**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**LAB REPORT**



**Department of Computer Science Engineering**

**Amrita School of Computing**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Name: B.SAI RISHITHA**

**Roll: AV.SC.U4CSE24016**

**Branch: CSE**

**Section: A**

**Verified by:**

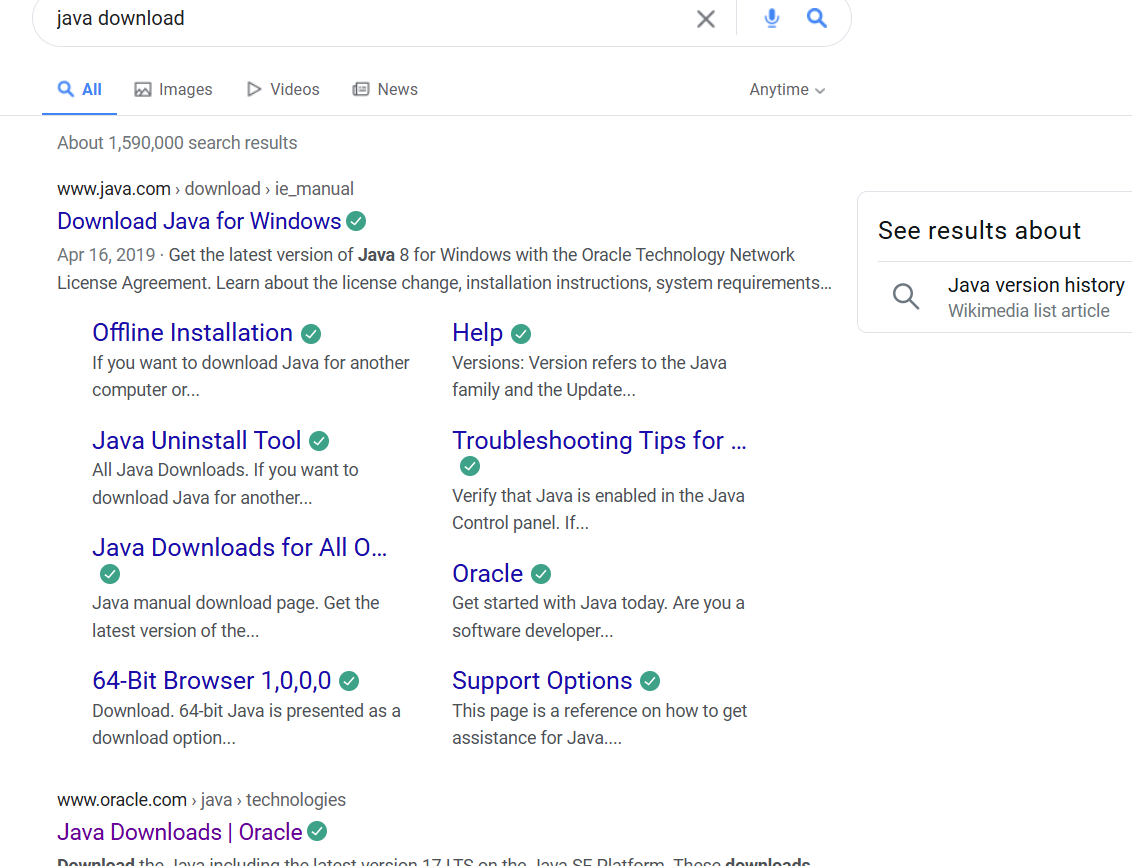
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No. | Title | Date | Page No. | Signature |
| Week  1 |  | 27-01-  2025 |  |  |
| 1. | How to download and install Java Software. |  |  |  |
| 2. | Write a Java Program to print the message  “Welcome to Java Programming”. |  |  |  |
| 3. | Write a Java program that prints: Name, Roll.no. , section of a student. |  |  |  |
| Week  2 |  | 10-02-  2025 |  |  |
| 1. | Write a java program to calculate the area of a rectangle. |  |  |  |
| 2. | Write a java program to temperature from Celsius to Fahrenheit and vica-versa. |  |  |  |
| 3. | Write a java program to calculate the simple interest. |  |  |  |
| 4. | Write a java program to find the largest of three numbers, using ternary operator. |  |  |  |
| 5. | Write a java program to find the factorial of a number. |  |  |  |
| Week  3 |  | 24-02-2025 |  |  |
| 1. | To create a java program with the following instructions:   1. Create a class with name “Car” 2. Create 4 attributes, named: car\_color, car\_brand, fuel\_type, mileage 3. Create 3 methods, named: start(), service(), stop() 4. Create 3 objects, named: car1, car2, car3 5. Create a constructor, which should print,   “Welcome to car  Garage”. |  |  |  |
| 2. | To write a java program to create a class named BankAccount, with 2 methods deposit() and withdraw().   1. deposit(): Whenever an amount is deposited, it has to be update the current amount. 2. withdraw():   Whenever an amount is withdraw,it has to be less than the current amount, else print(“Insufficient funds”) |  |  |  |
| Week  4 |  |  |  |  |
| 1. | Write a java program with class named “Book”.The class should contain various attributes “title of the book”, “author”, “year of publication”.It should also contain a constructor with parameters which initializes “title of the book”, “author”, “year of publication”. Create a method which displays the details of the book i.e  “title of the book”, “author”, “year of publication”.Display the details of two books by creating two objects. |  |  |  |
| 2. | To create a java program with class named myclass with a static variable count of “int” type initialize to zero and a constant variable “pi” of type double initialized to 3.1415. As attributes of that class now define a constructor for my class that increments the count variable each time an object of my class is created finally print hr final values of “count” and “pi” variables. |  |  |  |
| Week 5 |  |  |  |  |
| 1. | Create a calculator using the operations including addition,sub,mul,and division using multilevel inheritance display the desired output.[3 levels of calculators] |  |  |  |
| 2. | A vehicle rental company wants to devlop 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 additionalproperties no.of doors, seating capacity 2. Bikes should have a property indicating whether they have gares or not(yes or no) 3. The system should also include a function to display (details) about each vehicle and indicate when a vehicle is starting. 4. Each class should have a constructor 5. Which object oriented programming concept used in the above program example why it is used in the scenario? 6. If the company decides to add a new type of vehicle truck how would you modify the above program  * truck should include an additional property called capacity(in tones) * create and show truck details method to display the trucks capacity * write a constructor for truck that initializes all properties. |  |  |  |
| 3. | Impliment the truck class and update the main class to create truck object and also create an object for car and bike subclass.Finally display its details. |  |  |  |
| Week  6 |  |  |  |  |
| 1. | Write a Java program to create a vehicle class with a method displayInfo() override this method in the car subclass to provide specific info about a car. [car\_company, car\_model, car\_prize, seating\_capacity, petro\_or\_not 🡪boolean] |  |  |  |
| 2. | A college is developing an automated admission system that verifies student’s 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 require: min 60% * PG admission require: min 70% |  |  |  |
| 3. | Create a calculator class with overloaded methods to perform addition:   1. Add 2 int 2. Add 2 doubles 3. Add 3 int |  |  |  |
| Week  7 |  |  |  |  |
| 1. | Write a java program to create an abstract class Animal with an abstract method called sound() .Create sub classes Lion and Tiger that extend the Animal class and implements the sound() method to make a specific sound for each animal. |  |  |  |
| 2. | Write a java program to create an abstract class called Shape 3D with abstract methods calculate volume and calculate surfacearea()  Create subclasses Sphere and Cube that extend the Shape 3D class and implement the respective methods to calculate the volumn and surface area of each shape. |  |  |  |
| 3. | Write a java program using an abstract class to define a method for pattern printing   * Create an abstract class named PtternPrint 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. |  |  |  |
| Week  8 |  |  |  |  |
| 1. | Write a java program to create an interface Shape with the getPerimeter () method. Create three classes Rectangle, Circle , and Triangle that implements the Shape interface. 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 return void. Create three classes Football, Volleyball, and Basketball that implements the playable interface and override method to display score. |  |  |  |
| 3. | Write a java program to implement a login system using interfaces. |  |  |  |

