TP Western C++ n°1

© 2013 tv <tvaira@free.fr> - v.1.0

Sommaire

Histoire n°1 : Tous humains ... **2**

Objectifs . . . 2

Initiation à la programmation orientée objet **2**

Notion d'objets . . . 2

Notion de classe . . . 2

Notion de visibilité . . . 2

Notion d'encapsulation . . . 2

Exemple détaillé : les humains . . . 3

Travail demandé **6**

Les objectifs de ce tp sont de découvrir la programmation orientée objet en C++.
On désire réaliser un programme C++ permettant d'écrire facilement des histoires de Western.
Dans nos histoires, nous aurons des brigands, des cowboys, des shérifs, des barmen et des dames en détresses ... (à partir d'une idée de Laurent Provot)

Histoire n°1 : Tous humains ...

Objectifs

On désire réaliser un programme orienté objet en C++ qui **racontera une histoire dans laquelle un humain arrive, se présente et boit un coup**.

```
(Lucky Luke) -- Bonjour, je suis Lucky Luke et j'aime le coca-cola

(Lucky Luke) -- Ah ! un bon verre de coca-cola ! GLOUPS !
```

Initiation à la programmation orientée objet

Notion d'objets

La programmation orientée objet consiste à **définir des objets logiciels et à les faire interagir entre eux**.

Concrètement, un **objet** est une **structure de données** (**ses attributs** c.-à-d. des variables) qui définit son **état** et une **interface** (**ses méthodes** c.-à-d. des fonctions) qui définit son **comportement**.

Un objet est créé à partir d'un **modèle** appelé **classe**. Chaque objet créé à partir de cette classe est une **instance** de la classe en question.

Notion de classe

Une classe **déclare des propriétés communes** à un ensemble d'objets.

Une classe représentera donc une **catégorie d'objets**.

Elle apparaît comme un **type** ou un *moule* à partir duquel il sera possible de créer des objets.

Notion de visibilité

Le C++ permet de préciser le **type d'accès des membres** (attributs et méthodes) d'un objet. Cette opération s'effectue au sein des classes de ces objets :

- **public** : les membres publics peuvent être utilisés dans et par n'importe quelle partie du programme.
- **privé** (*private*) : les membres privés d'une classe ne sont accessibles que par les objets de cette classe et non par ceux d'une autre classe.

Notion d'encapsulation

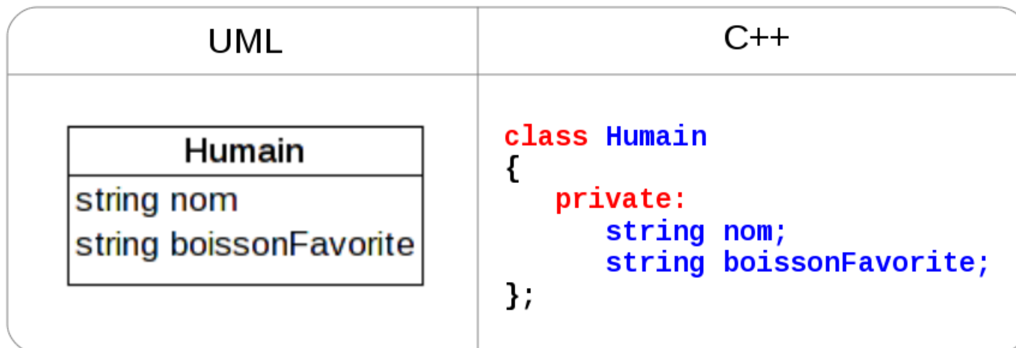
L'encapsulation est l'idée de **protéger les variables contenues dans un objet et de ne proposer que des méthodes pour les manipuler**.

L'objet est ainsi vu de l'extérieur comme une "boîte noire" possédant certaines propriétés et ayant un comportement spécifié.

Exemple détaillé : les humains

Les intervenants de nos histoires sont tous des humains. Notre **humain** est caractérisé par son **nom** et sa **boisson favorite**. La boisson favorite d'un humain est, **par défaut**, de l'eau.

On modélise donc une **classe `Humain`** :



Toutes les variables (attributs) de la classe `Humain` seront privées.

Pour créer des objets à partir de cette classe, il faudra ... **un constructeur**. On le **déclare** de la manière suivante :

```
class Humain
{
   public:
      // Constructeur :
      Humain(const string nom="inconnu", const string boissonFavorite="eau"); // de l'eau par défaut

   private:
      string nom;
      string boissonFavorite;
};
```

humain.h

Le constructeur porte toujours le même nom que la classe.

Il faut maintenant **définir** ce constructeur afin qu'il **initialise toutes les variables de l'objet au moment de sa création** :

```
#include "humain.h"

// Constructeur
Humain::Humain(const string nom/*="inconnu"*/, const string boissonFavorite/*="eau"*/)
   : nom(nom), boissonFavorite(boissonFavorite)
{
}
```

humain.cpp

Le constructeur est appelé automatiquement au moment de la création d'un objet.

