Cours n° 1
Langages de programmation
Introduction à la POO

Module Info3/M3105C

Sebastien.Kramm@univ-rouen.fr

IUT GEII Rouen

2018-2019

Sommaire

Niveau 0 : le processeur

- Un processeur ne peut exécuter que son langage natif ("code machine"), représenté sous une forme lisible par le langage dit "assembleur"

```
?add_pairs@@YAXPBMH@Z (void _cdecl add_pairs(float *,..
  00000000: 8B 54 24 0C    mov   edx,dword ptr [esp+0Ch]
  00000004: 85 D2          test  edx,edx
  00000006: 74 1F          je    00000027
  00000008: 8B 44 24 08    mov   eax,dword ptr [esp+8]
  0000000C: 8B 4C 24 04    mov   ecx,dword ptr [esp+4]
  00000010: D9 00          fld   dword ptr [eax]
```

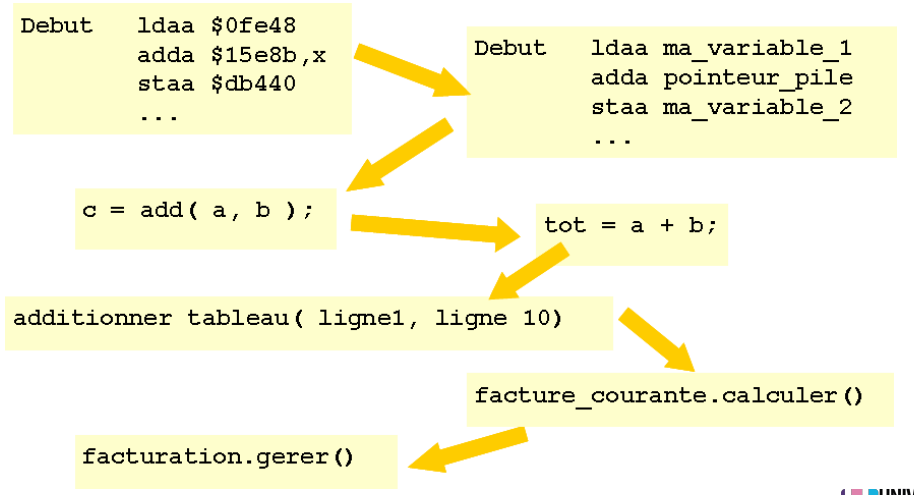
Question : que fait ce programme ? ? ?

- Idée : Rapprocher la programmation du langage naturel

FIGURE – classification sur un axe d'abstraction : de la machine à l'homme (source : DiScala)

Objectif d'un langage de programmation

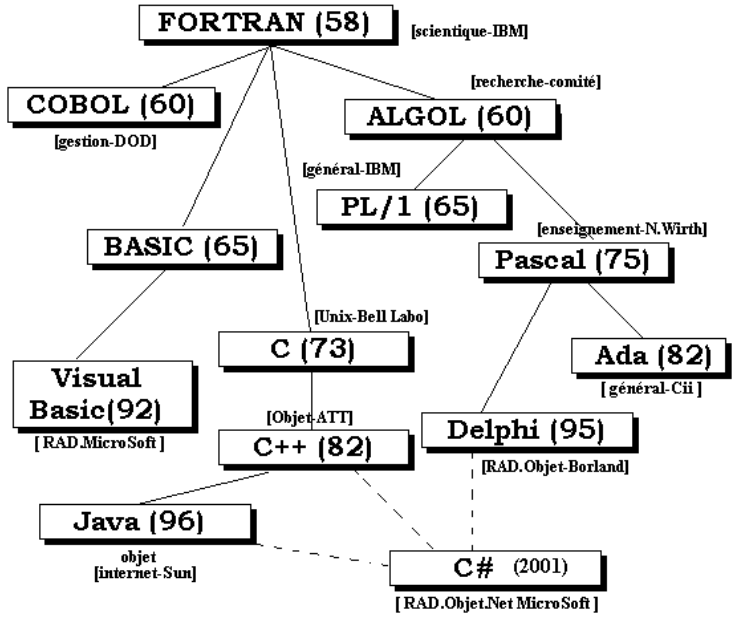
- Introduction d'un niveau d'abstraction, et suppression de l'information non essentielle :



