Camille Raymond – Christophe Gire
Damien Sendner – Thibaut Rouquette

Conception du projet URM

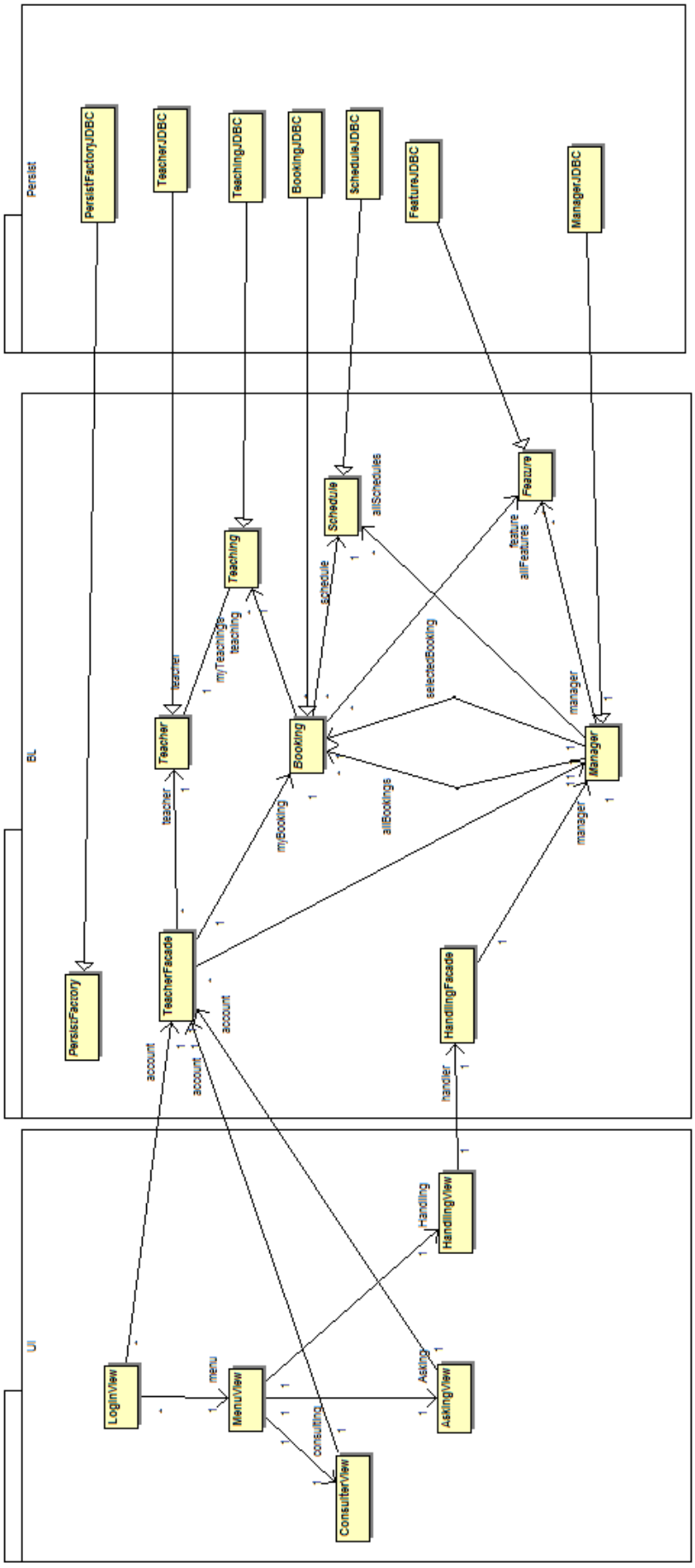
University Resources Management

Résumé et explication du diagramme de classe réalisé pour la conception du logiciel de gestion de réservation des salles et de la consultation de planning : URM.

05/03/2012

Contenu

I. Diagramme de classe



1) 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)

a) 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.

b) 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.

c) 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.

2) 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.

3) 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 une base Mysql à la place d'Oracle. Pour cela, il suffit juste de créer une fabrique concrète « PersistFactoryMySQL » ainsi que toutes les classes concrètes associées puis de créer ce type de fabrique au lancement de l'application.

II. Interface Graphique

Nous allons dans cette partie décrire plus en détails les classes de la couche Graphical User Interface.

LoginView

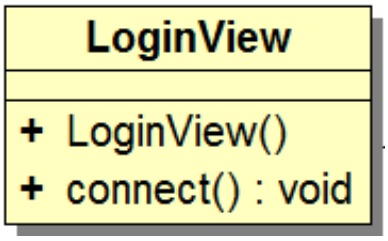


Figure 1 - Classe LoginView

Brève description

Cette fenêtre graphique permet à l'utilisateur de s'authentifier dans le système en renseignant son identifiant et son mot de passe.

Description des attributs

account

: attribut de type TeacherFacade qui correspond à l'enseignant qui s'authentifie.

menu

: attribut de type MenuView qui correspond au menu qui sera généré en fonction des droits de l'utilisateur.

