

Prérequis

- Chaque exercice se présente sous la forme de classes Java à compléter dont vous pourrez trouver un squelette d'implémentation dans /pub/ILO/TP2.zip.
 - Copiez cette archive chez vous et dézippez la dans un sous répertoire ILO par exemple
 - Le sujet se trouve dans le fichier Sujet.pdf que vous pouvez ouvrir avec « evince ».
 - Lancez votre IDE préféré et créez un nouveau projet Java ayant pour emplacement le répertoire où vous avez dézippé l'archive. N'oubliez pas d'ajouter la librairie Junit4 à votre projet sans quoi vous ne pourrez pas lancer les classes de test.
- Documentation Java :
 - <http://docs.oracle.com/javase/8/docs/api/>

1. Ensembles

On souhaite réaliser plusieurs implémentations des ensembles. Un ensemble est défini comme une collection non ordonnée d'éléments sans doublons. Les différentes opérations sur un ensemble sont les suivantes :

- Ajout d'un élément : $\{ a \ b \ c \} + b = \{ a \ b \ c \}$
 $\{ a \ b \ d \} + c = \{ a \ b \ d \ c \}$
- Retrait d'un élément : $\{ a \ b \ c \} - c = \{ a \ b \}$
- Ensemble vide : $\{ a \ b \ c \} = \emptyset = \text{faux}$
- Appartenance d'un élément à un ensemble : $a \in \{ a \ b \ c \} = \text{vrai}$
- Inclusion d'un ensemble dans un autre : $A \subset B = \{ a \ b \} \subset \{ a \ b \ c \} = \text{vrai}$
 $\emptyset \subset B = \emptyset \subset \{ a \ b \ c \} = \text{vrai}$, et $B \subset \emptyset = \text{faux}$
- Cardinal : $\text{card}(\{ a \ b \ c \}) = 3$
- Complément : $A - B = \{ a \ b \ c \} - \{ c \ d \ e \} = \{ a \ b \}$ et $A - B \neq B - A$
 $B - A = \{ c \ d \ e \} - \{ a \ b \ c \} = \{ d \ e \}$
- Union : $A \cup B = \{ a \ b \ c \} \cup \{ c \ d \ e \} = \{ a \ b \ c \ d \ e \}$ et $A \cup B = B \cup A$
- Intersection : $A \cap B = \{ a \ b \ c \} \cap \{ c \ d \ e \} = \{ c \}$ et $A \cap B = B \cap A$
- Différence symétrique : $A \Delta B = \{ a \ b \ c \} \Delta \{ c \ d \ e \} = \{ a \ b \ d \ e \}$
 $A \Delta B = B \Delta A = (A - B) \cup (B - A) = (A \cup B) - (B \cap A)$
- Egalité de deux ensembles: $A == B = (A \subset B) \& (B \subset A)$

Ces ensembles peuvent être implémentés de différentes manières, avec des « Vector » (vu au TD n°1), ou bien avec des Tableau<E> (voir Figure 1).

