**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**LAB REPORT**



**Department of Computer Science Engineering**

**Amrita School of Computing**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Name: N.Abhishek**

**Verified By Roll No: 24232**

INDEX

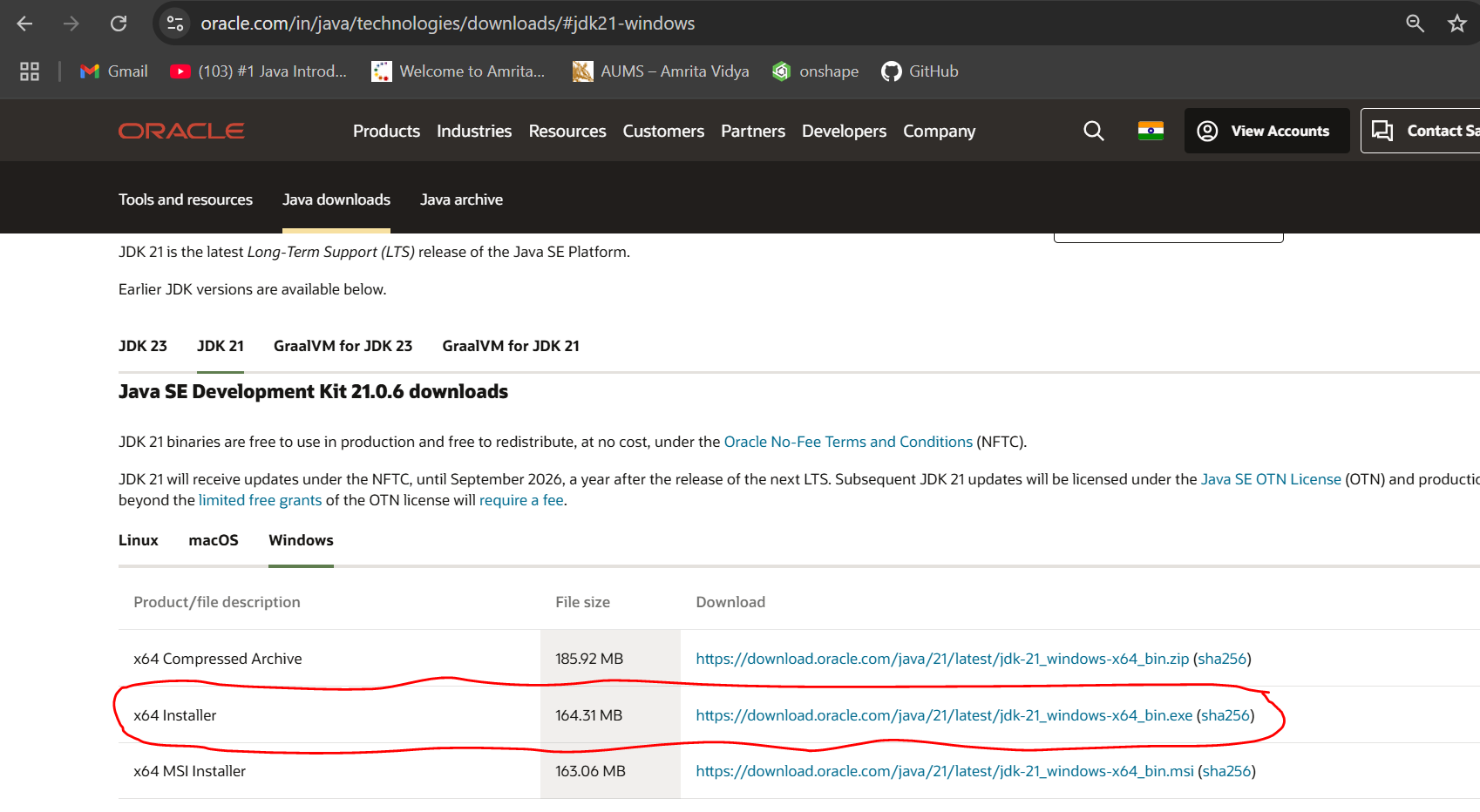
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S no | **Title** | **Page no** | **Date** | **Signature** |
|  | **WEEK -1** |  | **24/01/2025** |  |
| **1** | Explain the process of download & Installation of JDK | **10** |  |  |
| **2** | Write a program to print Student’s name, Roll no, Section | **11** |  |  |
|  | **WEEK-2** |  | **31/01/2025** |  |
| **1** | Write a simple java program to calculate factorial of a number | **13** |  |  |
| **2** | Write a simple java program to find the simple interest by taking iinputs from the user | **14** |  |  |
| **3** | Write a program to calculate the Fibonacci sequence and take the input from the user | **15** |  |  |
| **4** | Write a program to find the area of triangle using hereon’s formula  . | **17** |  |  |
| **5** | Write a program to convert temperature from celssius to Fahrenheit | **18** |  |  |
| **6** | Write a program to find the area of rectangle | **19** |  |  |
| **7** | Write a program to find the area of triangle by using the herons formula | **20** |  |  |
|  | **WEEK-3** |  | **07/02/2025** |  |
| **1** | **Write a java program with following instructions**  **a) Create a class with name car**  **b)Create four attributes named, car colour, car brand , fuel type , mileage.**  **c)Create 3 methods named start , stop, service .**  **d)Create 3 objects named c1,c2,c3**  **e)Create a constructor with parameters , car colour , car brand , fuel type and mileage.** | **21** |  |  |
| **2** | **create a class named bankaccount with method deposit and with draw where the deposit method should accepts a parameter and when this method is called the deposited amount should be current balance .In addition to that when a withdraw method is called it has to verify whether withdraw amount is less than the current balance .If not display a message saying insufficient funds.Use the constructer to display the details of the customer (Customer name,account number , IFSC,branch) .Also create two customer objects c1,c2** | **26** |  |  |
|  | **WEEK-4** |  | **21/02/2025** |  |
| **1** | **Write a java program with class named book the class should contain various attributes such as title ,author, year of publication .It should also contain a constructor with parameters with initializes title ,author and year of publication . create a method which displays the details of the book (display the details of two book i.e, create two books and objects with details).** | **30** |  |  |
| **2** | **write a java program to create a class named myclass with a static variable count of int type and initialize to zero and a constant variable pie of double data type ,initialize to 3.1415 as attributes of that class now define a constructor for my class that increments the count variables each time an object of my class is created variable each time an object of myclass is created. Finally print the final values of count and pie variables.** | **32** |  |  |
|  | WEEK-5 |  | **28/02/2025** |  |
| **1** | create a calculator using the operations including addition, subtraction, multiplication and division using multilevel inheritance and display the desired output . | **35** |  |  |
| **2** | A vehicle rental company wants to develop a system that maintains information about different types of vehicles available for rent. The company rents out cars and bikes and they need a program to store details about each variable such as brand and speed.  1. Cars should have an additional property: numbers of doors, seating capacity.  2. Bikes should have a property indicating whether they have gears or not.  3. The system should also include a function to display details about each vehicle & indicates when a vehicle is starting.  4. Every class should have constructor.  QUESTION:  1. Which OOP concept is used in the above program? Explain why it is useful in this scenario.  2. If the company decides to add a new type of vehicle: Truck, how would you modify the program?  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. | **39** |  |  |
| **3** | Implement the truck class and update the main method to create a truck object & also create an object for car & bike sub classed. Finally display its details. | **40** |  |  |
|  | WEEK -6 |  | **07/03/2025** |  |
| **1** | **Write a Java program to create a Vehicle class with a method displayInfo(). Override this method in the Car subclass to provide specific information about a car.** | **46** |  |  |
| **2** | A college is developing an automated admission system that verifies students' eligibility for undergraduate (UG) and postgraduate (PG) programs. Each program has different eligibility criteria based on the students' percentage in their previous qualifications.  (i)UG admissions require a minimum of 60%.  (ii)PG admissions require a minimum of 70% | **48** |  |  |
| **3** | Create a Calculator class with overloaded methods to perform addition:  (i) Add two integers.  (ii) Add two doubles.  (iii) Add three integers | **52** |  |  |
| **4** | Create a Calculator class with overloaded methods to perform addition:  (i) Add two integers.  (ii) Add two doubles.  (iii) Add three integers | **54** |  |  |
|  | WEEK-7 |  | **28/03/2025** |  |
| **1** | **Write a Java program to create an abstract class Animal with an abstract method called sound(). Create subclasses Lion and Tiger that extend the Animal class and implement the sound() method to make a specific sound for each animal.** | **57** |  |  |
| **2** | **Write a Java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extend the Shape3D class and implement the respective methods to calculate the volume and surface area of each shape** | **60** |  |  |
| **3** | **Write a java program using an abstract class to define a method for pattern printing Create an abstract class named pattern printer with an abstract method printpattern(int n) and a concrete method to display the pattern title.**  **Implement two subclasses:**  **1) Star pattern - Prints a right-angled triangle of stars(\*).**  **2) Number pattern - Prints a right- angled triangles of increasing numbers.**  **In the main() method, create Objects**  **Star Pattern Number pattern**  **\* 1**  **\*\* 1 2**  **\*\*\* 1 2 3**  **\*\*\*\* 1 2 3 4**  **\*\*\*\*\* 1 2 3 4 5** | **62** |  |  |
|  | WEEK-8 |  | **11/04/2025** |  |
| **1** | 1. **Write a Java program to create an interface Shape with the getPerimeter() method. Create three classes Rectangle, Circle, and Triangle that implement the Shape interface. Implement the getPerimeter() method for each of the three classes.** | **66** |  |  |
| **2** | **Write a Java program to create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play() method to play the respective sports.** | **68** |  |  |
|  | **WEEK-9** |  | **12/04/2025** |  |
| **1** | Write a java program to create a method that takes integer as parameter and throws an exception if the number is even. | **71** |  |  |
| **2** | Write a java program to create a method that reads a file and throws an exception if the file is not found. | **72** |  |  |
| **3** | write a java program to handle an arthematic exception using try catch finally. | **74** |  |  |
| **4** | Write a java program to simulate a University system using inner classes.   1. Create an outer class named university with a variable universityname. 2. Inside it, define two non-static inner classes: 3. Department- with variables like deptname and deptcode, and a method to display department details. 4. Student: with variables like studentname and rollnumber, and a method to display student details. 5. Create on object for each class and call their methods to display their details along with the universityName; |  |  |  |
|  | WEEK-10 |  | **25/04/2025** |  |
| **1** | Write a java program to generate a password for a student using his/her initials and age. the password displayed should be the string consists of first character of first name, middle name, last name with age. | **78** |  |  |
| **2** | Design and implement a Java program that will do the following operations to this string "Welcome! You are practicing strings concept."  a. convert all alphabets to capital letters and print out the result.  b. convert all alphabets to lower-case letters and print out the result.  c. print out the length of the string.  d. print out the index of Course. | **80** |  |  |
| **3** | Implement a java program using the below array methods.  a. Sorting the elements (numbers and Strings) of an array.  b. convert the array elements into string.  c. fill the part of an array.  d. copy the elements of one array into another. | **81** |  |  |
| **4** | Implement a java program using the below Array List methods.  a. insert an element at particular index in the array list.  b. Modify an element in the array list.  c. Access an element from the array list.  d. Remove an element from the array list.  e. clear the elements from the array list. | **83** |  |  |

**WEEK-1**

**Aim: To explain the process of Installing JDK (Java Development Kit)**

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

1. **Download JDK:**
   * Go to the Oracle JDK download page in your web browser and click on JDK-21 version which is Long term support (LTS) version.
   * Click on the download link for your operating system (Windows, macOS, or Linux).



