Analyste Programmeur en Automatisme, Robotique et Informatique Industrielle

TS ARII

Module MF 2.7

Développer un programme en langage objet

Programmer dans un langage orienté objet - concepts

Patrick MONASSIER

LA MACHE
- Supérieur -

Qualification des Services Intellectuels ISQ OPQF

MF 2.7 – Développer un programme en langage objet

Compétences

Traduire un modèle dans un langage objet

Objectifs

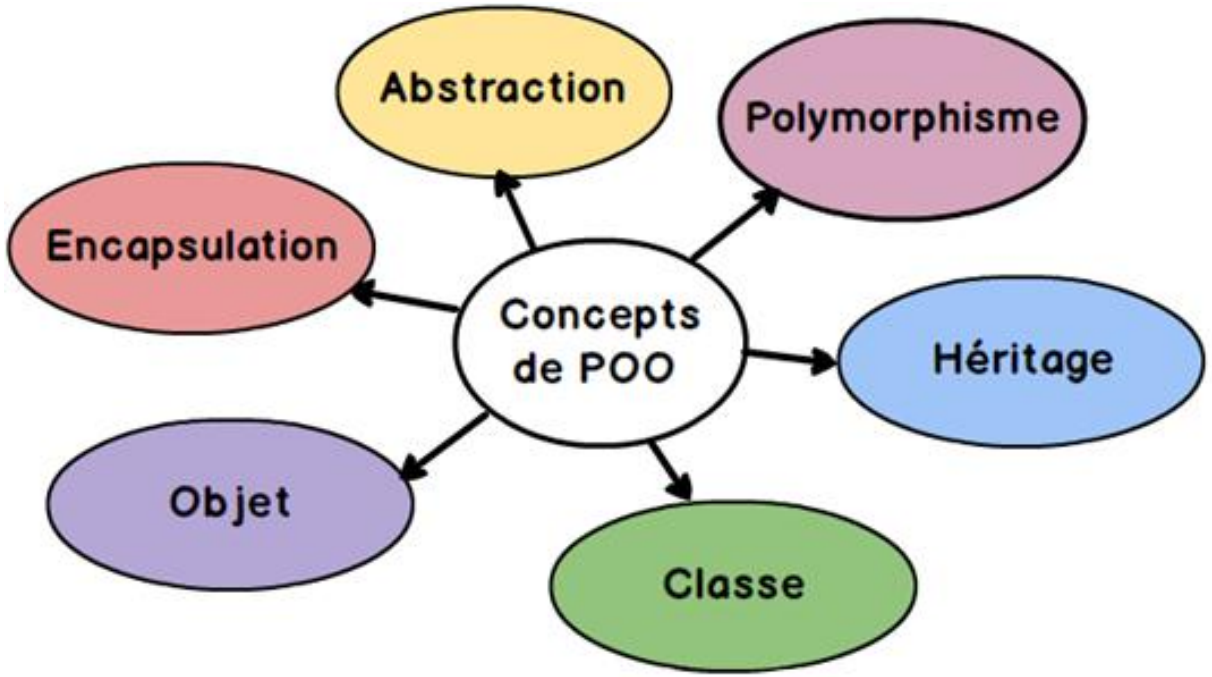
Programmer dans un langage orienté objet

Contenu

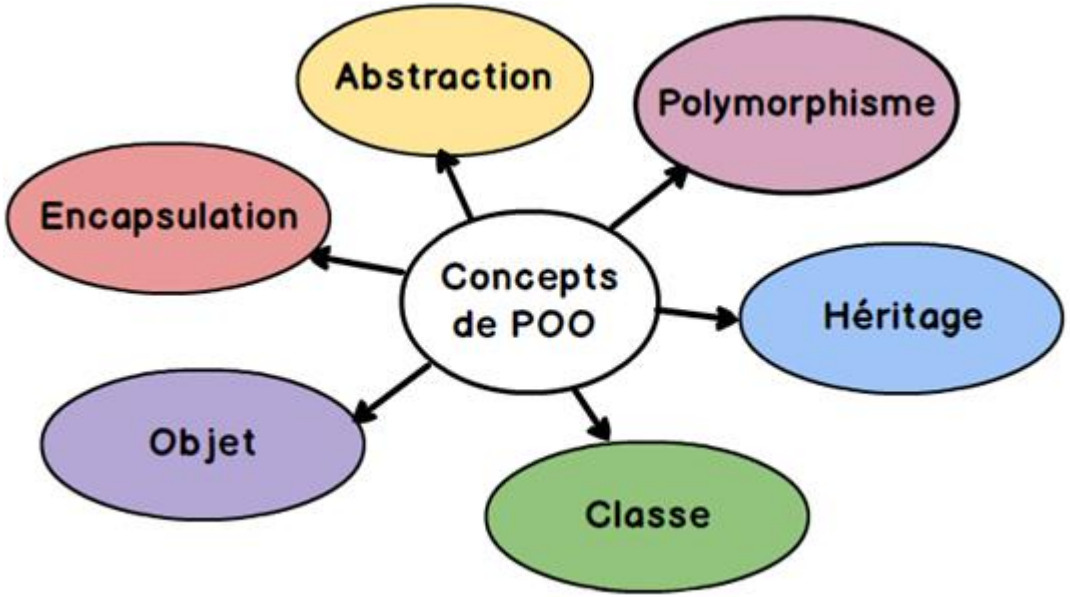
- *Les langage Visual Basic et VB Script*
  - *Transcription d'un algorithme, traduction d'un modèle objet*

