conception orientée objet

Après l'analyse et avant le développement

- L'objectif de cette présentation est
  - de faire un rappel sur les connaissances acquises sur la modélisation objet.
  - d'introduire la notion de conception orientée objet par rapport à l'analyse orientée objet.
  - d'approfondir le concept d'interface.
  - d'établir les deux grands principes de la conception orientée objet
    - Programmation générique plutôt que programmation spécifique
    - Réutiliser par la composition plutôt que par l'héritage
  - d'avoir une première approche des Design Patterns.

Analyse et conception

- L'analyse consiste sur la base d'un cahier de charge de modéliser les acteurs et le système en ne s'intéressant pas à l'implémentation.
  On parle d'approche Métier.

QUI (acteurs) fait QUOI (système)

- La conception consiste sur la base de l'analyse à modéliser la réalisation informatique.
  On parle d'approche informatique :

COMMENT ?

Rappels sur l'approche Objet

- Un objet réunit des **données** (attributs) et des **opérations** (méthodes) qui opèrent sur ces données.
- Un objet réalise une opération lorsqu'il reçoit une **requête** (ou message) de la part d'un client.
- *Les* **requêtes** sont les seuls moyens de faire exécuter une opération. Les **opérations** sont les seuls moyens de modifier les données internes d'un objet.

Les trois grands principes de l'approche Objet

- L'encapsulation : la structure interne d'un objet n'est pas connu de l'extérieur (intégrité ➔garantie d'un bon développement).
- L'héritage : une classe dite dérivée contient les attributs et opérations d'une classe existante (une façon de réutiliser).
- Le polymorphisme est la possibilité d'envoyer une requête à un objet sans connaître de type de l'objet.

Généralités sur la conception orientée objet

- La tâche essentielle et difficile de la conception orientée objet est la **décomposition d'un système en objets**.

Généralités sur la conception orientée objet

- La difficulté tient à plusieurs facteurs conflictuels :
  - **Encapsulation** : un objet n'est accessible de l'extérieur qu'à travers son interface.
  - **Granularité :** les objets sont plus ou moins fins
  - **Dépendance :** les objets doivent communiquer entre eux
  - **Performance :** l'exécution doit être la plus rapide possible
  - **Réutilisabilité :** On doit pouvoir réutiliser des objets dans des programmes différents

Généralités sur la conception orientée objet

- Des objets peuvent varier en taille. Un objet représentant une **application** est plus gros qu'un objet qui représente **un élément de bas niveau comme du matériel (souris, clavier) ou un conteneur (liste, vecteur, arbre)**.

- **Comment déterminer la granularité d'un objet et par la même quelles sont ses responsabilités ?**

Généralités sur l'approche objet

- Des objets peuvent représenter des entités du monde réel comme des étudiants mais d'autres n'existent pas dans la réalité comme des processus.

- La modélisation orientée objet permet à travers une même approche (celle de l'objet) de tout représenter. L'abstraction est donc obligatoire.

Opérations et Interface d'un objet

- La déclaration d'une opération d'un objet est constituée
  - du nom de l'opération
  - ses paramètres reçus
  - du retour

  Cet ensemble est la **signature de l'opération**.
- L'ensemble des signatures des **opérations publiques** d'un objet est appelée **interface de l'objet**.

Interfaces et Interface d'un objet

- Une requête d'un client à un objet x conforme à une signature de l'interface de x peut être envoyée à x.
- Rappel : Une interface est un ensemble de déclarations d'opérations publiques. C'est un ensemble de services.
- Rappel : Un classe réalise une interface **I** quand elle implémente toutes les opérations déclarées dans **I**.

Interfaces et Interface d'un objet

- Si un objet réalise une interface **I** alors **I** est contenu dans l'interface de l'objet.
- Un objet **O** peut contenir plusieurs **interfaces**. Deux clients peuvent utiliser différemment l'objet **O**.
  - Le **client simulateur de course** de voitures a besoin de connaître les services de fonctionnement d'une voiture.
  - Le **client réparateur de voiture** a besoins de connaître les services de diagnostics de pannes et d'accès aux pièces de la voiture.

Interfaces et Interface d'un objet

- Des objets de types différents peuvent avoir **une ou plusieurs interfaces** en commun.
  - Un simulateur de vol d'avions peut faire voler des F18 ou des canards Colvert du moment que ces objets fournissent les services demandés par les simulateur.
  - L'ensemble des déclarations de ces services sera une interface commune entre un F18 et un canard Colvert.
  - Les objets de type F18 et de type CanardColvert réalisent cette interface.

Implémentation du Polymorphisme

- Quand un client envoie une requête à un objet x, il a simplement besoin de savoir que cet objet peut recevoir cette requête.
- Les langages orientés objets comme le C++, Java et d'autres possèdent un mécanisme dit de liaison dynamique :
- **L'opération à exécuter suite à une requête envoyée à un objet n'est connue qu'à l'exécution**.

