Swinburne University of Technology

COS20007 Object Oriented Programming

D Level Custom Program Initial Plan

PDF generated at 16:34 on Sunday 26th November, 2023

Produced by OOP Submission Formatter

Design Overview for Bank ATM

Name: Le Gia Hoang An
Student ID: 104789808

Summary of Program

Describe what you want the program to do… one or two paragraphs.
My goal is project was to be able to replicate the software interfaces that we interacted with within ATM stands. I want to be able to capture the functionalities of an ATM software, such as performing transactions, viewing account details, and making changes to personal details. My program allows the user to make three different types of transactions: deposit, transfer, and withdraw. The program will also have a list of transactions history to keep track of the money that is going in and out of the user's account. Each transaction will contain details of the amount of money, date and time of occurrence, account number, transaction type, and a description of transaction. Specifically, the receiver of a transfer transaction can see who sent them the money in the transaction description.

Include a sketch of sample output to illustrate your idea.

Required Roles

Describe each of the classes, interfaces, and any enumerations you will create. Use a different table to describe each role you will have, using the following table templates.

Table 1 – Class: Account

Responsibility	Type Details	Notes
The role of this class is to hold account details such as account id, customer, pin, and account type for the Bank for records.	string _id Customer _customer int _pin string _typeAccount	This class is declared as an abstract and is used as the baseline for further development in SavingsAccount and CurrentAccount classes.

Table 2 – Class: Bank

Responsibility	Type Details	Notes
The role of this class is to hold a list of accounts that are registered under a bank brand.	string _bankName List<Account> _listAccount	Each object that was created from this class will act as a separate bank. While it has a list to hold Account objects, it can also loop through the list to

		verify if the account in question exist in the list. This function is used for user authentication.

Table 3.1 – Class: ATM

Responsibility	Type Details	Notes
The role of this class is to act as the main interface where all the command inputs from the user will be processed.	string _location Bank _bank Account _currentAccount Customer _currentUser AccountType _accountType	Each object that was created from this class will act as a separate ATM that is associated with either the same or different bank. The details of an ATM such as location and associated bank may vary.

Table 3.2 – Enumeration of Class ATM: AccountType

Value	Notes
Saving	This value is registered to the variable _accountType when the user selects their saving account.
Spending	This value is registered to the variable _accountType when the user selects their spending account.
None	This is the default value for the program's initialisation and user logging out.

Table 4 – Class: Customer

Responsibility	Type Details	Notes
The role of this class is to hold personal information of a customer. Details include name, phone, email, and address.	string _name string _phone string _email string _address	Each object that is created under this class will act as a separate customer and it is intended to act the scenario where account creation was done by a bank administrator. This class also allows user to view their information or update them.

Table 5 – Class: Transaction

Responsibility	Type Details	Notes
The role of this class is to record every transaction that was made by the user. Details of a transaction includes date, amount of money for transaction, description, and transaction type.	Date _transactionDate string _type string _desc double _amount	All details of a transaction will be compressed into one singular string and formatted before returning the value.

Table 6 – Class: SavingsAccount

Responsibility	Type Details	Notes
The role of this class is to act as a separate bank account that is primarily for saving money. When a customer register with the bank, they will receive both saving account and current account.	List<Transaction> _transaction double _balance	This class is the child class of Account abstract class, which it contains all of Account class's details as the base for SavingsAccount, with additional details such as transaction list and account balance.

Table 7 – Class: CurrentAccount

Responsibility	Type Details	Notes
The role of this class is to act as a separate bank account that is primarily for daily spending. When a customer register with the bank, they will receive both saving account and current account.	List<Transaction> _transaction double _balance	This class is the child class of Account abstract class, which it contains all of Account class's details as the base for CurrentAccount, with additional details such as transaction list and account balance.

