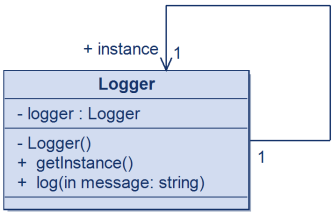
Fiche Design patterns

Design patterns créationnels

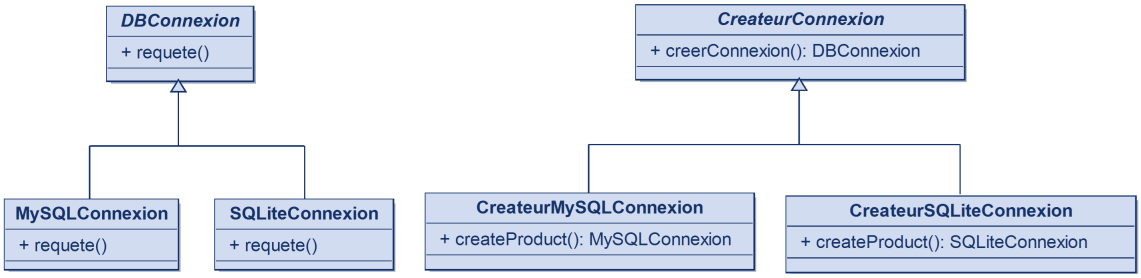
Singleton

Objectif : Garantir qu'une classe ne possède qu'une seule instance et fournir un point d'accès global à celle-ci.
Structure : Un constructeur privé empêchant la création d'instances en dehors de la classe, une instance statique stockée dans la classe, une méthode publique pour l'accès à l'instance unique.
Exemple : Une classe `Logger`[1], qui permet d'écrire dans un fichier les logs du système. Une seule instance ne doit être créée pour éviter les problèmes de concurrence (écriture dans un même fichier par différentes instances).



Factory method

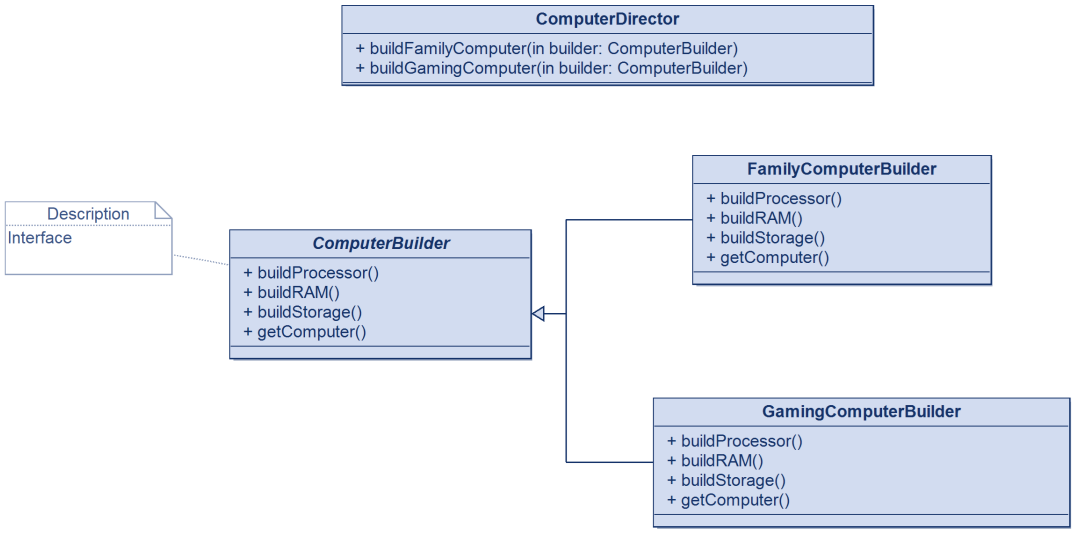
Objectif : Déléguer l'instanciation d'un objet à une sous-classe afin de ne pas spécifier la classe exacte de l'objet à créer, cela permet plus de modularité dans le code.
Structure : Une classe mère *factory* définit une méthode de fabrication qui est redéfinie par les sous-classes pour créer des objets spécifiques. Quant aux objects spécifiques, ils implémentent (ou héritent) une interface (ou d'une classe mère), qui correspond au type de retour de la classe *factory*.
Exemple : Un logiciel gère des bases de données de différents types. La connexion avec une base de données est déléguée à une *factory* (ici `CreateurConnexion`).



1. Dans l'exemple de diagramme UML pour la classe `Logger`, l'attribut `logger` ne devrait pas être précisé puisqu'il est de type `Logger`, il l'est cependant pour notifier le fait qu'il est privé

Builder

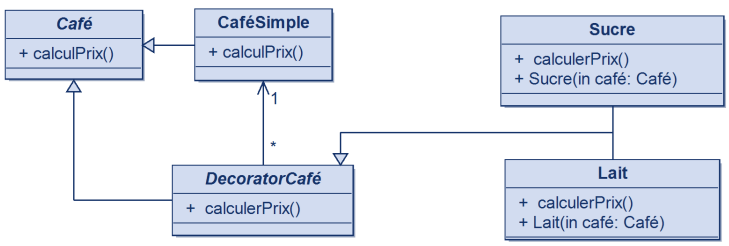
Objectif : Construire un objet complexe étape par étape pour améliorer la lisibilité et la maintenance du code.
Structure : Une interface *builder* fournit des méthodes pour configurer les différents aspects de l'objet et une méthode finale pour le récupérer, les classes des constructeurs des différents objets implémentent cette interface. Une classe *director* ordonnance les appels des méthodes des *builders*.
Exemple : Sur un site, une commande pour un ordinateur se résume au choix du processeur, de la RAM et du disque dur. Différents types d'ordinateurs sont alors référencés (gaming, familial, etc.) Le gestionnaire de commande crée l'objet de la commande via un *builder*. Pour cela, il créé un *director* qui gère la fabrication, puis un *builder* qui définie chacune des étapes. Puis l'objet est récupéré par la fonction *get* du *builder*.



