

**Yantra Technologies**

[www.yantra-technologies.com](http://www.yantra-technologies.com)

# Programmation Orientée Objet

## Les Concepts



**Facile**



**Normal**



**Difficile**



**Professionnel**



**Expert**

[carole.grondein@yantra-technologies.com](mailto:carole.grondein@yantra-technologies.com)  
[david.palermo@yantra-technologies.com](mailto:david.palermo@yantra-technologies.com)

[https://wiki.waze.com/wiki/Your\\_Rank\\_and\\_Points](https://wiki.waze.com/wiki/Your_Rank_and_Points)

- 1 - Présentation
- 2 - Concepts de bases
- 3 - Quelques Design Patterns
- 4 - Différences entres les langages
- 5 - Bibliographie



# 1 - Présentation

- 1.1 - Les styles de programmation
- 1.2 - L'approche Objet et Langage UML
- 1.3 - Qu'est-ce qu'un Objet ?
- 1.4 - La modélisation avec UML
- 1.5 - UML
- 1.6 - Utilisation UML
- 1.7- Les diagrammes UML 2.0
- 1.8 - L'approche orientée objet
- 1.9 - Langage Objet
- 1.10 - Les avantages de la POO

## 1.1 - Les styles de programmation



- **Fonctionnel ou Applicatif** : évaluation d'expressions comme une formule et d'appeler des fonctions à l'intérieur d'autres fonctions : *Lisp, Caml, APL* (récursivité)
- **Procédural ou Impératif** : exécution d'instructions étape par étape *Fortran, C, Pascal, Cobol* (itératif)
- **Logique** : répondre à une question par des recherches sur un ensemble, en utilisant des axiomes, des demandes et des règles de déduction *Prolog*
- **Objet** : *Simula, Smalltalk, C++, Java, ADA 95*
  - ensemble de composants autonomes (objets) qui disposent de moyens d'interaction,
  - utilisation de classes,
  - échange de message.



L'Approche Objet est une **démarche** qui consiste à utiliser **l'objet** pour **modéliser** le monde réel.

- Une démarche => Une méthodologie pour décrire comment s'organiser le système
- Une modélisation => Une Notation **UML**

## 1.2 - Qu'est-ce qu' un Objet ? (1)

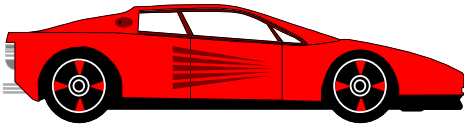


### Propriétés / Attributs

Ferrari

rouge

En\_stationnement



### Comportements / Méthodes

rouler

se\_garer

### Identité

ma\_ferrari

305 XV 13

## 1.2 - Qu'est-ce qu 'un Objet ? (2)



un **objet** représente un concept du monde réel possédant **une identité** et un **état** auquel peuvent être associés des **propriétés** et des **comportements**.

- L '**état** d'un objet comprend toutes les propriétés d'un objet (habituellement statiques) plus les valeurs courantes de celles-ci (habituellement dynamiques).
  - Une **propriété** d'un objet est une caractéristique inhérente et distincte qui contribue à l'unicité de l'objet.
  - Le **comportement** d'un objet c'est comment l'objet agit et réagit en termes de changement d'état et de passage de message.
- L'**identité** d'un objet permet de le distinguer des autres objets, indépendamment de son état.



En 1994, plus de 50 méthodes OO

- Fusion, Shlaer-Mellor, ROOM, Classe-Relation, Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS...

Les méta modèles se ressemblent de plus en plus

Les notations graphiques sont toutes différentes

L'industrie a besoin de standards

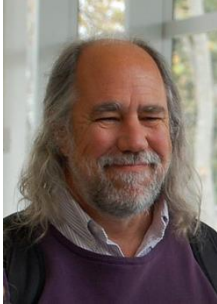




La pratique des méthodes a permis de faire le tri entre les différents concepts

***Jim Rumbaugh, Grady Booch*** (1993) et plus tard ***Ivar Jacobson*** ( 1994) décident d'unifier leurs travaux:

- Methode OMT(Object Modeling Technique)
- Methode Booch
- Methode OOSE (Object Oriented Software Engineering)



*Grady Booch*

### Méthode de Grady Booch

La méthode proposée par G. Booch est une méthode de conception, définie à l'origine pour une programmation Ada, puis généralisée à d'autres langages. Sans préciser un ordre strict dans l'enchaînement des opérations



*James Rumbaugh*

### Méthode OMT

La méthode OMT (Object Modeling Technique ) permet de couvrir l'ensemble des processus d'analyse et de conception en utilisant le même formalisme. L'analyse repose sur les trois points de vue: statique, dynamique, fonctionnel, donnant lieu à trois sous-modèles.



*Ivar Jacobson*

### Méthode OOSE

Object Oriented Software Engineering (OOSE) est un langage de modélisation objet créé par Ivar Jacobson. OOSE est une méthode pour l'analyse initiale des usages de logiciels, basée sur les « cas d'utilisation » et le cycle de vie des logiciels.



# UML : Unified Modeling Language

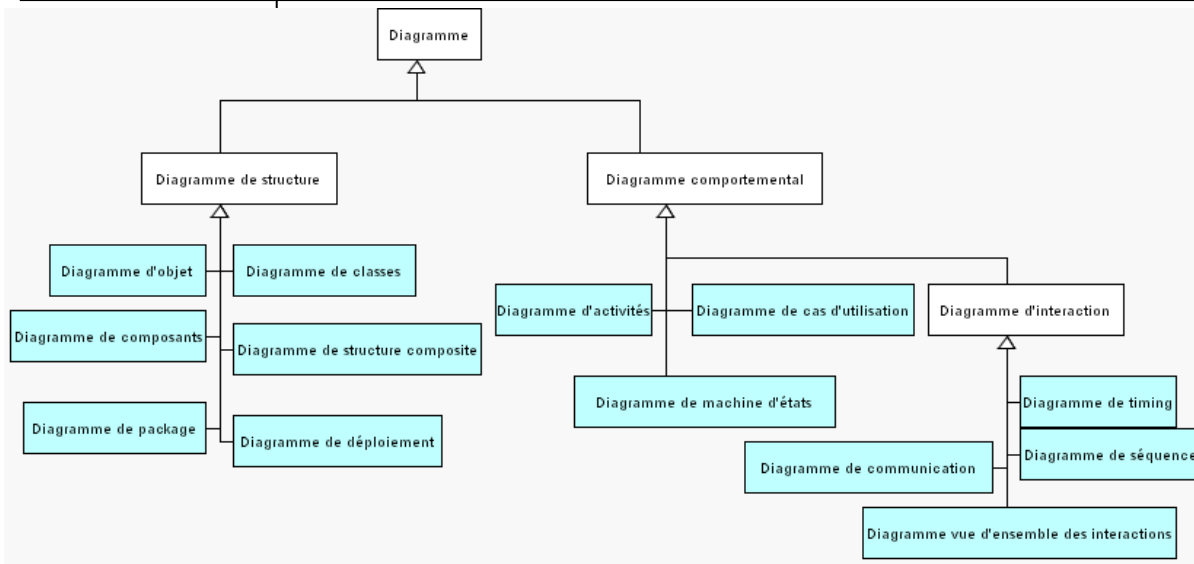


- **UML permet d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation.** Il a été pensé pour servir de support à une analyse basée sur les concepts objet.
- UML est un **langage formel**, défini par un **métamodèle**.
- Le métamodèle d'UML décrit de manière très précise tous les éléments de modélisation et la sémantique de ces éléments (leur définition et le sens de leur utilisation).  
**UML normalise les concepts objet.**
- UML est avant tout **un support de communication performant**, qui facilite la représentation et la compréhension de solutions objet



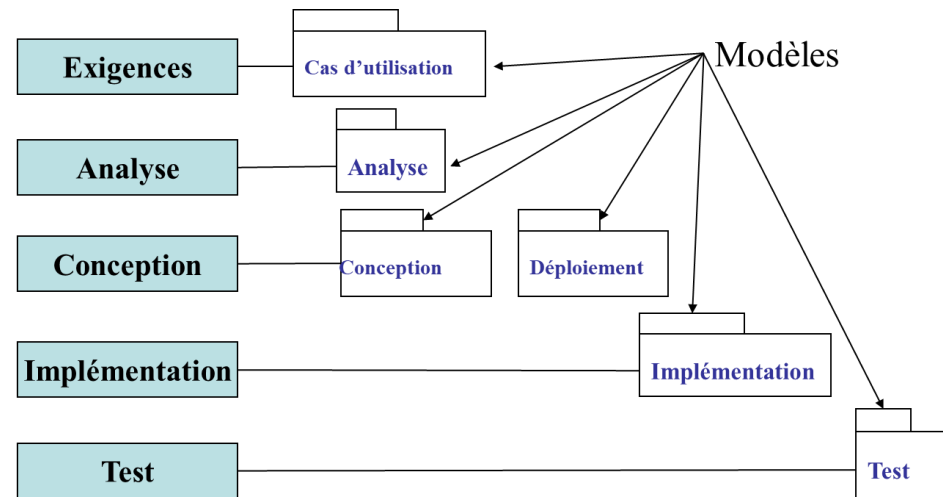
- Les points forts d'UML
  - UML est un langage formel et normalisé
  - UML est un support de communication performant
- Les points faibles d'UML
  - La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation.
  - Le processus (non couvert par UML) est une autre clé de la réussite d'un projet.

## 1.6 - Utilisation UML



### Modélisation UML

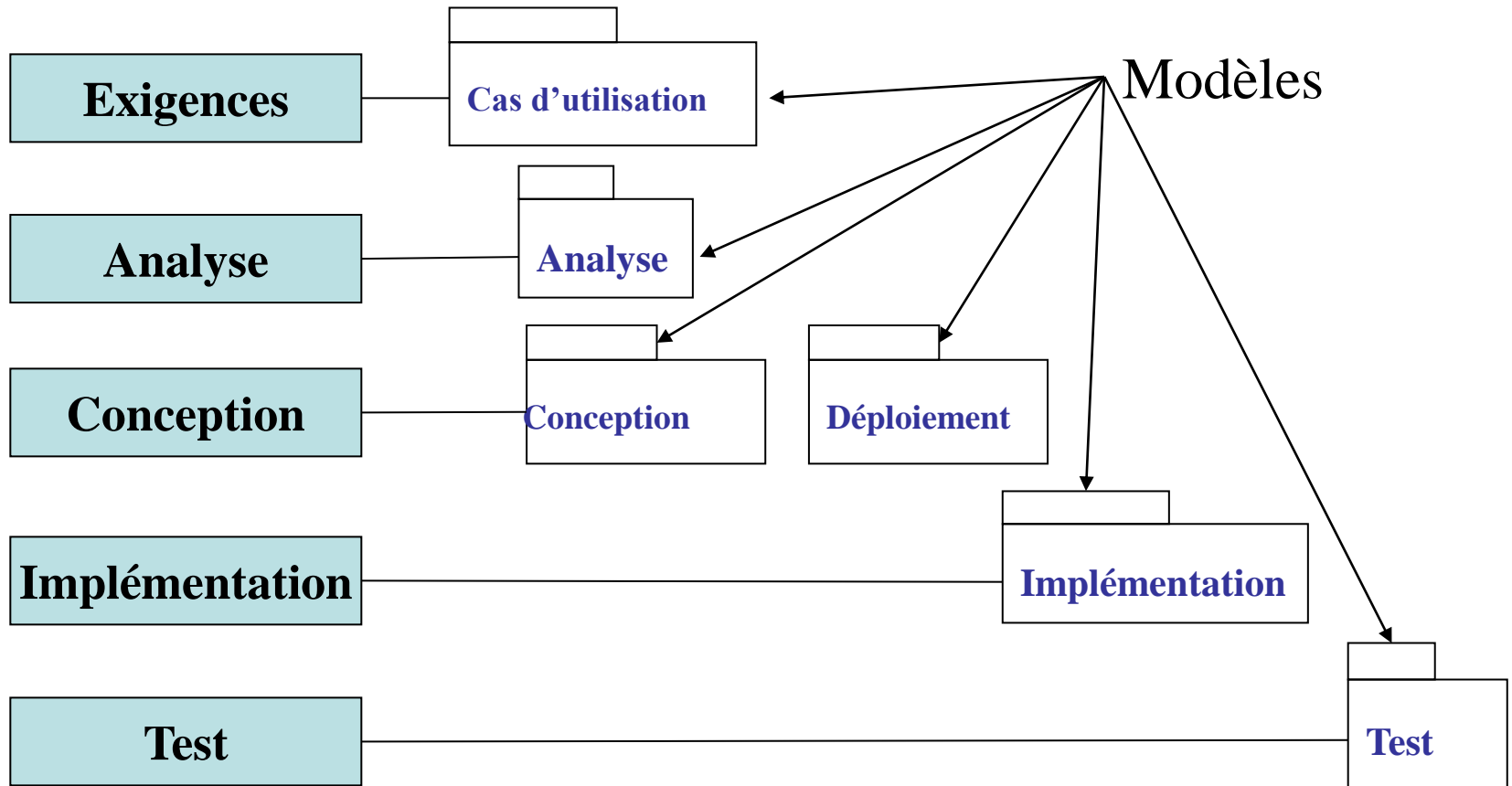
- 14 diagrammes ( diagramme de profile ajouter dans UML2.2)
- 1 Langage de contraintes objet (Object Constraint Language : OCL)



<https://manurenaux.wp.imt.fr/2013/09/27/interet-de-luml-dans-un-projet-informatique/>

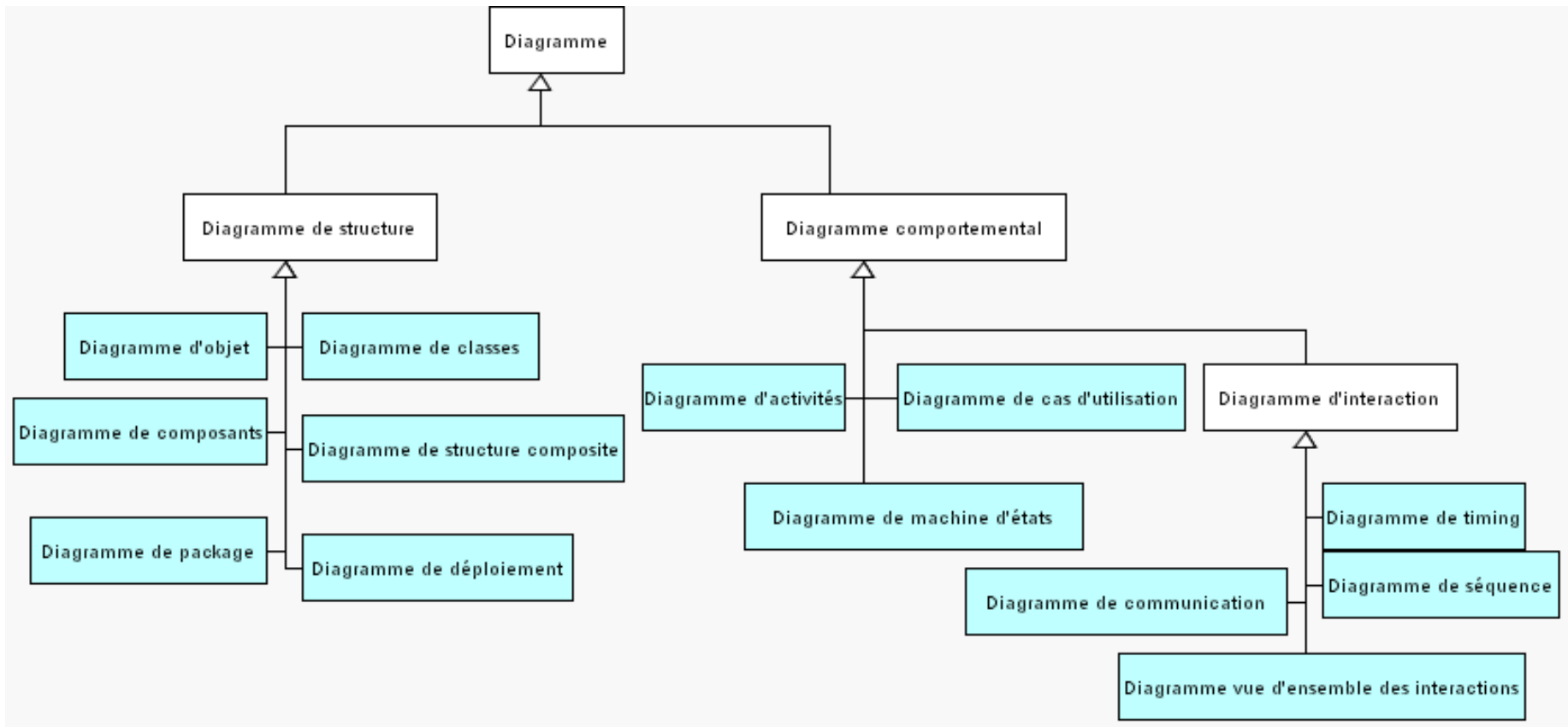


# Modélisation UML





## Modélisation UML 2.0



## 1.8 - L'approche orientée objet



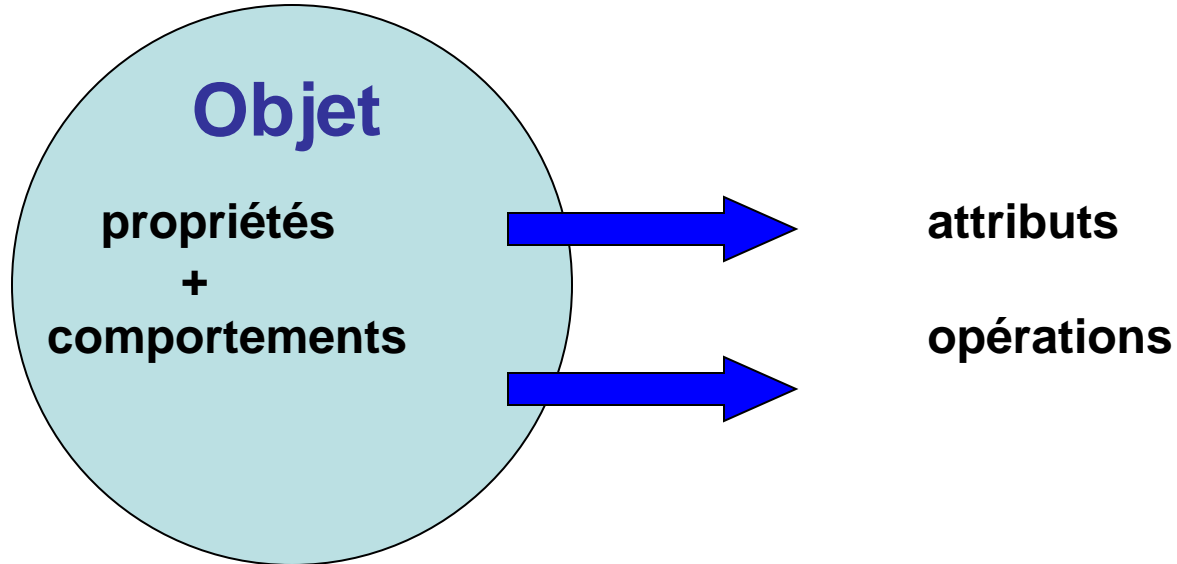
L'approche orientée objet considère le logiciel comme une collection d'objets dissociés définis par des **propriétés**.  
(propriété : **attribut** ou une **opération**)

- ⇒ Un objet comprend à la fois une structure de données et une collection d'opérations (son comportement).
- ⇒ Un certain nombre de caractéristiques pour qu'une approche soit dite orientée objet il faut : l'identité, la classification, le polymorphisme et l'héritage.





# Le modèle objet





### 3 concepts pour faire un langage objet :

- **Encapsulation** : combiner des données et un comportement dans un emballage unique,
- **Héritage** : chien et chat sont des mammifères, ils héritent du comportement du mammifère.
- **Polymorphisme** : Cercle et rectangle sont des formes géométriques, chacun doit calculer sa surface.



- **Facilite la programmation modulaire :**
  - composants réutilisables,
  - un composant offre des services et en utilise d'autres,
  - il expose ses services au travers d'une interface.
- **Facilite l'abstraction :**
  - elle sépare la définition de son implémentation,
  - elle extrait un modèle commun à plusieurs composants,
  - le modèle commun est partagé par le mécanisme d'héritage.
- **Facilite la spécialisation :**
  - elle traite des cas particuliers,
  - le mécanisme de dérivation rend les cas particuliers transparents.



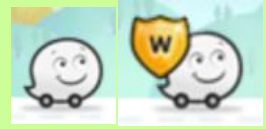
- 1- Présentation de la notion d 'Objet
- 2- La modélisation avec UML
- 3- Le principe de l'encapsulation
- 4- La Classe
- 5 -L'Héritage
- 6- Agrégation
- 7- Polymorphisme
- 8- Les classes abstraites
- 9- Relations d'association
- 10- Le Langage Objet
- 11- Développement d 'un projet orienté Objet

## 2.1- Présentation de la notion d'Objet

### Un Objet peut :

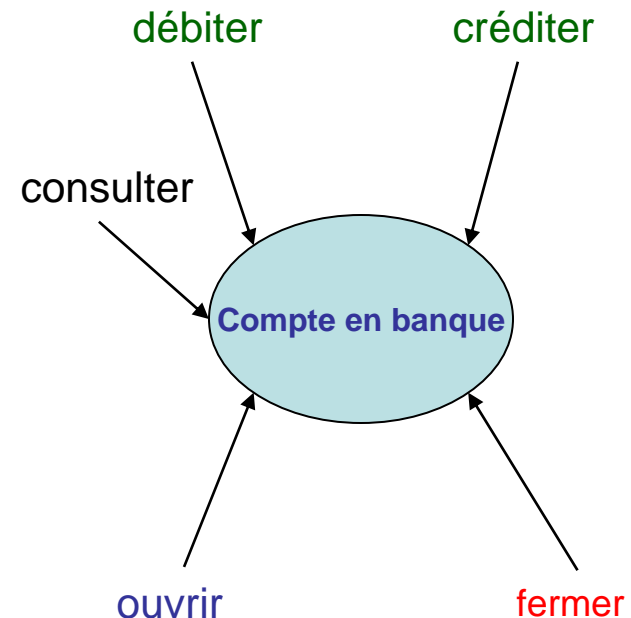
- changer d'état,
- se comporter de façon discernable,
- être manipulé par diverses formes de stimuli,
- être en relation avec d'autres objets.

Chaque objet a sa propre **identité** et donc une existence indépendante des autres.

