SD
1ère Informatique IPL

Dico

Exercices à soumettre

A Comparable – Comparator

Vous trouverez sur moodle les classes *Etudiant* et *GestionEtudiants*.
Tout étudiant de la classe *Etudiant* possède un numéro de matricule (identifiant unique), un nom et un prénom.

La classe *GestionEtudiants* propose le menu suivant :

```
1 -> ajouter un etudiant
2 -> afficher tous les etudiants
```

La structure de données utilisée est un *HashSet*.
Lors de l'affichage des étudiants, l'ordre dans lequel ceux-ci apparaissent ne suit pas un ordre logique.
(L'itérateur parcourt une table de hashing. Les éléments y sont placés en fonction du code de hashing.)

Exemple :
On ajoute successivement :

```
Entrez le numero de matricule : 75
Entrez le nom : dupont
Entrez le prenom : annick

Entrez le numero de matricule : 22
Entrez le nom : lecharlier
Entrez le prenom : loic

Entrez le numero de matricule : 64
Entrez le nom : cambron
Entrez le prenom : isabelle

Entrez le numero de matricule : 63
Entrez le nom : lapiere
Entrez le prenom : baptiste

Entrez le numero de matricule : 74
Entrez le nom : dupont
Entrez le prenom : pierre
```

Et voici l'affichage obtenu :

```
64 cambron isabelle
22 lecharlier loic
74 dupont pierre
75 dupont annick
63 lapiere baptiste
```

L'affichage va dépendre du choix de la structure de données !
On vous demande d'écrire 3 nouvelles classes de gestion des étudiants.
«Pour chacune de celles-ci, à vous d'introduire la bonne structure de données **qui n'est pas nécessairement un treeSet.**

Vous devez apporter des modifications à la classe *Etudiant*. Mais, attention, on veut que les 4 classes de gestion des étudiants fonctionnent avec celle-ci.

Pour faciliter vos tests, des commandes ont été introduites dans le fichier *InputA.txt*.
Il suffit de placer ce fichier à la racine de votre projet et d'utiliser la classe *MonScanner* à la place de la classe *Scanner*.

```
private static MonScanner scanner = new MonScanner("InputA.txt");
```

Pour plus d'info sur cette façon de procéder, allez relire le document *CommandesAPartirDUnFichier* fourni en semaine 9.

A1 On voudrait que les étudiants soient affichés selon l'ordre croissant des numéros de matricule.

```
22 lecharlier loic
63 lapiere baptiste
64 cambron isabelle
74 dupont pierre
75 dupont annick
```

Appelez cette classe *GestionEtudiantsViaMatricule*.

A2 On voudrait que les étudiants soient affichés selon l'ordre alphabétique des noms et pour un même nom selon l'ordre alphabétique des prénoms.

```
64 cambron isabelle
75 dupont annick
74 dupont pierre
63 lapiere baptiste
22 lecharlier loic
```

Appelez cette classe *GestionEtudiantsViaNom*.

A3 On voudrait que les étudiants soient affichés selon l'ordre des ajouts

```
75 dupont annick
22 lecharlier loic
64 cambron isabelle
63 lapiere baptiste
74 dupont pierre
```

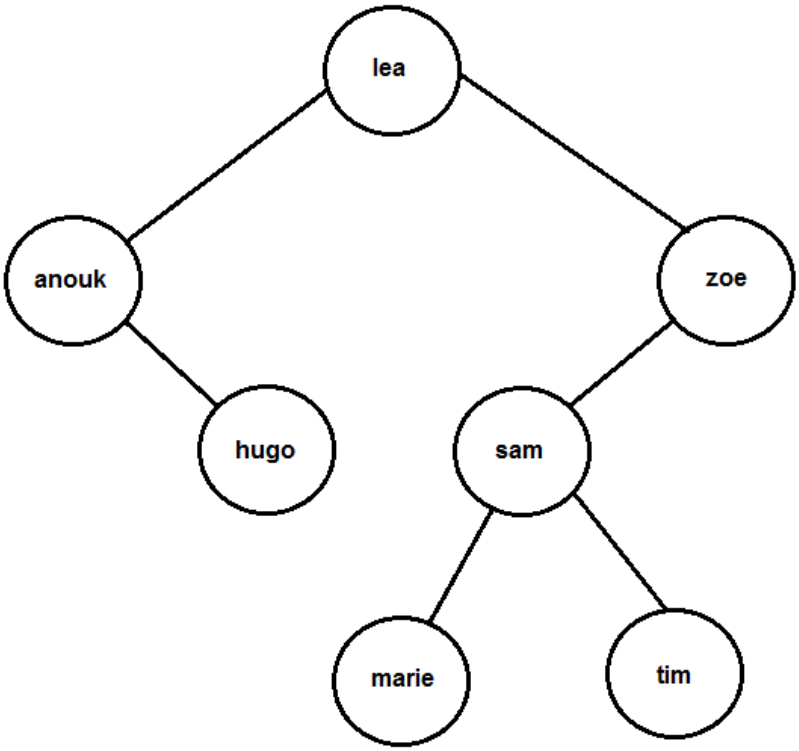
Appelez cette classe *GestionEtudiantsViaArrivee*.

