Prof. Dr. Peter Thiemann
Sergio Feo-Arenis
Sergiy Bogomolov

Summer Term 2014

Software Engineering

http://proglang.informatik.uni-freiburg.de/teaching/swt/2014/

Exercise Sheet 6

Exercise 1: OCL Collection Extension (5 Points)

Let *col* be a collection in OCL. Define OCL expressions that implement the following operations:

1. `hasNElements`: Returns `true` for some number n and some expression *expr*, if there exist exactly n elements in *col* that fulfill *expr*. The iteration variable in *expr* is `it`.
2. `isUnique`: Returns `true` if *col* does not contain duplicates. Do *not* use the builtin function of the same name.

Solution

1. `hasNElements`:

   col`->hasNElements(`*n*`, it|`*expr*`) = (`*col*`->select(it|`*expr*`)->size() = n)`

2. `isUnique`:

   col`->isUnique() = (`*col*`->asSet()->size() = `*col*`->size())`

Exercise 2: Class Extension (5 Points)

1. In the lecture, you have seen a precondition for the operation `move` of class `Meeting`. Refine this precondition so that meetings in different locations can take place at the same time. Keep in mind that each team member can only be in one location at one time.
2. The class `Meeting` from the lecture gets now extended by an operation

   `relocate(newLocation : Location)`

   which changes the location of a meeting. Find sensible pre- and postconditions for `relocate`.

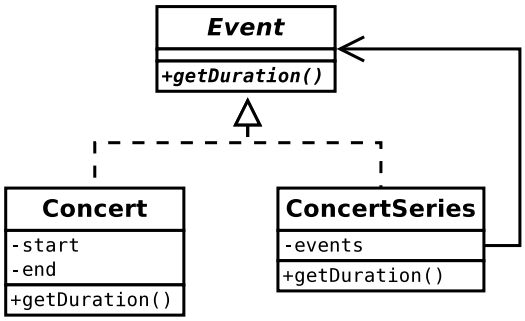
Solution

1.
```
context Meeting::move (newStart : Date)
pre: Meeting.allInstances()->forAll (m |
        m<>self implies
          disjoint(m, newStart, self.duration)
          or ((m.Location <> self.Location)
            and m.participants->excludesAll(self.participants)))
```

2. context Meeting::relocate (newLocation : Location)
   pre: Meeting.allInstances()->forAll (m |
           m<>self implies
             disjoint(m, start, self.duration)
             or (m.Location <> newLocation))
   post: self.Location = newLocation

Exercise 3: Composite and Visitor (10 Points)

In this exercise, we will consider the organization of tours for bands. A *concert* is modelled as an *event* with a certain duration. For simplicity, the duration is modelled as the difference of two values *start* and *end*. A *series of concerts* is modelled as a class that contains several concerts or series of concerts. The duration of a series of concerts is determined by the difference of the earliest beginning of and the latest ending of a concert contained in the series. The corresponding class diagram is defined as follows.



We consider the series of concerts *World Tour*. A world tour consists of the series of concerts *Europe Tour*, *Asia Tour*, and *America Tour*. Each series of concerts consists of several performances of type concert (e.g., "Live at Royal Albert Hall", "One Night in Bangkok", "Live at Madison Square Garden").

1. Become acquainted with the *composite pattern* [1]. Based on this pattern, provide an Java-like implementation that computes the duration for the World Tour series. Recall that the duration of a series of concerts S is determined by the difference of the earliest starting point and latest end point of a concert in S. Note that concerts may take place in parallel.

2. Apply the *visitor pattern* to provide an Java-like implementation that computes the duration for the World Tour.

3. Compare your solutions obtained with the composite and the visitor pattern. In this case, which pattern should be preferred for the computation of the duration?

Solution

See an attached file on the homepage of the lecture.

[1] http://en.wikipedia.org/wiki/Composite_pattern

Submission

- Submit this sheet *before* the lecture of Thursdays.
- Late submissions will not be accepted.
- Deadline: Thursday 11:59 a.m.