WATERMELON BANKING SYSTEM
BACK END SOFTWARE DESIGN DOCUMENT

Denesh Parthipan 100536986
Luisa Rojas 100518772
Truyen Truong 100516976

1 Introduction

Welcome to Watermelon Banking System's Transaction Processor! We process all valid transactions initiated by the user and updates the master bank accounts file reflecting the transactions completed.

This design document provides an overview of the functionality of the Transaction Processor and how it's components interact. The relationships are modeled in a UML Class diagram shown below. The classes and their corresponding functions are described in the table below.

2 Back End Design Requirements

General Account Constraints → Bank account balances should not be negative and new created accounts must have an unique bank account number

General Back End Constraint → Assume for correct input, but check for bad input and immediately stop and log a fatal error.

Master/Current Bank Accounts Files → Takes in the old master bank account file and applies the transactions made, and produces a new master bank accounts file and current bank accounts files. It follows a specific format, shown below:

1) Every line is exactly 42 characters (including newline)
2) numeric fields are right justified, filled with zeroes (e.g., 00023 for bank account 23)
3) alphabetic fields are left justified, filled with spaces (e.g. John_Doe____________ for bank account holder John Doe)
4) unused numeric fields are filled with zeros (e.g., 0000)
5) In a numeric field that is used to represent a monetary value, ".00" is appended to the end of the value (e.g. 00110.00 for 110)
6) unused alphabetic fields are filled with spaces (blanks) (e.g., ________________________)
7) the Master Bank Accounts File must always be kept in ascending order by bank account number

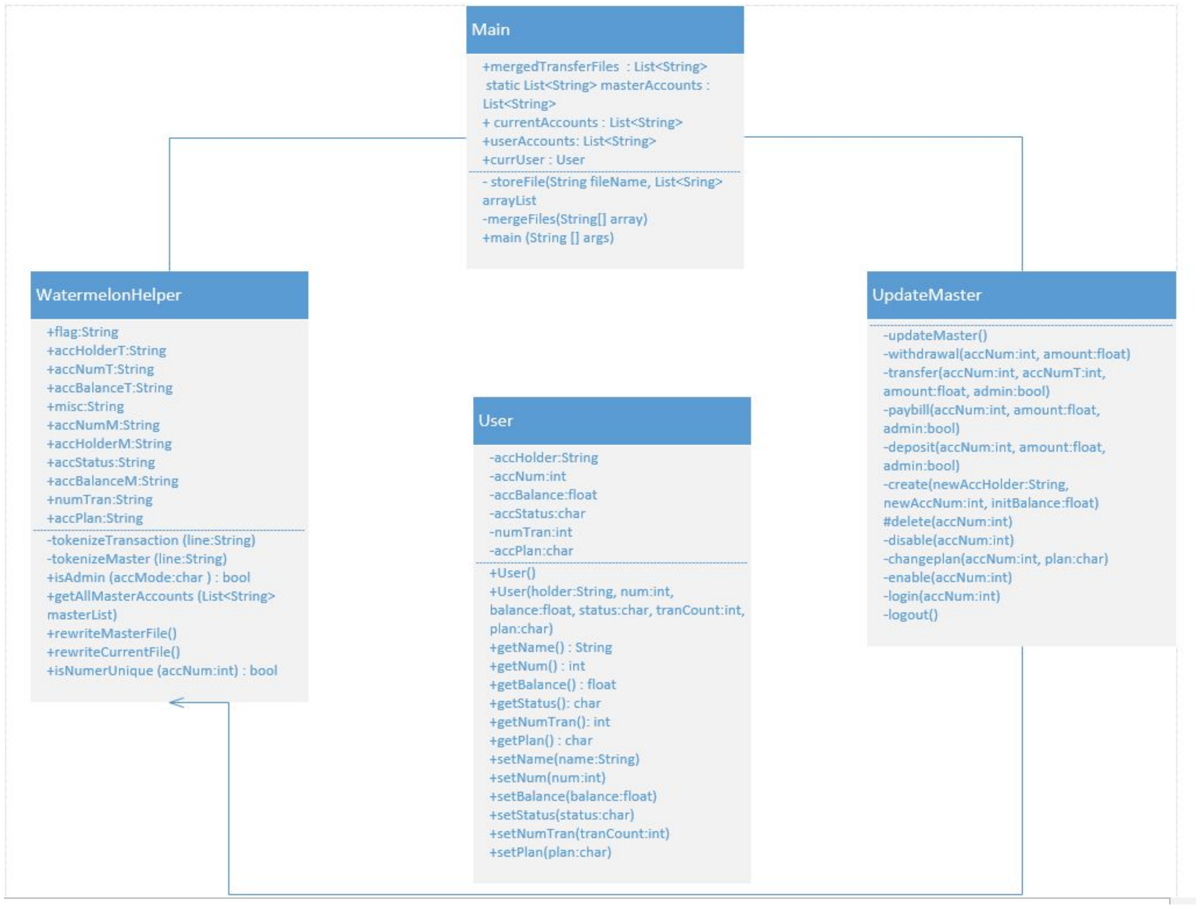
Bank Account Transaction File → Stores all of the user’s transactions for that day and writes them into a file that follows a specific format, shown below:

1) every line is exactly 40 characters (plus newline)
2) numeric fields are right justified, filled with zeroes (e.g., 00023 for bank account 23)
3) alphabetic fields are left justified, filled with spaces (e.g. John_Doe____________ for bank account holder John Doe)
4) unused numeric fields are filled with zeros (e.g., 0000)
5) In a numeric field that is used to represent a monetary value, “.00” is appended to the end of the value (e.g. 00110.00 for 110)
6) unused alphabetic fields are filled with spaces (blanks) (e.g., ________________________)
7) the sequence of transactions ends with an end of session (00) transaction code

Merge Bank Account Transaction File → Merges all bank accounts transactions files to be ready for processing.

Fees → Applies all fees based on the account plan and checks if the transaction is valid.

3 UML Diagram


Main
+mergedTransferFiles : List<String>
static List<String> masterAccounts : List<String>
+ currentAccounts : List<String>
+userAccounts: List<String>
+currUser : User
- storeFile(String fileName, List<Sring> arrayList
-mergeFiles(String[] array)
+main (String [] args)
WatermelonHelper
+flag:String
+accHolderT:String
+accNumT:String
+accBalanceT:String
+misc:String
+accNumM:String
+accHolderM:String
+accStatus:String
+accBalanceM:String
+numTran:String
+accPlan:String
-tokenizeTransaction (line:String)
-tokenizeMaster (line:String)
+isAdmin (accMode:char) : bool
+getAllMasterAccounts (List<String> masterList)
+rewriteMasterFile()
+rewriteCurrentFile()
+isNumerUnique (accNum:int) : bool
User
-accHolder:String
-accNum:int
-accBalance:float
-accStatus:char
-numTran:int
-accPlan:char
+User()
+User(holder:String, num:int, balance:float, status:char, tranCount:int, plan:char)
+getName() : String
+getNum() : int
+getBalance() : float
+getStatus(): char
+getNumTran(): int
+getPlan() : char
+setName(name:String)
+setNum(num:int)
+setBalance(balance:float)
+setStatus(status:char)
+setNumTran(tranCount:int)
+setPlan(plan:char)
UpdateMaster
-updateMaster()
-withdrawal(accNum:int, amount:float)
-transfer(accNum:int, accNumT:int, amount:float, admin:bool)
-paybill(accNum:int, amount:float, admin:bool)
-deposit(accNum:int, amount:float, admin:bool)
-create(newAccHolder:String, newAccNum:int, initBalance:float)
#delete(accNum:int)
-disable(accNum:int)
-changeplan(accNum:int, plan:char)
-enable(accNum:int)
-login(accNum:int)
-logout()

4 UML Description

<u>Main Class</u>

Brief Description: Main class with main function and file storing methods.	
Attributes	**Attribute Description**
`mergedTransferFiles : `*`List<String>`*	Holds the merged transfer files.
`masterAccounts : `*`List<String>`*	Holds master version for bank accounts.
`currentAccounts : `*`List<String>`*	Holds the current version for bank accounts.
`userAccounts : `*`List<User>`*	Holds the user bank accounts.
`currUser: `*`User`*	The current user.

Method Name	**Method Description**
`storeFile(fileName:`*`String`*`, arrayList:`*`List<String>`*`) :` *`void`*	Stores a file into a list.
`mergeFiles(array:`*`String[]`*`) :` *`void`*	Merges the transaction files into one list.

UpdateMaster Class

Brief Description: Update the master bank account file based on the merged transaction file.	
Attributes	**Attribute Description**
`None`	N/a
Method Name	**Method Description**
`updateMaster() :` *`void`*	Parses the lines from the merged transfer files. Based on the parsed lines, it will call the transaction method based on the flag of the transfer file line.
`withdrawal(accNum:`*`float`*`, amount:`*`float`*`, admin:`*`boolean`*`) :` *`void`*	Withdrawal transaction update to master bank accounts file.
`transfer(accNumF:`*`int`*`, accNumT:`*`int`*`, amount:`*`float`*`, admin:`*`boolean`*`) :` *`void`*	Transfer transaction update to master bank accounts file.
`paybill(accNum:`*`int`*`, amount:`*`float`*`, admin:`*`boolean`*`) :` *`void`*	Paybill transaction update to master bank accounts file.
`deposit(accNum:`*`int`*`, amount:`*`float`*`, admin:`*`boolean`*`) : void`	Deposit transaction update to master bank accounts file.
`create(newAccHolder:`*`String`*`, newAccNum:`*`int`*`, initBalance:`*`float`*`) :` *`void`*	Create transaction update to master bank accounts file.
`delete(accNum:`*`int`*`) :` *`void`*	Delete transaction update to master bank accounts file.
`disable(accNum:`*`int`*`) :` *`void`*	Disable transaction update to master bank accounts file.
`changeplan(accNum:`*`int`*`, plan:`*`char`*`) :` *`void`*	Changeplan transaction update to master bank accounts file.
`enable(accNum:`*`int`*`) :` *`void`*	Enable transaction update to master bank