- *L'AGL PC Soft WINDEV*
  - *Transcription d'un algorithme, traduction d'un modèle objet*

La Programmation Orientée Objet (ou **POO**) **est** un paradigme de programmation dans lequel les programmes sont écrits et structurés autour des objets plutôt **que** des fonctions ou de logique. Ici, les objets sont définis comme des champs de données qui ont des attributs et un comportement uniques.



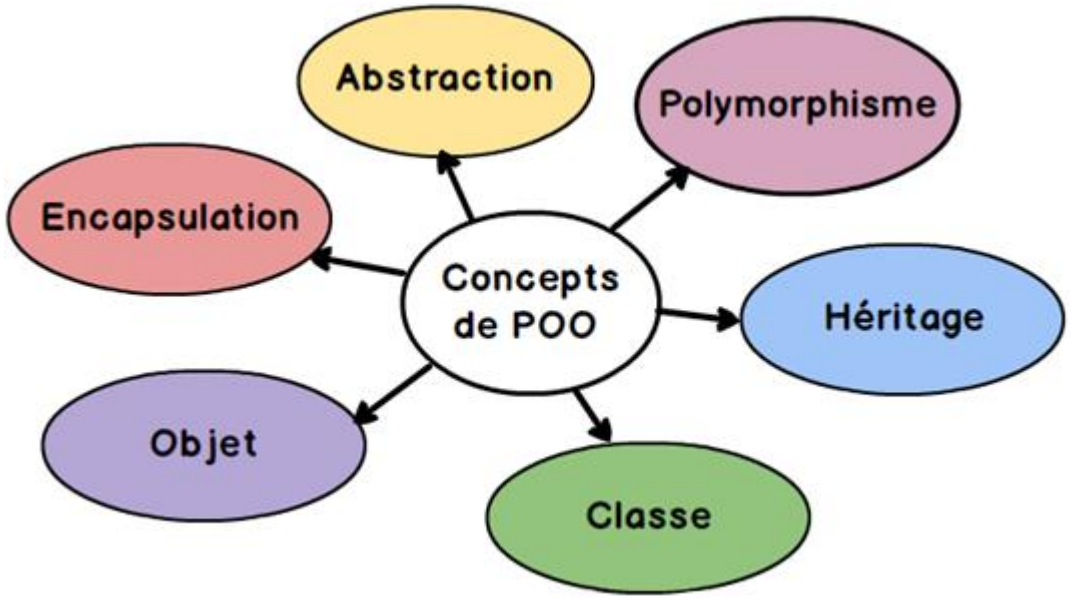
CLASSE : En programmation orientée objet, la déclaration d'une **classe** regroupe des membres, méthodes et propriétés (attributs) communs à un ensemble d'objets. La **classe** déclare, d'une part, des attributs représentant l'état des objets et, d'autre part, des méthodes représentant leur comportement.



OBJET : C'est le concept central de la programmation orientée **objet** (**POO**). En programmation orientée **objet**, un **objet** est créé à partir d'un modèle appelé classe ou prototype, dont il hérite les comportements et les caractéristiques.

L'**encapsulation** permet de définir des niveaux de visibilité des éléments de la classe. Ces niveaux de visibilité définissent les droits d'accès aux données selon que l'on y accède par une méthode de la classe elle-même, d'une classe héritière, ou bien d'une classe quelconque

L'abstraction est le processus qui consiste à représenter des objets qui appartiennent au monde réel dans le monde du programme que l'on écrit



Le nom de **polymorphisme** vient du grec et signifie qui peut prendre plusieurs formes. Cette caractéristique est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le **polymorphisme** est relatif aux méthodes des objets.

L'**héritage** *(en anglais inheritance)* **est** un principe propre à la programmation orientée objet, permettant de créer une nouvelle classe à partir d'une classe existante.

LE CONCEPT DE CLASSE :

Une classe est le **support de l'encapsulation** : c'est un ensemble de **données** et de **fonctions** regroupées dans une **même entité**. Une classe est une **description abstraite** d'un objet. Les fonctions qui opèrent sur les données sont appelées des **méthodes**. **Instancier** une classe consiste à créer un objet sur son modèle. Entre classe et objet il y a, en quelque sorte, le même rapport qu'entre type et variable.
Java est un langage orienté objet : tout appartient à une classe sauf les variables de types primitives.

