



# **SIMPLE BANKING APPLICATION**



## **A PROJECT REPORT**

*Submitted by*

**MOHANAVEL K (2302811724321070)**

*in partial fulfillment of requirements for the award of the course*

**CGB1201 – JAVA PROGRAMMING**

*in*

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY**

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by  
AICTE, New Delhi)

**SAMAYAPURAM – 621 112**

**DECEMBER, 2024**

**K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)**

**SAMAYAPURAM – 621 112**

**BONAFIDE CERTIFICATE**

Certified that this project report on “ **SIMPLE BANKING APPLICATION** ” is the bonafide work of **MOHANAVEL K (2302811724321070)** who carried out the project work during the academic year 2024 - 2025 under my supervision.



**Signature**

Dr. T. AVUDAIAPPAN M.E., Ph.D.,

**HEAD OF THE DEPARTMENT,**

Department of Artificial Intelligence,

K. Ramakrishnan College of Technology,

Samayapuram, Trichy -621 112.



**Signature**

Mrs. S. GEETHA M.E.,


**SUPERVISOR,**

Department of Artificial Intelligence,

K. Ramakrishnan College of Technology,

Samayapuram, Trichy -621 112.

Submitted for the viva-voce examination held on 3.12.24



**INTERNAL EXAMINER**



**EXTERNAL EXAMINER**

## **DECLARATION**

I declare that the project report on “ **SIMPLE BANKING APPLICATION**” is the result of original work done by me and best of my knowledge, similar work has not been submitted to “**ANNA UNIVERSITY CHENNAI**” for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This project report is submitted on the partial fulfillment of the requirement of the award of the **CGB1201 – JAVA PROGRAMMING**.



**Signature**

**MOHANAVEL K**

**Place:** Samayapuram

**Date:** 3/12/2024

## ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and indebtedness to our institution, **“K. Ramakrishnan College of Technology (Autonomous)”**, for providing us with the opportunity to do this project.

I extend our sincere acknowledgement and appreciation to the esteemed and honourable Chairman, **Dr. K. RAMAKRISHNAN, B.E.**, for having provided the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director, **Dr. S. KUPPUSAMY, MBA, Ph.D.**, for forwarding our project and offering an adequate duration to complete it.

I would like to thank **Dr. N. VASUDEVAN, M.TECH., Ph.D.**, Principal, who gave the opportunity to frame the project to full satisfaction.

I thank **Dr.T.AVUDAIAPPAN, M.E.,Ph.D.**, Head of the Department of **ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**, for providing her encouragement in pursuing this project.

I wish to convey our profound and heartfelt gratitude to our esteemed project guide **Mrs.S.GEETHA M.E.**, Department of **ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**, for her incalculable suggestions, creativity, assistance and patience, which motivated us to carry out this project.

I render our sincere thanks to the Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

## **VISION OF THE INSTITUTION**

To serve the society by offering top-notch technical education on par with global standards.

## **MISSION OF THE INSTITUTION**

- Be a centre of excellence for technical education in emerging technologies by exceeding the needs of industry and society.
- Be an institute with world class research facilities.
- Be an institute nurturing talent and enhancing competency of students to transform them as all- round personalities respecting moral and ethical values.

## **VISION AND MISSION OF THE DEPARTMENT**

To excel in education, innovation and research in Artificial Intelligence and Data Science to fulfill industrial demands and societal expectations.

Mission 1: To educate future engineers with solid fundamentals, continually improving teaching methods using modern tools.

Mission 2: To collaborate with industry and offer top-notch facilities in a conducive learning environment.

Mission 3: To foster skilled engineers and ethical innovation in AI and Data Science for global recognition and impactful research.

Mission 4: To tackle the societal challenge of producing capable professionals by instilling employability skills and human values.

