CONCURRENT PROGRAMMING - EXAM

First name: _____

Last name: _____

Matrikel: _____

Date: Wednesday, 19.12.2012
Allowed material: this paper and a pen
Number of exercises: 6
Total points: 29

Important:
You have 60 minutes to solve the exam.
Some exercises are split over more than one page, so carefully read all exercises before proceeding. More or less you have 2 minutes per point. If you don't know something don't waste time, go ahead and if you have time go back and try to solve unanswered questions afterwards. You must answer each question briefly and to the point, any answer that goes above 3 lines will not be corrected.

Exercise 1 (6 Points)

Answer the following questions (do not write more than 3 sentences):

a) What are safety and liveness? **(1 Point)**

b) Are counting semaphores more powerful than binary semaphores? Justify **(1 Point)**

c) What could you add to Petri nets to make them Turing-complete? **(1 Point)**

d) What is the difference between Flow Architecture and BlackBoard architecture? **(1 Point)**

e) Why is parallel programming always better than concurrency? justify **(1 Point)**

f) You implemented the mechanism to handle the requests to a web shopping cart service using a cached thread pool. Can you replace cached tread pool with better implementation? Justify. **(1 Point)**

Exercise 2 (4 Points)

Consider the follow FSP specification:

```
COUNT (N=3)    = COUNT[0],
COUNT[i:0..N] = (when(i<N) inc->COUNT[i+1]
                |when(i>0) dec->COUNT[i-1] ).
```

a) Draw the corresponding LTS diagram **(2 Points)**

b) Write an FSP specification that can model the life cycle of a Java Thread. **(2 Points)**

Exercise 3 (6 Points)

Consider the following FSP model. The corresponding LTS diagrams are shown in Figure 1. The '/' operator is relabelling: $/\{new_1/old_1, ...\}$.

```
P = (a.q -> b.p -> P).
Q = (a.p -> b.q -> Q).
||S = (P || Q) / {a/b}.
```

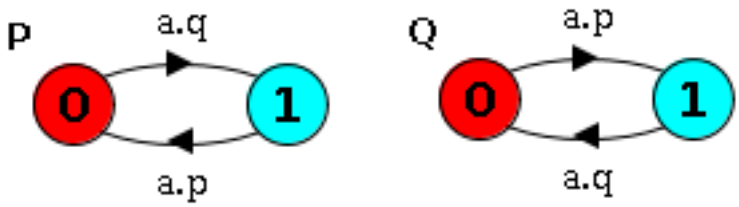

Figure 1: LTS graphs for P and Q.

a) The above system is deadlocked. Explain how the deadlock occurs and relate it to one of the four necessary and sufficient conditions for deadlock occurrence. **(2 Points)**

b) The next model attempts to fix the problem by allowing P and Q to timeout first operation.

```
P = (a.q -> b.p -> P
| timeout.p -> b.p -> P).
Q = (a.p -> b.q -> Q
| timeout.q -> b.q -> Q).
||S = (P || Q) / {a/b}.
```

Is deadlock still possible? If so, provide an execution trace leading to the deadlock.**(2 Points)**

c) Image the P and Q model described with a Petri Net. How does the deadlock manifest there? **(2 Points)**

Exercise 4 (4 Points)

Consider the following BoundedHoneyPot class (the Semaphore class simply implements a usual counting semaphore). A bounded honey pot has a limited number of slots for allocating honey:

```
public class BoundedHoneyPot {
  private Object[] honeyPot;
  private int in = 0;
  private int out = 0;
  private int size;
  private Semaphore full;  // counts number of items
  private Semaphore empty; // counts number of spaces
  private Semaphore mutex;

  public BoundedHoneyPot(int size) {
    this.size = size;
    honeyPot = new Object[size];
    full = new Semaphore(0);
    empty = new Semaphore(size);
    mutex = new Semaphore(1);
  }
  public synchronized void put(Object obj) {
    empty.down();
    honeyPot[in] = obj;
    in = (in + 1);
    full.up();
  }
  public synchronized Object get() {
    full.down();
    Object obj = honeyPot[out];
    honeyPot[out] = null;
    out = (out + 1);
    empty.up();
    return (obj);
    }
}
```

a) This code is not correctly synchronized. Explain what kind of error may occur if this class is used as is. **(2 Points)**

b) Fix the problem of the above code by adding some notes on the methods *put()* and *get()*. **(2 Points)**

Exercise 5 (4 Points)

Consider the petri nets shown in Figures 2 and 3.

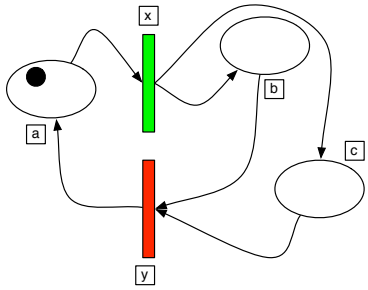

Figure 2: Net 1

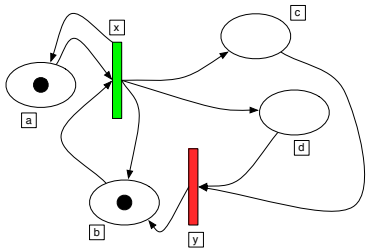

Figure 3: Net 2.

a) Provide the formal definition of both petri nets. **(2 Points)**

b) Are the Petri nets bounded? Safe? Conservative? Are all transitions live? **(2 Points)**

Exercise 6 (5 Points)

The following code shows the implementation of a read/write lock class in Java. The `lockRead` (`lockWrite` respectively) is called before reader (writer respectively) enters critical section. The `unlockRead` (`unlockWrite` respectively) is called after reader (writer respectively) leaves critical section.

```
public class ReadWriteLock{

  private int readers = 0;
  private int writers = 0;

  public synchronized void lockRead() throws InterruptedException{
    while(writers > 0){
      wait();
    }
    readers++;
  }

  public synchronized void unlockRead(){
    readers--;
  }

  public synchronized void lockWrite() throws InterruptedException{
    while(readers > 0 || writers > 0){
      wait();
    }
    writers++;
  }

  public synchronized void unlockWrite() throws InterruptedException{
    writers--;
  }
}
```

Answer the following questions:

a) There is a synchronization problem in the implementation of the ReadWriteLock. Describe how would you solve it? **(2 Points)**

b) Is your solution fair? if not describe how to make the implementation fair? **(2 Points)**

c) Is your read/write lock class reentrant? Justify your answer. **(1 Point)**

Points

Exercise 1

Task	Points	Score
1	1	
2	1	
3	1	
4	1	
5	1	
6	1	
Total	**6**	

Exercise 2

Task	Points	Score
1	2	
2	2	
Total	**4**	

Exercise 3

Task	Points	Score
1	2	
2	2	
3	2	
Total	**6**	

Exercise 4

Task	Points	Score
1	2	
2	2	
Total	**4**	

Exercise 5

Task	Points	Score
1	2	
2	2	
Total	**4**	

Exercise 6

Task	Points	Score
1	2	
2	2	
3	1	
Total	**5**	