

### Introduction

Le but de ce TP est de vous faire implémenter une collection. Pour ce faire nous allons spécifier puis implémenter (de deux manières différentes) un « ensemble » d'éléments génériques : Un « ensemble » est une collection d'éléments de mêmes types, sans doublons, sans éléments null, dans laquelle l'ordre des éléments n'est pas pris en compte (pour la comparaison entre deux ensembles par exemple).

Vous pourrez trouver une ébauche du code à réaliser dans l'archive : /pub/ILO/TPSets/TP-Sets.zip.

Copiez cette archive sur votre compte et dézippez la dans un sous-répertoire (ILO/Sets par exemple), puis après avoir lancé « eclipse » importez le projet : Import ... → Existing Projects into Workspace → Select root directory → Browse et allez chercher le répertoire dans lequel vous avez dézippé le projet (« TP Sets »). Vous verrez alors apparaître le projet « TP Sets » que vous pourrez alors importer dans Eclipse.

### Spécification de l'interface « ensemble » : Set

L'interface Set<E> définit les comportements attendus d'un ensemble générique. Cette interface hérite de l'interface Collection<E> qui elle-même hérite de l'interface Iterable<E> (Voir Figure 1 ci-dessous).

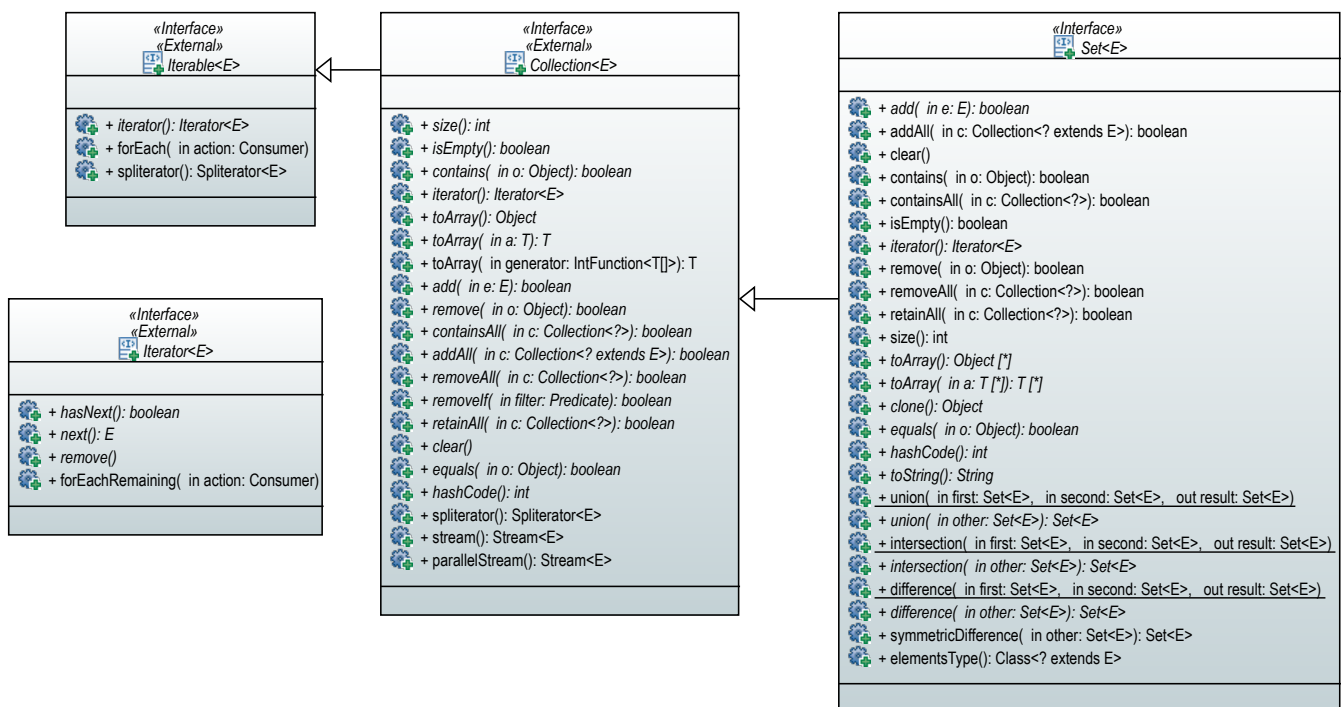


Figure 1 : Interface Set<E>

L'interface Iterable<E> définit une seule méthode abstraite (plus deux méthode concrètes que nous n'utiliserons pas) : iterator() : Iterator<E> qui renvoie un nouvel itérateur qui nous permettra d'itérer sur les éléments contenus dans l'Iterable.

