CSE 460

Programming Assignment Report

Posting ID: [5608-657]

[04/22/2019]

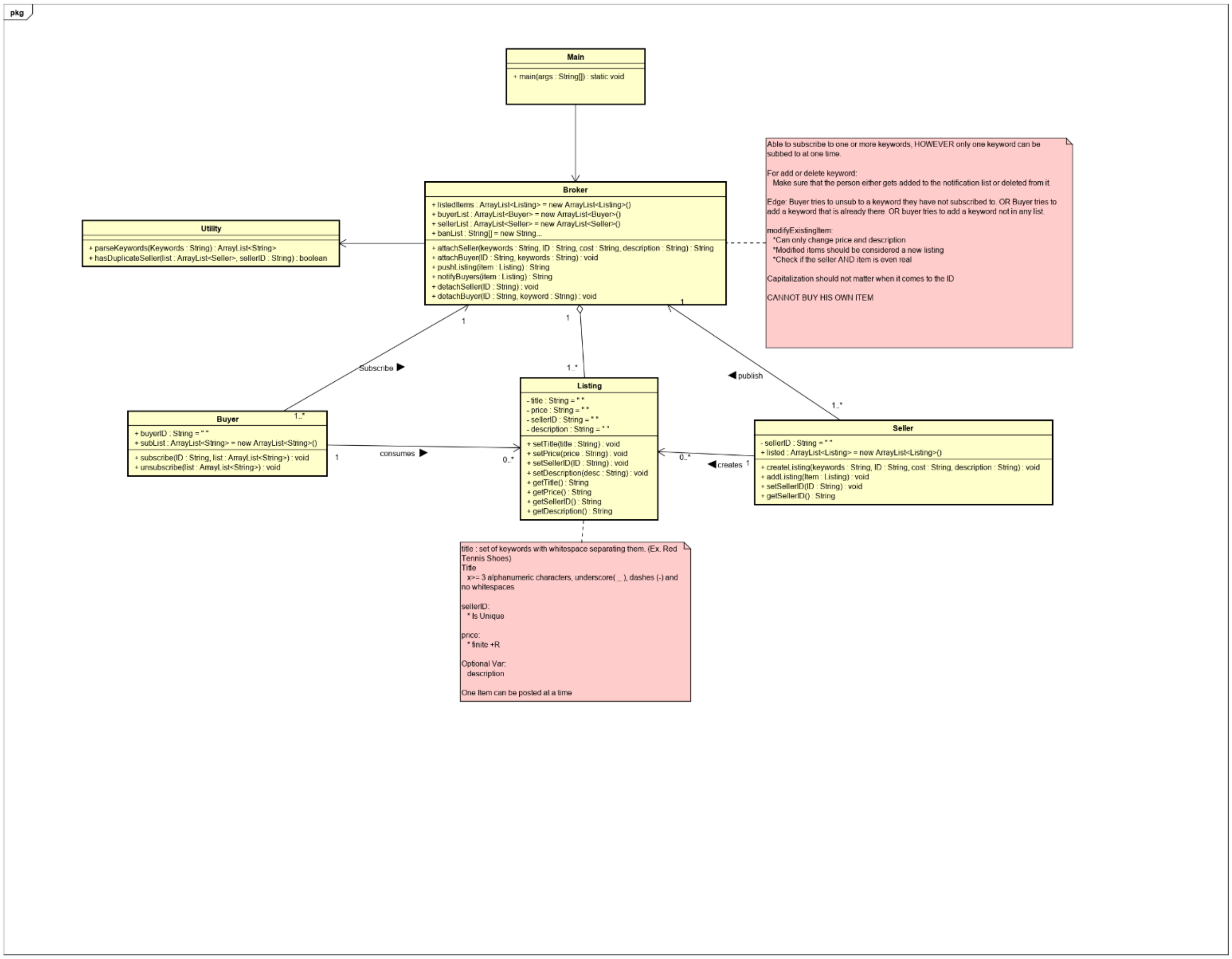
Spring 2019

1. Assumptions

I assumed that there will be no non-format following queries given as was told in one of the discussion points. I also assumed a couple of tests will be for some edge cases:

