

MODELE DE CONCEPTION DE FACADE



Fatou Kiné DIA
Soda GUEYE
Fatoumata NDOYE



Introduction

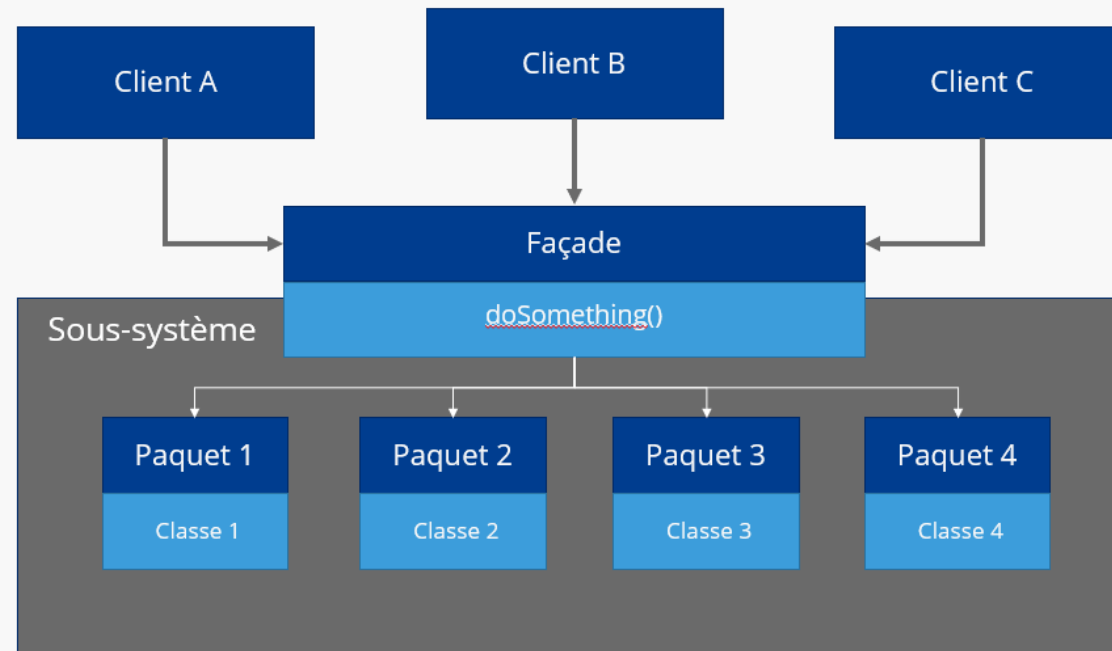
- En général, les patrons visent à simplifier la création de logiciels flexibles et réutilisables. Le patron de façade est un des 23 patrons de conceptions pour le **développement de logiciels** publiés en 1994 par Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides (connus sous le nom de GOF pour « Gang of Four ») dans « Design Patterns : Elements of Reusable Object-Oriented Software ».
- Ainsi, comme son nom l'indique, cela signifie le visage du bâtiment. 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.
- Le façade pattern propose une solution type pour fusionner facilement différentes interfaces dans des systèmes complexes.

■ Façade pattern : représentation graphique (UML)

- La façade ou classe façade est l'unité de structure la plus importante du patron de façade. Son **implémentation et son élaboration** constituent donc la tâche fondamentale des développeurs qui souhaitent simplifier leurs logiciels complexes en utilisant ce patron de conception pratique.
- Le diagramme de classes qui suit, en langage de modélisation UML, explicite l'interaction des clients, de la façade et des classes de sous-systèmes selon le facade pattern.

- Une fois implémentée, les objets clients concernés concentrent toute leur communication sur la **classe façade**, qui devient ainsi la seule **instance** de ce nouveau système dont les **clients dépendent directement**.

