12/12/2021

Projet JAVA

Machine à café

Rémi Maubanc & Yacine Ouyed
PROFESSEUR : M. DAILLY & MME ABY

Introduction

Le but du projet est de réaliser une machine à café avec une partie serveur et une partie client. Sera détaillé dans ce rapport, les détails de réalisations, les syntaxes pour la communication avec le serveur et les points de difficultés rencontrées pendant le développement. Bien que le projet soit fonctionnel, son application dans un contexte professionnel n'est pas possible dans l'état actuel. Le projet peut en revanche servir de démonstration technique.

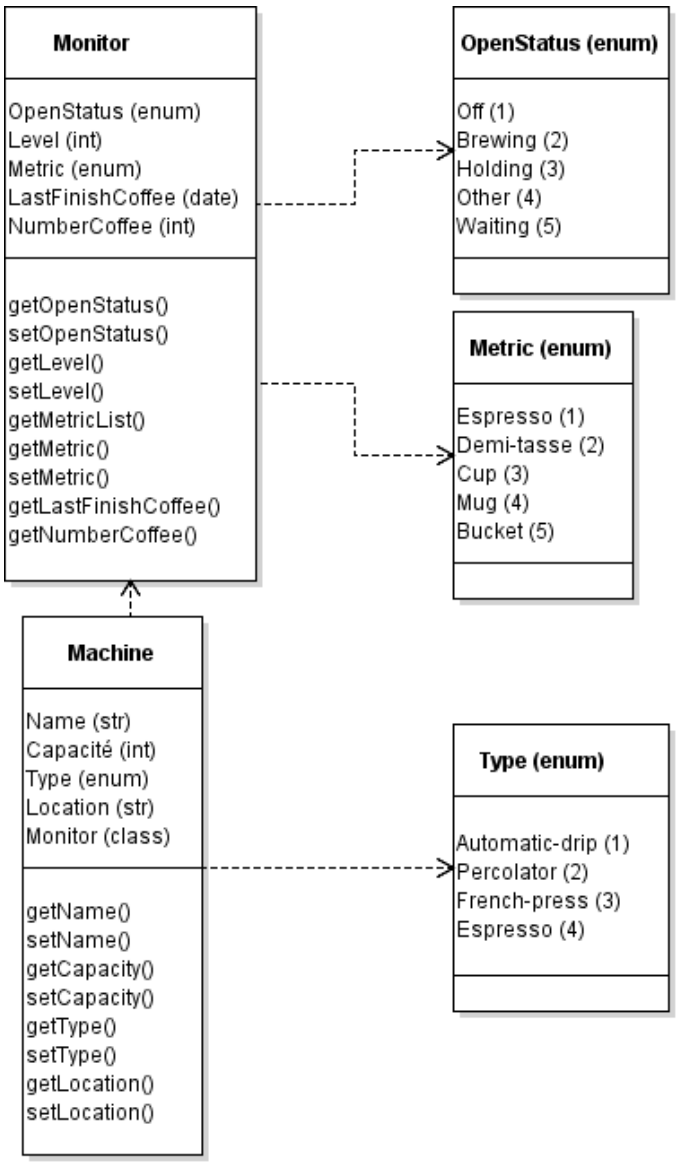
I/- Structure du projet

L'intégralité des sources du projet ont été hébergées et synchronisées grâce à GitHub. En effet, l'IDE IntelliJ prend en charge par défaut ce gestionnaire de version. Nous avons donc une branche par application du projet : *master* pour le serveur, *GUI_server* pour l'interface serveur et *GUI_client* pour l'interface client. Une présentation globale du projet est disponible en vidéo à cette adresse : https://youtu.be/2hBpXPttrkA

Serveur (API)

