Synthèse sur les structures de données

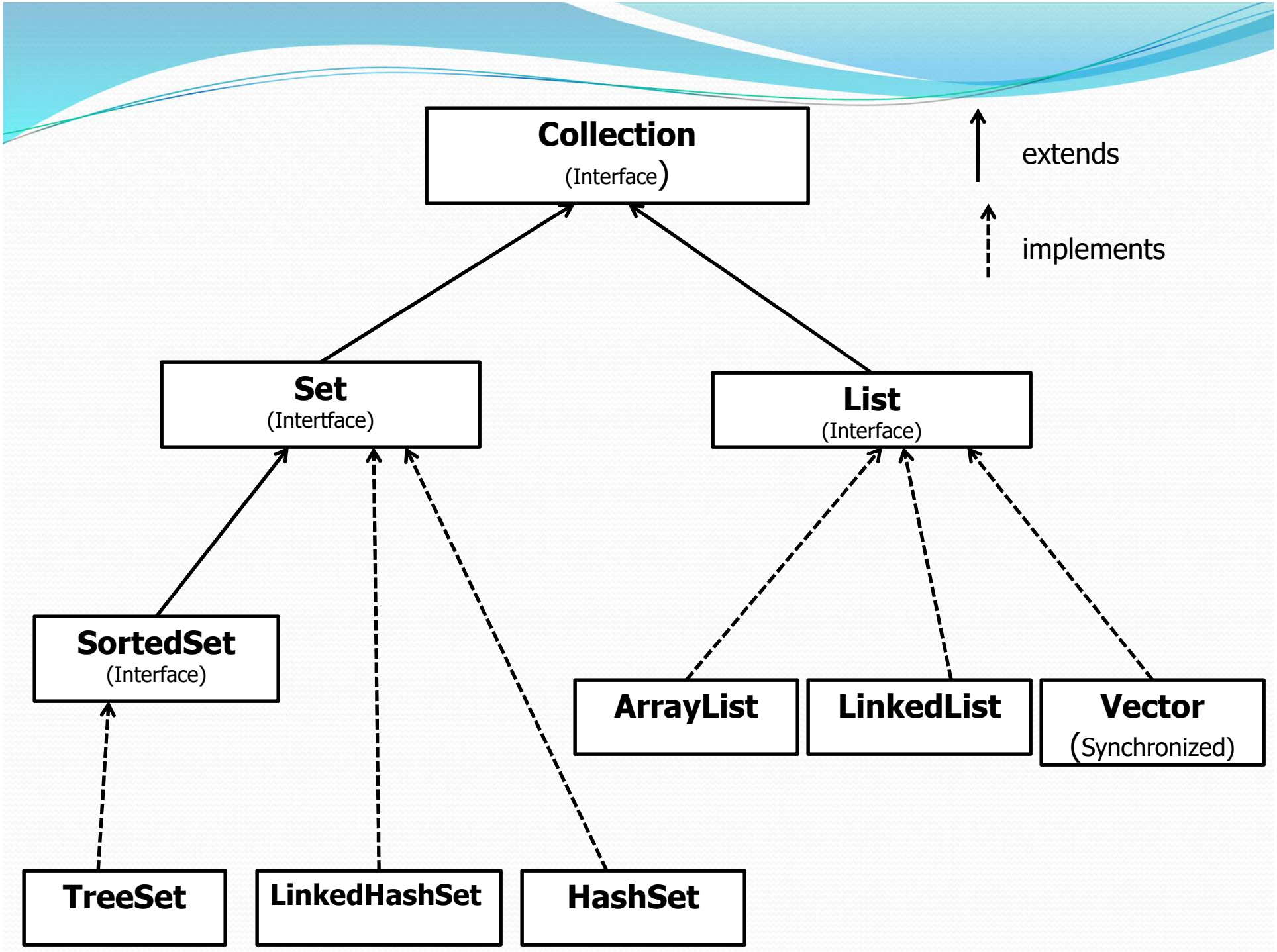
Les collections de Java
(Collections Framework)

