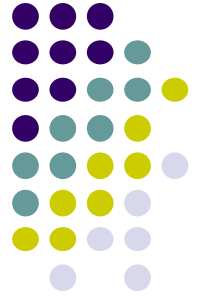


Les tables associatives (Maps)



Licence informatique

POO -Avancée

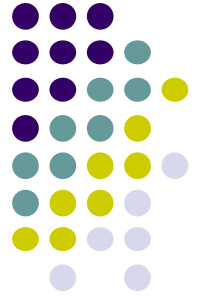
Françoise Greffier
(+ *Pierre-Alain Masson*)





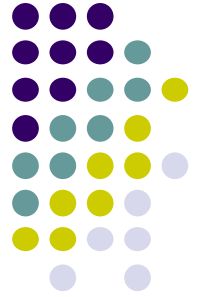
Sommaire

- 1) Qu'est-ce qu'une table associative ?
- 2) Implémentation d'une table associative
- 3) Manipulation d'une table associative
- 4) Un exemple de TreeMap



- 1) **Qu'est-ce qu'une table associative ?**
- 2) Implémentation d'une table associative
- 3) Manipulation d'une table associative
Un exemple de TreeMap
- 4) Notion de vue

Couples (clé , valeur)



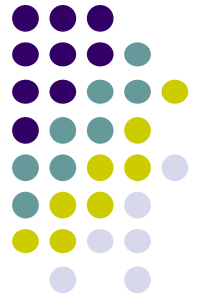
Une table associative est une collection permettant de contenir des couples (clé, valeur associée à la clé).

Les tables sont génériques, elles sont définies à l'aide de deux types :
un pour la clé (nommé généralement K)
et un autre pour la valeur (nommé généralement V).

Exemples de tables :

- *Annuaire* (Nom Prénom, Numéros de téléphone)
- *Dictionnaire* (Mot, Définitions)
- *Traductions* (Mot en français, Mots équivalents en anglais)

Couples (clé , valeur)



Exemples de tables :

- *Annuaire* (Nom Prénom, Numéros de téléphone)
- *Dictionnaire* (Mot, Définitions)
- *Traductions* (Mot en français, Mots équivalents en anglais)

- Une première collection K : Clé

Dans une table, deux clés ne **peuvent pas** être égales au sens de equals.

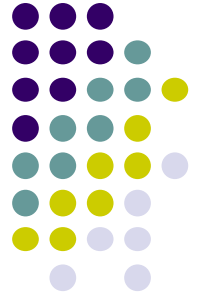
- Une deuxième collection V : Valeur (ex : List).

Par contre, une valeur peut apparaître plusieurs fois associée à des clés différentes.

- Une troisième collection : Couple (clé, valeur)

Un couple (clé, valeur) est de type **Map.Entry** <K,V>

Map.Entry<K,V> est une interface



- 1) Qu'est-ce qu'une table associative ?
- 2) **Implémentation d'une table associative**
- 3) Manipulation d'une table associative
Un exemple de TreeMap
- 4) Notion de vue

Implémentation d'une table



Une table est utile pour trouver rapidement une valeur associée à une clé.

Comme pour les ensembles, l'intérêt d'une table est de pouvoir rapidement retrouver une valeur liée à une clé donnée.

