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

TP n° 2

Domaines d'accès, associations de classes, encapsulation

Dans ce TP on se familiarise avec les classes et l'encapsulation.

1 Concrétiser le TD

Afin d'implémenter et de tester ce qui a été conçu en TD :

1. Programmer les classes et méthodes des exercices du TD 2 quand cela n'a pas déjà été fait, sinon les recopier et en les corrigeant si nécessaire.
2. Y ajouter les constructeurs nécessaires et les méthodes toString qui n'ont pas été explicitement demandés.
3. Programmer les tests suggérés (éventuellement en tant que méthodes supplémentaires).
4. Dans chaque classe, ajouter un main() qui lance tous les tests.
5. Pour chaque exercice/projet, programmer une classe principale avec un main() qui exécute un petit scénario utilisant les méthodes programmées dans l'exercice.

Ces étapes ne sont pas forcément à réaliser dans l'ordre. Essayez d'obtenir le plus vite possible un programme qui compile et ajoutez lui ses fonctionnalités de façon incrémentale, en même temps que les tests correspondants.

2 Ensembles et Piles

Exercice 1 Ensembles d'entiers

Dans cet exercice on veut construire une classe `Ensemble` qui représente un ensemble d'entiers. C'est à vous de choisir votre implémentation.

En sachant que :
— Il ne peut y avoir deux fois la même valeur dans l'ensemble (ie pas de doublon).
— L'ensemble doit pouvoir contenir au minimum 1000 entiers.
— Il faut que cette implémentation soit relativement efficace. (On peut, par exemple, utiliser un tableau d'entiers triés)

1. Écrire la classe `Ensemble` qui contiendra les méthodes publiques
   — `Ensemble()`
   — `void ajoute(int a)`
   — `boolean retire(int a)`
   — `boolean est_dans(int a)`
   — `String toString()`
   La fonction `toString()` renvoie une chaîne de caractère qui affiche les éléments d'une instance de type `Ensemble`.
   Exemple d'utilisation :

```
Ensemble E = new Ensemble();
System.out.println(E.toString())
>>>>  {}
```

```
E.ajoute(4);
System.out.println(E.toString())
>>>> {4}
```

```
E.ajoute(3);
E.ajoute(20000);
System.out.println(E.toString())
>>>> {3,4,20000}

E.ajoute(3);
System.out.println(E.toString())
>>>> {3,4,20000}
```

```
System.out.println(E.est_dans(3))
>>>> true

System.out.println(E.retire(3))
>>>> true

System.out.println(E.est_dans(3))
>>>> false
```

2. Implémenter les méthodes
   — `void union(Ensemble A)`
   — `void inter(Ensemble A)`
   — `void diff(Ensemble A)`
   `union` ajoute tous les éléments de A à l'instance. `inter` retire de l'instance tous les éléments qui ne sont pas dans l'ensemble A. `diff` retire de l'instance tous les éléments qui sont dans l'ensemble A. (Un conseil : Réutilisez du code)

Pour expérimenter le fonctionnement de votre classe vous pouvez remplir vos ensembles avec des nombres aléatoires.

Exercice 2 Ensembles de rationnels

1. Faire ou finir l'exercice sur les rationnels de la semaine dernière.
2. Implémenter la classe EnsembleRat qui représente des ensembles de rationnels.
   La classe EnsembleRat utilise les mêmes méthodes que la classe Ensemble. (Avec un mécanisme d'héritage la création de cette classe aurait été trés rapide!)
   Si nécessaire, ajoutez les méthodes manquantes à la classe Rationnel de la semaine dernière.

Exercice 3 Piles

Une pile est un ensemble d'objets dont le seul élément qui peut en être extrait est le dernier élément ajouté. Une pile peut possèder plusieur fois le même élément. On veut écrire une classe `Pile` qui représentera une pile d'entiers.

— Définir la classe Pile. Le constructeur de cette classe construira une pile vide. Les éléments de la pile seront placés dans un tableau de dimension 1000. Un champ privé contiendra le nombre d'éléments de la pile.
— Définir une méthode estVide permettant de tester si la pile est vide.
— Définir la méthode push qui ajoute un élément à la pile (ajoute au sommet de la pile).
— Définir la méthode pop qui retire un élément de la pile et retourne sa valeur. (L'élément retiré celui qui se trouve au sommet de la pile)
— Définir la méthode sommet qui retourne la valeur du sommet de la pile.
— Définir la méthode toString qui renvoie une chaîne de caractère qui représente la pile.