Cours de Génie Logiciel

Sciences-U Lyon

Design Pattern

http://www.rzo.free.fr

Plan du cours

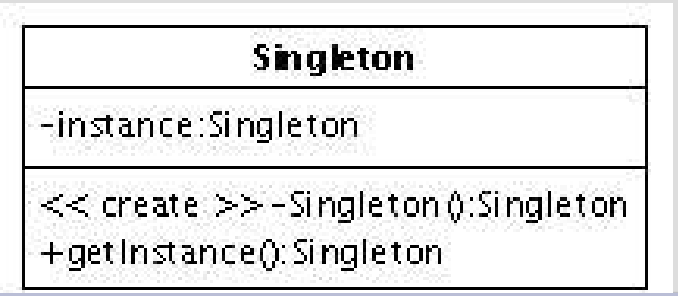
- Le génie logiciel
- Modélisation avec UML
- **Les Design Pattern**
- MDA – conception orientée modèles

Sommaire

- **Un Design Pattern**
- Motivations
- Les Design Pattern
- Les Frameworks

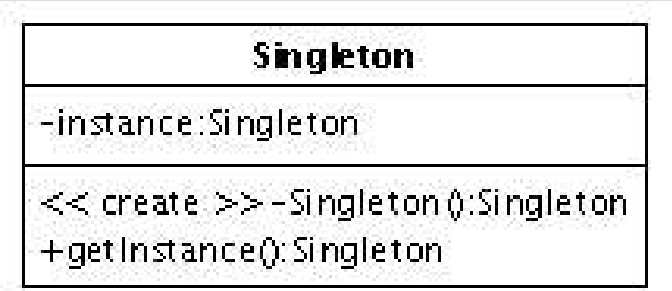
Un Design Pattern

- Un Design Pattern
  - Exemple : le Singleton
  - Problème : instance unique d'une classe
    - Ex :
      - Ballon de foot sur un terrain,
      - Compteur partagé



Un Design Pattern

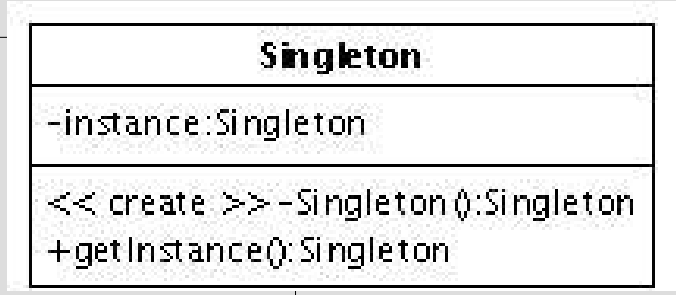
- Un Design Pattern, le Singleton
  - Un seul objet de la classe existe
    - Objet disponible dans la classe
      - Attribut de classe
    - Contrôle du Constructeur
      - privé
    - Obtention de l'instance par une méthode spécifique
      - getInstance()



Un Design Pattern

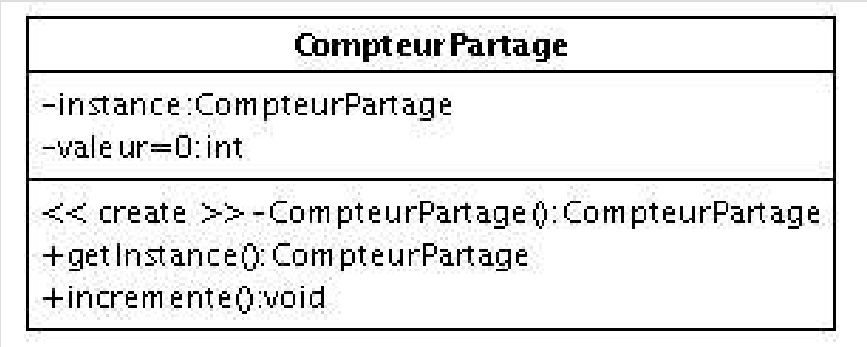
- Un Design Pattern, le Singleton

```
public class Singleton {

    private Singleton instance;

    private  Singleton() {
    }

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



Un Design Pattern

- Un Design Pattern, le Singleton
  - En contexte : le compteur partagé
    - Singleton
    - Valeur incrémentée



Un Design Pattern

- Un Design Pattern, le Singleton
  - En contexte : le compteur partagé