## **PROGRAM EDUCATIONAL OBJECTIVES (PEOS)**

**PEO 1:** Compete on a global scale for a professional career in Artificial Intelligence and Data Science.

**PEO 2:** Provide industry-specific solutions for the society with effective communication and ethics.

**PEO 3:** Hone their professional skills through research and lifelong learning initiatives.

### **PROGRAM OUTCOMES**

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

### **PROGRAM SPECIFIC OUTCOMES (PSOs)**

- **PSO 1:** Capable of working on data-related methodologies and providing industry-focussed solutions.
- **PSO2:** Capable of analysing and providing a solution to a given real-world problem by designing an effective program.

## **ABSTRACT**

The Simple Banking Application is a fundamental yet comprehensive project aimed at simulating basic banking operations through a command-line interface. It leverages Java programming to implement essential features like account creation, deposits, withdrawals, balance inquiry, and transaction history maintenance. This project showcases the integration of core Java concepts such as object-oriented programming, exception handling, file I/O operations, and collections framework. The main goal is to provide users with a simple and efficient banking solution while emphasizing data security and integrity. The application maintains a modular structure, facilitating scalability and future enhancement possibilities. In today's fast-paced world, financial management systems play a pivotal role in daily life. This project seeks to serve as a prototype for understanding the workings of real-world banking systems. By focusing on simplicity and user-centric design, it enables a seamless experience for both novice and experienced users. Additionally, the project provides a foundation for students and developers to explore Java's capabilities in building secure and reliable software solutions. Through persistent data storage and error-handling mechanisms, the application ensures that all user transactions are accurately recorded and retrievable, making it a valuable educational tool.



## TABLE OF CONTENTS

<b>CHAPTER No.</b>	<b>TITLE</b>	<b>PAGE No.</b>
	<b>ABSTRACT</b>	viii
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 INTRODUCTION	1
	1.2 OBJECTIVE	2
<b>2</b>	<b>PROJECT METHODOLOGY</b>	<b>3</b>
	2.1 PROPOSED WORK	3
	2.2 BLOCK DIAGRAM	4
<b>3</b>	<b>JAVA PROGRAMMING CONCEPTS</b>	<b>5</b>
	3.1 OBJECT-ORIENTED PROGRAMMING (OOP)	<b>5</b>
	3.2 EXCEPTION HANDLING	<b>5</b>
	3.3 COLLECTIONS FRAMEWORK	<b>5</b>
	3.4 FILE I/O	<b>5</b>
	3.5 USER INPUT AND OUTPUT	<b>5</b>
<b>4</b>	<b>MODULE DESCRIPTION</b>	<b>6</b>
	4.1 ACCOUNT MANAGEMENT MODULE	6
	4.2 DEPOSIT MANAGEMENT MODULE	7
	4.3 WITHDRAW MANAGEMENT MODULE	8
	4.4 TRANSACTION HISTORY MODULE	9
	4.5 FILE HANDLING MODULE	10
<b>5</b>	<b>CONCLUSION</b>	<b>11</b>
	<b>REFERENCES</b>	<b>12</b>
	<b>APPENDICES</b>	<b>13</b>
	APPENDIX A – SOURCE CODE	<b>13</b>
	APPENDIX B – SCREEN SHOTS	<b>21</b>

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 INTRODUCTION**

The Simple Banking Application is an interactive, console-based application designed to perform basic financial transactions. The application comprises multiple modules, each tailored to handle a specific task, such as account creation or deposit processing. It operates on the principle of maintaining simplicity while ensuring the accuracy and security of data.

The project uses an account-centric approach, where each account is uniquely identified by an account number. Users can create accounts, deposit or withdraw money, and check balances. All transactions are logged and stored persistently, ensuring that data remains intact even after the application is closed.

The application also incorporates Java's exception-handling mechanisms to deal with common issues like invalid inputs or insufficient balances. This not only makes the system robust but also enhances user trust. The use of file handling ensures that all account data and transaction history are stored securely and can be accessed whenever needed.

