Tests unitaires

Chantal Keller

19 novembre 2015

Plan

1 TD9

2 Tests unitaires

3 Ouverture

4 À vous

Problème

Possibilité (mais pas sûre) :

- dû au “garbage collector”
- change régulièrement entre les diverses API

“Garbage collector” :

- programme tournant en parallèle de l’interpréteur Java
- chargé de vider la mémoire des “variables dorénavant inutiles”

Plan

1 TD9

2 Tests unitaires

3 Ouverture

4 À vous

Le test jusqu'à présent

Pendant les TDs :

- on effectue une action sur la tablette et on regarde ce qui se passe

Inconvénients :

- nécessite une vérification manuelle : possibilité de se tromper
- nécessite une action : difficile de tester à chaque changement dans le code

Supprimer la vérification manuelle

Assertions :

- au lieu de vérifier la valeur de `resultat` *de visu*
- ajouter dans le code `assert(valeur_attendue == resultat)`

La bibliothèque `JUnit` :

- automatisation des tests
- on déclare une méthode de test par propriété à vérifier
- elle peut effectuer des calculs, appeler les méthodes du code que l'on cherche à tester
- à la fin, on utilise une assertion

Exemple d'utilisation de JUnit

Dans le programme à tester :

```
int max(int i, int j) {
    if (i >= j) {return i;} else {return j;}
}
```

Dans les tests unitaires :

```
public void testMax1() {
    assertEquals(42, max(42,17))
}
```

```
public void testMax2() {
    int quarantedeux = 17+25;
    assertEquals(quarantedeux, max(17,45-3))
}
```

↪ Règle : `public void testNomQueLOnVeut() { ... }`

Test unitaire

Test unitaire :

- vérifie le bon fonctionnement de chaque méthode individuellement (ou d'un petit groupe de méthodes)
- ne permet pas de tester le comportement global d'un grand programme

"Le bon fonctionnement" :

- *spécification* : relation entre les arguments et le résultat (et les variables globales) d'une méthode qui doit toujours être vérifiée
- définir des spécifications pour chaque méthode
- définir un test unitaire pour chaque spécification

Particularités d'une application graphique

Sorties :

- vérifier que l'affichage est correct
- nécessite l'accès aux éléments graphiques

Entrées :

- vérifier la réaction à des actions de l'utilisateur
- nécessite de simuler ces actions

La classe `ActivityInstrumentationTestCase2`

Pour tester une activité :

- donne accès à l'objet Java contenant l'activité
- à partir de là, donne accès aux objets de l'interface :
  - lire leur valeur, leur position, ...
  - simuler des actions dessus

Plan

1 TD9

2 Tests unitaires

3 Ouverture

4 À vous

Le test unitaire dans le développement

S'assurer continuellement que les spécifications restent vraies :

- définir des tests unitaires le plus tôt possible
- vérification de toutes les spécifications sur plusieurs exemples
- vérification régulière (par exemple, chaque nuit)

$\hookrightarrow$ coût de mise en place mais accélère énormément le développement

Autres aspects liés au test

En complément des tests unitaires :

- test de propriétés générales
- communication entre applications

À grande échelle :

- génération automatique des jeux de test
- critères de couverture : passer par tous les embranchements du code, toutes les instructions, ...

Au delà du test

Preuve formelle :

- garantit que le programme vérifie les spécifications **dans toutes les exécutions possibles**
- fige le programme
- beaucoup plus fastidieux (mais vers de plus en plus d'automatisation)

↪ Qui vérifie les spécifications ?

Plan

1 TD9

2 Tests unitaires

3 Ouverture

4 À vous

TD10

Mise en place de tests unitaires sur le TD8