Université Paris Diderot **Programmation Orientée Objet**
L2, EIDD et M1 linguistiques **Année 2015-2016**

TD n° 10

Collections

Exercice 1 Modéliser

On rappelle quelques interfaces de collections importantes :

- `List<E>` : liste d'éléments avec un ordre donné, accessibles par leur indice.
- `Set<E>` : ensemble d'éléments sans doublons
- `Map<K,E>` : ensemble d'associations (clé dans `K`, valeur dans `E`), tel qu'il n'existe qu'une seule association faisant intervenir une même clé.

Ces interfaces peuvent évidemment être composées les unes avec les autres. Exemple : un ensemble de séquences d'entiers se note `Set<List<Integer>>`.

Pour les situations suivantes, déclarez le type de la structure de données qui peut les modéliser :

1. Donnée des membres de l'équipe de France de Football.
2. Donnée des membres de l'équipe de France de Football avec leurs rôles respectifs dans l'équipe.
3. Marqueurs de buts lors du dernier match (en se rappelant la séquence).
4. Affectation des étudiants à un groupe de TD.
5. Pour chaque groupe de TD, la "liste" des enseignants.
6. Pour chaque groupe de TD, affectation de l'enseignant pour chaque UE.
7. Étudiants présents lors de chaque séance de TD de POOIG du semestre.

Exercice 2 Bibliothèque

Nous voulons programmer un logiciel de gestion de bibliothèque. Voici la situation dans ses grandes lignes :

- Une bibliothèque possède un certain nombre d'ouvrages, certains en plusieurs exemplaires.
- Une bibliothèque a une certaine "liste" de personnes abonnées.
- Chaque abonné peut emprunter des ouvrages (avec limite du nombre d'ouvrages et de la durée)
- La bibliothèque garde l'historique de tous les emprunts sur un an.

Par ailleurs, pour une gestion efficace, les bibliothécaires souhaitent disposer des fonctionnalités suivantes :

- Mettre à jour l'historique en effaçant les entrées datant de plus d'un an.
- Obtenir les ouvrages disponibles (au moins 1 exemplaire non prété).
- Obtenir la "liste" des abonnés ayant du retard dans leurs emprunts.
- Suggérer des "amis" à un abonné (personnes ayant emprunté un livre en commun avec cet abonné). Stocker (mettre en cache) le résultat de la recherche (pour donner un résultat plus rapidement la prochaine fois).

Modélisez le système de gestion de bibliothèque en UML.

Programmez les fonctionnalités suggérées.

Exercice 3 Itérateurs – programmez votre propre collection !

Pour pouvoir parcourir un objet paquet à l'aide d'une boucle *for each* (**for** (T obj: paquet) ...), il suffit que cet objet implémente l'interface Iterable<T> :

```
1 | public interface Iterable<T> {
2 |  Iterator<T> iterator();
3 | }
```

L'interface Iterator<T> étant elle-même :

```
1 | public interface Iterator<T> {
2 |  boolean hasNext();  // retourne s'il reste des éléments à parcourir
3 |  T next(); // retourne l'élément suivant
4 | }
```

1. Créez un itérateur (**class** IteFibo **implements** Iterator<Integer>) qui retourne les premiers termes de la suite de Fibonacci ($u_0 = 1; u_1 = 1; u_{n+2} = u_{n+1} + u_n$) jusqu'à une borne passée en paramètre du constructeur.
   Créez ensuite une classe **class** Fibonacci **implements** Iterable<Integer> telle que l'instruction

   ```
   1 | for (Integer x: new Fibonacci(20)) System.out.println(x);
   ```

   affiche les 20 premiers termes de la suite de Fibonacci.
2. (Plutôt pour le TP) Faites en sorte que la classe Arbre<T> du TD précédent implémente Iterable<T> en implémentant le parcours des nœuds de l'arbre dans l'ordre préfixe. Attention, la méthode next() parcourant un seul élément à chaque appel, le style récursif se prête mal à cet exercice. Il faut donc trouver un autre moyen de se rappeler ce qui reste à parcourir.

Note : Iterable<T> est la collection la plus simple mais on peut aussi faire en sorte que ces classes implémentent, par exemple, List<T> (beaucoup plus long).

Exercice 4 Généricité et héritage
On définit :

```
1 | class Base {
2 |     public String toString() { return "Base"; }
3 | }
4 | class Derivee extends Base {
5 |     public String toString() { return "Derivee"; }
6 | }
```

Observez le programme suivant :

```
1 | public class Test {
2 |   public static void main(String[] args) {
3 |     ArrayList<Derivee> l1 = new ArrayList<Derivee>();
4 |     Arraylist<Base> l2 = l1;
5 |     l2.add(new Base());
6 |     System.out.println(l1.get(0));
7 |   }
8 | }
```

1. Supposons que ce programme compile, que devrait-il afficher ? En quoi est-ce choquant ?
2. Quelle ligne contient d'après vous une erreur ? De quelle erreur s'agit-il ? Conclure en expliquant pourquoi Java interdit cela.