Classes / Objets (compléments)

Le modificateur **final**

Le langage java permet de bloquer la modification d'un attribut une fois initialisé à l'aide du modificateur **final** .

```
class Etudiant {
  final String formation = "Licence MIAGE";
}
```

On ne peut plus modifier la valeur de l'attribut d'instance formation .

Le modificateur **final**

On peut différer l'initialisation d'un attribut déclaré **final** en l'initialisant dans un constructeur :

```
class Etudiant {
  final String formation;

  Etudiant() {formation = "Licence MIAGE";}
}
```

Aucune méthode d'instance de l'objet ne pourra modifier la valeur de l'attribut d'instance `formation`.

> **Attention :** Si vous déclarez **final** un attribut, vous devez l'initialiser à sa déclaration ou dans un constructeur.

Surcharger un constructeur

En Java, il est possible de définir plusieurs versions d'une méthode ou d'un constructeur en faisant varier la liste des paramètres :

```
class Entier {
  private int x;

  public Entier() {x = 1;}
  public Entier(int valeur) {x = valeur;}
}
```

Surcharger un constructeur

Pourquoi surcharger un constructeur ?

La surcharge du constructeur de La classe `Entier` permet de créer des instances de la classe avec différents états initiaux :

```
Entier i1 = new Entier(); // état initial : x = 1
Entier i2 = new Entier(5); // état initial : x = 5
```

Surcharge des constructeurs

Combien de constructeurs est-il recommandé de définir dans une classe ?

La règle est que toutes les classes doivent possèder au moins un constructeur.
En revanche, il n'y a pas de recommandation sur le nombre de versions du constructeur qu'il faut définir. Dans la pratique, quand on définit une classe, la seule question est de se demander quel pourra être l'état initial d'une instance et quelle information devra donner un utilisateur de la classe pour l'initialiser.

Surcharge des méthodes

Il est possible de définir plusieurs versions d'une méthode en faisant varier la liste des arguments de la méthode :

```
class Entier {
  private int x;

  public void ajouter(int y) {x += y;}
  public void ajouter(Entier e) {x += e.x;}
}
```

La surchage de la méthode `ajouter` permet d'appeler la méthode en passant en argument un `int` ou la référence d'une instance de la classe `Entier` :

```
  Entier e1 = new Entier();
  e1.ajouter(2); e1.ajouter(new Entier(3));
```

Surcharge des méthodes

Il est possible de faire varier le type de retour entre deux versions d'une méthode :

```
class Entier {
  public int multiplier(int a) {
    x = a * x; return x;}
  public void multiplier(Entier a) {
    x = x * a.x;}
}
```

> **Attention :** Vous ne pouvez pas définir deux versions d'une méthode ayant la même liste d'arguments et des types de retour différents.

Types primitifs et surcharge

Attention aux types primitifs :

- Il existe quatre types pour les entiers : **byte** , **short** , **int** et **long**
- Il existe deux types pour les décimaux : **float** et **double**

Une remarque : évitez de surcharger une méthode en faisant varier le type d'un entier ou le type d'un décimal. Sachez qu'à l'exécution d'un programme java, il est utilisé la promotion de type pour sélectionner la version de la méthode à appeler avec les types entiers et décimaux.

Promotion de type : La promotion de type désigne le mécanisme selon lequel on recherche la représentation binaire pouvant recevoir une valeur numérique. En java, les règles de promotion de type entre les différents types numériques sont :

byte $\rightarrow$ **short** $\rightarrow$ **int** $\rightarrow$ **long** $\rightarrow$ **float** $\rightarrow$ **double** .