Description des méthodes

LoginView()

: constructeur public qui permet d'initialiser la fenêtre graphique pour l'authentification.

connect()

: méthode publique permettant de vérifier la concordance et l'existence d'un identifiant et d'un mot de passe et de se connecter au système, le cas échéant. Joue le rôle d'un écouteur. Ne retourne rien et se produit lorsque l'utilisateur clique sur le bouton « valider ». Elle sera implémenté en Java dans la méthode actionPerformed().

MenuView

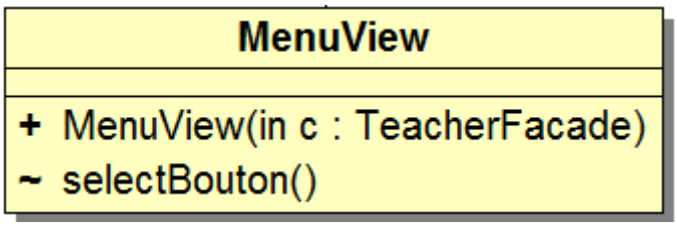


Figure 2 – MenuView

Brève description

Cette fenêtre graphique permet à l'utilisateur de sélectionner la fonctionnalité du logiciel qui l'intéresse : faire une demande de réservation, consulter son planning ou traiter une demande de réservation.

Description des attributs

consulting

: attribut de type ConsulterView qui correspond à la fenêtre de consultation du planning de l'enseignant authentifié.

asking

: attribut de type AskingView qui correspond à la fenêtre de demande de réservation de salle de l'enseignant authentifié.

handling

: attribut de type HandlingView qui correspond à la fenêtre traitement des demandes de réservation de l'enseignant authentifié aux droits supérieurs.

Description des méthodes

MenuView(c : TeacherFacade)

: constructeur publique permettant d'initialiser la fenêtre graphique correspondant au menu de l'utilisateur. En paramètre : c un objet de type *TeacherFacade* qui fait référence à un enseignant.

selectBouton()

: méthode a visibilité par défaut (paquetage). Ecoute les différents événements sur les boutons. Ne retourne rien. Lors de l'implémentation en Java, il s'agira de définir la méthode actionPerformed().

ConsulterView

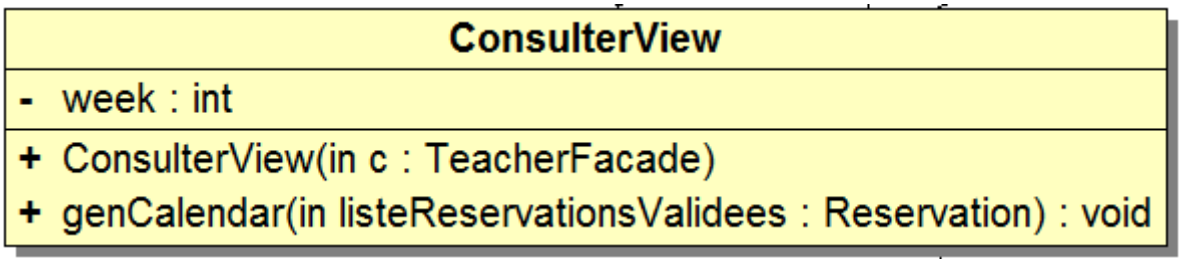


Figure 3 – ConsulterView

Brève description

Cette fenêtre graphique permet à l'utilisateur de consulter son emploi du temps.

Description des attributs

week : int

: attribut privé. Un entier qui représente le numéro de la semaine.

account

: attribut de type TeacherFacade qui correspond à l'enseignant authentifié.

Description des méthodes

ConsulterView(c : TeacherFacade)

: constructeur publique permettant d'initialiser la fenêtre graphique correspondante à la consultation de son planning par un enseignant. En paramètre : c un objet de type *TeacherFacade* qui fait référence à un enseignant.

genCalendar(Reservation listeReservationsValidees)
: méthode publique qui permet de générer un calendrier à partir d'une liste des réservations validées de l'enseignant. Cette méthode ne retourne rien.

