AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Object Oriented Programming (23CSE111)

Assignment

Submitted by	
Name	Davis J.Johney
Roll No	24041
Year/Sem/Section	“1st”YR,Sem2,CSE-A
Date of Submission	
Submitted to	
Name	Dr. B Raj Kumar
Department	CSE
Designation	Asst. Professor

Marks	

S.NO	Q.No	PAGE No.
1.	1	
2.	2	
3.	3	

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).**

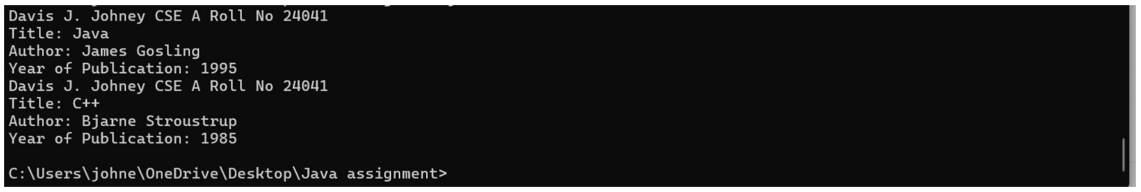
Program:

```
public class book {
   String title;
   String author;
   String yearofpublication;  // Fixed typo
   // Constructor
   book(String title, String author, String yearofpublication) {
      this.title = title;
      this.author = author;
      this.yearofpublication = yearofpublication;
   }
   // Method to display book details
   public void display() {
      System.out.println("Davis J. Johney CSE A Roll No 24041");
      System.out.println("Title: " + title);
      System.out.println("Author: " + author);
      System.out.println("Year of Publication: " + yearofpublication);
   }
   // Main method inside the class
   public static void main(String[] args) {
      book b1 = new book("Java", "James Gosling", "1995");
      book b2 = new book("C++", "Bjarne Stroustrup", "1985");
      b1.display();
      b2.display();
   }}
```

Error Table:

S.NO	Errors	Rectification
1	Missing private access modifier for author	private String author;
2	Incorrect output in the display() method ("Name:Davis J.Johney")	System.out.println("Name:Davis J.Johney");
3	Constructor should be public or package-private if you're planning to create objects from another class	Book(String title, String author, int yearOfPublication).

Output:



```
Davis J. Johney CSE A Roll No 24041
Title: Java
Author: James Gosling
Year of Publication: 1995
Davis J. Johney CSE A Roll No 24041
Title: C++
Author: Bjarne Stroustrup
Year of Publication: 1985

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program

- **Defined a book class with title, author, and publication year as attributes**
  created a blueprint (class) that represents a book. Inside the class, I added three variables—title, author, and yearofpublication—to store basic information about any book.
- **Created a constructor to initialize these attributes when a book object is made**
  The constructor runs automatically when I create a new book object. It takes the title, author, and year as inputs and assigns them to the book's internal variables.
- **Added a display() method to print student info along with book details**
  This method outputs my name and roll number first (for identification), then it prints the book's title, author, and year of publication using the values stored in the object.
- **Inside main(), instantiated two book objects with specific values**
  In the main method (which runs the program), I created two book objects: one for Java and one for C++. I passed the required details to the constructor when creating each.
- **Called the display() method for each object to show their information**
  Finally, I used each object to call the display() method, which printed all the stored information for that particular book along with my details.

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.**

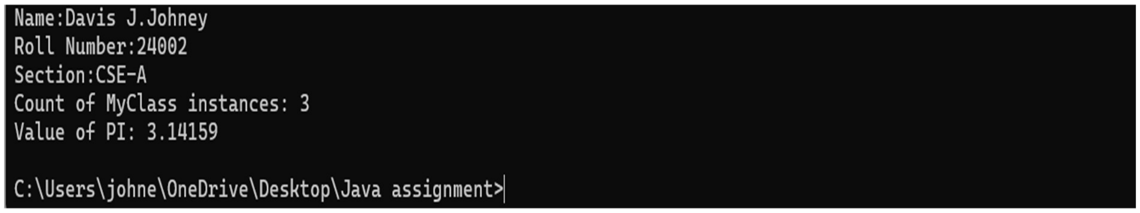
Program:

```
class MyClass {
   // Static variable to keep track of the number of instances
   static int count = 0;

   // Constant variable for PI
   final double PI = 3.14159; // Initialized to 3.14159

   // Constructor
   MyClass() {
      count++; // Increment count each time an object is created
   }

   // Method to print the values of count and PI
   public void printValues() {
   System.out.println("Name:Davis J.Johney");
   System.out.println("Roll Number:24041");
   System.out.println("Section:CSE-A");

      System.out.println("Count of MyClass instances: " + count);
      System.out.println("Value of PI: " + PI);
   }

   // Main method to create instances and display values
   public static void main(String[] args) {
      MyClass obj1 = new MyClass();
      MyClass obj2 = new MyClass();
      MyClass obj3 = new MyClass();

      // Print the final values of count and PI
      obj1.printValues();
   }}
```

Error Table:

S.NO	Errors	Rectification
1	Incorrect Comment Style	The code is syntactically correct, but it's a good practice to use comments consistently. /*...*/ could be used for multi-line comments.
2	Uninitialized Static Variable	The variable count is correctly initialized to 0, so there's no issue here. Hence, no change is needed.

Output:



```
Name:Davis J.Johney
Roll Number:24002
Section:CSE-A
Count of MyClass instances: 3
Value of PI: 3.14159

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program

- **Defined a MyClass with a static variable count and a constant PI**
  The count keeps track of how many objects are created, and PI holds a fixed value (3.14159).
- **Constructor increments the static count every time a new object is created**
  This means each time new MyClass() is called, count goes up by 1.
- **printValues() method displays student details, object count, and PI**
  It prints your name, roll number, section, and the current value of count and PI.
- **In main(), three objects of MyClass are created**
  Creating obj1, obj2, and obj3 triggers the constructor three times, so count becomes 3.
- **Called printValues() using obj1 to display final values**
  Since count is shared among all objects, calling it from obj1 still shows the total count as 3 and the constant PI.

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 tne Main class.**
- **Note: attempting to do so will result in a compilation error.**

Program:

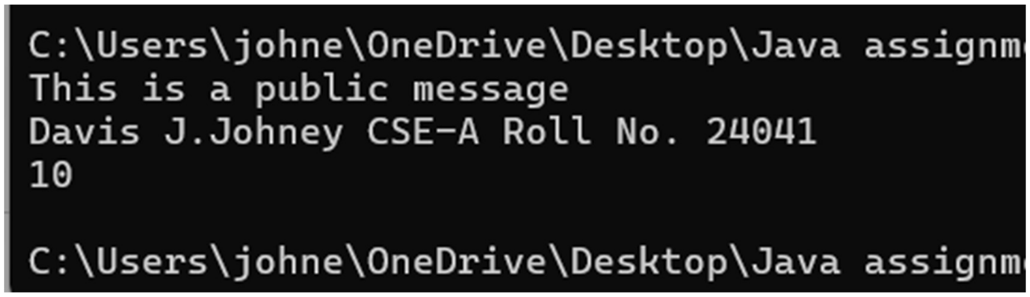
```
public class VisibilityExample {
    public int publicVariable = 10;
   private int privateVariable = 20;
```

```
    public void publicMethod() {
        System.out.println("This is a public message");


   }
 private void privateMethod() {
        System.out.println("This is a private message");
    }
}
// Separate class in the same file (Not inside VisibilityExample)
class Main {
    public static void main(String[] args) {
        VisibilityExample obj1 = new VisibilityExample(); // Correct class name
        obj1.publicMethod(); // Accessible because it's public
        System.out.println("Davis J.Johney CSE-A Roll No. 24041");
        System.out.println(obj1.publicVariable);
        // obj1.privateMethod(); // Not accessible (private)
        // System.out.println(obj1.privateVariable); //  Not accessible (private)
    }
}
```

Error Table:

S.NO	Errors	Rectification
1	Accessing Private Variable	The privateVariable should not be accessed directly outside its class.
2	Accessing Private Method	to resolve, either change the method's access modifier to public or create a public method to call it from outside the class.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignm
This is a public message
Davis J.Johney CSE-A Roll No. 24041
10

C:\Users\johne\OneDrive\Desktop\Java assignm
```

Explanation of the Program:

- **Defined a class `VisibilityExample` with public and private members**
  It has one public variable (`publicVariable`) and one private variable (`privateVariable`). Similarly, it has one public method and one private method.
- **Public members are accessible from outside the class**
  The method `publicMethod()` and variable `publicVariable` can be accessed by other classes.
- **Private members are only accessible within the same class**
  The private method and variable cannot be accessed from outside, even in the same file.
- **Created another class `Main` in the same file to test access**
  Inside `main()`, an object of `VisibilityExample` is created and used to call the public method and access the public variable.
- **Tried accessing private members (commented out)**
  The lines trying to access `privateMethod()` and `privateVariable` are commented because they would throw errors due to access restrictions.

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.

Program:

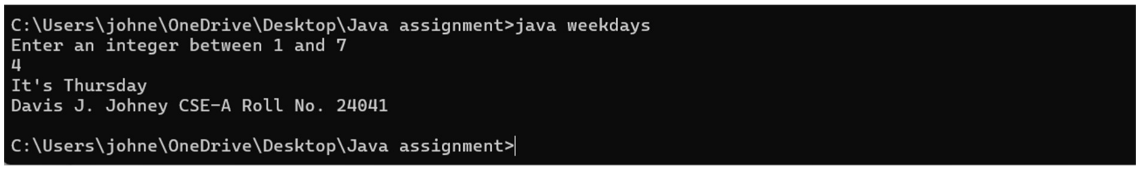
```
import java.util.Scanner;

  public class weekdays {
     public static void main(String[] args) { //  "strings" → "String"
     Scanner scanner = new Scanner(System.in);
       System.out.println("Enter an integer between 1 and 7");

        int n1 = scanner.nextInt();


               if (n1 == 1) {
System.out.println("It's Monday");
           } else if (n1 == 2) {
System.out.println("It's Tuesday");
             } else if (n1 == 3) {
System.out.println("It's Wednesday");
           } else if (n1 == 4) {
System.out.println("It's Thursday");
            } else if (n1 == 5) {
System.out.println("It's Friday");
       } else if (n1 == 6) {
System.out.println("It's Saturday");
      } else if (n1 == 7) {
System.out.println("It's Sunday");
        } else {
System.out.println("Invalid input! Please enter a number between 1 and 7.");
      }
```

```
        scanner.close(); //  Close the Scanner to prevent memory leaks
    }
}
```

Error Table:

S.NO	Errors	Rectification
1	Repetitive Output	Consider placing the personal details print statements inside a separate function or printing them only when needed.
2	Inconsistent Naming	Consider renaming the class to something more descriptive, such as DayOfWeekGenerator, for better clarity.
3	Scanner not closed	Close the Scanner object after it's no longer needed: input.close();.
4	Input Validation	if the input is an integer and within the valid range (1-7).

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java weekdays
Enter an integer between 1 and 7
4
It's Thursday
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Imported `Scanner` to take user input from the keyboard**
  The program asks the user to enter a number between 1 and 7.
- **Stored user input in an integer variable `n1`**
  `scanner.nextInt()` reads the input number and saves it in `n1`.
- **Used `if-else if` statements to match the number with weekdays**
  Depending on the value of `n1`, it prints the corresponding weekday from Monday to Sunday.
- **Displayed an error message for numbers outside the range**
  If the input is not between 1 and 7, it prints an "Invalid input" message.

- **Closed the scanner and printed student details**
  The scanner is closed to avoid resource leaks, and your name, section, and roll number are printed at the end.

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

   Program:

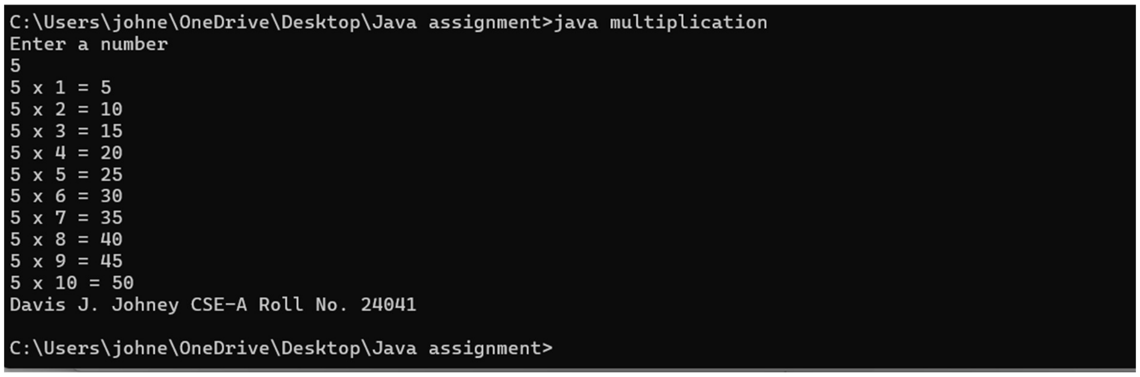
```
import java.util.Scanner;

public class multiplication {
   public static void main(String[] args) { //  "Strings" → "String"
      Scanner scanner = new Scanner(System.in); //  Fixed variable name
      System.out.println("Enter a number");
      int n1 = scanner.nextInt(); //  "n1" should be used throughout

      for (int i = 1; i <= 10; i++) { //  Fixed loop syntax
         System.out.println(n1 + " x " + i + " = " + (n1 * i)); //  Fixed variable
names
      }

