Programmation Par Les Objets En Java
Copie D'objets Ou Clonage(TD6)

Najib Tounsi

(Lien permanent: http://www.mescours.ma/Java/TD/tdJava6.html (.pdf))

L'objectif de ce TD est d'introduire les notions de copie, copie superficielle et copie en profondeurs d'objets. Il est proposé comment redéfinir la méthode `clone()` prédéfinie en Java pour réaliser les différents choix de copie.

Sommaire

1. Le Clonage

Le clonage en général est la création d'un nouvel objet (une instance d'une classe) à partir d'une instance déjà existante. En java le clonage sert à faire la copie d'un objet dans un autre. En effet, si on se contente d'écrire $x = y$; on obtient la copie des références (y sur x), et non la duplication de l'objet y dans x.

Pour qu'un objet soit "copiable" (*clonable*), sa classe doit implanter la méthode `clone()` de l'interface `Clonable`. En fait, la méthode `clone()` est définie pour la classe `Object` et s'applique donc à tous les objets Java. En pratique, les sous-classes qui le désirent doivent implanter l'interface `Clonable` et redéfinir la méthode `clone()`. Cette redéfinition peut se limiter à faire appel à `super.clone();` i.e. la méthode mère héritée. Celle-ci peut générer l'exception `CloneNotSupportedException`.

2. Simple Clonage

Exemple 1 : Création d'un objet `Cellule` suivi de son clonage

Soit la classe `Cellule`, constituée d'un entier et d'un tableau :

```
1 class Cellule extends Object implements Cloneable {
2  // Doit implementer la méthode clone() de l'interface Cl
3  //   Methode qui fait le clonage.
4
5     // Donnees
6     int i = 0;
7     int[] t = {1, 2};
8
```

```
    // Méthodes
    public Object clone(){
        try {
            return super.clone();
        }
        catch (CloneNotSupportedException e){
            throw new InternalError();
        }
    }

    public void afficher(){
        System.out.println(i +" "+ t[0]+" "+t[1]);
    }
}
```

Tester cette classe avec:

```
class TestClone {
     public static void main(String args[]){
        Cellule x = new Cellule();        // x Objet Cell
        x.afficher();
        Cellule y = (Cellule) x.clone();  // y clone de x
        y.afficher();
     }
}
```

et constater le résultat:

```
i=0 t=(1, 2)  // valeurs de la cellule x
i=0 t=(1, 2)  // valeurs de la cellule y
```

Remarquer qu'on aurait pu faire une affectation d'objets dans `main()`. Au lieu de faire

```
    Cellule y = (Cellule) x.clone();   // initialisation par
```

On fait

```
    Cellule y = new Cellule();
    y = (Cellule) x.clone();     // vraie affectation d'un
```

3. Clonage Superficiel.

En réalité, le clonage est *superficiel*. L'entier *x.i* a été copié dans *y.i*, mais pour le tableau *t*, c'est la référence qui est copiée (non les éléments du tableau.) Pour le constater, rajouter à la classe `Cellule` une méthode qui change les éléments d'un cellule.

Exemple 2 : Rajouter à la classe `Cellule` la méthode

```
public void changeMe(){
```

```
2       i = 10;
3       t[0] = 11;
4       t[1] = 12;
5   }
```

et vérifier qu'en rajoutant dans `main()` les instructions:

```
x.changeMe();
x.afficher();
y.afficher();
```

on obtient:

```
i=10 t=(11, 12)         // x change
i=0 t=(11, 12) // y aussi pour la partie tableau (même obj
```

4. Clonage En Profondeur

Pour copier tout l'objet `Cellule`, y compris le tableau, il faut cloner ce dernier (le copier aussi).

Exemple 3 : Cloner aussi le tableau.

On modifie la méthode `clone()` de la classe `Cellule`, pour cloner aussi le tableau (un tableau `x` supporte aussi la méthode clone `x.clone()` en adaptant le type).

```
1   public Object clone(){
2         try {
3             Cellule tmp = (Cellule) super.clone();
4             tmp.t = (int []) this.t.clone();  //clonage de
5             return tmp;
6          }
7          catch (CloneNotSupportedException e)
8             {throw new InternalError(); }
9   }
```

On crée par clonage une cellule temporaire `tmp` et on modifie son tableau `tmp.t` par recopie (`this->t.clone()`)

Exercice: le vérifier.

Exemple 3.1 : Variation sur le même thème, clonage à la C++

Cette fois-ci, on va copier un à un les éléments d'un objet `Cellule` sans faire appel à `super.clone()`. On modifie la méthode `clone()` de la classe `Cellule` comme suit.

```
1   public Object clone(){ // à la c++, sans clone super.clo
2           Cellule tmp = new Cellule();
3           tmp.i=i;
```

```
4           tmp.t = (int []) this.t.clone();      //clonage d
5           return tmp;
6   }
```

Remarque: on peut tout aussi aussi faire `for` pour copier les éléments de `t` (exercice: le faire)

Exemple 3.2 : Constructeur de `Cellule` par copie. Façon C++.

On peut créer un constructeur par copie pour la classe `Cellule`, comme en C++.

```
public Cellule (Cellule x){
     this.i = x.i;
     for (int i=0; i<2; i++)
         this.t[i]=x.t[i];
 }
```

NB. Définir d'abord un constructeur défaut sans paramètres `public Cellule(){...}`.

On peut alors instancier un objet *y* par copie

```
    Cellule x = new Cellule();
    ...
    Cellule y = new Cellule(x);  // initialisation par copi
```

Exercices:

- Le vérifier.
- Quelle est la différence par rapport à la méthode clone? (*réponse*: un constructeur sert à initialiser un objet nouveau, alors qu'un clonage en Java peut servir aussi à faire des affectations d'objets).