Modèles d'exécution

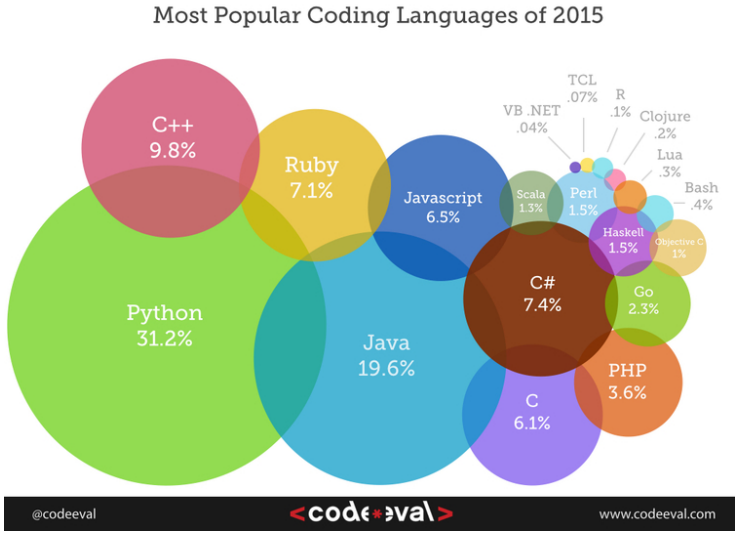
- Interprété (Basic, Perl, Javascript, ...) :
  - Le code est traduit en instructions-machine au moment de l'exécution,
  - Avantages : simplicité du développement,
  - Inconvénients : lent, peu efficace, structuration difficile.
- Compilé (C, C++, ...) :
  - Le code est traduit en instructions-machine lors de la phase de développement, on distribue les "binaires" (fichiers exécutables),
  - Avantage : code rapide,
  - Inconvénient : portabilité des binaires aléatoire.
- Mixte (Java, C#, ...) :
  - Le code est traduit dans un langage intermédiaire (JVM, MSIL), la machine-cible doit disposer d'un **interpreteur**,
  - Avantage : portabilité garantie,
  - Langages récents : haut niveau d'abstraction et large bibliothèques applicatives,
  - Inconvénients : moins rapide qu'un binaire pur.

Langages de programmation



source :Di Scala

Language utilisés



Source: devsaran.com

Beaucoup sont pensés pour la programmation Web !

Logiciel : historique

- A l'origine, travaux de recherche, labos publics ou privés
- 1970 : informatique = matériel, logiciel assimilé à du "réglage".
- Aujourd'hui : "*Big Business*"
  - Industrie majoritairement pilotée par des grands groupes (Microsoft, Sun, Borland, Adobe, Google, ...)
  - Les langages de programmation sont souvent liés à un acteur économique et sont partie intégrante de leur stratégie marketing :
    - Java ⇒ Sun (racheté par Oracle en janv. 2010)
    - C#, VB.NET ⇒ Microsoft ("plateforme .NET")
    - Delphi ⇒ Borland
    - Flash ⇒ Macromédia (Adobe)
  - Une exception : C++, langage indépendant
- Généralisation de l'exécution distribuée : un programme ne s'exécute plus sur **une** machine, mais sur **un réseau** de machines.

Modélisation : différentes approches

- On peut introduire différents niveaux d'abstraction, amenant différentes approches (**paradigmes** de programmation).
- On distingue (parmi d'autres...) :

Programmation procédurale

Axée sur les **traitements** : le problème est découpée en tâches de plus en plus détaillées
⇒ Programmes = Algorithmes + structures de données.

Programmation Orientée Objet (POO)

Axée sur les **données** : le problème est découpé en objets qui communiquent entre eux par envois de messages
⇒ Programmes = objets communiquant entre eux.

Modélisation : procédural vs. POO

Programmation procédurale

```
int a[50], b, c;
b = FonctionQuiCalculeUnTruc( a );
c = UneAutreFonction( b );
```

Programmation Orientée Objet (POO)

```
int    a; // creation d'un objet 'a' de type 'int'
MACHIN b; // creation d'un objet 'b' de type 'MACHIN'
b.CalculerCeci();
b.CalculerCela( a ); // passage d'argument
```

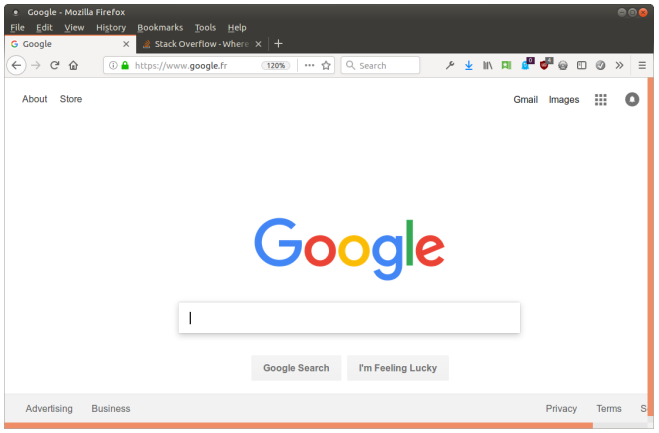

Sommaire

P.O.O. ? - Objet ?

