Université de Nice-Sophia Antipolis
ELEC5-TR

POLYTECH
2017–2018

Examen de Langage Java

Durée : 1h
Aucun document autorisé
Mobiles interdits *Note : la qualité des commentaires, avec notamment la présence d'affirmations significatives, ainsi que les noms donnés aux variables, et la bonne indentation rentreront pour une part importante dans l'appréciation du travail.*

Vous choisirez de répondre à UNE question parmi les 2 suivantes :

Question 1

▶ 1. On désire calculer la somme de tous les éléments d'une matrice de réels double. Pour cela, on calculera la somme des éléments de chacune des lignes de la matrice, puis sommera les sommes par ligne pour obtenir la somme totale.

Le calcul de la somme des éléments d'une ligne sera effectué à l'aide d'un thread. Il y aura donc autant de threads que de lignes que comporte la matrice.

▶ 2. Écrivez la classe `SommeLigne` qui définit un thread calculant la somme des réels d'une ligne d'une matrice.

▶ 3. Écrivez la classe `Matrice`, qui comporte la méthode `main` qui calcule la somme des éléments d'une matrice de réels (que vous déclarerez) en utilisant des threads du type `SommeLigne` que vous avez précédemment défini.

```java
/** Cette classe définit un thread qui
 *  calcule la somme des valeurs d'une ligne d'une matrice
 *  d'entiers et incrémente le compteur de la classe Matrice
 */
public class SommeLigne implements Runnable {

    public Thread proc = new Thread(this);
    private double[] ligne;

    //Rôle : mémorise une ligne de
    //matrice dont il faudra calculer la somme
    public SommeLigne(double [] l) {
        ligne = l;
    }
    //Rôle : calcule la somme des valeurs de
    //ligne et incrémente la somme totale
    public void run() {
        int somme = 0;
        for (double x : ligne)
            somme+=x;
        //incrémenter le compteur de la classe Matrice
        Matrice.addLigne(somme);
    }
}

public class Matrice {
```

```java
    private static int sommeTotale = 0;

    /* sommeTotale est une ressource critique,
     * on place un verrou (synchronized)
     */
    public static synchronized void addLigne(int x) {
        sommeTotale+=x;
    }

    public static void main(String [] args) {
        double [][]mat = { {2, 2, 3, 4},
                           {5, 6, 7, 8},
                           {9, 11, 11, 12} };
        //un thread par ligne de la matrice
        SommeLigne [] p = new SommeLigne[mat.length];
        //créer chacun des threads, les lancer
        //et attendre la fin de leur l'exécution
        //avant d'afficher le résultat
        try {
            for (int i=0; i<mat.length; i++) {
                p[i] = new SommeLigne(mat[i]);
                p[i].proc.start();
                p[i].proc.join();
            }
        }
        catch (InterruptedException e) {}
        //tous les threads ont terminé leur exécution
        //afficher la somme totale des éléments de la matrice
        System.out.println("total␣=␣" + sommeTotale);
    }
}
```

. . . .

Question 2

▶ 4. On souhaite écrire un serveur TCP pour faire des calculs sur des entiers avec les quatre opérations +, −, × et /. Le client envoie au serveur des commandes sur une ligne de la forme « `opération entier` » (*e.g.* `+ 6` ou `/ -3`). Le serveur envoie en retour le résultat de l'opération. Les opérations sont réalisées séquentiellement avec une initialisation à 0 ; tous les opérateurs ont la même priorité et sont associatifs à gauche. La commande `fin` termine les calculs. Voici un exemple d'échanges entre un client et le serveur qui correspond à l'opération ((0+20)-5)/3 :

```
Client> + 20
Serveur> 20
Client> - 5
Serveur> 15
Client> / 3
Serveur> 5
Client> fin
Serveur> Fin connexion
```

▶ 5. Écrivez la méthode `main` de la classe `Serveur` qui accepte les connexions de clients qui lui présentent des opérateurs arithmétiques à réaliser.

▶ 6. Écrivez la classe `GestionDuClient` (un thread) qui réalise sur le serveur les opérations de calcul communiquées par un client et qui lui renvoie les résultats. L'objet `s` représente une socket de communication TCP déjà ouverte vers un client.

Note : vous pourrez utiliser la méthode `substring(int i, int j)` de `String` qui renvoie une sous-chaîne comprise entre les indices i (inclus) et j (exclus). D'autre part, un client pourra être représenté par la classe suivante :

```java
 /*
 *  Usage : java Client serveur port
 */
import java.net.*;
import java.io.*;

public class Client {
    public static void main (String [] args) {
        int portDest = Integer.parseInt(args[1]);
        String serveur = args[0];
        //
        try (
             Socket socket = new Socket(serveur, portDest);
             PrintStream out = new PrintStream(socket.getOutputStream());
             BufferedReader in = new BufferedReader(
                               new InputStreamReader(socket.getInputStream())))
                {
                    Console console;
                    if ((console = System.console()) == null) {
                        System.err.println("Erreur␣:␣pas␣de␣console␣disponible");
                        System.exit(1);
                    }
                    boolean fin=false;
                    do {
                        String cmd = console.readLine("client>␣");
                        out.println(cmd);
                        String réponse = in.readLine();
                        if (réponse.equals("fin")) fin=true;
                        else
                            System.out.println(réponse);
                    } while (!fin);
                }
        catch (IOException e) {
            System.err.println(":␣"+ e);
        }
    }
}
```

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

/**
 * Usage : java Serveur port
 *
 * Cette classe représente accepte des connexions de clients qui lui
 * présentent des opérateurs arithmétiques à réaliser
 *
 */
public class Serveur {
    public static void main (String [] args) throws IOException {
        int port = Integer.parseInt(args[0]);
        //creation du socket
        ServerSocket socketServeur = new ServerSocket(port);
        System.out.println("Serveur␣actif␣sur␣le␣port␣" + port);
        while (true) {
            //attendre la prochaine connexion
            Socket socketClient = socketServeur.accept();
            //gérer le client dans un thread
            new GestionDuClient(socketClient);
        }
    }
} //fin classe Serveur

/**
 * gère la connexion avec un client particulier.
 */
class GestionDuClient implements Runnable {
    private Thread proc = new Thread(this);
    private Socket socket;
    private String client;

    public GestionDuClient(Socket s) {
        socket = s;
        proc.start();
        client = s.getInetAddress().getHostName();
    }

    public void run() {
        System.out.println(client + "␣connecté");
        try (
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInput
             PrintStream out = new PrintStream(socket.getOutputStream()))
            {
                String cmd, msgErreur = "";
                int résultat = 0;
                while (true) {
                    cmd = in.readLine();
                    if (cmd.equals("fin")) break;
                    //il existe une opération à traiter
                    //on considère qu’il n’y a pas d’erreur de syntaxe
                    char opérateur = cmd.charAt(0);
                    int opérande = Integer.parseInt(cmd.substring(2));

                    switch (opérateur) {
                       case ’+’: résultat += opérande; break;
                       case ’-’: résultat -= opérande; break;
                       case ’*’: résultat *= opérande; break;
                       case ’/’: if (opérande == 0)  msgErreur = "division␣par␣0";
                                 else  résultat /= opérande;
                                 break;
                       default: msgErreur = "opérateur␣inconnu";
                    }
                    out.println("Serveur␣:␣" + (msgErreur.length() != 0 ? msgErreur :
résultat));
                }
                out.println("fin");
            }
        catch (IOException e) { System.err.println(e); }
        finally {
            System.out.println(client + "␣déconnecté");
```

```
        }
    }
}
```

……