جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Chapitre 7

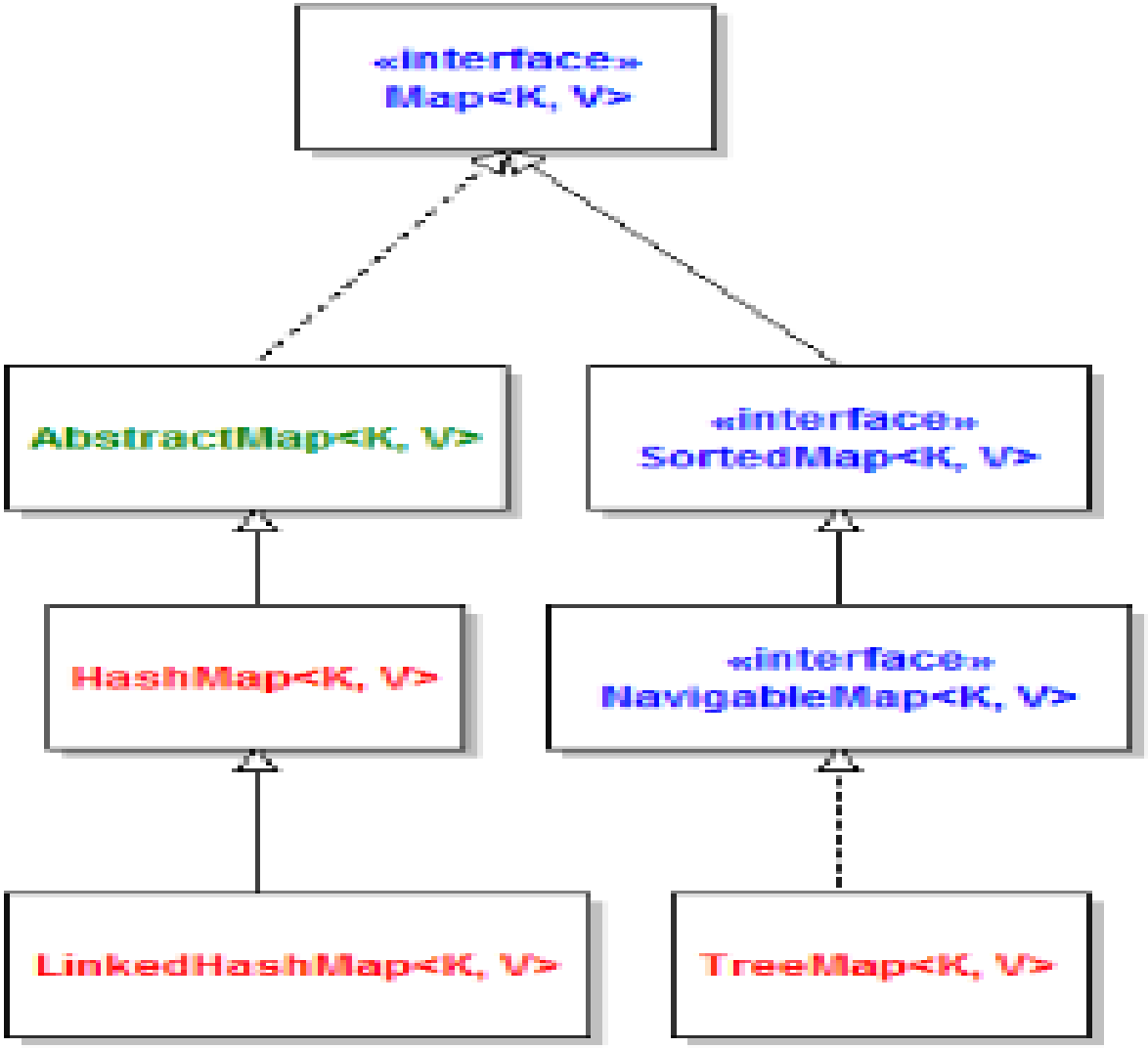
Les Collections en java (part3)

S. BOUKHEDOUMA

USTHB – FEI – département d'Informatique
Laboratoire des Systèmes Informatiques -LSI
sboukhedouma@usthb.dz

