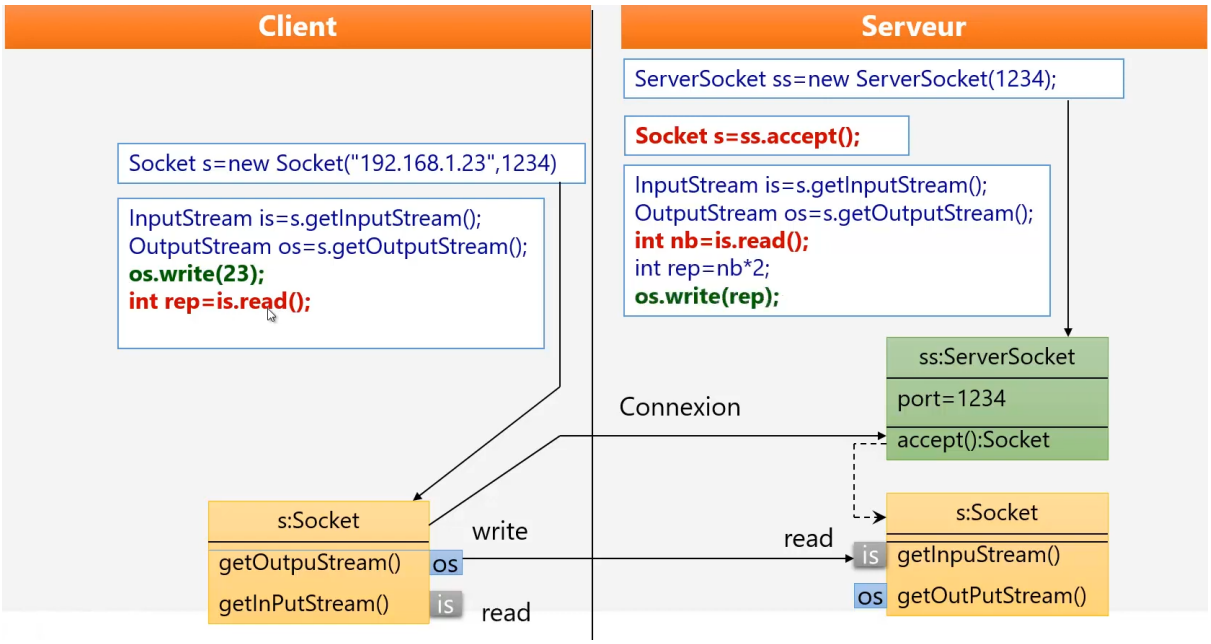
TP N°5

Sockets

PRÉPARÉ PAR : BEN ALLOU HAMZA & CHAJJAOUI SOUFAINE & OUAJIH ZAKIA

Objectif :

- L'objectif de cet exercice est de créer un système de communication simple entre client et serveur en utilisant les sockets Java.
- Le client va envoyer un nombre au serveur.
- Le serveur va recevoir ce nombre du client et le multiplier par deux.
- Le client va recevoir le resultat de calcul.



Partie I : Sockets client Serveur

Etape 1 :

- Creér une classe nommée Client et une autre nommée Serveur

Etape 2 : Dans la classe Serveur

- Instancier un objet de type ServerSocket en specifiant le numéro de port du serveur.
- Instancier les flux de communication pour la transmission des donnees entre le serveur et le client.
- Gérer les Exceptions.

