

APERÇU SUR LA
PROGRAMMATION AVEC ASSERTIONS
EN JAVA

Java

Ing. AWADA Adel

AWA . QWJ

CONTENU

1. Introduction
 - Définition
 - Syntaxe
2. Où ne pas utiliser les assertions
3. Où utiliser les assertions
 - Comme Invariant interne
 - Comme Précondition
 - Comme Post-condition
 - Comme Invariant de classe
4. Activer/Désactiver les assertions

LES ASSERTIONS : Définition

Une assertion est une déclaration en Java qui permet de tester les hypothèses sur le programme (une supposition).

Par exemple, si on écrit une méthode qui calcule la vitesse d'une particule, on peut supposer que la vitesse calculée est inférieure à la vitesse de la lumière.

Une assertion contient une expression booléenne qu'on croit être vraie lorsque l'assertion s'exécute. Si elle est fausse, le système va signaler une erreur.

LES ASSERTIONS : Syntaxe

La déclaration d'assertion comporte deux formes:

1. **assert Expression1 ;**

Expression1: expression booléenne. Si elle est évaluée à faux le système jette une **AssertionError**.

2. **assert Expression1 : Expression2;**

Expression1 : expression booléenne.

Expression2 : expression qui a une valeur. (Il ne peut pas s'agir d'une invocation d'une méthode qui renvoie void).

Utilisez cette dernière version pour fournir un message détaillé au constructeur de la classe **AssertionError** qui livre un détail sur l'erreur.



LES ASSERTIONS : Où ne pas utiliser les assertions

- Ne pas utiliser les assertions pour vérifier les arguments dans les méthodes publiques.
- Ne pas utiliser les assertions pour effectuer un travail dont votre application a besoin pour un fonctionnement correct.

Par exemple:

```
assert names.remove(null); // Incorrect
```

Le program marche si les assertions sont activées, sinon les éléments nuls ne seront pas supprimés!

```
boolean nullsRemoved = names.remove(null);
assert nullsRemoved; // Correct
```

LES ASSERTIONS : Où utiliser les assertions

Il est convenable d'utiliser des assertions dans les structures suivantes:

- Invariants internes
- Invariants de contrôle de flux du programme
- Préconditions, post-conditions et Invariants de classe.

Invariant signifie quelque chose qui devrait se tenir à des conditions, peu importe les changements ou celui qui l'utilise/transforme.

LES ASSERTIONS : comme invariant interne

Invariants internes

Exemple 1:

Avant les assertions on utilisait les commentaires:

```
if (i % 3 == 0) { ... }
else if (i % 3 == 1) { ... }
else { // On sait que (i % 3 == 2)
      ... }
```

Avec les assertions

```
if (i % 3 == 0) { ... }
else if (i % 3 == 1) { ... }
else { assert i % 3 == 2 : i;
      ... }
```

Notez, que l'assertion dans l'exemple ci-dessus peut échouer si $i < 0$!

LES ASSERTIONS : comme invariant interne

Invariants internes

Exemple 2:

```
switch(suit) {  
    case suit.CLUBS: ...; break;  
    case suit.DIAMONDS: ...; break;           4 sortes de cartes  
    case suit.HEARTS: ...; break;  
    case suit.SPADES: ...; }
```

Pour limiter les choix à 4, il faut ajouter l'assertion:

```
default: assert false : suit ;
```

Une autre alternative:

```
default: throw new AssertionError(suit);
```

LES ASSERTIONS : Comme Invariant de Contrôle de Flux

Invariants de contrôle de flux d'un programme

Le principe est: Placer une assertion à n'importe quel endroit que vous supposez ne sera pas atteint. L'assertion à utiliser est:

```
assert false;
```

Exemple:

```
void foo() {  
    for (...) {  
        if (...)  
            return;  
    }  
    // L'exécution ne doit jamais arriver ici!!!  
}
```

```
void foo() {  
    for (...) {  
        if (...)  
            return;  
    }  
    assert false;  
}
```

Si l'exécution atteint **assert false**; le programme jette une **AssertionError**

LES ASSERTIONS : Programmation par Contrat

Préconditions, Post-conditions, et Invariants de classe

Les assertions pourraient aider à soutenir un style informel de la programmation par contrat.

Préconditions

conditions qui doivent être vraies lors de l'invocation d'une méthode.

Post-conditions

conditions qui doivent être vraies après l'exécution d'une méthode avec succès.

Invariants de classe

conditions qui doivent être vraies pour chaque instance d'une classe.

LES ASSERTIONS : comme Précondition

Préconditions

conditions qui doivent être vraies lors de l'invocation d'une méthode.

