# Inf2C: Software Engineering
# Coursework 3

# Creating an abstract implementation of a auction house system

| | |
|---|---|
| Michael Andrejczuk | s1703773 |
| Dylan Joseph Thinnes | s1737075 |

November 27, 2018

## Contents

## 1 UML class diagram

Due to its size, this is provided as the last page of the document, with a separate diagram for object associations and for object methods and fields.

## 2 High-level design description

We give an overview on a per-class basis, giving particular attention to deviations from our Coursework 2 specification.

### 2.1 Actors

- **Actor** In our CW2, this was an interface which supported receiving/sending messages via the MessageService. We now implement it as a superclass, abstracting the concept of an actor associated with the system. In particular, an actor has two attributes associated with them:

  1. address (String), the messaging address. By including this in the superclass we ensure all actors have consistent behaviour in how their address is set and retrieved.
  2. auctionhouse (AuctionHouse), the AuctionHouse associated with the actor. This allows us to reference back to the auctionhouse object if needed, keeping our system understandable. It also means we could easily extend the system to support multiple Auction Houses.

  The Actor also now has the additional viewCatalogue function, which was previously implemented by all actor inheritors separately. It serves merely as an internal convenience function to calling viewCatalogue on the instance of AuctionHouseImp.

- **RegisteredUser** This now inherits from Actor, and extends it by requiring a specific username for each instance. This separates registered actors, such as Auctioneer, Client, and Seller, who need a username, from anonymous actors, such as MemberOfPublic.

- **Client** The client retains some functionality in Coursework 2, since it provides banking credentials. However, it is no longer responsible for handling personal details, as personal details are better put in RegisteredUser. Seller and Buyer inherit from Client, ensuring that both have relevant banking authentication details.
- **Seller** This remains the same as in CW2, except that all methods are implemented in superclasses.
- **Buyer** This remains the same as in CW2, except that all methods are implemented in superclasses.
- **Auctioneer** This remains largely the same as in CW2, except the Auctioneer now tracks if it is currently administering a lot. This needs to be tracked in order to ensure an auctioneer cannot run more than one auction at once. By keeping this in the Auctioneer class rather than storing this information in the AuctionHouse we ensure high cohesion and low coupling.

## 2.2 Auction House

AuctionHouseImp has largely the same shape as our CW2 implementation of AuctionHouse, with three major exceptions.

1. **Lots** These are now indexed by a Map of lotIds, which are Integers, to Lots. Since all lots are indexed using these, it is more efficient and clearer to have the data structure imply how we find and use these lots.
2. **Parameters** Parameters as provided during initialisation are stored in AuctionHouse as an attribute of type Parameters. We considered copying the attributes directly into AuctionHouse, but decided against it as it would tightly couple AuctionHouse to changes in the implementation of Parameters.
3. **Auctioneers** Our previous implementation did not store auctioneers, which was a major oversight. Now, we store them as a list, which simplifies interacting with and accessing them for purposes of lot administration.

## 2.3 Lot and Associates

- **CatalogueEntry** This remains the same as in CW2.
- **Lot** This remains the same as in CW2, except that there are no longer LotInformation and LotDescription attributes.
- **LotStatus** This follows our implementation in CW2, with the exception that the "Pre-auction" status has been replaced with "Unsold". This allows us to auction a lot again if it did not sell previously, a detail we had omitted in our CW2.
- **Bid** This remains the same as in CW2.

## 3 Implementation Decisions

We note particular points of interest in our implementation decisions:

- **Abstraction of actors**: we chose to represent all classes that may represent an individual (Seller, Buyer, Auctioneer, MemberOfPublic) as subclasses of Actor, RegisteredUser, and Client. This allowed us to follow a strongly object-oriented approach in which we inherit methods and necessary attributes, guaranteeing an easily composed and incrementally growing set of user features as we deal further down the inheritance chain, such as messaging on the first level (Actor), usernames on the second level (RegisteredUser), and banking authentication on the third (Client).

