Abstract:

The **SafeBanking Portal** is a Java-based desktop application designed to simulate an online banking system, providing users with essential banking functionalities in a simple and secure environment. Developed using Java Swing for the graphical user interface, the application offers a user-friendly platform that mimics the core operations of a real-world banking portal.

The primary objective of this project is to allow users to manage their personal banking activities, including logging in, viewing account details, depositing and withdrawing money, and accessing general banking information. The application begins with a home page that introduces users to the system and provides navigation to various modules via a menu bar.

The **Login Module** collects user information such as name, age, phone number, and bank name. Once logged in, users can access the **Account Module**, which displays their personal details and allows them to set an initial account balance. The **Deposit and Withdrawal Modules** enable users to update their balance with real-time validation to ensure accurate transactions. Additionally, the **Banking Info Module** provides users with a brief overview of the application's purpose and services.

Each feature is implemented using Swing components such as JFrame, JPanel, JLabel, JTextField, JButton, and JOptionPane. Layout managers like BorderLayout, FlowLayout, and GridLayout are used for proper component arrangement and responsive design.

The SafeBanking Portal serves as a foundational project for understanding GUI development and event-driven programming in Java. It demonstrates how simple yet essential banking functions can be modeled effectively using core programming concepts. In the future, the application can be extended to include database integration, authentication mechanisms, and transaction history to emulate a complete banking solution.

1. Introduction

In today's digital age, online banking has become an essential service, enabling users to manage their financial transactions conveniently and securely from anywhere. The **SafeBanking Portal** is a Java-based desktop application that simulates a basic online banking system, aimed at providing users with a simple and interactive banking experience.

This project is developed using **Java Swing**, a powerful GUI toolkit that enables the creation of user-friendly and responsive interfaces. The portal allows users to perform essential banking functions such as logging in with personal details, viewing account information, setting an initial balance, depositing funds, and withdrawing money. It also includes a brief informational module that highlights the features of the system.

The main goal of this application is to demonstrate the implementation of GUI-based applications in Java and to replicate core functionalities of a typical banking portal in an offline environment. Each module is carefully designed to ensure clarity, ease of use, and smooth interaction. This project lays the foundation for building more advanced banking systems by integrating databases, transaction history, and enhanced security features in the future.

2. Literature Survey:

This project is developed using **Java Swing**, a powerful GUI toolkit that enables the creation of user-friendly and responsive interfaces. The portal allows users to perform essential banking functions such as logging in with personal details, viewing account information, setting an initial balance, depositing funds, and withdrawing money.

2.1 Evolution of Online Banking Applications:

Online banking has evolved from simple balance-checking systems to sophisticated platforms offering a wide range of financial services. Early systems relied on basic HTML interfaces and local storage for maintaining user data. With the growth of digital banking, modern applications now integrate advanced technologies, robust databases, and enhanced security mechanisms to support secure and real-time transactions. This evolution reflects a shift toward customer-centric design, automation, and multi-platform accessibility.

2.2 GUI Development with Java Swing:

Java Swing is a widely used GUI toolkit for building desktop applications. It offers components like JFrame, JPanel, JButton, and JTextField to create interactive and visually structured interfaces. In this project, Swing is utilized to develop a user-friendly interface for banking functionalities. The use of layout managers such as GridLayout, FlowLayout, and BorderLayout ensures a responsive design that adapts well to various operations like login, account management, deposits, and withdrawals.

2.3 Event-Driven Programming in Java:

Swing applications are built on event-driven programming, where user actions like button clicks trigger specific methods. Action listeners are used to handle operations such as form submissions and balance updates. This approach enhances interactivity and allows for a modular and scalable design.

2.4 Simulating Core Banking Modules:

SafeBanking Portal integrates essential banking modules—Login, Account Management, Deposit, Withdrawal, and Banking Info—each mimicking real-life banking workflows. Though no real backend or database is used in this prototype, the logic and control flow are structured to replicate authentic user experiences.

