CS431: Assignment-1 Report

Umang(170101074)

Question 1

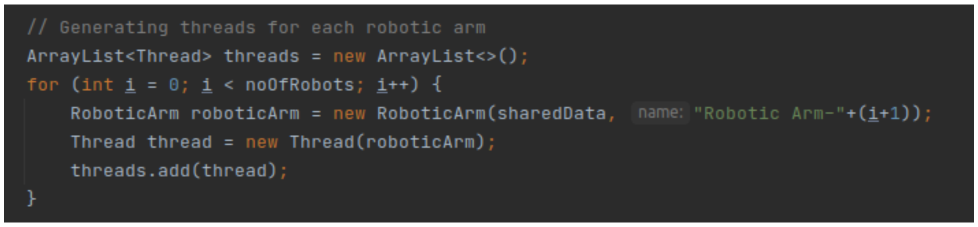
a) Role of concurrency and synchronization in the system

There is a heap of socks and several robotic arms are working concurrently and can pick up socks from it at the same time and pass it to the matching machine. Apart from the robotic arm, matching machine and shelf manager robot are also working simultaneously. Thus the system should be concurrent.

Socks from the pile are being picked up by several simultaneously, but no two robotic arms should pick up the same sock. Also, the variables that store the status of sock pairs ie. the variables that store whether a sock of any particular colour is present in the matching machine or not, should not be updated at the same time by different robotic arm threads. Thus the updating of status and passing the pair to shelf manager should be done synchronously. Hence synchronization is needed.

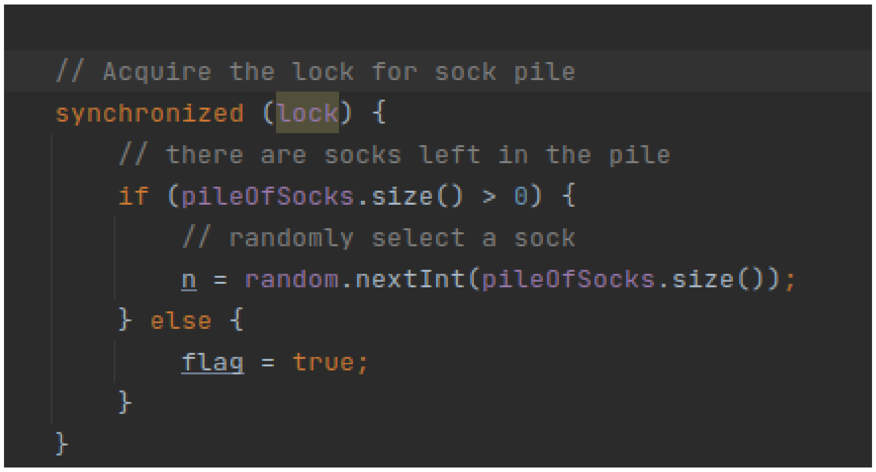
b) How is it handled?

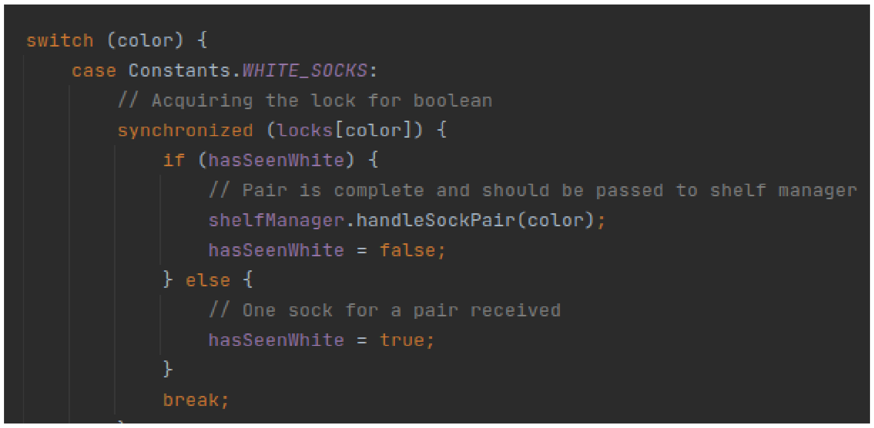
To handle concurrency threads for each robotic arm are created. These threads are executed concurrently thus achieving concurrency by multithreading.



```java
// Generating threads for each robotic arm
ArrayList<Thread> threads = new ArrayList<>();
for (int i = 0; i < noOfRobots; i++) {
    RoboticArm roboticArm = new RoboticArm(sharedData, name: "Robotic Arm-"+(i+1));
    Thread thread = new Thread(roboticArm);
    threads.add(thread);
}
```

