Chapitre 3 : Conception

21

Conception

- Le maître-mot durant la conception est la qualité
- Les attributs de la qualité sont
  - Fonctionnalité
  - Utilisabilité
  - Fiabilité
  - Performance
  - Adaptabilité

22

Conception

- Evolution de la conception logicielle
  - Programmes modulaires
  - Orienté objets
  - Architecture logicielle
  - Design patterns
  - Orienté aspects
  - Dirigé par les modèles
  - Dirigé par les tests

23

Conception

- Tâches de la conception (générique)
  - Choisir un style d'architecture
  - Décomposer le modèle d'analyse en sous-systèmes
  - Créer un ensemble de classes de conception, de composants
  - Concevoir les interfaces avec les systèmes externes
  - Concevoir les interfaces utilisateur
  - Concevoir les composants
  - Développer un modèle de déploiement

24

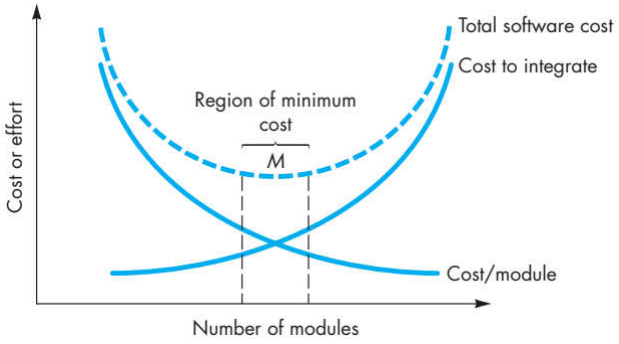
Concepts

- Abstraction
- Architecture
- Patterns
- Séparation des préoccupations (separation of concerns)
- Modularité
- Masquer les détails (Information hiding)
- Indépendance fonctionnelle
- Raffinement (Refinment)
- Aspects
- Refactoring (réusinage)
- Conception orientée objets
- Classes de conception
- Inversion de dépendance
- Design for test

25

Concepts

- Modularité



26

Concepts

- Masquer les détails
  - L'objectif est de masquer les détails des structures de données et des procédures derrière les interfaces des modules. Les utilisateurs du module n'ont pas besoin de connaître les détails de ce dernier.

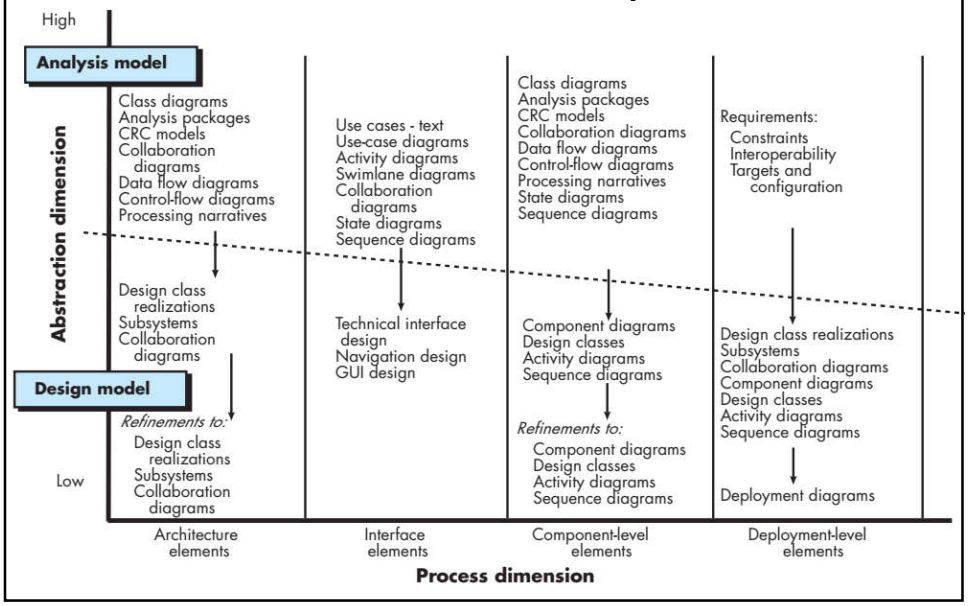
27

Concepts

- Indépendance fonctionnelle
  - Cohésion
    - La cohesion est une mesure qualitative du degré de concentration d'un module sur une seule chose.
  - Couplage
    - Le couplage est une mesure qualitative du degré de connexion d'un module à d'autres modules et aux systèmes externes.

