

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Architecture Logicielle



Les patrons d'interfaces

Florent Nicart

Université de Rouen

2016–2017

Les patrons d'interface

Les patrons d'interface opèrent sur les interfaces publiques des composants du système :

- **Adaptateur (adapter)** : fournit l'interface qu'un client attend en utilisant les services d'une classe dont l'interface est différente.
- **Façade (facade)** : fournit une interface facilitant l'emploi d'un sous système.
- **Composite (composite)** : permet au client de traiter de manière uniforme les objets et leurs composition.
- **Passerelle (bridge)** : découple une classe qui s'appuie sur des opérations abstraites de l'implémentation de ces opérations.

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

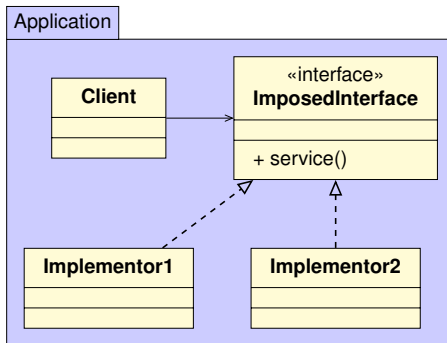
Implémentation

Conclusion

Le patron Adaptateur

Adapter une classe (ou une interface) existante à une interface imposée.

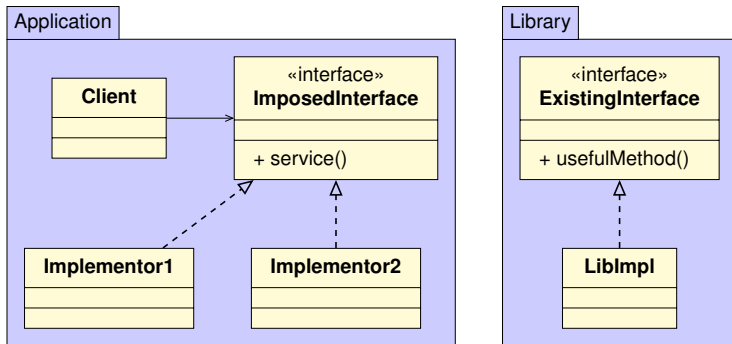
Situation Initiale



- Une application est capable de manipuler des composants logiciels¹ dont le comportement est spécifié par une interface (contrat).

1. Dont le nombre est arbitraire.

Situation Initiale



- On souhaite ajouter un nouveau composant compatible en s'appuyant sur du code existant.
- Problème : bien que la bibliothèque remplisse parfaitement la tâche demandée, l'interface existante n'est pas compatible avec l'interface imposée.

Une première solution

Modifier le code existant

Adaptateur

Motivation

Structure
Exemples Java
Comparaison
Conclusion

Façade

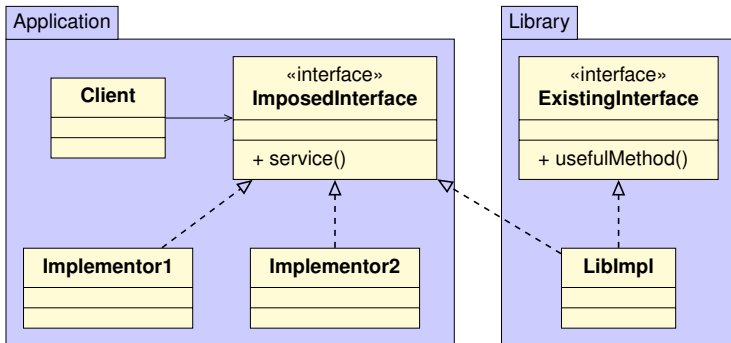
Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

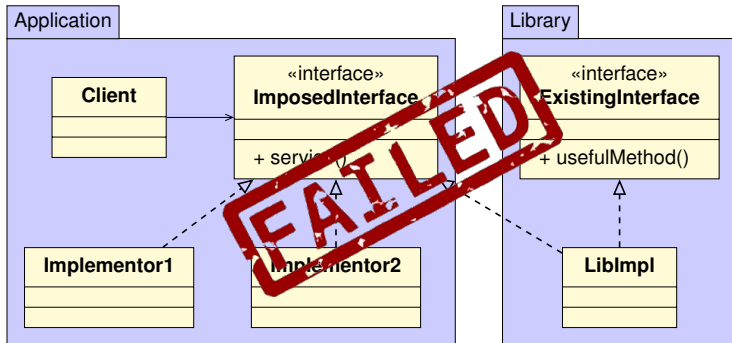
Composite

Motivation
Structure
Exemples
Implémentation
Conclusion



Une première solution

Modifier le code existant



- Ce faisant, nous violons l'**OCP**, le **SRP** et au mieux l'**ISP** (pollution de l'interface publique des implémentateurs de la bibliothèque),

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

Implémentation

Conclusion

Une seconde solution

Adapter l'application

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

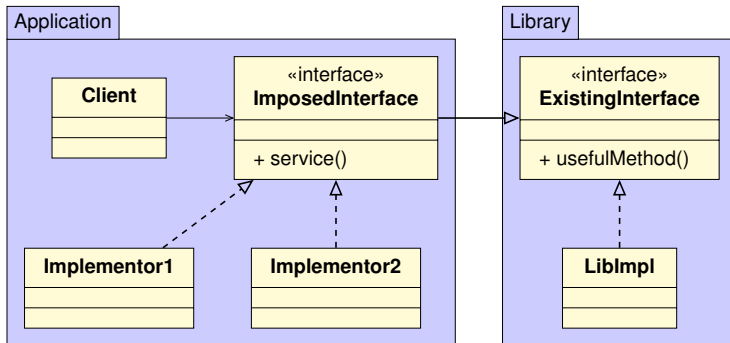
Motivation

Structure

Exemples

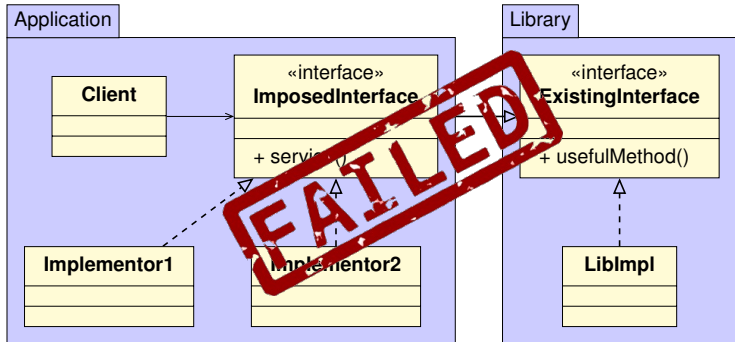
Implémentation

Conclusion



Une seconde solution

Adapter l'application



- Ce faisant, nous violons l'**ISP**, et au mieux l'**IDP** (**ImposedInterface** a été conçue pour les besoins de l'application).

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

Implémentation

Conclusion

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

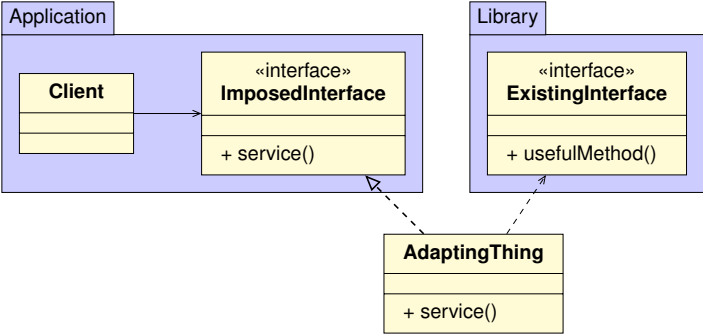
Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Adapter !



- Une solution propre consiste à ajouter un nouveau composant dans l'application qui laissera inchangés les composants et les spécifications existants.
- C'est ce que permet de réaliser le patron **Adapter**.

Le patron **Adapter**

Aussi connu comme Wrapper

Intention

- Adapter une classe existante à une interface imposée.
- Adapter permet à des classes de coopérer alors que leurs interfaces sont incompatibles.

Motivation

- Réutilisation d'une boîte à outils dont l'interface n'est pas compatible avec celle conçue pour l'application.
- La boîte à outils ne peut être modifiée (elle est publique → OCP) ou nous n'avons pas son code source.

Participants du patron **Adapter**

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

Implémentation

Conclusion

- **Client** : Utilise les objets conformément à l'interface `Target`.
- **Target** : Définit l'interface spécifique au domaine du Client.
- **Adaptee** : Interface ou classe existante qui doit être adaptée à l'interface `Target`.
- **Adapter** : Réalisation qui adapte l'interface `Adaptee` à l'interface cible `Target`.

Différent types d'adaptateur

F. Nicart

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

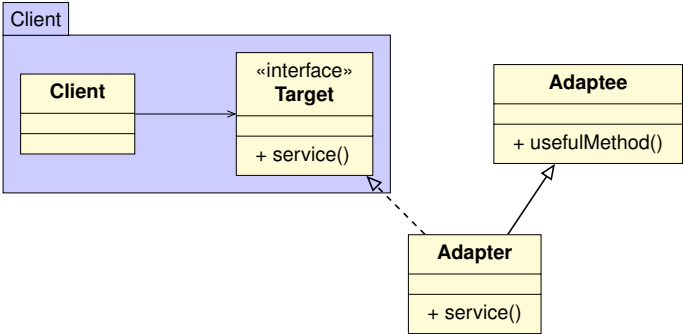
Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- Deux grands types d'adaptateurs : de classe, d'objets.
- Types secondaires : contraint, « Two way ».
- Le schéma de principe diffère légèrement pour chacun, mais le principe global reste le même.
- Chaque type possède des avantages suivant le contexte...

