FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Gestion des Collections

Licence Professionnelle : Développement Informatique

Pr: Youssef Ouassit

Plan Chapitre : Collections et tableaux

- Introduction
  - Les tableaux unidimensionnelles et multidimensionnels.
  - Qu'est-ce qu'une Collection?
  - Le Java Collections Framework.
- Interfaces
  - Collections ➔ « Set » et « List »
  - Maps ➔ « Map » et « SortedMap »
  - Comparateurs ➔ « Comparable » et « Comparator »
- Implémentations
  - HashSet et TreeSet
  - ArrayList et LinkedList
- Algorithmes
  - Tri
  - Autres: Recherche, Mélange, etc.

Introduction

Les tableaux : Définition

En Java, un tableau (tableau) est une **structure de données** contenant un groupe d'éléments tous du **même type**, avec des adresses consécutives sur la mémoire (memory). Le tableau **a le nombre fixé d'éléments et vous ne pouvez pas changer sa taille**.

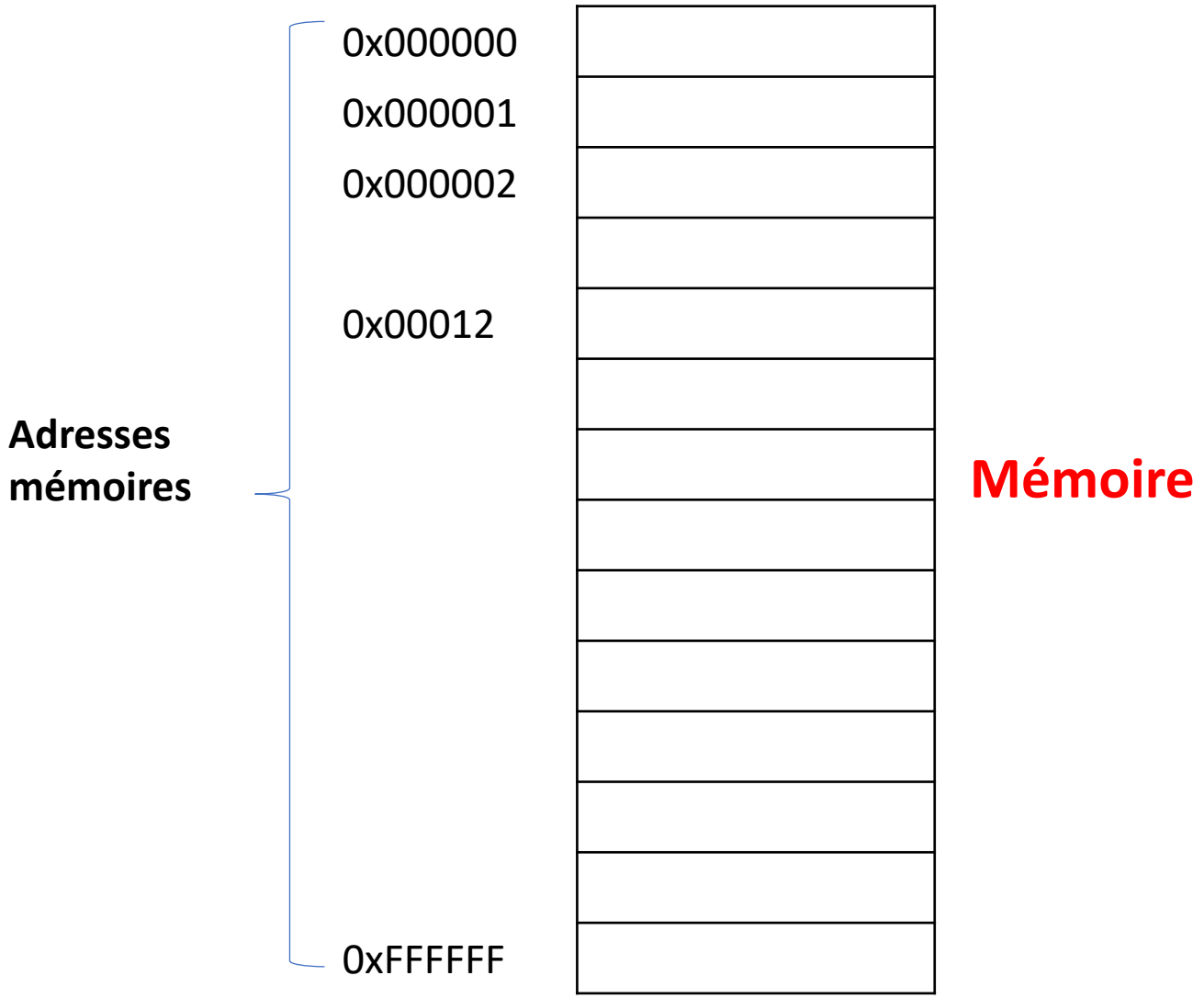
Introduction

Les tableaux : Définition

```
int a = 5;

int T[] = { 5, 6, 0}

int A[] = new int[4];
