Inf2C Software Engineering 2018-19
Coursework 3
Implementing an
Auction House System

Diana-Andreea Tanase s1534228
Valentine Dragan s1710228

November 2018

1 Introduction

This document presents the design and implementation for the BidIT Auction House System. The System enables Auld Reekie Auction House to have an online catalogue of lots that can be browsed by the general public. In addition, by using the System during an auction, the Auction House wants to replace the traditional bidding way with bids submitted from electronic devices (smartphones or personal computers), allowing some Buyers to be off-site.

This document captures the UML diagram for the classes of System, together with the association between them. The document also presents some implementation decisions. More information about the functionality of the System can be found in the coursework 1 handout. To learn more about the requirements for this design document, please refer to coursework 2 handout. Lastly, to read about the code skeleton provided, please refer to coursework 3 handout.

2 UML Class diagram

Figure 1 on the last page presents the static model into two separate class diagrams. Firstly, the upper subfigure presents the classes of the System along with their fields and methods. The second subfigure highlights the associations between the classes. A high-level description of the classes is presented in the next section.

3 High-level design description

Buyer stores the data of a registered buyer: name, messagingAdress, account, bank authorisation. This class is used to fetch information about a specific buyer when the AuctionHouseImp needs it for handling inputs or sending messages to its actor. The AuctonHouseImp stores a HashMap of all buyers. In coursework 2 we also used this class to handle input messages from Buyer actors, but we decided to separate those responsibilities and have messages from outside the system handled by the AuctionHouse only, for better cohesion. There exist 0 or more Buyers. A Buyer can note its interest and bid on 0 or more Lots.

Seller stores the data of a registered seller: name, messagingAddress, account. This class is used to fetch information about a specific seller when the AuctionHouseImp needs it for handling inputs or sending messages to its actor. The AuctionHouseImp stores a HashMap of all sellers. Similar to the buyer class, we decided to change the design of our Seller class from coursework 2, removing the methods that handle actions from outside the System. There exist 0 or more Sellers. A seller can add 0 or more Lots.

Auctioneer stores the data of a registered auctioneer. Similarly to buyer or seller, this class is used to fetch information about a specific auctioneer when the AuctionHouseImp needs it. Similarly to above, we have changed the bahavior of this class from coursework 2. An auctioneer can assist with 0 or more lots, while one lot can be opened and closed by only the same assigned Auctioneer. An auctionner object only saves its auctioneer name and messaging address. Although we could have made the code functional without it, we have felt that the class gives more readability and a cleaner way to save the messaging address of an actioneer.

CatalogueEntry contains the information about a lot that is only relevant to buyers. It is always associated to a Lot, which holds more information that the AuctionHouseImp needs. The catalogue entries are usually used when users are viewing/browsing the catalogue. We have decided to separate these two classes to avoid mixing information related to auction with information only relevant to buyers who only want to view all the catalogue entries. The AuctionHouseImp saves the Catalogue Entries ordered by their lot number.

Lot stores all information relevant to the auction of a lot and has methods for updating its fields when the AuctionHouseImp forwards a call to make a bid, add an interested buyer or open/close the auction. This class is also used to fetch information about a lot when the AuctionHouseImp needs it. The a lot's LotState changes when it's openLot and closeLot methods are called. We have decided to save a list of all interested buyers as well. Every Lot also has a unique associated CatalogueEntry with the same lotNumber, which is created when the Lot constructor is called. A lot has exactly one Seller, but can have 0 or more interested Buyers.

ActionHouseImp is the central component that all input messages are directed to. The inputs come from the outside world when triggered by actors interacting with the System. It extends the interface class AuctionHouse.

It usually fetches data from other classes: Buyer, Seller, Auctioneer, Lot or CatalogueEntry, checks if the input message is valid, and then forwards it to other classes. It calls BankingService when it needs to perform a transfer, and MessagingService when it needs to send out messages to actors. Additionally, the class has a private method sendMessageToBuyers which sends a message to a list of buyers, described below in *Implementation decisions*. Most methods in the AuctionHouseImp return a Status, which indicates how successful the methods have executed.

MessagingService is a singleton interface. Its functionality is essential for the System. It sends out messages to actors modelling people acting as buyers, sellers and auctioneers. Each type of message has its own different arguments for specifying who the message should be sent to, and additional arguments for the message itself.

BankingService is another singleton, its job is to take care of the money transfers between the sender and receiver (buyer and seller). The transfer is trigger by the **AuctionHouseImp** after an Auctionner closes the auction for a lot. Each transfer returns a Status and a transfer's Status can be fail or success.

LotStatus is an enumeration, representing the status in which a lot can be in, as described in the coursework skeleton.

Money is a class used to represent monetary values. It is used when a bid is made for an open lot and to transfer the correct amount from Buyer to AuctionHouse and from AuctionHouse to Seller.

Status represents the Status of any action required from the outside world. It consists of an enumeration of the Kind of Status, as described in the coursework handout.

MessageFlag is an enumartion, representing the type of message to send to a list of buyers.

4 Implementation decisions

4.1 Using HashMaps for the one-to-many associations in AuctionHouseImp

We have decided to use the Java collection class HashMap to implement the one-to-many associations of Buyer, Seller and Auctioneer in the class AuctionHouseImp. This is because all classes communicate with each other by passing around the usernames of other buyers, sellers and auctioneers. Additionally, all input messages contain the username of the specific Actor. Hence, the easiest way to find a Buyer, Seller or Auctioneer object given the username is to store it in a HashMap where its key is the username. HashMap ensures constant retrieval time of the object, given the key.