Adaptateur de classe

Schéma de principe



- Dans cette version, c'est la classe qui est adaptée : on instanciera `Adapter` plutôt que `Adaptee`.

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

Implémentation

Conclusion

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

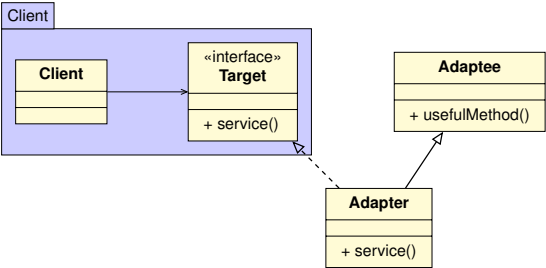
- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

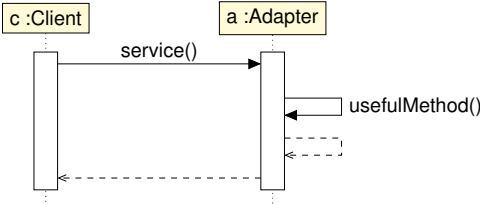
- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Adaptateur de classe

Schéma de principe



À l'exécution, on aura :



Adaptateur de classe

Exemple

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

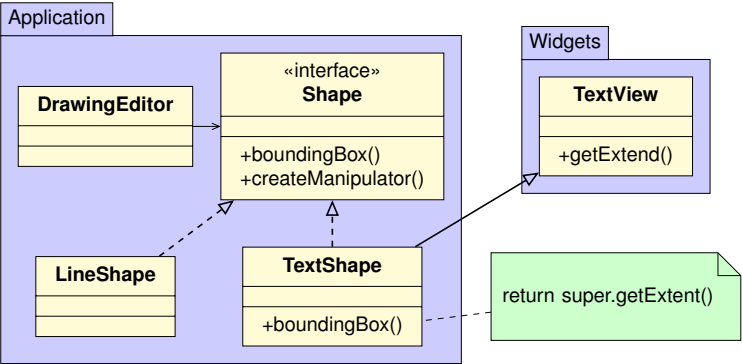
- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

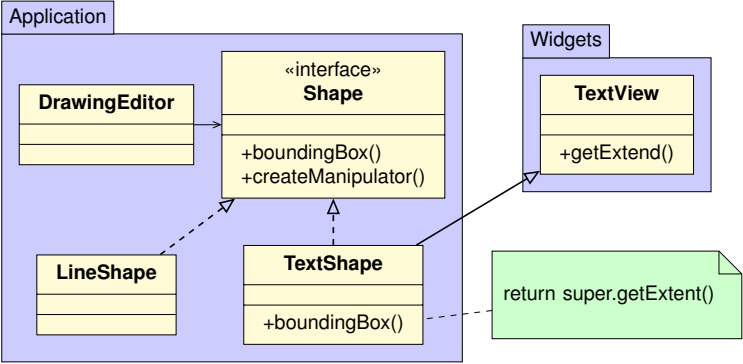
Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion



Adaptateur de classe

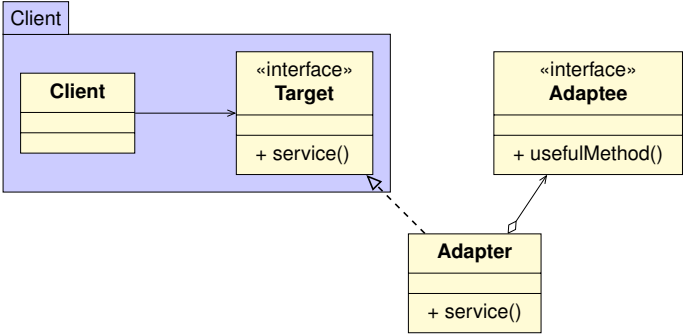
Exemple



- Sauf qu'une forme n'est pas tout à fait une vue !

Adaptateur d'objet

Schéma de principe



- L'adaptateur est une réalisation et une « composition »,
- Adapter déléguera à Adaptee.
- Adaptee peut être une interface cette fois.

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

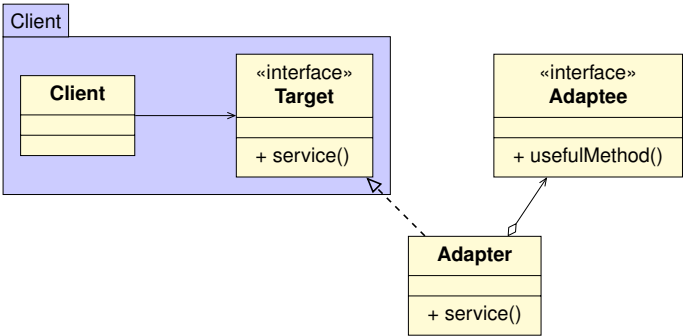
- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

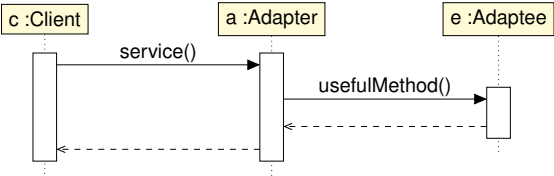
- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Adaptateur d'objet

Schéma de principe

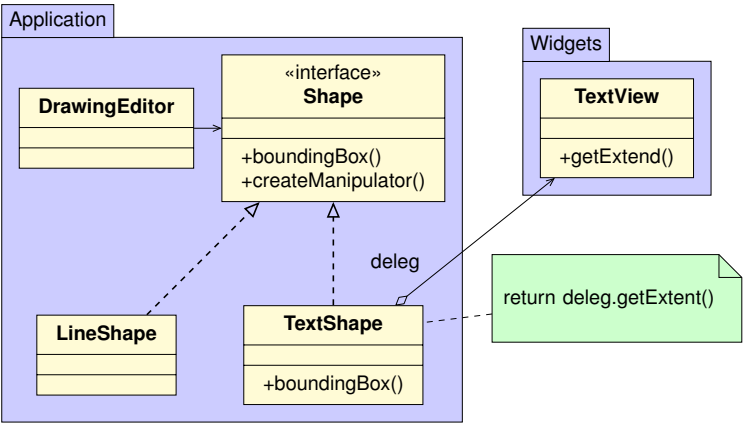


À l'exécution, on aura cette fois :



Adaptateur d'objet

Exemple



- L'adaptateur délègue à l'adapté,
- il est possible de remplacer l'adapté sans impacter le client.

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Adaptateur (d'objet) contraint

Schéma de principe

F. Nicart

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

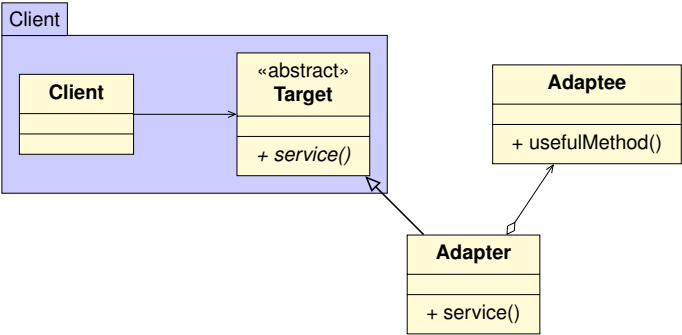
Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- Il peut arriver que l'interface `Target` soit imposée sous forme de classe abstraite (ou non).
- En l'absence d'héritage multiple, la délégation devient obligatoire.



Exemple 1

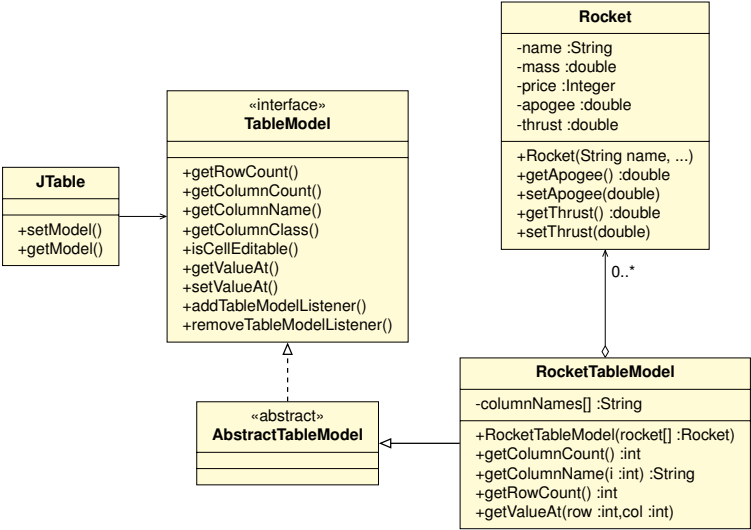
Feux d'artifices

- Une application de vente de feux d'artifice (fusée).
- On dispose de la classe `Rocket` pour représenter une fusée

Rocket
-name :String -mass :double -price :Integer -apogee :double -thrust :double
+Rocket(String name, ...) +getApogee() :double +setApogee(double) +getThrust() :double +setThrust(double)

- On désire naturellement une IHM pour lister les fusées grâce à une JTable :

Name	Price	Apogee
Shooter	\$3.95	50.0
Orbit	\$29.03	5000.0



Exemple 1

Feux d'artifices : code

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

Implémentation

Conclusion

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

```
package fire ;

public class Rocket {
    private String name;      private double mass;
    private Integer price;    private double apogee;
    private double thrust; // pousse

    public Rocket(String name, double mass, Integer price, double apogee,
        double thrust) {
        this.name = name;      this.mass = mass;
        this.price = price;    this.apogee = apogee;
        this.thrust = thrust;
    }
    // The height (in meters) that the rocket is expected to reach.
    public double getApogee() { return apogee; }
    public void setApogee(double value) { apogee = value; }
    // The rated thrust (or force, in newtons) of this rocket.
    public double getThrust() {return thrust;}
    public void setThrust(double value) {thrust = value;}

    public String getName() { return name; }
    public double getMass() { return mass; }
    public Integer getPrice() { return price; }
}
```


Exemple 1

Feux d'artifices : code

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

```
1 package adapter;
2 import javax.swing.table.*;
3 import fire.Rocket;
4
5 // Adapt a collection of rockets for display in a JTable.
6 public class RocketTableModel extends AbstractTableModel {
7     protected Rocket[] rockets;
8     protected String[] columnNames = new String []{ "Name", "Price", "Apogee" };
9
10    // Construct a rocket table from an array of rockets.
11    public RocketTableModel(Rocket[] rockets) {
12        this.rockets = rockets;
13    }
14
15    // Return the number of columns in this table.
16    public int getColumnCount() {
17        return columnNames.length;
18    }
19
20    // Return the name of the indicated column.
21    public String getColumnName(int i) {
22        return columnNames[i];
23    }
24
25    ...

