Collections

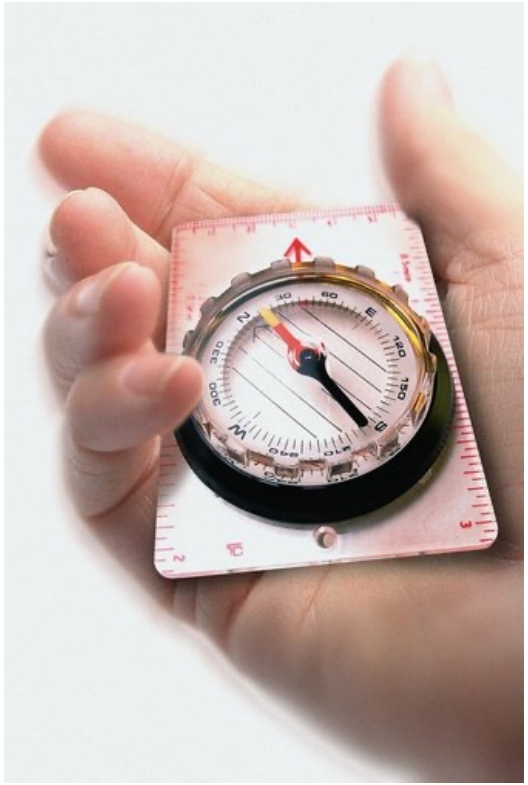
Java Standard Edition

Objectifs du cours

En complétant ce cours, vous serez en mesure de :

- **Expliquer** ce que sont les collections
- **Les utiliser et choisir** la meilleure solution
- **Énumérer** les collections les plus courantes

Plan de cours



- **Présentation**. Qu'est-ce qu'une collection ? A quoi cela sert? Pourquoi les utiliser ?
- **Interfaces et implémentations**. Plusieurs types pour tous les besoins
- **Itérateurs**. Comment itérer sur une collection.
- **Comparator & Comparable**. Comment trier les éléments d'une collection.
- **Arrays & Collections**. Deux classes utilitaires pour vos tableaux et collections.

Collections

PRÉSENTATION

Que sont les collections ?

Pourquoi les collections

- En programmation nous avons besoin de :
  - Stocker des éléments
    - Listes de cartes, propriétés, …
  - Manipuler un ensemble d’éléments
    - Récupérer tous les étudiants du campus CCI
  - Relier certains éléments à d'autres
    - Une propriété a une valeur
  - Une gestion adaptée, simple et puissante
    - Lire rapidement toutes les données stockées
    - Insérer rapidement des données…

Aperçu

- Les collections sont des classes qui stockent des objets
- Gestion automatique et dynamique
- Permet de trier facilement les éléments
- Plusieurs types de collections :
  - Adapté à vos besoins
  - Meilleures performances

Arrays

- Avec les tableaux, vous pouvez :
  - Rassemblez des objets comme le font les Collections…
  - … mais la taille des tableaux n'est pas dynamique
- Rappel…
  - Déclaration (trois façons mais une seule montrée ici)* :

```
int myTab[] = {1,2,3,4};
```

  - Accéder :

```
int firstElement = myTab[0];
```

Avantages des collections

- Avec les collections :
  - La taille est dynamique
  - Les méthodes fournissent des opérations courantes :
    - Ajouter, supprimer, obtenir, … des éléments

```
ArrayList<String> myString = new ArrayList<String>();

myString.add(" Hello ");
myString.add(" you ");
//a collection is never too small
myString.add("and you ");
```

Présentation

Type Generics

- Il spécifie le type de données stockées dans les collections
- Nouvelle fonctionnalité depuis Java 5
- Éviter les différents types d'objets stockés dans les collections
- Le casting est maintenant inutile
- Les données stockées sont vérifiées au moment de la compilation plutôt qu'au moment de l'exécution

Exemple de Generics et comparaison

BEFORE 5

```
ArrayList myCats =
                new ArrayList();

myCats.add(new Cat());
myCats.add(new Dog());

Cat c = (Cat) myCats.get(0);
// And getting the others
//      element?
// How am I sure it is a cat?
// You're never sure...
```

SINCE 5

```
ArrayList<Cat> myCats =
        new ArrayList<Cat>();

myCats.add(new Cat());
myCats.add(new Dog());
// Error at compilation...

Cat c = myCats.get(0);
// Here you're always sure !!
// Thanks to generics ☺
// Or almost...
```

