Inf2C Software Engineering 2017-18

Coursework 2

Creating a software design for an auction house system

s1891130	**Li Kaiyu**
s1891075	**Tai Yintao**

1.Introduction

The main idea of this system is when users, include buyers, sellers and auctioneers need to implement a use case, they only need to call the corresponding function in the singleton AuctionHouse class, and transfer the necessary information with parameters. Then the corresponding function in AuctionHouse will sequentially execute the specific codes and interact with other objects and classes, after complete the use case, return relative information included in status to the caller. Since the system will run in a single-thread nature, we assume that all use cases will be executed in a short time, and each request from users will be processed separately at high speed. In this ideal situation, users will not notice the delay.

2.Static Model

2.1 UML Class Model

See figure 1 on page 3 and the figure 2 on page 4.

2.2 High-level Description

- Modelling using message bursts: This system is designed with a single thread of control. Assume that all call messages(what we call inputMessage in the class diagram) are automatically stored in a buffer.In each time of the main loop, the system read all messages in the buffer and send them to theAuctionHouse object one by one until every call message is executed.
- InputMessage: Assume that every input message is a string which contains a specfic number that indicates correspond use case execution and also all parameters needed to provide.
- AuctionHouse class(singleton class): The AuctionHouse class is the most important class which is responsible for executing the scenarios triggered by each input message. First, switchInput operation is invoked to identify which use case should be executed as well as get all parameters needed from the input message. Then, correspond operation of the objects of relative class is invoked to execute the use case, if the use case is not executed successfully, error will be reported.
- MessageService: MessageService class is designed as a public class which means the objects of other class are able to invoke sendMessage operation to send reply messages to users.
- Account: Different from the instructions, we decide to design account as a type of class rather than string due to its confidentiality.
- Auction and lots: There are basically two ways to design the relationship between auction and lots- one to one and one to many. We decide to apply the latter to avoid more complicated system structure. Thus, Auction is a singleton class which holds all lots in the auction.

3 Dynamic Models

3.1 UML Sequence Diagram

The close-auction use case sequence is shown in figure 3 on page 5.

- The Auction class will initialize one object to handle all lots transaction. It has a list stores all ongoing lots objects, and they are instantiated objectives, include all bids record and relative buyers and sellers. Thus the auctioneer only needs to transfer the id of the lot could find the necessary information.
- The *sendCloseNotification* function will depend on the value of *isValid* to generate different message.
- The functions, such as getAccount, getBuyerAddress, getTheSellerAddress and getTheHighestBuyerAccount, without return value are not real method, they are directly accessing the objects' attribution.

3.2 Behavior Descriptions

Se Behavior Description in figure 4 on page 6.

