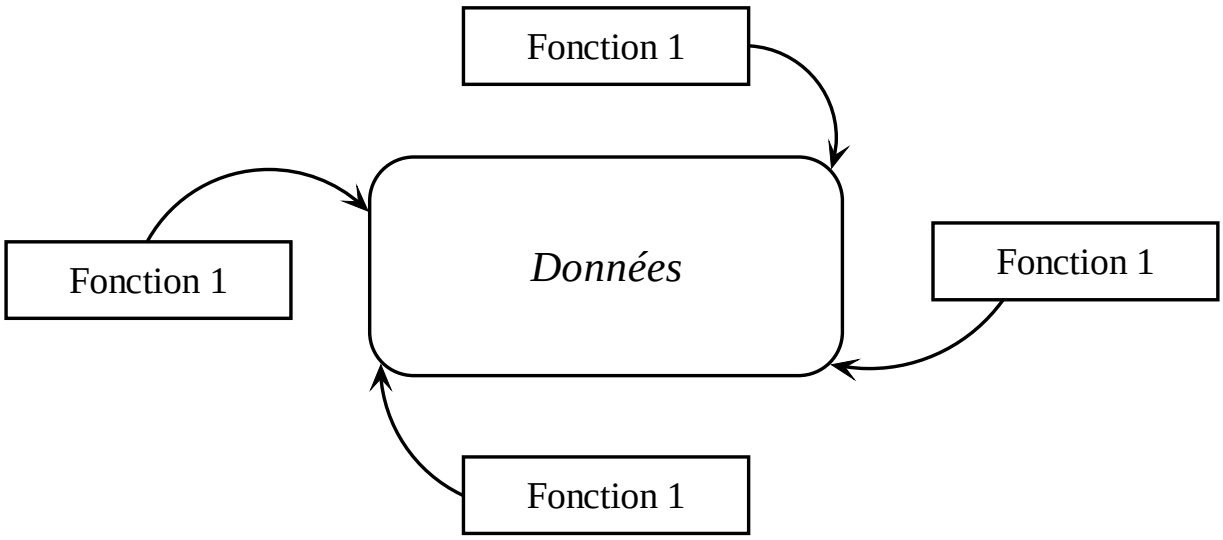
Principes de la Programmation Objets

1. La programmation procédurale

La programmation procédurale (C, Pascal, Basic, …) est constituée d'une suite d'instructions (souvent réunies en fonctions) exécutées par une machine. Ces instructions ont pour but d'agir sur des données pour produire une effet quelconque.