```



Les tableaux : Déclaration

Syntaxe :

```
type   nomTableau[] = new type[taille];
type[]   nomTableau = new type[taille];
type   nomTableau[] = { liste des valeurs };
```

Exemples:

```
int A[ ] = new int[10];   // un  tableau d'entiers de taille 10
double B[ ] = new double[20];   // un  tableau de réels de taille 20
boolean C[ ] = new boolean[5];   // un  tableau de booléen de taille 5
int D[ ] ={7, 3, 9, 3};   // un  tableau d'entiers initialisé par 4 valeurs
```

Les tableaux à deux dimensions: Déclaration

Syntaxe :

```
type   nomMatrice[][] = new type[lignes][colonnes];
Type[][]   nomMatrice = new type[lignes][colonnes];
type   nomMatrice[][] = { {liste des valeurs}, {liste valeurs} };
```

Exemples:

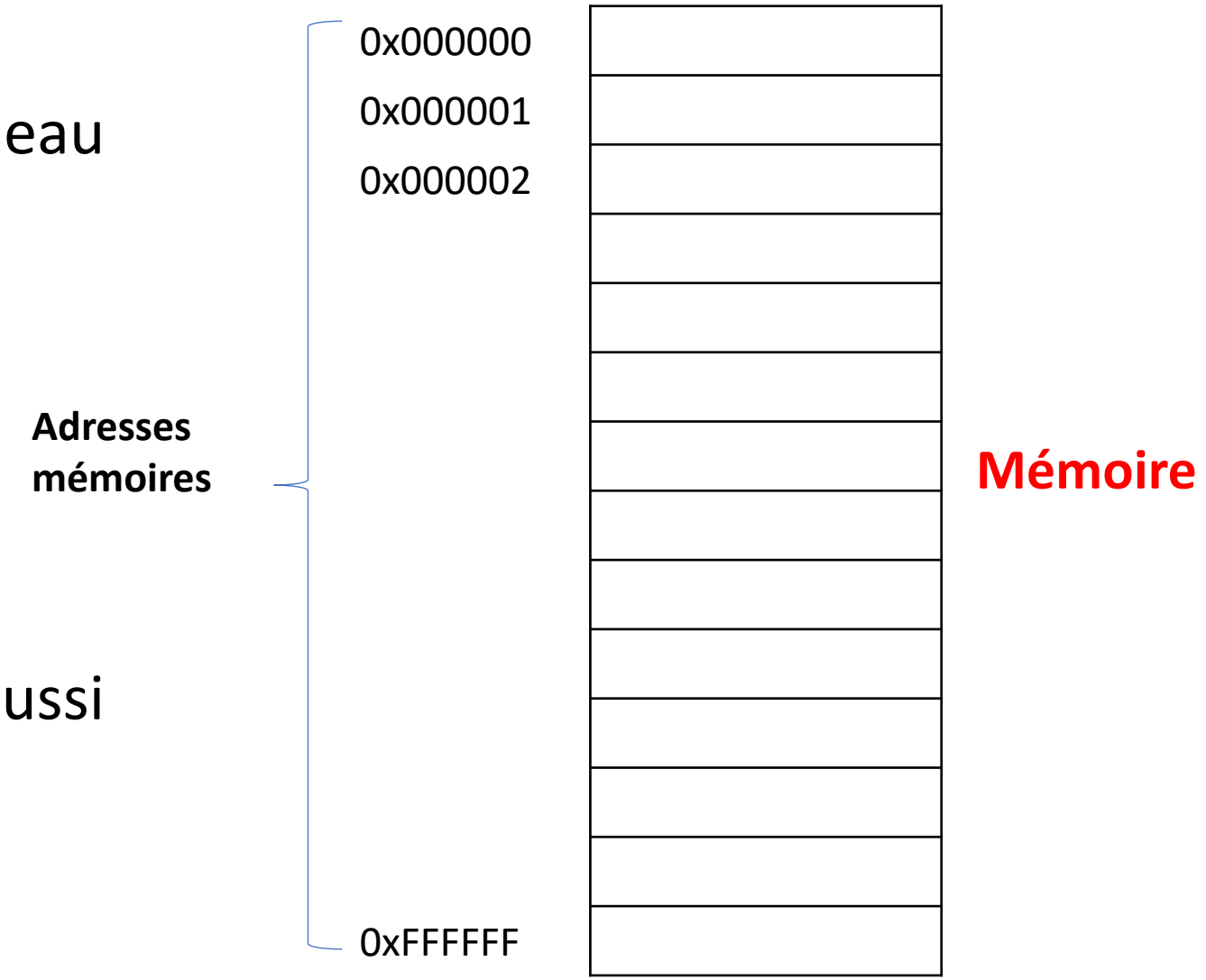
```
int A[][] = new int[5][4];   // une  matrice d'entiers de 5 lignes et 4 colonnes
double B[][] = new double[3][3];  // une  matrice de réels de 3 lignes et colonnes
int C[][]={{1, 0, 0}, {0, 1, 0} , {0, 0, 1}}; // une matrice initialisée dès la déclaration
```

