Université Joseph Fourier — UFR IM2AG
Master 2 Génie Informatique — année 2013-2014

JML - TD 1

Yves LEDRU
septembre 2013

1 Logiciels utilisés

1.1 Distribution JML5.6

La distribution de JML5.6 est installée sur les stations de travail Linux de l'UFR IM2AG. Elle est également disponible à l'URL suivant:
`http://sourceforge.net/projects/jmlspecs/files/jmlspecs/5.6_rc4/`

Pour l'installer sous Windows, il suffit de récupérer le fichier `JML.5.6_rc4.tar.gz`, de l'extraire sous `C:\JML` et de rajouter `C:\JML\bin` à votre path sous windows.

Une documentation plus complète sur JML est disponible sur le site
`http://jmlspecs.org` ou à l'URL suivant
`http://www.eecs.ucf.edu/~leavens/JML/jmlrefman/`

1.2 Autres outils

Les tests seront effectués avec JUnit, de préférence, avec la version 4 de JUnit, disponible sous `http://junit.org/` (mais également incluse dans la distribution de eclipse).

Si vous travaillez sous Windows, il est recommandé d'utiliser GNU make, disponible sous:
`http://gnuwin32.sourceforge.net/packages/make.htm`

Vérifiez également que `make` et `javac` sont dans votre path.

Si vous travaillez sous eclipse, vous devez configurer (setup) votre projet pour qu'il utilise JUnit4, et ajouter `jmlruntime.jar` dans votre build path (via le menu "Properties>Java Build Path>Libraries" de votre projet, et la commande "Add external JARs"

Les fichiers décrits dans ce TP sont disponibles sur le site Moodle du cours:
`http://imag-moodle.e.ujf-grenoble.fr/course/view.php?id=93`

2 Implantation d'un ensemble par un arbre

On considère la structure de données suivante, contrainte par un invariant JML (fichier `SetAsTree.java`).

```
/*@ nullable_by_default @*/
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 *);

    // Constructor
    public SetAsTree(){ // produces an empty set
        val = null;
        ltree = null;
        rtree = null;
    }
    public SetAsTree(int v){ // produces a singleton node
        val = new Integer(v);
        ltree = null;
        rtree = null;
    }
    public SetAsTree(int v, SetAsTree l, SetAsTree r){ // arbitrary node
        val = new Integer(v);
        ltree = l;
        rtree = r;
    }

    //getters and setters
    public Integer getVal() {
        return val;
    }
    public void setVal(Integer val) {
        this.val = val;
    }
    public SetAsTree getLtree() {
        return ltree;
    }
    public void setLtree(SetAsTree ltree) {
        this.ltree = ltree;
    }
    public SetAsTree getRtree() {
        return rtree;
    }
    public void setRtree(SetAsTree rtree) {
        this.rtree = rtree;
    }

    // Application specific methods
    public void insert(int v){
    }
    public void delete(int v){
    }

    // Pure functions used in the specification
    public /*@ pure @*/ boolean emptySet(){
         return val == null;
    }
    public /*@ pure @*/ int min(){
        if (ltree != null && ltree.getVal().intValue() < val.intValue()){return ltree.min();}
        else return val.intValue();
    }
```

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

        // Non side-effecting methods
        public /*@ non_null @*/  String toString(){
70          String s = "";
            if (ltree != null) {s=s+ltree.toString();};
            s = s+" "+val+" ";
            if (rtree != null) {s=s+rtree.toString();};
            return s;
75      }
        public void skip(){ } // useful to test the invariant.
    }
```

Cette structure de données code un ensemble d'entiers sous forme d'arbre. L'invariant garantit les propriétés suivantes:

1. Les éléments de l'arbre n'y apparaissent qu'une fois.
2. Une valeur est toujours plus grande que tous les éléments de l'arbre à gauche et plus petite que les éléments de l'arbre à droite.
3. Il n'y a pas d'éléments nuls inutiles.
4. Il n'existe qu'un seul chemin de la racine de l'arbre à un de ses éléments.
5. L'arbre ne comprend pas de circularités (spécifié informellement).

L'objectif de ce premier TD est de se familiariser avec JML et `JUnit` et d'implanter les opérations `insert` et `delete`. Cette implantation doit être compatible avec l'invariant.

- L'opération `insert` ajoute un entier, passé comme paramêtre, à l'arbre. Si l'élément est déjà dans l'arbre, l'arbre n'est pas modifié.
- L'opération `delete` enlève un entier, passé comme paramêtre, de l'arbre. Si l'élément n'est pas dans l'arbre, l'arbre n'est pas modifié.

3 Programmes de test

Pour tester votre implantation, vous disposez de plusieurs programmes de test. `MainSetAsTree.java` est un programme principal qui appelle les opérations `insert` et `delete` puis affiche le contenu de l'arbre. Ce même test (sans afficher l'état de l'arbre) est proposé sous forme de test JUnit dans le fichier `TestSetAsTreeJUnit4.java`.

Le fichier `FailingTestSetAsTreeJUnit4.java` comprend deux tests correspondant à des formes d'arbres interdites. Seul le premier test rapporte une erreur.

Le fichier `A1Test.java` propose une suite de test plus élaborée avec plusieurs centaines de tests.

4 Commandes utiles

Le fichier *Makefile* reprend la plupart des commandes nécessaires pour construire ou exécuter l'application et ses tests.

