

**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 - 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.

<https://www.commentcoder.com/types-programmation/>



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)

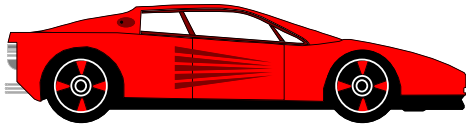


### Propriétés / Attributs

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

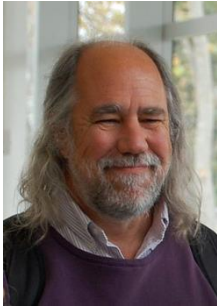




La pratique des méthodes a permis de faire le tri entre les différents concepts

***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

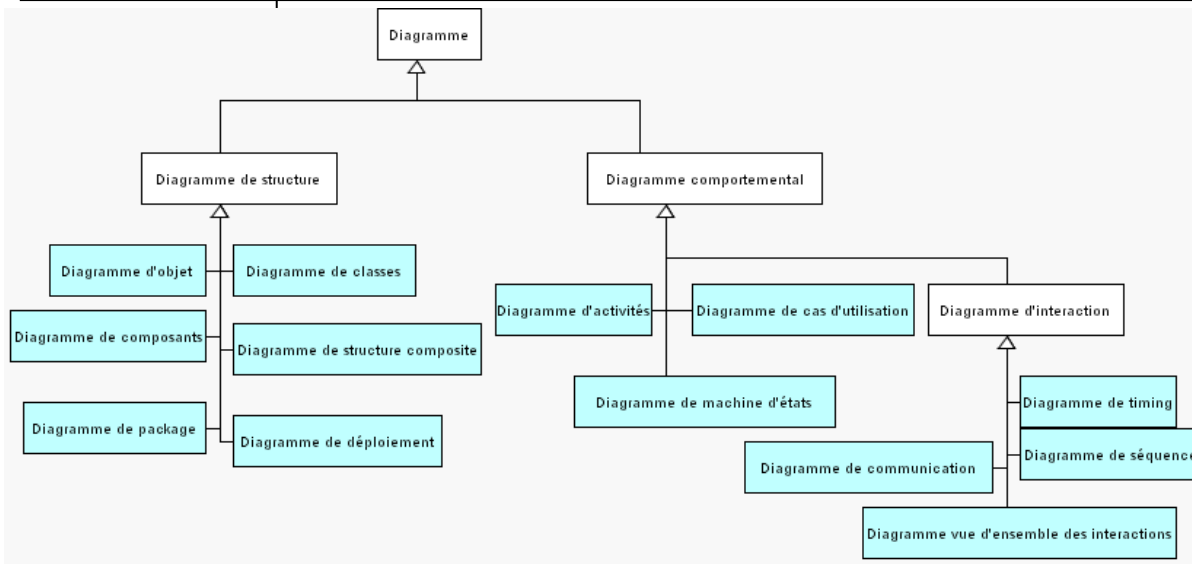


- **UML permet d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation.** Il a été pensé pour servir de support à une analyse basée sur les concepts objet.
- UML est un **langage formel**, défini par un **métamodèle**.
- Le métamodèle d'UML décrit de manière très précise 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



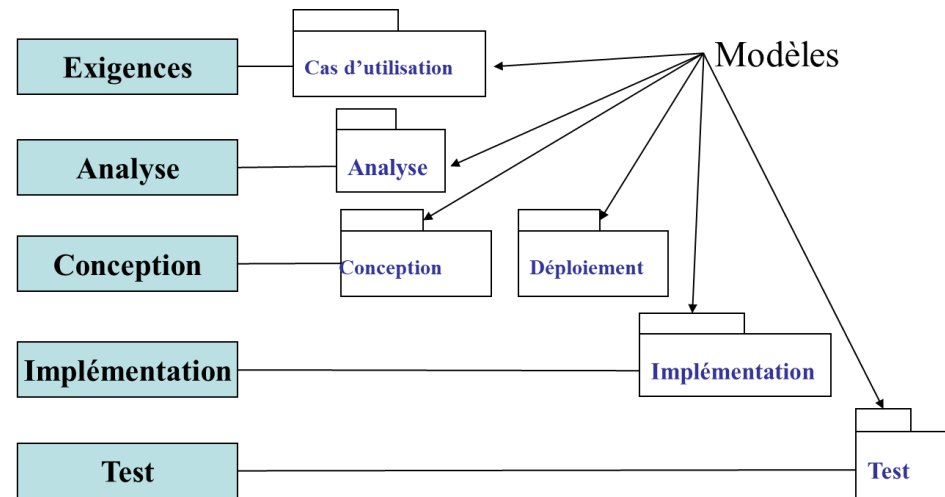
- Les points forts d'UML
  - UML est un langage formel et normalisé
  - UML est un support de communication performant
- Les points faibles d'UML
  - La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation.
  - Le processus (non couvert par UML) est une autre clé de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

## 1.8 - L'approche orientée objet

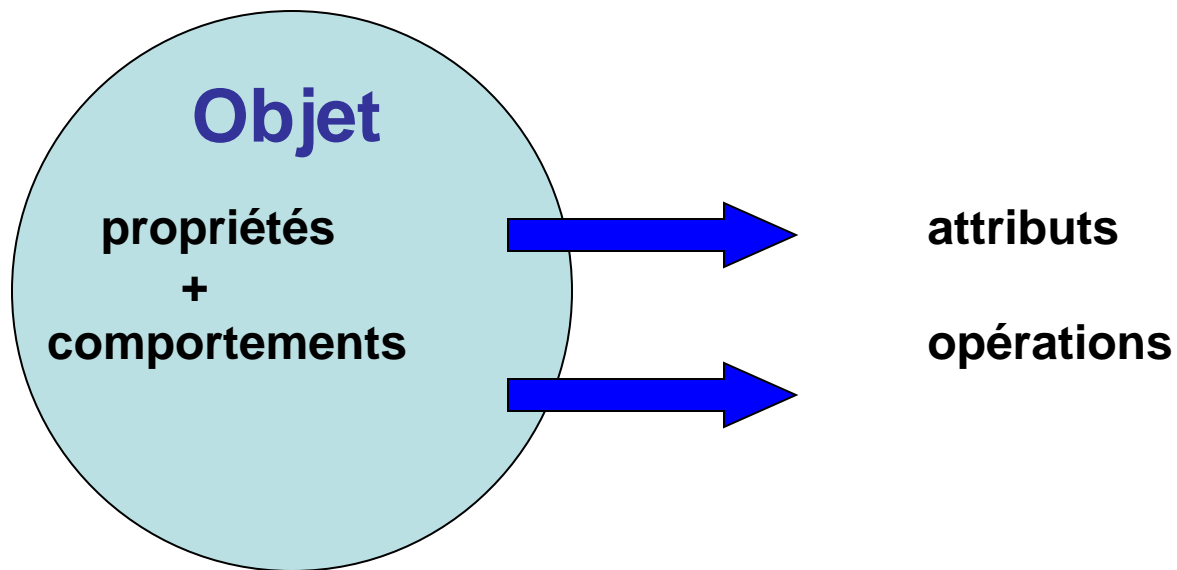


L'approche orientée objet considère le logiciel comme une collection d'objets dissociés définis par des **propriétés**.  
(propriété : **attribut** ou une **opération**)

- ⇒ 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.



