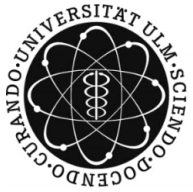

ulm university universität ulm

Programmierung von Systemen
Blatt 12

Marco Deuscher
marco.deuscher@uni-ulm.de
Benedikt Jutz
benedikt.jutz@uni-ulm.de

Juli 2018

1 Deadlocks

```
class DeadLock extends Thread {
    public static void main ( String[]args) throws
       Exception {
            Object l1 = new Object ( ) ;
            Object l2 = new Object ( ) ;
            Thread t1 = new DeadLock ( l1 , l2 ) ;
            Thread t2 = new DeadLock ( l1 , l2 ) ;
            t1.start();
            t2.start();
            t1.join();
            t2.join();
            }
    private Object lock1 ;
    private Object lock2 ;

    DeadLock (Object lock1 , Object lock2 ) {
            this . lock1 = lock1 ;
            this . lock2 = lock2 ;
            }
    public void run (){
        synchronized (lock1){
            System . out . println( "First lock
               accquired!" );
        synchronized (lock2){
            System.out.println("Second lock accquired");
            }
        }
            System.out.println("All locks released");
    }
}
\label{listing}
```

a)

Beim synchronisieren einer Methode, wird eine Sperre auf das Objekt gelegt, welchem die Methode gehört. Dabei hat jede Instanz ihre eigene synchronisierte Methode, so dass jeweils nur ein Thread auf der synchronisierten Methode einer Instanz arbeiten kann.
Bei der Verwendung der synchronized Blöcke, wird eine Sperre auf das, in den Klammern spezifizierte, Objekte gelegt. Somit kann innerhalb des synchronisiert Blocks immer nur ein Thread auf dem gleichen Monitor Objekt arbeiten.

b)

Die Threads bleiben in der run-Methode stecken, da die Objekte mit überkreuzten Argumenten im Konstruktor erstellt werden. Es kann nun dazu kommen, dass t1 eine Sperre auf das Objekt l1 hat und t2 eine Sperre auf das Objekt l2. Die beiden Threads warten dann darauf, dass der jeweils andere das Objekt freigibt um fortzufahren. Da dies nicht geschieht, sind die Threads in einem Deadlock gefangen.

c)

Die join-Methode erlaubt es, einen Thread warten zu lassen, bis dieser zu Ende kommt.
In diesem Fall wird gewartet bis t1 und danach t2 zu Ende kommt und kehrt danach jeweils in den Main-Thread zurück.
Werden die join-Aufrufe entfernt, wird nicht gewartet bis t1/t2 zum Ende kommt und der main-Thread wird vorzeitig beendet.

d)

Es kommt zum Deadlock, da t1 eine Sperre auf Objekt l1 hält und der Thread t2 eine Sperre auf das Objekt l2 hält. Da nun keine der beiden run-Methoden weitergeführt werden kann, befindet man sich in einem Deadlock.

e)

Der Deadlock kann verhindert werden indem beim erstellen der Threads diese nicht mit überkreuzten Argumenten initialisiert werden.
Das Listing in **??** zeigt die Verbesserung.