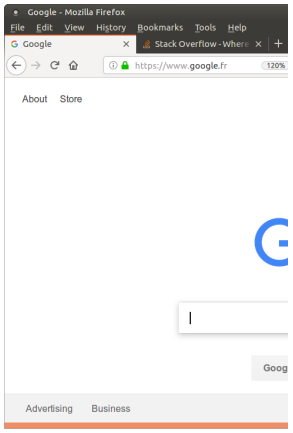
- POO : Tentative de rapprochement du comportement d'un programme avec le comportement du monde réel.
- On modélise la tâche à accomplir par des objets qui interagissent entre eux.
- Les objets au sens informatique peuvent représenter :
  - des objets réels : voiture, conducteur, chaise, porte, chien, ...,

  - des notions **immatérielles** : algorithme, point de l'espace, ligne de texte, rendez-vous, élement d'un programme...,
  - Des **agrégations** d'objets (listes), homogènes ou hétérogènes.

Exemple type : logiciel avec interface graphique



Exemple type : logiciel avec interface graphique



- Ce qu'on ne voit pas (noyau du programme)
- Ce qu'on voit (fenêtre)
  - Eléments de la fenêtre
    - Barre titre
    - Barre menus
    - Barre d'outils
    - Ascenseurs
    - ...
  - Contenu de la fenêtre (page HTML affichée en mode graphique)
    - image(s)
    - texte
    - hyperliens
    - ...
- ⇒ Autant d'objets !

Exemple de programme

- On pourrait écrire le programme précédent en 8 lignes !

En POO, les fonction *main()* sont courtes !

⇒ Tout est délégué aux objets

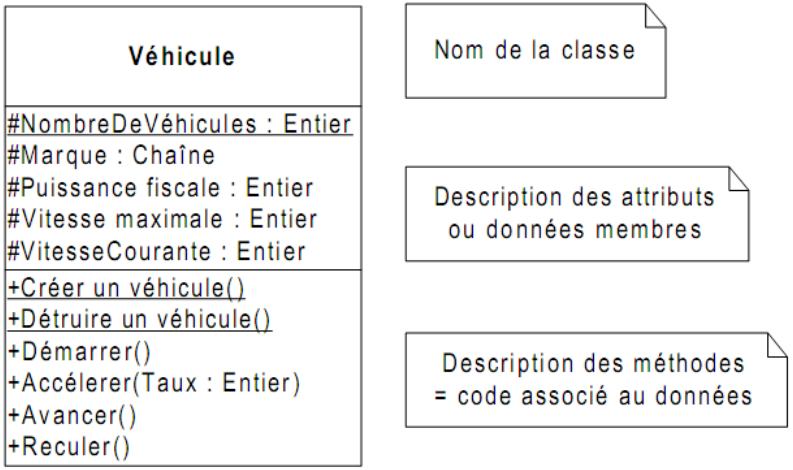
```
int main()
{
   Kernel noyau; // creation noyau
   Gui gui;  // creation  interface  graphique
   do {
      Event e = gui.GetEvent(); // evenement
      Content = noyau.Process( e ); // traitement par le noyau
      gui.Show( c );       // affichage  (éventuel)
   }
   while( e.GetMsg() != QUIT ); // tant que c'est  pas  fini  ...
}
```

- `noyau` : variable (=*objet*) de type 'Kernel'
- `gui` : variable (=*objet*) de type 'Gui'

Notion de classe

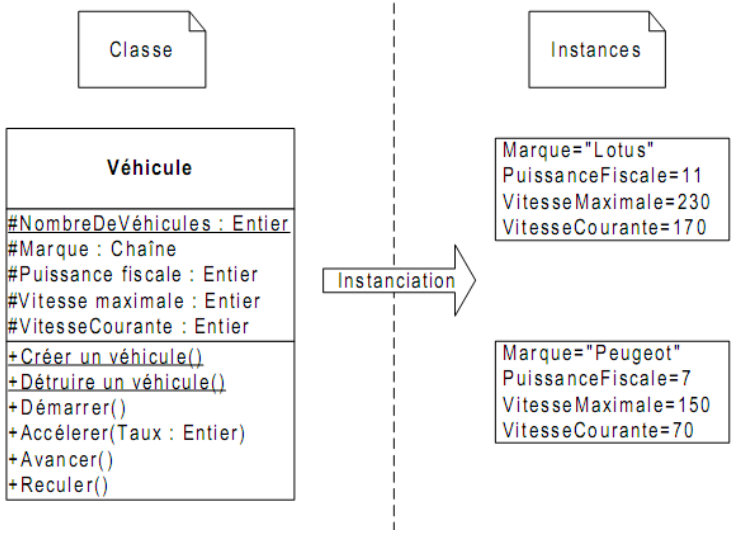
Une **classe** représente le modèle de l'objet ("moule").

