**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**LAB REPORT**



**Department of Computer Science Engineering**

## **Amrita School of Computing**

## **Amrita Vishwa Vidyapeetham, Amaravati Campus**

## 

VERIFIED BY:  **NAME: T. SRAVYA**

**ROLL NO: AV.SC.U4CSE24331**

**WEEK-1**

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

**Installing of JDK (Java Development Kit):**

1. **Download JDK:**
   1. Go to the Oracle JDK download page in your web browser and click on JDK-21 version which is long term support (LTS) version.
   2. Click on the download link for your operating system (Windows, macOS, or Linux).
2. **Install JDK:**
   1. Once downloaded, run the installer.
   2. Follow the instructions and keep clicking "Next" until it's done.
3. **Set Environment Variables (Windows):**
   1. Open file explorer, then right click on This PC next select on properties then it will take you to the settings app then click on advanced system settings and then  
       click on **Environment Variables**.
   2. Click **New** under **System Variables**:
      1. **Set Variable name as:** java home
      2. **Variable value:** The folder address where JDK is installed (like C:\Program Files\Java\jdk-21\bin)

Find Path under **System Variables**, click **Edit**, and add the path of the jdk-21(C:\Program Files\Java\jdk-21\bin)

A screenshot of a computer

AI-generated content may be incorrect. **Checking of JDK Version:**

1. **Open Command Prompt:**
   1. Press win+R, type cmd, and press Enter.
2. **Check Version:**
   1. Type java --version and press Enter.
   2. Type javac --version and press Enter.

A screenshot of a computer

AI-generated content may be incorrect.

**2.Simple Java Program for printing Name, Class, Roll No, of a Student**

**Write your code in Notepad and execute in cmd prompt**

**CODE:**

**class Main {**

**public static void main (String [] args ) {**

**System.out.println ("Name: T.Sravya");**

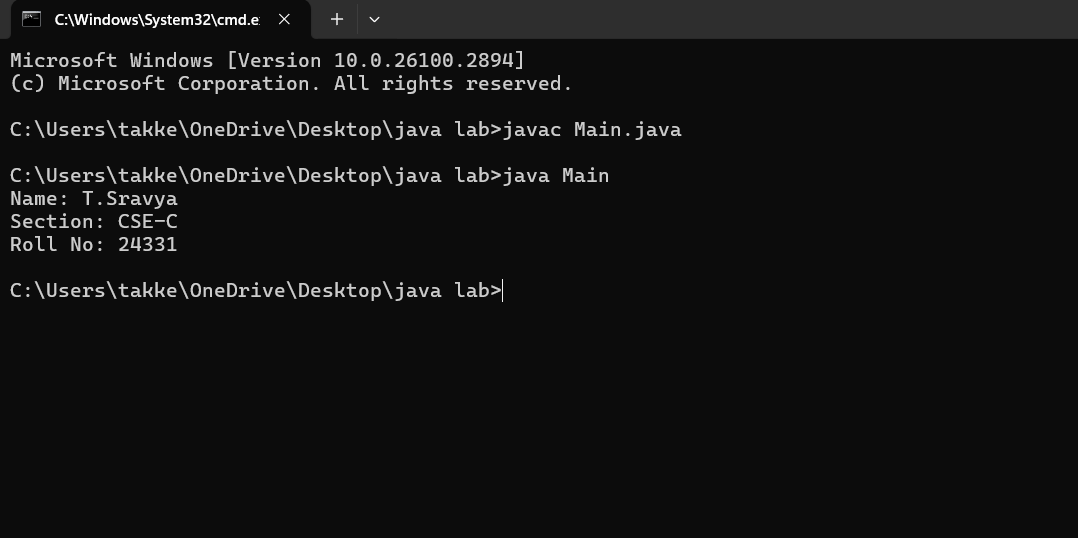
**System.out.println("Section: CSE-C");**

**System.out.println("Roll No: 24331");**

**}**

**}**

**OUTPUT:**

****

Week-2

**1.Write a Java program to Simple Interest where read values from users.**

**CODE:**

**import java.util.Scanner;**

**public class SimpleInterest {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the principal amount (P): ");**

**double principal = scanner.nextDouble();**

**System.out.print("Enter the rate of interest (R) in percentage: ");**

**double rate = scanner.nextDouble();**

**System.out.print("Enter the time period (T) in years: ");**

**double time = scanner.nextDouble();**

**double simpleInterest = (principal \* rate \* time) / 100;**

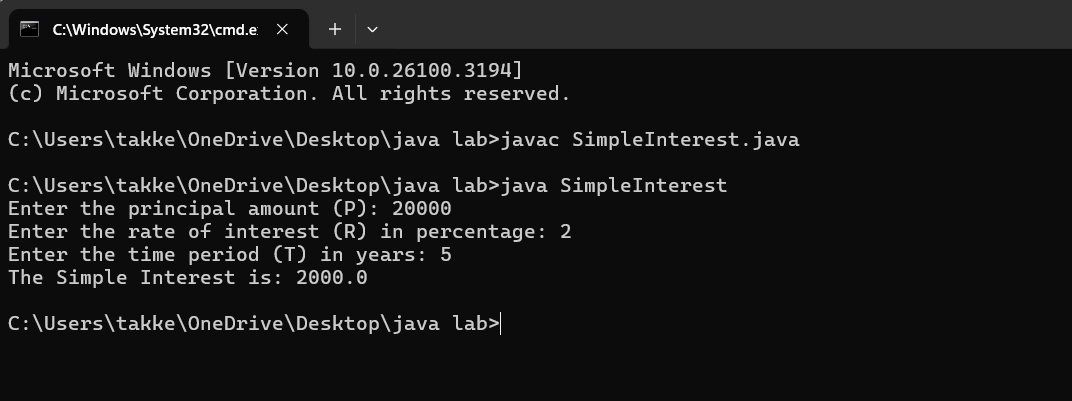
**System.out.println("The Simple Interest is: " + simpleInterest);**

**scanner.close();**

**}**

**}**

**OUTPUT:**

****

**ERROR:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | error: ';' expected  System.out.print("Enter the rate of interest (R) in percentage: ") | **Insert: ‘;’**  **System.out.print("Enter the rate of interest (R) in percentage: ");** |
| **2.** | **error: cannot find symbol**  **Scanner scanner = new scanner(System.in);**  **symbol: class scanner**  **location: class SimpleInterest** | **Replace capital S in s: Scanner scanner = new Scanner(System.in);** |

**IMPORTANT POINTS:**

1.used Scanner library to get input from user in run time.

