

Object Oriented Programming (23CSE111)  
Assignment

Submitted by	
Name	
Roll No	
Year/Sem/Section	
Date of Submission	
Submitted to	
Name	Dr. B Raj Kumar
Department	CSE
Designation	Asst. Professor
Marks	

1. Write a java program with class named “book”. The class should contain various attributes such as “title, author, yearofpublication”. It should also contain a “constructor” with parameters which initializes “title”, ”author”, and “yearofpublication”.Create a method which displays the details of the book i.e. “author, title, yearofpublication”.(Display the details of two books i.e. create 2 objects and display their details).

**CODE:**

```
class Book {
    // Attributes
    String title;

    String author;
    int YearOfPublication;
    // Constructor to initialize the attributes
    public Book(String title, String author, int YearOfPublication) {
        this.title = title;
        this.author = author;
        this.YearOfPublication = YearOfPublication; // Fixed typo here
    }
    // Method to display book details
    public void display() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Year of Publication: " + YearOfPublication);
    }
}

// Main class
public class Books {
    public static void main(String[] args) {
        // Creating two Book objects with different details
        Book b1 = new Book("Java", "Isha", 2020);
        Book b2 = new Book("Python", "Isha", 2021);
        // Displaying the details of both books
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
        System.out.println("Book 1 Details:");
        b1.display();
        System.out.println("Book 2 Details:");
```

```

        b2.display();
    }
}

```

## OUTPUT:

```

[Running] cd "c:\Users\91944\OneDrive\Desktop\MADHURI\java\" && javac Books.java && java Books
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Book 1 Details:
Title: Java
Author: Isha
Year of Publication: 2020
Book 2 Details:
Title: Python
Author: Isha
Year of Publication: 2021

```

### Errors

SI.NO	ERROR	ERROR RECTIFICATION
1	Missing semicolons	Insterted semicolons
2	<i>System.out.orintln</i>	<i>System.out.println</i>

### Important points

1. public String title\_of\_book; - Used to declare a variable named title\_of\_book, with data type as String with public accessibility.
2. Write a java program with class named “MyClass”, with a static variable “count” of “int” type, initialized to “0” and a constant variable “PI” of type “double” initialized to 3.14159 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.

### CODE :

```

class MyClass {
    static int count=0;
    static final double PI=3.14159;
    public MyClass(){
        count++;
    }
    public static void main(String[] args){

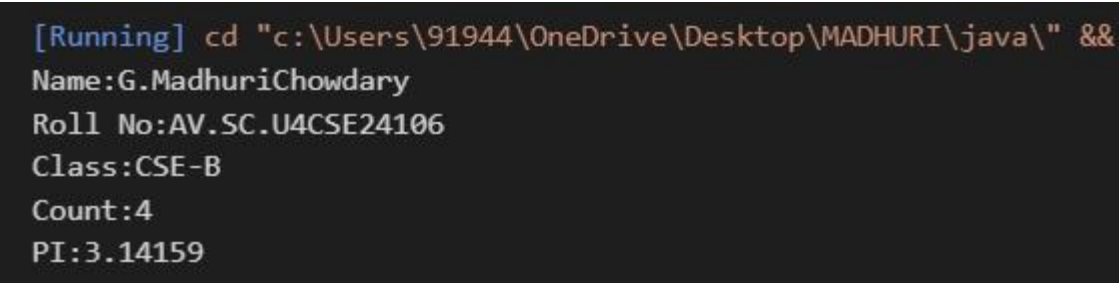
```

```

MyClass a1=new MyClass();
MyClass a2=new MyClass();
MyClass a3=new MyClass();
MyClass a4=new MyClass();
    System.out.println("Name:G.MadhuriChowdary");
    System.out.println("Roll No:AV.SC.U4CSE24106");
    System.out.println("Class:CSE-B");
System.out.println("Count:"+MyClass.count);
System.out.println("PI:"+MyClass.PI);
}
}

```

#### OUTPUT:



```

[Running] cd "c:\Users\91944\OneDrive\Desktop\MADHURI\java\" &&
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Count:4
PI:3.14159

```

#### ERROR:

- NO ERROR.

### 3. Define a Java class named VisibilityExample with the following attributes and methods:

#### Attributes:

- A public integer variable named publicVariable, initialized to 10.
- A private integer variable named privateVariable, initialized to 20.

#### Methods:

- A public method named publicMethod() that prints "This is a public method."
- A private method named privateMethod() that prints "This is a private method."
- In a separate Java class named Main, write the main method to demonstrate accessing the members of the VisibilityExample class:
- Create an object of the VisibilityExample class.
- Access and print the value of the public variable publicVariable.
- Call the public method publicMethod().
- Attempt to access the private variable privateVariable and call the private method privateMethod() in the Main class.
- Note: attempting to do so will result in a compilation error.

#### CODE :

```

class VisibilityExample {

```

```

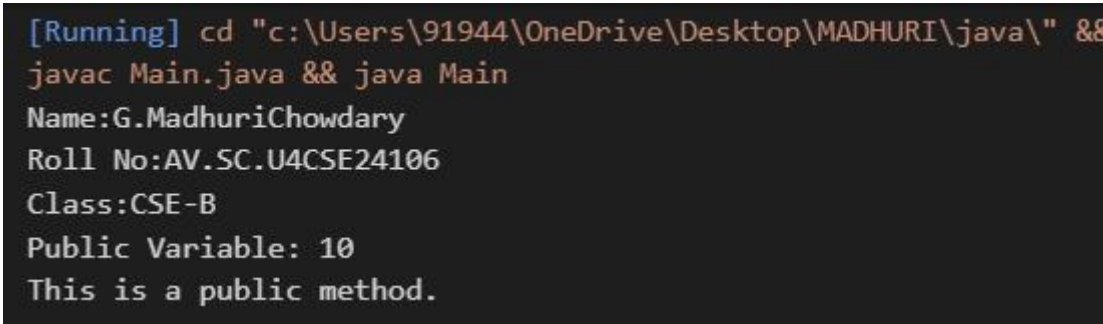
// Public variable
// Private variable
public int publicVariable = 10;
private int privateVariable = 20;

// Public method
public void publicMethod() {
    System.out.println("This is a public method.");
}
// Private method
private void privateMethod() {
    System.out.println("This is a private method.");
}
}

public class Main {
    public static void main(String[] args) {
        // Create object of VisibilityExample
        VisibilityExample obj = new VisibilityExample();
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
        // To get Access public variable
        System.out.println("Public Variable: " + obj.publicVariable);
        // Call public method
        obj.publicMethod();
    }
}

```

#### OUTPUT:



```

[Running] cd "c:\Users\91944\OneDrive\Desktop\MADHURI\java\" &&
javac Main.java && java Main
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Public Variable: 10
This is a public method.

```

#### ERRORS:

- Public members (publicVariable, publicMethod()) are accessible from other classes.
- Private members (privateVariable, privateMethod()) are not accessible outside their class, and trying to access them from Main will give compile-time errors.

**NOTE:**

Here in the above code only public variable and public method are taken so that only the code got executed if private variables, methods were taken it leads to compilation error

- 4. Write a Java program that takes a number from the user and generates an integer between 1 and 7. It displays the weekday name (Use Conditional Statements).**

***Ex: Sample Input***

**Input number: 3**

***Expected Output :***

**Wednesday**

**CODE:**

```
import java.util.Scanner;

public class Weekday {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // input from the user
        System.out.print("Input number (1 to 7): ");
        int number = scanner.nextInt();
        // conditional statements
        if (number == 1) {
            System.out.println("Monday");
        } else if (number == 2) {
            System.out.println("Tuesday");
        } else if (number == 3) {
            System.out.println("Wednesday");
        } else if (number == 4) {
            System.out.println("Thursday");
        } else if (number == 5) {
            System.out.println("Friday");
        } else if (number == 6) {
            System.out.println("Saturday");
        } else if (number == 7) {
            System.out.println("Sunday");
        }
    }
}
```

```

    } else {
        System.out.println("Invalid input! Please enter a number between 1 and 7.");
    }
    scanner.close();
}
}

```

### OUTPUT :

```

PS C:\Users\91944> & 'C:\Program Files\Java\jdk-21\bin\java.exe'
91944\AppData\Local\Temp\vscodesws_2694d\jdt_ws\jdt.ls-java-project
Input number (1 to 7): 4
Thursday

```

### Error:

Sl.NO	ERROR	ERROR RECTIFICATION
1.	Scanner object not closed	Fix: Always close the scanner after use.
2.	Cannot find symbol: Scanner	import java.util.Scanner;" at the beginning of the code.

