CENTRALE
L Y O N

Patterns

Approches pour la réutilisation
Patterns

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

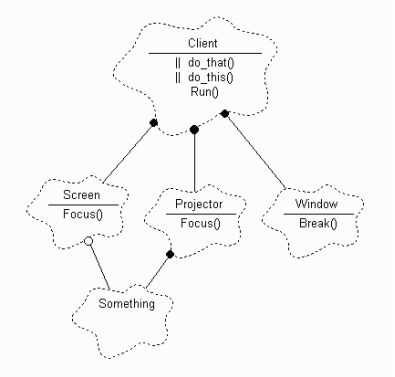
Design Patterns

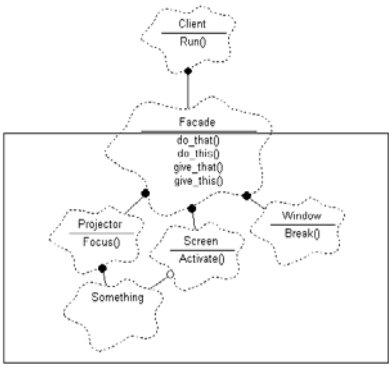
- Chaque pattern décrit un problème qui apparaît très souvent dans notre environnement et qui constitue un corps de solution à ce problème de façon que cette solution puisse être utilisée des millions de fois sans faire deux fois la même chose (Christopher Alexander 77).

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exemple de pattern : Façade





BTD/GL/Pat

3

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Quatre éléments décrivent un pattern

- **Le nom** : identifie le problème de conception, la solution et les conséquences en quelques mots. Ce nom augmente immédiatement le vocabulaire de conception. Ce vocabulaire permet de communiquer et augmenter le niveau d'abstraction.
- **Le problème** : décrit quand on peut appliquer le pattern. Il présente le problème dans son contexte. Il peut préciser les spécificités du problème et des conditions d'application.
- **La solution** : décrit les éléments qui constituent la réponse, leurs relations, responsabilités et collaborations. On ne décrit pas une implémentation particulière, car un pattern peut être considéré comme un schéma qui peut être appliqué dans différentes situations. En réalité, un pattern constitue une description abstraite d'un problème de conception et indique comment une structuration générale de classes et d'objets le résout.
- **Les conséquences** : indiquent les résultats et les principes d'application du pattern. Ces conséquences aident à comprendre et à évaluer le pattern.

BTD/GL/Pat

4

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Format de description d'un pattern (1)

- **Nom et classification** : le nom prend place dans le vocabulaire de conception, la classification place le pattern dans l'ensemble.
- **Objectif** : décrit brièvement les quoi, pourquoi et limitations
- **Egalement connu sous** : donne d'autres noms pour ce pattern
- **Motivation** : décrit un scénario qui illustre le problème et comment une configuration de classes et d'objets résout ce problème. Ce scénario va aider dans des descriptions plus abstraites de certains patterns.
- **Utilisation** : donne des situations dans lesquelles le pattern peut être appliqué. Quels sont les exemples de mauvaise utilisation du pattern et comment reconnaître ses situations.

BTD/GL/Pat 5

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Format de description d'un pattern (2)

- **Structure :** donne une représentation graphique des classes du pattern en utilisant la notation basée sur UML et les diagrammes d'interaction pour exprimer la séquence d'appels et de collaborations entre objets.
- **Participants :** donne la liste des classes, objets intervenant dans la pattern et leur rôles.
- **Collaborations :** indique comment les participants collaborent pour remplir leurs rôles.
- **Conséquences :** indique comment le pattern remplit ses objectifs, quels sont les tendances et les résultats d'utilisation du pattern et quels éléments de la structure laisse-t-il évoluer librement.

BTD/GL/Pat 6

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Format de description d'un pattern (3)

- **Implémentation :** donne des principes et des astuces pour obtenir une bonne implémentation.
- **Code de principe :** les fragments de codes indiquent comment on peut implémenter le pattern.
- **Utilisations connues :** décrit les exemples d'utilisation du pattern dans des systèmes utilisés.
- **Patterns connexes :** indique des patterns semblables, leurs différences et compatibilités éventuelles.

