**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**LAB REPORT**



**Department of Computer Science Engineering**

**Amrita School of Computing**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Name:M.SAI MOKSHITHA**

**Verified By Roll No: 24151**

**WEEK 01**

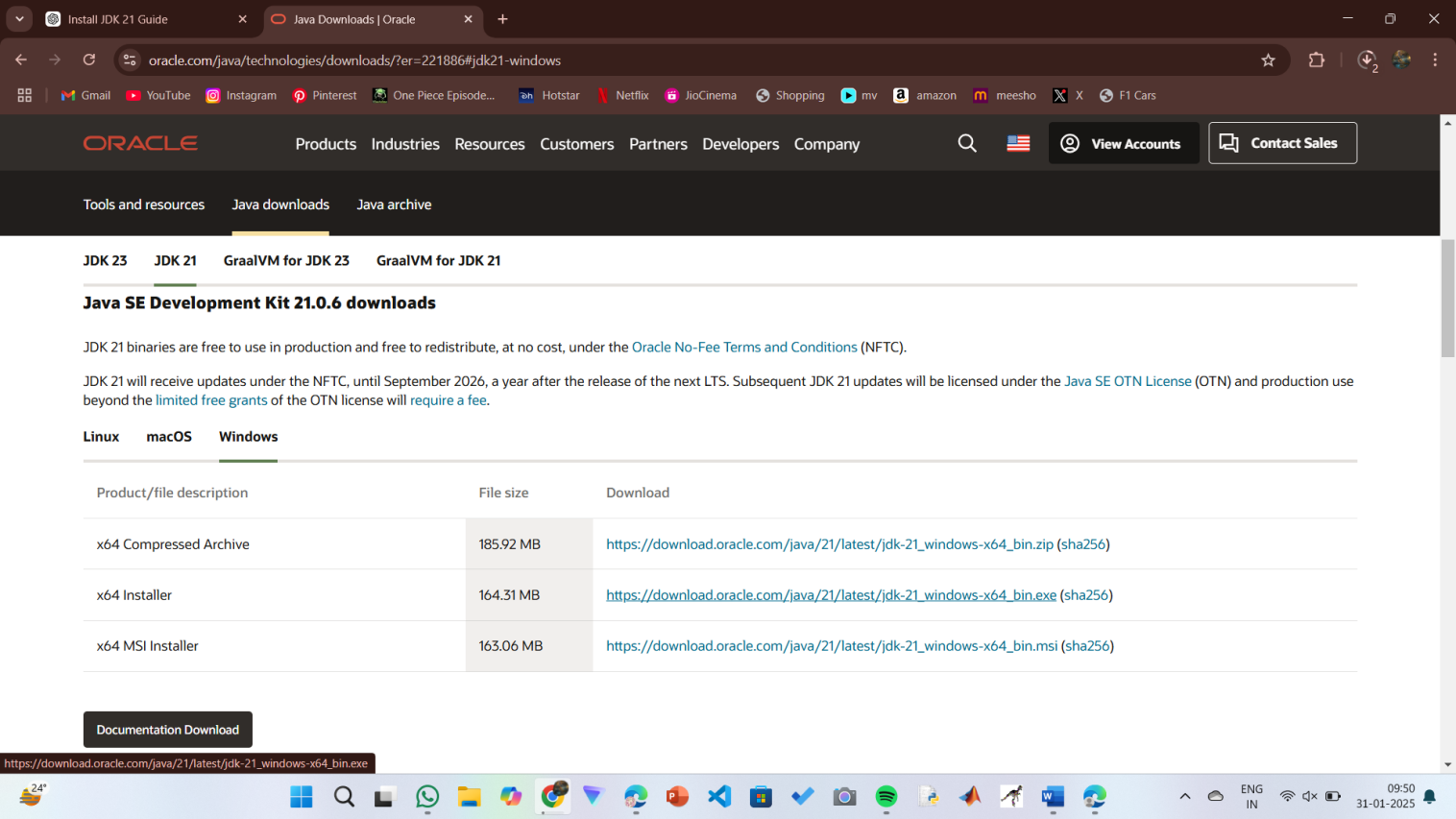
**PROGRAM-1:**

**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.

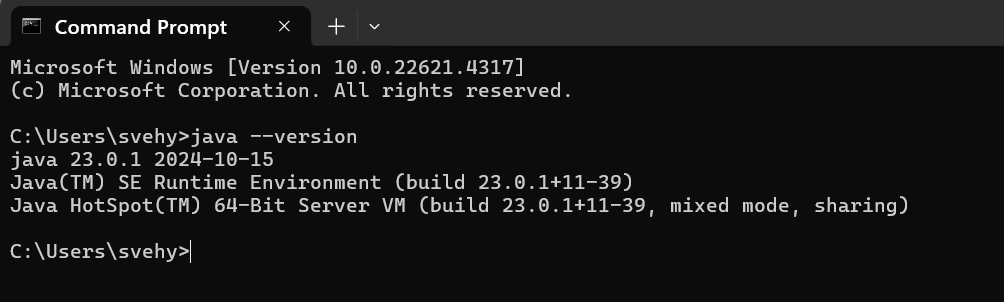
2)Click New and add: C:\Program Files\Java\jdk-21\bin

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.**



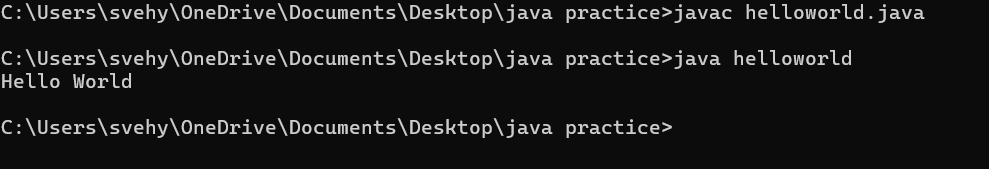
**PROGRAM-2:**

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

**CODE:**



**Output:**



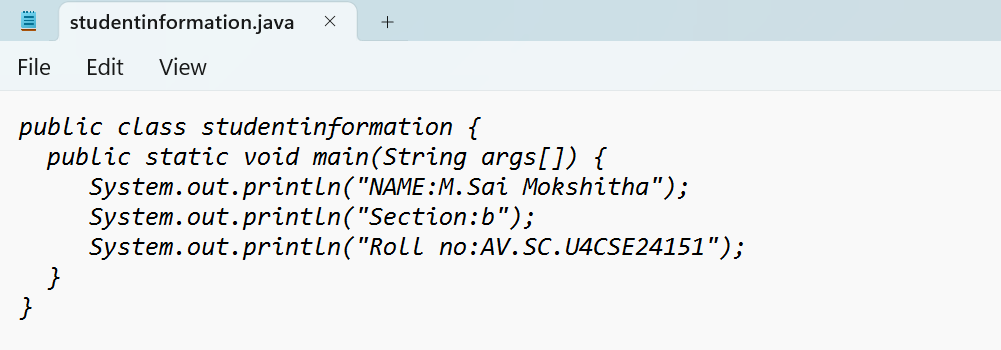
**ERRORS:** None found

**PROGRAM-3:**

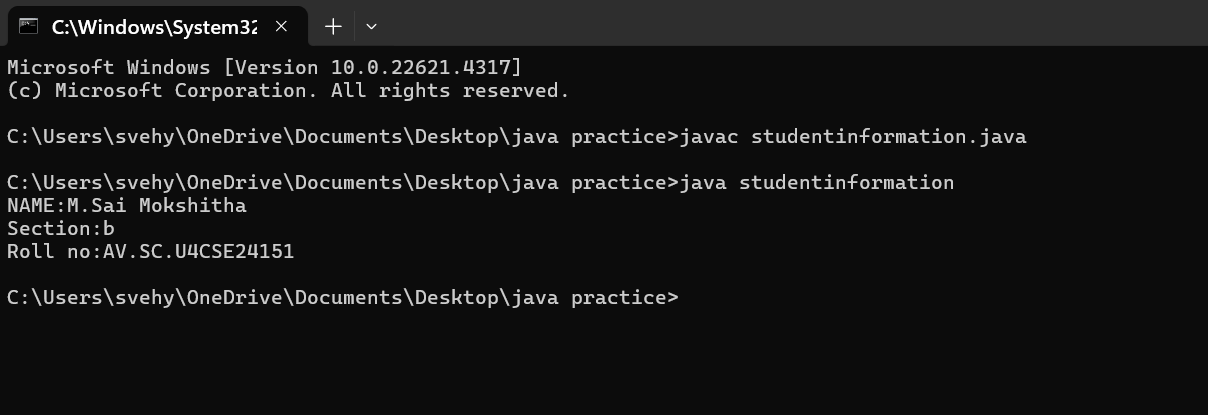
**AIM:** Write a Java Program that prints Name, Roll No, Section of a student.

**CODE:**

**STUDENT DETAILS:**



**OUTPUT:**



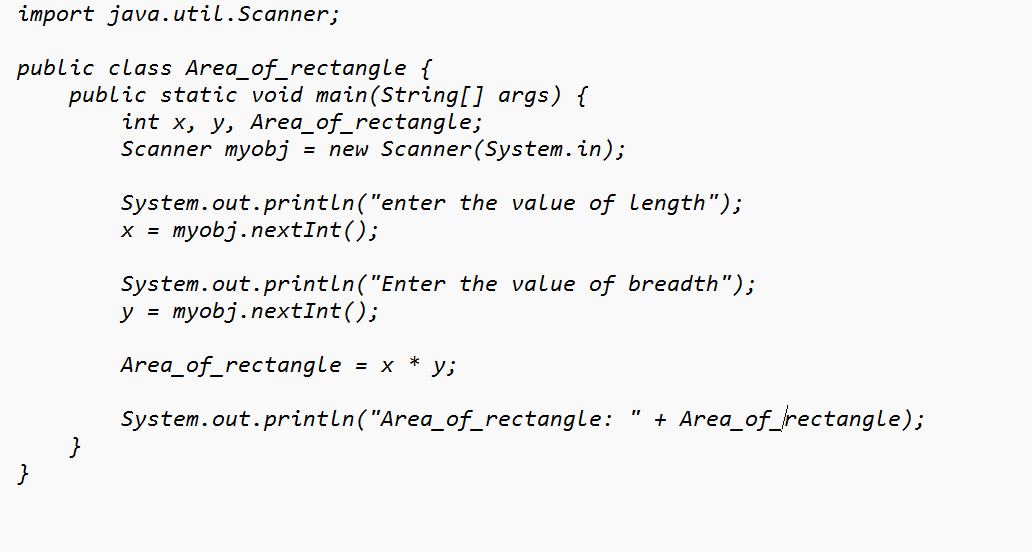
ERROR TABLE:

|  |  |
| --- | --- |
| **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 | 1. code is rectified by keeping capital “S” 2. Giving strings to name and section |

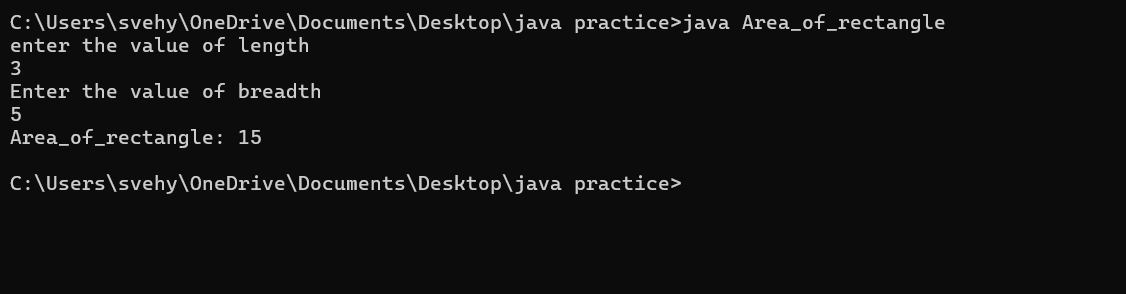
**PROGRAM-4:**

**AIM:** Calculate area of rectangle.

**CODE:**



**OUTPUT:**



ERROR TABLE:

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. While using for iteration, not giving the conditions correctly. 2. Declaring the data type as double instead of int. | 1. We should give iterative statements correctly. 2. We should give the data type as int for integers. |

IMPORTANT POINTS:

1. Area of a rectangle is area = l\*b, where

L = length of a side of the rectangle,

B= breadth of a side of the rectangle.

