|  |
| --- |
| TELECOM NANCY |
| Projet de SD |
| Programmation d’un reseau social |
|  |
| **Xavier MARCHAL & David RUBINO** |
| **5/13/2013** |

|  |
| --- |
| [Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.] |

# Introduction

L’objectif de ce projet était de construire un réseau social en langage de programmation JAVA en utilisant seulement le mode console. La sauvegarde des données se réalise uniquement avec des fichiers. Le projet fut réalisé en 2 phases, la première concernant la conception et le seconde concernant le codage à proprement parler des méthodes.

# Description des classes

## 3.1. La classe Invitation

Elle modélise l’objet Invitation qui sera utilisée pour permettre à 2 personnes du réseau social d’entrer en contact. Elle prend en paramètres deux profils, celui qui envoie et celui qui reçoit. La classe ne contient que des getters et une méthode d’affichage.

## 3.2. La classe Localization

Elle modélise le lieu où se trouve la personne et est utilisée dans profil. Il était plus simple de créer un objet Localization qui regroupait à la fois la ville, l’état et le pays plutôt que de rentrer tous ces paramètres en tant que String. La classe contient seulement des getters et des setters ainsi qu’une méthode d’affichage.

## 3.3. La classe Message

Elle caractérise l’objet qui sera envoyé lorsqu’un utilisateur voudra entrer en contact avec un autre. Elle prend 3 paramètres en entrée : la personne qui envoie, la date à laquelle le message est envoyé et le contenu. Les méthodes de la classe sont uniquement des getters et setters.

## 3.4. La classe Profile

Il s’agit de la plus grande classe du projet, celle qui contient toutes les méthodes qu’un abonne pourra utiliser. Un profile est constituée par un nom, prénom et mots de passe de l’utilisateur ainsi qu’une date mise sous forme de String en format spéciale et qui est parsee pour que Java reconnaisse le format, une localization géographique, un email, et un booléen qui détermine l’activation du compte. Lorsque le compte est créé, ce booléen est par défaut mis à true.

Outre de nombreux getters et setters renvoyant les différents paramètres du compte ainsi qu’une méthode d’affichage, la classe contient les méthodes suivantes.

**writeMessage :** à l’aide d’un Scanner, l’utilisateur rentre dans la console ce qu’il veut afficher sur son mur. Le message est ensuite affiché avec l’heure de publication (déterminée par l’horloge de l’ordinateur).

**writeMessageToFriend :** pour écrire un message a un ami. Aucun paramètre à fournir à l’appel de la méthode, ils sont tous demandes dans la console. On affiche la liste d’amis de l’utilisateur et on lui demande de choisir celui à qui il veut envoyer le message. On écrit ensuite le message. Une boucle if…else gère les cas où il n’y a pas d’amis / l’entrée est invalide.

**displayFriends :** affiche tous les amis de l’utilisateur courant. Il n’y a aucun paramètre à entrer.

**displayFriendProfile :** pour afficher le profil d’un ami. On affiche dans la console la liste d’amis de l’utilisateur courant et celui-ci choisi l’ami dont il veut afficher le profil en utilisant un Scanner. Une boucle gère les cas particuliers (entrées invalides et/ou pas d’amis).

**displayWall :** affiche le mur de l’utilisateur courant

**displayFriendWall :** comme avant, on affiche la liste d’amis de l’utilisateur courant, un Scanner récupère le choix de l’ami dont on souhaite voir le mur et l’affiche en utilisant la méthode displayWall précédemment définie.

**displayInvitation :** affiche la liste des invitations reçues, sinon un message texte si la liste est vide.

**sendInvitation :** prend un profil en paramètre et ajoute un objet Invitation dans la liste d’invitations du profil passe en paramètre.

**acceptInvitation :** cette méthode parcourt la liste de l’utilisateur courant tant qu’elle n’est pas vide. Chaque invitation est examinée par l’utilisateur qui a le choix de l’accepter ou de la refuser. Dans le premier cas, l’utilisateur qui a envoyé l’invitation est ajoute à la liste d’amis de l’utilisateur courant et vice versa. Rien ne se passe dans le deuxième cas. Dans les 2 cas, une fois que l’invitation a été examinée, elle est supprimée de la liste des invitations. Une boucle if…else gère le cas où la réponse rentrée dans la console n’est pas la bonne.

**cancelInvitation :** pour permettre à l’utilisateur courant d’annuler sa demande d’ajout a quelqu’un. On affiche toutes les invitations de la liste et ‘utilisateur choisit (au moyen d’un Scanner) l’invitation qu’il veut annuler. On gère les cas où la liste est vide et où l’utilisateur rentre n’importe quoi dans la console.

**Serialiser :** permet la sauvegarde du réseau social sous la forme d’un fichier .bin qui est enregistrée dans le répertoire du projet. Ainsi, l’état du réseau social peut être sauvegarde chaque fois que l’utilisateur se déconnecte et lui évite donc de tout recréer à chaque fois, ce qui peut s’avérer quand même très utile…

**Deserialiser :** prend un email en paramètre et permet de rechercher le fichier correspondant à l’utilisateur dont l’email est passe en paramètre. Le compte de la personne trouvée est alors supprimé.

