digitalent
FORMATION

Le langage Java

Collections

Sommaire

Limites des tableaux

- Les collections sont l'évolution logique des tableaux.
- Les tableaux peuvent être utilisés pour des opérations simples. Cependant :
  - La taille d'un tableau est définie lors de sa création
  - Si on supprime un élément au milieu d'un tableau, il faut gérer le décalage de tous les éléments qui se trouvaient après l'élément supprimé
  - Les éléments sont obligatoirement indexés par des entiers

```
int[] tab = new int[10];
tab[0]=1; tab[1]=2;…
```

exemple d'utilisation d'un tableau

Les collections (1/2)

- Les collections sont des classes Java permettant de faciliter la gestion d'ensembles d'éléments.
  - Opérations courantes :
    - Ajout d'élément
    - Suppression d'élément
    - Parcours de la Collection

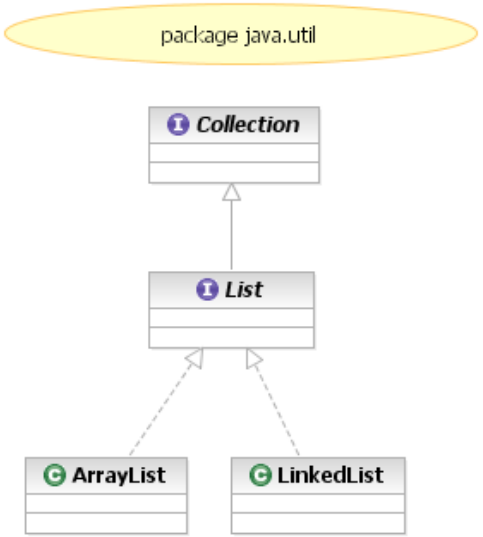
Les collections (2/2)

- De nombreuses collections sont fournies avec Java Standard Edition :
  - package java.util
  - Implémentent l'interface java.util.Collection
  - Permettent de stocker des références vers tous types d'objets
  - Pas de taille maximum prédéfinie
  - Elles sont optimisées en fonction de besoins précis.

Listes

Les listes

- Les listes sont des collections
  - Implémentent l'interface java.util.List qui hérite de l'interface Collection
- Les listes sont indexées



Les listes les plus courantes

- ArrayList
  - La plus utilisée
  - Souvent vue comme un "tableau dynamique"
  - Excellentes performances pour le parcours d'éléments
- LinkedList
  - Liste chaînée
  - Excellentes performances lors d'insertion/suppression d'éléments.
  - Mauvaises performances pour le parcours d'éléments

Il existe également la classe Vector (généralement déconseillée)

Ajout d'éléments (1/2)

- Les éléments sont ajoutés avec la méthode 'add(...)'
  - Par défaut, chaque élément est indexé en fonction de son ordre d'arrivée
  - Tout type d'objet peut être inséré dans une liste
    - tous les éléments sont stockés au format Object

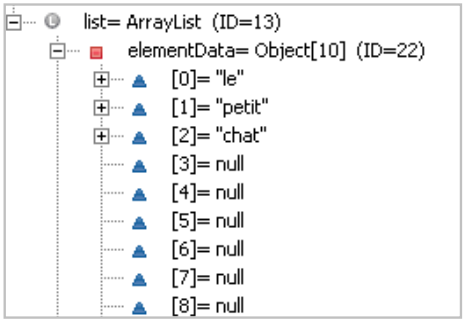
```
List list = new ArrayList();
list.add("le");
list.add("petit");
list.add("chat");
```

Il est possible (mais rare) d'insérer des éléments hétérogènes dans une même liste : String, Integer, CompteCourant…

Ajout d'éléments (2/2)

- Exemple : ArrayList
  - Éléments stockés dans un tableau à l'intérieur de l'objet ArrayList.
  - A chaque ajout, vérification que la taille maximum du tableau n'a pas été atteinte.
    - Si la taille maximum est atteinte, un nouveau tableau est créé.

```
List list = new ArrayList();
list.add("le");
list.add("petit");
list.add("chat");
```