28

Modèle de conception



Modèle de conception

- 4 éléments majeurs
  - Données
  - Architecture
  - Composants
  - Interfaces

30

Modèle de conception

- Données
  - Au niveau architecture : Fichiers et bases de données
  - Au niveau composants : structures de données et implémentation locale des objets
  - Au niveau gestion : datawarehouse et knowledge management

31

Modèle de conception

- Architecture
  - Le plan de la solution logicielle élaboré à partir de trois sources :
    1. Informations à propos du domaine de l'application
    2. Cas d'utilisation et classes d'analyse
    3. Patterns architecturaux disponibles (styles d'architectures)

32

Modèle de conception

- Interface
  - Interfaces utilisateurs
  - Interfaces entre composants
  - Interfaces avec les systèmes externes

33

Modèle de conception

- Composants
  - Description détaillée de chacun des composants
- Déploiement
  - Comment les fonctionnalités et les sous-systèmes vont être implémentés sur les composants physiques qui supportent le logiciel

34

Architecture

- L'architecture logicielle modélise la structure d'un système et la manière dont les composants (données et procédures) collaborent entre eux
- Un composant est une module de programme; une classe ; une base de données; etc.
- L'architecture offre une vue globale du système permettant de l'examiner en tant qu'un tout

35

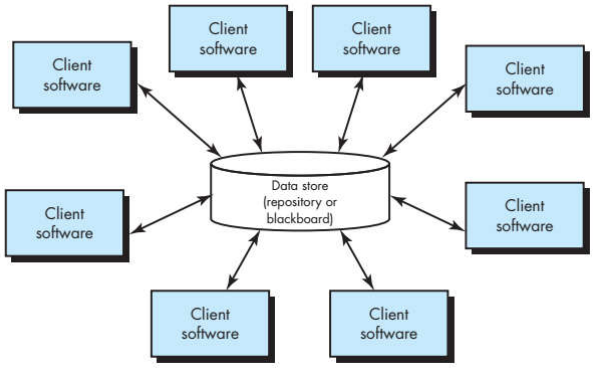
Architecture

- Un système logiciel se présente sous forme de styles architecturaux. Un style décrit (1) un ensemble de composants qui réalisent des fonctions requises par le système (2) un ensemble de connexions pour la communication et la coordination entre composants (3) les contraintes selon lesquelles s'intègrent ces composants et (4) les modèles sémantiques qui permettent de comprendre les propriétés des différents éléments du systèmes

