Concurrent Programming

Exercise Booklet 6: Monitors

Solutions to selected exercises (♢) are provided at the end of this document. Important: You should first try solving them before looking at the solutions. You will otherwise learn **nothing**.

Strategy: Signal and continue ($E = W < S$)

Exercise 1. Implement the fan bar exercise from Exercise Booklet 5, using monitors. Here is the statement. On Fridays the bar is usually full of Jets fans. Since the owners are Patriots fans they would like to implement an access control mechanism in which one Jets fan can enter for every two Patriots fans. Use the following stub as guideline:

```
class Bar {
    // your code here



}

Bar b = new Bar();
100.times {
    Thread.start {// jets
       b.jets();
    }
}

100.times {
    Thread.start {// patriots
       b.patriots();
    }
}
```

Exercise 2. (♢) We wish to implement a `three-way sequencer` using monitors in order to coordinate N threads. A three-way sequencer provides the following operations `first`, `second`, `third`. The idea is that each of the threads can invoke any of these operations. The sequencer will alternate cyclically the execution of `first`, then `second`, and finally `third`.

Exercise 3. We wish to implement a barrier using monitors in order to coordinate N threads.

A barrier provides a unique operation called `waitAtBarrier`. The idea is that each of the N threads invokes the operation `waitAtBarrier` and its effect is that the thread will block, and cannot continue, until all remaining threads invoke the `waitAtBarrier` operation. For example, if `b` is a barrier for coordinating 3 threads, the following use of `b` in each thread

```
class Barrier {
   // complete

}
```

```
Barrier b = new Barrier();

Thread.start { //T1
    print('a');
    b.waitAtBarrier();
    print(1);
}

Thread.start { //T2
    print('b');
    b.waitAtBarrier();
    print(2);
}

Thread.start {// T3
    print('c');
    b.waitAtBarrier();
    print(3);
}
```

guarantees that the letters will be displayed before the numbers. Supply an implementation for the barrier monitor. You may assume that once all N processes reach the barrier, they will be allowed to continue and so will all subsequent threads that call `waitAtBarrier` (non-cyclic barrier or count down latch).

Exercise 4. This exercise is a based on the Train exercise from Exercise Booklet 5. Trains run in both North-South and South-North direction, each on its own track. Two kinds of trains ride these tracks: passenger trains and freight trains.

- Passenger trains: A passenger train can only stop at the station if there are no other trains on the same track. It does not matter whether there is a train at the station on the track corresponding to trains travelling in the opposite direction.
- Freight trains: The station has the ability to load freight trains via a loading machine (which shall not be modeled). In order for a freight train to stop at the station, no other trains can be at the station in any of the two tracks.

Complete the following monitor operations:

```
import java.util.concurrent.locks.*;

class TrainStation {

    void acquireNorthTrackP() {

    }

    void releaseNorthTrackP() {

    }

    void acquireSouthTrackP() {

    }
```

```
16
        void releaseSouthTrackP() {
18
        }
20
        void acquireTracksF() {
22
        }
24
        void releaseTracksF() {
26
        }
28  }
30  TrainStation s = new TrainStation();

32  10.times {
      int id = it;
34    Thread.start { // Freight Train going in any direction
        s.acquireTracksF();
36      print "FT ${id} ";
        s.releaseTracksF()
38    }
    }
40
    100.times{
42    int id = it;
      Thread.start { // Passenger Train going North
44      s.acquireNorthTrackP();
        print "NPT ${id} ";
46      s.releaseNorthTrackP()
      }
48  }

50  100.times{
        int id = it;
52      Thread.start { // Passenger Train going South
          s.acquireSouthTrackP();
54        print "SPT ${id} ";
          s.releaseSouthTrackP()
56      }
    }
```

Exercise 5. Show that the following variation of readers/writers seen in class may deadlock. Recall that the solution in class gives priority to the writers.

```
1   monitor RW {
      int readers = 0;
3     int writers = 0;
      condition OKtoRead, OKtoWrite;
5
      public void StartRead() {
7       while (writers != 0 or not OKtoWrite.empty()) {
          OKtoRead.wait();
9       }
        readers = readers + 1;
```

```
    OKtoRead.signal();
  }

  public void EndRead {
    readers = readers - 1;
    if (readers==0) {
      OKtoWrite.signal();
    }
  }

  public void StartWrite() {
    while (writers != 0 or readers != 0) {
      OKtoWrite.wait();
    }
    writers = writers + 1;
  }

  public void EndWrite() {
    writers = writers - 1;
    if (OKtoRead.empty()) {
      OKtoWrite.signal();
    } else {
      OKtoRead.signal();
    };
  }
}
```

The only difference is the `EndWrite` operation which in the slides reads as follows:

```
public void EndWrite() {
    writers = writers - 1;
    OKtoWrite.signal();
    OKtoRead.signal();
}
```

Exercise 6. In a smart energy grid connected users can either behave as consumers or producers of energy (though not both at the same time). For example, the following code fragment shows a user that behaves as a consumer for an hour and then as a producer for two hours:

```
   grid.startConsuming();
   sleep(1h);
   grid.stopConsuming();
   grid.startProducing();
   sleep(2h);
   grid.stopProducing();
```

Address the following scenarios using monitors:

1. Users can always behave as producers, but can only behave as consumers if there is an equal or greater number of producers. Moreover, you must guarantee that a producer cannot stop producing if, in doing so, it leaves some consumer without a supply.

2. Modify your solution so that users cannot behave as producers (i.e. they block) if the grid has more than N producers.

3. Extend the previous solution so that the exit of producers is given priority over the entry of new consumers.

Exercise 7. In a local pizza shop two types of pizzas are produced: small and large. The cook places the pizzas on a counter so that the clients can help themselves and then pay at the register. Clients may either buy a small pizza or a large pizza. In the case of large pizzas, if there are no large pizzas, they reluctantly accept two small pizzas. Clients are not polite and hence do not wait in a queue (they all compete for the pizzas).

1. Provide a solution in which only the pizza shop is modeled.
2. Modify your solution assuming that the counter holds at most N pizzas at any given time.

Exercise 8. The organizing committee of a conference has a conference room for talks. People interested in attending a talk enter the room and wait until the talk starts. Talks start when the speaker has arrived in the room. Participants are respectful and do not leave the room until the talk has finished. Neither do they enter when the speaker has already begun. The room has capacity for 50 participants, excluding the speaker; when the room is full, new assistants must wait until the next talk starts, after the speakers rest.

Provide a solution using monitors that takes the following scenarios into account:

1. There is only one room and the same talk is given over and over again. The speaker rests 5 minutes between talks. If, at the moment in which the talk starts, the room is empty (disregarding for the speaker), the speaker rests five minutes waiting for the assistants to arrive.
2. The same as above except that three talks take place in the room, one after the other. The speakers must wait until there are at least 40 people in the room.

1 Solutions to Selected Exercises

Answer to exercise 2

```
monitor TWS{
  int state = 1;
  condition first, second, third;

  void first() {
      while (state != 1)
          first.wait();
      state = 2;
      second.signal();
  }

  void second() {
      while (state != 2)
          second.wait();
      state = 3;
      third.signal();
  }

  void third() {
      while (state != 3)
          third.wait();
      state = 1;
      first.signal();
    }
```