Programmation Orientée Objet

(Langage JAVA)

L2 ISIL USTHB
Dr Manel Seddiki

Présentation

Dr Manel Seddiki
Maître de conférences
Département Informatique
FEI-USTHB
sed.manel@gmail.com
https://manel-seddiki.jimdo.com/

Objectifs

- Apprendre les notions de programmation orientée objet.
- Apprendre les bases du langage Java

Pré-requis

- Notions d'algorithmique
- Connaissances en programmation procédurale

Organisation

- Un cours par semaine
- Un TD par semaine
- Un TP par semaine

Evaluation

- Une évaluation en TP : assiduité, participation, mini projet, test écrit
- Un évaluation en TD: assiduité, participation
- Un ou deux contrôles continus
- Une épreuve finale

Plan

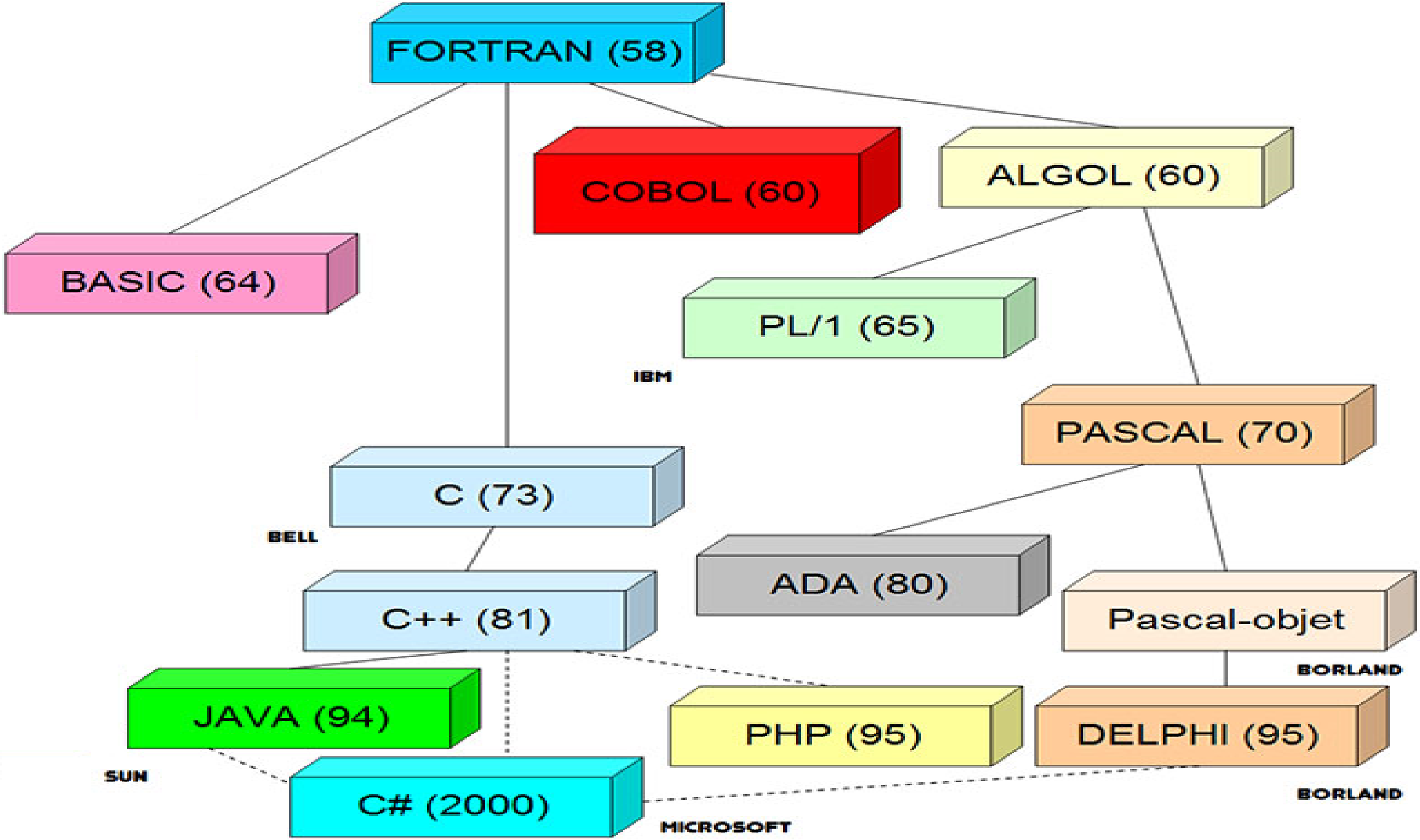
- Introduction
- Concepts de base de la programmation orientée objet
- Héritage et Polymorphisme
- Les classes abstraites et Interfaces
- Les exceptions
- Les collections

