Introduction programmation Java

Cours 8: Map et Notion d'ordre

Stéphane Airiau

Université Paris-Dauphine

Map

un `Map` représente une relation binaire : chaque élément d'un `Map` est une paire entre une clé et une valeur.

Dans un `Map`, chaque clé est unique, mais on peut avoir des doublons pour les valeurs.

Attention, `Map` n'est pas une sous interface de `Iterable`, donc on ne peut pas parcourir un `Map` avec une boucle `for each` !

On peut obtenir l'ensemble des clés, l'ensemble des valeurs, et l'ensemble des paires (clé,valeur) grace aux méthodes suivantes :

- `Set<K> keySet()`
- `Set<Map.Entry<K,V» entrySet()`
- `Collection<V> values()`

`Map.Entry` désigne une classe `Entry` qui est interne à la classe `Map`. On peut créer des classes à l'intérieur de classes, mais je n'en parlerai pas plus aujourd'hui.

Exemple parcours d'une `Map`

```
1  Map<Personnage,Region> origines = new HashMap<>();
2  ...
3  for (Map.Entry<Personnage,Region> paire: origines.entrySet()){
4     Personnage p = paire.getKey();
5     Region r = paire.getValue();
6     if (r.getName().equals("Ibère"))
7       System.out.println(p);
8   }
```

Dans cet exemple, on part une `Map` qui associe à chaque personnage sa région d'origine et on affiche seulement les personnages qui sont des ibères.

Ordre

Interface `Comparable` contient une seule méthode :

```
public int compareTo(T o)
```

Cette méthode retourne

- un entier négatif si l'objet est plus petit que l'objet passé en paramètre
- zéro s'ils sont égaux
- un entier positif si l'objet est plus grand que l'objet passé en paramètre.

les classes `String`, `Integer`, `Double`, `Date`, `GregorianCalendar` et beaucoup d'autres implémentent toutes l'interface `Comparable`.

Exemple

```
1  public class Gaulois extends Personnage
2                  implements Comparable<Gaulois>{
3    String nom;
4    int quantiteSanglier;
5      ...
6
7    public int compareTo(Gaulis ixis) {
8      return this.quantiteSanglier - ixis. quantiteSanglier;
9    }
10 }
```

interface `Comparator`

ex : trier les éléments d'une collection : utilisation interface `Collections`

```
   // interface Collections
1  public static <T extends Comparable<? super T»
2                  void sort(List<T> list)
3  public static <T> void sort
4                  (List<T> list, Comparator<? super T> c)
```

```
1  public interface Comparator<T> {
2     int compare(T o1, T o2);
3     boolean equals(Object obj);
4  }
```

Pour comparer des `Gaulois`, et même tous les `Personnage` selon leur taille, on peut écrire la classe suivante :

```
1  public class OrdreHauteur implements Comparator<Personnage> {
3      public int compare(Personnage gauche, Personnage droit){
4        return gauche.hauteur < droite.hauteur ? -1:
5              (gauche.hauteur == droite.hauteur ? 0 : 1);
5      }
6    }
```

Exemple

Ensuite, on peut utiliser cette nouvelle classe pour trier des `Personnages` selon leur taille.

```
1  public static void main(String[] args){
2    Personnage obelix = new IrreductibleGaulois("Obelix", 1.81);
3    Gaulois asterix = new IrreductibleGaulois("Astérix", 1.60);
4    Personnage cesar = new Personnage("César", 1.75);
5
6    ArrayList<Personnage> personnages =
7                    new ArrayList<Personnage>();
8    personnages.add(asterix);
9    personnages.add(obelix);
10   personnages.add(cesar);
11
12   for (Personnage p: personnages)
13     System.out.println(p.presentation());
14
15   Comparator<Personnage> ordre = new OrdreHauteur();
16   Collections.sort(personnages, ordre);
17
18   for (Personnage p: personnages)
19     System.out.println(p.presentation());
```