

# Basic Java assignment

## Basic Java:

1) Write a program to calculate the area of a circle, rectangle, or triangle based on user input.

```
import java.util.Scanner;

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

        System.out.println("Choose a shape to calculate the area:");
        System.out.println("1. Circle");
        System.out.println("2. Rectangle");
        System.out.println("3. Triangle");

        int choice = scanner.nextInt();
        double area = 0;

        if (choice == 1) {
            System.out.print("Enter the radius of the circle: ");
            double radius = scanner.nextDouble();
            area = Math.PI * radius * radius;
        } else if (choice == 2) {
            System.out.print("Enter the length of the rectangle: ");
            double length = scanner.nextDouble();
            System.out.print("Enter the width of the rectangle: ");
            double width = scanner.nextDouble();
            area = length * width;
        } else if (choice == 3) {
            System.out.print("Enter the base of the triangle: ");
            double base = scanner.nextDouble();
            System.out.print("Enter the height of the triangle: ");
            double height = scanner.nextDouble();
            area = 0.5 * base * height;
        } else {
            System.out.println("Invalid choice! Please choose a valid option.");
            scanner.close();
            return;
        }
    }
}
```

```
        System.out.println("The calculated area is: " + area);  
        scanner.close();  
    }  
}
```

## OUTPUT:

```
Choose a shape to calculate the area:  
1. Circle  
2. Rectangle  
3. Triangle  
1  
Enter the radius of the circle: 4  
The calculated area is: 50.26548245743669  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

2) Create a program to check if a number is even or odd.

```
import java.util.Scanner;  
  
public class EvenOddChecker {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        System.out.print("Enter a number: ");  
        int number = scanner.nextInt();  
  
        if (number % 2 == 0) {  
            System.out.println(number + " is an even number.");  
        } else {  
            System.out.println(number + " is an odd number.");  
        }  
  
        scanner.close();  
    }  
}
```

## OUTPUT:

```
Enter a number: 4  
4 is an even number.  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

3) Implement a program to find the factorial of a given number.

```
import java.util.Scanner;

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

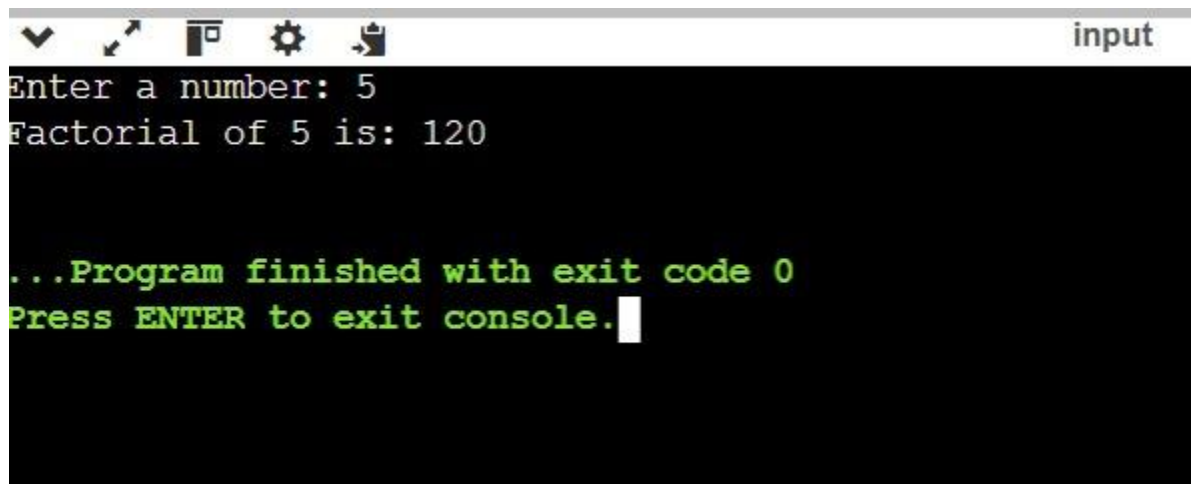
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

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

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

        scanner.close();
    }
}
```

