**Санкт-Петербургский национальный исследовательский университет**

**информационных технологий, механики и оптики**

**Кафедра вычислительной техники**

Программирование Интернет-Приложений

Лабораторная работа №5

Вариант 901

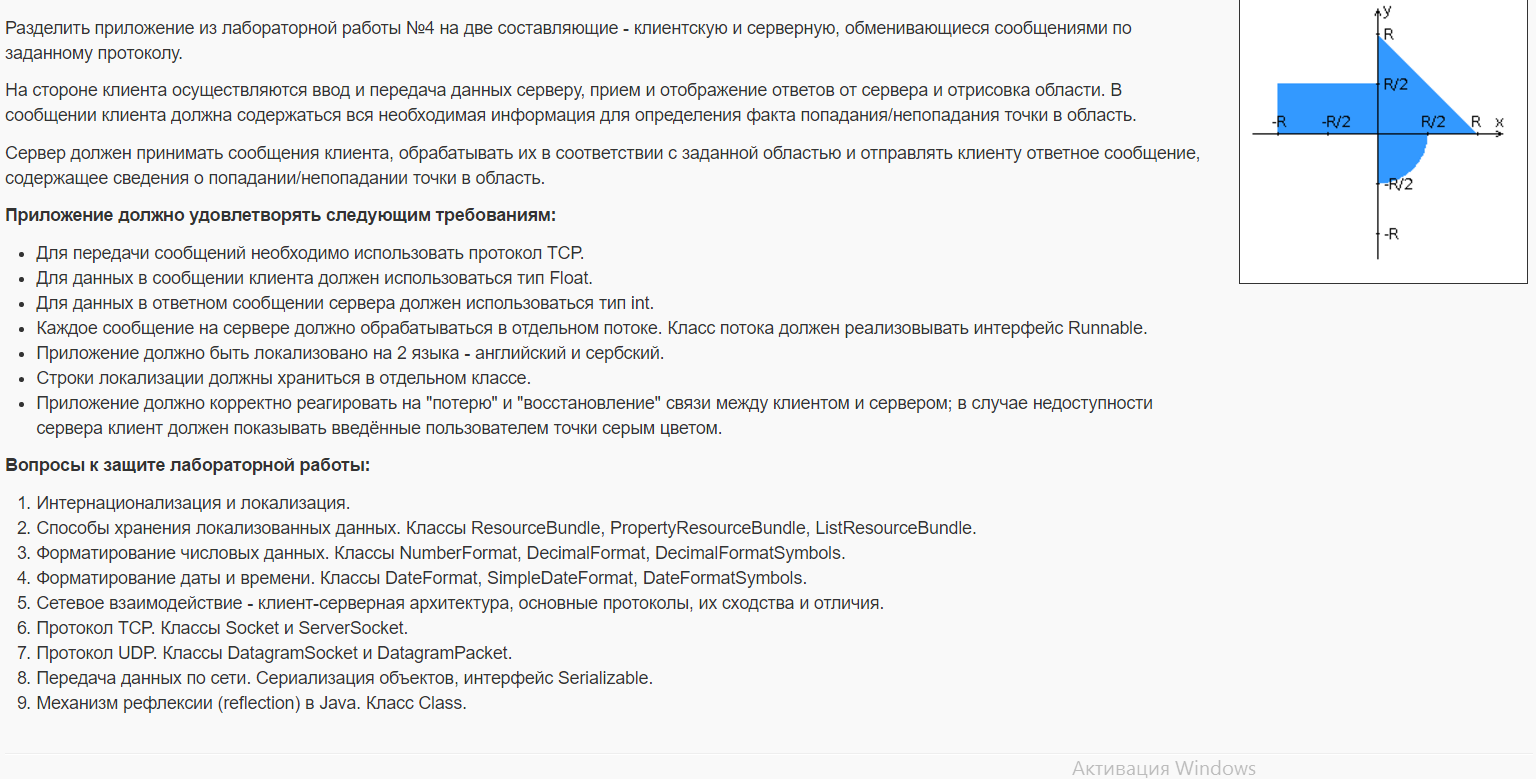
Выполнил: Гхази Даниэль

Группа P3218

Преподаватель: Гаврилов А.В.

2017 г.

**Текст задания**



**Текст программы**

**import** javafx.scene.control.ComboBox;

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

**import** java.awt.List;

**import** java.awt.event.WindowEvent;

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

**import** javax.swing.event.ChangeEvent;

**import** javax.swing.event.ChangeListener;

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

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.net.InetAddress;

**import** java.net.InetSocketAddress;

**import** java.net.Socket;

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

**public class** Lab4 **extends** JFrame **implements** ActionListener, ChangeListener {

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

**private** JPanel **allLabelsAndParameterSetters**;

**private** JPanel **labelAndParameterSetter1**;

**private** JPanel **labelAndParameterSetter2**;

**private** JPanel **labelAndParameterSetter3**;

**private static** JPanel *labelAndParameterSetter4* = **new** JPanel(**new** FlowLayout());;

**private** JButton **pointSetter**;

**private** JLabel **xLabel**;

**private** JLabel **yLabel**;

**private** JLabel **rLabel**;

**private** JButton **languageChanger**;

**private int languageToSet** = 1;

**private final** String[] **xChangingValues** = {**"+5"**, **"+1"**, **"-1"**, **"-5"** };

**private final int minValueOfR** = 0;

**private final int maxValueOfR** = 150;

**private final int initialValueOfR** = 75;

**public static** ArrayList<JLabel> *pointsLabels* = **new** ArrayList<JLabel>();;

**private** Chart **chart**;

**private static** java.util.List<Point> *points* = **new** ArrayList<Point>();

**private int x** = 0;

**private int y** = 0;

**static int** *R* = 75;

**private** Locale[] **supportedLocales**;

**private** Socket **clientSocket**;

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

**private** DataOutputStream **output**;

**private static** Lab4 *lab4*;

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

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

**public void** run() {

*lab4* = **new** Lab4();

