Alexandre TORREGROSSA 17/06/2021

Polytech Annecy - IDU3 – 2020/2021

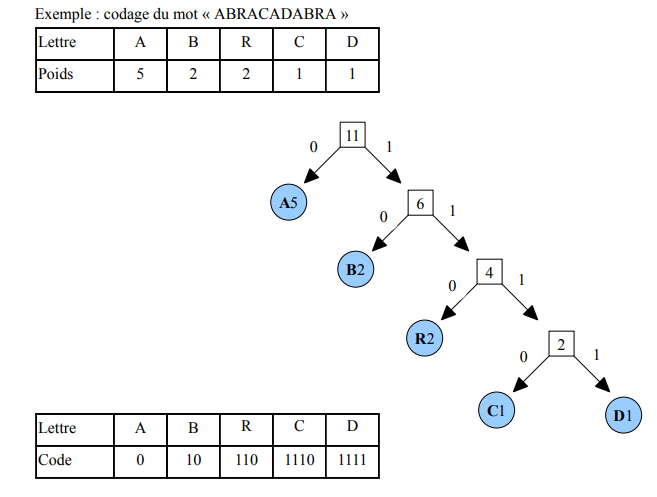
Proj631 – Projet algorithmique

**Rapport de projet**

**Lien GitHub :** https://github.com/garlam123/Codage-Huffman-Info631

1/ Introduction

Dans le cadre de la matière Projet algorithmique, j’ai choisi de travailler pour ce 2e Projet sur la compression de données par codage de Huffman. Le codage de Huffman permet de compresser les données sans perte d’information. Le principe est de repérer les caractères les plus fréquents dans un texte et de leur attribuer des codes binaires courts alors que les caractères les moins fréquents auront des codes plus longs. Pour cela nous devons tout d’abords compter le nombre d’occurrence de chaque caractère pour pouvoir évaluer son ’’poids’’. Puis dans un second temps, on crée un arbre composé de nœuds et de feuille. Les feuilles sont des Tuples représentant chaque caractère ainsi que son poids préalablement déterminé. Puis pour pouvoir coder notre texte par exemple, on parcourt l’arbre. Ainsi en partant de la racine préalablement déterminée on ajoute 0 à notre code binaire si on se déplace vers le fils gauche et 1 si on se déplace vers le fils droit. On peut voir ci-dessous un exemple concret du codage du mot ABRACADABRA.



2/ Structure du développement

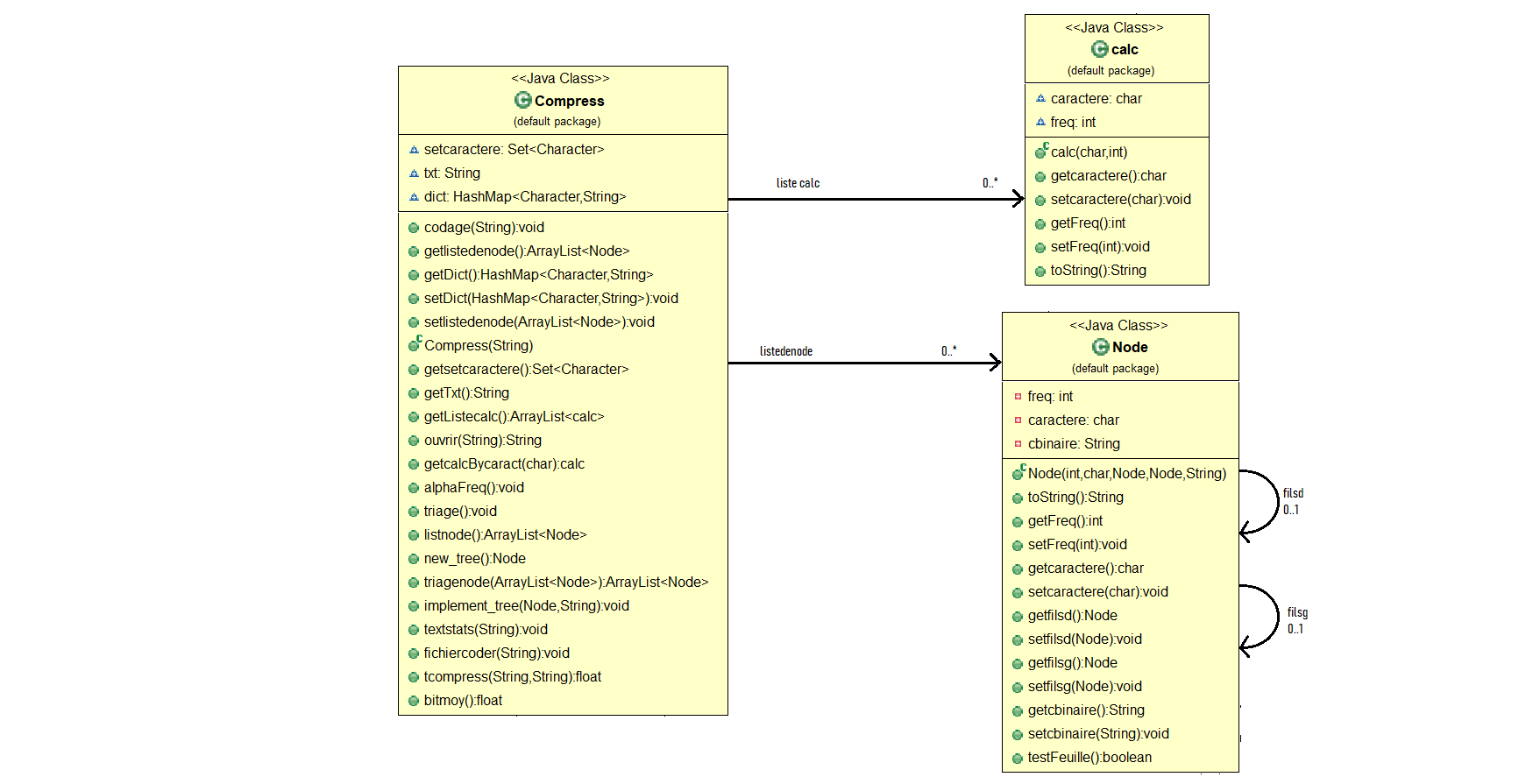
Pour pouvoir répondre à cette problématique, j’ai décidé de créer un programme simple. Pour cela, mon programme est composé de 4 fichiers java.

