



Chapitre 4 : Architecture & Conception de logiciels

UP GL-BD



Objectifs Chapitre 4

- Concevoir l'architecture logique et physique d'un logiciel.
- Etablir la conception détaillée d'un logiciel.
- S'assurer de l'adhérence aux exigences.
- Garantir la qualité du logiciel.



Plan Chapitre 4

- **Partie 1 : Conception logicielle :**
 - Conception logicielle - Définition.
 - Processus de conception.
 - Niveaux conceptuels.
- **Partie 2 : Architecture d'un logiciel :**
 - Architecture d'un logiciel - Définition.
 - Types d'architectures.
 - Patrons d'architecture.
 - Architecture physique.
 - Critères de qualité et patrons de conception.



► Partie 1 – Conception logicielle

- I. Conception logicielle - Définition.
- II. Processus de conception.
- III. Niveaux conceptuels
 - a. Conception globale
 - b. Conception détaillée



► Conception logicielle - Définition

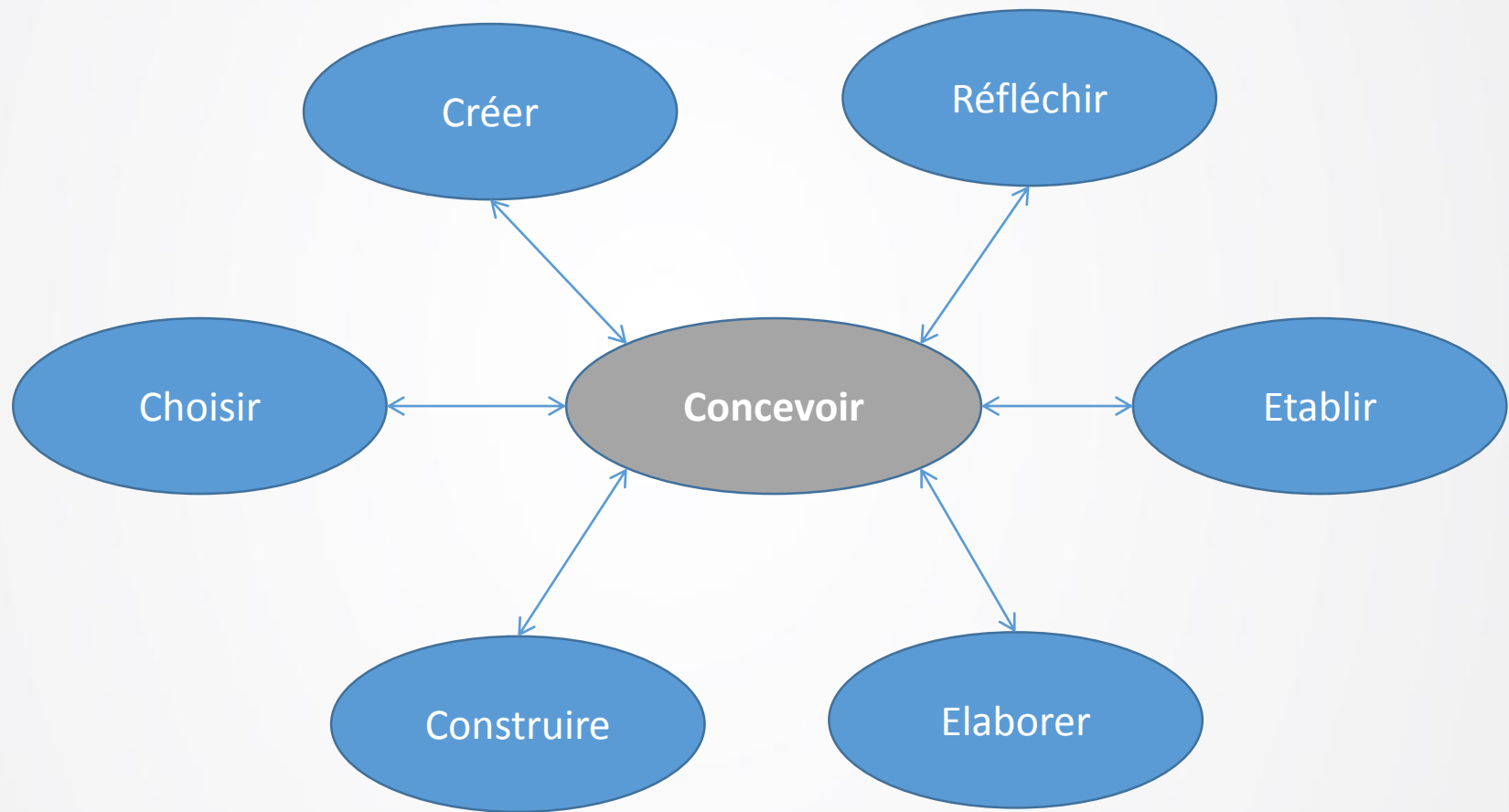
- **Spécification :**

- Qu'est-ce que le logiciel doit faire?
- Comment s'assurer qu'il le fait?
- Comment s'assurer qu'on développe le bon logiciel?

- **Conception :**

- 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



Processus de conception



- **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

► Processus de conception



Modéliser le logiciel dans son futur système



Méthode



Formalisme



Outils



Processus de conception

- **Modèle** : Abstraction du monde réel :
 - **Analyser avant de construire.**
 - Retrouver les informations nécessaires lors de modifications ou extensions.
 - **Simplifier la complexité** du problème d'origine.
 - Offrir des points de vue et **des niveaux d'abstraction** plus au moins détaillés selon les besoins.



Processus de conception

- **Exemple :**
 - **Modèle** : orienté objets.
 - **Méthode** : 2TUP (Two Track Unified Process).
 - **Formalisme** : Diagrammes UML.
 - **Outils** : Rational Rose, StarUML, etc.



► Niveaux conceptuels

Conception architecturale globale

- Architecture de haut niveau.
- Structure et organisation générale du système à concevoir.
- **Décomposer le logiciel en composants** plus simples, définis par leurs **interfaces** 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.



▶ Partie 2 – Architecture d'un logiciel

- I. Architecture d'un logiciel - Définition
- II. Types d'architectures
- III. Patrons d'architecture
- IV. Architecture physique
- V. Qualité de la conception architecturale
 - a. Critères de qualité
 - b. Patrons de conception



Architecture d'un logiciel – Définition



- Processus de conception et d'organisation globale du système incluant :
 - La **subdivision** du logiciel en **sous-systèmes**.
 - Les **décisions** à prendre concernant leurs **interactions**.
 - La détermination des **moyens d'interactions** (interfaces, connecteurs, protocoles, etc.).
- Impact de l'architecture sur l'efficacité, la réutilisabilité et la maintenabilité du système.

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.

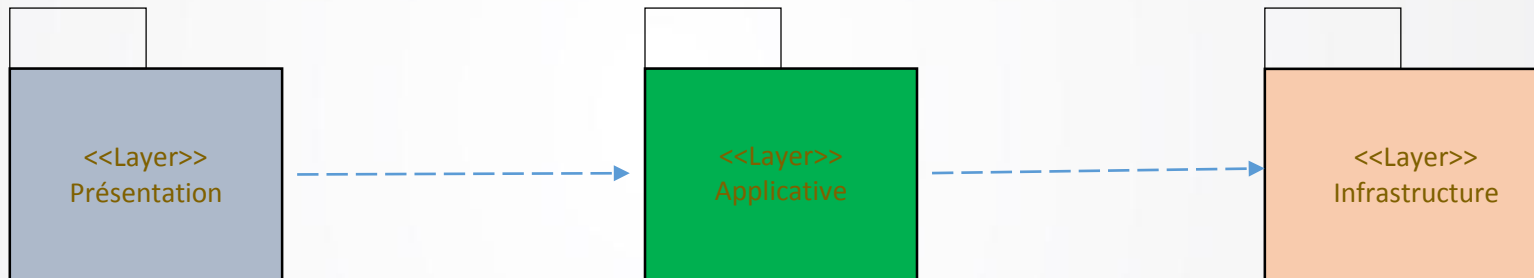
Exemples :

- **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

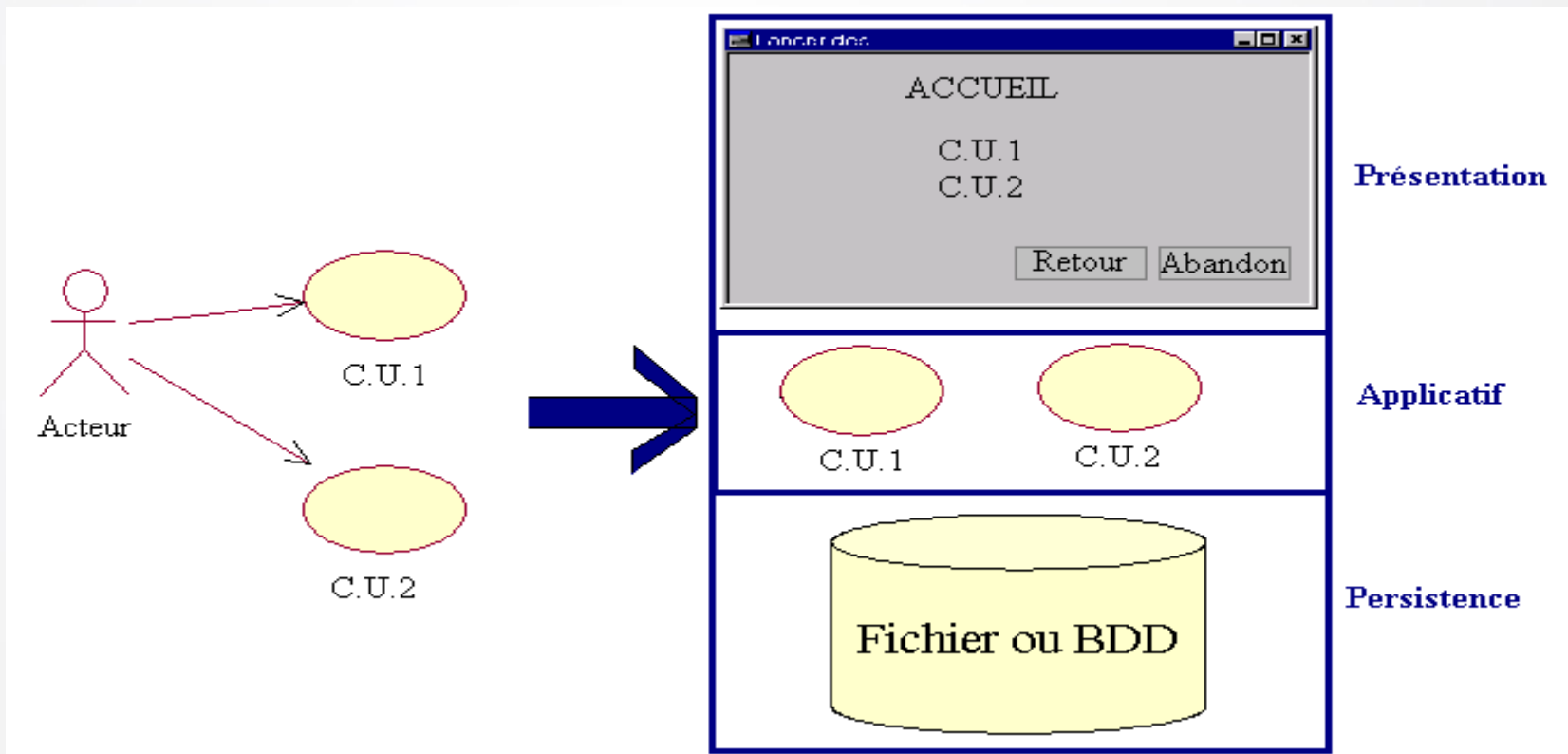
18

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 la base de donnée (couche infrastructure) sans remettre en question les autres niveaux.