To handle synchronization, *synchronized* blocks are used. Thus the shared resources, the pile of socks and the state variables in matching machine, are accessed using synchronized blocks giving access to one thread at a time.


```
// Acquire the lock for sock pile
synchronized (lock) {
    // there are socks left in the pile
    if (pileOfSocks.size() > 0) {
        // randomly select a sock
        n = random.nextInt(pileOfSocks.size());
    } else {
        flag = true;
    }
}
```


```
switch (color) {
    case Constants.WHITE_SOCKS:
        // Acquiring the lock for boolean
        synchronized (locks[color]) {
            if (hasSeenWhite) {
                // Pair is complete and should be passed to shelf manager
                shelfManager.handleSockPair(color);
                hasSeenWhite = false;
            } else {
                // One sock for a pair received
                hasSeenWhite = true;
            }
            break;
```

Question 2

a) Importance of concurrency

The evaluation is being done by CC, TA1 and TA2. If the system is not concurrent, each person would have to wait till the other person is done with their changes. Even if they want to update the record for different students there will be waiting. This can be overcome by concurrency. TA1, TA2 and CC can operate simultaneously to update the records of students.

b) What are the shared resources?

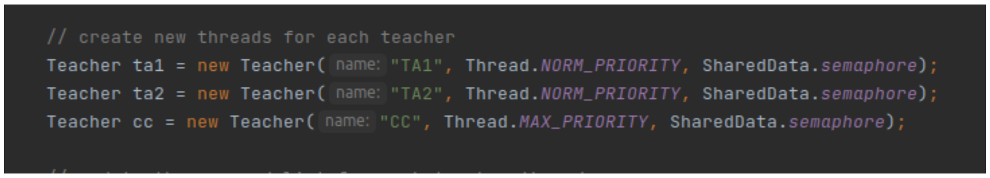
The student data files, thus the data stored in the system, are shared resources as they can be simultaneously accessed by persons on different threads.

c) What may happen if synchronization is not taken care of?

If we do not take care of synchronization, data might not get updated properly. Consider the case where the threads for TA1 and TA2 are operating simultaneously. Since they have the same priority, they can access the shared resource at the same time. If the marks of a student named Amit are 75, and TA1 tries to increase 5 marks and TA2 tries to decrease 6 marks. When both TA1 and TA2 access the resource at the same time, both get Amit's marks as 75. TA1 updates to 80 and TA2 updates it to 69. Depending on the order of their execution, Amit's marks will either be updated to 80 or 69. But the marks should have been updated to 74. Such cases can be avoided by synchronization.

