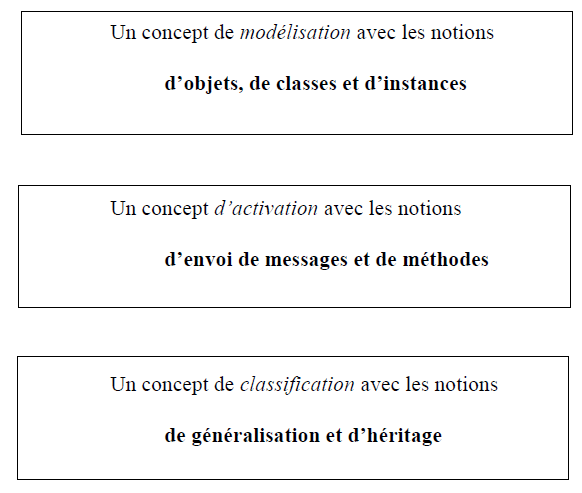
LA Programmation Orientée Objet

Les concepts de base :

La programmation par objets repose sur trois concepts de base.



On appelle réification l’opération essentielle de l’approche objet par laquelle quelque chose est représentée informatiquement sous la forme d’un objet.

Réifier : (Petit Robert) transformer en chose ; syn. de « chosifier ».

Cet objet informatique peut se caractériser ainsi :

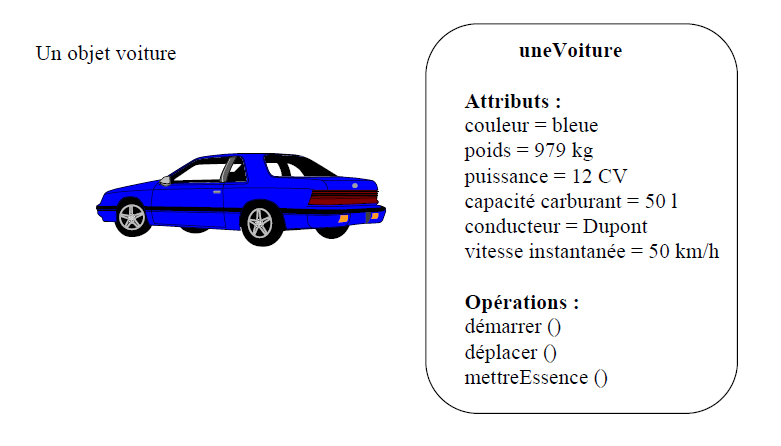


L’identité permet de différencier chaque objet parmi les autres.

L’état est représenté par des attributs (ou champs, variables d'instances).

Le comportement est assuré par des opérations.

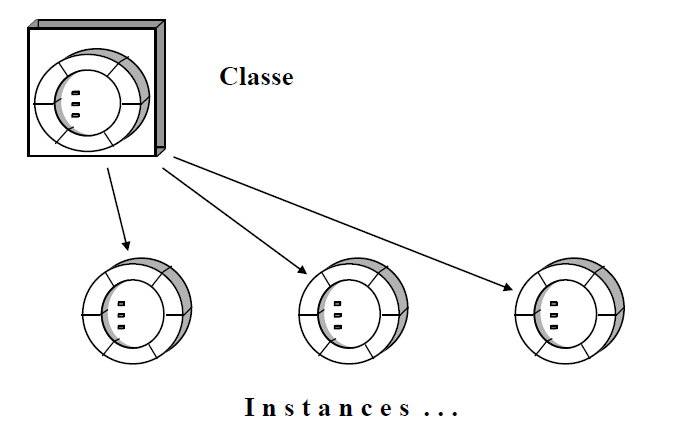
Voici un exemple d’objet et la classe qui lui correspond



Instance

Une instance est un représentant particulier d’une classe.

Elle possède les mêmes attributs et les mêmes opérations que les autres instances de la classe. Les attributs prennent des valeurs distinctes dans chacune des instances.

L'instanciation est le processus par lequel on crée de nouveaux objets, à partir du modèle défini par une classe.