	accounts file.
`login(accNum:int) : void`	Login transaction to check which account mode to use to update the master bank accounts file.
`logout() : void`	Logout transaction to remove account mode.

User Class

Brief Description: User class that holds the bank account information.	
Attributes	**Attribute Description**
`accHolder : String`	The account holder name.
`accNum : int`	The account number.
`accBalance : float`	The account balance.
`accStatus : char`	The account status.
`numTran : int`	The number of transactions the account has made.
`accPlan : char`	The account plan.
Method Name	**Method Description**
`User()`	Empty constructor for User class.
`User(holder:String, num:int, balance:float, status:char, tranCount:char, plan:char)`	Filled constructor for User class.
`getName() : String`	Get account holder name.
`getNum() : int`	Get account number.
`getBalance() : float`	Get account balance.
`getStatus() : char`	Get account status.
`getNumTran() : int`	Get number of transactions the account has made.
`getPlan() : char`	Get account plan.
`setName(name:String) : void`	Set account holder name.

`setNum(num:`*`int`*`) : `*`void`*	Set account number.
`setBalance(balance:`*`float`*`) : `*`void`*	Set account balance.
`setStatus(status:`*`char`*`) : `*`void`*	Set account status.
`setNumTran(tranCount:`*`int`*`) : `*`void`*	Set number of transactions the account has made.
`setPlan(plan:`*`char`*`) : `*`void`*	Set account plan.

<u>WatermelonHelper Class</u>

Brief Description: Helps with tokenizing, checks, and rewriting files.	
Attributes	**Attribute Description**
`flag : `*`String`*	Holds the flag that was parsed from the current merged transaction file line.
`accHolderT : `*`String`*	Holds the account holder name that was parsed from the current merged transaction file line.
`accNumT : `*`String`*	Holds the account number that was parsed from the current merged transaction file line.
`accBalanceT : `*`String`*	Holds the balance that was parsed from the current merged transaction file line.
`misc : `*`String`*	Holds the miscellaneous flag that was parsed from the current merged transaction file line.
`accNumM : `*`String`*	Holds the account number parsed from the master bank accounts file line.
`accHolderM : `*`String`*	Holds the account holder name parsed from the master bank accounts file line.
`accStatus : `*`String`*	Holds the account status parsed from the master bank accounts file line.
`accBalanceM : `*`String`*	Holds the account balance parsed from the master bank accounts file line.
`numTran: `*`String`*	Holds the number of transactions the bank account has made, which was parsed from the master bank accounts file line.

`accPlan : String`	Holds the account plan parsed from the master bank accounts file line.
Method Name	**Method Description**
`tokenizeTransaction(line:String) : void`	Tokenizes a line based on the format of a bank transaction file line.
`tokenizeMaster(line:String) : void`	Tokenizes a line based on the format of a master bank accounts file line.
`isAdmin(accPlan:char) : boolean`	Checks if the current logged in account is an admin. Returns true if current logged in account is an admin, and returns true if not admin.
`getAllMasterAccounts(masterList:List<String>) : void`	Parses from a list of strings and stores it in User. That User then goes in a list of users.
`rewriteMasterFile() : void`	Rewrites the master bank accounts file with the updated version.
`rewriteCurrentFile() : void`	Rewrites the current bank accounts file with the updated version.
`isNumberUnique(accNum:int) : boolean`	Checks if a bank account number is unique. If the bank account number is unique, it will return true. If the bank account number is not unique, it will return false.

5 Inputs/Outputs

Inputs →

1) Master Bank Accounts File - Read from master_bank_accounts_file.txt, and is inputted through a buffered reader. The contents are stored in a List array of type String.
2) All Transaction Files - Read from transaction_file_*.txt, where * is the transaction file number, and is inputted through a buffered reader, and stored in a List array. The contents are stored in a List array of type String.

Outputs →

1) Master Bank Accounts File - Written into master_bank_accounts_file.txt, and is outputted through a buffered writer. The contents were stored in List array of type String.
2) Current Bank Accounts File - Written into current_bank_accounts_file.txt, and is outputted

through a buffered writer. The contents were stored in List array of type String.