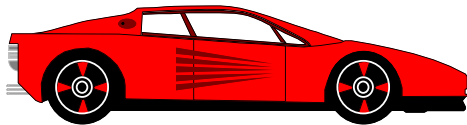


# Types d'opération sur le comportement

- **Modificateur** : une opération qui altère l'état d'un objet,
- **Sélecteur** : une opération qui accède à l'état d'un objet
- **Itérateur** : une opération qui permet d'avoir accès à toutes les parties d'un objet dans un ordre défini,
- **Constructeur** : une opération qui crée un objet et initialise son état,
- **Destructeur** : une opération qui détruit un objet.



## 2.1- Présentation de la notion d'Objet



propriétés

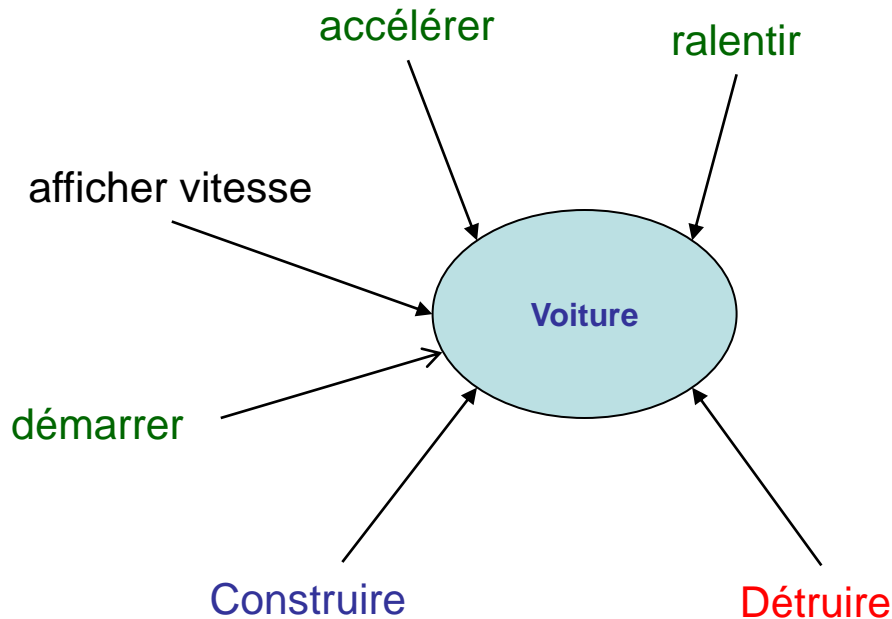
Ferrari  
rouge  
en\_stationnement

comportements

rouler  
se\_garer

identité

ma\_ferrari  
305 XV 13



class Test Model

**Voiture**

```
+ afficherVitesse(): void
+ demarrer(): void
+ accelerer(): void
+ ralentir(): void
«constructor»
+ Voiture(): void
«destructor»
+ ~Voiture(): void
```

## 2.1- Présentation de la notion d'Objet : Exercices – décrire l'objet carte

[https://img.yugioh-card.com/ygo\\_cms/ygo/all/uploads/Rulebook\\_v9\\_fr.pdf](https://img.yugioh-card.com/ygo_cms/ygo/all/uploads/Rulebook_v9_fr.pdf)





## 2.1 - Objet : PHP



```
print ("exemple 1 : PHP </br>" );  
$voiture = new Voiture("305 XV 13","Ferrari","rouge");  
$voiture->demarrer() ;  
$voiture->afficher_vitesse() ;  
$voiture->accelerer() ;  
$voiture->afficher_vitesse() ;  
$voiture->ralentir();  
$voiture->afficher_vitesse() ;  
$voiture->arreter();  
$voiture->afficher_vitesse() ;  
$voiture = null;
```

## 2.1 - Objet : Java



```
public static void main(String[] args) {  
    System.out.println("Exemple 1 : Java");  
    Voiture maFerrari = new Voiture("305 XV 13", "Ferrari", "rouge");  
    maFerrari.demarrer();  
    maFerrari.afficherVitesse();  
    maFerrari.accelerer();  
    maFerrari.afficherVitesse();  
    maFerrari.ralentir();  
    maFerrari.afficherVitesse();  
    maFerrari.arreter();  
    maFerrari.afficherVitesse();  
}
```

## 2.1 - Objet : Python



```
def Exemple_01_python():  
    print("Exemple 1 : python")  
    maFerrari = Voiture("305 XV 13","Ferrari","rouge")  
    maFerrari.demarrer();  
    maFerrari.afficherVitesse();  
    maFerrari.accelerer();  
    maFerrari.afficherVitesse();  
    maFerrari.ralentir();  
    maFerrari.afficherVitesse();  
    maFerrari.arreter();  
    maFerrari.afficherVitesse();  
    del maFerrari;
```

Exemple\_01\_python()

## 2.1- Objet : C++



```
void Exemple01_Cpp()
{
    std::cout << "Exemple 1 : C++" << std::endl;
    Voiture* maFerrari = new Voiture("305 XV 13", "Ferrari", "rouge");

    maFerrari->demarrer();
    maFerrari->afficherVitesse();
    maFerrari->accelerer();
    maFerrari->afficherVitesse();
    maFerrari->ralentir();
    maFerrari->afficherVitesse();
    maFerrari->arreter();
    maFerrari->afficherVitesse();

    delete maFerrari;
}
```

## 2.1- Objet : C#



```
public static void Main(string[] args)
{
    Console.WriteLine("Exemple 1 : C#");
    Voiture voiture = new Voiture("305 XV 13", "Ferrari", "rouge");
    voiture.demarrer();
    voiture.afficher_vitesse();
    voiture.accelerer();
    voiture.afficher_vitesse();
    voiture.ralentir();
    voiture.afficher_vitesse();
    voiture.arreter();
    voiture = null;
    Console.Read();
}
```

## 2.1 – Objet : procédural C



```
void Exemple01_C()  
{  
    std::cout << "Exemple 1 : C" << std::endl;  
    Voiture_struct* maFerrari = Voiture_construire("305 XV 13", "Ferrari", "rouge");  
  
    Voiture_demarrer(maFerrari);  
  
    Voiture_afficherVitesse(maFerrari);  
    Voiture_accelerer(maFerrari);  
    Voiture_afficherVitesse(maFerrari);  
    Voiture_ralentir(maFerrari);  
    Voiture_afficherVitesse(maFerrari);  
    Voiture_arreter(maFerrari);  
    Voiture_afficherVitesse(maFerrari);  
  
    Voiture_detruire(&maFerrari);  
}
```



# L' Encapsulation

C'est le processus qui consiste à cacher tous les détails d'un objet. L'objet peut être vu :

- de **l'intérieur** pour le concepteur : détail de la structure et du comportement, données et méthodes privées,
- de **l'extérieur** pour l'utilisateur : interface « publique » qui décrit l'utilisation de l'objet.

Permet de rendre indépendante la spécification de l'objet et son implémentation (*la vue externe doit être la plus indépendante possible de la vue interne*).

## 2.4 – Notion de classe



On appelle **classe** la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet.

Un objet est donc « issu » d'une classe, c'est le produit qui sort d'un moule.

On dit qu'un objet est une **instanciation** d'une classe, c'est la raison pour laquelle on pourra parler indifféremment d'**objet** ou d'**instance** (éventuellement d'*occurrence*).

Une classe est composée de deux parties :

- **Les attributs** (parfois appelés *données membres*) : il s'agit des données représentant l'état de l'objet
- **Les méthodes** (parfois appelées *fonctions membres*): il s'agit des opérations applicables aux objets



## 2.4 – Notion de classe



On définit la classe *voiture*

-> *Peugeot 406*, *Renault 18* seront des instances de cette classe donc des objets

Nous pourrions créer différents Objets *Peugeot 406*, *Renault 18* différenciés par leur numéro de série.

Les deux instance de classes pourront avoir tous leurs attributs égaux sans pour autant être un seul et même objet.

Exemple : deux T-shirts peuvent être strictement identiques et pourtant ils sont distincts.

Si on les mélangeant, il serait impossible de les distinguer...

## 2.4 – Notion de classe



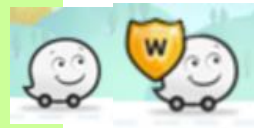
On appelle **classe** la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet. Un objet est donc « issu » d'une classe, c'est le produit qui sort d'un moule.

On dit qu'un objet est une **instanciation** d'une classe, c'est la raison pour laquelle on pourra parler indifféremment d'**objet** ou d'**instance** (éventuellement d'*occurrence*).

Une classe est composée de deux parties :

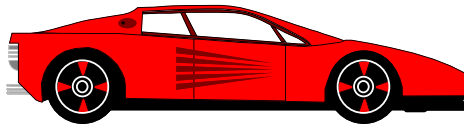
- **Les attributs** (parfois appelés *données membres*) : il s'agit des données représentant l'état de l'objet
- **Les méthodes** (parfois appelées *fonctions membres*): il s'agit des opérations applicables aux objets

Si on définit la classe *voiture*, les objets *Peugeot 406*, *Renault 18* seront des instanciations de cette classe. Il pourra éventuellement exister plusieurs objets *Peugeot 406*, différenciés par leur numéro de série. Mieux: deux instanciations de classes pourront avoir tous leurs attributs égaux sans pour autant être un seul et même objet. C'est le cas dans le monde réel, deux T-shirts peuvent être strictement identiques et pourtant ils sont distincts. D'ailleurs, en les mélangeant, il serait impossible de les distinguer...



# Une classe

C'est un ensemble d'objets ayant les mêmes propriétés et un comportement commun.







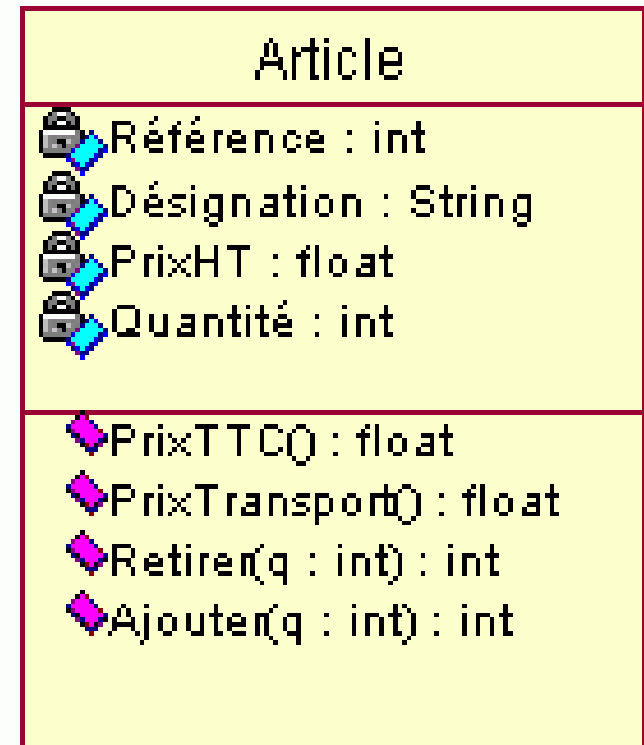
- Une **classe** est une construction du langage de programmation :
  - décrit les propriétés communes à des objets  
Class Point { }
  - un objet est une *instance* de classe
- Un **objet** est une entité en mémoire :
  - il a un état, un comportement, une identité.  
Point\* a = new Point(3,5); (C++)  
Point a = new Point(3,5); (java)

# Un objet sans classe n'existe pas



### La classe possède :

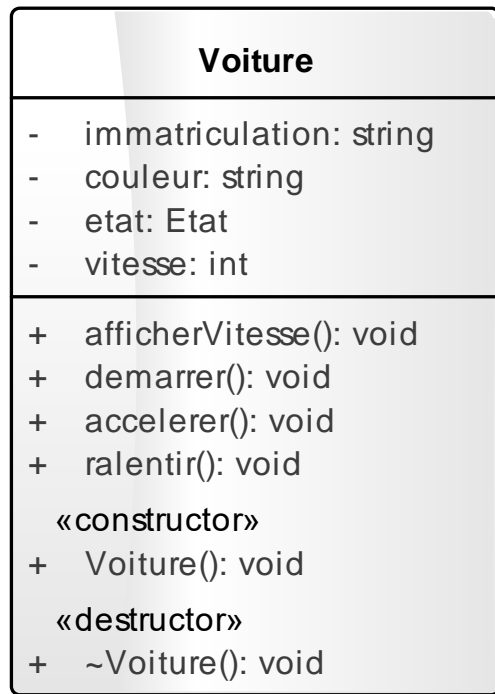
- un nom unique,
- une composante **statique** :  
les données qui sont des  
champs (attributs)  
⇒ Etat
- une composante  
**dynamique** : les méthodes  
(opérations)  
⇒ Comportement



## 2.4- La Classe

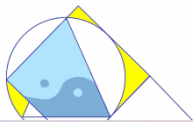


### class Test Model



Etat est soit "arreter" soit "demarrer"





2 Cartes du jeu

Cartes Monstre

COMMENT LIRE UNE CARTE

**1 Nom de Carte**

**7 Description de la Carte**

**4 Type**

**3 Attribut**

**2 Niveau**

**5 Numéro de Carte**

**6 ATK (Valeur d'Attaque) / DEF (Valeur de Défense)**

**1 Nom de Carte**  
C'est le nom de la carte. Lorsque le nom d'une carte est mentionné dans le texte d'une carte, il apparaît entre guillemets. Si des cartes ont le même nom, elles sont considérées comme étant la même carte.

**2 Niveau**  
Comptez le nombre d'étoiles pour déterminer le Niveau du monstre. Pour les Monstres Xyz, le nombre d'étoiles sur la droite indique le Rang du monstre.

6

3 Attribut

Chaque monstre a un Attribut. Cet Attribut peut être important pour des effets de carte.

4 Type

Les monstres sont répartis dans différents Types. Certains monstres ont également une information complémentaire à cet endroit, à côté de leur Type.

5 Numéro de Carte

Le numéro d'identification d'une carte est imprimé à cet endroit. Ce numéro vous sera utile pour gérer votre collection et trier vos cartes.

6 ATK (Valeur d'Attaque) / DEF (Valeur de Défense)

L'ATK d'un monstre représente sa valeur d'Attaque et la DEF représente sa valeur de Défense. Des valeurs d'Attaque et de Défense importantes sont très utiles pour les combats !

7 Description de la Carte

Les effets des cartes sont écrits ici, indiquant les capacités spéciales du monstre et comment les utiliser. En règle générale, les effets des monstres ne peuvent pas être utilisés lorsqu'ils sont Posés face verso sur le Terrain. Les Cartes Monstre Normal de couleur jaune n'ont pas d'effets, et comportent à cet endroit une description qui n'a pas d'impact sur le jeu.

7

[https://img.yugioh-card.com/ygo cms/ygo/all/uploads/Rulebook\\_v9\\_fr.pdf](https://img.yugioh-card.com/ygo cms/ygo/all/uploads/Rulebook_v9_fr.pdf)



## 2.4- La Classe : PHP



```
class Voiture{
    public $immatriculation ;
    public $typevoiture ;
    public $couleur ;
    public $etat = "arret" ;
    public $vitesse = 0 ;
    public function __construct($r,$v,$c){
        print ("Voiture : construction </br>" ) ;
        $this->immatriculation= $r;
        $this->typevoiture= $v;
        $this->couleur= $c;
    }
    public function demarrer(){
        print("Voiture : demarrer </br>" ) ;
        $this->etat = "demarrer";
    }
    public function arreter(){
        print ("Voiture : arreter </br>") ;
        $this->etat ="arreter";
    }
    public function afficher_vitesse(){
        print ("Vitesse ". $this->vitesse."</br>");
    }
    public function accelerer(){
        print ("Voiture : accelerer </br>") ;
        if ( $this->etat == "demarrer" )
            $this->vitesse+=1;
    }
    public function ralentir(){
        print ("Voiture : ralentir </br>" ) ;
        if ( $this->etat == "demarrer" and $this->vitesse != 0 )
            $this->vitesse-=1;
    }
}

public function __destruct(){
    print ("Voiture : destruction </br>" ) ;
}
}
```

## 2.4- La Classe : Java



```
package ExempleDP;

public class Voiture {

    public static void main(String[] args) { ...13 lines }

    private final String a_type;
    private final String a_immatriculation;
    private String a_couleur;
    private String a_etat;
    private int a_vitesse;

    public Voiture(String immatriculation, String typevoiture, String couleur) {
        System.out.println("Voiture : constructeur");
        this.a_immatriculation = immatriculation;
        this.a_couleur = couleur;
        this.a_type = typevoiture;
        this.a_etat = "arreter";
        this.a_vitesse = 0;
    }

    public void demarrer() {
        System.out.println("Voiture : demarrer");
        this.a_etat = "demarrer";
    }

    public void afficherVitesse() {
        System.out.println("Vitesse :" + this.a_vitesse);
    }
}
```

## 2.4- La Classe : Java



```

public void afficherVitesse() {
    System.out.println("Vitesse :" + this.a_vitesse);
}

public void accelerer() {
    System.out.println("Voiture : accelerer ");
    if ("demarrer".equals(this.a_etat)) {
        this.a_vitesse += 1;
    }
}

public void ralentir() {
    System.out.println("Voiture : ralentir ");
    if ("demarrer".equals(this.a_etat) && this.a_vitesse > 0) {
        this.a_vitesse -= 1;
    }
}

public void arreter() {
    System.out.println("Voiture : arreter ");
    this.a_etat = "arreter";
    this.a_vitesse = 0;
}

@Override
protected void finalize() throws Throwable {
    System.out.println("Voiture : destruction ");
    super.finalize();
}
}

```

## 2.4- La Classe : Python



```
class Voiture:
    def __init__(self,immatriculation,typevoiture,couleur):
        print("Voiture : construction ")
        self.immatriculation=immatriculation
        self.type=typevoiture
        self.couleur=couleur
        self.etat="arreter"
        self.vitesse=0
        pass
    def demarrer(self):
        print("Voiture : construction ")
        self.etat="demarrer"
        pass
    def afficherVitesse(self):
        print("Vitesse ",self.vitesse)
        pass
    def accelerer(self):
        print("Voiture : accelerer ")
        if ( self.etat == "demarrer"):
            self.vitesse+=1
        pass
    def ralentir(self):
        print("Voiture : ralentir ")
        if ( self.etat == "demarrer" and self.vitesse > 0):
            self.vitesse-=1
        pass
    def arreter(self):
        print("Voiture : arreter ")
        self.etat="arreter"
        self.vitesse=0
        pass
    def __del__(self):
        print("Voiture : destruction ")
    pass
```

## 2.4- La Classe : Python



```
def Exemple_01_python():  
    print("Exemple 1 : python")  
    maFerrari = Voiture("305 XV 13", "Ferrari", "rouge")  
    maFerrari.demarrer();  
    maFerrari.afficherVitesse();  
    maFerrari.accelerer();  
    maFerrari.afficherVitesse();  
    maFerrari.ralentir();  
    maFerrari.afficherVitesse();  
    maFerrari.arreter();  
    maFerrari.afficherVitesse();  
    del maFerrari;
```

Exemple\_01\_python()

## 2.4- La Classe : C++



```
class Voiture
{
public:
    Voiture(std::string immatriculation, std::string type, std::string couleur);

    ~Voiture();
    void demarrer();
    void afficherVitesse() const;
    void accélérer();
    void ralentir();
    void arrêter();

protected:
    const std::string _immatriculation;
    const std::string _type;
    std::string _couleur;

    std::string _etat;
    unsigned int _vitesse;

};
```

## 2.4- La Classe : C++



```
Voiture::Voiture(std::string immatriculation, std::string type, std::string couleur):
    _immatriculation(immatriculation),
    _type(type),
    _couleur(couleur),
    _etat("arreter"),
    _vitesse(0)
{
    std::cout << "Voiture : construction " << endl;
}

Voiture::~Voiture()
{
    std::cout << "Voiture : destruction " << endl;
}

void Voiture::Voiture::demarrer()
{
    std::cout << "Voiture : demarrer " << endl;
    this->_etat="demarrer";
}

void Voiture::afficherVitesse() const
{
    std::cout << "Vitesse : " << this->_vitesse << std::endl;
}

void Voiture::accelerer()
{
    std::cout << "Voiture : accelerer " << endl;
    if ( this->_etat=="demarrer") this->_vitesse++;
}

void Voiture::ralentir()
{
    std::cout << "Voiture : ralentir " << endl;
    if ( this->_vitesse != 0 ) this->_vitesse--;
}

void Voiture::arreter()
{
    std::cout << "Voiture : arreter " << endl;
    this->_vitesse=0;
    this->_etat="arreter";
}
```

## 2.1- Objet : C++



```
void Exemple01_Cpp()
{
    std::cout << "Exemple 1 : C++" << std::endl;
    Voiture* maFerrari = new Voiture("305 XV 13", "Ferrari", "rouge");

    maFerrari->demarrer();
    maFerrari->afficherVitesse();
    maFerrari->accelerer();
    maFerrari->afficherVitesse();
    maFerrari->ralentir();
    maFerrari->afficherVitesse();
    maFerrari->arreter();
    maFerrari->afficherVitesse();

    delete maFerrari;
}
```



## 2.1- Objet : C#



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace POO
{
    4 références
    class Voiture
    {
        0 références
        public static void Main(string[] args)
        {
            Console.WriteLine("Exemple 1 : C#");
            Voiture voiture = new Voiture("305 XV 13", "Ferrari", "rouge");
            voiture.demarrer();
            voiture.afficher_vitesse();
            voiture.accelerer();
            voiture.afficher_vitesse();
            voiture.ralentir();
            voiture.afficher_vitesse();
            voiture.arreter();
            voiture = null;
            GC.Collect();
            GC.WaitForPendingFinalizers();
            Console.Read();
        }

        1 référence
        public Voiture( string immatriculation, string type, string couleur) {
            Console.WriteLine("Voiture : construction");
            this.a_immatriculation = immatriculation ;
            this.a_type = type;
            this.a_couleur = couleur;
        }

        1 référence
        public void demarrer() {
            Console.WriteLine("Voiture : demarrer");
            this.a_etat = "demarrer"; }
    }
}
```

## 2.1- Objet : C#



```

1 référence
public void accelerer() {
    Console.WriteLine("Voiture : accelerer");
    if (this.a_etat == "demarrer") this.a_vitesse++;
}

1 référence
public void ralentir() {
    Console.WriteLine("Voiture : ralentir");
    if (this.a_vitesse > 0 ) this.a_vitesse--;
}

1 référence
public void arreter() {
    Console.WriteLine("Voiture : arreter");
    this.a_vitesse = 0;
    this.a_etat = "arreter";
}

