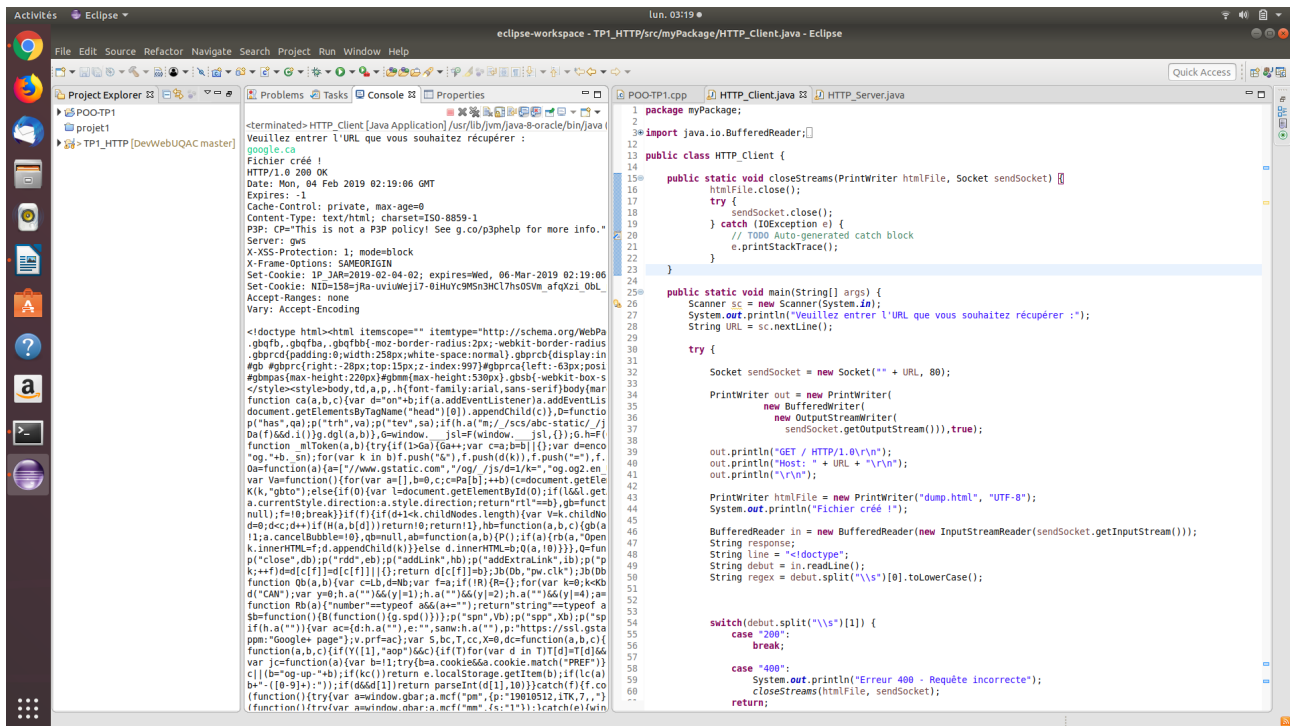


Trace d'exécution et Manuel d'utilisation

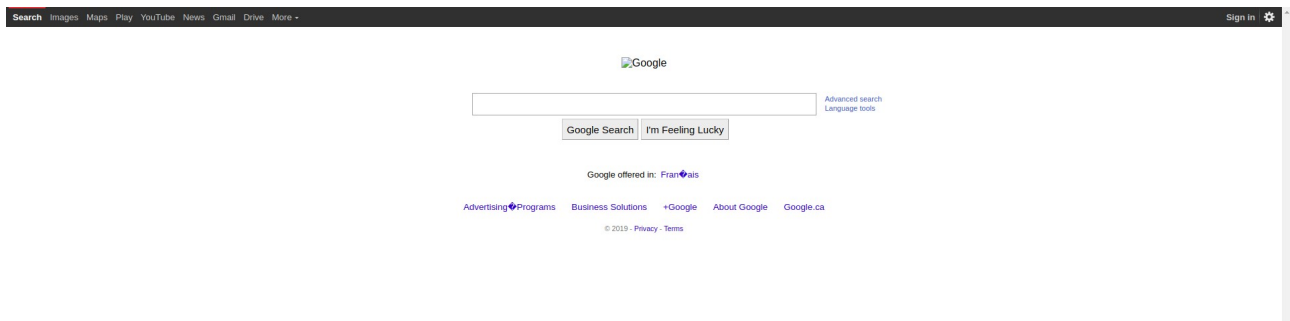
1) Trace d'exécution

1 – Client HTTP

Trace d'exécution avec le site google.ca :



Ouverture du fichier dump.html (fichier récupéré) :



2 – Serveur HTTP

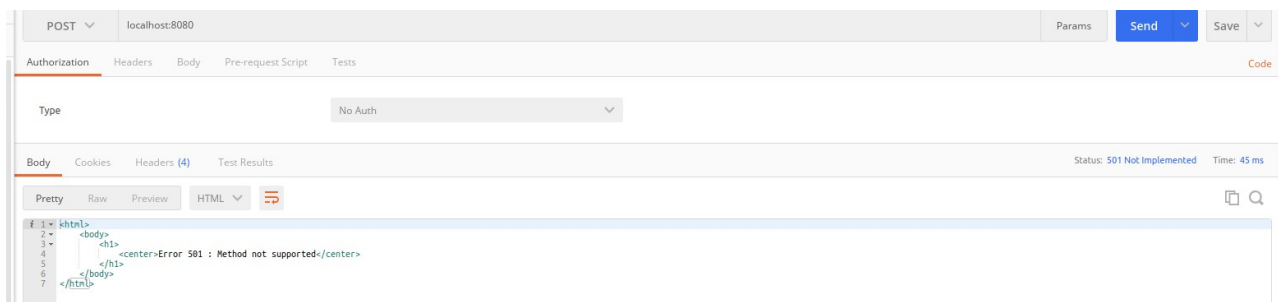
Après avoir lancé le serveur, trace d'exécution sur la route / :