AskingView

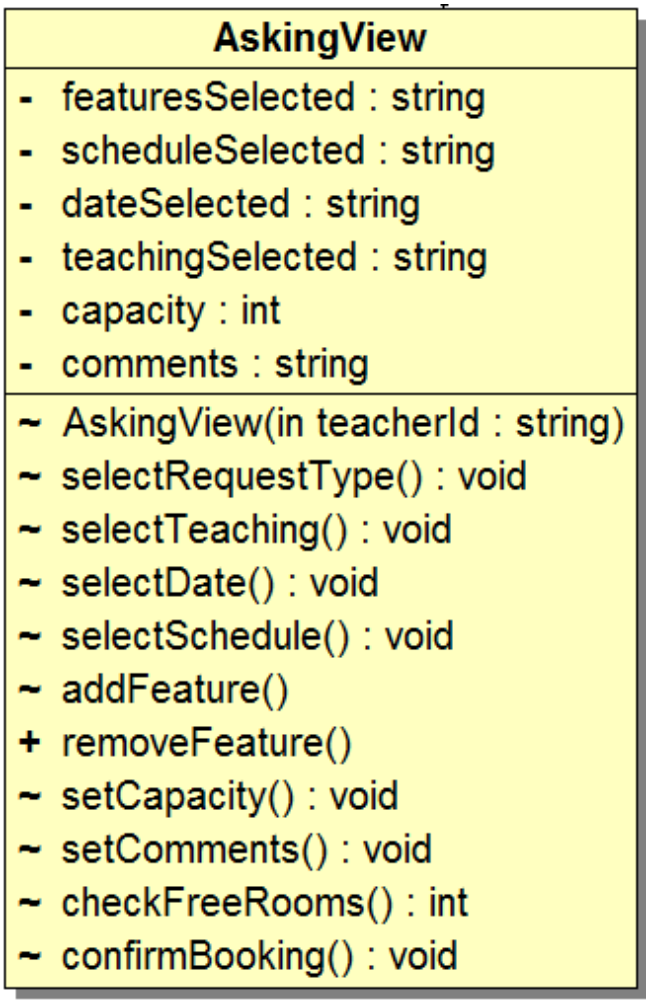


Figure 4 – AskingView

Brève description
Cette fenêtre graphique permet à l'utilisateur de réaliser une demande de réservation de salle pour une réunion ou un enseignement.

Description des attributs
featureSelected
: attribut privé de type chaine de caractères. Il correspond à l'ensemble des caractéristiques d'une salle sélectionné par l'utilisateur.

scheduleSelected
: attribut privé de type chaine de caractères. Il correspond au créneau horaire sélectionné par l'utilisateur.

dateSelected
: attribut privé de type chaine de caractères. Il correspond à la date sélectionnée par l'utilisateur.

teachingSelected
: attribut privé de type chaine de caractères. Il correspond à l'enseignement sélectionné par l'utilisateur.

capacity
: attribut privé de type entier. Il correspond à la capacité maximale souhaitée de la salle pour la réservation.

comments
: attribut privé de type chaine de caractères. Il correspond aux commentaires laissés par l'utilisateur lors de sa demande de réservation à l'attention du responsable emploi du temps.

Account
: attribut de type TeacherFacade qui correspond à l'enseignant authentifié.

Description des méthodes

AskingView(teacherId : String)
: constructeur visible dans le paquetage permettant d'initialiser la fenêtre graphique correspondante à la demande de réservation d'une salle. Il prend en paramètre une chaine de caractère correspondante à l'identifiant de l'enseignant utilisateur.

selectRequestType()
: nom abstrait de la méthode qui écoute les événements de sélection d'un radio-bouton. Elle est appelée lorsque l'utilisateur sélectionne le type de demande de réservation à effectuer (enseignement ou réunion).

selectTeaching()
: nom abstrait de la méthode qui écoute les événements de sélection dans une liste. Elle est appelée lorsque l'utilisateur sélectionne l'enseignement pour lequel il souhaite réserver une salle.

selectDate()
: nom abstrait de la méthode qui écoute les événements de sélection dans un calendrier. Elle est appelée lorsque l'utilisateur sélectionne une date dans le calendrier de la fenêtre de demande de réservation.

selectSchedule()
: nom abstrait de la méthode qui écoute les événements de sélection dans une liste. Elle est appelée lorsque l'utilisateur sélectionne un créneau horaire dans une liste.

addFeature()
: nom abstrait de la méthode qui écoute les événements liés au bouton d'ajout des caractéristiques. Elle est appelée lorsque l'utilisateur clique sur le bouton représentant une flèche vers la droite. Elle sera traitée lors de l'implémentation Java par actionPerformed().

removeFeature()
: nom abstrait de la méthode qui écoute les événements liés au bouton de suppression des caractéristiques. Elle est appelée lorsque l'utilisateur clique sur le bouton représentant une flèche vers la gauche. Elle sera traitée lors de l'implémentation Java par actionPerformed().

setCapacity()
: nom abstrait de la méthode qui écoute les événements liés à l'écriture d'informations dans le champ de texte destiné à la capacité maximale de la salle désirée par le demandeur.

setComments()
: nom abstrait de la méthode qui écoute les événements liés à l'écriture d'informations dans le champ de texte destiné aux commentaires.

checkFreeRooms()
: méthode visible au niveau du paquetage. Elle retourne le nombre de salles disponibles en fonction de la date, du créneau, de la capacité maximale et des caractéristiques sélectionnés par l'utilisateur.

confirmBooking()
: nom abstrait de la méthode qui écoute les événements liés à l'appui du bouton valider. Elle permet d'enregistrer la demande de réservation dans le système. Elle sera implémentée en Java à travers la méthode actionPerformed().

HandlingView

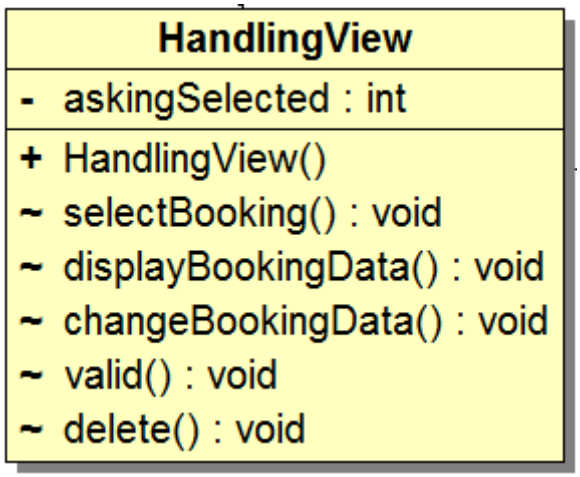


Figure 5 – HandlingView

Brève description
Cette fenêtre graphique permet à l'utilisateur ayant les droits requis, de traiter les demandes de réservations effectuées par les enseignants.

