Introduction à la Programmation Orientée Objet

Michael Mrissa

Département Informatique
UFR Sciences et Techniques
Université de Pau et des Pays de l'Adour

Note

- Ce support est une évolution de celui de Nicolas Belloir
- Il est inspiré des supports de
  - Jean-Michel Bruel (IUT Blagnac)
  - Marc Daniel (Ecole Supérieure d'Ingénieurs de Luminy)

*****

Préambule

Objectifs

Définition

Aujourd'hui, la programmation par objet est vue davantage comme un paradigme, le paradigme objet, que comme une simple technique de programmation. C'est pourquoi, lorsque l'on parle de nos jours de programmation par objets, on désigne avant tout la partie codage d'un modèle à objets obtenu par AOO et COO.

Source : `https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet`

- Éléments clés du développement OO
- Modèle objet, illustré en UML
- Mieux appréhender le C++ et/ou Java

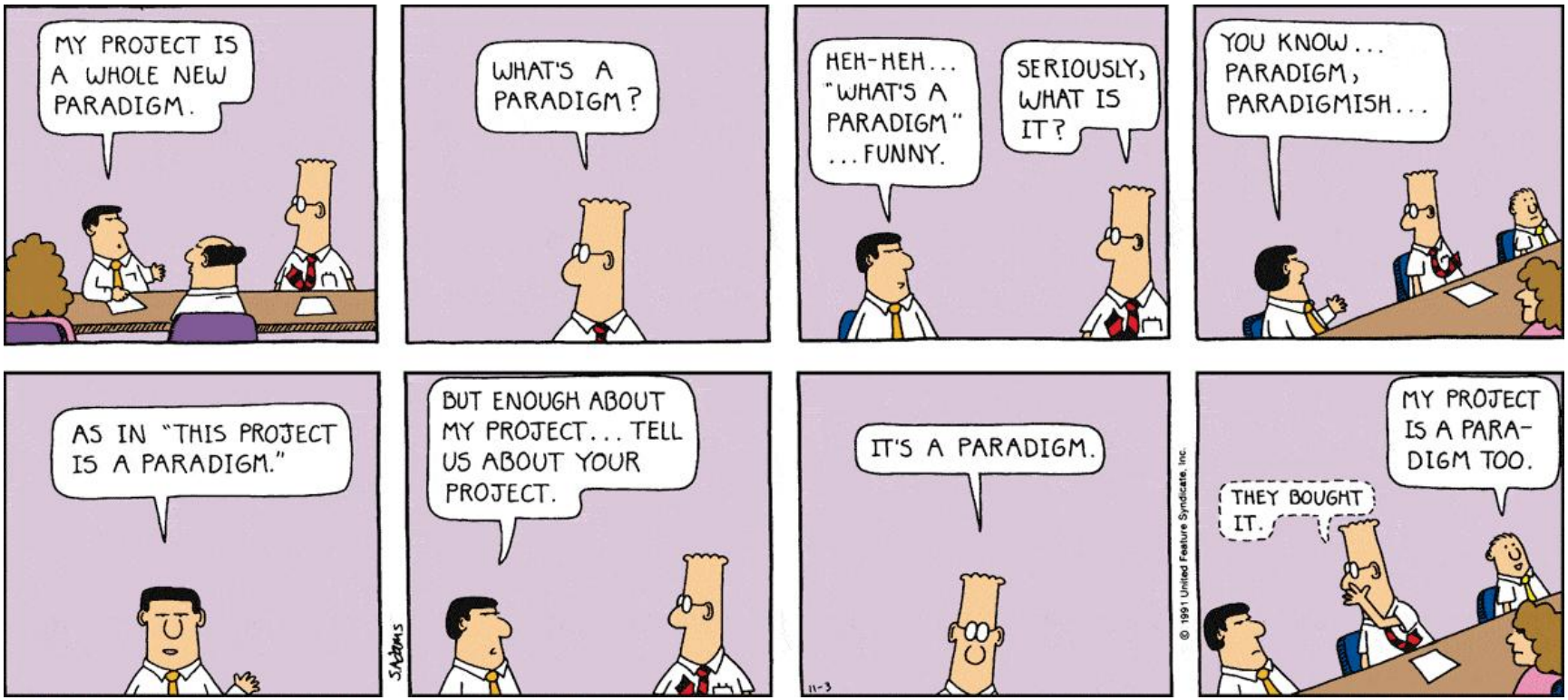

FIGURE – Paradigme : définition ?

