I. Introduction

Un **paradigme** est une représentation du monde, une manière de voir les choses. Le mot paradigme tient son origine des mots grecs paradeigma qui signifie « **modèle** ».

Un **paradigme de programmation** est un style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation. Il détermine la vue qu'a le développeur de l'exécution de son programme. Plusieurs paradigmes de programmation existent : Impératif, Procédural, Fonctionnel, Logique et Objet.

II. Petit Historique de la POO

1. La programmation OO a été introduite par le langage Simula à l'Université d'Oslo en 1967.
   - Le problème posé était celui de la simulation d'un ensemble de robots dans une entreprise.
   - Essayer de résoudre ce problème à l'aide de la programmation impérative est un vrai casse-tête tant les interactions sont nombreuses et imprévisibles.
   - Tentez d'imaginer un programme centralisé qui pilote les robots en modifiant une structure de données représentant l'état des robots et de l'environnement. Ce n'est pas impossible mais très difficile.
   - La solution: **l'Orienté Objet** en regroupant les données et les procédures.
   - Depuis Simula I (1972) formalise le concept d'objet. Un programme devient une collection d'objets actifs et autonomes.

2. Toutefois, La POO a debuté réellement par et avec Smalltalk 72 puis Smalltalk 80, inspirés en partie par Simula. Ces languages ont posés les concepts de base de celle-ci : objet, messages, encapsulation, polymorphisme, heritage,…etc.

3. Les années 1990 voient l'âge d'or de l'extension de la Programmation Orientée Objet dans les différents secteurs du développement logiciel. Java (1994): d'abord pour le web puis généraliste.

4. Actuellement, d'autres langages ont vu le jour : C#, Object pascal, Delphi, Python, etc...

III. Programmation procédurale VS Programmation Orienté Objet

- Dans la **programmation procédurale** on apprend à subdiviser un problème en une série d'étapes simples (des fonctions) permettant de résoudre un problème. D'abord, on décide de la manière dont on va manipuler les données, ensuite le type de structures de données les plus appropriées pour faciliter cette manipulation.

Programme = algorithme + structure de données

Inconvénients:

- Difficulté de la mise en place de contrôles destinés à garantir la cohérence et l'intégrité des données.
- Adaptabilité limité et coût des erreurs important.
- Donne des logiciels difficiles à maintenir et à réutiliser

Afin de pallier ces inconvénients, la Programmation Orientée Objet (POO) a été introduite.

- Dans la POO, un programme informatique est considéré comme étant un ensemble d'objets fonctionnant ensemble pour effectuer une tâche. On s'intéresse d'abord aux données, avant de déterminer l'algorithme qui sera utilisé pour opérer sur ces données. Les objets coopèrent par envois de messages (Appel de méthodes)

Programme = Un ensemble de composants (les objets) indépendants dont la collaboration dynamique fonde les fonctionnalités du système.

Remarques :

- La POO propose donc une vue différente : un programme devient un ensemble de petites entités informatiques qui interagissent et communiquent par messages. Ceci met en avant l'autonomie de chacune de ces entités informatiques que l'on appelle des **objets**, en considérant que la complexité croissante des programmes ne peut être réalisée que par des traitements locaux. La gestion de la globalité devient alors un effet secondaire des interactions locales.

- En revanche, la POO requiert de la part du programmeur un changement d'état d'esprit, ceci afin qu'il puisse penser en des termes issus de ce Paradigme.

IV. Principes de la POO

Les principes sous-jacents de la POO sont très simples et peuvent être appréhendés rapidement :

a) **Modularité** : les objets forment des modules compacts regroupant des données et un ensemble d'opérations. La cohésion et le couplage visent à guider la façon de réaliser la modularité afin d'améliorer les caractéristiques (réutilisabilité, etc.) d'un module.

b) **Abstraction** : les entités objets de la POO sont proches de celles du monde réel. Les concepts utilisés sont donc proches des abstractions familières que nous exploitons.

c) **Encapsulation**: vise à regrouper les données et les opérations pouvant y accéder. L'interface (l'ensemble des opérations) contrôle les accès aux données. Ceci dit, le principe de l'encapsulation est que, vu de l'extérieur, un objet se caractérise uniquement par ses méthodes visibles appelées interface. Les données restent invisibles et inaccessibles.

Ce principe permet de concevoir des programmes (logiciels) plus sûrs, plus lisibles et plus faciles à maintenir, car une modification de la structure des données d'un objet ne se répercute pas sur les objets qui communiquent avec lui (invoquent ses méthodes).

