جامعة ابن طفيل Université Ibn Tofail
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵟⴼⵉⵍ Ibn Tofail University

Concepts Classes et Objets sous C++

Med. AMNAI
Filière SMI - S5
Département d'Informatique

23 novembre 2020

Plan

Plan

Plan

Classe

- Une classe est la **généralisation** de la notion de **type** défini par l'utilisateur ;
- Elle regroupe à la fois des données (données membres, attributs ou propriétés) et des méthodes (fonctions membres).
- La partie **données** représente l'aspet *statique* et la partie **méthode** représente l'aspet *dynamique* de la classe ;
- En programmation orientée objet pure, les **données** sont **encapsulées** et leur accès ne peut se faire que par le biais des méthodes de la même classe.

Exemple

```
#include<iostream>
using namespace std;

class point{ // declaration de la classe point
    int x, y;
    public :
        void initialise(int,int);
        void affiche();
        void deplace(int,int);
};
//----Definition des fonctions membres-----
void point::initialise(int a,int b){
    x=a;
    y=b;
}

void point::affiche(){
    cout<< "on est a : " << x << " - " << y << endl;
}

void point::deplace(int a,int b){
    x = x + a ;
    y = y + b ;
}
```

Membres Privé et Public

Le mot clé "**public**" précise que tout ce qui le suit (données membres ou fonctions membres) sera *public*, le reste étant **privé**. Ainsi :

- *x* et y sont deux membre **privés**.
- "*initialise*", "*deplace* et "*affiche* sont trois fonctions membres **publics**.
- La classe "*point*" ci-dessus peut être définie et utilisée comme dans l'exemple suivant (classe.cpp) :

```
main(){
    point p1, p2, p3;
    p1.initialise(1,3);
    p1.affiche();
    p1.deplace(2,4);
    p1.affiche();
    p2.initialise(5,5);
    p2.affiche();
}
```

```
on est a : 1 - 3
on est a : 3 - 7
on est a : 5 - 5
```

Objet (Instance)

- On dit que **p1** et **p2** sont des **instances** de la classe "*point*" ou encore que se sont des **objets** de type "*point*".
- Tous les membres données de "*point*" sont **privés ou encapsulés**. Ainsi, on ne peut pas accéder à **x** ou à **y**.
- Toutefois, les membres données ou les fonctions membres peuvent être **privées** en utilisant le mot clé "**private**" ou **publiques** en utilisant le mot clé "**public**".

```
class C {
    public :
            ..............;
            ..............;
    private :
            ..............;
            ..............;
};
```

Affectation d'Objet

- Elle correspond à une **recopie** de valeurs des membres donnés (*publics* ou *privés*). Ainsi, avec les déclarations :

```
class point {
    int x, y;
    public :
        ..............;
};
    point p1, p2;
```

- L'affectation **p2=p1** ; signifie la recopie des valeurs **x** et **y** de **p1** dans les membres correspondants de **p2**.

Principe : Constructeur, Destructeur

Un **constructeur** est une **fonction** appelée **automatiquement** après la création d'un objet. Ceci aura lieu quelque soit la classe d'allocation de l'objet : **statique**, **automatique** ou **dynamique**.

- De la même façon, un objet pourra posséder un **destructeur**, il s'agit également d'une **fonction** membre qui est appelée **automatiquement** au moment de la destruction de l'objet correspondant.
- Par convention, le **constructeur porte le même nom que la classe**.
- Quand au **destructeur**, il **porte le même nom que la classe** précédé du symbole **(tild)**.

```
class point {
    int x, y;
    public :
            ..............;
};
    point p1, p2;
```

