AuctionBay

Project Phase Two

CSCI3060U, Software Quality Assurance

Miljanovic, M.

Ontario Tech University, Faculty of Science

February 17, 2023

Tuesday Lab (CRN: 72674):

Rija Baig (100746674)

Preet Panchal (100707094)

Eihab Syed (100707448)

Nathaniel Tai (100662284)

TABLE OF CONTENTS

Table of Contents 1

1.0. Introduction 2

1.1. Plan for Phase 3 2

2.0. UML Diagram 3

3.0. UML Method Descriptions 4

3.1. AuctionSystem 4

3.2. User 5

3.3. FileController 6

4.0. Instructions to Run Program 7

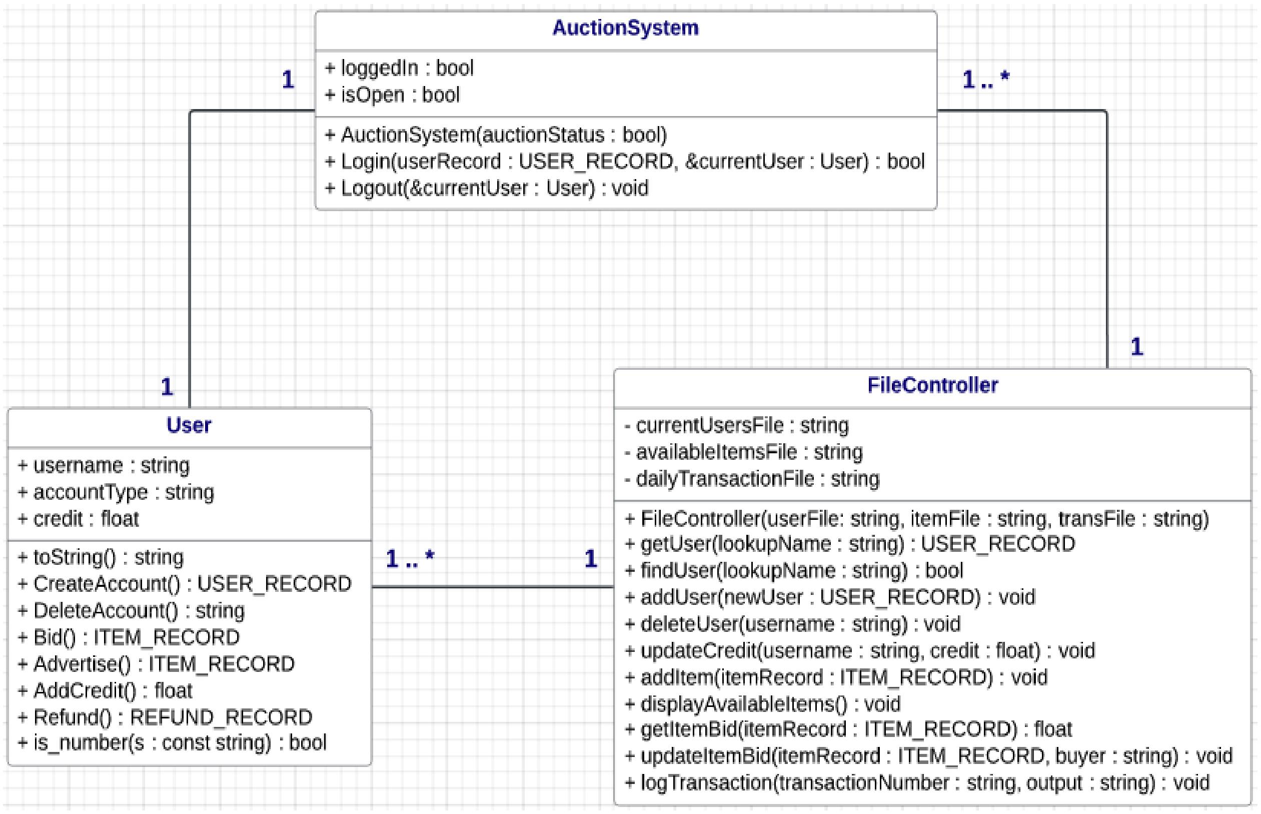
1.0. INTRODUCTION

AuctionBay is an Auction-style Sales Service. The system consists of two parts: the Front End, a point of purchase terminal for bidding and advertising items and the Back End, an overnight batch processor to maintain and update a master auctions file intended to run each day at midnight. This is Phase 2 of the project and it contains the design document for the overall structure of the frontend for AuctionBay. This design document will include a UML class diagram with class and method descriptions to explain each functionality. Phase 2 also represents the first version of our design implementation written in C++ source code.

1.1. PLAN FOR PHASE 3

- **Rapid Prototype Highlights:**
  - For Phase 2, we have built a complete rapid prototype for our AuctionBay program. We have a simple working console application that completes the basic intended functionality, however it is not ready for testing as only a few constraints were solved.
- **Missing Components:**
  - In this phase, as this is a rapid prototype for our frontend, there will be a few errors that are not handled properly and some operations such as ‘bid’, ‘refund’, and ‘addcredit’ do not function as intended.
- **Next Steps:**
  - For Phase 3, our main goal is to have all constraints for each operation solved and be able to handle all errors from our test cases in Phase 1. We also need to make sure that all formats for the frontend console application matches with what we have in these test cases as well as match the client requirements for recording transactions. Lastly, everything must be organized a little better, so we can ensure a smooth testing experience in future Phases.

