WATERMELON BANKING SYSTEM
SOFTWARE DESIGN DOCUMENT

Denesh Parthipan 100536986
Luisa Rojas 100518772
Truyen Truong 100516976

1 Introduction

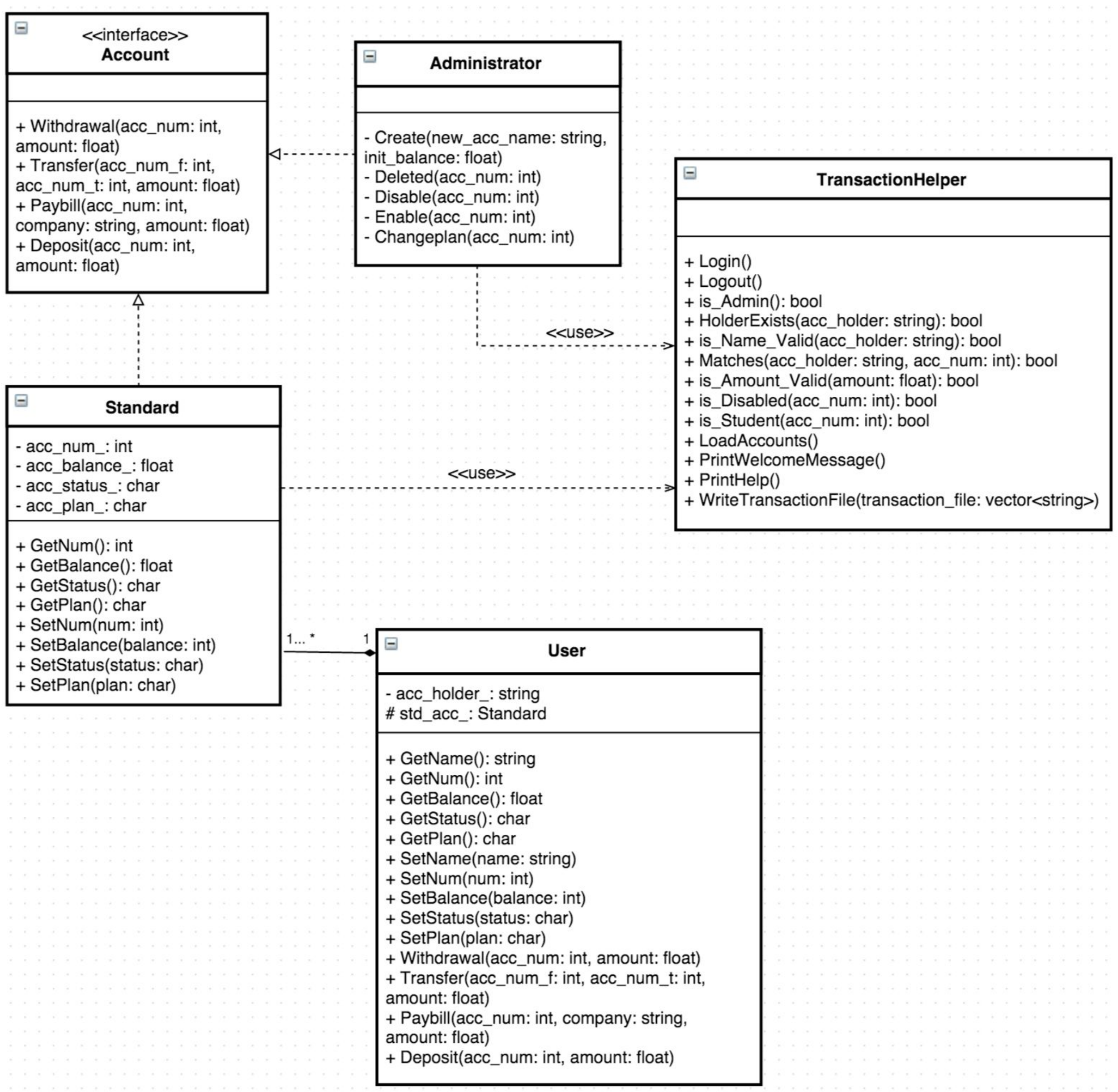
Watermelon Banking System is a banking system that allows users to create, manage, and perform transactions with their accounts. The design of our system is similar in functionality to how a typical banking system would operate. This software design document provides a high level structure of the front end design of the Watermelon Banking System. The architecture of the system is derived from the requirements that outline what users can do in the system.

