Final Project

Description

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)!

You will design and implement an online auction server and client. There are 3 main parts of the project – the server backend, the client GUI, and the client backend. You may expect each to require about 1/3 of the effort.

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.

See the Canvas Assignments page and Files for starter code.

GLHF!

Implementation Requirements

Required Features (72% of the grade)

Server Side

You will create an online auction server that handles customer bidding and selling of several items. 1. As it starts up, your server will read in a list of items containing their descriptions and minimum acceptable prices. - It may also read in other details, such as the time of completion of the entire auction or an individual item's auction, or the item's description or image. - You should do this through a file read or other similar means. 2. As customers place valid bids, your server should update the prices and inform all active customers of the new highest bid. 3. The server must accept only 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 `double`s. 4. 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 at least one of these. When the auction closes, an item should be marked sold to the highest bidder and your server should accept no more bids on the item. Clients should be notified of what the high bid at closure was, if any. They should also be informed that the item can no longer be bid upon. 5. The server must be able to handle at least 5 auction items and 5 customers concurrently. Customers and items should have unique names that are single words. 6. Concurrent bidding on the 5+ items should be allowed. You should employ proper synchronization so that multiple clients can bid on the same item at the same time in a thread-safe way. 7. The server is **not** required have a UI or GUI. We recommend that console be used only to output logs for debugging.

Client Side

You will also create an online auction client which can connect to your server and allow a customer to participate in the auction(s).

1. The client should provide each customer with a JavaFX graphical user interface. **You must use only JavaFX for the GUI.** You should use appropriate GUI elements to input and output data, and minimize typing. Only debugging output should go to the console.
2. A customer should get a start-up window where he or she logs in with a user name and, optionally, a password. The customer could also be a guest with no password.
3. After logging in, a customer must be able to select any item to bid on and place a bid, which is sent to the server.
4. A customer should be notified of all bids placed by all customers during their login session, and be able to view a history of those bids.
5. The client must always inform the customer of the most recent highest bid from any customer, so that customers are aware of how much to bid next.
6. The client must also show the customer whether an item can be bid upon further, or whether the auction for that item is closed, in which case the customer should be shown the winning customer (if any) and their final highest bid.
7. The client should tell a customer when a bid is invalid. Recall that 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.

8. You should provide "quit" and/or "logout" buttons that end the program or parts of the program gracefully.

General

1. All communication between the server and clients should be through `Socket`s.
   - 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.
   - During testing, no Java source file should be shared between the server and customers or between customers. **Make separate Eclipse (or other IDE) projects for the server and customer,** and duplicate files as necessary across projects.
2. 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 uncaught exception thrown.
3. The server's top class (containing `main`) should be called `Server`, and the client's top class should be called `Client`.
4. Follow good style, synchronization, and OOP practices.

Optional Features (25+% of the grade)

For full credit, you must implement at least 25 points worth of optional features in addition to the required features. The table below provides examples, but you can add anything you would like. For example, you may allow customers to search for items in the database using various filters, which would be worth about 2 percentage points. You could also allow automatic bidding, or securely-transmitted credit card payments. Look at sites like eBay for inspiration. Ask the instructors if you would like an estimate of how many points one of your features will be worth.

Points (%)	**Feature**
1	Set a minimum starting price > 0 for every item.
1	Set the duration for the auction for each item separately.
1	Set a high limit that is a 'Buy It Now' price. When a customer bids that amount, he/she gets it right away.
1-2	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	Items could have descriptions that are visible to customers. Could include images.
2	A search feature to search through items, customers, etc.
2-3	Non-volatile history of auctions, customer activity etc.
1	Sound effects.
1-5	Nice GUI in general.
1-3	Count-down clocks for items.
3	Using the Observable class and Observer interface.
3-5	Use a cloud server to host the auction Server.

Points (%)	Feature
2-5	Cryptography techniques – encryption of passwords, password hashing and salting, encryption of messages.
2-4	Use Apache Derby or other database.

Additional Requirements

Canvas/GitHub Instructions

As you start the project, you must follow the instructions on the Canvas assignment page.

First, clone your GitHub repo. Then, you can create two directories inside the repo directory (e.g. `ServerSide/` and `ClientSide/`). When creating your new projects in Eclipse, uncheck "Use default location" and select the new directory you made instead. (You should be able to specify the project location in a similar way in IntelliJ.). Make sure that you have two separate Projects for Client and Server, and not just two packages in one project.

Progress Check

There will be a progress check during recitation, and it will be a quiz grade. Attendance and presentation of your work is mandatory. For full credit, you should have completed about 1/3-1/2 of the work.

You do not need to have packaged your work as you will for final submission. You will just show us your progress by launching the project from your IDE.

