Bases de la POO / Java

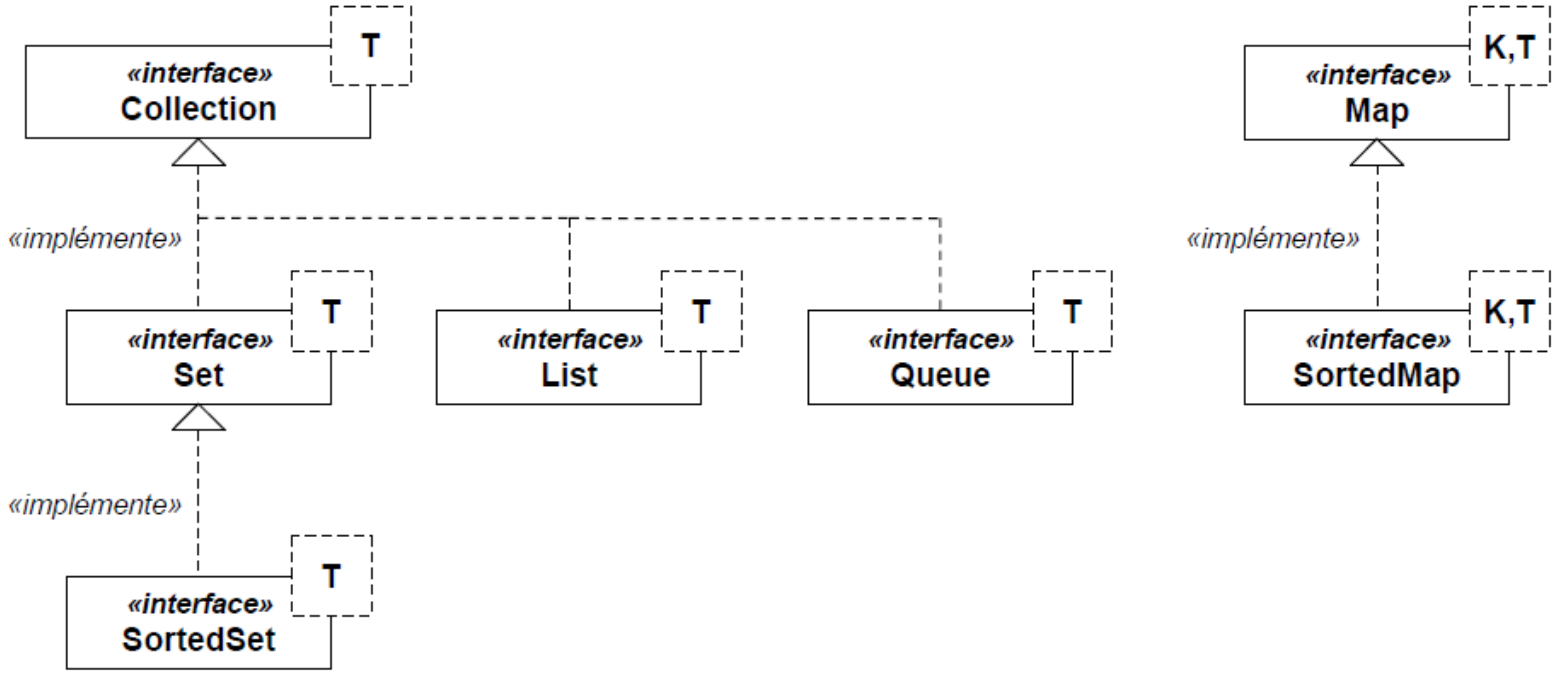
Les collections génériques

API Collections

- Toutes les collections non génériques sont passées en version générique avec Java 1.5
  - Package «**java.util**»
  - Ensembles: **Set**, **SortedSet**, **HashSet**, **TreeSet**
  - Listes: **List**, **ArrayList**, **Vector**, **LinkedList**
  - Files d'attente: **Queue**, **PriorityQueue**
  - Associations: **Map**, **SortedMap**, **HashMap**, **TreeMap**
- Compatibilité entre les deux versions
  - Avant Java 5 et après