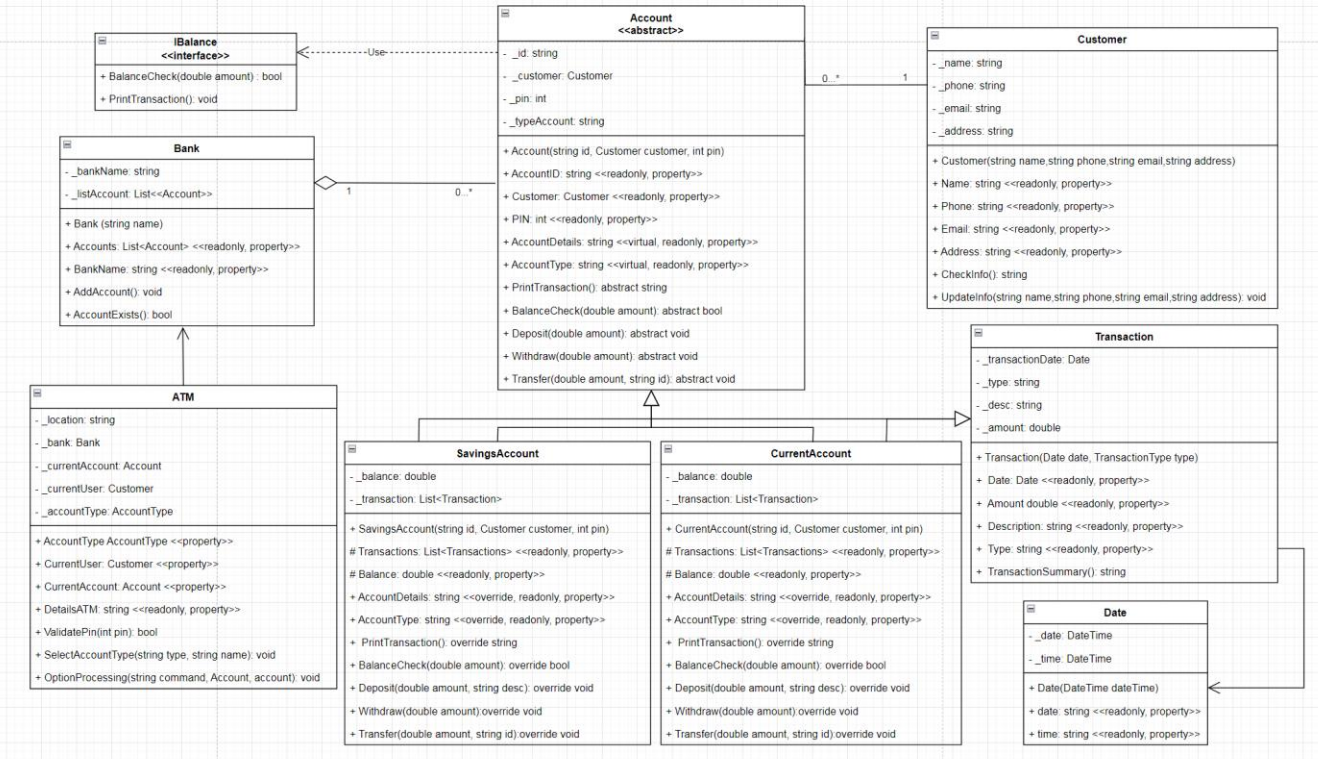
Table 8 – Interface: IBalance

Responsibility	Type Details	Notes

This interface is made to ensure consistency in Account class, so that both SavingsAccount and CurrentAccount are not missing any necessary functions.	bool BalanceCheck(double amount);
	void PrintTransaction();
		Since Account is the abstract class, the functions that were prewritten in IBalance will be enforced on SavingsAccount and CurrentAccount.

