

**Programmation Orientée Objet**  
**Et langage JAVA**

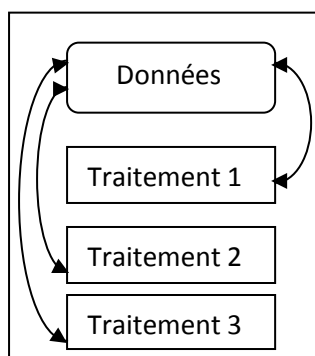
## 1. Introduction

Le programmeur par objets se pose en premier la question :

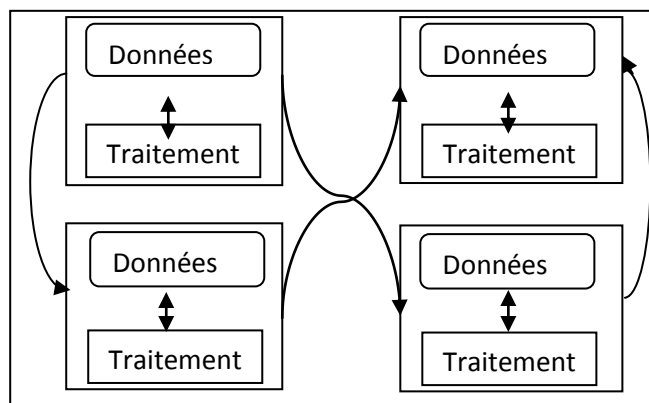
- ✓ quels sont les objets fondamentaux de l'application ?
- ✓ Quelles sont les choses et les concepts qui constituent le domaine de l'application ?

Il décrit alors chaque type d'objet sous forme de variables qui caractérisent son état puis il s'intéresse aux opérations que peuvent subir ces objets et les décrit sous forme de procédures. Ceci conduit à une **modularité** basée sur les types de données. Les traitements sont toujours définis à propos de la définition d'un type de données, et non pas isolément comme c'est le cas pour une modularité basée en priorité sur la décomposition des traitements.

### Langages procéduraux



### Langages orientés objet



En résumé, si la programmation structurée s'intéresse aux traitements puis aux données, la conception objet s'intéresse d'abord aux données, auxquelles elle associe ensuite les traitements. L'expérience a montré que les données sont ce qu'il y a de plus stable dans la vie d'un programme, il est donc intéressant d'architecturer le programme autour de ces données. Le concept objet permet d'obtenir des logiciels fiables, évolutifs et faciles à maintenir.

## 2. Concepts de base de la programmation orientée objet

### 2.1 Classes et Objets

#### Classe :

Une classe est la généralisation de la notion de type défini par l'utilisateur (les structures les unions et les énumérations), dans laquelle se trouvent associées à la fois des **données** et des **méthodes**. Une classe regroupe donc un ensemble de données (qui peuvent être des variables primitives ou des objets) et un ensemble de méthodes de traitement de ces données et/ou de données extérieures à la classe.

Les données se présentent sous forme de champs désignés par des identificateurs et dotés d'un type. Ces champs sont généralement des variables qui représentent l'état de l'objet.

Les procédures, également appelées **méthodes**, définissent les opérations possibles sur un tel objet. Ces données et procédures sont qualifiées de **membres** ou **composants** de la classe.

#### Objet :

Comme une *classe* (ou une structure) n'est qu'un simple type, les *objets* possèdent les mêmes caractéristiques que les variables ordinaires.

Un *objet* est donc une variable (presque) comme les autres. Il faut notamment qu'il soit déclaré avec son type. Le type d'un objet est un type complexe (par opposition aux types primitifs entier, caractère, . . .) qu'on appelle une *classe*.

Une *classe* est la définition d'un type, alors que l'*objet* est une déclaration de variable.

Après avoir créé une classe, on peut créer autant d'objets que l'on veut basés sur cette classe.

### 2.2 Etapes d'une conception orientée objet


L'approche objet offre une manière claire pour concevoir une architecture de modules autonomes pour une implémentation multiplateforme. Elle se déroule en trois étapes:

#### L'analyse:

Cette étape consiste à étudier les attentes des utilisateurs pour identifier les objets du monde réel qu'ils invoquent dans leur activité.

**Exemple :** Si on veut écrire un logiciel de « Gestion d'une bibliothèque », on se pose la question : de quoi est composé ce problème ?