BTD/GL/Pat 7

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Description

- **Un Design pattern c'est :**
  - ➔ la description d'un problème rémanent et d'une solution classique.
  - ➔ une solution au problème pouvant être réutilisée sans être toutefois identique dans ses emplois.
  - ➔ Un design pattern décrit une partie de la solution avec les relations avec les autres parties du système et la technique d'architecture logicielle.
  - ➔ Mais ce n'est pas une brique, car un pattern dépend de son environnement :
    - ✓ une règle, car un pattern ne s'applique pas mécaniquement
    - ✓ une méthode, car un pattern ne dirige pas la solution à prendre : c'est **la solution**

BTD/GL/Pat 8

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Avantages et désavantages

- **Avantages**
  - ➔ un vocabulaire commun
  - ➔ capitalisation de l'expérience
  - ➔ niveau d'abstraction élevée qui permet des architectures de haut niveau
  - ➔ réduction de la complexité
  - ➔ guide/catalogue de solutions
- **Inconvénients**
  - ➔ effort de synthèse : reconnaître, abstraire
  - ➔ apprentissage, expérience
  - ➔ patterns se dissolvent en étant utilisés
  - ➔ nombre de patterns lequel convient ?
  - ➔ différents niveaux de patterns s'appuyant sur d'autres patterns

BTD/GL/Pat

9

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Vocabulaire

- **Nom** : réunifie l'idée vocabulaire commun
- **Problème** : quand appliquer la forme le contexte
- **Solution** : éléments de la solution, relations, etc. pas de manière précise mais suggestive
- **Conséquences** : résultats et compromis d'utilisation

BTD/GL/Pat

10

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Interactions entre formes et langage

➔ Influence du langage sur le pattern
- ✓ langage implante les formes de bas niveau
- ✓ certaines formes utilisent des concepts spécifiques à certains langages donc non indépendance
- ✓ certaines formes conduisent à des implémentations compliquées

Influence pattern langage
- ✓ capitalise la réflexion de programmation

➔ Application lors de la conception :
- ✓ trouver les bons objets
- ✓ bien choisir granularité des objets
- ✓ spécifier interface des objets
- ✓ spécifier implantation des objets
- ✓ concevoir pour l'évolution

BTD/GL/Pat

11

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les différents patterns

➔ ***Creational patterns (formes de création)***

➔ ***Structural patterns (formes de structure)***

➔ ***Behavioural patterns (formes de comportement)***

BTD/GL/Pat

12

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Creational patterns (formes de création)

- **Abstraire le processus d'instanciation**
  - ➔ rendre indépendante la façon dont les objets sont créés utilisés implémentés
  - ➔ encapsuler la connaissance de la classe concrète qui instancie
  - ➔ cacher qui crée, ce qui crée et comment
- **Principes**
  - ➔ **abstract factory** : on passe un paramètre à la création qui défini ce que l'on va créer
  - ➔ **builder** : on passe en paramètre un objet qui sait construire l'objet à partir d'une description
  - ➔ **factory method** : la classe sollicitée appelle des méthodes abstraites, il suffit de sous classer
  - ➔ **prototype** : des prototypes variés existent qui sont copiés et utilisées
  - ➔ **singleton** : unique instance

BTD/GL/Pat

13

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Structural patterns (formes de structure)

- **Comment les objets sont complémentaires**
  - ➔ les patterns sont complémentaires les uns les autres
- **Principes**
  - ➔ **adapter** : rendre un objet conforme à un autre
  - ➔ **bridge** : pour lier une abstraction à une implémentation
  - ➔ **composite** : basé sur des objets primitifs et composants
  - ➔ **decorator** : ajoute des services à un objet
  - ➔ **facade** : cache une structure complexe
  - ➔ **flyweigth** : petits objets destinés à être partagés
  - ➔ **proxi** : un objet en cache un autre

BTD/GL/Pat

14

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Behavioural patterns (formes de comportement)

- **Des algorithmes**
  - ➔ des comportements entre objets
  - ➔ des formes de communication entre objets
- **Principes**
  - ➔ **Chain of responsability**
  - ➔ **Command**
  - ➔ **Interpreter**
  - ➔ **Iterator**
  - ➔ **Mediator**
  - ➔ **Memento**
  - ➔ **Observer**
  - ➔ **State**
  - ➔ **Strategy**
  - ➔ **template method**
  - ➔ **visitor**

BTD/GL/Pat

15

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exemple : Le pattern Strategy

- *Le problème*
  - ➔ Lors de manipulations de tableaux, on a besoin de trouver une valeur de pivot.
  - ➔ De meilleurs résultats peuvent être obtenus en utilisant différents algorithmes de recherche du pivot pour différentes données.
  - ➔ On peut alors utiliser le pattern **STRATEGY**.
- *But*
  - ➔ Définir différents algorithmes, les encapsuler et les rendre interchangeables.
  - ➔ Ce pattern résout les différents problèmes :
    - ✓ Comment modifier les règles de sélection d'un pivot sans modifier l'algorithme principal (quick sort) ?
    - ✓ Faire en sorte que les algorithmes soient compatibles avec l'interface sans tenir compte de l'implémentation.

