README

Contenu de l’archive

* Un script SQL permettant de créer la BDD et les tables en local et un autre pour créer celles du WEBTP (qui est déjà initialisée mais au cas où vous voudriez la réinitialiser pour diverses raisons).
* Les trois projets Maven qui contiennent le code source.
* Un fichier contenant le mot de passe de la base de données WEBTP.
* Ce fichier README.

Pré-requis

* Un ordinateur avec un JRE 8.
* Maven OU Eclipse munit de Maven (sinon bon courage pour le compiler à la main ..)
* De la patience.
* Avoir regardé l’intégrale de la petite maison dans la prairie.

Etapes à suivre

* Extraire l’archive
* Exécutez le script SQL dans la BDD MySQL de votre choix (la structure existe déjà sur le Webtp avec des données dummy et la session “admin”). Le script de la vdd local ne fonctionnera pas sur le WEBTP (c’est pour cela qu’il y’a deux script).
* Ouvrez une fenêtre de commande dans le dossier et exécuter la commande “mvn clean install; java -cp toku-swing/target/toku-swing-0.0.2-SNAPSHOT-jar-with-dependencies.jar fr.lille1.univ.coo.tp.Main” pour nettoyer, compiler et exécuter le programme.
* Choisissez la façon dont vous voulez vous connecter :
  + Soit à une nouvelle base de données MySQL (locale)
  + Soit à la base de données Webtp où vous aurez juste à taper le mot de passe fourni et où la structure existe déjà avec le compte administrateur du programme.

Architecture générale du projet

Une architecture par feature (<http://www.javapractices.com/topic/TopicAction.do?Id=205>)

Tout d’abord le projet s’appelle Toku ! (Talk en japonais car je suis un garçon très original). Le projet est en fait divisé en trois répertoires principaux. Un répertoire parent “toku” et deux sous répertoire “toku-core” et “toku-swing” qui sont en fait des modules Maven du projet parent “Toku”.

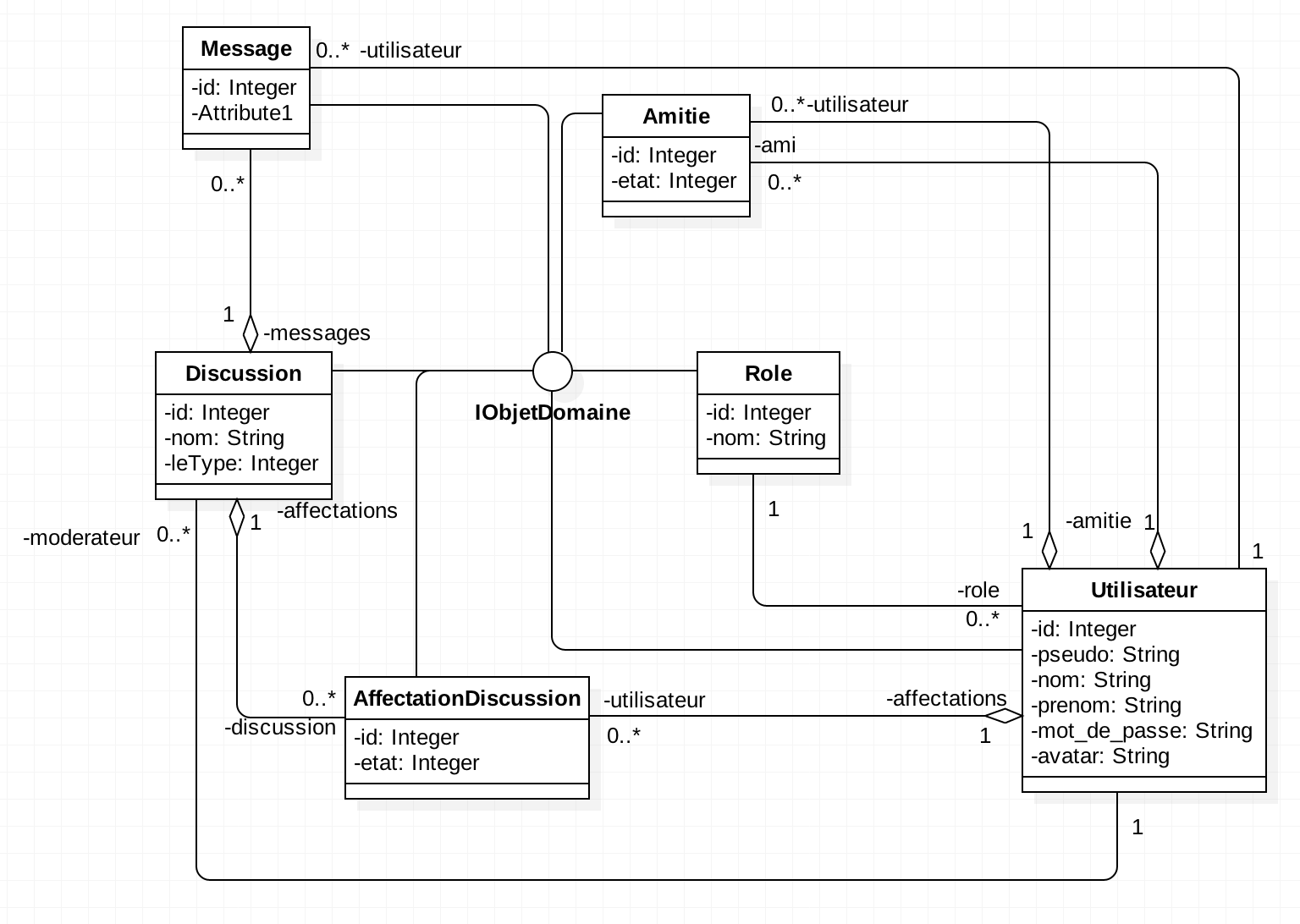
Comme vous l’aurez compris le dossier toku-core contient tous le modèle du programme et le dossier toku-swing contient la partie vue + contrôleurs.

