GOVERNMENT OF KERALA

DEPARTMENT OF TECHNICAL EDUCATION

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(GOVT. ENGINEERING COLLEGE)

KOTTAYAM - 686501

RECORD BOOK

GOVERNMENT OF KERALA

DEPARTMENT OF TECHNICAL EDUCATION

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(GOVT. ENGINEERING COLLEGE)

KOTTAYAM - 686501

20MCA132

OBJECT ORIENTED PROGRAMMING LAB

Name: DEVIKA B

Branch: Master of Computer Applications

Semester: 2

Roll No: 22

CERTIFIED BONAFIDE RECORD WORK DONE BY

Reg No. ....

STAFF IN CHARGE

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Contents

Experiment No: 1 **Date: 10/02/2025**

Even-Odd Classification

Aim

Write a Java program to check whether an input number is even or odd.

Algorithm

```
1. Start
2. Take an integer as input from the user.
3. Use an if-else statement to check if the number is even or odd.
4. Print the result accordingly.
5. Stop
```

Source Code

```java
import java.util.Scanner;
public class EvenOdd{
        public static void main(String arg[]){
                Scanner s=new  Scanner(System.in);
                System.out.print("enter the num:");
                int num=s.nextInt();
                if(num%2==0){
                        System.out.println("Even");

                }
                else{
                        System.out.println("Odd");
                }
        }
}
```

Result

The program was executed successfully.
When the input 2 was provided, the output was; ”Even”

```
enter the num:10
Even
```

Experiment No: 2 **Date: 10/02/2025**

Sum of First n Natural Numbers

Aim

Write a Java program to compute the sum of the first n natural numbers.

Algorithm

```
1. Start
2. Take an integer n as input from the user.
3. Use either a for loop or a while loop to compute the sum.
4. Print the result.
5. Stop
```

Source Code

```java
import java.util.Scanner;
public class Sum{
    public static void main(String arg[]){
        System.out.print("Enter the n:");
        Scanner s=new Scanner(System.in);
        int n=s.nextInt();
        int r=0;
        for(int i=0;i<=n;i++){
            r=r+i;
        }
        System.out.println("sum="+ r);
    }
}
```

Result

The program was executed successfully.
When the input 10 was provided, the output was: 55

```
Enter the n:6
sum= 21
```

Experiment No: 3 **Date: 10/02/2025**

Factorial of a Number

Aim

Write a Java program to compute the factorial of a given number.

Algorithm

```
1. Start
2. Take an integer as input from the user.
3. Compute the factorial using either a for loop or a while loop.
4. Print the result.
5. Stop
```

Source Code

```java
import java.util.Scanner;
public class Fact{
    public static void main(String arg[]){
        System.out.println("enter the num:");
        Scanner s=new Scanner(System.in);
        int n=s.nextInt();
        int r=1;
        for(int i=1;i<=n;i++){
            r=r*i;
        }
        System.out.println("fact:"+ r);
    }
}
```

Result

The program was executed successfully.
When the input 5 was provided, the output was: 120

```
enter the num:5
fact:120
```

Experiment No: 4 **Date: 10/02/2025**

Assigning Grades Based on Numeric Score

Aim

Write a Java program that assigns a grade based on a numeric score.

Algorithm

```
1. Start
2. Take a numeric score (0 - 100) as input from the user.
3. Use either an if-else if-else structure or a switch-case statement to assign a
   grade:
   • 90{100 → A
   • 80{89 → B
   • 70{79 → C
   • 60{69 → D
   • Below 60 → F
4. Print the assigned grade.
5. Stop
```

Source Code

```java
import java.util.Scanner;
public class Grade{
	public static void main(String[] args){
		Scanner obj = new Scanner(System.in);
		System.out.println("Enter a number : ");
		int grade = obj.nextInt();
		if(grade >= 90){
			System.out.println("Grade A");
		}
		else if(grade >= 80){
			System.out.println("Grade B");
		}
		else if(grade >= 70){
			System.out.println("Grade C");
		}
		else if(grade >= 60){
			System.out.println("Grade D");
		}
		else{
			System.out.println("Fail");
		}
	}
}
```

Result

The program was executed successfully.

When the input 64 was provided, the output was: D

```
Enter a number :
63
Grade D
Enter a number :
17
Fail
```

Experiment No: 5 **Date: 17/02/2025**

Find Product with Lowest Price

Aim

Write a Java program to define a class Product with data members pcode, pname, and price. Find and display the product with the lowest price.

Algorithm

```
1. Start
2. Create a ‘Product‘ class with attributes: ‘pcode‘, ‘pname‘, and ‘price‘.
3. Define a constructor to initialize product details.
4. In the ‘main‘ method:
   - Create a scanner object for user input.
   - Create an array ‘products‘ to store 5 product objects.
5. Repeat for each product (5 times):
   - Prompt the user to enter ‘pcode‘, ‘pname‘, and ‘price‘.
   - Store these values in a ‘Product‘ object.
   - Add the product to the ‘products‘ array.
6. Initialize ‘lowestPriceProduct‘ with the first product in the array.
7. Loop through the ‘products‘ array:
   - If a product has a lower price than ‘lowestPriceProduct‘,
     update ‘lowestPriceProduct‘.
8. Display the product with the lowest price (code, name, and price).
9. Close the scanner.
10. Stop
```

