JAVA

Les collections

La généricité en Java

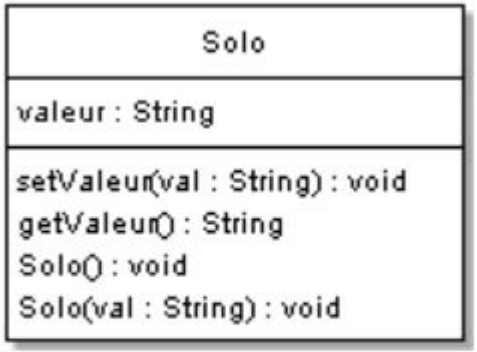
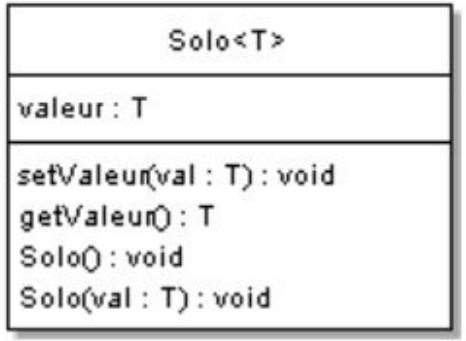


Diagramme de la classe Solo

Comment implémenter une classe ***Solo*** qui permet de travailler avec n'importe quel type de données?

La généricité en Java

```java
public class Solo<T> {
	private T valeur;
	public Solo(){
	  this.valeur = null;
	}
	public Solo(T val){
	  this.valeur = val;
	}
	public void setValeur(T val){
	  this.valeur = val;
	}
	public T getValeur(){
	  return this.valeur;
	}
	public static void main(String[] args) {
	    Solo<Integer> soloI = new Solo<Integer>();
	    Solo<String> soloS = new Solo<String>("TOTOTOTO");
	    Solo<Float> soloF = new Solo<Float>(12.2f);
	    Solo<Double> soloD = new Solo<Double>(12.202568);
	  }
}
```



Objet générique

Collections: Généralités

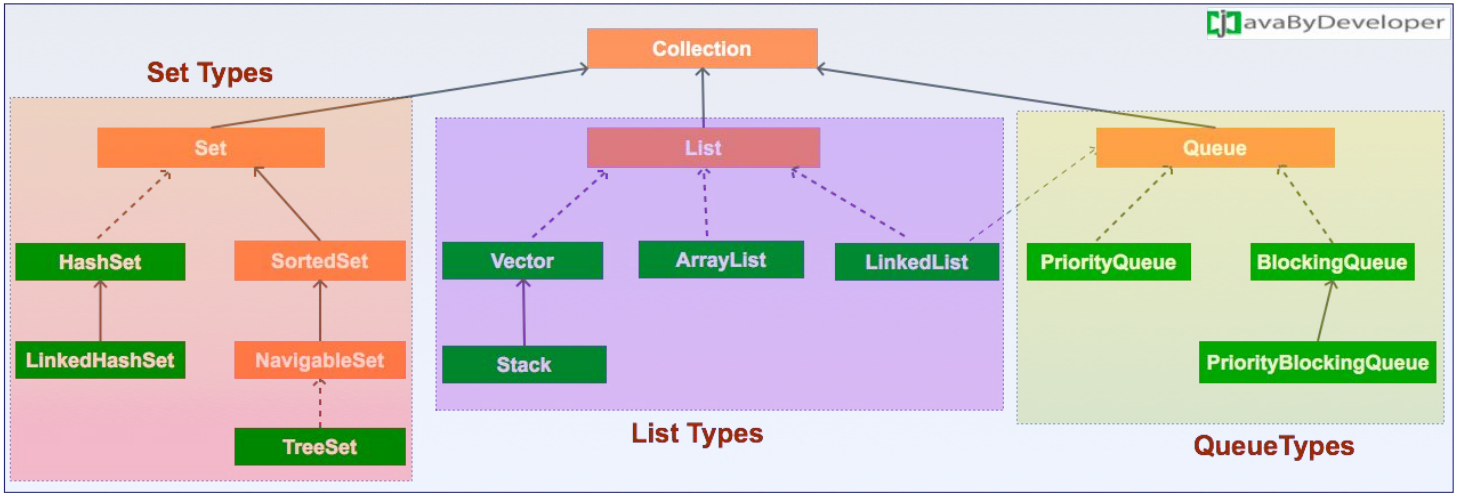
1-Les collections sont des objets qui permettent de stocker et de manipuler des objets autrement qu'avec un tableau conventionnel

