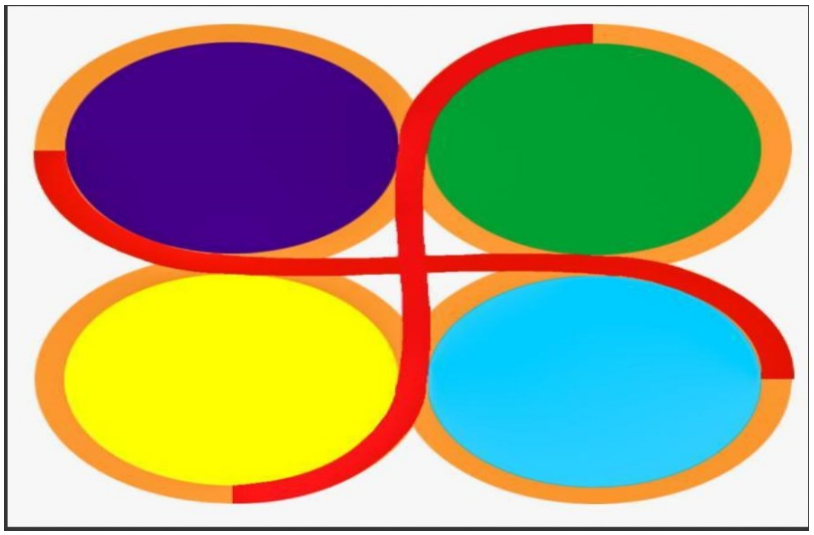


Government Polytechnic, Ahmedabad

Subject Name: Advance Object-Oriented Programming

Semester: 4th

Project Title: ATM Simulation System

Project report prepared by:

Name: Yash Kayastha

Enrolment no: 226170307072(A3)

Batch: 2022-25

About Project:

ATM Simulation System is a kind of personal banking system where users can perform various transactions like withdrawals, deposits, and checking the balance of the account in just one click. It has a Graphical User Interface (GUI) to make the process user-friendly.

External Libraries/packages:

- activation-1.1.jar :-

The activation-1.1.jar file is a Java Archive (JAR) that typically contains classes related to JavaBeans Activation Framework (JAF). This framework is used to determine the type of a file or data stream, and initiate an action or process based on that type. It enables applications to interact with various MIME types, allowing for dynamic handling of content types such as emails, documents, and multimedia files. By including activation-1.1.jar in a Java project's classpath, developers can utilize its functionalities to manage content types effectively, enhancing the versatility and interoperability of their applications.

- javax-mail-1.6.2.jar :-

The javax-mail-1.6.2.jar file is a Java Archive (JAR) that contains classes related to JavaMail API, a set of abstract APIs that model a mail system. This API enables Java applications to send, receive, and manipulate email messages programmatically. By including javax-mail-1.6.2.jar in a Java project's classpath, developers can leverage its functionalities to implement email-related features such as composing and sending emails, accessing and managing mailboxes, and handling attachments. This library provides a convenient and standardized way to integrate email capabilities into Java applications, facilitating communication and interaction with users via email channels.

IDE used :

Intelij :-

IntelliJ IDEA, commonly referred to as IntelliJ, is a popular integrated development environment (IDE) developed by JetBrains. It provides comprehensive support for various programming languages such as Java, Kotlin, Groovy, Scala, and more. IntelliJ offers a wide range of features to streamline the software development process, including code editing, debugging, version control integration, and advanced code analysis tools.

One of the standout features of IntelliJ is its intelligent code assistance, powered by sophisticated code analysis algorithms. It offers context-aware code completion, suggesting relevant code snippets, methods, and variables as you type. This helps developers write code faster and with fewer errors.

In-built Libraries/packages:

➢ javax.swing.*:

The javax.swing.* package in Java provides a set of graphical user interface (GUI) components and utilities for building desktop applications. These components include buttons, text fields, labels, menus, and more. Swing offers a rich set of features for creating interactive and visually appealing user interfaces, facilitating the development of cross-platform applications.

➢ java.awt.*:

The java.awt.* package contains classes and interfaces for building graphical user interfaces (GUIs) and handling basic input/output operations. It provides fundamental components such as windows, dialogs, buttons, and graphics objects. AWT (Abstract Window Toolkit) serves as the foundation for building GUI applications in Java, offering platform-independent functionality for creating and managing graphical elements.

➢ java.io.*:

The java.io.* package in Java provides classes and interfaces for performing input and output operations. It includes various streams for reading from and writing to files, as well as classes for working with input and output streams, readers, writers, and file system operations. Java I/O streams enable developers to manipulate data streams efficiently, facilitating file handling and data processing tasks.

➢ java.util.*:

The java.util.* package contains utility classes and data structures for performing common tasks such as data manipulation, collection management, date and time operations, and more. It includes classes like ArrayList, HashMap, and StringTokenizer for working with collections and managing data efficiently. Additionally, the java.util package provides support for handling dates, times, and calendars through classes like Date, Calendar, and DateFormat, making it a versatile toolkit for various programming tasks.

Mostly used in-build classes:

➢ JDialog:

JDialog is a subclass of Dialog in Java Swing. It represents a dialog window that can be used to display information, messages, or obtain user input in a modal or modeless manner. Developers can customize the appearance and behavior of a JDialog to suit their application's requirements.

➢ JFrame:

JFrame is a top-level container class in Java Swing used to create and manage a windowed GUI application. It provides a frame for holding and organizing other Swing components such as buttons, text fields, and panels. JFrame serves as the main window of a Swing application, providing a title bar, borders, and controls for resizing and closing the window.

➢ JPasswordField:

JPasswordField is a Swing component used to create a text field for entering passwords or other sensitive information. It behaves like a regular text field but obscures the characters entered by displaying them as asterisks or dots. This helps enhance security by preventing onlookers from seeing the actual characters typed by the user.

➢ JButton:

JButton is a Swing component used to create a clickable button in a GUI application. It can display text, icons, or both, and triggers an action when clicked by the user. Developers can add event listeners to JButton instances to perform specific tasks or handle user interactions.

➢ JLabel:

JLabel is a Swing component used to display text or an image on a GUI window. It is commonly used for providing descriptive text or captions for other components such as text fields, buttons, or images. JLabel can also be used to display dynamic information that may change during the execution of the application.

➢ FileReader:

FileReader is a class in Java IO used to read character data from a file. It provides methods for reading characters from a file stream, allowing developers to retrieve textual data stored in a file. FileReader is often used in conjunction with BufferedReader for efficient reading of large text files.

- FileWriter:

FileWriter is a class in Java IO used to write character data to a file. It provides methods for writing characters to a file stream, enabling developers to create or append text to a file. FileWriter is commonly used for writing textual data such as logs, configuration files, or user-generated content to disk.

- BufferedReader:

BufferedReader is a class in Java IO used to read text from a character-input stream with efficient reading of characters, arrays, and lines. It provides methods for reading data line by line from a Reader, such as FileReader, InputStreamReader, or StringReader. BufferedReader is often used to improve the performance of reading text files by buffering input from the underlying stream.

- IOException:

IOException is an exception class in Java IO used to handle input/output errors that may occur during file operations. It is a checked exception, meaning that it must be either caught or declared in the method signature using a throws clause. IOException is commonly used to handle situations such as file not found, permission denied, or disk full errors.

- ArrayList:

ArrayList is a class in the java.util package that implements the List interface. It provides a dynamic array-like data structure that can dynamically resize itself to accommodate elements. ArrayList is commonly used for storing and manipulating collections of objects in memory, offering features such as fast random access, efficient insertion and removal, and compatibility with Java's generic types.

Java Files:

- Main.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

/**
 * The Main class represents the main GUI window of the ATM simulation
application.
 */
public class Main extends JFrame {
    private JLabel cardNumLabel, pinNumLabel, banner;
    private JButton signIn, signUp, clear;
    private JTextField cardNumInput;
    private JPasswordField pinNumInput;
    private Validate validator;
    private List<UserData> userDataList;

    /**
     * Constructor for the Main class.
     * Initializes the main GUI window.
     */
    public Main() {
        setTitle("ATM Simulation APP");
        setSize(520, 360);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(null);
        setResizable(false);
        setIconImage(new ImageIcon("src/icon.png").getImage());
        initComponents();
    }

    /**
     * Initializes all the components of the main GUI window.
     */
    private void initComponents() {
        // Fetch user data from the file
        userDataList = UserData.fetchUserData();

        banner = new JLabel("Welcome To ATM");
        banner.setHorizontalAlignment(SwingConstants.CENTER);
        banner.setOpaque(true);
        banner.setBackground(Color.gray);
        banner.setBounds(20, 20, 450, 50);
        add(banner);

        cardNumLabel = new JLabel("Enter Card No.:");
```

```
        cardNumLabel.setBounds(20, 100, 120, 25);
        add(cardNumLabel);

        cardNumInput = new JTextField(12);
        cardNumInput.setBounds(150, 100, 320, 25);
        add(cardNumInput);

        pinNumLabel = new JLabel("Enter PIN No.:");
        pinNumLabel.setBounds(20, 150, 120, 25);
        add(pinNumLabel);

        pinNumInput = new JPasswordField(12);
        pinNumInput.setBounds(150, 150, 320, 25);
        add(pinNumInput);

        signIn = new JButton("Sign In");
        signIn.setBounds(150, 200, 100, 30);
        add(signIn);

        signUp = new JButton("Sign Up");
        signUp.setBounds(260, 200, 100, 30);
        add(signUp);

        clear = new JButton("Clear");
        clear.setBounds(370, 200, 100, 30);
        add(clear);

        // ActionListener for clear button
        clear.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Clear the card number and PIN input fields
                cardNumInput.setText("");
                pinNumInput.setText("");
            }
        });

        // ActionListener for sign up button
        signUp.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Open sign up dialog
                SignUp signUpDialog = new SignUp(Main.this,
userDataList);
                signUpDialog.setVisible(true);
            }
        });

        // ActionListener for sign in button
        signIn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Fetch user data
                userDataList = UserData.fetchUserData();
```

```
                validator = new Validate();
                String cardNumber = cardNumInput.getText();
                String pin = new String(pinNumInput.getPassword());

                if (validator.isCard(cardNumber) &&
validator.isPin(pin)) {
                    if (validator.authenticateUser(cardNumber, pin,
userDataList)) {
                        String eCheck = UserData.getData(cardNumber,
14, userDataList);
                        if ("Yes".equals(eCheck)) {
                            String email =
UserData.getData(cardNumber, 3, userDataList);
                            if (validator.otpVerify(email, Main.this))
{
                                JOptionPane.showMessageDialog(null,
"Authentication Successful!");
                                // Send login confirmation email
                                String
msg=Emsg.logAlert(UserData.getData(cardNumber, 4, userDataList));
                                new EmailService().sendEmail(email,
"Login Notification",msg,null,true);
                                cardNumInput.setText("");
                                pinNumInput.setText("");
                                // Open sign in dialog
                                SignIn signInDialog = new
SignIn(Main.this, cardNumber, userDataList);
                                signInDialog.setVisible(true);
                            } else {
                                cardNumInput.setText("");
                                pinNumInput.setText("");
                                JOptionPane.showMessageDialog(null,
"Email verification failed.");
                            }
                        } else {
                            cardNumInput.setText("");
                            pinNumInput.setText("");
                            JOptionPane.showMessageDialog(null, "Login
Successful!");
                            // Open sign in dialog
                            SignIn signInDialog = new
SignIn(Main.this, cardNumber, userDataList);
                            signInDialog.setVisible(true);
                        }
                    } else {
                        cardNumInput.setText("");
                        pinNumInput.setText("");
                        JOptionPane.showMessageDialog(null,
"Authentication Failed. Invalid Card Number or PIN.");
                    }
                } else {
                    cardNumInput.setText("");
                    pinNumInput.setText("");
                    JOptionPane.showMessageDialog(null, "Please Enter
```

```
Valid Card Number And PIN....");
                }
            }
        });
    }

    /**
     * The main method to start the application.
     * @param args Command-line arguments (not used)
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Main().setVisible(true);
            }
        });
    }

    /**
     * Static block to display a splash screen on application start.
     */
    static {
        JWindow splashScreen = new JWindow();
        splashScreen.setSize(520, 360);

        ImageIcon splashIcon = new
ImageIcon(Main.class.getResource("/icon.png"));
        JLabel splashLabel = new JLabel(splashIcon);
        splashScreen.add(splashLabel);

        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        splashScreen.setLocation(
                (screenSize.width - splashScreen.getWidth()) / 2,
                (screenSize.height - splashScreen.getHeight()) / 2
        );

        splashScreen.setVisible(true);

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        splashScreen.dispose();
    }

    /**
     * Restarts the application.
     */
    private void restartApplication() {
        dispose(); // Close the current window
        SwingUtilities.invokeLater(new Runnable() {
```