Description des attributs
askingSelected
: attribut entier privé. Il correspond au numéro de la demande de réservation sélectionnée.

handler
: attribut de type TeacherFacade qui représente l'enseignant qui a les droits nécessaire pour le traitement des demandes de réservations.

Description des méthodes
HandlingView()
: constructeur publique qui permet l'initialisation de la fenêtre graphique dédiée au traitement des demandes de réservations de salle.

selectBooking()
: nom abstrait de la méthode qui écoute les événements de sélection dans une liste. Elle est appelée lorsque l'utilisateur sélectionne une demande de réservation.

displayBookingData()
: méthode visible dans le paquetage. Elle permet d'afficher les informations relatives à une demande de réservation sélectionnée par le responsable.

changeBookingData()
: méthode visible dans le paquetage. Elle permet de modifier les informations relatives à une demande de réservation sélectionnée par le responsable.

valid()
: nom abstrait de la méthode qui écoute les événements liés à l'appui du bouton valider. Elle permet d'enregistrer la réservation dans le système. Elle sera implémentée en Java à travers la méthode actionPerformed().

delete()
: nom abstrait de la méthode qui écoute les événements liés à l'appui du bouton supprimer. Elle permet de supprimer la demande de réservation dans le système. Elle sera implémentée en Java à travers la méthode actionPerformed().

III. Business Logic

Cette couche représente la logique métier de l'application. Comme expliqué dans la première partie, trois Façades font le lien avec l'ensemble des autres classe de la logique métier : TeacherFacade pour la consultation de son planning et la demande de réservation et HandlingFacade pour le traitement des demandes de réservation.

TeacherFaçade

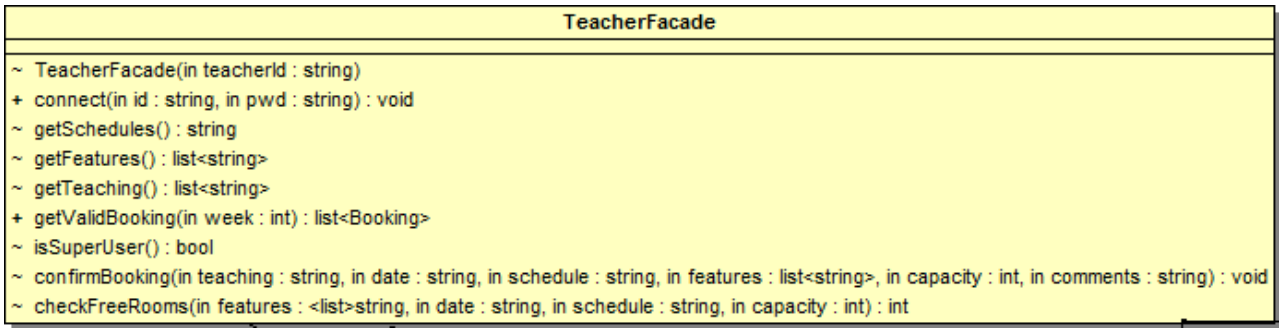


Figure 6 - TeacherFaçade

Brève description
La classe TeacherFacade est une façade pour les demandes relatives au traitement des réservations. Celle-ci se charge de transformer les objets réels de la couche métier en chaînes de caractères pouvant être affichées par l'interface, et en sens inverse, elles doivent pouvoir transformer les messages de la GUI en objet de la couche métier.

Description des attributs
Elle possède trois attributs référençant un enseignant, une réservation et un manager.

Description des méthodes
TeacherFacade(String teacherId)
: constructeur de la classe avec comme paramètre l'identifiant de l'enseignant.

connect(String id, String pwd) : void
: connecte l'utilisateur si id/pwd est valide.

getSchedules() : String
: retourne la liste des créneaux disponibles sous la forme d'une liste de String.

getFeatures() : list<String>
: retourne la liste des caractéristiques disponibles dans une liste de String.

getTeaching() : list<String>
: retourne la liste des enseignements de l'utilisateur (l'enseignant qui utilise l'application), dans une liste de String.

getValidBooking(int week) : list<Booking>
: renvoie la liste des Réservations faites par l'enseignant qui sont validées.

isSuperUser() : bool

: retourne vrai si l'utilisateur est le gestionnaire de l'emploi du temps, faux sinon.

confirmBooking(String teaching, String date, String schedule, list<String features, int capacity, String comments) : void
: valide définitivement la réservation, c'est à dire sauvegarde les données de celle-ci.

checkFreeRooms(list<String features, String date, String schedule, int capacity) : int
: retourne le nombre de salles disponibles avec les caractéristiques, la date, le créneau et la capacité choisie.

