TD Des Sockets aux Serveurs

Client Serveur 1 :

Les sockets en Java permettent la communication entre applications sur des machines différentes. Voici un exemple simple qui illustre l'utilisation de sockets pour une communication client-serveur. Examiner le code du serveur et du client.

Client Serveur 2 :

1) **Serveur A**
   a. Lisez le code du serveur A et expliquez sommairement ce qu'il réalise.
   b. Lancez le serveur A utiliser le Client du package clientserveur1 pour vous connecter.
   c. Lancez un navigateur internet ou postman et connecter vous au ServeurA
   d. Placez un point d'arrêt dans le code du serveur
      Par exemple ligne 42 : `out.write("HTTP/1.0 200 OK\r\n");`
      Et relancer le serveur.
   e. Lancez 2 clients successivement et observer ce qu'il se passe pour les réponses à chacun des clients.
   f. Expliquez ce comportement

2) **Serveur B**
   a. Lisez le code du serveur B et expliquez sommairement ce qu'il réalise, préciser la différence avec le serveur A.
   b. Lancez le serveur B un navigateur ou postman pour vous y connecter.
   c. Placez un point d'arrêt dans la classe Page
      Par exemple ligne 32 : `out.write("HTTP/1.0 200 OK\r\n");`
      Et relancer le serveur.
   d. Lancez 2 clients C1 et C2 successivement et observez ce qu'il se passe pour les réponses à chacun d'eux en jouant avec le mode débug et les points d'arrêts. Peut-on obtenir la réponse à C2 avec celle à C1 ?
   e. Si oui Expliquez ce comportement par étape.

3) **Serveur B : Améliorations**
   a. Récupérez les paramètres de la requête du client lorsque vous appelez l'adresse **http://localhost:1989/test?nom=deGrasseTyson&prenom=Neil** dans la classe Page et affichez dans la console.
   b. Renvoyez une page personnalisée au client avec ses noms, prénom, la date du jour.
   c. Incorporez le nom du système d'exploitation et du navigateur internet (ne restez pas bloqué sur cette question)

Client Serveur 3 :

Analyser le code de ce petit serveur MAIN, quels sont les avantages de cette classe ?
Attention le fichier unique contient 2 classes, mais ce n'est pas forcément la meilleure façon d'écrire du code !

`SimpleHttpClient` Un web browser en java …

TP Des Sockets aux Serveurs

En vous basant sur la version Client Serveur 3 écrire un serveur java qui donne accès à 3 contextes :

1) `/accueil`
   Renvoie une page HTML contenant un formulaire avec 2 champs input pour 2 nombres et un bouton submit.
   Ce dernier pointera sur l'url suivante.

2) `/calcul`
   Récuoère les paramètres de la requête précédente, calcul la somme et l'intègre dans une réponse HTML au client.
3) `/img`
   Challenge de lecture d'une image dans le répertoire de l'application que l'on renvoie au client lorsqu'il appelle cette url.

Allez-y étape par étape. Une fois la mécanique comprise, écrivez vos page html dans des fichiers textes que vous ferez lire par le serveur avant de les envoyer au client.

Bonus :

- Nous avons écrit un module de calcul au début du cours, réutilisez ce code (votre class Calcul) pour l'intégrer et proposer plusieurs opérations au client.
- L'image peut aussi servir de bandeau dans la page d'accueil et d'affichage du résultat si vous avez du temps.

Pour ces dernières étapes, aidez-vous des fragments de code ci-dessous.

Super Bonus :

- Vous vous sentez en forme ? Gérez votre liste de course à travers un petit serveur.
  - Une page Html unique affiche la possibilité d'entrer un nom d'article et une quantitée dans 2 champs d'un formulaire.
    Un bouton envoyer ajoute à la liste de course.
  - Dans cette même page, la liste s'affiche sous le formulaire, chaque ligne affiche la quantitée et le nom du produit.
    Au boude la ligne un bouton pour supprimer cet article de la liste.

Super Super Bonus :

- Etudiez la sérialization en java.
  - Au démarrage du serveur récupérer la liste des articles.
  - A l'arrêt propre du serveur (à prévoir par une commande STOP), enregistrer la liste des articles

Quelques snippets à vous approprier :

Pour lire un fichier texte en Java, vous pouvez utiliser la classe `BufferedReader` avec un `FileReader`. Cela offre une manière efficace de lire les lignes du fichier. Voici un exemple simple :

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderExample {
    public static void main(String[] args) {
        // Spécifiez le chemin du fichier à lire
        String filePath = "example.txt";

        // Utilisez try-with-resources pour s'assurer que les ressources sont fermées correctement
 try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;

            // Lire chaque ligne du fichier
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Celui-ci on n'en aura pas besoin dans ce TP mais autant les laisser ensemble :

Pour écrire dans un fichier texte en Java, vous pouvez utiliser la classe `BufferedWriter` avec un `FileWriter`.

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class FileWriterExample {
    public static void main(String[] args) {
        // Spécifiez le chemin du fichier à écrire
        String filePath = "output.txt";

        // Utilisez try-with-resources pour s'assurer que les ressources sont
fermées correctement
try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
            // Écrire du texte dans le fichier
            writer.write("Hello, FileWriter!\n");
            writer.write("This is an example of writing to a text file.");

            // Vous pouvez également utiliser newLine() pour ajouter une
nouvelle ligne
            writer.newLine();
            writer.write("Adding a new line.");

            // Flush pour s'assurer que toutes les données sont écrites dans le
fichier
            writer.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- BufferedWriter pour écrire le fichier de manière efficace.
- Le FileWriter est utilisé pour créer ou écraser le fichier spécifié.
- Les méthodes write pour écrire des chaînes de texte dans le fichier.
- La méthode newLine() est utilisée pour ajouter une nouvelle ligne.
- La méthode flush() est utilisée pour s'assurer que toutes les données sont écrites dans le fichier.

Lecture d’une image

```
File file =
new File("C:\\Users\\toto\\eclipse-
workspace\\ClientServeurDemoJava\\src\\clientserveur4\\img.jpg");
   System.out.println(file.getAbsolutePath());
   FileReader fileReader = new FileReader(file);

   // Création d'un bufferedReader qui utilise le fileReader
                    BufferedImage img = load(file);
```

Utiliser ImageIO classe outils pour injecter un flux de bytes dans un output stream.