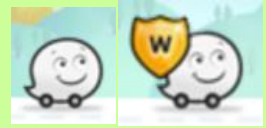
- 1- Présentation de la notion d 'Objet
- 2- La modélisation avec UML
- 3- Le principe de l'encapsulation
- 4- La Classe
- 5 -L'Héritage
- 6- Agrégation
- 7- Polymorphisme
- 8- Les classes abstraites
- 9- Relations d'association
- 10- Le Langage Objet
- 11- Développement d 'un projet orienté Objet

## 2.1- Présentation de la notion d'Objet

### Un Objet peut :

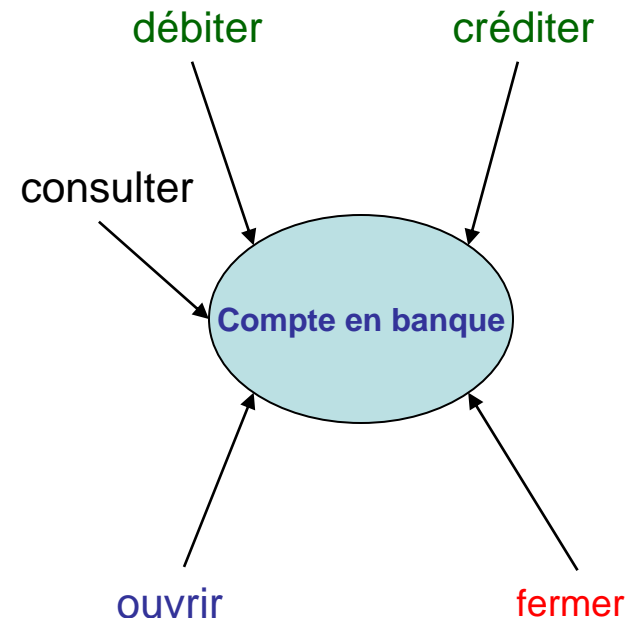
- changer d'état,
- se comporter de façon discernable,
- être manipulé par diverses formes de stimuli,
- être en relation avec d'autres objets.

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

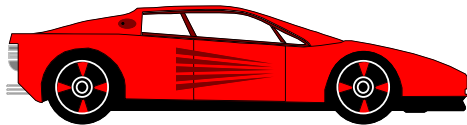


# Types d'opération sur le comportement

- **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

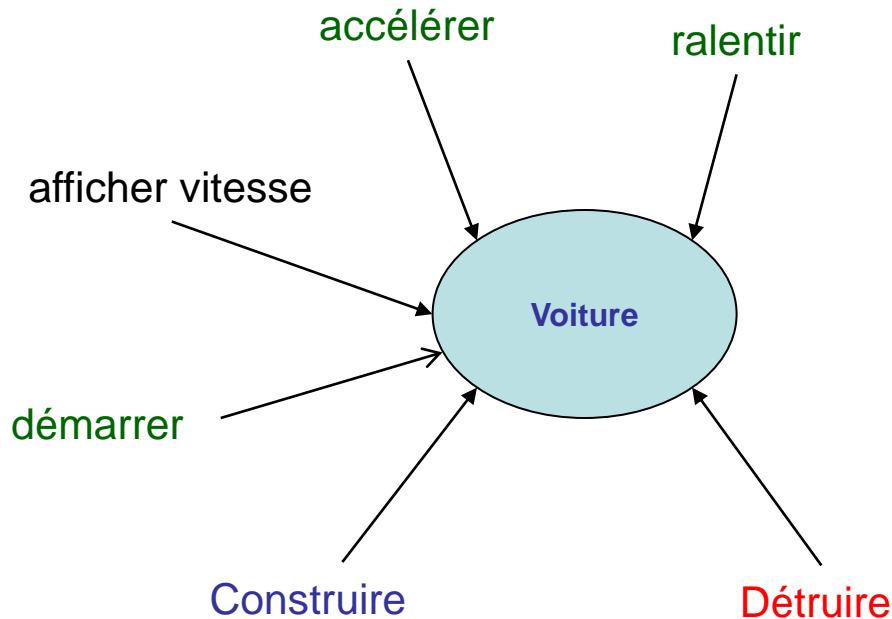
Ferrari  
rouge  
en\_stationnement

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

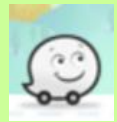
[https://img.yugioh-card.com/ygo\\_cms/ygo/all/uploads/Rulebook\\_v9\\_fr.pdf](https://img.yugioh-card.com/ygo_cms/ygo/all/uploads/Rulebook_v9_fr.pdf)



