Programmation Orientée Objet

avec Java

Par Dr F.Guerroudji

Année 2020/2021

Chapitre 1

Bases de la Programmation Orientée Objet

Introduction: Paradigme

Un paradigme est une représentation du monde, une manière de voir les choses, un modèle cohérent de vision du monde qui repose sur une base définie

Paradigme = modèle de pensée, façon de voir les choses.

Exemple:

Paradigme en Entreprise = Aspect fortement novateur à un projet.

Introduction: Paradigme

- **Paradigme en informatique est** la façon dont un système a été conçu et pensé dans ses grandes lignes.

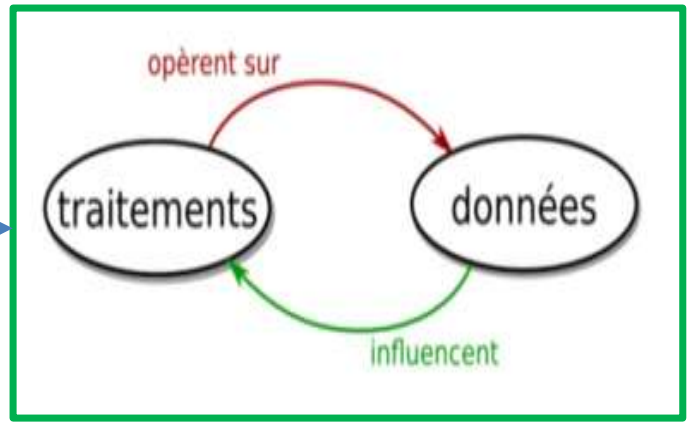
Introduction: Paradigme de programmation

- Un paradigme de programmation est une façon d'approcher la programmation informatique et de traiter les solutions aux problèmes et leur formulation dans un langage de programmation approprié

- Un paradigme de programmation est une manière de définir ce qu'est un programme et une exécution d'un programme

Programmation Procédurale VS Programmation Orientée-Objet (POO)

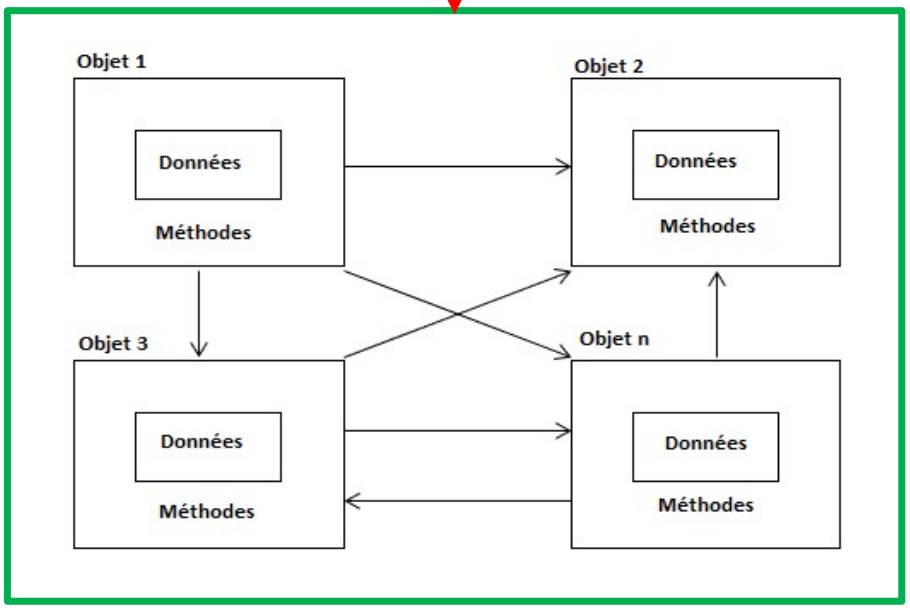
- Centrée sur les procédures (opérations)
  - ✓ Décomposition des fonctionnalités d'un programme en procédures qui vont s'exécuter séquentiellement
- Couplage procédures/données
  - ✓ Les données sont indépendantes procédures
  - ✓ Les données à traiter sont passées en arguments aux procédures
  - ✓ les données peuvent être facilement accessibles et modifiables,
- Ressemble peu à notre schème de penser
  - ✓ Vue algorithmique d'un programme donc très près du langage de la machine
  - ✓ Le programmeur doit faire un effort cognitif pour interpréter ce qu'il veut modéliser pour être "compris" par la machine
  - ✓ Peu intuitive lors de l'analyse d'un code existant
- Tend à générer du code:
  - ✓ dont la maintenance et l'ajout de nouvelles fonctionnalités demandent de modifier ou d'insérer des séquences dans ce qui existe déjà
  - ✓ Peu devenir complexe très rapidement



La programmation procédurale place l'action et la logique au centre,

Programmation Procédurale VS Programmation Orientée-Objet (POO)

- Toute entité identifiable, concrète ou abstraite, peut être considérée comme un objet
- Un objet réagit à certains messages qu'on lui envoie de l'extérieur.
  - ✓La façon avec laquelle il réagit détermine le comportement de l'objet.
  - ✓ Il ne réagit pas toujours de la même façon à un même événement, sa réaction dépend de l'état dans lequel il se trouve.
  - ✓Les données de l'objet ne sont pas facilement accessibles et modifiables.



