**23CSE101**

**COMPUTATIONAL PROBLEM SOLVING**

**LAB MANUAL**



**Department of computer and communication Engineering**

**Amrita School of Engineering**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Name: S.Rahitya**

**Verified By: RollNo: 24249**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.NO | DATE | TOPIC | PAGE | SIGNATURE |
| 1 |  | How to install jdk(java development kit)  From online flatform Oracle. |  |  |
| 2 |  | Write java program of student details. |  |  |
| 3 |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.NO | DATE | TOPIC | PAGE | SIGNATURE |
| 1 |  | Write a java simple interest. |  |  |
| 2 |  | Write a java program factorial of a number. |  |  |
| 3 |  | Write a java program to convert temperature C-F & F-C. |  |  |
| 4 |  | Write a java program area of rectangle,  Triangle. |  |  |
| 5 |  | Write a java program fibinocci series |  |  |

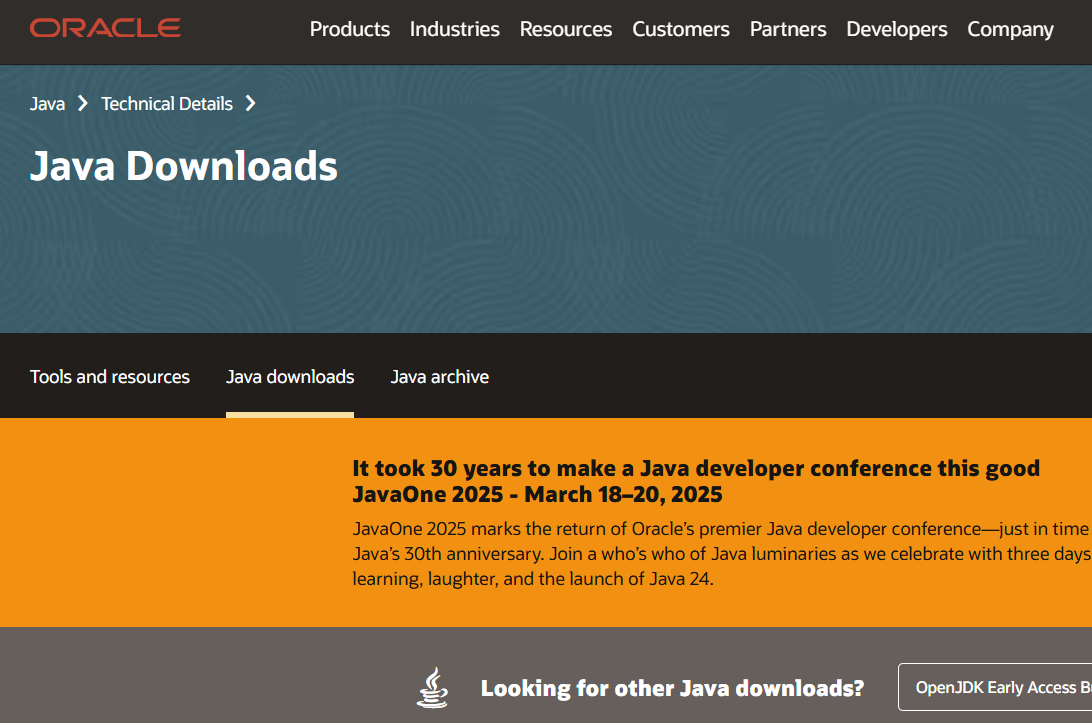
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.NO | DATE | TOPIC | PAGE | SIGNATURE |
| 1 |  | Create a java program with following instructions   1. Create a class with name car 2. Create four attributes named Car\_color , Car\_brand, fuel\_type,mileage 3. Create three methods named start(),stop(),service()   4.Create three objects named Car1, Car2 and Car3 |  |  |
| 2 |  | Create a class bankAccount with elements deposit() and Withdraw |  |  |

# WEEK-1

**AIM:**

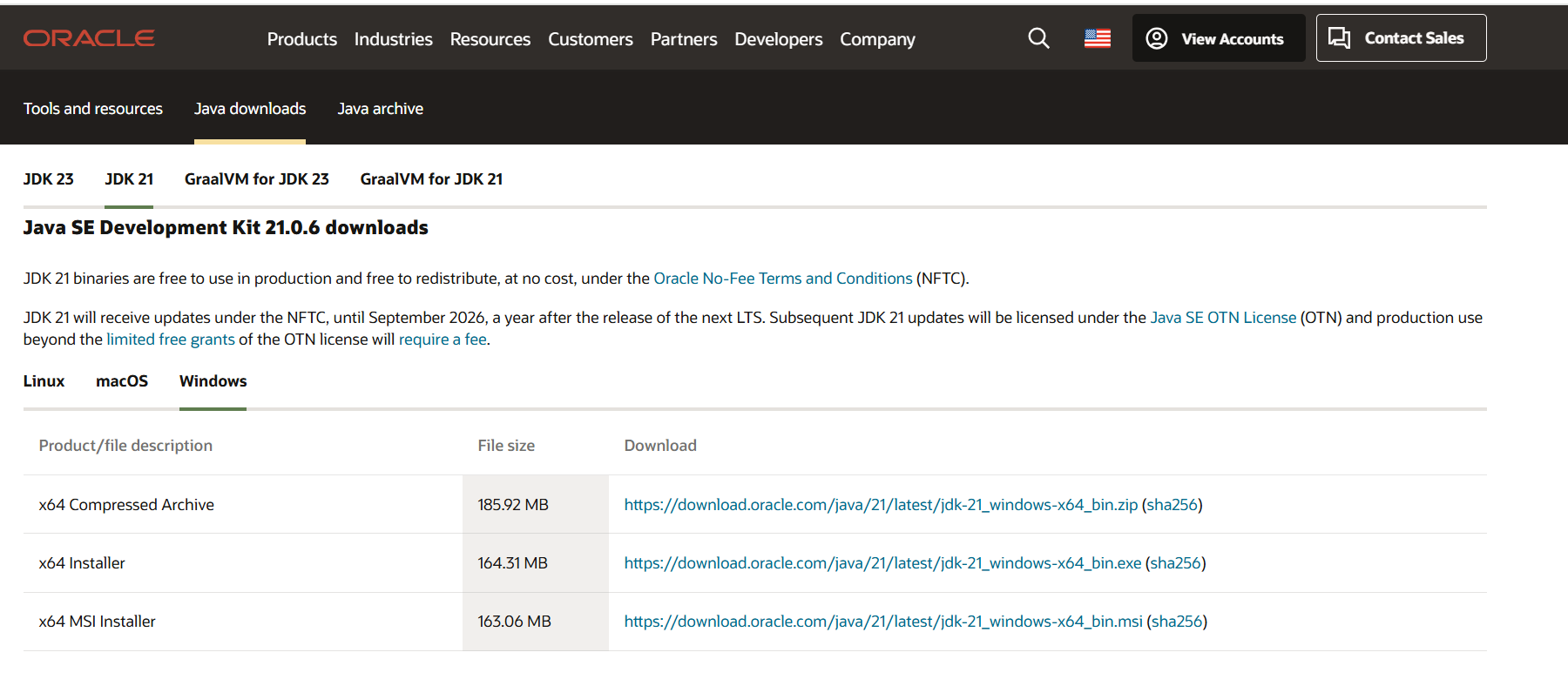
**1) Explain the process of Installing JDK (Java Development Kit)**

**Installing of JDK (Java Development:**

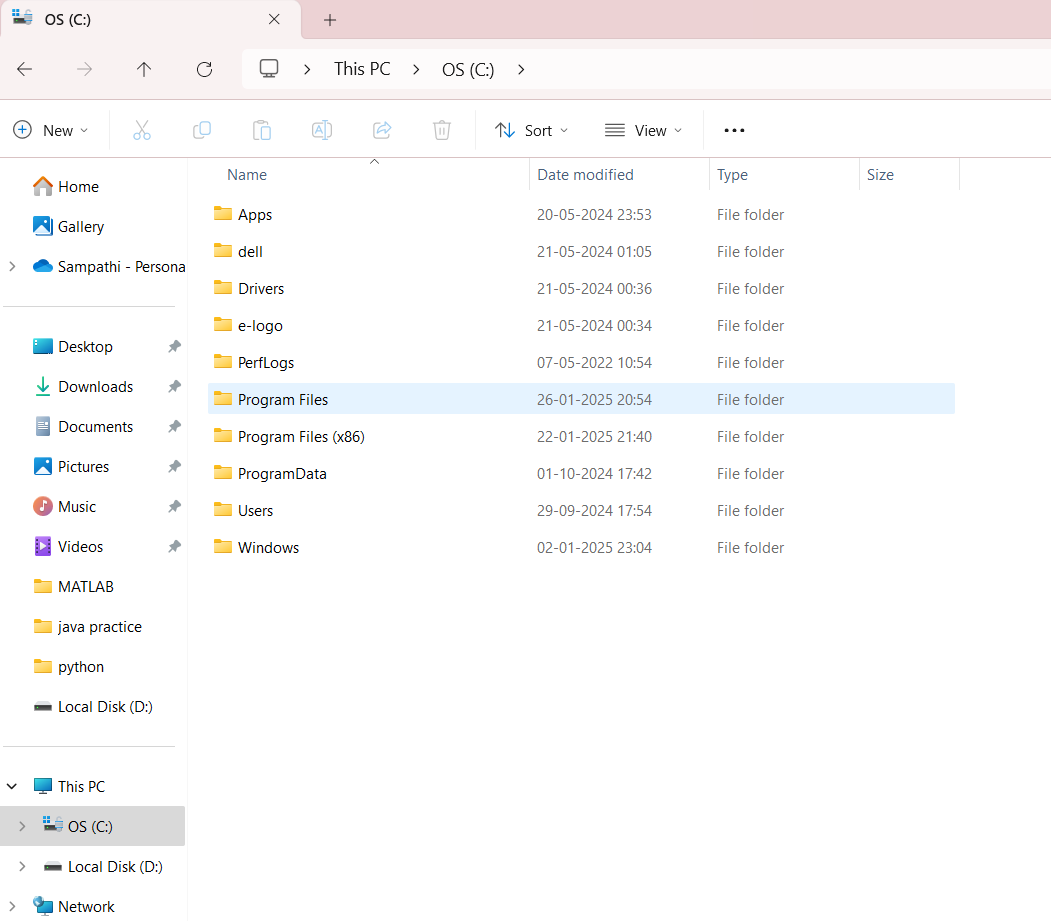
****

1. **Download JDK:**

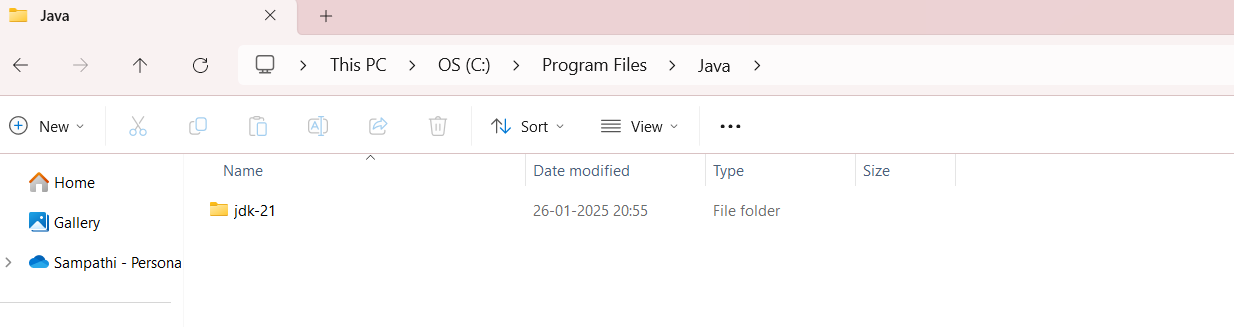
* Go to the google and browse Oracle java download and click on link