Pour accéder à une classe il faut en **déclarer une instance de classe ou objet**.

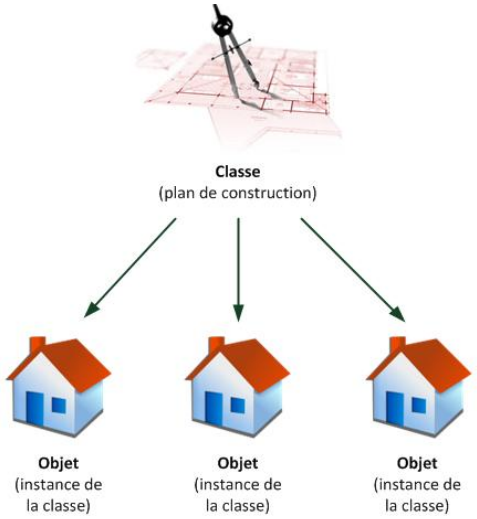
Une classe comporte sa **déclaration**, des **variables** et les définitions de ses **méthodes**.

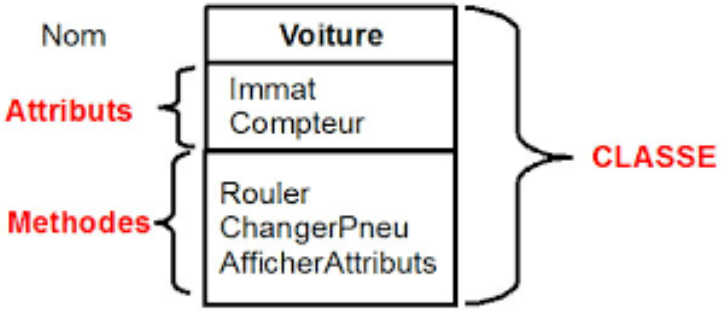
Une classe se compose de deux parties : un en-tête et un corps. Le corps peut être divisé en 2 sections : **la déclaration des données et des constantes** et la **définition des méthodes**. Les méthodes et les données sont pourvues d'attributs de visibilité qui gèrent leur accessibilité par les composants hors de la classe.

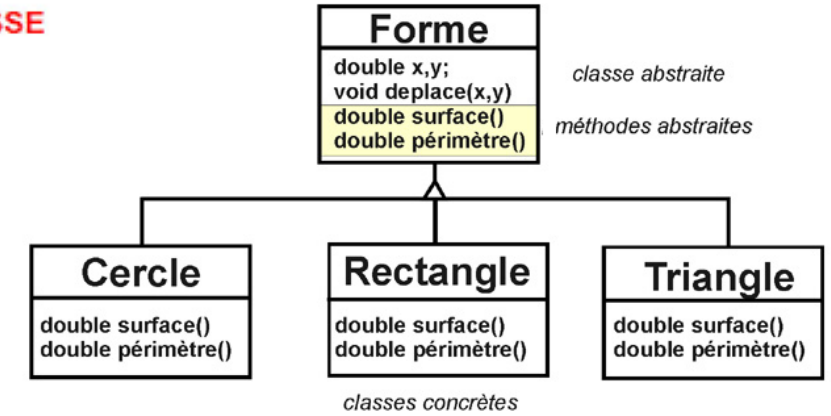
Intégrez ce vocabulaire, il est fondamental ! :

