**Университет ИТМО**

**Кафедра ВТ**

Лабораторная работа #5

Выполнили

Халанский Дмитрий, гр. 2125

Назарьев Сергей, гр. 2125

СПб, 2014

****

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 int.
* Каждое сообщение на сервере должно обрабатываться в отдельном потоке. Класс потока должен быть унаследован от класса Thread.
* Приложение должно быть локализовано на 2 языка - русский и сербский.
* Строки локализации должны храниться в текстовом файле.
* 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.

**Вопросы к защите лабораторной работы:**

1. Интернационализация и локализация.
2. Способы хранения локализованных данных. Классы ResourceBundle, PropertyResourceBundle, ListResourceBundle.
3. Форматирование числовых данных. Классы NumberFormat, DecimalFormat, DecimalFormatSymbols.
4. Форматирование даты и времени. Классы DateFormat, SimpleDateFormat, DateFormatSymbols.
5. Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия.
6. Протокол TCP. Классы Socket и ServerSocket.
7. Протокол UDP. Классы DatagramSocket и DatagramPacket.
8. Передача данных по сети. Сериализация объектов, интерфейс Serializable.

**import** java.util.List;

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.Set;

**import** java.awt.Polygon;

**import** java.awt.EventQueue;

**import** java.awt.GridLayout;

**import** java.awt.event.**\***;

**import** javax.swing.JScrollPane;

**import** javax.swing.ListSelectionModel;

**import** javax.swing.event.**\***;

**import** javax.swing.JFrame;

**import** javax.swing.JList;

**import** javax.swing.ButtonGroup;

**import** javax.swing.JRadioButton;

**import** javax.swing.JSpinner;

**import** javax.swing.JTextArea;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** javax.swing.JPanel;

**import** java.net.**\***;

**import** java.io.**\***;

**import** java.util.**\***;

**public** **class** Lab5 **extends** JFrame {

 **private** **Point** guiPoint = **new** **Point**(0,0);

 **private** Plot plot;

 **private** JTextArea textArea;

 **private** JSpinner spinner;

 **private** JRadioButton y0;

 **private** JRadioButton y1;

 **private** JList<**String**> list;

 **private** boolean busy = **false**;

 **private** SingleElementBuffer<LabMessage> messageBuffer = **new** SingleElementBuffer<LabMessage>();

 **private** SingleElementBuffer<**Integer**> resultBuffer = **new** SingleElementBuffer<**Integer**>();

 **private** **Socket** socket;

 **private** **DataInputStream** input = **null**;

 **private** **DataOutputStream** output = **null**;

 **static** **class** LabMessage {

 **private** double x;

 **private** double y;

 **private** double r;

 **public** LabMessage(double x, double y, double r) {

 **this**.x = x;

 **this**.y = y;

 **this**.r = r;

 }

 **public** double getX() { **return** **this**.x; }

 **public** double getY() { **return** **this**.y; }

 **public** double getR() { **return** **this**.r; }

 **public** int hashCode() {

 **return** **String**.format("%f:%f:%f", x, y, r).hashCode();

 }

 **public** boolean equals(**Object** message) {

 **if**(!(message **instanceof** LabMessage)) **return** **false**;

 LabMessage op = (LabMessage)message;

 **return** (op.getX() == x && op.getY() == y && op.getR() == r);

 }

 }

 **static** **class** SingleElementBuffer<T> {

 **private** T element = **null**;

 **public** **synchronized** void put(T element) **throws** **InterruptedException** {

 **while**(**this**.element != **null**)

 **this**.wait();

 **this**.element = element;

 **this**.notifyAll();

 }

 **public** **synchronized** T get() **throws** **InterruptedException** {

 **while**(**this**.element == **null**)

 **this**.wait();

 T result = **this**.element;