Source Code

```java
import java.util.Scanner;

class Product {
    String pcode;
    String pname;
    double price;

    public Product(String pcode, String pname, double price) {
        this.pcode = pcode;
        this.pname = pname;
        this.price = price;
    }
}

public class SimpleProductManager {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Product[] products = new Product[5];
        for (int i = 0; i < products.length; i++) {
            System.out.println("Enter product code:");
            String pcode = scanner.nextLine();
            System.out.println("Enter product name:");
            String pname = scanner.nextLine();
            System.out.println("Enter product price:");
            double price = scanner.nextDouble();
            scanner.nextLine();
            products[i] = new Product(pcode, pname, price);
        }
        Product lowestPriceProduct = products[0];
        for (Product product : products) {
            if (product.price < lowestPriceProduct.price) {
                lowestPriceProduct = product;
            }
        }
        System.out.println("Product with the lowest price:");
        System.out.println("Code: " + lowestPriceProduct.pcode);
        System.out.println("Name: " + lowestPriceProduct.pname);
        System.out.println("Price: $" + lowestPriceProduct.price);
        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter product code:
1
Enter product name:
milk
Enter product price:
500
Enter product code:
2
Enter product name:
egg
Enter product price:
300
Enter product code:
3
Enter product name:
apple
Enter product price:
560
Enter product code:
4
Enter product name:
orange
Enter product price:
600
Enter product code:
5
```

```
Enter product name:
banana
Enter product price:
520
Product with the lowest price:
Code: 2
Name: egg
Price: $300.0
```

Experiment No: 6 **Date: 17/02/2025**

Complex Number Operations

Aim

Write a Java program to perform addition and multiplication of complex numbers, with inputs provided by the user.

Algorithm

```
1. Start
2. Define a class ‘Complex‘ with attributes ‘real‘ and ‘imaginary‘.
3. Create a constructor to initialize ‘real‘ and ‘imaginary‘ values.
4. Define a method ‘add(Complex other)‘:
   - Add the real parts.
   - Add the imaginary parts.
   - Return a new ‘Complex‘ object with the sum.
5. Define a method ‘multiply(Complex other)‘:
   - Compute the real part using (real1 * real2 - imaginary1 * imaginary2).
   - Compute the imaginary part using (real1 * imaginary2 + imaginary1 * real2).
   - Return a new ‘Complex‘ object with the product.
6. In the ‘main‘ method:
   - Create a ‘Scanner‘ object for user input.
   - Prompt the user to enter the real and imaginary parts of the
     first complex number.
   - Read the input values and store them.
   - Prompt the user to enter the real and imaginary parts of the
     second complex number.
   - Read the input values and store them.
7. Create two ‘Complex‘ objects using the input values.
8. Call the ‘add‘ method and store the result in ‘sum‘.
9. Call the ‘multiply‘ method and store the result in ‘product‘.
10. Display the sum and product of the complex numbers.
11. Close the ‘Scanner‘ object.
12. Stop
```

Source Code

```java
import java.util.Scanner;

class Complex {
    double real, imaginary;

    Complex(double r, double i) {
        this.real = r;
        this.imaginary = i;
    }

    Complex add(Complex other) {
        return new Complex(this.real + other.real, this.imaginary + other.imaginary);
    }

    Complex multiply(Complex other) {
        return new Complex(this.real * other.real - this.imaginary * other.imaginary,
                           this.real * other.imaginary + this.imaginary * other.real);
    }

    @Override
    public String toString() {
        return real + " + " + imaginary + "i";
    }
}

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

        System.out.print("Enter real part of first complex number: ");
        double r1 = scanner.nextDouble();
        System.out.print("Enter imaginary part of first complex number: ");
        double i1 = scanner.nextDouble();

        System.out.print("Enter real part of second complex number: ");
        double r2 = scanner.nextDouble();
        System.out.print("Enter imaginary part of second complex number: ");
        double i2 = scanner.nextDouble();

        Complex c1 = new Complex(r1, i1);
        Complex c2 = new Complex(r2, i2);

        Complex sum = c1.add(c2);
        Complex product = c1.multiply(c2);

        System.out.println("Sum: " + sum);
        System.out.println("Product: " + product);
        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter real part of first complex number: 3
Enter imaginary part of first complex number: 4
Enter real part of second complex number: 5
Enter imaginary part of second complex number: 8
Sum: 8.0 + 12.0i
Product: -17.0 + 44.0i
```

Experiment No: 7 **Date: 17/02/2025**

Matrix Addition

Aim

Write a Java program to perform matrix addition.

Algorithm

```
1. Start
2. Define a class ‘Matrix‘ with attributes ‘rows‘, ‘columns‘, and ‘data‘.
3. Create a constructor to initialize ‘rows‘, ‘columns‘, and allocate memory
   for ‘data‘.
4. Define a method ‘fillMatrix(Scanner scanner)‘:
   - Prompt the user to enter matrix elements.
   - Read the input values and store them in ‘data‘.
5. Define a method ‘add(Matrix other)‘:
   - Create a new ‘Matrix‘ object ‘result‘ with the same dimensions.
   - For each element, add the corresponding elements from both matrices.
   - Return the ‘result‘ matrix.
6. Define a method ‘display()‘:
   - Print the elements of the matrix in a structured format.
7. In the ‘main‘ method:
   - Create a ‘Scanner‘ object for user input.
   - Prompt the user to enter the number of ‘rows‘ and ‘columns‘.
   - Read and store the values.
   - Create two ‘Matrix‘ objects with the given dimensions.
   - Call ‘fillMatrix()‘ for both matrices to take user input.
   - Compute the sum using the ‘add()‘ method.
   - Display the resultant matrix using ‘display()‘.
   - Close the ‘Scanner‘ object.
8. Stop
```

