Inf2C Software Engineering 2018-19 Coursework 2

Creating a software design for an auction house system

Billy Byiringiro s1794094

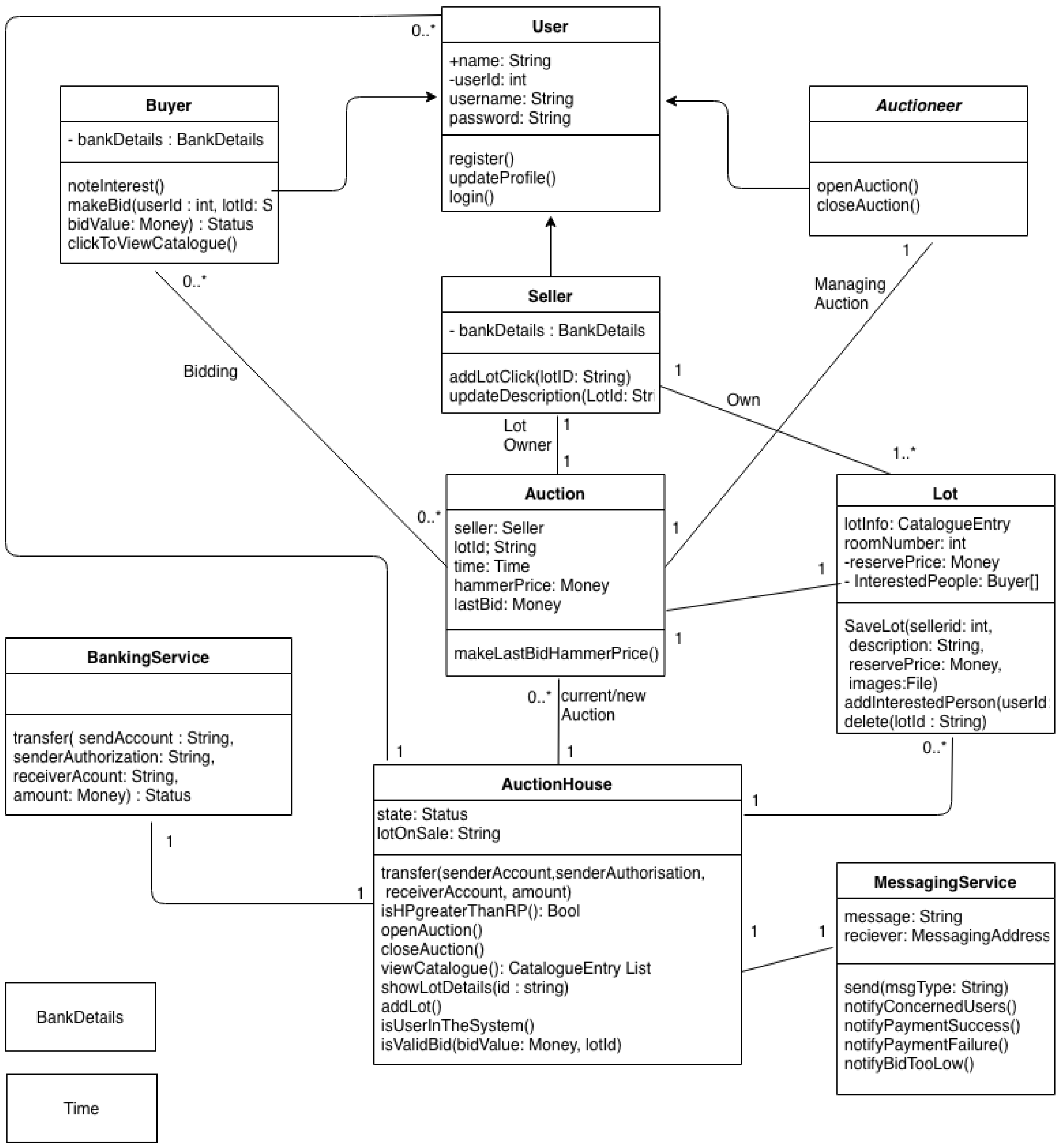
Alan Savio Paul s1768177

2.1

The Auction House System connects sellers’ lots to various buyers and allows auctioneers to manage the auction processes. Additionally, it handles the payment transferring from buyers to sellers and charges the buyer’s premium and adds the seller’s commission.

2.2

2.2.1 UML class model



2.2.2. High-level description

Our design includes an abstract User class which we separated into Buyer, Seller, and Auctioneer classes because they have unique attributes. We decided to not include a class called Member of Public because we assumed that every user who is not a seller or auctioneer, is a buyer (who isn't registered yet). And that will be addressed in the implementation of the classes. Where user who had logged into the system, will be allowed to do certain actions depending on their privilege/access in the system. While by default, and similarly for the general members, everyone will be allowed to browse through the catalogue.

We also assumed that Bank Details has different information for buyers and sellers. This is because purchasing requires money to be transferred **out**, which requires details like card number, account number, CVC number etc. whereas being a seller involves only receiving money i.e. money is transferred **into** their account. This only requires a number account number and sort code. However, we did not want to clutter the UML diagram with this detail. Thus, this should be taken care of in at the implementation stage.
We also decided to have the bank details to be private (-) access because it need not be shared with all other classes apart from Banking Service.
We created an AuctionHouse 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 AuctionHouse class becoming very large and having poor cohesion. We have learnt that this use of an AuctionHouse class is an example of the Facade design pattern 2.

