K. R. MANGALAM UNIVERSITY, GURUGRAM, HARYANA, INDIA

Practical File

Data Structure Report file

Name: Yash Tiwari

Course: B-Tech CSE (Data Science)

Semester: 3rd

Batch: 2024-2028

Submitted to- **Signature**

Dr Swati Gupta

S No.	Topics	Page No.
1.	Lab Sheet 1 (Inventory Management System)	3-9
2.	Lab Sheet 2 (Browsing Navigation)	10-13
3.	Lab Sheet 3 (Singly Linked List)	14-21
4.	Lab Sheet 4 (Balanced Parentheses using Stack)	22-23
5.	Lab Sheet 5 (Reverse of String Using Stack)	24
6.	Lab Sheet 6 (Ticket Management System using Linear Queue)	25-27

1.Inventory Management System

Code-

InventoryManager.class | InventoryManager.java

InventoryManager.java > ...

```java
import java.util.ArrayList;
import java.util.Scanner;

class Product {
    String sku;
    String name;
    int quantity;

    // Constructor
    Product(String sku, String name, int quantity) {
        this.sku = sku;
        this.name = name;
        this.quantity = quantity;
    }
}

public class InventoryManager {

    static ArrayList<Product> inventory = new ArrayList<>();
    static final int MAX_CAPACITY = 100;   // For TC11 (inventory overflow)
    static Scanner sc = new Scanner(System.in);

    // ----------------------------
    // Function to insert product
    // ----------------------------
    static void insertProduct() {
        if (inventory.size() >= MAX_CAPACITY) {
            System.out.println(x: "Inventory capacity exceeded! Cannot insert more products.");
            return;
        }

        System.out.print(s: "Enter SKU: ");
        String sku = sc.nextLine();

        // Check duplicate SKU
        Product found = findBySKU(sku);
        if (found != null) {
            System.out.println(x: "Product with this SKU already exists!");
            System.out.print(s: "Do you want to update its quantity? (yes/no): ");
            String opt = sc.nextLine();

            if (opt.equalsIgnoreCase(anotherString: "yes")) {
                System.out.print(s: "Enter new quantity: ");
                int qty = sc.nextInt();
                sc.nextLine();
```

```
46
47                  if (qty < 0) {
48                      System.out.println(x: "Quantity must be positive!");
49                      return;
50                  }
51
52                  found.quantity = qty;
53                  System.out.println(x: "Quantity updated successfully.");
54              }
55              return;
56          }
57
58          System.out.print(s: "Enter Product Name: ");
59          String name = sc.nextLine();
60
61          if (name.trim().isEmpty()) {
62              System.out.println(x: "Product name cannot be empty!");
63              return;
64          }
65
66          System.out.print(s: "Enter Quantity: ");
67          int quantity;
68
69          try {
70              quantity = Integer.parseInt(sc.nextLine());
71          } catch (Exception e) {
72              System.out.println(x: "Invalid input. Quantity must be numeric.");
73              return;
74          }
75
76          if (quantity < 0) {
77              System.out.println(x: "Quantity must be positive!");
78              return;
79          }
80
81          Product p = new Product(sku, name, quantity);
82          inventory.add(p);
83
84          System.out.println(x: "Product inserted successfully.");
85      }
86
```

```
86
87      // ----------------------------
88      // Display Inventory
89      // ----------------------------
90      static void displayInventory() {
91          if (inventory.isEmpty()) {
92              System.out.println(x: "Inventory is empty.");
93              return;
94          }
95
96          System.out.println(x: "\nCurrent Inventory:");
97          System.out.println(x: "SKU\t\tName\t\tQuantity");
98          System.out.println(x: "-----------------------------------------");
99
100         for (Product p : inventory) {
101             System.out.println(p.sku + "\t\t" + p.name + "\t\t" + p.quantity);
102         }
103     }
104
105     // ----------------------------
106     // Search by SKU
107     // ----------------------------
108     static Product findBySKU(String sku) {
109         for (Product p : inventory) {
110             if (p.sku.equals(sku)) return p;
111         }
112         return null;
113     }
114
115     static void searchBySKU() {
116         System.out.print(s: "Enter SKU to search: ");
117         String sku = sc.nextLine();
118
119         Product p = findBySKU(sku);
120         if (p != null) {
121             System.out.println("Product Found: " + p.name + " | Qty: " + p.quantity);
122         } else {
123             System.out.println(x: "Product not found.");
124         }
125     }
126
```

