Collections : ensembles

Pratique de la programmation orientée-objet

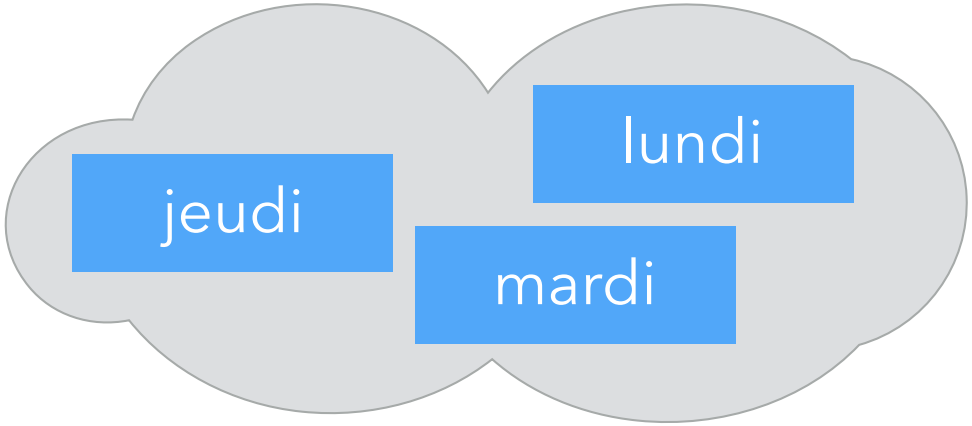
Michel Schinz – 2017-03-27

Collections étudiées

Listes
(*lists*)

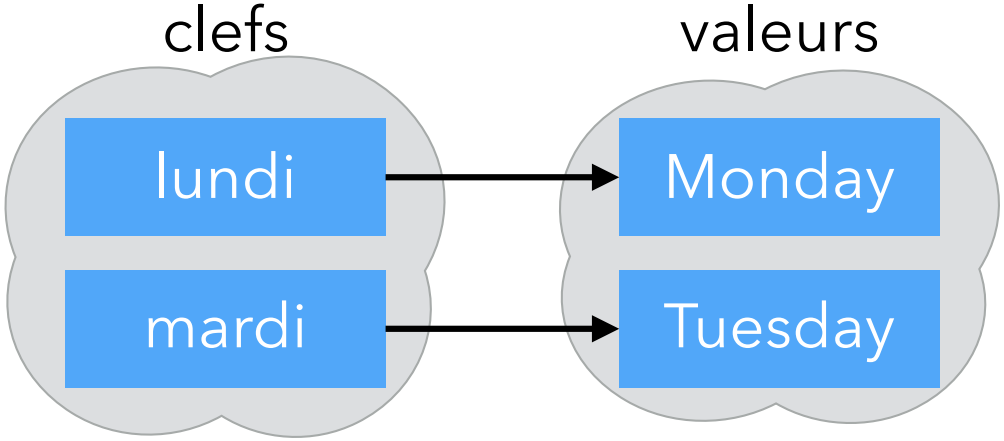
ordre : oui
doublons : oui

Ensembles
(*sets*)



