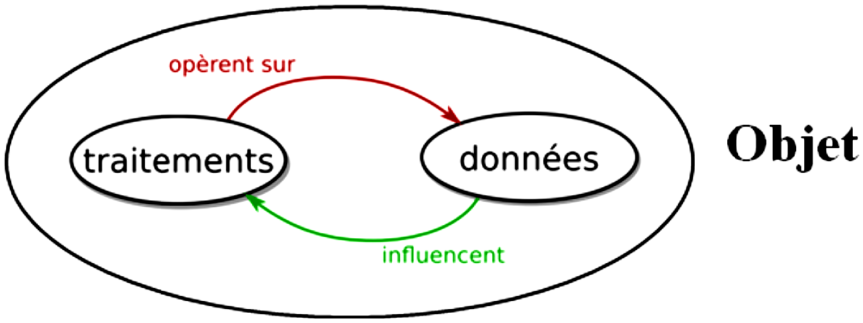
Chapitre V : Classes et objets

V.1 Concept objet

La **POO** permet d'améliorer [9]:

- La robustesse (changement de spécification)
- La modularité
- Lisibilité

} => la maintenabilité



POO : quatre concepts de base
Un des objectifs principaux de la notion d'objet : organiser des programmes complexes grâce aux notions :

- d'encapsulation
- d'abstraction
- d'héritage
- et de polymorphisme

V.2 Notion d'objet (instance):

Un ***objet*** représente une entité individuelle et identifiable, réelle ou abstraite, avec un rôle bien défini dans le domaine du problème, chaque objet peut être caractérisé par une identité, des états significatifs et par un comportement.

Objet = Etat + Comportement + Identité

V.2.1 Etat

- Un attribut est une caractéristique, qualité ou trait intrinsèque qui contribue à faire unique un objet
- L'état d'un objet comprend les propriétés statiques (attributs) et les valeurs de ces attributs qui peuvent être statiques ou dynamiques

Exemple:

Les attributs de l'objet point en deux dimensions sont les coordonnées x et y

p1: Point
x_ = 34.5
y_ = 18.8

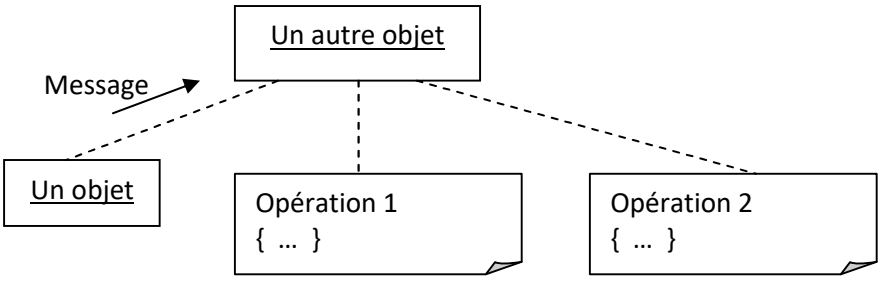
p2: Point
x_ = 0.0
y_ = 0.0

V.2.2 Comportement

Le comportement d'un objet se défini par l'ensemble des opérations qu'il peut exécuter en réaction aux messages envoyés (un message = demande d'exécution d'une opération) par les autres objets.

Exemple:

Un point peut être déplacé, tourné autour d'un autre point, etc.



V.2.3 Identité

Propriété d'un objet qui permet de le distinguer des autres objets de la même classe.

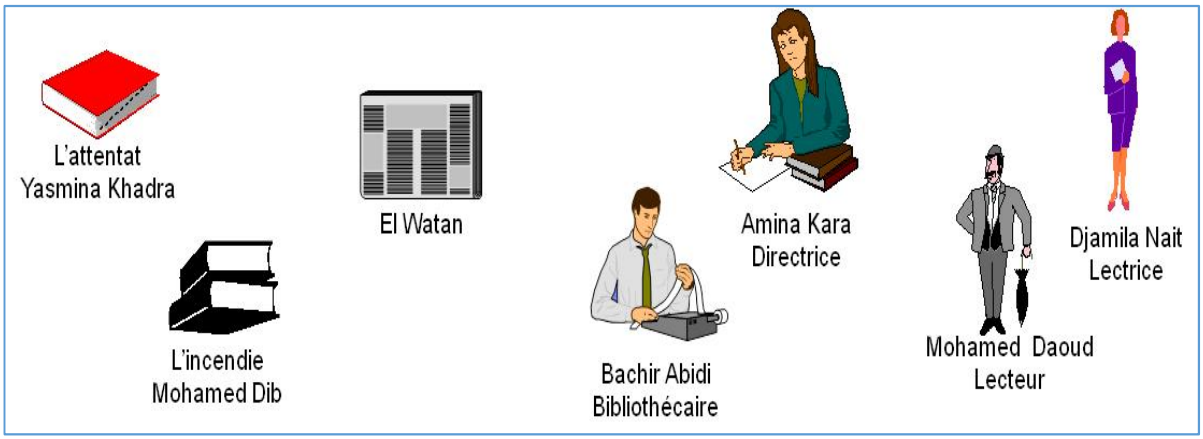
V.3 Notion de classe:

Lorsque des objets ont les mêmes attributs et comportements: ils sont regroupés dans une famille appelée**: Classe.**

Une classe est un modèle à partir duquel on peut générer un ensemble d'objets partageant des attributs et des méthodes communes. Les objets appartenant à celle-ci sont les **instances** de cette classe c.à.d. que **L'instanciation** est la création d'un objet d'une classe.

Exemple: Gestion d'une bibliothèque

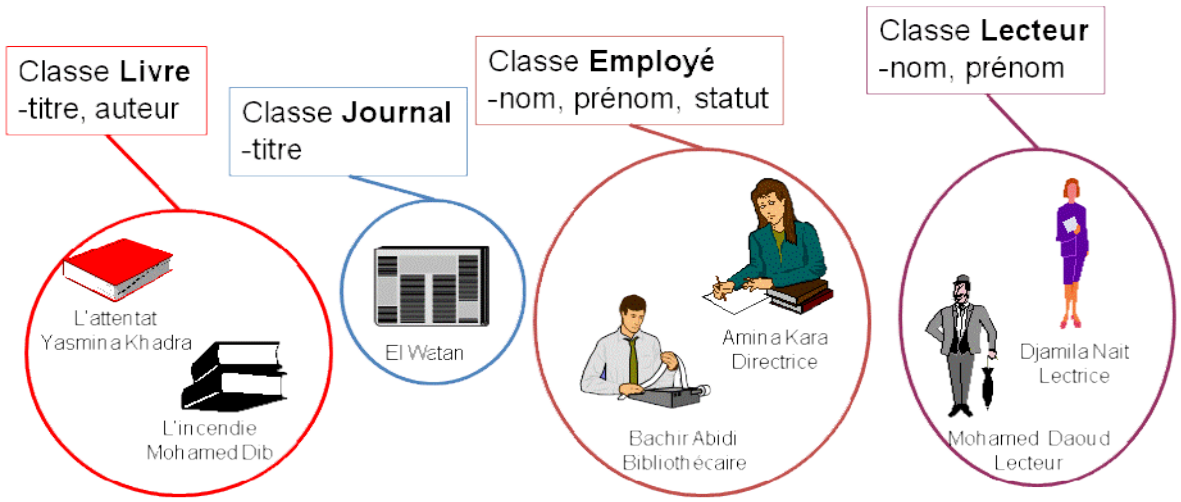
Objets



Classes

Des objets similaires peuvent être informatiquement décrits par une même abstraction: une **classe**

- même structure de données et méthodes de traitement
- valeurs différentes pour chaque objet



V.3.1 Déclaration

Une classe est une structure. Sauf que:

- Le mot réservé **class** remplace **struct**
- Certains champs sont des **fonctions**.

En C++, une classe se définit grâce au mot clef "**class**" suivi du nom de celle-ci. Ce nom commence généralement par une majuscule. A la suite, les données membres et les méthodes sont déclarées entre deux accolades. La définition de la classe se termine obligatoirement par un point virgule:

Syntaxe:

```
class Nom
{
   //données membres et fonctions membres
};
```

Prenons un exemple concret et simple: l'écriture d'une classe **Point**:

En **C**, nous aurions fait une structure comme suit:

```
struct Point
{
  int x; // Abscisse du point
  int y; // Ordonnée
};
```

La déclaration précédente fonctionne parfaitement en C++. Mais nous aimerions rajouter des fonctions qui sont fortement liées à ces données, comme l'affichage d'un point, son déplacement, etc.

