Université Grenoble Alpes — UFR IM2AG
Master 2 Génie Informatique — année 2017-2018

JML - TD 4 - Projet

Yves LEDRU
Janvier 2018

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])
      @              && (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;
      @        (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])))));
      @*/

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

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 (`TestExplosivesJUnit4.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 (`TestExplosivesJUnit4.java`), et laissez les autres tests dans un deuxième fichier (`TestExplosivesJUnit4Public.java`).

  Contrairement aux tests écrits lors des séances précédentes, vous n'indiquerez pas que l'exception JmlAssertionError est attendue. Vous aurez ainsi la possibilité de vérifier que le message d'erreur fait effectivement référence à la propriété que vous tentez d'invalider. Par exemple, le test `testSequence_0` fait explicitement référence à la propriété `Prop7`. Pour cela, le fichier (`TestExplosivesJUnit4.java`) est un peu plus complexe que ceux qui ont été vus lors des séances précédentes: les exceptions levées par chaque test sont triées pour séparer les tests inconclusifs des tests qui échouent.

Question 2 - calcul de préconditions

Pour chacune des deux premières méthodes de la classe, calculez sa précondition (de sorte que l'exécution de l'opération en vérifiant la précondition mène dans un état qui satisfait l'invariant). Insérez cette précondition dans le fichier et testez-la avec le fichier que vous aurez réalisé à la question 1 (les tests doivent maintenant devenir inconclusifs). Complétez le fichier par d'autres tests pour une validation plus complète de vos préconditions.

Note: Quand un test est inconclusif, OpenJML affiche toujours la première précondition de la méthode qui a été appelée de façon inconclusive. Il s'agit d'une imprécision de OpenJML car la levée d'exception peut résulter d'une autre précondition de la même méthode.

Question 3 - ajout d'assertions

Complétez la classe `Explosives.java` en exprimant les trois propriétés suivantes, sous forme d'invariant ou de contrainte d'histoire. Ces contraintes seront exprimées en JML **sans ajouter de méthode pure à la classe**.

- Prop8: toutes les lignes du tableau des affectations (`assign`) sont différentes deux à deux.
- Prop9: un produit ne peut pas être stocké dans plus de trois bâtiments.
- Prop10: le nombre d'incompatibilités (`nb_inc`) ne peut jamais diminuer.

Il n'est pas demandé de modifier les méthodes de la classe pour qu'elles satisfassent ces contraintes.

Question 4 - recherche d'un bâtiment

Spécifiez en JML et codez une méthode

```
//@ ...
//@ ensures \result.startsWith("Bat");
public String findBat(String prod)
```

qui, pour un produit donné passé en paramêtre, renvoie le bâtiment où il est possible de stocker ce produit sans qu'il soit stocké avec des produits incompatibles. (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.

Rapport et fichiers sources

Vous remettrez pour la date indiquée en cours:

- Un rapport (3 à 5 pages) qui comprendra les réponses aux diverses questions, en commentant vos préconditions (question 2) et en documentant `findBat` et sa spécification.
- Le fichier source de `Explosives.java` complété avec les préconditions des opérations, les assertions de la question 3 et le code de `findBat`.
- Trois fichiers de test:
  - le fichier `TestExplosivesJUnit4.java` qui répond aux questions 1 et 2 (sans utiliser les variables publiques)
  - un fichier `TestExplosivesJUnit4Public.java` qui complète le fichier précédent pour les invariants qui ne peuvent être invalidés qu'en accédant directement aux variables publiques de la classe
  - (optionnel) un fichier `TestExplosivesFindBat.java` qui teste `findBat`

Quelques conseils

- N'attendez pas la dernière minute pour commencer le devoir!
- Cet exercice n'est pas un exercice de programmation en Java. Ne négligez donc pas la partie spécification en JML!
- Assurez vous que votre fichier `Explosives.java` est bien compilé en JML avant de faire des tests.
- Si vous trouvez des bugs dans JML 0.8.24, vous pouvez essayer d'utiliser la version 0.8.7 qui est probablement plus stable.