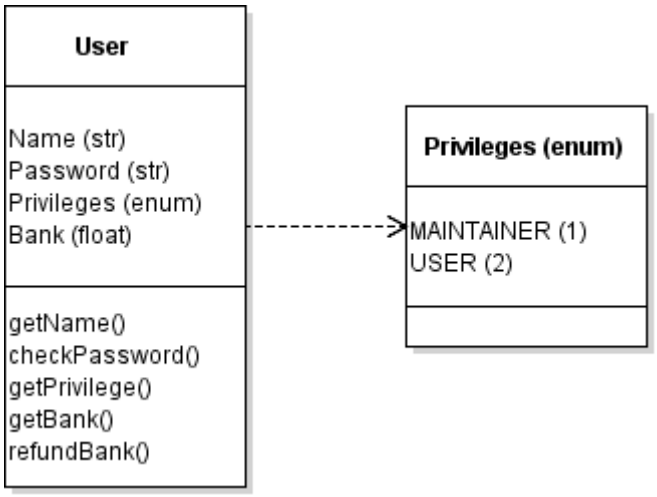
RFC

Dans le sujet du projet, le numéro d'une RFC à propos du fonctionnement hypothétique d'une machine à café ainsi que son protocole : la RFC 2325[1]. La première étape fut donc d'analyser la RFC pour en extraire son protocole pour ensuite le développer. Ici comme pour le reste du projet, l'intégralité des fonctions et structures du projet seront consignées sur papier avant d'être développé. C'est une manière de pouvoir partager les structures du code entre nous deux sans que l'un n'ait à lire le code de l'autre. Après lecture et analyse de la RFC, nous obtenons cette structure :



[1] https://datatracker.ietf.org/doc/html/rfc2325

On ajoute ensuite une classe pour nos utilisateurs et notre administrateur :

