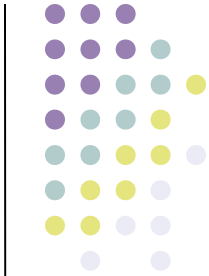
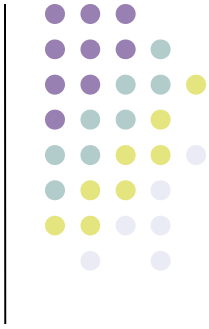
Classes et Objets



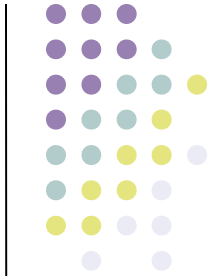
- Programmation orienté objet : POO
- Classe : généralisation de la notion de structure et de type
- Une classe contient:
  - Des données (données membres)
  - Des fonctions (fonctions membres ou méthode)
- Principe fondamental de la POO : L'encapsulation – les données ne sont accessibles que via des fonctions membres

Rappel structure



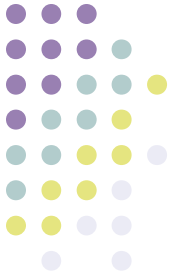
```
typedef struct Point {
    double x;
    double y;
} POINT;

POINT a, b; // Permet de déclarer et de créer
            // deux variables a et b de type point.

a.x = 3; // On accède aux données (ou champs)
a.y = 2; // de la structure via "."
```

Classe point : dans le fichier point.h



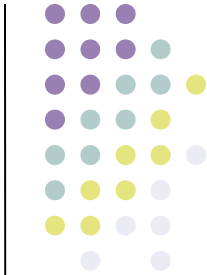
```
class Point {

    private :

        double x;
        double y;

    public :

        void initialise(double, double);
        void deplace(double, double);
        void affiche();

};
```

Classe point : dans le fichier point.cpp



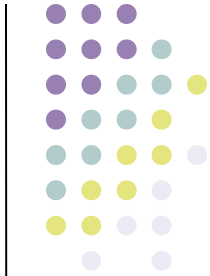
```
void Point::initialise(double abs, double ord) {
    x = abs;
    y = ord;
}

void Point::deplace(double  dx, double dy) {
    x += dx;
    y += dy;
}

void Point::affiche() {
    cout << '(' << x << ", " << y << ')' << endl;
}
```

Utilisation de la classe



```
# include "point.h"

int main(int argc, char ** argv)
{
    point a, b;
    a.initialise(5.7, 2.3);
    a.affiche();
    a.deplace(3.5, 7.2);
    a.affiche();
    b.affiche();
    return 0;
}
```

Remarques



- Dans le jargon de la POO, on dit que a et b sont:
  - → des instances de la classe point.
  - → des objets de type point
- par défaut, x et y sont des données membres privées :

→ En dehors des fonctions membres, l'instruction `a.x = 5;` est interdite

- → Il faut prévoir des fonctions d'accès en lecture et en écriture

- Les fonctions membres sont ici publiques. Il est possible d'en avoir des privées

Remarques

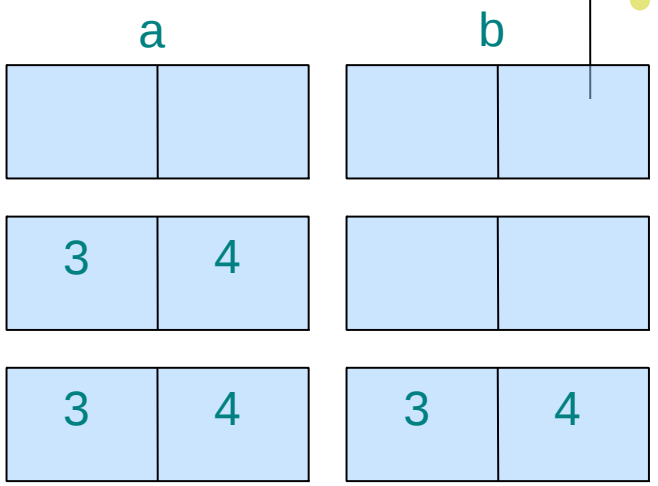
- Les mots clés *public* et *private* peuvent apparaître à plusieurs reprises:

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

-Par défaut les membres d'une classe sont privées

Affectation d'objets

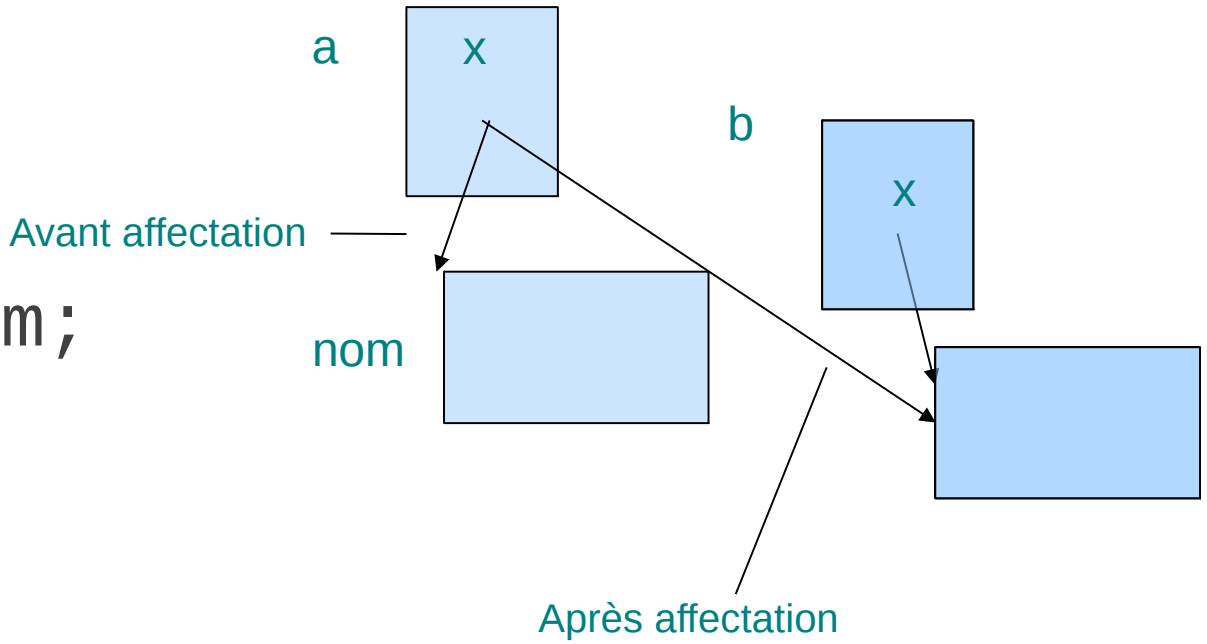
```
Point a, b;


a.initialise(3.0, 4.0);
b = a;
```



- On ne peut pas écrire : `b.x = a.x; b.y = a.x;` car x et y sont privées
- Notez que l'opération b = a est toujours légale. Elle marche lorsque les données ne sont pas allouées dynamiquement.

Attention à l'utilisation de « = »

```
class Classe
{
    int x;
    char *nom;
};

Classe a, b;
a = b;
```



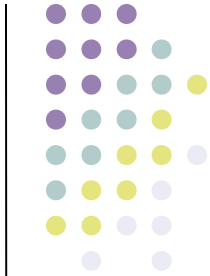
Constructeur

- C'est une fonction appelée automatiquement à la création d'un objet.
- Son rôle est d'initialiser l'objet convenablement
- De faire des allocation dynamiques si nécessaire
- Il est toujours nécessaire d'en faire une ou plusieurs
- Le constructeur à le même nom que la classe et ne n'a pas de valeur de retour

Constructeur

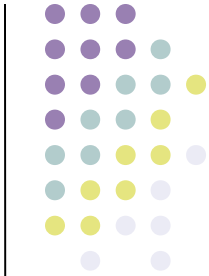
- La rôle principal du constructeur est de garantir la cohérence des objets
- Un objet carré ne doit pas avoir une valeur négative comme longueur d'un des côtés
- Un objet cercle ne doit pas avoir un rayon négatif
- .....
- .....

Exemple : la classe point



```
class Point {

    private :

        double x;
        double y;

    public :

        Point(double , double)
        void deplacer(double, double);
        void afficher();

};
```

Exemple la classe point



- On ne peut plus faire cela:

```
Point a;
```

- En fait, à partir du moment où un constructeur est défini, il doit pouvoir être appelé lors de la création de l'objet.

