ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

TP Noté ILO
Listes chaînées

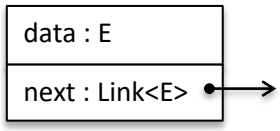
Listes génériques

Le but de cet exercice est d'implémenter une liste chaînée pour stocker des éléments génériques (de type E) puis d'utiliser cette liste chaînée pour réaliser une collection générique (Collection<E>) avec l'aide d'une implémentation partielle fournie par java (AbstractCollection<E>).

Vous pourrez trouver un squelette d'implémentation de cet exercice dans le fichier /pub/ILO/Listes.zip que vous dézipperez dans un répertoire Listes. Importez ensuite le projet existant dans Eclipse.

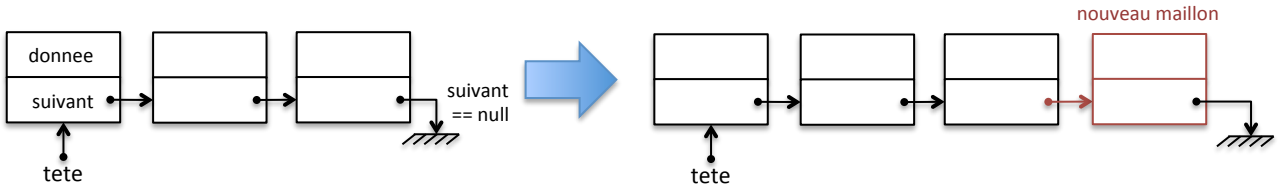
Ce TP est à rendre sur exam.ensiie.fr dans le dépôt **ilo-tp-listes** ouvert jusqu'au 10/04/2020.

Une liste générique utilise un paramètre de type (ici <E>) qui pourra être instancié au moment de son exécution par un type effectif (par exemple ForwardList<String>). Le graphe de classe de la **Figure 1** page 2 décrit une liste générique implémentant l'interface IForwardList<E> qui elle-même hérite de l'interface Iterable<E> en fournissant une méthode iterator() renvoyant un itérateur (qui servira à parcourir la liste). Cet itérateur est fourni ici par une instanciation de la classe ListIterator<E> qui sera définie en tant que classe interne de la liste.
L'élément constitutif d'une liste chaînée est un maillon (classe Link<E>) contenant une donnée (E data) et un lien vers le maillon suivant (Link<E> next) :

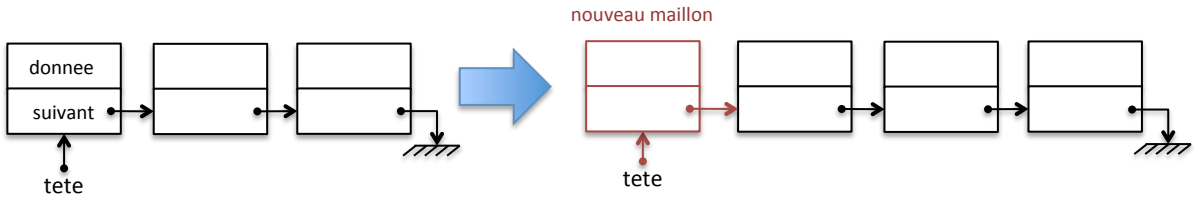


L'interface IForwardList<E> définit les opérations à réaliser sur une liste de la manière suivante :

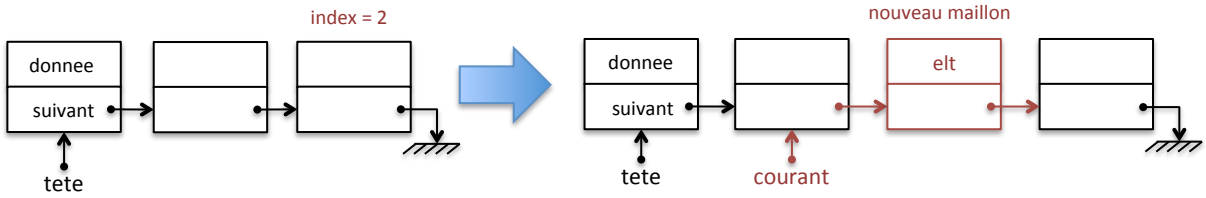
- `void add(E elt)` : ajoute l'élément elt à la fin de la liste en levant une NullPointerException si l'élément à insérer est null.



