# Program 1

**AIM**: Write a program that converts temperature from Fahrenheit to Celsius using an IDE. (Formula Celsius temp = (Fahrenheit-32)* 5/9).

## PROCEDURE:

1. Open your JAVA IDE.
2. Create a new Java project and class.
3. Write the conversion code.
4. Compile and run the program.
5. Observe the output.

## CODE:

```
class Conversion {
   public static void main(String args[]) {
      double celsius = 37.0;
      double fahrenheit = (celsius * 9/5) + 32;
      System.out.println("Celsius: " + celsius);
      System.out.println("Fahrenheit: " + fahrenheit);
   }
}
```

## Output:-

```
Celsius: 37.0
Fahrenheit: 98.6
```

**RESULT:** The java program of conversion has been successfully executed.

# Program 2

**AIM**: Write a java program to accept 10 integer values from the user, store them in an array,

1. Arrange the array in ascending and descending order,
2. Find the Maximum, minimum and average.
3. Print only either Odd or Even

## PROCEDURE:

1. Create an integer array of size 10.
2. Accept 10 integer values from the user and store them in the array.
3. Sort the array using `Arrays.sort()` method to arrange elements in ascending order.
4. Print the array in ascending order.
5. Traverse the sorted array in reverse order to display it in descending order.
6. Find the maximum element (last index of sorted array).
7. Find the minimum element (first index of sorted array).
8. Compute the average by calculating the sum of all elements and dividing by 10.
9. Ask the user whether to print Odd numbers or Even numbers.
10. If the user selects Odd, traverse the array and print only odd numbers.
11. If the user selects Even, traverse the array and print only even numbers.

## CODE:

```
import java.util.Scanner;
import java.util.Arrays;

public class ArrayOperations {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int[] arr = new int[10];

      // Accept 10 integers
      System.out.println("Enter 10 integer values:");
      for (int i = 0; i < 10; i++) {
         arr[i] = sc.nextInt();
      }

      // Sort array in ascending order
      Arrays.sort(arr);

      System.out.println("\nArray in Ascending Order:");
      for (int i : arr) {
         System.out.print(i + " ");
      }
```

```
        // Print array in descending order
        System.out.println("\nArray in Descending Order:");
        for (int i = arr.length - 1; i >= 0; i--) {
            System.out.print(arr[i] + " ");
        }

        // Find max, min, and average
        int max = arr[arr.length - 1];
        int min = arr[0];
        double sum = 0;
        for (int i : arr) {
            sum += i;
        }
        double avg = sum / arr.length;

        System.out.println("\n\nMaximum Value: " + max);
        System.out.println("Minimum Value: " + min);
        System.out.println("Average Value: " + avg);

        // Print only odd or even
        System.out.print("\nDo you want to print (O)dd or (E)ven numbers? ");
        char choice = sc.next().charAt(0);

        if (choice == 'O' || choice == 'o') {
            System.out.println("Odd Numbers:");
            for (int i : arr) {
                if (i % 2 != 0) {
                    System.out.print(i + " ");
                }
            }
        } else if (choice == 'E' || choice == 'e') {
            System.out.println("Even Numbers:");
            for (int i : arr) {
                if (i % 2 == 0) {
                    System.out.print(i + " ");
                }
            }
        } else {
            System.out.println("Invalid choice!");
        }

        sc.close();
    }
}
```

**Output:-**

```
Enter 10 integer values:
12 5 7 9 3 20 15 8 2 6

Array in Ascending Order:
2 3 5 6 7 8 9 12 15 20
Array in Descending Order:
20 15 12 9 8 7 6 5 3 2

Maximum Value: 20
Minimum Value: 2
Average Value: 8.7

Do you want to print (O)dd or (E)ven numbers? O
Odd Numbers:
3 5 7 9 15
```

**RESULT:** The program executed successfully without errors. Hence, the aim of this practical is achieved.

# Program 3

**AIM**: To implement the fundamental stack operations (push, pop, peek, and display) using an array in Java.

## PROCEDURE:

1. Create a stack class with push, pop, and display methods.
2. Use an array to store stack elements.
3. Implement stack overflow and underflow checks.
4. Call methods in main() and display results.
5. Compile and run the program.

## CODE:

```
class Stack {
   int top;
   int arr[] = new int[5];

   Stack() {
      top = -1;
   }

   void push(int x) {
      if(top == 4) {
         System.out.println("Stack Overflow");
      } else {
         arr[++top] = x;
         System.out.println(x + " pushed into stack");
      }
   }

   void pop() {
      if(top == -1) {
         System.out.println("Stack Underflow");
      } else {
         System.out.println(arr[top--] + " popped from stack");
      }
   }

   void display() {
      if(top == -1) {
         System.out.println("Stack is empty");
      } else {
         System.out.print("Stack elements: ");
```

```
            for(int i=0; i<=top; i++) {
               System.out.print(arr[i] + " ");
            }
            System.out.println();
         }
      }

      public static void main(String args[]) {
         Stack s = new Stack();
         s.push(10);
         s.push(20);
         s.display();
         s.pop();
         s.display();
      }
}
```

## Output:-

```
10 pushed into stack
20 pushed into stack
Stack elements: 10 20
20 popped from stack
Stack elements: 10
```

**RESULT:** The program executed successfully without errors. Hence, the aim of this practical is achieved.

# Program 4

**AIM**: To implement the basic Queue operations (enqueue, dequeue, peek, and display) using classes and objects in Java.

