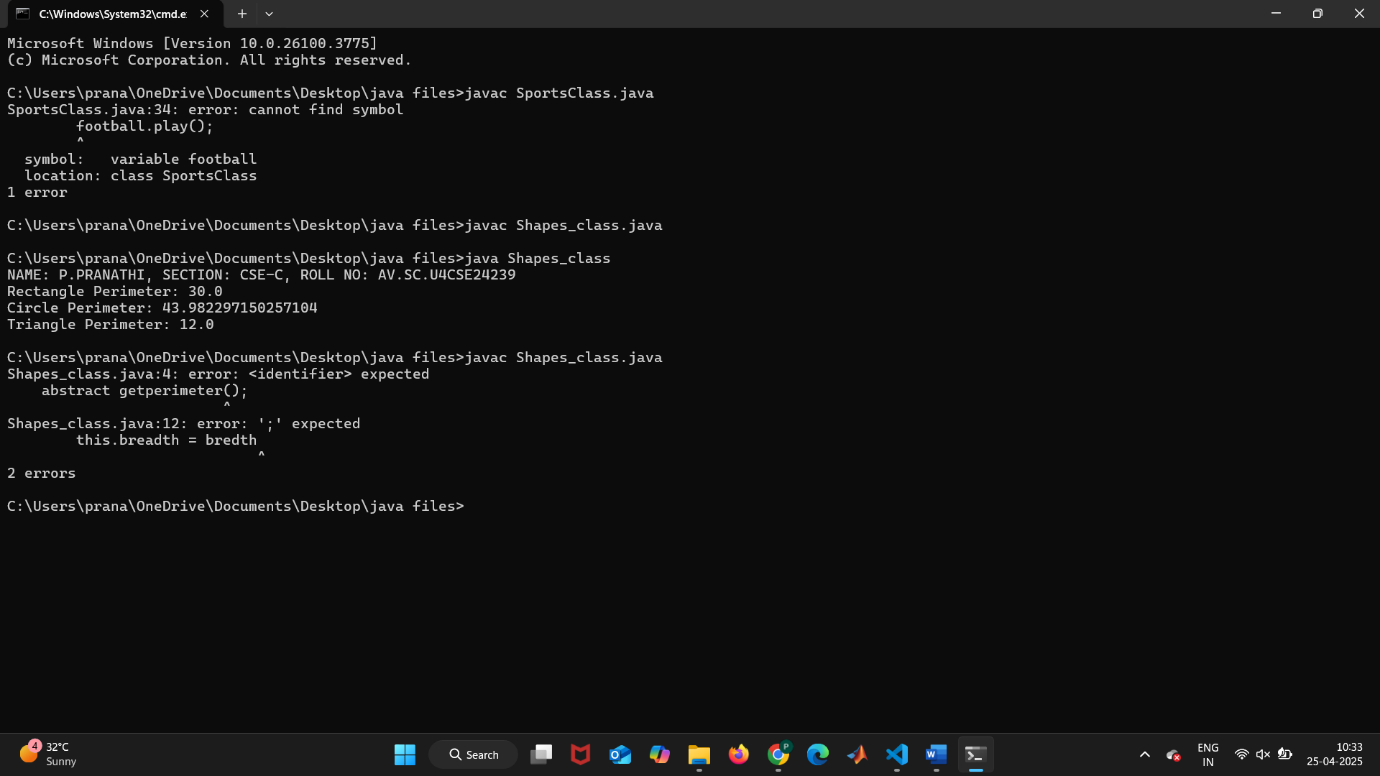
    }

}

**POSITIVE CASE OUTPUT:**

****

**NEGATIVE CASE OUTPUT:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | **Code rectification** |
| 1. Declaring an abstract class instead of interface class. 2. The play() method in the Playable interface and the implementing classes may not have the same visibility modifier. | 1. Declare an interface class instead of abstract class. 2. Ensure the play() method is public in both the Playable interface and the implementing classes. |

**IMPORTANT POINTS:**

1. The Playable interface enforces the implementation of the play() method in each class, ensuring different behaviors.
2. The play() method behaves differently based on the object type (Football, Volleyball, Basketball), demonstrating polymorphism.
3. Each sport class hides its specific behavior (how the sport is played), offering a clean interface.
4. SportsClass interacts only with Playable, making it independent of specific sport classes, allowing easy extension.
5. Each class has a single responsibility, making the code easier to maintain and modify.
6. New sport classes can implement Playable and be used interchangeably, promoting code reusability.