Handling Facade

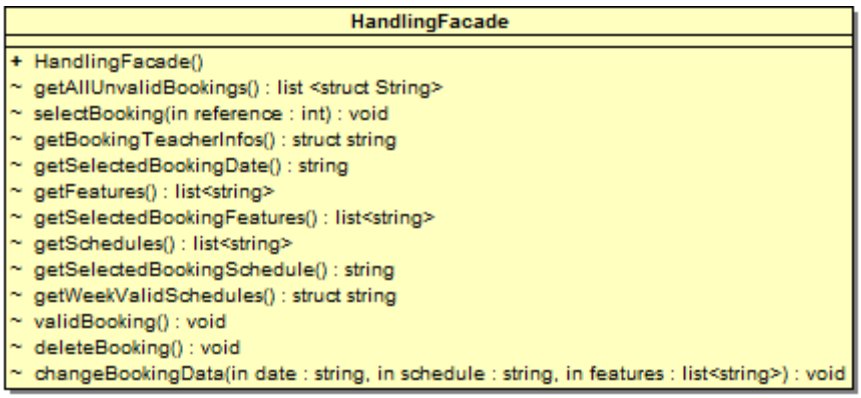


Figure 7 - Handling Facade

Brève description
La classe HandlingFacade est une façade pour les demandes relatives à l'enseignant. Celle-ci se charge de transformer les objets réels de la couche métier en chaînes de caractères pouvant être affichées par l'interface, et en sens inverse, elles doivent pouvoir transformer les messages de la GUI en objet de la couche métier.

Description des attributs
Elle possède un attribut référençant un manager.

Description des méthodes
getAllUnvalidBookings() : list<struct String>
: renvoie la liste de toutes les demandes de réservations des enseignants qui ne sont pas validées (pour lesquelles aucune salle n'a été attribuée).

selectBooking(int reference) : void
: sélectionne la réservation avec la référence donnée en paramètre.

getBookingTeacherInfos() : struct String
: retourne sous la forme d'une structure composée de strings toutes les informations sur l'enseignant, c'est à dire nom, prénom, téléphone, mail et enseignement.

getSelectedBookingDate() : String
: retourne la date de la réservation sélectionnée.

getFeatures() : list<String>
: retourne la liste de toutes les caractéristiques disponibles.

getSelectedBookingFeatures() : list<String>
: retourne la liste des caractéristiques de la réservation sélectionnée.

getSchedules(): list<String>
: retourne une liste de tous les créneaux disponibles.

getSelectedBookingSchedule() : String
: retourne le créneau de la réservation sélectionnée.

getWeekValidSchedules() : struct String
: retourne tous les créneaux pour toutes les dates de la semaine pour lesquels il y a des salles de disponibles avec cette réservation.

validBooking() : void
: sauvegarde la réservation et lui associe une salle.

deleteBooking() : void
: supprime la réservation.

changeBooking(String date, String schedule, list<String> features) : void
: remplace les données dans la réservation sélectionnée.

PersistFactory

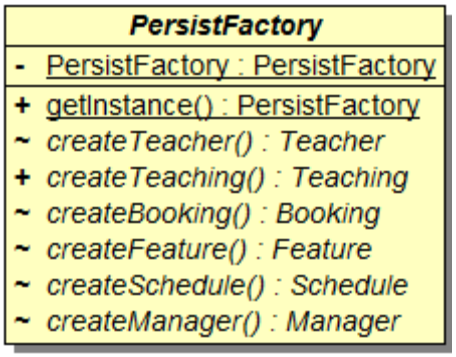


Figure 8 - PersistFactory

Brève description
La classe PersistFactory représente la fabrique abstraite. Elle fournit une interface permettant de créer différents produits (Enseignant, Enseignement, Réservation, Caractéristique, Créneau et Manager).

Description des attributs
- PersistFactory : PersistFactory
: représente la seule instance de PersistFactory (singleton).

Description des méthodes
createTeacher() : Teacher
: renvoie un objet de type enseignant.

createTeaching() : Teaching
: renvoie un objet de type enseignement.

createBooking() : Booking
: renvoie un objet de type réservation.

createFeature() : Feature
: renvoie un objet de type caractéristique.

createSchedule() : Schedule
: renvoie un objet de type créneau horaire.

createManager() : Manager
: renvoie un objet de type gestionnaire.

Manager

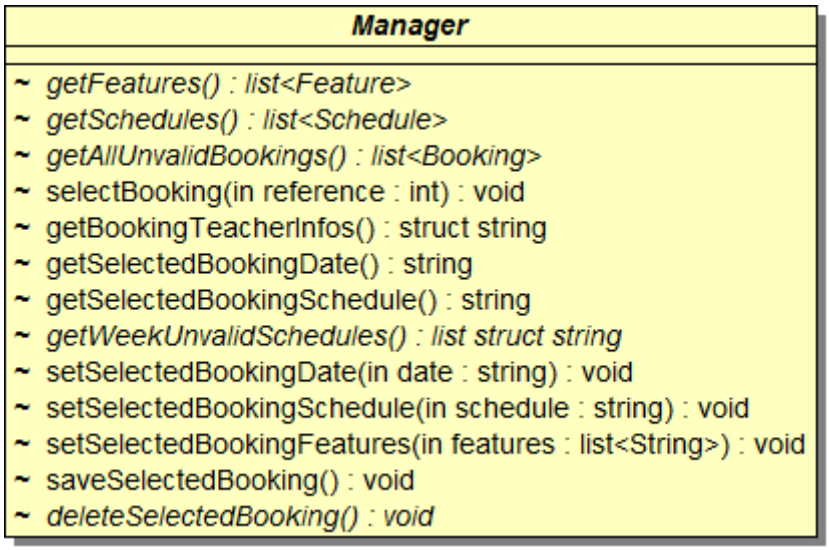


Figure 9 - Manager

Brève description
La classe Manager est le gestionnaire de l'application. Elle se charge de récupérer les données ensemblistes de la base afin d'y effectuer certains traitements. En effet, nous avons regroupé les données que l'on veut récupérer en deux types, les données liées à l'enseignant et les données générales. Pour récupérer des données générales on utilise cette classe.