- `void insert(E elt)` : ajoute l'élément elt en tête de liste en levant une NullPointerException si l'élément à insérer est null.



- `boolean insertAt(E elt, int index)` : Ajoute l'élément elt à la (index+1)ième place. Renvoie vrai si l'élément a pu être inséré à la bonne place, ou false si l'élément n'a pas pu être inséré ou si celui-ci était null.



- `boolean remove(E elt)` : supprime la première occurrence de l'élément elt dans la liste s'il est présent. [Implémentable en tant que méthode par défaut dans `IForwardList<E>` en utilisant l'itérateur]
- `boolean removeAll(E elt)` : supprime toutes les occurrences de l'élément elt dans la liste. [Implémentable en tant que méthode par défaut dans `IForwardList<E>` en utilisant l'itérateur]
- `void clear()` : supprime tous les éléments de la liste. [Implémentable en tant que méthode par défaut dans `IForwardList<E>` en utilisant l'itérateur]
- `boolean empty()` : renvoie vrai si la liste ne contient aucun élément. [Implémentable en tant que méthode par défaut dans `IForwardList<E>` en utilisant l'itérateur]
- `int size()` : renvoie le nombre d'éléments actuellement dans la liste. [Implémentable en tant que méthode par défaut dans `IForwardList<E>` en utilisant l'itérateur]
- `boolean equals(Object o)` : renvoie vrai si o est un `Iterable<E>` de même longueur et contenant les mêmes éléments (dans le même ordre) que la liste. [Non implémentable en tant que méthode par défaut dans `IForwardList<E>` car surcharge une méthode de la superclasse `Object`]
- `int hashCode()` : renvoie le hash code de la liste d'après les hashcodes de ses éléments (voir l'algorithme donné en cours). [Non implémentable en tant que méthode par défaut dans `IForwardList<E>` car surcharge une méthode de la superclasse `Object`]
- `String toString()` : renvoie une chaine de caractère représentant la liste sous la forme « [elt->elt->elt] ». [Non implémentable en tant que méthode par défaut dans `IForwardList<E>` car surcharge une méthode de la superclasse `Object`]
- `Iterator<E> iterator()` : Renvoie une instance de l'itérateur de la liste permettant de la parcourir. [Non implémentable en tant que méthode par défaut . Sera implémenté dans sa classe fille avec un `ListIterator<E>`].

