Banking Requirements and Design

# Problem Statement

A small town bank has been established and needs a system to maintain a list of about 1000 customers along with their various bank accounts. These account types may include: savings and checking (multiple of each if necessary). Interest will be paid towards checking and savings accounts. All checking accounts also will be charged monthly for service at a rate of $10; savings accounts aren’t charged monthly and both accounts gain compound interest monthly.

The bank will also be giving out fixed-term loans. These loans will be given out at an APR of 5% only to customers that currently have a bank account, and a positive balance. Interest will be compounded monthly and payments are due monthly.

Bank Tellers must be able to access the list of customers and their associated balances for each account. In addition, they must also be able to commit activities against each account from a customer. These operations include: deposit, withdraw, check balances, and moving money between accounts belonging to a single customer.

The Bank Manager must be able to get the total balance of all accounts and find customers whose accounts have the highest balances to send them special offers to thank them for their business.

# Interpretation

The bank allows only two types of accounts, checking and savings, which a customer may have multiple of. In addition, the bank also gives out loans to current customers with a bank account with a positive balance. The APR is compounded monthly and payments are also due monthly. Bank Tellers and Manager are very similar in roles, of course with most of the admin rights given solely to the Manager. The loan process itself is simple as the program calculations can get out of hand quickly.

# Classes

**BankAccount:** This class contains all information about an account. These include operations to maintain or adjust the balance through withdrawals, account number, who owns the account, interest rate application (if necessary), know which kind of account it is (checking or savings), and apply a service charge to the necessary account.

**BankAccounts:** This class is to maintain a list of accounts, more than likely an ArrayList. It should know who owns each account, the accountID stored as the account number for linking purposes, and what type of account each is.

**Bank:** Contains information about all accounts and customers. Certain things that are specific to a customer and would be security breaches, such as a customer’s SSN, will not be included. In addition, it will also provide a way to link customers and their owned accounts.

**Customer:** Contains details of a customer. These include SSN and first and last name.

**Customers:** Contains details of the entire customer base. It allows you to search lists of accounts, customers, add and delete customer accounts, in addition to adding and removing customers themselves.

**BankTeller:** This is a UI for the Tellers only. They can access customer accounts, deposit, withdraw, and move funds, and create new accounts for customers.

**BankManager:** This provides a UI for the Manager. It allows them to find accounts and associated information to send correspondence to a customer thanking them for their business. In addition, they also can view total deposits in the bank currently and outstanding loans.

**BankProgram:** Contains a main method, it allows a teller or a manager to log in and use the entire program from their own respective angles. It will also only allow a Teller or Manager to have a certain interface each as well as their own roles that the manager will have the ability to directly oversee.

**InsufficientFundsException:** This reports an exception detailing when someone attempts to withdraw more money than they already have.

**Loan:** This class is to calculate loans.

**Loans:** This class is designed to hold a list of loans and their associated information, such as payment dates, that are calculated in the Loan Class.

# Class Methods

## BankAccount

**Deposit:** Add funds to the account.

**Withdraw:** Remove funds from the account, provided it has enough funds to do so. The InsufficientFundsException is thrown otherwise.

**getAccountNumber:** Gets the associated bank account number.

**getBalance:** Get the total balance of an account.

**getCustomer:** Get the customer who owns this account.

**addInterest:** Apply interest to the account.

**applyServiceCharge:** Applies the service charge to the account, provided it’s a checking account.

## BankAccounts

**Find:** Find an account by account number.

**addAccount:** Add an account to the ArrayList. It also will include associated information related to said account, should it be a new customer.

**removeAccount:** Remove an account from the ArrayList.

## Bank

**findAccounts:** Retrieve a current list of all accounts.

**readAccounts:** Read the current account information from a file and store it in the BankAccounts ArrayList.

**readCustomers:** Read the current customer information from a file and store it in the Customers ArrayList.

**saveAccounts:** Save the account information to a file.

**saveCustomers:** Saves the customer information to a file.

## Customer

**getName:** Get the customer’s full name.

**setName:** Set the name for a customer, and only for themselves.

**getAddress:** Get the customer’s permanent address.

**setAddress:** Set the customer’s permanent address.

**setSSN:** Set the customer’s SSN.

**getSSN:** Get the customer’s SSN.

## Customers

**FindByCustomerNumber:** Search the list of customers by their associated Account Number.

**FindByCustomerName:** Search the list of customers by name.

**addCustomerAccount:** Add an account associated with a customer.

**removeCustomerAccount:** Remove an account specific to a customer.

**RemoveCustomer:** Remove a customer from the current list of customers and any associated bank account.

**addCustomer:** Add a customer to the current list of customers.

## BankTeller

**Teller:** This provides a menu for a teller to access a customer and their respective accounts. In addition, it provides access to create and remove a customer along with their account(s). They are also allowed to deposit or withdraw funds and get the current balance for any and all customer account balances.

## BankManager

**Manager:** This function provides a menu for the manager to find accounts, find loan information, and get a total calculation of how much money is deposited in the bank.

## BankProgram

**main:** This function contains the login for each banking user, either the BankTeller or BankManager. It will call only one of the main methods for either the Teller or Manager.

## InsufficientFundsException

**getMessage:** The purpose of this function is to get the description of the exception.

**printStackTrace:** The purpose of this function is to print a trace of where an exception occurred.

## Loan

**setAPR:** Sets the APR of the loan.

**getAPR:** Get the current loan’s APR.

**setLoanTerm:** Sets the loan’s term to pay off the loan.

**getLoanTerm:** Gets the loan’s current term.

**getLoanBalance:** Gets the current balance of the loan.

**getLoanOwner:** Finds the current owner of the loan to apply calculations.

**getPaymentSchedule:** This class calculates and provides a graphic of the payment schedule for the loan.

**calculateInterestPayment:** This class calculates APR against the current term’s loan balance, likely stored in an ArrayList.

**calculatePrincipalPayment:** This class calculates how much a customer must pay out of the principal as part of the monthly payment.

## Loans

**addLoan:** Adds a loan to the ArrayList and associates it to a customer.

**getLoanList:** Retrieves a list of loans that are associated with each customer, along with the APR, Term, Balance.

**SaveLoans:** Saves the list of loans to a file.

# Summary

The Bank, Customer, Customers, BankAccount, BankAccounts, Loan, and Loans classes provide the needed functionality to operate the bank. The BankTeller and BankManager classes provide functionality to ensure that there are no errors and can interface with the other classes to make adjustments, such as opening a new account. The BankProgram Glass provides a way to run the program and access accounts depending which type of user it is, either the Manager or the Teller.