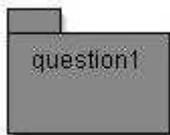


TP5

<p>Lectures préalables :</p> <ol style="list-style-type: none"> 1 Ces notes de cours <u>Collection<E></u> 1 Ce tutorial sur les <u>collections</u> et/ou <u>celui là</u> et/ou le <u>chapitre2</u> 1 la <u>JDC Tech Tips</u>, revoir <u>également les classes internes</u> 	<p>Thèmes du TP :</p> <p>classes abstraites, interface, héritage</p> <ol style="list-style-type: none"> 1 <code>package java.util</code> <ul style="list-style-type: none"> 1 <code>java.util.<u>AbstractCollection</u></code> 1 <code>java.util.<u>AbstractSet</u></code> 1 <code>java.util.<u>Set</u></code> 1 <code>java.util.<u>TreeSet</u></code> 1 <code>java.util.<u>Vector</u></code> 1 <code>java.util.<u>HashMap</u></code> 1 <code><u>Iterator</u> , <u>Comparator</u></code> 1 <code><u>classes internes</u></code>
---	---

(L'énoncé de la question 1 est inspiré du tutorial de Sun sur les collections.)

Rappel : Avec Konqueror sous Linux, les applettes ne seront visibles qu'après avoir autorisé le langage Java (Outils/Configuration_HTML/Java et rechargement de la page).



Une classe Ensemble

question1-1) Compléter la classe "Ensemble", nommée **Ensemble<T>**

- 1 *Seule une méthode est à développer ici :*
`public boolean add(T t).`
- 1 *L'implémentation préconisée utilise une instance de la classe `java.util.Vector<T>`.*
- 1 Pour vérifier le bon fonctionnement, comment voir le contenu d'un ensemble dans BlueJ ?

question1-2) Proposez une classe de tests unitaires de la classe "Ensemble<T>"
Utiliser la possibilité d'enregistrement des actions pour créer des méthodes de test.

question1-3) Enrichissez la classe **Ensemble<T>** avec ces **opérations** :

- 1 union
- 1 intersection
- 1 différence
- 1 différence symétrique ($(e \text{ union } e1) - (e \text{ inter } e1)$) : `diffSymetrique`

Pour ces 4 opérations, on veillera à n'utiliser ni boucle ni itérateur. Des méthodes utiles sont présentes dans `AbstractCollection`.

ensemble e1 :	<input type="text"/>
ensemble e2 :	<input type="text"/>
Opérations e1 Op e2 :	<input checked="" type="button" value="union"/> <input type="button" value="intersection"/> <input type="button" value="différence"/> <input type="button" value="différence symétrique"/>
Résultat	<input type="text"/>

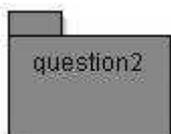
Applette de Test

Chaque opération retourne un nouvel ensemble, comme le suggère cette signature de la méthode "union"

```
public Ensemble<T> union( Ensemble<T>e1) ...
```

une utilisation possible :

```
Ensemble e = ...
System.out.println(" union de e et de e1 : " + e.union(e1));
```



les listes et dictionnaires

Le texte de la fenêtre de l'applette ci-dessous est une liste constituée de mots extraits du [chapitre 2 de CoreJava2](#) consacré au "**LinkedList**" (les mots sont rassemblés dans une constante de type "String", **CHAPITRE2** de la classe **Chapitre2CoreJava2**).

Complétez la classe **Chapitre2CoreJava2**
en développant ces deux méthodes de classe

Obtention d'une liste de mots à partir de la constante **CHAPITRE2**

```
public static List<String> listeDesMots()
```

Obtention d'une liste de couples <String,Integer>, chaque mot du **CHAPITRE2**, est associé son nombre d'occurrence

```
public static Map<String,Integer> OccurrencesDesMots()
```

Toutes les actions associées aux noms des boutons de cette IHM doivent être implémentées

rechercher : recherche du mot tapé dans la zone de saisie; le booléen, le résultat de la recherche est affiché. la touche Entrée du clavier a le même effet qu'une action effectuée sur ce bouton.

retirer : retrait de tous les mots commençant par le préfixe de la zone de saisie; le booléen, résultat du retrait est affiché.

croissant, décroissant : tri du texte selon cet ordre; utilisez **Collections.sort**, pour l'ordre décroissant des éléments, une solution est de proposer une classe implémentant l'interface Comparator.

occurrence : obtention du nombre d'occurrences du mot présent dans la zone de saisie

Complétez la classe **IHMListe**

une IHM possible :

java.util.LinkedList et java.util.HashMap

rechercher retirer tri du texte : croissant décroissant occurrence

[Core, Java, 2, Volume, II, by, Cay, S, Horstmann, and, Gary, Cornell, Chapter, 2, Collection interfaces, we, thought, it, would, be, helpful, to, first, discuss, the, concrete, data, structure what, classes, you, will, want, to, use, we, will, return, to, abstract, considerations, and, set arrays, and, their, dynamic, cousin, the, Vector, class, for, many, examples, in, Volume, 1, I from, the, middle, of, an, array, is, very, expensive, since, all, array, elements, beyond, the, The, same, is, true, for, inserting, elements, in, the, middle, Figure, 2-4, Removing, an, ele problem, Whereas, an, array, stores, object, references, in, consecutive, memory, location reference, to, the, next, link, in, the, sequence, In, the, Java, programming, language, all, list its, predecessor, see, Figure, 2-5, Figure, 2-5, A, doubly, linked, list, Removing, an, eleme

N'oubliez pas de faire un submit avant de quitter la salle !