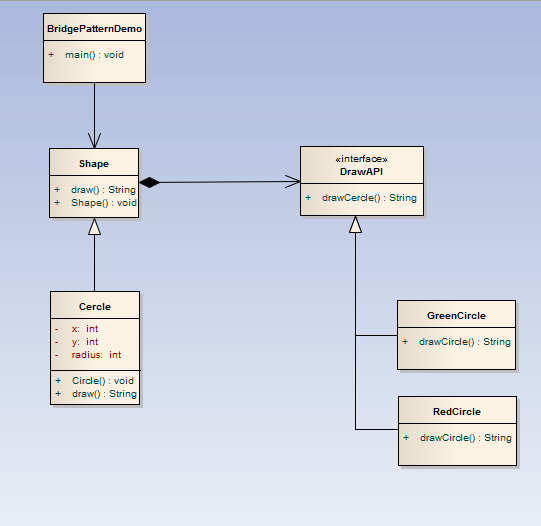
Tp3 : Designs Pattern de Structures

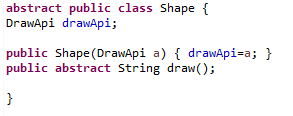
Exercice1 :

Utilisation de design pattern Bridge pour modéliser une API de traçage graphique qui découple l’abstraction de l’API de son implémentation.

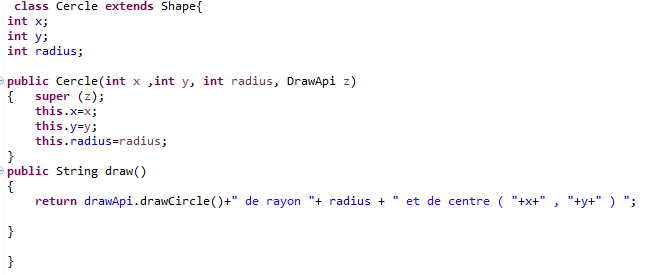


Implémentation :

Classe Shape :



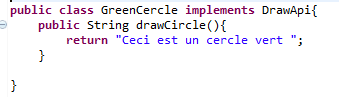
Classe Cercle :



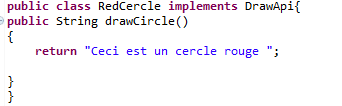
Classe DrawApi :



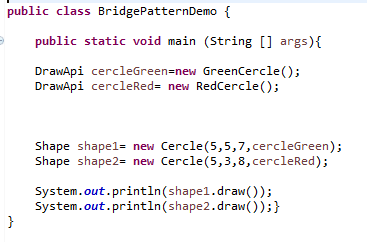
Classe GreenCercle :

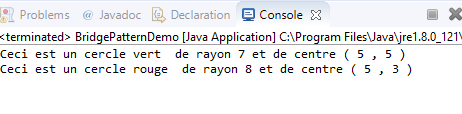


Classe RedCercle :



Classe BridgePatternDemo :

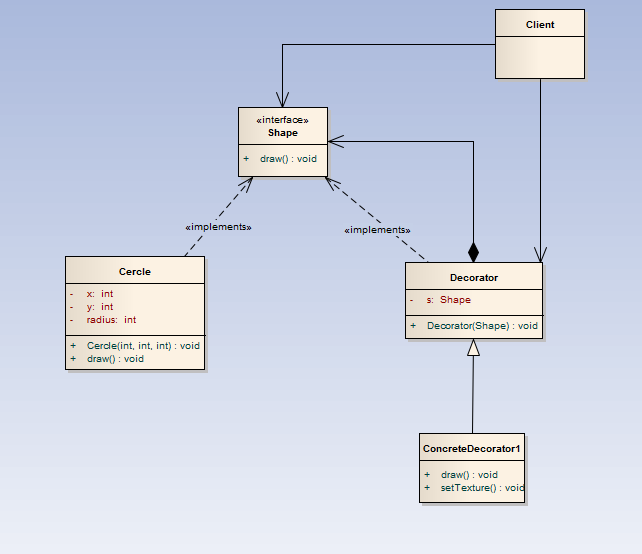




Exercice2 :

On cherche un design pattern qui permet d’attacher dynamiquement des capacités additionnelles à un objet existant(Shape) 🡪 on utilise alors le design pattern décorateur.

