3. PROGRAMMATION ORIENTEE OBJETS ?

➢ Programmation procédurale (en C par exemple)

Paradigme : Choisissez les ***traitements*** dont vous avez besoin et utilisez les meilleurs algorithmes

Programme : ensemble séquentiel de sous-programmes (des fonctions)

Caractéristiques :

Séparation complète entre code et données manipulées
Communication entre sous-programmes par paramètres et variables globales
Usage permanent des conditions portant sur le type des données manipulées

➢ Programmation par objets (en C++ par exemple)

Paradigme : Choisissez les ***données*** *et les* ***types*** dont vous avez besoin.

- Identifiez les entités que manipulera le programme **=> objets**
- Déduire les concepts (types) dont ces objets sont des instances **=> classes**
- Pour chaque classe :
  - Précisez les **attributs**, qui définissent l'état des objets
  - Définissez un ensemble complet **d'opérations** sur ces entités.
  - Identifiez l'interface : séparez ce qui est interne (privé) et ce qui est visible de l'extérieur (public) **=> interface publique et encapsulation**
- Rendez explicites les points communs entre ces classes et les relations entre classes **=> hiérarchie de classes et composition entre classes.**
- Organisez les données et leur sécurité (dépendances)

Programme : ensemble d'objets, caractérisés par leur état et par leur savoir faire (les opérations qu'ils connaissent), qui s'échangent des messages.

CONCEPTS ET MOTS CLES DE LA POO (1)

NOTION DE CLASSE ET D'OBJET

➢ Qu'est-ce qu'un objet ?

En POO, on utilise des **objets** pour représenter toute entité identifiable :
- une chose tangible ex: une ville, un étudiant, un bouton sur l'écran...
- une chose conceptuelle ex: une date, une réunion, une collection...

Un objet existe en mémoire et est caractérisé par :
- Son **état**, défini par la valeur de chacun de ses **attributs**.
  Chaque objet a un état qui lui est propre.
  Exemple: l'objet « *maVoiture* » a un attribut nommé 'couleur' qui vaut vert, un attribut *position* qui indique la position de la voiture sur une carte, un attribut *moteur*...
  L'objet *laVoitureDePaul* a une autre couleur, une autre position, un autre moteur.

- Son **comportement** : ce qu'il sait faire, ce qu'on peut faire avec. Défini par des **méthodes** (ou *fonctions membres*, ou *opérations*).
  Exemple: l'objet « *maVoiture* » réagit au message *démarrer()* en allumant le moteur son attribut moteur.

➢ Qu'est-ce qu' une classe ? un concept / un type / un moule

Un objet est créé à partir d'un « moule », ou « type » : sa **classe**.

Instancier une classe, c'est créer un objet qui suit le « moule ».
Tout objet est une **instance** d'une et une seule classe.

Une classe est donc une abstraction qui représente un ensemble d'objets de même nature (mêmes types d'attributs et mêmes méthodes). Une classe est donc un type d'objets.

Exemple : les objets *maVoiture* et *laVoitureDePaul* sont deux instances de la classe *Voiture*. Le type de *maVoiture* et *laVoitureDePaul* est *Voiture.*

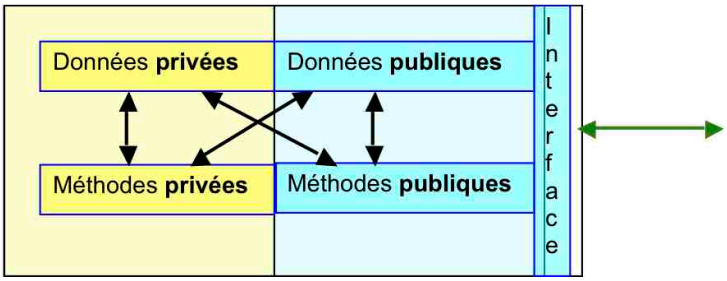
Une classe définit les membres qu'auront toutes ses instances :
- Les ***attributs*** et leur type (eg : toute « `Voiture` » a une « couleur » une « position » et un « `Moteur` »)
- Les ***opérations*** (ou *fonctions membres* ou *méthodes*) qui manipulent ces attributs (eg : toute Voiture a une opération *démarrer()*, qui allume le moteur).

CONCEPTS ET MOTS CLES DE LA POO (2)

➢ Encapsulation, interface publique

La POO permet de séparer dans les objets (et dans les classes) :
- Une partie visible, publique, qui définit la façon dont l'objet (la classe) s'utilise depuis le reste du programme.
- Une partie encapasulée, privée, invisible depuis l'extérieur.

L'interface publique d'un objet (ou d'une classe) est l'ensemble de ses attributs et méthodes publiques, accessibles de l'extérieur.
La **visibilité** d'un attribut ou d'une méthode détermine les conditions d'accès à cet attribut ou d'utilisation de cette méthode.



