Programmation Objet Avancée

Jonathan Caux
jcaux@adobe.com

Contexte du cours

Qui suis-je

- Jonathan Caux
- ZZ2 promo 2008
- Thèse au LIMOS soutenu en 2012
  - Cours C++
  - Cours GPGPU
- Depuis principalement dév’ C++
  - Murex, progiciel de finance (dév’ C++)
  - AVM Up, téléphonie (dév’ C++)
  - Thales service, ESN du groupe Thales (tech lead C++)

Adobe 3D&I

- Senior dev’ à Clermont depuis 2021 chez Adobe
- Division 3D&I (3D and Immersive) issue du rachat d’Allegorithmic
  - Allegorithmic, entreprise de texturing 3D créée à Clermont et d’abord hébergée à l’ISIMA
- Gamme de produit autour du texturing 3D
  - Designer, Painter (Clermont)
  - Sampler (Lyon)
  - Pleins d’autres logiciels et outils, ici et ailleurs

Adobe Substance 3D Designer

- Équipe (au sens large) de 11 personnes :
  - 1 tech lead + 4 dev’ (dont 2 seniors) + 1 recrutement en cours
  - 2 PM (plus ou moins PO/scrum master)
  - 2 ingé’ qualité
  - 1 UX/UI designer (partagé)

- Logiciel de création de texture 3D
  - C++ / Qt / Google Test
  - CMake / git / Jenkins

Contenu du cours

- C++ car largement où je suis le plus à l'aise
- Base de l'objet + Design Pattern
  - => Objectif de :
    - Comprendre quand on tombe sur un design pattern
    - Réaliser de bons designs objet
    - Surtout pas de mettre des design pattern partout
- N'hésitez pas à remonter vos remarques
- Loin de tout savoir, contenu très largement inspiré de ce que j'ai pu lire : Tête la première Design Patterns, Code Complete, …
- Évaluation : TP, examen écrit et projet

Origine des design patterns

Design Patterns

- Patron de conception en Français
- Historiquement formalisés de manière généraliste pour la première fois par le « Gang of Four » dans le livre « Design Patterns – Elements of Reusable Object-Oriented Software » (1994)
  - Contient la plupart des designs patterns les plus courant
  - Catégories (GoF) : créateurs (creational), structuraux (structural), comportementaux (behaviour)
  - Exception faite des patterns liés au web (et encore plus au cloud)

Design Patterns

- Solutions éprouvées à des problèmes courants
- Best-practice « clé en main » réutilisable
  - Jamais la seule solution, jamais certain d’être la meilleure solution et parfois la mauvaise solution
  - Inspiration au minimum
  - Vocabulaire à connaitre dans tous les cas
- Parfaitement intégré à l’orienté objet
- Certains redondant/dépréciés avec les langages/framework modernes

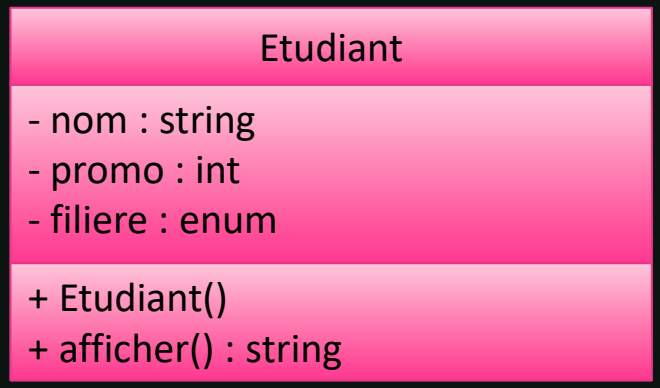
Bases de la conception objet

Quelques définitions autour de l’objet

- Objet
  - Instance d’une classe
  - Entité cohérente rassemblant des données et les fonctions traitant ces données
  - Données <=> attributs; fonctions <=> méthodes
- Classe
  - Fabrique à objets (description des objets)
  - Représente une catégorie d'objets

Quelques définitions autour de l'objet

Classe



Instanciation

Instances

nom = "Jonathan"
promo = 2008
filiere = F2

nom = "Daniel"
promo = 2013
filiere = F4

Quelques définitions autour de l’objet

- Attributs et méthodes d’instance ou de classe
- Attributs d’instance : données propres à une instance
- Attributs de classe : données communes à toutes les instances d’une même classe
- Méthodes d’instance : traites les données d’un object (attributs d’instance)
- Méthodes de classe : traites les données d’une même classe (attributs de classe)

Les concepts de base de l’idée d’orienté objet

- Abstraction : cacher les détails d’implémentation
- Encapsulation : combiner données et méthodes pour protéger les données
- Héritage : créer des classes basées sur d’autres
- Polymorphisme : traiter une classe fille comme si elle était une classe mère

Abstraction

- Au niveau du design de la classe
- Choix de ce que l'on met en visibilité de l'utilisateur de la classe
- Abstraction des données : on cache les données sous-jacentes
- Abstraction des fonctions : on cache une partie de l'implémentation
- Rend les modifications de ce qui est abstrait transparentes pour l'utilisateur
- Permet de se concentrer sur une interface et non des détails d'implémentation

- Note : différent de classes ou méthode abstraites

Encapsulation

- Au niveau de l’implémentation de la classe
- Regrouper en un tout cohérant les données et les actions réalisées dessus
- Créer une interface claire et limitée pour traiter un concept

- Va de pair avec l’abstraction
- Bien utiliser, permet de simplifier la compréhension et l’évolution du code

Héritage

- Notion de généralisation et spécialisation de classes
- Classe-mère / super-classe / classe générale
- Classe-fille / classe dérivée / sous-classe / classe spécialisée
- La classe fille hérite de toutes propriétés de la classe mère et lui rajoute les siennes

- Limite la duplication du code
- Outil roi des années 1990-2010, maintenant avec des détracteurs
- Puissant bien utilisé, problématique mal utilisé

Polymorphisme

- Polymorphisme de classe
- Permet de manipuler une instance d’une classe fille à travers une vue de la classe mère

```
MaClasseMere* ptrClasseMere = new MaClasseFille();
```

- Très important pour l’abstraction mais pas toujours évident

- Différent du polymorphisme de méthodes
- Permet à une classe fille de modifier une partie du comportement hérité de sa classe mère

Concept objet – principes directeurs

- Principes de haut niveau à garder en tête
- Encapsuler ce qui varie
- Préférer la composition à l’héritage
- Programmez vis-à-vis d’interfaces, pas d’implémentation

Concept objet - SOLID

- Single-responsiblity
  - Une classe, une responsabilité
- Open-closed
  - Ouvert à l’extension, fermé aux modifications
- Liskov Substitution
  - Toujours possible d’utiliser une instance de classe fille à travers l’interface de la classe mère sans connaitre l’existence de la classe fille
- Interface Segregation
  - Interface limitée au minimum de façon à limiter la dépendance au minimum
- Dependency Inversion
  - Déprendre d’interface, pas d’implémentation

Concept objet – et les autres

- DRY : Don’t Repeat Yourself
- KISS : Keep It Simple Stupid
- Et bien d’autres

Fin 1er cours