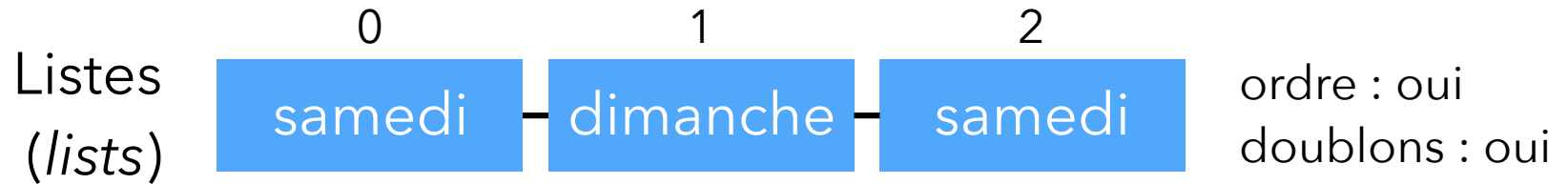


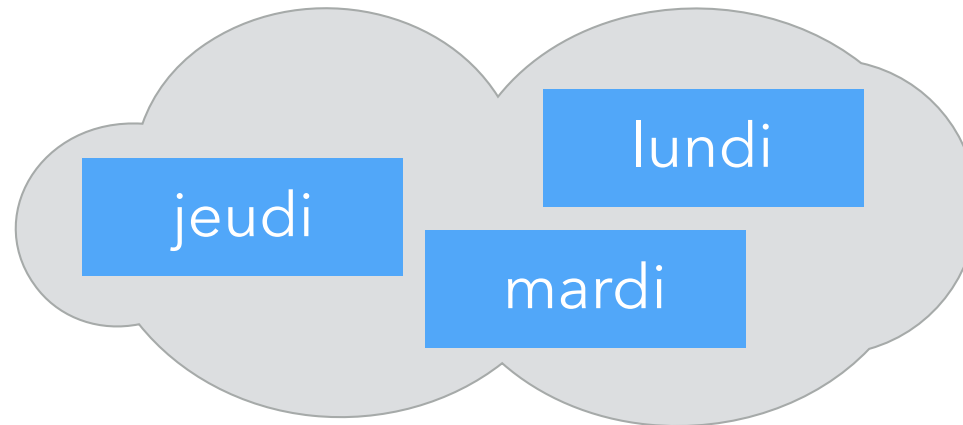
# Collections : introduction, listes

Pratique de la programmation orientée-objet  
Michel Schinz – 2017-03-20

# Collections étudiées

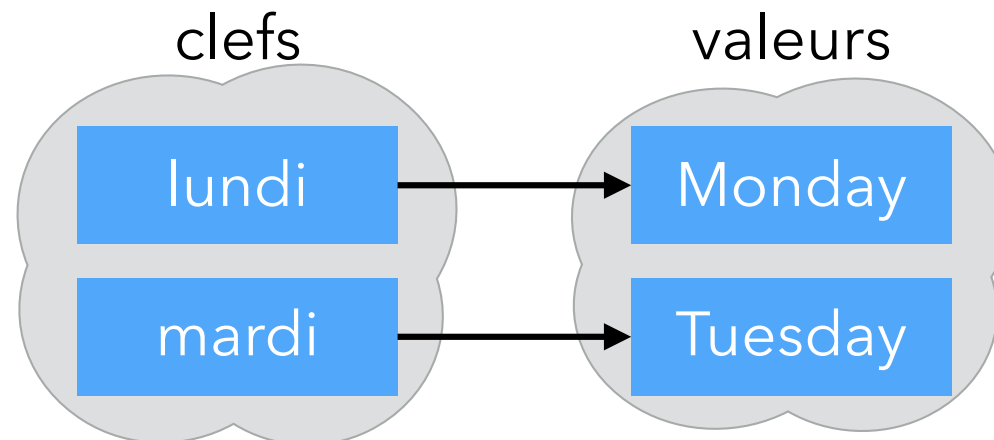


Ensembles  
(*sets*)



ordre : non  
doublons : non

Tables  
associatives  
(*maps*)

