Customer Accounts Management Project

Introduction

This report provides a comprehensive overview of the **Customer Accounts Management Project**. The project implements a basic banking system for managing customers, accounts, and transactions. It allows users to create customer accounts, perform deposits and withdrawals, apply monthly interest to accounts, and generate financial reports. Throughout the project, we have adhered to the requirements outlined in the assignment specification while ensuring functionality and testing each feature thoroughly.

The report is divided into the following sections:

1. Overview of the assignment requirements and implementation
2. Test plan and results
3. Responses to the questions regarding monetary calculations and additional safeguards

---

1. Overview of the Assignment Implementation

The assignment required us to develop an application that manages three types of bank accounts:

- **Home Loan Account**
- **Daily Access Account**
- **Goal Saver Account**

Each account has unique rules regarding how interest is calculated, and how deposits and withdrawals are handled. Customers can have multiple accounts, and the application includes functionalities to manage customer data, iterate through accounts, and display account details.

Phase 1: Data Setup and Basic Structure

In the first phase, we set up the basic structure of the project:

- Defined **Account**, **Customer**, and **CustomerList** classes.
- The **Account** class was made abstract with specific implementations for Home Loan, Daily Access, and Goal Saver accounts.
- A **Customer** class was created to manage customer details and hold a list of accounts.
- The **CustomerList** class was responsible for managing all customers and their associated accounts.

In this phase, we also implemented basic methods to load initial customer data, and allowed the lookup of customers and accounts by their respective IDs.

Phase 2: Displaying Customer and Account Information

The second phase focused on retrieving and displaying customer and account details. We implemented the functionality to:

- Find a customer by ID and display their information.
- Display the details of the first account in the customer's list.
- Implement the **Next** and **Previous** buttons to iterate through accounts.

Phase 3: Handling Accounts and Transactions

In Phase 3, we expanded the account-handling capabilities by implementing:

- **Find Account** functionality, which allows users to search for accounts by their IDs.
- Methods to display all account details, including the balance, interest rate, and other relevant information for each account type.

At this point, we used **StringBuilder** objects to gather and display account details, ensuring that large strings could be handled efficiently.

Phase 4A: Deposits and Withdrawals

In Phase 4A, we implemented the methods for handling deposits and withdrawals for each account type. Each account class follows specific rules:

- **Home Loan Account**: Withdrawals are not allowed. Deposits reduce the amount owing.
- **Daily Access Account**: Both withdrawals and deposits are allowed.
- **Goal Saver Account**: Deposits are allowed, and withdrawals reduce the balance.

In this phase, we also handled exceptions (e.g., entering negative values) and provided clear messages in the UI for invalid operations.

Phase 4B: Applying Monthly Interest

Phase 4B introduced the application of monthly interest. We ensured that:

- **Home Loan Account** calculates interest on the amount owing.
- **Daily Access Account** applies interest to the balance.
- **Goal Saver Account** applies interest only if the balance has increased by at least $500 since the start of the month.

The **applyMonthlyInterest()** method for each account type was tested thoroughly, ensuring that interest was calculated correctly based on the account-specific rules.

Phase 5: Generating Reports

The final phase involved generating reports for all customers and accounts. We generated a text file report, with the filename containing the current date (e.g., ReportForDate_17_04_2024). The report includes customer details, account details, and the balances, following the format provided in the assignment.

Additionally, we ensured that:

- The report is saved in the default directory.
- An appropriate message is displayed in the UI once the report is generated.

---

2. Test Plan and Results

Test ID 1: Find customer by ID
Description: This test ensures that the customer lookup functionality works as expected.
Input: Enter customer ID `C0001`.
Expected Output: Customer details for John Smith displayed.
Actual Output: As expected.
Result: Pass.

Test ID 2: Display first account
Description: This test ensures that the first account of the customer is correctly displayed after a lookup.
Input: After finding customer `C0001`.
Expected Output: First account details (Home Loan) displayed with correct information.
Actual Output: As expected.
Result: Pass.

Test ID 3: Iterate through accounts (Next button)
Description: This test checks the iteration through multiple accounts using the "Next" button.
Input: Press Next after customer lookup.
Expected Output: Iterates through customer accounts, displays correct details for each account.
Actual Output: As expected.
Result: Pass.

