Création du Modèle Serveur Client et mise en place des Sockets

Le Serveur:

J'ai choisi de créer un Serveur qui peut facilement évoluer et qui gère de façon programmé les demandes du client.

Le Client:

Avec un client qui pourra facilement évoluer mais aussi interagir avec son serveur et le comprendre.

Modèle de données :

- Réutilisation d'un modèle de conception du « Gang Of Four » de type comportemental qui s'appelle « chaine of responsibility».
- Chaque Classe du modèle a un comportement qu'elle définit elle-même.
- Ce paterne permet d'avoir un modèle évolutif, qu'on peut représenter sous forme de chaîne et auquel on peut ajouter facilement de nouveaux maillons.
- Les maillions sont tous les enfants de « ChaineDeCommande.java », qui est une classe abstraite.
- Cette classe abstraite oblige ses enfants à utiliser des signatures de méthode abstract qu'ils doivent redéfinir. Autrement dit c'est une obligation d'avoir cette structure.
- On peut donc accéder aux attributs des classes enfants par la classe abstraite

API XStream :

J'ai trouvé une façon de créer du XML à partir de classe Java existante. Pour cela, il m'a fallu importer l'API XStream. Comme il faut la classe en référence pour créer le XML avec XStream, je me suis orienté vers la construction d'une classe générique.

Classe Générique :

- Utilisation d'une classe générique qui a pour but de fonctionner dynamiquement avec le modèle mise en place.
- Cette classe permet de prendre en paramètre une autre classe :

```
public class Serialisation<T extends ChaineDeCommande>
```

  Ici, il s'agit de toutes les classes enfants de « ChaineDeCommande.java ».
- Le principal avantage est d'avoir un code qui ne fait référence à aucune classe du modèle directement
- Elle a comme fonction de transformer les classes du modèle en XML par la sérialisation.
- Elle peut aussi les désérialiser.

Auteur : Thibaut Masselin

Interface de Thread utilisé :

Dans la messagerie, j'avais besoin de différents comportements de thread en fonction de « si je reçois » ou « si j'écris ».

Thread Runnable :

Cette interface de Thread ne renvoie rien (ni retour, ni exception), donc c'est celui que j'ai utilisé pour l'envoi de données, mais aussi pour le serveur, l'authentification et le chat serveur. Simple à créer et à gérer.

Thread Collable< ?> :

Cette interface de Thread renvoie un résultat ou une exception. Cette interface est une classe générique. Dans la messagerie, la classe « Reception.java » implémente cette interface qui elle-même utilise ma classe générique**. J'utilise cette classe pour pouvoir lire tous les XML que je réceptionne.**

```
public class Reception implements Callable<Serialisation<?>>{
```

ServerSocket :

Le serveur utilise un ServerSocket qui attend la connexion d'un client (avec une Socket) par un port d'écoute. Quand un client vient se connecter à lui, le ServerSocket accepte. On doit alors stocker la connexion dans une Socket, que l'on met ensuite dans un Thread pour permettre au serveur de pouvoir de nouveau attendre la connexion d'un autre client.

Socket :

Le client utilise une Socket qui lui permet, avec l'adresse IP et le port, de viser le ServerSocket pour établir une connexion entre eux deux.

Les Test Unitaires :

Je n'ai pu tester qu'à hauteur de 33% le code du serveur, c'est-à-dire une partie du modèle et la classe de sérialisation.

Mise en œuvre :

- Arrêt de l'exécution du client -> le serveur ferme la socket en rapport avec lui.
- Lecture du XML opérationnel.
- Ecriture du XML opérationnel.
- Connexion, Inscription sont opérationnel au niveau serveur et client.
- Envoie du message par le client.
- Le serveur réception et redirige les messages correctement.
- Réception du message par le client.
- Gère l'authentification du client -> renvoie soit un message d'erreur si l'utilisateur n'existe pas, soit l'utilisateur authentifié s'il existe.

Par un manque de temps, nous n'avons pas pu réaliser les options suivantes :

- Pouvoir tester à hauteur d'au moins 80% le code du serveur
- Finaliser le code du serveur pour l'optimiser aussi bien au niveau de la lecture qu'au niveau des traitements.
- Gérer tous les cas d'erreurs possibles.
- Mettre plus de fonctionnalités à la messagerie.
- Formater les champs date (daptpicker -> date (util) -> date (sql)).