In terms of design, the application adheres to software development best practices, ensuring a clear separation of concerns. Each module handles a specific functionality, making the codebase easier to manage and extend. This modular approach also ensures that future developers can add new features, such as interest calculation or fund transfers, without disrupting existing functionality.

## 1.2 OBJECTIVE

The primary objective of the Simple Banking Application is to bridge the gap between theoretical knowledge and practical implementation of Java programming. The application seeks to replicate the core functionalities of a banking system, such as account management, secure transactions, and financial record maintenance. The objectives are outlined as follows:

1. **User-Centric Design:** Create an intuitive interface that allows users to perform banking operations with minimal effort.
2. **Data Security:** Implement secure methods for storing and accessing sensitive account information.
3. **Error Management:** Ensure the system can handle errors gracefully, such as insufficient funds or invalid inputs.
4. **Learning and Development:** Serve as a learning tool for understanding Java concepts like OOP, collections, file handling, and exception management.
5. **Modularity:** Develop the application in a modular fashion to facilitate easy maintenance and potential future enhancements.

The project also emphasizes the importance of ethical software development, ensuring that user data is handled responsibly. By achieving these objectives, the project aims to demonstrate how Java can be utilized to solve real-world problems effectively and efficiently.

## **CHAPTER 2**

### **PROJECT METHODOLOGY**

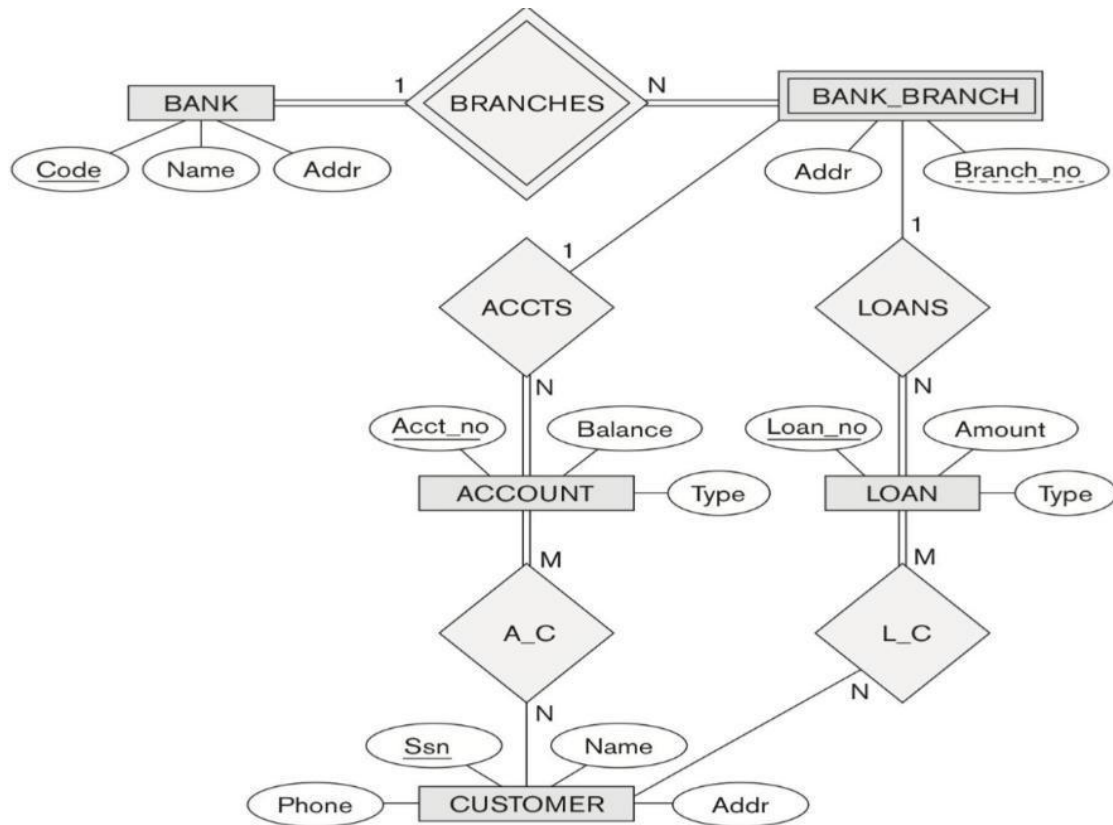
#### **2.1 PROPOSED WORK**

The proposed work for the Simple Banking Application is structured around the following tasks:

1. System Design:
  - Design a system that clearly defines modules for account management, transaction handling, and data storage.
2. Implementation:
  - Develop the core functionalities, such as account creation, deposit, withdrawal, and balance inquiry.
3. Data Security:
  - Use file encryption techniques or secure storage methods to protect sensitive account data.
4. Testing:
  - Conduct rigorous testing to identify and fix bugs, ensuring the system performs reliably under various scenarios.
5. Documentation:
  - Provide comprehensive documentation for the code and user manual, ensuring that future developers and users can understand the system.

This structured approach ensures that the application meets its objectives and delivers a high- quality user experience.

## 2.2 BLOCK DIAGRAM



## **CHAPTER 3**

### **JAVA PROGRAMMING CONCEPTS**

The Simple Banking Application utilizes a variety of Java programming concepts, each contributing to the robustness and efficiency of the system:

#### **3.1 OBJECT-ORIENTED PROGRAMMING (OOP)**

- Encapsulation: Account details are stored privately within the Account class, and access is provided through getter and setter methods.
- Inheritance and Polymorphism: Although not fully utilized in this project, the design allows for future extensions like different account types (e.g., savings, current).
- Abstraction: Users interact with a high-level interface, unaware of the underlying implementation details.

#### **3.2 EXCEPTION HANDLING**

- Custom exceptions are used to handle errors like insufficient funds or invalid inputs. This ensures that the application remains stable under all conditions.

#### **3.3 COLLECTIONS FRAMEWORK**

- A HashMap is used to store and manage multiple accounts efficiently, enabling quick retrieval and updates.

#### **3.4 FILE I/O**

- Transaction history and account details are stored persistently using file operations, ensuring data is not lost between sessions.

#### **3.5 USER INPUT AND OUTPUT**

- The Scanner class facilitates easy input and output, creating a seamless user experience.

These concepts not only make the application functional but also provide a solid foundation for understanding Java programming.

## **CHAPTER 4**

### **MODULE DESCRIPTION**

#### **4.1 ACCOUNT MANAGEMENT MODULE**

The Account Management Module is critical as it acts as the foundation of the application. It ensures that every user's account is unique and securely managed.

Features and Functionalities:

- **Account Creation:**
  - Generates a unique account number automatically.
  - Accepts user input for essential details like name, address, phone number, and initial deposit.
  - Validates all inputs, ensuring no blank or invalid data is saved.
- **Account Retrieval:**
  - Provides efficient access to account details using a HashMap or similar structure, where the account number serves as the key.
- **Account Modification:**
  - Allows users to update details like phone number or address securely. Changes are logged to prevent unauthorized edits.

Sub-Modules:

1. **Input Validation:**
  - Validates formats such as phone numbers, names, and addresses.
2. **Unique Identifier Generation:**
  - Ensures account numbers are non-repeating and sequential (or randomly

generated). Future Enhancements:

- Implement account categorization (e.g., Savings, Current, Premium).
- Introduce biometric authentication for secure access.
- Add APIs for external integration with other systems.

## 4.2 DEPOSIT MANAGEMENT MODULE

The Deposit Module ensures users can add money to their accounts seamlessly and securely. Features and Functionalities:

- Deposit Handling:
  - Accepts input for the account number and deposit amount.
  - Validates the amount to ensure it is positive.
  - Updates the balance immediately after successful validation.
- Transaction Logging:
  - Records the date, time, and amount of each deposit.
- Error Management:
  - Handles errors such as entering non-existent account numbers or non-numeric values.