**Week-1**

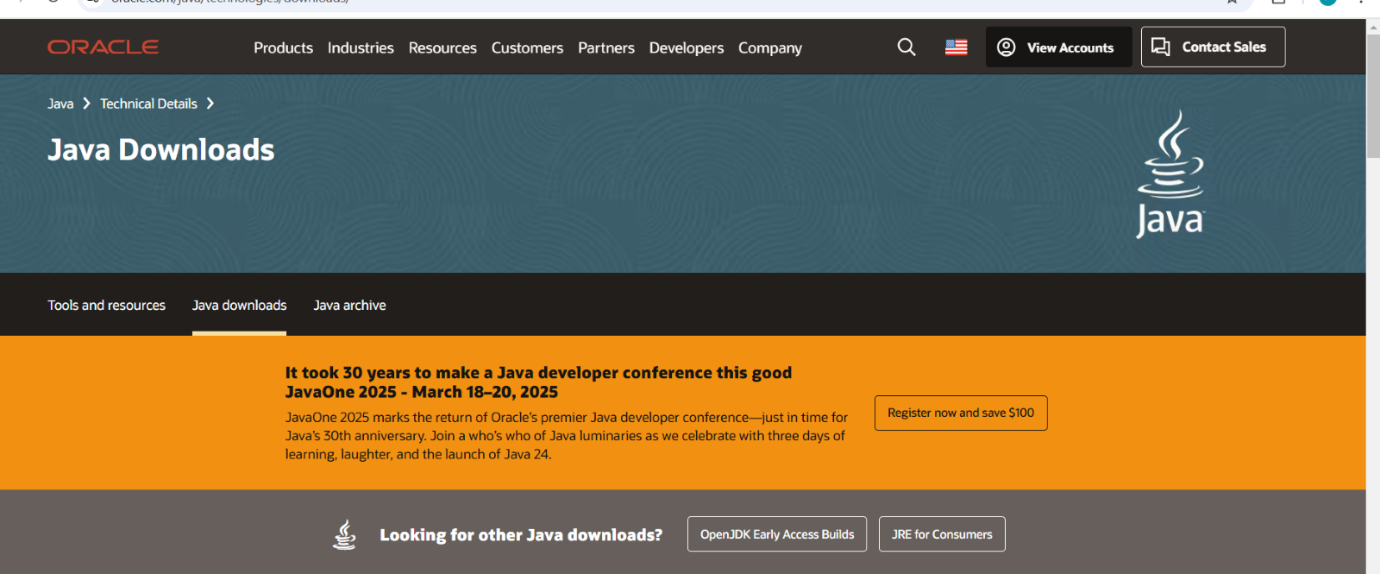
**TASK-1**

**AIM:** How to download and install JAVA software.

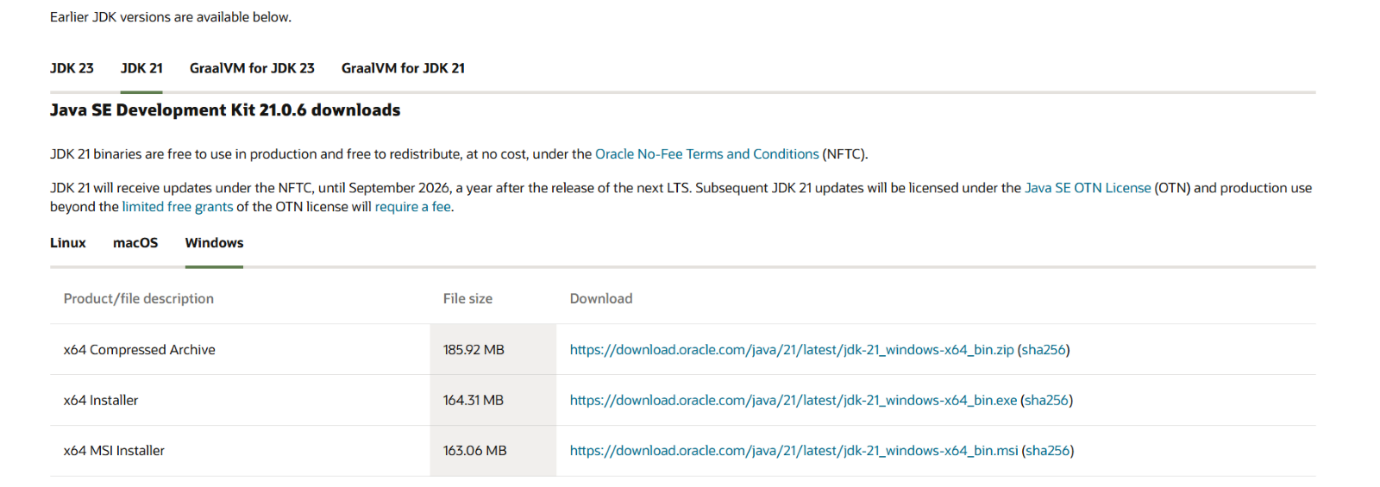
**PROCEDURE:**STEP 1: Go to browser and search for java download



Step2: Open website link with oracle



Step3: Click on JDK 21 and click on windows



Step4: After that click on x64 installer

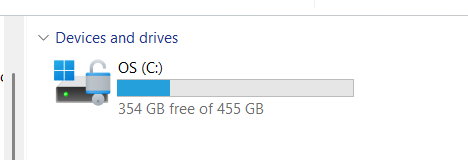
Step5: The download starts

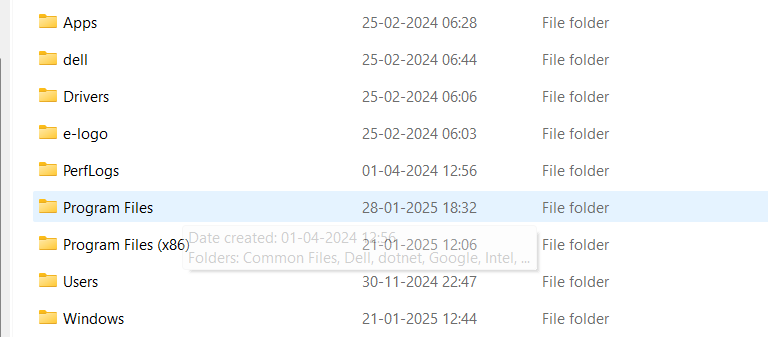
Step6: After downloading the next step installation starts



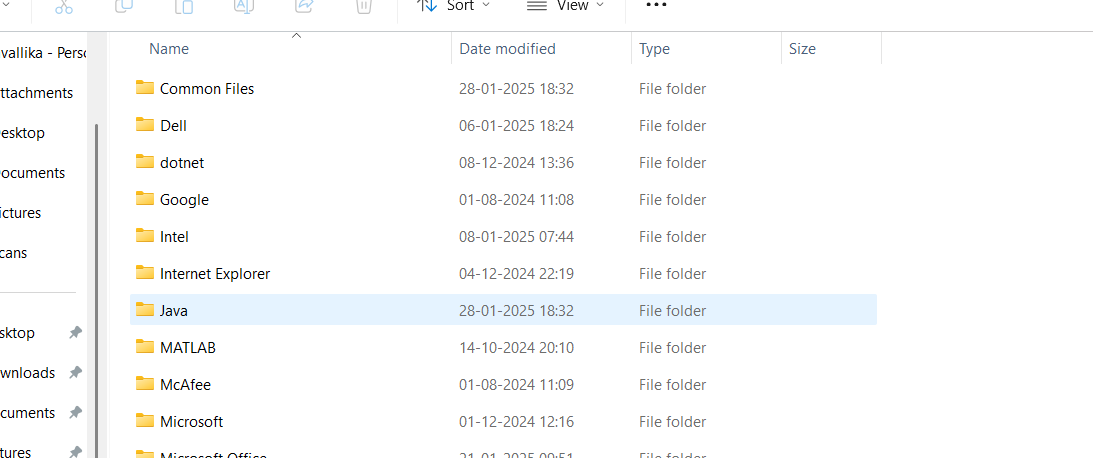
Step7: After installation we need to set a path

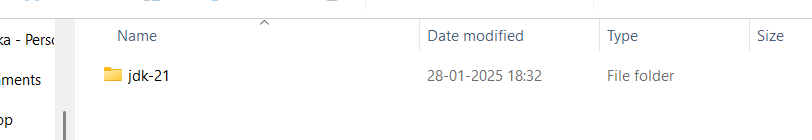
Step8: Go to “c” drive in c drive we must the program files



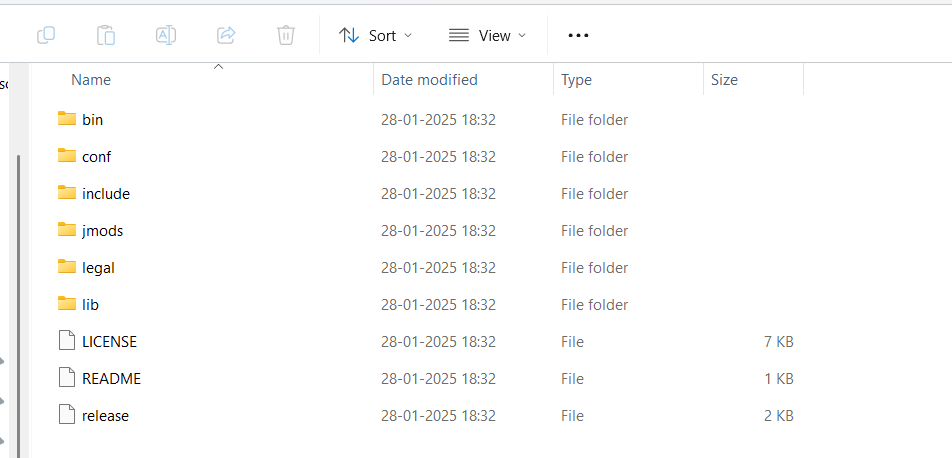


Step 9: Double click on the program files and double click on the java





 Step 10: Click on JDK-21 and select bin



Step11: Copy the link and paste the environment

Step12: Select the system variables and select the path

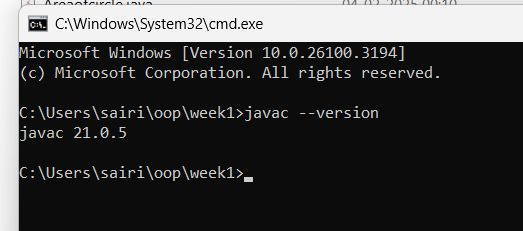
STEP13: After selecting the path we need select new one need to paste the link

Step14: After pasting the link we need to click the ok and the apply

Step15: In this way the downloading and installation of java takes place

Step16: After installation go to command prompt and type javac –version

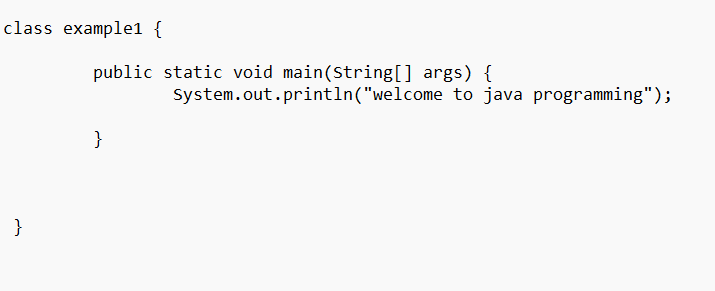
Step17: In command prompt we get the version which we download.



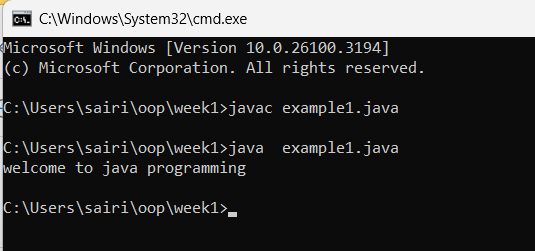
**TASK-2**

**AIM:** Write the java program to print the message “welcome to java programming “

**CODE:**



**OUTPUT:**

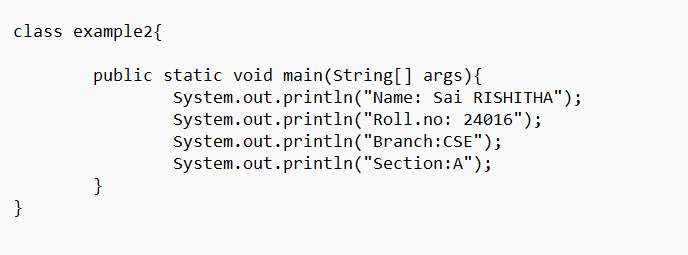


**ERRORS:** Not found

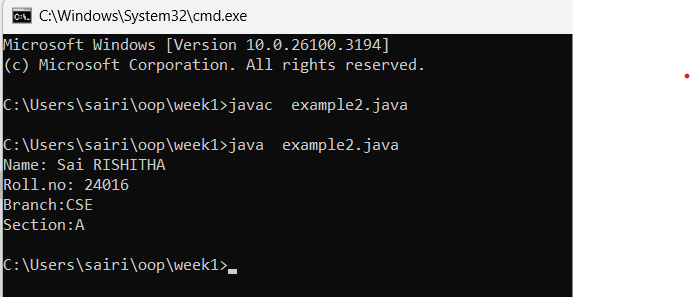
**TASK-3:**

**AIM:** write the java program to print the message “Name, class, roll.no, section”.

**CODE:**



**OUTPUT:**



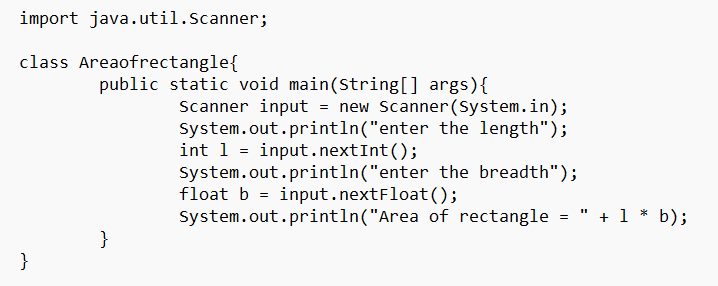
**ERROR:** Not found

**Week-2**

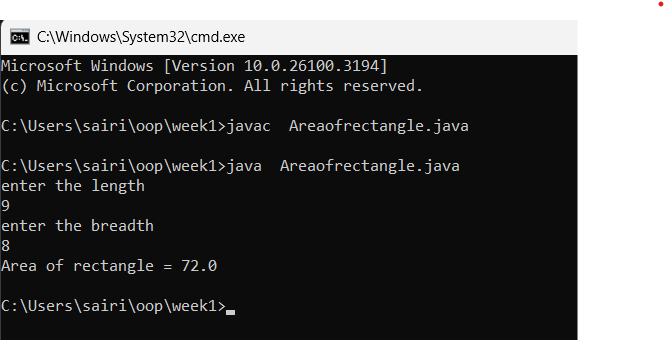
**TASK-1**

**AIM:** write a java program to calculate the area of the rectangle**.**

**INPUT CODE:**

****

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.No: | Errors | Error rectification |
| 1) | cannot find symbol: class Scanner | Import  java.util.Scanner at the beginning of the |
|  |  | code: import java.util.Scanner; | |
| 2) | Scanner input = new Scanner(System.in); | Ensure the correct spelling and case of Scanner. No change is needed after import | |

**IMPORTANT POINTS:**

1)import java.util.Scanner; - To accept input from user, Scanner class under util package has to be imported.

