EE 422C, Software Implementation II
Vallath Nandakumar
Summer 2020
Last modified: Nov 9, 2020

The final project

This project showcases a server-client architecture. You are expected to use Java Server and Client sockets, multithreading, JavaFX graphics, data structures, and OOP concepts -- basically, all that you have learned this semester (except perhaps recursion!).

There are 3 main parts of the project -- the server back-end, the client GUI, and the client back-end. One might say that they are each 1/3 of the effort.

On the server side is an online auction server that handles customer bidding and selling of several items. A list of items with their description and minimum acceptable price will be read in by your Server from a file or other database as the server starts up. An optional feature may permit Customers to search for items in the database with various filters. As customers place bids, the server updates prices and informs all active customes of the new highest bid. The item is marked sold to the highest bidder when the auction closes, and no more bids will be accepted. The server will be capable of handling multiple customers with the help of multiple threads. You should employ proper synchronization so that multiple people can bid on the same item at the same time with thread safety.

Other details, such as the time of completion of the auction or an individual item's auction, or the item's description or image, could also be read in as the server starts up.

On the client side, a client gets a start-up window where he or she logs in with a user name and, optionally, a password. He could also be a guest with no password. He selects an item to bid on and places a bid, which is sent to the server. An optional feature would be automatic bidding. Credit card payments might optionally be accepted, and should, optionally, be transmitted securely. Customers should be able to view their bid and purchase history for that login session, and, optionally, for previous sessions. The server too might have memory to store histories even after reboot.

This is an individual project, and absolutely no collaboration or even discussion is allowed between students and non-instructors outside Piazza. The answers on Piazza might be less explicit than in other projects, and might be more like hints.

GLHF!

Below are specific requirements and possible options, in 2 lists.

Required Features (75% of the grade)

1. Server that can handle at least 5 auction items and 5 customers. Customers and items will have unique names that are single words. Concurrent bidding on these 5+ items should be allowed.
2. The Server need not have a GUI. The console should be used only for debugging output.
3. All communication between the Server and Customer objects should be through Sockets. A program that will work only if they are on the same machine is not acceptable. During grading, we might run your software on multiple machines at our end. We realize that such testing may be difficult or impossible for you, so our grading plan will take that into account.
4. During testing, no Java source file should be shared between the server and customers or between customers. Make separate Eclipse projects for the server and customer, and duplicate files as necessary across projects.
5. The server initializes items for the auction upon starting up. You should do this through a file read or other similar means.
6. The server accepts valid bids. A valid bid is a bid that is greater than the previous high bid for that item, and on an item whose auction is not closed. Bid amounts should be doubles.
7. Server informs all customers when a new valid bid is received, so that customers are aware of how much to bid next. Customers are also aware of whether an item can be bid upon further, or whether the auction for that item is closed.
8. Each customer should have a GUI-based user interface. You should use GUI techniques to input and output data, and avoid typing. Only debugging output should go to the console. You must use only JavaFX for the GUI graphics.
9. Customers should be able to place bids, and the program should tell a customer when a bid is invalid. A bid can be invalid if the bid amount is too low, or if the auction for that item is closed. The reason should be visible to the customer.
10. Every item's auction should have a termination mechanism. This could be because of time running out, a set price being met, or the server somehow terminating the auction -- pick one of these at least. When the auction is over, a buying customer, if any, should be identified, or the item is marked unsold. Customers should be able

to determine what the high bid at closure was, if any.

11. Provide Quit and/or logout buttons that end the program or parts of the program gracefully.
12. Provide synchronization so that multiple clients bidding at the same time are handled gracefully.
13. Your code should not throw Exceptions under normal operation. For example, if the server terminates normally (such as when the auction time-out happens), or a client leaves normally, there should be no exception thrown.
14. Good style, synchronization, OOP practices. This will be worth 15% of your grade.
15. You will have to provide documentation for your program. The documentation should be in two parts -- one from a programmer's point of view, and the other from a user's point of view. Explain the mode of use and list the features, including the optional extras. This document could also include a diagram or two. The last page of the document should be a list of references (links, for example) from where the user has copied code (more than 3 lines) with or without modification. Do not include the textbook or material discussed or presented in class. The document should be 2-4 pages long. Name the document FinalProject_EID.pdf or FinalProject_EID.txt. Submit this separately, and not in the zip files. (3%)
16. Submit your solution in four zip files, two for the server and two for the customer. The Server top class should be called Server, and the customer top class should be called Client.
    Here is what you should do: Make an executable jar for the client or server. Then put all your other files and your jar file in a zip file, such that double-clicking the executable jar will read the initialization file, any sound files, etc. properly. Submit the two zip files. If double-clicking won't work, it is OK, as long as `java -jar your_jar_file` works.

Name the zip files *FinalExam_Server_jar_EID.zip* and *FinalExam_Client_jar_EID.zip*.

You should also turn in 2 additional zip files, with just your source code. This is for us to look at your source code if required. Name these *FinalExam_Server_code_EID.zip* and *FinalExam_Client_code_EID.zip*.