Présentation

Diamond operator

- Java 7 introduit l'opérateur **diamond** pour les génériques

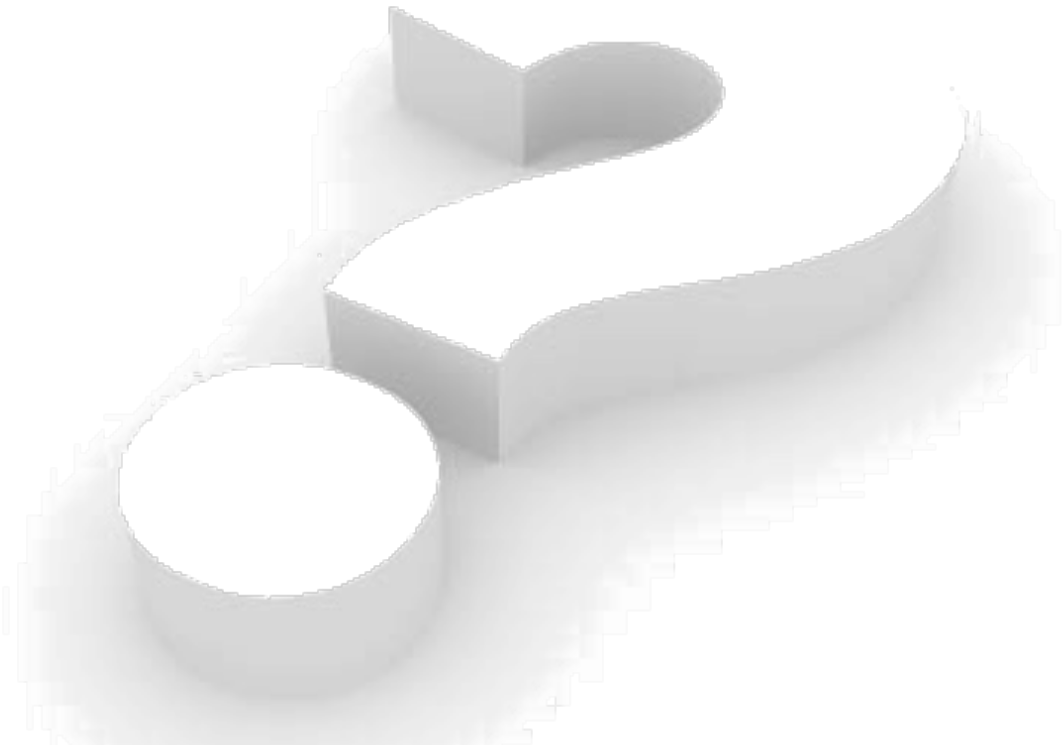
BEFORE JAVA 7

```
List<Mark> marks = new ArrayList<Mark>();

Map<Student, List<Mark>> students =
                        new HashMap<Student, List<Mark>>();
```

SINCE JAVA 7

```
List<Mark> marks = new ArrayList<>();

Map<Student, List<Mark>> students = new HashMap<>();
```

Questions ?



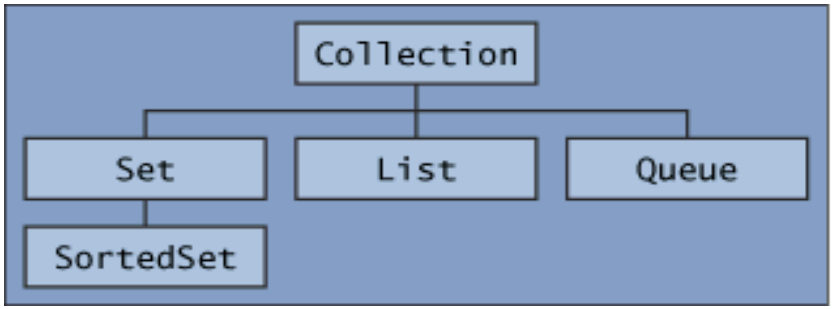
Collections

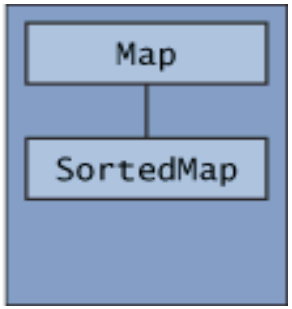
INTERFACES & IMPLÉMENTATIONS

Plusieurs types pour tous les besoins

L'arbre des connaissances

- Interfaces de collection :

