

Partie réseau du projet :

Server

```
ServerSocket servSock = new ServerSocket(port)
```

```
Socket sock = null;
```

```
Socket = servSock.accept();
```

```
DataInputStream in = new DataInputStream(sock.getInputStream());
```

```
DataOutputStream out = new DataOutputStream(sock.getOutputStream());
```

```
New clientHandler(port,in,out) => new Thread()
```

ClientHandler

```
Socket sock = new Socket(port) ;
```

```
DataInputStream in = new DataInputStream(sock.getInputStream());
```

```
DataOutputStream out = new DataOutputStream(sock.getOutputStream());
```

```
(functions between client and server like send date / time show numOfClients)
```

Server.java

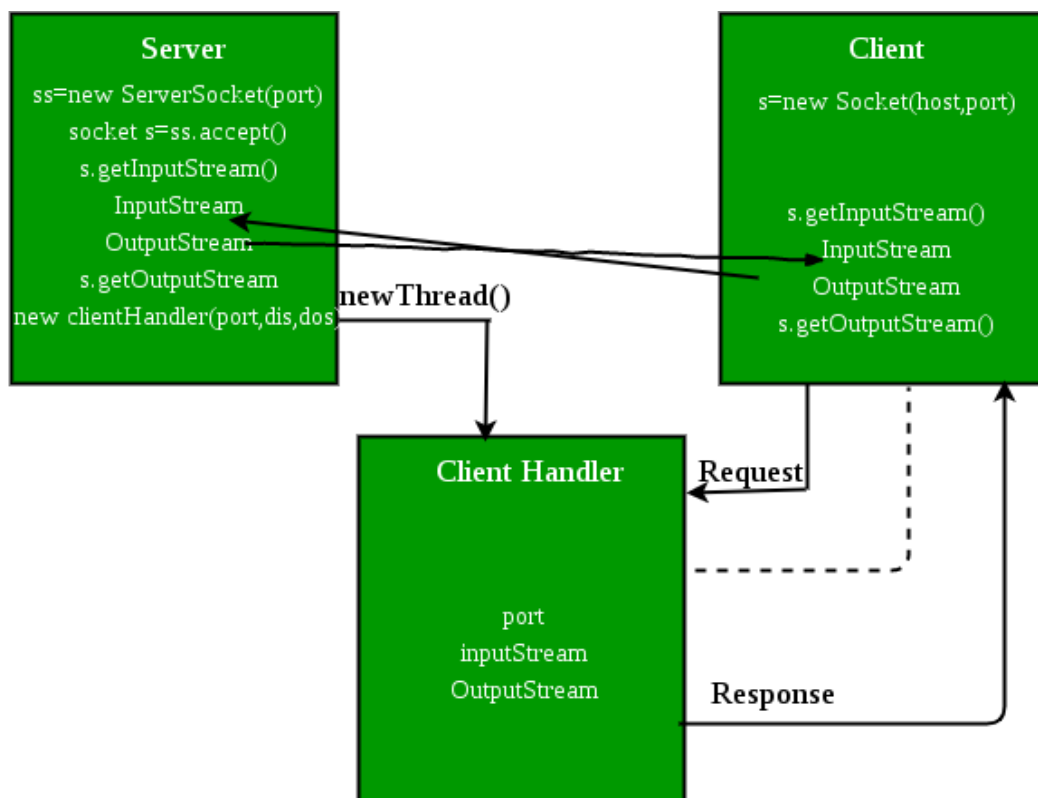
Client

```
Socket sock = new Socket(ip,port) ;
```

```
DataInputStream in = new DataInputStream(sock.getInputStream());
```