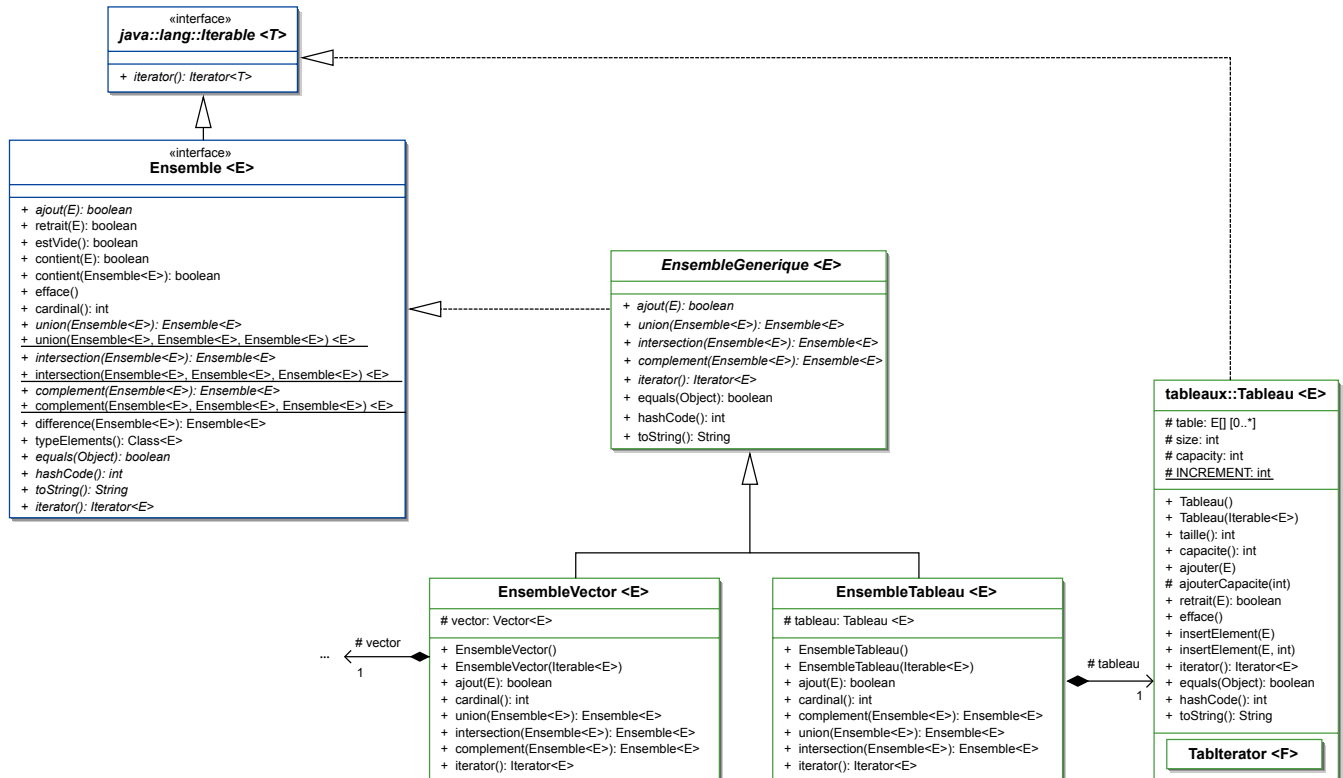


Figure 1: Implémentation des ensembles (en Java8¹)

L'interface « Ensemble<E> » définit les opérations à réaliser sur tous les ensembles

On veut réaliser deux implémentations des ensembles, une première utilisant comme conteneur sous-jacent un Tableau<E> et une seconde utilisant un Vector<E>. Le Tableau comme le Vector étant toutes les deux des classes Itérables (implémentant l'interface Iterable<E>) une grande partie des opérations sur les ensembles peuvent être réalisées grâce à un iterator, on peut donc factoriser une grande partie de ces opérations dans l'interface « Ensemble<E> » (en Java 8) en utilisant les opérations fournies par l'iterator.

1. Etudiez l'interface Ensemble<E> pour vous familiariser avec ses opérations.
2. Complétez les implémentations partielles en utilisant les indications contenues dans les commentaires de documentation de :
 - a. L'interface Ensemble<E> (en utilisant des méthodes par défaut et des méthodes de classes).
 - b. La classe abstraite EnsembleGenerique<E> implémentant (partiellement) l'interface Ensemble<E>.
3. Etudiez la classe Tableau<E> pour vous familiariser avec ses opérations.
4. Complétez la première implémentation concrète des ensembles (EnsembleTableau<E>) héritant de la classe abstraite EnsembleGenerique<E> et utilisant comme conteneur sous-jacent pour stocker les éléments de l'ensemble une instance de la classe Tableau. Vous remarquerez que comme la plupart des opérations ont déjà été implémentées dans les classes/interfaces mères, il ne reste plus à implémenter que les méthodes qui allouent ou qui ajoutent des éléments.

5. Testez vos ensembles avec la classe de test AllEnsembleTest.java du package tests : Menu contextuel → Run As → JUnit Test.