36

Styles architecturaux

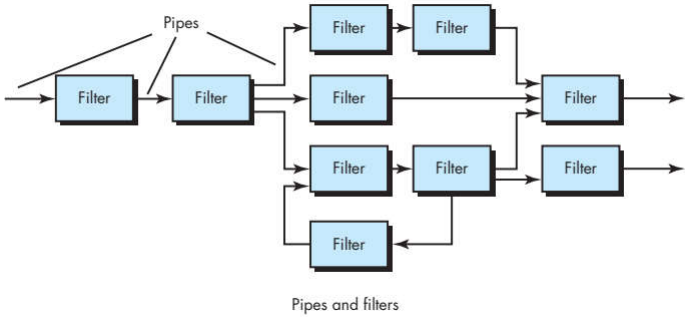
- Centralisé



37

Styles architecturaux

- Pipe and Filter

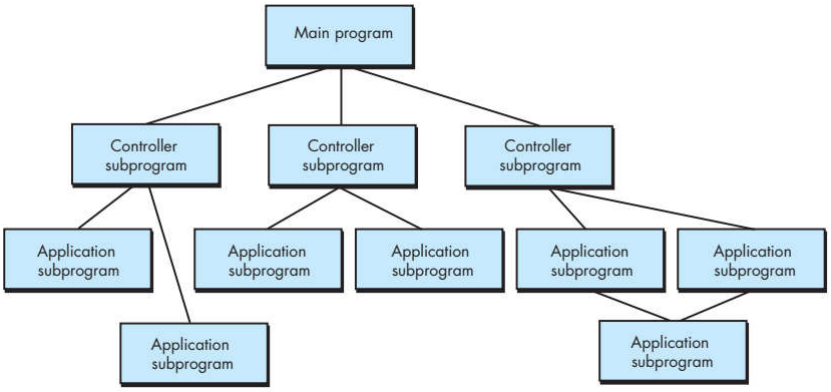


Pipes and filters

38

Styles architecturaux

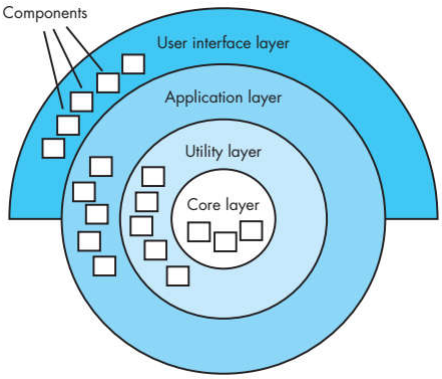
- Appel retour



39

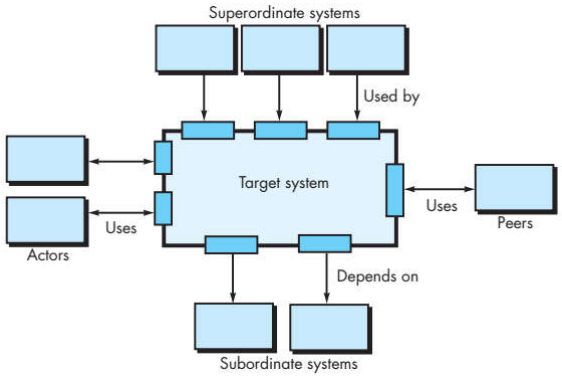
Styles architecturaux

- Appel retour



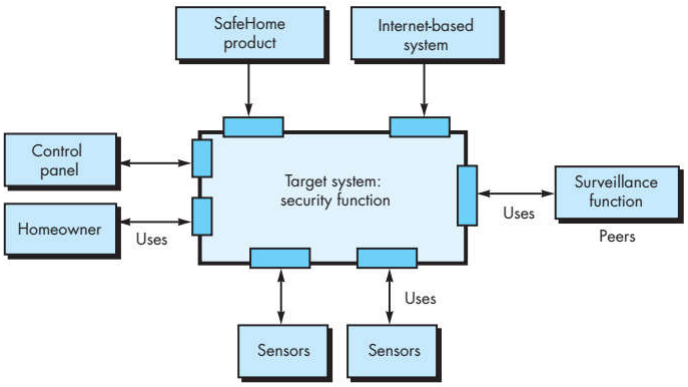
40

Contexte



41

Contexte : exemple



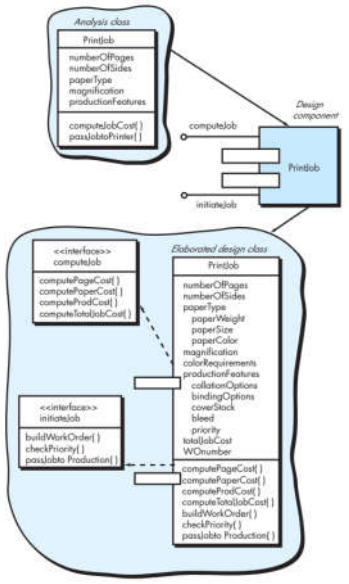
42

Composants

- Un composant est une « partie du système modulaire, déployable et remplaçable qui encapsule son implémentation et qui expose un ensemble d'interfaces »
- D'un point de vue objets, un composant est un ensemble de classes qui collaborent entre elles.

