101JMLSpecifications

Rodrigo Bonifácio
(rbonifacio@cic.unb.br)

Motivation

There exist several constraints
(informally)
specified in the 101 Wiki

Description

The system must support the human resources domain in the following manner.

There are companies, departments, and employees.

- Each company has a unique *name*.
- Each company aggregates a possibly empty list of *departments*.
- Each department has a unique *name* across the company.
- Each department must have a manager.
- Each department aggregates possibly empty lists of *employees* and *sub-departments*.
- Each employee has a *name*.
- Employees have additional properties for *salary* and *address*.
- Each employee serves only in one position in one company.
- Managers are employees, too.
- All properties (names, addresses, salaries) must be not null.

This data model is interesting in so far that it immediately exercises various facets of data modeling such as properties, cardinalities, recursion, and containment. Concrete implementations may easily assume refinements of the specification, if additional facets should be covered. For instance, inheritance can be exercised by deriving managers from employees through specialization. Further, the basic model at hand essentially suggests containment relationships (i.e., tree shape), but it is easy to involve reference relationships; see, for example, Graph structure. Finally, the basic model and straightforward refinements are suitable for the illustration of major programming techniques and design patterns. For instance, the basic structure at hand may suggest application of the Composite pattern.

Add comments: - 1 company versus many - uniqueness of employees

Illustration

Here is an illustrative description of a company called "meganalysis".

We use some concrete syntax to render the structure.

Some *101companies* Implementations support (a variant of) this concrete syntax.

See 101implementation:antlrAcceptor for example.

```
company "meganalysis" {
  department "Research" {
```

101feature:Global invariant

Path: Base → 101companies → 101main → 101feature → Structural 101feature → **Global invariant**

Headline

--- A constraint on salaries within the company hierarchy ---

Description

A manager of any department or sub-department is required to receive a salary that is higher than the salaries of all employees of the department and all sub-departments. (It is clear that this constraint is not universally adopted by companies in practice.) The constraint is interesting in so far that many type systems will not be able to model this constraint directly, but instead the constraint may need to be implemented explicitly by traversing the company structure.

Citations

Features

- Attribute editing
- Precedence

Contributions

- 101implementation:emftext
- 101implementation:gwt
- 101implementation:gwtTree
- 101implementation:html5tree
- 101implementation:javaExorcism

Are those constraints being checked?

Ar nstraints being d?

We evaluate this using the javaComposition implementation

Lets (formally) specify those constraints using JML

"JML (Java Modeling Language) is a formal behavioral interface specification language for Java"

Design by Contract with JML (Gary T Leavens and Yoonsik Cheon)

Constraints on employees' state

- name, address, and salary must be not null
- salary must be greater than zero

Employee JML Specification

```
/**
 * An employee has a name, an address, and a salary.
 */
public class Employee implements Serializable {

    private static final long serialVersionUID = -2108895926771652 50L;

    private /*@ spec_public non_null @*/ String name;
    private /*@ spec_public non_null @*/ String address;
    private /*@ spec_public @*/ double salary;

    //@ public invariant salary > 0;
```

A JML4C compiles the source code into byte code, but with runtime checkers

JMLUnitNG generates unit tests to check those constraints

JMLUnitNG generates unit tests to check those constraints

- a skipped test case does not satisfy either an invariant or a precondition.
- a failed test case satisfies all invariants and preconditions, but at least one postcondition does not hold

Test Results

```
bash-3.2$ make runTests
ant -f runTestNG.xml
Buildfile: /Users/jml/workspace/101JMLSpecifications/runTestNG.xml

test:
   [testng] [TestNG] Running:
   [testng]   Ant suite
   [testng]
   [testng]
   [testng] ===================================================
   [testng] Ant suite
   [testng] Total tests run: 13, Failures: 1, Skips: 11
   [testng] ===================================================
   [testng]
   [testng] The tests failed.

BUILD SUCCESSFUL
Total time: 2 seconds
```

Why so many tests were skipped?

The Employee class does not have a constructor that leads to a valid state. For this reason, it is not possible to construct a valid Employee using the JML constraints on slide 8

Test results

1 suite, 1 failed test

```
test_getSalary__0 ()
  "org.jmlspecs.jmlunitng.testng.PreconditionSkipException: could not construct an object to test
          at org.softlang.model.Employee_JML_Test.test_getSalary__0(Employee_JML_Test.java:315)
  ... Removed 23 stack frames
    org.jmlspecs.jmlunitng.testng.PreconditionSkipException: could not construct an object to test
          at org.softlang.model.Employee_JML_Test.test_getSalary__0(Employee_JML_Test.java:315)
          at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
          at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
          at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
          at java.lang.reflect.Method.invoke(Method.java:601)
          at org.testng.internal.MethodInvocationHelper.invokeMethod(MethodInvocationHelper.java:80)
          at org.testng.internal.Invoker.invokeMethod(Invoker.java:715)
          at org.testng.internal.Invoker.invokeTestMethod(Invoker.java:907)
          at org.testng.internal.Invoker.invokeTestMethods(Invoker.java:1237)
          at org.testng.internal.TestMethodWorker.invokeTestMethods(TestMethodWorker.java:127)
          at org.testng.internal.TestMethodWorker.run(TestMethodWorker.java:111)
          at org.testng.TestRunner.privateRun(TestRunner.java:767)
          at org.testng.TestRunner.run(TestRunner.java:617)
          at org.testng.SuiteRunner.runTest(SuiteRunner.java:334)
          at org.testng.SuiteRunner.runSequentially(SuiteRunner.java:329)
          at org.testng.SuiteRunner.privateRun(SuiteRunner.java:291)
          at org.testng.SuiteRunner.run(SuiteRunner.java:240)
          at org.testng.SuiteRunnerWorker.runSuite(SuiteRunnerWorker.java:51)
          at org.testng.SuiteRunnerWorker.run(SuiteRunnerWorker.java:85)
          at org.testng.TestNG.runSuitesSequentially(TestNG.java:1197)
          at org.testng.TestNG.runSuitesLocally(TestNG.java:1122)
          at org.testng.TestNG.run(TestNG.java:1030)
          at org.testng.TestNG.privateMain(TestNG.java:1337)
          at org.testng.TestNG.main(TestNG.java:1306)
  "
```