3 références
public void afficher_vitesse() { Console.WriteLine("Vitesse : " + this.a_vitesse); }

0 références
~Voiture() => Console.WriteLine("Voiture : destruction");

protected string a_immatriculation;
protected string a_type;
protected string a_couleur;

protected string a_etat = "arreter";
protected UInt16 a_vitesse = 0;
}
    
```

## 2.4- La Classe : procédural C



```
typedef struct voiture_struct
{
    char* _immatriculation;
    char* _type;
    char* _couleur;

    char _etat[260];
    unsigned int _vitesse;
} Voiture_struct;
```

```
Voiture_struct* Voiture_construire(const char* immatriculation, const char* type, const char* couleur);
void Voiture_demarrer(Voiture_struct* voiture);
void Voiture_afficherVitesse(const Voiture_struct* voiture);
void Voiture_accelerer(Voiture_struct* voiture);
void Voiture_arreter(Voiture_struct* voiture);
void Voiture_detruire(Voiture_struct** voiture);
```

## 2.4- La Classe : procédural C



```
Voiture_struct* Voiture_construire(const char* immatriculation, const char* type, const char* couleur) {
    Voiture_struct* res = NULL;
    printf("Voiture : construction\n");
    if ( (immatriculation == NULL) || type == NULL || couleur == NULL ) exit(2);
    res = (Voiture_struct*) malloc( sizeof(Voiture_struct) );
    res->_immatriculation = (char*) malloc( sizeof(char) * strlen(immatriculation) );
    strcpy(res->_immatriculation, immatriculation);
    res->_type = (char*) malloc( sizeof(char) * strlen(type) );
    strcpy(res->_type, type);
    res->_couleur = (char*) malloc( sizeof(char) * strlen(couleur) );
    strcpy(res->_couleur, couleur);

    strcpy(res->_etat, "arreter");
    res->_vitesse = 0;
    return res;
}
```

## 2.4- La Classe : procédural C



```
void Voiture_demarrer(Voiture_struct* voiture)
{
    printf("Voiture : demarrer \n");
    strcpy(voiture->_etat, "demarrer");
}

void Voiture_afficherVitesse(const Voiture_struct* voiture) {
    printf("Vitesse : %d \n" , voiture->_vitesse);
}

void Voiture_accelerer(Voiture_struct* voiture)
{
    printf("Voiture : accelerer \n");
    if ( strcmp(voiture->_etat, "demarrer") == 0 )  voiture->_vitesse++;
}

void Voiture_ralentir(Voiture_struct* voiture) {
    printf("Voiture : ralentir\n");
    if ( voiture->_vitesse != 0 ) voiture->_vitesse--;
}

void Voiture_arreter(Voiture_struct* voiture) {
    printf("Voiture : arreter \n");
    strcpy(voiture->_etat, "arreter");
    voiture->_vitesse = 0;
}

void Voiture_detruire(Voiture_struct** voiture){
    printf("Voiture : destruction\n");
    if ( voiture == NULL) return;
    if ( *voiture == NULL) return;
    if ( (*voiture)->_immatriculation != NULL ) free((*voiture)->_immatriculation);
    if ( (*voiture)->_type != NULL ) free((*voiture)->_type);
    if ( (*voiture)->_couleur != NULL ) free((*voiture)->_couleur);
    free(*voiture);
    *voiture=NULL;
}
}
```



# 1. Naissance d'un objet (*constructeur*)

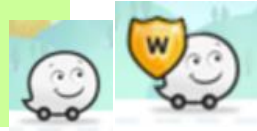
- Allouer de la mémoire
- Initialiser cette mémoire

# 2. Vie d'un objet

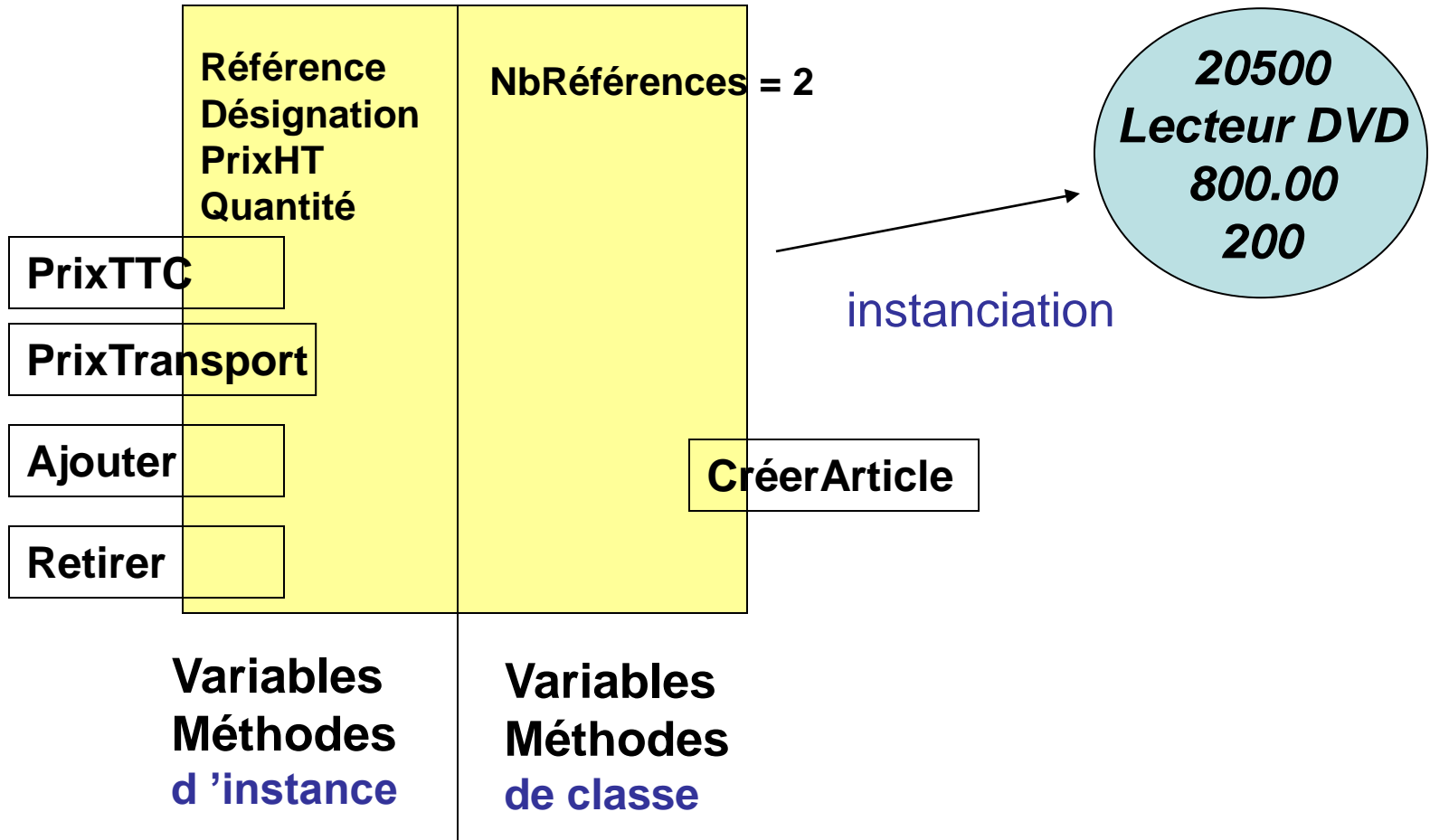
- Utilisation des méthodes en modification, sélection et itération

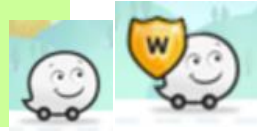
# 3. Mort d'un objet (*destructeur*)

- Libérer la mémoire allouée dynamiquement
- Rendre les ressources systèmes
- autres.....



# Méthodes, variables d'instance et de classe





### class POO

#### Voiture

- immatriculation: string
- couleur: string
- etat: Etat = "arreter"
- vitesse: int = 0
- NbDestruction: int = 0
- NbCreation: int = 0
- + afficherVitesse(): void
- + demarrer(): void
- + accelerer(): void
- + ralentir(): void
- + AfficherCreationDestruction(): void
- «constructor»
- + Voiture(): void
- «destructor»
- + ~Voiture(): void

Etat est soit "arreter" soit  
"demarrer"



## 2.4- La Classe : PHP



```
class Voiture{
    public static $NbCreation = 0;
    public static $NbDestruction = 0;
    public static function AfficherCreationDestruction() {
        print("Nombre de voiture creee ". Voiture::$NbCreation."</br>");
        print("Nombre de voiture detruite ". Voiture::$NbDestruction."</br>");
    }
    public $immatriculation ;
    public $typevoiture ;
    public $couleur ;
    public $etat = "arret" ;
    public $vitesse = 0 ;
    public function __construct($r,$v,$c){
        print ("Voiture : construction </br>" ) ;
        $this->immatriculation= $r;
        $this->typevoiture= $v;
        $this->couleur= $c;
        Voiture::$NbCreation++;
    }
    public function demarrer(){
        print("Voiture : demarrer </br>" ) ;
        return $this->etat = "demarrer";
    }
    public function arreter(){
        print ("Voiture : arreter </br>") ;
        return $this->etat ="arreter";
    }
    public function afficher_vitesse(){
        print ("Vitesse ". $this->vitesse."</br>");
    }
    public function accelerer(){
        print ("Voiture : accelerer </br>" ) ;
        if ( $this->etat =="demarrer" )
            $this->vitesse+=1;
    }
}
```

## 2.4- La Classe : PHP



```
public function ralentir() {
    print ("Voiture : ralentir </br>") ;
    if ( $this->etat == "demarrer" and $this->vitesse != 0 )
        $this->vitesse-=1;
}

public function __destruct() {
    print ("Voiture : destruction </br>") ;
    Voiture::$NbDestruction++;
}
}

print ("exemple 1 : PHP </br>" );
$voiture = new Voiture("305 XV 13","Ferrari","rouge");
print ($voiture->demarrer() . "</br>" ) ;
$voiture->afficher_vitesse() ;
print ($voiture->accelerer() . "</br>" ) ;
$voiture->afficher_vitesse() ;
print ($voiture->ralentir() . "</br>" ) ;
$voiture->afficher_vitesse() ;
print ($voiture->arreter() . "</br>" ) ;
$voiture->afficher_vitesse() ;
$voiture = null;

Voiture::AfficherCreationDestruction();
$voiture = new Voiture("305 XV 13","Ferrari","rouge");
Voiture::AfficherCreationDestruction();
```

## 2.4- La Classe : Java



```
public class Voiture {

    public static void ExempleJava() {
        Voiture maFerrari = new Voiture("305 XV 13", "Ferrari", "rouge");
        maFerrari.demarrer();
        maFerrari.afficherVitesse();
        maFerrari.accelerer();
        maFerrari.afficherVitesse();
        maFerrari.ralentir();
        maFerrari.afficherVitesse();
        maFerrari.arreter();
        maFerrari.afficherVitesse();
    }

    public static void main(String[] args) {
        System.out.println("Exemple 1 : Java");
        Voiture.AfficherCreateDestruction();
        Voiture.ExempleJava();
        System.gc();
        System.runFinalization();
        Voiture.AfficherCreateDestruction();
    }

    private final String a_type;
    private final String a_immatriculation;
    private String a_couleur;
    private String a_etat;
    private int a_vitesse;

    static int NbCreation = 0;
    static int NbDestruction = 0;
```

## 2.4- La Classe : Java



```
static void AfficherCreateDestruction() {
    System.out.println("Nombre de voiture creee\t:" + Voiture.NbCreation);
    System.out.println("Nombre de voiture detruite\t: " + Voiture.NbDestruction);
}

public Voiture(String immatriculation, String typevoiture, String couleur) {
    System.out.println("Voiture : constructeur");
    this.a_immatriculation = immatriculation;
    this.a_couleur = couleur;
    this.a_type = typevoiture;
    this.a_etat = "arreter";
    this.a_vitesse = 0;
    Voiture.NbCreation += 1;
}

public void demarrer() { ...4 lines }

public void afficherVitesse() { ...3 lines }

public void accelerer() { ...6 lines }

public void ralentir() { ...6 lines }

public void arreter() { ...5 lines }

@Override
protected void finalize() throws Throwable {
    System.out.println("Voiture : destruction ");
    super.finalize();
    Voiture.NbDestruction += 1;
}
}
```

## 2.4- La Classe : Python



```
class Voiture:
    NbCreation=0
    NbDestruction=0
    @classmethod
    def AfficherCreationDestruction(cls):
        print("Nombre de voiture creee      :",cls.NbCreation)
        print("Nombre de voiture detruite   :",cls.NbDestruction)
    pass
    vitesse=0
    etat="arreter"
    def __init__(self,immatriculation,typevoiture,couleur):
        print("Voiture : construction ")
        self.immatriculation=immatriculation
        self.type=typevoiture
        self.couleur=couleur
        self.__class__.NbCreation+=1
    pass
    def demarrer(self):
        print("Voiture : construction ")
        self.etat="demarrer"
    pass
    def afficherVitesse(self):
        print("Vitesse ",self.vitesse)
    pass
    def accelerer(self):
        print("Voiture : accelerer ")
        if ( self.etat == "demarrer"):
            self.vitesse+=1
    pass
```

## 2.4- La Classe : Python



```
def ralentir(self):
    print("Voiture : ralentir ")
    if ( self.etat == "demarrer" and self.vitesse > 0):
        self.vitesse-=1
    pass
def arreter(self):
    print("Voiture : arreter ")
    self.etat="arreter"
    self.vitesse=0
    pass
def __del__(self):
    print("Voiture : destruction ")
    self.__class__.NbDestruction+=1
```

```
def Exemple_01_python():
    print("Exemple 1 : python")
    maFerrari = Voiture("305 XV 13","Ferrari","rouge")
    maFerrari.demarrer();
    maFerrari.afficherVitesse();
    maFerrari.accelerer();
    maFerrari.afficherVitesse();
    maFerrari.ralentir();
    maFerrari.afficherVitesse();
    maFerrari.arreter();
    maFerrari.afficherVitesse();
    del maFerrari;
    Voiture.AfficherCreationDestruction()
    maFerrari2 = Voiture("305 XV 13","Ferrari","rouge")
    Voiture.AfficherCreationDestruction()
```

Exemple\_01\_python()

## 2.4- La Classe : C++



```
class Voiture
{
public:
    Voiture(std::string immatriculation, std::string type, std::string couleur);

    ~Voiture();
    void demarrer();
    void afficherVitesse() const;
    void accelerer();
    void ralentir();
    void arreter();

    static void AfficherCreationDestruction();

protected:
    const std::string _immatriculation;
    const std::string _type;
    std::string _couleur;

    std::string _etat;
    unsigned int _vitesse;

    static unsigned int NbCreation;
    static unsigned int NbDestruction;
};
```

## 2.4- La Classe : C++



```
unsigned int Voiture::NbCreation = 0;
unsigned int Voiture::NbDestruction = 0;

void Voiture::AfficherCreationDestruction() {
    std::cout << "Nombre de voiture creee      : " << Voiture::NbCreation << std::endl;
    std::cout << "Nombre de voiture detruite : " << Voiture::NbDestruction << std::endl;
}

Voiture::Voiture(std::string immatriculation, std::string type, std::string couleur):
    _immatriculation(immatriculation),
    _type(type),
    _couleur(couleur),
    _etat("arreter"),
    _vitesse(0)
{
    std::cout << "Voiture : construction " << endl;
    Voiture::NbCreation++;
}

Voiture::~Voiture()
{
    std::cout << "Voiture : destruction " << endl;
    Voiture::NbDestruction++;
}
|
```



## 2.4- Objet : C++



```
void Exemple01_Cpp()
{
    std::cout << "Exemple 1 : C++" << std::endl;
    Voiture* maFerrari = new Voiture("305 XV 13", "Ferrari", "rouge");

    maFerrari->demarrer();
    maFerrari->afficherVitesse();
    maFerrari->accelerer();
    maFerrari->afficherVitesse();
    maFerrari->ralentir();
    maFerrari->afficherVitesse();
    maFerrari->arreter();
    maFerrari->afficherVitesse();

    delete maFerrari;
}
```

## 2.4- Objet : C#



```
namespace POO
{
    11 références
    class Voiture
    {
        private static UInt16 NbCreation = 0;
        private static UInt16 NbDestruction = 0;

        3 références
        public static void AfficherCreationDestruction()
        {
            Console.WriteLine("Nombre de voiture creee   : " + Voiture.NbCreation);
            Console.WriteLine("Nombre de voiture detruite : " + Voiture.NbDestruction);
        }

        0 références
        public static void Main(string[] args)
        {
            Console.WriteLine("Exemple 1 : C#");
            Voiture.AfficherCreationDestruction();
            Voiture voiture = new Voiture("305 XV 13", "Ferrari", "rouge");
            Voiture.AfficherCreationDestruction();
            voiture.demarrer();
            voiture.afficher_vitesse();
            voiture.accelerer();
            voiture.afficher_vitesse();
            voiture.ralentir();
            voiture.afficher_vitesse();
            voiture.arreter();
            voiture = null;
            GC.Collect();
            GC.WaitForPendingFinalizers();
            Voiture.AfficherCreationDestruction();
            Console.Read();
        }
    }
}
```

## 2.4- Objet : C#



```
1 référence  
public void demarrer() ...  
1 référence  
public void accélérer() ...  
1 référence  
public void ralentir() ...  
1 référence  
public void arreter() ...  
3 références  
public void afficher_vitesse() { Console.WriteLine("Vitesse : " + this.a_vitesse); }  
  
protected string a_immatriculation;  
protected string a_type;  
protected string a_couleur;  
  
protected string a_etat = "arreter";  
protected UInt16 a_vitesse = 0;  
  
}  
}
```

## 2.4- La Classe : procédural C



```
static unsigned int Voiture_NbCreation = 0;
static unsigned int Voiture_NbDestruction = 0;

void Voiture_AfficherCreationDestruction() {
    printf("Nombre de voiture creee      : %d \n", Voiture_NbCreation );
    printf("Nombre de voiture detruite : %d \n", Voiture_NbDestruction);
}

Voiture_struct* Voiture_construire(const char* immatriculation, const char* type, const char* couleur) {
    Voiture_struct* res = NULL;
    printf("Voiture : construction\n");
    if ( (immatriculation == NULL) || type == NULL || couleur == NULL ) exit(2);
    res = (Voiture_struct*) malloc( sizeof(Voiture_struct));
    res->immatriculation = (char*) malloc(sizeof(char)*strlen(immatriculation));
    strcpy(res->immatriculation, immatriculation);
    res->_type = (char*) malloc(sizeof(char)*strlen(type));
    strcpy(res->_type, type);
    res->_couleur = (char*) malloc(sizeof(char)*strlen(couleur));
    strcpy(res->_couleur, couleur);

    strcpy(res->_etat, "arreter");
    res->_vitesse = 0;
    Voiture_NbCreation++;
    return res;
}

void Voiture_detruire(Voiture_struct** voiture) {
    printf("Voiture : destruction\n");
    if ( voiture == NULL) return;
    if ( *voiture == NULL) return;
    if ( (*voiture)->_immatriculation != NULL ) free((*voiture)->_immatriculation);
    if ( (*voiture)->_type != NULL ) free((*voiture)->_type);
    if ( (*voiture)->_couleur != NULL ) free((*voiture)->_couleur);
    free(*voiture);
    *voiture = NULL;
    Voiture_NbDestruction++;
}
```

## 2.4- La Classe : encapsulation des attributs & méthodes



En POO les attributs et méthodes d'une classe peuvent être visible depuis les instances de toutes les classes d'une application.

En POO il faudrait que :

- Les attributs ne soient lus ou modifiés que par l'intermédiaire de méthodes prenant en charge les vérifications et effets de bord éventuels.
- les méthodes "utilitaires" ne soient pas visibles, seules les fonctionnalités de l'objet, destinées à être utilisées par d'autres objets soient visibles.

## 2.4- La Classe : encapsulation des attributs & méthodes



**class POO**

### **PetiteCalculatrice**

- resultat: float

+ additioner(float, float): float

+ getResultat(): float

+ effacer(): void

«constructor»

+ PetiteCalculatrice(): void

«destructor»

+ ~PetiteCalculatrice(): void



[https://img.yugioh-card.com/ygo cms/ygo/all/uploads/Rulebook\\_v9\\_fr.pdf](https://img.yugioh-card.com/ygo cms/ygo/all/uploads/Rulebook_v9_fr.pdf)

## 2.4- La Classe : PHP



```
<?php
class PetiteCalculatrice{
    public $resultat = 0.0;
    public $nb1 = 0;
    public $nb2 = 0;

    public function __construct($nombre1, $nombre2){
        $this->nb1 = $nombre1;
        $this->nb2 = $nombre2;
    }
    public function additionner(){
        return $this->resultat = $this->nb1 + $this->nb2;
    }
    public function getResultat(){
        return $this->resultat;
    }
    public function effacer(){
        return $this->resultat = 0.0;
    }
    public function __destruct(){
        print ("Destruction de petite calculatrice " . "</br>");
    }
}
$PetiteCalculatrice = new PetiteCalculatrice(5,2);
print($PetiteCalculatrice->additionner() . "</br>");
?>
```



## 2.4- La Classe : Java



```
package ExempleDP;

public class PetiteCalculatrice {

    public static void main(String[] args) {
        PetiteCalculatrice cal = new PetiteCalculatrice();
        System.out.println( cal.additionner(5.,2.) );
    }

    private double res=0.0;

    public PetiteCalculatrice() {}
    public double additionner(double d1, double d0) {
        this.res = d1 + d0 ;
        return this.res;
    }

    public void effacer(){
        this.res=0.0;
    }

    public double getResultat() {
        return this.res;
    }

}
```

## 2.4- La Classe : Python



```
class PetiteCalculatrice:
    resultat=0.0
    def additionner(self,nb1,nb2):
        self.resultat=nb1+nb2
        return self.resultat
    def effacer(self):
        self.resultat=0.0
        pass
    def getResultat(self):
        return self.resultat

cal = PetiteCalculatrice()
print(cal.additionner(5.,2.))
```

## 2.4- La Classe : C++



```
class PetiteCalculatrice
{
private :
    double resultat;

public:
    PetiteCalculatrice():resultat(0){}

    double additionner(double nb1, double nb2)
    {
        return this->resultat = nb1 + nb2 ;
    }

    void effacer()
    {
        this->resultat = 0;
    }

    double getResultat() const
    {
        return this->resultat;
    }

};
```

## 2.4- La Classe : C#



```
=>using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;

namespace P00
{
    1 référence
    class PetiteCalculatrice
    {
        private double resultat = 0.0;
        0 références
        public PetiteCalculatrice() {}