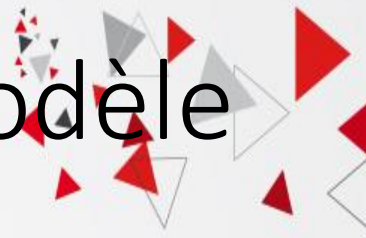
- L'architecture en 3 couches définit une dépendance bi-directionnelle entre « IHM » et « Metier »
- La « BDD » n'a aucune dépendance sur la couche « Metier » donc les modifications que nous apporterons la couche « Metier » n'auront pas d'impact sur « BDD ».

➡ En revanche le package « BDD » devra répondre aux besoins de « Metier ».



Découpage en couches – Modèle en 3 couches

21

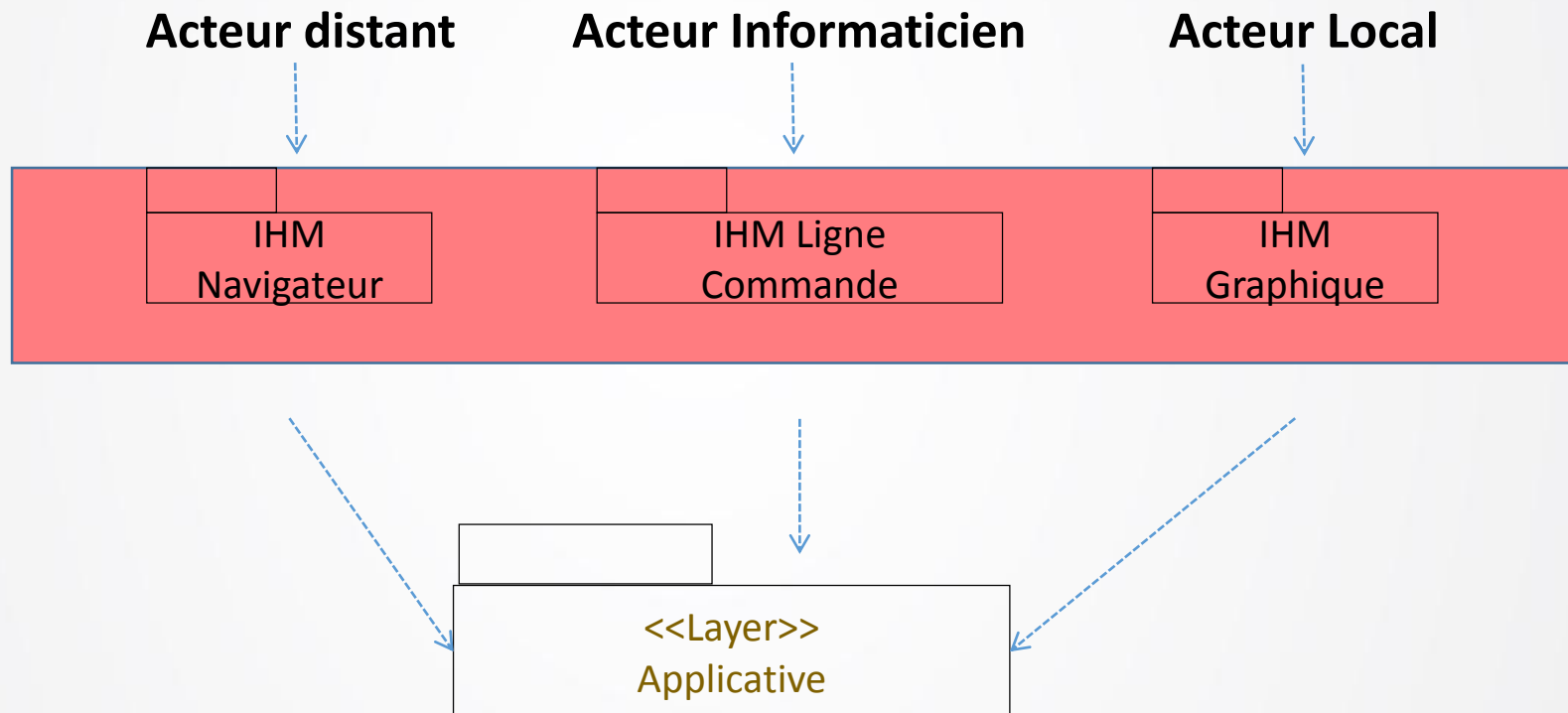


- **Couche présentation :**


- 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.
- Une application peut avoir plusieurs présentations (incarnations) :
 - Une couche présentation basée sur **une interface graphique** (swing ou GTK).
 - Une couche présentation basée sur l'utilisation d'un **navigateur web** (html, jsp, php, etc.).
 - Une interface de **commandes en ligne**.
- Chaque incarnation est contenu dans un paquetage indépendant.

Découpage en couches – Modèle en 3 couches

22



Exemple : Plusieurs présentations d'une application




▶ Découpage en couches – Modèle en 3 couches

23

- **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

24

- **Couche infrastructure :**

- Ce niveau se compose de serveurs de bases de données.
- Permet de stocker les informations et de les récupérer.
- Conserve les données neutres et indépendantes de serveurs d'applications ou de la logique métier (couche applicative).
- Elle peut être :
 - Des acteurs systèmes externes,
 - Des serveurs BD,
 - Des systèmes de messagerie, etc.

▶ Découpage en couches – Modèle en 3 couches

25

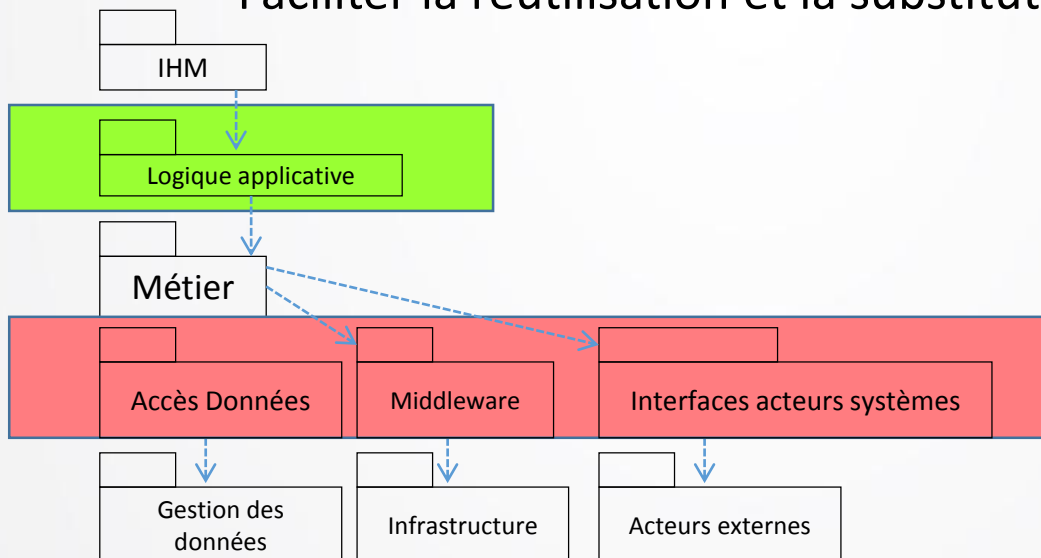


- Découpage conforme à une démarche structurée par les cas d'utilisation.
 - On peut s'occuper d'une couche sans savoir à connaître le détail des autres couches.
 - Minimise les indépendances entre couches.
 - Favorise la standardisation (framework).
 - Facilite la réutilisation et la substitution (couplage 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

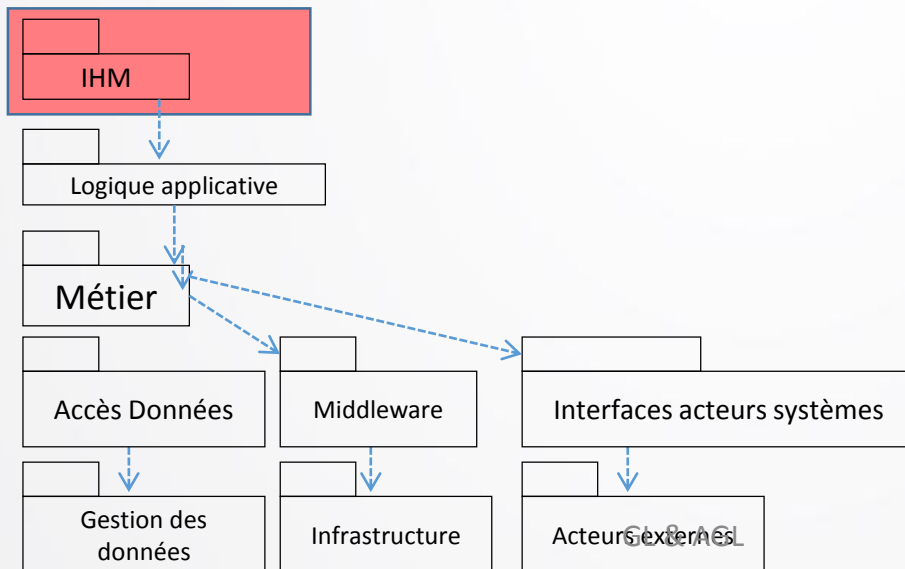
- **Autre découpage en couches : Modèle en 5 couches**

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



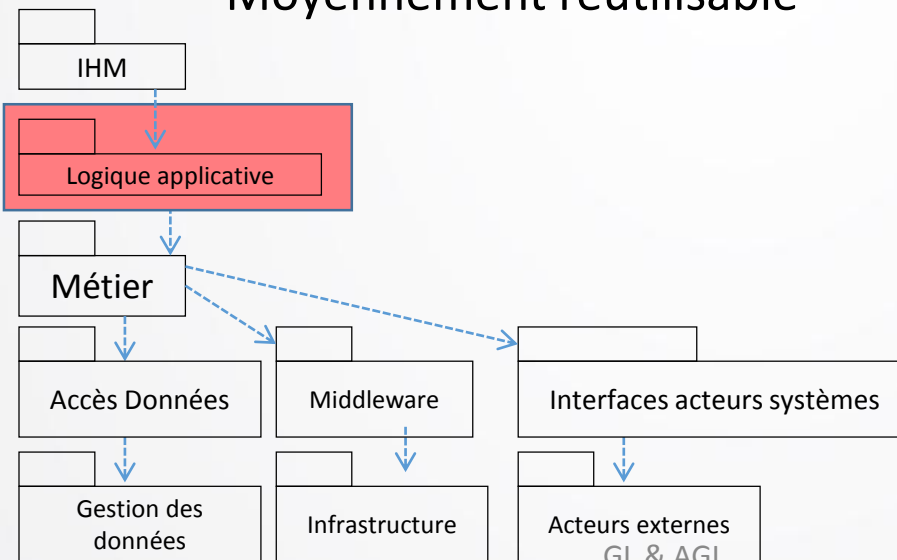
Découpage en couches – Modèle en 5 couches

- **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 événements sous forme d'envoi de message à destination de la couche applicative.
 - Récupérer la réponse et afficher les résultats.



