9/ Programmation orientée objet : POO

Historique : → Simula

- Java
- C++
- Python
- Delphi
- HTML 5

Principe :

POO → analogie avec les objets du monde réel

Exemple : Carte bancaire

- → solde
- → des opérations dépots, retraits
- des visualisation.

⇒ Il n'est pas nécessaire de connaître le cheminement des opérations.

⇒ C'est un assemblage d'objets.

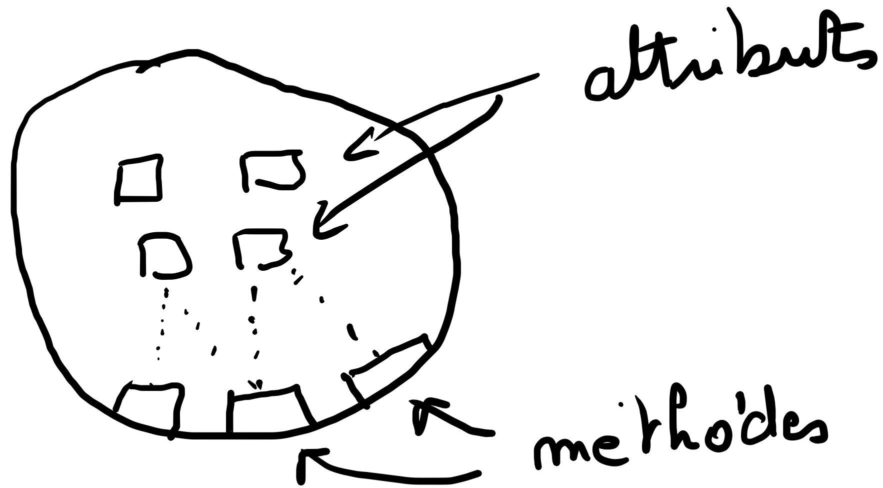
Objet :

→ C'est une entité qui possède des caractéristiques auxquelles on associe un comportement en fonction de tâches qu'il est capable d'accomplir

→ Un objet est caractérisé par ses attributs et des procédures qui interagissent sur lui (méthodes)

Les attributs de l'objet ne sont pas connus du monde extérieur, seules les méthodes sont accessibles.

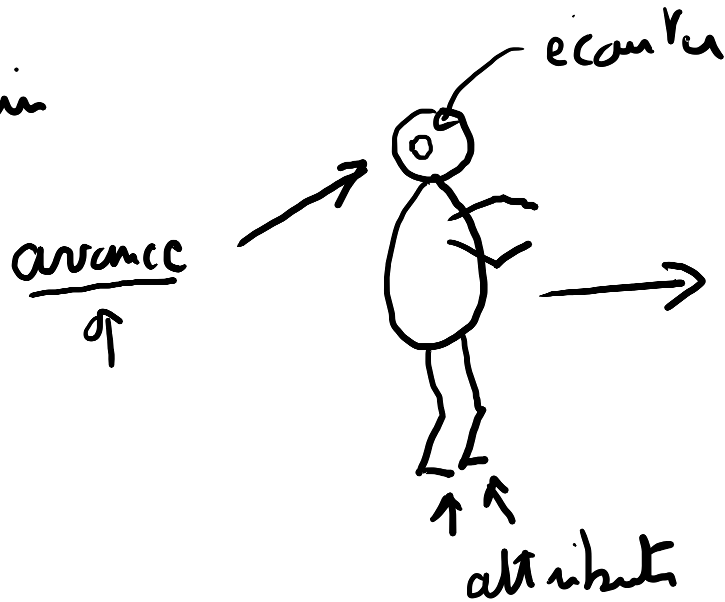
objet



méthodes sont les interfaces de l'objet.

Faire appel à une méthode => envoie d'un <u>message</u> à l'objet.