OUTPUT:



The screenshot shows a Java IDE window titled "input". The console output displays the program's execution: it prompts for a number, receives "5", and outputs "Factorial of 5 is: 120". Below this, a green message states "...Program finished with exit code 0" and "Press ENTER to exit console." with a cursor.

```
Enter a number: 5
Factorial of 5 is: 120

...Program finished with exit code 0
Press ENTER to exit console.
```

4) Write a program to print the Fibonacci sequence up to a specified number.

```

import java.util.Scanner;

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

        System.out.print("Enter the number of terms: ");
        int n = scanner.nextInt();

        int first = 0, second = 1;

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

        for (int i = 1; i <= n; i++) {
            System.out.print(first + " ");
            int next = first + second;
            first = second;
            second = next;
        }

        scanner.close();
    }
}

```

OUTPUT:

```

Enter the number of terms: 5
Fibonacci Sequence up to 5 terms:
0 1 1 2 3

...Program finished with exit code 0
Press ENTER to exit console.

```

5) Use loops to print patterns like a triangle or square.

```

import java.util.Scanner;

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

        System.out.println("Choose a pattern to print:");
        System.out.println("1. Triangle");
        System.out.println("2. Square");

        int choice = scanner.nextInt();

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

        if (choice == 1) {
            // Triangle Pattern
            for (int i = 1; i <= rows; i++) {
                for (int j = 1; j <= rows - i; j++) {
                    System.out.print(" ");
                }
                for (int k = 1; k <= (2 * i - 1); k++) {
                    System.out.print("*");
                }
                System.out.println();
            }
        } else if (choice == 2) {
            // Square Pattern
            for (int i = 1; i <= rows; i++) {
                for (int j = 1; j <= rows; j++) {
                    System.out.print("* ");
                }
                System.out.println();
            }
        } else {
            System.out.println("Please choose a valid option.");
        }

        scanner.close();
    }
}

```

OUTPUT:

```
Choose a pattern to print:
1. Triangle
2. Square
1
Enter the number of rows: 8
  *
 ***
*****
*****
*****
*****
*****
*****
*****
*****

...Program finished with exit code 0
Press ENTER to exit console.
```

## Data Types and Operators:

1) Explain the difference between primitive and reference data types with examples.

ANSWER:

Primitive Data Types (Basic Types)

## Definition:

These are built-in data types that store single values directly in memory.

They are not objects and operate at a lower level for better performance.

## Example:

```
public class PrimitiveExample {  
    public static void main(String[] args) {  
        int age = 25;  
        double price = 99.99;  
        boolean isAvailable = true;  
  
        System.out.println("Age: " + age);  
        System.out.println("Price: " + price);  
        System.out.println("Available: " + isAvailable);  
    }  
}
```

## Reference Data Types (Objects & Arrays)

### Definition:

These store memory addresses (references) to objects, rather than storing values directly.

Includes arrays, strings, objects, collections, etc.

Example:

```
class Student {  
    String name;  
    int rollNumber;  
  
    Student(String name, int rollNumber) {  
        this.name = name;  
        this.rollNumber = rollNumber;  
    }  
}  
  
public class ReferenceExample {  
    public static void main(String[] args) {  
        Student s1 = new Student("Tamim", 101);  
        System.out.println("Student Name: " + s1.name);  
        System.out.println("Roll Number: " + s1.rollNumber);  
    }  
}
```



```
}
```