On identifie donc à cette étape tous les objets qui interviennent dans une bibliothèque à savoir:

- Des livres
  - Des journaux
  - Une directrice ou un directeur
  - Un bibliothécaire
  - Des lecteurs
- 
- Les objets du problème  
(ensembles de données)

L'étape suivante est de regrouper les objets similaires dans une même **classe** :

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

Nous constatons que ces données peuvent être regroupées en 4 classes :

- Classe **livre** identifiée par : titre du livre et le nom de l'auteur;
- Classe **journal** identifiée par : titre du journal;
- Classe **employé** identifiée par : nom, prénom de l'employé et statut de l'employé ;
- Classe **lecteur** identifiée par : nom et prénom du lecteur.

### **La conception:**

Cette étape consiste à concevoir l'organisation des classes et le choix des structures de données pour le stockage.

### **L'implémentation:**

C'est l'étape de la réalisation pratique du logiciel, l'écriture des programmes, le débogage, les tests et la validation.

## **2.3 Création d'une classe**

Une classe peut comporter :

- ✓ Un ensemble de déclarations d'identificateurs : ***les attributs***, définissant une structure,
- ✓ un ensemble de définitions de fonctions : ***les méthodes***, définissant un comportement.
- ✓ des ***constructeurs***, qui permettent de créer des objets ;

### 2.3.1 Syntaxe

La déclaration d'une classe se fait de la façon suivante :

```
class NomClasse
{
    corps de la classe
}
```

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

### 2.3.2 Attributs :

Les attributs représentent la description des données propres à chaque classe d'objets. Ceux-ci peuvent être des objets d'autres classes ou des références sur d'autres objets.

### 2.3.3 Méthodes :

Les méthodes représentent l'ensemble des actions, procédures, fonctions ou opérations que l'on peut associer à une classe.

#### Écriture des méthodes :

La **signature** d'une méthode contient notamment :

- un nom de méthode;
- un type de données de retour;
- des types de données pour ses arguments;
- ...

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

Une méthode est composée de sa signature et d'un traitement associé.

```
<type de retour> <nom de méthode> (<arguments>)
{
    <traitement associé>
}
```

### 2.3.4 Encapsulation :

On parle d'*encapsulation* pour désigner le regroupement de données dans une classe. L'encapsulation de données dans un objet permet de cacher ou non leur existence aux autres objets du programme. Une donnée peut être déclarée en accès :

- **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).

### 2.3.5 Surcharge des méthodes

Dans Java, il est possible de créer dans une classe plusieurs méthodes qui portent le même nom mais avec différents paramètres et/ou valeurs de retour. Cela est connu sous le nom de *surcharge des méthodes*. Java décide de la méthode à appeler en regardant la valeur de retour et les paramètres.

## 2.4 Création d'un objet

### 2.4.1 Constructeurs

Un **constructeur** est une méthode particulière invoquée lors de la création d'un objet. Cette méthode, qui peut être vide, doit donner une valeur initiale à tous les champs de son type objet.

Chaque classe doit définir un ou plusieurs **constructeurs**.

Chaque 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).

#### Exemple :

```
class Date {  
    private int jour, mois, an;    // 3 attributs  
    final int max_mois = 12;    // un attribut constant  
  
    public Date() {                // définition du constructeur  
        jour = mois = an = 0;    // par défaut  
    }  
  
    public Date(int j, int m, int a) {    // constructeur surchargé  
        jour = j; mois = m; an = a;  
    }  
  
    public void printDate() {  
        // ...  
    }  
  
    // ...  
} // end class date
```

### 2.4.2 Instanciation

L'instanciation est l'opération qui consiste à créer un objet à partir d'une classe. Une définition de classe constitue un modèle d'objet dont on peut ensuite créer autant d'exemplaires que l'on veut (on dit également *instances*).

En Java, la *déclaration* et la *création* des objets sont deux choses séparées. La déclaration consiste à définir un identificateur et lui associer un type afin que cet identificateur puisse désigner un objet de ce type.

En Java, un identificateur de variable de type classe est associé à une *référence* qui désigne un objet. Une référence est *l'adresse* d'un objet, mais contrairement à un pointeur (tel qu'on en trouve en C, C++ ou Pascal) la valeur d'une référence n'est ni accessible, ni manipulable : elle ne peut que permettre l'accès à l'objet qu'elle désigne.