Instancier un objet à partir d’une classe :

L'instanciation est le processus par lequel on crée de nouveaux objets, à partir du modèle défini par une classe.

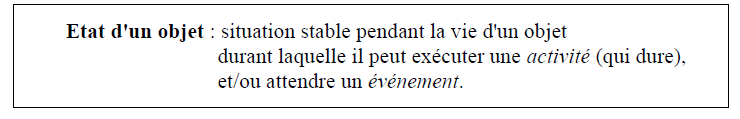
P2 = objet

Point2D = classe



Etat de l’objet

Représente l’aspect statique de l’objet.



C’est la valeur des attributs considérés dans leur ensemble, qui reflète l'état dans lequel se trouve l'objet à un instant donné.

Méthodes

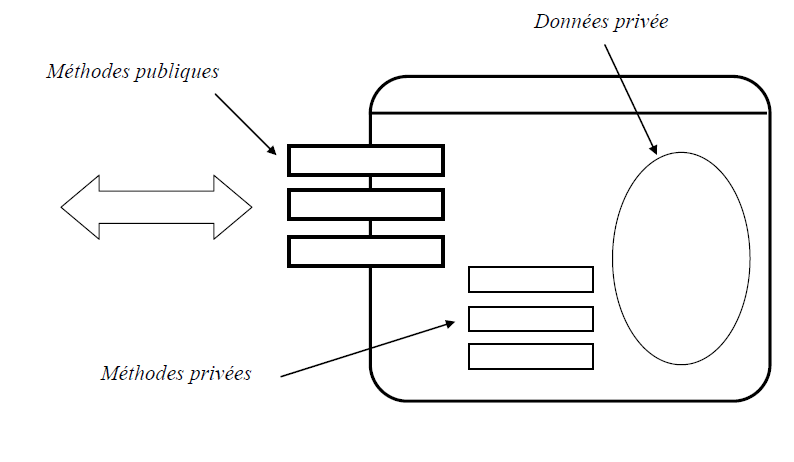
Une méthode est un « petit programme » qui exécute une opération.

Les méthodes sont décrites au niveau des classes.

Chacune à un nom (sélecteur) qui permet de l’activer.

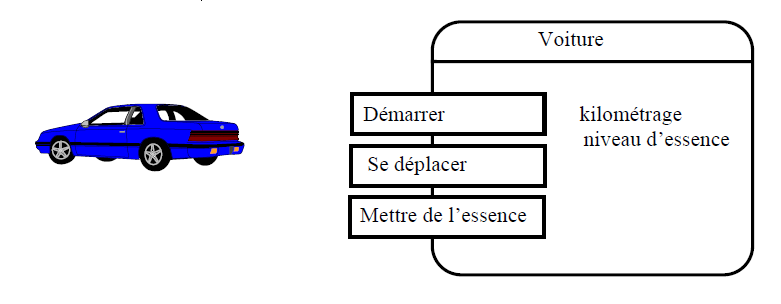
Comportement de l'objet

L’ensemble des méthodes permet d’assurer le comportement des objets.



Encapsulation

L’encapsulation consiste à masquer aux utilisateurs d’un objet les détails relevant de son implémentation (vue interne ou réalisation), et de ne laisser accessible que la vue externe ou interface.



Principe : les attributs ne doivent pas être accessibles de l’extérieur.

Seules les opérations sont habilitées à modifier les valeurs des attributs.

En conséquence, un changement de format de ces attributs ne se répercute pas à l’extérieur.

D’autre part, les opérations peuvent apporter une valeur ajoutée et non pas se contenter d’une simple restitution d’accès aux attributs encapsulés.

Exemple : on ne peut pas faire baisser le niveau d’essence sans se déplacer, ce qui modifie aussi le kilométrage.

Les avantages

L'intérieur de l'objet est protégé. L'accès direct aux données est impossible. Il faut passer par une méthode (publique) constituant l'interface de l'objet pour avoir accès aux données. Ceci garantit la sécurité et l’intégrité des données.

