Университет ИТМО

Кафедра ВТ

Программирование

Лабораторная работа №6

Группа P3110

Вариант 311003

Нгу Фыонг Ань

Проверил:

Писмак Алексей Евгеньевич

2018 год

Разделить программу из [лабораторной работы №5](https://se.ifmo.ru/courses/programming#lab5) на клиентский и серверный модули. Серверный модуль должен реализовывать все функции управления коллекцией в интерактивном режиме, кроме отображения текста в соответствии с сюжетом предметной области. Клиентский модуль должен запрашивать у сервера текущее состояние коллекции, генерировать сюжет, выводить его на консоль и завершать работу.

Хранящиеся в коллекции объекты должны иметь следующие характеристики:

* имя, название или аналогичный текстовый идентификатор;
* размер или аналогичный числовой параметр;
* характеристику, определяющую местоположение объекта на плоскости/в пространстве;
* время/дату рождения/создания объекта.

Если аналогичные характеристики уже есть, добавлять их не нужно.

**Необходимо выполнить следующие требования:**

 Коллекцию из ЛР №5 заменить на ее потокобезопасный аналог.

 Операции обработки объектов коллекции должны быть реализованы с помощью Stream API с использованием лямбда-выражений.

 Объекты между клиентом и сервером должны передаваться в сериализованном виде.

 Объекты в коллекции, передаваемой клиенту, должны быть отсортированы по умолчанию.

 Получив запрос, сервер должен создавать отдельный поток, который должен формировать и отправлять ответ клиенту.

 Клиент должен корректно обрабатывать временную недоступность сервера.

 Обмен данными между клиентом и сервером должен осуществляться по протоколу UDP.

 На стороне сервера должен использоваться датаграммы а на стороне клиента - сетевой канал.

**Отчёт по работе должен содержать:**

1. Текст задания.
2. Диаграмма классов разработанной программы (как клиентского, так и серверного приложения).
3. Исходный код программы.
4. Выводы по работе.

**Вопросы к защите лабораторной работы:**

1. Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия.
2. Протокол TCP. Классы Socket и ServerSocket.
3. Протокол UDP. Классы DatagramSocket и DatagramPacket.
4. Передача данных по сети. Сериализация объектов.
5. Интерфейс Serializable. Объектный граф, сериализация и десериализация полей и методов.
6. Многопоточные программы. Концепции.
7. Класс Thread и интерфейс Runnable.
8. Состояние потока. Синхронизация потока.
9. Пакет java.util.concurrent. Интерфейс Lock и его реализации.
10. Атомарные операции.
11. Java Stream APi. Создание конвейеров. Промежуточные и терминальные операции.

**Исходный код:**

**#ServerUDP1.Java**

package serverudp;

import classes.Human;

import java.io.\*;

import java.net.\*;

import java.time.LocalDateTime;

import java.util.\*;

import java.util.concurrent.ConcurrentHashMap;

import java.util.logging.\*;

public class ServerUDP1 extends Thread{

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

DatagramSocket socket = new DatagramSocket(9999);

//System.out.println(InetAddress.getLocalHost().getHostName());

Set<Human> human = Collections.newSetFromMap(new ConcurrentHashMap<Human, Boolean>());

String fileName = "D:\\NetBeansProjects\\ServerUDP\\src\\serverudp\\HumanList.xml";

try {XmlFile.read(fileName, human);}

catch (FileNotFoundException ex) {

System.out.println("FILE INPUT NOT FOUND \n");

}

System.out.println(LocalDateTime.now());

System.out.print("SERVER IS ONLINE \n");

human.stream().forEach((p) -> System.out.println(p));

while (true){

human.stream().sorted();

byte[] buf = new byte[1024];

DatagramPacket packet = new DatagramPacket(buf, buf.length);

socket.receive(packet);

ServerThread serThread = new ServerThread(socket, packet, human, fileName);

new Thread(serThread).start();

}

}

}

class ServerThread implements Runnable {

private DatagramSocket socket;

private DatagramPacket packet;

private Set<Human> human;

private String fileName;

public ServerThread( DatagramSocket socket, DatagramPacket packet, Set<Human> human, String fileName){

this.socket = socket;

this.packet = packet;

this.human = human;

this.fileName = fileName;

}

@Override

public void run() {

String s = new String(packet.getData());

InetAddress IPAddress = packet.getAddress();

int port = packet.getPort();

System.out.println("FROM CLIENT #" + port + ": " + s);

//System.out.println(IPAddress);

String msg = "";

for (int i = 1; i<1024; i++){

if (s.substring(0, i).endsWith("}")) {

s = s.substring(0, i);

break;

}

}

Set<Human> list = Collections.newSetFromMap(new ConcurrentHashMap<Human, Boolean>());