Introduction

Les tableaux : Déclaration

Un tableau d'objets est un tableau d'adresses de ces objets :

String A[] = new String[4];
Personne P[] = new Personne[3];



NB : Les noms des tableaux sont aussi des objets.

Les tableaux : Accéder à un élément

Pour accéder à un élément du tableau :

nomTableau[indice]

nomMatrice[indice_ligne][indice_colonne]

Exemples :

```
int A[] = new int[5];
A[0] = 10;
A[1] = 5;
A[2] = 9;
A[3] = 20;
A[4] = 100;
```

```
int M[][] = new int[2][2];
M[0][0] = 5;
M[0][1] = 15;
M[1][0] = 10;
M[1][1] = 20;
```

Introduction

Les tableaux : Affectation

Exercice : qu'affichera le code suivant ?:

```
int A[] = {3, 1, 5};
int B[] = A;
A[0] = 10;
System.out.println("A[0] = " + A[0]);
System.out.println("B[0] = " + B[0]);
```

Solution :

```
A[0] = 10
B[0] = 10  // int B[] = A; a copié l'adresse et par les valeurs.
```

Les tableaux : Itérer un tableau

Itérer les éléments d'un tableau :

```
int T[] = new int[10];
int l = T.length;  # donne le nombre des éléments d'un tableau

// Itérer par indice
for (int i = 0; i < T.length; i++) {
    System.out.println(T[i]);
}
// Itérer par valeur (foreach)
for (int a : T) {
    System.out.println(a);
}
```

Les tableaux : Itérer une matrice

Itérer les éléments d'une matrice :

```
int M[][] = new int[4][3];

// Itérer une matrice ligne par ligne
for (int i = 0; i < M.length; i++) {
    for (int j = 0; j < M[0].length; j++) {
        System.out.println(M[i][j]);
    }
}
```

Les tableaux : Affectation

Remarque:

En Java, la taille d'un tableau est fixe après sa création. Cela signifie que si vous allouez un tableau de taille 10 et que vous avez besoin de plus d'espace, vous ne pouvez pas simplement ajouter de nouveaux éléments au tableau existant. Vous devrez créer un nouveau tableau de plus grande taille et copier les éléments existants dans ce nouveau tableau, ce qui peut être coûteux en termes de performance.

Introduction

Qu'est ce qu'une collection ?