```
            public void run() {
                new Main().setVisible(true); // Launch a new instance
of Main
            }
        });
    }
}
```

- SignIn.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

/**
 * The SignIn class represents the dialog window for signed-in users.
 */
class SignIn extends JDialog {
    private String cardNum;
    private List<UserData> userDataList;
    private JButton depositB, withdrawB, changePinB, fastWithdrawalB,
requestBalanceB, requestStatementB,trasB, logOutB;
    private JLabel welcomeLabel;
    private JPanel buttonsPanel, welcomePanel;

    /**
     * Constructor for SignIn class.
     *
     * @param owner       The owner frame of the dialog.
     * @param cardNum     The card number of the signed-in user.
     * @param userDataList The list of user data.
     */
    public SignIn(Frame owner, String cardNum, List<UserData>
userDataList) {
        super(owner, true);
        setTitle("Welcome to Your Account");
        setResizable(false);
        setSize(520, 400); // Adjusted size to accommodate new layout
        setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());
        setIconImage(new ImageIcon("src/icon.png").getImage());
        this.cardNum = cardNum;
        this.userDataList = userDataList;
        initComponents();
    }

    /**
     * Initializes all the components of the SignIn dialog.
     */
    private void initComponents() {
        // Welcome label panel
```

```
        welcomePanel = new JPanel(new BorderLayout());
        welcomeLabel = new JLabel("Welcome, " +
UserData.getData(cardNum, 4, userDataList), SwingConstants.CENTER);
        welcomeLabel.setFont(new Font("Verdana", Font.BOLD, 16));
        welcomePanel.add(welcomeLabel, BorderLayout.CENTER);
        welcomePanel.setBorder(BorderFactory.createEmptyBorder(10, 10,
10, 10));

        // Buttons panel
        buttonsPanel = new JPanel(new GridLayout(4, 2, 10, 10));
        buttonsPanel.setBorder(BorderFactory.createEmptyBorder(10, 10,
10, 10));

        depositB = new JButton("Deposit");
        withdrawB = new JButton("Withdraw");
        changePinB = new JButton("Change PIN");
        fastWithdrawalB = new JButton("Fast Withdrawal");
        requestBalanceB = new JButton("Request Balance");
        requestStatementB = new JButton("Request Statement");
        trasB=new JButton("Transfer Money");
        logOutB = new JButton("Log Out");

        // Adding buttons to the panel
        buttonsPanel.add(depositB);
        buttonsPanel.add(withdrawB);
        buttonsPanel.add(changePinB);
        buttonsPanel.add(fastWithdrawalB);
        buttonsPanel.add(requestBalanceB);
        buttonsPanel.add(requestStatementB);
        buttonsPanel.add(trasB);
        buttonsPanel.add(logOutB);

        // Adding panels to the dialog
        add(welcomePanel, BorderLayout.NORTH);
        add(buttonsPanel, BorderLayout.CENTER);

        depositB.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Deposit d = new Deposit(((Frame)
SignIn.this.getParent()), cardNum, userDataList);
                d.setVisible(true);
            }
        });
        withdrawB.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Withdrawal w = new Withdrawal(((Frame)
SignIn.this.getParent()), cardNum, userDataList);
                w.setVisible(true);
            }
        });
        requestBalanceB.addActionListener(new ActionListener() {
            @Override
```

```
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "Your current
Amount is ₹" + UserData.getData(cardNum, 5, userDataList));
            }
        });
        changePinB.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                ChangePIN c = new ChangePIN(((Frame)
SignIn.this.getParent()), cardNum, userDataList);
                c.setVisible(true);
            }
        });
        fastWithdrawalB.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                FastWithdrawal f = new FastWithdrawal(((Frame)
SignIn.this.getParent()), cardNum, userDataList);
                f.setVisible(true);
            }
        });
        requestStatementB.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                GetStatement gs = new GetStatement(((Frame)
SignIn.this.getParent()), cardNum, userDataList);
                gs.setVisible(true);
            }
        });
        trasB.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                TMoney t=new TMoney(((Frame) SignIn.this.getParent()),
cardNum, userDataList);
                t.setVisible(true);
            }
        });
        logOutB.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(SignIn.this, "Logging
out, please wait a few seconds...");

                // Create a new thread to handle the logout delay
                Thread logoutThread = new Thread(new Runnable() {
                    @Override
                    public void run() {
                        try {
                            // Set the thread priority to low

Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
                            // Wait for 5 seconds
                            Thread.sleep(5000);
                        } catch (InterruptedException ex) {
```

```
                            ex.printStackTrace();
                        } finally {
                            // Dispose the dialog in the Event
Dispatch Thread
                            SwingUtilities.invokeLater(new Runnable()
{
                                @Override
                                public void run() {
                                    dispose();
                                }
                            });
                        }
                    }
                });

                // Start the thread
                logoutThread.start();
            }
        });
    }


}
```

- SignUp.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

class SignUp extends JDialog {
    private JPanel personalDetailsPanel, cardDetailsPanel;
    private JLabel banner, page1, page2;
    private JLabel nameL, fatherNameL, dobL, emailL, cityL, stateL,
pinCodeL, addressL, genderL, maritalL;
    private JTextField nameI, fatherNameI, dobI, emailI, cityI,
pinCodeI, stateI, addressI;
    private JRadioButton maleRadioButton, femaleRadioButton,
otherRadioButton, marriedRadioButton, singleRadioButton,
noToSayRadioButton;
    private JButton cancel, next, back, submit, cancel1;
    private JLabel cardNumLabel, pinNumLabel, startAmountL;
    private JTextField cardNumInput, pinNumInput, startAmountI;
    private JCheckBox eCheckL;
    private Set<String> registeredCardNumbers;
    private Validate v = new Validate();
    private String eCheck,accountNumber;
```

```
    private List<UserData> userDataList;
    public SignUp(Frame owner,List<UserData> userDataList) {
        super(owner, true);
        setTitle("NEW ACCOUNT APPLICATION FORM");
        setResizable(false);
        setSize(520, 720);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setIconImage(new ImageIcon("src/icon.png").getImage());
        setLocationRelativeTo(null);
        setLayout(null);
        this.userDataList = userDataList;
        initComponents();
    }

    private void initComponents() {
        registeredCardNumbers = fetchRegisteredCardNumbers();

        personalDetailsPanel = new JPanel();
        cardDetailsPanel = new JPanel();
        personalDetailsPanel.setSize(520, 720);
        personalDetailsPanel.setLayout(null);

        banner = new JLabel(" APPLICATION FORM ",
SwingConstants.CENTER);
        banner.setFont(new Font("Verdana", Font.BOLD, 14));
        banner.setBounds(20, 20, 480, 30);

        page1 = new JLabel("Page 1: Personal Details",
SwingConstants.CENTER);
        page1.setBounds(20, 60, 480, 20);

        nameL = new JLabel("Name:");
        nameL.setBounds(20, 100, 120, 25);
        nameI = new JTextField();
        nameI.setBounds(150, 100, 320, 25);

        fatherNameL = new JLabel("Father's Name:");
        fatherNameL.setBounds(20, 140, 120, 25);
        fatherNameI = new JTextField();
        fatherNameI.setBounds(150, 140, 320, 25);

        dobL = new JLabel("Date of Birth (yyyy-mm-dd):");
        dobL.setBounds(20, 180, 180, 25);
        dobI = new JTextField();
        dobI.setBounds(200, 180, 270, 25);

        genderL = new JLabel("Gender:");
        genderL.setBounds(20, 220, 120, 25);
        maleRadioButton = new JRadioButton("Male");
        maleRadioButton.setBounds(150, 220, 80, 25);
        femaleRadioButton = new JRadioButton("Female");
        femaleRadioButton.setBounds(250, 220, 100, 25);
        otherRadioButton = new JRadioButton("Other");
        otherRadioButton.setBounds(360, 220, 80, 25);
```

```
ButtonGroup genderGroup = new ButtonGroup();
genderGroup.add(maleRadioButton);
genderGroup.add(femaleRadioButton);
genderGroup.add(otherRadioButton);

maritalL = new JLabel("Marital Status:");
maritalL.setBounds(20, 260, 120, 25);
singleRadioButton = new JRadioButton("Single");
singleRadioButton.setBounds(150, 260, 80, 25);
marriedRadioButton = new JRadioButton("Married");
marriedRadioButton.setBounds(250, 260, 100, 25);
noToSayRadioButton = new JRadioButton("Prefer not to say");
noToSayRadioButton.setBounds(360, 260, 150, 25);
ButtonGroup maritalGroup = new ButtonGroup();
maritalGroup.add(singleRadioButton);
maritalGroup.add(marriedRadioButton);
maritalGroup.add(noToSayRadioButton);

emailL = new JLabel("Email:");
emailL.setBounds(20, 300, 120, 25);
emailI = new JTextField();
emailI.setBounds(150, 300, 320, 25);

cityL = new JLabel("City:");
cityL.setBounds(20, 340, 120, 25);
cityI = new JTextField();
cityI.setBounds(150, 340, 320, 25);

pinCodeL = new JLabel("Pin Code:");
pinCodeL.setBounds(20, 380, 120, 25);
pinCodeI = new JTextField();
pinCodeI.setBounds(150, 380, 320, 25);

stateL = new JLabel("State:");
stateL.setBounds(20, 420, 120, 25);
stateI = new JTextField();
stateI.setBounds(150, 420, 320, 25);

addressL = new JLabel("Address:");
addressL.setBounds(20, 460, 120, 25);
addressI = new JTextField();
addressI.setBounds(150, 460, 320, 25);

cancel = new JButton("Cancel");
cancel.setBounds(260, 500, 100, 30);

next = new JButton("Next");
next.setBounds(370, 500, 100, 30);

personalDetailsPanel.add(banner);
personalDetailsPanel.add(page1);
personalDetailsPanel.add(nameL);
personalDetailsPanel.add(nameI);
personalDetailsPanel.add(fatherNameL);
```

```
        personalDetailsPanel.add(fatherNameI);
        personalDetailsPanel.add(dobL);
        personalDetailsPanel.add(dobI);
        personalDetailsPanel.add(genderL);
        personalDetailsPanel.add(maleRadioButton);
        personalDetailsPanel.add(femaleRadioButton);
        personalDetailsPanel.add(otherRadioButton);
        personalDetailsPanel.add(maritalL);
        personalDetailsPanel.add(singleRadioButton);
        personalDetailsPanel.add(marriedRadioButton);
        personalDetailsPanel.add(noToSayRadioButton);
        personalDetailsPanel.add(emailL);
        personalDetailsPanel.add(emailI);
        personalDetailsPanel.add(cityL);
        personalDetailsPanel.add(cityI);
        personalDetailsPanel.add(pinCodeL);
        personalDetailsPanel.add(pinCodeI);
        personalDetailsPanel.add(stateL);
        personalDetailsPanel.add(stateI);
        personalDetailsPanel.add(addressL);
        personalDetailsPanel.add(addressI);
        personalDetailsPanel.add(cancel);
        personalDetailsPanel.add(next);

        add(personalDetailsPanel);

        cancel.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int i = JOptionPane.showConfirmDialog(null, "Are You
Sure ? ", "Confirmation", JOptionPane.YES_NO_OPTION);
                if (i == JOptionPane.YES_OPTION) {
                    dispose();
                }
            }
        });

        next.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                boolean a = verify1(nameI.getText(),
fatherNameI.getText(), dobI.getText(), emailI.getText(),
cityI.getText(), stateI.getText(), pinCodeI.getText(),
addressI.getText(), v.getSelectedRadioButtonText(genderGroup),
v.getSelectedRadioButtonText(maritalGroup));
                if (a) {
                    personalDetailsPanel.setVisible(false);
                    add(cardDetailsPanel);
                    cardDetailsPanel.setVisible(true);
                }
            }
        });

        // Page 2
```

```
        cardDetailsPanel.setSize(520, 720);
        cardDetailsPanel.setLayout(null);
        page2 = new JLabel("Page 2: Card Details",
SwingConstants.CENTER);
        page2.setBounds(20, 60, 480, 20);
        cardDetailsPanel.add(page2);

        cardNumLabel = new JLabel("Choose Your Card No. (12-digit):");
        cardNumLabel.setBounds(20, 100, 220, 25);
        cardDetailsPanel.add(cardNumLabel);

        cardNumInput = new JTextField(12);
        cardNumInput.setBounds(250, 100, 220, 25);
        cardDetailsPanel.add(cardNumInput);

        pinNumLabel = new JLabel("Choose Your PIN No. (6-digit):");
        pinNumLabel.setBounds(20, 150, 200, 25);
        cardDetailsPanel.add(pinNumLabel);

        pinNumInput = new JPasswordField(6);
        pinNumInput.setBounds(250, 150, 220, 25);
        cardDetailsPanel.add(pinNumInput);

        startAmountL = new JLabel("Start Amount:");
        startAmountL.setBounds(20, 200, 120, 25);
        cardDetailsPanel.add(startAmountL);

        startAmountI = new JTextField();
        startAmountI.setBounds(250, 200, 220, 25);
        cardDetailsPanel.add(startAmountI);

        eCheckL = new JCheckBox("Stay Updated With Email...");
        eCheckL.setBounds(150, 240, 320, 25);
        cardDetailsPanel.add(eCheckL);

        cancel1 = new JButton("Cancel");
        cancel1.setBounds(150, 270, 100, 30);
        cardDetailsPanel.add(cancel1);

        back = new JButton("Back");
        back.setBounds(260, 270, 100, 30);
        cardDetailsPanel.add(back);

        submit = new JButton("Submit");
        submit.setBounds(370, 270, 100, 30);
        cardDetailsPanel.add(submit);

        cancel1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int i = JOptionPane.showConfirmDialog(null, "Are You
Sure ? ", "Confirmation", JOptionPane.YES_NO_OPTION);
                if (i == JOptionPane.YES_OPTION) {
                    dispose();
```