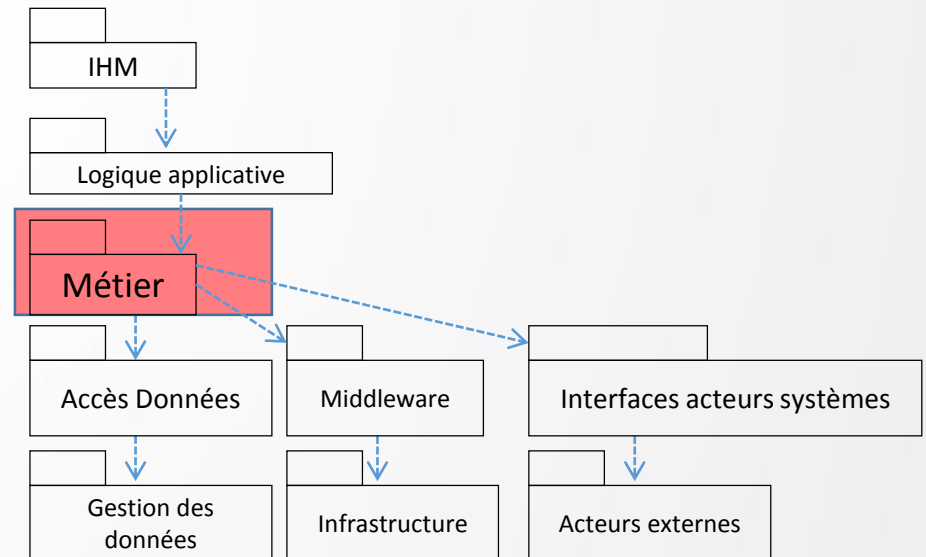
Découpage en couches – Modèle en 5 couches

- **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



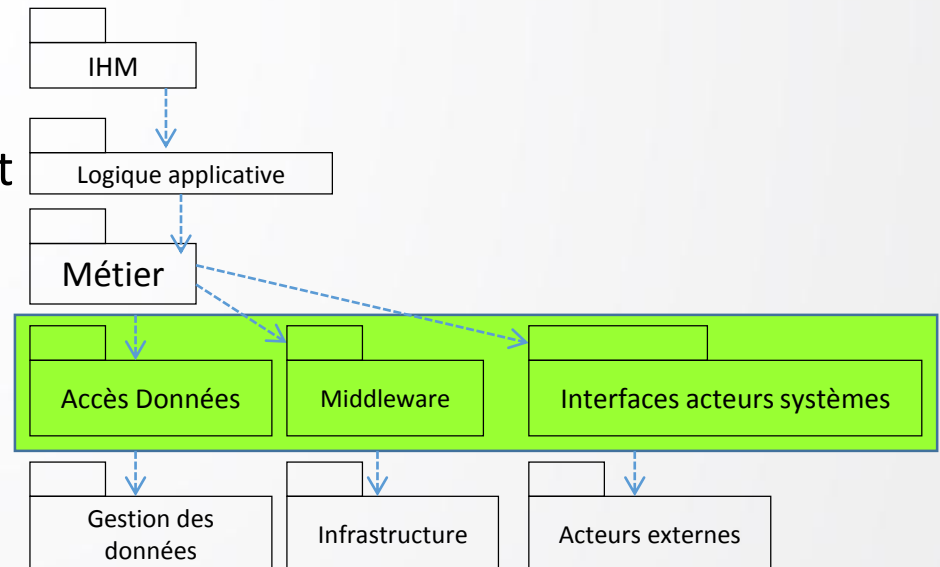
▶ Découpage en couches – Modèle en 5 couches

- **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 : consiste en la partie gérant l'accès aux données du système :**
 - Ces données peuvent être propres au système, ou gérées par un autre système.
 - La couche métier n'a pas à s'adapter à ces deux cas, ils sont transparents pour elle, et elle accède aux données de manière uniforme.
 - Permet d'éviter un couplage trop fort entre le modèle Objet et le modèle Physique des données.

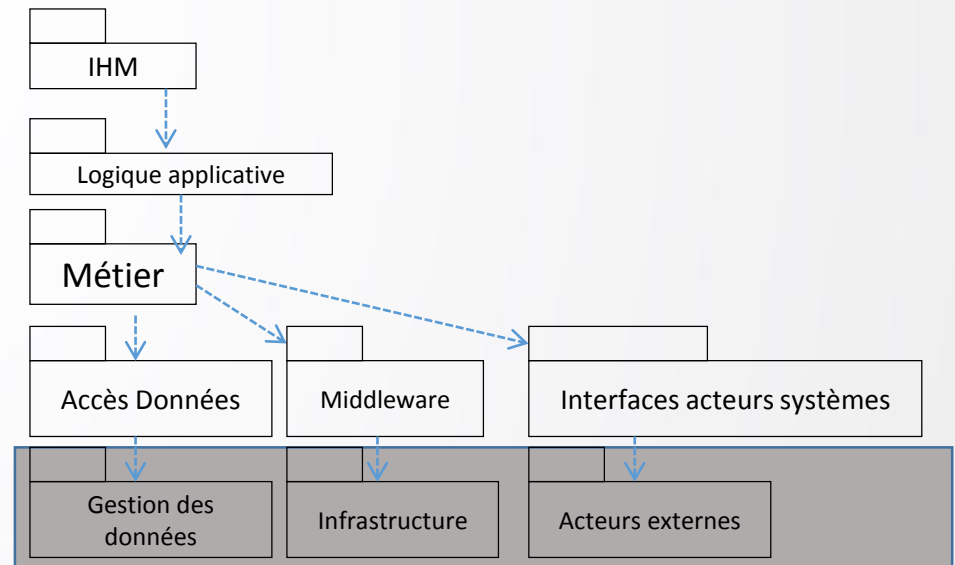


▶ Découpage en couches – Modèle en 5 couches

- **La couche de gestion des données : Stocker les données de manière persistante.**

Exemples :

- Base de données relationnelle, SGBD/R.
- Base de données Objet.
- Fichiers.

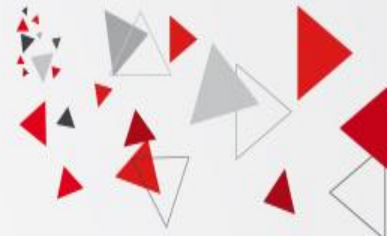




Un autre exemple d'architecture en couches: Système d'information orienté web

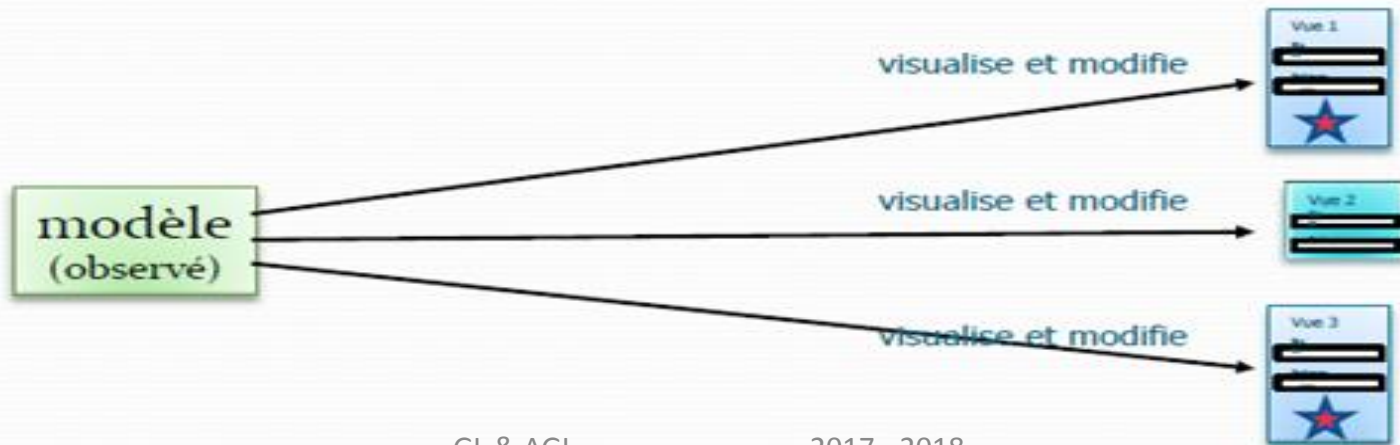
<u>Nom du niveau</u>	<u>Responsabilité</u>	<u>Technologie d'implémentation</u>
Présentation	Affichage interface utilisateur	JSP/ASP/PHP/HTML/DHTML/EcmaScript,Flash
Application	Flux cas d'utilisation UI, validation, interaction avec les services	Servlets, Script CGI
Services	Contrôle transactionnel, logique métier/flux procédural, comportement façade	Composant de session : Bean Session, Web Service
Domaine	Modèle du domaine, logique domaine/métier, validation sémantique	Composant d'entité : Bean entité, simple objet, SDO
Persistance	Stockage persistant de l'état des objets du domaine	O/R mappers, BDDOO, EJB CMP/BMP, JDO

► Patron d'architecture : MVC



- **Problème :**

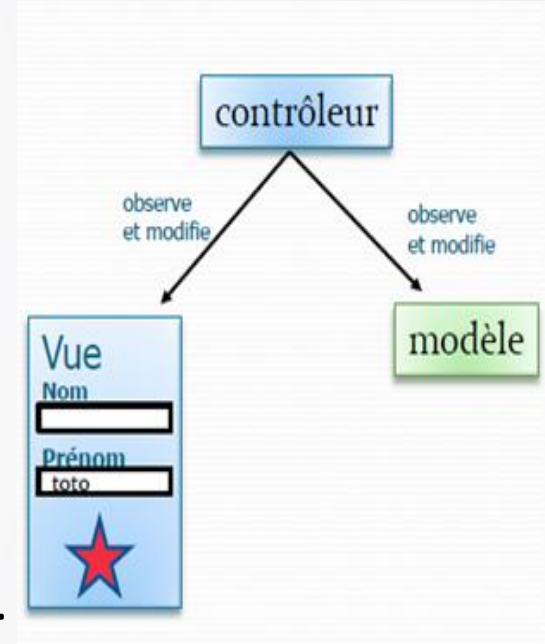
- Un modèle (=un ensemble de données) peut être visualisé à l'aide de différentes vues.
- Le modèle peut être modifié à partir de n'importe laquelle de ces vues.
- Quand le modèle est modifié, toutes les vues doivent être rafraichies.



► Patron d'architecture : MVC

- **Solution :**

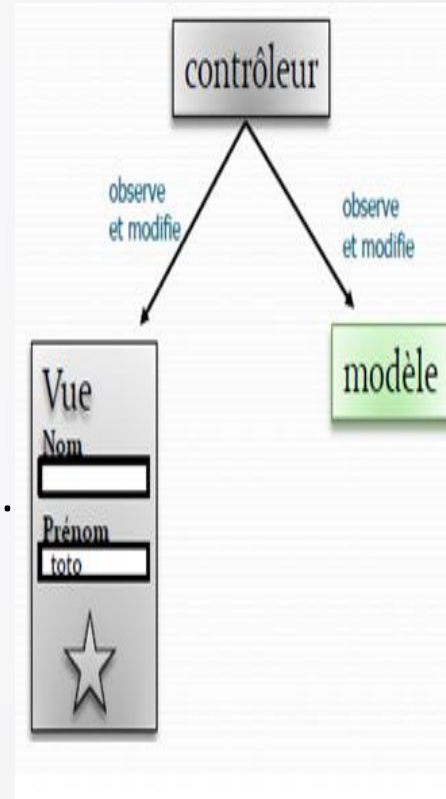
- On utilise 3 entités :
 - Modèle: contient les données à afficher
 - Vue: fait l'affichage
 - Contrôleur: coordonne les deux
- MVC : le plus connu des patrons d'architecture.
- Utilise le patron de conception observateur-observé.



