Serveurs et clients

Table des matières

Serveurs et clients 1
Programme serveur 2
La classe ServerSocket 2
Constructeur 2
Méthode accept 2
Programme modèle 2
Gérer la connexion 3
Déclarations 3
Le constructeur 4
Méthode run 4
La méthode stop 5
Utilisation du programme ServHeure 5
Programme client 6
Gestion d'une connexion 6
Obtenir une connexion 6
Flots d'entrée et de sortie 6
Principe du programme client 6
Programme client du serveur d'heure 7
Déclarations 7
Constructeur 7
Connexion et déconnexion 8
Méthode run 8
Envoi des requêtes 9
Gestion des évènements de la fenêtre 9
Méthode main 9
Utilisation du programme 9
Exercice 10

Deux ordinateurs peuvent communiquer entre eux en utilisant le modèle client-serveur. Le serveur attend qu'un client entre en communication avec lui, puis lorsqu'une communication est établie répond aux requêtes qu'il reçoit. Le client essaie d'entrer en communication avec le serveur, puis lui envoie des requêtes et reçoit des réponses. La communication se fait grâce aux objets de la classe **Socket** qui permettent d'obtenir des flots d'entrée et de sortie.

Programme serveur

Un programme serveur attend qu'un programme client demande une connexion, puis, lorsque la connexion est établie par l'intermédiaire d'un objet de la classe **Socket**, répond aux requêtes du client.

La classe ServerSocket

Le package java.net contient la classe ServerSocket qui représente un serveur. Celui-ci est associé à un **port** qui est un numéro d'identification supérieur à 1024 (les autres numéros sont réservés).

Constructeur

On construit un objet de la classe ServerSocket en indiquant son numéro de port.

```
serveur=new ServerSocket(port);
```

Méthode accept

Après la création du serveur, on utilise la méthode accept pour attendre les tentatives de connexion effectuées par les clients. Lorsqu'une connexion est détectée, la méthode accept renvoie un objet de la classe Socket qui permettra d'établir le dialogue à travers des flots dentrée et de sortie. Le serveur créera alors un processus qui gèrera cette communication par l'intermédiaire d'une classe à créer nommée MaConnexion. Tout se passe à l'intérieur d'une boucle infinie qui peut prendre la forme suivante :

```
while (true) {
 //création de nouvelles connexions
 Socket s=serveur.accept();
 new MaConnexion(s);
}
```

C'est la classe MaConnexion qui permet de définir le fonctionnement du serveur.

Programme modèle

Comme la plupart des méthodes utilisées sont susceptibles de provoquer des exceptions, elles seront utilisées dans des structures try...catch... Nous prendrons comme exemple un programme serveur destiné à envoyer l'heure courante, il pourra se présenter sous la forme suivante :

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

public class ServHeure {

 static int port;

 public static void main(String args[]) {
  ServerSocket serveur;
  //définition du port
  try {
   port=Integer.parseInt(args[0]);
  } catch (Exception e) {
   port=1234;  //valeur par défaut
  }
  //installation
  try {
   serveur=new ServerSocket(port);
   while (true) {
    //création de nouvelles connexions
    Socket s=serveur.accept();
    new MaConnexion(s);
   }
  } catch (IOException e) {
   System.out.println("Erreur à la creation d'un objet Socket :
"+e.getMessage());
   System.exit(1);
  }
 }
}
```

Il suffira ensuite de le compléter en définissant la classe MaConnexion.

Gérer la connexion

Lorsque la connexion est établie avec un client, le dialogue client-serveur est gérée par la classe MaConnexion qui va créer un processus par client en implémentant l'interface Runnable.

Nous allons prendre comme exemple un serveur d'heures dont la tâche sera d'envoyer l'heure courante lorsqu'il recevra le message "h".

Déclarations

La classe MaConnexion utilisera trois variables :

- **Socket client** : l'objet de la classe Socket fourni par le constructeur qui servira à créer des flots d'entrée et de sortie
- **BufferedReader depuisClient** : objet qui permettra de recevoir les lignes de commande envoyées par le client
- **PrintWriter versClient** : objet qui permettra d'envoyer des messages au client.

On obtient le code suivant :

```
class MaConnexion implements Runnable {
 Socket client;               //liaison avec client
 BufferedReader depuisClient; //réception de requête
 PrintWriter versClient;      //envoi des réponses
```

Le constructeur

Le constructeur est chargé d'initialiser les 3 variables utilisées, d'envoyer un message de bienvenue et de lancer le processus de dialogue.

Le Socket client permet d'obtenir des flots d'entrée et de sortie grâce aux méthodes **getInputStream** et **getOutputStream**. Ceci nous donne le code suivant :