L'interface Collection<E> définit les méthodes nécessaires pour :

- Ajouter des éléments, ou une collection d'éléments à la collection, ce qui se traduit dans le :
  - **boolean** add(E e)

$$\{a \quad b \quad c\} + d = \{a \quad b \quad c \quad d\}$$

- **boolean** `addAll(Collection<? extends E> c)`  
 $\{a \ b \ c\} + \{c \ d \ e\} = \{a \ b \ c \ d \ e\}$
- Retirer des éléments, ou une collection d'éléments de la collection :
  - **boolean** `remove(Object o)`  
 $\{a \ b \ c\} - b = \{a \ c\}$
  - **boolean** `removeAll(Collection<?> c)`  
 $\{a \ b \ c\} - \{c \ d \ e\} = \{a \ b\}$
- Effacer le contenu de la collection :
  - **void** `clear()`
- Tester si un élément ou une collection d'éléments est contenu dans la collection :
  - **boolean** `contains(Object o)`  
 $a \in \{a \ b \ c\} = true$
  - **boolean** `containsAll(Collection<?> c)`  
 $\{a \ b\} \subset \{a \ b \ c\} = true$
- Tester si la collection est vide :
  - **boolean** `isEmpty()`  
 $isEmpty(\{a \ b \ c\}) = false$
- Obtenir le nombre d'éléments dans la collection :
  - **int** `size()`  
 $size(\{a \ b \ c\}) = 3$
- Convertir le contenu de la collection en tableau :
  - `Object[] toArray()`
  - `<T> T[] toArray(T[] a)`
- Comparer la collection avec un autre objet.
  - **boolean** `equals(Object o)`  
 $\{b \ a \ c\} == \{a \ b \ c\} = true$
- Calculer le hashcode de la collection (dans le cas des ensembles pour lesquels l'ordre des éléments ne compte pas, il s'agira simplement de la somme des hashCodes des éléments) :
  - **int** `hashCode()`
- Obtention d'un itérateur sur les éléments contenus dans la collection :
  - `Iterator<E> iterator()`

L'interface `Set<E>` reprend l'interface `Collection<E>` et lui ajoute les opérations suivantes :

- Création d'une copie (distincte) de l'ensemble (avec le même contenu) :
  - `Object clone()`
- Opérations ensemblistes : union, intersection, différence et différence symétrique :
  - **static** `<E> void union(Set<E> first, Set<E> second, Set<E> result)`
  - `Set<E> union(Set<E> other)`
  - **static** `<E> void intersection(Set<E> first, Set<E> second, Set<E> result)`
  - `Set<E> intersection(Set<E> other)`
  - **static** `<E> void difference(Set<E> first, Set<E> second, Set<E> result)`
  - `Set<E> difference(Set<E> other)`
  - `Set<E> symmetricDifference(Set<E> other)`
- Détermination du type des éléments (ssi l'ensemble est non-vide)
  - `Class<? extends E> elementType()`

Du point de vue des ensembles les différentes opérations doivent être implémentées en suivant les descriptions suivantes : Soit deux ensembles  $A = \{a \ b \ c\}$  et  $B = \{c \ d \ e\}$

- Test d'égalité :  $A == B = (A \subset B) \ \& \ (B \subset A)$ .
- Union :  $A \cup B = \{a \ b \ c \ d \ e\} = B \cup A$ .
- Intersection :  $A \cap B = \{c\} = B \cap A$ .
- Différence :  $A - B = \{a \ b\} \neq B - A = \{d \ e\}$ .
- Différence symétrique :  $A \Delta B = \{a \ b \ d \ e\} = B \Delta A = (A - B) \cup (B - A) = (A \cup B) - (A \cap B)$ .

## Travail à réaliser

La Figure 2 ci-dessous présente les différentes classes à compléter : Nous allons réaliser deux implémentations concrètes de l'interface `Set<E>` :

- `ArrayListSet<E>` est une implémentation concrète de l'interface `Set<E>` utilisant comme conteneur sous-jacent une `ArrayList<E>` : `list`. Il s'agira donc pour implémenter la plupart des opérations de cette classe de déléguer les traitements à l'attribut `ArrayList<E>` : `list` qui implémente déjà toutes les méthodes d'une `Collection<E>`.
- `ArraySet<E>` est une deuxième implémentation concrète de l'interface `Set<E>` utilisant comme conteneur sous-jacent un simple tableau d'éléments `E[]` `elementData`; associé à un index `int` `elementCount`; indiquant le nombre d'éléments valides actuellement stockés dans le tableau.
  - Il faudra compléter la classe interne `ArraySetIterator<F>` qui permettra d'itérer sur les éléments contenus dans tableau `elementData`.