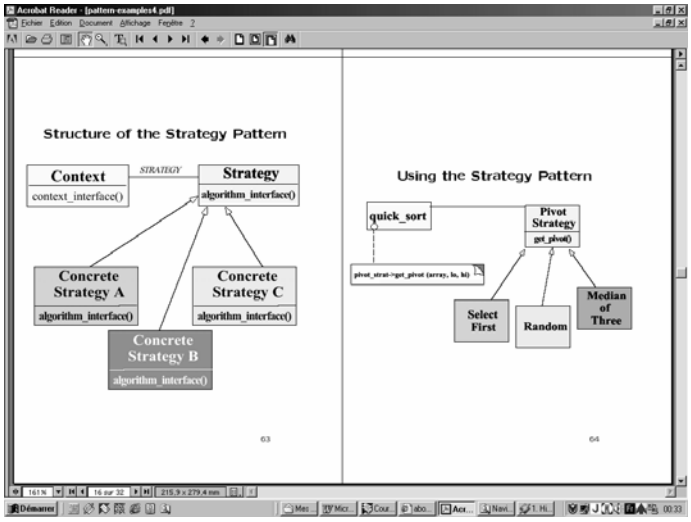
BTD/GL/Pat

16

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Structure



BTD/GL/Pat

17

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Implémentation

```
ARRAY est le type spécifique de données
template <class ARRAY>
void sort (ARRAY &array)
{
Pivot<ARRAY> *pivot_strat =
Pivot<ARRAY>::make_pivot(Options::instance ()->pivot_strat ());
quick_sort (array, pivot_strat);
}
template <class ARRAY, class PIVOT_STRAT>
quick_sort (ARRAY &array, PIVOT_STRAT *pivot_strat)
{
for (;;)
{
 ARRAY::TYPE pivot; // typename ARRAY::TYPE pivot...
 pivot = pivot_strat->get_pivot (array, lo, hi);
// Partition array[lo, hi] relative to pivot...
}
}
```

BTD/GL/Pat

18

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Adapter (adaptateur)

- **But (GOF)**
  - ➔ **Convertir l'interface d'une classe en une autre interface que le client souhaite. Adapter (adaptateur) permet aux classes de travailler ensemble ce qu'elles ne pouvaient pas faire à cause de l'incompatibilité des interfaces.**
  - ➔ Client **voudrait** Target **mais dispose de** Adaptee
  - ➔ Adapter **résout ce problème.**

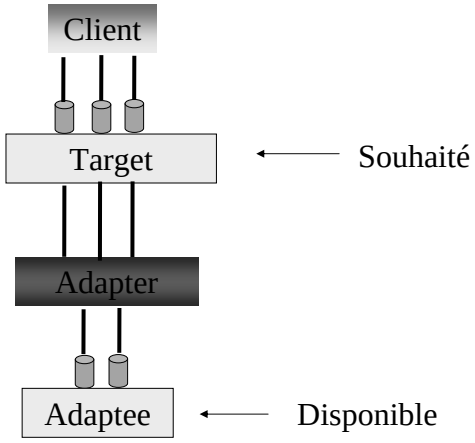
BTD/GL/Pat

19

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Adapter (suite)



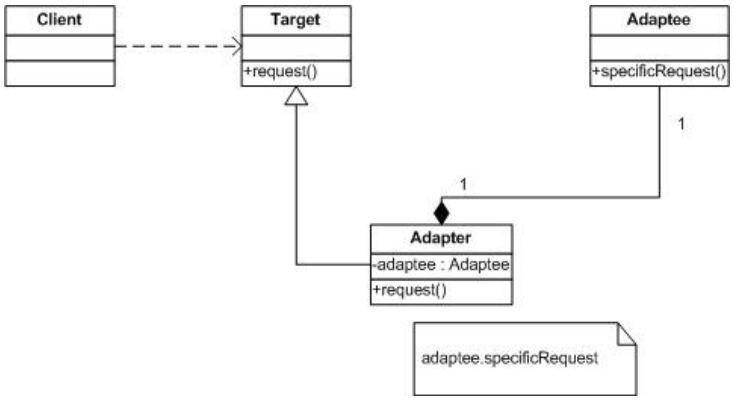
BTD/GL/Pat

20

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Adapter (suite)



