Project 1: Bidding Server

Contents
Project 1: Bidding Server ........................................................................................................... 1
  Goals........................................................................................................................................... 1
  Enlist the BitBucket repository ................................................................................................. 1
  Clients ......................................................................................................................................... 2
    Bidding Activities.................................................................................................................... 2
    Bidding Rules.......................................................................................................................... 3
  Project structure......................................................................................................................... 3
    Methods..................................................................................................................................... 4
    Fields......................................................................................................................................... 5
  Testing ......................................................................................................................................... 5
    Client Behaviors..................................................................................................................... 5
    Verification.............................................................................................................................. 6
  Submit your solution.................................................................................................................... 6

Goals
You have been contracted by Bidder.com, a fictional online auction site, to build a thread-safe bidding server. Some of the concepts and techniques you may find relevant and helpful to do this job successfully are:
  • Atomicity
  • Invariants
  • Concurrent collections

Enlist the BitBucket repository
Like for Project 0, the first thing you have to do is fork the repository containing the project.

If you haven’t already done it, create a BitBucket account, and make sure that your first and last names are set in the account settings page of BitBucket. (See the instructions in the requirements for Project 0.)

Fork Project 1. Go to:

https://bitbucket.org/cmsc433_spring2012/project1/overview and click on fork:
In the fork settings screen, make sure that the name you use for the project is `<Project Name>-<Directory ID>`. This is an important step you must follow in order to facilitate our grading process. Also, make sure that you check the Private checkbox, so that other students can’t see your solution. For example:

![Fork settings screen](image)

Now click on Fork Repository.

Now, in the Git window, paste the clone address associated with the cloned repository. Something like:

```
git clone https://jondoe@bitbucket.org/jondoe/project0-jondoe.git
```

You’re done. Now you can load the project in Eclipse.

(You can use various ways of cloning and importing your project into Eclipse. See the Piazza discussions for alternatives.)

**Clients**

**Bidding Activities**
The site will be visited by two types of clients: Sellers and Buyers. They will engage in the following activities:

- A Seller can:
  - Submit an item to be listed on the server.

- A Buyer can:
  - Retrieve a list of items currently listed
  - Check the price of an item
  - Place a bid for an item
  - Checks if a bid is successful

These activities will be supported by the methods you will implement for the bidding server. More details on these methods can be found later.
Bidding Rules
Bidder.com has specified a set of rules to guarantee the bidding process will proceed in a fair and orderly manner. The server class you are implementing will need to enforce these rules.

- **Listing:**
  - To ensure fairness, each seller will only have up to a fixed number of different items listed at any given time. This number will be called `maxSellerItems`.
  - To keep the site more focused, the entire site will only carry up to a fixed number of items at any given time. This number will be called `serverCapacity`.
  - To make accounting easier, all prices should be quoted in dollars (no cents).
  - To exclude luxury items, sellers can only set a lowest bidding price that is below $100 and minimum $0.
  - To help sellers and buyers identify a particular listing, when an item is listed, a unique listing ID is generated and assigned to the item. No two items should share the same listing ID.

- **Bidding:**
  - The bidding price for an item must be above the lowest bidding price set by the seller of the item.
  - A buyer can ask the server for the current highest bidding price for an item.
  - To ensure fairness, a buyer can hold active bids on no more than a fix number of different items. This number will be called `maxBidCount`. A buyer can hold only one maximum bid per item. In other words, once a buyer has placed a bid on an item, he can only do it again after another bidder has placed a higher bid.
  - If a buyer already has an active bid on an item and tries to place a new bid on the same item, the second bid will be ignored.
  - An item can receive an arbitrary number of bids as long as it has not expired (the bidding for it is still active).
  - Once a buyer has placed a bid, he is committed to buying the item if the buyer emerges as the winning bidder.
  - If the bidding for an item expires before anybody has placed a bid on it, the item is discarded – the seller won’t put it up for sale again, and the server will not claim any gains for it.

**Project structure**
For the purpose of this project, you will implement the server as a mutable singleton object that will be shared by a group of client threads, all running in the same process. You do not need to implement a real Web-based server/client application.

The only class you are required to modify is `Server.java`
Methods

- **int submitItem(String clientName, String itemName, int lowestBiddingPrice, int biddingDurationMs)**
  - A seller may call this method to submit an item to be listed by the bidding server. A seller client uses `clientName` and `itemName` to identify itself and the item that is submitted. The unit of BiddingDuration is in milliseconds. If the item can be successfully placed, this method returns the unique listing ID generated by the bidding server. If the item cannot be placed, for instance, the seller has already used up its quota or the site’s capacity has been reached (already `maxSellerItems` items listed), this method returns -1.

- **List<Item> getItems()**
  - A buyer client may call this method to retrieve a list of items currently listed. Each item object in the list gives information on the name of the item and the initial minimum bidding price (the actual bidding price of the item may have changed from its initial value – the new price can be retrieved by calling the method `itemPrice()`, described below.