Voici une solution en **C++**:

```
#include <iostream.h>
class Point
{  public : // voici les attributs
   int x;
   int y;
  // voici les méthodes
  void afficher()
  { cout <<x<<','<<y<<endl; }
```

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

Commentaires:

- Les champs x et y sont les **attributs** de classes, appelés aussi variables membres.
- Les fonctions **afficher**, **placer** et **deplace** sont appelées méthodes ou fonctions membres.
- Le terme **public** signifie que tous les membres qui suivent (données comme méthodes) sont accessibles de l'extérieur de la classe. Nous verrons les différentes possibilités plus tard.

V.3.2 Utilisation de la classe

V.3.2.1 Instance de classe

Une classe étant définie, il est possible de créer des objets (des variables ou des constantes) de ce type.
On les appelle alors des ***objets*** de la classe ou des ***instances*** de la classe on déclare un "objet" d'un type classe donné en faisant précéder son nom de celui de la classe, comme dans l'instruction suivante qui déclare deux objets a et b de type point.

Exemple :

```
point a, b ;
```

V.3.2.2 Accès au membre d'un objet

- ***Cas de déclaration statique:***

```
#include <iostream>
#include "Points.cpp"
void main()
{
  Point p;
  p.x=10;
  p.y=20;
  p.afficher();
  p.placer(1,5);
  p.afficher();
}
```

- ***Cas d'allocation Dynamique:***

```
#include <iostream>
#include "Points.cpp"
void main()
{
   Point *p=new Point;
```

```
 p->x=10;
 p->y=20;
 p->afficher();
 p->placer(1,5);
 p->afficher();
}
```

V.3.3 Opérateurs de résolution de portée

Les méthodes de la classe ***Point*** sont implémentées dans la classe même. Ceci fonctionne très bien, mais devient bien entendu assez lourd lorsque le code est plus long. C'est pourquoi il vaut mieux placer la déclaration seulement, au sein de la classe. Le code devient alors :

```
#include <iostream>
class Point
{ public :
  int x;
  int y;
  void placer(int a, int b);
  void deplace(int a, int b);
  void affiche();
};
void Point::placer(int a, int b)
{   x = a;
    y = b;
}
void Point::deplace(int a, int b)
{  x += a;
   y += b;
}
void Point::affiche()
{
  cout << x << ", " << y << endl;
}
void main()
{  Point p;
   p.placer (3,4);
   p.affiche();
   p.deplace(4,6);
   p.affiche();
}
```

✍ On remarque la présence du "**Point::**" (**:: opérateur de résolution de portée)** qui signifie que la fonction est en faite une méthode de la classe *Point*. Le reste est complètement identique.
La seule différence entre les 2 manières vient du fait qu'on dit que les fonctions de la première (dont l'implémentation est faite dans la classe), sont "**inline**". Ceci signifie que chaque appel à la méthode sera remplacé dans l'exécutable, par la méthode en elle-même. D'où un gain de temps certain, mais une augmentation de la taille du fichier en sortie.

V.3.4 Objets transmis en argument d'une fonction membre

Nous pouvons maintenant imaginer vouloir comparer deux points, afin de savoir s'ils sont égaux. Pour cela, nous allons mettre en œuvre une méthode "**Coincide**" qui renvoie "true" lorsque les coordonnées des deux points sont égales :

```
class Points
{  public :
   int x;
   int y;
   bool Coincide(Point p)
   {  if( (p.x==x) && (p.y==y) )
         return true;
     else
        return false;
   }
} ;
```

Cette partie de programme fonctionne parfaitement, mais elle possède un inconvénient majeur : le **passage de paramètre par valeur**, ce qui implique une "duplication" de l'objet d'origine. Cela n'est bien sûr pas très efficace.

La solution qui vous vient à l'esprit dans un premier temps est probablement de passer par un pointeur. Cette solution est possible, mais n'est pas la meilleure, dans la mesure où nous savons fort bien que ces pointeurs sont toujours sources d'erreurs (lorsqu'ils sont non initialisés, par exemple).

