Chapitre 3

Polymorphisme et Interfaces

Certains exemples décrits dans ce chapitre ont été pris du livre "Big Java", chapitre 9, que je vous recommande de lire.

Polymorphisme

1. Faculté qu'on des objets différents de réagir différemment en réponse au même message.

    - Marcher sur la queue d'un chat => il miaule,

    - Marcher sur la queue d'un chien => il aboie,

2. Même nom de fonction, plusieurs implantations:

    Fonction: opération addition (+)

        L'addition des nombres entiers :

            1 + 2 = 3

        L'addition des nombres complexes :

            1+2i + 3+4i = 4+6i

3. Polymorphisme permet de manipuler des objets sans connaître (tout a fait) le type.

4. Se traduit par:

    - la compatibilité par affectation entre un type classe et un type ascendant,

    - la ligature dynamique des méthodes.

    Lorsque la méthode appelée est déterminée à la compilation ont parle alors de lien statique. Lorsque cette méthode est déterminée à l'exécution on parle alors de lien dynamique.

5. Polymorphisme redéfinition et surdéfinition.

6. Conversion explicite de référence.

Réutilisation

Additionner les éléments passés en argument et garder en mémoire la valeur la plus élevée parmi un ensemble:

3 types d'arguments:

- Un double:

```
public void add( double x ) {
     sum = sum + x;
     if( count == 0 || maximum < x )
          maximum = x;
     count++;
}
```

- Un compte bancaire

```
public void add( CompteBanque x ) {
     sum = sum + balance( x );
     if( count == 0 || maximum.balance( ) < x.balance( ) )
          maximum = x;
     count++;
}
```

- Une pièce de monnaie

```
public void add( Piece x ) {
     sum = sum + valeur( x );
     if( count == 0 || maximum.valeur( ) < x.valeur( ) )
          maximum = x;
     count++;
}
```

Code identique, les différences sont dans la manière de mesurer!

```
sum = sum + getMesure( x );
if( count == 0 || maximum.getMesure( ) < x.getMesure( ) )
     maximum = x;
count++;
```

Reste à déterminer le type de la variable x?

Idéalement, x devrait être du type de toutes les classes qui possèdent la méthode getMesure. En Java, l'interface exprime ce concept.

```
public interface Mesurable {
     double getMesure( );
}
```

Une interface a la possibilité de représenter, sans les implémenter, un ensemble de comportements.

Méthode abstraite

Une méthode abstraite est une méthode dont la signature et le type de la valeur de retour sont fournis dans la classe et rien d'autre (pas le corps de la méthode i.e. définition).

Interface

C'est une classe dont toutes ses méthodes sont abstraites. Elle ne contient aucun champ uniquement des constantes.

- Syntaxe

  protection interface nom_interface { // etc.}

  protection peut-être public ou rien (paquetage). Si public, tous les membres de l'interface sont de facto "public".

- Implémentation

```
public interface I {
     void f(int n);
     void g();
}

class A implements I {
     // A doit redéfinir les méthodes f & g prévues dans l'interface
}
```

- On ne peut pas différer l'implémentation. (à revoir dans l'héritage)

- Une même classe peut implémenter plusieurs interfaces.

```
class X implements I,J {
}
```

- On pourra utiliser des variables de types interface, mais on ne peut pas instancier ces variables à des interfaces. On peut par contre utiliser le polymorphisme pour les affecter à des objets implémentant l'interface.

```
I m; // OUI!
I m = new I(); // FAUX!

I m = new A(); // OUI!

A a = new A();
a = m; // FAUX!
a = (A) m; // OUI!
```

(Pour plus de détails voir l'exemple herint.java)

- Une constante déclarée dans une interface est vue comme une variable static final. On ne peut pas donc modifier sa valeur. Elle est accessible de partout ou dans le paquetage (si l'interface n'est pas public).

InstanceOf

```
x instanceof y
```

Elle sert à déterminer si x est du type y. Cette méthode retourne true si le type de x est y, false sinon.

(Pour plus de détails voir l'exemple Testinstanceof.java)

Object

Un argument de type Object (le plus petit commun dénominateur de toutes les classes) peut accueillir n'importe quel type. Étudier les deux cas de figures se trouvant dans les répertoires mesure et mesure1.

Classes internes et classes anonymes

Une entité qui n'a pas de nom est dite "anonyme". Si vous utilisez une entité une seule fois dans un programme, pas besoin de lui assigner un nom.

```
DataSet d = new DataSet();

d.add(new CompteBanque(blabla));
d.add(new Piece(blabla));
```

Une classe anonyme c'est quand vous partez la définir en même temps!

```
d.add(new QcQchose(blablabla){
      // Les méthodes à définir ....
      // Nous allons retrouver cette forme prochainement! Bonjour les dégâts!
     }
)
```

Quand on utilise une classe dont un but est purement « technique », on peut la déclarer à l'intérieur de la méthode qui en a besoin. On peut aussi déclarer une classe interne dans une autre classe. Attention aux droits d'accès des variables locales des méthodes ou classes.