lab-sticc.univ-brest.fr/~babau/

les Design Patterns

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire Lab-STICC

UBO

Plan

- Introduction aux Design patterns
- Quelques Design Patterns
- Les anti-patterns
- Mise en œuvre des Design Patterns

jean-philippe.babau@univ-brest.fr

UBO

Historique des Design Patterns

- Patterns en architecture
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to this 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
- Application à divers domaine, dont la conception logicielle
- La thèse de Erich Gamma éditée en livre
  - « Design Patterns: Elements of Reusable Object-Oriented Software » Addison Wesley, 1995
  - E. Gamma, R. Helm, R. Johnson, J. Vlissides
  - Gang of Four' ou GoF 's Book

UBO

Objectif des Design Pattern

- Une solution classique à un problème récurrent
  - Solution éprouvée
  - Solution indépendante du contexte
- La documentation de la solution
  - Un nom
  - Explication du principe
- Haut-niveau d'abstraction
  - Solution et problème générique
  - Indépendant des langages de programmation
  - Classes et communications entre objets
- Application au contexte
  - Pas de recette toute faite
  - À adapter au problème
  - Choix de conception

UBO

Familles

- 23 Design pattern dans le GoF
- Création
  - Initialisation et configuration des classes et des objets
- Structure
  - Séparation des préoccupations et abstraction pour la réutilisation
- Comportement
  - Combiner interactions et structure

UBO

Familles

- Création
  - Abstract Factory : créer des objets sans se soucier des classes concrètes
  - Factory Method : déléguer la création à une sous-classe
  - Prototype : créer des instances à partir de prototypes
  - Object Pool : créer des ressources «recyclables» après utilisation
  - Builder : construction incrémentale d'objets complexes
  - Singleton : créer une classe ne possédant qu'une seule instance
- Structure
  - Facade : créer un interface à un sous-système
  - Adapter ou Wrapper: adapter une interface à une autre
  - Proxy : contrôler l'accès à un objet
  - Decorator : étendre dynamiquement les capacités d'un objet
  - Bridge: découpler un concept de son implémentation
  - Composite : gérer un ensemble comme un élément
  - FlyWeight : partage de données
  - MVC : découpler IHM et Données

UBO

Familles

- Comportement
  - Chain of responsability : découpler l'émetteur du receveur
  - Command : réifier, pour la manipuler, la requête
  - Interpreter: créer un langage d'interprétation
  - Mediator : gestion des interactions entre objets
  - Observer ou Publish/Subscribre: découpler vues et donnée
  - Visitor
  - Strategy,Template Method, State, Memento, Iterator
- Concurrence
  - Active object, monitor, …

UBO

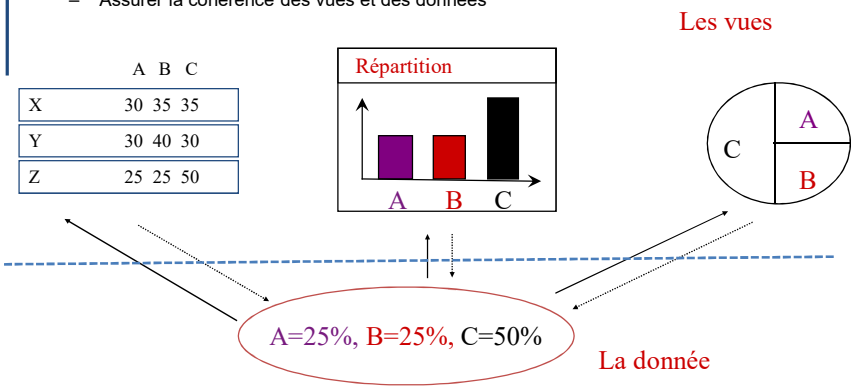
Plan

- Introduction aux Design patterns
- Quelques Design Patterns
- Les anti-patterns
- Mise en œuvre des Design Patterns

UBO

Design Pattern : Observer

- Le problème
  - Assurer la séparation des vues d'une donnée (objet)
  - Pour améliorer la réutilisation des vues et des données
  - Gestion dynamique des vues (création / suppression possible)
  - Assurer la cohérence des vues et des données

