Les Design Patterns

Reda Bendraou
reda.bendraou{{@}}Lip6.fr
http://pagesperso-systeme.lip6.fr/Reda.Bendraou/

Le contenu de ce support de cours a été influencé par les lectures citées à la fin de ce support.

Les Design Patterns

”Patterns expose knowledge about software construction that has been gained by many experts over many years. All work on patterns should therefore focus on making this precious resource widely available. Every software developer should be able to use patterns effectively when building software systems. When this is achieved, we will be able to celebrate the human intelligence that patterns reflect, both in each individual pattern and in all patterns in their entirety. ”

From ”Pattern oriented software architecture” by Buschmann et al.

Origines

- Cristopher Alexander et al.: A Pattern Language, 1977
- Cristopher Alexander: The Timeless Way of Building, 1979

Origines

Proposition d’Alexander dans le domaine de l’architecture:

- Description d'un problème récurrent et de sa solution
- Synonymes connus: Forme de conception, pattern, modèle, patron de conception, motif, etc.

Pattern: Définition

« Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice »

C. Alexander, "*The Timeless Way of Building*", 1979

- Ou tout simplement: une solution générale pour un problème récurrent dans un contexte donné

Pourquoi les Design Patterns

- Rendre disponible et explicite des pratiques de bonne conception
  - Capturer un savoir faire, le rendre pérenne réutilisable, etc.
- Nommer et rendre explicite une structure de haut niveau qui n'est pas directement exprimable sous forme de code
- Créer un vocabulaire commun pour les développeurs et les concepteurs

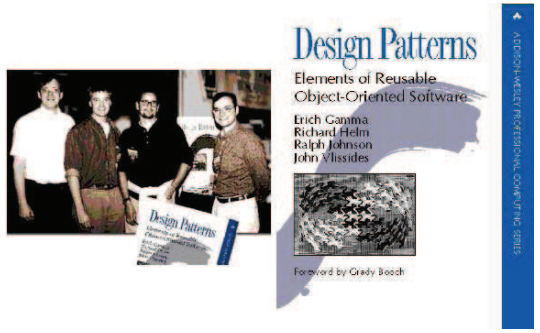
Pourquoi pas la même chose en Informatique?

Naissance – Historique des Software Design Patterns

- 1987 Ward Cunningham and Kent Beck: "Using Pattern
- Languages for Object Oriented Programming"
  - 5 pattern language for Smalltalk GUIs
  - future expectation: 100-150 patterns could cover OO programming!
- 1990- 1993 OOPSLA workshops, ideas developed
- 1993 The Hillside Group
- 1994 Start of PLoP conferences (pattern reviews), GoF book
- 1995 the first PLoP book
- 1996 A system of Patterns, Buchmann et. al.)

Design Patterns en Informatique: GoF

- **Un catalogue de 23 Patterns**



Documentation d'un Pattern selon le GoF

- Nom et classification
- Intention
- Autres noms connus
- Motivation (scénario)
- Applicabilité
- Structure (OMT)
- Participants (classes...)
- Collaborations
- Conséquences
- Implantation
- Exemple de code
- Usages connus
- Patterns associés

Remarque

Important

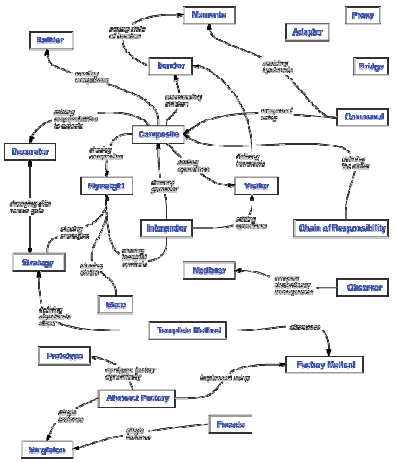
- La référence "Design Patterns" par le GoF décrit les formes par des diagrammes OMT...

- Mais l'héritage et les objets ne sont pas nécessaires

GoF Design Pattern: Classification

	PURPOSE		
SCOPE	**CREATIONAL**	**STRUCTURAL**	**BEHAVIOURAL**
CLASS	Factory Method	Adapter (class)	Interpreter
			Template Method
OBJECT	Abstract Factory	Adapter (object)	Command.
	Builder	Bridge	Iterator
	Prototype	Composite	Mediator
	Singleton	Decorator	Memento
		Facade	Observer
		Flyweight	State
		Proxy	Strategy
			Visitor
			Chain Of Resp.

