



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Rajiv Gandhi University of Knowledge Technologies-Nuzvid

Krishna, Andhra Pradesh -521202.

Banking System

Using Swing

A Summer Internship Report

Submitted in partial fulfillment for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

K. Pavani [N190061]

Under the Esteem Guidance of

Shri. T. Chandrasekhar sir [Asst. Prof]



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Rajiv Gandhi University of Knowledge Technologies - Nuzvid

Krishna, Andhra Pradesh – 521202

CERTIFICATE OF COMPLETION

This is to certify that the work entitled, “**Banking System**” is the bonafide work of **K. Pavani (ID No: N190061)** carried out under my guidance and supervision for third year summer internship of **Bachelor of Technology** in the department of Computer Science and Engineering under RGUKT IIIT Nuzvid. This work is done during the academic session May 2024 - June 2024, under our guidance.

T. Chandrasekhar

Assistant Professor,

Department of CSE

RGUKT Nuzvid.

Mrs. Nagarjuna Devi

Head of the Department,

Department of CSE

RGUKT Nuzvid



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – Nuzvid

Krishna, Andhra Pradesh – 521202.

CERTIFICATE OF EXAMINATION

This is to certify that the work entitled, “**Banking System**” is the bonafide work of **K. Pavani (ID No: N190061)** and here by accord our approval of it as a study carried out and presented in a manner required for its acceptance in Third year of **Bachelor of Technology** for which it has been submitted. This approval does not necessarily endorse or accept every statement made, opinion expressed or conclusion drawn, as recorded in this thesis. It only signifies the acceptance of this thesis for the purpose for which it has been submitted.

T.Chandrashekar

Assistant Professor,

Department of CSE

RGUKT Nuzvid.

Project Examiner

G.Jaya Krishna

Assistant Professor,

Department of CSE, RGUKT Nuzvid



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – Nuzvid

Krishna, Andhra Pradesh – 521202.

DECLARATION

We hereby declare that the project report entitled “**Banking System**” done by us under the guidance of **T.Chandrashekar**, Assistant Professor is submitted for the partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering during the academic session 2023- 2024 at RGUKT-Nuzvid.

We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites are mentioned in the references. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

Date: 06-07-2024

Place : Nuzvid

K.Pavani

N190061

ACKNOWLEDGEMENT

We would like to express our profound gratitude and deep regards to our guide **T.Chandrashekar** for his exemplary guidance, monitoring and constant encouragement to us throughout the B.Tech course. We shall always cherish the time spent with him during the course of this work due to the invaluable knowledge gained in the field of reliability engineering.

We are extremely grateful to for the confidence bestowed in us and entrusting our project entitled **“Banking System”**.

We express gratitude to Mrs **Nagarjuna Devi** (HOD of CSE) and other faculty members for being a source of inspiration and constant encouragement which helped us in completing the project successfully.

Our sincere thanks to all the batch mates of 2019 CSE, who have made our stay at RGUKT-NUZVID, a memorable one.

Finally, yet importantly, we would like to express our heartfelt thanks to our beloved God and parents for their blessings, our friends for their help and wishes for the successful completion of this project.

Table of Contents

ABSTRACT	7
CHAPTER 1	7
Introduction	7
1.1 Real-world Applications	7
CHAPTER 2	14
Proposed Method	8
2.1 Flowchart of Proposed Approach	8
2.2 Explanation of Each Component in the Flowchart	9
2.3 Algorithm	10-11
CHAPTER 3	33
3.1 Technologies Used	33
3.2 System Hardware	33
3.3 Results Table or Screenshots	33-36
3.4 Results Comparison with Other Approaches	36
CHAPTER 4	37
4.1 Conclusions	37
4.2 Key Achievements	37-38
4.3 Future Scope/Directions	38
4.4 References	38

ABSTRACT

The Java application presented is a Banking System implemented using Swing for graphical user interface (GUI). It allows users to perform essential banking operations such as creating accounts, logging in securely with password validation, conducting deposits and withdrawals with transaction management using locks for synchronization, and retrieving account balances and transaction histories. The system employs DAO (Data Access Object) design pattern for interacting with persistent storage, ensuring data integrity and security. With its intuitive interface and robust functionality, the Banking System provides a reliable platform for managing financial transactions effectively.

