SESSION 3: JASCO EXERCISES

Using the JAsCo-plugin for Eclipse.

For this exercise session, the Eclipse IDE and the JAsCo plugin are employed. Short User Guide:

- For writing a JAsCo-enabled application, create a new JAsCo project. The base behaviour of a JAsCo project (i.e. the application itself) is implemented by making use of regular Java classes.
- For implementing a JAsCo aspect bean, add a "JAsCo aspect" to the project.
- To connect the aspects to the concrete classes, add a "JAsCo connector" to the project.
- To enable aspect application, just run the project using a JAsCo runner (run→ run… → JAsCo Application → new), the aspects are automatically inserted at load-time. Make sure to employ Java 1.5 as running JRE!
- Use help→help contents→JAsCo help, for help for using the JAsCo IDE and for viewing the language reference.

Ex 0: AspectJ again

Import the figures example, available at the website.
Do this by File→import.

a) Implement a tracing aspect that traces the execution of all the classes contained in the figures.* package of the figures example (only these!). The trace prints the name of the method entered and the class on which the method is executed. When exiting the method, the trace shows this. The trace also includes the nesting level by adding tabs. Use an **around** advice to implement the tracing.

Example trace output:

→ entering main in class Test
    → entering bibi in class Test
    ← exiting bibi in class Test
    → entering baba in class Boe
        → entering do in class Door
        ← exiting do in class Door
    ← exiting baba in class Boe
← exiting main

b) Make it possible to define the maximum print depth. Only until invocation depth x, tracing info is printed.

c) Now try to reuse the same tracing aspect onto a different application with a different print depth. How can you do this? Just provide the aspects you need to create and some new dummy class names for the new application.

Ex 1: The basics

Import the test project available at the website. Because this project uses Java 1.5 features, you have to make sure to use the ant builder for compiling the project. This should already been configured correctly. For compiling the project, use project→clean. The Eclipse eager parser will still show errors in the project, but it should work fine. Test it first by running the project's main class Test (use a JAsCo runner!).

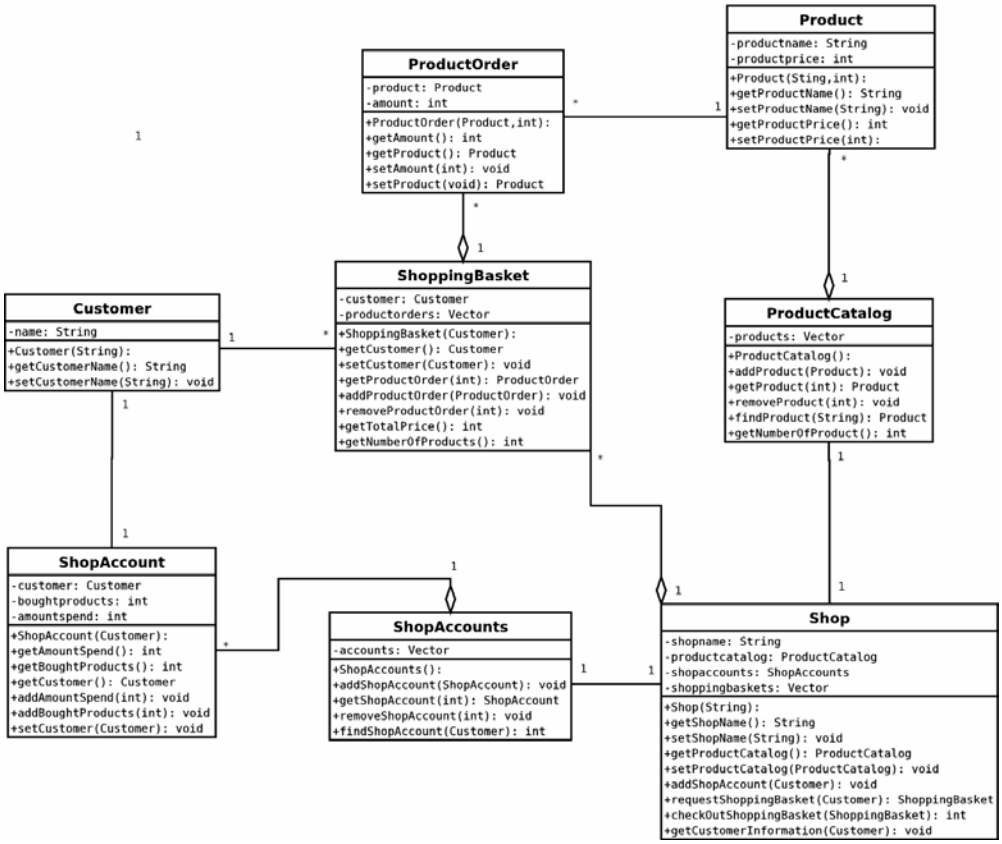
a) Write an aspect and connector that prints the method name after the start method is executed.
b) Write an aspect and connector that only prints the method name before the stop method is executed form the start method.
c) Write an aspect and connector that applies each time any method is executed in class Test. The aspect implements a before advice that prints out all annotations associated with the current joinpoint.
d) Write an aspect and connector that print a message each time a method with annotation `TestAnnotation` is executed.
e) Write an aspect and connector that prints a message each time an exception is thrown in class Test. *Use after throwing*
f) Write an aspect and connector that captures exceptions thrown in class Test and makes sure they are not rethrown. *Use around throwing*
g) Write an aspect and connector that only print a message when a String is returned from any given method in class Test. *Use after returning*
h) Write an aspect and connector that replaces the return value for methods returning a string in "biebabeloeba". *Use around returning*