Dans les deux dossier j’ai respecté une certaines normes : Toutes les classes en rapport avec un thème se trouve dans un même package. Par exemple les classes Utilisateur, UtilisateurService et UtilisateurValidateur se trouve dans un package “utilisateur”. Il me semble plus facile de s’y retrouver plutôt que d’avoir des package “model”, “service” et autres .. C’est une façon comme une autre de structurer les packages.

La couche Domaine (UML)

La couche domaine contient toutes les classes du domaine. Ainsi, nous y retrouvons les classes suivantes :

* Utilisateur : Un utilisateur
* Rôle : Le rôle d’un utilisateur (Normal ou Admin)
* Discussion : Une discussion (En groupe ou privée)
* Message : Un message de discussion
* Amitié : Un lien d’amitié entre deux utilisateurs
* AffectationDiscussion : L’affectation d’un utilisateur à une discussion à laquelle il participe (peut importe le type de discussion)



La couche persistance (MCD)

Voici le MCD correspondant à l’UML du Domaine :

A COMPLETER

Afin de mapper les tables de la BDD avec mes classes du Domaine, j’ai créé des annotations que j’ai placé sur ces dernières. Puis, grâce à un DAO générique, j’ai fait en sorte que l’on puisse, de façon générique, communiquer avec chaque table associée à sa classe. Ces annotations se trouvent dans le package “annotations” du modèle. Voici leurs détails :

* @Table : Cette annotation est à placer au dessus d’une classe. Par défaut le nom de la table a le même nom que la classe mais on peut spécifier le nom de la table s’il est différent du nom de la classe. On peut aussi spécifier le nom de la clé primaire de la table.
* @Id : Cette annotation est à placer au dessus de l’attribut qui représente la colonne contenant la clé primaire de la table. Le nom de la colonne est le même que le nom de l’attribut mais on peut en spécifier un s’il est différent.
* @Colonne : Cette annotation représente une colonne de la table. Le nom de celle-ci est le même que le nom de l’attribut mais on peut en spécifier un s’il est différent.
* @Transient : Cette annotation est à placer sur tout attribut qui ne doit pas être récupéré de la base de données pour des raison de sécurité (Par exemple un mot de passe).
* @Vue : Cette annotation est à coupler avec une annotation @Table sur une vue. On a ainsi l’information de la table qui à servi à créer la vue. Il faut alors spécifier le nom de la vue en base de données et spécifier dans l’annotation @Table, le nom de la table et le nom de la colonne représentant la clé primaire. Ceci est important car les requêtes SELECT se feront sur la Vue tandis que les autres types de requêtes se feront sur la table (car on ne peut pas faire de mise à jour sur la Vue à moins qu’elle ne respecte certaines règle : <http://dev.mysql.com/doc/refman/5.7/en/view-updatability.html>).
* @ColonneVue : Cette annotation est à placer au dessus d’un attribut qui représente une colonne de la Vue qui elle prend les valeur d’une colonne de la table. Ceci permet par exemple d’avoir les clé primaires de la table dans la Vue et de lier ces deux colonnes afin de savoir quelles clés de la tables doivent être utilisées pour mettre à jour la table lorsque l’on fera nos requêtes de mise à jour sur la Vue. Ce sont un peu des colonnes “proxy”.
* @UnAUn : Cette annotation n’est plus utilisée car elle fait la même chose que @PlusieursAUn mais je m’en suis rendu compte trop tard. Je l’ai donc passée en @Deprecated.
* @PlusieursAUn : Elle représente une relation n - 1. Par exemple : Plusieurs Discussion peuvent être créé par 1 utilisateur. Cette annotation prend en paramètre le nom de la clé étrangère et le type de la classe représentant la partie “1” de la relation. Le DAO générique injecte donc 1 objet du type spécifié dans les objets contenant celui-ci. Cette annotations se place sur l’attribut représentant l’objet à injecter.
* @UnAPlusieurs : Elle représente une relation 1 - n. Par exemple : Une Discussion peut avoir plusieurs messages. Cette annotation prend en paramètre le nom de la clé étrangère contenu dans la table représentant les objets de la partie “n” de la relation et le type de ces objets. Le DAO générique injecte donc une liste d’objet du type spécifié dans l’objet contenant cette liste. Cette annotations se place sur l’attribut représentant la liste à injecter.
* @PlusieursAPlusieurs : Cette annotation représente une relation n - n. Par exemple : Plusieurs utilisateurs peuvent être affectés à plusieurs Discussion. On aura ainsi d’un côté la liste des Discussion dans l’objet de type Utilisateur et de l’autre côté, la liste des membres dans l’objet de type Discussion. Cette annotation prend en paramètre le type de la classe représentant la table d’association, le nom de la colonne représentant la clé des objets contenant la liste, le nom de la colonne représentant la clé des objets contenus dans la liste.