****

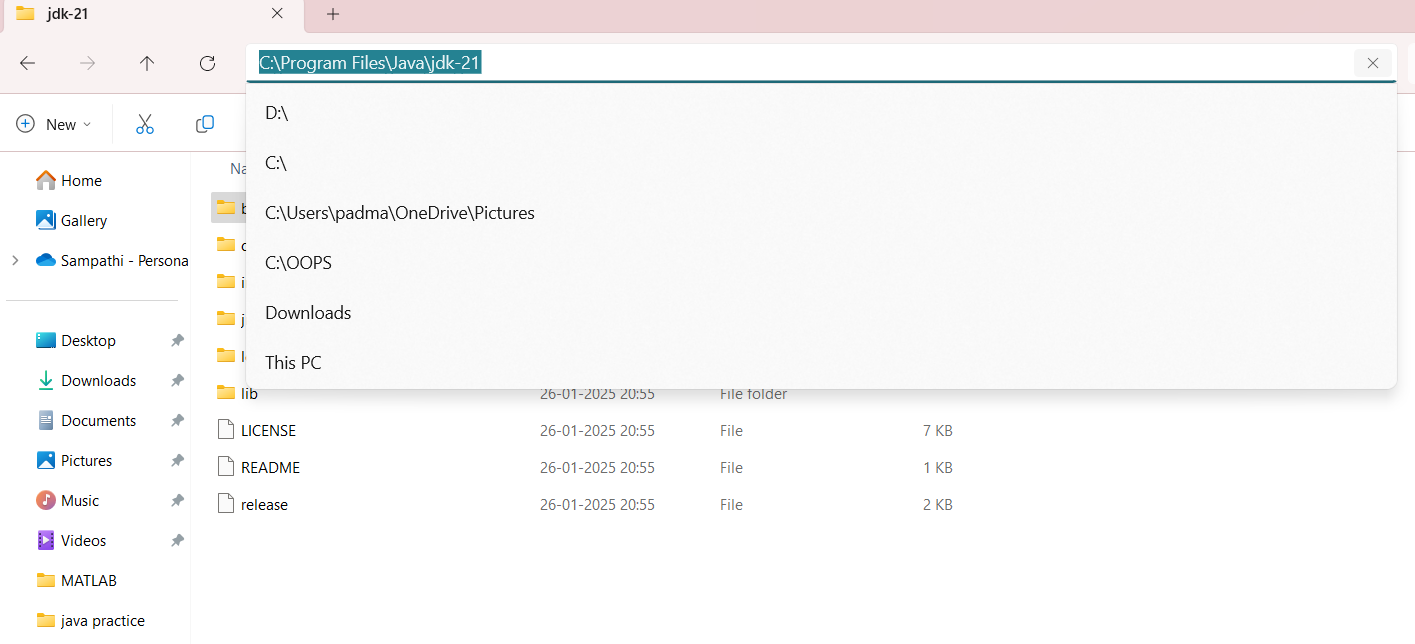
* After opening link scroll down and click on JDK-21 version which is long term support (LTS) version.
* Click on the download link for your operating system (Windows, macOS, or Linux).



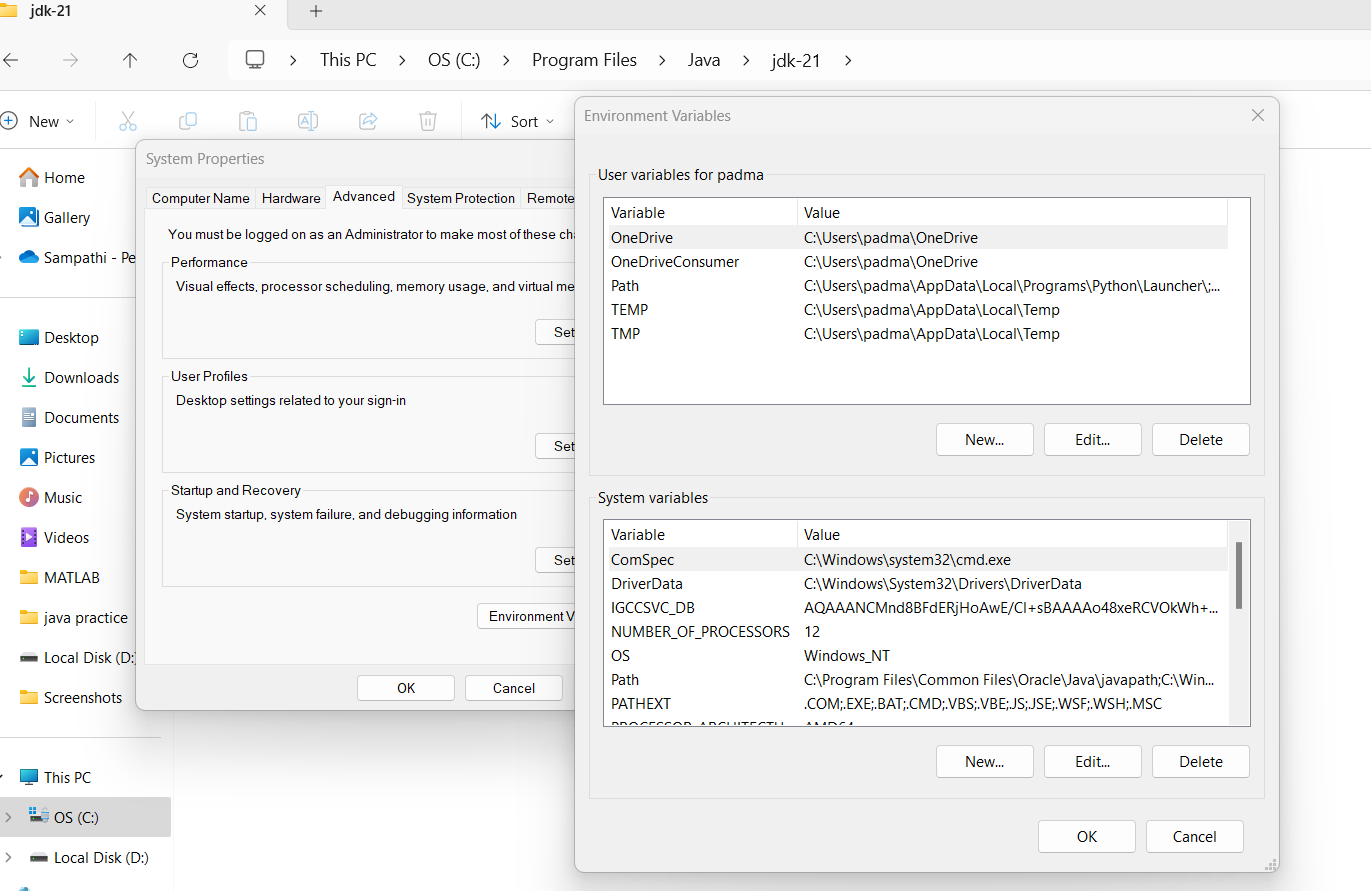
* Go to files and open os(c) after go to program files. Double clicks on program files.



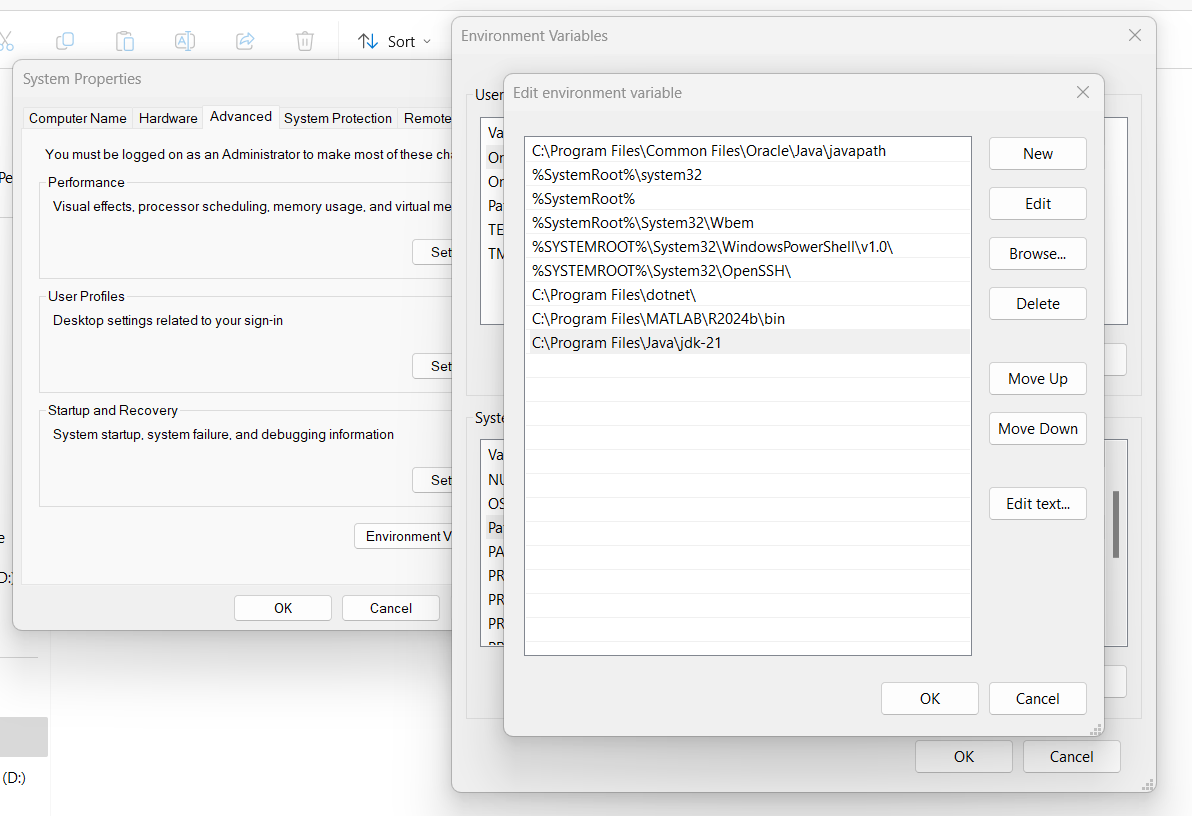
* Now again double click the jdk-21.



