Outils associés à Java Modeling Language (JML)
Corrigé

Attention : Ce sujet doit se faire dans un terminal et pas sous eclipse. Il existe un greffon OpenJML pour eclipse mais il n'est pas compatible avec la version d'eclipse installée.

Exercice 1 : JML et outils associés

Dans cet exercice, nous nous intéressons au programme ExempleDate1 (listing 1).

1.1. Lire le texte de la classe ExempleDate1, la compiler avec `javac` et l'exécuter avec `java`.

Solution : Pas de problème de compilation. Le programme s'exécute.

```
d1 = 20/05/1989
d2 = 13/02/1993
d3 = 31/06/2001
```

Rien à dire ?
A priori, non. Ceci montre bien qu'un humain n'est pas très fiable...

Listing 1 – Programme manipulant les dates

```
// Que penser du programme suivant ?
public class ExempleDate1 {
      public static void main (String args []) {
            // Construire les dates
            Date d1 = new Date(20, 5, 1989);
            Date d2 = new Date(13, 2, 1993);
            Date d3 = new Date(31, 6, 2001);

            // Afficher les dates
            System.out.println("d1 = " + d1);
            System.out.println("d2 = " + d2);
            System.out.println("d3 = " + d3);
      }
}
```

1.2. Attention, cette partie doit se faire dans un terminal !

Commençons par définir la variable d'environnement OPENJML :

```
export OPENJML=/mnt/n7fs/ens/tp_cregut/openjml
```

Compiler maintenant l'application en utilisant OpenJML. Il suffit de taper :

```
java -jar ${OPENJML}/openjml.jar -rac Date.java ExempleDate1.java
```

L'option `-rac` permet d'instrumenter le code pour vérifier pendant l'exécution du programme les contrats (Runtime Assertion Check).

Solution : Normalement, les classes devraient compiler sans erreur. Elles ont été instrumentées : du code a été ajouté pour vérifier les contrats.

1.3. Exécuter le programme en tapant :

```
java -cp ${OPENJML}/jmlruntime.jar:. ExempleDate1
```

Commenter.

Solution : Le programme compile et s'exécute mais affiche des erreurs. Plus précisément, il signale une violation de précondition lors de la création de la troisième date. En effet, 31, 6 et 2001 ne correspond pas à une date valide car il n'y a que 30 jours en juin.

Ici, nous avons utilisé JML pour instrumenter le code, c'est-à-dire ajouter des vérifications dans le code. Ceci permet de détecter (pendant la phase de mise au point du programme) des erreurs dans le programme : le code qui s'exécute ne respecte pas les contrats définis.

Remarque : Normalement, il devrait y avoir un outil pour engendrer la documentation en ajoutant les contrats.

Il pourrait aussi y avoir un générateur de tests unitaires statistiques. Le principe est de tirer au hasard des données de test (ou les choisir dans un ensemble prédéfini), de vérifier si elles respectent les préconditions pour décider si les données de test sont valides et, enfin, utiliser les postconditions pour déterminer si le test est réussi ou échoue.

Objectifs de ce sujet :

- Comprendre les outils fournis par JML ;
- Comprendre l'intérêt d'exprimer les propriétés d'un programme ;
- Utiliser les contrats comme une aide à la mise au point.