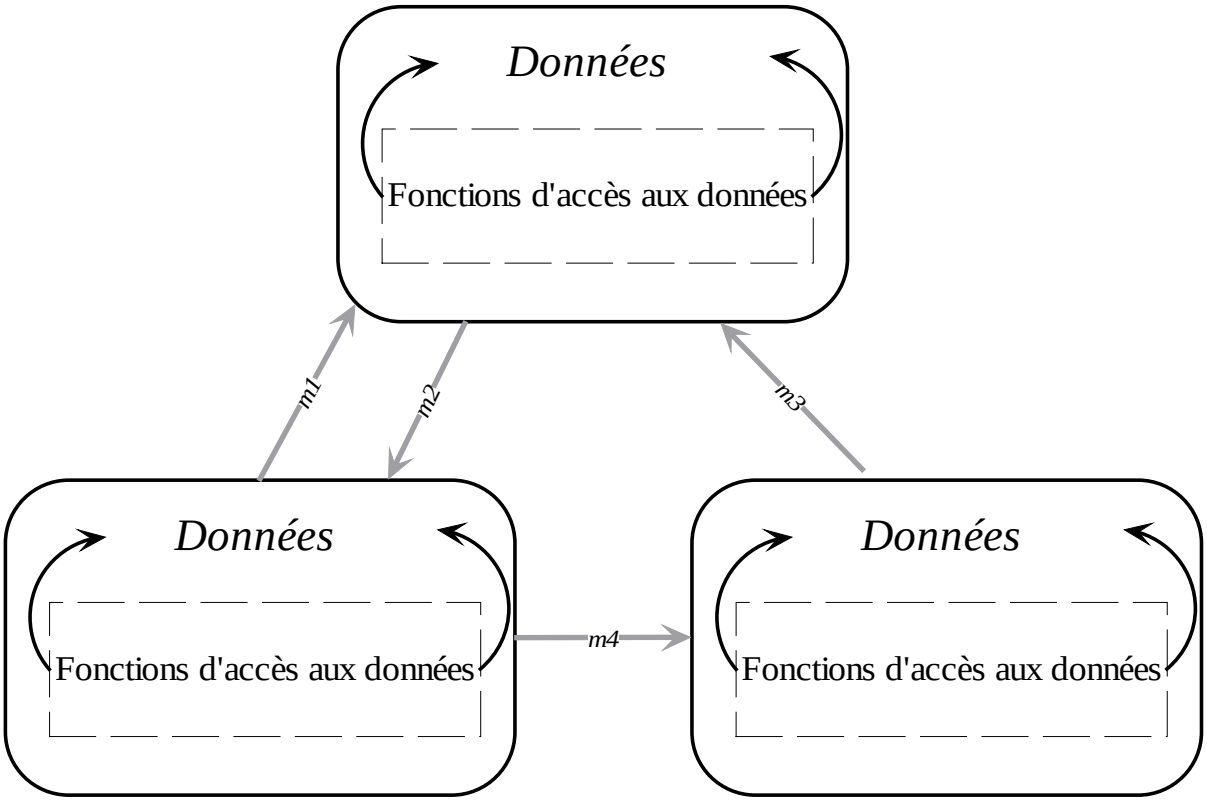


Les fonctions, procédures et autres suites d'instructions accèdent à une zone où sont stockées les données. Il y a donc une dissociation entre les données et les fonctions ce qui pose des difficultés lorsque l'on désire changer les structures de données.
Dans les langages procéduraux, les procédures s'appellent entre elles et peuvent donc agir sur les même données provoquant ainsi des effets de bord. De ces problèmes sont issus une autre manière de programmer : la programmation par objet.

2. Les objets

Un programme est vu comme un ensemble d'entités, une société d'entités. Au cours de son exécution, ces entités collaborent en s'envoient des messages dans un but commun.

Philippe.Roose@iutbayonne.univ-pau.fr



Nous avons dans ce schéma un lien fort entre les données et les fonctions qui y accèdent (tant pour les consulter que pour les créer).

Mais qu'appelle-t-on un objet ? Que représente un objet ?

Définition

Un objet représente une entité du monde réel, ou de monde virtuel dans le cas d'objets immatériels, qui se caractérisent pas une identité, des états significatifs et par un comportement.

L'*identité* d'un objet permet de distinguer les objets les uns par rapport aux autres. Son *état* correspond aux valeurs de tous les attributs à un instant donné. Ces propriétés sont définies dans la classe d'appartenance de l'objet. Enfin, le *comportement* d'un objet se défini par l'ensemble des opérations qu'il peut exécuter en réaction aux messages envoyés (un message = demande d'exécution d'une opération) par les autres objets. Ces opérations sont définies dans la classe d'appartenance de l'objet.

Prenons un exemple :

Dans une entreprise, un employé TOTO (son numéro de sécu 1 05 73 123 456 permet son identification unique dans l'entreprise), est employé sur un poste de technicien dans la société X située à Biarritz.
Son état est caractérisé par l'ensemble des valeurs de ses attributs.

- num sécu : 1 05 73 123 456
- nom : TOTO
- qualification : Technicien
- Lieu de travail : Biarritz

Son comportement est caractérisé par les opérations qu'il peut réaliser :

- entrer/sortir de la société
- changer de qualification
- changer de lieu de travail

Bref, l'ensemble des opérations pouvant affecter ses attributs.

Un peu de vocabulaire

Une classe est l'abstraction d'un ensemble d'objets qui possèdent une structure identique (liste des attributs) et un même comportement (liste des opérations). C'est un modèle décrivant le contenu et le comportement des futurs objets de la classe. Un objet est une *instance* d'une et une seule classe.

Si l'on reprend l'exemple précédent, la classe est l'employé, l'instance de classe (l'objet) est l'employé *1 05 73 123 456, TOTO, …*

Par rapport à une approche « classique » que l'on peut observer dans la programmation procédurale, l'approche objet se caractérise par le regroupement dans une même classe de la description des attributs et des opérations (méthodes). Ce regroupement s'appelle l'*encapsulation*. On masque l'information au monde extérieur.
Ce principe d'encapsulation améliore l'autonomie et l'indépendance de chaque classe et augmente le potentiel de réutilisation de chacune d'elles.
La visibilité des attributs et des opérations d'une classe vis à vis d'autres porte le nom d'*interface*.

Un objet est donc composé de données et d'opérations (appelées méthodes) sur ces données. C'est une variable de type abstrait représentant une entité du domaine du problème.

Il se compose de deux parties :

- une partie interface composée d'opérations que l'on peut faire dessus (partie publique) – c'est cette partie qui est visible par les utilisateurs de l'objet.
- une partie interne composées des données sensibles de l'objet (partie privée).

