Exercise 3: Rendez-vous.

Hand-out: 3 May
Due: 3 May

The goal of the exercise is to get acquainted with the concepts of rendez-vous and active objects. Several ways of programming a rendez-vous will be discussed and illustrated with simple examples.

1. Rendez-vous in Java: the beast and the beauty
We illustrate the use of rendez-vous with three simple examples: client-server synchronisation (unconditional rendez-vous), bank account (conditional rendez-vous), and dining philosophers (conditional rendez-vous). We will compare a solution based on a language with no native support for rendez-vous (Java) and a solution written in sJava (or Synchronous Java) that has built-in mechanisms for the support of rendez-vous synchronisation. Discussed examples are available online (see course page).

Questions:
1.1. Which solution (Java or sJava) is more elegant?
1.2. Which solution of the dining philosopher problem is more elegant: monitor-based (as seen last week) or rendez-vous based? Which should be more efficient and why?
1.3. When would you use (active objects with) rendez-vous synchronisation?

2. Is rendez-vous the ultimate synchronisation mechanism?
In the last lecture, we saw that semaphores can be easily implemented with the rendez-vous mechanism. Also producer-consumer scenario was implemented in a straightforward way. Does it mean that rendez-vous is all that we need to implement all synchronisation scenarios?

Questions:
2.1. Can we simulate a barrier with rendez-vous? How efficient would such emulation be? How elegant would it be? Compare it with a semaphore-based barrier.
2.2. How can we implement the reader-writer scenario?
2.3. Back to Santa: how would you implement the priority scheduling among reindeer and elves with rendez-vous?