# Conservatoire national des arts et métiers

# NFP121 : Programmation avancée
# TP n° 5 - Classes abstraites / interface / héritage

Thèmes du TP :

Classes abstraites, interface, héritage

- Le package java.util
  - java.util.AbstractCollection
  - java.util.AbstractSet
  - java.util.Set
  - java.util.TreeSet
  - java.util.Vector
  - java.util.HashMap
- Les interfaces
  - Iterator
  - Comparator

Lecture préalable :

- Les notes de cours
- Ce tutorial sur les collections
- Le chapitre 2 de ce livre
- Revoir également les classes internes (*Nested Classes*)

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

---


## Une classe Ensemble

Complétez la classe "Ensemble", nommée **Ensemble<T>**

- L'implémentation préconisée utilise par délégation une instance de la classe **java.util.Vector<T>**.
- Seule **une méthode est à développer ici** : `public boolean add(T t)`
- Cette méthode doit garantir la sémantique de l'ajout d'un élément dans un ensemble (au sens mathématique).
- Questions à se poser : Que se passe-t-il si on utilise this.add(...) dans cette méthode ? Et this.contains() ? Que déclenche la méthode addAll de la super-classe ?

---

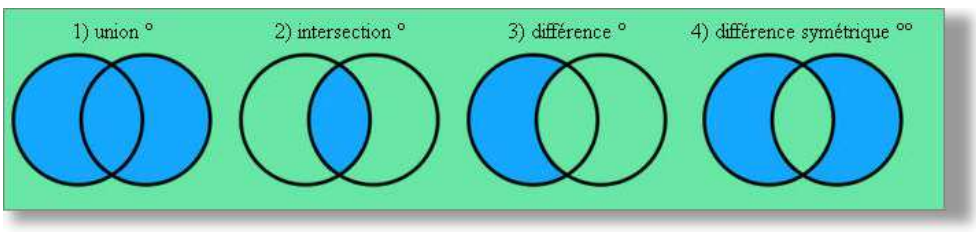

## Les tests unitaires

Proposez une classe de tests unitaires de la classe "Ensemble<T>" (pour la méthode add())

---


## La classe Ensemble<T> (suite)

Enrichissez la classe Ensemble<T> avec ces 4 opérations :



**Aide :**

° **voir les méthodes xxxAll** spécifiées dans l'interface java.util.Collection (**aucune boucle n'est nécessaire ! pour la réalisation de ces 4 méthodes**). Dans la javadoc, "... adds each object ..." signifie "... calls add() for each object ..."
°° ((e union e1) - (e inter e1))



**Applette de test**

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

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

Une utilisation possible :

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

---


## Les tests unitaires (suite)

Enrichissez la classe de tests unitaires demandée en 1.2 (chaque méthode doit avoir été testée au moins une fois).

---


## L'applette AppletteTestEnsemble

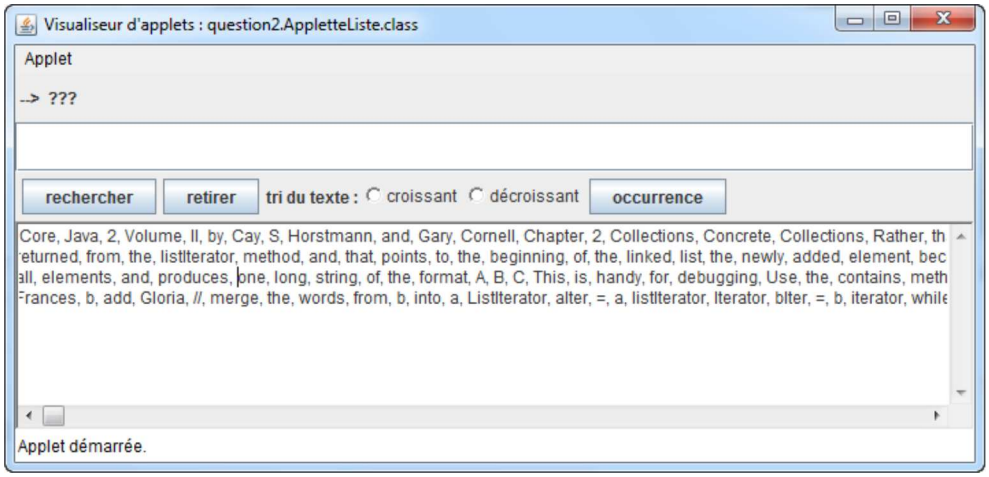
Complétez l'applette nommée AppletteTestEnsemble avec une classe anonyme pour chaque "JButton" de cette interface.