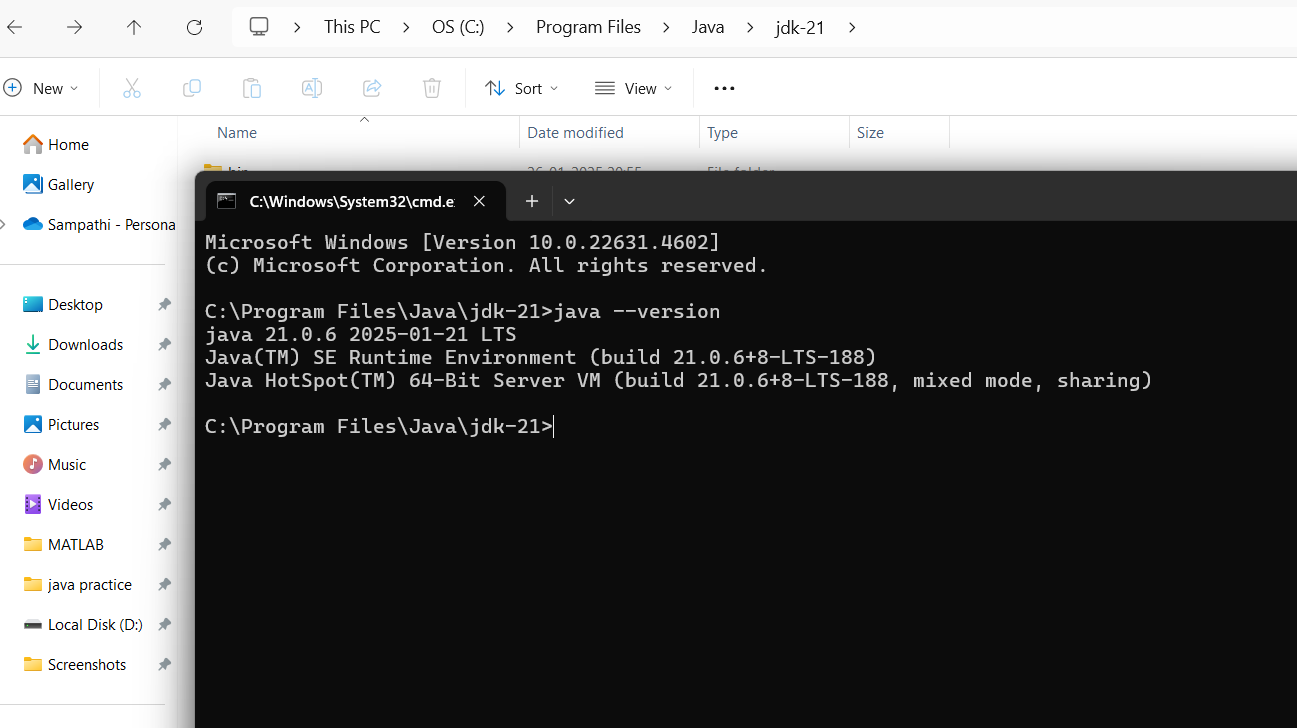
* After selecting bin keep cursor on search as shown fig then we can find link there just save that link.
* Go to search edit the environment.



* Now search edit the system environment.
* After go to environment variables choose system variables and select path.

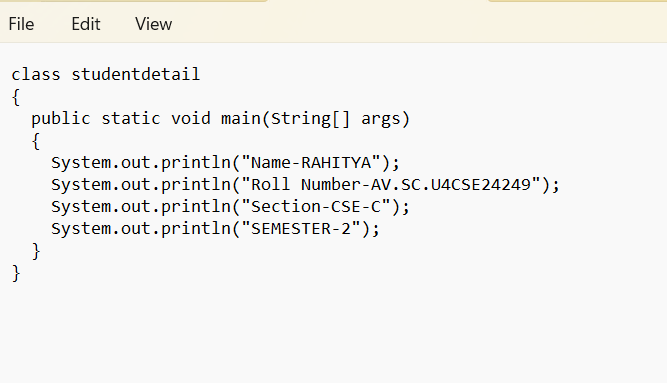


* Now go to path and paste the link which is saved before. After click on ok and apply.

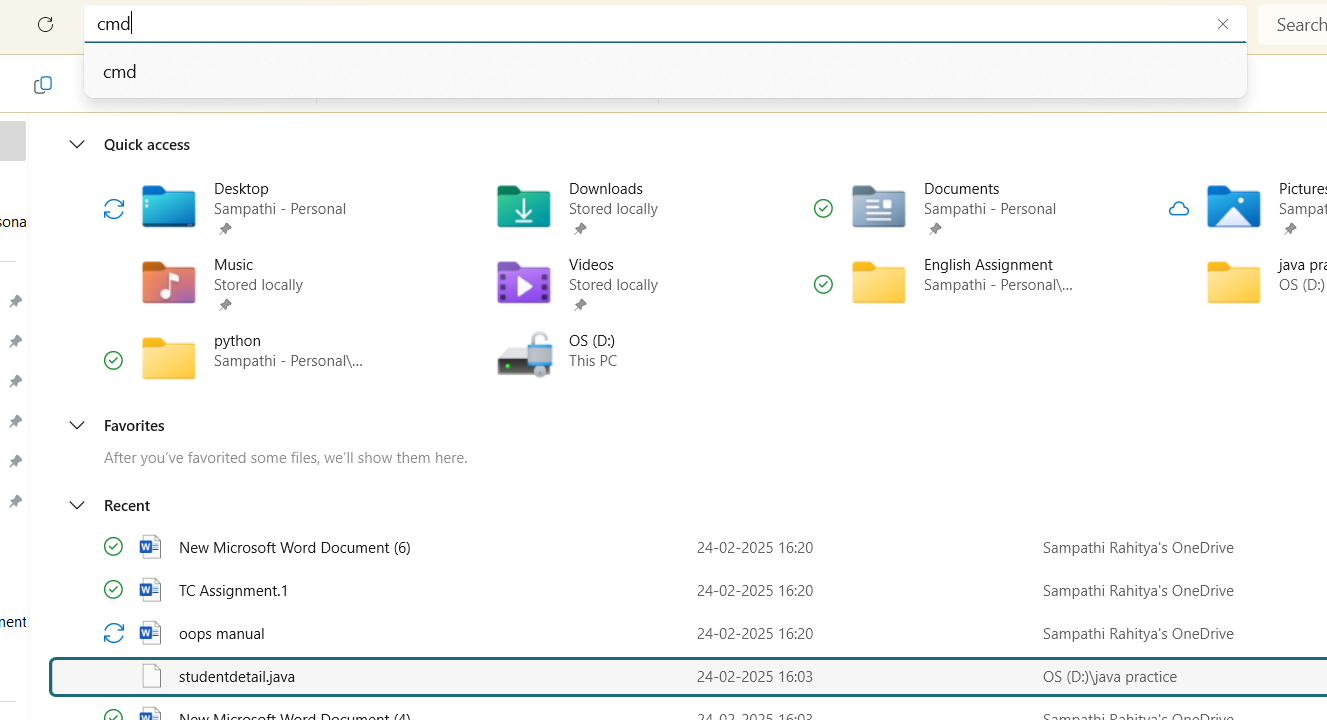


* Now select the file and check the version by going into cmd(command prompt) and type the java –-version. Then that show the which version we have download. If incase there is any error in the installation of the application then it shows the error in that.

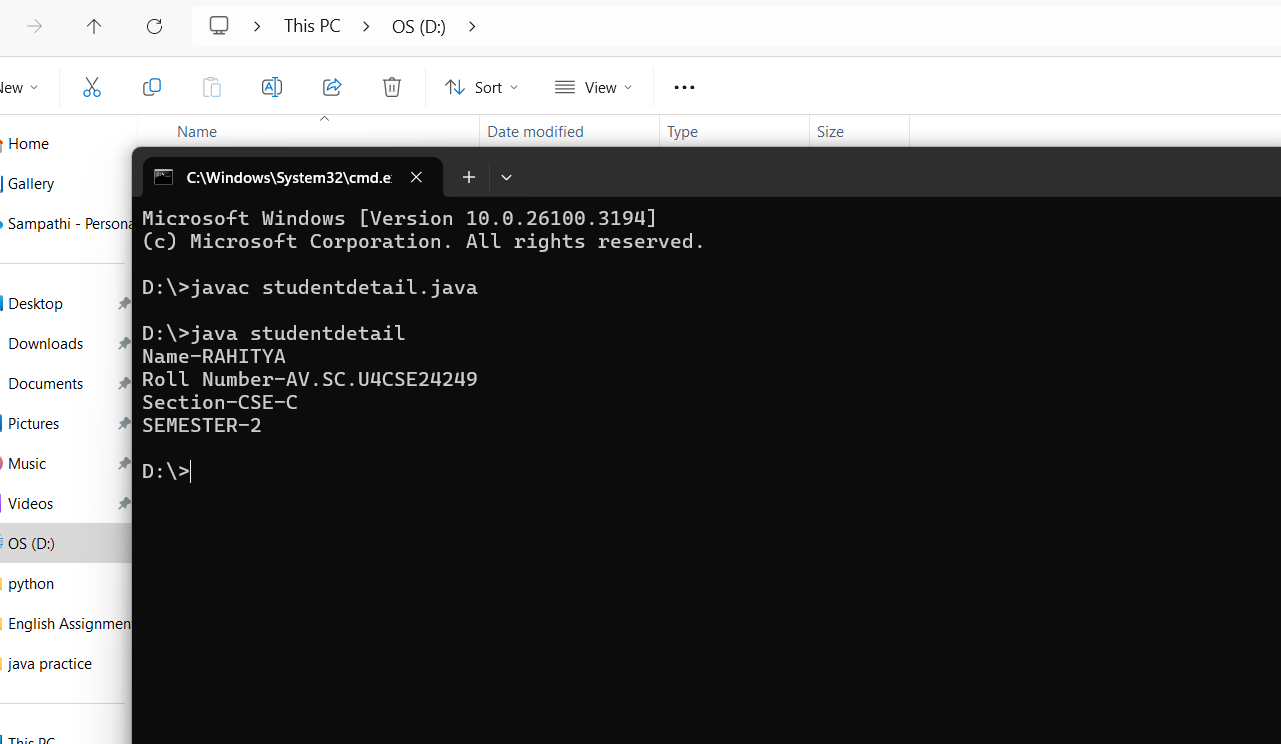
**STUDENT DETAILS**



* Now go to notepad and write student details by using java language with correct syntax.
* In class no space should be given ex: studentdetail.
* In main class (String), (System), S should be in capital. **ln** is used to print line by line.

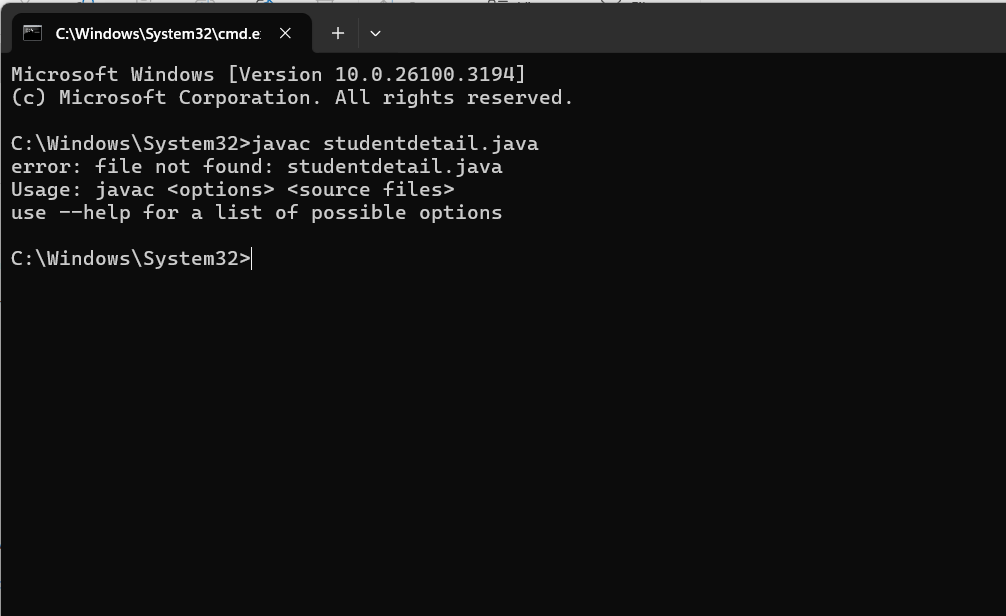
****

* Go to file and select the student details file which is saved and select the file and search the cmd(command prompt).