BTD/GL/Pat

21

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Adapter (suite)

```
class Target {
  public void request() {}
}

class Adaptee {
  public void specificRequest() {
    System.out.println("Adaptee: SpecificRequest");
  }
}

class Adapter extends Target {
  private Adaptee adaptee;
  public Adapter(Adaptee a) {
    adaptee = a;
  }
  public void request() {
    adaptee.specificRequest();
  }
}
```

BTD/GL/Pat

22

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Adapter (suite)

```
public class Client{

        public void test()
        {
          Adaptee a = new Adaptee();
          Target t = new Adapter(a);
          t.request();
        }

}
```

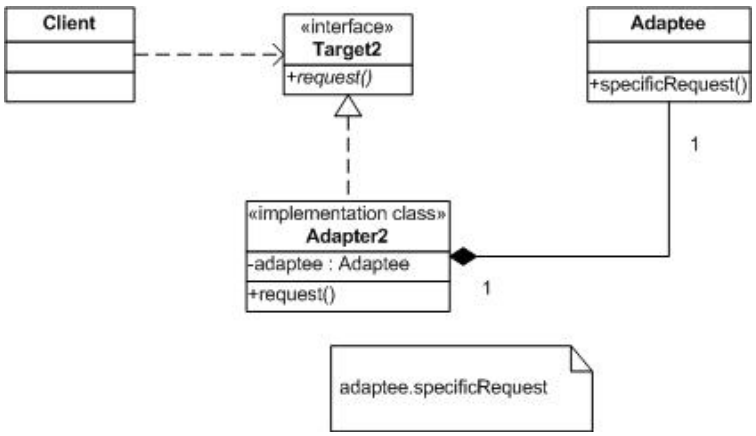
BTD/GL/Pat

23

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Adapter (suite)



BTD/GL/Pat

24

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge

- **But (GOF)**
  - ➔ **Découpler une abstraction de son implémentation pour que les deux puissent évoluer indépendamment.**

BTD/GL/Pat

25

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

- **Utilisation**
  - ➔ **Vous voulez éviter les liens permanents entre une abstraction et son implémentation. Ceci peut être le cas quand l'implémentation doit être sélectionnée à l'exécution.**
  - ➔ **Les deux, les abstractions et leurs implémentations veulent évoluer indépendamment. Dans ce cas le pattern Bridge permet de combiner différentes abstractions et implémentations et les étend indépendamment.**
  - ➔ **Les changements dans l'implémentation d'une abstraction ne devraient pas avoir l'impact sur les clients, c'est-à-dire que leur code ne devrait pas devoir être recompilé.**
  - ➔ **On veut cacher l'implémentation d'une abstraction complètement à l'utilisateur.**

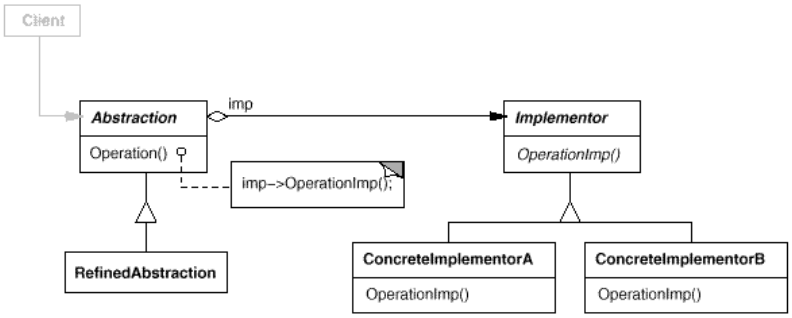
BTD/GL/Pat

26

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)



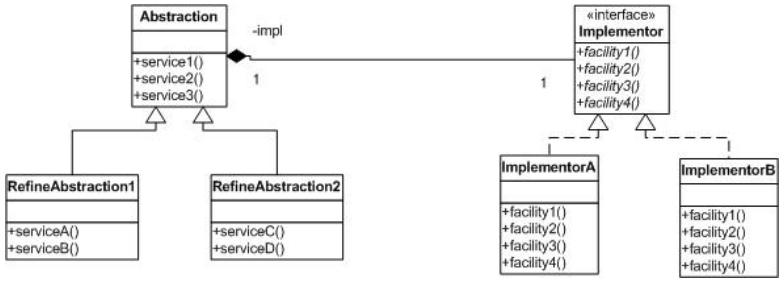
BTD/GL/Pat

27

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)