1. **Install JDK:**
   * Once downloaded, run the installer.
   * Follow the instructions and keep clicking "Next" until it's done.
2. **Set Environment Variables (Windows):**
   * For copying of path, go to c drive in file explorer
   * Then click on program files, in program files we can find the Java file click on it the in java file open bin file. Then click on the top of the file location then there we can select the file path location then copy it.
   * 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**.
   * Click **New** under **System Variables**:
     + **Set Variable name as:** java\_home
     + **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)  
       
       
       
     

**Checking of JDK Version:**

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



**Aim: To Implement a Java Program to print basic details of a Student**

Write your code in Notepad and execute in cmd prompt

**Important Points:**

* + - * **Understand the syntax of a class.**

class Main {

public static void main(String[] args) {

System.out.println(" ");

}

}

**CODE:**

**The file was saved with Main.java**

class Main

{

public static void main(String[] args)

{

// Printing my basic details

System.out.println("Name:N.ABHISHEK");

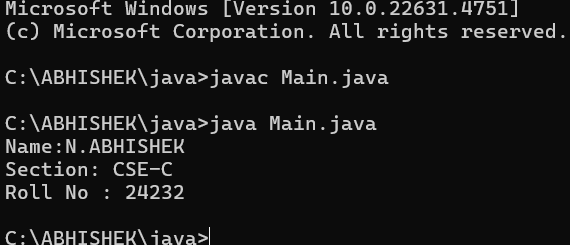
System.out.println("Section: CSE-C");

System.out.println("Roll No : 24232");

}

}

**Output:**

****

**Week 2**

1. **Aim: To implement Simple Java Program for finding simple interest by taking input from**

**User**

**Code:**

**import java.util.Scanner;**

**public class Si {**

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

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

**// Taking user inputs**

**System.out.print("Enter Principal amount: ");**

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

**System.out.print("Enter Rate of Interest (per annum): ");**

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

**System.out.print("Enter Time (in years): ");**

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

**// Calculate Simple Interest**

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

**// Display Output**

**System.out.println("Principal Amount: " + principal);**

**System.out.println("Rate of Interest: " + rate + "% per annum");**

**System.out.println("Time Period: " + time + " years");**

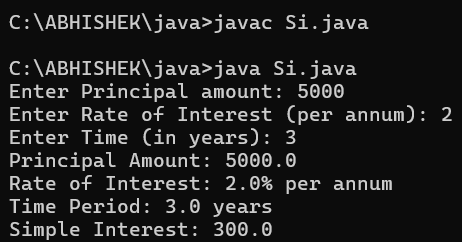
**System.out.println("Simple Interest: " + simpleInterest);**

**scanner.close();**

**}**

**}**

**Output**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **rectification** |
| **1** | **Syntax error** | **Did’nt declare variable type** | **Declared correct type** |
| **2** | **Syntax error** | **{ missing** | **{ added** |

**Important Points**

* The program uses Scanner to take input from the user for principal, rate of interest, and time, allowing dynamic input during runtime.
* It calculates simple interest using the standard formula: (Principal × Rate × Time) / 100.
* The program prints all input values along with the calculated simple interest, making the output user-friendly and easy to understand.
* It includes scanner.close() to properly release the Scanner resource, which is a good coding practice.

**2.Aim: To Write a simple program to calculate factorial of a number and read the**

**input from user**

**code:**

**import java.util.Scanner;**

**public class FactorialCalculator {**

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

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

**System.out.print("Enter a number: ");**

**int number = scanner.nextInt();**

**long factorial = 1;**

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

**factorial \*= i;**

**}**

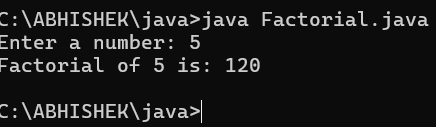
**System.out.println("Factorial of " + number + " is: " + factorial);**

**scanner.close();**

**}**

**}**

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| **1** | **Missing import statement** | **Not importing packages** | **Packages imported** |
| **2** | **Logical error** | **Wrong formula** | **Formula rectified** |

**Important Points**

* The program uses Scanner to take an integer input from the user, allowing runtime interaction.
* It calculates the factorial of the entered number using a for loop and multiplication logic.
* The result is displayed clearly with a descriptive message showing the original number and its factorial.
* The scanner.close() method is used at the end to release the input resource, following good programming practice.

**3.Aim: To Write a program to to calculate the fibonacii sequence and take the input from user**

**Code:**

**import java.util.\*;**

**class fibo**

**{**

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

**{**

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

**int num;**

**int f3;**

**int f1 = 0;**

**int f2 = 1;**

**int i = 2;**

**System.out.print("Enter a number:");**

**num = sc.nextInt();**

**System.out.println(f1);**

**System.out.println(f2);**

**while(i<num)**

**{**

**f3 = f1+f2;**

**f1 = f2;**

**f2 = f3;**

**System.out.println(f3);**

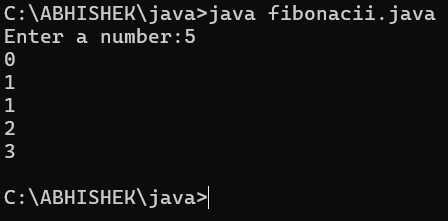
**i = i+1;**

**}**

**}**

**}**

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| **1** | **Logical error** | **Incorrect formula** | **Formula rectified** |
| **2** | **Run-time error** | **Incorrect path** | **Added correct path** |

**Important Points**

* The program uses Scanner to take an integer input from the user, which specifies how many terms of the Fibonacci series to print.
* It initializes the first two Fibonacci numbers (f1 = 0, f2 = 1) and prints them before entering the loop.
* A while loop is used to generate and print the remaining terms of the Fibonacci series by summing the previous two terms.
* The loop continues until the specified number of terms is printed, demonstrating iterative logic for sequence generation.

**4.Aim:Write a java program to convert temperature from Fahrenheit to celsius**

**Code:**

**import java.util.Scanner;**

**public class TemperatureConverter {**

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

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

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

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

**double celsius = (fahrenheit - 32) \* 5 / 9;**

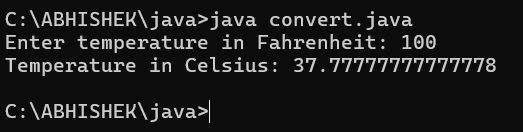
**System.out.println("Temperature in Celsius: " + celsius);**

**scanner.close();**

**}**

**}**

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **rectification** |
| **1** | **Syntax error** | **Missing ”** | **“ is added** |
| **2** | **Missing variable type** | **Didn’t declare variable** | **Declared variable** |

**Important Points**

* The program uses Scanner to take temperature input from the user in Fahrenheit, allowing real-time interaction.
* It converts the Fahrenheit value to Celsius using the formula: (fahrenheit - 32) \* 5 / 9.
* The result is printed with a clear message displaying the temperature in Celsius.
* The scanner.close() statement is included to properly close the input resource, following good programming practice.

**5.Aim: To Write a java program to convert temperature from Celsius to Fahrenheit**

**Code:**

**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: ");**

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

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

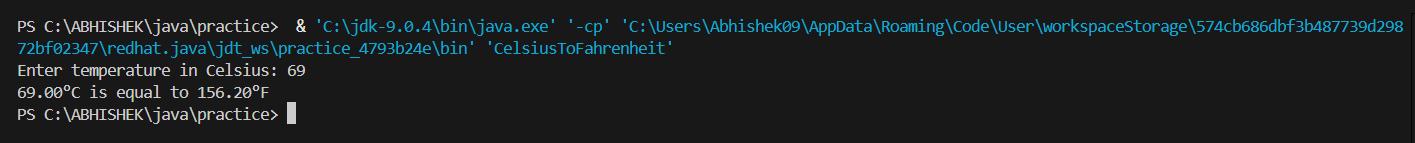
**System.out.printf("%.2f°C is equal to %.2f°F", celsius, fahrenheit);**

**scanner.close();**

**}**

**}**

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| **1** | **Runtime error** | **Incorrect path selection** | **Correct path added** |
| **2** | **Logical error** | **Incorrect logic** | **Correct logic** |

**Important Points**

* The program uses Scanner to take temperature input from the user in Celsius.
* It converts the Celsius temperature to Fahrenheit using the formula (celsius \* 9/5) + 32.
* The output is formatted to show the temperatures with two decimal places for better readability.
* The scanner.close() method is called to properly close the input resource, following good coding practice.

**6.Aim: To Write a simple program to find the area of rectangle:**

**Code:**

**import java.util.Scanner;**

**public class Area {**

**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;**

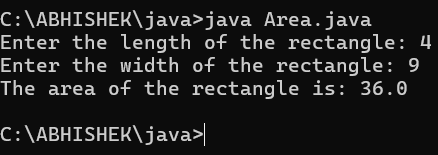
**System.out.println("The area of the rectangle is: " + area);**

**scanner.close();**

**}**

**}**

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| **1** | **Syntax error** | **Semi colon missing** | **Semi colon added** |
| **2** | **Missing import error** | **Import package missing** | **Import package added** |

**Important Points**

* The program uses Scanner to take user input for the length and width of a rectangle.
* It calculates the area of the rectangle using the formula length × width.
* The calculated area is displayed clearly to the user.
* The program closes the Scanner resource after use to ensure proper resource management.