Exemple de patron d'architecture : MVC

- **Le Modèle :**

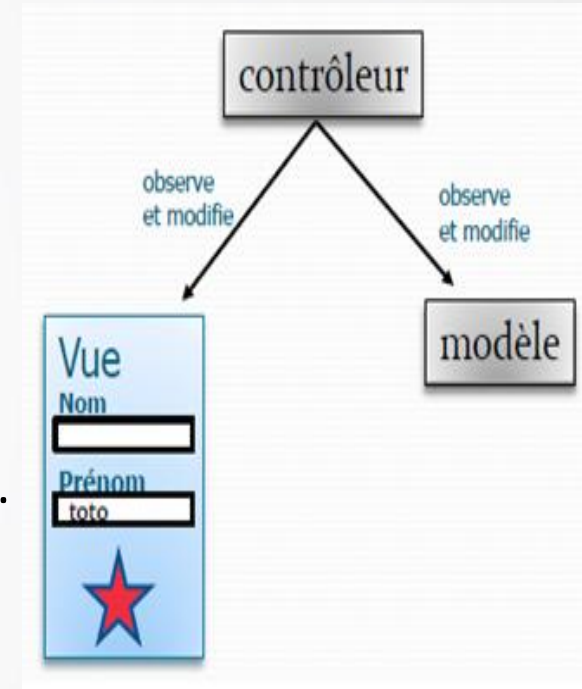
- Contient les données à afficher et à modifier.
- Définit la logique de manipulation de ces données.
- Envoie des événements quand les données sont modifiées.
- Ne connaît ni la vue ni l'API du contrôleur :
 - Le contrôleur et la vue sont des observateurs,
 - Le modèle est l'observé.



► Patron d'architecture : MVC

- **La Vue :**

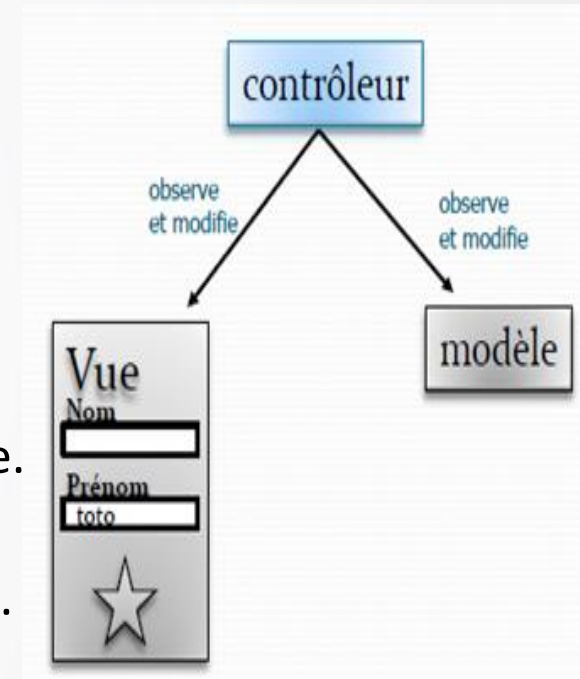
- Est chargée de l'affichage à l'écran.
- Envoie des événements correspondants aux actions de l'utilisateur (click, survol, sélection, etc.).
- Ne connaît ni l'API du contrôleur, ni le modèle.



► 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.

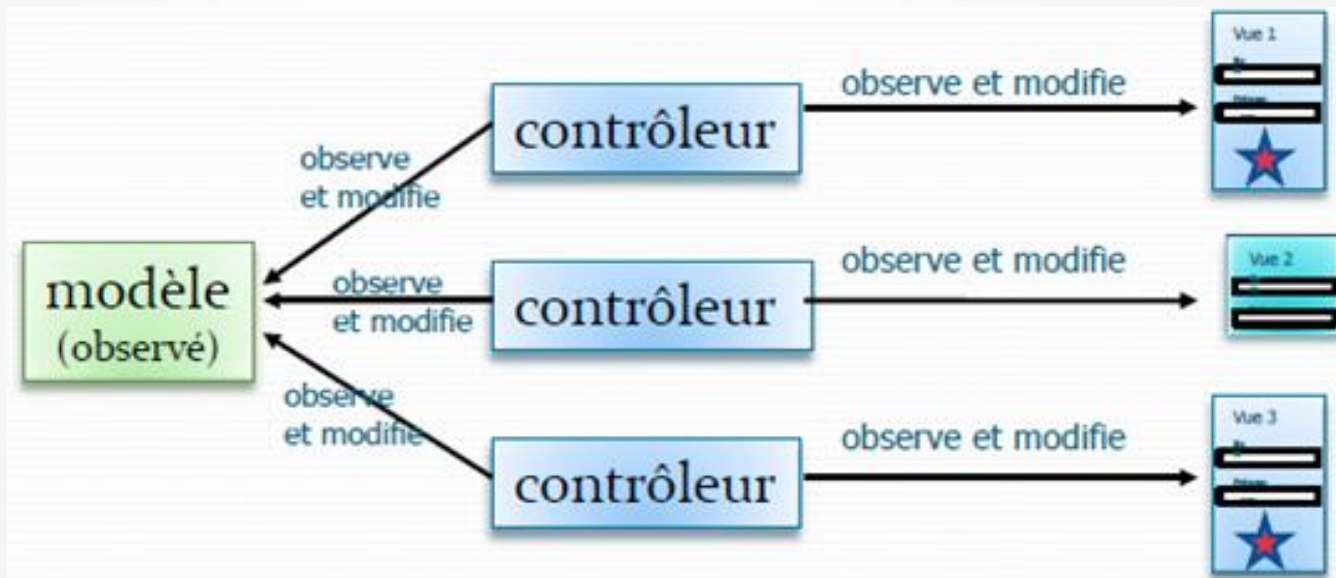


► Patron d'architecture : MVC



- **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.

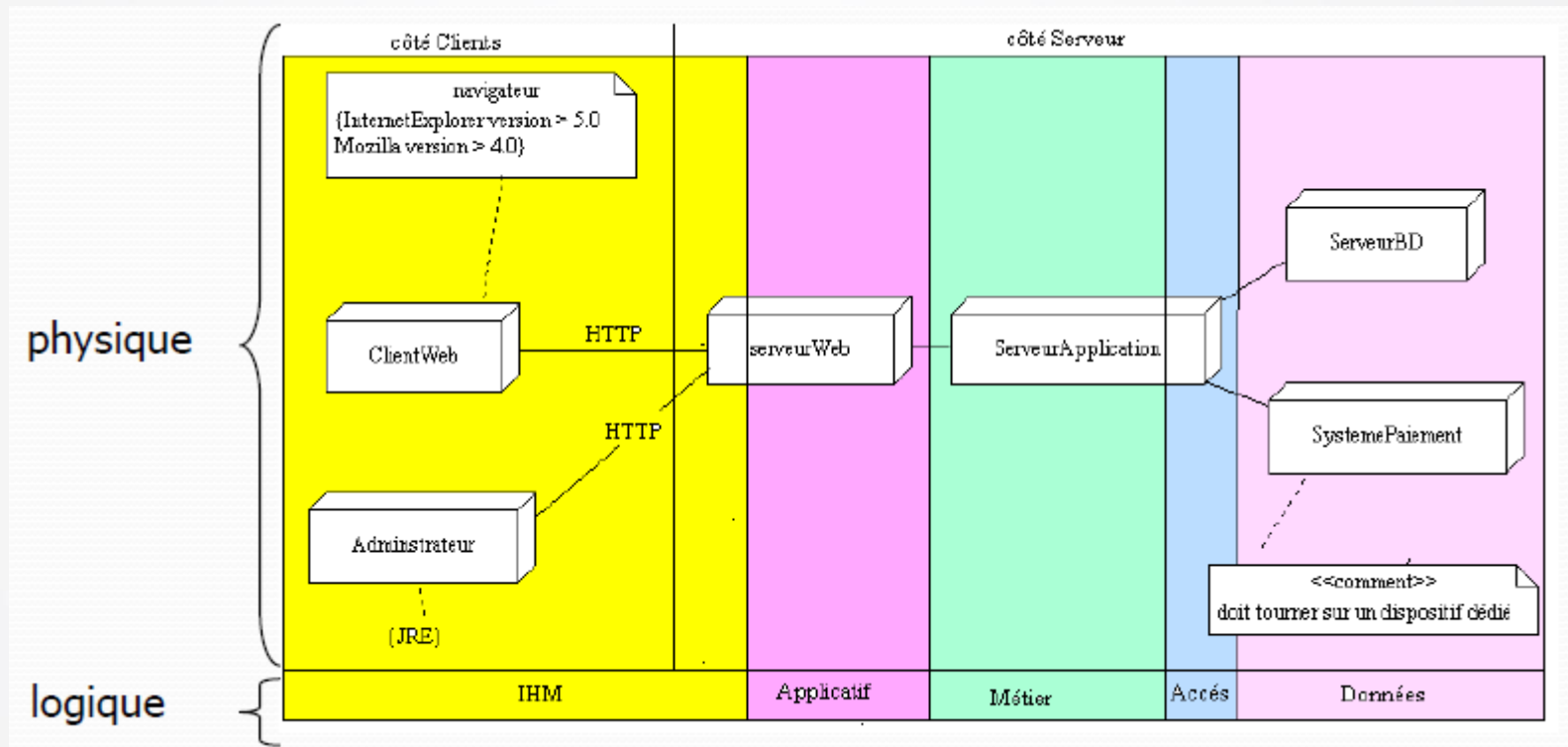




L'architecture physique

- **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



Exemple : Répartition de l'architecture logique sur l'architecture physique



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?

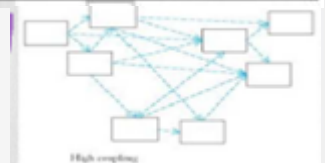
➡ Affecter les responsabilités de sorte à éviter tout couplage inutile.


- 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



• *Exemple : Facturation*

Dans un logiciel de gestion de vente, nous avons les classes suivantes :

- **Facture** : Contient un ensemble de produits facturés et est associée à un mode de paiement.
- **Paielement** : Décrit un mode de paiement (espèces, chèque, carte bancaire, à crédit).
- **Client** : Effectue les commandes.

On ajoute une méthode `payer()` à la classe Client. On étudie le couplage dans les deux cas suivants :

1. La méthode `payer()` créer une instance de Paielement et l'assigne à l'objet Facture.
2. La méthode `payer()` délègue l'action à la classe Facture qui crée une instance de Paielement et l'assigne.

Critères de qualité de la conception architecturale



Le couplage est plus faible dans le deuxième cas car la méthode `Payer()` de la classe `Client` n'a pas besoin de savoir qu'il existe une classe `Paiement`, c'est-à-dire qu'elle ne dépend pas de l'existence ou non de cette classe

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.



- **Exemple :**

- Considérons la modélisation d'une classe Eléphant ayant des propriétés particulières (hauteur et température corporelle).
- Ajouter des classes de type énumération relatives à l'hauteur et à la température.