2)Scanner input=new Scanner(System.in); - Used to create a Scanner object

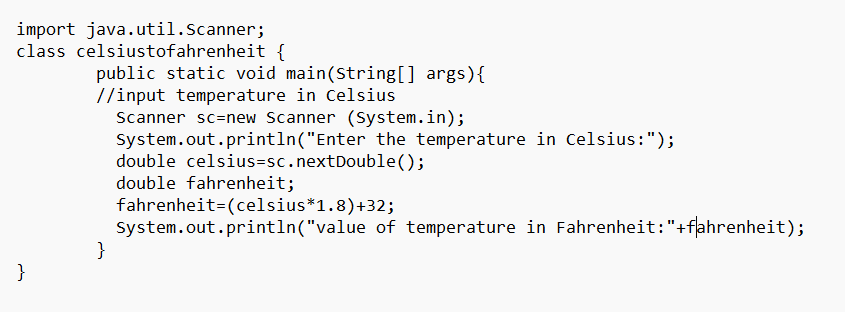
3)int ln=input.nextInt(); - Used to read the integer data type stored under the object created

4)System.out.println(“ “); - It is used to print string inside the quotes. After printing, the cursor moves to the beginning of the next line.

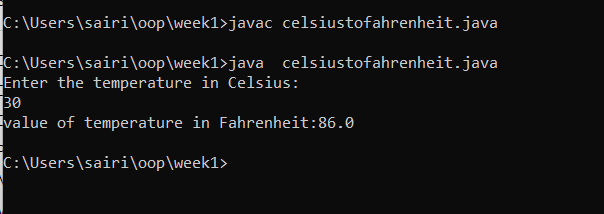
**TASK-2:**

**AIM:** write a java program to convert temperature from Celsius to Fahrenheit.

**INPUT CODE:**

****

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Error Rectification |
| 1 | variable tp is already defined in method main(String[]) | Rename the first and second tp variable to avoid duplication. Variable names must be unique within the same scope. |

**IMPORTANT POINTS:**

1)import java.util.Scanner; - To accept input from user, Scanner class under util package has to be imported.

2)Scanner input=new Scanner(System.in); - Used to create a Scanner object

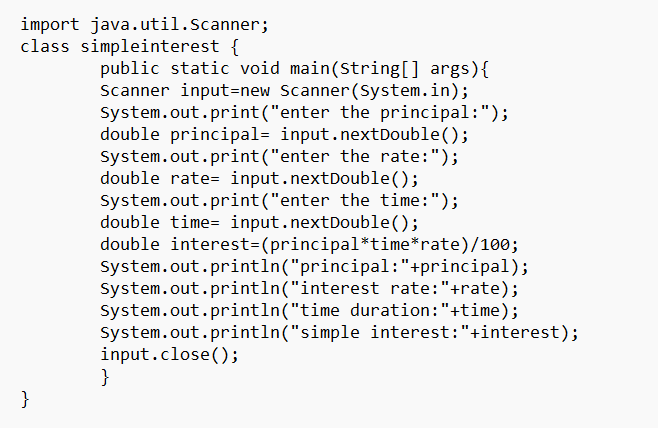
3)double fh=input.nextDouble(); - Used to read double data type stored under the object created

4)System.out.println(“ “); - It is used to print string inside the quotes. After printing, the cursor moves to the beginning of the next line.

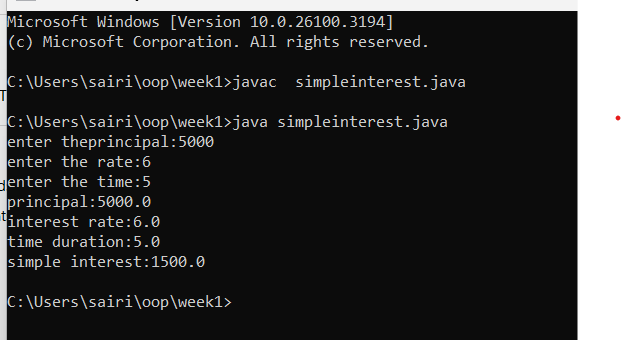
**TASK-3**

**AIM:** write a java program to calculate the simple interest

**CODE:**

****

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | error: ';' expected  double intr=(p\*r\*t)/100 | Add a semicolon at the end of the statement double intr=(p\*r\*t)/100; |
| 2. | error: cannot find symbol  double intr=(p\*r\*t)/100; symbol: variable p location: class interest | Create a reader object double  p=input.nextDouble(); |

**IMPORTANT POINTS:**

1)import java.util.Scanner; - To accept input from user, Scanner class under util package has to be imported.

2)Scanner input=new Scanner(System.in); - Used to create a Scanner object

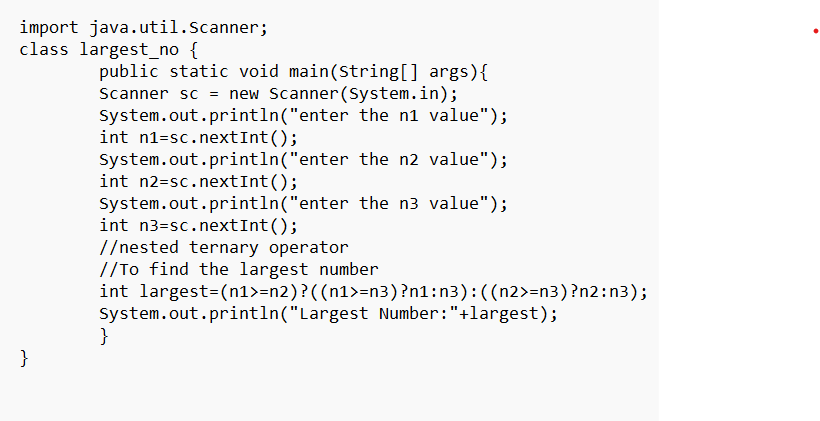
3)double p=input.nextDouble(); - Used to read double data type stored under the object created

4)System.out.println(“ “); - It is used to print string inside the quotes. After printing, the cursor moves to the beginning of the next line.

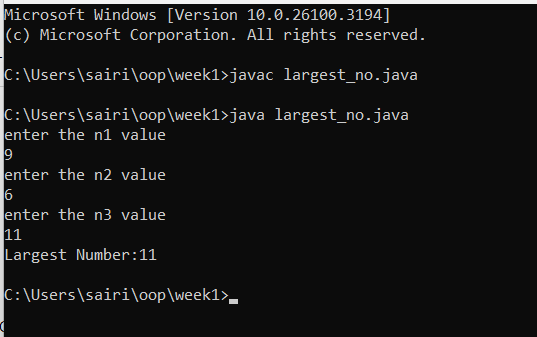
**Task-4**

**AIM**: Write a java program largest of three number using ternary operator**.**

**INPUT CODE:**

****

**Output:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1) | error: ';' expected int result=(a>b) ((a>c)? a:c) : ((b>c)? b:c); error: not a statement | Add a ‘?’ int result=(a>b)? ((a>c)?  a:c) : ((b>c)? b:c); |
|  | int result=(a>b)  ((a>c)? a:c) : ((b>c)? b:c); |  |
| 2) | error: ';' expected int result=(a>b)? ((a>c)? a:c) : ((b>c)? b:c) | Add a ‘;’ int result=(a>b)? ((a>c)? a:c) : ((b>c)? b:c); |

**IMPORTANT POINTS:**

1)import java.util.Scanner; - To accept input from user, Scanner class under util package has to be imported.

2)Scanner input=new Scanner(System.in); - Used to create a Scanner object

3)int a=input.nextInt (); - Used to read integer data type stored under the object created

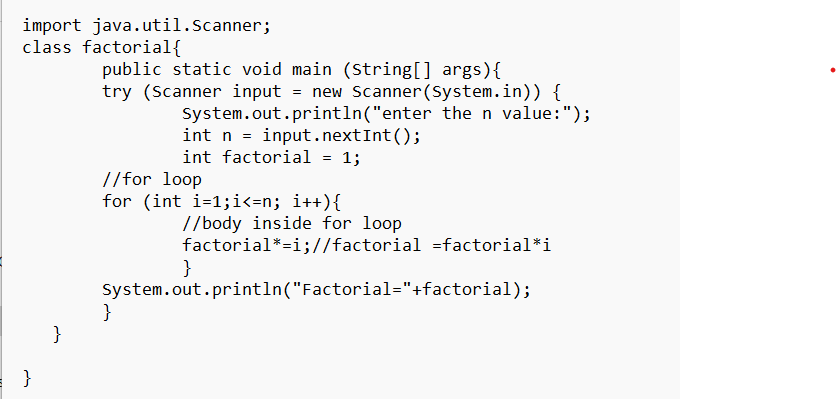
4)int result=(a>b)? ((a>c)? a:c) : ((b>c)? b:c); - Nested Ternary operator is used here.Syntax for ternary operator is- condition?

expression 1: expression 2; , whose answer is stored in a variable and then used

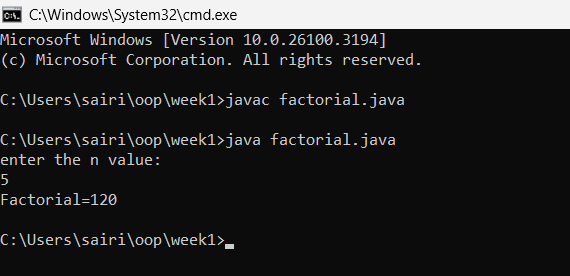
**TASK-5**

**AIM:** Write a java program to find the factorial of number.

**INPUT CODE:**

****

**Output:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Error** | **Error rectification** |
| 1. | error: ';' expected    fact\*=n | Add a “;”  fact\*=n; |

**IMPORTANT POINTS:**

for (int i=1; n>=i;--n){ } - For loop syntax: for(initial expression; test expression; update expression){} The loop is executed, until the test expression evaluates to be false.