The important thing here is not your file or submission format, but an instructor should be able to run your Server and Client with no help from you. So any deviation from the instructions should be documented carefully in your README. Normally, when we grade, we will have you run the programs after downloading them from Canvas. So, if you have a lot of trouble getting the jar files, you will not lose all the points as long as you are able to show us how to run your program.

You may also choose to submit maven project jar files instead of the ordinary executable jar files.

Remember to include the contents of the Header file at the top of every java file and PDF file that your submit.

17. When we grade you, you will be asked to explain your project. Your grade will be influenced by how well you answer the questions.

Optional Features (25+% of your grade)

1. Set a minimum starting price > 0 for every item. 1 pt
2. Set the duration for the auction for each item separately. 1 pt
3. Set a high limit that is a 'Buy It Now' price. When a customer bids that amount, he/she gets it right away. 1 pt.
4. Every customer can see the bid history of every item, including, perhaps, who made the bid. If the item has been sold, every customer should be able to see the buyer and the selling price. 1-2 pts
5. Items could have descriptions that are visible to customers. Could include images. 1-2 pts
6. A search feature to search through items, customers, etc. 2 pts
7. Non-volatile history of auctions, customer activity etc. 2-3 pts
8. Sound effects, nice GUI. 1-6 pts
9. Count-down clocks for items. 1-3 pts
10. Using the Observable class and Observer interface. 3 pts
11. Use a cloud server to host the auction Server. 3-5 pts
12. Cryptography techniques -- encryption of passwords, password hashing and salting, encryption of messages. 2-5 pts
13. Use Java SQL or other database. 2-4 pts
14. Other ideas? Think of eBay, perhaps. ???

Progress check

There will be a progress check the week before Thanksgiving during recitation. Attendance and presentation of your work is mandatory. For full credit, you should have completed about 1/3 of the work.

General Hints

Start early, and test often.
Use Git to safely store your code. *You must use the GitHub repository that we ask you to make, as per the instructions on Canvas.*
Don't get overly ambitious. Do the minimum first and make your GUI simple, unless you have time to fix bugs in more complex code or go back to a simple GUI if bugs are stubborn.
Scenebuilder has a steep learning curve, and unless you used it for project 5 or can start early, consider not using it here.

Starting code

See Canvas.

Changes to this document

This document is subject to change. Watch Piazza for changes, which, I hope, will be minimal.

Notes & FAQ

Submission file generation

If you have external libraries, such as JSON, you must make sure that your project will run on our machines easily. This can be done by making a Runnable Jar, which will include the external libraries.
A more ambitious and better system would be to make a Maven project, which downloads any external libraries from the Maven repository online before running a project. Warning: this can be time-consuming.

To create your jar files, please see the following instructions:

IntelliJ: https://www.jetbrains.com/help/idea/compiling-applications.html#package_into_jar

Eclipse: https://www.cs.utexas.edu/~scottm/cs307/handouts/Eclipse%20Help/CreateJARInEclipse.htm (This is a decent explanation, but please use the runnable jar option instead of just the jar option.

Maven command line: http://maven.apache.org/plugins/maven-jar-plugin/usage.html

Note - creating executable jar files from maven is slightly more complicated. Please see this post for information on creating a jar manifest to set the entry point for the jar: https://mkyong.com/maven/how-to-create-a-jar-file-with-maven/

To create your zip files, please zip the entire project folder structure for your server and for your client. If you have created a maven project for either, make sure to include all of the necessary components (basically make sure you include your pom.xml file).

Checkout

Project checkout will be conducted much like project 5 where you will meet with an instructor over Zoom and run your project. During this checkout, we might download your executable jar files and run them. If you have any kind of extra credit where your server is hosted somewhere externally, i.e. AWS, you will need to have that server set up and running before your checkout starts. When we download and run your program during checkout, your jar file can either allow an address to be input at runtime or it can be hardcoded to your server. (Note - if this is the case, please give us instructions in where the address is hardcoded so we can go back and change it if need be). We will do our best to evaluate your extra credit and project implementation during this time, so you don't have to leave your cloud resources running indefinitely.

We want to get through grading quickly, so make sure that you are fully ready by trying out your executable jars downloaded from Canvas. If they don't work by double-clicking, try them from the command line using

```
java -jar <executable_jar>
```

Frankly, we expect some of you to have problems with your executable jars. Try your best.

Please let us know if you have any other questions or concerns about formatting.

FAQ

1. Should I use JSON or ObjectInputStream/ObjectOutputStream?

   Both have advantages. JSON messages are in text format, so they might be easier to debug. On the other hand, they do need an external library, leading to some complication. Students have used both successfully.

2. ObjectOutputStream.writeObject resends the old object instead of the one I really want to send.

   Make sure that you run the command

   `objOutStrm.reset()` before writing the object, and `objOutStrm.flush()` after.

   https://stackoverflow.com/questions/2393179/streamcorruptedexception-invalid-type-code-ac

   Make sure that you don't create multiple sources of data going into a single ObjectOutputStream.

3. Serialization of objects is not working.

   Make sure that all your classes that are serialized have a serialization ID and implement Serializable.

4. ???

Instructions from Canvas

https://utexas.instructure.com/courses/1279392/assignments/5030927