```

Exemple 1

Feux d'artifices : code

```
1      ...
2
3      // Return the number of rows in this table.
4      public int getRowCount() {
5          return rockets.length;
6      }
7
8      // Return the value at the indicated row and column.
9      public Object getValueAt(int row, int col) {
10         switch (col) {
11             case 0 : return rockets[row].getName();
12             case 1 : return rockets[row].getPrice();
13             case 2 : return new Double(rockets[row].getApogee());
14             default : return null;
15         }
16     }
17 }
```

Exemple 1

Feux d'artifices : code

```
1 package FireShop ;
2 import javax.swing.adapter.* ;
3 import fire.Rocket ;
4
5 // Demonstration class.
6 public class ShowRocketTable {
7     private static RocketTableModel getRocketTable() {
8         Rocket r1 = new Rocket("Shooter", 1.0, new Integer(395), 50.0, 4.5) ;
9         Rocket r2 = new Rocket("Orbit", 2.0, new Integer(2903), 5000, 3.2) ;
10        return new RocketTableModel(new Rocket[] { r1, r2 }) ;
11    }
12
13    // Display a Swing component.
14    public static void display(Component c, String title) {
15        JFrame frame = new JFrame(title) ;
16        frame.getContentPane().add(c) ;
17        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) ;
18        frame.pack() ;
19        frame.setVisible(true) ;
20    }
21
22    public static void main(String[] args) {
23        JTable table = new JTable(getRocketTable()) ;
24        JScrollPane pane = new JScrollPane(table) ;
25        pane.setPreferredSize(new java.awt.Dimension(300, 100)) ;
26        display(pane, "Rockets") ;
27    }
28 }
```

Exemple 2

Gestionnaire d'événements

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

- Soit le code suivant :

```
1  ^^|public class ButtonDemo {
2  ^^|    public ButtonDemo() {
3  ^^|        Button button = new Button("Press_me");
4  ^^|        button.addActionListener(new ActionListener() {
5  ^^|            public void actionPerformed(ActionEvent e) {
6  ^^|                doOperation();
7  ^^|            }
8  ^^|        });
9  ^^|    }
10 ^^|    public void doOperation() {
11 ^^|        // whatever
12 ^^|    }
13 ^^|}
14 ^^|
```

Exemple 2

Gestionnaire d'événements

- Soit le code suivant :

```
1  ^^|public class ButtonDemo {
2  ^^|    public ButtonDemo() {
3  ^^|        Button button = new Button("Press_me");
4  ^^|        button.addActionListener(new ActionListener() {
5  ^^|            public void actionPerformed(ActionEvent e) {
6  ^^|                doOperation();
7  ^^|            }
8  ^^|        });
9  ^^|    }
10 ^^|    public void doOperation() {
11 ^^|        // whatever
12 ^^|    }
13 ^^|}
14 ^^|
```

- **Question** : où est le patron adaptateur² ?

2. Pour commencer où est la classe adaptateur ?

Exemple 2

Gestionnaire d'événements

- Soit le code suivant :

```
1  ^^|public class ButtonDemo {
2  ^^|    public ButtonDemo() {
3  ^^|        Button button = new Button("Press me");
4  ^^|        button.addActionListener(new ActionListener() {
5  ^^|            public void actionPerformed(ActionEvent e) {
6  ^^|                doOperation();
7  ^^|            }
8  ^^|        });
9  ^^|    }
10 ^^|    public void doOperation() {
11 ^^|        // whatever
12 ^^|    }
13 ^^|}
14 ^^|
```

- *Indice* : la syntaxe `new ActionListener() { ... }` correspond à l'instanciation d'une classe anonyme obtenue par héritage depuis `ActionListener`.

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

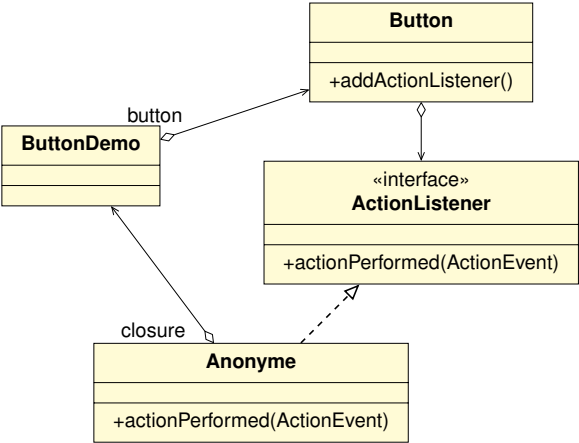
- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Exemple 2

Gestionnaire d'événements



Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

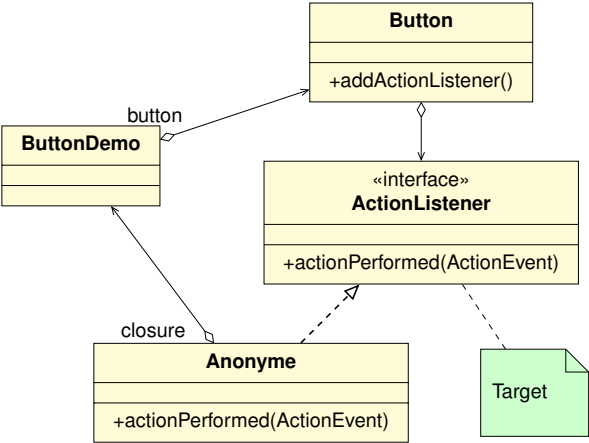
- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

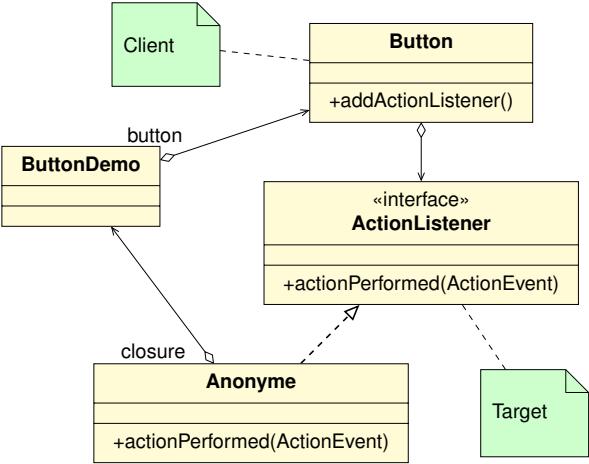
Exemple 2

Gestionnaire d'événements



Exemple 2

Gestionnaire d'événements



Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Exemple 2

Gestionnaire d'événements

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

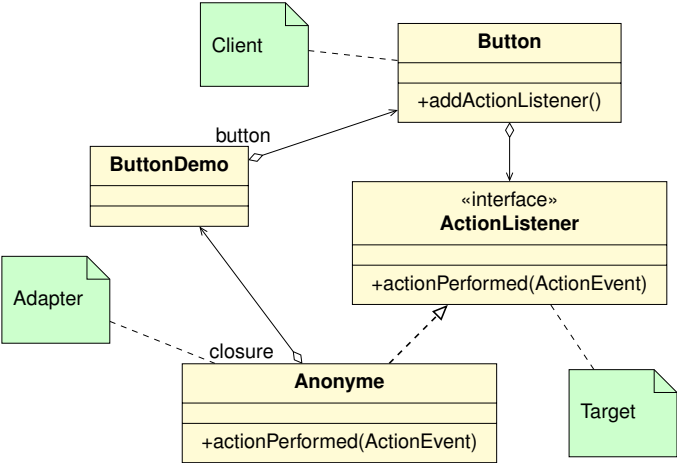
- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

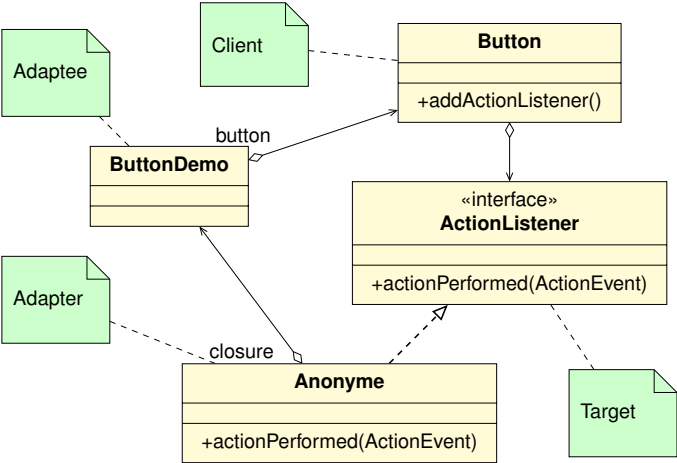
Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion



Exemple 2

Gestionnaire d'événements



Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Exemple 2

Gestionnaire d'événements

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

Implémentation

Conclusion

- Les participants dans le code :

```
1  ^^|public class ButtonDemo {^^|// ADAPTEE
2  ^^|    public ButtonDemo() {
3  ^^|        // CLIENT :
4  ^^|        Button button = new Button("Press_me");
5
6  ^^|        button.addActionListener(
7  ^^|            // ADAPTER : anonymous
8  ^^|            new ActionListener() { // TARGET : implicit inherit
9  ^^|                public void actionPerformed(ActionEvent e) {
10 ^^|                    // A closure is hapening here :
11 ^^|                    doOperation();
12 ^^|                }
13 ^^|            });
14 ^^|    }
15 ^^|    public void doOperation() {
16 ^^|        // whatever
17 ^^|    }
18 ^^|}
19 ^^|
```

Exemple 3

Auditeurs AWT

- Les interfaces de type auditeur (listener) de l'AWT possèdent plusieurs méthodes qui doivent toutes être réalisées par un écouteur d'événements.

