Program1

# Aim

To write a Java program that uses a `Hashtable` to implement a basic list, allowing the user to add, view, and remove elements interactively.

# Algorithm Start the program.

1. **Import** the required classes: `Hashtable` and `Scanner` from the Java library.
2. **Create** the main class and the `main()` method.
3. **Initialize** a hashtable to store integer keys and string values.
4. **Create** a scanner object to take input from the user.
5. **Set up** a loop that repeatedly displays a menu to the user until they choose to exit.
6. **Inside the loop**, ask the user to choose an operation.
7. **Use a switch-case** or conditional structure to:
    - If the user wants to add an element, take input and store it in the hashtable with a unique index.
    - If the user wants to view the list, go through all stored entries and display them.
    - If the user wants to remove an element, ask for the index and remove it from the hashtable if it exists.
    - If the user chooses to exit, break the loop and end the program.
    - For any other input, show an error message.
8. **Close** the scanner after exiting the loop.
9. **End** the program.

```
import java.util.Hashtable;        // Import Hashtable class for key-value storage

import java.util.Scanner;         // Import Scanner class to take user input


public class ListUsingHashtable {    // Main class definition

  public static void main(String[] args)

{

     Hashtable<Integer, String> list = new Hashtable<>();  // Create a Hashtable to act like a list

    Scanner scanner = new Scanner(System.in);           // Create Scanner object for reading input

    int index = 0;                              // Index to act like list position

    int choice;                                // Variable to store user's menu choice


    // Menu loop starts

    do {
```

```
// Display menu options
System.out.println("\nMenu:");
System.out.println("1. Add element");
System.out.println("2. View all elements");
System.out.println("3. Remove element");
System.out.println("4. Exit");
System.out.print("Enter your choice: ");
choice = scanner.nextInt();       // Read user's choice (1-4)
scanner.nextLine();               // Consume the leftover newline character

// Switch case to handle user input
switch (choice) {
  case 1:
    System.out.print("Enter element to add: "); // Ask for element
    String value = scanner.nextLine();        // Read element as string
    list.put(index, value);                   // Add element with index as key
    index++;                                  // Increment index for next element
    System.out.println("Element added.");
    break;

  case 2:
    System.out.println("Elements in the list:");
    for (int i = 0; i < index; i++) {          // Loop from 0 to latest index
      if (list.containsKey(i)) {              // Check if element exists at index
        System.out.println(i + ": " + list.get(i));  // Print index and value
      }
    }
    break;
  case 3:
    System.out.print("Enter index to remove: "); // Ask which index to remove
    int removeIndex = scanner.nextInt();       // Read index input
```

```
                if (list.containsKey(removeIndex)) {        // If key exists in hashtable
                    list.remove(removeIndex);              // Remove key-value pair
                    System.out.println("Element removed.");
                } else {
                    System.out.println("Index not found.");  // If index not in hashtable
                }
                break;
            case 4:
                System.out.println("Exiting...");           // Exit message
                break;
            default:
                System.out.println("Invalid choice.");      // Invalid option entered
        }
    } while (choice != 4);  // Repeat menu until user selects Exit
    scanner.close();       // Close the scanner to free resources
  }
}
```

