**ИТМО Кафедра Вычислительной техники**

Отчет по лабораторной работе №5

Программирование интернет-приложений

Вариант 1895

**Выполнил: студент группы P3217**

**Плюхин Дмитрий**

**Преподаватель: Гаврилов А. В.**

**2016 год**

1. **Задание к лабораторной работе**

Разделить приложение из лабораторной работы №4 на две составляющие - клиентскую и серверную, обменивающиеся сообщениями по заданному протоколу.

На стороне клиента осуществляются ввод и передача данных серверу, прием и отображение ответов от сервера и отрисовка области. В сообщении клиента должна содержаться вся необходимая информация для определения факта попадания/непопадания точки в область.

Сервер должен принимать сообщения клиента, обрабатывать их в соответствии с заданной областью и отправлять клиенту ответное сообщение, содержащее сведения о попадании/непопадании точки в область.

**Приложение должно удовлетворять следующим требованиям:**

* Для передачи сообщений необходимо использовать протокол TCP.
* Для данных в сообщении клиента должен использоваться тип double.
* Для данных в ответном сообщении сервера должен использоваться тип boolean.
* Каждое сообщение на сервере должно обрабатываться в отдельном потоке. Класс потока должен реализовывать интерфейс Runnable.
* Приложение должно быть локализовано на 2 языка - английский и сербский.
* Строки локализации должны храниться в отдельном классе.
* Приложение должно корректно реагировать на "потерю" и "восстановление" связи между клиентом и сервером; в случае недоступности сервера клиент должен показывать введённые пользователем точки серым цветом.

1. **Исходный код**

**//Файл ServerManager.java**

package net;

import jswing.Ponto;

import java.net.ServerSocket;

import java.net.Socket;

import java.net.SocketAddress;

import java.util.LinkedHashSet;

import java.util.Set;

public class ServerManager {

private static Server server = null;

public static void main(String[] args) {

server = new Server(6666);

server.start();

}

}

**//Файл Server.java**

package net;

import java.io.IOException;

import java.net.\*;

public class Server {

private final int port;

private ServerSocket serverSocket;

private Thread listenThread;

private boolean running = false;

public Server(int port) {

this.port = port;

try {

serverSocket = new ServerSocket(port);

} catch (IOException e) {

e.printStackTrace();

}

}

public void start(){

running = true;

listenThread = new Thread(() -> listen());

listenThread.start();

}

public void stop(){

running = false;

//closeServerSocket();

}

private void closeServerSocket(){

try {

serverSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private void listen(){

while(running) {

Socket socket;

try {

socket = serverSocket.accept();

new Thread(new ClientSession(socket)).start();

} catch (IOException e) {

System.out.println("[Server] Client disconnected");

}

}

}

public int getPort(){

return port;

}

public SocketAddress getSocketAddress(){

return serverSocket.getLocalSocketAddress();

}

}

**//Файл ClientSession.java**

package net;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.Socket;

public class ClientSession implements Runnable{

private Socket socket;

private InputStream in;

private OutputStream out;

public ClientSession(Socket socket) throws IOException{

this.socket = socket;

in = socket.getInputStream();

out = socket.getOutputStream();

}

@Override

public void run() {

Request request;

while(true) {

System.out.println("[Server] Receiving request...");

request = Channel.receiveRequest(socket);

if (request == null){

return;

}

System.out.println("[Server] Received : "+request.getX()+" "+request.getY());

System.out.println("[Server] Processing request...");

new Thread(new ClientProcess(socket,request)).start();

}

}

**//Файл ClientProcess.java**

package net;

import jswing.GeneralSilhouette;

import jswing.Ponto;

import jswing.Silhouette;

import java.net.Socket;

public class ClientProcess implements Runnable{

private Socket socket;

private Request request;

public ClientProcess(Socket socket, Request request){

this.socket = socket;

this.request = request;

}

@Override

public void run() {

double X = request.getX();

double Y = request.getY();

double R = request.getR();

System.out.println("[Server] Checking ponto in request...");

boolean supremumIudicium = new GeneralSilhouette(R).checkPonto(new Ponto(X,Y));

Response response = new Response(supremumIudicium);

Channel.sendResponse(response, socket);