2.5 Future Prospects in Online Banking Applications:

As financial services continue to digitize, future banking applications will likely integrate secure APIs, biometric authentication, real-time transaction tracking, and cloud storage. Projects like SafeBanking Portal serve as foundational prototypes for understanding the architecture and implementation of more robust systems.

3. Analysis and Design

3.1 System Requirements

The **SafeBanking Portal** is a standalone desktop-based Java application designed to simulate an online banking experience. It provides a graphical user interface for users to perform essential banking operations such as logging in, viewing account details, making deposits, and withdrawals. Below are the key system requirements:

Graphical User Interface (GUI): The application must offer a user-friendly interface using Java Swing to enable users to interact with banking modules easily.

User Authentication: The system must accept basic user credentials like name, age, phone number, and bank name through a login form.

Account Management: Users must be able to view their details and set an initial balance for their account.

Deposit and Withdrawal Operations: Users can deposit or withdraw money through intuitive forms and receive real-time balance updates.

Navigation Menu (Navbar): A top-level menu allows users to navigate across modules such as Home, Account, Deposit, Withdrawal, Login, and Banking Info.

Validation and Error Handling: The system must handle invalid inputs like non-numeric values or overdrawn withdrawals gracefully using error messages or alerts.

3.2 System Design

The design of the Online Banking Portal is divided into several logical components:

- **3.2.1 Graphical User Interface (GUI) Components:**

  Input Field: A text input (<input>) element to enter the user name

  Search Button: A button (<button>) that triggers the banking details fetch function when clicked.

Deposit & Withdrawal Button : Include input fields for entering an amount and buttons to execute the transaction and check updated balance.

Banking Info Page : Provides static information about the banking system's purpose and features.

- **3.2.2 Event Handling and Business Logic:**

  ActionListeners –

  Each button (e.g., Submit, Deposit, Withdraw) is linked with an ActionListener to perform respective tasks on click.

  Balance Management –

  Deposits add to the balance, and withdrawals subtract if sufficient funds are available. All changes reflect in real time.

  Error Handling –

  Withdrawals are only allowed if the requested amount is less than or equal to the balance. Otherwise, an alert is shown.

3.3 Use-Case Diagram of Online-Banking System

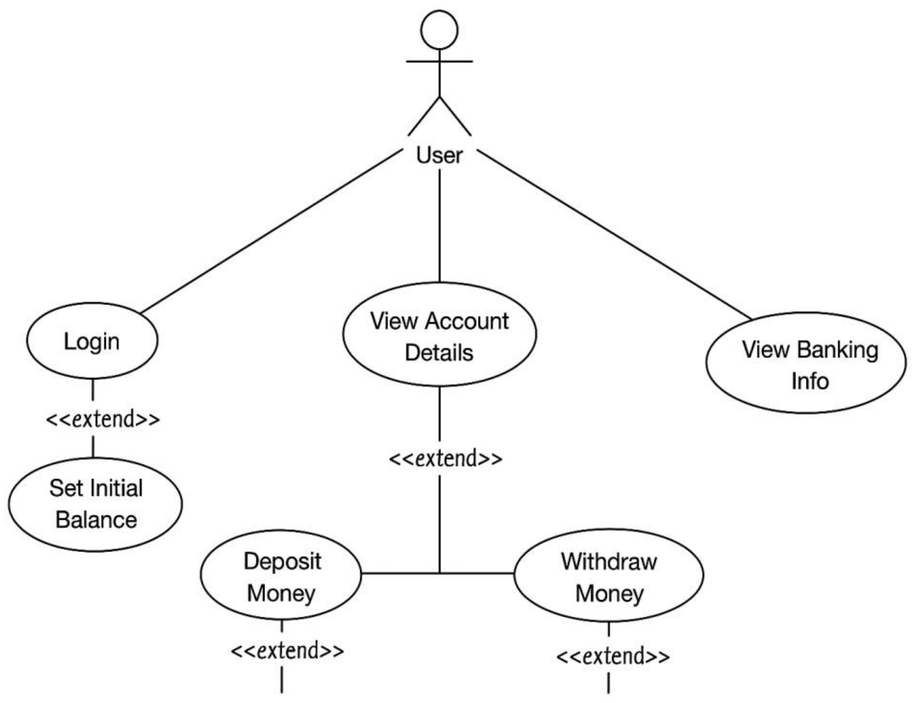


Fig1: Use-Case Diagram of Online-Banking Portal Management

This use case diagram shows how a User interacts with the Online Banking Portal. The main features include:

- Login, which can extend to Set Initial Balance for new users.
- View Account Details, with optional actions to Deposit or Withdraw Money.
- View Banking Info for general banking-related details.

The <<extend>> relationships indicate optional or conditional actions depending on the user's context, like viewing account info before making transactions.

3.4 Entity-Relationship Diagram of Online-Banking System

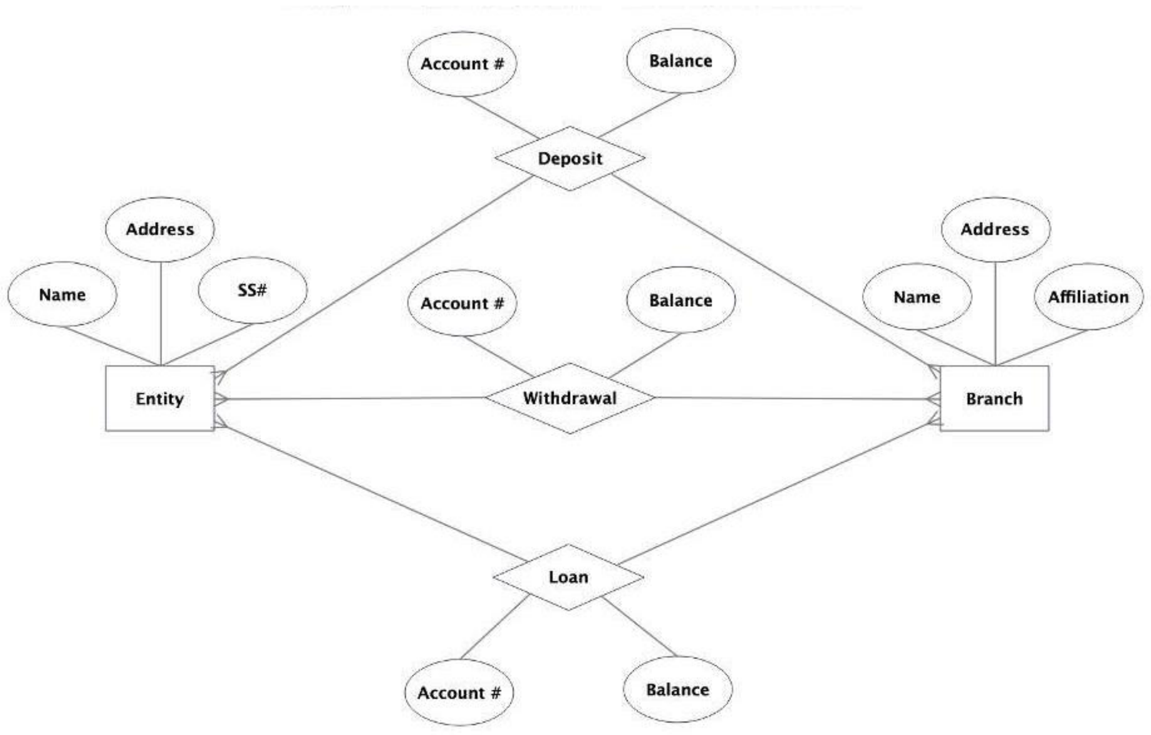


Fig2: E-R Diagram for Online-Banking Portal Management

This ER diagram represents the relationship between users, banking branches, and transactions in the system.

- Entity: Represents a customer, with attributes like Name, Address, and SS#.
- Branch: Represents a bank branch, with Name, Address, and Affiliation.