I. INTRODUCTION

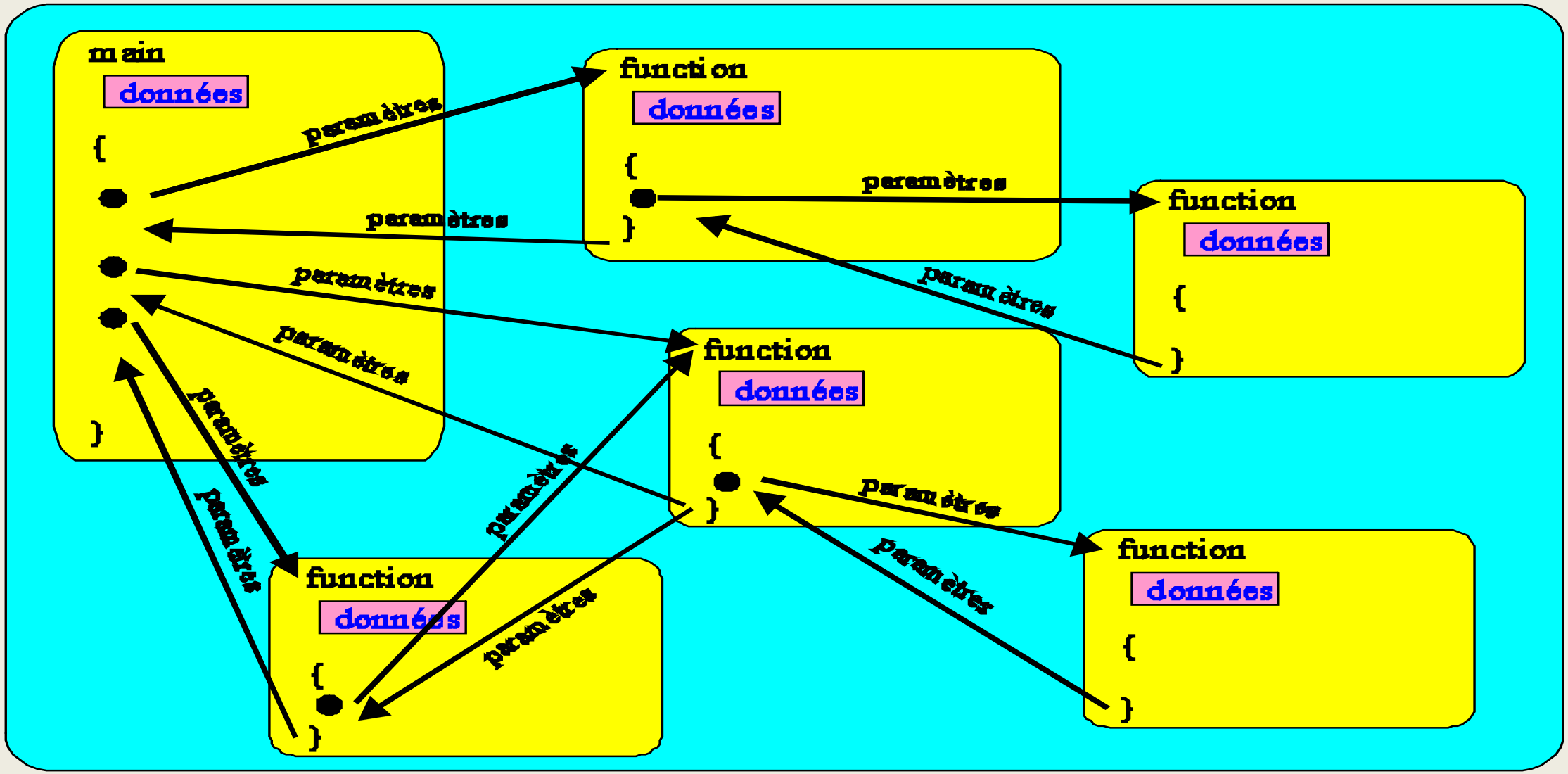
Les paradigmes de programmation

Trois principales catégories :

- Programmation procédurale : P.P. (Pascal, C,..)
- Programmation fonctionnelle (Ocaml, Haskel..)
- Programmation Orientée Objet : P.O.O. (C++, Java, Delphi,…)


FORTRAN (58)
COBOL (60)
ALGOL (60)
BASIC (64)
PL/1 (65)
IBM
PASCAL (70)
C (73)
BELL
ADA (80)
C++ (81)
Pascal-objet
BORLAND
JAVA (94)
PHP (95)
DELPHI (95)
SUN
BORLAND
C# (2000)
MICROSOFT

Un programme en C



Limitations

- Il n'y a pas de méthode ou de cadre pour bien organiser les fonctions.
- Les modifications d'une fonction entraînent d'autres modifications dans d'autres fonctions, etc.
- La portée d'une modification est trop grande et difficile à gérer.
- Redondance dans le code (la même chose est codé plusieurs fois)
- Propagation des erreurs et débogage difficile

Notion de Programmation Orientée Objet (P.O.O.)