La classe AllEnsembleTest teste les fonctionnalités des Ensemble<E> :

Pour faciliter la progression des tests, vous pourrez commenter / décommenter les lignes suivantes de la classe AllEnsembleTest.java (ligne 73) au fur et à mesure de votre progression dans l'implémentation des classes, le dernier élément non commenté ne doit pas contenir de virgule :

```
/**
 * Les différentes natures d'ensembles à tester
 */
@SuppressWarnings("unchecked")
private static final Class<? extends Ensemble<String>>[] typesEnsemble =
(Class<? extends Ensemble<String>>[]) new Class<?>[]
{
    EnsembleTableau.class,
    EnsembleVector.class // à commenter / décommenter
    ...
};
```

6. Complétez de la même manière la seconde implémentation concrète des ensembles (EnsembleVector<E>) utilisant comme conteneur sous-jacent une instance de la classe Vector.

2. Ensembles triés.

On veut maintenant réaliser de la même manière que précédemment deux implémentations des ensembles triés dans lesquels les éléments doivent être maintenus dans un ordre croissant au sens de l'interface Comparable<E> (voir la doc). On veut pour ce faire réutiliser autant que possible ce que l'on a déjà réalisé sur les ensembles.

Néanmoins, certaines méthodes devront être spécialisées pour respecter les spécificités des ensembles triés :

- L'opération d'ajout devra insérer un nouvel élément de manière triée dans l'ensemble $\{ a \ b \ d \} + c = \{ a \ b \ c \ d \}$. On rajoutera donc une opération de calcul de rang d'un élément dans l'interface « EnsembleTri<E> » définissant les opérations spécifiques aux ensembles triés.
- Les méthodes equals(Object o) et hashCode() devront être réimplémentées pour les ensembles triés :
 - Le test d'égalité (equals(Object o)) devra cette fois prendre en compte l'ordre des éléments ce qui correspond au equals d'une Collection<E> tel que vu en cours.
 - Le calcul du hashCode devra lui aussi prendre en compte l'ordre des éléments (on pourra donc s'inspirer du calcul du hashCode de la classe Tableau).

Il existe pour implémenter les ensembles triés deux méthodes (voir Figure 2, page 4) :

- La première méthode consiste à faire directement des classes filles à EnsembleTableau<E> et EnsembleVector<E> et implémentant l'interface EnsembleTri<E>.
- La seconde méthode consiste à introduire une classe abstraite EnsembleTriGenerique<E> implémentant partiellement l'interface EnsembleTri<E> et possédant une instance d'Ensemble<E> (qu'on appellera ensemble interne) décorant un Ensemble ordinaire avec les fonctionnalités propres au tri. Un tel ensemble trié contient donc une instance d'ensemble ordinaire et l'on s'appuie en grande partie sur ses méthodes pour implémenter les opérations spécifiques aux ensembles triés.
 - Les classes filles de EnsembleTriGenerique<E> fourniront les implémentations concrètes des ensembles triés en implémentant les méthodes qui sont restées abstraites dans l'EnsembleTriGenerique<E>.