Patrons de conception

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



Patrons de conception

- **Framework (ou canevas) :**

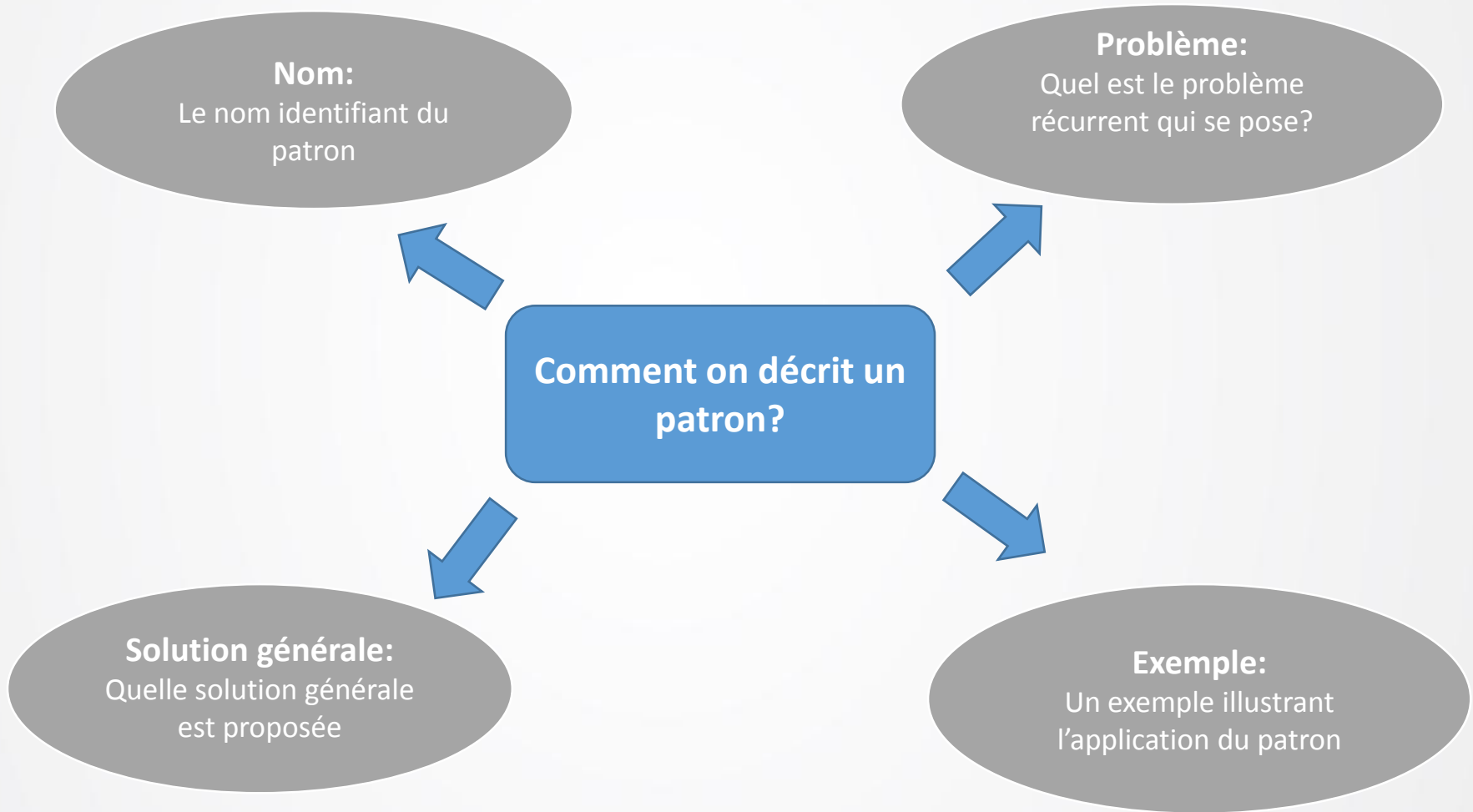
- C'est un squelette qui peut être utilisé ou adapté pour une famille d'applications.
- Il met en œuvre un modèle.
- Il comprend un ensemble de classes, souvent abstraites
 - À adapter à des environnements ou contraintes spécifiques.
- Est plus orienté implémentation que les patrons.
- Se situe au niveau des composants:
 - Fichier source, exécutable, fichier XML, répertoires, etc.
 - Par comparaison, le patron se situe au niveau du concept.
- Un Framework utilise des patrons
- Exemple de frameworks :
 - J2EE, Spring, Struts, GEF, etc.



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
Objectifs	<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>
Intérêt	<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>
Exemples	<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 1 de patron de conception de création

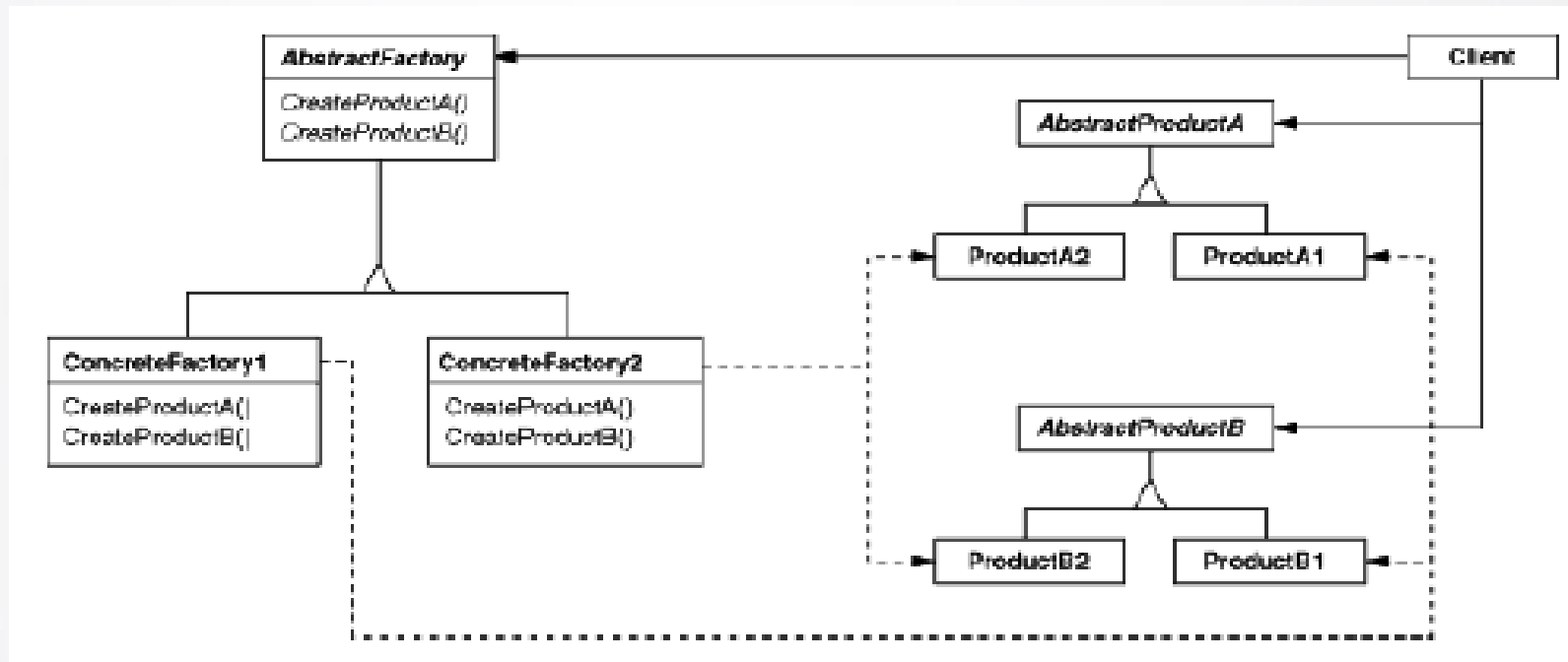
- **Abstract Factory :**

- Une super-fabrique permettant de créer d'autres fabriques.
- Création des fabriques à travers une interface.
- Les objets d'une fabrique sont créés sans avoir à expliciter leurs classes.



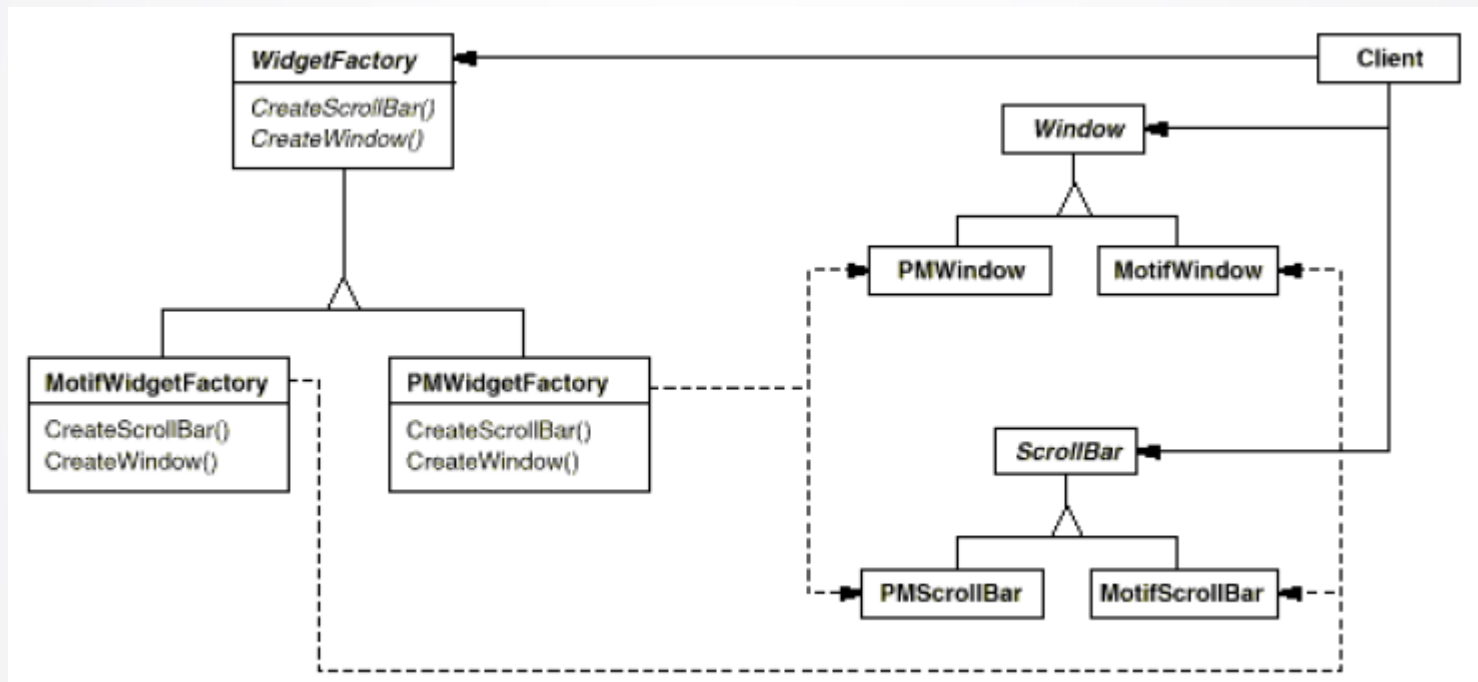
Exemple 1 de patron de conception de création

- **Structure de Abstract Factory :**



Exemple 1 de patron de conception de création

- Exemple avec Abstract Factory :





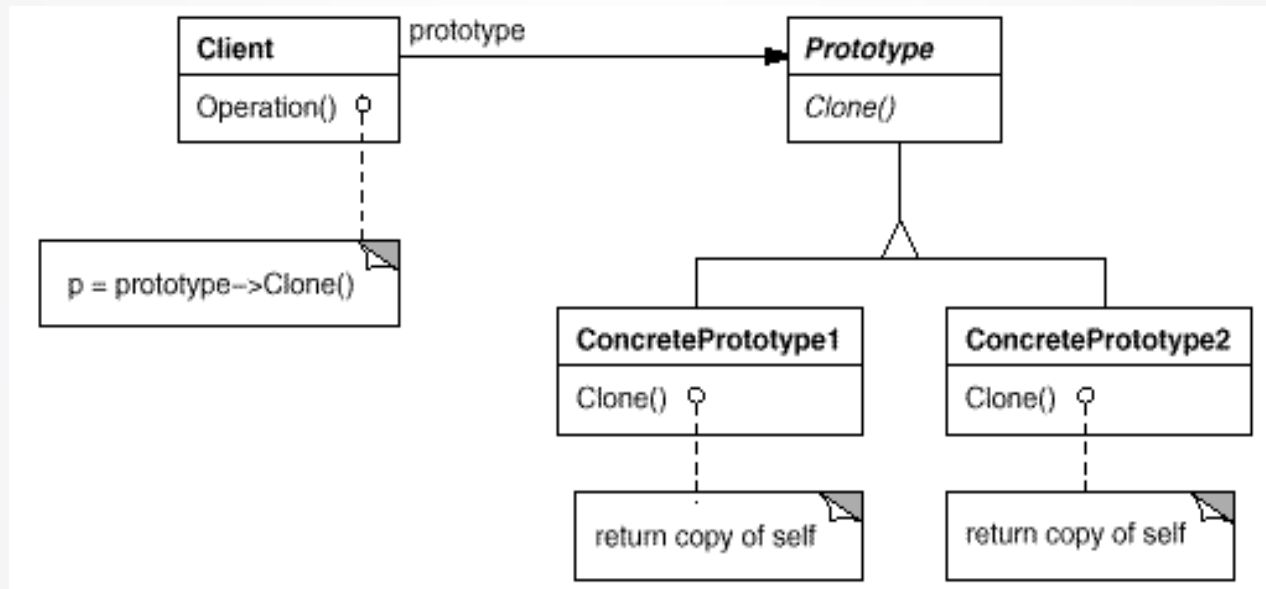
Exemple 2 de patron de conception de création