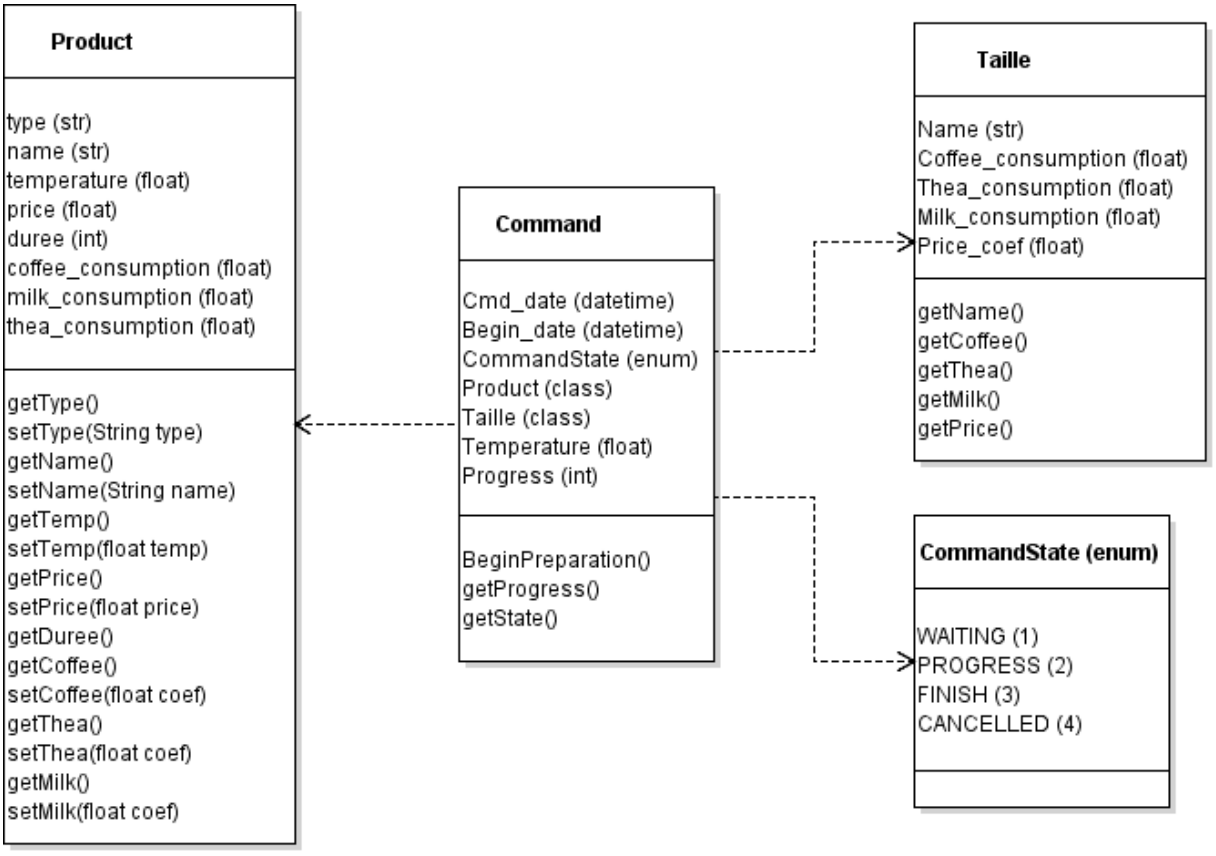


A partir de ce moment, nous commençons la programmation pour écrire les différentes classes, énumérateurs et leurs méthodes associées. Si le projet avait été plus long, nous aurions écrit des tests unitaires pour vérifier la validité des méthodes écrites et la gestion des erreurs y compris dans les cas exotiques. Cette vérification s'est donc faites à la volée au fur et à mesure du développement des classes.

Customisation

Evidemment, la RFC n'était pas totalement adaptée à la vision que l'on avait du projet demandé. Nous avons donc procédé à des modifications vis-à-vis de cette dernière.

- La première grosse modification est la réunion de la classe Machine et de la classe Monitor. Cette séparation entre les attributs des deux classes s'est révélée inutile en plus de complexifier le code en nécessitant des méthodes supplémentaires (à moins de rendre public les attributs). L'autre ajout concerne le stock de matière première directement dans la classe (deux attributs par type : Capacité, Stock restant).
- La deuxième modification est le remplacement de l'attribut « Metric » qui était un simple énumérateur par une classe complète. Ce qui permet de pouvoir choisir plus précisément sa commande. En effet, dans la RFC on ne peut choisir que le type de produit et l'énumérateur empêche toute modification (ajout, suppression, modification) pendant l'exécution. Cette modification en entraîne une série d'autre comme l'ajout d'une classe « Taille » pour pouvoir sélectionner sur sa commande la quantité que l'on veut avoir. Les attributs coefficients influent sur la consommation de la commande dans le stock de la machine.
- La dernière modification conséquente est l'ajout d'une classe « Command » qui associe chaque instance à une commande client. Elle dépend des classes « Taille » et « Produit ». Ainsi, la classe « Machine » n'est plus dépendante de « Taille » ou de « Produit ». En revanche c'est cette dernière qui gère la préparation des commandes non-préparées en fonction de leur date de création.



Pour la gestion des clients, nous avons utilisé le serveur multithread développé l'année dernière pendant la matière Système d'Exploitation, de même que la classe « IOCommand » pour la gestion des communications réseaux. Ce qui permet de pouvoir gérer les sessions clients en fonction des threads. Le serveur est en quelque sorte un serveur API qui ne possède pas d'interface graphique. Les interfaces graphiques serveur et clients sont des applications séparées ce qui évite qu'une modification du code de l'interface graphique puisse influer sur le comportement du serveur lui-même.

