5. Exercises: writing meta-model constraints in Xtend

Objectives

- To experience limitations of expressiveness of class diagrams
- To practice precise expression of constraints over structural models
- To identify constraints in a realistic project (JUnit) and express them

I estimate that these exercises can be done within 5 hours in total.

I recommend that you use similar architecture as in the lecture example on state machines

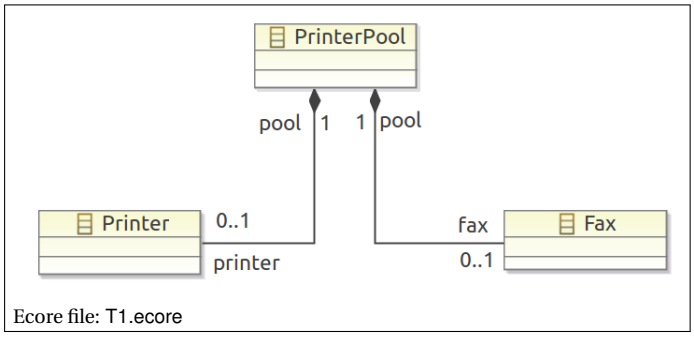
- Implement your constraints as static boolean methods with dynamic overloading over contexts
- Use a simple runner like in he lecture to test them

We provide a project containing the printer model[1] and an example project with the JUnit concept model[2]. These projects are already configured with the right build environment and should work out of the box on your Eclipse installation.

If you decide to use your own model, move it to my project, or configure your project to access EMF code: i) generate model (see end of the exercise Generating Model Code) code for your model and ii) add the suitable jar file to build path in order to access EMF runtime (`org.eclipse.emf.ecore.xmi` found in plugins directory of your Eclipse installation).

Let us first start with an example (ecore files available in `episode-12/dk.itu.smdp.printers` in our git repository):

[1] `episode-12/dk.itu.smdp.printers`
[2] `episode-12/dk.itu.smdp.junit.conceptModel`



Ecore file: T1.ecore

In order to write this constraint in Xtend (in our setup) we need to create a genmodel and generate model code for T1 (these steps have been already done before publishing the repository, but you may want to reload the genmodel and to regenerate the code with your installation of Eclipse and EMF).

Let's consider constraint that there must be a printer in the printer pool:

```
def static dispatch boolean constraint (PrinterPool it) {
     printer != null
}
```

We place the constraint in the file `Constraints.xtend`. You can check the constraint by executing the file `Main.xtend` as a simple Java application. The file name of the instance (which you, of course, need for checking constraints!) is set directly in the main method as a constant string.

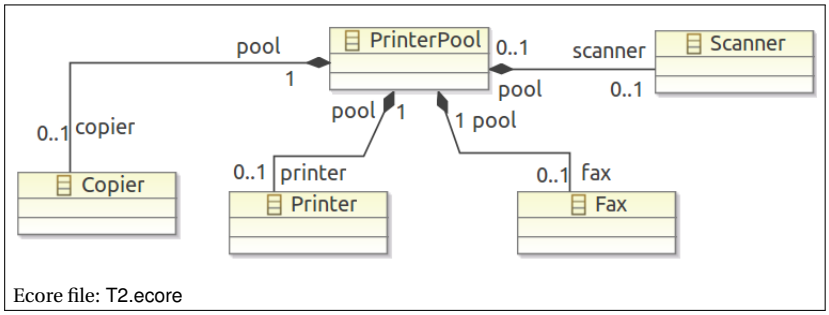
Task 1. Write each of the following constraints in Xtend. You are encouraged to perform these exercises using Eclipse (as opposed to on paper). The entire Task 1 can be solved by just filling the blanks in `Constraints.xtend`. ⚠

1. *Every printer pool that has a fax, also has a printer.* Write the constraint in the context of the PrinterPool class, of the diagram T1 (above).

   Create an instance of the above model that satisfies the constraint and verify by running `Main.main` if this is indeed the case. Create an instance of the above model that violates this constraint and verify in that this is indeed the case. Repeat these steps for every constraint below, to sanity check (test) your constraints.

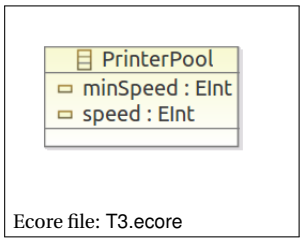
2. Write the constraint from the previous point in the context of (an instance of) class Fax.

3. Consider the new meta-model `T2.ecore` presented below. *Each Printer pool with a fax, must have a printer, and each printer pool with a copier must have a scanner and a printer.* Write this constraint in the context of the printer pool.