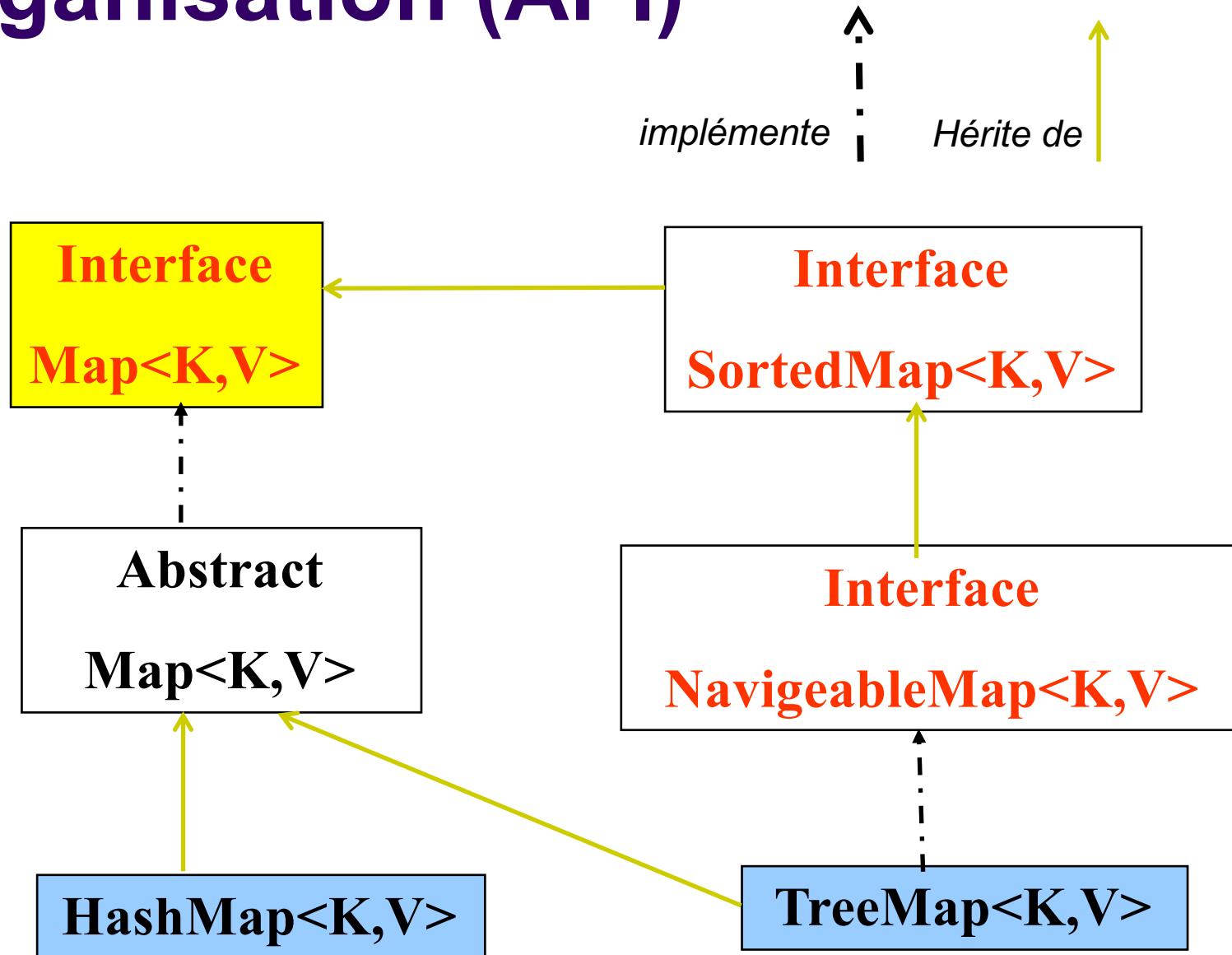
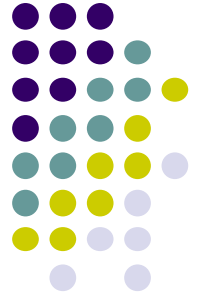
Chaque clé est unique (**pas de doublon**).

Rappel : un ensemble peut être implanté dans un `TreeSet` ou bien dans un `HashSet`.

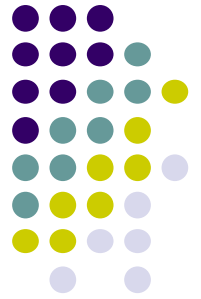
On retrouve les deux types d'organisation rencontrés pour les **ensembles** afin d'organiser les clés :

- Ensemble de clés basé sur une fonction de hachage => table de hachage : **HashMap<K, V>**
- Ensemble de clés basé sur un arbre binaire => **TreeMap<K, V>** ₇

Organisation (API)



Deux possibilités d'implémenter une table



Deux classes concrètes

HashMap<K,V>

Les clés sont organisées selon une table de hachage. Par conséquent, l'accès à une valeur a un temps (quasi) constant.

TreeMap<K,V>.

Les clés sont rangées dans un arbre binaire de recherche.

- Elles sont donc **ordonnées** (ex : ordre alphabétique sur les personnes)
- **L'accès à une valeur est en Log(N).**

La comparaison de clés utilise l'ordre sur K ou un comparateur associé à la création de la table.

Interface **Map**<K,V>



```
Public interface Map<K, V> {  
    boolean isEmpty();  
    int size();  
    void put(K key, V value);  
    void remove(Object key);  
    V get(K key);  
    boolean containsKey(Object key);  
    Set<K> keySet();  
    Collection<V> values();  
    ...  
}
```

Interface **Map**<K,V>



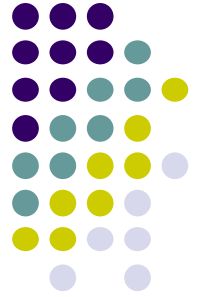
Les classes `HashMap` et `TreeMap` n'implémentent pas l'interface `Collection<E>`, elles implémentent l'interface `Map<K,V>`. En effet, les éléments d'une table ne sont plus des objets pris individuellement mais des couples (cle,valeur) appelés aussi paires.