- Le programmeur en langage C s'intéresse en priorité **aux traitements** que son programme devra effectuer.

Traitements → Données

- La programmation orientée objet propose une méthodologie centrée sur **les données**.

Données → Traitements

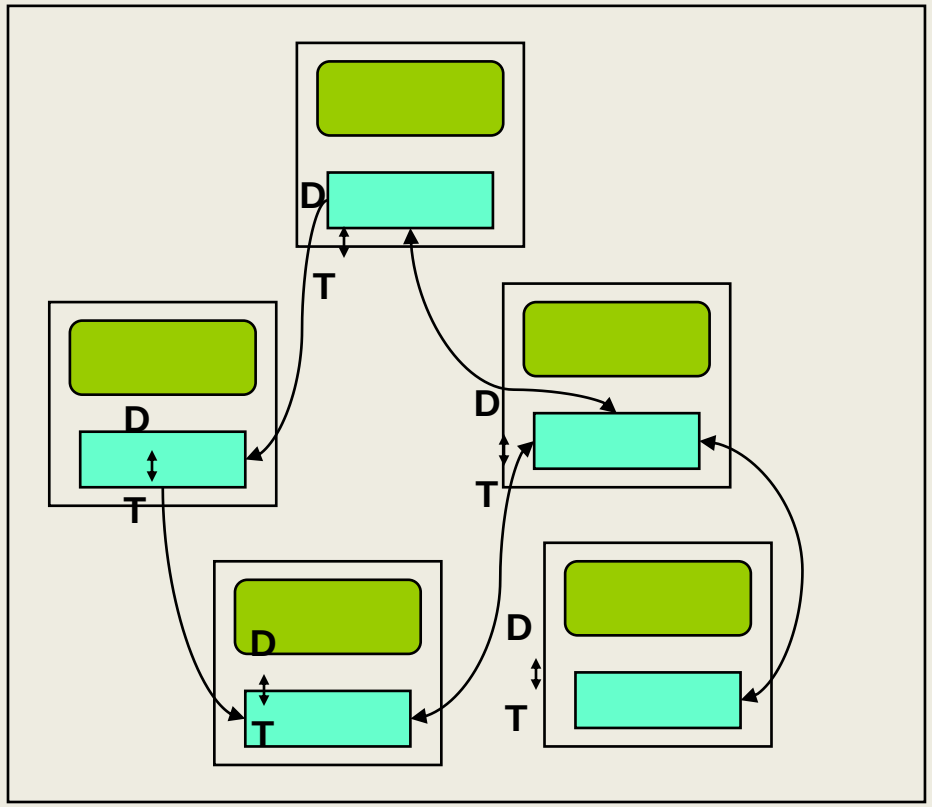
Les avantages de la POO

- Le concept objet permet d'obtenir des logiciels fiables, évolutifs et faciles à maintenir.
- Facilité d'organisation, lisibilité, réutilisation, possibilité d'héritage, facilité de correction, projets plus faciles à gérer.
- Ces programmes sont, de plus, souvent très stables.
- Le programme est sécurisé en interdisant ou autorisant l'accès à ces objets aux autres parties du programme.
- Les modèles à objets ont été créés pour modéliser le monde réel.

Langages Orientés-Objet

Un programme est composé de plusieurs objets qui contiennent :

- des données "internes"
- des traitements manipulant ces données internes ou d'autres données



Questions

- Approche procédurale :

  "Que doit faire mon programme ? "

- Approche orientée-objet :

"De quoi doit est composé mon programme? "

II. Concepts de base de la programmation orientée objet

Plan

- Notions Classes et Objets
- Création d'une classe
- Attributs et méthodes
- Encapsulation
- Surcharge des méthodes
- Constructeurs
- Instanciation (création d'un objet)
- Variables et méthodes d'instance
- Variables et méthodes de classe
- Accesseurs

Classes et Objets

- **Classe :**

Une classe = un type complexe qui regroupe des ***données*** (attributs) + fonctions (***méthodes***) *agissant sur ces données.*