«interface» WindowListener
+windowActivated(WindowEvent e) +windowClosed(WindowEvent e) +windowClosing(WindowEvent e) +windowDeactivated(WindowEvent e) +windowDeiconified(WindowEvent e) +windowIconified(WindowEvent e) +windowOpened(WindowEvent e)

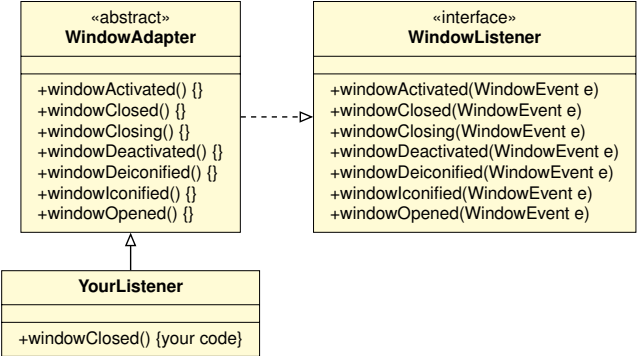
Par exemple l'interface `WindowListener` possède 7 méthodes.

Dans la plupart des cas seules quelques méthodes présentent un réel intérêt, comme celle qui guette l'événement `WindowClosing`.

Exemple 3

Auditeurs AWT

- La bibliothèque de Sun propose des classes comme WindowAdapter qui implémente WindowListener avec des définitions de méthodes vides.



F. Nicart

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

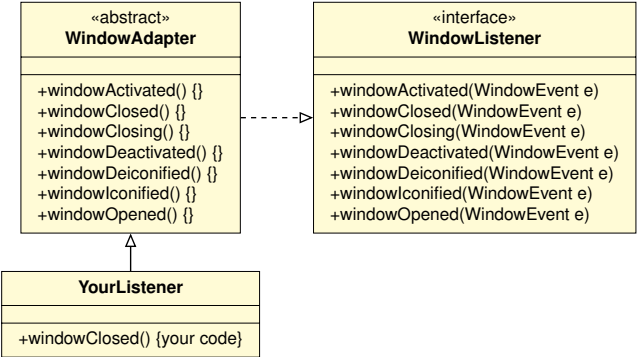
Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Exemple 3

Auditeurs AWT



Faux Ami !

- **WindowAdapter** est une souche et non un adaptateur au sens du patron.
- Il n'adapte pas une interface à une autre.

Comparaison Adaptateur de classe vs d'objet

Généricité

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

L'adaptateur de classe adapte une et une seule classe :

```
1 public class Adapter implements Target extends Adaptee { ...
```

L'adaptateur d'objet adapte toute classe dérivée ou implémenteurs :

```
1 public class Adapter implements Target {  
2     private Adaptee delegate ;  
3  
4     public Adapter(Adaptee theAdaptee) { delegate=theAdaptee ; }  
5     ...
```

- Object wins !

Comparaison Adaptateur de classe vs d'objet

Redéfinitions

F. Nicart

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

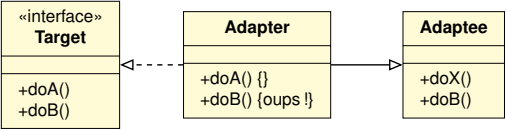
Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- L'adaptateur de classe peut redéfinir des méthodes héritées de l'adapté :

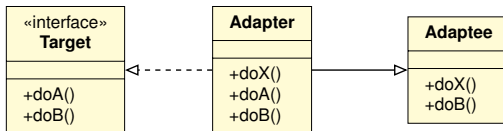


- Puisque les interfaces peuvent être proches, il est possible de redéfinir par accident une méthode de l'adapté (voir chapitre 1).
- L'adaptateur d'objet respecte l'encapsulation.
- Object wins !

Comparaison Adaptateur de classe vs d'objet

Opacité

- L'interface publique de l'adaptateur de classe contient les méthodes de l'interface imposée mais aussi celles héritées de l'adapté :



- Même si l'ISP n'est pas violé, l'interface de l'adaptateur est pollué par des synonymes d'opérations.
- Le client peut être « Tenté » d'utiliser cette connaissance visible.
- L'adaptateur d'objet cache ces détails et rend le découplage complet.
- Object wins !

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

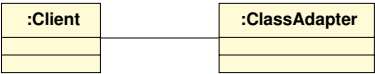
Implémentation

Conclusion

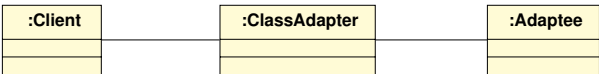
Comparaison Adaptateur de classe vs d'objet

Nombre d'objets à l'usage

- L'instance de l'adaptateur de classe se substitue à celle de l'adapté :



- Celle de l'adaptateur d'objet s'ajoute à celle de l'adapté :

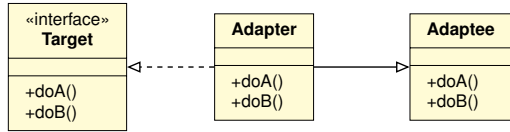


- Une instance peut être économisée si le client gère l'instanciation lui-même.
- Class wins ! (mais attention aux interfaces)

Comparaison Adaptateur de classe vs d'objet

Ajout d'interface

- L'adaptateur de classe peut être utilisé pour nommer simplement une interface déjà implémentée par l'adapté :

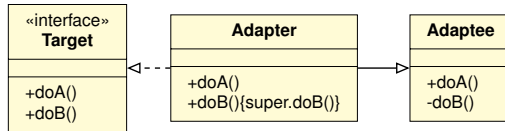


- En effet, mais si les méthodes de la cible sont intégralement implémentées dans l'adaptée, seul la déclaration `implements Target` le rend compatible avec cette interface.
- L'adaptateur de classe corrige ce problème sans ajouter d'objet (si l'instanciation est gérée par le client).
- Class wins !

Comparaison Adaptateur de classe vs d'objet

Changement de visibilité

- L'adaptateur de classe peut également suffire pour changer la visibilité d'une méthode (mais seulement de protégé vers publique) :



- Note : ce sont deux méthodes différentes, il est tout de même nécessaire de déléguer.
- Class wins !

Comparaison Adaptateur de classe vs d'objet

Contrainte d'instanciation

Adaptateur

Motivation

Structure

Exemples Java

Comparaison

Conclusion

Façade

Motivation

Structure

Exemples

Conclusion

Pont

Motivation

Structure

Exemples

Considérations

Conclusion

Composite

Motivation

Structure

Exemples

Implémentation

Conclusion

- Seul l'adaptateur d'objet peut être employé si le client ne gère pas l'instanciation lui-même.
- Ce peut être le cas si l'adapté est obtenu par une *Factory* (cf chapitre 5) :

```
1      public test() {  
2          Vessel m = Game.getVesselFactory().createVessel();  
3          Sprite s=new VesselSpriteAdapter(m);  
4          addToScene(s);  
5      }
```

- ou par paramètre :

```
1      public test(Vessel m) {  
2          addToScene(new VesselSpriteAdapter(m));  
3      }
```

- Object wins !

Principes respectés

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- **O.C.P.** : le code réutilisé n'est pas modifié, les bibliothèques sont conservées intactes et continueront de fonctionner avec fiabilité avec le reste du système.
- **D.I.P.** : `Target` est une abstraction issue de l'univers « métier » du client, `Adaptee` une abstraction de bas niveau fournie par la boîte à outils. Conserver ces deux interfaces intactes contribue à respecter le **D.I.P.**.
- **I.S.P.** : en conservant les interfaces `Target` et `Adaptee` séparées, on évite la pollution d'interface et l'on respecte l'**I.S.P.**.

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Le patron Façade

Fournir une classe facilitant l'accès à un sous système.