Interface Map<K,V>

```
V put (K cle, V valeur);  
// si la cle existe déjà, la nouvelle paire écrase l'ancienne.  
// put donne en retour : null ou ancienne valeur
```

```
V get (Object cle);  
// retourne la valeur associée à une clé donnée.  
// Valeur retournée est égale à null, si la clé n'est pas présente.
```

Interface **Map**<K,V>



```
boolean containsKey (Object cle);  
boolean containsValue (Object valeur);
```

```
V remove (Object cle)
```

```
// permet de supprimer un couple (clé,valeur).
```

```
// Valeur retournée est égale à la valeur associée à la clé, ou bien
```

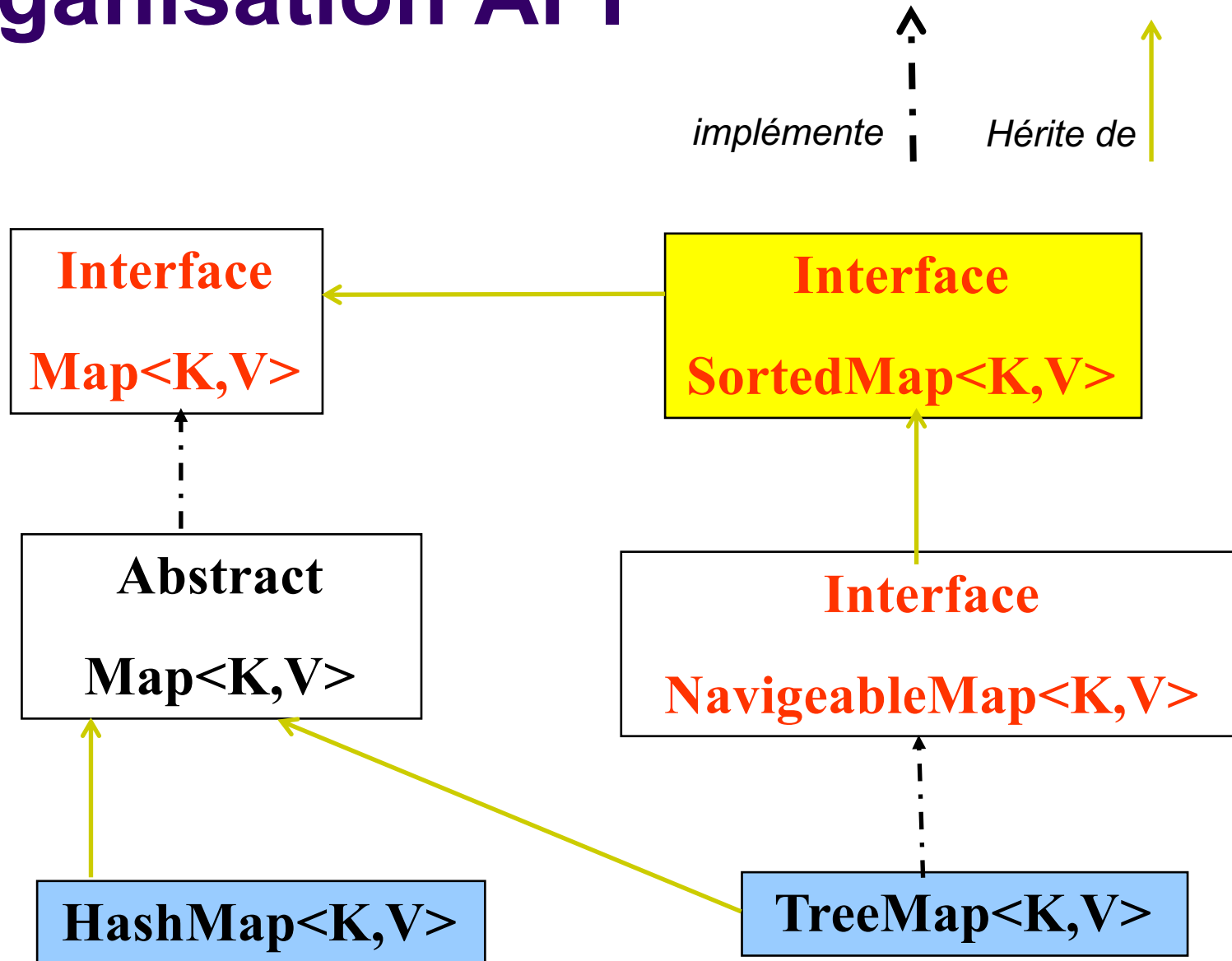
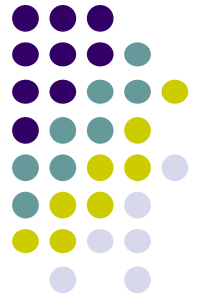
```
// null si la clé n'était pas présente dans la table.
```

```
int size( );
```

```
boolean isEmpty( );
```

Etc.

Organisation API



Interface **SortedMap**<K,V>



- L'interface `java.util.SortedMap<K,V>` (depuis Java 1.2) définit les fonctionnalités d'une Map **dont les clés sont triées (TreeMap)**. Elle hérite de l'interface `Map<K,V>` et fournit un **ordre total sur les clés**.
- L'ordre dans les clés respecte l'ordre en implémentant l'interface `Comparable<K>` ou en fournissant un `Comparator<K>` à la création de l'instance de la collection.
- Les clés doivent aussi avoir une implémentation de la méthode `equals()` qui soit en accord avec cette solution car elle est invoquée pour déterminer si la clé est déjà dans la collection.