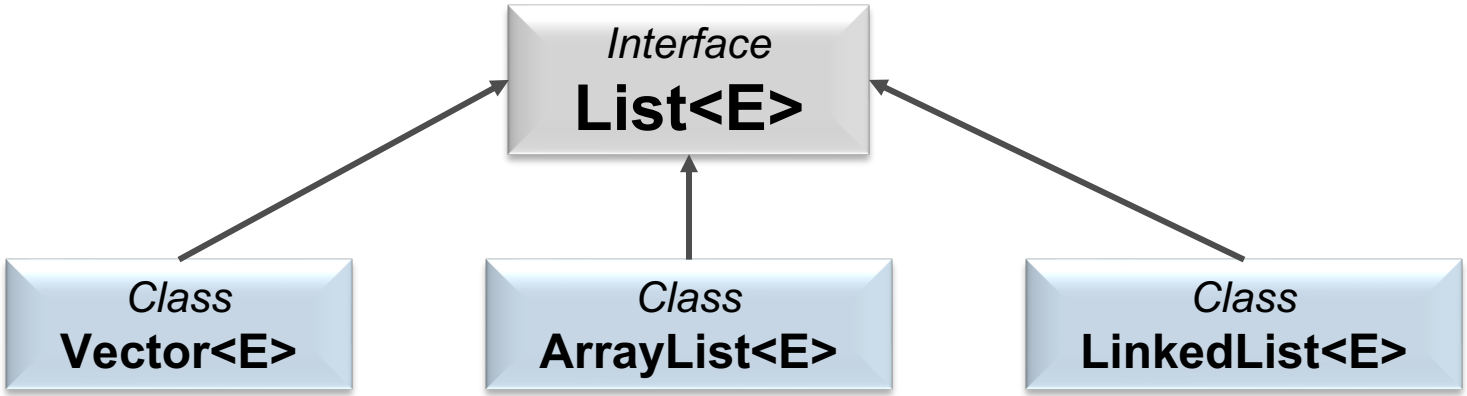




- Deux arbres distincts sur cette hiérarchie
  - Toutes les interfaces utilisent des génériques
- Chaque interface a ses avantages et ses inconvénients
  - Ex : une file d'attente peut agir comme une pile (LIFO)
  - Vous devez choisir la meilleure stratégie en fonction de vos besoins

Interface List

- Une collection ordonnée
- Tous les éléments à l'intérieur sont accessibles par un index entier
  - La position dans la liste !
- Peut contenir plusieurs fois le même élément



implémentations List

- ArrayList :
  - Bonne performance pour get et set
- LinkedList :
  - Meilleures performances pour ajouter/supprimer
  - De moins bonnes performances pour obtenir et définir la valeur
- Vector :
  - Thread safe (toutes les méthodes sont synchronisées)
  - Mauvaise performance

les méthodes courantes de List

- add(<E> element) :
  - Ajouter un élément en fin de liste
- <T> get(int index) :
  - Renvoie l'élément à la position spécifiée
- <T> remove(int index) :
  - Supprime l'élément à la position spécifiée
  - Renvoie l'élément supprimé

les méthodes courantes de List

- int size() :
  - Renvoie la taille de la liste

- Pour en savoir plus, consultez la Javadoc

Exemple

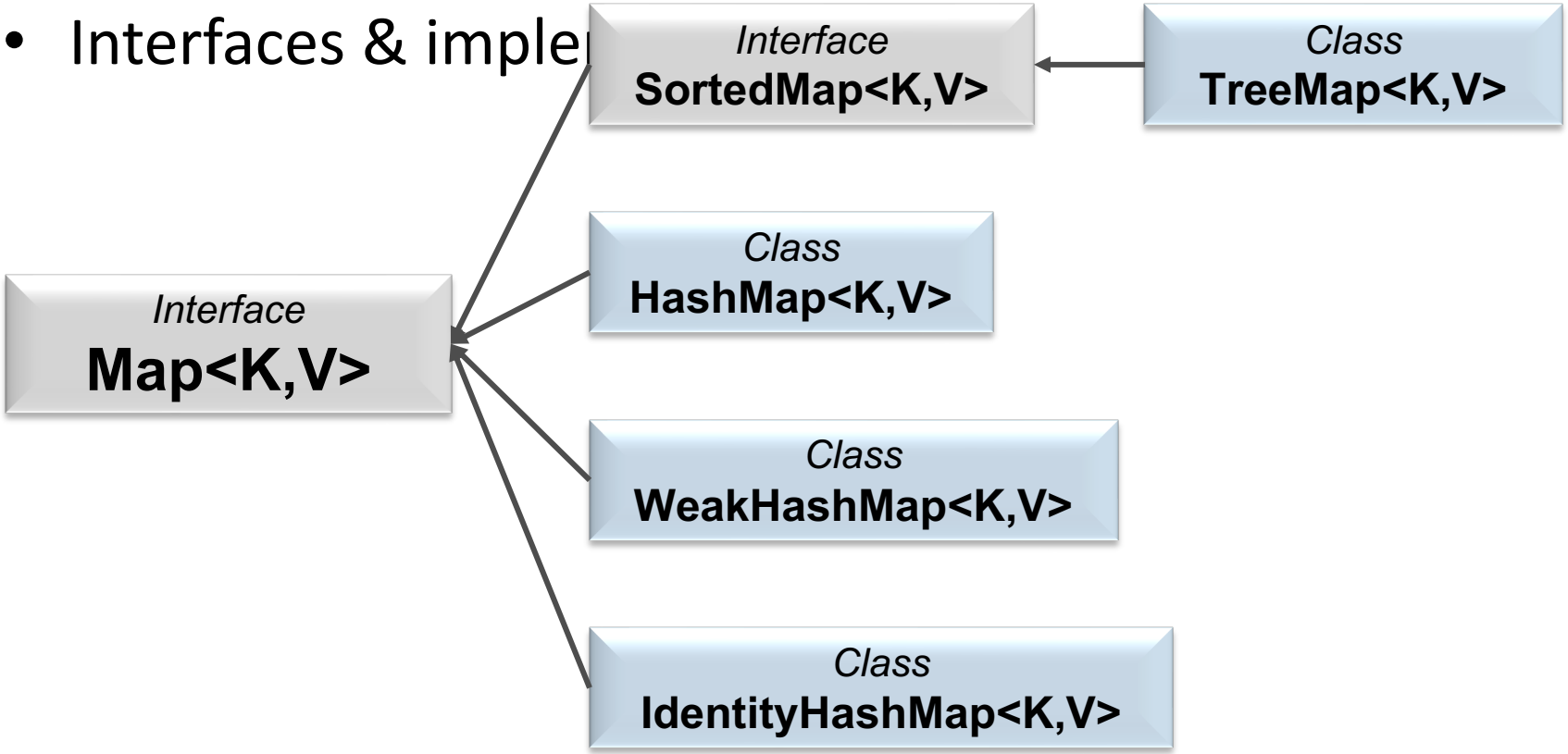
```
List<String> myList = new ArrayList<String>();
myList.add("Monday");
myList.add("Tuesday");
myList.add("Wednesday");
myList.add("Thursday");
myList.add("Friday");
myList.add("Saturday");
myList.add("Sunday");

String day = myList.get(3);
System.out.println("The fourth day is " + day );
String removeDay = myList.remove(6);
System.out.println("I removed " + removeDay );
```