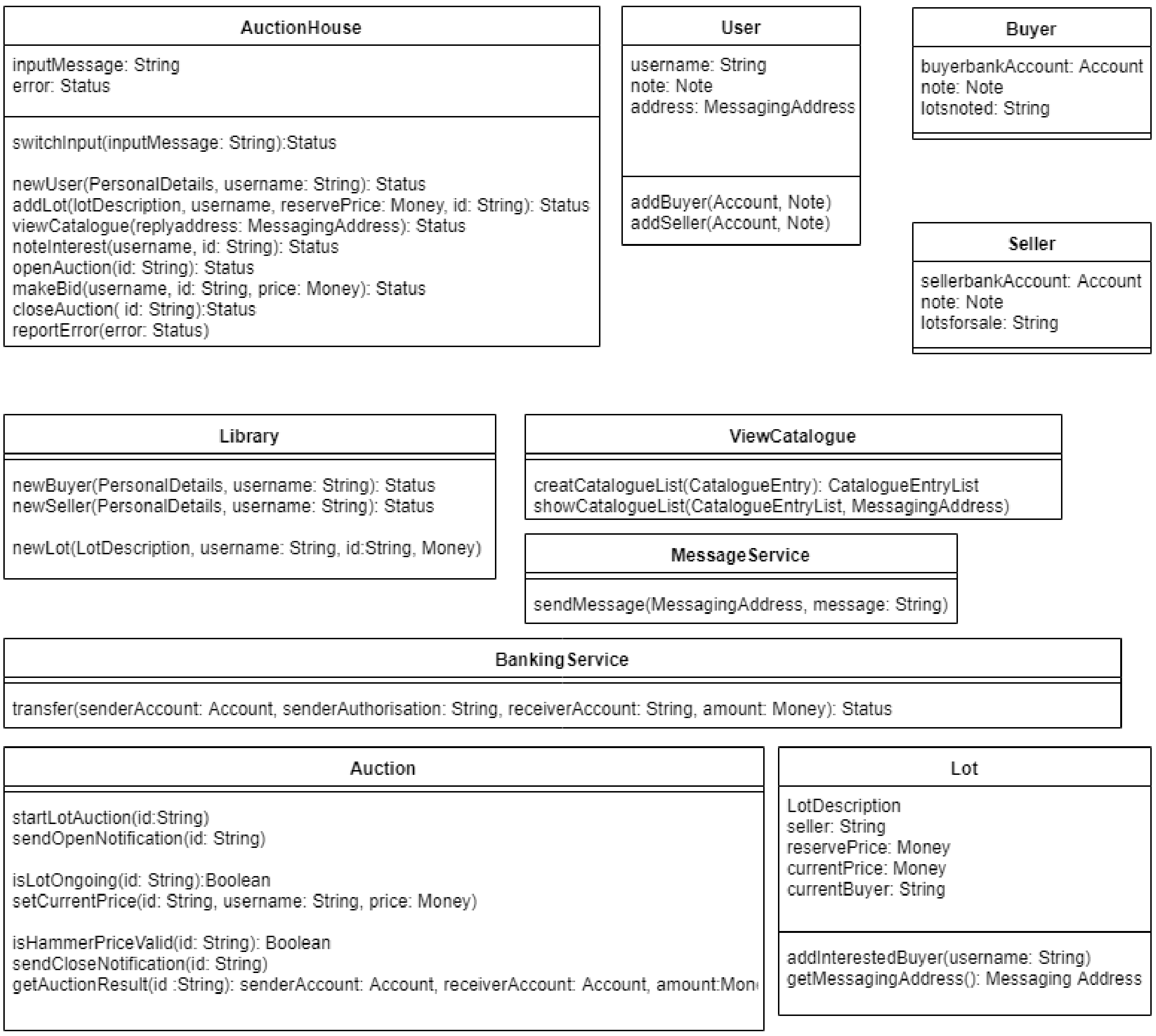

AuctionHouse
inputMessage: String
error: Status
switchInput(inputMessage: String):Status
newUser(PersonalDetails, username: String): Status
addLot(lotDescription, username, reservePrice: Money, id: String): Status
viewCatalogue(replyaddress: MessagingAddress): Status
noteInterest(username, id: String): Status
openAuction(id: String): Status
makeBid(username, id: String, price: Money): Status
closeAuction(id: String):Status
reportError(error: Status)
User
username: String
note: Note
address: MessagingAddress
addBuyer(Account, Note)
addSeller(Account, Note)
Buyer
buyerbankAccount: Account
note: Note
lotsnoted: String
Seller
sellerbankAccount: Account
note: Note
lotsforsale: String
Library
newBuyer(PersonalDetails, username: String): Status
newSeller(PersonalDetails, username: String): Status
newLot(LotDescription, username: String, id:String, Money)
ViewCatalogue
creatCatalogueList(CatalogueEntry): CatalogueEntryList
showCatalogueList(CatalogueEntryList, MessagingAddress)
MessageService
sendMessage(MessagingAddress, message: String)
BankingService
transfer(senderAccount: Account, senderAuthorisation: String, receiverAccount: String, amount: Money): Status
Auction
startLotAuction(id:String)
sendOpenNotification(id: String)
isLotOngoing(id: String):Boolean
setCurrentPrice(id: String, username: String, price: Money)
isHammerPriceValid(id: String): Boolean
sendCloseNotification(id: String)
getAuctionResult(id :String): senderAccount: Account, receiverAccount: Account, amount:Mon
Lot
LotDescription
seller: String
reservePrice: Money
currentPrice: Money
currentBuyer: String
addInterestedBuyer(username: String)
getMessagingAddress(): Messaging Address