## 2.1 - Objet : 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();  
}
```



# 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*

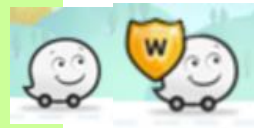
-> *Peugeot 406*, *Renault 18* seront des instances de cette classe donc des objets

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.

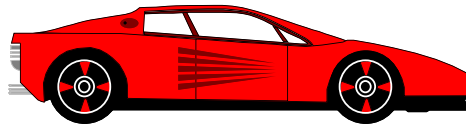
Exemple : deux T-shirts peuvent être strictement identiques et pourtant ils sont distincts.

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



# Une classe

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



## 2.4- La Classe





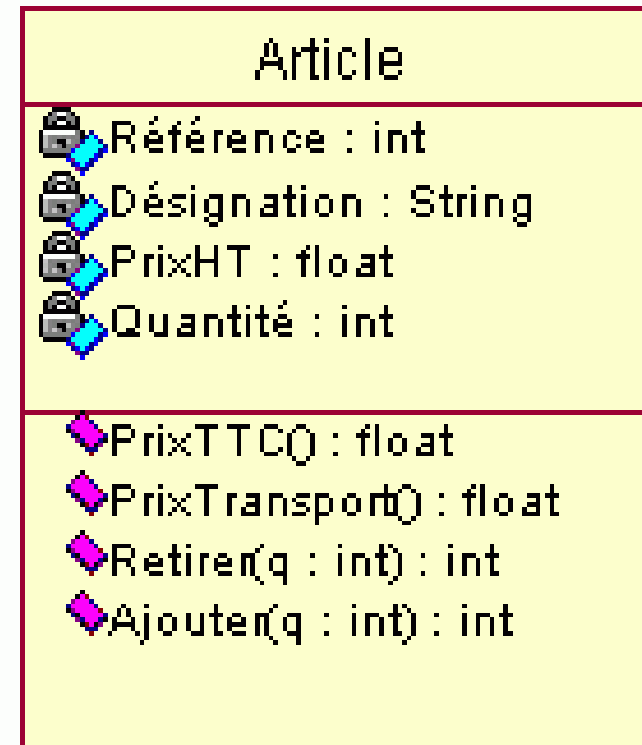
- Une **classe** est une construction du langage de programmation :
  - décrit les propriétés communes à des objets  
Class Point { }
  - un objet est une *instance* de classe
- Un **objet** est une entité en mémoire :
  - il a un état, un comportement, une identité.  
Point\* a = new Point(3,5); (C++)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

# Un objet sans classe n'existe pas



### La classe possède :

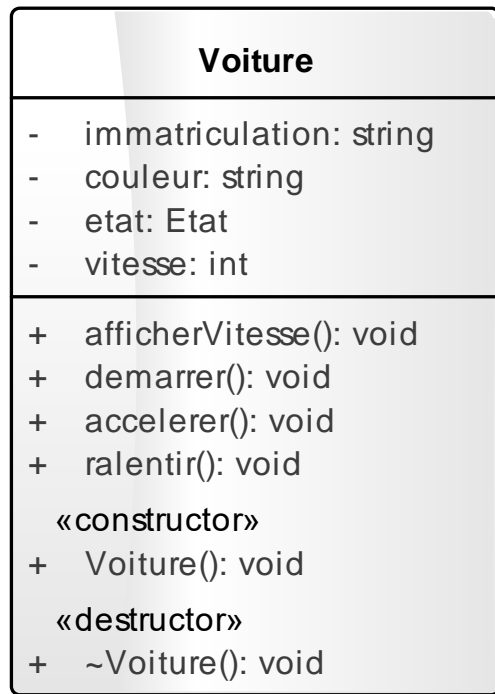
- un nom unique,
- une composante **statique** :  
les données qui sont des  
champs (attributs)  
⇒ Etat
- une composante  
**dynamique** : les méthodes  
(opérations)  
⇒ Comportement



## 2.4- La Classe



### class Test Model



Etat est soit "arreter" soit "demarrer"

## 2.4- 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 Cartes du jeu

### Cartes Monstre

#### COMMENT LIRE UNE CARTE

Cartes du jeu

**1 Nom de Carte** **INVOCATEUR DRAGON BLEU** **3 Attribut**

**2 Niveau**

**4 Type**

**5 Numéro de Carte**

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

**7 Description de la Carte**

**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.

**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.

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

Cartes du jeu

[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)



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

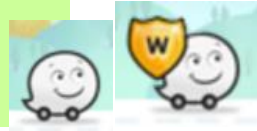
- Allouer de la mémoire
- Initialiser cette mémoire

# 2. Vie d'un objet

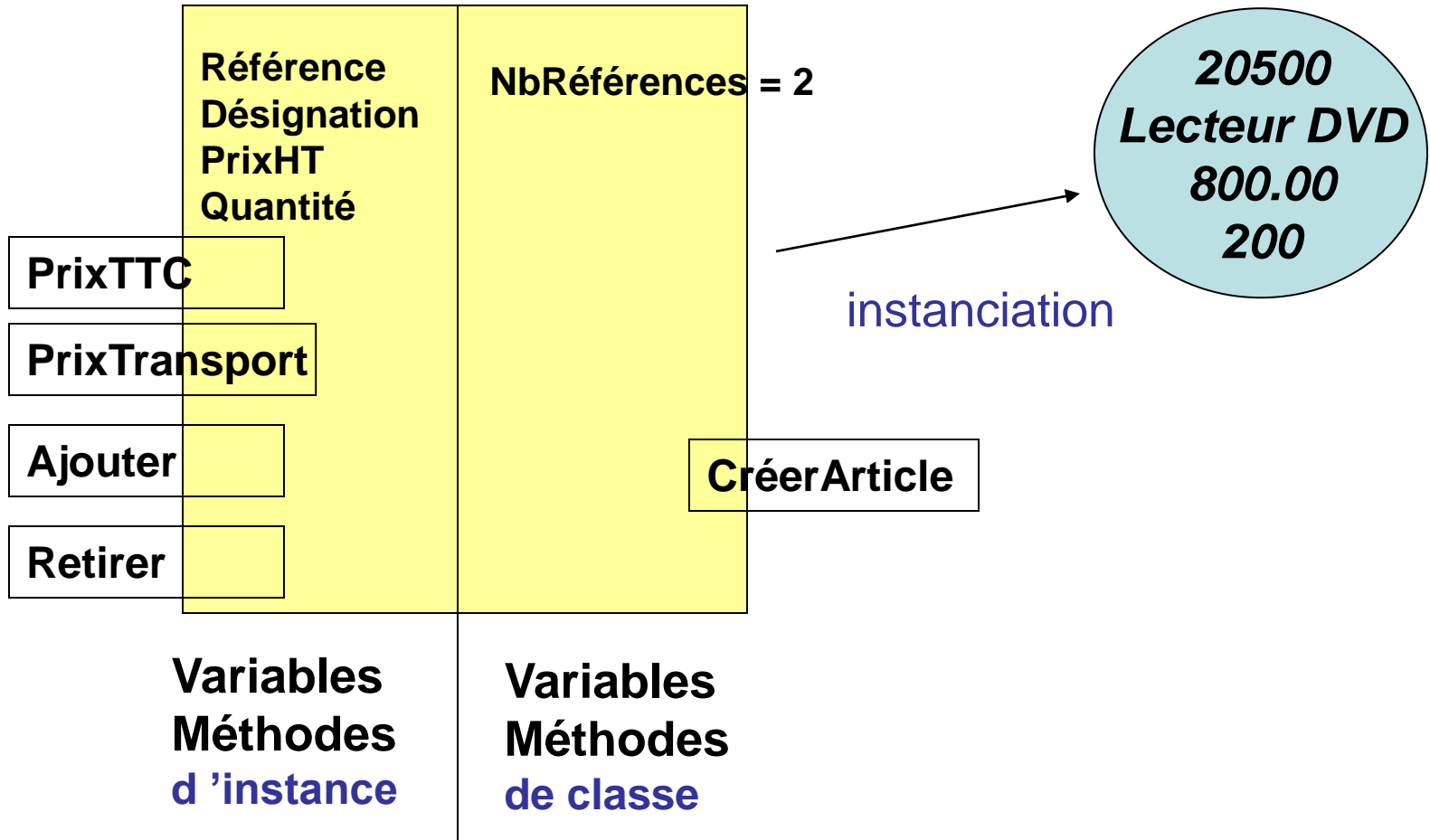
- Utilisation des méthodes en modification, sélection et itération

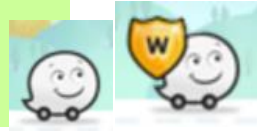
# 3. Mort d'un objet (*destructeur*)

- 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 : 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 : 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

## 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;

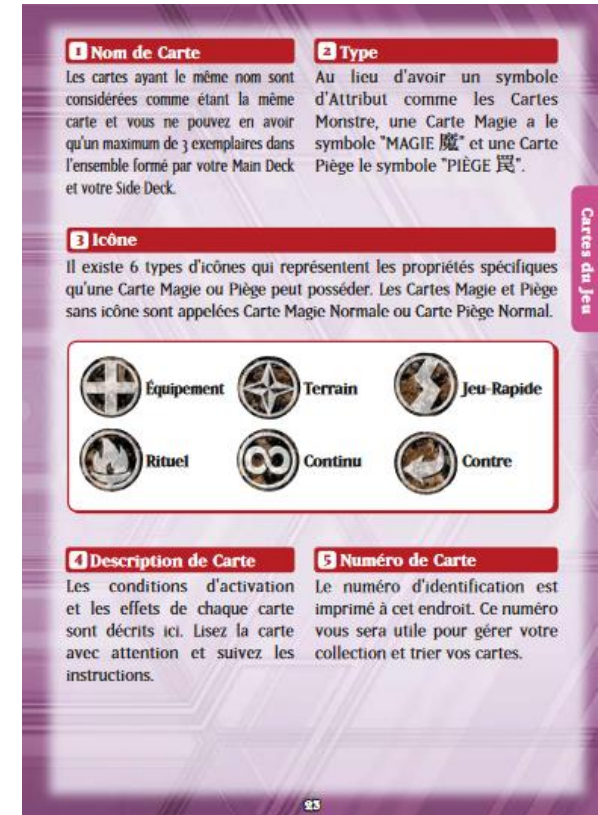
    public PetiteCalculatrice() {}
    public double additionner(double d1, double d0) {
        this.res = d1 + d0 ;
        return this.res;
    }

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

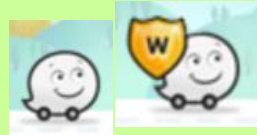
    public double getResultat() {
        return this.res;
    }

}
```

