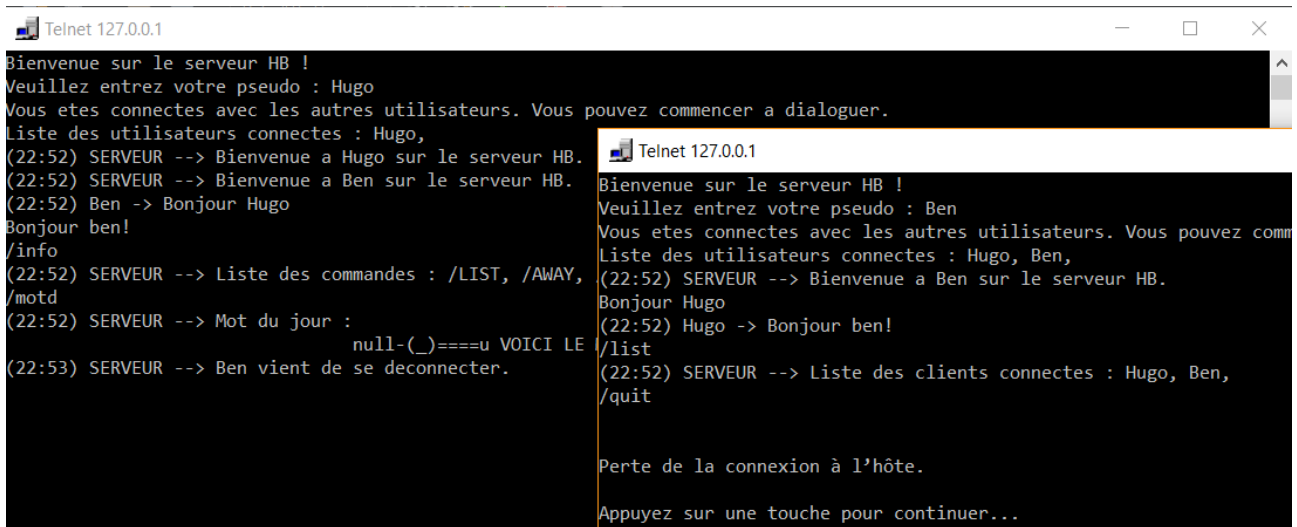


## Projet Réseau Java Groupe Hugo Da Roit, Benjamin Lévêque TD3



```
Telnet 127.0.0.1
Bienvenue sur le serveur HB !
Veuillez entrez votre pseudo : Hugo
Vous etes connectes avec les autres utilisateurs. Vous pouvez commencer a dialoguer.
Liste des utilisateurs connectes : Hugo,
(22:52) SERVEUR --> Bienvenue a Hugo sur le serveur HB.
(22:52) SERVEUR --> Bienvenue a Ben sur le serveur HB.
(22:52) Ben -> Bonjour Hugo
Bonjour ben!
/info
(22:52) SERVEUR --> Liste des commandes : /LIST, /AWAY,
/motd
(22:52) SERVEUR --> Mot du jour :
null-(_)====u VOICI LE
(22:53) SERVEUR --> Ben vient de se deconnecter.

Telnet 127.0.0.1
Bienvenue sur le serveur HB !
Veuillez entrez votre pseudo : Ben
Vous etes connectes avec les autres utilisateurs. Vous pouvez comm
Liste des utilisateurs connectes : Hugo, Ben,
(22:52) SERVEUR --> Bienvenue a Ben sur le serveur HB.
Bonjour Hugo
(22:52) Hugo -> Bonjour ben!
/list
(22:52) SERVEUR --> Liste des clients connectes : Hugo, Ben,
/quit

Perte de la connexion à l'hôte.
Appuyez sur une touche pour continuer...
```

Afin de réaliser le chat nous avons organisé notre code dans deux packages différents :

1. projetreseau qui contient les classes permettant de créer un serveur et de gérer les clients ainsi que les messages ;
2. projetreseau.commandes qui contient les classes représentant les commandes du chat (/quit, /info, /motd et /list).

Nous avons créé 11 classes différentes :

- ProjetReseau → permet de lancer le serveur
- Serveur → connexions avec les clients
- ShutdownHandler → afin de gérer une extinction propre du serveur
- ClientHandler → représente un client, permet d'écrire et de lire sur le Telnet du client
- BoiteAuxLettres → permet d'envoyer des messages aux clients
- CommandeHandler → pour traiter la commande que le client demande
- Commande (abstraite) → pour ensuite faire du polymorphisme
- CmdAway, CmdInfo, CmdList, CmdMotd → une classe par commande.

### Javadoc :

La javadoc est disponible dans le dossier javadoc, elle inclut toute la documentation sur les méthodes, classes et attributs. Le code est lui aussi commenté afin de faciliter la compréhension de celui-ci.

### Fonctionnement général de notre chat :

Pour lancer le chat il faut exécuter la classe ProjetReseau.java qui lancera le serveur. Nous avons respectés le protocole proposé dans le sujet.

La classe Serveur s'occupe d'instancier le socket pour le serveur et les sockets pour les clients. Pour cela il attend les connexions des clients et dès qu'un client se connecte il crée une instance de ClientHandler à laquelle il associe ce client (un Socket) puis on continue d'attendre un nouveau client. Cette instance est lancée dans un nouveau thread afin de pouvoir continuer à attendre les nouvelles connexions et ainsi pouvoir gérer plusieurs clients à la fois.

La classe ClientHandler possède un BufferedReader et un BufferedWriter afin de pouvoir lire et écrire sur le terminal du client. Cette classe possède une méthode (run) qui boucle tant que le client n'envoie pas le message « bye ». Dans cette boucle on récupère les entrées et on les envoie à la boîte aux lettres.