Description des attributs
Elle possède deux attributs référençant une liste de caractéristiques et une liste de créneaux.

Description des méthodes
getFeatures() : list<Feature>
: retourne une liste de toutes les caractéristiques existantes.

getSchedules() : list<Schedule>
: retourne une liste de tous les créneaux existants.

getAllUnvalidBookings(): list<Booking>
: retourne la liste de toutes les réservations qui ne sont pas encore validées, c'est à dire pour lesquelles aucune salle n'est attribuée.

selectBooking(int reference): void
: écoute les événements sur le tableau des réservations. Lorsqu'on clique sur celui-ci, on enregistre la réservation sélectionnée puis on affiche ses informations associées.

getBookingTeacherInfos() : struct String
: retourne sous la forme d'une structure composée de strings toutes les informations sur l'enseignant, c'est à dire nom, prénom, téléphone, mail et enseignement.

getSelectedBookingDate() : String
: retourne la date de la réservation sélectionnée.

getSelectedBookingSchedule() : String
: retourne le créneau de la réservation sélectionnée.

getWeekUnvalidSchedules() : list<struct String>
: retourne sous forme de liste de structure les dates et créneaux pour lesquelles il y aucune salle libre avec les paramètre de la réservation sélectionnée.

setSelectedBookingDate(String date) : void
: remplace l'ancienne date de la réservation par la nouvelle date.

setSelectedBookingSchedule(String schedule) : void
: remplace l'ancien créneau de la réservation par le nouveau créneau.

setSelectedBookingFeatures(list<String> features) : void
: remplace les anciennes caractéristiques de la réservation par les nouvelles.

saveSelectedBooking() : void
: sauvegarde la réservation sélectionnée et lui associe une salle.

deleteSelectedBooking() : void
: supprime la réservation sélectionnée.

Teacher

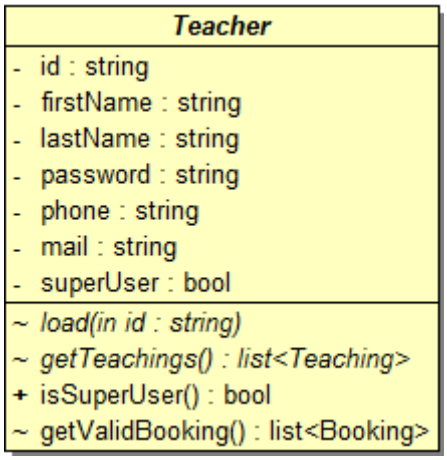


Figure 10 - Teacher

Brève description
La classe abstraite Enseignant représente la classe des enseignants.
Il y a que des enseignants sans titre, pas de titulaires vacataires…
On utilisera deux responsables seulement : le responsable UE et le responsable Matière. Si l'utilisateur n'est pas un responsable, c'est un simple enseignant.

Description des attributs
id : String
: représente l'identifiant de l'enseignant.

firstName : String
: indique le prénom de l'enseignant.

lastName : String
: indique le nom de l'enseignant.

password : String
: indique le mot de passe de l'enseignant.

phone : String
: indique le numéro de téléphone de l'enseignant.

mail : String
: indique le mail de l'enseignant.

superUser : bool
: permet à l'utilisateur de pouvoir modifier toutes les UE (plein droit) seulement si l'attribut est à vrai.

Elle possède de plus un attribut référençant une liste d'enseignements.

Description des méthodes
load(String id)
: charge l'enseignant dont l'id est donné en paramètre.

getTeachings() : list<Teaching>

: retourne la liste des enseignements de cet enseignant.

isSuperUser() : bool
: retourne vrai si l'enseignant est un responsable, faux sinon.

getValidBooking() : list<Booking>
: retourne la liste des réservations qui ont été faites par l'utilisateur et qui ont été validées (ayant reçues une salle).

