1. Introduction

The project depicted is that of a ticket selling system for multiple users. The purpose of this phase is to establish our data from phase 1 in a more informative setting. The data we had depicted is now in UML diagram form to better outline the cause and effect of each of our operations one by one. We have set out our requirements of our most important functions and have associated complimentary functions which can be seen among our more informative UML diagrams and the code descriptions.

2. Design Requirements

Login → start a front-end session by logging in as a normal user or an admin

Logout → a back end operation that terminates the program and writes to a transaction file

Create → an admin privilege that allows for the creation of a new user account (admin, buy, sell)

Delete → an admin privilege that terminates an existing account other than its own

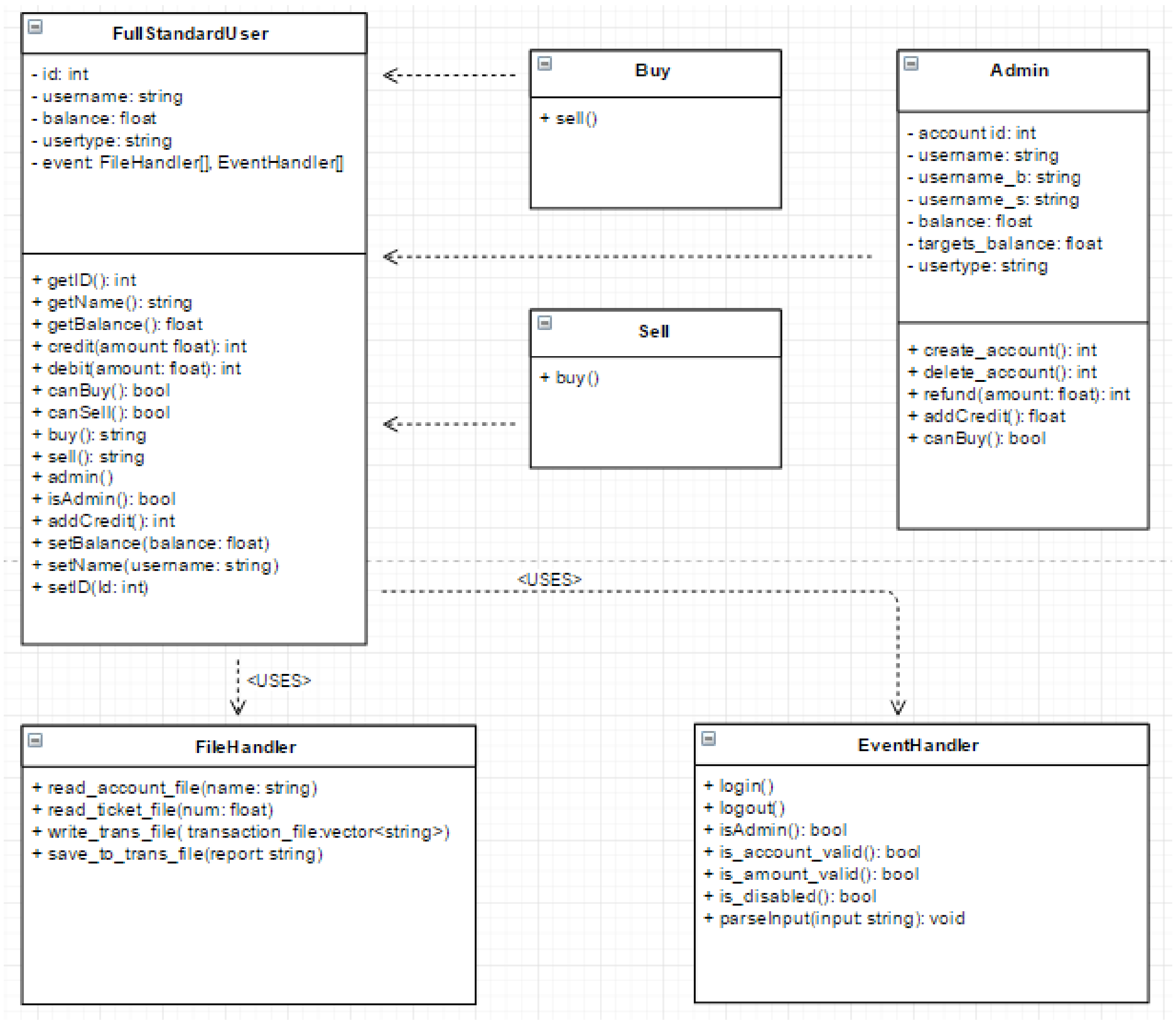
Buy → allows tickets to be acquired from one account and added to its own