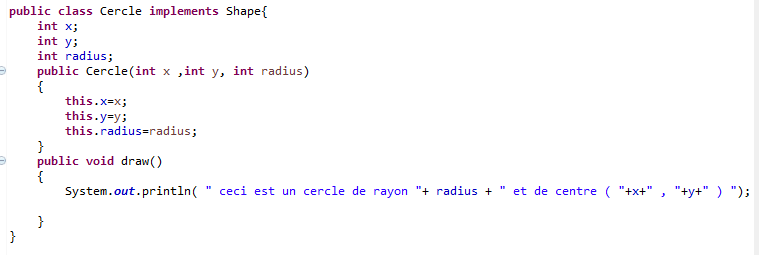
On va ajouter via un decorateur la méthode setTexture() qui va décrire la façon avec la quelle la methode draw() fonctionne



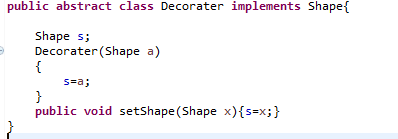
Interface Shape



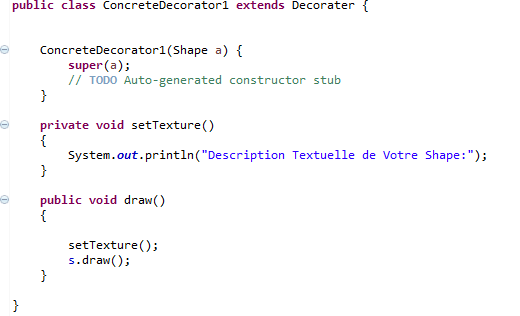
Classe Cercle :



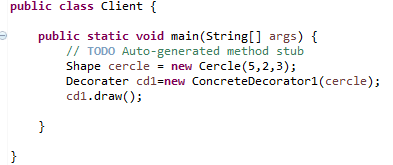
Classe Decorator

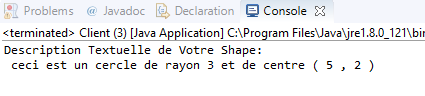


Classe ConcretDecorator 1:



Classe Client





Exercice3 :

1. Cette conception rend le système plus complexe et provoque une grande Independence entre le Client et les Classes de sous-systèmes, Sachant que le Client ne va pas utiliser tous les méthodes de ces classes, mais un ensemble de fonctionnalités. De plus elle n’est pas facilement compréhensible par les développeurs.
2. Il faut proposer une classe qui regroupe les méthodes qui vont réaliser les appels nécessaires aux classes du sous-système pour offrir des fonctionnalités cohérentes. Autrement dit offrir un niveau d'abstraction entre l'ensemble de classes et celles qui souhaitent les utiliser en proposant une interface de plus haut niveau pour utiliser les classes du sous-système🡪Façade

Avantages :

\*Réduire la complexité d’un système en le découpant en plusieurs sous-systèmes.

\*Eviter la dépendance entre les clients et les éléments de sous-systèmes.

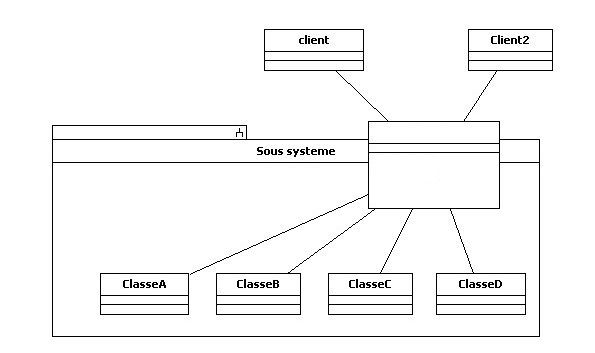
\*faciliter la mise en œuvre des classes.