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



[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)



### 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

## 2.9- Les Relations d'association

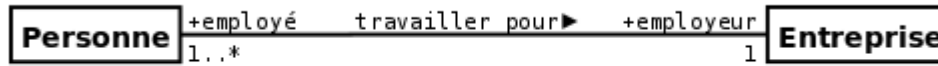


Figure 3.5 : Exemple d'association binaire.



Figure 3.7 : Navigabilité.

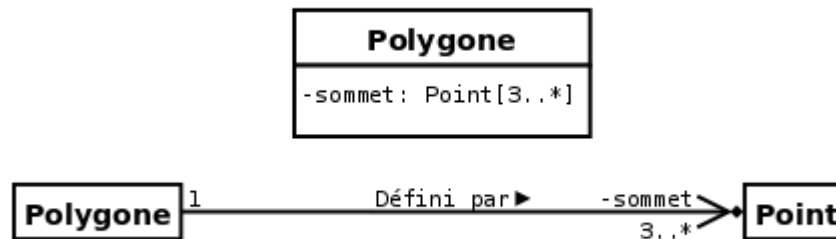
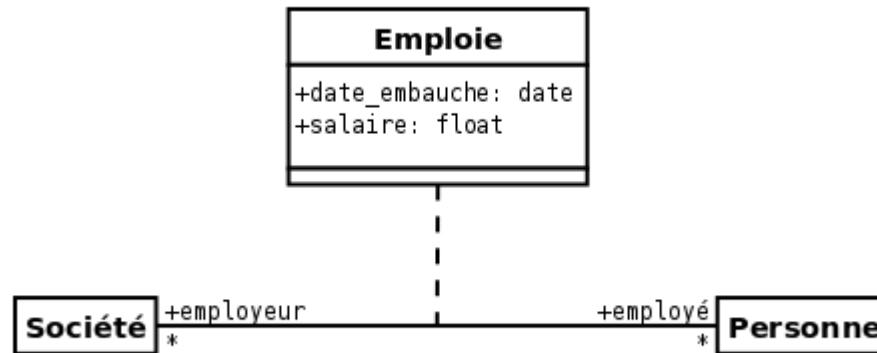


Figure 3.9 : Deux modélisations équivalentes.

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

## 2.9- Les Relations d'association



*Figure 3.11 : Exemple de classe-association.*

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

## 2.9- Les Relations d'association

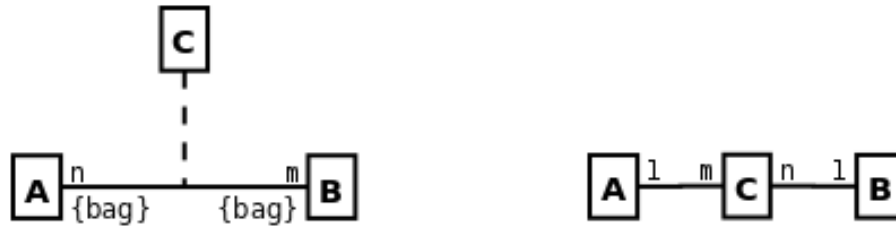


Figure 3.14 : Deux modélisations modélisant la même information.