V. Objectifs de la POO

La POO a pour objectifs de produire du code :

- facile à développer, à maintenir, à faire évoluer,
- réutilisable, tout ou en partie, sans avoir besoin de le dupliquer
- générique, et dont les spécialisations sont transparentes

Ce qui permet de :

- Gagner en productivité et abaisser les coûts
- Augmenter la qualité des logiciels et leurs fonctionnalités
- Faciliter la maintenance et l'évolution des applications

Remarques :

- Plus l'application est complexe et plus la POO est intéressante en termes de productivité.
- Le niveau de réutilisabilité est supérieur à la programmation procédurale.
- L'encapsulation et le typage des classes offrent une certaine robustesse aux applications.

VI. Concepts fondamentaux de la POO

La POO s'appuie sur des notions fondamentales :

- les objets et les classes
- les relations entre objets et classes : instanciation, Héritage
- le polymorphisme.

a) Objet :

L'objet est une entité logicielle qui représente la brique de base de la POO. Il se caractérise par :

• Une **identité** (nom ou référence) unique qui le distingue des autres objets. L'identité doit permettre d'identifier sans ambiguïté l'objet.

• Un **état** décrit par des variables appelées attributs ou champs.

• Un **comportement** implémenté à l'aide de fonctions ou procédures appelées méthodes. Une méthode est une opération (traitement) qu'un objet réalise soit de son propre gré soit en réaction à un envoi de message d'un autre objet. La signature d'une méthode représente la précision de son nom, du type de ses arguments et du type de donnée retournée.

Il existe différents types de méthodes :

- Constructeur : crée et initialise l'objet.
- Modifieur ou modificateur : modifie l'état de l'objet
- Accesseur ou observateur : donne des informations sur l'objet
- Destructeur : détruit l'objet.

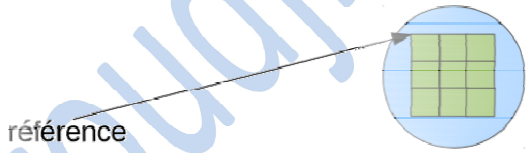
Exemple :

Dans le monde réel, un objet peut être: un étudiant X,

- L'état = Nom, Prénom, Date de naissance, lieu de naissance, …
- Le comportement = suivre des enseignements, passer des examens,…

Remarques :

- Un objet en POO a un **cycle de vie** de 3 phases :
  - Construction (en mémoire).
  - Utilisation (changements d'état par affectations, comportements par exécution de méthodes)
  - Destruction
- un **objet** représente une **abstraction** du domaine d'étude.
- Un **objet** est un composant situé dans une zone mémoire. On le manipule par son adresse appelée **référence**.



b) Notion de message

Un objet seul ne permet pas de concevoir une application garantissant les objectifs de la POO. Un programme est constitué d'objets. Ces derniers communiquent à l'aide de messages. Un message est composé :

- du *nom de l'objet* recevant le message,
- du *nom de la méthode* à exécuter
- et *des paramètres* nécessaires à la méthode.

c) Classe

Le monde réel regroupe des **objets du même type**. Il est pratique de concevoir une maquette (un moule) d'un objet et de produire les objets à partir de cette maquette. En POO, une maquette se nomme une **classe**.

Une **classe** est donc un **modèle** de la structure statique (variables d'instance) et du comportement dynamique (les méthodes) des objets associés à cette classe.

Autrement dit, la classe est une **description d'une famille d'objets** ayant une même structure et un même comportement. Elle est caractérisée par :

- Un **nom** (première lettre Majuscule)
- Une **composante statique** : des champs (ou attributs) nommés ayant une valeur. Ils caractérisent l'état des objets pendant l'exécution du programme
- Une **composante dynamique** : des méthodes représentant le comportement des objets de cette classe.

Exemple :

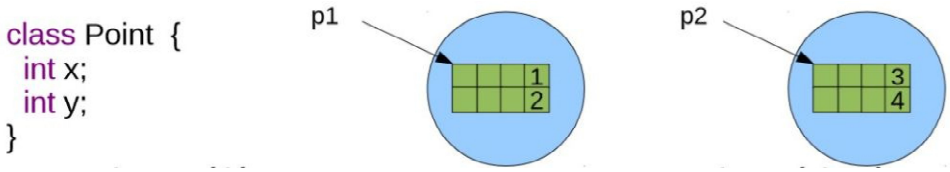
Par exemple, on pourrait créer une classe CompteBancaire dont les objets serviraient à représenter les comptes bancaires des clients d'une banque.

- Les attributs spécifiés pour cette classe seraient par exemple des champs :
  - **Nom** qui contiendrait le nom du possesseur du compte;
  - **Coordonnees** qui contiendrait les coordonnées du possesseur du compte;
  - **Operation** qui contiendrait une liste des opérations sur le compte.