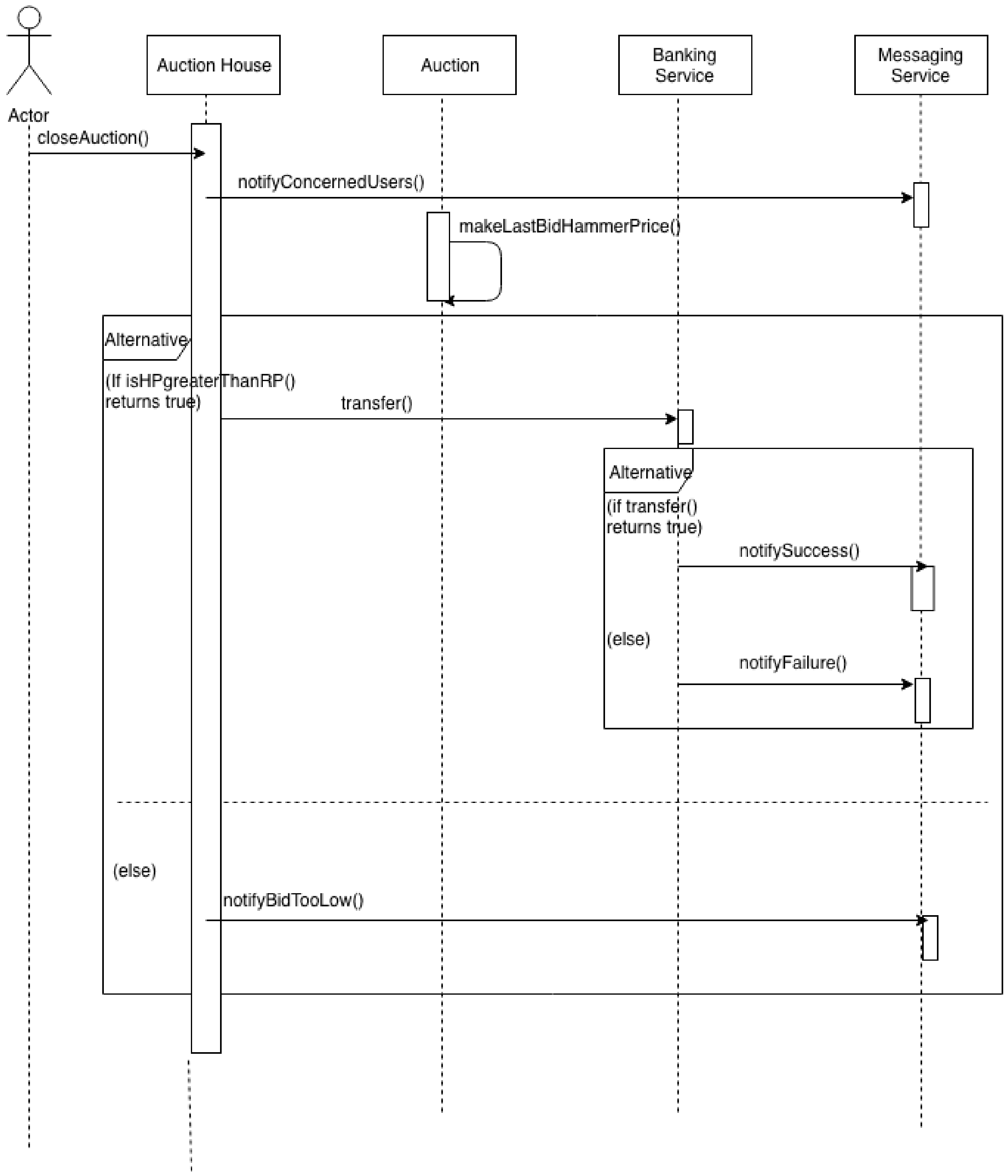
We assumed members of public will be handled by the user class. And the attributes of the user class will be null, as the members of the public can use the system without necessarily requiring to register or login into the system. In this case, the class user acts as the parent of the buyer, seller, and auctioneer class where the member of public needs to register and to login into the system to be classified into those categories.

In the quest of trying to keep our system cohesion high and coupling less, we tried to make sure that all system at least will be controlled by the AuctionHouse class, but each class is self-contained on particular functions related to what it does. That means AuctionHouse will still access their object to access their operation but not necessarily be concerned about their implementation.

We thought about making one operation in MessagingService that will handle a different kind of messages when given different parameters but we realized that it should be more descriptive at UML reader to mention them as distinct operations but it can be changed at the implementation level.

2.3 Dynamic models

2.3.1 UML sequence diagram



2.3.2 Behaviour descriptions

1. **Add lot**

   Seller → addLotClick(lotID) → AuctionHouse

   AuctionHouse → SaveLot(sellerid, description, reservePrice, images) → Lot

2. **Note Interest**

   Buyer → clickToViewCatalogue() → AuctionHouse

   AuctionHouse → viewCatalogue() → Buyer

   Buyer → noteInterest() → Lot

   Lot → addInterestedPerson(userId) → Lot

3. **Make bid**

   Buyer → MakeBid(userId, lotId, bidValue) → AuctionHouse

   IF [isValidBid(bidValue, lotId)]

   Auction→ makeLastBidHammerPrice() →AuctionHouse

   AuctionHouse → notifyConcernedUsers(lotId) → MessagingService