Implémentation du Polymorphisme

- Si le client utilise les services d'un objet O et que ces services sont définis dans l'interface I alors l'objet O peut être déclaré de type I.
- Les conséquences sont :
  - l'interface de O contient I.
  - La classe de O doit réaliser l'interface I
  - Le client ne connaît pas le vrai type de O.

1er principe de la conception orientée objet

- En résumé : Le client peut ignorer les classes d'objets qu'il utilise pour peu que les objets contiennent les interfaces utilisées par le client. Cela nous amène au premier principe de la conception orientée objet.

Programmer pour une interface, non pour un développement particulier

Classe abstraite

- Une classe abstraite A est une classe dont certaines opérations sont déclarées et non définies (comme pour les interfaces)
- On ne peut pas instancier des classes abstraites
- Les classes concrètes qui dérivent de la classe A doivent définir les méthodes abstraites déclarées dans A.

Classe abstraite et polymorphisme

- Dans certains cas, une requête peut connaître un objet à travers une référence d'une classe abstraite A.

- Il faut que la classe de l'objet dérive de la classe abstraite et que l'opération soit contenue dans l'interface de la classe A.

Concevoir pour mieux réutiliser

- Deux techniques :
  - L'héritage : réutilisation boîte blanche. On parle de boîte blanche, car le contenu des classes parentes est visible aux sous classes
  - La composition : réutilisation boîte noire . On parle de boîte noire, car la nouvelle classes n'a pas accès à la représentation interne de l'objet réutilisé.

L’héritage (1)

- **L'héritage et ses principes**
  - Il est défini de façon statique à la compilation
  - L'utilisation dans la sous classe est immédiate
  - La sous classe peut surcharger quelques opérations tout en gardant l'accès à la version de la classe Parent.
  - La sous classe dépend des changements de la classe Parent. L'héritage rompt l'encapsulation.
  - Impossibilité de modifier à l'exécution le code hérité des parents

L'héritage (2)

- **Réutilisabilité : comment le faire par héritage**

Class AC	Class NC extends AC
{	{
…	…
public void f1(…)	public void g1(…)
public void f2(…)	public void g2(…)
}	}

- **NC ref = new NC(…)**
- **ref.f1(); // la fille réutilise les opérations de la classe AC**

La composition (1)

- **La composition et ses principes**
  - Notation : le nouvel objet sera appelé objet composé.

  La composition est définie dynamiquement à l'exécution
  - L'accès des objets (composants) entrant dans la composition se fait par l'intermédiaire d'une de leurs interfaces.
  - L'encapsulation des composants est totalement respectée
  - A l'exécution, chaque composant peut être remplacé par un autre composant pourvu que le nouveau composant contienne l'interface utilisée.
  - Il faut que les composants contiennent les interfaces utilisées dans l'objet composé.

La composition (2)

```
Class AC
{
…
public void f1(…)
}
```

```
Class NC
{
AC oac
Public NC() {
   oac = new AC();
}
public void f1(…)
   {
   oac.f1(…)
   }
public void g1(…)
}
```

NC ref = new NC(…)

ref.f1(); // ref réutilise les opérations de la classe AC

2ème principe de la conception orientée objet

Pour la réutilisabilité, savoir différencier l'héritage sémantique de l'héritage fonctionnel

S'il s'agit uniquement d'un héritage fonctionnel , il faut impérativement utiliser la composition.

Généralités sur les designs patterns

- Un **design pattern** (ou modèle de conception) est un problème de conception récurrent accompagné de sa solution.

- Il est défini par :
  - un **nom** qui l'identifie sans ambiguïté
  - l'**énoncé du problème** récurrent
  - la **formalisation de sa solution** en UML et éventuellement dans tel ou tel langage orienté objet.

Généralités sur les designs patterns (2)

- Parmi les équipes les plus connues dans l'étude des designs patterns, nous avons l'équipe de GoF (gang of four) composée de
  - **Erich Gamma**
  - **Richard Helm**
  - **Ralph Johnson**
  - **John Vlissides**
- Dans un premier temps, nous étudierons les designs patterns de **GoF**.

Composition d'un Design Pattern

- De son **intention** : ce qu'il fait, son but,
- Sa **motivation** : un scénario qui illustre un cas de conception
- Ses **constituants** : les différentes classes (abstraites et concrètes), interfaces et objets utilisés
- Sa **structure** : sa solution quasiment tout le temps exprimée en UML.
- Ses **collaborations** : comment les instances communiquent entre elles.

Les familles de patterns

- Les créateurs : comment bien créer des objets ?
- Les structuraux : comment bien structurer un ensemble d'objets dépendants ?
- Les comportementaux : comment répartir au mieux les rôles entre les différents objets ?