

# Séance 3

## Classe (2)

[Vocabulaire objet](#)

[Ecriture de fonctions dans la déclaration de la classe](#)

[Place mémoire occupée par un objet](#)

[Constructeur par copie](#)

[Objet membre](#)

[Constructeur par défaut](#)

[Découpage en modules](#)

# Vocabulaire objet

Un objet : une **instance de classe**.

Une fonction membre : une **méthode**.

Une donnée membre : une **propriété** ou un **attribut** .

Mettre des données membres en partie privée : **l'encapsulation**.

# Ecriture de fonctions dans la déclaration de la classe

```
class Date
{
public :
    Date (int j, int m, int a);
    void afficher() const {
        cout << jour << ...;
    }
    ...
private :
    int jour, mois, annee;
};
```

Toute fonction peut être écrite dans la déclaration de la classe.

Pour l'appel, pas de changement :  
Date d1(2, 9, 2023);  
d1.afficher();

# Place mémoire occupée par un objet

```
struct Date
{
    int jour, mois, annee;
};
```

Date d1; en mémoire, structure d1 :

jour
mois
annee

```
class Date
{
public :
    Date (int j, int m, int a);
    void afficher() const {
        cout << jour << ...;
    }
    ...
private :
    int jour, mois, annee;
};
```

Date d1(2, 9, 2023); en mémoire, objet d1 :

jour
mois
annee

En mémoire, un objet contient seulement les données.

# Constructeur par copie

```
Date d1(6, 4, 2023);
```

Appel du constructeur Date(int, int, int) déclaré dans la classe.

```
Date d2 = d1; d2 est initialisé avec 6/4/2023
```

ou

```
Date d2(d1); écriture équivalente
```

Appel du constructeur par copie qui :

- est créé automatiquement par le compilateur,
- copie les champs de d1 (jour, mois, année) dans les champs de d2

*Pour certaines classes, le constructeur par copie créé par le compilateur doit être réécrit (voir exercice classe String).*


# Objet membre

```
class Date {  
public :  
    Date(int j, int m, int a);  
    ...  
};  
  
class Individu  
{  
public :  
    Individu(const char*no,  
             int jn, int mn, int an);  
    ...  
private :  
    char nom[20];  
    Date date_naiss; // objet membre  
};
```

Constructeur de Individu doit appeler le constructeur de l'objet membre date\_naiss.

Dans une zone appelée liste d'initialisation.

```
Individu::Individu (const char* no, int jn, int mn, int an)  
    : date_naiss (jn, mn, an)  
{  
    strcpy(nom, no);  
}
```



paramètres du  
constructeur de Date

Si plusieurs objets membres :

```
Individu::Individu(...)  
    : date_naiss(...), autre_objet(...)
```

# Constructeur par défaut

Constructeur par défaut : un **constructeur sans paramètre**.  
Pas d'obligation de définir un constructeur par défaut.

Objet membre :

```
class Individu {  
...  
private :  
    ...  
    Date date_naiss;  
};
```

```
Individu::Individu(const char *no, ...)  
{  
    ...  
}
```

← Si pas d'appel au constructeur de date\_naiss :  
appel automatique du constructeur par défaut de Date  
erreur de compilation si pas de constructeur par défaut

Tableau d'objets :

```
Date tab[100];
```

Les objets du tableau sont initialisés avec le constructeur par défaut de Date.  
Erreur de compilation si pas de constructeur par défaut.

# Découpage en modules

Une classe = un module    classe Date => Date.h Date.cpp

```
// Date .h
#pragma once
class Date
{
public :
    Date (int j, int m, int a);
    void afficher() const;
    ...
private :
    int jour, mois, annee;
};
```

*pragma once => pas d'erreur  
en cas d'inclusions multiples*

```
// Date.cpp
#include <iostream>
...
#include "Date.h"
using namespace std;

Date::Date(int j, int m, int a) {
    ...
}
void Date::afficher() const {
    ...
}
...
```

```
// autre_module.cpp, utilisant Date
#include <iostream>
...
#include "Date.h"
using namespace std;
...
```