Exemple (classeCstr.cpp)

```
class point{ // declaration de la classe point
    int x, y;
    public :
        point(int,int);
        void affiche();
        void deplace(int,int);
};
//----Definition des fonctions membres-----
point::point(int a,int b){
    x=a;
    y=b;
}
void point::affiche(){
    cout<< "on est a : " << x << " - " << y << endl;
}
void point::deplace(int a,int b){
    x = x + a ;
    y = y + b ;
}
main(){
    point p1(1,7), p2(2,5);
    p1.affiche(); p1.deplace(3,7); p1.affiche();
    p2.affiche(); p2.deplace(3,3); p2.affiche();
}
```

```
on est a : 1 - 7
on est a : 4 - 14
on est a : 2 - 5
on est a : 5 - 8
```

Exemple

```
#include<iostream>
using namespace std;


class demo{
    int num ;
    public :
    demo(int);
    ~demo();
};

demo::demo(int n){
    num=n;
    cout<< "Appel constr numero : " << num << endl;
}

demo::~demo(){
    cout<< "Appel destr numero : " << num << endl;
}
```

Exemple (suite)

Suivre la trace des objets automatiques créés dans le programme suivant en affichant les moments de leur **construction** et leur **destruction**.

```
main(){
     void f(int);
     demo obj(7);

     for (int i=0; i<4; i++)
          f(i);
}

void f(int m){
     demo obj(m) ;
}
```

```
Appel constr numero : 7
Appel constr numero : 0
Appel destr numero : 0
Appel constr numero : 1
Appel destr numero : 1
Appel constr numero : 2
Appel destr numero : 2
Appel constr numero : 3
Appel destr numero : 3
Appel destr numero : 7
```

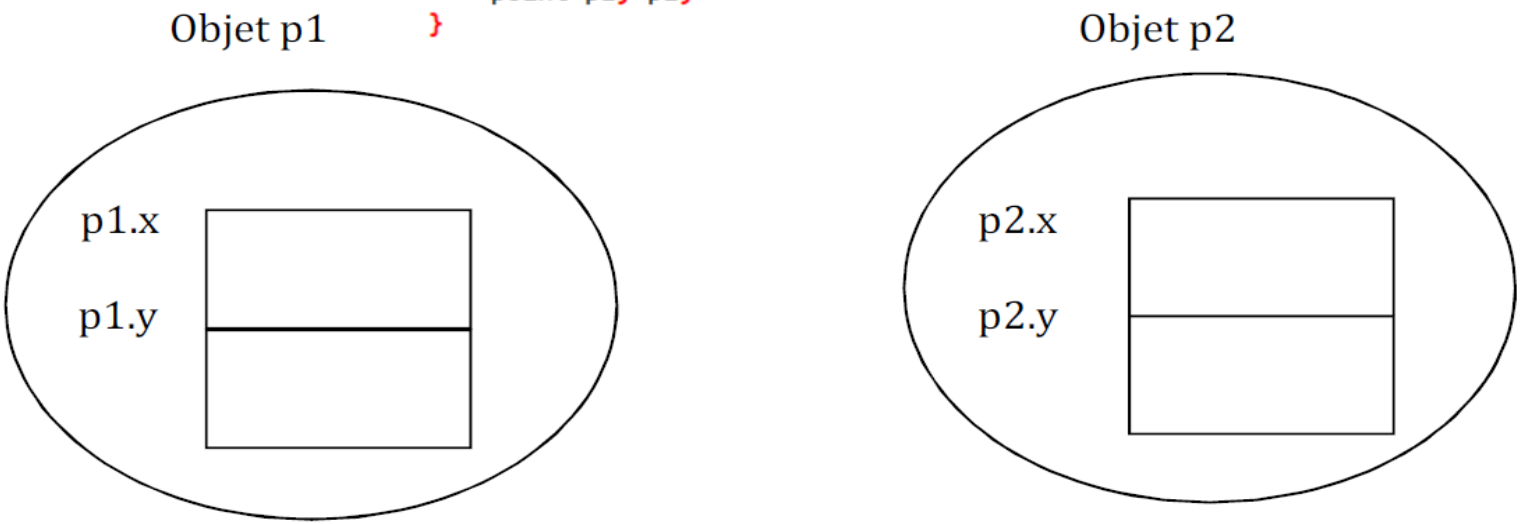
Règles d'utilisation

- Un constructeur peut ou non comporter quelques arguments.
- par définition, un constructeur ne renvoie pas de valeur et la présence de void (dans ce cas précis) est une erreur.
- Un destructeur, par définition, ne peut pas disposer d'arguments et ne renvoie pas de valeur.

