Exercice 22 - Prise de Notes

Remarque : Le but de ce TD est de se familiariser avec les notions d'association, d'agrégation et de composition entre classes et de leurs conséquences au niveau de l'implémentation des classes.

On souhaite développer une application destinée à **éditer et gérer un ensemble de *notes textuelles*** que l'on appelle *article*. Un article peut par exemple correspondre au compte-rendu d'une réunion ou à des notes prises lors d'une séance de cours. Un article est caractérisé par un titre et un texte.

D'un point de vue implémentation, un article correspond à un objet instance d'une classe `Article` permettant de le manipuler.

La classe `Article` comporte 3 attributs de type `string` désignant respectivement l'identificateur, le titre et le texte de l'article. L'identificateur permet de distinguer de manière unique un objet `Article`.
Les méthodes `getId()`, `getTitle()`, `getText()`, `setTitle()` et `setText()` permettent d'interagir avec les instances de cette classe. L'unique constructeur de cette classe a 3 paramètres de type `const string&` qui permettent d'initialiser les attributs d'un objet.

Dans l'application, l'ensemble des objets `Article` est géré par un module appelé `NotesManager` qui est responsable de leur création (et destruction) et de leur sauvegarde. La classe possède une méthode `getNewArticle` qui permet de créer un nouvel article dont l'identificateur (qui n'existe a priori pas encore) est transmis en argument. Le titre et le texte d'un nouvel article sont initialement vides. La méthode `getArticle` permet d'obtenir un objet `Article` correspondant à un identificateur déjà existant. Notons qu'un objet `NotesManager` a la responsabilité des objets `Article` qu'il crée (création/destruction).
La méthode `load` prenant un paramètre `f` de type `string` permet de charger un fichier contenant des notes enregistrées lors d'une session précédente. Ce paramètre affecte l'attribut `filename` de la classe (initialement égal à `tmp.dat` au moment de la création d'un objet `NoteManager`). La méthode `save` permet de sauvegarder l'ensemble des notes dans le fichier `filename`. Cette méthode est automatiquement appelée lors de la destruction d'un objet `NotesManager`

L'application dispose aussi d'un module `TagsManager` permettant de gérer des tags sur les articles. Un tag est un objet de la classe `Tag` qui comporte deux attributs : un attribut `name` de type `string` qui sert à stocker le texte du tag et un attribut `article` de type `const Article*` qui pointe sur l'objet `Article` auquel est associé une instance de `Tag`. La classe dispose aussi de deux accesseurs `getName` et `getArticle` permettant de lire la valeur de ces attributs. La classe dispose d'un constructeur permettant d'initialiser un objet avec des valeurs transmises en arguments. La classe `TagsManager` comporte deux méthodes `addTag` et `removeTag` prenant chacune un argument de type `const string&` représentant le nom d'un tag, et une référence sur un objet `Article`. Ces méthodes permettent respectivement d'associer ou de désassocier un tag à un article. Notons qu'un même article peut être référencé par 0, 1 ou plusieurs tags.

Remarque : Les méthodes `load` et `save` ne sont pas à développer dans le cadre de cet exercice.

Préparation : Créer un projet vide et ajouter trois fichiers `notes.h`, `notes.cpp` et `main.cpp`. Définir la fonction principale `main` dans le fichier `main.cpp`. S'assurer que le projet compile correctement. Dans cet exercice, on tâchera de mener une approche "compilation séparée". Au fur et à mesure de l'exercice, on pourra compléter la fonction principale en utilisant les éléments créés. Les situations exceptionnelles seront gérées en utilisant la classe d'exception suivante (à recopier dans le fichier `notes.h`) :

```
#include<string>
class NotesException{
public:
 NotesException(const string& message):info(message){}
 string getInfo() const { return info; }
private:
 string info;
};
```

notes.h

Question 1

Identifier les différentes entités du monde décrit ci-dessus. Identifier les associations qui existent entre ces classes. Quel type de lien existe t-il entre un objet `NotesManager` et les objets `Article` qu'il crée et auxquels il donne accès ? Quel type de lien existe t-il entre un objet `Tag` et l'objet `Article` auxquel il est associé ? Établir un modèle UML où apparaissent les différentes classes utilisées dans l'application.

Question 2

Définir la classe `Article` ainsi que l'ensemble de ses méthodes. Quel est l'intérêt d'utiliser des références **`const`** pour les paramètres du constructeur ? La classe `Article` nécessite t-elle (a priori) un destructeur, un constructeur de recopie et/ou un opérateur d'affectation ? Expliquer. Définir ces méthodes seulement si nécessaire. Surcharger l'opérateur `<<` de manière à pouvoir écrire un objet `Article` sur un flux `ostream`.