Source Code

```java
import java.util.Scanner;

class Matrix {
    private int rows, columns;
    private int[][] data;

    Matrix(int rows, int columns) {
        this.rows = rows;
        this.columns = columns;
        data = new int[rows][columns];
    }

    void fillMatrix(Scanner scanner) {
        System.out.println("Enter matrix elements:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                data[i][j] = scanner.nextInt();
            }
        }
    }

    Matrix add(Matrix other) {
        Matrix result = new Matrix(rows, columns);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                result.data[i][j] = this.data[i][j] + other.data[i][j];
            }
        }
        return result;
    }

    void display() {
        for (int[] row : data) {
            for (int value : row) {
                System.out.print(value + " ");
            }
            System.out.println();
        }
    }
}

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

        System.out.print("Enter number of rows: ");
        int rows = scanner.nextInt();
        System.out.print("Enter number of columns: ");
        int columns = scanner.nextInt();

        Matrix matrix1 = new Matrix(rows, columns);
        Matrix matrix2 = new Matrix(rows, columns);

        matrix1.fillMatrix(scanner);
        matrix2.fillMatrix(scanner);

        Matrix sumMatrix = matrix1.add(matrix2);

        System.out.println("Resultant Matrix after Addition:");
        sumMatrix.display();

        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter number of rows: 3
Enter number of columns: 3
Enter matrix elements:
1 2 3
4 5 6
7 8 9
Enter matrix elements:
2 3 5
1 7 4
2 4 7
Resultant Matrix after Addition:
3 5 8
5 12 10
9 12 16
```

Experiment No: 8 **Date: 24/02/2025**

Employee Search Using an Array of Objects

Aim

Write a Java program to store employee details including employee number, name, and salary, and search for an employee by employee number.

Algorithm

```
1. Start
2. Define a class ‘Employee‘ with:
   - Attributes: ‘empNumber‘, ‘empName‘, and ‘salary‘.
   - A constructor to initialize these attributes.
   - A method ‘display()‘ to print employee details.
3. In the ‘main()‘ method:
   - Create a ‘Scanner‘ object for user input.
   - Prompt the user to enter the number of employees (‘n‘).
   - Create an array ‘employees‘ of size ‘n‘.
4. Input Employee Details:
   - Loop from ‘i = 0‘ to ‘n-1‘:
     - Prompt the user for ‘empNumber‘, ‘empName‘, and ‘salary‘.
     - Store the details in an ‘Employee‘ object and add it to the array.
5. Search for an Employee:
   - Prompt the user to enter an ‘employee number‘ to search.
   - Initialize ‘found = false‘.
   - Loop through the ‘employees‘ array:
     - If ‘empNumber‘ matches the search number:
       - Display employee details.
       - Set ‘found = true‘ and exit the loop.
6. If ‘found == false‘, print ‘"Employee not found!"‘.
7. Close the ‘Scanner‘ object.
8. Stop
```

Source Code

```java
import java.util.Scanner;
class Employee {
    int empNumber;
    String empName;
    double salary;
    Employee(int empNumber, String empName, double salary) {
        this.empNumber = empNumber;
        this.empName = empName;
        this.salary = salary;
    }
    void display() {
        System.out.println("Employee Number: " + empNumber);
        System.out.println("Employee Name: " + empName);
        System.out.println("Salary: $" + salary);
    }
}
public class EmployeeSearch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of employees: ");
        int n = scanner.nextInt();
        Employee[] employees = new Employee[n];
        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for Employee " + (i + 1) + ":");
            System.out.print("Employee Number: ");
            int empNumber = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Employee Name: ");
            String empName = scanner.nextLine();
            System.out.print("Salary: ");
            double salary = scanner.nextDouble();
            employees[i] = new Employee(empNumber, empName, salary);
        }
        System.out.print("\nEnter employee number to search: ");
        int searchNumber = scanner.nextInt();
        boolean found = false;
        for (Employee emp : employees) {
            if (emp.empNumber == searchNumber) {
                System.out.println("\nEmployee Found:");
                emp.display();
                found = true;
                break;
            }
        }
        if (!found) {
            System.out.println("Employee not found!");
        }
        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter the number of employees: 3
Enter details for Employee 1:
Employee Number: 1
Employee Name: Devika
Salary: 2500
Enter details for Employee 2:
Employee Number: 2
Employee Name: Abhijith
Salary: 4000
Enter details for Employee 3:
Employee Number: 3
Employee Name: Achu
```

```
Salary: 5600

Enter employee number to search: 1

Employee Found:
Employee Number: 1
Employee Name: Devika
Salary: $2500.0
```

Experiment No: 9 **Date: 24/02/2025**

String Search in an Array

Aim

Write a Java program to store ‘n‘ strings in an array. Search for a given string. If found, print its index; otherwise, display ”String not found.”

Algorithm

```
1. Start
2. Create a ‘Scanner‘ object for user input.
3. Prompt the user to enter the number of strings (‘n‘).
4. Create an array ‘strings‘ of size ‘n‘ to store the strings.
5. Input ‘n‘ strings from the user and store them in the array.
6. Prompt the user to enter the string to search.
7. Initialize ‘index = -1‘ to track the search result.
8. Loop through the ‘strings‘ array:
   - If the current string matches the search string:
     - Store the index.
     - Break the loop.
9. If ‘index != -1‘, print ‘"String found at index: index"‘,
   else print ‘"String not found."‘
10. Close the ‘Scanner‘ object.
11. Stop
```

Source Code

```java
import java.util.Scanner;

public class StringSearch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of strings: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume the newline character
        String[] strings = new String[n];
        System.out.println("Enter the strings:");
        for (int i = 0; i < n; i++) {
            strings[i] = scanner.nextLine();
        }
        System.out.print("Enter the string to search: ");
        String searchString = scanner.nextLine();
        int index = -1;
        for (int i = 0; i < n; i++) {
            if (strings[i].equals(searchString)) {
                index = i;
                break;
            }
        }
        if (index != -1) {
            System.out.println("String found at index: " + index);
        } else {
            System.out.println("String not found.");
        }
        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter the number of strings: 4
Enter the strings:
Devika
Abhijith
Achu
Anu
Enter the string to search: Devika
String found at index: 0
```

Experiment No: 10 **Date: 24/02/2025**

String Manipulations

Aim

Write a Java program to perform various string manipulations, including finding the length, converting to uppercase and lowercase, extracting characters and substrings, and reversing the string.