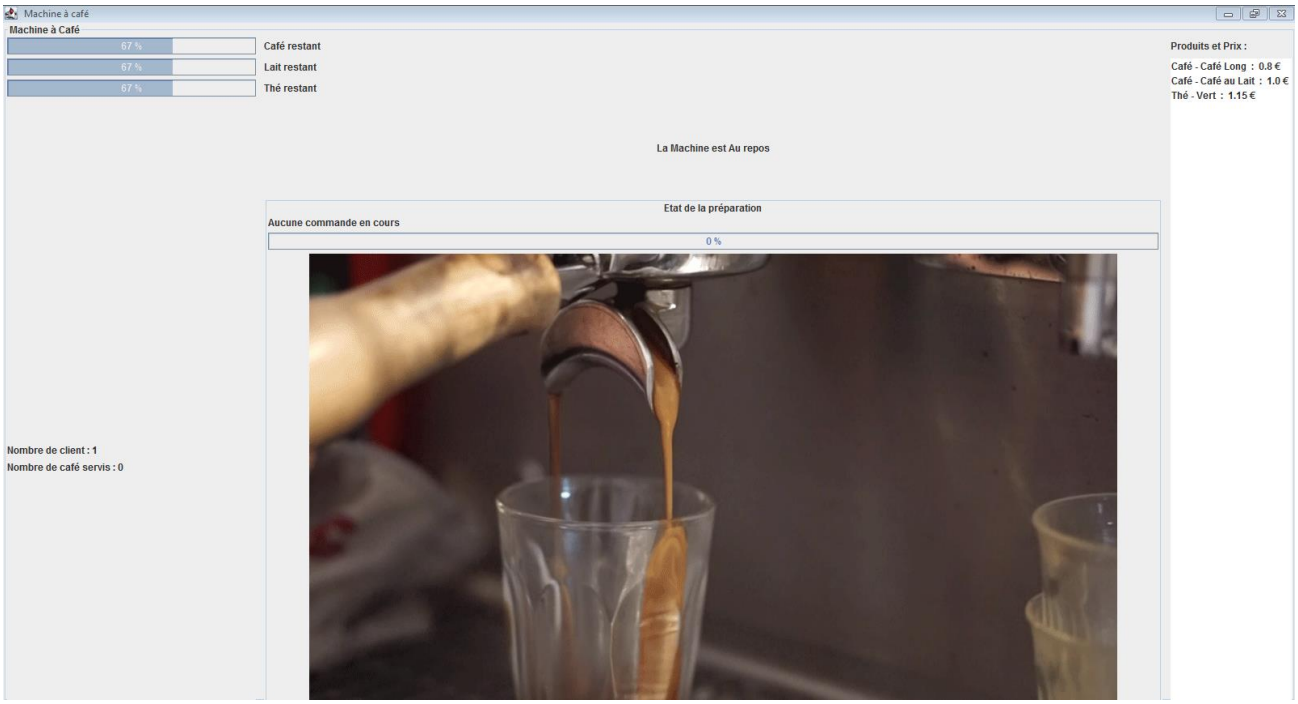
Interface Serveur

Cette interface a été conçue avec JFrame pour que le client puisse suivre sa commande et avoir des informations supplémentaires. Elle est composée de trois classes :

- IOCommandes : permet d'établir la communication avec le serveur
- Progress : permet de gérer les threads pour les barres de progression.
- Main : récupère et analyse les données à partir du serveur et contient la configuration de la fenêtre graphique.

L'interface serveur récupère les informations nécessaires avec des requêtes vers le serveur et les décodent (séparation des chaines de caractères reçus pour avoir l'information utile) permettant ainsi de :

- Pouvoir suivre la progression de la commande en cours grâce à une barre de progression et un Gif pour simuler le verre qui se rempli.
- Connaitre l'état de la machine et la quantité de café, de lait, de thé restante.
- Savoir combien d'autres clients sont connectés à la machine et combiens de cafés servis.
- Avoir une liste de tous les produits et le prix associé à chaque produit.

