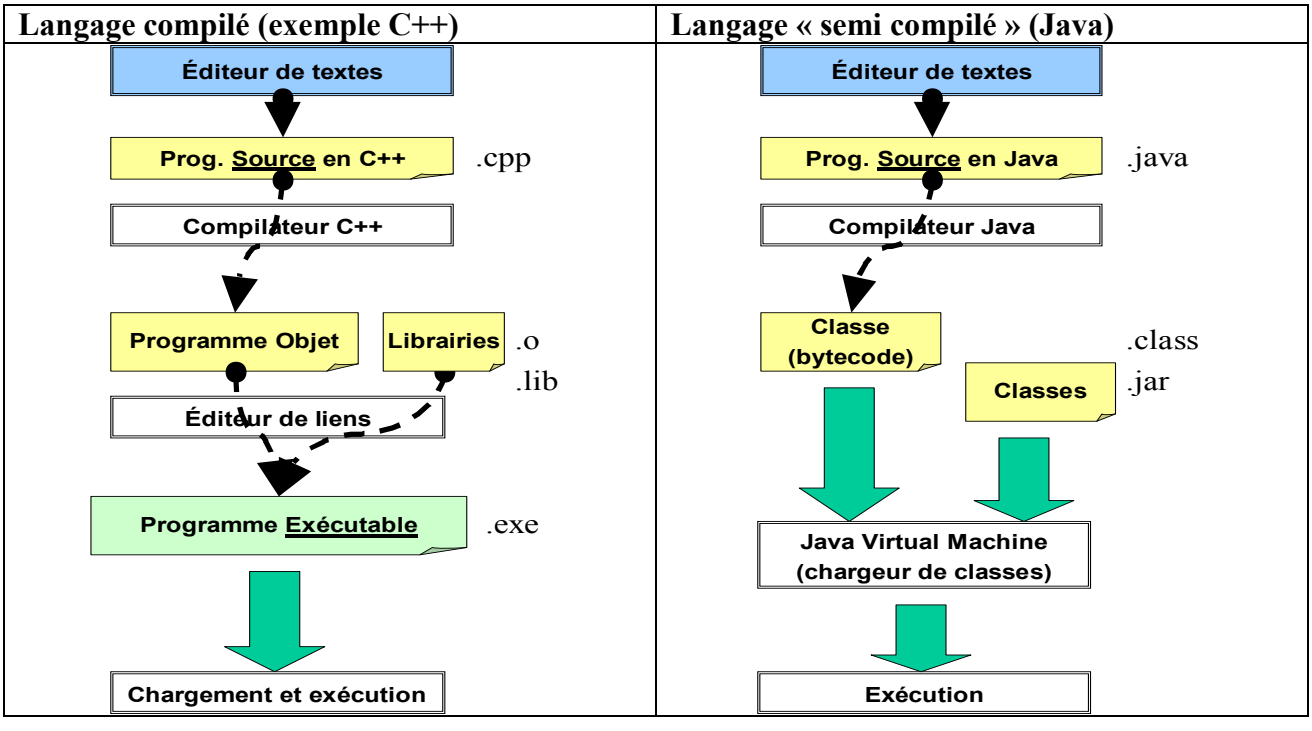
# Ch 1 – Introduction



# I. Introduction

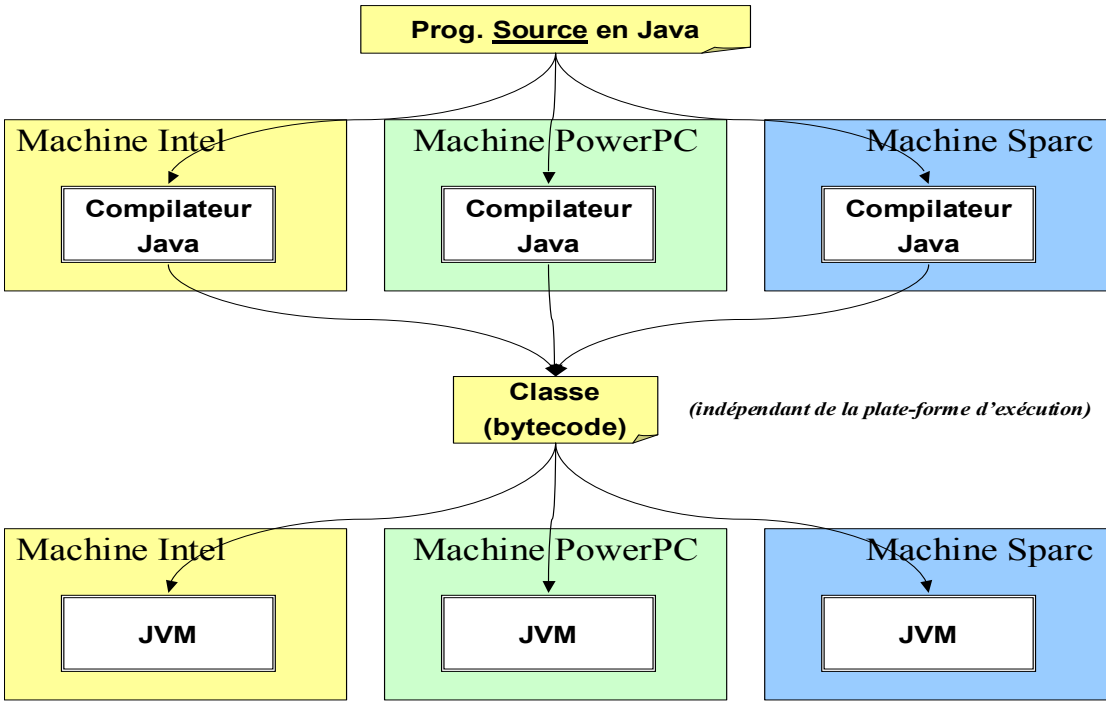
## A. Différence entre Java et les langages compilés

Le langage de programmation Java est un langage de haut niveau (L3G). Il est composé d'un ensemble restreint de types de base, d'instructions et d'opérateurs et un vaste ensemble vaste de classes (API Java). Il permet la Programmation Orientée Objet, POO.
Le langage est traduit dans un langage intermédiaire (bytecode) pour être ensuite exécuté par un chargeur de classes, la machine virtuelle Java (Java Virtual Machine, JVM).



## B. Java, indépendance des plateformes informatiques

Grâce à la compilation dans un langage intermédiaire (« bytecode »), chaque plateforme qui dispose de la JVM peut exécuter le code produit par une compilation réalisée sur n'importe quelle autre plateforme.




## C. Plateforme Java

La plateforme Java constitue l'environnement d'exécution dans lequel un programme Java va pouvoir s'exécuter. (Les programmes exécutables - .exe - ont pour plateforme d'exécution le système d'exploitation lui-même, qui les exécute directement).

La plateforme Java est constituée :

- De la machine virtuelle Java (environnement d'exécution, chargeur de classe) (anglais JVM, Java Virtual Machine)
- D'une collection de classes permettant d'accroître la productivité ; ces classes sont organisées en package, chaque package correspondant à une famille de classes

~~Cf. http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html~~ (Acquisirion de Sun par Oracle)
Cf. http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

La plateforme Java est déclinée en plusieurs versions et permet le développement d'applications classiques, pour le web et pour les mobiles.

