Part 1: Problem-Solving using Stacks and Maps

CallTracer Class:

```java
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.HashMap;
import java.util.Map;

public class CallTracer {
    private Deque<String> callStack;
    private Map<String, Integer> callCounts;

    public CallTracer() {
        callStack = new ArrayDeque<>();
        callCounts = new HashMap<>();
    }

    public void enter(String methodName){
        //if we cannot find the thing we found in the callCounts hasmap, create
an element with the stirng as the key
        if(!callCounts.containsKey(methodName)){
            callCounts.put(methodName, 1);
        }
        else if(callCounts.containsKey(methodName)){
            callCounts.put(methodName, callCounts.get(methodName).intValue() +
1);
        }
        System.out.println("\t".repeat(callStack.size()) + "Entering: " +
methodName);
        callStack.push(methodName);
    }

    public void exit(String methodName){
        if(callStack == null || callStack.peek() != methodName){
            System.out.println("Error");
        }
        else{
            callStack.pop();
        }
        System.out.println("\t".repeat(callStack.size()) + "Exiting: " +
methodName);
    }
    public void printSummary(){
        System.out.println();
        System.out.println("Call Summary :");
        for(Map.Entry<String, Integer> set : callCounts.entrySet()){
            System.out.println(set.getKey() + " was called " + set.getValue() + "
time(s)");
        }
    }

    public static void main(String[] args) {
        CallTracer tracer = new CallTracer();
        tracer.enter("main") ;
        tracer.enter("methodA") ;
        tracer.enter("methodB") ;
        tracer.exit("methodB") ;
        tracer.enter("methodC") ;
        tracer.exit("methodC") ;
        tracer.exit("methodA") ;
        tracer.enter("methodA") ; // Call methodA again
        tracer.enter("methodB") ;
        tracer.exit("methodB") ;
        tracer.exit("methodA") ;
        tracer.exit("main");
        tracer.printSummary();
    }
}
```

Part 2: Applications – Customer Support Ticket System

Ticket Class:

```
enum Status{
    OPEN, CLOSED;
}
public class Ticket {
    private int ticketId;
    private String customerName;
    private String issueDescription;
    private Status status;

    public Ticket() {
        ticketId = 0;
        customerName = null;
        issueDescription = null;
        status = Status.OPEN;
    }

    public Ticket(int ticketId, String customerName, String issueDescription) {
        this.ticketId = ticketId;
        this.customerName = customerName;
        this.issueDescription = issueDescription;
        status = Status.OPEN;
    }

    public int getTicketId() {
        return ticketId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public String getIssueDescription() {
        return issueDescription;
    }

    public Status getStatus() {
        return status;
    }

    public void setStatus(Status status) {
        this.status = status;
    }

    @Override
    public String toString() {
        return "Ticket [ticketId=" + ticketId + ", customerName=" + customerName
+ ", issueDescription="
                + issueDescription + ", status=" + status + "]";
    }


}
```