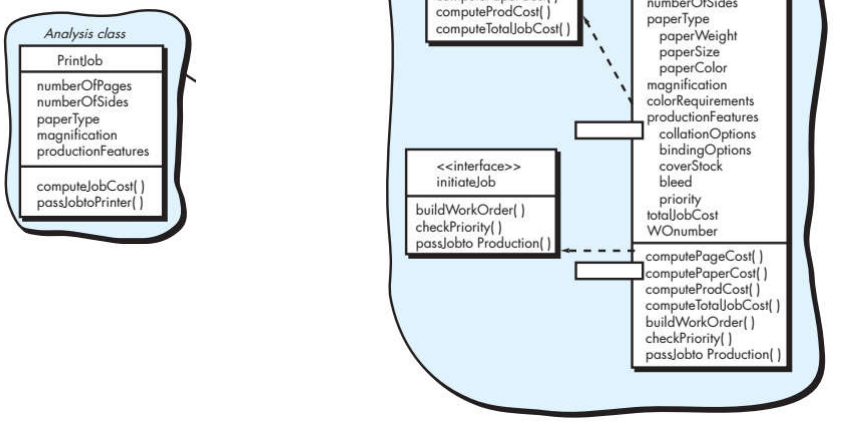
43

- Exemple



44

- Exemple



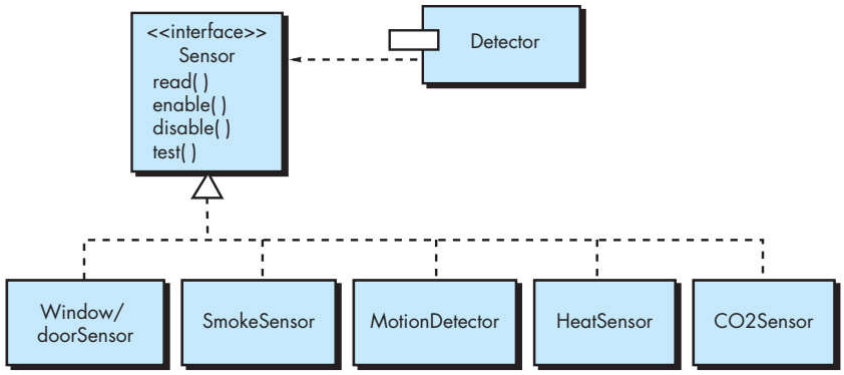
45

Principes de conception

- Open Closed Principle
  - A module should be open for extension but closed for modification

46

OCP : Exemple



47

Principes de conception

- Dependency Inversion Principle
  - Depend on abstractions. Do not depend on concretions
- Interface segregation principle
  - Many client-specific interfaces are better than a general purpose interface

48

Principes de conception

- Cohésion
  - La cohésion dot être aussi forte que possible
  - La cohésion implique qu'un composant ou classe encapsule seulement les attributs et les opérations qui sont liés les uns aux autres ou aux opérations et attributs du même composant

49

Principes de conception

- Couplage
  - Le couplage mesure le degré de connexions entre classes.
  - Il est important d'avoir un couplage faible entre différents composants

50

Conception interfaces utilisateurs

- Golden rules
  - Donner le contrôle au utilisateur
  - Réduire l'effort de mémorisation de l'utilisateur
  - Créer des interfaces cohérentes