Hiérarchie

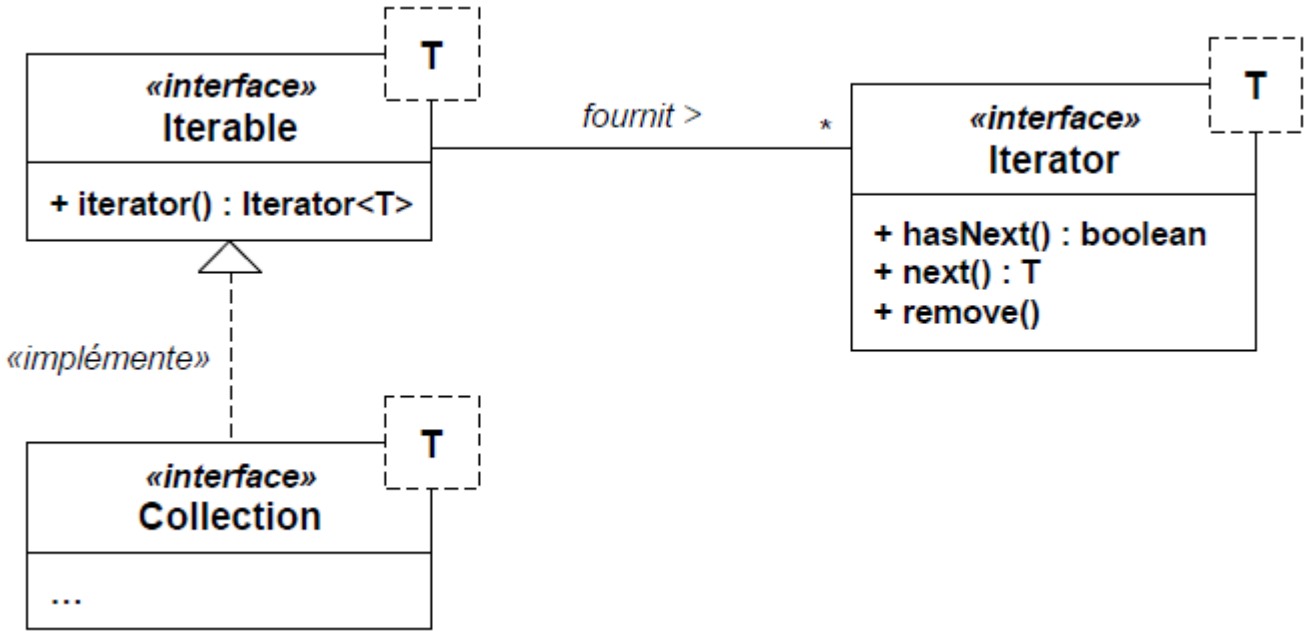
- Deux interfaces racines
  - `Collection`: collections classiques
  - `Map`: collections associatives



Itérateurs

- Pour parcourir une collection
  - Pointe sur un élément d'une collection
  - Permet de passer d'un élément à un autre

- «**Collection**» implémente l'interface «**Iterable**»
  - Méthode: **iterator() : Iterator<T>**


«interface»
Iterable
T
+ iterator() : Iterator<T>
fournit >
*
«interface»
Iterator
T
+ hasNext() : boolean
+ next() : T
+ remove()
«implémente»
«interface»
Collection
T
...

Parcours des éléments

- **Avec un itérateur**

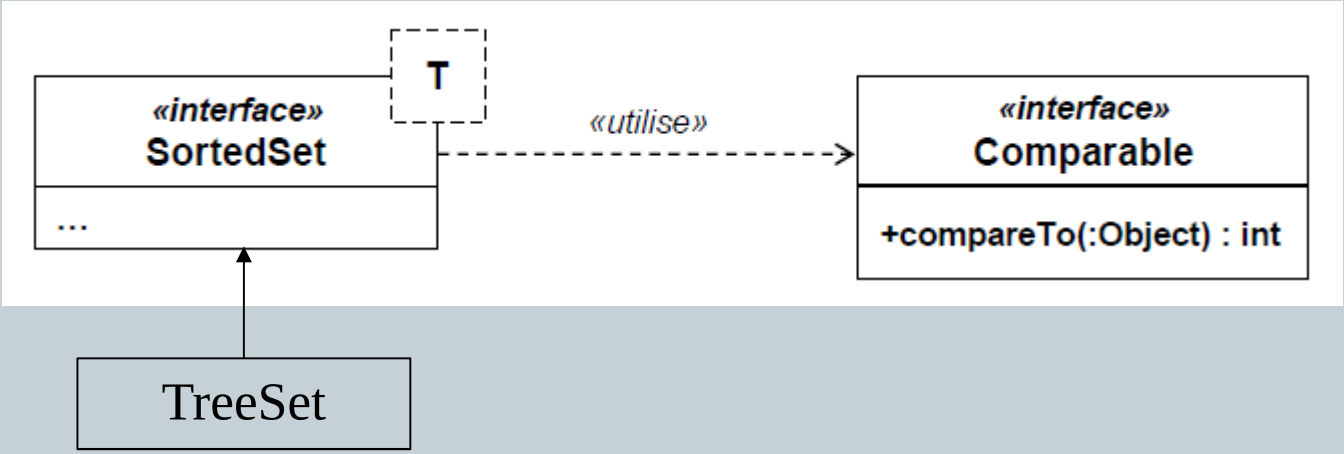
```
void afficher(Collection<String> c)
{
    Iterator<String> i = c.iterator();
    while (i.hasNext())
            System.out.println(i.next());
}
```