System.out.println(s + " " + s.length());

if (s.startsWith("print")) {

list = human;

msg = "HUMAN LIST< NAME, SIZE, POSX, POSY, TIME>:";

}

if (s.startsWith("add ")) {

Human p = Human.jsonToObject(s.replaceFirst("add ", ""));

human.add(p);

msg = "ADDED TARGET";

}

if (s.startsWith("delete ")) {

Human p = Human.jsonToObject(s.replaceFirst("delete ", ""));

final Set<Human> delList = Collections.newSetFromMap(new ConcurrentHashMap<Human, Boolean>());

human.stream().filter(h -> h.equal(p)).forEach((Human h) ->

{System.out.println("Deleting " + h);

delList.add(h);

human.remove(h);

});

list = delList;

if (list.isEmpty()){

msg = "TARGET IS NOT IN THE LIST";

} else msg = "DELETED LIST< NAME, SIZE, POSX, POSY, TIME>:";

}

if (s.startsWith("smaller ")) {

Human p = Human.jsonToObject(s.replaceFirst("smaller ", ""));

human.stream().filter((findP) -> (p.getSize()>findP.getSize())).forEachOrdered(list::add);

if (list.isEmpty()) {

msg ="NO ONE IS SMALLER THAN " + p.getSize() ;

} else msg = "RESULT LIST< NAME, SIZE, POSX, POSY, TIME>:";

}

if (s.startsWith("save")) {

try {

XmlFile.write(fileName, human);

} catch (UnsupportedEncodingException ex) {

Logger.getLogger(ServerUDP1.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

Logger.getLogger(ServerUDP1.class.getName()).log(Level.SEVERE, null, ex);

}

msg = "SAVED LIST";

}

if (s.startsWith("load")) {

human = Collections.newSetFromMap(new ConcurrentHashMap<Human, Boolean>());

try {XmlFile.read(fileName, human);}

catch (FileNotFoundException ex) {

System.out.println("INPUT FILE NOT FOUND");

}

msg = "LOADED NEW LIST";

}

try {

sendStr(msg, IPAddress, port);

} catch (IOException ex) {

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

}

list.stream().forEach(h -> {

try {

sendOb(h, IPAddress, port);

} catch (IOException ex) {

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

}

});

}

public void sendOb(Human ob, InetAddress IPAddress, int port) throws IOException {

System.out.println(ob.toString());

ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

ObjectOutputStream os = new ObjectOutputStream(outputStream);

os.writeObject(ob);

os.flush();

byte[] data = outputStream.toByteArray();

DatagramPacket sendPacket = new DatagramPacket(data, data.length, IPAddress, port);

socket.send(sendPacket);

}

public void sendStr(String msg, InetAddress IPAddress, int port) throws IOException {

System.out.println(msg);

byte[] buf = new byte[1024];

buf = msg.getBytes();

DatagramPacket sendPacket = new DatagramPacket(buf, buf.length, IPAddress, port);

socket.send(sendPacket);

}

}

**#ClientUDP1.Java**

package clientudp;

import classes.Human;

import java.io.\*;

import java.net.\*;

import java.nio.ByteBuffer;

import java.nio.channels.\*;

import java.util.\*;

public class ClientUDP1 {

DatagramChannel client = null;

SocketAddress address = new InetSocketAddress((int) Math.random()\*9998);

SocketAddress serverAdd = new InetSocketAddress("localhost", 9999);

public ClientUDP1() throws IOException {

client = DatagramChannel.open();

client.bind(address);

client.socket().setSoTimeout(5000);

}

public static void main(String args[]) throws SocketException, UnknownHostException, IOException, ClassNotFoundException{

ClientUDP1 client1 = new ClientUDP1();

client1.StartRequest();

}

public void makeConnection() throws IOException {

client.connect(serverAdd);

client.configureBlocking(true);

}

public void StartRequest() throws IOException, SocketTimeoutException, ClassNotFoundException{

makeConnection();

Scanner sc = new Scanner(System.in);

System.out.println("HELLO, PLEASE MAKE A REQUEST!");

String input;

while (true){

input = sc.nextLine();

if (input.equals("exit")) {

client.close();

System.out.println("CONNECTION CLOSED");

break;

}

sendRequest(input);

receiveResult();

}

}

public void sendRequest(String msg) throws IOException {

//Datagram Socket Channel UDP

ByteBuffer buffer = ByteBuffer.wrap(msg.getBytes());

client.send(buffer, serverAdd);

}

public void receiveResult() throws IOException, SocketTimeoutException, ClassNotFoundException{

//receive msg

ByteBuffer buffer = ByteBuffer.allocateDirect(1024);

try{

client.receive(buffer);

buffer.flip();

int limitBuf = buffer.limit();

byte[] byteBuf = new byte[limitBuf];

buffer.get(byteBuf, 0, limitBuf);

String msg = new String(byteBuf);