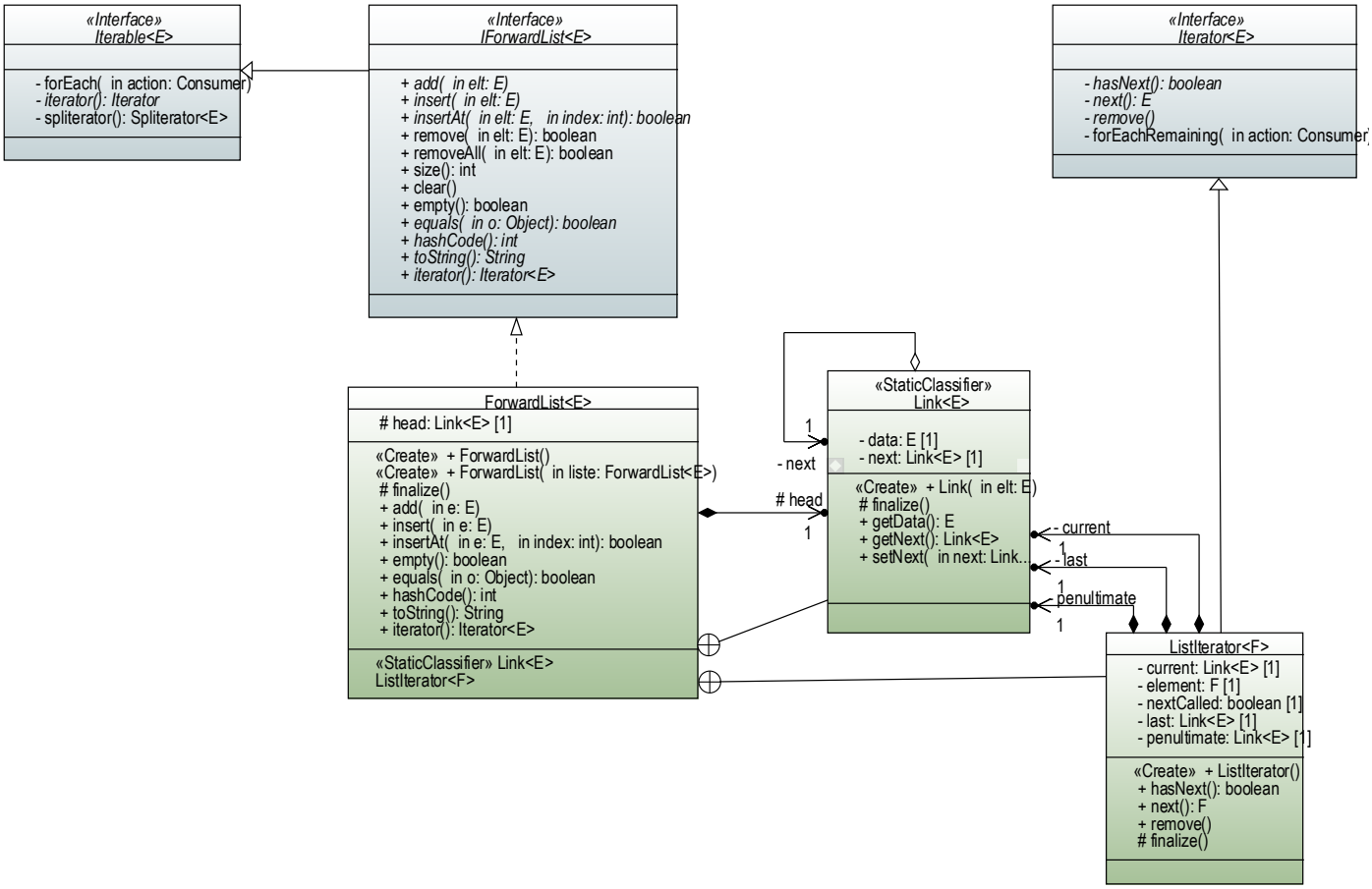


Figure 1 : Graphe de classe d'une liste générique

a) Complétez l'interface `IForwardList<E>`, en implémentant les méthodes par défaut définies dans cette interface grâce au pattern Template Method qui nous permet d'obtenir un itérateur (avec la Factory Method `Iterator<E> iterator()`) même si celui-ci n'existe pas encore.

b) Complétez la classe interne `ForwardList<E>.Link<E>` qui représente un maillon dans une liste en tant que classe imbriquée dans la classe `ForwardList` (Seule la liste utilise des maillons donc cette classe n'a pas besoin d'être déclarée à l'extérieur de la liste).

c) Puis la classe `ForwardList<E>` implémentant l'interface `IForwardList<E>` et contenant la classe imbriquée (static) privée `Link<E>` et une classe interne privée `ListIterator<F>` implémentant l'interface `Iterator<F>`.

Complétez la class `ForwardList<E>` implémentant l'interface `IForwardList<E>` en utilisant autant que faire ce peut l'itérateur fourni par la méthode `iterator() {return new ListIterator<E>();}` lorsque vous avez besoin de parcourir la liste. La classe interne `ListIterator<E>` est un itérateur dans le sens où elle permet de parcourir les différents éléments de la liste grâce à sa méthode `next()`. Typiquement le parcours d'une liste grâce à cet itérateur se fera de la manière suivante :

```
for (Iterator<E> it = iterator(); it.hasNext();)
{
        … it.next() …
}
```