La déclaration seule ne crée pas d'objet. Elle associe l'identificateur à une référence appelée **null** qui ne fait référence à rien.

Pour créer un objet de classe **C**, il faut exécuter **new** qui provoque une instanciation en faisant appel à un **constructeur** de la classe instanciée.

**Exemple :** `Date d1=new Date() ; // constructeur sans paramètres`

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

**Remarque importante :** en Java, la notion de pointeur est transparente pour le programmeur. Il faut néanmoins savoir que *toute variable désignant un objet est un pointeur*. Il s'ensuit alors que 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.

### 2.4.3 Garbage collector « ramasse miettes »

La libération de l'espace mémoire créée dynamiquement est automatique en java. Dès que la donnée n'est plus référencée par un pointeur. C'est le mécanisme du garbage collector, aussi appelé « ramasse miettes », qui s'en charge. Autrement dit, la destruction des objets est prise en charge par le "garbage collector". Ce dernier détruit les objets qui n'ont plus de référence

## 2.5 Invocation de méthodes :

En Java, une méthode ne peut pas être invoquée seule, elle est toujours appelée sur un objet.

Un point `.` sépare le nom de la méthode de l'objet sur lequel elle est invoquée.

(La syntaxe pour accéder aux attributs d'un objet est la même).

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

1. La variable ou la méthode appelée existe.
2. Une variable désignant l'objet visé existe et soit instanciée.
3. L'objet, au sein duquel est fait cet appel, ait le droit d'accéder à la méthode ou à la variable

### Exemple :

```
...
Date d1 ; // pas d'instanciation
Date d2=new Date(12,4,2011); // variable instanciée
d2.printDate(); // appel à la méthode printDate() (méthode public)
d2.jour=14; // modification de l'attribut jour s'il n'est pas déclaré en private
...
```

Rappel : d1 et d2 contiennent des références sur l'objet créé donc il est possible d'écrire **d1=null** ; // d1 ne pointe sur aucun objet et vous constaterez que le mot clé null en JAVA est en minuscule contrairement au langage C.

## 2.6 Le mot-clé **this** :

Ce mot-clé est une référence sur l'objet en cours, c'est à dire la classe dans laquelle on se trouve

Il désigne, en cours d'exécution d'une méthode, l'objet sur lequel elle est appelée.

Aussi, l'appel à **this** peut être utile pour bien différencier les variables de classe des variables de méthodes.

On peut aussi accéder aux constructeurs de la classe elle-même

### Exemple :

```
public class Maclasse {
    private Date date;

    public Maclasse() {
        date=new Date() ; // appel au constructeur par défaut
    }
}
```



```
void traitement1(Date date) {  
  
    this() ;           // appel au constructeur de la classe  
  
    this.date = date; /* this.date désigne l'attribut date de cet objet  
                       et date désigne le paramètre ici this permet  
                       de lever l'ambiguïté */  
}  
Date traitement2() {  
    // ...  
    traitement1( this ); // passage en paramètre de l'objet courant  
  
    // ...  
    return this;        // retourne l'objet courant  
}  
// ...  
} // fin Maclasse
```

## 2.7 Les accesseurs (set et get) :

L'encapsulation permet de sécuriser l'accès aux données d'une classe. Ainsi, les données déclarées « **private** » à l'intérieur d'une classe ne peuvent être accédées et modifiées que par des méthodes définies dans la même classe.

Si une autre classe veut accéder aux données de la classe, l'opération n'est possible que par l'intermédiaire d'une méthode de la classe prévue à cet effet. Ces appels de méthodes sont appelés « **échanges de message** ».

Un accesseur est une méthode public qui donne l'accès à une variable d'instance privé.

Pour une variable d'instance :

- il peut ne pas y avoir d'accesseur,
- un seul accesseur en lecture,
- un accesseur en lecture et un accesseur en écriture.

*Remarque :* Par convention, les accesseurs en lecture commencent par « get » et les accesseurs en écriture commencent par « set ».