### Important points

1. Always close the scanner to prevent resource leaks.
2. Use break statements in switch cases to avoid fall-through errors.

## 5. Write a Java program to display the multiplication table of a given integer.

*Ex: Sample Input*

**Input the number (Table to be calculated) : Input number of terms : 5**

*Expected Output :*

```

5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25

```

### **CODE:**

```

import java.util.Scanner;

public class Tables {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
    }
}

```

```

System.out.println("Class:CSE-B");
System.out.print("Input the number (Table to be calculated): ");
int number = scanner.nextInt();
System.out.print("Input number of terms: ");
int terms = scanner.nextInt();
for (int i = 0; i <= terms; i++) {
    System.out.println(number + " X " + i + " = " + (number * i));
}
scanner.close();
}
}

```

### OUTPUT:

```

PS C:\Users\91944> & 'C:\Program Files\Java\jdk-21\bin\
91944\AppData\Local\Temp\vscodesws_2694d\jdt_ws\jdt.ls-j
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Input the number (Table to be calculated): 5
Input number of terms: 4
5 X 0 = 0
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
PS C:\Users\91944>

```

### Errors

Sl.NO	ERROR	ERROR RECTIFICATION
<u>1.</u>	int number=.nextInt();	intnumber=Scanner.nextInt();
<u>2</u>	<u>System.out.println("number + " x " + l + " = "</u> <u>+ (number*i));</u>	<u>System.out.println(number + "</u> <u>* " + i + " = " +</u> <u>(number*i));</u>

### Important points

- 1.This program generates a multiplication table for a given number.
2. The user provides the number and the number of terms for the table.

**6.Write a Java program that reads two floating-point numbers and tests whether they are the same up to three decimal places (Use Conditional Statements).**

#### *Ex: Sample Input*

**Input floating-point number: 25.586**

**Input floating-point another number: 25.589**

#### *Expected Output :*

**They are different**



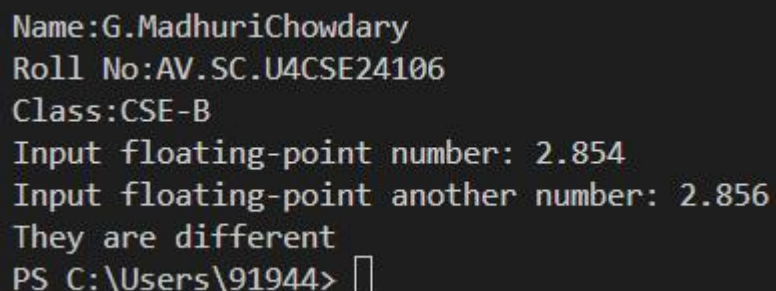
**CODE:**

```
import java.util.Scanner;
public class Decimals {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");

        System.out.print("Input floating-point number: ");
        double num1 = scanner.nextDouble();
        System.out.print("Input floating-point another number: ");
        double num2 = scanner.nextDouble();

        int intNum1 = (int)(num1 * 1000);
        int intNum2 = (int)(num2 * 1000);

        if (intNum1 == intNum2) {
            System.out.println("They are the same up to three decimal places.");
        } else {
            System.out.println("They are different");
        }
        scanner.close();
    }
}
```

**OUTPUT :**A screenshot of a terminal window showing the output of the Java program. The text is as follows:  
Name:G.MadhuriChowdary  
Roll No:AV.SC.U4CSE24106  
Class:CSE-B  
Input floating-point number: 2.854  
Input floating-point another number: 2.856  
They are different  
PS C:\Users\91944> **ERRORS :**

```
91944\AppData\Local\Temp\vscodesws_20940\jdk_ws\jdk-15-java-project\bin\Decimals
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    Scanner cannot be resolved to a type
    Scanner cannot be resolved to a type

    at Decimals.main(Decimals.java:4)
```

7. Write a program that accepts three numbers from the user and prints "increasing" if the numbers are in increasing order, "decreasing" if the numbers are in decreasing order, and "Neither increasing or decreasing order" otherwise (Use Conditional Statements).

*Ex: Sample Output*

**Input first number: 1524**

**Input second number: 2345**

**Input third number: 3321**

*Expected Output :*

**CODE :**

```
import java.util.Scanner;

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

        Scanner scanner = new Scanner(System.in);

        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");

        System.out.print("Input first number: ");
        int num1 = scanner.nextInt();

        System.out.print("Input second number: ");
        int num2 = scanner.nextInt();

        System.out.print("Input third number: ");
        int num3 = scanner.nextInt();

        // Check order

        if (num1 < num2 && num2 < num3) {
            System.out.println("Increasing");
        } else if (num1 > num2 && num2 > num3) {
            System.out.println("Decreasing");
        } else {
            System.out.println("Neither increasing or decreasing order");
        }
    }
}
```

```

        scanner.close();
    }
}

```

#### OUTPUT :

```

91944\AppData\Local\Temp\vscodesws_2694d\jdt_ws\jdt.ls-java-project\bin' 'NumberOrderChecker'
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Input first number: 1524
Input second number: 2345
Input third number: 3321
Increasing
PS C:\Users\91944>

```

#### ERRORS :

```

91944\AppData\Local\Temp\vscodesws_2694d\jdt_ws\jdt.ls-java-project\bin' 'NumberOrderChecker'
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    Syntax error, insert ";" to complete Statement

    at NumberOrderChecker.main(NumberOrderChecker.java:18)

```

8. Write a Java program that reads a positive integer and count the number of digits the number (less than ten billion) has (Use Conditional Statements).

**Ex: Sample Output**

**Input an integer number less than ten billion: 125463**

**Expected Output :**

**Number of digits in the number: 6**

#### CODE :

```

import java.util.Scanner;

public class DigitCounter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
        System.out.print("Input an integer number less than ten billion: ");

        long number = scanner.nextLong();
        if (number < 0) {
            System.out.println("The number is not positive.");
        } else if (number >= 10000000000L) {
            System.out.println("The number is too large.");
        }
    }
}

```

```

    } else {
        int digits = 0;
        long temp = number;

        while (temp > 0) {
            temp /= 10;
            digits++;
        }
        System.out.println("Number of digits in the number: " + digits);
    }
    scanner.close();
}
}

```

#### OUTPUT :

```

PS C:\Users\91944> & 'C:\Program Files\Java\jdk-21\bin\jav
91944\AppData\Local\Temp\vscodesws_2694d\jdt_ws\jdt.ls-java
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Input an integer number less than ten billion: 100000
Number of digits in the number: 6
PS C:\Users\91944> 

```

#### ERRORS :

- If the number is negative, shows a warning.
- If it's  $\geq 10,000,000,000$ , says it's too large.

#### IMPORTANT POINTS :

- We use `long` to handle numbers up to just below 10,000,000,000 (ten billion).
- The `while (temp > 0)` loop divides the number by 10 each time to count digits.
- `digits++` counts each division step = total number of digits.