51

Donner le contrôle à l'utilisateur

- Ne pas forcer l'utilisateur à faire des actions inutiles ou non souhaitables
- Essayer de garder l'interaction flexible (raccourcis clavier, commande vocable)
- Permettre l'interruption ou l'annulation des actions de l'utilisateur
- Permettre de personnaliser les actions (macros par exemple)
- Masquer les aspects techniques (système d'exploitation par exemple)
- Permettre l'interaction directe avec les objets de l'interface (drag & drop)

52

Réduire l'effort de mémorisation

- Réduire la demande en mémoire courte
- Définir des valeurs par défaut significatives
- Définir des raccourcis intuitifs (mnémoniques)
- Aspect visuel issu de la réalité (figures)
- Communiquer l'information de manière progressive

53

Cohérence des interfaces

- Mettre l'action courante dans son contexte (titre fenêtres, couleur)
- Garder la cohérence tout au long de la d'une ligne de produits

54

Design patterns

- Un design pattern peut être défini comme étant une règle qui relie un contexte, un problème et une solution.

55

Design patterns

- Différents types de patterns
  - Architecturaux
  - Données
  - Composants (design patterns)
  - Interfaces utilisateur
  - Applications web
  - Applications mobiles
  - Etc.

56

Design patterns

- Gof (orienté objets)
  - Création
  - Structure
  - Comportement

57

Design patterns

- Framework
  - Un framework est une « mini-architecture » réutilisable qui fournit une structure et un comportement génériques pour un ensemble d'abstractions logicielles situées dans un certain contexte et qui spécifie la manière dont ces abstractions collaborent entre elles.
  - Un framework n'est pas une architecture, c'est plutôt un squelette avec des réceptacles (plug points, slots) qui permettent de l'adapter à un problème spécifique

58

Design patterns

- Différence entre Framework et design pattern (Gof)
  - Les design patterns sont plus abstarits que les frameworks (un framework est incorporé dans du code)
  - Les design patterns sont plus petits que les frameworks (structure réduite)
  - Les design patterns sont plus généraux que les framewoks

59

Design patterns

- Patterns architecturaux
  - L'architecture d'un logiciel peut avoir un ensemble de patterns architecturaux qui traitent de problèmes tels que la concurrence, la persistance ou la distribution
  - Un pattern architectural s'inscrit dans un style aarchitectural

60

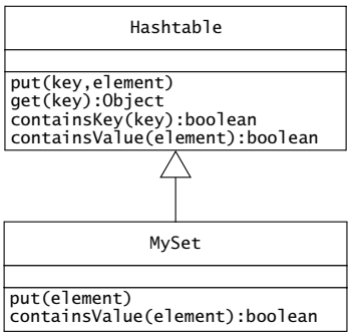
Design patterns

- Héritage de spécification
  - Définit une hiérarchie entre classes
- Héritage d'implémentation
  - Réutilisation du code

61

Design patterns

- Exemple : héritage d'implémentation



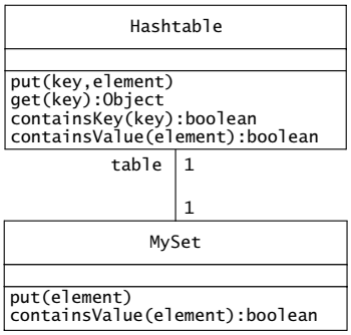

```
/* Implementation of MySet using
inheritance */
class MySet extends Hashtable {
    /* Constructor omitted */
    MySet() {
    }

    void put(Object element) {
        if (!containsKey(element)){
            put(element, this);
        }
    }
    boolean containsValue(Object
            element){
        return containsKey(element);
    }
    /* Other methods omitted */
}
```

