Collections

Arnaud Labourel arnaud.labourel@univ-amu.fr

5 avril 2022

Section 1

Collections

Rôle des collections en Java

Pourquoi des collections d'objets ?

- pour manipuler des ensembles d'objets simultanément,
- pour ranger et accéder à des objets selon les besoins,
- avoir une manière de grouper des objets plus souple que les tableaux.

Une variété de collections

On ne manipule pas toutes les collections de la même façon :

- grouper des valeurs pour les traiter ensemble,
- ordonner des valeurs,
- tester l'appartenance d'éléments,
- créer une file d'attente,
- associer des valeurs entre elles,

Différentes collections existent, plus ou moins efficaces selon les opérations.

Exemples de collection

Exemples d'utilisation de collections dans la vie courante.

- Réfrigérateur : **liste** (*list*) d'aliments mis au frais.
- File d'attente au guichet : **file** (*queue*) de personnes attendant dans l'ordre d'arrivée.
- Pile de T-shirts : **pile** (*stack*) de vêtements avec un ordre et un accès uniquement possible avec le dernier objet empilé.
- Annuaire : **association** (*map*) de noms avec des numéros de téléphone.

Section 2

Liste : l'interface `List`

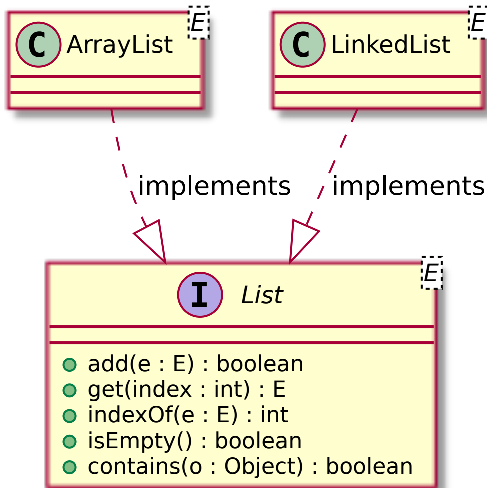
L'interface `List`

Vous avez déjà utiliser `List` : séquence ordonnée indexable par des entiers.

```
public interface List<E> {
  boolean add(E e);
  E get(int index);
  int indexOf(E e);
  boolean isEmpty();
  boolean contains(Object o);
  ...
}
```

Implémentations : `LinkedList`, `ArrayList`, ...

L'interface `List` version diagramme



Section 3

Ensembles : l'interface Set

Exemple d'utilisation d'ensemble

Exemples d'opérations :

- ajouter un ami,
- enlever un ami,
- déterminer si une personne est votre ami,
- déterminer si vous avez au moins un ami

Structure d'ensemble (*Set*)

L'interface Set<E>

```
public interface Set<E> {
  boolean add(E e);
  boolean remove(E e);
  boolean isEmpty();
  boolean contains(Object o);
  ...
}
```

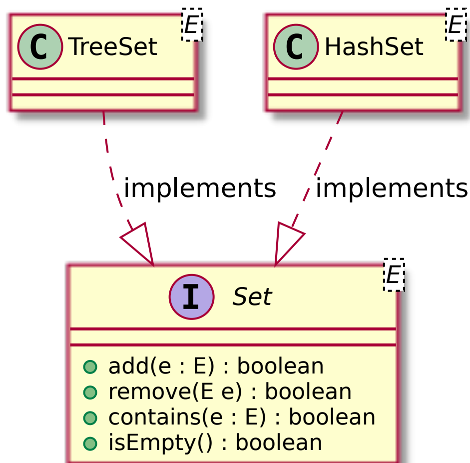
`E` peut être remplacé par le type d'objets de votre choix

Exemple:

`Set<Integer>`, `Set<String>`, `Set<List<Double>>`, ...

Implémentations disponibles : `TreeSet`, `HashSet`

L'interface Set



Type de retour

Pourquoi `add`, `remove` retournent des valeurs `true` ou `false` (de type `boolean`) ?

Règle

Par convention, les méthodes modifiant une collection retourne

- `true` si la modification a été réalisée,
- `false` si aucune modification n'a eu lieu.

Exemple

`false` est retournée par `set.remove(elt)` si `elt` n'était pas dans l'ensemble.

Exemple d'utilisation de Set

```
public Set<String> dictionary(List<String> text) {
  Set<String> words = new HashSet<>();
  for (String word : text) {
    words.add(word);
  }
  return words;
}
```

Remarques

- l'initialisation d'un ensemble vide,
- l'opération d'ajout d'un élément.

Différences entre `List` et `Set`

- un objet `Set` contient chaque élément *au plus une fois*, un objet `List` peut avoir des répétitions,
- un objet `Set` peut **rapidement** accomplir les opérations `add`, `contains`, `remove`. Un objet `List` accomplit `contains` et `remove` **très lentement**.
- les éléments d'un objet `List` sont classés en séquence (0,1,2, . . .) et indicés, les éléments d'un objet `Set` ne sont pas classés. On ne peut pas trier un `Set`.
- `Set` utilise la méthode `equals` de ses éléments, et `HashSet` utilise la méthode `hashCode`, il faut donc les implémenter ou utiliser les méthodes par défaut.