Figure 1: UML Class Diagram (click to return)

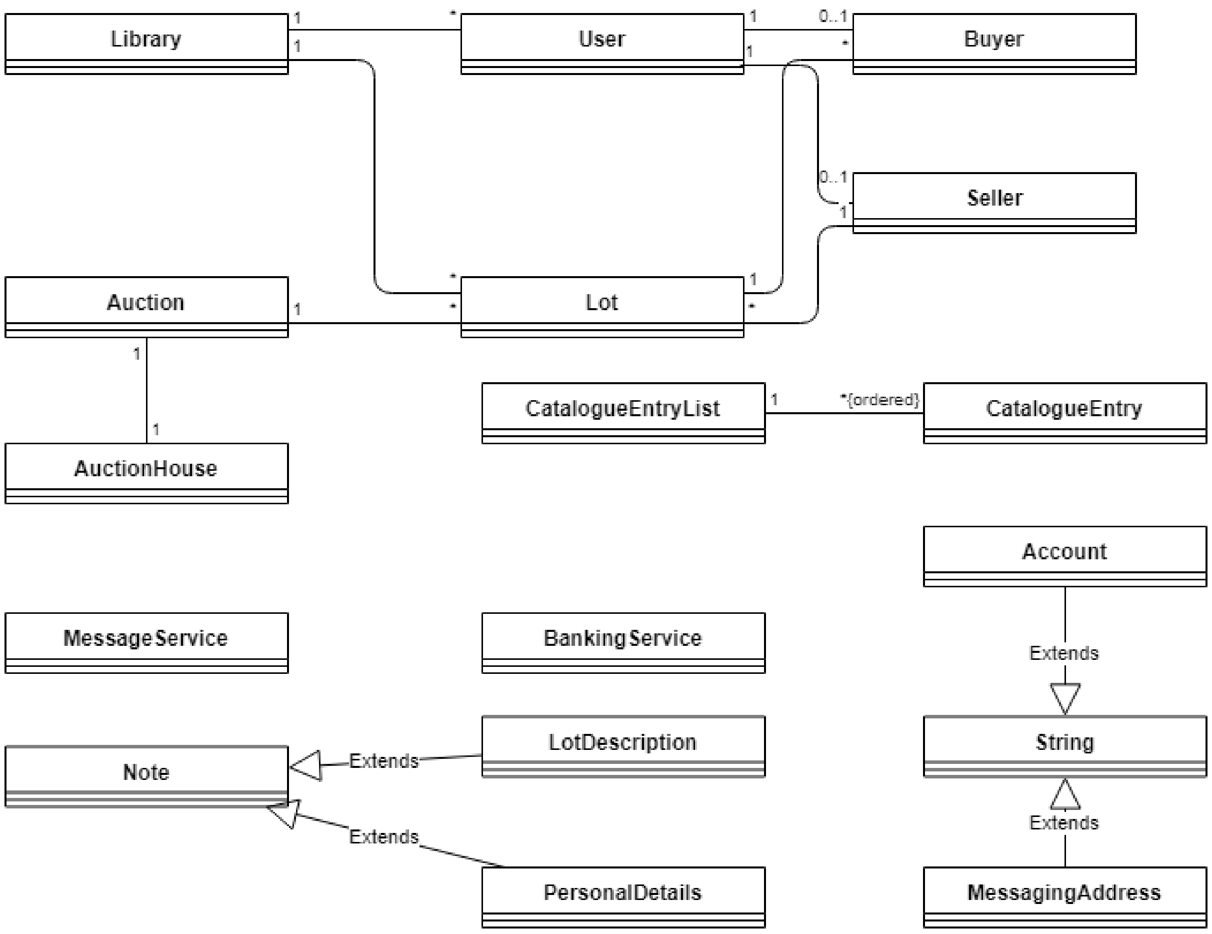


Figure 2: UML Class Association Diagram [*(click to return)*]

