CSCI3060U Project Phase #2

DESIGN DOCUMENT

Front End Rapid Prototype

Taylor Smith
Alexandar Mihaylov
Talha Zia

Contents

Introduction

Purpose

The purpose of this design document is to demonstrate the design and architecture of the Front End in our Banking system. The Banking system is designed to provide regular banking transactions that could be carried out by an admin or standard user. The details regarding the banking transactions are provided in the requirements and listed below in the use cases.

Scope

The banking system will consist of two major systems – account and funds management. Both of these system could be accessed and utilized by the bank admin, however, the standard user can only utilize the fund management system.

Definitions, Acronyms and Abbreviations

Admin – Bank employee who has administrative rights to the system
Standard user – Bank client who only has access to fund management
Account Management – transactions that add, delete or modify an account
Fund Management – transactions that require funds, i.e. withdraw/deposit money, pay bills etc.

Design Overview

Description of problems

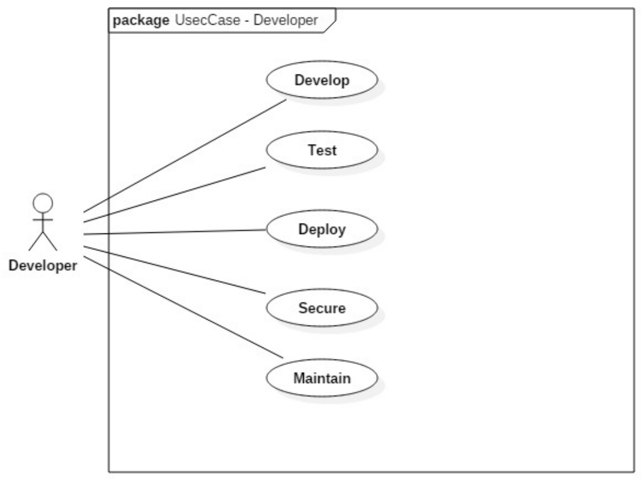
The design has to take into account the different rights each user will have, what the developers are required to do and how the system will function on a daily basis. Certain transactions can only be done by an admin, certain accounts could be disabled and inaccessible by a standard user and system has to take into account the requirements for the backend. Such details are displayed in the use cases below.

'Handler' variable names

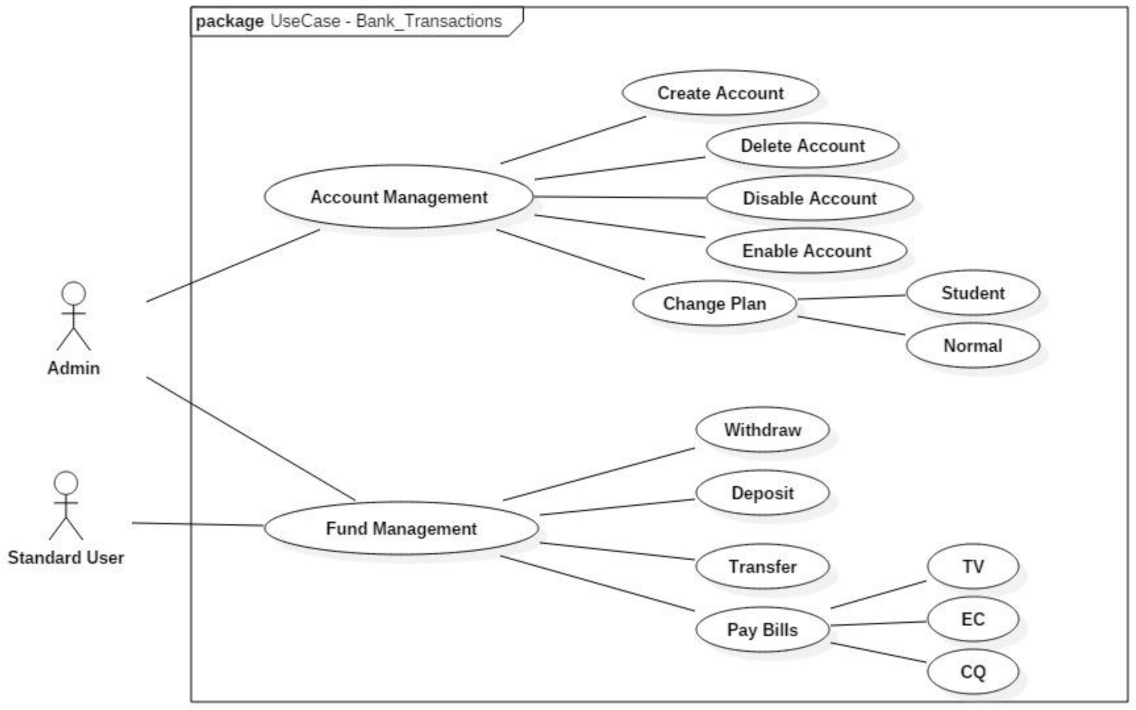
Although each handler currently has many overlapping variables (and names) this is because all current functionality uses very similar prompts. However for maintainability of code we cannot assume that all future transactions will also need these prompts and therefore cannot put them in the TransactionHandler super class. The way the Handler classes are currently implemented allows for complete separation from other handlers as all items required for handler functionality are contained within the derived handler class. This makes adding new functionality much easier as only new handlers need to be added and existing handlers or the parent TransactionHandler do not need to be modified.