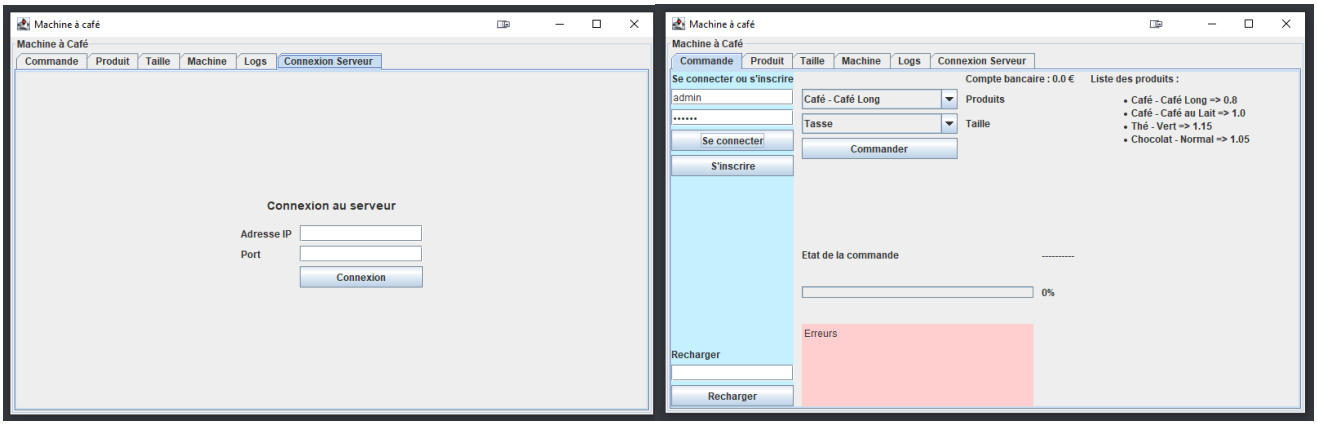


Interface Client

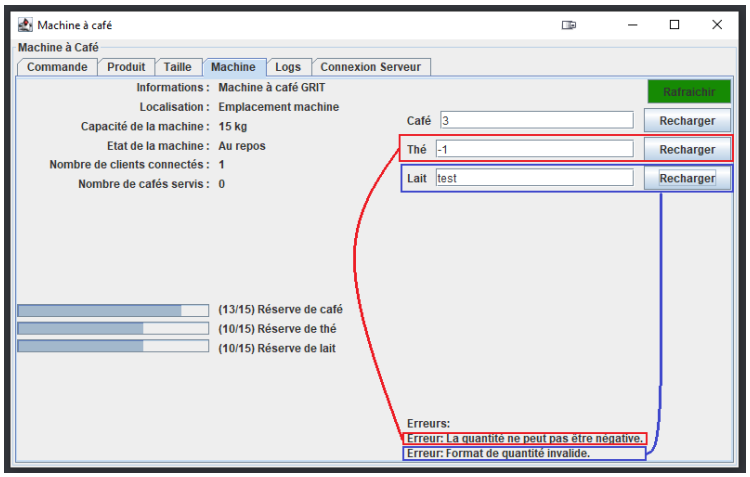
A l'instar de l'interface Serveur, l'interface Client est basé sur JFrame. L'interface client a été conçue pour pouvoir remplir les fonctions qu'un client dispose mais également pour l'administrateur. De cette manière, nous avons qu'une seule interface pour les deux types de compte. Pour réaliser cette séparation, nous utilisons des onglets un peu à la manière d'un navigateur.

Le client simple n'a que deux onglets utiles :

- L'onglet pour se connecter au serveur en renseignant l'adresse IP et le port.
- L'onglet pour gérer sa session, à savoir :
    - Se connecter ou créer un compte
    - Recharger son compte
    - Passer une commande et consulter son état.
    - Consulter les produits disponibles



L'administrateur quant à lui, utilise tous les onglets. Cependant à cause d'un manque de temps, seul l'onglet « Machine » est peuplé. De cet onglet, on peut recharger les stocks de la machine ainsi que consulter les informations à propos de cette dernière.



Comme pour l'interface client, nous avons une gestion des erreurs.

Syntaxe des commandes serveur

Pour pouvoir communiquer, nous avons établi une syntaxe qui liste toutes les commandes disponibles sur le serveur. On peut user de ces commandes avec une application CLI Java ou via un simple terminal Telnet. Cependant certaines commandes ne sont disponibles que si le client est connecté avec un compte administrateur. Autrement, les commandes renverront une erreur introuvable (pour éviter de donner des détails aux utilisateurs non-autorisés).

En gras la partie fixe ; en gris italique la partie dynamique.