 © 2013 tv <tvaira@free.fr>

On pourra alors créer nos propres humains :

UML	C++
lucky:Humain boissonFavorite = coca-cola nom = Luck Luke	`Humain lucky("Lucky Luke", "coca-cola");`
joe:Humain boissonFavorite = whisky nom = Joe Dalton	`Humain *joe = new Humain("Joe", "whisky");`

Les objets `lucky` et `joe` sont des instances de la classe `Humain`. Un objet possède sa propre existence et un état qui lui est spécifique (c.-à-d. les valeurs de ses attributs).

Un humain pourra **parler**. On aura donc une **méthode** `parle(texte)` qui affiche :

`(nom de l'humain) -- texte`

UML	C++
Humain string nom string boissonFavorite Humain(const string nom = "", const string boissonFavorite = "eau") void parle(const string texte)	(voir code ci-dessous)

```
class Humain
{
  private:
    string nom;
    string boissonFavorite;
  public:
    Humain(const string nom="", const
string boissonFavorite="eau");
    void parle(const string texte);
};

void Humain::parle(const string texte)
{
    cout << "(" << nom << ") -- " <<
texte << endl;
}
```

On utilisera cette méthode dès que l'on voudra faire dire quelque chose par un humain :

```
Humain lucky("Lucky Luke", "coca-cola");

lucky.parle("Je parle !"); // appel de la méthode parle()
```

Ce qui donnera à l'exécution :

```
(Lucky Luke) -- Je parle !
```

Une méthode publique est un service rendu à l'utilisateur de l'objet.

 © 2013 tv <tvaira@free.fr>

Toutes les variables de la classe `Humain` étant **privées par respect du principe d'encapsulation**, on veut néanmoins pouvoir connaître sa boisson favorite et pouvoir modifier cette dernière.

Il faut donc créer deux **méthodes publiques** pour **accéder** à l'**attribut** `boissonFavorite` :

UML	C++
Humain string nom string boissonFavorite Humain(const string nom = "", const string boissonFavorite = "eau") string getBoissonFavorite() void setBoissonFavorite(const string nouvelleBoissonFavorite) void parle(const string texte)	voir le code ci-dessous

```cpp
class Humain
{
  private:
   string nom;
   string boissonFavorite;
  public:
   Humain(const string nom="", const string
boissonFavorite="eau");
   string getBoissonFavorite() const;
   void setBoissonFavorite(const string
nouvelleBoissonFavorite);
   void parle(const string texte)
};

// Assesseur get
string Humain::getBoissonFavorite() const
{
  return boissonFavorite;
}

// Manipulateur set
void Humain::setBoissonFavorite(const string
nouvelleBoissonFavorite)
{
  if(!nouvelleBoissonFavorite.empty())
   BoissonFavorite = nouvelleBoissonFavorite;
}
```

La méthode publique `getBoissonFavorite()` est un accesseur (*get*) et `setBoissonFavorite()` est un manipulateur (*set*) de l'attribut `boissonFavorite`. L'utilisateur de cet objet ne pourra pas lire ou modifier directement une propriété sans passer par un accesseur ou un manipulateur.

 © 2013 tv <tvaira@free.fr>

Travail demandé

Un **humain** pourra également **se présenter** (il dit bonjour, son nom, et indique sa boisson favorite), et **boire** (il dira « Ah ! un bon verre de (sa boisson favorite) ! GLOUPS ! »). On veut aussi pouvoir connaître le **nom** d'un **humain** mais il n'est pas possible de le modifier.

Question 1. Compléter les fichiers `humain.cpp` et `humain.h` fournis afin d'assurer l'exécution du programme de test ci-dessous.

```cpp
#include <iostream>

using namespace std;

#include "humain.h"

/* TP Western C++ n˚1 : tous humains ... */

int main()
{
   Humain lucky("Lucky Luke", "coca-cola");

   cout << "Une histoire sur " << lucky.getNom() << endl;

   lucky.sePresente();

   lucky.boit();


   Humain *joe = new Humain("Joe");

   cout << "Une histoire sur " << joe->getNom() << endl ;

   joe->setBoissonFavorite("whisky");

   joe->sePresente();

   joe->boit();

   return 0;
}
```

histoire-1.cpp

```
Une histoire sur Lucky Luke
(Lucky Luke) -- Bonjour, je suis Lucky Luke et j'aime le coca-cola
(Lucky Luke) -- Ah ! un bon verre de coca-cola ! GLOUPS !
Une histoire sur Joe
(Joe) -- Bonjour, je suis Joe et j'aime le whisky
(Joe) -- Ah ! un bon verre de whisky ! GLOUPS !
```

 © 2013 tv <tvaira@free.fr>