}

}

**//Файл Channel.java**

package net;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.Socket;

import java.net.SocketException;

public class Channel {

public static void sendResponse(Response response, Socket socket){

try {

OutputStream out = socket.getOutputStream();

response.writeTo(out);

} catch (IOException e) {

e.printStackTrace();

}

}

public static void sendRequest(Request request, Socket socket) throws SocketException{

OutputStream out = null;

try {

out = socket.getOutputStream();

}catch (SocketException e){

throw e;

}catch (IOException e) {

e.printStackTrace();

}

request.writeTo(out);

}

public static Response receiveResponse(Socket socket) throws SocketException{

Response response = new Response(false);

System.out.println("Connection closed : "+socket.isClosed());

try {

InputStream in = socket.getInputStream();

response = Response.readFrom(in);

} catch (SocketException e){

throw e;

}

catch (IOException e) {

e.printStackTrace();

}

return response;

}

public static Request receiveRequest(Socket socket){

Request request = new Request(0.0,0.0,0.0);

try {

InputStream in = socket.getInputStream();

request = Request.readFrom(in);

} catch (IOException e) {

System.out.println("[Server] Client disconnected");

}

return request;

}

}

**//Файл Request.java**

package net;

import java.io.\*;

import java.net.SocketException;

public class Request implements Serializable{

private double X;

private double Y;

private double R;

public Request(double x, double y, double r){

X = x;

Y = y;

R = r;

}

public void writeTo(OutputStream out) throws SocketException{

try {

ObjectOutputStream serializer = new ObjectOutputStream(out);

serializer.writeObject(this);

} catch (SocketException e){

System.out.println("Connection reset");

throw e;

} catch (IOException e) {

e.printStackTrace();

}

}

public static Request readFrom(InputStream in){

Request request = null;

try {

ObjectInputStream deserializer = new ObjectInputStream(in);

request = (Request)deserializer.readObject();

} catch (IOException e) {

System.out.println("[Server] Client disconnected");

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

return request;

}

public double getX(){

return X;

}

public double getY(){

return Y;

}

public double getR(){

return R;

}

}

**//Файл Response.java**

package net;

import java.io.\*;

import java.net.SocketException;

public class Response implements Serializable{

private boolean supremumIudicium;

public void writeTo(OutputStream out){

try {

ObjectOutputStream serializer = new ObjectOutputStream(out);

serializer.writeObject(this);

} catch (IOException e) {

e.printStackTrace();

}

}

public static Response readFrom(InputStream in) throws SocketException{

Response response = null;

try {

ObjectInputStream deserializer = new ObjectInputStream(in);

response = (Response)deserializer.readObject();

} catch (SocketException e){

System.out.println("Connection reset");

throw e;

}

catch (IOException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

return response;

}

public Response(boolean supremumIudicium){

this.supremumIudicium = supremumIudicium;

}

public boolean getSupremumIudicium(){

return supremumIudicium;

}

}

**//Файл PontoCheckingSuperstructure.java**

package net;

import jswing.Lab4;

import jswing.Ponto;

import java.io.IOException;

import java.net.Socket;

import java.net.SocketAddress;

import java.net.SocketException;

import java.net.SocketTimeoutException;

import java.util.ConcurrentModificationException;

import java.util.Set;

public class PontoCheckingSuperstructure {

Lab4 lab4;

Thread checking;

Set<Ponto> pontos;

Socket socket;

SocketAddress socketAddress;

boolean changesShowed = true;

public PontoCheckingSuperstructure(Set<Ponto> pontos, SocketAddress socketAddress, Lab4 lab4){

this.pontos = pontos;

this.socketAddress = socketAddress;

this.socket = new Socket();

this.lab4 = lab4;

checking = new Thread(() -> circuloInferni());

checking.start();

}

private void circuloInferni(){

while(true){

try {

pontos.stream().filter((Ponto p) -> !p.isChecked()).forEach((Ponto p) -> {

System.out.println("Checking...");

Request request;

request = new Request(p.getX(), p.getY(), lab4.getR());

boolean sent = false;

do {

try {

System.out.println("Trying to send request...");

Channel.sendRequest(request, socket);

sent = true;

System.out.println("Successful!");

} catch (SocketException e) {

System.out.println("Failed!");

tryConnect();

}

} while (!sent);

System.out.println("Waiting for response...");