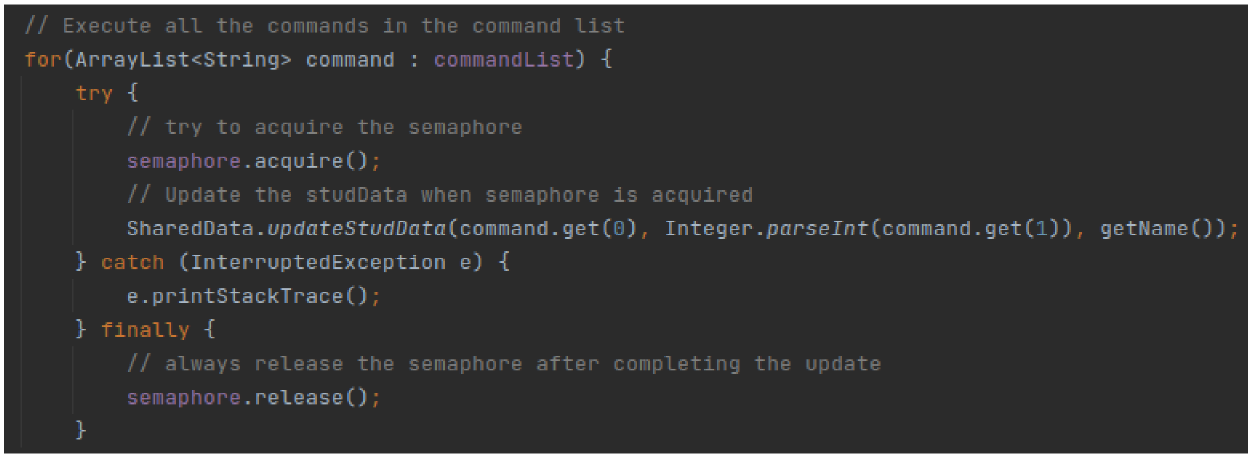
d) How are concurrency and synchronization handled?

To handle concurrency, different threads for TA1, TA2 and CC have been created. So the updates by them can be handled concurrently by multithreading. The priority of CC is set to maximum value whereas priorities for TA1 and TA2 are set to normal. So the updated by CC are being prioritized.



```
// create new threads for each teacher
Teacher ta1 = new Teacher( name: "TA1", Thread.NORM_PRIORITY, SharedData.semaphore);
Teacher ta2 = new Teacher( name: "TA2", Thread.NORM_PRIORITY, SharedData.semaphore);
Teacher cc = new Teacher( name: "CC", Thread.MAX_PRIORITY, SharedData.semaphore);
```

Synchronization is handled using *semaphore.* For accessing the shared resources, the thread needs to acquire the semaphore so that only one thread has access at one time preventing inconsistencies in the data. Following this, updates can be made to it.



```
// Execute all the commands in the command list
for(ArrayList<String> command : commandList) {
    try {
        // try to acquire the semaphore
        semaphore.acquire();
        // Update the studData when semaphore is acquired
        SharedData.updateStudData(command.get(0), Integer.parseInt(command.get(1)), getName());
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        // always release the semaphore after completing the update
        semaphore.release();
    }
```

Question 3

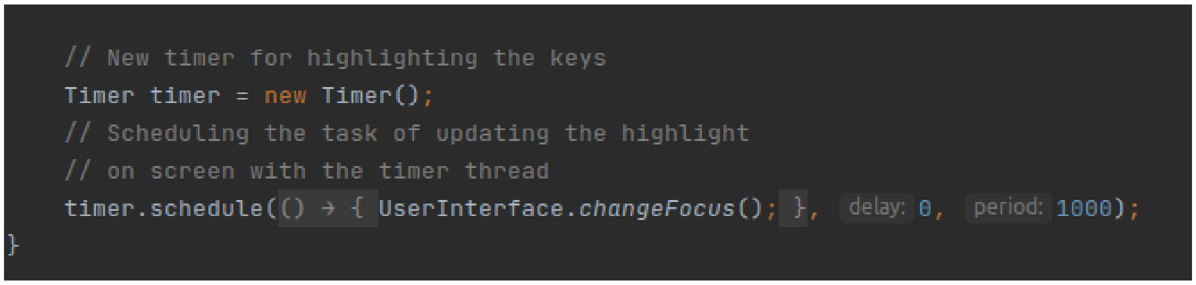
a) Need for concurrency and synchronization

We need to highlight the buttons on the calculator periodically by changing the colour. Thus this highlighting needs to perform parallel to the task of calculation of the result. This can be done by executing the highlighting task on a separate thread than the main thread. Hence concurrency is needed.