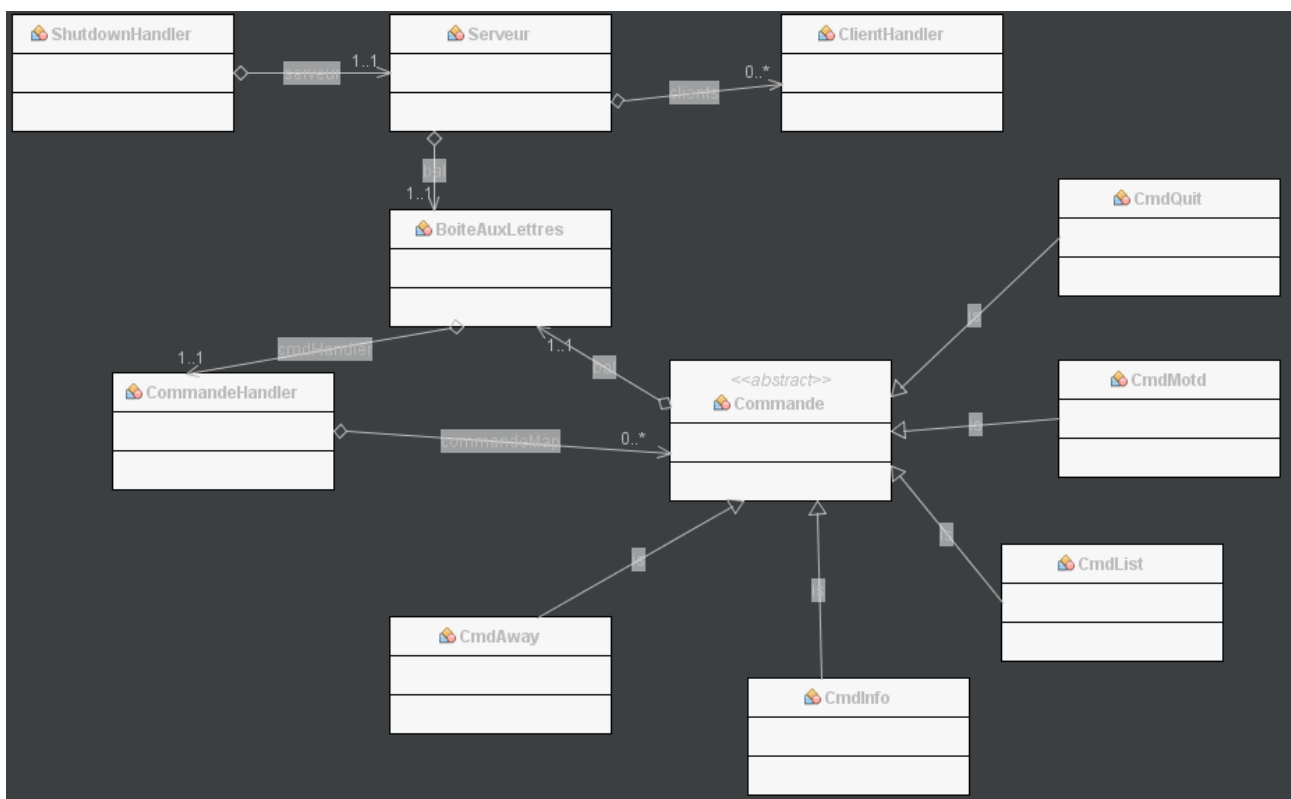
La classe BoiteAuxLettres possède des méthodes afin d'envoyer des messages aux clients avec le message en paramètre. Ces méthodes récupèrent la liste des clients (liste de ClientHandler) et utilise le BufferedWriter de chaque client pour envoyer les messages. Les méthodes d'envois sont « synchronized » afin de bien envoyer le message à tout les clients d'un seul coup pour s'assurer de l'envoi. En écrivant

directement dans chaque client via la BoiteAuxLettres nous n'avons pas besoin de le faire via les instances de ClientHandler. Cela nous évite aussi d'utiliser les méthodes wait et notify/all. Si un message commence par '/' alors c'est une commande du client, pour cela on va demander à un objet de type CommandeHandler de gérer cette commande.

La classe CommandeHandler possède un objet de type LinkedHashMap qui contient des clés associés à des valeurs. Les clés sont les noms des commandes et les valeurs des instances d'objet correspondant à la commande en question. Ainsi dès que l'on reçoit une requête on va chercher un objet via sa valeur et exécuter la méthode correspondante à la commande. Pour cela on va utiliser le polymorphisme via la classe Commande.

La classe Commande est abstraite et définit une méthode « exécuter » que les classes filles devront redéfinir. Les classes filles adaptent la méthode exécuter et renvoi une réponse que le client verra dans son terminal qui varie selon la commande.

Toutes les relations entre ces classes peuvent être modélisés via le diagramme de classe ci-dessous :



### Mode d'emploi :

Afin de se connecter il faut compiler toutes les classes puis lancer ProjetReseau. A partir de ce moment là le serveur fonctionne et tourne sur le port 2016.

Il faut ensuite s'y connecter via un logiciel tel que Telnet. Pour ce logiciel il suffit de taper : open 127.0.0.1 2016.

Une fois connecté il faut choisir son pseudo, puis on peut commencer à discuter avec les personnes connectés. La commande /info permet d'accéder à une liste des commandes existantes (list, quit, motd). Pour se déconnecter il est possible d'éteindre Telnet, mais on peut le faire proprement en tapant « bye » dans le chat ou en tapant /quit.