La complexité est dissimulée. Le programmeur n'a plus à se préoccuper ni du détail de la structure des données, ni de la manière dont sont effectués les traitements.

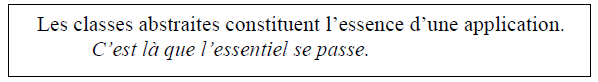
La maintenance est simplifiée. La portée des modifications est limitée à l’intérieur de l’objet concerné. L’implémentation d’une opération peut évoluer sans modifier l’interface de l’objet.

Les échanges avec l'extérieur sont codifiés. Seules les caractéristiques que l’on souhaite offrir aux utilisateurs potentiels de l’objet figurent dans la partie externe.

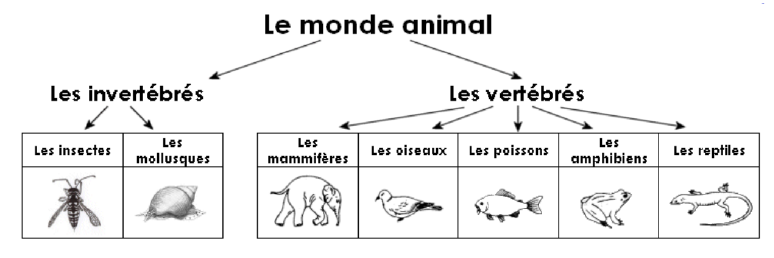
Cette interface constitue un contrat entre l’objet et ses clients, c’est-à-dire entre le concepteur et ses utilisateurs.

Classification

Les hiérarchies de classes ou classifications permettent de gérer la complexité en ordonnant les objets au sein d'arborescences de classes d'abstraction croissante.

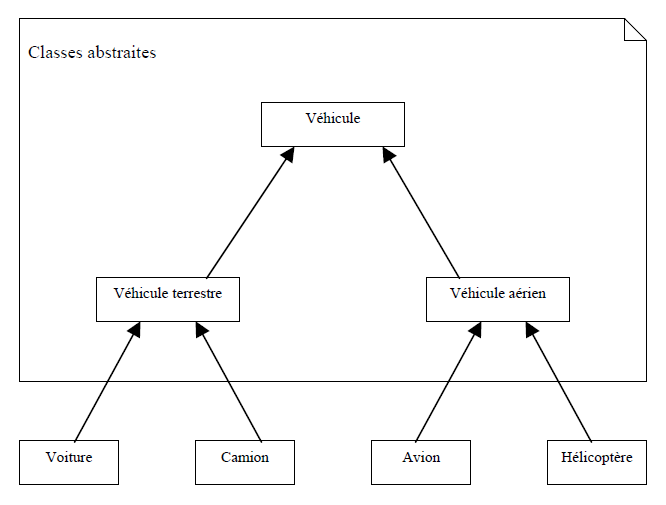


La hiérarchisation des classes correspond à une classification des « essences ».



Généralisation

La généralisation consiste à factoriser les éléments communs d'un ensemble de classes dans une classe plus générale : la super-classe ou sur-classe.



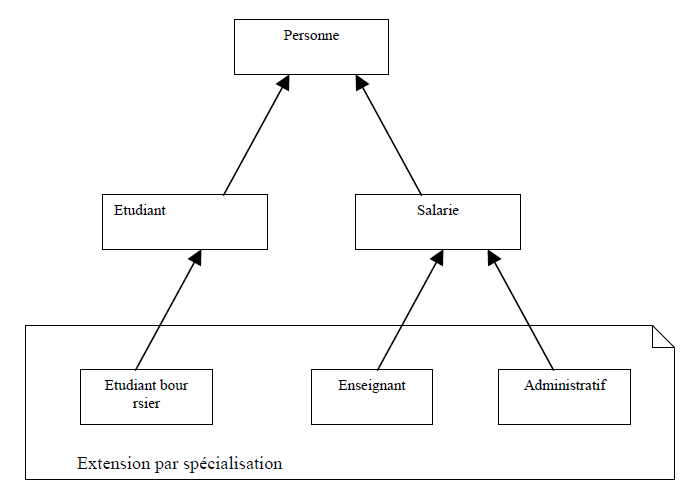
La généralisation est une démarche assez difficile car elle demande une bonne capacité d'abstraction.