Booking

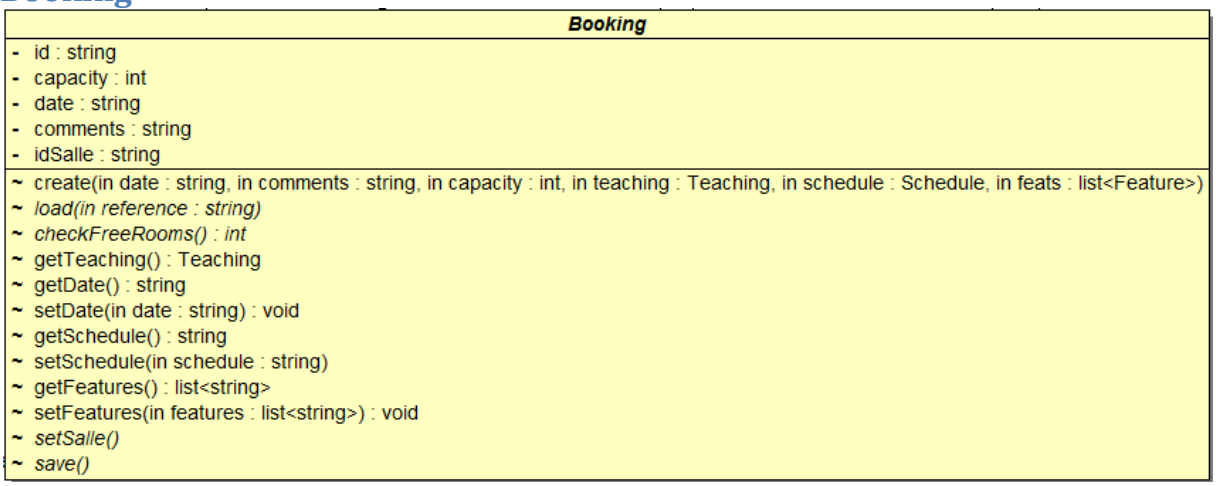


Figure 11 - Booking

Brève description
La classe abstraite Booking représente la classe des réservations.
Il est important de noter qu'on ne réserve pas un créneau de 3H, mais on réserve créneau par créneau. Une réunion est une réservation qui relie un enseignement mais qui n'a pas de groupe.

Description des attributs
id : String
: représente l'identifiant de la réservation

capacity : int
: indique la capacité maximale en nombre de places

date : String
: indique la date à laquelle l'enseignement a lieu

comments : String
: représente les commentaires que peut faire un enseignant sur la réservation de l'enseignement

idSalle : int
: indique le numéro de la salle dans laquelle l'enseignement aura lieu

Elle possède de plus trois attributs référençant un enseignement, un créneau et une liste de caractéristiques.

Description des méthodes

create(String date, String comments, int capacity, Teaching teaching, Schedule schedule, List<Feature> feats)
: crée une nouvelle réservation.

load(String reference) : void
: charge la réservation qui a pour identifiant reference.

checkFreeRooms() : int
: retourne le nombre de salles de disponibles en fonction des paramètres de cette réservation.

getTeaching() : Teaching
: retourne l'enseignement lié à cette réservation.

getDate() : String
: retourne la date de réservation.

setDate(String date) : void
: remplace la date de reservation.

getSchedule() : String
: retourne le créneau de réservation.

setSchedule(String schedule) : void
: remplace le créneau de réservation.

getFeatures() : List<String>
: retourne la liste des caractéristiques de la réservation.

setFeatures(List<String> features) : void
: remplace la liste des caractéristiques de la réservation.

setSalle() : void
: associe une salle à la réservation.

save() : void
: sauvegarde la réservation.

Schedule

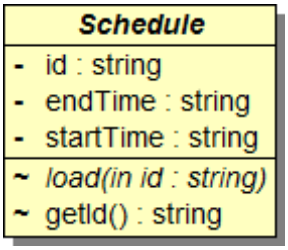


Figure 12 - Schedule

Brève description

La classe abstraite Schedule représente la classe des créneaux.

Les créneaux sont fixes (il n'y a donc pas de créneaux décalés).

Description des attributs
id : String
: représente l'identifiant du créneau.

endTime : String
: représente le début du créneau.

startTime : String
: représente la fin du créneau.

Description des méthodes
load(String id) : void
: charge le créneau avec l'identifiant correspondant.

getId() : String
: retourne l'identifiant du créneau.

Feature

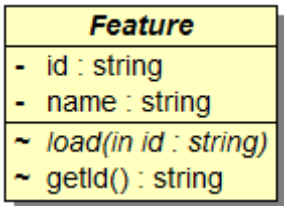


Figure 13 - Feature

Brève description
La classe abstraite Feature représente les caractéristiques d'une salle (exemple : tableau blanc, amphithéâtre, paillasse…).

Description des attributs
id : String
: représente l'identifiant de la caractéristique.

name : String
: indique l'intitulé de la caractéristique.

Description des méthodes
load(String id) : void
: charge la caractéristique correspondant à l'identifiant donné en paramètre.

getId() : String
: retourne l'identifiant de la caractéristique.

Teaching

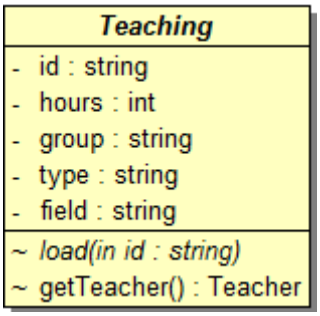


Figure 14 - Teaching

Brève description
La classe abstraite Enseignement représente la classe des enseignements.
Pour avoir le nombre d'heures total, on fait la somme du nombre d'heures prévues *(hours)* pour chaque enseignement. Cela permet d'avoir, par exemple, 10H de TD pour un prof d'une matière et 20H pour un autre. Au final, le nombre d'heures total est de 10+20 = 30h.

