TP collaboratif : Tableaux triés (Algorithmique – Programmation FIP – ING39)

V. Aponte, P. Courtieu

Ecriture de spécification informelle avec contrats de méthodes et jeu des tests couvrants ces contrats. Implantation de tableaux triés. Travail collaboratif.

Tableaux triés extensibles

On souhaite définir une interface puis écrire une classe d'implantation afin de modéliser des tableaux d'entiers triés **et** extensibles. Dans un tableau *extensible* on peut ajouter ou enlever des composantes, et obtenir la valeur d'une composante par son indice. Un tableau *trié* a la propriété d'être toujours trié, autrement dit, après création, avant et après chaque opération. Notez que dans ce Tp, nous travaillerons avec des *tableaux triés par construction*, autrement dit, un objet tableau trié **ne possède pas de méthode de tri** (voir les contraintes d'implantation pour plus de détails).

Définition de l'interface `TableauTrie`

Dans cet exercice vous devez définir une *interface java complète* : vous devez décider quelles seront les opérations disponibles dans tout tableau trié extensible. Votre interface doit définir également tous les contrats : l'invariant qui décrit un état interne cohérent, et les contrats pour toutes les méthodes au format javadoc.

En plus des opérations et contraintes mentionnés plus haut, on veut disposer des opérations suivantes :
— une méthode pour tester si un **tableau trié passé en paramètre** a les mêmes composantes que le tableau trié courant,
— une méthode `fusion` qui prend un tableau trié t et qui retourne en résultat un *nouveau tableau trié*, résultat de la fusion de t et du tableau trié courant.
— d'autres méthodes pourront être nécessaire pour rendre utilisibles ces tableaux.

Création d'un scénario pour valider votre interface

Avant de passer aux tests, ou à l'implantation, il est très important de valider la pertinence de votre spécification d'interface. Il faut pouvoir répondre à la question :

mon interface contient toutes les méthodes dont on peut avoir besoin sur ce genre de tableau ?

Nous vous proposons une manière de mettre votre interface à l'épreuve. Créez une nouvelle classe `Scenario`, où vous placerez une méthode main (vide pour l'instant), et la méthode statique suivante :

```
public static void jouerScene(TableauTrie t1, TableauTrie t2){}
```

Cette classe vous servira à coder un scénario d'utilisation de l'interface. Notez que cet exercice est à faire *après avoir fini d'écrire votre interface*, mais **avant de coder** son implantation, et **avant** d'écrire le moindre test. Dans le corps de cette méthode vous devez coder les actions suivantes :

1. afficher la taille de `t1` et la taille de `t2`.
2. afficher l'élément de position 1 dans `t2` en s'assurant que cette case existe bien dans `t2`.

3. ajouter successivement 7, 3, 10, 3 dans `t1` ;
4. afficher le min et le max de `t2`.
5. afficher le contenu de `t1` avant chaque ajout ;
6. obtenir l'indice de 3 dans `t1`.
7. retirer l'élément d'indice 0 dans `t1` ;
8. afficher la taille de `t1` et son contenu.
9. calculer la fusion de `t1` et de `t2` et afficher son contenu.

Si vous avez des problèmes pour écrire ce scénario, probablement votre interface ne contient pas toutes les méthodes nécessaires, ou alors le comportement que vous avez prévu n'est pas adapté. Plus tard, une fois votre interface implantée, vous serez capable de créer des objets de type `TableauTrie` dans le code de cette classe, et les utiliser pour jouer ce scénario.

Travail collaboratif

On formera des équipes de 3 à 4 personnes maximum.

1. Tous les membres de chaque équipe participent à la discussion sur la conception de l'interface.
2. Une fois l'interface validée (utilisez le scénario proposé pour vous aider), le travail de test et d'implantation sera distribué. Vous pouvez envisager plusieurs modes de fonctionnement. Par exemple :
   — un chef de projet pilote la repartition des tâches, les commits, etc.
   — chaque méthode est prise en charge par un binôme programmeur/testeur
   — deux sous-équipes travaillent l'une sur l'implantation, l'autre sur les tests.
   — tout autre organisation qui vous paraîtrait efficace.

Lors de chaque étape de conception et développement on veille à commiter uniquement du code java qui compile, et des contrats lisibles en javadoc.

Remarque Importante : Avec GIT Lorsqu'on fusionne (git pull ou git merge), on fait un nouveau commit. Lorsque celui-ci ne se fait pas automatiquement (conflits) il faut le faire après la résolution des conflits ET IL FAUT COMMITER TOUS LES FICHIERS. Sans quoi on annule certaines modification faites par les autres (et elles ne sont pas simples à récupérer). Au besoin faites appel aux enseignants pour vérifier avant le commit.

Contraintes d'implantation

On peut tirer avantage de la propriété d'être toujours trié afin de rendre plus efficaces certaines opérations. On impose donc les contraintes suivantes :

— aucune opération ne réalise de tri sur le tableau courant.
— l'opération de fusion fonctionne comme l'insertion du tri fusion (*merge sort*) dont la complexité est linéaire. Évidemment, elle fait l'hypothèse que le tableau courant et celui passé en paramètre **sont triés**.

Par exemple, pour implanter :

— Tester si n appartient au tableau : dans un parcours séquentiel du tableau trié, au 1er élément trouvé plus grand que n, on peut conclure que n est absent du tableau, sans aller jusqu'à la fin du tableau.
— Ajout n : on doit l'insérer à sa place dans le tableau trié. On doit pour cela parcourir le tableau trié pour chercher la bonne place, déplacer tous les éléments à droite de cette place, et y insérer l'élément.
— Le tri fusion parcourt les deux tableaux `t1` et `t2` en parallèle avec deux indices `i` et `j`. On insère dans le tableau résultat l'élément le plus petit parmi `t1[i]` et `t[j]`, et on fait avancer en conséquence l'un parmi `i` et `j`.

Challenge final : utilisation de votre implantation

Une fois votre classe implantée et testée, écrivez une méthode statique `rechercheDicothomique` qui implante la recherche dichotomique d'un entier dans un tableau trié.