



# Chapitre 4 : Architecture & Conception de logiciels

**UP GL-BD**



# ► Objectifs Chapitre 4

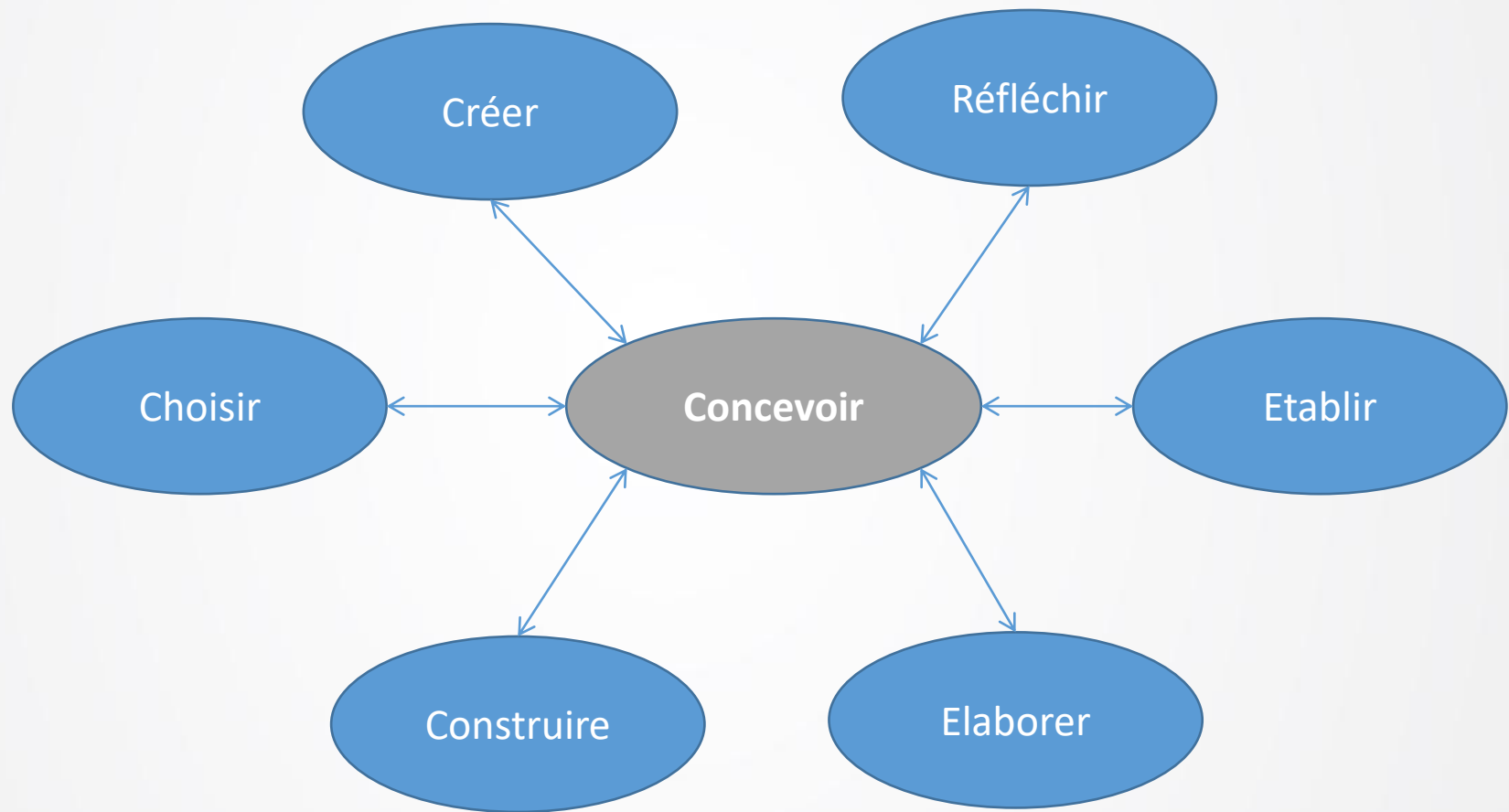
- Distinguer entre architecture logique et architecture physique
- Distinguer entre architecture globale et architecture détaillée
- Comprendre et exploiter les critères de qualité.
- Expliquer et distinguer entre les différents patrons architecturaux
- Démontrer l'intérêt de l'application des patrons de conception



# Conception logicielle - Définition

- Comment organiser le logiciel pour qu'il fasse ce qu'il doit faire?
- Quelles choix techniques faut-il faire pour que le logiciel fasse ce qu'il doit faire?
- Comment s'assurer que le logiciel est organisé et construit de manière à faire ce qu'il doit faire?

# ► Conception logicielle - Définition



# ► Conception logicielle - Définition

- **Conception logicielle** : Processus d'analyse et de résolution des problèmes



Identifier la meilleure façon d'implémenter les exigences fonctionnelles



Respecter l'ensemble des contraintes système



# ► Objectifs de la conception

## La réutilisabilité\*

- Réutiliser le code (code source ou API) produit par d'autres afin de réduire le temps de développement et minimiser la complexité de l'application
- Pourquoi réinventer la roue ? Autant réutiliser du code produit par d'autre, dans le respect du droit d'auteur et des licences d'utilisation de ce code.

\* Source: Le dictionnaire des développeurs

- <https://dico.developpez.com/html/1455-Langages-reutilisabilite.php>



# ► Objectifs de la conception

## **La maintenabilité**

- Est une caractéristique du logiciel.
- Est la facilité avec laquelle un logiciel peut être maintenu.

=> La maintenabilité est un des facteurs qui influencent les coûts de maintenance



# ► Niveaux conceptuels

## Conception architecturale globale

- Architecture de haut niveau.
- Structure et organisation générale du système à concevoir.
- **Décomposer le logiciel:**
  - en **composants (sous-systèmes)** plus simples, définis par leurs **interfaces (interactions)** et leurs **fonctions** (les services qu'ils rendent) ainsi que leur **déploiement** sur les différents **nœuds physiques**.

## Conception architecturale détaillée

- Détailler la conception générale.
- Fournir pour chaque composant une description de la **manière dont les fonctions ou les services sont réalisés** : structures de données, algorithmes, etc.



# ► Types d'architectures



## Architecture logique

- Structure logique de l'application.
- Décomposition en éléments logiques.
- **Exemples** : découpage en composants.

## Architecture physique

- Structure physique de l'application.
- Décomposition en éléments physiques.
- Ensemble de ressources physiques (serveurs, ordinateurs, etc.) nécessaires à l'exécution de l'application.
- **Exemple** : Découpage en nœuds physiques.

➡ **L'architecture logique est répartie sur l'architecture physique.**



# ► Types d'architectures

- **Architecture logique – Quelques concepts :**
  - **Décomposition** / Structuration du logiciel en **composants**.
  - Un **composant est une unité autonome** faisant partie d'un système ou d'un sous-système qui encapsule un comportement et qui offre une ou plusieurs interfaces publiques.
  - Un composant a une vocation bien déterminée et est censé fournir **un service** bien précis : les fonctionnalités qu'il encapsule doivent être cohérentes.
  - Les fonctionnalités d'un composant peuvent être appelées depuis une entité externe. Pour pouvoir être utilisé, le composant fournit **une interface** : l'ensemble de fonctions lui permettant de communiquer avec l'entité cliente.
  - Un composant peut être sujet lui-même à composition.
  - Un composant peut être isolé et remplacé par un autre composant ayant des fonctionnalités équivalentes. La plupart des composants devraient être **réutilisables**.



# Types d'architectures

- **Architecture logique – Approche de résolution :**

- **Approches de haut en bas (Top – Down) :**



1. Concevoir la structure générale.
2. Etudier les éléments du plus bas niveau.
3. Détailler tous les éléments de la structure : format des données, algorithmes, etc.

- **Approche de bas vers le haut (Bottom – Up) :**



1. Identifier les éléments de bas niveau.
2. Prendre des décisions concernant la réutilisation.
3. Décider de l'assemblage des éléments pour créer des structures de plus haut niveau.

- **Combinaison des 2 approches :**

- Une approche de haut en bas est nécessaire afin de garantir une bonne structure au système.
- Une approche de bas en haut est utile afin de s'assurer que des composantes réutilisables soient concevable et réalisables.



# Patrons d'architecture

- Modèles standards de structuration qui couvrent les types classiques d'application.