Use Cases

Developers

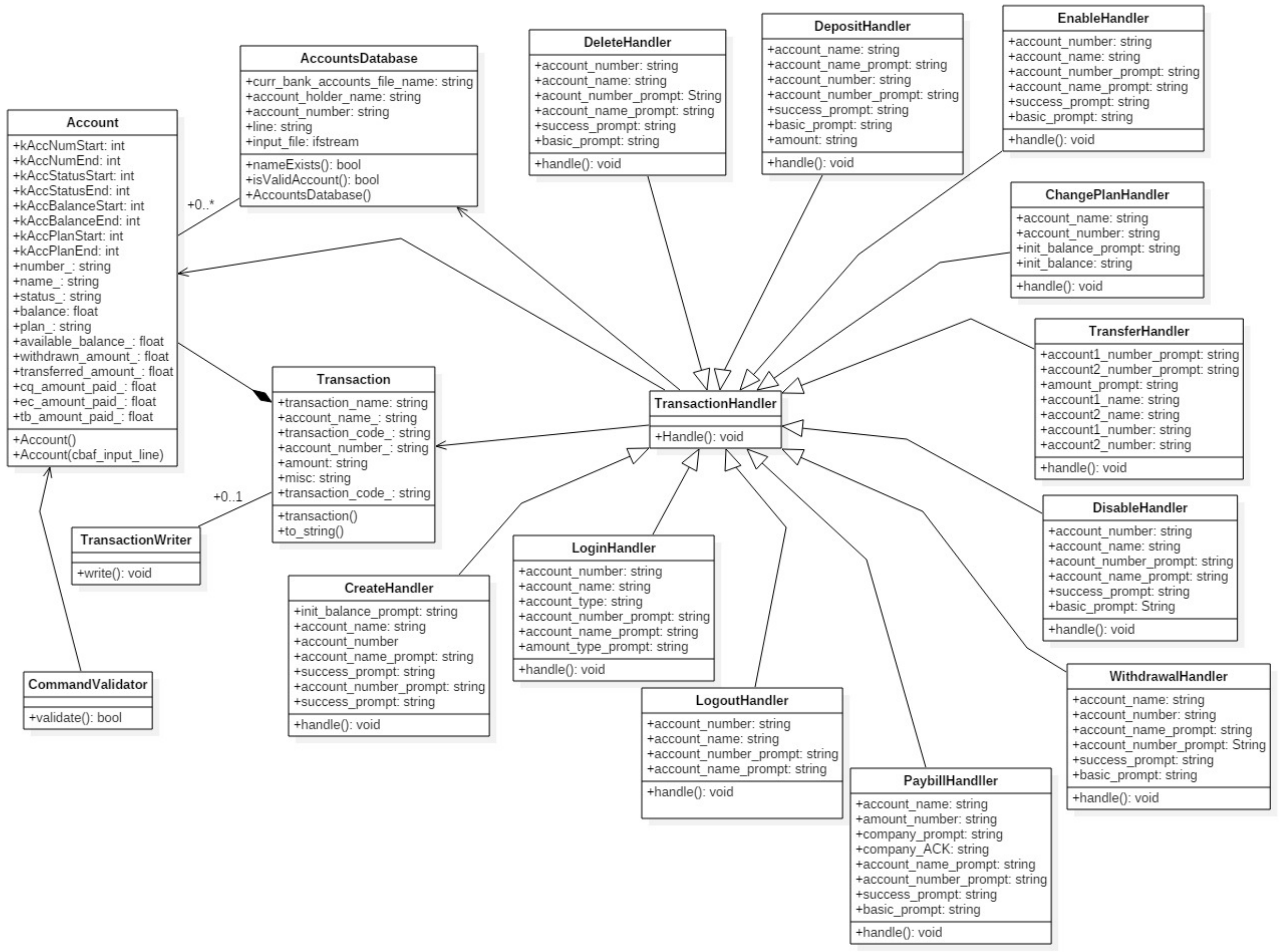


Admins and Standard User



System Architecture

Class Diagram



Description of classes

Class Name	Description	Methods
Account	Holds all information relating to a single account for the given day. Is able to take in raw input from a current bank account file and translate it	• **Account():** Constructor without any arguments, creates an admin account • **Account(cbaf_input_line):** Overloaded constructor that takes in a raw current bank account file line and parses it to fill in the appropriate members of an Account