#### 9. Write a Java program to display Pascal's triangle.

**Ex: Sample Output**

**Input number of rows: 5**

**Expected Output :**

**Input number of rows: 5**

```

    1
  1 1

```

```

    1 2 1
  1 3 3 1
1 4 6 4 1

```

### CODE:

```

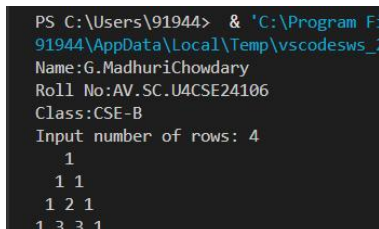
import java.util.Scanner;

public class PascalsTriangle {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
        System.out.print("Input number of rows: ");

        int rows = scanner.nextInt();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < rows - i - 1; j++) {
                System.out.print(" ");
            }
            int number = 1;
            for (int j = 0; j <= i; j++) {
                System.out.print(number + " ");
                number = number * (i - j) / (j + 1);
            }
            System.out.println();
        }
        scanner.close();
    }
}

```

### OUTPUT:



```

PS C:\Users\91944> & 'C:\Program Files\91944\AppData\Local\Temp\vscodesws_2
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Input number of rows: 4
  1
 1 1
1 2 1
1 3 3 1

```

**ERROR: NO ERROR.**

**10. Write a Java program to display the following character rhombus structure.**

**Ex: Sample Output**

**Input the number: 7**

*Expected Output :*

A  
ABA  
ABCBA  
ABCD CBA  
ABCDEDCBA  
ABCDEFEDCBA  
ABCDEFGFEDCBA  
ABCDEFEDCBA  
ABCDEDCBA  
ABCD CBA  
ABCBA  
ABA  
A

**CODE :**

```
import java.util.Scanner;

public class Rhombus {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Input the number: ");
        int n = scanner.nextInt();
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
        for (int i = 1; i <= n; i++) {
            for (int j = i; j < n; j++) {
                System.out.print(" ");
            }
            for (int j = 0; j < i; j++) {
                System.out.print((char)('A' + j));
            }
            for (int j = i - 2; j >= 0; j--) {
                System.out.print((char)('A' + j));
            }
            System.out.println();
        }
    }
}
```

```

    for (int i = n - 1; i >= 1; i--) {
        for (int j = n; j > i; j--) {
            System.out.print(" ");
        }
        for (int j = 0; j < i; j++) {
            System.out.print((char)('A' + j));
        }
        for (int j = i - 2; j >= 0; j--) {
            System.out.print((char)('A' + j));
        }
        System.out.println();
    }
    scanner.close();
}
}

```

### OUTPUT :

```

Input the number: 4
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
  A
 ABA
ABCBA
ABCD CBA
 ABCBA
  ABA
   A

```

### ERRORS

Sl.NO	ERRORS	ERRORS RECTIFICATION
<u>1.</u>	The leading spaces may not align correctly.	for (int j = 0; j < n - i - 1; j++)
<u>2</u>	The characters might not print symmetrically.	Ensure the loops print in correct order:

### Important points

1. Start the lower half loop from n - 2 to prevent duplication.
2. The first half prints characters from 'A' to the current row index

**11. Write a Java program to create a vehicle class hierarchy. The base class should be Vehicle, with subclasses Truck, Car and Motorcycle. Each subclass should have**

**properties such as make, model, year, and fuel type. Implement methods for calculating fuel efficiency, distance travelled, and maximum speed.**

**CODE:**

```
abstract class Vehicle {
    protected String make;
    protected String model;
    protected int year;
    protected String fuelType;
    public Vehicle(String make, String model, int year, String fuelType) {
        this.make = make;
        this.model = model;
        this.year = year;
        this.fuelType = fuelType;
    }
    public abstract double calculateFuelEfficiency();
    public double calculateDistanceTravelled(double fuelUsed) {
        return fuelUsed * calculateFuelEfficiency();
    }
    public abstract double getMaxSpeed();
    public void displayInfo() {
        System.out.println(year + " " + make + " " + model + " (" + fuelType + ")");
    }
}

class Truck extends Vehicle {
    public Truck(String make, String model, int year, String fuelType) {
        super(make, model, year, fuelType);
    }
    @Override
    public double calculateFuelEfficiency() {
        return 5.0;
    }
    @Override
    public double getMaxSpeed() {
        return 120.0;
    }
}

class Car extends Vehicle {
    public Car(String make, String model, int year, String fuelType) {
```



```

        super(make, model, year, fuelType);
    }

    @Override
    public double calculateFuelEfficiency() {
        return 15.0;
    }

    @Override
    public double getMaxSpeed() {
        return 180.0;
    }
}

class Motorcycle extends Vehicle {
    public Motorcycle(String make, String model, int year, String fuelType) {
        super(make, model, year, fuelType);
    }

    @Override
    public double calculateFuelEfficiency() {
        return 35.0;
    }

    @Override
    public double getMaxSpeed() {
        return 160.0;
    }
}

public class VehicleDemo {
    public static void main(String[] args) {
        Vehicle truck = new Truck("Volvo", "FH", 2020, "Diesel");
        Vehicle car = new Car("Toyota", "Camry", 2022, "Petrol");
        Vehicle motorcycle = new Motorcycle("Yamaha", "MT-15", 2023, "Petrol");
        Vehicle[] vehicles = { truck, car, motorcycle };
        for (Vehicle v : vehicles) {
            v.displayInfo();
            System.out.println("Fuel Efficiency: " + v.calculateFuelEfficiency() + " km/l");
            System.out.println("Distance with 10L: " + v.calculateDistanceTravelled(10) + " km");
            System.out.println("Max Speed: " + v.getMaxSpeed() + " km/h\n");
        }
    }
}

```

```
}
```

## OUTPUT :

```
91944\AppData\Local\Temp\vscode\sws_26
2020 Volvo FH (Diesel)
Fuel Efficiency: 5.0 km/l
Distance with 10L: 50.0 km
Max Speed: 120.0 km/h

2022 Toyota Camry (Petrol)
Fuel Efficiency: 15.0 km/l
Distance with 10L: 150.0 km
Max Speed: 180.0 km/h

2023 Yamaha MT-15 (Petrol)
Fuel Efficiency: 35.0 km/l
Distance with 10L: 350.0 km
Max Speed: 160.0 km/h
```

## ERRORS

SI.NO	ERROR	ERROR RECTIFICATION
1	Case sensitive	Fixed method name consistency
2	Missing semicolon (;)	all statements e end withsemicolon.

### Important points

- 1.Vehicle is an abstract class that defines a common interface for all vehicle types.
- 2.Contains abstract methods:

**12. Write a Java program to create a class called Employee with methods called work () and getSalary(). Create a subclass called HRManager that overrides the work () method and adds a new method called addEmployee().**

## CODE :

```
class Employee {
    protected String name;
    protected double salary;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
}
```

```

    public void work() {
        System.out.println(name + " is working.");
    }

    public double getSalary() {
        return salary;
    }
}

class HRManager extends Employee {
    public HRManager(String name, double salary) {
        super(name, salary);
    }
    @Override
    public void work() {
        System.out.println(name + " is managing HR activities.");
    }
    public void addEmployee(String newEmployeeName) {
        System.out.println(name + " added a new employee: " + newEmployeeName);
    }
}

public class EmployeeDemo {
    public static void main(String[] args) {
        Employee emp = new Employee("John", 40000);
        emp.work();
        System.out.println("Salary: $" + emp.getSalary());
        System.out.println();
        HRManager hr = new HRManager("Alice", 60000);
        hr.work();
        System.out.println("Salary: $" + hr.getSalary());
        hr.addEmployee("David");
    }
}

```

**OUTPUT :**

```

PS C:\Users\91944> & 'C:\Program Files\Java
91944\AppData\Local\Temp\vscodesws_2694d\jdk
John is working.
Salary: $40000.0

Alice is managing HR activities.
Salary: $60000.0
Alice added a new employee: David
PS C:\Users\91944>

```

#### error

SI.NO	ERROR	ERROR RECTIFICATION
1	Class name (case Sensitivity)	Corrected
2	In correct Calling methods	corrected

#### Important points

- 1.A subclass (HRManager) inherits methods and properties from a superclass (Employee).
2. A subclass can provide its own implementation of a superclass method