**Output**

**Menu:**

**1. Add element**

**2. View all elements**

**3. Remove element**

**4. Exit**

**Enter your choice: 1**

**Enter element to add: Apple**

**Element added.**

**Menu:**

**1. Add element**

**2. View all elements**

**3. Remove element**

**4. Exit**

**Enter your choice: 1**

**Enter element to add: Banana**

**Element added.**

**Menu:**

**1. Add element**

**2. View all elements**

**3. Remove element**

**4. Exit**

**Enter your choice: 2**

**Elements in the list:**

**0: Apple**

**1: Banana**

**Menu:**

**1. Add element**

**2. View all elements**

**3. Remove element**

**4. Exit**

**Enter your choice: 3**

**Enter index to remove: 0**

**Element removed.**

**Menu:**

**1. Add element**

**2. View all elements**

**3. Remove element**

**4. Exit**

**Enter your choice: 2**

**Elements in the list:**

**1: Banana**

**Menu:**

**1. Add element**

**2. View all elements**

**3. Remove element**

**4. Exit**

**Enter your choice: 4**

**Exiting...**

**Program: 2**

# Aim

# To write a simple Java program to implement stack operations (push, pop, and display) using a

# Algorithm

1. **Start** the program.
2. **Import** the `Scanner` class to take input from the user.
3. **Declare** a stack using an integer array with a fixed size (e.g., 5 elements).
4. **Initialize** the `top` variable as `-1` to indicate that the stack is empty.
5. **Define** the `push()` method:
   - Check if the stack is full (`top == max - 1`).
   - If not full, increment `top` and insert the new value at `stack[top]`.
   - If full, display a "Stack Overflow" message.
6. **Define** the `pop()` method:
   - Check if the stack is empty (`top == -1`).
   - If not empty, print and remove the top value, then decrement `top`.
   - If empty, display a "Stack Underflow" message.
7. **Define** the `display()` method:
   - If the stack is empty, show "Stack is Empty".
   - Otherwise, print all stack elements from `top` to `0`.
8. **In the `main()` method**:
   - Create a loop to display a menu with options to push, pop, display, or exit.

    - Based on user choice, call the appropriate method.
    - Exit the loop when the user chooses to quit.
9. **Close** the scanner object.
10. **End** the program.

```
import java.util.Scanner;                    // Import Scanner for user input

public class StackUsingArray {
  static int max = 5, top = -1;              // Max size and stack top pointer
  static int[] stack = new int[max];          // Array to store stack elements

  static void push(int val) {                // Push operation
    if (top == max - 1)                      // Check for stack overflow
      System.out.println("Stack Overflow");
    else
      stack[++top] = val;                    // Increment top and insert value
  }
  static void pop() {                        // Pop operation
    if (top == -1)                           // Check for stack underflow
      System.out.println("Stack Underflow");
    else
      System.out.println("Popped: " + stack[top--]);  // Print and decrement top
  }
  static void display() {                    // Display stack elements
    if (top == -1)                           // Check if stack is empty
      System.out.println("Stack is Empty");
    else {
      for (int i = top; i >= 0; i--)       // Print from top to bottom
        System.out.println(stack[i]);
    }
  }
```

```
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);      // Create Scanner object
    int ch, val;
    do {
      System.out.print("\n1.Push 2.Pop 3.Display 4.Exit\nEnter choice: ");
      ch = sc.nextInt();                    // Read user choice
      switch(ch) {
        case 1 -> {                       // If user selects Push
          System.out.print("Value to push: ");
          val = sc.nextInt();            // Read value to push
          push(val);                     // Call push method
        }
        case 2 -> pop();                  // Call pop method
        case 3 -> display();              // Call display method
        case 4 -> System.out.println("Exiting...");  // Exit message
        default -> System.out.println("Invalid choice");  // Invalid input
      }
    } while (ch != 4);                    // Loop until exit chosen
    sc.close();                           // Close scanner resource
  }
}
```

## Output

1.Push 2.Pop 3.Display 4.Exit

Enter choice: 1

Value to push: 10

1.Push 2.Pop 3.Display 4.Exit

Enter choice: 1

Value to push: 20

1.Push 2.Pop 3.Display 4.Exit

Enter choice: 3

20

10

1.Push 2.Pop 3.Display 4.Exit

Enter choice: 2

Popped: 20

1.Push 2.Pop 3.Display 4.Exit

Enter choice: 3

10

1.Push 2.Pop 3.Display 4.Exit

Enter choice: 2

Popped: 10

1.Push 2.Pop 3.Display 4.Exit

Enter choice: 2

Stack Underflow

1.Push 2.Pop 3.Display 4.Exit

Enter choice: 4

Exiting...

# Program: 3

## Aim

To write a Java program to implement basic Queue operations (enqueue, dequeue, and display) using an array