Contenu de l'objet list

Suppression d'élément

- Méthode **remove(...)**
  - Possibilité de supprimer l'objet à partir de l'objet lui-même ou à partir de son index.

```
List 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);


list.remove(u3);          // suppression de la référence vers
u3


list.remove(0);           // suppression de la référence vers
                          // l'élément en position 0
```

Autres méthodes

- **size()**
  - Renvoie le nombre d'éléments de la collection
- **isEmpty()**
  - Renvoie 'true' si la collection est vide
- **toArray()**
  - Créé un tableau contenant tous les éléments de la liste

Ces méthodes sont déclarées dans l'interface Collection. Elles ne sont pas spécifiques aux listes.

Parcours d'une liste (1/3)

digitalent FORMATION

- Iterator
  - Permet de parcourir une collection en récupérant les éléments successivement
  - Méthode de parcours homogène pour tous les types de collection
  - Garantit la cohérence de la collection parcourue
    - Pas de modifications externes pendant le parcours

Parcours d'une liste (2/3)

digitalent FORMATION

- Récupération d'un Iterator sur une collection donnée :
  - méthode maCollection.iterator()

- Iterator contient 3 méthodes :
  - boolean hasNext() : renvoie true s'il reste des éléments à parcourir dans la liste.
  - Object next() : renvoie le prochain objet stocké dans la liste.
  - void remove() : supprime l'élément en cours de la liste.

Parcours d'une liste (3/3)

digitalent FORMATION

- Exemple

```
List 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);

Iterator iterator = list.iterator();
while (iterator.hasNext()) {
User myUser = (User) iterator.next();
System.out.println(myUser);
}
```

La classe d'implémentation de l'interface Iterator n'est généralement pas connue.

Gestion des doublons

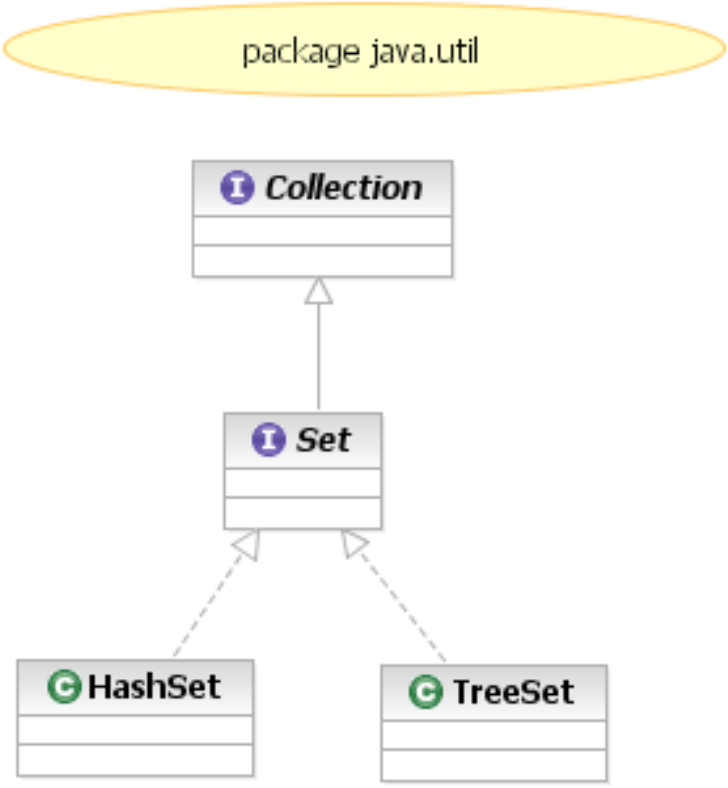
- Les listes acceptent les doublons
  - Ils sont positionnés à un index différent
  - La liste stocke plusieurs références vers le même objet

```
List list = new ArrayList();
User u1 = new User("jean", "dupont");
list.add(u1);
list.add(u1); //deux références pointent vers le même objet
```