2.”import java.util.Scanner;”-step to import library.

3.”Scanner input= new Scanner(System.in);”-step to use the scanner.[case sensitive]

4. Should give ‘;’ symbol at the end of System.out.print("Enter the rate of interest (R) in percentage: ").

**2.Write a program on java on Factorial of a number with inputs taken.**

**CODE:**

import java.util.Scanner;

public class Factorialofanumber {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number: ");

int number = scanner.nextInt();

long factorial = 1;

if (number < 0) {

System.out.println("No Factorial For Negative Numbers.");

} else {

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

factorial \*=i;

}

System.out.println("The factorial of " + number + " is " + factorial);

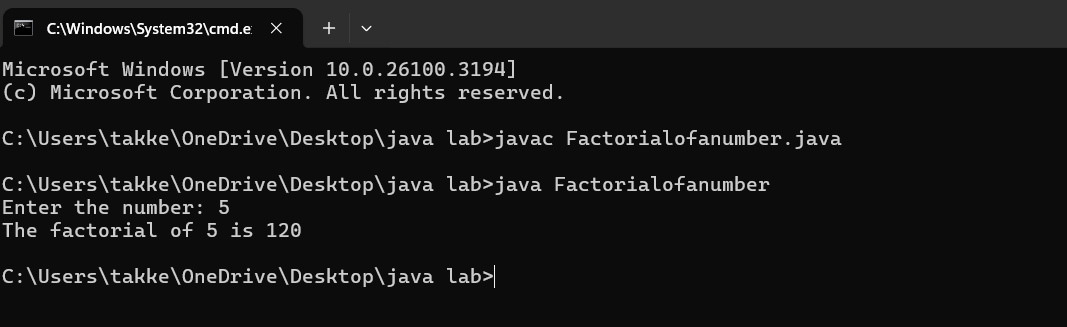
}

scanner.close();

}

}

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAAGE** | **ERROR RECTIFICATION** |
| **1.** | Enter the number: 5  The factorial of 5 is 1  Error: factorial \*=1; | Replace i in 1 place  Replace:factorial \*=i; |
| **2.** | error: 'else' without 'if'  } else {  Reason: if (number < 0); | Solution:  if (number < 0) { |

**Important points:**

**1.** In the loop, the expression factorial\*i; is incorrect because the result of the multiplication is not assigned back to factorial. It should be factorial\*i; for proper multiplication and assignment.

2. **Using a colon** after if will cause a syntax error.

**3. Curly braces** {} are used to group the code inside the if block.

**3.Write a java program to convert the Temperature from Celsius to Fahrenheit & Fahrenheit to Celsius.**

**CODE FOR CELSIUS TO FAHRENHEIT:**

**import java.util.Scanner;**

**public class CelsiustoFahrenheit {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter temperature in Celsius: ");**

**float celsius = scanner.nextFloat();**

**float fahrenheit = (celsius \* 9 / 5) + 32;**

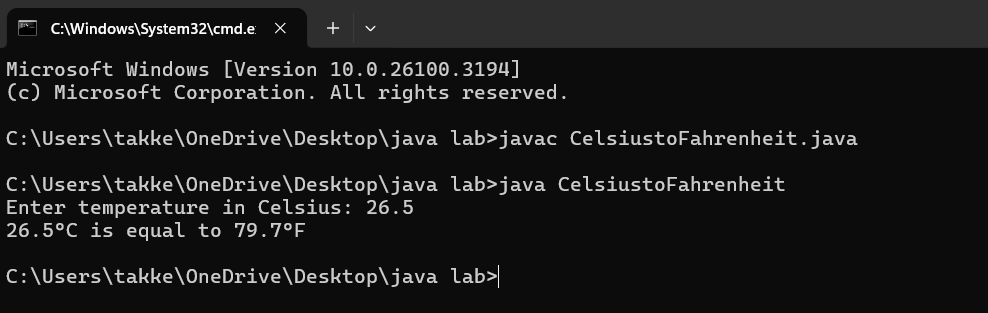
**System.out.println(celsius + "°C is equal to " + fahrenheit + "°F");**

**scanner.close();**

**}**

**}**

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | Error: float fahrenheit = celsius \* 9 / 5 + 32;  Reason:Formula mistake. | It should be ((celsius \* 9) / 5) + 32 |
| **2.** | Error: System.out.println(celsius + "°C is equal to " + fahrenheit "°F") | Print Missing semicolon here,  Print statement is missing a semicolon |

**Important Points:**

1. In the line float fahrenheit=celsius \* 9 / 5 + 32;, the formula does not have proper parentheses for the multiplication and division. It should be float fahrenheit = (celsius \* 9 / 5) + 32; for correct order of operations.

2. In the System.out.println() statement, there is a missing semicolon at the end of the line.

**CODE FOR FAHRENHEIT TO CELSIUS:**

**CODE:**

**import java.util.Scanner;**

**public class FahrenheittoCelsius {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter Fahrenheit temp: ");**

**float f = scanner.nextFloat();**

**float celsius = (f - 32) \* 5 / 9;**

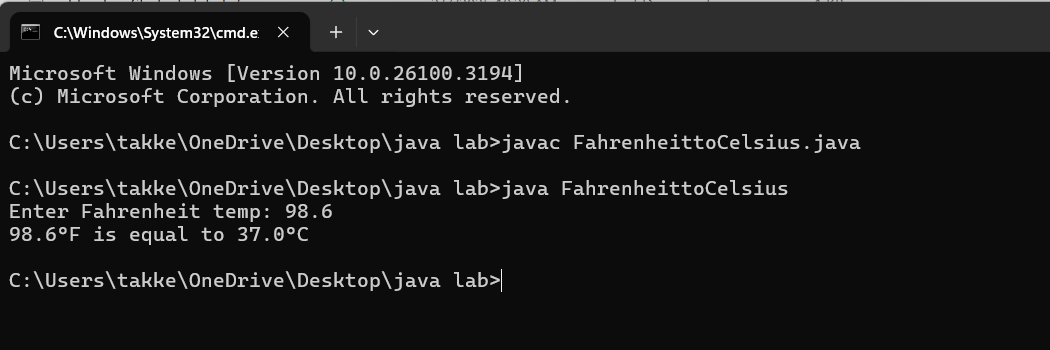
**System.out.println(f + "°F is equal to " + celsius + "°C");**

**scanner.close();**

**}**

**}**

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | Error: Print statement with incorrect variable name System.out.println(fahrenheit + "°F is equal to " + Celsius + "°C") | 'Celsius' should be lowercase  It should be ‘celsius’. |
| **2.** | Duplicate close of scanner- scanner.close(); scanner.close() | Reason:  Use only one  scanner.close(); |

**Important points:**

1. The System.out.println statement is using Celsius with a capital "C" which doesn't exist as a variable. Java is case-sensitive, so this will cause a compilation error. It should be lowercase celsius.