Sub-Modules:

1. Real-Time Balance Update:
  - Ensures no discrepancies during simultaneous deposits in a multi-threaded environment.
2. Input Verification:
  - Checks for account existence and appropriate input

formatting. Future Enhancements:

- Enable deposits from linked external sources like digital wallets or bank transfers.
- Add functionality to accept deposits in multiple currencies with automatic currency conversion.



## 4.3 WITHDRAW MANAGEMENT MODULE

This module is critical for managing the withdrawal of funds securely and efficiently while maintaining data integrity.

Features and Functionalities:

- **Withdrawal Processing:**
  - Verifies if the account exists and if sufficient funds are available.
  - Deducts the requested amount from the balance upon validation.
- **Security Measures:**
  - Requires authentication (e.g., PIN or password) before processing.
- **Error Management:**
  - Handles scenarios like insufficient funds or invalid withdrawal

amounts gracefully. Sub-Modules:

1. **Overdraft Check:**
  - Prevents overdrawing unless the account type supports it.
2. **Transaction Logging:**
  - Logs each withdrawal with details like date, time, and amount

withdrawn. Future Enhancements:

- Introduce withdrawal limits per day or per transaction.
- Enable cashless withdrawals using QR codes or linked devices.

## 4.4 TRANSACTION HISTORY MODULE

The Transaction History Module provides a complete overview of all activities in a user's account. Features and Functionalities:

- Transaction Records:
  - Maintains a chronological log of all deposits, withdrawals, and balance inquiries.
  - Includes metadata like timestamps, transaction IDs, and account balance after each transaction.
- Search and Filter:
  - Enables users to filter transactions by date, type (deposit/withdrawal), or amount range.
- Data Export:
  - Allows users to export their transaction history as a CSV or PDF for

offline records. Sub-Modules:

1. Search Engine:
  - Provides optimized searches for large transaction logs.
2. Summary Generation:
  - Summarizes activities for specific periods (e.g., monthly

or yearly). Future Enhancements:

- Add graphical visualizations for spending patterns (e.g., bar charts or pie charts).
- Integrate with tax management systems to generate tax-related reports.

## 4.5 FILE HANDLING MODULE

This module ensures the persistence and security of all account and transaction data across application sessions.

Features and Functionalities:

- Data Storage:
  - Saves account details and transaction history in structured formats such as JSON, XML, or text files.
  - Uses separate files for account data and transaction logs for better organization.
- Data Retrieval:
  - Loads all saved data at application startup, ensuring continuity.
- Error Management:
  - Handles scenarios like missing or corrupted files by using backup or recovery mechanisms.

Sub-Modules:

1. Backup and Recovery:
    - Maintains regular backups of all data to prevent data loss.
  2. Encryption and Security:
    - Encrypts files to ensure unauthorized users cannot access sensitive data.
- Future Enhancements:

- Transition from file-based storage to database systems (e.g., MySQL, MongoDB) for scalability.
- Enable cloud-based storage for better accessibility and security.

## **CHAPTER 5**

### **CONCLUSION**

The Simple Banking Application demonstrates a well-structured approach to building a user-centric banking solution. Its modular architecture ensures clear separation of responsibilities, enabling efficient handling of critical banking operations such as account management, deposits, withdrawals, and transaction history tracking. By leveraging Java's robust programming capabilities, including object-oriented design, exception handling, and file operations, the application maintains both reliability and flexibility. Each module is designed with scalability in mind, ensuring that enhancements, such as integrating new features or upgrading existing functionalities, can be achieved with minimal effort.

A key strength of this application lies in its focus on data integrity and user security. Features such as input validation, transaction logging, and file handling ensure accurate and persistent data storage, while error handling mechanisms safeguard against unexpected issues. The inclusion of a transaction history module enhances transparency, allowing users to track their financial activities effortlessly. The modular design not only simplifies development and debugging but also paves the way for future extensions, such as enabling online banking features, implementing multi-factor authentication, or transitioning to cloud-based storage for enhanced accessibility.

