Licence SIL -ACPI

Méthode Formelle – Java Modeling Language

Colette Johnen

johnen@labri.fr

www.labri.fr/~johnen

1

Méthodes formelles

- Les **méthodes formelles** sont des techniques permettant de raisonner rigoureusement, à l'aide de la logique sur des programmes afin de démontrer leur validité par rapport à leur spécification
- Ces méthodes permettent d'obtenir une très forte assurance de l'absence de bug
- elles sont coûteuses en ressources (humaines et matérielles) et actuellement réservées aux logiciels les plus critiques

2

Différentes phases du cycle de vie d'un logiciel

analyse : comprendre le problème, les besoins

conception : décomposer en sous-problèmes, trouver une architecture pour résoudre le problème

réalisation : développer les différents morceaux

intégration : faire marcher ensemble les différents morceaux

validation : s'assurer qu'on a bien répondu au problème

3

Spécification

- Chaque phase produit son lot de *spécifications* = document qui décrit tout ou partie de l'application, à un niveau d'abstraction donné, dans un formalisme qui peut être plus ou moins formel
- Une spécification est un ensemble explicite d'exigences que doit satisfaire le programme/système

4

Java Modeling Language (**JML**)

- **JML** est un langage de spécification formelle pour Java.
- Il utilise la logique de Hoare :

  Pré et Post condition aux méthodes
- Les spécifications sont ajoutées au code en Java (en commentaire)
- Il existe des logiciels (Esc/Java) vérifiant que le code Java réalise la spécification

5

PQ des Spécifications Formelles ?

Très peu utilisées en pratique.

Mais raisonner en termes de spécification est un + :

- identification rigoureuse des pré-conditions et réflexion sur leur mise en œuvre
- réflexion sur ce que fait la méthode : post-condition et invariant

du code mieux pensé, et mieux documenté =

code de meilleure qualité !

Aide à la réalisation des tests et validation du code

6

Validation versus Test d'un logiciel

La **validation** est une opération destinée à démontrer qu'un logiciel produit effectivement les résultats escomptés

. Un jeu de *tests* sert à mettre en évidence des défauts du logiciel via un ensemble de *tests*

Test : les données, les résultats escomptés

Test et validation s'appuie sur une spécification

7

Validation d'un code Java à l'aide de JML et Esc/Java

1. Spécifier le code (JML)
2. Ecrire le code (Java)
3. Exécuter Esc/Java : il vérifie que le code Java correspond à la spécification.
   - message d'erreurs le code ne réalise pas la spécification
   - Message d'alerte, impossible de vérifier que le code réalise la spécification

8

Défauts de la Validation à l'aide d'une spécification formelle

1. vérifie que le code réalise la spécification formelle :
   - Spécification partielle, car le langage de spécification est limité
   - Spécification incomplète
   - Spécification erronée
2. Outil de validation est lui-même bogué

9

Implémenter une pré-condition

indiquer la pré-condition dans la doc (IMPERATIF)

1. ne pas protéger la méthode (beurk !)
2. ne rien faire si la pré-condition est violée (beurk !)
3. lever une exception si la pré-condition est violée
4. Traiter les cas où la pré-condition est violée (complexifié le code).
5. protéger la méthode par une assertion

Dans les 4 premiers cas, la fonction est totale.
Dans le dernier cas, la fonction est partielle.

10

Exemple d'annotation JML (1)

```
public class CompteBancaireExemple
  {
    public static final int MAX = 1000;
    private int sol;
// Invariant en JML : une formule qui
// doit être valide
 //@ invariant sol >= 0 && sol <= MAX;
```

11

JML et visibilité

Souvent, la spécification d'une méthode Java publique utilise des attributs privés de la classe.

Par défaut, ce n'est pas autorise en JML :
la spécification d'une méthode publique est publique ; et on ne peut pas mentionner dans une spécification publique des attributs privés .
solution : `/*@ spec_public @*/`

12

Exemple d'annotation JML (1 corrigé)

```
public class CompteBancaireExemple
  {
    public static final int MAX = 1000;
    private /*@ spec_public @*/ int sol;
// Invariant en JML = une formule valide
    //@ invariant sol >= 0 && sol <= MAX;
```

13

Exemple d'annotation JML (2)

```
// post condition (mot clef : ensures)
//@ ensures sol == 0;
public CompteBancaireExemple() { ... }
```

14

Exemple d'annotation JML (3)

```
// pré-condition (mot clef : requires)
//@ requires mont > 0;
//@ requires mont <= MAX - sol;
// liste des variables modifiés
//@ modifies sol;
/* \old(sol) est la valeur de la variable
  sol avant l'exécution de la méthode */
//@ ensures sol == \old(sol) + mont;
 public void crediter(int mont)
           { ... }
```

15

Exemple d'annotation JML (4)

```
    //@ requires mont > 0;
    //@ ensures sol == \old(sol) - mont;
   //@ modifies sol
    public void debiter(int mont)
              { ... }
}
```

16

Exemple d'annotation JML (4 Corrigé)

```
    //@ requires sol >= mont;
    //@ requires mont > 0;
    //@ modifies sol ;
   //@ ensures sol == \old(sol) - mont;
    public void debiter(int mont)
              { ... }
```

17

Exemple d'annotation JML (5)

```
  /*
    \result est la valeur
     retournée  par la fonction
 */
   //@ ensures \result == sol;
   public int getSolde()
             { ... }
}
```

18

JML : Invariant de boucle

```
//@ requires b > 0;
//@ ensures \result == a+b;
 static int add (int a, int b) {
    int res = a; int cpt = b;
    //@ loop_invariant cpt >= 0;
    //@ loop_invariant cpt <= b;
    //@ loop_invariant a+b == res+cpt;
    while (cpt  > 0) {
        res = res + 1;
        cpt = cpt - 1;
    }
  return res;
  }
```

19

JML : Notation

\forall	\exists
==>	
<==>	<=!=>
==	!=
&&	||

Exemple :

```
(\forall int i;
          i < Max ==> a[i] < a[Max])
```

Consulter les documents de
\\info\net\Bibliotheque\MF-dev\TP_EscJava\doc

20

Remerciement

Cours de SVL (Spécification et Validation de Logiciel) http://www.lifl.fr/~nebut/ens/svl/

Mirabelle Nebut, MdC, à l'Université Lille 1.

Exercice et TP de Méthode formelle, http://www.labri.fr/perso/ly/enseignement/smv/

Olivier Ly, MdC, à l'IUT Bordeaux 1.

21