F. Nicart

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

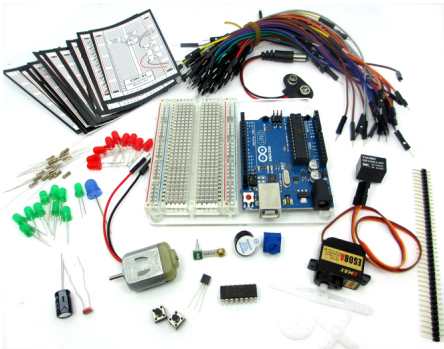
Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Motivations

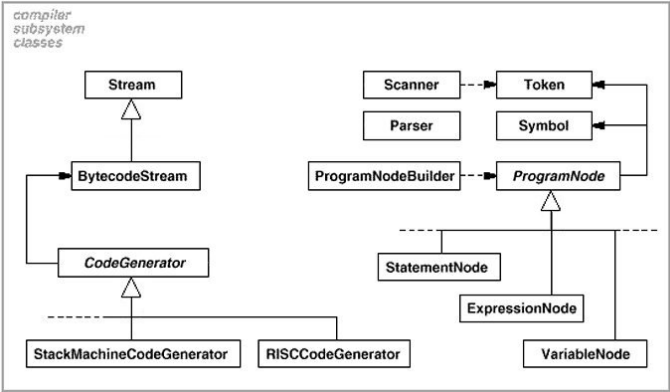
Vive le S.R.P

- Le principe de modularité permet de concevoir des composants réutilisables à l'infini.
- Une modularité extrême peut toutefois rendre difficile l'utilisation du système :



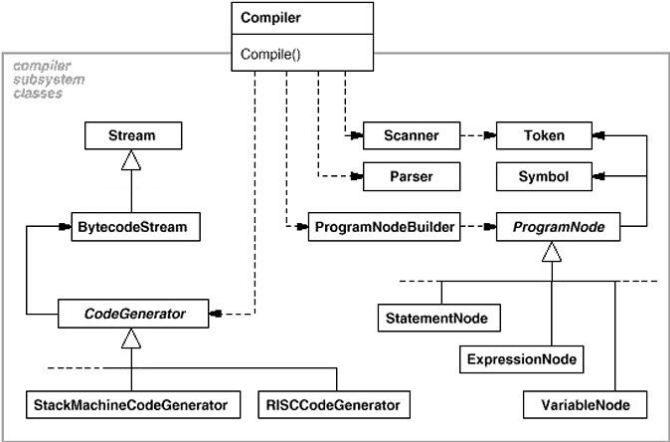
Motivations

- Voici le nécessaire pour compiler un programme :



- Amusez-vous bien ...

Motivations



- Merci ! C'est bien mieux !

Le patron **Façade**

Aussi connu comme
Utilitaires, Démonos

Intention

Fournir une interface/classe facilitant l'emploi d'un sous système (bibliothèque de composants).

Motivation

- Fournir du code de *démonstration* de l'utilisation de la bibliothèque.
- Répondre à un cas d'utilisation identifié de la bibliothèque (*façade*).
- Fournir une collection de méthodes de classe (statiques), dans ce cas, on l'appelle *utilitaire*.

Participants du patron **Façade**

F. Nicart

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

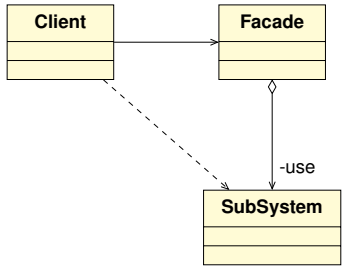
Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- **Facade** : Classe qui offre une interface simplifiée d'accès aux sous-système.
- **Subsystem** : Ensemble des composants de la bibliothèque intervenant dans la réalisation du service rendu par la façade.

Patron façade

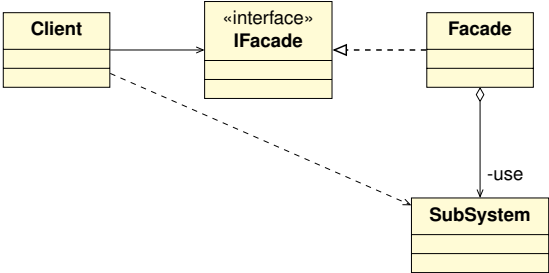
Schéma de principe



- Le client s'adresse à une classe pour manipuler l'ensemble du sous-système.
- Le client peut toutefois accéder directement à des éléments du sous-système.

Patron façade

Schéma de principe



- Idéalement, le client utilise la façade à travers une interface (un contrat).

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

F. Nicart

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

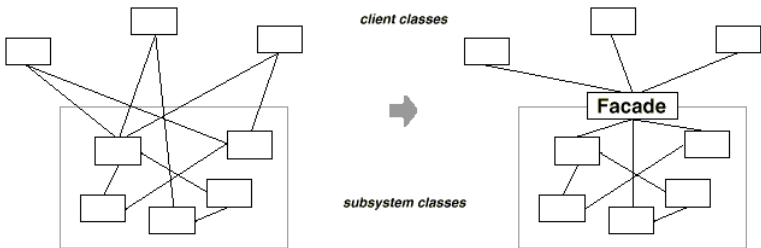
Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Patron façade

Avantages

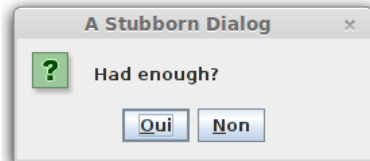


- Utilisation simplifiée de la bibliothèque.
- Réduit le nombre de points d'entrée de la bibliothèque.
- Découpler les clients par rapport aux/(une partie des) composants du sous-système.
- Répondre à un cas d'utilisation sans créer de couplage entre les composants et ce cas.

Exemple de façade

Dans l'API Swing

Si l'on souhaite obtenir ce genre de dialogue :



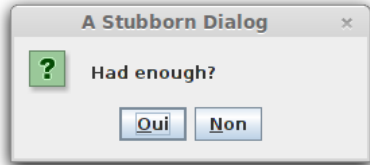
Alors il sera nécessaire d'employer :

- une JFrame,
- deux boutons,
- une icône,
- une classe anonyme(ou non) pour gérer les évènements et retourner le résultat,
- et écrire le code pour lier tout cela ...

Exemple de façade

Dans l'API Swing

Ce problème est récurrent et identifié lors de la conception de la bibliothèque :



Celle-ci fourni donc une façade pour répondre à ce cas d'utilisation :

1
2
3
4
5
6
7
8
9
10
11

```
import javax.swing.* ;  
  
public class ShowOptionPane {  
    public static void main( String[] args) {  
        int option ;  
        do {  
            option = JOptionPane.showConfirmDialog( null , "Had enough?" ,  
                "A Stubborn Dialog" , JOptionPane.YES_NO_OPTION) ;  
        } while ( option == JOptionPane.NO_OPTION) ;  
    }  
}
```

Principes respectés

- **I.S.P.** et S.R.P : permet conserver une mdulation fine des composants de la bibliothèque tout en accompagnant celle-ci d'une solution pour un problème particulier
- **O.C.P.** : le client est découplé de l'architecture retenue pour la solution à ce problème. Celle-ci pourra évoluer sans impact.
- **D.I.P.** : le découplage entre le code d'une solution et les composants utilisés conserve le découplage des couches. Une façade peut même être fournie dans un paquetage séparé.
- Une façade peut être configurable, en particulier lorsqu'elle est utilisée à travers une interface.

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

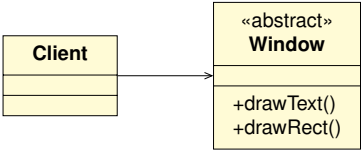
Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Le patron Pont/Passerelle (*Bridge*)

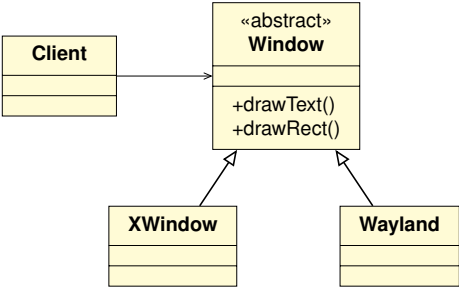
Découpler une abstraction de son implémentation de telle sorte que les deux peuvent varier indépendamment.

Situation initiale



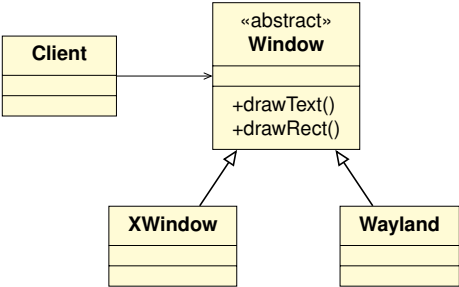
- **Réalisation d'un système de fenêtre**
- support des systèmes graphiques `XWindow` et `Wayland`
- La spécialisation peut apparaître comme une bonne approche...

Situation initiale



- Réalisation d'un système de fenêtre
- support des systèmes graphiques XWindow et Wayland
- La spécialisation peut apparaître comme une bonne approche...

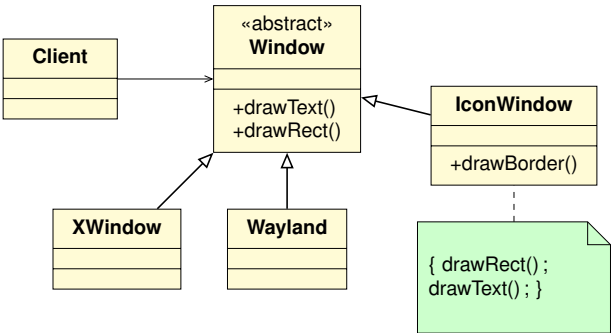
Situation initiale



- Réalisation d'un système de fenêtre
- support des systèmes graphiques `XWindow` et `Wayland`
- La spécialisation peut apparaître comme une bonne approche...

Situation initiale

Acte 2



- Ajout d'un nouveau type de fenêtre : `IconWindow`
- Là aussi, la spécialisation peut apparaître comme une bonne approche...
- ...avec toujours le support de `XWindow` et `Wayland`.

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

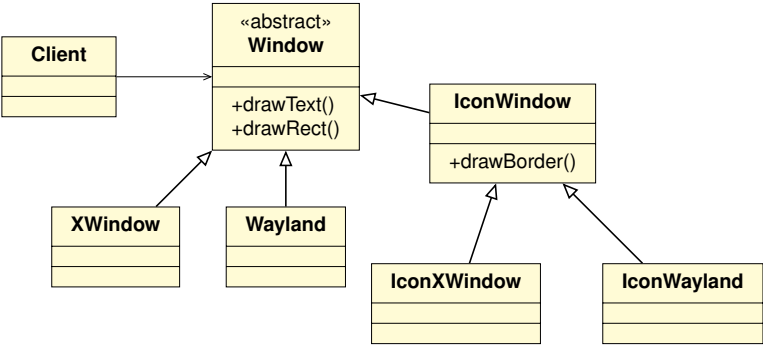
Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Situation initiale

Acte 2



- Ajout d'un nouveau type de fenêtre : `IconWindow`
- Là aussi, la spécialisation peut apparaître comme une bonne approche...
- ...avec toujours le support de `XWindow` et `Wayland`.

