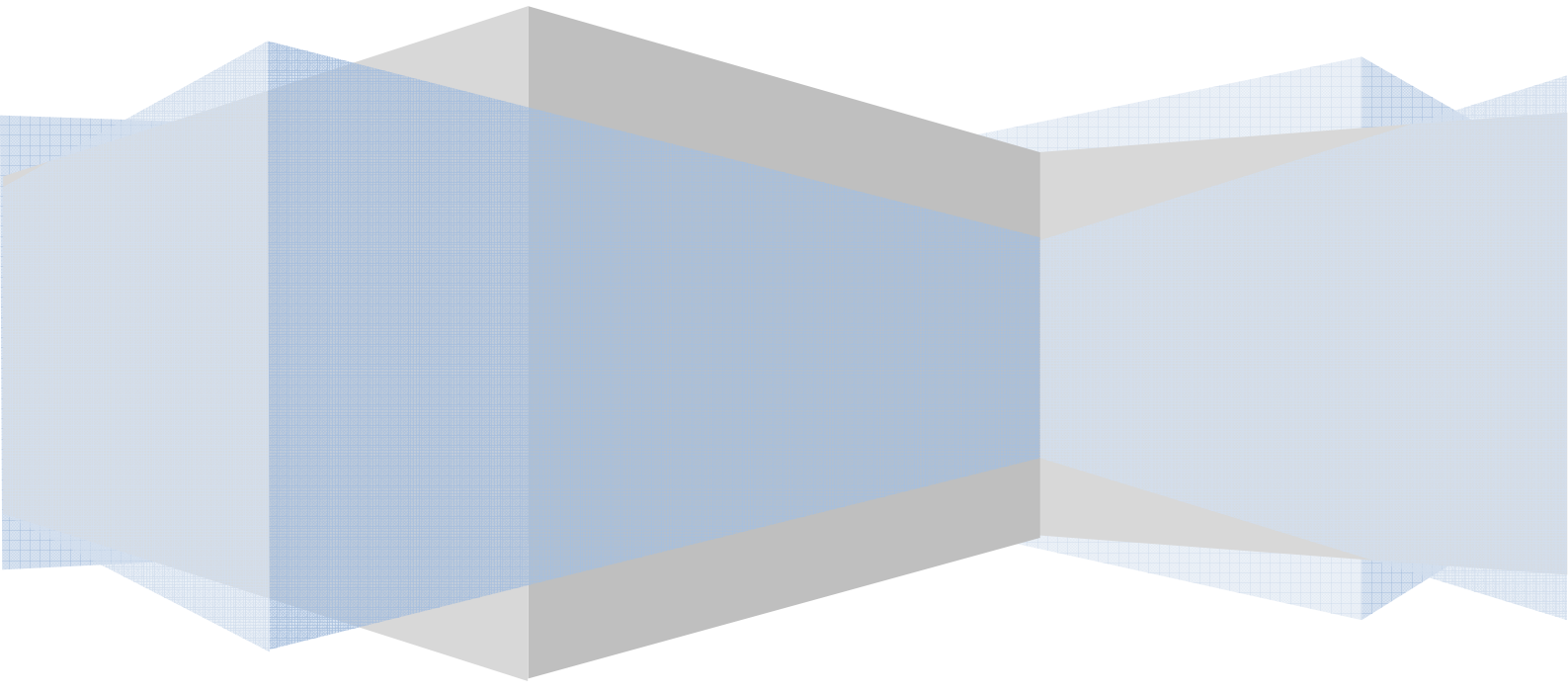
Licence 3 CDA – 2010/2011

Client FTP Java

Maxence Jaouan



Sommaire

Introduction

Le but du projet

Le projet qui nous a été demandé de réaliser consiste à créer un client FTP en Java pouvant être capable de transférer des fichiers, de naviguer dans les dossiers distants, de renommer et effacer un fichier ou dossier, ainsi que de créer de nouveaux dossiers.

Les moyens utilisés

Le projet devant être programmé en Java et sachant que les 2 meilleurs IDE pour ce langage étaient Eclipse et Netbeans, je me suis décidé d'utiliser Eclipse pour le concevoir.
Evidement un serveur FTP était nécessaire pour les tests. Le plus simple était d'en installer nous même. Le serveur FTP FileZilla Server a donc fait office de serveur pour tester mon client.

