R2.01 : Développement Orienté Objets C++

cours 1

François Delobel, Anaïs Durand, Abdel Hasbani, Laurent Provot, Carine Simon

IUT CLERMONT AUVERGNE
Aurillac - Clermont-Ferrand - Le Puy-en-Velay
Montluçon - Moulins - Vichy

Bref historique

Bref historique

- dans les années 1970 : invention du C par Dennis Ritchie
- au début des années 1980 : C with Classes par Bjarne Stroustrup
  (C + des notions objet venant de Simula)
- 1983 : C with Classes devient C++
- 1998 : C++ normalisé => C++98
- évolution successives : C++03, C++11, C++14, C++17, C++20

Intérêts

- un grand nombre de fonctionnalités communes au C et au C++
- multi-paradigme (objet, procédural, fonctionnel, ...)
- contrôle d'erreur plus fort qu'en C
- portabilité des fichiers sources
- performance du C
- ancien langage qui a su évoluer
- très utilisé dans le domaine du calcul haute performance, du jeu, de la recherche ...

Du procédural à l'objet

En C

```
typedef struct{
    char* nom;
    char* numTel;
}Contact;
Contact creerContact(char* nom, char* numTel);
void afficheContact(Contact c);
void modifContactNumTel(Contact* c, char* numtel);
```

Inconvénients :

- séparation des données et des fonctionnalités
- notation lourde (nom de fonction et paramètres)

Idée

- **tout** grouper dans Contact : données et fonctionnalités

$\Longrightarrow$ les Classes : prolongement des structures du C

Réalisation : C vs C++

déclaration

```c
typedef struct{
    // les données
   char* nom;
   char* numTel;
}Contact;
// fonctions :
void afficheContact(Contact c);
void modifContactNumTel(Contact* c,
                        char* numero);
```

```c++
class Contact{
   // les données
   string nom;
   string numTel;
public:
   // les fonctions
   void affiche();
   void modifNumTel(string numero);
};
```

utilisation :

```c
Contact juju;
afficheContact(juju);
modifContactNumTel(&juju, "06.66.66.66.66");
```

```c++
Contact juju;
juju.affiche();
juju.modifNumTel("06.66.66.66.66");
```

Vocabulaire

- Classe : type (comme une struct ou un type de base)
- Objet : instance de classe (c-à-d : variable du type de la classe)
  dans l'exemple précédent :
  - `Contact` est une classe,
  - `juju` une instance de cette classe
- un objet est défini par :
  - son identifiant : symbole (exemple : `juju`)
  - des attributs : données membres
  - des méthodes : fonctions membres

Cycle de vie d'un objet

Cycle de vie

```
0   {
1       Contact juju;
2       juju.affiche();
3   }
```

0 début d'un bloc de code

1 un objet est construit (par appel au constructeur)

2 puis utilisé

3 et enfin détruit (par appel au destructeur) car fin de ce bloc de code

Remarque : un bloc de code définit la portée d'une variable.
ici la portée de `juju` est le bloc de code allant de sa définition à "}"

Constructeur

avant de pouvoir être utilisé un objet est construit.
$\rightarrow$ appel du constructeur (méthode nommée comme la classe)

plusieurs constructeurs :

- constructeur par défaut

```
Contact();
```

- constructeur avec paramètres

```
Contact(string nom, string numero);
Contact(string nom, string numero="06.66.66.66.66");
```

Remarque : Si aucun constructeur défini dans la classe, un constructeur par défaut qui ne fait rien est créé automatiquement.

Constructeur (code)

```
class Contact{
    // les données
    std::string nom;
    std::string numTel;
public:
  // constructeurs
  Contact();
  Contact(std::string nom, std::string numero);
  // méthodes
  void affiche();
  void modifNumTel(std::string numero);
};
```

```
#include "contact.hpp"
Contact::Contact()
  :nom{"inconnu"}, numTel{"00.00.00.00.00"}
{std::cout << nom << "(" << numTel << ")␣est␣créé" << std::endl;}

Contact::Contact(std::string nom, std::string numero)
  :nom{nom}, numTel{numero}
{std::cout << "création␣de␣:␣"<< nom << "(" << numTel << ")" << std::endl;}

void Contact::affiche(){
  std::cout << nom << ":" << numTel << std::endl;
}
void Contact::modifNumTel(std::string numero)  { numTel = numero; }
```

Utilisation

```
...
Contact::Contact()
  :nom{"inconnu"}, numTel{"00.00.00.00.00"}
{std::cout << nom << "(" << numTel << ") est créé" << std::endl;}

Contact::Contact(std::string nom, std::string numero)
  :nom{nom}, numTel{numero}
{std::cout << "création de : "<< nom << "(" << numTel << ")" << std::endl;}

void Contact::affiche(){ std::cout << nom << ":" << numTel << std::endl; }
void Contact::modifNumTel(std::string numero)  { numTel = numero; }


Contact juju;
Contact lisi{"lisebeth", "06.11.22.33.44"};
juju.affiche();
lisi.affiche();
```

affichage obtenu à l'écran :

```
inconnu (00.00.00.00.00) est créé
création de lisebeth (06.11.22.33.44)
inconnu : 00.00.00.00.00
lisebeth : 06.11.22.33.44
```

Utilisation

Une fois construit l'objet est utilisé

```
juju.modifNumTel("07.77.77.77.77");
juju.affiche();
```

et les objets peuvent se demander des services en faisant appel aux méthodes.

Destruction

Comme en C,

- si déclaration automatique,
  $\Rightarrow$ destruction automatique à la fin du bloc
  Exemple :

```
{
  Contact juju;
  ....
}
```

- si déclaration dynamique (new ou new[ ] en C++),
  $\Rightarrow$ destruction manuelle (delete ou delete[ ] en C++).
  Exemples :

```
Contact* tom = new Contact{"Tom", "07.77.77.77.77"};
Contact* tab = new Contact[2]; // tableau de 2 Contact
...                                "inconnu"
delete tom;
delete[] tab;
```

Destruction

La destruction d'un objet est réalisée par appel au destructeur.

destructeur : méthode de la classe ayant même nom que la classe précédé de ~

Exemple : `~ Contact(); //dans la classe Contact`
avec le code :

```
Contact::~ Contact(){ }
```

Rmq : dans cet exemple, elle n'a rien à faire de particulier mais de manière générale elle permet de libérer proprement la mémoire allouée à l'objet

Destruction

Exemple :

```
class Contact{
    std::string nom;
    int nbNumTels;
    std::string * numTels;
public:
  Contact();
  Contact(std::string nom, std::string numero);
  void ajoutNumTel(std::string num);
  ~Contact();
};


Contact::Contact(): nom{"inconnu"}, nbNumTels{0}, numTels{new std::string[3]}
{}

Contact::Contact(std::string nom, std::string numero)
  : nom{nom}, nbNumTels{1}, numTels{new std::string[3]}
{
  numTels[0] = numero;
}

void Contact::ajoutNumTel(std::string numero)  {
  if(nbNumTels==3) std::cerr << "ajout impossible" << std::endl;
  else{
    numTels[nbNumTels] = numero;
    nbNumTels++;
  }
}

Contact::~Contact(){ delete[] numTels; }
```