Lets implement new constructors for the Employee class

```
public Employee() {
  name = "";
  address = "";
  salary = MIN_SALARY;
}
```

```
/*@
  @ requires pname != null && paddress != null && psalary > 0.0;
  @ ensures name == pname
  @       && address == paddress
  @       && salary == psalary;
  @*/
public Employee(String pname, String paddress, double psalary) {
  name = pname;
  address = pname;
  salary = psalary;
}
```

Test Results

```
bash-3.2$ make runTests
ant -f runTestNG.xml
Buildfile: /Users/jml/workspace/101JMLSpecifications/runTestNG.xml

test:
   [testng] [TestNG] Running:
   [testng]   Ant suite
   [testng]
   [testng]
   [testng] ===============================================
   [testng] Ant suite
   [testng] Total tests run: 25, Failures: 0, Skips: 17
   [testng] ===============================================
   [testng]

BUILD SUCCESSFUL
Total time: 2 seconds
```

But now our tests skip because they break our preconditions

Test results

1 suite

```
test_Employee__String_pname__String_paddress__double_psalary__20 (, , -Infinity)
  "org.jmlspecs.jmlunitng.testng.PreconditionSkipException:
  By method Employee.Employee
  Regarding specifications at
    File "/Users/jml/workspace/101JMLSpecifications/src/./org/softlang/model/Employee.java", line 27, character 18
  With values
    psalary: -Infinity
    paddress: ""
    pname: ""

        at org.softlang.model.Employee_JML_Test.test_Employee__String_pname__String_paddress__double_psalary__20(Employ
  ... Removed 23 stack frames
   org.jmlspecs.jmlunitng.testng.PreconditionSkipException:
  By method Employee.Employee
  Regarding specifications at
    File "/Users/jml/workspace/101JMLSpecifications/src/./org/softlang/model/Employee.java", line 27, character 18
  With values
    psalary: -Infinity
    paddress: ""
    pname: ""
```

What will happen if we run the same test suite against the original source code?

Test Results

```
bash-3.2$ make compileJava
make -C src-java compileJava
java -jar /Users/jml/tools/jml4c/jml4c.jar -d ../build .
bash-3.2$ make runTests
ant -f runTestNG.xml
Buildfile: /Users/jml/workspace/101JMLSpecifications/runTestNG.xml

test:
   [testng] [TestNG] Running:
   [testng]   Ant suite
   [testng]
   [testng]
   [testng] ===================================================
   [testng] Ant suite
   [testng] Total tests run: 25, Failures: 0, Skips: 0
   [testng] ===================================================
   [testng]

BUILD SUCCESSFUL
Total time: 5 seconds
```

No test case violates a precondition, but probably this occurs because the original implementation does not enforce any constraint

Lets specify some constraints applied to Departments and Companies

Department Constraints

- Name must be not null
- Manager must have a salary that is higher than the salary of any other employee of the department

Department JML Constraints

```
public class Department implements Serializable {

    private static final long serialVersionUID = -2008895922177165250L;

    private /*@ spec_public non_null @*/ String name;
    private /*@ spec_public non_null @*/ Employee manager;
    private /*@ spec_public non_null @*/ List<Department> subdepts;
    private /*@ spec_public non_null @*/ List<Employee> employees;

    //@ public invariant !name.trim().equals("");

    /*@ requires manager != null && employee != null;
      @ requires manager.getSalary() > employee.getSalary();
      @ requires (\forall Employee e;
      @                   employees.contains(e);
      @                   employee != e);
      @*/
     public void addEmployee(Employee employee) {
         employees.add(employee);
     }
```

Company Constraints

- Name must be not null... so, there is nothing new here.

Considering all constraints, JMLUnitNG generates more than 100 test cases

Test results

```
bash-3.2$ make runTests
ant -f runTestNG.xml
Buildfile: /Users/jml/workspace/101JMLSpecifications/runTestNG.xml

test:
   [testng] [TestNG] Running:
   [testng]   Ant suite
   [testng]
   [testng]
   [testng] ===============================================
   [testng] Ant suite
   [testng] Total tests run: 66, Failures: 4, Skips: 47
   [testng] ===============================================
   [testng]
   [testng] The tests failed.

BUILD SUCCESSFUL
Total time: 5 seconds
```

When we run this test suit against the original implementation, no test skips!

Are those constraints being checked?

Are these constraints being checked?

No!

In this way, we could not rely on simple assumptions

In this way, we could not rely on simple assumptions

```
@Test
public void testNegativeSalary() {
    Employee emp = new Employee("RBonifacio", "Brasilia", SALARY);
    double before = emp.getSalary();
    emp.cut();
    double after = emp.getSalary();
    assertTrue(before > after);
}
```

This test fails,
when SALARY <= 0

Summary

- We (manually) translate informal specifications of 101Companies available at a Wiki into formal JML specifications (reverse engineering)
- From the formal specification, we automatically generate code to check invariants, preconditions, postconditions, and unit testes (reengineering?).
- Findings: must constraints that are informally specified in the Wiki are not considered in the Java Implementations.
- This might be a new theme for exploring in the context of 101Companies