# Seminar 3 – Deadlock analysis

## Semaphores

Consider the following two thread classes S and T referring to the shared resources A, B, C, D, and E which are allocated using semaphores.

```
class S extends Thread {
    public void run() {
        // ....
        A.acquire();
        B.acquire();
        useAB();
        A.release();
        C.acquiree();
        useBC();
        B.release();
        C.release();
        // ....
    }
}
```

```
class T extends Thread {
    public void run() {
        // ....
        C.acquire();
        D.acquire();
        useCD();
        A.acquire();
        useACD();
        A.release();
        D.release();
        E.acquire();
        useCE();
        C.release();
        E.release();
        // ....
    }
}
```

1. Draw a resource allocation graph. Is the system safe concerning deadlock, that is, can we conclude that there is no risk for a deadlock?

2. Review the program and the graph from 1 again. Assume there is exactly one thread of type S and one thread of type T. Explain why the system cannot deadlock!

3. Assume that there are two threads of type S, and still one T thread. Can deadlock occur in this case?

4. Change one of the threads in such a way that there will be no risk for deadlock no matter how many threads there are of each type. Note that the same computations should be carried out and still with full synchronization. In case of several solutions, can any solution be considered to be more efficient than others? In what sense?

## Monitors

A Java program consists of three concurrently executing threads T1, T2, and T3. They communicate with each other using four shared objects A, B, C, and D. The shared objects contain methods declared *synchronized* as well as other, non-synchronized, methods (such as A.s()). Relevant parts of the program and the design of the monitors are shown below (subset of the total program only). Each thread and shared object is assumed to know references to the (other) shared objects A, B, C, and D named a, b, c, and d, respectively.

```
class T1
    extends Thread {
    void run() {
        b.z();
        a.x();
    }
}
```

```
class T2
    extends Thread {
    void run() {
        b.r();
        c.y();
        b.w();
    }
}
```

```
class T3
    extends Thread {
    void run() {
        d.u();
    }
}
```

```
class A {
    synchronized void x() {
        c.y();
    }

    void s() { // NOT SYNCHRONIZED
        // ...
    }

    synchronized void t() {
        // ...
    }
}
```

```
class B {
    synchronized void z() {
        // ...
    }

    synchronized void r() {
        a.s();
    }

    synchronized void w() {
        d.v();
    }
}
```

```
class C {
    synchronized void y() {
        b.z();
    }
}
```

```
class D {
    synchronized void u() {
        a.t();
    }

    synchronized void v() {
        // ...
    }
}
```

Hint: Entering a *synchronized* method can be interpreted as ***acquire***, leaving as ***release***.

1. Draw a resource allocation graph for the system above.
2. Can the system experience deadlock?