Développement en JAVA

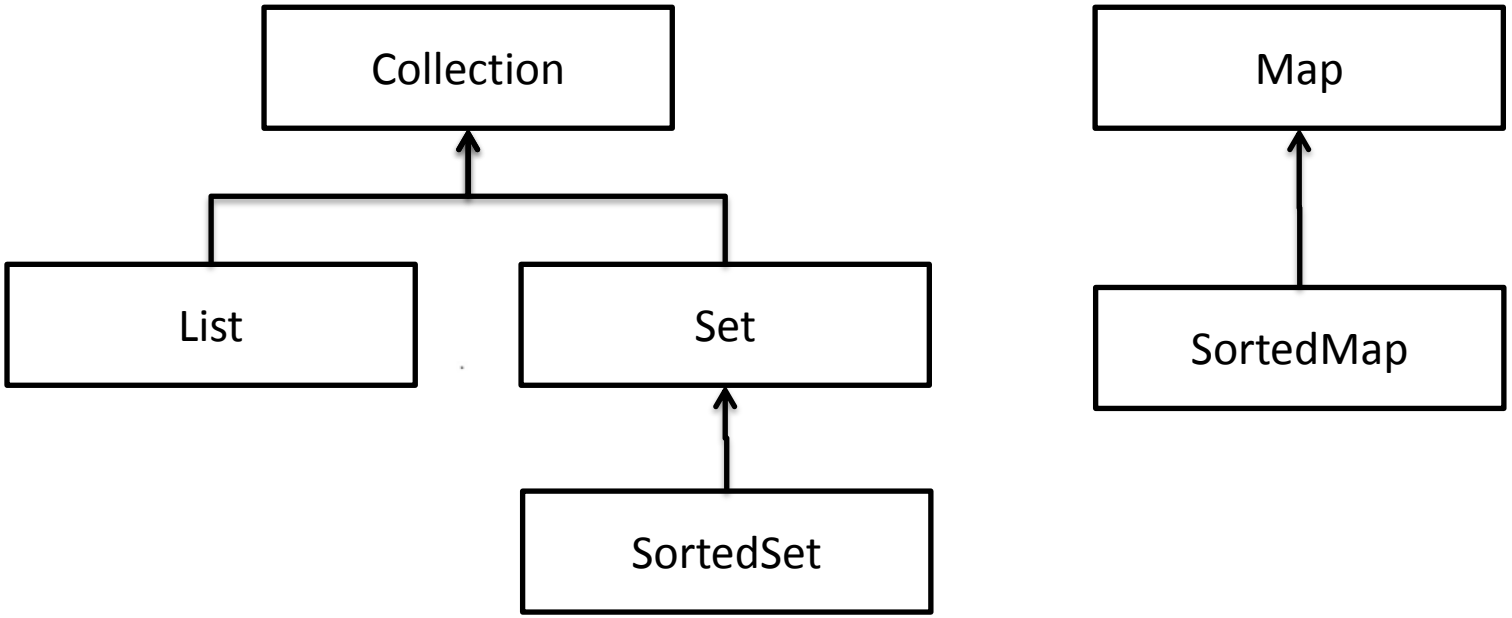
Les collections

Agenda

- Les interfaces Collections, List et Map
- La classe ArrayList
- La classe LinkedList
- La classe HashMap
- La classe Queue
- La classe Stack

Les collections

- L'API collections de Java offre des classes génériques pour pouvoir gérer des ensembles d'objets



L'interface Collection

- Une Collection est un ensemble d'objets
- L'interface Collection implémente l'interface Iterable
- Toute collection doit être paramétrisée en fonction d'une Classe
- Elle ne peut donc contenir des objets que d'un seul type ou de ses types dérivés

```
// Une collection d'utilisateurs
Collection<Utilisateur> utilisateurs;

// Une collection pouvant stocker n'importe quel objet
Collection<Object> objets;
```

L'interface Collection

- Les méthode les plus utiles
  - la collection possède-t-elle cet objet ?

    ```
    boolean contains(Object o)
    ```
  - la collection est-elle vide ?