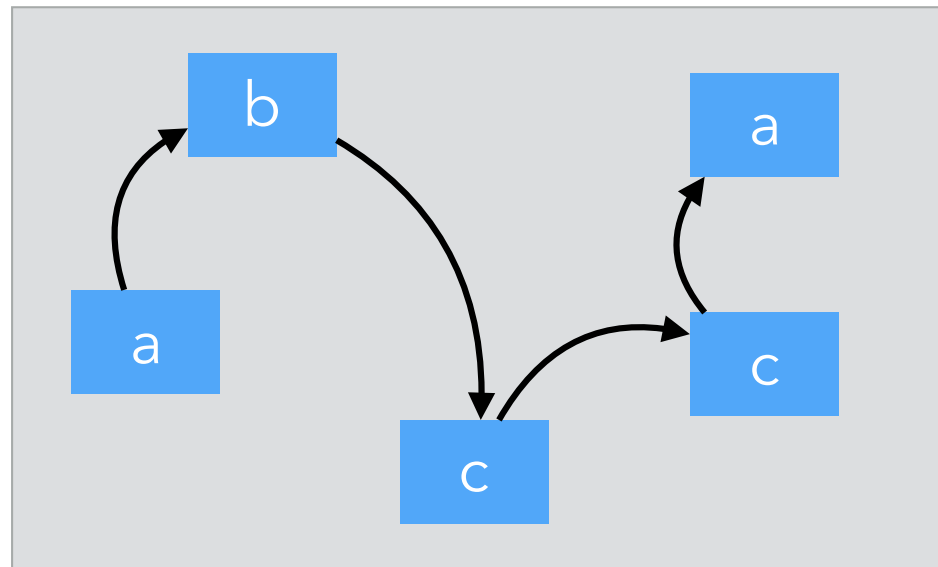


association  
clefs/valeurs

# Mises en œuvre

La représentation en mémoire de la liste  $[a,b,c,c,a]$  dépend de la mise en œuvre choisie :

Liste chaînée  
(LinkedList)



Accès séquentiel  
aux éléments

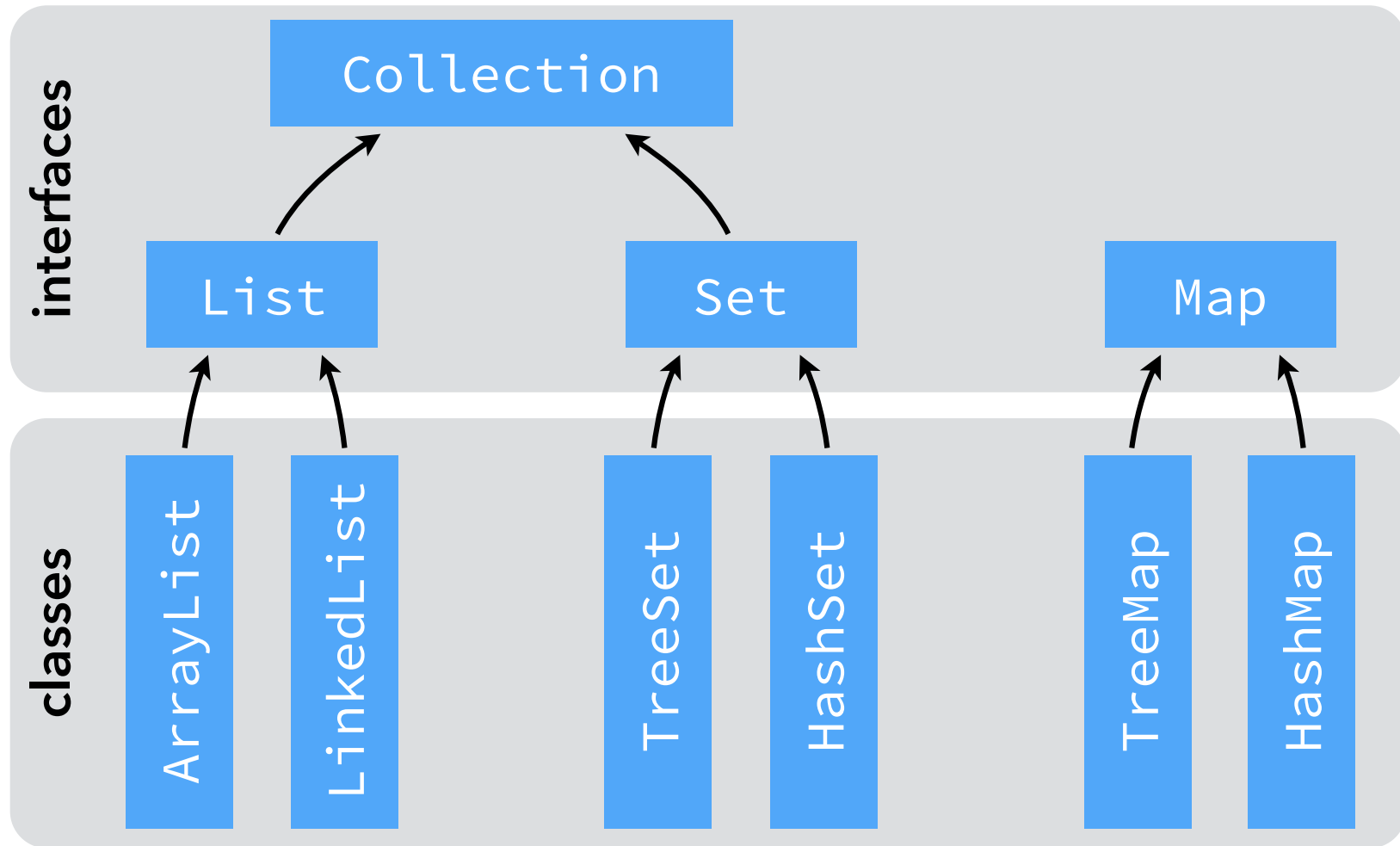
« Tableau-liste »  
(ArrayList)