62

Design patterns

- Délégation

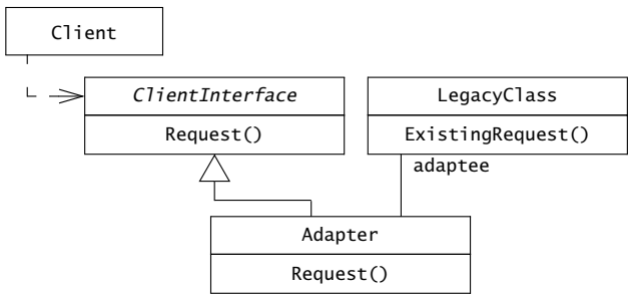



```
/* Implementation of MySet using
delegation */
class MySet {
    private Hashtable table;
    MySet() {
        table = Hashtable();
    }
    void put(Object element) {
        if (!containsValue(element)){
            table.put(element,this);
        }
    }
    boolean containsValue(Object
            element) {
        return
        (table.containsKey(element));
    }
    /* Other methods omitted */
}
```

63

Design patterns

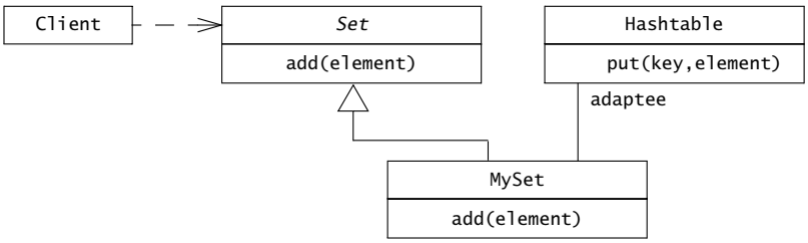