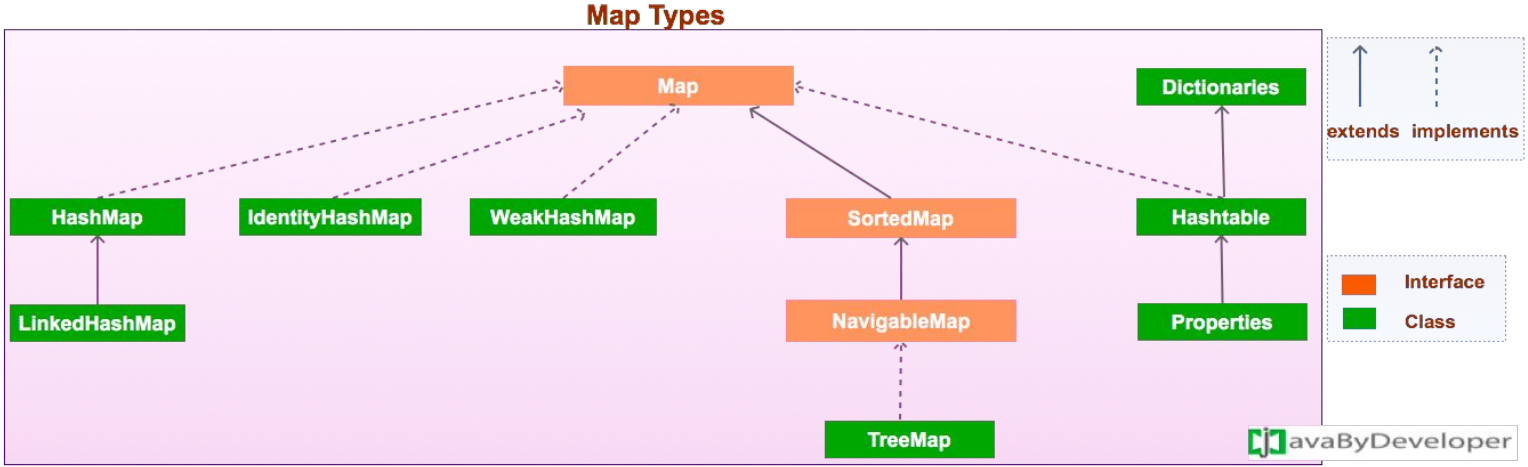
2-permettent de stocker des objets de différentes manières

- sous la forme d'une pile ;
- comme une liste chaînée ;
- sous la forme d'une structure clé-valeur ;

3- Les collections Java forment donc un framework constitué d'un ensemble d'interfaces.

Collection Framework





Généralités

Set: une collection qui ne tolère pas les doublons de données.

List: liste ordonnée qui accepte les valeurs en double.

Queue: ce type de collections peut s'apparenter à une file d'attente.

SortedMap: cette interface permet de stocker des éléments ordonnés par ordre croissant

SortedSet: permet aussi d'avoir un stockage ranger par ordre croissant.

Deque: permet d'insérer et d'enlever des éléments aux deux bouts de la collections

Parcourir une collection

Deux façons de parcourir une collection, en fonction de son type :

- Collection :

  On utilise un itérateur ou une simple boucle ;
- Map:

  On utilise une Collection.

