

Solution to exercise 2

Buffer

The *getLine* method is simply written symmetrical and analogous to the *putLine* method.

```
class Buffer {
    // Attributes and constructor...

    synchronized void putLine(String inp) {
        // According to exercise...
    }

    synchronized String getLine() {
        try {
            while (available==0) wait();
        } catch (InterruptedException exc) {
            throw new RError("Buffer.getLine interrupted: "+exc);
        };
        String ans = buffData[nextToGet];
        buffData[nextToGet] = null;
        if (++nextToGet >= size) nextToGet = 0;
        available--;
        notifyAll();
        return ans;
    }
}
```

Note that:

- The *synchronized* keyword tells the Java compiler to produce code that contain a *mutex* and data is unlocked when returning to the callee, even if the *return* is placed in the middle of the method and deeply nested in *if* statements, etc.
- For more than one producer, or more than one consumer, we need *while* for the *wait* even if we only use *notify* (not *notifyAll*). Make sure that is understood.
- With only one producer and one consumer, it would work with *if...wait* and *notify* instead of *while...wait* and *notifyAll*. But as a general rule we always write *while*.
- We could have used our variables to test and only issued *notify* or *notifyAll* when really necessary, that is, when there are threads waiting. On the other hand, if no threads are waiting, *notify* is almost only a test.
- To notify specifically for *buffer not empty*, or for *buffer not full*, we could have additional internal objects and use their event/condition/notification queue. That is, however, a bit complicated to accomplish in Java. In this course we keep it simple by only using notification for the object itself.

Queue system

1. What data is needed in the monitor?

- counter to give a new customer a unique number, and a counter for the next customer to serve.
 - data to decide if the queue is empty
 - information about how many clerks there are and which clerks that are not busy.
2. What conditions must be fulfilled to permit the monitor operations to be performed?
 - `customerArrived` only registers a new customers and can always be completed.
 - `clerkFree` registers an available clerk and can always be completed.
 - `getDisplayData` requires at least one customer in line and a clerk must be available.
 3. What events will cause the conditions to be fulfilled?
 - The conditions in `getDisplayData` will be fulfilled when a customer arrives and there is a clerk free.
 4. What happens if an eager clerk pushes its button several times when the queue is empty?
 - The number of free clerks may only be increased when the clerk changes from busy to free and not for each button press.
 5. Write the monitor!
 - Implementation follows:

```
class YourMonitor {
    private int nCounters;
    private int lastCustomerArrived = 99; // Start with zero after inc.
    private int lastCustomerServed = 99; // - " -
    private int nrOfFreeClerks; private int lca; // Last counter assigned.
    private boolean[] counterFree;

    YourMonitor(int n) {
        nCounters = n;
        counterFree = new boolean[n];
    }

    /**
     * Return the next queue number in the intervall 0...99.
     * There is never more than 100 customers waiting.
     */
    synchronized int customerArrived() {
        lastCustomerArrived = (lastCustomerArrived+1) % 100;
        notifyAll();
        return lastCustomerArrived;
    }

    /**
     * Register the clerk at counter id as free. Send a customer if any.
     */
    synchronized void clerkFree(int id) {
        if (!counterFree[id]) { // Ignoring multiple button pushes.
            nrOfFreeClerks++;
            counterFree[id] = true;
            notifyAll();
        }
    }

    /**
     * Wait for a free Clerk and a waiting Customer, then return the queue
     * number of next customer and the counter number of the clerk.
     */
    synchronized DispData getDisplayData() throws InterruptedException {
        while ( lastCustomerArrived==lastCustomerServed || nrOfFreeClerks==0 ) wait();
        while (!counterFree[lca]) lca = (lca+1)%nCounters; // Fairness.
        DispData ans = new DispData();
        ans.counter = lca;
        counterFree[lca] = false;
        nrOfFreeClerks--;
        ans.ticket = lastCustomerServed = (lastCustomerServed+1)%100;
        return ans;
    }
}
```