ordre : non
doublons : non

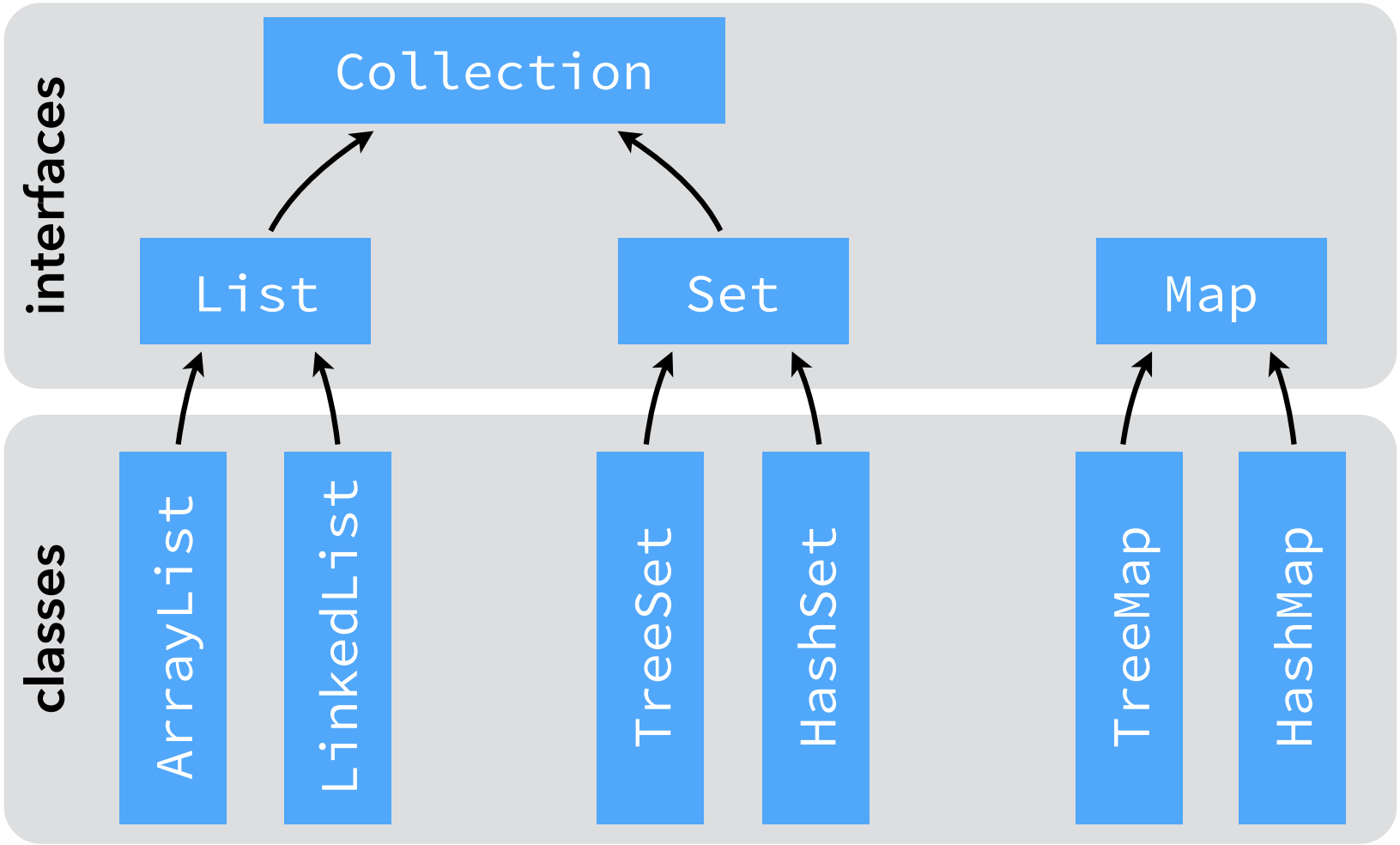
Tables
associatives
(*maps*)



association
clefs/valeurs

Collections de l'API Java

(vue *très* partielle et simplifiée du paquetage `java.util`)



\+ classes utilitaires `Collections` et `Arrays`

Mises en œuvre de `Set`

Opération	`TreeSet`	`HashSet`
ajout (`add`), test (`contains`), etc.	O(log *n*)	O(1)

Différence importante :

- `TreeSet` stocke ses éléments de manière ordonnée,
- `HashSet` stocke ses éléments dans un ordre quelconque.

Egalité, comparaison, hachage

TreeSet

`TreeSet` offre un constructeur prenant un comparateur :

```
public class TreeSet<E> {
  public TreeSet(Comparator<E> comparator);
  // … autres constructeurs et méthodes
}
```

Pour mémoire, l'interface `Comparator` est :

```
@FunctionalInterface
public interface Comparator<T> {
  public int compare(T v1, T v2);
}
```

Comparable

L'interface `java.lang.Comparable` est implémentée par les classes dont les instances savent se comparer entre elles :

```
public interface Comparable<T> {
  int compareTo(T that);
}
```

Ne pas confondre avec `java.util.Comparator` :

```
public interface Comparator<T> {
  int compare(T o1, T o2);
}
```

hashCode

Contrairement à la notion d'ordre, le hachage est prédéfini en Java, et donc disponible pour tous les objets :

```
public class Object {
   // … autres méthodes
   int hashCode();
}
```

La mise en œuvre par défaut utilise l'identité du récepteur.

hashCode / equals

Les méthodes `hashCode` et `equals` doivent toujours être **compatibles**, c-à-d que la propriété suivante doit être vraie pour toute paire d'objets o_1 et o_2 :

o_1`.equals(`o_2`)` ⇒ o_1`.hashCode() == `o_2`.hashCode()`

(Deux objets égaux ont la même valeur de hachage.)

Attention, l'implication inverse n'est pas vraie (en général) !

o_1`.hashCode() == `o_2`.hashCode()` ~~⇒~~ o_1`.equals(`o_2`)`

(Deux objets ayant la même valeur de hachage ne sont pas forcément égaux.)

Ensembles / tables assoc.

Ensembles et tables associatives sont très similaires :

`Set<E>` ≈ `Map<E, Void>`

`Map<K, V>` ≈ `Set<Map.Entry<K,V>>`

Conséquences :

1. tout ce qui a été dit sur les éléments des ensembles s'applique aux clefs des tables associatives,
2. les uns peuvent être mis en œuvre au moyen des autres (en pratique en Java : `TreeSet` mis en œuvre au moyen de `TreeMap`, idem pour `HashSet`/`HashMap`).

Eléments d'une collection

Collection	Contraintes
`List`	*aucune*
`Set`	`equals`
`TreeSet`	`Comparable.compareTo` ou `Comparator.compare`
`HashSet`	`hashCode` + `equals`

Les mises en œuvre de `Map` (`TreeMap` et `HashMap`) placent les mêmes contraintes que celles de `Set`, mais uniquement sur les *clefs* de la table.