

Chapitre 11

Introduction à la programmation orientée objet

1. De la programmation par Goto à la programmation structurée

1.1. Branchement par Goto

- Un simple test sur les valeurs des données:
 - Interruption de l'exécution du jeu d'instructions,
 - Branchement vers une autre partie du programme.
- Si nous sommes en présence d'une grosse application, le programme à réaliser sera alors gros et complexe:
 - Lisibilité, maintenance et réutilisation du code ne sont pas favorisées.

1.2. Programmation structurée

- Décomposition d'une tâche en termes de sous-programmes.
 - Analyse du problème de manière descendante ("Top-Down").
- N.B.: - La programmation structurée s'appuie sur des spécifications stables or les spécifications sont loin d'être stables => Fragilité du système logiciel ayant une conception "Top-Down",
- La spécialisation des fonctions ne favorise pas leur réutilisation.

2. Coût des logiciels par rapport aux résultats

- Une enquête réalisée par ACM sigsoft (Eng. Notes V10, No 5, Oct. 1985) sur 9 projets du gouvernement américain

Résultats	coûts	pourcentages
Logiciels utilisés tels que livrés	0.1 M\$US	1.5%
Logiciels utilisés après modifications	0.2 M\$US	3.0%
Logiciels abandonnés ou en cours de modifications	1.3 M\$US	19.0%
Logiciels livrés mais non utilisés	2.0 M\$US	29.5%
Logiciels payés mais non livrés	3.2 M\$US	47.0%

- S'orienter vers de nouvelles approches autres que celles utilisées, à l'époque (1985), dans la conception et le développement des applications.

3. Critères de qualité d'un logiciel

- Les facteurs de qualité d'un code [Object Oriented Software Construction, B.Meyer, Prentice Hall, 1997],

3.1. Correct

- Le logiciel est capable de produire exactement les fonctions qu'on lui demande par les spécifications.
- Demander 20\$ au guichet bancaire automatique => obtenir 20\$.

3.2. Robuste

- Le logiciel est capable de bien fonctionner dans des conditions anormales.
- Pitonner une séquence NON prévue au guichet bancaire automatique => ne donne pas 500\$ en conséquence.

3.3. Extensible

- Il est possible de modifier le logiciel simplement pour l'adapter à des modifications dans les spécifications.
- Ajouter le prêt hypothécaire dans le guichet bancaire automatique.

3.4. Réutilisable

- Le logiciel peut être utilisé intégralement ou en partie dans de nouvelles applications.
- Distributeur de boissons => Distributeur de tickets de métro.

3.5. Compatible

- C'est la possibilité de combiner le code du logiciel à d'autres codes.
- Formats des fichiers non compatibles d'un OS à un autre. En UNIX, format des fichiers « texte » a été standardisé à une simple séquence de caractères.

3.6. Portable

- C'est la facilité d'exécuter le logiciel sur différentes plates-formes.
- Utilisation des fonctions génériques plutôt que spécifiques.

3.7. Efficace

- Cela se traduit par la bonne utilisation des ressources (temps, mémoire, etc.)
- Attendre 15 minutes au guichet bancaire automatique pour retirer 20\$!

4. Approche orientée objet

- Idée de base de l'Approche Orientée Objet (A.O.O.) repose sur l'OBSERVATION de la façon dont nous procédons dans notre vie de tous les jours.
- Nous sommes entourés d'objets que nous manipulons. Il nous importe peu de savoir comment ils sont fabriqués.
- Dans le domaine du développement d'applications, rien n'existe sauf les DONNÉES.
- Construire les mécanismes qui:
 - structurent ces données;
 - les régissent;

⇒ Afin de les utiliser.

5. Programmation d'une Application de Gestion de Comptes Bancaires

- L'application sert à gérer des dépôts/retraits d'argent.
- Le Programmeur ne connaît que les données qui caractérisent les comptes ;
 - noms des clients,
 - numéros de comptes,
 - types de comptes (compte courant, compte épargne etc.),
 - soldes des comptes etc.
- La seule chose importante est:
 - Le dépôt ou le retrait d'argent,
- Pour ce faire, il faut:
 - Comprendre la mécanique qui régit le fonctionnement des comptes.

- À la disposition du programmeur des objets: les différents types de comptes bancaires.
- Chaque objet sait comment se comporter:
 - Un compte courant saurait qu'il n'accorde pas d'intérêts pour l'argent déposé.
- Programmation de l'application consiste à transmettre à ses objets un message pour leur dire qu'on désire déposer ou retirer de l'argent et, à eux de faire le reste.
- Écrire une application orientée objet => transmettre des ordres d'actions à des objets (préexistants et autonomes).
- Si cela est réalisable => l'écriture et la maintenance des applications doivent s'en trouver considérablement simplifiées.

6. Programmation Orientée Objet

- Programmation dans laquelle les programmes sont organisés comme des ensembles d'objets coopérant ensemble.

