CSCI 3901 Assignment 1

Tuesday, 21.09.2021

—

Nadipineni Hemanth Kumar [B00899473]

Overview

The problem given is to create an auction system where there are sets of rules and data associated. Also, we should be able to check the status in between. The approach being used here is to segregate data according to the set and manipulate it based on the input given. For a small overview, there is data related to Auction, Bidders, Lots. Whenever a new bid happens on a specific Lot, The data in the lot such as maximum remembered bid and next legal bid and the current bidder for the lot has to be changed. Whenever retrieving data using status/owed, we return these variables in the specific format given with "\t" and "\n"s;

Files and external data

The project consists of the following .java files and one .txt file for loadBids

1. MainUI.java

   The is the main class and it contains all the data related to console input and output. The data is taken from the input and will be passed into **OnlineAuctionSystem** or **Auction** classes based on the type of command the user enters.

2. OnlineAuctionSystem.java

   This class is essential for creating **Auction** ArrayList and **Bidder** Arraylist, processing **loadBids()** and **placeBids()** and retrieving data from **auctionStatus()** and **feesOwed()**

   - **createAuction() -** auction data has been validated and sent to A1 ArrayList.
     - null has been returned for all the bad data.

```
auction 1 10 5
null returned for auction
auction 1 10 5 null
null returned for auction
auction -1 10 5 auction1
null returned for auction
auction 1 -10 5 auction1
null returned for auction
auction 1 10 -5 auction1
null returned for auction
```

- Auction will be created in other cases too (1 character name, firstLot & lastLot=1, minimum bid increment = 1)

```
auction 1 1 1 A
Auction created.  Refer to it as auction 1
```

- **createBidder() -** bidder data has been validated and sent to B1 ArrayList.
  - null has been returned for all the bad data.

```
bidder
null returned for bidder
bidder null
null returned for bidder
```

  - 1 character bidder name is being allowed.

```
bidder B
Bidder returned with id 1 created.  Refer to it as bider 1
```

  - Multiple bidders with the same name are allowed.

```
bidder Hemanth
Bidder returned with id 3 created.  Refer to it as bider 3
bidder Hemanth
Bidder returned with id 4 created.  Refer to it as bider 4
```

- **loadBids()** - load a file with “bid” data and sends it to placeBids() for processing
  - Null/empty values for the filename are validated.

```
load
load "" returned null
load null
load "" returned null
```

- **placeBids() -** takes lotNumber, bidderID, bid and gets the Auction in which this lotNumber belongs to and passes this data to **runBids()** inside the Auction class.
    - bad data has been validated. (Negative lotNumber, bidderID, bid)
    -

```
bid -1 1 30
bid outcome 0
bid 1 1 -30
bid outcome 0
bid 0 1 20
bid outcome 0
bid 1 1 0
bid outcome 0
```

- **auctionStatus() -** for each Auction, returning **auction.name**, **auction.status** and **auction.getLotValue()** from Auction class.

```
//---------------------------Method to get Auction status-----------------------------------------
public String auctionStatus() {
    returnAuctionStatus = "";
    for (Auction auction : A1) {
        returnAuctionStatus += auction.name + "\t" + auction.status + "\t" + auction.getLotValue() + "\n";
    }
    return returnAuctionStatus;
}
//---------------------------Method to get Auction status-----------------------------------------
```

- **feesOwed() -** for each bidder, if auction is closed, return bidder name using **checkBidder.getname()**, bidder total lots using **checkBidder.noOfLots()**, bidder total sum using **checkBidder.sumOfBids().**

```
//---------------------------Method to get Owedfee-------------------------------------------------
public String feesOwed () {
    returnFeesOwed = "";
        for (Bidder checkBidder: B1) {
            for (Auction auctionBid:A1) {
                if (auctionBid.getStatus()=="close"){
                    auctionBid.findLots(checkBidder);
                }
            }
            returnFeesOwed += checkBidder.getName() + "\t" + checkBidder.noOfLots + "\t" + checkBidder.sumOfBids + "\n";
        }
        return returnFeesOwed;
    }
//---------------------------Method to get Owedfee-------------------------------------------------
```