# Algorithm

1. **Start** the program.
2. **Import** the `Scanner` class to take input from the user.
3. **Declare** an array to hold the queue elements and variables `front` and `rear` to keep track of the queue's front and rear positions. Initialize `front` and `rear` to -1.
4. **Create** an `enqueue` method to add elements:
   - Check if the queue is full (`rear == max - 1`).
   - If not full, increase `rear` by 1.
   - If it's the first element, set `front` to 0.
   - Add the new element to the queue at position `rear`.
5. **Create** a `dequeue` method to remove elements:
   - Check if the queue is empty (`front == -1` or `front > rear`).
   - If not empty, remove the element at `front`.
   - Increase `front` by 1.
6. **Create** a `display` method to show all elements:
   - If the queue is empty, display an appropriate message.
   - Otherwise, print all elements from `front` to `rear`.
7. **In the main method**, repeatedly display a menu and accept the user's choice.
8. Based on user choice, call the appropriate method (`enqueue`, `dequeue`, `display`) or exit.
9. **Close** the scanner and **end** the program.

```
import java.util.Scanner;                    // Import Scanner for input


public class QueueUsingArray {

  static int max = 5, front = -1, rear = -1;  // Max size, front and rear pointers

  static int[] queue = new int[max];          // Array to hold queue elements


  static void enqueue(int val) {              // Add element to queue

    if (rear == max - 1)                      // Check if queue is full

      System.out.println("Queue Overflow");

    else {

      if (front == -1) front = 0;             // First insertion sets front to 0

      queue[++rear] = val;                    // Increment rear and add element

      System.out.println(val + " enqueued");

    }
```

```java
}

static void dequeue() {                     // Remove element from queue
    if (front == -1 || front > rear)        // Check if queue is empty
        System.out.println("Queue Underflow");
    else
        System.out.println("Dequeued: " + queue[front++]); // Remove front element
}

static void display() {                     // Display queue elements
    if (front == -1 || front > rear)         // If empty
        System.out.println("Queue is empty");
    else {
        System.out.println("Queue elements:");
        for (int i = front; i <= rear; i++)  // Print from front to rear
            System.out.println(queue[i]);
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);      // Scanner for user input
    int choice, val;

    do {
        System.out.print("\n1.Enqueue 2.Dequeue 3.Display 4.Exit\nEnter choice: ");
        choice = sc.nextInt();               // Read user choice

        switch (choice) {
            case 1 -> {                      // If enqueue selected
```

```
                System.out.print("Enter value to enqueue: ");
                val = sc.nextInt();          // Read value to add
                enqueue(val);                // Call enqueue
            }
            case 2 -> dequeue();              // Call dequeue
            case 3 -> display();             // Call display
            case 4 -> System.out.println("Exiting...");  // Exit message
            default -> System.out.println("Invalid choice");  // Invalid input
        }
    } while (choice != 4);                   // Repeat until exit chosen

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

## Output

1.Enqueue 2.Dequeue 3.Display 4.Exit

Enter choice: 1

Enter value to enqueue: 10

10 enqueued

1.Enqueue 2.Dequeue 3.Display 4.Exit

Enter choice: 1

Enter value to enqueue: 20

20 enqueued

1.Enqueue 2.Dequeue 3.Display 4.Exit

Enter choice: 3

Queue elements:

10

20

1.Enqueue 2.Dequeue 3.Display 4.Exit

Enter choice: 2

Dequeued: 10

1.Enqueue 2.Dequeue 3.Display 4.Exit

Enter choice: 3

Queue elements:

20

1.Enqueue 2.Dequeue 3.Display 4.Exit

Enter choice: 2

Dequeued: 20

1.Enqueue 2.Dequeue 3.Display 4.Exit

Enter choice: 2

Queue Underflow

1.Enqueue 2.Dequeue 3.Display 4.Exit

Enter choice: 4

Exiting...

# Program:4

## Aim

To write a Java program that uses recursive functions to traverse a binary tree in Preorder, Inorder, and Postorder.

## Algorithm

1. **Start** the program.
2. **Create** a node structure with data, left child, and right child.

3. **Build** the binary tree by creating nodes and linking left and right children.
4. **Define** a recursive function for Preorder traversal:
   - If the current node is null, return.
   - Otherwise, process (print) the node data.
   - Recursively traverse the left subtree.
   - Recursively traverse the right subtree.
5. **Define** a recursive function for Inorder traversal:
   - If the current node is null, return.
   - Recursively traverse the left subtree.
   - Process (print) the node data.
   - Recursively traverse the right subtree.
6. **Define** a recursive function for Postorder traversal:
   - If the current node is null, return.
   - Recursively traverse the left subtree.
   - Recursively traverse the right subtree.
   - Process (print) the node data.
7. **Call** each traversal function starting from the root node to display the traversal orders.
8. **End** the program.