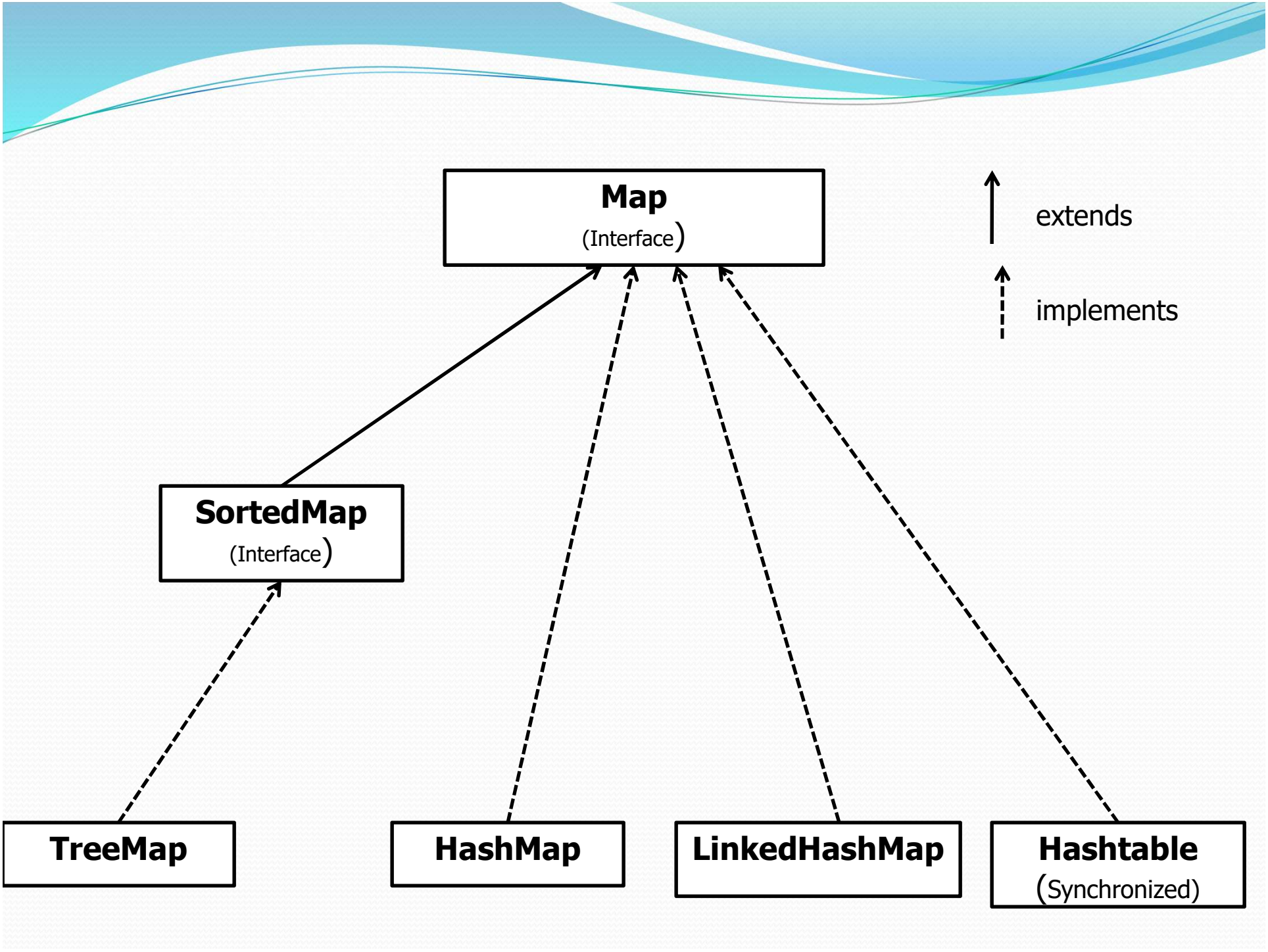
Les interfaces

- **List**: Collection ordonnée où les doublons sont autorisés avec un système d'index implémenté pour permettre un accès rapide aux données

- **Set**: Collection sans doublons.

- **Map**: utilise la paire *clé*/*valeur* pour chaque élément de la collection


Collection
(Interface)
extends
implements
Set
(Intertface)
List
(Interface)
SortedSet
(Interface)
ArrayList
LinkedList
Vector
(Synchronized)
TreeSet
LinkedHashSet
HashSet


Map
(Interface)
extends
implements
SortedMap
(Interface)
TreeMap
HashMap
LinkedHashMap
Hashtable
(Synchronized)

Les implantations de List

- **ArrayList vs LinkedList**
- **ArrayList**: plus efficace si accès à l'aide d'un index.
- **LinkedList**: plus efficace si beaucoup d'insertions

Les implantations de Set

- **HashSet**: si on a des données sans doublons.
- **TreeSet**: si les données doivent en plus être triées. Les insertions, les suppressions et les recherches se font en log N (N taille de l'ensemble)
- **LinkedHashSet**: préserve l'ordre d'insertion des éléments

Qu'est-ce qu'un doublon?

- Le *HashSet* utilise la valeur *hashCode* de l'objet pour déterminer sa place.

- Deux objets avec des *hashCode* différents ne peuvent être égaux.

- Si deux Objets ont la même valeur de *hashCode*, le *HashSet* exécute une méthode *equals* sur l'objet.

  Si les objets sont en plus égaux, on parle alors de doublons.

Règles de Java pour hashCode() et equals()

- Deux objets égaux doivent avoir la même valeur de *hashCode*
- L'égalité entre deux objets est une relation d'équivalence (réflexive, symétrique et transitive). Elle doit toujours retourner la même valeur si les objets comparés ne sont pas modifiés. De plus x.equals(null) doit retouner la valeur faux.
- Deux objets avec la même valeur de *hashCode*, ne sont pas nécessairement égaux

Règles de Java pour hashCode() et equals()…

- Donc si on redéfinit *equals()*, il faut redéfinir *hashCode()*

- Par défaut, *equals()*, utilise l'opérateur == pour vérifier si deux références pointent sur le même objet. Ce qui ne sera pas le cas en général , si nous ne redéfinissons pas la méthode *equals()*.

Les implantations de Map

- HashTable/HashMap: Un élément est un couple *clé/valeur*. Les clés son uniques.

- LinkedHashMap: implantation sous forme de liste chaînées. L'ordre d'insertion des clés est préservée.

- TreeMap: Les éléments sont triés par ordre des clés.

La recherche(containsKey(), l'ajout(put) , la lecture (get) et la suppression d'un élément se font en un temps log N (N étant le nombre d'entrées de la table)

tri des collections List

- La méthode ***Collections.sort(List l).*** Dans ce cas la méthode compareTo() détermine l'ordre des éléments de la liste. La classe de la liste DOIT implémenter l'interface Comparable

- Que se passe-t-il si on veut trier selon plusieurs champs? Exple nom, date,.....

tri des collections (autre méthode)

- **Collections.sort(List l, Comparator c)**

On définit une classe qui implémente Comparator.

Exemple:

```
class  TitreCompare  implements Comparator< Document >
   public int compare (Document un, Document deux)  {
return un.getTitre().compareTo(deux.getTitre());
```

Autres opérations sur les collections

- binarySearch
- reverse
- shuffle
- copy
- min, max
- addAll
- frequency
- disjoint
- swap