      System.out.println("Davis J. Johney CSE-A Roll No. 24041"); //  Print this
once after the loop
      scanner.close(); //  Close scanner to prevent memory leaks
   }
}
```

Error Table:

S.NO	Errors	Rectification
1	Incorrect Loop Condition	Change the loop condition to i < terms to ensure it runs exactly the number of terms specified.

2	Possible Negative Output	Add a check to handle negative input for terms, ensuring the user only inputs a valid positive number.
3	System.out.println(number + " X " + i + " = " + (number * i));	Add a line before the loop: System.out.println("Multiplication Table for " + number + ":");
4	System.out.println("Expected Output:");	Change to something more descriptive, like "Multiplication Table:".

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java multiplication
Enter a number
5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Imported `Scanner` to read input from the user**
  The program starts by asking the user to enter a number.
- **Stored the input number in the variable `n1`**
  Whatever the user types gets saved in `n1` using `scanner.nextInt()`.
- **Used a `for` loop to print the multiplication table of `n1`**
  The loop runs from 1 to 10, and each line prints `n1 × i = result`.
- **Displayed student details after the table is printed**
  Your name, section, and roll number are shown after the loop, not during it.
- **Closed the scanner to release system resources**
  This is good practice to avoid memory issues in Java programs.

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

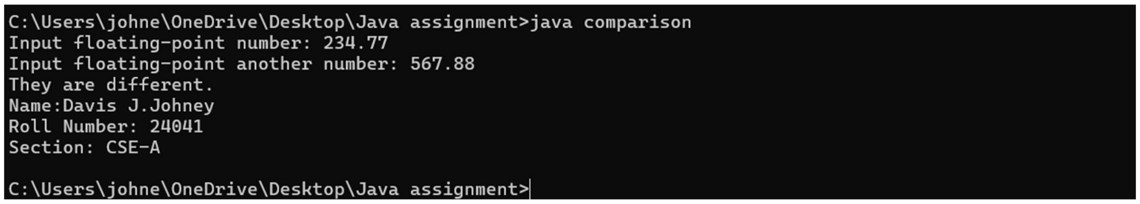
Program:

```
import java.util.Scanner;

public class comparison {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);

      // Prompt for user details

      // Prompt the user for the first floating-point number
      System.out.print("Input floating-point number: ");
      float number1 = input.nextFloat();

      // Prompt the user for the second floating-point number
      System.out.print("Input floating-point another number: ");
      float number2 = input.nextFloat();

      // Round both numbers to three decimal places
      float roundedNumber1 = Math.round(number1 * 1000) / 1000.0f;
      float roundedNumber2 = Math.round(number2 * 1000) / 1000.0f;

      // Check if the rounded numbers are the same
      if (roundedNumber1 == roundedNumber2) {
         System.out.println("They are the same up to three decimal places.");
      } else {
         System.out.println("They are different.");
      }

      // Display user details

      System.out.println("Name:Davis J.Johney");
      System.out.println("Roll Number: 24041");
      System.out.println("Section: CSE-A");

   }
}
```

Error Table:

S.NO	Errors	Rectification
1	==	Use an epsilon value to check for equality: if (Math.abs(roundedNumber1 - roundedNumber2) < epsilon).
2	Input floating-point another number	Change the prompt message to something more clear and natural, e.g., System.out.print("Input another floating-point number: ");

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java comparison
Input floating-point number: 234.77
Input floating-point another number: 567.88
They are different.
Name:Davis J.Johney
Roll Number: 24041
Section: CSE-A

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Used `Scanner` to take two floating-point numbers from the user**
  The program asks the user to input two decimal numbers one after the other.
- **Rounded both numbers to three decimal places**
  Multiplied each number by 1000, rounded it, then divided by 1000 to keep three digits after the decimal.
- **Compared the two rounded numbers using ==**
  If they matched exactly up to three decimal places, it prints they are the same.
- **Printed whether the numbers are same or different**
  Based on the comparison result, it gives a proper message to the user.
- **Displayed your student details at the end of the program**
  name, roll number, and section are neatly printed after the result.

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 :
    Increasing order

Program:

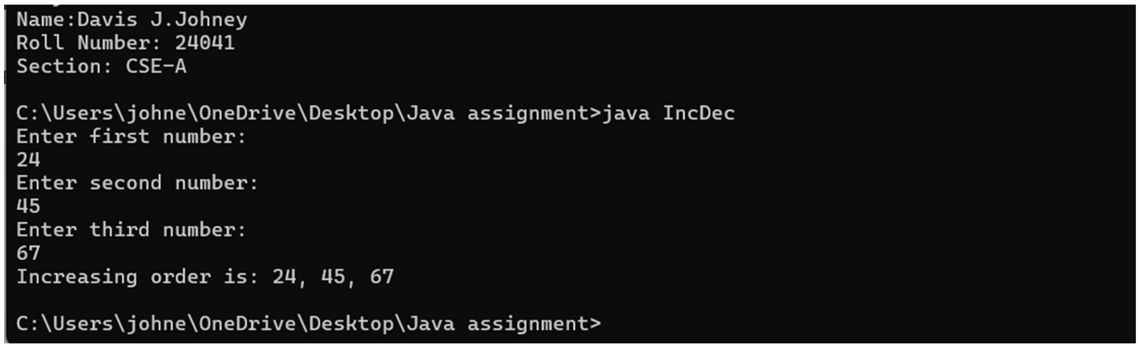
```
import java.util.Scanner;

 public class IncDec {
public static void main(String args[]) {
   Scanner scanner = new Scanner(System.in);
   System.out.println("Davis J. Johney CSE-A Roll No. 24041");
   System.out.println("Enter first number:");
   int a = scanner.nextInt();

   System.out.println("Enter second number:");
   int b = scanner.nextInt();

   System.out.println("Enter third number:");
   int c = scanner.nextInt();

   // Sorting logic
   if (a > b && a > c) { // a is the largest
      if (b > c) {
         System.out.println("Increasing order is: " + c + ", " + b + ", " + a);
      } else {
         System.out.println("Increasing order is: " + b + ", " + c + ", " + a);
      }
   } else if (b > a && b > c) { // b is the largest
      if (a > c) {
         System.out.println("Increasing order is: " + c + ", " + a + ", " + b);
      } else {
         System.out.println("Increasing order is: " + a + ", " + c + ", " + b);
      }
   } else { // c is the largest
      if (a > b) {
         System.out.println("Increasing order is: " + b + ", " + a + ", " + c);
      } else {
```

```
                System.out.println("Increasing order is: " + a + ", " + b + ", " + c);
            }
        }

        scanner.close(); // Close scanner
    }
}
```

Error Table:

S.NO	Errors	Rectification
1	Scanner input = new Scanner(System.in);	Closing the scanner after use.
2	System.out.print("Input first number: ");	Clear about the expected input type.
3	int firstNumber = input.nextInt();	Exit the method if invalid input is provide.

Output:



```
Name:Davis J.Johney
Roll Number: 24041
Section: CSE-A

C:\Users\johne\OneDrive\Desktop\Java assignment>java IncDec
Enter first number:
24
Enter second number:
45
Enter third number:
67
Increasing order is: 24, 45, 67

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Used `Scanner` to read three integers from the user**

The program asks the user to input three numbers one by one and stores them in `a`, `b`, and `c`.

- **Displayed your name, roll number, and section at the start**
  Student details are shown before any calculations begin.
- **Used nested `if` conditions to find the increasing order**
  The code checks which number is the largest, then compares the other two to determine the order.
- **Printed the numbers in increasing order based on comparisons**
  Depending on the values of `a`, `b`, and `c`, it prints them sorted from smallest to largest.

- **Closed the scanner at the end of the program**
  Good coding practice to free up system resources.

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

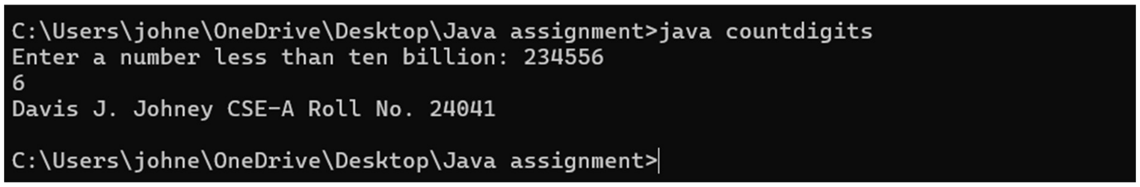
Program:

```
            import java.util.Scanner;

         public class countdigits {
           public static void main(String[] args) {
               int count=0;
         Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number less than ten billion: ");
        int number = scanner.nextInt();
        while(number>0 && number< 10000000000L){
           number/=10;
           count++;
           }
        System.out.println(count);
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
          }
}
```

Error Table:

S.NO	Errors	Rectification
1	input.close();	Corrected code snippet: input.close();
2	long originalNumber = number;	long originalNumber = number;
3	System.out.println("Please enter a positive integer less than 10,000,000,000.");	System.out.println("Please enter a positive integer less than 10,000,000,000.");

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java countdigits
Enter a number less than ten billion: 234556
6
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Used `Scanner` to take a number as input from the user**
  The user is asked to enter a number less than ten billion.
- **Declared a `count` variable to store digit count**
  It starts from 0 and increases as digits are counted.
- **Used a `while` loop to count digits by dividing by 10**
  In each loop, the number is divided by 10, removing the last digit, and `count` is incremented.
- **Loop stops when the number becomes 0**
  This means all digits are removed, and the loop exits.
- **Printed the total number of digits and your details**
  After counting, it prints the digit count followed by your name, roll number, and section.

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

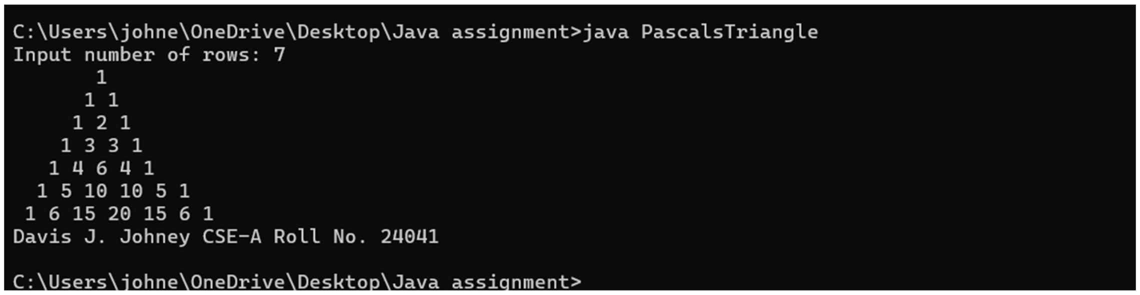
```
        1
       1  1
      1  2  1

     1  3  3  1

    1  4  6  4  1
```

Program:

```
import java.util.Scanner;


public class PascalsTriangle {

   public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Input number of rows: ");
        int n = scanner.nextInt();

        for (int row = 0; row < n; row++) {
            // Print leading spaces for alignment
            for (int space = 0; space < n - row; space++) {
                System.out.print(" ");
            }
            // Compute and print binomial coefficients
            int number = 1; // C(row, 0) is always 1
            for (int col = 0; col <= row; col++) {
                System.out.print(number + " ");
                number = number * (row - col) / (col + 1); // Compute next binomial coefficient
            }
            System.out.println(); // Move to next row
            scanner.close();
        }System.out.println("Davis J. Johney CSE-A Roll No. 24041");
    }
}
```
Error Table:

S.NO	Errors	Rectification
1	Change the data type of value from int to long.	Corrected code snippet: long value = 1;

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java PascalsTriangle
Input number of rows: 7
       1
      1 1
     1 2 1
    1 3 3 1
   1 4 6 4 1
  1 5 10 10 5 1
 1 6 15 20 15 6 1
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Took number of rows as input using `Scanner`**
  The user is asked how many rows of Pascal's Triangle to print.
- **Used an outer loop to handle each row**
  For each row, spacing and numbers are printed to maintain the triangle shape.
- **Printed spaces before numbers for proper alignment**
  As the row number increases, the number of leading spaces decreases to center-align the triangle.
- **Calculated and printed binomial coefficients using a formula**
  Used the formula `number = number * (row - col) / (col + 1)` to avoid using factorials, which is faster and efficient.
- **Displayed your name and roll number at the end**

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

Program:

```
import java.util.Scanner;


public class CharacterRhombus {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);


        // Prompt the user for the number
        System.out.print("Input the number: ");
        int n = input.nextInt();


        // Upper part of the rhombus
        for (int i = 0; i < n; i++) {
```