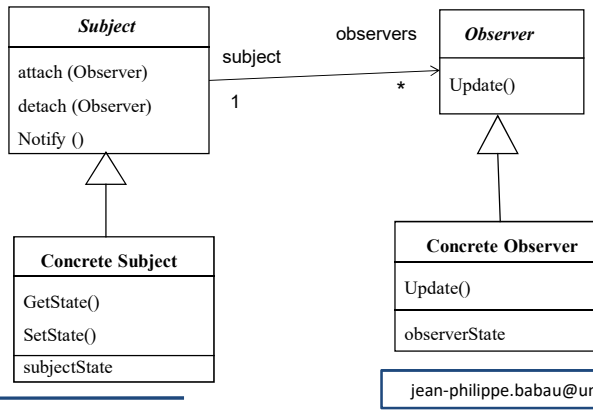


jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Observer

- La solution
  - Séparer la donnée (ConcretSubject) et les vues (ConcreteObserver)
  - Créer une classe pour la donnée (Subject) et une pour les vues (Observers)

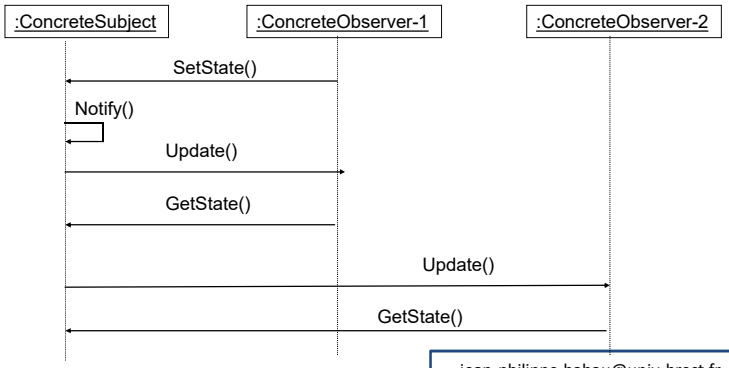


jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Observer

- La solution : le comportement
  - Communication de type publish / subscribe entre la vue et la donnée
  - Notify() : for all x in observers{ x .Update(); }
  - Update() : observerState = subject.getState();



UBO

Design Pattern : Observer

- Diffusion des informations
  - Communications non optimisées
- Mise à jour systématique
  - Pas toujours utile

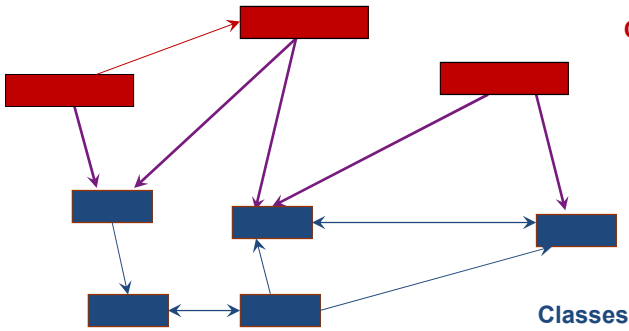
- Découplage fort entre vues et données
- Gestion cohérente des évolutions
- Diffusion entre vues qui ne se connaissent pas entre elles
  - Par le biais de la donnée

- Une implémentation disponible en Java
  http://blog.netapsys.fr/pratiquer-le-design-pattern-observer-en-15-min/
  http://www.javaworld.com/article/2077258/learn-java/observer-and-observable.html

UBO

Design Pattern : Facade

- Le problème
  - Assurer l'interface avec un sous-système
  - Pour améliorer la séparation des modules : évolutions, réutilisation
  - Masquer la complexité du sous-système