- Relationships:
    - Deposit, Withdrawal, and Loan are transactions involving:
        - The Entity (Customer)
        - The Branch
        - Each transaction includes Account and Balance.

This diagram shows how users interact with the bank through different financial operations across branches.

4. Implementation

4.1 Programming Language and Libraries

The Online Banking Portal Management is implemented using Java that bridges client-side interactivity with server-side data handling. The key technologies and libraries used are:

Some of the key Swing components used in this application include:

- JFrame: This is the main container used to create various windows like the Home Page, Login Page, Account Page, Deposit, Withdrawal, and Banking Info screens.
- JPanel: Panels are used to organize components into different sections such as the navigation bar and the center content area. The layout managers like BorderLayout, GridLayout, and FlowLayout are applied for structured alignment.
- JLabel: Labels are used to display static text such as titles, instructions, and user details like name, age, and bank name.
- JButton: These are interactive buttons like "Login", "Deposit", "Withdrawal", etc., each attached with ActionListener to define their specific behaviors upon clicking.
- JTextField: Input fields where users can enter data such as their name, age, phone number, bank name, and transaction amounts.
- JOptionPane: Used for displaying pop-up messages to inform the user of actions like balance updates or invalid operations (e.g., insufficient balance).

4.2 Entire code:

SafeBankingPortal.java

```
import java.awt.*;

import javax.swing.*;


class SafeBankingPortal {

    private double balance = 0.0;

    private String name, age, phone, bank;

    private JLabel balanceLabel;

    private JFrame frame;

    public SafeBankingPortal() {

        showHomePage();

    }

    private void showHomePage() {

        frame = new JFrame("SafeBanking Portal - Home");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setSize(800, 500);

        frame.setLayout(new BorderLayout());

        // Navbar Panel

        JPanel navbar = new JPanel();

        navbar.setBackground(Color.LIGHT_GRAY);
        navbar.setLayout(new FlowLayout(FlowLayout.LEFT, 20, 10));


        // Logo

        JLabel logo = new JLabel("SafeBanking", SwingConstants.LEFT);

        logo.setFont(new Font("Arial", Font.BOLD, 18));

        logo.setForeground(Color.BLUE);

        // Menu Buttons

        JButton homeButton = new JButton("Home");

        JButton accountButton = new JButton("Your Account");

        JButton bankingButton = new JButton("Banking");

        JButton depositButton = new JButton("Deposit");

        JButton withdrawalButton = new JButton("Withdrawal");

        JButton loginButton = new JButton("Login");

        // Add Buttons to Navbar

        navbar.add(logo);

        navbar.add(homeButton);

        navbar.add(accountButton);

        navbar.add(bankingButton);

        navbar.add(depositButton);

        navbar.add(withdrawalButton);

        navbar.add(loginButton);

        frame.add(navbar, BorderLayout.NORTH);

        // Welcome Message
```

```
        // Center Panel for Welcome Message and Description
        JPanel centerPanel = new JPanel(new GridLayout(2, 1));

                JLabel welcomeLabel = new JLabel("Welcome to SafeBanking", SwingConstants.CENTER);
        welcomeLabel.setFont(new Font("Arial", Font.BOLD, 30));
          JTextArea introText = new JTextArea("Welcome to SafeBanking – Your Trusted Online Banking Partner!\n\n" +
                                "Experience seamless, secure, and smart banking with SafeBanking! " +
                                        "Manage your accounts, transfer funds, pay bills, and track transactions—all from the comfort of your home. " +
                                 "With advanced security and an easy-to-use interface, banking has never been this effortless.\n\n" +
                                  "Bank smart, bank safe—only with SafeBanking!");
        introText.setFont(new Font("Arial", Font.PLAIN, 16));
        introText.setLineWrap(true);
        introText.setWrapStyleWord(true);
        introText.setEditable(false);
        introText.setOpaque(false);
        introText.setAlignmentX(Component.CENTER_ALIGNMENT);
        centerPanel.add(welcomeLabel);
        centerPanel.add(introText);
        frame.add(centerPanel, BorderLayout.CENTER);
        loginButton.addActionListener(e -> showLoginPage());
        accountButton.addActionListener(e -> showAccountPage());
        depositButton.addActionListener(e -> showDepositPage());