**7.Aim: To Write a program to find the area of triangle by using heron’s formula take the input from the user**

**Code:**

import java.util.Scanner;

public class TriangleAreaCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the first side of the triangle: ");

double a = scanner.nextDouble();

System.out.print("Enter the second side of the triangle: ");

double b = scanner.nextDouble();

System.out.print("Enter the third side of the triangle: ");

double c = scanner.nextDouble();

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

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

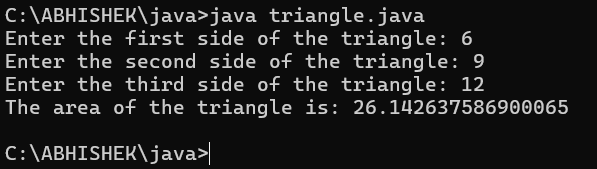
System.out.println("The area of the triangle is: " + area);

scanner.close();

}

}

**OUTPUT:**



|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| **1** | **Logical error** | **Incorrect formula** | **Formula rectified** |
| **2** | **Name error** | **Undeclared variable** | **Variable declared** |
|  |  |  |  |

**Important Points**

* The program uses Scanner to take the lengths of the three sides of a triangle as input from the user.
* It calculates the semi-perimeter s using the formula (a + b + c) / 2.
* The area is computed using **Heron's formula**:

area=s(s−a)(s−b)(s−c)\text{area} = \sqrt{s(s - a)(s - b)(s - c)}

* The program displays the calculated area and closes the Scanner resource to prevent resource leaks.

WEEK – 3

**1.Aim:** To Write a code for method attributes of a car for three objects.

Aim:writing java code using method attributes.

Code:

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="BMW";

car1.fuel\_type="Petrol";

car1.mileage=62.5F;

//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="Audi";

car2.fuel\_type="Petrol";

car2.mileage=64.5F;

// 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="Yellow";

car3.car\_brand="Benz";

car3.fuel\_type="Diesel";

car3.mileage=66.5F;

// 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);

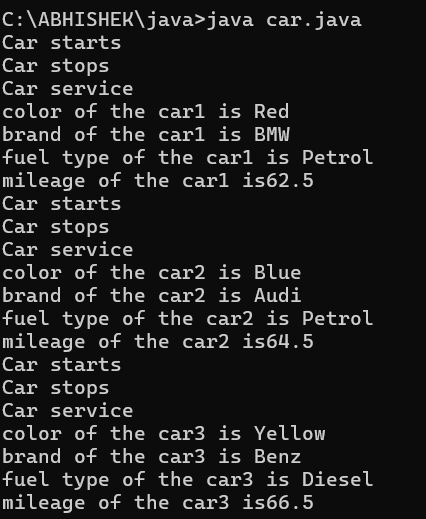
System.out.println("fuel type of the car3 is "+car3.fuel\_type);

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

}

}

OUTPUT:



|  |  |  |  |
| --- | --- | --- | --- |
| Error type | Description | impact | Solution |
| Syntax error | Typo in the variable name mileage in the constructor | Compilation error ;  Constructor won’t work | Corrected to mileage |
| Mthod signature | Getter methods color,brand,fuel,mile had parameters but were expected to return instance values | Causes incorrect behaviour when  Calling methods | Removed parameters from method definition |

CLASS DIAGRAM :

|  |
| --- |
| Main |
| -car\_color:string  -car\_brand: string  - fuel\_type:string  -mileage:int |
| +color():String  +brand():String  +fuel():String  +mile():int |

**Important Points**

* The car class defines attributes like car\_color, car\_brand, fuel\_type, and mileage, along with methods start(), stop(), and service().
* Three objects (car1, car2, car3) of the car class are created, each with different attribute values assigned.
* For each car object, the methods start(), stop(), and service() are called to simulate car actions.
* The program prints the details (color, brand, fuel type, mileage) of each car object, demonstrating object-oriented principles such as encapsulation and multiple object creation.

**2.Aim:** To writing a java code for the bank account details

Code:

class Bank\_Account {

private String accname;

private int acno;

private float balance;

public Bank\_Account(String accname, int acno, float balance) {

this.accname = accname;

this.acno = acno;

this.balance = balance;

}

public void withdraw(int amount) {

if (amount <= balance) {

balance -= amount;

System.out.println("Withdrawal of " + amount + " successful. Remaining balance: " + balance);

} else {

System.out.println("Insufficient balance for " + accname);

}

}

public void deposit(int amount) {

balance += amount;

System.out.println("Deposit of " + amount + " successful. Updated balance: " + balance);

}

public void displayDetails() {

System.out.println("Account Name: " + accname);

System.out.println("Account Number: " + acno);

System.out.println("Balance: " + balance);

}

public static void main(String[] args) {

Bank\_Account s = new Bank\_Account("Anil", 34571, 25000);

s.withdraw(20000);

s.deposit(2000);

s.displayDetails();

System.out.println();

Bank\_Account s1 = new Bank\_Account("Jeevan", 333226, 25000);

s1.withdraw(2500);

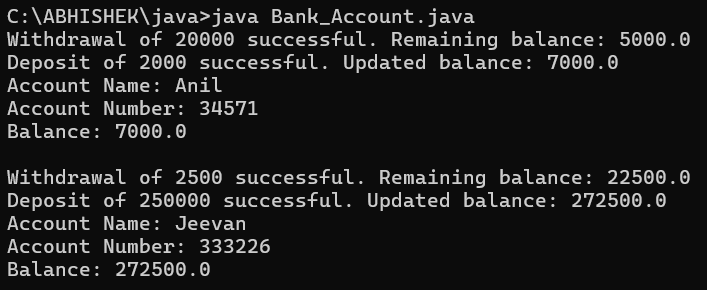
s1.deposit(250000);

s1.displayDetails();

}

}

Output:



|  |  |  |  |
| --- | --- | --- | --- |
| Error type | subscription | Impact | Solution |
| Missing balance dispaly | Withdraw and deposit methods update the balance but not dispaly | User cannot see the current balance after transactions | Add system.out.println on both methods |
| No getter methods | No method to retrieve account details | Cannot account details outside of the class | Add getter methods like getBalance(), getAccName(), etc. |

CLASS DIAGRAM:

|  |
| --- |
| Bank\_Account |
| -accname:String  -acno:int  -balance:float |
| +withdraw(amount:int):void  +deposit(amount:int):void  +displaydetails():void |

**Important Points**

* The Bank\_Account class uses private variables for account name, account number, and balance, ensuring encapsulation.
* It has a constructor to initialize the account details when an object is created.
* The withdraw method checks if there is sufficient balance before deducting the amount and provides feedback accordingly.
* The deposit method adds the given amount to the balance and confirms the successful deposit.
* The displayDetails method prints the current state of the account including name, number, and balance.
* The main method creates two Bank\_Account objects and demonstrates withdrawing, depositing, and displaying account details.

WEEK 4:

**1.AIM:** TO WRITE A JAVA PROGRAM WITH CLASS NAMED “Book”. THE CLASS SHOUKD CONTAIN VARIOUS ATTRIBUTES SUCH AS TITLE, AUTHOR, YEAR OF PUBLICATION. IT SHOULD ALSO CONTAIN A CONSTRUCTOR WITH PARAMETERS WHICH INITIALIZES TITLE, AUTHOR, YEAR OF PUBLICATION AND CREATE A METHOD WHICH DISPLAYS THE DETAILS OF 2 BOOKS.

CODE:

public class Book {

public String title;

public String author;

public int year;

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

this.title = title;

this.author = author;

this.year = year;

}

public void displayDetails() {

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

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

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

}

public static void main(String[] args) {

Book b1 = new Book("Math", "Ramanujan", 1950);

Book b2 = new Book("Physics", "CV Raman", 1960);

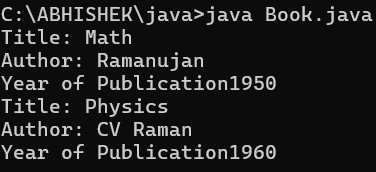
b1.displayDetails();

b2.displayDetails();

}

}

OUTPUT:



|  |  |  |  |
| --- | --- | --- | --- |
| ERROR TYPE | DESCRIPTION | IMPACT | SOLUTION |
| Compilation error | Missing semicolon | Code will not compile | Add semicolons at the end of the lines |
| Logical error | Missing spacing in the print statements | Output may be concatenated improperly | Add a space after year of publication in the display details () method |

CLASS DIAGRAM:

|  |
| --- |
| Book |
| - title: String  - author: String  - year: int |
| Book(String, String, int)  + displayDetails(): void |

**Important Points**

* The Book class has three public attributes: title, author, and year.
* It includes a constructor to initialize these attributes when a new Book object is created.
* The displayDetails method prints the book’s title, author, and year of publication.
* The main method creates two Book objects with different details and calls displayDetails for each to show their information.

**2. AIM:** TO WRITE A JAVA PROGRAM WITH CLASS NAMED “MyClass” WITH A STATIC VARIABLE COUNT OF INT TYPE. INTIALIZE IT TO ZERO AND A CONSTANT VARIABLE “Pi” OF TYPE DOUBLE INITIALIZED TO “3.14” AS ATTRIBUTES OF THAT CLASS. NOW DEFINE A CONSTRUCTOR FOR “MyClass”, THAT INCREMENTS THE COUNT VARIABLE EACH TIME AN OBJECT OF “MyClass” IS CREATED. FINALLY, PRINT THE FINAL VALUES OF ‘COUNT’ AND ‘PI’ VARIABLES AND CREATE 3 OBJECTS.

CODE:

public class MyClass {

static int count = 0;

static final double pi = 3.14;

MyClass() {

count++;

}

public static void main(String[] args) {

MyClass obj1 = new MyClass();

MyClass obj2 = new MyClass();

MyClass obj3 = new MyClass();

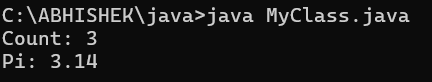
System.out.println("Count: " +count);

System.out.println("Pi: " +pi);

}

}

OUTPUT:



|  |  |  |  |
| --- | --- | --- | --- |
| Error type | Description | Impact | Solution |
| Logical error | Missing space in the println statement for count and pi | Output may be concatenated improperly. | Add a space before +count and+pi in the println statement. |

CLASS DIAGRAM:

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

**Important Points**

* The variable count is declared as static, so it is shared among all instances of the class and keeps track of how many objects are created.
* The constant pi is declared as static final, meaning its value is fixed and shared by all objects.
* Each time the constructor MyClass() is called, count is incremented by 1, tracking the number of objects created.
* The main method creates three objects of MyClass and then prints the total count of objects created and the value of pi.