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

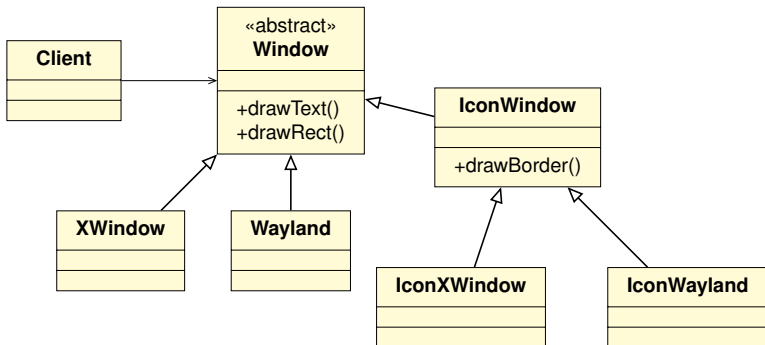
Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Situation initiale

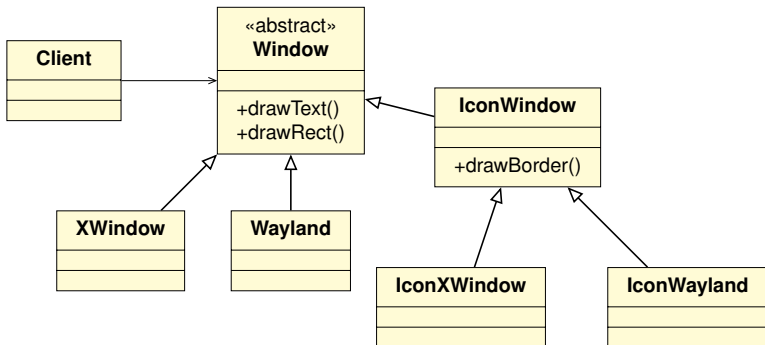
Problème



- L'arborescence mélange différents niveaux : abstraction (Window et IconWindow) et implémentation (XWindow et Wayland). ~ D.I.P.
- L'ajout d'une abstraction (resp. implémentation) nécessite l'écriture de $|Imp|$ (resp. $|Abs|$) classes !

Situation initiale

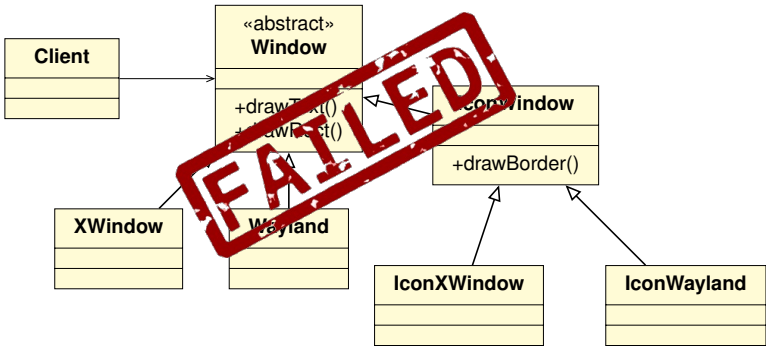
Problème



- L'arborescence mélange différents niveaux : abstraction (Window et IconWindow) et implémentation (XWindow et Wayland). ~ D.I.P.
- L'ajout d'une abstraction (resp. implémentation) nécessite l'écriture de $|Imp|$ (resp. $|Abs|$) classes !

Situation initiale

Problème



- L'arborescence mélange différents niveaux : abstraction (Window et IconWindow) et implémentation (XWindow et Wayland). ~ D.I.P.
- L'ajout d'une abstraction (resp. implémentation) nécessite l'écriture de $|Imp|$ (resp. $|Abs|$) classes !

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Le patron Passerelle/pont/Bridge

Aussi connu comme
Handle, Body

Intention

Découpler une abstraction de son implémentation de telle sorte que les deux peuvent varier indépendamment.

Motivation

- Éviter un couplage permanent des abstractions et des implémentations.
- Pouvoir ajouter des abstractions et des implémentations indépendamment.
- Garanti une séparation en couche et le respect du **D.I.P..**

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Participants du patron Passerelle

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

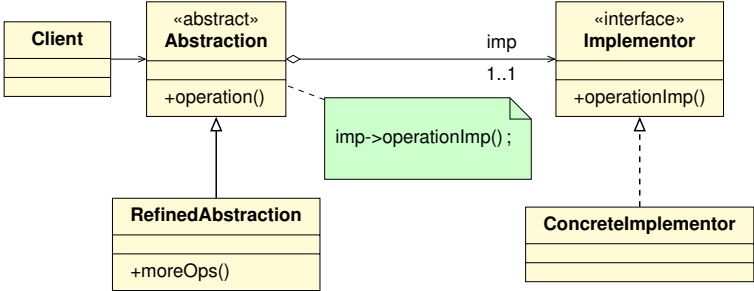
Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- **Abstraction** : définit l'interface abstraite et maintient une référence à un objet de type `Implementor`
- **RefinedAbstraction** : étend l'interface définie par `Abstraction`.
- **Implementor** : définit l'interface des classes d'implémentation.
- **ConcreteImplementor** : réalise l'interface `Implementor`

Patron passerelle

Schéma de principe



- Le client manipule indifféremment les abstractions,
- lesquels s'appuient indifféremment sur les implémentations.

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

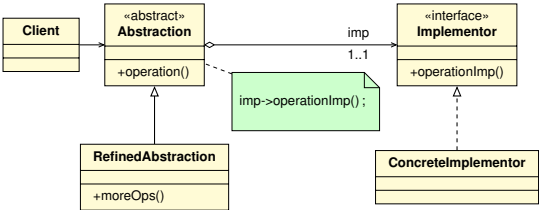
Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

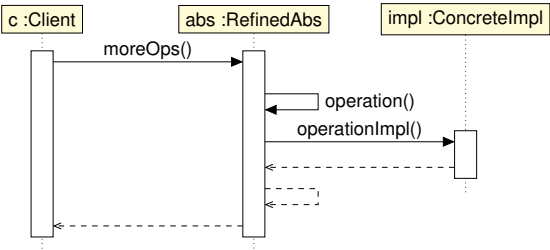
F. Nicart

Patron passerelle

Schéma de principe



- Les abstractions délèguent aux implémentations :



Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

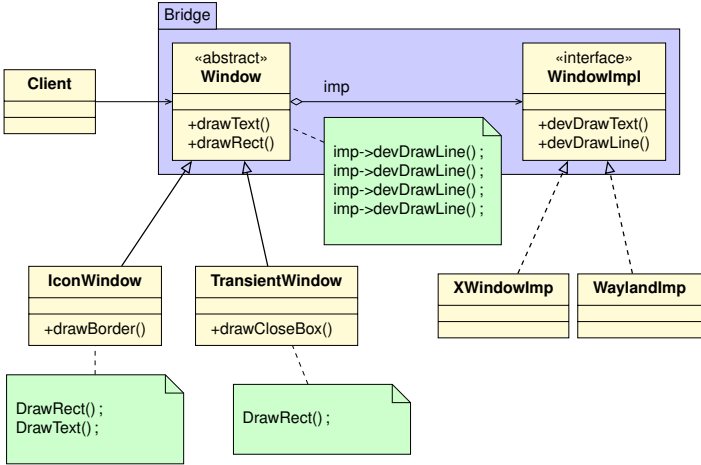
Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Patron passerelle

Application à l'exemple



Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion

Patron passerelle

Application à l'exemple

- Les systèmes à base de pilotes (*drivers*) constituent des passerelles :

```
1  ...
2  try {
3      // Load the driver :
4      Class.forName("com.mysql.jdbc.Driver").newInstance();
5
6      // Create a connection, driver selected from the url :
7      Connection conn = DriverManager.getConnection(
8          "jdbc:mysql://localhost/test?" + "user=monty&password=python");
9
10     // Subsequent operation will be realized by mysql implementor :
11     ResultSet rs = stmt.executeQuery("SELECT foo FROM bar");
12     catch (Exception e) {...}
13     ...
```

Considérations d'implémentation

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- Un seul implémenteur : cas dégénéré de passerelle. Toutefois, protège le client par rapport au changement ou à l'extensibilité.
- Où et quand l'implémenteur doit-il être créé ? Externaliser sa création par rapport aux abstractions.
- Les abstractions ne doivent pas connaître les implémenteurs : utiliser des `Factories`.

Relations avec d'autres patrons

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

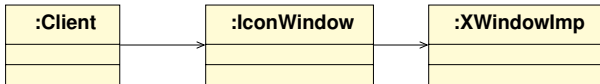
Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- Abstract Factory peut être utilisée pour instancier et configurer un pont.
- Bridge ressemble à adapter :



- Cependant adapter est utilisé pour adapter des interfaces à posteriori,
- Bridge sépare les abstractions des implémentations d'un même concept !

Patron passerelle

Bénéfices

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- Les implémentations ne sont plus liées de manière permanente aux abstractions.
- Réduction des dépendances (compilation)
- Favorise la conception en couches.
- Extensibilité améliorée : l'ajout d'abstractions et d'implémenteurs peut se faire individuellement et indépendamment.
- Isole d'avantage le client des détails d'implémentations.