```
JMLC = jmlc -O -Q
RM = rm
JAVA = java
JAVAC = javac
JUNIT4 = "C:\Program Files (x86)\JUnit\junit4.10\junit-4.10.jar"
JMLRUNTIME = "C:\JML\bin\jmlruntime.jar"
JMLCLASSPATH = .;$(JMLRUNTIME)
JMLJUNITCLASSPATH = $(JMLCLASSPATH);$(JUNIT4)

run : MainSetAsTree.class
	$(JAVA) -cp $(JMLCLASSPATH) MainSetAsTree

SetAsTree.class : SetAsTree.java
	$(JMLC) SetAsTree.java

MainSetAsTree.class : SetAsTree.class MainSetAsTree.java
	$(JAVAC) MainSetAsTree.java

A1Test.class : SetAsTree.class A1Test.java
	$(JAVAC) -classpath $(JMLJUNITCLASSPATH)  A1Test.java

A1Test.run : A1Test.class
	 java -cp $(JMLJUNITCLASSPATH) A1Test

TestSetAsTreeJUnit4.class : SetAsTree.class TestSetAsTreeJUnit4.java
	$(JAVAC) -classpath $(JMLJUNITCLASSPATH)  TestSetAsTreeJUnit4.java

TestSetAsTreeJUnit4.run : TestSetAsTreeJUnit4.class
	 java -cp $(JMLJUNITCLASSPATH) TestSetAsTreeJUnit4

FailingTestSetAsTreeJUnit4.class : SetAsTree.class FailingTestSetAsTreeJUnit4.java
	$(JAVAC) -classpath $(JMLJUNITCLASSPATH)  FailingTestSetAsTreeJUnit4.java

FailingTestSetAsTreeJUnit4.run : FailingTestSetAsTreeJUnit4.class
	java -cp $(JMLJUNITCLASSPATH) FailingTestSetAsTreeJUnit4

eclipseBin :
	cp *.class ../bin

clean :
	$(RM) *.class
```

Pour construire l'application et la tester avec le programme `MainSetAsTree` tapez:

```
make run
```

Pour exécuter la suite de tests `TestSetAsTreeJUnit4.class`:

```
make TestSetAsTreeJUnit4.run
```

Pour exécuter la suite de tests `FailingTestSetAsTreeJUnit4.class`:

```
make FailingTestSetAsTreeJUnit4.run
```

5 Questions

Le but de cette première séance de travaux dirigés est de prendre en main les outils et de constater qu'un invariant JML permet de détecter des mauvaises instantiations de la structure de données.

1. Compilez et exécutez le programme `MainSetAsTree` avec `make run`.

2. Compilez et exécutez la suite de test `TestSetAsTreeJUnit4` avec
   `make TestSetAsTreeJUnit4.run`
   Vous constatez que tous les tests réussissent.

3. Compilez et exécutez la suite de test `FailingTestSetAsTreeJUnit4` avec
   `make FailingTestSetAsTreeJUnit4.run`
   Vous constatez qu'un des deux tests échoue. Le deuxième test devrait échouer, mais pour que JML "voie" l'erreur, il faut qu'il soit appelé depuis la racine de l'arbre.

4. Modifiez `FailingTestSetAsTreeJUnit4` en ajoutant `s5.skip();` au 2e test et constatez que l'erreur est détectée.

5. Créez un troisième test pour `FailingTestSetAsTreeJUnit4` qui corresponde à un arbre interdit par l'invariant et vérifiez qu'il échoue à cause d'une exception JML.

6. Recompilez `SetAsTree.java` avec `javac` uniquement et jouez les tests:

   ```
   javac SetAsTree.java
   make FailingTestSetAsTreeJUnit4.run
   ```

   Vous constatez que la spécification JML, qui est à l'intérieur de commentaires, n'est pas prise en compte par `javac` et que tous les tests réussissent.

7. Implantez l'opération `insert`.

   Vous prendrez garde à passer un argument de type `Integer` (et non `int`) à `setVal(Integer val)` car `jmlc` fait la distinction entre ces deux types.

   Testez votre implantation avec les fichiers mis à votre disposition. Complétez les fichiers de test si nécessaire.

8. Si il vous reste du temps: implantez l'opération `delete`.