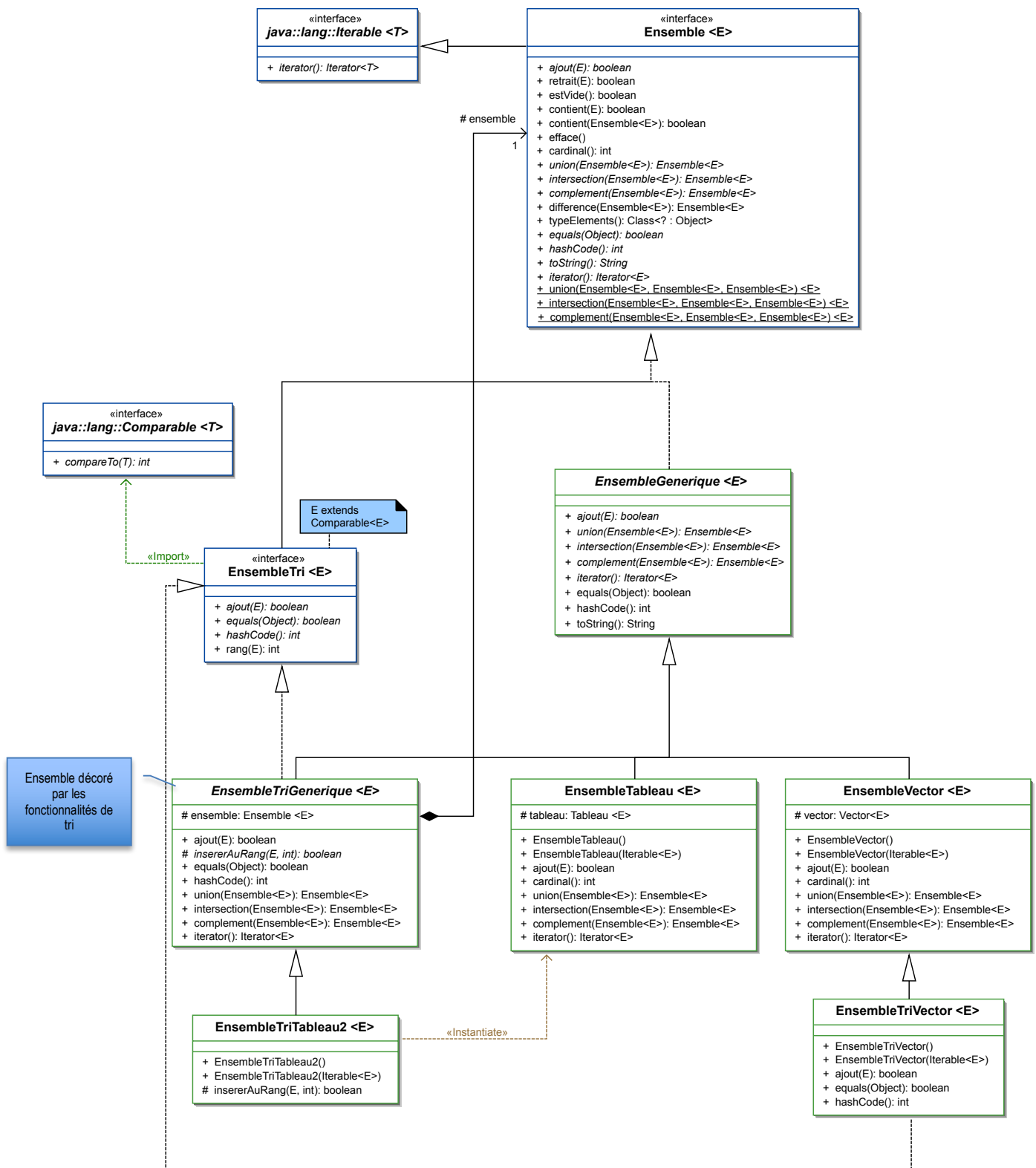


Figure 2 : Implémentation des ensembles triés

1. Créez la classe `EnsembleTriVector<E>` ou la classe `EnsembleTriTableau<E>` avec la première méthode implémentant l'interface `EnsembleTri<E>` et héritant (respectivement) de la classe `EnsembleVector<E>` ou `EnsembleTableau<E>`.
2. Complétez la classe abstraite `EnsembleTriGenerique<E>` implémentant l'interface `EnsembleTri<E>` et héritant de la classe `EnsembleGenerique<E>`. Celle ci servira de base aux classes `EnsembleTriTableau2<E>` et `EnsembleTriVector2<E>`. **Vous pourrez remarquer que**

l'implémentation de ces classes est très similaire à celles de EnsembleTriVector<E> ou EnsembleTriTableau<E>.

3. Créez la classe concrète EnsembleTriTableau2<E> implémentant la seconde méthode en héritant de la classe EnsembleTriGenerique<E>. Cette classeinstanciera comme ensemble décoré un EnsembleTableau<E>.
4. Créez la classe EnsembleTriVector2<E> implémentant la seconde méthode en héritant de la classe EnsembleTriGenerique<E>. Cette classeinstanciera comme ensemble décoré un EnsembleVector<E>.
 - a. Si l'on avait choisi qu'une seule méthode sur les deux, laquelle serait la plus avantageuse lorsque l'on veut deux implémentations (avec Tableau et avec Vector) des ensembles triés ?

7. Testez vos ensembles triés avec la classe de test AllTest.java qui teste à la fois les ensembles (tous les ensembles triés ou pas : AllEnsembleTest.java) et les ensembles triés (EnsembleTriTest.java) : Menu contextuel → Run As → JUnit Test.

