Software Construction and Development

Final Project Report

Submitted to:

Mam. Marriam Daud

Mam. Fatima Gillani

Submitted by:

Muhammad Taha (FL-21154)

Saif Ullah (FL-21168)

Muhammad Talha (FL-21166)

Jawad Akbar (FL-21176)

Muhammad Shaheer (FL-21180)

BSSE-30A-Morning

Date: December 21, 2023

Table of Contents

Banking App

Abstract:

This project encapsulates the development of a feature-rich **Java banking application** with a comprehensive suite of functionalities, including **account creation**, **deposit**, **withdrawal**, **balance checking**, and a detailed **display of all accounts**. An integral aspect of this development is the incorporation of robust exception handling mechanisms and a meticulously crafted set of test cases to ensure the application's reliability and performance.

Exception handling plays a crucial role in fortifying the application against unforeseen circumstances and errors. Through strategic implementation of **try-catch blocks**, the application can gracefully handle exceptions, maintaining data integrity and user experience. Exception handling extends beyond mere error detection; it serves as a mechanism to communicate meaningful feedback to users and administrators, fostering a smoother interaction with the banking system.

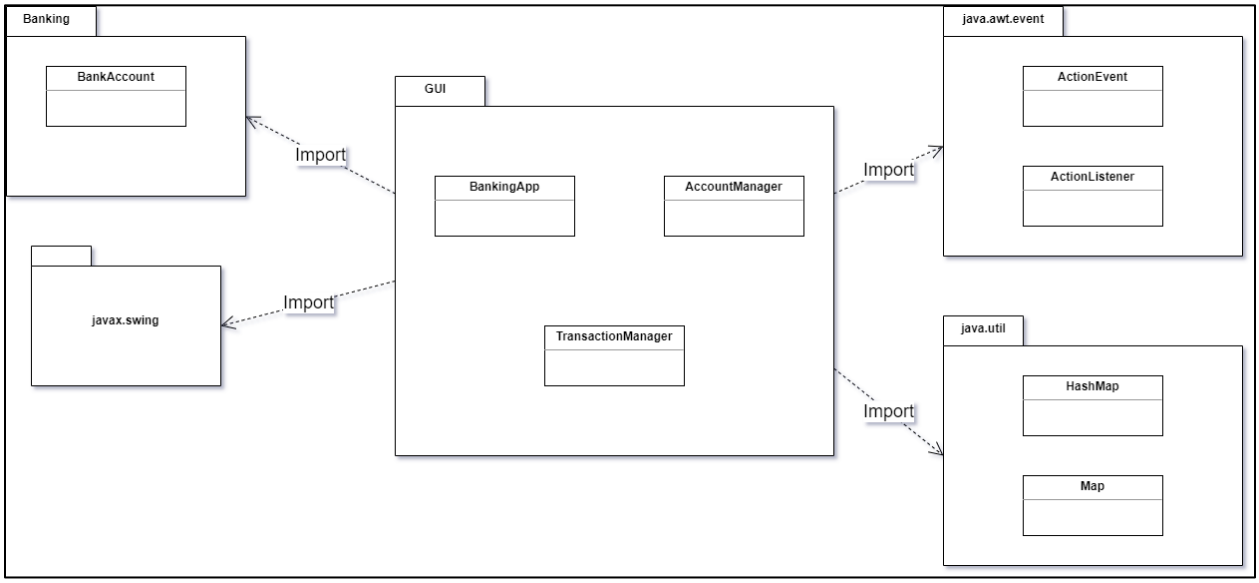
Furthermore, a comprehensive set of **test cases** has been meticulously designed and executed, with a particular focus on the deposit and withdrawal functionalities. These test cases rigorously evaluate the system's behavior under various scenarios, including normal operation, edge cases, and error conditions. The test suite not only validates the correctness of the implemented features but also ensures that the application performs optimally and consistently across a spectrum of usage scenarios.

To foster collaboration and transparency, the entire project, including the source code, documentation, and test cases, has been published on **GitHub**. This platform serves as a central repository for version control, facilitating collaborative development and allowing for community contributions. The open-source nature of the project encourages peer review and provides a foundation for future enhancements, bug fixes, and collaborative improvements.