- Adapter



64

Design patterns

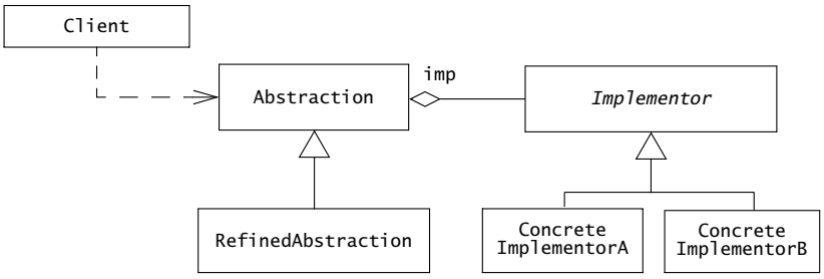
- Exemple : Adapter



65

Design patterns

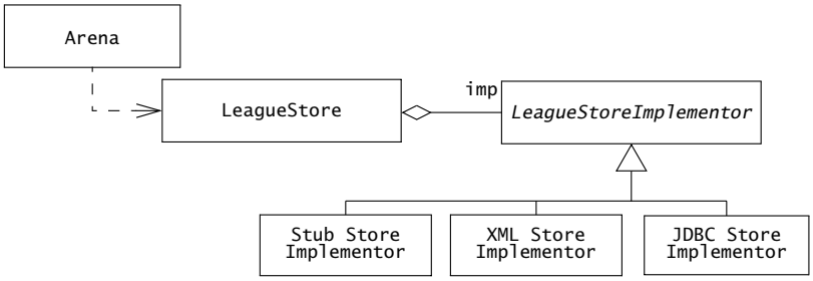
- Bridge pattern



66

Design patterns

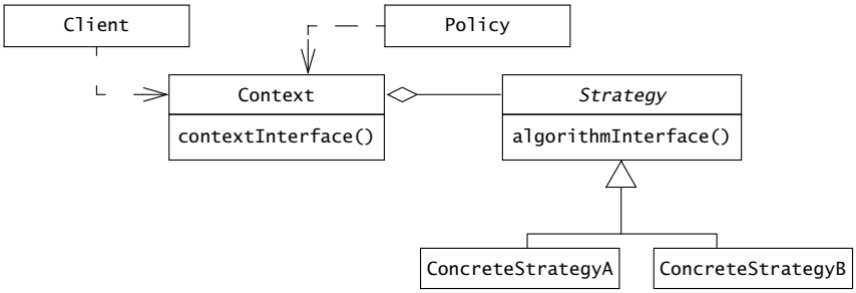
- Bridge pattern : Exemple



67

Design patterns

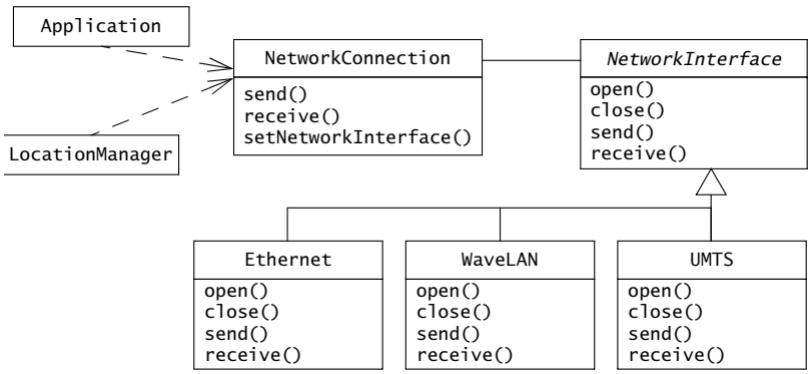
- Strategy pattern



68

Design patterns

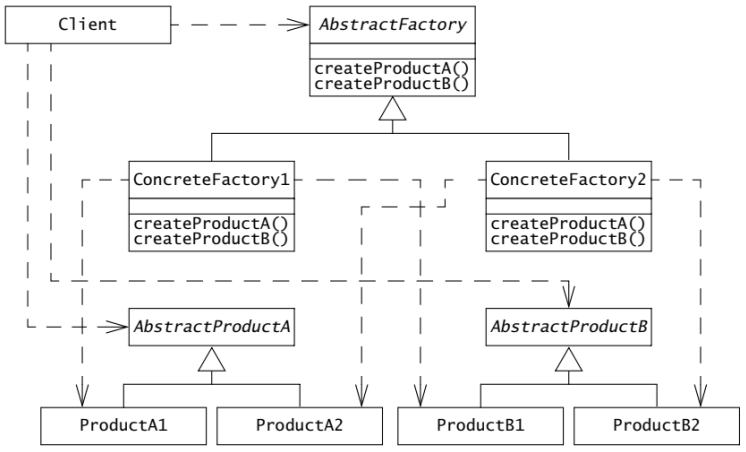
- Strategy pattern : Exemple



69

Design patterns

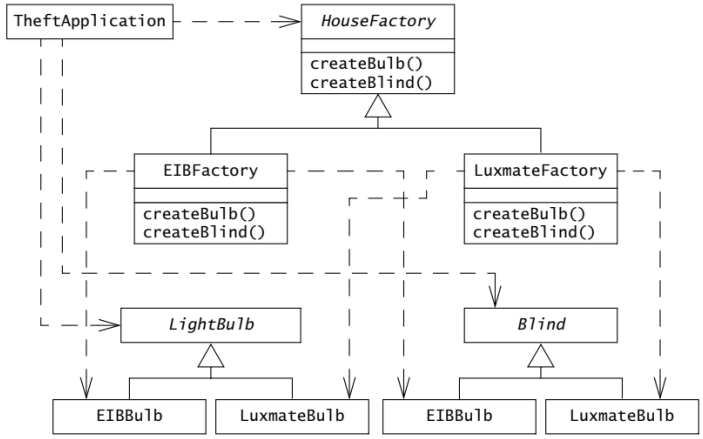
- Abstract Factory



70