2) Write a program to demonstrate the use of arithmetic, logical, and relational operators.

```
import java.util.Scanner;

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

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

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

        // Arithmetic Operators
        System.out.println("Arithmetic Operations:");
        System.out.println("Addition: " + (num1 + num2));
        System.out.println("Subtraction: " + (num1 - num2));
        System.out.println("Multiplication: " + (num1 * num2));
        System.out.println("Division: " + (num1 / num2));
        System.out.println("Modulus: " + (num1 % num2));

        // Relational Operators
        System.out.println("\nRelational Operations:");
        System.out.println(num1 + " == " + num2 + " : " + (num1 == num2));
        System.out.println(num1 + " != " + num2 + " : " + (num1 != num2));
        System.out.println(num1 + " > " + num2 + " : " + (num1 > num2));
        System.out.println(num1 + " < " + num2 + " : " + (num1 < num2));
        System.out.println(num1 + " >= " + num2 + " : " + (num1 >= num2));
        System.out.println(num1 + " <= " + num2 + " : " + (num1 <= num2));

        // Logical Operators
        System.out.println("\nLogical Operations:");
        boolean condition1 = (num1 > 0 && num2 > 0);
        boolean condition2 = (num1 < 0 || num2 < 0);

        System.out.println("Both numbers are positive: " + condition1);
        System.out.println("At least one number is negative: " + condition2);
        System.out.println("Negation of condition1: " + !condition1);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter first number: 4
Enter second number: 4
Arithmetic Operations:
Addition: 8
Subtraction: 0
Multiplication: 16
Division: 1
Modulus: 0

Relational Operations:
4 == 4 : true
4 != 4 : false
4 > 4 : false
4 < 4 : false
4 >= 4 : true
4 <= 4 : true

Logical Operations:
Both numbers are positive: true
At least one number is negative: false
Negation of condition1: false

...Program finished with exit code 0
Press ENTER to exit console.
```

3) Create a program to convert a temperature from Celsius to Fahrenheit and vice versa.

```
import java.util.Scanner;

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

        System.out.println("Choose conversion:");
        System.out.println("1. Celsius to Fahrenheit");
        System.out.println("2. Fahrenheit to Celsius");

        int choice = scanner.nextInt();

        if (choice == 1) {
            System.out.print("Enter temperature in Celsius: ");
            double celsius = scanner.nextDouble();
            double fahrenheit = (celsius * 9/5) + 32;
            System.out.println("Temperature in Fahrenheit: " + fahrenheit);
        } else if (choice == 2) {
            System.out.print("Enter temperature in Fahrenheit: ");
            double fahrenheit = scanner.nextDouble();
            double celsius = (fahrenheit - 32) * 5/9;
            System.out.println("Temperature in Celsius: " + celsius);
        } else {
            System.out.println("Invalid choice! Please select 1 or 2.");
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Choose conversion:
1. Celsius to Fahrenheit
2. Fahrenheit to Celsius
1
Enter temperature in Celsius: 100
Temperature in Fahrenheit: 212.0

...Program finished with exit code 0
Press ENTER to exit console.
```

## Control Flow Statements:

1) Write a program to check if a given number is prime using an if-else statement.

```
import java.util.Scanner;

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

        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        boolean isPrime = true;

        if (number <= 1) {
            isPrime = false;
        } else {
            for (int i = 2; i <= number / 2; i++) {
                if (number % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }

        if (isPrime) {
            System.out.println(number + " is a prime number.");
        } else {
            System.out.println(number + " is not a prime number.");
        }

        scanner.close();
    }
}
```

## OUTPUT:

```
Enter a number: 11
11 is a prime number.

...Program finished with exit code 0
Press ENTER to exit console.
```

2) Implement a program to find the largest number among three given numbers using a conditional statement.

```
import java.util.Scanner;

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

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

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

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

        int largest;

        if (num1 >= num2 && num1 >= num3) {
            largest = num1;
        } else if (num2 >= num1 && num2 >= num3) {
            largest = num2;
        } else {
            largest = num3;
        }

        System.out.println("The largest number is: " + largest);
        scanner.close();
    }
}
```



## OUTPUT:

```
Enter first number: 5
Enter second number: 6
Enter third number: 9
The largest number is: 9

...Program finished with exit code 0
Press ENTER to exit console.
```

## 3) Use a for loop to print a multiplication table.

```
import java.util.Scanner;

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

        System.out.print("Enter a number to print its multiplication table: ");
        int number = scanner.nextInt();

        System.out.println("Multiplication Table for " + number + ":");
        for (int i = 1; i <= 10; i++) {
            System.out.println(number + " x " + i + " = " + (number * i));
        }

        scanner.close();
    }
}
```

