Advanced Programming

Second Assignment

Exhibition

Daniel Hernández Arcos 02758969G

Juan Gómez Hernández 50338597S

UAH 2018-2019 | G. Ing. Informática | Advanced Programming

# **INDEX**

## **HIGH LEVEL DESCRIPTION**

* Introduction

## **LOW LEVEL DESCRIPTION**

* Changes made

## **HIGH LEVEL DESCRIPTION**

**Introduction**

The aim of this practice consists of essentially modifying the code given for the exhibition program in order to allow an external module to initiate it and stop it remotely. To achieve this, we’ve used TCP sockets, consisting of a Server and Client connection, which will be main programs independently. Once the server is executed, the Exhibition program will initiate. Now, in order to manipulate it remotely, a Client program is executed with the same IP and port number as the server (up to 10 clients can be connected).

## **LOW LEVEL DESCRIPTION**

In this section we will explain the main functioning of the classes implemented for this program, as well as the methods used for each class. Each of them will have a task and are essential for the correct functioning of the program.

## **Changes Made**

Classes implemented in the previous program have been slightly modified with the inclusion of 3 new added classes used for TCP connection implementation. Main changes or added code will be marked in a different color from black.

**Exhibition**

package cal2;

import java.util.concurrent.Semaphore;

import javax.swing.JTextField;

public class Exhibition {

int capacity;

Queue queueWaiting,queueInside;

Semaphore sem;

public Queue getQueueWaiting() {

return queueWaiting;

}

public Queue getQueueInside() {

return queueInside;

}

public Exhibition(int capacity, JTextField waiting, JTextField inside){

this.capacity=capacity;

sem=new Semaphore(capacity,true);

queueWaiting=new Queue(100,waiting);

queueInside=new Queue(capacity,inside);

}

public void enter(int v) throws InterruptedException{

queueWaiting.push(v);

try{ sem.acquire();} catch(InterruptedException e){}

queueWaiting.popW(v);

queueInside.pushIn(v);

}

public void leave(int v) throws InterruptedException{

queueInside.pop(v);

sem.release();

}

public void look(int v){

try {

Thread.sleep(2000+(int) (3000\*Math.random()));

} catch (InterruptedException e){}

}

}

In this class we have added 4 new methods in relation to the queues, which will allow the program to know the state of the queues to then be able to manipulate them. This is done automatically by the program, but user can also interfere by the use of the buttons “Resume” and “Stop”.

getQueueWaiting and getQueueInside will return the Queues running in the program, which is later used to verify the state of the queue (Stopped/Closed or not).

**Main**

This class was the previous main class for the program, which now has been changed to the server so that class is the one to run the program.

package cal2;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JTextField;

public class Main extends javax.swing.JFrame {

Exhibition goya;

/\*\* Creates new form Main \*/

public Main() throws IOException {

initComponents();

}

public Exhibition getGoya() {

return goya;

}

public void setGoya(Exhibition goya) {

this.goya = goya;

}

public JTextField getjTextField1() {

return jTextField1;

}

public void setjTextField1(JTextField jTextField1) {

this.jTextField1 = jTextField1;

}

public JTextField getjTextField2() {

return jTextField2;

}

public void setjTextField2(JTextField jTextField2) {

this.jTextField2 = jTextField2;

}

private void StopActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

goya.queueWaiting.setS(true);

goya.queueInside.setS(true);

}

private void CloseActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

goya.queueWaiting.setC(true);

goya.queueInside.setC(true);

}

private void ResumeActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

goya.queueWaiting.setS(false);

goya.queueInside.setS(false);

goya.queueInside.signal();

goya.queueWaiting.signal();

}

private void OpenActionPerformed(java.awt.event.ActionEvent evt) {

goya.queueWaiting.setC(false);

goya.queueInside.setC(false);

goya.queueInside.signal();

goya.queueWaiting.signal();

}

/\*\*

\* @param args the command line arguments

\*/

// Variables declaration - do not modify

private javax.swing.JButton Close;

private javax.swing.JButton Open;

private javax.swing.JButton Resume;

private javax.swing.JButton Stop;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

private javax.swing.JTextField jTextField1;

private javax.swing.JTextField jTextField2;

// End of variables declaration

}

The moficication of the methods which interact with the buttons had already been done, but it is highlighted either way to show the changes in the original code. These methods modify the textfields in the window opened as well as stop and resume the queues in the program. Also, the queue can to enter the exhibition can be openned or closed, which means that if it is closed no thread will be able to enter until it´s open again, while people inside the exhibition keep executing.