Description des attributs
id : String
: représente l'identifiant de l'enseignement.

hours : int
: indique le nombre d'heures de l'enseignement.

group : int
: indique le groupe de l'enseignement.

type : String
: indique le type de l'enseignement (Cours, TD, TP).

field : String
: indique l'intitulé de l'enseignement.

Elle possède de plus un attribut référençant un enseignant.

Description des méthodes

load(String id)
: charge l'enseignement dont l'id est donné en paramètre.

getTeacher() : Teacher
: retourne l'enseignant qui enseigne cet enseignement.

IV. Persist Layer

Cette couche représente la couche persistante des données. Nous utiliserons dans notre implémentation une base de données sous Oracle, fournit par Polytech Montpellier.

PersistFactoryJDBC

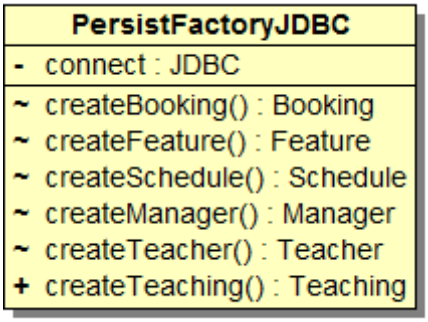


Figure 15 - PersistFactoryJDBC

Brève description
La classe PersistFactoryJDBC hérite de la classe PersistFactory (elle hérite donc des attributs et méthodes).
Le système possède des versions concrètes dérivées de la classe PersistFactory, comme par exemple PersistFactoryJDBC. Celle-ci possède sa propre implémentation des fonctions citées ci-dessus, pouvant créer des objets tels que ReservationJDBC, CaracteristiqueJDBC, CreneauJDBC, ManagerJDBC, EnseignantJDBC, EnseignementJDBC.
Chacun de ces produits dérive d'une classe abstraite comme Teacher, Teaching…
Cette classe permet de réaliser la connexion à la base de données en lui « passant » l'identifiant de l'utilisateur.

Description des attributs
connect
: Objet qui contient la connexion à la base de données.

BookingJDBC

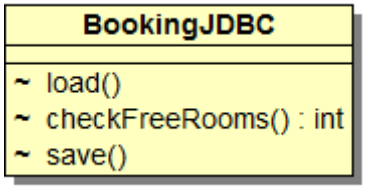


Figure 16 - BookingJDBC

Brève description
La classe BookingJDBC hérite de la classe Booking (elle hérite donc des attributs et méthodes). Cette classe se charge de récupérer toutes les informations relatives à une réservation à travers une base de données.

ScheduleJDBC

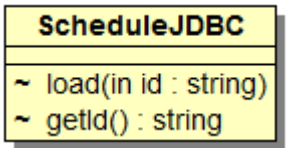


Figure 17 - ScheduleJDBC

Brève description
La classe ScheduleDBC hérite de la classe Schedule (elle hérite donc des attributs et méthodes).
Cette classe se charge de récupérer toutes les informations relatives à un créneau à travers une base de données.

FeatureJDBC

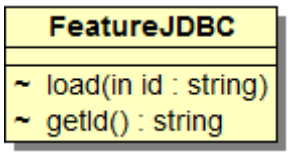


Figure 18 - FeatureJDBC

Brève description
La classe FeatureJDBC hérite de la classe Feature (elle hérite donc des attributs et méthodes).
Cette classe se charge de récupérer toutes les informations relatives à une caractéristique à travers une base de données.

ManagerJDBC

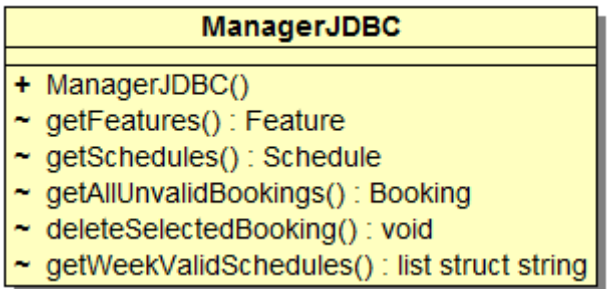


Figure 19 - ManagerJDBC

Brève description
La classe FeatureJDBC hérite de la classe Feature (elle hérite donc des attributs et méthodes).
Cette classe se charge de récupérer toutes les informations générales à travers une base de données.

TeacherJDBC

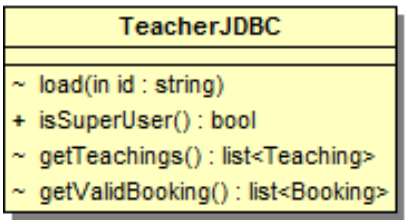


Figure 20 - TeacherJDBC

Brève description
La classe TeacherJDBC hérite de la classe Teacher (elle hérite donc des attributs et méthodes).
Cette classe se charge de récupérer toutes les informations relatives à un enseignant à travers une base de données.

TeachingJDBC

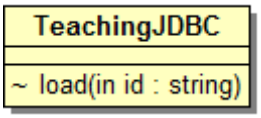


Figure 21 - TeachingJDBC

Brève description
La classe TeachingJDBC hérite de la classe Teaching (elle hérite donc des attributs et méthodes).
Cette classe se charge de récupérer toutes les informations relatives à un enseignement à travers une base de données.