## PROCEDURE:

1. Create a queue class with enqueue, dequeue, and display methods.
2. Use an array to implement queue storage.
3. Handle queue overflow and underflow.
4. Execute operations inside main() and display results.
5. Compile and test the output.

## CODE:

```
class Queue {
   int front, rear, size;
   int arr[] = new int[5];

   Queue() {
      front = 0;
      rear = -1;
      size = 0;
   }

   void enqueue(int x) {
      if(size == 5) {
         System.out.println("Queue Overflow");
      } else {
         arr[++rear] = x;
         size++;
         System.out.println(x + " enqueued to queue");
      }
   }

   void dequeue() {
      if(size == 0) {
         System.out.println("Queue Underflow");
      } else {
         System.out.println(arr[front++] + " dequeued from queue");
         size--;
      }
   }

   void display() {
      if(size == 0) {
```

```
            System.out.println("Queue is empty");
        } else {
            System.out.print("Queue elements: ");
            for(int i=front; i<=rear; i++) {
                System.out.print(arr[i] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String args[]) {
        Queue q = new Queue();
        q.enqueue(10);
        q.enqueue(20);
        q.display();
        q.dequeue();
        q.display();
    }
}
```

## Output:-

```
10 enqueued to queue
20 enqueued to queue
Queue elements: 10 20
10 dequeued from queue
Queue elements: 20
```

**RESULT:** The program executed successfully without errors. Hence, the aim of this practical is achieved.

# Program 5

**AIM**: Write a java program to create a calculator. Use classes and methods to perform +,-,*,/,%.

## PROCEDURE:

1. Create a Calculator class with a main method. Import the java.util.Scanner class to read user input.
2. Use the Scanner object to prompt for and read the first number, the operator, and the second number.
3. Implement a switch statement that checks the operator and calls the appropriate method for the calculation.
4. Write a dedicated method for each operation (addition, subtraction, multiplication, and division) that takes two numbers as arguments and returns the result.
5. Display the outcome of the calculation on the console.

## CODE:

```
import java.util.Scanner;
class calculator{
	float n1, n2;
	Scanner sc = new Scanner(System.in);
	public void input()
	{
		System.out.print("Enter first number: ");
		n1 = sc.nextInt();
		System.out.print("Enter second number: ");
		n2 = sc.nextInt();
	}
	public void addition()
	{
		System.out.println(n1 + " + " + n2 + " = " + (n1+n2));
	}
	public void subtraction()
	{
		System.out.println(n1 + " - " + n2 + " = " + (n1-n2));
	}
	public void division()
	{
		System.out.println(n1 + " / " + n2 + " = " + (n1/n2));
	}
	public void multiplication()
	{
		System.out.println(n1 + " * " + n2 + " = " + (n1*n2));
	}
	public void modulus()
	{
```

```
		System.out.println(n1 + " % " + n2 + " = " + (n1%n2));
	}
}
public class Q6 {

	public static void main(String[] args) {
		int input;
		Scanner sc = new Scanner(System.in);
		calculator c = new calculator();

		while(true)
		{
			System.out.println("Choose one option to perform operation:-");
			System.out.println("1.Addition");
			System.out.println("2.Subtraction");
			System.out.println("3.Multiplication");
			System.out.println("4.Division");
			System.out.println("5.Modulus");
			System.out.println("6.Exit");
			input = sc.nextInt();

			switch(input)
			{
				case 1:
					c.input();
					c.addition();
					break;
				case 2:
					c.input();
					c.subtraction();
					break;
				case 3:
					c.input();
					c.multiplication();
					break;
				case 4:
					c.input();
					c.division();
					break;
				case 5:
					c.input();
					c.modulus();
					break;
				case 6:
					System.exit(0);
				default:
					System.out.println("Invalid Input");
					break;
			}
		}
```

```
        }
}
```

## Output:-

```
Choose one option to perform operation:-
1.Addition
2.Substraction
3.Multiplication
4.Division
5.Modulus
6.Exit
3
Enter first number: 3
Enter second number: 2
3.0 * 2.0 = 6.0
Choose one option to perform operation:-
1.Addition
2.Substraction
3.Multiplication
4.Division
5.Modulus
6.Exit
4
Enter first number: 15
Enter second number: 5
15.0 / 5.0 = 3.0
```

**RESULT:** The java program to create a calculator using classes and methods has been successfully executed.

# Program 6(A)

**AIM**: To write a Java program to check whether a given number is part of the Fibonacci series or not, and also print the Fibonacci series up to that number.

## PROCEDURE:

1. Start the program.
2. Accept a number from the user.
3. Initialize the first two Fibonacci numbers as 0 and 1.
4. Use a loop to generate Fibonacci numbers until the value becomes greater than or equal to the input number.
5. Print each Fibonacci number while generating.
6. If the input number is found in the generated series, declare it as part of the Fibonacci series.
7. If the input number is not found, declare it as not part of the Fibonacci series.
8. End the program.