- Collection
  - Un objet utilisé afin de représenter un groupe d’objets
- Utilisé pour:
  - Stocker, retrouver et manipuler des données
  - Transmettre des données d’une méthode à une autre
- Exemples:
  - Un dossier d’emails (collection de messages)
  - Un répertoire téléphonique (collection de correspondances NOM – N°)

Introduction

Java Collections Framework

- Définit toute la structure des collections en Java
- Constitue une architecture unifiée pour
  - la représentation des collections
  - la manipulation des collections
- Contient des:
  - Interfaces
    - Types de données abstraits représentant des collections
    - Permettent de manipuler les collections indépendamment de leur représentation
    - Organisées en une hiérarchie unique
  - Implémentations
    - Implémentations concrètes des interfaces ➔ Structures de données réutilisables
    - Fournies pour accélérer le développement
  - Algorithmes
    - Méthodes standards fournissant des opérations comme le tri, la recherche, etc.

Introduction

Java Collections Framework

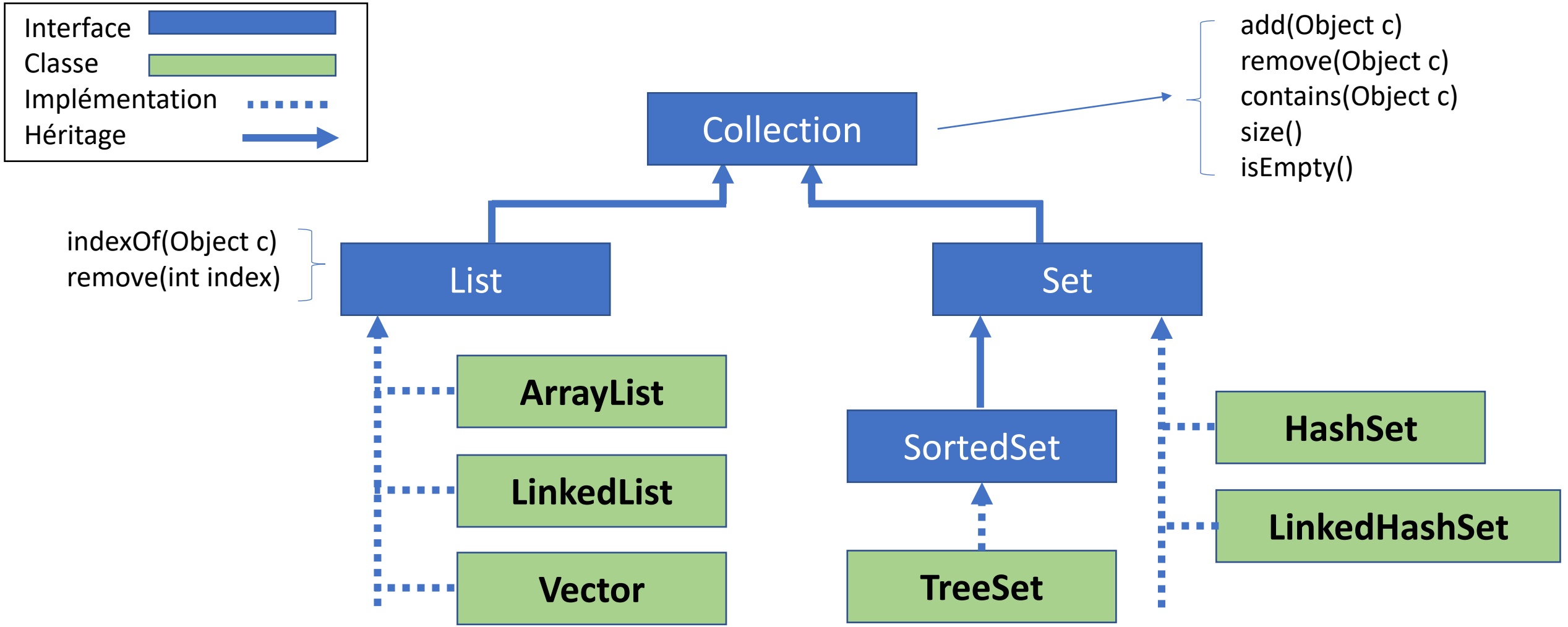
Il existe 3 groupes principals d'interfaces liées aux collections

- Collection
  - Racine de la structure des collections
  - Représente un groupe d'objets (dits « éléments »)
  - Peut autoriser ou non les duplicats
  - Peut contenir un ordre intrinsèque ou pas
- Map
  - Un objet qui établit des correspondances entre des clés et des valeurs
  - Ne peut en aucun cas contenir des duplicats