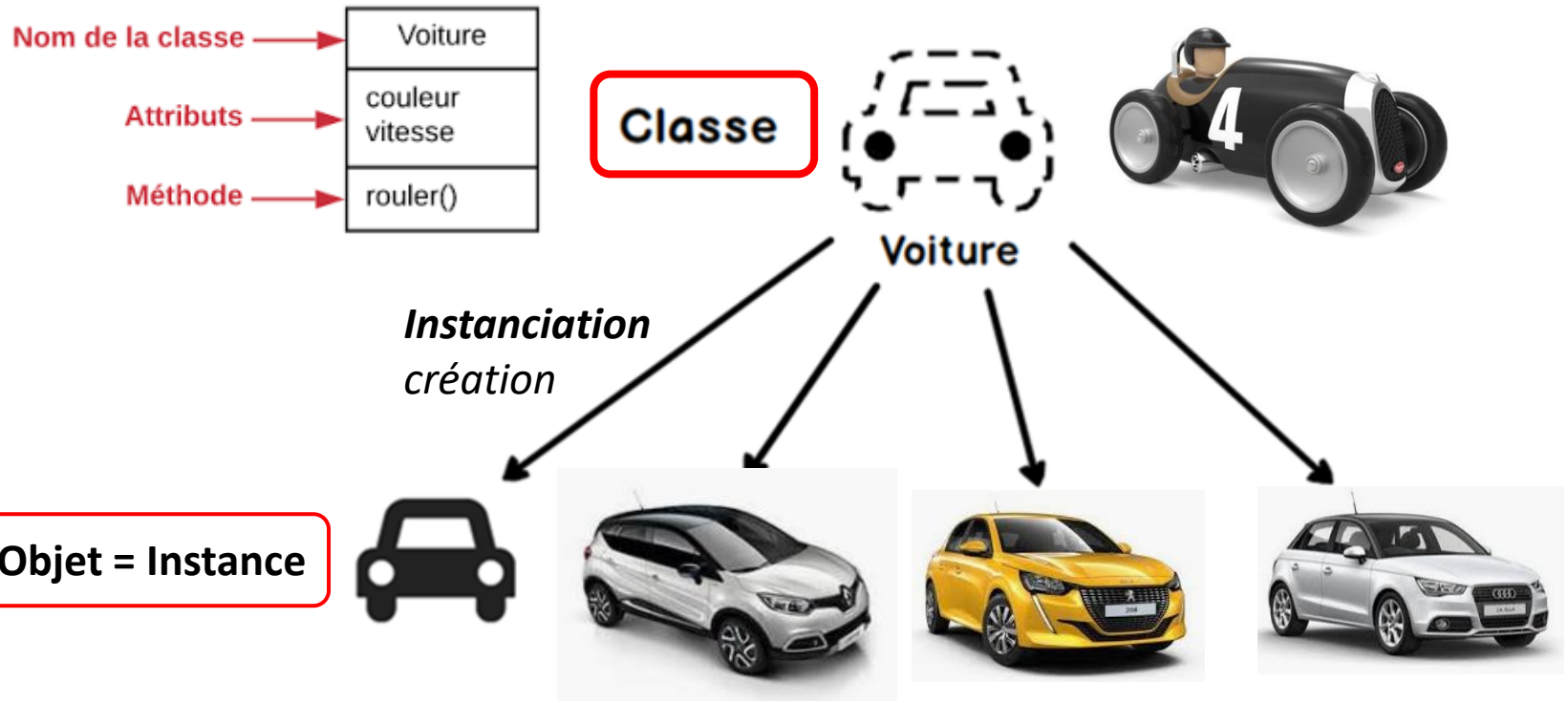
Une **classe** est constituée :

- de **variables**, appelées **attributs ou propriétés** *(on parle aussi de **variables membres**)*
- de **fonctions**, appelées **méthodes** ou **comportements** *(on parle aussi de **fonctions membres**)*









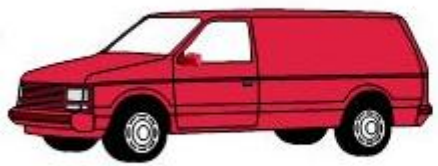
Qu'est-ce que une classe, un objet et une instance?

Les **objets** sont des **instances** de **classes**. En d'autres termes, une instance d'une classe est un objet défini par cette classe particulière.

La **création** d'une nouvelle **instance** ou **d'un objet** s'appelle **instanciation**.

Le programme informatique ne manipule pas des classes, mais des **objets**.
Un **objet**, c'est pour le programme une **instance** d'une **classe** donnée.
L'objet a donc une **existence réelle** pour le programme. Il représente un **objet de la vie réelle.**

CLASSE VOITURE	
ATTRIBUTS	**METHODES**
Marque Modèle Immatriculation Vitesse Couleur Puissance fiscale Etc ...	Démarrer Accélérer Freiner Stopper Vidanger Etc ....



PEUGEOT
PARTNER
120 TN 1234
160 Km /h
rouge
6 CV

RENAUT
Laguna
130 TN5671
190 Km /h
bleu
5 CV

Concevoir une application comme un système d'objets interagissant entre eux apporte une certaine **souplesse** et une forte **abstraction**.

Prenons un exemple : **la machine à café du bureau**.
Nous insérons nos pièces dans le monnayeur, choisissons la boisson et nous nous retrouvons avec un gobelet de la boisson commandée. Nous nous moquons complètement de savoir comment cela fonctionne à l'intérieur et nous pouvons complètement ignorer si le café est en poudre, en grain, comment l'eau est ajoutée, chauffée, comment le sucre est distribué, etc.

Tout ce qui nous importe c'est que le fait de mettre des sous dans la machine nous permet d'obtenir un café qui va nous permettre d'attaquer la journée.

Voilà un bel exemple de **programmation orientée objet**. Nous manipulons un **objet** MachineACafe qui a des **propriétés** (Allumée/éteinte, présence de café, présence de gobelet, …) et qui sait faire des **actions** (AccepterMonnaie, DonnerCafe, …). Et c'est tout ce que nous avons besoin de savoir.