        withdrawalButton.addActionListener(e -> showWithdrawalPage());
        bankingButton.addActionListener(e -> showBankingInfo());
        frame.setVisible(true);
    }
    private void showLoginPage() {
        JFrame loginFrame = new JFrame("Login");
        loginFrame.setSize(400, 300);
        loginFrame.setLayout(new GridLayout(5, 2));
        loginFrame.add(new JLabel("Name:"));
        JTextField nameField = new JTextField();
        loginFrame.add(nameField);
        loginFrame.add(new JLabel("Age:"));
        JTextField ageField = new JTextField();
        loginFrame.add(ageField);
        loginFrame.add(new JLabel("Phone:"));
        JTextField phoneField = new JTextField();
        loginFrame.add(phoneField);
        loginFrame.add(new JLabel("Bank Name:"));
        JTextField bankField = new JTextField();
        loginFrame.add(bankField);
        JButton submitButton = new JButton("Submit");
        loginFrame.add(submitButton);


        submitButton.addActionListener(e -> {
            name = nameField.getText();
            age = ageField.getText();
            phone = phoneField.getText();
            bank = bankField.getText();
            loginFrame.dispose();
        });
        loginFrame.setVisible(true);
    }
    private void showAccountPage() {
        JFrame accountFrame = new JFrame("Your Account");
        accountFrame.setSize(400, 300);
        accountFrame.setLayout(new GridLayout(5, 1));
        accountFrame.add(new JLabel("Name: " + name));
        accountFrame.add(new JLabel("Age: " + age));
        accountFrame.add(new JLabel("Phone: " + phone));
        accountFrame.add(new JLabel("Bank: " + bank));
        JTextField balanceField = new JTextField("Set Initial Balance");
        accountFrame.add(balanceField);
        JButton setBalanceButton = new JButton("Set Balance");
        accountFrame.add(setBalanceButton);
        setBalanceButton.addActionListener(e -> {
            balance = Double.parseDouble(balanceField.getText());
                    JOptionPane.showMessageDialog(accountFrame, "Balance successfully
initialized!");
            accountFrame.dispose();
        });
        accountFrame.setVisible(true);
    }
    private void showDepositPage() {
        JFrame depositFrame = new JFrame("Deposit");
        depositFrame.setSize(300, 200);
        depositFrame.setLayout(new GridLayout(3, 1));
        JTextField depositField = new JTextField();
        depositFrame.add(new JLabel("Enter Amount to Deposit:"));
        depositFrame.add(depositField);
        JButton depositButton = new JButton("Deposit");
        depositFrame.add(depositButton);
        JButton checkBalanceButton = new JButton("Check Balance");
        depositFrame.add(checkBalanceButton);
        depositButton.addActionListener(e -> {
            balance += Double.parseDouble(depositField.getText());
        });
                                        checkBalanceButton.addActionListener(e          ->
JOptionPane.showMessageDialog(depositFrame, "Updated Balance: " + balance));
```

```
        depositFrame.setVisible(true);
    }

    private void showWithdrawalPage() {
        JFrame withdrawalFrame = new JFrame("Withdrawal");
        withdrawalFrame.setSize(300, 200);
        withdrawalFrame.setLayout(new GridLayout(3, 1));

        JTextField withdrawalField = new JTextField();
        withdrawalFrame.add(new JLabel("Enter Amount to Withdraw:"));
        withdrawalFrame.add(withdrawalField);
        JButton withdrawButton = new JButton("Withdraw");
        withdrawalFrame.add(withdrawButton);
        JButton checkBalanceButton = new JButton("Check Balance");
        withdrawalFrame.add(checkBalanceButton);

        withdrawButton.addActionListener(e -> {
            double amount = Double.parseDouble(withdrawalField.getText());
            if (amount <= balance) {
                balance -= amount;
            } else {
                JOptionPane.showMessageDialog(withdrawalFrame, "Insufficient Balance!");
            }
        });

                                        checkBalanceButton.addActionListener(e              ->
JOptionPane.showMessageDialog(withdrawalFrame, "Updated Balance: " + balance));

        withdrawalFrame.setVisible(true);

    }

    private void showBankingInfo() {

        JFrame bankingFrame = new JFrame("About SafeBanking");

        bankingFrame.setSize(400, 200);

            bankingFrame.add(new JLabel("SafeBanking helps users manage their money
efficiently!"), BorderLayout.CENTER);

        bankingFrame.setVisible(true);

    }


    public static void main(String[] args) {

        new SafeBankingPortal();

    }

}
```