L'interface « Map »
Les tables de hachage

Les Collections de type Map



Les Collections – l'interface « Map »

L'interface « Map »

« Map » n'est pas liée à l'interface « Collection »

Une collection de type **Map** est une collection où on considère des éléments **de type (clé, valeur)**

La clé peut être de type (Integer, String, Long …ou autre objet)
Valeur représente un **objet associé à la clé**

Dans une collection de type Map , on n'accepte pas **les doublons des clés (la clé est unique).**
Les objets (Valeur) peuvent se répéter

La **clé** sert à identifier l'élément

Les Collections – l'interface « Map »

L'interface « Map »

« Map » n'est pas liée à l'interface « Collection »

L'interface **Map** (introduite dans java 1.2) remplace la **classe abstraite Dictionnary** de Java 1.1

Organisation des éléments dans une Map

- Les éléments **seront repartis dans plusieurs blocs (buckets)** selon les valeurs de leurs clés (en appliquant une fonction de hachage).

-Lorsqu'on veut ajouter un élément, il sera ajouté dans le bloc adéquat selon la valeur de sa clé (en calculant sa fonction de hachage)

- La recherche d'un élément se fait dans le bloc correspondant à la valeur de la clé

Donc, réduction de l'espace de recherche des éléments

Les Collections – l'interface « Map »

L'interface « Map »

Les méthodes de l'interface « Map »

Méthode	Rôle
void clear()	Supprimer tous les éléments de la collection
boolean containsKey(Object)	Indiquer si la clé est contenue dans la collection
boolean containsValue(Object)	Indiquer si la valeur est contenue dans la collection
Set entrySet()	Renvoyer un ensemble contenant les paires clé/valeur de la collection
Object get(Object)	Renvoyer la valeur associée à la clé fournie en paramètre
boolean isEmpty()	Indiquer si la collection est vide
Set keySet()	Renvoyer un ensemble contenant les clés de la collection
Object put(Object, Object)	Insérer la clé et sa valeur associée fournies en paramètres
void putAll(Map)	Insérer toutes les clés/valeurs de l'objet fourni en paramètre
Collection values()	Renvoyer une collection qui contient toutes les valeurs des éléments
Object remove(Object)	Supprimer l'élément dont la clé est fournie en paramètre
int size()	Renvoyer le nombre d'éléments de la collection

Les Collections – l'interface « Map »

L'interface « Map »

« Map » n'est pas liée à l'interface « Collection »

La méthode **keySet()** permet d'obtenir un **ensemble (Set)** contenant toutes les clés de la Map.

Il est recommandé d'utiliser des **objets immuables comme clés** (String, Integer, Long, …) pour ne pas les modifier.

La méthode **values()** permet d'obtenir une collection contenant toutes les valeurs. La valeur de retour est une **Collection** (pas Set) car il peut y avoir des doublons (dans la partie valeur)

plusieurs clés peuvent être associées à la même valeur.

Les Collections – l'interface « Map »

L'interface « Map »

« Map » n'est pas liée à l'interface « Collection »

Parcours d'une collection de type « Map »

Une collection de type **Map** ne propose pas directement d'Iterator sur ses éléments

la collection peut être parcourue de trois manières

- parcours de l'ensemble des clés
- parcours des valeurs
- parcours d'un ensemble de paires clé/valeur

Les Collections – l'interface « Map »

Les implémentations de l'interface **« Map »** (*le lien implements*)

La classe HashTable

représente une table d'éléments associatifs (clé, valeur)
d'une taille variable et non trié

public class HashTable implements Map {…
//implémentation de toutes les méthodes de « Map »}

Les constructeurs de « HashTable »

Plusieurs constructeurs sont fournis

- sans paramètre
- avec un paramètre de type Map
- avec un paramètre (capacité initiale)
- avec deux paramètres (capacité initiale, facteur de charge)

Les Collections – l'interface « Map »

La classe « HashTable »

- La classe HashTable est **Thread-safe**

- Ses méthodes sont **synchronized** (gestion des accès concurrents)

- Une HashTable n'autorise pas **la valeur « null »** pour la clé ou pour la valeur

Remarques

Eviter de donner une **capacité initiale trop petite** ou un **facteur de charge trop petit**

La création d'une nouvelle table (plus grande) nécessite de réorganiser les blocs des éléments, ce qui est **coûteux en temps d'exécution**

Les Collections – l'interface « Map »

La classe « HashMap »

Entre «HashTable » et «HashMap»

La classe HashMap est similaire à la classe HashTable sauf que:

HashMap **n'est pas Thread-safe**

HashMap autorise l'utilisation de la **valeur null**.