2. The scanner.close() method is called twice, which is unnecessary and could cause issues. It should only be called once.

**4.Write a Program on Fibonacci Seqquence where input taken from user.**

**CODE:**

**public class FibonacciSeries {**

**public static void main(String[] args) {**

**int n = 10;**

**int firstTerm = 0, secondTerm = 1;**

**System.out.println("Fibonacci Series up to " + n + " terms:");**

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

**System.out.print(firstTerm + ", ");**

**int nextTerm = firstTerm + secondTerm;**

**firstTerm = secondTerm;**

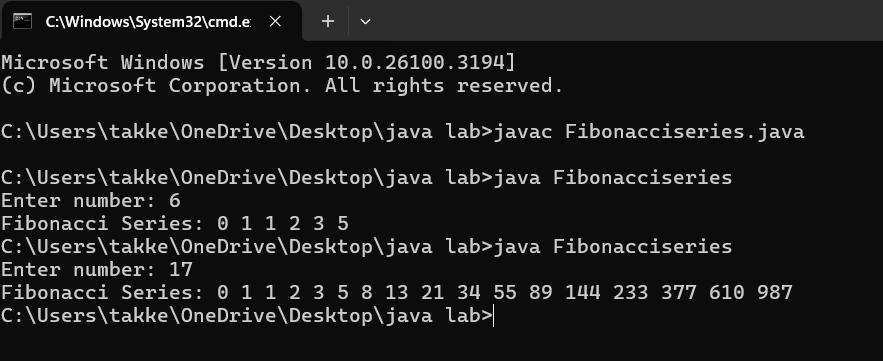
**secondTerm = nextTerm;**

**}**

**}**

**}**

**OUTPUT:**

ERRORS**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | int firstTerm;  Error: variables not initialized properly | should be initialized  int n = 10; |
| **2.** | int[] fibonacci = new int[n];  System.out.println(fibonacci[n + 1]); | invalid access  replace:  System.out.println("Fibonacci Series up to " + n + " terms:"); |

**IMPORTANT POINTS:**

1.Variables should be initialized properly

2.Accessing an array element outside its bounds.

3.Loop condition should be properly defined. No syntax error should be there.

**5.Write a program with java on How to calculate Area of Rectangle & Area of Triangle.**

**CODE FOR CALCULATING AREA OF RECTANGLE:**

**import java.util.Scanner;**

**public class Rectanglearea {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the length of the rectangle: ");**

**double length = scanner.nextDouble();**

**System.out.print("Enter the width of the rectangle: ");**

**double width = scanner.nextDouble();**

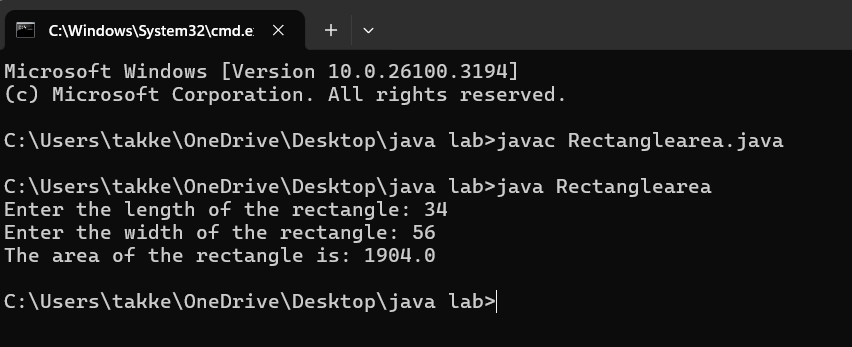
**double area = length \* width;**

**System.out.println("The area of the rectangle is: " + area);**

**}**

**}**

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | int length = 5.0; // Should be double | **Replace:** double length |
| **2.** | public class AreaOfRectangle {  public static void main(String[] args) | **Replace:**  Add flower braces at the end. |

**Important points:**

1. Trying to assign a double value to an int variable.

2. Incorrect placement of braces,should be placed correctly.

3.Should not give incorrect variables.

**CODE FOR CALCULATING AREA OF TRIANGLE:**

import java.util.Scanner;

public class heronstriangle {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Taking input from the user

System.out.print("Enter the length of side a: ");

double a = scanner.nextDouble();

System.out.print("Enter the length of side b: ");

double b = scanner.nextDouble();

System.out.print("Enter the length of side c: ");

double c = scanner.nextDouble();

// Calculate the semi-perimeter

double s = (a + b + c) / 2;

// Calculate the area using Heron's Formula

double area = Math.sqrt(s \* (s - a) \* (s - b) \* (s - c));

// Print the result

System.out.println("The area of the triangle is: " + area);

scanner.close();

}

}

**OUTPUT:**

A computer screen shot of a black screen

AI-generated content may be incorrect.

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICARION** |
| **1.** | Main class name:triangle kept instead of herons | **Replace :**main class name by heronstriangle |
| **2.** | (),; closing tags forgot for  **double a = scanner.nextDouble** | **Replace:**  **double a = scanner.nextDouble();** |

**IMPORTANT POINTS:**

**1.**Verify that the input values can form a valid triangle (i.e., the sum of any two sides must be greater than the third side).

**2.**Calculate the semi-perimeter ss using the formula:

s=a+b+c/2.