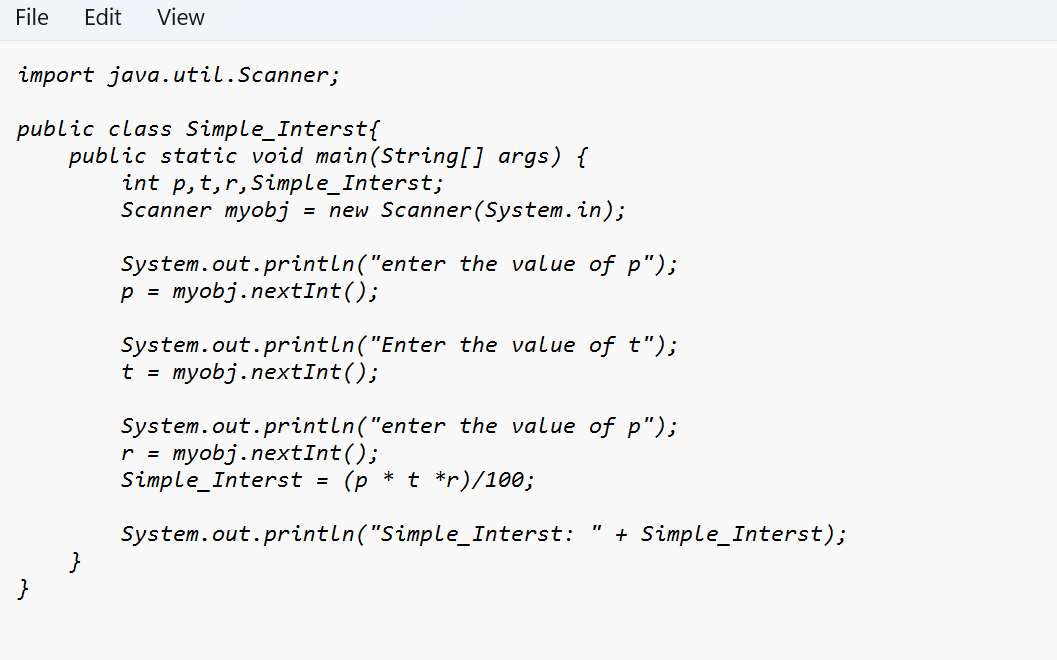
1. Here, we must be sure that all the expressions/conditions inside for

the for loop must be given correctly.

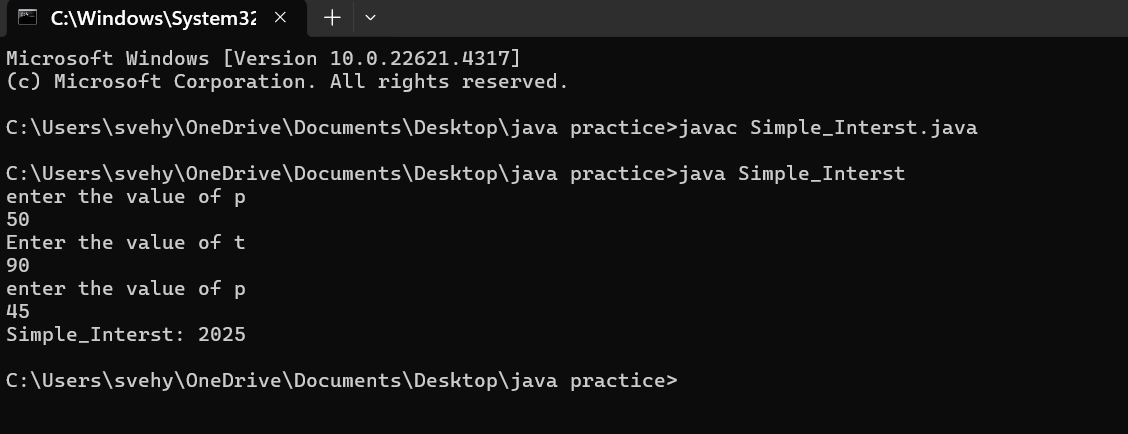
**PROGRAM-6:**

**AIM:**Calculate the simple interest

**CODE:**



**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 | 1. code is rectified by keeping capital “S” 2. Giving strings to name and section |

IMPORTANT POINTS:

1. Simple interest formula is: (p\*t\*r)/100, where:

P: Principal amount

R: Rate of interest

T: Time period

1. The data type double indicates the floating points in the integers.
2. The line “import java.util.Scanner” indicates:

Import: tells the java compiler that we want to use a specific class or package in your code.

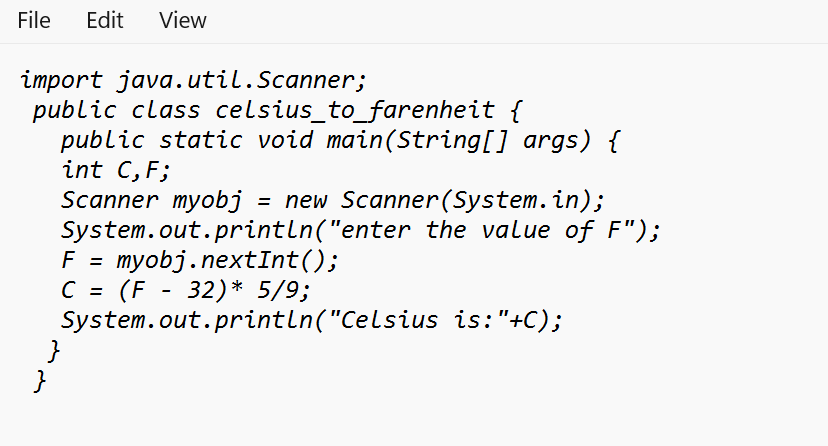
Java.util : This is the package that contains utility classes for Java programming, including the “Scanner” class.

Scanner: this is the class that allows you to read input from the keyboard.

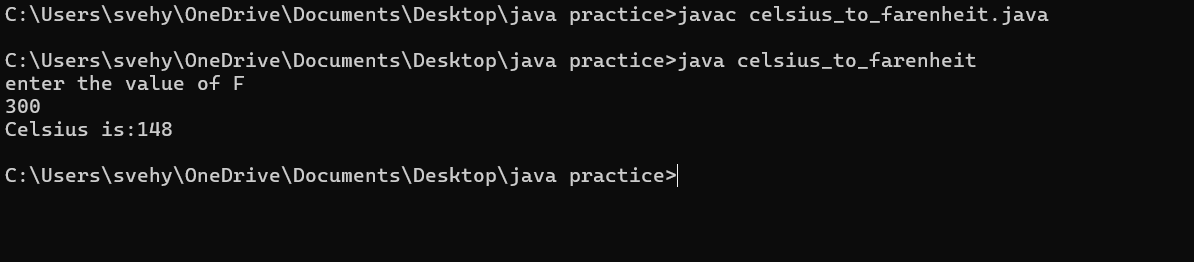
**PROGRAM-5:**

**AIM:**Convert temperature celsius into fahrenheit

**CODE:**



**OUTPUT:**



ERROR TABLE:

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. While printing the variable not giving + sign. 2. Not closing the scanner. | 1. We should give correct indentation. 2. Closing the scanner is must. |

IMPORTANT POINTS:

1. The formula to convert a Fahrenheit to Celsius is

Celsius = (Fahrenheit-32)\*5/9

1. The formula to convert a Celsius to Fahrenheit is

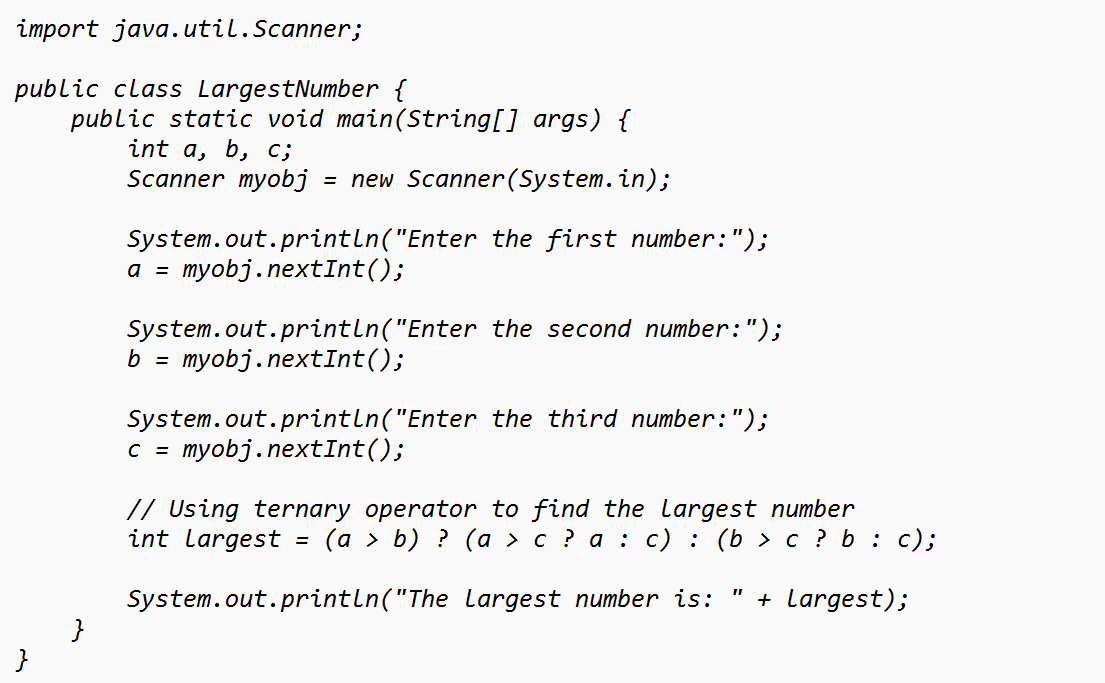
Fahrenheit = (Celsius\*9/5)+32.

1. 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.