```
public class CompteurPartage {

    private CompteurPartage instance;
    private int valeur=0;

    private  CompteurPartage(){}

    public CompteurPartage getInstance() {
        if(instance==null)instance=new CompteurPartage();
        return instance;}

    public void incremente() {
        valeur++;}
 }
```

Un Design Pattern

- Un Design Pattern, le Singleton
  - *Nom*
    - Singleton
  - *Contexte initial*
    - Besoin de classes à instance unique
  - *Problème*
    - Comment mettre à disposition une seule instance d'une classe ?
  - *Forces en présence*
    - La classe concernée, les classes l'utilisant

Un Design Pattern

- Un Design Pattern, le Singleton
  - *Solution*
    - Constructeur privé, contrôle par une méthode annexe
  - *Conséquence de l'application*
    - Pas de constructeur accessible
  - *Utilisations connues du pattern*
    - Compteur, etc.

Sommaire

- Un Design Pattern
- **Motivations**
- Les Design Pattern
- Les Frameworks

Motivations

- Ingénierie logicielle (entreprise, recherche)
  - Création de :
  - Code
  - Modèles
  - Applications

Motivations

- Pourquoi ne pas réutiliser l'existant ?
  - Gain de temps
    - Donc d'argent
    - 'Time-to-market' réduit (temps de mise sur le marché)
    - Avantage concurrentiel
    - Indispensable à la (sur)vie de l'entreprise
  - Gain de savoir-faire
    - Capitalisation
    - Qualité, les solutions étant déjà validées
    - Indipensable à la satisfaction du client

Motivations

- Pourquoi ne pas réutiliser l'existant ?
  - Gain technologique
    - Attention portée sur les nouveautés
      - Et non la réécriture de solutions existantes
    - Amélioration des produits
    - Meilleure motivation des développeurs

Motivations

- Réutilisation de Code
  - Dépend du contexte
  - Possible au niveau algorithmique
  - Dans la conception Objet
    - Fonctionnalités dépendant
      - Pas des algorithmes
      - Mais des messages échangés entre les objets
    - Apport limité
      - Algorithmes de tri, de cryptage, de routage
      - Très spécifique
    - Hors de notre problèmatique

Motivations

- Réutilisation de modèles
  - Modélisation UML
  - Pour une situation donnée
  - Performant
    - Visibilité
    - Modifiable
  - Facile à modifier

Motivations

- Réutilisation de modèles
  - Design Pattern
  - Modèles génériques
  - Solutions à des problèmes récurrents
