**Final Project Documentation**

Nikhil Dhoka, CSCI 24000

**Project Title**

ArtHouse: An Object-Oriented Auction House System in Java

**Longer Description**

ArtHouse is a command-line program that showcases an Auction House by deploying skills in concepts of Object-Oriented Programming such as aggregation, encapsulation, abstraction, interfaces, polymorphism, Linked Lists, Array Lists, etc. I also plan to include features such as data persistence, login system, implement client-side and server-side programming to simulate an application in the real world, etc.

**Intended User**

The intended users of this program are auction houses and customers of these houses that are involved in activities that involve the selling of antique, limited edition, or exotic items such as rare art pieces, paintings, sculptures, technology, etc. This program can also be used by government agencies that often auction seized items to get money out of them. Lastly, this could also serve as a platform that banks can use when they want to auction items that were kept as collateral while giving out loans to customers who weren’t able to service the loans.

**Problem to be Solved**

Auction houses often have activities that are done offline. Using this program, the processes of acquiring, inventory tracking, and tracking customer accounts, payments, etc. can be automated thereby adapting to this era of automation. To solve this problem, I plan to include features such as a multiuser program allowing multiple users to log in at the same time thereby simulation an actual environment of the auction. Additionally, I also plan to include two lists of inventories for the auction house. One list contains verified items that are ready to be auctioned and another list of items that is yet to be verified. The admin of the auction house shall have the power to change the status of these items. This will again translate real-world scenarios into the program.

**Technologies to be Used**

1. Java Programming Language

2. Makefile

3. Dat file for data persistence using object serialization

4. Java web Sockets to set up multiple user access to the same Java program allowing a user to log in from their respective machines and place their bids for the auction item

5. UML for Code Design

6. Git for Version Control

7. GitHub to store the repo

**Use Case Analysis**

The Auction House shall be the center of the program that will have all the inventory of the verified and unverified items. Additionally, it will also contain accounts of all the users that could take part in the auction of the various items that the auction house possesses. The power to change any data in the Auction House shall solely rest with the admin that will have his/her own unique id and password. The auction house will also have a money account wherein money would be kept when an item is successfully sold to a user.

Additionally, another major part of the program is the customer's program which shall have its own interface wherein they would be able to add money to their account, withdraw money from their account in case they no longer want to buy anything, view all their info, and their inventory. Each user will get to set their pin and shall be assigned account numbers by the admin. The user will also have their own inventory when they successfully win an auction, and the item is transferred from the auction house to the user.

Lastly, as mentioned earlier, I also intend to have multiuser interactions with the program to replicate an auction house wherein multiple users can place bets at the same time in real-time.

**Data Design**

Data in the program –

1. Customer Data such as the Name, DOB, Account No., PIN, No. of Auction wins, No. of Auctions Attended, List of Inventory, etc.

2. Auction House Data such as verified inventory, unverified inventory, money in the auction house, admin of the auction house, and customers of the auction house.

3. Admin of the auction house that has access to add users, manage the inventory of the auction house, money of the auction house, delete a user, etc.

4. I have a class for the auction house that has the list of all the users, admin, inventories, money, etc. of the auction house.

5. I also have a class for customers that has all the functionality that the customer can perform.

6. I also have a class for the auction that implements a timer and the Timer Task class to simulate an actual auction that resets the timer when a player places a bid.

7. I also have a leaderboard class that takes care of converting the customer and auction history data into a leaderboard to get the contenders and the auctions that exist or took place in the auction house.

8. Lastly, I also have an admin that has the power to manipulate the customer and auction house data as the need may be.

**Data that needs persistence** – Customer Data, Auction House Data, Admin Data, Auction History, etc.

**Data that needs to be aggregated:**

1. I have Arraylists for the customers, inventories, admins, etc.
2. I have data from the users and the auction history that is stored in Hand built generic Linked Lists that can be made with any data type. This data is used to create the leaderboard for the auction.

**UI Design**

Customer Interface using Web Sockets demonstrating Network Programming and implementing a Client and a Server Side (Command Line):

Please enter your Account Number:

Please enter your Pin:

This step will send this data to the server, which will find a user with the respective credentials to send back the customer object to the client. The client then can change the information according to their will. In the end, the client sends back the customer object to the server which updates it in its record.