Question 3

Est-il possible de définir un tableau (alloué dynamiquement ou non) d'objets `Article` ? Expliquer. Est-il possible de créer un tableau (alloué dynamiquement ou non) de pointeurs d'objet `Article` ? Expliquer.

Question 4

Soit la classe `NotesManager` dont voici une définition partielle :

```
class NotesManager {
 Article** articles;
 unsigned int nbArticles;
 unsigned int nbMaxArticles;
 void addArticle(Article* a);
 string filename;
public:
 NotesManager();
 Article& getNewArticle(const string& id);
 Article& getArticle(const string& id);
};
```

notes.h

L'attribut `articles` est un pointeur vers un tableau de pointeurs d'objets `Article` qui est alloué dynamiquement. La méthode `getNewArticle` permet de créer un nouvel article dont l'identificateur (qui n'existe a priori pas encore) est transmis en argument. Le titre et le texte de cet article sont initialement vides. Si un client essaye de créer un article dont l'identificateur est déjà présent dans l'objet `NotesManager` qui appelle la méthode, une exception est déclenchée. Pour créer un nouvel objet `Article`, la méthode `getNewArticle` *alloue dynamiquement* un objet `Article`. L'adresse de cet objet est alors sauvegardée en appelant la méthode privée `addArticle`. Dans la méthode `addArticle`, l'adresse est alors sauvegardée dans un tableau de pointeurs d'objets `Article`. L'adresse de ce tableau est stockée dans l'attribut `articles` de type `Article**`. L'attribut `nbArticles` représente le nombre d'adresses sauvegardées dans ce tableau. L'attribut `nbMaxArticles` représente le nombre maximum d'adresses qui peut être sauvegardé avant un agrandissement du tableau (*c.-à-d.* la taille du tableau pointé par `articles`). La méthode `addArticle` gère les éventuels besoins en agrandissement du tableau. La méthode `getArticle` permet d'obtenir un objet `Article` correspondant à un identificateur déjà existant. Si un client essaye d'accéder à un article dont l'identificateur n'est pas encore présent dans l'objet `NotesManager` qui appelle la méthode, une exception est déclenchée. La classe `NotesManager` possède un unique constructeur sans argument. Initialement, un objet `NotesManager` ne possède aucun article et l'attribut `filename` est égal à `tmp.dat`.
Définir les méthodes `NotesManager()`, `getNewArticle()`, `addArticle()` et `getArticle()`.

Question 5

La classe `NotesManager` nécessite t-elle le développement d'un destructeur ? Pourquoi ? Si oui, implémenter ce destructeur (on ne tiendra pas compte de la partie "gestion de fichier" dans cette question).

Question 6

Dans l'hypothèse où la duplication d'un objet `NotesManager` est autorisée, la classe `NotesManager` nécessite t-elle le développement d'un constructeur de recopie et/ou d'un opérateur d'affectation ? Si oui, implémenter ces méthodes. Définir la classe `Tag` ainsi que l'ensemble de ses méthodes.

Question 7

Dans l'hypothèse où la duplication d'un objet `Tag` est autorisée, la classe `Tag` nécessite t-elle le développement d'un constructeur de recopie et/ou d'un opérateur d'affectation ? Si oui, implémenter ces méthodes.

Question 8

Définir la classe `TagsManager` ainsi que l'ensemble de ses méthodes. En pratique, la classe possède aussi un tableau (dont la capacité sera étendue selon les besoins) pour stocker ses différents tags. La classe `TagsManager` nécessite t-elle le développement d'un destructeur ? Pourquoi ? Si oui, implémenter ce destructeur. Dans l'hypothèse où la duplication d'un objet `TagsManager` est autorisée, la classe `TagsManager` nécessite t-elle le développement d'un constructeur de recopie et/ou d'un opérateur d'affectation ? Si oui, implémenter ces méthodes.

Exercice 23 - Gestion de fichier *-Exercice d'approfondissement à faire à la maison-*

Dans cet exercice, on veut mettre en place la partie "*gestion de fichier*" de la classe `NotesManager` de l'Exercice 22. Cet exercice, qui n'est pas à faire dans le cadre de la séance de TD, a pour but de se familiariser avec la gestion des fichiers en C++.

Question 1

Définir la méthode `load` de la classe `NotesManager`.

Question 2

Dans la classe `NotesManager`, définir la méthodes `save` qui permet d'enregistrer les informations des objets `Article` de l'objet `NotesManager`. Apporter les modifications nécessaires afin de mettre en oeuvre la sauvegarde des articles lors de la destruction d'un objet `NotesManager`.