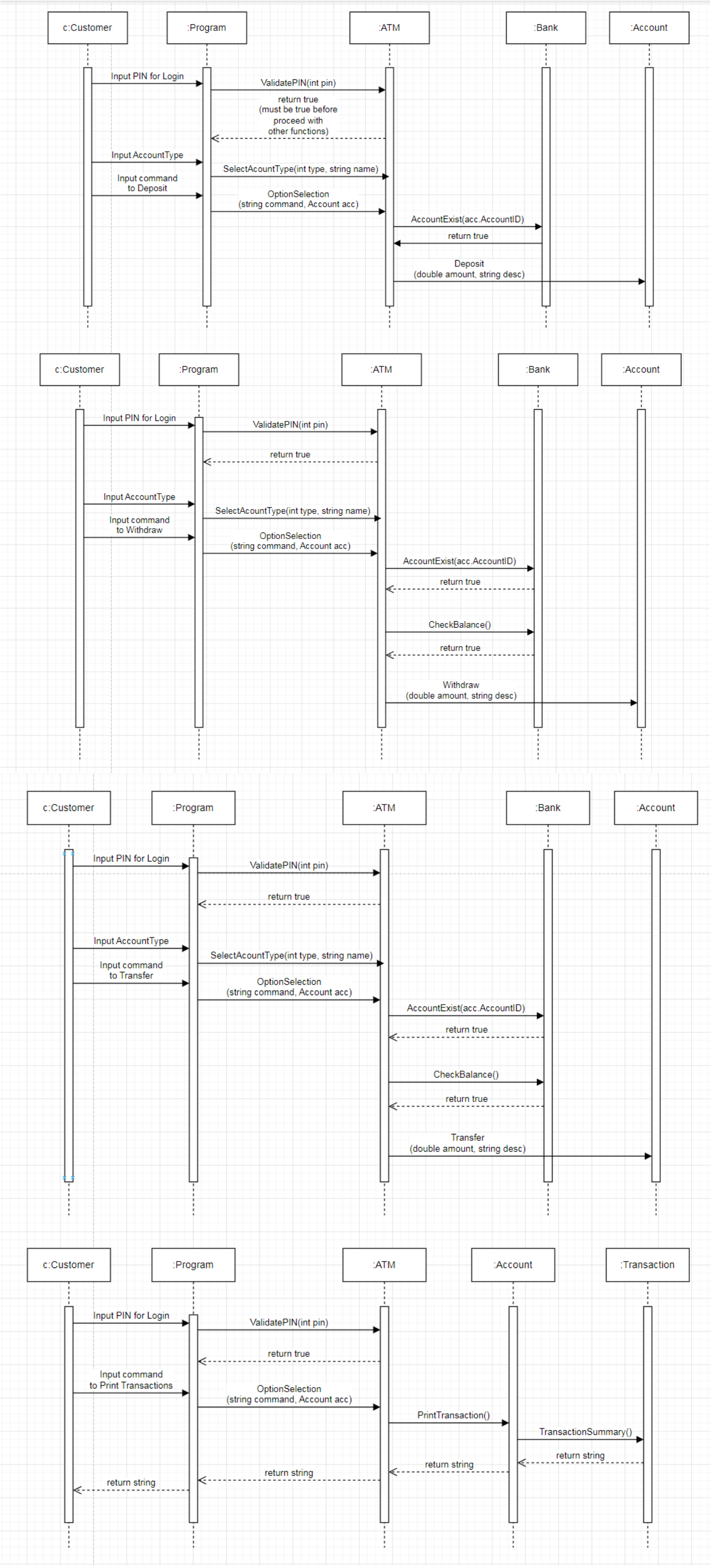
Class Diagram

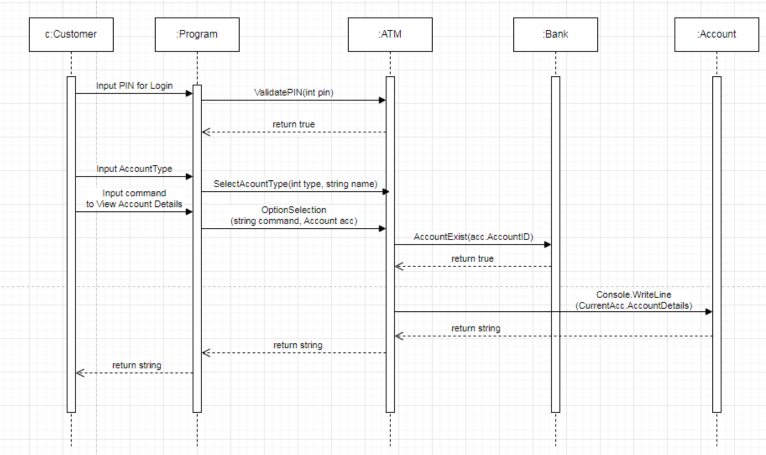
Provide an initial design for your program in the form of a class diagram.