## CODE:

```
import java.util.Scanner;

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

      // Input from user
      System.out.print("Enter a number: ");
      int num = sc.nextInt();

      int a = 0, b = 1, c;
      boolean isFibonacci = false;

      System.out.println("Fibonacci Series till " + num + ":");
      System.out.print(a + " " + b + " ");

      // Generate Fibonacci series
      while (b <= num) {
         c = a + b;
         a = b;
         b = c;

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

         if (b == num) {
            isFibonacci = true;
```

```
        }
    }

    // Check if number is part of Fibonacci series
    if (num == 0 || num == 1) {
        isFibonacci = true;
    }

    System.out.println(); // new line
    if (isFibonacci) {
        System.out.println(num + " is part of the Fibonacci series.");
    } else {
        System.out.println(num + " is NOT part of the Fibonacci series.");
    }

    sc.close();
  }
}
```

**Output:-**

```
Enter a number: 21
Fibonacci Series till 21:
0 1 1 2 3 5 8 13 21
21 is part of the Fibonacci series.

Enter a number: 22
Fibonacci Series till 22:
0 1 1 2 3 5 8 13 21 34
22 is NOT part of the Fibonacci series.
```

**RESULT:** Thus, the program to check if a given number is part of the Fibonacci series and print the series up to that number was implemented successfully.

# Program 6(B)

**AIM**: To solve Tower of Hanoi problem using recursion.

## PROCEDURE:

1. Define a recursive method to move disks.
2. Base case: If only one disk, move directly.
3. Recursive case: Move n-1 disks to auxiliary, then largest to destination.
4. Continue until all disks are moved.
5. Compile and run program with n=3.

## CODE:

```
class TowerOfHanoi {
   static void solve(int n, char source, char auxiliary, char destination) {
      if(n == 1) {
         System.out.println("Move disk 1 from " + source + " to " + destination);
         return;
      }
      solve(n-1, source, destination, auxiliary);
      System.out.println("Move disk " + n + " from " + source + " to " + destination);
      solve(n-1, auxiliary, source, destination);
   }

   public static void main(String args[]) {
      int n = 3;
      solve(n, 'A', 'B', 'C');
   }
}
```

## Output:-

```
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

**RESULT:** The program executed successfully without errors. Hence, the aim of this practical is achieved.

# Program 7

**AIM**: To implement a simple Linked List in Java using classes and objects.

## PROCEDURE:

1. Start the program.
2. Create a `Node` class that contains `data` and a reference to the next node.
3. Create a `LinkedListExample` class to handle operations such as insertion and display.
4. In the `main()` method, insert few elements into the linked list.
5. Display the linked list elements sequentially.
6. End the program.

## CODE:

```
class Node {

   int data;
   Node next;

   Node(int d) {
      data = d;
      next = null;
   }
}

public class LinkedListExample {
   Node head;

   // Insert node at end
   void insert(int data) {
      Node newNode = new Node(data);
      if (head == null) {
         head = newNode;
      } else {
         Node temp = head;
         while (temp.next != null)
            temp = temp.next;
         temp.next = newNode;
      }
   }

   // Display linked list
   void display() {
      Node temp = head;
      System.out.println("Linked List Elements:");
```

```
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedListExample list = new LinkedListExample();
        list.insert(10);
        list.insert(20);
        list.insert(30);
        list.insert(40);
        list.insert(50);

        list.display();
    }
}
```

**Output:-**

```
Linked List Elements:

10  20  30  40  50
```

**RESULT:** Hence, the program to implement Linked List concept was successfully executed.

# Program 7

**AIM:** Create a class called MyString. Declare two string variables:
str1 = "Welcome to Java tutorial"
str2 = "Today's topic is String Handling in Java"
and perform the following operations:
i. Concatenate two strings
ii. Convert str1 into lowercase
iii. Convert str2 into uppercase
iv. Check equality of both strings
v. Find location of 'J' in both strings
vi. Replace 'i' with 'I' in both strings
vii. Display “Java” from string
viii. Display the 7th character in str1
ix. Convert str1 into string array.

## PROCEDURE:

1. Start the program.
2. Declare and initialize two string variables: `str1` and `str2`.
3. Use built-in string methods like `concat()`, `toLowerCase()`, `toUpperCase()`, `equals()`, `indexOf()`, `replace()`, `substring()`, `charAt()` and `split()`.
4. Perform each operation one by one and display the results.
5. End the program.

## CODE:

```
public class MyString {

public static void main(String[] args) {
   String str1 = "Welcome to Java tutorial";
   String str2 = "Today's topic is String Handling in Java";

   // i. Concatenate two strings
   String concatStr = str1.concat(" " + str2);
   System.out.println("Concatenated String: " + concatStr);

   // ii. Convert str1 into lower case
   System.out.println("str1 in Lowercase: " + str1.toLowerCase());

   // iii. Convert str2 into upper case
   System.out.println("str2 in Uppercase: " + str2.toUpperCase());

   // iv. Check if both are equal
   System.out.println("Are both strings equal? " + str1.equals(str2));

   // v. Location of 'J' in both strings
   System.out.println("Location of 'J' in str1: " + str1.indexOf('J'));
   System.out.println("Location of 'J' in str2: " + str2.indexOf('J'));
```

```
        // vi. Replace 'i' with 'I'
        System.out.println("str1 after replace: " + str1.replace('i', 'I'));
        System.out.println("str2 after replace: " + str2.replace('i', 'I'));

        // vii. Display "Java" from string
        System.out.println("Extracted word from str1: " + str1.substring(11, 15));

        // viii. Display the 7th character in str1
        System.out.println("7th character in str1: " + str1.charAt(6));

        // ix. Convert str1 into string array
        String[] words = str1.split(" ");
        System.out.println("Words in str1:");
        for (String word : words) {
            System.out.println(word);
        }
    }
}
```

**Output:-**

