Projet de Golfier Maxime, Lathaud Quentin, Majeune Nicolas

Scénario

I - Le client

Celui-ci est capable de soumettre une requête HTTP via un socket sur le port 80 il va se connecter au serveur web en lui demandant le fichier index.html.

Ce fichier sera retourné par le serveur et sera affiché dans une webview. Il est possible grâce à une zone de texte de saisir n'importe quelle adresse internet (qui sera formatée par notre programme en fonction de la manière dont l'user va taper sa recherche (http://www.monadresse.com ou simplement monadresse.com)

Une fois saisie et entrée, le client sera connecté de manière TCP au serveur. Concrètement dans notre programme le client est un singleton qui a pour méthode se connecter évitant ainsi la prolifération d'instance inutile.

```
public class Client {


    private static Client instance;

    protected Client () {}

    public static Client getInstance () {

        if(instance==null) {
            instance = new Client();
        }
        return instance;

    }
```

Dans notre méthode connexion nous utilisons un bufferedReader pour l'input, un printWriter pour l'outuput ainsi que les sockets pour établir la connexion.

```
public String connexion (String URL) {

    String message = "";
    String retour=System.getProperty("line.separator");

    try {

        URL aURL = new URL(URL);

        URL = aURL.getHost();

        Socket socketClient = new Socket(URL,80);

        BufferedReader  in = new BufferedReader(new InputStreamReader(socketClient.getInputStream(), "utf-8"));

        PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socketClient.getOutputStream())));

        out.println("GET /index.html HTTP/1.0\n");
        out.flush();

        message += "Information : Client connecté sur le serveur " + socketClient.getRemoteSocketAddress() + retour;

        message += "GET /index.html HTTP/1.0" + retour;

        while (in.readLine() != null) {
            message += in.readLine()+retour; // Afficher le message envoyé par le serveur
        }

        in.close();
        out.close();
        socketClient.close();
    }
    catch(IOException e) {
        message = "Whoops! ça marche Pas ! \n Message d'erreur : "+e.getMessage();
    }

    return message;
}
```

II - Le serveur

Le serveur a un constructeur composé d'une textarea (binding des informations pour pouvoir avoir pour l'administrateur les informations envoyés par le serveur) ainsi qu'une serveur socket.

```
public class Serveur extends Thread {

    private Socket socket;
        private PrintStream ps;

    public Serveur (Socket socketClient, TextArea ta){
        socket = socketClient;
                ps = new PrintStream(new Console(ta)) ;
        start();
    }
```

Le serveur dispose d'une methode run qu'on a commenté dans le code. (nous avons bien évidemment l'input, l'outuput et la socket d'utilisé ici). La réponse est traité dans une fonction à part entière.

```java
public void run(){
        //declaration de variable
        String reponse,mot[],separateur=" ";

        //initialisation permettant d'afficher la sortie standard et d'erreur sur la textArea grâce à la classe console
        System.setOut(ps);
        System.setErr(ps);

        try {

            // Flux des Messages d'entrée (en provenance du serveur)
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream(), "utf-8"));

            // Flux des Messages de sortie (à destination du serveur)
            PrintStream out= new PrintStream(socket.getOutputStream());

            // Lecture du message du client
            String messageClient = in.readLine();

            //partager le message reçu
            mot=messageClient.split(separateur);

            //traitement de ce message partagé par l'algorithme de traitement
            reponse = traitementError(mot);

            //affichage de la réponse avant son envoi
            System.out.println(reponse);

            // Envoi de la réponse
            out.println(reponse);

            //fermeture de la socket, message d'entrée et de sorties
            in.close();
            out.close();
            socket.close();

            } catch (IOException e) {
                e.getMessage();
            }
        }
```

Le serveur est capable de répondre à une requête HTTP envoyé via un socket sur le port 80, quand il reçoit un message d'un client il vérifie tout d'abord si la syntaxe de la requête http est correcte.

```
private String traitementError (String mot[]) throws IOException {
    //declaration
    String url,reponse;
    char c;
    int charInt;

    //analyse de la réponse
    if (mot[0].equals("GET") && mot[2].equals("HTTP/1.1")){

        url=mot[1];
        System.out.println(url+"\n");

        if (url.equals("/")) {
            url="index.html";
        } else{
            url=url.substring(1);
            url.replace("/", "\\");
        }

        try {

            File inputFile = new File(url);
            FileReader htmlFile=new FileReader(inputFile);
            reponse="HTTP/1.1 200 OK\nServer: test\nMime-Version: 1.0\nContent-Type: text/html\n\n";
            System.out.println(reponse);
            charInt=htmlFile.read();

            while (charInt!=-1){
                    c=(char)charInt;
                    reponse=reponse+c;
                    charInt=htmlFile.read();
            }

            htmlFile.close();
```

Si la syntaxe n'est pas le cas il renvoie une erreur 303 indiquant au client que sa requête est incorrecte.

Une fois que la requête est jugé acceptable le serveur va analyser l'adresse du fichier qui sera demandé :

➔Si aucun fichier n'est demandé (que l'url est un /) le serveur va retourner le fichier index.html par défaut, si l'url demande un fichier spécifique, le serveur va chercher dans son arborescence.

➔Si ce fichier existe si c'est le cas il va le retourner, si ce n'est pas le cas, il va retourner une erreur 404 qui informera le client que ce fichier n'existe pas.

```
        try {

            File inputFile = new File(url);
            FileReader htmlFile=new FileReader(inputFile);
            reponse="HTTP/1.1 200 OK\nServer: test\nMime-Version: 1.0\nContent-Type: text/html\n\n";
            System.out.println(reponse);
            charInt=htmlFile.read();

            while (charInt!=-1){
                    c=(char)charInt;
                    reponse=reponse+c;
                    charInt=htmlFile.read();
            }

            htmlFile.close();

        } catch(FileNotFoundException e){
            //affichage du message d'erreur pour le client
            reponse="HTTP/1.1 404 NOT FOUND\nServer: test\nMime-Version: 1.0\nContent-Type: text/html\n\n<html><body><h1>404
            //affichage du message d'erreur sur le programme
            System.out.println("Fichier non trouvé ! \n Message d'erreur : "+e.getMessage());
        }
    } else{
        //affichage du message d'erreur pour le client
        reponse="HTTP/1.1 300 HTTP Server Problem";
        //affichage du message d'erreur sur le programme
        System.out.println("Problème lié au serveur ! \n");
    }

    return reponse;
}
```

Pour récapituler : ce serveur est multi threding il supporte donc des connexions simultanées, un thread est créé par connexion.

Veuillez retrouver le scenario afin de voir l'interface graphique.