Section 4

Files et piles

File d'attente

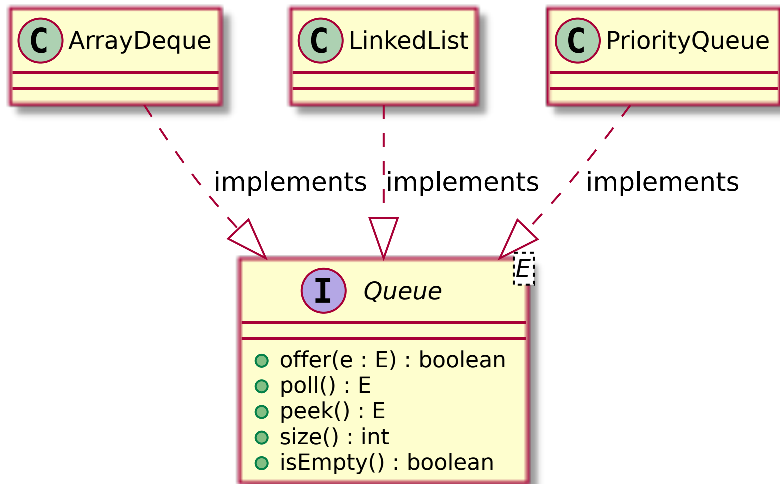
Exemples d'opérations :

- ajouter une personne en fin de file,
- récupérer la personne en début de file,
- tester si la file est vide,
- compter le nombre de personnes en attente.

Structure de file (*Queue*).

Les éléments sortent dans l'ordre où ils sont rentrés (ordre FIFO, *First-in First-out*).

L'interface Queue



Implémentations : `ArrayDeque`, `LinkedList`, `PriorityQueue`

Pile de T-shirts

Exemples d'opérations :

- mettre un T-shirt en haut de la pile,
- récupérer le T-shirt du haut de la pile,
- tester s'il y a un T-shirt.

Structure de pile (*Stack*).

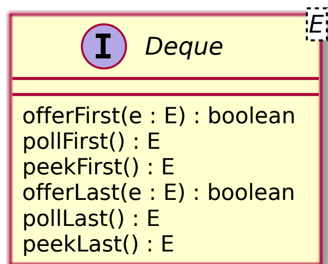
Les éléments sortent dans l'ordre **inverse** où ils sont rentrés (LIFO, *Last-in First-out*).

Piles et files

- **File** : premier entré, premier sorti, une file a deux extrémités actives, une extrémité où l'on rentre et l'autre dont l'on sort.
- **Pile** : dernier entré, premier sorti, une pile a une seule extrémité active (le haut de la pile), à laquelle on ajoute ou retire des éléments.

En Java, **une seule interface commune aux deux : file à double extrémité** (*deque* : abréviation de *double-ended queue*).

L'interface Deque



- `poll` : retirer et retourner l'élément à l'extrémité,
- `peek` : retourner l'élément à l'extrémité, sans le retirer de la deque.

Implémentations : `ArrayDeque`, `LinkedList`

Patron de méthode des files

Traiter tous les éléments de la file, dans l'ordre :

```
public void process(Deque<E> queue) {
  while (!queue.isEmpty()) {
    E first = queue.pollFirst();
    process(first);
  }
}
```

Le traitement d'un élément peut provoquer l'ajout de nouveaux éléments dans la file.

Attention à tester que la file est non-vide avant d'extraire un élément ! (sinon vous obtiendrez `null`).

Exemple d'utilisation d'une Deque

Calcul des scores possibles au rugby (< 100) :

- Au rugby on peut marquer soit 3 points, soit 5 points, soit 7 points d'un coup.
- Le score d'une équipe ne peut donc pas être un seul point, ou 2 points, ou 4 points, mais peut être 6 points $(3+3)$, 8 points $(3+5)$, 9 points $(3+3+3)$, ...
- Quels sont les scores réalisables ?
- Pour chaque score n réalisable, $n+3$, $n+5$ et $n+7$ sont aussi réalisables. 0 est réalisable.

Exemple d'utilisation d'une Deque

```
Set<Integer> scores = new HashSet<>();
public void computeScores() {
  Deque<Integer> feasibles = new ArrayDeque<>();
  feasibles.offerLast(0);
  while (!feasibles.isEmpty()) {
    addScore(feasibles.pollFirst(), feasibles);
  }
}
public void addScore(int n, Deque<Integer> feasibles) {
  if (n >= 100 || scores.contains(n)) { return; }
  scores.add(n);
  feasibles.offerLast(n+3);
  feasibles.offerLast(n+5);
  feasibles.offerLast(n+7);
}
```

Exemple d’utilisation d’une Deque

- `offerLast` + `pollFirst` : utilisation de la deque comme une `file`.
- Instanciation avec `new ArrayDeque<>()`.
- La file contient les scores réalisables.
- Si un score est déjà connu, on le saute.

