Les conteneurs

Le package *java.util* ¶ §

Résumé du package

Le package `java.util` regroupe des classes relatives à la gestion du temps, à la gestion des événements, ainsi qu'aux conteneurs (collections et tableaux associatifs).

Un **conteneur** est un ensemble générique d'objets (jeu de cartes, répertoire de fichiers, sac de perles, annuaire téléphonique, dictionnaire, etc.).

L'efficacité du codage particulier d'une collection dépend de ses caractéristiques (ordre, doublons, etc.) et des opérations envisagées (parcours, ajout, suppression, concaténation, etc.).

Conteneurs

Le package `java.util` propose les conteneurs suivants:

- `ArrayList`, `ArrayDeque`
- `LinkedList`
- `HashMap`, `HashSet`
- `TreeMap`, `TreeSet`.

(Cette liste n'est pas exhaustive).

On utilise le plus souvent `ArrayList`, `HashSet`, `HashMap`.

Ex.1. Conteneurs (10 min)

- Parcourez la documentation relative aux conteneurs citées.
- Quelle est la différence entre `Collection`, `List`, `Set`, `Map`?
- Quelle est la différence entre `ArrayList` et `LinkedList`?
- Quelle est la différence entre `HashMap` et `TreeMap`?

L'interface `Collection`

Les interfaces `List` et `Set` dérivent de `Collection`, dérivant elle-même de `Iterable`.

Elle exige la présence, entre autres, des méthodes suivantes:

- `contains(Object o)`
- `isEmpty()`

- `toArray()`
- `iterator()`

La méthode `iterator()`, requise par `Iterable`, renvoie un objet issu d'une classe implémentant l'interface `Iterator`, avec lequel on peut parcourir la collection.

L'interface `Iterator`

Un objet de type `Iterator` offre un moyen de parcourir la collection à laquelle il réfère:

```
Collection<Integer> c = new ArrayList<Integer>();
...
for (Iterator<Integer> it = c.iterator(); it.hasNext(); ) {
    System.out.println( it.next() );
}
```

Depuis java 5, vous pouvez utiliser ce raccourci:

```
for (Integer i: c) {
    System.out.println( i );
}
```

Généricité

Comme vous le voyez, depuis java 5, les classes et les interfaces peuvent être paramétrées par un type.

```
public class StackByLinkedList<E> {
    ...
}
```

Lors de la déclaration, le type formel est remplacé par le type effectif (non primitif).

```
StackByLinkedList<Integer> s = new StackByLinkedList<Integer>();
```

Le fonctionnement est le même que celui du passage de paramètres pour une méthode.

L'interface `Map`

L'interface `Map` décrit des objets qui mettent en correspondance des clés et des valeurs (à une clé étant associé au plus une valeur).

En plus de la méthode `get()` renvoyant la valeur associée à une clé donnée, elle offre trois vues de type `Collection`:

- l'ensemble de clés est renvoyé par la méthode `keySet()`,
- la collection de valeurs est renvoyé par la méthode `values()`,
- l'ensemble de paires clé-valeur est renvoyé par la méthode `entrySet()`.

L'interface `Map.Entry`

```
Map<Integer,String> annuaire = new HashMap<Integer,String>();
```

L'interface `Map.Entry` représente une paire clé-valeur: `getKey()` retourne la clé, tandis que `getValue()` retourne la valeur.

```
Iterator<Map.Entry<Integer,String> > it;
for (it = annuaire.entrySet().iterator(); it.hasNext(); ) {
    Map.Entry<Integer,String> e = it.next();
    System.out.println(e.getKey() + ": " + e.getValue());
}
```

```
for (Map.Entry<Integer,String> e: annuaire.entrySet()) {
    System.out.println(e.getKey() + ": " + e.getValue());
}
```

Exemple d'application

Nous allons développer un programme qui résoud une grille de sudoku de 9 par 9 en utilisant intensivement les conteneurs de type `List`, `Set` et `Map`.

Commençons par nous mettre d'accord sur les mots. La grille de sudoku comprend 81 positions (**squares**), regroupées en unités (**units**) : ligne (**row**), colonne (**column**), régions 3x3 (**box**). Une position donnée appartient à 3 unités (une ligne, une colonne, un région 3x3) contenant toutes les positions amies (**friends**) de cette position.

Ex.2. Positions et unités (10 min)

La classe **`Square`** modélise une position comme une paire de deux indices entre 0 et 8. Une position peut notamment renvoyer une représentation textuelle (concaténation du chiffre des deux indices), renvoyer la ligne, la colonne, la région à laquelle elle appartient.

La classe abstraite **`Unit`** modélise une unité dont on peut obtenir les 9 positions qu'elle recouvre, comme une liste de positions. Les classes **`Row`**, `Col` et **`Box`** héritent de `Unit`.

Complétez la classe `Row` et codez la classe `Col`.

Ex.3. Ensemble de chiffres possibles (10 min)

Nous allons ensuite distinguer la *grille originale* (pour certaines positions, un chiffre entre 1 et 9 est donné), de la *grille de travail* (à chaque position, il y a un *ensemble de chiffres possibles*).

Nous allons compléter maintenant la classe **`DigitSet`** qui modélise un ensemble de 1 à 9 chiffres possibles.

Ex.4. Structures de données (10 min)

La classe **`SudokuSolver`** à compléter possède plusieurs structures de données dont les clés sont les positions:

- *originalGrid*, l'association entre positions et entiers affectés à ces positions au départ,
- *workingGrid*, l'association entre positions et ensemble de chiffres possibles pour ces positions, sur laquelle on travaille.
- *friends*, l'association entre toutes les positions et la liste de leurs positions amies,
- *marks*, l'ensemble des positions auxquelles le *solver* a déjà affecté une valeur.

Complétez la méthode *fillExtraDataStructures* de manière à initialiser les champs *workingGrid* et *friends*.

Ex.5. Démonstrateur (10 min)

Ecrire une classe exécutable `DemoSudoku` dans laquelle vous:

- instanciez `SudokuSolver` en lisant la grille sur l'entrée standard,
- affichez la grille originale, puis la grille de travail sur la sortie standard.

Téléchargez les fichiers **`easy1.txt`** et **`hard1.txt`** puis, après avoir compilé le tout dans un répertoire `build`, tapez la commande `java -cp build SudokuSolver < easy1.txt`.

Ex.6. Résolution

- Codez la méthode *solve* de `SudokuSolver` (10 min).
- Cette approche par propagation de contraintes n'aboutit pas toujours. Pour améliorer ce solver, vous pourrez implémenter une recherche inspirée de cette solution: choisir une position non résolue (mais dont le nombre de possibilités est le plus petit), puis essayer chacune des possibilités. A chaque fois, qu'une possibilité mène à une solution invalide, revenir en arrière.

Ce qu'il faut retenir

- Un conteneur est un ensemble générique d'objets.
- On distingue deux types:
    - `Collection` dont dérivent:
        - `List` (ensemble ordonné)

        - `Set` (ensemble sans doublon)
    - `Map` (ensemble de paires clés-valeurs)
- `Collection`, `List`, `Map`, `Set` sont des interfaces, seules les classes concrètes qui les implémentent sont instanciables.
- Les objets de type `Collection` sont parcourus d'une manière uniforme (iterator ou boucle for étendue). Les objets de type `Map` offrent trois vues de type `Collection`.