**Week-3**

**TASK-1**

**AIM**: Create the java program with the following instruction

1)create a class with name car

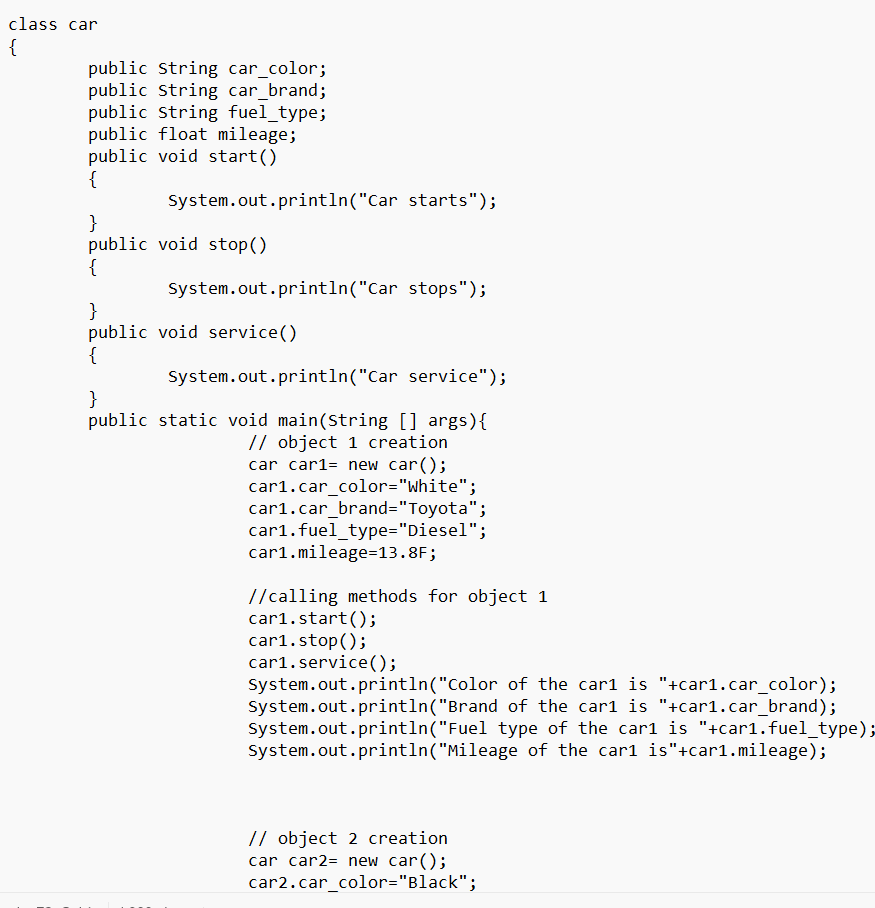
2)create four attributes name, car\_color, car\_brand, fueltype, mileage

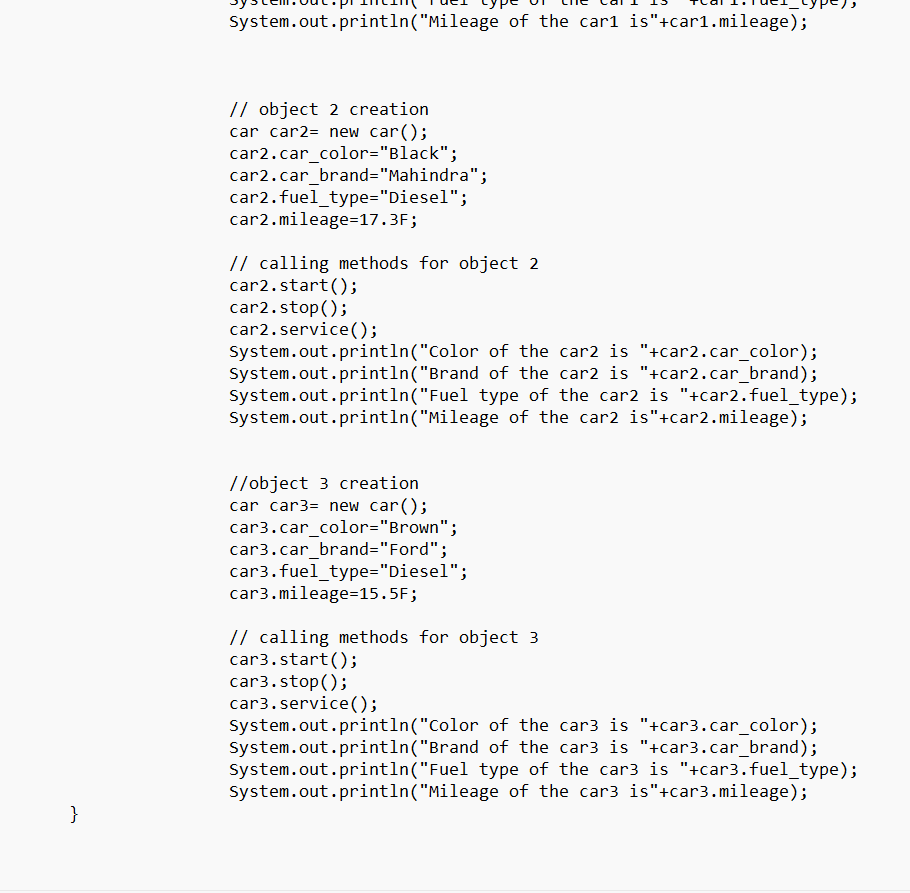
3)create three methods named Start(), Stop(), Services()

4)create three objects name car1,car2,car3

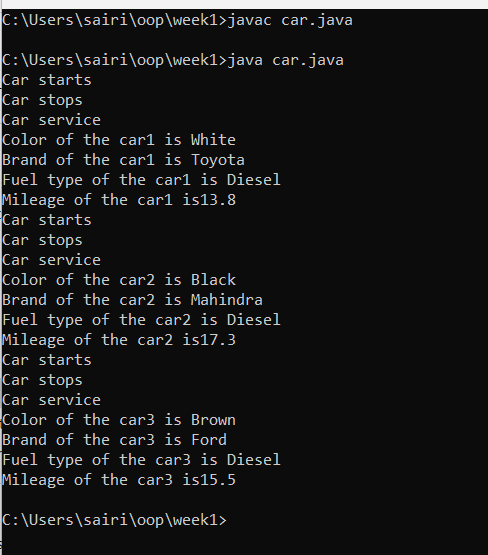
5)create a constructor which should print “welcome to car garage.”

**INPUT CODE:**

****

****

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.No | Error | Error rectification |
| 1) | error: ')' or ',' expected at line  System.out.println(car\_brand + " is being serviced."); | There is an issue with the string  concatenation. Ensure proper quotes and spacing. |
| 2) | error: not a statement at line System.out.println(car\_brand + " is being serviced."); | Verify that all syntax is correct. It is likely due to a misplaced or missing character. |
| 3) | error: ';' expected at line  System.out.println(car\_brand + " is being serviced."); | Check for missing or extra quotes, plus signs, or misplaced semicolons. |

**CLASS DIAGRAM:**

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

**IMPORTANT POINTS:**

1)String car\_color; declares an instance variable car\_color of type String, which will store the color of the car.

2)Car() is a constructor that gets executed when an object of the Car class is created, printing "Welcome to Carshowdown".

3)this.car\_color = car\_color; is used inside a constructor to assign the passed parameter to the instance variable of the same name.

4)Methods like start(), service(), and stop() define specific actions for the car, such as printing messages related to the car’s state.

5)Creating objects like Car car1 = new Car(); allows assigning values to attributes and calling methods like car1.start(); to execute their functionality.

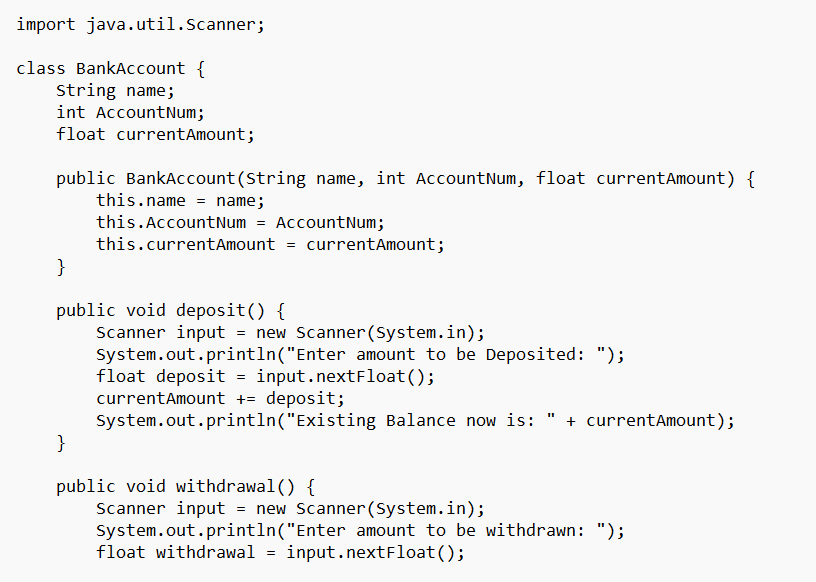
**TASK-2**

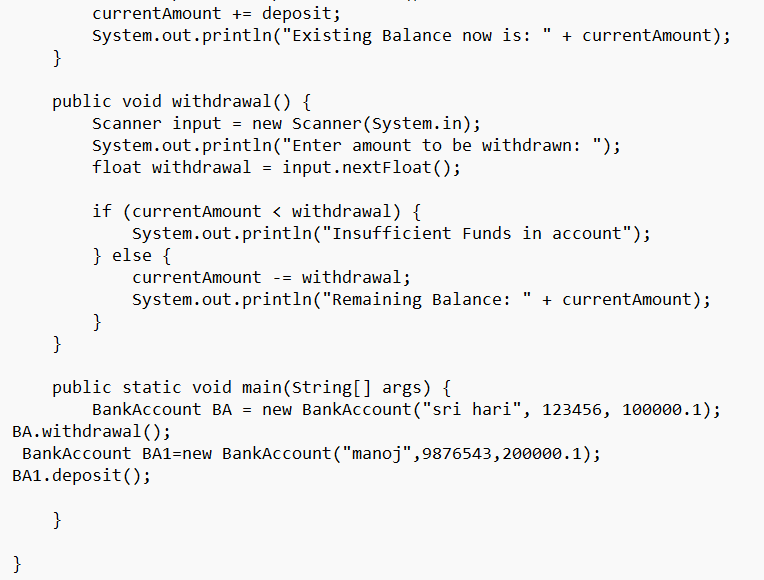
**AIM:** Write a java program to create a class named BankAccount with two methods deposit and withdraw:

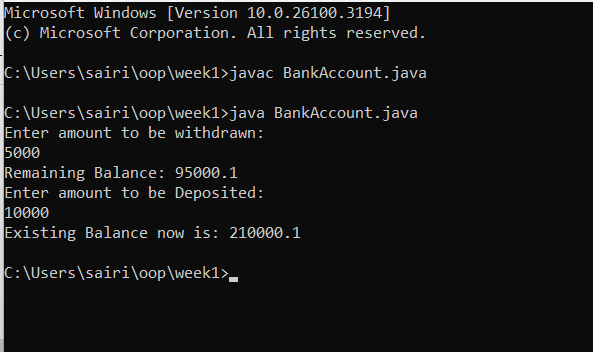
1)In deposit method whenever an amount is deposited it has to be updated with current amount.