**Week-3**

1. **To create java program with following :**
2. **Create a class with name car**
3. **Create four attributes named Car\_colour,Car\_brand,Fuel\_type,mileage**
4. **Create three methods named Strat(),Stop(),Service()**
5. **Create three obj named Car1,Car2,Car3.**

**CODE FOR CREATING CAR CLASS:**

class car{

public String car\_color;

public String car\_brand;

public String fuel\_type;

public float mileage;

public void start(){

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

}

public void stop(){

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

}

public void service(){

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

}

public static void main(String [] args){

// object one creation

car car1= new car();

car1.car\_color="Red";

car1.car\_brand="Ferrari";

car1.fuel\_type="Petrol";

car1.mileage=75.69F;

//calling methods for object 1

car1.start();

car1.stop();

car1.service();

System.out.println("color of the car1 is "+car1.car\_color);

System.out.println("brand of the car1 is "+car1.car\_brand);

System.out.println("fuel type of the car1 is"+car1.fuel\_type);

System.out.println("mileage of the car1 is"+car1.mileage);

// object two creation

car car2= new car();

car2.car\_color="Blue";

car2.car\_brand="BMW";

car2.fuel\_type="Petrol";

car2.mileage=11.96F;

// calling methods for object 2

car2.start();

car2.stop();

car2.service();

System.out.println("color of the car2 is "+car2.car\_color);

System.out.println("brand of the car2 is "+car2.car\_brand);

System.out.println("fuel type of the car2 is "+car2.fuel\_type);

System.out.println("mileage of the car2 is"+car2.mileage);

//object three creation

car car3= new car();

car3.car\_color="black";

car3.car\_brand="Audi";

car3.fuel\_type="Diesel";

car3.mileage=11.76F ;

// calling methods for object 3

car3.start();

car3.stop();

car3.service();

System.out.println("color of the car3 is "+car3.car\_color);

System.out.println("brand of the car3 is "+car3.car\_brand);

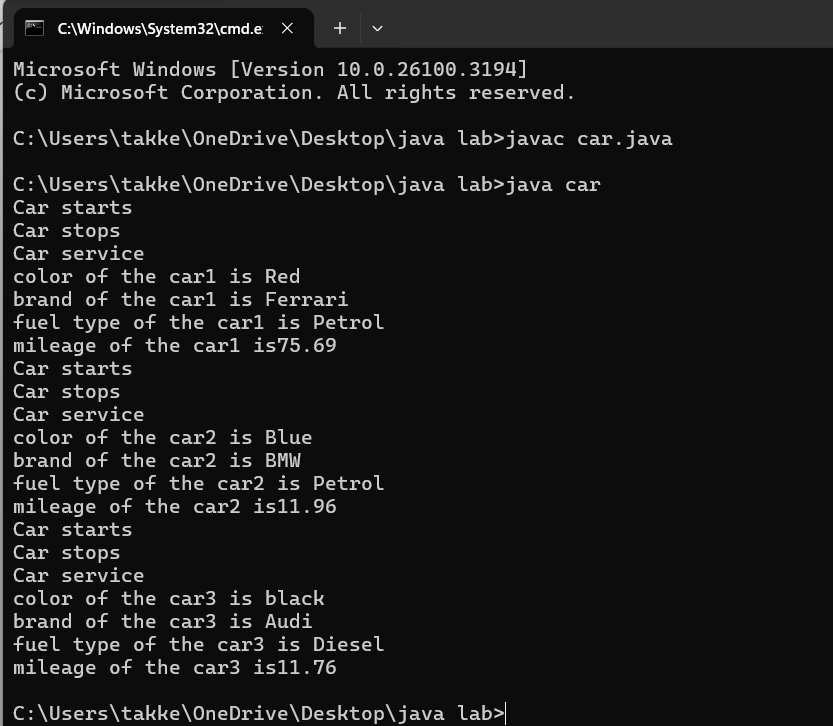
System.out.println("fuel type of the car3 is "+car3.fuel\_type);

System.out.println("mileage of the car3 is"+car3.mileage);

}

}

**OUTPUT:**

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | Error: car\_Color is undefined, should be car\_color | **Replace:** car\_Color with car\_color |
| **2.** | Missing semicolon after 'car2.start()' will cause an error | **Replace:** add semi colon at the end |

**CLASS DIAGRAM:**

|  |
| --- |
| car |
| car\_color:string  car\_brand:string  Fuel\_type:string  -Mileage:double |
| +start():void  +stop():void  +service():void |

**IMPORTANT POINTS:**

**1. Variable name mismatch**: The variable car\_Color in the code should be car\_color

2.**Incorrect variable name**: car1.car\_color is used when the actual variable is car1.car\_Color, which will cause an error due to case sensitivity.

3. **Missing Semicolon**: Forgetting to add a semicolon at the end of a statement will cause a compilation error.