## OUTPUT:

```
Enter a number to print its multiplication table: 5
Multiplication Table for 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

...Program finished with exit code 0
Press ENTER to exit console.[]
```

4) Create a program to calculate the sum of even numbers from 1 to 10 using a while loop.

```
public class SumEvenNumbers {
    public static void main(String[] args) {
        int sum = 0;
        int num = 2;

        while (num <= 10) {
            sum += num;
            num += 2;
        }

        System.out.println("The sum of even numbers from 1 to 10 is: " + sum);
    }
}
```

OUTPUT:

```
The sum of even numbers from 1 to 10 is: 30

...Program finished with exit code 0
Press ENTER to exit console.[]
```



## Arrays:

1) Write a program to find the average of elements in an array.

```
import java.util.Scanner;

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

        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        int[] numbers = new int[n];
        int sum = 0;

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            numbers[i] = scanner.nextInt();
            sum += numbers[i];
        }

        double average = (double) sum / n;
        System.out.println("The average is: " + average);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of elements: 5
Enter 5 elements:
4
2
5
7
6
The average is: 4.8

...Program finished with exit code 0
Press ENTER to exit console.□
```

2) Implement a function to sort an array in ascending order using bubble sort or selection sort.

```

import java.util.Scanner;

public class BubbleSort {

    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public static void printArray(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }

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

        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];

        System.out.println("Enter " + n + " numbers:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        bubbleSort(arr);

        System.out.println("Sorted array using Bubble Sort:");
        printArray(arr);

        scanner.close();
    }
}

```

OUTPUT:

```
Enter the number of elements: 5
Enter 5 numbers:
4
5
1
2
9
Sorted array using Bubble Sort:
1 2 4 5 9

...Program finished with exit code 0
Press ENTER to exit console.
```

3) Create a program to search for a specific element within an array using linear search.

```

import java.util.Scanner;

public class LinearSearch {

    public static int linearSearch(int[] arr, int key) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == key) {
                return i;
            }
        }
        return -1;
    }

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

        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];

        System.out.println("Enter " + n + " numbers:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.print("Enter the element to search: ");
        int key = scanner.nextInt();

        int result = linearSearch(arr, key);

        if (result != -1) {
            System.out.println("Element found at index: " + result);
        } else {
            System.out.println("Element not found in the array.");
        }

        scanner.close();
    }
}

```

OUTPUT:

```
Enter the number of elements: 4
Enter 4 numbers:
22
33
44
98
Enter the element to search: 98
Element found at index: 3

...Program finished with exit code 0
Press ENTER to exit console.□
```

## Object Oriented Programming (OOP):

Create a class to represent a student with attributes like name, roll number, and marks.

Implement inheritance to create a "GraduateStudent" class that extends the "Student" class with additional features.

Demonstrate polymorphism by creating methods with the same name but different parameters in a parent and child class.

```

class Student {
    private String name;
    private int rollNumber;
    private double marks;

    public Student(String name, int rollNumber, double marks) {
        this.name = name;
        this.rollNumber = rollNumber;
        this.marks = marks;
    }

    public void displayInfo() {
        System.out.println("Student Name: " + name);
        System.out.println("Roll Number: " + rollNumber);
        System.out.println("Marks: " + marks);
    }

    public void displayInfo(String message) {
        System.out.println(message);
        this.displayInfo();
    }
}

class GraduateStudent extends Student {
    private String researchTopic;

    public GraduateStudent(String name, int rollNumber, double marks, String researchTopic) {
        super(name, rollNumber, marks);
        this.researchTopic = researchTopic;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Research Topic: " + researchTopic);
    }
}

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

        Student student1 = new Student("Tamim", 101, 85.5);
        System.out.println("Student Details ");
        student1.displayInfo();

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

        GraduateStudent gradStudent = new GraduateStudent("Ali", 201, 90.2, "Artificial Intelligence");
        gradStudent.displayInfo();

        System.out.println("\n Overloaded Method Demo ");
        student1.displayInfo("Student Information:");
    }
}

```

OUTPUT:



```
Student Details
Student Name: Tamim
Roll Number: 101
Marks: 85.5

Graduate Student Details
Student Name: Ali
Roll Number: 201
Marks: 90.2
Research Topic: Artificial Intelligence

Overloaded Method Demo
Student Information:
Student Name: Tamim
Roll Number: 101
Marks: 85.5

...Program finished with exit code 0
Press ENTER to exit console.█
```

Explain the concept of encapsulation with a suitable example.