2.1/ Le diagramme UML

Ce diagramme permet de représenter les différentes classes et les relations entre elles. Il permet de modéliser la problématique avant de procéder au développement du programme.

Ce diagramme représente les trois classes que j’ai instanciées :

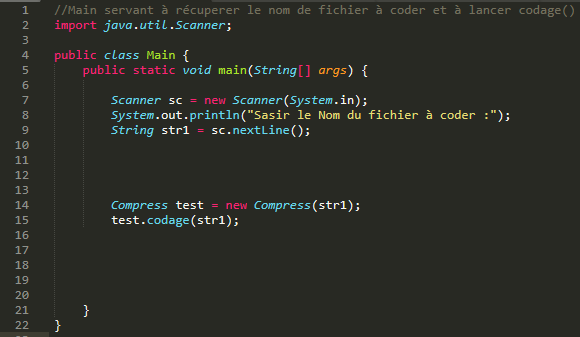
* La classe Compress qui contient la grande majorité des méthodes de mon programme
* La classe calc qui me permet de calculer les occurrences
* La classe Node qui me permet de crée mon arbre



*Diagramme de classes*

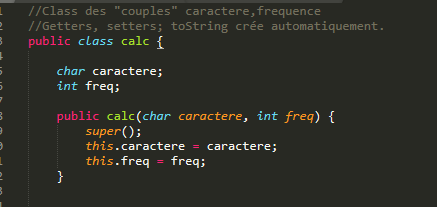
2.2/ Le fichier Main

Ce fichier définit ma classe Main et me permet de « communiquer » avec l’utilisateur. Pour ce faire, j’utilise le package java.util.scanner qui me sert à récupérer le nom du fichier texte à coder. Puis dans un second temps je lance ma méthode codage() sur le fichier indiqué par l’utilisateur.

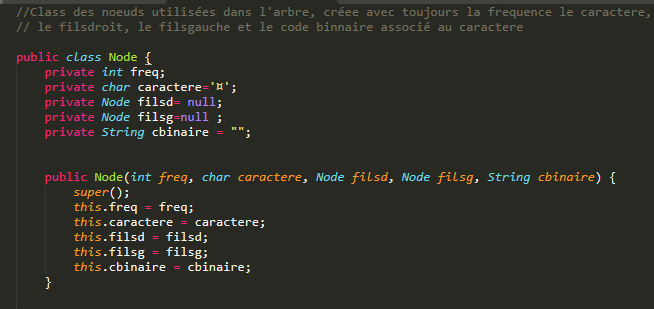


2.3/ Le fichier calc et le fichier node

Mon programme comporte deux fichiers me permettant de déclarer deux de mes trois classes. On a tout d’abords le fichier calc qui définit la classe calc. Cette classe me sert à pouvoir calculer les occurrences de chaque caractère dans mon texte. Ma classe possède donc deux variables, un char (caractère) et un int (entier) qui représente respectivement la lettre et son nombre d’occurrence. De plus j’ai généré grâce à éclipse les getters & setters correspondant à ma méthode calc. La méthode ToString me permet de retourner les informations contenues dans un calc