Design patterns

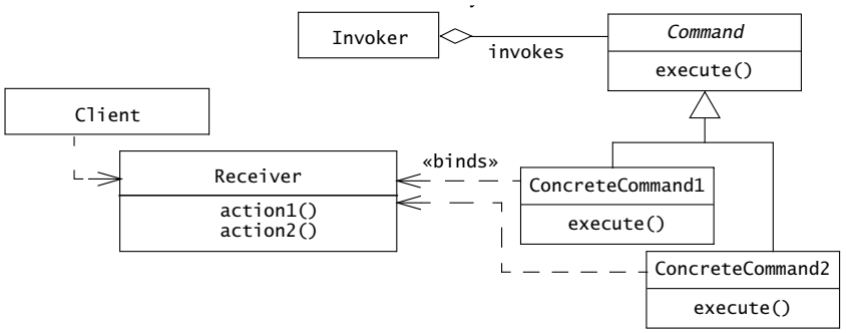
- Abstract Factory : Exemple



71

Design patterns

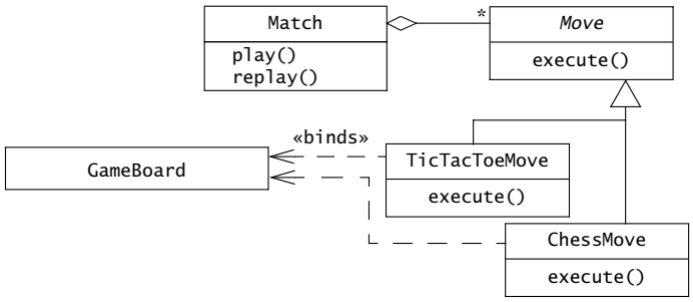
- Command pattern



72

Design patterns

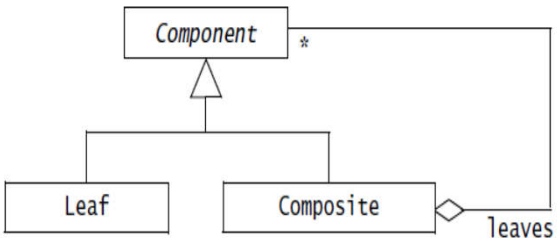
- Command pattern : Exemple



73

Design pattern

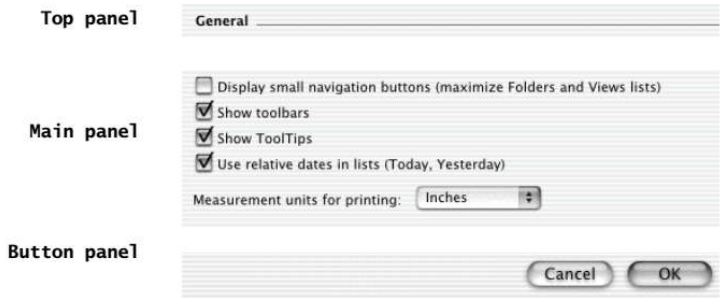
- Composite design pattern



74

Design patterns

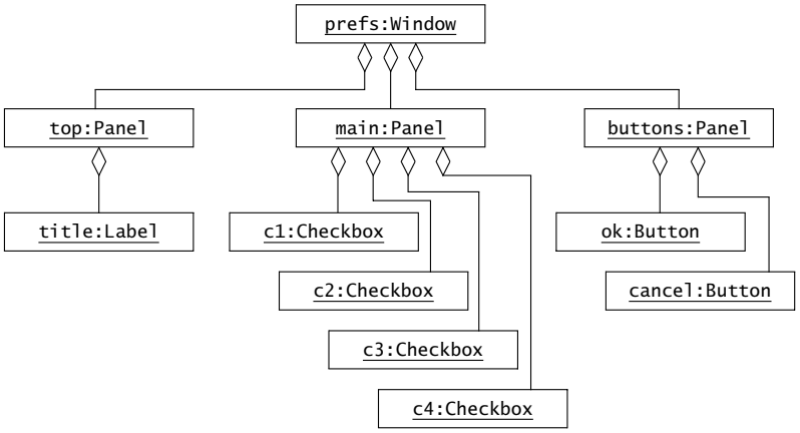
- Composite design pattern



75

Design pattern

- Composite design pattern : Exemple



76

Design pattern

- Composite design pattern