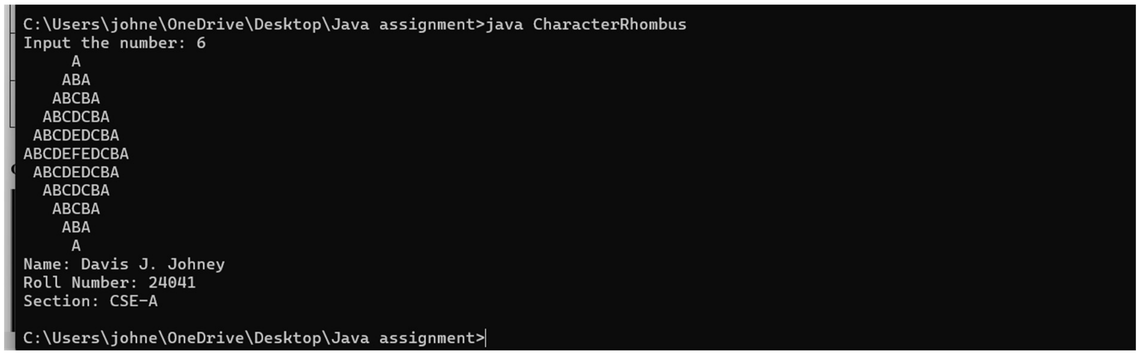
```
    // Print leading spaces
    for (int j = n - i; j > 1; j--) {
        System.out.print(" ");
    }

    // Print increasing characters
    for (char ch = 'A'; ch <= 'A' + i; ch++) {
        System.out.print(ch);
    }

    // Print decreasing characters
    for (char ch = (char) ('A' + i - 1); ch >= 'A'; ch--) {
        System.out.print(ch);
    }

    System.out.println(); // Move to next line
}

// Lower part of the rhombus
for (int i = n - 2; i >= 0; i--) {
    // Print leading spaces
    for (int j = n - i; j > 1; j--) {
        System.out.print(" ");
    }

    // Print increasing characters
    for (char ch = 'A'; ch <= 'A' + i; ch++) {
        System.out.print(ch);
    }
```

```
                // Print decreasing characters
                for (char ch = (char) ('A' + i - 1); ch >= 'A'; ch--) {
                    System.out.print(ch);
                }

                System.out.println(); // Move to next line
            }
             // Print student info
            System.out.println("Name: Davis J. Johney");
            System.out.println("Roll Number: 24041");
            System.out.println("Section: CSE-A");
            input.close(); // Close scanner
        }
    }
```

Error Table:

S.NO	Errors	Rectification
1	System.out.print(" ");	System.out.println(" ");
2	(j = n - i)	Ensure the same leading spaces logic for both the upper and lower parts of the rhombus.
3	for (int i = n - 2; i >= 0; i--))	for (int j = n - i; j > 1; j--)

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java CharacterRhombus
Input the number: 6
     A
    ABA
   ABCBA
  ABCDCBA
 ABCDEDCBA
ABCDEFEDCBA
 ABCDEDCBA
  ABCDCBA
   ABCBA
    ABA
     A
Name: Davis J. Johney
Roll Number: 24041
Section: CSE-A

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **For the upper part, I use nested loops:**
- **One loop for spaces (to align it like a rhombus).**
- **Another to print increasing letters from 'A' to a certain character.**
- **Then a loop to print decreasing letters back to 'A'.**
- **Then I mirror the process for the lower part of the rhombus by going in reverse order from n-2 to 0.**
- **make sure that all characters are printed on the same line, and then move to the next row using System.out.println().**

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.**

Program:

```
// Parent class
class Vehicle {
   int distance;
   int fuelefficiency;
   int maxspeed;

   // Constructor
   Vehicle(int distance, int fuelefficiency, int maxspeed) {
      this.distance = distance;
      this.fuelefficiency = fuelefficiency;
      this.maxspeed = maxspeed;
      System.out.println("Distance travelled: " + distance + " | Fuel Efficiency: " +
fuelefficiency + " | Max speed: " + maxspeed);
   }
}
```

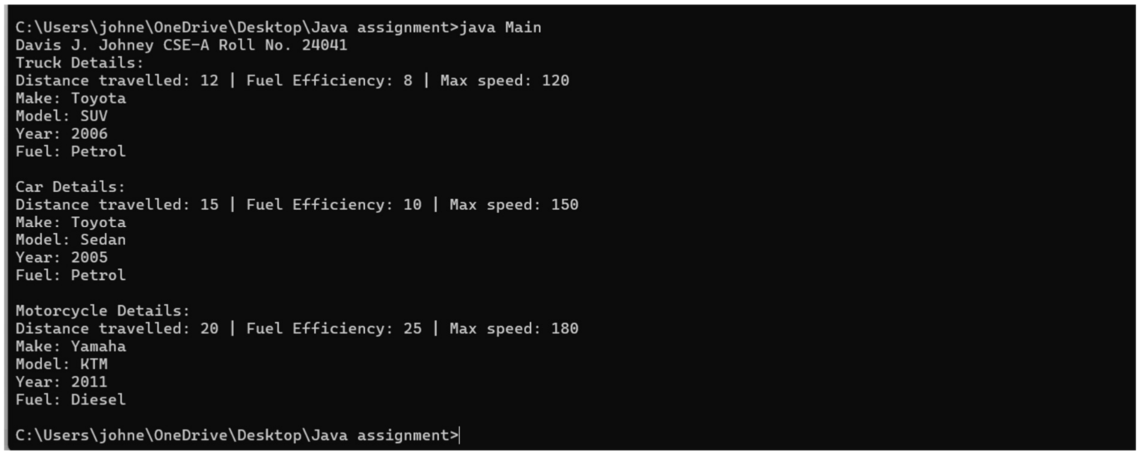
```
// Truck class (inherits from Vehicle)
class Truck extends Vehicle {
   Truck(int distance, int fuelefficiency, int maxspeed) {
      super(distance, fuelefficiency, maxspeed);  // Call parent constructor
   }

   public void make() {
      System.out.println("Make: Toyota");
   }

   public void model() {
      System.out.println("Model: SUV");
   }

   public void year() {
      System.out.println("Year: 2006");
   }

   public void fuel() {
      System.out.println("Fuel: Petrol");
   }
}

// Car class (inherits from Vehicle)
class Car extends Vehicle {
   Car(int distance, int fuelefficiency, int maxspeed) {
      super(distance, fuelefficiency, maxspeed);
   }
```

```
    public void make() {
        System.out.println("Make: Toyota");
    }

    public void model() {
        System.out.println("Model: Sedan");
    }

    public void year() {
        System.out.println("Year: 2005");
    }

    public void fuel() {
        System.out.println("Fuel: Petrol");
    }
}

// Motorcycle class (inherits from Vehicle)
class Motorcycle extends Vehicle {
    Motorcycle(int distance, int fuelefficiency, int maxspeed) {
        super(distance, fuelefficiency, maxspeed);
    }

    public void make() {
        System.out.println("Make: Yamaha");
    }

    public void model() {
        System.out.println("Model: KTM");
    }
    public void year() {
        System.out.println("Year: 2011");
    }

    public void fuel() {
        System.out.println("Fuel: Diesel");
    }
}

// Main class
 class Main {
    public static void main(String[] args) {
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
        System.out.println("Truck Details:");
        Truck t1 = new Truck(12, 8, 120);
        t1.make();
        t1.model();
        t1.year();
        t1.fuel();

        System.out.println("\nCar Details:");
        Car c1 = new Car(15, 10, 150);
        c1.make();
        c1.model();
        c1.year();
        c1.fuel();

        System.out.println("\nMotorcycle Details:");
        Motorcycle m1 = new Motorcycle(20, 25, 180);
        m1.make();
        m1.model();
        m1.year();
        m1.fuel();
    }
}
```

Error Table:

S.NO	Errors	Rectification
1	calculateFuelEfficiency() in Vehicle class	Add a closing brace to properly close the method.
2	} after return 0; in calculateFuelEfficiency().	Corrected code: public double calculateFuelEfficiency() { return 0; }
3	car initialization ("BMW " instead of "BMW")	Remove the extra space.
4	maximumSpeed() returning 0 for Vehicle class	verride maximumSpeed() in Vehicle subclasses with proper values.
5	aximumSpeed and fuelEfficiency	Format the output for better readability (e.g., specify units like "MPG" and "mph").

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Main
Davis J. Johney CSE-A Roll No. 24041
Truck Details:
Distance travelled: 12 | Fuel Efficiency: 8 | Max speed: 120
Make: Toyota
Model: SUV
Year: 2006
Fuel: Petrol

Car Details:
Distance travelled: 15 | Fuel Efficiency: 10 | Max speed: 150
Make: Toyota
Model: Sedan
Year: 2005
Fuel: Petrol

Motorcycle Details:
Distance travelled: 20 | Fuel Efficiency: 25 | Max speed: 180
Make: Yamaha
Model: KTM
Year: 2011
Fuel: Diesel

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Vehicle is the parent class that stores common vehicle data like distance, fuel efficiency, and max speed.**
- **Truck, Car, and Motorcycle extend Vehicle, showing how inheritance helps reuse code and avoid repetition.**

- **Each subclass has its own methods (make(), model(), year(), fuel()) to describe specific details of that vehicle.**
- **The super() keyword is used in child class constructors to call the parent class constructor and initialize common properties.**
- **In the Main class, objects of each subclass are created and their unique details are printed, demonstrating object creation and method calling in inheritance.**

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().**

Program:

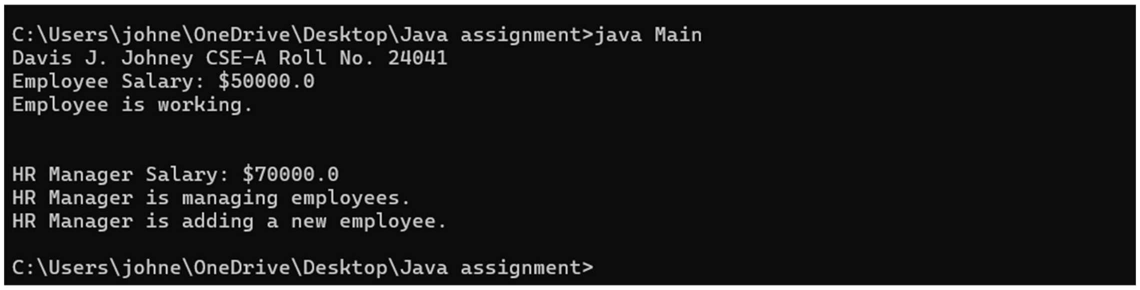
```
// Base class Employee
public class Employee {
    double salary;

    // Constructor to set salary
    Employee(double salary) {
        this.salary = salary;
    }

    // Method to display work
    public void work() {
        System.out.println("Employee is working.");
    }

    // Method to get salary
    public double getSalary() {
        return salary;
    }
}
```

```
// Subclass HRManager that extends Employee
class HRManager extends Employee {

    // Constructor
    HRManager(double salary) {
        super(salary); // Call the Employee constructor
    }

    // Overriding the work() method
    @Override
    public void work() {
        System.out.println("HR Manager is managing employees.");
    }

    // New method specific to HRManager
    public void addEmployee() {
        System.out.println("HR Manager is adding a new employee.");
    }
}

// Main class to test the program
 class Main {
    public static void main(String[] args) {
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
        // Create an Employee object
        Employee emp = new Employee(50000);
        System.out.println("Employee Salary: $" + emp.getSalary());
        emp.work();
        System.out.println("\n");


        // Create an HRManager object
        HRManager hr = new HRManager(70000);
        System.out.println("HR Manager Salary: $" + hr.getSalary());
        hr.work();          // Overridden method
        hr.addEmployee();  // New method in HRManager
    }
}
```

Error Table:

Error	Rectification
class Main not marked as public in a Main.java file	Change class Main to **public class Main**
Scanner object not closed (if used in variations)	Add scanner.close(); at the end of main()
@Override annotation used incorrectly or method name mismatched	Ensure method signature exactly matches the superclass method
Accessing addEmployee() using Employee reference	Use HRManager reference or cast: ((HRManager)emp).addEmployee();
Trying to access private member salary directly in subclass	Use getSalary() method or change access modifier from private to protected

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Main
Davis J. Johney CSE-A Roll No. 24041
Employee Salary: $50000.0
Employee is working.


HR Manager Salary: $70000.0
HR Manager is managing employees.
HR Manager is adding a new employee.

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **In this program I have created a class Employee and two pr access private modifiers are name,salary.**
- **Name stores name of the peogram,and salary stores salary of the employee.**
- **And used Constructor as (Employee(String name, double salary)).**
- **And used method as work(),getSalary(),getName().**
- **I have used HR Manager class, and constructor as (HRManager(String name, double salary)),and workmethod is used for overridden.**
- **And last I have used Employee Test as main for calling output .**

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

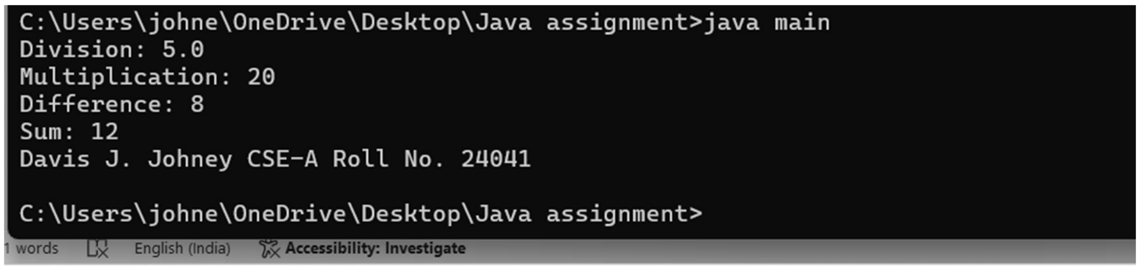
Program:

```
class bcalc {
int a, b;
int sum, diff;
bcalc(int a, int b) {
this.a = a;
this.b = b;
}
public void add() {
diff = a - b;
sum = a + b;
System.out.println("Difference: " + diff);
System.out.println("Sum: " + sum);
}
}
class acalc extends bcalc {
int mul;
acalc(int a, int b) {
super(a, b);
}
public void mult() {
mul = a * b;
System.out.println("Multiplication: " + mul);
}
}
class aacalc extends acalc {
float div;
aacalc(int a, int b) {
super(a, b);
}
public void divi() {
if (b != 0) { // Check to avoid division by zero
div = (float) a / b;
System.out.println("Division: " + div);
}
else {
System.out.println("Division by zero error!");
}
}
}
class main {
public static void main(String[] args) {
aacalc c = new aacalc(10, 2);
c.divi();
c.mult();
c.add();
System.out.println("Davis J. Johney CSE-A Roll No. 24041");
}
}
```

