TIT TECHNOCRATS

Estd. : 1999

NBA ACCREDITED CSE (AIML, AIDS, AI, DS, CY, CSE, EX, EC, CS, ME, IT

nirf 2021-22 Ranking

Celebrating 26 YEARS of Excellence

Only Group of MP & CG with NBA in 3 Engg. Colleges

UNBEATABLE PLACEMENTS

Technocrats Group Campus, BHEL, Bhopal - 462021 MP, India
Phone : +91-755-2751679 | Mobile : +91-9826374295, +91-9893141968
e-mail : placements@titbhopal.net | website : http://www.technocratsgroup.edu.in

Campus Specific Training

By: Mr.Gautam Singh

Technocrats Group of Institutions
Technocrats Group Campus, BHEL, Bhopal - 462021 MP, India
Phone : +91-755-2751679
e-mail: info@titbhopal.net | website: www.technocratsgroup.edu.in

1. Implementation of Singly Linked List in Java

A **Singly Linked List** consists of nodes where each node contains a data field and a reference (or link) to the next node.

```
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
```

```
class SinglyLinkedList {
   Node head; // Head (first node) of the list

   void insertAtBeginning(int data) {
            Node newNode = new Node(data);
            newNode.next = head;
            head = newNode;
 }
   void insertAtEnd(int data) {
             Node newNode = new Node(data);
             if (head == null) {
                    head = newNode;
                    return;
              }
              Node temp = head;
              while (temp.next != null) {
                        temp = temp.next;
               }
             temp.next = newNode;
   }
```

```
void insertAtPosition(int data, int position) {
    Node newNode = new Node(data);
    if (position == 1) {
        newNode.next = head;
        head = newNode;
        return;
    }
    Node temp = head;
    for (int i = 1; i < position - 1 && temp != null; i++) {
        temp = temp.next;
    }
    if (temp == null) {
        System.out.println("Invalid position");
        return;
    }
    newNode.next = temp.next;
    temp.next = newNode;
}
```

```
void deleteFromBeginning() {
    if (head == null) {
        System.out.println("List is empty");
        return;
    }
    head = head.next;
}
void deleteFromEnd() {
    if (head == null) {
        System.out.println("List is empty");
        return;
    }
    if (head.next == null) {
        head = null;
        return;
    }
    Node temp = head;
    while (temp.next.next != null) {
        temp = temp.next;
    }
    temp.next = null;
}
```

```
void deleteAtPosition(int position) {
    if (head == null) {
        System.out.println("List is empty");
        return;
    }
    if (position == 1) {
        head = head.next;
        return;
    }
    Node temp = head;
    for (int i = 1; i < position - 1 && temp.next != null; i++) {
        temp = temp.next;
    }
    if (temp.next == null) {
        System.out.println("Invalid position");
        return;
    }
    temp.next = temp.next.next;
}
```

```
boolean search(int key) {
        Node temp = head;
         while (temp != null) {
              if (temp.data == key) {
                        return true;
                }
            temp = temp.next;
    }
    return false;
}
void reverse() {
            Node prev = null;
            Node current = head;
            Node next = null;
            while (current != null) {
                    next = current.next;
                    current.next = prev;
                    prev = current;
                    current = next;
                }
                 head = prev;
    }
```

```
void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        SinglyLinkedList list = new SinglyLinkedList();
        list.insertAtEnd(10);
        list.insertAtEnd(20);
        list.insertAtEnd(30);
        list.display();

        list.reverse();
        list.display();
    }
}
```

1. Stack Implementation using Array

A **stack** follows the **LIFO (Last In, First Out)** principle.

Operations:

1. **Push** – Insert an element into the stack.
2. **Pop** – Remove the top element.
3. **Peek** – Retrieve the top element without removing it.
4. **isEmpty** – Check if the stack is empty.
5. **isFull** – Check if the stack is full.

```
class Stack {
    private int arr[];
    private int top;
    private int capacity;

    // Constructor to initialize the stack
    public Stack(int size) {
        arr = new int[size];
        capacity = size;
        top = -1;
    }

    // Push an element onto the stack
    public void push(int data) {
        if (isFull()) {
            System.out.println("Stack Overflow! Cannot insert " + data);
            return;
        }
        arr[++top] = data;
    }
```

```
// Pop an element from the stack
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow! Nothing to remove.");
            return -1;
        }
        return arr[top--];
    }

    // Peek the top element
    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1;
        }
        return arr[top];
    }

    // Check if the stack is empty
    public boolean isEmpty() {
        return top == -1;
    }

    // Check if the stack is full
    public boolean isFull() {
        return top == capacity - 1;
    }
```

```
// Display the stack elements
    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return;
        }
        System.out.print("Stack: ");
        for (int i = 0; i <= top; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    // Main method to test stack operations
    public static void main(String[] args) {
        Stack stack = new Stack(5);

        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.display();

        System.out.println("Peek: " + stack.peek());
        System.out.println("Popped: " + stack.pop());
        stack.display();
    }
}
```

2. Queue Implementation using Array

A **queue** follows the **FIFO (First In, First Out)** principle.

Operations:

1. **Enqueue** – Insert an element into the queue.
2. **Dequeue** – Remove an element from the front.
3. **Front** – Get the front element.
4. **isEmpty** – Check if the queue is empty.
5. **isFull** – Check if the queue is full.

```
class Queue {
    private int arr[];
    private int front, rear, capacity, size;

    // Constructor to initialize the queue
    public Queue(int capacity) {
        this.capacity = capacity;
        arr = new int[capacity];
        front = 0;
        rear = -1;
        size = 0;
    }

    // Enqueue operation
    public void enqueue(int data) {
        if (isFull()) {
            System.out.println("Queue Overflow! Cannot insert " + data);
            return;
        }
        rear = (rear + 1) % capacity;
        arr[rear] = data;
        size++;
    }
```

```
// Dequeue operation
  public int dequeue() {
      if (isEmpty()) {
          System.out.println("Queue Underflow! Nothing to remove.");
          return -1;
      }
      int removed = arr[front];
      front = (front + 1) % capacity;
      size--;
      return removed;
  }

  // Peek (Front element)
  public int peek() {
      if (isEmpty()) {
          System.out.println("Queue is empty!");
          return -1;
      }
      return arr[front];
  }
```

```
// Check if the queue is empty
    public boolean isEmpty() {
        return size == 0;
    }

    // Check if the queue is full
    public boolean isFull() {
        return size == capacity;
    }

    // Display the queue elements
    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is empty!");
            return;
        }
        System.out.print("Queue: ");
        for (int i = 0; i < size; i++) {
            System.out.print(arr[(front + i) % capacity] + " ");
        }
        System.out.println();
    }
```

```
// Main method to test queue operations
    public static void main(String[] args) {
        Queue queue = new Queue(5);

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.display();

        System.out.println("Front: " + queue.peek());
        System.out.println("Dequeued: " + queue.dequeue());
        queue.display();
    }
}
```