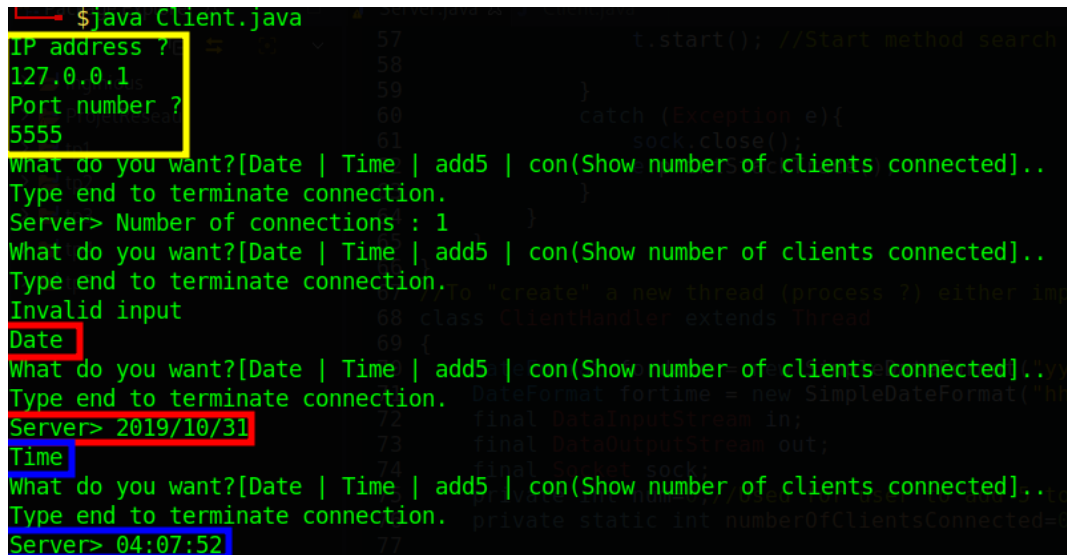
```
DataOutputStream out = new DataOutputStream(sock.getOutputStream());
```

Client.java



Amélioration (choses faites) :

- ✓ Le server peut accueillir plusieurs clients
- ✓ Le nombre de clients connectés est stockée dans une variable (numberOfConnectedClients)
 - S'il n'y a plus de clients (numberOfConnectedClients == 0)
 - ⇒ On ferme les ressources (in, out, port)
 - ⇒ Exit le programme.
 - S'il y a plus de 2 clients connectés (numberOfConnectedClients > 2)
 - ⇒ On ferme les ressources (...)
 - ⇒ Exit le programme.
- ✓ Le client peut demander la date / l'heure au serveur :



```
$ java Client.java
IP address ? 127.0.0.1
Port number ? 5555
What do you want?[Date | Time | add5 | con(Show number of clients connected)..
Type end to terminate connection.
Server> Number of connections : 1
What do you want?[Date | Time | add5 | con(Show number of clients connected)..
Type end to terminate connection.
Invalid input
Date
What do you want?[Date | Time | add5 | con(Show number of clients connected)..
Type end to terminate connection.
Server> 2019/10/31
Time
What do you want?[Date | Time | add5 | con(Show number of clients connected)..
Type end to terminate connection.
Server> 04:07:52
```

- ⇒ En jaune => Les arguments passés par l'utilisateur (ip, port)
- ⇒ En rouge => Demande la date au serveur et reçoit la réponse.
- ⇒ En bleu => Demande l'heure au serveur et reçoit la réponse.

A faire :

- ❖ Catcher quelques erreurs et masquer ce que l'utilisateur voit.
- ❖ Rendre plus beau ...
- ❖ Gérer la communication entre Thread => C'est-à-dire échanger des informations entre clients. Pour l'instant chaque client peut demander des infos aux serveurs mais pas aux autres clients.
 - Exemple : Dans le code, j'ai implémenté une fonction « add5 » qui ajoute 5 à une variable num.
 - Client A envoie add5 3 fois de suite (serveur répond 5,10,15) donc num est bien égale à 15.
 - MAIS quand le client B envoie add5, le serveur répond 5 et PAS 15.
 - Cela signifie que les variables ne sont partagées => Concurrency and Thread synchronizing.
- ❖ Voilà ma progression.

NOTE / IMPORTANT :

J'ai repris le code du super site : <https://www.geeksforgeeks.org/introducing-threads-socket-programming-java/>

Je l'ai modifié et documenté mais je ne vois franchement pas comment faire autrement. Du coup voilà.

Et aussi, faudra discuter entre nous, mais vos fonctions du genre tirer sur une case qui envoie une info et reçoit une autre info (genre tirer(x,y) et puis « vous avez touché une unité ») peuvent être implémentées dans la partie « fonction » (fluo bleu) du fichier Server.java.