La mise au point d'une hiérarchie optimale est délicate et itérative.

Les arbres de classes sont construits en partant des feuilles qui appartiennent au monde réel alors que les niveaux supérieurs sont des abstractions construites pour ordonner et comprendre.

Spécialisation

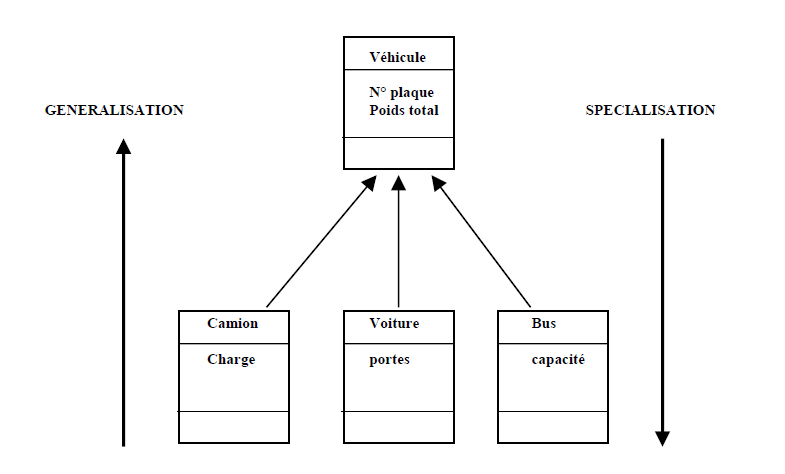
La spécialisation permet de capturer les particularités d'un ensemble d'objets non discriminées par les classes déjà identifiées. Les nouvelles caractéristiques sont représentées par une nouvelle classe, sous-classe d'une des classes existantes.



Généralisation et spécialisation

La généralisation et la spécialisation sont deux points de vue antagonistes du concept de classification.

Elles expriment dans quel sens une hiérarchie de classes est exploitée. Dans toute application réelle, les deux points de vue sont mis en œuvre simultanément.