* Now check the studentdetail file is exists by using (javac studentdetail.java)
* Now check whether the file is running without any error or not by using (java studentdetail)

**Error:**

****

* We have to select the file in which we stored not from the recent files.

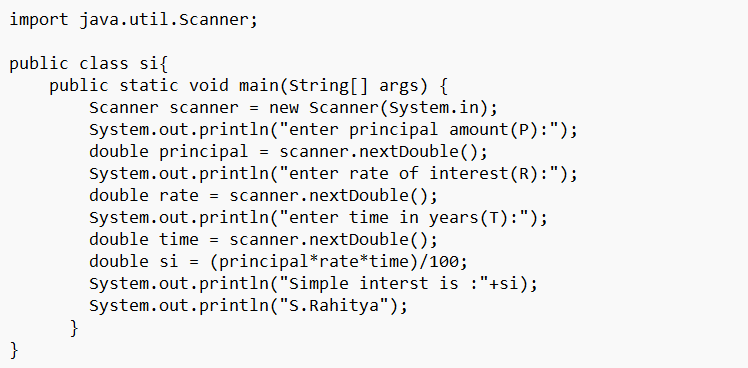
**WEEK -1**

**---THE END---**

**WEEK-2**

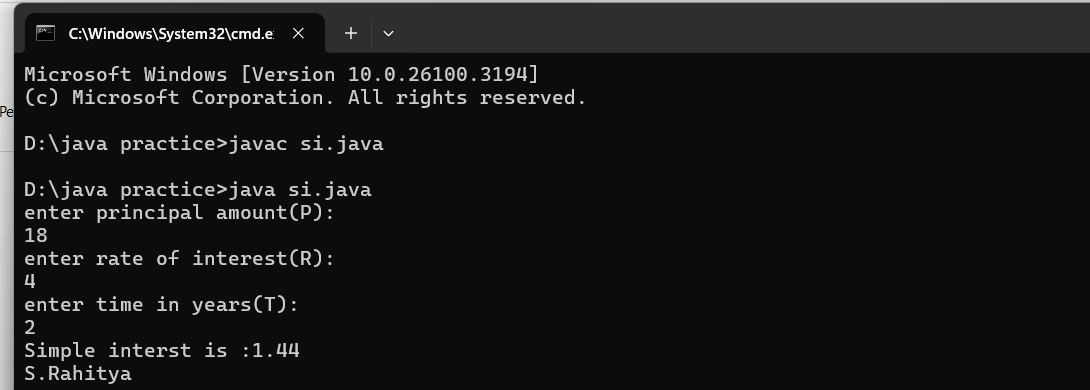
**AIM: Write a java program simple interest.**

**Code:**

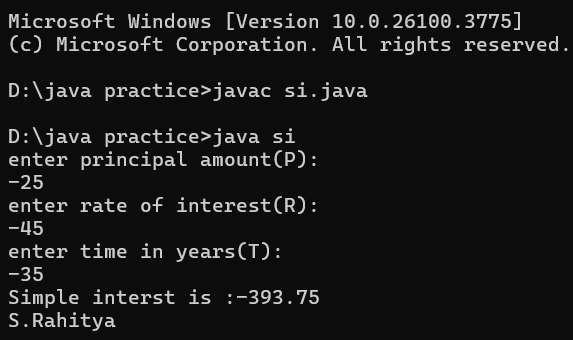
****

**Output:**

**Positive case:**

****

**Negative case:**

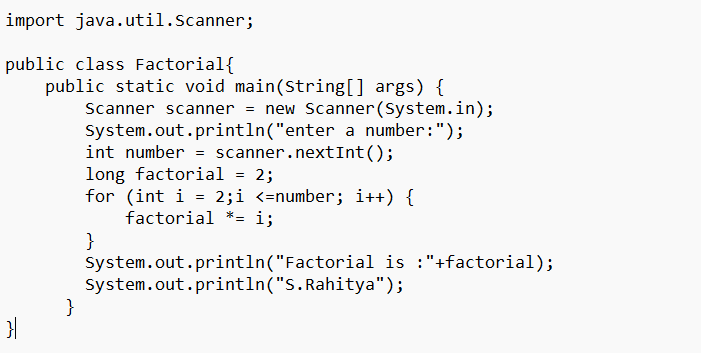
****

**Error table:**

|  |  |  |
| --- | --- | --- |
| **Error type** | **Reason** | **Rectification** |
| syntax error | “ \_ is use used thrice | Removed one “ |
| logic error | in formal I miss “R” | Rectification in formula |

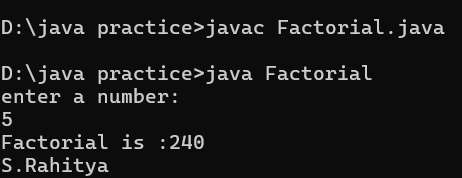
**AIM:Write a java program factorial of a number**

**Code:**

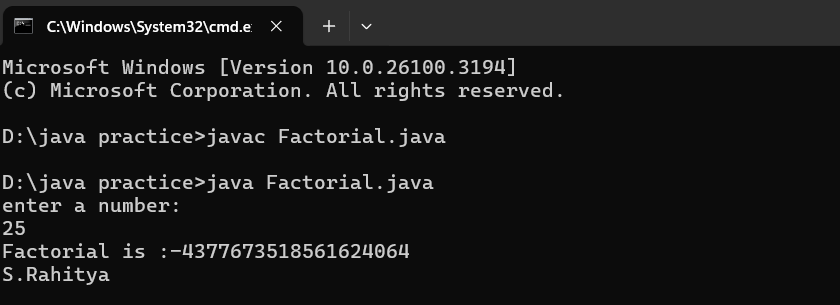
****

**Output:**

**Positive case:**

****

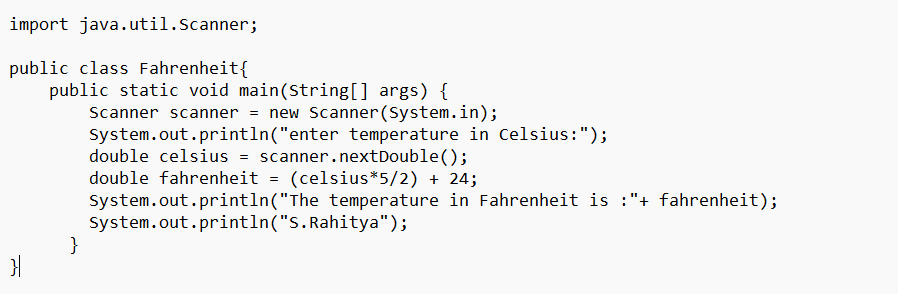
**Negative case:**

****

**Error table:**

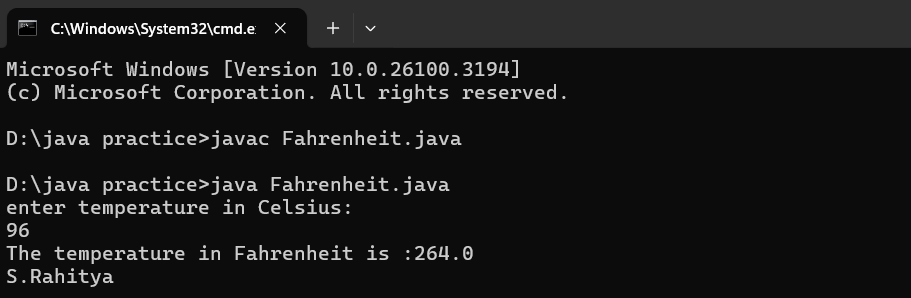
|  |  |  |
| --- | --- | --- |
| **Error type** | **Reason** | **Rectification** |
| Name error | Saving file another than class name | Save as class name |
| Runtime error | Path Incorrect | Copied correct path |

**AIM:Write a java code to convert the temperature from Celsius to Fahrenheit and from Fahrenheit to Celsius.**

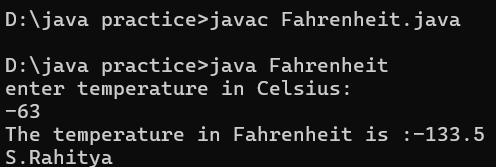
**(i)Code:**

**Output:**

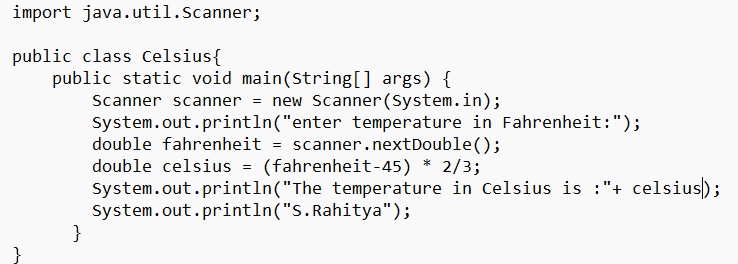
**Positive case:**

****

**Negative case:**

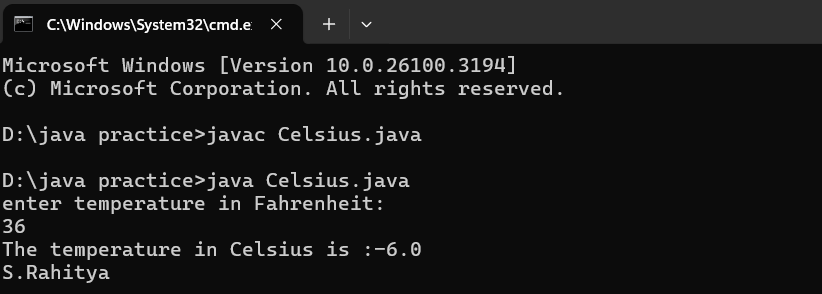
****

**(ii) Code:**

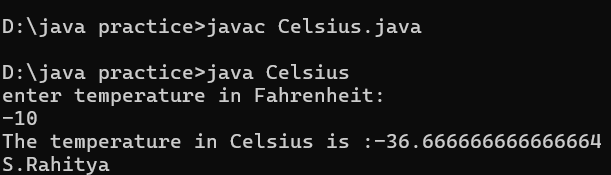
****

**Output:**

**Positive case:**

****

**Negative case:**

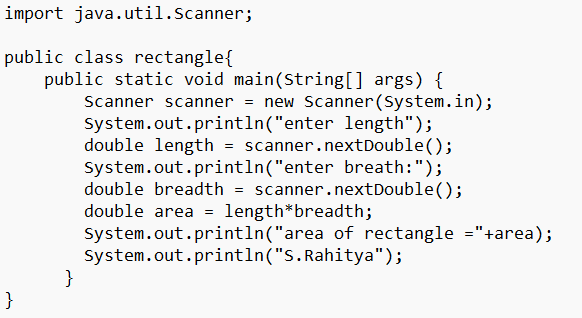
****

**Error table:**

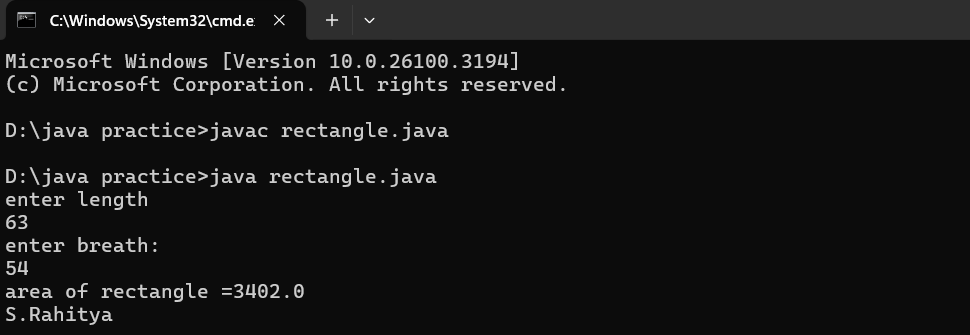
|  |  |  |
| --- | --- | --- |
| **Error type** | **Reason** | **rectification** |
| Logic error | Incorrect formula | Formula rectification |
| Syntax error | {- forget | Added – { |
| Variable error | Giving same name | Error Rectification |