  ➔ Chaque clé ne peut correspondre qu'à une valeur au plus
- Interfaces de comparaison
  - Permettent le tri des objets du type implémentant l'interface
  - Deux versions: « Comparator » et « Comparable »

Java Collections Framework

Interface: Collection



Java Collections Framework

Interface: Collection

La classe ArrayList :

La classe ArrayList en Java est une des implémentations les plus utilisées de l'interface List de la bibliothèque Java Collections Framework. Elle fournit une structure de données dynamique qui peut contenir des éléments et dont la taille s'ajuste automatiquement lorsque vous ajoutez ou supprimez des éléments.

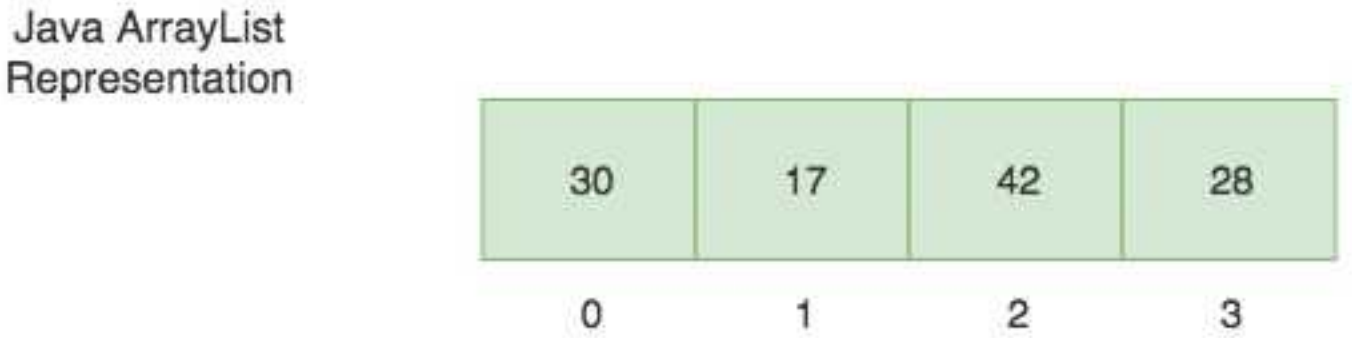
Les méthodes:

add(Object c) ; add(Object c, int index); remove(Object c); remove(int index); get(int index); toArray(); size(); ….

Java Collections Framework

Interface: Collection

La classe ArrayList :



Java Collections Framework

Interface: Collection

La classe LinkedList :

La classe LinkedList en Java fait partie du Java Collections Framework et implémente l'interface List (tout comme ArrayList), mais elle fonctionne différemment en termes de structure interne. Contrairement à ArrayList, qui utilise un tableau dynamique sous-jacent, LinkedList est une liste chaînée, ce qui signifie que chaque élément de la liste contient une référence (ou un lien) vers l'élément suivant et, dans certains cas, vers l'élément précédent..

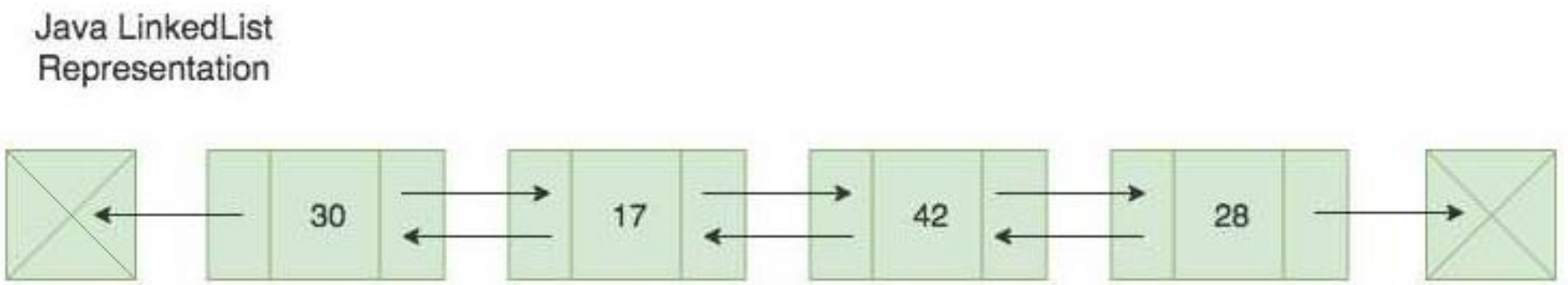
Les méthodes:

add(Object c) ; add(Object c, int index); remove(Object c); remove(int index); get(int index); toArray(); addFirst(Object c); addLast(Object c); ….