Source : `http://dilbert.com/strip/1991-11-03`

Plan

- Le paradigme objet
  - Apprendre les concepts généraux
  - Communs et indépendants des langages
- Le langage C++
  - Pour pouvoir manipuler en TP
  - Syntaxe déjà connue (langage C)
  - Algorithmique connue
  - Uniquement une illustration des concepts

Me joindre

Contact

- adresse : Faculté des Sciences, Bât B3, 2e étage
- mail : michael.mrissa@univ-pau.fr
- téléphone : 05 59 40 76 41

Objectifs pédagogiques

Programme

- Introduction à la programmation orientée objet
  - Comprendre les principes fondamentaux
- Un langage de programmation : le langage C++
  - Savoir écrire des programmes selon le paradigme objet
  - Maîtriser les points techniques

Evaluation

- Examen : 50 %
- Contrôle continu : 50 % (partiel + contrôle TP + contrôle surprise ?)

Bibliographie

- "C++ pour les programmeurs", *Claude Delannoy*, Eyrolles, 2007
- "Conception et Programmation Objet", *Bertrand Meyer*, Eyrolles, 2000
- WebCampus

*****

Introduction

Historique (1/)

- Les débuts :
  - Les années 60 et le début des langages de programmation
- Les années 70-80
  - Analyse descendante
    - Décomposition en fonction des traitements
    - Les logiciels sont organisés autour des actions qui traitent les objets
  - Séparation des traitements et des données
    - Un programme = algorithme + données

Historique (1/)

- Les débuts :
  - Les années 60 et le début des langages de programmation
- Les années 70-80
  - Analyse descendante
    - Décomposition en fonction des traitements
    - Les logiciels sont organisés autour des actions qui traitent les objets
  - Séparation des traitements et des données
    - Un programme = algorithme + données

Historique (2/)

- Programmation et conception orientées objet
  - Naissance au milieu des années 70
  - Essor dans les années 85
  - Utilisation réelle à partir des années 90

La programmation procédurale

- La programmation procédurale est :
  - Organisée autour des actions qui traitent les objets
- Mais
  - Entrelacement entre les appels de procédures
  - Problème en cas de modification de la structure des données
  - Maintenance difficile

La programmation orientée objet

- Organisée autour des objets

citation

"La conception orientée objet repose sur une idée apparemment élémentaire. Les systèmes informatiques réalisent certaines actions sur certains objets. Pour obtenir des systèmes souples et réutilisables, il vaut mieux structurer le logiciel autour des objets que des actions". Bertrand Meyer.

La programmation orientée objet

- Organisée autour des objets

citation

"La conception orientée objet repose sur une idée apparemment élémentaire. Les systèmes informatiques réalisent certaines actions sur certains objets. Pour obtenir des systèmes souples et réutilisables, il vaut mieux structurer le logiciel autour des objets que des actions". Bertrand Meyer.

Objectif du génie logiciel 1/

- Permettre la réutilisation
  - Réutiliser directement (code, modèle ...)
  - Ou adapter des existants ($\rightarrow$héritage)
- Accroître la qualité
  - Rapidité de la conception
  - Facilité de la maintenance
  - Ergonomie d'utilisation
  - Fonctionnement simple et efficace

Objectif du génie logiciel 1/

- Permettre la réutilisation
  - Réutiliser directement (code, modèle ...)
  - Ou adapter des existants ($\rightarrow$héritage)
- Accroître la qualité
  - Rapidité de la conception
  - Facilité de la maintenance
  - Ergonomie d'utilisation
  - Fonctionnement simple et efficace

Objectif du génie logiciel 2/

- Caractéristiques d'un "bon" logiciel
  - Valide
  - Robuste
  - Modifiable
  - Réutilisable
  - Portable
  - Ergonomique
  - Efficace
  - Compatible avec son environnement

Objectif du génie logiciel 3/

- La facilité de la maintenance
  - Evolution du logiciel dans le temps
  - Correction des bugs

En conclusion

- Le logiciel doit être **modulaire**
- et les modules doivent être **indépendants**

Critères de modularité 1/

- Décomposition modulaire
  - Décomposer un problème en sous-problèmes dont les solutions peuvent être recherchées séparément
- Composition modulaire
  - La méthode fournit des éléments (modules) qui peuvent être combinés pour traiter de nouveaux problèmes
- Compréhension modulaire
  - La méthode fournit des modules compréhensibles indépendamment

