PG208, Exercice préparatoire

Bertrand LE GAL[*], Jérémie CRENNE[†], Guillaume DELBERGUE[‡] et Thibaud TONNELLIER[§]

Filière électronique 2^{eme} année - Année universitaire 2014-2015

Avant d'aborder le projet de conception objet, une séance de mise en pratique des notions vues en cours est indispensable. Afin d'optimiser le déroulement de la séance, il vous est demandé de préparer à l'avance l'exercice. Cette préparation **préliminaire** sera évaluée par votre enseignant.

Présentation du sujet de l'étude

Afin d'illustrer au mieux les différentes notions vues en cours, nous avons décidé de vous faire développer une classe permettant de mémoriser les informations relatives à une personne. Cette classe nommée Personne devra permettre de décrire un individu sachant que l'on souhaite avoir autant d'informations que dans la phrase suivante :

$\Longrightarrow$ Mr John Doe est né en 1969 (43 ans), il est célibataire.

Travail préparatoire

Dans ce TP vous allez être à la fois concepteur et utilisateur de la classe Personne. Votre première tache consiste à définir la structure de la classe (attributs/méthodes). Pour cela vous devrez identifier les informations à mémoriser à partir de l'exemple précédent. Durant cette étape, vous prendrez soin d'utiliser à bon escient les niveaux de visibilité offerts par le langage C++. De plus, afin d'accéder aux attributs de la classe, vous définirez les accesseurs et mutateurs qui vous semblent nécessaires. Une fois ce travail terminé, vous n'aurez plus qu'à développer le code associé aux méthodes.

Enfin, vous écrirez un programme (main) qui permet de tester le bon fonctionnement de votre classe. Pour cela vous créerez deux Personnes et afficherez leurs informations.

[*]bertrand.legal(at)ims-bordeaux.fr
[†]jeremie.crenne(at)ims-bordeaux.fr
[‡]guillaume.delbergue(at)ims-bordeaux.fr
[§]thibaud.tonnellier(at)ims-bordeaux.fr

Remarque 1 : afin de vous simplifier la vie, vous pourrez utiliser toutes les classes disponibles dans la STL.

Remarque 2 : votre classe devra posséder une méthode publique nommée $void\ afficheInformations()$ qui aura pour rôle d'afficher les informations à l'écran (tel que cela est présenté dans l'énoncé).

Remarque 3 : votre classe possédera un constructeur dont le nombre de paramètres sera suffisant pour permettre la création d'une Personne à partir de toutes les informations nécessaires.

Travail demandé en séance

Etape 1

Dans un premier temps utiliser l ?outil QtCreator afin de coder la classe et le programme main que vous avez imaginé dans le travail préparatoire. Vérifier le fonctionnement de votre code.

Etape 2

Nous souhaitons maintenant pouvoir créer des personnes à partir d'informations contenues dans des fichiers textuels. Ajouter à votre classe deux méthodes nommées $bool\ loadFromFile(const\ char*filename)$ et $bool\ saveToFile(const\ char*filename)$.

De plus, ajouter un constructeur dans votre classe afin de permettre la création d ?une Personne à partir du nom du fichier qui contient les informations. Vous testerez votre programme en enregistrant les 2 personnes que vous avez créées précédemment « en dur ».

Remarque 4 : pour vous simplifier la vie, vous pourrez mémoriser une seule information par ligne.

Etape 3

Nous souhaitons pouvoir gérer simplement un ensemble de personnes. Les informations relatives à ces Personnes seront mémorisées dans des fichiers sauvegardés sur le disque dur. Le nom des fichiers sera standardisé $personne\_XX.txt$ avec XX un nombre entier sur 2 digits. Modifiez votre programme afin dans un premier temps de charger tous les fichiers. Puis d'afficher les informations relatives aux personnes. Enfin de faire le proprement le ménage.

Etape 4

Nous souhaitons rajouter des fonctionnalités à la classe Personne. Toutefois nous ne souhaitons pas alourdir l'occupation mémoire de cette dernière. Afin de répondre à ce besoin, vous allez devoir définir

une nouvelle classe (ExtPersonne).

Nous souhaitons que cette nouvelle classe modifie l'affichage des informations relative à une personne de la manière suivante :

$\Longrightarrow$ [ACTIF : J.DOE-69] Mr John Doe est né en 1969 (43 ans), il est célibataire.