5. Testing and Debugging

5.1. Testing

The SafeBanking Portal application was thoroughly tested to ensure robust functionality, user data integrity, and a seamless user experience. Each module of the system—from user login to account operations—was validated under various test cases. The following key aspects were tested:

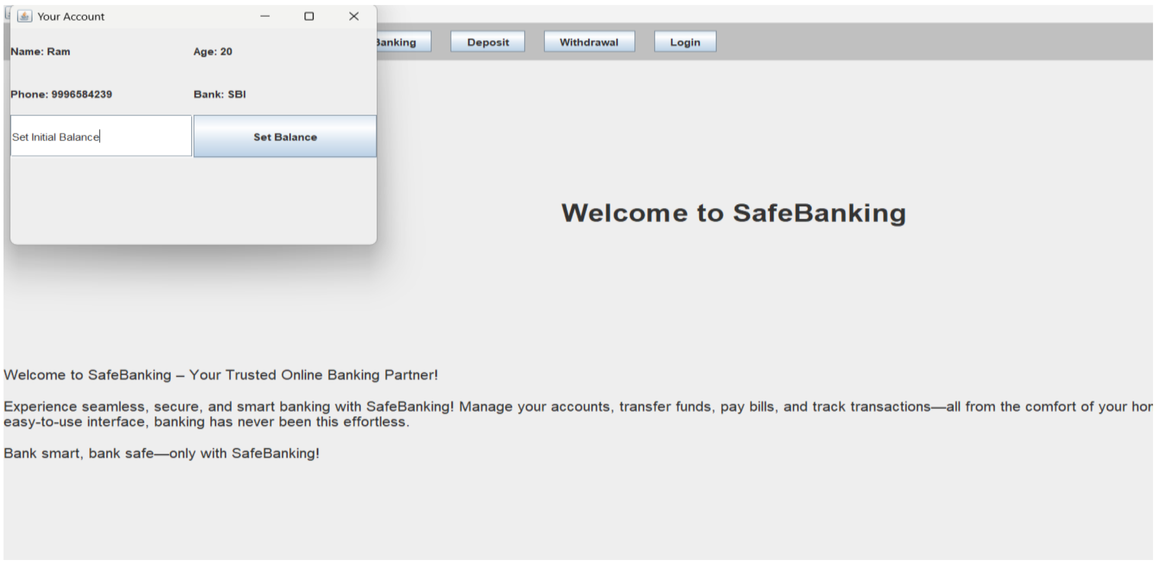
User Login and Input Validation: The application ensures that all input fields (name, age, phone number, and bank name) are properly filled before proceeding. The system prompts users with error messages using JOptionPane if any mandatory field is left blank, preventing incomplete or invalid data from being stored.

Account Creation and Navigation Flow: After successful login, users are directed to the main homepage, where all navigation options (Account, Deposit, Withdrawal, Check Balance, Banking Info) were tested to ensure smooth transitions between pages.

Balance Verification: The "Check Balance" option was tested to ensure it reflects the correct, updated balance after each transaction. All calculations were verified for accuracy.