- la liste de ses caractéristiques ⇒ **attributs**,
- les choses qu'il peut faire ⇒ **méthodes** (fonctions associées à l'objet).



Classe ≠ Instance

Un **objet** est une **instance** de la classe (création en mémoire d'une zone pour le stockage de ses données propres).



Principes fondamentaux de la POO

Trois principes à retenir :

- **Encapsulation** : on cache/protège ce qui n'a pas besoin d'être connu/manipulé par l'utilisateur de l'objet,
- **Heritage** : On hierarchise les objets afin de mutualiser ce qui peut l'être (factorisation) : attributs et/ou méthodes,
- **Polymorphisme** : on définit des méthodes communes qui peuvent s'adapter à la nature réelle de l'objet.

A - Encapsulation

- On ne montre de l'objet que ce qui est nécessaire à son utilisation : l'objet est une "boite noire", munie de "boutons" sur lesquelles on peut agir.
- Les données (attributs) sont protégés : leur modification est contrôlée par l'objet.
- Avantages :
  - simplification de l'utilisation de l'objet,
  - meilleure robustesse (les valeurs peuvent être contrôlées),
  - simplification maintenance,
  - optimisations possibles sans conséquence pour l'utilisateur.
- On fournit à l'utilisateur des méthodes d'accès aux données en lecture et en écriture ($\Rightarrow$ notion d'**interface**).

A - Encapsulation : analogie avec le monde réel

Exemple : automobile

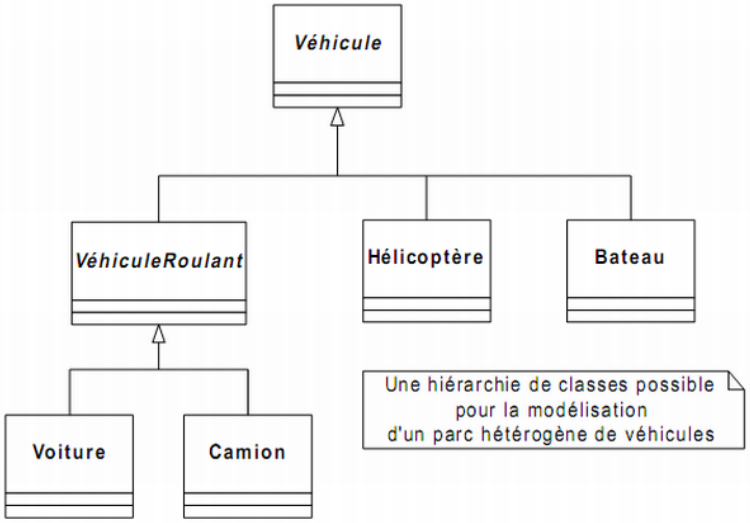
- L'utilisateur utilise une **interface** pour contrôler sa voiture (volant, pédales, levier de vitesse, ...)
- Il n'a pas besoin d'aller ouvrir le capot pour démarrer : le fonctionnement est **caché**.
- On peut changer des pièces sous le capot sans conséquences pour l'utilisateur.

Exemple

```
class Auto
{
   public:            // interface
      void Demarrer();
      void PasserUneVitesse( int num );
       ...
   private:      // données privées
      int QteCarburant;
      int Vitesse;
      int RapportBoite;
       ...
};
```


B - Heritage - 1

On peut regrouper les objets par "famille d'objets" : on introduit une **hiérarchie** entre les classes.



B - Heritage - 2

- On créera d'abord la classe "Vehicule", ayant des attributs et des méthodes communes à tous les véhicules. Par exemple :
  - Attributs : couleur, vitesse-maxi, ...
  - Méthodes : demarrer(), arreter(), ...
- Les classes "VehiculeRoulant", Helicoptère", "Bateau" seront dérivées de la classe "Vehicule" : elles **héritent** de tous les attributs et méthodes de la classe de base.
- On parle de **classe de base** et de **classes dérivées**.