**2.To create a class Bank Account with Methods deposit() and Withdrawal()**

**CODE:**

**import java.util.Scanner;**

**class BankAccount {**

**private float existing; // Class-level variable to store balance**

**private Scanner input; // Single Scanner instance for input**

**// Constructor**

**public BankAccount() {**

**input = new Scanner(System.in);**

**System.out.print("Enter existing amount in bank account: ");**

**this.existing = input.nextFloat();**

**}**

**// Deposit method**

**public void deposit() {**

**System.out.print("Enter amount to be deposited: ");**

**float deposit = input.nextFloat();**

**existing += deposit;**

**System.out.println("Existing amount now is: " + existing);**

**}**

**// Withdrawal method**

**public void withdrawal() {**

**System.out.print("Enter amount to be withdrawn: ");**

**float withdrawal = input.nextFloat();**

**if (existing < withdrawal) {**

**System.out.println("Not sufficient balance.");**

**} else {**

**existing -= withdrawal;**

**System.out.println("Remaining balance: " + existing);**

**}**

**}**

**// Main method**

**public static void main(String[] args) {**

**BankAccount customer1 = new BankAccount();**

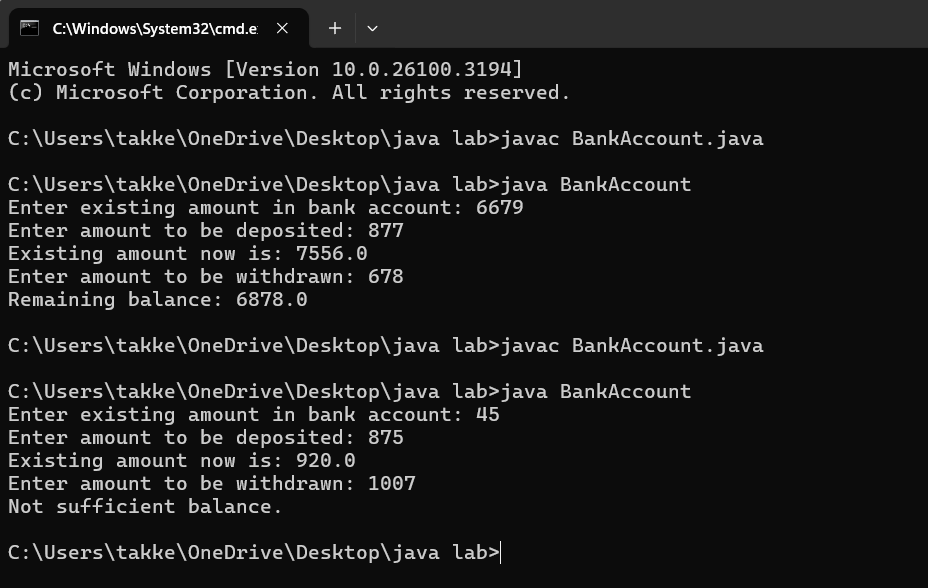
**customer1.deposit();**

**customer1.withdrawal();**

**}**

**}**

**OUTPUT:**

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | this.existing = int.nextFloat(); | this.existing = input.nextFloat(); |
| **2.** | public int deposit() is incorrect return method | REPLACE:public VOID deposit() |

**CLASS DIAGRAM:**

|  |
| --- |
| Bank Account |
| -balance:double |
| +Bank  Acccount(intitialBalance:double)  +deposit(amount:double):void  +withdrawal(amount:double):void |

**IMPORTANT POINTS:**

1.The balance should be a float or double to handle decimal values correctly, but it's declared as an int.

2. **Incorrect deposit method signature**: The method DEPOSIT()has an incorrect return type int(), while it should be void since it doesn't need to return any value.

3. **Fixed the return type of deposit**: Changed from int to void, as the method does not need to return anything.

**WEEK-4**

**AIM :: Write a java program with class named Book**

**a)a class should contain various attributes such as title,author,year of publication.**

**b)it should also contain a constructor with parameters which initializes title,author,year of publication.**

**c)create a method which displays the details of the book title ,author ,year of publication**

**Display the details of two books.**

**CODE:**

**class Book {**

**// Fields to hold book details**

**public String bookTitle;**

**public String bookAuthor;**

**public int bookYearOfPublication;**

**// Method to print book title**

**public void title() {**

**System.out.println("Book Title");**

**}**

**// Method to print book author**

**public void author() {**

**System.out.println("Book Year of Publishing");**

**}**

**// Main Method**

**public static void main(String[] args) {**

**// First book**

**Book book1 = new Book();**

**book1.bookTitle = "Atomic Habits";**

**book1.bookAuthor = "James Clear";**

**book1.bookYearOfPublication = 2018;**

**// Calling methods**

**book1.title();**

**book1.author();**

**// Printing book details**

**System.out.println("Book title is: " + book1.bookTitle);**

**System.out.println("Book author is: " + book1.bookAuthor);**

**System.out.println("Book year of publication is: " + book1.bookYearOfPublication);**

**// Second book**

**Book book2 = new Book();**

**book2.bookTitle = "South Pole Pig";**

**book2.bookAuthor = "James";**

**book2.bookYearOfPublication = 2014;**

**// Calling methods**

**book2.title();**

**book2.author();**

**// Printing book details**

**System.out.println("Book title is: " + book2.bookTitle);**

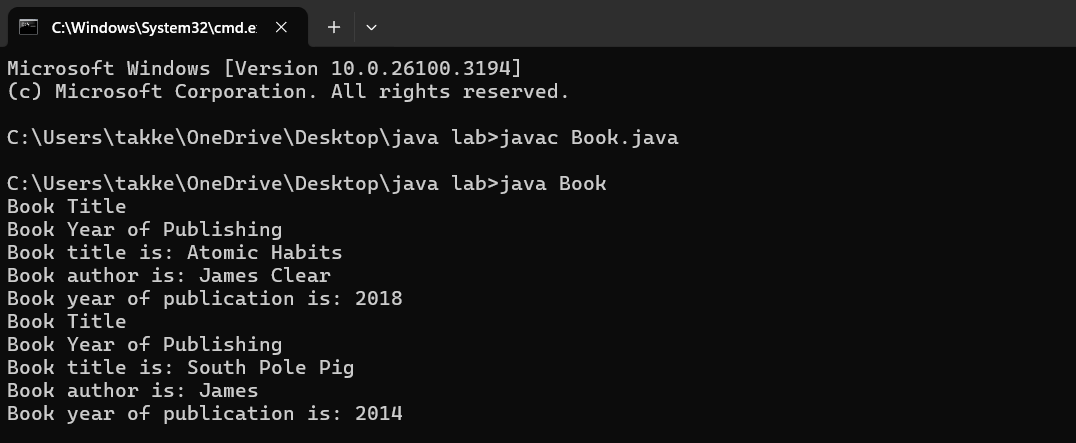
**System.out.println("Book author is: " + book2.bookAuthor);**

**System.out.println("Book year of publication is: " + book2.bookYearOfPublication);**

**}**

**}**

**OUTPUT:**

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | Not defining the function in a file. | To call the method we must define a function in a file. |
| **2.** | Two public class files should not be saved in the same file. | Two public class files should be saved in different files. |

**IMPORTANT POINTS:**

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

**CLASS DIAGRAM:**

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

**AIM :: Create a java program with class named ‘MyClass’ with static variable count of int type,initaialized to zero and a constant variable ‘pi’ or type double initialized to 3.14 as attributes of the class. Now define a constructor for “MyClass”that increments the count variable eachtime an object of MyClass is created.Finally print the final values of count and pi variables.**