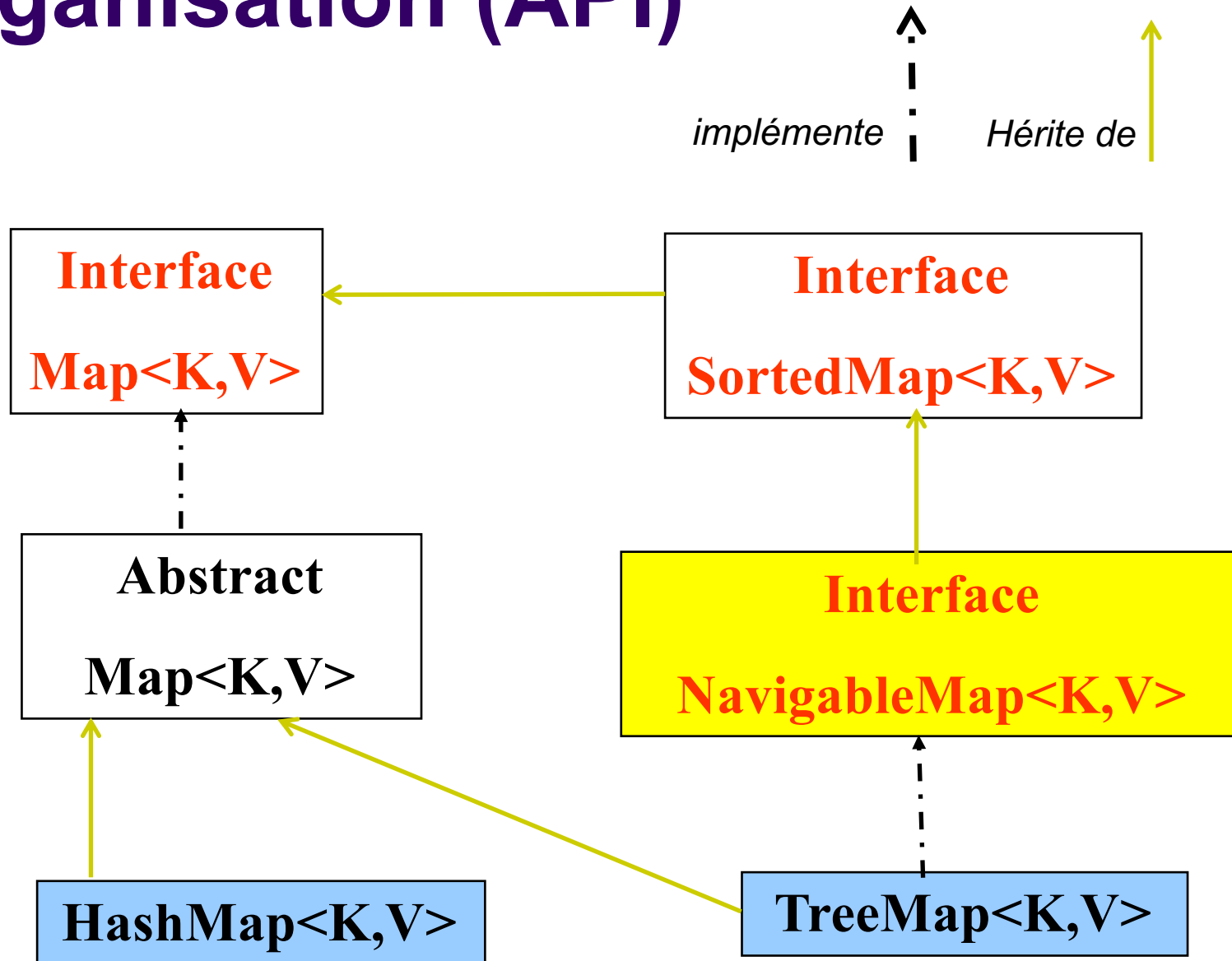
Interface `SortedMap<K,V>`



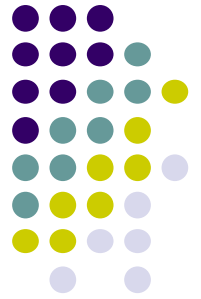
On retrouve des méthodes conçues suivant la même logique que `SortedSet`.

- `K firstKey()` et `K lastKey()` : retournent respectivement la plus petite clé et la plus grande clé.
- `SortedMap<K,V> headMap(K toKey)` : retourne une vue de la table maître, contenant les clés `< toKey`
- `SortedMap<K,V> tailMap(K fromKey)` retourne une vue de la table maître, contenant les clés `>= fromKey`
- `SortedMap<K,V> subMap(K fromKey, K toKey)` retourne une vue sur la table maître, de la clé `fromKey` incluse, à la clé `toKey` exclue.