- **Objet :**

Un *objet* = une **variable** de type complexe

Le type d'un objet est appelé : Classe

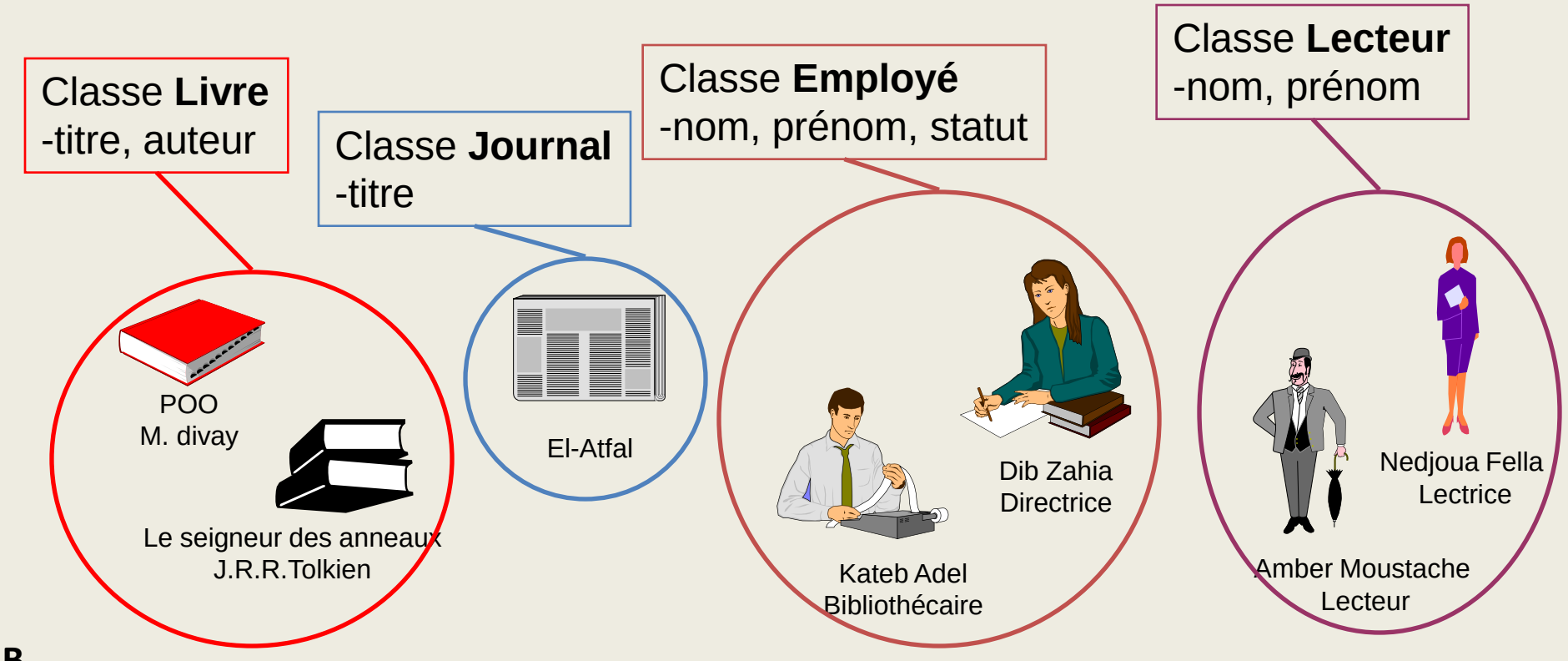
Classes et Objets

- Une *classe* est la définition d'un type, alors que l'*objet* est une déclaration de variables.
- Après avoir créé une classe, on peut créer autant d'objets que l'on veut basés sur cette classe.

Classes et Objets

Des objets similaires peuvent être décrits par une même abstraction : **une classe**

- même **structure de données** et **méthodes de traitement**
- valeurs différentes pour chaque objet