On se moque du fonctionnement interne, peu importe ce qu'il se passe à l'intérieur, notre **objet** nous donne du **café**, point !

LES OBJETS :

Les objets contiennent des **attributs** et des **méthodes**. Les attributs sont des variables ou des objets **nécessaires au fonctionnement** de l'objet. En Java, une application est un objet. La classe est la **description** d'un objet. Un objet est une **instance** d'une classe. Pour chaque instance d'une classe, le code est le même, seules les données sont différentes à chaque objet.

LES PROPRIETES OU LES ATTRIBUTS :

Les données d'une classe sont contenues dans des variables nommées **propriétés** ou **attributs**. Ce sont des variables qui peuvent être des variables d'instances, des variables de classes ou des constantes.

LES METHODES :

Les **méthodes** sont des fonctions qui implémentent les **traitements** de la classe.

LA SURCHARGE DES METHODES :

La **surcharge** d'une méthode permet de définir **plusieurs fois une même méthode** avec des arguments différents. Le compilateur choisi la méthode qui doit être appelée en fonction du nombre et du type des arguments. Ceci permet de **simplifier** l'interface des classes vis à vis des autres classes.

Une méthode est **surchargée** lorsqu'elle exécute des **actions différentes** selon le type et le nombre de paramètres transmis.

Il est donc possible de donner le même nom à deux méthodes différentes à condition que les signatures de ces deux méthodes soient différentes. La **signature** d'une méthode comprend le **nom de la classe**, le **nom de la méthode** et les **types des paramètres**.

LES CONSTRUCTEURS :

La **déclaration d'un objet** est suivie d'une sorte d'initialisation par le moyen d'une méthode particulière appelée **constructeur** pour que les variables aient une valeur de départ. Elle n'est systématiquement invoquée que lors de la création d'un objet.

Le **constructeur** suit la définition des autres méthodes excepté que son nom doit obligatoirement correspondre à celui de la classe et qu'il n'est pas typé, pas même *void*, donc il ne peut pas y avoir d'instruction *return* dans un constructeur. On peut surcharger un constructeur.

La définition d'un constructeur est facultative. Si aucun constructeur n'est explicitement défini dans la classe, le compilateur va créer un **constructeur par défaut** sans argument. Dès qu'un constructeur est explicitement défini, le compilateur considère que le programmeur prend en charge la création des constructeurs et que le mécanisme par défaut, qui correspond à un constructeur sans paramètres, n'est pas mis en oeuvre. Si on souhaite maintenir ce mécanisme, il faut définir explicitement un constructeur sans paramètres en plus des autres constructeurs.

LE DESTRUCTEURS :

Un **destructeur** permet d'exécuter du code lors de la libération, par le *garbage collector*, de l'espace mémoire occupé par l'objet. En Java, les destructeurs appelés **finaliseurs** *(finalizers)*, sont automatiquement invoqués par le *garbage collector*.

LES ACCESSEURS :

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 messages* ».

L'HERITAGE :

L'héritage est un mécanisme qui facilite la **réutilisation du code** et la **gestion de son évolution**.
Elle définit une **relation** entre deux classes :

- une **classe mère** *ou super-classe*
- une **classe fille** *ou sous-classe qui hérite de sa classe mère*

LE PRINCIPE DE L'HERITAGE :

Grâce à l'héritage, les objets d'une classe fille ont accès aux données et aux méthodes de la classe parente et peuvent les étendre. Les **sous-classes** peuvent redéfinir les variables et les méthodes héritées. Pour les variables, il suffit de les redéclarer sous le même nom avec un type différent. Les **méthodes sont redéfinies avec le même nom**, les **mêmes types** et le **même nombre d'arguments**, sinon il s'agit d'une **surcharge**.

LE POLYMORPHISME :

Le **polymorphisme** est la capacité, pour un même message de correspondre à **plusieurs formes de traitements** selon l'objet auquel ce message est adressé.

LES PACKAGES :

Pour réaliser un **package**, on écrit un nombre quelconque de **classes dans plusieurs fichiers** d'un même répertoire et au début de chaque fichier on met la directive ci-dessous où *nom-du-package* doit être composé des répertoires séparés par un caractère point :
package nom-du-package

D'une façon générale, **l'instruction package associe toutes les classes** qui sont définies dans un fichier source à un même package.

La programmation orientée objet – *source JM Doudoux*

L'idée de base de la programmation orientée objet est de rassembler dans une même entité appelée objet les données et les traitements qui s'y appliquent.