 **this**.element = **null**;

 **this**.notifyAll();

 **return** result;

 }

 }

 **static** **class** Point {

 **private** int x;

 **private** int y;

 **public** **Point**(int x, int y) {

 **this**.x = x;

 **this**.y = y;

 }

 **public** void setX(int x) { **this**.x = x; }

 **public** void setY(int y) { **this**.y = y; }

 **public** int getX() { **return** **this**.x; }

 **public** int getY() { **return** **this**.y; }

 }

 **interface** PlotListener {

 **public** void plotRefreshed();

 **public** void plotAnimationStarted();

 **public** void plotAnimationFinished();

 }

 **interface** SocketListener {

 **public** void socketConnected();

 }

 **static** **class** Resolver {

 **private** SingleElementBuffer<LabMessage> messageBuffer;

 **private** SingleElementBuffer<**Integer**> resultBuffer;

 **private** Map<LabMessage, **Integer**> resultMap = **new** HashMap<LabMessage, **Integer**>();

 **public** Resolver(SingleElementBuffer<LabMessage> messageBuffer, SingleElementBuffer<**Integer**> resultBuffer) {

 **this**.messageBuffer = messageBuffer;

 **this**.resultBuffer = resultBuffer;

 }

 **public** int statusPoint(**Point** point, int r) {

 LabMessage message = **new** LabMessage((double)point.getX(), (double)point.getY(), (double)r);

 **if**(resultMap.containsKey(message)) **return** resultMap.get(message);

 int result = -1;

 **try** {

 messageBuffer.put(message);

 result = resultBuffer.get();

 **if**(result != -1) resultMap.put(message, result);

 } **catch**(**InterruptedException** e) {

 e.printStackTrace();

 }

 **return** result;

 }

 }

 **static** **class** Plot **extends** JPanel {

 **public** **static** **final** **Color** COLOR\_BLACK = **new** **Color**(0,0,0);

 **public** **static** **final** **Color** COLOR\_LIGHTGREEN = **new** **Color**(124,252,0);

 **public** **static** **final** **Color** COLOR\_BROWN = **new** **Color**(150,75,0);

 **public** **static** **final** int POINT\_RADIUS = 5;

 **public** **static** **Color** areaColor = COLOR\_BROWN;

 **public** **static** **final** **Color** COLOR\_WHITE = **new** **Color**(255,255,255);

 **public** **static** **final** **Color** COLOR\_GRAY= **new** **Color**(128,128,128);

 **private** int centerX;

 **private** int centerY;

 **private** int R = 40;

 **private** Resolver resolver;

 **public** Plot(Resolver resolver) {

 **this**.resolver = resolver;

 }

 **public** int getR() { **return** R; }

 **public** void setR(int R) {

 int oldR = **this**.R;

 **this**.R = (R > 0 ? R : -R);

 **for**(**Point** point : points) {

 **if**(resolver.statusPoint(point, oldR) == 1 && resolver.statusPoint(point, **this**.R) == 0)

 {

 startAnimation();

 **break**;

 }

 }

 refresh();

 }

 **public** int getCenterX() { **return** centerX; }

 **public** int getCenterY() { **return** centerY; }

 **private** Set<PlotListener> listeners = **new** HashSet<PlotListener>();

 **public** void subscribe(PlotListener listener) {

 listeners.add(listener);

 }

 **public** void unsubscribe(PlotListener listener) {

 listeners.remove(listener);

 }

 **private** **List**<**Point**> points = **new** ArrayList<**Point**>();

 **public** **List**<**Point**> getPoints() {

 **return** points;

 }

 **public** void addPoint(**Point** point) {

 points.add(point);

 refresh();

 }

 **public** void startAnimation() {

 **new** **Thread**(**new** Runnable() {

 **public** void run() {

 **try** {

 int R0 = COLOR\_BROWN.getRed();

 int G0 = COLOR\_BROWN.getGreen();

 **final** int B0 = COLOR\_BROWN.getBlue();

 **final** int N = 50;

 int R = R0;

 int G = G0;

 int B = B0;

 **for**(int i = 0; i < N; i++) {

 R += (255-R0)/N;