        0 références
        public double additionner(double nb1, double nb2) => this.resultat = nb1 + nb2;

        0 références
        public void effacer() => this.resultat = 0;

        0 références
        public double getResultat() => this.resultat;
    }
}
```

## 2.4- La Classe : procédural C



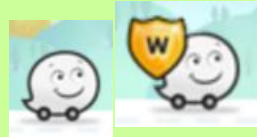
```
typedef struct
{
    double resultat;
} PetiteCalculatrice_struct;

PetiteCalculatrice_struct* PetiteCalculatrice(){
    PetiteCalculatrice_struct* res = (PetiteCalculatrice_struct*)malloc(sizeof(PetiteCalculatrice_struct));
    res->resultat = 0;
    return res;
}

double PetiteCalculatrice_additionner(PetiteCalculatrice_struct* calcul, double nb1, double nb2)
{
    return calcul->resultat = nb1 + nb2 ;
}

void PetiteCalculatrice_effacer(PetiteCalculatrice_struct* calcul)
{
    calcul->resultat = 0;
}

double PetiteCalculatrice_getResultat(const PetiteCalculatrice_struct* calcul)
{
    return calcul->resultat;
}
```



### La relation d'association

- exprime une dépendance sémantique entre des classes
- une association est bidirectionnelle
- une association a une cardinalité
  - 0 ou 1
  - un pour un
  - un pour n
  - n pour n

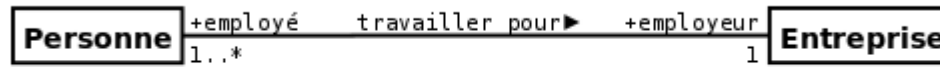


Figure 3.5 : Exemple d'association binaire.

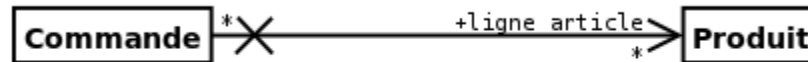


Figure 3.7 : Navigabilité.

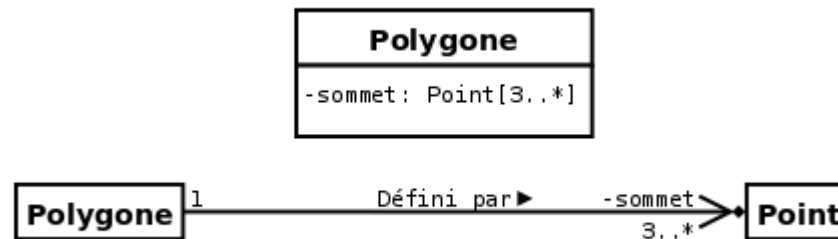
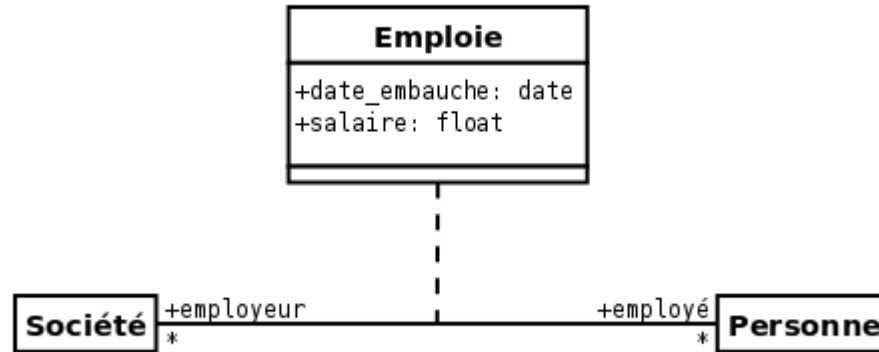


Figure 3.9 : Deux modélisations équivalentes.

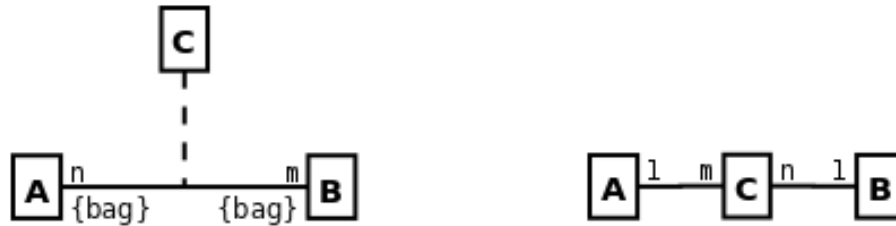
<https://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-classes#L3-3-3>



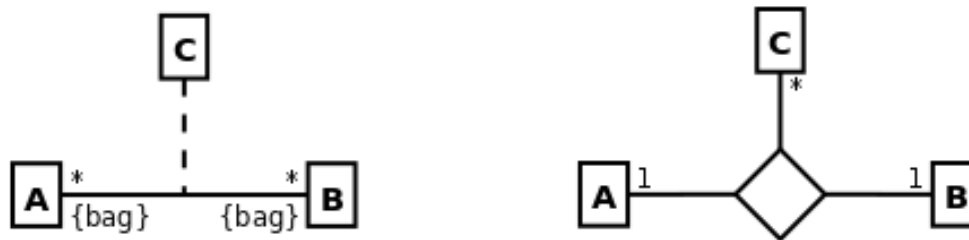
*Figure 3.11 : Exemple de classe-association.*

<https://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-classes#L3-3-3>





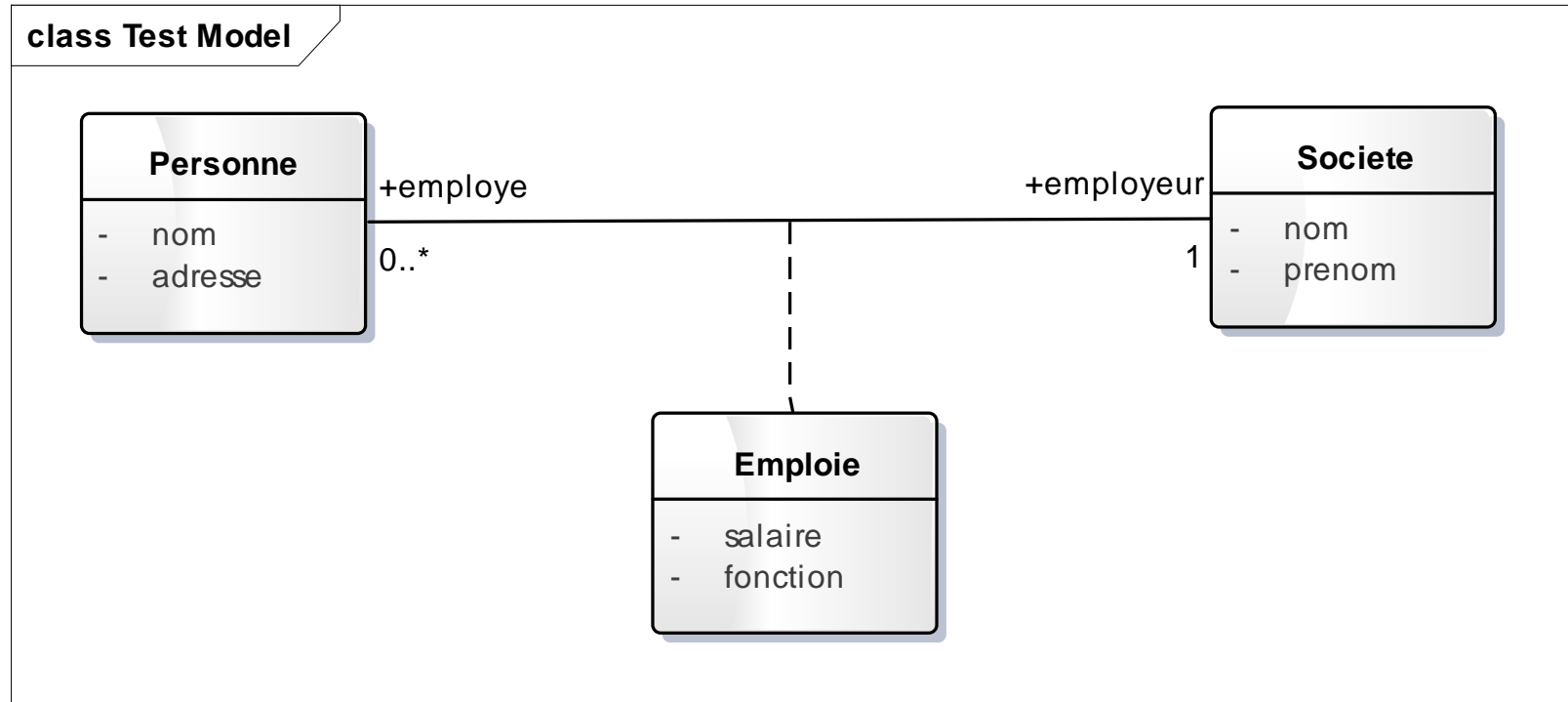
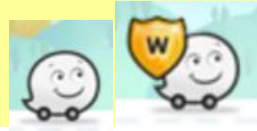
**Figure 3.14 : Deux modélisations modélisant la même information.**



**Figure 3.15 : Deux modélisations modélisant la même information.**

<https://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-classes#L3-3-3>

## 2.9- Les Relations d'association



## 2.4- La Classe : PHP



```
class Personne{  
    public $nom;  
    public $adresse;  
    public function __construct($n, $a){  
        $this->nom = $n;  
        $this->adresse = $a;  
    }  
    public function getNom(){  
        return $this->nom;  
    }  
    public function getAdresse(){  
        return $this->adresse;  
    }  
}
```

```
class Societe{  
    public $nom;  
    public $adresse;  
    public function __construct($n, $a){  
        $this->nom = $n;  
        $this->adresse = $a;  
    }  
    public function getNom(){  
        return $this->nom;  
    }  
    public function getAdresse(){  
        return $this->adresse;  
    }  
}
```

## 2.4- La Classe : PHP



```
class emploi{
    public $employeur;
    public $lienEmployeEmploiSalaire = [];
    public function __construct($e){
        $this->employeur = $e;
    }
    public function set($clef, $per,$salaire,$emploi){
        $this->lienEmployeEmploiSalaire[$clef][0] = $per;
        $this->lienEmployeEmploiSalaire[$clef][1] = $salaire;
        $this->lienEmployeEmploiSalaire[$clef][2] = $emploi;
    }
    public function getPersonne($clef){
        return $this->lienEmployeEmploiSalaire[$clef][0][0];
    }
    public function getSalaire($clef){
        return $this->lienEmployeEmploiSalaire[$clef][0][1] ;
    }
    public function getEmploi($clef){
        return $this->lienEmployeEmploiSalaire[$clef][0][2];
    }
    public function getSociete(){
        return $this->employeur ;
    }
}
```

## 2.4- La Classe : PHP



```
$personne = new Personne("david", "pertuis");  
$societe = new societe ("YNOV", "Aix-en-Provence");  
$emploi = new emploi($societe);  
$emploi->set("x0124589",$personne, "variable", "formateur");  
print($emploi->getPersonne("x0124589")->getNom() . "</br>");  
print($emploi->getSalaire("x0124589") . "</br>");  
print($emploi->getEmploi("x0124589") . "</br>");  
print($emploi->nbEmployer() . "</br>");  
print($emploi->getSociete()->getNom() . " " . $emploi->nbEmployer() . " employer.</br>");
```

## 2.4- La Classe : Java



```
public class Personne {  
  
    private String a_nom = null;  
    private String a_adresse = null;  
  
    Personne(String nom, String adresse) {  
        this.a_nom = nom;  
        this.a_adresse = adresse;  
    }  
  
    String getAdresse() {  
        return this.a_adresse;  
    }  
  
    String getNom() {  
        return this.a_nom;  
    }  
}
```

```
public class Societe {  
  
    private String a_nom = null;  
    private String a_adresse = null;  
  
    Societe(String nom, String adresse) {  
        this.a_nom = nom;  
        this.a_adresse = adresse;  
    }  
  
    String getAdresse() {  
        return this.a_adresse;  
    }  
  
    String getNom() {  
        return this.a_nom;  
    }  
}
```

## 2.4- La Classe : Java



```
public class Emploi {  
  
    private Societe a_employeur = null;  
    private HashMap<String, ArrayList<Object>> a_lienEmployeEmploiSalaire = null;  
  
    public Emploi(Societe employeur) {  
        this.a_employeur = employeur;  
        this.a_lienEmployeEmploiSalaire = new HashMap<>();  
    }  
  
    public void set(String cle, Personne personne, Double salaire, String emploi) {  
        ArrayList<Object> val = new ArrayList<>();  
        val.add(personne);  
        val.add(salaire);  
        val.add(emploi);  
        this.a_lienEmployeEmploiSalaire.put(cle, val);  
    }  
  
    public Personne getPersonne(String cle) {  
        return (Personne) this.a_lienEmployeEmploiSalaire.get(cle).get(0);  
    }  
  
    public Double getSalaire(String cle) {  
        return (Double) this.a_lienEmployeEmploiSalaire.get(cle).get(1);  
    }  
  
    public String getEmploi(String cle) {  
        return (String) this.a_lienEmployeEmploiSalaire.get(cle).get(2);  
    }  
}
```

## 2.4- La Classe : Java



```
public Personne getPersonne(String cle) {  
    return (Personne) this.a_lienEmployeEmploiSalaire.get(cle).get(0);  
}  
  
public Double getSalaire(String cle) {  
    return (Double) this.a_lienEmployeEmploiSalaire.get(cle).get(1);  
}  
  
public String getEmploi(String cle) {  
    return (String) this.a_lienEmployeEmploiSalaire.get(cle).get(2);  
}  
  
private int nbEmployer() {  
    return this.a_lienEmployeEmploiSalaire.size();  
}  
  
public Societe getSociete() {  
    return this.a_employeur;  
}  
  
public static void main(String[] args) { ...11 lines }  
}
```



## 2.4- La Classe : Java



```
public static void main(String[] args) {  
    Personne personne = new Personne("david", "pertuis");  
    Societe societe = new Societe("YNOV", "Aix");  
    Emploi emploi = new Emploi(societe);  
    emploi.set("x0123456789", personne, 500., "formateur");  
    System.out.println(emploi.getPersonne("x0123456789").getNom());  
    System.out.println(emploi.getSalaire("x0123456789"));  
    System.out.println(emploi.getEmploi("x0123456789"));  
  
    System.out.println(emploi.getSociete().getNom() + " " + emploi.nbEmployer() + " employer");  
}
```

## 2.4- La Classe : Python



```
class Personne:
    nom=""
    adresse=""
    def __init__(self,nom,adresse):
        self.nom =nom
        self.adresse=adresse
    def getAdresse(self):
        return self.adresse
    def getNom(self):
        return self.nom
```

```
class Societe:
    nom=""
    adresse=""
    def __init__(self,nom,adresse):
        self.nom =nom
        self.adresse=adresse
    def getAdresse(self):
        return self.adresse
    def getNom(self):
        return self.nom
```

## 2.4- La Classe : Python



```
class Emploi:
    employeur=None
    lienEmployeEmploiSalaire={}
    def __init__(self,employeur):
        self.employeur=employeur
    def set(self,cle, per,salaire,emploi ):
        self.lienEmployeEmploiSalaire.update({cle:[per,salaire,emploi]})
    def getPersonne(self, cle) :
        return self.lienEmployeEmploiSalaire[cle][0]
    def getSalaire(self, cle) :
        return self.lienEmployeEmploiSalaire[cle][1]
    def getEmploi(self, cle):
        return self.lienEmployeEmploiSalaire[cle][2]
    def nbEmployer(self) :
        return len(self.lienEmployeEmploiSalaire)
    def getSociete(self) :
        return self.employeur

personne =Personne("david","pertuis")
societe=Societe("YNOV","Aix")
emploi =Emploi(societe)
emploi.set("x0124589",personne,"variable","formateur");
print(emploi.getPersonne("x0124589").getNom())
print(emploi.getSalaire("x0124589"))
print(emploi.getEmploi("x0124589"))
print(emploi.nbEmployer())
print(emploi.getSociete().getNom(), " ", emploi.nbEmployer()," employer")
```

## 2.4- La Classe : C++



```
class Personne {
private :
    std::string _nom;
    std::string _adresse;
public:
    Personne(std::string nom, std::string adresse) : _nom(nom), _adresse(adresse) {}
    Personne(const Personne& per) : _nom(per._nom), _adresse(per._adresse) {}
    std::string getAdresse() const { return this->_adresse; }
    std::string getNom() const { return this->_nom; }
};

class Societe {
private :
    std::string _nom;
    std::string _adresse;
public:
    Societe(std::string nom, std::string adresse) : _nom(nom), _adresse(adresse) {}
    Societe(const Societe& soc) : _nom(soc._nom), _adresse(soc._adresse) {}
    std::string getAdresse() const { return this->_adresse; }
    std::string getNom() const { return this->_nom; }
};
```

## 2.4- La Classe : C++



```
class Emploi {
private :
    Societe _employeur;
    std::map< unsigned, Personne> _employe;
    std::map< unsigned, std::string> _emploi;
    std::map< unsigned, unsigned int> _salaire;

public:

    Emploi(std::string employeur);
    void set(unsigned cle, Personne per, unsigned int salaire, std::string emploi );

    Personne getPersonne(unsigned cle) const { return this->_employe.at(cle); }
    unsigned int getSalaire(unsigned cle) const { return this->_salaire.at(cle); }
    std::string getEmploi(unsigned cle) const { return this->_emploi.at(cle); }

    unsigned nbEmployer() const { return this->_emploi.size(); }
    Societe getSociete() const { return this->_employeur; }

};
```

```
class Personne
{
    private string a_nom;
    private string a_adresse;

    1 référence
    public Personne(string nom, string adresse)
    {
        this.a_nom = nom;
        this.a_adresse = adresse;
    }
    0 références
    public string Adresse { get => a_adresse; set => a_adresse = value; }
    1 référence
    public string Nom { get => a_nom; set => a_nom = value; }
}

6 références
class Societe
{
    private string a_nom;
    private string a_adresse;

    1 référence
    public Societe(string nom, string adresse)
    {
        this.a_nom = nom;
        this.a_adresse = adresse;
    }
    0 références
    public string Adresse { get => a_adresse; set => a_adresse = value; }
    1 référence
    public string Nom { get => a_nom; set => a_nom = value; }
}
```

```

class Emploi
{
    private Societe a_employeur;
    private Dictionary<string, Personne> a_employe= new Dictionary< string, Personne>();
    private Dictionary<string, string> a_emploi = new Dictionary<string, string>();
    private Dictionary<string, UInt32> a_salaire =new Dictionary<string, UInt32>();

    1 référence
    public Emploi(Societe employeur) => this.a_employeur = employeur;
    1 référence
    public void Set(string numSecu, Personne employe, string emploi, UInt32 salaire)
    {
        this.a_employe[numSecu] = employe;
        this.a_emploi[numSecu] = emploi;
        this.a_salaire[numSecu] = salaire;
    }

    1 référence
    public Societe GetEmployeur() => this.a_employeur;

    1 référence
    public Personne GetPersonne(string numSecu) { return this.a_employe[numSecu]; }
    0 références
    public string GetEmploi(string numSecu) { return this.a_emploi[numSecu]; }
    1 référence
    public UInt32 GetSalaire(string numSecu) { return this.a_salaire[numSecu]; }

    0 références
    public int NbEmployer => this.a_employe.Count;

    0 références
    public static void Main(string[] args)
    {
        Console.WriteLine("Exemple 2 : C#");
        Personne personne = new Personne("david", "pertuis");
        Societe societe = new Societe("YNOV", "Aix");
        Emploi emploi = new Emploi(societe);
        emploi.Set("x0123456789", personne, "formateur", 5000000);
        Console.WriteLine(emploi.GetEmployeur().Nom);
        Console.WriteLine(emploi.GetPersonne("x0123456789").Nom);
        Console.WriteLine(emploi.GetSalaire("x0123456789"));

        GC.Collect();
        GC.WaitForPendingFinalizers();
        Console.Read();
    }
}

```

## 2.4- La Classe : procédural C



```
typedef struct
```

```
{  
    char _nom[256];  
    char _adresse[256];  
} Personne_struct ;
```

```
Personne_struct *CreatePersonne(const char* nom, const char* adresse)  
{  
    Personne_struct *per=(Personne_struct *) malloc(sizeof(Personne_struct));  
    strcpy(per->_nom,nom);  
    strcpy(per->_adresse,adresse);  
    return per;  
}
```

```
Personne_struct * CopyPersonne(const Personne_struct* per){ return CreatePersonne(per->_nom,per->_adresse);}  
std::string Personne_getAdresse(const Personne_struct* per){ return per->_adresse; }  
std::string Personne_getNom(const Personne_struct* per) { return per->_nom; }
```

```
typedef struct
```

```
{  
    char _nom[256];  
    char _adresse[256];  
} Societe_struct ;
```

```
Societe_struct *CreateSociete(const char* nom, const char* adresse)  
{  
    Societe_struct *per=(Societe_struct *) malloc(sizeof(Societe_struct));  
    strcpy(per->_nom,nom);  
    strcpy(per->_adresse,adresse);  
    return per;  
}
```



## 2.4- La Classe : procédural C



```
#define NBMAX 1000
typedef struct
{
    Societe_struct* _employeur;
    Personne_struct* _employe[NBMAX];
    char _emploi[NBMAX][256];
    unsigned _salaire[NBMAX];
    unsigned _cle;
} Emploi_struct;

Eemploi_struct* CreateEmploi( std::string employeur) {
    Emploi_struct* lien=(Eemploi_struct*) malloc(sizeof(Eemploi_struct));
    lien->_employeur = employeur;
    return lien;
}

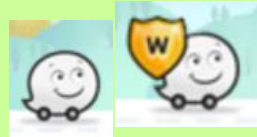
void Emploi_set(Eemploi_struct* lien,Personne_struct* per,unsigned int salaire,const char* emploi )
{
    if ( lien->_cle >= NBMAX) {printf("Nombre maximum de personnel atteint\n");return;}
    lien->_employe[lien->_cle]=per;
    lien->_salaire[lien->_cle]=salaire;
    strcpy(lien->_emploi[lien->_cle],emploi);
    lien->_cle++;
}