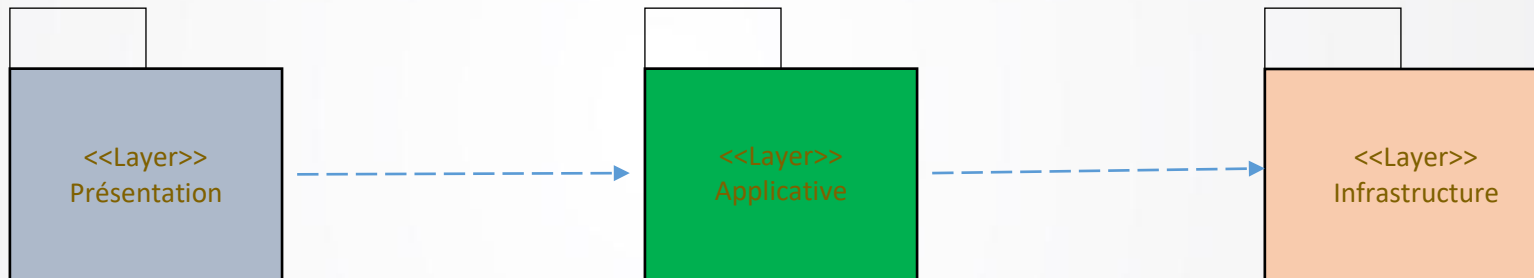
## Exemples :

- **Modèle en couches :**
  - S'applique aux applications munies d'une interface graphique manipulant des données persistantes.
  - Architecture logique en 3 couches.
  - Architecture logique en 5 couches.
- **MVC :**
  - Modèle : contient les données à afficher
  - Vue : fait l'affichage
  - Contrôleur : coordonne les deux

# Patrons d'architecture– Modèle en 3 couches

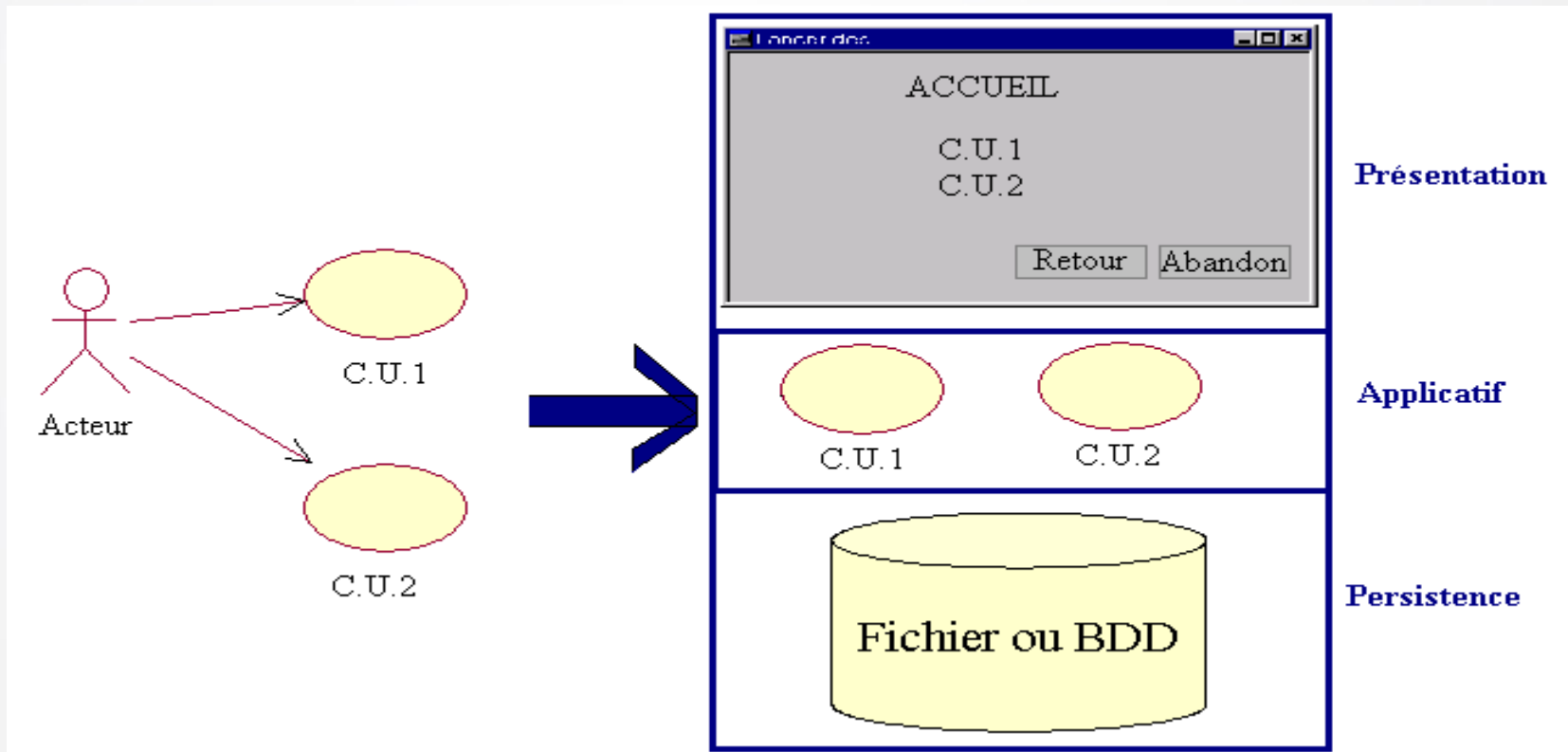
13

**3 couches de bases :**



# ▶ Patrons d'architecture– Modèle en 3 couches

- **Exemple :**



# ▶ Découpage en couches – Modèle en 3 couches

- **Principe :**

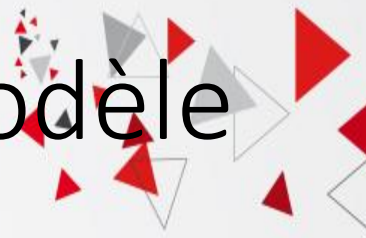
Ce type d'architecture permet de faire évoluer distinctement l'IHM (couche présentation) et/ou le métier (couche applicative) et/ou l'image base de donnée(les entités) (couche infrastructure) sans remettre en question les autres niveaux.

- L'architecture en 3 couches définit une dépendance bidirectionnelle entre « IHM » et « Métier »
- La couche infrastructure n'a aucune dépendance sur la couche « Métier » donc les modifications que nous apporterons la couche « Métier » n'auront pas d'impact sur la couche infrastructure.

➡ En revanche le package « infrastructure » devra répondre aux besoins de « Métier ».



# Découpage en couches – Modèle en 3 couches



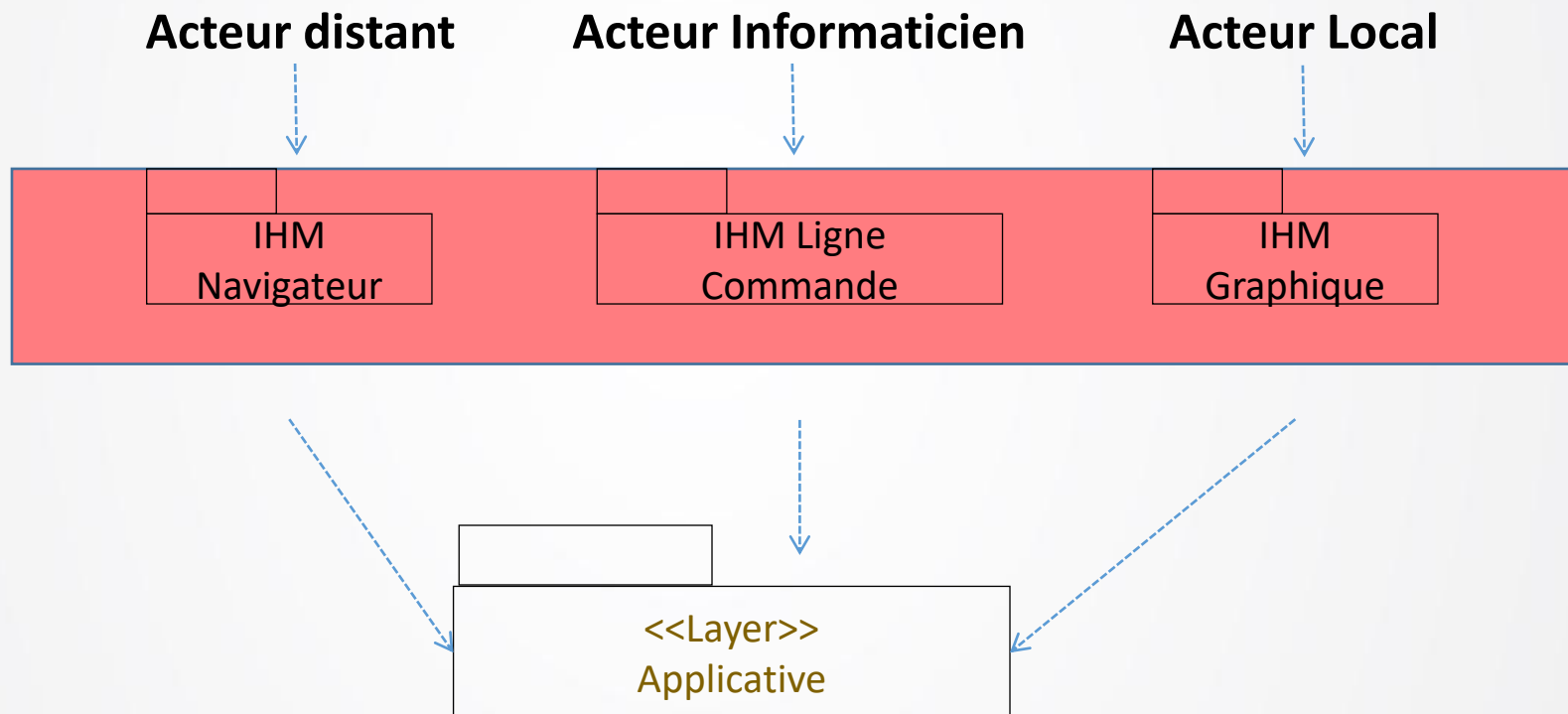
- **Couche présentation :**

- Prend en charge les interactions entre l'utilisateur et le logiciel.
- Permet de visualiser les informations.
- Permet de traduire les commandes de l'utilisateur en actions sur les autres couches.