```
                }
            }
        });

        back.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                cardDetailsPanel.setVisible(false);
                personalDetailsPanel.setVisible(true);
            }
        });

        submit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if ((verify2(cardNumInput.getText(),
pinNumInput.getText(), startAmountI.getText(), eCheckL.isSelected())))
{
                    // Generate account number
                    accountNumber = FileHandler.generateAccNum();
                    if(accountNumber!=null){
                        System.out.println("Done");
                    }
                    JOptionPane.showMessageDialog(null, "Registration
successful...");
                    dispose();
                    FileHandler.writeUserDataToFile(nameI.getText(),
fatherNameI.getText(), dobI.getText(), emailI.getText(),
cityI.getText(), stateI.getText(), pinCodeI.getText(),
addressI.getText(), v.getSelectedRadioButtonText(genderGroup),
v.getSelectedRadioButtonText(maritalGroup), cardNumInput.getText(),
pinNumInput.getText(), startAmountI.getText(), eCheck,accountNumber);
                    registeredCardNumbers.add(cardNumInput.getText());

                    // Send email with the correct account number

                    String
msg=Emsg.registrationSuccess(nameI.getText(),accountNumber,cardNumInpu
t.getText(),pinNumInput.getText());
                    if (eCheckL.isSelected()) {
                        new EmailService().sendEmail(emailI.getText(),
"Successfully Registered", msg, null,true);
                    }
                    dispose(); // Close the current window

                }else {}
            }
        });

    }

    private Set<String> fetchRegisteredCardNumbers() {
        Set<String> cardNumbers = new HashSet<>();
        try (BufferedReader reader = new BufferedReader(new
```

```
FileReader("user_data.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.startsWith("Card Number:")) {
                    cardNumbers.add(line.substring(line.indexOf(":") +
2).trim());
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Error fetching
registered card numbers.");
        }
        return cardNumbers;
    }

    boolean verify1(String name, String fatherName, String dob, String
email, String city, String state, String pinCode, String address,
String gender, String maritalState) {
        if (name.isEmpty() || fatherName.isEmpty() || dob.isEmpty() ||
email.isEmpty() || city.isEmpty() || state.isEmpty() ||
pinCode.isEmpty() || address.isEmpty() || gender.isEmpty() ||
maritalState.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Make Sure You Are Not
Leaving Any Field Empty! ");
            return false;
        }
        if (!v.isPlainText(name) || !v.isPlainText(fatherName) ||
!v.isPlainText(city) || !v.isPlainText(state)) {
            JOptionPane.showMessageDialog(null, "Check Your Name,
Father's Name, City, State Fields. They Can Only Contain [A-Z], [a-z],
and White Spaces.");
            return false;
        }
        if (!v.isDate(dob)) {
            JOptionPane.showMessageDialog(null, "Make Sure You Entered
Date Of Birth In Correct Format (yyyy-mm-dd)!");
            return false;
        }
        if (!v.isEmail(email)) {
            JOptionPane.showMessageDialog(null, "Please Enter a Valid
Email!");
            return false;
        }
        if (!v.isPin(pinCode)) {
            JOptionPane.showMessageDialog(null, "Please Enter a Valid
PIN Code!");
            return false;
        }
        return true;
    }

    boolean verify2(String cardNum, String pinNum, String amount,
boolean eCheck) {
```

```
        if (!v.isCard(cardNum)) {
            JOptionPane.showMessageDialog(null, "Please Choose a Valid
Card Number!");
            return false;
        }

        if (userDataList.stream().anyMatch(userData ->
userData.getCardNumber().equals(cardNum))) {
            JOptionPane.showMessageDialog(null, "Card number already
registered. Please choose a different card number.");
            return false;
        }

        if (!v.isPin(pinNum)) {
            JOptionPane.showMessageDialog(null, "Please Choose a Valid
PIN Number!");
            return false;
        }

        if (v.isAmount(amount)) {
            if (!v.isAmountNeg(amount)) {
                JOptionPane.showMessageDialog(null, "Starting Amount
Should Not Be Negative...");
                return false;
            }
        } else {
            JOptionPane.showMessageDialog(null, "Please Enter a Valid
Starting Amount !");
            return false;
        }

        if (!(eCheck)) {
            int i = JOptionPane.showConfirmDialog(null, "Are You Sure
You Not Want To Email Services ? ", "Confirmation",
JOptionPane.YES_NO_OPTION);
            if (i == JOptionPane.YES_OPTION) {
                this.eCheck = "No";
                return true;
            } else {
                return false;
            }
        } else {
            this.eCheck = "Yes";
            return v.otpVerify(emailI.getText(), (Frame)
SignUp.this.getParent());
        }
    }

}
```

- EmailService.java

```
import java.io.File;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
import javax.swing.*;

/**
 * EmailService class represents a utility for sending emails using
SMTP protocol.
 */
class EmailService extends JDialog {

    // Username and password for the email service
    private String username;
    private String password;

    // Properties for configuring the email session
    private Properties properties;

    // Email session to be reused
    private Session emailSession;

    /**
     * Default constructor with default email credentials.
     */
    public EmailService() {
        this("72mrvirus@gmail.com", "goyp mxtb wavl tjhg");
    }

    /**
     * Constructor with custom email credentials.
     *
     * @param username The username for the email service.
     * @param password The password for the email service.
     */
    public EmailService(String username, String password) {
        this.username = username;
        this.password = password;
        configureProperties();
        this.emailSession = getSession();
    }

    /**
     * Configures properties for the email session.
     */
    private void configureProperties() {
        this.properties = new Properties();
        properties.put("mail.smtp.host", "smtp.gmail.com");
        properties.put("mail.smtp.port", "465");
        properties.put("mail.smtp.auth", "true");
        properties.put("mail.smtp.socketFactory.port", "465");
```

```
        properties.put("mail.smtp.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
    }

    /**
     * Private method to get the email session.
     *
     * @return The email session.
     */
    private Session getSession() {
        return Session.getInstance(properties, new
javax.mail.Authenticator() {
            protected PasswordAuthentication
getPasswordAuthentication() {
                return new PasswordAuthentication(username, password);
            }
        });
    }

    /**
     * Public method to send email with optional attachments and HTML
or plain text format.
     *
     * @param recipients  Comma-separated email addresses.
     * @param subject     Email subject.
     * @param body        Email body.
     * @param attachments List of file paths to attachments.
     * @param html        True if the email body is in HTML format,
false for plain text.
     */
    public void sendEmail(String recipients, String subject, String
body, List<String> attachments, boolean html) {
        SwingWorker<Void, Void> worker = new SwingWorker<Void, Void>()
{
            @Override
            protected Void doInBackground() throws Exception {
                try {
                    Message message = new MimeMessage(emailSession);
                    message.setFrom(new InternetAddress(username));
                    message.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(recipients));
                    message.setSubject(subject);

                    // Create the message part
                    MimeBodyPart messageBodyPart = new MimeBodyPart();
                    if (html) {
                        messageBodyPart.setContent(body, "text/html");
                    } else {
                        messageBodyPart.setText(body);
                    }

                    Multipart multipart = new MimeMultipart();
                    multipart.addBodyPart(messageBodyPart);
```

```
                    // Add attachments if any
                    if (attachments != null && !attachments.isEmpty())
{
                        for (String filePath : attachments) {
                            MimeBodyPart attachmentPart = new
MimeBodyPart();
                            DataSource source = new
FileDataSource(filePath);
                            attachmentPart.setDataHandler(new
DataHandler(source));
                            attachmentPart.setFileName(new
File(filePath).getName());
                            multipart.addBodyPart(attachmentPart);
                        }
                    }

                    message.setContent(multipart);

                    // Send the message
                    Transport.send(message);
                    System.out.println("Email sent successfully!");
                } catch (MessagingException e) {
                    e.printStackTrace();
                }
                return null;
            }
        };

        worker.execute();
    }
}
```

➢ Emsg.java

```
class Emsg {

    static String logAlert(String holderName) {
        String emailBody = "<!DOCTYPE html>"
                + "<html>"
                + "<head>"
                + "<style>"
                + "body { font-family: Arial, sans-serif; }"
                + ".container { width: 80%; margin: 0 auto; padding:
20px; border: 1px solid #dcdcdc; border-radius: 10px; background-
color: #f9f9f9; }"
                + ".header { font-size: 24px; font-weight: bold;
margin-bottom: 20px; }"
                + ".content { font-size: 16px; line-height: 1.6; }"
                + ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }"
                + "</style>"
                + "</head>"
```

```
                + "<body>"
                + "<div class='container'>"
                + "<div class='header'>Login Notification</div>"
                + "<div class='content'>"
                + "Dear " + holderName + ",<br><br>"
                + "Recently, you logged into Apna Bank. If this was
not you, please check your account immediately.<br><br>"
                + "If you have any questions or need further
assistance, please do not hesitate to contact our support
team.<br><br>"
                + "Best regards,<br>"
                + "Apna Bank"
                + "</div>"
                + "<div class='footer'>"
                + "This is an automated message, please do not reply."
                + "</div>"
                + "</div>"
                + "</body>"
                + "</html>";
        return emailBody;
    }

    static String pinChange(String holderName) {
        String emailBody = "<!DOCTYPE html>"
                + "<html>"
                + "<head>"
                + "<style>"
                + "body { font-family: Arial, sans-serif; background-
color: #f6f6f6; padding: 20px; }"
                + ".container { max-width: 600px; margin: 0 auto;
background-color: #ffffff; padding: 20px; border-radius: 10px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); }"
                + ".header { font-size: 24px; font-weight: bold;
margin-bottom: 20px; color: #333333; }"
                + ".content { font-size: 16px; line-height: 1.6;
color: #333333; }"
                + ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }"
                + ".footer a { color: #0066cc; text-decoration: none;
}"
                + "</style>"
                + "</head>"
                + "<body>"
                + "<div class='container'>"
                + "<div class='header'>Security Alert: PIN
Changed</div>"
                + "<div class='content'>"
                + "Dear " + holderName + ",<br><br>"
                + "We wanted to let you know that your PIN has been
successfully changed. If you did not request this change, please
contact our support team immediately to secure your account.<br><br>"
                + "For your security, please do not share your PIN
with anyone and ensure that you use a strong and unique PIN.<br><br>"
                + "If you have any questions or need further
```

```
assistance, please do not hesitate to contact our support
team.<br><br>"
                + "Best regards,<br>"
                + "Apna Bank"
                + "</div>"
                + "<div class='footer'>"
                + "This is an automated message, please do not reply.
If you need assistance, please contact our support team."
                + "</div>"
                + "</div>"
                + "</body>"
                + "</html>";
        return emailBody;
    }

    static String moneyDeposited(String holderName, String amount,
String accountNumber, String updatedBalance) {
        String emailBody = "<!DOCTYPE html>"
                + "<html>"
                + "<head>"
                + "<style>"
                + "body { font-family: Arial, sans-serif; background-
color: #f6f6f6; padding: 20px; }"
                + ".container { max-width: 600px; margin: 0 auto;
background-color: #ffffff; padding: 20px; border-radius: 10px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); }"
                + ".header { font-size: 24px; font-weight: bold;
margin-bottom: 20px; color: #333333; }"
                + ".content { font-size: 16px; line-height: 1.6;
color: #333333; }"
                + ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }"
                + ".footer a { color: #0066cc; text-decoration: none;
}"
                + "</style>"
                + "</head>"
                + "<body>"
                + "<div class='container'>"
                + "<div class='header'>Deposit Confirmation</div>"
                + "<div class='content'>"
                + "Dear " + holderName + ",<br><br>"
                + "We are pleased to inform you that ₹ " + amount + "
has been successfully deposited into your account No. " +
accountNumber + ".<br><br>"
                + "Your updated balance is ₹ " + updatedBalance +
".<br><br>"
                + "If you have any questions or need further
assistance, please do not hesitate to contact our support
team.<br><br>"
                + "Best regards,<br>"
                + "Apna Bank"
                + "</div>"
                + "<div class='footer'>"
                + "This is an automated message, please do not reply.
```

```
If you need assistance, please contact our support team."
                + "</div>"
                + "</div>"
                + "</body>"
                + "</html>";
        return emailBody;
    }
    static String moneyWithdrawn(String holderName, String amount,
String accountNumber, String updatedBalance) {
        String emailBody = "<!DOCTYPE html>"
                + "<html>"
                + "<head>"
                + "<style>"
                + "body { font-family: Arial, sans-serif; background-
color: #f6f6f6; padding: 20px; }"
                + ".container { max-width: 600px; margin: 0 auto;
background-color: #ffffff; padding: 20px; border-radius: 10px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); }"
                + ".header { font-size: 24px; font-weight: bold;
margin-bottom: 20px; color: #333333; }"
                + ".content { font-size: 16px; line-height: 1.6;
color: #333333; }"
                + ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }"
                + ".footer a { color: #0066cc; text-decoration: none;
}"
                + "</style>"
                + "</head>"
                + "<body>"
                + "<div class='container'>"
                + "<div class='header'>Withdrawal Confirmation</div>"
                + "<div class='content'>"
                + "Dear " + holderName + ",<br><br>"
                + "We are writing to inform you that ₹ " + amount + "
has been successfully withdrawn from your account No. " +
accountNumber + ".<br><br>"
                + "Your updated balance is ₹" + updatedBalance +
".<br><br>"
                + "If you have any questions or need further
assistance, please do not hesitate to contact our support
team.<br><br>"
                + "Best regards,<br>"
                + "Apna Bank"
                + "</div>"
                + "<div class='footer'>"
                + "This is an automated message, please do not reply.
If you need assistance, please contact our support team."
                + "</div>"
                + "</div>"
                + "</body>"
                + "</html>";
        return emailBody;
    }
    static String accountStatement(String holderName, String
```