2.0. UML DIAGRAM



AuctionSystem
+ loggedIn : bool
+ isOpen : bool
+ AuctionSystem(auctionStatus : bool)
+ Login(userRecord : USER_RECORD, &currentUser : User) : bool
+ Logout(&currentUser : User) : void
1
1 .. *
1
1
User
+ username : string
+ accountType : string
+ credit : float
+ toString() : string
+ CreateAccount() : USER_RECORD
+ DeleteAccount() : string
+ Bid() : ITEM_RECORD
+ Advertise() : ITEM_RECORD
+ AddCredit() : float
+ Refund() : REFUND_RECORD
+ is_number(s : const string) : bool
1 .. *
1
FileController
- currentUsersFile : string
- availableItemsFile : string
- dailyTransactionFile : string
+ FileController(userFile: string, itemFile : string, transFile : string)
+ getUser(lookupName : string) : USER_RECORD
+ findUser(lookupName : string) : bool
+ addUser(newUser : USER_RECORD) : void
+ deleteUser(username : string) : void
+ updateCredit(username : string, credit : float) : void
+ addItem(itemRecord : ITEM_RECORD) : void
+ displayAvailableItems() : void
+ getItemBid(itemRecord : ITEM_RECORD) : float
+ updateItemBid(itemRecord : ITEM_RECORD, buyer : string) : void
+ logTransaction(transactionNumber : string, output : string) : void

3.0. UML METHOD DESCRIPTIONS

3.1. AuctionSystem

The AuctionSystem Class is used to determine menu views and is the controller in the application that allows the user to perform transactions while ensuring proper authorization.

- **+ AuctionSystem(auctionStatus : bool)**
    - This function is the constructor for the class AuctionSystem which initializes the auctionStatus in order to control the run of the program.
- **+ Login(userRecord : USER_RECORD, &currentUser : User) : bool**
    - This function assigns values to a structure named USER_RECORD by taking the current user information retrieved from the login and returns user login status.
- **+ Logout(&currentUser : User) : void**
    - This function logs the current user out of the program and returns a success message to the console.

3.2. User

The User Class is an object that's used when a user is logged in and informs the user what transactions they can perform in the system.

- **+ toString() : string**
  - This function returns the user information in a string format.
- **+ CreateAccount() : USER_RECORD**
  - This function allows an admin user to create another user object and handles errors with user input during the process of user creation.
- **+ DeleteAccount() : string**
  - This function allows an admin to delete another user object and their items in the auction system.
- **+ Bid() : ITEM_RECORD**
  - This function creates an ITEM_RECORD that can be used to verify the users bid attempt and allows the user to bid on an item that is active in the auction system.
- **+ Advertise() : ITEM_RECORD**
  - This function allows a user to advertise an item in the auction system and handles errors with user input during the process of item advertising.
- **+ AddCredit() : float**
  - This function allows an admin user to add credits to any user or any non-admin users to add credits to their own user accounts.
- **+ Refund() : REFUND_RECORD**
  - This function gives the user their credit back from a seller's account and handles errors with the user input during the process of refunding.
- **+ is_number(s : const string) : bool**
  - This function is used in error handling to verify user input regarding credits is of the proper type.

3.3. FileController

The FileController class is used for file Input and Output controls for text documents.

- **+ FileController(userFile: string, itemFile : string, transFile : string)**
    - This function is the constructor for the class FileController which initializes a user-accounts file, available-items file and daily-transactions file in order to control the reading and writing to these files.
- **+ getUser(lookupName : string) : USER_RECORD**
    - This function looks up a user from the user-account file and returns a USER_RECORD of the user.
- **+ findUser(lookupName : string) : bool**
    - This function searches the user-accounts file and returns true if user found.
- **+ addUser(newUser : USER_RECORD) : void**
    - This function adds a USER_RECORD of the user to the user-accounts file.
- **+ deleteUser(username : string) : void**
    - This function looks up the user to delete from the user-accounts file and removes it from the user-accounts file if it exists.
- **+ updateCredit(username : string, credit : float) : void**
    - This function updates the credit balance of the user based on their provided username and current credit balance.
- **+ addItem(itemRecord : ITEM_RECORD) : void**
    - This function adds an ITEM_RECORD of the item to the available-items file when the user inputs the 'advertise' operation.
- **+ displayAvailableItems() : void**
    - This function displays all of the existing items that are in the available-items file when the user inputs the 'listall' operation.
- **+ getItemBid(itemRecord : ITEM_RECORD) : float**
    - This function returns the current highest bid on an item active in the auction by checking the available-items file. It returns -1 if the item record is not found in the available-items file for the item requested to be bidded.
- **+ updateItemBid(itemRecord : ITEM_RECORD, buyer : string) : void**
    - This function updates the current highest bid amount on an item active in the auction by checking the available-items file and validating the buyer.
- **+ logTransaction(transactionNumber : string, output : string) : void**
    - This function writes a transaction to the daily-transaction file using the transaction number based on the operation and the recorded information from that transaction.

4.0. INSTRUCTIONS TO RUN PROGRAM

1) Open a new terminal window.
2) Go to folder: `cd AuctionBay/PHASE-2/src`
3) Run the makefile: `make`
4) Run:

```
./auction_system ./iofiles/current_users.txt
./iofiles/available_items.txt ./iofiles/daily_transaction.txt
```