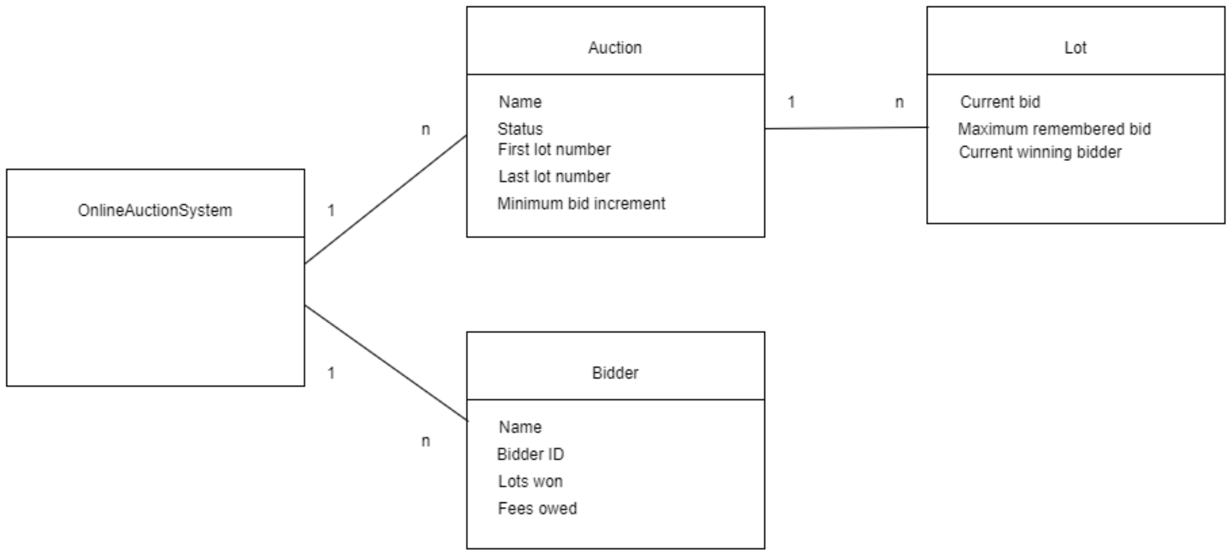
Assignment 1

Overview:

The online auction system allows multiple auctions to run at the same time. Each auction consists of multiple lots (A lot can contain multiple items). Multiple bidders can be added to the system who can bid on an open lot of their choice. The system has an auto-bid increment feature where the system can bid on behalf of the currently winning bidder if certain conditions are satisfied. When an auction closes, the bidder with the highest bid on the lot wins the lot.

Note: A comprehensive description of the auction system is in the "CSCI 3901 Assignment 1.pdf" file.

Implementation Overview:



1. In the implementation, the *OnlineAuctionSystem* entity keeps track of multiple auctions and bidders. The relationship between *OnlineAuctionSystem* and the *Auction* entity is 1:n. Similarly, the relationship between *OnlineAuctionSystem* and the *Bidder* entity is 1:n.
2. An auction entity keeps track of multiple lots. The relationship between *Auction* and the *Lot* entity is 1:n.
3. The *Auction* entity has the following attributes: Name, Status, First lot number, Last lot number and Minimum bid increment. In addition, it also has an attribute to keep track of lots.
4. The *Bidder* entity has the following attributes: Name, Bidder ID, Lots won and Fees owed. Lots won and Fees owed keep track of the status of the bidder's winning lots.

5. I have added an additional *Lot* entity in the implementation which keeps track of the current bid, maximum remembered bid and the current winning bid for a lot.

Files and external data:

I have created 4 separate java files for maintaining the following classes: *OnlineAuctionSystem*, *Auction*, *Bidder*, and *Lot*.

In addition to this, the mainUI.java file can be used to test the implementation.

Also, a simple text file can be used to pass as an input to the *loadBids* method.

Data structures and their relation to each other:

In the *OnlineAuctionSystem* class, I have defined two list fields(Dynamic arrays) to maintain the auction and bidder objects. The class has *createAuction* and *createBidder* methods to add these objects to the corresponding list. With this approach, it is easy for the system to maintain the state of the overall system. Also, I am using a dynamic array(An *ArrayList* implementation) because the size of the bidders and auctions is not known at the beginning(The information is passed during runtime).

I have created a new class called *Lot* to maintain the state of lots such as current winning bid, maximum remembered bid, and winning bidder. In the *Auction* class, I have defined a static array called lots to keep track of the lots in that auction. During the creation of an *Auction* object, I initialize all these slots in the array to a new *Lot* object. I have used a static array to maintain the lots because the lot range in a given auction does not change at any point in time.