Algorithm

```
1. Start
2. Create a ‘Scanner‘ object for user input.
3. Prompt the user to enter a string and store it in ‘str‘.
4. Display the length of the string using ‘str.length()‘.
5. Convert and display the string in uppercase using ‘str.toUpperCase()‘.
6. Convert and display the string in lowercase using ‘str.toLowerCase()‘.
7. Prompt the user to enter an index to extract a character.
   - If the index is valid, display the character using ‘str.charAt(index)‘.
   - Else, display ‘"Invalid index!"‘.
8. Prompt the user to enter the start and end index for a substring.
   - If the range is valid, extract and display the substring using
    ‘str.substring(start, end)‘.
   - Else, display ‘"Invalid substring range!"‘.
9. Reverse the string using ‘StringBuilder(str).reverse().toString()‘
   and display it.
10. Close the ‘Scanner‘ object.
11. Stop
```

Source Code

```java
import java.util.Scanner;

public class StringManipulations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
        System.out.println("Length of the string: " + str.length());
        System.out.println("Uppercase: " + str.toUpperCase());
        System.out.println("Lowercase: " + str.toLowerCase());
        System.out.print("Enter index to extract character: ");
        int index = scanner.nextInt();
        if (index >= 0 && index < str.length()) {
            System.out.println("Character at index " + index + ": " +
    str.charAt(index));
        } else {
            System.out.println("Invalid index!");
        }
        System.out.print("Enter start and end index for substring: ");
        int start = scanner.nextInt();
        int end = scanner.nextInt();
        if (start >= 0 && end <= str.length() && start < end) {
            System.out.println("Substring: " + str.substring(start, end
    ));
        } else {
            System.out.println("Invalid substring range!");
        }
        String reversed = new StringBuilder(str).reverse().toString();
        System.out.println("Reversed string: " + reversed);
        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter a string: Devika
Length of the string: 6
Uppercase: DEVIKA
Lowercase: devika
Enter index to extract character: 3
Character at index 3: i
Enter start and end index for substring: 1 5
Substring: evik
Reversed string: akiveD
```

Experiment No: 11 **Date: 24/02/2025**

Inheritance in Java

Aim

Write a Java program to implement hierarchical inheritance for a book management system. Define a base class 'Publisher', a derived class 'Book', and two subclasses 'Literature' and 'Fiction'. Include methods to read and display book details and demonstrate the functionality using user input.

Algorithm

```
1. Start
2. Define a base class ‘Publisher‘ with:
   - Attribute ‘publisherName‘.
   - Constructor to initialize ‘publisherName‘.
   - Method ‘displayPublisher()‘ to display publisher details.
3. Define a derived class ‘Book‘ (inherits from ‘Publisher‘) with:
   - Attributes ‘bookTitle‘ and ‘author‘.
   - Constructor to initialize ‘bookTitle‘ and ‘author‘.
   - Method ‘displayBook()‘ to display book details.
4. Define a subclass ‘Literature‘ (inherits from ‘Book‘) with:
   - Attribute ‘genre‘ set to ‘"Literature"‘.
   - Constructor to initialize inherited attributes.
   - Method ‘displayDetails()‘ to display book details along with genre.
5. Define a subclass ‘Fiction‘ (inherits from ‘Book‘) with:
   - Attribute ‘genre‘ set to ‘"Fiction"‘.
   - Constructor to initialize inherited attributes.
   - Method ‘displayDetails()‘ to display book details along with genre.
6. In ‘main‘ method:
   - Create a ‘Scanner‘ object for user input.
   - Prompt the user to enter details for a Literature book and store inputs.
   - Create an object of ‘Literature‘ class using user inputs.
   - Prompt the user to enter details for a Fiction book and store inputs.
   - Create an object of ‘Fiction‘ class using user inputs.
7. Display Literature book details using ‘displayDetails()‘.
8. Display Fiction book details using ‘displayDetails()‘.
9. Close the ‘Scanner‘ object.
10. Stop
```

Source Code

```java
import java.util.Scanner;

class Publisher {
    String publisherName;

    Publisher(String publisherName) {
        this.publisherName = publisherName;
    }

    void displayPublisher() {
        System.out.println("Publisher: " + publisherName);
    }
}

class Book extends Publisher {
    String bookTitle;
    String author;

    Book(String publisherName, String bookTitle, String author) {
        super(publisherName);
        this.bookTitle = bookTitle;
        this.author = author;
    }

    void displayBook() {
        displayPublisher();
        System.out.println("Book Title: " + bookTitle);
        System.out.println("Author: " + author);
    }
}

class Literature extends Book {
    String genre = "Literature";

    Literature(String publisherName, String bookTitle, String author) {
        super(publisherName, bookTitle, author);
    }

    void displayDetails() {
        displayBook();
        System.out.println("Genre: " + genre);
    }
}

class Fiction extends Book {
    String genre = "Fiction";

    Fiction(String publisherName, String bookTitle, String author) {
        super(publisherName, bookTitle, author);
    }

    void displayDetails() {
        displayBook();
        System.out.println("Genre: " + genre);
    }
}

public class BookManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Literature Book Details:");
        System.out.print("Publisher: ");
        String pub1 = scanner.nextLine();
        System.out.print("Book Title: ");
        String title1 = scanner.nextLine();
        System.out.print("Author: ");
        String author1 = scanner.nextLine();
        Literature literatureBook = new Literature(pub1, title1,
    author1);
        System.out.println("\nEnter Fiction Book Details:");
        System.out.print("Publisher: ");
        String pub2 = scanner.nextLine();
        System.out.print("Book Title: ");
        String title2 = scanner.nextLine();
        System.out.print("Author: ");
        String author2 = scanner.nextLine();
        Fiction fictionBook = new Fiction(pub2, title2, author2);
        System.out.println("\n--- Literature Book Details ---");
        literatureBook.displayDetails();
        System.out.println("\n--- Fiction Book Details ---");
        fictionBook.displayDetails();
        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter Literature Book Details:
Publisher: HarperTorch
Book Title: The Alchemist
Author: Paulo Coelho

Enter Fiction Book Details:
Publisher: T. Egerton
Book Title: Pride and Prejudice
Author: Jane Austen

--- Literature Book Details ---
Publisher: HarperTorch
Book Title: The Alchemist
Author: Paulo Coelho
Genre: Literature

--- Fiction Book Details ---
Publisher: T. Egerton
Book Title: Pride and Prejudice
Author: Jane Austen
Genre: Fiction
```