SupportSystem Class:

```
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Queue;

public class SupportSystem {
    private Queue<Ticket> openTickets;
    private Map<Integer, Ticket> allTickets;
    private Map<String, Integer> ticketsPerCustomer;
    private int nextTicketId = 1;

    public SupportSystem(){
        openTickets = new LinkedList<>();
        allTickets = new HashMap<>();
        ticketsPerCustomer = new HashMap<>();
    }

    public void createTicket(String customer, String issue){
        Ticket newTicket = new Ticket(nextTicketId, customer, issue);
        openTickets.add(newTicket);
        allTickets.put(nextTicketId, newTicket);
        if(!ticketsPerCustomer.containsKey(customer)){
            ticketsPerCustomer.put(customer, 1);
            nextTicketId++;
        }
        else if(ticketsPerCustomer.containsKey(customer)){
            ticketsPerCustomer.put(customer,
ticketsPerCustomer.get(customer).intValue() + 1);
            nextTicketId++;
        }

    }

    public Ticket processNextTicket(){
        if(openTickets == null){
            return null;
        }
        else{
            Ticket temp = openTickets.poll();
            temp.setStatus(Status.CLOSED);
            return temp;
        }
    }

    public Ticket getTicketById(int ticketId){
        return allTickets.get(ticketId);
    }

    public void printCustomerSummary(){
        System.out.println("Header thing");
        for(Map.Entry<String, Integer> set : ticketsPerCustomer.entrySet()){
            System.out.println("Name: " + set.getKey() + " Number of tickets: " +
set.getValue());
        }
    }

    public static void main(String[] args) {
        SupportSystem system = new SupportSystem();

        system.createTicket("Alice ", " Printer is not working ");
        system.createTicket("Bob ", " Cannot connect to WiFi ");
        system.createTicket("Alice ", " Monitor is flickering ");
        system.createTicket("Charlie ", " Software license expired");
        system.createTicket("Bob ", " Mouse is broken ");

        System.out.println(" --- Processing Next Ticket ---");
        Ticket processedTicket1 = system.processNextTicket();
        System.out.println("Processed : " + processedTicket1 );
        System.out.println("----------------------------\n");


        System.out.println("--- Looking up Ticket ID 3 ---");
        Ticket foundTicket = system.getTicketById (3);
        System.out.println(" Found : " + foundTicket );
```

```java
        System.out.println("----------------------------\n");

        System.out.println("--- Processing Next Ticket ---");
        Ticket processedTicket2 = system.processNextTicket();
        System.out.println("Processed : " + processedTicket2);
        System.out.println("----------------------------\n");

        System.out.println("--- Looking up Closed Ticket ID 1---");
        Ticket closedTicket = system.getTicketById(1);
        System.out.println("Found: " + closedTicket);
        System.out.println("-----------------------------------\n");

        system.printCustomerSummary();

    }
}
```

Part 3: (Optional): Application – Text Editor with Undo/Redo and Versioning

TextEditor Class:

```java
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.HashMap;
import java.util.Map;

public class TextEditor {
    private StringBuilder text;
    private Deque<String> undoStack;
    private Deque<String> redoStack;
    private Map<String, String> versions;

    public TextEditor(){
        text = new StringBuilder();
        undoStack = new ArrayDeque<>();
        redoStack = new ArrayDeque<>();
        versions = new HashMap<>();
    }

    public void type(String newText){
        //Push the current state (text.toString()) onto the undoStack
        undoStack.push(text.toString());
        //Append newText to the StringBuilder
        text.append(newText);
        //Clear the redoStack
        redoStack.clear();
    }

    public void undo(){
        //Undoing an action moves a state from the undoStack to the redoStack
        redoStack.push(text.toString());
        text = new StringBuilder();
        text.append(undoStack.pop());
    }

    public void redo(){
        //Redoing an action moves a state moves back from the redoStack to the
undoStack
        undoStack.push(text.toString());
        text = new StringBuilder();
        text.append(redoStack.pop());

    }

    public void saveVersion(String name){
        //Puts the current text (text.toString()) into the versions map with name
as the key
        versions.put(name, text.toString());
    }

    public void loadVersion(String name){
        //Find the saved text in the versions map
        //If it exists, push the current text state onto the undoStack before
making a change
        if(versions.containsKey(name)){
            undoStack.push(text.toString());
            //Replace the current text with the loaded version
            text = new StringBuilder();
            text.append(versions.get(name));
            //Clear the redoStack, as loading a version starts a new history
branch
            redoStack.clear();
        }
        else{
            System.err.println("Error");
        }
    }

    public String getText(){
        //Getter that returns text.toString()
        return text.toString();
    }

    public static void main(String[] args) {
        TextEditor editor = new TextEditor();
        editor.type("Hello ");
        editor.type("World");
        System.out.println(editor.getText());
        editor.saveVersion("v1");
        editor.type(" from Java!"); System.out.println(editor.getText());
        editor.undo(); System.out.println(editor.getText());
        editor.loadVersion("v1"); System.out.println(editor.getText());
        editor.undo(); System.out.println(editor.getText());
    }
}
```