Pour faciliter la progression des tests, vous pourrez commenter / décommenter les lignes suivantes de la classe AllEnsembleTest.java (ligne 73) au fur et à mesure de votre progression dans l'implémentation des classes, le dernier élément non commenté ne doit pas contenir de virgule :

```
/**
 * Les différentes natures d'ensembles à tester
 */
@SuppressWarnings("unchecked")
private static final Class<? extends Ensemble<String>>[] typesEnsemble =
(Class<? extends Ensemble<String>>[]) new Class<?>[]
{
    EnsembleVector.class,
    EnsembleTableau.class,
    EnsembleTriVector.class,           // à commenter / décommenter
    EnsembleTriTableau2.class,       // à commenter / décommenter
    EnsembleTriVector2.class         // à commenter / décommenter
};
```

La classe EnsembleTriTest teste les fonctionnalités spécifiques aux ensembles.

Pour faciliter la progression des tests, vous pourrez commenter / décommenter les lignes suivantes de la classe EnsembleTriTest.java (ligne 67) au fur et à mesure de votre progression dans l'implémentation des classes, le dernier élément non commenté ne doit pas contenir de virgule :

```
/**
 * Les différentes natures d'ensembles à tester
 */
@SuppressWarnings("unchecked")
private static final Class<? extends EnsembleTri<String>>[] typesEnsemble =
(Class<? extends EnsembleTri<String>>[]) new Class<?>[]
{
    EnsembleTriVector.class,
    EnsembleTriTableau2.class,       // à commenter / décommenter
    EnsembleTriVector2.class         // à commenter / décommenter
};
```

3. Ensembles utilisant des Listes (à rendre sur le serveur de dépôt avant minuit, le jour du TP).

Nous souhaitons maintenant créer une troisième implémentation des ensembles en utilisant cette fois une instance de la classe Liste<E> que vous avez créée lors du dernier TD.

1. Adaptez votre classe Liste<E> à l'interface IListe<E> du package « listes » (en Java 8 toutes les méthodes utilisant uniquement l'itérateur peuvent être implémentées en tant que méthodes par défaut directement dans l'interface), ou plus simplement copiez vos fichiers IListe.java et Liste.java dans le package « listes ».
 2. Testez votre classe Liste<E> en utilisant la classe de test ListeTest.
 3. Créez une classe EnsembleListe<E> héritant de la classe abstraite EnsembleGenerique<E> et utilisant comme conteneur sous jacent une instance de la classe Liste<E>.
 4. Testez la classe EnsembleListe<E> avec la classe de test AllEnsembleListeTest.
 5. Créez une classe EnsembleTriListe<E> héritant de la classe EnsembleListe<E> et implémentant l'interface EnsembleTri<E>.
 6. Testez la classe EnsembleTriListe<E>
 7. Créez une classe EnsembleTriListe2<E> héritant de classe abstraite EnsembleTriGenerique<E>
- Testez la classe EnsembleTriListe2CréezVous aurez jusqu'au soir du TP 23h55 pour remettre votre travail sur le serveur de dépôt, en utilisant le dépôt nommé « ilo_tp2 ».

Votre rendu devra contenir au moins les classes et interfaces suivantes (uniquement les fichiers java) :