```
accountNumber) {
        String emailBody = "<!DOCTYPE html>"
                + "<html>"
                + "<head>"
                + "<style>"
                + "body { font-family: Arial, sans-serif; background-
color: #f6f6f6; padding: 20px; }"
                + ".container { max-width: 600px; margin: 0 auto;
background-color: #ffffff; padding: 20px; border-radius: 10px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); }"
                + ".header { font-size: 24px; font-weight: bold;
margin-bottom: 20px; color: #333333; }"
                + ".content { font-size: 16px; line-height: 1.6;
color: #333333; }"
                + ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }"
                + ".footer a { color: #0066cc; text-decoration: none;
}"
                + "</style>"
                + "</head>"
                + "<body>"
                + "<div class='container'>"
                + "<div class='header'>Account Statement</div>"
                + "<div class='content'>"
                + "Dear " + holderName + ",<br><br>"
                + "Your account statement for " + accountNumber + " is
ready.<br><br>"
                + "Please find your account statement attached to this
email.<br><br>"
                + "If you have any questions or need further
assistance, please do not hesitate to contact our support
team.<br><br>"
                + "Best regards,<br>"
                + "Apna Bank"
                + "</div>"
                + "<div class='footer'>"
                + "This is an automated message, please do not reply.
If you need assistance, please contact our support team."
                + "</div>"
                + "</div>"
                + "</body>"
                + "</html>";
        return emailBody;
    }
    static String otpMessage(String otp) {
        String emailBody = "<!DOCTYPE html>"
                + "<html>"
                + "<head>"
                + "<style>"
                + "body { font-family: Arial, sans-serif; background-
color: #f6f6f6; padding: 20px; }"
                + ".container { max-width: 600px; margin: 0 auto;
background-color: #ffffff; padding: 20px; border-radius: 10px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); }"
```

```
                + ".header { font-size: 24px; font-weight: bold;
margin-bottom: 20px; color: #333333; }"
                + ".content { font-size: 16px; line-height: 1.6;
color: #333333; }"
                + ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }"
                + ".footer a { color: #0066cc; text-decoration: none;
}"
                + "</style>"
                + "</head>"
                + "<body>"
                + "<div class='container'>"
                + "<div class='header'>Verification</div>"
                + "<div class='content'>"
                + "Dear User,<br><br>"
                + "Your OTP for verification is: <strong>" + otp +
"</strong>.<br><br>"
                + "Please use this OTP to complete the verification
process.<br><br>"
                + "If you did not initiate this request, please
disregard this email.<br><br>"
                + "Best regards,<br>"
                + "Apna Bank"
                + "</div>"
                + "<div class='footer'>"
                + "This is an automated message, please do not reply.
If you need assistance, please contact our support team."
                + "</div>"
                + "</div>"
                + "</body>"
                + "</html>";
        return emailBody;
    }
    static String registrationSuccess(String name, String
accountNumber, String cardNumber, String pinNumber) {
        String emailBody = "<!DOCTYPE html>"
                + "<html>"
                + "<head>"
                + "<style>"
                + "body { font-family: Arial, sans-serif; background-
color: #f6f6f6; padding: 20px; }"
                + ".container { max-width: 600px; margin: 0 auto;
background-color: #ffffff; padding: 20px; border-radius: 10px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); }"
                + ".header { font-size: 24px; font-weight: bold;
margin-bottom: 20px; color: #333333; }"
                + ".content { font-size: 16px; line-height: 1.6;
color: #333333; }"
                + ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }"
                + ".footer a { color: #0066cc; text-decoration: none;
}"
                + "</style>"
                + "</head>"
```

```
                + "<body>"
                + "<div class='container'>"
                + "<div class='header'>Successfully Registered</div>"
                + "<div class='content'>"
                + "Dear " + name + ",<br><br>"
                + "Your Bank Account has been successfully
created.<br><br>"
                + "Here are your account details:<br>"
                + "Account Number: " + accountNumber + "<br>"
                + "Card Number: " + cardNumber + "<br>"
                + "PIN Number: " + pinNumber + "<br><br>"
                + "If you have any questions or need further
assistance, please do not hesitate to contact our support
team.<br><br>"
                + "Best regards,<br>"
                + "Apna Bank"
                + "</div>"
                + "<div class='footer'>"
                + "This is an automated message, please do not reply.
If you need assistance, please contact our support team."
                + "</div>"
                + "</div>"
                + "</body>"
                + "</html>";
        return emailBody;
    }
    static String moneyTransferSuccess(String recipientName,String
acc1,String acc2, double amountTransferred, double
newRecipientBalance) {
        String emailBody = "<!DOCTYPE html>" +
                "<html>" +
                "<head>" +
                "<style>" +
                "body { font-family: Arial, sans-serif; background-
color: #f6f6f6; padding: 20px; }" +
                ".container { max-width: 600px; margin: 0 auto;
background-color: #ffffff; padding: 20px; border-radius: 10px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); }" +
                ".header { font-size: 24px; font-weight: bold; margin-
bottom: 20px; color: #333333; }" +
                ".content { font-size: 16px; line-height: 1.6; color:
#333333; }" +
                ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }" +
                ".footer a { color: #0066cc; text-decoration: none; }"
+
                "</style>" +
                "</head>" +
                "<body>" +
                "<div class='container'>" +
                "<div class='header'>Money Transfer Successful</div>"
+
                "<div class='content'>" +
                "Dear " + recipientName + ",<br><br>" +
```

```
                "You have successfully  a transfer of ₹ " +
String.format("%.2f", amountTransferred) + ".<br>" +
                "Account No : " + acc1 + ".<br>" +
                "From : " + acc2 + ".<br>" +
                "Your new account balance is: ₹ " +
String.format("%.2f", newRecipientBalance) + ".<br><br>" +
                "Thank you for using our service.<br><br>" +
                "Best regards,<br>" +
                "Apna Bank Name" +
                "</div>" +
                "<div class='footer'>" +
                "This is an automated message, please do not reply. If
you need assistance, please contact our support team." +
                "</div>" +
                "</div>" +
                "</body>" +
                "</html>";
        return emailBody;
    }

    static String moneyReciverSuccess(String recipientName,String
acc1,String acc2, double amountTransferred, double
newRecipientBalance) {
        String emailBody = "<!DOCTYPE html>" +
                "<html>" +
                "<head>" +
                "<style>" +
                "body { font-family: Arial, sans-serif; background-
color: #f6f6f6; padding: 20px; }" +
                ".container { max-width: 600px; margin: 0 auto;
background-color: #ffffff; padding: 20px; border-radius: 10px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); }" +
                ".header { font-size: 24px; font-weight: bold; margin-
bottom: 20px; color: #333333; }" +
                ".content { font-size: 16px; line-height: 1.6; color:
#333333; }" +
                ".footer { font-size: 14px; color: #555555; margin-
top: 20px; }" +
                ".footer a { color: #0066cc; text-decoration: none; }"
+
                "</style>" +
                "</head>" +
                "<body>" +
                "<div class='container'>" +
                "<div class='header'>Money Received Successful</div>"
+
                "<div class='content'>" +
                "Dear " + recipientName + ",<br><br>" +
                "You have successfully received of ₹ " +
String.format("%.2f", amountTransferred) + ".<br>" +
                "Account No : " + acc1 + ".<br>" +
                "From : " + acc2 + ".<br>" +
                "Your new account balance is: ₹ " +
String.format("%.2f", newRecipientBalance) + ".<br><br>" +
```

```
                "Thank you for using our service.<br><br>" +
                "Best regards,<br>" +
                "Apna Bank Name" +
                "</div>" +
                "<div class='footer'>" +
                "This is an automated message, please do not reply. If
you need assistance, please contact our support team." +
                "</div>" +
                "</div>" +
                "</body>" +
                "</html>";
        return emailBody;
    }
}
```

➢ FileHandler.java

```
import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;

/**
 * FileHandler class manages reading from and writing to files.
 */
public class FileHandler {
    // Set to store registered account numbers for ensuring uniqueness
    private static Set<String> registeredAccountNumbers = new
HashSet<>();

    /**
     * Method to write user data to a file.
     *
     * @param name          Name of the user.
     * @param fatherName    Father's name of the user.
     * @param dob           Date of birth of the user.
     * @param email         Email of the user.
     * @param city          City of the user.
     * @param state         State of the user.
     * @param pinCode       PIN code of the user.
     * @param address       Address of the user.
     * @param gender        Gender of the user.
     * @param maritalState  Marital status of the user.
     * @param cardNum       Card number of the user.
     * @param pinNum        PIN number of the user.
     * @param amount        Starting amount in the account.
     * @param eCheck        Email service check.
     * @param accNum        Account number of the user.
     */
    public static void writeUserDataToFile(String name, String
fatherName, String dob, String email, String city,
```

```
                                                String state, String
pinCode, String address, String gender, String maritalState,
                                                String cardNum, String
pinNum, String amount, String eCheck, String accNum) {
        String fileName = "user_data.txt";
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyyMMddHHmmss");
        String timestamp = LocalDateTime.now().format(formatter);
        String entry = "Entry Time: " + timestamp + "\n" +
                "Account Number: " + accNum + "\n" +
                "Name: " + name + "\n" +
                "Father's Name: " + fatherName + "\n" +
                "Date of Birth: " + dob + "\n" +
                "Email: " + email + "\n" +
                "City: " + city + "\n" +
                "State: " + state + "\n" +
                "Pin Code: " + pinCode + "\n" +
                "Address: " + address + "\n" +
                "Gender: " + gender + "\n" +
                "Marital Status: " + maritalState + "\n" +
                "Card Number: " + cardNum + "\n" +
                "PIN: " + pinNum + "\n" +
                "Start Amount: " + amount + "\n" +
                "Email Service: " + eCheck + "\n\n";
        try {
            PrintWriter writer = new PrintWriter(new
FileWriter(fileName, true)); // Append mode
            writer.println(entry);
            writer.close();
            System.out.println("User data saved successfully.");
        } catch (IOException e) {
            System.out.println("Error writing user data to file: " +
e.getMessage());
        }
    }
    /**
     * Records the transaction in the file.
     *
     * @param amount          The amount of money involved in the
transaction.
     * @param transactionType The type of transaction (e.g., deposit,
withdrawal).
     * @param accNum          The account number associated with the
card.
     */
    static public void recordTransaction(String accNum, double amount,
String transactionType) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter("records.txt", true))) {
            writer.write("Account Number: " + accNum + "\n");
            writer.write(transactionType + " Time: " +
java.time.LocalDateTime.now() + "\n");
            writer.write(transactionType + " Amount: " + amount + "
₹\n\n");
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        /**
         * Method to generate a unique account number.
         *
         * @return A unique account number.
         */
        static String generateAccNum() {
            Random random = new Random();
            StringBuilder accNumBuilder = new StringBuilder();
            for (int i = 0; i < 16; i++) {
                accNumBuilder.append(random.nextInt(10)); // Append a
random digit (0-9)
            }
            String accNum = accNumBuilder.toString();
            // Ensure uniqueness
            while (registeredAccountNumbers.contains(accNum)) {
                accNumBuilder = new StringBuilder();
                for (int i = 0; i < 16; i++) {
                    accNumBuilder.append(random.nextInt(10));
                }
                accNum = accNumBuilder.toString();
            }
            registeredAccountNumbers.add(accNum);
            return accNum;
        }
}
```

- OtpManner.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

/**
 * OtpManner class represents a dialog for OTP verification.
 */
class OtpManner extends JDialog {
    private JLabel l1;
    private JPasswordField passI;
    private JButton check, resend;
    private int otp;
    private boolean e = false; // Flag to indicate if verification is
successful
    private EmailService emailService; // Email service for sending
OTP
    private Validate v;

    /**
```

```
     * Constructor to initialize the OTP verification dialog.
     *
     * @param owner    The parent frame.
     * @param receiver The email address to which the OTP will be
sent.
     */
    public OtpManner(Frame owner, String receiver) {
        super(owner, true);
        setTitle("Verification");
        setSize(520, 160);
        setResizable(false);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setIconImage(new ImageIcon("src/icon.png").getImage());
        setLocationRelativeTo(null);
        setLayout(null);
        v = new Validate();

        // Initialize components
        l1 = new JLabel("Enter Your OTP :");
        passI = new JPasswordField();
        check = new JButton("Check");
        resend = new JButton("Resend");

        // Set positions of components
        l1.setBounds(20, 20, 480, 30);
        passI.setBounds(20, 60, 200, 30);
        check.setBounds(240, 60, 100, 30);
        resend.setBounds(350, 60, 100, 30);

        // Add components to dialog
        add(l1);
        add(passI);
        add(check);
        add(resend);

        // Initialize email service
        emailService = new EmailService();

        // Action listener for check button
        check.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent evt) {
                if (isValidOTP(passI.getText())) {
                    int pass = Integer.parseInt(passI.getText());
                    if (otp == pass) {
                        e = true;
                        dispose(); // Close the dialog if verification
is successful
                    } else {
                        JOptionPane.showMessageDialog(null, "Incorrect
OTP! OR Need 6-Digit OTP"+"\n(TIP:Use Resend"); // Display error
message for incorrect OTP
                    }
                } else {
```

```
                    JOptionPane.showMessageDialog(null, "Invalid OTP
Format!"); // Display error message for invalid OTP format
                }
            }
        });

        // Action listener for resend button
        resend.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                generateAndSendOTP(receiver);
                JOptionPane.showMessageDialog(null, "OTP resend
successfully...");// Generate and send a new OTP
            }
        });

        generateAndSendOTP(receiver); // Automatically send the first
OTP when the dialog is created
    }

    /**
     * Method to check if the entered OTP is in the correct format and
matches the generated OTP.
     *
     * @param otpInput The OTP entered by the user.
     * @return true if the OTP is valid, false otherwise.
     */
    private boolean isValidOTP(String otpInput) {
        return v.isPin(otpInput);
    }

    /**
     * Method to generate and send OTP.
     *
     * @param receiver The email address to which the OTP will be
sent.
     */

    private void generateAndSendOTP(String receiver) {
        otp = generateOTP(); // Generate a new OTP
        String msg=Emsg.otpMessage(Integer.toString(otp));
        emailService.sendEmail(receiver, "Verification", msg,
null,true); // Send OTP via email
    }

    /**
     * Method to check if verification is successful.
     *
     * @return true if verification is successful, false otherwise.
     */
    public boolean isVerified() {
        return e;
    }
```

```
    /**
     * Method to generate a random OTP.
     *
     * @return The generated OTP.
     */
    private int generateOTP() {
        Random random = new Random();
        int generatedOTP = 0;

        for (int i = 0; i < 6; i++) {
            generatedOTP = generatedOTP * 10 + random.nextInt(10); // Generates a new digit and appends it to otp
        }
        return generatedOTP;
    }
}
```