La classe HashMap utilise un tableau de listes chaînées (blocs) pour le stockage de ses éléments.

L'index d'une clé dans le tableau est déterminé grâce à un algorithme utilisant la valeur de hachage de l'objet (fonction de hachage).

Si deux objets possèdent la même valeur de hachage, ils doivent être insérés dans le même bloc (la même liste).

Les Collections – l'interface « Map »

Les implémentations de l'interface **« Map »** (*le lien implements*)

La classe HashMap

représente une table d'éléments associatifs (clé, valeur)
d'une taille variable et non trié

public abstract class AbstractMap implements Map {…
}
public class HashMap extends AbstractMap {…
//implémentation de toutes les méthodes de « Map »}

Les constructeurs de « HashMap »

Plusieurs constructeurs sont fournis

- sans paramètre
- avec un paramètre de type Map
- avec un paramètre (capacité initiale)
- avec deux paramètres (capacité initiale, facteur de charge)

Les Collections – l'interface « Map »

La classe HashMap

Pour l'utiliser : import java.util.HashMap;

Exemple

```
Map <Integer, String>  HM = new HashMap <Integer, String> ();
HM.put (new Integer(10), ”Alger-Est” );
HM.put (new Integer(30), ”Alger-centre");
HM.put ( new Integer (20), ”Alger-Ouest ” );
Map <Integer, String>  HM1 = new HashMap <Integer, String> (HM);
                                    // on met dans HM1 tous les éléments de HM.
```

Equivalent à écrire:

```
Map <Integer, String>  HM1 = new HashMap <Integer, String> ();
                                                    // créer une collection vide
HM1.putAll(HM);          // ajouter tous les éléments de HM à HM1
```

La classe « HashMap »

Exemple (suite)

```
// affichage des éléments
System.out.print ("Les éléments de HM sont : ");
System.out.println (HM);

// affichage des clés
Set <Integer> S1 = HM.keySet();
                // récupérer les clés de la collection dans S1

System.out.print ("Les clés HM sont : ");
System.out.println (S1);
```

On aura

Les éléments de HM sont : { 10 = Alger-Est, 30 = Alger-centre, 20 = Alger-Ouest}

Les clés de HM sont : [10 , 30, 20]

La classe « HashMap »

Exemple (suite)

```
// récupérer les éléments clé/valeur dans un Set
Set  S2  =  HM.entrySet();
System.out.println ("Les éléments de  HM  récupérés dans S2 sont : ");
System.out.println (S2);
```

On aura

Les éléments de HM récupérés dans S2 sont :

[10 = Alger-Est, 30 = Alger-centre, 20 = Alger-Ouest]

La classe « HashMap »

Exemple (suite)

```
// récupérer éléments clé/valeur dans un Set
Set   S2  =  HM.entrySet();
System.out.println ("Les éléments de  HM  récupérés dans S2 sont : ");
for (Map.Entry <integer, String>   m: S2)
        System.out.println (m.getkey()  + "\t" +m.getvalue());
```

On aura

Les éléments de HM récupérés dans S2 sont :

10 Alger-Est

30 Alger-centre

20 Alger-Ouest

La classe « HashMap »

Exemple (suite)

```
// récupérer éléments clé/valeur dans un Set
Set   S2  =  HM.entrySet();
Iterator  it = S2.iterator();          // en utilisant un itérateur
While (it.hasnext())
     { Map.Entry   m = (Map.Entry) it.next();
          System.out.println (m.getkey()  + "\t" + m.getvalue()); }
```

On aura

Les éléments de HM récupérés dans S2 sont :