### Exemple :

```
import java.lang.Math; // importer la classe Math
public class Point {
private double x,y;
    public Point () {
        x=y=0;
    }
    public Point (double x, double y) {
        this.x=x;this.y=y;
    }
    public double getx(){
        return x;
    }
    public double gety() {
        return y;
    }
    public void setx(double x){
        this.x=x;
    }
    public void sety(double y) {
        this.y=y ;
    }
    public void afficher() {
        System.out.println("x= " + x + " y= " + y);
    }
}
```

```
import java.util.Scanner; /* importer la classe Scanner contenant les méthodes
                           de lecture des données */
public class Ppoint {    // classe exécutable contenant la méthode main

    public static void main(String[] args) {
        Scanner Entree=new Scanner(System.in); /* créer un objet de type
            Scanner pour pouvoir invoquer les méthodes de lecture */

        System.out.println("Donnez les coordonnées d'un points");
        double ax=Entree.nextDouble();
        double ay=Entree.nextDouble();
        Point p1=new Point(ax,ay); // instantiation de l'objet p1

        p1.afficher();    p1.setx(2);    p1.afficher();
    }
}
```

## 2.7 Composants non liés à des objets

On a parfois besoin de méthodes non associées à un objet. Une telle méthode doit être rédigée dans une classe, car il n'y a pas de méthodes isolées en Java. Cela est souvent naturel, car la classe joue alors un rôle structurant en regroupant les méthodes autour d'un même thème. Pour cela, il faut déclarer la méthode avec l'attribut **static**.

**static** : indique que la variable est en fait globale à toutes les instances de la classe ou des ses sous-classes<sup>1</sup>. Elle est directement accessible par la classe.

**final** : indique que la valeur de la variable ne peut pas être changée (représente en fait une constante).

Les constantes sont généralement déclarées comme **public static final**.

**Exemples:**

- **final**

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

- **Méthodes non static**

```
import java.lang.Math;  
public class Point {  
    private double x,y;  
    // ...  
    public Point milieu( Point p){  
        return new Point((x+p.x)/2,(y+p.y)/2);  
    }  
    public double distance (Point p) {  
        return Math.sqrt(Math.pow((x-p.x),2.0)+Math.pow(y-p.y),2.0));  
    }  
    public void deplace (double dx,double dy){  
        setx(x+dx);  
        sety(y+dy); }  
} // Fin Point
```

```
import java.util.Scanner;  
public class Ppoint {
```

---

<sup>1</sup> Sous-classe : sera définie dans le chapitre Héritage.

```
public static void main(String[] args) {
    Scanner Entree=new Scanner(System.in);
    System.out.println("Donnez 2 points");
    Point p1=new Point(Entree.nextDouble(), Entree.nextDouble());
    Point p2=new Point(Entree.nextDouble(), Entree.nextDouble());

    Point m=p1.milieu(p2); // nom de l'objet • nom de la méthode
    m.afficher();
    System.out.println("distance "+ p1.distance(p2));
}
} // Fin Ppoint
```

- **Méthodes static**

```
import java.lang.Math;
public class Point {
private double x,y;

// ...

public static Point milieu( Point p1,Point p2){
    return new Point((p1.x+p2.x)/2,(p1.y+p2.y)/2);
}
public static double distance (Point p1,Point p2) {
    return Math.sqrt(Math.pow((p2.x-p1.x),2.0)+Math.pow((p2.y-p1.y),2.0));
}
public void deplace (double dx,double dy){
    setx(x+dx);
    sety(y+dy); }
} // Fin Point
```

```
import java.util.Scanner;
public class Ppoint { // classe exécutable

    public static void main(String[] args) {
        Scanner Entree=new Scanner(System.in);
        System.out.println("Donnez 2 points");
        double ax=Entree.nextDouble();
        double ay=Entree.nextDouble();
        double bx=Entree.nextDouble();
        double by=Entree.nextDouble();

        //-----
```

```
        Point p1=new Point(ax,ay);
        Point p2=new Point(bx,by);
```

```
Point m=new Point(0,0);
m=Point.milieu(p1,p2); // nom de la classe • nom de la méthode
m.afficher(); // nom de l'objet . nom de la méthode
p1.deplace(1,1);
p1.afficher();
} // fin main
} fin Ppoint
```

- **Attribut static**

```
import java.util.Scanner;
public class PointTab {
    public static int k;    // Une variable static
    private int n;
    private Point[] tab;    // Un tableau de points
    public static final int max=100;    // Une constante static

    public PointTab(){
        tab=new Point[max];
        n=0;
    }

    // ...

} // fin PointTab
```