Le principe est Ne pas utiliser des conditions pour vérifier les arguments d'une méthode déclarée **public**.

Dans les méthodes **public** on utilise plutôt les exceptions.

```
public void setRefreshRate(int rate) {  
    if (rate <= 0 || rate > MAX_REFRESH_RATE)  
        throw new IllegalArgumentException("Illegal rate: " + rate);  
    setRefreshInterval(1000/rate);    }
```

utilisation de l'exception **IllegalArgumentException** pour contrôler l'argument **rate**

LES ASSERTIONS : comme Précondition

Préconditions

conditions qui doivent être vraies lors de l'invocation d'une méthode.

Dans les méthodes **private** on peut utiliser les assertions pour contrôler les arguments d'une méthode.

Exemple 2: Le mois d'année doit être [1, 12]:

```
private int daysOfMonth(int month) {  
    assert month>=1 && month<=12;  
    switch(month) {  
        case 1: case 3: case 5: case 7: case 8: case 10: case 12: return 31;  
        case 4: case 6: case 9: case 11: return 30;  
        default: if(year%400 == 0 || (year%100 !=0 && year%4 == 0)) return 29;  
                  else return 28;  
    }  
}
```

LES ASSERTIONS : comme Précondition

Préconditions

conditions qui doivent être vraies lors de l'invocation d'une méthode.

Dans les méthodes **private** on peut utiliser les assertions pour contrôler les arguments d'une méthode.

Exemple 1: Pour effectuer la multiplication de deux matrices A[][] et B[][], il faut que le nombre de colonnes de A soit égal au nombre de lignes de B:

```
private int[][] multiplicationMatrices(int[][] a, int[][] b)
{
    assert a[0].length == b.length;

    // faire la multiplication

    return resultat;
}
```

LES ASSERTIONS : comme Post-condition

Post-conditions

conditions qui doivent être vraies après l'exécution d'une méthode avec succès.

On peut tester les post-conditions avec des assertions dans les méthodes déclarées **public** et **nonpublic**.

Exemple 1: Prenons l'exemple de recherche binaire d'un entier dans un tableau d'entiers. Le tableau doit être trié et le résultat doit être entre -1 et la longueur du tableau -1:

```
private int rechercheBinaire(int[] t, int x)
{
    assert isSorted(t);

    // faire la recherche

    assert position>=-1 && position<t.length : position;
    return position;
}
```

LES ASSERTIONS : comme Post-condition

Post-conditions

conditions qui doivent être vraies après l'exécution d'une méthode avec succès.

Exemple 2:

Supposons qu'on a une méthode :

```
public boolean isSorted(int[] t) { . . . }
```

qui vérifie si les éléments du tableau sont ordonnés.

Cette méthode peut être utilisée en post-condition comme suit:

```
public void sort(int[] t)
{
    . . . // sorting

    assert isSorted(t);

}
```

LES ASSERTIONS : comme invariant de classe

Invariants de classe

conditions qui doivent être vaires pour chaque instance d'une classe.

Exemple1 :

Supposons qu'on implémente une structure de données d'arbre équilibré. Un invariant de classe pourrait être que l'arbre est équilibré et correctement ordonné.

Une certaine méthode:

vérifie que l'arbre est équilibré.

```
private boolean balanced() { ... }
```

Pour généraliser cette condition sur toutes les instances de la classe Arbre on peut faire l'assertion:

```
assert balanced();
```

Cette assertion doit être placée avant le retour de chaque constructeur et méthode de la classe.

LES ASSERTIONS : comme invariant de classe

Invariants de classe

conditions qui doivent être vaires pour chaque instance d'une classe.

Exemple 2 :

Supposons qu'on implémente une classe Compte où le solde du compte ne doit jamais être négatif.

Dans chaque constructeur ou méthodes, et immédiatement avant le retour de la méthode, on place l'instruction:

```
assert solde >= 0 : solde;
```

LES ASSERTIONS : Activer/Désactiver les assertions

Par défaut, les assertions sont désactivées au moment de l'exécution. Deux options (switches) de ligne de commande permettent d'activer ou désactiver sélectivement les assertions.

Activer: **-enableassertions** ou **-ea**
Désactiver: **-dissassertions** ou **-da**

On spécifie la granularité avec les arguments qu'on fournit avec le switch:

Pas d'argument	: toutes les classes exceptant les classes système
NomPaquetage	: les classes du paquetage et ses sous paquetages
...	: les classes du répertoire courant.
NomClasse	: dans la classe nommée.

```
java -ea:com.wombat.fruitbat... BatTutor
```