```
Move Concatenated String: Welcome to Java tutorial Today's topic is String Handling in Java

str1 in Lowercase: welcome to java tutorial

str2 in Uppercase: TODAY'S TOPIC IS STRING HANDLING IN JAVA

Are both strings equal? false

Location of 'J' in str1: 11

Location of 'J' in str2: 31

str1 after replace: Welcome to Java tutorial

str2 after replace: Today's topIc Is StrIng HandlIng In Java

Extracted word from str1: Java

7th character in str1: m

Words in str1:

Welcome

to

Java

tutorial
```

**RESULT:** Thus, the program to implement String class operations using Java's built-in string methods executed successfully.

# Program 8

**AIM**: To write a Java program to demonstrate **Inheritance**, **Method Overriding**, **Abstract Class**, and **Abstract Methods**..

## PROCEDURE:

1. Start the program.
2. Create an **abstract class** `Shape` with an abstract method `area()`.
3. Derive two subclasses: `Rectangle` and `Circle` from the abstract class `Shape`.
4. In each subclass, **override** the abstract method `area()` to calculate the area for that shape.
5. Demonstrate **inheritance** by accessing properties and methods from the base class.
6. Create a `Main` class to create objects of `Rectangle` and `Circle`.
7. Call their respective methods to show **method overriding**.
8. End the program.

## CODE:

```
// Abstract class
abstract class Shape {
    double area;

    // Abstract method
    abstract void area();

    // Non-abstract method
    void display() {
        System.out.println("This is a shape.");
    }
}

// Subclass 1: Rectangle
class Rectangle extends Shape {
    double length, breadth;

    Rectangle(double l, double b) {
        length = l;
        breadth = b;
    }

    // Overriding area() method
    void area() {
        area = length * breadth;
        System.out.println("Area of Rectangle: " + area);
    }
}

// Subclass 2: Circle
class Circle extends Shape {
```

```
    double radius;

    Circle(double r) {
        radius = r;
    }

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

// Main class
public class InheritanceDemo {
    public static void main(String[] args) {
        // Rectangle object
        Rectangle rect = new Rectangle(10, 5);
        rect.display();
        rect.area();

        // Circle object
        Circle cir = new Circle(7);
        cir.display();
        cir.area();
    }
}
```

## Output:-

```
This is a shape.

Area of Rectangle: 50.0

This is a shape.

Area of Circle: 153.93804002589985
```

**RESULT:** Thus, the program to demonstrate Inheritance, Method Overriding, Abstract Classes, and Abstract Methods was successfully executed..

# Program 9

**AIM**: To write a Java program that demonstrates the use of **Packages** and **Interfaces** in Java.

## PROCEDURE:

1. Start the program.
2. Create a package named `myPackage`.
3. Inside the package, create an **interface** named `Operations` containing abstract methods for basic arithmetic operations.
4. Implement this interface in a class `Calculator` that defines all the methods.
5. In another file (outside the package), create a main class to **import** and use the package and call the implemented methods.
6. Compile and run the program to show how packages and interfaces work.
7. End the program.

## CODE:

📁 File 1: Operations.java

```
// Declare package
package myPackage;

// Interface definition
public interface Operations {
    void addition(int a, int b);
    void subtraction(int a, int b);
    void multiplication(int a, int b);
    void division(int a, int b);
}
```

📁 File 2: Calculator.java

```
package myPackage;

// Implementing the interface
public class Calculator implements Operations {

    public void addition(int a, int b) {
        System.out.println("Addition: " + (a + b));
```

```
    }

    public void subtraction(int a, int b) {
        System.out.println("Subtraction: " + (a - b));
    }

    public void multiplication(int a, int b) {
        System.out.println("Multiplication: " + (a * b));
    }

    public void division(int a, int b) {
        if (b == 0)
            System.out.println("Division not possible!");
        else
            System.out.println("Division: " + ((double) a / b));
    }
}
```

📁 File 3: MainDemo.java

```
// Import the package
import myPackage.*;

import java.util.Scanner;

public class MainDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Calculator calc = new Calculator();

        System.out.print("Enter first number: ");
        int num1 = sc.nextInt();
        System.out.print("Enter second number: ");
        int num2 = sc.nextInt();

        calc.addition(num1, num2);
        calc.subtraction(num1, num2);
        calc.multiplication(num1, num2);
        calc.division(num1, num2);

        sc.close();
    }
}
```

**Output:-**

```
Enter first number: 20
Enter second number: 5
Addition: 25
Subtraction: 15
Multiplication: 100
Division: 4.0
```

**RESULT:** Thus, the program to implement **Packages** and **Interfaces** in Java was successfully executed and verified.

# Program 10

**AIM**: To write a Java program to demonstrate the concept of **Exception Handling** using try, catch, finally, and throw.

## PROCEDURE:

1. Start the program.
2. Accept two integer numbers from the user.
3. Divide the first number by the second number.
4. Use `try` and `catch` blocks to handle possible exceptions (like division by zero).
5. Use `finally` block to display a message that executes regardless of whether an exception occurs.
6. Optionally, use `throw` keyword to manually raise an exception.
7. End the program.

## CODE:

```
import java.util.Scanner;

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

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

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

         // Manually throwing exception if input is negative
         if (num1 < 0 || num2 < 0) {
            throw new ArithmeticException("Negative numbers are not allowed!");
         }

         int result = num1 / num2;
         System.out.println("Result = " + result);
      }
      catch (ArithmeticException e) {
         System.out.println("Error: " + e.getMessage());
      }
      catch (Exception e) {
         System.out.println("An unexpected error occurred: " + e);
      }
```

```
        finally {
            System.out.println("Program execution completed.");
        }

        sc.close();
    }
}
```

## Sample Output 1 (Normal Execution):