# II. Paradigme de Programmation à Objets

## A. Notions de classe et d'objets, principe d'abstraction

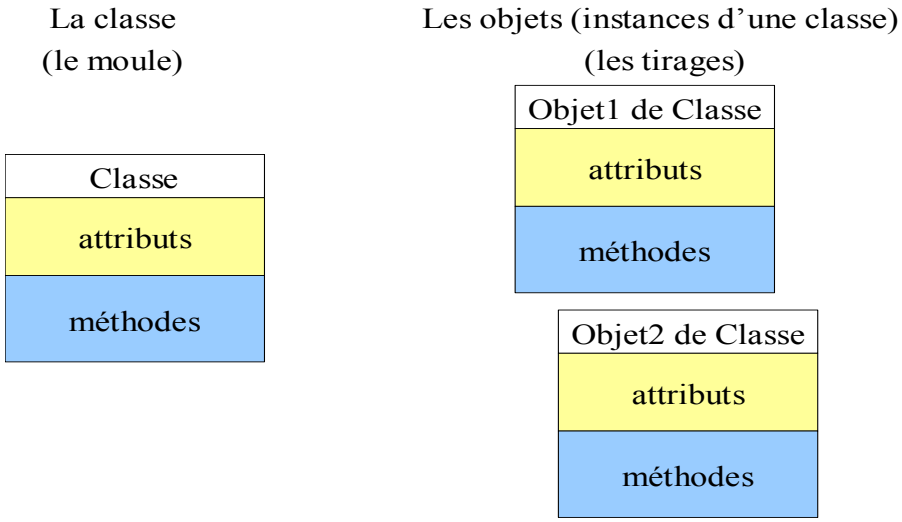
***« le monde est une collection d'objets qui interagissent entre eux »***

Une classe représente un modèle, un moule à partir duquel on pourra tirer des exemplaires, les objets.
Lorsque, à partir d'une classe, on créé un nouvel objet, on parle d'**instanciation** un objet.
**L'objet est une instance d'une classe**.

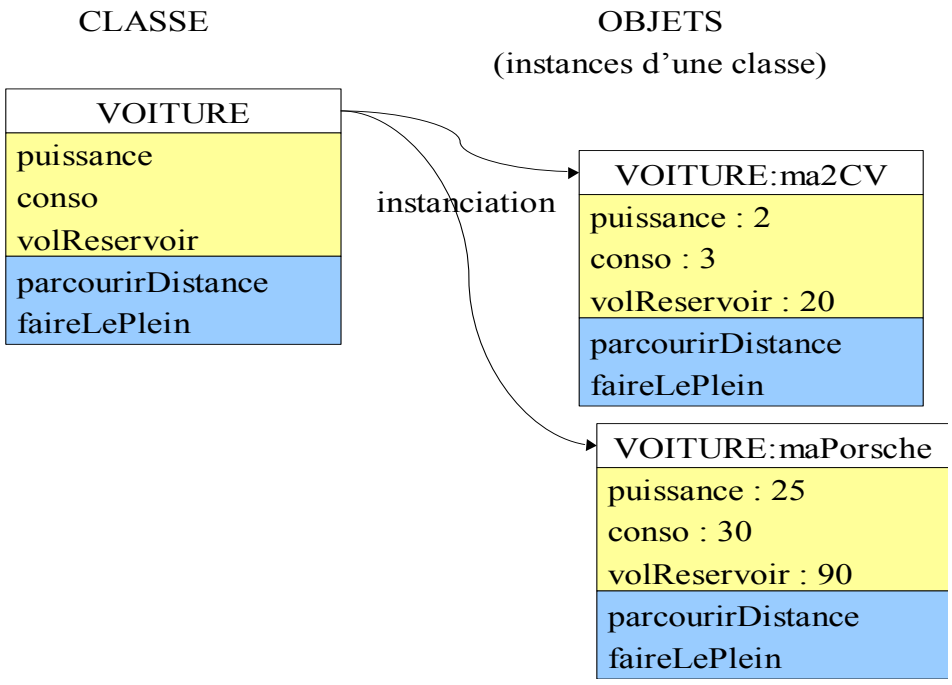
Une classe possède des **membres** :

- membres **attributs**: ce sont les caractéristiques qui permettront de décrire les objets de cette classe (cela ressemble fort aux variables d'une structure) = ce sont des **variables internes à une classe**.
- membres **méthodes** : ce sont des sous-programmes qui permettront de modifier l'état des objets de cette classe (c'est-à-dire modifier les valeurs des attributs d'un objet) = ce sont des **fonctions associées à une classe**.

Une classe regroupe dans un concept unique les attributs et les méthodes chargées d'exploiter ces attributs.