Diagramme UML : modèle de facade pattern



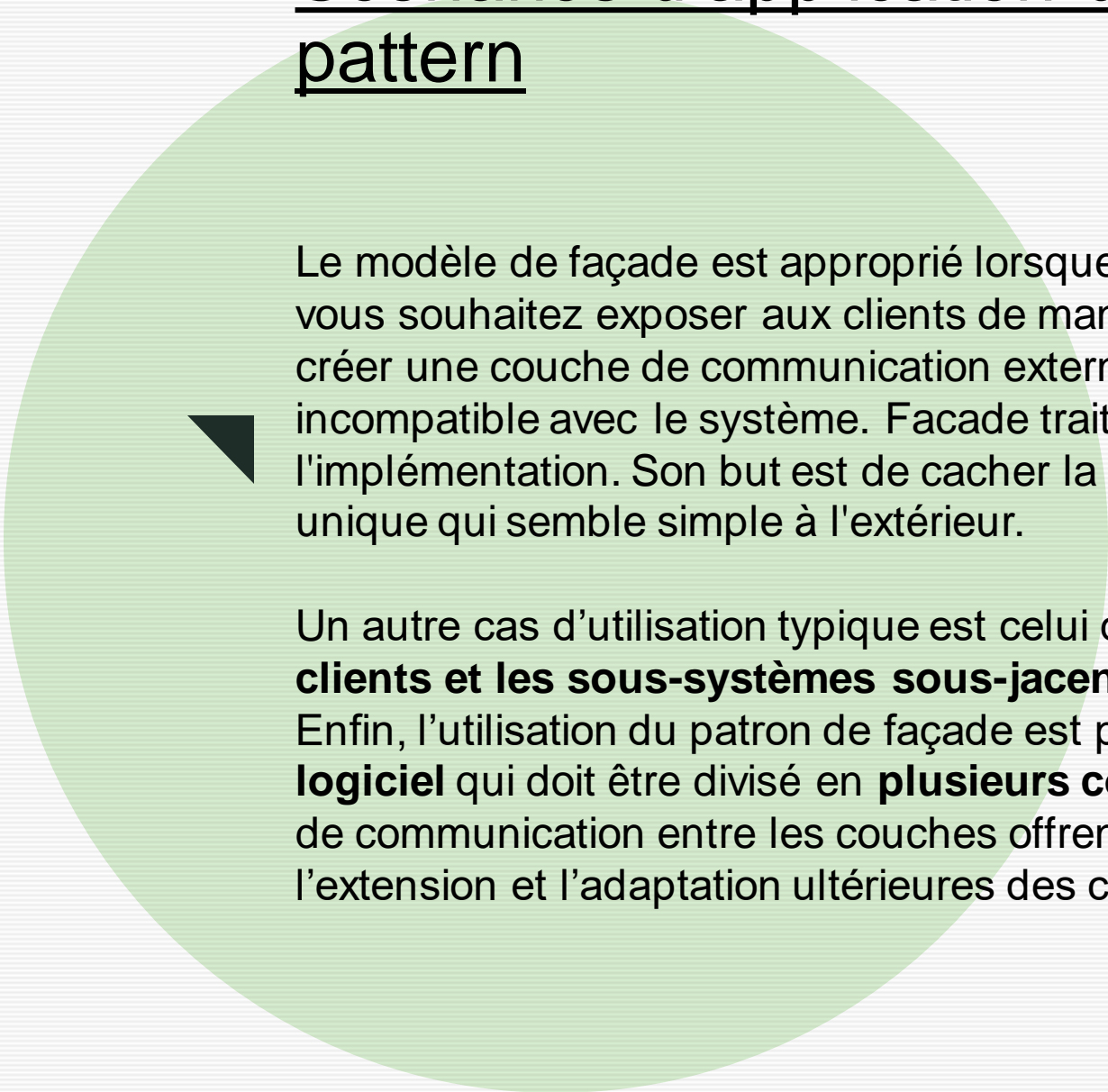
IONOS

Le nombre de quatre paquets de classes dans le diagramme UML a ici valeur d'exemple. En théorie, vous pouvez utiliser la classe Façade pour contrôler un nombre illimité de classes dans le sous-système.

▀ Avantages et inconvénients du façade pattern

Avantages	Inconvénients
Réduit la complexité des sous-systèmes	Implémentation complexe (surtout avec un code déjà existant)
Favorise le principe du couplage faible	La solution est couplée à une étape indirecte supplémentaire
Les logiciels sont plus flexibles et plus faciles à étendre	Degré élevé de dépendance de l'interface de la façade

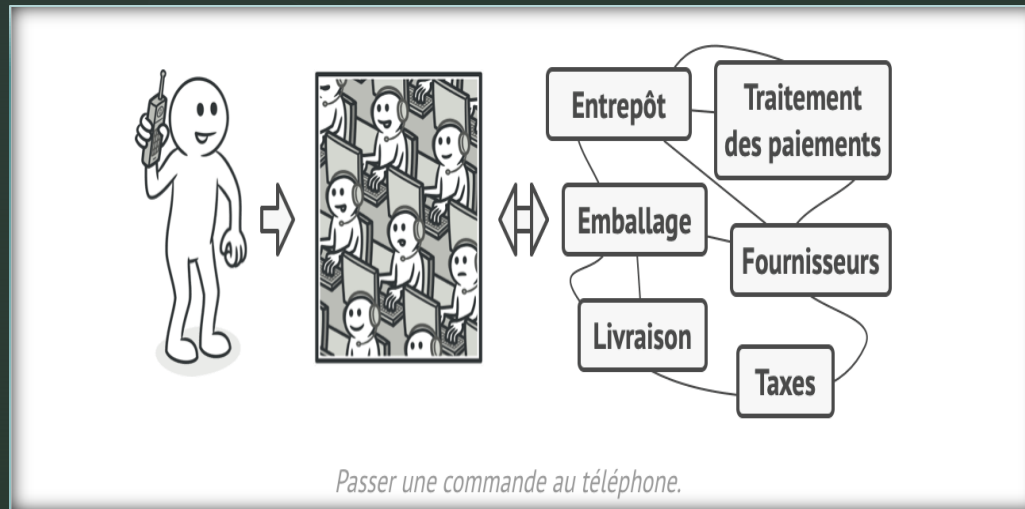
Scénarios d'application du facade pattern



