Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. A. B. Cremers

Aspect-Oriented Software Development
Course in the summer semester 2009
Dr. Günter Kniesel, Daniel Speicher

Exercise Sheet 8

Due: Sunday 21.06.2009, 23:59:59 via SVN

For help, contact aosd-staff@lists.iai.uni-bonn.de (staff only) or aosd-course@lists.iai.uni-bonn.de (staff and participants).

Please start working on the exercises early enough so that you can contact us in time in case of problems. Don't expect us to be available during weekend!

Exercise 1: *"Visitor Pattern"* (15 Points)

This exercise is about implementing a generic variant of the visitor pattern (see attached file **aosd-exercisesheet_8-addendum.pdf**) and applying it to the ES08_E01_Visitor project in your repository.
The project contains a program that gets an HTML document from a URL and translates it to wiki syntax and prints it on the console. The URL from which HTML is read is http://roots.iai.uni-bonn.de/research/logicaj/downloads/test.html. It is hardcoded in the **Html2Wiki.java** class, which also contains the main() method. The output should look like this:

```
LogicAJ


====LogicAJ - An Uniformly Generic Aspect Language, short test introduction
for LogicAJDemo====


//LogicAJ// is an extension of AspectJ by uniform genericity and
interference analysis.

* **Uniform genericity**
lets aspects concisely model effects that
may vary depending on the static context of a join point.
Moreover, it alleviates the dependency of aspects on names
of base entities, making them more reusable and more
resilient to changes in base programs.

* **Interference analysis** enables unanticipated joint use of
independently developed aspects, for which there can be
no explicit order specifications (by dominates directives).
```

In the project you will also find two .aj files. **ConcreteVisitorAspect.aj** contains the concrete pointcuts that we have already implemented in order to apply the generic aspect to our project.

Your job is to implement all the missing parts of the file **AbstractVisitorAspect.aj**. This includes, in particular, the following pointcuts and introductions:

a) The abstract pointcuts whose concrete versions are implemented in the `ConcreteVisitorAspect`.

b) In order to generate an abstract visitor class you will need a pointcut and an introduction:
   i. The `pointcut abstractVisitorClass(?visitor)` should bind the `?visitor` meta-variable to the fully qualified name of the abstract visitor class. By convention this name should be the result of appending the string "`$Visitor`" to the fully qualified name of the base type of the visited object hierarchy. Make sure that the base type really exists. To do this use the LogicAJ "`class(Class)`" predicate.
   ii. `introduce introAbstractVisitorClass(?visitor)` then should create an empty class with `?visitor` as its name.

c) The created abstract visitor class must have a `visit()` and a `visitAfter()` method for each possible subtype of the base node type. To create these methods you should implement:
   i. `pointcut visitedType(?nodeType)`, a pointcut that binds `?nodeType` to any subtype of the base node type.
   ii. `introduce visit(?visitor,?nodeType)` and `introduce visitAfterMethods(?visitor,?nodeType)`. For every type matched by `visitedType(?nodeType)`, these introductions should create empty `visit(?NodeType node)` and `visitAfter(?NodeType node)` methods in the abstract class that has been created in part b).

d) Every type that should be visited has to have an accept method that is introduced by `introduce acceptMethods(?visitor)`. The body of the introduction is already given in **AbstractVisitorAspect.aj**. You just have to fill in the missing pointcut and the missing types marked by ???? in the body.

e) The last step is to make the abstract visitor class the super type of all concrete visitor classes.