# Découpage en couches – Modèle en 3 couches

17



**Exemple : Plusieurs présentations d'une application**




# ▶ Découpage en couches – Modèle en 3 couches

18

- **Couche applicative :**

- Correspond à la partie fonctionnelle de l'application.
- Décrit les opérations que l'application opère sur les données en fonction des requêtes des utilisateurs, effectuées au travers de la couche présentation.
- Offre des services applicatifs et métiers à la couche présentation :
  - S'appuie sur les données de la couche inférieure.
  - Renvoie à la couche présentation les résultats qu'elle a calculés.



# Découpage en couches – Modèle en 3 couches

19

## Couche infrastructure :

- Est la partie du code responsable de l'accès aux données dans une application multiniveau doit être encapsulée dans une couche dédiée aux interactions avec la base de données de l'architecture.

Celle-ci permet notamment :

- d'ajouter un niveau d'abstraction entre la base de données et l'utilisation qui en est faite.
- de simplifier la couche métier qui utilise les traitements de cette couche
- de masquer les traitements réalisés pour mapper les objets dans la base de données et vice versa
- de faciliter le remplacement de la base de données utilisée
- La couche métier qui va utiliser la couche infrastructure reste indépendante du code dédié à l'accès à la base de données. Ainsi la couche métier ne contient aucune requête SQL, ni code de connexion ou d'accès à la base de données. La couche métier utilise les classes de la couche persistance qui encapsulent ces traitements. Ainsi la couche métier manipule uniquement des objets pour les accès à la base de données.

# ▶ Découpage en couches – Modèle en 3 couches

20

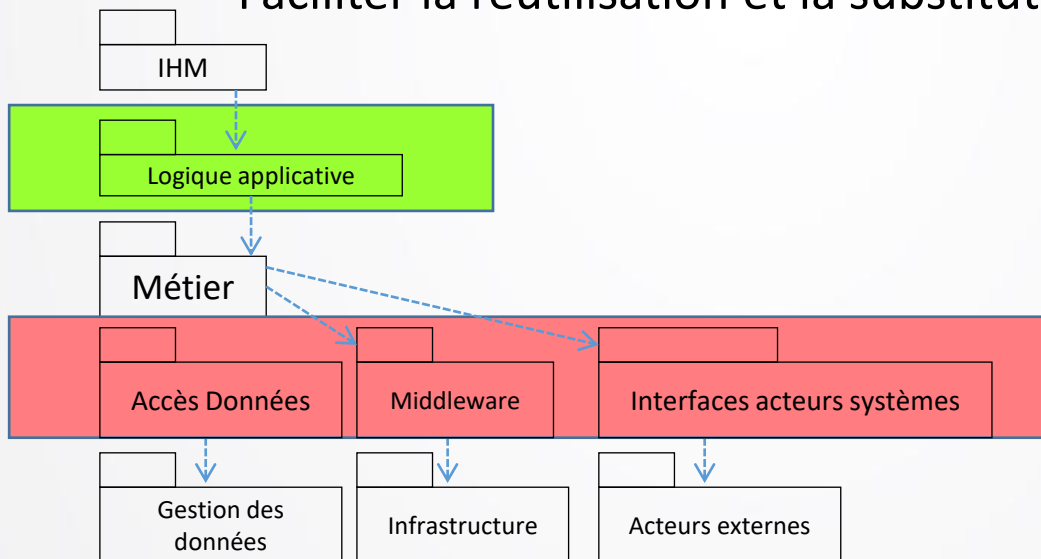


- Découpage conforme à une démarche structurée par les cas d'utilisation.
  - On peut s'occuper d'une couche sans savoir à connaître le détail des autres couches.
  - Minimise les dépendances entre couches
  - Favorise la standardisation (framework).
  - Facilite la réutilisation et la substitution (couplage plus faible et mieux contrôlé).
  - La sécurité peut être renforcée.
- Complexe.
  - Plus d'exigence.
  - Problèmes au niveau des performances.

# Découpage en couches – Modèle en 5 couches

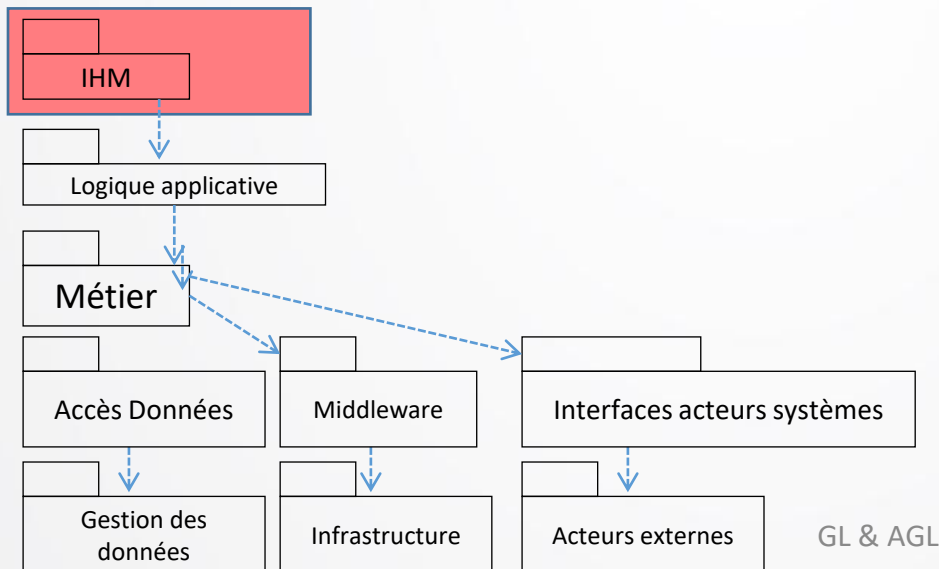
- **Autre découpage en couches : Modèle en 5 couches**

- Modèle en 3 couches + couches intermédiaires : logique applicative + accès données.
- **Objectifs :**
  - Réduire la complexité.
  - Faciliter la réutilisation et la substitution.



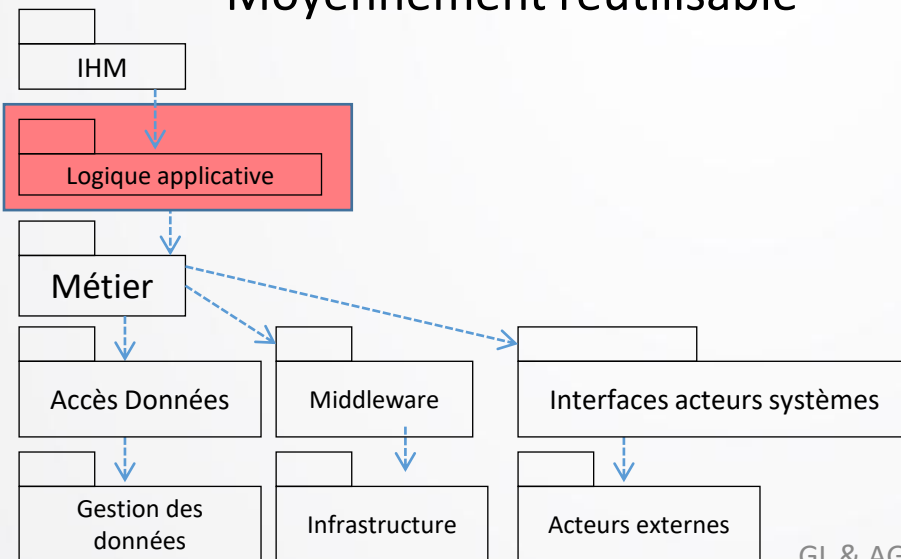
# Découpage en couches – Modèle en 5 couches

- **La couche IHM (Présentation) : Gérer le dialogue Humain-machine :**
  - Capturer, sous forme d'évènements, les requêtes provenant de l'utilisateur (clavier, souris, voix, etc.)
  - Retranscrire ces évènements sous forme d'envoi de message à destination de la couche applicative.
  - Récupérer la réponse et afficher les résultats.



