

TME 8 : Collections ad hoc

Objectifs pédagogiques : création d'une Collection générique ad hoc, itérateur générique.

Lors du TME2, nous avons choisi d'implémenter la population avec une `ArrayList`. On souhaite dans cet exercice réaliser notre propre implémentation d'une `ArrayList` contenant une liste de tableaux.

8.1 MyArrayList

La classe `MyArrayList` est une liste de tableaux de taille constante, ce qui permet :

- d'ajouter des éléments en créant un nouveau chaînon contenant un tableau dans la liste, opération de complexité constante ;
- d'accéder aux éléments avec une bonne complexité tant que la liste ne contient pas trop de chaînons (trouver le bon tableau = linéaire dans la liste, accès aux cases de chaque tableau = temps constant).

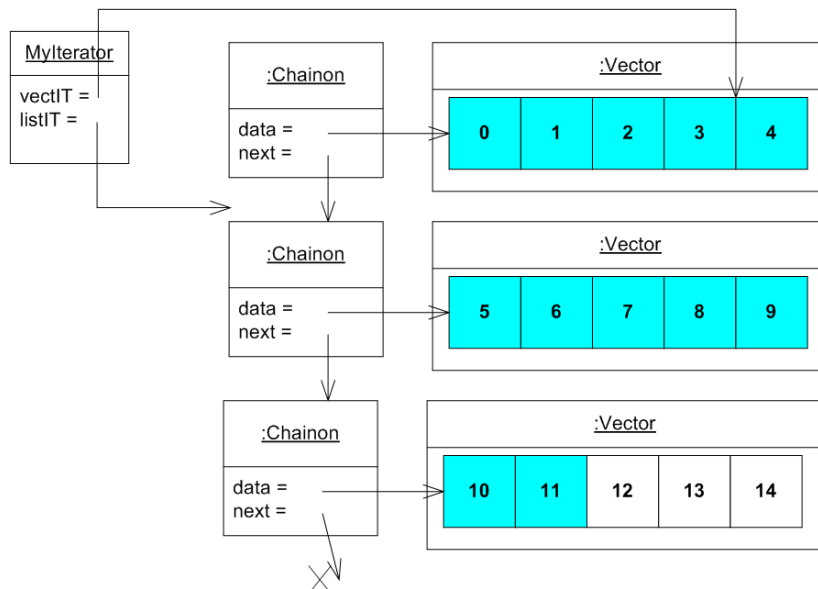


Figure 1: Illustration de l'organisation d'une `MyArrayList` et de ses itérateurs internes. Les vecteurs sont alloués ici avec une taille de 5. La `MyArrayList` contient ici 12 éléments (indices 0 à 11), et le dernier vecteur n'est pas encore plein. L'ajout d'un 16ème élément doit provoquer l'allocation d'un nouveau chaînon et de son vecteur. Le `MyIterator` vient de lire ici le quatrième élément (indice 3): son `listIt` pointe juste après le premier élément de la liste et son `vectIt` pointe juste après le quatrième élément de ce vecteur. NB : les `Iterator` de Java sont toujours entre deux éléments.

`MyArrayList` est donc une structure qui profite d'une bonne complexité en insertion/allocation de la liste et en accès du vecteur.

L'exercice a vocation à s'entraîner avec les collections. On utilisera donc une `LinkedList` pour la liste et des `Vector` plutôt que des tableaux (`T[]`) pour le stockage. Notre implémentation s'appuiera donc par délégation sur une `LinkedList<Vector<T>`.

On se donne une taille d'allocation des blocs ou vecteurs, qu'on supposera constante (attribut `final`) au cours de la vie de l'`ArrayList`. Ceci est une hypothèse simplificatrice, en pratique on double la taille des tableaux alloués à chaque réallocation.

⇒ Dans le répertoire TME8, créez un projet `tme8`. Chargez l'archive du TME7. **Créez un nouveau package `pobj.util`**. Dans ce package, **créez une nouvelle classe générique `MyArrayList<T>`**. Vous doterez cette classe de deux constructeurs, un sans paramètre qui positionne une taille par défaut pour les tableaux alloués et un qui prend cette taille en paramètre. Créez dans ce constructeur une instance de `LinkedList` vide.

