Java inside

Lab 5 : **Switch on String et Tests paramétrés**

SORAN Altan - ESIPE INFO 2019

https://github.com/asoran/java-inside

JUnit - Parameterized Test

Quand on écris des tests, on veut parfois tester une méthode avec plusieurs valeurs ou bien testé un comportement avec plusieurs implémentations (méthodes). On se retrouve donc à dupliquer du code. C’est pour cela que les tests paramétrés ont été créés.

Une test annoté de @ParameterizedTest à besoin d’au moins 1 ArgumentsProvider (source des différentes valeurs du test). Ce fournisseur est responsable de la fourniture d'un Stream<?> d'arguments qui sera utilisé pour appeler la méthode.

Tests conditionnelles multiples

Parfois on veut faire plusieurs tests sur une même variables pour savoir si elle est égale à tel ou tel ou tel valeur, puis faire quelque chose en fonction.

Il y a plusieurs façon d'implémenter ceci, on peut par exemple naturellement imbriquer plein de if/else if …

```
if(variable == valeur1) {
    // …
} else if(variable == valeur2) {
    // …
}
...
```

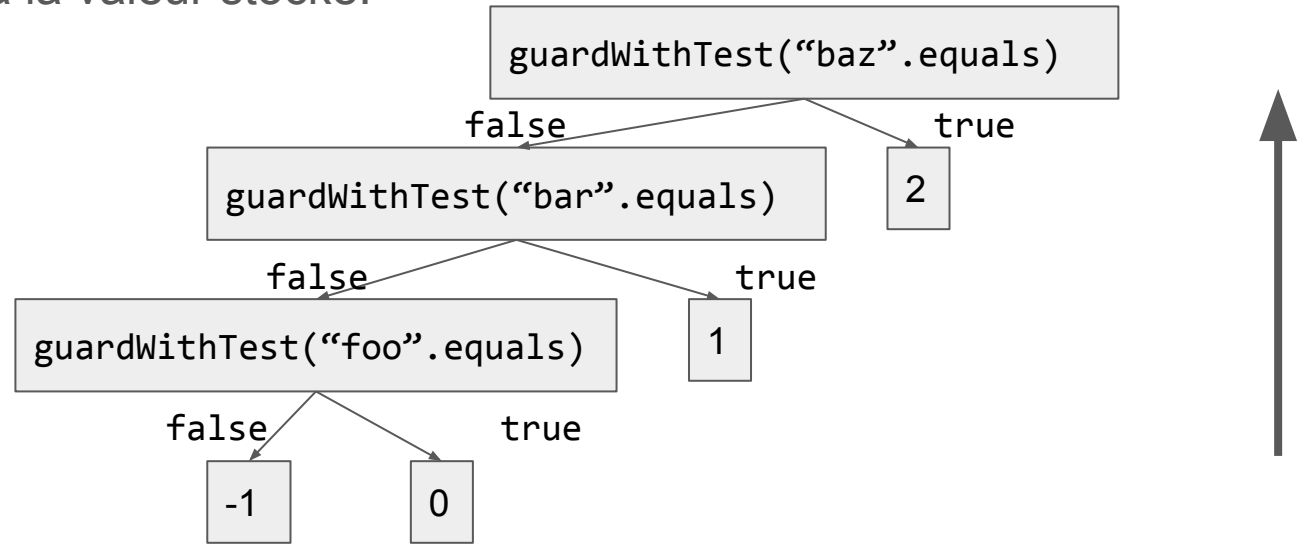
Tests conditionnelles multiples - Switch

Dans beaucoup de langage dont Java, on peut aussi écrire ces tests multiples avec l'instruction switch, qui est généralement plus efficace qu'un ensemble de if imbriqués.

```
switch(variable) {
    case valeur1:
        // ...
        break;
    case valeur2:
        break;
    default:
}
```

Tests conditionnelles multiples - guardWithTest

De façon un peu plus avancé, on peut utiliser les MethodHandler et essayer de reproduire le comportement d'un switch en combinant plein de guardWithTest. Le cas false appelera le prochain guardWithTest, et le cas true, renverra la valeur stocké.



Tests conditionnelles multiples - Inline caching

Dans la slide précédente, l'arbre était petit car le nombre de choix était petit.

La taille de l’arbre et le temps que va prendre la fonction à s'exécuter va augmenter proportionnellement. Et même plus, l'ordre dans lequel on fait les tests influe sur les performances.
Au lieu de choisir un ordre arbitraire il existe une technique nommée inlining cache qui permet de créer l'arbre au fur et à mesure que le code est appelé avec des valeurs différentes.

On peut donc sauvegarder notre MethodHandler dans un MutableCallSite, et construire l’arbre à chaque appel !

Tests conditionnelles multiples - Inline caching - note

La façon dont on génère l'arbre dont les 2 slides précédentes c’est qu’on ajoute à chaque fois une branche “au dessus”, c’est à dire que l’on utilise l’arbre d’avant comme embranchement pour la cas échouant du guardWithTest.

Sachant que statistiquement la première chaîne de caractère que l'on demande est celle qui est le plus demandée, on peut augmenter encore plus la performance en ajoutant des branches “en dessous”, donc changer la valeur du cas échouant en un nouveau guardWithTest !

Tests conditionnelles multiples - Benchmark

J’ai codé un benchmark comparant 3 différents façon de faire des tests conditionnelles multiples sur des String

`stringSwitch_1` : un Switch/Case classique

`stringSwitch_2` : L’arbre de guardWithTest, sans inline chaching

`stringSwitch_3` : L’arbre de guardWithTest, avec inline chaching mais sans l’optimisation de la slide précédente

Voici les résultats en temps exprimés en secondes par opérations:

```
StringSwitchBenchMark.stringSwitch_1  avgt     0,007          s/op
StringSwitchBenchMark.stringSwitch_2  avgt     9,092          s/op
StringSwitchBenchMark.stringSwitch_3  avgt   112,421          s/op
```