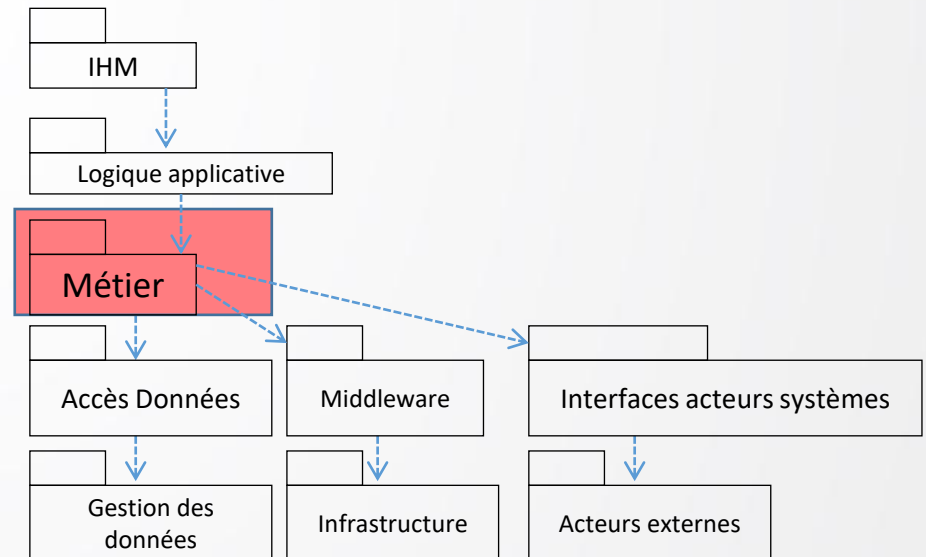
# Découpage en couches – Modèle en 5 couches

- **La couche Applicative : Implémenter les services (cas d'utilisation) demandés par l'utilisateur :**
  - Utilise les services métier (propres au domaine de l'application, couche inférieure).
  - Utilise les services techniques (authentification, autorisation, etc.).
  - Sollicitée la plupart du temps par l'IHM.
  - Moyennement réutilisable



# ▶ Découpage en couches – Modèle en 5 couches

- **La couche Métier : Implémenter les services atomiques métiers propres au domaine et réutilisables par les applications :**
  - Permet de capitaliser le savoir faire de la structure en matière de règles métiers, de règles de gestion et de contrôle de cohérence.
  - Sollicitée par la couche applicative.
  - Potentiellement réutilisable.



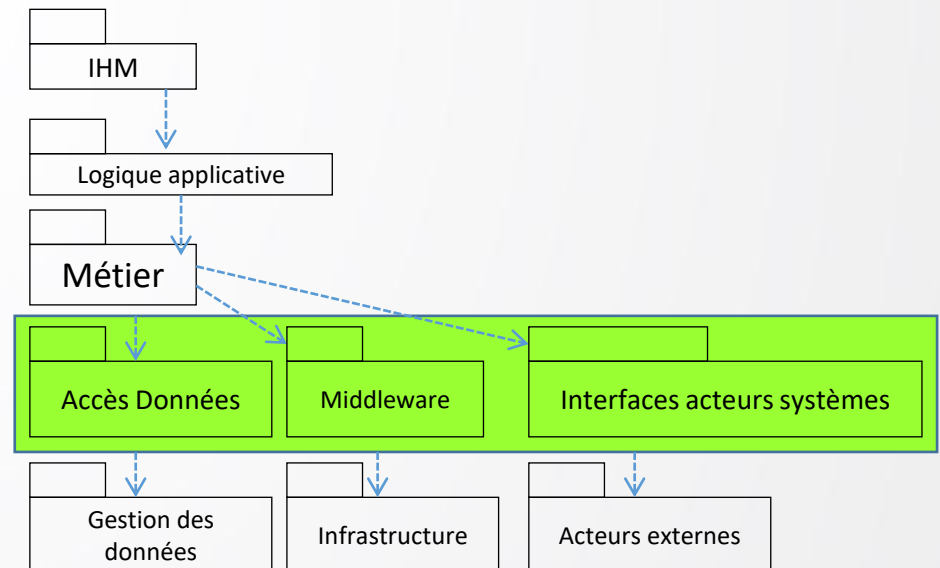


# ▶ Découpage en couches – Modèle en 5 couches

- **La couche d'accès aux données :**

Elle gère le stockage des données, logiquement nommée couche de données.

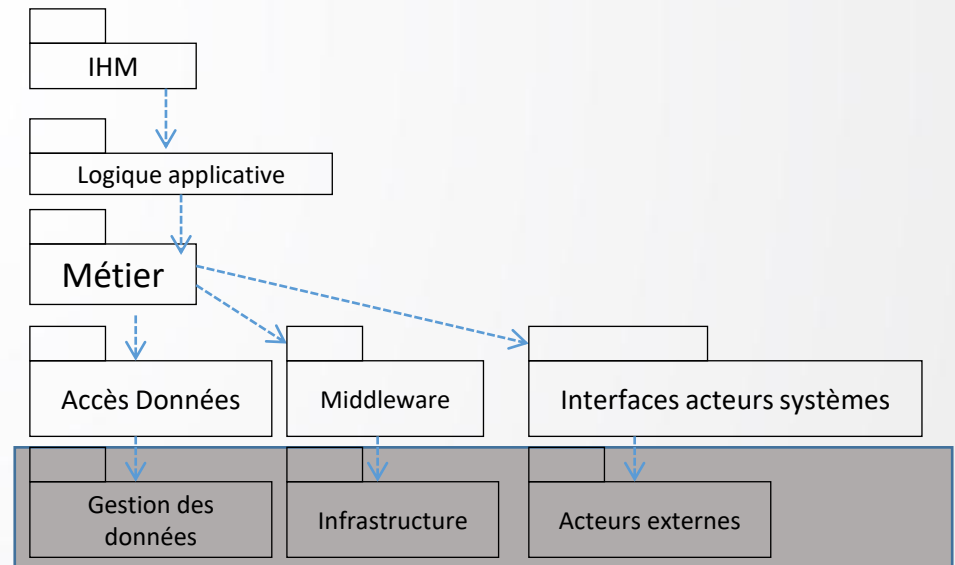
- Il s'agit là des opérations classiques de stockage : la création, la lecture, la modification et la suppression. Ces quatre tâches basiques sont souvent raccourcies à l'anglaise en *CRUD*.



# ► Découpage en couches – Modèle en 5 couches

- **La couche de gestion des données :**

Elle sert de communicateur avec la base de données. Son avantage est le rôle de sas de sécurité entre l'application et la base de données. Si la structure de données change, seule (en théorie) cette couche est à modifier.

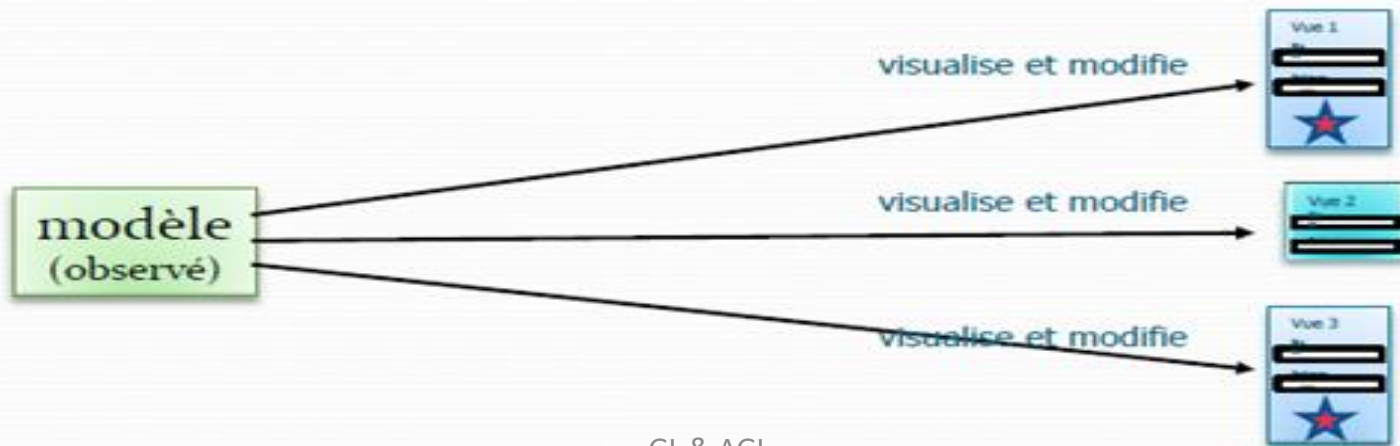


# ► Patron d'architecture : MVC



- **Problème :**

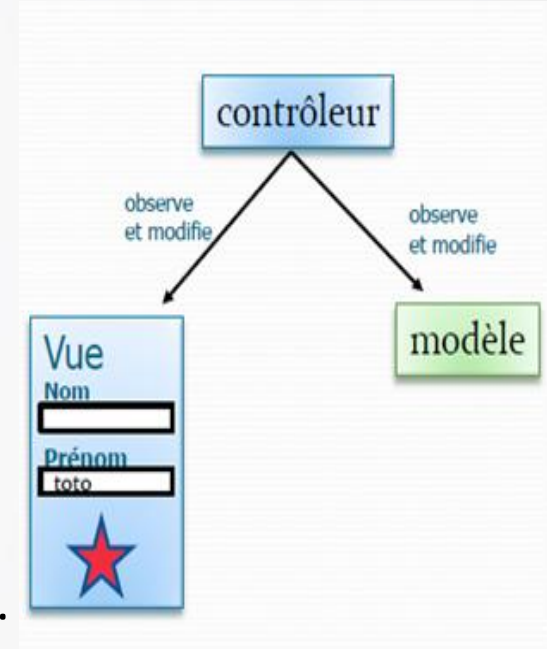
- Un modèle (=un ensemble de données) peut être visualisé à l'aide de différentes vues.
- Le modèle peut être modifié à partir de n'importe laquelle de ces vues.
- Quand le modèle est modifié, toutes les vues doivent être rafraichies.