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

#### **CODE :**

```

import java.util.Scanner;

class Addition {
    public double add(double a, double b) {
        return a + b;
    }
}

class Subtraction extends Addition {
    public double subtract(double a, double b) {
        return a - b;
    }
}

class Multiplication extends Subtraction {
    public double multiply(double a, double b) {
        return a * b;
    }
}

class Division extends Multiplication {
    public double divide(double a, double b) {

```

```

        if (b == 0) {
            System.out.println("Cannot divide by zero.");
            return 0;
        }
        return a / b;
    }
}

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: ");
        double num1 = scanner.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = scanner.nextDouble();

        System.out.println("Addition: " + calc.add(num1, num2));
        System.out.println("Subtraction: " + calc.subtract(num1, num2));
        System.out.println("Multiplication: " + calc.multiply(num1, num2));
        System.out.println("Division: " + calc.divide(num1, num2));

        scanner.close();
    }
}

```

### OUTPUT :

```

PS C:\Users\91944> & 'C:\Program
91944\AppData\Local\Temp\vscode
Enter first number: 5
Enter second number: 1
Addition: 6.0
Subtraction: 4.0
Multiplication: 5.0
Division: 5.0
PS C:\Users\91944> 

```

14. Consider a software system for a company that manages its employees. The company categorizes its employees into two primary types: **RegularEmployee** and **Manager**. Both types of employees share common attributes such as name and employee ID, but managers have attributes such as a bonus. You are tasked with designing the Java classes for this scenario and add up the salary for each type.

### CODE:

```

class company{
    String name;
    String employee_id;
    company(String name,String employee_id){
        this.name=name;
        this.employee_id=employee_id;
        System.out.println("name of the employee:"+name);
        System.out.println(" employee ID:"+employee_id);
    }
    void regularemployee(){
        System.out.println("he is regularemployee");
        System.out.println("salary:1,25,00");

    }
    void managers(){
        int bonus;
        System.out.println("he is a manager");
        System.out.println("salary:1,80,789");
    }
}
class Main14{
    public static void main(String[] args) {
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
        company c1=new company("max","ABC779D3");
        c1.regularemployee();
        c1.managers();
    }
}

```

### OUTPUT :

```

Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
name of the employee:max
employee ID:ABC779D3

```

```

he is regularemployee
salary:1,25,00
he is a manager
salary:1,80,789

```

SI.NO	ERROR	ERROR RECTIFICATION
1	Class name and file name mismatch	The file name matches the class name (my class) must be saved in myclass.java
2	Missing semicolon (;)	all statements e end withsemicolon.

### Important points

**1.Encapsulation** – Data (name, ID, salary) is stored in instance variables and accessed via methods

**15. A superclass named “Shapes” has a method called “area()”. Subclasses of “Shapes” can be “Triangle”, “circle”, “Rectangle”, etc. Each subclass has its own way of calculating area. Using base class as Shapes with subclasses triangle, circle and rectangle, use overriding polymorphism and find the area for each shape.**

### **CODE:**

```

abstract class Shapes {
    abstract double area();
}
class Triangle extends Shapes {
    double base, height;
    Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }
    double area() {
        return 0.5 * base * height;
    }
}
class Circle extends Shapes {
    double radius;
    Circle(double radius) {

```

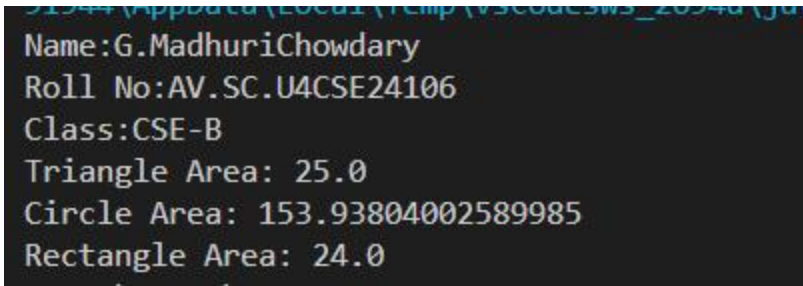
```

        this.radius = radius;
    }
    double area() {
        return Math.PI * radius * radius;
    }
}
class Rectangle extends Shapes {
    double length, width;
    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
    double area() {
        return length * width;
    }
}
public class Main15 {
    public static void main(String[] args) {
        Shapes triangle = new Triangle(10, 5);
        Shapes circle = new Circle(7);
        Shapes rectangle = new Rectangle(4, 6);
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");

        System.out.println("Triangle Area: " + triangle.area());
        System.out.println("Circle Area: " + circle.area());
        System.out.println("Rectangle Area: " + rectangle.area());
    }
}

```

### OUTPUT:



```

Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Triangle Area: 25.0
Circle Area: 153.93804002589985
Rectangle Area: 24.0

```



### Errors

Sl.NO	Error	Error rectification
<u>1</u>	Case sensitive	corrected
<u>2</u>	Class file not found	corrected

**16. creating one superclass Animal and three subclasses, Herbivores, Carnivores, and Omnivores. Subclasses extend the superclass and override its eat() method. Returning the method for the required type of animals.**

### **CODE:**

```
class Animal {
    public String eat() {
        return "This animal eats food.";
    }
}

class Herbivores extends Animal {
    @Override
    public String eat() {
        return "Herbivores eat plants.";
    }
}

class Carnivores extends Animal {
    @Override
    public String eat() {
        return "Carnivores eat meat.";
    }
}

class Omnivores extends Animal {
    @Override
    public String eat() {
        return "Omnivores eat both plants and meat.";
    }
}

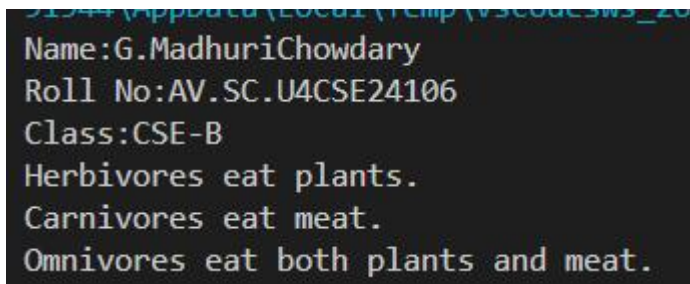
public class Main16 {
    public static void main(String[] args) {
        Animal herbivore = new Herbivores();
        Animal carnivore = new Carnivores();
    }
}
```

```

Animal omnivore = new Omnivores();
System.out.println("Name:G.MadhuriChowdary");
System.out.println("Roll No:AV.SC.U4CSE24106");
System.out.println("Class:CSE-B");
System.out.println(herbivore.eat());
System.out.println(carnivore.eat());
System.out.println(omnivore.eat());
}
}

```

### OUTPUT:



```

Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Herbivores eat plants.
Carnivores eat meat.
Omnivores eat both plants and meat.

```

### Errors

SI.NO	ERRORS	ERROR RECTIFICATION
1	Class names should follow PascalCase	Always start <b>class names</b> with an uppercase letter.
2	if an object isn't properly initialized,	Always <b>initialize</b> objects before calling methods on them.

### Important points

1. Inheritance (extends) → Subclasses (Herbivores, Carnivores, Omnivores) inherit from Animal.
2. Polymorphism → Using Animal references to call subclass methods dynamically.
3. Encapsulation → Restricting access to superclass properties/method

**17. Write a Java program to create an abstract class Animal with an abstract method called sound(). Create subclasses Lion and Tiger that extend the Animal class and implement the sound() method to make a specific sound for each animal.**

### CODE:

```

abstract class Animal {
    abstract void sound();
}

```

```

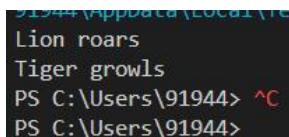
}
class Lion extends Animal {
    void sound() {
        System.out.println("Lion roars");
    }
}
class Tiger extends Animal {
    void sound() {
        System.out.println("Tiger growls");
    }
}

public class Main17 {
    public static void main(String[] args) {
        Animal lion = new Lion();
        Animal tiger = new Tiger();

        lion.sound();
        tiger.sound();
    }
}

```

### OUTPUT:



```

91944\AppData\Local\Te
Lion roars
Tiger growls
PS C:\Users\91944> ^C
PS C:\Users\91944>

```

### Errors

SI.NO	ERROR	ERROR RECTIFICATION
1	Case sensitive	Fixed method name consistency
2	Improper class naming	Renamed Main to AnimalTest for clarity

**Important points** : `Animal` is an abstract class, so it cannot be instantiate

18. Write a Java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extend the Shape3D class and implement the respective methods to calculate the volume and surface area of each shape.