Membres statiques

Lorsqu'on crée différents objets d'une même classe, chaque objet possède ces propres données membres.

```
calss point{
    int x ;
    int y ;
    ............
    ............
};
int main(){
    point p1, p2;
}
```



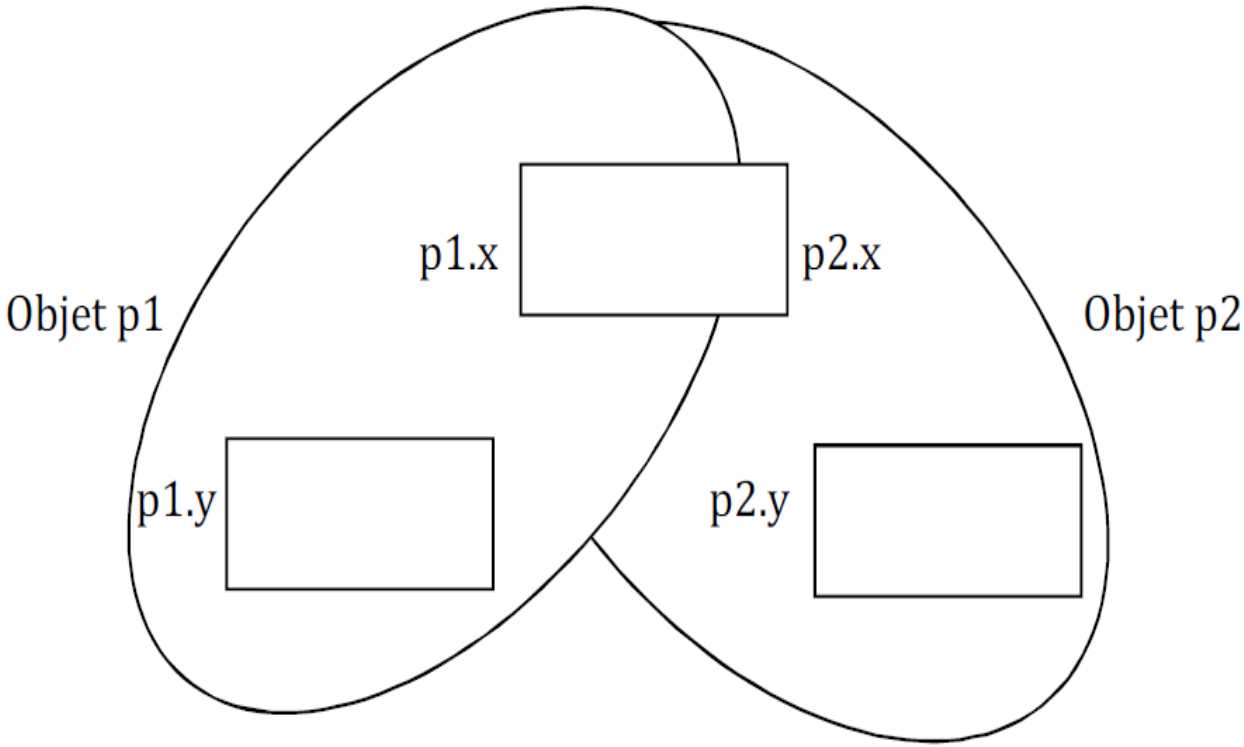
Membres statiques (suite)

Pour pouvoir **partager** des données entre objets de la même classe, **on les déclare statiques**.

```
class point{
    static int x;
    int y;
    ..........
};
int main(){
    point p1, p2; //Une déclaration provoquera
}
```

Membres statiques (suite)

Une déclaration telle que : **point p1, p2 ;** provoquera :



p1.x est identique à p2.x mais p1.y <> p2.y.