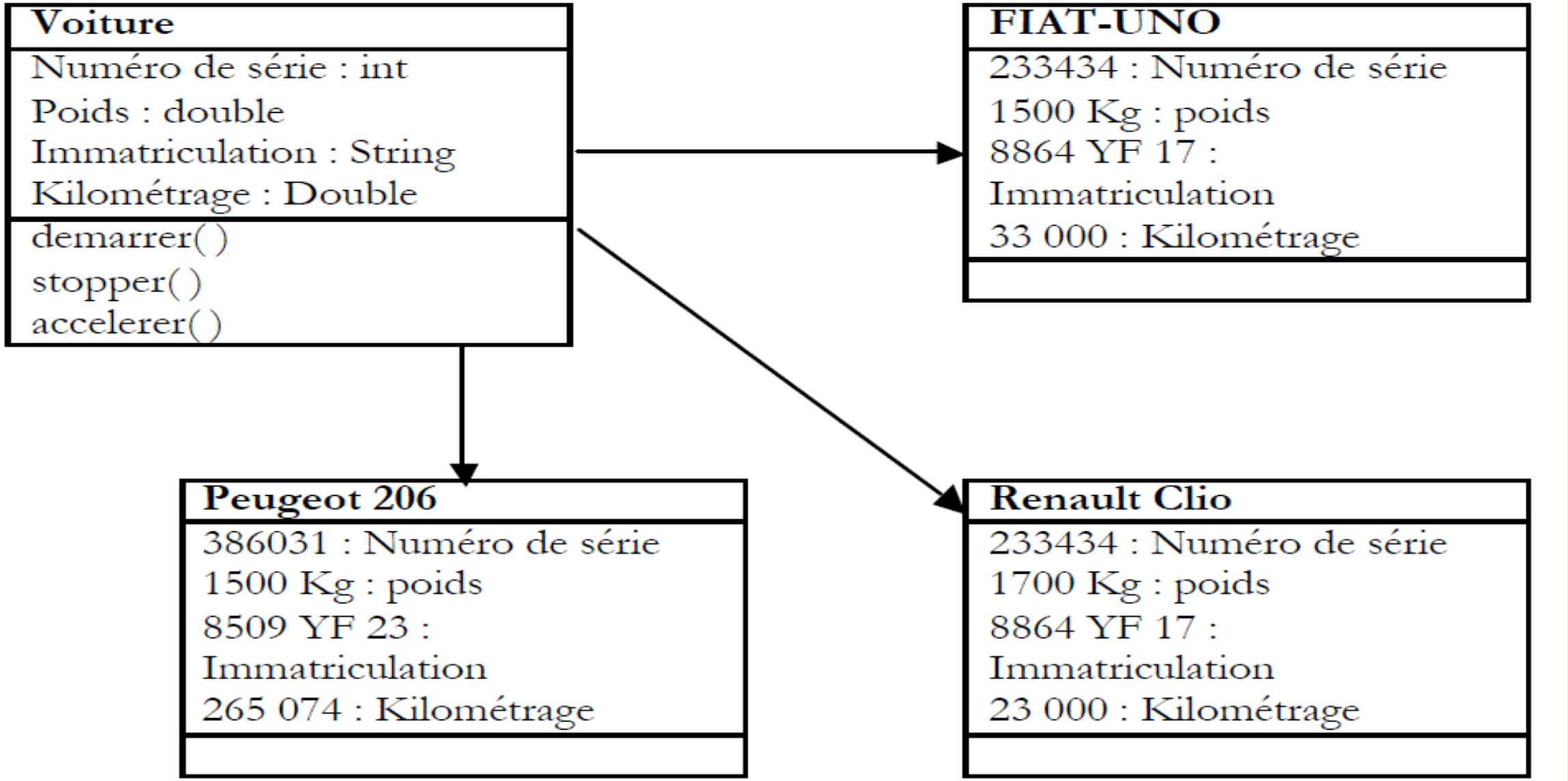


Résumé

- *Classe = type*
- *Objet = variable ayant une classe comme type*
- *Un objet est caractérisé par :*
  - *Des données (appelées attributs)*
  - *Des traitements (appelés méthodes)*
- *On distingue :*

- La description des objets : CLASSE (des livres, des lecteurs…)

- Les objets de chaque type : INSTANCE (Algorithmique, Abed Mohamed…)


Voiture
Numéro de série : int
Poids : double
Immatriculation : String
Kilométrage : Double
demarrer( )
stopper( )
accelerer( )
FIAT-UNO
233434 : Numéro de série
1500 Kg : poids
8864 YF 17 :
Immatriculation
33 000 : Kilométrage
Peugeot 206
386031 : Numéro de série
1500 Kg : poids
8509 YF 23 :
Immatriculation
265 074 : Kilométrage
Renault Clio
233434 : Numéro de série
1700 Kg : poids
8864 YF 17 :
Immatriculation
23 000 : Kilométrage