TextEditorGUI Class:

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;

public class TextEditorGUI {
    // Our backend logic from Exercise 2
    private final TextEditor editor = new TextEditor();
    private JFrame frame;
    private JTextArea textArea;
    private JTextField inputField;
    private JTextField versionNameField;

    public TextEditorGUI() {
        // Frame and Main Panel Setup
        frame = new JFrame(" Simple Text Editor ");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 400);
        frame.setLayout(new BorderLayout());

        // Text Area (Top)
        textArea = new JTextArea();
        textArea.setEditable(false);
        textArea.setFont(new Font("Monospaced", Font.PLAIN, 14));
        JScrollPane scrollPane = new JScrollPane(textArea);
        frame.add(scrollPane, BorderLayout.CENTER);

        // Controls Panel (Bottom)
        JPanel controlsPanel = new JPanel(new GridLayout(3, 1, 5, 5));

        // Typing controls
        JPanel typePanel = new JPanel(new BorderLayout(5, 0));
        inputField = new JTextField();
        JButton typeButton = new JButton("Append Text");
        typePanel.add(new JLabel("Text to Append: "), BorderLayout.WEST);
        typePanel.add(inputField, BorderLayout.CENTER);
        typePanel.add(typeButton, BorderLayout.EAST);

        // Undo / Redo controls
        JPanel undoRedoPanel = new JPanel(new FlowLayout());
        JButton undoButton = new JButton("Undo");
        JButton redoButton = new JButton("Redo");
        undoRedoPanel.add(undoButton);
        undoRedoPanel.add(redoButton);

        // Versioning controls
        JPanel versionPanel = new JPanel(new BorderLayout(5, 0));
        versionNameField = new JTextField();
        JButton saveButton = new JButton(" Save Version ");
        JButton loadButton = new JButton(" Load Version ");
        versionPanel.add(new JLabel("Version Name: "), BorderLayout.WEST);
        versionPanel.add(versionNameField, BorderLayout.CENTER);
        JPanel versionButtons = new JPanel(new GridLayout(1, 2));
        versionButtons.add(saveButton);
        versionButtons.add(loadButton);
        versionPanel.add(versionButtons, BorderLayout.EAST);
        controlsPanel.add(typePanel);
        controlsPanel.add(undoRedoPanel);
        controlsPanel.add(versionPanel);
        frame.add(controlsPanel, BorderLayout.SOUTH);

        // Action Listeners
        typeButton.addActionListener(e -> {
            // TODO : Get text from inputField and call the editor’s type method.
            editor.type(inputField.getText());
            updateTextArea();
            inputField.setText("");
        });
        undoButton.addActionListener(e -> {
            // TODO : Call the editor’s undo method.
            editor.undo();
            updateTextArea();
        });
        redoButton.addActionListener(e -> {
            // TODO : Call the editor’s redo method.
            editor.redo();
            updateTextArea();
        });
        saveButton.addActionListener(e -> {
            // TODO : Get version name and call the editor’s saveVersion method.
            String name = versionNameField.getText();
            editor.saveVersion(name);
        });
        loadButton.addActionListener(e -> {
            // TODO : Get version name and call the editor’s loadVersion method.
            String name = versionNameField.getText();
            editor.loadVersion(name);
            updateTextArea();
        });
        // Finalize
        updateTextArea(); // Initial state
        frame.setLocationRelativeTo(null); // Center thewindow
        frame.setVisible(true);
    }

    private void updateTextArea() {
        textArea.setText(editor.getText());
    }

    public static void main(String[] args) {
        // Ensure the GUI is created on the Event Dispatch Thread
        SwingUtilities.invokeLater(TextEditorGUI::new);
    }
}
```