Interface Map

- Un ensemble clé/valeur
- Chaque clé est unique
- Une valeur peut avoir plusieurs clés
- Les clés et les valeurs sont des objets
- Similaire à un tableau associatif

Interface Map

- Interfaces & imple



Implémentations Map

- HashMap :
  - Manipule la valeur de hachage de la clé pour être efficace
  - Peut avoir une clé nulle et plusieurs valeurs nulles
- IdentityHashMap :
  - Comme HashMap
  - Utilise l'opérateur == pour vérifier si deux clés sont égales
- TreeMap :
  - Trie des tuples par clé, dans l'ordre croissant

Les méthodes courantes Map

- put(<any> key, <any> value) :
  - Ajouter une paire clé/valeur dans la Map
- <any> get(<any> key) :
  - Renvoie la valeur liée à la clé
- <any> remove(int index) :
  - Supprime l'élément à la position spécifiée
  - Renvoie l'élément supprimé

Les méthodes courantes Map

- boolean containsKey(<any> key) :
  - Vérifie si la clé existe dans la Map
- boolean containsValue(<any> value) :
  - Vérifie si la valeur existe dans la Map

- Pour en savoir plus, consultez la Javadoc

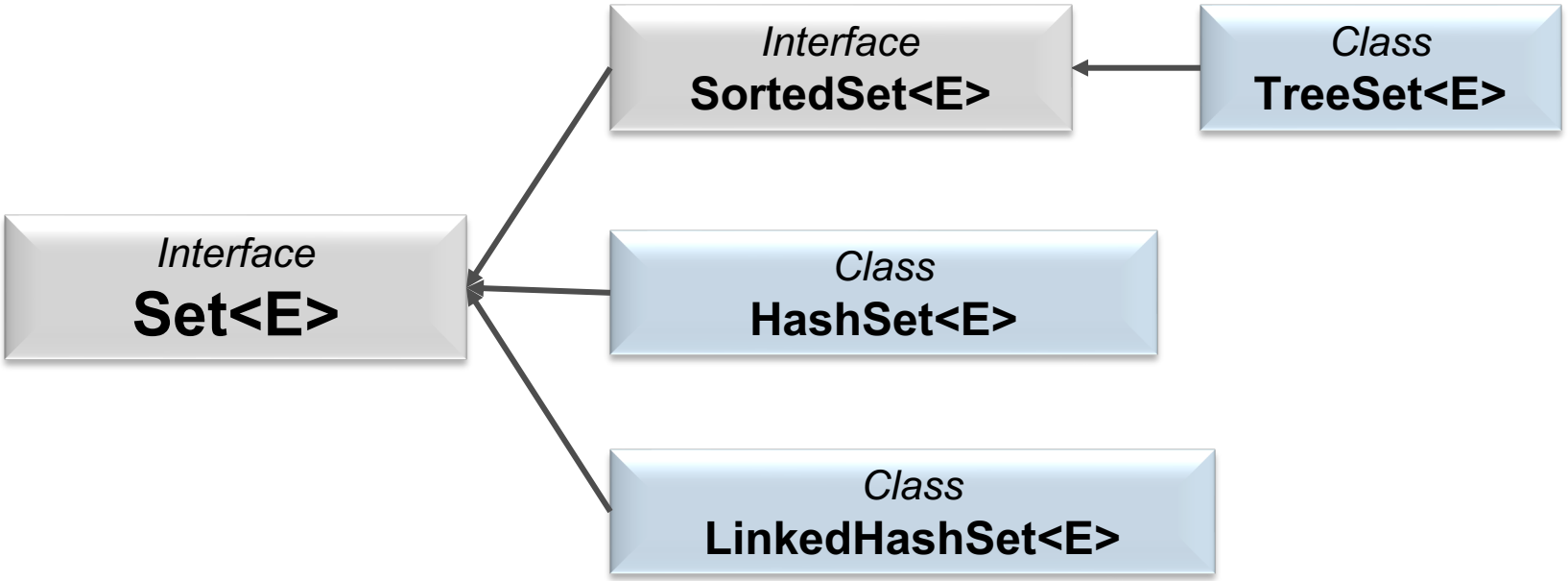
Exemple

- Interfaces and implementations

```
Map<String, int[]> myMap = new HashMap<String, int[]>();
String student = "Bob";
int[] marks = {12,14,10,7};

myMap.put(student , marks);
int[] result =  myMap.get(student);
System.out.println(student+ " has got " + result[1]);
```

