INF431 - TD2

Quelques erreurs fréquentes et conseils . . .

Marc Mezzarobba Yann Ponty

17 février 2010

Stockages locaux de propriétés dans des structures récursives

But : Éviter calculs répétitifs et coûteux.

$\Rightarrow$ Mémoriser et actualiser au lieu de recalculer à chaque fois.

Exemple typique : `size()` dans une liste chaînée `LinkedList`.

```
1 LinkedList<Integer> l = new LinkedList<Integer>();
2 for(int i=0;i<N;i++) {
3   l.add(i);
4   System.out.println(l.size());
5 }
```

Recalcul de `size()` : $\Rightarrow \mathcal{O}(N^2)$ opérations.

Stockage dans un champs `_size` (Mémoisation) :

\+ Mise à jour (add $\rightarrow$ `_size++`, `remove` $\rightarrow$ `_size--`)

$\Rightarrow \mathcal{O}(N)$ opérations.

Stockages locaux de propriétés dans des structures récursives

But : Éviter calculs répétitifs et coûteux.
$\Rightarrow$ Mémoriser et actualiser au lieu de recalculer à chaque fois.

Exemple typique : `size()` dans une liste chaînée `LinkedList`.

```
1 LinkedList<Integer> l = new LinkedList<Integer>();
2 for(int i=0;i<N;i++) {
3   l.add(i);
4   System.out.println(l.size());
5 }
```

Recalcul de `size()` : $\Rightarrow \mathcal{O}(N^2)$ opérations.
Stockage dans un champs `_size` (Mémoisation) :
+ Mise à jour (`add` $\rightarrow$ `_size++`, `remove` $\rightarrow$ `_size--`)
$\Rightarrow \mathcal{O}(N)$ opérations.

Stockages locaux de propriétés dans des structures récursives

But : Éviter calculs répétitifs et coûteux.
$\Rightarrow$ Mémoriser et actualiser au lieu de recalculer à chaque fois.

Exemple typique : `size()` dans une liste chaînée `LinkedList`.

Relève de l'optimisation algorithmique
Point important, mais moins que la correction du programme ...
$\Rightarrow$ Prioriser le respect de la spécification.

Attention à la cohérence interne des objets !

Exemple : Liste chaînée $L_1 \to L_2 \to \varnothing$
Si L_2 est exposée, alors l'ajout direct d'un élément à L_2 n'est pas nécessairement reflété à L_1, qui a alors une idée fausse de sa taille.

Initialiser les structures de données

Initialisez systématiquement vos structures de données.

Sauf mention explicite, recopiez les instances passées comme argument des constructeurs.
Sinon, l'objet crée partage une référence avec quelqu'un d'autre qui peut en modifier l'état interne.

De même, le code ci-dessous est à proscrire.

```
1 if (_v==null)
2 {_v = new ...;}
```

(Quid de la fois où on oubliera de tester ? ? ?)

Itérateurs

Les itérateurs permettent de parcourir les structures de données de façon efficace.

Spécifient souvent un (`Iterator`), plusieurs (`ListIterator`) ou une absence d'ordre.
$\Rightarrow$ Bien lire la `JAVADOC`.

Leur utilisation permet l'exploitation des points forts de la structure de données (Opérations en $\mathcal{O}(1)$) sans en connaître les mécanismes internes.

Minimiser les tests dans les `while`

Utilisez des tests courts pour les boucles conditionnels, afin d'éviter d'évaluer des expression illégales

```
1 int i=0;
2 while((i<t.length)&(t[i]=='.')) {i++;}
```

$\Rightarrow$ `IndexArrayOutOfBoundException` (Boum !)

```
1 int i=0; boolean over=false;
2 while(!over) {
3   if (i<t.length)
4   {over = (t[i]!='.');}
5   else
6   {over=true;}
7   i++;}
```

En vrac . . .

- Encore des paramètres de types manquants . . .
  Écoutez Eclipse râler ! !
- Math.min, Math.max
- Tester avant de diviser
  Pour toute opération de type moyenne (ou impliquant une division), on doit vérifier que le dénominateur est non-nul (Liste vide).
- Factoriser le code
  Deux parcours (non-imbriquées) dans une fonction :
  $\Rightarrow$ Deux fonctions ?
  Réutilisez l'existant au maximum (Essayez de reformuler en récursif).

En vrac . . .

- Encore des paramètres de types manquants . . .
  Écoutez Eclipse râler ! !
- Math.min, Math.max
- Tester avant de diviser
  Pour toute opération de type moyenne (ou impliquant une division), on doit vérifier que le dénominateur est non-nul (Liste vide).
- Factoriser le code
  Deux parcours (non-imbriquées) dans une fonction :
  $\Rightarrow$ Deux fonctions ?
  Réutilisez l'existant au maximum (Essayez de reformuler en récursif).

En vrac . . .

- Encore des paramètres de types manquants . . .
  Écoutez Eclipse râler ! !
- Math.min, Math.max
- Tester avant de diviser
  Pour toute opération de type moyenne (ou impliquant une division), on doit vérifier que le dénominateur est non-nul (Liste vide).
- Factoriser le code
  Deux parcours (non-imbriquées) dans une fonction :
  $\Rightarrow$ Deux fonctions ?
  Réutilisez l'existant au maximum (Essayez de reformuler en récursif).

En vrac . . .

- Encore des paramètres de types manquants . . .
  Écoutez Eclipse râler ! !
- Math.min, Math.max
- Tester avant de diviser
  Pour toute opération de type moyenne (ou impliquant une division), on doit vérifier que le dénominateur est non-nul (Liste vide).
- Factoriser le code
  Deux parcours (non-imbriquées) dans une fonction :
  $\Rightarrow$ Deux fonctions ?
  Réutilisez l'existant au maximum (Essayez de reformuler en récursif).