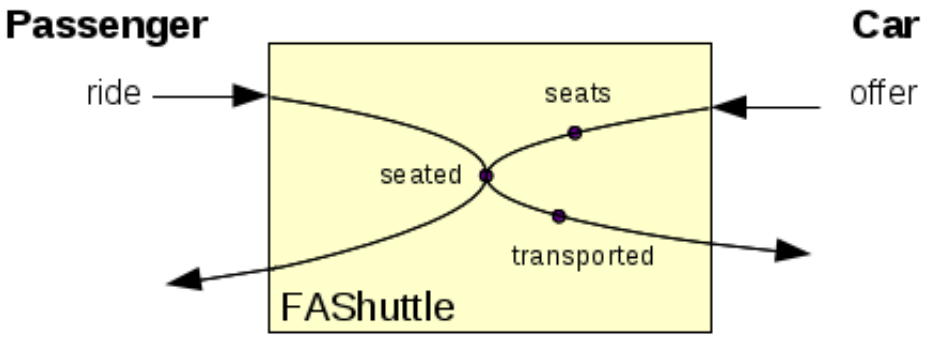
Parallel Programming WS 2014/2015 - Assignment 4

Kastens, Pfahler
Institut für Informatik, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn
Dec 08, 2014

Exercise 1 (Rendezvous: The Fürstenallee Shuttle)

A new driverless shuttle train connects the Fürstenallee building with the University campus. It consists of a single rail car with seats for 8 passengers. The shuttle only starts if the car is fully occupied.

We apply the design method for the rendezvous of processes to design a monitor for the FA shuttle simulation. The following illustration shows the entry procedures and the counter variables:



a) Determine the monitor invariant.
b) Determine the waiting conditions and the counter increments (Slide 36). Fill in the following table:

Entry Procedure	**wait while ...**	**Modify counters**
ride		
offer		

c) Substitute the increasing counters by limited counters.
d) LAB: Implement a Java monitor class that can be used with the FA shuttle simulation in directory `blatt4/FAShuttle`.

Exercise 2 (Barrier Synchronization: Rolling Dices)

We use N dices to generate endless sequences of random numbers:

```
public class DiceTest {
    final static int N = 4;
    public static void main(String[] args) {
        DiceBarrier x = new DiceBarrier(N);

        for (int i = 1; i <= N; i++) {
            new Dice(i, x).start();
        }
    }
}
```

The results of each round have to be added and output. Therefore the dices have to be synchronized using a barrier after each throw:

```
public class Dice extends Thread {
    private DiceBarrier x;
    private int number;
    private int val;

    public Dice(int number, DiceBarrier x) {
        this.number = number;
        this.x = x;
    }

    public void run() {
        while (true) {
            val = (int) (Math.random() * 6) + 1;
            x.barrier(number, val);
        }
    }
}
```

Use a simple shared counter barrier (Slide 44) to complete the implementation (`blatt4/dices`) of `class DiceBarrier`:

```
public class DiceBarrier {
   private final int N; // number of dices

   DiceBarrier(int n)  {
        N = n;
   }

   synchronized public void barrier(int dicenumber, int value) {
    // to be completed
    }
}
```

Hint: The barrier method is also responsible for computing the sum of the dice values. The `dicenumber` parameter can be used to generate log output like:

```
Dice 4 arrived
Dice 2 arrived
Dice 3 arrived
Dice 1 arrived
Sum = 15
Dice 1 arrived
Dice 4 arrived
Dice 2 arrived
Dice 3 arrived
Sum = 17
...
```