1. Consider two threads "playing" a game of ping-pong.  In this game, each thread hits the ball by printing "PING" and "PONG," respectively.  Write a threaded application to ensure that PING and PONG always print in order -- that is, no thread can hit the ball more than once without the other thread hitting the ball.    The output must be "PING" "PONG" "PING" "PONG"...

Moreover, starvation must not occur - that is, the threads must not "stop playing" and cease output

1. A library system has a certain number of each book available for checkout. The checkout can be performed using a remote client. Implement the server and the client for this system. The server interface allows to query for all available books, try to checkout a particular book (this request also specifies whether to wait for it to become available or not, the wait is blocking), and return a previously checked out book. To test out this system, create several clients that randomly request books, ‘read’ them for a random period of time, and then return them
2. Write a Java program that spawns *n* threads, where *n* is a program argument. These threads access a shared counter (initialized as 0) in a loop. In each iteration, they read the counter to a local (stack) variable, increment it, and store it back to the counter. When all threads complete 10000 iterations **each**, the program stops and prints the value of the shared counter.

Note that the final value may be smaller than the total number of iterations. Run the program on 1, 4, 8 and 16 threads and report the results in a textual table in the “doc.txt file”.

In your report, include the run time in milliseconds.

1. **Search** - Your search mechanism will take as input a multi-word query and will return a list of documents where the given words appear.

The easiest solution is to simply return a list of documents where any of the words appear. Another approach is to give greater weight to (by placing earlier in the result set) documents where more than one of the query words appears. Yet another option is to give greater weight to the documents where the words specified appear closer to one another. For example, if you search for "computer science" and document 1 has *computer* at position 1 and *science* at position 2 while document 2 has *computer* at position 1 and *science* at position 100, document 1 would be given higher weight.

Your program will take as input the directory as before along with a file containing a set of queries. Each line of the file will contain a multi-word query. The output of your program will be text file *results.txt* that contains the result of running each query.

**Thread Pool** - You will use a thread pool or work queue and process up to 10 text files in parallel. As your program traverses the directory specified by the user, for each txt file found insert a new job into the queue.

1. To accomplish this, you will need to implement a work queue and a locking mechanism for your inverted index data structure.

2. Your locking mechanism will ensure that only 1 thread may change the inverted index at a time. Multiple threads may read data in the inverted index simultaneously.

1. There are multiple client terminals that share one printer. Each client terminal maintains a queue of files to print. The shared printer can hold and print only one file each time. Thus, the printer should notify the terminals of the fact that the printer is available once the last printing job is finished. Afterwards, a terminal that occupies the printer through competition can request the printer to print the remaining file in its queue. The printer should be shared in a fair manner. Thus, one terminal should not keep occupying the printer. Develop a Java implementation that simulates such a situation. The main method creates three threads for client terminals and another thread for a shared printer. Note that the printer thread does not terminate on purpose. After running this simulation, the following two requirements should be fulfilled. 1. All the files belonging to three client terminals must be printed out successfully. 2. No client should keep occupying the printer. Therefore, if the first client uses the printer, then the next time the second or the third client must use the printer unless the first client is the only client trying to use the printer and the rest of the clients already finished their jobs.
2. Write a simulation program for the fruit market. The farmer will be able to produce different types of fruits (apple, orange, grape, and watermelon), and put them in the market to sell. The market has limited capacity and farmers have to stand in a queue if the capacity is exceeded to sell their fruits. Consumers can come to the market any time and purchase their desired fruits; and if the fruits they want to buy runs out, they are willing to wait until the supply of that kind is ready