Ex 2: Runtime adaptation: Juggler

Import the juggler project available at the website. If necessary: add the juggler.jar to the build path. Do this by Project→properties→java build path→libraries→add external jar. Test the application by running the RunJuggler class (use a JAsCo runner!).

a) Add an aspect bean that implements an around advice, which prints the message "replaced" and does NOT invoke the original behavior.
c) Instantiate the aspect bean onto the `startJuggling` and `stopJuggling` methods so that the invocation of these methods has no effect any longer. Observe the run-time behavior.
d) Try using the introspector () to disable and enable the connector at run-time and observe the run-time program adaptation.
e) Subtype patterns exercise: Add a logging aspect that only logs method executions on subtypes of java.awt.Component

Ex 3: Logging in The ECommerce Shop



Import the ecommerce example, available at the website.
Do this by File→import.

a) Implement a tracing aspect that traces the execution of all the classes contained in the Ecommerce Shop (only those!). The trace prints the name of the method entered and the class on which the method is executed. When exiting the method, the trace shows this. The trace also includes the nesting level by adding tabs. Use an **around** advice to implement the tracing. You can reuse code from the tracing aspect of Ex0.

Example trace output:

→ entering main in class Test
    → entering bibi in class Test
    ← exiting bibi in class Test
    → entering baba in class Boe
        → entering do in class Door
        ← exiting do in class Door
    ← exiting baba in class Boe
← exiting main

b) Now imagine having to apply the tracing aspect to a different application. What needs to change? Compare this to the AspectJ solution.

Ex 4: Access Control in The ECommerce Shop

Import the ecommerce example, available at the website.

a) Implement an access control aspect that validates the identity of the current user, only users supplying the correct password can employ the shop. Once validated, the user is logged in and no additional password queries should be done. The following methods are subject of access control behavior:

- AddProductOrder and GetProductOrder methods in ShoppingBasket

Hint: The following code fragment prompts for a password:
String password = javax.swing.JOptionPane.showInputDialog(new javax.swing.JFrame(), "secret word");

You can assume that only one user is logged in at the same time.

b) - Add a refinable method for the no access policy (when access control has failed). The default implementation of the refinable method throws an exception. Refine the method in the connector to log to the error stream.

- Add a refinable method for retrieving the customer so that the aspect is completely independent from the concrete application. This method does not have a default implementation

c) Alter the aspect in such a way that the aspect is not triggered any longer when an administrator is logged in. An administrator can access the complete system, so no access control has to be performed any longer. (HINT: use the *isApplicable* construct)

d) Add the logging aspect from ex 3, make sure that logging is always executed before the access control aspect. (HINT: employ a precedence strategy)

e) Add another instance of the logging aspect that logs before the method is executed, but after the access control check has passed.

f) Make sure that the no logging is performed when an administrator is logged in. HINT: when an administrator is logged in, the access control aspect does not trigger. As such, make sure the logging aspects cannot trigger either when the access control aspect does not trigger by employing combination strategies.