---


## 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", **nommée CHAPITRE2** de la classe **Chapitre2CoreJava2**).

**L'objectif est de pouvoir faire différents traitements sur cette liste de mots.**



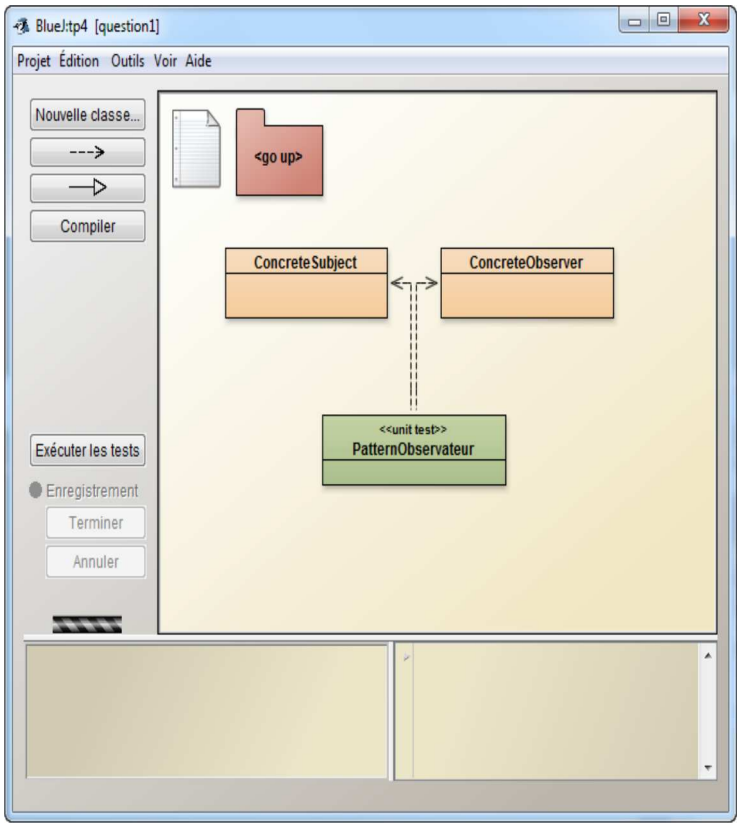

Complétez la classe **Chapitre2CoreJava2** en développant ces deux méthodes de classe

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

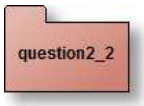
```
public static List<String> listeDesMots()(utilisez une LinkedList)
```

- Obtention d'une liste de couples <String,Integer>, à partir la liste des mots ci-dessus.

```
public static Map<String,Integer> occurrencesDesMots(List<String> listeDesMots)(utilisez une HashMap)
```




---



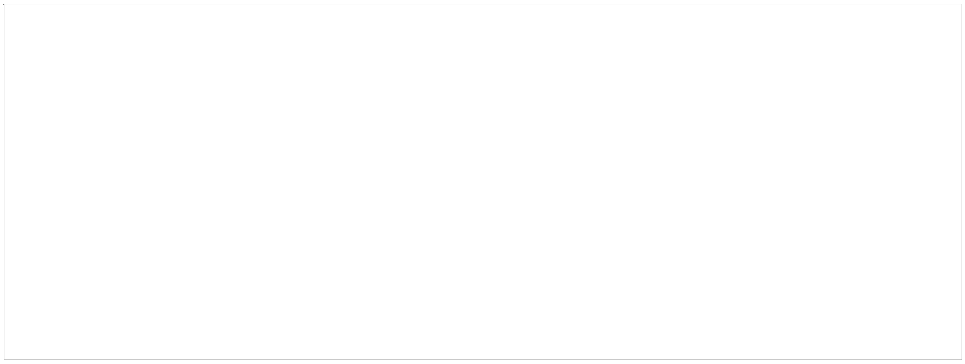

## la classe IHMListe

Complétez la classe **IHMListe** afin d'implanter, toutes les actions associées aux noms des boutons

- **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** : tri du texte selon cet ordre, utilisez Collections.sort.
- **décroissant** : proposer une classe interne implémentant l'interface Comparator.
- **occurrence** : obtention du nombre d'occurrences du mot présent dans la zone de saisie

une IHM au comportement attendu :



**ATTENTION** le retrait d'un élément qui a déjà été présent dans la table retourne un nombre d'occurrences égal à **zéro**. Demander le nombre d'occurrences d'un élément qui n'a jamais été présent dans la liste initiale doit retourner **???**