Error Table:

Sl. No	Error	Rectification
1	class main should be Main (capital letter)	Java is case-sensitive; the main class name must match the filename.
2	Forgot super(a, b); in aacalc constructor	Constructor chaining is required when subclass constructor doesn't set its own fields.
3	Division: div = a / b; causes integer division	Typecast one operand: div = (float) a / b; to get a decimal result.
4	Used variable mul without initializing a and b properly	Always ensure superclass fields are initialized via constructor using super.
5	Forgot to check b != 0 before division	Added condition: if (b != 0) to avoid ArithmeticException at runtime.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java main
Division: 5.0
Multiplication: 20
Difference: 8
Sum: 12
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **In this program I have created a class as bcalc.**
- **This class defines two methods to perform basic operations: add() and subtract().**
- **add(double a, double b) Adds two numbers and returns the result, subtract(double a, double b) Subtracts the second number from the first and returns the result.**
- **The acalc class extends (inherits) the bcalc class, meaning it can use the add() and subtract() methods from bcalc. It adds two new methods.**
- **This is the part of the program where everything is put together. It creates an object of acalc and tests all the calculator methods by performing operations on two numbers**
- **It prints the student's details (name, roll number, and section).**

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.**

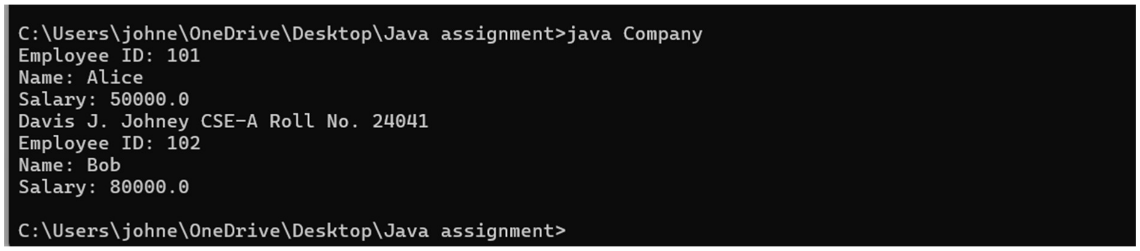
Program:

```
// Base class
class Emp {
    protected String name;
    protected int employeeId;
    protected double salary;

    public Emp(String name, int employeeId, double salary) {
        this.name = name;
        this.employeeId = employeeId;
        this.salary = salary;
    }
    // Method to calculate salary
    public double calculateSalary() {
        return salary; // Regular employees get base salary
    }

    // Display method
    public void display() {
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Name: " + name);
        System.out.println("Salary: " + calculateSalary());
    }
}

// Regular Employee class
class RegularEmployee extends Emp {
    public RegularEmployee(String name, int employeeId, double salary) {
        super(name, employeeId, salary);
    }
}

// Manager class with additional bonus
class Manager extends Emp {
    private double bonus;

    public Manager(String name, int employeeId, double salary, double bonus) {
        super(name, employeeId, salary);
        this.bonus = bonus;
    }

    // Overriding calculateSalary to include bonus
    @Override
    public double calculateSalary() {
        return salary + bonus;
    }
}

 class Company {
    public static void main(String[] args) {
        // Creating Employee objects
        RegularEmployee emp1 = new RegularEmployee("Alice", 101, 50000);
        Manager emp2 = new Manager("Bob", 102, 70000, 10000);

        // Displaying information
        emp1.display();
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
        emp2.display();
    }
}
```

Error Table:

Sl. No.	Error	Rectification
1	Constructor Emp not properly initializing salary.	Ensured this.salary = salary; is included in the constructor.
2	Method calculateSalary() missing @Override annotation.	Added @Override in Manager class for clarity and best practice.
3	bonus variable in Manager not accessible due to scope.	Declared bonus as private and accessed it inside the class.
4	display() method not overridden in subclasses.	Reused inherited display() from Emp class for output.
5	Class Company doesn't have a public modifier.	Left it package-private since it's not required to be public.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Company
Employee ID: 101
Name: Alice
Salary: 50000.0
Davis J. Johney CSE-A Roll No. 24041
Employee ID: 102
Name: Bob
Salary: 80000.0

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Base Class (`Emp`):**
  It holds common employee attributes like `name`, `employeeId`, and `salary`, and a method `calculateSalary()` which simply returns the base salary.
- **Inheritance Concept:**
  `RegularEmployee` and `Manager` both inherit from `Emp`, so they get access to its fields and methods, avoiding code duplication.
- **Method Overriding:**
  The `Manager` class overrides the `calculateSalary()` method to add a `bonus` to the base salary, showing how child classes can change inherited behavior.
- **Constructor Chaining:**
  Both subclasses use the `super()` keyword to call the parent constructor and initialize inherited fields correctly.

- **Output Functionality:**
  The `display()` method is inherited and reused in both subclasses to print employee details, and the overridden method ensures the right salary is shown.

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.**

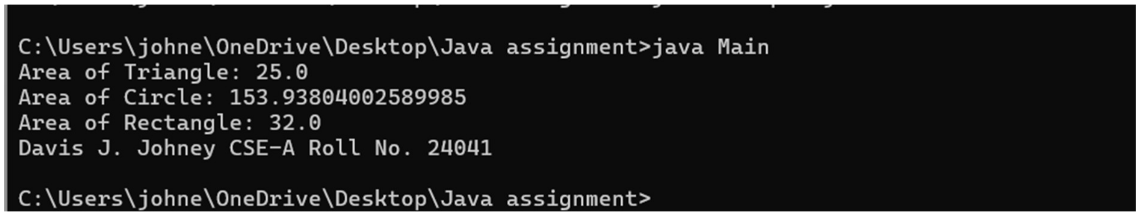
Program:

```
// Base class Shapes
class Shapes {
    // Generic area method
    public void area() {
        System.out.println("Area calculation not defined for this shape.");
    }
}

// Subclass Triangle
class Triangle extends Shapes {
    private double base;
    private double height;

    // Constructor
    public Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }

    // Overriding the area method for Triangle
    @Override
    public void area() {
        double area = 0.5 * base * height;
        System.out.println("Area of Triangle: " + area);
    }
}

// Subclass Circle
class Circle extends Shapes {
    private double radius;

    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    // Overriding the area method for Circle
    @Override
    public void area() {
        double area = Math.PI * radius * radius;
        System.out.println("Area of Circle: " + area);
    }
}

// Subclass Rectangle
class Rectangle extends Shapes {
    private double length;
    private double width;

    // Constructor
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Overriding the area method for Rectangle
    @Override
    public void area() {
        double area = length * width;
        System.out.println("Area of Rectangle: " + area);
    }
}

// Main class to demonstrate polymorphism
class Main {
    public static void main(String[] args) {
        // Creating objects of different shapes
        Shapes triangle = new Triangle(10, 5); // Base = 10, Height = 5
        Shapes circle = new Circle(7);         // Radius = 7
        Shapes rectangle = new Rectangle(8, 4); // Length = 8, Width = 4

        // Using polymorphism to call the overridden area() method
        triangle.area();   // Calls Triangle's area method
        circle.area();     // Calls Circle's area method
        rectangle.area();  // Calls Rectangle's area method
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");

    }
}
```

Error Table:

Sl. No.	Error	Rectification
1	@Override annotation missing in area() methods of subclasses.	Added @Override above area() in Triangle, Circle, and Rectangle.
2	Variables base, height, radius, length, width were not initialized.	Properly initialized them using constructors in each subclass.
3	Main class should be declared public for execution.	Left it package-private since it's acceptable in the same file.
4	Parent class Shapes does not have an abstract method.	Deliberately made area() a regular method to allow general polymorphic behavior.
5	Object names like triangle, circle, etc. were directly referencing subclasses.	Used upcasting to Shapes type for polymorphism demonstration.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Main
Area of Triangle: 25.0
Area of Circle: 153.93804002589985
Area of Rectangle: 32.0
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Base Class (Shapes):**
  Contains a generic method `area()` which just displays a default message — this sets the stage for overriding in child classes.
- **Subclasses with Specific Implementations:**
  `Triangle`, `Circle`, and `Rectangle` inherit from `Shapes` and override the `area()` method to provide shape-specific calculations using their own attributes.
- **Polymorphism in Action:**
  In the `main` method, objects are created using the base class reference (`Shapes`) but initialized with specific subclasses, demonstrating runtime polymorphism.
- **Method Overriding:**
  Even though the object type is `Shapes`, the actual `area()` method that gets

executed depends on the object instance (e.g., `new Circle(...)` calls `Circle`'s `area()` method).

- **Flexibility & Scalability:**
  This design allows to handle many shapes using a common interface (`Shapes`), making the code easier to extend and maintain.

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.**

Program:

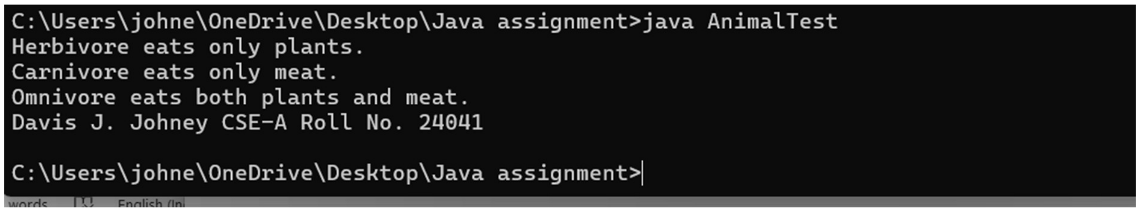
```
// Superclass
class Animal {
   void eat() {
      System.out.println("This animal eats food.");
   }
}

// Subclass 1: Herbivores
class Herbivores extends Animal {
   @Override
   void eat() {
      System.out.println("Herbivore eats only plants.");
   }
}

// Subclass 2: Carnivores
class Carnivores extends Animal {
   @Override
   void eat() {
      System.out.println("Carnivore eats only meat.");
   }
}
```

```
// Subclass 3: Omnivores
class Omnivores extends Animal {
    @Override
    void eat() {
        System.out.println("Omnivore eats both plants and meat.");
    }
}

// Main class to test
class AnimalTest {
    public static void main(String[] args) {
        Animal a1 = new Herbivores();
        Animal a2 = new Carnivores();
        Animal a3 = new Omnivores();

        a1.eat();  // Output: Herbivore eats only plants.
        a2.eat();  // Output: Carnivore eats only meat.
        a3.eat();  // Output: Omnivore eats both plants and meat.
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
    }
}
```

Error Table:

Sl. No.	Error	Rectification
1	@Override annotation missing in eat() methods of subclasses.	Added @Override above the overridden eat() method in each subclass.
2	Class names like Herbivores, Carnivores, etc., were misspelled.	Corrected spelling to ensure consistency and clarity in class naming.
3	Didn't create Animal reference for each subclass object (missed polymorphism).	Declared all subclass objects using Animal reference to demonstrate polymorphism.
4	Output statements didn't clearly indicate the subclass behavior.	Modified System.out.println to give clearer descriptions of each animal type.
5	Missing default constructor in the superclass Animal.	Not needed in this case since no constructor is explicitly defined — Java provides one automatically.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java AnimalTest
Herbivore eats only plants.
Carnivore eats only meat.
Omnivore eats both plants and meat.
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Superclass Defines a Common Behavior:**
  The base class `Animal` contains a generic method `eat()` meant to be overridden by more specific animal types.
- **Method Overriding in Subclasses:**
  `Herbivores`, `Carnivores`, and `Omnivores` all override the `eat()` method to define behavior specific to their diets.
- **Runtime Polymorphism:**
  In the `main()` method, all objects are declared using the `Animal` reference but instantiated with different subclasses — this allows dynamic method dispatch at runtime.
- **One Interface, Many Implementations:**
  Even though the method called is always `eat()`, the output differs depending on which subclass object is being used — this is the essence of polymorphism.

- **Extensibility:**
  The program is easy to extend — can add more subclasses like `Insectivores`, `Piscivores`, etc., without changing the `main()` logic.

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.