CHAPTER 1

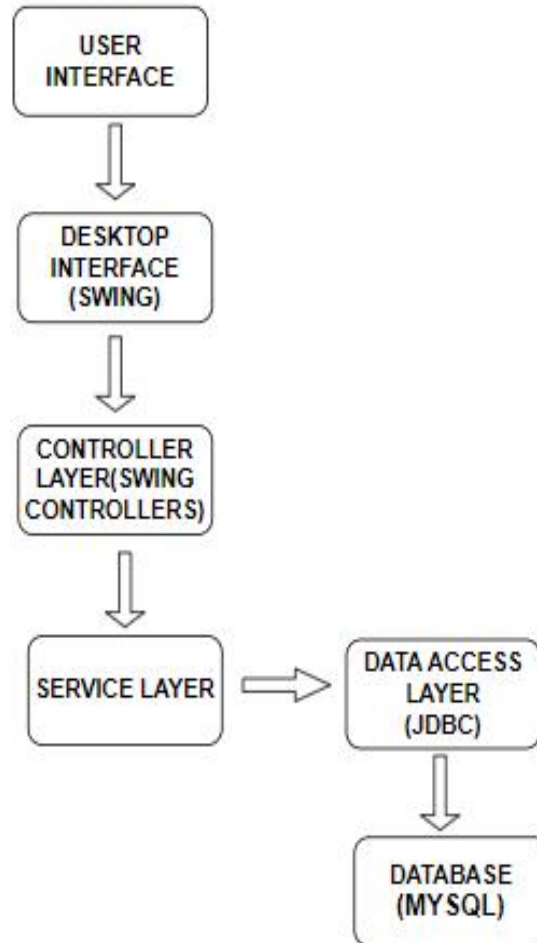
Introduction:

1.1 Real-world Applications

- **Secure Account Management:** Ensures account creation and login require valid passwords (`validatePassword`) for secure access and operation.
- **Transaction Handling:** Enables secure deposit and withdrawal operations (`performDeposit`, `performWithdrawal`) with synchronized locks (`depositLock`, `withdrawLock`) to prevent concurrent access issues.
- **Transaction History and Balance Inquiry:** Users can view their transaction history (`showTransactionHistory` method) and check their current balance (`showBalanceInquiry` method).
- **Data Access Object (DAO) Pattern:** Implements `AccountDAO` and `TransactionDAO` classes to abstract database operations, ensuring separation of concerns and facilitating maintenance.
- **Error Handling and Validation:** Incorporates comprehensive input validation to handle user errors, ensuring data integrity (`createAccount`, `login`, etc.) and providing meaningful error messages (`JOptionPane.showMessageDialog`).
- **Account Deletion :** Allows users to securely delete their accounts (`deleteAccount`), including associated transaction data deletion (`deleteTransactionsByAccount`), ensuring complete and safe account management.
- **User Interface (UI) Design:** Utilizes Java Swing components (`JFrame`, `JButton`, `JLabel`, etc.) to create an intuitive graphical interface for user interaction and information display.

CHAPTER 2

3.1 Flowchart of Proposed Approach



3.2 Explanation of Each Component in the Flowchart

3.2.1 User Interface:

Desktop Interface (Swing):

- Displays information and collects input from users.
- Presents data in a user-friendly manner using graphical elements such as forms, buttons, and menus.
- Sends user actions and inputs to the Controller Layer for processing.

3.2.2 Controller Layer:

Desktop Controller (Swing Handlers):

- Receives user requests and input from the UI.
- Validates and processes incoming data, ensuring it meets business rules and constraints.
- Coordinates the execution of tasks by invoking appropriate methods in the Service Layer.

3.2.3 Service Layer:

- Implements complex business operations and workflows that fulfill user requests.
- Executes tasks such as data processing, calculations, and validations.
- Orchestrates interactions between multiple components (e.g., coordinating multiple DAO calls).
- Ensures that business rules are enforced consistently across different parts of the application.

3.2.4 Data Access Layer:

- Performs CRUD (Create, Read, Update, Delete) operations on data entities.
- Abstracts and encapsulates database-specific operations to provide a consistent interface for the Service Layer.
- Optimizes data access and retrieval, handling transactions and ensuring data integrity.
- Converts data between database representations and Java objects (or application-specific data structures).

3.3 Algorithm

1. Initialization

- Initialize necessary variables and objects:
 - Account database (`AccountDAO`)
 - Transaction database (`TransactionDAO`)
 - User interface components (`JFrame`, buttons, input fields)

2. Main Menu Display

- Display the main menu options:
 - Create Account
 - Login
 - Delete Account
 - Exit

3. Menu Selection

- Wait for user input to select an option from the main menu.

4. Option Handling

- Depending on the selected option:
 - **Create Account:**
 - Prompt user for account details (account number, holder name, initial balance, password).
 - Validate input.
 - Create a new `AccountDetails` object.
 - Store account details in `AccountDAO`.
 - Add initial deposit transaction in `TransactionDAO`.
 - Display success/failure message.
 - **Login:**
 - Prompt user for account number and password.
 - Validate credentials against stored accounts in `AccountDAO`.
 - If valid, display banking operations menu.
 - If invalid, display error message and return to main menu.
 - **Delete Account:**
 - Prompt user for account number to delete.
 - Validate input.
 - Check if account exists in `AccountDAO`.
 - Delete associated transactions in `TransactionDAO`.
 - Delete account from `AccountDAO`.
 - Display success/failure message.
 - **Exit:**
 - Terminate the application.