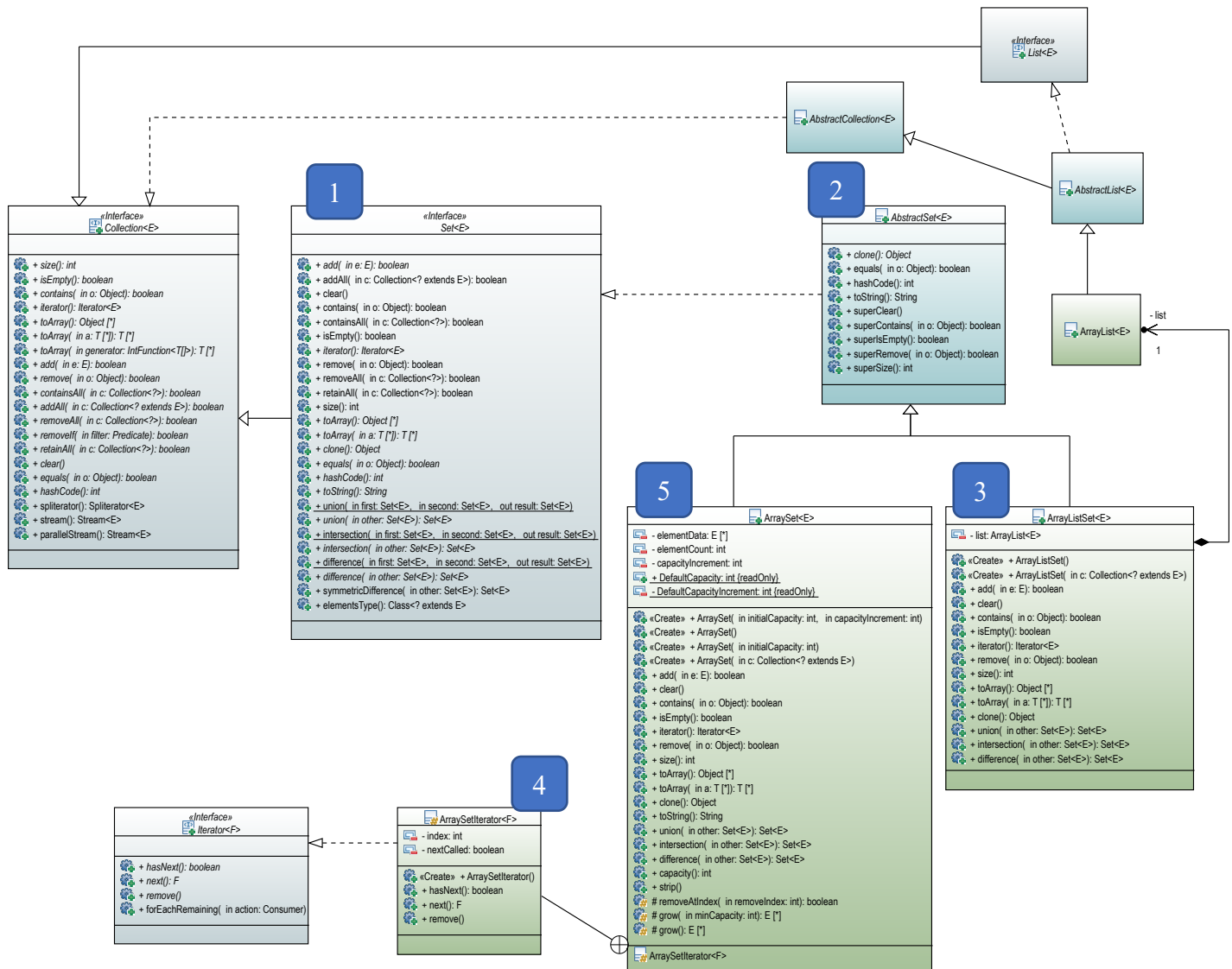
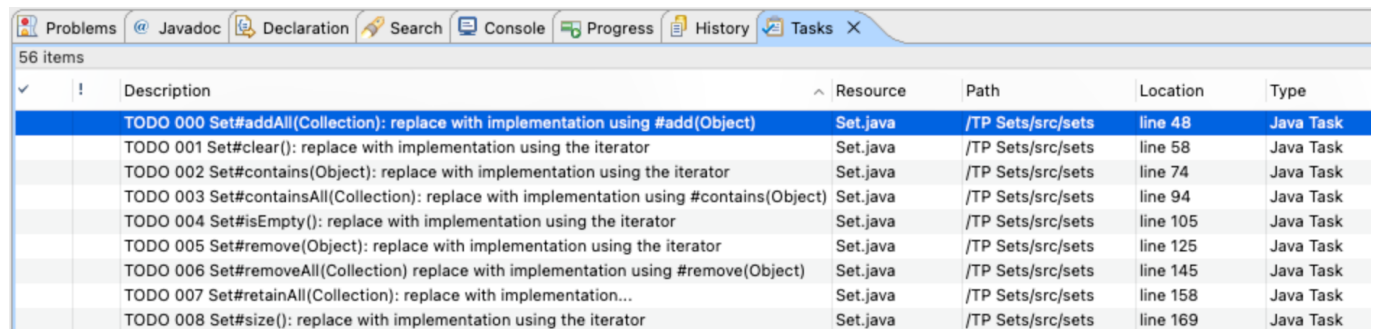