Le modèle de façade est approprié lorsque vous avez un **système complexe** que vous souhaitez exposer aux clients de manière simplifiée, ou que vous souhaitez créer une couche de communication externe sur un système existant qui est incompatible avec le système. Facade traite des interfaces, pas de l'implémentation. Son but est de cacher la complexité interne derrière une interface unique qui semble simple à l'extérieur.

Un autre cas d'utilisation typique est celui des logiciels où la **dépendance entre les clients et les sous-systèmes sous-jacents doit être réduite au minimum**. Enfin, l'utilisation du patron de façade est payante lorsque vous planifiez un **projet de logiciel** qui doit être divisé en **plusieurs couches**. Les façades en tant qu'interfaces de communication entre les couches offrent également plus de flexibilité pour l'extension et l'adaptation ultérieures des composants.

Analogie à une structure du façade pattern



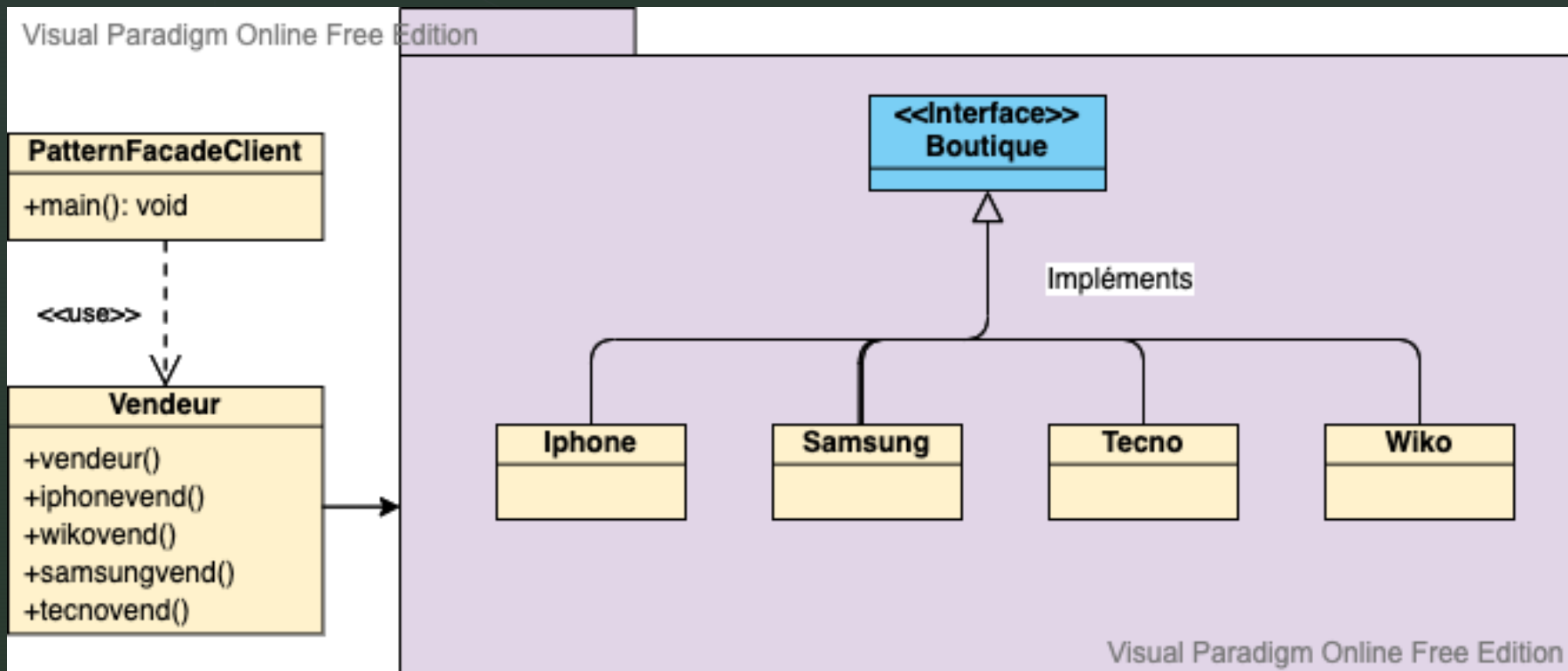
- Lorsque vous téléphonez à un magasin pour commander, un opérateur joue le rôle de la façade pour tous ses services. L'opérateur vous sert d'interface vocale avec le système de commandes, les moyens de paiement et les différents services de livraison.