Interface Set

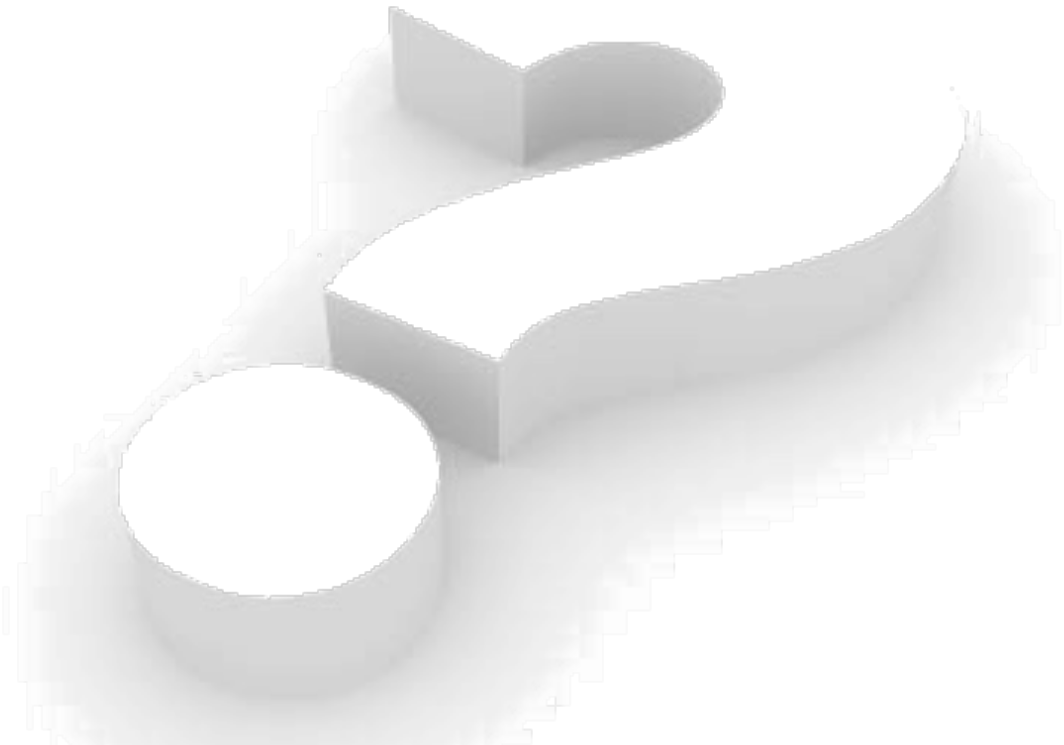
- Ensemble d'éléments uniques:



Implementations Set

- HashSet :
  - Manipule la valeur de hachage de l'élément pour être efficace
  - Un seul élément nul
- LinkedHashSet :
  - Comme HashSet
  - Conserve l'ordre dans lequel les éléments ont été insérés
- TreeSet :
  - Trie l'élément par ordre croissant

Questions ?



Collections

ITERATORS

Présentations

- Un Iterators :
  - Est un objet générique
  - Permet d'itérer sur une Collection de manière linéaire
  - Lit uniquement dans l'ordre croissant
  - Peut supprimer un élément de la Collection grâce à la méthode remove()

Exemple Iterators

- Quelques méthodes :

```
boolean hasNext(); //Check if there is a next element


E next(); //Get the next element


void remove(); // Remove the current element
```

Iterator Exemple

```
Collection<String> myCollection = new
        ArrayList<String>();
// Add elements to the collection.
myCollection.add("Remove me");
myCollection.add("Keep me");
Iterator<String> it = myCollection.iterator();
while (it.hasNext()) {
    String myElement = it.next();
    if (myElement.equals("Remove me")) {
        it.remove();
    } else {
        System.out.println(myElement);
    }
}
```

Interface ListIterator

- Permet de lire une Liste dans les deux sens :
  - Uniquement disponible pour les implémentations de liste

```
ListIterator<String> listIt = myList.listIterator();
```

- Méthodes :

```
boolean hasNext(); //Check if there is a next element
boolean hasPrevious(); //Check if there is a previous element
E next(); //Get the next element
E previous(); //Get the previous element
void remove(); // Remove the actual element
```

Foreach

- Disponible depuis Java 5

Avant :