Gestion des comptes

- Se connecter : **Login:***<name>*,*<pass>*
- Créer un compte : **Signup:***<name>*,*<pass>*
- Se déconnecter : **Logout**

Gestion du compte

- Recharger son compte : **Bank:***<value>*

Gestion des commandes

- Passer une commande : **Cmd:***<product_name>*,*<product_type>*,*<taille>*
- Avancement de la dernière commande (Interface client) : **Progress**
- Avancement de la commande en cours (Interface serveur) : **CmdStatus**

Gestion de la machine

- Etat de la machine : **MachineState**
- Arrêt de la machine (*admin only*) : **Poweroff**
- Recharge du stock (admin only) : **StockAdd:***<product>*,*<value>*
- Modification des capacités : **CapacityMod:***<name>*,*<quantity>*
- État des stocks : **StockGet**
- Nombre de clients : **ClientStat**
- Nombre de café servis : **CafeNb**

Gestion des produits

- Ajouter un produit (admin only) :
  ProductAdd:*<name(str)>*,*<type(str)>*,*<duree(int)>*,*<temp(float)>*,*<prix(float)>*,*<coffee(float)>*,
  <thea(float)>,*<milk(float)>*
- Modifier un produit (admin only) : **ProductModify:***<name>*,*<type>*,*<attribut>*,*<new_value>*
- Supprimer un produit (admin only) : **ProductRemove:***<name>*,*<type>*
- Liste des produits : **ProductList**

Gestion des tailles

- Ajouter une taille :
  TailleAdd:*<name(str)>*,*<coffee(float)>*,*<thea(float)>*,*<milk(float)>*,*<price(float)>*
- Supprimer une taille : **TailleRemove:***<name(str)>*

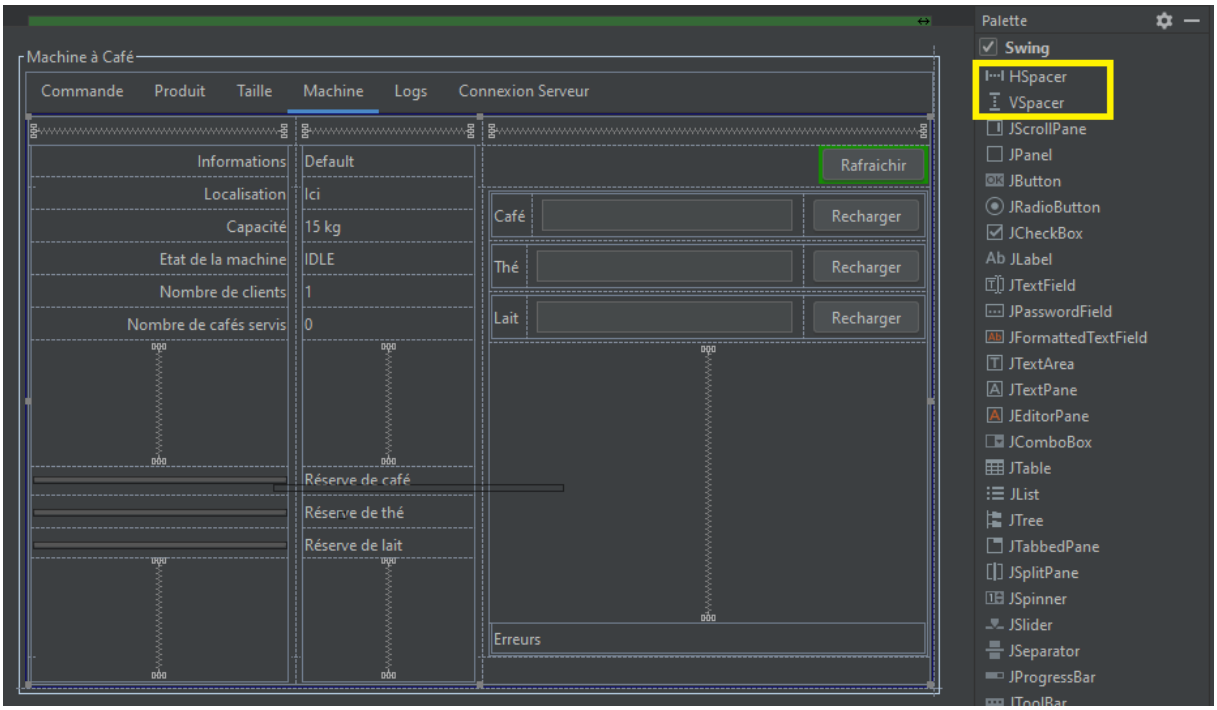
L'affichage des logs n'est pas encore disponible depuis les terminaux, mais un fichier « log.txt » est accessible à la racine de l'application serveur.

