A Question 1 Code

A.1 Filter algorithm

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

class Filter implements Lock{
    private int level[];
    private int victim[];

    /**
     * Constructor for the filter class
     * @param n the amount of threads
     */
    public Filter (int n){
        level = new int[n];
        victim = new int[n];

        // Initialize the levels
        for (int i = 0; i < n; i++){
            level[i] = 0;
        }
    }

    /**
     * Method to lock the lock
     */
    public void lock(){
        int threadId = (int) Thread.currentThread().getId() % level.length;
        for (int L = 1; L < level.length; L++){
            level[threadId] = L;
            victim[L] = threadId;

            boolean spinwait = true;
            while (spinwait) {
                spinwait = false;

                for (int k = 1; k < level.length; k++) {
                    if (k != threadId && level[k] >= L && victim[L] == threadId) {
                        spinwait = true;
                        break;
                    }
                }
            }
        }
    }

    /**
     * Method to release the lock
     */
    public void unlock(){
        int threadId = (int) Thread.currentThread().getId() % level.length;
        level[threadId] = 0;
    }

    /*
    * Abstract methods that need to be defined. They do nothing basically since they
    * is no the focus of the filter lock
    */
    public java.util.concurrent.locks.Condition newCondition() {
        return null;
    }

    public boolean tryLock(long timeout, java.util.concurrent.TimeUnit unit) throws
    java.lang.InterruptedException {
```

```
62          return false;
63      }
64
65      public boolean tryLock() {
66          return false;
67      }
68
69      public void lockInterruptibly() throws java.lang.InterruptedException {
70      }
71  }
```

A.2 Bakery Algorithm

```
1   import java.util.concurrent.locks.Lock;
2
3   class Bakery implements Lock{
4       boolean[] flag;
5       int[] label;
6
7       /**
8        * Number of threads. Used for interation in the lock
9        * method.
10       */
11      private int n;
12
13      public Bakery(int n){
14          flag = new boolean[n];
15          label = new int[n];
16          this.n = n;
17          for(int i = 0; i < n; i++){
18              flag[i] = false;
19              label[i] = 0;
20          }
21      }
22
23      public void lock(){
24          int me = (int) Thread.currentThread().getId() % n;
25          flag[me] = true;
26          label[me] = findMax(label) + 1;
27
28          boolean spinwait = true;
29          while (spinwait) {
30              spinwait = false;
31
32              for (int k = 0; k < n; k++) {
33                  if ((k != me) && flag[k]
34                  && ((label[k] < label[me]) || ((label[k] == label[me]) && k < me))){
35                      spinwait = true;
36                      break;
37                  }
38              }
39          }
40      }
41
42      public void unlock(){
43          int me = (int) Thread.currentThread().getId() % n;
44          flag[me] = false;
45      }
46
47      private int findMax(int[] array) {
48          int max = Integer.MIN_VALUE;
49          for (int i = 0; i < array.length; i++)
50          {
51              if(array[i] > max){
52                  max = array[i];
53              }
```

```
        }
        return max;
    }

    /*
    * Abstract methods that need to be defined. They do nothing basically since they
    * is no the focus of the filter lock
    */
    public java.util.concurrent.locks.Condition newCondition() {
        return null;
    }

    public boolean tryLock(long timeout, java.util.concurrent.TimeUnit unit) throws
    java.lang.InterruptedException {
        return false;
    }

    public boolean tryLock() {
        return false;
    }

    public void lockInterruptibly() throws java.lang.InterruptedException {
    }
}
```

A.3 Test Code for Bakery Algorithm

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class BakeryTest {

    public static Bakery lock1 = new Bakery(5);

    public static void main(String arg[]) {
        BakeryClient[] clients = new BakeryClient[5];

        for(int i = 0; i < clients.length; i++)
        {
            clients[i] = new BakeryClient();
            clients[i].start();
        }
    }

    private static class BakeryClient extends Thread {
        public void run() {
            System.out.println("Thread with id = " + this.getId() + " acquiring Lock 1"
    );
            lock1.lock();
            System.out.println("Thread with id = " + this.getId() + " Holding Lock 1");

            try
            {
                sleep(5000);
            }
            catch(Exception e)
            {
                System.out.println("Error: " + e.getMessage());
                lock1.unlock();
                return;
            }

            System.out.println("Thread with id = " + this.getId() + " Releasing lock");
            lock1.unlock();
        }
```

```
39          }
40  }
```