**Queue**

package cal2;

// The class Queue manages the waiting queues (actually lists, but it allow us

// to represent the content of the process queues of the monitors with push and

// pop of integers. Every time that the queue is modified, it content is printed

// in the JTextField passed in the contructor as a parameter.

import javax.swing.JTextField;

public class Queue {

int[] content;

JTextField tf;

private boolean closed=false, stop=false;

int ptr;

public boolean isClosed() {

return closed;

}

public boolean isStop() {

return stop;

}

public void setC(boolean c)

{

closed=c;

}

public void setS(boolean c)

{

stop=c;

}

public Queue(int capacity, JTextField tf){

content = new int[capacity];

ptr=0;

this.tf=tf;

}

public synchronized void pushIn(int n) throws InterruptedException{

if(stop || closed) wait();

content[ptr]=n;

ptr++;

print();

}

public synchronized void push(int n) throws InterruptedException{

if(stop) wait();

content[ptr]=n;

ptr++;

print();

}

public synchronized void pop(int n) throws InterruptedException{

if(stop) wait();

boolean flag=false;

for (int i=0;i<ptr-1;i++) {

if (n==content[i]) flag=true;

if (flag) content[i]=content[i+1];

}

ptr--;

print();

}

public synchronized void popW(int n) throws InterruptedException{

if(stop || closed) wait();

boolean flag=false;

for (int i=0;i<ptr-1;i++) {

if (n==content[i]) flag=true;

if (flag) content[i]=content[i+1];

}

ptr--;

print();

}

public synchronized void emptyQueue(){ptr=0;}

public synchronized int first(){return content[0];}

public synchronized int noOfItems(){return ptr;}

public synchronized String ptrString(){

String str="";

for (int i=0;i<ptr;i++) str=str+"-"+content[i];

return str;

}

public synchronized void print(){ tf.setText(ptrString());}

public synchronized void signal()

{

if(!closed || !stop) notifyAll();

}

}

Several methods were added to maintain queues controlled. The are in charge of updating the state of each queue, through Boolean parameters. pushIn(), push() and pushW() will function according to those parameters using monitors. Signal() will notify all threads to proceed to enter the queue if they are open and not stopped.

**NewClass**

In addition to the previous classes already created we’ve added this class where the exhibition will be created, and where all queue actions take place.

package cal2;

import cal2.Exhibition;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author dcc

\*/

public class NewClass extends Thread {

Socket connection;

DataOutputStream output;

DataInputStream input;

Exhibition goya;

public NewClass(Socket connection, Exhibition goya)

{

this.goya=goya;

this.connection=connection;

start();

}

public void run()

{

try {

//Open input-output channels

input = new DataInputStream(connection.getInputStream());

output = new DataOutputStream(connection.getOutputStream());

//Read message from the client

boolean mensaje = input.readBoolean();

System.out.println("el mensaje ha sido leido: "+mensaje);

goya.getQueueInside().setS(mensaje);

goya.getQueueWaiting().setS(mensaje);

if(!goya.getQueueInside().isClosed() || !goya.getQueueWaiting().isClosed())

{

goya.queueInside.signal();

goya.queueWaiting.signal();

}

//Close connection

connection.close();

} catch (IOException ex) {

}

}

}

**Server**

Server class is created specifically for the creation of TCP shockets (In Green the socket) and also works as the main class of the program. Here previous classes are created with their due parameters, and shocket for connection is stated. It searches automatically for a non-privilege port to connect the socket, until ir reaches the maximum TCP port available in the system

package cal2;

import java.io.\*;

import java.net.\*;

import cal2.\*;

import java.util.concurrent.CountDownLatch;

import java.util.concurrent.Executor;

import java.util.concurrent.LinkedBlockingQueue;

import java.util.concurrent.ThreadPoolExecutor;

import java.util.concurrent.TimeUnit;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Server extends Thread{

private int Puerto;

private BufferedReader entrada;

ServerSocket server;

Socket connection;

DataOutputStream output;

DataInputStream input;

private Exhibition goya;

public Server(Main main, CountDownLatch parar) throws IOException {

entrada=new BufferedReader(new InputStreamReader(System.in));

InetAddress localhost = InetAddress.getLocalHost();

System.out.println("System IP Address : " + (localhost.getHostAddress()).trim());

System.out.println();

goya=main.getGoya();

Puerto=1024; //try to find automatically a non-privilege port

do

{

try

{

server = new ServerSocket(Puerto);

bound=true;

}catch(java.net.SocketException e)

{

bound=false;

Puerto++;

if(Puerto>=49151)

{

System.out.println("there is no port available in the system, program terminated");

System.exit(0);

}

}

}while(!bound && Puerto<=49151); //if the bind process is correct or either no more tcp ports available in the system

System.out.println("Socket port : " + Puerto);

parar.countDown();

start();

}

public void run()

{

try {

// Create socket Port 5000

System.out.println("Starting server...");

Executor conjunto=new ThreadPoolExecutor(0,10, 1000, TimeUnit.MILLISECONDS, new LinkedBlockingQueue());

while (true) {

// Wait for a connection

connection = server.accept();

NewClass thread=new NewClass(connection, goya); //create a new thread for each client created

conjunto.execute(thread); //a maximum of 10 clents can connect to the server

}

} catch (IOException e) {

System.out.println(e);

}

}

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

Exhibition goya;

Visitor v;

CountDownLatch parar=new CountDownLatch(1);

Main main = new Main();

goya=new Exhibition(10,main.getjTextField1(),main.getjTextField2());

main.setGoya(goya);

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

main.setVisible(true);

}

});

Server server = new Server(main, parar);

try {

parar.await(); //wait for the server to find the port and connect the socket to start.

