ALP IV: Übung 5

Tutor: Christoph Beuck

Lisa Dohrmann (4130066), Adrian Neumann (4140810), Naja v. Schumde (4127652)

4. Juni 2008

Aufgabe 1

1. mit Warteanweisungen In der Vorlesung werden die bevorzugt, die auf den Specialdrucker warten, wenn der Specialdrucker frei wird. Das können wir mit AWAIT nicht ausdrücken.

```
1  MONITOR TwoPrinters {
2     private boolean normalbusy, specialbusy;
3     public boolean request(boolean special) {
4        if(special) {
5           //Naja: Das kann man doch gleich zu AWAIT !specialbusy machen
                ?
6           if(specialbusy) {
7              AWAIT specialbusy == false;
8           }
9           specialbusy = true;
10       }
11       else {
12          // Ebenso: AWAIT (!specialbusy || !normalbusy)
13          if(specialbusy && normalbusy) {
14             AWAIT normalbusy == false
15                || specialbusy == false;
16          }
17          if(normalbusy) specialbusy = true;
18          else {
19             normalbusy = true;
20             return false; //nur normal
21          }
22       }
23       return true; //special
24    }
25
26    public void release(boolean special) {
27       if(special) {
28          specialbusy = false;
29       }
30       else normalbusy = false;
31    }
32 }
```

   In Bezug auf die Nebenläufigkeit unterscheidet sich die Performance dieses Programms nicht von der Version aus der Vorlesung. Lediglich die Bevorzugung der Anwärter auf den Specialdrucker fehlt.

2. mit Wachen

```
1  MONITOR TwoPrinters {
2    private boolean normalbusy, specialbusy;
3    public boolean request(boolean special) WHEN !(normalbusy &&
        specialbusy) {
4      if (special) {
5        if (specialbusy) {
6          //wir wollen special, kriegen's aber
```

```
7           //nicht. Wir verlassen den Monitor und
8           //geben einem anderen Thread die Chance
9           //ranzukommen.
10
11          return request(special);
12
13        }
14        else { //special belegen
15          specialbusy=true;
16          return true;
17        }
18      }
19      else {
20        if (!normalbusy) { //normal ist frei
21          normalbusy = true;
22          return false;
23        } else {    //special ist frei
24          specialbusy =true;
25          return true;
26        }
27      }
28    }
29      public void release(boolean special) {
30      if(special) {
31        specialbusy = false;
32      }
33      else normalbusy = false;
34    }
35  }
```

Auch in dieser Version können wir die Bevorzugung der Anwärter auf den Specialdrucker nicht ausdrücken. In Bezug auf die Nebenläufigkeit unterscheidet sich auch dieses Programm nicht von der Version aus der Vorlesung, sofern man davon ausgeht, dass mit dem `return` tatsächlich der Monitor verlassen wird und eine neue Auslosung stattfindet. Ist dies nicht der Fall, müssen alle Threads warten, bis der Specialdrucker frei wird.

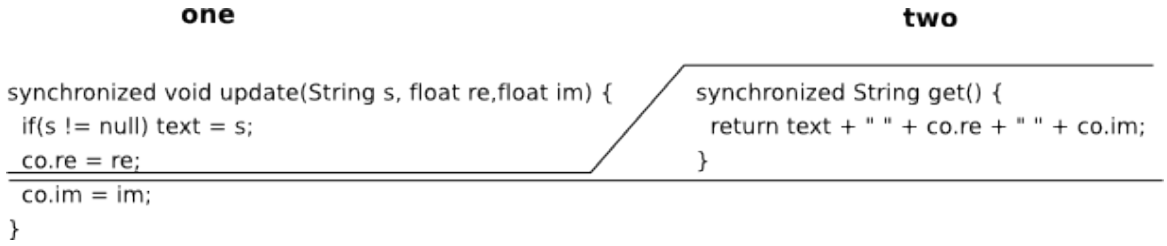
Auch diese Version sollte bezogen auf Nebenläufigkeit ebenso effizient sein wie die aus der Vorlesung. Ebenso werden hier blockierte Prozesse bevorzugt, da der Monitor blockiert wird, bevor die Bedingung getestet wird.

Aufgabe 2

```
1  class ReentrantLock {
2      private volatile Thread lock;
3      private int count=0;
4      synchronized void lock() {
5          if (lock.equals(Thread.currentThread())) {
6              ++count;
7              return;
8          }
9          AWAIT lock==null;
10         lock=Thread.currentThread();
11         count = 1;
12      }
13     synchronized void unlock() throws IllegalMonitorStateException {
14       if(lock.equals(Thread.currentThread())) {
15         count--;
16         if (count == 0) {
17           lock = null;
18           return;
19         }
20       }
21       else throw new IllegalMonitorStateException();
22     }
23 }
```

Aufgabe 3

Es ist in der Tat so, dass die X-Klasse die komplexen Zahlen kapselt. Da aber in one und two eine Referenz auf genau das selbe complex-Objekt steht, kann es dennoch zu Problemem kommen, weil ja so zwei Threads „gleichzeitig“ auf dem selben Objekt arbeiten können und so all die üblichen Schwierigkeiten entstehen. Es könnte so z.B. zu folgendem Szenario kommen:



Es ist also sinnvoll, wenn man dem Monitor nicht ein fertig erstellten Objekt übergibt, sondern der sich selbst in seinem Konstruktor darum kümmert sich eines zu erstellen. So können nicht mehrere Monitore das selbe Objekt manipulieren.
Wenn allerdings dies unumgänglich ist, so sollte darauf geachtet werden, dass nicht über `this` synchronisiert wird, sondern über das Objekt, welches von mehreren Monitoren benutzt wird, in diesem Fall also über `co`.