TP5

Lectures préalables :	Thèmes du TP :
1. Ces notes de cours Collection<E> 1. Ce tutorial sur les **collections** et/ou **celui_là** et/ou le **chapitre2** 1. la JDC Tech Tips, revoir **également les classes internes**	classes abstraites, interface, héritage 1. **package java.util** ¡ ***java.util.AbstractCollection*** ¡ ***java.util.AbstractSet*** ¡ ***java.util.Set*** ¡ ***java.util.TreeSet*** ¡ ***java.util.Vector*** ¡ ***java.util.HashMap*** 1. ***Iterator , Comparator*** 1. ***classes internes***

(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>`**

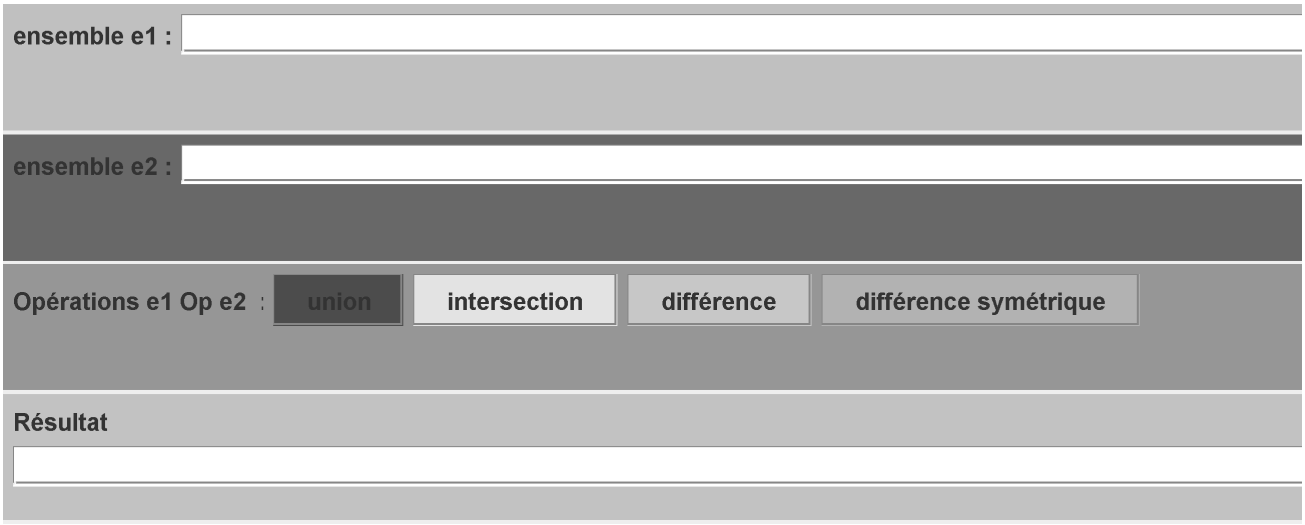
1. *Seule **une méthode est à développer ici** :*
   `public boolean add(T t)`.
1. *L'implémentation préconisée utilise une instance de la classe*
   `java.util.Vector<T>`.
1. Pour vérifier le bon fonctionnement, comment voir le contenu d'un ensemble dans BlueJ ?

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 union e1) - (e inter e1)*) : diffSymetrique

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



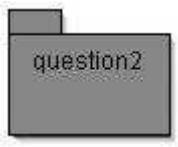
Applette de Test

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

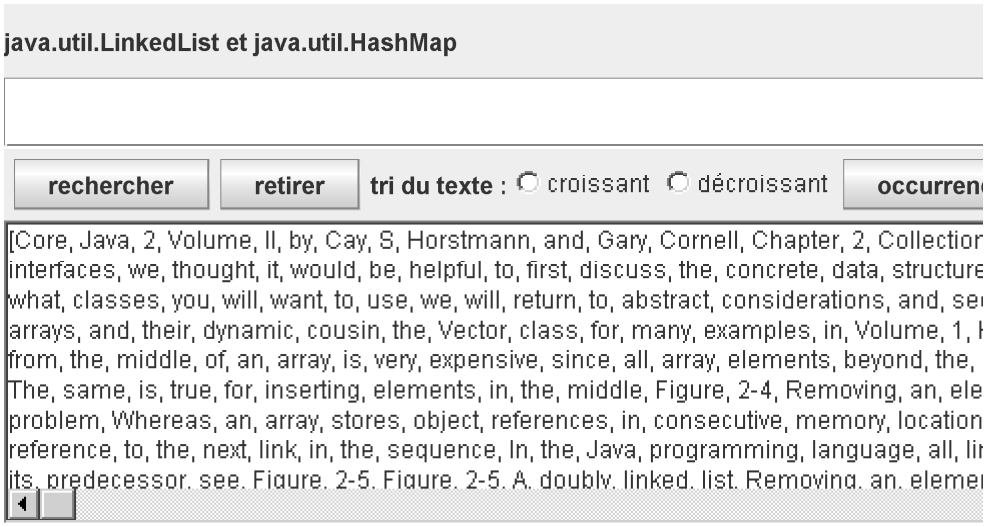
```
public static Map<String,Integer> OccurrencesDesMots()
```

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 :



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