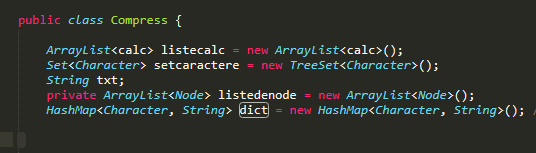


Mon fichier node quant à lui représente ma classe de nœud pour mon futur arbre de codage. Mes nœuds sont caractérisés par la fréquence (int), le caractère associé (char), le fils gauche (node), le fils droit (node) et le code binaire associé au caractère (String). De plus, comme dans ma classe calc, j’ai généré automatiquement les getters & setters de ma méthode Node ainsi que ma méthode ToString me permettant de récupérer les informations d’un nœud, ce qui me sera utile lorsque je construirai mes fichiers de décodage, c’est-à-dire mon fichier liant les caractères à leurs occurrences ainsi qu’à leur code binaire. Je définis aussi une petite méthode me permettant de savoir si le nœud est une feuille ou non. Cette méthode se nomme testFeuille et est définie dans ma classe node.

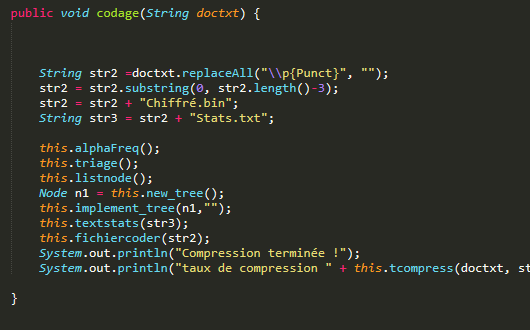


2.4/ Le fichier Compress.java

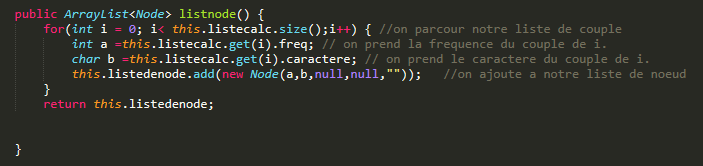
Mon fichier Compress contient ma classe Compress qui représente mon texte, cette classe contient donc toutes les méthodes qui seront effectuées sur le texte pour pouvoir l’encoder. Ma classe Compress est caractérisée par le nom du texte (String), une liste de calc (Tuple) contenant les caractères et leurs fréquences, une liste des caractères dans le texte, une liste des nœuds et une liste associant les caractères à leur code binaire.



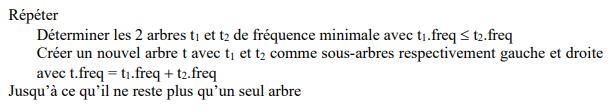
De plus, mon fichier Compress contient aussi toutes les méthodes qui m’ont servi pour pouvoir coder un texte demandé. Dans un premier temps, lorsque le Main appelle la méthode codage, celle-ci va lancer une multitude d’autres méthodes dites principales.



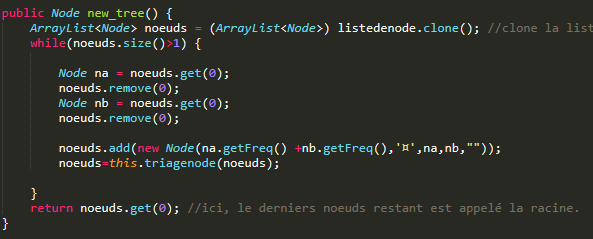
En premier lieu, on cherche à créer une liste des caractères et de leur fréquence associée, pour cela j’ai créé la méthode alphaFreq qui me permet de créer mes calc (Tuple) de chaque caractère. Puis j’utilise une méthode de tri qui me permet d’ordonner cette liste de calc par ordre croissant des fréquences. Puis à l’aide de la méthode listenode, j’initialise tous mes nœuds en prenant chaque calc et en ajoutant les données dans mes nouveaux nœuds. Ayant déjà trié mes calc, ma liste de nœuds créée est classée aussi par ordre croissant en termes de fréquence.