## 3.5. La classe SocialNetwork

Cette classe contient les méthodes définissant un réseau social et est utilisée pour créer Facebook. Elle est constituée d’un constructeur vide ainsi que d’un constructeur prenant en paramètre une HashMap qui associe la clé (dans notre cas l’adresse mail) à une valeur (le profil d’une personne). Les méthodes de la classe sont les suivantes.

**displayRegisteredUsers :** affiche tous les utilisateurs inscrits sur le réseau social.

**friendOfFriend :** permet d’afficher les amis des amis d’un utilisateur en choisissant une profondeur limite qui est rentrée par l’utilisateur dans la console.

**getUsersNumbers :** retourne le nombre d’utilisateurs du réseau.

**createAccount :** méthode interactive avec l’utilisateur qui utilise le Scanner pour créer un compte Facebook en demandant à l’utilisateur d’entrer tous les paramètres. Une fois que le Profile est créé, on vérifie qu’il n’existe pas déjà dans le réseau social. Si ce n’est pas le cas, on l’y ajoute.

**openSession :** permet à un utilisateur d’ouvrir une session. L’utilisateur doit rentrer son adresse mail et son password pour que l’authentification puisse se faire.

**closeSession :** méthode triviale qui ferme le compte de l’utilisateur courant en mettant celui-ci a null.

**activateAccount :** permet d’activer un compte après sa création ou sa désactivation. Par défaut, tous les comptes crées sont actives. On peut bien sur modifier cette option pour obliger tous les utilisateurs à activer leurs comptes après création, mais nous avons pensé que cela n’était pas très utile. Cette méthode sert donc les gens qui souhaitent réactiver leur compte après une période de longue inactivité. Un compte doit d’abord être actif pour que l’utilisateur puisse utiliser Facebook.

**deactivateAccount :** méthode inverse la précédente, qui permet donc de désactiver un compte. L’utilisateur doit rentrer son adresse mail et son password pour que la méthode soit prise en compte.

**Search :** permet à un utilisateur de rechercher sur Facebook quelqu’un à l’aide du nom et du prénom. On utilise ici une structure LinkedList de Profile pour créer une liste temporaire dans laquelle on ajoute tous les utilisateurs enregistres sur Facebook. On parcourt la liste et on retourne la première personne correspondant à la personne recherchée. On suppose donc ici que 2 personnes ne peuvent pas avoir les mêmes noms et prénoms ce qui, sur un réseau social mondial, n’est pas tout le temps vérifié.

**getCurrentUser :** retourne l’utilisateur qui est actuellement connecte sur Facebook.

**getResgisteredUsers :** retourne une HashMap<String,Profile> contenant tous les utilisateurs enregistres sur Facebook.

**Serialiser :** cette méthode permet d’enregistrer le réseau social dans son état dans un fichier .bin sauvegarde dans le répertoire du projet. Elle permet donc de sauver tous les utilisateurs du réseau et leurs états respectifs (messages, demandes d’amis) pour que lorsqu’on relance l’application, les utilisateurs n’aient pas à tout recommencer. Plutôt utile, donc.

**Deserialiser :** inverse de la méthode précédente, on remet le SocialNetwork à 0. Utile si on souhaite réinitialiser le système complet.

## 3.6. La classe Wall

Cette classe caractérise le mur d’un utilisateur et est constituée d’une ArrayList de Message. En plus d’un getter et d’un setter pour retourner / modifier l’ArrayList et d’une méthode toString pour afficher le mur, la classe contient d’autres méthodes.

**addMessage :** pour ajouter un message sur le mur (dans l’ArrayList).

**deleteMessage :** permet d’effacer un message du mur en passant le message à effacer en paramètre. Une fois cette opération faite, il n’y a plus de moyens de revenir en arrière pour récupérer le message efface.

## 3.7. La classe Main

C’est ici qu’aura lieu l’interface avec l’utilisateur dans la console en utilisant un objet Scanner. On commence par créer un réseau social puis on demande à l’utilisateur de sélectionner ce qu’il souhaite à l’aide d’un menu. On utilise la structure switch…case dans un do…while afin de proposer toutes les options crées dans les classes du dessus. Tant que l’utilisateur ne choisit pas l’option quitter, le programme tournera. Une fois qu’il choisit quitter, l’état de Facebook est enregistré en utilisant la méthode serialiser de la classe SocialNetwork.

## 3.8. Les classes de Test (TestInvitation, TestLocalization, TestProfile, TestSocialNetwork, TestWall)

Ces classes ont été écrites pour tester les différentes classes du projet. Tous les tests sont indépendants et peuvent donc être exécutés seuls. Ils permettent de s’assurer du bon fonctionnement des méthodes mais ne sont pas utilisés dans le Main.

# Conclusion

Ce projet nous a permis de développer une première application console d’un réseau social. On y retrouve les principales interactions qui ont lieu sur Facebook. Evidemment, des améliorations peuvent être portées à ce projet. En tout premier lieu, une interface graphique plutôt qu’une interface console est bien plus agréable pour l’utilisateur. Enfin, au sujet de la sauvegarde des données, il serait plus judicieux d’utiliser des bases de données relationnelles plutôt que des systèmes de fichiers.