 G += (255-G0)/N;

 B += (255-B0)/N;

 areaColor = **new** **Color**(R,G,B);

 repaint();

 **Thread**.sleep(1000/60);

 }

 **for**(int i = 0; i < N; i++) {

 R -= (255-R0)/N;

 G -= (255-G0)/N;

 B -= (255-B0)/N;

 areaColor = **new** **Color**(R,G,B);

 repaint();

 **Thread**.sleep(1000/60);

 }

 **for**(PlotListener listener : listeners)

 listener.plotAnimationFinished();

 } **catch**(**InterruptedException** e) {

 e.printStackTrace();

 }

 }

 }).start();

 **for**(PlotListener listener : listeners)

 listener.plotAnimationStarted();

 }

 **public** void refresh() {

 repaint();

 **for**(PlotListener listener : listeners)

 listener.plotRefreshed();

 }

 **private** void paintArea(**Graphics** g, **Color** color) {

 g.setColor(color);

 **for**(int x = 0; x < getWidth(); x++)

 **for**(int y = 0; y < getHeight(); y++)

 **if**(containsPointInBatman((double)(x - centerX), (double)(centerY - y), (double)R))

 g.drawLine((int)**Math**.floor(x), (int)**Math**.floor(y), (int)**Math**.floor(x), (int)**Math**.floor(y));

 }

 **public** **static** boolean f(double x, double y) {

 **if** (**Math**.pow(x, 2.0) / 49.0 + **Math**.pow(y, 2.0) / 9.0 - 1.0 <= 0 && **Math**.abs(x) >= 4.0 && -(3.0 \* **Math**.sqrt(33.0)) / 7.0 <= y && y

 <= 0) {

 **return** **true**;

 }

 **if** (**Math**.pow(x, 2.0) / 49.0 + **Math**.pow(y, 2.0) / 9.0 - 1.0 <= 0 && **Math**.abs(x) >= 3.0 && -(3.0 \* **Math**.sqrt(33.0)) / 7.0 <= y && y

 >= 0) {

 **return** **true**;

 }

 **if** (-3.0 <= y

 && y <= 0

 && -4.0 <= x

 && x <= 4.0

 && (**Math**.abs(x)) / 2.0 + **Math**.sqrt(1.0 - **Math**.pow(**Math**.abs(**Math**.abs(x) - 2.0) - 1.0, 2.0)) - 1.0 / 112.0 \* (3.0 \* **Math**.sqrt(33.0) - 7.0)

 \* **Math**.pow(x, 2.0) - y - 3.0 <= 0) {

 **return** **true**;

 }

 **if** (y >= 0 && 3.0 / 4.0 <= **Math**.abs(x) && **Math**.abs(x) <= 1.0 && -8.0 \* **Math**.abs(x) - y + 9.0 >= 0) {

 **return** **true**;

 }

 **if** (1.0 / 2.0 <= **Math**.abs(x) && **Math**.abs(x) <= 3.0 / 4.0 && 3.0 \* **Math**.abs(x) - y + 3.0 / 4.0 >= 0 && y >= 0) {

 **return** **true**;

 }

 **if** (**Math**.abs(x) <= 1.0 / 2.0 && y >= 0 && 9.0 / 4.0 - y >= 0) {

 **return** **true**;

 }

 **if** (**Math**.abs(x) >= 1.0

 && y >= 0

 && -(**Math**.abs(x)) / 2.0 - 3.0 / 7.0 \* **Math**.sqrt(10.0) \* **Math**.sqrt(4.0 - **Math**.pow(**Math**.abs(x) - 1.0, 2.0)) - y + (6.0 \* **Math**.sqrt(10.0)) / 7.0

