0-M02 – Introduction à la programmation
Introduction à l'algorithmique

Cours n°4
Surcharges, collections et attributs de classe

Camille Coti
camille.coti@iutv.univ-paris13.fr

IUT de Villetaneuse, département R&T

2011 – 2012

La documentation Java

`http://download.oracle.com/javase/6/docs/api/`

Documentation de l'API Java

- En ligne sur le site de Oracle / Sun : API standard du JDK / SDK
- Possibilité de générer la Javadoc d'un projet
  - Système d'annotation du code
  - Génération automatique avec l'outil `javadoc`

Référence pour programmer !

Surcharge des méthodes

Surcharge

Une méthode (procédure ou fonction) peut être définie plusieurs fois dans une classe en recevant des arguments différents en types et en nombres : on dit alors que cette méthode est **surchargée**

Exemple : un constructeur qui prend

- Des arguments correspondant à des valeurs d'initialisation d'attributs de l'objet créé
- Ou aucun argument, l'objet étant créé en utilisant des valeurs par défaut.

Attention cependant :

- On ne peut pas avoir deux fonctions qui prennent les mêmes arguments mais ne retournent pas la même chose
- Les **types** sont importants, attention aux ambiguïtés

D'où l'importance de passer les bons paramètres quand on appelle des méthodes ou que l'on définit des méthodes d'une interface !

Surcharge des méthodes : exemple

Exemple avec la classe Cercle :

```
1  class Cercle {
2    /* Constante */
3    final int DEFAULT 20
4    /* Attributs */
5    private int diametre;
6    /* Constructeur */
7    public Cercle( ) {
8      diametre = DEFAULT;
9    }
10   public Cercle(int _diametre){
11     this.diametre = _diametre;
12   }
13 }
```

On a défini 2 constructeurs :

- Un constructeur qui ne prend pas d'arguments et initialise le diamètre à une valeur par défaut
- Un constructeur qui prend le diamètre du cercle par défaut

Utilisation :

```
1 Cercle c1, c2;
2 c1 = new Cercle();
3 c2 = new Cercle( 10 );
```

Lors de l'appel, le système choisit la méthode dont le prototype correspond à la forme de l'appel.

Attributs de classe

Attributs d'objet

Par défaut les attributs sont propres à un objet et font partie de la structure de données stockée en mémoire correspondant à cet objet : ce sont des **attributs d'objet**.

Attributs de classe

On peut définir des attributs comme étant présents en un seul exemplaire dans le programme, partagé entre tous les objets de la classe correspondante : ce sont des **attributs de classe**.

```
1  class Cercle {
2    /* Attributs */
3    private int diametre;
4    public static int nb = 0;
5    /* Constructeur */
6    public Cercle( int _diam ) {
7      diametre = _diam;
8      nb++;
9    }
10 }
```

- Définition d'un attribut de classe avec le mot-clé **static**
- Accès depuis une autre classe : **Cercle.nb**
  - On utilise le **nom de la classe**

Méthode de classe

Méthode de classe

Une **méthode de classe** est une méthode dont le comportement n'est pas lié à un objet en particulier, et qui n'accède qu'à des attributs de classe.

Exemples :

- `Math.random()` est une méthode de la classe Math
- La méthode `main()`
  - `public static void main( String[] ) /* ... */`

Définition d'une méthode de classe :

- Utilisation du mot-clé **static**
  - `public static int count()`

Utilisation d'une méthode de classe :

- Appel avec le **nom de la classe**
- Car la méthode est liée à la classe et non pas à un objet en particulier
  - `Cercle.count()`

```
1  class Cercle {
2    /* Attributs */
3    private int diametre;
4    public static int nb = 0;
5    /* Constructeur */
6    public Cercle( int _diam ) {
7      diametre = _diam;
8      nb++;
9    }
10   /* Méthode de classe */
11   public static int count( ) {
12     return nb;
13   }
14 }
```

Collection

Définition

Une **collection** est une structure de donnée amenée à contenir n'importe quel type de données, ou n'importe quelle classe dérivant de la classe `Object`.

Concrètement :

- Structure de données
- Les données sont organisées de façon à avoir certaines propriétés
  - Les propriétés sont spécifiques suivant la collection
  - Chaque collection fournit des outils pour manipuler les données
- On ne copie pas les objets dans la collection
  - on insère une référence (en Java, tout est référence)

Parcours d'une collection

Itérateur monodirectionnel

- On a une interface **Iterator**
- Méthodes **next()**, **hasNext()**
- Méthode **remove()**

```
1 /* on a une collection c */
2 Iterator iter = c.iterator();
3 while( iter.hasNext() ) {
4   Toto t = iter.next();
5   /* Opérations sur l'objet t */
6   ter.remove();
7 }
```

Itérateur bidirectionnel

- On a une interface **ListIterator**
- Méthodes supplémentaires **previous()**, **hasPrevious()**, **add()**

```
1 /* on a une liste l */
2 Iterator iter = l.listIterator( l.size());
3 while( iter.hasPrevious() ) {
4   Toto t = iter.previous();
5   /* Opérations sur l'objet t */
6   ter.remove();
7 }
```

Les listes

Classe **LinkedList** : liste chaînée

- Création : **LinkedList l = new LinkedList();**
  - Possibilité de passer en paramètre une collection
- Ajout, suppression : **add()**, **addFirst()**, **remove()**
- Taille : **size()**

Classe **ArrayList** : vecteur dynamique

- Création : **ArrayList a = new ArrayList();**
- Mêmes méthodes que LinkedList
- Supression de plusieurs éléments consécutifs : **a.removeRange(3, 6)**
- Accès à un élément d'indice `i` : **a.get(i)**

Les classes `LinkedList` et `ArrayList` héritent de `List`

- Possibilité d'utiliser un itérateur **bidirectionnel**

Les ensembles

Un ensemble est une **collection non-ordonnée d'éléments**

- Aucun élément ne peut s'y trouver plusieurs fois

Classe **HashSet** : table de hachage

- Création : **HashSet h = new HashSet();**
  - Possibilité de passer en paramètre une collection
- Possibilité d'utiliser un itérateur **monodirectionnel**
- Ajout, suppression : **add()**, **remove()**
- `add()` retourne un **booléen** : false si l'objet s'y trouve déjà
- Code de hachage d'un objet : **hashCode()**

Classe **TreeSet** : arbre binaire

- Création : **TreeSet t = new TreeSet();**
- Ajout, supression : comme HashSet
- `TreeSet` est un ensemble **ordonné**
- Notion de premier et dernier élément : **first()** et **last()**

Outils sur les collections

Recherche de max ou de min

- Les objets doivent implémenter l'interface `Comparable`
  - Méthode `compareTo()`
- Méthodes **min()**, **max()**
- Exemple : si on a une liste l : `Collections.max( l )`

Tri

- Les objets doivent implémenter l'interface `Comparable`
- Méthode **sort()**
- Exemple : si on a une liste l : `Collections.sort( l )`

Mélange

- Méthode **shuffle()**
- Exemple : si on a une liste l : `Collections.shuffle( l )`