Sets

Les sets

- Les sets sont des collections
  - Implémentent l'interface java.util.Set qui hérite de l'interface Collection
- Les sets ne sont pas indexés



Les sets les plus courants

- HashSet
  - Implémentation de base de l'interface Set
- TreeSet
  - Les éléments sont triés

Ajout d'éléments

- Les éléments sont ajoutés avec la méthode 'add(...)'
  - Les éléments ne sont pas indexés.
  - Tout type d'objet peut être inséré dans un set.

```
Set set = new HashSet();
set.add("le");
set.add("petit");
set.add("chat");
```

Suppression d'élément

digitalent FORMATION

- Méthode remove(...)
  - Suppression de l'objet passé en paramètre

```
Set set = new HashSet();
User u1 = new User("jean", "dupont");
set.add(u1);
User u2 = new User("jean", "durand");
set.add(u2);
User u3 = new User("jean", "martin");
set.add(u3);

set.remove(u3);           // suppression de la référence vers u3
```

Parcours d'un set

- Utilisation d'un Iterator pour le parcours

```
Set set = new HashSet();
User u1 = new User("jean", "dupont");
set.add(u1);
User u2 = new User("jean", "durand");
set.add(u2);
User u3 = new User("jean", "martin");
set.add(u3);

Iterator iterator = set.iterator();
while (iterator.hasNext()) {
User myUser = (User) iterator.next();
System.out.println(myUser);
}
```

Gestion des doublons

digitalent FORMATION

- Les Sets n'acceptent pas les doublons
  - Si on ajoute un élément plusieurs fois, il ne sera présent qu'une seule fois dans le set.

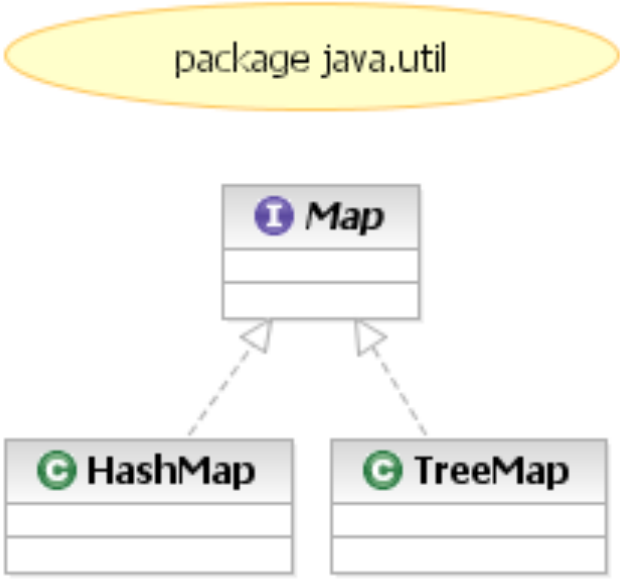
```
Set set = new HashSet();
User u1 = new User("jean", "dupont");
set.add(u1);
set.add(u1); //sans effet
```

La méthode add(…) renvoie un booléen. Elle renvoie false si l'élément était déjà présent.