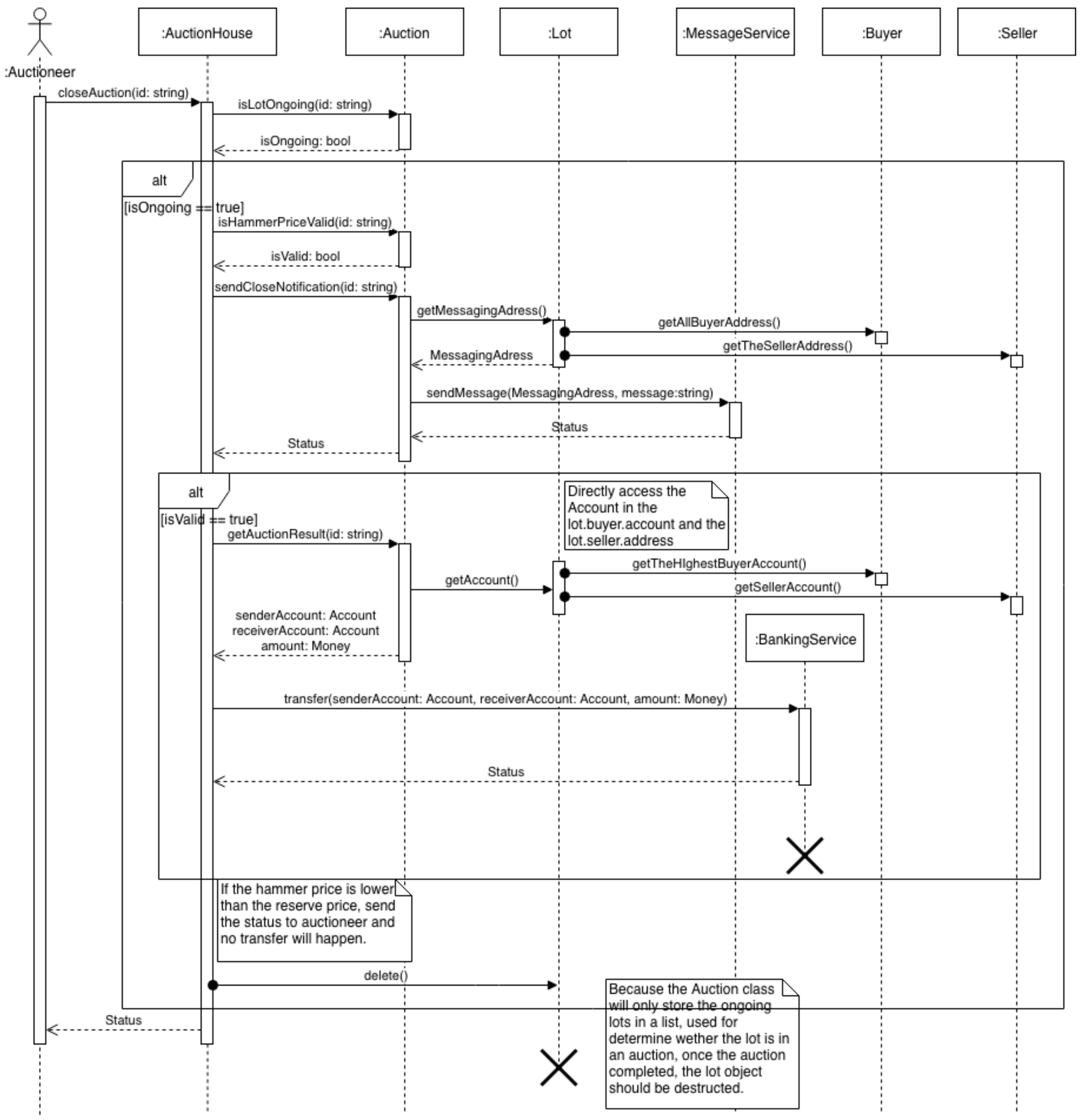
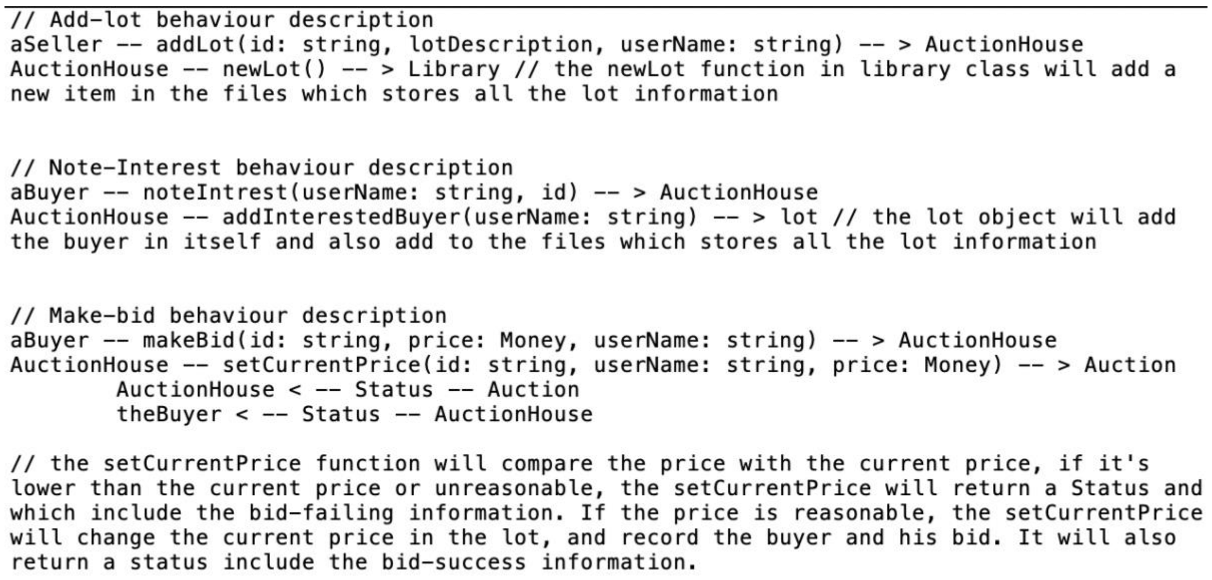