⇒ Implémentez dans `MyArrayList<T>` les méthodes déclarées dans `Collection<T>` suivantes :

- **`public boolean add(T object)`**
NB : si le dernier `Vector` est plein (`size() = capacité`), en allouer un nouveau. Si la liste est vide, allouer un chaînon. Pour allouer un vecteur de capacité `N` faire `new Vector(N)`.
- **`public T get(int location)`**
NB : on peut calculer directement avec des modulo et des divisions les bonnes positions
- **`public T set(int location, T object)`**
NB : on peut calculer directement avec des modulo et des divisions les bonnes positions
- **`public int size()`**
NB : on pourrait mesurer juste le dernier vecteur et la taille de la liste

Pour accéder au dernier élément de la liste on pourra utiliser la forme :

```
Vector<T> last = list.getLast();
```

... à condition que la liste ne soit pas vide.

⇒ Testez le comportement : créez un `MyArrayList` d'entiers, ajoutez des valeurs dans la liste... Mettez quelques traces (`System.out`) pour vous assurer que les réallocations se produisent bien là où vous le pensez.

8.1.1 Itérateurs

On veut pouvoir itérer sur notre `MyArrayList` avec une boucle *foreach*. Quelle interface faut-il implémenter ? Ajoutez la déclaration `implements` adaptée à `MyArrayList<T>`.

Pour réaliser un `Iterator<T>` pour notre classe `MyArrayList`, on créera une nouvelle classe de visibilité package.

Cet itérateur s'appuiera par délégation sur deux itérateurs, un qui itère sur la liste globale de `MyArrayList` et un sur chaque vecteur local au sein de la liste, comme illustré sur la figure 1. Appelez-les dans la suite `listIT` et `vectorIT`.

⇒ Donnez le type de ces deux itérateurs.

⇒ Réalisez cet itérateur `MyIterator` sur `MyArrayList`.

Le constructeur, qui prend un `List<Vector<T>` en paramètre, positionne le curseur `listIT` sur le début de la liste, et le curseur `vectorIT` à vide (`hasNext() = false`). Pour positionner à vide, on utilisera la constante `static Collections.EMPTY_LIST`.

L'opération `hasNext()` teste si au moins un des deux curseurs n'est pas arrivé au bout de son itération. On considère donc qu'on n'a pas de chaînon vide dans la liste (ce qui doit être le cas, il faudrait juste faire attention à bien désallouer dans le `remove()` si on l'implémentait).

L'opération `next()` teste si `vectorIT` est au bout. Si c'est le cas, il décale le curseur `listIT` et repositionne `vectorIT` sur le prochain élément de la liste. Ensuite il rend le `next()` de l'itérateur `vectorIT`. On suppose donc que le dernier vecteur n'est jamais vide.

L'opération `remove()` lève une `UnsupportedOperationException`.

⇒ Dans votre classe `Population`, remplacez la `List` ou `ArrayList` que vous utilisiez par une `MyArrayList`. Quelles opérations manquent encore pour compiler la classe `Population` ?

⇒ Ajoutez à `MyArrayList` le fait qu'elle étend `java.util.AbstractList<T>`. Ajoutez à `MyArrayList` un constructeur qui prend une `Collection` en paramètre, afin de se conformer au « contrat » d'une `Collection`.

Votre population doit maintenant être en mesure de fonctionner avec une `MyArrayList`.

⇒ Documentez votre classe et développez des tests unitaires pour `MyArrayList`.

⇒ Lancez votre batterie de tests et assurez-vous du bon fonctionnement.

NB: L'itérateur développé plus haut n'est pas redondant avec celui fourni via `AbstractList`, qui s'appuie sur `get(index)` dans son implémentation.)

Récupérez sur le site de l'UE la classe `Chrono` fournie. Branchez-la dans votre application de façon à comparer la performance de calcul entre vos deux versions de la classe `Population`. Vous lancerez pour cela votre application avec une population contenant 10 000 individus.

8.2 Remise du TME

⇒ Copiez-collez le code de votre opération `add` de `MyArrayList` et le code de vos opérations `next()` et `hasNext()` dans l'itérateur. Donnez les résultats de votre étude de performance de vos deux implémentations de la population.