### 3.4. Archive Java - Partie 2 : Ajout

L’option *2. Ajout* du menu, propose à l’utilisateur d’insérer de nouvelles informations au bloc de donnés.

Tout d’abord, il est demandé à l’utilisateur de renseigner un nom pour l’information à stocker, puis d’écrire cette information en toutes lettres. Ensuite, le programme doit effectuer les tests suivants :

* S’assurer que le nom de l’information n’existe pas déjà et que son nom ne soit pas ’0’
* S’assurer que l’id de l’information est supérieure à l’id le plus élevé déjà présent
* S’assurer qu’il reste suffisamment de place dans le bloc de données partie ’table’
* S’assurer qu’il reste suffisamment de place dans le bloc de données partie ’données’

Si un de ces points n’est pas valide, le programme doit le signaler à l’utilisateur et revenir au menu. Si toutes les conditions sont validées, la donnée est sauvegardée. L’écriture de ces données doit être reflétée à la fois dans les données stockées dans le programme mais également dans le fichier qui sert de source d’informations.

1. Les vérifications :

Pour réaliser la partie *ajout*, écrire une méthode par critère de vérification (donc 4) qui seront **privées** et retourneront un **booléen**.

Ces quatre méthodes seront utilisées dans une cinquième, qui aura pour paramètre l’id de l’information et sa valeur remplie par l’utilisateur. Cette méthode retournera *true* si tous les critères sont respectés, *false* dans le cas contraire.

Ecrire une méthode permettant d’écrire le contenu des listes *data* et *table* dans un fichier texte. Le but de cette méthode sera de faire l’inverse de la méthode *readCassette()*.

Ecrire une méthode permettant d’encoder l’information renseignée par l’utilisateur en ASCII et de la mettre dans la liste *data*.

Ecrire une méthode permettant d’écrire l’id et la taille de l’information dans la liste *table*.

Conseils :

* Les méthodes qui dialogueront avec la cassette seront écrites dans la classe *Cassette*.

### 3.5. Archive Java - Partie 3 : Suppression

La dernière option du menu permet de supprimer un élément du bloc de données.

Contrairement à la procédure d’écriture, l’utilisateur renseigne un nom d’information, mais aucun autre élément. Le programme vérifie alors que l’information existe effectivement. Si elle n’existe pas, l’utilisateur est informé et est reconduit au menu principal.

Si l’information existe bien, elle est présentée à l’utilisateur pour vérification (nom et contenu) qui doit valider la suppression. La suppression est une opération plus complexe que l’addition : il faut effectuer une défragmentation après retrait d’une information, c’est à dire s’assurer que les informations restantes les unes à la suite des autres afin de ne pas perdre de place.

Créer une méthode privée retournant un booléen qui permet de vérifier si l’id renseignée par l’utilisateur existe.

Créer une méthode qui permet de supprimer une information de la liste *data* en fonction de son id. Cette méthode remplace les charactères de l’information par des « 0 ».

Créer une méthode qui permet de défragmenter la liste *data.* Cela signifie que, si il existe des zéros entre des données, ces zéros sont supprimés, et les données décalées.

Exemple :

Avant :

1,2,3,0,0,0,4,5,6

Après :

1,2,3,4,5,6,0,0,0

Pour ce faire, il faudra donc décaler les informations restantes de la cassette pour combler la place nouvellement libérée.

Exemple de suppression de l’info "14" :

Avant :

104,101,108,108,111,102,

111,110,99,116,105,111,

110,110,101,108,104,101,

105,53,105,116,105,0

3,5,14,11,1,7

Après :

104,101,108,108,111,104,

101,105,53,105,116,105,

0,0,0,0,0,0,

0,0,0,0,0,0,

3,5,1,7,0,0

Tout comme pour l’écriture, cette opération doit être effectuée dans le programme puis dans le fichier (il est recommandé d’envisager de réécrire le fichier entier après avoir effectué la procédure dans le programme !)