BTD/GL/Pat

28

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
class Abstraction {
        private Implementor imp;
        public Abstraction(Implementor imp) {
                this.imp = imp;
        }
        public void service1() {
                imp.facility1();
                imp.facility2();
        }

        public void service2() {
                imp.facility2();
                imp.facility3();
        }

        public void service3() {
                imp.facility1();
                imp.facility2();
                imp.facility4();
        }
}
```

BTD/GL/Pat

29

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
class RefineAbstraction1 extends Abstraction {
        public RefineAbstraction1 (Implementor imp) {
                        super(imp);
        }
        public void serviceA() {
                service1();
                service2();
         }

         public void serviceB() {
                service3();
         }
}
```

BTD/GL/Pat

30

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
class RefineAbstraction2 extends Abstraction {
       public RefineAbstraction2 (Implementor imp){
              super(imp);
        }

        public void serviceC() {
              service2();
              service3();
        }

        public void serviceD() {
              service1();
              service3();
        }
}
```

BTD/GL/Pat

31

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
interface Implementor {
           void facility1();
           void facility2();
           void facility3();
           void facility4();
}
```

BTD/GL/Pat

32

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
class ImplementorA implements Implementor {
       public void facility1() {
              System.out.println("
ImplementorA.facility1");
       }

       public void facility2() {
              System.out.println("
ImplementorA.facility2");
       }

       public void facility3() {
              System.out.println("
ImplementorA.facility3");
       }

       public void facility4() {
              System.out.println("
ImplementorA.facility4");
       }
}
```

BTD/GL/Pat

33

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
class ImplementorB implements Implementor {
       public void facility1() {
              System.out.println(" ImplementorB.facility1");
       }

       public void facility2() {
              System.out.println(" ImplementorB.facility2");
       }

       public void facility3() {
              System.out.println(" ImplementorB.facility3");
       }

       public void facility4() {
              System.out.println(" ImplementorB.facility4");
       }
}
```

BTD/GL/Pat

34

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
class Client{
public void test1() {
       RefineAbstraction1 r1 =
              new RefineAbstraction1(new ImplementorA());

       System.out.println("RefineAbstraction1.ServiceA");
       r1.serviceA();
       System.out.println("RefineAbstraction1.ServiceB");
       r1.serviceB();
}

public void test2() {
       RefineAbstraction1 r1 =
              new RefineAbstraction1 (new ImplementorB());

       System.out.println("RefineAbstraction1.ServiceA");
       r1.serviceA();
       System.out.println("RefineAbstraction1.ServiceB");
       r1.serviceB();
}
```

BTD/GL/Pat

35

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
  public void test3() {
       RefineAbstraction2 r2 =
              new RefineAbstraction2(new ImplementorA());
       System.out.println("RefineAbstraction2.ServiceC");
       r2.serviceC();
       System.out.println("RefineAbstraction2.ServiceD");
       r2.serviceD();
}

public void test4() {
       RefineAbstraction2 r2 =
              new RefineAbstraction2 (new ImplementorB());
       System.out.println("RefineAbstraction2.ServiceC");
       r2.serviceC();
       System.out.println("RefineAbstraction2.ServiceD");
       r2.serviceD();
}
}
```

BTD/GL/Pat

36

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
public class Bridge
{
	public static void main(String[] args) {
		Client client = new Client();
		client.test1();
		client.test2();
		client.test3();
		client.test4();
	}
}
```

BTD/GL/Pat

37

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
RefineAbstraction1.ServiceA
 ImplementorA.facility1
 ImplementorA.facility2
 ImplementorA.facility2
 ImplementorA.facility3
RefineAbstraction1.ServiceB
 ImplementorA.facility1
 ImplementorA.facility2
 ImplementorA.facility4
RefineAbstraction1.ServiceA
 ImplementorB.facility1
 ImplementorB.facility2
 ImplementorB.facility2
 ImplementorB.facility3
RefineAbstraction1.ServiceB
 ImplementorB.facility1
 ImplementorB.facility2
 ImplementorB.facility4
```