4.2 sendMessageToBuyers method and MessageFlag

We have decided to add a private method in AuctionHouseImp which sends a specific message to an entire list of buyers. This is very useful when a new buyer makes a bid, or when an auctioneer opens/closes an auction, and helps us keep the code tidy. We have implemented an enumerator class MessageFlag, which shows the type of message that needs to be sent out. The sendMessageToBuyers method gets a MessageFlag argument, to know what king of message to send.

4.3 Lot and CatalogueEntry

We have separated the two classes Lot and CatalogueEntry so that the latter only contains information relevant to Buyers who view the catalogue to read lot descriptions or decide whether to note interest. Lot stores more sensitive information such as reserve price or the names of all interested buyers, which we have decided to keep available only to the AuctionHouseImp class. For ease, we have decided that Lot and CatalogueEntry have the same lotNumber. The Lot/CatalogueEntry separation is the same as in our solution for Coursework 2.

4.4 CatalogueEntryComparator

We have implemented a comparator class CatalogueEntryComparator, whose purpose is to order all CatalogueEntries by their lotNumber. This is necessary when a buyer wants to view the catalogue. Moreover, we keep all the catalogue entries in the AuctionHouseImpl using a priority queue that takes our CatalogueEntryComparator, to ensure the desired order is preserved.

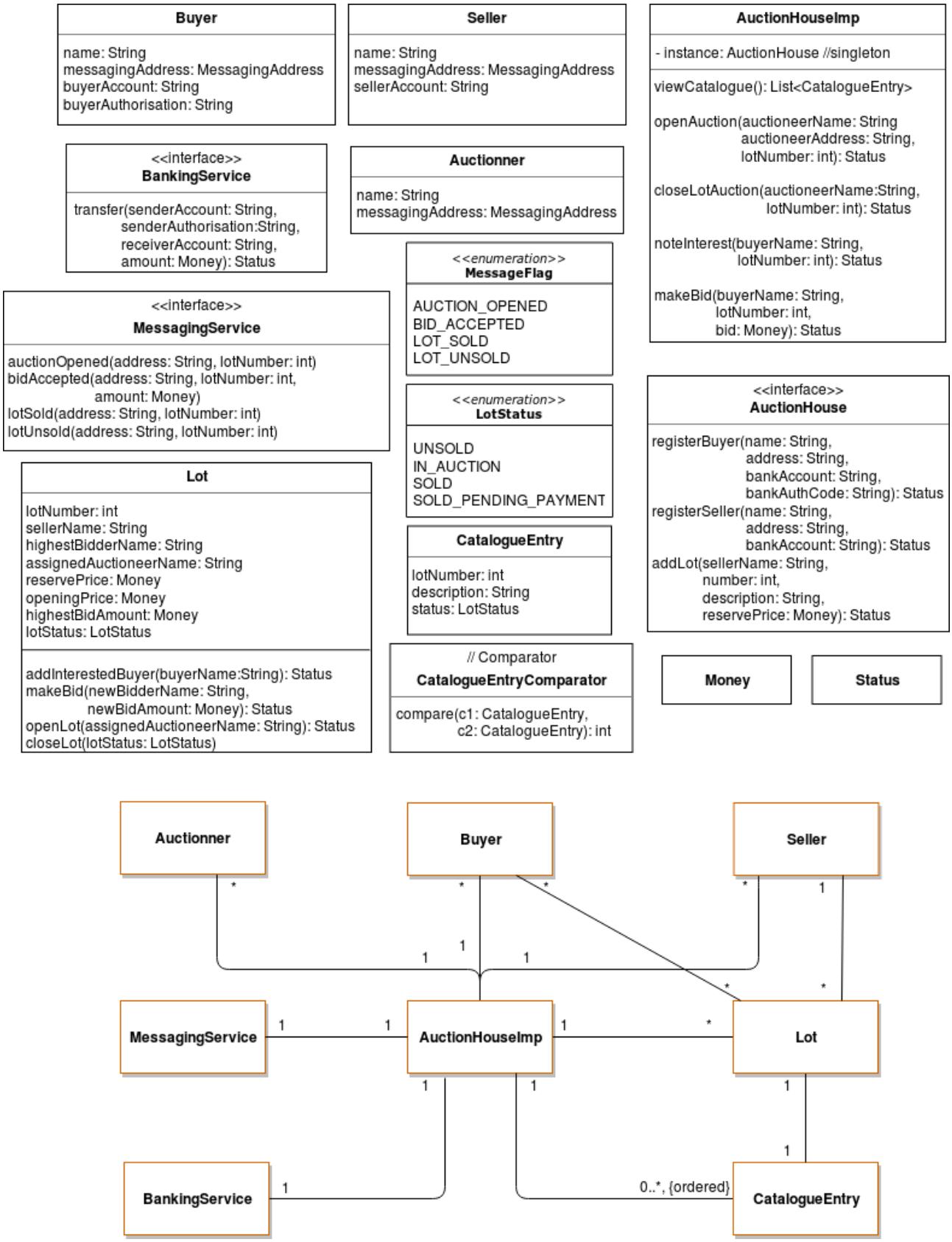

Figure 1: UML Class diagram