L'interface Iterator:

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove();
}
```

Parcourir une collection:

```
import java.util.ArrayList;
import java.util.Iterator;
public class Main {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("1");
        list.add("2");
        list.add("3");
        list.add("4");
        list.add("5");
        Iterator it = list.iterator();
        while(it.hasNext()){
            String str = (String)it.next();
            if(str.equals("4")|| str.equals("2")) it.remove();
        }
    }
}
```

Parcourir un objet Map

Les classes implémentant cette interface gèrent des couples d'éléments <K ,V>. On a alors trois façons de parcourir une telle collection, en implémentant des méthodes de l'interface ***Collection***:

1. la méthode ***keySet()*** qui retourne une collection de type ***Set<K>***
2. La methode ***entrySet()*** qui retourne une collection de type ***Set<Entry<K, V>>***
3. La methode ***values()*** qui retourne une collection de type ***Collection<K>***

Parcourir un objet Map : *keySet()*

```
1  package collections;
2  import java.util.Collection;
3  import java.util.HashMap;
4  import java.util.Iterator;
5  import java.util.Map;
6  import java.util.Map.Entry;
7  import java.util.Set;
8  public class Main2 {
9      public static void main(String[] args) {
10         Map<Integer, String> map = new HashMap<Integer, String>();
11         map.put(1, "un");
12         map.put(2, "deux");
13         map.put(3, "trois");
14         map.put(4, "quatres");
15         map.put(5, "cinq");
16         Set<Integer> setInt = map.keySet();
17         Iterator<Integer> it = setInt.iterator();
18         System.out.println("Parcours d'une Map avec keySet : ");
19         while(it.hasNext()){
20            int key = it.next();
21            System.out.println("Valeur pour la clé " + key + " = " + map.get(key));
22         }
```

Parcourir un objet Map : *entrySet()*

```java
package collections;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
public class Main2 {
   public static void main(String[] args) {
      Map<Integer, String> map = new HashMap<Integer, String>();
      map.put(1, "un");
      map.put(2, "deux");
      map.put(3, "trois");
      map.put(4, "quatres");
      map.put(5, "cinq");
      Set<Entry<Integer, String>> setEntry = map.entrySet();
      Iterator<Entry<Integer, String>> itEntry = setEntry.iterator();
      System.out.println("Parcours d'une Map avec setEntry : ");
      while(itEntry.hasNext()){
         Entry<Integer, String> entry = itEntry.next();
         System.out.println("Valeur pour la clé " + entry.getKey() + " = " + entry.getValue());
      }

   }
}
```

Parcourir un objet Map : *values()*

```
1  package collections;
2  import java.util.Collection;
3  import java.util.HashMap;
4  import java.util.Iterator;
5  import java.util.Map;
6  import java.util.Map.Entry;
7  import java.util.Set;
8  public class Main2 {
9      public static void main(String[] args) {
10         Map<Integer, String> map = new HashMap<Integer, String>();
11         map.put(1, "un");
12         map.put(2, "deux");
13         map.put(3, "trois");
14         map.put(4, "quatres");
15         map.put(5, "cinq");
16         Collection<String> col = map.values();
17         Iterator<String> itString = col.iterator();
18         System.out.println("Parcours de la liste des valeurs d'une Map avec values : ");
19         while(itString.hasNext()){
20            String value = itString.next();
21            System.out.println("Valeur : " + value);
22         }
23         System.out.println("-----------------------------------");
24     }
25 }
```

Trier des Collections: *TreeSet<T>*

Certaines implémentations de l'interface Collection savent par défaut trier leur contenu: ***TreeSet***

```
3  import java.util.Iterator;
4  import java.util.Set;
5  import java.util.TreeSet;
6
7  public class Trie {
8      public static void main(String[] args) {
9          Set<String> tree = new TreeSet<String>();
10         tree.add("Nadia");
11         tree.add("Yasser");
12         tree.add("Mohammed");
13         tree.add("Hanane");
14         tree.add("Badre");
15         Iterator<String> it = tree.iterator();
16         while(it.hasNext())
17            System.out.println(it.next());
18     }
19 }
```

Parcourir une collection: *Collections.sorte()*

Utiliser la méthode ***Collections.sorte()*** pour trier uniquement des ***List<T>***

