

La programmation orientée objets

Introduction au C++ et à la programmation objet

E. Courcelle

CALMIP, UMS 3669

Mai 2016

- 1 Les modules, ancêtres des objets
 - Variables locales, variables globales
 - L'approche modulaire
- 2 Les objets
 - Une pile de char
 - Des objets intégrés au système de typage
 - Opérations sur les objets
- 3 L'héritage
 - Classer les objets
 - Dessiner des formes
- 4 Résumé
 - Donner un sens à ses classes
- 5 Résumé
 - Classes et objets

Une variable globale...

Une **variable globale** est accessible à partir de
toutes les fonctions du programme

```
int x=0;
int fonction1() {
    x=1;
    return 2 * x;
}
int fonction2() {
    x = x + 2;
    return 2 * x;
}
```

...Une variable locale

Une **variable locale** n'est accessible qu'à partir d'un **bloc déterminé** (par exemple une fonction):

```
int fonction() {  
    int x=1;  
    return 2 * x;  
}
```


Un module: Interface et implémentation

Interface du module

- Noms des fonctions
- Ce qu'elles font
- Ce qui rentre
(les paramètres des fonctions)
- Ce qu'elles renvoient
(le type de la valeur de retour)

Implémentation du module

- Variables du module
("globales")
- Code des fonctions

Prototypage et travail en équipe

L'encapsulation des données permet:

- De travailler **par prototypage** en commençant par l'interface publique
- De travailler en équipe:
 - On se met d'accord **sur l'interface**
 - Chacun écrit l'implémentation **"comme il/elle veut"**

Un module qui définit une pile de caractères

L'interface du module est constitué par deux fonctions: l'une pour empiler, l'autre pour dépiler.

```
// Mettre un caractere en haut de la pile  
// et la faire grossir d'un caractere  
void push_char(char);  
  
// Renvoyer le caractere du haut de la pile  
// et diminuer la pile d'un caractere  
char pop_char();
```

Gérer plusieurs piles de caractères

On ajoute un entier qui permettra d'identifier la pile:

```
// Mettre un caractere en haut de la pile  
// et la faire grossir d'un caractere  
// id permet de choisir une pile parmi plusieurs  
void push_char(int id, char c);  
  
// Renvoyer le caractere du haut de la pile  
// et diminuer la pile d'un caractere  
// id permet de choisir une pile parmi plusieurs  
char pop_char(int id);
```

Créer un nouveau type de variables

La solution du C++: créer un **nouveau type** de variables

```
class Stack_char {  
  
    // Mettre un caractere en haut de la pile  
    // et la faire grossir d'un caractere  
    void push(char c);  
  
    // Renvoyer le caractere du haut de la pile  
    // et diminuer la pile d'un caractere  
    char pop();  
  
}  
  
// Créer trois piles dans notre programme  
Stack_char a,b,c;  
  
a.push('a');  
a.push('z');  
b.push('u');  
c.push('t');  
cout << a.pop(); // Imprime z
```

Comme un type natif !

Un tableau de piles de caractères:

```
Stack_char [10] tableau_de_piles;
```

Une pile initialisée (c++11):

```
Stack_char a = { 'a', 'z', 'e', 'r', 't', 'y' };
```

Recopier une pile dans une autre:

```
Stack_char b = a;
```

Effectuer un transtypage:

```
Stack_char a;  
Stack_int b = (Stack_int) a;
```

... à condition de définir ce que tout ça signifie !

Ciel, quel désordre dans mon argenterie !



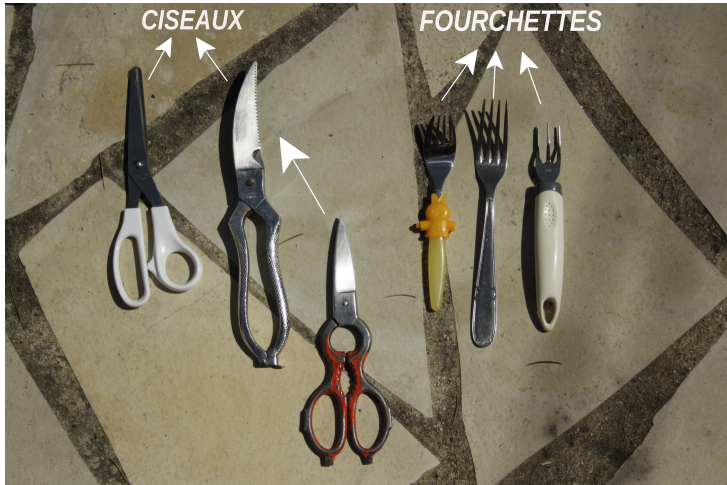
C'est mieux comme ça !