Les Patterns sont liés!



Les Design Patterns Créateurs

Patterns Créateurs: Objectif

- Abstraire le processus d'instanciation
- Rendre indépendant de la façon dont les objets sont créés, composés, assemblés, représentés
- Encapsuler la connaissance de la classe concrète qui instancie
- Cacher ce qui est créé, qui crée, comment et quand

Patterns Créateurs: Exemples

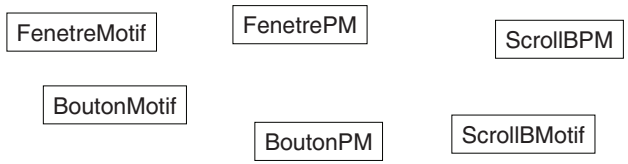
-Abstract Factory
-Singleton

Abstract Factory

- But
  - permettre de créer des familles de produits
  - en masquant les mécanismes de choix des classes de mise en œuvre de ces produits
- Exemple
  - création d'une interface homme-machine indépendante de la plate-forme
  - Le fameux jeux du Labyrinthe enchanté

Exemple : interface homme-machine (IHM)

- On veut développer une application graphique multiplateformes
  - il existe une bibliothèque graphique pour chaque système
  - d'une plate-forme à l'autre les classes d'IHM sont différentes
  - les plate-formes sont Windows™, MacOS™, Linux, Solaris™



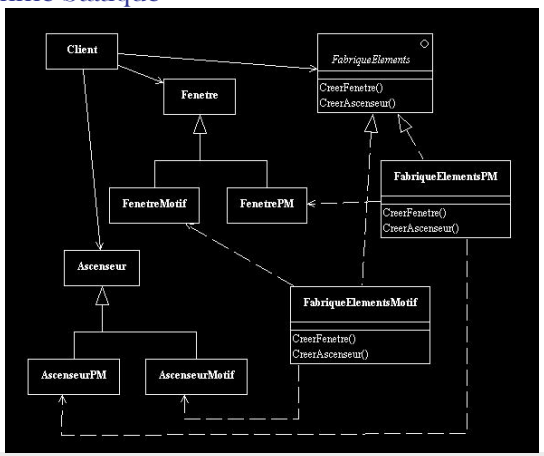
Exemple : interface homme-machine (IHM)

Solutions possibles

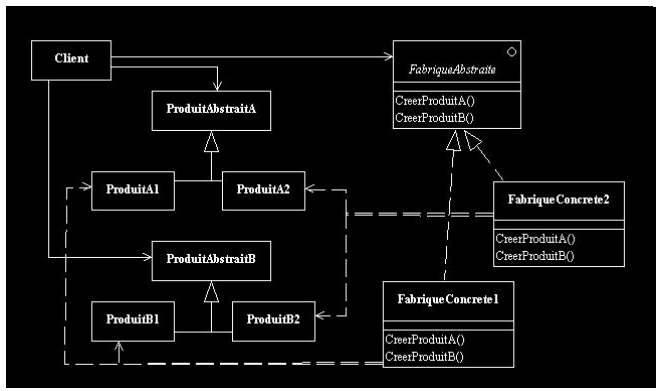
- Quatre applications différentes
  - quatre sources qui vont vite diverger
- Un seul source
  - avec des si alors sinon
  - avec des #ifdef #endif
- Emploi de Abstract Factory

Exemple interface homme-machine (IHM): Solution

- Diagramme Statique



Abstract Factory: Structure Générale



Abstract Factory: Rôle de client

- Client
  - détient une référence sur une Abstract factory
  - crée des produits par appel des opérations de cette référence
  - ne connaît pas la classe concrète des produits

Abstract Factory: Rôle de Abstract Product

- Masquer la classe concrète
- Offrir un ensemble d'opérations applicables à toutes les variantes d'un même produit

Abstract Factory: Rôle de Abstract Factory

- Comporte une opération de création
  - pour chaque produit, une opération de création retourne un objet produit
- La classe concrète des produits est masquée