Accès aléatoire  
aux éléments

# Collections de l'API Java

(vue très partielle et simplifiée du paquetage `java.util`)



+ classes utilitaires **Collections** et **Arrays**

# L'interface Collection

```
public interface Collection<E> {  
    // méthodes de consultation :  
    boolean isEmpty();  
    int size();  
    boolean contains(Object e);  
    boolean containsAll(Collection<E> c);  
  
    // ... à suivre
```

# L'interface Collection

```
// ... suite
```

```
// méthodes d'ajout :
```

```
boolean add(E e);
```

```
boolean addAll(Collection<E> c);
```

```
// méthodes de suppression :
```

```
void clear();
```

```
boolean remove(E e);
```

```
boolean removeAll(Collection<E> c);
```

```
boolean removeAll(Collection<E> c);
```

```
boolean retainAll(Collection<E> c);
```

```
}
```

# L'interface List

```
public interface List<E>
    extends Collection<E> {
    // méthodes de consultation :
    E get(int i);
    int indexOf(E e);
    int lastIndexOf(E e);
    // méthodes d'ajout :
    void add(int i, E e)
    boolean addAll(int i, Collection<E> c);
    // méthodes de suppression :
    E remove(int i);
    E set(int i, E e)
    // ... + quelques autres méthodes
}
```

# Modifiabilité

Toutes les méthodes qui modifient le contenu des collections (add, remove, set, etc.) sont **optionnelles** et peuvent lever `UnsupportedOperationException` !

En général, une collection est soit :

- **non modifiable** : *toutes* ses méthodes de modification lèvent `UnsupportedOperationException`,
- **modifiable** : *aucune* de ses méthodes de modification ne lève `UnsupportedOperationException`

# Mises en œuvre de List

Deux mises en œuvre principales de l'interface `List` sont fournies :

1. `ArrayList`, qui stocke les éléments dans un tableau redimensionné (par copie) au besoin,
2. `LinkedList`, qui stocke chaque élément dans un nœud, liés les uns aux autres.

En conséquence, `ArrayList` fournit un accès aléatoire aux éléments, tandis que `LinkedList` fournit un accès séquentiel.

Ces deux mises en œuvre sont modifiables.

# Mises en œuvre de List

Opération	ArrayList	LinkedList
ajout (add), suppression (remove)	$O(n)$	$O(1)$
accès (get), modification (set)	$O(1)$	$O(n)$

Attention : beaucoup de cas particuliers !

# Vues

La méthode `subList` de `List` permet d'obtenir une vue sur une sous-liste :

```
List<E> subList(int f, int t)
```

La méthode `asList` de `Arrays` permet de voir un tableau comme une liste (de taille fixe) :

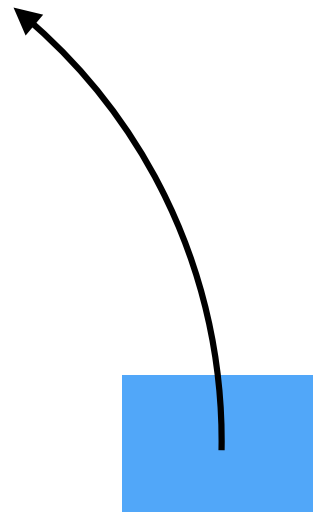
```
<E> List<E> asList(E... a)
```

La méthode `unmodifiableList` de `Collections` permet d'obtenir une vue non modifiable sur une liste :

```
<E> List<E> unmodifiableList(List<E> l)
```

Attention : une vue n'est pas une copie !

# Itérateur



Itérateur désignant le second élément de la liste. Il « sait » comment se déplacer sur l'élément suivant.