**WEEK-5**

**1)Aim :** To create a calculator using the operations including addition, subtraction, multiplication and division using multilevel inheritance and display the desired output .

**Program :**

import java.util.Scanner;

class Addition {

int add(int a, int b) {

return a + b;

}

}

class Subtraction extends Addition {

int subtract(int a, int b) {

return a - b;

}

}

class Multiplication extends Subtraction {

int multiply(int a, int b) {

return a \* b;

}

}

class Division extends Multiplication {

double divide(int a, int b) {

if (b != 0) {

return (double) a / b;

} else {

return Double.NaN;

}

}

}

public class Calculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Division calc = new Division();

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

int num1 = scanner.nextInt();

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

int num2 = scanner.nextInt();

System.out.println("Addition: " + calc.add(num1, num2));

System.out.println("Subtraction: " + calc.subtract(num1, num2));

System.out.println("Multiplication: " + calc.multiply(num1, num2));

double divisionResult = calc.divide(num1, num2);

if (Double.isNaN(divisionResult)) {

System.out.println("Division: Error! Division by zero.");

} else {

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

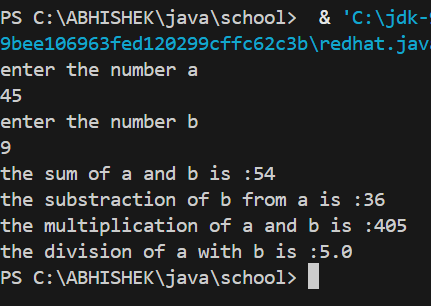
}

scanner.close();

}

}

**OUTPUT:**

****

**Class Diagram :**

**-----------------------------------**

**| Addition |**

**-----------------------------------**

**| |**

**-----------------------------------**

**| + add(a: int, b: int): int |**

**-----------------------------------**

**▲**

**|**

**-----------------------------------**

**| Subtraction |**

**-----------------------------------**

**| |**

**-----------------------------------**

**| + subtract(a: int, b: int): int|**

**-----------------------------------**

**▲**

**|**

**-----------------------------------**

**| Multiplication |**

**-----------------------------------**

**| |**

**-----------------------------------**

**| + multiply(a: int, b: int): int|**

**-----------------------------------**

**▲**

**|**

**-----------------------------------**

**| Division |**

**-----------------------------------**

**| |**

**-----------------------------------**

**| + divide(a: int, b: int): double|**

**-----------------------------------**

**▲**

**|**

**-----------------------------------**

**| Calculator |**

**-----------------------------------**

**| |**

**-----------------------------------**

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

**-----------------------------------**

**Important Points**

* The program demonstrates **multilevel inheritance** where Division inherits from Multiplication, which inherits from Subtraction, which inherits from Addition.
* Each class adds a specific arithmetic operation method: add(), subtract(), multiply(), and divide().
* The divide method handles division by zero by returning Double.NaN and provides an error message in such cases.
* The Calculator class uses an object of Division to perform all arithmetic operations.
* User inputs two numbers via Scanner, and the program displays the results of all operations.
* Proper resource management is done by closing the Scanner object at the end.

**ERROR :**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| 1 | syntax error | String forgot in main function | String is added |
| 2 | Logical error | Incorrect logic | Correct logic |

**2.Aim:** To A vehicle rental company wants to develop a system that maintains information about different types of vehicles available for rent. The company rents out cars and bikes and they need a program to store details about each variable such as brand and speed.

• Cars should have an additional property: numbers of doors, seating capacity.

• Bikes should have a property indicating whether they have gears or not.

• The system should also include a function to display details about each vehicle & indicates when a vehicle is starting.

• Every class should have constructor.

1. Which OOP concept is used in the above program? Explain why it is useful in this scenario.

2. If the company decides to add a new type of vehicle: Truck, how would you modify the program?

• Truck should include an additional property capacity(in tons)

• Create a ShowTruckDetails() method to display the truck’s capacity.

• Write a constructor for Truck that initializes all properties.

3)Aim: Implement the truck class and update the main method to create a truck object & also create an object for car & bike sub classed. Finally display its details

**Program :**

import java.util.Scanner;

class Vehicle {

String brand;

int speed;

Vehicle(String brand, int speed) {

this.brand = brand;

this.speed = speed;

}

void start() {

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

}

void showDetails() {

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

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

}

}

class Car extends Vehicle {

int numberOfDoors;

int seatingCapacity;

Car(String brand, int speed, int numberOfDoors, int seatingCapacity) {

super(brand, speed);

this.numberOfDoors = numberOfDoors;

this.seatingCapacity = seatingCapacity;

}

void showCarDetails() {

showDetails();

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

System.out.println("Seating Capacity: " + seatingCapacity);

}

}

class Bike extends Vehicle {

boolean hasGears;

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

super(brand, speed);

this.hasGears = hasGears;

}

void showBikeDetails() {

showDetails();

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

}

}

class Truck extends Vehicle {

double capacity;

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

super(brand, speed);

this.capacity = capacity;

}

void showTruckDetails() {

showDetails();

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

}

}

public class RentalSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Car car = new Car("Toyota", 180, 4, 5);

Bike bike = new Bike("Yamaha", 120, true);

Truck truck = new Truck("Volvo", 100, 15.5);

car.start();

car.showCarDetails();

System.out.println();

bike.start();

bike.showBikeDetails();

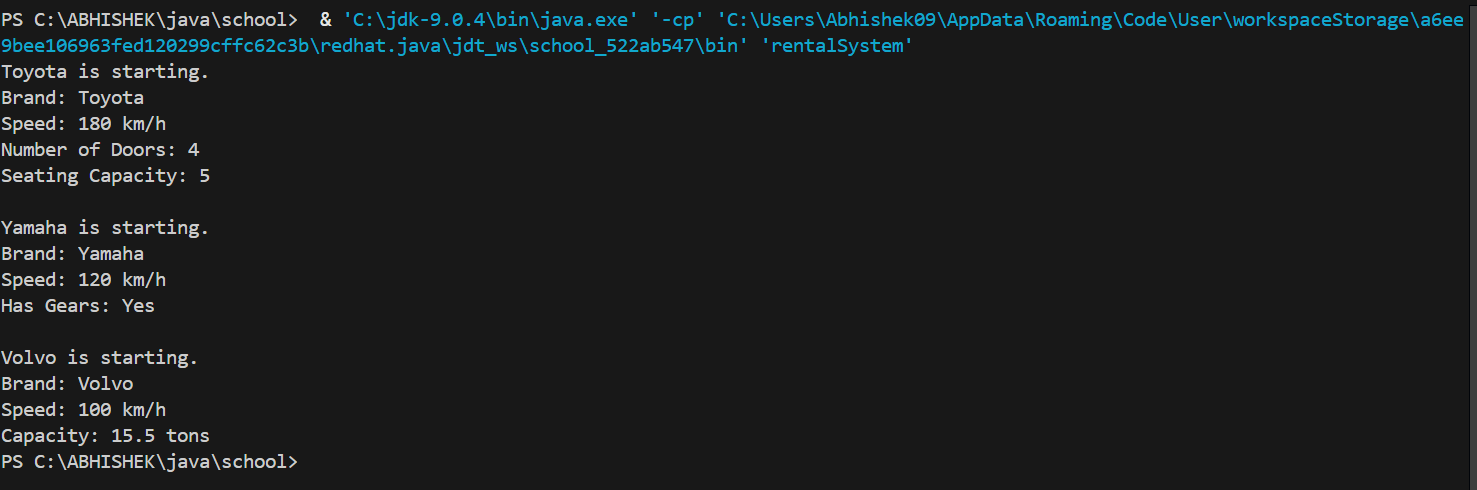
System.out.println();

truck.start();

truck.showTruckDetails();

scanner.close();

}

}**OUTPUT :** ****

**Class Diagram :**

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

| Vehicle |

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

| - brand: String |

| - speed: int |

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

| + Vehicle(brand: String, speed: int) |

| + start(): void |

| + showDetails(): void |

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

▲

|

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

| Car |

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

| - numberOfDoors: int |

| - seatingCapacity: int |

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

| + Car(brand: String, speed: int, numberOfDoors: int, seatingCapacity: int) |

| + showCarDetails(): void |

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

▲

|

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

| Bike |

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

| - hasGears: boolean |

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

| + Bike(brand: String, speed: int, hasGears: boolean) |

| + showBikeDetails(): void |

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

▲

|

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

| Truck |

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

| - capacity: double |

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

| + Truck(brand: String, speed: int, capacity: double) |

| + showTruckDetails(): void |

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

▲

|

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

| RentalSystem |

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

| |

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

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

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

**ERROR :**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| 1 | syntax error | String forgot in main function | String is added |
| 2 | Logical error | Incorrect logic | Correct logic |

**Important Points**

* The program demonstrates **inheritance** where Car, Bike, and Truck classes inherit from the base class Vehicle.
* The Vehicle class contains common attributes (brand, speed) and methods (start(), showDetails()) shared by all vehicle types.
* Each subclass (Car, Bike, Truck) adds specific attributes and methods relevant to its vehicle type, like numberOfDoors and seatingCapacity for Car, hasGears for Bike, and capacity for Truck.
* Constructors in subclasses use super() to initialize the inherited properties from Vehicle.
* The RentalSystem class contains the main method which creates objects of each subclass and calls their methods to display details and simulate starting the vehicles.
* This design promotes **code reuse** and **polymorphism**, as common functionality is centralized in the base class, reducing duplication.