6.1. Objet

- Entité fermée douée de mémoire et de capacité de traitement,
- agissant sur réception d'un message,
- pouvant fournir un résultat.
- Objet: voiture,
 - répond au message: tourner la clé de contact.
- Un objet est formé de :
 - données => définissent ce qu'il est,
 - programmes ou procédures => définissent ce qu'il peut faire,
- Un objet est un regroupement dans une entité indépendante de données et de procédures qui manipulent ces données. Ces procédures sont appelées MÉTHODES.
- Les données sont séparées du monde extérieur par les méthodes. Ces dernières définissent l'interface de l'objet ~ notion d'ENCAPSULATION.

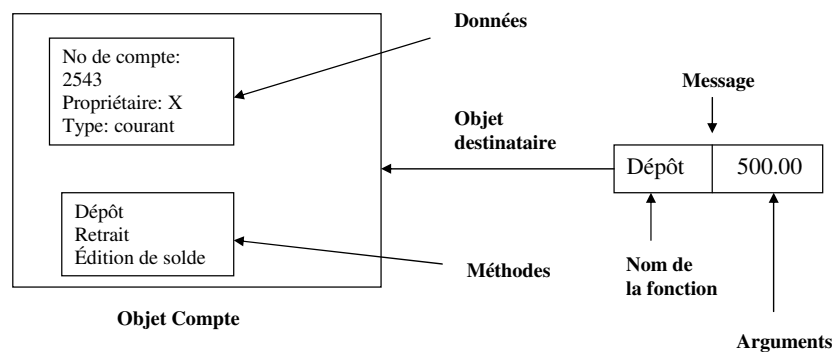
- Un objet est composé de 2 parties:
 - Partie interface: opérations qu'on peut faire dessus (partie publique)
 - Partie interne: données sensibles de l'objet (partie privée)
- Les utilisateurs de l'objet ne voient que la partie interface.
- Nous ne verrons de l'objet que ce qu'il peut faire et le résultat de ce qu'il fait mais jamais la façon avec laquelle il a procédé pour le faire,
 - Retrait: objet permet le retrait de l'argent,
 - Résultat: l'argent retiré,
 - Méthodes (processus): aucune idée (peu importe).

6.2. Encapsulation

- regroupement de codes et de données,
- dissimulation d'informations au monde extérieur,
- Parmi les avantages:
 - meilleure modularité => les communications entre objets sont traitées par les opérations d'interface.
 - meilleure sécurité => certaines parties de l'objet sont inaccessibles et n'ont d'ailleurs pas à être connues.
 - simplicité apparente pour l'utilisateur => Il n'y aura pas d'impact sur l'utilisateur de l'objet si le contenu de ce dernier est amené à changer et à condition que l'interface ne soit pas modifiée.

6.3. Communication avec les objets

- MESSAGE:
 - transporte l'information nécessaire aux demandes à satisfaire,
 - moyen UNIQUE de communication avec les objets (impossible d'accéder directement aux données encapsulées d'un objet).
 - contient:
 - nom de l'objet destinataire,
 - énoncé de la demande (exemple: le nom d'une fonction),
 - les arguments nécessaires (pour réaliser la demande)



6.4. Polymorphisme

- Faculté qu'ont des objets différents de réagir différemment en réponse au même message.

- Marcher sur la queue d'un chat => il miaule,
- Marcher sur la queue d'un chien => il aboie.

- Même nom de fonction, plusieurs implantations:

- Fonction: opération addition (+)
 - Addition des nombres entiers,

$$1 + 2 = 3$$

- Addition des nombres complexes,

$$1+2i + 3+4i = 4+6i$$

```
classe CARRE
    dessiner();
fin classe

classe CERCLE
    dessiner();
fin classe

classe TRIANGLE
    dessiner();
fin classe
```

- Même libellé de la fonction **dessiner()**.
 - Les fonctions ayant la même sémantique ont le même nom.
- Programmation plus souple => ajouter une classe RECTANGLE ...
 - Il suffit de le faire et d'écrire la méthode **dessiner()** dans cette classe.
- En programmation procédurale, il faut reprendre le code **dessiner()** (il faut déjà l'avoir ce qui n'est toujours pas le cas) et l'enrichir (sans le détériorer, histoire de pouvoir toujours **dessiner()** un cercle par exemple).

6.5. Classes instances

- Opération mathématique: $2 + 6$

Objet: **2**

Message: **+6**

Le signe **+** est le sélecteur ... méthode addition.

L'objet **2** reçoit le message **+6**, la méthode dont le nom correspond au sélecteur (ici **+**) est exécutée.

Mais le **6** est lui-même un objet,

$6 + 2$

Objet: **6**

Message: **+2**

Le signe **+** est le sélecteur ... méthode addition.

- Objets : Nombres entiers, différents par leur partie DONNÉES.
- Créer un nombre infini d'objets? IMPOSSIBLE.
- Créer un objet générique "MOULE ou PROTOTYPE", contient les méthodes communes à tous les objets "nombres entiers" Sa donnée est en réalité une variable susceptible de contenir au moment de l'appel, la valeur correspondant à ce que nous voulons faire.