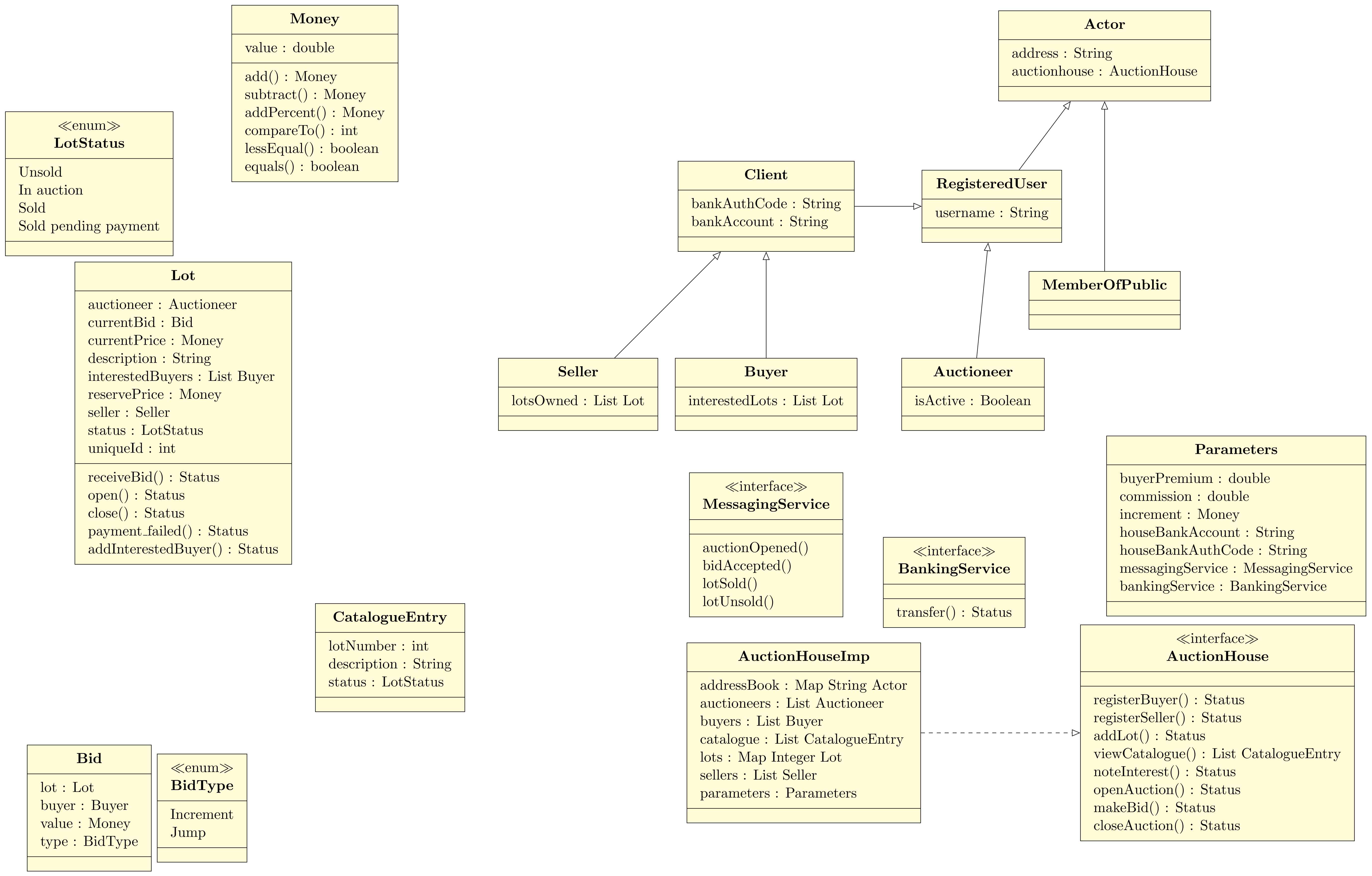
- **Removal of Client methods**: in our CW2, each Client had some set of actions they performed, represented by a method: for example, 'bidOnLot' for Buyer. These have all been moved into AuctionHouseImp, and actors can interact with the AuctionHouse as necessary to mutate the program state. This models the real world more closely — most dealings are by actors with the auction house, not between each other. It also has the far more important benefit of keeping logic properly encapsulated to each specific actor instance, whether that is Seller or Buyer or other actors. This keeps our system less dependent on the interfaces of these actors, and gives us more freedom for improvement in the future.