- **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

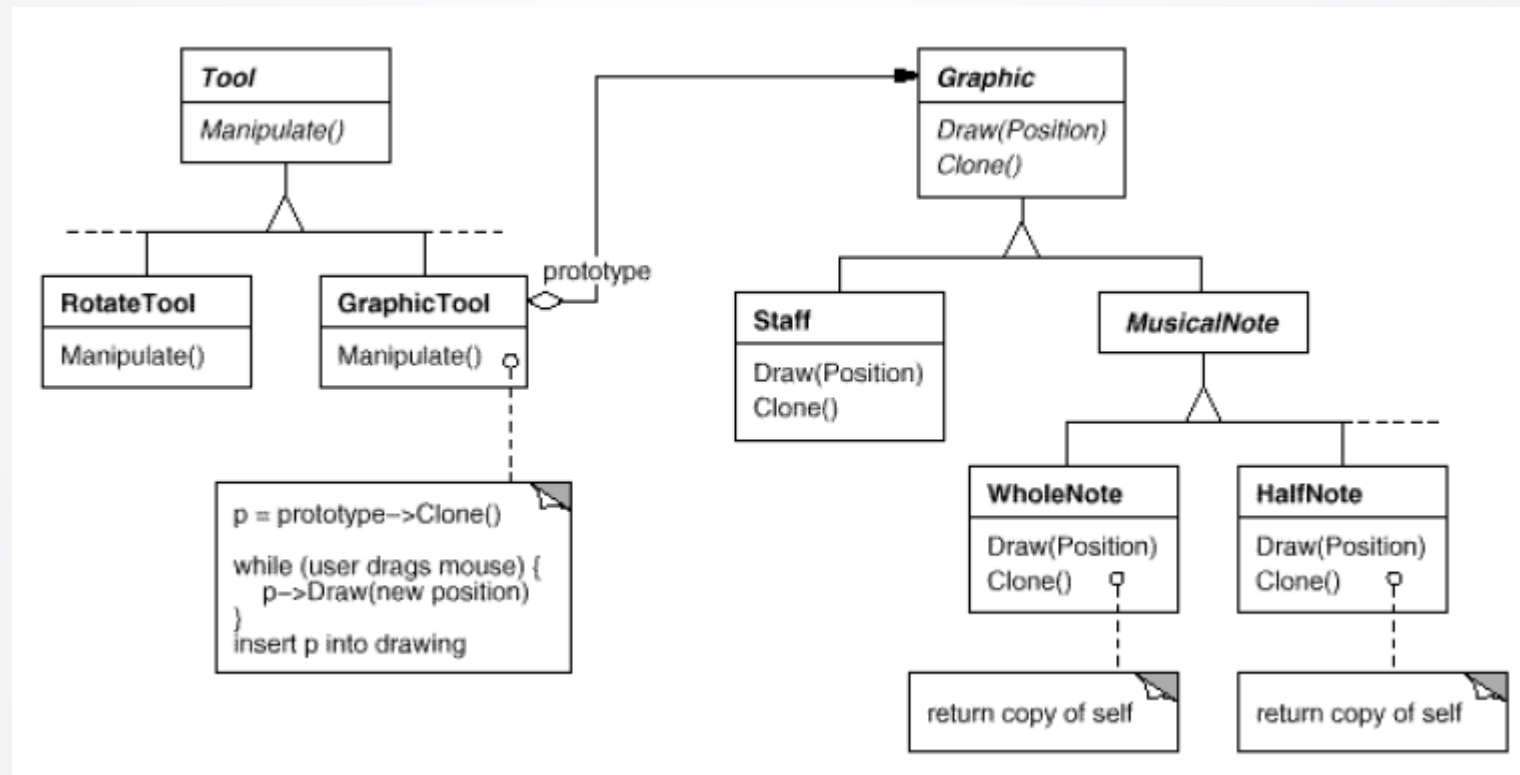
- **Structure du prototype :**





Exemple 2 de patron de conception de création

- Exemple avec Prototype :

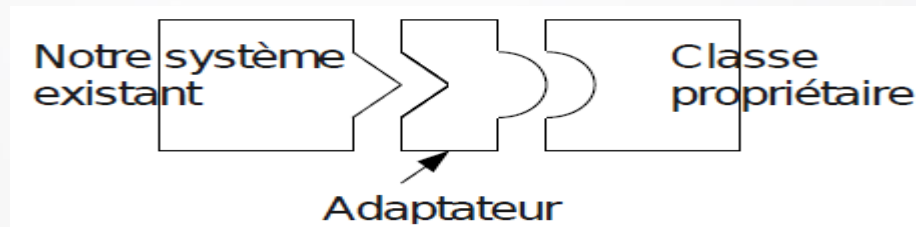




Exemple 1 de patron de conception de structure

- **Adapter :**

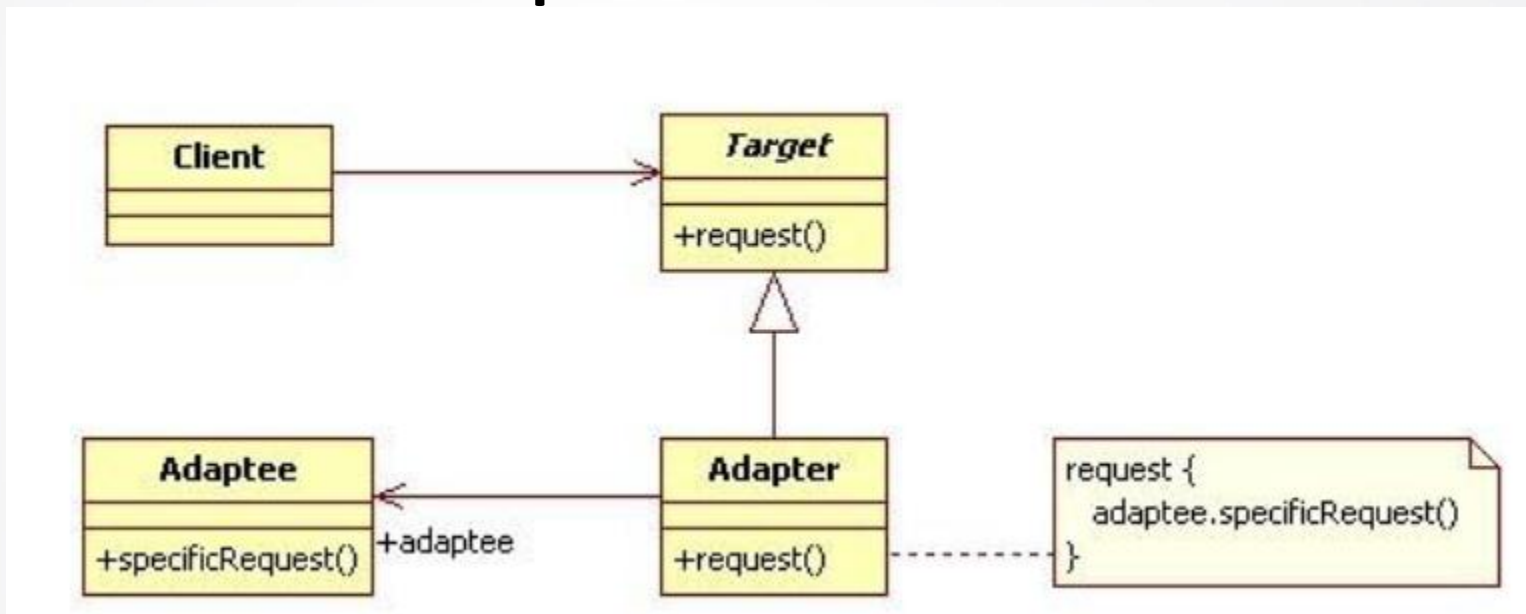
- Permet de faire le pont entre deux interfaces incompatibles.
- Jointure des fonctionnalités de différentes interfaces (indépendantes ou incompatibles) à travers une seule classe.
- Encapsuler un ensemble de fonctions sous la même implémentation (wrapping/emballage).





Exemple 1 de patron de conception de structure

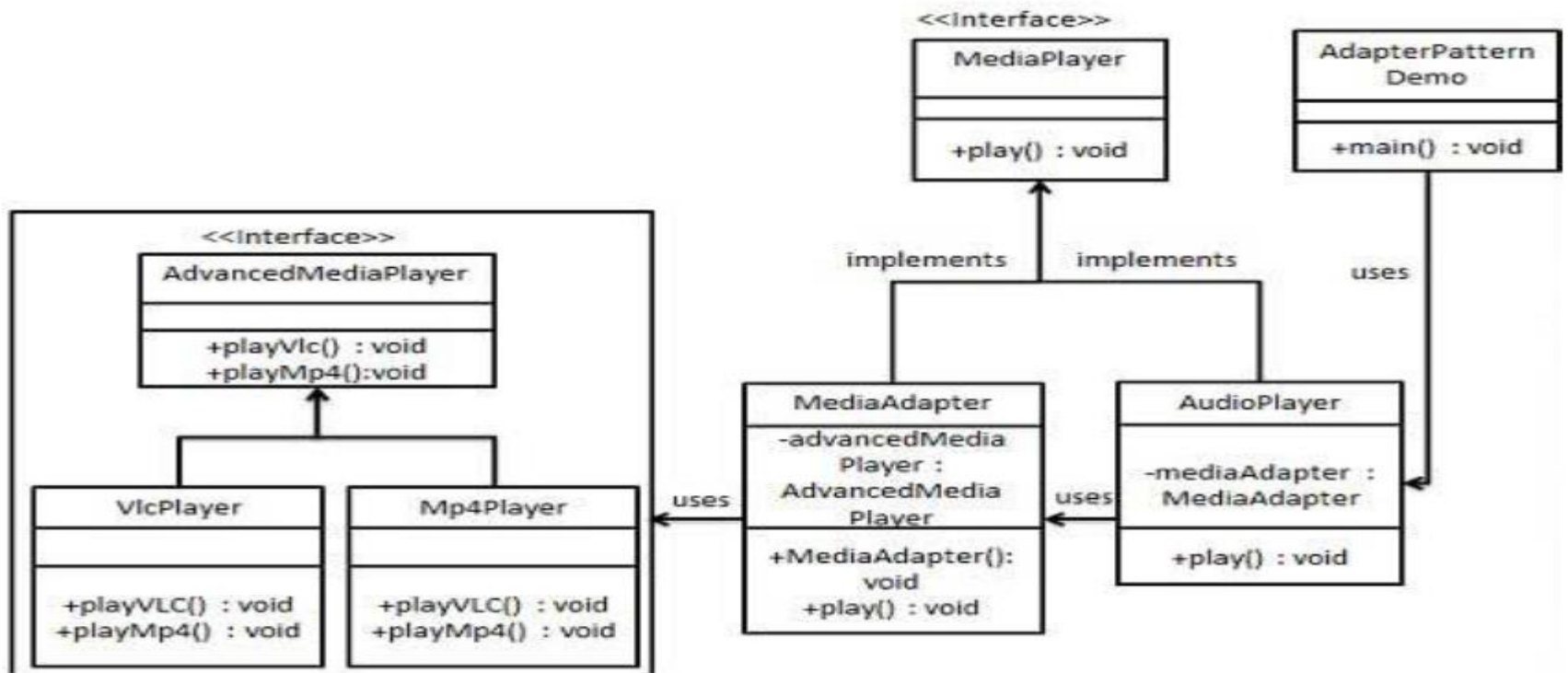
- **Structure de Adapter :**





Exemple 1 de patron de conception de structure

- Exemple avec Adapter :