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

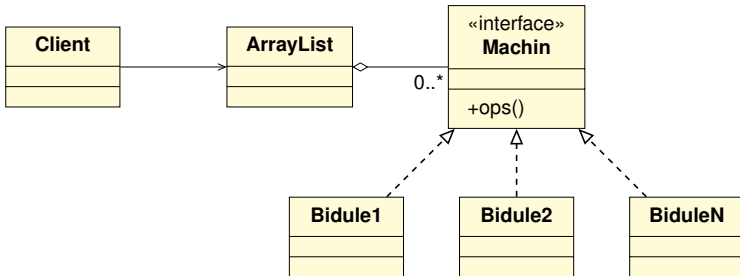
Motivation
Structure
Exemples
Implémentation
Conclusion

Le patron Composite

Uniformiser le traitement d'objets et de lots d'objets.

Uniformisation de traitement

Uniformiser le traitement d'objets et de lots d'objets.

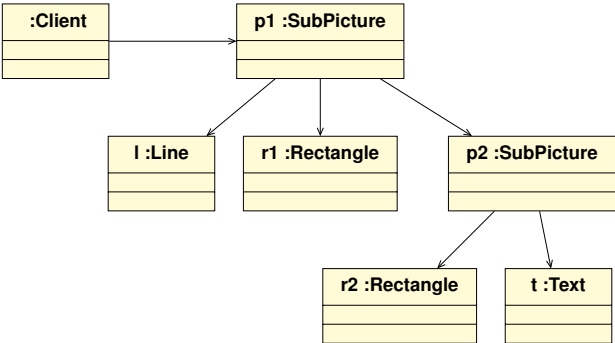


- Le paradigme objet offre déjà l'uniformisation de traitement sur des collections d'objets de différents types :
- le polymorphisme (en particulier d'interface).
- Ce n'est donc pas cela que l'on entend ici par *lots*.

Situation initiale

Logiciel de dessin vectoriel

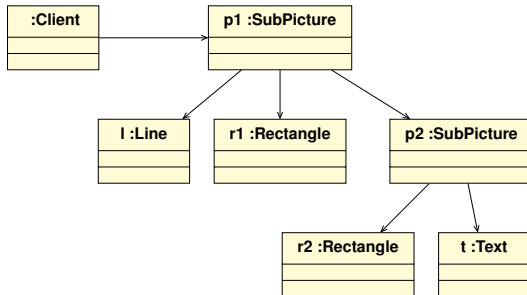
- La problématique de lots intervient lorsque l'un des objets est lui même un conteneur (un agrégat) de ces différents types d'objets (y compris de lui même) :



- Structure non linéaire (arbre/graphe).

Situation initiale

Logiciel de dessin vectoriel



Plus précisément lorsque le conteneur :

- induit une composition récursive (il fait parti des types d'objets qu'il peut contenir),
- l'agrégat qu'il compose devra pouvoir être traité comme un seul objet sans que le client s'en rende compte (uniformisation).
- Le patron *composite* formalise cette situation.

Le patron **Composite**

Aussi connu comme Composite

Intention

Permettre à un client de traiter des objets individuels et des compositions d'objets uniformément.

Motivation

- Composer des objets sous forme d'une structure d'arbre (voire de graphe).
- Cacher au client les différences entre les nuds et les feuilles.
- Uniformiser les traitements entre les objets simples et les composés.

Participants du patron **Composite**

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

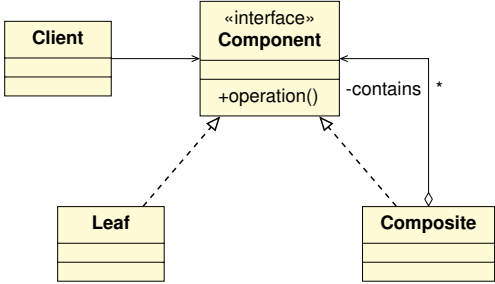
Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

- **Client** : le code qui doit bénéficier d'une vision uniforme de la structure et de ses éléments.
- **Component** : l'interface qui déclare les opérations uniformes (applicables sur les nuds et les composites).
- **Leaf** : les objets simples (les feuilles de l'arbre).
- **Composite** : définit les composants qui peuvent avoir des fils (nuds internes).

Patron composite

Schéma de principe



- Le client manipule indifféremment les éléments de l'arbre en tant que *Component*,
- lesquels peuvent être composés (*Composite*).

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

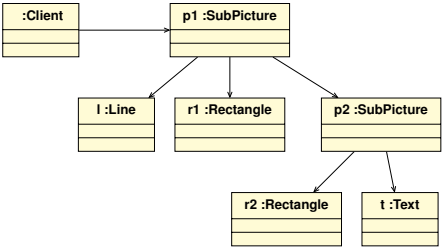
Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Exemple graphique

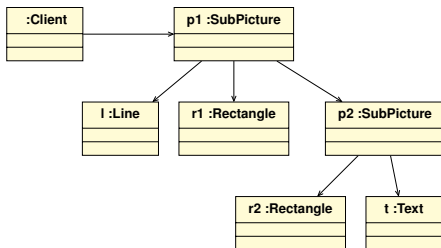


Le client doit pouvoir appliquer les opérations suivantes indifféremment sur les composants simples et les sous-images :

- `draw()` : provoque le dessin,
- `scale(s:float)` : mise à l'échelle,
- `move(int dx,int dy)` : déplacement de (dx,dy),
- `setColor(c:Color)` : définit la couleur de tracé.

Que faut-il faire ? ...

Exemple graphique



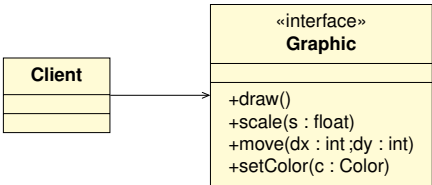
Le client doit pouvoir appliquer les opérations suivantes indifféremment sur les composants simples et les sous-images :

- `draw()` : provoque le dessin,
- `scale(s:float)` : mise à l'échelle,
- `move(int dx, int dy)` : déplacement de (dx,dy),
- `setColor(c:Color)` : définit la couleur de tracé.

Que faut-il faire ? ...

Exemple graphique

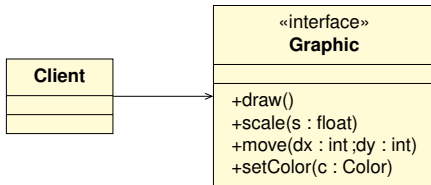
Standardiser le vocabulaire des opérations :



- L'interface *Component* (ici *Graphic* spécifie le contrat entre le client et le reste de la structure.
- Cette vision doit être uniforme ...

Exemple graphique

Standardiser le vocabulaire des opérations :



- L'interface *Component* (ici *Graphic* spécifie le contrat entre le client et le reste de la structure.
- Cette vision doit être uniforme ...

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

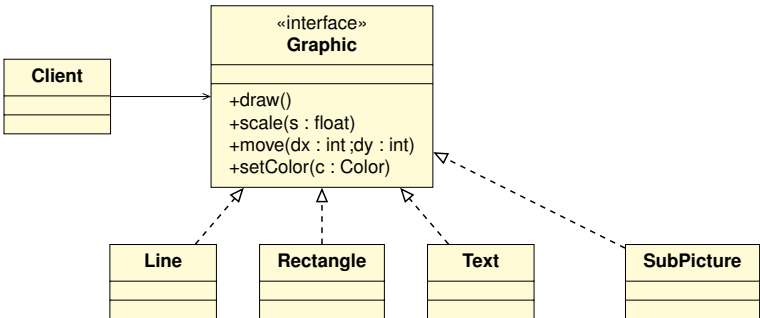
Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

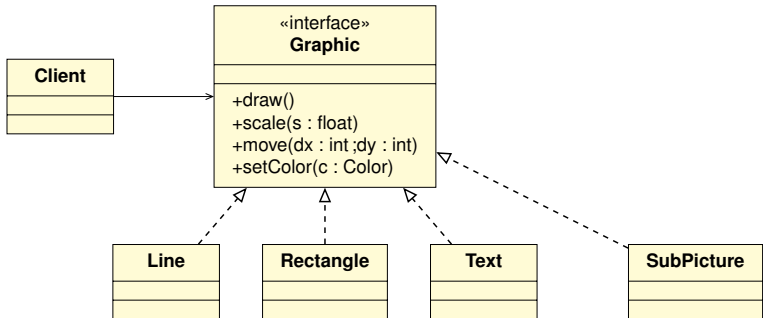
Motivation
Structure
Exemples
Implémentation
Conclusion

Exemple graphique



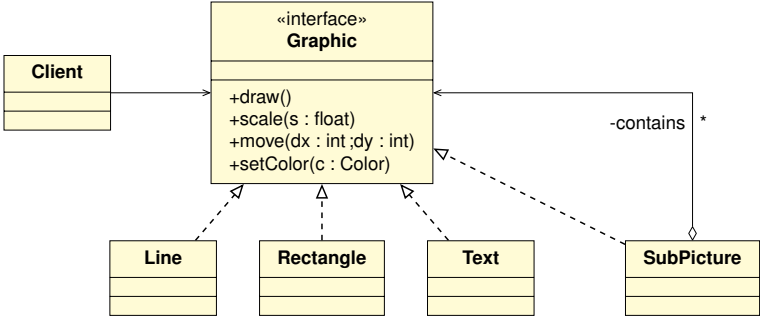
- L'uniformisation découle de l'implémentation de cette interface par tous les composants (y compris le composite).
- Pour être un composite, `SubPicture` doit pouvoir agréger tous les types de composants ...

Exemple graphique



- L'uniformisation découle de l'implémentation de cette interface par tous les composants (y compris le composite).
- Pour être un composite, `SubPicture` doit pouvoir agréger tous les types de composants ...

