CS3211 Parallel and Concurrent Programming – Tutorial Week 7

Sample Exercises:
[Please conduct these as an interactive discussion, rather than an evaluation. Please also make it clear to the students that they are not being evaluated for their performance in these exercises, so that they are not afraid to make mistakes while answering.]

1. A single-slot buffer may be modeled by
   ONEBUF = (put -> get -> ONEBUF)

   Program a Java class, OneBuf, that implements this one-slot buffer as a monitor.

```
public class OneBuf {
      private int buf;
      private boolean empty = true;

      public synchronized void put(int x) throws InterruptedException{
            while(!empty) wait();
            buf = x;
            empty = false;
            notifyAll();
      }

      public synchronized int get() throws InterruptedException{
            while(empty) wait();
            empty = true;
            notifyAll();
            return buf;
      }
}
```

2. A museum allows visitors to enter through the east entrance and leave through its west exit. Arrivals and departures are signalled to the museum controller by the turnstiles at the entrance and exit. At opening time, the museum director signals the controller that the museum is open and then the controller permits both arrivals and departures.. At closing time, the director signals that the museum is closed, at which point only departures are permitted by the controller. Given that it consists of the four processes EAST, WEST, CONTROL and DIRECTOR, identify which process should be monitors. Provide a Java implementation of the monitors.

// This is a simplified implementation without the visitor count being maintained.

```
public class Control {
	private boolean allowEnter;
	private boolean allowExit;

	public synchronized  void arrive () throws InterruptedException{
		while(!allowEnter) wait();
	}

	public synchronized void depart () throws InterruptedException {
		while(!allowExit) wait();
	}

	public synchronized void open(){
		allowEnter = true;
		allowExit = true;
		notifyAll();
	}

	public synchronized void close(){
		allowEnter = false;
		notifyAll();
	}
}
```

3. The Dining Savages: A tribe of savages eats communal dinners from a large pot that can hold M servings. When a savage wants to eat, he helps himself from the pot unless it is empty in which case he waits for the pot to be filled. If the pot is empty, the cook refills the pot with M servings. The behavior of the savages and the cook is described by:

SAVAGE = (getserving -> SAVAGE)
COOK = (fillpot -> COOK)

Provide a Java implementation of this problem involving monitors.

// You can think of it as a variation of the producer-consumer problem
// the savage process is the consumer
// the cook is the producer – except that instead of producing 1 item, it produces M items
// in one go.

```
import java.util.Queue;

	/*
	 * range T = 0..M
	 * POT = MYPOT[0]
	 * MYPOT[i:T] = when i> 0 getserving -> MYPOT[i-1]
	 * MYPOT[i:T]= when i == 0 fillpot -> MYPOT[M]
	 */

	public class Pot {
		private Queue<Integer> buf;
		private boolean empty;
		private int M;

		public synchronized Integer getserving() throws InterruptedException{
			while(empty) wait();
			Integer serving = buf.remove();
			if(buf.isEmpty())
			{
				empty = true;
				notifyAll();
			}
			return serving;
		}

		public synchronized void fillpot() throws InterruptedException{
			while(!empty) wait();
			for(int i=0;i<M;i++){
				buf.add(i);
			}
			empty = false;
			notifyAll();
		}

	}
```