3. Auction.java

- **Auction() -** Constructor to store Auction Data
- **runBids()** - Data passed from OnlineAuctionSystem will be processed and returned output 0 or 1 or 2 or 3 or 4 based on certain conditions.
- **openAuction()** - sets the value of status variable to “open” after checking. If it’s not closed previously at any time.

```
//----------------------Method to make an Auction Open for bidding---------------------------
public boolean openAuction() {
    if(Objects.equals(getStatus(), b: "close")){
        return false;
    }
    setStatus("open");
    return true;
}
//----------------------Method to make an Auction Open for bidding---------------------------
```

- **closeAuction() -** sets the value of status variable to “close” after checking. If it’s not opened previously at any time.

```
//----------------------Method to make an Auction Close for bidding---------------------------
public boolean closeAuction() {
    if(!Objects.equals(getStatus(), b: "open")){
        return false;
    }
    setStatus("close");
    return true;
}
//----------------------Method to make an Auction Open for bidding---------------------------
```

- **winningBids() -** Loop through all the lots and find out which bidder is winning and the lot amount.

```
//--------------------Method to print winning bids for each Lot--------------------------
public String winningBids() {
    returnWinningBids = "";
    for (Lot lot: L1) {
        returnWinningBids = returnWinningBids + (lot.getNumber() + "\t" + lot.getCurrentBid() + "\t" + lot.getBidderID() + "\n");
    }
    return returnWinningBids;
}
//--------------------Method to print winning bids for each Lot--------------------------
```

- **getLotValue() - auctionStatus()** from OnlineAutionSystem class uses this method to calculate the total lot value for each auction. In this method, we loop through all the lots and add the current bid value to **totalLotValue** and

finally we return this **totalLotValue** to be printed in auctionStatus.

```
//-------------Method to calculate lotValue to be used in auctionStatus--------------------
public int getLotValue(){
    int totalLotValue = 0;
    for (Lot checkLot: L1) {
        totalLotValue = totalLotValue + checkLot.getCurrentBid();
    }
    return totalLotValue;
}
//-------------Method to calculate lotValue to be used in auctionStatus--------------------
```

- **findLots() -** Loop through each lot and bidders and check how many lots and bid amount each bidder owed. This method is used for **feesOwed()** in **OnlineAuctionSystem.**

```
//-------------Method to to update number of lots and sum of fee Owed-----------------------
public void findLots(Bidder checkBidder) {
    for (Lot checkLot:L1) {
        if(checkLot.getBidderID() == checkBidder.getBidderId()){
            checkBidder.noOfLots++;
            checkBidder.sumOfBids = checkBidder.sumOfBids + checkLot.getCurrentBid();
        }
    }
}
//-------------Method to to update number of lots and sum of fee Owed-----------------------
```

4. Bidder.java

- **Bidder() -** Constructor to store Bidder Data

- **getBidderId() -** Method to get particular bidderId.

```
//----------------------------Method to get Bidder ID-------------------------------------
public Integer getBidderId() {
    return getBidderID();
}
//----------------------------Method to get Bidder ID-------------------------------------
```

5. Lot.java

- **Lot() -** Constructor to store Lot data

```
public int number;
private String belongsToAuction;
private int bidderID = 0;
private int currentBid = 0;
public int nextBid = 0;
private int minIncrementBid = 0;
private int rememberedBid = 0;

//------------------------------Lot Constructor------------------------------
public Lot(int number, String belongsToAuction, int minIncrementBid){
   setNumber(number);
   setBelongsToAuction(belongsToAuction);
   setMinIncrementBid(minIncrementBid);
   nextBid = getMinIncrementBid();
}
//------------------------------Lot Constructor------------------------------
```