} catch (InterruptedException ex) {

Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);

}

for(int i=1; i<=60; i++) v=new Visitor(i,goya);

}

}

**Client**

The client is the other part necessary for TCP connection, where another socket is created to bind with the server. This object is later created in another class which will act as a main class for the program concerning the client. The port and ip is entered by the user, if it’s a privilege port or it its out o TCP port range, it must repeat the introduction, also, if there is a connection error, meaning that the ip or port to connect to its not correct.

package cal2;

import cal2.\*;

import java.io.\*;

import java.net.\*;

import java.io.BufferedReader;

import java.io.\*;

public class Client {

private static String IP;

private static int Puerto;

private BufferedReader entrada;

private Socket client;

boolean connected=false;

private DataInputStream input;

private DataOutputStream output;

public Client() throws IOException {

entrada=new BufferedReader(new InputStreamReader(System.in));

System.out.print("Introduzca la IP del sistema al que conectarse: ");

IP=entrada.readLine();

System.out.print("Introduzca el Puerto del sistema al que conectarse: ");

try

{

Puerto=Integer.valueOf(entrada.readLine());

while(Puerto<=1023) //if the customer has chosen a privilege port, try again

{

System.out.println("try with a non privileged port (>=1024): ");

Puerto=Integer.valueOf(entrada.readLine());

}

while(Puerto>49151)

{

System.out.println("try with a port in the TCP range: ");

Puerto=Integer.valueOf(entrada.readLine());

}

}catch(java.lang.NumberFormatException e)

{

System.out.println("The port must be a number");

}

}

public synchronized void stop() throws IOException

{

do{

try

{

client = new Socket(IP,Puerto);

input = new DataInputStream(client.getInputStream());

output = new DataOutputStream(client.getOutputStream());

output.writeBoolean(true);

connected=true;

}

catch(java.net.SocketException e) //if the bind process fails, try again with another port/ip

{

System.out.println("There was an error in connecting the socket, please try again: ");

System.out.print("Introduzca la IP del sistema al que conectarse: ");

IP=entrada.readLine();

System.out.print("Introduzca el Puerto del sistema al que conectarse: ");

try

{

Puerto=Integer.valueOf(entrada.readLine());

}catch(java.lang.NumberFormatException ex)

{

System.out.println("The port must be a number");

}

System.out.println();

connected=false;

}

catch(java.net.UnknownHostException e)

{

System.out.println("the ip format was incorrect");

System.out.print("Introduzca la IP del sistema al que conectarse: ");

IP=entrada.readLine();

System.out.print("Introduzca el Puerto del sistema al que conectarse: ");

try

{

Puerto=Integer.valueOf(entrada.readLine());

}catch(java.lang.NumberFormatException ex)

{

System.out.println("The port must be a number");

}

System.out.println();

connected=false;

}

}while(!connected);

client.close();

}

public synchronized void resume() throws IOException

{

do{

try

{

client = new Socket(IP,Puerto);

input = new DataInputStream(client.getInputStream());

output = new DataOutputStream(client.getOutputStream());

output.writeBoolean(false);

connected=true;

}

catch(java.net.SocketException e) //if the bind process fails, try again with another port/ip

{

System.out.println("There was an error in connecting the socket, please try again: ");

System.out.print("Introduzca la IP del sistema al que conectarse: ");

IP=entrada.readLine();

System.out.print("Introduzca el Puerto del sistema al que conectarse: ");

try

{

Puerto=Integer.valueOf(entrada.readLine());

}catch(java.lang.NumberFormatException ex)

{

System.out.println("The port must be a number");

}

System.out.println();

connected=false;

}

catch(java.net.UnknownHostException e)

{

System.out.println("the ip format was incorrect");

System.out.print("Introduzca la IP del sistema al que conectarse: ");

IP=entrada.readLine();

System.out.print("Introduzca el Puerto del sistema al que conectarse: ");

try

{

Puerto=Integer.valueOf(entrada.readLine());

}catch(java.lang.NumberFormatException ex)

{

System.out.println("The port must be a number");

}

System.out.println();

connected=false;

}

}while(!connected);

client.close();

}

}

**ControlModule**

This class is the main class for the client program. This will consist of a JFrame with the buttons to resume and stop the Server program, using the Client socket and the Server shocket to connect both programs.

package Client;

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

import cal2.Client;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

public class ControlModule extends javax.swing.JFrame {

Client cliente;

/\*\*

\* Creates new form ControlModul

\*/

public ControlModule() throws IOException {

cliente=new Client();

initComponents();

}

private void StopActionPerformed(java.awt.event.ActionEvent evt) {

try {

// TODO add your handling code here:

cliente.stop();

} catch (IOException ex) {

Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void ResumeActionPerformed(java.awt.event.ActionEvent evt) {

try {

// TODO add your handling code here:

cliente.resume();

} catch (IOException ex) {

Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

/\* Set the Nimbus look and feel \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

try {

new ControlModule().setVisible(true);

} catch (IOException ex) {

Logger.getLogger(ControlModule.class.getName()).log(Level.SEVERE, null, ex);

}

}

});

}