**Create three objects and a constructor.**

**CODE:**

**class MyClass {**

**// Static variable to hold count of objects created**

**static int count = 0;**

**// Constant variable for pi, initialized to 3.14**

**final double pi = 3.14;**

**// Constructor**

**public MyClass() {**

**// Increment count every time an object is created**

**count++;**

**}**

**// Main method to create objects and print values**

**public static void main(String[] args) {**

**// Creating three objects of MyClass**

**MyClass object1 = new MyClass();**

**MyClass object2 = new MyClass();**

**MyClass object3 = new MyClass();**

**// Printing the final values of count and pi**

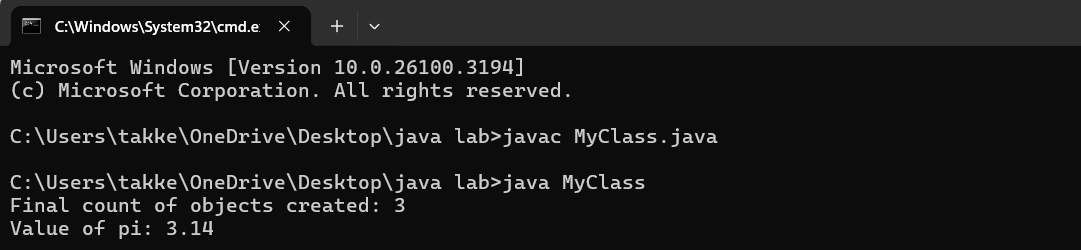
**System.out.println("Final count of objects created: " + count);**

**System.out.println("Value of pi: " + object1.pi); // Pi is constant, and we can access it via any object**

**}**

**}**

**OUTPUT:**

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | Not Putting the semi-colon after calling a function, | Put the semi-colon after calling a function. |
| **2.** | Not giving the indentation properly. | All the indentation must be correct to run the code correct |

**IMPORTANT POINTS:**

1. We must declare the initial value of the variable before declaring the final one.
2. Here the main objective is to increase the count according to the number of objects we make, i.e the count increases when the no.of objects are increasing.

**CLASS DIAGRAM:**

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

**LAB –5**

**AIM:: Create a calculator using the operator including addition, substraction, multiplication& divison using multilevel inheritance and display the desired output**

**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());

}

}

**OUTPUT:**

A screen shot of a computer

AI-generated content may be incorrect.

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1** | The class inherit should be renamed to Inherit | **public class Inherit {** |
| **2.** | The Calculator class is an empty base class: | **abstract class Calculator {**  **// Abstract method (optional)**  **public abstract int add(int a, int b);**  **}** |

**IMPORTANT POINTS:**

**1.Inheritance Structure:**

Simple extends Calculator, Super extends Simple, and Advanced extends Super, which shows a good hierarchical structure for building more complex functionality on top of simpler functionality.

**2.Use of Math.sqrt(a) in Super:**

The squareRoot method in the Super class is good because it provides an additional operation not found in Simple.

**3.Method Overriding and Polymorphism:**

Although you are not directly overriding methods (except in the Advanced class with divide), the class hierarchy could potentially be extended with polymorphism for more dynamic functionality.

**CLASS DIAGRAM:**

|  |
| --- |
| Calculator |
| -a:double |
| -b:double |

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

|  |
| --- |
| **Substraction** |
| +substract():double |

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

|  |
| --- |
| **Division** |
| +division():double |

**AIM:: Write a java program to create a baseclass bankaccount with methods deposit &withdrawal .create two subclasses savingsaccount &checking acccount which overrides the withdrawal method in each subclass to impose different withdrawal limits &fees.**

**CODE:**

import java.util.Scanner;

class bankingaccount {

public float existing; // Class-level variable to store balance

public Scanner input; // Single Scanner instance for input

// Constructor

public bankingaccount() {

input = new Scanner(System.in);

System.out.print("Enter existing amount in bank account: ");

this.existing = input.nextFloat();

}

// Deposit method

public void deposit() {

System.out.print("Enter amount to be deposited: ");

float deposit = input.nextFloat();

existing += deposit;

System.out.println("Existing amount now is: " + existing);

}

// Withdrawal method

public void withdrawal() {

System.out.print("withdraw amount: ");

}

}

class Savingaccount extends bankingaccount{

public void withdrawal(){

System.out.print("Enter amount to be withdrawn: ");

float withdrawal = input.nextFloat();

if (existing < withdrawal) {

System.out.println("Not sufficient balance.");

} else {

existing -= withdrawal;

System.out.println("Remaining balance: " + existing);

}

}

}

class Checkingaccount extends bankingaccount{

public void withdrawal(){

System.out.print("Enter amount to be withdrawn: ");

float withdrawal = input.nextFloat();

if (existing < withdrawal)

System.out.println("Not sufficient balance.");

} else {

existing -= withdrawal;

System.out.println("Remaining balance: " + existing);

}

}

}

public class Main {

public static void main(String args[]){

Savingaccount S1 = new Savingaccount();

Checkingaccount C1= new Checkingaccount();

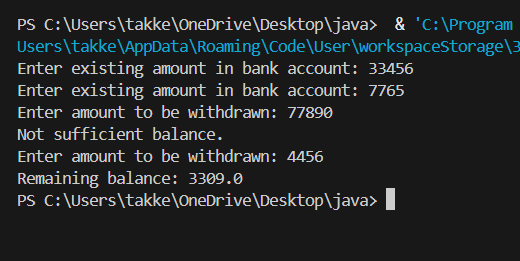
S1.withdrawal();

C1.withdrawal();

}

}

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| **1.** | USE OF CAPITAL B IN BANKING ACC | **USE bankingaccount** |
| **2.** | **Existing>withdrawal** | **Existing=withdrawal** |

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

**IMPORTANT POINTS:**

1. **Class Naming Conventions**: Always follow Java naming conventions for readability and maintainability.
2. **Resource Management**: Always close resources like Scanner to avoid memory leaks. Using try-with-resources is a good practice.
3. **Input Validation**: Consider adding error handling for user inputs to ensure the program does not crash on invalid input (e.g., using try-catch).
4. **Encapsulation**: Consider making the existing variable private and provide public getter and setter methods to access and modify it.

**AIM::** 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?

a. Truck should include an additional property capacity (in tons)

b. Create a showTruckdetails() method to display the truck’s capacity.

c. Write a constructor for Truck that initializes all properties

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.