Graphical User Interface Responsiveness: Swing components such as JButtons, JTextFields, and panels were checked for responsiveness and visual alignment across different resolutions. All user inputs and actions result in immediate visual feedback.

Error Handling: Scenarios such as invalid city names, API errors, and server connectivity issues were tested to ensure the application displays appropriate error messages and does not crash.



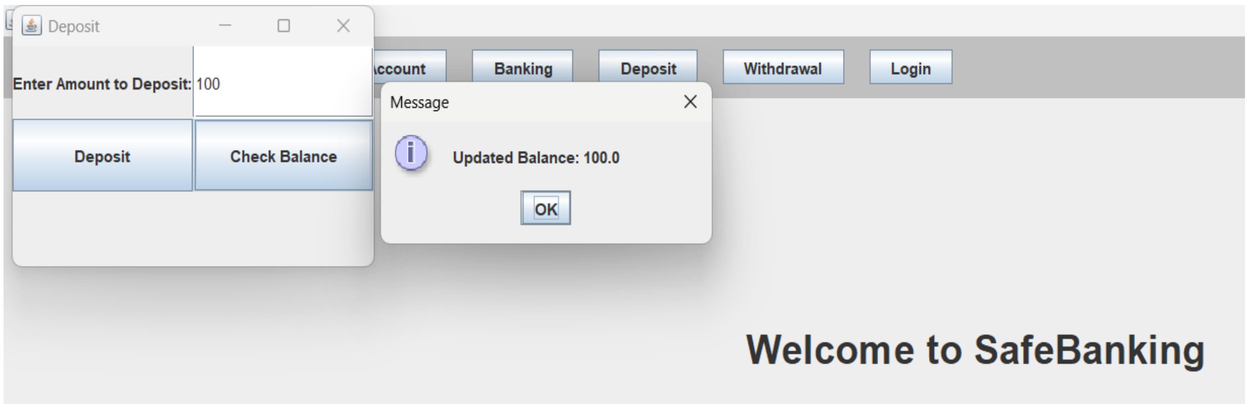
Name: Ram
Age: 20
Phone: 9996584239
Bank: SBI
Set Initial Balance

Welcome to SafeBanking

Welcome to SafeBanking – Your Trusted Online Banking Partner!

Experience seamless, secure, and smart banking with SafeBanking! Manage your accounts, transfer funds, pay bills, and track transactions—all from the comfort of your hor
easy-to-use interface, banking has never been this effortless.

Bank smart, bank safe—only with SafeBanking!