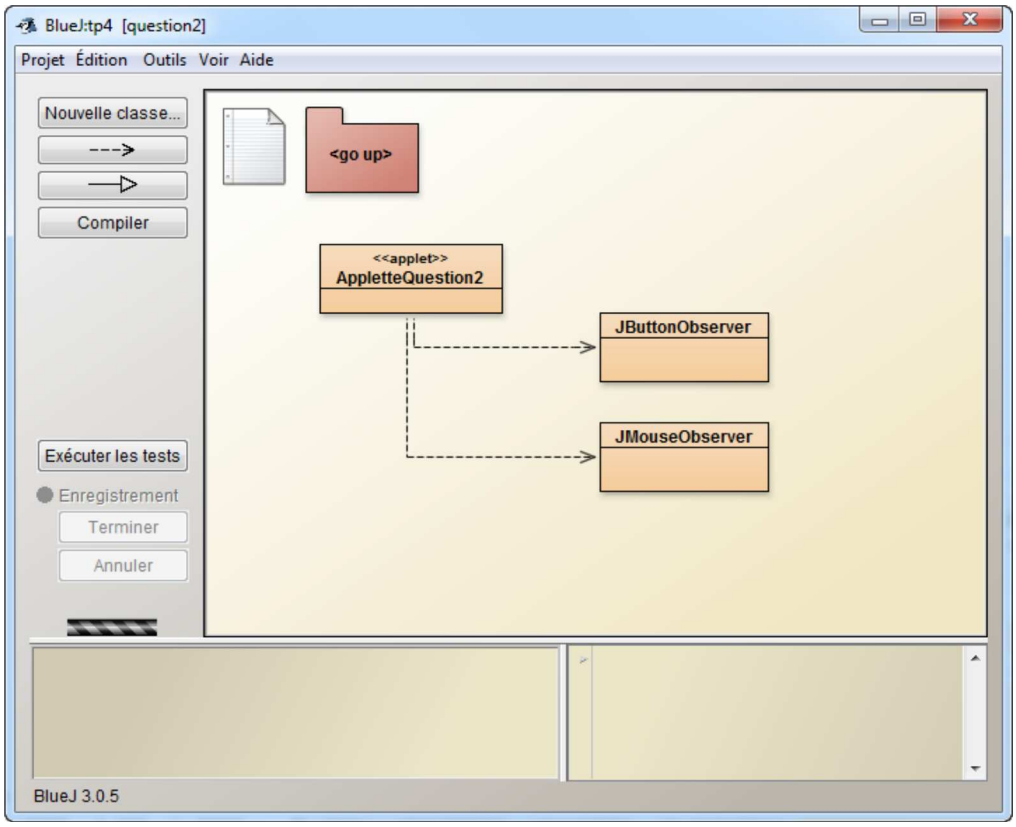
---


## la classe IHMListe2

Complétez maintenant, la classe IHMListe2 afin d'implanter la possibilité d'annuler les actions de modification du texte comme le retrait ou le tri sur la liste.

L'idée est de stocker l'état de la 'liste de String' à chaque action ('retirer' , 'croissant' , 'decroisssant') dans une Pile (java.util.Stack). Le dernier état de la 'liste de String' empilé est restitué à chaque action 'annuler'. Quand la pile est vide le bouton 'annuler' est sans effet.

**ATTENTION de mettre à jour la table des occurrences**. Pour la fonctionnalité annuler, Il est fortement encouragé d'utiliser le patron Memento, cette vidéo peut vous être utile.




Comportement attendu :

**Soumettez en fin de question votre réponse**

---


## Le pattern fabrique

Selon la bibliographie habituelle (GoF95), l'objectif du Pattern Factory est de définir une interface pour la création d'un objet, en laissant aux classes implémentant cette interface le choix de la classe à instancier pour cet objet.

Interface *Factory<T>*, l'implémentation de la méthode create est laissée aux "clients"

```
package question3;

   public interface Factory<T> {
      public T create();
   }
```

Exemple : TextFactory

```
public class TextFactory1 implements Factory<TextComponent> {
  public TextComponent create() {
    return new TextArea(100,50);
  }
}
public class TextFactory2 implements Factory<TextComponent> {
  public TextComponent create(){
    return new TextField(40);
  }
}
```

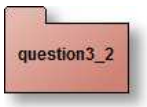
Un usage :

```
  public void utilisation(Factory fabrique){
        TextComponent tc = fabrique.create();
        tc.setText( "essai" );
  }

  utilisation(new TextFactory1());
  utilisation(new TextFactory2());
```

Proposez les fabriques d'ensembles **HashSetFactory**, (en utilisant la classe concrète java.util.HashSet) et **TreeSetFactory**, (en utilisant la classe concrète java.util.TreeSet, dans ce cas les éléments doivent être comparables...).

---




## Les tests unitaires

Complétez les classes et la classe de Tests unitaires.