 + 3.0 / 2.0 >= 0) {

 **return** **true**;

 }

 **return** **false**;

 }

 **public** **static** boolean containsPointInBatman(double x, double y, double r) {

 **return** f(x/r\*7,y/r\*7);

 }

 **protected** void paintComponent(**Graphics** graphics) {

 **super**.paintComponent(graphics);

 graphics.setColor(COLOR\_LIGHTGREEN);

 graphics.fillRect(0,0,getWidth(),getHeight());

 centerX = getWidth() >> 1;

 centerY = getHeight() >> 1;

 paintArea(graphics, areaColor);

 graphics.setColor(COLOR\_BLACK);

 graphics.drawLine(0,centerY,getWidth(),centerY);

 graphics.drawLine(centerX,0,centerX,getHeight());

 **for**(**Point** point : points) {

 **switch**(resolver.statusPoint(point, R)) {

 **case** -1:

 graphics.setColor(COLOR\_GRAY);

 **break**;

 **case** 0:

 graphics.setColor(COLOR\_BLACK);

 **break**;

 **case** 1:

 graphics.setColor(COLOR\_WHITE);

 **break**;

 }

 graphics.fillOval(centerX + point.getX()-(POINT\_RADIUS>>1), centerY - point.getY()-(POINT\_RADIUS>>1),POINT\_RADIUS,POINT\_RADIUS);

 }

 }

 }

 **public** void AddList() {

 int[] xSet = **new** int[] {0,3,8};

 list = **new** JList<**String**>(**new** **String**[] {"X = 0", "X = 3", "X = 8"});

 list.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

 list.setSelectedValue(list.getModel().getElementAt(0), **false**);

 list.addListSelectionListener(**new** ListSelectionListener() {

 **public** void valueChanged(ListSelectionEvent e) {

 guiPoint.setX(xSet[list.getSelectedIndex()]);

 plot.refresh();

 }

 });

 add(list);

 }

 **public** void AddRadioButtons() {

 **GridLayout** yLayout = **new** **GridLayout**(0,1);

 JPanel yPanel = **new** JPanel();

 yPanel.setLayout(yLayout);

 ButtonGroup yGroup = **new** ButtonGroup();

 y0 = **new** JRadioButton("Y = 0");

 y1 = **new** JRadioButton("Y = 32");

 y0.setSelected(**true**);

 **ActionListener** actionListener = **new** ActionListener() {

 **public** void actionPerformed(**ActionEvent** e) {

 **if**(e.getSource() **instanceof** JRadioButton) {

 JRadioButton radioButton = (JRadioButton) e.getSource();

 **if**(radioButton.isSelected()) {

 **if**(radioButton == y0)

 guiPoint.setY(0);

 **else** **if**(radioButton == y1)

 guiPoint.setY(32);

 plot.refresh();

 }

 }

 }

 };

 yGroup.add(y0);

 y0.addActionListener(actionListener);

 y1.addActionListener(actionListener);

 yGroup.add(y1);

 yPanel.add(y0);

 yPanel.add(y1);

 add(yPanel);

 }

 **public** void AddSpinner() {

 spinner = **new** JSpinner();

 ChangeListener listener = **new** ChangeListener() {

 **public** void stateChanged(ChangeEvent e) {

 **if**(e.getSource() **instanceof** JSpinner) {

 JSpinner spinner = (JSpinner)e.getSource();

 **String** data = spinner.getValue().toString();

 plot.setR(**new** **Integer**(data));

 }

 }

 };

 spinner.setValue(40);

 spinner.addChangeListener(listener);

 add(spinner);

 }

 **public** void AddTextArea() {

 textArea = **new** JTextArea();

 plot.subscribe(**new** PlotListener() {

 **public** void plotRefreshed() {

 StringBuilder builder = **new** StringBuilder();

 **for**(**Point** point : plot.getPoints())

 builder.append("{" + point.getX() + "," + point.getY() + "}\n");

 textArea.setText(builder.toString());

 }

 **public** void plotAnimationStarted() {}

 **public** void plotAnimationFinished() {}

 });

 JScrollPane scroll = **new** JScrollPane(textArea, JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

 textArea.setEditable(**false**);

 add(scroll);