While highlighting the indices of currently highlighted buttons will be updated periodically. These indices will also be used when a key will be pressed to get the current highlighted value. Thus synchronization is needed. If synchronization is not used, the highlighted button might be updated before it can be used to get the selected value.

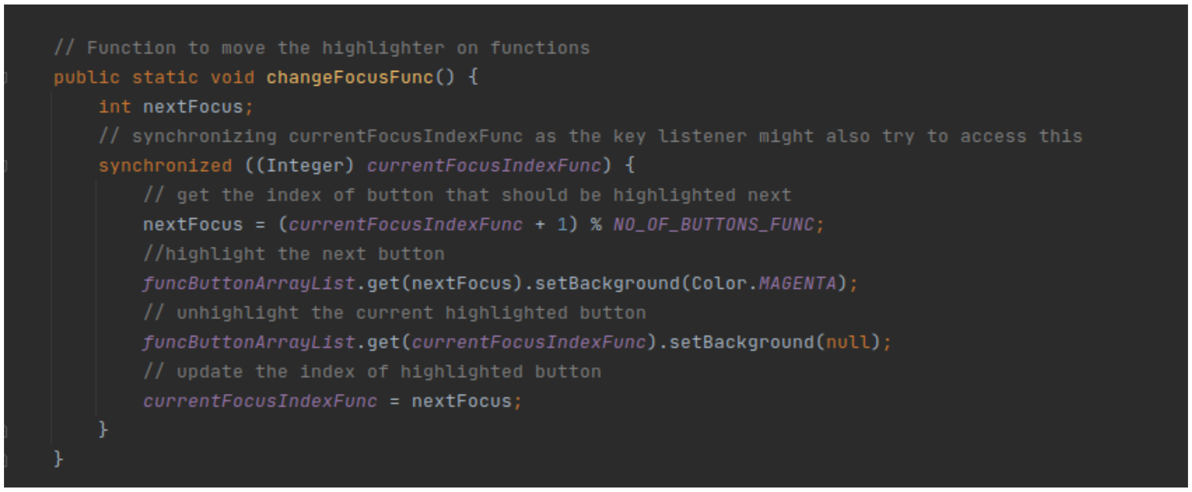
b) How is it handled

Concurrency is achieved by the use of *Timer* class. Timer class provides a method call to schedule task using which we can run a block of code after some regular instant of time. Each timer object is associated with a background thread. Highlighting task is scheduled with timer and thus, it happens on the background thread. For the first version, only one timer object is used which oscillates the highlighter between number keys and function keys whereas for the second two separate timers are used.



```
    // New timer for highlighting the keys
    Timer timer = new Timer();
    // Scheduling the task of updating the highlight
    // on screen with the timer thread
    timer.schedule(() → { UserInterface.changeFocus(); }, delay: 0, period: 1000);
}
```

Both the highlighter task and the key listener access the indices of the currently highlighted button. Synchronization is achieved by the use of synchronized block so that only one of the two tasks can access it at one time.



```java
// Function to move the highlighter on functions
public static void changeFocusFunc() {
    int nextFocus;
    // synchronizing currentFocusIndexFunc as the key listener might also try to access this
    synchronized ((Integer) currentFocusIndexFunc) {
        // get the index of button that should be highlighted next
        nextFocus = (currentFocusIndexFunc + 1) % NO_OF_BUTTONS_FUNC;
        //highlight the next button
        funcButtonArrayList.get(nextFocus).setBackground(Color.MAGENTA);
        // unhighlight the current highlighted button
        funcButtonArrayList.get(currentFocusIndexFunc).setBackground(null);
        // update the index of highlighted button
        currentFocusIndexFunc = nextFocus;
    }
}
```