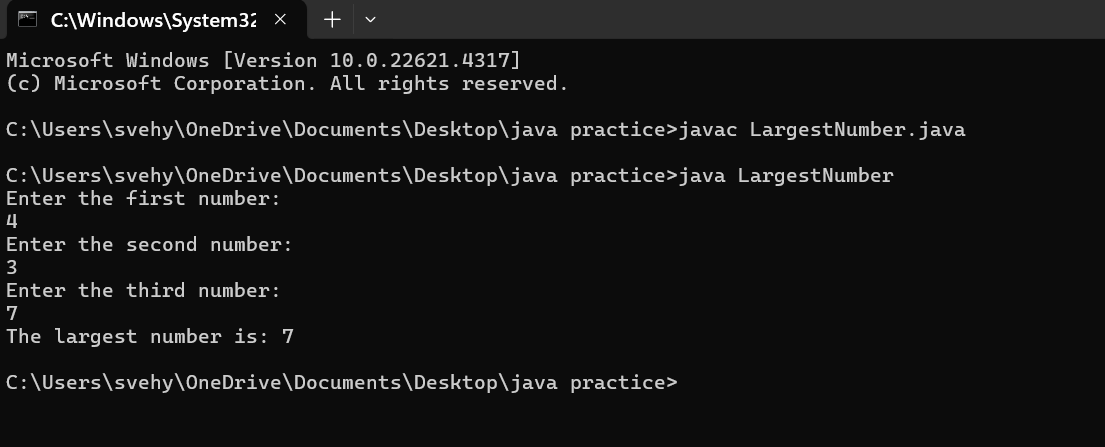
**PROGRAM-8:**

**AIM:**Largest of 3 numbers by ternary operators

**CODE:**



**OUTPUT:**



ERROR TABLE:

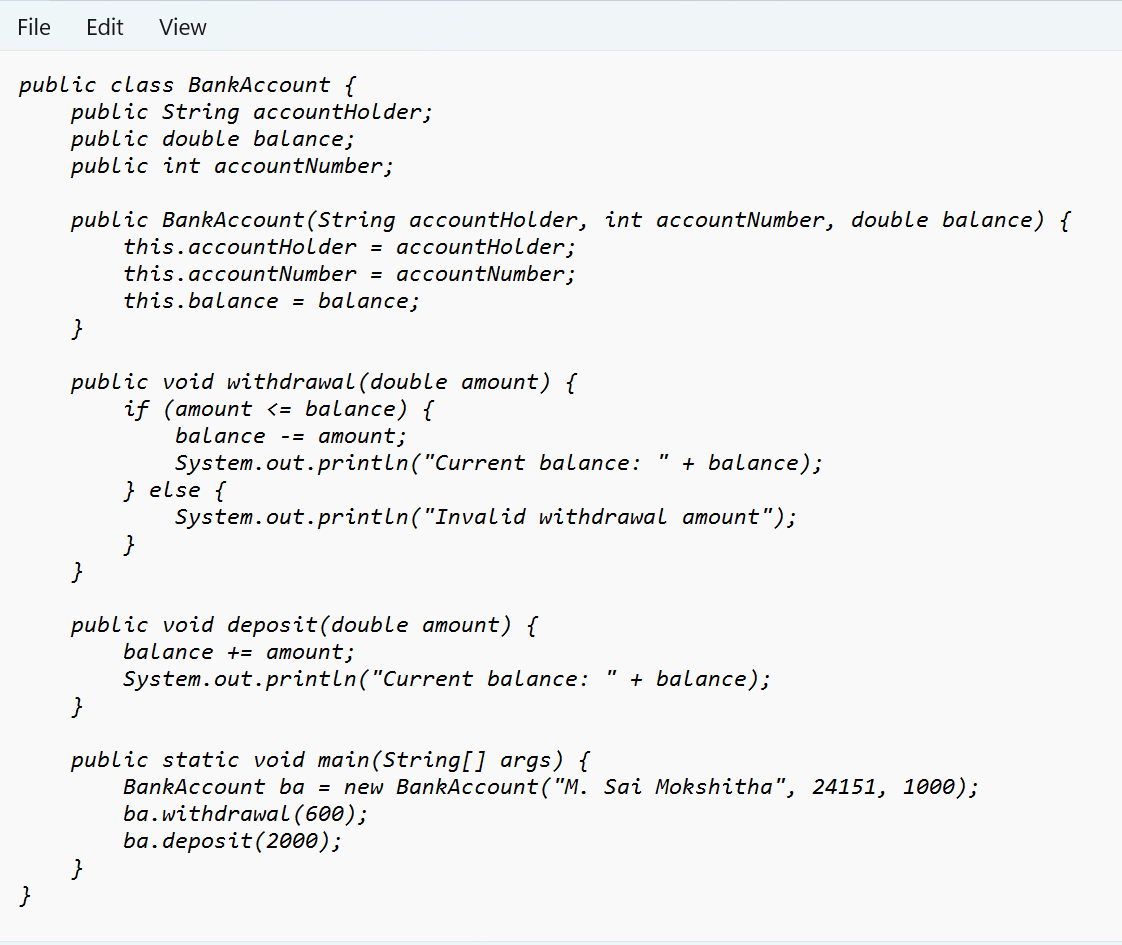
|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. writing small “S” in place of ”S”   In system.out.println()  2)not giving strings to the name and section | 1. code is rectified by keeping capital “S” 2. Giving strings to name and section |

WEEK – 3:

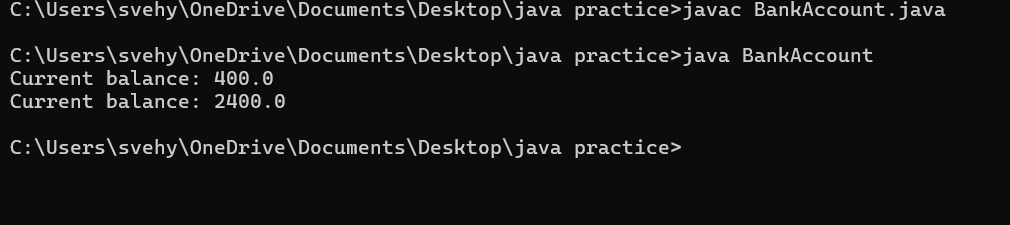
PROGRAM- 1:

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 TABLE:

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. Not putting the semi-colon; after calling the function. 2. After Withdrawal,deposit 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.

**Class diagram:**

|  |
| --- |
| **BankAccount**  **----------------------------------------------------------**  **-balance: double**  **----------------------------------------------------------**  **+BankAccount(intialBalance: double)**  **+deposit(amount: double):void**  **+withdraw(amount: double):void** |

PROGRAM- 2:

AIM: To create java program with following instructions :

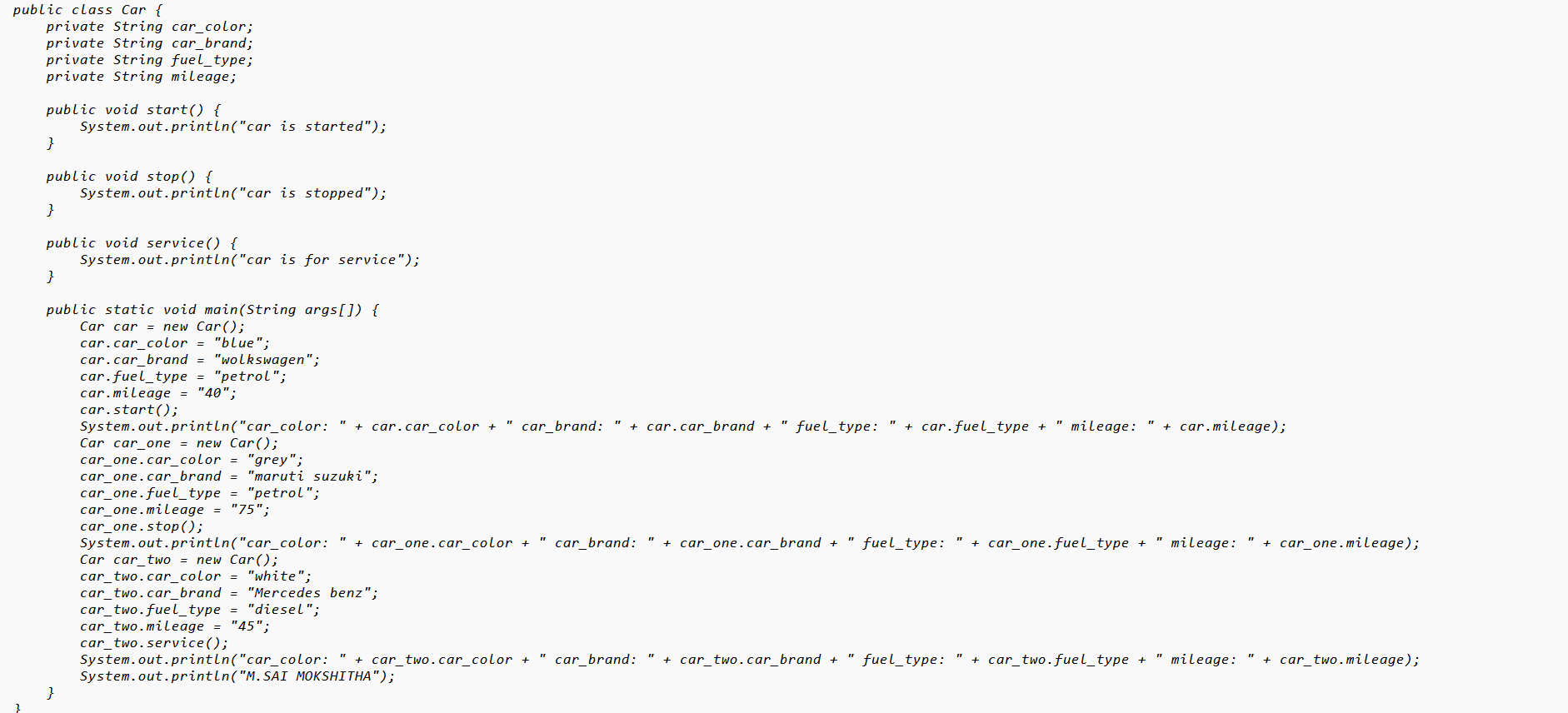
1. Create a class with name Car

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

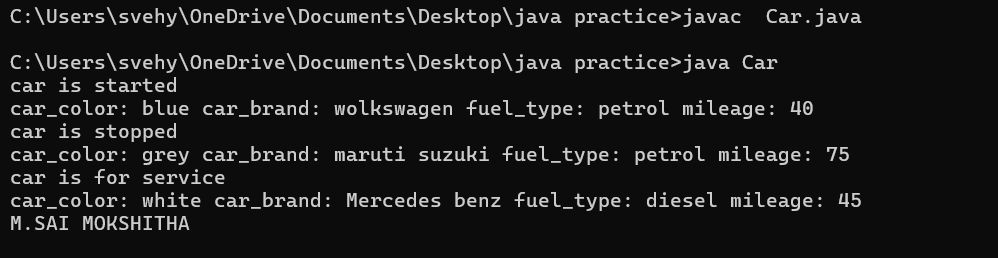
3. Create these methods named start(),stop(),service()

4. Create the objects named car, car1,car2.

CODE:



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. 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**  **-milage:double**  **----------------------**  **+start():void**  **+stop():void**  **+service():void** |

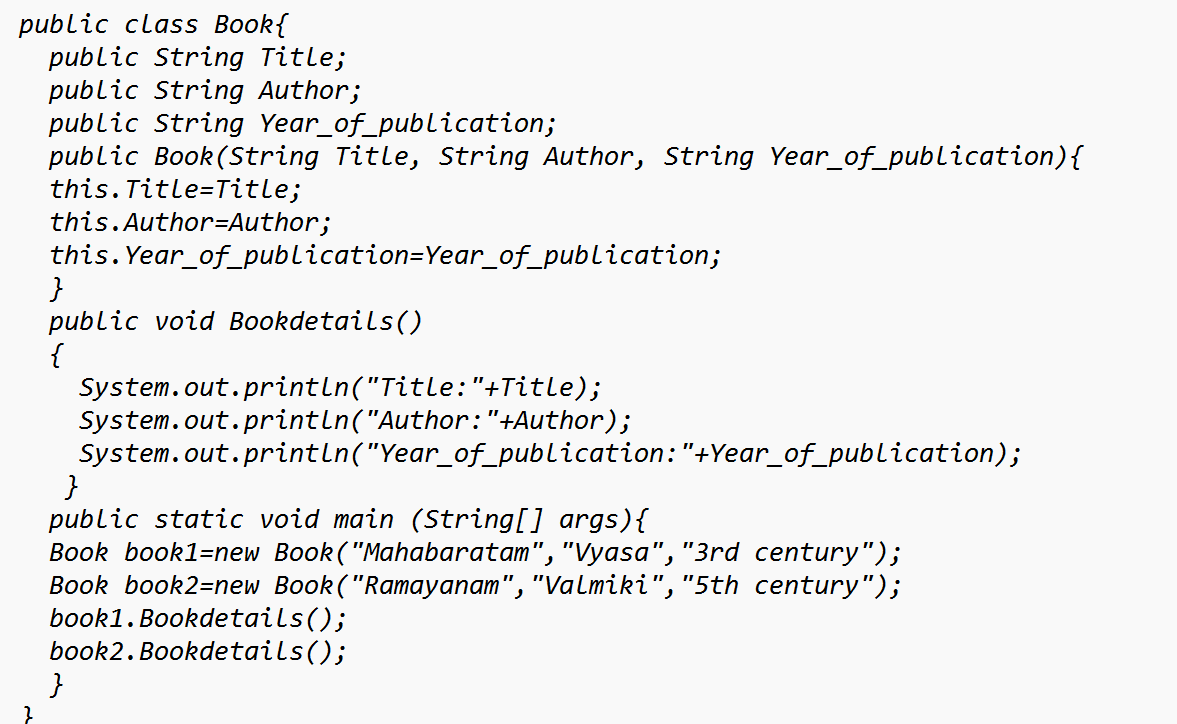
WEEK -4

PROGRAM – 1:

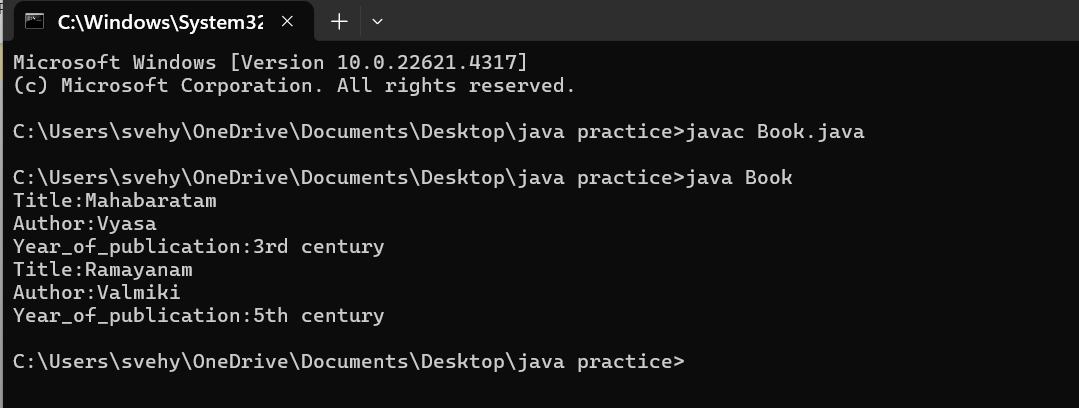
AIM: Write a java program with class named “book”, the class should contain various attributes such as title, author, year of publication it should also contain a constructor with parameters which initializes, title, author, and year of publication.

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

CODE:



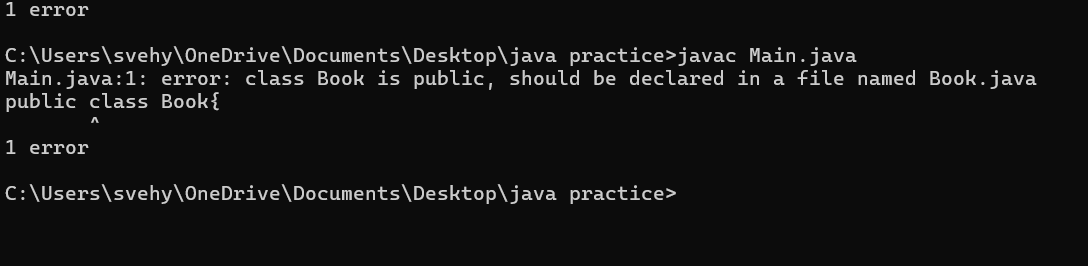
OUTPUT:



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. |

NEGATIVE CASE:



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.

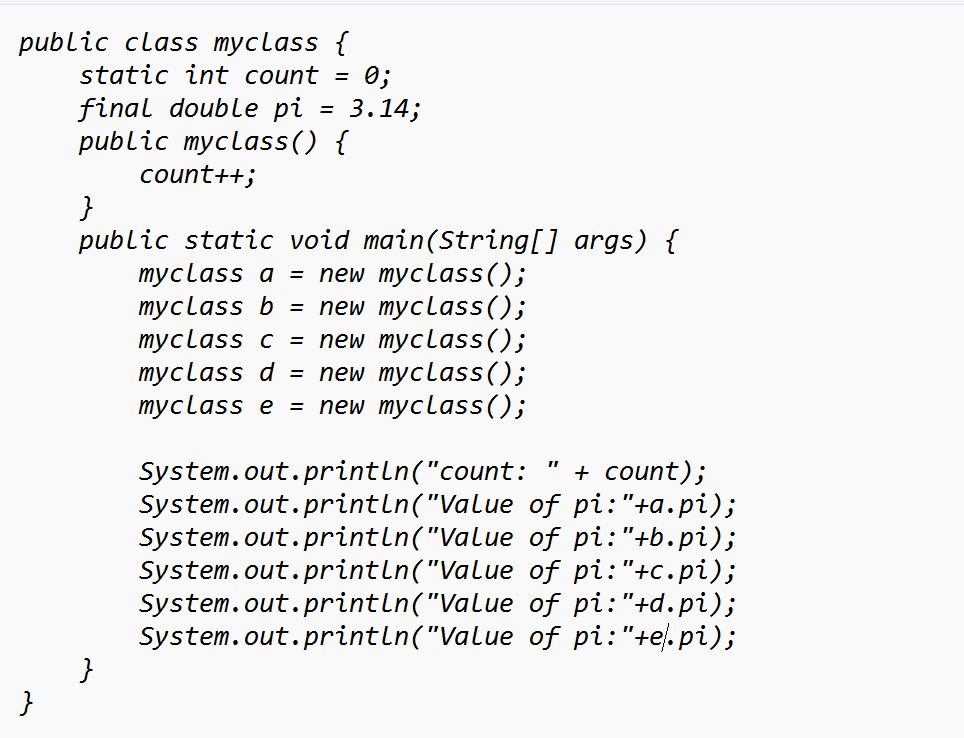
CLASS DIAGRAM:

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

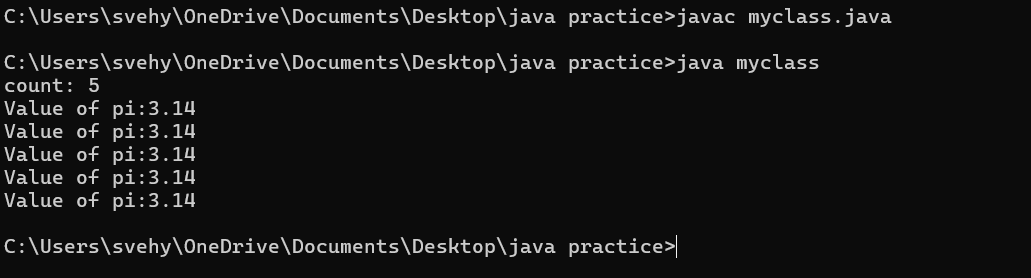
PROGRAM – 2:

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.

CODE:



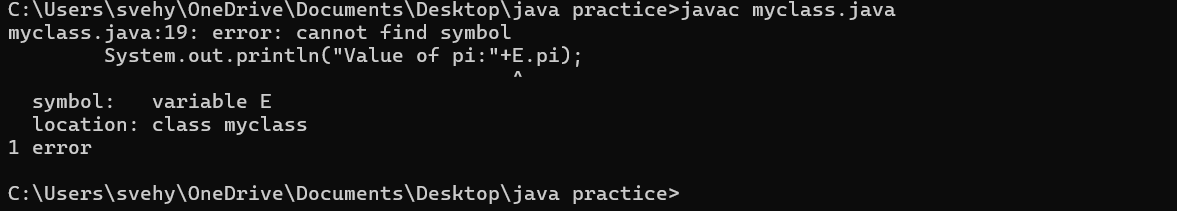
OUTPUT:



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. |

NEGATIVE CASE:



IMPORTANT POINTS:

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.

CLASS DIAGRAM:

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

WEEK-5

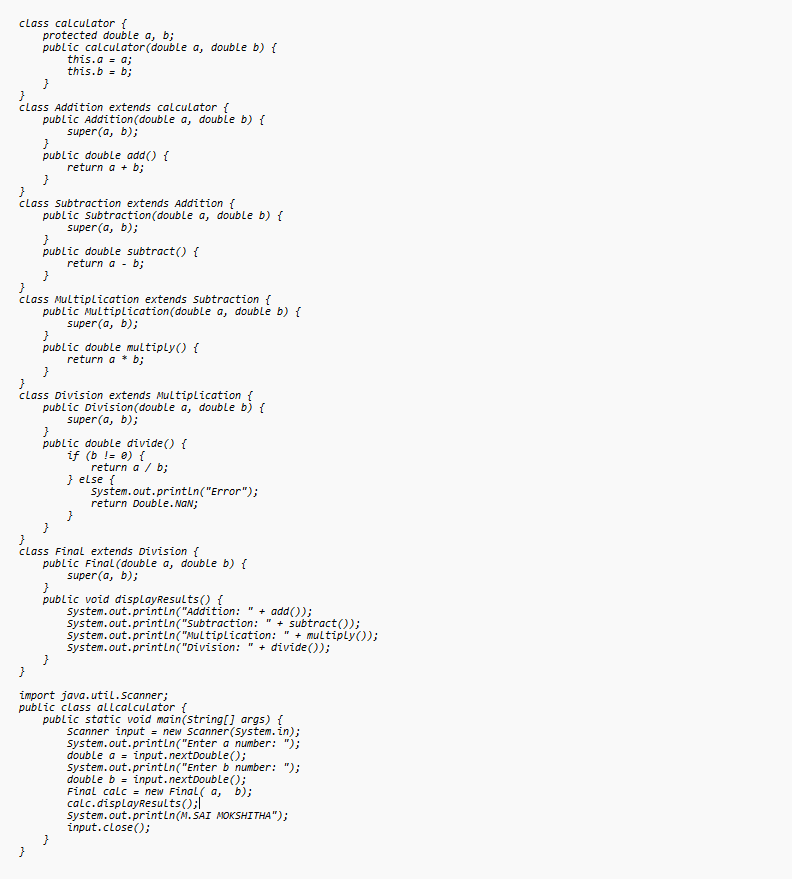
PROGRAM-1:

AIM: Create a calculator using the operations including addition, subtraction, multiplication, and division using multi-level inheritance and display the desired output.

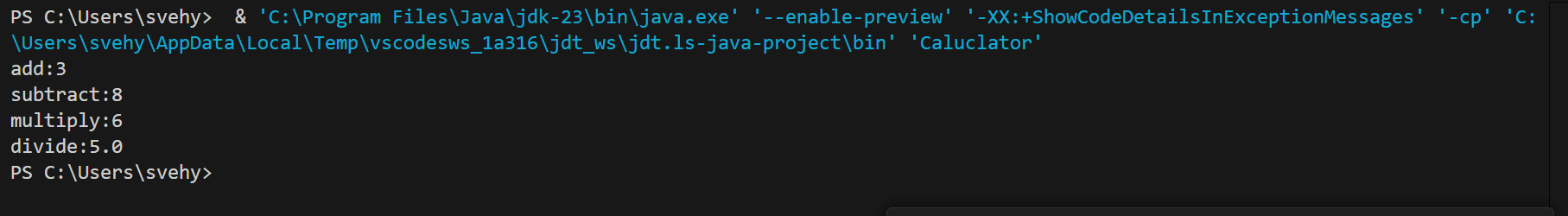
Hint: collect required variables using super class,

Create each class for a parameter and each class must contain a method.