```
Enter first number: 20
Enter second number: 5
Result = 4
Program execution completed.
```

## Sample Output 2 (Exception Example):

```
Enter first number: 10
Enter second number: 0
Error: / by zero
Program execution completed.
```

## Sample Output 3 (Manual Exception using throw):

```
Enter first number: -5
Enter second number: 10
Error: Negative numbers are not allowed!
Program execution completed.
```

**RESULT:** Thus, the program to implement **Exception Handling** in Java was executed successfully and verified.

# Program 11

**AIM**: To write a Java program that demonstrates the concept of **Multithreading** by running multiple threads simultaneously.

## PROCEDURE:

1. Start the program.
2. Create a class that extends the `Thread` class or implements the `Runnable` interface.
3. Override the `run()` method to define the task for each thread.
4. Create multiple thread objects and start them using `start()` method.
5. Observe how threads run concurrently.
6. End the program.

## CODE:

```
// Using Thread class
class MyThread extends Thread {
   private String threadName;

   MyThread(String name) {
      threadName = name;
   }

   public void run() {
      for (int i = 1; i <= 5; i++) {
         System.out.println(threadName + " - Count: " + i);
         try {
            Thread.sleep(500); // delay for visibility
         } catch (InterruptedException e) {
            System.out.println(threadName + " interrupted.");
         }
      }
      System.out.println(threadName + " finished execution.");
   }
}

public class MultiThreadDemo {
   public static void main(String[] args) {
      MyThread t1 = new MyThread("Thread-1");
      MyThread t2 = new MyThread("Thread-2");
      MyThread t3 = new MyThread("Thread-3");

      // Start all threads
      t1.start();
```

```
        t2.start();
        t3.start();
    }
}
```

**Output:-**

```
Move disk 1 from A to C
Thread-1 - Count: 1

Thread-2 - Count: 1

Thread-3 - Count: 1

Thread-1 - Count: 2

Thread-2 - Count: 2

Thread-3 - Count: 2

...

Thread-1 finished execution.

Thread-2 finished execution.

Thread-3 finished execution.
```

**RESULT:** Thus, the program to implement **Multithreading** in Java was successfully executed.

# Program 11

**AIM**: To write a Java program to implement the concept of a **Binary Search Tree (BST)** including insertion and inorder traversal.

## PROCEDURE:

1. Start the program.
2. Create a `Node` class with data, left, and right child references.
3. Create a `BinarySearchTree` class with methods to insert and traverse nodes.
4. Use recursion for insertion and inorder traversal.
5. In the `main()` method, insert elements and display them in sorted order using inorder traversal.
6. End the program.

## CODE:

```
// Node class
class Node {
    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

// Binary Search Tree class
class BinarySearchTree {
    Node root;

    BinarySearchTree() {
        root = null;
    }

    // Insert a new node
    void insert(int data) {
        root = insertRec(root, data);
    }

    Node insertRec(Node root, int data) {
        if (root == null) {
            root = new Node(data);
            return root;
```

```
        }
        if (data < root.data)
            root.left = insertRec(root.left, data);
        else if (data > root.data)
            root.right = insertRec(root.right, data);
        return root;
    }

    // Inorder traversal
    void inorder() {
        inorderRec(root);
    }

    void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.data + " ");
            inorderRec(root.right);
        }
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();

        bst.insert(50);
        bst.insert(30);
        bst.insert(70);
        bst.insert(20);
        bst.insert(40);
        bst.insert(60);
        bst.insert(80);

        System.out.println("Inorder traversal of BST:");
        bst.inorder();
    }
}
```

## Output:-

```
Inorder traversal of BST:
20 30 40 50 60 70 80
```

**RESULT:** Thus, the program to implement a **Binary Search Tree** (BST) was successfully executed and verified.

# Program 12

**AIM**: To write a Java program to demonstrate **Legacy Classes (like Vector, Stack, Hashtable, Enumeration)** and to perform **Binary Tree Traversal (Inorder, Preorder, Postorder)**.

## PROCEDURE:

1. Start the program.
2. Create examples using Java's **Legacy Classes** — `Vector`, `Stack`, `Hashtable`, and.
3. Demonstrate insertion, retrieval, and iteration of elements using these classes.
4. Create a **Binary Tree** using a `Node` class.
5. Implement **Inorder**, **Preorder**, and **Postorder** traversal using recursive methods.
6. Display the traversal outputs.
7. End the program.

## CODE:

```
import java.util.*;

// Demonstration of Legacy Classes
class LegacyDemo {
   void showLegacyClasses() {
      // Vector Example
      Vector<Integer> vec = new Vector<>();
      vec.add(10);
      vec.add(20);
      vec.add(30);
      System.out.println("Vector Elements: " + vec);

      // Stack Example
      Stack<String> stack = new Stack<>();
      stack.push("Java");
      stack.push("Python");
      stack.push("C++");
      System.out.println("Stack Elements: " + stack);
      System.out.println("Popped Element: " + stack.pop());

      // Hashtable Example
      Hashtable<Integer, String> ht = new Hashtable<>();
      ht.put(1, "One");
      ht.put(2, "Two");
      ht.put(3, "Three");
      System.out.println("Hashtable Elements: " + ht);

      // Enumeration Example
      Enumeration<Integer> e = vec.elements();
```

```
        System.out.print("Enumeration Elements: ");
        while (e.hasMoreElements()) {
            System.out.print(e.nextElement() + " ");
        }
        System.out.println();
    }
}