Java HTTP Server

Trace d'exécution sur la route /test404 : (et sur toute les autres routes en méthode GET)

**Error 404 : File not found
Try again !!**

Trace d'exécution en méthode POST sur la route / :



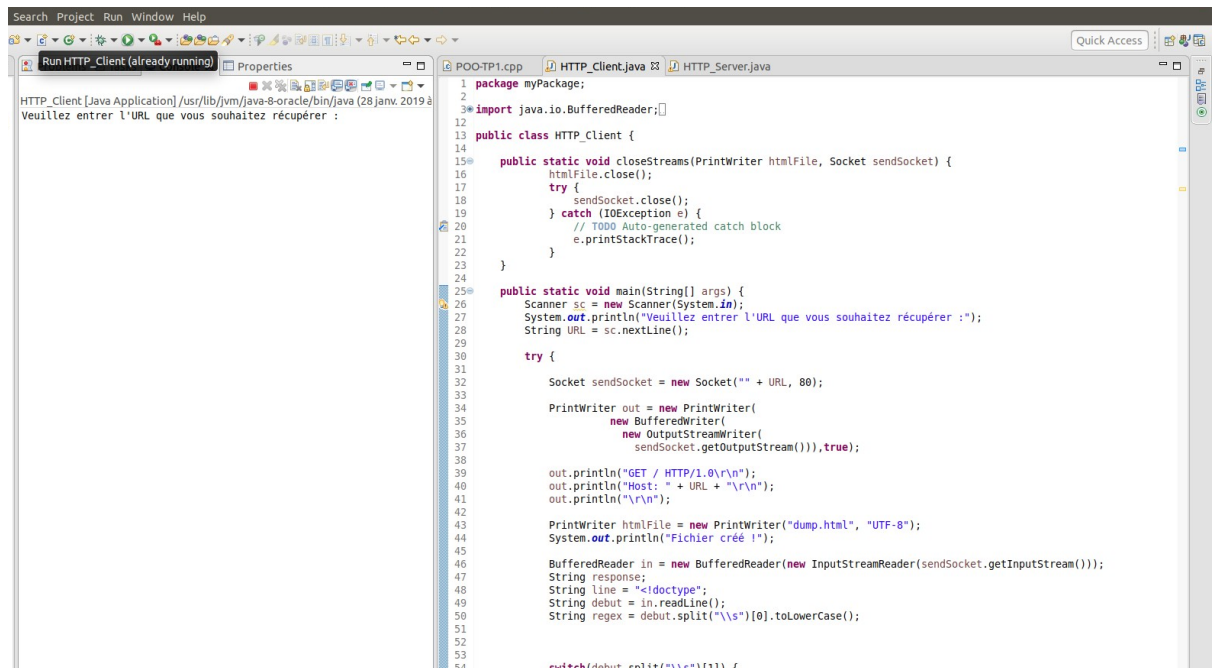
Nous avons bien une page d'erreur 501 – Method not supported. Avec un code d'erreur 501.