La Programmation orientée objet rassemble des informations en entités uniques appelées objets.

Programmation Procédurale VS Programmation Orientée-Objet (POO)

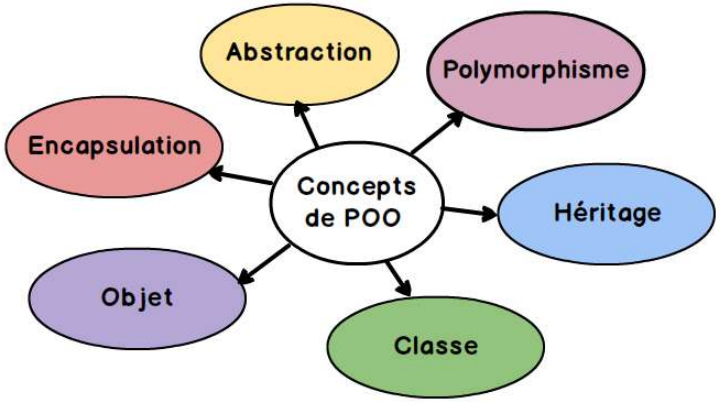
	Programmation Procédurale	Programmation Orientée Objet
Programmes	Le programme principal est divisé en petites parties selon les fonctions.	Le programme principal est divisé en petit objet en fonction du problème.
Les données	Chaque fonction contient des données différentes.	Les données et les fonctions de chaque objet individuel agissent comme une seule unité.
Permission	Pour ajouter de nouvelles données au programme, l'utilisateur doit s'assurer que la fonction le permet.	Le passage de message garantit l'autorisation d'accéder au membre d'un objet à partir d'un autre objet.
Exemples	Pascal, Fortran	PHP5, C ++, Java.
Accès	Aucun spécificateur d'accès n'est utilisé.	Les spécificateurs d'accès public, private, et protected sont utilisés.
La communication	Les fonctions communiquent avec d'autres fonctions en gardant les règles habituelles.	Un objet communique entre eux via des messages.
Contrôle des données	La plupart des fonctions utilisent des données globales.	Chaque objet contrôle ses propres données.
Importance	Les fonctions ou les algorithmes ont plus d'importance que les données dans le programme.	Les données prennent plus d'importance que les fonctions du programme.
Masquage des données	Il n'y a pas de moyen idéal pour masquer les données.	Le masquage des données est possible, ce qui empêche l'accès illégal de la fonction depuis l'extérieur.

Programmation Procédurale VS Programmation Orientée-Objet (POO)

- **Attention à la fausse POO, où les instances ne font que simuler une programmation procédurale**

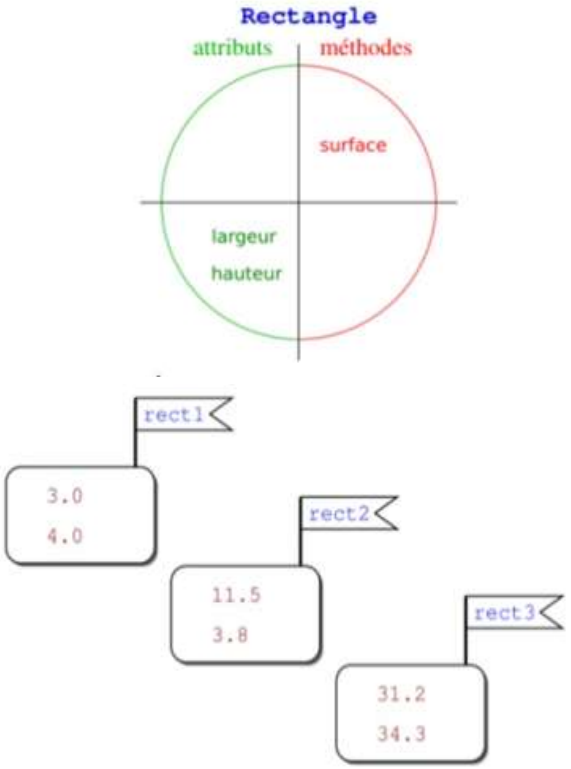
POO: Concepts de base

- Chaque **objet** est une instance unique d'une **classe**.
- L'**abstraction** cache le fonctionnement interne d'un objet lorsqu'il n'est pas nécessaire de le voir.
- L'**encapsulation** stocke les variables et méthodes associées dans des objets et les protège.
- L'**héritage** permet aux sous-classes d'utiliser les attributs des classes parentes.
- Le **polymorphisme** permet aux objets et aux méthodes de gérer plusieurs situations différentes avec une seule interface.