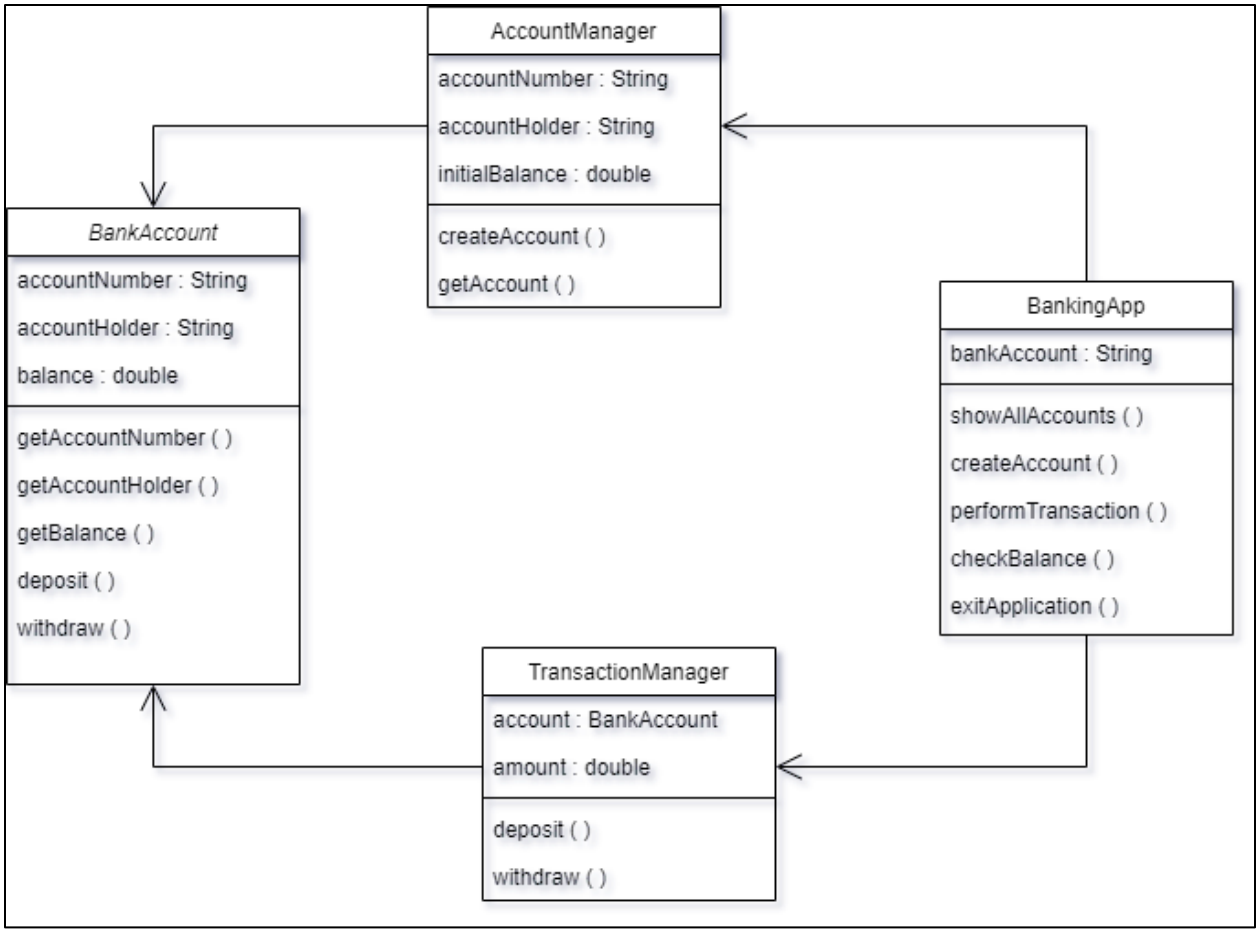
In conclusion, this project represents a holistic approach to banking application development, encompassing robust functionalities, meticulous exception handling, and a comprehensive set of test cases. The integration of these elements not only enhances the application's reliability and user experience but also establishes a foundation for ongoing collaboration and improvement within the development community, exemplified by the project's presence on GitHub.

UML Diagrams:

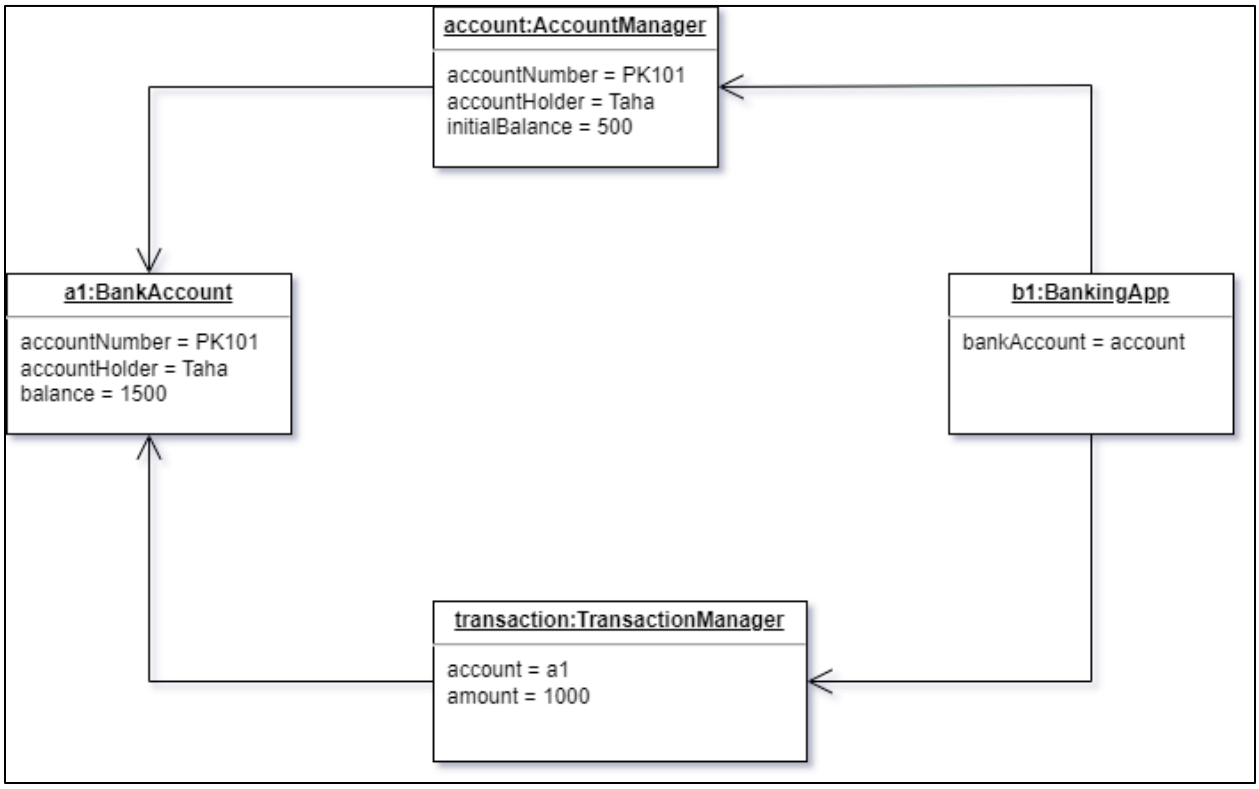
1. Package Diagram:



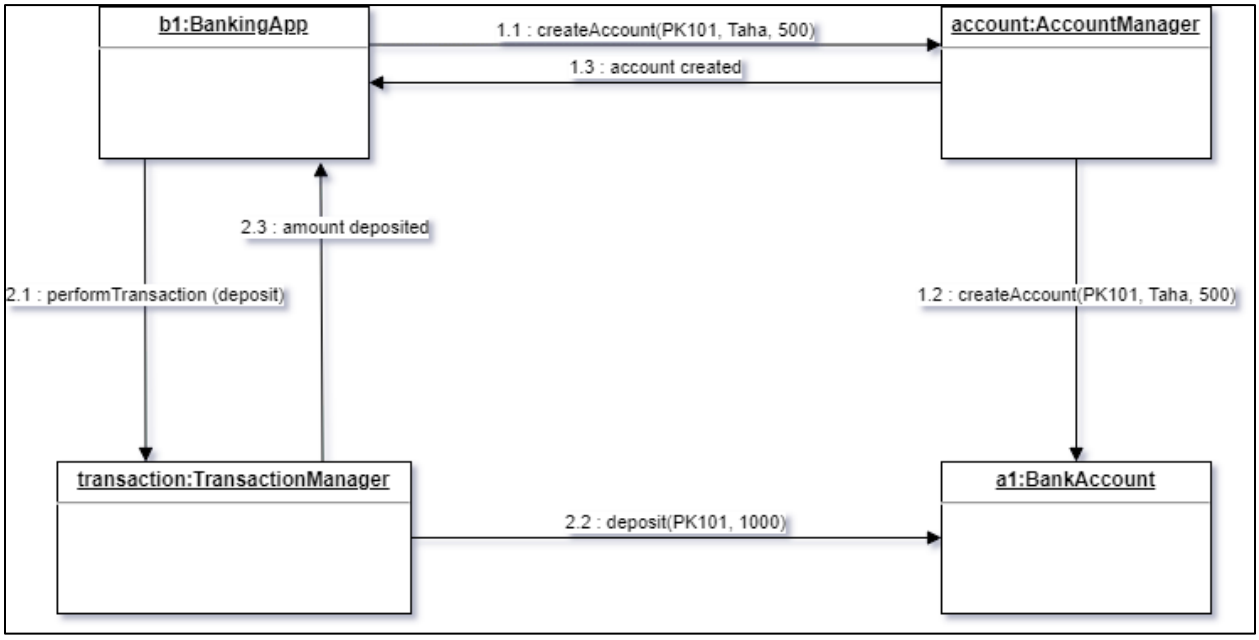
2. Class Diagram:



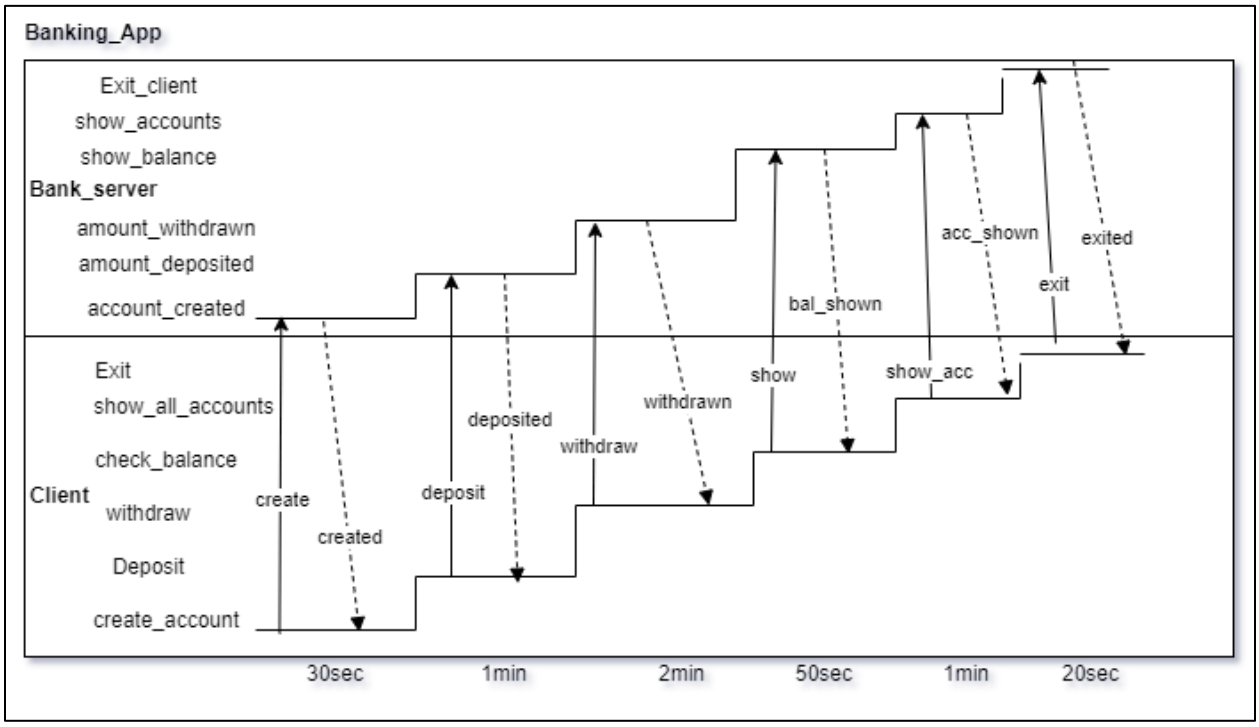
3. Object Diagram:



4. Communication Diagram:



5. Timing Diagram:



Implementation:

Package: gui

BankingApp.java

```
package gui;

import banking.BankAccount;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.util.Map;

public class BankingApp extends JFrame {
    private static Map<String, BankAccount> accounts = new HashMap<>();

    public BankingApp() {
        setTitle("Banking App");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Center the frame on the screen
        setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setBackground(new Color(173, 216, 230)); // Light Blue color, you
can change it to your desired bank-themed color
        add(panel);
        placeComponents(panel);

        setVisible(true);
    }

    private void placeComponents(JPanel panel) {
        panel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5);

        JLabel titleLabel = new JLabel("Banking App");
        titleLabel.setFont(new Font("Arial", Font.BOLD, 20));
        gbc.gridx = 0;
        gbc.gridy = 0;
        panel.add(titleLabel, gbc);
```

```
        JButton createAccountButton = new JButton("Create Account");
        createAccountButton.setPreferredSize(new Dimension(150, 25));
        gbc.gridx = 0;
        gbc.gridy = 1;
        panel.add(createAccountButton, gbc);

        JButton depositButton = new JButton("Deposit");
        depositButton.setPreferredSize(new Dimension(150, 25));
        gbc.gridx = 0;
        gbc.gridy = 2;
        panel.add(depositButton, gbc);

        JButton withdrawButton = new JButton("Withdraw");
        withdrawButton.setPreferredSize(new Dimension(150, 25));
        gbc.gridx = 0;
        gbc.gridy = 3;
        panel.add(withdrawButton, gbc);

        JButton checkBalanceButton = new JButton("Check Balance");
        checkBalanceButton.setPreferredSize(new Dimension(150, 25));
        gbc.gridx = 0;
        gbc.gridy = 4;
        panel.add(checkBalanceButton, gbc);

        JButton showAllAccountsButton = new JButton("Show All Accounts");
        showAllAccountsButton.setPreferredSize(new Dimension(150, 25));
        gbc.gridx = 0;
        gbc.gridy = 5;
        panel.add(showAllAccountsButton, gbc);

        JButton exitButton = new JButton("Exit");
        exitButton.setPreferredSize(new Dimension(150, 25));
        gbc.gridx = 0;
        gbc.gridy = 6;
        panel.add(exitButton, gbc);

        createAccountButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createAccount();
            }
        });

        depositButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
                performTransaction("deposit");
            }
        });

        withdrawButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                performTransaction("withdraw");
            }
        });

        checkBalanceButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                checkBalance();
            }
        });

        showAllAccountsButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showAllAccounts();
            }
        });

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

    private void showAllAccounts() {
        if (AccountManager.getAllAccounts().isEmpty()) {
            JOptionPane.showMessageDialog(this, "No accounts found.");
        } else {
            StringBuilder allAccountsInfo = new StringBuilder("All Accounts:\n");
            for (Map.Entry<String, BankAccount> entry :
AccountManager.getAllAccounts().entrySet()) {
                allAccountsInfo.append("Account Number:
").append(entry.getKey()).append("\n");
                allAccountsInfo.append(entry.getValue()).append("\n----------
-\n");
            }
            JOptionPane.showMessageDialog(this, allAccountsInfo.toString());
        }
    }
```

```
    private void createAccount() {
        try {
            String accountNumber = validateAccountNumber();
            if (accountNumber != null) {
                if (AccountManager.getAccount(accountNumber) != null) {
                    showErrorDialog("Account already exists.");
                } else {
                    String accountHolder = validateAccountHolder();
                    String initialBalanceStr = validateInitialBalance();
                    if (accountHolder != null && initialBalanceStr != null) {
                        double initialBalance =
Double.parseDouble(initialBalanceStr);
                        if (initialBalance >= 0) {
                            if (AccountManager.createAccount(accountNumber,
accountHolder, initialBalance)) {
                                showSuccessDialog("Account created
successfully.");
                            } else {
                                showErrorDialog("Failed to create account.");
                            }
                        } else {
                            showErrorDialog("Initial balance should be greater
than or equal to zero.");
                        }
                    }
                }
            }
        } catch (NumberFormatException ex) {
            showErrorDialog("Invalid initial balance. Please enter a valid
number.");
        }
    }

private String validateAccountNumber() {
    String accountNumber;
    do {
        accountNumber = JOptionPane.showInputDialog(this, "Enter account
number:");
        if (accountNumber == null) {
            return null;  // User clicked Cancel
        }
        if (!accountNumber.isEmpty() && accountNumber.matches("[a-zA-Z0-9]+")) {
            return accountNumber;
        } else {
```

```
            showErrorDialog("Invalid account number. Please enter a non-empty
string containing letters or digits.");
        }
    } while (true);
}


    private String validateAccountHolder() {
    String accountHolder;
    do {
        accountHolder = JOptionPane.showInputDialog(this, "Enter account holder
name:");
        if (accountHolder == null) {
            return null;  // User clicked Cancel
        }
        if (accountHolder.matches("[a-zA-Z]+")) {
            return accountHolder;
        } else {
            showErrorDialog("Invalid account holder name. Please enter a non-
empty string containing only alphabets.");
        }
    } while (true);
}

private String validateInitialBalance() {
    do {
        String initialBalanceStr = JOptionPane.showInputDialog(this, "Enter
initial balance:");
        if (initialBalanceStr == null) {
            return null;  // User clicked Cancel
        }
        try {
            double initialBalance = Double.parseDouble(initialBalanceStr);
            if (initialBalance >= 100 && initialBalance <= 1000000) {
                return initialBalanceStr;
            } else {
                showErrorDialog("Initial balance should be between $100 and
$1,000,000.");
            }
        } catch (NumberFormatException ex) {
            showErrorDialog("Invalid initial balance. Please enter a valid
number.");
        }
    } while (true);
}
```

```
private void performTransaction(String transactionType) {
    try {
        String accountNumber = validateAccountNumber();
        if (accountNumber != null) {
            BankAccount account = AccountManager.getAccount(accountNumber);
            if (account != null) {
                String amountStr = JOptionPane.showInputDialog(this, "Enter
amount to " + transactionType + ":");
                if (amountStr == null) {
                    return;  // User clicked Cancel
                }
                double amount = Double.parseDouble(amountStr);
                if (amount <= 0) {
                    showErrorDialog("Invalid amount. Please enter a positive
value.");
                } else {
                    if (transactionType.equals("deposit")) {
                        TransactionManager.deposit(account, amount);
                    } else {
                        TransactionManager.withdraw(account, amount);
                    }
                }
            } else {
                showErrorDialog("Account not found.");
            }
        }
    } catch (NumberFormatException ex) {
        showErrorDialog("Invalid amount. Please enter a valid number.");
    }
}

    private void showErrorDialog(String message) {
        JOptionPane.showMessageDialog(this, message, "Error",
JOptionPane.ERROR_MESSAGE);
    }

    private void showSuccessDialog(String message) {
        JOptionPane.showMessageDialog(this, message, "Success",
JOptionPane.INFORMATION_MESSAGE);
    }

    private void checkBalance() {
        String accountNumber = validateAccountNumber();
        if (accountNumber != null) {
```

```
            BankAccount account = AccountManager.getAccount(accountNumber);
            if (account != null) {
                JOptionPane.showMessageDialog(this,
                        "Account Holder: " + account.getAccountHolder() +
                                "\nAccount Balance: $" + account.getBalance());
            } else {
                showErrorDialog("Account not found.");
            }
        }
    }

    private void exitApplication() {
        System.out.println("Exiting the banking app. Goodbye!");
        System.exit(0);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new BankingApp();
            }
        });
    }
}
```

AccountManager.java

```
package gui;

import banking.BankAccount;

import java.util.HashMap;
import java.util.Map;

public class AccountManager {
    private static Map<String, BankAccount> accounts = new HashMap<>();

    public static boolean createAccount(String accountNumber, String
accountHolder, double initialBalance) {
        if (!accounts.containsKey(accountNumber)) {
            BankAccount account = new BankAccount(accountNumber, accountHolder,
initialBalance);
            accounts.put(accountNumber, account);
            return true;
        }
```

```
        return false;
    }

    public static BankAccount getAccount(String accountNumber) {
        return accounts.get(accountNumber);
    }

    public static Map<String, BankAccount> getAllAccounts() {
        return accounts;
    }
}
```