 }

 **public** void AddPlot(Resolver resolver) {

 plot = **new** Plot(resolver);

 plot.addPoint(guiPoint);

 plot.addMouseListener(**new** MouseAdapter() {

 @Override

 **public** void mousePressed(**MouseEvent** e) {

 plot.addPoint(**new** **Point**(e.getX() - plot.getCenterX(), plot.getCenterY() - e.getY()));

 }

 @Override

 **public** void mouseReleased(**MouseEvent** e) {

 }

 });

 add(plot);

 }

 **public** void PrepareFrame() {

 setLayout(**new** **GridLayout**(0,2));

 **ResourceBundle** titleBundle = **ResourceBundle**.getBundle("Lab5", **Locale**.getDefault());

 setTitle(titleBundle.getString("title"));

 setTitle("Lab5");

 setSize(600, 400);

 setResizable(**false**);

 setLocationRelativeTo(**null**);

 setDefaultCloseOperation(EXIT\_ON\_CLOSE);

 }

 **public** void ProcessMessages() {

 **new** **Thread**(**new** Runnable() {

 **public** void run() {

 **try** {

 LabMessage message = **null**;

 **while**((message = messageBuffer.get()) != **null**) {

 int result = -1;

 **if**(socket != **null** && !socket.isClosed() && input != **null** && output != **null**) {

 **try** {

 output.writeDouble(message.getX());

 output.writeDouble(message.getY());

 output.writeDouble(message.getR());

 result = input.readInt();

 } **catch**(**Exception** e) {

 }

 }

 resultBuffer.put(result);

 }

 } **catch**(**InterruptedException** e) {

 e.printStackTrace();

 }

 }

 }).start();

 }

 **public** void ConnectToServer() {

 **new** **Thread**(**new** Runnable() {

 **public** void run() {

 **while**(**true**) {

 **System**.out.println("Connecting.");

 **try** {

 **if**(output != **null**) output.flush();

 socket = **new** **Socket**();

 socket.connect(**new** InetSocketAddress(**InetAddress**.getByName("127.0.0.1"), 1927), 150);

 input = **new** **DataInputStream**(socket.getInputStream());

 output = **new** **DataOutputStream**(socket.getOutputStream());

 plot.refresh();

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

 } **catch**(**IOException** e) {

 **System**.out.println("Disconnected.");

 }

 **try** {

 **System**.out.println("Sleeping.");

 **Thread**.sleep(5000);

 } **catch**(**Exception** e) {

 e.printStackTrace();

 }

 }

 }

 }).start();

 }

 **public** Lab5() {

 ProcessMessages();

 Resolver resolver = **new** Resolver(messageBuffer, resultBuffer);

 AddPlot(resolver);

 ConnectToServer();

 AddList();

 AddRadioButtons();

 AddSpinner();

 AddTextArea();

 PrepareFrame();

 plot.subscribe(**new** PlotListener() {

 **public** void plotRefreshed() {

 }

 **public** void plotAnimationStarted() {

 plot.setEnabled(**false**);

 list.setEnabled(**false**);

 spinner.setEnabled(**false**);

 y0.setEnabled(**false**);

 y1.setEnabled(**false**);

 }

 **public** void plotAnimationFinished() {

 plot.setEnabled(**true**);

 list.setEnabled(**true**);

 spinner.setEnabled(**true**);

 y0.setEnabled(**true**);

 y1.setEnabled(**true**);

 }

 });

 plot.refresh();

 }

 **public** **static** void main(**String**[] args) {

 **EventQueue**.invokeLater(**new** Runnable() {

 @Override

 **public** void run() {

 Lab5 ex = **new** Lab5();

 ex.setVisible(**true**);

 }

 });

