Tutorial: basic class and sequence diagrams

September 23, 2012

Purpose

Check that you know how to read and write basic class and sequence diagrams. In principle, you do from Inf2C-SE!

Exercises

Everyone should become able to do these, and should have a go before the tutorial. If you have trouble ask your tutor.

1. Draw a class diagram that shows a class `Party`, a class `Food`, a class `Entertainer` and a class `Venue`. Each of `Food`, `Entertainer` and `Venue` should have an integer attribute `cost` and getter and setter methods for it. `Party` should have an operation with selector `getTotalCost`. (What should the signatures of these operations be?) For now, do not show associations between the classes.

2. It is decided that the `getTotalCost` operation should be implemented by the object of class `Party` asking an object of each class `Food`, `Entertainer`, `Venue` for the value of its `cost` attribute, summing them and returning the result. Draw a sequence diagram that shows this behaviour being invoked by an actor.

3. Since objects of our classes communicate, there should be associations among the classes in the class diagram. Add suitable associations. Show navigability and multiplicities.

4. Next consider a modification of the design in which, instead of containing a fixed `cost`, a `Venue` object will calculate its cost on request, using information about the timing of the event. Suppose that a `Party` can report its date, start time and duration. Consider two possible designs:

   (a) the relevant information is passed to `Venue` along with the request to calculate a cost; or

   (b) the `Venue` object will take a no-argument request for a cost as usual, but then will ask for the extra information it needs.

   Make versions of, or annotate, your diagrams to show the difference between the two options. Take special care with the second. What are the pros and cons of each design?

Less simple UML

Do these if you have time once you are confident with the basic exercises. If you don't do them now, do them later for revision.

1. Many interactions involve loops. Modify the `Party` example so that instead of a party involving a single entertainer it involves a collection of entertainers, whose costs have to be summed. First, think about how you could represent this in a sequence diagram and sketch possibilities. Then, look up sequence fragments in UML sequence diagrams and see how UML2 does represent this situation.

2. Consider a situation involving a callback. Show on a sequence diagram:

   (a) an object `a:A` sends a message `registerObserver(a)` to `b:B`, i.e. with a reference to itself as argument (assume `b:B` replies without sending any messages itself);

   (b) an actor sending `newValue(17)` to `b`;

   (c) `b`, as part of its response, sending message `notifyObservers()` to itself;

   (d) `b`, as part of its response to its own message `notifyObservers()`, sending message `notify()` to `a`;

   (e) `a`, as part of its response to `notify()`, sending message `getNewValue()` to `b`, which replies with 17.

   Put in nested activations and all return arrows, and check that what you've written makes sense (everything, except the initial message and the actor's message, should be caused by something).

   This is what happens in the Observer pattern, which we'll discuss later. Can you see what it's for, designwise?