Encapsulation is one of the fundamental principles of Object-Oriented Programming (OOP). It refers to wrapping data (variables) and code (methods) together into a single unit and restricting direct access to the data to protect it from unintended modification.



## Key Features of Encapsulation

Data Hiding → The internal data of a class is hidden from direct access.

Controlled Access → Getter and Setter methods provide controlled access to data.

Improved Security → Prevents unauthorized modification of data.

Better Maintainability → Changes in implementation do not affect external code.

EXAMPLE:

```
class Person {  
    private String name;  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
}  
  
public class EncapsulationExample {  
    public static void main(String[] args) {  
        Person p = new Person();  
        p.setName("Tamim");  
        System.out.println("Name: " + p.getName());  
    }  
}
```

Name: Tamim

...Program finished with exit code 0  
Press ENTER to exit console.

OUTPUT:

## String Manipulation:

Write a program to reverse a given string.

```
import java.util.Scanner;

public class ReverseStringBuilder {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();

        String reversedString = new StringBuilder(input).reverse().toString();
        System.out.println("Reversed String: " + reversedString);

        scanner.close();
    }
}
```

OUTPUT:

```
Enter a string: abad
Reversed String: daba

...Program finished with exit code 0
Press ENTER to exit console.
```

Implement a function to count the number of vowels in a string.

```

import java.util.Scanner;

public class VowelCounter {

    public static int countVowels(String str) {
        int count = 0;
        str = str.toLowerCase();

        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
                count++;
            }
        }
        return count;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();

        int vowelCount = countVowels(input);
        System.out.println("Number of vowels: " + vowelCount);

        scanner.close();
    }
}

```

OUTPUT:

```

Enter a string: aeiou
Number of vowels: 5

...Program finished with exit code 0
Press ENTER to exit console.

```

Create a program to check if two strings are anagrams.

```

import java.util.Scanner;

public class AnagramCheckerOptimized {

    public static boolean areAnagrams(String str1, String str2) {

        str1 = str1.toLowerCase();
        str2 = str2.toLowerCase();

        if (str1.length() != str2.length()) {
            return false;
        }

        int[] charCount = new int[26];

        for (int i = 0; i < str1.length(); i++) {
            charCount[str1.charAt(i) - 'a']++;
            charCount[str2.charAt(i) - 'a']--;
        }

        for (int count : charCount) {
            if (count != 0) {
                return false;
            }
        }

        return true;
    }

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

        System.out.print("Enter the first string: ");
        String str1 = scanner.nextLine();

        System.out.print("Enter the second string: ");
        String str2 = scanner.nextLine();

        if (areAnagrams(str1, str2)) {
            System.out.println("The strings are anagrams.");
        } else {
            System.out.println("The strings are not anagrams.");
        }

        scanner.close();
    }
}

```

OUTPUT:

```
Enter the first string: silent
Enter the second string: listen
The strings are anagrams.

...Program finished with exit code 0
Press ENTER to exit console.
```

## Advanced Topics:

Explain the concept of interfaces and abstract classes with examples.

### Abstract Class

An abstract class is a class that cannot be instantiated and may contain both abstract (unimplemented) and concrete (implemented) methods.

It is used when we want to share some common behavior among multiple related classes.

Supports constructors, instance variables, and method implementations.

