Le langage Java

Le tri, classes utilitaires et génériques

Le tri d’une List

En Java il est possible de trier le contenu d’une liste.

- Collections.sort(**maListe**);
- Cependant pour que cela fonctionne il faut que les objets contenus dans « **maListe** » soient **triables**, i.e. que Java sache comment les trier.
- Java sait trier une liste de String, une liste de wrappers (Integer, Float, Double, Long, etc…)
- Java ne sait pas trier vos classes.

Le tri d’une List

Le tri d'une liste est possible si la classe est "Comparable" (implémente l'interface java.util.Comparable).

La classe doit implémenter une méthode qui permet à Java d'ordonner le contenu de la liste.

java.lang.Comparable

```
interface Comparable<T> {
        int compareTo(T o);
}
```

- En implémentant l'interface **Comparable<T>**, la classe doit redéfinir la méthode de comparaison entre 2 instances.
- Ici **T** est un type générique qui peut être remplacé par n'importe quelle classe comme un User (cf. ci-dessous), String, Double, Integer, etc…

Exemple:

```
interface Comparable<User> {
        int compareTo(User o);
}
```

Comment fonctionne compareTo ?

```
interface Comparable<T> {
        int compareTo(T o);
}
```

Si la méthode compareTo

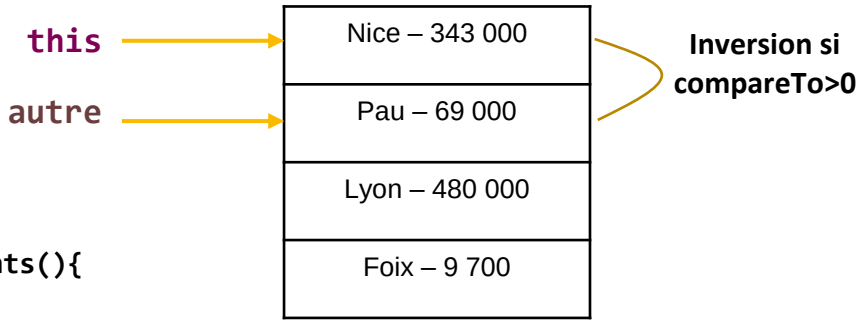
- retourne > 0 alors l'objet courant est « plus grand » que l'objet passé en paramètre.
- retourne < 0 alors l'objet courant est « plus petit » que l'objet passé en paramètre.
- retourne 0 alors les 2 objets sont identiques du point de vue du tri.

Exemple: la classe Ville

```
public class Ville implements Comparable<Ville> {
        private int nbHabitants;
        private String nom;

        public int compareTo(Ville autre) {

             if (this.nbHabitants > autre.getNbHabitants(){
                  return 1;
             }
             if (this.nbHabitants < autre.getNbHabitants(){
                  return -1;
             }
             return 0;
        }
}
```



Exemple: la classe String

```
System.out.println("Coucou".compareTo("Bonjour"));
```

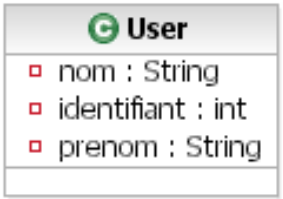
➔ *Affiche 1 car « Coucou » est plus grand que « Bonjour »*

Problématique du tri

Considérons une classe ayant plusieurs attributs.

- Plusieurs objets de cette classe sont placés dans une Collection
- Comment spécifier les critères de tri des éléments ?

Ex : on veut que les objets User soient triés par leur nom.



Solution 1 : Comparable

Première solution :

- Implémentation de l'interface **java.lang.Comparable**
- Exemple de comparaison sur **le nom**:

```
public class User implements Comparable<User> {

        private String nom;

        public int compareTo(User user2) {
            int result = this.nom.compareTo(user2.getNom());
            return result;
        }
}
```

Comparable - fonctionnement

Le concept du **Comparable** est de :

- Comparer l'objet courant à un autre objet passé en paramètre. Dans l'exemple ci-dessous l'autre objet est **user2**.
- La méthode **compareTo** retourne:
  - *1 si l'objet courant est **plus grand** que **user2***
  - *0 si les objets sont identiques*
  - *-1 si l'objet courant est **plus petit** que **user2***

```
public class User implements Comparable<User> {

        private String nom;

         public int compareTo(User user2) {
            int result = this.nom.compareTo(user2.getNom());
            return result;
         }
}
```