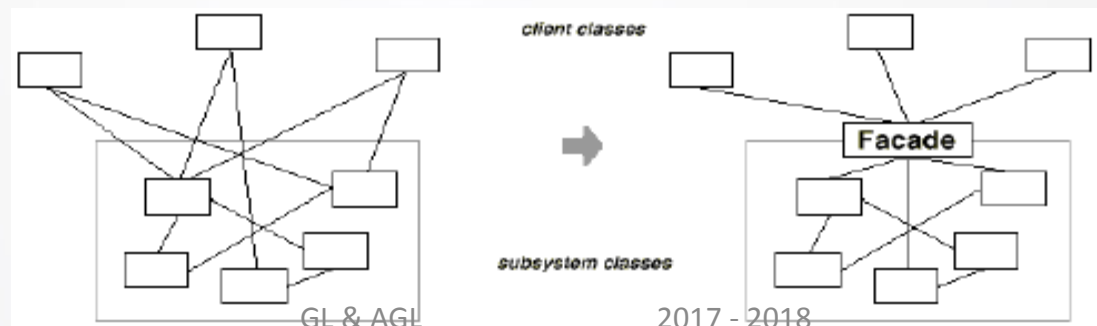




Exemple 2 de patron de conception de structure

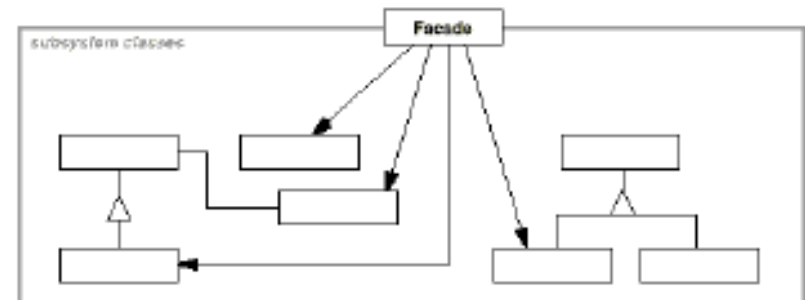
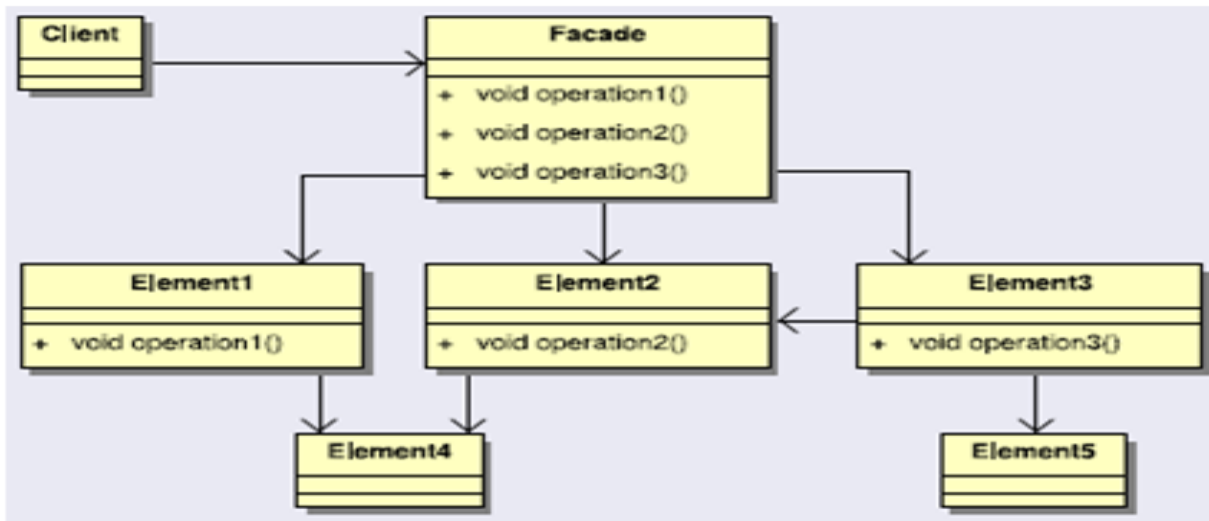
- **Façade :**

- 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

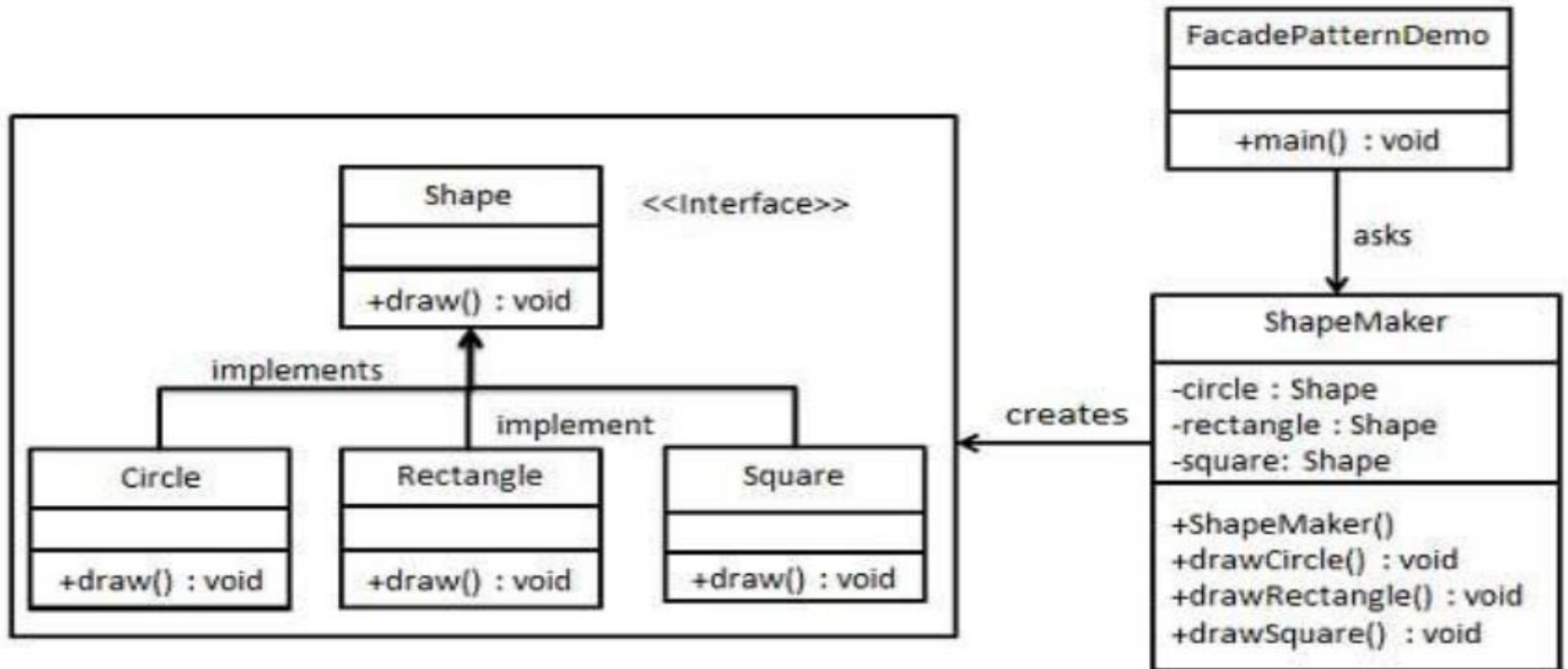
- **Structure de Façade :**





Exemple 2 de patron de conception de structure

- Exemple avec Façade :





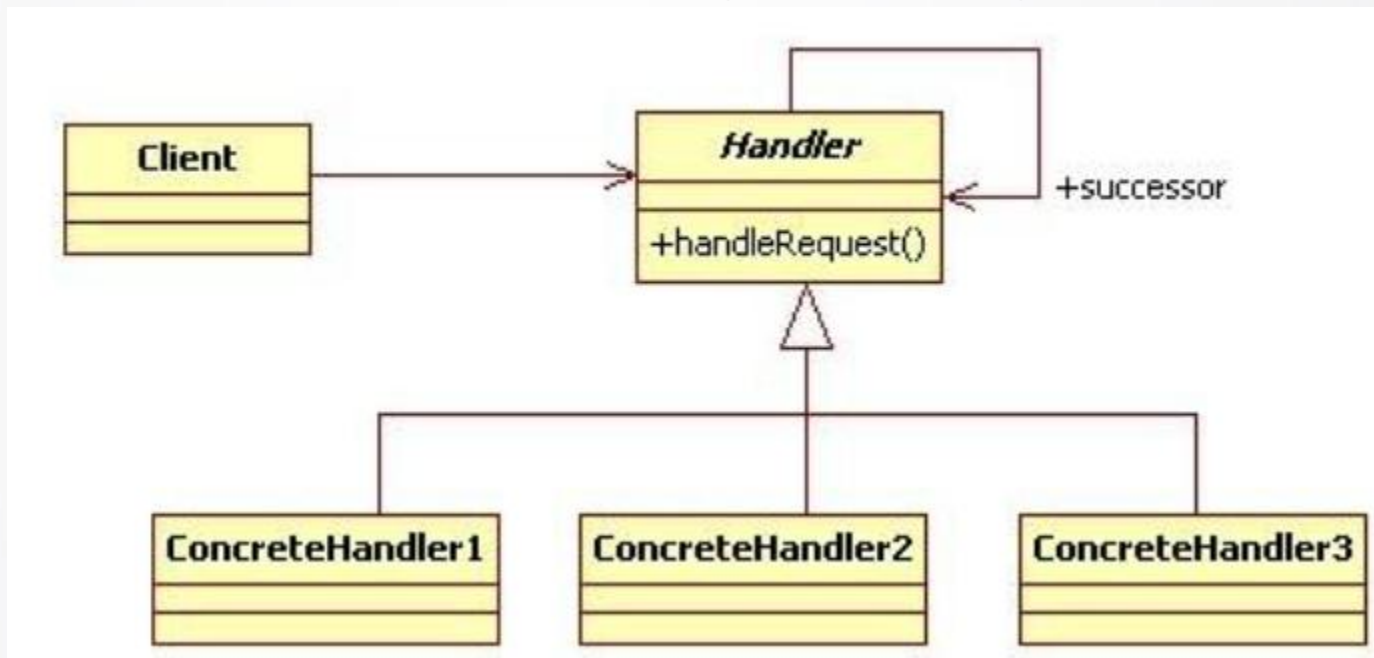
Exemple 1 de patron de conception de comportement

- **Chain of Responsibility :**

- Créer une chaine d'objets receveurs pour une requête donnée.
- Chaque receveur contient une référence à un autre receveur.
- Si un objet ne peut pas traiter une requête, il la fait passer au receveur suivant et ainsi de suite.