// l'humain



Exemple :

point

- entier x
- entier y

) attributs

→ initialiser
→ déplacer
→ afficher

) méthodes

Voiture :

- marque
- puissance
- nb portes
- couleur

} attributs

→ construire
→ rouler
→ freiner

} méthodes

Méthodes + **attributs** = **objet**

Le fait qu'il n'est pas possible d'agir directement sur les attributs de l'objet :

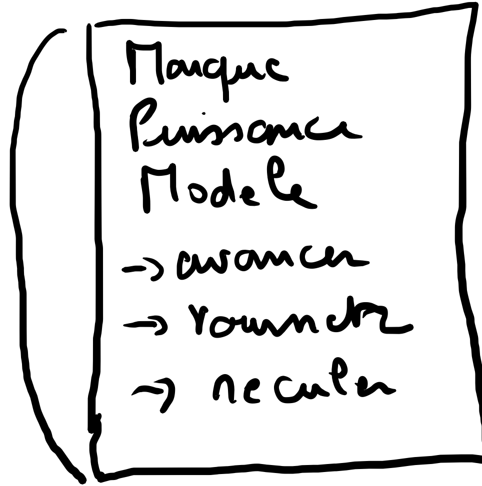
l'<u>encapsulation</u>.

Les classes :

On regroupe les objets aux caractéristiques et comportements identiques dans une classe.

Définition : attributs + méthodes identiques.

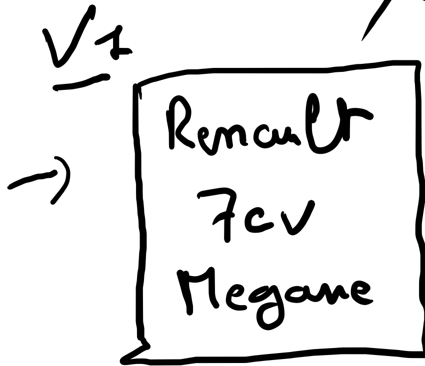
Classe Voiture



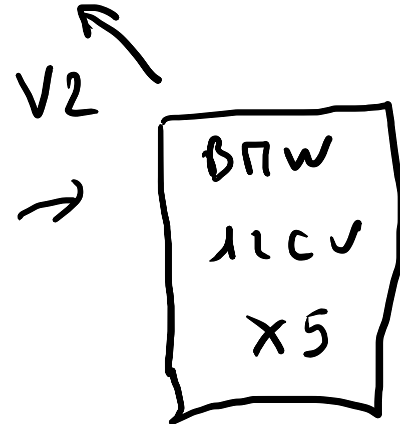
V1 et V2 sont des exemplaires de la classe voiture

Ce sont des **instances** de voiture

V1

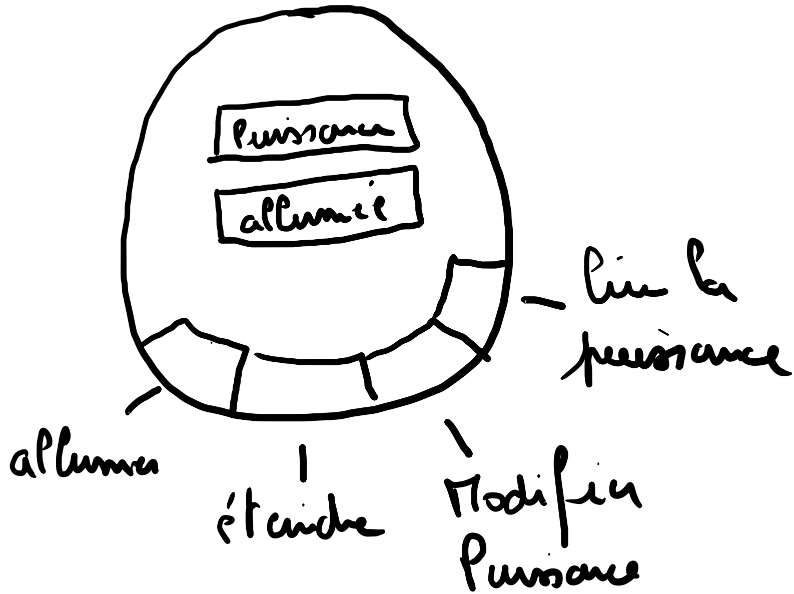


V2



En C++ :

Pompe



Creation de la classe

```
Class Pompe
{ private :
      int puissance;
      bool allumee;
  public :
  ->  void modifPuissance(int p);
  ->  int lirePuissance();
  ->  void allumer();
  ->  void eteindre();
};    Pompe();
```

Instanciation :

Il y a instanciation lorsque l'objet est définie, on dit que l'objet est une instance de sa classe.

