Développement WEB : JEE

TP 2 : La généricité et les Structures de Données Avancées

Objectifs : Dans ce TP vous allez écrire une classe pour représenter un arbre binaire de recherche dans lequel les éléments seront rangés selon leur ordre naturel (ou selon un ordre déterminé, pour une des dernières questions).

1ère partie : Classe Nœud

Cette classe représentera un nœud d'un arbre binaire (*pas nécessairement d'un arbre binaire de recherche*). Un nœud a une valeur et peut avoir 2 nœuds fils : le nœud gauche et le nœud droit.

La classe Nœud doit être **générique**. On devra pouvoir indiquer le type E des valeurs qui seront conservées dans le nœud. Les éléments de E ne sont pas nécessairement ordonnés.

Mettez dans la classe

- un constructeur sans paramètre,
- un constructeur qui prend une valeur en paramètre,
- un constructeur qui prend une valeur et les 2 nœuds gauche et droit en paramètres,
- les accesseurs et les modificateurs pour la valeur et les 2 nœuds gauche et droit,
- une méthode toString() qui affiche les valeurs du nœud et les sous-nœuds attachés à ce nœud.

Vous écrirez une classe TestNœud pour tester votre classe. Testez seulement la création d'un nœud qui contient un nœud fils ; utilisez la méthode toString() pour vérifier que tout marche bien.

2ème partie : Classe ArbreBinaireRecherche

Cette classe sera générique. On pourra indiquer le type E des éléments contenus dans l'arbre.

Un arbre binaire de recherche a un nœud racine. Pour tout nœud de l'arbre

- le sous-nœud gauche est la racine d'un arbre qui contient les éléments inférieurs ou égaux à la valeur du nœud ;
- le sous-nœud droit est la racine d'un arbre qui contient les éléments supérieurs à la valeur du nœud.

Un arbre binaire de recherche doit nécessairement travailler avec une relation d'ordre sur les éléments de type E (ça n'est pas nécessaire pour le nœud d'un arbre binaire quelconque). Vous allez commencer par écrire un arbre dont les éléments ont un ordre naturel, comme c'est le cas, par exemple, pour Integer ou String. Cet ordre naturel sera représenté par l'interface **générique** java.lang.Comparable. C'est à vous de compléter la définition de Comparable pour trouver la bonne instanciation.

Cette classe contiendra 2 constructeurs de signatures : ArbreBinaireRecherche() (construit un arbre vide) et ArbreBinaireRecherche(E) (indique la valeur de la racine de l'arbre).

Elle contiendra aussi

- une méthode pour ajouter un élément dans l'arbre ;
- une méthode qui indique si un objet appartient à l'arbre ;
- une méthode pour afficher tous les éléments de l'arbre dans leur ordre naturel.

La recherche dans un arbre binaire d'un nœud ayant une clé particulière est un procédé récursif. On commence par examiner la racine. Si sa clé est la clé recherchée, l'algorithme termine et renvoie la racine. Si elle est strictement inférieure, alors elle est dans le sous-arbre gauche, sur lequel on effectue alors récursivement la recherche. De même si la clé de la racine est strictement supérieure à la clé recherchée la recherche continue sur le sous-arbre droit. Si on atteint une feuille dont la clé n'est pas celle recherchée, on sait alors que la clé recherchée n'appartient à aucun nœud. On peut la comparer avec la recherche par dichotomie qui procède à peu près de la même manière sauf qu'elle accède directement à chaque élément d'un tableau au lieu de suivre des liens.

3ème partie : Test de ArbreBinaireRecherche avec une instanciation

Écrivez une classe `TestArbreBinaire` qui crée un arbre binaire qui contient des `Integer` et un autre qui contient des `String`. Pour tester avec des entiers, vous pouvez générer un grand nombre d'entiers de façon aléatoire en utilisant la classe `java.util.Random` et sa méthode `nextInt()` ou `nextInt(int)`.

Testez aussi avec un arbre binaire qui contient des employés. Utilisez pour cela les 2 classes `Personne` et `Employe` (classe fille de `Personne`). Les employés seront rangés dans l'arbre suivant l'ordre alphabétique de leur nom.

4ème partie : Arbre binaire de recherche pour des éléments sans ordre naturel

Les types qui ont un ordre naturel sont assez rares. On veut pouvoir ranger les éléments d'autres types dans un arbre binaire de recherche, en indiquant le critère de comparaison. Par exemple, on voudrait ranger des employés dans un arbre binaire en les rangeant suivant la valeur de leur salaire ou suivant l'ordre alphabétique de leur nom. On veut aussi pouvoir ranger les éléments selon un autre ordre que leur ordre naturel s'ils en ont un.

Modifiez votre classe `ArbreBinaireRecherche` en conséquence. Testez avec un arbre qui contient des employés, d'abord en les rangeant par ordre alphabétique des noms, puis par ordre des salaires croissants.

5ème partie : Les itérateurs

Ajoutez ce qu'il faut pour que l'arbre binaire soit parcourable par une boucle "for-each". Ajoutez un test dans la classe `TestArbreBinaire` vérifier que ça fonctionne.

Plusieurs solutions sont possibles. Sans doute la plus simple est de copier les éléments de l'arbre dans une liste et de renvoyer l'itérateur de la liste. Sinon vous aurez besoin de remonter dans l'arbre vers la racine. Vous pouvez le faire en ajoutant dans les nœuds une référence vers le nœud parent ou en utilisant une pile pour garder le parcours depuis la "racine" des nœuds non encore traités vers le nœud "courant".

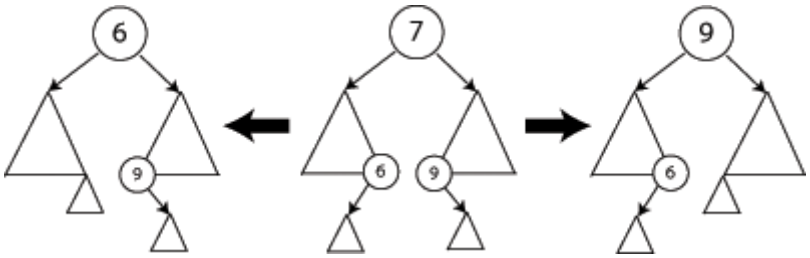
Testez avec une boucle "for-each" dans `TestArbreBinaireRecherche`.

6ème partie : Pour ceux qui ont fini

Complétez le code pour ajouter une méthode qui supprime un élément de l'arbre.

La Suppression pour les arbres binaires, plusieurs cas sont à considérer, une fois que le nœud à supprimer a été trouvé à partir de sa clé :

- ***Suppression d'une feuille** : Il suffit de l'enlever de l'arbre vu qu'elle n'a pas de fils.*
- ***Suppression d'un nœud avec un enfant** : Il faut l'enlever de l'arbre en le remplaçant par son fils.*
- ***Suppression d'un nœud avec deux enfants** : Supposons que le nœud à supprimer soit appelé N (le nœud de valeur 7 dans le graphique ci-dessous). On échange le nœud N avec son successeur le plus proche (le nœud le plus à gauche du sous-arbre droit - ci-dessous, le nœud de valeur 9) ou son plus proche prédécesseur (le nœud le plus à droite du sous-arbre gauche - ci-dessous, le nœud de valeur 6). Cela permet de garder une structure d'arbre binaire de recherche. Puis on applique à nouveau la procédure de suppression à N, qui est maintenant une feuille ou un nœud avec un seul fils.*



Pour une implémentation efficace, il est déconseillé d'utiliser uniquement le successeur ou le prédécesseur car cela contribue à déséquilibrer l'arbre.