2)In withdraw method whenever an amount is being withdraw it has to be less then the current amount else print insufficient funds.

**INPUT CODE:**

****

****

**Output:**

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | error: ';' expected cust1.withdraw(5000) | Add a “;”    cust1.withdraw(5000); |
| 2. | error: cannot find  symbol  thisCurrBal=CurrBal; | Add a “.”  this.CurrBal=CurrBal; |

CLASS DIAGRAM:

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

IMPORTANT POINTS:

1)Classes and Objects – The program defines a BankAccount class and creates an object (cust1) to access methods and store account details.

2)Constructors – The constructor BankAccount(String name, int AccNo, int CurrBal) initializes the object with values when a new bank account is created.

3)Instance Variables – The program uses name, AccNo, and CurrBal as private instance variables to store customer details and account balance.

4)Access Modifiers – The private keyword ensures that instance variables cannot be accessed directly from outside the class, maintaining encapsulation.

5)Methods (Functions) – The withdraw(int WAmt) method deducts money from the balance, and deposit(int DAmt) adds money and returns the updated balance.

6)Conditional Statements – The if-else condition in withdraw checks if the withdrawal amount is less than the current balance before proceeding.

7)Return Statements – The deposit method returns the updated balance after adding the deposited amount.

8)Printing Output (System.out.println) – The program prints account details, withdrawal status, and the new balance after deposits.

9)main Method – The program starts execution from the main method, where an object is created, and methods are called.

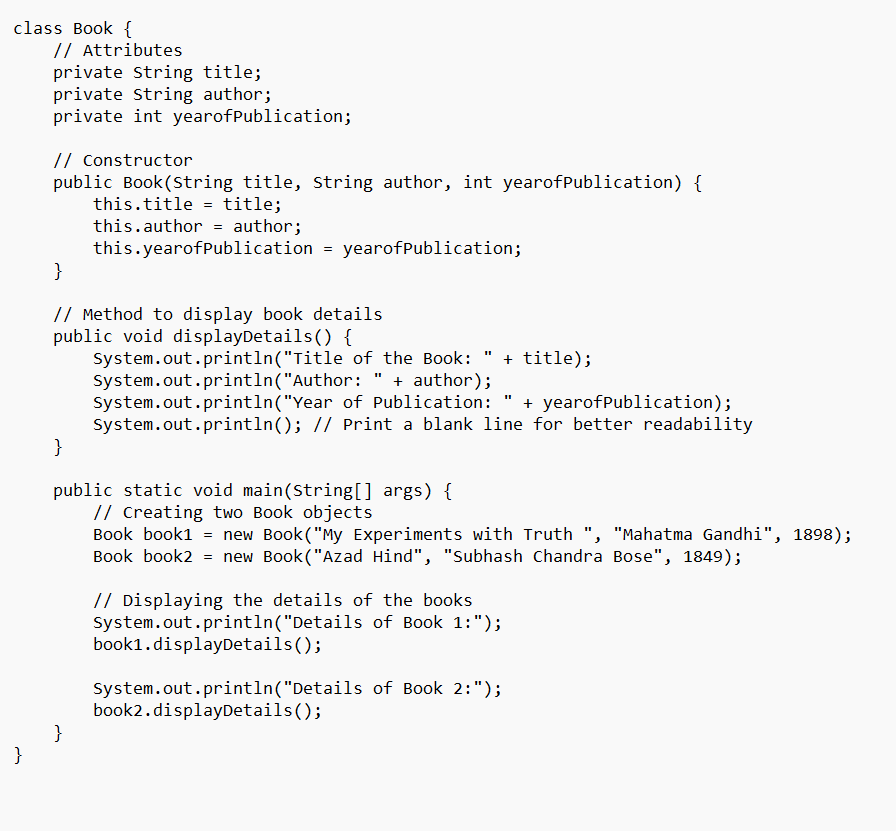
10)Basic Error Debugging – Understanding common Java errors like misspelled method names (depost → deposit), incorrect keywords (retirn → return), and missing braces helps in fixing compilation issues.

**Week-4**

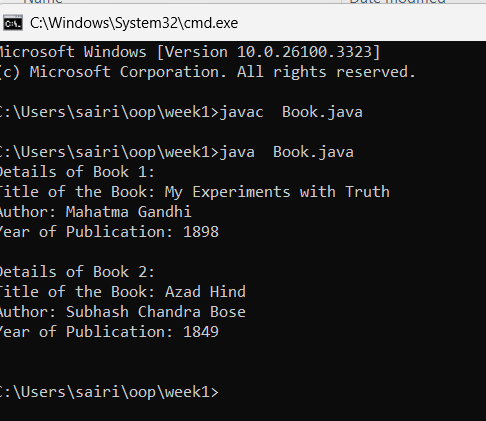
**TASK-1**

**AIM:** To create a java program with class named “Book” the class should contain various attributes “title of the book”,” author”, “year of publication”. It should contain a constructor with parameters which initializes “title of the book”,” author”, “year of publication”. create a method which displays the details of the book i.e “title of the book”,” author”, “year of publication”. Display the details of two books by creating two objects.

**INPUT CODE:**

****

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Error Message** | **Error rectification** |
| 1. | System.out.println(" Your book: "+this Title\_of\_the\_book); - Syntax error | Change this  Title\_of\_the\_book to this.Title\_of\_the\_book. |
| 2. | book.getbook(); - Error in calling method | Replace book.getbook(); with book1.getbook(); and book2.getbook();. |
| 3. | book book1 = new book("Azad Hind", "J.K.Rowling", 1849); - Class name issue | Change book to Book everywhere in the file |

**IMPORTANT POINTS:**

1)The class Book follows Java naming conventions and represents a real-world book entity.

2)Stores book details (Title\_of\_the\_book, Author, Year\_of\_publication).

3)Initializes object properties using the this keyword to avoid variable name conflicts.

4)Displays book details, demonstrating encapsulation and controlled data access.

5)Two book objects (book1, book2) are instantiated and used.

6)Class names should start with an uppercase letter, and the constructor must match the class name exactly.

**CLASS DIAGRAM:**

Book

-

Title\_of\_the\_book: String

-

Author: String

-

Year\_of\_publication: int

Book(title: String,Author:

+

String;Year

of publication: int

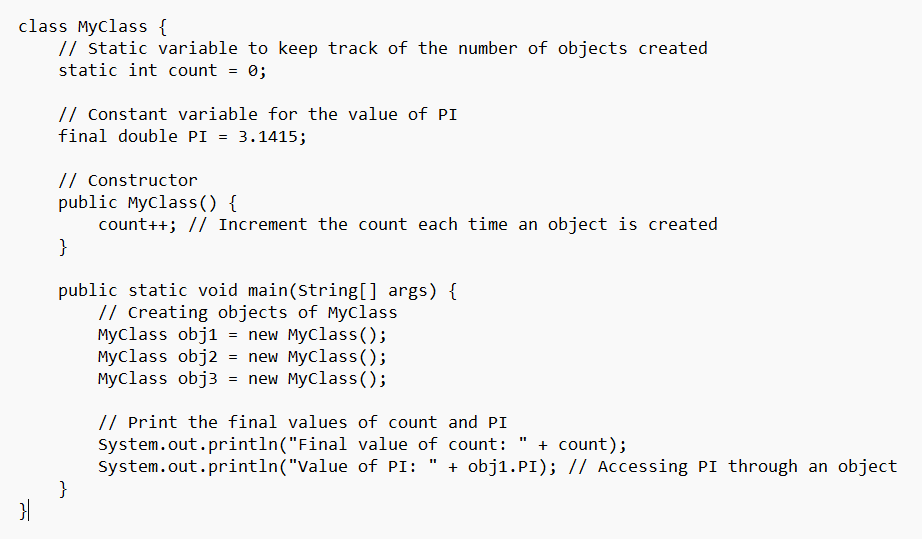
+

getbook( ): void

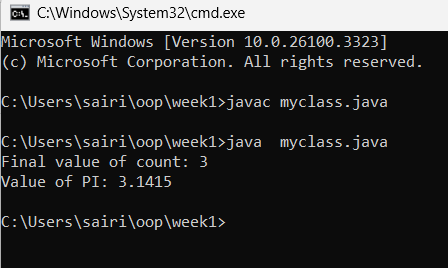
**TASK-2**

**AIM:** To create a java program with class named myclass with a static variable count of “int” type initialize to zero and a constant variable “pi” of type double initialize to 3.1415 as attributes of that class now define a constructor for my class that increments the count variable each time an object of myclass is created finally print the finial values of “count” and “pi” variables.

**INPUT CODE:**

****

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | rectification |
| 1. | count.java:2: error: <identifier> expected at Static int count = 0; | Change Static to static. |
| 2. | count.java:3: error: <identifier> expected at final doublepi = 3.1415; | Change final doublepi = 3.1415; to final double pi = 3.1415;. |
| 3. | count.java:16: error: <identifier> expected at public Static void main(String[] args) { | Change Static to static. |

**IMPORTANT POINTS:**

1)Asec.display() and Bsec.display() access the instance methods and variables through their respective object references.

2)System.out.println(“Double is :”+Bsec.pi); accesses that pi variable of the Bsec object.

3)new keyword followed by the class constructor. This allocates memory for the object and initializes its attributes.

4)new is necessary for creating objects and invoking constructors.

5)Object References are needed to access instance variables and methods.

6)final double pi means that once pi is initialized with the value

3.14, it cannot be changed.