- Same Seller Name
- Negative Price
- Buyer Tries to subscribe to already subscribed word
- Buyer unsubscribes to a non-subscribed word
- Unregistered buyer tries to subscribe or unsubscribe

2. Class Diagrams


pkg
Main
+ main(args : String[]) : static void
Broker
+ listedItems : ArrayList<Listing> = new ArrayList<Listing>()
+ buyerList : ArrayList<Buyer> = new ArrayList<Buyer>()
+ sellerList : ArrayList<Seller> = new ArrayList<Seller>()
+ banList : String[] = new String...
+ attachSeller(keywords : String, ID : String, cost : String, description : String) : String
+ attachBuyer(ID : String, keywords : String) : void
+ pushListing(item : Listing) : String
+ notifyBuyers(item : Listing) : String
+ detachSeller(ID : String) : void
+ detachBuyer(ID : String, keyword : String) : void
Utility
+ parseKeywords(Keywords : String) : ArrayList<String>
+ hasDuplicateSeller(list : ArrayList<Seller>, sellerID : String) : boolean
1
1
1
Subscribe ▶
1..*
◀ publish
1..*
Buyer
1..*
+ buyerID : String = " "
+ subList : ArrayList<String> = new ArrayList<String>()
+ subscribe(ID : String, list : ArrayList<String>) : void
+ unsubscribe(list : ArrayList<String>) : void
1
consumes ▶
0..*
Listing
- title : String = " "
- price : String = " "
- sellerID : String = " "
- description : String = " "
+ setTitle(title : String) : void
+ setPrice(price : String) : void
+ setSellerID(ID : String) : void
+ setDescription(desc : String) : void
+ getTitle() : String
+ getPrice() : String
+ getSellerID() : String
+ getDescription() : String
0..*
◀ creates
1
Seller
- sellerID : String = " "
+ listed : ArrayList<Listing> = new ArrayList<Listing>()
+ createListing(keywords : String, ID : String, cost : String, description : String) : void
+ addListing(Item : Listing) : void
+ setSellerID(ID : String) : void
+ getSellerID() : String

Able to subscribe to one or more keywords, HOWEVER only one keyword can be subbed to at one time.

For add or delete keyword:
Make sure that the person either gets added to the notification list or deleted from it.

Edge: Buyer tries to unsub to a keyword they have not subscribed to. OR Buyer tries to add a keyword that is already there. OR buyer tries to add a keyword not in any list.

modifyExistingItem:
*Can only change price and description
*Modified items should be considered a new listing
*Check if the seller AND item is even real

Capitalization should not matter when it comes to the ID

CANNOT BUY HIS OWN ITEM

title : set of keywords with whitespace separating them. (Ex. Red Tennis Shoes)
Title
x>= 3 alphanumeric characters, underscore(_), dashes (-) and no whitespaces

sellerID:
* Is Unique

price:
* finite +R

Optional Var:
description

One Item can be posted at a time

This program takes an input from the user that is assumed to be an input.txt and an output.txt file that will be used for automation testing. After creating an instance of the broker, the user inputs are pushed into the broker telling it what the user wants to do. The buyer class consumes a listing and subscribes to the broker given certain keywords and a unique buyer id. The seller creates a listing and publishes them to the broker and then the broker takes these new listings and notifies the proper buyers. The broker notifies the buyers by checking the databases that are being kept in the one instance of the broker.

Class Main:

The Main class handles all the user inputs and communicates ONLY with the broker. The communication only includes inputs to the instance broker with the given input. The main calls on certain broker functions depending on the user input.

Class Broker:

The class broker has an instance in main. The broker has a database of all listed items, sellers and buyers within it. This class is only instanced one time since there can only be one broker in a single run. The broker communicates to all the sellers and buyers. If there is a new listing or buyer then it creates an instance of a buyer or seller depending on the situation. If there is a new listing then it sends out a notification based on which buyers are subscribed to.

Class Utility:

This is a simple utility class that has a couple helper functions that I made. This is used inside the Broker class only.