Experiment No: 12 **Date: 03/03/2025**

Calculate Area and Perimeter Using Interfaces

Aim

Write a Java Program to create an interface having prototypes of functions ‘area()‘ and ‘perimeter()‘. Create two classes ‘Circle‘ and ‘Rectangle‘ which implement the above interface. Develop a menu-driven program to find the area and perimeter of these shapes.

Algorithm

```
1. Start
2. Define an interface ‘Shape‘ with methods:
   - ‘area()‘
   - ‘perimeter()‘
3. Create class ‘Circle‘ implementing ‘Shape‘:
   - Define attribute ‘radius‘
   - Implement ‘area()‘ as  * $radius^2$
   - Implement ‘perimeter()‘ as 2 *  * radius
4. Create class ‘Rectangle‘ implementing ‘Shape‘:
   - Define attributes ‘length‘ and ‘width‘
   - Implement ‘area()‘ as length * width
   - Implement ‘perimeter()‘ as 2 * (length + width)
5. In ‘main()‘ method:
   - Display menu options:
     1. Calculate area & perimeter of Circle
     2. Calculate area & perimeter of Rectangle
     3. Exit program
   - Read user choice
   - If Circle is chosen:
     - Prompt user to enter radius
     - Create ‘Circle‘ object and display area & perimeter
   - If Rectangle is chosen:
     - Prompt user to enter length and width
     - Create ‘Rectangle‘ object and display area & perimeter
   - If Exit is chosen, terminate program
   - If an invalid choice is entered, prompt user to re-enter
6. Repeat until the user exits
7. Stop
```

Source Code

```java
import java.util.Scanner;

// Interface with method prototypes for area and perimeter
interface Shape {
    double area();
    double perimeter();
```

```java
}

// Circle class implementing Shape interface
class Circle implements Shape {
    double radius;

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

    @Override
    public double area() {
        return Math.PI * radius * radius;
    }

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

// Rectangle class implementing Shape interface
class Rectangle implements Shape {
    double length, width;

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

    @Override
    public double area() {
        return length * width;
    }

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

// Main class for menu-driven execution
public class AreaPerimeterCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Circle - Area & Perimeter");
            System.out.println("2. Rectangle - Area & Perimeter");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter radius of the circle: ");
                    double radius = scanner.nextDouble();
                    Circle circle = new Circle(radius);
                    System.out.println("Area of Circle: " + circle.area());
                    System.out.println("Perimeter of Circle: " + circle.perimeter());
                    break;
                case 2:
                    System.out.print("Enter length of the rectangle: ");
                    double length = scanner.nextDouble();
                    System.out.print("Enter width of the rectangle: ");
                    double width = scanner.nextDouble();
                    Rectangle rectangle = new Rectangle(length, width);
                    System.out.println("Area of Rectangle: " + rectangle.area());
                    System.out.println("Perimeter of Rectangle: " + rectangle.perimeter());
                    break;
                case 3:
                    System.out.println("Exiting program...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        }
    }
}
```

Result

The program was executed successfully.

```
Menu:
1. Circle - Area & Perimeter
2. Rectangle - Area & Perimeter
3. Exit
Enter your choice: 1
Enter radius of the circle: 3
Area of Circle: 28.274333882308138
Perimeter of Circle: 18.84955592153876

Menu:
1. Circle - Area & Perimeter
2. Rectangle - Area & Perimeter
3. Exit
Enter your choice: 2
Enter length of the rectangle: 4
Enter width of the rectangle: 5
Area of Rectangle: 20.0
Perimeter of Rectangle: 18.0
```

```
Menu:
1. Circle - Area & Perimeter
2. Rectangle - Area & Perimeter
3. Exit
Enter your choice: 3
Exiting program...
```

Experiment No: 13 **Date: 03/03/2025**

Program to Manage Employee Collection

Aim

Create a Java program to manage a collection of employees in a company. Implement an abstract class Employee with fields name (String) and salary (double), and an abstract method calculateSalary(). Create two subclasses: Manager (with a bonus field) and Developer (with an experience field), both overriding calculateSalary() to calculate the total salary. Implement an interface Benefits with a method calculateBenefits(), where Manager provides a fixed insurance benefit and Developer provides an allowance based on experience. Use polymorphism to store Employee objects in a list and display employee details and salary. Add method overloading in Manager for project assignment, where one method takes just a project name and the other takes both the project name and the number of team members.

