CSCI 3901 Assignment 1

Due date: 11:59pm Thursday, September 23, 2021 in Brightspace

Problem 1

Goal

Get practice in decomposing a problem, creating a design for a program, and implementing and testing a program.

Background

An auction is a way to sell one or more items. Items are grouped into lots for the auction; an auction can have several lots and each lot has one or more item in it.

During the period of an auction, individuals state a price, called a bid, they are willing to spend to buy a specific lot. At the end of the auction period, the individual with the highest bid on a lot gets all the items in the lot for the bid value.

During an auction period, an individual can place multiple bids on a lot and they cannot withdraw or rescind a bid once it is made.

Auctions often have a minimum amount by which a bid can increase over the current highest bid. For example, if the minimum increment is $5 and the highest bid for a lot right now is $10 then the next higher bid for that lot must be $15 or more. A bid of $11 in that case would not be accepted. The minimum bid also applies to the opening bid: even if there is no bid on the lot, we won't accept a bid for the lot whose value is below the minimum increment.

We will consider an online auction where the bids arrive electronically. In an online auction, the auction period is several days to give people time to bid without being online all the time.

With this online auction, we have a special feature for the bidders: if a bidder makes a bid on a lot that is higher than the next legal bid for the lot then the official bid is for the next legal bid value, but that higher value is remembered for the bidder. If another bidder bids on the same lot for a value above the current bid but below the remembered higher value then the auction system automatically looks to the stored higher value to make an automatic new minimum bid. If ever a new bid is submitted that matches the remembered higher bid then the first bidder, whose bid is remembered, wins the tie.

As an example, consider a bid on lot 18 where the current bid is $10 and the minimum bid increment is $5. There are currently no "remembered" bids on lot 18. Alice submits a bid of $35 for lot 18, which is higher than the next valid bid. The system records Alice's bid as $15 (the next valid bid of $10 + $5 increment) and remembers her maximum of $35. Bob sees that the current bid is at $15 for lot 18. He bids $25 for lot 18. Since there is a remembered bid for the lot of $35 from Alice and $25 < $35, Bob's full bid of $25 is accepted and the system automatically considers a bid by Alice for the next legal bid value, namely $30 (Bob's $25 + the $5 increment). This $30 bid is below Alice's maximum of $35 so it is made on Alice's behalf and the top bid for lot 18 is now $30.

If Bob re-bids $35 for lot 18 then his bid is considered since it is bigger by the minimum increment. However, his bid of $35 matches Alice's top "remembered" bid, so Alice still has the top bid for the lot, though for $35 now. If Bob re-bids again for $50 for lot 18 then his bid is higher than Alice's and he now has the lot for $40 (Alice's current $35 bid + $5 minimum increment) and the system remembers Bob's higher bid of $50 for the future.

If ever the winning bidder for a lot re-bids on the lot with a new bid, that bid can only increase the remembered maximum bid. The bidder remains the winner of the lot.

Problem

Write a class that will track and report on several auctions.

The data that you are working with are

- A set of bidders
- A set of lots
- A set of auctions

A lot belongs to a single auction. An auction has 1 or more lots. A bidder can bid on zero or more lots from zero or more auctions.

I will provide a "main" program that will then let a user invoke the operations on the data.

The normal use of your class will have the following sequence:

- Create an auction, which assigns lots to the auction
- Add a bidder to the system
- Open an auction for bidding
- Retrieve the current bid on a lot
- Place a bid on a lot
- Close an auction
- Report on the winning bids for all lots in an auction

Multiple auctions can be open at the same time.