Java Collections Framework

Interface: Collection

La classe LinkedList :



Java Collections Framework

Interface: Collection

La classe HashSet:

La classe **HashSet** fait partie de la bibliothèque Java Collections Framework et implémente l'interface Set. Un Set est une collection d'éléments qui ne permet pas les doublons : cela signifie que chaque élément ajouté dans un HashSet doit être unique.

Les méthodes:

add(Object c) ; remove(Object c); toArray();length(); ….

Java Collections Framework

Interface: Collection

La classe TreeSet:

La classe **TreeSet** fait partie du Java Collections Framework et implémente l'interface Set. Elle est basée sur un arbre binaire de recherche (un arbre rouge-noir). Contrairement aux autres implémentations de Set comme HashSet ou LinkedHashSet, qui ne garantissent pas l'ordre des éléments, TreeSet trie automatiquement ses éléments..

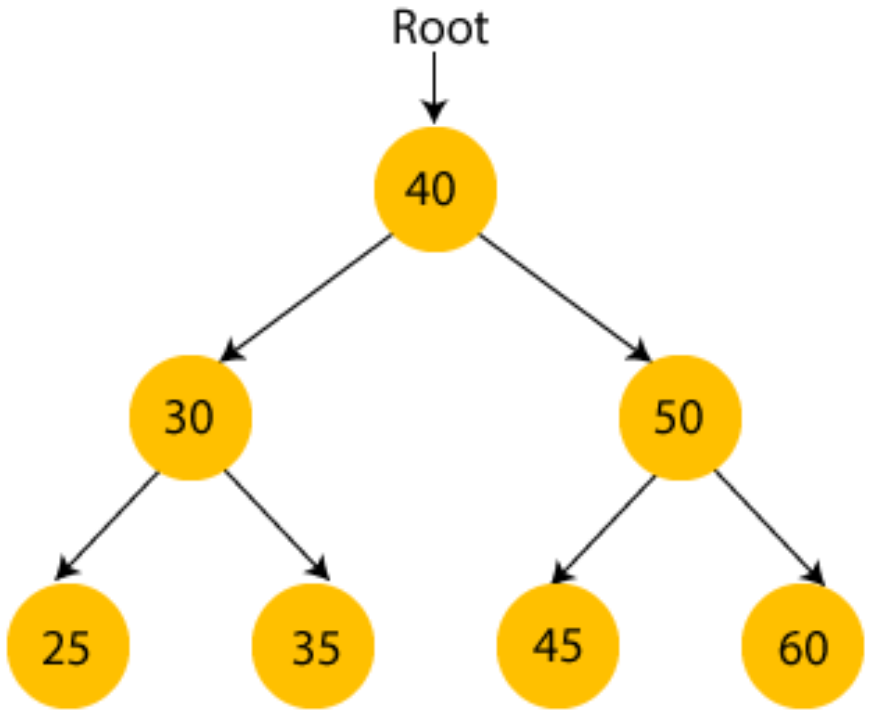
Les méthodes:

add(Object c) ; remove(Object c); toArray();first(); last();….