Organisation (API)



Interface **NavigableMap**<K,V>



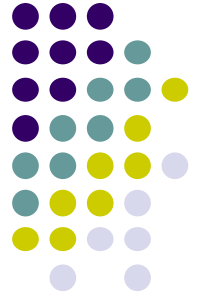
Ajoute des fonctionnalités à un `SortedMap<K,V>`.

Exemples :

`NavigableMap<K,V> headMap(K sup, boolean inclus)`
retourne une vue de la table dont les clés sont plus petites que la borne `sup` passée en paramètre. Le fait que l'inégalité soit stricte ou non est réglé par la valeur du booléen `inclus`.

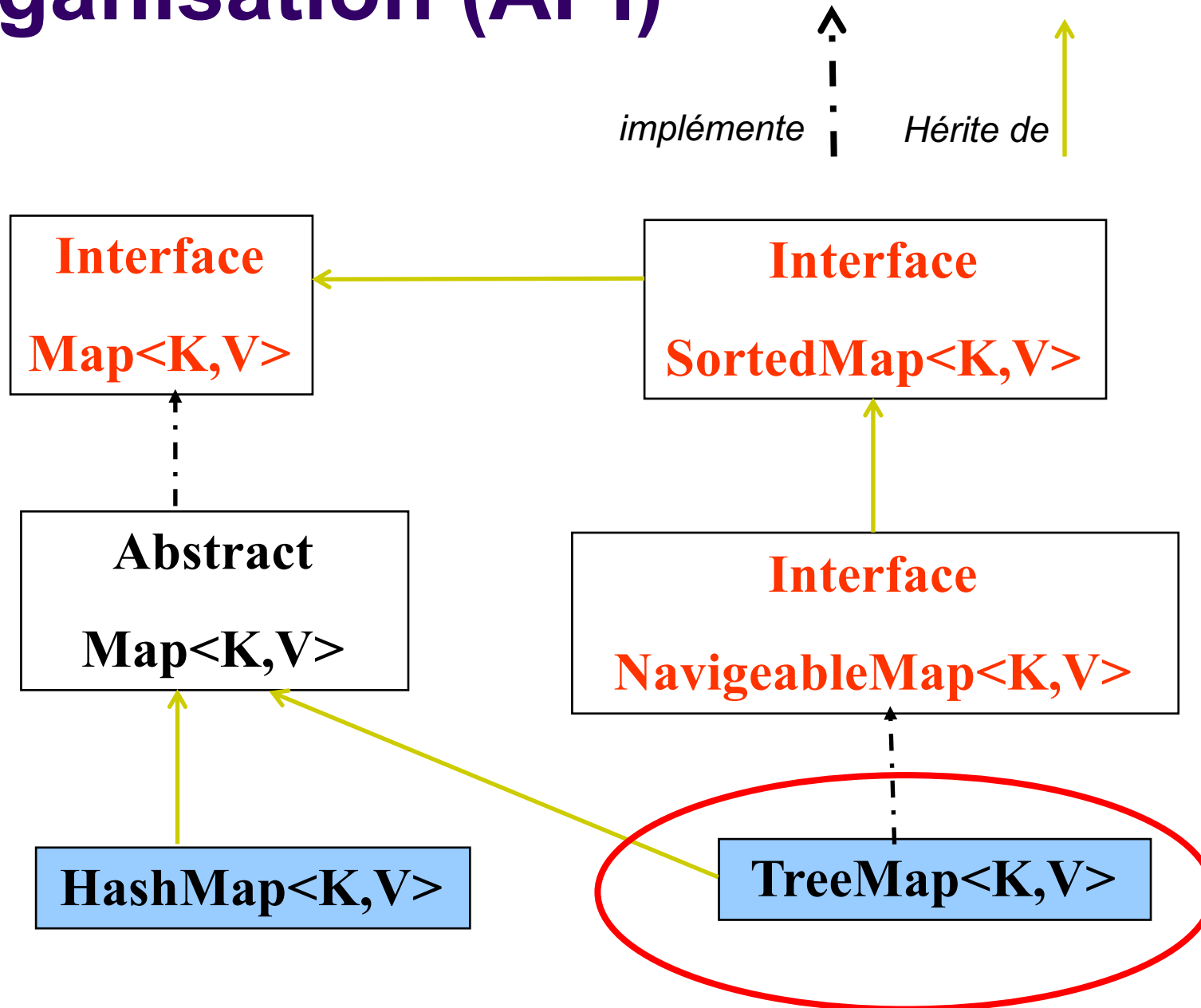
`Map.entry<K,V> firstEntry()`
retourne la « paire » (couple) de plus petite clé de la table, ou `null` si la table est vide