5. Banking Operations Menu Display

- Display options available after successful login:
 - Deposit
 - Withdraw
 - Balance Inquiry
 - Transaction History
 - Logout

6. Operation Selection

- Wait for user input to select an operation from the banking operations menu.

7. Operation Handling

- Depending on the selected operation:
 - **Deposit:**
 - Prompt user for deposit amount.
 - Validate input.
 - Perform deposit operation in `TransactionDAO`.
 - Display success/failure message.
 - **Withdraw:**
 - Prompt user for withdrawal amount.
 - Validate input.
 - Perform withdrawal operation in `TransactionDAO`.
 - Display success/failure message.
 - **Balance Inquiry:**
 - Retrieve account balance from `AccountDAO`.
 - Display current balance.
 - **Transaction History:**
 - Retrieve transaction history from `TransactionDAO`.
 - Display transaction details.
 - **Logout:**
 - Return to the main menu.

8. Error Handling

- Implement error handling and validation throughout:
 - Invalid input formats.
 - Insufficient balances for withdrawals.
 - Account not found or already exists.

9. Termination

- Ensure proper termination of the application when exiting.

Implementations:

BankingSystem.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class Transaction {
    private String transactionType;
    private double amount;
    private String timestamp;

    public Transaction(String transactionType, double amount, String timestamp) {
        this.transactionType = transactionType;
        this.amount = amount;
        this.timestamp = timestamp;
    }

    // Getters and Setters
    public String getTransactionType() {
        return transactionType;
    }

    public void setTransactionType(String transactionType) {
        this.transactionType = transactionType;
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public String getTimestamp() {
        return timestamp;
    }
}
```

```

    public void setTimestamp(String timestamp) {
        this.timestamp = timestamp;
    }
}

class AccountDetails {
    private String accountNumber;
    private String accountHolderName;
    private double balance;
    private String password;

    public AccountDetails(String accountNumber, String accountHolderName, double balance,
String password) {
        this.accountNumber = accountNumber;
        this.accountHolderName = accountHolderName;
        this.balance = balance;
        this.password = password;
    }

    // Getters and Setters
    public String getAccountNumber() {
        return accountNumber;
    }

    public void setAccountNumber(String accountNumber) {
        this.accountNumber = accountNumber;
    }

    public String getAccountHolderName() {
        return accountHolderName;
    }

    public void setAccountHolderName(String accountHolderName) {
        this.accountHolderName = accountHolderName;
    }

    public double getBalance() {
        return balance;
    }

    public synchronized void setBalance(double balance) {

```

```

        this.balance = balance;

    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    // Method to validate password
    public boolean validatePassword(String inputPassword) {
        return password.equals(inputPassword);
    }
}

public class BankingSystem extends JFrame {
    private final AccountDAO accountDAO = new AccountDAO();
    private final TransactionDAO transactionDAO = new TransactionDAO(accountDAO); // Pass
    accountDAO to TransactionDAO
    private AccountDetails loggedInAccount;

    // Components for main menu
    private JLabel titleLabel;
    private JButton createAccountButton;
    private JButton loginButton;
    private JButton deleteAccountButton;
    private JButton exitButton;

    // Components for banking operations menu
    private JLabel bankingTitleLabel;
    private JButton depositButton;
    private JButton withdrawButton;
    private JButton balanceInquiryButton;
    private JButton transactionHistoryButton;
    private JButton logoutButton;

    // Main menu panel
    private JPanel mainMenuPanel;

```

```

// Locks for synchronization
private final Lock depositLock = new ReentrantLock();
private final Lock withdrawLock = new ReentrantLock();

public BankingSystem() {
    initializeUI();}
private void initializeUI() {
    try {
        // Apply Nimbus look and feel for modern appearance
        UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
    } catch (Exception e) {
        e.printStackTrace();
    }
    setTitle("Banking System");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 400);
    setLocationRelativeTo(null); // Center the frame on the screen
    setLayout(new BorderLayout());

    // Disable frame resizing
    setResizable(false);

    // Initialize main menu components
    mainMenuPanel = new JPanel(new GridLayout(5, 1, 10, 10));
    titleLabel = new JLabel("Banking System Menu", SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 20)); // Setting font size and style
    titleLabel.setForeground(Color.BLUE); // Setting font color
    createAccountButton = new JButton("Create Account");
    loginButton = new JButton("Login");
    deleteAccountButton = new JButton("Delete Account");
    exitButton = new JButton("Exit");

    mainMenuPanel.add(titleLabel);
    mainMenuPanel.add(createAccountButton);
    mainMenuPanel.add(loginButton);
    mainMenuPanel.add(deleteAccountButton);
    mainMenuPanel.add(exitButton);