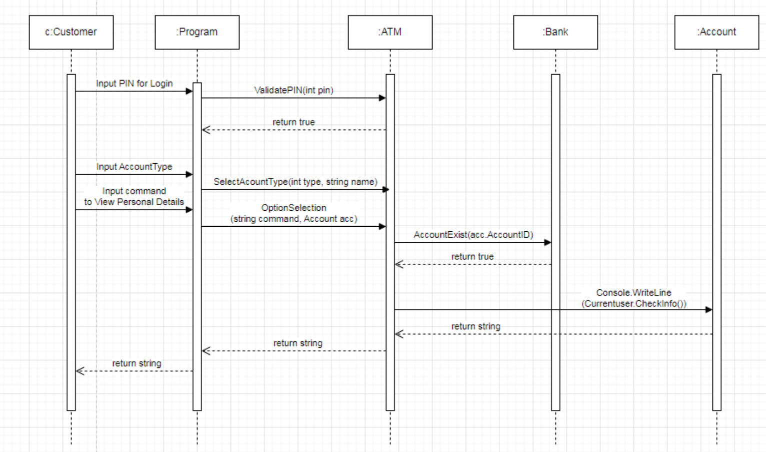

IBalance
<<interface>>
+ BalanceCheck(double amount) : bool
+ PrintTransaction(): void
Use
Account
<<abstract>>
- _id: string
- _customer: Customer
- _pin: int
- _typeAccount: string
+ Account(string id, Customer customer, int pin)
+ AccountID: string <<readonly, property>>
+ Customer: Customer <<readonly, property>>
+ PIN: int <<readonly, property>>
+ AccountDetails: string <<virtual, readonly, property>>
+ AccountType: string <<virtual, readonly, property>>
+ PrintTransaction(): abstract string
+ BalanceCheck(double amount): abstract bool
+ Deposit(double amount): abstract void
+ Withdraw(double amount): abstract void
+ Transfer(double amount, string id): abstract void
0...*
1
Customer
- _name: string
- _phone: string
- _email: string
- _address: string
+ Customer(string name,string phone,string email,string address)
+ Name: string <<readonly, property>>
+ Phone: string <<readonly, property>>
+ Email: string <<readonly, property>>
+ Address: string <<readonly, property>>
+ CheckInfo(): string
+ UpdateInfo(string name,string phone,string email,string address): void
Bank
- _bankName: string
- _listAccount: List<<Account>>
+ Bank (string name)
+ Accounts: List<Account> <<readonly, property>>
+ BankName: string <<readonly, property>>
+ AddAccount(): void
+ AccountExists(): bool
1
0...*
ATM
- _location: string
- _bank: Bank
- _currentAccount: Account
- _currentUser: Customer
- _accountType: AccountType
+ AccountType AccountType <<property>>
+ CurrentUser: Customer <<property>>
+ CurrentAccount: Account <<property>>
+ DetailsATM: string <<readonly, property>>
+ ValidatePin(int pin): bool
+ SelectAccountType(string type, string name): void
+ OptionProcessing(string command, Account, account): void
SavingsAccount
- _balance: double
- _transaction: List<Transaction>
+ SavingsAccount(string id, Customer customer, int pin)
# Transactions: List<Transactions> <<readonly, property>>
# Balance: double <<readonly, property>>
+ AccountDetails: string <<override, readonly, property>>
+ AccountType: string <<override, readonly, property>>
+ PrintTransaction(): override string
+ BalanceCheck(double amount): override bool
+ Deposit(double amount, string desc): override void
+ Withdraw(double amount):override void
+ Transfer(double amount, string id):override void
CurrentAccount
- _balance: double
- _transaction: List<Transaction>
+ CurrentAccount(string id, Customer customer, int pin)
# Transactions: List<Transactions> <<readonly, property>>
# Balance: double <<readonly, property>>
+ AccountDetails: string <<override, readonly, property>>
+ AccountType: string <<override, readonly, property>>
+ PrintTransaction(): override string
+ BalanceCheck(double amount): override bool
+ Deposit(double amount, string desc): override void
+ Withdraw(double amount):override void
+ Transfer(double amount, string id):override void
Transaction
- _transactionDate: Date
- _type: string
- _desc: string
- _amount: double
+ Transaction(Date date, TransactionType type)
+ Date: Date <<readonly, property>>
+ Amount double <<readonly, property>>
+ Description: string <<readonly, property>>
+ Type: string <<readonly, property>>
+ TransactionSummary(): string
Date
- _date: DateTime
- _time: DateTime
+ Date(DateTime dateTime)
+ date: string <<readonly, property>>
+ time: string <<readonly, property>>