- **A partir de Java 1.5, nouvelle boucle for**
- **for (**variable **:** collection**)**

```
void afficher(Collection<String> c)
{
    for (String e : c) System.out.println(e);
}
```

Collections triées (1)

- Pour trier des éléments, il faut pouvoir les comparer
- Soit ils implémentent l'interface «**Comparable**»
  - Solution retenue avec une construction par défaut
  - on utilise **TreeSet** qui hérite de **SortedSet**



Exemple 1

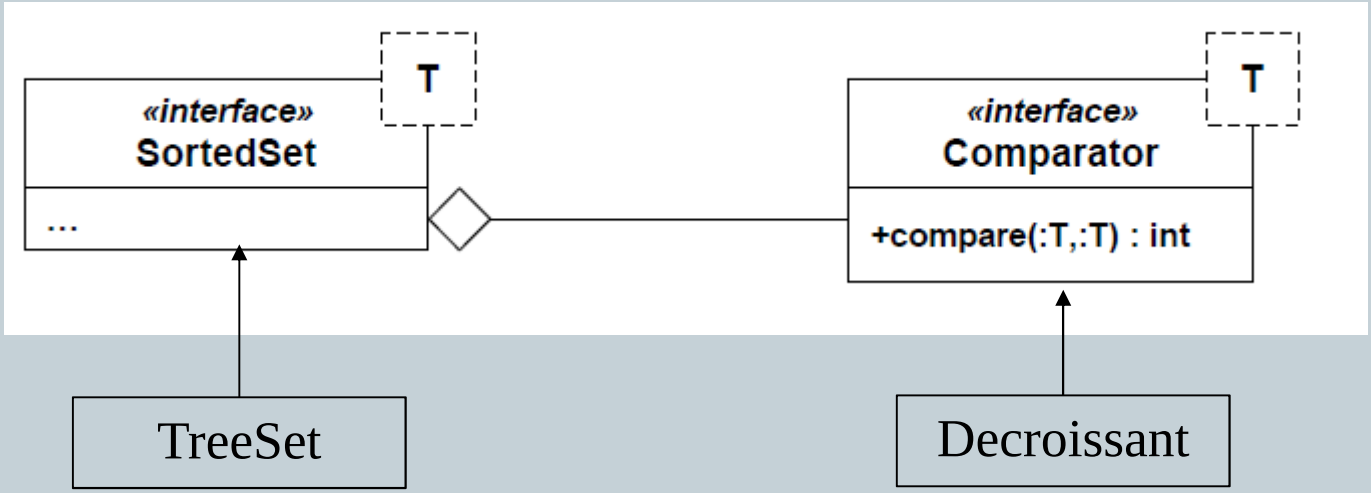

```
Ensemble trié de String
Jean
Marie
Paul
ordre naturel
```

```
System.out.println("\nEnsemble trié de String ");

TreeSet<String> s = new TreeSet<String>();
            s.add("Marie");
            s.add("Jean");
            s.add("Paul");
afficher(s);

Comparator t = s.comparator();
if (t == null) System.out.println("ordre naturel");
```

Collections triées (2)

- Soit la collection possède un objet de type «**Comparator**»
  - Comparateur fourni à la construction

  - On utilise toujours TreeSet, mais un 2e constructeur
  - **ss = new TreeSet<String>(new Decroissant());**
  - **La classe Decroissant définit la méthode compare**



Ordre décroissant


```
import java.util.*;
public class Decroissant implements Comparator<String>
{
    public int compare(String o1, String o2)
       {
              int resu = o1.compareTo(o2);
              if (resu<0) return 1;
              if (resu>0) return -1;
              return resu;
       }
}
```

Exemple 2


```
Ensemble trié de String
Paul
Marie
Jean
ordre decroissant
```

```
System.out.println("\nEnsemble trié de String");

Decroissant d = new Decroissant();
TreeSet<String> ss = new TreeSet<String>(d);

            ss.add("Marie");
            ss.add("Jean");
            ss.add("Paul");
afficher(ss);

Comparator ts = ss.comparator();
if (ts == d) System.out.println("ordre decroissant");
```