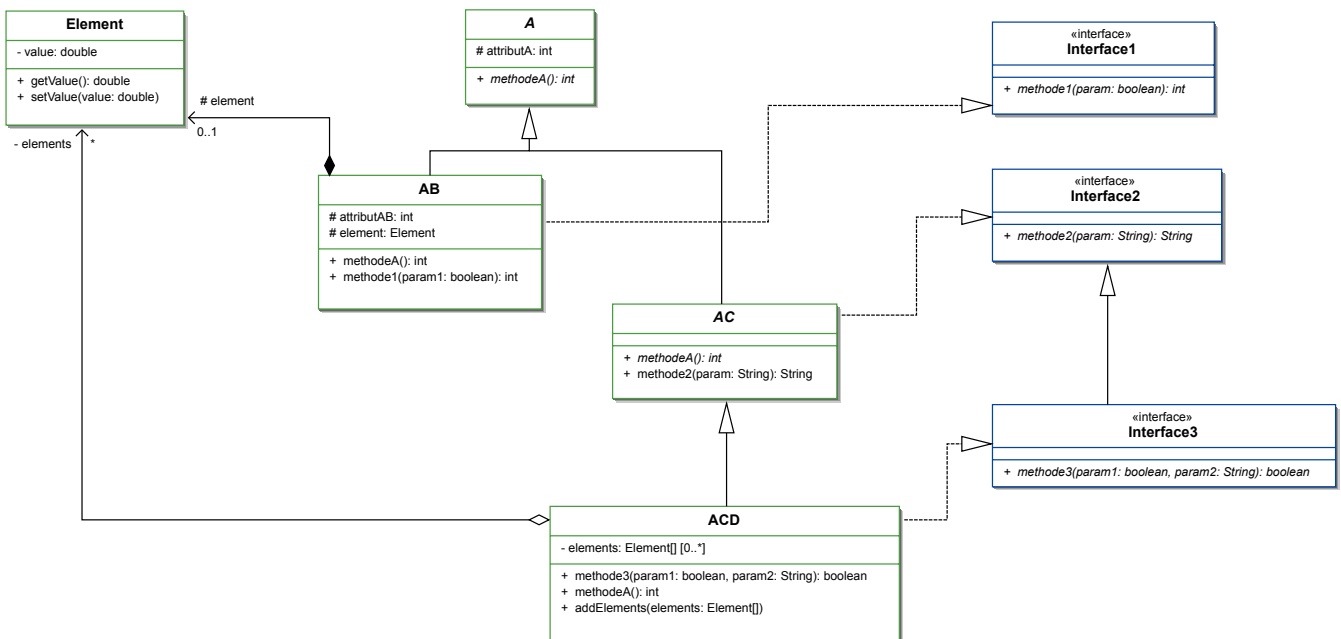
- IListe<E>
- Liste<E>
- Ensemble<E>
- EnsembleGenerique<E>
- EnsembleListe<E>
- EnsembleTri<E>
- EnsembleTriListe<E>
- EnsembleTriGenerique<E>
- EnsembleTriListe2<E>

Vous pourrez avantageusement utiliser la commande « make clean archive » à la racine de votre projet pour générer une archive de votre projet dans un sous-répertoire « archives ».

Rappels UML :

- L'héritage multiple n'existe pas en Java, néanmoins :
 - o Une classe peut implémenter plusieurs interfaces; on parle alors d'héritage multiple de comportements.
 - o Une interface peut hériter d'une autre interface
- Les membres publics sont précédés de : +
- Les membres protégés sont précédés de : #
- Les membres privés sont précédés de : -
- Les relations d'héritage sont indiquées par les flèches en traits pleins.
- Les relations d'implémentation sont indiquées par les flèches en traits pointillés.
- Les membres ou les classes concrète(s) sont indiqués en texte droit.
- Les membres ou les classes abstraite(s) sont indiqués en texte italique.
- On précisera pour les classes abstraites s'il s'agit d'interfaces ou de classes abstraites.
- Les compositions sont indiquées par un losange plein : par exemple la classe AB possède une instance de la classe « Element » et elle est responsable de la création et de la destruction de cette instance.
- Les agrégations sont indiquées par un losange vide : par exemple la classe ACD possède une collection d'instances de la classe « Element », mais elle n'est pas responsable de l'instanciation de cette collection (on supposera qu'elle lui est donnée par la méthode addElements).

Exemple :



Questions (à rendre à la fin de la séance à votre chargé de TD)

Nom :	Prénom :	Groupe :
-------	----------	----------

- 1) Quel était l'intérêt d'implémenter l'algorithmique des opérations ensemblistes (union, intersection, complément) en tant que méthodes de classes dans l'interface Ensemble<E> ou dans la classe abstraite EnsembleGenerique<E> ?

- 2) Et pourquoi n'a t'on pas fait de même avec l'opération de différence symétrique ?

- 3) Pourquoi les méthodes hashCode des ensembles triés doivent elles être différentes des méthodes hashCode des ensembles non triés ?

- 4) Parmi les deux manières d'implémenter des ensembles triés, lorsque l'on souhaite avoir au moins deux implémentations différentes, laquelle est la plus économique en termes de développements à réaliser ?

- a) Des classes EnsembleTriXXX<E> implémentant l'interface EnsembleTri<E> héritant de EnsembleXXX<E>
- b) Des classes EnsembleTriXXX<E> héritant d'une classe abstraite EnsembleTriGenerique<E> implémentant l'interface EnsembleTri<E>.

La méthode ... car ...