**2nd ASSIGNMENT**

**Data Structure and Algorithm (Java)**

**L1F23BSSE0389**

SCENARIO 01 (Undo And Redo Operations)

import java.util.Scanner;  
import java.util.Stack;  
  
public class TextEditorUndoRedo {  
  
 private static Scanner *input* = new Scanner(System.*in*);  
 private static Stack<String> *undo* = new Stack<>();  
 private static Stack<String> *redo* = new Stack<>();  
  
 public static void addText() {  
 System.*out*.print("Enter String: ");  
 String text = *input*.nextLine();  
 *undo*.push(text);  
 *redo*.clear();  
 System.*out*.println("Text added.");  
 }  
  
 public static void deleteText() {  
 if (!*undo*.isEmpty()) {  
 String text = *undo*.pop();  
 *redo*.push(text);  
 System.*out*.println("Last text deleted.");  
 } else {  
 System.*out*.println("Nothing to delete.");  
 }  
 }  
  
 public static void undoAction() {  
 if (!*undo*.isEmpty()) {  
 String text = *undo*.pop();  
 *redo*.push(text);  
 System.*out*.println("Undo performed.");  
 } else {  
 System.*out*.println("Nothing to undo.");  
 }  
 }  
  
 public static void redoAction() {  
 if (!*redo*.isEmpty()) {  
 String text = *redo*.pop();  
 *undo*.push(text);  
 System.*out*.println("Redo performed.");  
 } else {  
 System.*out*.println("Nothing to redo.");  
 }  
 }  
  
 public static void displayText() {  
 if (!*undo*.isEmpty()) {  
 String text = *undo*.peek();  
 System.*out*.println("Current Text: " + text);  
 } else {  
 System.*out*.println("No text available.");  
 }  
 }  
  
 public static void displayHistory() {  
 System.*out*.println("\n--- Undo Stack (History) ---");  
 if (*undo*.isEmpty()) {  
 System.*out*.println("No text history.");  
 } else {  
 for (String text : *undo*) {  
 System.*out*.println(text);  
 }  
 }  
  
 System.*out*.println("\n--- Redo Stack ---");  
 if (*redo*.isEmpty()) {  
 System.*out*.println("No redo history.");  
 } else {  
 for (String text : *redo*) {  
 System.*out*.println(text);  
 }  
 }  
 System.*out*.println();  
 }  
  
 public static void displayMenu() {  
 System.*out*.println("\nText Undo Redo Menu:");  
 System.*out*.println("1. Add Text");  
 System.*out*.println("2. Delete Text");  
 System.*out*.println("3. Undo");  
 System.*out*.println("4. Redo");  
 System.*out*.println("5. Display Current Text");  
 System.*out*.println("6. Display History");  
 System.*out*.println("7. Exit");  
 }  
  
 public static void main(String[] args) {  
  
 while (true) {  
 *displayMenu*();  
 System.*out*.print("Enter your choice: ");  
 int choice;  
 if (*input*.hasNextInt()) {  
 choice = *input*.nextInt();  
 *input*.nextLine();  
 } else {  
 System.*out*.println("Please enter a valid number.");  
 *input*.nextLine();  
 continue;  
 }  
  
 switch (choice) {  
 case 1:  
 *addText*();  
 break;  
 case 2:  
 *deleteText*();  
 break;  
 case 3:  
 *undoAction*();  
 break;  
 case 4:  
 *redoAction*();  
 break;  
 case 5:  
 *displayText*();  
 break;  
 case 6:  
 *displayHistory*();  
 break;  
 case 7:  
 System.*out*.println("Exiting...");  
 return;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 }  
 }  
 }  
  
}

Scenario 02 (Helpdesk Queue System)

import java.util.LinkedList;  
import java.util.Queue;  
import java.util.Scanner;  
  
class Customer {  
 String name;  
 String issueDescription;  
  
