OpenJMLUnit: Improvements to JMLUnit

A Preview

Daniel M. Zimmerman
Institute of Technology
University of Washington Tacoma

JMLUnit

- JMLUnit is an excellent tool, but has several shortcomings:

JMLUnit

- JMLUnit is an excellent tool, but has several shortcomings:
  - Generated tests are only as good as the specs... not much to do about this one but write better specs—that's why we're here!

JMLUnit

- JMLUnit is an excellent tool, but has several shortcomings:
  - Generated tests are only as good as the specs... not much to do about this one but write better specs—that's why we're here!
  - You need to create test data, and sometimes extra code, and place it in various parts of the generated test files.

JMLUnit

- JMLUnit is an excellent tool, but has several shortcomings:
  - Generated tests are only as good as the specs... not much to do about this one but write better specs—that's why we're here!
  - You need to create test data, and sometimes extra code, and place it in various parts of the generated test files.
  - It can use extreme amounts of memory very, very easily.

Consider A Trivial Class

```
public class Add
{
  //@ invariant sum() > 0;

  private int my_x;
  private int my_y;

  //@ requires the_x > 0;
  //@ requires the_y > 0;
  //@ ensures x() == the_x;
  //@ ensures y() == the_y;
  public Add(final int the_x, final int the_y)
  {
    my_x = the_x;
    my_y = the_y;
  }

  public /*@ pure @*/ int x() { return my_x; }
  public /*@ pure @*/ int y() { return my_y; }

  //@ ensures \result == x() + y() + the_operand;
  public /*@ pure @*/ int sum(final int the_operand)
  {
    return my_x + my_y + the_operand;
  }
}
```

Test Generation with JMLUnit

- Running JMLUnit creates 2 Java classes.
  - One (the unit tests) you leave alone.
  - The other is the unit test data... it's 162 lines long, and the two places you need to edit are on lines 122 and 154.
  - If you want to generate different sets of integers for different parameters, you need to write code.

Test Generation with OpenJMLUnit

- Running OpenJMLUnit will also create 2 Java classes.
  - One (the unit tests) you leave alone.
  - The other is the unit test data... at the very top of the class you fill in sets of test data for every context.
  - You can specify separate data sets for each parameter of each method, and also for each necessary data type globally.

Another Improvement: Objects To Test

- JMLUnit does not automatically construct objects for you to test (though it does test constructors)—you have to decide on them yourself.
- OpenJMLUnit can automatically use the test data you specify for the constructor to construct objects for testing the other methods... and you can add more yourself.

Memory Considerations

- JMLUnit uses very large amounts of memory for large test sets, because it constructs all the JUnit tests before feeding them to JUnit to run.
- New JMLUnit will use TestNG's support for *parameterized tests with iterators* to lazily provide test cases and eliminate the need to construct them all in memory first.

Memory Considerations

- A year or so ago, Joe and I wanted to try a test data selection method we came up with on the KOA counting system.
- We added our test data to JMLUnit and ran it on a 3GHz 8-core server with 16GB of heap for over a month.
- It never finished or gave us any output—it spent all its time garbage collecting!
- Hopefully OpenJMLUnit can do better!

Current Status

- An interim piece of software, JMLUnitNG, is currently under development by M.S. student Rinkesh Nagmoti.
  - Still using JML2.
  - Maybe I'll get to demo it this week...
- I will begin working on OpenJMLUnit soon.