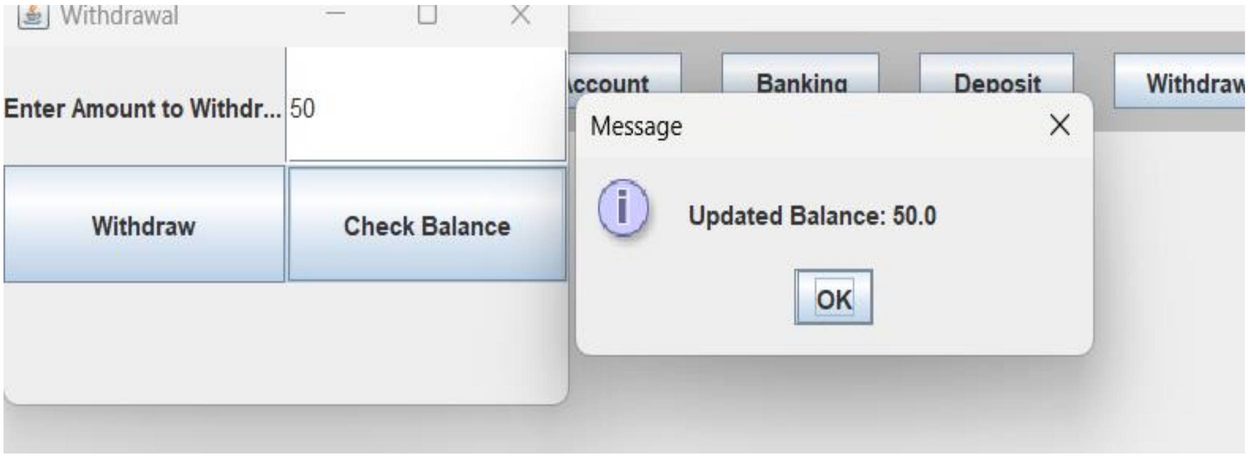


Enter Amount to Deposit: 100

Message

Updated Balance: 100.0

Welcome to SafeBanking



Enter Amount to Withdr... 50

Message

Updated Balance: 50.0

5.2. Debugging

The development journey of the SafeBanking Portal was accompanied by a meticulous debugging and testing phase, aimed at refining its functionality and ensuring a seamless, user-friendly experience. During the initial phases, challenges were encountered particularly in the transitioning of JPanels and the rendering of components within the application window. These visual and layout-related inconsistencies were resolved by carefully adjusting the layout managers (such as CardLayout, BorderLayout, etc.) and toggling the visibility of Swing components to ensure that the user interface responded predictably to user navigation and interactions.

One of the key aspects of this debugging phase involved input validation. Issues arising from invalid or empty user inputs were effectively addressed using try-catch blocks to capture exceptions like NumberFormatException, and informative JOptionPane dialog alerts were implemented to provide immediate feedback to users. This not only improved usability but also guided users towards proper data entry, enhancing the overall reliability of the application.

A significant focus was also placed on fine-tuning the core banking logic, especially around balance-related operations. Edge cases such as attempts to withdraw more money than available, or entering negative values, were thoroughly tested and corrected. The system now includes safeguards to prevent such erroneous operations, ensuring that the account balance remains consistent and secure under all circumstances.

Navigation across different screens—such as login, dashboard, deposit, and withdrawal interfaces—was another critical area that underwent debugging. Problems related to frame transitions and inconsistencies in screen rendering were resolved by efficiently managing action listeners and controlling the display logic of frames and panels. The result is a more intuitive and responsive user journey, free from glitches or confusion.

6. Conclusion:

The **SafeBanking Portal** serves as a well-rounded demonstration of the essential features typically expected in an online banking system, all developed using the robust capabilities of Java Swing. Designed with simplicity and interactivity in mind, this portal provides users with a clean and intuitive Graphical User Interface (GUI), enabling them to perform key banking operations with ease and confidence. From logging into the system, viewing account details, depositing or withdrawing funds, to instantly checking their updated balance, each feature is implemented with a focus on functionality and user experience.

At the heart of the application lies the intelligent integration of various Java Swing components, such as JButtons, JLabels, JPanels, JTextFields, and more. These elements work in harmony to craft a responsive interface that supports smooth navigation and seamless user interaction. By leveraging event-driven programming, the portal effectively responds to user actions, making it both dynamic and practical for real-world banking scenarios.

In addition to being a learning experience in building GUI-based financial software, the SafeBanking Portal underscores the significance of data handling, user feedback mechanisms, and real-time computation. As a standalone application, it already lays a solid foundation; however, there are promising opportunities for future development. Enhancements could include integration with a database management system (DBMS) for persistent storage of user data, a transaction history log for better financial tracking, and advanced security features such as password encryption and two-factor authentication to strengthen user data protection.

7. References:

1. **Java Documentation** – Official Java documentation for reference on language syntax, servlet APIs, and best practices. Available at: https://docs.oracle.com/en/java/

2. **HTML & CSS Tutorials by W3Schools** – Beginner-friendly guides to building and styling responsive web pages. Available at: https://www.w3schools.com/

3. **Stack Overflow** – Community-driven forum that provided support in resolving implementation issues related to JavaScript, Servlets, and API handling. Available at: https://stackoverflow.com/

4. **Modern JavaScript for the Impatient by Cay S. Horstmann (2020)** – A practical book focusing on modern JavaScript techniques including fetch, async/await, and DOM manipulation.