Lab 10

A simple barrier synchronization pattern

Considering a critical section and N threads, let's assume we do not want to let any thread to execute it more times than the other threads, informally meaning that we desire to achieve a level of fairness concerning the access to the critical section. This would mean that we guarantee that one thread will not wait indefinitely, being bypassed by the others, until it will enter the critical section. For ensuring the mutual exclusion in the critical section we assume that a mechanism is already enforced, like a normal lock for example:

```
while (true) {
        CS.lock();
        CS
        CS.unlock();
}
```

Considering the pseudocode above the loop for one thread we detail the next simple mechanism to enhance it in order to achieve the desired restrictions mentioned above, for N threads:

```
while (true) {
        CS.lock();//We want to deny the
        CS//bypassing possibility between
        CS.unlock();//threads for this code section

        B.lock();
        passedthreads++;
        if (passedthreads == N) {
                exitdoor = false;
                entrancedoor = true;
        }
        B.unlock();
        while (!entrancedoor){};

        B.lock();
        passedthreads--;
        if (passedthreads == 0) {
                entrancedoor = false;
                exitdoor = true;
        }
        B.unlock();
        while (!exitdoor){};
}
```

Here entrancedoor is initialized as *false* and exitdoor as true; passedthreads starts at 0

Using a barrier to delay thread execution

In the previous section an approach was presented that was used to "trap" all the threads between two positions in the code. In some cases you might just want to delay the execution of your threads until all of them reach a certain point. The most common example would be to delay the threads until all of them are started. For this case a simpler approach than the one presented might be effective:

```
// an integer that must be shared by all threads
public static AtomicInteger mySyncPoint = new AtomicInteger(0);
public static int numberOfThreads;
...
public void run() { // the run method of one thread
        ...
        mySyncPoint.addAndGet(1);
        // waiting on this point until all the threads increment
        while (mySyncPoint.get() < numberOfThreads) {};
        ...
}
```