Example :

```
abstract class Animal {  
    String name;  
  
    public Animal(String name) {  
        this.name = name;  
    }  
  
    abstract void makeSound();  
  
    public void sleep() {  
        System.out.println(name + " is sleeping.");  
    }  
}  
  
class Dog extends Animal {  
    public Dog(String name) {  
        super(name);  
    }  
  
    public void makeSound() {  
        System.out.println(name + " barks!");  
    }  
}
```

```
}
```

```
class Cat extends Animal {  
    public Cat(String name) {  
        super(name);  
    }  
  
    public void makeSound() {  
        System.out.println(name + " meows!");  
    }  
}
```

```
public class AbstractClassExample {  
    public static void main(String[] args) {  
        Dog dog = new Dog("Buddy");  
        dog.makeSound();  
        dog.sleep();  
  
        Cat cat = new Cat("Whiskers");  
        cat.makeSound();  
        cat.sleep();  
    }  
}
```



```
}  
}
```

## Interface

An interface is a blueprint for a class that only contains abstract methods (until Java 7).

By default, all methods in an interface are public and abstract.

A class can implement multiple interfaces, unlike abstract classes (which support only single inheritance).

Since Java 8, interfaces can have default methods (with implementation) and static methods.

Example:

```
interface Vehicle {  
    void start();  
}
```

```
]class Car implements Vehicle {  
    public void start() {  
        System.out.println("Car is starting with a key.");  
    }  
}
```

```
    }  
}
```

```
class Bike implements Vehicle {  
    public void start() {  
        System.out.println("Bike is starting with a self-start  
button.");  
    }  
}
```

```
public class InterfaceExample {  
    public static void main(String[] args) {  
        Vehicle myCar = new Car();  
        myCar.start();  
        Vehicle myBike = new Bike();  
        myBike.start();  
    }  
}
```

Create a program to handle exceptions using try-catch blocks.

```

import java.util.Scanner;

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

        try {
            System.out.print("Enter numerator: ");
            int numerator = scanner.nextInt();

            System.out.print("Enter denominator: ");
            int denominator = scanner.nextInt();

            int result = numerator / denominator;

            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed!");
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());
        } finally {
            System.out.println("Execution completed.");
            scanner.close();
        }
    }
}

```

OUTPUT:

```

Enter numerator: 5
Enter denominator: 0
Error: Division by zero is not allowed!
Execution completed.

...Program finished with exit code 0
Press ENTER to exit console.

```

Implement a simple file I/O operation to read data from a text file

```

public class FileIOExample {

    public static void writeToFile(String fileName, String data) {
        try {
            FileWriter writer = new FileWriter(fileName);
            writer.write(data);
            writer.close();
            System.out.println("Data successfully written to " + fileName);
        } catch (IOException e) {
            System.out.println("Error writing to file: " + e.getMessage());
        }
    }

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

    public static void main(String[] args) {
        String fileName = "sample.txt";
        String data = "Hello, this is a simple File I/O example in Java!";

        writeToFile(fileName, data);
        readFromFile(fileName);
    }
}

```

OUTPUT:

```

Data successfully written to sample.txt

Contents of the file:
Hello, this is a simple File I/O example in Java!

...Program finished with exit code 0
Press ENTER to exit console.

```

Explore multithreading in Java to perform multiple tasks concurrently.

## Multithreading in Java

Multithreading allows a Java program to execute multiple tasks simultaneously, improving performance and efficiency.

### Extending the Thread class

Implementing the Runnable interface (Recommended for better flexibility)

Example:

```
class MyRunnable implements Runnable {  
    public void run() {  
        for (int i = 1; i <= 5; i++) {  
            System.out.println(Thread.currentThread().getName() +  
                " - Count: " + i);  
            try {  
                Thread.sleep(500);  
            } catch (InterruptedException e) {  
                System.out.println("Thread interrupted: " +  
                    e.getMessage());  
            }  
        }  
    }  
}
```

```
}  
}
```

```
public class RunnableExample {  
    public static void main(String[] args) {  
        Thread t1 = new Thread(new MyRunnable(), "Worker-1");  
        Thread t2 = new Thread(new MyRunnable(), "Worker-2");  
  
        t1.start();  
        t2.start();  
    }  
}
```