Identité

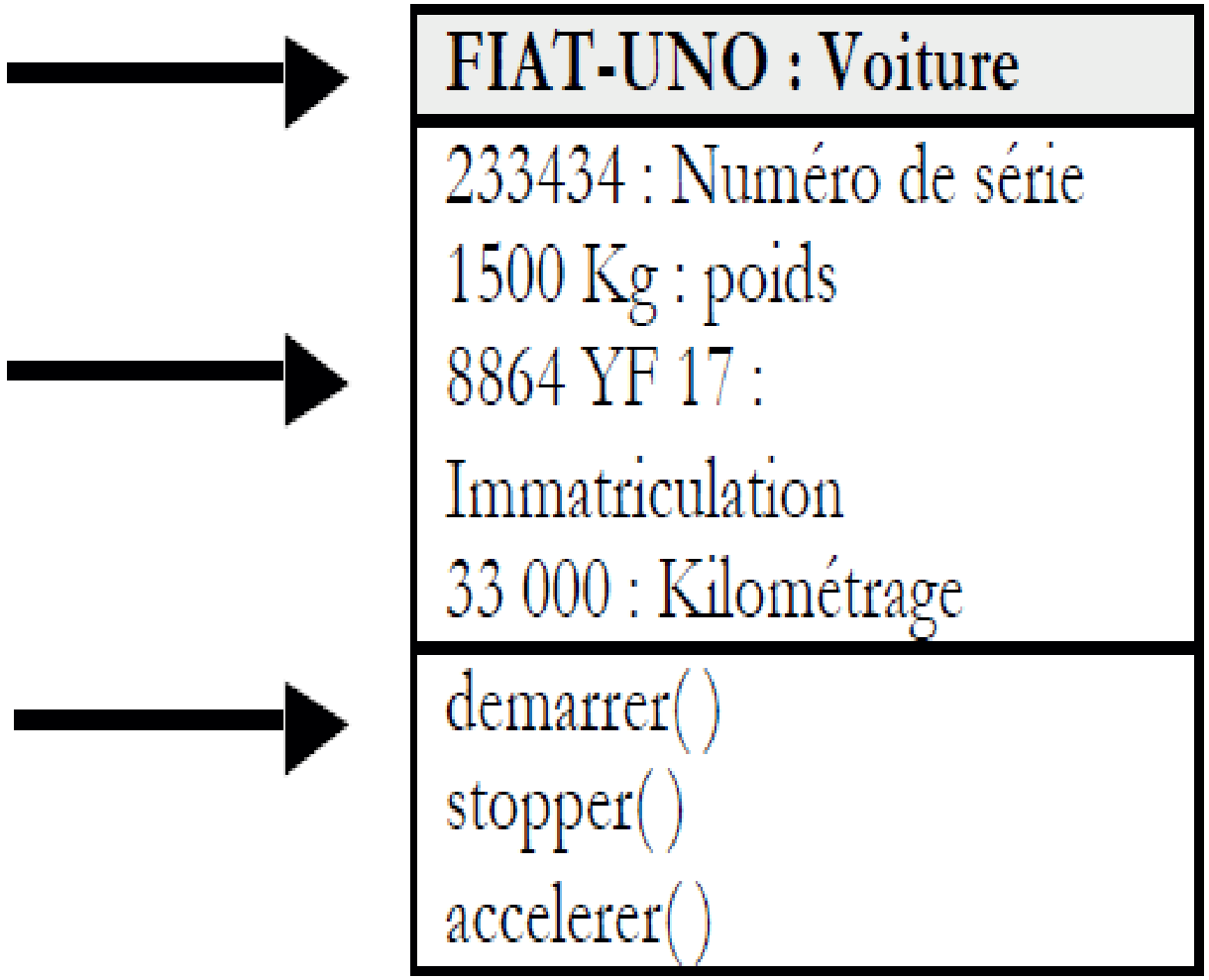
Permet de le distinguer des autres objets

Attributs

Données caractérisant l'objet

Méthodes

Actions que l'objet est à même de réaliser



Création d'une classe

```
class NomClasse    {
            //Attributs
                ...
            //Méthodes
                ...
}
```

Par convention le nom de la classe doit débuter par une majuscule.

Attributs

- *Un attribut est appelé aussi :variable, champ, donnée membre.*
- *Un attribut représente la **caractéristique** de l'objet et la valeur de l'attributs représente l'**état** de l'objet.*
- *Un attribut peut être de type:*
  - *primitif (simple : int, float, char , String…)*
  - *ou de type structuré (objet)*
- *Définir un attribut*

<typeAttribut> :<idf_attribut>;

Méthodes

- Les méthodes représentent l'ensemble des traitement, **procédures, fonctions** ou opérations que l'on peut associer à une classe.
- La ***signature*** d'une méthode contient notamment :
  - ✓un nom de méthode;
  - ✓un type de données de retour;
  - ✓des types de données pour ses arguments;
  - ✓Les arguments peuvent être de type primitif ou structuré

Méthodes

<typeRetour> <nomMéthode> (<arguments>) {

<Traitement associé>

}

*Par convention le nom des **méthodes et attributs** commencent toujours par une **minuscule.***

Exemple

```
class Date {
    int jour, mois, année;      // 3 attributs
boolean compareDate (Date d1, Date d2) {
...                             // arguments de type structurés
}
void modifieAnnée(int an) {  // argument de type simple
...
}
void printDate( ) {  // sans arguments
      ...
  }
  ...       } // end class Date
```

Encapsulation

- On parle d'*encapsulation* pour désigner :
  - le regroupement des données et des traitement au sein d'une classe
  - Avec la possibilité de cacher leur existence ou non en dehors de la classe.

Exemple: La machine à café.

Encapsulation

- **public** : les autres objets peuvent **accéder** à la valeur de cette donnée ainsi que la **modifier** ;
- **private** : les autres objets **n'ont pas le droit d'accéder** directement à la valeur de cette donnée (**ni de la modifier**). En revanche, ils peuvent le faire indirectement par des méthodes de l'objet concerné (si celles-ci existent en accès public).

Exemple

```
class Date {
 private int jour, mois, an;      // 3 attributs  (ou données membres)
  …
 public boolean compareDate (Date d1, Date d2) { // argument de type structuré
  …
 }
 public void modifieAnnée(int an) {  // argument de type simple
  …
 }
 public void printDate() {
  …
 }
  …       } // end class date
```

Surcharge des méthodes

- Des méthodes avec **le même nom** mais avec des paramètres et/ou valeurs de retour différents.
- Java décide de la méthode à appeler en regardant la valeur de retour et les paramètres.