Class Buyer:

This class contains the subscribers ID and the keywords that they are subscribed to. The buyer class is an instantiable class that is communicating ONLY with the Broker class but is using the listing class. It functions that are called up by the broken depending whether or not this instance of the buyer wants to subscribe or unsubscribe to a keyword.

Class Seller:

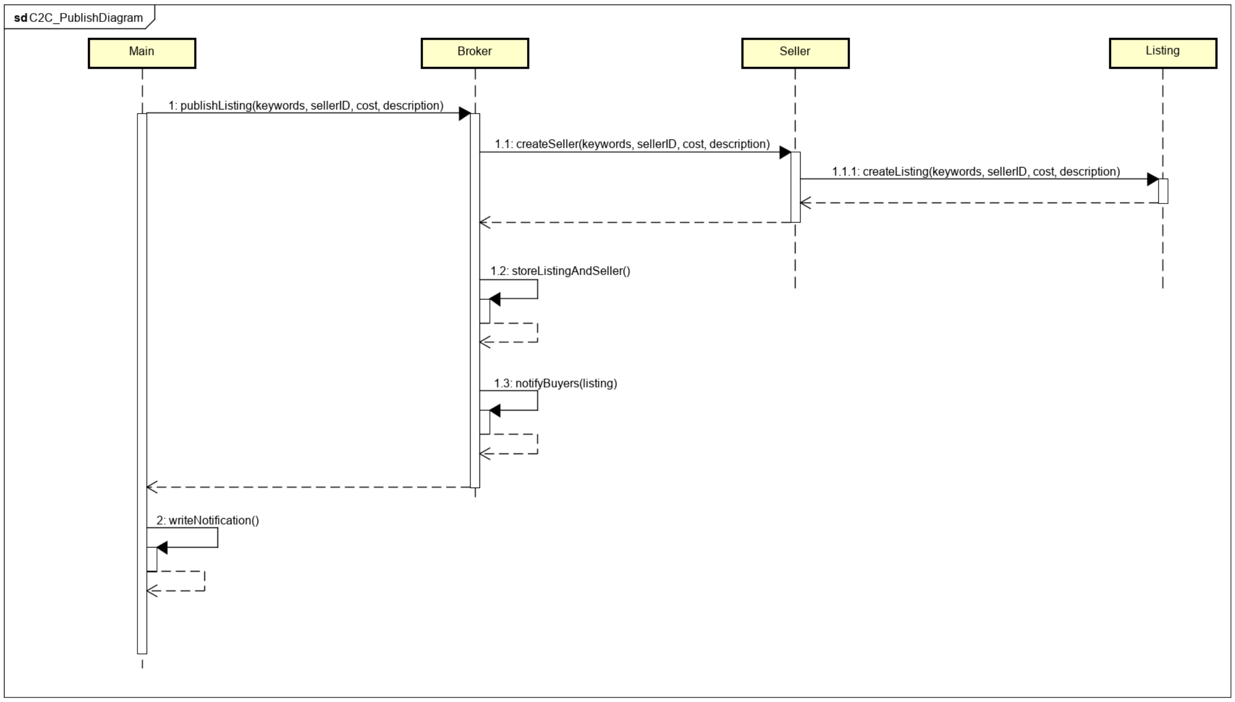
This class contains the sellers ID and listed items. The seller class is communicating ONLY with the broker class and it uses the listing class. This class has functions that help the broker instantiate a new seller or simple add to the current sellers listed items.

Class Listing:

This is a simple listing class that is used by Broker, Buyer and Seller as the main event that is being created or consumed depending on the situation.