    // Action listeners for main menu buttons
    createAccountButton.addActionListener(new ActionListener() {
        @Override

```

```

        public void actionPerformed(ActionEvent e) {
            createAccount();
        });

loginButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        login();
    }
});

deleteAccountButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        deleteAccount();
    }
});

exitButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        exitApplication();
    }
});

// Add main menu panel to frame
add(mainMenuPanel, BorderLayout.CENTER);
setVisible(true);
}

public void createAccount() {
    while (true) {
        JTextField accountNumberField = new JTextField();
        JTextField accountHolderField = new JTextField();
        JTextField balanceField = new JTextField();
        JPasswordField passwordField = new JPasswordField();
        Object[] message = {
            "Account Number:", accountNumberField,
            "Account Holder Name:", accountHolderField,
            "Initial Balance:", balanceField,
            "Password:", passwordField
        };
    }
};

```



```

        int option = JOptionPane.showConfirmDialog(this, message, "Create Account",
JOptionPane.OK_CANCEL_OPTION);
        if (option == JOptionPane.OK_OPTION) {

try {
    String accountNumber = accountNumberField.getText().trim();
    String accountHolderName = accountHolderField.getText().trim();
    String balanceStr = balanceField.getText().trim();
    double initialBalance = Double.parseDouble(balanceStr);
    String password = new String(passwordField.getPassword());

    // Validate account number
    if (accountNumber.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Account number cannot be empty.");
        continue; // Show dialog again
    }

    // Validate account holder name
    if (accountHolderName.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Account holder name cannot be empty.");
        continue; // Show dialog again
    }

    // Validate initial balance
    if (balanceStr.isEmpty() || initialBalance < 0) {
        JOptionPane.showMessageDialog(this, "Initial balance must be a valid positive
number.");
        continue; // Show dialog again
    }

    // Password validation
    if (password.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Password cannot be empty.");
        continue; // Show dialog again
    }

    // Create new account object
    AccountDetails newAccount = new AccountDetails(accountNumber,
accountHolderName, initialBalance, password);
    boolean success = accountDAO.createAccount(newAccount);

```

```

if (success) {
    // Add initial deposit transaction
    boolean transactionSuccess = transactionDAO.addTransaction(new
Transaction("Deposit", initialBalance, transactionDAO.getCurrentTimestamp()),
accountNumber);
    if (transactionSuccess) {

        JOptionPane.showMessageDialog(this, "Account created successfully.");
    }
else {
        JOptionPane.showMessageDialog(this, "Failed to add initial deposit transaction.");
    }
    break; // Exit the loop on successful account creation
} else {
    JOptionPane.showMessageDialog(this, "Failed to create account.");
}
}

catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(this, "Invalid input format. Please enter valid input
values.");
}
} else {
    // User clicked Cancel or closed the dialog
    break; // Exit the loop
}
}
}

public void login() {
    boolean loggedIn = false;

    while (!loggedIn) {
        JTextField accountNumberField = new JTextField();
        JPasswordField passwordField = new JPasswordField();

        Object[] message = {
            "Account Number:", accountNumberField,
            "Password:", passwordField
        };

        int option = JOptionPane.showConfirmDialog(this, message, "Login",
JOptionPane.OK_CANCEL_OPTION);

```

```

        if (option == JOptionPane.OK_OPTION) {
            String accountNumber = accountNumberField.getText();
            String password = new String(passwordField.getPassword());

            AccountDetails account = accountDAO.getAccount(accountNumber);
            if (account != null && account.validatePassword(password)) {

                JOptionPane.showMessageDialog(this, "Login successful. Welcome " +
                    account.getAccountHolderName());
                loggedInAccount = account;
                showBankingOperations();
                loggedIn = true;
            } else {
                JOptionPane.showMessageDialog(this, "Invalid account number or password. Please
try again.");
            }
        } else {
            // User clicked cancel or closed the dialog
            return;
        }
    }
}

public void deleteAccount() {
    boolean inputValidated = false;

    while (!inputValidated) {
        String accountNumber = JOptionPane.showInputDialog(this, "Enter the account number to
delete:");

        // Check if user canceled the input dialog
        if (accountNumber == null) {
            return; // Exit method if user canceled
        }

        // Trim whitespace and check if account number is empty
        accountNumber = accountNumber.trim();
        if (accountNumber.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please enter account number.");
            continue; // Prompt user again if account number is empty
        }

        // Check if account exists

```

```

AccountDetails account = accountDAO.getAccount(accountNumber);
if (account == null) {
    JOptionPane.showMessageDialog(this, "Account does not exist.");
    continue; // Prompt user again if account does not exist
}