Design patterns structurels

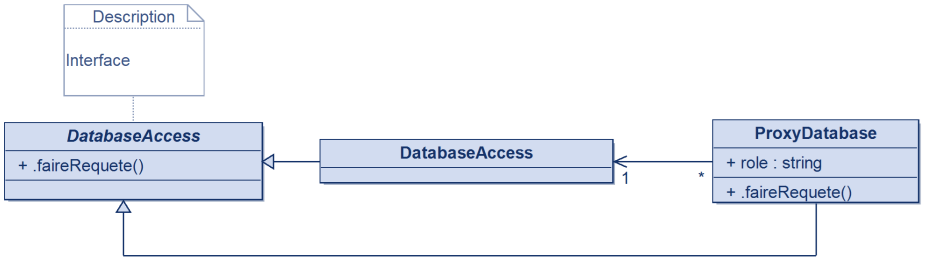
Decorator

Objectif : Ajouter dynamiquement des fonctionnalités à un objet sans modifier son code source.
Structure : Une classe *decorator* implémente la même interface que l'objet décoré et contient une instance de cet objet. Une nouvelle fonctionnalité est une nouvelle classe.
Exemple : Dans un distributeur de café, le café peut être amélioré en ajoutant du sucre, du lait, etc. Chaque ajout possible correspond à une classe qui va 'décorer' le café passé en argument (possiblement déjà décoré).



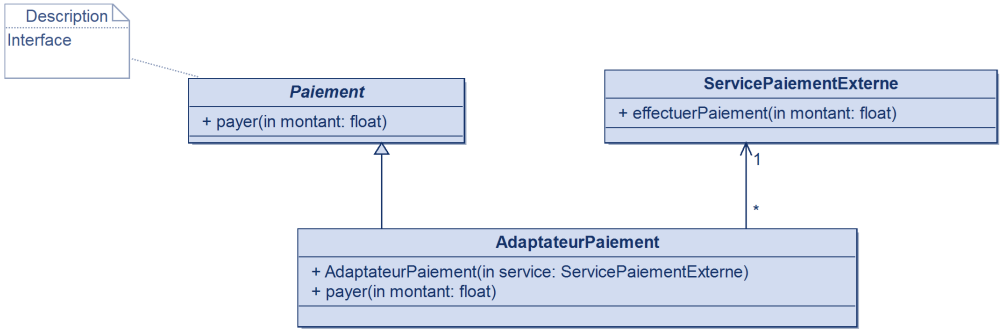
Proxy

Objectif : Fournir un substitut ou un intermédiaire contrôlant l'accès à un objet.
Structure : Une classe proxy implémente la même interface que l'objet cible et contrôle son accès en ajoutant une couche intermédiaire.
Exemple : Une base de données accessible seulement à certains membres, le *proxy* permet donc de faire l'étape de vérification avant d'accéder au résultat de la requête.



Adapter

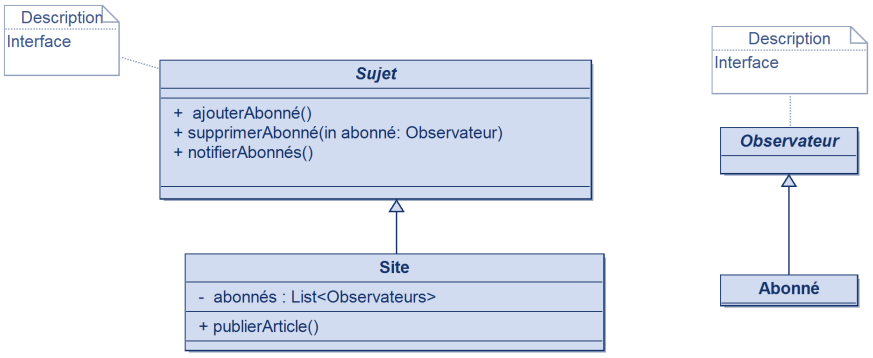
Objectif : Créer une interface entre deux entitées incompatibles.
Structure : Une classe *adapter* implémente l'interface cible et contient une instance de la classe existante qu'elle adapte. Les actions définies par l'interface cible sont alors redéfinies dans l'*adapter* via l'objet qu'elle adapte.
Exemple : Dans une application, un nouveau mode de paiement veut être ajouté, mais l'API de ce service n'est pas compatible avec la structure du paiement actuel. L'*adapter* permet de faire passerelle entre l'application et l'API.



Design patterns comportementaux

Observer

Objectif : Permettre à un objet (sujet) de notifier automatiquement plusieurs des observateurs lorsqu'un état change.
Structure : Un *publisher* (sujet) maintient une liste d'abonnés et fournit des méthodes pour s'inscrire, se désinscrire et notifier les observateurs.
Exemple : Un site d'actualités envoie une notification à chaque abonné lors de la publication d'un nouvel article.



Strategy

Objectif : Permettre de sélectionner dynamiquement un algorithme parmi plusieurs implémentations sans modifier le contexte d'utilisation.

Structure : Une interface commune est implémentée par différentes stratégies. Le contexte contient une référence vers une stratégie et l'appelle dynamiquement.

Exemple : Dans un jeu, le joueur peut jouer contre une IA qui peut avoir différent niveau de difficulté.