```
public MaConnexion(Socket client) {
 this.client=client;
 try {
  // création des flots de/vers le client
  depuisClient=new BufferedReader(
    new InputStreamReader(client.getInputStream()));
  versClient=new PrintWriter(
   new OutputStreamWriter(client.getOutputStream()),true);
  // message d'accueil
  versClient.println("Bienvenue sur le serveur d'heure.");
  versClient.println("Entrez h pour obtenir l'heure.");
  versClient.println("Envoyez une chaîne vide pour fermer la
connexion.");
 } catch (IOException e) {
  try {
   client.close();
  } catch (IOException ee) {}
 }
 //mise en route du processus par appel de la méthode run
 new Thread(this).start();
}
```

Méthode run

La méthode run (exigée par l'interface Runnable) gère le dialogue en attendant des messages et en répondant aux messages reçus.

Dans notre exemple, seuls les messages "h" sont pris en compte et provoquent l'envoi de l'heure. Les autres messages provoquent l'arrêt de la connexion. Ceci nous donne le code suivant :

```
          public void run() {
           boolean fini=false; //drapeau
           try {
            String lue;  //la requête
            String rep;  //la réponse
            while (!fini) {
             lue=depuisClient.readLine();
             if (lue==null) fini=true;
             else if (!lue.equals("h")) fini=true;
             else {
              //on envoie l'heure
              Calendar cal=Calendar.getInstance();
              rep="Il est "+cal.get(Calendar.HOUR)+"h"

                +cal.get(Calendar.MINUTE)+"mn"+cal.get(Calendar.SECOND)+"s.;
              versClient.println(rep);
             }
            }
           } catch (IOException e) {
            System.out.println("Exception entrée/sortie : "+e.getMessage());
           }
           //fermeture de la connexion
           stop();
          }
```

La méthode stop

Il nous reste à écrire la méthode stop qui met fin à la connexion. Il suffit pour cela d'appeler la méthode close du Socket client.

```
          public void stop() {
           try {
            versClient.println("Au revoir !");
            client.close();
           } catch (IOException e) {
            System.out.println("Exception à la fermeture d'une connexion :
"+e);
           }
          }
```

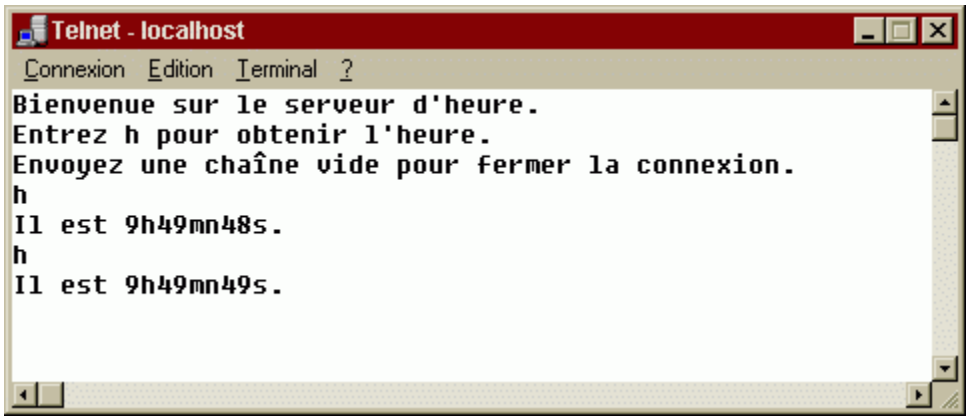
Utilisation du programme ServHeure

Après avoir compilé le fichier ServHeure.java, nous pouvons l'exécuter dans une console DOS en utilisant la commande :

```
java ServHeure
```

Le serveur attend alors qu'un programme client établisse une connexion. Sous Windows nous pouvons utiliser **telnet** pour le faire.

On ouvre la connexion en utilisant le menu Connexion et en indiquant l'adresse de l'ordinateur sur lequel fonctionne le serveur (localhost pour votre propre ordinateur) ainsi que le numéro de port 1234. Le message de bienvenue du serveur apparait, puis l'heure est affichée à chaque envoi de la commande "h".



On met fin au programme serveur en activant la console DOS qui a permis de l'exécuter et en appuyant sur la combinaison de touches Ctrl-C.

Programme client

Un programme client essaie d'établir une connexion à un programme serveur par l'intermédiaire d'un objet de la classe **Socket**. Il envoie ensuite des requêtes et reçoit des réponses.

Gestion d'une connexion

Obtenir une connexion

On obtient une connexion à un serveur en créant un objet de la classe Socket. Le constructeur utilisé doit fournir en paramètres le nom de l'ordinateur contenant le serveur et le numéro de port du serveur. On pourra par exemple écrire :

```
Socket sk=new Socket(adresseServeur,numPort);
```

Flots d'entrée et de sortie

L'objet Socket obtenu lors de la connexion fournit des flots d'entrée et de sortie grâce aux méthodes **getInputStream** et **getOutputStream**. Ceci permet d'obtenir des objets BufferedReader et PrintWriter pour dialoguer avec le serveur par l'intermédiaire de chaines de caractères.