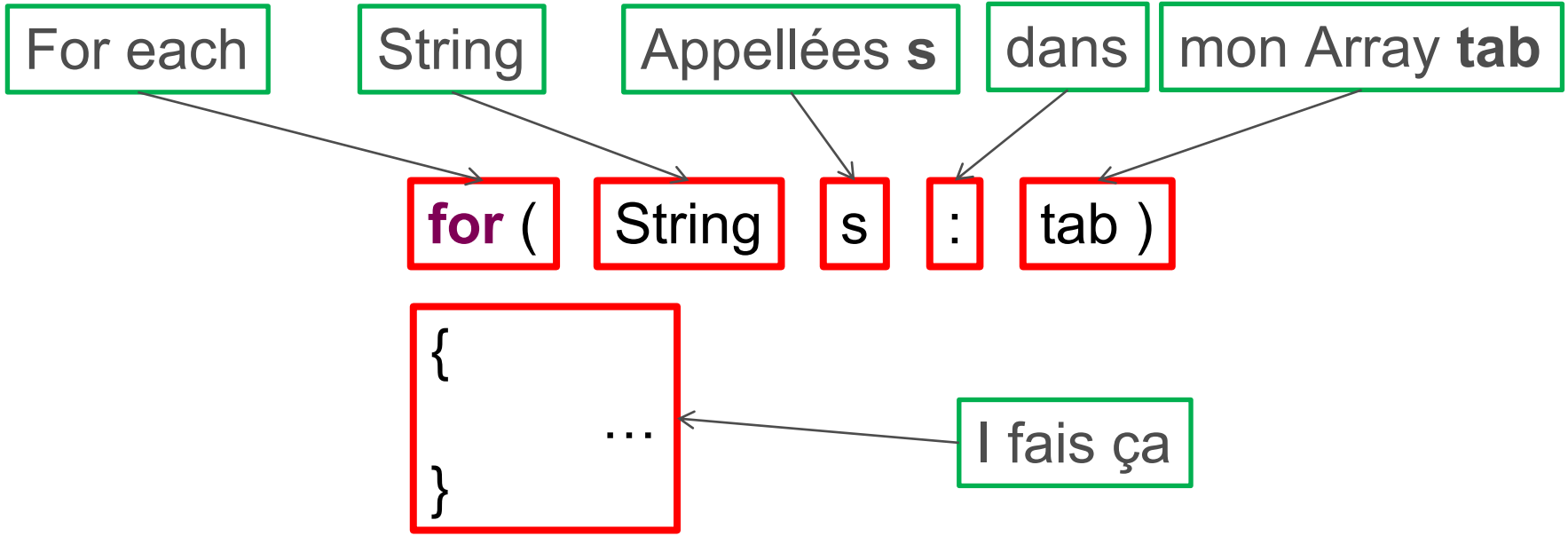
```
String [] tab = {"one",
    "two", "three", "four" };

for(int i = 0; i < 4; i++) {
    System.out.println(tab[i]);
}
```

Maintenant :

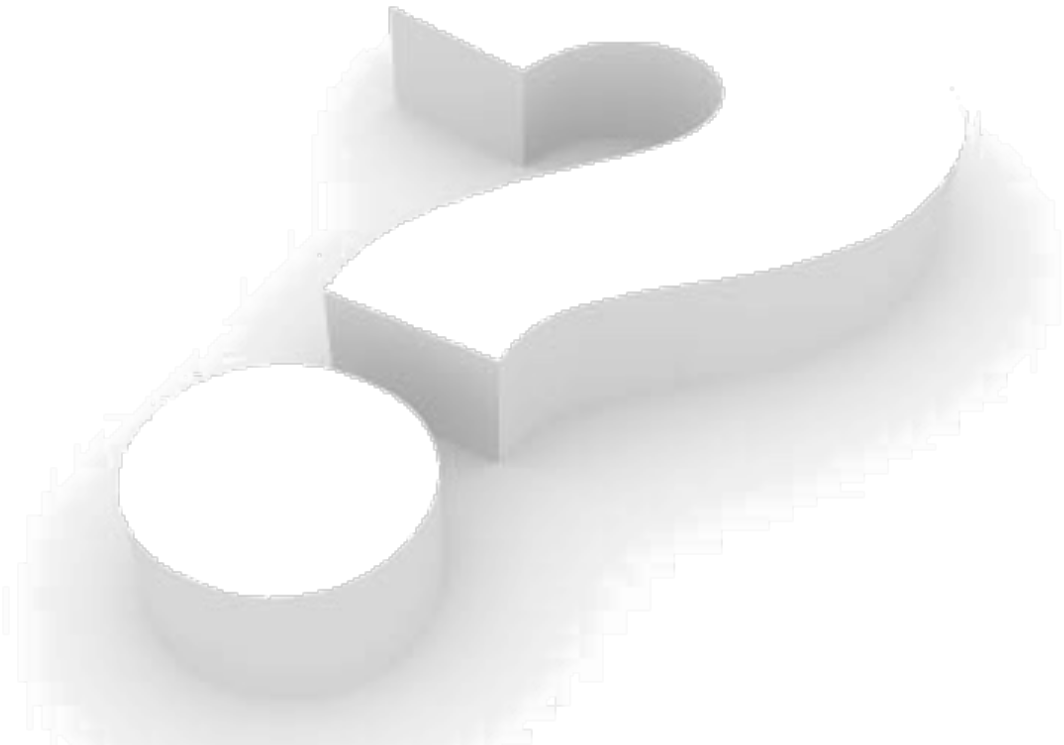
```
String [] tab = {"one",
    "two", "three", "four" };

for(String s : tab) {
    System.out.println(s);
}
```