Algorithm

```
1. Start
2. Define an abstract class ‘Employee‘:
   - Attributes: ‘name‘, ‘salary‘
   - Abstract method: ‘calculateSalary()‘
   - Method: ‘display()‘
3. Define an interface ‘Benefits‘:
   - Method ‘calculateBenefits()‘
4. Create class ‘Manager‘ inheriting ‘Employee‘ and implementing ‘Benefits‘:
   - Attribute: ‘bonus‘
   - Override ‘calculateSalary()‘ as ‘salary + bonus‘
   - Override ‘calculateBenefits()‘ as a fixed value
   - Method overloading for project assignment:
     - Assign project with just project name
     - Assign project with project name and number of team members
5. Create class ‘Developer‘ inheriting ‘Employee‘ and implementing ‘Benefits‘:
   - Attribute: ‘experience‘
   - Override ‘calculateSalary()‘ as ‘salary + (experience * 500)‘
   - Override ‘calculateBenefits()‘ as ‘experience * 200‘
6. In ‘main()‘:
   - Create a list to store ‘Employee‘ objects
   - Prompt user for the number of employees
   - Loop to collect employee details:
     - If ‘Manager‘, collect bonus and project assignment
     - If ‘Developer‘, collect experience
     - Store objects in the list
   - Display details of all employees with salaries and benefits
7. Stop
```

Source Code

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

abstract class Employee {
    String name;
    double salary;

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

    abstract double calculateSalary();

    void display() {
        System.out.println("Name: " + name);
        System.out.println("Base Salary: $" + salary);
        System.out.println("Total Salary: $" + calculateSalary());
    }
}

interface Benefits {
    double calculateBenefits();
}

class Manager extends Employee implements Benefits {
    double bonus;

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

    @Override
    double calculateSalary() {
        return salary + bonus;
    }

    @Override
    public double calculateBenefits() {
        return 5000;
    }

    void assignProject(String projectName) {
        System.out.println(name + " assigned to project: " +
    projectName);
    }

    void assignProject(String projectName, int teamMembers) {
        System.out.println(name + " assigned to project: " +
    projectName + " with " + teamMembers + " team members.");
    }
}

class Developer extends Employee implements Benefits {
    int experience;

    Developer(String name, double salary, int experience) {
        super(name, salary);
        this.experience = experience;
    }

    @Override
    double calculateSalary() {
        return salary + (experience * 500);
    }

    @Override
    public double calculateBenefits() {
        return experience * 200;
    }
}

// Main Class
public class EmployeeManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Employee> employees = new ArrayList<>();

        System.out.print("Enter number of employees: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        for (int i = 0; i < n; i++) {
            System.out.println("\nEnter details for Employee " + (i +
    1));
            System.out.print("Type (Manager/Developer): ");
            String type = scanner.nextLine();

            System.out.print("Name: ");
            String name = scanner.nextLine();
            System.out.print("Base Salary: ");
            double salary = scanner.nextDouble();

            if (type.equalsIgnoreCase("Manager")) {
                System.out.print("Bonus: ");
                double bonus = scanner.nextDouble();
                scanner.nextLine();
                Manager manager = new Manager(name, salary, bonus);
                employees.add(manager);

                System.out.print("Enter project name: ");
                String projectName = scanner.nextLine();
                System.out.print("Enter number of team members (or 0 to
    skip): ");
                int teamMembers = scanner.nextInt();
                scanner.nextLine();

                if (teamMembers > 0) {
                    manager.assignProject(projectName, teamMembers);
                } else {
                    manager.assignProject(projectName);
                }

            } else if (type.equalsIgnoreCase("Developer")) {
                System.out.print("Experience (years): ");
                int experience = scanner.nextInt();
                scanner.nextLine();
                employees.add(new Developer(name, salary, experience));
            } else {
                System.out.println("Invalid Employee Type! Skipping..."
    );
            }
        }

        System.out.println("\n--- Employee Details ---");
        for (Employee emp : employees) {
            emp.display();
            if (emp instanceof Benefits) {
                System.out.println("Benefits: $" + ((Benefits) emp).
    calculateBenefits());
            }
            System.out.println("---------------------------");
        }

        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter number of employees: 2

Enter details for Employee 1
Type (Manager/Developer): Manager
Name: Milan
Base Salary: 60000
Bonus: 4000
Enter project name: Gitlab
Enter number of team members (or 0 to skip): 7
Milan assigned to project: Gitlab with 7 team members.

Enter details for Employee 2
Type (Manager/Developer): Developer
Name: David
Base Salary: 70000
Experience (years): 4

--- Employee Details ---
Name: Milan
Base Salary: $60000.0
Total Salary: $64000.0
Benefits: $5000.0
---------------------------
Name: David
```

```
Base Salary: $70000.0
Total Salary: $72000.0
Benefits: $800.0
---------------------------
```

Experiment No: 14 **Date: 03/03/2025**

Graphics Package for Geometric Figures

Aim

Write a Java program to store employee details including employee number, name, and salary, and search for an employee by employee number.

Algorithm

```
1. Start
2. Create a package ‘graphics‘ containing:
   - An interface ‘Shape‘ with an ‘area()‘ method.
   - Classes ‘Rectangle‘, ‘Triangle‘, ‘Square‘, ‘Circle‘ implementing ‘Shape‘:
     - Each class has a constructor to initialize dimensions.
     - Implement the ‘area()‘ method to return the area of the respective shape.
3. Create a ‘Main‘ class:
   - Import ‘graphics‘ package.
   - Use ‘Scanner‘ to take user input for shape dimensions.
   - Create objects for ‘Rectangle‘, ‘Square‘, ‘Triangle‘, and ‘Circle‘.
   - Call the ‘area()‘ method for each shape and display the result.
4. Stop
```

Source Code

graphics/Shape.java

```java
package graphics;

public interface Shape {
    double area();
}
```

graphics/Rectangle.java

```java
package graphics;

public class Rectangle implements Shape {
    double length, width;

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

    @Override
    public double area() {
        return length * width;
    }
}
```

graphics/Square.java

```java
package graphics;

public class Square implements Shape {
    double side;

    public Square(double side) {
        this.side = side;
    }

    @Override
    public double area() {
        return side * side;
    }
}
```

graphics/Triangle.java

```java
package graphics;

public class Triangle implements Shape {
    double base, height;