 Customer(String name, String issueDescription) {  
 this.name = name;  
 this.issueDescription = issueDescription;  
 }  
}  
  
class HelpdeskQueue {  
 private Queue<Customer> queue = new LinkedList<>();  
  
 public void enqueue(String name, String issueDescription) {  
 if (name.isBlank() || issueDescription.isBlank()) {  
 System.*out*.println("Error: Name and Issue Description cannot be blank.");  
 return;  
 }  
 Customer customer = new Customer(name, issueDescription);  
 queue.add(customer);  
 System.*out*.println("Customer added to the queue.");  
 }  
  
 public void dequeue() {  
 if (queue.isEmpty()) {  
 System.*out*.println("Queue is empty. No customer to serve.");  
 } else {  
 Customer servedCustomer = queue.poll();  
 System.*out*.println("Serving Customer: " + servedCustomer.name);  
 System.*out*.println("Issue: " + servedCustomer.issueDescription);  
 }  
 }  
  
 public void peek() {  
 if (queue.isEmpty()) {  
 System.*out*.println("Queue is empty.");  
 } else {  
 Customer nextCustomer = queue.peek();  
 System.*out*.println("Next Customer: " + nextCustomer.name);  
 System.*out*.println("Issue: " + nextCustomer.issueDescription);  
 }  
 }  
  
 public boolean isEmpty() {  
 return queue.isEmpty();  
 }  
  
 public void displayQueue() {  
 if (queue.isEmpty()) {  
 System.*out*.println("Queue is empty.");  
 } else {  
 System.*out*.println("Customers in Queue:");  
 for (Customer customer : queue) {  
 System.*out*.println("- " + customer.name + ": " + customer.issueDescription);  
 }  
 }  
 }  
}  
  
public class HelpdeskSystem {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 HelpdeskQueue helpdesk = new HelpdeskQueue();  
  
 while (true) {  
 System.*out*.println("\nHelpdesk Menu:");  
 System.*out*.println("1. Add Customer");  
 System.*out*.println("2. Serve Customer");  
 System.*out*.println("3. View Next Customer");  
 System.*out*.println("4. Check if Queue is Empty");  
 System.*out*.println("5. Display All Customers");  
 System.*out*.println("6. Exit");  
 System.*out*.print("Enter your choice: ");  
  
 int choice = scanner.nextInt();  
 scanner.nextLine();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter Customer Name: ");  
 String name = scanner.nextLine();  
 System.*out*.print("Enter Issue Description: ");  
 String issue = scanner.nextLine();  
 helpdesk.enqueue(name, issue);  
 break;  
 case 2:  
 helpdesk.dequeue();  
 break;  
 case 3:  
 helpdesk.peek();  
 break;  
 case 4:  
 if (helpdesk.isEmpty()) {  
 System.*out*.println("Queue is empty.");  
 } else {  
 System.*out*.println("Queue is NOT empty.");  
 }  
 break;  
 case 5:  
 helpdesk.displayQueue();  
 break;  
 case 6:  
 System.*out*.println("Exiting Helpdesk System. Goodbye!");  
 scanner.close();  
 return;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 }  
 }  
 }  
}

Scenario 03 (Postfix Evaluator)

import java.util.Stack;  
import java.util.Scanner;  
  
public class PostfixEvaluator {  
  
 public static int evaluatePostfix(String expression) {  
 Stack<Integer> stack = new Stack<>();  
 String[] tokens = expression.split(" ");  
  
 for (String token : tokens) {  
 if (*isNumeric*(token)) {  
 stack.push(Integer.*parseInt*(token));  
 }  
 else if (*isOperator*(token)) {  
 if (stack.size() < 2) {  
 throw new IllegalArgumentException("Error: Not enough operands.");  
 }  
 int b = stack.pop();  
 int a = stack.pop();  
 int result = *applyOperation*(a, b, token);  
 stack.push(result);  
 }  
 else {  
 throw new IllegalArgumentException("Error: Invalid token '" + token + "'");  
 }  
 }  
  
 if (stack.size() != 1) {  
 throw new IllegalArgumentException("Error: Invalid postfix expression.");  
 }  
  
 return stack.pop();  
 }  
  
 private static boolean isNumeric(String str) {  
 try {  
 Integer.*parseInt*(str);  
 return true;  
 } catch (NumberFormatException e) {  
 return false;  
 }  
 }  
  
