Programmation Par Les Objets En Java
Interfaces Et Classes Abstraites (TD5)

Najib Tounsi

(Lien permanent: http://www.mescours.ma/Java/TD/tdJava5.html (.pdf))

Dans ce TD seront vues les interfaces et leur réalisation ainsi les classes abstraites. Sera discuté aussi la nuance entre ces deux concepts;

Sommaire

1. Les Interfaces

Une interface est une collection nommée de déclarations de méthodes (sans les implémentations). Une interface peut aussi déclarer des constantes.

1.1. Definition D'une Interface Simple

Exemple : Une interface simple avec une constante et une fonction non `static`.

```
interface Loisir {
    public int distance = 21;
    public void courirOuMarcher();
}
```

- Noter le mot **`interface`** au lieu de `class`.
- Une interface ne peut pas avoir de méthodes déclarées `static` (contrainte levée en Java8).
- Ne peut pas avoir de méthodes implémentées non plus (à la différence des classes abstraites (contrainte levée en Java8, voir plus loin)
- `distance` considérée comme *final variable* et ne peut être modifiée.

1.2. Réalisation D'une Interface

La réalisation (ou "implémentation") d'une interface, c'est définir des méthodes déclarées. Elle doit être faite dans une classe séparée. Voici un exemple de classe qui implémente l'interface ci-dessus :

```
class Coureur implements Loisir {

  //Implémentation de la méthode courirOuMarcher
  public void courirOuMarcher(){
        System.out.println("Je cours "+distance+" Km.");
    }
};
```

Noter le mot clé ***implements***.

Exemple d'utilisation :

```
class Test{
    static public void main(String args[]){
        Coureur c = new Coureur() ;
        c.courirOuMarcher();
    }
}
```

Compiler l'interface et ensuite ces deux classes. Exécuter le teste.

Résultat :

```
Je cours 21 Km.
```

Exercices :

1. Peut-on déclarer la variable de type `Loisir` et l'instancier avec avec un objet `Coureur` ? C'est à dire :

   (a) `Loisir  c = new Coureur();` au lieu de
   (b) `Coureur c = new Coureur();`

   Vérifier que c'est possible. Vérifier aussi que :

   (c) `Loisir l = new Loisir();  // :-(`

   par contre, n'a pas de sens. On n'*instancie pas une interface*, car les méthodes n'y sont pas définies. Ne pas confondre avec la ligne (a) où l'instance est un objet `Coureur`.

2. Rajouter à la classe `Coureur` la méthode :

   ```
   public  void courirMoins() {
        System.out.println("Je cours "+(distance/2)+" Km
   }
   ```

   et dans *main*, rajouter l'appel :

   ```
   c.courirMoins();
   ```

   Vérifier cet appel dans les deux cas (a) et (b) ci-dessus.

Réponse: Erreur de compilation dans le cas où `c` est de type `Loisir`. Normal, puisque la méthode `courirMoins` n'est pas déclarée dans l'interface.

3. La constante distance est `final`, et ne peut être modifiée ni dans la classe `Coureur` ni ailleurs. Le vérifier en faisant :

```
distance /= 2;
```

dans la méthode `CourirMoins`.

Ailleurs, l'accès à la constante, en lecture donc, peut se faire soit par la notation `c.distance` soit par `Loisir.distance` tout simplement.

4. Créer une autre classe `Marcheur`, qui implémente la même interface `Loisir`. L'implémentation affichera un simple message "`Moi, je marche...`".

5. Faire un programme qui crée un tableau `mesLoisirs` d'objets `Loisir`, et qui l'instancie indifféremment avec des objets de type `Coureur` ou `Marcheur`.

```
Loisir mesLoisirs[] = { new Marcheur(), new Coureur() } ;
```

Vérifier que les appels à la méthode `coureurOuMarcher()` sur les objets du tableau donne le message correspondant à chaque objet. (Se contenter d'un tableau deux éléments.)
Cela rappelle le polymorphisme d'héritage. La bonne méthode exécutée en fonction de l'instance actuelle.

6. Concevoir deux interfaces *A* et *B* et une classe *C* qui réalise (implémente) ces deux interfaces.

syntaxe: `class C implements A, B { corps de la classe..}`

7. Que se passe t-il si les deux interfaces *A* et *B* déclarent une même méthode *f*()? Une même constante *x*?

Indication: pour le savoir, créer une classe *Test* qui utilise *f*() et *x*.

Pourquoi ne peut-on utiliser *x*? (réponse: La définition de *x* est donnée *dans* les interfaces *A* et *B* et non dans les classes qui les réalisent. Il faudrait donc préfixer *x* par le nom de l'interface (`A.x` ou `B.x`)).

Par contre, pour la méthode *f*() il n'y pas de problème. Elle est définie *dans* la classe qui implémente. Si on déclare `C c = new C();` il n'y a pas de problème à écrire `c.f();`. C'est un appel de méthode normal. Même si on déclare **`A c`** `= new C();`, `c.f();` a toujours le même sens.