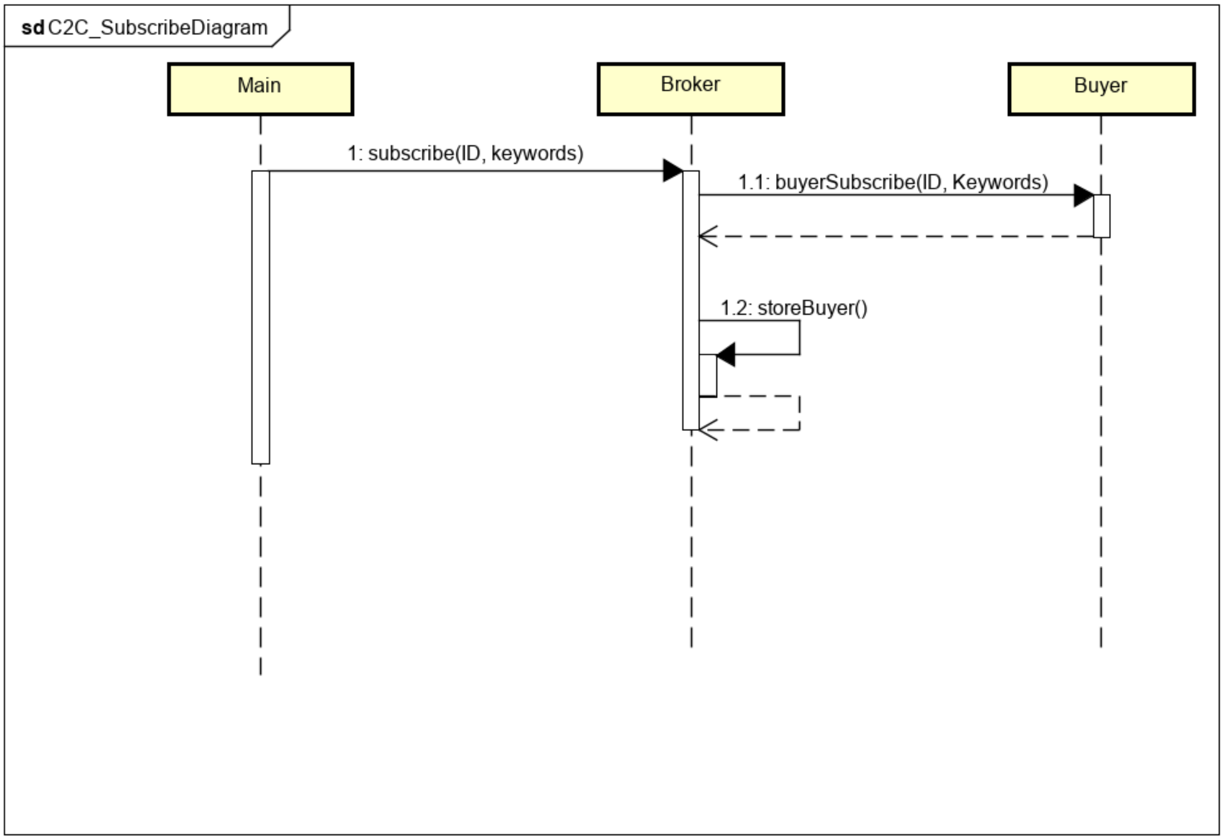
3. Sequence Diagrams

For all sequence diagrams, main is assumed to take input first always.


sd C2C_PublishDiagram
Main
Broker
Seller
Listing
1: publishListing(keywords, sellerID, cost, description)
1.1: createSeller(keywords, sellerID, cost, description)
1.1.1: createListing(keywords, sellerID, cost, description)
1.2: storeListingAndSeller()
1.3: notifyBuyers(listing)
2: writeNotification()