**CLASS DIAGRAM:**

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

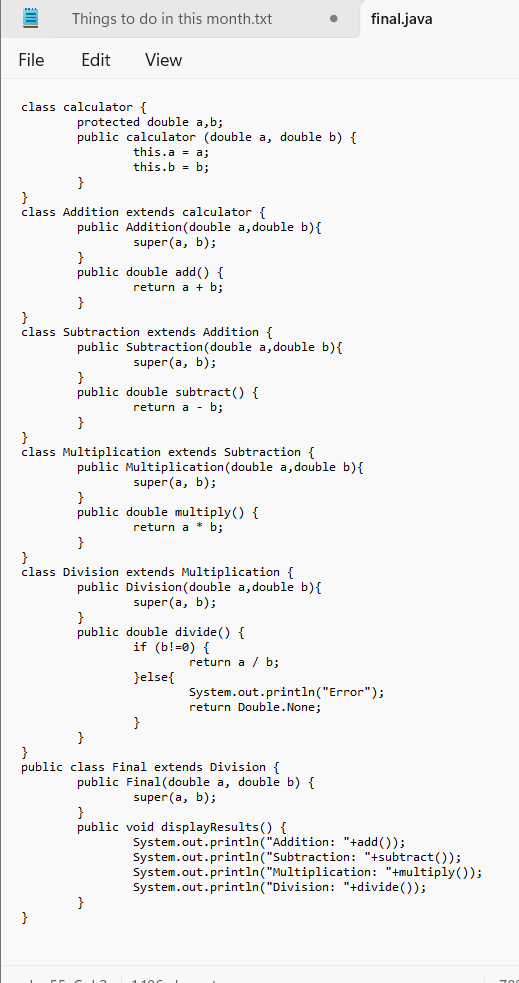
**WEEK-05**

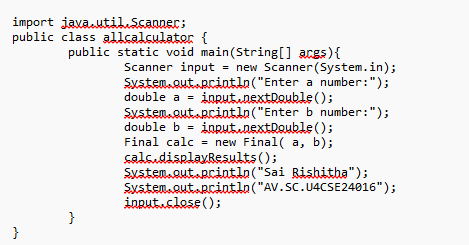
**PROGRAM-01**

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

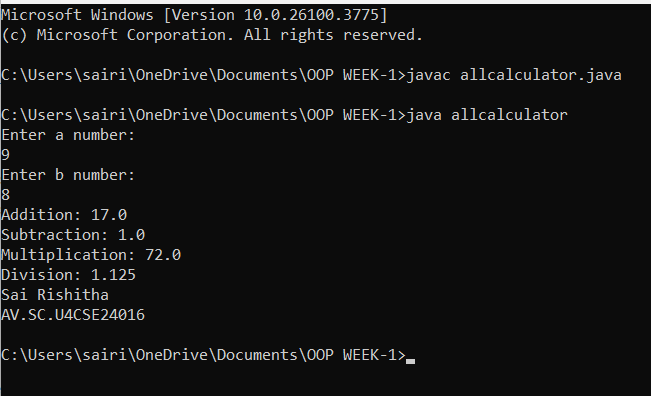
Hint: collect required variables using super class, create each class for a parameter and each class must contain a method.

**CODE:**

****

****

**OUTPUT:**



**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(\*).

**ERRORS:**

|  |  |
| --- | --- |
| **Error** | **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. |

**CLASS DIAGRAM:**

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

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

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

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

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

**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?
   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 sub classes Finally, display the details.

**IMPORTANT POINTS:**

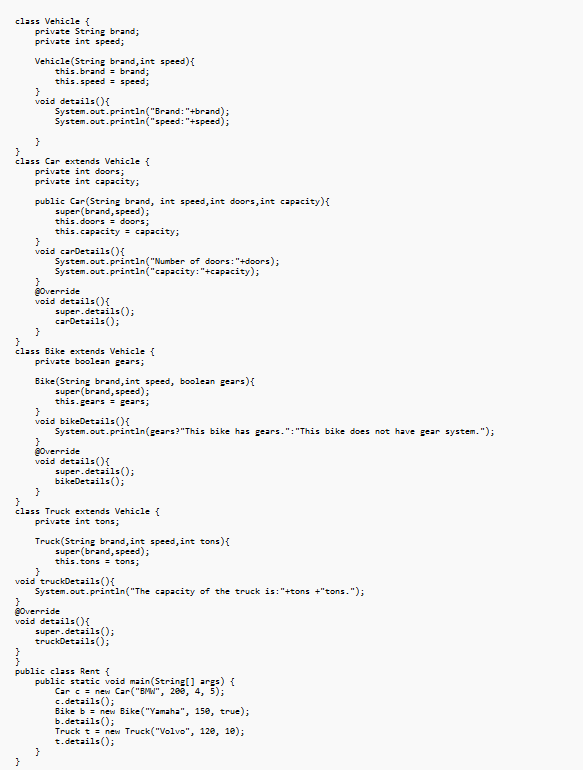
1. a constructor helps in initializing an object that doesn't exist.
2. a method performs functions on pre-constructed or already developed objects.
3. a double method can represent more decimal point numbers than float method.
4. the void keyword in java is used to specify that a method does not return any value. it is a return type that indicates the method performs a function and doesn't produce a result.

**Answer for Q1:**

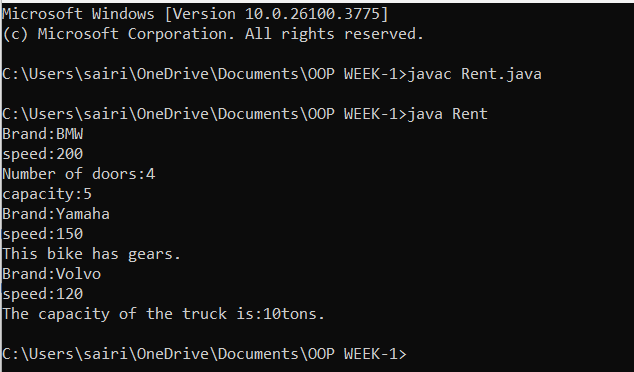
The oops concepts used in the above program are:

Inheritance, encapsulation, polymorphism, abstraction.

**CODE:**

****

**OUTPUT:**

****

**ERRORS:**

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

**CLASS DIAGRAM:**

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

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

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

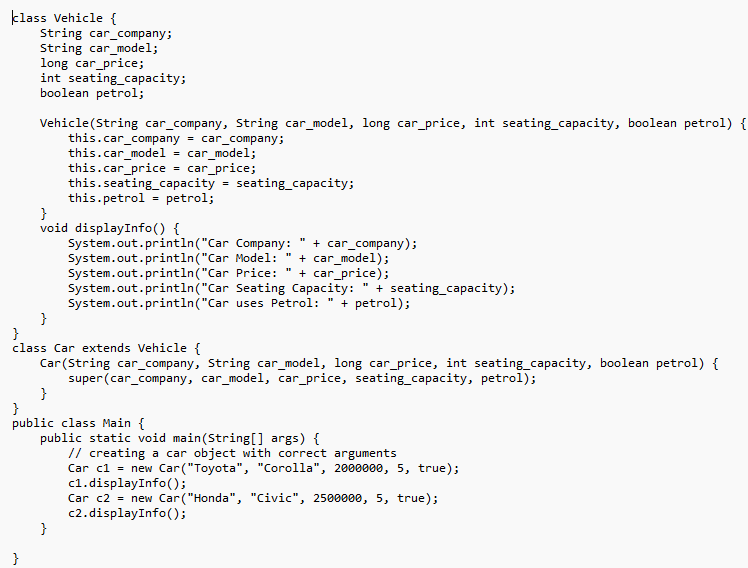
|  |
| --- |
| Calculator |
| -a : double  -b: double |
| +Calculator (a,b) |

**WEEK-06**

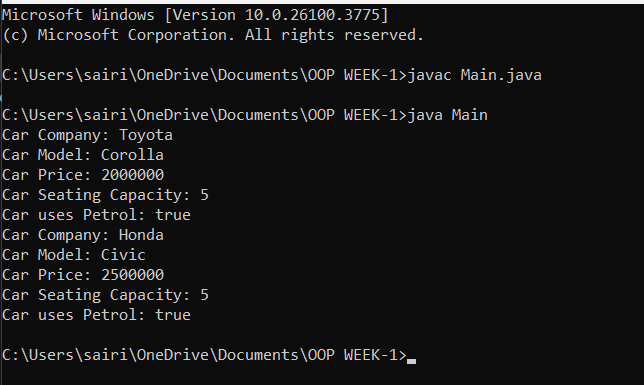
**PROGRAM-1:**

**AIM:** Write a java program to create a vehicle class with a method displayinfo(). Override this method in the car subclass to provide specific information about car (car company, seating capacity, petrol or not).

**CODE:**



**OUTPUT:**



**ERRORS:**

|  |  |
| --- | --- |
| **Code error** | **Code rectification** |
| 1.Incorrect class name for main method(Truck).    2.Inconsistent car model output in displayinfo(). | 1.Rename Truck to Main or place main inside car or vehicle.  2. Ensure Car correctly passes Toyota” to  super(car\_model,color,fueltype) |

**IMPORTANT POINTS:**

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

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

**3.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.

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

**CLASS DIAGRAM:**

|  |
| --- |
| **Vehicle** |
| -Brand: String  -Speed: int |
| + vehicle(brand: string  Speed: int)  +start vehicle(): void  +displaydetails():void |

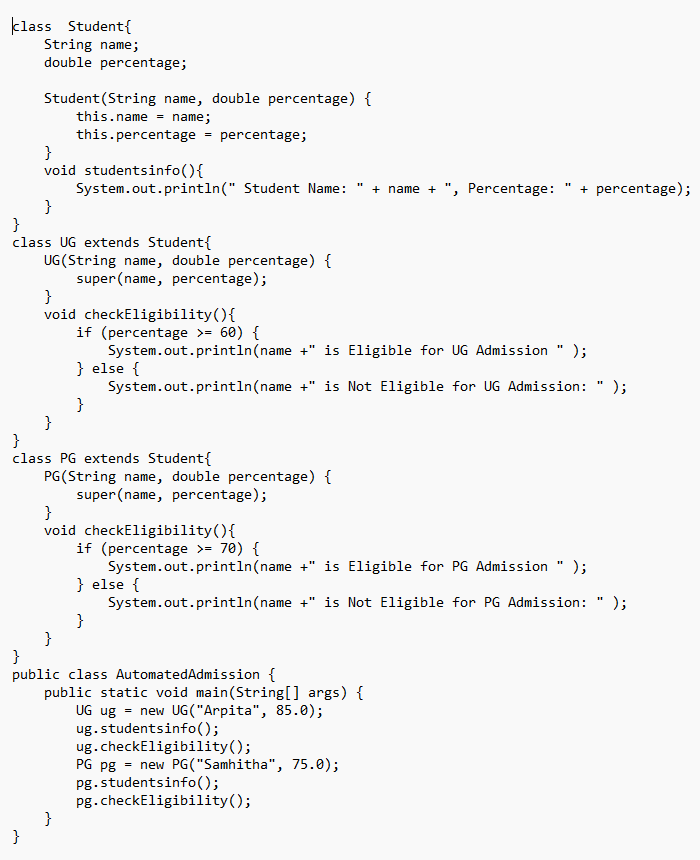
**PROGRAM-2:**

**AIM:** A college is developing an automated admission system that verifies students eligibility(UG) and postgraduation(PG) programs. Each program has different eligibility criteria based on the students percentage in their previous qualification.