Exemple 1 de patron de conception de comportement

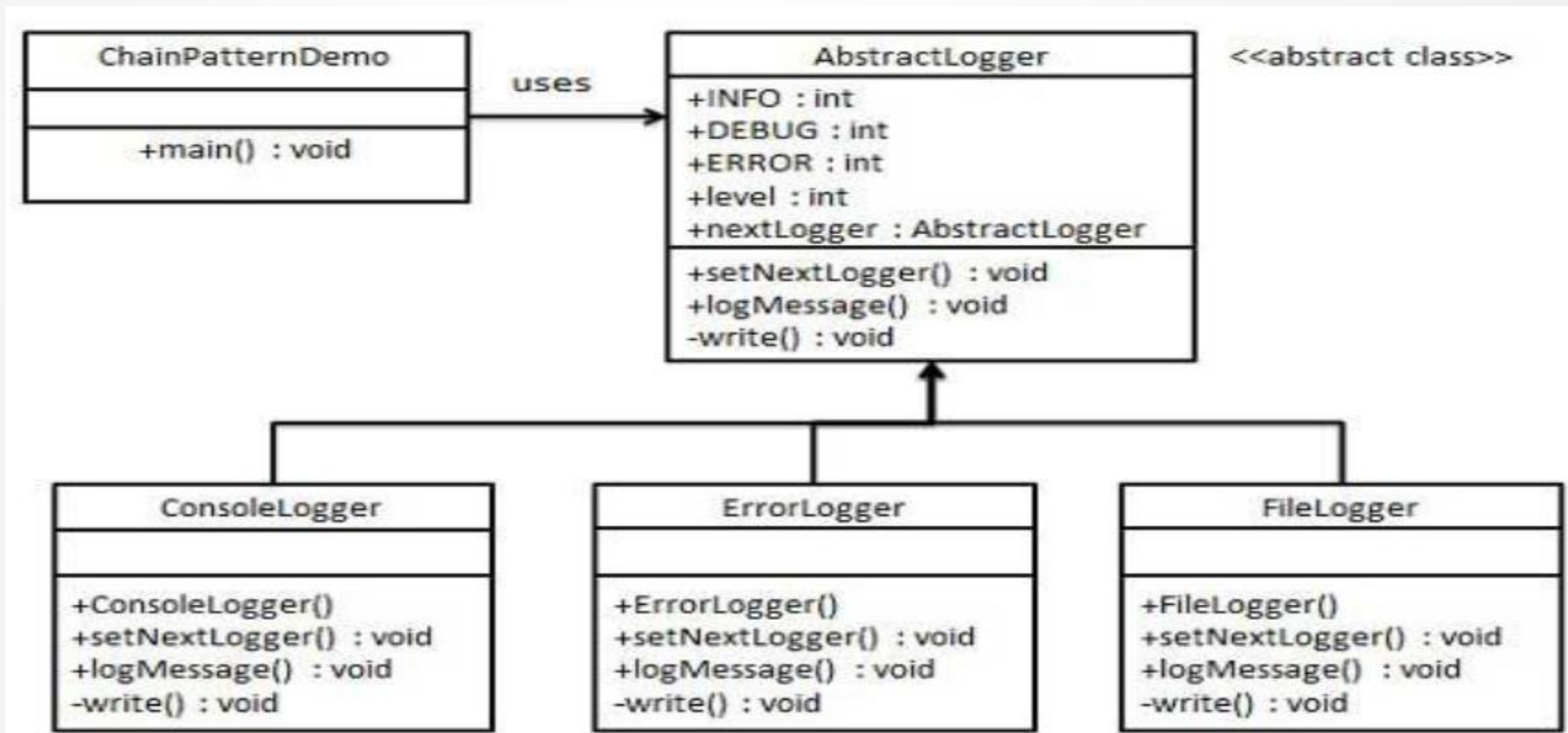
- **Structure de Chain of Responsibility :**





Exemple 1 de patron de conception de comportement

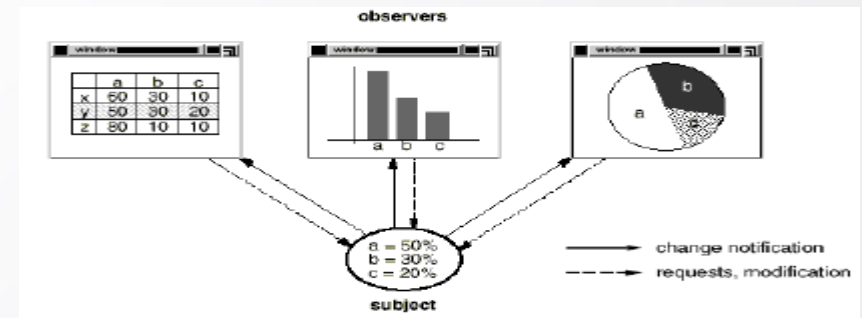
- **Exemple avec Chain of Responsibility :**



Exemple 2 de patron de conception de comportement

- **Observer :**

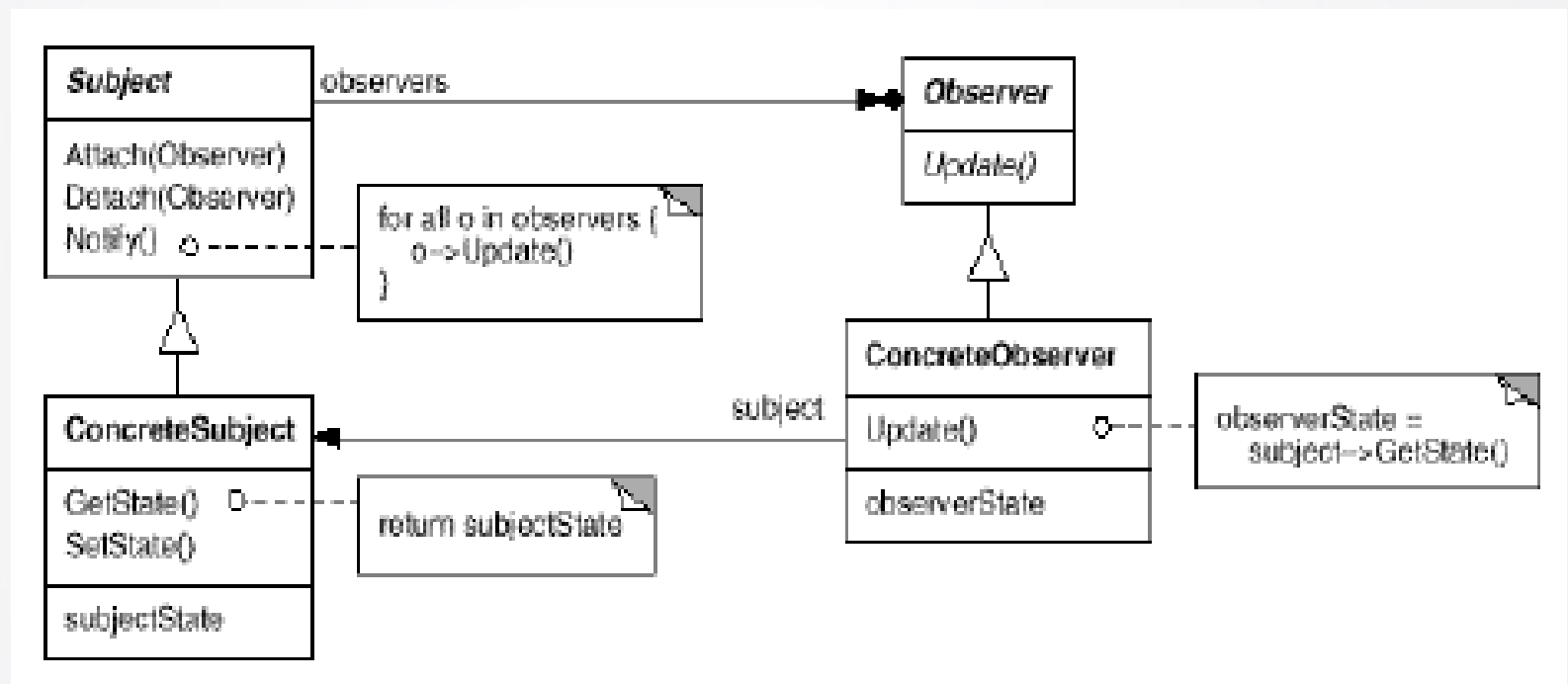
- Définit une dépendance « one-to-many » (un à plusieurs) entre objets de sorte que lorsque l'état d'un objet change, tous ses objets dépendants sont **notifiés et mis à jours** automatiquement.
- L'objet observé (Observable) gère une liste d'observateurs (Observer) dotés d'une méthode de mise à jour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





Exemple 2 de patron de conception de comportement

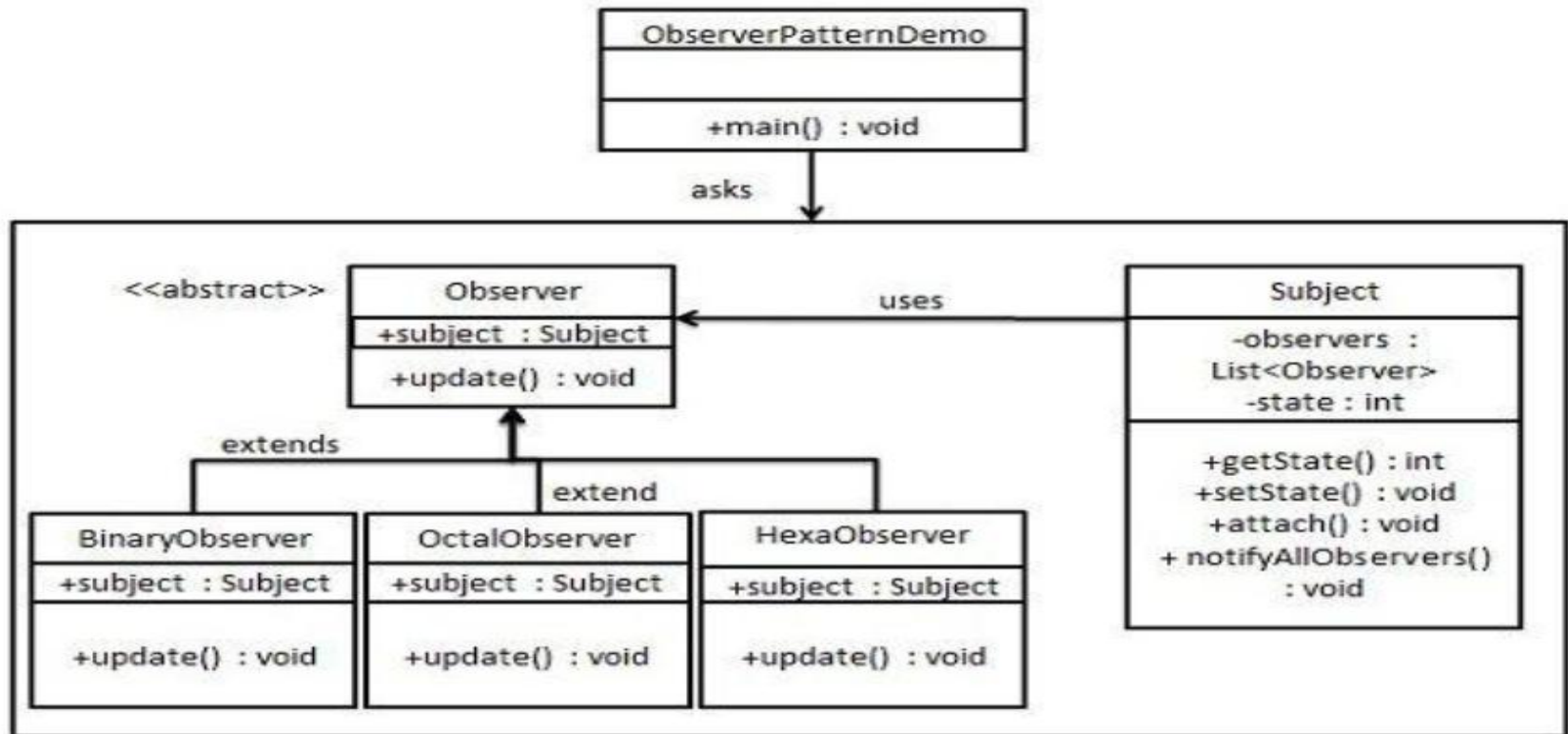
- **Structure d'Observer :**





Exemple 2 de patron de conception de comportement

- Exemple avec Observer :



Exemple : Architecture logique en 5 couches/ frameworks et patrons