Java Collections Framework

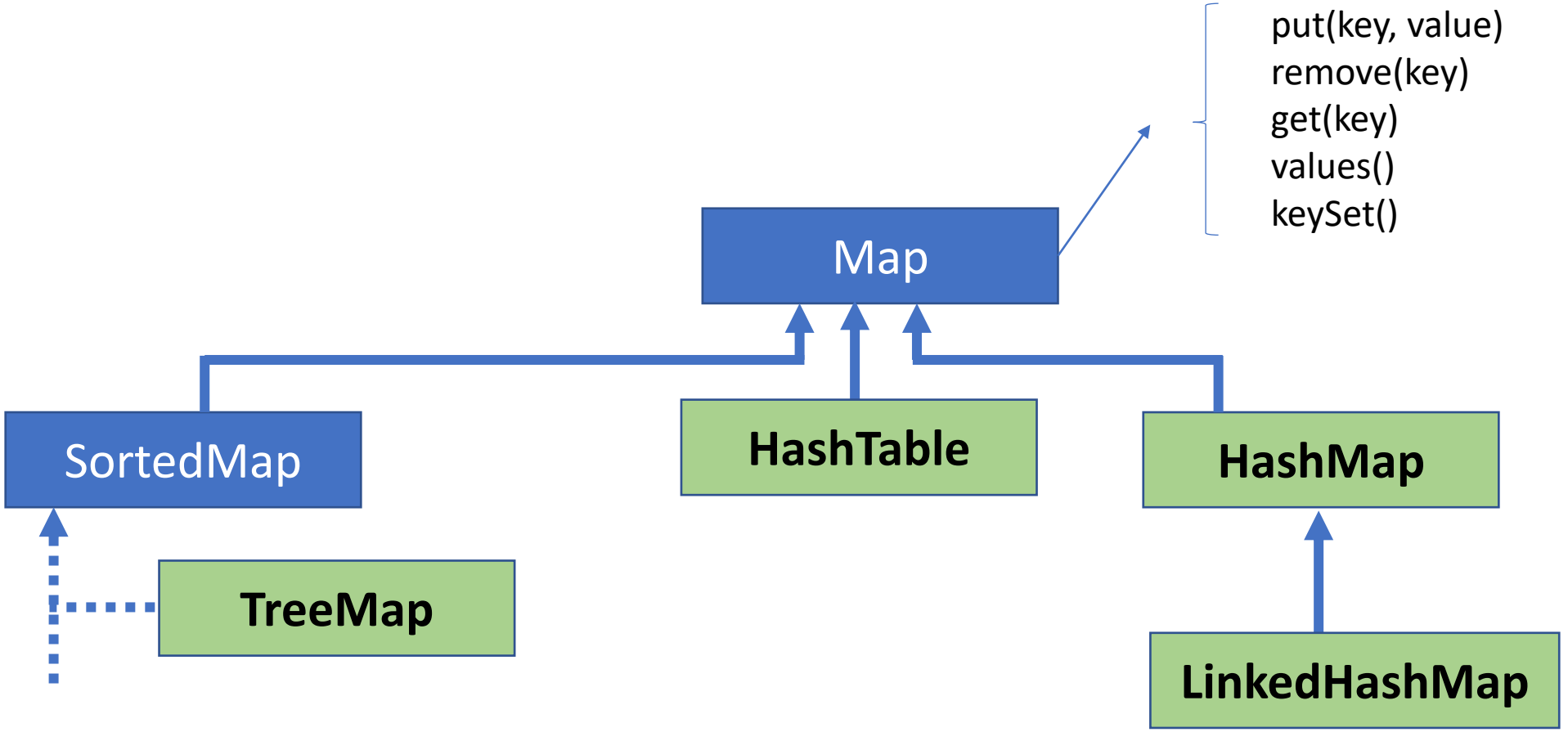
Interface: Collection

La classe TreeSet:



Java Collections Framework

Interface: Map



Java Collections Framework

Interface: Collection

La classe HashMap:

La classe HashMap fait partie de Java Collections Framework et implémente l'interface Map. Elle permet de stocker des paires clé-valeur, où chaque clé est unique et associée à une valeur. Les éléments dans une HashMap sont stockés dans des structures de données appelées tables de hachage. L'idée principale derrière une HashMap est d'offrir un accès rapide aux valeurs via des clés.