**WEEK - 6**

1. **Aim : To Write a Java program to create a Vehicle class with a method displayInfo(). Override this method in the Car subclass to provide specific information about a car.**

**Program :**

class Vehicle {

void displayInfo() {

System.out.println("This is a vehicle.");

}

}

class Car extends Vehicle {

@Override

void displayInfo() {

System.out.println("This is a car. Cars are comfortable and fast.");

}

}

public class carInfo {

public static void main(String[] args) {

Vehicle genericVehicle = new Vehicle();

genericVehicle.displayInfo();

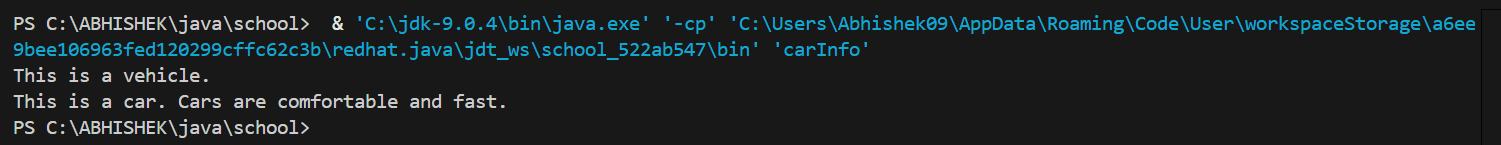
Car car = new Car();

car.displayInfo();

}

}

**OUTPUT :**

****

**Class Diagram :**

**-----------------------------------**

**| Vehicle |**

**-----------------------------------**

**| |**

**-----------------------------------**

**| + displayInfo(): void |**

**-----------------------------------**

**▲**

**|**

**-----------------------------------**

**| Car |**

**-----------------------------------**

**| |**

**-----------------------------------**

**| + displayInfo(): void |**

**-----------------------------------**

**ERRORS :**

**No errors**

**Important Points**

* The program demonstrates **method overriding**, where the Car class provides its own implementation of the displayInfo() method defined in the Vehicle class.
* The @Override annotation in the Car class ensures the method correctly overrides the parent class method.
* When displayInfo() is called on a Vehicle object, it prints a generic message. When called on a Car object, it prints a more specific message.
* This shows **runtime polymorphism**, where the method executed depends on the actual object's class at runtime.
* The main method creates instances of both Vehicle and Car and calls displayInfo() on each to demonstrate the difference.

**2) Aim :** To write a java program forA college that is developing an automated admission system that verifies students' eligibility for undergraduate (UG) and postgraduate (PG) programs. Each program has different eligibility criteria based on the students' percentage in their previous qualifications.

(i)UG admissions require a minimum of 60%.

(ii)PG admissions require a minimum of 70%

**Program :**

import java.util.Scanner;

class Admission {

void checkEligibility(double percentage) {

System.out.println("Checking general eligibility.");

}

}

class UGAdmission extends Admission {

@Override

void checkEligibility(double percentage) {

if (percentage >= 60) {

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

} else {

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

}

}

}

class PGAdmission extends Admission {

@Override

void checkEligibility(double percentage) {

if (percentage >= 70) {

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

} else {

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

}

}

}

public class admissionSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter your percentage for UG admission: ");

double ugPercentage = scanner.nextDouble();

UGAdmission ug = new UGAdmission();

ug.checkEligibility(ugPercentage);

System.out.print("Enter your percentage for PG admission: ");

double pgPercentage = scanner.nextDouble();

PGAdmission pg = new PGAdmission();

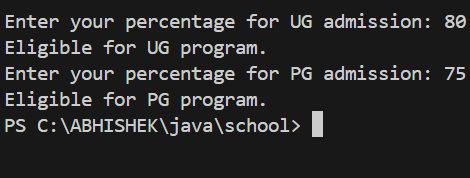
pg.checkEligibility(pgPercentage);

scanner.close();

}

}

**OUPUT :**

****

**Class Diagram :**

**-----------------------------------**

**| Admission |**

**-----------------------------------**

**| |**

**-----------------------------------**

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

**-----------------------------------**

**▲**

**|**

**-----------------------------------**

**| UGAdmission |**

**-----------------------------------**

**| |**

**-----------------------------------**

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

**-----------------------------------**

**▲**

**|**

**-----------------------------------**

**| PGAdmission |**

**-----------------------------------**

**| |**

**-----------------------------------**

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

**-----------------------------------**

**ERRORS :**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| 1 | Syntax error | ; is missed | ; is added |
| 2 | Logical error | Incorrect logic | Correct logic |

**Important Points**

* The program demonstrates **method overriding** with the checkEligibility() method in subclasses UGAdmission and PGAdmission overriding the base class Admission method.
* UGAdmission checks if the percentage is at least 60% to determine eligibility for undergraduate programs.
* PGAdmission checks if the percentage is at least 70% for postgraduate program eligibility.
* This structure allows different eligibility rules for UG and PG admissions while keeping a common interface via the Admission class.
* The main method takes user input for both UG and PG percentages, creates corresponding objects, and calls their respective checkEligibility() methods.
* Use of **runtime polymorphism** allows calling the overridden methods according to the object type.

**3) Aim :** Create a Calculator class with overloaded methods to perform addition:

(i) Add two integers.

(ii) Add two doubles.

(iii) Add three integers.

**Program :**

class Calculator1 {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

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

return a + b + c;

}

public static void main(String[] args) {

Calculator1 calc = new Calculator1();

int sumTwoInts = calc.add(5, 10);

double sumTwoDoubles = calc.add(5.5, 10.5);

int sumThreeInts = calc.add(1, 2, 3);

System.out.println("Sum of two integers: " + sumTwoInts);

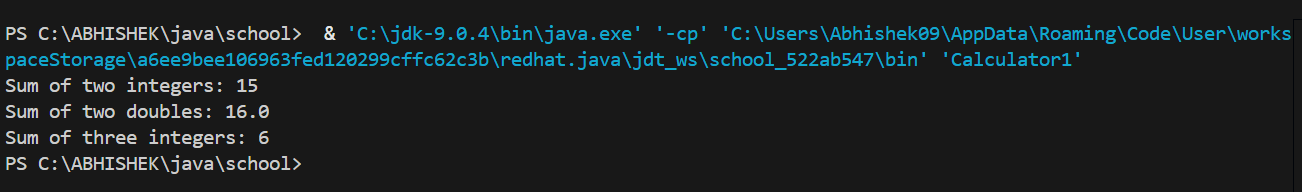
System.out.println("Sum of two doubles: " + sumTwoDoubles);

System.out.println("Sum of three integers: " + sumThreeInts);

}

}

**OUTPUT :**

****

**Class Diagram :**

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

| Calculator1 |

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

| |

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

| + add(a: int, b: int): int |

| + add(a: double, b: double): double |

| + add(a: int, b: int, c: int): int |

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

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

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

**Important Points**

* The class demonstrates **method overloading** where multiple add methods have the same name but different parameter lists (different type or number of parameters).
* Overloaded methods allow calling add with either two integers, two doubles, or three integers, with the appropriate method executed based on arguments.
* This improves code readability and usability by allowing the same method name for related operations with different inputs.
* The main method tests all overloaded versions and prints the results for each.
* Method overloading is resolved at **compile-time** (static polymorphism).

**ERROR :**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| 1 | syntax error | String forgot in main function | String is added |
| 2 | Logical error | Incorrect logic | Correct logic |

**4)Aim :**ToCreate a Shape class with a method calculateArea() that is overloaded for different shapes (e.g., square, rectangle). Then, create a subclass Circle that overrides the calculateArea() method for a circle.

**Program :**

class Shape {

double calculateArea(double side) {

return side \* side;

}

double calculateArea(double length, double breadth) {

return length \* breadth;

}

}

class Circle extends Shape {

@Override

double calculateArea(double radius) {

return Math.PI \* radius \* radius;

}

public static void main(String[] args) {

Shape shape = new Shape();

Circle circle = new Circle();

double squareArea = shape.calculateArea(5);

double rectangleArea = shape.calculateArea(4, 6);

double circleArea = circle.calculateArea(3);

System.out.println("Area of Square: " + squareArea);

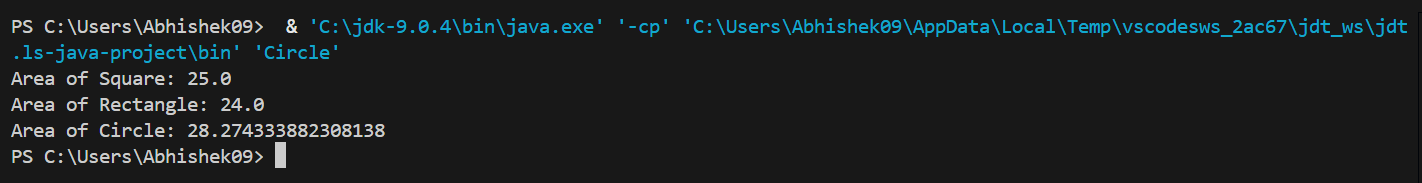
System.out.println("Area of Rectangle: " + rectangleArea);

System.out.println("Area of Circle: " + circleArea);

}

}

**OUTPUT :**

****

**Class Diagram :**

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

| Shape |

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

| |

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

| + calculateArea(side: double): double |

| + calculateArea(length: double, breadth: double): double |

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

▲

|

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

| Circle |

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

| |

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

| + calculateArea(radius: double): double |

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

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

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

**Important Points**

* The Shape class demonstrates **method overloading** with two calculateArea methods: one for square (one parameter) and one for rectangle (two parameters).
* The Circle class **extends** Shape and **overrides** the calculateArea(double) method to calculate the area of a circle using radius.
* The @Override annotation confirms that the circle’s method replaces the shape’s single-parameter area calculation.
* In main, Shape object is used to calculate areas of square and rectangle, while Circle object is used for the circle’s area.
* This example combines **method overloading** (compile-time polymorphism) in Shape and **method overriding** (runtime polymorphism) in Circle.
* Using inheritance allows Circle to inherit other methods if needed, while providing its own specific implementation for area calculation.

**ERROR :**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Error type** | **Reason for error** | **Rectification** |
| 1 | syntax error | String forgot in main function | String is added |
| 2 | Logical error | Incorrect logic | Correct logic |

WEEK-7

1. **AIM :** Write a java program to create an abstract class animal with an abstract

method called sound() create subclasses lion and tiger that extends the animal class

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

**CLASS DIAGRAM :**

|  |
| --- |
| <<abstract>>  Animal |
| + sound():void |

▲ ▲

\_ | |\_

| |

|  |  |  |
| --- | --- | --- |
| Lion |  | Tiger |
| +sound(): void |
| +sound(): void |

|  |
| --- |
| Wild |
| +main():void |

**Code :**

abstract class Animal {

abstract void sound();

}

class Lion extends Animal {

void sound() {

System.out.println("Lion roars: Roaaar");

}

}

class Tiger extends Animal {

void sound() {

System.out.println("Tiger growls: Grrrr");

}

}

public class Wild {

public static void main(String[] args) {

Animal lio = new Lion();

lio.sound();

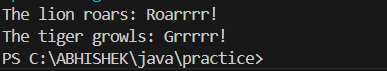
Animal tig =new Tiger();

tig.sound();

}

}

output:



**ERROR TABLE :**

|  |  |  |
| --- | --- | --- |
| s.no | Error | Error rectification |
| 1 | ; is missing in the end of print statement | We have to put semicolon at end |
| 2 | String error | ‘s’ should be capital in string |

**IMPORTATNT POINTS :**

|  |
| --- |
| **Abstraction** |

|  |  |
| --- | --- |
| Hides unnecessary details; only shows essential info. | |
| **Abstract Class** |  |

|  |  |
| --- | --- |
| Cannot be instantiated directly; used as a base. | |
| **Polymorphism** |  |

|  |  |
| --- | --- |
| Same method behaves differently based on object (Lion or Tiger). | |
| **Method Overriding** |  |

|  |
| --- |
| Subclasses define their own version of a parent class method. |
|  |

2)**AIM :** Write a java program to create abstract class Shape3D with abstract methods calculatevolume() and calculatearea() create a sub class sphere and cube that extends

The shape3D class and implement the respective method to calculate the volume and

The surfacearea of the sphere.