Ce chapitre contient plusieurs sections :

- Le concept de classe : présente le concept et la syntaxe de la déclaration d'une classe
- Les objets : présente la création d'un objet, sa durée de vie, le clonage d'objets, les références et la comparaison d'objets, l'objet null, les variables de classes, la variable this et l'opérateur instanceof.
- Les modificateurs d'accès : présente les modificateurs d'accès des entités classes, méthodes et attributs ainsi que les mots clés qui permettent de qualifier ces entités
- Les propriétés ou attributs : présente les données d'une classe : les propriétés ou attributs
- Les méthodes : présente la déclaration d'une méthode, la transmission de paramètres, l'émissio
  messages, la surcharge, la signature d'une méthode et le polymorphisme et des méth
  les constructeurs, le destructeur et les accesseurs
- L'héritage : présente l'héritage : son principe, sa mise en oe
  redéfinition d'une méthode héritée et les interfac
- Les packages : présente la définitio
- Les classes internes : pré
  autre.
-

La POO en Java

Voir le document : MF 2.7_2 - La POO en Java.pdf

La démarche de développement objet

Les paradigmes de représentation permettent de conduire la démarche de développement de façon différente et de pallier dans une large mesure les inconvénients des méthodes classiques.

- La communication entre informaticiens et non-informaticiens est meilleure dans la mesure ou le **concept d'objet est plus intuitif**. D'autre part, les langages objets sont plus proches des schémas de représentation et permettent de mettre facilement en oeuvre les **concepts de l'analyse objet**.

- **L'encapsulation** permet de progresser dans la résolution de l'incompatibilité entre structuration des traitements et structuration des données puisqu'un objet en constitue une synthèse. L'approche orientée objet permet de **substituer à la dualité données et procédures une structure unique.**

- Toutefois, les schémas de représentation de type données et ceux de type traitements sont toujours utilisés dans les phase d'analyse et de conception. Cependant, le premier souci de l'analyste n'est plus de trouver les fonctionnalités du système, mais **d'identifier les objets**. Il faut ensuite attribuer les procédures à ceux-ci. Les priorités sont radicalement différentes : On s'appuie sur la structure sous-jacente du domaine d'application plutôt que sur des besoins fonctionnels liés à un seul problème.

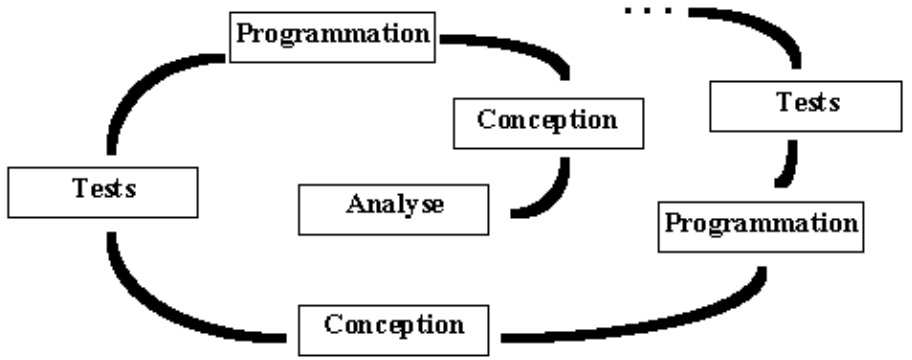
La démarche de développement objet

- Les phases d'études préliminaires sont raccourcies et la conception se fait **par un cycle en spirale** au lieu **d'un cycle en V**. En effet, **l'héritage** permet de tester facilement un objet avant même de le définir complètement.

- Le temps global consommé par les activités de développement n'est par contre pas **plus court en objet** : les gains se font par la suite lors des phases de **maintenance**. D'ailleurs la réversibilité des choix est aisée par l'introduction de nouveaux objets ou l'ajout d'attributs ou de services. Finalement la nature décentralisée de l'architecture objet permet de répartir les décisions tout le long du cycle de développement.