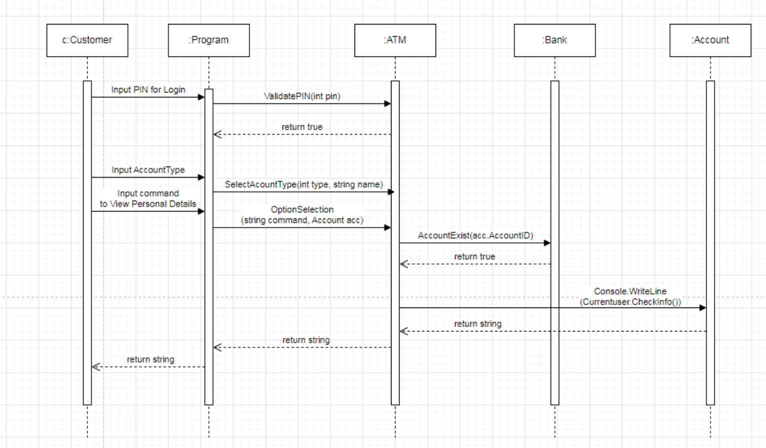
Sequence Diagram

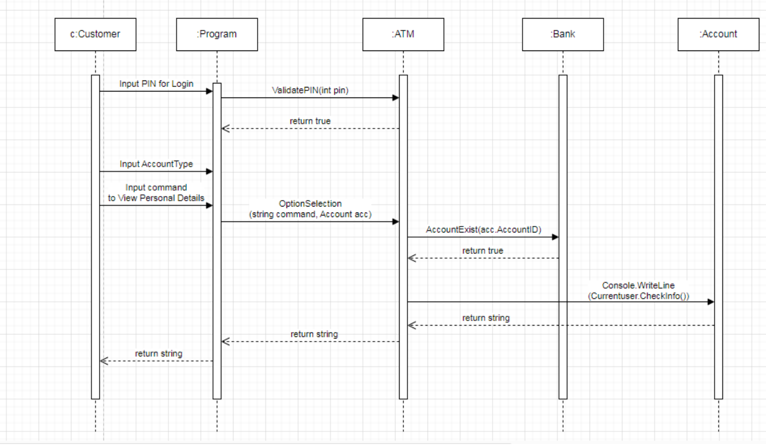
Provide a sequence diagram showing how your proposed classes will interact to achieve a specific piece of functionality in your program.




c:Customer
:Program
:ATM
:Bank
:Account
Input PIN for Login
ValidatePIN(int pin)
return true
Input AccountType
SelectAcountType(int type, string name)
Input command
to View Account Details
OptionSelection
(string command, Account acc)
AccountExist(acc.AccountID)
return true
Console.WriteLine
(CurrentAcc.AccountDetails)
return string
return string
return string


c:Customer
:Program
:ATM
:Bank
:Account
Input PIN for Login
ValidatePIN(int pin)
return true
Input AccountType
SelectAcountType(int type, string name)
Input command
to View Personal Details
OptionSelection
(string command, Account acc)
AccountExist(acc.AccountID)
return true
Console.WriteLine
(Currentuser.CheckInfo())
return string
return string
return string


c:Customer
:Program
:ATM
:Bank
:Account
Input PIN for Login
ValidatePIN(int pin)
return true
Input AccountType
SelectAcountType(int type, string name)
Input command
to View Personal Details
OptionSelection
(string command, Account acc)
AccountExist(acc.AccountID)
return true
Console.WriteLine
(Currentuser.CheckInfo())
return string
return string
return string


c:Customer
:Program
:ATM
:Bank
:Account
Input PIN for Login
ValidatePIN(int pin)
return true
Input AccountType
Input command
to View Personal Details
OptionSelection
(string command, Account acc)
AccountExist(acc.AccountID)
return true
Console.WriteLine
(Currentuser.CheckInfo())
return string
return string
return string