La clef a insérer au début de l'affichage des informations est constituée de 4 champs : le statut de la personne, l ?initiale du prénom, le nom de famille en majuscule ainsi que les deux derniers chiffres de l'année de naissance. Le statut d'une personne sera sélection en fonction de l'age de la personne :

— $0 \leq age < 12 \Rightarrow ENFANT$
— $12 \leq age < 18 \Rightarrow ADOLESCENT$
— $18 \leq age < 65 \Rightarrow ACTIF$
— $65 \leq age < 99 \Rightarrow RETRAITE$

Ecrivez cette nouvelle classe :

— sans utiliser d'attributs supplémentaires,
— en prenant soin de minimiser le nombre de lignes de code que vous écrivez.

Modifiez votre main pour tester cette nouvelle classe.

Etape 5

Normalement dans la question précédente, vous avez du remplacer toutes les utilisations de la classe Personne par des utilisations de la classe ExtPersonne. Ce choix est discutable, en effet, des ExtPersonnes sont aussi des Personnes (par construction). Modifiez votre programme en conséquence & corriger le problème qui vient d'apparaître...

Exemples de codes sources utiles

Le premier exemple de code vous montre comment il est possible de lire l'ensemble d'un fichier de type textuel en C++. Cet exemple lit les informations ligne par ligne et les affiche à l'écran.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

void main ()
{
        string   STRING;
        ifstream infile;
        infile.open ("names.txt");
        while(!infile.eof)
        {
                getline(infile,STRING);
                cout << STRING;
        }
        infile.close();
}
```

Dans ce second exemple, le code C++ présenté permet de lire une chaîne de caractère tapée par l'utilisateur dans la console. La lecture de la chaîne de caractères est réalisée lorsque l'utilisateur appuie sur la touche *return*.

```
void lecture_clavier ()
{
        string chaine;
        cout << "Entrez une phrase : " << endl;
        getline(cin, chaine);
        cout << "La phrase est : " << chaine << endl;
}
```

Ce troisième exemple illustre les méthodes disponibles dans la classe string. Ces dernières vous seront sûrement utiles lors de votre développement.

```
void test_string( ){
        string c = "Voici une phrase !";
        cout << "Taille de la chaine      : " << c.length() << endl;
```

```
        cout << "Pos. du premier espace ... : " << c.find (" ") << endl;
        cout << "Pos. du dernier espace ... : " << c.rfind(" ") << endl;
        cout << "Pos. du mot 'une' ........ : " << c.find("une") << endl;
        cout << "Premier mot .............. : " << c.substr(0, c.find (" ")) << endl;
        cout << "Comparaison (true) ....... : " << c.compare( c ) << endl;
        cout << "Comparaison (false) ...... : " << c.compare( "TOTO" ) << endl;
}
```

Ce quatrième exemple vous permet de découper une chaîne de caractères en sous parties en fonction d'un séparateur (ici le caractère "=").

```
void decoupe_string ()
{
        string c;
        cout << "Entrez une phrase : " << endl;
        getline(cin, c);
        int taille = c.size();
        if( taille != 0 ){
                int pos = c.find('=');
                cout << "Ligne lue : *" << c << "*" << endl;
                cout << "Position: " << pos << endl;
                cout << "Mot 1 : " << c.substr(0, pos) << endl;
                cout << "Mot 2 : " << c.substr(pos+1, taille -(pos+1)) << endl;
        }
}
```

Ce cinquième exemple vous indique la procédure à suivre pour récupérer un pointeur sur la chaine de caractère contenue dans un objet de type string.

```
void pointeur_char ()
{
        string   STRING;
        cout << "Votre message = ";
        getline(cin, STRING);
        char* chaine = STRING.c_str();
        cout << "INTEGER = " << atoi(chaine);
}
```

Ce dernier exemple vous montre une méthode efficace pour convertir les données contenues dans une chaîne de caractères en majuscules. Cette approche nécessite l'inclusion de la bibliothèque $< algorithm >$ dans votre programme.

```
string str_to_upper(string str)
{
        string str_up = str;
        std::transform(str_up.begin(), str_up.end(), str_up.begin(), ::toupper);
        cout << str << endl;
        return str_up;
}
```