Lab 8 - Auctioneer Observer Pattern

Rahul Kukreja & Jeffrey Yim

jyim3@bcit.ca

Welcome!

In today's lab, you will:

1. Simulate an auction involving an auctioneer and bidders using the observer pattern

Grading

This lab and all future labs will be marked out of 10

For full marks this week, you must:

1. (2 points) Commit and push to GitHub after each non-trivial change to your code
2. (6 points) Successfully write a program that correctly and fully implements these requirements using the observer design pattern
3. (2 points) Write code that is commented and formatted correctly using good variable names, efficient design choices, atomic functions. Ensure your code has been tested thoroughly.

Requirements

Clone the project from: https://classroom.github.com/a/aSLY5tRE

For this lab you will implement an Auction House simulation that consists of an `Auction`, an `Auctioneer` and a number of `Bidders`.

Let's break down each of our classes. I've listed the attributes I expect to see**. I have provided you with some skeleton code to start off with**. Feel free to add/remove any functions or change the class design however you feel. The rest of the class design is up to you. **DO NOT** create any other classes besides these 3.

Bidder - The Observer

Bidders are the people who place the bids during an auction. These observe the auctioneer. Every time the auctioneer accepts a bid, all the bidders should be notified and given the chance to retaliate by placing a new bid. A bidder cannot retaliate to their own bid.

Each bidder has the following attributes:

- `name`
    - The name of the bidder
- `budget`
    - The amount of money that the bidder is willing to spend. The bidder will not make a bid greater than this amount.
- `bid_probability`
    - This is a floating point value between 0 and 100. This represents the percentage probability chance that this bidder will retaliate to a bid with their own bid.
- `bid_increase_perc`
    - This is a number greater than `1` . A value of `1.4` translates to `140%`. This percentage is the value of the new bid that the bidder places. For example, if the bidder has a `bid_increase_perc` value of `1.5` (that is, 150%) then if this bidder were to place a bid, it would be `1.5` times the current highest bid on the item.
- `highest_bid`
    - The highest bid amount that was bid by this bidder.

The bidder should implement function `update(Auctioneer *auctioneer)`. This method should allow the bidder to place a new bid with the auctioneer (thereby causing another new bid, causing the auctioneer to notify all the observers again). Remember, a bidder should only bid if:

- The bid is against another bidder and not against themselves
- The amount that they bid (dictated by `bid_increase_perc` is not greater than their budget
- After accounting for their `bid_probability` . (Hint: use `random number generation` to generate a random double between 0 and 100)

Auctioneer - The Subject

The Auctioneer is the subject being observed in our observer pattern. The auctioneer is responsible for maintaining a list of bidders and notifying them (calling their update function) if and when it accepts a new bid.

An auctioneer (not a static class in case you were planning to make it static) contains the following attributes:

- `bidders` A vector of bidder pointers that the auctioneer can notify.
- `highest_current_bid` The value of the highest current bid.

- `highest_current_bidder` A pointer to the highest current bidder.

The auctioneer can accept new bids and if this bid is greater than the `highest_current_bid` then it accepts the bid and notifies all the bidders **EXCEPT** for the bidder that placed the new bid. Remember, each `Bidder` expects a pointer of the auctioneer to be passed to it when it is updated

Auction - Our controller that starts the simulation

The `Auction` class represents an auction object. Again this is not a static class (I do not want to see any static functions in this lab).

The `Auction` is responsible for setting up and starting an auction for an item with a starting price. An `Auction` should implement a constructor `Auction(vector<Bidder *> bidders`. `bidders` refers to a vector of `Bidder` pointers to participating bidder objects.

The `Auction` is responsible for registering the bidders to the `Auctioneer` (so they can be called back as part of the observer pattern). Once registered, the auction starts by placing the `item` and `starting_price` as the first bid via the `simulate_auction` function. The auctioneer's `accept_bid` function is called, which calls the auctioneer's `notify_bidders` function. This should notify all the observers by calling their `update` function. If an observer chooses to bid, they will call the auctioneer's `accept_bid` function, leading to a chain of new bids which in turn, cause the observers to be notified again.

Once the chain has come to an end (either no one bids or everyone has exhausted their budget), the Auction object should print out the result of the auction. This should be the name of the `Bidder` and the winning bid.

Print out a summary where each bidder's name and their highest bid is displayed.

Your `main()` method should:

1. Have some test bidders preloaded

Prompt the user for the following:

2. The name of the item being auctioned
3. The starting price
4. Provide the user with an option to:
    a. Start the auction with the preloaded test users
    b. Add more bidders. Prompt the user for the details of new bidders (name, budget, bid probability etc)

Once this has been done, create an `Auction` object and start the auction simulation! A possible output can look like the snippet below. Since we aren't worrying about a dedicated UI class, your print statements can be put wherever you deem them to be necessary.

Sample Output

```
Starting Auction!!
------------------
Auctioning Antique Vase at starting price $100
Priya bidded $110 in response to Starting bid's bid of $100
Jojo bidded $132 in response to Priya's bid of $110
Priya bidded $145.2 in response to Jojo's bid of $132
Scott bidded $290 in response to Priya's bid of $145
Melissa bidded $435 in response to Scott's bid of $290
Priya bidded $478.5 in response to Melissa's bid of $435
Melissa bidded $717 in response to Priya's bid of $478
Priya bidded $788.7 in response to Melissa's bid of $717
Scott bidded $1576 in response to Priya's bid of $788
Priya bidded $1733.6 in response to Scott's bid of $1576
Scott bidded $3466 in response to Priya's bid of $1733
Melissa bidded $5199 in response to Scott's bid of $3466
Priya bidded $5718.9 in response to Melissa's bid of $5199

The Antique Vase goes to Priya for $5718

Bidder: Jojo, Highest bid: $132
Bidder: Melissa, Highest bid: $5199
Bidder: Priya, Highest bid: $5718.9
Bidder: Kewei, Highest bid: $0
Bidder: Scott, Highest bid: $3466
```

- Ensure you push your work to github classroom. I'd like to see sensible git commits comments, and commits must take place at logical points in development.

That's it. Good luck, and have fun!