

Patron de Conception : Façade



Présenté par
Mama Ndiaye
Daba Mbaye

PLAN

I- Définition

II- Principe

III- Structure

IV- Exemples d'utilisation du pattern *Façade*

V- Implémentation d'un exemple

Conclusion



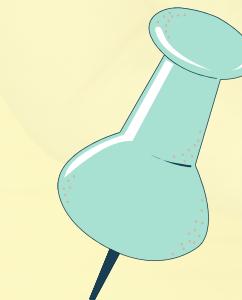
Définition

Façade est un patron de conception structurel qui procure une interface offrant un accès simplifié à une librairie, un framework ou à n'importe quel ensemble complexe de classes.

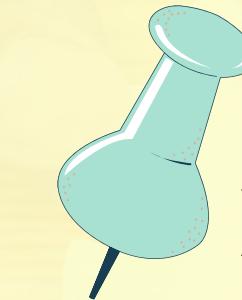
Autrement dit, il s'agit là du visage d'un bâtiment métaphoriquement.

Les passants devant la route ne peuvent voir que cette face vitrée du bâtiment. Ils ne savent rien à ce sujet, le câblage, les tuyaux et autres complexités. Il cache toutes les complexités du bâtiment et affiche un visage amical.

Principe

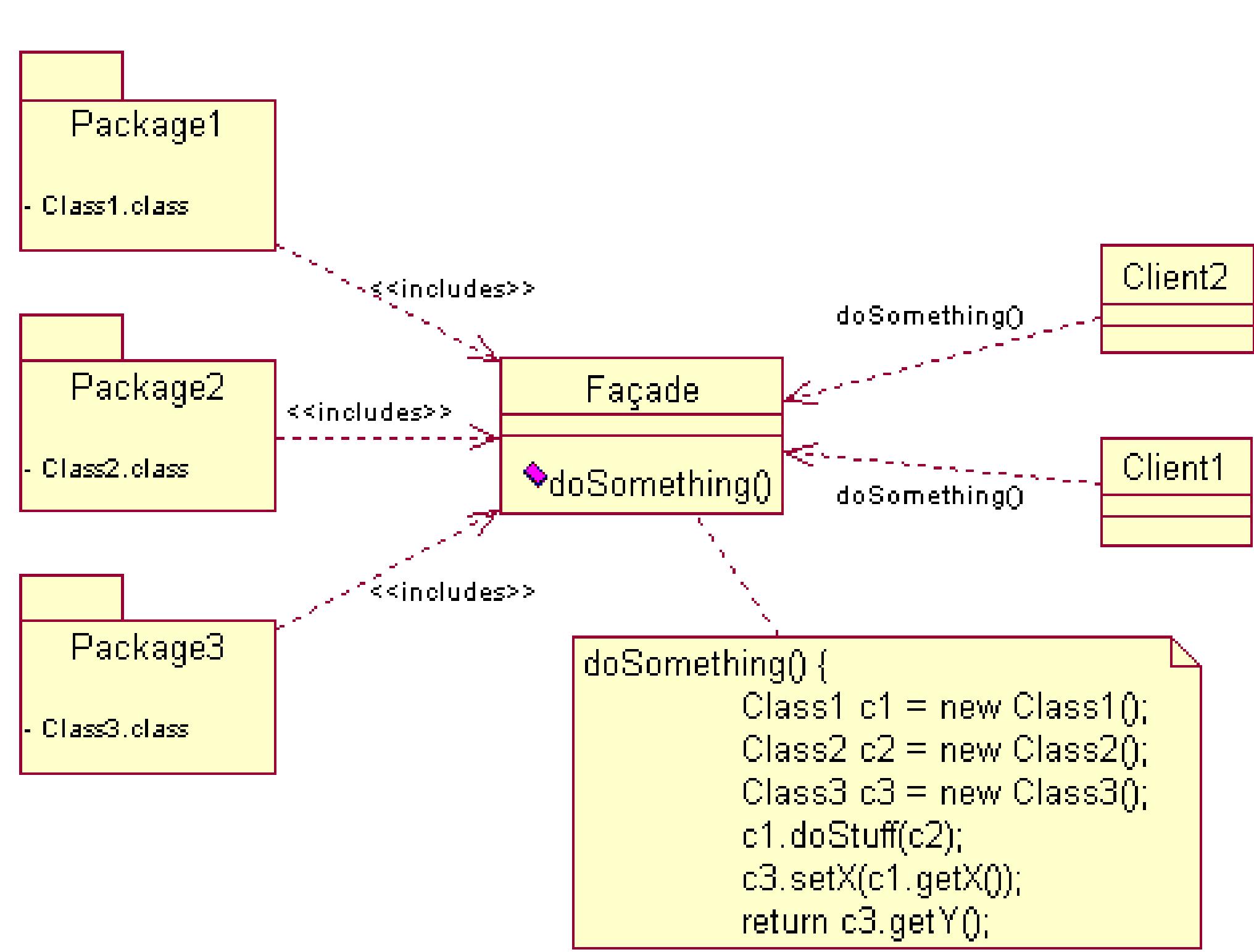


Ce modèle implique une classe unique qui fournit les méthodes simplifiées requises par le client et délègue les appels aux méthodes des classes système existantes.



Habituellement, la façade est réalisée en réduisant les fonctionnalités de ce dernier, mais en fournissant toutes les fonctions nécessaires à la plupart des utilisateurs.

Structure



La classe Façade :

La façade fait abstraction des packages 1, 2 et 3 du reste de l'application.

Clients ;

Les objets utilisant le patron de conception Façade pour accéder aux ressources abstraites.

Quelques Exemples d'utilisation du pattern Façade

CAS

Commande dans un magasin

Démarrage d'un ordinateur

Interface JDBC