This project serves as a practical demonstration of core software engineering principles, particularly modularity, reusability, and scalability. It is a solid foundation for exploring advanced concepts in financial technology, such as integration with APIs, blockchain-based security, or real-time analytics. The Simple Banking Application bridges the gap between theoretical knowledge and practical implementation, offering a valuable learning experience for developers. In conclusion, this application not only meets its functional requirements effectively but also exemplifies a scalable and maintainable solution that can be expanded into a comprehensive digital banking platform in the future.

## REFERENCES

### Web Resources:

Stack Overflow for practical problem-solving:

<https://stackoverflow.com/> Medium articles on modular design in

Java applications: <https://medium.com/>

Java Code Examples for File Handling and Exception

Management: <https://www.javatpoint.com/>

### Libraries and APIs Used:

Java Collections Framework: HashMap, ArrayList (Oracle Java SE 17 API). Java IO Package for File Handling: BufferedReader, BufferedWriter.

### Technical Blogs:

Baeldung, *"Understanding Java Exception Handling"*:

<https://www.baeldung.com/> DZone, *"Best Practices in Modular Java Development"*: <https://dzone.com/>

### Code Repositories and Samples:

GitHub repositories demonstrating banking applications:

<https://github.com/>

Open-source implementations of simple banking systems:

<https://opensource.com/>

## APPENDIX A-SOURCE CODE

```
import java.awt.*;
    import java.awt.event.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

class Account {
    private String
    accountNumber; private
    String accountHolder;
    private double balance;
    private List<String> transactionHistory = new ArrayList<>();

    public Account(String accountNumber, String accountHolder, double balance)
    { this.accountNumber = accountNumber;
    this.accountHolder =
    accountHolder; this.balance =
    balance;
    transactionHistory.add("Account created with initial balance: $" + balance);
    }

    public void deposit(double amount)
    { balance += amount;
    transactionHistory.add("Deposited: $" + amount + ", New Balance: $" + balance);
    }
```

```

public void withdraw(double amount) throws
    Exception { if (amount > balance) {
        throw new Exception("Insufficient Balance!");
    }
    balance -= amount;
    transactionHistory.add("Withdrew: $" + amount + ", New Balance: $" + balance);
}

public double getBalance()
    { return balance;
    }

public void viewTransactionHistory(TextArea textArea)
    { textArea.setText(""); // Clear previous text
    textArea.append("--- Transaction History for Account: " + accountNumber
        + " ---\n"); for (String record : transactionHistory) {
        textArea.append(record + "\n");
    }
    textArea.append("-----\n");
}

}

public class BankingAppAWT extends Frame {
    private static Map<String, Account> accounts = new
    HashMap<>(); private TextField accountNumberField,
    amountField, nameField; private TextArea
    transactionHistoryArea;
    private Label balanceLabel;

    public BankingAppAWT()

```

```

{ setTitle("Banking App");
setSize(600, 400);
setLayout(new
FlowLayout());
setResizable(false);

// Initialize components
accountNumberField = new
TextField(15); amountField = new
TextField(10); nameField = new
TextField(15);
transactionHistoryArea = new TextArea(10, 40);
transactionHistoryArea.setEditable(false);

balanceLabel = new Label("Balance:
$0.0"); Button createButton = new
Button("Create Account");
Button depositButton = new
Button("Deposit"); Button withdrawButton =
new Button("Withdraw");
Button checkBalanceButton = new Button("Check Balance");
Button viewHistoryButton = new Button("View Transaction History");

// Add components to the frame
add(new Label("Account
Number:"));
add(accountNumberField);
add(new Label("Account Holder Name:"));
add(nameField);
add(new Label("Amount:")); add(amountField);