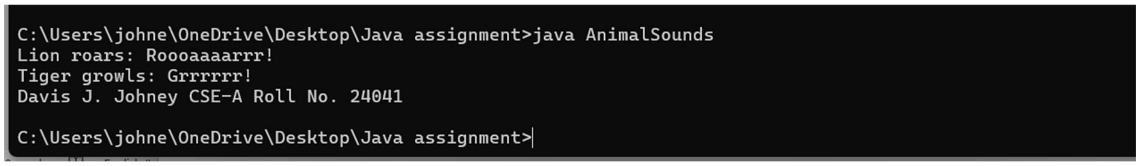
Program:

```
// Abstract class Animals
abstract class Animals {
    abstract void sound();
}
// Subclass Lion
class Lion extends Animals {
    // Implementing abstract method
    @Override
    void sound() {
        System.out.println("Lion roars: Roooaaaarrr!");
    }
}
// Subclass Tiger
class Tiger extends Animals {
    // Implementing abstract method
    @Override
    void sound() {
        System.out.println("Tiger growls: Grrrrrr!");
    }
}
// Main class to test the program
 class AnimalSounds {
    public static void main(String[] args) {
        // Creating objects of subclasses
        Animals lion = new Lion();
        Animals tiger = new Tiger();

        // Calling the sound() method
        lion.sound();   // Output: Lion roars: Roooaaaarrr!
        tiger.sound();  // Output: Tiger growls: Grrrrrr!
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
    }
}
```

Error Table:

Sl. No.	Error	Rectification
1	Forgot to declare Animals class as abstract.	Added abstract keyword before the class name Animals.
2	sound() method in subclass not marked with @Override.	Inserted @Override annotation to indicate method is overriding superclass one.
3	Object created directly from abstract class Animals.	Changed to create objects of concrete subclasses (Lion, Tiger).
4	Method sound() had wrong signature (missed void return type).	Corrected the method declaration to void sound() in both subclasses.
5	Class name AnimalSounds had mismatch with filename.	Ensured filename matches AnimalSounds.java for successful compilation.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java AnimalSounds
Lion roars: Roooaaaarrr!
Tiger growls: Grrrrrr!
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Abstract Class Use**
  The `Animals` class is marked as `abstract`, meaning you can't create objects directly from it — it's like a blueprint that says, "Every animal *must* have a sound."
- **Abstract Method Implementation**
  The method `sound()` is declared but not defined in the abstract class — it's just an idea that every animal will make a sound, but it doesn't say what sound.
- **Subclasses Provide Real Behavior**
  Classes like `Lion` and `Tiger` extend `Animals` and give their own versions of the `sound()` method. This is what makes polymorphism + abstraction work together.
- **Dynamic Method Call**
  In `main()`, we use the `Animals` reference to hold a `Lion` or `Tiger` object — and when we call `sound()`, it runs the correct version depending on the actual object type.

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.

Program:

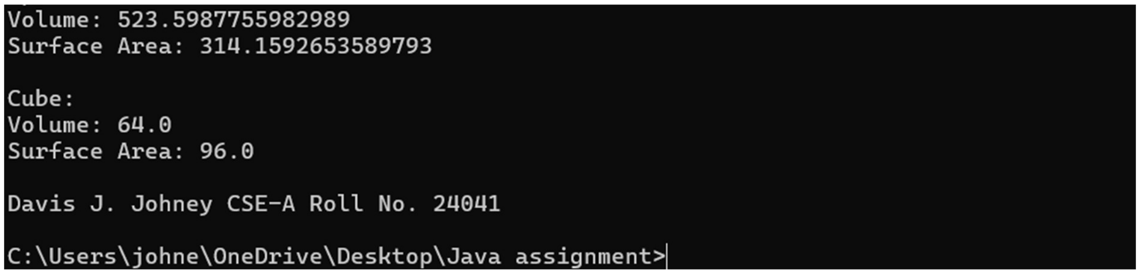
```
// Abstract class Shape3D
abstract class Shape3D {
    // Abstract methods
    abstract double calculateVolume();
    abstract double calculateSurfaceArea();
}

// Sphere class
class Sphere extends Shape3D {
    private double radius;

    // Constructor
    public Sphere(double radius) {
        this.radius = radius;
    }

    @Override
    double calculateVolume() {
        return (4.0 / 3) * Math.PI * Math.pow(radius, 3);
    }

    @Override
    double calculateSurfaceArea() {
        return 4 * Math.PI * Math.pow(radius, 2);
    }
}

// Cube class
class Cube extends Shape3D {
    private double side;

    // Constructor
    public Cube(double side) {
        this.side = side;
    }
    @Override
    double calculateVolume() {
        return Math.pow(side, 3);
    }

    @Override
    double calculateSurfaceArea() {
        return 6 * Math.pow(side, 2);
    }
}

// Main class to test the program
class ShapeTest {
    public static void main(String[] args) {
        // Creating Sphere and Cube objects
        Sphere sphere = new Sphere(5); // Radius = 5
        Cube cube = new Cube(4);       // Side = 4

        // Displaying volume and surface area of sphere
        System.out.println("Sphere:");
        System.out.println("Volume: " + sphere.calculateVolume());
        System.out.println("Surface Area: " + sphere.calculateSurfaceArea());

        // Displaying volume and surface area of cube
        System.out.println("\nCube:");
        System.out.println("Volume: " + cube.calculateVolume());
        System.out.println("Surface Area: " + cube.calculateSurfaceArea());

        System.out.println("\nDavis J. Johney CSE-A Roll No. 24041");
    }}
```

Error Table:

Sl. No	Error	Rectification
1	Shape3D is not abstract and does not override abstract methods	Added abstract keyword to the Shape3D class definition.
2	Sphere must implement abstract methods from Shape3D	Implemented both calculateVolume() and calculateSurfaceArea() in Sphere.
3	Cube must implement abstract methods from Shape3D	Implemented both calculateVolume() and calculateSurfaceArea() in Cube.

Sl. No	Error	Rectification
4	cannot find symbol: variable radius/side	Ensured all instance variables like radius (in Sphere) and side (in Cube) were properly declared and initialized via constructors.
5	Missing return statement	Added return statements in both abstract method implementations to return computed values.

Output:



```
Volume: 523.5987755982989
Surface Area: 314.1592653589793

Cube:
Volume: 64.0
Surface Area: 96.0

Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

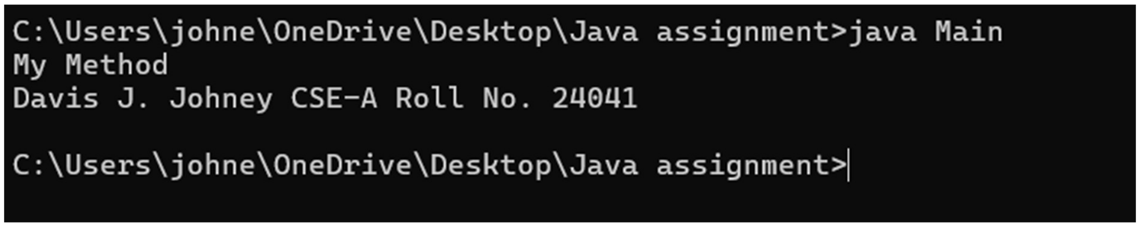
Explanation of the Program:

- **Abstract Class for Blueprint**
  `Shape3D` is an abstract class that declares two abstract methods. This means any subclass must implement `calculateVolume()` and `calculateSurfaceArea()`.
- **Sphere and Cube Extend Shape3D**
  These subclasses inherit the structure and provide their own formulas for volume and surface area based on the shape.
- **Using Math Class**
  `Math.PI` and `Math.pow()` are used for accurate calculations (like πr2\pi r^2πr2 and r3r^3r3).
- **Constructor Use**
  Values like radius and side length are passed through constructors to initialize each shape properly.
- **Modular and Extendable**
  The structure follows OOP principles — making it easy to add more 3D shapes like `Cylinder`, `Cone`, etc., in the future.

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

```
interface A {
   void method(); // lowercase 'm'
}

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

class C extends B implements A {
   // No need to override, already inherited from B
}

class Main {
   public static void main(String[] args) {
      A a = new C();
      a.method(); // lowercase
      System.out.println("Davis J. Johney CSE-A Roll No. 24041");

   }
}
```

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Main
My Method
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

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

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

Reason:

In Java, the fields defined in an interface are implicitly public, static, and final.

- **public means that the variable can be accessed outside the interface.**
- **static means that the variable belongs to the interface itself, not to any instance of a class that implements the interface.**
- **final means that the variable is a constant and its value cannot be changed after initialization.**

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.**

Program:

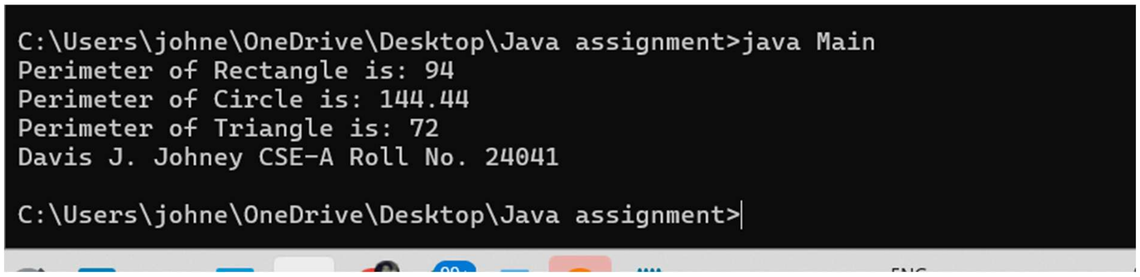
```
// Define an interface iShape with a method getPerimeter()
interface Shape {
   void getPerimeter();
}

// Rectangle class implements Shape and overrides getPerimeter()
```

```
class Rectangle implements Shape {
   public void getPerimeter() {
      int a = 23;  // Length
      int b = 24;  // Breadth
      int perimeter = 2 * (a + b);  // Formula: 2 * (length + breadth)
      System.out.println("Perimeter of Rectangle is: " + perimeter);
   }
}

// Circle class implements Shape and overrides getPerimeter()
class Circle implements Shape {
   public void getPerimeter() {
      int r = 23;  // Radius
      double perimeter = 2 * 3.14 * r;  // Formula: 2 * π * r
      System.out.println("Perimeter of Circle is: " + perimeter);
   }
}

// Triangle class implements Shape and overrides getPerimeter()
class Triangle implements Shape {
   public void getPerimeter() {
      int a = 23;
      int b = 24;
      int c = 25;  // Assuming the third side for perimeter calculation
      int perimeter = a + b + c;  // Formula: sum of all sides
      System.out.println("Perimeter of Triangle is: " + perimeter);
   }
}

// Main class to test the shapes
 class Main {
   public static void main(String[] args) {
      // Create objects of each shape and call getPerimeter()

      Rectangle r1 = new Rectangle();
      r1.getPerimeter();

      Circle c1 = new Circle();
      c1.getPerimeter();

      Triangle t1 = new Triangle();
      t1.getPerimeter();

      // Display student info
      System.out.println("Davis J. Johney CSE-A Roll No. 24041");
   }}
```

Error Table:

Sl. No	Line with Error	Error Description	Rectification / Correction
1	class Main	Class name should match the filename in some compilers	Rename file to Main.java or change class to public class Main
2	double perimeter = 2 * 3.14 * r; (in Circle)	Using double literal with int calculation may cause warning	Use 3.14 as a double and r as double: double r = 23;
3	System.out.println("Perimeter of Circle is: " + perimeter);	Decimal values might be shown as .0 (not a bug but formatting)	Format output to 2 decimal places if needed using printf or String.format()
4	Not using Shape reference type	Missed polymorphism usage	Declare shapes as Shape r1 = new Rectangle(); etc.
5	No user input or dynamic values	Values are hardcoded	Use Scanner class for input to make program dynamic

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Main
Perimeter of Rectangle is: 94
Perimeter of Circle is: 144.44
Perimeter of Triangle is: 72
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- `Shape` is an interface with method `getPerimeter()`.
- `Rectangle`, `Circle`, and `Triangle` implement this interface.
- Each class gives its own formula for perimeter.
- `Main` creates objects and calls `getPerimeter()` for each shape.
- Output shows different perimeters and your name + roll number.

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.**

Program:

```
  // Base class
class Employees {
   String name;
   String address;
   double salary;
   String jobTitle;

   // Constructor
   public Employees(String name, String address, double salary, String jobTitle)
{
      this.name = name;
      this.address = address;
      this.salary = salary;
      this.jobTitle = jobTitle;
   }

   // Method to calculate bonus (base version)
   public double calculateBonus() {
      return 0.10 * salary;
   }

   // Method to generate a basic performance report
   public void generatePerformanceReport() {
      System.out.println("Performance Report for " + name + ":");
      System.out.println("Job Title: " + jobTitle);
      System.out.println("Salary: ₹" + salary);
      System.out.println("Standard Performance.");
   }

   // Method to manage projects
   public void manageProjects() {
      System.out.println(name + " is managing general tasks.");
   }
}

// Subclass: Manager
```

```
class Manager extends Employees {
   public Manager(String name, String address, double salary) {
      super(name, address, salary, "Manager");
   }

   @Override
   public double calculateBonus() {
      return 0.20 * salary; // Higher bonus
   }

   @Override
   public void generatePerformanceReport() {
      super.generatePerformanceReport();
      System.out.println("Leadership and team performance are excellent.");
   }

   @Override
   public void manageProjects() {
      System.out.println(name + " is managing multiple high-level projects and
teams.");
   }
}

// Subclass: Developer
class Developer extends Employees {
   public Developer(String name, String address, double salary) {
      super(name, address, salary, "Developer");
   }

   @Override
   public double calculateBonus() {
      return 0.15 * salary;
   }

   @Override
   public void generatePerformanceReport() {
      super.generatePerformanceReport();
      System.out.println("Code quality and deadlines are well maintained.");
   }