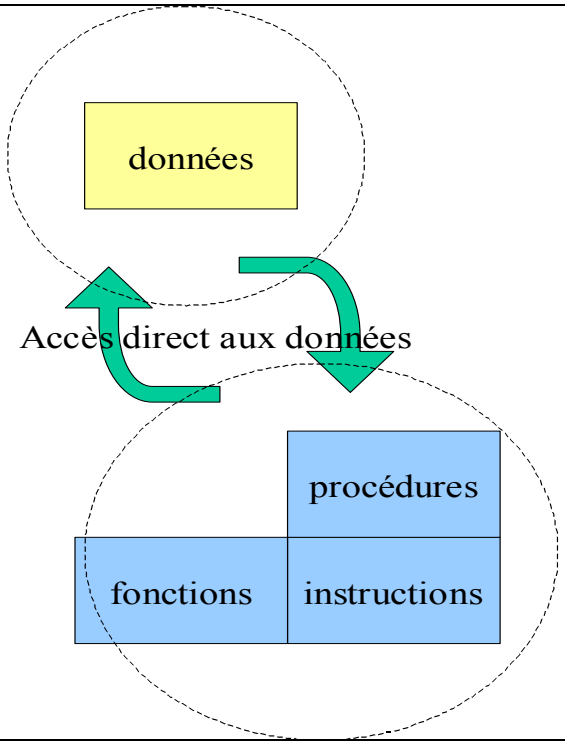
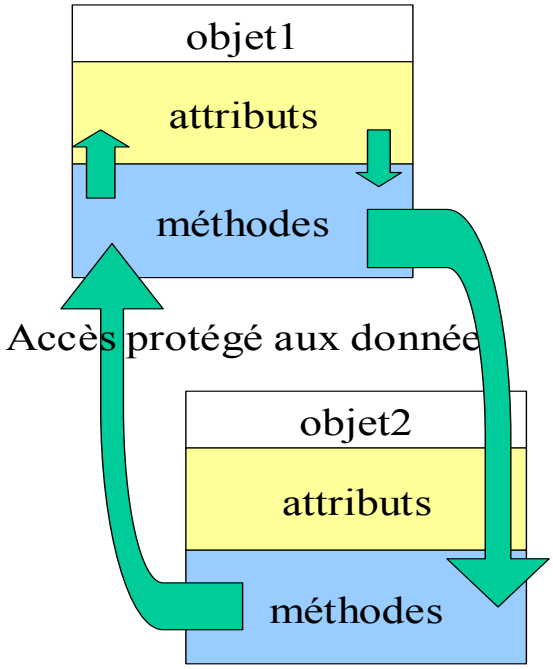

Exemple : une classe VOITURE et 2 instances de cette classe



## B. Objets et échange de messages

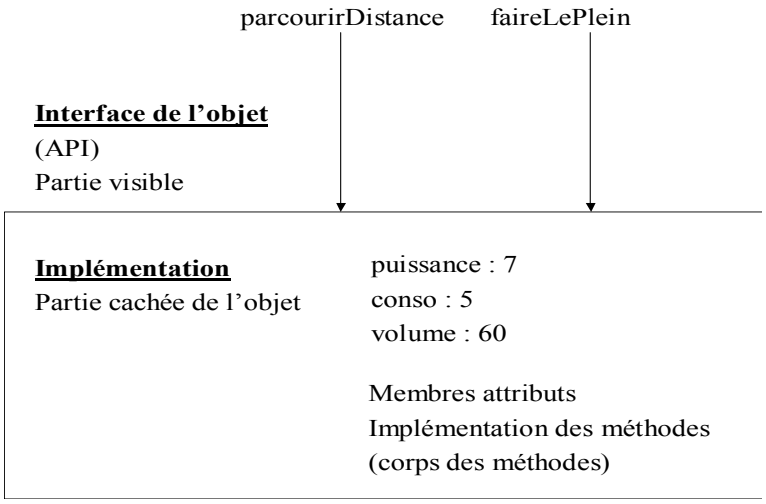
Alors qu'en programmation procédurale, un programme est vu comme une hiérarchie de procédures et fonctions modifiant des données communes, le programme objet est vu comme une collaboration entre objets, avec échange de messages (correspondant aux appels des méthodes).