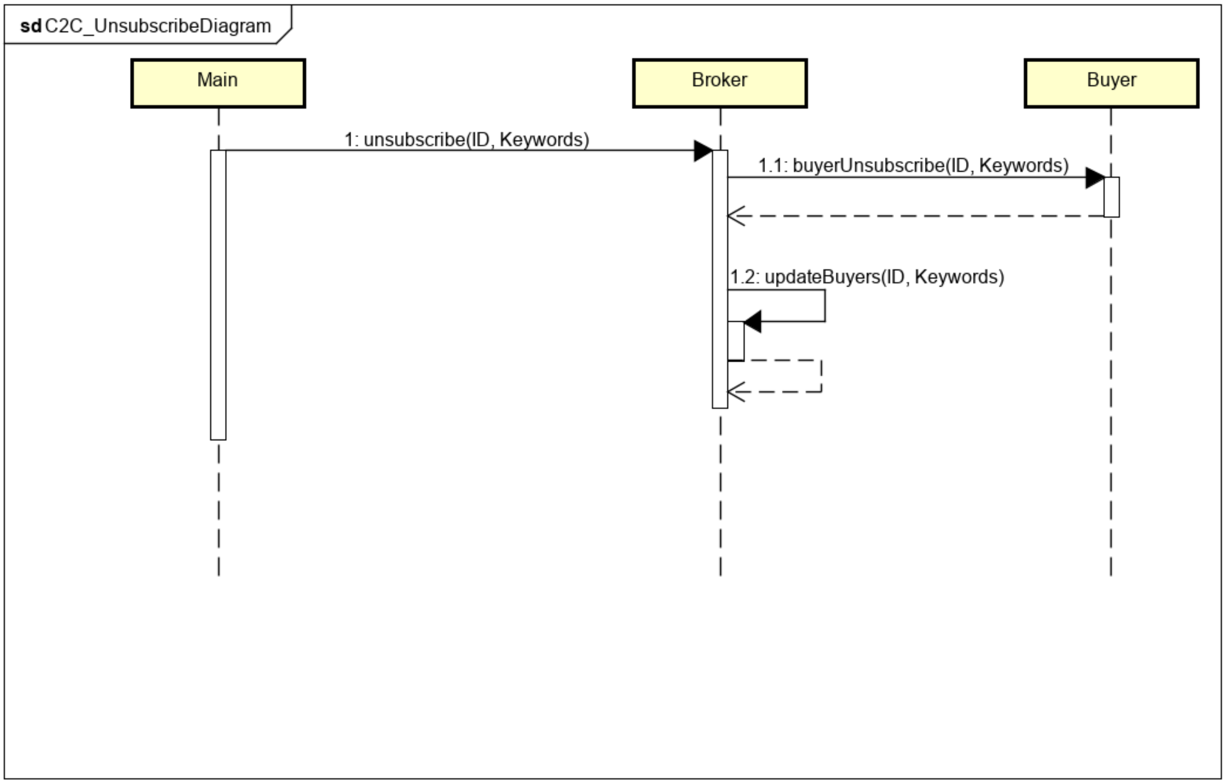
Publish Listing Sequence

This sequence above shows how the code interacts with the broker and seller class when a publish listing is called. After the input is taken, main sees that publish listing has been called, it passes in the keywords, sellerID, cost and description given by the input to the broker. The broker then creates a seller instance with all the information, in the process creating a new listing. The seller class then returns a seller instance with all the information. The broker then stores this information into the database it holds. Then the broker finally notifies all the buyers that are subscribed to the given keywords. Finally the broker class returns a string of notification to the main and the main writes this notification to the output file.



Subscribe Sequence Diagram

The diagram above shows how the system interacts with each other after a subscribe input is found. The main calls the broker instance with a subscribe call and passes in the buyerID and the keywords from the input. The broker then calls the buyer class to create a new instance of the buyer with the buyerID and the keywords. After this the buyer class then returns the buyer instance and then the Broker stores the new buyer instance into the database it holds for future use.


sd C2C_UnsubscribeDiagram
Main
Broker
Buyer
1: unsubscribe(ID, Keywords)
1.1: buyerUnsubscribe(ID, Keywords)
1.2: updateBuyers(ID, Keywords)

Unsubscribe Sequence Diagram

The sequence diagram above shows how the system reacts to an unsubscribe call. The main calls the broker instance with the given buyerID and the keywords that are needing to be unsubscribed to. The broker class then finds the buyer class and then calls that buyer with the unsubscribe call. Then the returns the updated buyer instance. Then the broker will update the database.

4. Readme

How I run the java:

- Put all the java files into the same folder.
- Put all the input files into the same folder as the java files.
- Compile the javafile
  - javac Main.java
- Call the java with the input txt and desired output file
  - java Main “Input.txt” output.txt

5. Appendix & Credits

Java Documentation – Used to clear up how to use certain java utils

https://docs.oracle.com/javase/7/docs/api/