Exemple

```
class Date {
    private int jour, mois, année;      // 3 attributs
    …
    public void printDate( ) {  // sans arguments
        …
    }
   // Méthode surchargée
    public void printDate(Date d) {  // avec  arguments
        …
     }
…
 } // end class Date
```

Le constructeur

- **Une méthode** particulière invoquée lors de la création d'un objet.
- Son rôle est d'initialiser les valeurs des attributs de l'objet créé.
- Un constructeur doit avoir le **même nom que la classe** où il est défini et n'a aucune valeur de retour (c'est l'objet créé qui est renvoyé).
- Il est possible de surcharger les constructeurs
- En java, il y'a un constructeur vide par défaut (si aucun constructeur n'est défini)

Exemple

```
class Date {
  private int jour, mois, an;      // 3 attributs
  public Date() {                  // définition du constructeur par défaut
    jour = mois = an = 0;
  }
  public Date(int j, int m, int a) {     // constructeur surchargé
    jour = j; mois = m; an = a;
  }
  public void printDate() {
    // ...
  }
  // ...       } // end class date
```

Attention: pas de mot clé **void** dans un constructeur!

Instanciation

- **Créer un objet** à partir d'une classe => créer une instance de la classe.
- Avant de créer l'objet, **il faut le déclarer**

Attention: La déclaration seule ne crée pas d'objet. Elle l'initialise à **null** qui ne fait référence à rien.

Création d'un objet: Instancier une classe

•Exécuter **new** suivi du **constructeur** de la classe instanciée.

•Exemple:

```
// appel au constructeur sans paramètres
Date d1=new Date();
// appel au constructeur avec paramètres
Date d2=new Date(12,4,2014);
```

Remarque importante

- *Toute variable désignant un objet est un pointeur*
  - Le passage d'**objets** comme paramètres d'une méthode est **toujours un passage par référence**.
  - À l'inverse, le passage de **variables primitives** comme paramètres est **toujours un passage par valeur**.

Garbage collector « ramasse miettes »

- Un système de libération de l'espace mémoire
  - L'objet est détruit dès qu'il n'est plus référencé par un pointeur.

Le mot-clé This

- **This** sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation.
- Il est utilisé dans la classe dans laquelle on se trouve.
- Permet de lever l’ambiguité des identificateurs

Exemple 1

```
public class Maclasse {
   private Date d;
   public Maclasse() {
         d=new Date() ;  // appel au constructeur par défaut
   }
public Maclasse(Date d) {
   this() ;          // appel au constructeur sans paramètres
   this.d = d;  /* this.d désigne l'attribut d de cet objet
                   et d désigne le paramètre ici */
    }
```

Exemple 2

```
public void traitement1(Date d)  {
    // ...
}
public Date traitement2( ) {    // ...
     // passage en paramètre de l'objet courant
     traitement1( this );
          //              ...
     return this;          // retourne l'objet courant
 }  // ...
} // fin Maclasse
```

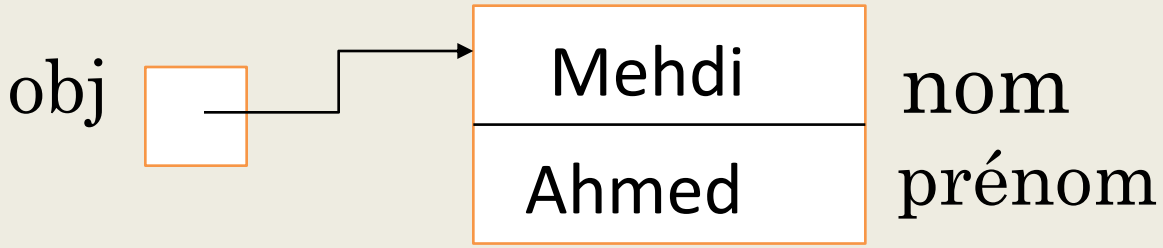
Variable d'instance

Une variable déclarée dans le corps de la classe et appelé **variable d'instance** car elle appartient à l'instance de classe.

Variable d'instance

Exemple:
```
class Personne {
    private  String nom, prénom;
    …
}
Personne obj=new Personne("Mehdi ", "Ahmed");
```

obj est une instance de classe



nom et prénom sont des variables d'instance.

Variable d'instance

L'accès à une variable d'instance:

Avec les enregistrements pour accéder à un champ on écrit:

Nom de la variable . nom du champ

En java on écrit:

Nom de l'objet . nom de l'attribut

Méthode d'instance

Une méthode d'instance est liée à un objet particulier ainsi elle est nécessairement appelée à travers un objet.