```
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.Iterator;
6  import java.util.List;
7
8  public class Trie1 {
9      public static void main(String[] args) {
10         List<Integer> list = new ArrayList<Integer>();
11         list.add(24);
12         list.add(10);
13         list.add(52);
14         list.add(2);
15         Collections.sort(list);
16         Iterator<Integer> it = list.iterator();
17         while(it.hasNext())
18            System.out.println(it.next());
19     }
20 }
```

L’interface Comparable<T>

Un mécanisme dans Java pour savoir si un objet est plus grand qu'un autre.

la plupart des objets représentant les types de bases possèdent ce mécanisme.

L’interface ***Comparable*** possède une seule méthode ***copareTo(T objet)*** qui retourne:

- un entier négatif si le paramètre est plus petit,
- 0 si le paramètre est égal
- un entier positif négatif si le paramètre est plus grand

L’interface Comparable<T>

Les class java qui implémente par défaut l’interface Comparable<T>

Byte, Long, Integer, Short, Double, Float, BigInteger, BigDecimal	Tri du plus petit au plus grand.
Character	Tri du plus petit au plus grand en se servant de la représentation numérique du caractère.
Boolean	Tri du plus petit au plus grand, avec `false` plus petit que `true`.
String	Tri par ordre alphabétique.
File	Dépend des chemins d'accès aux fichiers et donc du système d'exploitation, mais se base sur l'ordre alphabétique des chemins.
Date	Tri chronologique.

L'interface Comparable<T>: Example

```
3  public class CD {
4          private String auteur, titre;
5          private double prix;
6          public CD(String auteur, String titre, double prix) {
7             this.auteur = auteur;
8             this.titre = titre;
9             this.prix = prix;
10         }
11         public String toString() {
12            return "CD [auteur=" + auteur + ", titre=" + titre + ", prix=" + prix + "]";
13         }
14         public String getAuteur() {
15            return auteur;
16         }
17         public String getTitre() {
18            return titre;
19         }
20         public double getPrix() {
21            return prix;
22         }
23     }
24
```

L'interface Comparable<T>: Example

```
3  public class CD implements Comparable{
4          private String auteur, titre;
5          private double prix;
6          public CD(String auteur, String titre, double prix) {
7             this.auteur = auteur;
8             this.titre = titre;
9             this.prix = prix;
10         }
11         public String toString() {
12            return "CD [auteur=" + auteur + ", titre=" + titre + ", prix=" + prix + "]";
13         }
14         @Override
15         public int compareTo(Object o) {
16            if(o.getClass().equals(CD.class)){
17               CD cd = (CD)o;
18               return this.auteur.compareTo(cd.getAuteur());
19            }
20            return -1;
21         }
22         public String getAuteur() {
23            return auteur;
24         }
25         public String getTitre() {
26            return titre;
27         }
28         public double getPrix() {
29            return prix;
30         }
```

L'interface Comparable<T>: Example

```
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.Iterator;
6  import java.util.List;
7
8  public class MainCD {
9     public static void main(String[] args) {
10       List<CD> list = new ArrayList<CD>();
11       list.add(new CD("AbdElhalime", "Kalimate", 7d));
12       list.add(new CD("Abd elouahab doukali", "Rahel Joual", 10.25d));
13       list.add(new CD("Bob Marly", "No women No cry", 10.25d));
14       list.add(new CD("Cat stevens", "Wild World", 15.30d));
15       System.out.println("Avant le tri : ");
16       Iterator<CD> it = list.iterator();
17       while(it.hasNext())
18          System.out.println(it.next());
19       Collections.sort(list);
20       System.out.println("Après le tri : ");
21       it = list.iterator();
22       while(it.hasNext())
23          System.out.println(it.next());
24    }
25 }
```

Note: Cas de deux CD de meme auteur et deux titres différents