Une règle :

"La classe dérivée est une version **spécialisée** de la classe de base"

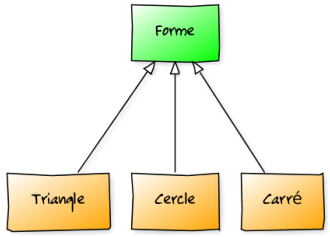
Remarque : une classe dérivée peut à son tour être dérivée.

B - Heritage - 3

- Intérêt de l'héritage : on peut mutualiser des données ou des méthodes communes.
- Dans l'exemple précédent :
  - Inutile d'avoir une méthode `Demarrer()` à la fois dans "Voiture" et dans "Camion" : on peut l'inclure dans la classe "VéhiculeRoulant".
  - Un attribut "VitesseMaxi" pourra être inclus dans la classe "Véhicule" : il sera alors automatiquement disponible dans **toutes** les classes dérivées.

C - Polymorphisme

- Le polymorphisme est la **capacité** qu'à une méthode de s'**adapter** automatiquement à l'objet manipulé (lié à la notion d'héritage).
- Exemple : soit le modèle d'héritage suivant (4 classes) :



  - Dessiner un carré ne se fait pas comme dessiner un cercle.
  - Pourtant, les trois formes doivent pouvoir se dessiner ⇒ on va inclure une méthode "dessiner()" dans la classe "Forme".
  - On doit pouvoir regrouper plusieurs formes différentes dans un tableau, et appeler leur méthode "dessiner()" sur chacun d'eux.

⇒ On dit que la méthode "dessiner()" est **polymorphe** : elle s'**adapte** à la nature de l'objet qu'elle manipule.

Regroupement d'objets

- Une classe "voiture" peut dériver (hériter) d'une classe "véhicule".

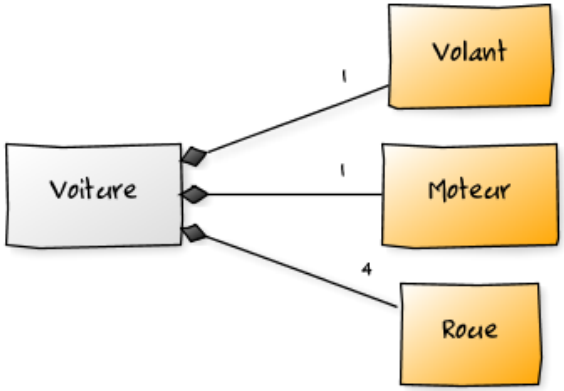
  - Mais une classe "roue" ne peut pas dériver d'une classe "voiture" !
  - Par contre une classe "voiture" pourra **contenir** d'autres objets :
    - 4 objets de la classe "roue",
    - 1 objet de la classe "moteur",
    - 1 objet de la classe "volant".

On parle de **relation d'association**, et plus précisémment :

- de **composition** si la destruction de l'objet entraîne la destruction des éléments contenus,
- d'**agregation** si les objets contenus ont une existence indépendante.

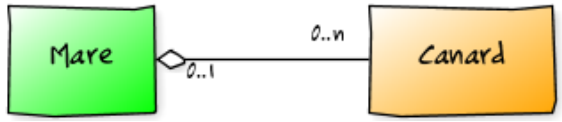
Relation de composition

- Si l'objet "Voiture" est détruit, les objets contenus le seront aussi.



Relations d'agregation

- Les objets ont une existence indépendante :
  - la mare "contient" des canards,
  - la mare existe sans canards,
  - les canards existent sans la mare.



- En C++, l'agregation sera implémentée par des **pointeurs**.

Difficultés liée à la POO

- L'analyse préalable du problème est primordiale. Il faut :
  - Identifier les objets en jeu,
  - Identifier leurs interactions, statiques et dynamiques.
- Dans la réalité, la modélisation et la décomposition en objets sont difficile.
- Attention à ne pas faire des décompositions trop lourdes.
  Ex : une classe jamais instanciée peut éventuellement être supprimée.
- Un outil : **UML** : Unified Modeling Language (langage de modélisation objet unifié).

Pour aller plus loin...

- Références

  `http://fr.wikipedia.org/wiki/Programmation_orientée_objet`
  `http://fr.wikipedia.org/wiki/Unified_Modeling_Language`
  `http://laurent-audibert.developpez.com/Cours-UML/`
- Sources des images
  - Bruno Garcia (`http://bruno-garcia.net/`)
  - yUML (`http://yuml.me/`)
  - RM Di Scala (`http://rmdiscala.free.fr/`)