Une classe décrit tous les objets du même type (possédant les mêmes propriétés et mêmes méthodes). Les méthodes pourront être décrites en terme d'algorithmes, mais l'ensemble du programme sera présenté comme un diagramme de collaboration entre objets..

| **Programmation procédurale** | **Programmation orientée objet** |
|---|---|
| On définit les variables nécessaires (données) et on créée les instructions (groupées en procédures et fonctions) qui vont manipuler ces données | Les données et les sous-programmes qui les manipulent sont regroupés dans un objet avec des propriétés (les données caractérisant l'objet), des méthodes (procédures et fonctions, déterminant le comportement de l'objet, c'est-à-dire ce qu'il propose de faire de ses propriétés) |
| Un programme est décomposé en une hiérarchie de sous-programmes | Un programme est décomposé en un certain nombre de d'objets qui vont communiquer grâce à l'appel de méthodes |
|  |  |
| Evolution et réutilisation difficile | Evolution facile par la notion d'héritage |

## C. Principe d'encapsulation, contrôler l'accès aux membres

Un objet ne peut être manipulé que via les méthodes qui lui sont associées. L'encapsulation correspond aux mécanismes de protection des membres attributs, uniquement (en général) accessibles par l'intermédiaire des méthodes autorisées (méthodes d'interface). En cachant l'implémentation des propriétés d'une classe, des modifications de la classe sont possibles sans remettre en cause le code utilisant cette classe. L'encapsulation assure la protection des données et une indépendance d'implémentation de ces données.




La restriction d'accès direct à la partie privée de l'objet induit la définition de méthodes publiques de lecture et de modification des propriétés de l'objet.

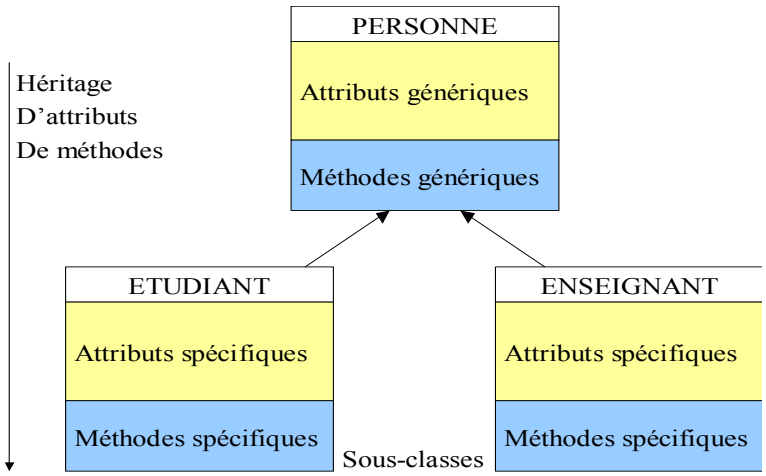
- Accesseur (anglais : Accessor ) ou « getter »: méthode permettant l'accès à une information privée (=on questionne l'objet au sujet de lui-même)
- Mutateur (Mutator) ou « setter » : méthode permettant la modification d'une information (= on modifie l'état d'un objet)

Exemple (incomplet) :

```
/* fichier  Personne.java */
public class Personne extends Object { // debut de la classe
  // membres attributs
  private String nom = « dupont »;

  // membres méthodes
     // accesseur nom
  public String getNom() {
    return nom;
  } // fin méthode accesseur nom

     // mutateur nom
  public void setNom(String unNom) {
    this.nom = unNom;
  } // fin méthode mutateur nom
...
} // fin de la classe
```

## D. Principe d'héritage, bénéficier des comportements des ancêtres

Différents types de liens peuvent mettre des objets en relations : « a un » (has_a), « utilise un » (uses_a ), « est un » (is_a). L'héritage correspond à la relation « est_un ».
Le lien d'héritage permet le transfert de toutes les propriétés d'une classe (la classe maître, classe de base ou superclasse) à une ou plusieurs sous-classes(ou classes dérivées) Il permet de factoriser (capitaliser) les connaissances pour créer de nouveaux objets.




Les sous-classes bénéficient des membres hérités, peuvent redéfinir certains membres et en définir de nouveaux.
Toute instance d'une sous-classe est instance de la classe.