CODE:



OUTPUT:



CLASS DIAGRAM:

|  |
| --- |
| Calulator   * a : double * b : double   + Calculator(a,b) |

|  |
| --- |
| Addition  + add(): double |

|  |
| --- |
| Subtraction  + subtract(): double |

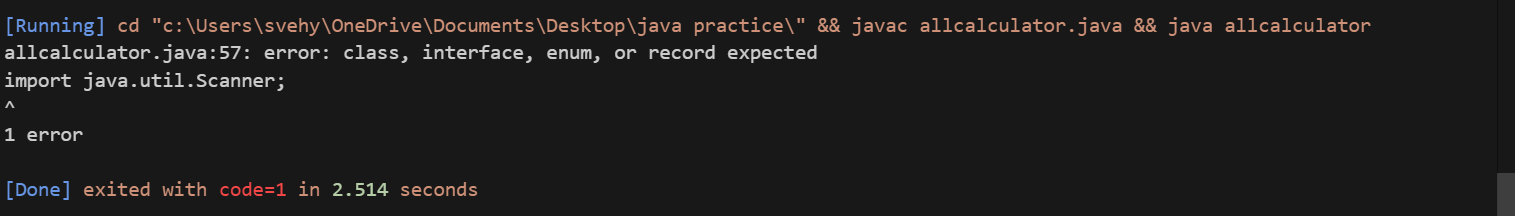
|  |
| --- |
| Multiplication  +multiply(): double |

|  |
| --- |
| Division  +divide(): double |

ERROR TABLE:

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. not providing the return method correctly. 2. Not mentioning super to obtain the super class constructor. | 1. After declaring methods, we must provide the return method correctly. 2. To obtain the super class we need to mention super. |

NEGATIVE CASE:



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(\*).

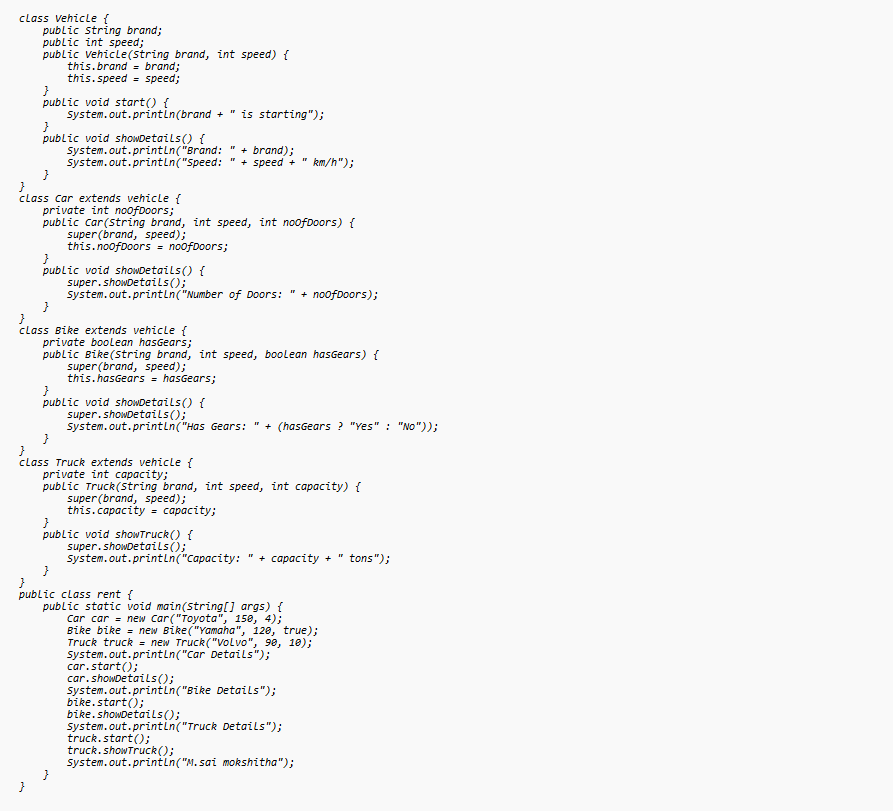
PROGRAM-2

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)

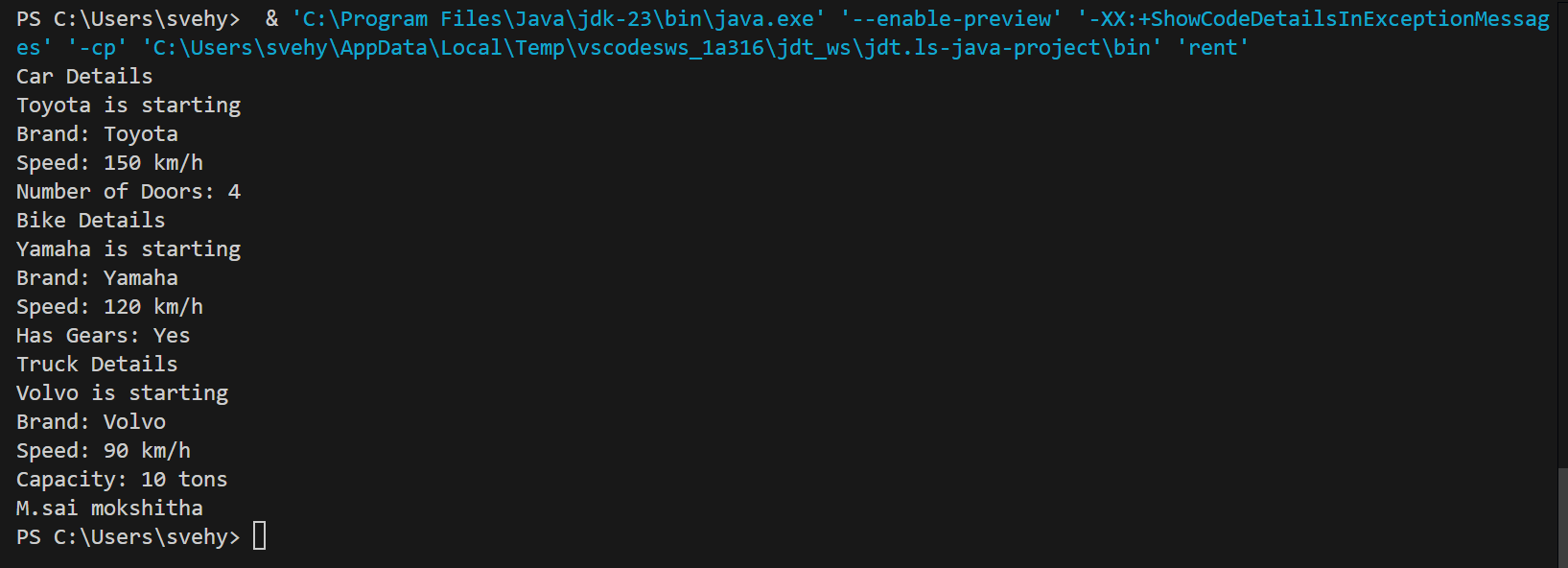
1. cars should have an additional property: no.of doors
2. Bikes should have a property indicating whether they have gears or not.
3. The system should also include a function to display details about each vehicle and indicate when a vehicle is starting.
4. Every class should have a constructor

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.



**OUTPUT:**



CLASS DIAGRAM:

|  |
| --- |
| Vehicle  -Brand : str  -Speed: int  + init (brand, speed)  + start\_vehicle()  + display\_details() |

Car Bike Truck

-

-no.of.doors: int -has gears: bool - capacity: float

+init (brand, speed +init (brand, speed displaydetails()

, no.of.doors) , has gears)

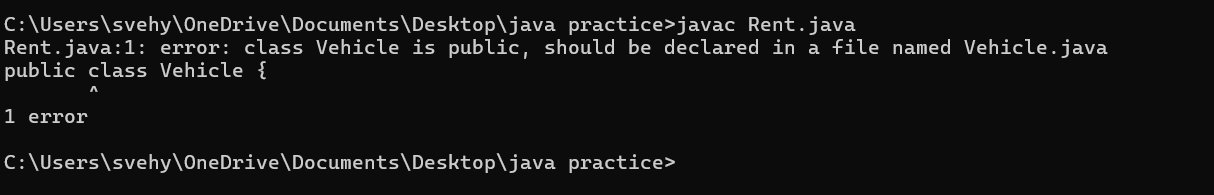
+displaydetails() +showtruckdetails()

+displaydetails()

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:

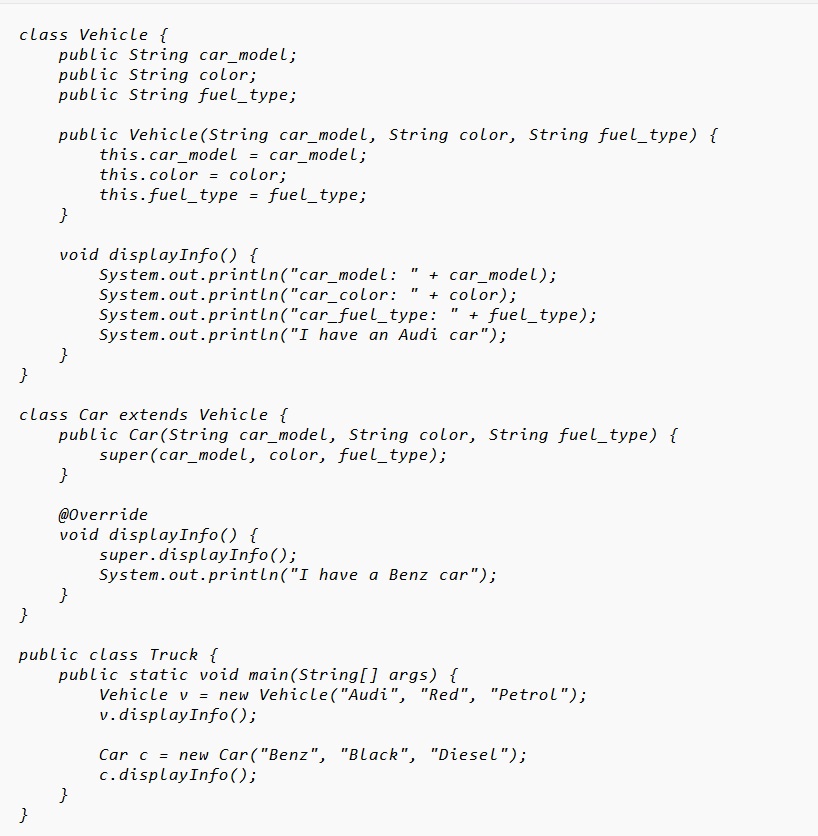
* Include an additional property capacity (in tons).
* Implement a showtruckdetials() method to display the truck's capacity.
* Implement a constructor for the truck class to initialize all its properties.

**WEEK-6:**

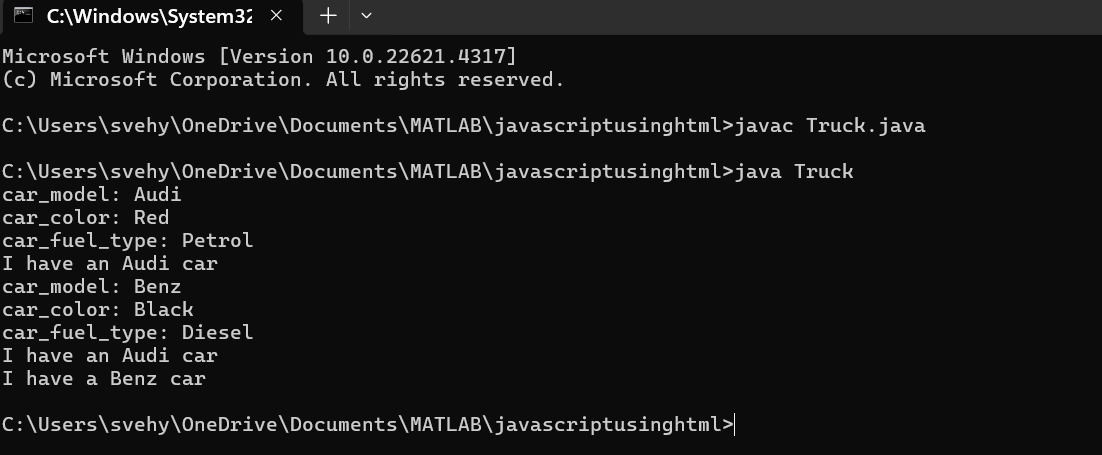
PROGRAM-1

AIM:

Write a Java program to create a vehicle class with a method displayInfo(). Override this method in the car subclass to display specific information about a car.