  - Parfois difficiles
    - Non triviaux

Motivations

- Réutilisation d'applications
  - Fonctionnalités réutilisables ?
  - Services nécessaires
    - Pour un type d'applications données
  - Framework
    - = Applications semi-finies

Sommaire

- Un Design Pattern
- Motivations
- **Les Design Pattern**
- Les Frameworks

Design Pattern

- Les Design Pattern - DP

Solution récurrente qui décrit et résout un problème général dans un contexte particulier

Design Pattern

- Les Design Pattern - DP
  - = patron
  - = schème
  - = micro-architecture
  - = canevas
  - = guide

Design Pattern

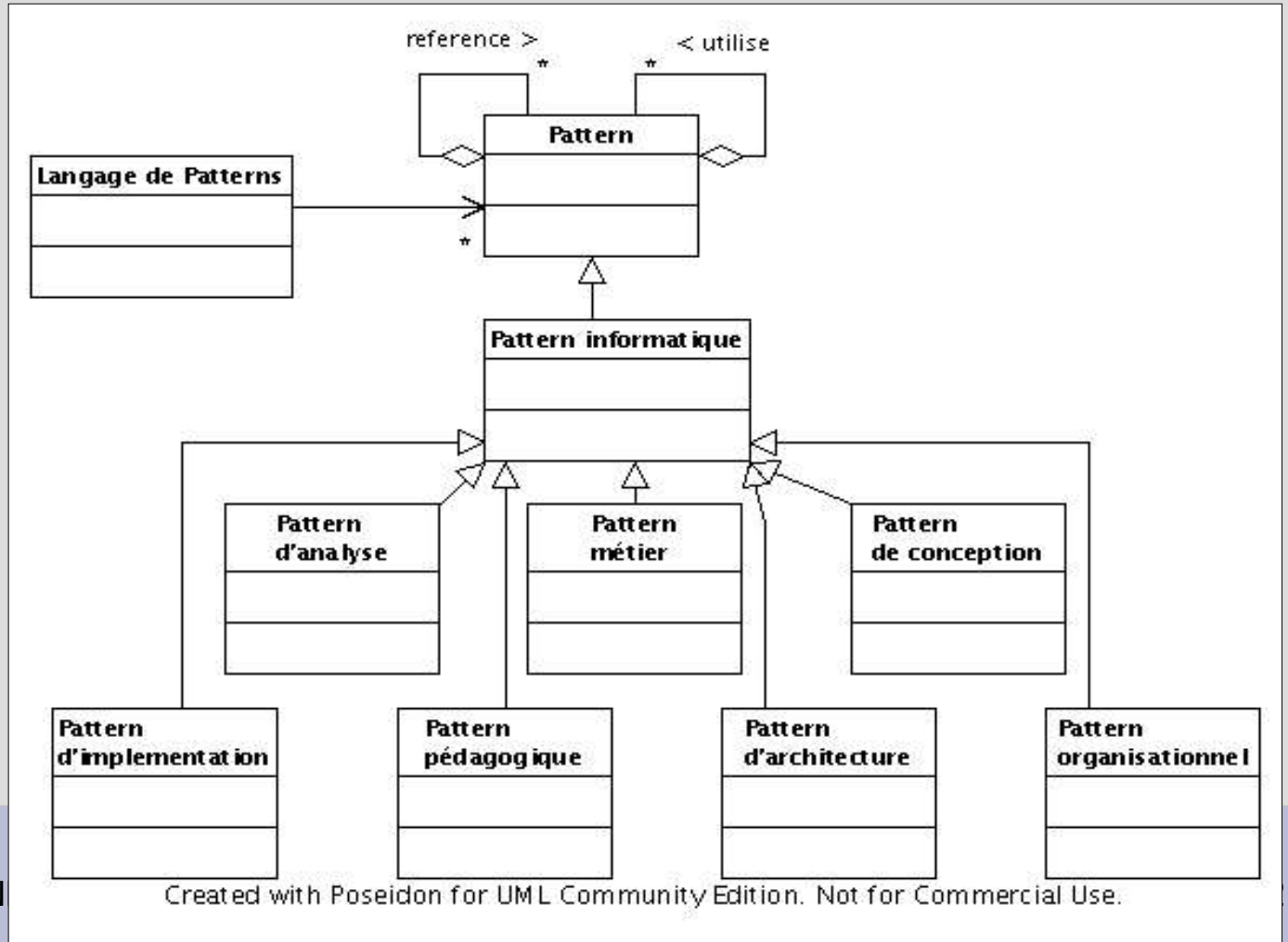
- Intérêts
  - Proposition de solution(s)
    - Pas de prescription
  - Définition d'un ensemble de classes ou d'objets collaborants
    - Granularité supérieure aux simples classes
  - Documentation associée
    - Nom, contexte, problème, conséquence
  - Nom
    - Définition d'un vocabulaire précis

Design Pattern

- Intérêts
  - Solutions éprouvées
    - Transmission de savoir-faire, d'expertise
  - Solutions adaptables
    - Comme tout modèle
  - Ensemble de solutions
    - *'Langages de patterns'*
    - Liens entre les patterns
    - Pour un contexte applicatif donné

Design Pattern

- Familles de Pattern



Design Pattern

- Familles de Pattern
  - Patterns de conception = Design Pattern
  - Patterns métier = pattern de domaine
    - Analyse + conception
    - ~= Pattern d'analyse
  - Patterns d'architecture
    - Structuration d'architecture logicielle
    - Ex : structuration en couche, client/serveur

Design Pattern

- Familles de Pattern
  - Patterns organisationnels
    - Solution à des problèmes organisationnels
    - Ex : organisation des activités de développement logiciel
  - Patterns Pédagiques
    - Solutions dans le domaine de l'apprentissage, la formation
    - Ex : introduction des DP dans une entreprise
  - Patterns d'Implémentation
    - Exploitation d'un langage de programmation donnée
    - Pour résoudre un problème typique

Design Pattern

- Description
  - *Nom*
  - *Contexte initial*
  - *Problème*
  - *Forces en présence*
  - *Solution*
  - *Conséquence de l'application*
  - *Utilisations connues du pattern*
  - *Liens*

Sommaire

- Un Design Pattern
- Motivations
- Les Design Pattern
- **Les Frameworks**

Frameworks

- Les Frameworks
  - Applications pour un domaine donné
  - Fonctionnalités récurrentes
