Exercise Sheet 10

1. **Dining Philosophers**
   N philosophers are sitting at a table. They are either thinking or eating. When they think, they do not eat. And when they eat, they are not thinking. The philosophers are sitting at a circular table with a large plate of sushi in the middle. A chopstick is placed between each philosopher. That is, every philosopher has a chopstick to their right and to their left. The philosopher requires two chopsticks to eat sushi. They do not speak to each other and they can only use the chopstick to their immediate right or left.

   Each chopstick is shared by two philosophers and is a *shared* resource. The philosophers act as concurrent processes trying to grab the chopsticks when they are hungry. If a philosopher reaches for a chopstick that is already in use, we may have a race condition. The philosopher must wait until the chopstick is free. What are the possible situations that can happen with what we have described so far? How can you solve them? Program a simulation of this into your computer and try to reproduce these situations and derive your solution.

2. **Example from the exam!**
   A company wants to install a shower for its employees. To cut costs both man and woman should use the same shower, but not at the same time. To determine the capacity and the efficiency of the showers, the CEO wants to simulate with a Java-program how the 50 male and the 50 female employees take a shower. Given the following program, perform the following tasks:
   a) Limit the Shower capacity to no more then 10 people.
   b) Ensure there there is only men or women in the shower, and not both
   c) Ensure that the shower is fair. Write the program such that eventually, every gender will have the possibility to shower.

```
class Shower {
  static int women=0, men=0;
  static void enter(Person p) {
    if (p instanceof Man) men = men+1;
    else women=women+1;
  }
  static void leave(Person p) {
    if (p instanceof Man) men=men-1;
    else women=women-1;
  }
  static void shower(Person p) { }
  public static void main(String[] argv) {
    for (int i=0; i<50; i++) {
      new Man();
      new Woman();
    }
```

```
    }
  }
  abstract class Person extends Thread {
    Person() { start (); }
    public void run() {
      Shower.enter(this);
      Shower.shower(this);
      Shower.leave(this);
    }
  }
  class Man extends Person {}
  class Woman extends Person {}
```

3. **Optional Homework** The common practice – in Switzerland -- of indicating friendship between friends is to give three kisses. When somebody walks into a room, they must go around and greet everybody who is a friend by giving them three kisses. The following action must be performed three times: the initiator kisses the cheek of the friend, the friend kisses the cheek of the initiator. The initiator must not move to another friend until they have finished the three kisses. Everybody in the room acts as a concurrent process. Program a simulation of this scenario and argue why it is free of deadlock, live-lock, and data races.