➢ UserData.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * UserData class represents user data and provides methods to fetch and update user information.
 */
class UserData {
    private String name;
    private String fatherName;
    private String dob;
    private String email;
    private String city;
    private String state;
    private String pinCode;
    private String address;
    private String gender;
    private String maritalStatus;
    private String cardNumber;
    private String pin;
    private String startAmount;
    private String eCheck;
    private String accNum;

    // Constructor
    public UserData(String name, String fatherName, String dob, String email, String city, String state, String pinCode, String address, String gender, String maritalStatus, String cardNumber, String pin, String startAmount, String eCheck, String accNum) {
        this.name = name;
        this.fatherName = fatherName;
```

```
        this.dob = dob;
        this.email = email;
        this.city = city;
        this.state = state;
        this.pinCode = pinCode;
        this.address = address;
        this.gender = gender;
        this.maritalStatus = maritalStatus;
        this.cardNumber = cardNumber;
        this.pin = pin;
        this.startAmount = startAmount;
        this.eCheck = eCheck;
        this.accNum = accNum;
    }

    // Getters and setters
    // Getters for UserData fields
    public String getName() {
        return name;
    }

    public String getFatherName() {
        return fatherName;
    }

    public String getDob() {
        return dob;
    }

    public String getEmail() {
        return email;
    }

    public String getCity() {
        return city;
    }

    public String getState() {
        return state;
    }

    public String getPinCode() {
        return pinCode;
    }

    public String getAddress() {
        return address;
    }

    public String getGender() {
        return gender;
    }

    public String getMaritalStatus() {
```

```
        return maritalStatus;
    }

    public String getCardNumber() {
        return cardNumber;
    }

    public String getPin() {
        return pin;
    }

    public String getStartAmount() {
        return startAmount;
    }

    public String getECheck() {
        return eCheck;
    }

    public String getAccNum() {
        return accNum;
    }

    // Setter for User's name
    public void setName(String name) {
        this.name = name;
    }

    // Setter for User's father's name
    public void setFatherName(String fatherName) {
        this.fatherName = fatherName;
    }

    // Setter for User's date of birth
    public void setDob(String dob) {
        this.dob = dob;
    }

    // Setter for User's email
    public void setEmail(String email) {
        this.email = email;
    }

    // Setter for User's city
    public void setCity(String city) {
        this.city = city;
    }

    // Setter for User's state
    public void setState(String state) {
        this.state = state;
    }

    // Setter for User's pin code
```

```
    public void setPinCode(String pinCode) {
        this.pinCode = pinCode;
    }

    // Setter for User's address
    public void setAddress(String address) {
        this.address = address;
    }

    // Setter for User's gender
    public void setGender(String gender) {
        this.gender = gender;
    }

    // Setter for User's marital status
    public void setMaritalStatus(String maritalStatus) {
        this.maritalStatus = maritalStatus;
    }

    // Setter for User's card number
    public void setCardNumber(String cardNumber) {
        this.cardNumber = cardNumber;
    }

    // Setter for User's PIN
    public void setPin(String pin) {
        this.pin = pin;
    }

    // Setter for User's starting balance
    public void setStartAmount(String startAmount) {
        this.startAmount = startAmount;
    }

    // Setter for User's email notification preference
    public void setECheck(String eCheck) {
        this.eCheck = eCheck;
    }

    // Setter for User's account number
    public void setAccNum(String accNum) {
        this.accNum = accNum;
    }

    // Fetch user data from file
    public static List<UserData> fetchUserData() {
        List<UserData> userList = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader("user_data.txt"))) {
            String line;
            String accNum = null, name = null, fatherName = null, dob
= null, email = null, city = null, state = null, pinCode = null,
address = null, gender = null, maritalStatus = null, cardNumber =
null, pin = null, startAmount = null, eCheck = null;
```

```
            while ((line = reader.readLine()) != null) {
                if (line.startsWith("Account Number:")) {
                    accNum = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Name:")) {
                    name = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Father's Name:")) {
                    fatherName = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Date of Birth:")) {
                    dob = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Email:")) {
                    email = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("City:")) {
                    city = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("State:")) {
                    state = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Pin Code:")) {
                    pinCode = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Address:")) {
                    address = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Gender:")) {
                    gender = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Marital Status:")) {
                    maritalStatus = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Card Number:")) {
                    cardNumber = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("PIN:")) {
                    pin = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Start Amount:")) {
                    startAmount = line.substring(line.indexOf(":") +
2).trim();
                } else if (line.startsWith("Email Service:")) {
                    eCheck = line.substring(line.indexOf(":") +
2).trim();
                    userList.add(new UserData(name, fatherName, dob,
email, city, state, pinCode, address, gender, maritalStatus,
cardNumber, pin, startAmount, eCheck, accNum));
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
        return userList;
    }

    // Get specific data field for a user
    public static String getData(String cardNumber, int i,
List<UserData> userDataList) {
        for (UserData u : userDataList) {
            if (u.getCardNumber().equals(cardNumber)) {
                switch (i) {
                    case 1:
                        return u.getCardNumber(); // Card Number
                    case 2:
                        return u.getPin(); // PIN
                    case 3:
                        return u.getEmail(); // Email
                    case 4:
                        return u.getName(); // Name
                    case 5:
                        return u.getStartAmount(); // Start Amount
                    case 6:
                        return u.getFatherName(); // Father's Name
                    case 7:
                        return u.getDob(); // Date of Birth
                    case 8:
                        return u.getGender(); // Gender
                    case 9:
                        return u.getMaritalStatus(); // Marital Status
                    case 10:
                        return u.getCity(); // City
                    case 11:
                        return u.getState(); // State
                    case 12:
                        return u.getPinCode(); // Pin Code
                    case 13:
                        return u.getAddress(); // Address
                    case 14:
                        return u.getECheck(); // Email Check
                    case 15:
                        return u.getAccNum(); // Account Number
                }
            }
        }
        return null;
    }
    public static String getData(String accNumber, List<UserData>
userDataList) {
        for (UserData u : userDataList) {
            if (u.getAccNum().equals(accNumber)) {
                    return u.getCardNumber(); // Account Number
                }
            }

        return null;
    }
```

```
    // Set specific data field for a user
    public static void setData(String cardNum, int index, String
value, List<UserData> userDataList) {
        for (UserData userData : userDataList) {
            if (userData.getCardNumber().equals(cardNum)) {
                switch (index) {
                    case 2:
                        userData.setPin(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing PIN
                        break;
                    case 5:
                        userData.setStartAmount(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing start amount
                        break;
                    case 6:
                        userData.setName(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing name
                        break;
                    case 7:
                        userData.setFatherName(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing father's name
                        break;
                    case 8:
                        userData.setDob(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing date of birth
                        break;
                    case 9:
                        userData.setEmail(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing email
                        break;
                    case 10:
                        userData.setCity(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing city
                        break;
                    case 11:
                        userData.setState(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing state
                        break;
                    case 12:
                        userData.setPinCode(value);
                        updateUserDataFile(userDataList); // Update
user_data.txt after changing pin code
                        break;
                    case 13:
                        userData.setAddress(value);
                        updateUserDataFile(userDataList); // Update
```

```
user_data.txt after changing address
                            break;
                        case 14:
                            userData.setECheck(value);
                            updateUserDataFile(userDataList); // Update
user_data.txt after changing email check
                            break;
                        case 15:
                            userData.setAccNum(value);
                            updateUserDataFile(userDataList); // Update
user_data.txt after changing account number
                            break;
                    }
                    break;
                }
            }
        }

        // Update user_data.txt file
        private static void updateUserDataFile(List<UserData>
userDataList) {
            try (FileWriter writer = new FileWriter("user_data.txt")) {
                for (UserData userData : userDataList) {
                    writer.write("Account Number: " + userData.getAccNum()
+ "\n");
                    writer.write("Name: " + userData.getName() + "\n");
                    writer.write("Father's Name: " +
userData.getFatherName() + "\n");
                    writer.write("Date of Birth: " + userData.getDob() +
"\n");
                    writer.write("Email: " + userData.getEmail() + "\n");
                    writer.write("City: " + userData.getCity() + "\n");
                    writer.write("State: " + userData.getState() + "\n");
                    writer.write("Pin Code: " + userData.getPinCode() +
"\n");
                    writer.write("Address: " + userData.getAddress() +
"\n");
                    writer.write("Gender: " + userData.getGender() +
"\n");
                    writer.write("Marital Status: " +
userData.getMaritalStatus() + "\n");
                    writer.write("Card Number: " +
userData.getCardNumber() + "\n");
                    writer.write("PIN: " + userData.getPin() + "\n");
                    writer.write("Start Amount: " +
userData.getStartAmount() + "\n");
                    writer.write("Email Service: " + userData.getECheck()
+ "\n\n");
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        public static boolean exitsAccNo(String accNo) {
```

```
        try (BufferedReader reader = new BufferedReader(new
FileReader("user_data.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.startsWith("Account Number:")) {
                    String existingAccNo =
line.substring(line.indexOf(":") + 2).trim();
                    if (existingAccNo.equals(accNo)) {
                        return true;
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return false;
    }

}
```

➢ Validate.java

```
import javax.swing.*;
import java.awt.*;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Enumeration;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Validate class provides various validation methods for different
purposes.
 */
class Validate {

    /**
     * Method to validate PIN code.
     *
     * @param pinCode The PIN code to validate.
     * @return true if the PIN is valid, false otherwise.
     */
    boolean isPin(String pinCode) {
        if (pinCode == null || pinCode.isEmpty()) {
            return false;
        }
        if (!pinCode.matches("\\d+")) {
            return false;
        }
        return pinCode.length() == 6;
    }
```

```
    /**
     * Method to get the text of the selected radio button in a
ButtonGroup.
     *
     * @param group The ButtonGroup containing the radio buttons.
     * @return The text of the selected radio button.
     */
    String getSelectedRadioButtonText(ButtonGroup group) {
        for (Enumeration<AbstractButton> buttons =
group.getElements(); buttons.hasMoreElements(); ) {
            AbstractButton button = buttons.nextElement();
            if (button.isSelected()) {
                return button.getText();
            }
        }
        return null;
    }

    /**
     * Method to check if a string contains only plain text.
     *
     * @param str The string to check.
     * @return true if the string contains only plain text, false
otherwise.
     */
    boolean isPlainText(String str) {
        for (char c : str.toCharArray()) {
            if (!(Character.isLetter(c) || Character.isWhitespace(c)))
{
                return false;
            }
        }
        return true;
    }

    /**
     * Method to check if a string is in date format (yyyy-MM-dd).
     *
     * @param str The string to check.
     * @return true if the string is a valid date, false otherwise.
     */
    boolean isDate(String str) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        sdf.setLenient(false);

        try {
            sdf.parse(str);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
```

```
    /**
     * Method to validate an email address.
     *
     * @param email The email address to validate.
     * @return true if the email address is valid, false otherwise.
     */
    boolean isEmail(String email) {
        String regex = "^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-
]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }

    /**
     * Method to validate a 12-digit card number.
     *
     * @param card The card number to validate.
     * @return true if the card number is valid, false otherwise.
     */
    boolean isCard(String card) {
        if (card == null || card.isEmpty()) {
            return false;
        }
        if (!card.matches("\\d+")) {
            return false;
        }
        return card.length() == 12;
    }