  - Réutilisation des services possible
  - Granularité plus large
    - Composants, couches d'architecture logicielle
    - != Design Pattern, granularité de niveau classe

Frameworks

- Caractéristiques
  - Définition de
    - La structure globale
    - Mécanisme de contrôle
    - Collaboration des objets entre eux
    - Mécanisme d'extensibilité
  - Implémentation
    - Restriction à un langage de programmation donné
    - Liberté de modification limitée

Frameworks

- Caractéristiques
  - Principe d'Hollywood
    - 'Don't call us, we'll call you'
    - 'Ne nous appelez pas, nous vous appelerons'
    - L'utilisateur précise seulement les classes dont il a besoin
      - Par héritage des classes du framework
    - Le framework fournit
      - le lien entre ces classes,
      - les méthodes types (héritées par les classes utilisateur)

Frameworks

- Types
  - Framework métier
    - = framework vertical
    - Dépendant d'un domaine
  - Framework d'application
    - = framework horizontal
    - Expertise nécessaire à une application générique
    - Ex : j2ee/struts pour les serveurs web
    - Ex : Framework IHM (interface Homme-Machine)

Frameworks

- Types
  - Framework Boîte Blanche
    - Manipulation des classes
    - Héritage et implémentation par l'utilisateur
    - Extension des abstractions existantes
  - Framework Boîte Noire
    - Utilisation de classes déjà implémentées
    - Plus simple

Frameworks

- Avantages
  - Amélioration de la productivité
    - L'architecture générale fournie, et validée
      - Analyse
      - Conception
      - Implémentation
    - Attention portée sur les spécificités de l'application
  - Prototypage rapide
  - Comportement flexible et extensible
  - Maintenance facilitée : documentation disponible

Frameworks

- Inconvénients
  - Nombreuses classes interconnectées
    - Compréhension difficile
    - Mise en oeuvre idem
  - Sur-ensemble des besoins d'une application
    - En grande partie inutile
  - Liberté de conception restreinte
  - Incompatibilité des frameworks entre eux
    - *'architecture mismatch'*
    - Pb de versions des technologies employées
    - Flots de contrôle multiples

Frameworks

- DP et frameworks
  - Utilisation conjointe
  - Meilleure compréhensibilité
  - Niveaux de granularité
    - Framework
    - Pattern
    - Classe

Design Pattern Introduction

- Bilan