**CLASS DIAGRAM :**

|  |
| --- |
| <<abstract>>  Shape3D |
| +calculateVolume(): double +calculateSurfaceArea(): double |

▲ ▲

\_ | |\_

| |

|  |  |  |
| --- | --- | --- |
| Sphere |  | Cube |
| - side: double  +calculateVolume(): double  +calculateSurfaceArea(): double |
| - radius: double  +calculateVolume(): double +calculateSurfaceArea(): double |

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

**Program :**

abstract class Shape3D {

abstract double calculateVolume();

abstract double calculateSurfaceArea();

}

class Sphere extends Shape3D {

double radius;

Sphere(double radius) {

this.radius = radius;

}

double calculateVolume() {

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

}

double calculateSurfaceArea() {

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

}

}

class Cube extends Shape3D {

double side;

Cube(double side) {

this.side = side;

}

double calculateVolume() {

return Math.pow(side, 3);

}

double calculateSurfaceArea() {

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

}

}

public class ShapeTest {

public static void main(String[] args) {

Shape3D sphere = new Sphere(5);

System.out.println("Sphere Volume: " + sphere.calculateVolume());

System.out.println("Sphere Surface Area: " + sphere.calculateSurfaceArea());

Shape3D cube = new Cube(4);

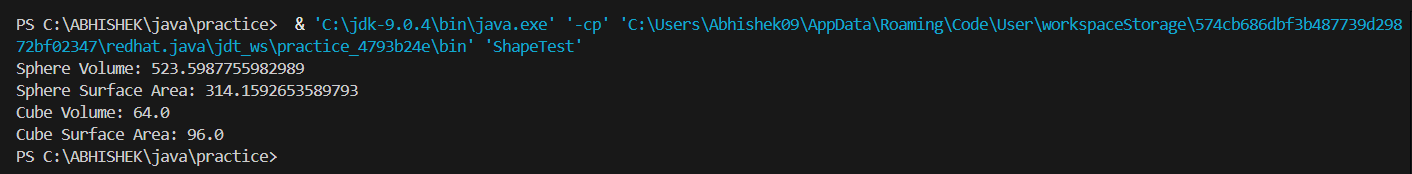
System.out.println("Cube Volume: " + cube.calculateVolume());

System.out.println("Cube Surface Area: " + cube.calculateSurfaceArea());

}

}

**Output :**



**ERROR TABLE :**

|  |  |  |
| --- | --- | --- |
| s.no | Error | Error rectification |
| 1 | Semicolon error | ‘;’ should be there in the end of every statement. |
| 2 | Extend syntax error | Extends is the correct syntax |

**IMPORTATNT POINTS:**

Encapsulation: class has its own private fields (radius, side).

Constructor: Both Sphere and Cube use constructors to initialize their dimensions.

Abstract Class (Shape3D): Defines a common structure for all 3D shapes, forcing

subclasses to implement volume and surface area methods.

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

Printing

Creata an abstract class named pattern printer with an abstract method print pattern

(int n) and a concrete method to display the pattern title.

Implement two subclasses

Star pattern-print a right angled triangle of stars(\*)

Number pattern-print a right angled triangle of increasing numbers

In the main() method create objects of both subclasses and print the pattern for

Given number of rows

CLASS DIAGRAM :

|  |
| --- |
| PatternPrinter |
| + displayTitle()  # printPattern(n) |

▲ ▲

\_ | |\_

| |

|  |  |  |
| --- | --- | --- |
| StarPattern |  | NumberPattern |
| +printPattern(n) |
| +printPattern(n) |

|  |
| --- |
| PatternDemo |
| +main() |

**Program:**

abstract class PatternPrinter {

abstract void printPattern(int n);

public void displayTitle(String title) {

System.out.println( title );

}

}

class StarPattern extends PatternPrinter {

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 PatternPrinter {

void printPattern(int n) {

int num = 1;

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

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

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

num++;

}

System.out.println();

}

}

}

public class PatternDemo {

public static void main(String[] args) {

int rows = 5;

StarPattern star = new StarPattern();

star.displayTitle("Star Pattern");

star.printPattern(rows);

System.out.println();

NumberPattern number = new NumberPattern();

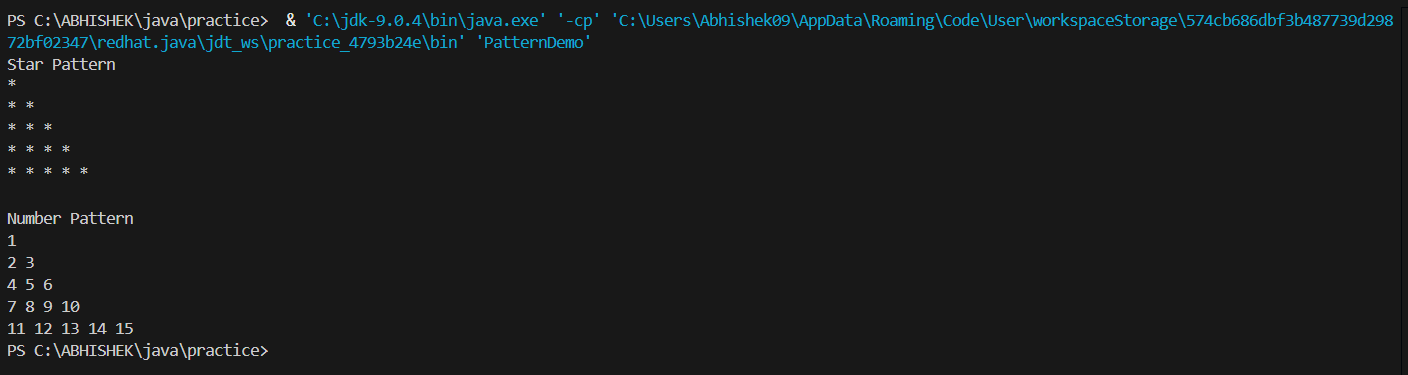
number.displayTitle("Number Pattern");

number.printPattern(rows);

}

}

Output:



**ERROR TABLE :**

|  |  |  |
| --- | --- | --- |
| s.no | Error | Error rectification |
| 1 | Prinln error | There is a syntax error println is the corrext syntax |
| 2 | for loop condition error | Codition should be inside the brackets |

**IMPORTATNT POINTS:**

Abstract Class (PatternPrinter):

Defines a blueprint for pattern printing.

Contains:

An abstract method printPattern(int n) that must be implemented by subclasses.

A concrete method displayTitle(String title) to print the title of the pattern.

WEEK-8

1. AIM : Write a java program to create an interface shape with the get perimeter()

method create three classes rectangle circle and triangle that implement the shape

interface implement the getperimeter() method for each of the three classes

CLASS DIAGRAM :

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

|  |
| --- |
| Rectangle |
| +getPerimeter(): void |

|  |
| --- |
| Circle |
| +getPerimeter(): void |

|  |
| --- |
| Triangle |
| +getPerimeter(): void |

**Code :**

interface Shape{

int getPerimeter(int a,int b,int c);

}

class Circle implements Shape{

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

double Perimeter = 2\*Math.PI\*a ;

System.out.println("The perimeter of cirlce "+Perimeter+".");

return (int) Perimeter ;

}

}

class Rectangle implements Shape{

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

double Perimeter = (a+b)/2 ;

System.out.println("The perimeter of Retangle "+Perimeter+".");

return (int) Perimeter ;

}

}

class Triangle implements Shape{

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

double Perimeter = a+b+c ;

System.out.println("The perimeter of triangle "+Perimeter+".");

return (int) Perimeter ;

}

}

class Peri {

public static void main(String args[]){

Circle c1 = new Circle();

c1.getPerimeter(3,2,2);

Rectangle r1 = new Rectangle();

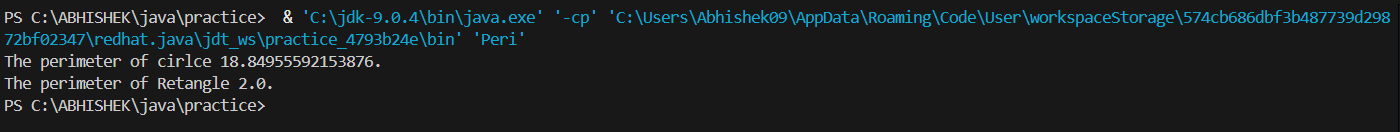
r1.getPerimeter(2,2,3);

Triangle t1 = new Triangle();

}

}

Output:



**ERROR TABLE :**

|  |  |  |
| --- | --- | --- |
| s.no | Error | Error rectification |
| 1 | System.out.println("The perimeter of the circle is: "+perimeter) | System.out.println("The perimeter of the triangle is: "+perimeter); |

**IMPORTATNT POINTS:**

Multiple Inheritance: A class can implement multiple interfaces, providing a way to achieve multiple inheritance.

Abstract Methods: All methods in an interface are implicitly abstract (in many languages) and must be implemented by the implementing class.

2)AIM : write a java program to create an interface with method play() that takes no

Arguments and returns void create three classes football, volleyball, basketball

That implement the playable interface and override the play()method to play the

Respective sports.