:Auctioneer
:AuctionHouse
:Auction
:Lot
:MessageService
:Buyer
:Seller
closeAuction(id: string)
isLotOngoing(id: string)
isOngoing: bool
alt
[isOngoing == true]
isHammerPriceValid(id: string)
isValid: bool
sendCloseNotification(id: string)
getMessagingAdress()
getAllBuyerAddress()
getTheSellerAddress()
MessagingAdress
sendMessage(MessagingAdress, message:string)
Status
Status
alt
[isValid == true]
Directly access the Account in the lot.buyer.account and the lot.seller.address
getAuctionResult(id: string)
getTheHIghestBuyerAccount()
getAccount()
getSellerAccount()
senderAccount: Account
receiverAccount: Account
amount: Money
:BankingService
transfer(senderAccount: Account, receiverAccount: Account, amount: Money)
Status
If the hammer price is lower than the reserve price, send the status to auctioneer and no transfer will happen.
delete()
Because the Auction class will only store the ongoing lots in a list, used for determine wether the lot is in an auction, once the auction completed, the lot object should be destructed.
Status

Figure 3: UML Sequence Diagram (click to return)



```
// Add-lot behaviour description
aSeller -- addLot(id: string, lotDescription, userName: string) -- > AuctionHouse
AuctionHouse -- newLot() -- > Library // the newLot function in library class will add a
new item in the files which stores all the lot information


// Note-Interest behaviour description
aBuyer -- noteIntrest(userName: string, id) -- > AuctionHouse
AuctionHouse -- addInterestedBuyer(userName: string) -- > lot // the lot object will add
the buyer in itself and also add to the files which stores all the lot information


// Make-bid behaviour description
aBuyer -- makeBid(id: string, price: Money, userName: string) -- > AuctionHouse
AuctionHouse -- setCurrentPrice(id: string, userName: string, price: Money) -- > Auction
        AuctionHouse < -- Status -- Auction
        theBuyer < -- Status -- AuctionHouse

// the setCurrentPrice function will compare the price with the current price, if it's
lower than the current price or unreasonable, the setCurrentPrice will return a Status and
which include the bid-failing information. If the price is reasonable, the setCurrentPrice
will change the current price in the lot, and record the buyer and his bid. It will also
return a status include the bid-success information.
```

Figure 4: Behaviour Descriptions (click to return)