Extrait du référentiel : BTS CIEL option A (Informatique et Réseaux)		Niveau(x)
C08 CODER	Langages de développement, de description, de création d'API et les IDE associés	4

Objectifs du cours :

- Programmation orientée objet :
  - principe
  - notion de classe
  - notion d'objets
  - notion de visibilité
  - notion de messages
  - notion de surcharge

PROGRAMMATION ORIENTÉE OBJET

PRINCIPE

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

Dans ce cours, le ou les codes présents sont donnés sans modularité.

NOTION DE CLASSE

Une classe permet de regrouper dans une même entité des données et des fonctions membres (appelées aussi **méthodes**) permettant de manipuler ces données. La classe est la notion de base de la programmation orientée objet. Il s'agit en fait d'une évolution de la notion de structure, qui apporte de nouvelles notions orientées objet absolument fondamentales. Un **objet** est un élément d'une certaine classe : on parle de **l'instance d'une classe**.

NOTION D'OBJETS

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

Un objet posséde une **identité** qui permet de distinguer un objet d'un autre objet (son nom, une adresse mémoire).

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.

Il existe aussi un accès protégé (protected) en C++ que l'on abordera plus tard avec la notion d'héritage.

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**.

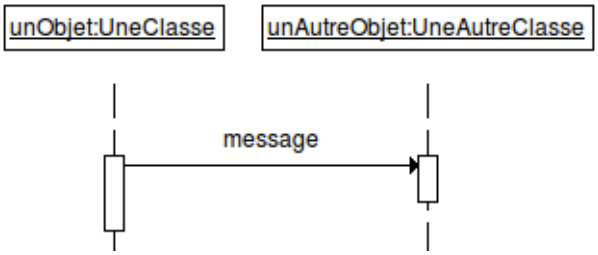
En respectant ce principe, toutes les variables (attributs) d'une classe seront donc privées.

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

NOTION DE MESSAGES

Un objet est une structure de données encapsulées qui répond à un **ensemble de messages**. Cette **structure de données (ses attributs) définit son état** tandis que l'**ensemble des messages (ses méthodes) décrit son comportement**.

L'ensemble des messages forme ce que l'on appelle l'**interface de l'objet**. Les objets interagissent entre eux en **s'échangeant des messages**.



La réponse à la réception d'un message par un objet est appelée **une méthode**. Une **méthode est donc la mise en oeuvre du message** : elle décrit la réponse qui doit être donnée au message.

NOTION DE SURCHARGE

Il est généralement conseillé d'attribuer des noms distincts à des fonctions différentes. Cependant, lorsque des fonctions effectuent la même tache sur des objets de type différent, il peut être pratique de leur attribuer des noms identiques. L'utilisation d'un même nom pour des fonctions (ou méthodes) s'appliquant à des types différents est nommée **surcharge**.

La surcharge n'est pas possible en C mais seulement en C++ !

```
#include<iostream>

using namespace std;

void print(int);
void print(float);

int main ()
{
 int n = 2;
 float x = 2.5;
 print(n);
 print(x);
 return 0;

}
void print(int a)
{
 cout << "je suis print et j affiche un int : " << a << endl;
}
void print(float a)
{
 cout << "je suis print et j affiche un float : " << a << endl;
}
```

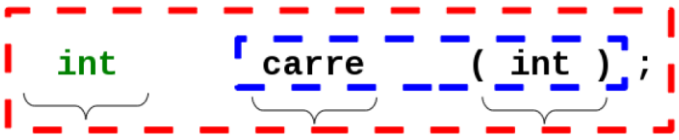
Surcharge de fonctions en C++

Résultat :

```
je suis print et j affiche un int : 2
je suis print et j affiche un float : 2.5
```

Une **surcharge** permettra donc **d'utiliser plusieurs méthodes qui portent le même nom au sein d'une même classe avec une signature différente**.

Il existe une forme recommandée dite **prototype** qui comprend sa **signature** et son **type de retour** :



type de retour
Si on ne veut pas retourner de résultat, on spécifie **void** (qui signifie « rien »)

le nom de la fonction

liste de paramètres
Ses éléments se nomment des paramètres. On indique leur type mais leur nom est facultatif dans une déclaration. Et la liste peut être **vide** ().

Le type de retour ne fait pas partie de la signature d'une fonction (ou d'une méthode). La surcharge ne peut donc s'appliquer que sur le type et/ou le nombre des paramètres.