**CODE:**

```
abstract class Shape3D {
    abstract double calculateVolume();
    abstract double calculateSurfaceArea();
}

class Sphere extends Shape3D {
    double radius;

    Sphere(double radius) {
        this.radius = radius;
    }

    double calculateVolume() {
        return (4.0 / 3.0) * 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 Main18 {

    public static void main(String[] args) {
        Shape3D sphere = new Sphere(5);
        Shape3D cube = new Cube(3);
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");

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

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

```

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

```

### OUTPUT:

```

Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Sphere Volume: 523.5987755982989
Sphere Surface Area: 314.1592653589793
Cube Volume: 27.0
Cube Surface Area: 54.0
PS C:\Users\91944>

```

### Errors

Sl.NO	ERROR	ERROR RECTIFICATION
1	System.out.println("Sphere Surface Area: " + sphere.calculateSurfaceArea())	System.out.println("Sphere Surface Area: " + sphere.calculateSurfaceArea());
2	Shape3D shape = new Shape3D();	Shape3D sphere = new Sphere(5);

### Important points

1. Abstract Class & Method Usage
2. Method Overriding in Subclasses
3. . Constructor Usage

### 19. What will be the output of the following program?

**interface A**

```

{
    void Method ();
}

```

**class B**

```

{
    public void Method ()
    {
        System.out.println ("My Method");
    }
}

```

**class C extends B implements A**

```

{

```

```

}
class Main
{
    public static void main (String [] args)
    {
        A a = new C ();
        a. Method ();
    }
}

```

#### CODE:

```

interface A {
    void Method();
}
class B {
    public void Method() {
        System.out.println("My Method");
    }
}
class C extends B implements A {
}
class Main19 {
    public static void main(String[] args) {
        A a = new C();
        a.Method();
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
    }
}

```

#### OUTPUT:

```

My Method
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
PS C:\Users\91944>

```

19. Does below code compile successfully? If not, why?

```

interface A

```

```

{

```

```

    int i = 111;
}
class B implements A
{
    void methodB()
    {
        i = 222;
    }
}

```

#### CODE :

```

interface A {
    int i = 111;.
}
class B implements A {
    void methodB() {
        i = 222;
    }
}

```

#### COMPILATION ERROR:

##### NOTE

The code does not compile successfully because:

Interface variables are implicitly final (constants)

The code attempts to modify final variable i in methodB()

error: final field i cannot be assigned

```

    i = 222;

```

^

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

#### CODE :

```

interface Shape {
    double getPerimeter();
}

```

```
class Rectangle implements Shape {  
    double length, width;
```

```
    Rectangle(double length, double width) {  
        this.length = length;  
        this.width = width;  
    }  
}
```

```
    @Override  
    public double getPerimeter() {  
        return 2 * (length + width);  
    }  
}
```

```
class Circle implements Shape {  
    double radius;
```

```
    Circle(double radius) {  
        this.radius = radius;  
    }  
}
```

```
    @Override  
    public double getPerimeter() {  
        return 2 * Math.PI * radius;  
    }  
}
```

```
class Triangle implements Shape {  
    double side1, side2, side3;
```

```
    Triangle(double side1, double side2, double side3) {  
        this.side1 = side1;  
        this.side2 = side2;  
        this.side3 = side3;  
    }  
}
```

```
    @Override  
    public double getPerimeter() {  
        return side1 + side2 + side3;  
    }  
}
```

```
public class Main20 {  
    public static void main(String[] args) {  
        Shape rectangle = new Rectangle(5, 3);  
        Shape circle = new Circle(7);  
        Shape triangle = new Triangle(3, 4, 5);  
    }  
}
```



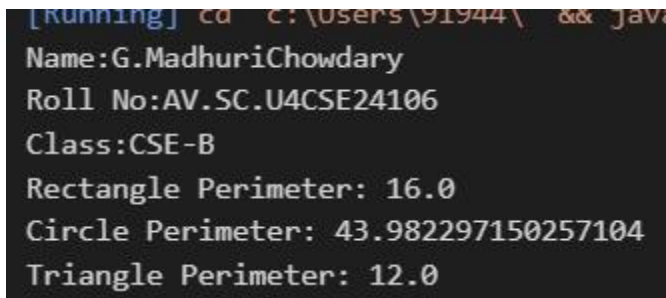
```

        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");

        System.out.println("Rectangle Perimeter: " + rectangle.getPerimeter());
        System.out.println("Circle Perimeter: " + circle.getPerimeter());
        System.out.println("Triangle Perimeter: " + triangle.getPerimeter());
    }
}

```

## OUTPUT:



```

[Running] cd C:\Users\91944\ && java
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Rectangle Perimeter: 16.0
Circle Perimeter: 43.982297150257104
Triangle Perimeter: 12.0

```

## Errors

Sl.NO	ERROR	ERROR RECTIFICATION
1	triangle.getperimeter(); (Wrong - Java is case-sensitive)	triangle.getPerimeter();
2	Inconsistent Formatting & Spacing (Not an Error but Improves Readability)	Keep proper indentation and spacing for better readability.

## Important points

- 1.Method Name Should Match Exactly
2. Use Math.PI Instead of 3.1416

**21. Write a Java program that creates a class hierarchy for employees of a company. The base class should be Employee, with subclasses Manager, Developer, and Programmer. Each subclass should have properties such as name, address, salary, and job title. Implement methods for calculating bonuses, generating performance reports, and managing projects**

## CODE :

```

abstract class Employee {
    String name;
    String address;
    double salary;
}

```

```
String jobTitle;
```

```
Employee(String name, String address, double salary, String jobTitle) {  
    this.name = name;  
    this.address = address;  
    this.salary = salary;  
    this.jobTitle = jobTitle;  
}
```

```
abstract double calculateBonus();
```

```
abstract String generatePerformanceReport();
```

```
abstract void manageProject();
```

```
}
```

```
class Manager extends Employee {
```

```
    Manager(String name, String address, double salary) {
```

```
        super(name, address, salary, "Manager");
```

```
    }
```

```
    @Override
```

```
    double calculateBonus() {
```

```
        return salary * 0.2;
```

```
    }
```

```
    @Override
```

```
    String generatePerformanceReport() {
```

```
        return "Manager " + name + "'s team exceeded project goals.";
```

```
    }
```

```
    @Override
```

```
    void manageProject() {
```

```
        System.out.println("Manager " + name + " is overseeing the entire project.");
```

```
    }
```

```
}
```

```
class Developer extends Employee {
```

```
    Developer(String name, String address, double salary) {
```

```
        super(name, address, salary, "Developer");
```

```
    }
```

```
    @Override
```

```
    double calculateBonus() {
```

```
        return salary * 0.15;
```

```
    }
```

```
    @Override
```

```

String generatePerformanceReport() {
    return "Developer " + name + " delivered all modules on time.";
}

@Override
void manageProject() {
    System.out.println("Developer " + name + " is managing backend modules.");
}
}

class Programmer extends Employee {
    Programmer(String name, String address, double salary) {
        super(name, address, salary, "Programmer");
    }

    @Override
    double calculateBonus() {
        return salary * 0.1;
    }

    @Override
    String generatePerformanceReport() {
        return "Programmer " + name + " maintained code quality throughout.";
    }