OUTPUT:



CLASS DIAGRAM:

Vehicle

──────────────────────────────

- brand: String

- speed: int

├──────────────────────────────

+ Vehicle(brand: String,

speed: int)

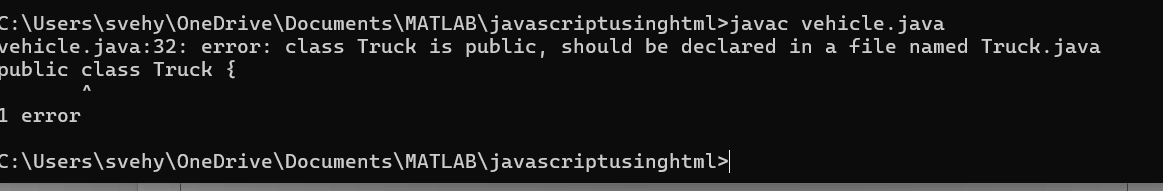
+ startVehicle(): void

+ displayDetails(): void

ERROR TABLE:

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

Negative case:



IMPORTANT POINTS:

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

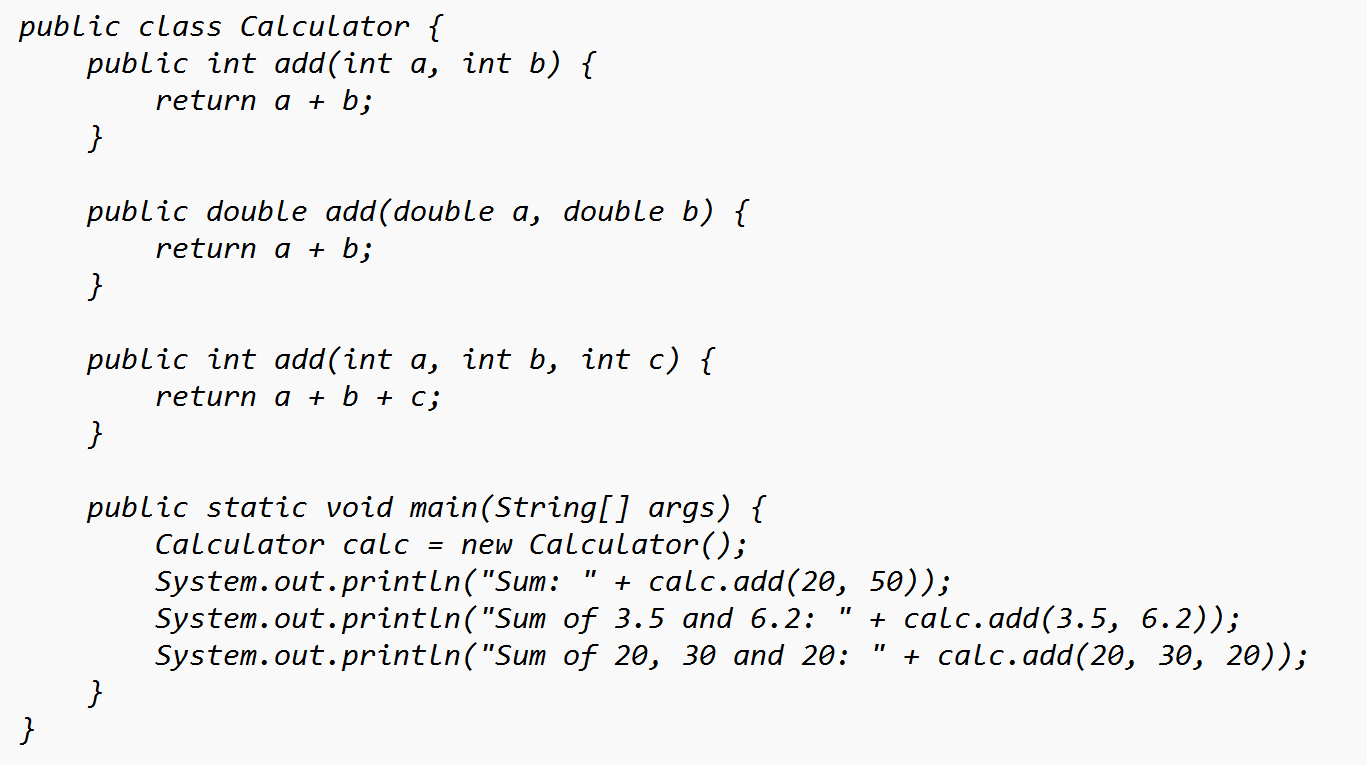
**Constructor Chaining:**The Car class calls the parent constructor using super(car\_model, color, fuel\_type); to initialize inherited attributes.

**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.

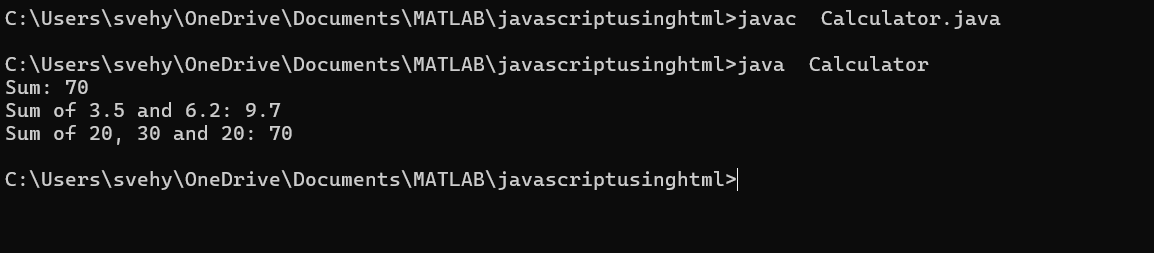
**Incorrect** main **Class Name:**The main method is inside Truck, which is unrelated to Vehicle and Car. The class should be renamed for clarity.

**program-2**

Create a calculator with overloaded methods to perform addition:  
(i) Add two integers.  
(ii) Add two doubles.  
(iii) Add three integers.



OUTPUT:



CLASS DIAGRAM:

Calculator

+add(int, int): int

+add(double, double): double

+add(int, int, int): int

+main(String[]): void

ERROR TABLE:

Code Errors CODE RECTIFICATIONS

1. Method parameters missing spaces 1.Add proper spacing between

Parameters:

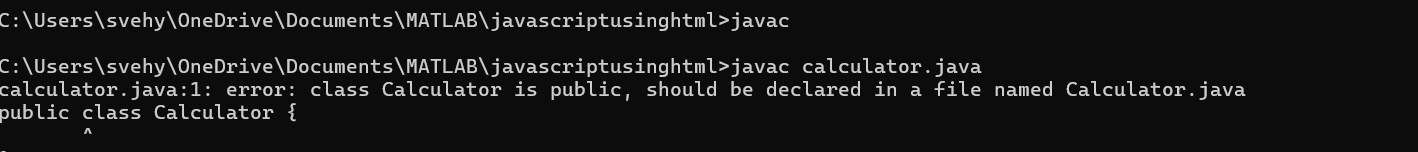
(e.g., "int a,int b" should be "int a, int b") -old:add(int a,intb)

-New:add(int a,int b)

2. Inconsistent indentation in method bodies 2.Fix indentation:

(some lines not properly aligned) consist 4 space of indentation

NEGATIVE CASE:

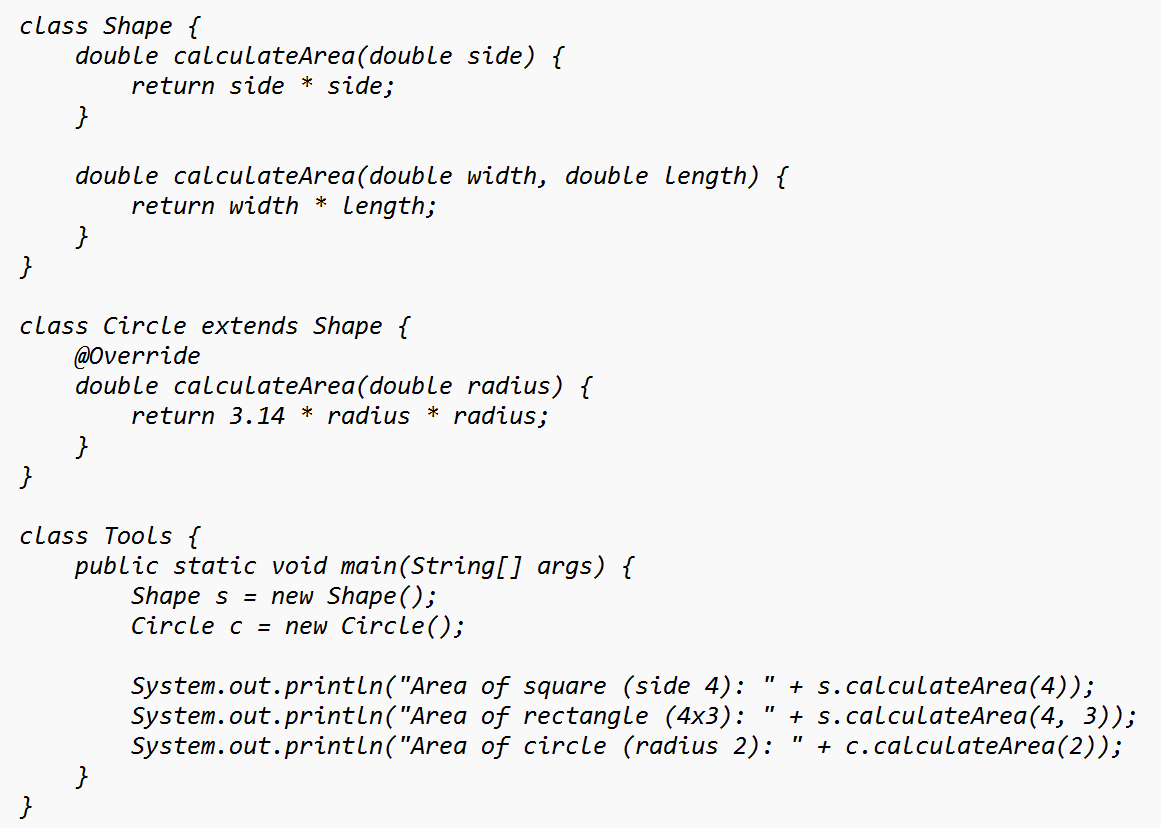


IMPORTANT POINTS:

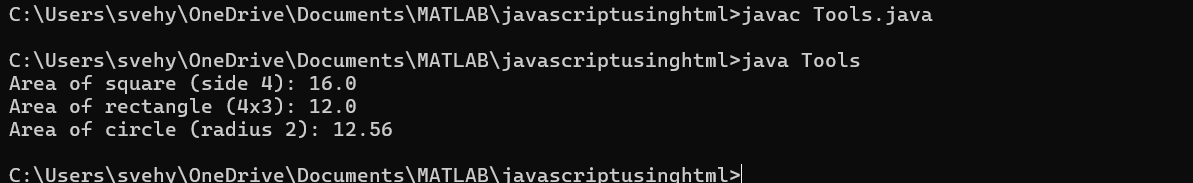
1. **Method Overloading:** The add method is overloaded with different parameter types and counts, demonstrating compile-time polymorphism.
2. **Automatic Method Selection:** Java selects the appropriate add method based on the argument types during compilation.

Program-3

Create a class with a method `calculateArea()` that is overloaded for different shapes (e.g., square, rectangle). Then create a subclass `Circle` that extends this class and overrides the method to calculate the area of a circle.