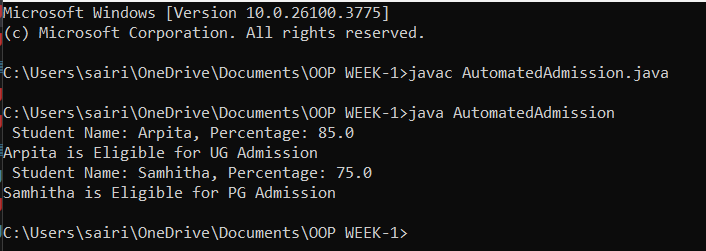
1.UG admission recquire a minimum of 60%.

2.PG admission recquire a minimum of 70%.

**CODE:**



**OUTPUT:**



**ERRORS:**

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

**IMPORTANT POINTS:**

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

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

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

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

**CLASS DIAGRAM:**

|  |
| --- |
| **AutomatedAdmission** |
| -Scanner: scanner  -Name: String  -Percentage : double  -Program: stirng |
| + main(args:String[]): void  +takeInput(): void  +checkEligibility(): void  +closeScanner(); void |

**PROGRAM-3:**

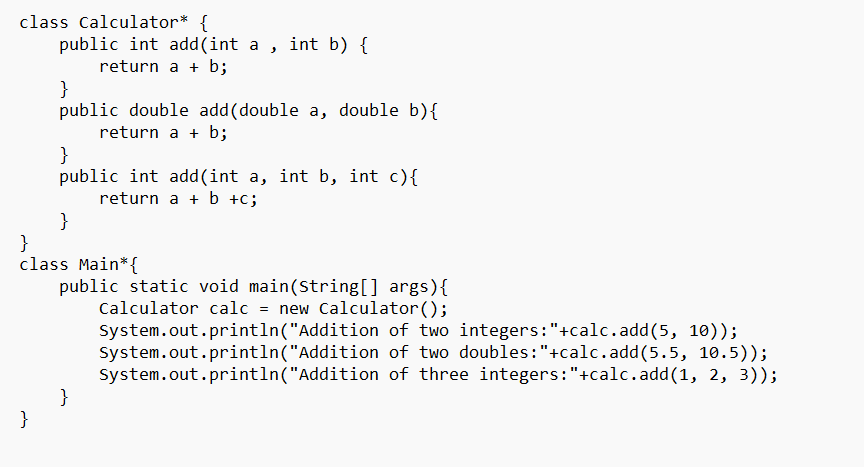
**AIM:** Create a calculator class with overloaded methods to perform addition of:

1.Add two integers

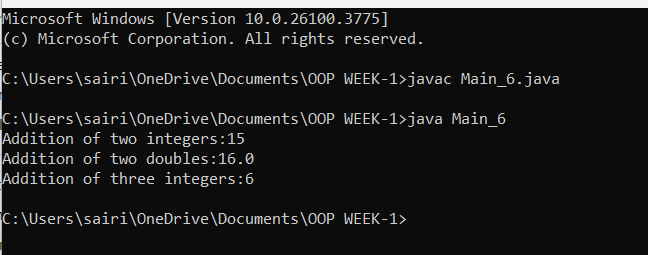
2.Add two doubles

3.Add three integers

**CODE:**



**OUTPUT:**



**ERRORS:**

|  |  |
| --- | --- |
| **Code error** | **Code rectification** |
| 1.Method parameters missing spaces. E.g.,”inta, intb”should be “int a, int b”  2.Inconsistent indentation in method bodies | 1.Add proper spacing between parameters: (int a, int b)  2.Fix indentation:  Consistent 4 space o indentation. |

**IMPORTANT POINTS:**

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

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

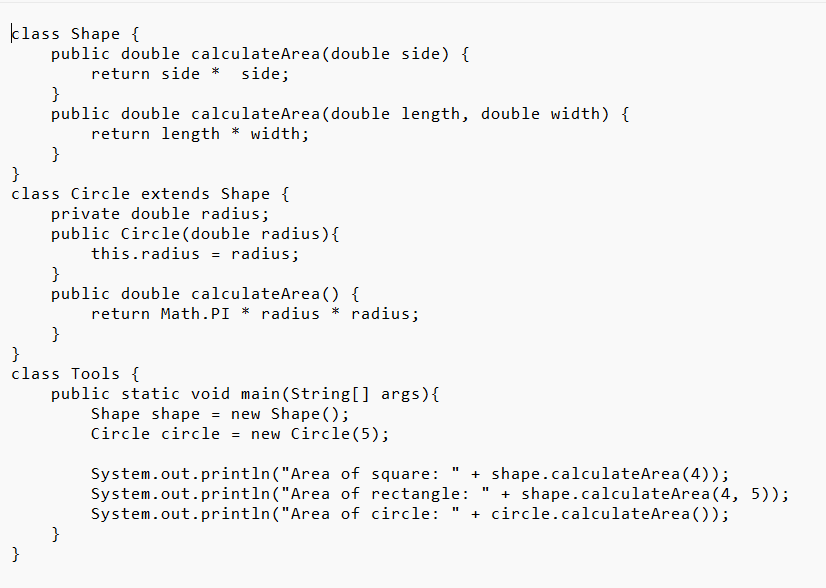
**CLASS DIAGRAM:**

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

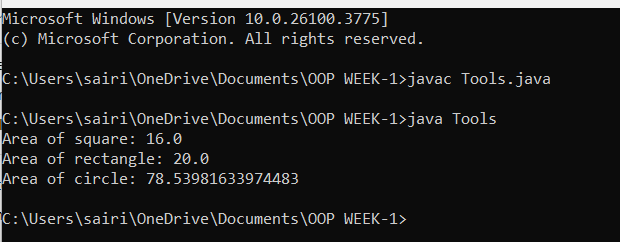
**PROGRAM-4:**

**AIM:** Create a shape class with a method to calculate area i.e., overloaded for different shapes eg: Squares, Recatangle. Then create a subclass circle that overrides the calculateArea() method for a circle.

**CODE:**



**OUTPUT:**



**ERRORS:**

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

**CLASS DIAGRAM:**

|  |
| --- |
| **SHAPE** |
| + CalculateArea(side:double): double  +CalculateArea(width:double,length:double):double |

|  |
| --- |
| **CIRCLE** |
| + CalculateArea(radius : double): double |

|  |
| --- |
| **TOOLS** |
| +main(args:String[]):Void |

**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)).

**WEEK-07**

**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 implements the sound() method to make a specific sound for each animal.

**CODE:**

abstract class Animal {

*// Abstract method*

public abstract void sound(); }

*// Subclass Lion*

class Lion extends Animal {

@Override public void sound() {

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

} *// Subclass Tiger*

class Tiger extends Animal {

@Override public void sound() {

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

} *// Main class*

public class Main {

public static void main(String[] args) {

Animal lion = new Lion();

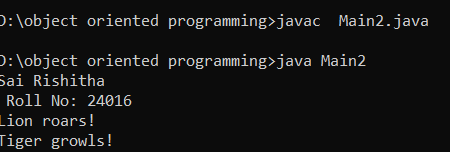
lion.sound();

Animal tiger = new Tiger();

tiger.sound();

} }

**OUTPUT:**

****

**ERROR:**

| Error Type | Cause | Recification |
| --- | --- | --- |
| Method Not Overridden | Subclass does not override sound() | Ensure all subclasses implement sound() |
| Typo in Method Name | Misspelling sound() in subclass | Use @Override annotation for compile-time check |

**CLASSDIAGRAM:**

|  |
| --- |
| Animal |
| (abstract)sound():void |

|  |
| --- |
| Tiger |
| +Sound();void; |

|  |
| --- |
| Lion |
| +Sound():void; |

**IMPORTANT POINTS:**

* Abstraction: The Animal class is abstract and cannot be instantiated directly. It defines the contract for subclasses to implement the sound() method[3](https://www.w3resource.com/java-exercises/java-abstract-class-exercise-1.php)[4](https://www.upgrad.com/tutorials/software-engineering/java-tutorial/abstract-method-in-java/)[5](https://iqratechnology.com/academy/java/java-basic/java-abstract/).
* Inheritance: Lion and Tiger inherit from Animal and provide their own implementation of the sound() method[3](https://www.w3resource.com/java-exercises/java-abstract-class-exercise-1.php).
* Polymorphism: The main method demonstrates polymorphism by using Animal references for Lion and Tiger objects, allowing dynamic method binding[3](https://www.w3resource.com/java-exercises/java-abstract-class-exercise-1.php)[5](https://iqratechnology.com/academy/java/java-basic/java-abstract/).
* Extensibility: New animal types can easily be added by extending Animal and implementing the sound() method.

**PROGRAM-2**

**AIM:**  Write a java program to create an abstract class called shape 3D with abstract methods volume() and calculate surfacearea() .Create subclasses Sphere and Cube that extend the shape 3D class and implement the respective methods to calculate the volume and surface area of each shape.

**CODE:**

abstract class Shape3D {

public abstract double calculateVolume();

public abstract double calculateSurfaceArea();

}

*// Sphere subclass*

class Sphere extends Shape3D {

private double radius;

public Sphere(double 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);

} }

*// Cube subclass*

class Cube extends Shape3D {

private double sideLength;

public Cube(double sideLength) { this.sideLength = sideLength;

}

@Override public double calculateVolume() {

return Math.pow(sideLength, 3);

}

@Override public double calculateSurfaceArea() {

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

} }

*// Main class*

public class Main { public static void main(String[] args) {

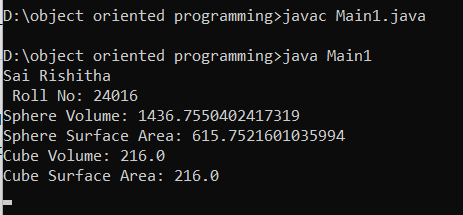
Shape3D sphere = new Sphere(7.0);

Shape3D cube = new Cube(6.0);

System.out.println("Sphere Volume: " + sphere.calculateVolume()); System.out.println("Sphere Surface Area: " + sphere.calculateSurfaceArea()); System.out.println("Cube Volume: " + cube.calculateVolume()); System.out.println("Cube Surface Area: " + cube.calculateSurfaceArea());

} }

**OUTPUT:**

****

**ERROR:**

| Error Type | Cause | Rectification |
| --- | --- | --- |
| Instantiation Error | Attempt to instantiate Shape3D directly | Only instantiate subclasses, not abstract class |
| Method Not Implemented | Subclass does not override abstract methods | Ensure all subclasses implement required methods |

**CLASSDIAGRAM:**

|  |
| --- |
| Shape3D |
|  |
| (abstract)calculatevolume:void  (abstract)calculatesurfacearea:void |

|  |
| --- |
| Sphere |
|  |
| +calcualtevolume:void  +calculatesurfacearea:void |

|  |
| --- |
| Cube |
|  |
| +calculatevolume:void  +calculatesurfacearea:void |

**IMPORTANT POINTS:**

* Abstraction: Shape3D is an abstract class that cannot be instantiated directly. It defines a contract for all 3D shapes to implement calculateVolume() and calculateSurfaceArea() methods[1](https://www.w3resource.com/java-exercises/java-abstract-class-exercise-6.php).
* Inheritance: Sphere and Cube extend Shape3D and provide their own implementations for volume and surface area calculations[1](https://www.w3resource.com/java-exercises/java-abstract-class-exercise-6.php).
* Extensibility: Additional 3D shapes can be added by extending Shape3D and implementing the required methods.

