Inf2C Software Engineering 2018-19

Coursework 3

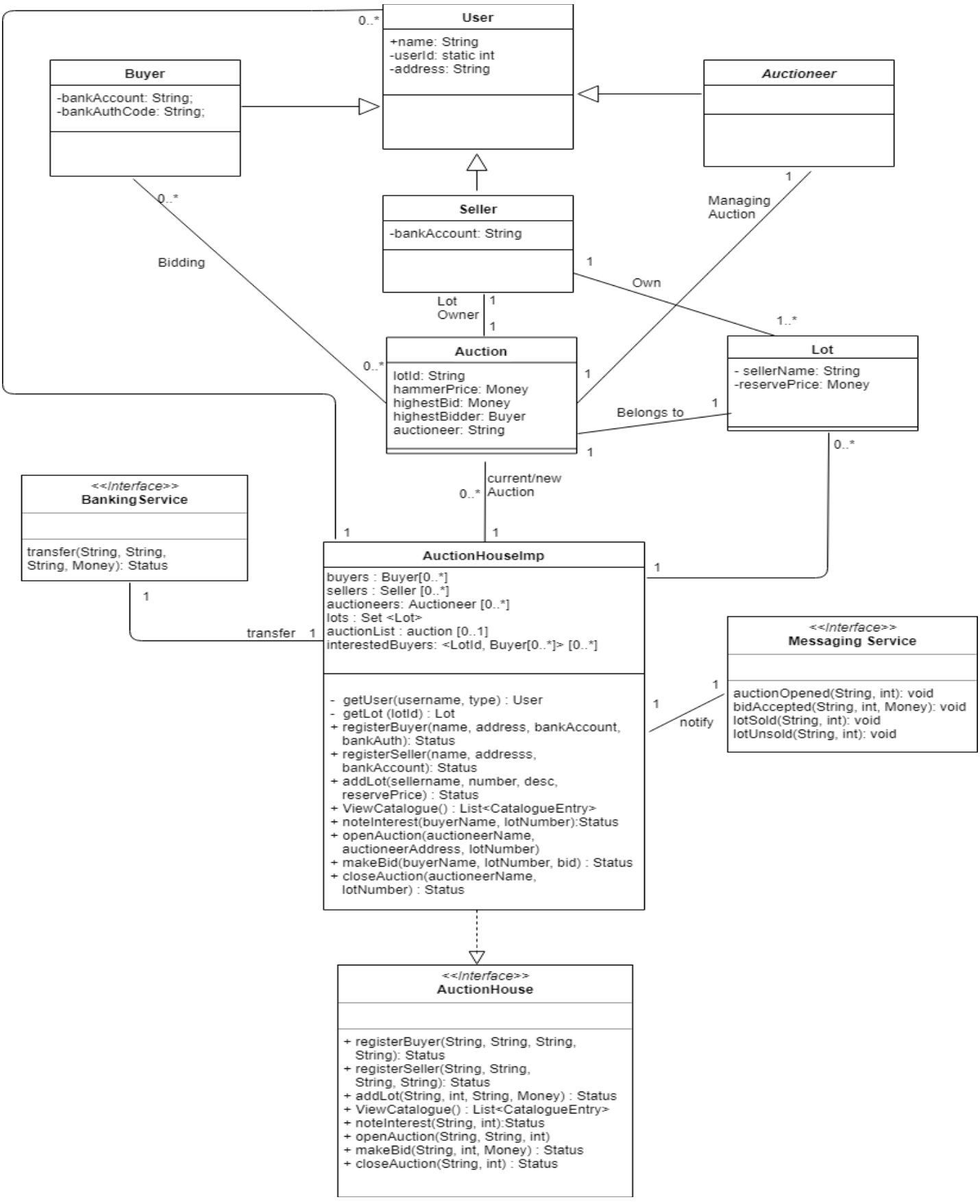
Creating an abstract implementation of a auction house system

REPORT

Billy Byiringiro s1794094

Alan Savio Paul s1768177

UML Class Diagram:



High-level design description:

We have an AuctionHouseImp class (singleton object) as a portal into the system and to have the operations handling input messages mostly just forward on those messages to suitable other classes. This avoids the AuctionHouseImp class becoming very large and having poor cohesion. We have learnt that this use of an AuctionHouseImp class is an example of the Facade design pattern 2. However, because we were given AuctionHouseImp as a controller class of all messages across the system, we had to ensure the cohesion of our implementation by putting the relevant methods and attributes in the classes in which they are most relevant and hence less coupling.

For example, we had to create an Auction class to hold information related to a particular auction, and auctionList to keep track of all auctions and their status. Auction had to keep information only related to it. Instead to storing seller name to Auction class, we stored a lot associated to that auction, and we could retrieve seller name by through the lot, as the seller is the owner of that particular lot.

Because we need to keep track of all buyers who noted interest and who bid on a lot, we decided to add them in the same HashMap mapping users to lot's number for two reasons: First, we realized that the only operation to be acted upon those users is to send a notification when someone bids or when the auction is closed. Secondly, and most importantly, it is more efficient because when we want to know the user to send a message we can use lotNumber in the HashMap to know which users to notify directly.

We previously had username and password to include authentication details in the User class. But, in coursework 3 we realized this should not be given focus. We had most methods split between the different Users and AuctionHouseImp, but now we've added most of them to the controller class, i.e. the AuctionHouseImp class. In the Lot class, we had a function called SaveLot() that would save the lot. However, that would only be useful in front-end design of our application, so we realised this was unnecessary as we would just add the lot to the list of lots when addLot is called.

Implementation Decisions:

We realized that we will need to validate User (buyer, seller, and auctioneer) many times, so we decided that implementing a helper function getUser(user, type) : User that receive a username as primary key and type of user would be handy for not only validating but also tells us that a user doesn't exist when it returns null, otherwise it returns an object of that user on which we can find more information while also reducing of number of codes. The same case applies to getLot(lotId) : Lot.

Also as we mentioned in the CW2, we found that having three types of user classes is important and having the list of them as normally we would be fetching them from databases.
In the auction class, we added a hashmap called bidAndBidder which is a hashmap that stores all the bidders and their bids. This would be useful in an actual implementation of an AuctionHouse app so every user can see their last bid for a lot. We also added a variable that saves the highest bidder as a Buyer object and a variable that saves the bid of the highest bidder. This would make the code more efficient as it removes the need to search through the hashmap and compare values to find the highest bid and bidder. These variables are used repeatedly to know who wins the auction and also during payment processes.

We also added lots in a collection SortedSet, which automatically avoid duplicates of lots in our system, but also provided convenience in the implementation of viewCatlogue method as the items as are sorted as they are added in that collection.