Redéfinition des propriétés :

- Surcharge : redéfinition d'une méthode héritée avec un code spécifique
- Polymorphisme : faculté d'une méthode à pouvoir être appliquée à des objets de classes différentes (paramètres différents)

Deux formes de généralisation :

- Spécialisation de catégorie : est_un (une personne est un homme ou est une femme)
- Spécialisation d'état : a_le_role_de (une commande est une commande en attente ou une commande livrée)

## E. Principe du polymorphisme

Le principe de polymorphisme autorise le fait d'avoir plusieurs méthodes avec un nom identique mais mettant en œuvre un comportement différent.
On peut relier le polymorphisme au principe d'héritage : une superclasse définit une méthode générale ayant un comportement standard (le comportement par défaut), des méthodes dérivées redéfinissant cette méthode (si nécessaire) avec un comportement spécifique.
On peut également disposer de plusieurs méthodes de même nom ayant chacune une signature différente. L'appel de la méthode adéquate sera effectué grâce à la signature.
Exemple (incomplet) :

```
public class Biblio extends Object { // debut de la classe
  // membres attributs
  static final private double penaliteParJour = 2 ;
  private Date dateRetour ;

  // membres méthodes
  public void setDateRetour (Date uneDateRetour) {
    this.dateRetour = uneDateRetour;
  } // fin méthode
  public Date getDateRetour () {
    return this.dateRetour;
  } // fin méthode

  public int joursRetard () {
```

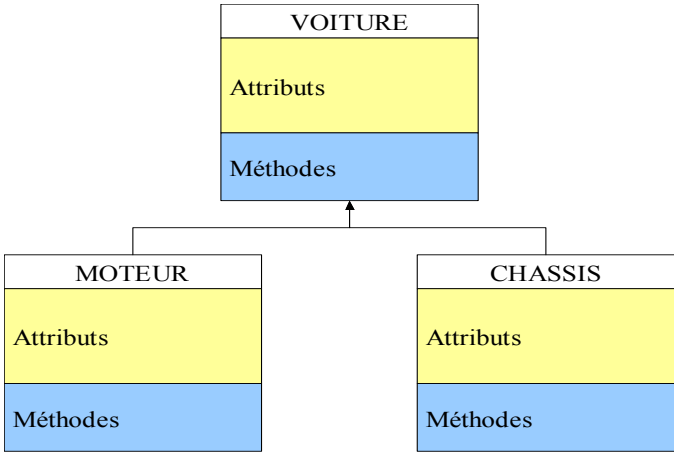
```
    return dateDiff(date(), dateRetour);
  } // fin méthode


  public int penalités () {
    if (joursRetard < 0) return 0
    else return joursRetard()* penaliteParJour;
  } // fin méthode


} // fin de la classe

public class Magazine extends Biblio { // debut de la classe

} // fin de la classe

public class Livre extends Biblio { // debut de la classe
  public int penalités () {
    if (joursRetard < 0) return 0
    else return joursRetard()* 0.75;
  } // fin méthode
} // fin de la classe
```

### F. Lien d'agrégation, relation d'appartenance

Un lien d'agrégation définit une relation d'appartenance entre un objet agrégat et un ou plusieurs objets composants.
Deux formes de liens d'agrégat :

- Les composants sont dépendants cd l'agrégat (des composants visuels de la fenêtre)
- Les composants ont une existence propre

Le lien d'agrégat permet de créer des objets complexes.




## III. Une Classe en Java

Exemple (incomplet) :

```
/* fichier  Voiture.java */
```

```
public class Voiture extends Object { // debut de la classe
  // membres attributs
  private int puissance ;
  private double conso ;
  private int volumeReserv ;
  private int reste ;

  // membres méthodes
     // constructeur
  public Voiture(int puissance, double conso, int volumeReserv)
{
    this.puissance = puissance;
    this.conso = conso;
    this.volumeReserv = volumeReserv;
    reste = volumeReserv;
  } // fin méthode constructeur

  public void parcourirDistance(int uneDistance) {
        int carburant = (int)(uneDistance * getConso() / 100);
        reste = reste - carburant;
        if (reste<0) reste = 0;
  } // fin méthode parcourirDistance

  public double getConso() {
        return conso;
  } // fin méthode getConso
} // fin de la classe
```