Informations sur le client FTP

Pourquoi une version Linux et Windows ?

Parce que j'utilise la librairie graphique SWT d'eclipse, qui est basée sur du code C. Elle utilise GTK (version 2.4.1 au minimum, peut être 2.2.1) sous Linux, et le système d'interface de base dans la version Windows.
La version Windows ne marche pas sous Linux.
La version Linux marche **peut-être** sous Windows si GTK est installé, je n'ai pas eu l'occasion de tester.

Quelques fonctionnalités

- les raccourcis clavier (tel que F2 pour renommer, Entrée pour transférer, Suppr pour effacer, F5 pour rafraichir un dossier).
- le transfert de fichier (Upload et Download)
- le Renommage, effacement de fichier
- la création de dossier
- la reprise d'un transfert (via fonction APPE pour l'Upload et REST/STOR pour le Download), l'écrasement de fichier, le changement du nom du fichier de destination, une demande sera effectuée auprès de l'utilisateur
- la gestion de plusieurs comptes FTP (sauvegarde du nom du serveur, du port, du nom d'utilisateur et du mot de passe) (Je n'ai pas eu le temps de mettre en connexion anonyme de base, au cas où les zones de saisies soient vides)

Les améliorations possibles

- le transfert de dossier (Upload et Download)
- la multi sélection de fichier dans la liste
- le glisser/déposer
- les droits. Il est capable de savoir et de modifier les permissions via la classe Permission, mais il ne permet pas à l'utilisateur de modifier des droits d'un fichier/dossier via CHMOD. Et n'en prend pas compte dans les transferts
- le nom d'utilisateur/groupe auquel un fichier/dossier local est lié.
- la mise à jour de lecteur/fichier/dossier en temps réel. Il faut rafraichir le dossier parent pour voir les modifications.
- si la connexion avec le serveur FTP plante au milieu d'une procédure nécessitant plusieurs commandes, le client se reconnectera automatiquement au serveur, cependant il ne reprendra pas au début de la procédure, il reprendra ou il en était

Communication entres objets

Les objets communiquent avec l'instance qui les crée via des écouteurs. De manière à retirer les dépendances entre les classes.

Divers :

- Le logiciel gère 2 uploads et 2 downloads en simultané, si d'autres sont rajoutés, ils sont rajoutés dans la liste d'attente. (J'ai mis arbitrairement sur 2, il suffit de changer la valeur de la variable "max_tasks" dans le fichier Scheduler.java
- Un transfert devient manuel à partir du moment où il a été mis en pause.
- Un transfert est affiché comme automatique lorsqu'il y a "<<<" ou ">>>" dans la colonne "<>". Au cas où il n'y ait que "<<" ou ">>", le transfert est manuel et doit être démarré via clique droit.

Toutes les fonctions ont été testées et sont parfaitement fonctionnelles avec FileZilla Server sous Windows Seven.

Les classes

MainFTP.java

C'est le cœur du programme. C'est cette classe qui gère l'object FTP, et les fenêtres. Chacun de ces objets étant totalement indépendant, c'est aussi cette classe qui va faire office de messager. Lorsque l'interface aura besoin d'une information (liste des fichiers par exemple), il préviendra le cœur du programme via un écouteur, le cœur interrogera l'objet FTP, puis renverra la réponse à l'interface.

FTP.java

Classe de communication FTP, aucune commande FTP n'est présente dans les autres fichiers, excepté dans UploadTask.java et DownloadTask.java qui gèrent eux-mêmes les transferts

FTPFile.java

Fichier virtuel associé à un fichier FTP, hérite de java.io .File, de manière à simplifier l'utilisation des fichiers dans FileTreeItem.

LabelText.java

Zone de saisie avec un texte d'indication
Exemple : Une zone de saisie avec comme texte "Cliquez-ici", et lorsque la zone de saisie prend le focus, le texte disparait pour laisser l'utilisateur rentrer une valeur, si l'utilisateur laisse la zone de saisie vide, le texte "Cliquer-ici" réapparait.

