Programmation orientée objet
TP4 - Collections

L3 MIAGE

19 octobre 2023

1 Collections de la guerre des étoiles

Reprenez votre code (ou celui de la correction) sur la guerre des étoiles. Vous devriez maintenant disposer des fichiers suivants : `Arme.java` , `Blaster.java`, `Jedi.java` , `Personnage.java`, `Sabre.java` et `SoldatClone.java`.

1.1 Générer des soldats clones

Écrire une méthode statique `genererListClone` dans la classe `SoldatClone` afin d'instancier `n` soldats clones ayant pour nom *Clonei* où i est le i-ème soldat clone instancié. Pour cela, utiliser la collection `ArrayList` ou `LinkedList`. Cette méthode retourne la collection.

```
static List<SoldatClone> genererListClone(int n);
```

1.2 Afficher une collection

Dans un programme test, ajouter les instructions suivantes pour instancier une collection de 1000 clones.

```
List<SoldatClone> listClone = SoldatClone.genererListClone(1000);
System.out.println(listClone);
```

Que s'est-il passé ? Pourquoi ?

1.3 Des clones en moins

Dans ce même programme test, modifier la collection afin de supprimer tous les clones ayant pour nom *Clonei* où i est pair. Afficher à nouveau la collection pour vérifier que l'opération a bien été effectué, mais cette fois-ci, utiliser un « foreach ».

1.4 Générer des jedis

Écrire une méthode statique `genererListJedi` dans la classe `Jedi` afin d'instancier n jedis ayant pour nom *Jedik* où k est le k-ième jedi instancié. Chaque jedi aura une force générée aléatoirement entre 10 et n. Pour cela, utiliser la classe `Random` et la méthode `nextInt(limite)` de la librairie `Java`.

```
static List<Jedi> genererListJedi(int n);
```

1.5 Trier les jedis, la débrouille

On souhaite trier les jedis du moins fort au plus fort, c-à-d de les trier en fonction de leur force. Coder une méthode statique `trier` qui étant donné une collection de type `List<Jedi>` retourne une `List<Jedi>` trié.

```
public static List<Jedi> trier(List<Jedi>)
```

1.6 Qui est le plus fort des Jedis ?

Heureusement, l'exercice précédent n'est plus à refaire puisque la bibliothèque `Java` permet déjà de trier les éléments d'une collection. Pour cela, nous proposons d'utiliser la méthode statique suivante permettant de trier des collections de type `List` :

```
Collections.sort(List<T>, Comparator<T>)
```

Nous avons déjà la liste mais nous devons créer notre propre classe `Comparator<T>` afin de comparer des objets de type `Jedi`. Pour cela, créer une classe `JediComparator` implémentant l'interface `Comparator<T>`. Dans cette classe, il faut redéfinir la méthode `int compare(T o1, T o2)`. Cette méthode retourne 0 si les objets sont égaux, un nombre négatif si `o1 < o2`, et un nombre positif sinon.
Attention `Comparator<T>` est un générique, il faut donc spécifier le bon type correspondant aux éléments de notre collection.

Vérification : pour tester que tout fonctionne correctement, générer une collection de 10 jedis dans un programme test et ajouter les instructions suivantes avant d'afficher le contenu de la collection :

```
Collections.sort(listJedi, new JediComparator());
```

Question : pourquoi n'a-t-on pas utilisé la collection `TreeSet` ?

1.7 Générer des soldats clones

Surdéfinir (surcharger/overload) la méthode `genererListClone` de la classe `SoldatClone` afin de générer n clones avec une force générée aléatoirement entre 10 et `alea`.

```
static List<SoldatClone> genererListClone(int n, int alea);
```

1.8 Qui est le plus fort des clones ?

Comme pour les jedis, on souhaite trier les soldats clones du moins fort au plus fort, c-à-d de les trier en fonction de leur force. Pour ce faire, nous proposons une autre solution qui ne passe pas par la création d'une classe `Comparator<T>`.
Voici les étapes à suivre :

1) votre classe `SoldatClone` doit implémenter l'interface `Comparable<T>` (attention au type du générique).

2) votre classe doit redéfinir la méthode `int compareTo(Object o)` qui retourne 0 si les objets sont égaux, un nombre négatif si l'objet appelant la méthode est plus petit que `o2` et un nombre positif sinon.

Vérification : pour tester que tout fonctionne correctement, générer une collection de 10 soldats clones dans un programme test et ajouter les instructions suivantes avant d'afficher le contenu de la collection :

```
Collections.sort(listClones);
```

1.9 Des clones de même force

Nous souhaitons trier les clones possédant la même force. Soit `o1` et `o2` deux clones de même force. Nous disons que `o1 < o2` si et seulement si `o1` a été généré avant `o2` par la méthode `genererListClone`.
Exemple :

- Clone14 < Clone55
- Clone88 > Clone2

Modifier votre méthode `compareTo` pour satisfaire les exigences précédentes. Tester ensuite les modifications depuis un programme test.

1.10 Dictionnaire de jedis

Écrire une méthode statique `genererMapJedi` dans la classe `Jedi` afin d'instancier n jedis ayant pour nom *Jedik* où k est un entier généré aléatoirement entre 1 et 1000. Pour cela, utiliser la collection `HashMap<K, V>` où les clefs sont de type `String` et correspondent au nom des jedis.

```
static Map<String, Jedi> genererMapJedi(int n);
```

1.11 Parcourir un dictionnaire

Pour afficher le contenu du dictionnaire de `Jedi`, il est possible d'appeler la méthode `toString` défini dans la classe `HashMap`. Celle-ci va directement faire appel à la méthode `toString` défini dans les classes du type des éléments présents dans la collection.

```
Map<String, Jedi> mapJedi = Jedi.genererMapJedi(10);
System.out.println(mapJedi);
```

Il est aussi possible de parcourir le dictionnaire avec une boucle « foreach ».
Pour parcourir seulement les valeurs de la collection :

```
for (Jedi j : mapJedi.values()) {
  System.out.println("valeur: " + j);
}
```

Pour parcourir seulement les clefs de la collection :

```
for (Jedi j : mapJedi.keySet()) {
  System.out.println("clef: " + j);
}
```

Pour parcourir le dictionnaire par pair de clef-valeur :

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

Question : nous remarquons que la méthode `keySet` retourne une collection de type `Set`. Pourquoi ?

1.12 Un ensemble de Jedi

Écrire une méthode statique `dictVersEnsemblePair` dans la classe `Jedi` qui étant donné un dictionnaire de Jedi retourne une collection de type `Set` contenant des Jedis ayant pour nom *Jedik* où k est un nombre premier. Pour rappel, un nombre premier est un entier positif strictement supérieur à 1 qui n'est divisible que par 1 et par lui même.

```
public Set<Jedi> dictVersEnsemblePair(Map<String, Jedi> mapJedi);
```

1.13 Bonus : manipuler des ensembles

Refaire tous les exercices précédents sur les listes afin de manipuler des collections de type `Set`. Pour le code de hachage, nous proposons de le calculer à partir du nom et de la force du personnage. Ainsi, nous supposerons que deux personnages de même force et de même nom sont identiques.