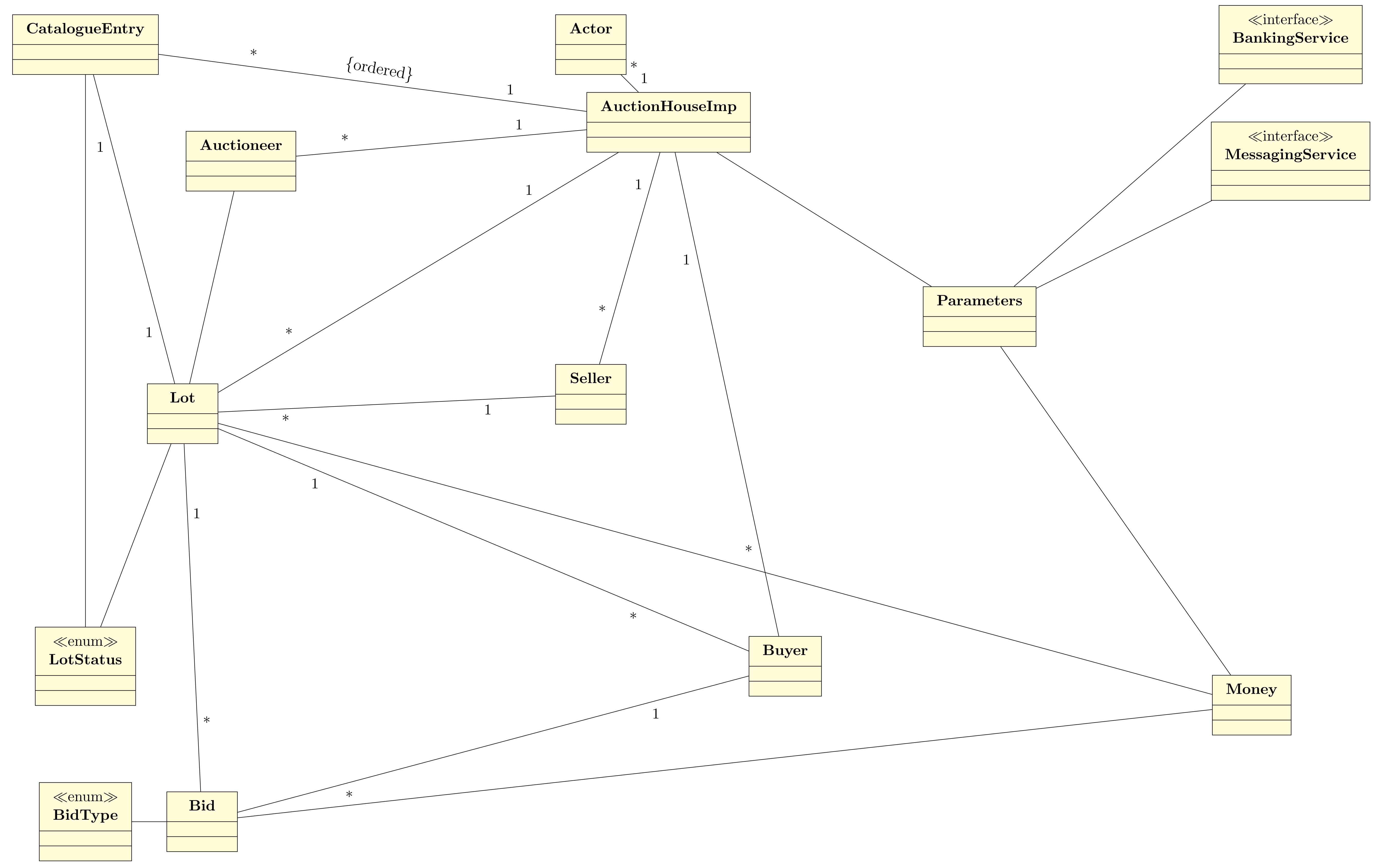
- **Consistent choice of collection classes**: we use ArrayLists and HashMaps where needed. Their APIs are well understood by even the most basic Java programmers, making our system easy to understand and, by extension, maintain and extend.