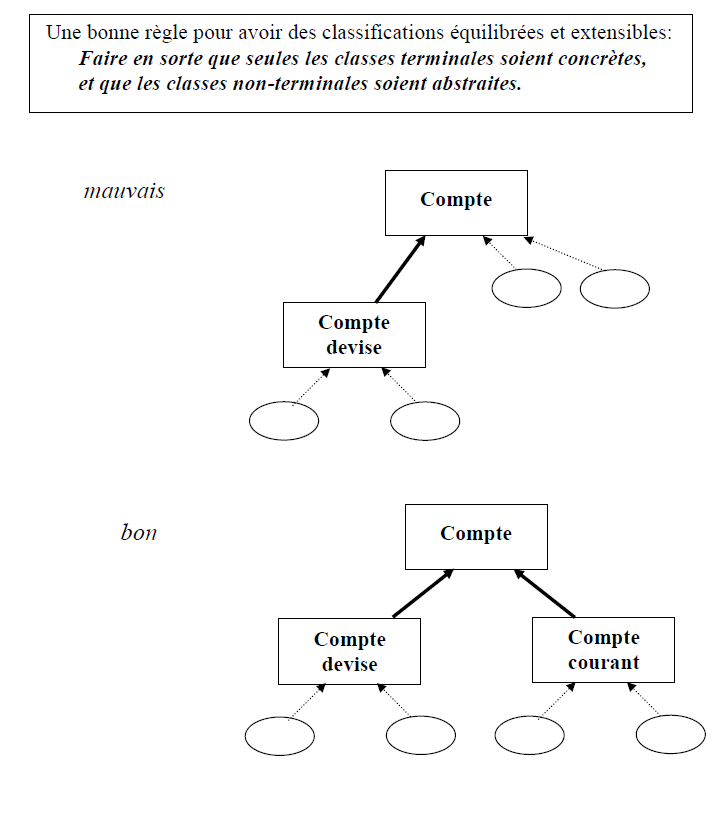
Classes abstraites

Abstraction

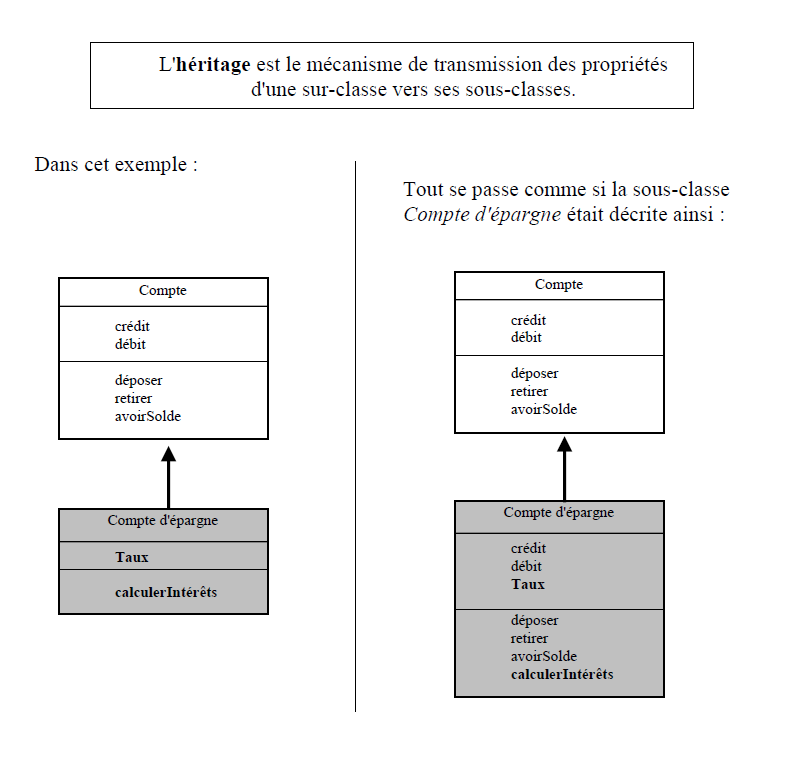
* Classe abstraite : qui ne peut pas avoir d’instance
* Classe concrète : qui peut avoir des instances

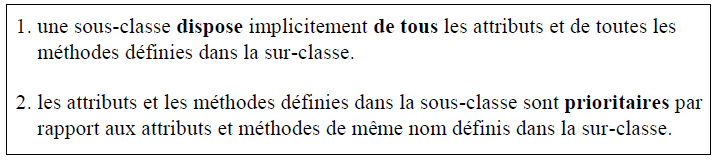
Positionnement

* Classe terminale : qui ne possède pas de sous-classe
* Classe non-terminale : qui possède des sous-classes

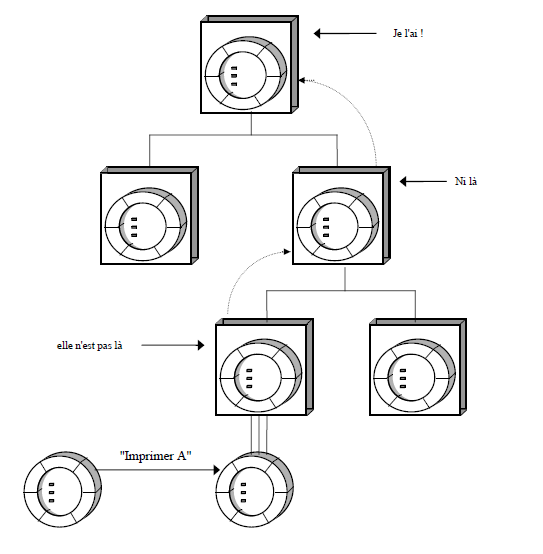
Héritage

On appelle aussi les sur-classes des ancêtres et les sous-classes des descendants.



