INF431 - TD1

Quelques erreurs fréquentes et des conseils . . .

Marc Mezzarobba Yann Ponty

2 février 2010

Égalité de chaînes

En Java, l'égalité de contenu (ou équivalence sémantique) de deux objets se teste grâce à la méthode `Object.equals(Object o)`, et **non pas par l'opérateur** ==, qui teste l'identité des références (pointeurs).

En particulier, on utilisera cette méthode `Object.equals(Object o)` pour tester l'égalité de chaînes (`String`) au cours d'une recherche (cf TD1 Exos 4 et 7).

```
1 String a = "Joe";
2 String b = "Joe";
3 if (a.equals(b))
4 { System.out.println("We may go here..."); }
5 if (a == b)
6 { System.out.println(" but definitely NOT there!"); }
```

Toujours paramétrer les types génériques

L'intérêt des conteneurs génériques paramétrés (`Collection<ObjetA>`), introduits dans `Java 1.5` est de permettre une vérification du typage.

Avant son apparition dans `Java`, il fallait faire

```
1 Collection col = new Vector();
2 v.add(new ObjectA());
3 ...
4 for (Object o: col){
5   ObjectA x = (ObjectA) o;
6   ... // Do something with x
7 }
```

A la ligne 5, on *suppose* (dangereusement) que les objets de `v` sont uniquement des instances de `ObjectA`, mais rien ne le garantit ...

Conclusion : Pour des raisons de compatibilité ascendante, la présence du type-paramètre est parfois optionnelle dans les constructeurs. Une telle pratique est cependant prohibée dans le cadre d'INF431.

~~`Collection<ObjectA> col = new Vector();`~~

Modificateurs d'accès

Il est possible de définir la visibilité de nombreux éléments du langage `Java` (Méthodes, classes, champs, constructeurs, ...) grâce aux modificateurs suivants :

- `public` : Accès de n'importe quelle classe, n'importe où.
- `protected` : Accès des classes du package, **ou** des classes filles.
- $\varnothing$ (*default*) : Accès des classes du package courant uniquement.
- `private` : Accès au sein de la classe uniquement.

En règle générale, les constructeurs seront `public`, les champs seront `private` et accédés de l'extérieur de la classe via des *getters/setters*. On préservera ainsi la cohérence interne de l'objet.

Exemple

```
1 Kid A = new SmartKid();
2 A.setFavBand("Radiohead");
3 System.out.println(A.getFavBand());
4 A.setFavBand("Britney Spears"); // Throws Error...
```

Arbres et schémas récursifs

Lors du parcours d'un arbre à la recherche d'un élément satisfaisant une propriété (Ex : `name` dans la fonction `search` du TD 1), on cherche à éviter toute comparaison redondante, tout en restant complet.

Une approche correcte consiste à tester au niveau du noeud courant, puis de rappeler sur les fils.

Exemple

```
1  public AverageJoe findSuperHero(){
2    if (this.glowsInTheDark() || this.wearsFlashyTights())
3    { return this; }
4    else{
5      for(AverageJoe child: this.getChildren()){
6        AverageJoe savior = child.findSuperHero()
7        if (savior != null)
8        {return savior;}
9      }
10   }
11   return null;
12 }
```

Conseils en vrac

Pour le TD2, nous conseillons de :

- Bien lire l'énoncé : Particulièrement pour les types des arguments et de retour des méthodes et constructeurs.
- Modifier au minimum les fichiers fournis : Si ça ne marche pas, l'erreur vient (presque) sûrement de votre code . . . adaptez vous !
- Écouter Eclipse (mais pas trop . . .) : Ne laissez pas de `warning` dans votre code, même si ça compile et marche, mais soyez critiques par rapport aux corrections proposées.
- N'oubliez pas de re-uploader vos fichiers en cas d'erreur : Des erreurs à la question i deviennent parfois évidentes à la question $i + 1$, et vous risquez d'oublier de re-soumettre la version corrigée.
- Vous amuser et de ne pas hésiter à aller au delà des exigences du TD (Tout en respectant les spécifications ...)