Solving the critical section problem using test-and-set function

integer common ← 0

p	q
integer local1 loop forever p1: non-critical section repeat p2: test-and-set(common, local1) p3: until local1 = 0 p4: critical section p5: common ← 0	integer local2 loop forever q1: non-critical section repeat q2: test-and-set(common, local2) q3: until local2 = 0 q4: critical section q5: common ← 0

```
/* Copyright (C) 2006 M. Ben-Ari. See copyright.txt */

/* Programmed by Panu Pitkämäki */


/* Critical section problem with test-and-set  */

class TestSet extends Thread {

   /* Number of processes currently in critical section */

   static volatile int inCS = 0;

   /* Common value */

   static volatile int common = 0;

   /* Local value */

   int local;

  // char pro;
```

```
synchronized void testAndSet() {
   local = common;
   common = 1;
}

public void run() {
            int x=0;
   while (x<4) {
               x++;
      /* Non-critical section */
      do
         testAndSet();
      while (local == 1);
      inCS++;
      Thread.yield();
      /* Critical section */
      System.out.println("Number of processes in critical section: "
            + inCS);
      inCS--;
      common = 0;
   }
}

public static void main(String[] args) {
```

```
        TestSet p = new TestSet();
        TestSet q = new TestSet();
        p.start();
        q.start();
    }
}
```

2- solving the critical section problem using the Exchange function

integer common ← 1	
p	q
integer local1 ← 0 loop forever p1: non-critical section repeat p2: exchange(common, local1) p3: until local1 = 1 p4: critical section p5: exchange(common, local1)	integer local2 ← 0 loop forever q1: non-critical section repeat q2: exchange(common, local2) q3: until local2 = 1 q4: critical section q5: exchange(common, local2)

```
/* Copyright (C) 2006 M. Ben-Ari. See copyright.txt */
/* Programmed by Panu Pitkämäki */

/* Critical section problem with exchange  */
class Exchange extends Thread {
```

```
/* Number of processes currently in critical section */
static volatile int inCS = 0;
/* Common value */
static volatile int common = 1;
/* Local value */
int local = 0;

synchronized void exchange() {
   int temp;
   temp = common;
   common = local;
   local = temp;
}

public void run() {
   while (true) {
      /* Non-critical section  */
      do
         exchange();
      while (local == 0);
      inCS++;
      Thread.yield();
      /* Critical section */
      System.out.println("Number of processes in critical section: "
            + inCS);
```

```
            inCS--;
            exchange();
        }
    }

    public static void main(String[] args) {
        Exchange p = new Exchange();
        Exchange q = new Exchange();
        p.start();
        q.start();
    }
}
```