BTD/GL/Pat

38

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Bridge (suite)

```
RefineAbstraction2.ServiceC
 ImplementorA.facility2
 ImplementorA.facility3
 ImplementorA.facility1
 ImplementorA.facility2
 ImplementorA.facility4
RefineAbstraction2.ServiceD
 ImplementorA.facility1
 ImplementorA.facility2
 ImplementorA.facility1
 ImplementorA.facility2
 ImplementorA.facility4
RefineAbstraction2.ServiceC
 ImplementorB.facility2
 ImplementorB.facility3
 ImplementorB.facility1
 ImplementorB.facility2
 ImplementorB.facility4
RefineAbstraction2.ServiceD
 ImplementorB.facility1
 ImplementorB.facility2
 ImplementorB.facility1
 ImplementorB.facility2
 ImplementorB.facility4
```

BTD/GL/Pat

39

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Façade

- **But (GoF)**
  - ➔ **Fournir une interface unifiée pour un ensemble d'interfaces dans un sous-système.**
  - ➔ **Le pattern Façade définit une interface de plus haut niveau qui conduit à utiliser plus facilement les sous-systèmes.**

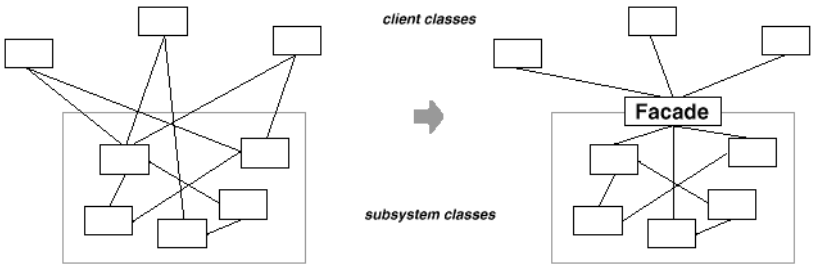
BTD/GL/Pat

40

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Façade (cont'd)



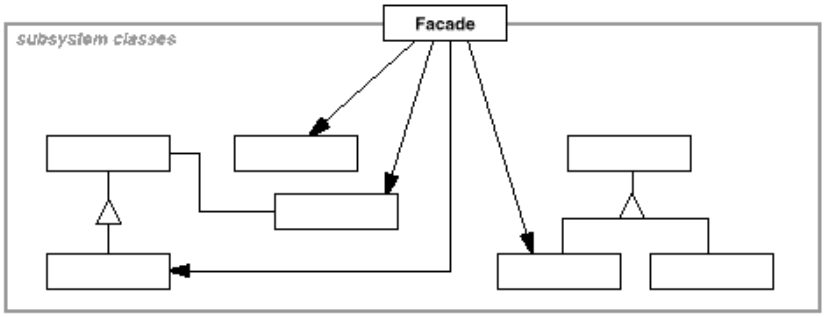
BTD/GL/Pat

41

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Façade (cont'd)



BTD/GL/Pat

42

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Façade (cont'd)

- **Utilisation**
  - ➔ **Les sous-systèmes deviennent souvent plus complexes quand ils évoluent. Une façade peut fournir une simple vue par défaut du sous-système qui est assez bonne pour la plupart des clients. Seulement les clients ayant besoin de plus d'adaptation auront besoin de regarder au-delà de la façade.**
  - ➔ **Il peut y avoir beaucoup de dépendances entre les clients et les implémentations d'une abstraction de classe. L'introduction de la façade permet de découpler le sous-système des clients et d'autres sous-systèmes, de sorte à promouvoir l'indépendance et portabilité du sous-système.**
  - ➔ **Hiérarchiser des sous-systèmes. Utiliser la façade permet de définir des points d'entrée à chaque niveau du sous-système. Si les sous-systèmes sont dépendants, ceci peut simplifier les dépendances entre eux par la communication qui peut se faire via leurs façades.**

BTD/GL/Pat

43

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Flyweight

- **But**
  - ➔ **Utiliser le partage pour supporter un grand nombre d'objets de petit grain efficacement.**

BTD/GL/Pat

44

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Flyweight (suite)

➔ **Un** flyweight **est un objet partagé qui peut être utilisé simultanément dans multiples contextes. Le flyweight agit comme un objet indépendant dans chaque contexte – il est indissociable de l'instance de l'objet qui n'est pas partagé. Flyweights ne peuvent pas prévoir le contexte dans lequel ils agissent.**