Nom de l'objet . nom de la méthode

Pour qu'un tel appel soit possible, il faut que trois conditions soient remplies :

1. L'objet visé doit être instanciée.
2. La variable ou la méthode appelée doit exister.
3. L'objet, au sein duquel est fait cet appel, ait le droit d'accéder à la méthode ou à la variable (accessibilité ou encapsulation)

Exemple

Cas 1

```
Date d1;
System.out.print(d1.jour);
```

Cas 2

```
Date  d2=new Date(12,4,2011);
D2.Affichage();
```

Cas 3

```
Date  d2=new Date(12,4,2011);
d2.jour=14;
```

Cas4

```
Date  d2=new Date(12,4,2011);
D2.printDate();
```

Variable de classe et mot clé static

- Une variable de classe appartient à la classe et n' elle n'est pas lié à l'objet.
- Elle ne sont définies qu'une seule fois quel que soit le nombre d'objets instanciés de la classe.
- Leur déclaration est accompagnée du mot clé **static**.
- Equivalente à une variable globale

Méthode de classe

- Une méthode qui peut être appelée même sans avoir instancié la classe.
- Ne peut accéder qu'à des attributs et méthodes statiques.
- **Une méthode de classe appartient à la classe et elle n'est pas lié à l'objet.**

Exemple

```
class Personne {
  private String nom, prénom;      // 2 attributs d'instance
  public static int nbPersonne=0;  // variable de classe
  public Personne(String nom, String prénom) {    // constructeur
       this.nom=nom; this.prénom=prénom;   nbPersonne++;    }
  public static int nbPersonne( ) { return nbPersonne;  // méthode de classe  }
  public void printPersonne() {  // méthode d'instance
     System.out.println(" nom " +nom+ " \n " + " prénom " +prénom);
  }
public static void printPersonne(Personne p) {  // méthode de classe
     System.out.println(" nom " +p.nom+ " \n " + " prénom " +p.prénom);
  }
  // ...       } // end class Personne
```

Invocation d'un attribut ou d'une méthode de classe

Nom de la classe . Nom de l'attribut

Nom de la classe . Nom de la méthode

Exemple:

```
System.out.println(Personne.nbPersonne);  // attribut static

int nb=Personne.nbPersonne();  // méthode static  nbPersonne()
```

Mot clé final

•Est attribué a une variable qui ne peut plus changer après l'initialisation

Exemple

Final PI=3.14;

Remarque: static et final sont différentes !!

Accesseurs et modifieurs (getters et setters)

- Des méthodes permettant d'accéder ou de modifier des attributs
- Très utiles lorsque les attributs sont privés
- Les méthodes permettant d'accéder aux attributs sont appelées **accesseurs**, parfois *getters*
- Les méthodes permettant de modifier les attributs sont appelées **modifieurs**, parfois *setters*

Exemple

```
class Personne {
private String nom, prénom;      // 2 attributs d'instance
        …
public String getNom() {    return nom;   }  // getter
public String getPrénom() {    return prénom;   } // getter
public void setNom(String nom) {   this.nom=nom;  }  // setter
public void setPrénom(String prénom) {     // setter
            this.prénom=prénom;
 }
            …
} // end class Personne
```

Le mot clé final

- Le mot clé final s'applique aux variables non modifiables après la déclaration et l'initialisation.
- Ce mot clé désigne donc **une constante**.

Exemple :

```
public static final double PI = 3.14159265358979323846 ;
```

Remarques

- Ce qui **n'est pas statique** est appelé **dynamique ou d'instance**.
- Dans une méthode statique, l'emploi de **this** n'est pas autorisé.
- Une méthode statique ne peut pas appeler une méthode dynamique.
- À l'inverse, une méthode dynamique peut appeler une méthode statique.
- Le point d'entrée d'un programme Java, à savoir sa méthode **main**, est une méthode statique :

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

Les packages

- Un grand nombre de classes, fournies par *SUN*, implémentent des données et des traitements génériques utilisables par un grand nombre d'applications.
- Ces classes forment l'API (*Application Programmer Interface)* du langage Java.
- Toutes ces classes sont organisées en ***packages*** (ou bibliothèques) dédiés à un thème précis.

Quelques bibliothèques

- **java.lang** contient les classes de base (chaînes de caractères, mathématiques, tâches, exceptions ...)
- **java.util** contient des classes comme vecteurs, piles, files, tables ...
- **java.io** contient les classes liées aux entrées/sorties : texte et fichier
- **java.awt** contient les classes pour les interfaces (fenêtres, boutons, menus, graphique, événements ...)
- **javax.swing** contient les classes pour les interfaces (évolution de awt)
- **java.net** contient les classes relatives à internet (sockets, URL ...)
- **java.applet** contient les classes permettant de créer des applications contenues dans des pages en HTML

Mot clé import

- Pour pouvoir utiliser les classes des bibliothèques il faut y faire référence en début de fichier en utilisant la directive **import** suivie du nom de la classe de la bibliothèque ou de * pour accéder à toutes les classes de cette bibliothèque.

  Exemple: **import java.util.Scanner** ou **import java.util.***

- Certaines de ces bibliothèques contiennent des sous-bibliothèques qu'il faut explicitement nommer.

  Exemple: **java.awt.events.*** pour les événements