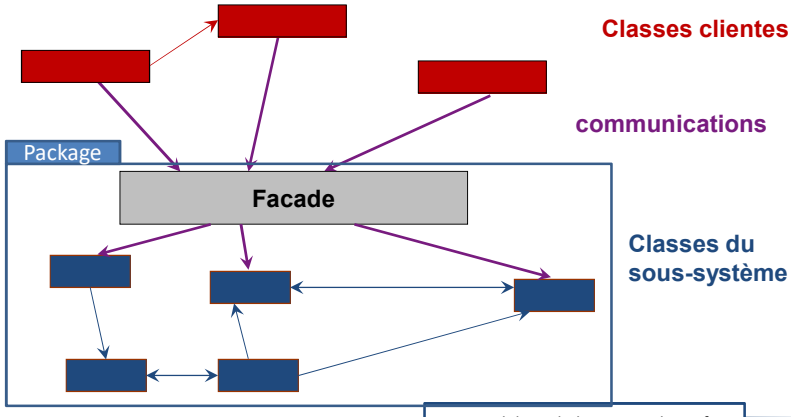


jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Facade

- La solution
  - Insérer une classe intermédiaire qui se charge du traitement des requêtes



jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Facade

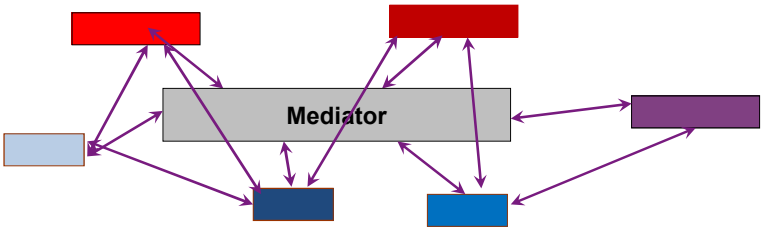
- Simplification pour le client
  - Requête à la façade qui transmet au sous-système
- Découplage client / service
  - Contrat d'utilisation et de communication
- Couche d'abstraction
  - Standardisation
  - API
- Architecture en couche
  - Attention aux performances

jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Mediator

- Le problème
  - Assurer l'interface avec plusieurs objets
- La solution
  - Insérer une classe intermédiaire qui se charge des échanges entre les divers éléments

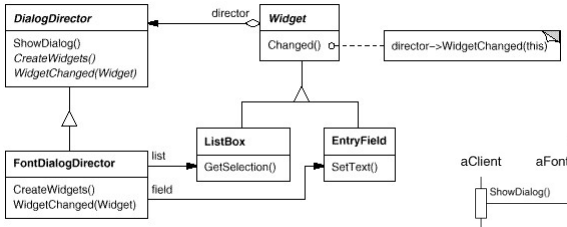


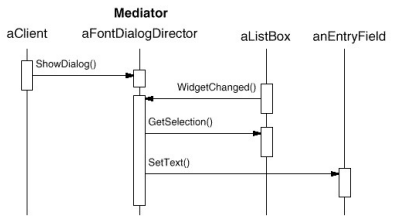
jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Mediator

- Utilisation
  - Encapsuler une séquence d'interactions
  - Découpler les objets qui interagissent



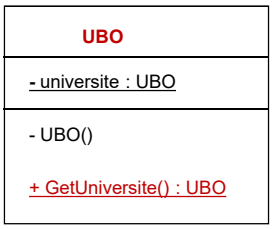


jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Singleton

- Le problème
  - Assurer qu'une classe n'a qu'une seule instance
- La solution
  - Le constructeur est privé
  - Une méthode renvoie la seule instance existante
    - Instance créée si il n'en existe pas encore

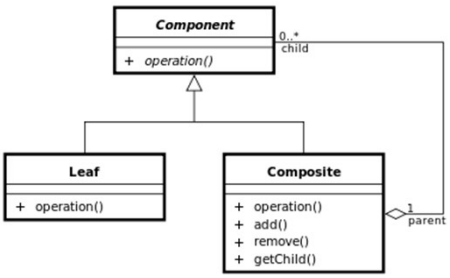


jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Composite

- Le problème
  - Gérer un objet comme un groupe d'objets
- La solution
  - Une feuille et le composite sont des éléments implémentant les méthodes de l'élément
  - Le composite implémente des méthodes de gestion des feuilles
- Utilisations
  - Interface graphique
  - Bureautique
  - Souvent des versions dégradées