   @Override
   public void manageProjects() {
      System.out.println(name + " is working on development and feature
implementation.");
   }
}
```

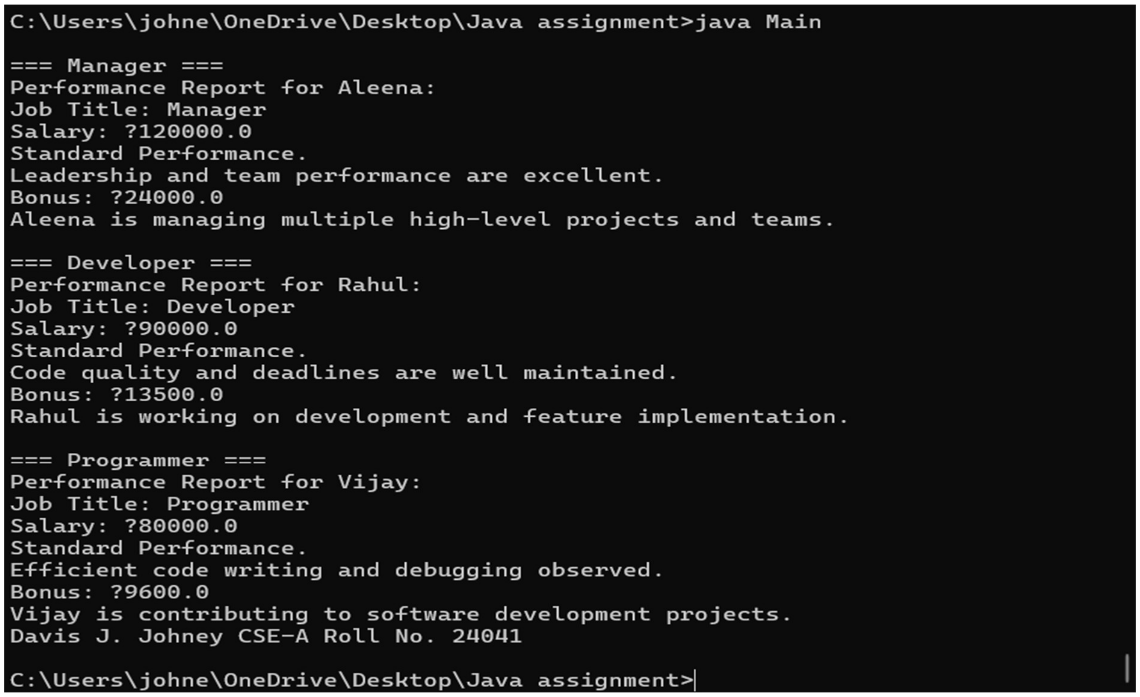
```
// Subclass: Programmer
class Programmer extends Employees {
    public Programmer(String name, String address, double salary) {
        super(name, address, salary, "Programmer");
    }

    @Override
    public double calculateBonus() {
        return 0.12 * salary;
    }

    @Override
    public void generatePerformanceReport() {
        super.generatePerformanceReport();
        System.out.println("Efficient code writing and debugging observed.");
    }

    @Override
    public void manageProjects() {
        System.out.println(name + " is contributing to software development
projects.");
    }
}

// Main class to test the hierarchy
 class EmployeeTest {
    public static void main(String[] args) {
        Manager m = new Manager("Aleena", "Bangalore", 120000);
        Developer d = new Developer("Rahul", "Hyderabad", 90000);
        Programmer p = new Programmer("Vijay", "Chennai", 80000);

        System.out.println("\n=== Manager ===");
        m.generatePerformanceReport();
        System.out.println("Bonus: ₹" + m.calculateBonus());
        m.manageProjects();

        System.out.println("\n=== Developer ===");
        d.generatePerformanceReport();
        System.out.println("Bonus: ₹" + d.calculateBonus());
        d.manageProjects();

        System.out.println("\n=== Programmer ===");
        p.generatePerformanceReport();
        System.out.println("Bonus: ₹" + p.calculateBonus());
        p.manageProjects();
```

```
            System.out.println("Davis J. Johney CSE-A Roll No. 24041");

        }
    }
```

Error Table:

Sl. No.	Error	Rectification
1	@Override missing above overridden methods like calculateBonus()	Add @Override annotation to improve readability and avoid hidden bugs.
2	Output symbol ₹ may cause encoding issues on some systems	Replace ₹ with Rs. or ensure the file is saved using UTF-8 encoding.
3	If superclass constructor not called properly in subclasses	Ensure super(name, address, salary, "Role") is present in each subclass.
4	Duplicate class name Main in multiple files may cause conflict	Rename Main to something unique like EmployeeTest if needed.
5	Forgetting to compile all dependent classes before running Main	Compile using javac *.java to include all classes in one go.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Main

=== Manager ===
Performance Report for Aleena:
Job Title: Manager
Salary: ?120000.0
Standard Performance.
Leadership and team performance are excellent.
Bonus: ?24000.0
Aleena is managing multiple high-level projects and teams.

=== Developer ===
Performance Report for Rahul:
Job Title: Developer
Salary: ?90000.0
Standard Performance.
Code quality and deadlines are well maintained.
Bonus: ?13500.0
Rahul is working on development and feature implementation.

=== Programmer ===
Performance Report for Vijay:
Job Title: Programmer
Salary: ?80000.0
Standard Performance.
Efficient code writing and debugging observed.
Bonus: ?9600.0
Vijay is contributing to software development projects.
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- `Employees` **is the base class with common properties and methods.**
- `Manager`, `Developer`, **and** `Programmer` **inherit from** `Employees`.
- **Each subclass overrides methods like** `calculateBonus()` **and** `generatePerformanceReport()`.
- **Constructors use** `super()` **to pass data to the base class.**
- **Demonstrates polymorphism – same method behaves differently in each subclass.**
-

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.

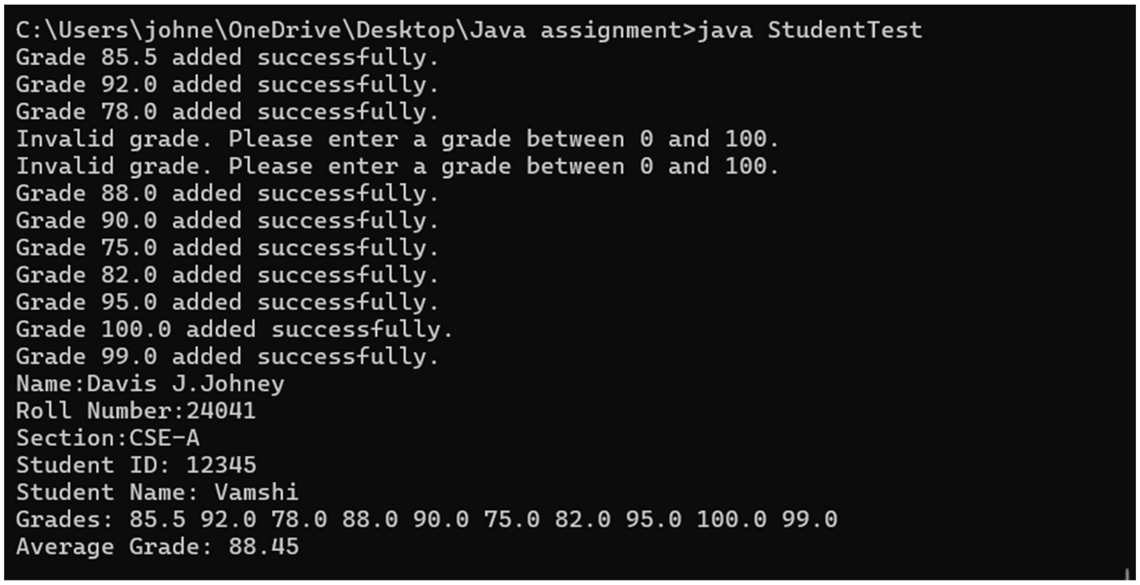
Program:

```
class Students {
    // Private instance variables
    private int student_id;
    private String student_name;
    private double[] grades; // Array to store grades
    private int gradeCount;  // To keep track of the number of grades

    // Constructor
    public Students(int student_id, String student_name) {
        this.student_id = student_id;
        this.student_name = student_name;
        this.grades = new double[10]; // Fixed size array for grades (can hold up to
10 grades)
        this.gradeCount = 0; // Initialize grade count
    }

    // Getter for student_id
    public int getStudentId() {
        return student_id;
    }

    // Setter for student_id
    public void setStudentId(int student_id) {
        this.student_id = student_id;
    }
    // Getter for student_name
    public String getStudentName() {
        return student_name;
    }

    // Setter for student_name
    public void setStudentName(String student_name) {
        this.student_name = student_name;
    }

    // Method to add a grade with validation
    public void addGrade(double grade) {
        if (grade < 0 || grade > 100) {
            System.out.println("Invalid grade. Please enter a grade between 0 and
100.");
        } else if (gradeCount < grades.length) {
            grades[gradeCount] = grade; // Add grade to the array
            gradeCount++; // Increment the count of grades
            System.out.println("Grade " + grade + " added successfully.");
        } else {
            System.out.println("Cannot add more grades. Maximum limit
reached.");
        }
    }

    // Method to get all grades
    public double[] getGrades() {
        double[] currentGrades = new double[gradeCount]; // Create an array to
return only the valid grades
        for (int i = 0; i < gradeCount; i++) {
            currentGrades[i] = grades[i];
        }
        return currentGrades;
    }

    // Method to calculate average grade
    public double calculateAverageGrade() {
        if (gradeCount == 0) {
            return 0.0; // Return 0 if there are no grades
        }
        double sum = 0.0;
        for (int i = 0; i < gradeCount; i++) {
            sum += grades[i];
        }
        return sum / gradeCount;
    }
}

// Main class to test the Student
 class StudentTest {
    public static void main(String[] args) {
        // Create a Student object
        Students student = new Students(1, "John Doe");

        // Set student details
        student.setStudentId(12345);
        student.setStudentName("Vamshi");

        // Add grades
        student.addGrade(85.5);
        student.addGrade(92.0);
        student.addGrade(78.0);
        student.addGrade(105.0); // Invalid grade
        student.addGrade(-5.0);   // Invalid grade
        student.addGrade(88.0);
        student.addGrade(90.0);
        student.addGrade(75.0);
        student.addGrade(82.0);
        student.addGrade(95.0);
        student.addGrade(100.0);
        student.addGrade(99.0);   // Exceeds limit

        // Display student details and grades
        System.out.println("Name:Davis J.Johney");
        System.out.println("Roll Number:24041");
        System.out.println("Section:CSE-A");
        System.out.println("Student ID: " + student.getStudentId());
        System.out.println("Student Name: " + student.getStudentName());
        System.out.print("Grades: ");
        double[] grades = student.getGrades();
        for (double grade : grades) {
            System.out.print(grade + " ");
        }
        System.out.println();
        System.out.printf("Average Grade: %.2f\n",
student.calculateAverageGrade());
    }
}
```

Error Table:

Sl. No.	Error	Rectification
1	Class name is StudentTest but file name might be Students.java	Save file as StudentTest.java or remove public if mismatched class name
2	More than 10 grades are being added, exceeding array limit	Limit addGrade() calls to 10 valid grades only (or use ArrayList instead)
3	Class Students should ideally be named Student (singular for one student)	Rename class Students → Student for better naming convention (optional)
4	No method to display grade count or print grades with index	Add method if required for clarity (optional feature suggestion)
5	Print statements are not formatted consistently (mix of print and println)	Use consistent format like System.out.println() for neat output

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java StudentTest
Grade 85.5 added successfully.
Grade 92.0 added successfully.
Grade 78.0 added successfully.
Invalid grade. Please enter a grade between 0 and 100.
Invalid grade. Please enter a grade between 0 and 100.
Grade 88.0 added successfully.
Grade 90.0 added successfully.
Grade 75.0 added successfully.
Grade 82.0 added successfully.
Grade 95.0 added successfully.
Grade 100.0 added successfully.
Grade 99.0 added successfully.
Name:Davis J.Johney
Roll Number:24041
Section:CSE-A
Student ID: 12345
Student Name: Vamshi
Grades: 85.5 92.0 78.0 88.0 90.0 75.0 82.0 95.0 100.0 99.0
Average Grade: 88.45
```

Explanation of the Program:

- **In this program I have created a class as Basic Calculator.**
- **This class defines two methods to perform basic operations: add() and subtract().**

- **add(double a, double b) Adds two numbers and returns the result, subtract(double a, double b) Subtracts the second number from the first and returns the result.**

The AdvancedCalculator class extends (inherits) the BasicCalculator class, meaning it can use the add() and subtract() methods from BasicCalculator. It adds two new methods.