II) Manuel Utilisateur

1 – Client HTTP

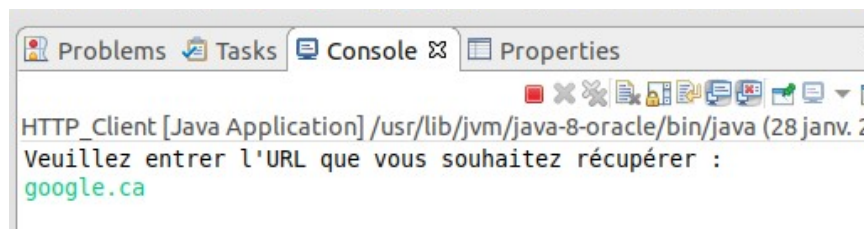
On souhaite ici récupérer la réponse d'un serveur web existant (nous prendrons l'exemple de google.ca dans cet exercice).

Il suffit alors de lancer notre programme :



```
1 package myPackage;
2
3 import java.io.BufferedReader;
4
5 public class HTTP_Client {
6
7     public static void closeStreams(PrintWriter htmlFile, Socket sendSocket) {
8         htmlFile.close();
9         try {
10             sendSocket.close();
11         } catch (IOException e) {
12             // TODO Auto-generated catch block
13             e.printStackTrace();
14         }
15     }
16
17     public static void main(String[] args) {
18         Scanner sc = new Scanner(System.in);
19         System.out.println("Veuillez entrer l'URL que vous souhaitez récupérer :");
20         String URL = sc.nextLine();
21
22         try {
23
24             Socket sendSocket = new Socket(" " + URL, 80);
25
26             PrintWriter out = new PrintWriter(
27                 new BufferedWriter(
28                     new OutputStreamWriter(
29                         sendSocket.getOutputStream()), true);
30
31             out.println("GET / HTTP/1.0\r\n");
32             out.println("Host: " + URL + "\r\n");
33             out.println("\r\n");
34
35             PrintWriter htmlFile = new PrintWriter("dump.html", "UTF-8");
36             System.out.println("Fichier créé !");
37
38             BufferedReader in = new BufferedReader(new InputStreamReader(sendSocket.getInputStream()));
39             String response;
40             String line = "<doctype>";
41             String debut = in.readLine();
42             String regex = debut.split("\n")[0].toLowerCase();
43
44             switch(debut.split("\n")[1]) {
45
46
47
48
49
50
51
52
53
54 }
```