`K higherKey(K key)` : retourne la plus petite clé supérieure ou égale à `K`

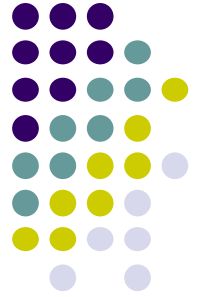


- 1) Qu'est-ce qu'une table associative ?
- 2) Implémentation d'une table associative
- 3) Manipulation d'une table associative
Un exemple de TreeMap
- 4) Notion de vue

Organisation (API)



TreeMap : constructeurs

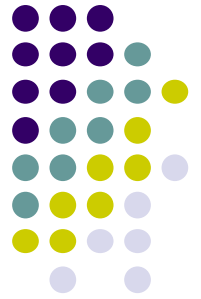


`TreeMap()` : constructeur par défaut qui crée une collection vide utilisant l'ordre naturel des clés des éléments (donné par `compareTo` sur l'ensemble `K`).

`TreeMap(Comparator<? super K> comparator)` : créer une instance vide qui utilisera le `Comparator` fourni en paramètre pour déterminer l'ordre des clés des éléments.

Exemple TreeMap

TreeMap-annuaire



Créer une table de couples (Personne, N° de tel), servant à représenter un annuaire dans lequel, chaque personne est associée à une liste de numéros de téléphone.

```
import java.util.TreeMap;

public class TableTelPersonne {
    private TreeMap <Personne,ListNo> annuaire;

    public TableTel( ) {
        annuaire = new TreeMap<Personne,ListNo>();
    }
}
```

Le constructeur crée une table vide.

Ajout d'une liste de numéros



table(Personne, Liste de numéros)

Ensuite on ajoute des couples dans la table à l'aide de la méthode nommée ici `ajouter`.

```
public void ajouter (Personne p, ListNo liste) {  
    if (! annuaire.containsKey (p))  
        annuaire.put (p, liste);  
}
```

// Méthode put(cle,valeur) :

// si la cle existe déjà, la nouvelle paire écrase l'ancienne.

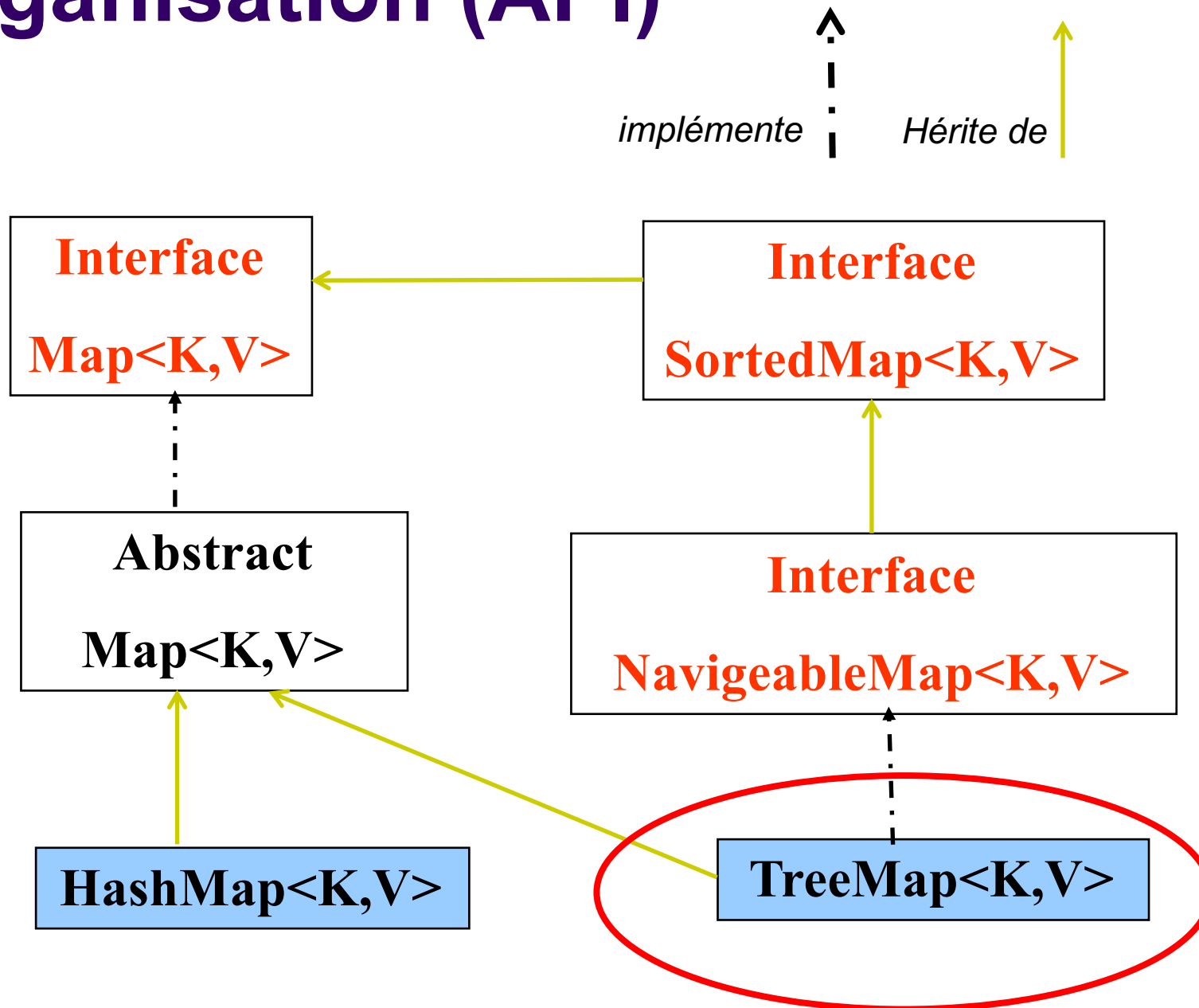
// put donne en retour : null ou ancienne cle