Programmation Procédurale	**VARIABLE**	**TYPE**
Programmation Orientée-Objet	**OBJET**	**CLASSE**

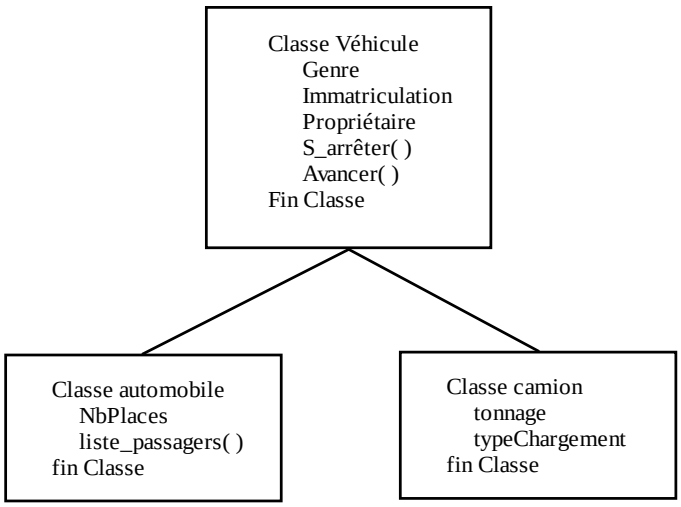
Les avantages des propriétés ci-dessus sont multiples :

- meilleure modularité. Les communications entre objets sont réalisées par le biais d'opérations d'interface.
- meilleure sécurité. Le code de chaque méthode ne s'applique que sur les données de l'objet. Certaines parties de l'objet ne sont pas accessibles pour certains (et n'ont pas d'ailleurs à être connues).
- meilleure conception. Les données et les méthodes sont spécifiées en même temps.
- meilleure lisibilité. Les données et les méthodes sont décrites au même endroit.
- meilleure portabilité. Les parties masquées pourront être modifiées sans que l'utilisateur n'ai à changer son code puisqu'il n'a pas directement accès à celles-ci. C'est ici que l'on pourra mettre des points dépendant des machines.

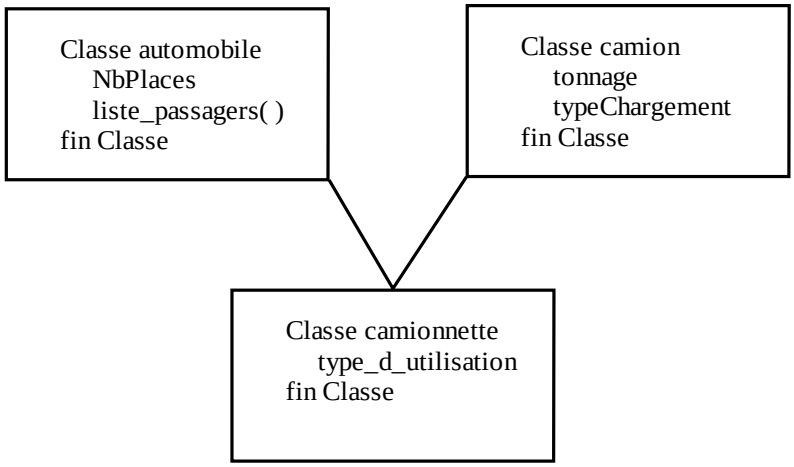
3. Héritage

L'héritage permet de construire une classe à partir d'une ou de plusieurs autres.

Exemple :