10 Alger-Est

30 Alger-centre

20 Alger-Ouest

La classe « HashMap »

Exemple (suite)

```
// récupérer la partie Valeur des éléments dans une collection
Collection   C  =  HM.values();
System.out.println ("Les éléments de  HM  récupérés dans C sont : ");
System.out.println (C);
```

On aura

Les éléments de HM récupérés dans C sont :

[Alger-Est, Alger-centre, Alger-Ouest]

La classe « HashMap »

Remarques

- Avec la méthode values, au lieu de « Collection », on peut utiliser un type spécifique « List » par exemple
- Un type ensemble « Set » ne peut pas être utilisé car il n'accepte pas les doublons de valeurs (au cas où ils existent dans la HashMap)
- Avec la méthode entryKey , Si on précise un « TreeSet » pour récupérer les clés, on aura un ensemble trié de clés
- Pour le parcours d'une collection (Collection, Set, List,…), on peut utiliser une boucle for ou même un itérateur (Iterator) sur les éléments.

La classe « HashMap »

Exemple (suite)

```
                    // recherche d'éléments dans HM
If (HM.containsKey(25)) System.out.println (" 25 Existe");
  else    System.out.println (" 25 n'existe pas");      // recherche par clé

If (HM.containsValue("Alger-centre"))      // recherche par valeur
                    System.out.println ("Alger-centre Existe");
  else    System.out.println (" Alger-centre  n'existe pas");
```

On aura

25 n'existe pas

Alger-centre existe

L'interface « SortedMap »
Les tables de hachage triées

L'interface « SortedMap »

L'interface **SortedMap<K,V>,** ajoutée à Java 1.2, définit les fonctionnalités d'une Map dont **les clés sont triées**. Elle hérite de l'interface Map.

```
Interface SortedMap extends Map
                    { // signatures de méthodes …}
```

L'ordre dans les clés correspond à l'ordre naturel (prédéfini) ou
- en implémentant l'interface Comparable) ou
- en fournissant un Comparator à la création de l'instance de la collection.

L'ordre des éléments est respecté lors de l'invocation des méthodes entrySet(), keySet() et values().

Les Collections – l'interface « Map »

L'interface « SortedMap »

Les méthodes de l'interface « SortedMap »

Méthode	Rôle
Comparator< ? super K) comparator()	Retourner l'instance de type Comparator associée à la collection ou null si c'est l'ordre naturel qui doit être utilisé
Set<Map.Entry<K,V>> entrySet()	Retourner un ensemble des paires clé/valeur de la collection
K firstKey()	Retourner la première clé de la collection. Lève une exception de type NoSuchElementException si la collection est vide
SortedMap<K,V> headMap(K toKey)	Retourner un sous-ensemble de la collection contenant les éléments dont les clés sont strictement inférieures à celle fournie en paramètre
Set<K> keySet()	Retourner un ensemble des clés de la collection
K lastKey()	Retourner la dernière clé de la collection. Lève une exception de type NoSuchElementException si la collection est vide
sortedMap<K, V> subMap(K fromKey, K toKey)	Retourner un sous-ensemble de la collection contenant les éléments dont les clés sont strictement inférieures à celle fournie en premier paramètre et supérieures ou égales à celle fournie en second paramètre
SortedMap<K,V> tailMap(K fromKey)	Retourner un sous-ensemble de la collection contenant les éléments dont les clés sont supérieures ou égales à celle fournie en paramètre
Collection(V) values()	Retourner une collection de toutes les valeurs de la Map

Les Collections – l'interface « SortedMap »

Les implémentations de l'interface **« SortedMap »** (*le lien implements*)

La classe TreeMap

représente une Map d'éléments associatifs (clé, valeur)
d'une taille variable et trié selon les clés

public class TreeMap implements Map, SortedMap {…
//implémentation de toutes les méthodes de « Map » et « SortedMap »}

La classe « TreeMap »

Les implémentations de l'interface « SortedMap » (*le lien implements*)

La classe TreeMap

La classe TreeMap organise ses éléments de manière triée dans une ***structure d'arbre***