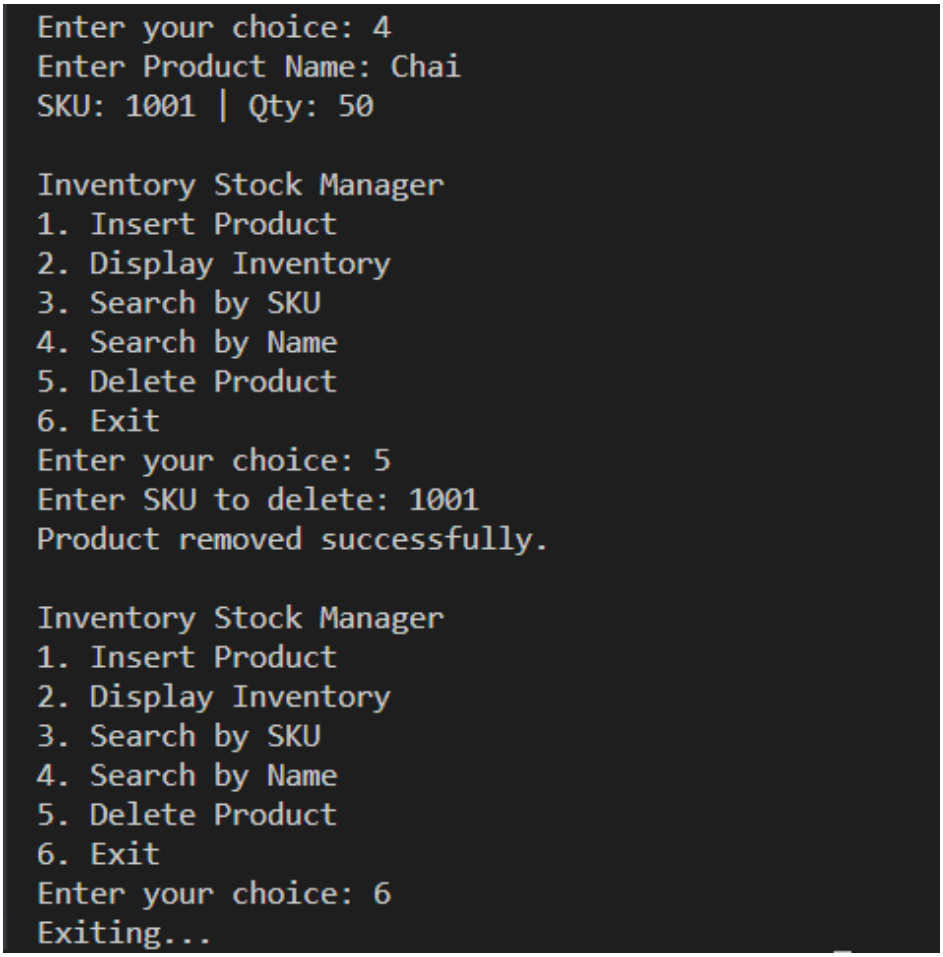
```java
    // ----------------------------
    // Search by Name
    // ----------------------------
    static void searchByName() {
        System.out.print(s: "Enter Product Name: ");
        String name = sc.nextLine();

        boolean found = false;

        for (Product p : inventory) {
            if (p.name.equalsIgnoreCase(name)) {
                System.out.println("SKU: " + p.sku + " | Qty: " + p.quantity);
                found = true;
            }
        }

        if (!found) {
            System.out.println(x: "No product found with this name.");
        }
    }

    // ----------------------------
    // Delete Product
    // ----------------------------
    static void deleteProduct() {
        System.out.print(s: "Enter SKU to delete: ");
        String sku = sc.nextLine();

        Product p = findBySKU(sku);
        if (p != null) {
            inventory.remove(p);
            System.out.println(x: "Product removed successfully.");
        } else {
            System.out.println(x: "Product not found.");
        }
    }
```

```java
    // ----------------------------
    // Main Menu
    // ----------------------------
    Run | Debug
    public static void main(String[] args) {

        while (true) {
            System.out.println(x: "\nInventory Stock Manager");
            System.out.println(x: "1. Insert Product");
            System.out.println(x: "2. Display Inventory");
            System.out.println(x: "3. Search by SKU");
            System.out.println(x: "4. Search by Name");
            System.out.println(x: "5. Delete Product");
            System.out.println(x: "6. Exit");

            System.out.print(s: "Enter your choice: ");
            String ch = sc.nextLine();

            switch (ch) {
                case "1": insertProduct(); break;
                case "2": displayInventory(); break;
                case "3": searchBySKU(); break;
                case "4": searchByName(); break;
                case "5": deleteProduct(); break;
                case "6": System.out.println(x: "Exiting..."); return;
                default: System.out.println(x: "Invalid choice!");
            }
        }
    }
}
```

