

Author : Christophe Garion <garion@isae.fr>
Public : SUPAERO 2A
Date :

Résumé

Le but de ce TP est de manipuler la notion d'interface pour réaliser un itérateur sur un agrégat.

1 Objectifs

Les objectifs de ce TP sont les suivants :

- comprendre la notion d'interface ;
- écrire des interfaces ;
- écrire des classes réalisant des interfaces ;
- découvrir un premier patron de conception, l'itérateur.

2 Présentation du problème

Supposons que l'on dispose d'une classe représentant une liste sous forme d'un tableau pour stocker des objets quelconques servant dans une application. L'accès aux éléments dans ce tableau se fait en utilisant un index et permet des recherches avec une *complexité* particulière à l'ensemble.

Supposons maintenant que l'on se rende compte qu'un arbre binaire serait beaucoup plus adapté pour l'application que l'on développe. Dans ce cas, l'accès aux éléments stockés dans l'arbre binaire se fait selon un autre mécanisme. Si l'on a construit l'application en ne considérant que l'ensemble implanté sous forme de tableau, on va devoir revoir une grande partie du logiciel.

Lorsque l'on dispose d'un agrégat d'objets (comme par exemple une liste implantée sous forme d'un tableau, une liste chaînée ou un arbre), il est utile de disposer d'un mécanisme d'accès commun à ses éléments (pour les afficher par exemple). Nous allons mettre en œuvre le mécanisme d'*itérateur*. Un itérateur permet de réaliser un parcours de tous les éléments de l'agrégat.

Des exemples d'agrégats d'éléments de type **double** et les itérateurs associés sont présentés sur la figure 1.

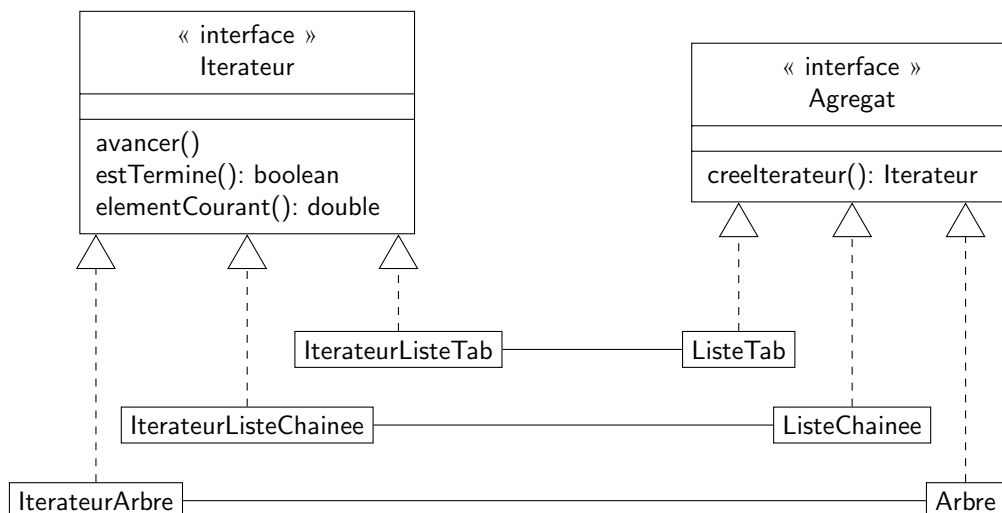


FIGURE 1 – Diagramme de classe présentant les interfaces Iterateur et Agregat et quelques agrégats et leurs itérateurs associés

Un itérateur sur un agrégat permet donc d'accéder aux éléments de l'agrégat suivant un parcours déterminé en protégeant la structure interne de l'agrégat. On peut ainsi disposer d'un mécanisme commun à tous les types d'agrégats et à tous les parcours possibles sur un même type d'agrégat. Cette solution de conception est ce que l'on appelle un *design pattern* (patron de conception), cf. [2, 1].



Pour réaliser le TP, vous allez devoir créer un projet Java sous Eclipse en utilisant votre dépôt Subversion. Si vous avez configuré votre dépôt pour qu'il soit disponible dans la vue *SVN Repository Exploring* d'Eclipse, vous créez votre projet en faisant un *checkout* à partir du répertoire **TP4** de votre dépôt en cliquant droit dessus. N'oubliez pas de configurer votre *build path* pour pouvoir utiliser les éventuelles archives JAR placées dans le répertoire *lib*.

3 Implantation des interfaces

1. le diagramme UML de la figure 1 présente deux interfaces spécifiant les services rendus par deux catégories d'objets :
 - les itérateurs, qui fournissent les services suivants :
 - `avancer()` qui positionne l'élément courant sur l'élément suivant de l'agrégat ;
 - `estTermine()` qui renvoie un booléen qui est vrai si on a terminé le parcours de l'agrégat. Cela signifie que *estTermine()* sera vrai si et seulement si l'on est au delà du dernier élément de l'agrégat ;
 - `elementCourant()` qui renvoie un **double** qui est l'élément courant dans l'agrégat.
 - les agrégats, qui fournissent un seul service, `creerIterateur()` qui renvoie un `Iterateur` sur l'agrégat.

Écrire le code Java correspondant aux deux interfaces.

2. écrire une classe Utilitaire possédant une méthode `nbElements` prenant un `Agregat` en paramètre et renvoyant le nombre d'éléments de l'agrégat.

4 Réalisation d'un agrégat et de son itérateur

On va maintenant réaliser un agrégat particulier, `ListeChaine` et son itérateur associé, `IterateurListeChaine`. Vous trouverez la classe `ListeChaine` dans votre dépôt (répertoire `src` et les autres classes nécessaires sont disponibles dans l'archive `lab4.jar` disponible sous le répertoire `lib`. `ListChaine` réalise l'interface `Liste` vue en cours.

1. écrire le diagramme UML présentant les relations existant entre les différentes classes et interfaces du problème ;
2. on va maintenant compléter la classe `ListeChaine` pour qu'elle réalise l'interface `Agregat` avant de construire la classe correspondant à l'itérateur associé, `IterateurListeChaine`. Réfléchir en particulier à ce dont on a besoin pour construire l'itérateur associé ;
3. écrire la classe `IterateurListeChaine` et utiliser `JUnit` pour tester les méthodes de la classe ;
4. tester de façon élémentaire la classe `IterateurListeChaine` en utilisant la classe `Utilitaire`.

Références

- [1] E. FREEMAN et al. *Head first design patterns*. O' Reilly, 2005.
- [2] E. GAMMA et al. *Design Patterns - Elements of reusable object-oriented software*. Addison-Wesley professional computing series. Addison-Wesley, 1994.

License CC BY-NC-SA 3.0



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported license (CC BY-NC-SA 3.0)

You are free to Share (copy, distribute and transmute) and to Remix (adapt) this work under the following conditions:



Attribution – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Noncommercial – You may not use this work for commercial purposes.



Share Alike – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

See <http://creativecommons.org/licenses/by-nc-sa/3.0/> for more details.