Name: Sahil Prajapati

Register Number: *RA2411003012283*

Problem 1: Text Editor Undo (Stack)

Java

```
import java.util.*;

public class TextEditorUndo {
    public static void main(String[] args) {
        Stack<String> stack = new Stack<>();
        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.print("Enter command (TYPE <word>/UNDO/PRINT/EXIT): ");
            String cmd = sc.next();

            // Handle TYPE
            if (cmd.equals("TYPE")) {
                String word = sc.next();
                stack.push(word); [cite_start]// [cite: 20]
            }
            // Handle UNDO
            else if (cmd.equals("UNDO")) {
                if (!stack.isEmpty()) {
                    stack.pop(); [cite_start]// [cite: 21]
                } else {
                    System.out.println("Nothing to undo.");
                }
            }
            // Handle PRINT
            else if (cmd.equals("PRINT")) {
                [cite_start]// [cite: 22]
                String sentence = "";
                for (String word : stack) {
                    sentence += word + " ";
                }
```

```
            System.out.println("Current Text: " + sentence.trim());
        }
        // Handle EXIT
        else if (cmd.equals("EXIT")) {
            System.out.println("Exiting editor.");
            break; [cite_start]// [cite: 24]
        }
        else {
            System.out.println("Invalid command.");
        }
    }
    sc.close();
  }
}
```

Problem 2: Browser Navigation Simulation (Stack)

Java

```
import java.util.*;

public class BrowserNavigation {
  public static void main(String[] args) {
    Stack<String> backStack = new Stack<>();
    Stack<String> forwardStack = new Stack<>();
    Scanner sc = new Scanner(System.in);
    String current = "Home"; [cite_start]// [cite: 50, 51]

    while (true) {
      System.out.print("Command (VISIT/BACK/FORWARD/PRINT/EXIT): ");
      String cmd = sc.next();

      // Implement VISIT
      if (cmd.equals("VISIT")) {
        String page = sc.next();
        backStack.push(current); [cite_start]// [cite: 56]
        current = page;
        forwardStack.clear(); // Visiting a new page clears the forward history
        System.out.println("Visited: " + current);
```

```
        }
        // Implement BACK
        else if (cmd.equals("BACK")) {
            if (!backStack.isEmpty()) {
                forwardStack.push(current); [cite_start]// [cite: 58]
                current = backStack.pop();
                System.out.println("Went back. Current Page: " + current);
            } else {
                System.out.println("Cannot go back. At Home.");
            }
        }
        // Implement FORWARD
        else if (cmd.equals("FORWARD")) {
            if (!forwardStack.isEmpty()) {
                backStack.push(current); [cite_start]// [cite: 60]
                current = forwardStack.pop();
                System.out.println("Went forward. Current Page: " + current);
            } else {
                System.out.println("Cannot go forward.");
            }
        }
        // PRINT current page
        else if (cmd.equals("PRINT")) {
            System.out.println("Current Page: " + current); [cite_start]// [cite: 61, 72]
        }
        // Handle EXIT
        else if (cmd.equals("EXIT")) {
            System.out.println("Exiting browser.");
            break;
        }
        else {
            System.out.println("Invalid command.");
        }
    }
    sc.close();
  }
}
```

Problem 3: Print Queue System (Queue)

Java

```
import java.util.*;

public class PrintQueueSystem {
   public static void main(String[] args) {
      Queue<String> printQueue = new LinkedList<>(); [cite_start]// [cite: 84]
      Scanner sc = new Scanner(System.in); [cite_start]// [cite: 85]

      while (true) {
         System.out.print("Command (ADD <doc>/PRINT/EXIT): ");
         String cmd = sc.next();

         // Handle ADD
         if (cmd.equals("ADD")) {
            String doc = sc.next();
            printQueue.offer(doc); [cite_start]// [cite: 91]
            System.out.println("Added: " + doc);
         }
         // Handle PRINT
         else if (cmd.equals("PRINT")) {
            if (!printQueue.isEmpty()) {
               String doc = printQueue.poll(); [cite_start]// [cite: 92]
               System.out.println("Printing " + doc); [cite_start]// [cite: 105, 106]
            } else {
               System.out.println("No jobs left!"); [cite_start]// [cite: 107]
            }
         }
         // Handle EXIT
         else if (cmd.equals("EXIT")) {
            System.out.println("Shutting down printer.");
            break; [cite_start]// [cite: 96]
         }
         else {
            System.out.println("Invalid command.");
         }
      }
      sc.close();
   }
}
```

Problem 4: Customer Service Counter (Queue)