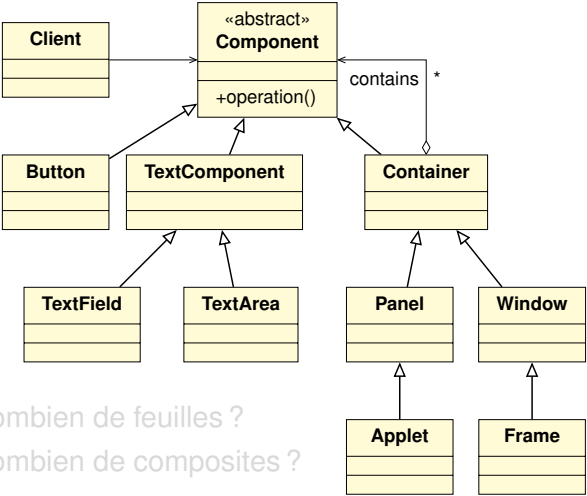
Exemple graphique



- **SubPicture** agrège des *Component* (ici **Graphic**) ce qui lui permet d'avoir également une vision uniforme de son contenu (et récursive).

Autre exemple

Pris dans l'API Java



- Combien de feuilles ?
- Combien de composites ?

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

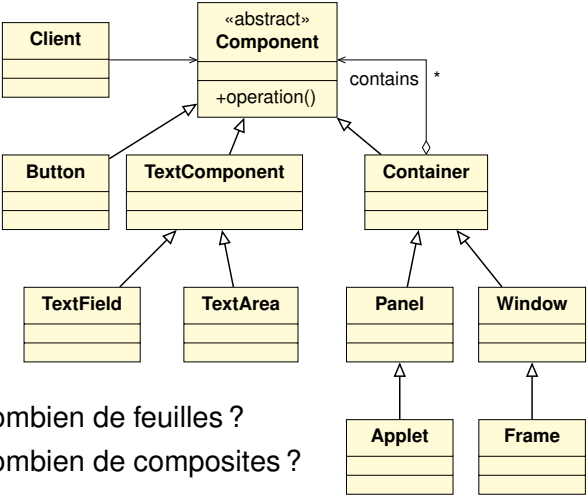
Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Autre exemple

Pris dans l'API Java



- Combien de feuilles ?
- Combien de composites ?

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Considérations d'implémentation

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

Motivation
Structure
Exemples
Considérations
Conclusion

Composite

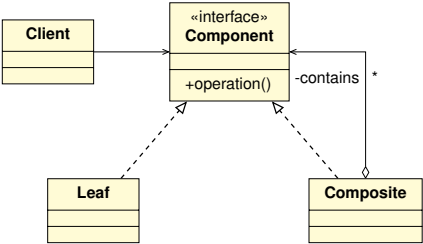
Motivation
Structure
Exemples
Implémentation
Conclusion

L'implémentation du patron *Composite* nécessite de s'interroger sur les aspects suivants :

- l'implémentation du lien d'agréations des composites,
- la construction et la modification de l'arbre/graphe,
- l'implémentation des fonctionnalités uniformes dans le(s) composite(s).

Considérations d'implémentation

Lien d'agréations des composites



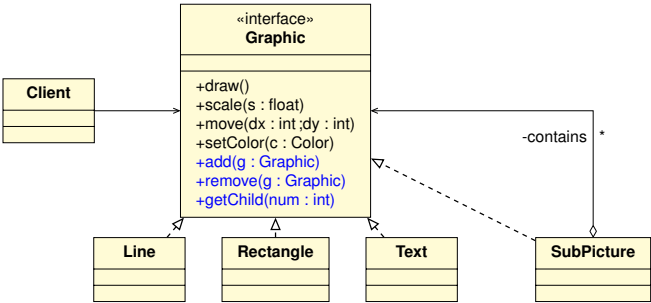
Comment un composite maintient-il l'accès à ses fils ?

- l'ordre des fils a-t-il une importance ?
- efficacité : insertion, parcours, recherche ?
- les fils doivent-ils maintenir un lien avec le parent ?

Considérations d'implémentation

Fonctions de construction/modification

Où placer les fonctions permettant de greffer/retirer des fils ?

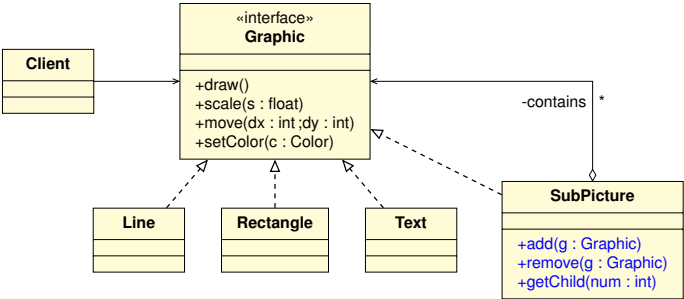


Vraiment ???

Considérations d'implémentation

Fonctions de construction/modification

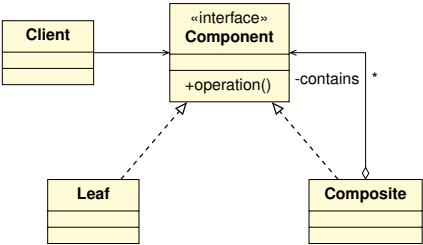
Ces fonctions ne concernent que le code constructeur (un autre client) :



Mieux !!!

Considérations d'implémentation

Fonctions uniformes dans le composite



- Les fonctions uniformes sont déclarées dans l'interface *Component*,
- Chaque type de feuille apporte sa propre implémentation (polymorphisme),

Adaptateur

Motivation
Structure
Exemples Java
Comparaison
Conclusion

Façade

Motivation
Structure
Exemples
Conclusion

Pont

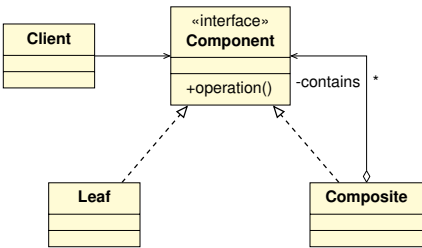
Motivation
Structure
Exemples
Considérations
Conclusion

Composite

Motivation
Structure
Exemples
Implémentation
Conclusion

Considérations d'implémentation

Fonctions uniformes dans le composite



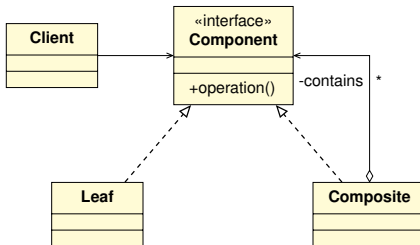
Qu'en est-il des *composites* ?

- Ceux-ci ne doivent pas connaître la nature de leurs fils,
- ils doivent donc leur déléguer une partie du travail.

Le patron composite conduit souvent à une définition « récursive » des méthodes.

Considérations d'implémentation

Fonctions uniformes dans le composite

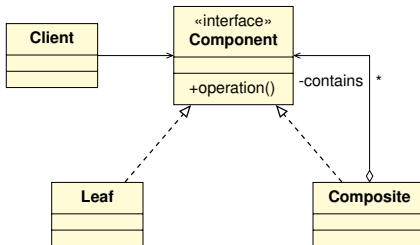


Le choix de la récursivité peut être dangereux :

- On doit dans ce cas s'assurer que le programme constructeur ne construit pas de graphe avec cycles

Considérations d'implémentation

Fonctions uniformes dans le composite



Le choix de la récursivité peut être dangereux :

- On doit dans ce cas s'assurer que le programme constructeur ne construit pas de graphe avec cycles

Considérations d'implémentation

Fonctions uniformes dans le composite

Sans cette garantie, on doit procéder à un parcours couvrant :

Procédure marquage(S: Ensemble des sommets)
Début

```
    pour chaque sommet s de S faire
        marque[s] := A_VOIR
    fait
    pour s de S faire
        si marque[s]=A_VOIR alors
            profondeur (s)
        fsi
    fait
fin
```

Considérations d'implémentation

Fonctions uniformes dans le composite

Sans cette garantie, on doit procéder à un parcours
couvrant :

Procédure profondeur(s : Sommet)

Début

```
marque[s] := EN_COURS
action préfixe sur s
pour t incident_extérieur_à(s) faire
    action sur l'arc (s,t)
    si marque[t]=A_VOIR alors
        profondeur(t)
    fsi
fait
action suffixe sur s
marque[s] := VU
```

fin

Le patron *Visiteur* peut venir à notre secours ici !

Principes respectés

Quizz

- **O.C.P.** : le modèle permet d'ajouter des nouveaux types de feuille et de composite.
- **L.S.P.** : bien sur par l'emploi correct du polymorphisme.
- **D.I.P.** : *Component* est conçue a priori par rapport à *Client*.
- **I.S.P.** : distinction d'interfaces *uniforme*, *composite*, le reste

Principes respectés

Quizz

- **O.C.P.** : le modèle permet d'ajouter des nouveaux types de feuille et de composite.
- **L.S.P.** : bien sur par l'emploi correct du polymorphisme.
- **D.I.P.** : *Component* est conçue a priori par rapport à *Client*.
- **I.S.P.** : distinction d'interfaces *uniforme*, *composite*, le reste

F. Nicart

Adaptateur

- Motivation
- Structure
- Exemples Java
- Comparaison
- Conclusion

Façade

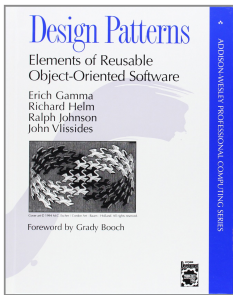
- Motivation
- Structure
- Exemples
- Conclusion

Pont

- Motivation
- Structure
- Exemples
- Considérations
- Conclusion

Composite

- Motivation
- Structure
- Exemples
- Implémentation
- Conclusion



Quelques références

Design Patterns : Elements of Reusable Object-Oriented Software.,
Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison Wesley (1994).
ISBN-13 : 978-0201633610.

Les Design patterns en Java : Les 23 modèles de conception fondamentaux,
Steven John Metsker, William C. Wake , Pearson (2009).
ISBN-13 : 978-2744023965.