Documentation (3% of the grade)

You must provide documentation for your program. The documentation should contain two main parts – one from a programmer's point of view, and the other from a user's point of view. - For the programmer, explain how your code is structured and how to properly start the server. - For the user, explain how to use the client and list the features, including the optional ones you added.

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.

This document could also include a diagram or two. The document should be 2-4 pages long.

Packaging the Project

If you have external libraries, such as GSON, you must make sure that your project will run on our machines easily. You will do this by making a Runnable/Executable JAR file, which will include the external libraries.

Optionally, 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:

- Eclipse
- IntelliJ
- Maven
    - Note: creating executable JAR files with Maven is slightly more complicated. Please also see this post for information on creating a JAR manifest to set the entry point for the JAR.

You will submit **four zip files,** two for the server and two for the client.

To create your first 2 deliverable zip files, do the following twice – once for the server, and once for the client: - Create an executable JAR file by following the instructions above. - Put your JAR file and all other files required for it to run in a zip file, such that unzipping the file and double-clicking the executable JAR will read the initialization file, any sound files, etc. properly. If double-clicking won't work, it is OK, as long as `java -jar your_jar_file` works.

Optionally, you may submit Maven project JAR files instead of the ordinary executable JAR files. If you do, make sure to include all of the necessary components in your zip file (basically, make sure you include your pom.xml file).

You should also turn in 2 zip files – one for the server, and one for the client – **containing only your source code.** This is for us to examine if required. The format of the *contents* of this zip file is not that important; you could, for example, use jar (not executable jar) instead of zip, making sure that you avoid most non-source files.

Checkout

Project checkout will be conducted much like Project 5. 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.

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

If you have any kind of optional feature where your server is hosted somewhere externally, e.g. 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 on where the address is hardcoded so we can go back and change it if need be). We will do our best to evaluate these optional features 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 JAR downloaded from Canvas. If they don't work by double-clicking, you may run

them from the command line using
```
java -jar <executable_jar>
```
Frankly, we expect you may encounter problems with your executable JARs. Try your best.

Suggestions

- Start early.
- Test often.
- Make frequent Git commits to safely store your code.
  - *Note that this is* ***required*** *to some extent: 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. If bugs are stubborn, you can then fall back to a simple GUI.
- Scene Builder can help greatly in making nice JavaFX layouts, but it has a steep learning curve. Unless you used it for Project 5, or can start early, we recommend not using it here.

Suggested Project Flow to get started

Here's a list of required features I think you can safely add one by one:
1. Load item data from a file when the server starts
2. Send item data to client when a client connects to the server
3. Present item data on client GUI
4. Accept bid on client GUI
5. Send bid from client to server
6. Validate bid in server and update item data appropriately
7. Notify all clients of the new bid

This should give you a good start, and establishes two of the biggest use cases (output item data on client, input bids to system). Come by office hours regularly if you want advice on what optional features you want to aim for, or a plan more tailored to you.

Deliverables

- All deliverables must be submitted to [the corresponding Canvas assignment](#) **before the deadline.**
- **Remember to include the contents of the Header file at the top of every .java file you submit and your documentation file.**
- Note: Unlike the automatically graded projects, your file or submission format is not critically important. An instructor should just be able to run your Server and Client, view your source code, and view your documentation with no help from you. Any deviation from the instructions should be recorded carefully in your documentation deliverable. Normally, when we grade, we will have you run the programs after downloading them

from Canvas. So, for example, if you have a lot of trouble getting the JAR files to work, you will not lose all the points as long as you are able to show us how to run your program.

Documentation

- FinalProject_EID.pdf

OR

- FinalProject_EID.txt

Program

- FinalProject_Server_jar_EID.zip
  - Zip file structure:

```
FinalProject_Server_jar_EID.zip (zipped file)
  server.jar
  items.csv (optional)
  resources/ (optional folder)
     item1.png (optional)
     ...
  ...
```

- FinalProject_Client_jar_EID.zip
  - Zip file structure:

```
FinalProject_Client_jar_EID.zip (zipped file)
  client.jar
  resources/ (optional folder)
     sound_effect.wav (optional)
     style.css (optional)
     ...
  ...
```

- FinalProject_Server_code_EID.zip
  - Should include all source (.java) files for the server, but the exact structure is not important.
- FinalProject_Client_code_EID.zip
  - Should include all source (.java) files for the client, but the exact structure is not important.

Changes to this Document

This document is subject to change. Watch Piazza for announcements of changes, which we expect will be minimal.

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
   - Use objOutStrm.writeUnshared() instead of writeObject.
   - 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`.
   - Make sure that the class is in the same package on the server and the client.
4. I get an Exception "Not on FX application thread".
   - Anytime you want to change an aspect of the GUI from another thread, you should put it in a `Platform.runLater()` block. You provide a `Runnable` that just contains the code you want to execute and `Platform.runLater()` will run it on the JavaFX thread.