Output

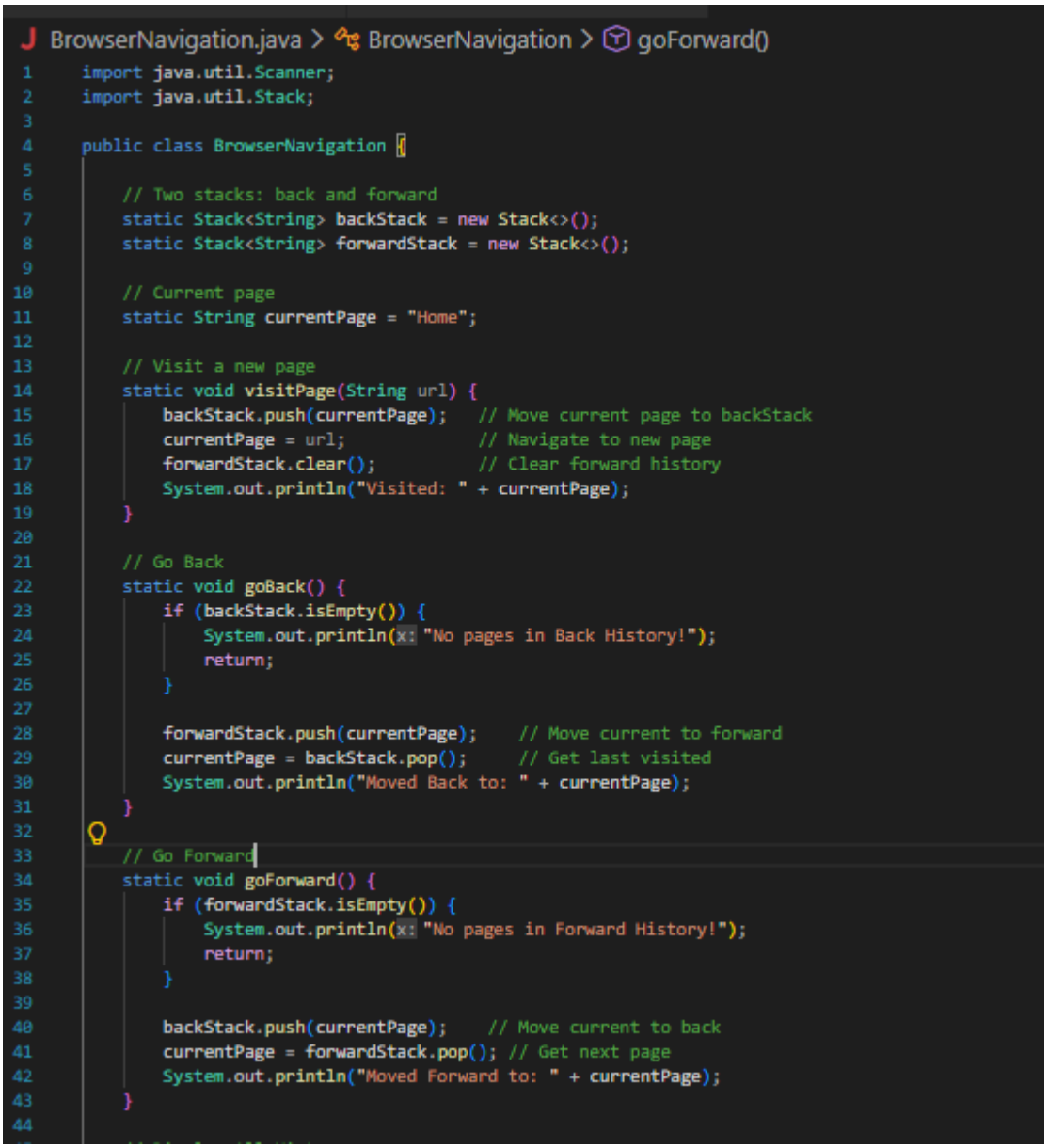
```
Inventory Stock Manager
1. Insert Product
2. Display Inventory
3. Search by SKU
4. Search by Name
5. Delete Product
6. Exit
Enter your choice: 1
Enter SKU: 1001
Enter Product Name: Chai
Enter Quantity: 50
Product inserted successfully.

Inventory Stock Manager
1. Insert Product
2. Display Inventory
3. Search by SKU
4. Search by Name
5. Delete Product
6. Exit
```

```
Enter your choice: 2

Current Inventory:
SKU             Name            Quantity
-----------------------------------------
1001            Chai            50

Inventory Stock Manager
1. Insert Product
2. Display Inventory
3. Search by SKU
4. Search by Name
5. Delete Product
6. Exit
Enter your choice: 3
Enter SKU to search: 1001
Product Found: Chai | Qty: 50

Inventory Stock Manager
1. Insert Product
2. Display Inventory
3. Search by SKU
4. Search by Name
5. Delete Product
6. Exit
```