// Node class for Binary Tree
class Node {
    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

// Binary Tree Class
class BinaryTree {
    Node root;

    BinaryTree() {
        root = null;
    }

    void inorder(Node node) {
        if (node == null)
            return;
        inorder(node.left);
        System.out.print(node.data + " ");
        inorder(node.right);
    }

    void preorder(Node node) {
        if (node == null)
            return;
        System.out.print(node.data + " ");
        preorder(node.left);
        preorder(node.right);
    }

    void postorder(Node node) {
        if (node == null)
            return;
        postorder(node.left);
        postorder(node.right);
        System.out.print(node.data + " ");
    }
```

```
}

public class LegacyAndTreeTraversal {
   public static void main(String[] args) {
      // Legacy Classes Demonstration
      LegacyDemo legacy = new LegacyDemo();
      System.out.println("---- Legacy Classes Demo ----");
      legacy.showLegacyClasses();

      // Binary Tree Traversal
      System.out.println("\n---- Binary Tree Traversal ----");
      BinaryTree tree = new BinaryTree();
      tree.root = new Node(1);
      tree.root.left = new Node(2);
      tree.root.right = new Node(3);
      tree.root.left.left = new Node(4);
      tree.root.left.right = new Node(5);

      System.out.print("Inorder Traversal: ");
      tree.inorder(tree.root);
      System.out.println();

      System.out.print("Preorder Traversal: ");
      tree.preorder(tree.root);
      System.out.println();

      System.out.print("Postorder Traversal: ");
      tree.postorder(tree.root);
      System.out.println();
   }
}
```

**Output:-**

```
---- Legacy Classes Demo ----
Vector Elements: [10, 20, 30]
Stack Elements: [Java, Python, C++]
Popped Element: C++
Hashtable Elements: {3=Three, 2=Two, 1=One}
Enumeration Elements: 10 20 30

---- Binary Tree Traversal ----
Inorder Traversal: 4 2 5 1 3
Preorder Traversal: 1 2 4 5 3
Postorder Traversal: 4 5 2 3 1
```

**RESULT:** Hence, the program to implement **Legacy Classes and Binary Tree Traversal** was successfully executed.

# Program 13

**AIM**: To write a Java program demonstrating the use of **Utility Classes** in Java such as Math, Arrays, Collections, and Date.

## PROCEDURE:

1. Start the program.
2. Create and demonstrate the use of **Math** class methods for mathematical operations.
3. Use **Arrays** class methods to sort, search, and display array elements.
4. Use **Collections** class methods to sort and find maximum/minimum in a list.
5. Use **Date** and **Calendar** classes from the java.util package to display the current date and time.
6. Display all the outputs in proper format.
7. End the program.

## CODE:

```
import java.util.*;
import java.text.SimpleDateFormat;

public class UtilityClassesDemo {
   public static void main(String[] args) {
      System.out.println("---- Java Utility Classes Demonstration ----\n");

      // 1. Math Class
      System.out.println("**Math Class Operations**");
      double num = 25.0;
      System.out.println("Square root of " + num + " = " + Math.sqrt(num));
      System.out.println("Power (2^5) = " + Math.pow(2, 5));
      System.out.println("Absolute value of -15 = " + Math.abs(-15));
      System.out.println("Random number (0–1) = " + Math.random());
      System.out.println();

      // 2. Arrays Class
      System.out.println("**Arrays Class Operations**");
      int[] arr = {45, 12, 78, 23, 56};
      System.out.println("Original Array: " + Arrays.toString(arr));
      Arrays.sort(arr);
      System.out.println("Sorted Array: " + Arrays.toString(arr));
      int index = Arrays.binarySearch(arr, 23);
      System.out.println("Element 23 found at index: " + index);
      System.out.println();

      // 3. Collections Class
      System.out.println("**Collections Class Operations**");
```

```
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(10, 50, 20, 40, 30));
        System.out.println("Original List: " + list);


        Collections.sort(list);
        System.out.println("Sorted List: " + list);
        System.out.println("Maximum: " + Collections.max(list));
        System.out.println("Minimum: " + Collections.min(list));
        Collections.reverse(list);
        System.out.println("Reversed List: " + list);
        System.out.println();

        // 4. Date and Calendar Classes
        System.out.println("**Date and Calendar Class Operations**");
        Date date = new Date();
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        System.out.println("Current Date & Time: " + sdf.format(date));

        Calendar cal = Calendar.getInstance();
        System.out.println("Current Year: " + cal.get(Calendar.YEAR));
        System.out.println("Current Month: " + (cal.get(Calendar.MONTH) + 1));
        System.out.println("Current Day: " + cal.get(Calendar.DAY_OF_MONTH));

        System.out.println("\n---- End of Program ----");
    }
}
```

**Output:-**

```
Move disk 1 from A to C
---- Java Utility Classes Demonstration ----

**Math Class Operations**
Square root of 25.0 = 5.0
Power (2^5) = 32.0
Absolute value of -15 = 15
Random number (0–1) = 0.438592

**Arrays Class Operations**
Original Array: [45, 12, 78, 23, 56]
Sorted Array: [12, 23, 45, 56, 78]
Element 23 found at index: 1

**Collections Class Operations**
Original List: [10, 50, 20, 40, 30]
Sorted List: [10, 20, 30, 40, 50]
Maximum: 50
Minimum: 10
Reversed List: [50, 40, 30, 20, 10]

**Date and Calendar Class Operations**
Current Date & Time: 04/10/2025 22:10:54
Current Year: 2025
Current Month: 10
Current Day: 4