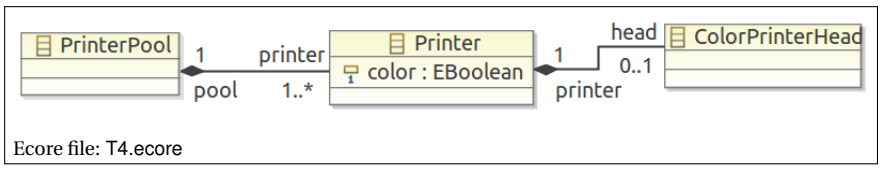
Ecore file: T2.ecore

4. *PrinterPool's minimum speed must be 300 lower than its regular speed.* Write the constraint in the context of the PrinterPool. Write the constraint in the context of the class Printer in the model `T3` (see below).



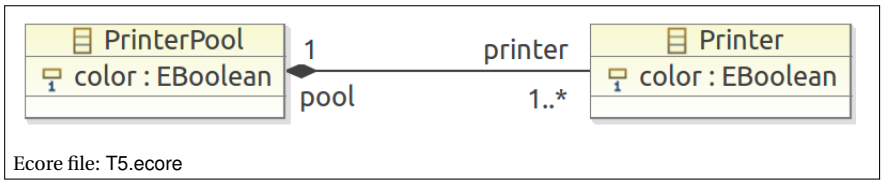
Ecore file: T3.ecore

5. *Every color printer has a colorPrinterHead.* Write the constraint in the context of the class Printer in the model `T4` (see below).



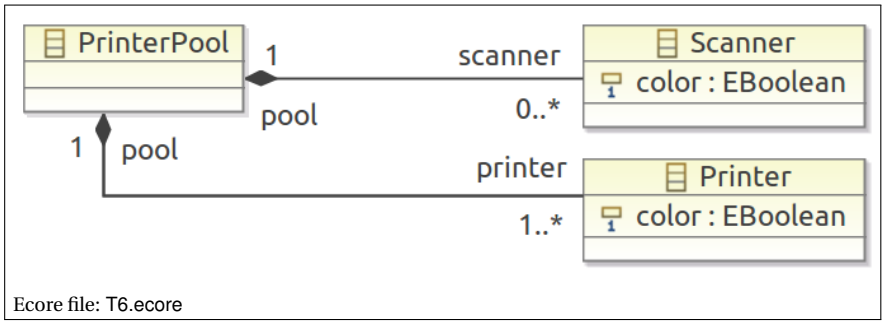
Ecore file: T4.ecore

6. *A color capable printer pool contains at least one color capable printer.* (using `T5`)



Ecore file: T5.ecore

7. *If a Printer pool contains a color scanner, then it must contain a color printer.*



8. *If a printer pool contains a color scanner, then all its printers must be color printers.* Use the same diagram as above.

9. Use the same diagram to assert that *there is at most one color printer in any pool.*

Task 2.* Now analyze your notes from last week's exercise, and identify constraints that cannot be expressed in the diagrams.

Once you have the constraints, write them in Xtend, and verify on example model instances.

For example, in my model any Before method must have @Before annotation. Annotations and fixtures are modeled separately in my model. I needed to add a constraint that the *method which takes the role of Before method with relation to a Fixture class, must also have an annotation, and that annotation is Before.*

You should be able to identify a small handful of similar constraints in your model. It is unlikely that there will be many, because class diagrams already allow to express most typical patterns. So only atypical, usually few, aspects of the model require use of extra constraints.

If you did not complete the model last week, or if you lack interesting constraints, or if you simply prefer to start with a clean project you can work with a standard start model. It is available in `episode-12/dk.itu.smdp.junit.conceptModel`. In this model we included some constraints as free text annotations in English. Implement them in `Constraints.xtend` class and verify on some instances whether you got it right. As usual use both positive and negative instances as tests. The advantage of using the provided project is that it is already configured, with appropriate build path, models, genmodel and generated code. It also has the constraints file prepared with English formulations, you just need to fill them in, using Xtend syntax. On the other hand you will train the workflow of generating model code and integrating constraints less.

Hand-In: Hand in a signed printout of your junit conceptual model (presumably the same or very similar to the one handed in last week) together with your constraints (for example the `Constraint.xtend` file). Deadline: before exercises the following week.

Generating Model Code. To generate Model code you have to create a genmodel file. Right click on your Ecore file, New → Other → Eclipse Modeling Framework → EMF Generator Model. Use the default path and name, then click Next. Select Ecore model and click Next. Then you need to load the Ecore file by pressing Load. Then press Next and then Finish. The file should open automatically. Now right click on the root and select Generate Model Code. If you make changes to the Ecore file, you need to regenerate the model code. You can reload the genmodel file by right-clicking on it, and selecting Reload. Then you generate the model code as before.

The mysterious genmodel file is just a configuration file for the code generator. If we wanted to modify the behaviour of the code generator, we would have to tinker with properties in this model. We are unlikely to need it in this course though, the default setup should suffice.

The so called "model code" that you generate are Java interfaces and classes for your meta-model, so that you can manipulate instances in Java programs.