Ajout d'un numéro

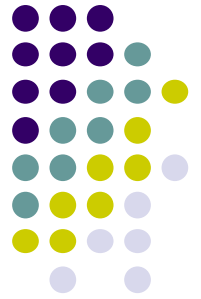


```
public void ajouter (Personne p, String n) {  
    ListNo li=annuaire.get(p);  
    if (li!=null) { //la liste existe  
        li.ajouterNo(n);  
    }  
    else { //le couple est créé  
        ListNo l=new ListNo();  
        l.ajouterNo(n);  
        annuaire.put(p,l);  
    }  
}
```

Organisation (API)

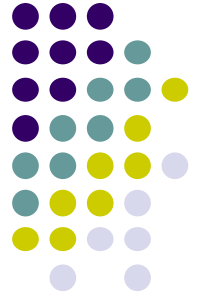


Afficher les personnes de l'annuaire par ordre alphabétique



```
public void afficherPersonnes () {  
    Set<Personne> cles=annuaire.keySet () ;  
    System.out.println(cles) ;  
}
```

Les personnes de l'annuaire sont affichés par ordre alphabétique sur les noms et les prénoms (voir CompareTo dans la classe Personne).



- 1) Qu'est-ce qu'une table associative ?
- 2) Implémentation d'une table associative
- 3) Manipulation d'une table associative
Un exemple de TreeMap
- 4) **Notion de vue**

Parcours d'une table (Notion de vue)



Une table ne dispose pas d'itérateur.

On utilise une méthode nommée **entrySet()** pour « voir » une table comme un **ensemble** (Set) de paires (clé,valeur).

```
Set<Map.Entry<Personne,ListNo>> paires =annuaire.entrySet();  
// paires est un ensemble de paires (couples (clé, valeur))
```

L'ensemble **paires** est une « **VUE** » de la table annuaire.



L'interface **Map.Entry<K,V>**

Une table ne dispose pas d'itérateur.

Pour parcourir une table et traiter les couples (ou entrées) de la table, on utilise les méthodes données par l'interface **Map.Entry<K,V>**

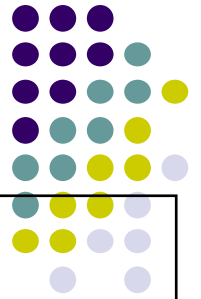
L'interface **Map.Entry<K,V>** modélise les couples (clé, valeur).

La méthode nommée **Map.entrySet()** renvoie une référence sur un « ensemble (un Set) de Entry », autrement dit sur un set de couples (clé, valeur)

L'interface **Map.Entry<K,V>** contient ces deux méthodes :

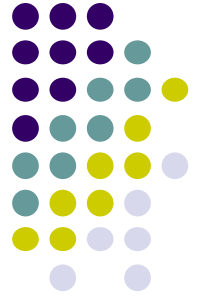
- **K getKey();** //retourne la clé du couple
- **V getValue();** //retourne la valeur du couple