Enter your choice: 4
Enter Product Name: Chai
SKU: 1001 | Qty: 50
Inventory Stock Manager
1. Insert Product
2. Display Inventory
3. Search by SKU
4. Search by Name
5. Delete Product
6. Exit
Enter your choice: 5
Enter SKU to delete: 1001
Product removed successfully.
Inventory Stock Manager
1. Insert Product
2. Display Inventory
3. Search by SKU
4. Search by Name
5. Delete Product
6. Exit
Enter your choice: 6
Exiting...

2.Browsing Navigation

Code-




```java
import java.util.Scanner;
import java.util.Stack;

public class BrowserNavigation {

    // Two stacks: back and forward
    static Stack<String> backStack = new Stack<>();
    static Stack<String> forwardStack = new Stack<>();

    // Current page
    static String currentPage = "Home";

    // Visit a new page
    static void visitPage(String url) {
        backStack.push(currentPage);   // Move current page to backStack
        currentPage = url;             // Navigate to new page
        forwardStack.clear();          // Clear forward history
        System.out.println("Visited: " + currentPage);
    }

    // Go Back
    static void goBack() {
        if (backStack.isEmpty()) {
            System.out.println(x: "No pages in Back History!");
            return;
        }

        forwardStack.push(currentPage);    // Move current to forward
        currentPage = backStack.pop();     // Get last visited
        System.out.println("Moved Back to: " + currentPage);
    }

    // Go Forward
    static void goForward() {
        if (forwardStack.isEmpty()) {
            System.out.println(x: "No pages in Forward History!");
            return;
        }

        backStack.push(currentPage);    // Move current to back
        currentPage = forwardStack.pop(); // Get next page
        System.out.println("Moved Forward to: " + currentPage);
    }

```

```java
    // Display All History
    static void displayHistory() {
        System.out.println("\n------ Browser History ------");
        System.out.println("Back Stack: " + backStack);
        System.out.println("Current Page: " + currentPage);
        System.out.println("Forward Stack: " + forwardStack);
        System.out.println("-----------------------------\n");
    }
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int choice;
        while (true) {
            System.out.println("===== Browser Navigation Menu =====");
            System.out.println("1. Visit New Page");
            System.out.println("2. Go Back");
            System.out.println("3. Go Forward");
            System.out.println("4. Show History");
            System.out.println("5. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // consume newline
            switch (choice) {
                case 1:
                    System.out.print("Enter URL to visit: ");
                    String url = sc.nextLine();
                    visitPage(url);
                    break;
                case 2:
                    goBack();
                    break;
                case 3:
                    goForward();
                    break;
                case 4:
                    displayHistory();
                    break;
                case 5:
                    System.out.println("Exiting Browser...");
                    return;
                default:
                    System.out.println("Invalid choice! Try again.");
            }
        }
    }
}
```