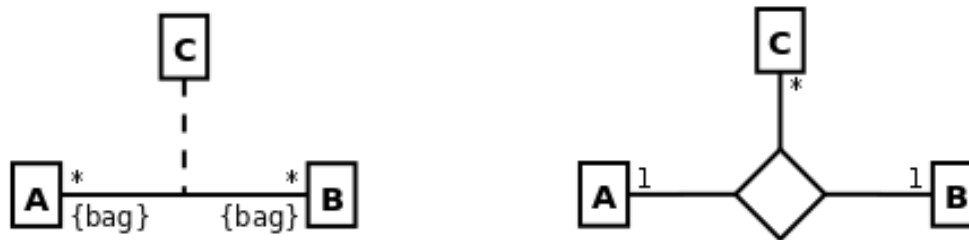
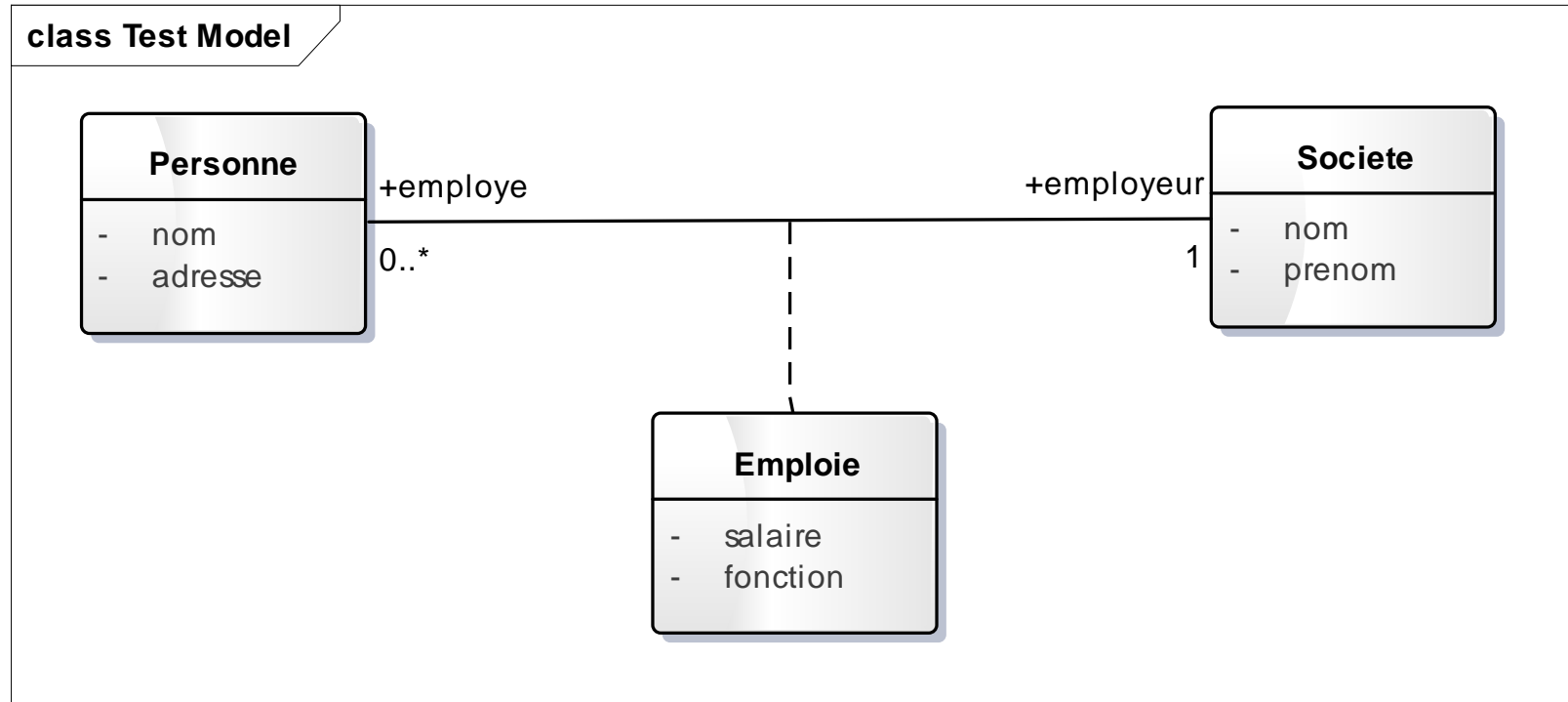
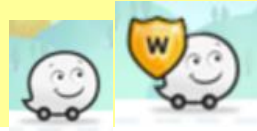


Figure 3.15 : Deux modélisations modélisant la même information.