```
class Node {
    int data;          // Node data
    Node left, right;  // Left and right child nodes
    Node(int d) { data = d; }  // Constructor to set data
}

public class BinaryTreeTraversal {
    Node root;        // Root of the tree

    void preorder(Node n) {      // Preorder traversal: Root-Left-Right
        if (n == null) return;    // Base case: if node is null, return
        System.out.print(n.data + " ");  // Print node data
        preorder(n.left);        // Traverse left subtree
        preorder(n.right);       // Traverse right subtree
    }

    void inorder(Node n) {       // Inorder traversal: Left-Root-Right
```

```
    if (n == null) return;
    inorder(n.left);          // Traverse left subtree
    System.out.print(n.data + " ");  // Print node data
    inorder(n.right);         // Traverse right subtree
}

void postorder(Node n) {      // Postorder traversal: Left-Right-Root
    if (n == null) return;
    postorder(n.left);        // Traverse left subtree
    postorder(n.right);       // Traverse right subtree
    System.out.print(n.data + " ");  // Print node data
}

public static void main(String[] args) {
    BinaryTreeTraversal tree = new BinaryTreeTraversal();

    // Build the binary tree:
    //       1
    //      / \
    //     2   3
    //    / \   \
    //   4   5   6
    tree.root = new Node(1);               // Create root node
    tree.root.left = new Node(2);          // Left child of root
    tree.root.right = new Node(3);         // Right child of root
    tree.root.left.left = new Node(4);     // Left child of node 2
    tree.root.left.right = new Node(5);    // Right child of node 2
    tree.root.right.right = new Node(6);   // Right child of node 3
```

```
        System.out.print("Preorder: ");        // Print label
        tree.preorder(tree.root);              // Call preorder traversal
        System.out.println();

        System.out.print("Inorder: ");         // Print label
        tree.inorder(tree.root);               // Call inorder traversal
        System.out.println();

        System.out.print("Postorder: ");       // Print label
        tree.postorder(tree.root);             // Call postorder traversal
        System.out.println();
    }
}
```

## Output

Preorder: 1 2 4 5 3 6

Inorder: 4 2 5 1 3 6

Postorder: 4 5 2 6 3 1

## Program: 5

# Aim

To write a Java program to implement **Binary Search** on a sorted array using iterative method.

# Algorithm

1. **Start** the program.
2. **Input** the number of elements (`n`) from the user.
3. **Declare** an array of size `n`.
4. **Input** `n` sorted elements into the array.

5. **Input** the element to search (called `key`).
6. **Initialize** the search range:
   - Set `low = 0`
   - Set `high = n - 1`
7. **Repeat** the following steps while `low` is less than or equal to `high`:
   - Calculate `mid = (low + high) / 2`
   - If `arr[mid] == key`, the element is found. Print the position and **stop**.
   - If `key < arr[mid]`, set `high = mid - 1` to search the **left half**.
   - If `key > arr[mid]`, set `low = mid + 1` to search the **right half**.
8. If the loop ends without finding the element, print **"Element not found."**
9. **End** the program.

```
import java.util.Scanner; // To take input from user

public class BinarySearchExample {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);  // Create scanner object

        // Step 1: Get array size and elements

        System.out.print("Enter number of elements: ");

        int n = sc.nextInt();  // Read number of elements

        int[] arr = new int[n];  // Declare array

        System.out.println("Enter " + n + " sorted elements:");

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();  // Read array elements

        }

        // Step 2: Get the element to search

        System.out.print("Enter the element to search: ");

        int key = sc.nextInt();  // Read key to search


        // Step 3: Perform Binary Search

        int low = 0, high = n - 1, mid;

        boolean found = false;

        while (low <= high) {

            mid = (low + high) / 2;
```

```
            if (arr[mid] == key) {
                System.out.println("Element found at position: " + (mid + 1)); // 1-based position
                found = true;
                break;
            } else if (key < arr[mid]) {
                high = mid - 1; // Search in left half
            } else {
                low = mid + 1;  // Search in right half
            }
        }
        if (!found) {
            System.out.println("Element not found in the array.");
        }
        sc.close(); // Close the scanner
    }
}
```

Output:

Enter number of elements: 5

Enter 5 sorted elements:

5

15

25

35

45

Enter the element to search: 40

Element not found in the array.