L’héritage évite la duplication et facilite la réutilisation.

Recherche d’une méthode

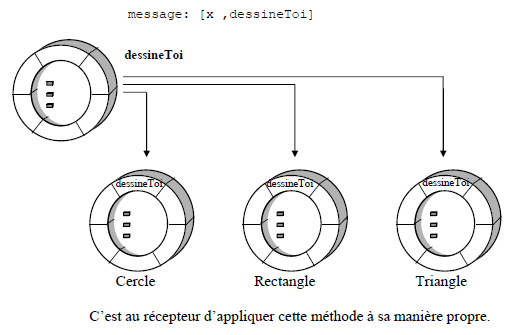


Surcharge

En langage objet, chaque type de forme est représenté par une classe différente.

Grâce à une technique que l’on appelle la surcharge, on utilise le même nom pour les méthodes permettant de dessiner, à l’intérieur de chacune des classes.

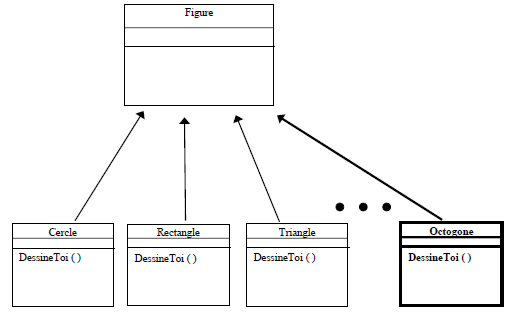
Le fait de demander à une forme de s’auto-dessiner devient étonnamment simple :



Polymorphisme

Le véritable intérêt du polymorphisme est de permettre l’évolution du logiciel avec un minimum de modifications.

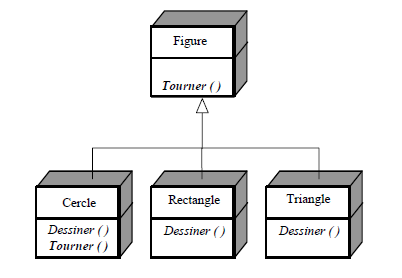
Ajoutons une nouvelle classe (octogone) à l’exemple précédent :

Il suffira de coder la méthode dessineToi dans la classe octogone.

Cet exemple montre l’importance de la dissimulation de l’information. Chaque objet ne doit pas trop en savoir sur les autres.

Redéfinition de méthode

Redéfinir une opération héritée avec un code différent.

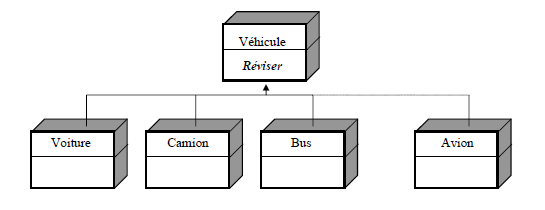
Considérons l'opération Tourner.

Elle est définie dans la classe Figure et est applicable à toutes les formes.

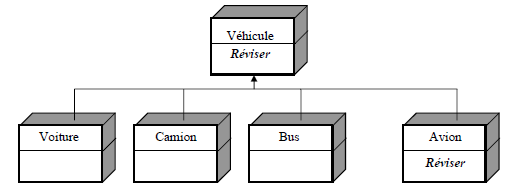
Pour des raisons de performance, on peut être amené à "surcharger" cette opération dans la sous-classe Cercle pour qu'elle soit une opération nulle.

Masquage de méthode

Supposons une hiérarchie de la classe des véhicules comprenant dans la super-classe véhicule une méthode réviser. Ajoutons maintenant la sous-classe avion.



La révision d’un avion n’ayant rien de semblable à celle des autres véhicules, la manière de résoudre ce problème consiste à redéfinir Réviser pour la sous-classe avion afin de tenir compte de ce cas particulier.