 }

}

**import** java.util.**\***;

**import** java.net.**\***;

**import** java.io.**\***;

**public** **class** Lab5Server {

 **static** **class** Contorno {

**public** **static** boolean f(double x, double y) {

// Wings bottom

**if** (**Math**.pow(x, 2.0) / 49.0 + **Math**.pow(y, 2.0) / 9.0 - 1.0 <= 0 && **Math**.abs(x) >= 4.0 && -(3.0 \* **Math**.sqrt(33.0)) / 7.0 <= y && y

<= 0) {

**return** **true**;

}

// Wings top (with the fix of the formula there are missing parenthesis in the original

**if** (**Math**.pow(x, 2.0) / 49.0 + **Math**.pow(y, 2.0) / 9.0 - 1.0 <= 0 && **Math**.abs(x) >= 3.0 && -(3.0 \* **Math**.sqrt(33.0)) / 7.0 <= y && y

>= 0) {

**return** **true**;

}

// Tail

**if** (-3.0 <= y

&& y <= 0

&& -4.0 <= x

&& x <= 4.0

&& (**Math**.abs(x)) / 2.0 + **Math**.sqrt(1.0 - **Math**.pow(**Math**.abs(**Math**.abs(x) - 2.0) - 1.0, 2.0)) - 1.0 / 112.0 \* (3.0 \* **Math**.sqrt(33.0) - 7.0)

\* **Math**.pow(x, 2.0) - y - 3.0 <= 0) {

**return** **true**;

}

// Ears outside

**if** (y >= 0 && 3.0 / 4.0 <= **Math**.abs(x) && **Math**.abs(x) <= 1.0 && -8.0 \* **Math**.abs(x) - y + 9.0 >= 0) {

**return** **true**;

}

// Ears inside

**if** (1.0 / 2.0 <= **Math**.abs(x) && **Math**.abs(x) <= 3.0 / 4.0 && 3.0 \* **Math**.abs(x) - y + 3.0 / 4.0 >= 0 && y >= 0) {

**return** **true**;

}

// Chest

**if** (**Math**.abs(x) <= 1.0 / 2.0 && y >= 0 && 9.0 / 4.0 - y >= 0) {

**return** **true**;

}

// Shoulders

**if** (**Math**.abs(x) >= 1.0

&& y >= 0

&& -(**Math**.abs(x)) / 2.0 - 3.0 / 7.0 \* **Math**.sqrt(10.0) \* **Math**.sqrt(4.0 - **Math**.pow(**Math**.abs(x) - 1.0, 2.0)) - y + (6.0 \* **Math**.sqrt(10.0)) / 7.0

+ 3.0 / 2.0 >= 0) {

**return** **true**;

}

**return** **false**;

}

 **public** **static** boolean containsPoint(double x, double y, double r) {

 **return** f(x/r\*7,y/r\*7);

 }

 }

 **static** **class** SocketProcessingThread **extends** **Thread** {

 **private** **Socket** socket;

 **public** SocketProcessingThread(**Socket** socket) {

 **this**.socket = socket;

 }

 **public** void run() {

 **try** {

 **DataInputStream** input = **new** **DataInputStream**(socket.getInputStream());

 **DataOutputStream** output = **new** **DataOutputStream**(socket.getOutputStream());

 **while**(**true**) {

 double x = input.readDouble();

 double y = input.readDouble();

 double r = input.readDouble();

 output.writeInt(Contorno.containsPoint((int)x, (int)y, (int)r) ? 1 : 0);

 }

 } **catch**(**Exception** e) {

 }

 }

 }

 **public** **static** void main(**String**[] args) {

 **try** {

 **ServerSocket** ss = **new** **ServerSocket**(1927);

 **Socket** socket = **null**;

 **while**((socket = ss.accept()) != **null**)

 **new** SocketProcessingThread(socket).start();

 } **catch**(**IOException** e) {

 e.printStackTrace();

 }

 }

}

**Вывод:** В результате проделанной работы мы изучили основы сетевого межпроцессового взаимодействия, поддерживаемого языком Java, главные различия сетевых протоколов TCP и UDP, а также наиболее частотно используемые инструменты интернационализации приложений, доступные в Java.