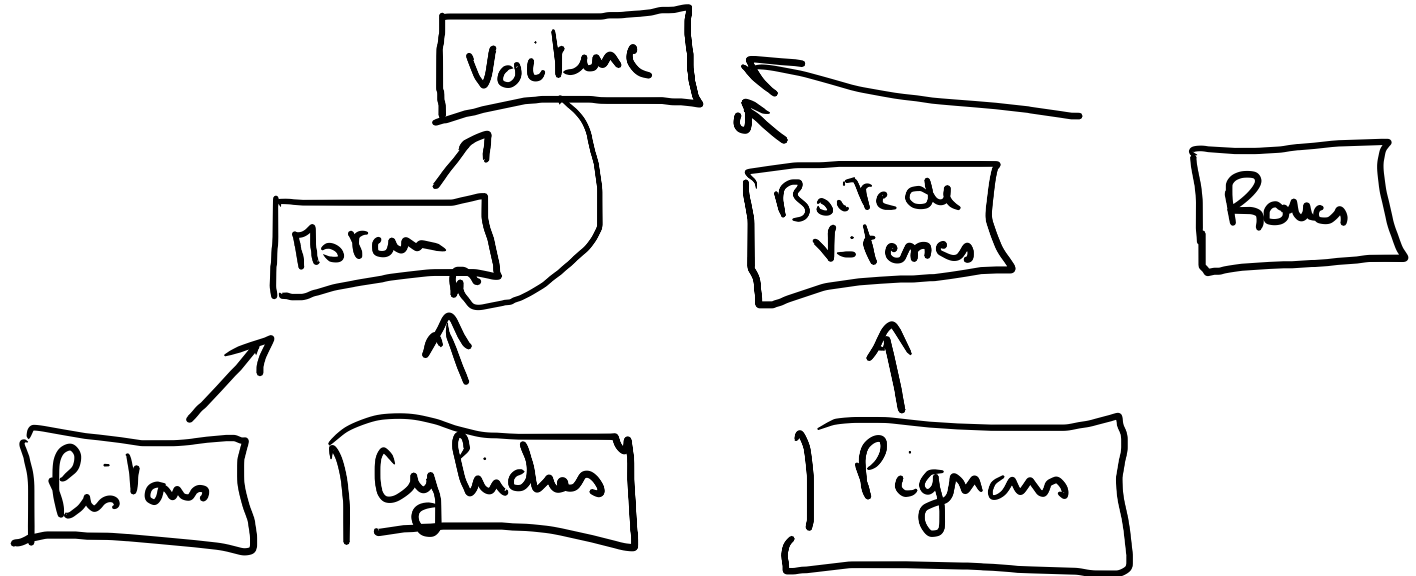
⇒ Pompe $\underline{p_1}$, $\underline{p_2}$;

↑ ↑

objets

Aggrégation :

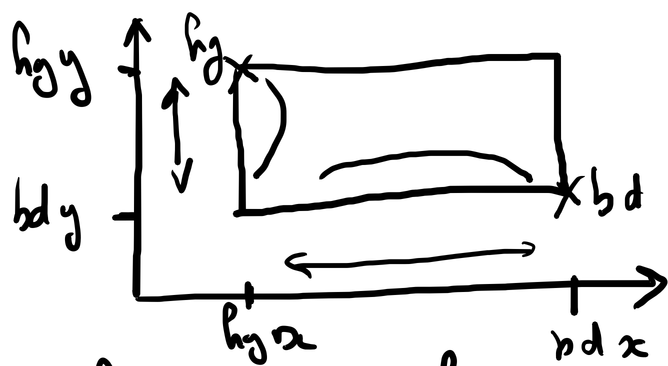
Un objet peut être composé de sous objets
=> définition d'une hiérarchie



Attention: Une méthode ne peut accéder qu'à ses attributs, et pas à ceux de ses sous-objets, il faut alors utiliser les méthodes de ses sous-objets.

Exemple :

```
class point
{ private :
      int x;
      int y;
  public :
      void init(int px, int py);
      int lireX();
      int lireY();
};
```




```
class rectangle
{ private :
      point hg;
      point bd;
      int epaisseur;
  public :
      void init(int hgx, int hgy,
                int bdx, int bdy, int ep);
      int surface();
};
```

Définition des méthodes:

Elles décrivent les actions à entreprendre sur les objets.

Exemple de la classe Lampe

```
void   Lampe :: allumer()
{  allumee = true;
}
void   Lampe :: eteindre()
{   allumee = false;
}
```

:: opérateur de résolution de portée

```
int   lampe :: lirePuissance()
{
    return puissance;
}
void lampe :: modifPuissance(int p)
{
    puissance = p;
}
lampe :: lampe ()   <—— constructeur
{
  puissance = 0;
  allumee = false;
}
```

Messages :

Pour communiquer avec un objet, on lui envoie un message pour l'intermédiaire de ses méthodes.