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

    @Override
    public double area() {
        return 0.5 * base * height;
    }
}
```

graphics/Circle.java

```java


package graphics;

public class Circle implements Shape {
    double radius;
    final double PI = 3.14159;

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

    @Override
    public double area() {
        return PI * radius * radius;
    }
}
```

Main.java

```java
import graphics.*;
import java.util.Scanner;

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

        // Rectangle
        System.out.print("Enter length and width of the rectangle: ");
        double length = scanner.nextDouble();
        double width = scanner.nextDouble();
        Shape rectangle = new Rectangle(length, width);
        System.out.println("Area of Rectangle: " + rectangle.area());

        // Square
        System.out.print("\nEnter side length of the square: ");
        double side = scanner.nextDouble();
        Shape square = new Square(side);
        System.out.println("Area of Square: " + square.area());

        // Triangle
        System.out.print("\nEnter base and height of the triangle: ");
        double base = scanner.nextDouble();
        double height = scanner.nextDouble();
        Shape triangle = new Triangle(base, height);
        System.out.println("Area of Triangle: " + triangle.area());

        // Circle
        System.out.print("\nEnter radius of the circle: ");
        double radius = scanner.nextDouble();
        Shape circle = new Circle(radius);
        System.out.println("Area of Circle: " + circle.area());

        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter length and width of the rectangle: 5 6
Area of Rectangle: 30.0

Enter side length of the square: 4
Area of Square: 16.0

Enter base and height of the triangle: 3
6
Area of Triangle: 9.0

Enter radius of the circle: 7
Area of Circle: 153.93791
```

Experiment No: 8 **Date: 17/03/2025**

File Operations in Java

Aim

Write a program that performs various file operations such as reading, writing, and appending data to a file

Algorithm

```
1. Start
2. Initialize ‘sourceFile‘ (file to be copied)
3. Initialize ‘destinationFile‘ (file to store copied content)
4. Open ‘sourceFile‘ in read mode using ‘FileReader‘ or ‘BufferedReader‘
5. Open ‘destinationFile‘ in write mode using ‘FileWriter‘ or ‘BufferedWriter‘
6. Read contents from ‘sourceFile‘ line by line or character by character
7. Write each read line/character into ‘destinationFile‘
8. Close both files to save resources
9. Display "File copied successfully."
10. Handle exceptions (e.g., ‘FileNotFoundException‘, ‘IOException‘)
11. End
```

Source Code

```java
import java.io.*;
import java.util.Scanner;

public class FileOperations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String filename = "sample.txt";

        while (true) {
            System.out.println("\nFile Operations Menu:");
            System.out.println("1. Write to File");
            System.out.println("2. Read from File");
            System.out.println("3. Append to File");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    writeToFile(filename, scanner);
                    break;
                case 2:
                    readFromFile(filename);
                    break;
                case 3:
                    appendToFile(filename, scanner);
                    break;
                case 4:
                    System.out.println("Exiting program...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice! Try again.");
            }
        }
    }

    // Method to write to file
    public static void writeToFile(String filename, Scanner scanner) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
            System.out.print("Enter text to write to file: ");
            String content = scanner.nextLine();
            writer.write(content);
            writer.newLine();
            System.out.println("Data written to file successfully.");
        } catch (IOException e) {
            System.out.println("Error writing to file: " + e.getMessage());
        }
    }

    // Method to read from file
    public static void readFromFile(String filename) {
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            System.out.println("\nContents of the file:");
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }

    // Method to append to file
    public static void appendToFile(String filename, Scanner scanner) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename, true))) {
            System.out.print("Enter text to append to file: ");
            String content = scanner.nextLine();
            writer.write(content);
            writer.newLine();
            System.out.println("Data appended to file successfully.");
        } catch (IOException e) {
            System.out.println("Error appending to file: " + e.getMessage());
        }
    }
}
```

Result

The program was executed successfully.

```
File Operations Menu:
1. Write to File
2. Read from File
3. Append to File
4. Exit
Choose an option: 1
Enter text to write to file: The miracle is not that we do this work,but that we are h
Data written to file successfully.

File Operations Menu:
1. Write to File
2. Read from File
3. Append to File
4. Exit
Choose an option: 2

Contents of the file:
The miracle is not that we do this work,but that we are happy to do it

File Operations Menu:
1. Write to File
2. Read from File
3. Append to File
4. Exit
Choose an option: 3
Enter text to append to file: by MOTHER TERESA
Data appended to file successfully.

File Operations Menu:
1. Write to File
2. Read from File
3. Append to File
4. Exit
Choose an option: 2

Contents of the file:
The miracle is not that we do this work,but that we are happy to do it
by MOTHER TERESA

File Operations Menu:
1. Write to File
2. Read from File
3. Append to File
4. Exit
Choose an option: 4
```

```
Exiting program...
```

Experiment No: 16 **Date: 17/02/2025**

System-Defined and User-Defined Exception for Authentication

Aim

Write a Java program that demonstrates both system-defined exceptions (such as FileNot- FoundException and IOException) and user-defined exceptions for authentication fail- ures. Implement a readFile(String filename) method that attempts to read a file and prints its contents while handling FileNotFoundException if the file does not exist and IOException for other input/output errors. Define a custom exception class AuthenticationException that extends Exception and create an authenticate(String username, String password) method to validate user credentials against predefined values (e.g., "admin" with password "admin123"), throwing an AuthenticationException if authentication fails. In the main method, prompt the user to enter a filename, attempt to read the file, then request login credentials, invoking authenticate() and handling exceptions using try-catch blocks to display appropriate error messages, ensuring meaningful feedback to the user.

Algorithm