B EnsembleTrie

Un ensemble trié possède les caractéristiques d'un ensemble mais les éléments y sont triés.
Les éléments qui s'y trouvent sont comparables.
Dans son interface, en plus de toutes les méthodes de l'ensemble, on retrouve des méthodes qui sont intéressantes avec des éléments comparables.
Nous vous avons fourni une interface *EnsembleTrie* réduite.

Nous vous demandons d'implémenter cette interface à l'aide d'un arbre binaire de recherche (ABR).
Cet ABR ne peut pas contenir de doublons.
On fait le choix que la descendance **gauche** d'un nœud ne contiendra que des éléments « **inférieurs »** à l'élément de ce nœud et la descendance **droite** d'un nœud ne contiendra que des éléments « **supérieurs »**.

Voici un exemple :
(Les chaînes de caractères sont triées selon l'ordre alphabétique.)



Vous allez compléter la classe *EnsembleTrieImpl* qui implémente l'interface *EnsembleTrie*.

Cette classe contient une classe interne *Nœud*.

Elle possède 2 attributs, le nœud `racine` et la `taille`.

Elle propose 2 constructeurs dont un va servir pour la classe de tests.
La méthode `toString()` vous est donnée. Pour l'arbre ci-dessus la méthode renvoie :

```
[  [  [ ] anouk [ hugo ]  ] lea [  [  [ marie ] sam [ tim ]  ] zoe [ ]  ]
]
```

Cette classe est générique. Elle ne doit pas tester si ses éléments appartiennent à une classe qui implémente l'interface *Comparable*.

Attention, pour pouvoir utiliser la méthode `compareTo()`, il faut « caster » l'élément en *Comparable*.
Par exemple, pour vérifier si l'élément1 est « inférieur » à l'élément2 :

```
if(((Comparable<E>)element1).compareTo(element2)<0) …
```

Les méthodes `taille()` et `estVide()` vous sont données.

B1 Dans la classe *EnsembleTrieImpl*, vous allez compléter les méthodes `min()`, `contient()`, `ajouter()` et `predecesseur()`.

Complétez ces méthodes en respectant bien la *JavaDoc* (reprise dans l'interface) et les choix d'implémentation imposés.
Vous pouvez ajouter des méthodes.

La classe *TestEnsembleTrieImpl* permet de tester vos méthodes avec l'arbre mis en exemple ci-dessus ainsi que l'arbre vide.

La classe *String* implémente *Comparable*.

Exercice défi

B2 Pour implémenter la méthode `predecesseur()`, on vous a suggéré de remplir une liste avec tous les éléments de l'arbre.
Il n'est pas nécessaire que cette liste soit complète. Dès que l'élément pour lequel on cherche son prédécesseur est rencontré, il devient inutile de compléter la liste.
Et puis ! Pourquoi remplir une liste ? Cela prend de la place inutilement. Seul le dernier élément mis dans la liste avant « rencontre » de l'élément recherché doit être retenu.
Revoyez votre méthode `predecesseur()`.

Exercice supplémentaire

A4 avec homonymes
Tester votre classe *GestionEtudiantsViaNom* en ajoutant 2 étudiants qui ont le même nom et le même prénom et qui ont reçu des numéros de matricule différents.
Ceux-ci ont-ils été acceptés ?
Si ce n'est pas le cas, faites les corrections nécessaires pour accepter les homonymes !