1. Quit
2. Customer Menu
   1. Quit
   2. View all information
   3. Deposit Money
   4. Withdraw Money
   5. View Inventory

4. Menu for the Admin

1. Quit
2. List Customers
3. List Un Verified Inventory
4. List Verified Inventory
5. List Auction History
6. Add a Customer
7. Delete a Customer
8. Verify Items
9. Un Verify Items
10. Get Auction Account Balance
11. Wihtdraw from Auction Account
12. Deposit Auction Account
13. Add a New Item

5. Menu for the LeaderBoard

1. Exit
2. Customer Leaderboard by Auction Wins
3. Customer Leaderboard by Auctions Attended
4. Auction Leaderboard by Highest Winning Prize
5. Auction Leaderboard by Highest Percentage Increase

**Instructions to Login as a Customer**

1. Run the command **make run** to run the auction house(server) side of the program.
2. On the same machine open another terminal and type run client to run the client side of the program.
3. Enter the credentials of the customer to access all the customer data using the customer's menu.

**Algorithm**

1. User.java
   1. Goal – To serve as a blueprint for the Customer and the Admin class.
   2. Steps:
      * + 1. To make the abstract class as it is never going to be initialized.
          2. To declare variables such as name account Number, pin, and the name of the user
          3. To have getters for each of them.
          4. To make an abstract method called menu that shall be declared in the Customer and the Admin class.
2. Customer.java
   1. Goal – To define all the functionalities of a customer in the auction house.
   2. Steps –
      1. The class build upon the User class by adding more data member such as the account of the customer, the auction Wins, the No. of Auctions Attended, and the birthday of the customer.
      2. We define the constructor of the customer that takes user input to create a new customer to initialize all the data members.
      3. The customer constructor takes in the account number as a parameter. This will be fed by the admin when they create a new customer to make sure that there is always a new account number that is being assigned to the customer.
      4. The customer also has an ArrayList of the type of Item that contains all the items of the customer that they have won in auctions
      5. We now make functions to increase auction wins, and increase auctions attended.
      6. We now make getters for the above functions.
      7. We now make getter for the birthday.
      8. We now function like the add Inventory, remove inventory, and list Inventory to add an item to the customer’s inventory, remove items from the customer's inventory, and list all the items in the customer's inventory respectively.
      9. Make the viewALL method that prints out all the customer’s information.
      10. Make the viewInfoLeaderBoard function that prints out restricted data.
      11. Define the menu function that was declared as an abstract method in the User class by creating a menu for the customer to have all functionalities that the customer can engage with.
3. Account.java
   1. Goal – To define an account like a bank account that can hold money and carry out functionalities such as deposit and withdrawal.
   2. Steps –
      1. Make the account serializable so that we can save all the account info using object serializable when I have the account as a data member in the Customer and the Auction House
      2. Declare the balance of the account of the type of double.
      3. Make a getter for the account.
      4. Make a withdraw Balance method with the return type Boolean to so that the function can return false if the user is trying to withdraw more than the available balance.
      5. Make the deposit Balance function that lets us deposit money into the account.
4. Auction.java
   1. Goal – To facilitate an auction of an item wherein two customers can participate to place bids and claim to win the item.
   2. Steps -
      1. Declare variables such as the time left, currentBid, the item, the customers that are participating, the winner of the auction, a Boolean that tells if a bid was placed or not, and an int named percentage increase that tells the percentage increase between the starting bid and the winning bid.
      2. We make the Timer object transient as we do not want to save the info on that as it is not serializable.
      3. We make the constructor of the Auction that takes in the two customers that will participate and the item that needs to be auctioned as parameters.
      4. The constructor then starts the bidding process by having a timer in the command line interface. The timer resets when a legitimate bid is placed by the customer.
      5. If no bids are placed, then the program stops and marks the item as unsold.
      6. The timer is implemented in the program using the Timer Task and the Timer class in the Java util package.
      7. To aid the timer, I created functions that help in determining various things.
      8. First, I created a function named place bid that returns a Boolean showcasing if the bid placed was successful or not. The place then checks if the bid was greater than the previous one and makes sure that the bid is not less than the customer’s current account balance. If these two conditions are met, then the bid is placed successfully and then the timer resets, and the next customer is asked to place their next bid.
      9. Second, I created the run method for the TimerTask class that would specify as to how the timer task needs to run and when it needs to stop.