Tout objet dont le type est spécifié en paramètre d’une annotation va être créé de façon paresseuse grâce au virtual proxy générique et à des factory générique qui effectuent cette remontée grâce aux informations des 4 annotations de cardinalités (UnAUnFactory, PlusieursAPlusieursFactory, etc …). Les autres objet de type plus primitif seront récupéré de façon normale grâces aux resultsets.

Lors d’une requête, le DAO va utiliser la réflexivité pour scanner la classe représentant la table à questionner et va utilisateur les Factory pour créer des Proxy.

Enfin les objets du domaine sont stockés en mémoire grâce à des Map d’identité (Identity Map) afin de ne pas dupliquer les références vers ces objets. Cela permet d’optimiser les requêtes sur la base de données mais aussi d’avoir une seule référence par objet et ainsi d’économiser la mémoire.

Voici l’UML de la couche persistance.

A COMPLETER

Mots clé : Design pattern Proxy, Design pattern Factory, Design pattern identity map

La couche Service

La couche Service est composé de deux classes : UtilisateurService et DiscussionService. Comme leur nom l’indique, la première gère tout ce est en rapport avec les utilisateurs comme la création de compte, la connexion, la gestion des rôle, l’ajout d’ami, etc, tandis que la deuxième gère tout ce qui touche aux discussions comme la création de discussion, l’envoie de message, l’affectation à une discussion etc. Ces classes sont utilisées par les contrôleurs de la couche Présentation.

UML

Le Unit of work

Afin de limiter les appels à la base de données, j’ai utilisé un Unit Of Work (UOW) qui est une sorte de cache où sont stockés tous les objets du Domaine créés, modifiés ou supprimés. Ce UOW suit le design pattern Observer pour observer tout les objets du Domaine sur lesquels ont effectue des changement. Il est notifié de la création, de la modification de la suppression de tous ces objets. Dès qu’un objet est créé ou supprimé dans la couche service, il est envoyé à ce UOW. Dès qu’un objet est modifié, il notifie lui même le UOW de ces changements en précisant le ou les paramètres modifiés. Enfin, j’ai créé une classe IObservableList qui notifie de l’ajout ou la suppression d’un objet du domaine comme par exemple lors de l’ajout d’un ami, le UOW est notifié de la création d’un nouvel objet Amitié qui a été ajouté à l’IObservableList “amitiés” de la classe Utilisateur.

Une fois que l’on veut faire persister ces changements en base, on appelle la fonction “commit” de ce UOW qui va utiliser le design pattern Visitor pour connaître le type des objets à traiter et donc à faire persister. Il se sert de trois objet : Le CreationCommiter, le ModificationCommiter et le SuppressionCommiter. Ce sont ces trois objets qui impléments la classe Visitor ce qui leur permet de savoir comment créer/modifier/supprimer chaque type d’objet du Domaine.