Constructeur	Rôle
TreeMap()	Constructeur par défaut qui crée une collection vide utilisant l'ordre naturel des clés des éléments
TreeMap(Comparator<? super K> comparator)	Créer une instance vide qui utilisera le Comparator fourni en paramètre pour déterminer l'ordre des éléments
TreeMap(Map<? extends K,? extends V> m)	Créer une instance contenant les éléments fournis en paramètres qui utilisera l'ordre naturel des clés des éléments
TreeMap(SortedMap<K,? extends V> m)	Créer une instance contenant les éléments fournis en paramètres

La classe TreeMap

Pour l'utiliser : import java.util.TreeMap;

Exemple

```
TreeMap  <Integer, String> TM = new TreeMap  <Integer, String> ();
/*Ajouter des éléments à TreeMap*/
 TM.put(12, "val1"); TM.put (5, "val2");
 TM.put(8, "val3"); TM.put (2, "val4");
/* Afficher le contenu en utilisant Iterator */
Set  S = TM.entrySet();
Iterator it = S.iterator();
while(it.hasNext())
{ Map.Entry m = (Map.Entry)it.next();
System.out.println
    ("clé: "+ m.getKey() + " + "- valeur: "+m.getValue()); }
```

La classe « TreeMap »

On aura

clé: 2 - valeur: val4
clé: 5 - valeur: val2
clé: 8 - valeur: val3
Clé:12 – valeur: val1

On voit que les éléments sont triés dans l'ordre des clés

La classe TreeMap

Pour l'utiliser : import java.util.TreeMap;

Exemple (suite)

```
TreeMap  <Integer, String> TM = new TreeMap  <Integer, String> ();
                /*Ajouter des éléments à TreeMap*/
 TM.put(12, "val1"); TM.put (5, "val2");
 TM.put(8, "val3"); TM.put (2, "val4");

                /* Créer une autre Map TM2*/
Map  <Integer, String> TM2 = new TreeMap  <Integer, String> ();
TM2.put(20, "val5");
 TM2.putAll (TM); /* ajout des éléments de TM*/
TM2.put (10 , "val6");
```

La classe TreeMap

Pour l'utiliser : import java.util.TreeMap;

Exemple (suite)

```
/* récupérer les éléments dans S2*/
Set  S2 = TM2.entrySet();      Map.Entry  m;

/* parcourir avec un itérateur */
Iterator it2 = S2.iterator();
while(it2. hasNext())
{m = (Map.Entry)it2.next();
 System.out.println
        ("clé: "+ m.getKey() + " " + "- valeur: "+m.getValue()+"\n"); }
```

La classe « TreeMap »

Exemple (suite)

On aura

clé: 2 - valeur: val4
clé: 5 - valeur: val2
clé: 8 - valeur: val3
clé: 10 - valeur: val6
Clé:12 – valeur: val1
clé: 20 - valeur: val5

On voit que les éléments sont triés dans l'ordre des clés

La classe TreeMap

Pour l'utiliser : import java.util.TreeMap;

Exemple (suite)

```
/* accès à des éléments spécifiques*/
System.out.print("clé: 8" + " - valeur: "+TM2.get(8)+"\n");
System.out.print("clé: 20" + " - valeur: "+TM2.get(20)+"\n");

/*première et dernière clé*/
System.out.print("première clé : "+TM2.firstKey() +"\n" +" dernière clé :
"+TM2.lastKey()) ;} }

/*créer une sous-Map*/
SortedMap  <Integer, String>  SM  = TM2.tail(8);    // les clés >= 8
System.out.print("les éléments de la sous-Map sont : +"\n" + SM);
```

La classe « TreeMap »

Exemple (suite)

On aura

clé: 8 - valeur: val3
clé: 20 - valeur: val5

première clé: 2
dernière clé: 20

les éléments de la sous-Map sont:
{ 8 = val3, 10 = val6, 12 = val1, 20 = val5 }

Autres classes

Autre interface de « SortedMap »

NavigableMap: permet le tri et le parcours dans les deux sens (croissant, décroissant)

Pour les accès concurrents (multi-threads)

class ConcurrentNavigableMap

class ConcurrentSkipListMap

…