

# Cours de base d'Ingénierie des applications objet. Introduction

Support de Cours  
Christophe Dony  
Université Montpellier-II

## Contenu du cours

- concepts de base de la programmation par objets avec Java.
- La mise en pratique des concepts objet avec Java (on parlera également de C++ et Smalltalk).
- la réutilisation et les architectures logicielles, frameworks, design patterns.
- La pratique des classes de base de la bibliothèque.
- Les principes de réalisation d'interfaces graphiques à base d'objets.

# 1 Importance des objets dans l'informatique logicielle

- Programmation généraliste : Simula (67), Smalltalk (72, 76, 80), C with classes(80), Objective-C(84), C++(84), Flavors(80), Clos (85), Eiffel (88), ADA9X(95), Object-Cobol(96), JAVA(96), Visual-Basic ....
- Méthodes d'analyse et de conception des logiciels et des systèmes d'information - Informatique de gestion
  - ...
  - OOD (Object Oriented Design) de BOOCH
  - Objectory de Jacobson
  - OMT (Object Modeling Technique)
  - UML : notation unifiée pour OMT, OOD et Objectory.
- Interfaces et applications graphiques - domaine initiateur.

Exemples : Smalltalk graphic library, JAva AWT, Java Swing, ...

- Génie logiciel.  
conception, environnements de développement évolués (Smalltalk-80), modularité, réutilisation, composants.
- programmation concurrente et distribuée :
  - programmation concurrente intégrée dans les langages à objets concurrents (processus Smalltalk, *threads* Java).
  - architectures spécialisées pour la réalisation d'applications distribuées utilisent la méthodologie objet : Corba, J2EE, Microsoft .NET.

## 2 Grandes idées relatives à la programmation par objets

### 2.1 Une vision différente de la réalisation des applications

L'approche objet suggère la décomposition modulaire d'un système informatique sur les sortes d'objets qui constituent le système et que le système va permettre de manipuler plutôt que sur les fonctionnalités qu'il va offrir.

Exemple 1 : Interfaces (exemple canonique), Les entités qui constituent le système.

Entités "visibles"

- fenêtres
- figures
- menus

- boutons

- icônes

### Entités cachées

- vues

- sous-vues

- contrôleurs de la souris et du clavier

- etc.

### Exemple 2 : Un système de gestion d'une entreprise:

- clients

- produits

- livraisons

- factures

- fournisseurs

- dates
- etc

## 2.2 De nouvelles techniques de programmation

Un ensemble de techniques de programmation (Abstraction de données, abstraction procédurale, surcharge des noms d'opération, polymorphisme d'inclusion) fournissant des mécanismes d'abstraction et rendant les programmes plus facilement extensibles et réutilisables.

Définitions:

- **Extensible:** programme auquel il est possible d'ajouter de nouvelles fonctionnalités ou de nouvelles sortes de données sans modifier (ou en minimisant les modifications) ce qui est déjà écrit.
- **Réutilisable:** programme pour lequel il est possible d'adapter une fonctionnalité existante à une nouvelle sorte de données sans modifier le code existant.

## 2.3 Une intuition de la programmation par objets

### 2.3.1 Programmation fonctionnelle

programme ==	ensemble de fonctions
	ensemble de données

### 2.3.2 programmation par objets

**objet** : **encapsulation** d'un ensemble de données et d'un ensemble de fonctions spécialisées pour manipuler ces données.