Class Server : Server.java

```
1 import java.io.IOException;
2 import java.io.InputStream;
3 import java.io.OutputStream;
```

```java
import java.net.ServerSocket;
import java.net.Socket;

public class SimpleServer {

    public static void main(String[] args) {
        final int PORT = 1234; // Choisissez un num ro de port

        try {
            // Cr er une socket serveur
            ServerSocket ss = new ServerSocket(PORT);
            System.out.println("En attente d'une connexion d'un client");

            // Attendre qu'un client se connecte
            Socket s = ss.accept(); // Bloquant pour attendre une connexion d'un client
            System.out.println("Client connect  : " + s.getInetAddress());

            // Cr er des flux d'entr e et de sortie pour la communication avec le client
            InputStream is = s.getInputStream(); // Les donn es envoy es par le client
            OutputStream os = s.getOutputStream(); // Les donn es envoy es au client

            System.out.println("En attente d'un nombre");

            // Lire un entier envoy  par le client (bloquant)
            int val = is.read();
            int rep = val * 2;
            System.out.println("Envoyer la r ponse : " + rep);

            // Envoyer la r ponse au client
            os.write(rep);

            // Fermer la connexion avec le client
            s.close();
            System.out.println("Client d connect ");

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

Etape 3 : Dans la class Client

- Instancier un objet de type Socket en spécifiant l'adresse IP et le numéro de port du serveur.
- Instancier les flux de communication pour la transmission des données entre le serveur et le client.
- Gérer les Exceptions

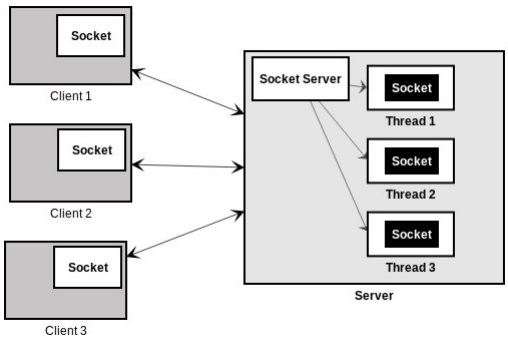
Class Client : Client.java

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.util.Scanner;

public class SimpleClient {

    public static void main(String[] args) {
        try {
            // Cr er une socket client et se connecter au serveur local sur le port 1234
            Socket c = new Socket("localhost", 1234);
            System.out.println("Connexion  tablie  avec le serveur");

            // Cr er des flux d'entr e et de sortie pour la communication avec le serveur
            InputStream is = c.getInputStream(); // Les donn es re ues du serveur
            OutputStream os = c.getOutputStream(); // Les donn es envoy es au serveur

            // Lire un nombre depuis la console
            Scanner input = new Scanner(System.in);
            System.out.println("Entrez un nombre : ");
            int val = input.nextInt();

            // Envoyer le nombre au serveur
            os.write(val);

            // Attendre la r ponse du serveur (bloquant)
            int res = is.read();
            System.out.println("R ponse du serveur : " + res);

            // Fermer la connexion avec le serveur
            c.close();

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

Note :
Veuillez noter qu'il est essentiel d'éxecuter le serveur avant le client pour assurer un fonctionnement correct du système.

Partie II : Création d'un modèle serveur multithread

Objectif :
L'objectif de cet exercice est de créer un serveur multithread en Java. Un serveur multithread est capable de gérer simultanément plusieurs connexions clientes en utilisant des threads distincts.



Etape 1 :Créer la classe principale du serveur (realSerever dans votre cas)

- Définissez une classe qui étend la classe Thread pour représenter le serveur.
- Dans la méthode main, créez une instance du serveur et démarrez-la.

Etape 2 : Dans la classe du serveur (realSerever) :

- Créez une ServerSocket pour écouter les connexions entrantes sur un port spécifique (6060 dans votre cas).
- Dans une boucle infinie, attendez les connexions des clients en utilisant ServerSocket.accept().
- Pour chaque nouvelle connexion, créez une instance de la classe Conversation pour gérer la communication avec ce client.
- Stockez chaque instance de Conversation dans une liste partagée (par exemple, Conversations).

Etape 3 : Créer la classe de conversation (Conversation)

- Dans cette classe, initialisez les flux de lecture (BufferedReader) et d'écriture (PrintWriter) pour la communication avec le client.
- Commencez une boucle infinie pour lire les messages du client.
- Si le message est "exit", supprimez la conversation de la liste et fermez la connexion avec ce client.
- Sinon, transmettez le message à tous les autres clients connectés.

Etape 4 : Gestion des Threads

- Chaque fois qu'une nouvelle connexion est acceptée, créez une instance de Conversation dans un nouveau thread.
- La classe Conversation gérera la communication avec ce client spécifique.

Etape 5 : Fermeture propre

- Assurez-vous de gérer correctement la fermeture des connexions et des threads lorsque le serveur est arrêté.

Etape 6 : Tests et Débogage

- Testez votre serveur avec plusieurs clients pour vous assurer qu'il peut gérer simultanément plusieurs connexions.

Partie III : Pour tester le serveur avec le client Telnet, suivez ces étapes :

Etape 1 : Déployer le serveur

- Assurez-vous que le serveur est en cours d'exécution. Si ce n'est pas le cas, exécutez le programme qui contient le serveur (realSerever dans votre cas).

Etape 2 : Ouvrir une console Telnet

- Ouvrez une nouvelle fenêtre de terminal ou de console.

Etape 3 : Utiliser Telnet pour se connecter au serveur

- Utilisez la commande Telnet avec l'adresse IP du serveur et le port spécifié (6060 dans votre cas). Remplacez adresse IP par l'adresse IP réelle du serveur.

Etape 4 : Interagir avec le serveur

- Une fois connecté avec Telnet, vous devriez voir les messages envoyés par le serveur.
- Vous pouvez taper des messages et les envoyer au serveur. Les messages seront relayés à tous les autres clients connectés.

Etape 5 : Tester la fonction "exit"

- Essayez de taper "exit" pour voir comment le serveur gère la déconnexion d'un client.

Etape 6 : Répétez le processus avec plusieurs clients

- Ouvrez plusieurs fenêtres de terminal ou de console et répétez le processus avec plusieurs clients Telnet pour tester la capacité du serveur à gérer plusieurs connexions simultanées.