

Auteur

ADOU KOUADIO AIME TRESOR

SORO ANGE SONGMON MIGUEL

**Objectif**

**Vous devez développez en équipe de deux un protocole et un serveur de fichiers qui permettent à des clients de téléchargez des fichiers. Les fichiers transférables sont stockés dans un répertoire spécifié dans la configuration du serveur.**

**Protocole de transfert de fichiers**

* **Un client peut demander au serveur la liste des fichiers hébergés**
* **Le serveur retourne la liste des fichiers hébergés**
* **Le client peut indiquer le téléchargement d’un ou plusieurs fichiers de cette liste avant de terminer la session.**
* **Plusieurs clients peuvent ouvrir une session simultanément sur le serveur.**

**Langages**

* **Vous pouvez utiliser les langages Python, Java ou Go**



# Introduction

Les codes fournis permettent d'implémenter un protocole simple de transfert de fichiers entre un serveur et un client via un réseau. Le serveur est chargé de fournir la liste des fichiers disponibles pour téléchargement et de transférer ces fichiers sur demande du client. Le client se connecte au serveur, demande la liste des fichiers disponibles et télécharge les fichiers souhaités à partir de cette liste. Ces codes peuvent être utilisés pour faciliter le partage de fichiers entre des machines connectées à un réseau local ou à Internet.

# 

## Documentation du Code Serveur

**Voici le code Serveur :**

package reseau\_protocol;

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class Server {

private static int port = 8080;

private static String directory = "C:\\Files"; // Chemin absolu du répertoire de fichiers

public static void main(String[] args) throws IOException {

ServerSocket serverSocket = new ServerSocket(port);

System.out.println("Serveur démarré sur le port " + port);

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Connexion entrante de " + clientSocket.getInetAddress().getHostAddress());

Thread t = new Thread(new ClientHandler(clientSocket));

t.start();

}

}

private static class ClientHandler implements Runnable {

private Socket clientSocket;

public ClientHandler(Socket socket) {

this.clientSocket = socket;

}

public void run() {

try {

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

System.out.println("Le client est connecté: " + clientSocket.getInetAddress().getHostAddress());

out.println(getFileList());

String files = in.readLine();

String[] filenames = files.split(",");

for (String filename : filenames) {

sendFile(filename, clientSocket);

}

} catch (IOException e) {

System.err.println("Erreur: " + e.getMessage());

} finally {

try {

clientSocket.close();

} catch (IOException e) {

System.err.println("Erreur lors de la fermeture du socket: " + e.getMessage());

}

}

}

private String getFileList() {

File folder = new File(directory);

File[] files = folder.listFiles();

StringBuilder sb = new StringBuilder();

for (File file : files) {

if (file.isFile()) {

sb.append(file.getName() + ",");

}

}

return sb.toString();

}

private void sendFile(String filename, Socket clientSocket) throws IOException {

FileInputStream fileInputStream = null;

BufferedInputStream bufferedInputStream = null;

OutputStream outputStream = null;

try {

File file = new File(directory + "/" + filename);

byte[] byteArray = new byte[(int) file.length()];

fileInputStream = new FileInputStream(file);

bufferedInputStream = new BufferedInputStream(fileInputStream);

bufferedInputStream.read(byteArray, 0, byteArray.length);

outputStream = clientSocket.getOutputStream();

outputStream.write(byteArray, 0, byteArray.length);

outputStream.flush();

System.out.println("Fichier envoyé: " + filename);

} catch (IOException e) {

System.err.println("Erreur lors de l'envoi du fichier: " + e.getMessage());

} finally {

if (bufferedInputStream != null) bufferedInputStream.close();

if (outputStream != null) outputStream.close();

}

}

}

}

### Package

Le code serveur est inclus dans le package **serveur**.

### Variables

* **port**: un entier définissant le port sur lequel le serveur écoute. Par défaut, la valeur est **8080**.
* **directory**: une chaîne de caractères contenant le chemin absolu du répertoire dans lequel se trouvent les fichiers à partager.

### Classe Server

Cette classe contient la méthode **main()** qui est le point d'entrée de l'application. Elle est responsable de la création d'une instance de **ServerSocket** pour écouter les connexions entrantes, d'accepter les connexions entrantes et de créer un thread pour chaque client connecté.

* **main()**: cette méthode initialise un objet **ServerSocket** sur le port spécifié et attend les connexions entrantes. Lorsqu'un client se connecte, un nouveau thread **ClientHandler** est créé pour gérer la connexion.

### Classe ClientHandler

Cette classe implémente **Runnable** et est utilisée pour traiter chaque client connecté.

