Programming Languages

Dr. Michael Petter, Raphaela Palenta WS 2018/19

Exercise Sheet 11

Assignment 11.1 Tracing

Write an ascpect that logs method calls and returns as well as field accesses of objects. Test your implementation with a small program that creates an object with some fields. Call a method on this object. This method should in fact then call some other method. Report whether a method returned regularly or by throwing an exception. Add indentation to your logging output such that your output looks e.g. like this

```
Entering void TestClass.f()
 Entering void TestClass.g()
  Entering void TestClass.h()
   int TestClass.pub is accessed
  Method void TestClass.h() returned
 Method void TestClass.g() returned
Method void TestClass.f() returned
```

Suggested Solution 11.1

⇒ `TestLogging.java`

Assignment 11.2 Default Implementation for Interfaces

Aspects can be used to provide default implementations for interfaces. Consider the following example of an interface.

```java
interface Sortable {
        public int compare(Object other);

        public boolean equalTo(Object other);

        public boolean greaterThan(Object other);

        public boolean lessThan(Object other);
}
```

Implement the three methods `equalTo(Object other)`, `greaterThan(Object other)` and `lessThan(Object other)` using the method `compare(Object other)` in an aspect `DefaultSortableAspect`. Provide a class `Sort implements Sortable` representing an integer number. Therefore give an implementation for the method `compare(Object other)` and test your implementation.

Suggested Solution 11.2

⇒ `Sort.java`

Assignment 11.3 Access restriction

Write some aspects for fine-grained access restrictions. Test your implementation with two "toy" classes.

1. Write an aspect that ensures that objects of class `A` can only be created within methods of class `B`.
2. Write an aspect that ensured that there is no write access to non-private fields.
3. Write an aspect to make a field "object-private" i.e., it may only be accessed from within the same object, not from other objects (of the same class).

Suggested Solution 11.3

⇒ `LittleAOP.java` (+ `A.java`, `B.java`)