Maps

Les Maps

- Associations clé-valeur
  - Chaque élément stocké est associé à une clé
  - Conceptuellement, une Map est un ensemble d'éléments.
  - Pour des raisons techniques, l'interface java.util.Map n'hérite pas de java.util.Collection



Il existe également la classe Hashtable (généralement déconseillée)

Les listes les plus courantes

digitalent FORMATION

- HashMap
  - Implémentation de base de l'interface Map
- TreeMap
  - Les éléments sont triés

Ajout et récupération d'éléments

- Les éléments sont ajoutés avec la méthode put(...)'
  - put(Object key, Object value)
- Ils sont récupérés avec la méthode get(...)
  - get(Object key)
  - Nécessité de caster l'élément retourné

```
User u1 = new User("jean", "dupont");
User u2 = new User("jean", "durand");

Map map = new HashMap();
// association d'une clé à chaque élément
map.put(new Integer(1), u1);
map.put(new Integer(2), u2);

// récupération en fonction de la clé
User u1bis = (User) map.get(new Integer(1));
```

Suppression d'élément

- Méthode remove(...)
  - remove(Object key)

```
User u1 = new User("jean", "dupont");
User u2 = new User("jean", "durand");

Map map = new HashMap();
// association d'une clé à chaque élément
map.put(new Integer(1), u1);
map.put(new Integer(2), u2);

// suppression
map.remove(new Integer(1));
```

Attention à ne pas passer l'objet mais la clé en paramètre

Parcours d'une Map (1/2)

digitalent FORMATION

- Problématique particulière
  - S'agit-il de parcourir les clés ou les valeurs ?
  - Méthode Map.keySet() : permet de récupérer un set contenant l'ensemble des clés.
  - Méthode Map.values() : permet de récupérer une collection contenant l'ensemble des valeurs.

```
Map map = new HashMap();
// …
Iterator keysIte = map.keySet().iterator();
// …
Iterator valuesIte = map.values().iterator();
// …
```

Parcours d'une Map (2/2)

digitalent FORMATION

- Exemple complet :

```
Map map = new HashMap();
User u1 = new User("jean", "dupont");
User u2 = new User("jean", "durand");

map.put(new Integer(1), u1);
map.put(new Integer(2), u2);

Iterator valuesIte = map.values().iterator();
while (valuesIte.hasNext()) {
  User u = (User) valuesIte.next();
  // …
}
```

Gestion des doublons

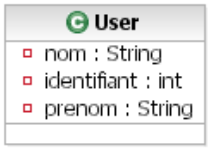
- Les maps acceptent les doublons
  - Un élément peut être présent plusieurs fois s'il est référencé par des clés différentes.
  - Rajout d'un élément avec une clé existant déjà :

    Si la clé était déjà présente dans la map, sa précédente valeur est écrasée.

Tri d’éléments

Problématique

- 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 : lors d'un parcours de collection, on veut que les objets User arrivent triés par leur nom.
      S'ils ont le même nom, le deuxième critère est le prénom.



Solution 1 : Comparable

- Première solution :
  - Implémentation de l'interface java.lang.Comparable

```
public class User implements Comparable {
//…


 public int compareTo(Object o) {
  User user2 = (User) o;
  int resultComparaisonNoms = this.nom.compareTo(user2.getNom());
  if (resultComparaisonNoms != 0) {
   // renvoie un nombre négatif si l'objet en cours a un nom
   //situé avant le nom de l'objet passé en paramètre (ordre alphabétique)
   // renvoie un nombre positif si l'objet en cours a un nom
   //situé après le nom de l'objet passé en paramètre
   return resultComparaisonNoms;
  } else {
  // même test pour les prénoms
  }
 }
}
```

Solution 2 : Comparator

digitalent FORMATION

- Deuxième solution :
  - Création d'un Comparator

```
import java.util.Comparator;

public class UserComparator implements Comparator {

  public int compare(Object o1, Object o2) {
    User u1 = (User) o1;
    User u2 = (User) o2;

    int result = u1.getNom().compareTo(u2.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.

Classes utilitaires

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 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
```

Java 5+ - 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.
- Si un mauvais type est employé lors de la récupération d'un élément de la liste, l'erreur n'est pas détectée en phase de compilation.

```
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.
- Il ne faut pas faire de transtypage (cast) lors de la récupération d'un élément.
  - Une éventuelle erreur de type est détectée en phase de compilation.

```
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();

}
```