Parcours d'une table - exemple

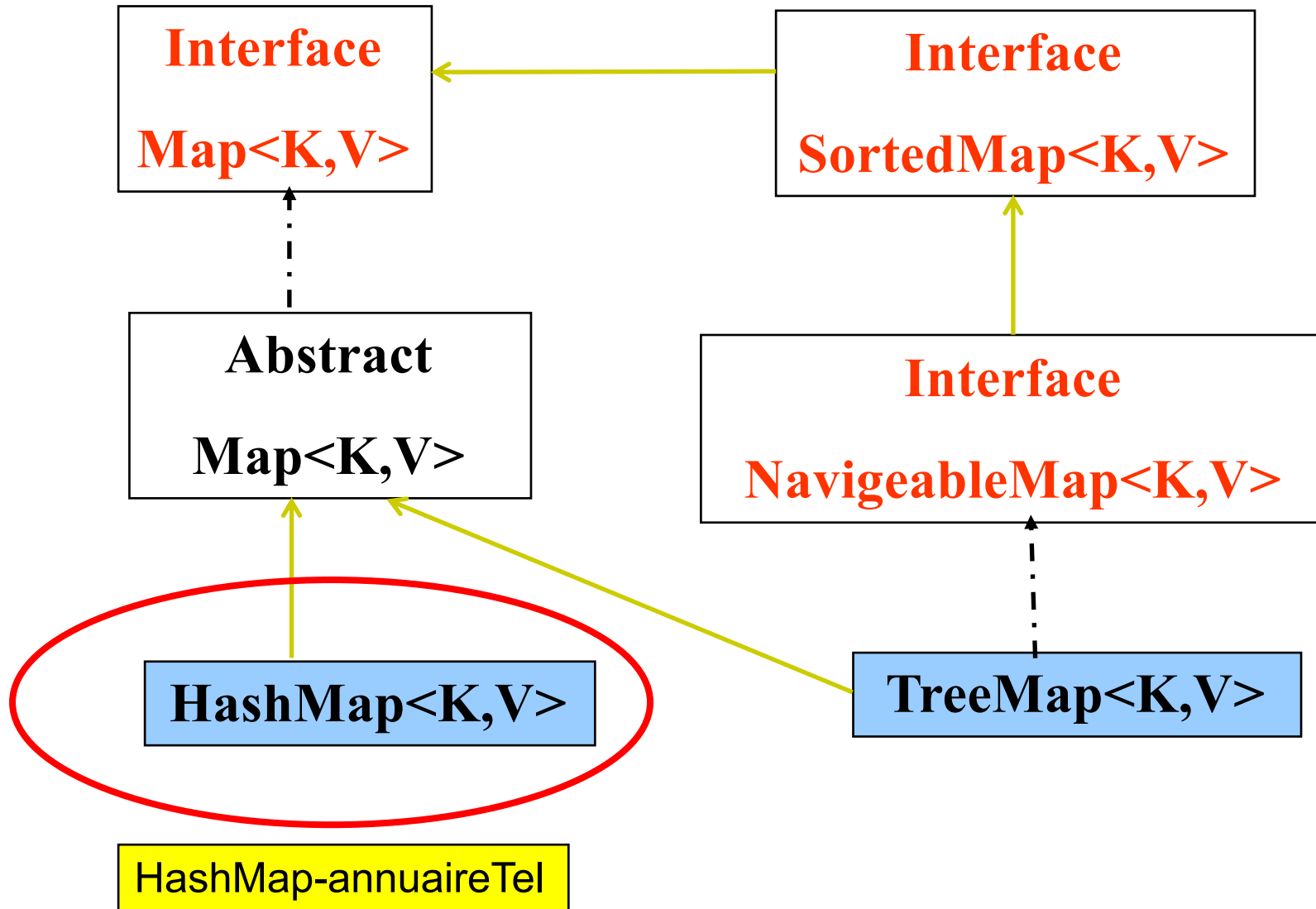


```
public void afficher() {  
    Set<Map.Entry<Personne,ListNo>> paires =  
                                                annuaire.entrySet();  
  
    //paires est un ensemble (Set)  
    //de couples (personne, liste n°)  
  
    Iterator <Map.Entry<Personne,ListNo>> iter =  
    paires.iterator();  
    //iterateur sur les couples  
  
    while (iter.hasNext()) {  
        Map.Entry<Personne,ListNo> paire = iter.next();  
        Personne p = paire.getKey();  
        ListNo liste = paire.getValue();  
        System.out.println (p+" : "+liste);  
    }  
}
```

Organisation (API)



implémente  *Hérite de* 



Autres vues pour une table



Vue de l'ensemble des clés :

Méthode **keySet()**

```
HashMap table; ou TreeMap Table;
```

...

```
Set<K> cles=table.keySet();
```

Vue de la collection des valeurs :

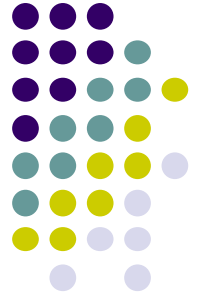
Méthode **values()**

```
HashMap table; ou TreeMap Table;
```

...

```
Collection<V> valeurs=table.values();
```

On obtient une collection et non pas un ensemble car les valeurs peuvent renfermer des doublons.



Opérations sur les vues

Une vue est une autre façon de « *voir* » la table.

Toutes les opérations associées à une table ne sont pas forcément applicables sur une vue.

Par exemple, dans la vue des couples (de type `Map.Entry <K, V>`) on peut supprimer (`remove`) un couple.

La suppression sera alors validée dans la table.

Par contre, l'ajout n'est pas autorisé dans la vue.

Même chose pour les deux autres vues : clés et valeurs



MERCI
de votre attention