CLASS DIAGRAM :

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

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

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

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

|  |
| --- |
| Sports |
| +main(args:String[]): void |

**Program:**

interface Playable {

void play();

}

class Football implements Playable {

public void play() {

System.out.println("Playing football: Kicking the ball across the field.");

}

}

class Volleyball implements Playable {

public void play() {

System.out.println("Playing volleyball: Serving and spiking the ball over the net.");

}

}

class Basketball implements Playable {

public void play() {

System.out.println("Playing basketball: Dribbling and shooting the ball into the hoop.");

}

}

public class SportsTest {

public static void main(String[] args) {

Playable football = new Football();

Playable volleyball = new Volleyball();

Playable basketball = new Basketball();

football.play();

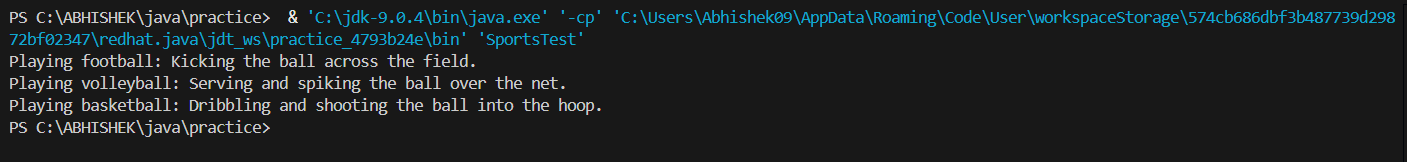
volleyball.play();

basketball.play();

}

}

**Output:**



**ERROR TABLE :**

|  |  |  |
| --- | --- | --- |
| s.no | Error | Error rectification |
| 1 | Not implementing interface method | Ensure all implementing classes define the play() method. |
| 2 | Case sensitivity errors | Check for exact casing in syntax. |

**IMPORTATNT POINTS:**

Interface Usage (Playable):

Defines a method play() that must be implemented by all classes that implement it.

Implementation by Multiple Classes:

Football, Volleyball, and Basketball all implement Playable and define their own version of the play()

method.

This shows polymorphism and code reuse.

WEEK-9

1. AIM : write a java program to create a method that takes integer as a parameter

And throws an exception if the number is even.

CLASS DIAGRAM :

|  |
| --- |
| EvenNumberExceptionDemo |
|  |
| + static void checkOdd(int): void  + static void main(String[]): void |

▲

|

|  |
| --- |
| EvenNumberException |
| + EvenNumberException(message: String) |
| extends: Exception |

**Program:**

public class EvenNumberExceptionDemo {

static class EvenNumberException extends Exception {

public EvenNumberException(String message) {

super(message);

}

}

public static void checkOdd(int number) throws EvenNumberException {

if (number % 2 == 0) {

throw new EvenNumberException("The number " + number + " is even!");

} else {

System.out.println("The number " + number + " is odd. Accepted.");

}

}

public static void main(String[] args) {

int testNumber = 3;

try {

checkOdd(testNumber);

} catch (EvenNumberException e) {

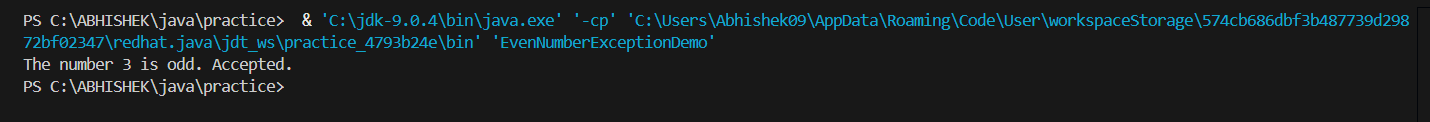
System.out.println("Exception caught: " + e.getMessage());

}

}

}

Output:



**ERROR TABLE :**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| Semicolon did not mentioned | Kept ; after the statement |

**IMPORTATNT POINTS:**

EvenNumberException is a user-defined exception that extends the built-in Exception class.

Accepts a String message and passes it to the superclass Exception using super(message).

checkOdd(int number) checks if the number is odd.

If the number is even, it throws an EvenNumberException.

2)AIM : write a java program to create a method that reads a file and throws an

Exception if the file is not found.

CLASS DIAGRAM :

|  |
| --- |
| FileReaderDemo |
|  |
| +readFile(filename: String): void throws FileNotFoundException  + main(args: String[]): void |

**Program:**

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class FileReaderDemo {

public static void readFile(String filename) throws FileNotFoundException {

File file = new File(filename);

Scanner scanner = new Scanner(file);

System.out.println("File content:");

while (scanner.hasNextLine()) {

System.out.println(scanner.nextLine());

}

scanner.close();

}

public static void main(String[] args) {

try {

readFile("sample.txt");

} catch (FileNotFoundException e) {

System.out.println("Error: File not found!");

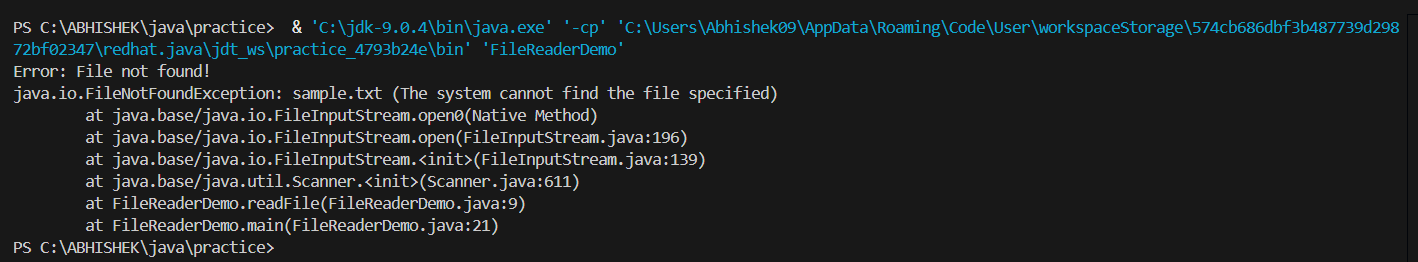
e.printStackTrace();

}

}

}

Output:



**ERROR TABLE :**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| catch(){  ^ 1 error | catch(FileNotFoundException e){ |

**IMPORTATNT POINTS:**

The program demonstrates reading data from a file using java.io.File and java.util.Scanner.

The method readFile includes throws FileNotFoundException because Scanner can throw this checked exception if the file is missing.

3)AIM : write a java program to handle arithematic exceptions using try, catch and

finally

**CLASS DIAGRAM :**

|  |
| --- |
| DivisionChecker |
| +checkDenominator(a: int, b: int): void throws ArithmeticException +main(args: String[]): void |

**Program:**

public class ArithmeticException {

public static void main(String[] args) {

int a = 10;

int b = 0;

try {

int result = a / b;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Exception caught: Division by zero is not allowed.");

} finally {

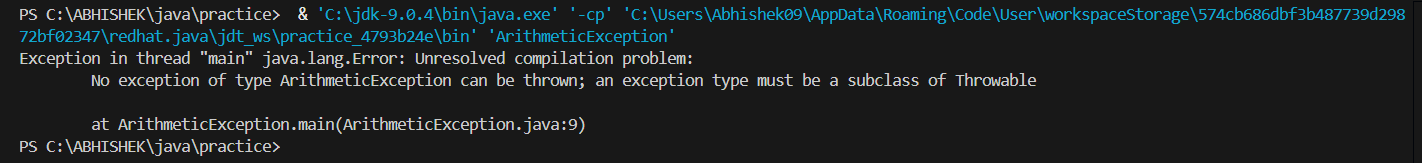
System.out.println("This block always executes (finally block).");

}

}

}

Output:



**ERROR TABLE :**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| try  ^ 1 error | try { } |

**IMPORTATNT POINTS :**

try block: Contains code that might cause an exception (a / b).

catch block: Catches and handles the ArithmeticException if b is 0.

finally block: Executes regardless of whether an exception occurs or not.

4)AIM : write a java program to stimulate a university system using inner classes.

Create an outer class named university with a variable UniversityName.

Inside it ,define two non static inner classes:

Department – with variables like deptName & deptCode , and a method to display

department details

Student – with variables like StudentName & rollNumber & a method to display student

Details

Create an object

CLASS DIAGRAM :

|  |
| --- |
| University |
| - universityName: String |
| + University(name: String) |
| <<inner>> Department  <<inner>> Student |

▲ ▲

\_ | |\_ \_

| |

|  |  |  |
| --- | --- | --- |
| University.Department |  | University.Student |
| - studentName: String  - rollNumber: int |
| - deptName: String  - deptCode: String |
| + Department(name, code)  + displayDepartmentDetails() |  | + Student(name, roll)  + displayStudentDetails() |

**Program:**

public class University {

private String universityName;

public University(String name) {

this.universityName = name;

}

class Department {

private String deptName;

private String deptCode;

public Department(String name, String code) {

this.deptName = name;

this.deptCode = code;

}

public void displayDepartmentDetails() {

System.out.println("University: " + universityName);

System.out.println("Department Name: " + deptName);

System.out.println("Department Code: " + deptCode);

}

}

class Student {

private String studentName;

private int rollNumber;

public Student(String name, int roll) {

this.studentName = name;

this.rollNumber = roll;

}

public void displayStudentDetails() {

System.out.println("University: " + universityName);

System.out.println("Student Name: " + studentName);

System.out.println("Roll Number: " + rollNumber);

}

}

public static void main(String[] args) {

University uni = new University("Amrita University");

University.Department dept = uni.new Department("Computer Science", "CSE101");

University.Student student = uni.new Student("Lalith", 24213);

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

dept.displayDepartmentDetails();

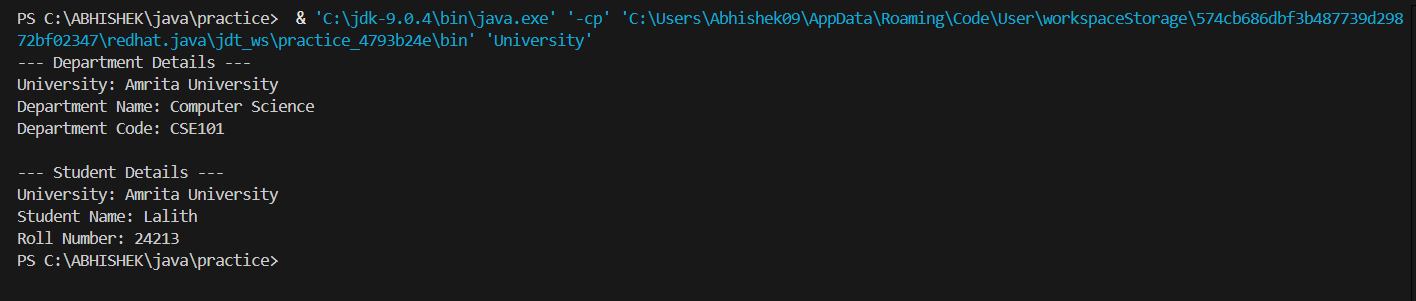
System.out.println("\n--- Student Details ---");

student.displayStudentDetails();

}

}

Output:



**ERROR TABLE :**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| Students.displayStudentInfo(); | students.displayStudentInfo(); |

**IMPORTATNT POINTS :**

The program defines Department and Student as inner classes inside the University class, showing a strong association between the university and its departments/students.