    /**
     * Method to validate a 16-digit account number.
     *
     * @param card The account number to validate.
     * @return true if the account number is valid, false otherwise.
     */
    boolean isAccountNumber(String card) {
        if (card == null || card.isEmpty()) {
            return false;
        }
        if (!card.matches("\\d+")) {
            return false;
        }
        return card.length() == 16;
    }

    /**
     * Method to validate an amount (numeric string).
     *
     * @param amount The amount to validate.
     * @return true if the amount is valid, false otherwise.
     */
    boolean isAmount(String amount) {
        if (amount == null || amount.isEmpty()) {
```

```
            return false;
        }
        return amount.matches("\\d+");
    }

    /**
     * Method to validate if the amount is non-negative.
     *
     * @param amount The amount to validate.
     * @return true if the amount is non-negative, false otherwise.
     */
    boolean isAmountNeg(String amount) {
        try {
            double d = Double.parseDouble(amount);
            return d >= 0;
        } catch (NumberFormatException e) {
            return false;
        }
    }

    /**
     * Method to authenticate a user using card number and PIN.
     *
     * @param cardNumber  The card number to authenticate.
     * @param pin         The PIN to authenticate.
     * @param userDataList The list of user data to search for
authentication.
     * @return true if the user is authenticated, false otherwise.
     */
    static boolean authenticateUser(String cardNumber, String pin,
List<UserData> userDataList) {
        for (UserData userData : userDataList) {
            if (userData.getCardNumber().equals(cardNumber) &&
userData.getPin().equals(pin)) {
                return true; // User authenticated
            }
        }
        return false; // User not found or authentication failed
    }

    /**
     * Method to verify OTP through email.
     *
     * @param email The email address to send OTP.
     * @param owner The parent frame.
     * @return true if OTP verification is successful, false
otherwise.
     */
    boolean otpVerify(String email, Frame owner) {
        if (!isInternetAvailable()) {
            JOptionPane.showMessageDialog(null, "No internet
connection available.");
            return false;
        }
```

```
            JOptionPane.showMessageDialog(null, "OTP Sent to " + email);
            OtpManner otpManner = new OtpManner(owner, email);
            otpManner.setVisible(true);
            return otpManner.isVerified();
        }

        /**
         * Method to check if internet is available.
         *
         * @return true if internet is available, false otherwise.
         */
        static boolean isInternetAvailable() {
            try {
                Socket socket = new Socket();
                socket.connect(new InetSocketAddress("www.google.com",
80), 2000);
                socket.close();
                return true;
            } catch (IOException e) {
                return false;
            }
        }

}
```

➢ ChangePIN.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

/**
 * ChangePIN class represents a dialog for changing PIN.
 */
public class ChangePIN extends JDialog {

    private JPasswordField currentPinField, newPinField;
    private JButton changeButton, cancelButton;
    private List<UserData> userDataList;
    private String cardNum;
    private Validate v;

    /**
     * Constructor to initialize the Change PIN dialog.
     *
     * @param owner       The owner frame of the dialog.
     * @param cardNum     The card number of the user.
     * @param userDataList The list of user data.
     */
    public ChangePIN(Frame owner, String cardNum, List<UserData>
userDataList) {
        super(owner, "Change PIN", true);
```

```
        setIconImage(new ImageIcon("src/icon.png").getImage());
        this.cardNum = cardNum;
        this.userDataList = userDataList;
        v = new Validate();

        setLayout(new GridLayout(4, 2, 10, 10));
        currentPinField = new JPasswordField();
        newPinField = new JPasswordField();
        changeButton = new JButton("Change");
        cancelButton = new JButton("Cancel");

        add(new JLabel("Enter Your Current PIN:"));
        add(currentPinField);
        add(new JLabel("Enter Your New PIN:"));
        add(newPinField);
        add(changeButton);
        add(cancelButton);

        // Action listener for the change button
        changeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String currentPin = new
String(currentPinField.getPassword());
                String storedPin = UserData.getData(cardNum, 2,
userDataList);

                if (currentPin.equals(storedPin)) { // Check if
current PIN matches the stored PIN
                    String newPin = new
String(newPinField.getPassword());

                    if (v.isPin(newPin)) { // Check if the new PIN is
valid
                        UserData.setData(cardNum, 2, newPin,
userDataList); // Update PIN in the userDataList
                        updatePinInFile(cardNum, newPin); // Update
PIN in the user_data.txt file
                        JOptionPane.showMessageDialog(null, "PIN
changed successfully.");

                        // Send email notification if enabled
                        if ("Yes".equals(UserData.getData(cardNum, 14,
userDataList))) {
                            String
msg=Emsg.pinChange(UserData.getData(cardNum, 4, userDataList));
                            new
EmailService().sendEmail(UserData.getData(cardNum, 3, userDataList),
"Security Alert: PIN Changed",
                                    msg, null,true);
                        }
                    } else {
                        JOptionPane.showMessageDialog(null, "Please
enter a valid 6-digit PIN.");
```

```
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Incorrect
current PIN. Please try again.");
                }
            }
        });

        // Action listener for the cancel button
        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the dialog
            }
        });

        pack();
        setLocationRelativeTo(null);
    }

    /**
     * Method to update PIN in the userDataList.
     *
     * @param cardNum The card number associated with the PIN.
     * @param newPin  The new PIN to be updated.
     */
    private void updatePinInFile(String cardNum, String newPin) {
        for (int i = 0; i < userDataList.size(); i++) {
            if (userDataList.get(i).getCardNumber().equals(cardNum)) {
                userDataList.get(i).setPin(newPin); // Update the pin
in the userDataList
                break;
            }
        }
    }
}
```

- Deposit.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

/**
 * The Deposit class represents the dialog window for depositing money
into the account.
 */
public class Deposit extends JDialog {
```

```
    // Components
    private JTextField textField; // Input field for amount to deposit
    private JButton depositButton, cancelButton; // Deposit and cancel
buttons

    // Data
    private List<UserData> userDataList; // List of user data
    private String cardNum; // Card number of the user
    private Validate v; // Validator
    private String accNum; // Account number associated with the card

    /**
     * Constructor for Deposit class.
     *
     * @param owner       The owner frame of the dialog.
     * @param cardNum     The card number of the user.
     * @param userDataList The list of user data.
     */
    public Deposit(Frame owner, String cardNum, List<UserData>
userDataList) {
        super(owner, "Deposit", true);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setIconImage(new ImageIcon("src/icon.png").getImage());
        this.cardNum = cardNum;
        this.userDataList = userDataList;
        v = new Validate();

        // Layout setup
        setLayout(new GridLayout(3, 2, 10, 10));
        textField = new JTextField();
        depositButton = new JButton("Deposit");
        cancelButton = new JButton("Cancel");

        // Adding components to the dialog
        add(new JLabel("Enter Amount to Deposit:"));
        add(textField);
        add(depositButton);
        add(cancelButton);

        // Action listeners
        depositButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                accNum = UserData.getData(cardNum, 15, userDataList);
// Initialize accNum here
                if (v.isAmount(textField.getText())) {
                    double oldBalance =
Double.parseDouble(UserData.getData(cardNum, 5, userDataList));
                    double depositAmount =
Double.parseDouble(textField.getText());
                    double newBalance = oldBalance + depositAmount;
                    UserData.setData(cardNum, 5,
Double.toString(newBalance), userDataList); // Update balance
```

```
                    // Record the deposit in the file
                    FileHandler.recordTransaction(accNum,
depositAmount, "Deposit");

                    JOptionPane.showMessageDialog(null,
textField.getText() + " ₹ deposited successfully into your account.");

                    // Send email notification if enabled
                    if ("Yes".equals(UserData.getData(cardNum, 14,
userDataList))) {
                        String
msg=Emsg.moneyDeposited(UserData.getData(cardNum,4,userDataList),Doubl
e.toString(depositAmount),UserData.getData(cardNum,15,userDataList),Do
uble.toString(newBalance));
                        new
EmailService().sendEmail(UserData.getData(cardNum, 3, userDataList),
"Deposit Confirmation", msg, null,true);
                    }

                    dispose(); // Dispose dialog after depositing
                } else {
                    JOptionPane.showMessageDialog(null, "Please enter
a valid amount.");
                }
            }
        });

        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Dispose dialog directly when cancel
button is clicked
            }
        });

        pack();
        setLocationRelativeTo(null);
    }

}
```

- FastWithdrawal.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

/**
 * FastWithdrawal class represents a dialog for fast withdrawal
functionality.
 */
```

```
public class FastWithdrawal extends JDialog {

    // Buttons for withdrawal
    private JButton button100, button200, button500, button1000,
button2000, button5000, button10000, cancelButton;

    // User data list and card number
    private List<UserData> userDataList;
    private String cardNum;

    // Validator
    private Validate v;

    /**
     * Constructor to initialize the Fast Withdrawal dialog.
     *
     * @param owner       The owner frame of the dialog.
     * @param cardNum     The card number of the user.
     * @param userDataList The list of user data.
     */
    public FastWithdrawal(Frame owner, String cardNum, List<UserData>
userDataList) {
        super(owner, "Fast Withdrawal", true);
        setResizable(false);
        setSize(520, 400); // Adjusted size to accommodate new layout
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());
        setIconImage(new ImageIcon("src/icon.png").getImage());
        this.cardNum = cardNum;
        this.userDataList = userDataList;
        v = new Validate(); // Instantiating Validator
        initComponents(); // Initializing components
    }

    // Initialize components
    private void initComponents() {
        // Buttons panel
        JPanel buttonsPanel = new JPanel(new GridLayout(3, 3, 10,
10));
        buttonsPanel.setBorder(BorderFactory.createEmptyBorder(10, 10,
10, 10));

        // Initializing buttons
        button100 = new JButton("₹100");
        button200 = new JButton("₹200");
        button500 = new JButton("₹500");
        button1000 = new JButton("₹1000");
        button2000 = new JButton("₹2000");
        button5000 = new JButton("₹5000");
        button10000 = new JButton("₹10000");
        cancelButton = new JButton("Cancel");

        // Adding buttons to the panel
```

```
        buttonsPanel.add(button100);
        buttonsPanel.add(button200);
        buttonsPanel.add(button500);
        buttonsPanel.add(button1000);
        buttonsPanel.add(button2000);
        buttonsPanel.add(button5000);
        buttonsPanel.add(button10000);
        buttonsPanel.add(cancelButton);

        // Add buttons panel to the dialog
        add(buttonsPanel, BorderLayout.CENTER);

        // Action listeners for withdrawal buttons
        button100.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                processWithdrawal(100); // Process withdrawal of ₹100
            }
        });

        button200.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                processWithdrawal(200); // Process withdrawal of ₹200
            }
        });

        button500.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                processWithdrawal(500); // Process withdrawal of ₹500
            }
        });

        button1000.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                processWithdrawal(1000); // Process withdrawal of
₹1000
            }
        });

        button2000.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                processWithdrawal(2000); // Process withdrawal of
₹2000
            }
        });

        button5000.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                processWithdrawal(5000); // Process withdrawal of
```

```
₹5000
            }
        });

        button10000.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                processWithdrawal(10000); // Process withdrawal of
₹10000
            }
        });

        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close dialog when cancel button is
clicked
            }
        });
    }

    // Method to process withdrawal
    private void processWithdrawal(double withdrawalAmount) {
        String accNum = "";
        for (UserData userData : userDataList) {
            if (userData.getCardNumber().equals(cardNum)) {
                accNum = userData.getAccNum(); // Get account number
associated with the card
                break;
            }
        }

        double oldBalance =
Double.parseDouble(UserData.getData(cardNum, 5, userDataList));
        double newBalance = oldBalance - withdrawalAmount;

        // Check if withdrawal amount is valid and balance is
sufficient
        if (withdrawalAmount >= 100 && withdrawalAmount <= 10000 &&
newBalance >= 1000) {
            UserData.setData(cardNum, 5, Double.toString(newBalance),
userDataList); // Update balance

            // Record the withdrawal in the file
            FileHandler.recordTransaction(accNum, withdrawalAmount,
"Fast Withdrawal");

            JOptionPane.showMessageDialog(null, "Please collect your
money...");

            // Send email notification if enabled
            if ("Yes".equals(UserData.getData(cardNum, 14,
userDataList))) {
                String
```

```
msg=Emsg.moneyWithdrawn(UserData.getData(cardNum, 4,
userDataList),Double.toString(withdrawalAmount),UserData.getData(cardN
um, 15, userDataList),Double.toString(newBalance));
                new EmailService().sendEmail(UserData.getData(cardNum,
3, userDataList), "Withdrawal Confirmation", msg, null,true);
            }
        } else {
            JOptionPane.showMessageDialog(null, "Withdrawal amount
must be between 100₹ and 10,000₹, and your balance should remain above
1000₹.");
        }
    }
}
```

- TMoney.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

class TMoney extends JDialog {
    private JTextField accNo, moneyT;
    private JButton transButton, cancelButton;
    private List<UserData> userDataList;
    private String cardNum;
    private Validate v;