Assumptions:

1. Duplicate auction name is allowed.
2. Minimum bid increment cannot be 0 or a negative value.
3. Bid value cannot be 0 or a negative value.
4. If the currently winning bidder of a lot bids again on the same lot with the same value(Maximum remembered value), then the current bid should not be updated and the return code should be 2.
5. When a bid value(A valid bid for a bidder other than the winning bidder) is less than the maximum remembered bid value and the next valid auto-bid value(bid + minBidIncrement) is greater than the maximum remembered bid, then the new current bid should be updated to the bid value.
6. When a bid value(by a bidder other than the currently winning bidder) is greater than the maximum remembered bid value and the auto-bid

value(maximumRememberedBid + minBidIncrement) is less than or equal to the bid value, then the new current bid should be updated to the next valid bid value.

7. When a bid value(by a bidder other than the currently winning bidder) is greater than the maximum remembered bid value and the auto-bid value(maximumRememberedBid + minBidIncrement) is greater than the bid value, then the new current bid should be updated to the old maximum remembered bid value.
8. There is no future requirement to change the size of a lot range in a given auction.

Key algorithms and design elements:

1. **Creating a dummy bidder:**
   I am creating a dummy bidder object and adding it to the bidders list while creating the *OnlineAuctionSystem* class. I am maintaining a *Bidder* object attribute in the *Lot* class to keep track of the currently winning bidder for that lot. To initialize the initial state of this attribute, I am passing down this bidder while creating the Auction object in the *createAuction* method.

2. **Creating auctions and bidders:**
   The *createAuction* checks for the bad inputs. If the inputs are correct, then it creates a new auction object and adds it to the auctions list. Similarly, there's a *createBidder* method to create a new bidder object and adds it to the bidders list.

3. **Initializing lots:**
   I have created a *Lot* class to maintain the state of a lot. In the auction class, I have used a static array to maintain the states of all the lots in that auction. During the creation of an auction, I am initializing each index of this array to a new lot object.

4. **Calculating lots won and fees owed for a bidder:**

   To avoid calculation of fees owed by a bidder by iterating over the lots, I have defined two additional attributes called *lotsWon* and *feesOwed* in the Bidder class. Whenever an auction closes, I pick up the current winning bidder and set lots won and fees owed by that bidder in the *updateBidderStatus* method.

5. **Calculating cumulative winning bids for an auction:**
   I am using the *cumulativeWinningBids* method to calculate the sum of the currently winning bids for an auction. It iterates over all the lots and adds all the currently winning bids.

6. **Checking if the bid is a valid next bid:**
   I have created *isValidNextBid* method to check if the bid is a valid next bid for a lot.

7. **Maintaining an auction status:**
   I am using a string attribute called status in the *Auction* class to maintain the status of an auction. The initial state of this attribute will be "open". There's also a method called *isOpen* to check if the auction is open.

8. **Processing bids:**
   In the place bid method, I have divided the bid processing into four parts.
   1) Handling bad inputs:
      - Invalid : bidder, bid, lot number
      - *Auction* is status not open
   2) Scenario when the winning bidder bids again:
      - I have created a method called *processWinningBiddersBid* to handle this scenario.
      - When the bid value is greater than the maximum remembered bid value, then update the max bid to the bid value. The current bid remains same.
   3) Scenario when a bidder other than the winning bidder bids and the bid value is invalid:
      - The *Auction* class has a method called *isValidNextBid* to handle this case.
   4) Scenario when a bidder other than the winning bidder bids and the bid value is valid:
      - There are three different cases to handle here. I have created a method called *processOtherBiddersBid* to handle these cases.
      - Case 1: bid < maxRememberedBid. When autoBidValue(bid + minBidIncrement) <= maxRememberedBid, update the current bid to the autoBidValue. Otherwise, set it to the bid value.
      - Case 2: when bid = maxRememberedBid. In this case, set the current bid to the bid value.
      - Case 3: bid > maxRememberedBid. When autoBidValue(maxRememberedBid + minBidIncrement) <= bid. In this case set the current bid value to the autoBidValue. Otherwise, set it to

the old maxRememberedBid. Also, set the maximumRemembered bid to the bid value and bidder to the current bidder.

9. **Calculating return codes:**
   I have written *getReturnCode* method in the *OnlineAuctionSystem* class to calculate return codes for the cases when the bid value is greater than the maximum remembered bid. I have used the below formula:

   When *Max bid – (Current Bid + Minimum Bid Increment) >= 0* return 4(Proxy bid is possible)
   Otherwise return 3(Proxy bid is not possible)

Limitations:

1. It is not possible to open an already closed auction.
2. It is not possible to change the lot range for a given auction.
3. In the current implementation, items are not defined in a lot.
4. A null value is returned for the bad input data.
5. Try Catch block is not used to handle bad input data in the place bid method.