FTPInfo.java

Classe faisant office de sorte de structure contenant les informations sur un compte FTP, hérite de java.io.Serializable afin de pouvoir la sérialiser pour l'enregistrer dans un fichier.

Scheduler.java

Ordonnanceur de tâches (Upload/Download), il gère la mise en attente de tâche lorsqu'un certain nombre de tâche est en cours d'exécution.

Task.java

Classe abstraite définissant une tâche de transfert. Elle est capable de prévenir lorsque son état change (En attente, en transfert, en pause, annulée, etc.).

TransfertItem.java
Elément d'une table qui affiche des transferts, cette classe a pour but de simplifier les éléments en liant l'item à une tache, lui permette de gérer lui même s'il a besoin d'être rafraichi, la barre de progression, ainsi que quelques actions sur le transfert, et de gérer lui même son menu contextuel

AskWindow.java
Fenêtre d'interaction avec l'utilisateur, demandant quelque chose tel un nom de fichier/dossier pour un renommage.

MainWindow.java
Fenêtre principale. Communique avec le cœur du programme en cas de nécessité (communication avec le serveur FTP, ajout de transfert, etc.).

Message.java
Classe statique pour envoyer des messages au cœur du programme, qui seront ensuite affichés dans la fenêtre principale.

MSocket.java
Classe "simplifiant" les communications par Socket, intégrant directement les flux d'entrée/sortie.

ConnectionForm.java
Fenêtre de gestion de comptes FTP et de connexion.

FormFuncs.java
Classe de fonctions statiques destinée à alléger le fichier MainWindow.java.

Misc.java
Classe de diverses fonctions statiques (lecture/écriture d'objet dans des fichiers, convertir une taille de fichier en chaine de caractère (1500 -> 1.50Ko ou 1.46Kio), récupérer l'icône d'un fichier, etc.).

FTPFile.java
Fichier virtuel associé à un fichier FTP.

Permission.java

Classe permettant de gérer les permissions à partir d'une chaine de permission du type RWXRWXRWX.

AboutBox.java

Fenêtre « À propos ».

ConnectionManager.java

Classe de fonctions statiques permettant de gérer l'enregistrement et le chargement des comptes FTP.

MStringTokenizer.java

Classe remplaçant plus ou moins StringTokenizer, qui ne possédait pas de fonction pour récupérer tout le texte restant, du moins je ne l'ai pas trouvé.

FileTreeItem.java

Elément de l'arbre qui affiche les fichiers, hérite de TreeItem.
Cette classe a pour but de simplifier les elements en liant l'item a un fichier, lui permette de gerer lui meme l'icone du fichier, ainsi que quelques actions sur le fichier, et de gerer lui même son menu contextuel.

DownloadTask.java / UploadTask.java

Tache de transfert de fichier, herite de la classe Task.

Principes de fonctionnement

Cœur du programme

Initialisation de l'objet FTP (FTP.java).

Initialisation des ordonnanceurs (1 pour l'Upload et 1 pour le Download) (Scheduler.java).

Initialisation et affichage de la fenêtre principale (MainWindow.java).

Chargement des comptes FTP enregistrées (ConnectionManager.java)

Mise en écoute des messages (Message.java) (lorsqu'un message arrive, le cœur du programme peut choisir de le transférer à la fenêtre principale pour qu'elle l'affiche. C'est cela qui permet l'affichage d'informations ou d'erreurs dans la fenêtre principale).

Mise en écoute des communications sockets (MSocket.java) (lorsqu'une ligne est envoyée ou lue, le cœur du programme en est averti et il peut l'afficher dans la fenêtre principale).

Mise en écoute des requêtes de la fenêtre principale (lorsque la fenêtre demande une liste de fichier par exemple, le cœur du programme en est averti et peut ainsi interroger l'objet FTP et renvoyer la réponse à la fenêtre).

Mise en écoute lors de la création d'un FileTreeItem (lorsqu'un FileTreeItem est crée, il se renvoi lui-même au cœur du programme, ce qui permet au cœur de se mettre directement en écoute lorsque le FileTreeItem veut démarrer un transfert, créer un nouveau dossier, rafraichir ses sous-fichiers, etc.).

Ordonnanceur

Contient une ArrayList de tache. On rajoute une tache via addTask() (removeTask() pour en retirer une).