    @Override
    void manageProject() {
        System.out.println("Programmer " + name + " is writing and debugging code.");
    }
}

public class Main21 {
    public static void main(String[] args) {
        Employee manager = new Manager("Alice", "New York", 80000);
        Employee developer = new Developer("Bob", "San Francisco", 70000);
        Employee programmer = new Programmer("Charlie", "Chicago", 60000);

        Employee[] employees = { manager, developer, programmer };

        for (Employee emp : employees) {
            System.out.println("Name: " + emp.name);
            System.out.println("Address: " + emp.address);
            System.out.println("Job Title: " + emp.jobTitle);
            System.out.println("Salary: $" + emp.salary);
            System.out.println("Bonus: $" + emp.calculateBonus());
            System.out.println("Performance Report: " + emp.generatePerformanceReport());
            emp.manageProject();
        }
    }
}

```

```

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

```

### OUTPUT :

```

Name: Alice
Address: New York
Job Title: Manager
Salary: $80000.0
Bonus: $16000.0
Performance Report: Manager Alice's team exceeded project goals.
Manager Alice is overseeing the entire project.
-----
Name: Bob
Address: San Francisco
Job Title: Developer
Salary: $70000.0
Bonus: $10500.0
Performance Report: Developer Bob delivered all modules on time.
Developer Bob is managing backend modules.
-----
Name: Charlie
Address: Chicago
Job Title: Programmer
Salary: $60000.0
Bonus: $6000.0
Performance Report: Programmer Charlie maintained code quality throughout.
Programmer Charlie is writing and debugging code.

```

### Errors

SI.NO	ERROR	ERROR RECTIFICATION
1	EmmployeeTest` is not matching	Rename to `EmployeeTest`
2	Messy Code Layout	Proper spacing and indentation for neatness

### Important points

- 1.Proper indentation makes the code readable and clean
2. Use method overriding correctly for `calculateBonus()` in subclasses

**22. Write a Java program to create a class called Student with private instance variables student\_id, student\_name, and grades. Provide public getter and setter methods to access and modify the student\_id and student\_name variables. However, provide a method called addGrade() that allows adding a grade to the grades variable while performing additional validation.**

### CODE :

```

import java.util.ArrayList;
import java.util.List;

class Student {

```

```

private int student_id;
private String student_name;
private List<Double> grades;
public Student(int student_id, String student_name) {
    this.student_id = student_id;
    this.student_name = student_name;
    this.grades = new ArrayList<>();
}

public int getStudentId() {
    return student_id;
}
public void setStudentId(int student_id) {
    this.student_id = student_id;
}
public String getStudentName() {
    return student_name;
}
public void setStudentName(String student_name) {
    this.student_name = student_name;
}
public boolean addGrade(double grade) {
    if (grade >= 0.0 && grade <= 100.0) {
        grades.add(grade);
        return true;
    } else {
        System.out.println("Invalid grade: " + grade + ". Grade must be between 0 and 100.");
        return false;
    }
}
public List<Double> getGrades() {
    return grades;
}
}

public class Main22 {
    public static void main(String[] args) {
        Student student = new Student(101, "John Doe");

        student.addGrade(95.5);
        student.addGrade(88.0);
        student.addGrade(-10); // Invalid
        student.addGrade(102); // Invalid
        student.addGrade(76.5);
    }
}

```

```

        System.out.println("Student ID: " + student.getId());
        System.out.println("Student Name: " + student.getName());
        System.out.println("Grades: " + student.getGrades());
    }
}

```

### OUTPUT :

```

Invalid grade: -10.0. Grade must be between 0 and 100.
Invalid grade: 102.0. Grade must be between 0 and 100.
Student ID: 101
Student Name: John Doe
Grades: [95.5, 88.0, 76.5]

```

### Errors

SI.NO	ERRORS	ERROR RECTIFICATION
1	Incorrect constructor syntax with a semicolon.	Define the constructor properly:
2	Incorrect method implementation and missing validation.	Implement the addGrade method with validation:
3	Incorrect method name and usage of undefined variables.	Implement the method correctly

### Important points

1. Declare class variables as private to protect data integrity.
2. Provide public getter and setter methods for controlled access.

**23. Write a Java program to create a base class BankAccount with methods deposit() and withdraw(). Create two subclasses SavingsAccount and CheckingAccount. Override the withdraw() method in each subclass to impose different withdrawal limits and fees.**

### CODE :

```

class BankAccount {
    protected double balance;

    BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) {

```

```

        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew: $" + amount);
        } else {
            System.out.println("Insufficient funds or invalid amount.");
        }
    }

    public double getBalance() {
        return balance;
    }
}

class SavingsAccount extends BankAccount {
    private final double withdrawalLimit = 1000;

    SavingsAccount(double initialBalance) {
        super(initialBalance);
    }

    @Override
    public void withdraw(double amount) {
        if (amount > withdrawalLimit) {
            System.out.println("Withdrawal exceeds the savings account limit of $" +
withdrawalLimit);
        } else if (amount <= balance) {
            balance -= amount;
            System.out.println("Savings Withdrawal: $" + amount);
        } else {
            System.out.println("Insufficient funds in savings account.");
        }
    }
}

class CheckingAccount extends BankAccount {
    private final double fee = 1.50;

    CheckingAccount(double initialBalance) {

```

```

    super(initialBalance);
}

@Override
public void withdraw(double amount) {
    double totalAmount = amount + fee;
    if (amount <= 0) {
        System.out.println("Invalid withdrawal amount.");
    } else if (totalAmount <= balance) {
        balance -= totalAmount;
        System.out.println("Checking Withdrawal: $" + amount + " (Fee: $" + fee + ")");
    } else {
        System.out.println("Insufficient funds in checking account.");
    }
}
}

public class Main23 {
    public static void main(String[] args) {
        SavingsAccount savings = new SavingsAccount(1500);
        CheckingAccount checking = new CheckingAccount(1000);

        savings.deposit(500);
        savings.withdraw(1200);
        savings.withdraw(2000);

        checking.deposit(300);
        checking.withdraw(100);
        checking.withdraw(1300);

        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
        System.out.println("Savings Balance: $" + savings.getBalance());
        System.out.println("Checking Balance: $" + checking.getBalance());
    }
}

```

### OUTPUT :

```

Deposited: $500.0
Withdrawal exceeds the savings account limit of $1000.0
Withdrawal exceeds the savings account limit of $1000.0
Deposited: $300.0
Checking Withdrawal: $100.0 (Fee: $1.5)
Insufficient funds in checking account.
Name:G.MadhuriChowdary

```



Roll No:AV.SC.U4CSE24106  
Class:CSE-B  
Savings Balance: \$2000.0  
Checking Balance: \$1198.5

### Errors

SI.NO	ERRORS	ERROR RECTIFICATION
1	Calling getbalance() instead of the correct method name getBalance().	Use the correct method name:
2	The filename does not match	the filename matches

### Important points

1. Both subclasses override the withdraw method to implement specific behaviors
2. Subclasses call the constructor of the superclass using super(initialBalance), ensuring that the balance is properly initialized

**24. Write a Java program to create an abstract class Bird with abstract methods fly() and makeSound(). Create subclasses Eagle and Hawk that extend the Bird class and implement the respective methods to describe how each bird flies and makes a sound.**

### **CODE :**

```
abstract class Bird {
    abstract void fly();
    abstract void makeSound();
}

class Eagle extends Bird {
    @Override
    void fly() {
        System.out.println("Eagle soars high with powerful wing strokes.");
    }

    @Override
    void makeSound() {
        System.out.println("Eagle screeches sharply.");
    }
}

class Hawk extends Bird {
    @Override
```

```

void fly() {
    System.out.println("Name:G.MadhuriChowdary");
    System.out.println("Roll No:AV.SC.U4CSE24106");
    System.out.println("Class:CSE-B");
    System.out.println("Hawk glides smoothly and swiftly through the air.");
}