      10. I also made getters for data members of the Auction class.
      11. Additionally, I also created a method called start, which starts the auction and by starting the timer by feeding in an instance of the timer task class as a parameter.
      12. Lastly, I also created a method called endAuction that is called from the constructor when we reach the end of the auction.
5. ItemCategory.java
   1. Goal – To make an enum that lists out all the categories of the items.
   2. Values – ELECTRONICS, FURNITURE, ART, COLLECTIBLES, FASHION, OTHER
6. ItemStatus.java
   1. Goal – To make an enum that states the status values of an item that could be either VERIFIED or NOT VERIFIED
7. Item.java
   1. Goal – To make the Item class that can store the various items in the auction house that would be auctioned to the customers.
   2. Steps –
      1. Declare the data members such as the name, itemID, the start Bid, the category that would be of the type ItemCategory, and the status that would be of the type ItemStatus
      2. Make the constructor that initializes all the variables by taking user input.
      3. The itemID is taken in as a parameter that would be passed in by the admin which through a process makes sure that the itemID of an item is always unique.
      4. We make getters for all the data members
      5. We make the selectEnumItemCategory method that lists all the categories of items and then takes user input to set that type to the respective item.
      6. We make the changeItemStatus method that allows the admin to change the status of an item from verified to not verified and vice versa.
      7. Lastly, we make the viewAll that basically prints out all the information of that item
8. Admin.java
   1. Goal – To define all functionalities for the Admin of the Auction House
   2. Steps -
      1. We first extend the Admin from the User abstract class.
      2. We then declare the data members of the admin which are the Array List of customers, Array List of unverified inventory, Array List of verified inventory, Array List of the auction history, and the auction house’s bank account with the money.
      3. We now define the Admin constructor that takes in the customerArr, unverified inventory, verified inventory, auction History, auction account, and the account number as a parameter to initialize all the data members.
      4. We now make the delete Customer method that takes the admin’s input to delete a customer using their account number.
      5. We also make the list customer method that prints out all the customer info in an aesthetic manner.
      6. We now make the add customer method that calls on the customer constructor and then creates the customer using the admin’s input and then adds it to an array list of customers.
      7. We now make the verified Item and the unverified item methods that take the admin’s input for the item that they want to verify or unverified. The methods then prompt for y/n response to confirm the admin’s action. Upon confirmation, the item is moved from the verified inventory to the unverified inventory or vice versa depending on the situation. The item status is also updated accordingly.
      8. We now make the get Auction account balance, withdraw auction account, and deposit auction account methods that handle transactions that the admin wants to make with the auction bank account.
      9. We now make methods that list the verified, unverified, and auction history methods so that they can be called whenever required before a transaction needs to be done. These methods loop through the respective array lists and list all the information about the respective item or auction.
      10. We now make the getNewItemID method that checks the item ID’s that have been previously assigned and then based on that gives a new and unique item id for an item that needs to be created. This method is used to add a new unverified item to the auction’s inventory.
      11. The method first checks if the unverified inventory, the verified inventory, or the auction history array is empty.
          1. If any of these arrays are empty, it checks for the highest assigned itemID from the arrays that are not empty.
          2. If all the arrays are empty, it assigns the starting itemID of 1000.
      12. IF not the first situation, which means that all the arrays have at least 1 element In them, the method then compares the items of the last elements of these arrays to find the largest item id ever assigned. After that, it just adds 1 to assign a new and unique item id each time.
      13. We now make the start Auction method that takes in basic information about the item that the admins want to start an auction for. Next, the method requires the customers to login into the auction to proceed further. Once this is done, the method creates a new instance of the auction class to carry out the proceedings of the auction. Once the auction is done, the method increases the auction wins of the winner, auctions attended by all the participants, and debits money from the winner’s bank account and then adds that to the auction’s bank account. This method also keeps a track of the auction history by adding all the data of the auction to the auction history array.
      14. Finally, we create a menu function for the admin as this method is an abstract method declared in the User class.
      15. The method basically prints out all the options to the admin to carry out respective tasks.
9. Client.java
   1. Goal – To deploy the client side of the customer login system for the auction house using web sockets in Java.