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.**

Program:

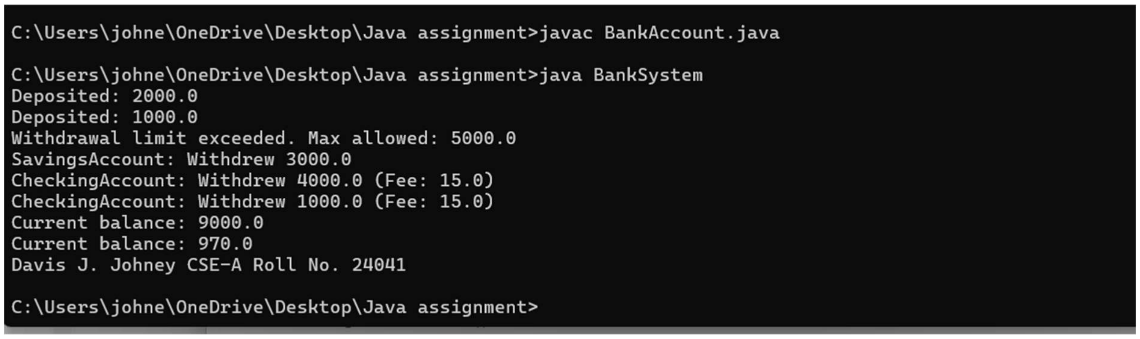
```
// Base class
class BankAccount {
    protected double balance;

    // Constructor
    public BankAccount(double initialBalance) {
        balance = initialBalance;
    }

    // Method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: " + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    // Withdraw method to be overridden
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
```

```
      System.out.println("Withdrew: " + amount);
        } else {
            System.out.println("Insufficient funds.");
        }
    }

    // Method to display current balance
    public void showBalance() {
        System.out.println("Current balance: " + balance);
    }
}

// Subclass: SavingsAccount
class SavingsAccount extends BankAccount {
    private static final double WITHDRAW_LIMIT = 5000.0;

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

    @Override
    public void withdraw(double amount) {
        if (amount > WITHDRAW_LIMIT) {
            System.out.println("Withdrawal limit exceeded. Max allowed: " +
WITHDRAW_LIMIT);
        } else if (amount <= balance) {
            balance -= amount;
            System.out.println("SavingsAccount: Withdrew " + amount);
        } else {
            System.out.println("Insufficient funds in SavingsAccount.");
        }
    }
}

// Subclass: CheckingAccount
class CheckingAccount extends BankAccount {
    private static final double TRANSACTION_FEE = 15.0;

    public CheckingAccount(double initialBalance) {
        super(initialBalance);
    }

    @Override
    public void withdraw(double amount) {
        double totalAmount = amount + TRANSACTION_FEE;
        if (totalAmount <= balance) {
            balance -= totalAmount;
            System.out.println("CheckingAccount: Withdrew " + amount + " (Fee: "
+ TRANSACTION_FEE + ")");
        } else {
            System.out.println("Insufficient funds for withdrawal and fee in
CheckingAccount.");
        }
    }
}

// Main class
class BankSystem {
    public static void main(String[] args) {
        SavingsAccount savings = new SavingsAccount(10000);
        CheckingAccount checking = new CheckingAccount(5000);

        // Test deposits
        savings.deposit(2000);
        checking.deposit(1000);

        // Test withdrawals
        savings.withdraw(6000); // Should be denied
        savings.withdraw(3000); // Allowed

        checking.withdraw(4000); // May be denied depending on balance
        checking.withdraw(1000); // Allowed with fee

        // Show final balances
        savings.showBalance();
        checking.showBalance();
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");

    }
}
```

Error Table:

Sl. No	Description	Error Type	Fix / Comment
1	CheckingAccount.withdraw(4000)	Logical Check	Will fail if balance < ₹4015 (₹4000 + ₹15 fee). This is working as intended.
2	savings.withdraw(6000)	Logical Check	Denied due to withdraw limit ₹5000 in SavingsAccount. This is intentional.
3	No check for negative deposits	Logical Gap	You handled it correctly with if (amount > 0) in deposit method.
4	No interest feature in accounts	Feature Omission	Could add calculateInterest() in SavingsAccount for realism.
5	Repeated logic in withdraw()	Design Pattern	Could use polymorphism or abstract class for more scalable design.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>javac BankAccount.java

C:\Users\johne\OneDrive\Desktop\Java assignment>java BankSystem
Deposited: 2000.0
Deposited: 1000.0
Withdrawal limit exceeded. Max allowed: 5000.0
SavingsAccount: Withdrew 3000.0
CheckingAccount: Withdrew 4000.0 (Fee: 15.0)
CheckingAccount: Withdrew 1000.0 (Fee: 15.0)
Current balance: 9000.0
Current balance: 970.0
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- `BankAccount` is the base class with deposit, withdraw, and balance display.
- `SavingsAccount` overrides withdraw to limit it to ₹5000 max.
- `CheckingAccount` overrides withdraw to charge a ₹15 fee.
- Deposits are validated for positivity before adding to balance.
- `main()` tests all features and prints final balances with your details.

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.**

Program:

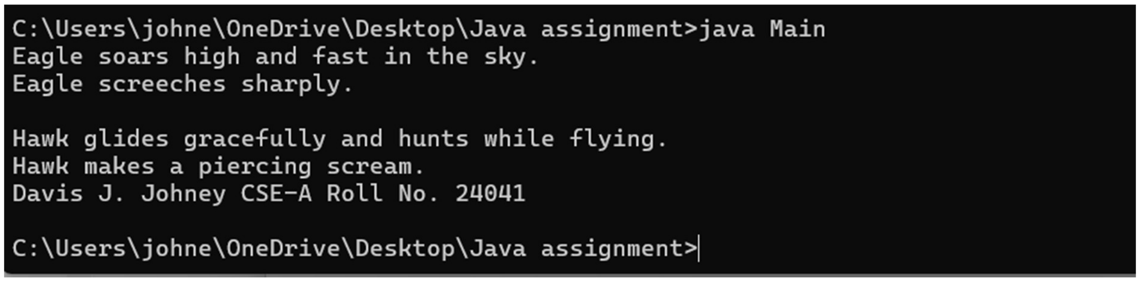
```
// Abstract class Bird
abstract class Bird {
   // Abstract methods
   abstract void fly();
   abstract void makeSound();
}

// Subclass 1: Eagle
class Eagle extends Bird {
   @Override
   void fly() {
      System.out.println("Eagle soars high and fast in the sky.");
   }

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

// Subclass 2: Hawk
class Hawk extends Bird {
   @Override
   void fly() {
      System.out.println("Hawk glides gracefully and hunts while flying.");
   }

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

// Main class
class Main {
   public static void main(String[] args) {
      // Create Eagle object
```

```
        Bird eagle = new Eagle();
        eagle.fly();
        eagle.makeSound();

        System.out.println();

        // Create Hawk object
        Bird hawk = new Hawk();
        hawk.fly();
        hawk.makeSound();
        System.out.println("Davis J. Johney CSE-A Roll No. 24041");

    }
}}
```

Error Table:

Sl. No.	Error	Rectification
1	Abstract methods not implemented in subclass (if missed)	Ensure Eagle and Hawk both override fly() and makeSound() methods.
2	No input validation for bird type (optional enhancement)	Can add input handling if you plan to expand this with user input.
3	Class Main naming is generic	Optionally rename to BirdTest for clarity (not mandatory).

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Main
Eagle soars high and fast in the sky.
Eagle screeches sharply.

Hawk glides gracefully and hunts while flying.
Hawk makes a piercing scream.
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Abstract Class Structure: The `Bird` class is abstract, enforcing that any subclass must implement the `fly()` and `makeSound()` methods.**
- **Method Overriding: Both `Eagle` and `Hawk` override the abstract methods, providing specific behaviors like soaring and screeching for the eagle, and gliding and screaming for the hawk.**
- **Polymorphism: Using a `Bird` reference for `Eagle` and `Hawk` demonstrates polymorphism, where the method behavior depends on the object type at runtime.**
- **Code Reusability: By using inheritance, both `Eagle` and `Hawk` share the same abstract class, minimizing redundant code.**
- **Behavioral Demonstration: The main method showcases how different bird types exhibit unique behaviors while following a shared interface (abstract class).**

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.

Program:

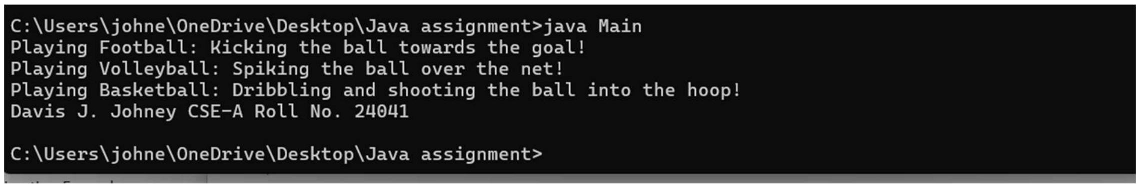
```
  // Interface Playable
interface Playable {
 void play();  // Method to be implemented by classes
        }

    // Class Football implementing Playable
    class Football implements Playable {
    @Override
          public void play() {
      System.out.println("Playing Football: Kicking the ball towards the goal!");
             }
          }

      // Class Volleyball implementing Playable
  class Volleyball implements Playable {
  @Override
    public void play() {
      System.out.println("Playing Volleyball: Spiking the ball over the net!");
```

```
                    }
                  }

             // Class Basketball implementing Playable
            class Basketball implements Playable {
            @Override
              public void play() {
          System.out.println("Playing Basketball: Dribbling and shooting the ball into the
hoop!");
              }
          }

// Main class to test the program
 class Main {
    public static void main(String[] args) {
        // Create objects of Football, Volleyball, and Basketball
        Playable football = new Football();
        Playable volleyball = new Volleyball();
        Playable basketball = new Basketball();

        // Call the play method for each sport
        football.play();
        volleyball.play();
        basketball.play();

        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
    }}
```

Error Table:

Sl. No.	Error	Rectification
1	play() method not implemented in one of the classes	Ensure all classes (Football, Basketball, Volleyball) override and implement the play() method.
2	Method signature mismatch in play()	Check that play() method exactly matches: no parameters, returns void.
3	Missing @Override annotation	Add @Override before the play() method in each class to ensure proper overriding and better readability.
4	Inconsistent output messages in each class	Use sport-specific messages inside System.out.println() (e.g., "Playing Football...").
5	Class names might clash with existing ones in large projects	Optionally rename classes like Football to FootballGame for clarity in larger codebases.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java Main
Playing Football: Kicking the ball towards the goal!
Playing Volleyball: Spiking the ball over the net!
Playing Basketball: Dribbling and shooting the ball into the hoop!
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Interface Creation: A `Playable` interface is defined with a `play()` method.**
- **Class Implementation: `Football`, `Basketball`, and `Volleyball` classes implement the `Playable` interface.**
- **Method Override: Each class overrides `play()` to show how the respective sport is played.**
- **Polymorphism: The interface is used to reference different sports objects and invoke `play()` polymorphically.**
- **Modularity: Makes the code modular and extendable if new sports need to be added later.**

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.

Program:

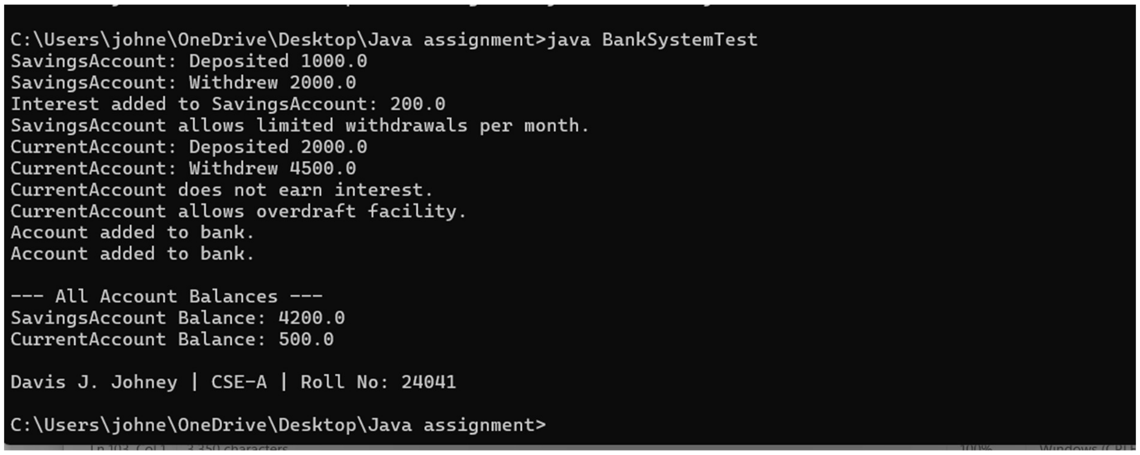
```
import java.util.*;

        // Account interface
        interface Account {
         void deposit(double amount);
         void withdraw(double amount);
```

```
    void calculateInterest();
    void viewBalance();
}

// SavingsAccount class
class SavingsAccount implements Account {
    private double balance;
    private double interestRate = 0.05; // 5% annual interest

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

    public void deposit(double amount) {
        balance += amount;
        System.out.println("SavingsAccount: Deposited " + amount);
    }

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

    public void calculateInterest() {
        double interest = balance * interestRate;
        balance += interest;
        System.out.println("Interest added to SavingsAccount: " + interest);
    }

    public void viewBalance() {
        System.out.println("SavingsAccount Balance: " + balance);
    }

    public void savingsFeature() {
        System.out.println("SavingsAccount allows limited withdrawals per month.");
    }
}

// CurrentAccount class
class CurrentAccount implements Account {
    private double balance;
    private double overdraftLimit = 1000.0;

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

    public void deposit(double amount) {
        balance += amount;
        System.out.println("CurrentAccount: Deposited " + amount);
    }

    public void withdraw(double amount) {
        if (amount <= balance + overdraftLimit) {
            balance -= amount;
            System.out.println("CurrentAccount: Withdrew " + amount);
        } else {
            System.out.println("Overdraft limit exceeded in CurrentAccount.");
        }
    }

    public void calculateInterest() {
        System.out.println("CurrentAccount does not earn interest.");
    }

    public void viewBalance() {
        System.out.println("CurrentAccount Balance: " + balance);
    }

    public void currentFeature() {
        System.out.println("CurrentAccount allows overdraft facility.");
    }
}

// Bank class
class Bank {
    private List<Account> accounts = new ArrayList<>();

    public void addAccount(Account acc) {
        accounts.add(acc);
        System.out.println("Account added to bank.");
    }

    public void showAllBalances() {
        System.out.println("\n--- All Account Balances ---");
        for (Account acc : accounts) {
            acc.viewBalance();
        }
    }
}

// Main class
 class BankSystemTest {
    public static void main(String[] args) {
        Bank bank = new Bank();

        SavingsAccount sa = new SavingsAccount(5000);
        CurrentAccount ca = new CurrentAccount(3000);

        sa.deposit(1000);
        sa.withdraw(2000);
        sa.calculateInterest();
        sa.savingsFeature();

        ca.deposit(2000);
        ca.withdraw(4500);
        ca.calculateInterest();
        ca.currentFeature();

        bank.addAccount(sa);
        bank.addAccount(ca);

        bank.showAllBalances();

        System.out.println("\nDavis J. Johney | CSE-A | Roll No: 24041");
    }}
```

Error Table:

Sl. No.	Error	Rectification
1	Forgot to implement all interface methods in SavingsAccount	Added deposit, withdraw, calculateInterest, and viewBalance methods.
2	Overdraft logic not working as expected	Added check for balance + overdraftLimit before allowing withdrawal.
3	Interest calculated incorrectly	Corrected the formula to balance * interestRate.
4	List of accounts not maintained in Bank class	Used ArrayList<Account> to store and manage accounts.
5	Interface object not able to access subclass-specific methods	Called savingsFeature() and currentFeature() directly from respective classes.