Personne_struct* Emploi_getPersonne(const Emploi_struct* lien,unsigned cle) {return lien->_employe[cle];}
unsigned int Emploi_getSalaire(const Emploi_struct* lien,unsigned cle) { return lien->_salaire[cle]; }
const char* Emploi_getEmploi(const Emploi_struct* lien,unsigned cle) { return lien->_emploi[cle];}
unsigned Emploi_nbEmployer(const Emploi_struct* lien){ return lien->_cle; }
Societe_struct* Emploi_getSociete(const Emploi_struct* lien) { return lien->_employeur; }
```

La composition peut être vue comme une relation “**fait partie de**” (“part of”), c’est à dire que si un objet B fait partie d’un objet A alors B ne peut pas exister sans A. Ainsi **si A disparaît alors B également.**

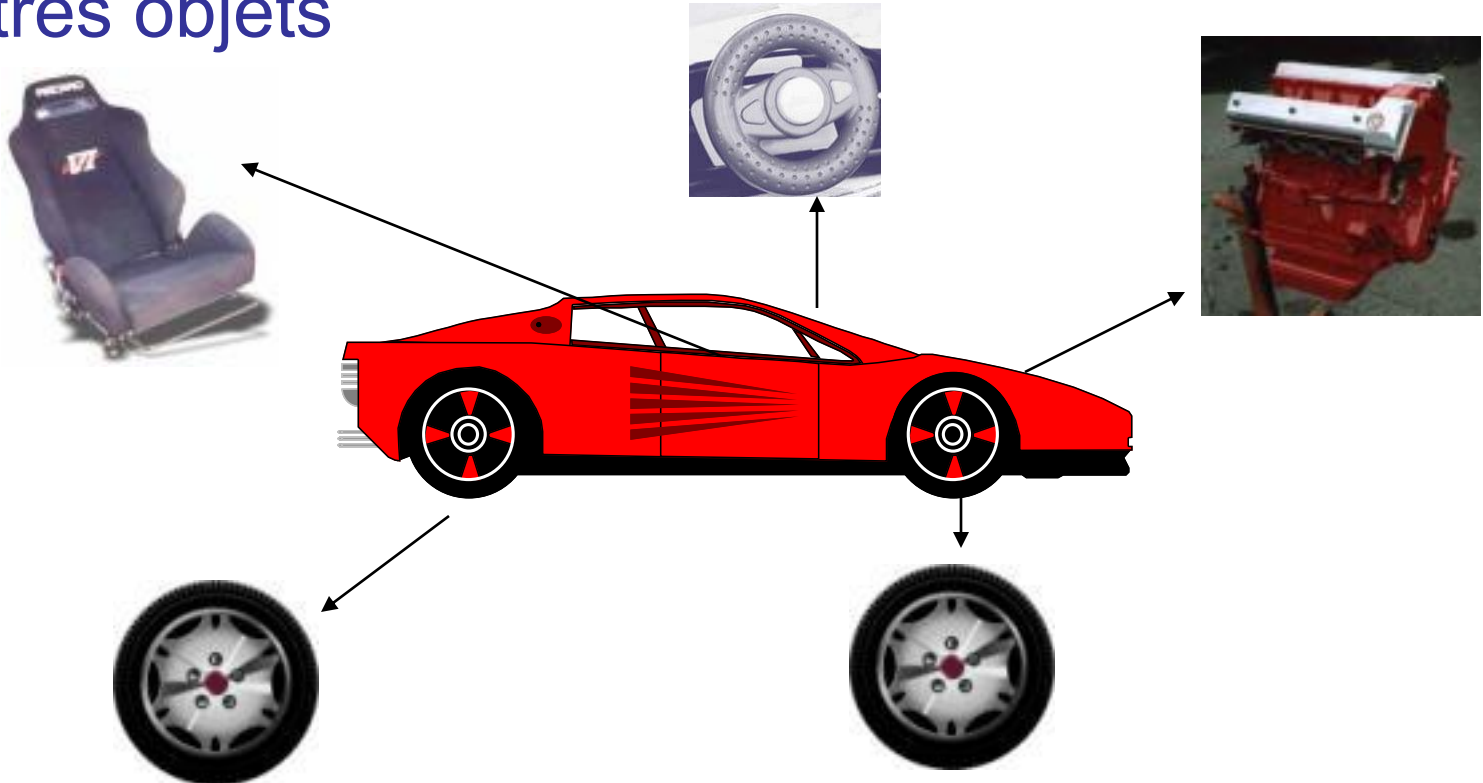
L’agrégation quant à elle est vue comme une relation de type “**a un**” (“has a”), c’est à dire que si un objet A a un objet B alors **B peut vivre sans A**

<http://www.lechatcode.com/architecture/agregation-et-composition-cest-quoi/>

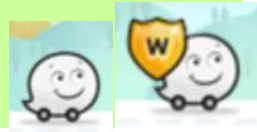


# Agrégation

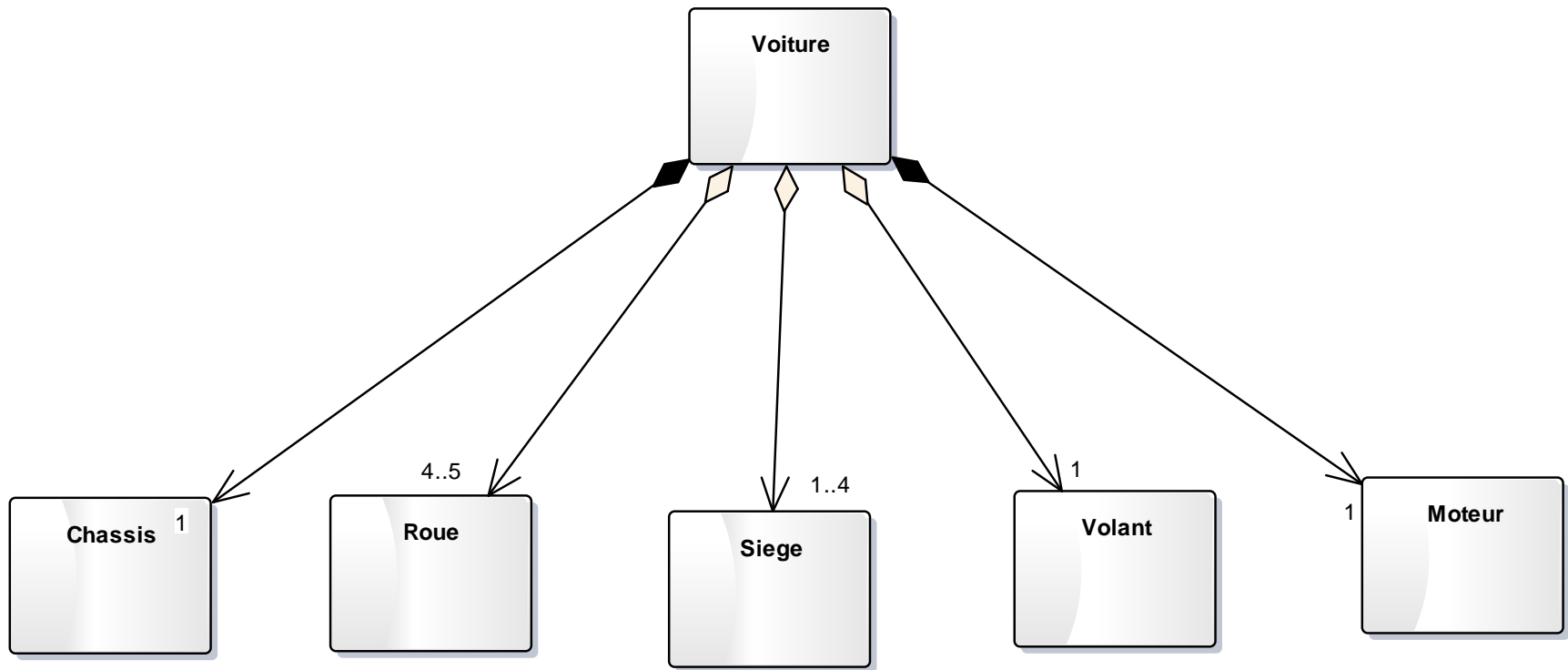
Les objets peuvent contenir des références à d'autres objets



## 2.6- L'Agrégation & Composition



class Class Model



**Se préparer à jouer**

### Le Tapis de Jeu

Le Tapis de Jeu est là pour vous aider à agencer vos cartes pendant un Duel. Lorsque vous utilisez vos cartes, vous les placez sur le Tapis de Jeu. Les différents types de cartes sont placés dans différentes Zones.

Chaque Dueliste doit posséder son propre Tapis de Jeu, qui devra être placé en face de celui de l'adversaire lors des Duels. L'ensemble des deux tapis forme "le Terrain". Le Tapis de Jeu fourni représente uniquement votre moitié de Terrain. Les cartes que vous "contrôlez" sont les cartes placées sur votre Terrain.

Vous pouvez également faire des Duels sans utiliser de Tapis de Jeu, à partir du moment où vous vous rappelez où placer les cartes.

**1**  
**Zone Monstre**

C'est là où vous mettez vos monstres lorsque vous les jouez. Vous pouvez y avoir jusqu'à 5 cartes au même moment. Il y a 3 façons de positionner vos Cartes Monstre : Position d'Attaque face recto, Position de Défense face recto et Position de Défense face verso. Placez la carte verticalement pour indiquer une Position d'Attaque, et horizontalement pour une Position de Défense.

**2**  
**Zone Magie & Piège**

C'est là où vous mettez vos cartes Magie et Piège. Vous pouvez y avoir jusqu'à 5 cartes au même moment. Vous les placez face recto pour les activer, ou face verso. Dans la mesure où une Carte Magie est placée dans cette zone lorsqu'elle est activée, aucune Carte Magie supplémentaire ne peut être jouée si les 5 emplacements sont occupés.

**3**  
**Cimetière**

Lorsque les Cartes Monstre sont détruites ou lorsque les Cartes Magie et Piège sont utilisées, elles sont envoyées face recto à cet emplacement. Le contenu du Cimetière de chaque joueur est une donnée publique et votre adversaire peut regarder le contenu du vôtre à n'importe quel moment durant le Duel. L'ordre des cartes dans le Cimetière ne doit pas être modifié.

**4**  
**Zone Terrain**

Des Cartes Magie spécifiques appelées Cartes Magie de Terrain sont jouées dans cet emplacement. Chaque joueur peut avoir 1 Carte Magie de Terrain sur son Terrain. Pour utiliser une autre Carte Magie de Terrain, vous devez envoyer la précédente au Cimetière. Les Cartes Magie de Terrain n'entrent pas en compte dans la limite de 5 cartes de votre Zone Magie & Piège.

**5**  
**Zone Extra Deck**

Placez votre Extra Deck face verso à cet emplacement. Vous pouvez regarder le contenu de votre Extra Deck à tout moment durant la partie. Cette zone était auparavant réservée au Fusion Deck. Tous les effets de carte qui s'appliquaient au Fusion Deck s'appliquent dorénavant à l'Extra Deck.

**6**  
**Zone Pendule**

Lorsque vous activez une Carte Monstre Pendule comme Carte Magie, vous la placez face recto dans cet emplacement. Les Cartes Monstre Pendule jouées comme Cartes Magie n'entrent pas en compte dans la limite de 5 cartes de votre Zone Magie & Piège et Zone Monstre (voir page 12 pour plus de détails sur les Cartes Monstre Pendule).

**Se préparer à jouer**

**1**  
**Zone Monstre**

C'est là où vous mettez vos monstres lorsque vous les jouez. Vous pouvez y avoir jusqu'à 5 cartes au même moment. Il y a 3 façons de positionner vos Cartes Monstre : Position d'Attaque face recto, Position de Défense face recto et Position de Défense face verso. Placez la carte verticalement pour indiquer une Position d'Attaque, et horizontalement pour une Position de Défense.

**2**  
**Zone Magie & Piège**

C'est là où vous mettez vos cartes Magie et Piège. Vous pouvez y avoir jusqu'à 5 cartes au même moment. Vous les placez face recto pour les activer, ou face verso. Dans la mesure où une Carte Magie est placée dans cette zone lorsqu'elle est activée, aucune Carte Magie supplémentaire ne peut être jouée si les 5 emplacements sont occupés.

**3**  
**Cimetière**

Lorsque les Cartes Monstre sont détruites ou lorsque les Cartes Magie et Piège sont utilisées, elles sont envoyées face recto à cet emplacement. Le contenu du Cimetière de chaque joueur est une donnée publique et votre adversaire peut regarder le contenu du vôtre à n'importe quel moment durant le Duel. L'ordre des cartes dans le Cimetière ne doit pas être modifié.

**4**  
**Zone Deck**

Votre Deck est placé face verso dans cet emplacement. Les joueurs y piochent les cartes et les ajoutent à leur main. Si l'effet d'une carte vous demande de révéler des cartes de votre Deck ou de regarder son contenu, mélangez le Deck et remplacez-le à cet endroit après avoir résolu l'effet.

**5**  
**Zone Terrain**

Des Cartes Magie spécifiques appelées Cartes Magie de Terrain sont jouées dans cet emplacement. Chaque joueur peut avoir 1 Carte Magie de Terrain sur son Terrain. Pour utiliser une autre Carte Magie de Terrain, vous devez envoyer la précédente au Cimetière. Les Cartes Magie de Terrain n'entrent pas en compte dans la limite de 5 cartes de votre Zone Magie & Piège.

**6**  
**Zone Extra Deck**

Placez votre Extra Deck face verso à cet emplacement. Vous pouvez regarder le contenu de votre Extra Deck à tout moment durant la partie. Cette zone était auparavant réservée au Fusion Deck. Tous les effets de carte qui s'appliquaient au Fusion Deck s'appliquent dorénavant à l'Extra Deck.

**7**  
**Zone Pendule**

Lorsque vous activez une Carte Monstre Pendule comme Carte Magie, vous la placez face recto dans cet emplacement. Les Cartes Monstre Pendule jouées comme Cartes Magie n'entrent pas en compte dans la limite de 5 cartes de votre Zone Magie & Piège et Zone Monstre (voir page 12 pour plus de détails sur les Cartes Monstre Pendule).

**7**  
**Zone Pendule**

Lorsque vous activez une Carte Monstre Pendule comme Carte Magie, vous la placez face recto dans cet emplacement. Les Cartes Monstre Pendule jouées comme Cartes Magie n'entrent pas en compte dans la limite de 5 cartes de votre Zone Magie & Piège et Zone Monstre (voir page 12 pour plus de détails sur les Cartes Monstre Pendule).

[https://img.yugioh-card.com/ygo\\_cms/ygo/all/uploads/Rulebook\\_v9\\_fr.pdf](https://img.yugioh-card.com/ygo_cms/ygo/all/uploads/Rulebook_v9_fr.pdf)

## 2.4- La Classe : PHP



```
class Chassis{};
class Siege{};
class Volant{};
class Moteur{};
class Roue{};

class Voiture{
    public function __construct() {
        $this->roue = array(new Roue(), new Roue(), new Roue(), new Roue());
        $this->siege = array(new Siege(), new Siege(), new Siege(), new Siege());
        $this->chassis = new Chassis();
        $this->volant = new Volant();
        $this->moteur = new Moteur();
    }
    public function __destruct() {
    }
}
```

## 2.4- La Classe : Java



```
public class Moteur {  
}  
  
public class Chassis {  
  
}  
  
public class Roue {  
  
}  
  
public class Siege {  
  
}  
  
public class Volant {  
  
}
```

```
public class Voiture {  
  
    Siege a_siege[] = new Siege[4];  
    Roue a_roue[] = new Roue[4];  
    Moteur a_moteur = new Moteur();  
    Volant a_volant = new Volant();  
    Chassis a_chassis = new Chassis();  
}
```

## 2.4- La Classe : Python



```
class Chassis:
    pass

class Roue:
    pass

class Siege :
    pass

class Volant :
    pass

class Moteur :
    pass

class Voiture:
    def __init__(self):
        self.chassis = Chassis()
        self.roue = [Roue(),Roue(),Roue(),Roue()]
        self.siege = [Siege(),Siege(),Siege(),Siege()]
        self.volant = Volant()
        self.moteur = Moteur ()
    def __del__(self):
        del self.chassis
        del self.moteur
```



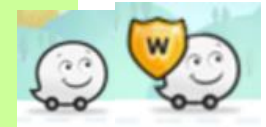
## 2.4- La Classe : C++



```
class Chassis{};
class Roue{};
class Siege{};
class Volant{};
class Moteur{};

class Voiture
{
private:
    Chassis* _chassis;
    Roue* _roue[4];
    Siege* _siege[4];
    Volant* _volant;
    Moteur* _moteur;
public:
    Voiture();
    ~Voiture() {
        delete this->_chassis;
        delete this->_moteur;
    }
};
```

## 2.4- La Classe : C#



```

3 references
class Chassis { }

3 références
class Roue { }

3 références
class Siege { }

3 références
class Volant { }

3 références
class Moteur { }

4 références
class Voiture
{
    private Chassis a_chassis = new Chassis();
    private Roue[] a_roue = new Roue[4];
    private Siege[] a_siege = new Siege[4];
    private Volant a_volant = new Volant();
    private Moteur a_moteur = new Moteur();

    1 référence
    public Voiture()...

    0 références
    ~Voiture()
    {
        this.a_chassis = null;
        this.a_moteur = null;
    }

    0 références
    public Chassis aChassis { get => a_chassis; set => a_chassis = value; }
    0 références
    public Roue[] aRoue { get => a_roue; set => a_roue = value; }
    0 références
    public Siege[] aSiege { get => a_siege; set => a_siege = value; }
    0 références
    public Volant aVolant { get => a_volant; set => a_volant = value; }
    0 références
    public Moteur aMoteur { get => a_moteur; set => a_moteur = value; }
}

```

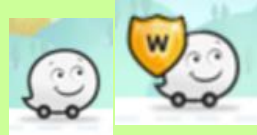
## 2.4- La Classe : procédural C



```
typedef struct {} Chassis_struct;
typedef struct {} Roue_struct;
typedef struct {} Siege_struct;
typedef struct {} Volant_struct;
typedef struct {} Moteur_struct;

typedef struct
{
    Chassis_struct* _chassis;
    Roue_struct* _roue[4];
    Siege_struct* _siege[4];
    Volant_struct* _volant;
    Moteur_struct* _moteur;
} Voiture_struct;