➔ **Le concept clé est la distinction entre l'état** intrinsèque **et** extrinsèque**. L'état Intrinsèque est stocké dans le flyweight; il correspond à l'information qui est indépendante du contexte, ce qui la rend partageable. L'état Extrinsèque dépend de et varie avec le contexte et pour cela il ne peut pas être partagé.**

➔ **Les objets clients sont responsables de transmission de l'état extrinsèque à flyweight quand celui-ci en a besoin.**

BTD/GL/Pat

45

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Flyweight (suite)

- Utilisation

  L'efficience du pattern Flyweight dépend principalement de la façon et où il est utilisé. Pour utiliser le pattern Flyweight tout ce qui suit doit être vrai :

  ➔ L'application utilise un grand nombre d'objets.

  ➔ Le coût de stockage est important à cause de la quantité importante de ces objets.

  ➔ L'état dominant est extrinsèque.

  ➔ Nombreux objets peuvent être remplacés par relativement peut d'objets partagés une fois l'état extrinsèque enlevé.

  ➔ L'application ne dépend pas de l'identité des objets. Puisque les objets flyweight peuvent être partagés, les tests d'identité retournent vrai pour les objets conceptuellement distincts**.**

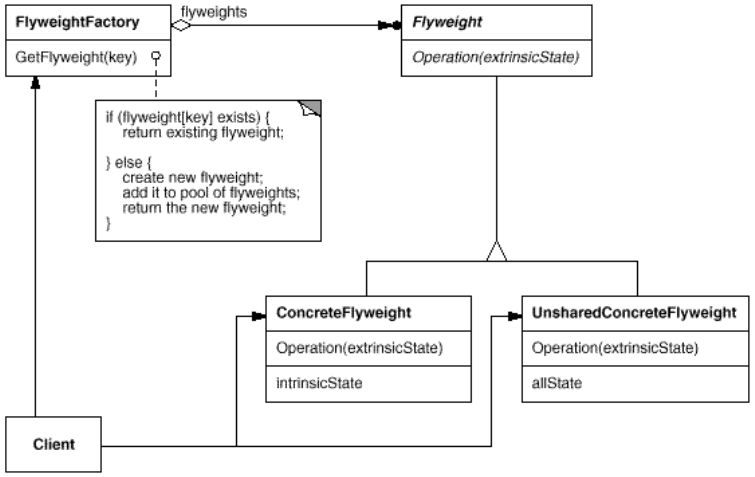
BTD/GL/Pat

46

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Flyweight (suite)



BTD/GL/Pat

47

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Too Many Objects Example

```
class DataPoint {
        private static int count = 0;
        private int id = count++;
        private int i;
        private float f;
        public int getI() { return i; }
        public void setI(int i) { this.i = i; }
        public float getF() { return f; }
        public void setF(float f) { this.f = f; }
        public String toString() {
        return "id: " + id + ", i = " + i + ", f = " + f;
        }
}
```

BTD/GL/Pat

48

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Too Many Objects Example (suite)

```
public class ManyObjects {
        static final int size = 2000000;
        public static void main(String[] args) {
                DataPoint[] array = new DataPoint[size];
                for(int i = 0; i < array.length; i++)
                        array[i] = new DataPoint();

                for(int i = 0; i < array.length; i++) {
                        DataPoint dp = array[i];
                        dp.setI(dp.getI() + 1);
                        dp.setF(47.0f);
                }
                System.out.println(array[size -1]);
        }
}
```

BTD/GL/Pat

49

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Flyweight Example

```
class ExternalizedData {
        static final int size = 5000000;
        static int[] id = new int[size];
        static int[] i = new int[size];
        static float[] f = new float[size];
        static {
                for(int i = 0; i < size; i++)
                id[i] = i;
        }
}
```

BTD/GL/Pat

50

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Flyweight Example (suite)

```
class FlyPoint {
        private FlyPoint() {}
        public static int getI(int obnum) {
                return ExternalizedData.i[obnum];
        }
        public static void setI(int obnum, int i) {
                ExternalizedData.i[obnum] = i;
        }
        public static float getF(int obnum) {
                return ExternalizedData.f[obnum];
        }
        public static void setF(int obnum, float f) {
                ExternalizedData.f[obnum] = f;
        }
```