// Attempt to delete associated transactions
boolean transactionsDeleted = transactionDAO.deleteTransactionsByAccount(accountNumber);

if (transactionsDeleted) {
    JOptionPane.showMessageDialog(this, "Associated transactions deleted successfully.");
} else {
    JOptionPane.showMessageDialog(this, "No transactions made by this account.");
}

// Delete account
boolean deleted = accountDAO.deleteAccount(accountNumber);
if (deleted) {
    JOptionPane.showMessageDialog(this, "Account deleted successfully.");
    inputValidated = true; // Exit the loop since account deletion was successful
} else {
    JOptionPane.showMessageDialog(this, "Failed to delete account.");
}
}}
private void showBankingOperations() {
    // Hide banking operations panel
    getContentPane().removeAll();

    // Initialize components for banking operations menu
    JPanel bankingMenuPanel = new JPanel(new GridLayout(6, 1, 10, 10));
    bankingTitleLabel = new JLabel("Account Services", SwingConstants.CENTER);
    bankingTitleLabel.setFont(new Font("Arial", Font.BOLD, 20)); // Setting font size and
style bankingTitleLabel.setForeground(Color.RED); // Setting font color
    depositButton = new JButton("Deposit");
    withdrawButton = new JButton("Withdraw");
    balanceInquiryButton = new JButton("Balance Inquiry");
    transactionHistoryButton = new JButton("Transaction History");
    logoutButton = new JButton("Logout");

```

```

bankingMenuPanel.add(bankingTitleLabel);
    bankingMenuPanel.add(depositButton);
    bankingMenuPanel.add(withdrawButton);
    bankingMenuPanel.add(balanceInquiryButton);
    bankingMenuPanel.add(transactionHistoryButton);
    bankingMenuPanel.add(logoutButton);

    // Action listeners for banking operations menu buttons
    depositButton.addActionListener(new ActionListener() {
        @Override

public void actionPerformed(ActionEvent e) {
    performDeposit();
    });

    withdrawButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            performWithdrawal();
        }
    });

    balanceInquiryButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            showBalanceInquiry();
        }
    });

    transactionHistoryButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            showTransactionHistory();
        }
    });

    logoutButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            logout();

```

```

    } });

    // Add banking menu panel to frame
    add(bankingMenuPanel, BorderLayout.CENTER);
    revalidate();
    repaint();
}

private void performDeposit() {
    boolean validAmountEntered = false;

    while (!validAmountEntered) {
        try {
            String amountStr = JOptionPane.showInputDialog(this, "Enter deposit amount:");
            if (amountStr == null) {
                // User clicked cancel or closed the dialog
                return;
            }

            double depositAmount = Double.parseDouble(amountStr);
            if (depositAmount <= 0) {
                JOptionPane.showMessageDialog(this, "Deposit amount must be greater than zero.");
            } else {
                synchronized (depositLock) {
                    boolean depositSuccess = transactionDAO.deposit(depositAmount,
loggedInAccount.getAccountNumber());
                    if (depositSuccess) {
                        JOptionPane.showMessageDialog(this, "Deposit successful.");
                        validAmountEntered = true;
                    } else {
                        JOptionPane.showMessageDialog(this, "Failed to process deposit.");
                    }
                }
            }

        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Invalid input format. Please enter a valid
number.");
        }
    }
}

```

```

private void performWithdrawal() {
    boolean validAmountEntered = false;

    while (!validAmountEntered) {
        try {
            String amountStr = JOptionPane.showInputDialog(this, "Enter withdrawal amount:");
            if (amountStr == null) {
                // User clicked cancel or closed the dialog
                return;
            }

            double withdrawAmount = Double.parseDouble(amountStr);
            if (withdrawAmount <= 0) {
                JOptionPane.showMessageDialog(this, "Withdrawal amount must be greater than
zero.");
            } else {
                synchronized (withdrawLock) {
                    boolean withdrawSuccess = transactionDAO.withdraw(withdrawAmount,
loggedInAccount.getAccountNumber());
                    if (withdrawSuccess) {
                        JOptionPane.showMessageDialog(this, "Withdrawal successful.");
                        validAmountEntered = true;
                    } else {
                        JOptionPane.showMessageDialog(this, "Failed to process withdrawal.
Insufficient balance.");
                    }
                }
            }
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Invalid input format. Please enter a valid
number.");
        }
    }

    private void showBalanceInquiry() {
        double balance = accountDAO.getAccountBalance(loggedInAccount.getAccountNumber());
        JOptionPane.showMessageDialog(this, "Current balance: $" + String.format("%.2f",
balance));
    }

    private void showTransactionHistory() {
        List<Transaction> transactions =
transactionDAO.getAllTransactions(loggedInAccount.getAccountNumber());

```

// Check if there are no transactions