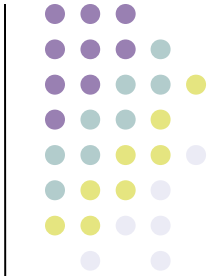
```
Point a(3.5, 6.2);
Point b(0.0 , 0.0);
```

- C'est un bon moyen de garder la cohérence des objets

Le fichier point.h :

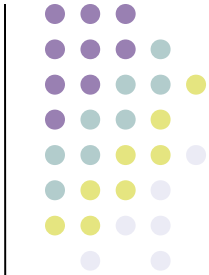
```
# ifndef _POINT_H_
# define _POINT_H_

# include <iostream>

class Point {

    private :

        double x;
        double y;

    public :

        Point(double , double);
        void deplacer(double, double);
        void afficher();

};

# endif // _POINT_H_
```

Le fichier point.cpp :



```
# include "point.h"

using namespace std;

Point::Point(double abs, double ord) {
    x = abs;
    y = ord;
}

void Point::deplacer(double  dx, double dy) {
    x += dx ;
    y += dy;
}

void Point::afficher() {
    cout << '(' << x << ", " << y << ')' << endl;
}
```

Le fichier main.cpp :



```
# include "point.h"

int main(int argc, char ** argv)
{
    Point a(5.0, 7.1);
    a.afficher();
    a.deplacer(-1.0, 8.3);
    a.afficher();
    return 0;
}
```

Le destructeur

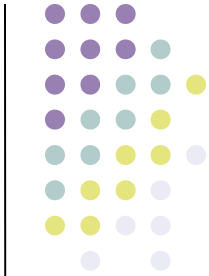
- Le destructeur est une fonction appelée au moment ou un objet doit disparaître
- L'appel au destructeur est fait automatiquement

```
......
{
    Point  a(3.5, 5.5);
    ....
}
```

C'est ici que le destructeur est appelé

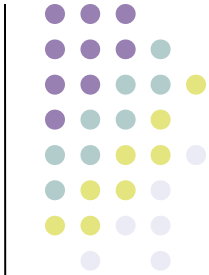
- Le destructeur a le même nom que la classe précédé d'un "~" : `~Point()`
- Le destructeur n'a pas d'arguments

La classe test : test.h



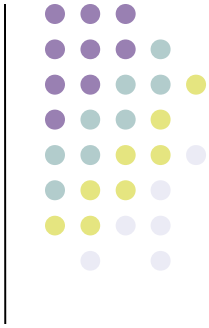
```
# ifndef _TEST_H_
# define _TEST_H_

# include <iostream>

class Test {

    private :

        int num;


    public :

        Test(int);
       ~Test();

};

# endif // _TEST_H_
```

La classe test : test.cpp



```
# include "test.h"

using namespace std;


Test::Test(int n)
{
    num = n;
    cout << " ++ appel du constructeur -- num = "
         << num << endl;
}

Test::~Test()
{
    cout << " -- appel du destructeur -- num = "
         << num << endl;
}
```

La classe test : main.cpp



```
# include "test.h"

void function(int p)
{
    Test t(2 * p);
}

int main(int argc, char ** argv)
{
    Test t(1);
    for (int i = 1 ; i <= 2 ; i++)
        function(i);
    return 0;
}
```

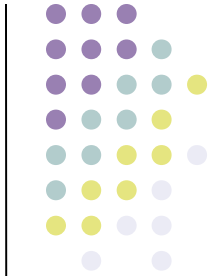
++ appel du constructeur --> num = 1
++ appel du constructeur --> num = 2
- - appel du destructeur --> num = 2
++ appel du constructeur --> num = 4
- - appel du destructeur --> num = 4
- - appel du destructeur --> num = 1

Les fonctions membre constantes

- `const int n = 3;`
- `n = 12;` est une instruction interdite
- En C++ une fonction peut être déclarer comme constante, ca veut dire:
  - la fonction ne peut pas modifier les données membres de la classe.

```
class Point {

    ......

    public :

        Point(double x, double y);
        void affiche() const;
        Void deplace(double dx, double dy);

};
```