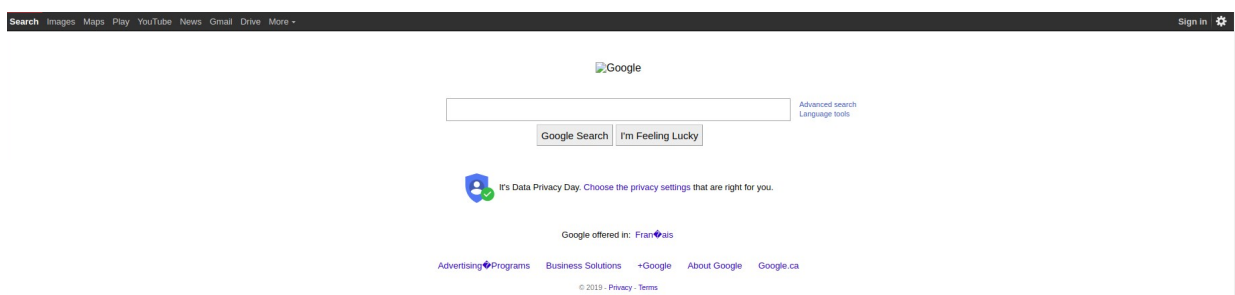
Et d'entrer l'URL du serveur sur lequel nous souhaitons effectuer notre requête, ici google.ca.



Notre programme affiche alors l'ensemble des lignes du fichier HTML récupéré, en commençant par le header de la réponse. De plus, nous enregistrons le document renvoyé par le serveur dans un fichier « dump.html ».

Dans le cas où la réponse n'est pas OK (Code 200), le message est personnalisé selon le code d'erreur et un header « propre » est renvoyé au client.

On peut alors afficher notre fichier dump.html sur un navigateur afin de vérifier que nous avons le bon résultat :



Si nous essayons un lien qui devrait nous renvoyer une erreur, par exemple [linkedin.com](https://www.linkedin.com).

```
<terminated> HTTP_Client [Java Application] /usr/lib/jvm/java-8-oracle/bin/java
Veuillez entrer l'URL que vous souhaitez récupérer :
linkedin.com
Fichier créé !
Erreur 400 - Requête incorrecte
```

Nous avons bien une erreur « propre ».

2 – Serveur HTTP

L'objectif ici est de mettre en place un serveur HTTP, qui renvoie une page HTML selon la requête. Pour simplifier l'exercice, nous considérons trois types de requêtes ici :

- Les requêtes GET sur le chemin à la racine : /. Dans ce cas, nous renvoyons notre page index.html, la requête se passe bien.
- Les requêtes GET sur les autres chemins. Dans ces cas là, nous renvoyons une page d'erreur 404, car seul la page index.html existe.
- Les autres requêtes, sur l'ensemble des routes possibles. Nous renvoyons dans ce cas une page précisant le code d'erreur 501 – Not Supported.

Ici il suffit de lancer notre serveur (depuis Eclipse dans notre cas) :

```
Server started.
Listening for connections on port : 8080 ...
```

Nous utilisons ici le port 8080, mais il est facilement modifiable dans le programme.

Nous pouvons alors nous rendre sur le serveur : localhost:8080.

Java HTTP Server

Nous observons bien notre page index.html. On remarque également sur eclipse que notre serveur nous signale la requête qui a eu lieu :

```
HTTP_Server [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (28 janv. 2019)
Server started.
Listening for connections on port : 8080 ...

Connection opened @ Mon Jan 28 21:23:55 CET 2019
Connection opened @ Mon Jan 28 21:23:55 CET 2019
Method : GET.
File requested : /
File /index.html of type text/html returned.
Connection closed.
```

Si nous allons sur une autre page que la racine, nous arrivons bien sur une page d'erreur 404 :

Error 404 : File not found
Try again !!

Enfin, pour essayer nos requêtes d'un autre type que GET, nous avons utilisé l'extension de google chrome : Postman. Voilà ce que nous avons avec une requête POST par exemple :