System.out.println(msg);

} catch (PortUnreachableException ex){

System.out.println("SERVER IS NOT AVAILABLE AT THE MOMENT, PLEASE TRY AGAIN LATER");

}

//receive Object

byte[] incomingData = new byte[1024];

DatagramPacket packet = new DatagramPacket(incomingData, incomingData.length);

while (true){

try {

client.socket().receive(packet);

byte[] data = packet.getData();

ByteArrayInputStream in = new ByteArrayInputStream(data);

ObjectInputStream is = new ObjectInputStream(in);

Human h = (Human) is.readObject();

System.out.println(h);

} catch (SocketTimeoutException e) {

break;

}

catch (ClassNotFoundException e) {

System.out.println("CLASS HUMAN NOT FOUND");

break;

}

}

}

}

**#XmlFile.java**

package serverudp;

import classes.Human;

import java.io.\*;

import java.util.\*;

public class XmlFile {

public static void read(String fileName, Set<Human> human)throws FileNotFoundException{

File text = new File(fileName);

String s = "";

try (Scanner scanner = new Scanner(text)) {

s = scanner.nextLine();

while(scanner.hasNext()){

s = scanner.nextLine();

Human p = new Human(SplitString(scanner.next()),

SplitNum(scanner.next()),

SplitNum(scanner.next()),

SplitNum(scanner.next()),

SplitString(scanner.next()));

human.add(p);

s = scanner.nextLine();

s = scanner.nextLine();

}

scanner.close();

}

}

public static String SplitString(String s){

while (false==s.startsWith(">")){

s = s.substring(1,s.length()-1);

}

s = s.substring(1,s.length()-2);

return s;

}

public static int SplitNum(String s){

return Integer.parseInt(SplitString(s));

}

public static void write(String fileName, Set<Human> human)throws FileNotFoundException, UnsupportedEncodingException, IOException{

File text = new File(fileName);

try (OutputStreamWriter writer = new OutputStreamWriter(new FileOutputStream(fileName), "UTF-8")) {

writer.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>"+ "\n");

for (Human w : human){

writer.write("<Human>"+"\n");

writer.write("<name>"+w.getName()+"</name>"+"\n");

writer.write("<size>"+w.getSize()+"</size>"+"\n");

writer.write("<PosX>"+w.getPosX()+"</PosX>"+"\n");

writer.write("<PosY>"+w.getPosY()+"</PosY>"+"\n");

writer.write("<time>"+w.getTime()+"</time>"+"\n");

writer.write("</Human>"+"\n");

}

writer.close();

}

}

}

**#Human.java**

package classes;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

import java.time.LocalDateTime;

public class Human implements java.io.Serializable{

String name;

int size;

int PosX;

int PosY;

String time;

public Human(String name, int size, int PosX, int PosY, String time) {

this.name = name;

this.size = size;

this.PosX = PosX;

this.PosY = PosY;

this.time = time;

}

public String getName() {

return this.name;

}

public int getSize() {

return this.size;

}

public int getPosX() {

return this.PosX;

}

public int getPosY() {

return this.PosY;

}

public String getTime(){

return this.time;

}

public void setName(String name) {

this.name = name;

}

public void setSize(int size) {

this.size = size;

}

public void setPosX(int PosX) {

this.PosX = PosX;

}

public void setPosY(int PosY) {

this.PosY = PosY;

}

public void setTime(String time){

this.time = time;

}

@Override

public String toString() {

return name + " " + size + " " + PosX + " " + PosY + " " + time;

}

public int compareTo (Human p){

return (this.getName()+this.getSize()).compareTo(p.getName()+p.getSize());

}

public boolean equal(Human h){

if ((this.name.equals(h.name)) && (this.size == h.size)) return true;

return false;

}

public static Human jsonToObject(String s){

Human p = new Human("", 0, 0, 0, LocalDateTime.now().toString());

//System.out.println(LocalDateTime.now());

try {

s = s.toString();

//System.out.println(s + " " + s.length());

// 1. Create a JSONParser

JSONParser jsP = new JSONParser();

// 2. Parser string JSON into a JSONObject

JSONObject jsO = (JSONObject) jsP.parse(s);

// 3. Get values from jsonObject through Key

String name = (String) jsO.get("name");

if (name != null) p.setName(name);

try{

int size = Integer.parseInt((String) jsO.get("size"));

p.setSize(size);

} catch (NumberFormatException ex) {}

try{

int PosX = Integer.parseInt((String) jsO.get("PosX"));

p.setPosX(PosX);

} catch (NumberFormatException ex) {}

try{

int PosY = Integer.parseInt((String) jsO.get("PosY"));

p.setPosY(PosY);

} catch (NumberFormatException ex) {}

System.out.println(p.toString());

} catch (ParseException e) {

e.printStackTrace();

}

return p;

}

}