d) La classe `ListIterator<F>` est une classe interne à `ForwardList<E>` et implémentant l'interface `Iteratror<F>`. Comme c'est une classe interne (non-static) elle a accès aux différents membres de `ForwardList<E>`, y compris la tête de liste (`head`). En contrepartie, le paramètre générique `F` de `ListIterator<F>` doit être différent du paramètre générique `E` de `ForwardList<E>` afin que celui-ci ne "masque" pas le parmètre générique de `ListIterator<F>`. Implémentez le constructeur, la méthode `hasNext` et la méthode `next` de la classe `ListIterator<F>` : Vous aurez sans doute besoin dans le constructeur de « caster » le maillon `Link<E> head` en `Link<F>` pour initialiser le maillon `current` de l'itérateur.

e) Complétez la collection (`CollectionList<E>`) utilisant comme conteneur sous-jacent une `ForwardList<E>` en vous aidant de la classe `AbstractCollection<E>` (voir l'annexe page 5) qui est une implémentation partielle de l'interface `Collection<E>` (voir Figure 2, ci-dessous). Il suffit pour cela d'implémenter les constructeurs, la méthode `add`, la méthode `iterator`, la méthode `size` ainsi que les méthodes `hashCode` et `equals`.

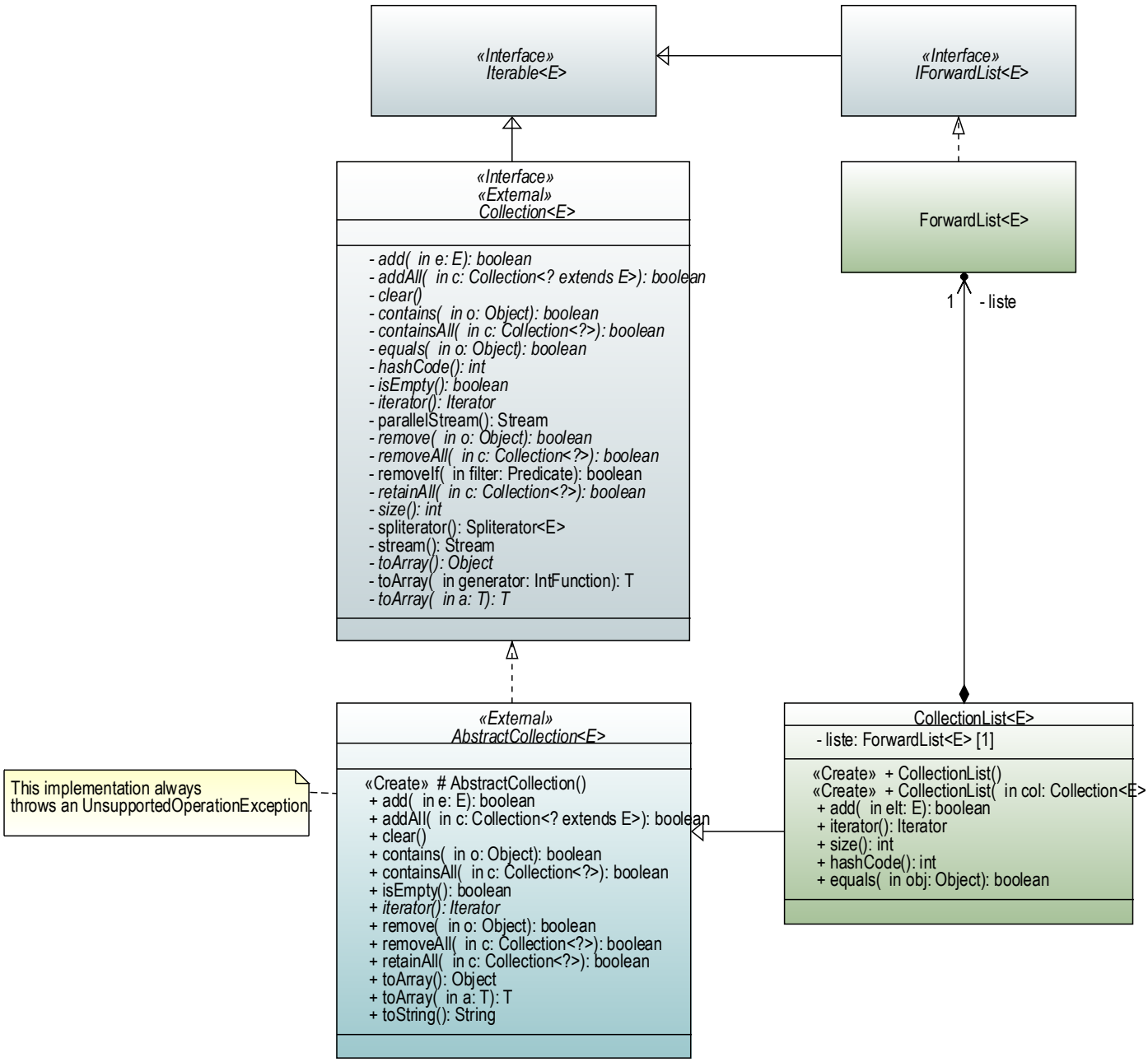


Figure 2 : Collection à partir d'une liste

La méthode `remove` du `ListIterator<F>` est déjà implémentée : La méthode `remove` de l'itérateur est censée détruire l'élément qui vient d'être renvoyé pour la méthode `next`. La méthode `remove` n'est donc pas appelable tant que la méthode `next` n'a pas été appelée. Il faudra prendre en compte cette alternance entre les méthodes `next` et `remove` dans l'implémentation de l'itérateur (attribut `nextCalled`). Par ailleurs, comme la liste n'est que simplement chainée (un maillon ne possède qu'un lien vers le suivant). Supprimer l'élément qui vient d'être renvoyé par `next` revient donc à relier l'avant dernier élément (`penultimate`) avec l'élément courant (`current`).

N'oubliez pas par ailleurs que `remove` ne peut pas être appelée deux fois de suite sans que `next` soit appelé entre temps.

Annexe : Extraits de la documentation de la classe AbstractCollection

```
public abstract class AbstractCollection<E>
extends Object
implements Collection<E>
```

This class provides a skeletal implementation of the `Collection` interface, to minimize the effort required to implement this interface.

To implement an unmodifiable collection, the programmer needs only to extend this class and provide implementations for the `iterator` and `size` methods. (The iterator returned by the `iterator` method must implement `hasNext` and `next`.)