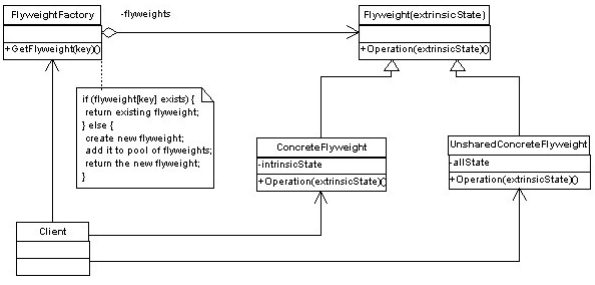


jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Flyweight

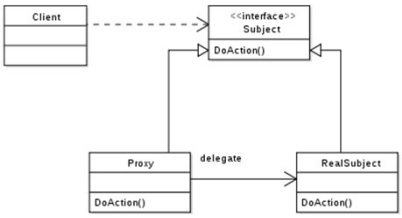
- Le problème
  - Limiter la duplication de données
- La solution
  - Créer des objets partageables (ou non) via une Fabrique qui contrôle la création
- Utilisation
  - Partage d'informations redondantes, création d 'un seul objet , optimisation de ressources



UBO

Design Pattern : Proxy

- Le problème
  - Contrôler l'accès à un objet
- La solution
  - Créer un objet qui intercepte les messages et délègue le traitement



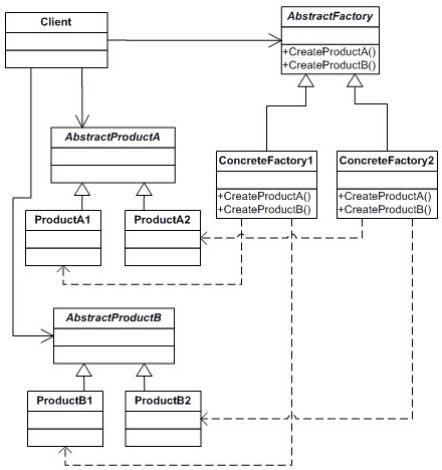
- Utilisation
  - Communication distante : encapsulation de la communication
  - Proxy virtuel : création si nécessaire de l'objet réel
  - Contrôler l'utilisation d'un objet : droits, nombre d'accès, …
  - Insérer d'autres traitements : persistence, suivi des appels, …
  - Combiné à FlyWeight : partage optimisé d'information

jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Abstract Factory

- Le problème
  - Créer un objet sans se préoccuper de son implémentation
  - Il existe plusieurs façons de construire un objet
- La solution
  - Séparation fabrique / élément
  - Séparation abstrait / concret

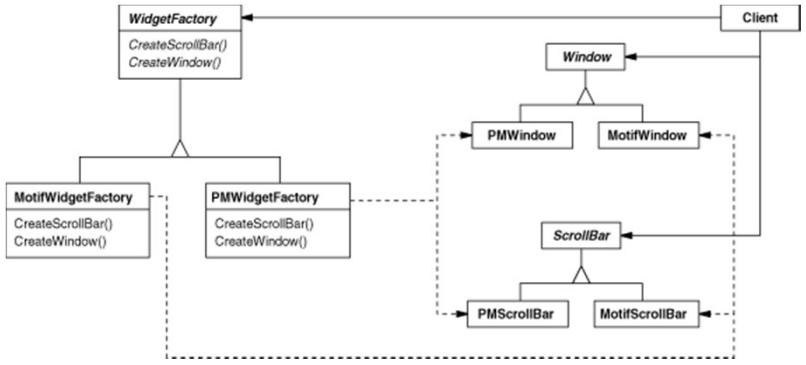


jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : Abstract Factory

- Utilisation
  - Plateformes multiples



jean-philippe.babau@univ-brest.fr

UBO

Autres Design Pattern

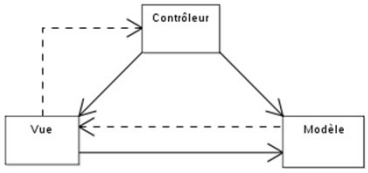
- Le Visiteur
  - Cf. exemple
- Le Delegator
  - Cf. exemple

jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : MVC

- Le problème
  - Mettre en place un IHM
  - Pour améliorer la séparation des modules : évolutions, réutilisation
- La solution : structure
  - Séparation Modèle / Vue / Contrôleur (Model / View / Controller)



jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : MVC

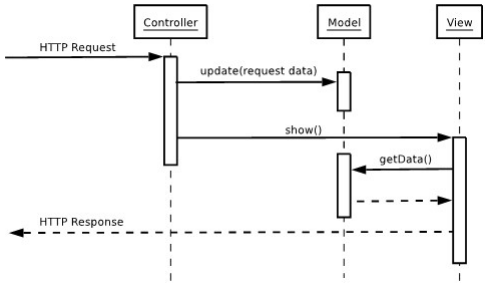
- Le Modèle
  - Les données et le traitement
  - Orienté métier
  - Indépendant des interactions
    - Indépendant des vues et du contrôleur
  - Pas de mise en forme des données
- La Vue
  - Les interactions utilisateurs
  - Actions utilisateurs et présentation du modèle
  - Pas de traitement
  - Plusieurs vues possible d'une même donnée (Observer)
- Le Contrôleur
  - Contrôle la synchronisation des vues et du modèle
  - Est informé des actions utilisateurs et agit en conséquence sur le modèle
  - Modifie les vues en fonction des actions utilisateurs
  - N'effectue aucun traitement, ne modifie pas les données

jean-philippe.babau@univ-brest.fr

UBO

Design Pattern : MVC

- La solution : comportement
  - Contrôleur
    - Analyse la requête
    - Demande si nécessaire la mise à jour du modèle
    - Détermine la vue à afficher et demande son affichage
  - La vue
    - Interroge le modèle



jean-philippe.babau@univ-brest.fr

UBO

MVC en Java

- *Listener* Java
  - Spécialisé pour la réception d'événements spécifiques
    - Changement dans le modèle
- Le modèle
  - Le modèle + gestion d'une liste de *listener*
- Le contrôleur
  - Connait le modèle et les vues
    - Abstraction des vues pour être indépendant d'une technologie
  - Initialise la liste des *listeners*
- Une vue
  - Avise le contrôleur à chaque action
  - Implémente un *listener*
    - Est notifié du changement du modèle

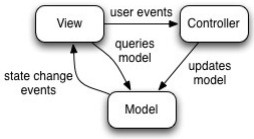
- Des exemples d'implémentation
  http://www.infres.enst.fr/~hudry/coursJava/interSwing/boutons5.html
  https://openclassrooms.com/courses/apprenez-a-programmer-en-java/mieux-structurer-son-code-le-pattern-mvc

jean-philippe.babau@univ-brest.fr

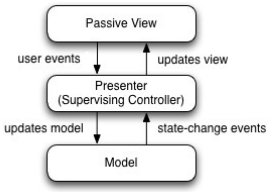
UBO

Design Pattern : MVP

- Model / View / Presenter
  - Presenter : Mediator
  - Plus adapté pour une architecture n-tiers



MVC



MVP

UBO

Design Pattern : MVVM

- Model / View / ViewModel
  - ViewModel : Adapter
    - Traitement pour la mise en forme des données
  - View
    - Gère certains contrôles liés à l'IHM
  - A appliquer lorsque le modèle ne peut être exploité par la vue
    - Modèle complexe
    - Actions utilisateurs complexes

UBO

Plan

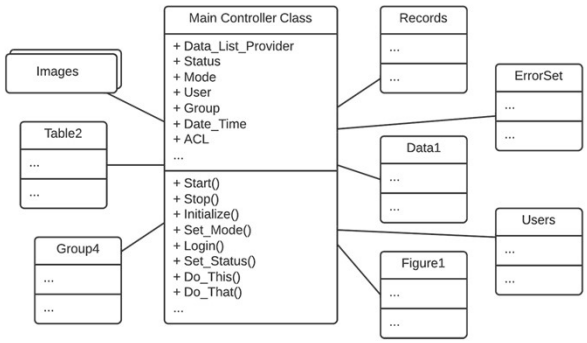
- Introduction aux Design patterns
- Quelques Design Patterns
- Les anti-patterns
- Mise en œuvre des Design Patterns