* **run()**: cette méthode est exécutée dans un thread séparé pour chaque client connecté. Elle gère l'envoi de la liste des fichiers disponibles au client et la réception des demandes de téléchargement de fichiers. Elle crée également un objet **PrintWriter** pour envoyer les données au client et un objet **BufferedReader** pour recevoir les données du client.
* **getFileList()**: cette méthode renvoie une chaîne de caractères contenant la liste des noms de fichiers disponibles dans le répertoire spécifié.
* **sendFile()**: cette méthode est utilisée pour envoyer un fichier demandé par le client. Elle utilise un flux **FileInputStream** pour lire le contenu du fichier et un flux **BufferedOutputStream** pour l'écrire dans le flux de sortie associé au socket client.
* Le chemin des fichiers téléchargeable peut être modifié dans le code a la ligne 8.

## Documentation du Code Client

**Voici le code Client :**

**package reseau\_protocol;**

**import java.net.\*;**

**import java.io.\*;**

**import java.util.\*;**

**public class Client {**

**private static int *port* = 8080;**

**private static String *serverAddress* = "127.0.0.1";**

**private static String *downloadDirectory* = "../"; // Chemin absolu du répertoire de téléchargement**

**public static void main(String[] args) throws IOException {**

**Socket socket = null;**

**BufferedReader consoleIn = new BufferedReader(new InputStreamReader(System.*in*));**

**try {**

**socket = new Socket(*serverAddress*, *port*);**

**BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));**

**PrintWriter out = new PrintWriter(socket.getOutputStream(), true);**

**System.*out*.println("Connecté au serveur " + *serverAddress* + ":" + *port*);**

**System.*out*.println("Liste des fichiers hébergés:");**

**String fileList = in.readLine();**

**String[] files = fileList.split(",");**

**for (String file : files) {**

**System.*out*.println(file);**

**}**

**System.*out*.print("Entrez le nom du fichier à télécharger (séparé par une virgule si plusieurs fichiers): ");**

**String input = consoleIn.readLine();**

**out.println(input);**

**String[] filenames = input.split(",");**

**for (String filename : filenames) {**

**byte[] byteArray = new byte[1024\*1024];**

**InputStream inputStream = socket.getInputStream();**

**FileOutputStream fileOutputStream = new FileOutputStream(*downloadDirectory* + "/" + filename);**

**BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(fileOutputStream);**

**int bytesRead = 0;**

**while ((bytesRead = inputStream.read(byteArray)) != -1) {**

**bufferedOutputStream.write(byteArray, 0, bytesRead);**

**bufferedOutputStream.flush();**

**}**

**bufferedOutputStream.close();**

**System.*out*.println("Fichier téléchargé: " + filename);**

**}**

**} catch (IOException e) {**

**System.*err*.println("Erreur: " + e.getMessage());**

**} finally {**

**if (socket != null) socket.close();**

**}**

**}**

}

### Package

Le code client est inclus dans le package **client**.

### Variables

* **port**: un entier définissant le port sur lequel le client se connecte au serveur. Par défaut, la valeur est **8080**.
* **serverAddress**: une chaîne de caractères contenant l'adresse IP du serveur auquel le client doit se connecter.
* **downloadDirectory**: une chaîne de caractères contenant le chemin absolu du répertoire dans lequel les fichiers téléchargés doivent être enregistrés.