Si l'ordonnanceur est activé (en Auto-Start), dès qu'une tache est rajoutée, mise en pause, arrêtée ou qu'une se termine (une tache prévient son ordonnanceur qu'elle a fini son travail), tant qu'on n'a pas atteint le nombre maximum de taches en simultané et qu'il reste une tâche de l'ArrayList en attente (donc ni en travail, ni annulée, ni en pause), alors la démarre.

Ajout d'un transfert

Vérification si le fichier existe déjà

Si c'est le cas, demander a l'utilisateur si il veut reprendre (si possible, après avoir comparé les tailles) ou écraser le fichier

Demander au cœur du programme d'ajouter le transfert a un des ordonnanceurs

Le cœur ajoute le transfert, et prévient la fenêtre qu'il faut rajouter un transfert à la liste.

Gestion des comptes FTP

Pour enregistrer les comptes FTP, on enregistre directement l'objet ArrayList<FTPInfo> qui contient la liste des informations sur les comptes FTP dans un fichier.

Pour récupérer les comptes FTP, on charge directement l'objet enregistré dans le fichier, puis on le cast en ArrayList<FTPInfo>.

Clique droit/Raccourcis clavier

Les cliques droits et les raccourcis clavier sont d'abord gérés par la fenêtre principale. Selon si cela a été fait alors qu'un élément a été sélectionné (pas le cas par exemple si on clique droit dans un endroit vide de la liste), on le prévient en appelant une de ses fonctions. Autrement on gère directement l'interaction (un menu contextuelle avec certaines options ne nécessitant pas la sélection d'un élément par exemple).

Arrêt/Mise en pause d'une tâche

Une tâche est démarrée dans un thread qui lui est propre. Pour l'arrêtée au cours de son exécution, il y a 2 possibilités.
Tout d'abord il existe la méthode Stop() de la classe Thread qui permet l'arrêt pur et simple du Thread. Mais cette méthode est dépréciée du fait de son caractère brutal. En effet si la tâche s'arrête au milieu d'une communication, cela peut être préjudiciable (pour un download par exemple, la tâche remplit son tampon, mais le thread s'arrête, la tâche n'a pas le temps d'écrire ce qu'elle a reçu dans le fichier de destination. De plus, s'il s'agissait de la toute fin du fichier, il aurait été préférable de finaliser le transfert et le considérer comme terminé). D'autre part, il est préférable de fermer les flux ouverts (socket, fichier). Cette méthode est donc à éviter.
L'autre technique consiste à prévenir le thread qu'il devra s'arrêter quand il pourra. Dans ce programme, l'ordonnanceur mettra à Vrai une des variables de l'objet de transfert (booléen hasToStop). À chaque boucle de transfert (lecture du fichier/écriture dans le socket ou lecture du socket/écriture dans le fichier), la tâche vérifiera si on lui a demandé de s'arrêter, si c'est le cas et qu'il ne s'agissait pas de la fin du fichier (auquel cas le transfert sera finalisé et la demande d'arrêt sera ignorée), elle enregistra son état d'arrêt et de finalisera complètement (fermeture des flux et des connexions).

Communication Thread-Interface

La plupart des librairies graphiques n'autorisent pas l'accès au thread graphique depuis un thread non graphique, cela est dû en partie à la gestion des interfaces par l'OS. La libraire SWT en fait partie, c'est pour cela que le seul moyen d'accéder à l'interface (pour changer l'évolution des barres de progressions, le texte des zones de saisies, etc.) est d'utiliser des méthodes de la classe Display (syncExec() et asyncExec()).

Par exemple, pour modifier le texte d'une zone de saisie, si nous utilisons ce code, une exception « Invalid thread access » sera lancée :

```
textBox.setText("Hello !");
```

Pour éviter cela, voici le code à utiliser :

```
Display.getDefault().asyncExec(new Runnable() {
        public void run() {
              textBox.setText("Hello !");
        }
});
```

Conclusion

La conception de ce client FTP m'a permis d'apprendre au mieux le langage Java en mettant en pratique ce que l'on a vu en cours. L'idée du client FTP est d'autant plus intéressante que le logiciel conçu pourra nous servir.