TransactionManager.java

```
package gui;

import banking.BankAccount;

public class TransactionManager {
    public static void deposit(BankAccount account, double amount) {
        account.deposit(amount);
    }

    public static void withdraw(BankAccount account, double amount) {
        account.withdraw(amount);
    }
}
```

Package: banking

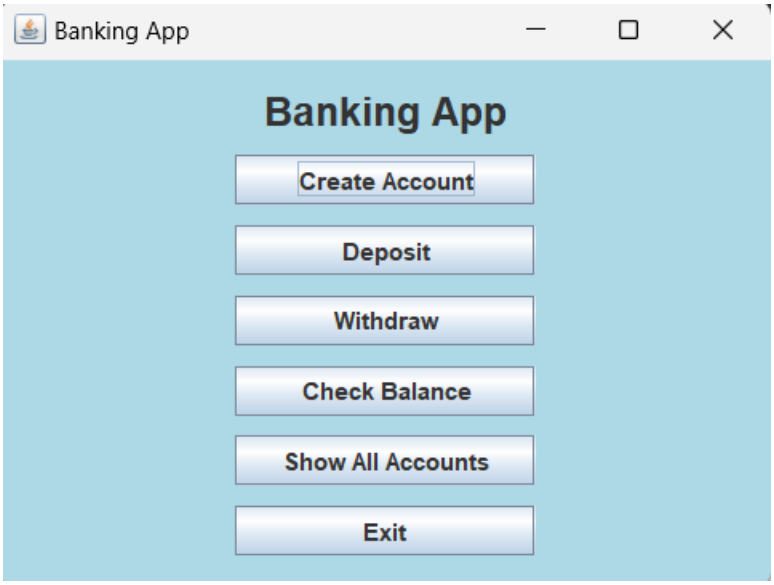
BankAccount.java

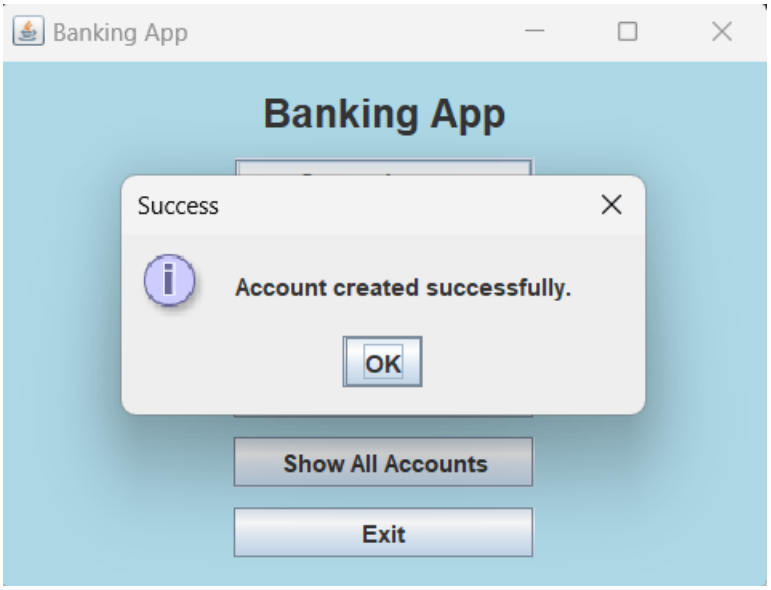
```
package banking;

import javax.swing.JOptionPane;

public class BankAccount {
    private String accountNumber;
    private String accountHolder;
    private double balance;
    private static final double MAX_DEPOSIT_AMOUNT = 1000000;

    public BankAccount(String accountNumber, String accountHolder, double
initialBalance) {
```