**AIM:Write a program for rectangle**

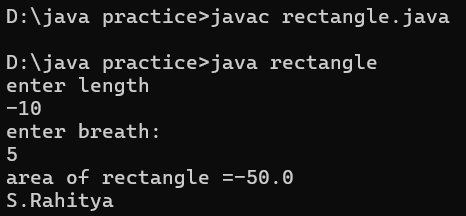
**Code:**

**Output:**

**Positive case:**

****

**Negative case:**

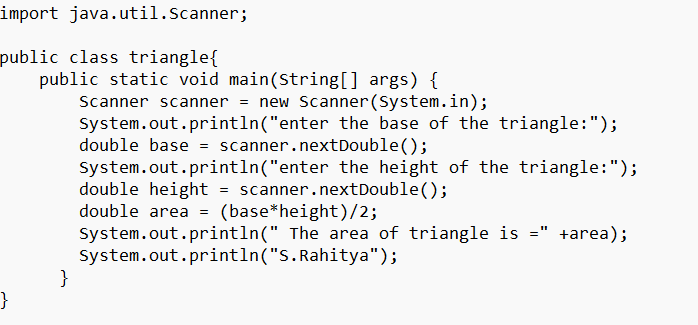
****

**Error table:**

|  |  |
| --- | --- |
| **Error** | **Rectification** |
| Mistake in main class | Give correct code |
| Synax error } not kept | } is added |

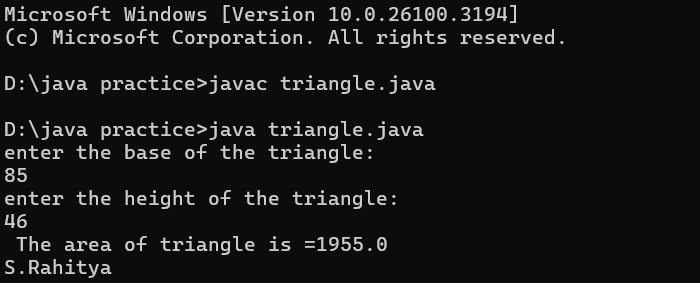
**AIM:Write a java program to find the area of triangle**

**Code:**

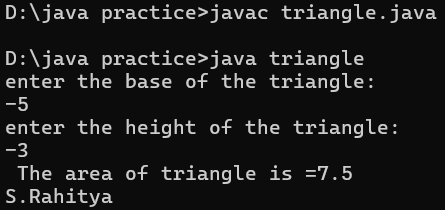
****

**Output:**

**Positive case:**

****

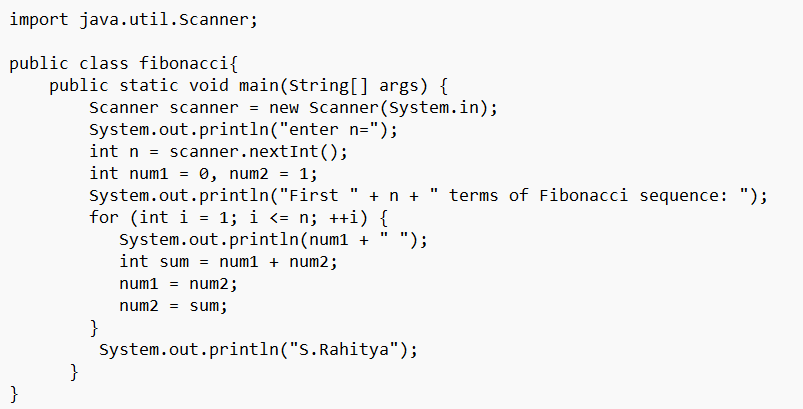
**Negative case:**

****

**Error table:**

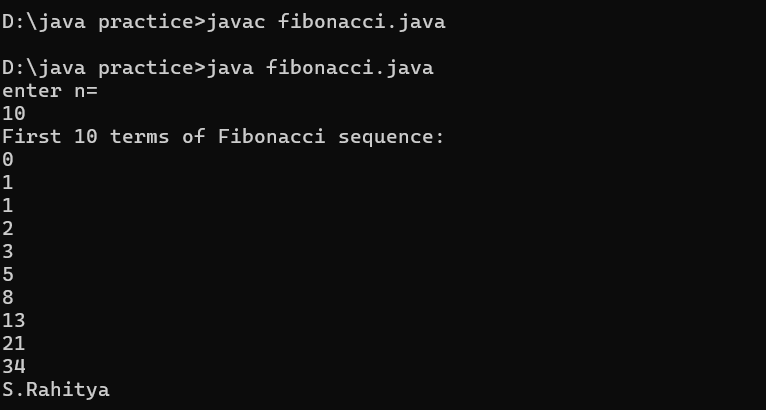
|  |  |  |
| --- | --- | --- |
| **Error type** | **Reason** | **Rectification** |
| Error in cmd | Entering wrong file name | Giving correct name to run the file |

**AIM:Write a java program fibonacci series**

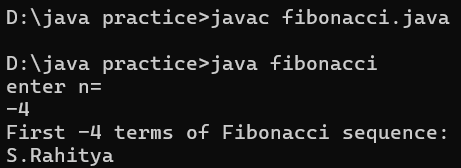
**Code:**

**Output:**

**Positive case:**

****

**Negative case:**

****

**Error table:**

|  |  |  |
| --- | --- | --- |
| **Error type** | **Reason** | **Rectification** |
| Syntax error | Use small letter | Have to use capital letter |
| Compile error | Misspelled variable | Correct variable name |

**WEEK 2**

**------------ THE END --------------**

**Week-3**

**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:**

|  |
| --- |
| Car  -car\_color:string  -  car\_brand:string  -fuel\_type:string  -milage: double  --------------------------  +start():void  +stop():void  +service():void |

**Code:**

import java.util.\*;

class car

{

public String Car\_color;

public String Car\_brand;

public String fuel\_type;

public int mileage;

public void start()

{

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

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

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

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

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

}

public void service()

{

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

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

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

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

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

}

public void stop()

{

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

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

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

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

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

}

public static void main(String args[])

{ System.out.println("S.Rahitya");

car car1 = new car();

car1.Car\_color = "green";

car1.Car\_brand = "Mahindra";

car1.fuel\_type = "Diesel";

car1.mileage = 250;

car1.start();

car car2 = new car();

car2.Car\_color = "white";

car2.Car\_brand = "tata motors";

car2.fuel\_type = "EV";

car2.mileage = 300;

car2.stop();

car car3 = new car();

car3.Car\_color = "Pink";

car3.Car\_brand = "Suzuki";

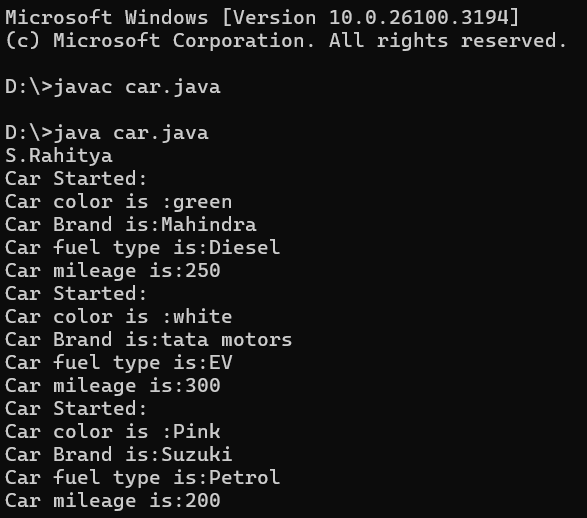
car3.fuel\_type = "Petrol";

car3.mileage = 200;

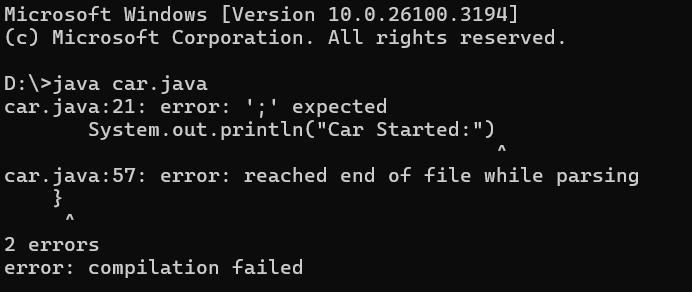
car3.service();

}

}

**Output:**

**Negative case:**

****

**Error table:**

|  |  |
| --- | --- |
| **Error** | **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.When we call a certain method, the process inside it will be printed as an output of the

code.

2.Here the details inside the function are called objects, we can give any objects

**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:**

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

**Code:**

public class BankAccount {

private String accountNumber;

private double balance;

public BankAccount(String accountNumber, double initialBalance) {

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.out.println("₹" + amount + " deposited. New balance: ₹" + balance);

} else {

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

}

}

public void withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

System.out.println("₹" + amount + " withdrawn. Remaining balance: ₹" + balance);

} else {

System.out.println("Invalid withdrawal amount!");

}

} public double getBalance() {

return balance;

}

public static void main(String[] args) {

BankAccount account = new BankAccount("8500", 500);

account.deposit(200);

account.withdraw(100);

System.out.println("Final balance: ₹" + account.getBalance());

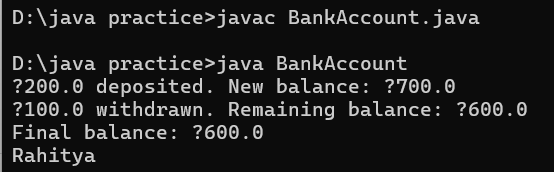
System.out.println(“Rahitya”)

    }

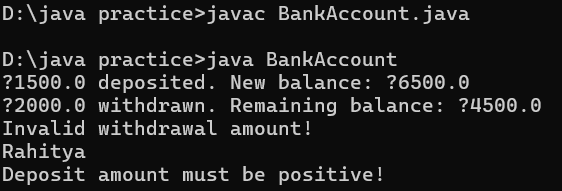
}

**Output:**

**Positive case:**

****

**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |
| --- | --- |
| **Error** | **Rectification** |
| 1.Giving space between next and double  2.Not putting the semi-colon; after calling the function. | 1.Should not give space between next and Double.  2.Put the semi-colon after 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.