```
BufferedReader depuisServeur=new BufferedReader(
 new InputStreamReader(sk.getInputStream()));
PrintWriter versServeur=new PrintWriter(
 new OutputStreamWriter(sk.getOutputStream()),true);
```

Principe du programme client

Après avoir obtenu une connexion, le programme client pourra créer un processus d'écoute des messages du serveur. Les requêtes seront envoyées à l'aide d'un bouton, les réponses seront affichées dans un composant.

Programme client du serveur d'heure

Ecrivons un programme client pour le serveur d'heure créé précédemment. Celui-ci renvoie l'heure sous la forme d'une chaine de caractères du type "Il est 10h26mn45s." lorsqu'on lui transmet la requête "h".

Déclarations

Le programme client utilisera les variables suivantes :

- int portServeur : numéro de port du serveur
- String adresseServeur : nom de l'ordinateur contenant le serveur
- Thread processus : processus de réception des messages
- Label heure : pour afficher l'heure
- Socket sk : Socket de communication
- BufferedReader depuisServeur : pour recevoir les messages du serveur
- PrintWriter versServeur : pour envoyer des requêtes au serveur

On obtient le code suivant :

```
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;

public class CliHeure extends Frame
 implements Runnable, ActionListener {

 int portServeur;
 String adresseServeur;
 Thread processus=null;
 Label heure=new Label("Connexion au serveur d'heure");
 Socket sk;
 BufferedReader depuisServeur;
 PrintWriter versServeur;
```

Constructeur

Le constructeur récupère le nom d'ordinateur et le numéro de port passés en paramètres, construit l'interface du programme (un Label et un Button), crée le processus d'attente des messages et le lance avec la méthode run qui commencera par établir la connexion. On obtient le code suivant:

```
 public CliHeure(String adresse, int port) {
  portServeur=port;
  adresseServeur=adresse;
  //évènement fermeture
  addWindowListener(new Fermeture());
  //création de l'interface
  Panel p=new Panel();
  p.setBackground(Color.white);
  add(BorderLayout.CENTER,p);
  p.setLayout(new GridLayout(2,1));
  p.add(heure);
```

```
 heure.setAlignment(Label.CENTER);
 Panel p1=new Panel();
 p.add(p1);
 Button b=new Button("Heure");
 p1.add(b);
 b.addActionListener(this);
 pack();
 //processus d'attente des messages
 processus=new Thread(this);
 processus.start();
}
```

Connexion et déconnexion

La connexion et la déconnexion au serveur se feront avec les méthodes connect et disconnect.

La méthode connect crée le Socket de communication et initialise les flots d'entrée et de sortie.

```
public void connect() {
 try {
  sk=new Socket(adresseServeur,portServeur);
  depuisServeur=new BufferedReader(
   new InputStreamReader(sk.getInputStream()));
  versServeur=new PrintWriter(
   new OutputStreamWriter(sk.getOutputStream()),true);
  //demander l'heure
  versServeur.println("h");
 } catch (Exception e) {
  heure.setText(e.toString());
 }
}
```

La méthode disconnect se contente de fermer le Socket obtenu lors de la connexion en appelant sa méthode close.

```
public void disconnect() {
 try {
  sk.close();
 } catch (IOException e) {
 }
}
```

Méthode run

La méthode run attendue pour l'implémentation de l'interface Runnable va créer la connexion, puis attendre les messages du serveur. Seuls ceux commençant par "Il est" seront pris en compte et affichés dans le Label heure.

```
public void run() {
 boolean fini=false;
 try {
  connect();
  String ligne;
  while (!fini) {
   ligne=depuisServeur.readLine();
   if (ligne==null) fini=true;
   else if (ligne.startsWith("Au revoir")) fini=true;
   else if (ligne.startsWith("Il est")) heure.setText(ligne);
```

```
          }
         } catch (IOException e) {
          heure.setText("Connexion impossible.");
         } finally {
          processus=null;
         }
        }
```

Envoi des requêtes

Les requêtes sont envoyées lors de l'appui sur le bouton "Heure". Elles se réduisent à la chaine "h". Ceci nous donne la méthode actionPerformed suivante:

```
        public void actionPerformed(ActionEvent e) {
         if (e.getActionCommand().equals("Heure"))
versServeur.println("h");
        }
```

Gestion des évènements de la fenêtre

La classe Fermeture dérivée de WindowAdapter gèrera l'évènement fermeture de la fenêtre.

```
        class Fermeture extends WindowAdapter {
         public void windowClosing(WindowEvent e) {
          versServeur.println("");
          disconnect();
          System.exit(0);
         }
        }
```

Méthode main

Il reste à écrire la méthode main qui crée une instance de la fenêtre du programme et l'affiche.

```
        public static void main(String args[]) {
         System.out.println("Chargement en cours...");
         String adr="localhost";
         if (args.length>0) adr=args[0];
         int p=1234;
         if (args.length>1) {
          try {
           p=Integer.parseInt(args[1]);
          }
          catch (Exception e) {p=1234;}
         }
         CliHeure cli=new CliHeure(adr,p);
         cli.setTitle("Heure");
         cli.setVisible(true);
        }
```

Utilisation du programme

Après avoir lancé le serveur d'heure, on lance le programme obtenu.

A chaque appui sur le bouton "Heure" l'heure est mise à jour.

Exercice

- Numéros de téléphone
  Créer un système client-serveur de gestion de numéros de téléphone.