# L'interface Iterator

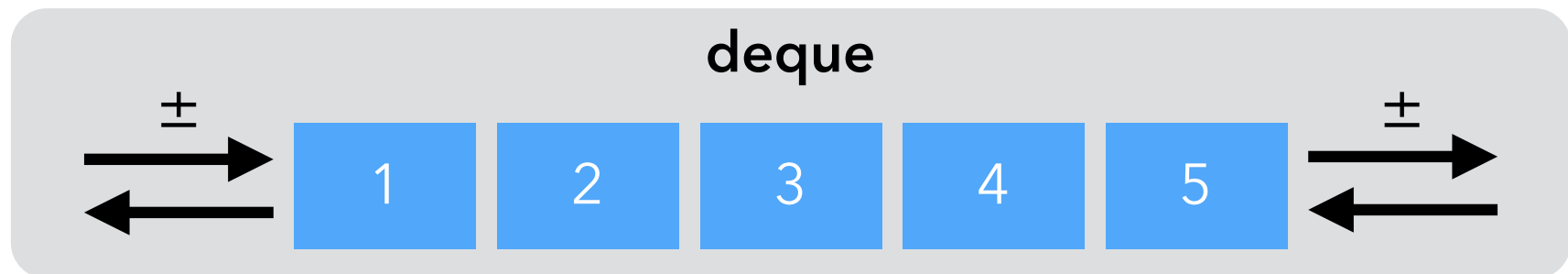
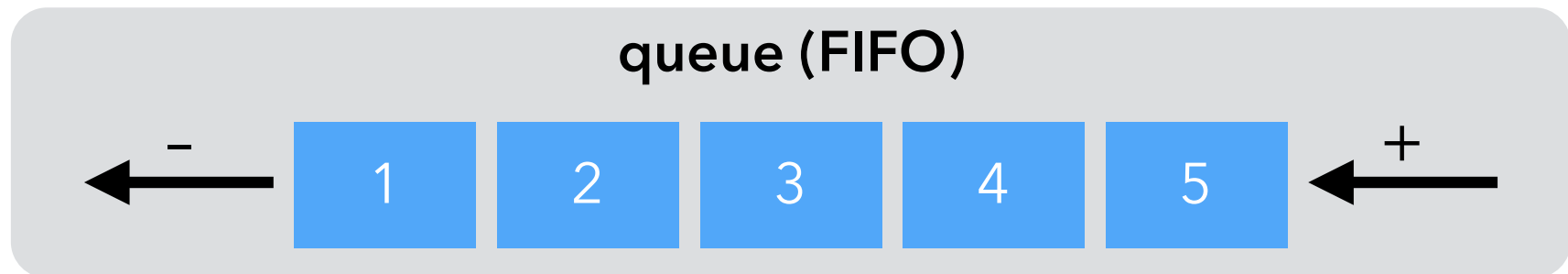
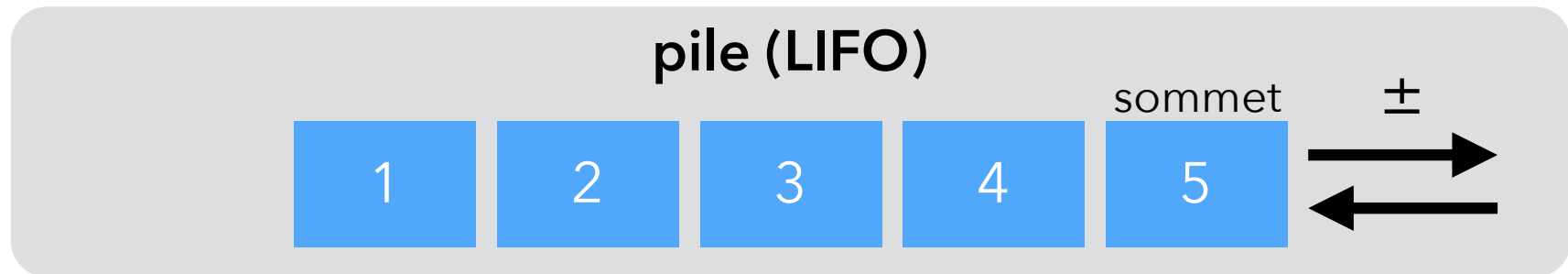
```
public interface Iterator<E> {  
    boolean hasNext();  
    E next();  
    void remove();  
}
```

La méthode `iterator` de `List` permet d'obtenir un itérateur pointant avant le premier élément.