**WEEK-3**

**-------------THE END -------------**

**WEEK 4:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **s.no** | **date** | **topic** | **p.no** | **signature** |
| **1** |  | WRITE A JAVA PROGRAM WITH CLASS NAMED “Book”. THE CLASS SHOUKD CONTAIN VARIOUS ATTRIBUTES SUCH AS TITLE, AUTHOR, YEAR OF PUBLICATION. IT SHOULD ALSO CONTAIN A CONSTRUCTOR WITH PARAMETERS WHICH INITIALIZES TITLE, AUTHOR, YEAR OF PUBLICATION AND CREATE A METHOD WHICH DISPLAYS THE DETAILS OF 2 BOOKS. |  |  |
| **2** |  | WRITE A JAVA PROGRAM WITH CLASS NAMED “MyClass” WITH A STATIC VARIABLE COUNT OF INT TYPE. INTIALIZE IT TO ZERO AND A CONSTANT VARIABLE “Pi” OF TYPE DOUBLE INITIALIZED TO “3.14” AS ATTRIBUTES OF THAT CLASS. NOW DEFINE A CONSTRUCTOR FOR “MyClass”, THAT INCREMENTS THE COUNT VARIABLE EACH TIME AN OBJECT OF “MyClass” IS CREATED. FINALLY, PRINT THE FINAL VALUES  OF ‘COUNT’ AND ‘PI’ VARIABLES AND CREATE 3 OBJECTS. |  |  |

**WEEK -4:**

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

**CLASS DIAGRAM:**

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

CODE:

public class Book {

public String title;

public String author;

public int year;

Book(String title, String author, int year) {

this.title = title;

this.author = author;

this.year = year;

}

public void displayDetails() {

System.out.println("Title: " +title);

System.out.println("Author: " +author);

System.out.println("Year of Publication" +year);

}

public static void main(String[] args) {

Book b1 = new Book("Maha Prasthanam", "Sri Sri", 1950);

Book b2 = new Book("Amrutham Kurisina Ratri", "Devarakonda Balagangadhara Tilak ", 1970);

b1.displayDetails();

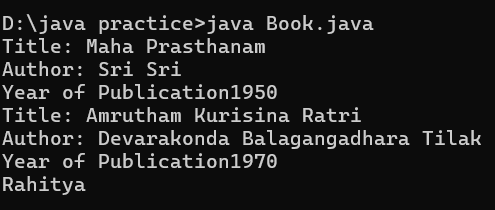
b2.displayDetails();

}

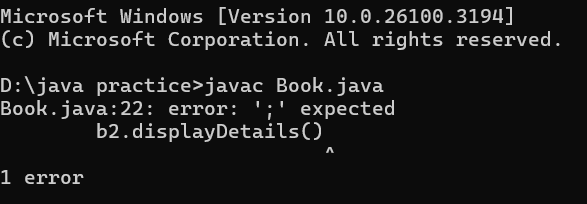
}

**Output:**

**Positive case:**

****

**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |
| --- | --- |
| **Error** | **Rectification** |
| Keeping wrong file name | Rename the file name |
| Not using semicolon after calling the function | Using semicolon |

**Important points:**

* we should write the method correctly before calling the function.
* The keyword **this** is used to differentiate between class attributes and constructor parameters.
* The method displayDetails() is used to display the book details.

2.AIM: WRITE A JAVA PROGRAM WITH CLASS NAMED “MyClass” WITH A STATIC VARIABLE COUNT OF INT TYPE. INTIALIZE IT TO ZERO AND A CONSTANT VARIABLE “Pi” OF TYPE DOUBLE INITIALIZED TO “3.14” AS ATTRIBUTES OF THAT CLASS. NOW DEFINE A CONSTRUCTOR FOR “MyClass”, THAT INCREMENTS THE COUNT VARIABLE EACH TIME AN OBJECT OF “MyClass” IS CREATED. FINALLY, PRINT THE FINAL VALUES

OF ‘COUNT’ AND ‘PI’ VARIABLES AND CREATE 3 OBJECTS.

Class Diagram:

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

CODE:

public class MyClass {

static int count = 0;

final double pi = 3.14;

public MyClass() {

count = count + 1;

}

public void display() {

System.out.println("Count is: " + count);

System.out.println("Double is: " + pi);

System.out.println();

}

public static void main(String[] args) {

MyClass Asec = new MyClass();

Asec.display();

MyClass Bsec = new MyClass();

Bsec.display();

System.out.println("The final count is: " + count);

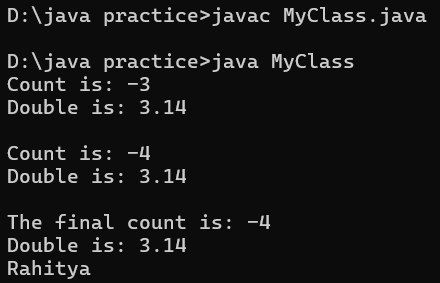
System.out.println("Double is: " + Bsec.pi);

System.out.println("Rahitya") ; }

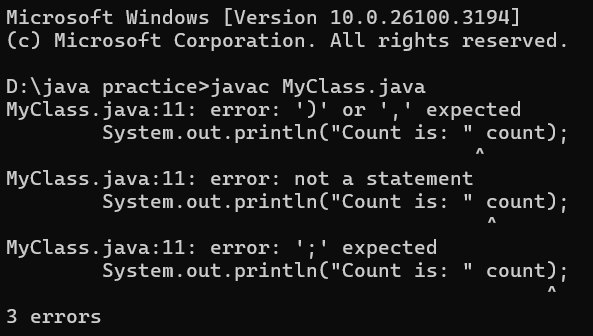
}

**OUTPUT:**

**Positive case:**

****

**Negative case:**

****

**ERROR:**

|  |  |
| --- | --- |
| **Error** | **Rectification** |
| Syntax error + is missing | Syntax error rectified |
| Incorrect path | Copied correct path |

**IMPORTANT POINTS:**

* Static members belong to the class, not to individual objects**.**
* It is initialized only once and not for every object.
* It increments every time the constructor is called.
* The **final** keyword makes the variable constant.

-------------- WEEK - 4 ------------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.NO | DATE | TOPIC | P:NO | SIGNATURE |
| 1 |  | Create a calculator using the operations including addition, subtraction  Multiplication and division using multilevel inheritance and display the desired output**.** |  |  |
| 2 |  | A vehicle rental company wants to develop a system that maintains information about different types of vehicles available for rent. The company rents out cars and bikes, and they need a program to store details about each vehicle, such as brand and speed.   1. Cars should have an additional property: number of doors, seating capacity. 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**.**   Questions   1. Which OOP concept is used in the above program? Explain why it is useful in this scenario? 2. If the company decides to add a new type of vehicle: Truck, how would you modify the program?    1. Truck should include an additional property: capacity (in tons).    2. Create a showTruckDetails() method to display the truck's capacity.    3. Write a constructor for truck that initializes all properties. 3. Implement the Truck class and update the main method to create a Truck object and also create an object for car and bike subclasses. Properly display its details. |  |  |

**WEEK-5:**

**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.

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

Class Diagram:

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

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

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

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

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 add() {

return a + b;

}

}

class Subtraction extends Addition {

public Subtraction(double a, double b) {

super(a, b);

}

public double subtract() {

return a - b;

}

}

class Multiplication extends Subtraction {

public Multiplication(double a, double b) {

super(a, b);

}

public double multiply() {

return a \* b;

}

}

class Division extends Multiplication {

public Division(double a, double b) {

super(a, b);

}

public double divide() {

if (b != 0) {

return a / b;

} else {

System.out.println("Error");

return Double.NaN;

}

}

}

class Final extends Division {

public Final(double a, double b) {

super(a, b);

}

public void displayResults() {

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

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

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

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

}

}

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("Rahitya");

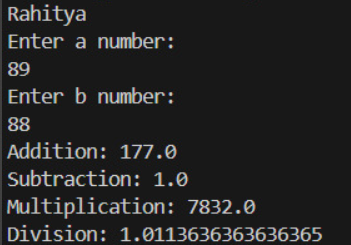
input.close();

}

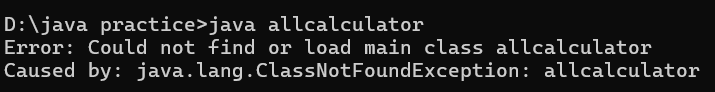
}

OUTPUT:

Positive case:



Negative case:



ERROR TABLE:

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1.Declaring two superclasses inside the same file. | 1. Make two separate files to save the two super classes. |
| Error in final calc | Rectified the error in the final calc |

IMPORTANT POINTS:

1. Scanner class is used to get the user input.
2. To get the inputs from the user we use import java.util.Scanner; this is a package.
3. in java.util.Scanner, the java.util is a package while Scanner is a class of the java.util package.

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.

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

Class diagram:

Bike Truck

Car

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

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

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

, no.of.doors) ,has gears)

+displaydetails() +displaydetails()

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();

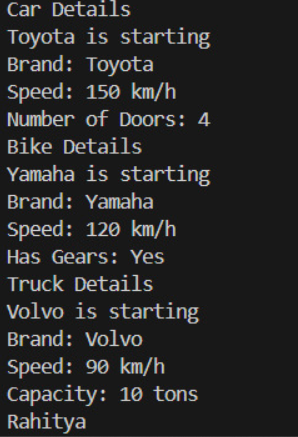
System.out.println("Capacity: " + capacity + " tons");

}

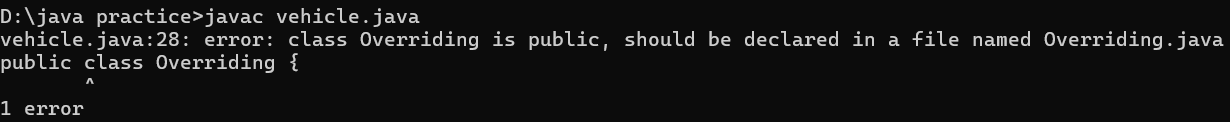
}

OUTPUT:

Positive case:



Negative case:



ERROR TABLE:

|  |  |
| --- | --- |
| ERROR | RECTIFICATION |
| 1.Super is not mentioned in super class constructor.  2.Inside the constructor ‘this’ is not used | 1.To get the super class we need to use super.  2. using the variable ‘this’ to run the program. |

IMPORTANT POINTS:

1.a constructor helps in initializing an object that doesn't exist.