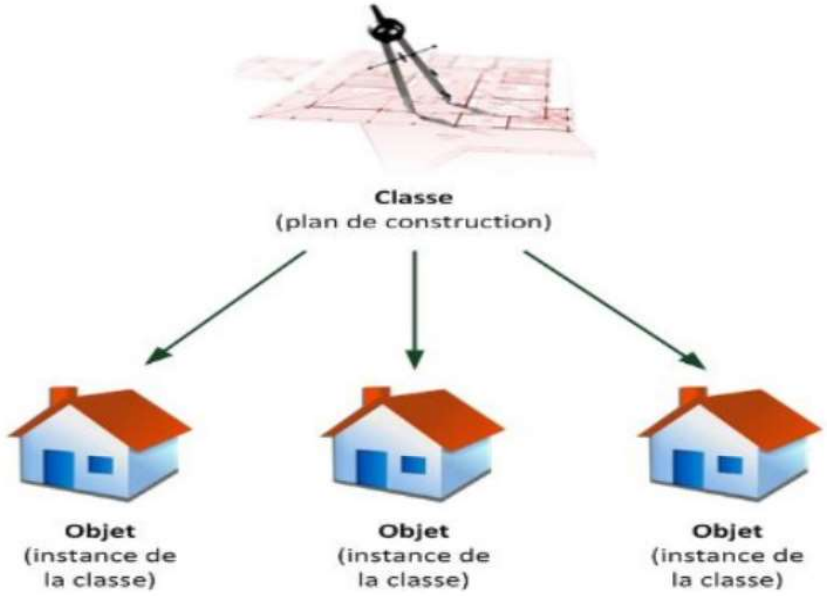
POO: CLASSE ET OBJET

- Une classe définit des **champs** et des **méthodes**
  - ✓ Les *champs* décrivent l'*état*
  - ✓ Les *méthodes* définissent les *interactions*
- Une classe peut être instanciée
  - ✓ L'instance est une concrétisation en mémoire d'un modèle défini par la classe



Remarque: La modélisation par classes permet une abstraction du domaine à modéliser

POO: Abstraction



Les classes sont une "abstraction". Les objets peuvent correspondre à des objets du monde réel, les classes correspondent à **une entité abstraite** de niveau supérieur, elle est une **description générique** de l'ensemble considéré.

Une classe est une entité indépendante (module) qui compose le programme.

Le processus d'abstraction consiste à identifier pour un ensemble d'éléments :

- des caractéristiques communes à tous les éléments (attributs)
- des mécanismes (des comportements) communs à tous les éléments (méthodes)

Exemple : Rectangles

- la notion d'« objet rectangle » n'est intéressante que si l'on peut lui associer des propriétés et/ou mécanismes généraux (valables pour l'ensemble des rectangles)
- Les notions de largeur et hauteur sont des propriétés générales des rectangles (attributs),
- Le mécanisme permettant de calculer la surface d'un rectangle (surface = largeur × hauteur) est commun à tous les rectangles (méthodes)

POO: Encapsulation

- **Un objet est constitué d'attributs qui caractérisent son état.**

- **ENCAPSULATION**: les attributs d'un objet ne pourront être modifiés ou lus que par les méthodes de l'objet.

- **ENCAPSULATION:** masquer les détails d'implémentation d'un objet, en définissant une interface. L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.

- **ENCAPSULATION:** facilite l'évolution d'une application car elle stabilise l'utilisation des objets : nous pouvons ainsi modifier l'implémentation des attributs d'un objet sans modifier son interface, et donc la façon dont l'objet est utilisé.

- **ENCAPSULATION** garantit l'intégrité des données, car elle permet d'interdire, ou de restreindre, l'accès direct aux attributs des objets.

Remarque: L'état d'une instance est principalement manipulable par cette instance, évitant ainsi les risques d'erreur

POO: Encapsulation

- Un objet est donc constitué :
- ✓ D'une partie privée qui ne peut pas être directement accessible de l'extérieur de l'objet.

Remarque :La partie privée contient généralement les attributs de l'objet.

- ✓ D'une partie publique qui est accessible de l'extérieur de l'objet.

Remarque: La partie publique contient généralement les méthodes de l'objet (méthodes qui par contre permettent d'accéder aux attributs).

Encapsulation: Intérêts

- ✓ **Centraliser les contrôles** sur l'état de l'objet
- ✓ **Imposer des règles et limitations** de visualisation ou de manipulation de l'objet

POO: Héritage

- On peut créer des classes dérivant chacune d'une classe mère (héritage simple) ou de plusieurs classes mères (héritage multiple)

Remarque: Les membres et méthodes de la classe mère sont accessibles sur la classe fille si encapsulation public ou protected

POO: Polymorphisme

- **Une instance** peut se présenter sous la forme de **plusieurs classes**