Your class will be called "OnlineAuctionSystem", with supporting classes of Auction and Bidder. OnlineAuctionSystem must include the following methods:

- Auction createAuction (String auctionName, int firstLotNumber, int lastLotNumber, int minBidIncrement)
- Bidder createBidder (String bidderName)
- String auctionStatus ( )
- int loadBids (String filename)

- int placeBid (int lotNumber, int bidderId, int bid)
- String feesOwed ( )

Auction must include the following methods:

- boolean openAuction ( )
- boolean closeAuction ( )
- String winningBids ( )

Bidder must include the following methods:

- int getBidderId ( )

You may add additional classes or additional methods if you want.

Functionality

Auction createAuction (String auctionName, int firstLotNumber, int lastLotNumber, int minBidIncrement)

Create an auction with the given name and with lots given by the range of lot numbers (inclusive, so this auction contains both firstLotNumber and lastLotNumber). The lot numbers are positive integers and cannot overlap with any other auction.

Return an Auction object for the new auction, or null if there is an error.

Bidder createBidder (String bidderName)

Create a bidder with the given name. We are allowed to have more than one bidder with the same name. Assign bidder numbers to your bidders. Bidder numbers should be positive integers that increase from 1 as the bidders are created.

Return a Bidder object for the new bidder, or null if there is an error.

String auctionStatus ( )

Report on the state of each auction. Each auction will appear as a line in the returned string and be separated from each other auction with a carriage return (\n), meaning that printing the returned string will make it look like each auction is on a separate line.

Auctions will be reported in the order that they were created. For each auction report the auction name, status (new, open or closed), and the sum of currently-winning bids in the auction. Each of these fields for one auction will be separated from the others with a tab character.

An example of the output for two auctions, where the first auction is called "auction A" that is open with $1024 in bids so far and "second auction" that is closed with $4096 in bids would look as follows, when printed:

```
auction A	open	1024
second auction	closed	4096
```

and, as a string in Java, would be seen as (adding surrounding quotes):
"auction A\topen\t1024\nsecond auction\tclosed\t4096\n"

Return null if there is an error, and a valid (possibly empty) string otherwise.

int loadBids (String filename)

The given file contains a list of bids on lots. For each bid in the file (one bid per line), record the bid. Return the number of successfully-recorded bids from the file; a successfully-recorded bid is one that is validly-placed, not necessarily one that wins the lot.

Each line of the file contains one bid. It has the bidder number, the lot number and the positive integer bid. Each field is separated from the others with a tab character.

For example, the file could contain the following two lines:

```
17	672	10
23	824	256
```

Meaning that bidder 17 is bidding $10 on lot 672 and then bidder 23 is bidding $256 on lot 824.

int placeBid (int lotNumber, int bidderId, int bid)

Record a bid for a lot number according to the bidding rules described in the background section, including any automatic bids that result from this current bid. The bid value can only be a positive integer.

We will use return values to identify the outcome (advanced Java programmers may notice that exceptions would be better in some cases, but we will cover those later, so do not add exceptions):
0 – an outcome happened that isn't covered by the rest of the return codes
1 – the bid was not accepted
2 – the bid was accepted, but it is not the currently leading bid for the lot
3 – the bid was accepted, is the leading bid for the lot, and there will be no further automatic bids made for this bidder on the lot
4 – as in 3, but there is still room for an automatic bid to be made in the future on behalf of this bidder

If the winner for the lot re-bids on the lot but their new bid is below the current value then the bid is considered accepted but not leading. The bidders' previous bid on the lot remains valid.

String feesOwed ( )

Report on the state of each bidder. Each bidder will appear as a line in the returned string and be separated from each other auction with a carriage return (\n), meaning that printing the returned string will make it look like each bidder is on a separate line.

Bidders will be reported in the order of bidder numbers. For each bidder, report the bidder's name, the number of lots in _closed_ auctions that the bidder has won, and the sum of the bids for lots that the bidder has won in _closed_ auctions.

A sample output for two bidders (Alice and Bob), when printed, is

```
Alice	5	200
Bob	3	175
```

In which Alice won 5 lots totaling $200 and Bob won 3 lots totaling $175.