Critères de modularité 1/

- Décomposition modulaire
  - Décomposer un problème en sous-problèmes dont les solutions peuvent être recherchées séparément
- Composition modulaire
  - La méthode fournit des éléments (modules) qui peuvent être combinés pour traiter de nouveaux problèmes
- Compréhension modulaire
  - La méthode fournit des modules compréhensibles indépendamment

Critères de modularité 1/

- Décomposition modulaire
  - Décomposer un problème en sous-problèmes dont les solutions peuvent être recherchées séparément
- Composition modulaire
  - La méthode fournit des éléments (modules) qui peuvent être combinés pour traiter de nouveaux problèmes
- Compréhension modulaire
  - La méthode fournit des modules compréhensibles indépendamment

Critères de modularité 2/

- Continuité modulaire
  - Une petite modification de la spécification n'amène des modifications que sur un ou quelques modules. Il ne doit pas y avoir d'impact sur l'architecture du système
- Protection modulaire
  - Pas de propagation de l'effet d'une condition anormale

Critères de modularité 2/

- Continuité modulaire
  - Une petite modification de la spécification n'amène des modifications que sur un ou quelques modules. Il ne doit pas y avoir d'impact sur l'architecture du système
- Protection modulaire
  - Pas de propagation de l'effet d'une condition anormale

Une société de modules respectant les critères de modularité

- Module : unité syntaxique du langage
- Un module doit communiquer avec peu de modules
- Les échanges entre modules doivent être le plus réduit possible
- Les échanges entre modules sont clairement explicités dans la définition des modules
- Toute information concernant un module est privée, sauf si elle est explicitement déclarée publique

Les objets

Un objet est un module qui

- Parle peu
- A une conversation qui tient en peu de mots
- A des échanges publics, codifiés (à haute voix)
- A des informations et des traitements privés

Première vision des objets 1/

- Décomposition en modules en fonction de leurs structures de données
- Un module est décrit comme une implémentation d'un type abstrait de données et des traitements associés
- Tout type est un module et vice-versa
- Chaque module a une interface (publique/privée) pour dialoguer avec les autres

Première vision des objets 2/

- Un module se définit comme une extension ou une restriction d'un autre module (*héritage*)
- Une même entité de programme peut faire référence à différents modules (*polymorphisme*).
- A l'exécution, une opération peut avoir des comportements différents dans des modules différents (*liaison dynamique*).

Programmation orientée objet 1/

- On s'intéresse aux objets avant de s'intéresser aux traitements
- La POO relève de la création d'objets et de l'exécution d'envois de messages entre les objets

Définir un objet

- Quels sont les objets du système ?
- Quels traitements doivent-ils subir ?
- Quels liens ont les objets entre eux ?

Programmation orientée objet 2/

Plusieurs points de vue possibles

Le plus important est l'utilisation de classes ou de langages de classes

Conclusion

Un Programme Orienté Objet est un ensemble d'objets autonomes et responsables qui s'entraident pour résoudre un problème final en s'envoyant des messages.

Langages objets

- Simula
  - années 70
  - compilé
  - années 70
  - compilé
- Smalltalk
  - années 72-80
  - interprété $-- >$ adapté au prototypage
  - aussi un système d'exploitation
- C++
  - année 1985
  - compilé
- Eiffel
  - 1989
  - resté au stade universitaire
- Java
  - année 1995
  - interprété
- Javascript, Objective C, Visual C++, Ruby, Python ...

*****

Objets et classes

L'encapsulation un principe fondateur

- On découple :
  - L'interface
    - Ce qu'on voit d'un objet
    - Ce qu'on manipule en tant qu'utilisateur
  - L'implémentation
    - Façon concrète de représenter l'objet
    - Données (attributs) et opérations (méthodes)

Les objets 1/

- Un objet représente une unité atomique formée :
  - d'une identité
  - d'un état
  - d'un comportement

FIGURE – l'objet *maFenetre* : un exemple

Les objets 2/

- Identité
  - Constante
  - Permet à un objet d'être manipulé, référencé par d'autres
- Etat
  - variable
  - conforme à un certain type (classe)
  - caractérise la "valeur" d'un objet à un instant donné
- Comportement
  - opérations que peut accomplir l'objet
  - invoqué via l'interface de l'objet
  - modifie ou non l'état

La communication entre objets

- Les services offerts par un objet sont déclenchés soit par cet objet soit par un autre objet.
- L'unité de communication entre objets s'appelle le message.
- C'est un concept abstrait qui peut être mis en œuvre de différentes façons.