To implement a modifiable collection, the programmer must additionally override this class's `add` method (which otherwise throws an `UnsupportedOperationException`), and the iterator returned by the `iterator` method must additionally implement its `remove` method.

The programmer should generally provide a void (no argument) and `Collection` constructor, as per the recommendation in the `Collection` interface specification.

The documentation for each non-abstract method in this class describes its implementation in detail. Each of these methods may be overridden if the collection being implemented admits a more efficient implementation.

add

```
public boolean add(E e)
```

Ensures that this collection contains the specified element (optional operation). Returns `true` if this collection changed as a result of the call. (Returns `false` if this collection does not permit duplicates and already contains the specified element.)

Collections that support this operation may place limitations on what elements may be added to this collection. In particular, some collections will refuse to add `null` elements, and others will impose restrictions on the type of elements that may be added. Collection classes should clearly specify in their documentation any restrictions on what elements may be added.

If a collection refuses to add a particular element for any reason other than that it already contains the element, it *must* throw an exception (rather than returning `false`). This preserves the invariant that a collection always contains the specified element after this call returns.

This implementation always throws an `UnsupportedOperationException`.

Specified by:
`add` in interface `Collection<E>`
Parameters:
`e` - element whose presence in this collection is to be ensured
Returns:
`true` if this collection changed as a result of the call
Throws:
`UnsupportedOperationException` - if the `add` operation is not supported by this collection
`ClassCastException` - if the class of the specified element prevents it from being added to this collection
`NullPointerException` - if the specified element is null and this collection does not permit null elements
`IllegalArgumentException` - if some property of the element prevents it from being added to this collection
`IllegalStateException` - if the element cannot be added at this time due to insertion restrictions