Voiture_struct* CreateVoiture();
void Voiture_detruire(Voiture_struct* vo)
{
    free( vo->_chassis);
    free( vo->_moteur);
}
```

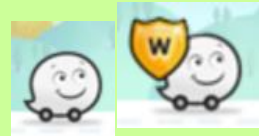


### L 'Héritage

Par héritage, une classe dérivée possède les attributs et les méthodes de la superclasse.

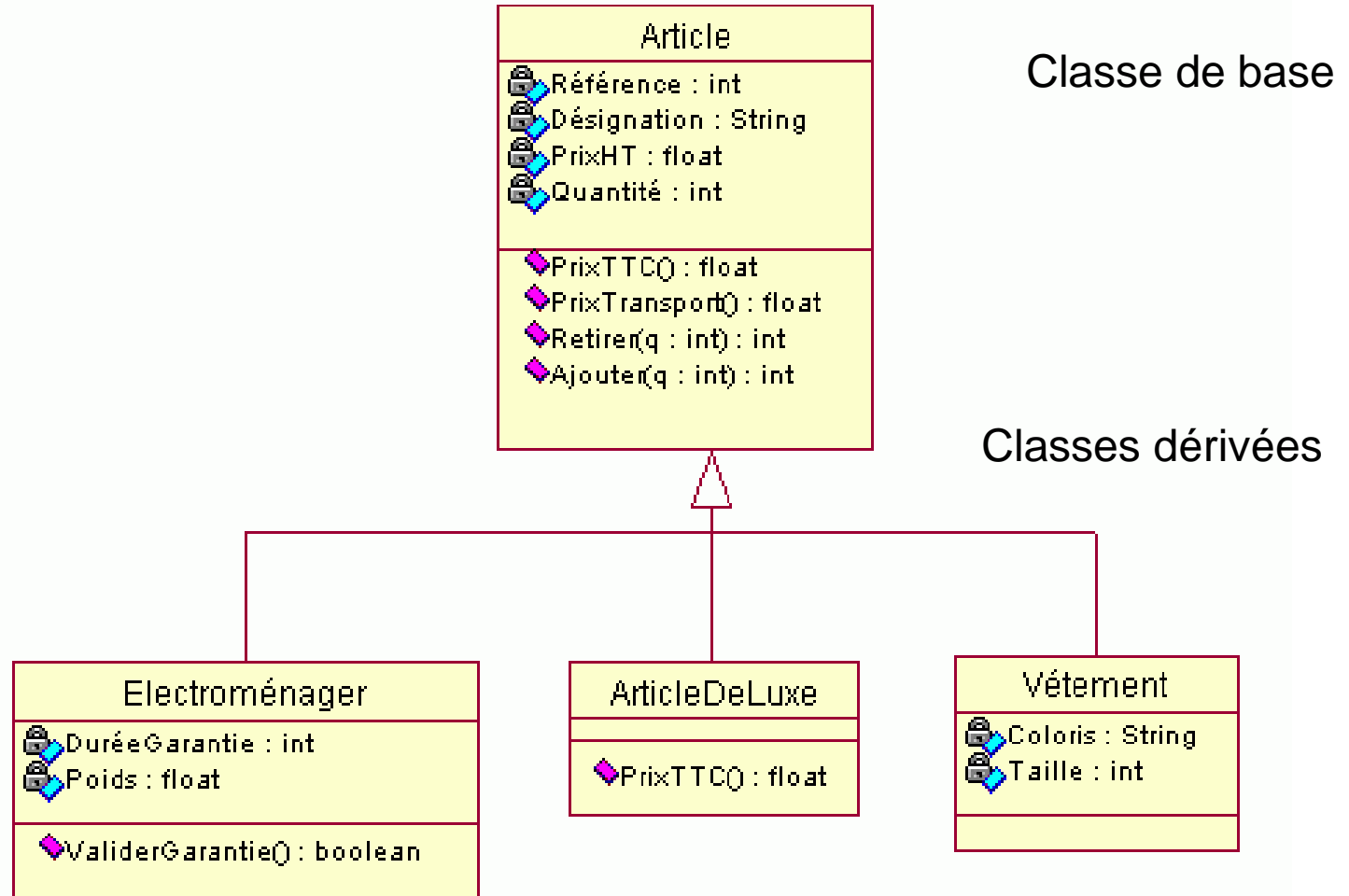
- ⇒ La classe dérivée possède les attributs et méthodes de la classe de base,
- ⇒ la classe dérivée peut en ajouter ou en masquer,
- ⇒ facilite la programmation par raffinement,
- ⇒ facilite la prise en compte de la spécialisation.

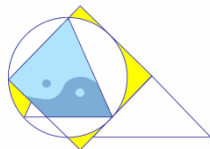
## 2.5- L 'Héritage



↑  
GENERALISATION

↓  
SPECIALISATION



**«« Monstres à Effet**

Un Monstre à Effet est un monstre qui possède des capacités spéciales.

Les effets de ces monstres sont répartis dans quatre catégories:

- Effet Continu
- Effet d'Ignition
- Effet Rapide
- Effet Déclencheur (incluant l'effet Flip)

**«« Monstres Normaux**

Ce sont les Cartes Monstre de base, sans capacité spéciale. La plupart des Monstres Normaux ont des valeurs d'Attaque et de Défense supérieures aux Monstres à Effet, au lieu d'avoir des pouvoirs spéciaux.

**«« Monstres Xyz**

Les Monstres Xyz sont un genre de monstres très puissants ! Vous pouvez invoquer un Monstre Xyz quand vous contrôlez des monstres d'un même Niveau. Les Monstres Xyz sont placés dans votre Extra Deck, et non votre Main Deck, et se tiennent prêts à votre appel.

**«« Cartes Monstre Pendule**

Les Cartes Monstre Pendule sont un nouveau genre de cartes qui peuvent être jouées comme des Monstres ou des Magies ! Vous pouvez les invoquer comme des monstres pour attaquer ou défendre, ou les activer comme des Cartes Magie dans vos Zones Pendule pour activer des compétences supplémentaires spéciales et vous permettre d'invoquer par Pendulation !

**«« Monstres Synchro**

Les Monstres Synchro sont placés dans l'Extra Deck, séparés du Main Deck. Vous pouvez invoquer spécialement sur le Terrain un puissant Monstre Synchro en quelques secondes en utilisant correctement le Niveau de vos monstres. Ils peuvent être invoqués par Synchronisation depuis l'Extra Deck en envoyant au Cimetière depuis votre Terrain 1 monstre "Synchroiseur" face recto et le nombre de votre choix de monstres non-Synchroiseur face recto, dont la somme des Niveaux est exactement égale au Niveau du Monstre Synchro.

**«« Monstres Synchroiseur pour Invocation Synchro**

Pour invoquer par Synchronisation un Monstre Synchro, vous aurez besoin d'un Synchroiseur (cherchez le mot "Synchroiseur" à côté de son Type). Le monstre Synchroiseur et les autres monstres face recto utilisés pour l'Invocation Synchro sont appelés Matériels de Synchro. La somme de leurs Niveaux est le Niveau du Monstre Synchro que vous pouvez invoquer.

**«« Monstres de Fusion**

Les Monstres de Fusion sont également placés dans votre Extra Deck (pas dans votre Main Deck). Ils sont invoqués en utilisant les monstres spécifiques listés sur la carte (appelés Matériels de Fusion) en même temps qu'une carte d'Invocation comme "Polymérisation". Leurs valeurs d'Attaque sont souvent très élevées et la majorité dispose également de capacités spéciales.

**«« Monstres Rituel**

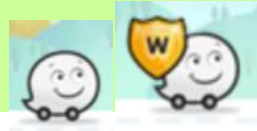
Les Monstres Rituels sont des monstres spéciaux qui sont invoqués spécialement à l'aide d'une Carte Magie Rituelle spécifique, ainsi qu'un Sacrifice requis. Les Cartes Monstre Rituel sont placées dans le Main Deck et ne peuvent pas être invoquées à moins d'avoir l'ensemble des cartes nécessaires dans votre main ou sur le Terrain. Les Monstres Rituels ont généralement des valeurs d'ATK et de DEF importantes, et certains d'entre eux ont des capacités spéciales, tout comme les Monstres de Fusion.

[https://img.yugioh-card.com/ygo\\_cms/ygo/all/uploads/Rulebook\\_v9\\_fr.pdf](https://img.yugioh-card.com/ygo_cms/ygo/all/uploads/Rulebook_v9_fr.pdf)





[https://img.yugioh-card.com/ygo cms/ygo/all/uploads/Rulebook\\_v9\\_fr.pdf](https://img.yugioh-card.com/ygo cms/ygo/all/uploads/Rulebook_v9_fr.pdf)



# Le Polymorphisme

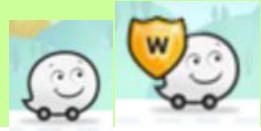
C 'est un concept selon lequel le même nom peut désigner des méthodes différentes.

La méthode désignée dépend de l'objet auquel on s'adresse.

- Le polymorphisme **Statique**: l'objet est connu à la compilation
- Le polymorphisme **Dynamique**: l'objet n'est connu qu'ultérieurement. L'identité ne sera qualifiée qu'au moment de l'exécution.



## 2.7- Le Polymorphisme



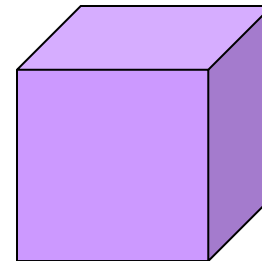
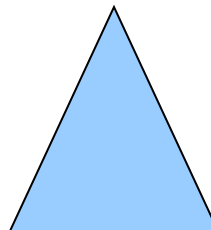
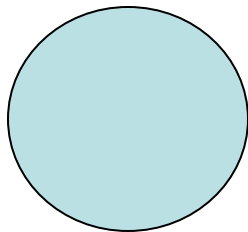
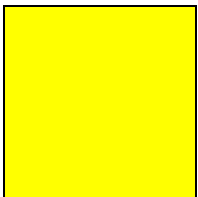
### STATIQUE

```
class Forme
{ afficher(); }
class Rectangle
{afficher();}
class Cercle
{afficher();}
```

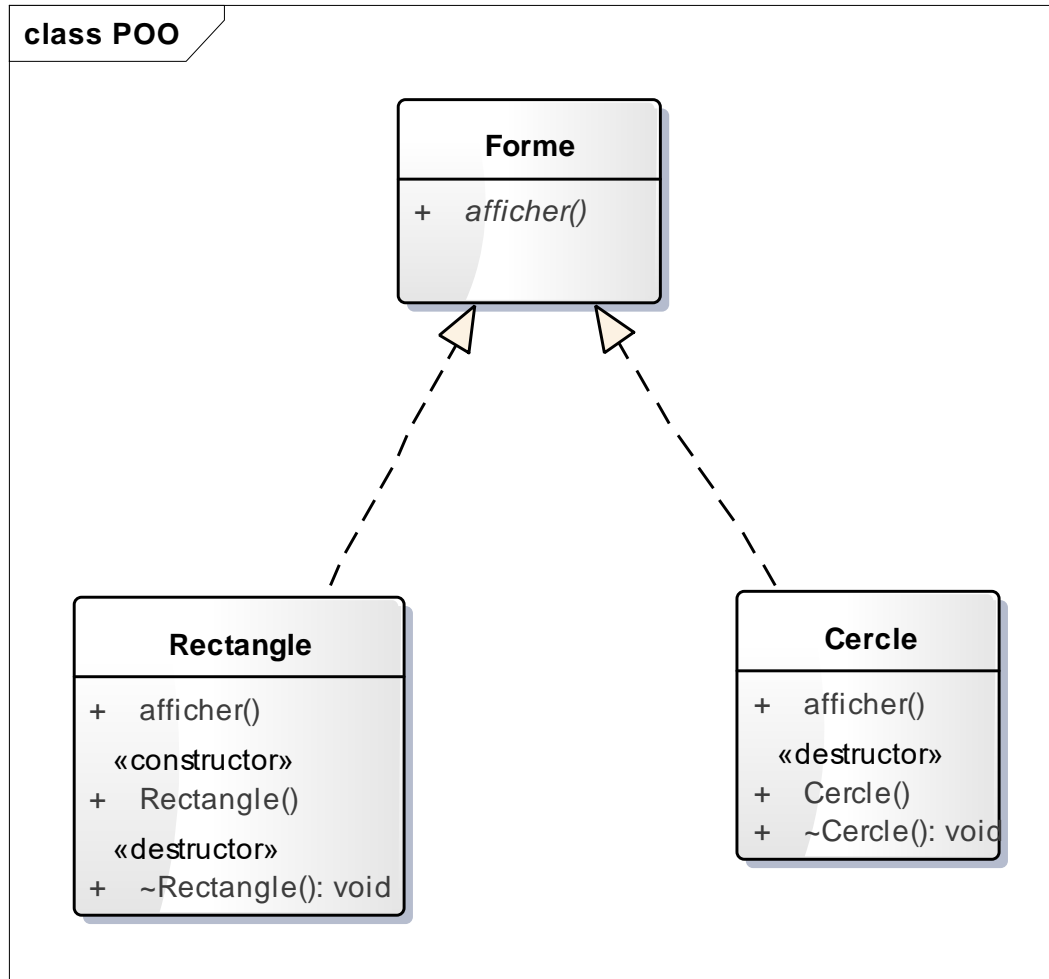
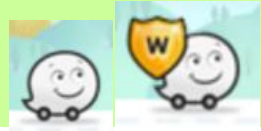
```
un Rectangle.afficher();
un Cercle.afficher();
```

### DYNAMIQUE

```
forme.afficher();
```



## 2.7- Le Polymorphisme



## 2.4- La Classe : PHP



<?php

```
abstract class Forme{
    abstract public function Afficher();
}

class Rectangle{
    public function Afficher(){
        print ("Je suis un rectangle </br>");
    }
}

class Cercle{
    public function Afficher(){
        print ("Je suis un cercle </br>");
    }
}

function afficherForme($f){
    $f->Afficher();
}

$cercle = new Cercle;
$rectangle = new Rectangle;
$cercle->Afficher();
$rectangle->Afficher();
echo "</br>";
afficherForme($rectangle);
afficherForme($cercle);
```

?>

Je suis un cercle  
Je suis un rectangle

Je suis un rectangle  
Je suis un cercle

## 2.4- La Classe : Java



```
public interface Forme {  
    public abstract void afficher();  
}
```

```
public class Rectangle implements Forme {  
  
    @Override  
    public void afficher() {  
        System.out.println("Je suis un rectangle");  
    }  
}
```

```
public class Cercle implements Forme {  
  
    @Override  
    public void afficher() {  
        System.out.println("Je suis un cercle");  
    }  
}
```

```
public class TestForme {  
  
    static void afficherForme(Forme f) {  
        f.afficher();  
    }  
  
    public static void main(String[] args) {  
        Cercle cer = new Cercle();  
        Rectangle rec = new Rectangle();  
  
        cer.afficher();  
        rec.afficher();  
  
        TestForme.afficherForme(cer);  
        TestForme.afficherForme(rec);  
    }  
}
```

```
run:  
  
Je suis un cercle  
Je suis un rectangle  
Je suis un cercle  
Je suis un rectangle
```

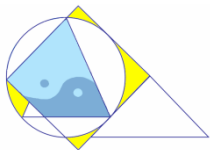
## 2.4- La Classe : Python



```
class Forme :  
    def afficher(self):  
        pass  
  
class Rectangle(Forme):  
    def afficher(self):  
        print("je suis un rectangle")  
  
class Cercle(Forme):  
    def afficher(self):  
        print("je suis un cercle")  
  
def afficherForme(f):  
    f.afficher()  
  
cer = Cercle()  
rec = Rectangle()  
cer.afficher()  
rec.afficher()  
print("\n")  
afficherForme(rec)  
afficherForme(cer)
```

```
je suis un cercle  
je suis un rectangle
```

```
je suis un rectangle  
je suis un cercle
```



## 2.4- La Classe : C++



```
#include <iostream>

using namespace std;

class Forme
{
public:
    virtual void afficher() const = 0;
};

class Rectangle:public Forme
{
public:
    void afficher() const { cout << "je suis un rectangle" << endl; }
};

class Cercle:public Forme
{
public:
    void afficher() const { cout << "je suis un cercle" << endl; }
};

void afficherForme(const Forme& f) { f.afficher(); }

int main()
{
    Cercle cer;
    Rectangle rec;
    cer.afficher();
    rec.afficher();

    afficherForme(cer);
    afficherForme(rec);
}
```

```
D:\YantraTechnologies\Yantra-Client\Client-CEA\CEA_250315\POO6\bin\Debug
je suis un cercle
je suis un rectangle
je suis un cercle
je suis un rectangle

Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

## 2.4- La Classe : C#



3 références

`interface IForme`

```
{
    5 références
    void afficher();
}
```

2 références

`class Cercle : IForme`

```
{
    5 références
    public void afficher() => Console.WriteLine("je suis un Cercle");
}
```

2 références

`class Rectangle : IForme`

```
{
    5 références
    public void afficher() => Console.WriteLine("je suis un Rectangle");
}
```

2 références

`class Programme`

```
{
    2 références
    public static void afficherForme(IForme f) { f.afficher(); }
    0 références
    public static void Main(string[] args)
    {
        Console.WriteLine("Exemple 3 : C#");
        Cercle cercle = new Cercle();
        Rectangle rectangle = new Rectangle();
        cercle.afficher();
        rectangle.afficher();

        Programme.afficherForme(cercle);
        Programme.afficherForme(rectangle);
        GC.Collect();
        GC.WaitForPendingFinalizers();
        Console.Read();
    }
}
```

```
Exemple 3 : C#
je suis un Cercle
je suis un Rectangle
je suis un Cercle
je suis un Rectangle
```

## 2.4- La Classe : procédural C



```
typedef enum { FORME, CERCLE, RECTANGLE } TypeForme;
```

```
typedef struct {  
    TypeForme _type;  
    void* _fils;  
} Forme_struct;
```

```
typedef struct {  
    Forme_struct* _parent;  
} Rectangle_struct;
```

```
typedef struct {  
    Forme_struct* _parent;  
} Cercle_struct;
```



## 2.4- La Classe : procédural C



```
Forme_struct * creerForme() {
    Forme_struct* res=(Forme_struct*)malloc(sizeof(Forme_struct));
    res->_type=FORME;
    res->_fils=NULL;
    return res;
}

Cercle_struct * creerCercle() {
    Cercle_struct* res=(Cercle_struct*)malloc(sizeof(Cercle_struct));
    res->_parent=creerForme();
    res->_parent->_type=CERCLE;
    res->_parent->_fils=(void*)res;
    return res;
}

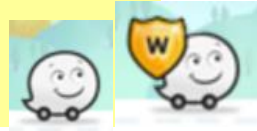
Rectangle_struct *creerRectangle() {
    Rectangle_struct* res=(Rectangle_struct*)malloc(sizeof(Rectangle_struct));
    res->_parent=creerForme();
    res->_parent->_type=RECTANGLE;
    res->_parent->_fils=(void*)res;
    return res;
}
```

## 2.4- La Classe : procédural C



```
void afficherRectangle(Rectangle_struct *) { printf("je suis un rectangle\n"); }
void afficherCercle(Cercle_struct *) { printf("je suis un cercle\n"); }
void afficherForme(const Forme_struct* f)
{
    switch(f->_type){
        case CERCLE:afficherCercle((Cercle_struct*)f);break;
        case RECTANGLE:afficherRectangle((Rectangle_struct*)f);break;
        default:break;
    };
}

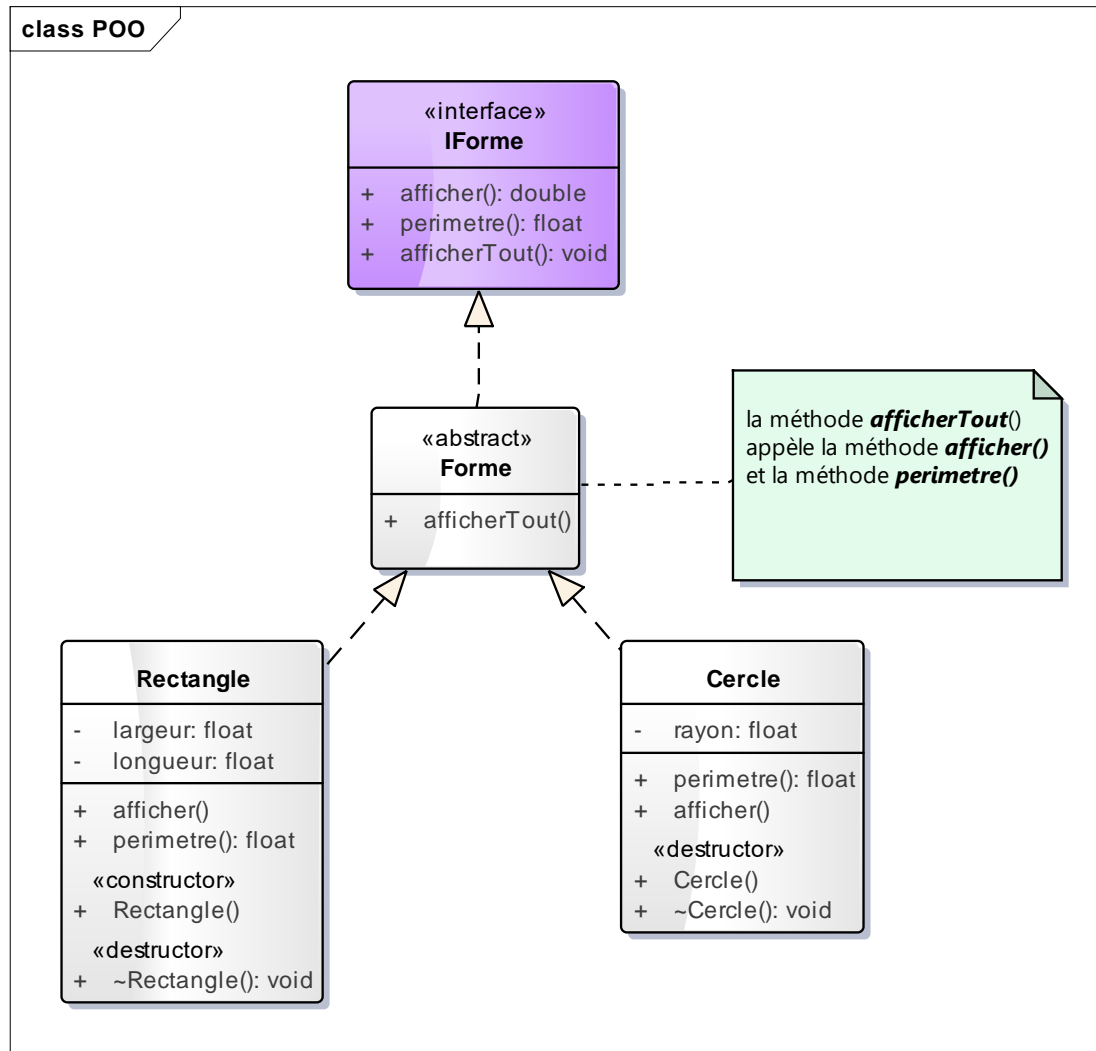
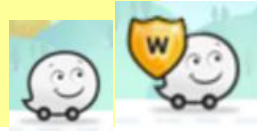
int main()
{
    Cercle_struct *cer = creerCercle();
    Rectangle_struct *rec = creerRectangle();
    afficherCercle(cer);
    afficherRectangle(rec);
    afficherForme(cer->_parent);
    afficherForme(rec->_parent);
    free(cer->_parent);
    free(cer);
    free(rec->_parent);
    free(rec);
}
```

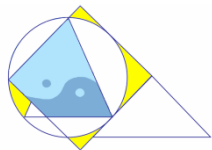


# Les Classes Abstraites

- Elles représentent des concepts : *Mammifère*
- Elles peuvent contenir des méthodes abstraites :
  - méthodes sans implémentation (corps),
  - la mise en œuvre pour chaque sous-classe peut être différente (*calculerSurface*)
  - polymorphisme

## 2.8- Les Classes Abstraites





## 2.4- La Classe : PHP



```
abstract class IForme{

    abstract public function afficher();
    abstract public function perimetre();
    abstract public function afficherTout();

}

abstract class Forme extends IForme{
    public function afficherTout(){
        $this->afficher();
        print("le perimetre de la forme est : ".$this->perimetre(). "</br></br>");
    }
}

class Rectangle extends Forme{
    public $largeur = 0.0;
    public $longueur = 0.0;
    public function __construct($la, $lo){
        $this->largeur = $la;
        $this->longueur = $lo;
    }
    public function afficher(){
        print ("Je suis un rectangle</br>");
    }
    public function perimetre(){
        return 2 * ($this->longueur + $this->largeur) ;
    }
}

class Cercle extends Forme{
    public $rayon = 0.0;
    public function __construct($r){
        $this->rayon = $r;
    }
    public function afficher(){
        print ("Je suis un cercle</br>");
    }
    public function perimetre(){
        return M_PI * ($this->rayon * 2);
    }
}
```

## 2.4- La Classe : PHP



```
$cercle = new Cercle(10);  
$rectangle = new Rectangle(5,2);  
$cercle->afficherTout();  
$rectangle->afficherTout();
```

Je suis un cercle  
le perimetre de la forme est :62.831853071796

Je suis un rectangle  
le perimetre de la forme est :14

## 2.4- La Classe : Java



```
public interface IForme {  
  
    public abstract void afficher();  
  
    public abstract double perimetre();  
  
    public abstract void afficherTout();  
  
}
```

```
public abstract class Forme implements IForme {  
  
    @Override  
    public void afficherTout() {  
        this.afficher();  
        System.out.println("Le perimetre de l  forme est " + this.perimetre());  
    }  
  
}
```

## 2.4- La Classe : Java



```
public class Cercle extends Forme {

    private final double a_rayon;

    public Cercle( double rayon) {
        this.a_rayon = rayon;
    }
    @Override
    public double perimetre() {
        return 2 * this.a_rayon * Math.PI;
    }
    @Override
    public void afficher() {
        System.out.println("Je suis un cercle");
    }
}
```

```
public class TestForme {

    static void afficherForme(Forme f) {
        f.afficher();
    }

    public static void main(String[] args) {
        Cercle cer = new Cercle(10);
        Rectangle rec = new Rectangle(5,2);

        cer.afficherTout();
        rec.afficherTout();
    }
}
```

```
public class Rectangle extends Forme {

    private final double a_longueur;
    private final double a_largeur;

    public Rectangle(double longueur, double largeur) {
        this.a_longueur = longueur;
        this.a_largeur = largeur;
    }

    @Override
    public double perimetre() {
        return (this.a_largeur + this.a_longueur) * 2.;
    }

    @Override
    public void afficher() {
        System.out.println("Je suis un rectangle");
    }
}
```

run:

```
Je suis un cercle
Le perimetre de 1 forme est 62.83185307179586
Je suis un rectangle
Le perimetre de 1 forme est 14.0
BUILD SUCCESSFUL (total time: 0 seconds)
```