**CODE:**

class Vehicle{

String brand;

int speed;

Vehicle(String brand,int speed){

this.brand=brand;

this.speed=speed;

}

void Details(){

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

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

}

}//End of super class

class CARS extends Vehicle{

int doors;

int capacity;

public CARS(String brand,int speed,int doors,int capacity){

super(brand, speed);

this.doors=doors;

this.capacity=capacity;

}

void cardetails(){

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

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

}

}//End of car sub-class

class Bikes extends Vehicle{

Boolean gears;

Bikes(String brand,int speed,Boolean gears){

super(brand, speed);

this.gears=gears;

}

void bikedetails(){

if (gears==true)

System.out.println("This bike has gears.");

else

System.out.println("This bike does not have gear system.");

}

}//End of bike sub-class

class Trucks extends Vehicle{

int tons;

Trucks(String brand,int speed,int tons){

super(brand, speed);

this.tons=tons;

}

void truckdetails(){

System.out.println("The capacity of truck is: "+tons);

}

}//End of truck sub-class

class Rent{

public static void main(String[] args){

CARS c=new CARS("Tayota",120,5,5);

c.cardetails();

c.Details();

Bikes b=new Bikes("KTM",80,true);

b.bikedetails();

b.Details();

Trucks t=new Trucks("TATA",100,1);

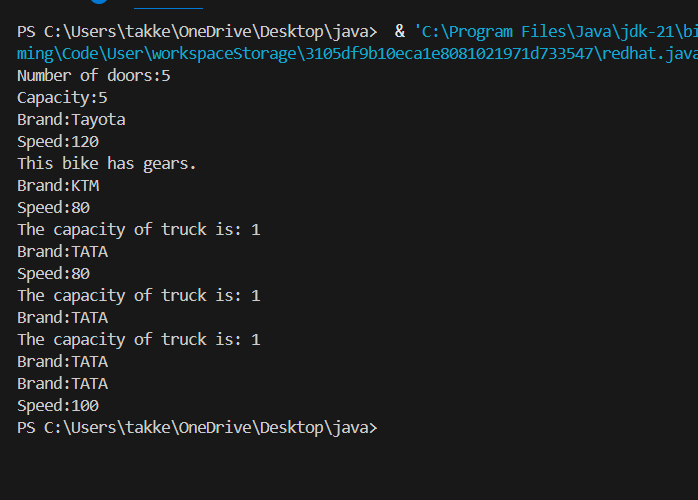
t.truckdetails();

t.Details();

}

}

**OUTPUT:**



**ERROR:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| 1 | Java class names should start with an uppercase letter | Change CARS to Cars, Bikes to Bikes, and Trucks to Trucks |
| 2 | Instead of checking if (gears == true) | you can simply use if (gears). |

**Class diagram:**

A diagram of a vehicle

AI-generated content may be incorrect.

**IMPORTANT PIONTS:**

1. **Encapsulation**: Consider making the fields in the Vehicle class private and providing public getter methods to access them. This promotes encapsulation.
2. **Code Reusability**: If there are common functionalities across classes, consider abstract classes or interfaces to promote code reusability.
3. **Constructor Overloading**: If you plan to have different constructors for different types of vehicles, consider using constructor overloading for better flexibility.

LAB-6

**AIM:: Write a java program to create a Vehicle class with a method in the Car subclass to provie specific information about a car.**

**CODE:**

class Vehicle {

    public void displayinfo() {

        System.out.println("Vehicle display info");

    }

}

class Car extends Vehicle {

    @Override

    public void displayinfo() {

        System.out.println("BMW is known for their superior engineering, innovative technology, luxurious comfort, and impressive resale value. BMW was founded in 1916. All BMW hatchbacks excel when it comes to low emissions and fuel efficiency.");

    }

}

class Main5 {

    public static void main(String[] args) {

        Vehicle myVehicle = new Vehicle();

        Car myCar = new Car();

        myVehicle.displayinfo();

        myCar.displayinfo();

System.out.println("name:sravya.T");

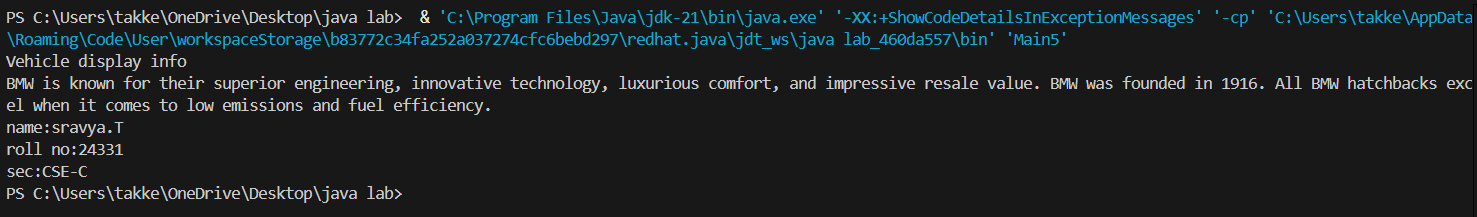
System.out.println("roll no:24331");

System.out.println("sec:CSE-C");

    }

}

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| 1. | Vehicle.java:12: error: class Main is public, should be declared in a file named Main.java  public class Main { | **Change the file into last class name** |
| 2. | Main method not found in class Vehicle, please define the main method as:  public static void main(String[] args) | **public static void main(String[] args);** |

**CLASS DIAGRAM:**

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

**IMPORTANT POINTS:**

1. The @Override annotation is used to indicate that the method is meant to override a method in the superclass.

2.In order to run the java program which involves the overriding concept then we have to name the file with last class name.

