An introduction to formal specifications and JML

Invariant properties

Yves Ledru
Université Grenoble-1
Laboratoire d'Informatique de Grenoble
Yves.Ledru@imag.fr

2013

© Yves.Ledru@imag.fr 2013

Invariant properties

- Invariants are properties that should be true at « visible » instants in the execution of the class.
- Here, we expect that the 5 properties of SetAsTree are invariant!
- Invariant properties are expressed in JML as meaningful comments.

© Yves.Ledru@imag.fr 2013

JML syntax

- JML assertions (invariants, but also pre- and post-conditions, …) are expressed as
  - //@ JML text of the assertion
  - /*@ JML text of the assertion @*/
- They appear as java comments and are ignored by the java compiler.
- But the JML compiler (jmlc) recognizes them and processes them adequately.
- JML assertions are based on the Java syntax.

© Yves.Ledru@imag.fr 2013

Invariants for SetAsTree

```
public class SetAsTree{
  public Integer val;
  public SetAsTree ltree;
  public SetAsTree rtree;

  /*@ public invariant
    @((val != null) || (ltree == null && rtree == null));
    @*/
  /*@ public invariant
    @((ltree == null) ||
    @   (!ltree.emptySet() && ( ltree.max() < val.intValue() )));
    @*/
  /*@ public invariant
    @((rtree == null) ||
    @   (!rtree.emptySet() && ( rtree.min() > val.intValue() )));
    @*/
  //@ public invariant (* no cycle in the tree *);
```

Val may only be null if ltree and rtree are null (definition of the empty set)

Property 2: Elements in the left tree are less than val

Property 3: Elements in the right tree are greater than val

Non executable, natural language, assertion (for documentation purposes)

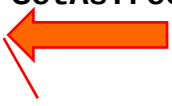
© Yves.Ledru@imag.fr 2013

Checking the properties

- When compiled with jmlc the assertions will be checked at run-time:
  - before and after the execution of any method of the class
  - after the execution of any constructor

Example of a failing test

```
@Test
public void  testSequence_1() {
  try{
    SetAsTree s5=new SetAsTree(5);
    SetAsTree s1=new SetAsTree(1);
    s5.setRtree(s1);
    } catch (…
  }
}
```



There was 1 failure:
1) testSequence_1(FailingTestSetAsTreeJUnit4)
junit.framework.AssertionFailedError:
by method SetAsTree.setRtree@post<File "SetAsTree.java", line 45, character 11> regarding specifications at
File "SetAsTree.java", line 9, character 26 when
'val' is 5
'rtree' is 1
'this' is 5 1

Invariants express a « contract »

- Each method must establish the invariant on exit
- It relies on the invariant on entry
- The invariant may be false inside the method.

- Just like the rules of the classroom:
  - Chairs and tables must be aligned when leaving the room, at the end of the lesson
  - It is guaranteed that they will be aligned when entering the room
  - Alignment may be broken during the lesson, for pedagogical purposes!

© Yves.Ledru@imag.fr 2013

A trade-off between quality and cost

- Invariants are written once…
- … and apply to all methods of the class

- As will be seen in exercise 2, it is easier to express an invariant that constraints the tree than to write operations which comply with the invariant.

© Yves.Ledru@imag.fr 2013

« Pure » functions (1)

- The assertions make use of methods of class SetAsTree(): emptySet(), max(), min()

```
/*@ public invariant
    @((ltree == null) ||
    @   (!ltree.emptySet() && ( ltree.max() < val.intValue() )));
    @*/
```

- Since the assertions are executed at run-time, care must be taken that they should not modify the behaviour of the class!
- Therefore, only « pure functions » may be used in the JML assertions.

© Yves.Ledru@imag.fr 2013

« Pure » functions (2)

- A « pure » function is a method that is not side-effecting, i.e. which does not modify the state of the object where it is executed.
- It must be declared as **`/*@ pure @*/`**
- Examples:

```
public /*@ pure @*/ boolean emptySet(){
  return val == null;
}
public /*@ pure @*/ int max(){
  if (rtree != null
      && rtree.getVal().intValue() > val.intValue())
        {return rtree.max();}
  else return val.intValue();
}
```

© Yves.Ledru@imag.fr 2013

Other restrictions in assertions

- Side-effecting java constructs, such as = , += , ++ may not be used inside JML assertions.

© Yves.Ledru@imag.fr 2013

Invariants and assertions

- Please note that the invariant is not checked inside an assertion!
- E.g. it is not checked when entering max() inside the invariant.
- (otherwise it would loop)
- But it is checked when entering max() inside the code of a method.

© Yves.Ledru@imag.fr 2013

The jmlc compiler

- Jmlc instruments the methods of the class by adding code on entry and exit of each method.
- It then compiles this instrumented code into java bytecode.
- The compiled code checks the assertions at run-time and raises a JML exception when they are violated.
- In this course, we always use the –O option (old semantics).

© Yves.Ledru@imag.fr 2013

Checking that methods are pure

- The jmlc compiler performs some static checks that a method declared as « pure » is actually pure.

© Yves.Ledru@imag.fr 2013

Development process

- First compile your java class with jmlc
- Then test it until you are confident that it is correct.
- Finally, recompile your class with javac. Invariants will no longer be checked at execution time, but the class will keep the same (validated) behaviour.

© Yves.Ledru@imag.fr 2013

Performance

- Adding JML assertions slows down the execution of the code and increase memory consumption.
- This is one of the reasons for recompiling the validated class in Java.
- It is sometimes tempting to write more « efficient » assertions, to speed up the evaluation of invariants.
- But efficiency may be at the cost of readibility, and complex assertions are error-prone!

© Yves.Ledru@imag.fr 2013

Null objects

- Numerous run-time errors are due to the use of null objects.
- JML is able to check that an object includes a null value and raises an exception.
- For data structures such as SetAsTree, this would be a problem because the fields of the class may become null.
- Therefore, the class is marked as `/*@ nullable_by_default @*/`
- Individual variables may also be marked as `/*@ nullable @*/` or `/*@ non_null @*/`

© Yves.Ledru@imag.fr 2013

Tools used in this course

© Yves.Ledru@imag.fr 2013

JML 5.6

- JML5.6 is available at the following address: http://sourceforge.net/projects/jmlspecs/files/jmlspecs/5.6_rc4/
- To install it under windows:
  - download file JML.5.6_rc4.tar.gz
  - Extract it under C:\JML
  - And add C:\JML\bin# à to your path
- A more complete documentation on JML is available at the following address: http://jmlspecs.org
- Or at http://www.eecs.ucf.edu/~leavens/JML/jmlrefman/

© Yves.Ledru@imag.fr 2013

JUnit4

- Is a test driver for Java
- A test case is a java method of a test class annotated by @test
- JUnit discovers the test methods of the class and executes them sequentially.
- If test execution does not raise an exception, the test succeeds.
- If test execution raises an exception, the test:
  - Is a failure, if the default was anticipated (e.g. detected by an assert)
  - Is an error, if the default was not expected (e.g. run-time exception).

© Yves.Ledru@imag.fr 2013

JUnit4 (2)

- Often all test cases start with the same code or end with the same code. In JUnit4 it is possible to annotate methods as
  - @Before
  - @After

  In order to execute these before/after every test.
- Static methods annotated as
  - @BeforeClass
  - @AfterClass

  Are executed once at the beginning or at the end of the tests.

© Yves.Ledru@imag.fr 2013