Distributed Systems

TU856 - 4

Assignment

Twila Habab
C2036521

School of Computer Science
TU Dublin – City Campus

23/11/2023

Table of Contents

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Twila Habab

C20361521

23/11/2023

Application Architecture

This application involves a simple implementation of a client-server architecture, where the Seller instances are the servers and Buyer instances are the clients.

Buyer and Seller instances are threads, so that we can have multiple instances of either class.

Communication between these instances is established through multicasting. In this application, everyone who is joined in the multicasting group can see messages from other instances (including themselves). Parsing of messages and error handling is implemented in order for the messages to be sent and received by the intended sender and/or receiver.

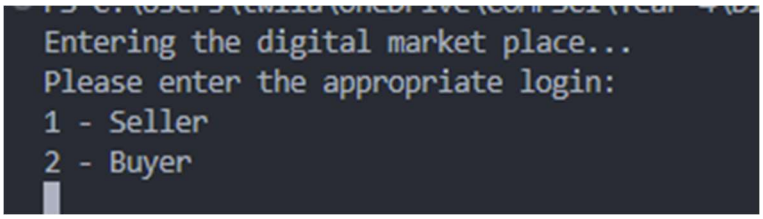
Setup, code directory structure

Code Directory structure

Java files will be in the root directory, and a bin folder is available for the class files.

Main.java

This is the main drive of the system. The user can either choose to login as a Seller or as a Buyer. After choosing the appropriate login, a thread instance is started.

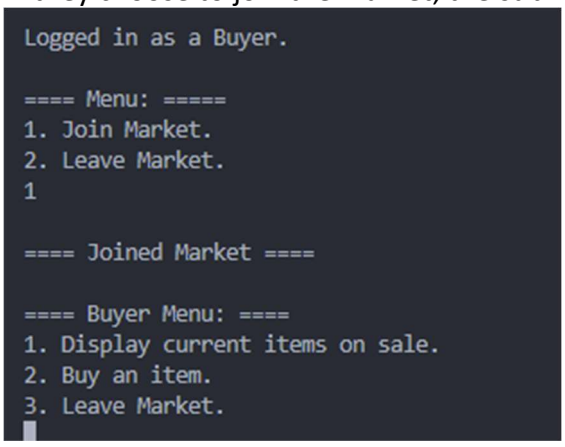


Item.java

This is the Item class for Sellers to use for their own array lists. This contains the node ID of the seller, the product name, and the amount of the product.

Buyer.java

Once the buyer thread is initialized, the user will be greeted with the main menu, to either join or leave the market. If they choose to join the market, the submenu will be displayed:



The buyer will then be able to choose an option to either display the items currently on sale, to buy an item, or leave the market.

If they choose to display the current items on sale:

```
==== Buyer Menu: ====
1. Display current items on sale.
2. Buy an item.
3. Leave Market.
2
Getting items for sale now from buyers....
NodeID: 4267 ProductName: Potatoes, Amount: 5
NodeID of seller: 4267
NodeID of seller: 9240
NodeID: 4267 ProductName: Potatoes, Amount: 5
NodeID of seller: 4267
```

The buyer will receive 5 items before a prompt will be displayed to choose which items they wish to buy. In order for them to buy an item, they have to follow this format:

<NODEID> <AMOUNT>

If purchase is successful, it displays this as the receipt:

```
Buy Item (Press 'q' to Quit): 2907 3
1329 bought 3 of Potatoes from 2907
```

Seller.java

Once the seller thread is initialized, a menu will be displayed:

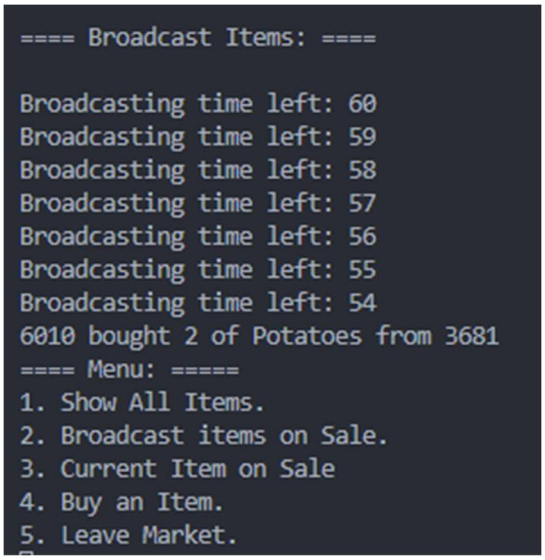
```
Entering the digital market place...
Please enter the appropriate login:
1 - Seller
2 - Buyer
1

Logged in as Seller.

==== Menu: =====
1. Show All Items.
2. Broadcast items on Sale.
3. Current Item on Sale
4. Buy an Item.
5. Leave Market.
```

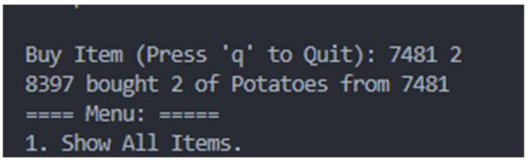
The seller has two options, either to sell (broadcast items) or to buy items (from other sellers). In this application, the buyer **can not** buy **and** sell at the same time.