Message → objet

C++

nom de l'objet . méthode()

Tout message comporte 3 éléments
- le destinataire (objet)
- le sélecteur (méthode)
- arguments

<u>Exemple</u> :

```
Lampe l1, l2;   ← instancie 2 objets de la classe Lampe
l1.modifPuissance(100);
l2.modifPuissance(l1.lirePuissance());
l1.allumer();
l2.allumer();
```

Exemple rectangle (aggrégation)

```
void point::init(int px, int py)
{   x = px;
    y = py;
}
int point::lireX()
{ return x;
}
int point::lireY()
{ return y;
}
```

```
void rectangle::init(int hgx, int hgy, int bdx,
                     int bdy, int ep)
{
    hg.init(hgx, hgy);
    bd.init(bdx, bdy);
    epaisseur = ep;
}
int rectangle::surface()
{
  return((bd.lireX()-hg.lireX())*(hg.lireY()-bd.lireY()))
}
```

Le constructeur :

C'est une méthode appelée automatiquement à chaque fois que l'on crée un objet (instance) basé sur une classe

→ permet de s'affranchir de toute initialisation d'attributs

Règles à respecter :

- le nom du constructeur est le même que le nom de la classe dans lequel il est défini
- il n'a pas de type de retour (pas de void)
- il peut avoir des arguments.

Classe

```
class Pompe
{ private :
      ═══
   public :
      void modifPuissance(int p);
      ═══
      Pompe();
}
```

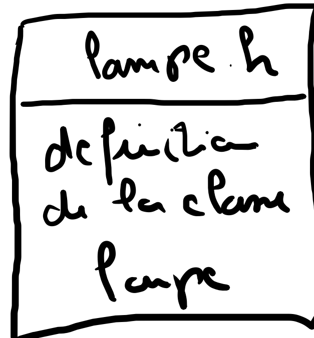
Méthodes

```
Pompe::Pompe()
{ puissance = 0; allumee = false; }
```

Construction d'un exécutable :

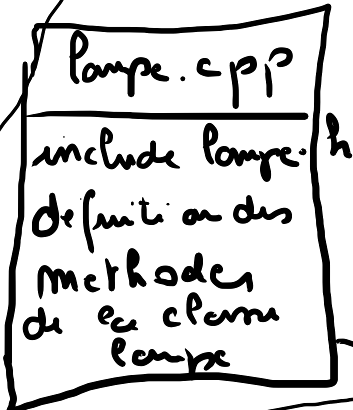
Création des classes

fichier .h

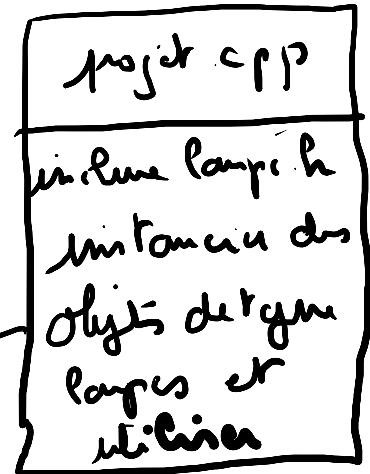


projet.cpp

fichier.cpp



Exé



Exemple :

1 Créer le fichier des classes → **point.h**

```
class point
{ private:
     int x;
     int y;
  public:
     point();
     void affecte(int x1, int y1);
     void deplace(int dx, int dy);
     void affiche();
};
```

2 Créer le fichier des méthodes **point.cpp**

```
#include <iostream.h>
#include "point.h"

point::point()
{   x=0;
    y=0;
}
void point::affecte(int x1, int y1)
{   x=x1;
    y=y1;
}
```

```
void point :: deplace (int dx, int dy)
{   x += dx;
    y += dy;
}
void point :: affiche ()
{
cout << " le point se trouve en " << x
     << "  " << y << " \n";
}
```

3 Creation du projet deplacePoint.cpp

```
#include "point.h"

int main()
{ point p;
  p.affiche();
  p.affecte(3,4);
  p.affiche();
  p.deplace(8,12);
  p.affiche();
}
```

x=0 y=0

le point se trouve en 0 0

cin >> x
cin >> y

3 4

11 16