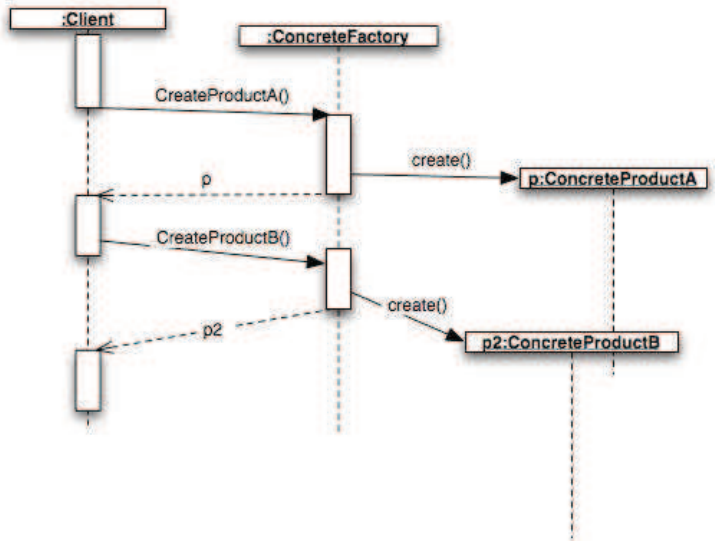
Abstract Factory: Rôle de Concrete Product

- Contient la mise en œuvre spécifique des opérations
- Non accessible au client
- Peut être amené à jouer un rôle d'adaptateur

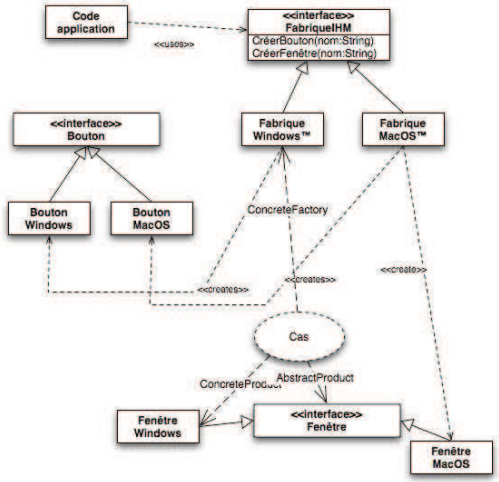
Abstract Factory: Rôle de Concrete Factory

- Chargée de mettre en œuvre la création des produits concrets
- Une fabrique concrète pour une plate-forme/variante/version donnée ne fait que des produits concrets de la même plate-forme/variante/version

Abstract Factory: Diagramme de séquence générique



Exemple de l'IHM



Abstract Factory: Quand l'utiliser?

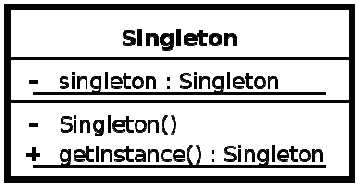
- un système doit être indépendant de la façon dont ses produits sont créés, assemblés, représentés
- un système repose sur un produit d'une famille de produits
- une famille de produits doit être utilisée ensemble, pour renforcer cette contrainte
- on veut définir une interface unique à une famille de produits concrets

Singleton

Quand l'utiliser?

- Quand il n'y a qu'une unique instance d'une classe et qu'elle doit être accessible de manière connue
- Lorsqu'une instance unique peut être sous-classée et que les clients peuvent référencer cette extension sans avoir à modifier leur code

Singleton: Solution (Structure)



Singleton: Solution (Code)

```
public class Singleton
{
private static Singleton singleton = null;

// Le constructeur en privé pour interdire l'instanciation de classe de
    //l'extérieur
private Singleton() {}
// On passera par cette méthode pour instancier la classe

public static Singleton getInstance()
{
    if (theInstance == null)
        theInstance = new Singleton();
    return theInstance;
}
}
```


Patterns Créateurs: Les autres Patterns

- Builder: Factory for building complex objects incrementally
- Factory Method: Lets a class defer instantiation to subclasses
- Prototype: Factory for cloning new instances from a prototype

Les Design Patterns Comportementaux

Patterns Comportementaux: Objectifs

Décrire:

- des algorithmes
- des comportements entre objets
- des formes de communication entre objets

Patterns Comportementaux: Exemple

- Observer

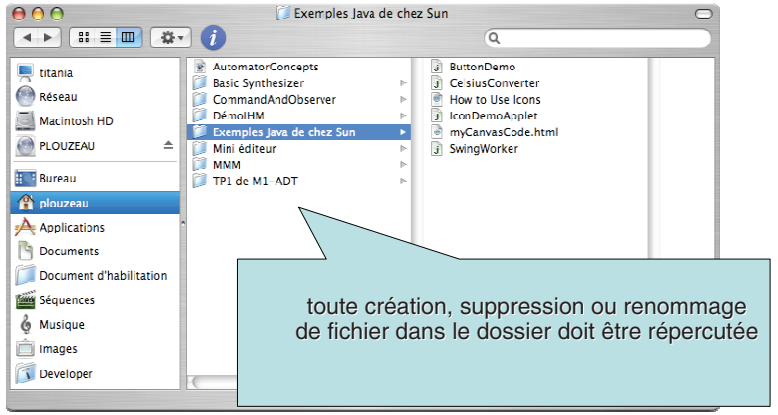
Pattern Observer

- Le patron de conception Observer (observateur) permet de coordonner deux objets : un sujet et un observateur (ou +ieurs)
  - le sujet a un état interne (défini par la valeur de ses attributs) qui change
  - l'observateur doit se synchroniser avec les changements

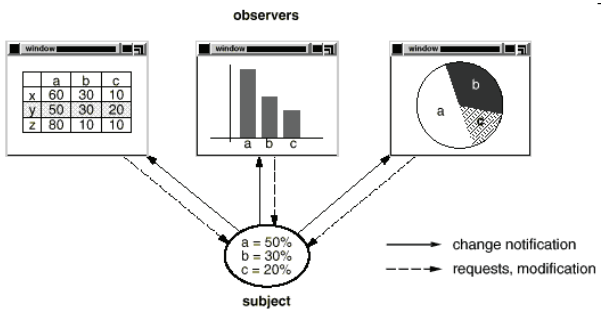
Pattern Observer: Quand l'utiliser?

- Quand le changement d'un objet se répercute vers d'autres
- Une abstraction a plusieurs aspects, dépendant l'un de l'autre. Encapsuler ces aspects indépendamment permet de les réutiliser séparément
- Quand un objet doit prévenir d'autres objets sans pour autant les connaitre

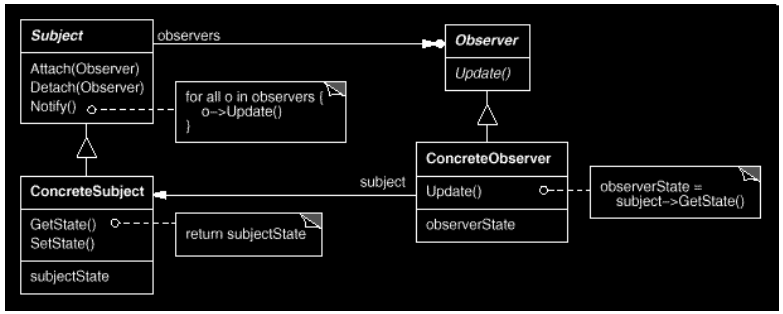
Pattern Observer: Exemple



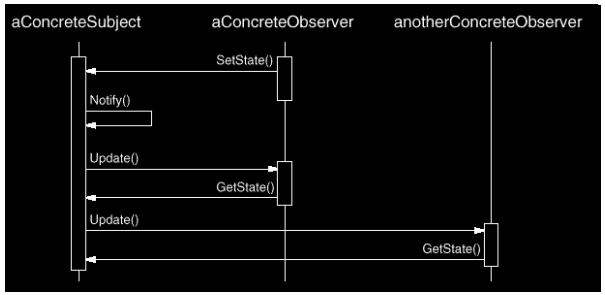
Pattern Observer: Autre exemple d'emploi



