Les sockets TCP en Java

Exercice 1 : Client web

Dans cet exercice, il vous est demandé d'écrire un client réseau capable d'envoyer une requête http minimalise à un serveur web et d'en afficher la réponse dans une console. Pour ce faire, vous devrez mettre en œuvre une classe *ClientWeb* qui possède la déclaration partielle suivante :

```
public class WebClient {
    public WebClient(String server, int port);
    public void connect();
    public void close();
    public void sendRequest(String request);
    public void printResponse();
}
```

La requête minimale HTTP est définie par la chaine de caractère suivante : « GET / http 1.1 \n\n ».

```
package fr.utt.sit.lo02.finaux.A09;

import java.io.*;
import java.net.*;

public class WebClient {

    private String server;
    private int port;
    private Socket socket;

    public WebClient(String s, int p) {
      this.server = s;
      this.port = p;
    }

    public void connect() throws IOException {
      InetAddress ia = InetAddress.getByName(server);
      socket = new Socket(ia, port);
      System.out.println(socket);
    }

    public void close() throws IOException {
      socket.close();
      System.out.println("Connection closed.");
    }

    public void sendRequest(String request) throws IOException {
      OutputStream os = socket.getOutputStream();
      BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(os));
      bw.write(request);
      bw.flush();
    }

    public void printResponse() throws IOException {
      InputStream is = socket.getInputStream();
      BufferedReader br = new BufferedReader(new InputStreamReader(is));

      String line = null;

      do {
          line = br.readLine();
          System.out.println(line);
```

```
      } while (line != null);
    }

    public static void main (String[] args) {

      WebClient wc = new WebClient("www.utt.fr", 80);

      try {
      wc.connect();

      wc.sendRequest("GET / HTTP 1.0\n\n");

      wc.printResponse();

      Thread.sleep(500);

      wc.close();
      } catch (Exception e) {
          e.printStackTrace();
      }
    }
}
```

Exercice 2 : Un micro serveur web (*extrait du final A09*) – à faire en séance et terminer en THE

On propose ici de mettre en œuvre un serveur web dont les fonctionnalités seront minimales. Son principe de fonctionnement est le suivant : Le serveur web est une application Java qui écoute en permanence sur le port TCP 8080.

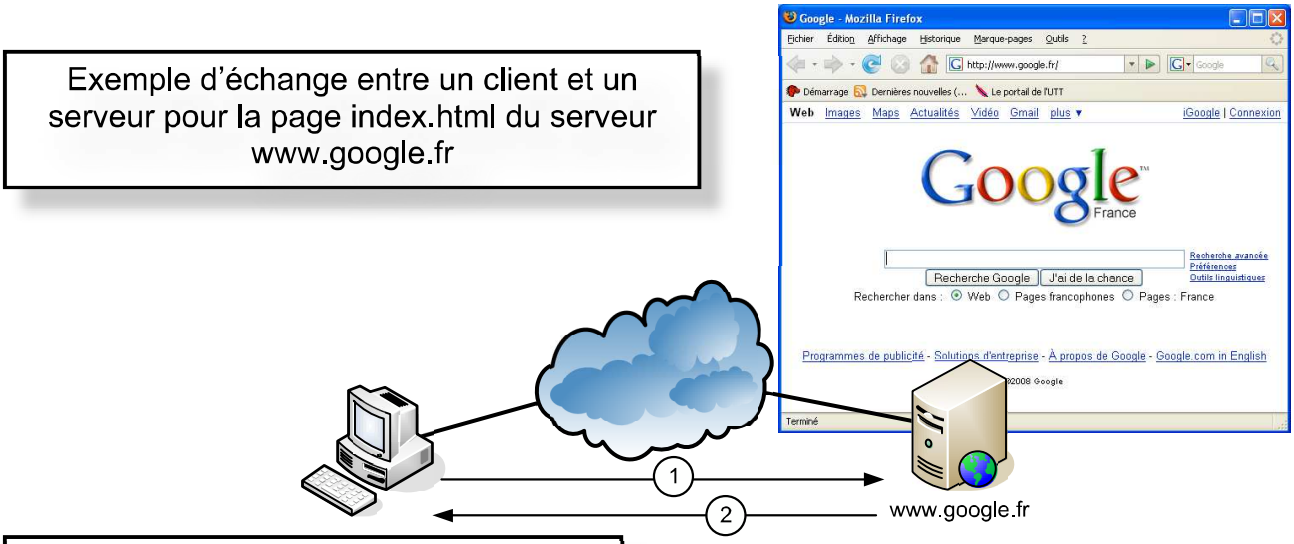
- Lorsqu'il reçoit une requête sur ce port, il vérifie que le protocole de la requête est bien http.
  - Si c'est le cas, il récupère dans la requête le nom de la page web à laquelle un client veut accéder.
  - Si ce n'est pas le cas, il ferme le socket et lève une exception qui affiche une erreur sur la console.
- Le serveur vérifie ensuite l'existence du fichier correspondant à la page web à laquelle le client veut accéder.
  - Si le fichier existe, il construit une réponse suivant le protocole http et place le contenu du fichier dans cette réponse. L'ensemble est ensuite envoyée au client à travers le socket ouvert.
  - Si le fichier n'existe pas, il construit une réponse suivant le protocole http dans laquelle il indique que la page n'existe pas sur ce serveur.