    ```
    boolean contains(Object o)
    ```
  - récupération d'un iterateur

    ```
    Iterator<E> iterator()
    ```
  - récupération de la taille de la collection

    ```
    int size()
    ```
  - suppression de tous les éléments de la collection

    ```
    void clear()
    ```

Iterator

- Un iterator est un objet qui permet de parcourir une collection
- Un iterator possède notamment les méthodes
  - hasNext()
  - next()
  - remove()

Iterator

```
List<Objet> uneList = methodeQuiRetourneUneListe();

Iterator<Objet> it = uneListe.iterator();

while(it.hasNext()){
  System.out.println(it.next());
}
```

La boucle foreach

- La boucle `foreach` offre une manière simple de parcourir l'ensemble des éléments d'une collection

```
List<Objet> uneList = methodeQuiRetourneUneListe();

for(Objet obj: uneList){
  System.out.println(obj);
}
```

- On peut également l'utiliser avec les tableaux

L'interface List

- Les méthodes les plus utiles
  - Ajout d'un élément

```
void add(E element)
```

  - Ajout d'un élément à une position donnée

```
void add(int index, E element)
```

  - Récupération d'un élément

```
E get(int index)
```

  - Suppression d'un élément

```
void remove(int index)
```

L'interface List

- Hérite de l'interface Collection
- Représente une collections ordonnée d'éléments
- Un élément peut être inséré ou retiré à une position précise dans la liste
- Une collection de type List peut être converti en tableau à l'aide de la méhtode toArray()

```
List<String> uneListDeString;
String[] unTableauDeString = uneListeDeString.toArray();
```

ArrayList

- Correspond à une structure de données de type tableau
- Caractéristiques
  - Accès à une valeur rapide O(1)
  - Ajout d'une valeur en fin de liste rapide O(1)
  - Ajout d'une valeur en milieu de liste lente O(n)

```
List<String> uneListDeString = new ArrayList<String>;
```

LinkedList

- Correspond à une structure de données de type liste chainée
- Caractéristiques:
  - Lent en lecture O(n)
  - Insertion rapide à n'importe quel endroit de la liste O(1)

```
List<String> uneListDeString = new LinkedList<String>;
```

Exemple

```
List<Concert> concerts = new ArrayList<Concert>();

for(Concert c : concerts){
  System.out.println("Concert de " + c.chanteur + " le  "  +
c.date);
}
```

Map

- Une Map est un dictionnaire de type <clé, valeur>
- La classe Map n'hérite pas de Collection
- Une Map ne peut pas être parcourue avec une boucle foreach
- Les principales implémentation sont
  - **HashMap**
  - HashTable
  - TreeMap

Map

- Les méthodes les plus utiles
  - Ajout d'un élément

```
V put(K clé, V valeur)
```

  - Récupération d'un élément

```
V get(Object clé)
```

  - Suppression d'un élément

```
void remove(Object clé)
```

  - Récupération de toutes la valeurs

```
Collection<V> values()
```

Map

```
// La clé le nom d'utilisateur et la valeur un objet User
Map<String, User> usersMap = new HashMap<String, User>();

// Ajout d'un élément
usersMap.add(user1.name, user1);

// Rechcerche d'un élément existant ayant pour clé test
// Si la clé n'existe pas on récupère null
User user2 = usersMap.get("test");
```

- Pour qu'une Map soit performante il faut redéfinir la méthode equals() et la méthode hashCode()

Set

- Un Set est une collection
- C'est un ensemble d'objets n'acceptant pas les doublons
- Contrairement à une liste les éléments ne sont pas ordonnées, il n'est donc possible que de:
  - Ajouter un élément
  - Supprimer un élément
  - Parcourir l'ensemble des éléments du Set
- Comme pour une Map, l'utilisation d'un Set n'est pas pertinent en l'absence de redéfinition des méthodes equals et hashCode

Set

- Les méthodes les plus utiles
  - Ajout d'un élément

```
boolean add(E element)
```

  - Recherche d'un élément

```
boolean contains(E element)
```

  - Suppression d'un élément

```
void remove(Object o)
```

Set

```
Set<User> usersSet = new HashSet<String, User>();

// Ajout d'un élément
if(usersSet.add(user1)){
  // L'utilisateur a bien été ajouté au Set
}else{
  // Sinon c'est qu'il est déjà présent
}
```

Autres collections

- Autres types de collections couramment utilisés:
  - File de type FIFO

  `Queue<E>`
  - Pile de type LIFO

  `Stack<E>`