## 2.4- La Classe : Python



```
class IForme:
    def afficher(self) :
        pass
    def perimetre(self):
        pass
    def afficherTout(self):
        pass

class Forme (IForme):
    def afficherTout(self):
        self.afficher();
        print("le perimetre de la forme est : ",self.perimetre())
```

```
class Rectangle(Forme):
    def __init__(self, longueur, largeur):
        self.longueur = longueur
        self.largeur=largeur
    def afficher(self):
        print("je suis un rectangle")
    def perimetre(self):
        return (self.longueur+self.largeur) * 2
```

```
class Cercle(Forme):
    def __init__(self, rayon):
        self.rayon = rayon
    def afficher(self):
        print("je suis un cercle")
    def perimetre(self):
        return 2 * 3.1415 * self.rayon
```

```
cer = Cercle(10)
rec =Rectangle(5,2)
cer.afficherTout()
rec.afficherTout()
```

```
je suis un cercle
le perimetre de la forme est : 62.830000000000005
je suis un rectangle
le perimetre de la forme est : 14
```

## 2.4- La Classe : C++



```
class Forme
{
public:
    virtual void afficher() const = 0;
    virtual double perimetre() const = 0;
    void afficherTout()
    {
        this->afficher();
        cout << "le perimetre de la forme est : " << this->perimetre() << endl;
    }
};

class Rectangle:public Forme
{
private:
    double _longueur;
    double _largeur;
public:
    Rectangle(double longueur, double largeur):
        _longueur(longueur), _largeur(largeur) {}
    void afficher() const { cout << "je suis un rectangle" << endl; }
    double perimetre() const { return (this->_longueur + this->_largeur) * 2 ; }
};

class Cercle:public Forme
{
private:
    double _rayon;
public:
    Cercle(double rayon):
        _rayon(rayon) {}
    void afficher() const { cout << "je suis un cercle" << endl; }
    double perimetre() const { return 2 * 3.1415 * this->_rayon; }
};
```

## 2.4- La Classe : C++



```
je suis un cercle  
le perimetre de la forme est : 62.83  
je suis un rectangle  
le perimetre de la forme est : 14
```

```
int main()  
{  
    Cercle cer(10);  
    Rectangle rec(5,2);  
    cer.afficherTout();  
    rec.afficherTout();  
  
    return 0;  
}
```

## 2.4- La Classe : C#



```
interface IForme
```

```
{  
    4 références  
    void afficher();  
    4 références  
    double perimetre();  
    3 références  
    void afficherTout();  
}
```

```
2 références
```

```
abstract class AForme : IForme
```

```
{  
    4 références  
    public abstract void afficher();  
  
    3 références  
    public void afficherTout()  
    {  
        this.afficher();  
        Console.WriteLine("Le perimetre de la forme est : " + this.perimetre());  
    }  
    4 références  
    public abstract double perimetre();  
}
```

## 2.4- La Classe : C++



```
class Cercle : AForme
{
    private double a_rayon;
    private static readonly double pi = Math.PI;

    1 référence
    public Cercle(double rayon)
    {
        this.a_rayon = rayon;
    }

    4 références
    public override void afficher() => Console.WriteLine("je suis un Cercle");

    4 références
    public override double perimetre() => 2 * Cercle.pi * this.a_rayon ;
}

2 références
class Rectangle : AForme
{
    private double a_longueur;
    private double a_largeur;

    1 référence
    public Rectangle(double longueur, double largeur)
    {
        this.a_longueur = longueur;
        this.a_largeur = largeur;
    }

    4 références
    public override void afficher() => Console.WriteLine("je suis un Rectangle");

    4 références
    public override double perimetre() => 2 * (this.a_largeur + this.a_longueur);
}
```

## 2.4- La Classe : C++



0 références

```
class Programme
```

```
{
```

0 références

```
public static void Main(string[] args)
```

```
{
```

```
    Console.WriteLine("Exemple 3 : C#");
```

```
    IForme cercle = new Cercle(10);
```

```
    IForme rectangle = new Rectangle(5, 2);
```

```
    cercle.afficherTout();
```

```
    rectangle.afficherTout();
```

```
    GC.Collect();
```

```
    GC.WaitForPendingFinalizers();
```

```
    Console.Read();
```

```
}
```

```
}
```

```
Exemple 3 : C#
je suis un Cercle
Le perimetre de la forme est : 62,8318530717959
je suis un Rectangle
Le perimetre de la forme est : 14
```

## 2.4- La Classe : procédural C



```
typedef enum { FORME, CERCLE, RECTANGLE } TypeForme;
```

```
typedef struct {  
    TypeForme _type;  
    void* _fils;  
    double (*perimetre) (void*);  
    void (*afficher) (void*);  
} Forme_struct;
```

```
typedef struct {  
    Forme_struct* _parent;  
    double _largeur;  
    double _longueur;  
} Rectangle_struct;
```

```
typedef struct {  
    Forme_struct* _parent;  
    double _rayon;  
} Cercle_struct;
```

## 2.4- La Classe : procédural C



```
double Rectangle_perimetre(void * ele) {  
    Rectangle_struct *rec = (Rectangle_struct *) ele;  
    return (rec->_largeur + rec->_longueur)*2;  
}  
  
double Cercle_perimetre(void *ele) {  
    Cercle_struct *rec = (Cercle_struct *) ele;  
    return 2*3.1415*rec->_rayon;}  
  
double Forme_perimetre(const Forme_struct* f) {  
    return f->perimetre(f->_fils);}
```



## 2.4- La Classe : procédural C



```
Forme_struct * creerForme() {
    Forme_struct* res=(Forme_struct*)malloc(sizeof(Forme_struct));
    res->_type=FORME;
    res->_fils=NULL;
    return res;
}

Cercle_struct * creerCercle(double rayon) {
    Cercle_struct* res=(Cercle_struct*)malloc(sizeof(Cercle_struct));
    res->_parent=creerForme();
    res->_rayon = rayon;
    res->_parent->_type=CERCLE;
    res->_parent->_fils=(void*)res;
    res->_parent->perimetre = Cercle_perimetre;
    res->_parent->afficher = Cercle_afficher;
    return res;
}

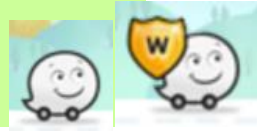
Rectangle_struct *creerRectangle(double longueur,double largeur) {
    Rectangle_struct* res=(Rectangle_struct*)malloc(sizeof(Rectangle_struct));
    res->_parent=creerForme();
    res->_longueur=longueur;
    res->_largeur=largeur;
    res->_parent->_type=RECTANGLE;
    res->_parent->_fils=(void*)res;
    res->_parent->perimetre = Rectangle_perimetre;
    res->_parent->afficher = Rectangle_afficher;
    return res;
}
```

## 2.4- La Classe : procédural C



```
void libererForme( Forme_struct* f) {  
    switch(f->_type) {  
        case CERCLE: free((Cercle_struct*)f->_fils); break;  
        case RECTANGLE: free((Rectangle_struct*)f->_fils); break;  
        default: break;  
    }  
    free(f);  
}  
  
int main()  
{  
    Forme_struct *cer = creerCercle(10)->_parent;  
    Forme_struct *rec = creerRectangle(5,2)->_parent;  
    Forme_afficher(cer);  
    printf("perimetre cercle %f\n", Forme_perimetre(cer));  
    Forme_afficher(rec);  
    printf("perimetre rectangle %f\n", Forme_perimetre(rec));  
    libererForme(cer);  
    libererForme(rec);  
}
```

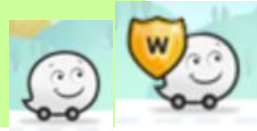
```
je suis un cercle  
perimetre cercle 62.830000  
je suis un rectangle  
perimetre rectangle 14.000000
```



Il s'agit donc de l'ensemble des méthodes accessibles depuis l'extérieur de la classe, par lesquelles on peut modifier l'objet. Pour rappel la différenciation publique/privée ou portée de variable permet :

- d'éviter de manipuler l'objet de façon non voulue, en limitant ses modifications à celles autorisées comme publiques par le concepteur de la classe
- Au concepteur, de modifier l'implémentation interne de ces méthodes de manière transparente.

[https://fr.wikipedia.org/wiki/Interface\\_\(programmation\\_orient%C3%A9e\\_objet\)](https://fr.wikipedia.org/wiki/Interface_(programmation_orient%C3%A9e_objet))



**class POO**

«interface»  
**IForme**

- + afficher(): double
- + perimetre(): float
- + afficherTout(): void

## 2.4- La Classe : PHP



```
abstract class IForme{  
    abstract public function afficher();  
    abstract public function perimetre();  
    abstract public function afficherTout();  
}
```

## 2.4- La Classe : Java



```
public interface IForme {  
  
    public abstract void afficher();  
  
    public abstract double perimetre();  
  
    public abstract void afficherTout();  
  
}
```

## 2.4- La Classe : Python



```
class IForme:
    def afficher(self) :
        pass
    def perimetre(self):
        pass
    def afficherTout(self):
        pass
```

## 2.4- La Classe : C++



```
class IForme {  
    public:  
        virtual void afficher() const = 0;  
        virtual double perimetre() const = 0;  
        virtual void afficherTout() const = 0;  
};
```



## 2.4- La Classe : C#



3 références

```
interface IForme
```

```
{
```

4 références

```
void afficher();
```

4 références

```
double perimetre();
```

3 références

```
void afficherTout();
```

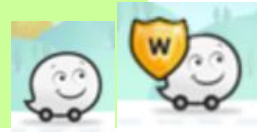
```
}
```

## 2.4- La Classe : procédural C



```
]typedef struct {  
    TypeForme _type;  
    void* _fils;  
    double (*perimetre) (void*);  
    void (*afficher) (void*);  
-} Forme_struct;
```

## 2.10- Clonage, comparaison et assignation



### Clonage d'un Objet

- Créer un objet *identique* à un autre



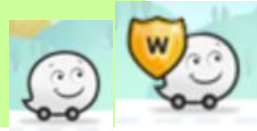
### Comparaison Objet :

- Comparer 2 objet pour savoir si il sont *identiques*



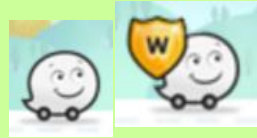
### Assignation Objet

- Copier l'état d'un objet dans un autre afin qu'il soit *identique*



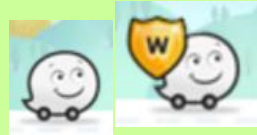
### 3 concepts pour faire un langage objet :

- **Encapsulation** : combiner des données et un comportement dans un emballage unique,
- **Héritage** : chiens et chats sont des mammifères, ils héritent du comportement du mammifère.
- **Polymorphisme** : Cercle et rectangle sont des formes géométriques, chacun doit calculer sa surface.



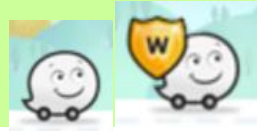
### Cycle de vie itératif

- **Spécification**
  - **étude du contexte** : objets du domaine, objets applicatifs
- **Analyse**
  - **analyse du domaine** : classes du domaine
  - **analyse applicative** : classes applicatives
- **Conception**
  - **conception préliminaire** : classes d'interface
  - **conception objet** : classes techniques
- **Implémentation**
  - **codage** : classes techniques spécifiques au langage utilisé

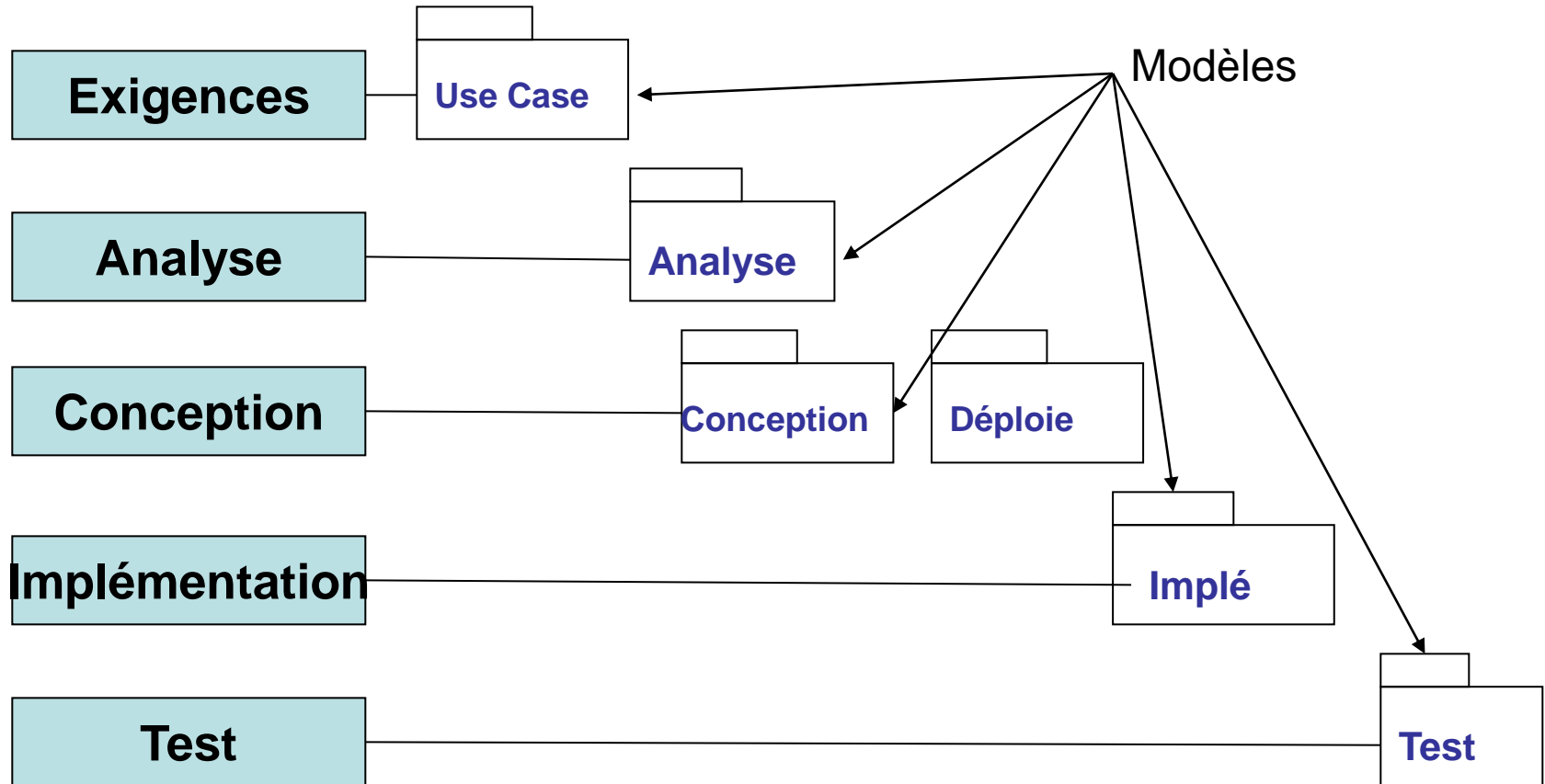


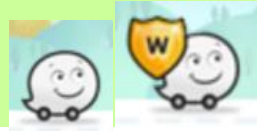
### Approche itérative et incrémentale

- **Développement par approches successives**
  - **1er cycle** : prototype de base
  - **2ème cycle** : amélioration du prototype de base, intégration de nouvelles fonctionnalités
  - **3ème ...** : continuer jusqu'à épuisement des fonctionnalités
- **Étapes**
  - **définition du domaine** : design général
  - **définition des sous-systèmes** : design du sous-système
  - **remonter les problèmes dans le design général**
  - **réitérer pour chaque sous-système.**

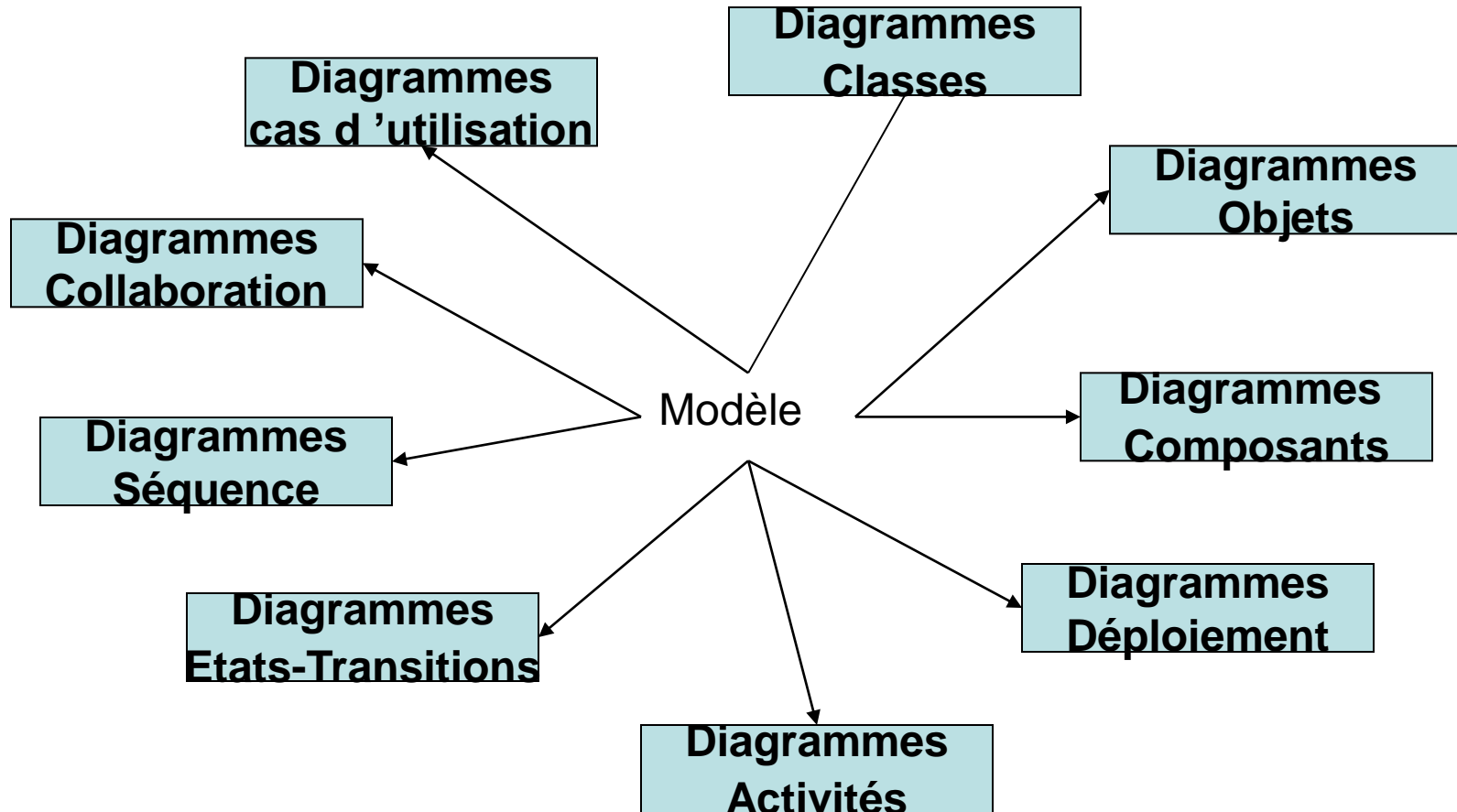


# Modélisation UML

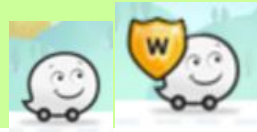




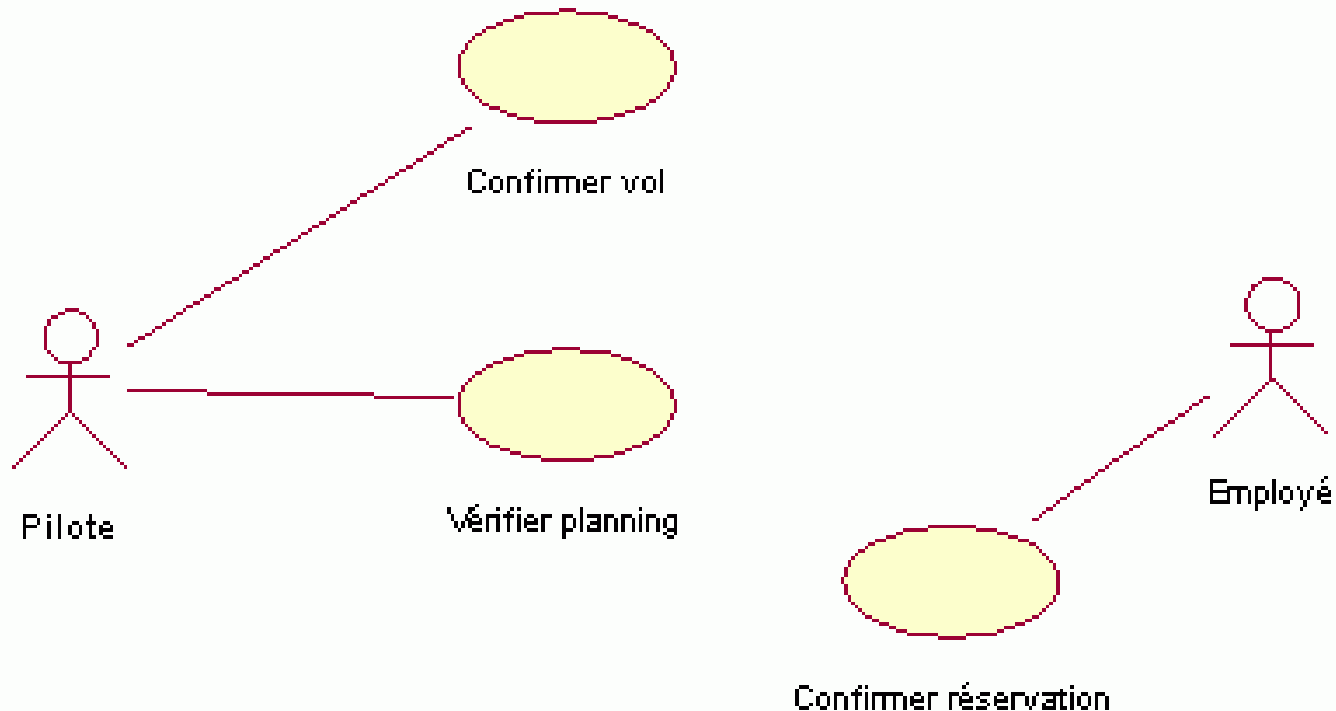
# Modélisation UML

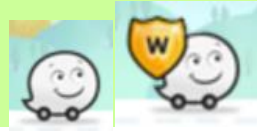






# Diagrammes de cas d'utilisation : Use Case





### Avantages du style objet

- **Facilite la programmation modulaire :**
  - composants réutilisables,
  - un composant offre des services et en utilise d'autres,
  - il expose ses services au travers d'une interface
- **Facilite l'abstraction :**
  - elle sépare la définition de son implémentation,
  - elle extrait un modèle commun à plusieurs composants,
  - le modèle commun est partagé par le mécanisme d'héritage.
- **Facilite la spécialisation :**
  - elle traite des cas particuliers,
  - le mécanisme de dérivation rend les cas particuliers transparents.