@Override
void makeSound() {
    System.out.println("Hawk gives a piercing scream.");
}
}

public class Main24 {
    public static void main(String[] args) {
        Bird eagle = new Eagle();
        Bird hawk = new Hawk();

        eagle.fly();
        eagle.makeSound();

        hawk.fly();
        hawk.makeSound();
    }
}

```

## OUTPUT :

```

Eagle soars high with powerful wing strokes.
Eagle screeches sharply.
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Hawk glides smoothly and swiftly through the air.
Hawk gives a piercing scream.

```

## Errors

SI.NO	ERRORS	ERRORS RECTIFICATION
1	Case sensitive	Fixed method name consistency
2	Improper class naming	Renamed Main to Main_v3 for clarity

## Important points

1. Ensure all names are spelled correctly.

2. Java is case-sensitive; VariableName and variablename are different

**25. Write a Java program to create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play() method to play the respective sports.**

**CODE :**

```
interface Playable {
    void play();
}

class Football implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Football on the field.");
        System.out.println("Name:G.MadhuriChowdary");
        System.out.println("Roll No:AV.SC.U4CSE24106");
        System.out.println("Class:CSE-B");
    }
}

class Volleyball implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Volleyball on the court.");
    }
}

class Basketball implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Basketball in the arena.");
    }
}

public class Main25 {
    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 :

```
Playing Football on the field.  
Name:G.MadhuriChowdary  
Roll No:AV.SC.U4CSE24106  
Class:CSE-B  
Playing Volleyball on the court.  
Playing Basketball in the arena.
```

### Errors

Sl.NO	ERROR	ERROR RECTIFICATION
1	Case sensitive	corrected
2	Class file not found	corrected

### Important points

1. An interface named Playable is defined with an abstract method play().

**26. Write a Java programming to create a banking system with three classes - Bank, Account, SavingsAccount, and CurrentAccount. The bank should have a list of accounts and methods for adding them. Accounts should be an interface with methods to deposit, withdraw, calculate interest, and view balances. SavingsAccount and CurrentAccount should implement the Account interface and have their own unique methods.**

### CODE:

```
import java.util.*;  
  
interface Account {  
    void deposit(double amount);  
    void withdraw(double amount);  
    double calculateInterest();  
    double getBalance();  
}  
  
class SavingsAccount implements Account {  
    private double balance;  
    private final double interestRate = 0.03;  
  
    public SavingsAccount(double initialBalance) {  
        this.balance = initialBalance;  
    }  
}
```

```
public void deposit(double amount) {
    balance += amount;
}

public void withdraw(double amount) {
    if (amount <= balance) balance -= amount;
    else System.out.println("Insufficient funds in Savings Account.");
}

public double calculateInterest() {
    return balance * interestRate;
}

public double getBalance() {
    return balance;
}

public void applyAnnualInterest() {
    balance += calculateInterest();
}
}

class CurrentAccount implements Account {
    private double balance;
    private final double overdraftLimit = 500;

    public CurrentAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        if (amount <= balance + overdraftLimit) balance -= amount;
        else System.out.println("Withdrawal exceeds overdraft limit.");
    }

    public double calculateInterest() {
        return 0;
    }

    public double getBalance() {
        return balance;
    }
}
```

```

    }

    public double getOverdraftLimit() {
        return overdraftLimit;
    }
}

class Bank {
    private List<Account> accounts;

    public Bank() {
        accounts = new ArrayList<>();
    }

    public void addAccount(Account account) {
        accounts.add(account);
    }

    public void showAllBalances() {
        for (Account acc : accounts) {
            System.out.println("Balance: $" + acc.getBalance());

            System.out.println("Name:G.MadhuriChowdary");
            System.out.println("Roll No:AV.SC.U4CSE24106");
            System.out.println("Class:CSE-B");
        }
    }
}

public class Main26 {
    public static void main(String[] args) {
        Bank bank = new Bank();
        Account savings = new SavingsAccount(1000);
        Account current = new CurrentAccount(500);

        bank.addAccount(savings);
        bank.addAccount(current);

        savings.deposit(200);
        current.withdraw(300);
        ((SavingsAccount) savings).applyAnnualInterest();

        bank.showAllBalances();
    }
}

```

```
}
```

### OUTPUT:

```
Balance: $1236.0
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
Balance: $200.0
Name:G.MadhuriChowdary
Roll No:AV.SC.U4CSE24106
Class:CSE-B
```

### Errors

SI.NO	ERRORS	ERROR RECTIFICATION
1	balance has private access in SavingsAccount	use public methods instead
2	',' expected	Check every line – especially after:

### Important points

1. Account is an interface, not a class.
2. SavingsAccount and CurrentAccount implement the Account interface.

**27. How would you demonstrate the initialization and usage of arrays in Java? Discuss the various methods of declaring, initializing, and populating arrays. Using the arrays concept write a java program to initialize a matrix, addition of two matrices, multiplication of two matrices and display the output.**

### CODE:

```
public class MatrixOperations {

    public static void main(String[] args) {
        int[][] matrix1 = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
    }
}
```

```

int[][] matrix2 = {
    {9, 8, 7},
    {6, 5, 4},
    {3, 2, 1}
};

int[][] sum = addMatrices(matrix1, matrix2);
int[][] product = multiplyMatrices(matrix1, matrix2);

System.out.println("Matrix 1:");
displayMatrix(matrix1);
System.out.println("Matrix 2:");
displayMatrix(matrix2);
System.out.println("Sum of matrices:");
displayMatrix(sum);
System.out.println("Product of matrices:");
displayMatrix(product);
}

public static int[][] addMatrices(int[][] m1, int[][] m2) {
    int rows = m1.length;
    int cols = m1[0].length;
    int[][] result = new int[rows][cols];

    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            result[i][j] = m1[i][j] + m2[i][j];

    return result;
}

public static int[][] multiplyMatrices(int[][] m1, int[][] m2) {
    int rows = m1.length;
    int cols = m2[0].length;
    int[][] result = new int[rows][cols];

    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            for (int k = 0; k < m2.length; k++)
                result[i][j] += m1[i][k] * m2[k][j];

    return result;
}

```



```

public static void displayMatrix(int[][] matrix) {
    for (int[] row : matrix) {
        for (int val : row)
            System.out.print(val + " ");
        System.out.println();
    }
}
}

```

### OUTPUT:

```

Matrix 1:
1 2 3
4 5 6
7 8 9
Matrix 2:
9 8 7
6 5 4
3 2 1
Sum of matrices:
10 10 10
10 10 10
10 10 10
Product of matrices:
30 24 18
84 69 54
138 114 90

```

### Errors:

Sl.NO	ERROR	ERROR RECTIFICATION
1.	Incomplete for loop at the end	Complete the loop or remove the incomplete line
2.	printMatrix method is missing	Add the method printMatrix() at the end

### Important points

1. Arrays are used to store multiple values of the same data type in a single variable.
2. Arrays can be one-dimensional (like a list) or multi-dimensional (like a matrix).
3. In Java, arrays have a fixed size once declared