- **int itemPrice(int listingID)**
  - A buyer can check the price for an item, by supplying the unique listing ID of that item. The returned value will be the highest bid made so far, or if nobody made a bid on the item, the minimum bid value supplied by the seller when he submitted the item. If there is no item with the supplied listing ID, the method returns an error value of -1.

- **boolean submitBid(String clientName, int listingID, int biddingAmount)**
  - A buyer client calls this method to submit a bid for a listed item. This method returns true if the bid is successfully submitted and false if the submission request is rejected. There are several situations when a bid submission request can be rejected. If a bidder already has bid on too many items, the bidder is not allowed to place bids on new items. If a bidder already has a bid on an item, the bidder is not allowed to place a new bid on the same item until another buyer has placed a higher bid. The bid can also be rejected if the item is no longer for sale, or if the listing ID corresponds to none of the items submitted by the sellers.

- **int checkBidStatus(String clientName, int listingID)**
  - A buyer client calls this method to poll the bidding server to check the status of a bid the buyer may have on an item. There are three statuses. (1) SUCCESS: if this item’s bidding duration has passed and the buyer has the highest bid. (2) OPEN: if this item is still receiving bids, and (3) FAILED: If bidding is over and this buyer did not win, or if the listing ID doesn’t correspond to any item submitted by the sellers.
Fields
The limits mentioned earlier are stored in three public constant integers (that may be changed at grading time):
- `int maxBidCount`
- `int maxSellerItems`
- `int serverCapacity`

Also, for publicity and accounting purposes, Bidder.com wishes to have frequent access to two statistics:
- the total number of items it has sold: `int soldItemsCount`
- the revenue generated through the site: `int revenue`

While it is possible to compute these statistics from the database every time they are requested, let us assume that (for the sack of this project) it is desirable to simply maintain two mutable fields to keep track of these two statistics, updating them incrementally when new items are sold and more revenue is generated. However, since the server is running in a multi-threaded environment, you will need to ensure concurrent modifications to these fields can be done safely.

Testing
To test if the bidding server is functional and behaving correctly, a number of seller and buyer client classes have been implemented for you. Keep in mind that different tests may be used for grading, so when implementing your solution, don’t rely solely on the provided tests to point to your mistakes. A solution that passes all the given tests may still be penalized at grading time if it is not correct.

A test program is written to run instances of these client classes in multiple threads, accessing the shared singleton instance of the bidding server class you are implementing. The lifecycle of the test program has three stages:

1. Create several clients and execute them on multiple threads
2. Wait for all the clients to finish
3. Verify correctness

Client Behaviors
The behaviors of the seller and buyer clients are described here. The source code of these clients is provided to allow you to better understand how these clients wish to interact with the bidding server. This understanding will help you program the server to handle these clients correctly.

A typical seller client behaves as follows:
A seller client is initialized to carry a unique name and hold a list of 100 items to sell.
It enters a loop with a predefined number of cycles.
During each cycle,
  o It randomly picks an item and tries to submit to the server.
  o If the submission is successful, the item is removed from its to-sell list.
  o After each submission attempt, it sleeps for some random amount of time.

A buyer client behaves as follows:
  A buyer client is initialized to carry some amount of cash (e.g., $500).
  It enters a loop with a predefined number of cycles and an exit condition (running out of cash).
  During each cycle,
    o It retrieves a list of items.
    o It randomly picks an affordable item to bid.
    o It adds $1 to the highest bidding price.
    o For each item the buyer has an active bid on
      ▪ Check the status of the bid
      ▪ If the bid is successful, deduct the price from its cash.

Note that these clients are not designed to follow the bidding rules specified by Bidder.com. For instance, a seller client may submit an item even though it has already used up its quota (i.e., no more than 10 items). It is the responsibility of the server to enforce the rules and report failures when rules are violated.

Verification
Correctness will be verified by the test program in two ways. First, the test program will verify whether the statistics collectively seen by the clients should match those maintained internally by the server. For instance, the total revenue generated can also be calculated from clients’ perspective, by iterating through the buyer objects, summing over the prices each buyer has paid for the items won. This total revenue as seen by both the clients and by the server should be the same.

Second, the test program will verify whether bidding rules have been strictly enforced. In other words, the invariants underlying these bidding rules must hold. For example, the number of items currently listed should not exceed Server.serverCapacity.

You should make sure all the test cases provided in the project are passing before submitting your code. If the single threaded tests don’t pass, or if your code does not compile, you will not be given credit for the project.

Submit your solution
Follow the same steps described for Project 0 when you’re ready to submit your solution.

Note that you may submit your code changes any number of times before the project deadline. We will only grade on the last submission. To be sure you don’t get penalized uselessly, don’t change any other classes except for Server.java, or the directory structure!

Questions
Any questions you have, you should post on the Piazza forum. If you’re unsure if a question is allowed public audience, use the forum’s capabilities of sending private posts to the instructors. We will examine your question, and if we decide it’s a general interest question, we’ll change its status to public.

You’re encouraged to discuss the project with each other, and answer your classmates’ questions, but remember you must not share any part of your solution code online.

Good luck!