```

if (transactions.isEmpty()) {
    JOptionPane.showMessageDialog(this, "No transactions found.");
    return;
}

StringBuilder transactionDetails = new StringBuilder("Transaction History:\n");

/* // Add initial balance transaction first
transactionDetails.append("Initial Balance of $")
    .append(String.format("%.2f", loggedInAccount.getBalance()))
    .append(" at Account Creation\n");*/
// Append other transactions
for (Transaction transaction : transactions) {

transactionDetails.append(transaction.getTransactionType())
    .append(" of $").append(String.format("%.2f", transaction.getAmount()))
    .append(" at ").append(transaction.getTimestamp())
    .append("\n");
}
JOptionPane.showMessageDialog(this, transactionDetails.toString());
}

private void logout() {
    JOptionPane.showMessageDialog(this, "Logout successful.");
    // Clear logged in account and show main menu again
    loggedInAccount = null;
    getContentPane().removeAll();
    add(mainMenuPanel, BorderLayout.CENTER);
    revalidate();
    repaint();
}

private void exitApplication() {
    JOptionPane.showMessageDialog(this, "Thank you for using our Banking System!");
    System.exit(0);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {

```



```

        new BankingSystem();
    }
    });
}
}

```

AccountDAO.java

```

import java.util.ArrayList;
import java.util.List;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

```

```

public class AccountDAO {
    private static final String URL = "jdbc:mysql://localhost:3306/banking_system";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "1234567890";

    public boolean createAccount(AccountDetails account) {
        String query = "INSERT INTO accounts (account_number, account_holder_name, balance, password) VALUES (?, ?, ?, ?)";

        try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            PreparedStatement statement = conn.prepareStatement(query)) {

            statement.setString(1, account.getAccountNumber());
            statement.setString(2, account.getAccountHolderName());
            statement.setDouble(3, account.getBalance());
            statement.setString(4, account.getPassword());

            int rowsInserted = statement.executeUpdate();
            return rowsInserted > 0;
        } catch (SQLException ex) {
            ex.printStackTrace();
            return false;
        }
    }
}

```

```

public AccountDetails getAccount(String accountNumber) {
    String query = "SELECT * FROM accounts WHERE account_number = ?";
    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {

        statement.setString(1, accountNumber);
        try (ResultSet resultSet = statement.executeQuery()) {
            if (resultSet.next()) {
                return new AccountDetails(
                    resultSet.getString("account_number"),
                    resultSet.getString("account_holder_name"),
                    resultSet.getDouble("balance"),
                    resultSet.getString("password")
                );
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return null;
}

public boolean updateAccount(AccountDetails account) {
    String query = "UPDATE accounts SET account_holder_name=?, balance=?, password=?
WHERE account_number=?";

    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {

        statement.setString(1, account.getAccountHolderName());
        statement.setDouble(2, account.getBalance());
        statement.setString(3, account.getPassword());
        statement.setString(4, account.getAccountNumber());
        int rowsUpdated = statement.executeUpdate();
        return rowsUpdated > 0;
    }
}

```

```

    } catch (SQLException ex) {
        ex.printStackTrace();
        return false;
    }
}

```

```

public boolean deleteAccount(String accountNumber) {
    String query = "DELETE FROM accounts WHERE account_number=?";
    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {
        statement.setString(1, accountNumber);
        int rowsDeleted = statement.executeUpdate();
        return rowsDeleted > 0;

    } catch (SQLException ex) {
        ex.printStackTrace();
        return false;
    }
}

```

```

public double getAccountBalance(String accountNumber) {
    String query = "SELECT balance FROM accounts WHERE account_number = ?";
    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {

        statement.setString(1, accountNumber);
        try (ResultSet resultSet = statement.executeQuery()) {
            if (resultSet.next()) {
                return resultSet.getDouble("balance");
            }
        }
    }
    catch (SQLException ex) {
        ex.printStackTrace();
    }
    return 0; // or throw an exception if the account is not found
}

```

```

public List<AccountDetails> getAccountsAboveThreshold(double threshold) {
    List<AccountDetails> accountsAboveThreshold = new ArrayList<>();
    String query = "SELECT * FROM accounts WHERE balance > ?";
    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {
        statement.setDouble(1, threshold);
        try (ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {

                AccountDetails account = new AccountDetails(
                    resultSet.getString("account_number"),
                    resultSet.getString("account_holder_name"),
                    resultSet.getDouble("balance"),
                    resultSet.getString("password")
                );
                accountsAboveThreshold.add(account);
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return accountsAboveThreshold;
}

public double getTotalBalance() {
    String query = "SELECT SUM(balance) AS total_balance FROM accounts";
    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query);
        ResultSet resultSet = statement.executeQuery()) {
        if (resultSet.next()) {
            return resultSet.getDouble("total_balance");
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return 0; // or throw an exception if unable to retrieve total balance
}
}

```

TransactionDAO.java

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class TransactionDAO {
    private AccountDAO accountDAO;
    public TransactionDAO(AccountDAO accountDAO) {
        this.accountDAO = accountDAO;
    }
    String getCurrentTimestamp() {
        LocalDateTime currentDateTime = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        return currentDateTime.format(formatter);
    }
    private static final String URL = "jdbc:mysql://localhost:3306/banking_system";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "1234567890";

    public boolean addTransaction(Transaction transaction, String accountNumber) {
        String query = "INSERT INTO transactions (account_number, transaction_type, amount, timestamp) VALUES (?, ?, ?, ?)";

        try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            PreparedStatement statement = conn.prepareStatement(query)) {

            statement.setString(1, accountNumber);
            statement.setString(2, transaction.getTransactionType());
            statement.setDouble(3, transaction.getAmount());
            statement.setString(4, transaction.getTimestamp());
            int rowsInserted = statement.executeUpdate();
            return rowsInserted > 0;
        }
    }
}
```

```

    } catch (SQLException ex) {
        ex.printStackTrace();
        return false;
    }
}

public List<Transaction> getAllTransactions(String accountNumber) {
    List<Transaction> transactions = new ArrayList<>();
    String query = "SELECT * FROM transactions WHERE account_number = ?";

    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {
        statement.setString(1, accountNumber);
        ResultSet resultSet = statement.executeQuery();

        while (resultSet.next()) {
            String transactionType = resultSet.getString("transaction_type");
            double amount = resultSet.getDouble("amount");
            String timestamp = resultSet.getString("timestamp");
            Transaction transaction = new Transaction(transactionType, amount, timestamp);
            transactions.add(transaction);
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    return transactions;
}

public boolean updateTransaction(Transaction transaction, String accountNumber) {
    String query = "UPDATE transactions SET transaction_type = ?, amount = ?, timestamp = ?
WHERE account_number = ?";

    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {
statement.setString(1, transaction.getTransactionType());
        statement.setDouble(2, transaction.getAmount());
        statement.setString(3, transaction.getTimestamp());
        statement.setString(4, accountNumber);
    }
}

```

```

        int rowsUpdated = statement.executeUpdate();
        return rowsUpdated > 0;

    } catch (SQLException ex) {
        ex.printStackTrace();
        return false;
    }
}

public boolean deleteTransactionsByAccount(String accountNumber) {

    String query = "DELETE FROM transactions WHERE account_number=?";
    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {

        statement.setString(1, accountNumber);

        int rowsDeleted = statement.executeUpdate();
        return rowsDeleted > 0;

    } catch (SQLException ex) {
        ex.printStackTrace();
        return false;
    }
}

public boolean deposit(double amount, String accountNumber) {
    String query = "UPDATE accounts SET balance = balance + ? WHERE account_number = ?";
    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement statement = conn.prepareStatement(query)) {

        statement.setDouble(1, amount);
        statement.setString(2, accountNumber);

        int rowsUpdated = statement.executeUpdate();
        if (rowsUpdated > 0) {
            // Create deposit transaction record

```

```
return addTransaction(new Transaction("Deposit", amount, getCurrentTimestamp()),
accountNumber);
```

```
}
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return false;
}
```

```
public boolean withdraw(double amount, String accountNumber) {
    // Check if the account has sufficient balance
    double currentBalance = accountDAO.getAccountBalance(accountNumber);

    if (currentBalance >= amount) {
        String query = "UPDATE accounts SET balance = balance - ? WHERE account_number
= ?";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            PreparedStatement statement = conn.prepareStatement(query)) {

            statement.setDouble(1, amount);
            statement.setString(2, accountNumber);

            int rowsUpdated = statement.executeUpdate();
            if (rowsUpdated > 0) {
                // Create withdrawal transaction record
                return addTransaction(new Transaction("Withdrawal", -amount,
getCurrentTimestamp()), accountNumber);
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
    return false;
}}
```


CHAPTER 4

Experimental Setup

4.1 Technologies Used

- **Programming Language:** Java 8
- **Frameworks:**
 - JDBC for database connectivity
 - Swing for desktop application
- **Database:** MySQL

4.2 System Hardware

- **Processor:** Intel Core i5 or higher
- **RAM:** 8GB or higher
- **Storage:** 100GB SSD or higher

4.3 Results Table or Screenshots



Login

Account Number:

Password:

OK Cancel

Input

Enter the account number to delete:

OK Cancel

Message

Login successful. Welcome Pavani

OK

Banking System

Account Services

Deposit

Withdraw

Balance Inquiry

Transaction History

Logout

Input

Enter withdrawal amount:

OK Cancel

Input ✕




Enter deposit amount:

Message ✕




Current balance: \$150.00

Message ✕



Transaction History:
Deposit of \$100.00 at 2024-06-16 14:17:39
Withdrawal of \$-50.00 at 2024-06-16 14:17:47

Message ✕



Logout successful.