**OUTPUT:**



****Class Diagram****

SHAPE

+calculateArea(side: double): double

+calculateArea(width: double, length: double): double

▲

CIRCLE

+calculateArea(radius: double): double

▲

Tools

+main(args: String[]): void

### ERROR TABLE:

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

Negative case:

### 

### **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)).

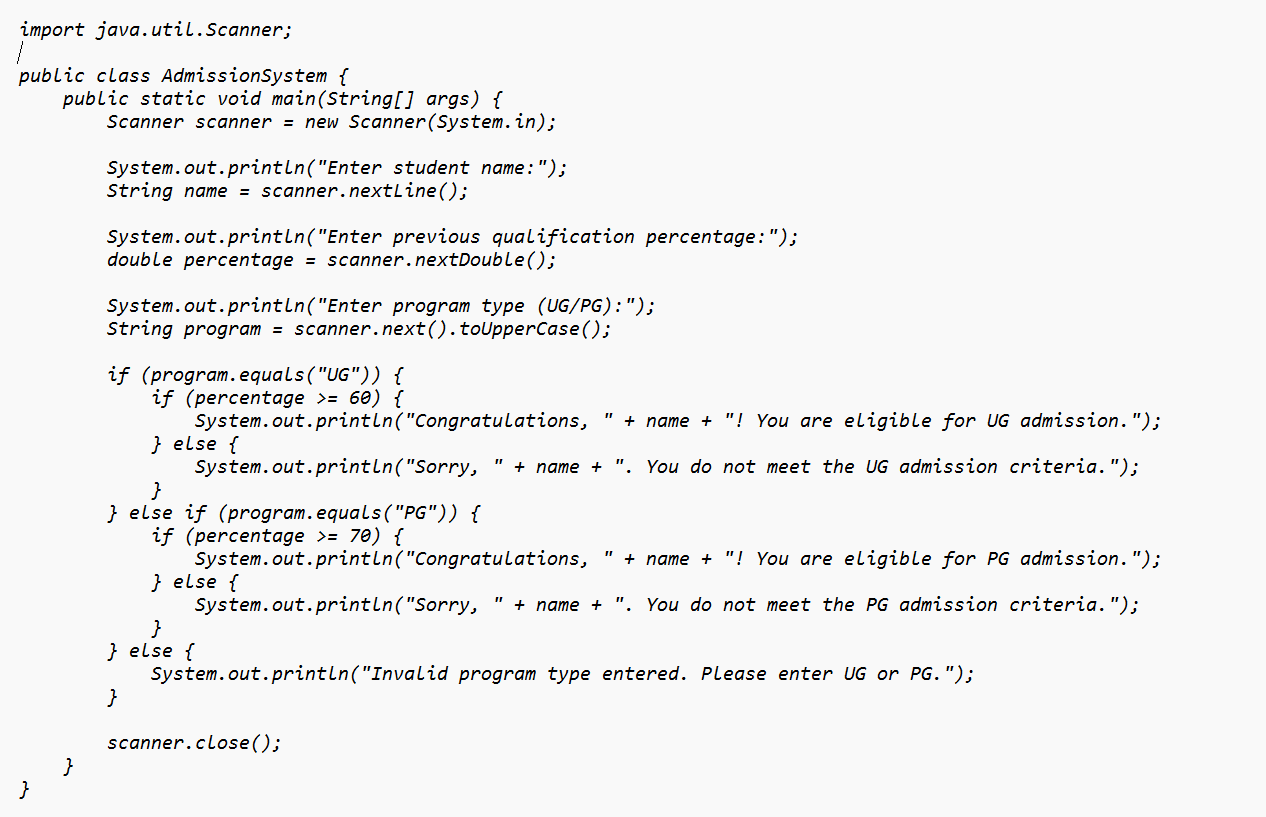
Program-4:

AIM:

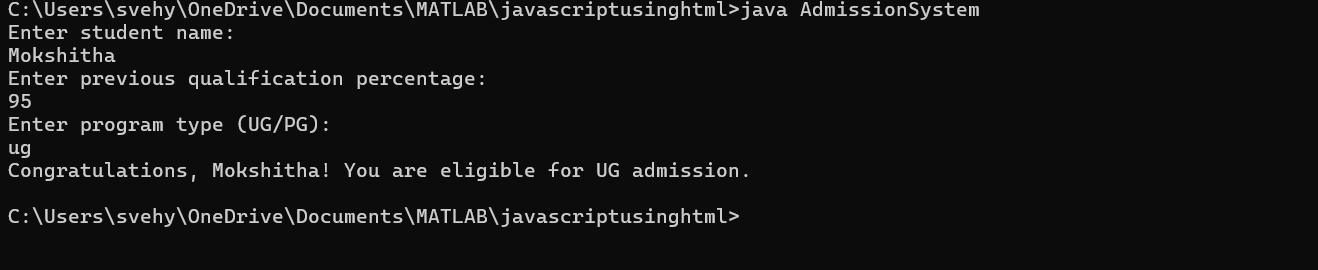
A college is developing an automated admission system that verifies students' eligibility for undergraduate (UG) and postgraduate (PG) programs. Each program has different eligibility criteria based on the student's percentage in their previous qualification:

UG admission requires a minimum of 60%.

PG admission requires a minimum of 40%.



OUTPUT:



CLASS DIAGRAM:

AdmissionSystem

- scanner: Scanner

- name: String

- percentage: double

- program: String

+ main(args: String[]): void

+ takeInput(): void

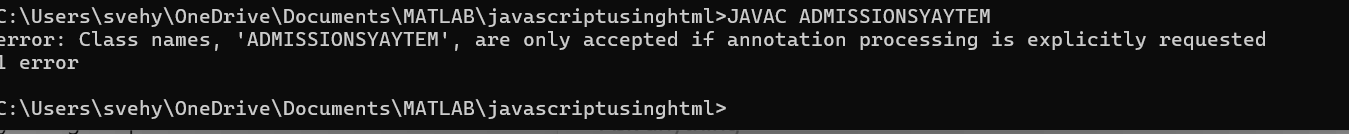
+ checkEligibility(): void

+ closeScanner(): void

### ****Error Table:****

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

NEGATIVE CASE:



### ****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 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: M.sai mokshitha, Rollno: AV.SC.U4CSE2451, Section: B");

Lion l = new Lion();

Tiger t = new Tiger();

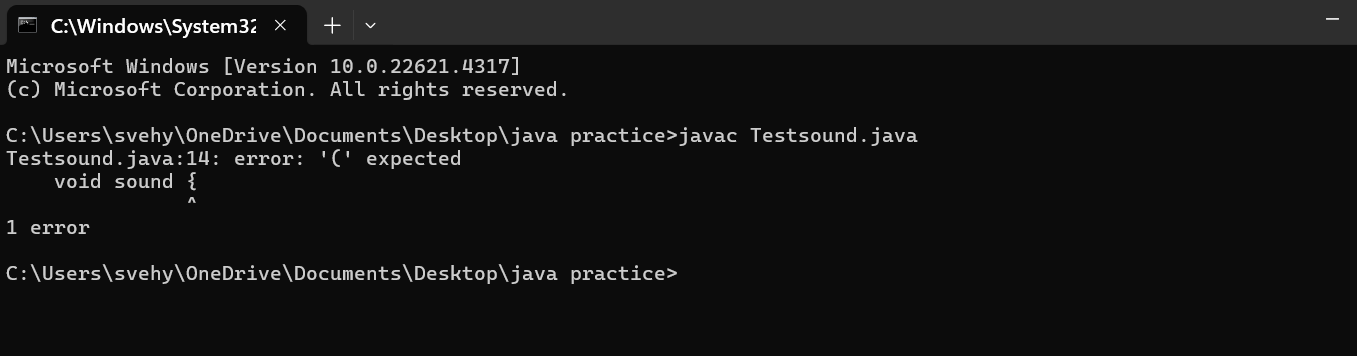
l.sound();

t.sound();

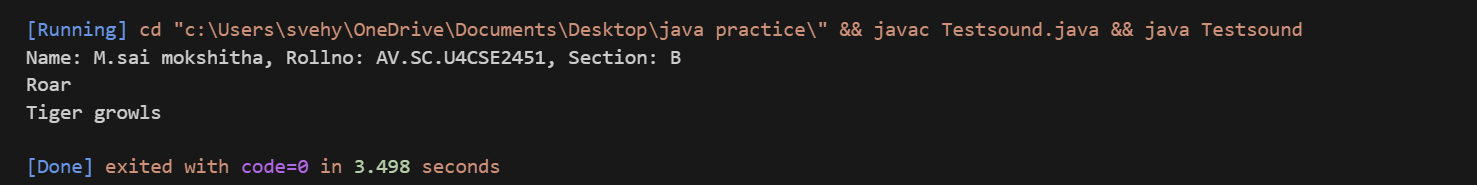
}

}

**NEGATIVE CASE**:



**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.

**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) \* 3.14 \* radius \* radius \* radius;

}

@Override

double surfaceArea() {

return 4 \* 3.14 \* 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 3.14 \* radius \* radius \* height;

}

@Override

double surfaceArea() {

return (2 \* 3.14 \* radius \* height) + (2 \* 3.14 \* radius \* radius);

}

}

class ShapeTest {

public static void main(String[] args) {

System.out.println("Name: M.sai mokshitha, Rollno: AV.SC.U4CSE2451, Section: B");

Sphere s1 = new Sphere(5.0);

System.out.println("Sphere Volume: " + s1.volume());

System.out.println("Sphere Surface Area: " + s1.surfaceArea());

Cylinder c1 = new Cylinder(3.0, 7.0);

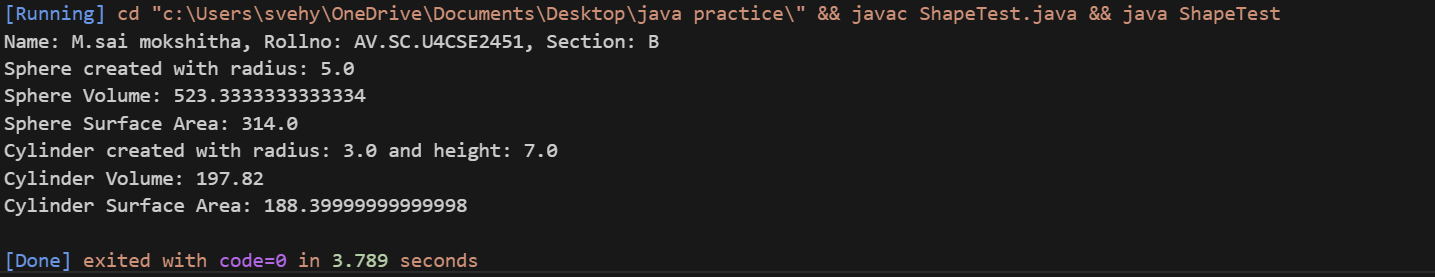
System.out.println("Cylinder Volume: " + c1.volume());

System.out.println("Cylinder Surface Area: " + c1.surfaceArea());

