Program Monitoring

Lecture 3 : Monitoring with AspectJ

Monday May 11, 2009

This third lecture illustrates through examples how to write monitors in AspectJ. The note in addition contains 1 assignment (the first in this second part of CS 119). Finally, the note mentions some additional installation of weak identity hash-maps and sets.

Reading

See note for lecture 1 (same reading): basically learning how to use AspectJ.

Assignment 1

To be submitted to `havelund@gmail.com` before Monday May 18, at 11:59 pm.

Consider the availability of a class `Stack` implementing the following interface:

```
interface StackInterface {
  public void push(Object t);
  public Object pop();
  public Object top();
  public boolean isEmpty();
  public int size();
}
```

The following program uses the `Stack` class by creating an object of this class, and then pushing a series of elements on the stack and popping them off again:

```
class Test {
  public static void main(String[] args) {
    StackInterface stack = new Stack();

    for (int i = 0; i < 1500; i++) {
        stack.push(i);
    }

    for (int i = 0; i < 1500; i++) {
        stack.pop();
    }
  }
}
```

Running this program reveals no errors. Your job is to write one or more aspects that check correct use of the stack as well as the correctness of the stack by monitoring calls of the stack operations specified in `StackInterface`. That is, check that the following rules are obeyed:

1. The `top()` and `pop()` methods are never called on empty stacks in the `Test` class (checked by the `isEmpty()` method).
2. The `push(StackInferface s)` method increases the stack size by 1, and `pop()` decreases the size by 1 (the value returned by the `size()` method).
3. The `Stack` class generally implements a stack as one would expect. That is: elements are popped off in the reverse order that they are pushed. For example, the following program should not cause any assertions to be violated.

```
stack.push(1); stack.push(2); stack.push(3);
assert (Integer)stack.pop() == 3;
assert (Integer)stack.pop() == 2;
assert (Integer)stack.pop() == 1;
```

Hints: The properties above are increasingly complex to solve. Property 3 can be solved by comparing the stack behavior with the *reference implementation* provided by SUN: `java.util.Stack`.

The interface `StackInterface` and the classes `Stack` and `Test` mentioned above are accessible from the course website.

Installation

In order to write monitors, we need various forms of sets and maps. the `java.util` library provides the basic `HashMap` and `HashSet`. In addition we need various combina-

tions of these being *weak* (garbage collector removes entries when the monitored application no longer refers to them) and/or *identity* (entries are compared with == and not by the `equals()` method). Again, `java.util` offers `WeakhashMap` and `IdentityHashMap`. We further need `IdentityHashSet`, `WeakIdentityHashSet`, and `WeakIdentityHashMap`. These are downloadable from the website. They require the apache commons collections library. The instructions are below.

1. Install the apache commons collections package:

   `http://commons.apache.org/collections/`

   This package is used to implement the following classes.

2. Install the following (from the course website):

   (a) `IdentityHashSet`

   (b) `WeakIdentityHashSet`

   (c) `WeakIdentityHashMap`