Foreach

- N'oubliez pas que : "Pour chaque type d'élément que j'appelle xxx dans ma collection/tableau, je fais ça…"



Questions ?



Collections

TRIER UNE COLLECTION

Comparator

- Utilisé pour trier les collections
- Classe implémentant l'interface Comparator<E>
  - Méthode à définir :

```
int compare(E e1, E e2);
```

- La valeur de retour est un entier :
  - Négatif si le premier élément est inférieur au second
  - 0 si égal
  - Positif si le premier élément est supérieur au second

Comparator – Exemple 1/3

- Définir une implémentation :
  - Ici on trie de la plus petite longueur à la plus grande

```
public class MyComparator implements Comparator<String> {
	public int compare(String s1, String s2){
		if(s1.length() == s2.length()) return 0;
		else if(s1.length() > s2.length()) return 1;
		else return -1;
	}
}
```

Comparator – Exemple 2/3

- Trier les éléments à l'intérieur d'un TreeSet :

```
TreeSet<String> mySet = new TreeSet<String>(new MyComparator());
mySet.add("John");
mySet.add("Michael");
mySet.add("Maria");
```

- L'itération affichera le nom de la plus petite longueur à la plus grande : John, Maria, Michael

Comparator – Exemple 3/3

- Trier un élément dans une liste :

```
ArrayList<String> myList= new ArrayList<>();
myList.add("John");
myList.add("Michael");
myList.add("Maria");
Collections.sort(myList, new MyComparator());
```

- L'itération affichera le nom de la plus petite longueur à la plus grande : John, Maria, Michael

Comparable

- Une autre façon de trier les collections
- Classe implémentant l'interface Comparable<E> pour les classes que nous voulons trier
  - Méthode à définir :

```
int compareTo(E e2);
```

Trier une collection

Comparable

```
int compareTo(E e2);
```

- La valeur de retour est un entier :
  - Négatif si l'instance courante est inférieure au paramètre
  - 0 si égal
  - Positif si l'instance courante est supérieure au paramètre
- Les wrappers et String implémentent déjà Comparable

Comparable – Exemple (1/2)

- Définir une implémentation :

```
public class User implements Comparable<User> {
	public String name;

	public User(String name) {
		this.name = name;
	}
	public int compareTo(User u2){
		//Reverse sort by name
		return - name.compareTo(u2.name);
	}
}
```

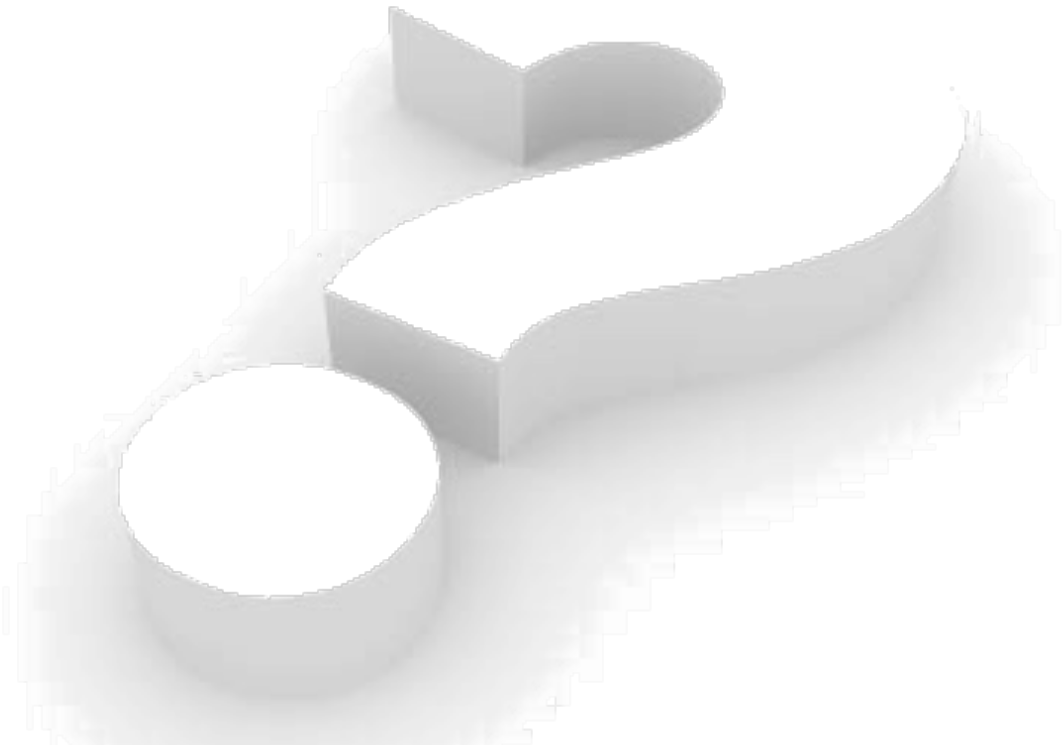
Comparable – Exemple (2/2)