\* cacher au client l'existence du sous-système

\* Faciliter l'utilisation partielle d'un sous-système complexe ou de plusieurs classes

\* Ajouter des fonctionnalités sans modifier le sous-système

\* Assurer un découplage entre le client et le sous-système



1. Cette solution appartient au design pattern de structure Façade.
2. Le modèle de conception de façade est souvent utilisé lorsqu'un système est très complexe ou difficile à comprendre, car le système possède un grand nombre de classes interdépendantes. Ce modèle cache les complexités du système plus large et fournit une interface plus simple au client(Façade)
3. Les participants à ce design pattern :

**\*le client** : qui va utiliser la façade.

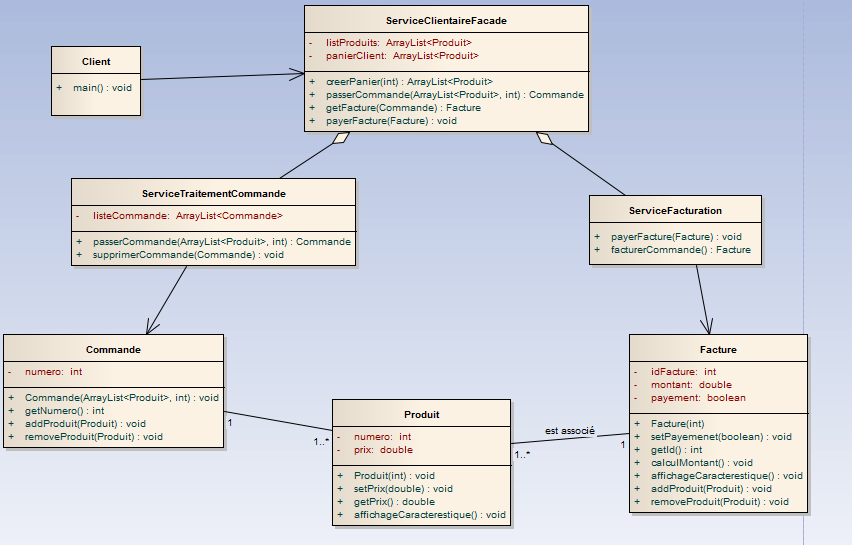
**\*La façade**

**\*les classes de sous Système :** utilisés par la façade.

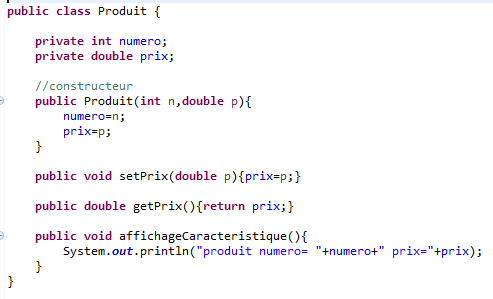
6)

Problème : vente de produits en ligne.

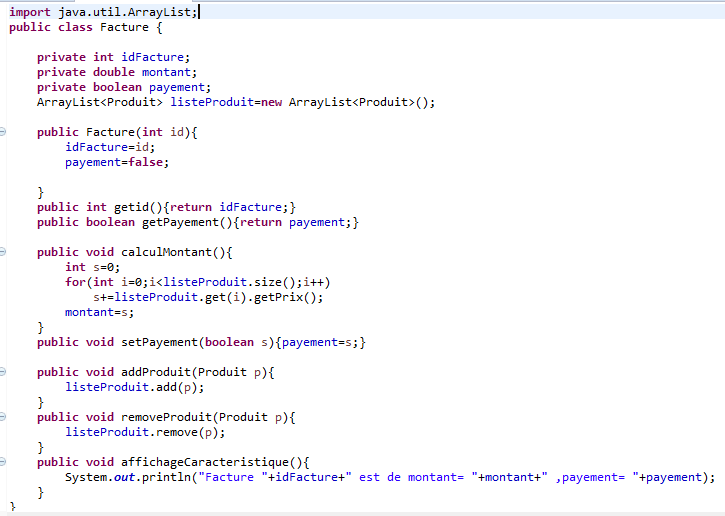
Un client appelle le ServiceClientaireFacade pour passer ou annuler une commande via le SerrviceTraitemenetCommande ou pour avoir la facture et la payer viaServiceFacturation



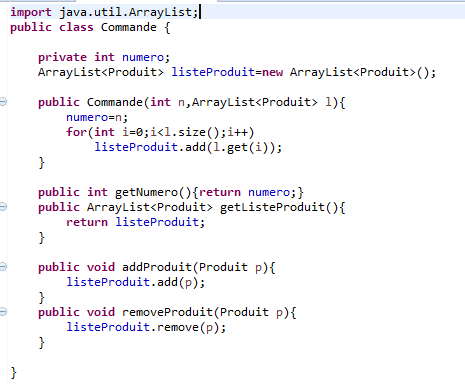
Classe Produit :