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

* **UG admission require a min of 60%**
* **PG admission require a min of 70%**

**CODE:**

class Students {

    String name;

    double percentage;

    public Students(String name, double percentage) {

        this.name = name;

        this.percentage = percentage;

    }

    void getEligibility() {

        System.out.println(name + " is eligible for admission.");

    }

}

class UG extends Students {

    public UG(String name, double percentage) {

        super(name, percentage);

    }

    @Override

    void getEligibility() {

        if (percentage >= 60) {

            System.out.println(name + " is eligible for UG admission.");

        } else {

            System.out.println(name + " is not eligible for UG admission.");

        }

    }

}

class PG extends Students {

    public PG(String name, double percentage) {

        super(name, percentage);

    }

    @Override

    void getEligibility() {

        if (percentage >= 70) {

            System.out.println(name + " is eligible for PG admission.");

        } else {

            System.out.println(name + " is not eligible for PG admission.");

        }

    }

}

public class CollegeAdmission {

    public static void main(String[] args) {

        UG ugStudent = new UG("Sravya", 65);

        PG pgStudent = new PG("Joshita", 75);

        ugStudent.getEligibility();

        pgStudent.getEligibility();

System.out.println("name:sravya.T");

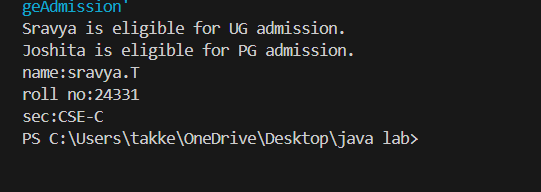
System.out.println("roll no:24331");

System.out.println("sec:CSE-C");

    }

}

**OUTPUT:**



**CLASS DIAGRAM:**

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

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| 1. | In the Students class, the field percentage is **private** by default (since no access modifier is specified), meaning it cannot be accessed by subclasses UG and PG. | To allow subclasses to access the percentage field, you should either make it protected or provide a getter method for it. |
| 2. | ou are using the super(name, percentage) constructor in both UG and PG classes correctly, | Let's change the percentage field to be protected so that subclasses can access it directly, or alternatively, create a getPercentage() method. |

**IMPORTANT POINTS:**

**Access Modifiers**:

In this case, percentage was set to **default** visibility (which means it’s package-private). By changing it to protected, the subclasses UG and PG are able to access it directly.

**Method Overriding**:

* You’ve correctly overridden the getEligibility method in the UG and PG classes. This allows you to provide specific behavior for these subclasses.

**Constructor Call**:

* The use of super(name, percentage) in the constructors of UG and PG is correct as it calls the constructor of the superclass (Students).

**Polymorphism**:

* When you call getEligibility() on objects ugStudent and pgStudent, the correct overridden version of the method is invoked due to polymorphism. This is a great demonstration of inheritance and method overriding.

**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:**

class Calculator {

public void add(int a, int b) {

System.out.println("Sum of " + a + " and " + b + " is: " + (a + b));

}

public void add(double a, double b) {

System.out.println("Sum of " + a + " and " + b + " is: " + (a + b));

}

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

System.out.println("Sum of " + a + ", " + b + ", and " + c + " is: " + (a + b + c));

}

public static void main(String[] args) {

Calculator calc = new Calculator();

calc.add(5, 10);

calc.add(3.5, 2.5);

calc.add(1, 2, 3);

System.out.println("name:sravya.T");

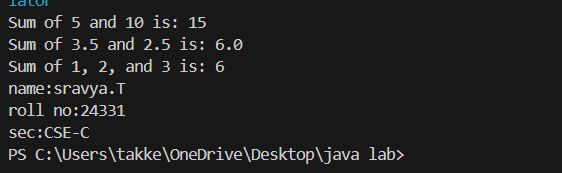
System.out.println("roll no:24331");

System.out.println("sec:CSE-C");

}

}

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROR MESSAGE** | **ERROR RECTIFICATION** |
| 1. | **Method Signature Ambiguity :**If you mix int and double types in certain combinations when overloading methods, | public void add(double a, int b) {  System.out.println("Sum of " + a + " and " + b + " is: " + (a + b));  } |
| 2. | **Implicit Type Casting and Method Overloading**: If you attempt to mix int and double in a way that causes Java to need type casting, sometimes it could be ambiguous | calc.add(5, 10.0); |

**CLASS DIAGRAM:**

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

**IMPORTANT POINTS:**

**Method Overloading**:

Java allows overloading methods, which means you can define multiple methods with the same name but different parameter lists.

**Avoid Ambiguity**:

While method overloading is a great feature, it can sometimes lead to ambiguity when methods have similar signatures.

**Clarity in Method Naming**:

If you have several similar methods, consider naming them in a way that makes their intent clearer. For example, if the methods are intended for int and double, rather than using the same name (add), you could have addIntegers, addDoubles, and addThreeIntegers.

**No Return Type**:

Your methods are correctly defined with void, as they are printing the results.

**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:**

class Shape {

return side \* side;

}

public double calculateArea(double length, double width) {  
 return length \* width;  
}

}

class Circle extends Shape {

@Override

public double calculateArea(double radius) {

return Math.PI \* radius \* radius;

}

}

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

Shape shape = new Shape();

Circle circle = new Circle();

System.out.println("Area of Square with side 5: " + shape.calculateArea(5));  
 System.out.println("Area of Rectangle with length 4 and width 6: " + shape.calculateArea(4, 6));  
 System.out.println("Area of Circle with radius 3: " + circle.calculateArea(3));

System.out.println("name:sravya.T");

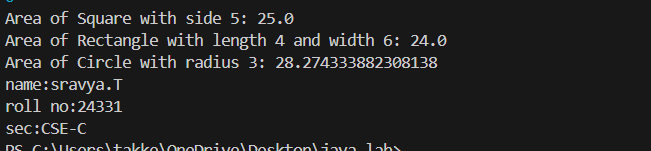
System.out.println("roll no:24331");

System.out.println("sec:CSE-C");

}

}

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **ERROOR MESSAGE** | **ERROR RECTIFICATION** |
| 1. | **Error in Circle class:**  The Circle class should override the calculateArea method properly. | **Override the method in Circle** to calculate the area of a circle |
| 2. | **Incorrect Shape Object Handling in Main**  In the Main class, you're calling shape.calculateArea(5). | **Use inheritance** properly to calculate areas of different shapes. |

**IMPORTANT POINTS:**

**Inheritance**:

In this case, Circle inherits from Shape, and the calculateArea method is **overridden** in the Circle class to calculate the area of a circle.

**Method Overloading** occurs when you define multiple methods with the same name but different parameter lists (for example, calculating area for squares, rectangles, and circles using different parameters).

**Method Overriding** :Here, Circle overrides the calculateArea method to calculate the area of a circle specifically.

**Polymorphism**:

The way the method calculateArea is used in the Shape and Circle classes demonstrates **polymorphism**.

**CLASSDIAGRAM**:

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

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

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