

# Solution proposal to selected exercises from Lecture 4

Jens Chr. Godskesen  
IT University of Copenhagen

## Exercise 4, (A bank account)

```
agent P(i,j) = 'i.'j.P<i,j>
agent Person(i,i25,i50,i100,w,w25,w50,w100) =
    P<i25,i> + P<i50,i> + P<i100,i> + P<w25,w> + P<w50,w> + P<w100,w>

agent Account(i25,i50,i100,w25,w50,w100) =
    i25.('w25.0 | Account(i25,i50,i100,w25,w50,w100))
    + i50.('w50.0 | Account(i25,i50,i100,w25,w50,w100))
    + i100.('w100.0 | Account(i25,i50,i100,w25,w50,w100))

agent System(i1,i2,w1,w2) =
    ( (^ i25,i50,i100,w25,w50,w100)
      ( Account<i25,i50,i100,w25,w50,w100>
        | Person(i1,i25,i50,i100,w1,w25,w50,w100)
        | Person(i2,i25,i50,i100,w2,w25,w50,w100)))
```

## Exercise 6, (Programming Lab)

```
for (int l = 1; l <= 10; l++) { // Sorting is repeated 10 times

    for (int k = 1; k <= twon; k++) a[k] = input[k];

    Sort  sort1 = new Sort(1,n);
    Sort  sort2 = new Sort(n+1,twon);
    Merge merge = new Merge(1,n+1,twon);

    sort1.run();
    sort2.run();
    merge.run();

    System.out.println();
}
```

Make the sortings be done concurrently, followed by the merge.

## Exercise 6, (Programming Lab, sol. 1)

```
for (int l = 1; l <= 10; l++) { // Sorting is repeated 10 times

    for (int k = 1; k <= twon; k++) a[k] = input[k];

    Thread sort1 = new Thread(new Sort(1,n));
    Thread sort2 = new Thread(new Sort(n+1,twon));
    Thread merge = new Thread(new Merge(1,n+1,twon));

    sort1.start();
    sort2.start();

    sort1.join();
    sort2.join();

    merge.start();
    merge.join();

    System.out.println();
}
```

## Exercise 6, (Programming Lab, sol. 2)

Make sortings as well as the merge take place concurrently.

```
for (int l = 1; l <= 10; l++) { // Sorting is repeated 10 times

    for (int k = 1; k <= twon; k++) a[k] = input[k];

    Semaphore s1 = new Semaphore(0);
    Semaphore s2 = new Semaphore(0);
    Thread sort1 = new Thread(new Sort(1,n,s1));
    Thread sort2 = new Thread(new Sort(n+1,twon,s2));
    Thread merge = new Thread(new Merge(1,n+1,twon,s1,s2));

    sort1.start(); sort2.start(); merge.start();

    sort1.join(); sort2.join(); merge.join();

    System.out.println();
}
```

## Exercise 6, (Programming Lab, sol. 2, continued)

The strategi is to:

Use two semaphores, one for each sorting thread. Each time the sorting method finishes an iteration of its outer most loop a new integer is “sorted” and the semaphore is counted up.

Merge is waiting for to two integers to become sorted, ie. it waits counting down both the semaphores. It prints out the smallest of the sorted values and increases the unused semaphore. When one of the array parts is printed merge just finishes printing out the remaining part but still waiting for the integers to become sorted.

## Exercise 6, (Programming Lab, sol. 2, continued)

```
static class Sort implements Runnable {  
  
    ...  
  
    public void run() {  
        for (int i = low; i <= high-1; i++) {  
            for (int j = i+1; j <= high; j++) {  
                if (a[j] < a[i]) {  
                    int temp = a[j];  
                    a[j] = a[i];  
                    a[i] = temp;  
                }  
            }  
            semaphore.Up(); // semaphore signals one more integer sorted  
        }  
        semaphore.Up(); // ... because loop until high-1  
    }  
}
```

## Exercise 6, (Programming Lab, sol. 2, continued)

```
while (count1 < middle) { // loop in body of run method of class Merge
    s_left.Down(); s_right.Down(); // awaiting two new sorted numbers
    if (a[count1] < a[count2]) {
        System.out.print(" "+a[count1++]);
        s_right.Up();
        if (count1 >= middle) { // if left subarray sorted ...
            for (index2 = count2; index2 <= high; index2++) {
                s_right.Down();
                System.out.print(" "+a[index2]);
            }
        }
    }
    else {
        System.out.print(" "+a[count2++]);
        s_left.Up();
        if (count2 > high) { // if right subarray sorted ...
            for (index1 = count1; index1 <= middle-1; index1++) {
                s_left.Down();
                System.out.print(" "+a[index1]);
            }
            count1 = middle; // terminate
        }
    }
}
```