Il existe donc trois messages qui peuvent être échangés entre un client et le serveur web :

Message	Contenu du message http	Description
Requête du client pour une page web	`GET index.html HTTP/1.1`	La requête indique : - GET : requête sur une page - index.html : le nom du fichier de la page web - http/1.1 : le protocole utilisé pour la requête

Réponse du serveur avec la page requise	`HTTP/1.1 200 OK`
	`// contenu de la page`
	`<HTML>`
	`  <HEAD>`
	`  <title>Google</title>`
		La première ligne de la réponse indique :
		- http/1.1 : le protocole utilisé par la réponse
		- 200 OK : code qui indique que la page existe
		Les lignes suivantes sont le contenu du fichier index.html

Réponse du serveur : page non trouvée	`HTTP/1.1 404 Not found`	Cette réponse indique : - http/1.1 : le protocole utilisé pour la réponse - 404 Not found : code qui indique que la page n'existe pas sur ce serveur



Questions : Les questions portent sur la construction étape par étape de ce serveur

1. Ecrire une classe *MessageHTTP* qui représente un message http. Utiliser l'héritage pour définir les classes *RequêteHTTP* et *ReponseHTTP*. Chaque classe, à travers ses attributs, représente l'ensemble des informations stockées dans les messages http.

```
package fr.utt.sit.lo02.finaux.A09;

public abstract class HTTPMessage {

    public final static String SUPPORTED_HTTP_VERSION = "HTTP/1.1";
    public final static String SEPARATOR = " ";

    protected String version;

    public String getVersion() {
      return version;
    }

    public void setVersion(String version) {
      this.version = version;
    }
}
```

2. Dans la classe *RequeteHTTP*, écrire la méthode *public static RequêteHTTP lireRequête(InputStream is)*. Cette méthode reçoit en paramètre un flux binaire en lecture pour construire une instance de requête http.

```java
package fr.utt.sit.lo02.finaux.A09;

import java.io.*;

public class HTTPRequest extends HTTPMessage {

    public final static String GET_TYPE = "GET";

    private String type;
    private String file;

    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public String getFile() {
        return file;
    }
    public void setFile(String file) {
        this.file = file;
    }

    public static HTTPRequest buildRequest(InputStream is) {

      InputStreamReader isr = new InputStreamReader(is);
      BufferedReader br = new BufferedReader(isr);
      String line;
      try {
          line = br.readLine();

            HTTPRequest hr= new HTTPRequest();

            String[] tokens = line.split("\\s");

            if (tokens.length == 3) {
                  hr.setType(tokens[0]);
                  hr.setFile(tokens[1]);
                  hr.setVersion(tokens[2]);
            } else {
                System.err.println("Malformed request.");
                return null;
            }

            boolean endRequest = false;

            while (endRequest == false) {
                line = br.readLine();
                if (line.equals("")) {
                  endRequest = true;
                }
            }

            return hr;

      } catch (IOException e) {
          System.err.println(e.getMessage());
          return null;
      }
    }
```

```
    public String toString() {
      StringBuffer sb = new StringBuffer();
      sb.append(this.type);
      sb.append(HTTPMessage.SEPARATOR);
      sb.append(this.file);
      sb.append(HTTPMessage.SEPARATOR);
      sb.append(super.version);

      return sb.toString();
    }
}
```

3. Dans la classe *ReponseHTTP*, écrire la méthode *public void ecrireReponse (OutputStream os)* qui permet d'envoyer une réponse http dans un flux binaire en écriture.

```
package fr.utt.sit.lo02.finaux.A09;

import java.io.*;
import java.util.*;

public class HTTPResponse extends HTTPMessage {

    public final static String OK_CODE = "200";
    public final static String NOTFOUND_CODE = "404";
    public final static String OK_MESSAGE = "OK";
    public final static String NOTFOUND_MESSAGE = "Not Found";

    private ArrayList<String> content;
    private String code;
    private String message;

    public HTTPResponse() {
      content = new ArrayList<String>();
    }

    public String getCode() {
      return code;
    }

    public void setCode(String code) {
      this.code = code;
    }

    public String getMessage() {
      return message;
    }

    public void setMessage(String message) {
      this.message = message;
    }

    public ArrayList<String> getContent() {
      return content;
    }

    public void addContent(String line) {
      content.add(line);
    }

    public void writeResponse(OutputStream os) {

      OutputStreamWriter osw = new OutputStreamWriter(os);
```

```
      BufferedWriter bw = new BufferedWriter(osw);

      String header = super.version + HTTPMessage.SEPARATOR + this.code +
HTTPMessage.SEPARATOR + this.message;

      try {
          bw.write(header);
          bw.newLine();
          bw.newLine();

          Iterator<String> it = content.iterator();

          while (it.hasNext()) {
            String line = it.next();
            if (line != null) {
                bw.write(line);
                bw.newLine();
            }
          }

          bw.newLine();
          bw.flush();

      } catch (IOException e) {
          System.err.println(e.getMessage());
      }
    }

    public String toString() {
      StringBuffer sb = new StringBuffer();
      sb.append(super.version);
      sb.append(HTTPMessage.SEPARATOR);
      sb.append(this.code);
      sb.append(HTTPMessage.SEPARATOR);
      sb.append(this.message);

      return sb.toString();
    }
}
```