FIGURE – Exemple de communication par message

Le concept de classe 1/

- grande variété d'objets dans le monde réel
- on ne définit pas les objets un par un on décrit le domaine de définition d'un ensemble d'objets dans **une classe**

Remarque

Une classe est un module

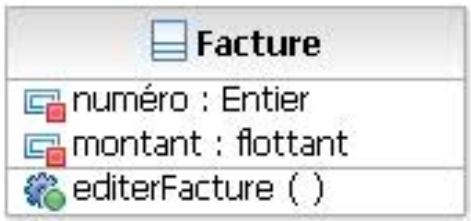

FIGURE – Exemple de classe

Le concept de classe 2/

- Une classe déclare les parties communes à un ensemble d'objets en définissant
  - Une structure de données
    - Partie structurelle composée de champs (appelés **attributs**)
  - Des traitements à effectuer sur ces structures
    - services disponibles sur la structure (appelés **méthodes**)
  - Une interface clairement identifiée définissant
    - La partie publique (ce que voit le client)
    - Les services auxquels on peut faire appel

Deux méthodes importantes : les constructeurs et destructeurs

- Lorsqu'un objet est instancié à partir d'une classe, une méthode particulière est automatiquement exécutée : **le constructeur**
  - rôle du constructeur : créer une instance de l'objet
- Lorsqu'un objet n'est plus utilisée, le **destructeur** de la classe est exécuté

Quelques particularités

- Attributs et méthodes de classe
  - Certains éléments d'une classe ne concernent que la classe elle-même
- Classe abstraite
  - Lorsqu'une classe ne peut pas avoir d'instances, on parle de classe abstraite.
- Surcharge d'opérateurs

*****

Les hiérarchies de classes

Généralisation et Spécialisation 1/

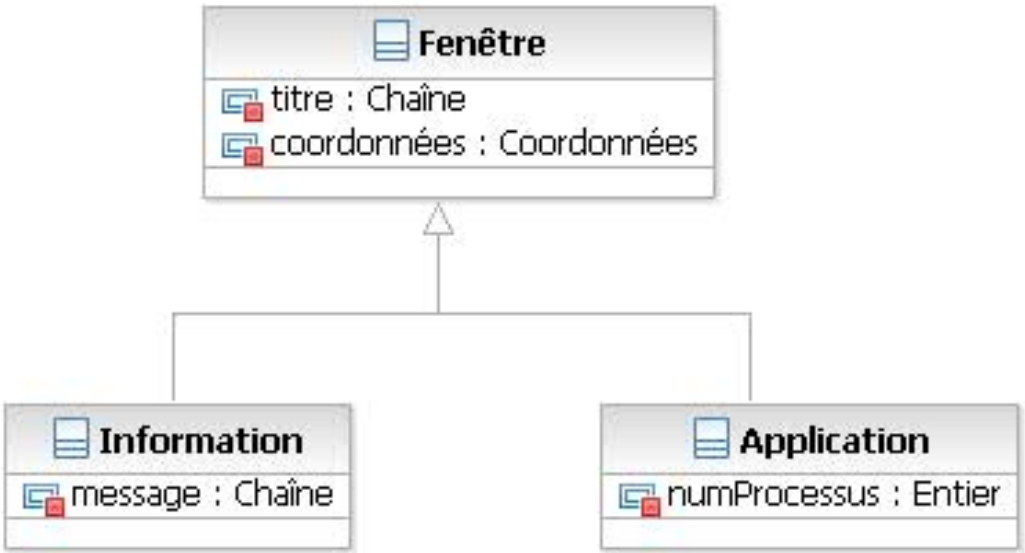

FIGURE – Généralisation et Spécialisation

Généralisation et Spécialisation 2/

- Héritage
  - Définir une classe à partir d'une autre
    - La classe enfant hérite de la totalité de la classe parente
  - Réutiliser
  - Une classe dérivée peut se différencier de la classe de base par :
    - ajout de méthodes
    - ajout de données membres
    - redéfinition de méthodes héritées de la classe de base
  - Possibilité d'héritage multiple

Généralisation et Spécialisation 3/

- Polymorphisme
  - Un objet peut avoir plusieurs types
  - Différents types de polymorphismes

Les relations entre classes 1/

Définition

Les liens qui relient les objets entre eux sont représentés par des relations entre classes.

- L'association

FIGURE – Association

Les relations entre classes 2/

- L'agrégation

FIGURE – Agrégation