Exemple d'utilisation du pattern

- Essayons maintenant de mieux comprendre le motif de la façade à l'aide d'un exemple simple.
- En tant que patron de conception, le façade pattern n'est pas lié à un langage de programmation spécifique. Cette solution est notamment utilisée en C++, C#, JavaScript, Java, PHP et Python. L'**exemple de patron de façade** suivant est basé sur un **code Java**.
- (explication de l'exemple.....)

Exemple: Boutique de vente de téléphones portables

- Diagramme UML pour le modèle de façade de notre exemple :



- Créons une interface Boutique

```
J Boutique.java
1
2  public interface Boutique {
3
4
5      public void modele();
6
7      public void prix();
8
9  }
10
```

- Créons les classes des téléphones qui implémentent l'interface Boutique

```
1 Iphone.java
2 public class Iphone implements Boutique {
3
4     @Override
5     public void modele() {
6         System.out.println(" Iphone 12 PRO MAX");
7     }
8
9
10    @Override
11    public void prix() {
12        System.out.println(" 740.000 FCFA ");
13    }
14
15
16 }
17
```

1 Samsung.java

```
1
2 public class Samsung implements Boutique {
3
4
5     public void modele() {
6         System.out.println(" Samsung Galaxy A71 ");
7
8     }
9     public void prix() {
10        System.out.println(" 214.000 FCFA");
11
12    }
13
14 }
15
```

J Tecno.java

```
1
2  public class Tecno implements Boutique {
3
4      public void modele() {
5          System.out.println(" Tecno Camon 17 ");
6      }
7
8
9      public void prix() {
10         System.out.println(" 93.000 FCFA");
11     }
12 }
13
14 }
15
```

Wiko.java

```
1
2  ∨ public class Wiko implements Boutique {
3
4
5  ∨     public void modele() {
6         System.out.println(" Wiko Power U20  ");
7
8     }
9  ∨     public void prix() {
10         System.out.println(" 110.000 FCFA");
11
12     }
13
14 }
15
```

- Créons une classe Vendeur qui utilisera l'interface Boutique

Vendeur.java

```
1
2  public class Vendeur {
3      private Boutique iphone;
4      private Boutique samsung;
5      private Boutique wiko;
6      private Boutique tecno;
7
8      public Vendeur(){
9          iphone= new Iphone();
10         samsung=new Samsung();
11         wiko=new Wiko();
12         tecno=new Tecno();
13     }
14
15     public void iphoneVend(){
16         iphone.modele();
17         iphone.prix();
18     }
19
```



```
public void samsungVend(){  
    samsung.modele();  
    samsung.prix();  
}
```

```
public void wikoVend(){  
    wiko.modele();  
    wiko.prix();  
}
```

```
public void tecnoVend(){  
    tecno.modele();  
    tecno.prix();  
}
```

```
}
```

■ Résultat

```
a@MacBook-Air-de-Kine 5-Facade Pattern % java PatternFacadeClient.java
***** Boutique de Vente de Telephones portables *****
1. IPHONE
2. SAMSUNG
3. WIKO
4. Tecno.
5. Exit.
Entrer votre choix: 3
Wiko Power U20
110.000 FCFA
***** Boutique de Vente de Telephones portables *****
1. IPHONE
2. SAMSUNG
3. WIKO
4. Tecno.
5. Exit.
Entrer votre choix: 4
Tecno Camon 17
93.000 FCFA
***** Boutique de Vente de Telephones portables *****
1. IPHONE
2. SAMSUNG
3. WIKO
4. Tecno.
5. Exit.
Entrer votre choix: 5
Rien Acheté
a@MacBook-Air-de-Kine 5-Facade Pattern %
```

conclusion

- L'utilisation de ces modèles conduit à la création d'une classe supplémentaire (une façade) qui peut être un effort supplémentaire et inutile, en particulier dans les cas où il n'y a pas une grande complexité attendue du système. Le véritable avantage ne peut être obtenu que dans des systèmes grands et complexes avec une grande arborescence de sous-systèmes.