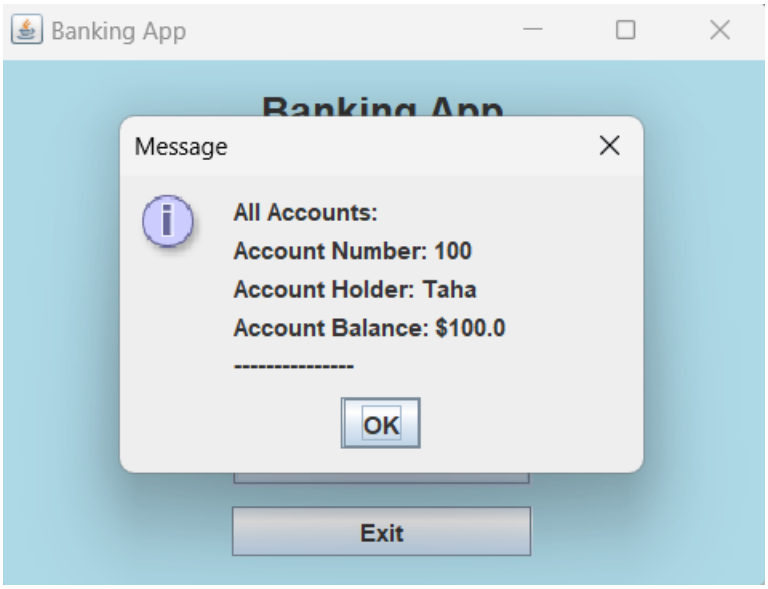
```
        this.accountNumber = accountNumber;
        this.accountHolder = accountHolder;
        this.balance = initialBalance;
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public String getAccountHolder() {
        return accountHolder;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if (amount <= 0) {
            showErrorDialog("Invalid deposit amount. Please enter a positive
value.");
            return;
        }

        if (amount + balance > MAX_DEPOSIT_AMOUNT) {
            showErrorDialog("Deposit amount exceeds the maximum limit of
$1,000,000. Deposit failed.");
            return;
        }

        balance += amount;
        showSuccessDialog("Deposit successful. Current balance: $" + balance);
    }

    public void withdraw(double amount) {
        if (amount <= 0) {
            showErrorDialog("Invalid withdrawal amount. Please enter a positive
value.");
            return;
        }
        if (amount > balance) {
            showErrorDialog("Insufficient funds. Withdrawal failed.");
        } else {
            balance -= amount;
```

```
            showSuccessDialog("Withdrawal successful. Current balance: $" +
balance);
        }
    }

    @Override
    public String toString() {
        return "Account Holder: " + accountHolder + "\nAccount Balance: $" +
balance;
    }

    private void showErrorDialog(String message) {
        JOptionPane.showMessageDialog(null, message, "Error",
JOptionPane.ERROR_MESSAGE);
    }

    private void showSuccessDialog(String message) {
        JOptionPane.showMessageDialog(null, message, "Success",
JOptionPane.INFORMATION_MESSAGE);
    }
}
```

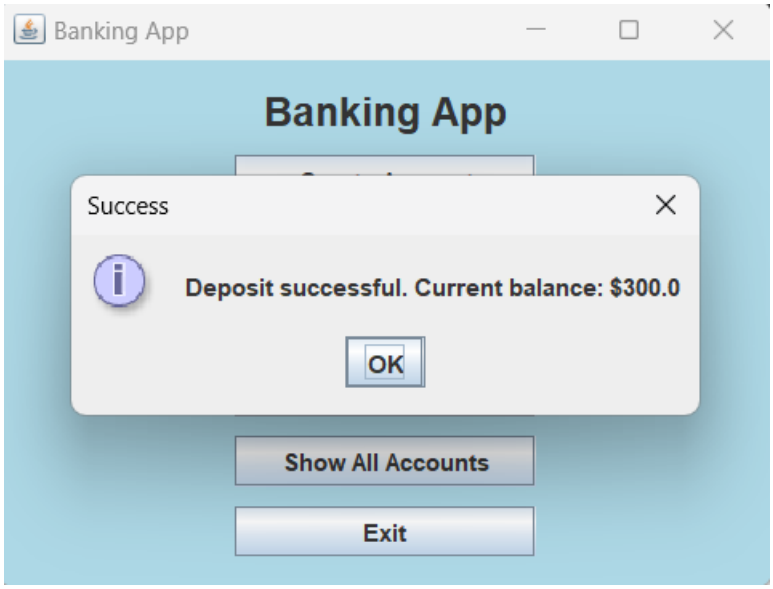
Output:

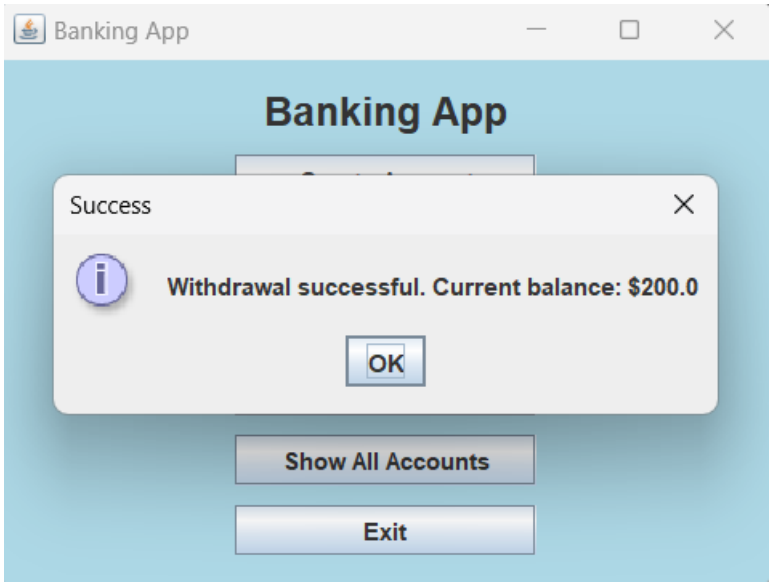



Banking App
Banking App
Success
Account created successfully.
OK
Show All Accounts
Exit