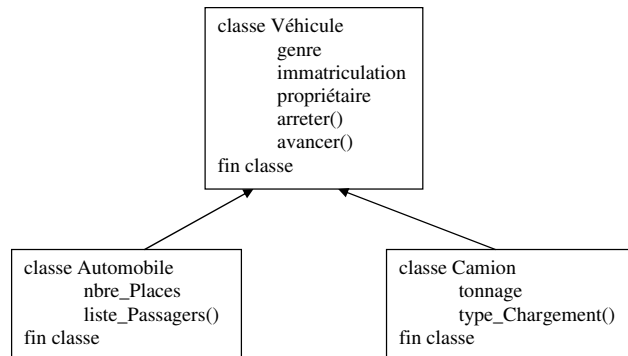
- **CLASSE** => modèle décrivant le contenu et le comportement des futurs objets de la classe, ensemble d'objets,
 - le contenu = les données,
 - le comportement = les méthodes,
- **INSTANCIATION** => fabrication à partir du modèle de la classe d'un objet particulier, élément de cette classe.
 - Objet = instance de la classe.

Programmation procédurale	VARIABLE (exemple: X)	TYPE (exemple: int)
Programmation orientée objet	OBJET	CLASSE

- Envoyer un message à un objet, c'est lui demander d'exécuter une de ses méthodes.

6.6. Héritage

- Construire une classe à partir d'une ou plusieurs autres classes.
- Exemple => Héritage simple



- Les classes Automobile et Camion héritent (ou dérivent) de la classe Véhicule.
- La classe dont on dérive est dite CLASSE DE BASE.
- Les classes obtenues par dérivation sont dites CLASSES DÉRIVÉES.
 - vision descendante => pouvoir reprendre intégralement tout ce qui a déjà été fait et pouvoir l'enrichir.
 - vision ascendante => possibilité de regrouper en un seul endroit ce qui est commun à plusieurs.
- L'héritage s'utilise dans les deux sens:
 - vers le haut (en analyse O.O.) => on regroupe dans une classe ce qui est commun à plusieurs classes. Dans la classe Véhicule, on regroupe les caractéristiques communes aux Camions et aux Automobiles.
 - vers le bas (lors de la réutilisabilité) => la classe de base étant définie, on peut la reprendre intégralement pour construire la classe dérivée. La classe Véhicule étant définie, on peut la reprendre intégralement pour construire la classe Bicyclette.

6.7. Style de Programmation

- Quelles sont les entités qui interviennent dans mon problème?
- Programmation Orientée Objet est la programmation dans laquelle les programmes sont organisés comme des ensembles d'objets coopérant ensemble:
 - Chaque objet représente une instance d'une classe,
 - Les classes appartiennent à une hiérarchie suivant la relation d'héritage.
- La connaissance des objets que l'on manipule et la connaissance de leur fonctionnement sont indispensables avant l'utilisation de ces objets.

- Modélisation d'un logiciel de trafic routier
 - Les entités sont:
 - les feux tricolores,
 - les carrefours,
 - les véhicules,
 - les agents de la circulation.
 - Lorsqu'un feu tricolore passe au vert, il envoie cette connaissance (ce message) à l'agent posté au carrefour. L'agent prend une décision et en informe (envoi de messages) les chauffeurs des véhicules.
- Structure d'une application objet => flots de messages entre un certain nombre d'objets, les objets sont " presque " indépendants les uns des autres.
- Cette indépendance (l'une des grandes forces de l'approche O.O.) permet la réutilisation des objets par de nombreuses applications.
- Les objets sont plus stables que les spécifications qui définissent leurs interactions => les applications sont plus simples à écrire et à faire évoluer.

6.8. Quelques critiques sur la P.O.O.

- Impossible de prévoir toutes les actions possibles que l'on peut avoir à effectuer sur un objet
⇒ Solution: les sous-classes.
- Comment choisir les classes?
⇒ Il dépend de l'application en question (ou du domaine). Ce choix ne peut être réalisé qu'à partir d'une étude approfondie de ce domaine.

7. Un Processus d'Analyse Orientée Objet

- Dans cet ordre:
 1. Répertorier les entités du domaine du problème et leur comportement,
 2. En déduire les classes auxquelles ces entités appartiennent,
 3. Architecturer l'ensemble des classes regroupant les données ou procédures communes à certaines classes d'où elles héritent.

8. Langages de Programmation Orientée Objet

- Pour qu'un langage soit orienté objet
⇒ conforme aux 3 principes de base de l'orientation objet:
 1. Encapsulation des données
 2. Structure de Classes
 3. Héritage

Langages Programmation système	Langage Programmation Orientée Objet
BCPL (65 à 67)	
B (67 à 70)	Simula (67)
C (72)	Smalltalk (80)
K&R (Kernigham & Ritchie) (78) (2 ^e , 88)	C++ (82)
Ansi-C (publié en 1990)	Java (95)