# ► Patron d'architecture : MVC

- **Solution :**

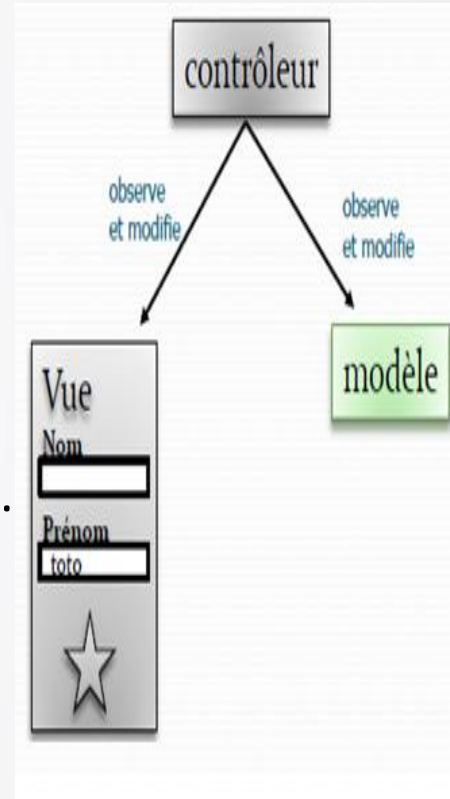
- On utilise 3 entités :
  - Modèle: contient les données à afficher
  - Vue: fait l'affichage
  - Contrôleur: coordonne les deux
- MVC : le plus connu des patrons d'architecture.
- Utilise le patron de conception observateur-observé.



# Exemple de patron d'architecture : MVC

- **Le Modèle :**

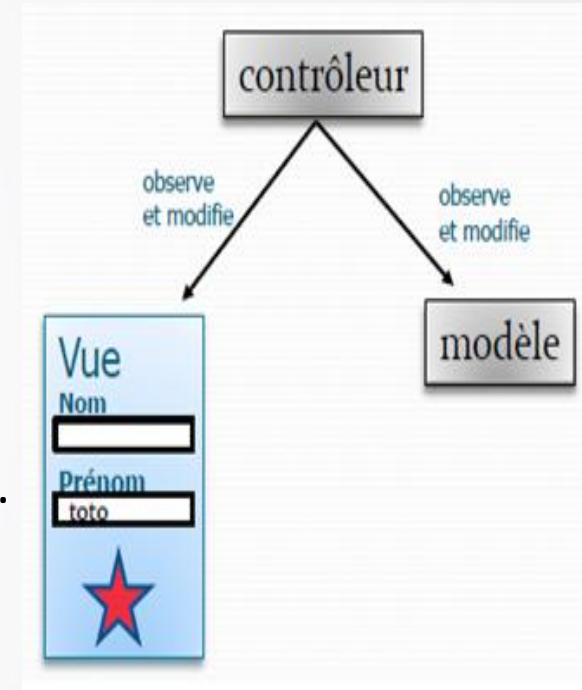
- Contient les données à afficher et à modifier.
- Définit la logique de manipulation de ces données.
- Envoie des événements quand les données sont modifiées.
- Ne connaît ni la vue ni l'API du contrôleur :
  - Le contrôleur et la vue sont des observateurs,
  - Le modèle est l'observé.



# ► Patron d'architecture : MVC

- **La Vue :**

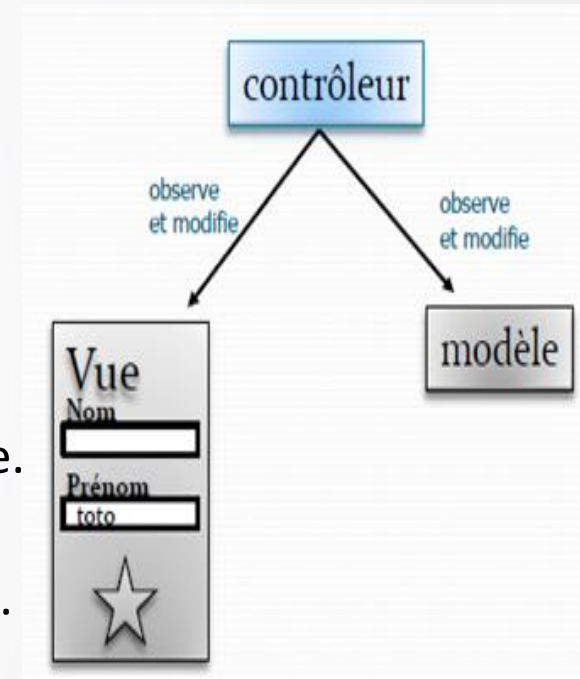
- Est chargée de l'affichage à l'écran.
- Envoie des événements correspondants aux actions de l'utilisateur (click, survol, sélection, etc.).
- Ne connaît ni l'API du contrôleur, ni le modèle.



# ► Patron d'architecture : MVC

- **Le contrôleur :**

- Assure l'interaction entre données et vue.
- Connait la vue et le modèle.
- Observe le modèle, modifie la vue en conséquence.
- Observe la vue, modifie le modèle en conséquence.

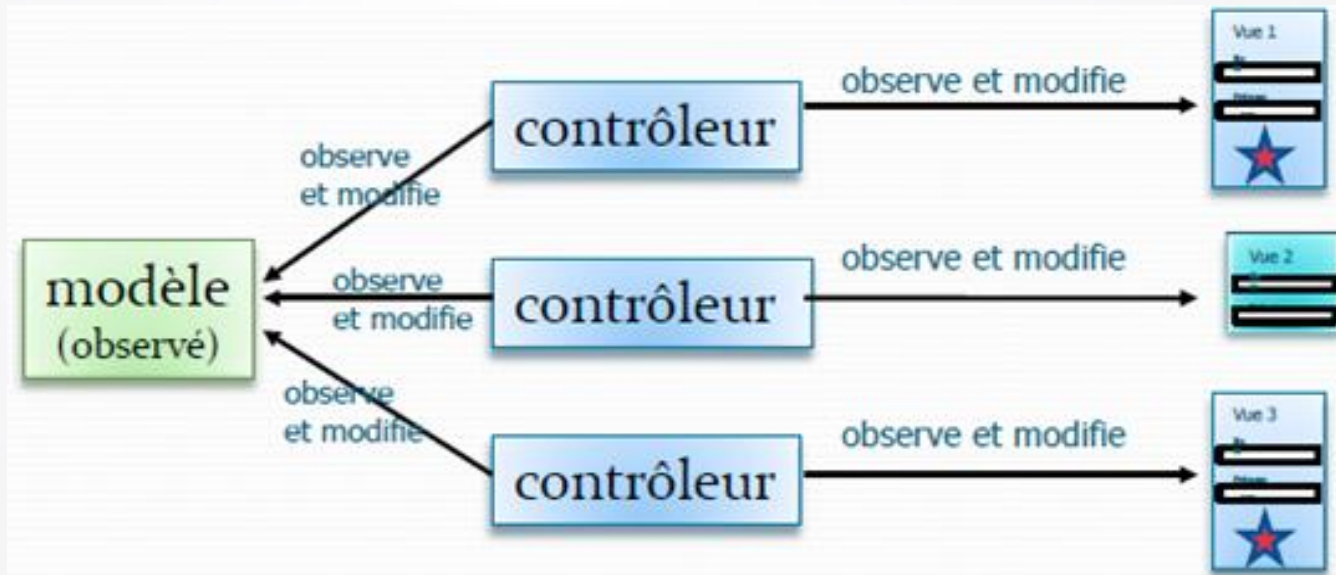


# ► Patron d'architecture : MVC



- **Avantages du MVC :**

- Il peut y avoir plusieurs vues sur le même modèle.
- Plusieurs contrôleurs peuvent modifier le même modèle.
- Toutes les vues seront notifiées des modifications.