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

## 2.9- Les Relations d'association





## 2.4- La Classe : 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



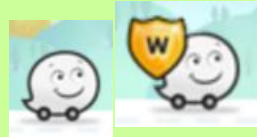
```
public static void main(String[] args) {  
    Personne personne = new Personne("david", "pertuis");  
    Societe societe = new Societe("YNOV", "Aix");  
    Emploi emploi = new Emploi(societe);  
    emploi.set("x0123456789", personne, 500., "formateur");  
    System.out.println(emploi.getPersonne("x0123456789").getNom());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

## 2.6- L'Agrégation & Composition

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.**

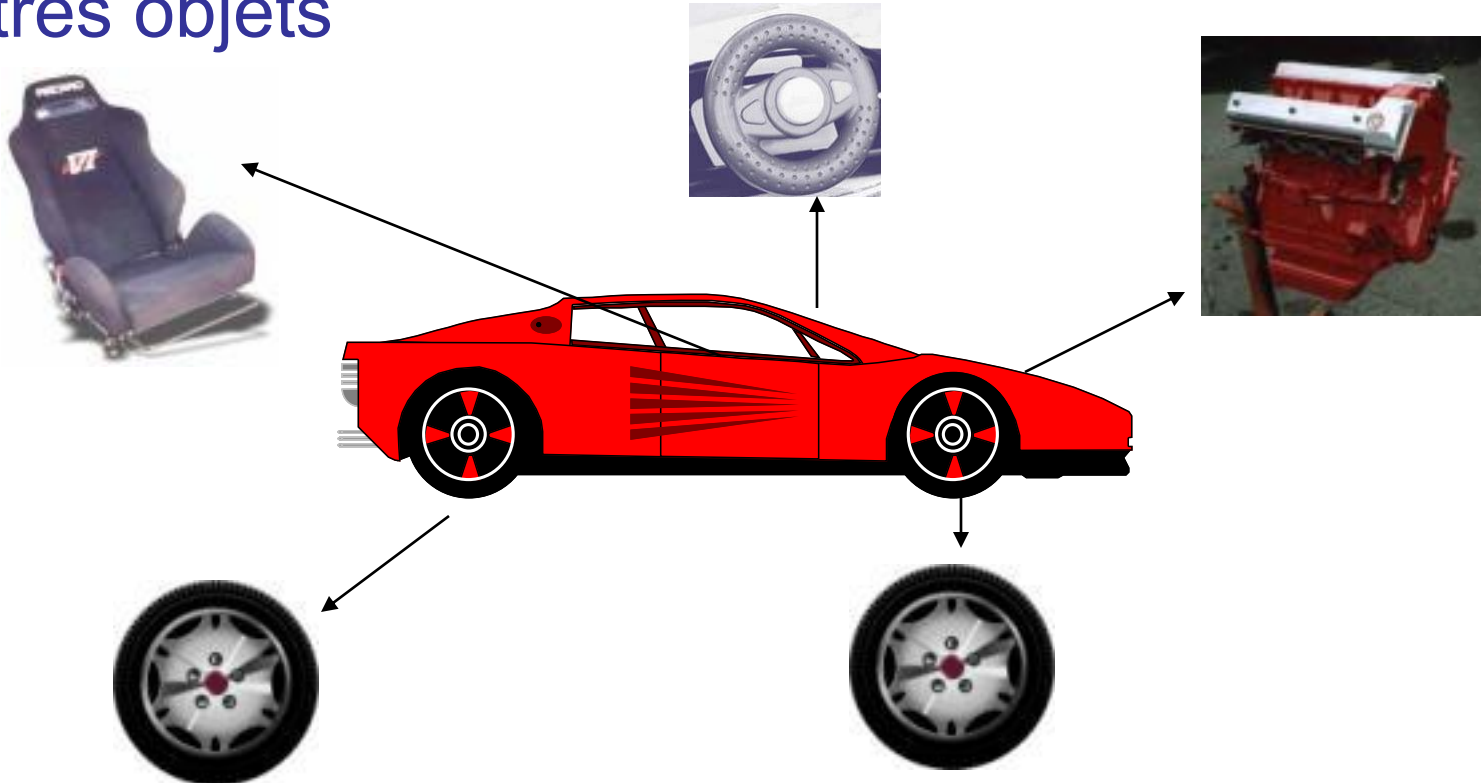
L’agrégation quant à elle est vue comme une relation de type “**a un**” (“has a”), c’est à dire que si un objet A a un objet B alors **B peut vivre sans A**

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

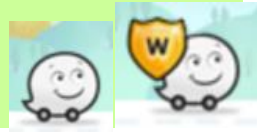


# Agrégation

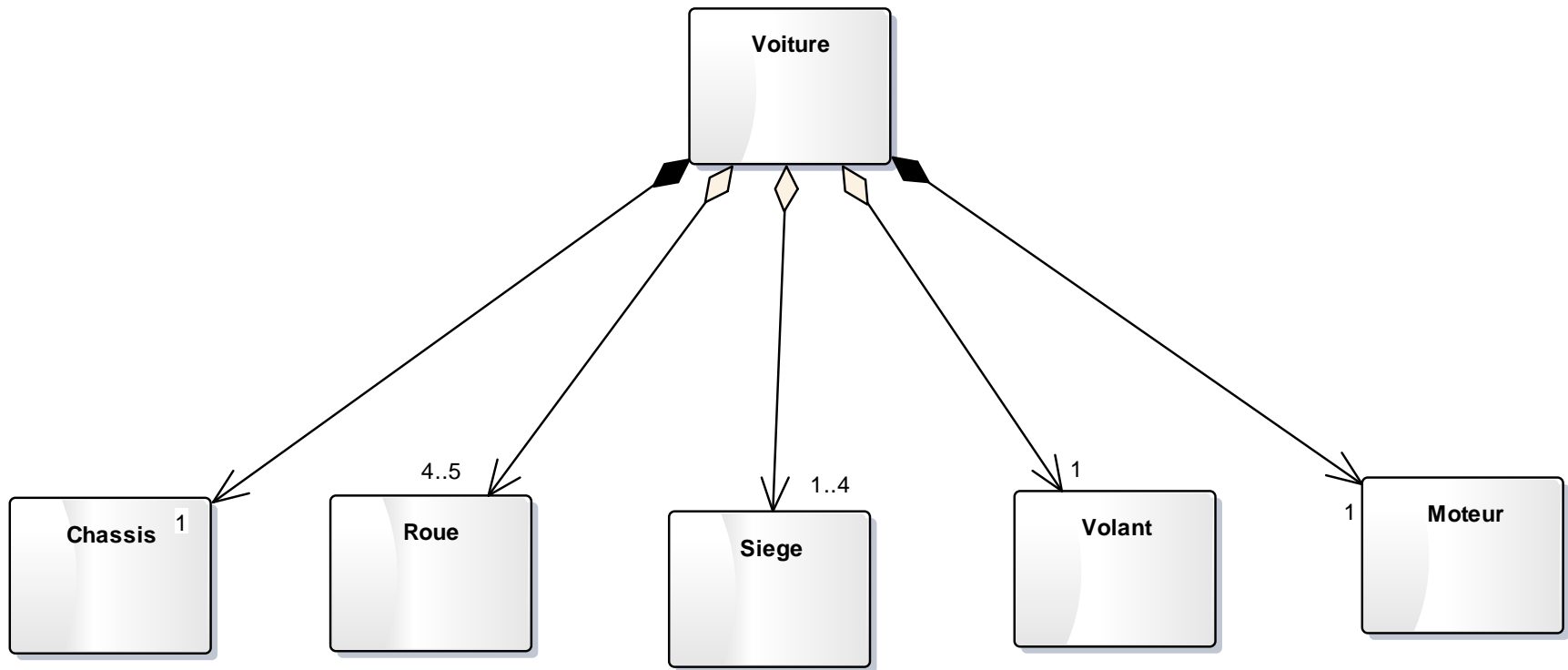
Les objets peuvent contenir des références à d'autres objets



## 2.6- L'Agrégation & Composition



class Class Model



## 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();  
}
```



## 2.6- L'Agrégation & Composition

**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.



**Se préparer à jouer**

**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**

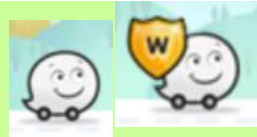
Placez votre Extra Deck face verso à cet emplacement. Vous pouvez regarder le contenu de votre Extra Deck à tout moment durant la partie. Cette zone était auparavant réservée au Fusion Deck. Tous les effets de carte qui s'appliquaient au Fusion Deck s'appliquent dorénavant à l'Extra Deck.

**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).



[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)