Pattern Observer: Diagramme statique générique



Pattern Observer: Diagramme de séquence



Pattern Observer: Rôles

Rôles

- **Subject**:
  - Comporte un état interne
    type non spécifié
    un patron de conception est indépendant de ce genre de détails
  - Est chargé de gérer une collection d'abonnés capable de recevoir des notifications
  - Est chargé d'envoyer un message aux abonnés lorsque son état change
- **Observer**: Est capable de réagir à la réception d'un message de notification venant d'un sujet

Pattern Observer: Définition de la structure

- Une interface Subject
  - comporte les opérations de gestion d'abonnement
  - ne comporte pas les opérations d'accès à l'état
- Une classe ConcreteSubject
  - contient les opérations d'accès à l'état
- Une interface Observer
  - contient l'opération update()
- Une classe concrète ConcreteObserver
  - cette opération sera mise en œuvre par les classes concrètes héritant de Observer
  - chaque méthode mettant en œuvre update pourra interroger le sujet pour déterminer comment se synchroniser

Patterns Comportementaux: Les autres Patterns

- Chain of Responsibility: Uncouple request sender from precise receiver on a chain.
- Command: Request reified as first-class object
- Interpreter: Language interpreter for a grammar
- Iterator: Sequential access to elements of any aggregate
- Mediator: Manages interactions between objects
- Memento: Captures and restores object states (snapshot)
- State: State reified as first-class object
- Strategy: Flexibly choose among interchangeable algorithms
- Template Method: Skeleton algo. with steps supplied in subclass
- Visitor: Add operations to a set of classes without modifying them each time

Les Design Patterns Structuraux

Patterns Structuraux: Objectifs

- Découpler interfaces et implantations de classes et d'objets
- Décrire comment les objets sont assemblés

Patterns Structuraux: Exemple

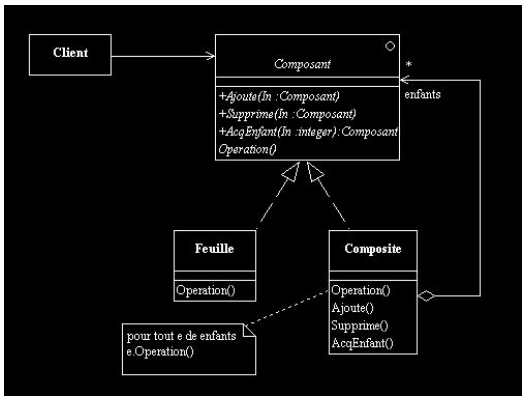
- Composite

Pattern Composite

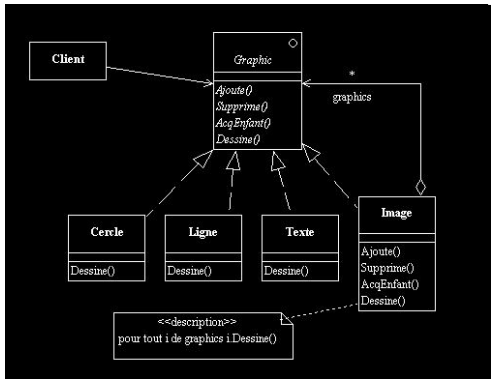
Objectifs:

- représenter une hiérarchie d'objets
- ignorer la différence entre un composant simple et un composant en contenant d'autres. (interface uniforme)

Pattern Composite: Structure



Pattern Composite: Exemple



Pattern Composite: Rôles

- Composant
  - déclare les opérations structurelles
  - déclare les opérations de traitement
- Client
  - Détient la structure, peut la modifier et demander des traitements
- Feuille
  - ne contient jamais de descendants
- Composite
  - peut contenir des descendants
  - met en œuvre un parcours si besoin

Patterns Structuraux: Les autres Patterns

- Adapter: Convert the interface of a class into another interface clients expect.
- Bridge: Decouple an abstraction from its implementations
- Decorator: Extends an object functionalities dynamically.
- Façade: Simple interface for a subsystem
- Flyweight: Efficiently sharing many Fine-Grained Objects
- Proxy: Provide a surrogate or placeholder for another object to control access to it.

Design Patterns: Résumons!

Design Pattern: le plus dur!

