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 9

Due: Sunday 28.06.2009, 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).

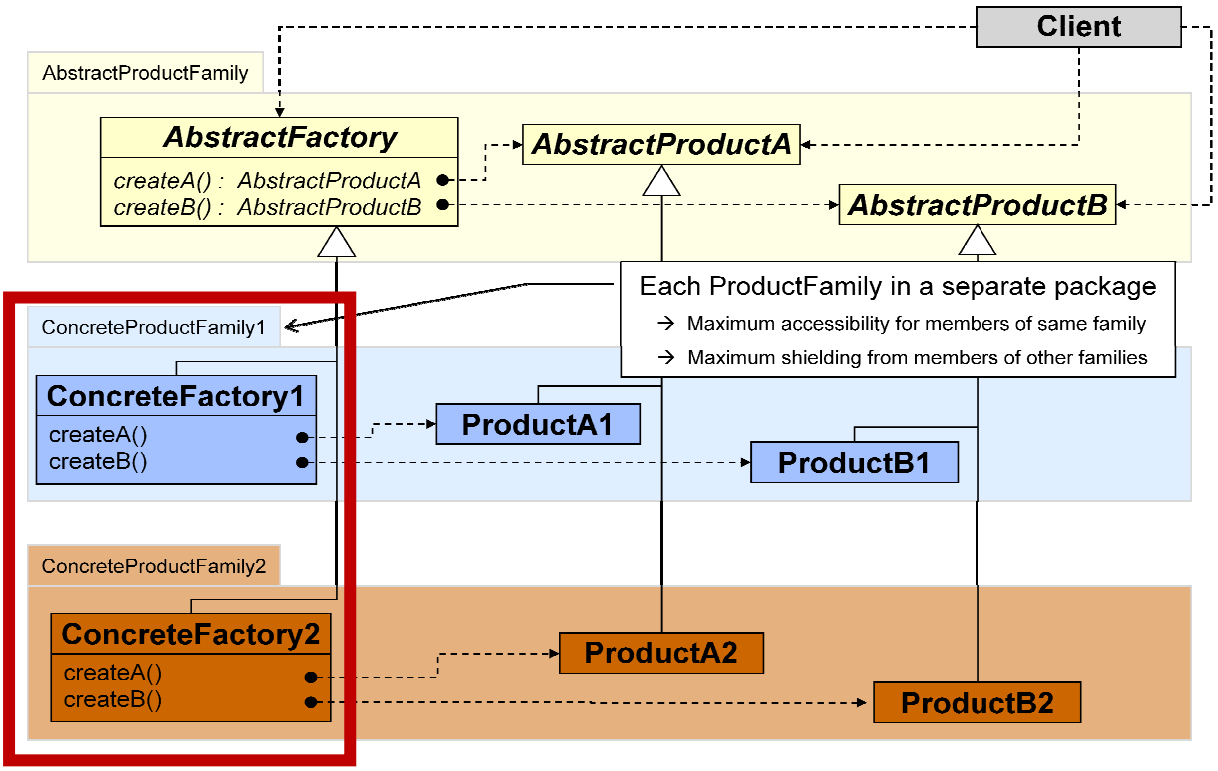
Please start working on the exercises early enough so that you can contact us in time in case of problems. Don't expect us to be available during weekend!

Preparation

Before reading any further, please refresh your knowledge of the "abstract factory" design pattern!

Provided material

In your repository you will find the project **ES09_E01_AbstractFactory**. Its main class (see **Main.java)** creates elements of an object structure using an instance of the abstract factory implemented in **AbstractFactory.java.** To get the unique factory instance the method `getFactory()` is used. For better encapsulation, each abstract or concrete product family is implemented in its own package. The structure of the application is described in the diagram below, the two concrete factories have to be implemented by you.



Exercise 1: *"(No) Abstract Factory with AspectJ"* (2 Points)

The categorization of Hanneman and Kiczales shows that it is not possible to implement the "*Abstract Factory Pattern*" with AspectJ. Discuss the shortcomings of AspectJ that might be the cause.

Exercise 2: *"Abstract Factory with LogicAJ"* (16 Points)

Your job is to implement the abstract factory pattern with LogicAJ. In particular:

- Create an abstract aspect named **AbstractFactoryAspect.aj** that implements the automatic introduction of concrete factory classes for an arbitrary product hierarchy contained in some package of the base application. The names of the classes in the product hierarchy and the name of the package containing the products must not be hard-coded in the aspect!

- Create a concrete subaspect **ConcreteFactories.aj** that implements
  - the concrete pointcuts specifying the package to which to apply the generic aspect (that is, the package for which the concrete factories should be generated)
  - an advice `around getFactoryMethod(String pckg, ?concreteFactory)` that generates the concrete factory when the `getFactory(String)` method is executed (provided it had not already been created at a previous invocation).

Here are some hints how to implement **AbstractFactoryAspect.aj**.

- Define an abstract pointcut that is supposed to match each package `?package` that contains a concrete product family for a given abstract factory class `?abstractFactory`:
  `abstractFactory(?abstractFactory, ?package).`

- Implement a concrete pointcut expressing that the name for the concrete factory class for the products contained in the package `?package`, is also contained in `?package` and has the name "Factory":
  `concreteFactory(?package, ?concreteFactory)`

- Implement an introduction that introduces an concrete factory class `?concreteFactory` that is empty but inherits from the related abstract factory class:
  `makeFactory(?concreteFactory)`

- Implement a pointcut `concreteAndAbstractProduct(?concreteFactory, ?abstractFactory, ?abstractProduct, ?concreteProduct, ?methodname, ??params)`
  that relates:
  - a concrete factory `?concreteFactory`
  - its abstract factory `?abstractFactory`
  - the signature (return value `?abstractProduct`, method name `?methodname`, parameter `??params`) of a method of the abstract factory
  - a concrete product `?concreteProduct` created by the respective method of the concrete factory `?concretFactroy`

- Introduce into the concrete factories `?concreteFactory` all concrete factory methods that it must contain according to the above pointcut:
  `makeFactoryMethods(?concreteFactory).`