Figure 2 : Interfaces et classes à compléter

- 1) Complétez les méthodes définies par défaut dans l'interface `Set<E>` en utilisant autant que faire ce peut l'itérateur fourni par la méthode `Iterator<E>` `iterator()` (même si celle-ci n'est pas encore implémentée), ou bien d'autres méthodes implémentées par défaut de l'interface `Set<E>`.
- 2) Complétez la classe abstraite `AbstractSet<E>` en implémentant les méthodes :
  - a) `boolean` `equals(Object o)`
  - b) `int` `hashCode()`
  - c) `String` `toString()`

- 3) Complétez la classe concrète `AbstractSet<E>` en implémentant les méthodes `equals`, `hashCode` et `toString`. Les méthodes `superXXX` seront utilisées dans les tests pour comparer l'exécution des opérations fournies dans `Set<E>` et `AbstractSet<E>` aux implémentations fournies dans les classes filles (qui sont censées être plus performantes).
- 4) Complétez l'implémentation de la classe interne `ArraySetIterator<F>` dans la classe `ArraySet<E>`. On notera au passage que l'on ne peut pas utiliser la lettre `<E>` pour le paramètre de type de la classe interne `ArraySetIterator` car ce `<E>` masquerait alors type générique `E` utilisé dans le contexte de la classe `ArraySet<E>`. Nous avons donc utilisé une autre lettre `<F>` ce qui peut entraîner la nécessité de caster en `F`, e.g. : `(F) elementData[i]`;
- 5) Complétez l'implémentation de la classe `ArraySet<E>`.

Vous pourrez avantageusement utiliser l'onglet « Tasks » pour afficher les différents TODOs à réaliser durant ce TP. Si cet onglet n'est pas déjà présent, vous pouvez l'afficher avec Menu Window → Show View → Tasks. Vous pourrez ainsi naviguer rapidement entre les TODOs.



Description	Resource	Path	Location	Type
TODO 000 Set#addAll(Collection): replace with implementation using #add(Object)	Set.java	/TP Sets/src/sets	line 48	Java Task
TODO 001 Set#clear(): replace with implementation using the iterator	Set.java	/TP Sets/src/sets	line 58	Java Task
TODO 002 Set#contains(Object): replace with implementation using the iterator	Set.java	/TP Sets/src/sets	line 74	Java Task
TODO 003 Set#containsAll(Collection): replace with implementation using #contains(Object)	Set.java	/TP Sets/src/sets	line 94	Java Task
TODO 004 Set#isEmpty(): replace with implementation using the iterator	Set.java	/TP Sets/src/sets	line 105	Java Task
TODO 005 Set#remove(Object): replace with implementation using the iterator	Set.java	/TP Sets/src/sets	line 125	Java Task
TODO 006 Set#removeAll(Collection) replace with implementation using #remove(Object)	Set.java	/TP Sets/src/sets	line 145	Java Task
TODO 007 Set#retainAll(Collection): replace with implementation...	Set.java	/TP Sets/src/sets	line 158	Java Task
TODO 008 Set#size(): replace with implementation using the iterator	Set.java	/TP Sets/src/sets	line 169	Java Task

**Figure 3 : Onglet des tâches contenant les TODOs à compléter**

Pour tester les classes que vous aurez complétées vous pourrez utiliser les classes de test du package `tests` :

- `SetTest` : teste les deux implémentations de l'interface `Set<E>` : `ArrayListSet<E>` et `ArraySet<E>`.
- `ArraySetTest` : teste les méthodes spécifiques à la classe `ArraySet<E>`.

Vous pourrez déposer votre projet complété sur le dépôt « ilo-tp-ensembles » sur [exam.ensiie.fr](http://exam.ensiie.fr) avant 12h45.

Pour exporter votre projet sous forme d'archive : Clic droit à la racine de votre projet → Export ... → General → Archive File → next : sélectionnez les fichiers à placer dans l'archive (inutile d'exporter le dossier `bin`, mais n'oubliez pas d'exporter le répertoire `src` ainsi que les fichiers `.project` et `.classpath` à la racine), puis dans le champ « To archive file » sélectionnez le nom de l'archive à créer. Il n'est pas nécessaire de mentionner votre nom dans le nom de l'archive, le dépôt sur [exam.ensiie.fr](http://exam.ensiie.fr) se chargera d'ajouter votre nom à l'archive déposée.