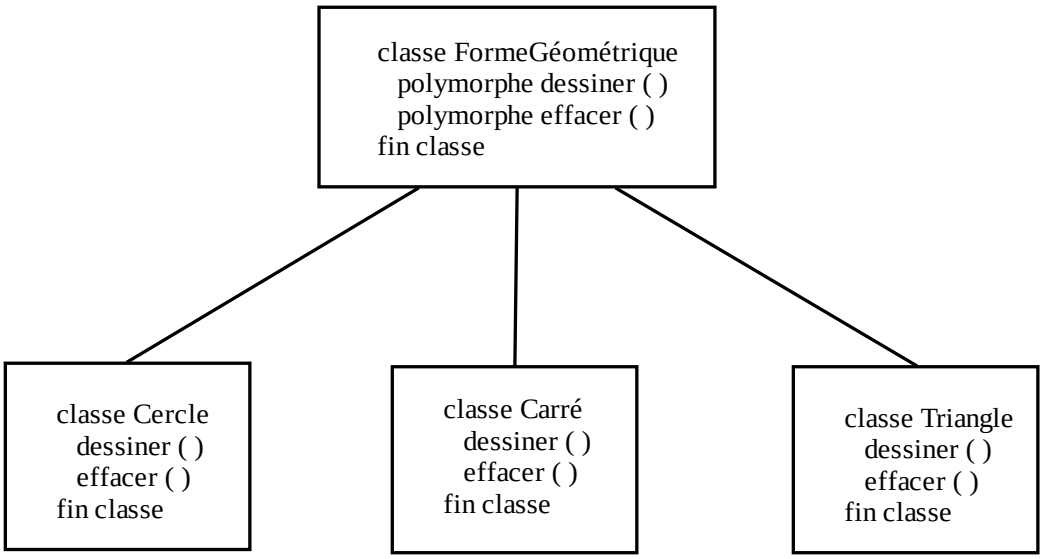


Une classe dont on dérive est appelée *classe de base*, celle qui l'utilisent est appelé *classe dérivée*. Les classes *automobile* et *camion* sont donc des classes dérivées, la classe de base est *véhicule*.



L'héritage multiple offre la possibilité de pouvoir reprendre intégralement des travaux déjà réalisés. Il offre la possibilité de regrouper en un seul endroit ce qui est commun à plusieurs.

4. Polymorphisme



Les méthodes *dessiner( )* et *effacer( )* sont polymorphes. Leur nom est similaire dans les trois classes dérivées, mais dessiner un cercle est différent de dessiner un carré ou un triangle. Ainsi le polymorphisme se résume par : un même nom, plusieurs implémentations.

A chaque fois que l'on appelle une des deux méthodes dessiner ou effacer, il est nécessaire que l'on appelle celle qui est associée à la forme géométrique correspondante. Ce choix ne peut se faire qu'à l'exécution, on a donc une liaison dynamique. La détermination du choix de la bonne fonction se fait de manière automatique. On ne s'en s'occupe pas au moment du codage.

En programmation procédurale, on aurait :

```
dessiner(type_dessin) {
  switch (type_dessin)
  case CARRE :
  … break ;
  case CERCLE :
  … break ;
  case TRIANGLE :
  … break ;
}
```

En POO (Programmation Orientée Objet), on écrit :

```
Classe CARRE {
  Dessiner ( ) ;
}

Classe CERCLE {
  Dessiner ( ) ;
}
Classe TRIANGLE {
  Dessiner ( ) ;
}
```

Selon les langages, il est nécessaire de préciser si une méthode doit être polymorphe ou pas. C'est le cas en C++, ça ne l'est pas en Java ou en SmallTalk.

Les avantages du polymorphisme sont les suivants :

- Les fonctions ayant la même sémantique ont le même nom,
- Le programmation est plus souple. Si on veut ajouter une classe Rectangle…y'a qu'à ! Il suffit d'ajouter les méthodes dessiner et effacer dans cette classe. Dans le cas de la programmation procédurale, il aurait fallu reprendre le code et l'enrichir.

5. Les Langages Orientés Objets

Un langage est orienté objet s'il possède les mécanismes supportant le style de programmation orienté objet, c'est à dire, s'il autorise les facilités inhérentes à ce type de programmation.

Le C++ est sans contexte le langage objet le plus utilisé. C'est néanmoins un langage hybride autorisant à la fois la programmation procédurale dite « classique » mais aussi et surtout la programmation orientée objet.
Ses avantages sont sans contexte sont efficacité et sa portabilité. Il possède un typage fort et est supporté par nombres d'ateliers de génie logiciel.
Néanmoins, tout n'est pas rose dans ce bas monde, il possède quelques défauts comme l'absence de gestion automatique de la mémoire (le fameux ramasse-miette que l'on retrouve en Lisp et en Java par exemple). Il est également complexe et réservé aux expert. Ces critiques peuvent se discuter.

Le C++ a été conçu par Bjarne Stroustrup aux laboratoires ATT & Bell. C'est une extension (d'ou l'opérateur ++ d'incrémentation de C) du langage C.
Il vise trois objectif : efficacité, vérification de types, programmation orientée objet.

Points de repères historiques :

- 1978 : La langage C de Kernighan et Ritchie, ATT & Bell Laboratory
- 1980 : SmallTalk 80 de Goldberg du Xerox Palo Alto Research Center.
- 1983 : C++ de Stroustrup de ATT & Bell – c'est aussi le C ANSI qui est crée à cette époque.
- 1985 : Commercialisation C++ Version 1 – liaison dynamique, surcharge des opérateurs, références.
- 1987 : Début des efforts de normalisation
- 1989 : C++ Version 2 – héritage multiple
- 1991 : C++ Version 3 – classes paramétrées (templates), exceptions
- milieu 1990 : Normalisation ANSI su C++
- 1995 : JAVA