```



```
add(createButton); add(depositButton);
add(withdrawButton); add(checkBalanceButton);
add(viewHistoryButton); add(balanceLabel);
add(transactionHistoryArea);

// Add button actions
createButton.addActionListener(e ->
createAccount());
depositButton.addActionListener(e ->
deposit());
withdrawButton.addActionListener(e ->
withdraw());
checkBalanceButton.addActionListener(e -> checkBalance());
viewHistoryButton.addActionListener(e -> viewTransactionHistory());

// Some sample accounts for testing
accounts.put("1001", new Account("1001", "John Doe",
500.0)); accounts.put("1002", new Account("1002", "Jane
Smith", 1000.0));
```

```

// Window close event
addWindowListener(new
    WindowAdapter() {
public void windowClosing(WindowEvent we)
    { dispose();
      }
    });

setVisible(true);
}

private void createAccount() {
    String accountNumber =
        accountNumberField.getText(); String
        accountHolder = nameField.getText();

    try {
        double balance =
            Double.parseDouble(amountField.getText()); if
            (accounts.containsKey(accountNumber)) {
                showMessage("Account number already
                    exists!"); return;
            }
        Account account = new Account(accountNumber, accountHolder, balance);
        accounts.put(accountNumber, account);
        showMessage("Account created
            successfully!"); clearFields();
    } catch (NumberFormatException e) {
        showMessage("Invalid initial balance! Please enter a valid number.");
    }
}

```

```
}
```

```
private void deposit() {  
    String accountNumber =  
        accountNumberField.getText(); Account account =  
        accounts.get(accountNumber);  
  
    if (account != null)  
  
        { try {  
double depositAmount =  
            Double.parseDouble(amountField.getText()); if  
            (depositAmount <= 0) {  
                showMessage("Deposit amount must be greater than  
                zero!"); return;  
            }  
            account.deposit(depositAmount);  
            showMessage("Deposit  
            successful!");  
            balanceLabel.setText("Balance: $" + account.getBalance());  
        } catch (NumberFormatException e) {  
            showMessage("Invalid deposit amount! Please enter a valid number.");  
        }  
        } else {  
            showMessage("Account not found!");  
        }  
    }  
}
```

```
private void withdraw() {  
    String accountNumber =  
        accountNumberField.getText(); Account account =  
        accounts.get(accountNumber);  
  
    if (account != null)  
        { try {  
            double withdrawalAmount =  
                Double.parseDouble(amountField.getText()); if  
                (withdrawalAmount <= 0) {  
                    showMessage("Withdrawal amount must be greater than  
                    zero!"); return;  
                }  
            }  
        }  
    }
```

```

    }
    account.withdraw(withdrawalAmount);
    showMessage("Withdrawal successful!");
    balanceLabel.setText("Balance: $" +
        account.getBalance());
} catch (Exception e)
{ showMessage("Error: " +
    e.getMessage());
}
} else {
showMessage("Account not found!");
}
}

```

```

private void checkBalance() {
    String accountNumber =
        accountNumberField.getText(); Account account =
        accounts.get(accountNumber);

    if (account != null) {
        balanceLabel.setText("Balance: $" + account.getBalance());
    } else {
        showMessage("Account not found!");
    }
}

```

```

private void viewTransactionHistory() {
    String accountNumber =
        accountNumberField.getText(); Account account =
        accounts.get(accountNumber);

    if (account != null)
        { account.viewTransactionHistory(transactionHistoryAr
            ea);
        } else {
            showMessage("Account not found!");
        }
}

```

```

private void showMessage(String message) {
    Dialog dialog = new Dialog(this,
        "Message", true); dialog.setLayout(new
        FlowLayout()); dialog.add(new
        Label(message));
}

```

```

        Button okButton = new Button("OK");
        okButton.addActionListener(e -> dialog.dispose());
        dialog.add(okButton);
    dialog.setSize(300, 150);
        dialog.setVisible(true);
    }

    private void clearFields()
    { accountNumberField.setText("");
      nameField.setText("");
      amountField.setText("");
    }

    public static void main(String[] args)
    { new BankingAppAWT();
    }
}

```

## APPENDIX B-SCREEN SHOT