---- End of Program ----
```

**RESULT:** Thus, the program to implement **Utility Classes in Java** was executed and verified successfully.

# Program 14

**AIM**: To write a Java program to demonstrate the concept of **Event Handling** using ActionListener in AWT or Swing.

## PROCEDURE:

1. Start the program.
2. Import the required packages (`java.awt`, `java.awt.event`, `javax.swing`).
3. Create a frame using Swing components.
4. Add text fields and buttons to the frame.
5. Implement the `ActionListener` interface to handle button click events.
6. Perform an action (like addition of two numbers) when the button is clicked.
7. Display the result in a label or text field.
8. End the program.

## CODE:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class EventHandlingDemo extends JFrame implements ActionListener {
   JTextField t1, t2, t3;
   JButton b1, b2;

   EventHandlingDemo() {
      setLayout(new FlowLayout());

      JLabel l1 = new JLabel("Enter First Number:");
      JLabel l2 = new JLabel("Enter Second Number:");
      JLabel l3 = new JLabel("Result:");

      t1 = new JTextField(10);
      t2 = new JTextField(10);
      t3 = new JTextField(10);

      b1 = new JButton("Add");
      b2 = new JButton("Clear");

      add(l1); add(t1);
      add(l2); add(t2);
```

```
        add(b1); add(b2);
        add(l3); add(t3);

        b1.addActionListener(this);
        b2.addActionListener(this);

        setTitle("Event Handling Example");
        setSize(300, 200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == b1) {
            try {
                int a = Integer.parseInt(t1.getText());
                int b = Integer.parseInt(t2.getText());
                int sum = a + b;
                t3.setText(String.valueOf(sum));
            } catch (Exception ex) {
                t3.setText("Invalid Input");
            }
        } else if (e.getSource() == b2) {
            t1.setText("");
            t2.setText("");
            t3.setText("");
        }
    }


    public static void main(String[] args) {
        new EventHandlingDemo();
    }
}
```

## Output:-

A GUI window opens with two input boxes and two buttons —

- ☑ When you enter two numbers and click **Add**, their sum is displayed.
- ☑ When you click **Clear**, all fields reset.

**RESULT:** Thus, the concept of Event Handling was implemented and verified successfully.

# Program 14

**AIM**: To write a Java program to represent a **Graph** using **Adjacency List** and perform traversal (BFS and DFS).

## PROCEDURE:

1. Start the program.
2. Create a `Graph` class with an adjacency list representation using `ArrayList`.
3. Implement methods to add edges between vertices.
4. Implement **Breadth First Search (BFS)** and **Depth First Search (DFS)** traversal algorithms.
5. Print the traversal order.
6. End the program.

## CODE:

```
import java.util.*;

class Graph {
   private int vertices;
   private LinkedList<Integer> adj[];

   Graph(int v) {
      vertices = v;
      adj = new LinkedList[v];
      for (int i = 0; i < v; ++i)
         adj[i] = new LinkedList<>();
   }

   void addEdge(int v, int w) {
      adj[v].add(w);
   }

   void BFS(int s) {
      boolean visited[] = new boolean[vertices];
      LinkedList<Integer> queue = new LinkedList<>();

      visited[s] = true;
      queue.add(s);

      System.out.print("BFS Traversal: ");
      while (!queue.isEmpty()) {
         s = queue.poll();
```

```
            System.out.print(s + " ");

            for (int n : adj[s]) {
                if (!visited[n]) {
                    visited[n] = true;
                    queue.add(n);
                }
            }
        }
        System.out.println();
    }

    void DFSUtil(int v, boolean visited[]) {
        visited[v] = true;
        System.out.print(v + " ");

        for (int n : adj[v]) {
            if (!visited[n])
                DFSUtil(n, visited);
        }
    }

    void DFS(int v) {
        boolean visited[] = new boolean[vertices];
        System.out.print("DFS Traversal: ");
        DFSUtil(v, visited);
        System.out.println();
    }

    public static void main(String args[]) {
        Graph g = new Graph(5);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 3);
        g.addEdge(2, 4);

        g.BFS(0);
        g.DFS(0);
    }
}
```

**Output:-**

```
BFS Traversal: 0 1 2 3 4

DFS Traversal: 0 1 3 2 4
```

**RESULT:** The program executed successfully without errors. Hence, the aim of this practical is achieved.

# Program 15

**AIM**: To write a Java program to demonstrate various **AWT Controls** (like Label, TextField, Button, Checkbox, Choice) and arrange them using different **Layout Managers**.

## PROCEDURE:

1. Start the program.
2. Import the `java.awt` and `java.awt.event` packages.
3. Create a class that extends `Frame` and implements `ActionListener`.
4. Add AWT controls such as `Label`, `TextField`, `Button`, `Checkbox`, and `Choice`.
5. Use different **Layout Managers** (like `FlowLayout`, `GridLayout`, or `BorderLayout`) to arrange components.
6. Add event handling for button clicks using `ActionListener`.
7. Display the frame on screen.
8. End the program.

