# C:\Users\D-Day\Pictures\classe.pngDiagramme de classe

## Le modèle en couche

Nous avons implémenté un modèle en couche pour permettre de mettre en application les concepts de l’ingénierie objet.

Nous avons donc décidé de séparer notre application en trois couches:

* l’interface graphique (GUI)
* la Business Logic (BL)
* la persistance des données (Persist)

### GUI

L’interface graphique rassemble toutes les classes qui constituent l’interface utilisateur, c’est-à-dire la fenêtre d’authentification, le menu, la fenêtre de consultation de son emploi du temps, la fenêtre de demande de réservation ainsi que la fenêtre de traitement des réservations.

Ici les classes de la couche GUI sont représentées de façon simplifiée, on considère que chaque classe représente une fenêtre, celle-ci écoute ses propres événements, on ne détaille pas la composition des fenêtres (par exemple les panneaux, les boutons, etc.). Lors du développement, la conception de ses fenêtres sera adaptée au langage, à la librairie utilisée et correspondra aux maquettes fournies dans le rapport des uses cases.

### BL

La couche Business Logic contient les classes métiers, c’est-à-dire la logique interne de notre application. Elle comprend les enseignants, les réservations, les créneaux, etc. Ainsi que les relations entre tous ses objets qui représentent le cœur de notre application.

### Persistance

La couche de persistance quant à elle permet « le dialogue » avec les données de la BL qui ont été sauvegardées. Cette couche permet de récupérer les données sauvegardées puis on les modifie dans la business logic et enfin on les réenregistre grâce à cette couche. La persistance peut être assurée sur différents supports par exemple sur des fichiers ou sur une base de données. Ici nous allons concevoir une persistance des données dans une base Oracle.

## Séparation UI/BL

Pour permettre la mise en place d’une interface très indépendante de notre cœur métier dans le but d’augmenter la réutilisabilité de notre code nous avons utilisé le **design pattern « Façade »**.

Nous avons mis en place deux façades, une pour chaque « vue » de l’application. En effet, nous avons identifié deux types de demandes à notre Business Logic de la part de la GUI :

* Les demandes qui concernent directement un enseignant donc les demandes en lien avec la personne qui est authentifiée.
* Les demandes qui concernent le traitement de toutes les réservations de façon générale sans lien avec l’utilisateur (uniquement l’administrateur peut effectuer celles-ci).

Nous avons donc deux façades, « TeacherFacade » pour les demandes relatives à l’enseignant et « HandlingFacade » pour les demandes relatives au traitement des réservations.

Toutes les communications entre les deux couches passent par ses façades. Celles-ci sont chargées de transformer les objets réels de la couche métier en chaînes de caractères pouvant être affichées par l’interface, en sens inverse elles doivent pouvoir transformer les messages de la GUI en objet de la couche métier.

Ainsi l’interface utilisateur ne connaît aucun objet de la couche métier, elle est donc totalement indépendante de celle-ci et se contente uniquement d’afficher les informations qui lui sont envoyées et d’envoyer les informations que l’utilisateur sélectionne aux façades de la BL.

Les façades manipulent les objets métiers de manière à répondre aux demandes d’informations de la couche d’interface utilisateur.

Cette séparation pourrait permettre entre autres de créer une nouvelle interface de type console sans toucher à la couche métier où bien de modifier cette couche sans incidence sur l’interface utilisateur.

## Séparation BL/Persistance

La séparation de ces couches permet de définir le cœur de notre application sans se soucier de la façon dont les données sont enregistrées. En effet, on peut ainsi permettre le changement de type de persistance en changeant seulement une couche de l’application.

Pour cela, on utilise le **design pattern « Factory »** qui permet à la couche business logic de manipuler les objets de la couche persistance sans s’en rendre compte, et ainsi de pouvoir créer une couche métier indépendante et définir la couche persistance réelle au lancement de l’application.

Pour réaliser l’indépendance, toutes les classes de la couche métier sont abstraites.

Pour pouvoir manipuler des objets avec ces types abstraits on utilise une fabrique abstraite (« PersistFactory ») qui est un singleton (il en existe donc qu’une seule dans toute l’application). Celle-ci est créée de manière concrète (« PersistFactoryJDBC ») au démarrage de l’application, elle comporte des fonctions qui permettent de récupérer chacune des classes abstraites sous forme concrète sans connaître réellement la classe réelle. Par la suite, la couche métier peut donc s’en servir pour manipuler des objets concrets sans le savoir, en effet, elle ne connaît que les objets abstraits.

Les classes concrètes héritent des classes abstraites et contiennent toutes les fonctions qui comportent un accès à la base de données.

Ceci peut permettre entre autre de proposer par exemple à l’utilisateur de sauvegarder ses données dans dans des fichiers à la place d’une base de données. Pour cela, il suffit juste de créer une fabrique concrète « PersistFactorySerializable » ainsi que toutes les classes concrètes associées puis de créer ce type de fabrique au lancement de l’application.