# COL 106: Assignment 2
# Sem-I: 2019-20
# Due Date: 19.08.2019 (Monday) 11:55 P.M.

## Problem

Suppose you have decided to build your own online selling platform known as *Amazing.com*. You have multiple sellers and buyers registered with *Amazing.com*. It has open access for both buyers and sellers. The sellers add the items to the catalog and once the buyers make a purchase from the catalog, the corresponding item is removed from the catalog. For the sake of simplicity, you may assume all the items are of a single type such as books or shoes. You listed some sellers as *"preferred"* sellers who have donated you some money. The *preferred* sellers have been given a priority order based on the amount they have donated you. When buyers make a purchase, they would always be delivered an item listed by a seller who is higher in the priority order. For example assume sellers $s_1, s_2$, and $s_3$ have listed their books in the catalog. Also assume the priority order as $s_1 > s_2 > s_3$ ($s_1$ is most preferred). Now buyers have to first purchase all the items listed by $s_1$ before making a purchase from other sellers. Further, consider a situation where a buyer is buying from $s_2$ after all the items listed by $s_1$ exhausted. In the meantime $s_1$ again adds some items to the catalog. Once the current purchase from $s_2$ completes, the next buyer must buy the item(s) listed by $s_1$ before buying from any other seller. You may assume that the priorities are unique (no duplicates) and non-negative. In the starter code, the priorities are given as simple integers. Assume the lower number to be of higher priority. Being a student you're on a budget and can't afford expensive infrastructure. As a result, you have very limited space for the catalog. At any time at most **N** items can be stored in the catalog. If the catalog is already full, then the sellers need to wait for some buyers to buy items before they can add their product to the catalog. Also, if the catalog is empty, then the buyers need to wait until a seller adds their items to the catalog. Your task is to implement the `Seller` and `Buyer` classes using Java Threads.

## Hint

You can model this question as to the classical *Producer-Consumer* problem. The sellers are the producers who produce items to the shared catalog and the buyers are the consumers who consume items from the catalog. As you can guess, since the buyers always buy from the preferred sellers, the catalog data-structure should be implemented using a *Priority Queue*. For the design of the system, each of the sellers takes an item from the shared queue called inventory and put it in the catalog. The buyers get (remove the item with the highest priority) items from the catalog. As we are using multiple buyer and seller threads all the operations have to be thread-safe (i.e., properly synchronized using concurrency primitives such as lock and conditions). Remember the thread-safe operation always happens on the shared resources. There are two shared resources in this design, the inventory list (shared between multiple producers) and the catalog priority queue

(shared between producers and consumers). Any operation on these data structures should be properly synchronized.

## Starter Code

Download the `Assignment2.tar.gz` and extract it using the following command,

```
tar -xvf Assignment2.tar.gz
```

Inside the extracted `Assignment2` folder you would find the `Bases.java` file which describes the abstract base classes and the interfaces that you would be implementing.
You are not supposed to change anything in the `Bases.java` and the `Item.java` files. Read the comments in the files to understand which parts you need to implement. You need to complete the implementations of the `Node, Queue, PriorityQueue, Buyer, and Seller` classes. The signatures and partial implementation for these classes are given to you. After you have implemented these classes, complete the `Assignment2Driver` class. Please go over the starter code thoroughly and make sure you understand the design. The parts that you need to complete are marked as **TODO**. You don't need to alter any other part of the starter code.

## Compilation and Testing

You can use the makefile for compilation and running the code. To compile your code, use

```
make compile
```

To run the code, use

```
make run
```

To clean the class files, use

```
make clean
```

There is a sample test case given to you inside the `tests/input.txt` file. The `tests/sample_out.txt` contains the expected output. You can also use the python script to check the correctness of your file. Run the `runtest.py` file, and you would see whether your implementation passes the given test case or not.

```
python3 runtest.py
```

## Submission

- You need to put all your source files (.java files) into a directory named *src*. You have to compress this directory (to zip format) and rename the zip file in this format:

  ```
  your-entry-number_assignment2.zip
  ```

  Example: If your entry number is 2012CSZ8019, the zip file should be named 2012CSZ8019_assignment2.zip. It should expand to a directory called *src*, which contains all your .java files. Please make sure that you follow exactly this naming format.

- Next, convert the zip file to base64 format. On any linux system, you can do this easily by issuing the following command on the terminal :

  ```
  base64 entrynumber_assignment2.zip > entrynumber_assignment2.zip.b64
  ```

  This will create a file with the same name with a b64 extension appended at the end. Upload this b64 file on moodle and submit it (in the "Submission" section).

- Alternatively, you can use the makefile to create the submission base64 file. Use the following command,

  ```
  make handin ENTRY=your-entry-number
  ```