BTD/GL/Pat

51

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Flyweight Example (suite)

```
        public static String str(int obnum) {
                return "id: " +
                        ExternalizedData.id[obnum] +
                        ", i = " +
                        ExternalizedData.i[obnum] +
                        ", f = " +
                        ExternalizedData.f[obnum];
        }
}
```

BTD/GL/Pat

52

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Flyweight Example (suite)

```
public class FlyWeightObjects {
public static void main(String[] args) {
        for(int i = 0; i < ExternalizedData.size; i++) {
                FlyPoint.setI(i, FlyPoint.getI(i) + 1);
                FlyPoint.setF(i, 47.0f);
        }
        System.out.println(
                FlyPoint.str(ExternalizedData.size -1));
}
}
```

BTD/GL/Pat

53

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Contexte

- **De très nombreux projets logiciels font apparaître des éléments comparables**

  ➔ **Réinventer les meilleures solutions connues dans chaque nouveau projet est inutile et risqué**

- **Analysis Patterns**
- **Design Patterns**
- **Frameworks**
- **Workflow Patterns**

BTD/GL/Pat

54

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Pourquoi les Patterns

- La réussite prime sur la nouveauté
- Importance de la clarté de la documentation
- Validation qualitative des acquis et de la connaissance pratique
- Importance de la dimension humaine dans le développement logiciel

- Faciliter la réutilisation de savoir faire

BTD/GL/Pat

55

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Sur la réutilisation

Les langages informatiques modernes orientés objet permettent la réutilisation

- par importation de classes
- par héritage : extension / spécialisation
- par l'inversion de contrôle (aspects)

BTD/GL/Pat

56

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les patterns c'est quoi?

Design Pattern

=

Schéma de conception réutilisable

=

Organisation et hiérarchie de *plusieurs* modèles de classes réutilisable par simple implémentation, adaptation, extension

BTD/GL/Pat

57

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Comment?

- **Les Design Patterns sont présentés en utilisant la notation graphique standard UML**
- **Pour l'essentiel, seule la partie statique (diagrammes de classes) de UML est utilisée**

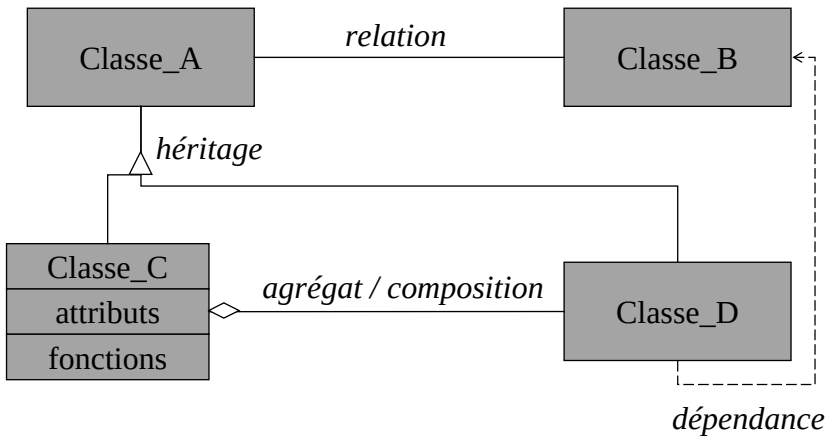
BTD/GL/Pat

58

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Diagrammes de Classes UML2 (éléments)



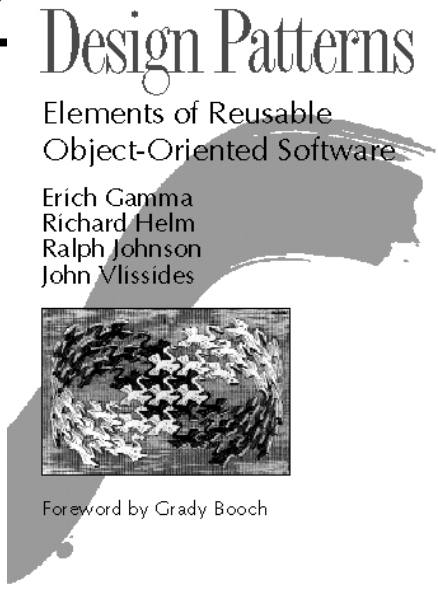
BTD/GL/Pat

59

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Références



BTD/GL/Pat

60

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Références Web

- **http://patterndigest.com/**
- **http://norvig.com/design-patterns**
- **http://www.industriallogic.com/papers/index.html**
- **… google…**

BTD/GL/Pat

61