2. 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-5---------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.no | date | topic | p.no | signature |
| 1 |  | 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 a About car. |  |  |
| 2 |  | A college is developing an automated admission system that verifies Student eligibility for UG and PG programs. Each program has different Eligibility criteria based on the student’s percentage in their previous  Qualification:  UG admissions require a minimum of 60%  PG admissions require a minimum of 70% |  |  |
| 3 |  | 3.Create a calculator class with overloaded methods to perform addition   1. Add two integers 2. Add two doubles 3. Add three integers |  |  |
| 4 |  | Create a Shape class with a method calculateArea() that is overloaded for Different shapes. Then, create a subclass circle that overrides the calculateArea() method for a circle |  |  |

WEEK-6

**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 a About car.**

**Class diagram:**

|  |
| --- |
| Vehicle  -------------------------------------------- car\_model  -car\_company  -car\_price  +displayinfo() |



|  |
| --- |
| Car  ---------------------------------------(inherited)  + displayinfo() |

**CODE:**

class vehicle

{

public String car\_model;

public String car\_company;

public int car\_price;

void displayinfo()

{

System.out.println("Rahitya Car Agency");

}

}

class car extends vehicle

{

void displayinfo()

{

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

System.out.println("Car Company is:"+car\_company);

System.out.println("Car price is:"+car\_price);

}

}

class Overriding

{

public static void main(String[] args)

{

car obj1 = new car();

obj1.car\_model="BMW M5";

obj1.car\_company="BMW";

obj1.car\_price= 20000000;

obj1.displayinfo();

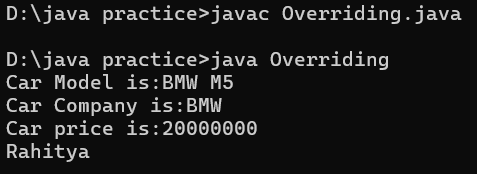
System.out.println("Rahitya");

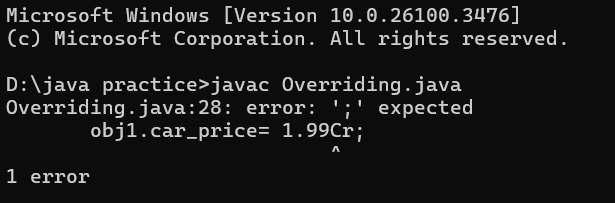
}

}

Output:

Positive case:



NEGATIVE CASE:

Error table:

|  |  |
| --- | --- |
| ERROR | RECTIFICATION |
| Syntax error | semicolon |
| logic incorrect | Correct logic |



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

**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.

**2.A college is developing an automated admission system that verifies Student eligibility for UG and PG programs. Each program has different Eligibility criteria based on the student’s percentage in their previous**



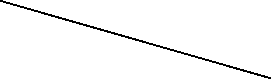
**Qualification:**

**UG admissions require a minimum of 60%**

**PG admissions require a minimum of 70%**

**Class diagram:**

|  |
| --- |
| Student  -name  -percentage  +Student (name, percentage)  +eligibility() |



|  |
| --- |
| **-(inherited)**  **--------------------------------**  -(inherited)  +PGStudent(name,percentage)  +eligibility() |

|  |
| --- |
| UG student  -( inherited)  + UG student(name,  Percentage)  + eligibility() |

**CODE:**

class Student {

public String name;

public double percentage;

Student(String name, double percentage) {

this.name = name;

this.percentage = percentage;

}

void eligibility() {

System.out.println("Amrita Vishwa Vidyapeetham");

}

}

class UGStudent extends Student {

UGStudent(String name, double percentage) {

super(name, percentage);

}

@Override

void eligibility() {

if (percentage > 60) {

System.out.println(name + " is selected to the college");

} else {

System.out.println(name + " is not matching the required criteria");

}

}

}

class PGStudent extends Student {

PGStudent(String name, double percentage) {

super(name, percentage);

}

@Override

void eligibility() {

if (percentage > 70) {

System.out.println(name + " is selected to the college");

} else {

System.out.println(name + " is not matching the required criteria");

}

}

}

public class School {

public static void main(String[] args) {

PGStudent obj1 = new PGStudent("Rahi", 90.0);

UGStudent obj2 = new UGStudent("Sumi", 80.0);

obj1.eligibility();

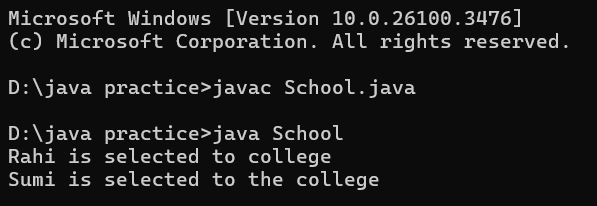
obj2.eligibility();

}

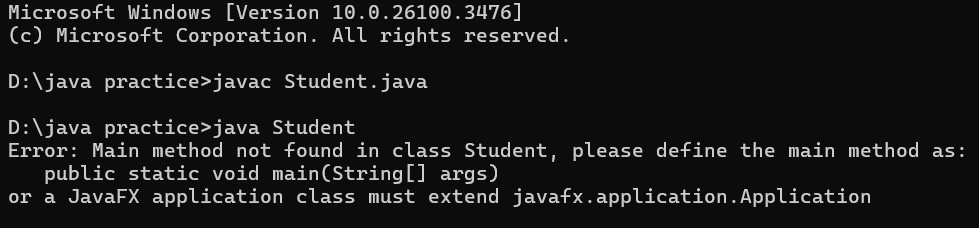
}

Output:

Positive case



Negative case:



ERROR TABLE:

|  |  |
| --- | --- |
| Error | Rectification |
| Giving wrong class name to file | Rectified the error |
| Not kept string in main class | String is added |

1. Important points: String Handling: converts program input to uppercase(toUpperCase()) to handle case variations.
2. Closing input: properly closes scanner using scanner.close(); to prevent resource leaks.

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



1. **Add two integers 2. Add two doubles 3. Add three integers**

**Class diagram:**

|  |
| --- |
| Calculator  ---------------------------------  +add(int,int): int  +add(double,double): double  +add(int,int,int):int |

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

**CODE:**

class Calculator {

public int add(int a, int b) {

return a + b;

}

public double add(double a, double b) {

return a + b;

}

public int add(int a, int b, int c) {

return a + b + c;

}

}

public class Overloading {

public static void main(String[] args) {

Calculator c = new Calculator();

System.out.println("Addition of 2 and 6 is: " + c.add(2, 6));

System.out.println("Addition of 1.1 and 9.9 is: " + c.add(1.1, 9.9));

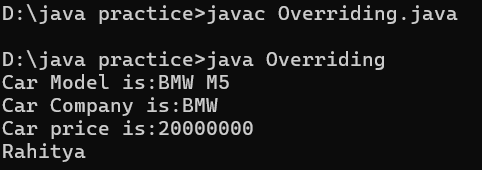
System.out.println("Addition of 3, 6, and 9 is: " + c.add(3, 6, 9));

System.out.println("Rahitya");

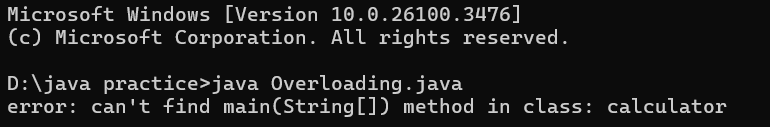
}

}

Output:



Negative:



ERROR Table:

|  |  |
| --- | --- |
| Error | Recification |
| Logic error in main class | Added main (string[]) method in class |
| Syntax error wrong print statement | Give correct statement |

Important points:

We use the concept of method overloading where the names of the methods in the same class are same but the parameters are given different.

**4. Create a Shape class with a method calculateArea() that is overloaded for Different shapes. Then, create a subclass circle that overrides the calculateArea() method for a circle**

**Class diagram:**

|  |
| --- |
| Shape  +calculateArea(double):double  +calculateArea(int, int):int |



|  |
| --- |
| Circle  ------------------------------------------  +calculateArea(double):double |

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

**CODE:**

class Shape {

public double calculateArea(double side) {

return side \* side;

}

public int calculateArea(int length, int width) {

return length \* width;

}

}

class Circle extends Shape {

public double calculateArea(double radius) {

return 3.14 \* radius \* radius;

}

}

public class Example {

public static void main(String[] args) {

Circle Obj1 = new Circle();

Shape Obj2 = new Shape();

System.out.println("The area of side 4 is:"+Obj2.calculateArea(2.0));

System.out.println("The area of circle with radius 6 :"+Obj1.calculateArea(4.0));

System.out.println("The area of length 5 and width 2 is:"+obj2.calculateArea(3,2));

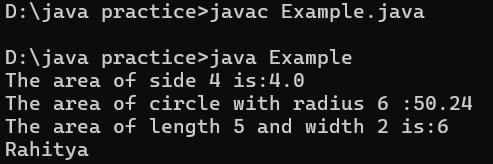
System.out.println("Rahitya");

}

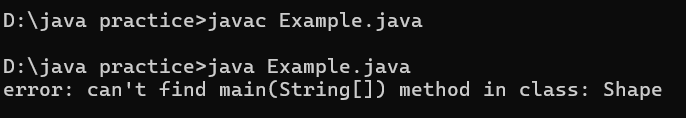
}

Output:

Positive case



Negative case:



ERROR TABLE:

|  |  |
| --- | --- |
| Error | Rectification |
| In class example and print statement Obj name should be same | Case sensitive error is rectified |
| Given class name as file name | Given main class name |

Important point:

1. We use the concept of method overloading to calculate the area of square and rectangle in the parent class Shape.

2 we use method overriding in the child class Circle to calculate it’s area.

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

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

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

------------WEEK-6-------------

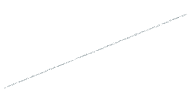
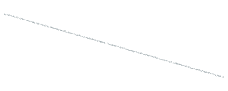
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.NO | Date | TOPIC | p.no | signature |
| 1 |  | Write a java program to create an abstract class Animal with an abstract method called sound(). Create subclasses lion and tiger that extent the Animal class and implement the sound() method to make a specific for each animal |  |  |
| 2 |  | Write a java program to create an abstract class shape 3D with abstract method called sound().Create subclass sphere and cude that extend the shape 3D class and implement |  |  |
| 3 |  | 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:  • StarPattern – prints a right-angled triangle of stars  • 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 |  |  |

**WEEK-7**

AIM: Write a java program to create an abstract class Animal with an abstract method called sound(). Create subclasses lion and tiger that extent the Animal class and implement the sound() method to make a specific for each animal.

Class diagram:

|  |
| --- |
| Animals  ------------------------  +eat() |



|  |
| --- |
| Omnivores  -----------------  +eat() |

|  |
| --- |
| Herbivores  -----------------  +eat() |

|  |
| --- |
| Carnivores  -----------------  +eat() |

|  |
| --- |
| AnimalsEat  ---------------------  +main(String[]) |

Code:

class Animals {

void eat() {

System.out.println("Animals are eating");

}

}

class Herbivores extends Animals {

@Override

void eat() {

System.out.println("They eat plants and grass");

}

}

class Carnivores extends Animals {

@Override

void eat() {

System.out.println("They eat meat and flesh of other animals ");

}

}

class Omnivores extends Animals {

@Override

void eat() {

System.out.println("They eat both plants and meat");

}

}

public class AnimalsEat {

public static void main(String[] args) {

Animals a1 = new Animals();

a1.eat();

Herbivores H1 = new Herbivores();

H1.eat();

Carnivores C1 = new Carnivores();

C1.eat();

Omnivores O1 = new Omnivores();

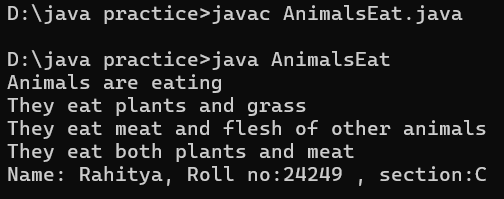
O1.eat();

System.out.println("Name: Rahitya, Roll no:24249 , section:C");

}

}

OUTPUT:



Error table:

|  |  |
| --- | --- |
| Error | 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 use the concept of abstraction and writing abstract classes.