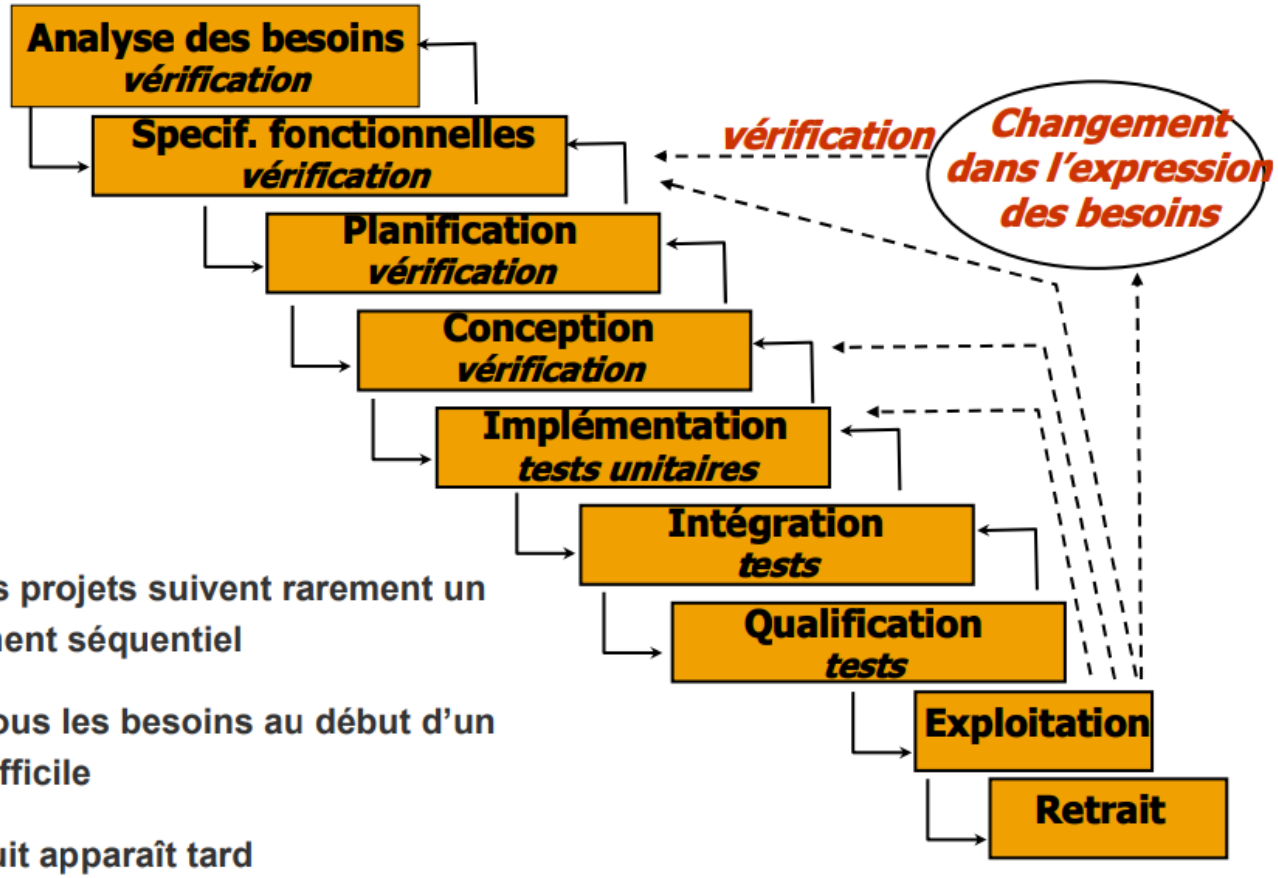
Cycle de conception en spirale

Le processus de développement devient **itératif** plutôt que séquentiel. On peut comparer ainsi les méthodes de développement classiques et objet.



Méthodes classiques	Méthodes objet
Approche globale	Approche locale
Cycle de conception en cascade séquentiel	Cycle de conception en spirale itératif
Problème décomposé en sous-problèmes	Problème réparti entre un réseau objets
Sous-programmes partageant des données	Encapsulation, communication par messages
Séparation des données et des traitements	Pas de séparation
Décisions importantes prises au début du développement	Décisions réparties au cours du cycle de développement