Le plus dur quand on veut appliquer un Pattern:

- Trouver les bons objets
- Bien choisir la granularité des objets
- Spécifier les interfaces des objets
- Spécifier l'implantation des objets
- Mieux réutiliser
  - héritage vs composition
  - délégation
- Compiled-Time vs Run-Time Structures
- Concevoir pour l'évolution

Ce que n'est pas un Pattern

- Une brique
  - Un pattern dépend de son environnement
  - Ce n'est pas du code
- Une règle
  - Un pattern ne peut pas s 'appliquer mécaniquement
  - Ne pas hésiter à l'adapter à vos besoins (notion de variantes)
- Une méthode
  - Ne guide pas une prise de décision ; un pattern est *la* décision prise
- Sans problèmes potentiels
  - Plus de classes, plus de dépendances, besoin de documenter, etc.
  - Mal utilisé (inapproprié) peut affecter les performances

Inconvénients

- Effort de synthèse ; reconnaître, abstraire...
- Apprentissage, expérience
- Les patterns « se dissolvent » en étant utilisés
- Nombreux...
  - lesquels sont identiques ?
  - De niveaux différents ... des patterns s 'appuient sur d 'autres...

Avantages

- Un vocabulaire commun, facilite la communication
- Capitalisation de l 'expérience
- Un niveau d 'abstraction plus élevé qui permet d 'élaborer des constructions logicielles de meilleure qualité
- Réduire la complexité
- Guide/catalogue de solutions

Quelques Conseils pour savoir quel pattern utiliser!

- Création d'un objet en référençant sa classe explicitement...Lien à une implantation particulière...pour éviter utilisez AbstractFactory, FactoryMethod, Prototype
- Dépendance d'une opération spécifique...pour rendre plus souple utilisez Chain Of Responsibility, Command
- Dépendance d'une couche matérielle ou logicielle utilisez AbstractFactory, Bridge

Quelques Conseils pour savoir quel pattern utiliser!

- Dépendance d'une implantation...pour rendre plus souple utilisez AbstractFactory, Bridge, Memento, Proxy
- Dépendance d'un algorithme particulier...Builder, Iterator, Strategy, TemplateMethod, Strategy
- Couplage fort...relâcher les relations utilisez AbstractFactory, Bridge, Chain Of Responsibility, Command, Facade, Mediator, Observer

Quelques Conseils pour savoir quel pattern utiliser!

- Etendre les fonctionnalités en sous-classant peut être couteux (tests, compréhension des superclasses, etc) utilisez aussi la délégation, la composition...Bridge, Chain Of Responsibility, Composite, Decorator, Observer, Strategy, Proxy
- Impossibilité de modifier une classe...absence du source, trop de répercussions, voyez Adapter, Decorator, Visitor

Autres Patterns

- Il existe aussi d'autres patterns
  - Architecture
  - Analyse

Lectures

- Software Engineering,
  - Ian Sommerville, Addison Wesley; 8 edition (15 Jun 2006), ISBN-10: 0321313798
- The Mythical Man-Month
  - Frederick P. Brooks JR., Addison-Wesley, 1995
- Cours de Software Engineering du Prof. Bertrand Meyer à cette @:
  - http://se.ethz.ch/teaching/ss2007/252-0204-00/lecture.html
- Cours d'Antoine Beugnard à cette @:
  - http://public.enst-bretagne.fr/~beugnard/

- UML Distilled 3rd édition, a brief guide to the standard object modeling language
  - Martin Fowler, Addison-Wesley Object Technology Series, 2003, ISBN-10: 0321193687
- UML2 pour les développeurs, cours avec exercices et corrigés
  - Xavier Blanc, Isabelle Mounier et Cédric Besse, Edition Eyrolles, 2006, ISBN-2-212-12029-X
- UML 2 par la pratique, études de cas et exercices corrigés,
  - Pascal Roques, 6ème édition, Edition Eyrolles, 2008
- Cours très intéressant du Prof. Jean-Marc Jézéquel à cette @:
  - http://www.irisa.fr/prive/jezequel/enseignement/PolyUML/poly.pdf
- La page de l'OMG dédiée à UML: http://www.uml.org/

- Design patterns. Catalogue des modèles de conception réutilisables
  - Richard Helm (Auteur), Ralph Johnson (Auteur), John Vlissides (Auteur), Eric Gamma (Auteur), Vuibert informatique (5 juillet 1999), ISBN-10: 2711786447