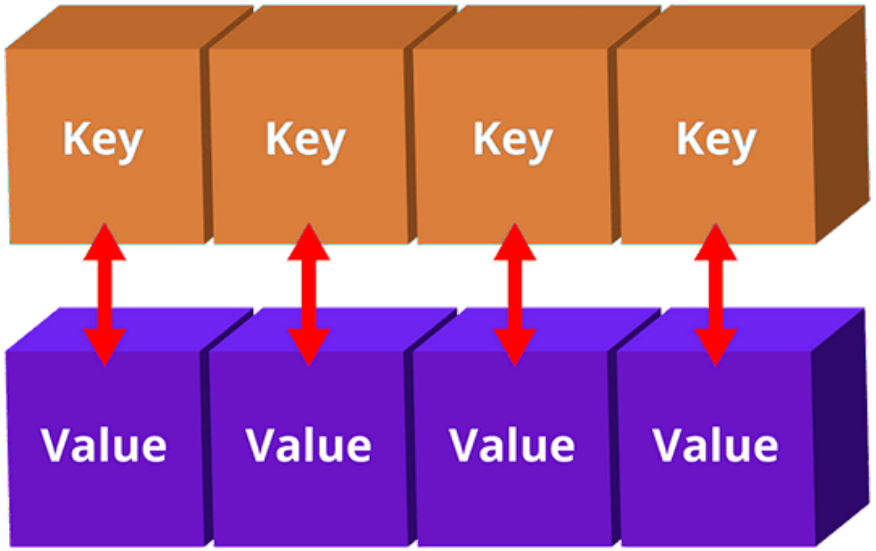
Les méthodes:

put(Object key, Object value) ; remove(Object key); …

Java Collections Framework

Interface: Map

- HashMap :



Java Collections Framework

La comparaison des objets dans les collections

Deux interfaces :

- Comparable
  - Fournit une méthode de comparaison au sein d'une classe
  - Impose de redéfinir une seule méthode `public int compareTo(Object o)` qui renvoie un entier:
    1 si l'objet courant > l'objet « o » fourni dans la méthode
    0 si l'objet courant = l'objet « o » fourni dans la méthode
    -1 si l'objet courant < l'objet « o » fourni dans la méthode

- Comparator
  - Permet de définir une classe servant de comparateur à une autre
  - Impose de redéfinir une seule méthode
    `public int compare(Object o1, Object o2)` qui renvoie un entier:
    1 si *o1* > *o2*
    0 si *o1* = *o2*
    -1 si *o1* < *o2*

Java Collections Framework

La comparaison des objets dans les collections

Exemple d'implémentation de *Comparable*

```
import java.util.*;
public class Name implements Comparable {
    private String  firstName, lastName;
    public Name(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    public int compareTo(Object o) {
        Name n = (Name) o;
        int lastCmp = lastName.compareTo(n.lastName);
        if(lastCmp!=0)
            lastCmp = firstName.compareTo(n.firstName);
        return lastCmp;
    }
}
```

Java Collections Framework

La comparaison des objets dans les collections

- Les types primitifs contiennent toujours un ordre naturel

Class	Natural Ordering
`Byte`	signed numerical
`Character`	unsigned numerical
`Long`	signed numerical
`Integer`	signed numerical
`Short`	signed numerical
`Double`	signed numerical
`Float`	signed numerical
`BigInteger`	signed numerical
`BigDecimal`	signed numerical
`File`	system-dependent lexicographic on pathname.
`String`	lexicographic
`Date`	chronological

Java Collections Framework

Algorithmes : Tri

On peut trier un tableau/collection au moyen de méthodes simples:

- Trier un tableau ➔ méthode « sort » de la classe « Arrays »
  - `Arrays.sort(<type>[])`
- Trier une collection ➔ méthodes « sort » de la classe « Collections »
  - Ne fonctionne qu'avec les collections dérivant de « List »
  - Si les éléments de la collection sont comparables (implémentent l'interface « Comparable »)
    ➔ `Collections.sort(List)`
  - Si les éléments de la collection ne sont pas comparables, il faut alors indiquer quelle classe servira de comparateur
    ➔ `Collections.sort(List, Comparator)`

Java Collections Framework

Algorithmes : Autre

- D'autres opérations sont fournies par le Java Collections Framework:
  - Mélange
    - *Collections.shuffle(liste)*
  - Inversion de l'ordre
    - *Collections.reverse(liste)*
  - Réinitialisation des éléments (remplace tous les éléments par l'objet spécifié)
    - *Collections.fill(liste, objetParDefaut)*
  - Copie des éléments d'une liste dans une autre
    - *Collections.copy(listeSource, listeDestination)*
  - Recherche d'extrema
    - Sur base de la position des éléments *min(liste)* et *max(liste)*
    - Sur base d'un comparateur *min(liste, comparateur)* et *max(liste, comparateur)*