# L'interface Iterable

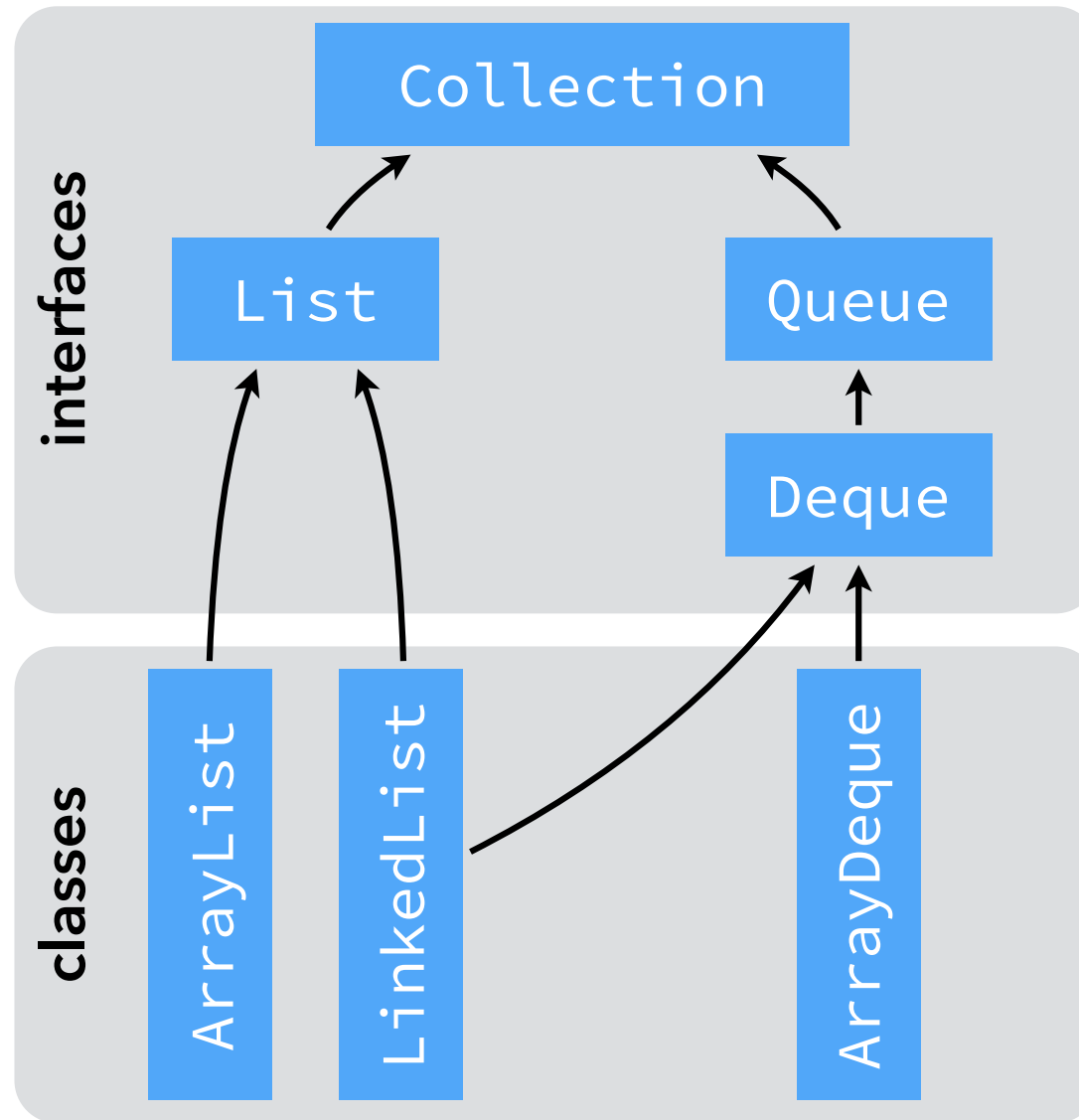
```
public interface Iterable<E> {  
    Iterator<E> iterator();  
  
    default void forEach(Consumer<E> c);  
}
```

# Piles, queues, dequeues



+ : ajout, - : suppression, ± : ajout/suppression

# Piles, queues, dequeues Java



# Règle des listes

---

Pour représenter une pile, une queue ou une « deque », utilisez `ArrayDeque`.

Pour représenter une liste dans toute sa généralité, utilisez `ArrayList` si les opérations d'indexation (`get`, `set`) dominant, sinon `LinkedList`.

---

Note : `ArrayList` peut également s'utiliser comme une pile, pour peu que les ajouts/suppressions se fassent toujours à la fin de la liste et pas au début.