Data structures and their relations to each other

<table>
<tr><td>OnlineAuctionSystem</td><td></td></tr>
<tr><td>Auction ArrayList - A1</td><td>
<table>
<tr><td>Auction</td><td></td></tr>
<tr><td>Lot ArrayList-L1</td><td>
<table>
<tr><td>Lot</td></tr>
<tr><td>number</td></tr>
<tr><td>belongsToAuction</td></tr>
<tr><td>bidderID</td></tr>
<tr><td>currentBid</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

	nextBid minIncrementBid rememberedBid
firstLot, lastLot, minIncrementBid, name	
Auction()	
runBids()	
openAuction()	
closeAuction()	
winningBids()	
getLotValue()	
findLots()	

Bidder ArrayList - B1

Bidder
bidderID, name
Bidder()
getBidderID()

loadBids()

placeBids()	
auctionStatus()	
feesOwed()	

Had used multiple variables, methods, classes to store, validate and process the data.

General flow of data:

- (auction 1 10 5 auction1) - Whenever data is given to the auction command, it will be stored inside A1 ArrayList as a new Auction object. Every time this auction command is used, a new object of Auction will be pushed into A1 ArrayList. Simultaneously, Lot objects with Lot data are created and these objects are pushed into L1 ArrayList.
- (bidder bidder1) - Whenever data is given to the bidder command, it will be stored inside B1 ArrayList as a new Bidder object. Every time this command is used, a new object of Bidder will be pushed into B1 ArrayList.
- (open 1) - This command checks if the Auction with number 1 is closed previously. If it's not closed, it sends the respective auction status to "open".
- (bid 2 1 30) - This command manipulates the data of lotNUmber-2 so that its bidderID will become 1 and currentBid will become 30. Before changing the data, it checks whether the Auction to which this particular lot belongs is open or not. Also, bids have to be processed in **runBids()** whether it leads to the previous bids or such conditions.
- (close 1) - This command sets the auction status to "close" and no more bids can be placed on lots in this auction.

Assumptions

- The primary assumption is that all numerical inputs are of data type - "int" and the input values are within -2147483648 to 2147483647.
- Another assumption was made on the place bids scenario. Consider the situation:

auction 1 10 5 auction1
Auction created. Refer to it as auction 1
open 1
Returned boolean value: true
bidder bidder1
Bidder returned with id 1 created. Refer to it as bider 1
bidder bidder2
Bidder returned with id 2 created. Refer to it as bider 2
bid 2 1 50
bid outcome 4
winning 1
Returned string:
1 0 0
2 5 1
3 0 0
4 0 0
5 0 0
6 0 0
7 0 0
8 0 0
9 0 0
10 0 0
bid 2 2 49
bid outcome 2
winning 1
Returned string:
1 0 0
2 49 1
3 0 0
4 0 0
5 0 0
6 0 0
7 0 0
8 0 0
9 0 0
10 0 0
bid 2 2 55
bid outcome 3
winning 1
Returned string:
1 0 0
2 55 2
3 0 0

4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0

Here, the previous remembered was 50 by bidder1 and the previous currentBid was 49. When the incoming bid is 55 by bidder2, currentBid may be updated to 49+5 or 50+5. It is assumed updating it from the previous remembered bid as if the previous remembered bid is more than incoming, updating with previous_bid+minimum_increment wouldn't satisfy the bidding logic.

Key algorithms and design elements

- Design is based on accessing objects and manipulating data of that object.
- Used ArrayLists to store objects.
- Created several methods for processing bids and calculating bid data
- Setters and Getters were used to validate and prevent data leaks.
- Constructors were used for Encapsulation.
- Abstraction concepts were used to minimize the redundant code.
- Similarly, several techniques like enhanced for loops, incrementors...etc. were used.

Limitations

- MainUI.java isn't altered. So that when an Id out of the ArrayList is given as the input, an Indexation error is thrown.
- In a real-world scenario, where billions of people and trillions of transactions happen, it isn't advisable to use int data type due to storage limitations.
- It is advisable to mutex to prevent data corruption for simultaneous transactions.