Initialisation

- Les membres statiques existent **en un seul exemplaire** quelque soit le nombre d'objets de la classe correspondante, et même si aucun objet de la même classe n'a été créé.
- L'**initialisation** ne peut pas être faite par le constructeur de la classe.
- Un membre statique doit être **initialisé** explicitement à l'**extérieur** de la déclaration de la classe.

```
class explClass{
        static int n = 2 ; //erreur

} ;

int explClss::n = 2 ;
```

Exemple

```
#include<iostream>
using namespace std;

class cpt_obj{
    static int c;
    public :
        cpt_obj();
        ~cpt_obj();
};
int cpt_obj::c=0;
cpt_obj()::cpt_obj(){
    cout<< "Construction, il y a maintenant : "<< ++c << " Objet\n";
}

cpt_obj::~cpt_obj(){
    cout<< "Destruction, il reste : "<< --c << " Objet\n";
}

main(){
    void f();
    cpt_obj O1; f(); cpt_obj O2;
}

void f(){
    cpt_obj a, b;
}
```

```
Construction, il y a maintenant : 1 Objet
Construction, il y a maintenant : 2 Objet
Construction, il y a maintenant : 3 Objet
Destruction, il reste : 2 Objet
Destruction, il reste : 1 Objet
Construction, il y a maintenant : 2 Objet
Destruction, il reste : 1 Objet
Destruction, il reste : 0 Objet
```

Exercice 1

Ecrire un programme permettant de créer des objets ayant chacun :

- un tableau de 5 éléments de type entier en tant que donnée ;
- une fonction pour remplir le tableau, une fonction pour trier le tableau et une fonction pour afficher le contenu du tableau en tant que méthodes.

Exercice 1 (solution)

```
class tableau{
    int t[5];
    public :
        tableau(){ for(int i=0;i<5;i++) t[i]=0; }
        void remplir();
        void afficher();
        void trier();
};
void tableau::remplir(void){
    cout<<"Veuillez remlir le tableau avec 5 entiers :"<<endl;
    for(int i=0;i<5;i++)
        cin>>t[i];
}
void tableau::afficher(void){
    for(int i=0;i<5;i++)
        cout<<t[i]<<"\t";
    cout<<endl;
}
void tableau::trier(void){
    int a;
    for(int i=0;i<4;i++)
        for(int j=i+1;j<5;j++)
            if(t[i]<t[j]){
                a=t[i];
                t[i]=t[j];
                t[j]=a;
            }
}
main(){
    tableau t1;
    t1.remplir();
    cout<<"Avant tri : "<<endl; t1.afficher();
    t1.trier();
    cout<<"Apres tri : "<<endl; t1.afficher();
}
```

Exercice 2

Reprendre le même programme de l'exercice 1 :

- en remplaçant le tableau de 5 éléments par un tableau dynamique de 'ne' éléments ;
- ajouter un destructeur ;
- instancier des objets ayant des tableaux dynamiques de différentes tailles.

Exercice 2 (solution)

```
class tableau{
        int *t;
        int ne;
    public :
        tableau(int n){
            ne=n;
            t=new int[ne];
            for(int i=0;i<ne;i++)
                t[i]=0;
        }
        void remplir();
        void afficher();
        void trier();
        ~tableau(){
            delete []t;
        }
};
//----------------------------------------
void tableau::remplir(){
    cout<<"Veuillez remplir le tableau avec "<<ne<<" entiers :"<<endl;
        for(int i=0;i<ne;i++)
            cin>>t[i];
}
//----------------------------------------
void tableau::afficher(){
//----------------------------------------
void tableau::trier(){
//----------------------------------------
main(){
    tableau t1(5);
    t1.remplir();
    cout<<"Avant tri : "<<endl; t1.afficher();
    t1.trier();
    cout<<"Apres tri : "<<endl; t1.afficher();
}
```