    /**
     * Constructor to initialize the Change PIN dialog.
     *
     * @param owner        The owner frame of the dialog.
     * @param cardNum      The card number of the user.
     * @param userDataList The list of user data.
     */
    public TMoney(Frame owner, String cardNum, List<UserData>
userDataList) {

        super(owner, "Transfer Money", true);
        setIconImage(new ImageIcon("src/icon.png").getImage());
        this.cardNum = cardNum;
        this.userDataList = userDataList;
        v = new Validate();

        setLayout(new GridLayout(4, 2, 10, 10));
        accNo = new JTextField();
        moneyT = new JTextField();
        transButton = new JButton("Transfer Money");
        cancelButton = new JButton("Cancel");
```

```
        add(new JLabel("Enter Receiver's Amount Number:"));
        add(accNo);
        add(new JLabel("Enter Money:"));
        add(moneyT);
        add(transButton);
        add(cancelButton);

        // Action listener for the change button
        transButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (!v.isAccountNumber(accNo.getText()) &&
UserData.exitsAccNo(accNo.getText())) { // Fixed closing parenthesis
                    JOptionPane.showMessageDialog(null, "Please enter
valid account number..");
                }
                if (!v.isAmount(moneyT.getText())) {
                    JOptionPane.showMessageDialog(null, "Please enter
valid money number..");
                }
                double myOldBal =
Double.parseDouble(UserData.getData(cardNum, 5, userDataList));
                double reqAmount =
Double.parseDouble(moneyT.getText());
                String rCard = UserData.getData(accNo.getText(),
userDataList);
                double reOldBal =
Double.parseDouble(UserData.getData(rCard, 5, userDataList));
                double d = myOldBal - reqAmount;
                if (1000 >= d) {
                    JOptionPane.showMessageDialog(null, "You must
contain at least 1000 rupees in your account..");
                } else {
                    double myNewBal = myOldBal - reqAmount;
                    double reNewBal = reOldBal + reqAmount;
                    String eCheckS = UserData.getData(cardNum, 14,
userDataList);
                    String eCheckR = UserData.getData(rCard, 14,
userDataList);
                    UserData.setData(cardNum, 5,
Double.toString(myNewBal), userDataList);
                    UserData.setData(rCard, 5,
Double.toString(reNewBal), userDataList);
                    recordTransaction(UserData.getData(cardNum, 15,
userDataList),reqAmount,accNo.getText(),reqAmount);
                    if ("Yes".equals(eCheckS)) {
                        String msg1=
Emsg.moneyTransferSuccess(UserData.getData(cardNum,4,userDataList),Use
rData.getData(cardNum,15,userDataList),UserData.getData(rCard,15,userD
ataList),reqAmount,myNewBal);
                        new
EmailService().sendEmail(UserData.getData(cardNum, 3, userDataList),
"Money Transfer Successful", msg1, null,true); // Fixed closing
```

```
parenthesis
                    }
                    if ("Yes".equals(eCheckR)) {
                        String msg2=
Emsg.moneyReciverSuccess(UserData.getData(rCard,4,userDataList),UserDa
ta.getData(rCard,15,userDataList),UserData.getData(cardNum,15,userData
List),reqAmount,reNewBal);
                        new
EmailService().sendEmail(UserData.getData(rCard, 3, userDataList),
"Money Recived Successful", msg2, null,true); // Fixed closing
parenthesis
                    }
                    JOptionPane.showMessageDialog(null,"Money Transfer
Successful");
                }
            }
        });

        // Action listener for the cancel button
        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the dialog
            }
        });

        pack();
        setLocationRelativeTo(null);
    }
    private void recordTransaction(String accNumS,double
amountS,String accNumR,double amountR ) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter("records.txt", true))) {
            writer.write("Account Number: " + accNumS + "  TO  " +
accNumR +"\n");
            writer.write(  "Transfer" +" Time: " +
java.time.LocalDateTime.now() + "\n");
            writer.write( "Transferred"+" Amount: " + amountS + "
₹\n\n");
            writer.write("Account Number: " + accNumR + "  FROM  " +
accNumS +"\n");
            writer.write(  "Received" +" Time: " +
java.time.LocalDateTime.now() + "\n");
            writer.write( "Received"+" Amount: " + amountR + "
₹\n\n");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

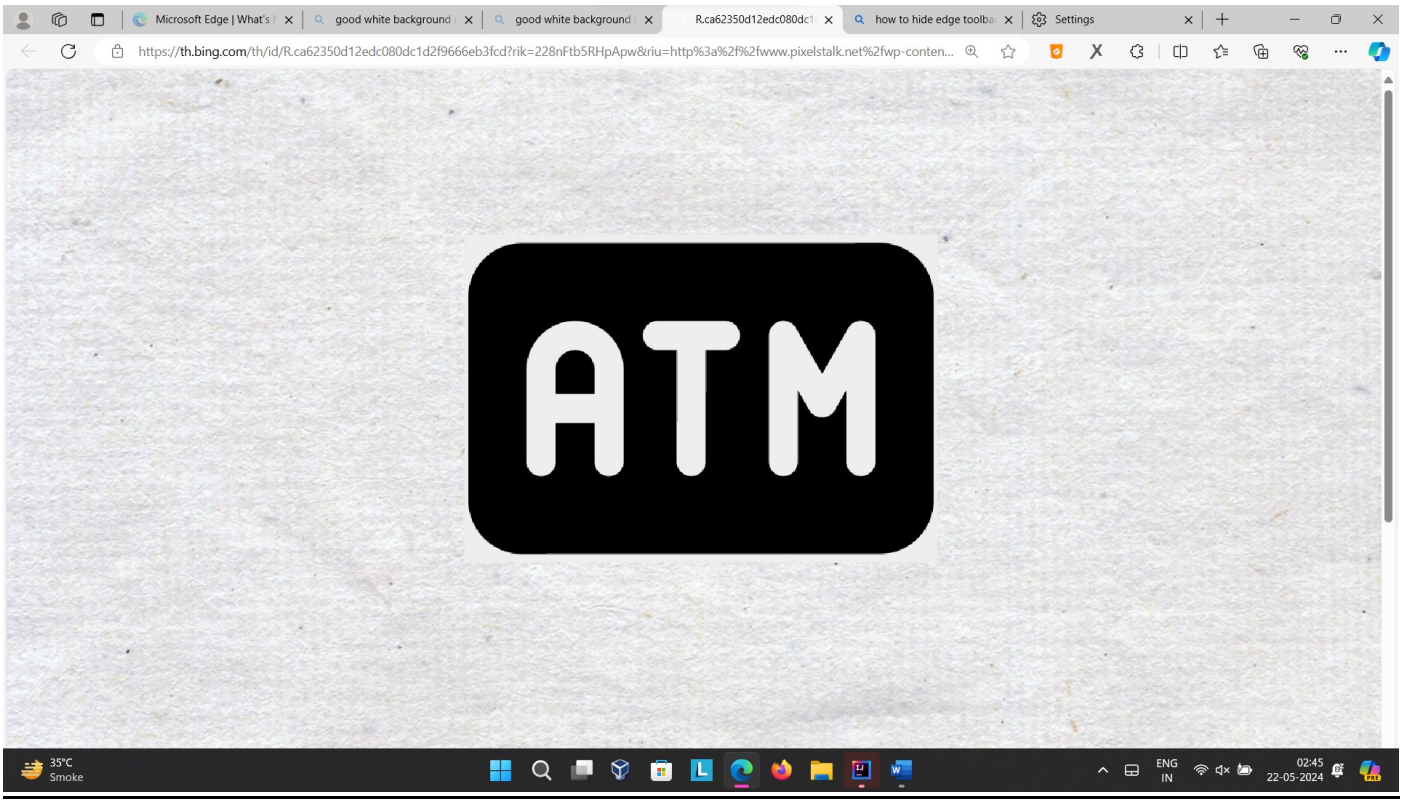
- Withdrawl.java

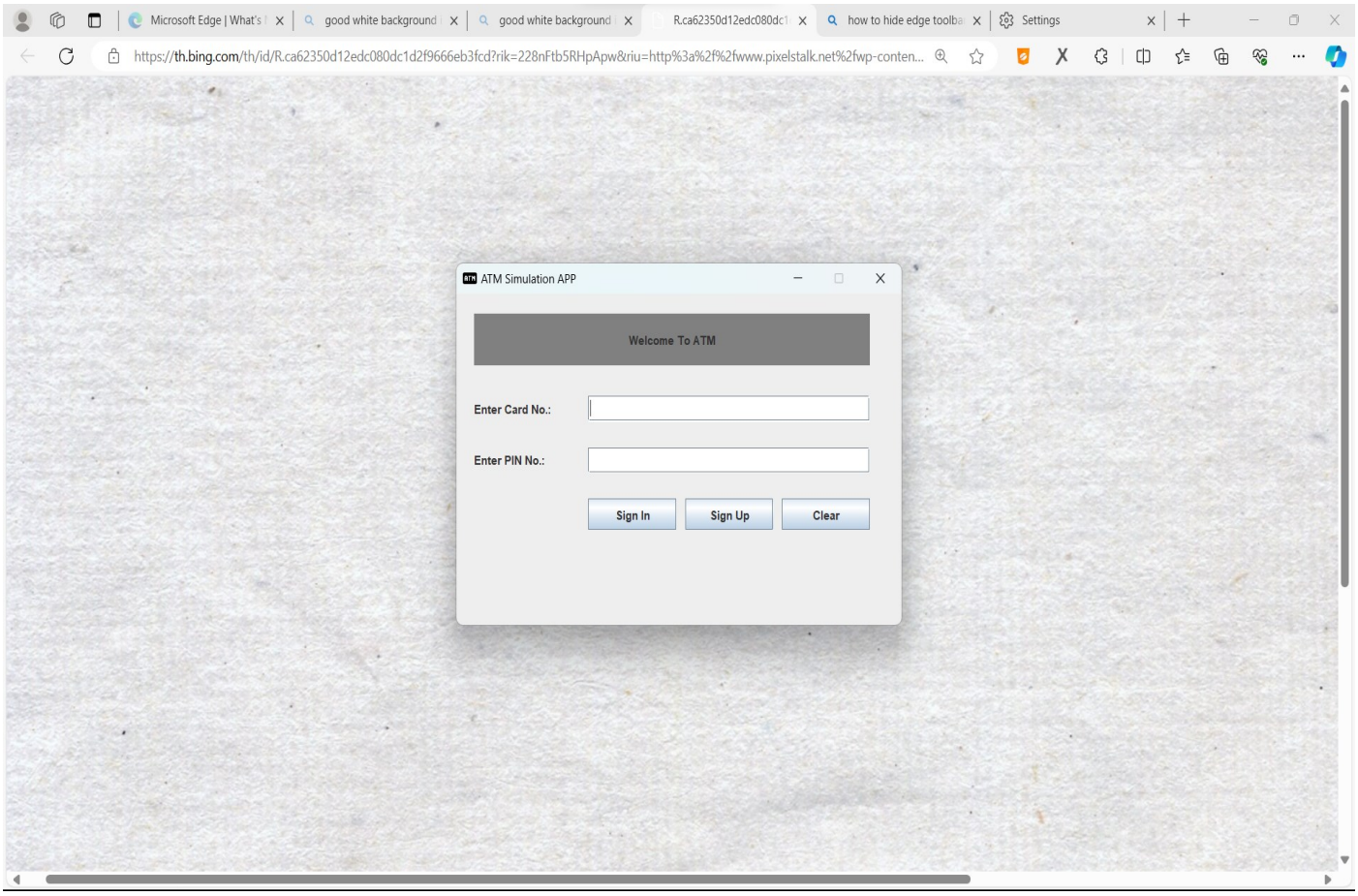
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

/**
 * Withdrawal class represents a dialog for withdrawing money from an
account.
 */
public class Withdrawal extends JDialog {

    private JTextField textField;
    private JButton withdrawButton, cancelButton;
    private List<UserData> userDataList;
    private String cardNum;
    private Validate v;
    private String accNum;

    /**
     * Constructor to initialize the Withdrawal dialog.
     *
     * @param owner        The parent frame.
     * @param cardNum      The card number for the account.
     * @param userDataList The list of user data.
     */
    public Withdrawal(Frame owner, String cardNum, List<UserData>
userDataList) {
        super(owner, "Withdrawal", true);
        setIconImage(new ImageIcon("src/icon.png").getImage());
        this.cardNum = cardNum;
        this.userDataList = userDataList;
        v = new Validate();

