Prof. Oscar Nierstrasz
Haidar Osman, Leonel Merino, Nevena Milojković

SolutionSerie 2 - Concurrency and Java

Exercise 1 (2 Points)

Answer the following questions (0.5 point each):

a. What states can a Java thread be in? **Answer:**

   See lecture slides No 2, pp. slide 10. For more details, check the Java 7 API documentation: `https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html`

b. How can you turn a Java class into a monitor?

   By simply declaring all public methods as synchronized. More on that in the safety patterns lecture (forth lecture).

c. What is the Runnable interface good for? **Answer:**

   An object that implements Runnable can be made active, i.e. you can create a thread based on a runnable object. This is useful since Java does not allow multiple inheritance, so inheriting from the Thread class (to define an active object) makes it impossible to inherit from something else. Instead, you can implement Runnable.

d. Specify an FSP that repeatedly performs hello, but may stop at any time?

```
HELLO = (hello -> HELLO | hello -> STOP).
```

Exercise 2 (2 Points)

Consider the following Java implementation of a Singleton:

```
Public class Singleton {
   private static Singleton instance = null;
   private Singleton() {}
   public static Singleton getInstance() {
      if(instance == null) {
         instance = new Singleton();
      }
      return instance;
   }
}
```

This implementation assumes a single-threaded application.

a. What happens if the application is multithreaded? **Answer:**

   There is a possibility that many instances of the Singleton are created. Multiple threads can check (instance == null) and succeed then move to the instance creation.

b. How to implement a thread-safe singleton in Java? **Answer:**

   Make the getInstance() method synchronized.

c. Suppose there is 1000 requests/second from different threads to this Singleton. Does your implementation introduce a bottleneck? If yes, how can you improve it? Making the getInstance() method synchronized will lock the object every time a thread asks for the instance. This is not efficient. A better implementation is as follows:

```
        public static Singleton getInstance() {
           if(instance == null) {
              synchronized (Singleton.class) {
                 if(instance == null) {
                    instance = new Singleton();
                 }
              }
           }
           return instance;
        }
```

Exercise 3 (3.5 Points)

Download LTSA from http://www.doc.ic.ac.uk/~jnm/book/ltsa/download.html. For each of the following processes shown in Figure 1, give the Finite State Process (FSP) description of the corresponding Labeled Transition System (LTS) graph. You may verify the FSP descriptions by generating the corresponding state machines using the analysis tool. **Answer:**

```
APPOINTMENT = (hello -> converse -> goodbye -> STOP).

HOLIDAY = (arrive->relax->leave->HOLIDAY).

SPEED = (on->DRIVING),
DRIVING = (speed->DRIVING | off->SPEED).

LEFTONCE = (ahead-> (left->STOP |right->LEFTONCE)).

TREBLE = (in[i:1..3] -> out[i*3] -> TREBLE).

FIVETICK (N=5) = FIVETICK[1],
FIVETICK[i:1..N] = ( when(i<N) tick -> FIVETICK[i+1]
                   | when(i==N) tick -> STOP).

PERSON = (
  workday -> sleep -> work -> PERSON
  | holiday -> sleep -> (
                           play -> PERSON
                           | shop -> PERSON
                         )
).
```

Exercise 4 (2.5 Points)

Consider the full Race5K FSP from the lecture.

a. How many states and how many possible traces does it have if the number of steps is 5 (as in the lecture)? **Answer:**

   36 states (cartesian product of individual state spaces), and 252 traces (see next question)

b. What is the number of states and traces in the general case (i.e. for n steps)? **Answer:**

We assume 2 processes. Number of states: $(n+1)^2$ since n is the number of steps, so we have $(n+1)$ states (per process).
Number of traces: $\frac{(2 \cdot n)!}{n! \cdot n!}$. This is the number of possible paths in a $n \times n$ regular, directed grid graph from node $(0,0)$ to node (n,n). Another explanation is number of traces = $2n$ choose n
For k processes we have $\frac{(k \cdot n)!}{(n!)^k}$ traces.

c. Check your solution using the LTSA tool.

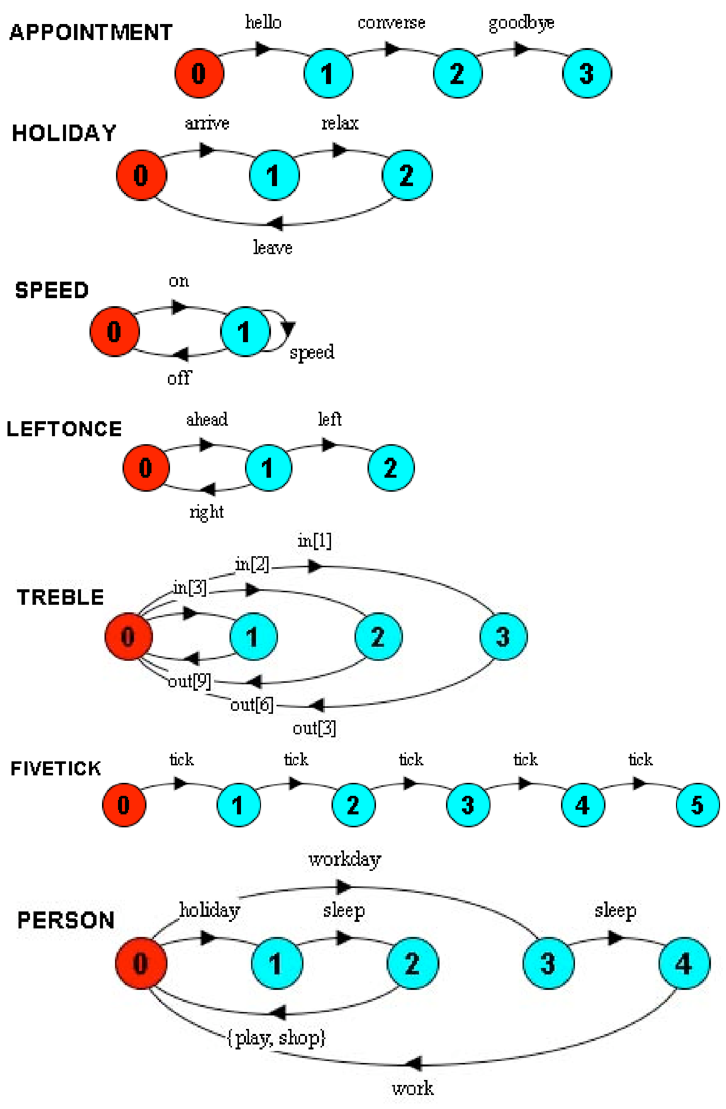

Figure 1: LTSA graphs.