- Les méthodes spécifiées pour cette classe seraient par exemple des fonctions :
  - **Créditer()** qui calculerait les entrées d'argent du compte (à partir de l'attribut operation);
  - **Débiter()** qui calculerait les sorties d'argent du compte (à partir de l'attribut operation);

Remarques :

- Un **type** d'objet est décrit par une **classe**.
  - la classe décrit les **attributs** : nom et type de valeur ;
  - La classe décrit **les méthodes** utilisées pour répondre aux messages.
- Le programmeur peut créer des objets à partir de la classe. C'est le processus d'**instanciation**.
- On dit que les **objets** sont des **instances** de la classe ou que les objets appartiennent à la classe.
- A partir d'une classe tout langage objet permet (par l'intermédiaire de l'opérateur new) de créer des objets qui seront des emplacements en M.C.
- Tous les objets d'une même classe sont organisés de la même manière (même type), mais ils ont un état qui leur est propre.



Les deux références, p1 et p2, permettent d'accéder à deux instances différentes de la même classe Point. Chaque objet a son propre état.

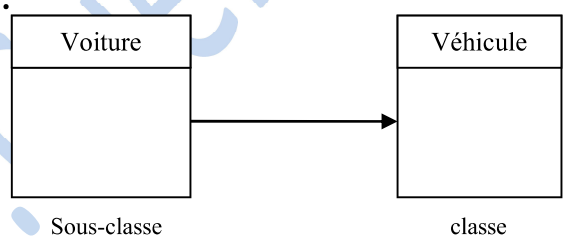
d) La notion de package (paquetage).

- Il existe en POO, une notion modulaire supplémentaire qui est celle de package. Un package est un ensemble de classes qui ont une sémantique commune. Le package permet de regrouper des classes publiques qui ont un lien entre elles
- Dans un package, il y a deux types d'informations :
  - Les informations publiques: ceux sont des classes ou des interfaces qui sont visibles de l'extérieur du package.
  - Les informations locales aux paquetages: ceux des classes ou des interfaces qui ne sont visibles qu'à l'intérieur du package, elles servent en principe à l'implémentation des informations publiques du package.
- Les packages offrent une organisation structurée des classes
- La répartition des packages correspond à l'organisation physique
- Les packages conditionnent les importations de classes
- La variable CLASSPATH indique le chemin d'accès aux packages
- Les packages sont des espaces de noms (permettent la coexistence de classes de même nom)
- Les mots-clé associés sont « package » et « import »

e) Héritage :

- L'héritage permet de définir une nouvelle classe (une sous-classe, classe fille) à partir d'une classe existante (super- classe, classe-mère) à laquelle on ajoute de nouvelles données, de nouvelles méthodes.
- Autrement dit, l'héritage consiste à définir différent niveaux d'abstraction permettant ainsi de factoriser certains attributs et/ou méthodes communs à plusieurs classes. Une classe générale définit alors un ensemble d'attributs et/ou méthodes qui sont partagés par d'autres classes, dont on dira qu'elles héritent de (ou spécialisent) cette classe générale.

Exemple :



Héritage= Généralisation / Spécialisation

Remarques

- L'héritage spécialise des classes anciennes ce qui permet une :
  - économie du code.
  - structuration des fonctionnalités d'un programme (lisibilité).
- L'héritage peut- être réitéré (transitif) : la classe C hérite de la classe B qui elle-même hérite de la classe A. Les classes sont organisées en hiérarchie.
- La relation d'héritage est équivalente à la relation « est un »
- Une instance n'hérite pas de sa classe.
- l'héritage multiple est le cas où une classe hérite de plusieurs super-classes.

f) Le polymorphisme :

Le Polymorphisme :

- Le terme polymorphisme issu du grec signifie la faculté de prendre plusieurs formes.
- C'est un concept puissant de la POO qui vient compléter celui de l'héritage.
- En POO, cela peut s'appliquer aussi bien aux objets qu'aux méthodes.

➢ Polymorphisme d'objets

- Il permet une certaine flexibilité dans la manipulation des objets en exploitant la relation d'héritage entre les classes.

- Il applique la règle suivante : Si la classe B est dérivée de la classe A alors un objet de la classe B peut aussi être considéré comme étant un objet de la classe A.

➢ Polymorphisme de méthodes :

- La **surcharge** est la redéfinition d'une méthode héritée pour pouvoir lui donner une implémentation différente.

- La méthode garde le même sens mais réalise une **spécialisation**. On parle de **méthode polymorphe**.

- D'où, en POO le polymorphisme de méthodes est la faculté d'une même méthode à pouvoir s'appliquer différemment suivant la classe.