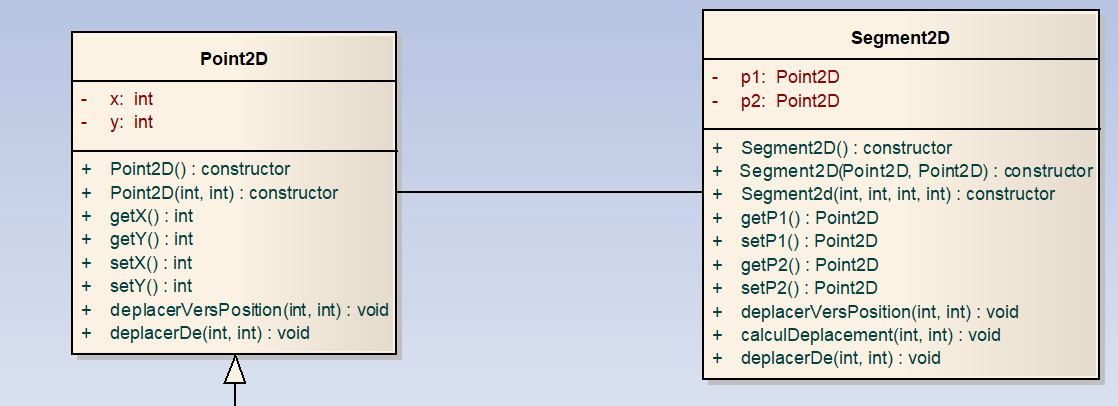
La méthode Réviser de la classe Véhicule traite le cas général.

La méthode Réviser de la classe Avion traite l'exception.

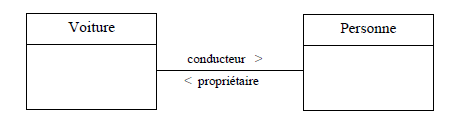
Association

Au niveau objet, certains attributs représentent des liens (ou relations) avec d'autres objets.

Ces liens durent au cours du temps et impliquent que deux objets connaissent chacun l'existence de l'autre.



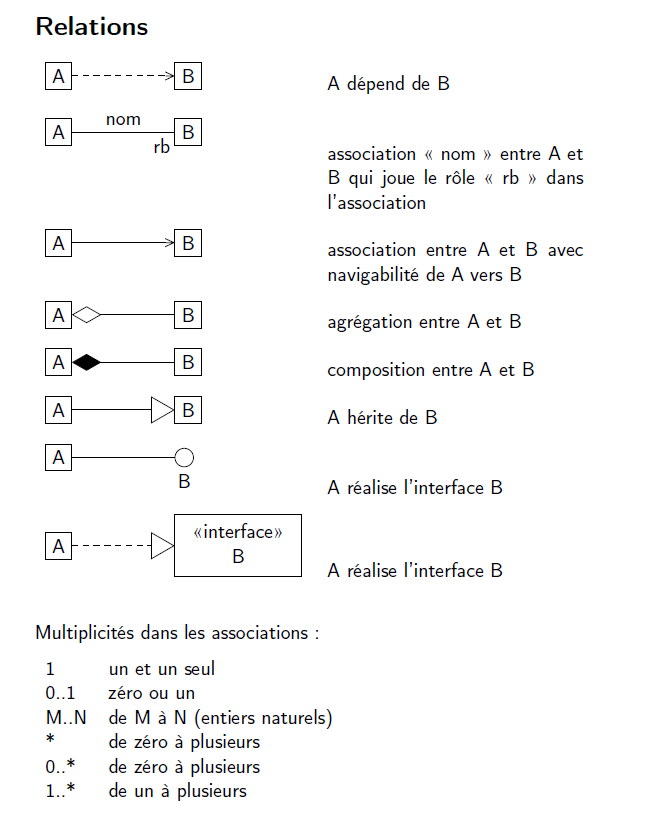
Au niveau classe, l'association exprime une connexion sémantique bidirectionnelle entre classes.