   2. Steps –
      1. We start by establishing a connection with the server that starts listening for connections when we select the login as a customer option on the server side.
      2. Next, we take the user’s input for their account number and pin. Then, we send the data to the server using streams.
      3. If the user exists in the server’s data, then it sends the object of that object back to the client which can now use the menu to perform all the actions on their account such as deposit/withdraw money, view all their information, and view their inventory.
      4. Lastly, when the user is done, we send the updated customer object back to the server to be updated in the server’s data.
10. AuctionHouse.java
    1. Goal – To make a user interaction for the customer and the admin to interact with
    2. Steps –
       1. We make the data members for the Auction House that include the Customer Array, the Verified Inventory, the Unverified Inventory, the Auction History, and the Auction’s Account.
       2. We Now make load and write functions for all the arrays and data members for data persistence using object serialization.
       3. We now make the constructor of the Auction House that begins with loading all the data that was stored previously.
       4. Then the user is prompted with a menu with options to quit, log in as an admin, and log in as a customer.
       5. Once the user enters valid credentials for either the admin or the customer, they are prompted with the respective menu’s.
       6. The only difference between the two is that the customer login is setup using web sockets, so the user would have to run the client.java in another terminal (for demonstration purposes in this project. Can set it up for an IP address so that the user can log in remotely). Once the customer logs in with valid credentials, they can carry out their respective transactions. Once they are done the customer object is sent back to the server to be updated on the server side.
       7. When the user logs in as an admin, they are prompted to enter their account number and pin. IF the user inputted credentials match (Account num = 12345 & Pin = 1234) then they are prompted with the admin’s menu.
       8. The user also has the option to view the leaderboard.
       9. When the user chooses to view the leaderboard, they are prompted with the menu for the leaderboard.
       10. Lastly, when the user is done with all their desired actions and quits the program, all the data is stored back into .dat files for data persistence.
11. Node.java
    1. Goal – To define every single node for a linked list that can be of any data type.
    2. Steps –
       1. We make declare the data members for the node class that are the data and a pointer to the next node.
       2. We now make the Node constructor that initializes the data of the node and sets the next element to null. This is used to append a node to a linked list.
       3. We make getters and setters for the data and the next variables.
12. GenericLL.java
    1. Goal – To integrate all the nodes that will be created into a linked list
    2. Steps –
       1. The linked list keeps a track of only the head node and then all the next elements shall be accessed using the head.
       2. The head and the size variables are declared as the data members for the linked list.
       3. We now make the GenericLL constructor that basically sets the head to null.
       4. We now make the add method that lets you append an object of the same type as the linked list to the end of the Linked List.
       5. Similarly, we also make the remove, count, traverse, get, size, and get head methods to assist us with operations on the linked list.
       6. We now make the swap method that takes in the nodes that need to be swapped and simply interchanges the data of the two nodes.
13. Leaderboard.java
    1. Goal – To make a leaderboard for the customers and the past auctions for the Auction House
    2. Steps –
       1. We start by declaring the data members which are the linked lists of the customers and the auction history.
       2. We now make the constructor by taking in the array lists of the customer and the auction history as parameters and then copying all the data to the Generic linked lists instances of the customers and the auction history.
       3. Now we make the sortCustomerWins method that sorts the customer array by having the customer with the most no of wins at the start of the linked list using the swap method created in the GenericLL class. This is done using the bubble sort algorithm.
       4. Now we make the sortAuctionsAttended method that sorts the customer array by having the customer with the most no of auctions attended at the start of the linked list using the swap method created in the GenericLL class. This is done using the bubble sort algorithm.
       5. We now make the sortHighest bid method for the auctions by sorting the auction history linked list in a way so that we can print the auction with the highest winning bid at the top.
       6. Lastly, we make the sortHighestPercentageIncrease method using for the auctions wherein the auction with the highest percentage increase with respective to their starting bid is at the start of the linked list thereby allowing us to print the auctions in decreasing order of their percentage increase.
       7. Finally, we make the menu method that allows the users to avail the various leaderboards based on their input.