Lorsque vous téléphonez à un magasin pour commander, un opérateur joue le rôle de la façade pour tous ses services. L'opérateur vous sert d'interface vocale avec le système de commandes, les moyens de paiement et les différents services de livraison.

Lorsqu'un ordinateur démarre, cela implique le travail du processeur, de la mémoire, du disque dur, etc. Pour le rendre facile à utiliser pour les utilisateurs, nous pouvons ajouter une façade qui enveloppe la complexité de la tâche et fournit une interface simple à la place.

En Java, l'interface JDBC peut être appelée une façade car, en tant qu'utilisateurs ou clients, nous créons une connexion à l'aide de l'interface « `java.sql.Connection` », dont la mise en œuvre ne nous préoccupe pas. La mise en œuvre est laissée au vendeur du pilote.



Implementation d'un exemple

CAS : Générer un rapport PDF, EXCEL et HTML

1

D'abord, nous avons créé une interface pour chaque type de rapport puis l'avons implémenté

```
1 package Facade;  
2  
3 import java.sql.Connection;  
4  
5 public interface PdfRapport {  
6  
7     public abstract void genererPdf(Connection connection, String uneTable);  
8  
Implementation > ① PdfRapportImpl.java > 📁 PdfRapportImpl  
1 package Implementation;  
2  
3 import java.sql.Connection;  
4  
5 import Facade.PdfRapport;  
6  
7 public class PdfRapportImpl implements PdfRapport {  
8  
9     public void genererPdf (Connection connection, String tableName) {  
10        System.out.println("Generer rapport PDF...");  
11    }  
12}
```