Output-

```
PS C:\Users\admin\OneDrive\Desktop\lab file> 
 BrowserNavigation.java } ; if ($?) { java B
===== Browser Navigation Menu =====
1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 1
Enter URL to visit: www.google.com
Visited: www.google.com
===== Browser Navigation Menu =====
1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 2
Moved Back to: Home
===== Browser Navigation Menu =====
1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
```

```
Enter choice: 3
Moved Forward to: www.google.com
===== Browser Navigation Menu =====
1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice: 4

------ Browser History ------
Back Stack: [Home]
Current Page: www.google.com
Forward Stack: []
-----------------------------

===== Browser Navigation Menu =====
1. Visit New Page
2. Go Back
3. Go Forward
4. Show History
5. Exit
Enter choice:
```

3. Singly Linked List

Code-

SinglyLinkedList.java > ...

```java
import java.util.Scanner;

class Node {
    int data;
    Node next;

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

public class SinglyLinkedList {

    Node head = null;

    // Insert at beginning
    void insertAtBeginning(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
        System.out.println(x: "Inserted at beginning.");
    }

    // Insert at end
    void insertAtEnd(int data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = newNode;
            System.out.println(x: "Inserted at end.");
            return;
        }

        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }

        temp.next = newNode;
        System.out.println(x: "Inserted at end.");
    }

```

```java

    // Insert at position
    void insertAtPosition(int data, int pos) {
        Node newNode = new Node(data);

        if (pos == 1) {
            newNode.next = head;
            head = newNode;
            System.out.println("Inserted at position " + pos);
            return;
        }

        Node temp = head;
        for (int i = 1; i < pos - 1 && temp != null; i++) {
            temp = temp.next;
        }

        if (temp == null) {
            System.out.println(x: "Position out of range!");
            return;
        }

        newNode.next = temp.next;
        temp.next = newNode;

        System.out.println("Inserted at position " + pos);
    }

    // Delete first node
    void deleteFirst() {
        if (head == null) {
            System.out.println(x: "List is empty!");
            return;
        }

        head = head.next;
        System.out.println(x: "First node deleted.");
    }

    // Delete last node
    void deleteLast() {
        if (head == null) {
            System.out.println(x: "List is empty!");
            return;
        }
```

```
89          if (head.next == null) {
90              head = null;
91              System.out.println(x: "Last node deleted.");
92              return;
93          }
94
95          Node temp = head;
96          while (temp.next.next != null) {
97              temp = temp.next;
98          }
99
100         temp.next = null;
101         System.out.println(x: "Last node deleted.");
102     }
103
104     // Delete by value
105     void deleteByValue(int value) {
106         if (head == null) {
107             System.out.println(x: "List is empty!");
108             return;
109         }
110
111         if (head.data == value) {
112             head = head.next;
113             System.out.println("Node with value " + value + " deleted.");
114             return;
115         }
116
117         Node temp = head;
118         while (temp.next != null && temp.next.data != value) {
119             temp = temp.next;
120         }
121
122         if (temp.next == null) {
123             System.out.println(x: "Value not found!");
124         } else {
125             temp.next = temp.next.next;
126             System.out.println("Node with value " + value + " deleted.");
127         }
128     }
129
```

```java
    // Search value
    void search(int value) {
        Node temp = head;
        int pos = 1;

        while (temp != null) {
            if (temp.data == value) {
                System.out.println("Value " + value + " found at position " + pos);
                return;
            }
            temp = temp.next;
            pos++;
        }

        System.out.println(x: "Value not found!");
    }

    // Sort list (Ascending)
    void sortList() {
        if (head == null) {
            System.out.println(x: "List is empty!");
            return;
        }

        Node current = head;
        Node index = null;

        while (current != null) {
            index = current.next;

            while (index != null) {
                if (current.data > index.data) {
                    // swap
                    int temp = current.data;
                    current.data = index.data;
                    index.data = temp;
                }
                index = index.next;
            }
            current = current.next;
        }

        System.out.println(x: "List sorted successfully.");
    }
```

```
175     // Display list
176     void display() {
177         if (head == null) {
178             System.out.println(x: "List is empty!");
179             return;
180         }
181 
182         Node temp = head;
183         System.out.print(s: "Linked List: ");
184 
185         while (temp != null) {
186             System.out.print(temp.data + " -> ");
187             temp = temp.next;
188         }
189 
190         System.out.println(x: "NULL");
191     }
192 
193     // Main function / Menu-driven program
        Run | Debug
194     public static void main(String[] args) {
195         Scanner sc = new Scanner(System.in);
196         SinglyLinkedList list = new SinglyLinkedList();
197 
198         while (true) {
199             System.out.println(x: "\n===== Singly Linked List Menu =====");
200             System.out.println(x: "1. Insert at Beginning");
201             System.out.println(x: "2. Insert at End");
202             System.out.println(x: "3. Insert at Position");
203             System.out.println(x: "4. Delete First");
204             System.out.println(x: "5. Delete Last");
205             System.out.println(x: "6. Delete by Value");
206             System.out.println(x: "7. Search");
207             System.out.println(x: "8. Sort List");
208             System.out.println(x: "9. Display List");
209             System.out.println(x: "10. Exit");
210             System.out.print(s: "Enter choice: ");
211 
212             int choice = sc.nextInt();
213 
214             switch (choice) {
215                 case 1:
216                     System.out.print(s: "Enter value: ");
217                     list.insertAtBeginning(sc.nextInt());
218                     break;
219 
```

```
                case 2:
                    System.out.print(s: "Enter value: ");
                    list.insertAtEnd(sc.nextInt());
                    break;
                case 3:
                    System.out.print(s: "Enter value: ");
                    int val = sc.nextInt();
                    System.out.print(s: "Enter position: ");
                    int pos = sc.nextInt();
                    list.insertAtPosition(val, pos);
                    break;
                case 4:
                    list.deleteFirst();
                    break;
                case 5:
                    list.deleteLast();
                    break;
                case 6:
                    System.out.print(s: "Enter value to delete: ");
                    list.deleteByValue(sc.nextInt());
                    break;
                case 7:
                    System.out.print(s: "Enter value to search: ");
                    list.search(sc.nextInt());
                    break;
                case 8:
                    list.sortList();
                    break;
                case 9:
                    list.display();
                    break;
                case 10:
                    System.out.println(x: "Exiting program...");
                    return;

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

Output-