L’association peut être exprimée dans les deux sens :

* Conducteur : de voiture vers personne ;
* Propriétaire : de personne vers voiture.

Relations



Paradigme

Un programme devient un ensemble de petites entités informatiques qui interagissent et

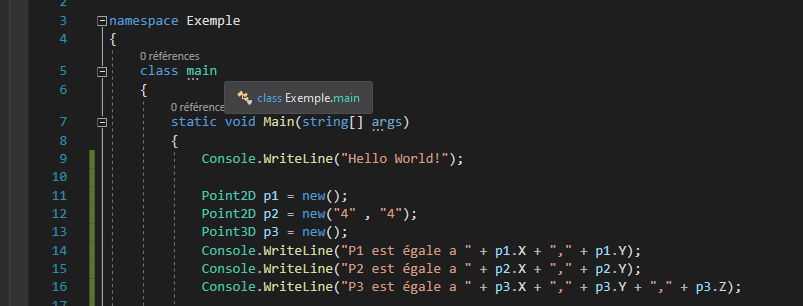
communiquent par messages.

Chacune de ces entités informatiques est autonome et comprend une partie donnée et

une partie traitement.

C’est ce qu’on appelle un objet.

Voici le main :



La 1ere classe utilisé dans cet exemple :

En C# les variables commencent par un \_ exemple :

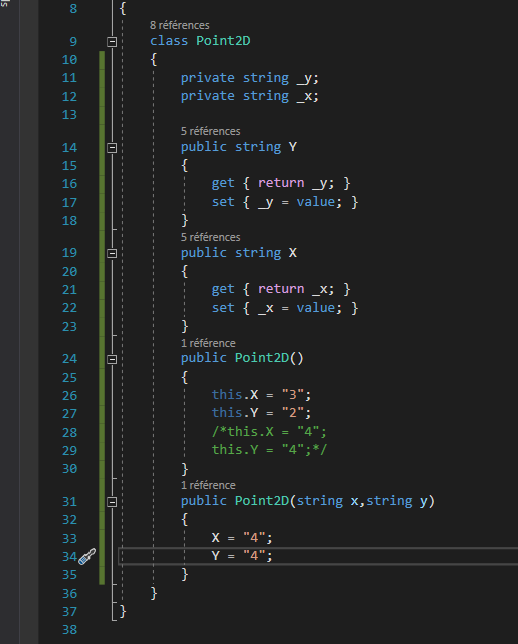
private string \_exemple;

Les variables (attributs) sont généralement en privé

Les méthodes (opération) sont en public

Dans les méthodes les **get** permettent de retourner la valeur

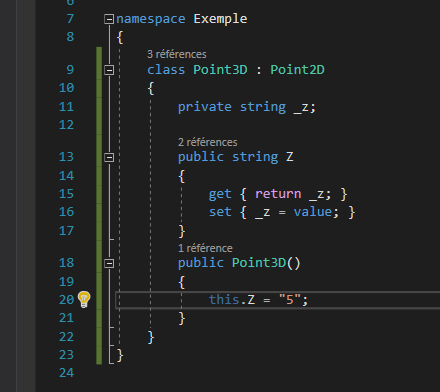
Les **set** permettent de paramétrer la variable



Un exemple d’héritage :

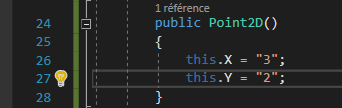
class Point2D : Point3D

Le **:** est la notion d’héritage dans le code qui hérite de la classe Point2D



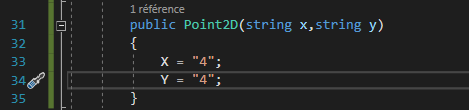
Un constructeur par défaut :

Tous les objets créés ont les mêmes paramètres



Un constructeur avec paramètre :

Les objets crée peuvent avoir diffèrent paramètre



: base () fait appelle à la super classe (ses paramètres)

Dans ce cas la super classe est Points2D (exo point 2D)