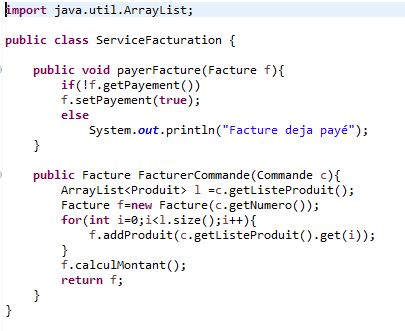
Classe Facture :



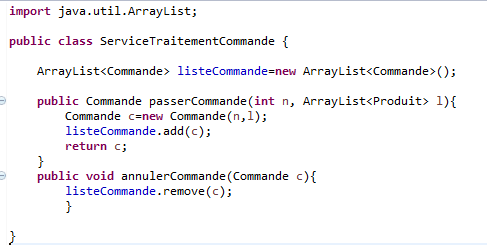
Classe Commande :



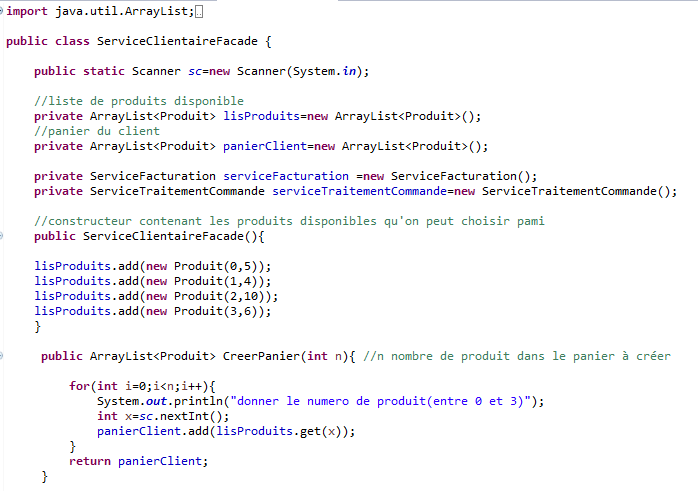
Classe ServiceFacturation :

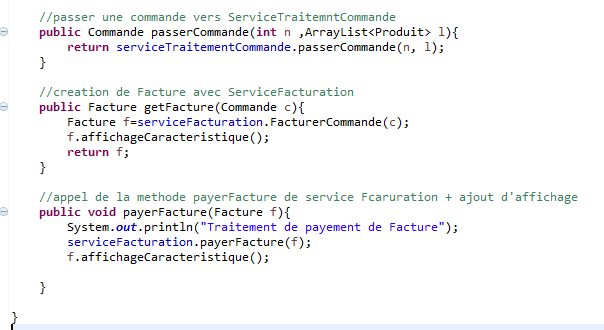


ServiceTraitementCommande :

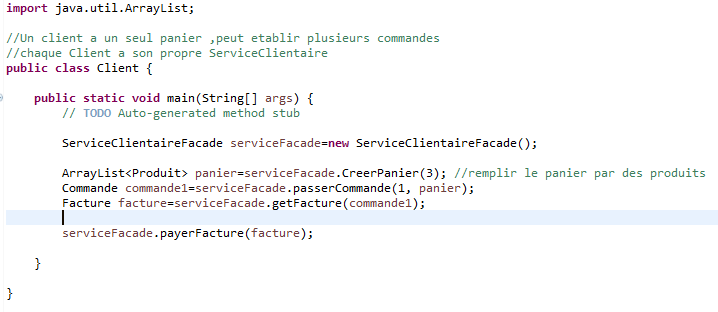


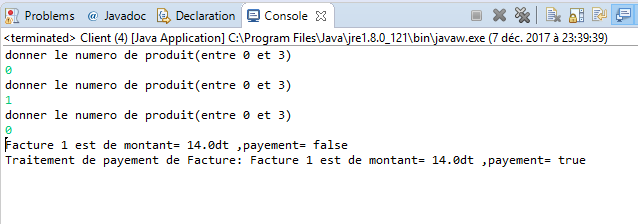
Classe ServiceClientaireFacade :





Classe Client :





Mtibaa Amal

Riahi Mohamed Wassim Gl3/gp1