exemple : Objet compte bancaire

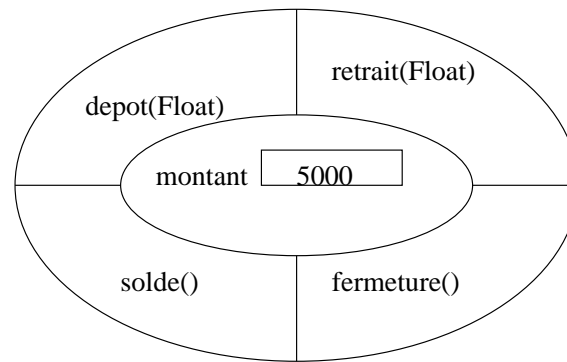


Figure 1: Objet : Encapsulation données-fonctions

**programme** : ensemble d'objets communiquant en s'envoyant des messages.

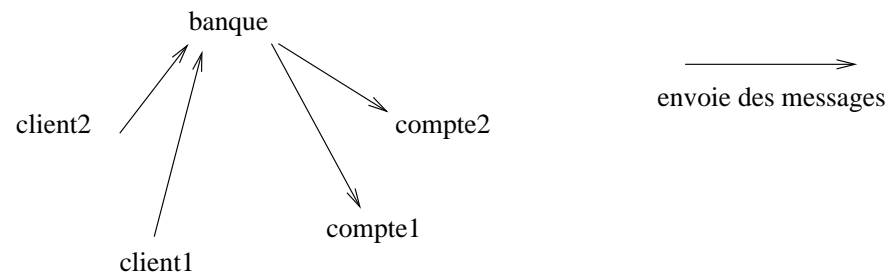


Figure 2: programme en exécution

## 2.4 Qualités de l'approche objet

- Stabilité de la conception.

les objets manipulés changent moins que les fonctionnalités offertes

Les fonctionnalités peuvent être modifiées, ou de nouvelles être ajoutées sans que le modèle sous-jacent n'ait à être modifié.

- Gestion de la complexité et de la taille des logiciels.

La spécification séparée des objets du système induit une décomposition modulaire et compréhensible du monde.

- Extensibilité : ajout simple de nouvelles sortes d'objets.

- Réutilisabilité : réutilisation de fonctionnalités existantes pour de nouvelles sortes d'objets.

## 3 Histoire

### 3.1 Emergence - 65-80

- Simuler des systèmes

Le but visé est faire fonctionner un programme non pas comme un algorithme mais comme l'animation d'un modèle réduit. Ce modèle est construit à partir d'entités informatiques qui reproduisent fidèlement les entités du monde réel. On appellera ces entités des objets. [Perrot96]

Simula

- Représenter des connaissances symboliques - Intelligence Artificielle.

Lisp - Flavors

- Traiter informatiquement et simplement toutes sortes d'entités.

Smalltalk

### **3.2 Intérêts pour le génie logiciel : 80-90**

Découverte par la R-D puis par les gros industriels des vertus de la programmation par objets.

Développement des langages : C++, Clos, Eiffel, Smalltalk, ...

### **3.3 Diffusion généralisée : 1990 - 2010**

Les modèles à objets gagnent la plupart des champs du domaine logiciel.

## 4 grands principes de conception d'applications

Exemple d'énoncé : Logiciel de gestion des inscriptions pour un club sportif. Les inscrits peuvent être des "travailleurs" ou des "scolaires". Une inscription lie, en respectant les contraintes de dates et de nombre, un inscrit à une activité, à une certaine date, avec un certain moniteur.

### 4.1 Analyse : Recensement des principales sortes d'objets du système à modéliser.

Définir un modèle conceptuel du domaine concerné i.e. décomposer le domaine en un ensemble de concepts individuels ou sortes d'objets.

Travail très similaire à celui effectué lors de la conception d'un système d'information.

Exemple : *Club, Activité, membre, employé, planning, periode, date, heure.*

A cette étape, on ne recense aucune sorte d'objet informatique.

## 4.2 Recensement des caractéristiques de chaque sorte d'objets

**Caractéristique d'une sorte d'objets** : les différentes sortes de relations qu'elle possède avec les autres sortes d'objets.

- **Relation d'association** : relation spécifiant qu'un l'objet est associé, est sémantiquement connecté, à un **autre** objet indépendant.

Exemples :

Une entreprise emploie des personnes.

CLubSportif

“est dirigé par” un responsable : (de type) employe

“emploie” des moniteurs : collection d'employes

“gère” des membres : collection de personnes

“utilise” un lieu\_d'activité : salle

Les relations peuvent avoir un nom (*emploi*), des rôles (*employé - employeur*), une multiplicité (un employeur emploie plusieurs employés), etc.

- **Relations d'Agrégation et de Composition** : relations binaires entre un tout et ses parties, les parties étant des éléments constitutifs du tout.