If the seller chooses to **broadcast** the item, it will have a 20 second time limit of broadcasting. Whilst broadcasting it will listen to buyers if they want to buy their current item on sale. Once 20 seconds is up **or** a buyer buys their item, it will return the user to the menu:



The item to be broadcasted changes every **60 seconds spent broadcasting** or when the item is sold out.

Sellers can also buy items from other sellers and follows a similar format to the buyers:



How to run the system on Windows

Compile application:

Open a terminal in the folder.

This compiles all the files into a folder called “bin”:

javac -d ./bin Buyer.java Item.java Main.java Seller.java

Run application:

java -classpath ./bin Main

Have multiple instances of Sellers and Buyers:

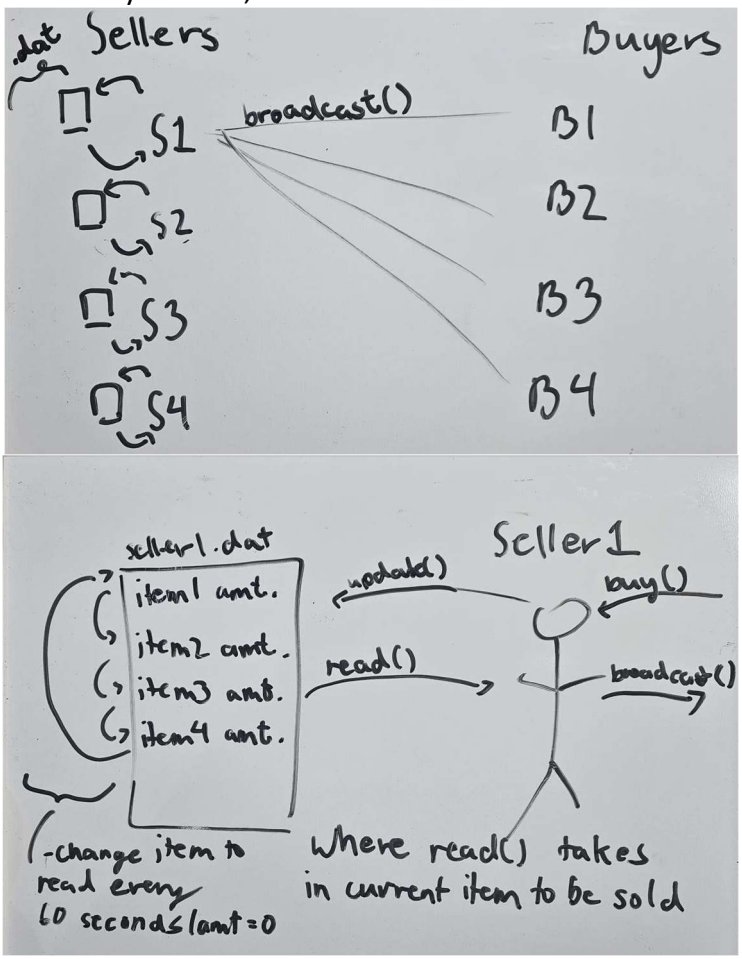
Please open up a new terminal and run the application again.

To force close the application:
If by chance the thread seems to have stopped working, force close the terminal by inputting (**CTRL + C**) or manually close the terminal. Then run it again.

Alternative design

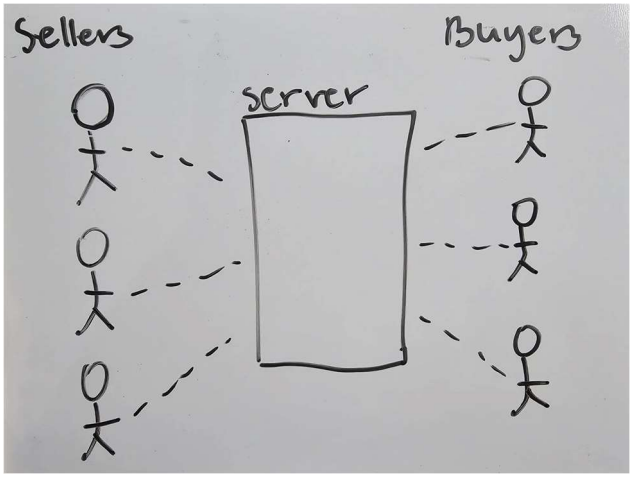
Serialization:

The system could have it so that sellers will serialize their own array lists into a file where the file is named after the seller's node ID (i.e., 5331.dat). Every time they wish to broadcast to their buyers (and other sellers), they will read into the file and broadcast it. When a buyer wishes to buy an item, the seller can then write into their own serialized file.



Centralized Server:

Another alternative solution is to have a centralized server that handles all the incoming and outgoing messages. This keeps the code **DRY** and handles/prevents errors more efficiently.



Centralized Server AND Serialization (server caching):

Another more efficient alternative is to implement these two methods together. Each seller will have their own serialized file and the server will have its own. Essentially the sellers will write into the server's serialized file to update which item is now being currently broadcast, whereas the server will then read onto the file to broadcast the items.