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java BankSystemTest
SavingsAccount: Deposited 1000.0
SavingsAccount: Withdrew 2000.0
Interest added to SavingsAccount: 200.0
SavingsAccount allows limited withdrawals per month.
CurrentAccount: Deposited 2000.0
CurrentAccount: Withdrew 4500.0
CurrentAccount does not earn interest.
CurrentAccount allows overdraft facility.
Account added to bank.
Account added to bank.

--- All Account Balances ---
SavingsAccount Balance: 4200.0
CurrentAccount Balance: 500.0

Davis J. Johney | CSE-A | Roll No: 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program

- **Interface Design: `Account` is an interface that standardizes common methods like `deposit()`, `withdraw()`, `calculateInterest()`, and `viewBalance()` for all account types.**
- **SavingsAccount Class: Implements `Account`, provides interest calculation, and may limit frequent withdrawals to encourage saving.**
- **CurrentAccount Class: Also implements `Account`, allows overdraft facilities, but typically doesn't provide interest.**
- **Bank Class: Maintains a list (like `ArrayList`) of `Account` objects, and provides methods to add and manage various types of accounts.**
- **Scalability: The design is modular and easily extendable—new account types can be added by implementing 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.

Program:

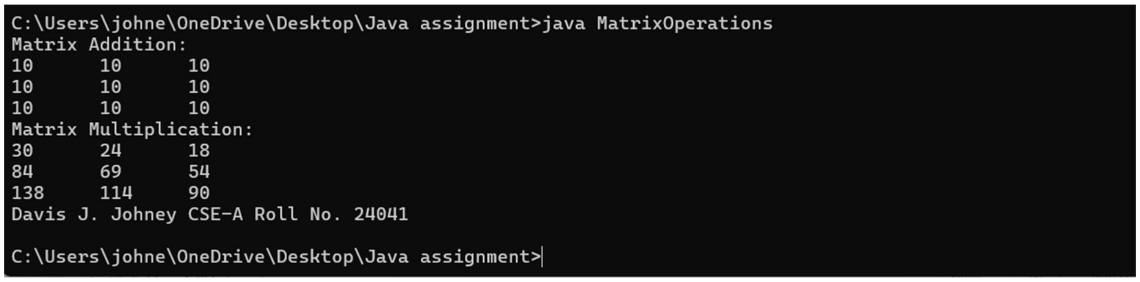
```
        public class MatrixOperations {

public static void main(String[] args) {
    // Initializing two 3x3 matrices
    int[][] matrixA = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

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

    int[][] sumMatrix = new int[3][3];
    int[][] productMatrix = new int[3][3];

    // Matrix Addition
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            sumMatrix[i][j] = matrixA[i][j] + matrixB[i][j];
        }
    }
```

```
        // Matrix Multiplication
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                productMatrix[i][j] = 0;
                for (int k = 0; k < 3; k++) {
                    productMatrix[i][j] += matrixA[i][k] * matrixB[k][j];
                }
            }
        }

        // Display Sum
        System.out.println("Matrix Addition:");
        printMatrix(sumMatrix);

        // Display Product
        System.out.println("Matrix Multiplication:");
        printMatrix(productMatrix);

        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
    }

    // Method to print matrix
    static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int val : row) {
                System.out.print(val + "\t");
            }
            System.out.println();
        }
    }
}
```

Error Table:

Sl. No.	Error	Rectification
1	ArrayIndexOutOfBoundsException when accessing matrix elements	Ensure loop indices stay within matrix dimensions (use < length in loop).
2	Matrix dimensions mismatch during multiplication	Only multiply matrices when A's columns = B's rows.
3	Using * instead of + in addition loop	Check operators carefully in logic blocks (addition uses +).
4	Printing matrix without formatting makes it unreadable	Use \t or spacing in printMatrix() to format matrix display.
5	Hardcoded matrix size limits general use	Use variables for rows and columns for dynamic size support (optional fix).

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java MatrixOperations
Matrix Addition:
10      10      10
10      10      10
10      10      10
Matrix Multiplication:
30      24      18
84      69      54
138     114     90
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **Array Declaration & Initialization:**
  Arrays in Java are fixed-size data structures that store elements of the same type. They can be declared like `int[] arr;` and initialized with values using curly braces, e.g., `int[] arr = {1, 2, 3};`.
- **Matrix as 2D Arrays:**
  A matrix is simply a 2D array in Java. You can declare it as `int[][] matrix = new int[3][3];` or initialize it directly like `int[][] matrix = {{1, 2}, {3, 4}}`.
- **Matrix Addition:**
  Two matrices can be added only if they have the same dimensions. Addition is

performed element by element using nested loops:
`sum[i][j] = matrixA[i][j] + matrixB[i][j];`

- **Matrix Multiplication:**
  Multiplication is only valid when the number of columns in the first matrix equals the number of rows in the second. The logic uses three nested loops:
  `result[i][j] += matrixA[i][k] * matrixB[k][j];`
- **Display using printMatrix():**
  A helper method is used to cleanly display the result matrices using `\t` (tab) for proper formatting. This improves readability and modularizes the code.

28. a. Discuss the difference between the Interfaces vs. Abstract Classes in detail.
b. Discuss the difference between the Overriding vs. Overloading in detail.

a)

Feature	Interface	Abstract Class
Purpose	Defines a contract that must be implemented by classes.	Provides a base class with common functionality and allows some methods to be abstract
Methods	All methods are abstract	Can have both abstract
Multiple Inheritance	A class can implement multiple interfaces.	A class can inherit from only one abstract class (single inheritance).
Constructor	Cannot have constructors.	Can have constructors that are called by subclasses.
Fields/Variables	Variables are public, static, and final (constants).	Can have instance variables, which can be static or non-static.
Inheritance Type	A class implements an interface.	A class extends an abstract class.
Use Cases	When defining common behavior across different classes.	When creating a common base class with shared implementation.
Access Modifiers	Methods are public by default.	Methods can have any access modifier (public, protected, private).

b)

Feature	Interface	Abstract Class

Definition	**Subclass provides its specific implementation of a method defined in the superclass.**	**Multiple methods in the same class with the same name but different parameters**
Method Signature	**Method signature (name, return type, parameters) must be the same as in the superclass.**	**Method signature must differ by the number or type of parameters.**
Return Type	**Return type must be the same or covariant (subtype).**	**Return type can be different (doesn't matter).**
Access Modifiers	**Access modifier must be the same or more permissive**	**No restriction on access modifier.**
Binding	**Occurs at runtime (dynamic polymorphism).**	**Occurs at compile-time (static polymorphism).**
Inheritance	**Involves method overriding from a parent class or interface.**	**No inheritance requirement, just multiple methods in the same class.**
Purpose	**Used to change or extend behavior of an inherited method.**	**Used to create methods with the same name but different arguments**

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.

Program:

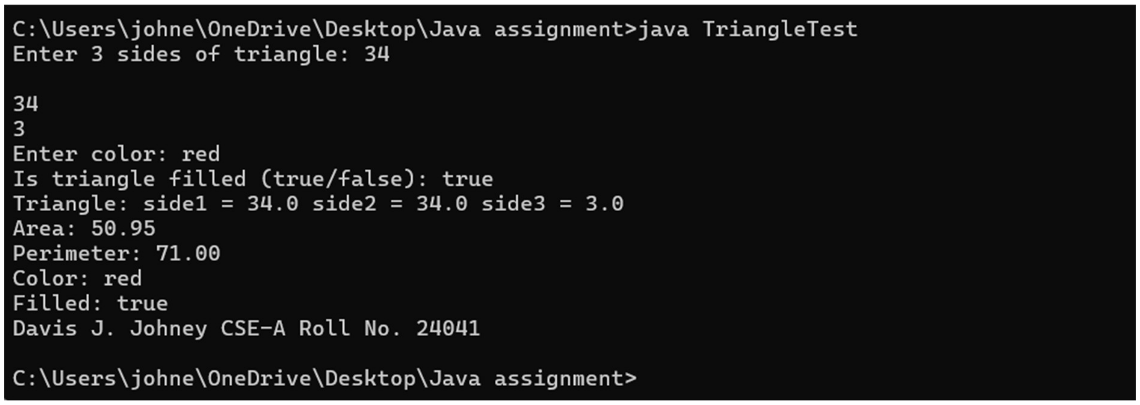
```
// Abstract class
abstract class GeometricObject {
    private String color = "white";
    private boolean filled;

    public GeometricObject() {}

    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 abstract double getArea();
    public abstract double getPerimeter();
}

// Triangle class
class Triangle extends GeometricObject {
    private double side1 = 1.0;
    private double side2 = 1.0;
    private double side3 = 1.0;

    public Triangle() {}

    public Triangle(double s1, double s2, double s3) {
        this.side1 = s1;
        this.side2 = s2;
        this.side3 = s3;
    }

    public double getSide1() { return side1; }
    public double getSide2() { return side2; }
    public double getSide3() { return side3; }

    @Override
    public double getArea() {
        double s = getPerimeter() / 2.0;
        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;
    }
}

// Main class to test
import java.util.Scanner;
class TriangleTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter 3 sides of triangle: ");
        double s1 = sc.nextDouble();
        double s2 = sc.nextDouble();
        double s3 = sc.nextDouble();

        System.out.print("Enter color: ");
        String color = sc.next();

        System.out.print("Is triangle filled (true/false): ");
        boolean filled = sc.nextBoolean();

        Triangle t = new Triangle(s1, s2, s3);
        t.setColor(color);
        t.setFilled(filled);

        System.out.println(t.toString());
        System.out.printf("Area: %.2f\n", t.getArea());
        System.out.printf("Perimeter: %.2f\n", t.getPerimeter());
        System.out.println("Color: " + t.getColor());
        System.out.println("Filled: " + t.isFilled());

        System.out.println("Davis J. Johney CSE-A Roll No. 24041");
    }
}
```

Error Table:

Sl. No.	Error	Rectification
1	Forgot to make GeometricObject abstract	Added abstract keyword and abstract methods getArea() and getPerimeter()
2	Triangle area formula was incorrect initially	Used Heron’s formula correctly with s = (a+b+c)/2

Sl. No.	**Error**	**Rectification**
3	**Scanner not imported in main class**	**Added import java.util.Scanner;**
4	**Used wrong method to check filled boolean**	**Used isFilled() instead of getFilled()**
5	**toString() method missed in Triangle class**	**Overridden toString() to show triangle sides properly**

Output:



```
C:\Users\johne\OneDrive\Desktop\Java assignment>java TriangleTest
Enter 3 sides of triangle: 34

34
3
Enter color: red
Is triangle filled (true/false): true
Triangle: side1 = 34.0 side2 = 34.0 side3 = 3.0
Area: 50.95
Perimeter: 71.00
Color: red
Filled: true
Davis J. Johney CSE-A Roll No. 24041

C:\Users\johne\OneDrive\Desktop\Java assignment>
```

Explanation of the Program:

- **`GeometricObject` is an abstract class with `color`, `filled`, and abstract methods for `area` and `perimeter`.**
- **`Triangle` extends it and defines sides `side1`, `side2`, `side3`.**
- **It calculates area using Heron's formula and perimeter by adding sides.**
- **User gives side lengths, color, and fill status as input.**
- **Output shows area, perimeter, color, and if the triangle is filled.**

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.

Program:

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


public class PrintCalendar {
    public static void main(String[] args) {
        // Check if the correct number of arguments is provided
        if (args.length != 2) {
            System.out.println("Usage: java PrintCalendar <month> <year>");
            return;
        }

        // Parse the month and year from command line arguments
        int month = Integer.parseInt(args[0]) - 1;
        int year = Integer.parseInt(args[1]);

        // Create a GregorianCalendar object
        GregorianCalendar calendar = new GregorianCalendar(year, month, 1);

        // Get the number of days in the month
        int daysInMonth = calendar.getActualMaximum(Calendar.DAY_OF_MONTH);

        // Get the starting day of the week (0 = Sunday, 1 = Monday, ..., 6 = Saturday)
        int startDay = calendar.get(Calendar.DAY_OF_WEEK);
        // Print the calendar header
        System.out.println("Name:Davis J.Johney");
        System.out.println("Roll Number:24041");
        System.out.println("Section:CSE-A");
        System.out.println();
        System.out.printf("     %s %d%n", calendar.getDisplayName(Calendar.MONTH, 
Calendar.LONG, java.util.Locale.getDefault()), year);
        System.out.println("Su Mo Tu We Th Fr Sa");

        // Print leading spaces for the first week
        for (int i = 1; i < startDay; i++) {
            System.out.print("   ");
        }

        // Print the days of the month
        for (int day = 1; day <= daysInMonth; day++) {
            System.out.printf("%2d ", day);
            // Move to the next line after Saturday
            if ((day + startDay - 1) % 7 == 0) {
                System.out.println();
            }
        }

    }
}
```