LAB MANUAL

K. R. MANGALAM UNIVERSITY

B. TECH CSE (DataScience)

SOET

(School of Engineering &Technology)

(DATA- STRUCTURE)

Submitted by- NAJIM KHAN
Submitted to- DR SWARTI

Roll No – 2401420003.

1. 1. Inventory Stock Management System

A. Aim:

To implement an Inventory Stock Management System using arrays in Java.

B. Theory:

An inventory system stores items and allows operations like:

- Add item
- Display all items
- Search item by ID

Java classes and objects are used to store item details.

C. Java Code:

```
import java.util.*;


class Item {
        int id;
        String name;
        int qty;


        Item(int id, String name, int qty) {
        this.id = id;
        this.name = name;
        this.qty = qty;
        }
}


public class InventorySystem {
```

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
Item[] items = new Item[50];
int count = 0;

while (true) {
System.out.println("\n1. Add Item\n2. Display Items\n3. Search Item\n4. Exit");
System.out.print("Enter choice: ");
int ch = sc.nextInt();

switch (ch) {
        case 1:
        System.out.print("Enter ID, Name, Quantity: ");
        int id = sc.nextInt();
        String name = sc.next();
        int qty = sc.nextInt();
        items[count++] = new Item(id, name, qty);
        break;

        case 2:
        System.out.println("\n--- Inventory List ---");
        for (int i = 0; i < count; i++)
        System.out.println(items[i].id + "  " + items[i].name + "  " + items[i].qty);
        break;

        case 3:
        System.out.print("Enter ID to search: ");
        int sid = sc.nextInt();
        boolean found = false;
```

```
                for (int i = 0; i < count; i++) {
                if (items[i].id == sid) {
                        System.out.println("Found: " + items[i].name + " Qty: " +
items[i].qty);
                        found = true;
                }
                }
                if (!found) System.out.println("Item not found.");
                break;

                case 4:
                System.exit(0);

                default:
                System.out.println("Invalid choice!");
        }
        }
        }
}
```

2. 2. Ticketing System Using Queue (Linear Queue)

A. Aim:

To implement a Ticketing System using a Linear Queue in Java.

B. Theory:

Queue follows FIFO (First In First Out).
Used for:

- Ticket counters
- Scheduling

Operations:

- Enqueue
- Dequeue
- Display

C. Java Code:

```
import java.util.*;


public class TicketQueue {
        static int front = 0, rear = -1;
        static int max = 50;
        static int[] queue = new int[max];

        static void enqueue(int x) {
        if (rear == max - 1) {
        System.out.println("Queue Full!");
        } else {
        queue[++rear] = x;
        System.out.println("Ticket added: " + x);
        }
        }

        static void dequeue() {
        if (front > rear) {
        System.out.println("Queue Empty!");
        } else {
        System.out.println("Ticket served: " + queue[front++]);
```

```
        }
        }

        static void display() {
        if (front > rear) {
        System.out.println("Queue Empty!");
        } else {
        System.out.print("Queue: ");
        for (int i = front; i <= rear; i++)
                System.out.print(queue[i] + " ");
        System.out.println();
        }
        }

        public static void main(String[] args) {
        enqueue(101);
        enqueue(102);
        enqueue(103);
        display();
        dequeue();
        display();
        }
}
```

3. 3. Browser History Navigation Using Stack

A. Aim:

To implement Browser Back navigation using Stack.

B. Theory:

Stack follows LIFO (Last In First Out).
Operations:

- Push (visit page)
- Pop (go back)

C. Java Code:

```
import java.util.Stack;


public class BrowserHistory {
        public static void main(String[] args) {
        Stack<String> history = new Stack<>();


        history.push("Google");
        history.push("YouTube");
        history.push("Instagram");


        System.out.println("Current Stack: " + history);
        System.out.println("Go Back: " + history.pop());
        System.out.println("After Back: " + history);
        }
}
```

4. 4. Singly Linked List (Insert, Delete, Search, Display)

A. Aim:

To perform Insert, Delete, Search and Display operations on a Singly Linked List.

B. Theory:

A linked list contains nodes that store:

- Data
- Next pointer

Operations:

- Insert at beginning
- Delete node
- Search node
- Display list

C. Java Code:

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

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

public class SinglyLinkedList {
        Node head = null;

        void insert(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
```

```
}

void delete(int key) {
Node temp = head, prev = null;

if (temp != null && temp.data == key) {
head = temp.next;
return;
}

while (temp != null && temp.data != key) {
prev = temp;
temp = temp.next;
}

if (temp == null) return;
prev.next = temp.next;
}

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

```
    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 s = new SinglyLinkedList();
    s.insert(10);
    s.insert(20);
    s.insert(30);
    s.display();
    System.out.println("Search 20: " + s.search(20));
    s.delete(20);
    s.display();
    }
}
```

5. 5. Circular Singly Linked List (Insert, Search, Delete, Display)

A. Java Code:

```
class CNode {
    int data;
    CNode next;
```

```
        CNode(int data) {
        this.data = data;
        }
}

public class CircularList {

        CNode head = null;

        void insert(int data) {
        CNode newNode = new CNode(data);

        if (head == null) {
        head = newNode;
        newNode.next = head;
        return;
        }

        CNode temp = head;
        while (temp.next != head)
        temp = temp.next;

        temp.next = newNode;
        newNode.next = head;
        }

        void display() {
        if (head == null) return;
```

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

        boolean search(int key) {
        CNode temp = head;
        do {
        if (temp.data == key)
                return true;
        temp = temp.next;
        } while (temp != head);
        return false;
        }

        public static void main(String[] args) {
        CircularList c = new CircularList();
        c.insert(1);
        c.insert(2);
        c.insert(3);
        c.display();
        }
}
```

6. 6. Reverse a String Using Stack

A. Aim:

To reverse a string using stack operations.

B. Theory:

Push all characters → Pop to reverse.

C. Java Code:

```
import java.util.Stack;


public class ReverseStringStack {
        public static void main(String[] args) {
        String str = "HELLO";
        Stack<Character> stack = new Stack<>();


        for (char c : str.toCharArray())
        stack.push(c);


        String reversed = "";
        while (!stack.isEmpty())
        reversed += stack.pop();


        System.out.println("Original: " + str);
        System.out.println("Reversed: " + reversed);
        }
}
```