Inner classes can directly access private members (universityName) of the outer class (University), demonstrating encapsulation and nested class behavior.

WEEK-10

1)AIM : Write a java program to generate a password for a student using his/her initials

And age. the password displayed should be the string consists of first character of first

name, middle name, last name with age.

CLASS DIAGRAM :

|  |
| --- |
| StudentPasswordGenerator |
|  |
| + main(String[] args): void |

**Program:**

import java.util.Scanner;

public class StudentPasswordGenerator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter First Name: ");

String firstName = scanner.nextLine();

System.out.print("Enter Middle Name: ");

String middleName = scanner.nextLine();

System.out.print("Enter Last Name: ");

String lastName = scanner.nextLine();

System.out.print("Enter Age: ");

int age = scanner.nextInt();

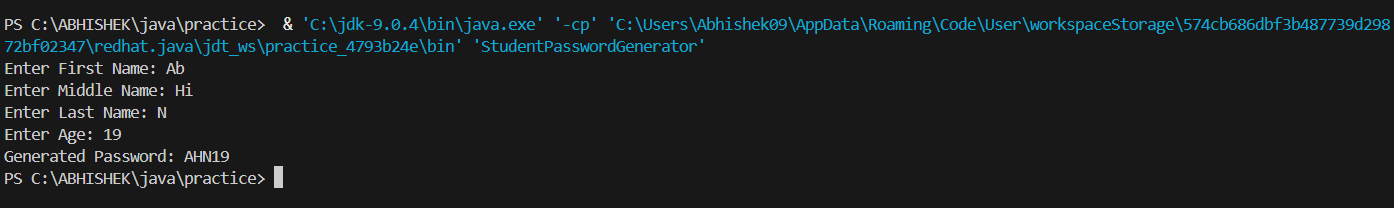
String password = "" + firstName.charAt(0) + middleName.charAt(0) + lastName.charAt(0) + age;

System.out.println("Generated Password: " + password);

}

}

Output:



**ERROR TABLE :**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| scanner = new Scanner (System.in): | Scanner scanner = new Scanner (System.in); |

**IMPORTATNT POINTS :**

String Character Access Using charAt(0):

The program extracts the initials (first letters) of the first, middle, and last names using charAt(0). This

method is essential for building the password based on user input.

Password Construction Using String Concatenation:

The password is constructed by concatenating characters and an integer (age). In Java, when combining characters and integers in a string, the + operator automatically converts them into a single string.

2)AIM : Design and implement a Java program that will do the following operations to this string "Welcome! You are practicing strings concept."

a. convert all alphabets to capital letters and print out the result.

b. convert all alphabets to lower-case letters and print out the result.

c. print out the length of the string.

d. print out the index of Course.

CLASS DIAGRAM :

|  |
| --- |
| StringOperations |
|  |
| + main(String[] args): void |

**Program:**

public class StringOperations {

public static void main(String[] args) {

String input = "Welcome! You are practicing strings concept.";

String upperCase = input.toUpperCase();

System.out.println("Uppercase: " + upperCase);

String lowerCase = input.toLowerCase();

System.out.println("Lowercase: " + lowerCase);

int length = input.length();

System.out.println("Length of the string: " + length);

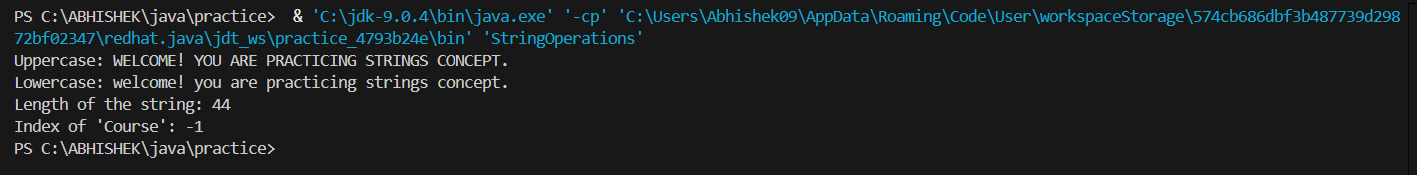
int index = input.indexOf("Course");

System.out.println("Index of 'Course': " + index);

}

}

Output:



|  |  |
| --- | --- |
| Error Found | Error Rectified |
| scanner = new Scanner (System.in): | Scanner scanner = new Scanner (System.in); |

**ERROR TABLE :**

**IMPORTATNT POINTS :**

toUpperCase() converts the entire string to uppercase.

toLowerCase() converts the entire string to lowercase.

length() gives the total number of characters in the string.

indexOf("Course") searches for the position of a substring and returns -1 if not found.

3)AIM : Implement a java program using the below array methods.

a. Sorting the elements (numbers and Strings) of an array.

b. convert the array elements into string.

c. fill the part of an array.

d. copy the elements of one array into another.

CLASS DIAGRAM :

|  |
| --- |
| ArrayMethods |
|  |
| + main(String[] args): void |

**Program:**

import java.util.Arrays;

public class ArrayMethods {

public static void main(String[] args) {

int[] numbers = {5, 2, 9, 1, 7};

String[] words = {"Banana", "Apple", "Mango", "Grapes"};

Arrays.sort(numbers);

Arrays.sort(words);

System.out.println("Sorted Numbers: " + Arrays.toString(numbers));

System.out.println("Sorted Strings: " + Arrays.toString(words));

String numbersAsString = Arrays.toString(numbers);

System.out.println("Numbers as String: " + numbersAsString);

int[] marks = new int[5];

Arrays.fill(marks, 1, 4, 100);

System.out.println("Partially filled array: " + Arrays.toString(marks));

int[] original = {10, 20, 30, 40};

int[] copy = Arrays.copyOf(original, original.length); // full copy

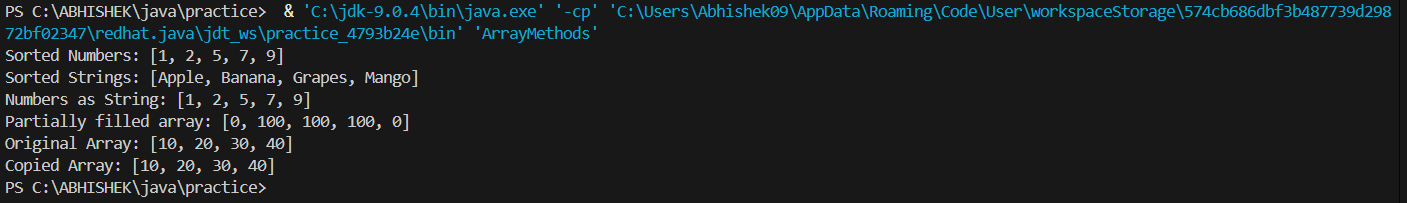
System.out.println("Original Array: " + Arrays.toString(original));

System.out.println("Copied Array: " + Arrays.toString(copy));

}

}

Output:



**ERROR TABLE :**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| .sort(words);  ^ 1 error | Arrays.sort(words); |

**IMPORTATNT POINTS:**

Arrays.sort(numbers) sorts the integer array in ascending order.

Arrays.sort(words) sorts the string array alphabetically.

Arrays.toString(array) converts an array to a printable string format.

4)AIM : Implement a java program using the below Array List methods.

a. insert an element at particular index in the array list.

b. Modify an element in the array list.

c. Access an element from the array list.

d. Remove an element from the array list.

e. clear the elements from the array list.

CLASS DIAGRAM :

|  |
| --- |
| ArrayListMethods |
| - fruits: ArrayList<String>  - element: String |
| + main(args: String[]): void |

**Program:**

import java.util.ArrayList;

public class ArrayListMethods {

public static void main(String[] args) {

ArrayList<String> fruits = new ArrayList<>();

fruits.add("Apple");

fruits.add("Banana");

fruits.add("Mango");

System.out.println("Initial ArrayList: " + fruits);

fruits.add(1, "Grapes");

System.out.println("After inserting Grapes at index 1: " + fruits);

fruits.set(2, "Orange");

System.out.println("After modifying element at index 2 to Orange: " + fruits);

String element = fruits.get(3);

System.out.println("Element at index 3: " + element);

fruits.remove(0);

System.out.println("After removing element at index 0: " + fruits);

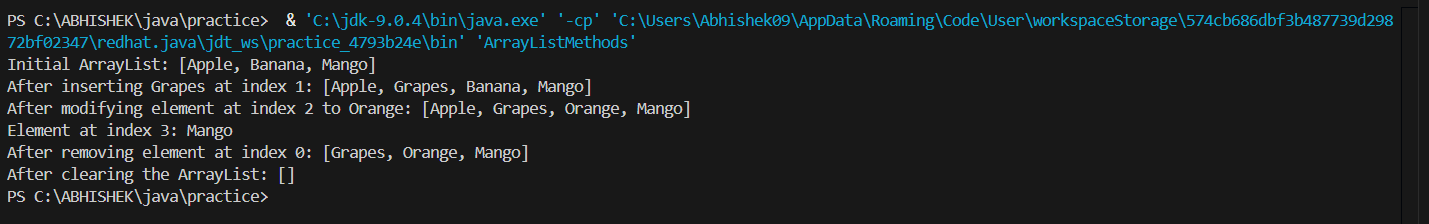
fruits.clear();

System.out.println("After clearing the ArrayList: " + fruits);

}

}

Output:



**ERROR TABLE :**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| ArrayList<String> fruits = new ArrayList | ArrayList<String> fruits = new ArrayList<String>(); |

**IMPORTATNT POINTS :**

ArrayList<String> fruits = new ArrayList<>();

Initializes an empty ArrayList to store strings (fruit names).

fruits.add("Apple");, fruits.add("Banana");, fruits.add("Mango");

Adds elements to the end of the list.

fruits.add(1, "Grapes");