**28. a. Discuss the difference between the Interfaces vs. Abstract Classes in detail.**

**b. Discuss the difference between the Overriding vs. Overloading in detail.**

**Table A:**

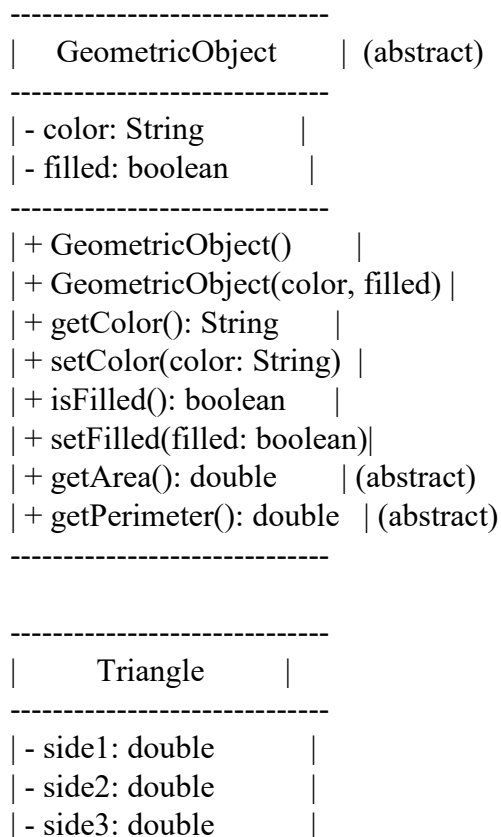
Feature	Interface	Abstract Class
Definition	A blueprint of a class that contains only abstract methods (by default) and constants.	A class that cannot be instantiated and can contain both abstract methods and concrete (implemented) methods.
Methods	Only abstract methods (until Java 8); from Java 8 onwards, default and static methods are allowed.	Can have both abstract and non-abstract methods.
Fields	Variables are <code>public</code> , <code>static</code> , and <code>final</code> by default.	Can have instance variables with any access modifier.
Multiple Inheritance	A class can implement multiple interfaces (supports multiple inheritance).	A class can inherit only one abstract class (single inheritance).
Constructors	Interfaces cannot have constructors.	Abstract classes can have constructors.
When to Use	When you want to specify behavior without worrying about how it's implemented.	When you want to share code among several closely related classes.

**Table B:**

Feature	Method Overriding	Method Overloading
Definition	Redefining a superclass method in a subclass.	Defining multiple methods with the same name but different parameters within the same class.
Inheritance	Requires inheritance (child class must override parent class method).	No inheritance required. Can occur in the same class.
Method Signature	Must have the same method name, return type, and parameters.	Same method name, but different parameter list (type, number, or order).
Runtime vs Compile-time	Happens at runtime (Dynamic polymorphism).	Happens at compile-time (Static polymorphism).
Return Type	Must be the same or subtype (covariant return).	Can have different return types (but must differ in parameters).

29. (Triangle class) Design a new Triangle class that extends the abstract GeometricObject class. Draw the UML diagram for the classes Triangle and GeometricObject and then implement the Triangle class. Write a test program that prompts the user to enter three sides of the triangle, a color, and a Boolean value to indicate whether the triangle is filled. The program should create a Triangle object with these sides and set the color and filled properties using the input. The program should display the area, perimeter, color, and true or false to indicate whether it is filled or not.

#### UML DAIGRAM :



```
-----  
| + Triangle(s1, s2, s3) |  
| + getArea(): double   |  
| + getPerimeter(): double |  
-----
```

### CODE:

```
import java.util.Scanner;  
abstract class GeometricObject {  
    private String color;  
    private boolean filled;  
    public GeometricObject() {  
        this.color = "white";  
        this.filled = false;  
    }  
    public GeometricObject(String color, boolean filled) {  
        this.color = color;  
        this.filled = filled;  
    }  
    public String getColor() {  
        return color;  
    }  
    public void setColor(String color) {  
        this.color = color;  
    }  
    public boolean isFilled() {  
        return filled;  
    }  
    public void setFilled(boolean filled) {  
        this.filled = filled;  
    }  
    public String toString() {  
        return "Color: " + color + ", Filled: " + filled;  
    }  
    public abstract double getArea();  
    public abstract double getPerimeter();  
}  
class Triangle extends GeometricObject {  
    private double side1;  
    private double side2;  
    private double side3;  
    public Triangle() {  
        this(1.0, 1.0, 1.0);  
    }  
    public Triangle(double side1, double side2, double side3) {
```

```

this.side1 = side1;
this.side2 = side2;
this.side3 = side3;
}
public double getSide1() {
return side1;
}
public double getSide2() {
return side2;
}
public double getSide3() {
return side3;
}
@Override
public double getArea() {
double s = (side1 + side2 + side3) / 2;
return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
}
@Override
public double getPerimeter() {
return side1 + side2 + side3;
}
@Override
public String toString() {
return "Triangle: side1 = " + side1 + " side2 = " + side2 +
" side3 = " + side3 + " " + super.toString();
}
}
public class Main29 {
public static void main(String[] args) {
System.out.println("Name MADHURI CHOWDARY; Roll No 24106; Sec CSE-B");

```

```

Scanner scanner = new Scanner(System.in);
System.out.print("Enter side 1 of the triangle: ");
double side1 = scanner.nextDouble();
System.out.print("Enter side 2 of the triangle: ");
double side2 = scanner.nextDouble();
System.out.print("Enter side 3 of the triangle: ");
double side3 = scanner.nextDouble();
System.out.print("Enter the color of the triangle: ");
String color = scanner.next();
System.out.print("Is the triangle filled? (true/false): ");
boolean filled = scanner.nextBoolean();
Triangle triangle = new Triangle(side1, side2, side3);
triangle.setColor(color);

```

```

triangle.setFilled(filled);
System.out.println(triangle.toString());
System.out.println("Area: " + triangle.getArea());
System.out.println("Perimeter: " + triangle.getPerimeter());
}
}

```

### OUTPUT:

```

Name MADHURI CHOWDARY; Roll No 24106; Sec CSE B
Enter side 1 of the triangle: 3
Enter side 2 of the triangle: 3
Enter side 3 of the triangle:
3

```

### Important Points:

- This Keyword - identifies the attributes & methods of the object
- Constructor - a special method used to set the book's title, author, and year

**30. Rewrite the PrintCalendar class in Listing 6.12 to display a calendar for a specified month using the Calendar and GregorianCalendar classes. Your program receives the month and year from the command line. For**

**example:**

**java Exercise13\_04 5 2016**

**This displays the calendar shown in Figure.**

### CODE:

```

import java.util.Calendar;
import java.util.GregorianCalendar;

public class PrintCalendar {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java PrintCalendar month year");
            System.exit(1);
        }

        int month = Integer.parseInt(args[0]);
        int year = Integer.parseInt(args[1]);

        Calendar calendar = new GregorianCalendar(year, month - 1, 1);
        printMonthHeader(year, month);

        int startDay = calendar.get(Calendar.DAY_OF_WEEK);
        int numberOfDaysInMonth = calendar.getActualMaximum(Calendar.DAY_OF_MONTH);

        printMonthBody(startDay, numberOfDaysInMonth);
    }
}

```

```

}

public static void printMonthHeader(int year, int month) {
    String[] monthNames = {
        "", "January", "February", "March", "April", "May", "June",
        "July", "August", "September", "October", "November", "December"
    };

    System.out.println("      " + monthNames[month] + " " + year);
    System.out.println("-----");
    System.out.println("Sun Mon Tue Wed Thu Fri Sat");
}

public static void printMonthBody(int startDay, int numberOfDaysInMonth) {
    for (int i = 1; i < startDay; i++) {
        System.out.print("  ");
    }

    for (int day = 1; day <= numberOfDaysInMonth; day++) {
        System.out.printf("%3d ", day);
        if ((day + startDay - 1) % 7 == 0) {
            System.out.println();
        }
    }

    System.out.println();
}
}

```

## OUTPUT:

```

C:\assignment>javac PrintCalendar.java
C:\assignment>java PrintCalendar 4 2025
April 2025
-----
Sun Mon Tue Wed Thu Fri Sat
   6  7  8  9 10 11 12
 13 14 15 16 17 18 19
 20 21 22 23 24 25 26
 27 28 29 30
C:\assignment>

```

## Errors

Sl.NO	ERROR	ERROR RECTIFICATION
1.	not found	Add: import Scanner java.util.Scanner;
2.	Case sensitive	corrected

## Important points:

1. Inputs are taken from the command line using args[ ].

2.args[0] = month, args[1] = year.

S.NO.	Q.No	PAGE No
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	10	
11	11	
12	12	
13	13	
14	14	
15	15	
16	16	
17	17	
18	18	
19	19	
20	20	
21	21	
22	22	
23	23	
24	24	
25	25	
26	26	
27	27	

28	28	
29	29	
30	30	