Dear All,

In the past Thursday (in the 23th of May), I have had a discussion with Robert Kovacs from 924, related to the specification of the observer `couldBeReserved(rt:RoomType, sd:Date, ed:Date)`. This observer is expected to return true if a room of the requested type could be reserved(booked) and false, contrary.

The question of Robert was if the situation in which there is at least a room which has no bookings must be included (treated) by the specification or not. In case in which this situation is covered by the second part of the specification where we test if there is at least a reserved room which does not contains days from the period `sd…ed`, (excepting of course `sd` and `ed`).

As my answer was not convincing for Robert, I decided to evaluate the specification in OCLE. This means to specify the model, the observer, to instantiate the model and to evaluate different situations, requiring more time than I supposed initially. I promised to send you the solution by email. I try to keep my promising. So, I attached few print screen comments.

At the seminars, I proposed a solution in which the set of rooms having the type desired by the client was evaluated by means of a `let` definition and reused after, in both cases.

Working with OCLE, I noticed that the specification can be done in a simpler manner like:
```
let couldBeBookedR(tr:RoomType, st:Date, en:Date):Boolean = self.roomTypes->any(rt | rt = tr).rooms
->reject(r | r.booked->exists(b |b.isBooked))->notEmpty
```

specification which returns the same values widht

```
let couldBeBooked(tr:RoomType, st:Date, en:Date):Boolean = self.roomTypes->any(rt | rt = tr).rooms
->exists(r | r.booked->isEmpty) or
      self.roomTypes->any(rt | rt = tr).rooms->reject(r | r.booked->exists(b |b.isBooked))->notEmpty
```

I noticed that the usage of the `any(…)` operation twice, in the specification proposed at seminars was a bad idea. The composition of `reject(…)` and `exists(…)` operations as you can view in above specification is a better solution as you can notice from the attached printScreens.

Initially, the specification of the observer existent in the class Booked was named `isBooked(…)`. In my opinion, in this case, the name is not important. If the Boolean value returned by these functions is concordant with their names all is OK. So, the function was renamed to, `isNotBooked(...)`. This must return `true` if the room was not booked for the period of interest for the client, and `isBooked(…)`, `false` (in the same context). In the solution attached, I named the function `isBooked(…)`. As I didn't specified the observer, in order to evaluate `couldBeBooked()`, I replaced the observer with a Boolean attribute `isBooked`.

In the printScreen containing the evaluation results the RoomType parameter was set two times for each RoomType value {rt1, rt2, rt3, rt4}. The value mentioned bellow the parameter is the value of the observer (one for the first and another for the second). In the printScreen snapshots, the instances of the associated class Booked are named A_r6_c2, A_c1_r6 (A from the Association, r6 from room 6, c1 and c2 for client 1 and client2. The difference in notation is due to the fact that this name was automatically specified and, when the association was navigated from r6 towards c2 irrespective from c1 towards r6. This does not matter because it is about a bidirectional association.

At the end of my email, I would like to say you that, I am disappointed by the lack of interest of most of students. The level and amount of knowledge and skills of any of us can be improved. It's pity to notice that young people seem to be convinced that they know enough and a technology or other they manage is the best. Technologies change rapidly – principles and technology independent specifications are more stable.

A question for Java masters. In Java, does the evaluation of followings expressions, return the same answer?

2 = null
null = 2

Best regards,
Dan Chiorean