This design document specifies all the transactions that make up the Watermelon Banking System. The transactions are organized into methods, and a corresponding description is provided for all methods. Moreover, key variables are also included in the design document and their purpose. A UML class diagram is used to illustrate this.

2 Design Requirements

Login → start a Front End session, logging in as either admin or a standard user
Withdrawal → withdraw money from your own bank account, an amount that is within range, and also withdraw to from account as a privileged transaction
Transfer → transfer money from your bank account to another, an amount that is within range, and also transfer to and from any account as a privileged transaction
Paybill → pay a bill from your bank account to any company, an amount that is within range, and also pay bill from any account as a privileged transaction
Deposit → deposit money from your own bank account, an amount that is within range, and also deposit to any account as a privileged transaction
Create → Create a bank account (privileged transaction)
Delete → Delete a bank account (privileged transaction)
Enable → Enables all transactions in a bank account
Disable → Disable all transactions in a bank account (privileged transaction)
Changeplan → Change the transaction plan of a bank account (privileged transaction)

3 UML Class Diagram



<<interface>>
Account
+ Withdrawal(acc_num: int, amount: float)
+ Transfer(acc_num_f: int, acc_num_t: int, amount: float)
+ Paybill(acc_num: int, company: string, amount: float)
+ Deposit(acc_num: int, amount: float)
Administrator
- Create(new_acc_name: string, init_balance: float)
- Deleted(acc_num: int)
- Disable(acc_num: int)
- Enable(acc_num: int)
- Changeplan(acc_num: int)
TransactionHelper
+ Login()
+ Logout()
+ is_Admin(): bool
+ HolderExists(acc_holder: string): bool
+ is_Name_Valid(acc_holder: string): bool
+ Matches(acc_holder: string, acc_num: int): bool
+ is_Amount_Valid(amount: float): bool
+ is_Disabled(acc_num: int): bool
+ is_Student(acc_num: int): bool
+ LoadAccounts()
+ PrintWelcomeMessage()
+ PrintHelp()
+ WriteTransactionFile(transaction_file: vector<string>)
<<use>>
<<use>>
Standard
- acc_num_: int
- acc_balance_: float
- acc_status_: char
- acc_plan_: char
+ GetNum(): int
+ GetBalance(): float
+ GetStatus(): char
+ GetPlan(): char
+ SetNum(num: int)
+ SetBalance(balance: int)
+ SetStatus(status: char)
+ SetPlan(plan: char)
1... *
1
User
- acc_holder_: string
# std_acc_: Standard
+ GetName(): string
+ GetNum(): int
+ GetBalance(): float
+ GetStatus(): char
+ GetPlan(): char
+ SetName(name: string)
+ SetNum(num: int)
+ SetBalance(balance: int)
+ SetStatus(status: char)
+ SetPlan(plan: char)
+ Withdrawal(acc_num: int, amount: float)
+ Transfer(acc_num_f: int, acc_num_t: int, amount: float)
+ Paybill(acc_num: int, company: string, amount: float)
+ Deposit(acc_num: int, amount: float)

4 Code Design

Standard Class

Brief Description: This class contains the properties of a standard user	
Attributes	**Attribute Description**
`acc_num_ : int`	Stores the bank account number for the standard user owning this account.
`acc_balance_ : float`	Stores the account balance for the standard user owning this account.
`acc_status_ : char`	Stores the account status (Active or Disabled) for the standard user owning this account.
`acc_plan_ : char`	Stores the account transaction payment plan for the standard user owning this account.
Method Name	**Method Description**
`GetNum() : int`	Returns the bank account number for the standard user owning this account.
`GetBalance() : float`	Returns the account balance for the standard user owning this account.
`GetStatus() : char`	Returns the account status for the standard user owning this account.
`GetPlan() : char`	Returns the account plan for the standard user owning this account.
`GetNum(num: int)`	Takes in an integer input, and sets this as the account number for the standard user owning this account.
`SetBalance(balance: float)`	Takes in a float input, and sets the account balance amount for the standard user
`SetStatus(status: char)`	Takes in a character input, and updates the account status for the standard user using this value.
`SetPlan(plan: char)`	Takes in a character input, and sets the account plan type for the standard user using this value.

