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

JML - TD 4 - Projet

Yves LEDRU

Septembre 2013

Ce TD fait l'objet d'un mini-projet sous forme de devoir. Répondez aux trois questions ci-dessous et renvoyez un rapport ainsi que vos fichiers sources dans les délais indiqués pendant le cours!
Les fichiers décrits dans ce TP sont déposés sur le site Moodle du cours:
`http://imag-moodle.e.ujf-grenoble.fr/course/view.php?id=93`

Stockage de produits dangereux

On étudie le stockage de produits dangereux, potentiellement explosifs, dans un ensemble de bâtiments. Les informations relatives à ce problème sont enregistrées dans deux tableaux à deux colonnes:

- le tableau des incompatibilités (`incomp`) recense les paires de produits qui ne peuvent pas être stockés dans le même bâtiment. (On fait l'hypothèse que ces incompatibilités peuvent se résumer à des paires de produits incompatibles)
- le tableau des affectations des produits dans les bâtiments (`assign`) .

L'invariant exprime notamment que deux produits incompatibles ne peuvent être stockés dans le même bâtiment. La classe `explosives` comprend deux méthodes. La première permet de déclarer une incompatibilité entre deux produits et la seconde sert à affecter un produit à un bâtiment.

```
1   // Based on a B specification from Marie-Laure Potet.

    public class Explosives{
        public int nb_inc = 0;
5       public String [][] incomp = new String[50][2];
        public int nb_assign = 0;
        public String [][] assign = new String[30][2];

        /*@ public invariant // Prop 1
10        @ (0 <= nb_inc && nb_inc < 50);
          @*/
        /*@ public invariant // Prop 2
          @ (0 <= nb_assign && nb_assign < 30);
          @*/
15      /*@ public invariant // Prop 3
          @ (\forall int i; 0 <= i && i < nb_inc;
          @         incomp[i][0].startsWith("Prod") && incomp[i][1].startsWith("Prod"));
          @*/
        /*@ public invariant // Prop 4
20        @ (\forall int i; 0 <= i && i < nb_assign;
          @         assign[i][0].startsWith("Bat") && assign[i][1].startsWith("Prod"));
          @*/
        /*@ public invariant // Prop 5
          @ (\forall int i; 0 <= i && i < nb_inc; !(incomp[i][0]).equals(incomp[i][1]));
25        @*/
        /*@ public invariant // Prop 6
```

```
          @ (\forall int i; 0 <= i && i < nb_inc;
          @        (\exists int j; 0 <= j && j < nb_inc;
          @           (incomp[i][0]).equals(incomp[j][1])
30        @              && (incomp[j][0]).equals(incomp[i][1])));
          @*/
        /*@ public invariant // Prop 7
          @ (\forall int i; 0 <= i &&  i < nb_assign;
          @     (\forall int j; 0 <= j && j < nb_assign;
35        @        (i != j && (assign[i][0]).equals(assign [j][0])) ==>
          @        (\forall int k; 0 <= k && k < nb_inc;
          @           (!(assign[i][1]).equals(incomp[k][0]))
          @              || (!(assign[j][1]).equals(incomp[k][1])))));
          @*/
40

        public void add_incomp(String prod1, String prod2){
            incomp[nb_inc][0] = prod1;
            incomp[nb_inc][1] = prod2;
45          incomp[nb_inc+1][1] = prod1;
            incomp[nb_inc+1][0] = prod2;
            nb_inc = nb_inc+2;
         }
        public void add_assign(String bat, String prod){
50          assign[nb_assign][0] = bat;
            assign[nb_assign][1] = prod;
            nb_assign = nb_assign+1;
        }
        public void skip(){
55      }
    }
```

Question 1 - lecture et test d'invariant

Pour chacune des 7 propriétés de l'invariant,

- insérez un commentaire expliquant cette propriété
- écrivez dans un fichier de test pour JUnit (`TestExplosivesJUnit.java`) un test qui invalide cette propriété. Idéalement, le test ne fera appel qu'aux opérations de la classe. Si il est impossible d'invalider la propriété avec les seules opérations de la classe, vous pouvez exploiter le caractère "public" des variables de la classe, puis appeler l'opération `skip()`.

Groupez les tests qui ne font appel qu'aux méthodes de la classe dans un fichier JUnit, et laissez les autres tests dans un deuxième fichier.

Question 2 - calcul de préconditions

Pour chacune des deux premières méthodes de la classe, calculez sa précondition. Insérez cette précondition dans le fichier et testez-la avec le fichier que vous aurez réalisé à la question 1. Complétez le fichier par d'autres tests pour une validation plus complète de vos préconditions.

Question 3 - Recherche d'un bâtiment

Spécifiez et codez une méthode qui, pour un produit donné passé en paramêtre, renvoie le bâtiment où il est possible de stocker ce produit. (Notez que cet énoncé en français est imprécis; dans votre

réponse vous expliquerez comment vous avez rendu l'énoncé plus précis dans la spécification JML).

Quelques pistes:

- commencez par créer une méthode `public boolean compatible(String prod1, String prod2)` qui teste si deux produits sont déclarés comme compatibles.
- Il existe une solution (trop) simple qui donne une réponse correcte (mais inefficace) à la recherche. Vous pouvez mentionner cette solution dans votre rapport, mais évitez de vous limiter à cette solution.