- Trier une collection de Comparable :

```
...
TreeSet<User> mySet = new TreeSet<User>();
mySet.add(new User("John"));
mySet.add(new User("Michael"));
mySet.add(new User("Maria"));
...
```

- L'itération affichera : Michael, Maria, John

Questions ?



Collections

ARRAYS ET COLLECTIONS

Présentation

- JDK propose deux classes utilitaires pour manipuler les Arrays et Collections (et avec des noms très simples):
  - **Arrays**
  - **Collections**

- Ces classes sont exclusivement composées de méthodes statiques
  - Vous ne pouvez pas créer d'instances avec eux ! (Singleton)
- Classes utilitaires aux manipulations courantes

Méthodes communes Collections

- static int binarySearch(List<T>, T) :
  - Recherche dans une liste "triée" une valeur donnée, retourne un index (même si l'élément n'est pas dans la List)
  - Il retourne l'index supposé
- static void reverse(List<T>) :
  - Inverse l'ordre des éléments dans une liste
- static Comparator reverseOrder(Comparator) :
  - Renvoie un Comparator qui trie l'inverse de celui passé en paramètre

Méthodes communes Collections

- static void sort(List<T>, Comparator) :
  - Trie une liste par un comparateur

- Pour en savoir plus, consultez la Javadoc

Méthodes courantes Arrays

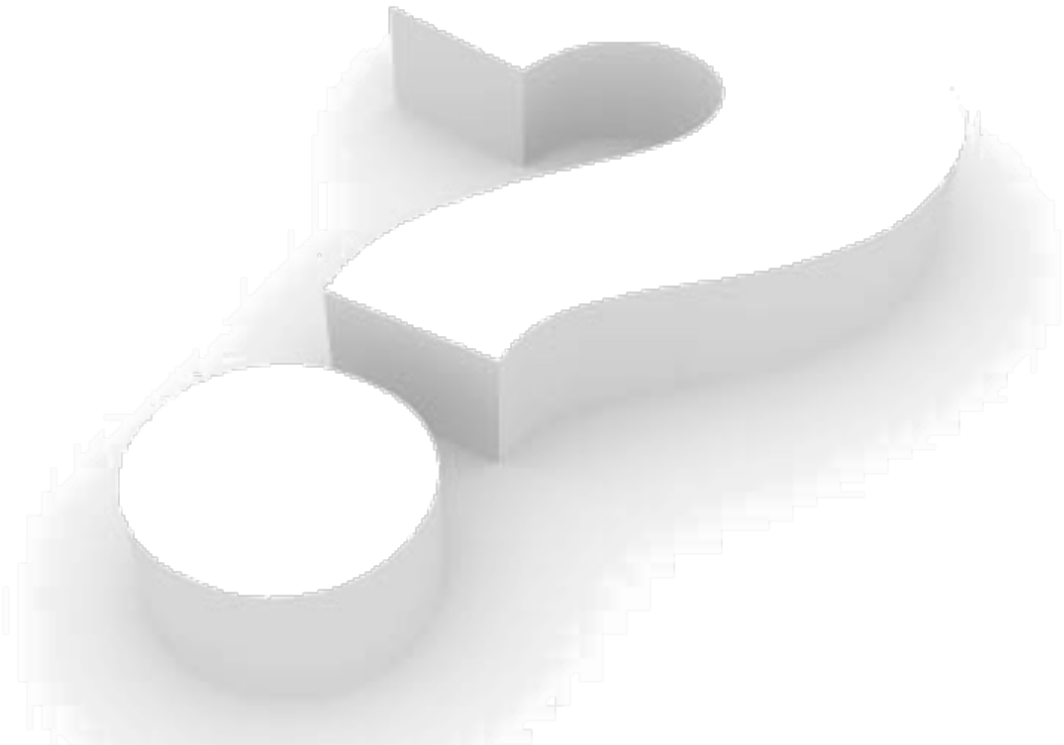
- static List asList(T[]) :
  - Convertit un tableau en List
- static int binarySearch(Object[], Object) :
  - Recherche dans un tableau trié une valeur donnée, renvoie un index
- static void sort(Object[], Comparator) :
  - Trie un tableau par un comparateur

Méthodes courantes Arrays

- static String toString() :
  - Créer une String contenant le contenu d'un tableau

- Pour en savoir plus, consultez la Javadoc

Questions ?



Fin

Merci de votre attention