	over to an Account object.	object: number, name, status, balance, plan
AccountsDatabase	Data Structure containing all accounts and helper methods, and also loads the currentBankAccounts File into an internal data structure of Accounts.	• **AccountsDatabase(curr_bank_accounts_file_name):** Constructor that takes in the current bank account file for the day and populates an internal data structure that holds all Account Classes • **nameExists(account_holder_name):** Checks to see if the given account name exists in the database, if it does it returns true, otherwise it returns false • **isValidAccount(account_holder_name, account_number):** Checks that the given account name and number correspond to the same account in the database, returns true if they do, false otherwise
AccountConstants	Holds the constants for the maximum amount for transfer/paybill/withdrawal as well as the fees for student/non-student accounts	
Transaction	Holds all the information relevant to a single transaction, as well as a method that converts that transaction into a properly formatted string to print to the transaction file.	• **Transaction(transaction_name, account_name, account_number, amount, misc):** Constructor that stores the transaction name/type, account name, number, amount, and any misc information relating to a particular transaction in the respective member variables • **string to_string():** Converts the current Transaction object into the properly formatted transaction string that will eventually be written to
TransactionWriter	Takes all the Transaction objects and prints them to a	• **write(filename, session_transactions):** Takes in a list of Transaction objects pertaining to a given session, and writes

	file at the end of a session	all of them in the correct format to the provided filename
TransactionMapper	Contains a map from all the transactions to their two digit code.	
CommandValidator	It validates user input to ensure that they are commands in the system and do not break the Front End. Also checks to see that functionality is only accessed by correct type of account.	• **validate(curr_account, user_input):** Given an account and the user provided input by that same account, checks to see if the command is a valid command for that particular type of account
TransactionHander	Parent Class of all individual transaction handlers that will be implemented	• **handle(current_account, account_database, session_transactions):** A generic function that takes in an account, the current daily database, a list of the current transactions for that session, and uniquely handles them depending on the inherited version of the type of transaction. The account database is then updated to reflect the transaction.
LogoutHandler	Handles the action for when user uses the logout command	• **handle(current_account, account_database, session_transactions):** Adds a new logout transaction to the session transactions and saves them to a uniquely numbered transaction file in the appropriate format. The account database is then updated to reflect the transaction.
WithdrawalHandler	Handles the action for when user uses the withdrawal command	• **handle(current_account, account_database, session_transactions):** Handles the removal of money from the given bank account and adds a withdrawal transaction to the session_transactions once it completes successfully. The account database is then updated to reflect the transaction.

TransferHandler	Handles the action for when user uses the transfer command	• **handle(current_account, account_database, session_transactions):** Handles the transfer of money from one account to another. On success, adds two transactions to the session_transactions, one for the first account and another for the second. The account database is then updated to reflect the transaction.
PaybillHandler	Handles the action for when user uses the paybill command	• **handle(current_account, account_database, session_transactions):** Handles the transaction paybill whereby money is withdrawn from the provided account and added to one of three permitted companies. The account database is then updated to reflect the transaction.
DepositHandler	Handles the action for when user uses the deposit command	• **handle(current_account, account_database, session_transactions):** Handles the addition of money into the provided bank account. A transaction is created and added to the session_transactions on successful completion of the deposit and finally the account database is updated to reflect the deposit transaction.
CreateHandler	Handles the action for when user uses the create command	• **handle(current_account, account_database, session_transactions):** This handles the admin action of creating an account in which the admin account specifies the account holder name and initial amount of the new account. A transaction is then created and added to session_transactions.
DeleteHandler	Handles the action for when user uses the delete command	• **handle(current_account, account_database, session_transactions):** Handles the deleting of an account by ensuring that the name and number provided by an admin match that of an account in a database. On success, adds

		the transaction to the session transactions.
DisableHandler	Handles the action for when user uses the disable command	• **handle(current_account, account_database, session_transactions):** Handles the disabling of an account by ensuring that the name and number provided by an admin match that of an account in the database. On success, adds the transaction to the session_transactions.
ChangeplanHandler	Handles the action for when user uses the changeplan command	• **handle(current_account, account_database, session_transactions):** Handles the changing of plan on a given account by ensuring that the name and number provided by an admin match that of an account in the database. On success, adds the transaction to the session_transactions.
EnableHandler	Handles the action for when user uses the enable command	• **handle(current_account, account_database, session_transactions):** Handles the enabling of an account by ensuring that the name and number provided by an admin match that of an account in the database. On success, adds the transaction to the session_transactions.
LoginHandler	Handles the action for when user uses the login command	• **handle(current_account, account_database, session_transactions):** Handles the login in of a given user by verifying that the user selects a correct account type and if standard is selected they select a valid name from the database. On success, adds the transaction to the session_transactions.