Java

```
import java.util.*;

public class CustomerServiceCounter {
   public static void main(String[] args) {
      Queue<String> queue = new LinkedList<>(); [cite_start]// [cite: 119]
      Scanner sc = new Scanner(System.in); [cite_start]// [cite: 120]

      while (true) {
         System.out.print("Command (ARRIVE <name>/SERVE/STATUS/EXIT): ");
         String cmd = sc.next();

         // Handle ARRIVE
         if (cmd.equals("ARRIVE")) {
            String name = sc.next();
            queue.add(name); [cite_start]// [cite: 126]
            System.out.println(name + " joined the queue.");
         }
         // Handle SERVE
         else if (cmd.equals("SERVE")) {
            if (!queue.isEmpty()) {
               String name = queue.remove(); [cite_start]// [cite: 126]
               System.out.println("Serving " + name); [cite_start]// [cite: 136, 137]
            } else {
               System.out.println("No customers waiting.");
            }
         }
         // Handle STATUS
         else if (cmd.equals("STATUS")) {
            System.out.println("Waiting: " + queue); [cite_start]// [cite: 126, 138]
         }
         // Handle EXIT
         else if (cmd.equals("EXIT")) {
            System.out.println("Closing counter.");
            break; [cite_start]// [cite: 126]
         }
         else {
```

```
            System.out.println("Invalid command.");
        }
    }
    sc.close();
  }
}
```

Problem 5: Expression Validator (Stack)

Java

```
import java.util.*;

public class ExpressionValidator {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter expression: ");
    String exp = sc.nextLine(); [cite_start]// [cite: 150]

    Stack<Character> stack = new Stack<>(); [cite_start]// [cite: 151]
    boolean isBalanced = true;

    // Traverse characters
    for (char ch : exp.toCharArray()) {
      // Push opening brackets
      if (ch == '(' || ch == '{' || ch == '[') {
        [cite_start]stack.push(ch); // [cite: 153]
      }
      // Pop for closing brackets and validate
      [cite_start]else if (ch == ')' || ch == '}' || ch == ']') { // [cite: 154]
        if (stack.isEmpty()) {
          isBalanced = false;
          break;
        }

        char top = stack.pop();
        if ((ch == ')' && top != '(') ||
          (ch == '}' && top != '{') ||
          (ch == ']' && top != '[')) {
```

```
                isBalanced = false;
                break;
            }
        }
    }

    // At end, print if balanced or not
    [cite_start]// [cite: 155]
    if (isBalanced && stack.isEmpty()) {
        System.out.println("Expression is Balanced"); [cite_start]// [cite: 160]
    } else {
        System.out.println("Expression is Not Balanced");
    }

    sc.close();
  }
}
```

Problem 6: Hospital Emergency Room (Priority Queue)

Java

```
import java.util.*;

[cite_start]// [cite: 169]
class Patient {
  String name; [cite_start]// [cite: 171]
  int priority; [cite_start]// [cite: 172]

  [cite_start]Patient(String n, int p) { // [cite: 173]
    name = n;
    priority = p;
  }

  @Override
  public String toString() {
    // Custom toString for printing the patient's name in the queue status
    return name + " (Priority " + priority + ")";
  }
```

```
}

public class EmergencyRoom {
   public static void main(String[] args) {
      [cite_start]// [cite: 176]
      [cite_start]// [cite: 177]
      // The comparator uses -p.priority to make it a max-heap based on priority
      PriorityQueue<Patient> pq =
         new PriorityQueue<>(Comparator.comparingInt(p -> -p.priority)); [cite_start]// [cite:
178]

      Scanner sc = new Scanner(System.in); [cite_start]// [cite: 179]

      while (true) {
         System.out.print("Command (ARRIVE <name> <priority>/TREAT/STATUS/EXIT): ");
         String cmd = sc.next();

         // Handle ARRIVE
         if (cmd.equals("ARRIVE")) {
            String name = sc.next();
            int priority = sc.nextInt();
            pq.add(new Patient(name, priority)); [cite_start]// [cite: 183]
            System.out.println(name + " arrived (Priority " + priority + ")");
         }
         // Handle TREAT
         else if (cmd.equals("TREAT")) {
            if (!pq.isEmpty()) {
               Patient p = pq.poll(); [cite_start]// [cite: 183]
               System.out.println("Treating " + p.name + " (Priority " + p.priority + ")");
[cite_start]// [cite: 193, 194]
            } else {
               System.out.println("No patients to treat.");
            }
         }
         // Handle STATUS
         else if (cmd.equals("STATUS")) {
            // We create a temporary queue to print, as peeking/polling modifies the heap
            System.out.println("Waiting: " + new ArrayList<>(pq)); [cite_start]// [cite: 183, 194]
         }
         // Handle EXIT
         else if (cmd.equals("EXIT")) {
            System.out.println("Closing ER.");
            break; [cite_start]// [cite: 184]
```

```
            }
            else {
                System.out.println("Invalid command.");
            }
        }
        sc.close();
    }
}
```