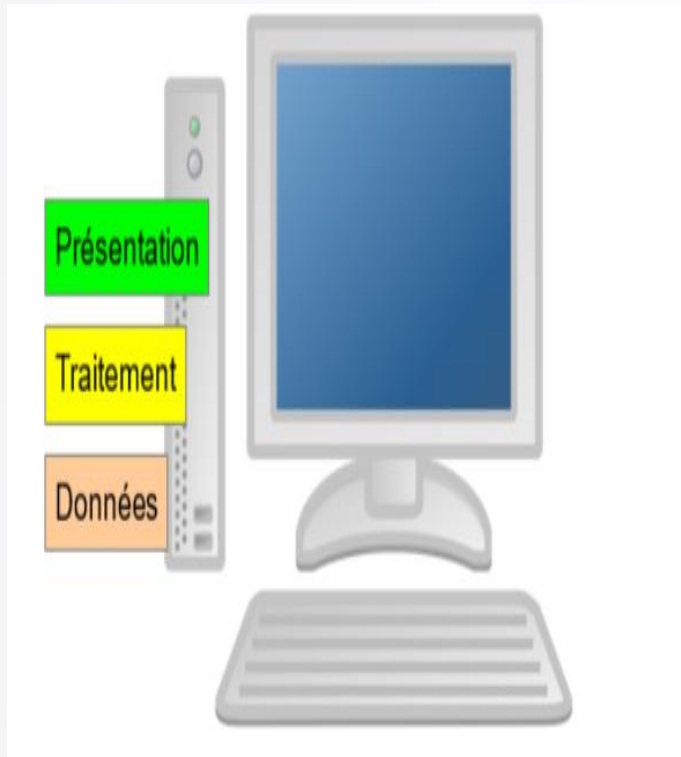




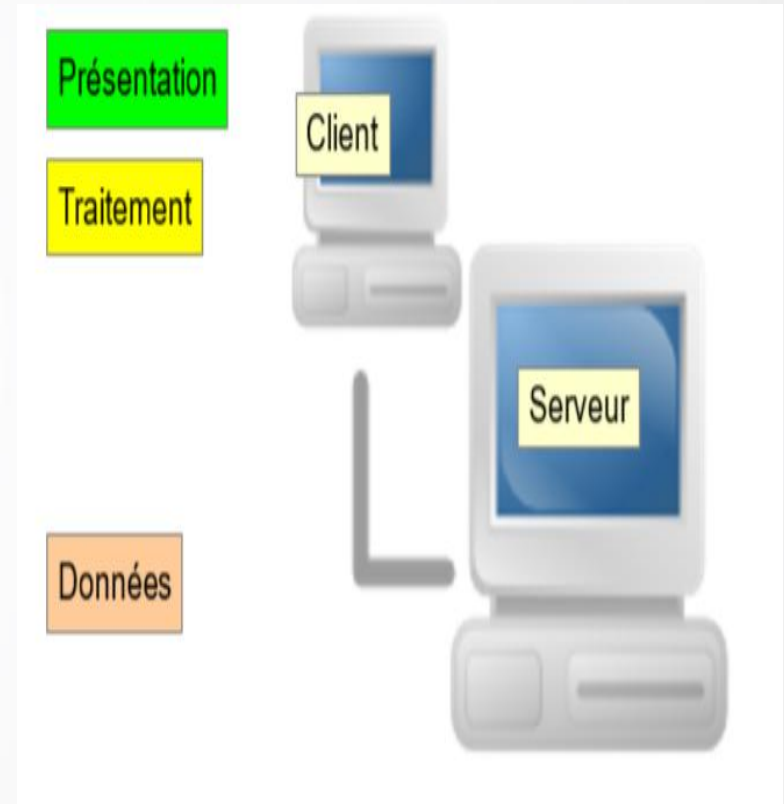
# L'architecture physique

- **Les architecture physique 1-niveau (1-tiers)** : Toutes les couches logiques sont situées sur la même machine (Exemple : un mainframe avec des postes passifs).
- **Les architecture 2-niveaux (2-tiers)** : Les couches logiques sont séparés sur deux sites :
  - Les dispositifs du serveur (BD, service de messagerie, etc.) sur un site
  - Le reste de l'architecture sur les postes client.
  - Permet le partage d'information entre utilisateurs : Accès simultanés, Synchronisation des données.
  - Des variantes existent pour alléger le poste client (Plus d'applicatifs sur le site serveur).

# ► L'architecture physique



Architecture 1-Tiers



Architecture 2-Tiers



# Qualité de la conception architecturale



- **Application de critères de qualité :**
  - Faible couplage.
  - Forte cohésion.
- **Utilisation de patrons de conception :**
  - Patrons GoF (Gang of Four).
  - 3 catégories de patrons GoF :
    - De création.
    - De structure.
    - De comportement.

# Critères de qualité de la conception architecturale

- **Problème** : comment réduire l'impact des modifications?

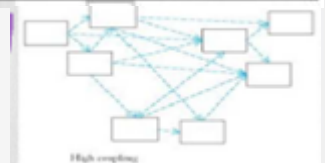
➡ Affecter les responsabilités de sorte à éviter tout couplage inutile.

- Le couplage est une mesure de degré auquel un élément est lié à un autre, en a connaissance ou en dépend.

- S'il y a couplage ou dépendance, l'objet dépendant peut être affecté par les modifications de celui dont il dépend.

- **Exemple** : une sous-classe est fortement dépendante de sa super-classe.

- Un objet A faisant appel aux méthodes d'un objet B a un couplage aux services de B



# Critères de qualité de la conception architecturale



- Un système a une cohésion si:
  - ✓ Les éléments inter reliés sont groupés ensemble.
  - ✓ Les éléments indépendants sont dans des groupes distincts.





# Patrons de conception

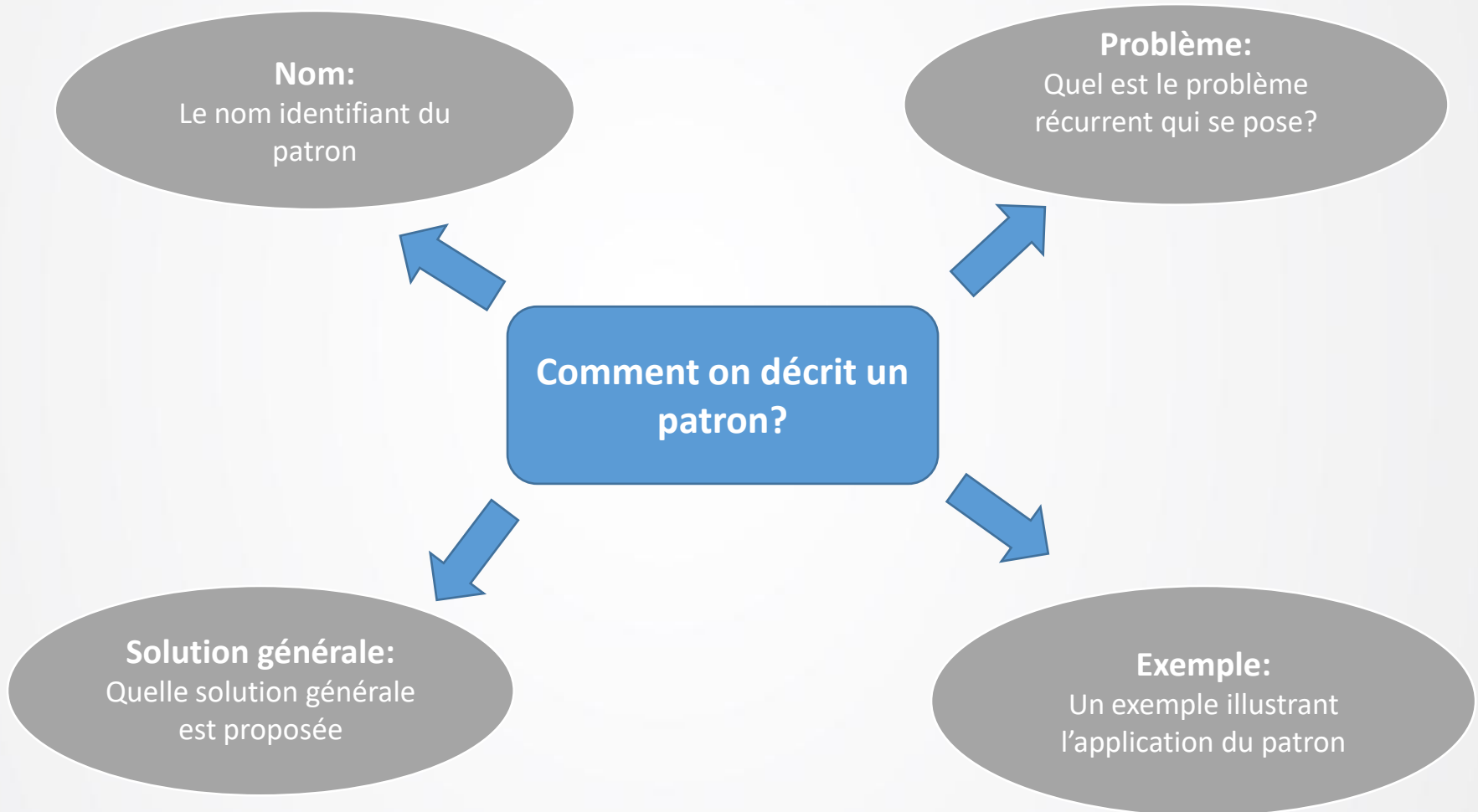
- La normalisation des architectures est grandement facilitée par l'utilisation de Framework et de Patrons (Patterns).
- Ils favorisent:
  - La réutilisation
  - La capitalisation d'expérience.