### L 'Héritage

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

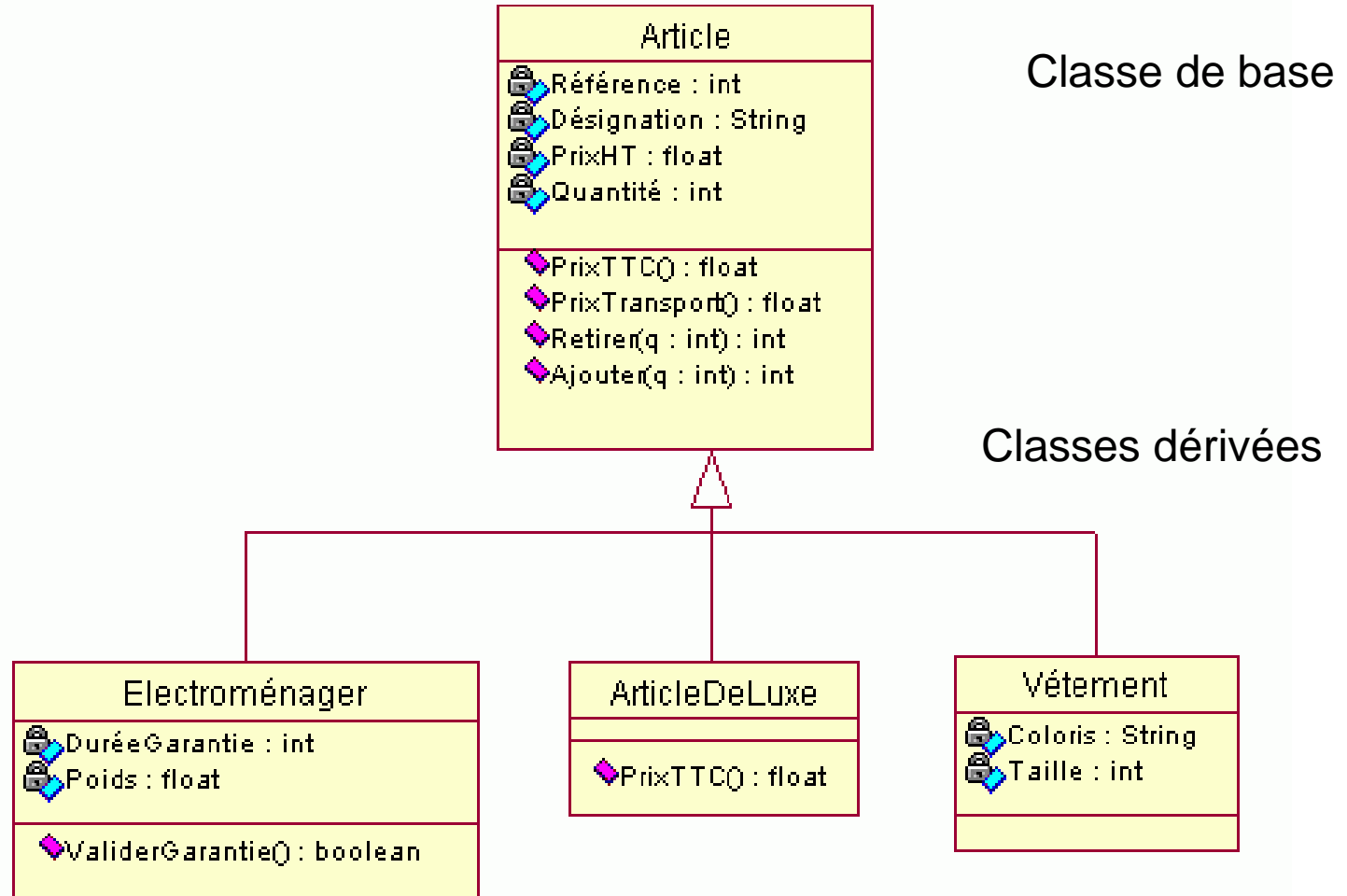
- ⇒ La classe dérivée possède les attributs et méthodes de la classe de base,
- ⇒ la classe dérivée peut en ajouter ou en masquer,
- ⇒ facilite la programmation par raffinement,
- ⇒ facilite la prise en compte de la spécialisation.

## 2.5- L 'Héritage



GENERALISATION

SPECIALISATION



## 2.5- L 'Héritage

**«« 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 Invoker 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 Invoker 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'Invoker par Pendulation !



**«« Monstres Synchro**

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



**«« Monstres Syntoniseur pour Invocation Synchro**

Pour Invoker par Synchronisation un Monstre Synchro, vous aurez besoin d'un Syntoniseur (cherchez le mot "Syn-toniseur" à côté de son Type). Le monstre Syntoniseur 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 Invoker.



**«« Monstres de Fusion**

Les Monstres de Fusion sont également placés dans votre Extra Deck (pas dans votre Main Deck). Ils sont Invokés 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 Invokés 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 Invokées 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.



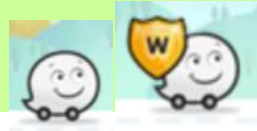
[https://img.yugioh-card.com/ygo\\_cms/ygo/all/uploads/Rulebook\\_v9\\_fr.pdf](https://img.yugioh-card.com/ygo_cms/ygo/all/uploads/Rulebook_v9_fr.pdf)



## 2.5- L 'Héritage



[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)

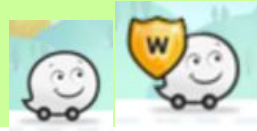


# Le Polymorphisme

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.



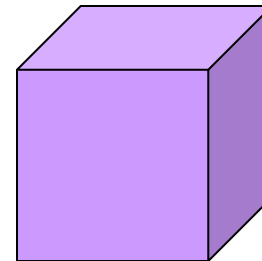
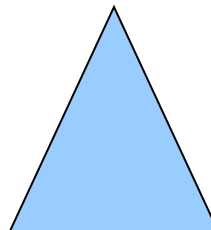
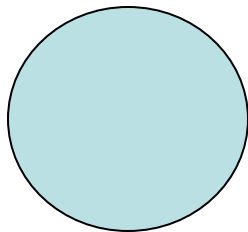
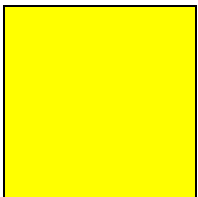
### STATIQUE

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

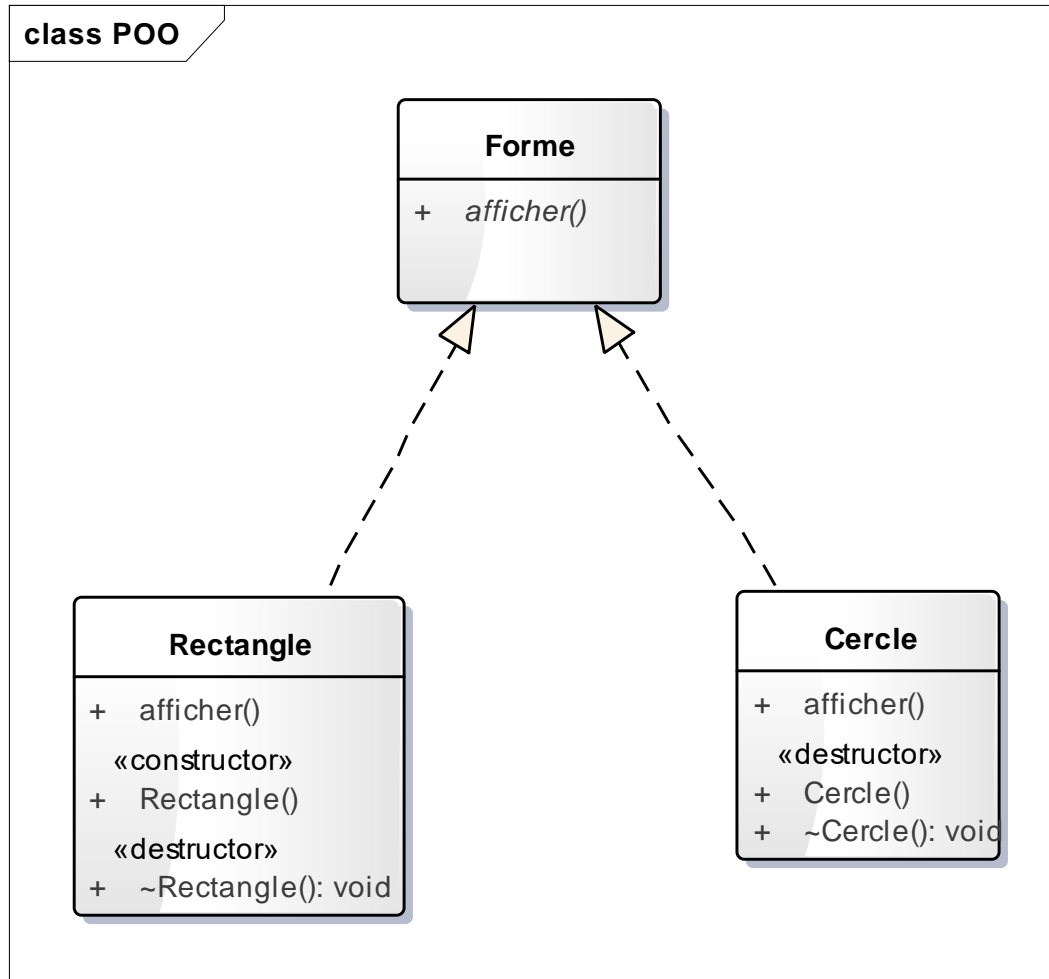
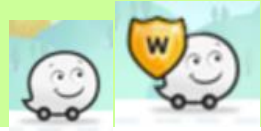
```
un Rectangle.afficher();
un Cercle.afficher();
```

