# <C++>

les classes

Dorian.H Mekni

17 | APRIL | 2020

# class | A

- Un modele de fabrication qui illustre un concept, une idée, ou une category
- Le protocole préfère que l’on crée un fichier par classe
- Par exemple le genre humain pour être une classe disséminée en groupe ethniques, couleurs, sexe, religions, orientation politiques et conditions économiques et sociales ainsi que la position salariale occupée
- Cela est très proche de la programmation modulaire déjà vu en C avec les fonctions mais dans laquelle cette fois-ci on y ajoute des classes.
- La structure est une sorte de classe en C++

# class | B

- La classe est instanciée afin de pourvoir créer des objets.
- La classe aura ses propriétés qui formeront les attributs
- Dans le genre humain, ce sera pour chaque personne ce qui le constitue , ou bien ses possessions.
- Dans un deuxième temps nous aurons son comportement, et donc ses habilités à faire, produire.

C++

# class

/*

La portée c'est ce qui nous donne accès à une classe ou non.
Pour préciser que la construction aura un accès public,
nous devons le notifier dans les paramètres de notre classe.
Lorsque nous codons du code procédural ou impératif,
nous avons des fonctions et des variables.
Dans la programmation objet le nom d'une fonction liée à une classe s'appelle une méthode.
Aussi une variable liée à une classe s'appelle un attribut.

*/

C++

# class | création ds le fichier en tête

/ Nous créons nous même notre propre constructeur.
Notre constructeur ici sera Human(), qui sera réutilisé au moment d'étayer notre classe.

```
1  #ifndef class_hpp
2  #define class_hpp
3  #include <stdio.h>
4
5
6  class Human
7   {
8
9      public:
10       Human(); // Nous créeons nous même notre propre constructeur.
11
12  };
13
14
15 #endif /* class_hpp */
16
```

C++

# class | construction basique

Nous créons nous même notre propre constructeur.
Notre constructeur ici sera Human(), qui sera réutilisé au moment d'étayer notre classe.
Au moment de sa structuration, nous connectons notre constructeur à l'aide de l'opérateur de portée ::

```cpp
#include "class.hpp"
#include <iostream>

Human::Human() // :: est un opérateur de portée|Human() est le constructeur
{
    std::cout << "We are building the prototype of a human person " << this << std::endl;

}

```

```
We are building the prototype of a human person 0x7ffeefbff568
Program ended with exit code: 0
```

C++

# class | Compilation

```cpp
#include <iostream>
#include "class.hpp"

int main()
{

    Human h{};

    return 0;

}

```

```
We building the prototype of a human person
Program ended with exit code: 0
```

C++

# this

/*

this fait appel à l'instance en cours.
Une fois qu'on accède à la méthode Human en affichant
"We are building the prototype of a human person" ,
le this fait référence à l'instance, ici l'objet qui appelle cette méthode.

*/

C++

# class | this

Il s'agira dans notre exemple de h. this ira le récupérer dans son emplacement mémoire.
Au moment de son utilisation nous obtiendrons donc son adresse ->

```cpp
#include "class.hpp"
#include <iostream>

Human::Human() // :: est un opérateur de portée|Human() est le constructeur
{
    std::cout << "We are building the prototype of a human person " << this << std::endl;

}

```

```
We are building the prototype of a human person 0x7ffeefbff568
Program ended with exit code: 0
```

C++

# class | this x 2

Dans le cas où nous créons un 2ème human, une fois compilé on s'aperçoit bien qu'il y a deux instances, ou bien deux objets et qu'ils occupent bien deux adresses distinctes.

```cpp
#include <iostream>
#include "class.hpp"

int main()
{

    Human h{};
    Human hA{};

    return 0;

}
```

```
We are building the prototype of a human person 0x7ffeefbff568
We are building the prototype of a human person 0x7ffeefbff560
Program ended with exit code: 0
```

C++

# class | destructeur ~

Une fois le constructeur crée, la classe initialisé, nous l'utilisons.
Si nous ne créons pas de destructeur , le compilateur en créera un automatiquement.
Ds le fichier header on le crée à la suite du constructeur.

```
1  #ifndef class_hpp
2  #define class_hpp
3
4  class Human
5   {
6
7      public:
8        Human(); // Nous créeons nous même notre propre constructeur.
9       ~Human();
10
11  };
12
13
14 #endif /* class_hpp */
15
```

C++

# class | destructeur ~ | Compilation

Une fois appelé, nous voyons que le constructeur execute sa tâche en mode FILO, à savoir le dernier rentré sera le premier éliminé. Avec this, les deux addresses dans lesquelles les deux objets avaient été placées sont alors vidées

```cpp
#ifndef class_hpp
#define class_hpp

class Human
 {

    public:
      Human(); // Nous créeons nous même notre propre constructeur.
     ~Human();

 };


#endif /* class_hpp */

```

```
We are building the prototype of a human person 0x7ffeefbff568
We are building the prototype of a human person 0x7ffeefbff560
We are now destroying the prototype of a human person 0x7ffeefbff560
We are now destroying the prototype of a human person 0x7ffeefbff568
Program ended with exit code: 0
```

C++