Return null if there is an error, and a valid (possibly empty) string otherwise.

boolean openAuction ( )

Sets an auction as open. That auction cannot already have been opened in the past.

Return true if the auction has been opened for bids and false otherwise.

boolean closeAuction ( )

Sets an auction as closed. You cannot close an auction that isn't open.

Return true if the auction has been closed for bids and false otherwise.

String winningBids ( )

For each lot in the auction (reported in order of lot numbers), report the lot number, the current maximum bid for the lot, and the id of the bidder who is winning the lot. Separate each field with a tab character and separate each lot line with a carriage return (\n).

If a lot has no bid then the bidder winning the lot is bidder number 0.

Consider an auction with 3 lots (123, 124, and 125). We would get the following output, when the string is printed to the screen, of

```
123	42	5
124	0	0
125	64	12
```

Representing lot 123 bid by bidder 5 for $42, lot 124 with no bids, and lot 125 by bidder 12 for $64. As a string, that output would be (with quotes added)
"123\t42\t5\n124\t0\t0\n125\t64\t12\n"

Return null if there is an error, and a valid (possibly empty) string otherwise.

int getBidderId ( )

For the bidder object, return the bidder id to be used in the bidding process.

Assumptions
You may assume that

- All bids are integers.
- Bidder names and auction names do not contain any special characters (including tabs).
- Each line in the file of bids is at most 80 characters long.
- Bids in the file for loadBids() are assumed to be happening in the order provided by the file.
- You don't need to worry about two bids trying to happen at identically the same time.

Constraints

- Write your solution in Java. You are allowed to use data structures from the Java Collection Framework.
- If in doubt for testing, I will be running your program on timberlea.cs.dal.ca. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Notes

- You will be provided with a "main" class on the course web page that will give you an interface through which to invoke your class.
- It is possible (and acceptable) to do this assignment with basic Java data types and ArrayLists. You are allowed to use or create more complex data types if you want.
- The functionality of your assignment will be assessed based on the results of test cases. Consequently, code that doesn't compile will not get marks for functionality.
- You are permitted to create more than one class in support of your work. You are permitted (and encouraged) to write additional methods for your classes if they help your overall design.
- Use your design notes to begin the external and internal documentation for your program. For internal documentation, if I hide the Java code, I should still get a reasonable sense of your method from the sequence of comments.
- Review the "lessons learned" slides from the course notes. For example, avoid magic constants, global variables, and public attributes in a class.
- When testing, your methods will be invoked with error conditions. You are expected to handle these errors without throwing exceptions.

- Review the test cases to identify special cases that you will want to consider in your work.
- Your class should never end by producing a Java error stack trace.
- The marking scheme has no marks for time or space efficiency.
- Use the number of marks for the parts as guides for how much time to devote to each method. Notice that there are significant marks for activities outside of writing executable lines of code.
- Recognize when you are spending time with no progress and ask for help before you find yourself spending hours without accomplishing something.

Marking scheme

- Documentation (internal and external) – 3 marks
- Program clarity, style, and coding best practices – 3 marks
- Design modularity, design, and use of support methods – 3 marks
- Resilience to bad data – 2 marks
- Auction and bidder management – 2 marks
- Processing of bids on lots – 6 marks
- Bid processing from a file – 2 marks
- Reporting functions – 5 marks

The functional testing will be done with an automated script, so stick to the input and output formats and to the given method declarations.