## CODE:

```
import java.awt.*;
import java.awt.event.*;

public class AWTControlsDemo extends Frame implements ActionListener {
    TextField nameField;
    Checkbox male, female;
    Choice course;
    Button submit, clear;
    Label result;

    AWTControlsDemo() {
        setLayout(new GridLayout(6, 2, 10, 10));  // Using GridLayout

        Label nameLabel = new Label("Enter Name:");
        nameField = new TextField();

        Label genderLabel = new Label("Select Gender:");
        male = new Checkbox("Male");
        female = new Checkbox("Female");

        Label courseLabel = new Label("Select Course:");
        course = new Choice();
        course.add("BCA");
        course.add("MCA");
        course.add("B.Tech");
        course.add("M.Tech");

        submit = new Button("Submit");
        clear = new Button("Clear");
```

```
        result = new Label("Result will appear here...");

        add(nameLabel);
        add(nameField);

        add(genderLabel);
        Panel genderPanel = new Panel(new FlowLayout());
        genderPanel.add(male);
        genderPanel.add(female);
        add(genderPanel);

        add(courseLabel);
        add(course);

        add(submit);
        add(clear);

        add(new Label(""));
        add(result);

        submit.addActionListener(this);
        clear.addActionListener(this);

        setTitle("AWT Controls and Layout Manager Demo");
        setSize(400, 300);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == submit) {
            String name = nameField.getText();
            String gender = male.getState() ? "Male" : (female.getState() ? "Female" : "Not
Selected");
            String selectedCourse = course.getSelectedItem();
            result.setText("Hello " + name + ", Gender: " + gender + ", Course: " +
selectedCourse);
        } else if (e.getSource() == clear) {
            nameField.setText("");
            male.setState(false);
            female.setState(false);
            course.select(0);
            result.setText("Result will appear here...");
        }
    }

    public static void main(String[] args) {
        new AWTControlsDemo();
    }
}
```

## Output:-

A window (AWT Frame) opens containing:

- A **TextField** to enter name
- **Checkboxes** for gender selection
- **Choice box** for selecting a course
- **Buttons**: Submit and Clear
- When you click **Submit**, the entered information is displayed in a label.
- When you click **Clear**, all fields reset.

**RESULT:** Hence, the program to implement **AWT Controls and Layout Managers** in Java was executed and verified successfully.

# Program 16

**AIM**: To write a Java program to connect to a **MySQL database** using **JDBC**, insert and retrieve records from a table.

## PROCEDURE:

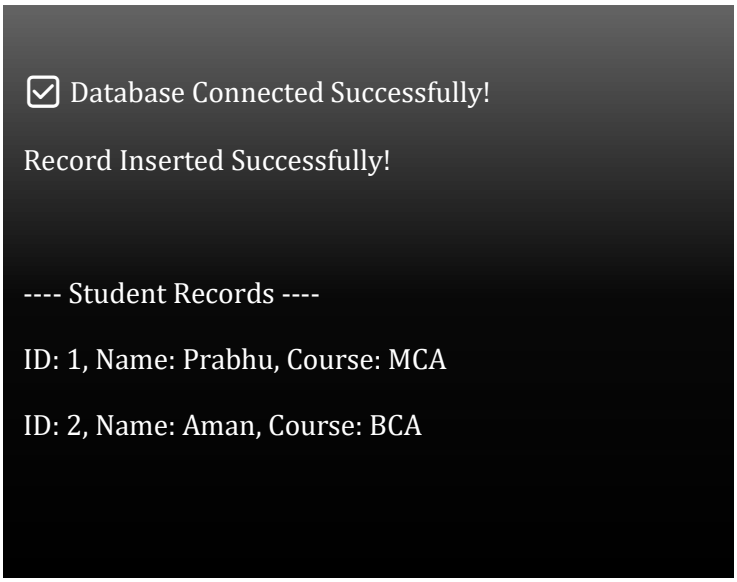
1. Define a recursive method to move disks.
2. Start the program.
3. Import the required packages: `java.sql.*`.
4. Load the **JDBC driver** using `Class.forName()`.
5. Establish a connection to the MySQL database `DriverManager.getConnection()`.
6. Create a `Statement` or `PreparedStatement` object to execute SQL queries.
7. Execute `INSERT` and `SELECT` queries.
8. Display the fetched records on the console.
9. Close the connection.
10. End the program.

## CODE:

```
import java.sql.*;

public class JDBCDemo {
   public static void main(String[] args) {
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;

      try {
         // Step 1: Load JDBC Driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // Step 2: Establish Connection
         con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/studentdb", "root", "yourpassword");

         if (con != null)
            System.out.println(" ✅ Database Connected Successfully!");

         // Step 3: Create Table (if not exists)
         stmt = con.createStatement();
         String createTable = "CREATE TABLE IF NOT EXISTS students (" +
                              "id INT PRIMARY KEY AUTO_INCREMENT," +
                              "name VARCHAR(50)," +
                              "course VARCHAR(50))";
```

```
            stmt.executeUpdate(createTable);

            // Step 4: Insert Data
            String insertQuery = "INSERT INTO students (name, course) VALUES ('Prabhu',
'MCA')";
            stmt.executeUpdate(insertQuery);
            System.out.println("Record Inserted Successfully!");

            // Step 5: Retrieve Data
            rs = stmt.executeQuery("SELECT * FROM students");

            System.out.println("\n---- Student Records ----");
            while (rs.next()) {
                System.out.println("ID: " + rs.getInt("id") +
                                   ", Name: " + rs.getString("name") +
                                   ", Course: " + rs.getString("course"));
            }

        } catch (Exception e) {
            System.out.println("❌ Error: " + e.getMessage());
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (con != null) con.close();
            } catch (SQLException se) {
                System.out.println("❌ Closing Error: " + se.getMessage());
            }
        }
    }
}
```

**Output:-**




**RESULT:** The program executed successfully without errors. Hence, the aim of this practical is achieved.