Software Requirements Specification (SRS)

Bank Account Management System

Mahmoud Reda Abouelazm

September 12, 2024

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide a detailed description of the requirements for the *Bank Account Management System*. It includes both functional and non-functional requirements to guide developers and stakeholders.

1.2 Scope

The system will support multiple account types, such as SavingsAccount, CheckingAccount, CreditAccount, and InvestmentAccount. It will allow users to perform banking operations interactively, including deposits, withdrawals, and interest calculation.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **BankAccount**: A base class representing a bank account.
- **IInterestEarning**: An interface defining methods for interest calculation and application.

2 System Overview

The system will manage different types of bank accounts, allowing users to perform various operations such as depositing, withdrawing, and calculating interest. The system will implement object-oriented principles with inheritance and interfaces, providing a modular and extensible design.

3 Functional Requirements

3.1 Base Class: BankAccount

Description: The system must include a base class `BankAccount` with the following properties and methods:

- **AccountNumber** (string, 10 characters): Automatically generated, unmodifiable, and unique. The first three characters are capital letters referring to the account type, and the last seven characters are random digits.
- **AccountHolderName** (string): Name of the account holder.
- **Balance** (float): Represents the current balance of the account.
- `Deposit(amount)`: Adds a specified amount to the account balance.
- `Withdraw(amount)`: Deducts a specified amount from the account balance if sufficient.
- `CheckBalance()`: Returns the current balance of the account.

3.2 Derived Classes

The system will have the following derived classes that inherit from `BankAccount`:

- **SavingsAccount**: Implements `IInterestEarning`. A 3% monthly interest is applied.
- **CheckingAccount**: No interest, standard bank operations.
- **InvestmentAccount**: Implements `IInterestEarning` with 5% monthly interest.
- **CreditAccount**: Includes an additional `CreditLimit` attribute and implements specific withdrawal behavior.

3.3 Interface: IInterestEarning

Description: The system will define an interface `IInterestEarning`, which includes:

- `CalculateInterest()`: Calculates the interest as a percentage of the balance.
- `ApplyInterest()`: Adds the calculated interest to the balance and reports the change.

3.4 Specific Account Behaviors

- **CreditAccount**:
  - `Withdraw(amount)`: If the balance is insufficient, the system will deduct from the `CreditLimit`. If both the balance and `CreditLimit` are insufficient, an error will be reported.
- **SavingsAccount and InvestmentAccount**:
  - Calculate and apply interest based on the balance.
  - An extension method will be introduced to calculate the total interest earned between two dates (in `dd/MM/yyyy` format).

3.5 Exception Handling

The system must handle exceptions for deposit and withdrawal operations, ensuring that no transaction can proceed if the amount is non-positive. Error messages must start with the word `Error` and provide details on the issue.

3.6 Testing

Description: The system will include a `Main` method that creates objects for each account type and stores them in a list. Operations such as deposit, withdrawal, and interest calculation will be performed for testing purposes. Specifically:

- Perform deposit and withdrawal operations for each account.
- For `SavingsAccount` and `InvestmentAccount`, calculate total interest earned between two given dates.
- Apply interest only to `InvestmentAccount`.
- If any transaction fails, only that specific transaction will stop, and the appropriate error message will be displayed.

4 Non-Functional Requirements

4.1 Performance Requirements

The system must process deposits, withdrawals, and interest calculations within 1 second of input.

4.2 Usability

The system must provide clear and concise error messages for invalid operations.

4.3 Reliability

The system must ensure data consistency for all transactions.

4.4 Formatting

All monetary values must be formatted with two decimal places, e.g., `E£9.00`.

5 Examples

5.1 Sample Run

```
Account Type: SavingsAccount
AccountHolder: Mohammed Kareem

Account SAV1640361 balance: E£600.00
Please enter amount deposited:
10
Deposited E£10.00 into account SAV1640361. New balance: E£610.00
Please enter amount withdrawn:
500
Withdrawn E£500.00 from account SAV1640361. New balance: E£110.00
Calculating interest between two dates
Total interest earned in 38 months: E£122.10
```