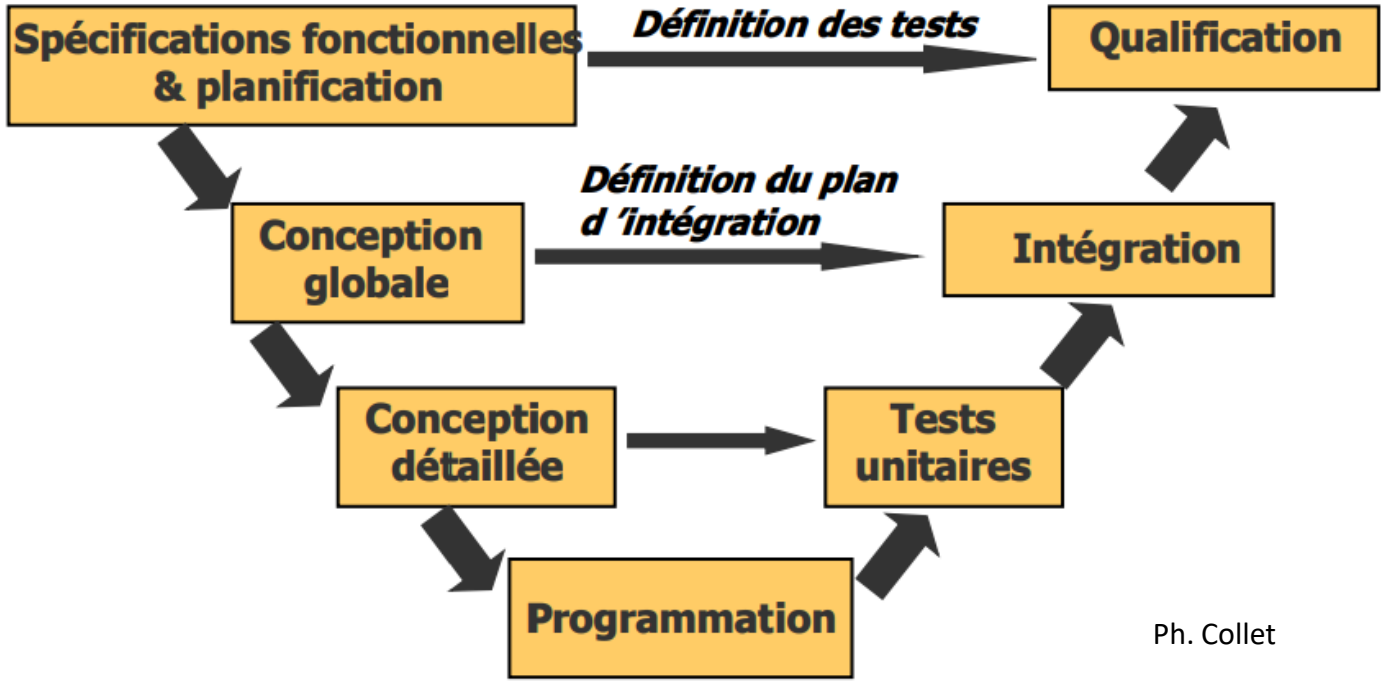
Cycle de conception en cascade



- Les vrais projets suivent rarement un développement séquentiel
- Établir tous les besoins au début d'un projet est difficile
- Le produit apparaît tard
- Echecs majeurs sur de grands systèmes

Ph. Collet

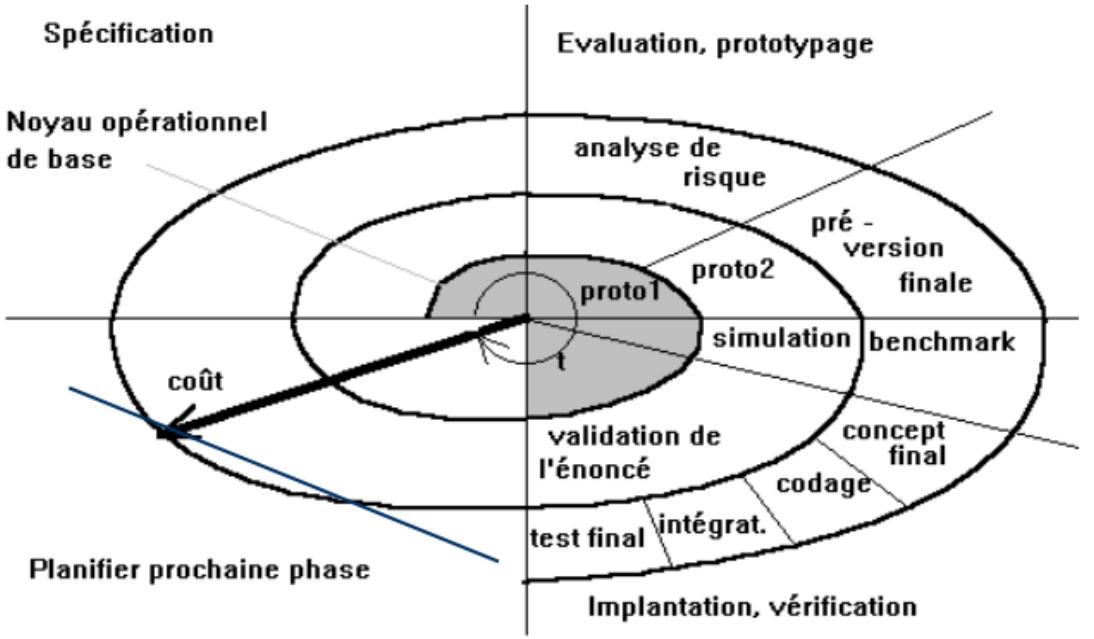
Cycle de conception en V



Ph. Collet

- Meilleure anticipation
- Bonne structuration des tests
- Cadre de développement rigide
- Le produit apparaît toujours tard

Cycle de conception en spirale



- **Incréments successifs => itérations**
  - Approche souvent à base de prototypes
  - Nécessite de bien spécifier les incréments
  - Figement progressif de l'application
- **Mais gestion de projet pas évidente**
- **Pourtant, les méthodes objets dérivent de ce modèle !**

Ph. Collet

Analyste Programmeur en Automatisme, Robotique et Informatique Industrielle
TS ARII

Module MF 2.7
Développer un programme en langage objet
Programmer dans un langage orienté objet

Fin de Présentation