Récursivité multiple et début des arbres.

Un peu de code vous est fourni pour ce TP. Vous le retrouverez sur e-campus. Vous n'êtes a priori pas censé modifier les classes fournies, mais plutôt ajouter des classes qui feront votre traitement.

Dans ce TP, les questions précédées d'une (*) sont plus difficiles, ou un peu en avance sur les attentes. Cela signifie que toutes les questions sans (*) sont à faire avant le TD de la semaine suivante!

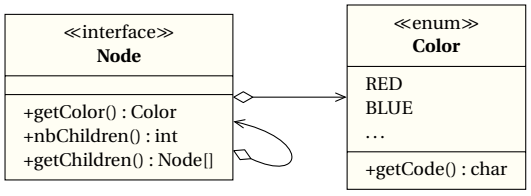
1 Parcours en profondeur d'arbres

Vous trouverez dans le paquetage du tp une classe `AnagramNode`, qui permet de manipuler des arbres de lettres.

1. Écrivez une méthode permettant de faire un parcours en profondeur avec affichage prefixe de l'arbre. Pour rappel, cela veut dire que l'on affiche l'étiquette d'un noeud la première fois qu'on le rencontre. Par exemple, sur l'arbre de l'exemple 2, vous devez trouver le mot "gerant".
2. Écrivez une autre méthode pour faire maintenant un parcours en profondeur avec affichage postfixe. Cela signifie que vous devez afficher le contenu d'un noeud la dernière fois que vous le rencontrez (au moment de remonter dans l'arbre). Pour l'arbre exemple 2, vous devriez trouver le mot "grenat".

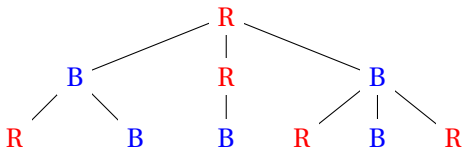
2 Arbres coloriés

Une structure d'arbre vous est fournie dans le fichier interface `Node.java`.
Nous vous rappelons que **vous pouvez introduire des fonctions intermédiaires** si elles vous paraissent nécessaires.



3. Complétez la fonction *countColor(Node root, Color color)* pour qu'elle compte le nombre de nœuds de la couleur donnée dans l'arbre.

On souhaite encoder un arbre par une chaîne de caractère. Le principe est le suivant : chaque couleur a un code d'une lettre intégré. Un nœud est codé par sa couleur, suivi si le nœud a des enfants du caractère '<', des codes de ses enfants accolés, puis du caractère '>'.
L'arbre à droite est donc codé par le code : R<B<RB>R<B>B<RBR>>



4. Complétez la fonction *encode(Node root)* pour qu'elle retourne le code correspondant à l'arbre.

Pour qu'il n'y ait pas d'ambiguïté, on rappelle la convention que la profondeur de la racine est 0.

5. Complétez la fonction *countAtDepth(Node root, Color col, int depth)* pour qu'elle retourne le nombre de nœuds de la couleur donnée à la profondeur donnée.

Une dernière question pour le sport. On donne un score à chaque nœud de l'arbre. Le score d'un nœud est initialement de 1. De plus, chaque nœud ajoute à son score le score de chaque enfant dont la couleur est différente de la sienne. Dans l'arbre ci-dessus, le score de la racine est 6, ses enfants ont pour score 2, 2 et 3, les feuilles valent toutes 1.

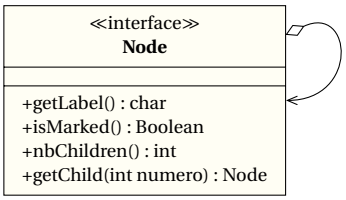
6. Complétez la fonction *score(Node tree)* pour qu'elle retourne le score du nœud donné en entrée. Vous pouvez passer par des fonctions intermédiaires si vous souhaitez ajouter des arguments à vos fonctions.

3 Petit parcours en largeur

7. En reprenant les arbres du début du TP, implémentez un parcours en largeur des arbres. Sur l'arbre exemple2, vous devriez obtenir "geatrn".

4 Arbres marqués

Une structure d'arbre vous est fournie dans le fichier interface `Node.java`. Pour les affichages, utilisez la fonction `affiche` fournie dans `Reponse.java`. Nous vous rappelons que vous pouvez introduire des fonctions intermédiaires si elles vous paraissent nécessaires.



8. Complétez la fonction *afficheMarques(Node tree)* pour qu'elle affiche dans l'ordre d'un parcours préfixe tous les nœuds marqués de l'arbre et seulement ceux-là.

Pour qu'il n'y ait pas d'ambiguïté, on fixe comme convention que la profondeur de la racine est 0.

9. Complétez la fonction *retourneProfondeurMaxMarque(Node tree)* pour qu'elle retourne la profondeur du plus profond des nœuds marqués, -1 si aucun nœud n'est marqué.
10. Complétez la fonction *afficheCheminMarques(Node tree)* pour qu'elle affiche les chaînes constituées de l'ensembles des étiquettes trouvées sur le chemin de la racine à chacun des nœuds marqués, encore en n'utilisant que la fonction affiche.