# Patrons de conception

- **Patrons (Patterns) :**
  - Solution générique à un problème (générique).
  - Solution éprouvée.
  - Permet de capitaliser sa propre expérience et celle des autres.
  - L'utilisation de patrons renforce l'abstraction.
    - Le patron fournit une solution à un problème (abstrait) indépendant du domaine.
  - Il existe différents types de patrons :
    - Les patrons d'analyse.
      - Fournissent des solutions réutilisables pour les étapes d'analyse.
    - Les patrons architecturaux.
      - Exp. : MVC (Model View Controller).
    - Les patrons de conception (Design Pattern).
      - Exp. : Singleton, Observer, Factory, etc.

# ► Patrons de conception







# Patrons de conception – Catégorie des patrons GoF

	Création	Structurels	Comportementaux
<b>Objectifs</b>	<p>Solutions aux problèmes liés à l'instanciation des classes</p> <p>Abstraction du processus d'instanciation</p> <p>Créer des objets sans avoir à connaître la logique de création</p>	<p>Solutions aux problèmes de structuration des classes, d'abstraction, de réutilisation</p> <p>Composition de classes et d'objets pour obtenir des structures plus complexes</p>	<p>Solutions aux problèmes de communication entre objets et d'algorithmique</p>
<b>Intérêt</b>	<p>Plus de flexibilité aux programmes</p> <p>Décider quel objet créer pour un cas d'utilisation donné</p>	<p>Définir des moyens de composer des objets pour obtenir de nouvelles fonctionnalités</p>	<p>Distribution des responsabilités</p>
<b>Exemples</b>	<p>Abstract Factory</p> <p>Builder</p> <p>Factory Method</p> <p>Prototype</p> <p>Singleton</p>	<p>Adapter</p> <p>Bridge</p> <p>Composite</p> <p>Decorator</p> <p>Facade</p> <p>Flyweight</p> <p>Proxy</p>	<p>Chain of Responsibility</p> <p>Command</p> <p>Interpreter</p> <p>Iterator</p> <p>Mediator</p> <p>Memento</p> <p>Observer</p> <p>State</p> <p>Strategy</p> <p>Template Method</p> <p>Visitor</p>

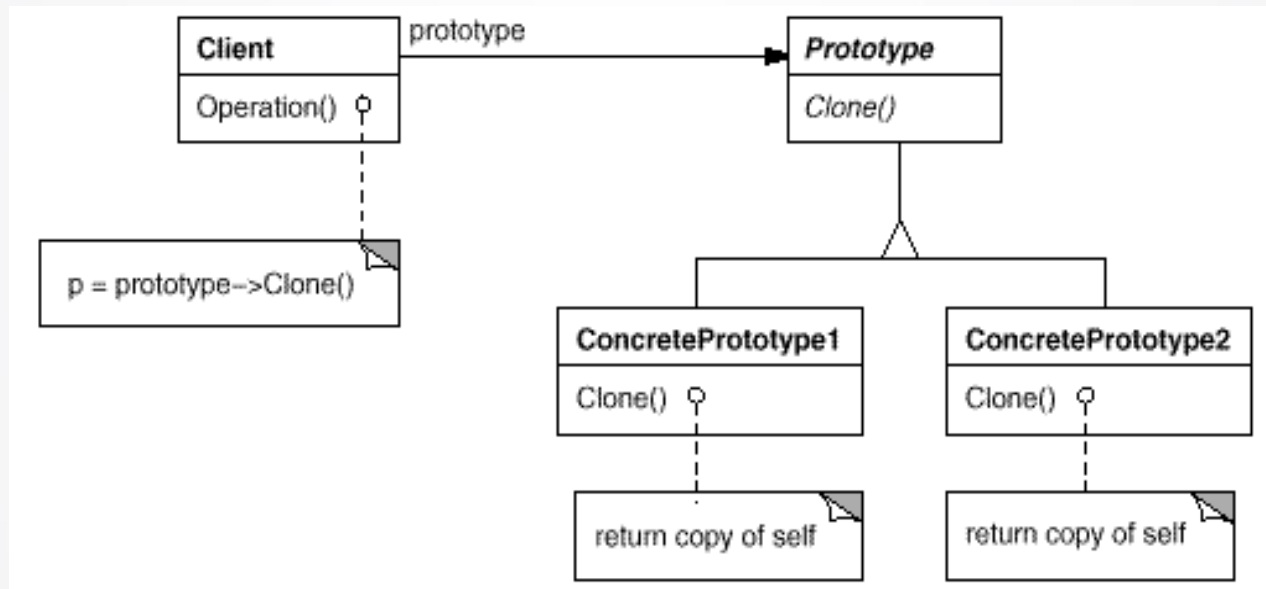


# Exemple 2 de patron de conception de création

- **Prototype :**
  - Spécifie les types d'objets à créer en utilisant un prototype.
  - Créer de nouveaux objets en copiant le prototype (clonage).
  - Fournir de nouveaux objets par la **copie d'un exemple** plutôt que de produire de nouvelles instances non initialisées d'une classe.

# Exemple 2 de patron de conception de création

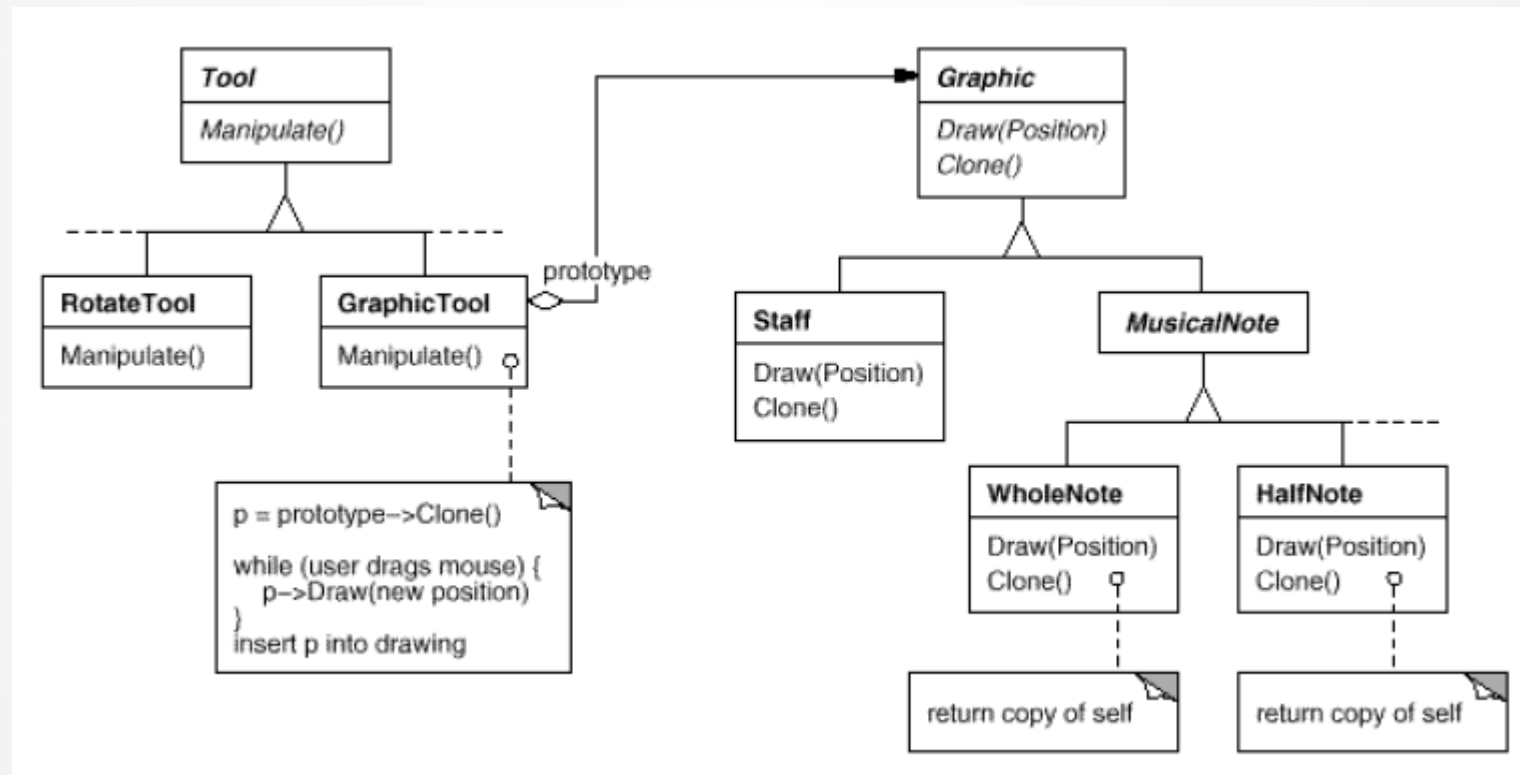
- **Structure du prototype :**





# Exemple 2 de patron de conception de création

- Exemple avec Prototype :