Administrator Class

Brief Description: This class contains all the privileged transaction available only for an admin	
Method Name	**Method Description**
`User create(new_acc_name: string, init_balance: float):` *`User`*	Takes in the account holder's name and the initial balance, and creates a new user as well as a new bank account in the system and adds it to the program-local `users` vector.
`Deleted(acc_num: int)`	Takes in the account number the administrator account wishes to delete and removes it from the `users` vector.
`Disable(acc_num: int)`	Takes in the account number the administrator account wishes to disable and keeps it from making any transaction commands, except for `login()` and `logout()`.
`Enable(acc_num: int)`	Takes in the account number the administrator account wishes to enable and enables that bank account
`Changeplan(acc_num: int)`	Takes in the account number you want to change the payment plan for, and changes the plan

User Class

Brief Description: Class spawns Standard objects	
Attributes	**Attribute Description**
`acc_holder_:` *`string`*	Stores the account holder's name
`std_acc_ :` *`Standard`*	Stores an object of the Standard class
Method Name	**Method Description**
`GetName() :` *`string`*	Returns the account name for the user object
`GetNum() :` *`int`*	Returns the account number from the Standard object, which is owned by the current user

`GetBalance():` *`float`*	Returns the bank balance from the Standard object, which is owned by the user
`GetStatus():` *`char`*	Returns the bank status from the Standard object, which is owned by the user
`GetPlan():` *`char`*	Returns the transaction payment plan from the Standard object, which is owned by the user
`SetName(name: string)`	Sets the account name variable in the User object
`SetNum(num: int)`	Sets the account number variable in the Standard object through User
`SetBalance(balance: float)`	Sets the account balance variable in the Standard object through User
`SetStatus(status: char)`	Sets the account status variable in the Standard object through User
`SetPlan(plan: chra)`	Sets the transaction payment plan variable in the Standard object through User
`Withdrawal(acc_num: int, amount: float)`	Invokes the `Withdrawal(acc_num: int, amount: float)` function in the Standard object
`Transfer(acc_num_f: int, acc_num_t: int, amount: float)`	Invokes the `Transfer(acc_num_f: int, acc_num_t: int, amount: float)` function in the Standard object
`Paybill(acc_num: int, company: string, amount: float)`	Invokes the `Paybill(acc_num: int, company: string, amount: float)` function in the Standard object
`Deposit(acc_num: int, amount: float)`	Invokes the `Deposit(acc_num: int, amount: float)` function in the Standard object

TransactionHelper Class

Brief Description: This class contains all the error checks and constraint checks necessary to operate a functional banking system	
Attributes	**Attribute Description**

curr_user_ : *User*	Stores an object of the User class that is currently performing transactions
users_ : vector<*User*>	Stores the list of all users
acc_status_ : *vector<String>*	Stores the list of all account statuses
mode_ : *String*	Stores the mode of operation
Method Name	**Method Description**
is_Admin() : *bool*	Returns true if user is an admin and false if not
HolderExists(acc_holder: string) : *bool*	Returns true if the account holder exists in the banking system, and false if not
is_Name_Valid(acc_holder: string) : *bool*	Returns true if the account name is valid and false if not
Matches(acc_holder: string, acc_num: int): *bool*	Returns true if the account holder’s name matches with the corresponding account number
is_Amount_Valid(amount: float): *bool*	Returns true if the amount if valid
is_Disabled(acc_num: int): *bool*	Returns true if account is disabled, false if not
is_Student(acc_num: int): *bool*	Returns true if account is a student, false is not
LloadAccounts()	Loads the account you want to use
PrintWelcomeMessage()	Outputs Banking Welcome messages
PrintHelp()	Outputs the help screen
WriteTransactionFile(transaction_file: vector<string>)	Outputs the transaction log file for the user

Account Class

Brief Description: This class contains transactions that can be done by both standard users and admin	
Method Name	**Method Description**

`Withdrawal(acc_num: int, amount: float)`	Takes in the account number and withdraws amount from that bank account
`Transfer(acc_num_f: int, acc_num_t: int, amount: float)`	Takes in two account numbers and transfers amount from the first account to the second account
`Paybill(acc_num: int, company: string, amount: float)`	Takes in the account number and pays amount from the account to company
`Deposit(acc_num: int, amount: float)`	Takes in the account number and deposits amount into that account
`Changeplan(acc_num: int)`	Takes in the account number and changes the payment plan for that account