TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Software and Multimedia Technology, Software Technology Group

SS2018 – Model-driven Software Development in Technical Spaces

Aspect-Oriented Programming (AOP)

Professor: Prof. Dr. Uwe Aßmann
Tutor: Dr.-Ing. Thomas Kühn

Task 1 Aspect-Oriented Programming (AOP)

The goal of *aspect-oriented programming* (AOP) is to elucidate design decisions that cross-cut the system's basic functionality. According to Kiczales et al. [3] *aspect-oriented programming* (AOP) *"makes it possible to clearly express programs involving such aspects, including appropriate isolation, composition and reuse of the aspect code."*

a) Describe the terms *scattering* and *tangling*. Why are both effects bad?

b) What does *cross-cutting* mean wrt. AOP?

c) Describe the terms *Pointcut*, *Join Point*, *Advice* and *Weaving*. Give examples for each.

d) Where is the difference between *static* and *dynamic* join points?

e) Where is the difference between *static* and *dynamic* weaving?

f) Describe the *component model*, *composition technique* and *composition language* of `AspectJ`.

Task 2 Composition Filters

In contrast to AOP, one of the main goals of *Composition Filters* is to *"extend existing (object-oriented) programming models in a modular way, instead of replacing or adapting them"* [1].

a) What are the basic concepts of *Composition Filters*? How are they related? Draw an analysis class diagram.

b) In what ways can *Filters* adapt messages before passing them one to the next *Filter*?

c) Can the `java.util.stream` library of Java be considered a Composition Filter architecture?

d) Describe the *component model*, *composition technique* and *composition language* of Composition Filters [1]?

Task 3 Programming with `AspectJ`

In this task, you will learn how learn how aspect-oriented software is implemented. You are tasked to write simple aspects for a Java application using `AspectJ`, an aspect-oriented extension to Java. Luckily, there exists direct *Eclipse* support by means of the *`AspectJ` Development Tools* (AJDT) [2].

As our example we consider a tiny library that represents the structure of houses. The library offers an interface to construct houses. In the current version any combination of `ComplexHouseParts` is possible (e.g., a level can contain hallways, which can contain levels etc.). Furthermore, rooms can either be clean or not clean. They can be cleaned by calling the `setClean(boolean clean)` method. House parts can be entered and visited. Visiting a house part means entering the part itself and visiting all the subparts. However, as we want to use our library for any kind of domain, we want to exclude the consistency checking code from the base implementation. Rather than, we want to use customer-specific aspects, which are woven into the base implementation. Use `AspectJ` to implement the following features.

a) Download and install `AspectJ` and the corresponding example.

   1. Install the *`AspectJ` Development Tools* (AJDT).[1]
   2. Download the source of the core project from the CBSE website.
   3. Edit the core project to solve the task (by adding aspects).

b) Define an Aspect that logs (prints to the console) when the program starts and stops running.
c) Define an Aspect, which restricts that levels cannot contain other levels.
d) Define an Aspect, which marks a bathroom as not clean, when it is entered.
e) Define an Aspect, which prints a warning when a room is entered which is not cleaned.
f) Define an Aspect, which prohibits (e.g., throws an exception) any person entering a bathroom which is not cleaned.
g) In the initialization Script (`buildSimpleHouse()`), a house with a private room is created. Define an aspects, which prohibits any other person that the owner of the house, to enter the private room.

Hand in your solution as archive `*.zip` before the next exercise.

[1] Choose the right update site wrt. your Eclipse version `http://www.eclipse.org/ajdt/downloads/index.php`

Additional Information

- `AspectJ` Development Tools[2]
- AJDT Documentation[3]
- Tutorials to getting started[4]
- More elaborate tutorial[5]

References

[1] Lodewijk Bergmans and Mehmet Aksit. Principles and design rationale of composition filters. *Aspect-Oriented Software Development*, pages 63–95, 2004.

[2] Adrian Colyer, Andy Clement, George Harley, and Matthew Webster. *Eclipse aspectj: aspect-oriented programming with aspectj and the eclipse aspectj development tools.* Addison-Wesley Professional, 2004.

[3] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *European conference on object-oriented programming*, pages 220–242. Springer, 1997.

[2] https://www.eclipse.org/ajdt/
[3] https://eclipse.org/aspectj/doc/released/progguide/starting.html
[4] https://eclipse.org/ajdt/demos/
[5] https://o7planning.org/en/10257/java-aspect-oriented-programming-tutorial-with-aspectj