Difficultés rencontrées

Rémi Maubanc

Ma principale difficulté dans ce projet fut à propos des interfaces graphiques. De l'entièreté du domaine de la programmation, c'est la partie que j'apprécie le moins si ce n'est que je la déteste. C'est donc en toute logique que le développement de l'interface graphique du client fut la partie qui m'a opposée le plus de résistance.

L'un des points en particulier fut pour comprendre la manière de mettre en page les interfaces, au niveau des alignements. Après près de deux heures d'essais, j'ai pu rapprocher la mise en page avec les interfaces HTML/CSS et le principe de conteneur.



Chaque catégorie se trouve dans un JPanel. Et le nombre de JPanel semble parfois bien excessif pour obtenir le résultat obtenu, comme pour une page HTML avec le nombre de balise « div ».

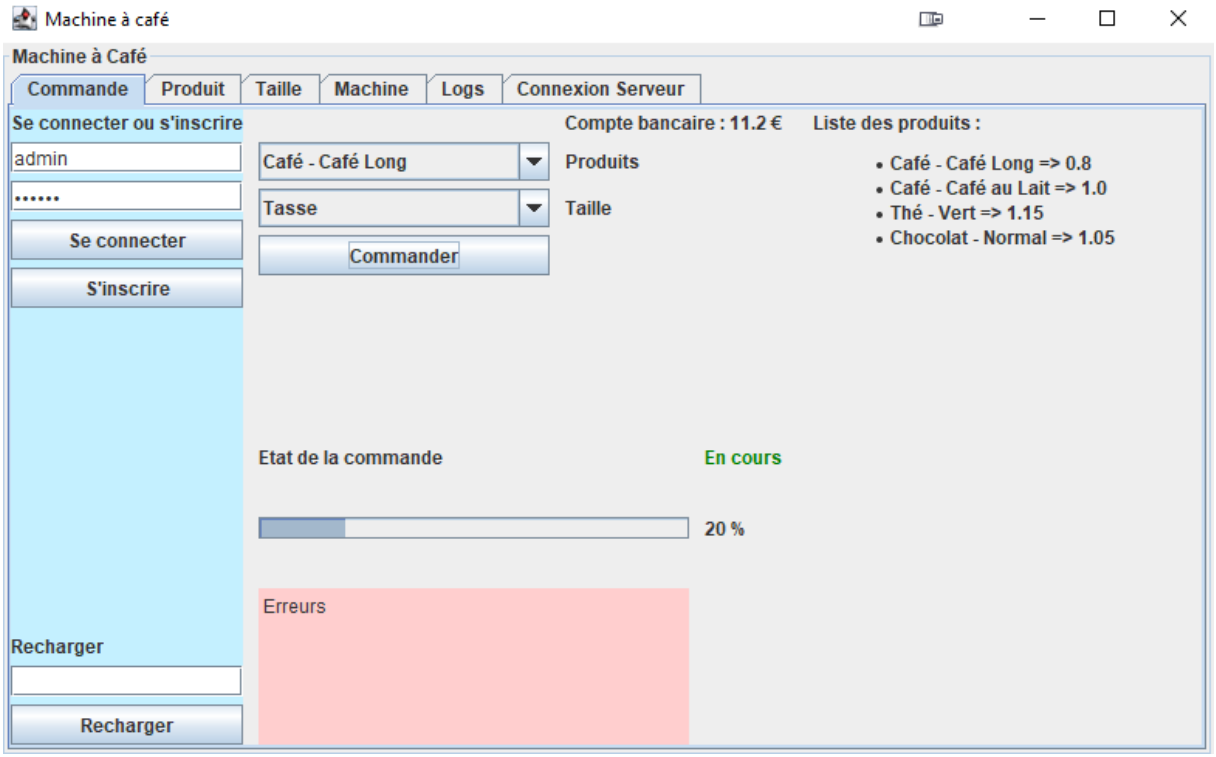
Yacine Ouyed

Lorsque j'ai débuté l'interface graphique du serveur, j'ai eu beaucoup de mal à comprendre le fonctionnement. Par exemple le moment de remplir la liste des produits (JList) que je remplissais à partir d'une liste de caractères mais à la finale il fallait une ListModel (conteneur de définitions ListElement).