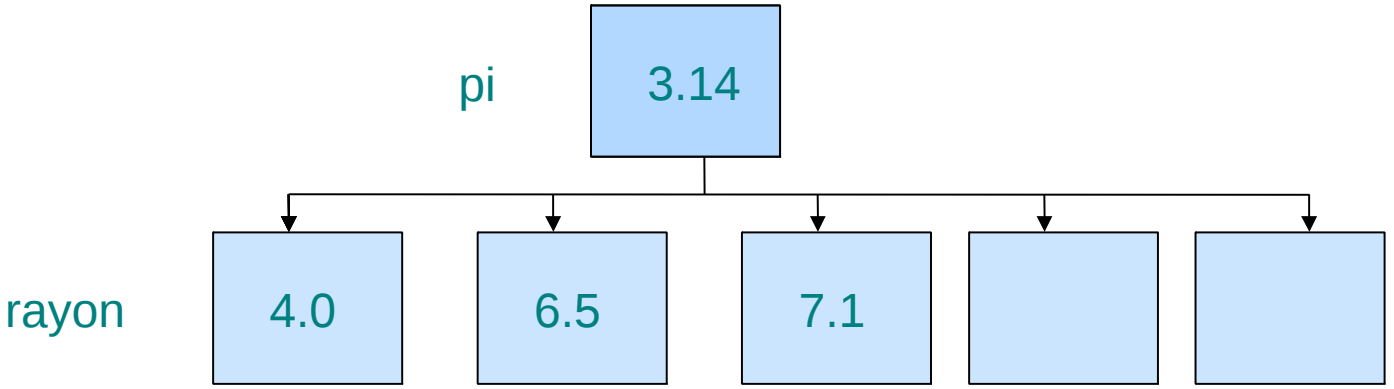
Les données membre statiques



```
class Cercle {

    private :

        static double pi;
        double rayon;

    public :

        ......

};
```

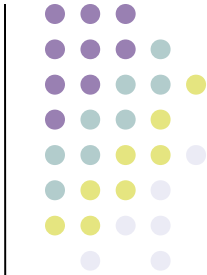
Données membre statiques

- Tous les objets de type cercle partage la même donnée membre pi
- Économie de l'espace mémoire
- Cohérence



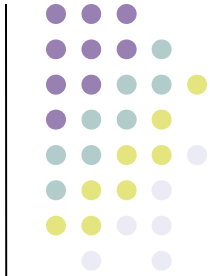
- Pour initialiser : `double cercle::pi = 3.14;`

Utilisation : le nombre d'occurrences des objets



```
# include <iostream>

using namespace std;

class Comptable {

    private :

        static int compteur;

    public :

        Comptable();
       ~Comptable();

};


int Comptable::compteur = 0;

Comptable::Comptable() {
    compteur++;
    cout << " ++ il y a " << compteur
         << " objets." << endl;
}

Comptable::~Comptable() {
    compteur--;
    cout << " -- il y a " << compteur
         << " objets." << endl;
}

void fonction()
{
    Comptable u, v;
}

int main(int argc, char ** argv)
{
    Comptable a;
    fonction();
    Comptable b;
    return 0;
}
```

Valeur par défaut

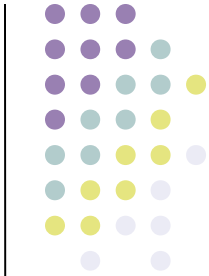


```
void initialise(double a = 0.0, double b = 0.0)
{
    x = a;
    y = b;
}
```

Que fait :

```
Point p(3.0, 4.0);

.....
   p.initialise();

.....
```

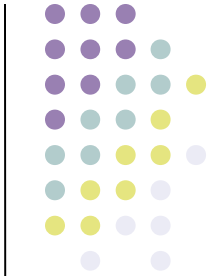
Sur-définition de fonction



- On peut définir plusieurs fonctions avec les mêmes noms.
- Pour les différencier, il faut que les arguments soient différents en nombre ou en types.

```
void initialise(double a, double b)
void initialise(point & p);
void initialise(point *p);
```

Classe ensemble de points



```
class Points {

    private :

        Point *points;
        int nbe;
        int capacite;

    public :

        Points(int cap);
       ~Points();

        bool present(Point &p);
        void affiche();
        bool inserer(Point &p);
        int supprimer(Point &p);

};
```