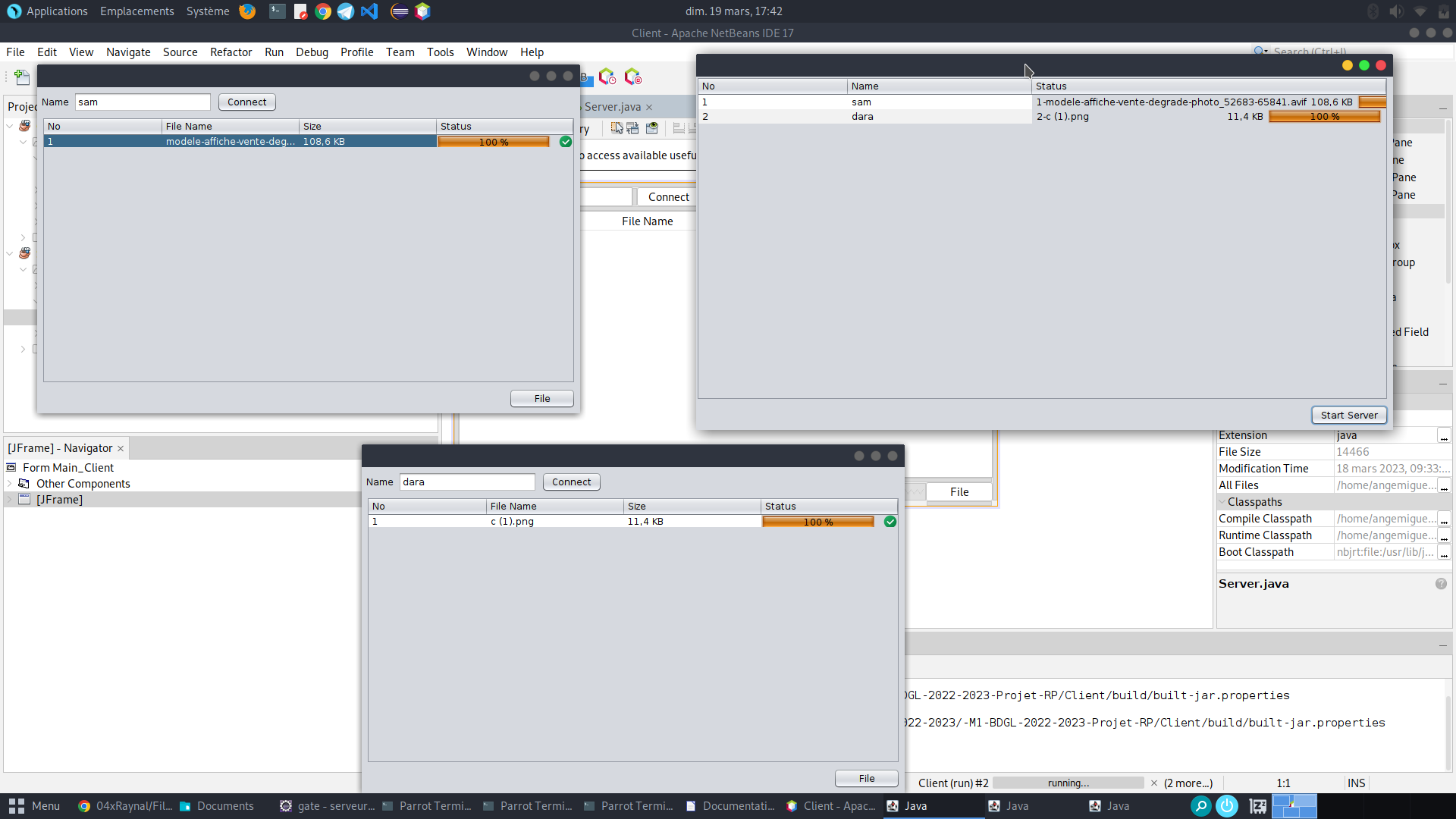
### Classe Client

Cette classe contient la méthode **main()** qui est le point d'entrée de l'application. Elle est responsable de la création d'une instance de **Socket** pour se connecter au serveur, de l'envoi des demandes de fichiers au serveur et du téléchargement des fichiers.

* **main()**: cette méthode initialise un objet **Socket** en se connectant au serveur spécifié. Elle crée ensuite un objet **PrintWriter** pour envoyer les données au serveur et un objet **BufferedReader** pour recevoir les données du serveur. Elle demande ensuite la liste des fichiers disponibles et affiche les noms des fichiers au client.
* La méthode **main()** déclare une socket **socket** et une instance de **BufferedReader** nommée **consoleIn**. Elle essaie d'établir une connexion avec le serveur en utilisant la classe **Socket** et l'adresse IP et le port du serveur. Elle crée également une instance de **BufferedReader** nommée **in** pour lire les données envoyées par le serveur via la socket, et une instance de **PrintWriter** nommée **out** pour envoyer des données au serveur via la socket.
* La méthode demande ensuite la liste des fichiers hébergés sur le serveur en lisant une ligne de **in** et en la stockant dans la chaîne **fileList**. Elle sépare la chaîne en utilisant une virgule comme délimiteur et affiche chaque nom de fichier dans la liste à l'aide d'une boucle for.
* Enfin, la méthode invite l'utilisateur à entrer le nom du fichier qu'il souhaite télécharger en utilisant **consoleIn.readLine()**, stocke la chaîne entrée dans **input** et l'envoie au serveur via **out.println(input)**. Elle sépare également la chaîne en utilisant une virgule comme délimiteur et stocke les noms de fichier individuels dans **filenames**. Pour chaque nom de fichier dans **filenames**, elle crée un tableau de byte, un objet **InputStream** pour lire les données envoyées par le serveur via la socket, et des objets **FileOutputStream** et **BufferedOutputStream** pour écrire les données dans le fichier de destination. Elle lit ensuite les données envoyées par le serveur, les écrit dans le fichier de destination et ferme les flux de sortie. Enfin, elle affiche un message pour indiquer que le fichier a été téléchargé.
* C'est ainsi que les deux codes sont utilisés pour permettre à un client de télécharger des fichiers hébergés sur un serveur distant.

**VERSION 2**

Dans cette nouvelle version nous avons fais une interface graphique avec les jFrame pour illustré le transfert de fichier.



# Vue de dessus d’une poignée de main au-dessus de documents d’affairesConclusion

En conclusion, les codes fournis implémentent un système client-serveur permettant le téléchargement de fichiers depuis le serveur vers le client. Le serveur est capable de lister les fichiers disponibles à télécharger et le client peut sélectionner les fichiers qu'il souhaite télécharger. Les codes sont bien documentés avec des commentaires clairs pour faciliter la compréhension de leur fonctionnement.