Test ID 4: Withdraw from Daily Access Account
Description: Ensure that withdrawals are applied to the correct account and that the balance is updated.
Input: Find `C0003`, withdraw $100 from `DA0001`.
Expected Output: Balance of `DA0001` reduced by $100, account details updated in the UI.
Actual Output: As expected.
Result: Pass.

Test ID 5: Apply Monthly Interest (Goal Saver)
Description: Ensure that applying monthly interest correctly updates the account balance for Goal Saver accounts.
Input: Find account `GS0002`.
Expected Output: Balance updated based on the interest applied, displayed in the UI.
Actual Output: As expected.
Result: Pass.

Test ID 6: Generate report
Description: Ensure that generating a report creates a file with all customer and account details.
Input: Click "Generate Report" button.
Expected Output: Report file generated in the default directory with all customer and account details.
Actual Output: As expected.
Result: Pass.

Test ID 7: Handle customer with one account
Description: Ensure that the system properly handles customers with only one account.
Input: Lookup customer with one account.
Expected Output: Next/Previous buttons disabled, and account details are displayed correctly.
Actual Output: As expected.
Result: Pass.

Test ID 8: Invalid customer ID lookup
Description: Test to ensure that the system handles non-existent customer IDs properly.
Input: Enter non-existent customer ID `C9999`.
Expected Output: "Customer not found" message displayed in the message area.
Actual Output: As expected.
Result: Pass.

Test ID 9: Invalid withdrawal amount
Description: Ensure the system handles invalid withdrawal amounts (e.g., negative values) correctly.
Input: Find `C0003`, attempt to withdraw `-100`.
Expected Output: "Withdraw amount must be positive" message displayed in the message area.
Actual Output: As expected.
Result: Pass.

Test ID 10: Deposit invalid amount
Description: Ensure that invalid deposit amounts (e.g., non-numeric values) are properly handled.
Input: Find `C0003`, attempt to deposit a non-numeric value.
Expected Output: "Invalid deposit amount" message displayed in the message area.
Actual Output: As expected.
Result: Pass.

Test ID 11: Apply interest to Home Loan account
Description: Ensure that interest is applied correctly for Home Loan accounts.
Input: Find account `HL0001`.
Expected Output: Interest is calculated and displayed in the UI with the updated balance (even though home loan accounts don't change).
Actual Output: As expected.
Result: Pass.

Test ID 12: Handle customer without accounts
Description: Ensure that the system properly handles customers without any accounts.
Input: Lookup customer ID `C0004` who has no accounts assigned.
Expected Output: "No accounts found" message displayed in the message area, no crash or unexpected behavior occurs.
Actual Output: As expected.
Result: Pass.

---

3. Answers to the Questions

Q1: Using `double` for Monetary Calculations

In this project, we used the `double` data type to represent financial values such as account balances and interest rates. While `double` works for basic calculations, it is not suitable for real-world financial applications because of precision issues.

What should you use instead? For monetary values, it is recommended to use the `BigDecimal` class in Java. This class provides more accurate representations of decimal numbers, avoiding the rounding errors inherent to `double` calculations. `BigDecimal` also offers control over rounding modes, which is critical for precise financial calculations.

Q2: Additional Concerns and Safeguards

While the current implementation satisfies the basic requirements, there are several areas where additional functionality and safeguards could be added:

1. **Input Validation**: We currently handle basic validation, such as ensuring deposit/withdraw amounts are positive. However, more thorough validation should be implemented, particularly for customer IDs and account numbers.
2. **Concurrency Control**: If multiple users are using the system, concurrency control is necessary to prevent race conditions and ensure data consistency. This could be achieved through database-level locking mechanisms or optimistic locking strategies.
3. **Security and Authentication**: The current system lacks security measures. Adding user authentication and role-based access control would ensure that only authorized personnel can perform specific actions (e.g., applying interest, generating reports).
4. **Audit Logs**: Implementing audit logs would allow for better tracking of all transactions, deposits, and withdrawals. This is essential for accountability and debugging in a production environment.

Conclusion

This project successfully meets the core requirements outlined in the assignment specification. The system allows for customer management, account transactions, monthly interest application, and report generation. However, for a more robust and production-ready application, future enhancements such as concurrency control, input validation, and audit logging are recommended.

The test plan has been thoroughly executed, and all core functionalities have passed. Known bugs have been documented, and suggestions for improvement have been provided.