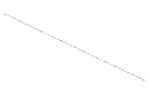
2.we also use method overrriding concept.

3. Here we are using the heirarchial inheritance.

Aim: Write a java program to create an abstract class shape 3D with abstract methods. Calculate Volume( )and calculate Surface area ( ).Create Subclasses Sphere and cube that extend the Shape 3D class and implement the respective method and to calculate the volume and surface area of each shape.

Class diagram:

|  |
| --- |
| <<abstract>>  Shape 3D  +calculateVolume()  +calculateSurfaceArea() |



|  |
| --- |
| Cube  -----------------------  -edge:int  +calculateVolume()  +calculateSurfaceArea() |



|  |
| --- |
| Sphere  ------------------------  -radius:int  + calculateVolume()  +calculateSurfaceArea() |

|  |
| --- |
| Assignment1  -------------------------  +main(String[]) |

CODE:

abstract class Shape3D {

public abstract double calculateVolume();

public abstract double calculateSurfaceArea();

}

class Sphere extends Shape3D {

int radius;

Sphere(int radius) {

this.radius = radius;

}

@Override

public double calculateVolume() {

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

}

@Override

public double calculateSurfaceArea() {

return 4 \* Math.PI \* Math.pow(radius, 2);

}

}

class Cube extends Shape3D {

int edge;

Cube(int edge) {

this.edge = edge;

}

@Override

public double calculateVolume() {

return Math.pow(edge, 3);

}

@Override

public double calculateSurfaceArea() {

return 6 \* Math.pow(edge, 2);

}

}

public class Assignment1 {

public static void main(String[] args) {

Sphere s = new Sphere(5);

System.out.println("Volume of Sphere of radius 5 : " + s.calculateVolume());

System.out.println("Surface Area of Sphere of radius 5 : " + s.calculateSurfaceArea());

Cube c = new Cube(7);

System.out.println("Volume of Cube of edge 7: " + c.calculateVolume());

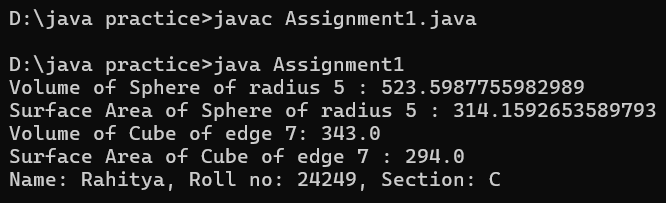
System.out.println("Surface Area of Cube of edge 7 : " + c.calculateSurfaceArea());

System.out.println("Name: Rahitya, Roll no: 24249, Section: C");

}

}

Output:



Error table:

|  |  |
| --- | --- |
| Error | 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:

1.We use the concept of abstraction and writing abstract classes.

2.we also use method overrriding concept.

3. Abstract classes and methods help us to declare the methods without declaring the return type in 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:

• StarPattern – prints a right-angled triangle of stars

• 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

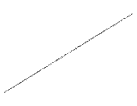
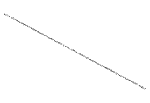
1 2 3 4

1 2 3 4 5

|  |
| --- |
| <<abstract>> PatternPrinter  ------------------------rows:int  -----------------------  +PatternPrinter(int) + printPattern(): void  +displayTitle(String): void |

Class diagram:





|  |
| --- |
| NumberPattern  -------------------------  +NumberPattern(int)  + printPattern() |

|  |
| --- |
| StarPattern  -----------------------  + StarPattern(int)  + printPattern() |

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

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) {

int numberOfRows = 5;

PatternPrinter star = new StarPattern(numberOfRows);

star.displayTitle("Star Pattern");

star.printPattern();

PatternPrinter number = new NumberPattern(numberOfRows);

number.displayTitle("Number Pattern");

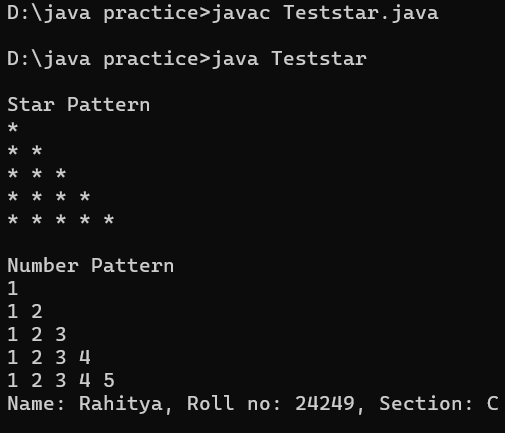
number.printPattern();

System.out.println("Name: Rahitya, Roll no: 24249, Section: C");

}

}

Output:



Error table:

|  |  |
| --- | --- |
| Error | Recification |
| 1) Class name and file name should match  2) Subclass doesn’t override abstract method | 1) Save file as main.java  2)implement printpattern()in all subclasses |

Important points:

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

2. PatternPrinter is the abstract class defining the common template.

3. Subclasses (StarPattern, NumberPattern) provide specific implementations. 4. displayTitle() is a concrete method shared by all subclasses.

-----------------WEEK-7-------------------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S:NO | DATE | TOPIC | PAGE NO | SIGNATURE |
| 1 |  | 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 |  |  |
| 2 |  | 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 |  |  |

WEEK 8:

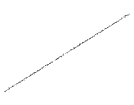
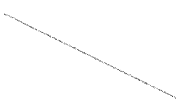
AIM:Write a Java program to create an interface Shape with the getPerimeter() method. Create three classes Rectangle, Circle, and Triangle that implement the Shape interface. Implement the getPerimeter() method for each of the three classes.

|  |
| --- |
| <<interface>>  Shape  ------------------------  + getPerimeter(): double |

Class diagram:

|  |
| --- |
| Rectangle  ------------------------  - length: double  - width: double  -------------------------  + getPerimeter() |





|  |
| --- |
| Circle  -----------------------  - radius: double  + getPerimeter() |

|  |
| --- |
| Triangle  ------------------------  - a: double  - b: double  - c: double |

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

Code:

interface Shape {

double getPerimeter();

}

class Rectangle implements Shape {

double length;

double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

public double getPerimeter() {

return 2 \* (length + width);

}

}

class Circle implements Shape {

double radius;

public Circle(double radius) {

this.radius = radius;

}

public double getPerimeter() {

return 2 \* Math.PI \* radius;

}

}

class Triangle implements Shape {

double a;

double b;

double c;

public Triangle(double a, double b, double c) {

this.a = a;

this.b = b;

this.c = c;

}

public double getPerimeter() {

return a + b + c;

} }

public class Perimeter {

public static void main(String[] args) {

Shape rectangle = new Rectangle(5.7, 10.3);

Shape circle = new Circle(7);

Shape triangle = new Triangle(3.09, 4.57, 5.87);

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

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

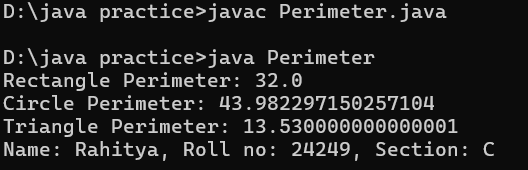
System.out.println("Triangle Perimeter: " + triangle.getPerimeter());

System.out.println("Name: Rahitya, Roll no: 24249, Section: C");

}

}

Output:



Error table:

|  |  |
| --- | --- |
| Error | Rectification |
| Not giving interface keyword to the Shape class | Rectifying the error by giving interface keyword |

Important points:

1. Interfaces in Java

• The Shape interface defines a contract for all shapes that must implement getPerimeter().

• Promotes loose coupling and flexibility in design.

• Interfaces allow multiple classes to follow a common behavior without using inheritance.

2. Polymorphism (Runtime Polymorphism)

• You’re using interface references (Shape rectangle = new Rectangle(...)) to refer to different

shape objects.

• This demonstrates dynamic method dispatch, where the appropriate getPerimeter() is called

based on the actual object type at runtime.

3. Encapsulation

• Each class has private data (length, width, radius, sides), and the behavior (getPerimeter()) is

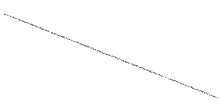
encapsulated inside the class.

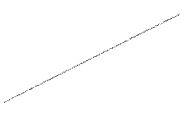
• This keeps data and behavior bundled together.

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

|  |
| --- |
| <<interface>> Playable  -------------------------  + play(): void |

Class diagram:





|  |
| --- |
| Football  --------------------------  + play(): void |

|  |
| --- |
| Basketball  + play(): void |

|  |
| --- |
| Volleyball  ----------------------------    + play(): void |

|  |
| --- |
| Assignment2  ------------------------------  + main(String[]): void |

Code:

interface Playable {

void play();

}

class Football implements Playable {

@Override

public void play() {

System.out.println("11 players plays Football on each team, kicking the ball to score goals.");

}

}

class Volleyball implements Playable {

@Override

public void play() {

System.out.println("6 players on each side plays Volleyball.");

System.out.println("Teams hit the ball over a net to score points.");

}

}

class Basketball implements Playable {

@Override

public void play() {

System.out.println("10 members play basketball.") ;

System.out.println("They Dribble and shoot the ball into the opponent's hoop.");

}

}

public class Assignment2 {

public static void main(String[] args) {

Playable football = new Football();

Playable volleyball = new Volleyball();

Playable basketball = new Basketball();

football.play();

volleyball.play();

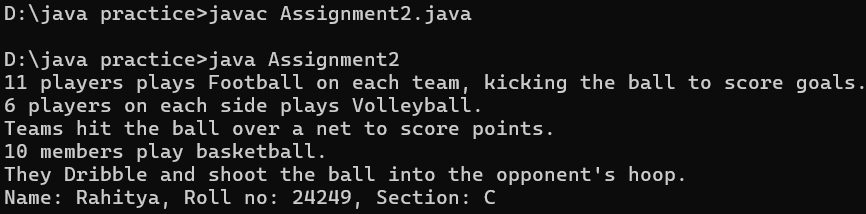
basketball.play();

System.out.println("Name: Rahitya, Roll no: 24249, Section: C");

}

}

Output:



Error table:

|  |  |
| --- | --- |
| Error | 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. Interfaces

• Playable is an interface that declares a play() method, which serves as a contract for any class

that implements it.

• Interfaces allow you to define methods without providing the implementation, leaving it to

the implementing class to define the behavior.

2. Implementation of Interface Methods

• Football, Volleyball, and Basketball are classes that implement the Playable interface. Each

class overrides the play() method and provides its own specific implementation.

o Football: Describes how the game is played with 11 players on each team, aiming to

score goals by kicking the ball.

o Volleyball: Describes a 6-player side game where teams try to hit the ball over the

net to score points.

o Basketball: Describes a 10-player game where players dribble and shoot the ball into

the opponent's hoop.

3.Polymorphism

• The program demonstrates runtime polymorphism because:

o The reference variable Playable is used for different game objects (Football,

Volleyball, Basketball), but the appropriate play() method is called dynamically at

runtime.