Section 5

Tables d'associations

Annuaire

Exemples d'opérations :

- ajouter un contact (nom + numéro),
- chercher le numéro d'un contact,
- retirer un contact.

Structure de table d'associations (`Map`), en Python : `dict`.

L'interface Map

```
public interface Map<K,V> {
  Value put(K k, V v);
  boolean containsKey(K k);
  Value get(K k);
  Value remove(K k);
  Set<Key> keySet();
  Collection<Value> values();
  ...
}
```

De nouveau, on peut remplacer `K` et `V` par n'importe quels types.

Implémentations : `HashMap`, `TreeMap`

Exemple d'utilisation de `Map`

On veut compter le nombre d'appels téléphoniques effectués vers chaque numéro.

- Associer à chaque numéro le nombre d'appel.
- Lorsqu'un appel est effectué, récupérer le nombre d'appels associé et l'augmenter de 1. Cas spécial : premier appel.
- Notez bien dans le prochain transparent : l'initialisation, l'ajout d'une association, le test d'une clé, la récupération d'une valeur.

Exemple d'utilisation de Map

```
public class PhoneCallStats {
  Map<PhoneNumber,Integer> callCounts = new HashMap<>();

  public void call(PhoneNumber number) {
    int oldCount = this.getCallCount(number);
    callCounts.put(number,oldCount+1);
  }

  public int getCallCount(PhoneNumber number) {
    if(callCounts.containsKey(number))
      return callCounts.getNumber();
    return 0;
  }
}
```

Section 6

Collections

L'interface `Collection`

`Set`, `Queue`, `Deque`, `List` **étendent** (*extends*) l'interface `Collection` (**mais pas `Map` !**).

Ce qui signifie que toutes ces interfaces possèdent aussi les services définies par l'interface `Collection`.

```
public interface Set<E> extends Collection<E> {
  ...
}
```

Extension

Une interface `A` étends une interface `B` :

- `interface A extends B` à la déclaration de l'interface.
- l'interface `A` définit toutes les méthodes de l'interface `B`.

L'interface `Collection`

```
public interface Collection<E> {
  boolean add(E e);
  boolean remove (E e);
  int size();
  Iterator<E> iterator();
  ...
}
```

Définition d'une collection dans votre programme :

```
Collection<Item> myItems = new ArrayList<Item>();
```

Itérer sur une collection

Comment faire une manipulation pour chaque item d'une collection ?

La boucle `for` permet de faire les mêmes instructions pour tous les éléments d'une collection.

```
Collection<Item> itemList = ...;
for (Item item : itemList) {
  ... // instructions utilisant item
}
```

Les éléments sont traités un par un.

Ordre des éléments : dépend de l'implémentation de la collection.

Syntaxe du `for` (`foreach`)

```
for (Item item : collection) { ... }
```

- `for` : mot-clé de répétition,
- les paramètres de la répétition :
  - `Item` : le type des éléments de la collection,
  - `item` : un identifiant pour une **nouvelle variable** désignant l'élément de la collection, qui change à chaque itération,
  - `:` se lit *in* (contenu dans),
  - `collection` : la collection sur laquelle itérer.
- `...` : les instructions à répéter pour chaque élément de la collection.

Interrompre une itération

Il peut être nécessaire d'interrompre l'itération au milieu de la collection.

- `return` fait sortir de la méthode, donc de l'itération.
- `break` fait sortir de l'itération. La prochaine instruction évaluée est celle suivant la fin de la boucle `for`.

Exemple d'interruption

- Choisir un objet avec une probabilité proportionnelle à son poids.
- totalWeight est la somme des poids des objets.

```
int s = randomGen.nextInt(totalWeight);
Item selected = null;
for (Item item : itemList) {
  s = s - item.weight;
  if (s <= 0) {
    selected = item;
    break;
  }
}
selected.display();
// prochaine instruction après break
```

Interrompre une itération

Il peut être nécessaire de sauter certains éléments d'une collection.

- `continue` arrête le traitement de l'élément en cours et passe à l'élément suivant de l'itération. La prochaine instruction évaluée est la première instruction du bloc de la boucle avec l'élément suivant (s'il reste au moins un élément).

```
int sum = 0;
int product = 1;
for (Item item : items) {
  if (item.isBad()) { continue; }
  sum = sum + item.value();
  product = product * item.value();
}
```

Patron de méthode : itérer une opération

Modèle pour faire une opération sur chaque élément.

```
public void iterateProcess(Collection<Elt> elements) {
  initialise();
  for (Elt element : elements) {
    process(element);
  }
  return;
}
```

Résumé des interfaces des collections et map

nom	interface	instanciation
liste	List	new ArrayList<>()
pile/file	Deque	new ArrayDeque<>()
ensemble	Set	new HashSet<>()
table d'associations	Map	new HashMap<>()

Les noms des interfaces sont accompagnés du type des éléments : List<Integer>, Set<Node>, Map<Node, Point2D>, ...

Entre <...>, on utilise Integer, Double, Boolean plutôt que int, double, boolean.