Tests

(To be divided into input validation, boundary tests, control flow test, and data flow tests)

Input Validation (generally, tests on bad input data for which you shouldn't crash)

createAuction():

- Null value passed as auction name
- Empty string passed as auction name
- For each of first lot and last lot:
  - Value is negative
  - Value is 0
- Minimum bid increment is negative
- Minimum bid increment is 0

createBidder():

- Null value for bidder name
- Empty string for bidder name

loadBids():

- Null value for filename
- Empty string for filename

placeBid():

- Negative bidder number
- 0 bidder number
- Negative lot number
- 0 lot number
- Negative bid value
- 0 bid value

Boundary Cases (tests at the edge of inputs or problem structures)

createAuction():

- 1 character auction name
- For each of first lot and last lot:
  - Value is 1
- Minimum bid increment is 1

createBidder():

- 1 character bidder name

auctionStatus():

- No auctions defined
- 1 auction defined
- Auction has a bid on a single lot

- Auction has bids on all its lots

loadBids():
- File doesn't exist
- File has 0 lines in it

feesOwed():
- There are no bidders defined
- There is 1 bidder defined

winningBids():
- Auction with 1 lot
- Auction with many lots

Control Flow Cases (tests of the core operations)

createAuction():
- Duplicate auction name
- Create an auction when there are no auctions already created
- Create an auction when there is 1 auction already created
- Create an auction when there are many auctions created
- For each of first lot and last lot:
  - Value is more than 1
- First lot < last lot
- First lot = last lot
- First lot > last lot
- Lot for a new auction overlaps with the lot numbers of another auction
- Lot for a new auction is contained in the range of lots of another auction
- Minimum bid increment is more than 1

createBidder():
- Duplicate name provided
- Create a bidder when there are no other bidders
- Create a bidder when there is already one bidder
- Create a bidder when there are already many bidders

auctionStatus():
- There are many auctions defined.
- One auction is in the "new" state
- One auction is in the "open" state
- One auction is in the "closed" state
- An auction has no bids on its lots
- An auction has bids on just some of its lots (more than 1)

loadBids():
- File exists
- File has 1 line in it
- File has many lines in it
- File has bids for lots all in same auction
- File has bids for lots in different auctions
- One line is accepted
- All lines are accepted
- Some lines are accepted

placeBid():
- Place a bid with a valid bidder number and there is 1 bidder defined
- Place a bid with a valid bidder number and there are many bidders defined
- Place a bid with an invalid bidder number and there is no bidder defined
- Place a bid with an invalid bidder number and there is 1 bidder defined
- Place a bid with an invalid bidder number and there are many bidders defined
- Place a bid with a valid lot number when there is just 1 lot defined
- Place a bid with a valid lot number when there are many lots defined
- Place a bid with an invalid lot number when there are no lots defined
- Place a bid with an invalid lot number when there is 1 lot defined
- Place a bid with an invalid lot number when there are many lots defined
- Place a bid on a lot of an "open" auction

- Place bids according to the following configurations:

current bid	min bid increment	remembered bid	the new bid being tested
none	1	none	1
none	5	none	1
none	1	none	100
10	5	none	5
10	5	none	10
10	5	none	15
10	5	none	20
10	5	20	15
10	5	20	20
10	5	20	25
10	5	30	20
10	5	none	11
10	5	20	23

feesOwed():
- There are many bidders defined

winningBids():
- A lot has no bids
- A lot has a single bid at the minimum increment level
- An auction with a lot with a high bid
- An auction with a lot whose current bid is lower than the lot's "remembered" bid

Data Flow Cases (tests around the order in which things are done)

placeBid():
- Place a bid on a lot of a "new" auction
- Place a bid on a lot of a "closed" auction

openAuction():
- On a "new" auction
- On an "open" auction
- On a "closed" auction

closeAuction():
- On a "new" auction
- On an "open" auction
- OPn a "closed" auction

feesOwed():
- There is a bidder who has won 0 lots in closed auctions and 0 lots in open auctions
- There is a bidder who has won 0 lots in closed auctions and 1 lot in an open auction
- There is a bidder who has won 1 lot in a closed auction and 0 lots in open auctions
- There is a bidder who has won 1 lot in closed auction and 1 lot in an open auction
- There is a bidder who has won lots in several closed auctions and 0 lots in open auctions
- There is a bidder who has won lots in several closed auctions and some lots in open auctions