        // Fetch account number from userDataList
        for (UserData userData : userDataList) {
            if (userData.getCardNumber().equals(cardNum)) {
                accNum = userData.getAccNum();
                break;
            }
        }

        setLayout(new GridLayout(3, 2, 10, 10));
        textField = new JTextField();
        withdrawButton = new JButton("Withdraw");
        cancelButton = new JButton("Cancel");

        add(new JLabel("Enter Amount to Withdraw:"));
        add(textField);
```

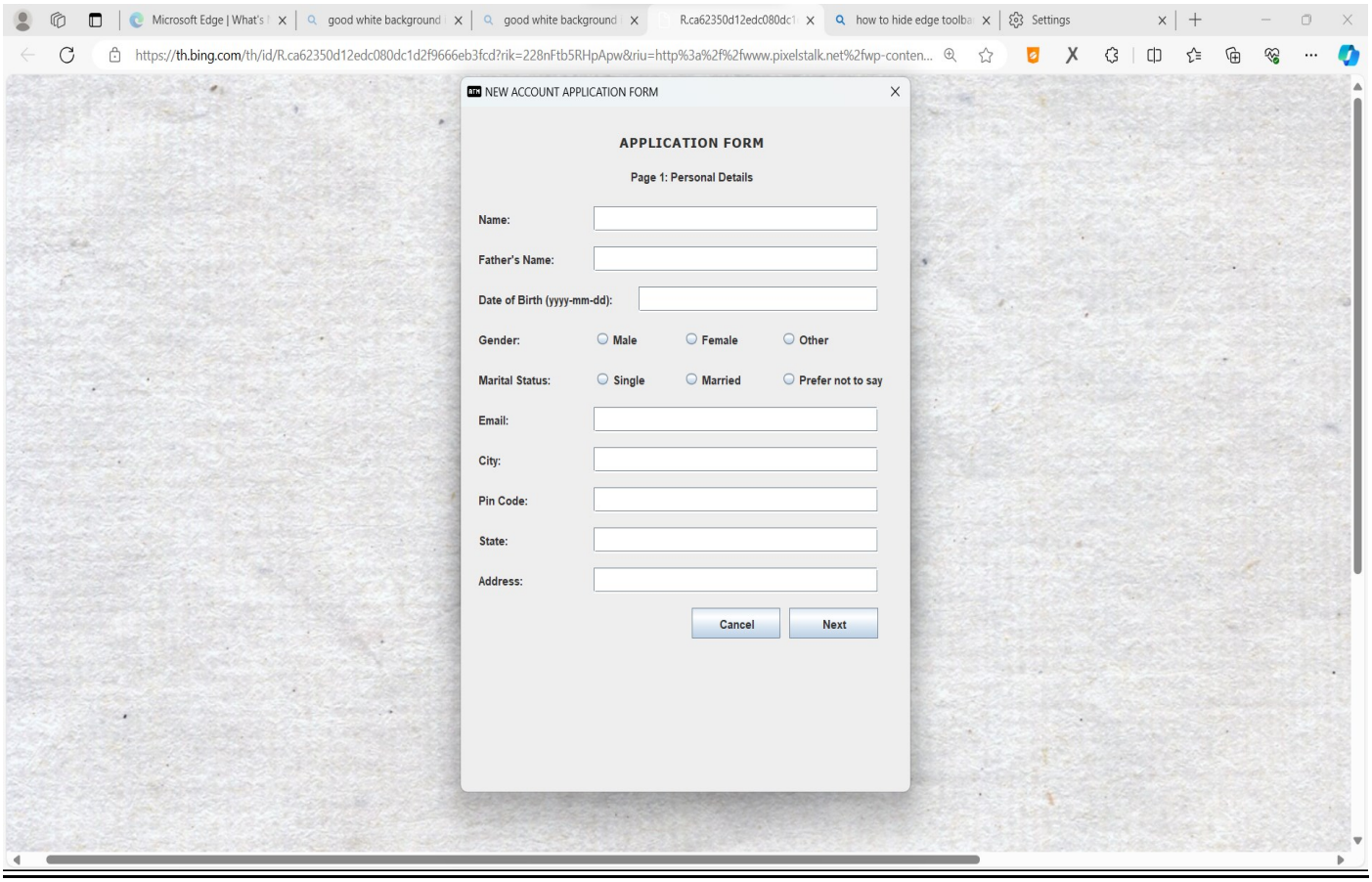
```
        add(withdrawButton);
        add(cancelButton);

        withdrawButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (v.isAmount(textField.getText())) {
                    double oldBalance =
Double.parseDouble(UserData.getData(cardNum, 5, userDataList));
                    double withdrawalAmount =
Double.parseDouble(textField.getText());

                    // Maximum withdrawal limit is 10,000₹
                    if (withdrawalAmount <= 10000) {
                        // Check if the withdrawal amount doesn't
cause the balance to go below the start amount of 1000₹
                        if (oldBalance - withdrawalAmount >= 1000) {
                            double newBalance = oldBalance -
withdrawalAmount;
                            UserData.setData(cardNum, 5,
Double.toString(newBalance), userDataList); // Update balance

                            // Record the withdrawal in the file
                            FileHandler.recordTransaction(accNum,
withdrawalAmount, "Withdrawal");

                            JOptionPane.showMessageDialog(null,
"Please collect your  money...");

                            if ("Yes".equals(UserData.getData(cardNum,
14, userDataList))) {
                                String
msg=Emsg.moneyWithdrawn(UserData.getData(cardNum, 4,
userDataList),Double.toString(withdrawalAmount),UserData.getData(cardN
um, 15, userDataList),Double.toString(newBalance));
                                new
EmailService().sendEmail(UserData.getData(cardNum, 3, userDataList),
"Withdrawal Confirmation", msg, null,true);
                            }

                            dispose(); // Close the dialog after
withdrawal
                        } else {
                            JOptionPane.showMessageDialog(null, "Your
balance cannot go below 1000₹.");
                        }
                    } else {
                        JOptionPane.showMessageDialog(null, "Maximum
withdrawal limit is 10,000₹.");
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Please enter
a valid amount.");
                }
```

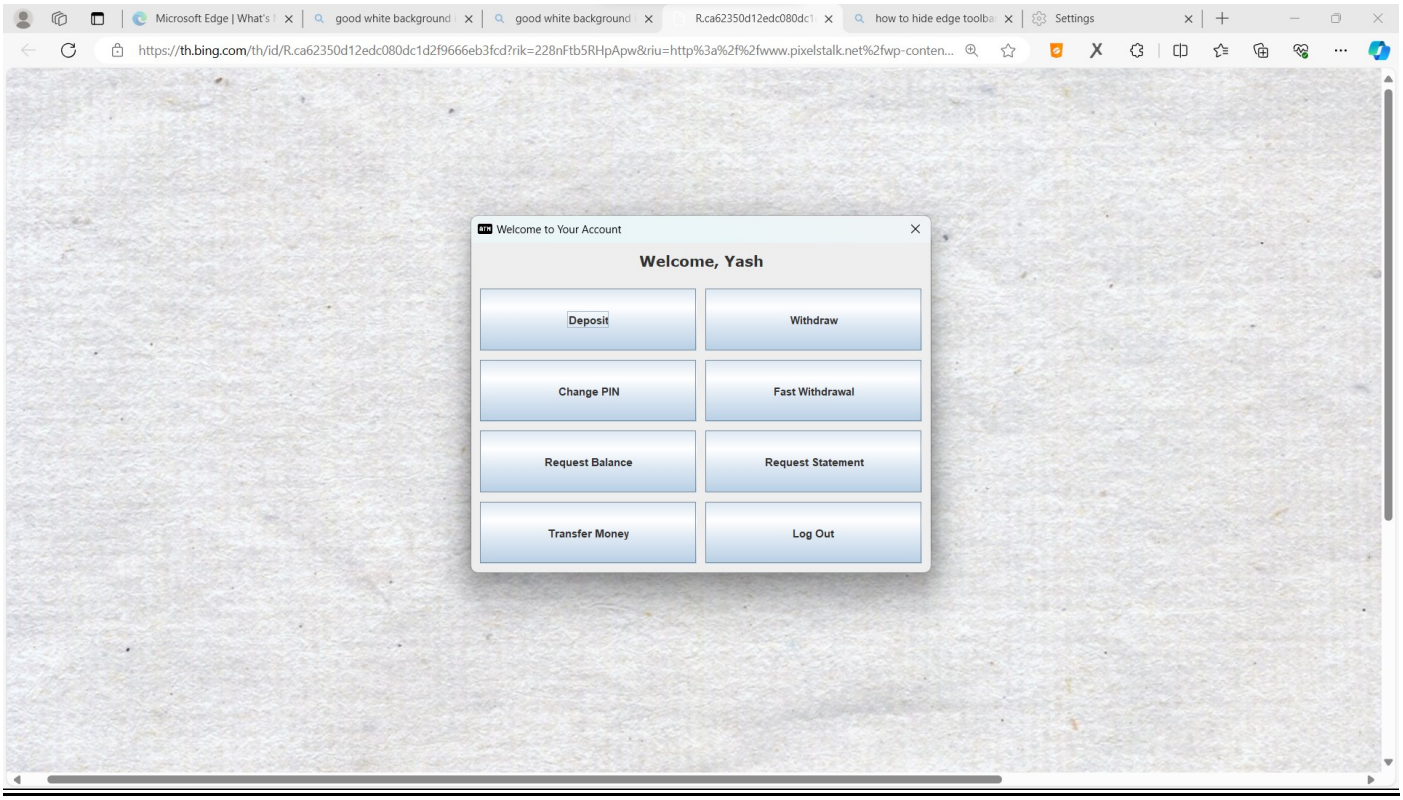
```
            }
        });

        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Close the dialog when Cancel button is
clicked
            }
        });

        pack();
        setLocationRelativeTo(null);
    }
}
```

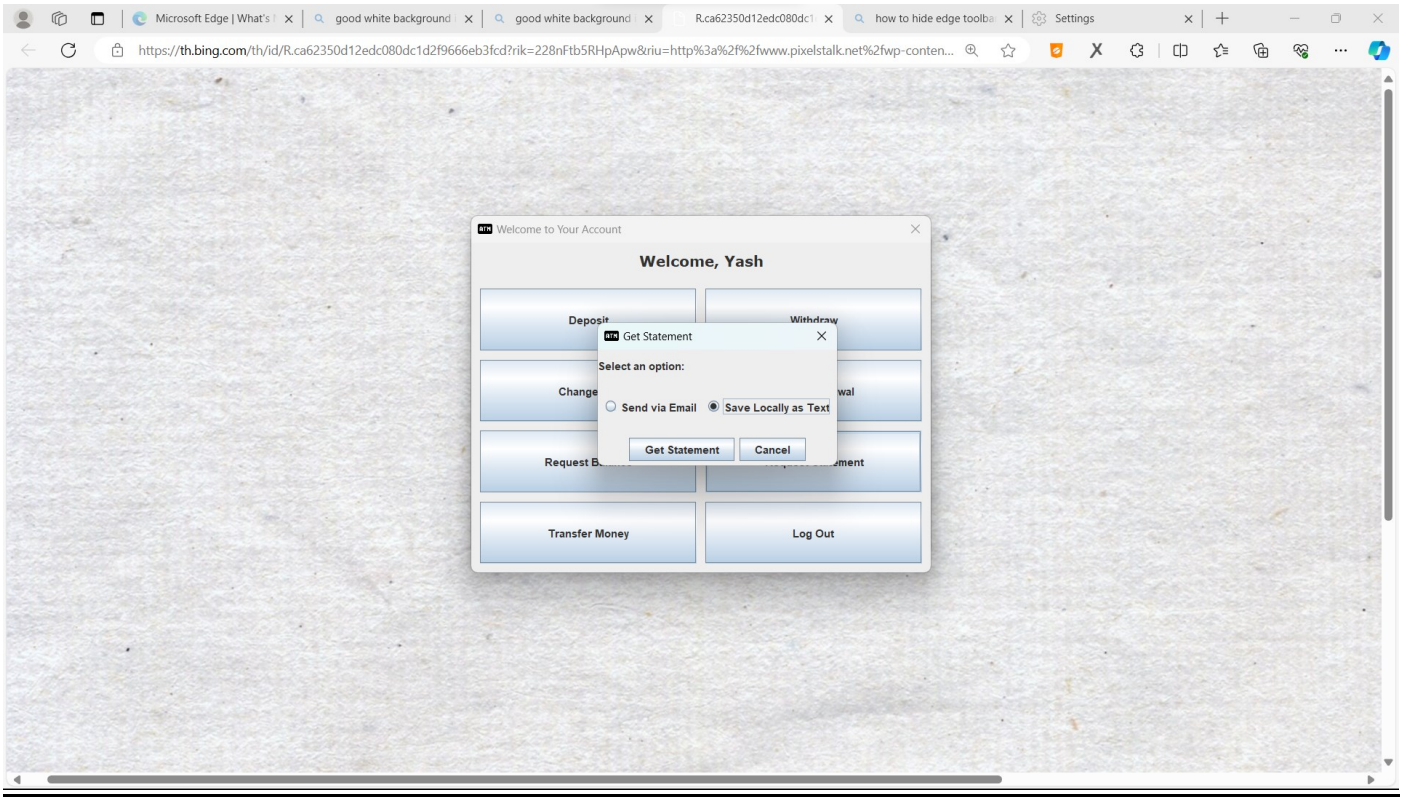
OUTPUT:




ATM Simulation APP
Welcome To ATM
Enter Card No.:
Enter PIN No.:
Sign In
Sign Up
Clear


NEW ACCOUNT APPLICATION FORM
APPLICATION FORM
Page 1: Personal Details
Name:
Father's Name:
Date of Birth (yyyy-mm-dd):
Gender:
Male
Female
Other
Marital Status:
Single
Married
Prefer not to say
Email:
City:
Pin Code:
State:
Address:
Cancel
Next


Welcome to Your Account
Welcome, Yash
Deposit
Withdraw
Change PIN
Fast Withdrawal
Request Balance
Request Statement
Transfer Money
Log Out


Welcome to Your Account
Welcome, Yash
Get Statement
Select an option:
Send via Email
Save Locally as Text
Get Statement
Cancel
Transfer Money
Log Out

THE END