```
===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 1
Enter value: 10
Inserted at beginning.

===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 2
Enter value: 10
Inserted at end.

===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 3
Enter value: 5
Enter position: 2
Inserted at position 2
```

```
===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 4
First node deleted.

===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 5
Last node deleted.

===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 6
Enter value to delete: 5
Node with value 5 deleted.

===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 7
Enter value to search: 10
Value not found!
```


===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 8
List is empty!
===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 9
List is empty!
===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 1
Enter value: 10
Inserted at beginning.
===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 10
9. Display List
10. Exit


10. Exit
Enter choice: 1
Enter value: 10
Inserted at beginning.
===== Singly Linked List Menu =====
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 10
4. Delete First
5. Delete Last
6. Delete by Value
7. Search
8. Sort List
9. Display List
10. Exit
Enter choice: 10
8. Sort List
9. Display List
10. Exit
Enter choice: 10
10. Exit
Enter choice: 10
Exiting program...
PS C:\Users\admin\OneDrive\Desktop\lab file>

4.Balanced Parentheses Using Stack

BalancedParentheses.java > ...

```java
import java.util.Scanner;
import java.util.Stack;

public class BalancedParentheses {

    // Function to check balanced parentheses
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();

        // Mapping closing brackets to opening brackets
        for (char ch : expression.toCharArray()) {

            // Push opening brackets
            if (ch == '(' || ch == '{' || ch == '[') {
                stack.push(ch);
            }

            // If closing bracket is found
            else if (ch == ')' || ch == '}' || ch == ']') {
                // Stack empty → no matching opening bracket
                if (stack.isEmpty()) {
                    return false;
                }

                char top = stack.pop();

                // Check correct matching
                if ((ch == ')' && top != '(') ||
                    (ch == '}' && top != '{') ||
                    (ch == ']' && top != '[')) {
                    return false;
                }
            }
        }

        // If stack empty → all brackets matched properly
        return stack.isEmpty();
    }

    // Main function
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print(s: "Enter an expression: ");
        String expr = sc.nextLine();

        if (isBalanced(expr)) {
            System.out.println(x: "Parentheses are balanced.");
        } else {
            System.out.println(x: "Parentheses are NOT balanced.");
        }
    }
}
```

Output-

```
rentheses }
Enter an expression: (a + b) * (c + d)
Parentheses are balanced.
PS C:\Users\admin\OneDrive\Desktop\lab file>
```

```
Enter an expression: (a+b}*(2050]
Parentheses are NOT balanced.
PS C:\Users\admin\OneDrive\Desktop\lab file>
```

5.Reverse of String Using Stack

Code-

ReverseStringUsingStack.java > ...