4. Ecrire la classe *TraitementRequete*, thread traitant une requête sur le serveur. Son constructeur reçoit en paramètre un socket qui permet de dialoguer avec le client pour récupérer sa requête et lui envoyer la réponse.

```
package fr.utt.sit.lo02.finaux.A09;

import java.io.*;
import java.net.Socket;

public class RequestProcessor implements Runnable {

    private static long REQUEST_NUMBER = 0;

    private Socket socket;

    public RequestProcessor (Socket s) {
      this.socket = s;
    }

    public void process() {
      Thread t = new Thread(this);
      RequestProcessor.REQUEST_NUMBER++;
```

```
      t.start();
    }

    public void run() {
      try {
          System.out.println("Request processor started...");
          System.out.println("Request #" +
RequestProcessor.REQUEST_NUMBER);
          System.out.println("Client: " +
socket.getInetAddress().getHostAddress());

          HTTPRequest hReq =
HTTPRequest.buildRequest(socket.getInputStream());
          System.out.println("Request: " + hReq);

          if (hReq.getVersion().equals(HTTPMessage.SUPPORTED_HTTP_VERSION))
{

            if (hReq.getType().equals(HTTPRequest.GET_TYPE)) {

                File file = new File(WebServer.HTTP_FILES_FOLDER +
hReq.getFile());

                HTTPResponse hRsp = new HTTPResponse();
                hRsp.setVersion(HTTPMessage.SUPPORTED_HTTP_VERSION);

                if (file.exists()) {
                  System.out.println("Found file: " + file);

                  hRsp.setCode(HTTPResponse.OK_CODE);
                  hRsp.setMessage(HTTPResponse.OK_MESSAGE);

                  FileReader fr = new FileReader(file);
                  BufferedReader br = new BufferedReader(fr);

                  String line = null;

                  do {
                      line = br.readLine();
                      hRsp.addContent(line);
                  } while (line != null);

                  fr.close();

                } else {
                  System.out.println("Not found file: " + file);

                  hRsp.setCode(HTTPResponse.NOTFOUND_CODE);
                  hRsp.setMessage(HTTPResponse.NOTFOUND_MESSAGE);
                }

                System.out.println("Response: " + hRsp);

                hRsp.writeResponse(socket.getOutputStream());

            } else {
                System.err.println("Unsupported request type: " +
hReq.getType());
            }
          } else {
            System.err.println("Unsupported protocol version: " +
hReq.getVersion());
          }
```

```
            socket.close();
        } catch (IOException e) {
            //System.err.println(e.getMessage());
            e.printStackTrace();
        }

        System.out.println("Request processor stopped...\n");

    }
}
```

5. Ecrire la classe *ServeurWeb*, thread qui écoute sur le port TCP 80. Pour chaque nouvelle connexion, le serveur crée une instance de *TraitementRequête* et se replace en écoute.

```
package fr.utt.sit.lo02.finaux.A09;

import java.io.IOException;
import java.net.*;

public class WebServer implements Runnable {

    public final static String HTTP_FILES_FOLDER = "./";

    private boolean stop;
    private int port;

    ServerSocket sSocket;

    public WebServer(int port) {
      this.stop = true;
      this.port = port;
    }

    public void run() {
      stop = false;

      System.out.println("Web server started...");

      try {
          sSocket = new ServerSocket(this.port);

          System.out.println("Listening on port " + this.port + "...\n");

          while (stop == false) {
            Socket s = sSocket.accept();
            RequestProcessor rq = new RequestProcessor(s);
            rq.process();
          }

      } catch (IOException e) {
          System.err.println(e.getMessage());
      }
      System.out.println("Web server stopped...");
    }

    public void stopServer() {
      stop = true;
      try {
          sSocket.close();
      } catch (IOException e) {
          System.err.println(e.getMessage());
```

```
      }
    }

    public void startServer() {
      Thread t = new Thread(this);
      t.start();
    }

    public static void main(String[] args) {
      WebServer ws = new WebServer(8080);
      ws.startServer();
    }
}
```