La vraie solution offerte par le C++ est de passer par des références. Avec ce type de passage de paramètre, aucune erreur est possible puisque l'objet à passer doit déjà exister (être instancier). En plus, les références offrent une simplification d'écriture, par rapport aux pointeurs :

```
#include <iostream>
class Points
{  public :
   int x;
   int y;
  bool Coincide(Point &p)
  { if( (p.x==x) && (p.y==y) )
     return true;
   else
     return false;
  }
};
void main()
{
Points p;
Points pp;
pp.x=0;pp.y=1;
p.x=0; p.y=1;
if( p.Coincide(pp) )
```

```
cout << "p et pp coincident !" << endl;
if( pp.Coincide(p) )
cout << "pp et p coincident !" << endl;
}
```

V.3.5 Fonction membre dont le type est le même que la classe:

Supposons qu'on a besoin de déterminer le point milieu entre deux points, donc on peut ajouter à la classe point une méthode qui permet de fournir ce point, cette méthode accepte comme argument un point P et elle doit produire le point milieu entre P et le point représenter par la classe d'où le résultat (type de retour) est un point.

Exemple:

```
class Points
{ public :
int x;
int y;
void init(int a, int b)
{ x=a;
  y=b;
}
Points Milieu(Point &p)
{  Point M;
   M.x=(x+P.x)/2 ;
   M.y=(y+P.y)/2 ;
   return M ;
}
};
void main()
{
Points P1, P2,M;
P1.init(1,3);
P2.init(4,4);
M=P1.Milieu(P2);
//ou bien M=P2.Milieu(P1);
}
```

V.3.6 Donnée membre de type structure (struct)

On veut définir une classe permettant de créer des 'personne'; une personne est caractérisée par son nom, prénom, numéro CIN et date de naissance.
Or la date de naissance est un champ composé par jour, mois et année, donc il faut le regroupé sous un seul nom a l'aide d'une structure Date:

1ère méthode : La structure est définie en dehors de la classe

```
typedef struct Date
{ int j,m,a;
} ;
class Etudiant
{ public :
   char Nom[50] ;
```

```
   char Prenom[50] ;
   int  NCIN ;
Date DN
//…
};
```

- La structure Date est connue dans tout le programme.

```
void main()
{  Etudiant E ;
   Date D ;
   D.j=1 ; D.m=1 ; D.a=2008 ;
   strcpy(E.Nom, "ali") ;
   E.DN=D ;
};
```

2ème méthode : La structure est définie à l'intérieur de la classe

```
class Etudiant
{ typedef struct Date
  { int j, m, a;  } ;
  public :
  char Nom[30] ;
  char Prenom[30] ;
  int  NCIN ;
  Date DN ;
//…
};
```

- La structure Date est connue uniquement à l'intérieur de la classe Etudiant.

```
void main()
{  Etudiant E ;
   Date D ; // erreur de compilation
   E.DN.j=1 ;
   D.DN.m=1 ;
   D.DN.a=2008 ;
   strcpy(E.Nom, "Nabil") ;
//…
};
```

V.3.7 Classe composée par d'autres classes:

Ici on parle **d'objets membres**: lorsqu'un attribut de la classe est lui même de type une autre classe.

Exemple:

```
class Date
{ public:
   int j, m, a ;
   void afficher()
    { cout<<j<< "/"<<m<<"/"<<a<<endl ;}
   void init(int jj, int mm, int aa)
    {  j=jj ;  m=mm ; a=aa ; }
};
```

```
class Etudiant
{   public :
     char Nom[30] ;
     char Prenom[30] ;
     int  NCIN ;
     Date DN ; // DN est une instance de Date
    void afficher()
    {
      cout<< "Nom="<<Nom<<endl ;
      cout<< "Prénom="<<Prenom<<endl ;
      cout<< "numéro de CIN="<<NCIN<<endl ;
      cout<< "Date de naissance= " ;
      DN.afficher() ;
    }
//...
};
```

- Déclaration de la classe Date dans Etudiant :

```
class Etudiant
{
 class Date
 {public :
    int j;
    int m;
    int a;
    void afficher()
    { cout<<j<< "/"<<m<<"/"<<a<<endl ;}
    void init(int jj, int mm, int aa)
    { j=jj ; m=mm ; a=aa ; }
  };
 public :
   char Nom[30] ;
   char Prenom[30] ;
   int NCIN ;
   Date DN ; // DN est une instance de Date
   void afficher()
   {  cout<< "Nom="<<Nom<<endl ;
      cout<< "Prénom="<<Prenom<<endl ;
      cout<< "numéro de CIN="<<NCIN<<endl ;
      cout<< "Date de naissance= " ;
      DN.afficher() ;
   }
//...
};
```

✍ Date est utilisable uniquement dans la classe Etudiant.