- **Relation de Composition** : le mot composition est utilisé lorsque la partie n'a pas d'existence possible sans son tout : une personne possède deux mains. Représentation graphique : losange noir.

- **Relation d'agrégation** : le cas opposé. Une voiture possède un moteur. losange blanc.

### 4.3 Définition du savoir-faire d'une sorte d'objets

**Savoir Faire** : Ensemble des messages qu'un objet comprend et pour lesquels il sait exécuter un comportement.

**Comportement** : Propriété fonctionnelle d'une sorte d'objets.

Exemples:

Un club sait donner son emploi du temps général, inscrire une nouvelle personne (objets inanimés avez vous donc une âme ...)

Une activité sait dire quel est son responsable, sa plage horaire et son nombre d'inscrits.

Une liste sait insérer ou supprimer un nouvel élément.

Une période de temps sait dire si elle contient une certaine date.

#### **4.4 Prise en compte de la relation de spécialisation**

Exemples :

Un club est-une sorte d'association.

Un membre est-une sorte de personne.

Un moniteur est-une sorte de personne.

## 5 Les grands principes de la réalisation d'applications

Réalisation informatique d'une spécification de logiciel basée autour d'un modèle conceptuel objet.

### 5.1 Sortes d'objets et Classes

Une classe est un moyen de représenter dans un programme la notion de sorte d'objets.

La plupart des langages à objets sont des langages à classes

**Classe** : entité

- représentant un ensemble d'objets ayant les mêmes caractéristiques et les mêmes comportements,
- définissant leur structure interne,
- définissant et détenant ces comportements,

- capable de générer des instances.

## 5.2 Représentation des relations

Les relations d'association, d'agrégation et de composition sont représentées par des (un ou plusieurs) attributs des classes.

Une relation d'association peut entraîner la création d'une classe. ex: personne emprunte livre (dans une bibliothèque) (classe Emprunt éventuelle)

Un attribut possède un nom, éventuellement un type et une valeur par défaut. Un attribut peut être **public** ou **privé** selon qu'il est accessible ou non.

Exemple :

Activite

nom

mixité

responsable

salle

creneaux : collection de periodes

### 5.3 Savoir-faire et Méthodes

**Méthode** (ou fonction membre). Fonction définie sur une classe, représentant un savoir-faire d'un objet.

Ex : La classe Club  
Attribut : nom : String, moniteurs : Collection d'employés ...  
Methode : inscrire-membre

Ex : La classe date  
Attribut : j, m, a : int  
Methode : j, jour, num-an, numJC

**Remarque : Type abstrait** En première approche, il existe une analogie totale entre classe et type abstrait, entre instance de la classe et élément d'un type.

Langages permettant de définir des type abstraits: Simula (classes),

Modula (modules), Ada (Packages), Clu (Clusters)

## 5.4 Les objets

- tout objet est instance d'une classe. La classe détermine la structure et les comportements des objets qui sont ses instances.
- un objet est une entité individuelle, repérée par une adresse unique.
- **champs** : un objet est une entité informatique, constituée d'un ensemble de champs. Il y a autant de champs que d'attributs déclarés par la classe.
- Chaque champs contient une valeur qui peut varier au cours du temps. L'ensemble de ces valeurs constitue à chaque instant **l'état courant de l'objet**.
- **Encapsulation** : l'état courant d'un objet n'est par défaut pas accessible par le monde extérieur à l'objet.

## 5.5 L'objet et ses comportements

- un comportement d'un objet est activé par un envoi de message.
- L'ensemble des méthodes définies par sa classe définit le comportement global d'un objet.
- un envoi de message comporte un receveur, un sélecteur et des arguments.
- Un message est une demande à un objet d'exécuter une de ses opérations ou procédure ou fonction. L'objet répond au message en exécutant la procédure dont le nom correspond au sélecteur du message.

exemples:

MUC.inscrire(eric, "Volley")

unRectangle.surface(),

unRectangle.volume() : (echec).

## **5.6** Calculs

L'exécution d'une application par objet se résume à une succession d'envois de messages entre objets.