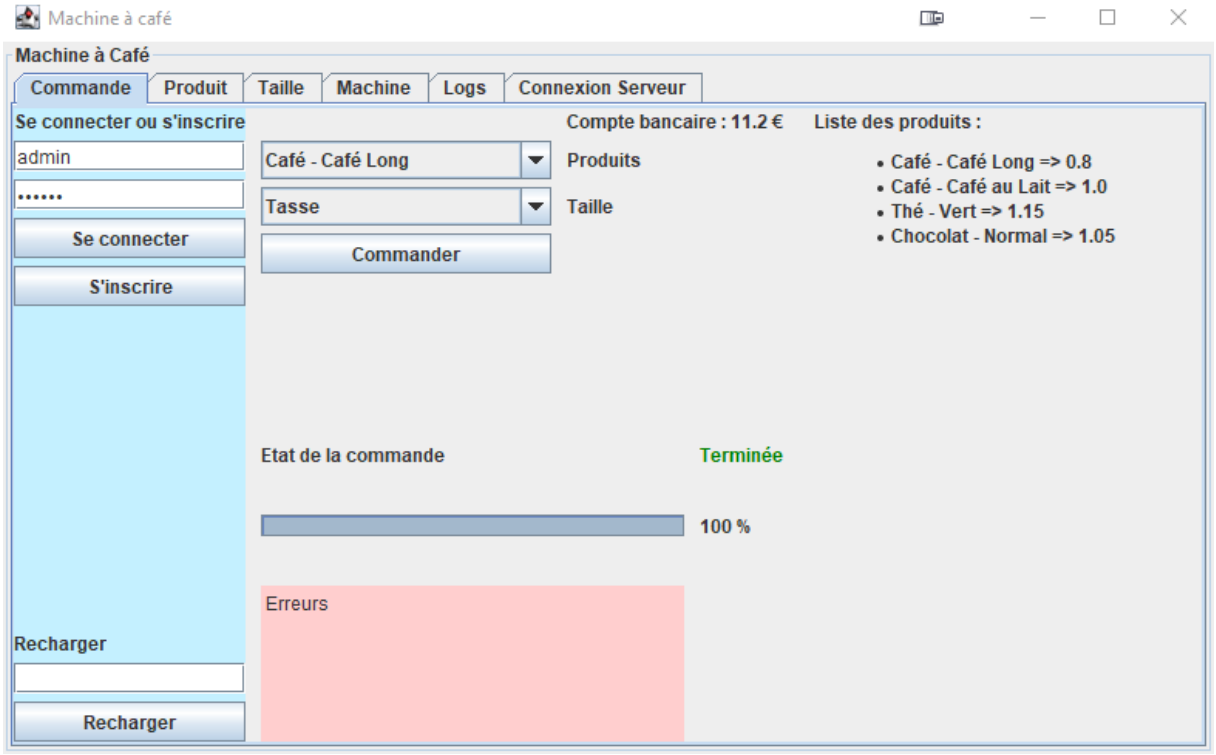
```
//Remplissage de produits
DefaultListModel produits = new DefaultListModel();
produits.addElement(listeProd[1] + " - " +listeProd[2] + "  :  " + listeProd[3]+ ' ' + '\u20AC' );
produits.addElement(listeProd[4] + " - " + listeProd[5] + "  :  " + listeProd[6]+ ' ' + '\u20AC');
produits.addElement(listeProd[7] + " - " + listeProd[8] + "  :  " + listeProd[9]+ ' ' + '\u20AC');
listeProduits.setModel(produits);
```

Au final grâce à Rémi et à toutes les difficultés rencontrées lors de ce projet j'ai pu apprendre beaucoup du langage Java et j'en apprendrai plus à l'avenir.

Annexes



Annexe 1 : Interface client - commande en cours



Annexe 2 : Interface client - commande terminée