**PROGRAM-3:**

**AIM:** Write a java program using an abstract class to define a method for pattern printing

* Create an abstract class named PtternPrint 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.

**CODE:** abstract class PatternPrint {

*// Abstract method to be implemented by subclasses*

public abstract void printPattern(int n);

*// Concrete method to display pattern title*

public void displayTitle(String title) {

System.out.println("\n" + title);

System.out.println("----------------");

}

}

class StarPattern extends PatternPrint {

@Override

public void printPattern(int n) {

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

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

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

}

System.out.println();

}

}

}

class NumberPattern extends PatternPrint {

@Override

public void printPattern(int n) {

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

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

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

}

System.out.println();

}

}

}

public class Main {

public static void main(String[] args) {

int rows = 5;

StarPattern star = new StarPattern();

star.displayTitle("Star Pattern");

star.printPattern(rows);

NumberPattern num = new NumberPattern();

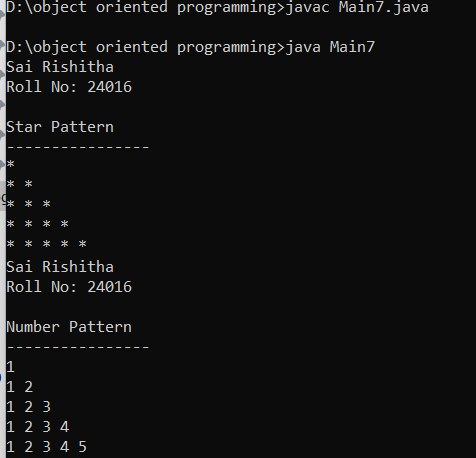
num.displayTitle("Number Pattern");

num.printPattern(rows);

}

}

**OUTPUT:**

****

**ERROR:**

| Error Type | Cause | Recitification |
| --- | --- | --- |
| NegativeArraySize | Negative input value for rows | Add input validation |
| TypeMismatch | Non-integer input for rows | Use try-catch block |
| NullPointer | Uninitialized subclass object | Check object initialization |
| StackOverflow | Extremely large input values | Add maximum limit validation |

**CLASSDIAGRAM:**

|  |
| --- |
| <<abstract>>  Pattran |
| +printpattran():void  +concrete(): void |

|  |
| --- |
| Pattran1 |
| +printpattran():void  +concrete():void |

|  |
| --- |
| Pattran2 |
| +printpattran():void  +concrete():void |

**IMPORTANT POINTS:**

1. Abstraction: PatternPrint abstract class defines the structure without implementation
2. Inheritance: Subclasses inherit displayTitle method from parent
3. Polymorphism: Runtime method binding through method overriding
4. Pattern Logic:
   * Star: Nested loops with i rows and i stars per row
   * Number: Inner loop prints sequential numbers in each row
5. Scalability: Easy to add new patterns by extending PatternPrint

**WEEK-08**

**PROGRAM-1:**

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

**CODE:**

interface Shape {

    double getPerimeter();

}

class Rectangle implements Shape {

    private double length;

    private double width;

    public Rectangle(double length, double width) {

        this.length = length;

        this.width = width;

    }

    @Override

    public double getPerimeter() {

        return 2 \* (length + width);

    }

}

class Circle implements Shape {

    private double radius;

    public Circle(double radius) {

        this.radius = radius;

    }

    @Override

    public double getPerimeter() {

        return 2 \* Math.PI \* radius;

    }

}

class Triangle implements Shape {

    private double sideA;

    private double sideB;

    private double sideC;

    public Triangle(double sideA, double sideB, double sideC) {

        this.sideA = sideA;

        this.sideB = sideB;

        this.sideC = sideC;

    }

    @Override

    public double getPerimeter() {

        return sideA + sideB + sideC;

    }

}

public class Shapeperi {

    public static void main(String[] args) {

        Shape rectangle = new Rectangle(5, 3);

        Shape circle = new Circle(4);

        Shape triangle = new Triangle(3, 4, 5);

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

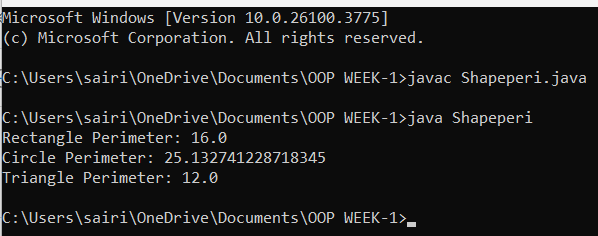
        System.out.println("Circle Perimeter: " + circle.getPerimeter());

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

    }

}

**OUTPUT:**

****

**ERROR:**

| **Line** | **Error** | **Rectification** |
| --- | --- | --- |
| No explicit package | No package declaration | Optional, but recommended to organize code |
| Shapeperi class name | Class name Shapeperi is unconventional | Use ShapePerimeter or ShapeDemo for clarity |
| getPerimeter() method | No errors | Correctly overridden in all classes |

**CLASS DIAGRAM:**

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

| <<interface>> |

| Shape |

|-----------------|

| +getPerimeter() : double |

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

^

|

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

| | |

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

| Rectangle | | Circle | | Triangle |

|-----------| |----------| |------------|

| -length: double | | -radius: double | | -sideA: double |

| -width: double | | | | -sideB: double |

|-----------------------| |----------------------| | -sideC: double |

| +Rectangle(length, width) | +Circle(radius) | | +Triangle(sideA, sideB, sideC) |

| +getPerimeter(): double | +getPerimeter(): double | +getPerimeter(): double |

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

**IMPORTANT POINTS:**

1. Interface Usage:
   * Shape is an interface with a method getPerimeter() which all shapes must implement.
   * This promotes polymorphism, allowing different shapes to be treated uniformly.
2. Classes Implementing Shape:
   * Rectangle, Circle, and Triangle all implement the Shape interface.
   * Each class provides its own implementation of getPerimeter() based on its geometry.
3. Encapsulation:
   * Each class encapsulates its properties (length, width, radius, sideA, sideB, sideC) as private fields.
   * These fields are initialized via constructors.
4. Polymorphism in Main:
   * In the main method, objects of different shapes are created but referenced by the Shape interface type.
   * This allows calling getPerimeter() polymorphically.
5. Mathematical correctness:
   * Perimeter formulas are correctly implemented:
     + Rectangle: 2 \* (length + width)
     + Circle: 2 \* π \* radius
     + Triangle: sum of all three sides

**PROGRAM-2:**

**AIM:** Write a java program to create an interface Playable with a method play() that takes no arguments and return void. Create three classes Football, Volleyball, and Basketball that implements the playable interface and override method to display score.

**CODE:**

interface Playable {

    void play();

}

class Football implements Playable {

    @Override

    public void play() {

        System.out.println("Football Score: Team A 2 - 1 Team B");

    }

}

class Volleyball implements Playable {

    @Override

    public void play() {

        System.out.println("Volleyball Score: Team A 3 - 2 Team B");

    }

}

class Basketball implements Playable {

    @Override

    public void play() {

        System.out.println("Basketball Score: Team A 98 - 95 Team B");

    }

}

public class Playablegame {

    public static void main(String[] args) {

        Playable football = new Football();

        Playable volleyball = new Volleyball();

        Playable basketball = new Basketball();

        football.play();

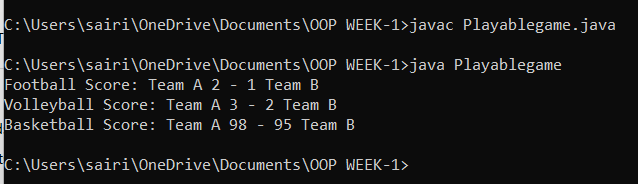
        volleyball.play();

        basketball.play();

    }

}

**OUTPUT:**

****

**ERROR:**

| Line | Error | Rectification |
| --- | --- | --- |
| No package declaration | Not an error, but recommended for larger projects | Optional for small examples |
| Class name Playablegame | Naming convention: class names should be PascalCase (PlayableGame) | Rename for better readability and convention |
| play() method | No errors | Properly overridden in all implementing classes |

**CLASS DIAGRAM:**

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

| <<interface>> |

| Playable |

|-----------------|

| +play(): void |

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

^

|

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

| | |

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

| Football | | Volleyball | | Basketball |

|------------| |------------| |-------------|

| +play() | | +play() | | +play() |

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

**IMPORTANT POINTS:**

1. Interface Definition:
   * Playable is an interface with a single method play().
   * This defines a contract that any class implementing Playable must provide an implementation for play().
2. Implementing Classes:
   * Football, Volleyball, and Basketball classes implement the Playable interface.
   * Each class overrides the play() method to print a specific score message.
3. Polymorphism:
   * In the main method, objects of different game types are created but referenced by the Playable interface type.
   * This allows calling the play() method polymorphically, demonstrating runtime polymorphism.

**PROGRAM-3:**

**AIM:** Write a java program to implement a login system using interfaces.

**CODE:**

interface Loginsystem {

    boolean login(String id,String password);

}

class checking implements Loginsystem {

    public boolean login(String id, String password){

        if (id.equals("AV.SC.U4CSE24016") && password.equals("123456")) {

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

            return true;

        } else{

            System.out.println("Invalid login Id or Password!");

            return false;

        }

        }

}

public class calculate {

    public static void main(String[] args) {

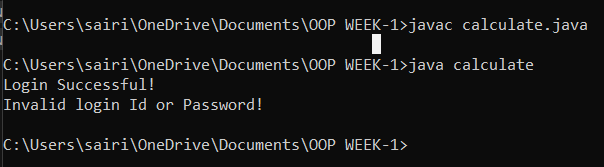
        checking obj = new checking();

        obj.login("AV.SC.U4CSE24016", "123456");

        obj.login("AV.SC.U4CSE24011", "1234567");

    }}

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| S.no | Error message | rectification |
| 1. | ‘{‘ expected | Insert { |
| 2. | Class,interface,enum, or record expected | Removed extra } |

**CLASS DIAGRAM:**

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

          | <<interface>>     |

          |  LoginSystem      |

          +-------------------+

          | +login(id, password): boolean |

          +-------------------+

                    ▲

                    |

          +-------------------+

          |  checking         |

          +-------------------+

          | +login(id, password): boolean |

          +-------------------+

          +-------------------+

          |  calculate        |

          +-------------------+

          | +main(args): void |

          +-------------------+

**IMPORTANT POINTS:**

1. The checking class implements the LoginSystem interface using the login method.
2. Hardcoded credentials are used for login validation.
3. String comparison with equals() is case-sensitive.