8. Concevoir une interface *A* avec une méthode *f*() et une classe *C* qui la réalise.
Tester si on peut rajouter une méthode *g*() dans l'interface *A*.
Vérifier qu'on doit *aussi* rajouter la définition de cette méthode dans *C* et la recompiler.

Ceci est contraignant, si on a plusieurs classes qui réalisent l'interface *A*. A partir de la version 8 de Java, on peut rajouter une méthode à une interface sans contrainte. Il suffit de la déclarer, par exemple, dans *A* : `default void g() {};` C'est une définition par défaut donnée dans l'interface. Les classes qui réalisent l'interface pourront alors la redéfinir si elles le souhaitent. Bénéfice : on peut ajouter une fonctionnalité supplémentaire à un certains nombre de classes sans toucher à leur code. On ajoute simplement une méthode par défaut dans l'interface qu'elles implémentent.

2. Les Classes Abstraites

Une classe abstraite est une classe qui peut contenir des méthodes sans corps, dites méthodes abstraites. L'implantation est laissée (déléguée) aux futures sous classes de la classe abstraite.

Une classe abstraite ne contient pas forcément des méthodes abstraites. Le fait qu'une classe soit abstraite, implique qu'on ne peut pas en créer des instances. Il faut alors en dériver des sous classes pour pouvoir instancier des objets et leur appliquer des méthodes.

Mais une classe qui contient une méthode abstraite doit être déclarée abstraite. De même, une sous-classe qui ne fournit pas l'implantation d'une méthode abstraite héritée (déclarée dans une classe mère), doit être déclarée abstraite à son tour.

Exemple : Illustration de ces mécanisme.

```
1  abstract class Generale {
2
3       public int x=2;              // x variable d'instanc
4
5       abstract public void qui();  // méthode abstraite à
6
7       public void moi(){
8           System.out.println("Methode générale");
9       }
10 }
```

NB. Une méthode sans corps doit toujours être déclarée abstraite. Par ailleurs, une méthode abstraite ne peut être déclarée `static` (pourquoi?)

Implantation :

```
1  class Speciale1 extends Generale{
2
3      public  void qui() {
4          // Implementation de qui()
5          System.out.println("C'est la sous-classe Speciale
6      }
```

```
7 }
```

Noter **`extends`** (*vs.* `implements`).

Test :

```
1  class Test{
2      static public void main(String args[]){
3
4          Speciale1 s = new Speciale1();
5          s.moi();
6          s.qui();
7          s.x++;
8          System.out.println(s.x);
9      }
10 }
```

NB. Compiler d'abord la classe `Generale`.

Résultat obtenue :

```
$ java Test
Methode générale                // méthode moi héritée exéc
C'est la sous-classe Speciale1  // méthode qui propre exécu
3                               // x incrémentée
```

Exercices:

1. Ajouter dans la classe `Speciale1` une redéfinition de la méthode `moi` déjà définie dans la classe `Generale` (afficher un message approprié dans cette redéfinition). Refaire le teste pour vérifier que ce nouveau message s'affiche la place du message "`Methode générale`" précédent.
2. Faire une hiérarchie de plusieurs classes abstraites. Vérifier les règles énoncées précédemment (§ classes abstraites), à savoir l'implémentation d'une méthode abstraite peut être différée d'une sous classe à une autre, par exemple.

3. Classe Abstraite Vs Interface

- Une classe abstraite permet de déclarer des membres non publics. Dans une interface, toutes les méthodes doivent être publiques.
- Une classe peut implanter plusieurs interfaces mais ne peut avoir qu'une seule superclasse.
- Une interface n'appartient pas à la hiérarchie des classes issues de `Object`. Des classes sans aucun rapport entre elles peuvent implanter la même interface. Alors que les classes qui dérivent d'une même classe abstraite appartiennent à une même famille.
- On peut rajouter de nouvelles méthodes à une classe abstraites sans conséquences

sur le reste. Si on veut rajouter un profile de méthode à une interface, il faut la définir dans toutes les classes qui réalisent l'interface. A moins d'en fournir une définition (par défaut) dans l'interface elle-même (nouveau depuis Java8).

- Une interface peut faire *extends* de plusieurs autres interfaces (Le vérifier sur un exemple `interface C extends A , B {}`).

En réalité, à part les différences de forme, une interface est une *spécification* (abstraite) appelée à être implémentée (concrétisée) de *plusieurs façons* par plusieurs classes différentes, chacune à sa manière. Le fait, est que ces classes sont quelconques et n'ont aucune relations entre elles. C'est une vision génie logiciel plus générale.

Dans le cas des classes abstraites, les classes qui implémentent l'abstraction *sont* des sous-classes. C'est plus une vision classification et développement progressif et structuré.

> **Extrait de la documentation Sun** *Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.*
> (http://docs.oracle.com/javase/tutorial/java/concepts/interface.html)

(A partir de Java 8, une interface peut fournir une implémentation par défaut pour une méthode. Les classes qui réalisent l'interface peuvent redéfinir ou pas cette méthode. Si elles ne ne la redéfinissent pas, c'est la méthode donnée dans l'interface qui sera invoquée par les classes appelantes.

That's all folks

Revision : Mai 2019