Banking App
Message
All Accounts:
Account Number: 100
Account Holder: Taha
Account Balance: $100.0

OK
Exit


Banking App
Banking App
Success
Deposit successful. Current balance: $300.0
OK
Show All Accounts
Exit


Banking App
Banking App
Success
Withdrawal successful. Current balance: $200.0
OK
Show All Accounts
Exit

Exception Handling:

The provided Java banking application demonstrates a thoughtful approach to exception handling, ensuring a robust and user-friendly experience. Exception handling is critical for gracefully managing unexpected situations and errors that may arise during the execution of the program. The primary focus of the exception handling in this application is on user input validation and transaction processing. Here are some key aspects of exception handling within the code:

Input Validation:

The application employs input validation methods, such as validateAccountNumber, validateAccountHolder, and validateInitialBalance, to ensure that user inputs adhere to specified criteria.

If the user provides invalid input, an error dialog is displayed, providing clear feedback on the nature of the error and guiding the user toward rectifying it.

NumberFormatException Handling:

The application handles NumberFormatException gracefully, particularly when parsing input strings to numeric values.

For instance, when parsing initial balance or transaction amounts, the code catches NumberFormatException and displays an error dialog prompting the user to enter a valid numeric value.

Transaction Processing:

The performTransaction method ensures that the entered amount for deposit or withdrawal is a valid positive numeric value. In case of invalid amounts, appropriate error dialogs are displayed.

Overall, the inclusion of these exception handling mechanisms enhances the application's resilience by providing informative feedback to users and preventing unintended program disruptions. It reflects a proactive approach to user interactions, contributing to the overall reliability and usability of the banking application.

Unit Test:

Test case 1: Deposit Money (Valid)	
Test Case ID	1
Input	**Current Balance:** 100 **New Deposit:** 200
Partition tested	Valid Class
Expected Output	Deposit successful. Current balance: $300.0
Actual Output	Deposit successful. Current balance: $300.0
Pass/Fail	Pass

Test case 2: Deposit Money(Invalid)	
Test Case ID	2
Input	**Current Balance:** 10000 **New Deposit:** 9999999
Partition tested	Invalid Class
Expected Output	Deposit amount exceeds the maximum limit of $1,000,000. Deposit failed.
Actual Output	Deposit amount exceeds the maximum limit of $1,000,000. Deposit failed.
Pass/Fail	Pass

Test case 3: Deposit Money (Invalid)	
Test Case ID	3
Input	**Current Balance:** 100 **New Deposit:** -1
Partition tested	Invalid Class
Expected Output	Invalid deposit amount. Please enter a positive value.
Actual Output	Invalid deposit amount. Please enter a positive value.
Pass/Fail	Pass

Test case 4: Withdraw Money (Valid)	
Test Case ID	4
Input	**Current Balance:** 100 **Withdraw Amount:** 50
Partition tested	Valid Class
Expected Output	Withdrawal successful. Current balance: $50.0"
Actual Output	Withdrawal successful. Current balance: $50.0"
Pass/Fail	Pass

Test case 5: Withdraw Money (Invalid)	
Test Case ID	5
Input	**Current Balance:** 100 **Withdraw Amount:** 200
Partition tested	Invalid Class
Expected Output	Insufficient funds. Withdrawal failed.
Actual Output	Insufficient funds. Withdrawal failed.
Pass/Fail	Pass

Test case 6: Withdraw Money (Invalid)	
Test Case ID	6
Input	**Current Balance:** 100 **Withdraw Amount:** -1
Partition tested	Invalid Class
Expected Output	Invalid withdrawal amount. Please enter a positive value.
Actual Output	Invalid withdrawal amount. Please enter a positive value.
Pass/Fail	Pass

Pushing to GitHub:

The source code for the Java banking application has been successfully uploaded to GitHub and is now accessible at the following URL: https://github.com/raja-taha/SCD-Final-Project. This GitHub repository serves as a centralized platform for version control, collaboration, and transparent management of the project's development. Developers, contributors, and stakeholders can use this URL to explore the codebase, propose changes, report issues, and actively participate in the ongoing evolution of the Java banking application.

Conclusion:

In conclusion, the development of the Java banking application represents a significant milestone in creating a user-friendly, secure, and feature-rich solution for modern banking needs. The project's meticulous design, incorporating functionalities such as account creation, deposits, withdrawals, and balance checks, is aimed at providing a comprehensive banking experience. The integration of robust exception handling mechanisms ensures the application's resilience to unexpected scenarios, enhancing user experience and system reliability. Furthermore, the implementation of a suite of test cases, particularly for deposit and withdrawal functionalities, attests to the commitment to delivering a thoroughly validated and dependable banking system.

The decision to publish the project on GitHub marks a strategic move towards collaboration and transparency within the development community. By sharing the codebase at https://github.com/raja-taha/SCD-Final-Project, the project invites contributions, feedback, and engagement from a wider audience. This open-source approach not only fosters innovation but also ensures the adaptability and sustainability of the banking application, making it a valuable asset for continuous improvement and future developments in the realm of digital financial services.