Exemple :
En interne, dans une classe `Montre` on peut choisir de stocker l'heure courante *soit* en nombre de millisecondes écoulées depuis 1970 *soit* avec trois attributs : heure, minute, seconde. Ceci relève d'un détail d'implémentation, qui sera privé.

Quel que soit ce choix « encapsulé » dans la classe, l'interface publique de `Montre` permettra d'accéder à l'heure courante de toutes manières utiles.

➢ Composition entre objets et classes

Possibilité de définir des objets composites, fabriqués à partir d'autres objets.

"composé-de" (lien *a-un* ou *a-des*)
relation contenant, champ de la classe ou de la structure

Exemple : Un chien a des pattes, une gueule, etc.
- ⇨ Un objet instance de `Chien` possède 4 objets instances de `Patte`, et un objet instance de `Gueule`, etc.
- ⇨ La classe `Chien` est composée avec les classes `Patte` et `Gueule`.

CONCEPTS ET MOTS CLES DE LA POO (3)

➢ Héritage

Une classe X peut servir de base pour définir une nouvelle classe Y.

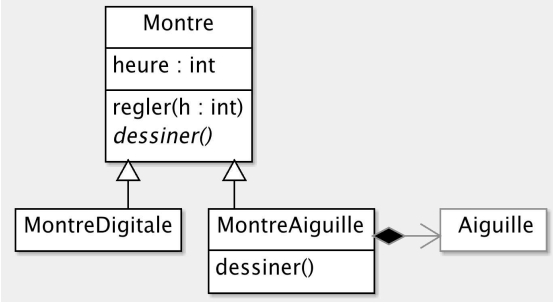
⟶▷ **"sorte-de"** (lien *est-un-type-particulier-de*)
relation hiérarchique père-fils, notion d'héritage

On réutilise ce qui a été déjà fait (classe X), en ajoutant de nouveaux membres (extension) et/ou en modifiant les méthodes de la classe X (redéfinition).

Exemple :
Intuitivement, une « montre digitale » et une « montre à aiguille » sont deux types de « montres ». En programmation objet, on fait hériter les classe `MontreDigitale` et `MontreAiguille` d'une classe `Montre`.

Tout ce qui est commun à toutes les montres (e.g : le fait qu'elles ont l'heure, qu'on peut les regler(), ...) :

- est défini dans la classe `Montre` (écrit pour tous les types de montres)
- est *hérité* dans les classes `MontreDigitale` et `MontreAiguille`.



Un objet instance de `MontreAiguille` :

- **hérite** des propriétés (attributs et méthodes) de la classe `Montre`
- y ajoute ses propres spécificités (par exemple, deux instances d'une classe `Aiguille`).
- peut redéfinir certains comportement hérités : `dessiner()` par exemple.

On dit que :

- Les classes `MontreDigitale` et `MontreAiguille` **héritent de** ou **spécialisent** ou **étendent** ou sont des **classes filles** ou des **sous-classes** de la classe `Montre`.
- La classe `Montre` **généralise**, ou est une **classe mère**, ou est une **super-classe** des classes `MontreDigirale` et `MontreAiguille`.

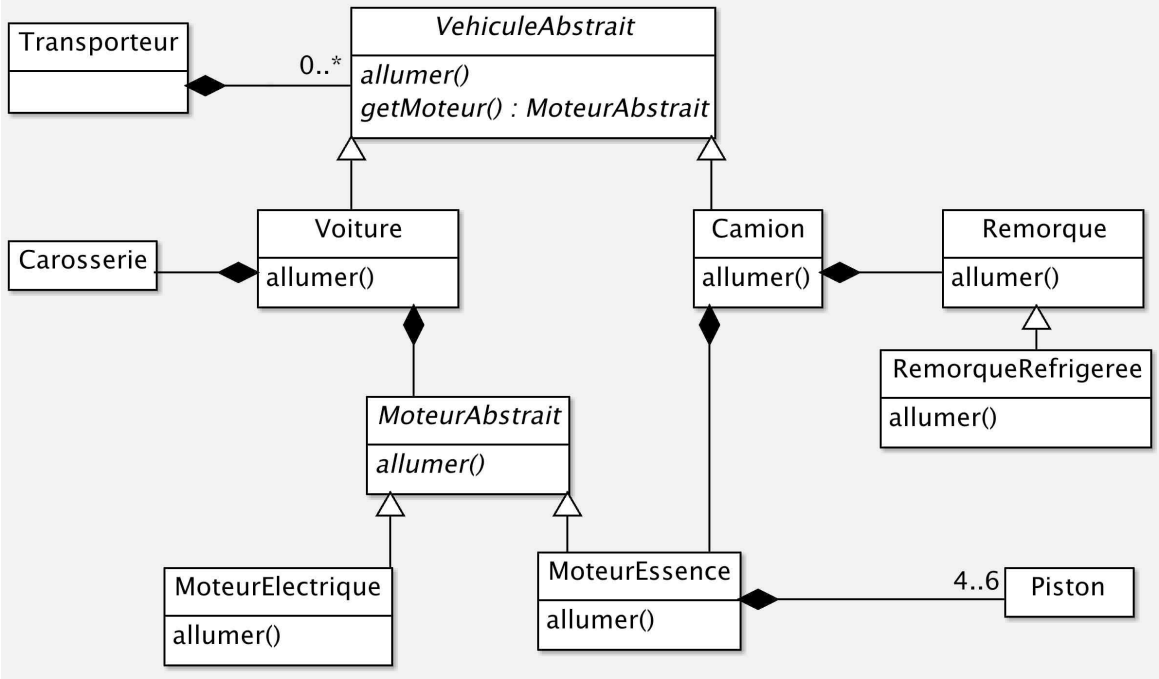
Intérêts:

- factorisation du code commun à toutes les montres dans `Montre`
- **Polymorphisme** : possibilité de manipuler à la fois des objets intances de `MontreDigitale` ou `MontreAiguille` en tant que `Montre` (sans s'intéresser à ce qu'elles sont véritablement)

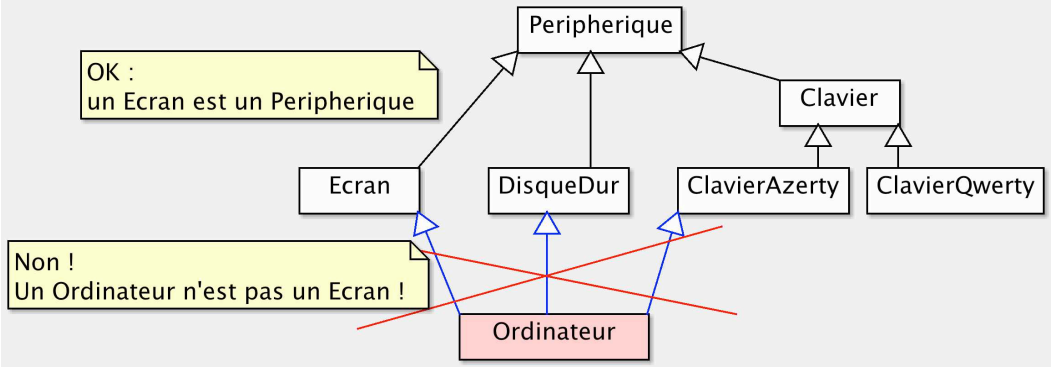
CONCEPTS ET MOTS CLES DE LA POO (4)

➢ Exemple

Une voiture est un véhicule. Un camion est aussi un véhicule.
Une voiture est composée d'une carrosserie et d'un moteur.
Le moteur peut être électrique ou à essence.
Un moteur à essence est composé de pistons. Il y a 4 ou 6 pistons.
Un camion a un moteur à essence et une remorque. Il existe des remorques réfrigérées.
Tous les véhicules peuvent être allumés.
Allumer une voiture, c'est allumer son moteur.
Allumer un camion, c'est allumer son moteur et sa remorque, si elle est réfrigérée.
Un transporteur a plusieurs véhicules.



➢ Ne pas confondre héritage et composition !



Un ordinateur *n'est pas* une sorte de disque ou de clavier. Mais il est *composé* d'un clavier, d'un type particulier (AZERTY ou QWERTY) , d'un disque dur ...

INTERETS DE LA POO

L'approche objet :

- permet la conception et l'implantation de logiciels *plus fiables* et *plus aisés à maintenir*.

- accompagne toutes les phases de la vie d'un programme : analyse du problème, conception, implantation.
  Les concepts objets se prêtent bien à l'échange entre concepteurs et développeurs, entre développeurs, entre utilisateurs et concepteurs...

- s'appuie naturellement sur des représentations graphiques
  L'approche objet est « naturelle ».
  UML (Unified Modeling Language) : un standard graphique pour la POO.

- permet de préciser et manipuler plus aisément les relations entre les concepts du programme.
  Exemple : exprimer immédiatement des relations est-un *ou* est-composé-de *et leurs variantes.*

- permet d'exprimer et de maitriser des architectures logicielles complexes.

- permet d'organiser ce qui relève de la partie publique (*l'interface*) d'un objet, de ce qui relève de la partie *privée* (le fonctionnement interne, les choix d'implantation...).
  Exemple : une classe Complexe permettra de manipuler un complexe indifféremment en représentation cartésienne ou polaire. Peu importe le choix fait pour stocker « en interne » la valeur du nombre Complexe !

Remarques :

- On peut « penser objet » / « concevoir un programme orienté objet » quelque soit le langage utilisé.
  Mais certains langages sont bien sûr plus adaptés.
- Un programme orienté objet n'est pas (nécessairement) plus lourd qu'un programme !