```java
import java.util.Scanner;
import java.util.Stack;

public class ReverseStringUsingStack {

    // Function to reverse string using stack
    public static String reverseString(String str) {
        Stack<Character> stack = new Stack<>();

        // Push all characters into stack
        for (char ch : str.toCharArray()) {
            stack.push(ch);
        }

        // Pop characters to build reversed string
        StringBuilder reversed = new StringBuilder();
        while (!stack.isEmpty()) {
            reversed.append(stack.pop());
        }

        return reversed.toString();
    }

    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print(s: "Enter a string: ");
        String input = sc.nextLine();

        String result = reverseString(input);
        System.out.println("Reversed String: " + result);
    }
}

```

Output-

```
eStringUsingStack }
Enter a string: YahsTiwari
Reversed String: irawiTshaY
PS C:\Users\admin\OneDrive\Desktop\lab file>
```

6. Ticket Management System using Linear Queue

Code-

```python
# ------------------------------------------------------
# Ticketing System using Linear Queue (Python)
# ------------------------------------------------------

class TicketQueue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size   # fixed-size linear queue
        self.front = -1
        self.rear = -1

    # Check if queue is full
    def isFull(self):
        return self.rear == self.size - 1

    # Check if queue is empty
    def isEmpty(self):
        return self.front == -1 or self.front > self.rear

    # Add a ticket request (Enqueue)
    def enqueue(self, ticket_id):
        if self.isFull():
            print("Queue is Full! Cannot add more ticket requests.")
            return

        if self.front == -1:
            self.front = 0

        self.rear += 1
        self.queue[self.rear] = ticket_id
        print(f"Ticket Request Added: {ticket_id}")

    # Process a ticket request (Dequeue)
    def dequeue(self):
        if self.isEmpty():
            print("Queue is Empty! No ticket to process.")
            return None

        ticket = self.queue[self.front]
        print(f"Ticket Processed: {ticket}")
        self.front += 1
        return ticket
```

```
        # Display all pending tickets
        def display(self):
            if self.isEmpty():
                print("No pending ticket requests.")
                return

            print("Pending Tickets:", end=" ")
            for i in range(self.front, self.rear + 1):
                print(self.queue[i], end=" ")
            print()


    # ----------------------------------------------------
    # Main Program (Menu Driven)
    # ----------------------------------------------------

    def main():
        q = TicketQueue(size=5)   # queue can hold 5 ticket requests

        while True:
            print("\n------- Ticketing System -------")
            print("1. Add Ticket Request (Enqueue)")
            print("2. Process Ticket (Dequeue)")
            print("3. Show Pending Tickets")
            print("4. Exit")

            choice = input("Enter your choice (1-4): ")

            if choice == "1":
                ticket_id = input("Enter Ticket ID: ")
                q.enqueue(ticket_id)

            elif choice == "2":
                q.dequeue()

            elif choice == "3":
                q.display()

            elif choice == "4":
                print("Exiting Ticketing System.")
                break

            else:
                print("Invalid choice! Please enter a number between 1-4.")


    # Run the program
    if __name__ == "__main__":
        main()

```

Output-

```
======= Ticketing System =======
1. Add Ticket Request (Enqueue)
2. Process Ticket (Dequeue)
3. Show Pending Tickets
4. Exit
Enter your choice (1-4): 1
Enter Ticket ID: 1001
Ticket Request Added: 1001

======= Ticketing System =======
1. Add Ticket Request (Enqueue)
2. Process Ticket (Dequeue)
3. Show Pending Tickets
4. Exit
Enter your choice (1-4): 2
Ticket Processed: 1001

======= Ticketing System =======
1. Add Ticket Request (Enqueue)
2. Process Ticket (Dequeue)
3. Show Pending Tickets
4. Exit
Enter your choice (1-4): 3
No pending ticket requests.

======= Ticketing System =======
1. Add Ticket Request (Enqueue)
2. Process Ticket (Dequeue)
3. Show Pending Tickets
4. Exit
Enter your choice (1-4): 4
Exiting Ticketing System.
PS C:\Users\admin\OneDrive\Desktop\lab file>
```