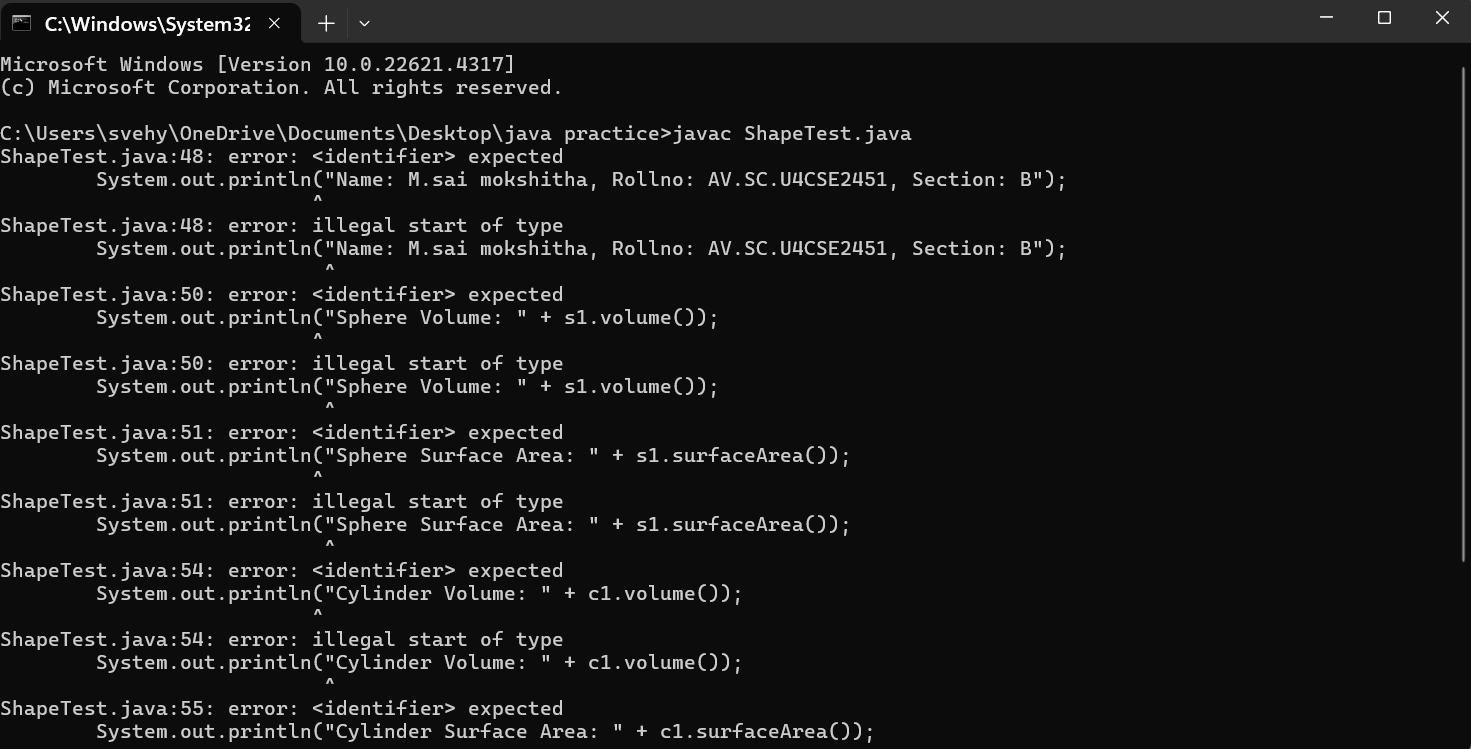
}

}

**OUTPUT:**



**NEGATIVE CASE:**



**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

**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:M.sai Mokshitha,Section:B,Roll NO:AV.SC.U4CSE24151");

int numberOfRows = 5;

PatternPrinter star = new StarPattern(numberOfRows);

star.displayTitle("Star Pattern");

star.printPattern();

PatternPrinter number = new NumberPattern(numberOfRows);

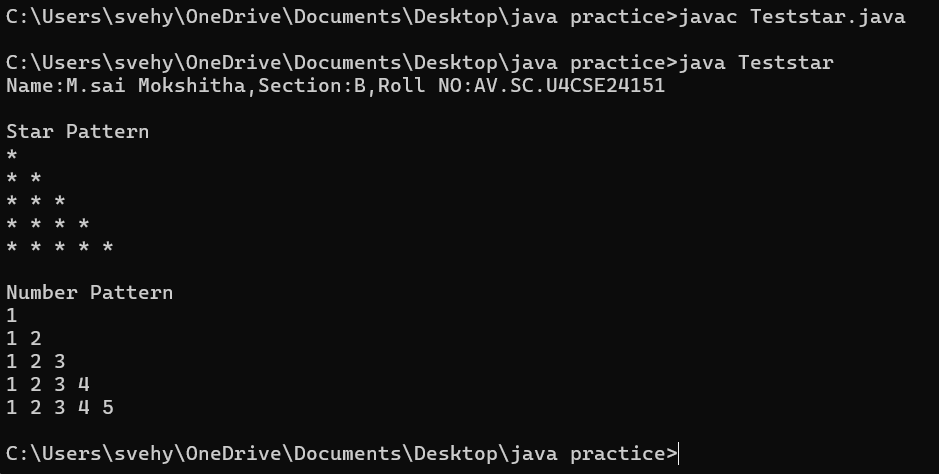
number.displayTitle("Number Pattern");

number.printPattern();

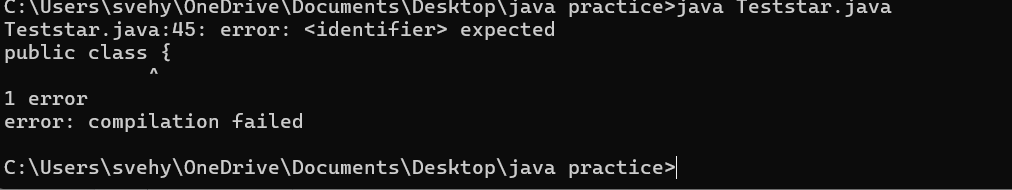
}

}

OUTPUT:



NEGATIVE CASE:



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.

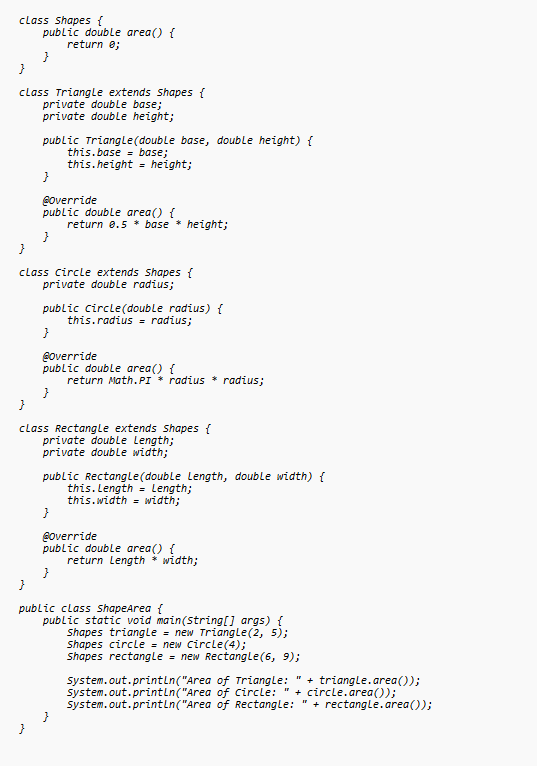
**WEEK-8**

**PROGRAM-1**

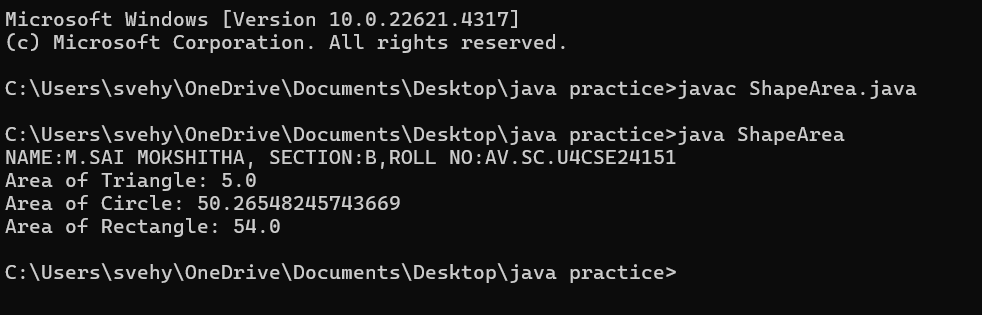
**AIM:**

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.

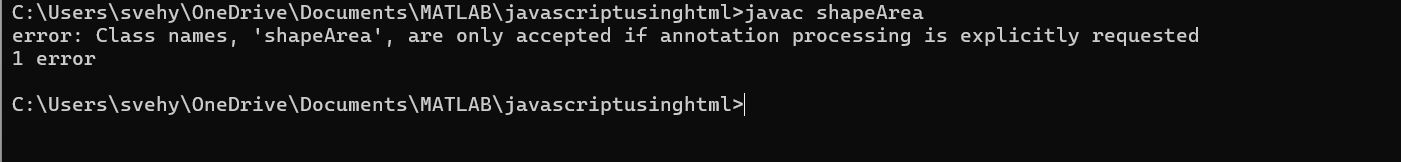
**Code:**



OUTPUT:



NEGATIVE CASE:



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 TestSports {

public static void main(String[] args) {

System.out.println("NAME:M.SAI MOKSHITHA,SECTION:B,ROLL NO:AV.SC.U4CSE24151");

Playable football = new Football();

Playable volleyball = new Volleyball();

Playable basketball = new Basketball();

football.play();

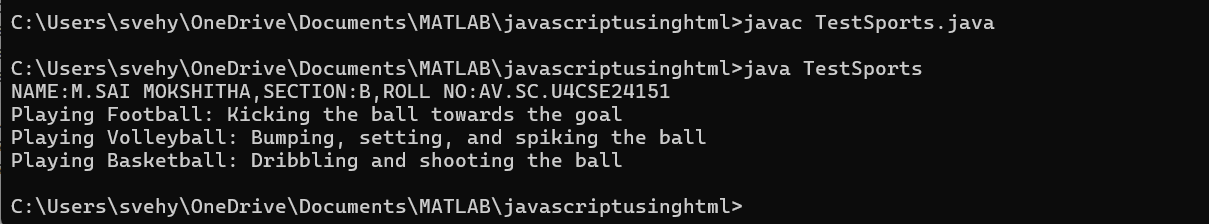
volleyball.play();

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("Name:M.Sai Mokshitha,section:B,RollNo:AV.SC.U4CSE24151");

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

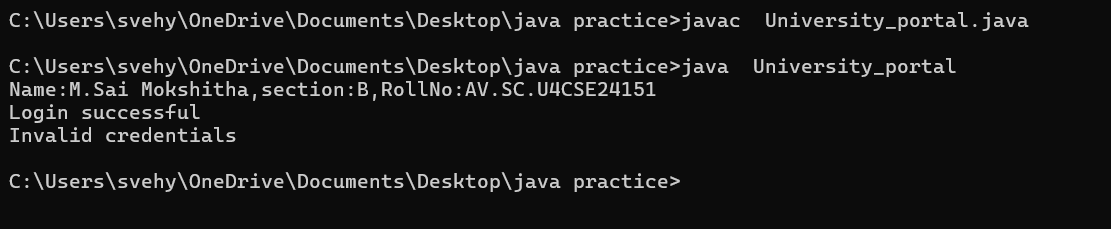
p1.login("Student123", "pass02");

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

}

}

OUTPUT:



CLASS DAIGRAM:

interface :Login system

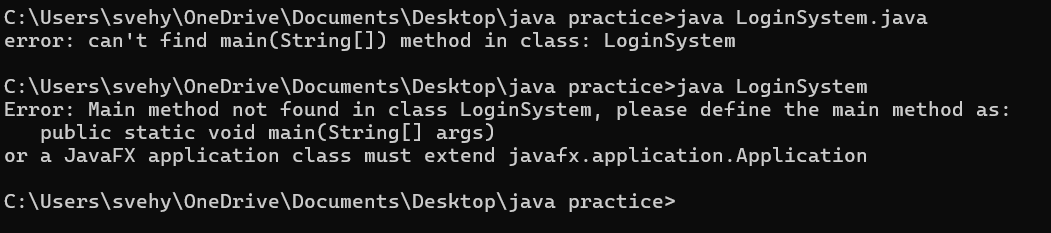
+ login(String,String): boolean

University\_portal

+ login(StringString): boolean

+ main(String[]): void

NEGATIVE CASE:



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