- 3.1 Introduction aux design Patterns
- 3.2 Exemples simples intuitifs (non officiels)
- 3.3 Les patterns ?
- 3.4 Comment ?
- 3.5 Description des modèles de conception
- 3.6 Classification
- 3-7 Exemples :
  - Pattern Singleton
  - Pattern Façade

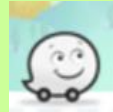


Un **Design Pattern** est un **motif ou patron** de Conception dont le rôle est d'implémenter un **mécanisme générique** répondant à une problématique **récurrente** sur un projet.

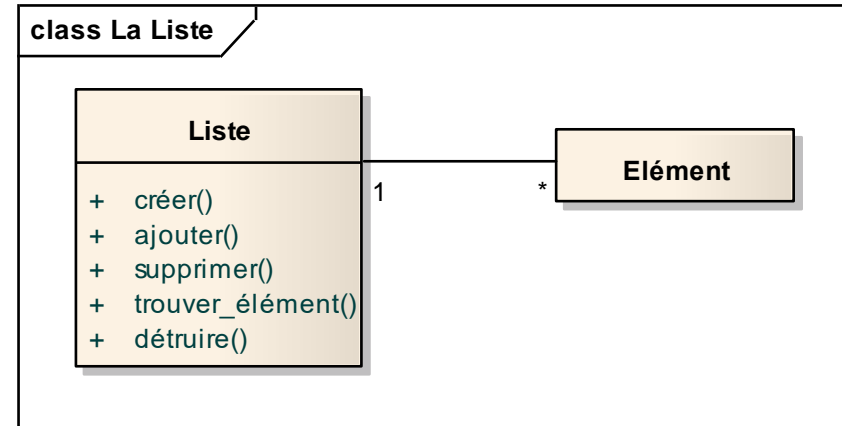
Les patrons de conception les plus répandus sont au nombre de 23.

Ils sont couramment appelés « patrons GoF » « Gang of Four » : Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides (1995).

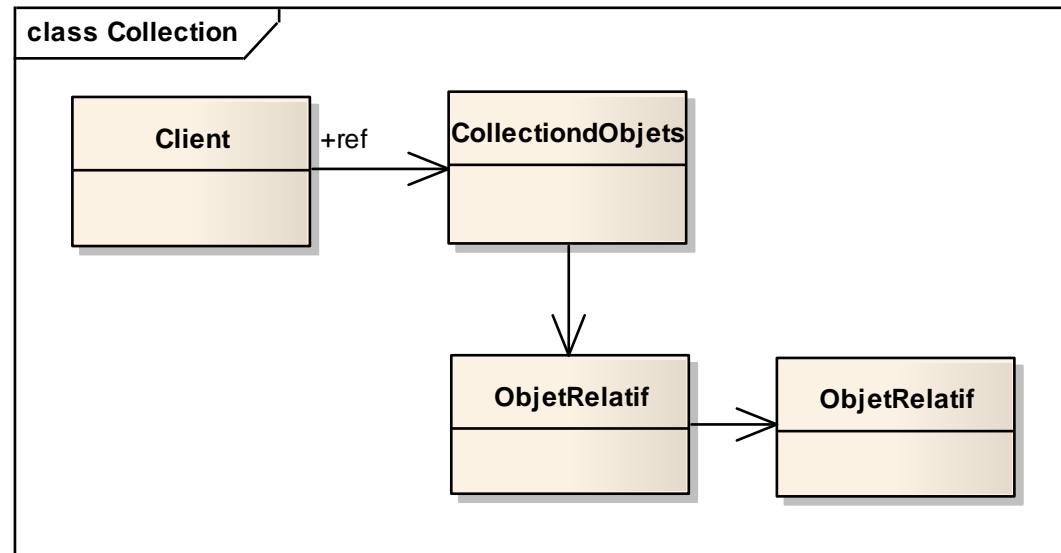
## 3.2 – Exemples simples intuitifs (non officiels)



La Liste : des défauts ?



Gestion de collections via une classe : UML ? des défauts ?





Design Pattern

=

Schéma de conception réutilisable

=

Organisation et hiérarchie de *plusieurs*  
modèles de classes réutilisables par simple  
implémentation, adaptation et extension



- Les Design Patterns sont présentés en utilisant la notation graphique standard UML.
- En général, seule la partie statique (diagrammes de classe) de UML est utilisée.
- La description d'un patron de conception suivant un formalisme fixe :
  - Nom
  - Description du problème à résoudre
  - Description de la solution : les éléments de la solution, avec leurs relations. La solution est appelée **patron de conception**.
  - Conséquences : résultats issus de la solution
  - ...

## 3.5 - Description des modèles de conception



- Nom et classification
- Implémentation
- Intention
- Exemple de code
- Autres noms connus
- Usages connus
- Motivation (scénario)
- Formes associées
- Applicabilité
- Structure
- Participants (classes...)
- Collaborations
- Conséquences



## 3.6 - Classification



		Modèles		
		Créateur	Structure	Comportement
	Classe	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

## 3.7.1 Exemple: Pattern Singleton



- Description

Assurer qu'une classe ne possède qu'une instance et fournir une méthode de classe unique retournant cette instance.

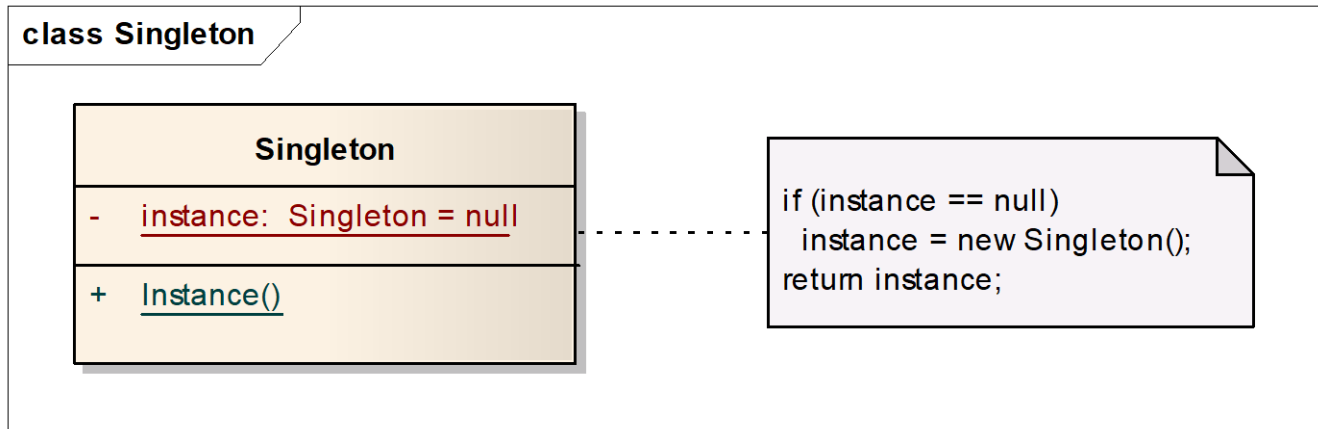
- Domaine d'utilisation

- ✓ Il ne doit y avoir qu'une seule instance de classe,
- ✓ Cette instance ne doit être accessible qu'au travers d'une méthode de classe.

## 3.7.1 Exemple: Pattern Singleton



- Structure





- **Description**

Regrouper les interfaces d'un ensemble d'objets en une interface unifiée pour rendre cet ensemble plus facile à utiliser pour un client.

Encapsule l'interface de chaque objet considérée comme interface de bas niveau dans une interface unique de niveau plus élevé.

Cela peut nécessiter d'implanter des méthodes destinées à composer des interfaces de bas niveau..

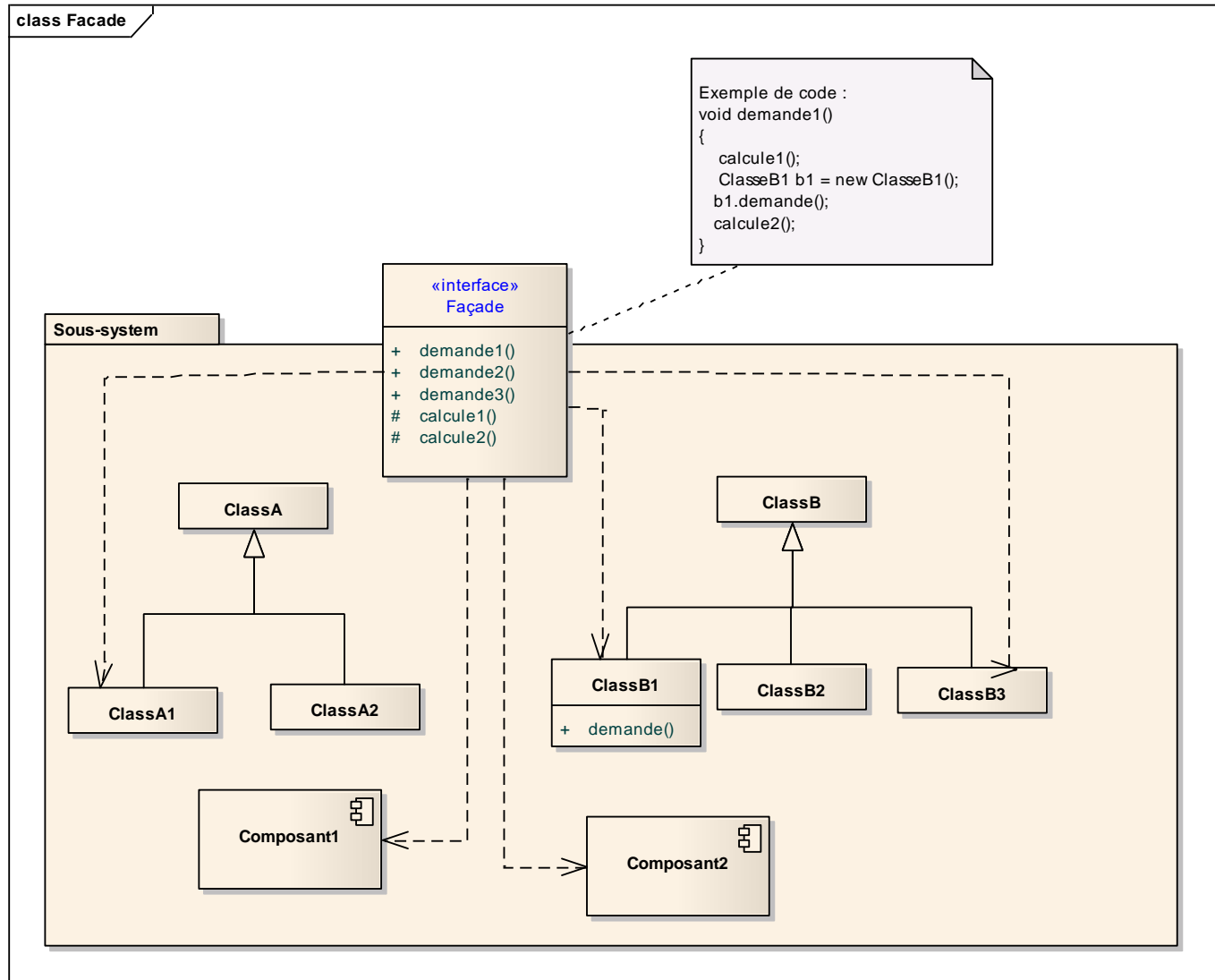
- **Domaine d'utilisation**

- ✓ Fournir une interface simple d'un système complexe.
- ✓ Diviser un système en sous-systèmes, la communication entre sous-systèmes étant mise en œuvre de façon abstraite de leur implantation grâce aux façades.
- ✓ Systématiser l'encapsulation de l'implantation d'un système vis-à-vis de l'extérieur.

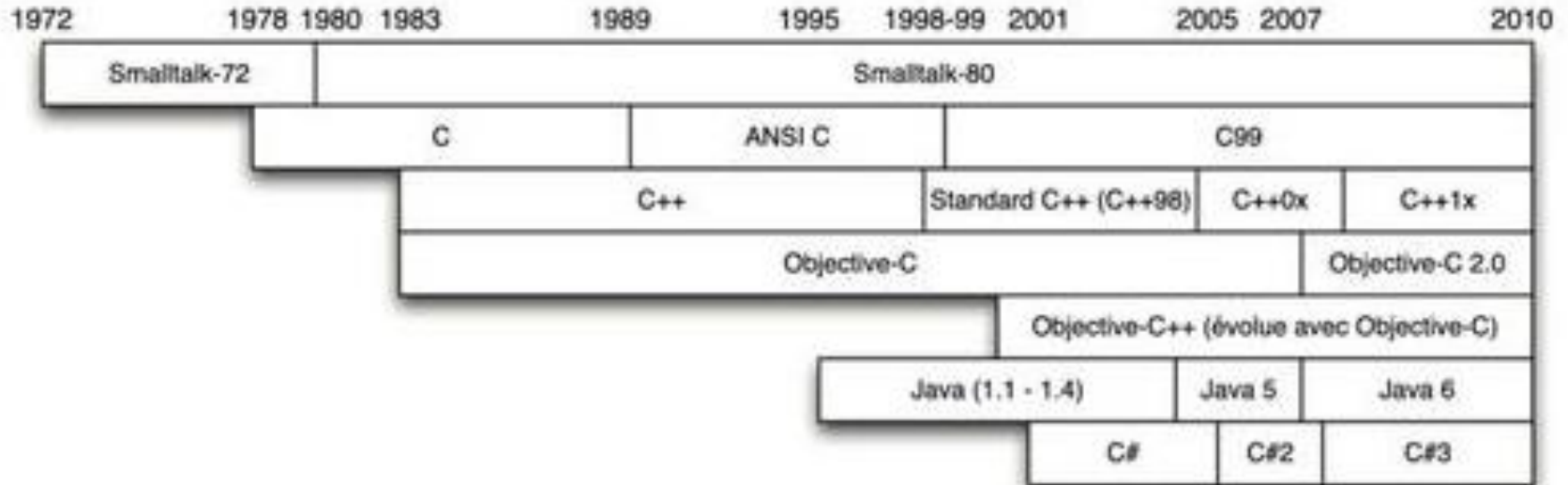
## 3.7.2 Exemple : Pattern Facade



### • Structure

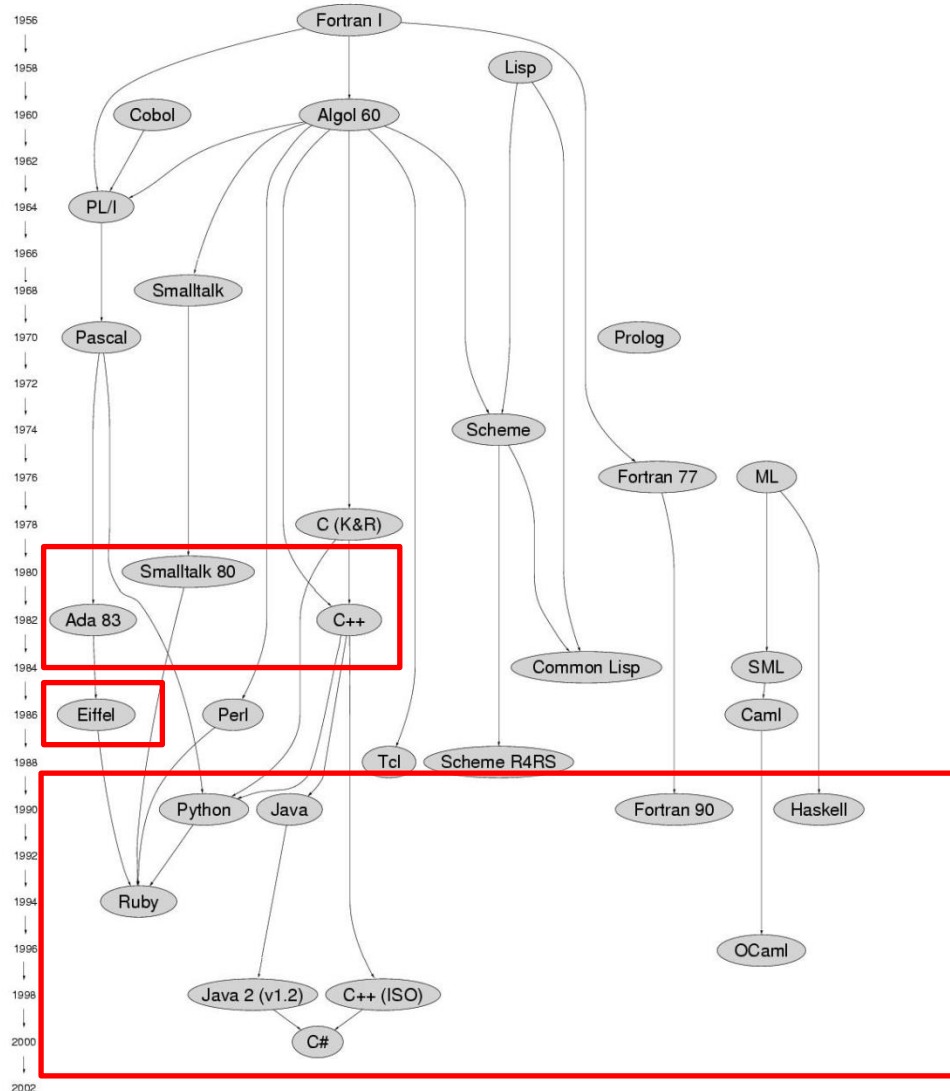


## 4 – Les Différents langages objets



[http://librairie.immateriel.fr/fr/read\\_book/9782212127515/chap001](http://librairie.immateriel.fr/fr/read_book/9782212127515/chap001)

## 4 – Les Différents langages objets



## 4 – Les Différents langages objets

















- Ada
- C++
- C#
- Delphi (=Pascal orienté objet)
- Java
- Kylix
- Objective-C
- Objective Caml (ocaml)
- Perl
- PHP (Depuis la version 4)
- Python
- SmallTalk (totalement objet)
- Ruby



## 4 – Les Différents langages objets



Oct 2021	Oct 2020	Change	Programming Language	Ratings	Change
1	3	↑	 Python	11.27%	-0.00%
2	1	↓	 C	11.16%	-5.79%
3	2	↓	 Java	10.46%	-2.11%
4	4		 C++	7.50%	+0.57%
5	5		 C#	5.26%	+1.10%
6	6		 Visual Basic	5.24%	+1.27%
7	7		 JavaScript	2.19%	+0.05%
8	10	↑	 SQL	2.17%	+0.61%
9	8	↓	 PHP	2.10%	+0.01%
10	17	↑↑	 Assembly language	2.06%	+0.99%
11	19	↑↑	 Classic Visual Basic	1.83%	+1.06%
12	14	↑	 Go	1.28%	+0.13%
13	15	↑	 MATLAB	1.20%	+0.08%
14	9	↓↓	 R	1.20%	-0.79%

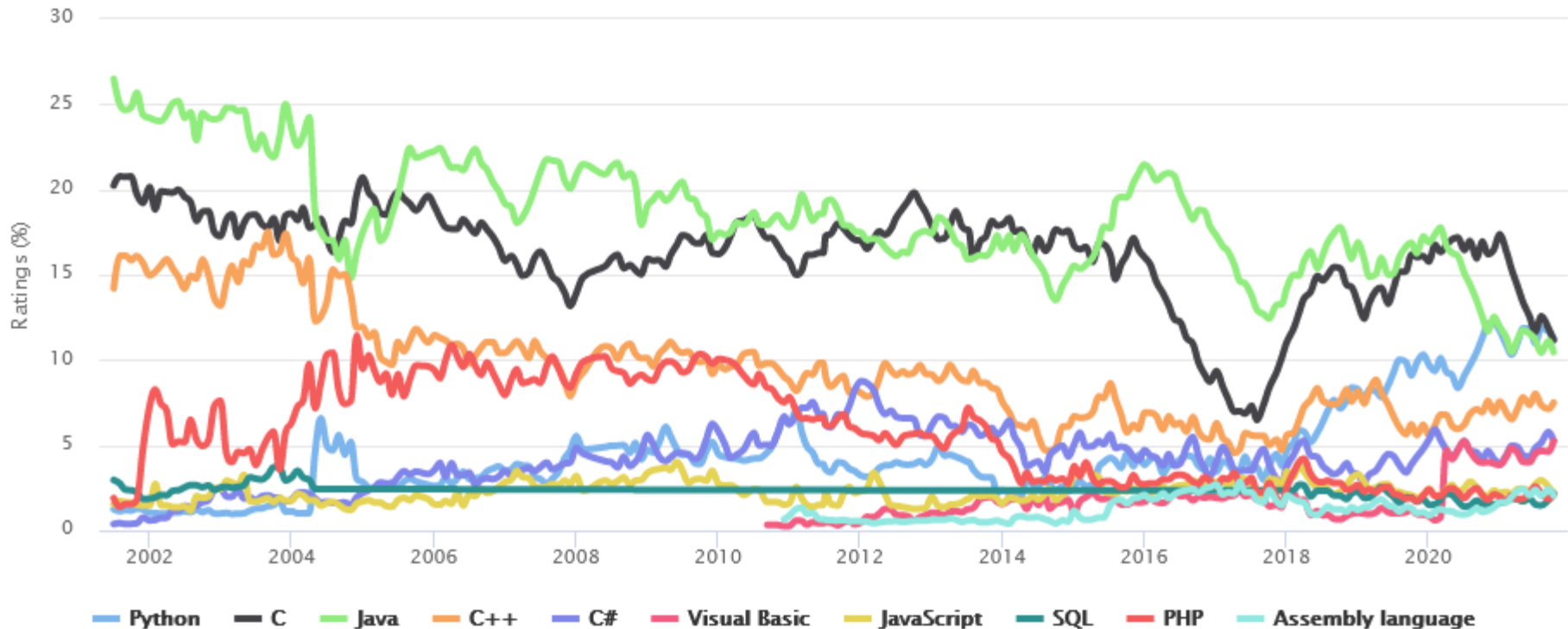
<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

## 4 – Les Différents langages objets



### TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)



<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

## 5 - Bibliographies