```
1. Start
2. Initialize Scanner for user input
3. Prompt user to enter filename
4. Call ‘readFile(filename)‘ method:
   - Try to open file using ‘BufferedReader‘
   - Read and print file contents line by line
   - Handle ‘FileNotFoundException‘ if file doesn’t exist
   - Handle ‘IOException‘ for other read errors
5. Prompt user to enter username and password
6. Call ‘authenticate(username, password)‘ method:
   - Compare input with predefined credentials ("admin", "admin123")
   - If incorrect, throw ‘AuthenticationException‘
   - Catch exception and display error message if authentication fails
7. If authentication succeeds, display welcome message
8. Close Scanner
9. End
```

Source Code

```java
import java.io.*;
import java.util.Scanner;

// Custom Exception for Authentication Failure
class AuthenticationException extends Exception {
    public AuthenticationException(String message) {
        super(message);
    }
}

```

```java
public class AuthenticationSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Step 1: File Handling
        System.out.print("Enter filename to read: ");
        String filename = scanner.nextLine();
        readFile(filename);

        // Step 2: Authentication
        System.out.print("\nEnter username: ");
        String username = scanner.nextLine();
        System.out.print("Enter password: ");
        String password = scanner.nextLine();

        try {
            authenticate(username, password);
            System.out.println("Authentication successful! Welcome, " +
    username + ".");
        } catch (AuthenticationException e) {
            System.out.println("Authentication failed: " + e.getMessage
    ());
        }

        scanner.close();
    }

    // Method to read a file and handle exceptions
    public static void readFile(String filename) {
        try (BufferedReader reader = new BufferedReader(new FileReader(
    filename))) {
            System.out.println("\nContents of " + filename + ":");
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found - " + filename);
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage())
    ;
        }
    }

    // Method to authenticate user
    public static void authenticate(String username, String password)
    throws AuthenticationException {
        String correctUsername = "admin";
        String correctPassword = "admin123";

        if (!username.equals(correctUsername) || !password.equals(
    correctPassword)) {
            throw new AuthenticationException("Invalid username or
    password.");
        }
    }
}
```

Result

The program was executed successfully.

```
Enter filename to read: file.txt

Contents of file.txt:
The miracle is not that we do this work,but that we are happy to do it
by MOTHER TERESA

Enter username: admin
Enter password: admin123
Authentication successful! Welcome, admin.
```

Experiment No: 17 **Date:17/03/2025**

Multithreading

Aim

Write a Java program that defines two classes: one for generating and displaying the mul- tiplication table of 5 and another for printing the first N prime numbers. Implement both classes using multithreading, demonstrating both approaches—by extending the Thread class and implementing the Runnable interface. Ensure proper thread management and synchronization if needed.

Algorithm

```
1. Start
2. Initialize Scanner for user input
3. Prompt user to enter the number of prime numbers to generate
4. Read user input and store it in ‘n‘
5. Create a thread ‘tableThread‘ by extending ‘Thread‘ to generate the multiplication
   - Print multiplication table from 1 to 10
   - Sleep for 500ms after each multiplication
6. Create a thread ‘primeThread‘ by implementing ‘Runnable‘ to generate first ‘n‘ pri
   - Check if each number is prime
   - Print prime numbers sequentially
   - Sleep for 300ms after each prime number
7. Start both threads using ‘start()‘ method
8. Use ‘join()‘ to ensure both threads complete execution before continuing
9. Print message indicating both threads have finished execution
10. Close Scanner
11. End
```

Source Code

```java
import java.util.Scanner;

// Thread using "extends Thread" for multiplication table of 5
class MultiplicationTable extends Thread {
    public void run() {
        System.out.println("\nMultiplication Table of 5:");
        for (int i = 1; i <= 10; i++) {
            System.out.println("5 x " + i + " = " + (5 * i));
            try {
                Thread.sleep(500); // Sleep for 500ms to simulate processing
            } catch (InterruptedException e) {
                System.out.println("Multiplication thread interrupted.");
            }
        }
    }
}
```

```java

// Thread using "implements Runnable" for generating prime numbers
class PrimeNumbers implements Runnable {
    private int count;

    PrimeNumbers(int count) {
        this.count = count;
    }

    public void run() {
        System.out.println("\nFirst " + count + " Prime Numbers:");
        int num = 2, primeCount = 0;
        while (primeCount < count) {
            if (isPrime(num)) {
                System.out.print(num + " ");
                primeCount++;
                try {
                    Thread.sleep(300); // Sleep for 300ms to simulate processing
                } catch (InterruptedException e) {
                    System.out.println("Prime thread interrupted.");
                }
            }
            num++;
        }
        System.out.println();
    }

    private boolean isPrime(int num) {
        if (num < 2) return false;
        for (int i = 2; i * i <= num; i++) {
            if (num % i == 0) return false;
        }
        return true;
    }
}

// Main class to start both threads
public class MultiThreadingExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Getting user input for prime numbers
        System.out.print("Enter the number of prime numbers to generate: ");
        int n = scanner.nextInt();

        // Creating thread objects
        MultiplicationTable tableThread = new MultiplicationTable();
        Thread primeThread = new Thread(new PrimeNumbers(n));

        // Starting both threads
        tableThread.start();
        primeThread.start();

        // Ensuring both threads complete execution before continuing
        try {
            tableThread.join();
            primeThread.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }

        System.out.println("\nBoth threads have finished execution.");
        scanner.close();
    }
}
```

Result

The program was executed successfully.

```
Enter the number of prime numbers to generate: 12

Multiplication Table of 5:

First 12 Prime Numbers:
5 x 1 = 5
2 3 5 x 2 = 10
5 7 5 x 3 = 15
11 5 x 4 = 20
13 17 5 x 5 = 25
19 23 5 x 6 = 30
29 5 x 7 = 35
31 37 5 x 8 = 40

5 x 9 = 45
5 x 10 = 50

Both threads have finished execution.
```