UML

La couche Présentation

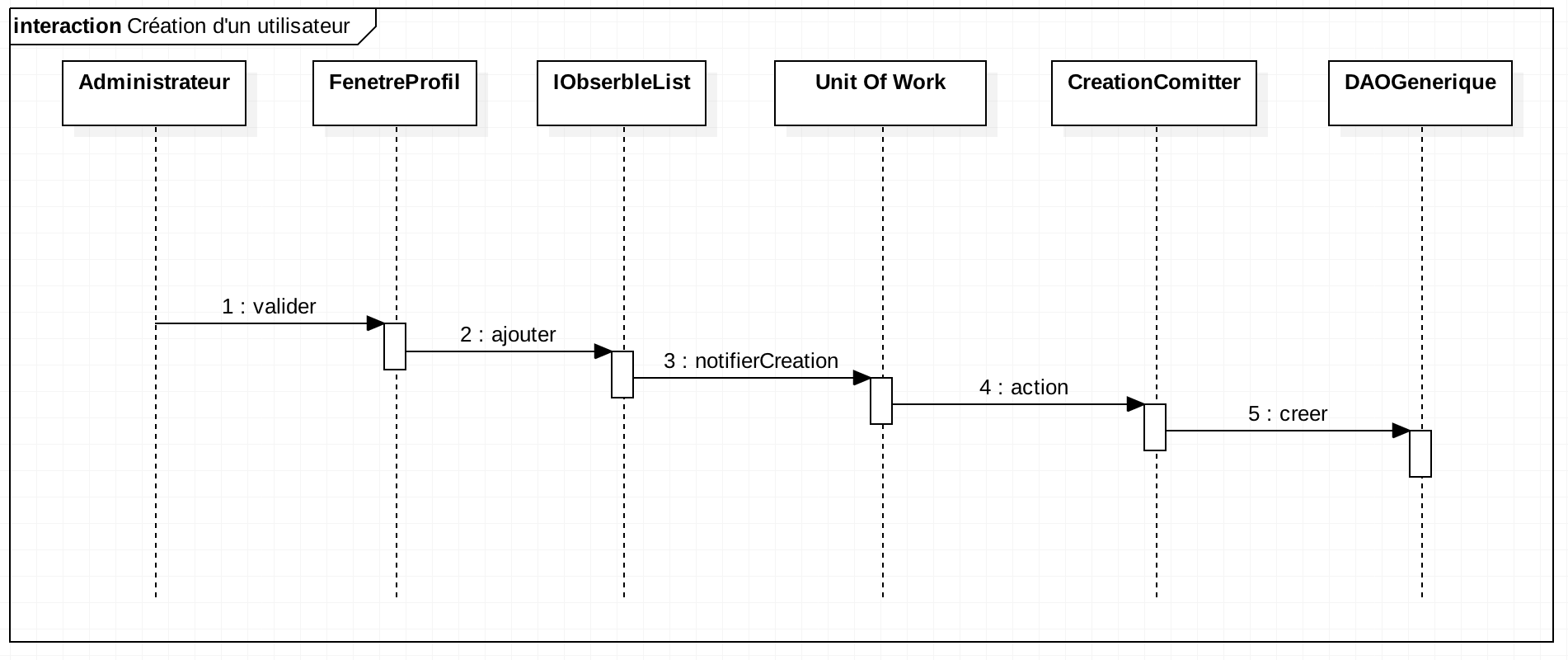
Cette couche rassemble toutes les classes représentant l’interface utilisateur. Elle est divisé en deux catégorie : Les fenêtres créés avec des composants de base ou custom et les contrôleurs qui interceptent les actions de l’utilisateur, récupèrent les informations à traiter et les passent à la couche Service. L’interface utilisateur sera décrite dans un prochain chapitre.

Cas d’utilisation

Création et suppression de compte utilisateur

Un administrateur peut créer et supprimer un utilisateur. Il doit pour cela renseigner les infos suivantes :

* Pseudo
* Mot de passe
* Nom
* Prénom
* Rôle (Admin ou utilisateur normal)



Axes d’amélioration

* Ajouter la gestion des avatar. Il suffit de demander à l’utilisateur de sélectionner une image. A ce moment on la copie dans un dossier dont on va stocker l’url en base.

Problèmes rencontrés

- Afin de pouvoir gérer les amis, j’ai créé une vue projet\_amitie\_view. Sauf que comme une vue ne peut pas être mise à jour avec UPDATE j’ai du créé des annotations spécifiques à mes vue. Cela n’a pas été facile de gérer ce cas car j’ai essayé au maximum de rester générique dans mon DAO..