- **Keeping bid type distinctions**: while the CW2 spec considered only JUMP bids, we decided to implement a BidType enum to support either JUMP or INCREMENT bids in the future. This improves maintainability and extensibility if changes should wish to be made in the future.

- **Lot behavior on UNSOLD**: where a lot fails to move to SOLD after an auction closes (for example, it does not reach its reserve price), the question of which variables to reset is challenging. We decided to reset only currentBid (and by extension currentPrice), meaning that the list of interested buyers is preserved across multiple auctions of a single Lot. This supports good user experience: it is undesirable for a buyer to need to mark their interest in the same lot twice (and, indeed, it is considered an error for a buyer to mark interest in the same lot more than once). It also means we can carry over a large amount of information across lot auctions.

- **Lot logic handled in object**: Most of the logic relating to how a lot behaves (eg. when someone marks interest in it, or when being put on auction) is handled by the Lot object itself rather than AuctionHouse. This ensures high cohesion.


Money
value : double
add() : Money
subtract() : Money
addPercent() : Money
compareTo() : int
lessEqual() : boolean
equals() : boolean
≪enum≫ LotStatus
Unsold
In auction
Sold
Sold pending payment
Lot
auctioneer : Auctioneer
currentBid : Bid
currentPrice : Money
description : String
interestedBuyers : List Buyer
reservePrice : Money
seller : Seller
status : LotStatus
uniqueId : int
receiveBid() : Status
open() : Status
close() : Status
payment_failed() : Status
addInterestedBuyer() : Status
CatalogueEntry
lotNumber : int
description : String
status : LotStatus
Bid
lot : Lot
buyer : Buyer
value : Money
type : BidType
≪enum≫ BidType
Increment
Jump
Actor
address : String
auctionhouse : AuctionHouse
Client
bankAuthCode : String
bankAccount : String
RegisteredUser
username : String
MemberOfPublic
Seller
lotsOwned : List Lot
Buyer
interestedLots : List Lot
Auctioneer
isActive : Boolean
Parameters
buyerPremium : double
commission : double
increment : Money
houseBankAccount : String
houseBankAuthCode : String
messagingService : MessagingService
bankingService : BankingService
≪interface≫ MessagingService
auctionOpened()
bidAccepted()
lotSold()
lotUnsold()
≪interface≫ BankingService
transfer() : Status
AuctionHouseImp
addressBook : Map String Actor
auctioneers : List Auctioneer
buyers : List Buyer
catalogue : List CatalogueEntry
lots : Map Integer Lot
sellers : List Seller
parameters : Parameters
≪interface≫ AuctionHouse
registerBuyer() : Status
registerSeller() : Status
addLot() : Status
viewCatalogue() : List CatalogueEntry
noteInterest() : Status
openAuction() : Status
makeBid() : Status
closeAuction() : Status


CatalogueEntry
Actor
≪interface≫ BankingService
AuctionHouseImp
≪interface≫ MessagingService
Auctioneer
Parameters
Seller
Lot
≪enum≫ LotStatus
Buyer
Money
≪enum≫ BidType
Bid
{ordered}
*
1