Tri avec la classe Collections

java.util.Collections

- Méthodes utilitaires pour la manipulation des collections.
- Méthode **sort(List)** pour trier:
  - *Fonctionne uniquement si la classe User implémente Comparable*

```
List<User> list = new ArrayList<>();
User u1 = new User("jean", "dupont");
list.add(u1);
User u2 = new User("jean", "durand");
list.add(u2);
User u3 = new User("jean", "martin");
list.add(u3);

Collections.sort(list);
```

Solution 2 : Comparator

Deuxième solution :

- Création d'un Comparator

```
import java.util.Comparator;

public class UserComparator implements Comparator<User> {

        public int compare(User o1, User o2) {
            int result = o1.getNom().compareTo(o2.getNom());
            return result;
        }

}
```

Avantage de cette solution : il est possible de créer plusieurs Comparators pour une même classe, et de choisir lequel appliquer selon les cas.

Comparator - fonctionnement

Le concept du **Comparator** est de :

- Comparer 2 objets passés en paramètre de la méthode **compareTo**.
- La méthode **compare** retourne:
  - *1 si le premier paramètre est "plus grand" que le second paramètre*
  - *0 si les objets sont identiques du point de vue du tri.*
  - *-1 si le premier paramètre est "plus petit" que le second paramètre*

```
public class UserComparator implements Comparator<User> {

        public int compare(User o1, User o2) {
            int result = o1.getNom().compareTo(user2.getNom());
            return result;
        }

}
```

Tri avec la classe Collections

java.util.Collections

- Méthode **sort(List, Comparator)** pour trier:

```
List<User> list = new ArrayList<>();
User u1 = new User("jean", "dupont");
list.add(u1);
User u2 = new User("jean", "durand");
list.add(u2);
User u3 = new User("jean", "martin");
list.add(u3);

Collections.sort(list, new UserComparator());
```

Chapitre 5
Les classes utilitaires

Collections et Arrays

La classe Collections

java.util.Collections

- Méthodes utilitaires pour la manipulation des collections.

```
List list = …

Collections.sort(list);
// tri de la liste (les éléments doivent implémenter
// l'interface Comparable)

Collections.sort(list, new UserComparator());
// tri de la liste en fonction d'un Comparator

List l1 = Collections.unmodifiableList(list);
// renvoie une collection non-modifiable

List l2 = Collections.synchronizedList(list);
// liste synchronisée pour gérer les accès concurrents
```

La classe Collections – ajout d'éléments

java.util.Collections

- Méthode pour peupler une liste.

```
ArrayList<String> liste = new ArrayList<>();
Collections.addAll(liste, "coucou", "hello", "bonjour");
```

La classe Arrays

java.util.Arrays

- Méthodes utilitaires pour les tableaux

```
User[] userTab = new User[5];
// initialisation du tableau
List userList = Arrays.asList(userTab);

Arrays.sort(userTab);
// trie le tableau
```

Atelier (TP)

Objectifs du TP: savoir trier des collections d'objets

Description du TP:

Dans ce TP, vous allez mettre en place du tri sur des ArrayList.

Chapitre 6
Les génériques

Java 5+ et les génériques

Présentation

Les Génériques sont très attendus par la communauté Java.

- Il existait un comité de réflexion sur les génériques depuis plus de 5 ans.

Les Génériques devraient permettre d'éviter de nombreuses erreurs d'exécution.

Principe de base : typer les éléments d'une collection.

Exemple sans les génériques

Il n'est pas possible de spécifier le type d'élément que doit stocker la liste.

Obligation de caster un élement dès qu'on l'extrait de la liste.

```
List voitureList  = new ArrayList();
voitureList.add(new Voiture());

//…

Voiture v1 = (Voiture) voitureList.get(0);
```

Cette méthode était la seule possible avant Java 5.0

Exemple avec les génériques

La liste est typée lors de sa création.

- elle ne peut recevoir qu'un type d'élément donné.

On n'a plus besoin de caster lors de l'extraction.

```
List<Voiture> voitureList  = new ArrayList<Voiture>();
voitureList.add(new Voiture());
//…
Voiture v1 = voitureList.get(0);
```

Exemple de définition

Démo List
Démo Map

```
public interface Iterable<T> {
    /**
     * Returns an iterator over elements of type {@code T}.
     *
     * @return an Iterator.
     */
    Iterator<T> iterator();

}
```

FIN

FIN