Response response = null;

try {

response = Channel.receiveResponse(socket);

} catch (SocketException e) {

System.out.println("Unable to receive response");

return;

}

System.out.println("Response submitted!");

p.checkOn(response.getSupremumIudicium());

setChangesShowed(false);

System.out.println("Checked! For the " + p.getX() + " " + p.getY() + " it is " + response.getSupremumIudicium());

});

if (!changesShowed){

lab4.redrawPontos();

changesShowed = true;

}

} catch (ConcurrentModificationException e){

System.out.println("Collection changed until iterating");

}

}// while

}

private void setChangesShowed(boolean value){

changesShowed = value;

}

private void tryConnect(){

assert(!socket.isConnected());

boolean connected = false;

boolean firstTry = true;

do{

if (!firstTry){

try{

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

firstTry = false;

}

socket = new Socket();

try {

System.out.println("Trying to connect...");

try {

socket.connect(socketAddress, 1000);

connected = true;

} catch (SocketTimeoutException e){

System.out.println("Connection timed out...");

}

} catch (IOException e) {

e.printStackTrace();

}

} while(!connected);

System.out.println("Connected!");

}

}

**//Файл Resources\_eng.properties**

error=Error

animation\_interrupted=Animation thread interrupted

title=Fourth Lab

select\_x=Please, select x :

select\_y=Please, select y :

select\_r=Please, select R :

selected\_point=Selected point :

none=None

**//Файл Resources\_serb.properties**

error=Грешка

animation\_interrupted=Анимација прекинута

title=Peti Лаб

select\_x=Молимо одаберите x :

select\_y=Молимо одаберите y :

select\_r=Молимо одаберите R :

selected\_point=Изабрана тачка :

none=Ниједан

**//Файл Lab4.java (приведен не полностью)**

package jswing;

...

public class Lab4 extends JFrame{

public static PropertyResourceBundle localization = null;

...

PontoCheckingSuperstructure pcss = null;

public static void main(String[] args) {

new Lab4();

}

static {

try {

localization = new PropertyResourceBundle(new FileReader("L10n\\Resources\_serb.properties"));

} catch (IOException e) {

e.printStackTrace();

}

}

public Lab4()

{

...

try {

pcss = new PontoCheckingSuperstructure(pontos,new InetSocketAddress(InetAddress.getByName("127.0.0.1"),6666),this);

} catch (UnknownHostException e) {

e.printStackTrace();

}

}

...

public double getR(){

return R;

}

public void checkOffAll(){

for(Ponto p : pontos){

p.checkOff();

}

}

}

**//Файл GraphPanel.java (приведен не полностью)**

package jswing;

...

public class GraphPanel extends JPanel {

...

private boolean addPontoToGraph(Ponto p, Graphics g, double R){

boolean inArea = false;

if (p.isChecked()) {

if (p.isInSilhouette()) {

g.setColor(INNER\_POINT\_COLOR);

inArea = true;

} else {

g.setColor(OUTER\_POINT\_COLOR);

}

} else {

g.setColor(NOTCHECKED\_POINT\_COLOR);

}

g.fillOval((int)p.getGraphX(R)-SIZE\_OF\_POINT,(int)p.getGraphY(R)-SIZE\_OF\_POINT,SIZE\_OF\_POINT\*2,SIZE\_OF\_POINT\*2);

return inArea;

}

...

}

1. **Вывод**

Таким образом, при организации клиент-серверной архитектуры на Java ключевыми объектами при использовании протокола TCP являются ServerSocket и Socket, а при использовании UDP – DatagramSocket и DatagramPacket, причем достаточно затруднительно ответить, какой протокол является более предпочтительным для универсального использования. Я узнал, что для локализации приложения следует использовать специальные классы, наиболее удобным из которых, по моему мнению, является PropertyResourceBundle в силу простоты модификации уже существующих ресурсов – мы должны поменять всего несколько строк в .property-файле. Однако этот класс следует использовать с умом, поскольку при попытке доступа к .property-файлу без применения FileReader можно приобрести проблемы, связанные с кодировкой. Кроме всего прочего, я познакомился с классами, предназначенными для форматирования дат и чисел, и сделал вывод о том, что это в достаточной мере полезные ресурсы, которые могут стать незаменимыми помощниками при локализации приложения.