jean-philippe.babau@univ-brest.fr

UBO

Anti Pattern: Blob

- Séparation données / traitements
  - Tous les traitements dans une classe



jean-philippe.babau@univ-brest.fr

UBO

Autres anti Patterns

- Lava flow : conservation de code ancien
  - Non utilisé
  - Non documenté
  - Il est là, on ne sait pas pourquoi, dans le doute … on le garde
  - Pas de transmission entre les versions
  - Pas de maitrise de l'architecture et du code…
- Spaghetti Code
  - Une classe concerne plusieurs aspects
  - Une classe est une fonctionnalité de l'application
  - Les classes sont peu reliées entre elles
  - Mise au point difficile, réutilisation limitée…
- Cut and Paste
  - Des lignes de code sont dupliquées dans le code
  - Pas de maitrise de la conception objet
  - Mise au point plus complexe, bugs difficiles à corriger

jean-philippe.babau@univ-brest.fr

UBO

Plan

- Introduction aux Design patterns
- Quelques Design Patterns
- Les anti-patterns
- Mise en œuvre des Design Patterns

jean-philippe.babau@univ-brest.fr

UBO

Principes généraux

- Découplage abstrait/concret
  - Observer, Abstract Factory
- Découplage donnée / vues ou donnée / constructeur ou donnée / traitements
  - Observer, MVC
  - Fabrique
  - Délégation, visiteur
- Introduction d'objets intermédiaires
  - Facade, Mediator, proxy, adapter

UBO

Principes de génie logiciel

- Une classe complexe -> un ensemble de classes
  - Une classe pour les données (avec *getter* / setter)
  - Une classe pour un constructeur complexe (*fabrique*)
  - Une classe de service (*visiteur* et *délégation*)
- Un ensemble de classes -> des classes de connexion
  - Une classe d'adaptation et de communication (*médiateur*)
  - Une classe d'accès métier (*façade*)
- Séparation vue / métier
  - Plusieurs fenêtres : enchainement des fenêtres dans une classe (*controleur*)
  - Une vue : des compartiments, onglets; des boutons, menus; des labels; des zones de saisie
  - Une vue : pas de traitement, couplage faible avec le métier

UBO

Design Pattern dans la pratique

- Découverte
  - Bon sens
  - Bons sang, mais c'est bien sûr
- Première lecture
  - Catalogue universitaire, proposition académique
- 1ère expérimentation
  - Je connais, je vais adapter
  - Heu… J'aurais du lire plus attentivement
- Deuxième lecture
  - C'est puissant
- 2ème expérimentation
  - On les voit partout
  - On en met partout

UBO

Design Pattern : Pro et Cons

- Un niveau d 'abstraction élevé
  - qui permet d 'élaborer des constructions logicielles de meilleure qualité
- Réduction de la complexité par séparation des préoccupations
- Capitalisation de l'expérience en conception de logiciel
  - Catalogue de solutions
- Nombreuses mises en œuvres dans les langages de programmation

UBO

Design Pattern : Pro et Cons

- Apprentissage
  - Effort de synthèse : reconnaître, abstraire et appliquer
- Expérience de mise en œuvre
  - Choisir les « bons » Design Patterns à appliquer
    - Mise en œuvre dégradées
  - Nombreuses solutions
    - Composition de patterns …
- Complexité et efficacité du code
  - Beaucoup de couches, beaucoup de classes
  - Fondre les patterns dans le code

UBO

Bibliographie

- OMG et UML
  - http://www.omg.org/
  - http://www.uml.org/
- Wikipedia
  - http://en.wikipedia.org/wiki/Design_pattern_(computer_science)
  - https://sourcemaking.com/design_patterns
- Cours
  - Introduction to Design Patterns, Jean-Marc Jézéquel
  - Design Patterns CS 406 Software Engineering I, Aditya P. Mathur, Purdue University
  - Les Design Patterns d'Antoine Beugnard
  - Design Patterns de A. Nauwynck et N. Mellouli