### DYNAMIQUE

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



## 2.7- Le Polymorphisme





## 2.4- La Classe : Java



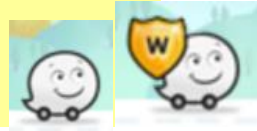
```
public interface Forme {  
    public abstract void afficher();  
}
```

```
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);  
    }  
}
```

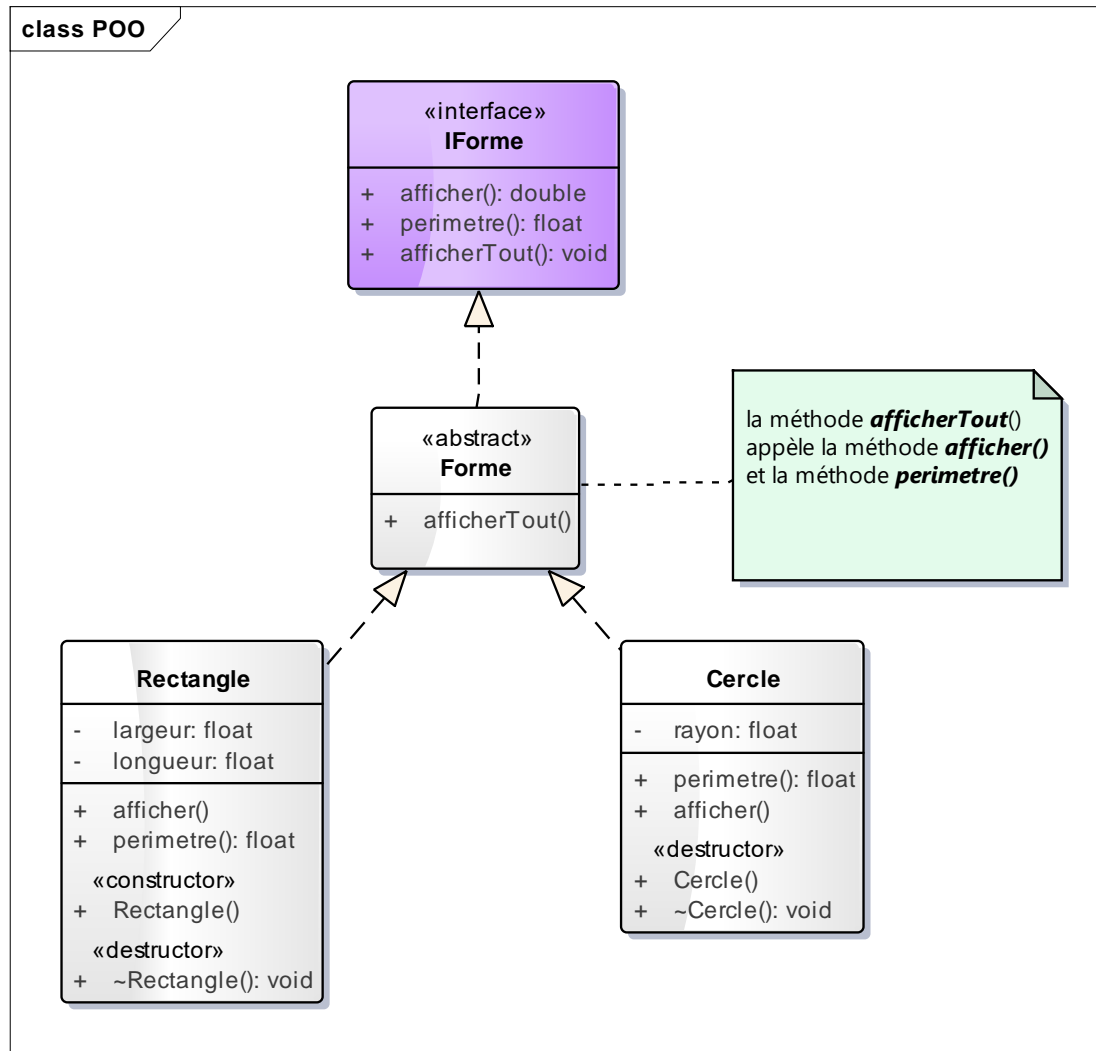
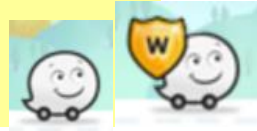
```
run:  
  
Je suis un cercle  
Je suis un rectangle  
Je suis un cercle  
Je suis un rectangle
```



# 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 : 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;

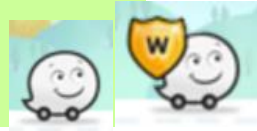
    public Rectangle(double longueur, double largeur) {
        this.a_longueur = longueur;
        this.a_largeur = largeur;
    }

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

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

run:

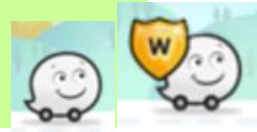
```
Je suis un cercle
Le perimetre de 1 forme est 62.83185307179586
Je suis un rectangle
Le perimetre de 1 forme est 14.0
BUILD SUCCESSFUL (total time: 0 seconds)
```



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

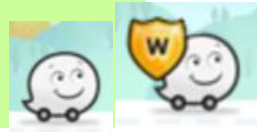
## 2.4- La Classe : Java



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



## 2.10- Clonage, comparaison et assignation



### Clonage d'un Objet

- Créer un objet *identique* à un autre



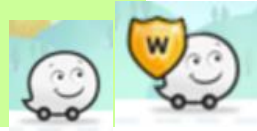
### Comparaison Objet :

- 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.

## 3 - Bibliographies