Sell → allows tickets to be given to another account in exchange for currency/credit

AddCredit → allows credit to be added to one account

Refund → proposes that the transaction be undone and each user gets back their ticket/credit

3. External UML Diagram (draw.io)



4. Code design

Standard Class

This class contains the characteristic of a **FULL STANDARD USER**	
ATTRIBUTES	**ATTRIBUTE DESCRIPTION**
Id: int	Initializes the “Id” variable with the type: int

Username: string	Initializes the “username” variable with the type: string
Balance: float	Initializes the “balance” variable with the type: float
Usertype: string	Initializes the “usertype” variable with the type: string
Event: FileHandler[], EventHandler[]	corresponds the “event” variable with FileHandler events and EventHandler events
METHOD	**METHOD DESCRIPTION**
getID() : int	Return the value associated with ID
getName() : string	Return the value associated with username
getBalance() : float	Return the value associated with balance
Credit(amount float) : int	Adds the users new credited value
Debit(amount float) : int	Debits the users balance
canBuy() : bool	Signifies that this account has buying privileges
canSell() : bool	Signifies that this account has selling privileges
Buy()	Calls the buy sub class
Sell()	Calls the sell sub class
Admin()	Calls the admin sub class
isAdmin() : bool	Signifies that this account has admin privileges
addCredit() : int	Adds credit to the current users account
setBalance(balance: float)	Declares the balance of a particular user

setName(username: string)	Declares the username for a particular user
setID(Id: int)	Declares the ID for a particular user
This class contains the characteristic of a **BUY STANDARD USER**	
METHOD	**METHOD DESCRIPTION**
Sell()	Informs user that this is illegal action
This class contains the characteristic of a **SELL STANDARD USER**	
METHOD	**METHOD DESCRIPTION**
Buy()	Informs user that this is illegal action
This class contains the characteristic of an **ADMIN USER**	
ATTRIBUTES	**ATTRIBUTE DESCRIPTION**
AccountID(): Int	Initializes the “AccountID” variable with the type: int
Username(): String	Enter username of target account
Username_b(): string	Enter username of buying account
Username_s(): string	Enter username of selling account
Balance() : float	Enter the balance of credit a user has
Target_balance(): float	Signifies a user’s balance other than one’s own
Usertype() : string	Checks the user type of a particular account
METHOD	**METHOD DESCRIPTION**
Create_account() : int	Creates an account for a particular user with certain privileges

Delete_account() : int	Deletes an account for a particular user with certain privileges
refund(amount: float) : int	Refunds funds between a particular buyer and particular seller
addCredit() : float	Increment credit for any particular user
canBuy() : bool	Is able to buy but is not restricted in purchasing power
This class contains the characteristic of the **FILE HANDLER**	
METHOD	**METHOD DESCRIPTION**
Read_account_file(name: string)	After login, system reads the data from the account file
Read_ticket_file(num: float)	After login, system reads the data from the ticket file
Write_trans_file(transaction_file:vector<string>)	When called, it would write to the transaction file
Save_to_trans_file(report: string)	Appends data to the transaction file
This class contains the characteristic of the **EVENT HANDLER**	
METHOD	**METHOD DESCRIPTION**
login()	start a front-end session by logging in as a normal user or an admin
Logout()	operation that terminates the program and writes to a transaction file
isAdmin() : bool	Checks for admin privileges in case special privilieges must activated
Is_account_valid() : bool	Checks if account exists
Is_amount_valid() : bool	Checks if amount is within the bounds allowed

Is_disabled() : bool	Checks if account has been deleted by admin within past 24 hrs
parseInput(Input: string): void	takes user input and redirects it to one of the eight specific commands