 private static boolean isOperator(String str) {  
 return str.equals("+") || str.equals("-") || str.equals("\*") || str.equals("/");  
 }  
  
 private static int applyOperation(int a, int b, String operator) {  
 switch (operator) {  
 case "+":  
 return a + b;  
 case "-":  
 return a - b;  
 case "\*":  
 return a \* b;  
 case "/":  
 if (b == 0) {  
 throw new ArithmeticException("Error: Division by zero.");  
 }  
 return a / b;  
 default:  
 throw new IllegalArgumentException("Unknown operator: " + operator);  
 }  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println("Enter a postfix expression (tokens separated by space):");  
 String expression = scanner.nextLine();  
  
 try {  
 int result = *evaluatePostfix*(expression);  
 System.*out*.println("Result: " + result);  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 scanner.close();  
 }  
}

Scenario 04 (Runway Queue System)

import java.util.LinkedList;  
import java.util.Queue;  
import java.util.Scanner;  
  
class Aircraft {  
 String flightNumber;  
 String destination;  
  
 Aircraft(String flightNumber, String destination) {  
 this.flightNumber = flightNumber;  
 this.destination = destination;  
 }  
}  
  
class RunwaySystem {  
 private Queue<Aircraft> queue = new LinkedList<>();  
  
 private boolean flightExists(String flightNumber) {  
 for (Aircraft aircraft : queue) {  
 if (aircraft.flightNumber.equalsIgnoreCase(flightNumber)) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 public void addFlight(String flightNumber, String destination) {  
 if (flightNumber.isBlank() || destination.isBlank()) {  
 System.*out*.println("Error: Flight number and destination cannot be blank.");  
 return;  
 }  
 if (flightExists(flightNumber)) {  
 System.*out*.println("Error: Flight number already exists.");  
 return;  
 }  
 Aircraft newFlight = new Aircraft(flightNumber, destination);  
 queue.add(newFlight);  
 System.*out*.println("Flight added to queue successfully.");  
 }  
  
 public void authorizeTakeoff() {  
 if (queue.isEmpty()) {  
 System.*out*.println("No flights waiting for takeoff.");  
 } else {  
 Aircraft flight = queue.poll();  
 System.*out*.println("Flight " + flight.flightNumber + " to " + flight.destination + " is authorized for takeoff!");  
 }  
 }  
  
 public void peekNextFlight() {  
 if (queue.isEmpty()) {  
 System.*out*.println("No flights waiting.");  
 } else {  
 Aircraft nextFlight = queue.peek();  
 System.*out*.println("Next flight: " + nextFlight.flightNumber + " heading to " + nextFlight.destination);  
 }  
 }  
  
 public void displayQueue() {  
 if (queue.isEmpty()) {  
 System.*out*.println("No flights waiting.");  
 } else {  
 System.*out*.println("Flights in Queue:");  
 for (Aircraft aircraft : queue) {  
 System.*out*.println("- " + aircraft.flightNumber + " -> " + aircraft.destination);  
 }  
 }  
 }  
}  
  
public class RunwayQueueSystem {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 RunwaySystem runway = new RunwaySystem();  
  
 while (true) {  
 System.*out*.println("\nRunway System Menu:");  
 System.*out*.println("1. Add Flight");  
 System.*out*.println("2. Authorize Takeoff");  
 System.*out*.println("3. View Next Flight");  
 System.*out*.println("4. Display All Flights");  
 System.*out*.println("5. Exit");  
 System.*out*.print("Enter your choice: ");  
  
 int choice = scanner.nextInt();  
 scanner.nextLine();   
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter Flight Number: ");  
 String flightNumber = scanner.nextLine();  
 System.*out*.print("Enter Destination: ");  
 String destination = scanner.nextLine();  
 runway.addFlight(flightNumber, destination);  
 break;  
 case 2:  
 runway.authorizeTakeoff();  
 break;  
 case 3:  
 runway.peekNextFlight();  
 break;  
 case 4:  
 runway.displayQueue();  
 break;  
 case 5:  
 System.*out*.println("Exiting Runway System. Goodbye!");  
 scanner.close();  
 return;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 }  
 }  
 }  
}