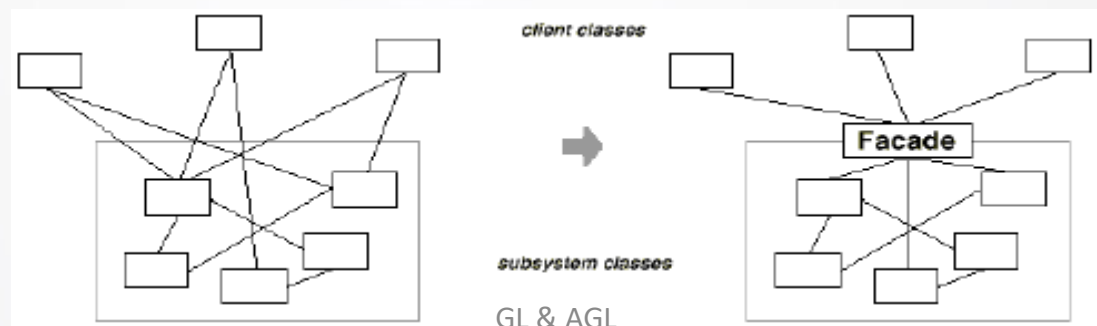




# Exemple 2 de patron de conception de structure

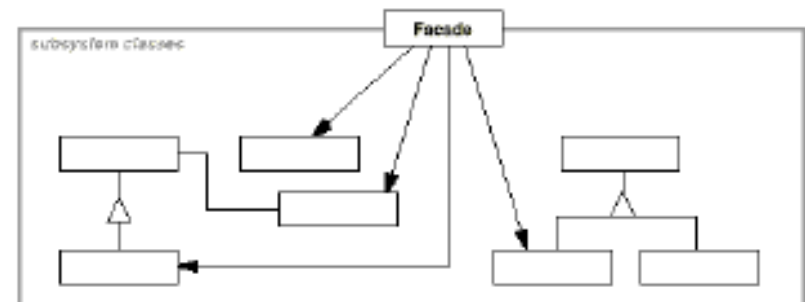
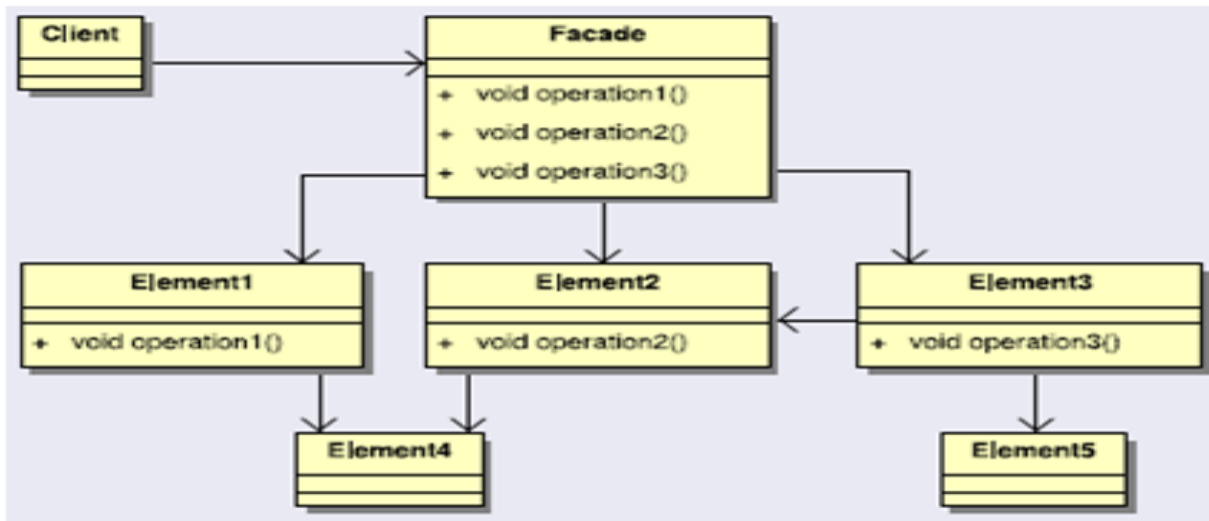
- **Façade :**

- Cacher la complexité d'un système.
- Minimiser les communications et les dépendances entre sous-systèmes.
- Fournir une interface au client à travers laquelle il pourra accéder au système.
- Fournir au client des méthodes simples à travers une classe unique.
- Déléguer les appels aux méthodes des classes du système.



# Exemple 2 de patron de conception de structure

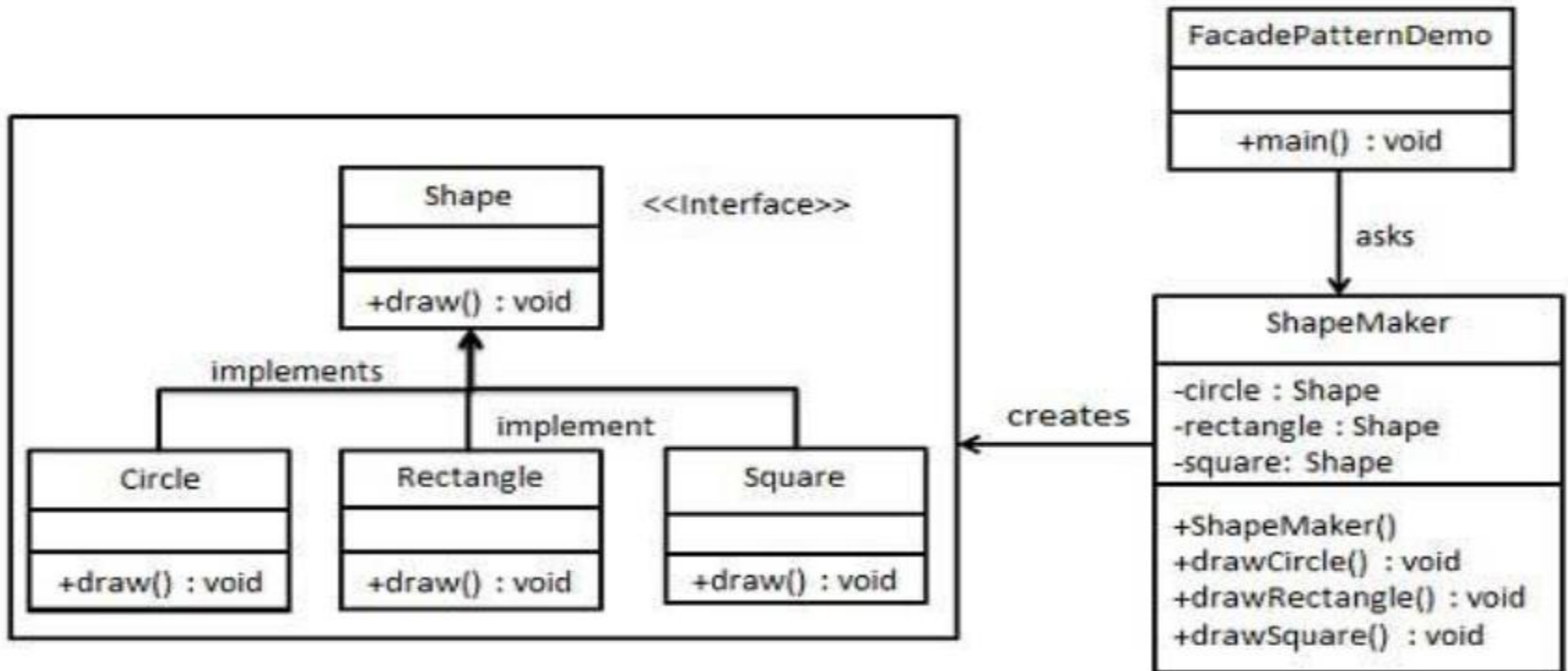
- **Structure de Façade :**





# Exemple 2 de patron de conception de structure

- Exemple avec Façade :



# Exemple 2 de patron de conception de comportement

- **Observer :**

- Définit une dépendance « one-to-many » (un à plusieurs) entre objets de sorte que lorsque l'état d'un objet change, tous ses objets dépendants sont **notifiés et mis à jours** automatiquement.
- L'objet observé (Observable) gère une liste d'observateurs (Observer) dotés d'une méthode de mise à jour (update) et notifie les changements aux observateurs en appelant leurs méthodes de mise à jour.