Rapport PDF

```
Facade > ① ExcelRapport.java > ...
1 package Facade;
2 import java.sql.Connection;
3
4 public interface ExcelRapport {
5
6     public abstract void genererExcel(Connection connection, String uneTable);
7 }
```

```
Implementation > ② ExcelRapportImpl.java > 📈 ExcelRapportImpl
1 package Implementation;
2
3 import java.sql.Connection;
4
5 import Facade.ExcelRapport;
6
7 public class ExcelRapportImpl implements ExcelRapport {
8
9     public void genererExcel(Connection connection, String uneTable) {
10        System.out.println("Generer rapport EXCEL...");
11    }
12 }
```

```
1 package Facade;  
2  
3 import java.sql.Connection;  
4  
5 public interface HtmlRapport {  
6  
7     public abstract void genererHtml(Connection connection, String uneTable);  
implementation > ↗ HtmlRapportImpl.java > ...  
1 package Implementation;  
2  
3 import java.sql.Connection;  
4  
5 import Facade.HtmlRapport;  
6  
7 public class HtmlRapportImpl implements HtmlRapport {  
8  
9     public void genererHtml(Connection connection, String uneTable) {  
10         System.out.println("Generer rapport HTML...");  
11     }  
12 }  
13 }
```

2.A

Voici le code sans l'utilisation du pattern Façade !

On crée un objet de chaque type de rapport afin de pouvoir utiliser la méthode *generer* de ce dernier en lui passant les paramètres.

```
11
12 public class ClientTest {
13
14     public static void main(String[] args) {
15
16         Connection connection = null;
17         String uneTable = "Ma_Table";
18
19         System.out.println("----- Sans utiliser le pattern Facade -----");
20
21         //Sans utiliser le pattern Facade
22         PdfRapport pdfRapport = new PdfRapportImpl();
23         pdfRapport.genererPdf(connection, uneTable);
24
25         HtmlRapport htmlRapport = new HtmlRapportImpl();
26         htmlRapport.genererHtml(connection, uneTable);
27
28         ExcelRapport excelRapport = new ExcelRapportImpl();
29         excelRapport.genererExcel(connection, uneTable);
30
31         System.out.println("-----");
```

D:\licence\LGSSI\2\PATRON DE CONCEPTION\TP\FacadeCode>
javac Client/ClientTest.java

D:\licence\LGSSI\2\PATRON DE CONCEPTION\TP\FacadeCode>java Client/ClientTest
----- Sans utiliser le pattern Facade -----
Generer rapport PDF...
Generer rapport HTML...
Generer rapport EXCEL...

2.B1

Pour adopter le pattern Façade, nous devons créer une interface (une façade) nommée ici *RapportFacade* pour masquer l'implémentation des différents types de rapport. Cette dernière étant inutile à l'utilisateur !

Il revient donc à cette interface de faire toutes les instructions nécessaires pour générer un rapport. A savoir créer un objet de type de chaque rapport puis appeler la méthode *generer* à partir de celui-ci.

```
public class RapportFacade {  
  
    private PdfRapport pdfRapport;  
    private HtmlRapport htmlRapport;  
    private ExcelRapport excelRapport;  
  
    public RapportFacade() {  
        pdfRapport = new PdfReportImpl();  
        htmlRapport = new HtmlReportImpl();  
        excelRapport = new ExcelReportImpl();  
    }  
  
    public void genererPdf(Connection connection, String uneTable) {  
        pdfRapport.genererPdf(connection, uneTable);  
    }  
  
    public void genererHtml(Connection connection, String uneTable) {  
        htmlRapport.genererHtml(connection, uneTable);  
    }  
  
    public void genererExcel(Connection connection, String uneTable) {  
        excelRapport.genererExcel(connection, uneTable);  
    }  
}
```

2.B2

Voici le code en utilisant le pattern Façade !

On fait appel à la façade créée à cet effet *RapportFacade*, en créant un objet de ce type.

Puis on passe par cet objet de type *RapportFacade* pour appeler les méthodes *generer* dont on a besoin

----- En utilisant le pattern Facade -----

Generer rapport PDF....

Generer rapport HTML....

Generer rapport EXCEL....

Conclusion

Une façade promet ici une grande simplification ; c'est pourquoi l'utilisation de la stratégie de patron de façade devrait jouer un rôle primordial dans la planification d'un projet. Grâce au faible degré de dépendance des différents composants les uns des autres, les changements (modifications, maintenance) sont possibles à tout moment. Cependant le degré élevé de dépendance de l'interface de la façade lui-même ne constitue t-elle pas un risque ?