Message ✕



Thank you for using our Banking System!

Data stored in Database:

```
mysql> use banking_system;
Database changed
mysql> show tables;
+-----+
| Tables_in_banking_system |
+-----+
| accounts                  |
| transactions              |
+-----+
2 rows in set (0.14 sec)

mysql> select * from accounts;
+-----+-----+-----+-----+
| account_number | account_holder_name | balance | password |
+-----+-----+-----+-----+
| 11             | Pavani              | 100     | 11        |
| 123            | 1234                | 10000   | 12345    |
| 12345          | 123                 | 8000    | 12345    |
| 15             | Pavan               | 150     | 15        |
+-----+-----+-----+-----+
4 rows in set (0.17 sec)
```

```
mysql> select * from transactions;
+-----+-----+-----+-----+-----+
| id | account_number | transaction_type | amount | timestamp          |
+-----+-----+-----+-----+-----+
| 1  | 11             | Deposit         | 100    | 2024-06-19 13:23:50 |
| 2  | 11             | Deposit         | 100    | 2024-06-19 13:24:06 |
| 3  | 11             | Withdrawal      | -100   | 2024-06-19 13:25:38 |
| 4  | 12345          | Deposit         | 10000  | 2024-06-19 13:44:13 |
| 5  | 12345          | Deposit         | 10000  | 2024-06-19 13:44:43 |
| 6  | 12345          | Withdrawal      | -12000 | 2024-06-19 13:44:53 |
| 7  | 123            | Deposit         | 10000  | 2024-06-19 13:45:48 |
| 8  | 15             | Deposit         | 100    | 2024-06-19 13:55:42 |
| 9  | 15             | Deposit         | 100    | 2024-06-19 13:56:12 |
| 10 | 15             | Withdrawal      | -50    | 2024-06-19 13:56:23 |
| 11 | 15             | Deposit         | 300    | 2024-06-19 13:58:59 |
| 12 | 15             | Withdrawal      | -300   | 2024-06-19 13:59:10 |
+-----+-----+-----+-----+-----+
12 rows in set (1.17 sec)
```

3.4. Results Comparison with Other Approaches:

Comparing results with other approaches, our banking system shows robustness in transaction handling and account management, supporting secure operations and real-time updates.

CHAPTER 5

Conclusion

- 1.The banking system demonstrates robust functionality with secure account management, efficient transaction handling, and clear user interfaces.
- 2.User feedback highlights positive experiences with intuitive navigation and reliable transaction processing, enhancing overall satisfaction.
- 3.Future improvements could focus on scalability and optimization to meet growing user demands and further enhance system performance.

5.2 Key Achievements

Secure Account Management:

- Implementation of password validation and secure storage for account details.
- Ensuring transactional security through synchronized methods and locks (depositLock and withdrawLock).

Transaction Handling:

- Facilitation of deposit and withdrawal transactions with validation checks (amount, balance).
- Integration of transaction logging (Transaction class) for audit trails (TransactionDAO).

Transaction History and Balance Inquiry:

- Display of transaction history (showTransactionHistory) including transaction type, amount, and timestamp.
- Real-time balance inquiry (showBalanceInquiry) to retrieve current account balance.

- **Data Access Object (DAO) Pattern:**

- Use of DAO pattern (AccountDAO and TransactionDAO) for encapsulating data access and database operations.
- Ensuring data integrity and separation of concerns between business logic and data persistence.

User Interface (UI) Design:

- Creation of intuitive GUI using Swing components (JFrame, JPanel, JButton, JOptionPane).
- Organized menu navigation (mainMenuPanel and banking operation panels) for user-friendly interaction.

Error Handling and Validation:

- Robust input validation (`createAccount`, `performDeposit`, `performWithdrawal`) to prevent invalid data entry.
- Error messages (`JOptionPane.showMessageDialog`) for notifying users of input errors or transaction failures.

Account Deletion:

- Secure deletion of accounts (`deleteAccount`) including associated transaction records (`deleteTransactionsByAccount`).
- Confirmation messages (`JOptionPane.showMessageDialog`) upon successful deletion of accounts and transactions.

5.2 Future Scope/Directions

Enhanced Security: Implementing advanced encryption and biometric authentication.

AI and Machine Learning: Utilizing for personalized services and fraud detection.

Mobile and Contactless Banking: Developing intuitive apps for seamless transactions.

Blockchain Integration: Exploring for transparent and secure transaction handling.

5.3 References

- [Java Collections Framework Documentation.](#)
- [JDBC API Guide.](#)
- [Swing API Guide.](#)
- [SQL Documentation.](#)