C'est mieux comme ça !



C'est mieux comme ça !



Plusieurs formes

On voudrait faire un programme pour dessiner plusieurs formes:
Créons donc plusieurs types d'objets:

```
class Circle;  
class Triangle;  
class Square:
```

Dessin de formes... et d'une pile !
mouais... Mais que ce soient des formes ou des piles, on mettrait tout sur le même plan ?

NON ! Une forme est une forme !

Une forme est un objet qu'on peut **dessiner** et qui a **un centre** et **une couleur**.
Et il y en a **plusieurs sortes**.

```
enum kind {circle , triangle , square};  
class Shape {  
    point center;  
    color col;  
    kind k;  
public:  
    point where() {return center};  
    void draw();  
};
```

Voilà qui est déjà mieux !

La fonction draw

Imaginons à quoi va ressembler la fonction draw:

```
void Shape::draw() {  
    if (kind==cercle) {  
        // dessine-moi un cercle !  
    } else if (kind==triangle) {  
        // dessine-moi un triangle !  
    } else if (kind==square) {  
        // dessine-moi un carre !  
    } else {  
        // oups je ne sais pas dessiner cela !  
    }  
}
```

Pour le passage à l'échelle on repassera !

Plus il y a de formes à dessiner, plus la fonction draw deviendra compliquée et illisible !

Ben OUI ! Une forme est... une forme !

Reprenons: une forme, c'est un truc qu'on peut dessiner,
qui a une couleur et un centre.

Sauf que je ne sais pas à quoi elle ressemble, donc je ne sais pas le dessiner.

En c++, ça s'écrit comme ça:

```
class Shape {  
private:  
    point center;  
    color col;  
public:  
    point where() { return center; };  
    virtual void draw()=0;  
}
```

draw est une **fonction virtuelle pure**, on sait qu'elle existera un jour, mais on ne sait pas ce qu'elle fera !

Une classe abstraite...

Les objets de classe Shape ne **ne peuvent pas** être créés !
C'est normal puisque Shape est une classe virtuelle:

```
Shape une_forme; // NE COMPILE PAS !
```

Par contre, je peux imaginer une fonction qui prenne une forme en paramètre:

```
void arriere_plan(const & Shape); // PAS DE PROBLEME !
```

Je ne peux pas créer une forme, mais si j'en ai une je peux la passer à une fonction...
Mais comment pourrais-je avoir une forme ?

...Des classes concrètes...

Mais à quoi sert donc ce concept de forme, si ce n'est à dessiner des cercles, des triangles, des carrés ?

On va donc écrire en c++ que carre, cercle, triangle sont **des sortes de** forme:

```
class Circle: public Shape {  
    ...  
public:  
    virtual void draw();  
};  
class Triangle: public Shape {  
    ...  
public:  
    virtual void draw();  
};  
class Square: public Shape {  
    ...  
public:  
    virtual void draw();  
};
```


...qu'on sait dessiner et créer !

Pour utiliser une forme on fait comme d'habitude:

- Déclarer les variables en leur donnant le type adéquat
- Appeler la fonction draw pour dessiner

```
Circle c;  
Triangle d;  
Square s;  
c.draw(); d.draw(); s.draw();
```

Pour créer une nouvelle forme on doit:

- Créer une classe qui dérive de Shape
- Implémenter la fonction draw associée
- Ainsi on ne touche pas à ce qui existe déjà !
- ...Et maintenant ça passe à l'échelle

Héritage correct

L'héritage permettra de faire évoluer votre code en douceur...
à condition qu'il soit correctement défini !

Héritage correct

Les fonctions virtuelles des classes dérivées doivent:

- En faire au moins autant que la fonction de la classe de base
- Ne pas avoir d'exigence supérieure à celles de la classe de base

Donner un sens à ses classes

Des classes chargées de sens

Trois types principaux, du point-de-vue de la sémantique

- Sémantique de **valeur**
- Sémantique d'**entité**
- Gestionnaires de ressources

Classes et objets

Qu'est-ce qu'une classe ?

Une classe est la description d'un type d'objets.

- C'est en quelque sorte un **moule**
- Il permettra par la suite de créer de nouveaux objets
- Techniquement, c'est un type de données

Qu'est-ce qu'un objet ?

Un objet est une **variable** du programme, qui se caractérise par :

- Un état
- Un comportement
- Une identité