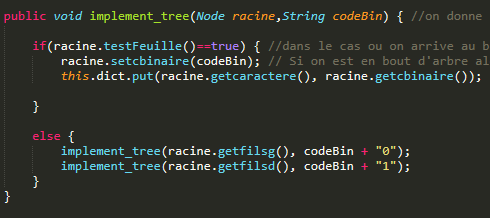
Ayant créé ma liste de nœuds, il me faut maintenant créer l’arbre qui me servira à encoder mon texte. Pour cela j’utilise ma méthode new\_tree. Pour pouvoir coder cette méthode j’ai utilisé les informations disponibles dans le sujet.



En effet, on nous précise comment créer l’arbre, mon programme suis donc la logique ci-dessus en générant des nouveaux nœuds avec les fréquences les plus petites additionnées. De plus, à la fin de cette méthode je retrie ma liste de nœuds fraichement créée dans l’ordre croissant. A la fin de la constitution de mon arbre, je dispose donc de ma racine qui est l’élément le plus présent dans le texte (car classé par ordre croissant) ainsi que ma liste de nœud également classée.



Apres avoir généré mon arbre de codage, je le parcours à l’aide de ma méthode Implement\_tree qui me permet d’associer chaque caractère avec son code binaire donné par mon arbre. C’est une méthode récursive qui va parcourir l’arbre en partant à chaque boucle de la racine. Je stocke ensuite mon code binaire généré ainsi que le caractère correspondant dans ma liste prévue à cet effet.



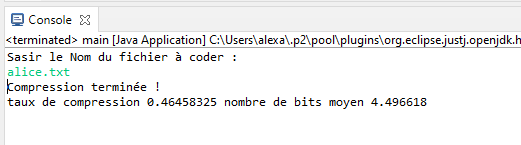
Lorsque le texte a été codé et que toutes les données ont été récoltées, je peux générer les documents qui seront destinés à l’utilisateur. Pour cela j’utilise dans un premier temps ma méthode Textstats qui va me permettre grâce au package FileWritter de pouvoir créer et écrire dans le nouveau fichier. Je génère donc mon premier fichier qui sera un fichier texte contenant les caractères associés à leurs fréquences dans le texte ainsi que leur code binaire utilisé pour coder celui-ci.

Puis dans un second temps, j’utilise de la même façon ma méthode Fichiercoder qui me permet d’écrire mon fichier bin. Pour cela on récupère les codes binaires contenus dans ma variable que nous avons vue ci-dessus. Puis on s’assure que ces codes soient un multiple de 8 pour pouvoir découper par byte, après cela fait on écrit le code obtenu dans un fichier.

Pour finir j’utilise mes méthodes tcompress et bitmoy qui me permettent de calculer mon taux de compression ainsi que mon nombre de byte moyen par caractère.

3/ Rendu Utilisateur

Lorsque l’utilisateur va lancer le Main, le programme demandera dans la console de lui donner le nom du fichier à coder. Puis après avoir validé avec la touche entrée, il faudra patienter quelques secondes en fonction de la longueur du texte pour voir apparaitre sur la console un message de fin de compression ainsi que les données statistiques sur le taux de compression et le nombre moyen de byte par caractères.



Puis lorsque l’on consulte le répertoire contenant les fichiers de notre programme nous voyons que 2 fichiers ont été générés.



*Voici un exemple pour le fichier alice.txt*

4/ En conclusion

Ce projet m’a permis de m’améliorer dans la compréhension et l’utilisation du langage java appliqué à des programmes simple. La compréhension de l’algorithme d’Huffman n’a pas été simple mais la retranscription de celui-ci en code m’a permis de mieux le comprendre. Si j’avais pu disposer de plus de temps j’aurais aimé pouvoir intégrer une interface graphique développée par exemple avec le package JFrame qui aurait pu me permettre de simplifier l’utilisation de mon programme et à fortiori de le rendre accessible à un public plus large.

De plus j’aurais aimé pouvoir rajouter une fonctionnalité permettant de pouvoir sélectionner son document sur le disque de l’ordinateur, pour éviter d’avoir à saisir manuellement le nom de fichier en ligne de commande.

Durant ce projet j’ai mieux compris le langage java et sa structuration ainsi que la façon de concevoir et développer un programme.