*lab4*.setVisible(**true**);

}

});

}

**public** Lab4(){

connectToServer();

*// Localization*

**supportedLocales** = **new** Locale[]{

**new** Locale(**"en"**, **"US"**),

**new** Locale(**"sr"**, **"RS"**),

};

*// General settings.*

setSize(700,800);

setMinimumSize(**new** Dimension(500,600));

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

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

*// Creating container.*

**allLabelsAndParameterSetters** = **new** JPanel();

*// Adding components to Frame.*

addChart();

add(**allLabelsAndParameterSetters**);

fillFormWithComponents();

showPointsLabels();

pack();

}

*// Filling and setting panels.*

**private void** fillFormWithComponents(){

**allLabelsAndParameterSetters**.setLayout(**new** GridLayout(4,0));

**labelAndParameterSetter1** = **new** JPanel(**new** FlowLayout());

**labelAndParameterSetter2** = **new** JPanel(**new** FlowLayout());

**labelAndParameterSetter3** = **new** JPanel(**new** FlowLayout());

*// Filling panel with X label and ComboBox.*

**labelAndParameterSetter1**.setBorder(BorderFactory.*createLineBorder*(Color.***black***));

**xLabel** = **new** JLabel(**"X = "** + **x** + **" "**);

**labelAndParameterSetter1**.add(**xLabel**);

JComboBox xChangingValuesComboBox = **new** JComboBox(**xChangingValues**);

xChangingValuesComboBox.addActionListener(**this**);

xChangingValuesComboBox.setActionCommand(**"xChangingValuesComboBox"**);

**labelAndParameterSetter1**.add(xChangingValuesComboBox);

*// Filling panel with Y label and RadioButtons.*

**labelAndParameterSetter2**.setBorder(BorderFactory.*createLineBorder*(Color.***black***));

**yLabel** = **new** JLabel(**"Y = "** + **y** + **" "**);

**labelAndParameterSetter2**.add(**yLabel**);

JRadioButton yPlus5 = **new** JRadioButton(**"+5"**);

yPlus5.setSelected(**true**);

JRadioButton yPlus1 = **new** JRadioButton(**"+1"**);

JRadioButton yMinus1 = **new** JRadioButton(**"-1"**);

JRadioButton yMinus5 = **new** JRadioButton(**"-5"**);

ButtonGroup yRadioButtons = **new** ButtonGroup();

yRadioButtons.add(yPlus5);

yRadioButtons.add(yPlus1);

yRadioButtons.add(yMinus1);

yRadioButtons.add(yMinus5);

**labelAndParameterSetter2**.add(yPlus5);

**labelAndParameterSetter2**.add(yPlus1);

**labelAndParameterSetter2**.add(yMinus1);

**labelAndParameterSetter2**.add(yMinus5);

yPlus5.addActionListener(**this**);

yPlus1.addActionListener(**this**);

yMinus1.addActionListener(**this**);

yMinus5.addActionListener(**this**);

yPlus5.setActionCommand(**"yPlus5"**);

yPlus1.setActionCommand(**"yPlus1"**);

yMinus1.setActionCommand(**"yMinus1"**);

yMinus5.setActionCommand(**"yMinus5"**);

*// Filling panel with R label and Slider.*

**labelAndParameterSetter3**.setBorder(BorderFactory.*createLineBorder*(Color.***black***));

**rLabel** = **new** JLabel(**"R = "** + *R* + **" "**);

**labelAndParameterSetter3**.add(**rLabel**);

JSlider setterOfR = **new** JSlider(**minValueOfR**, **maxValueOfR**);

setterOfR.addChangeListener(**this**);

setterOfR.setValue(**initialValueOfR**);

setterOfR.setMajorTickSpacing(25);

setterOfR.setMinorTickSpacing(5);

setterOfR.setPaintTicks(**true**);

setterOfR.setPaintLabels(**true**);

**labelAndParameterSetter3**.add(setterOfR);

*// Adding language changing button, point setting button and points coordinates screen.*

**languageChanger** = **new** JButton(**"sr"**);

**languageChanger**.addActionListener(**new** ActionListener() {

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

**if** (**languageToSet** == 0){

ResourceBundle curLang = ResourceBundle.*getBundle*(**"LocBundle"**, **supportedLocales**[0]);

**pointSetter**.setText((curLang.getString(**"Set"**)));

**languageChanger**.setText(**"sr"**);

**languageToSet** = 1;

}

**else** {

ResourceBundle curLang = ResourceBundle.*getBundle*(**"LocBundle"**, **supportedLocales**[1]);

**pointSetter**.setText((curLang.getString(**"Set"**)));

**languageChanger**.setText(**"eng"**);

**languageToSet** = 0;

}

}

});

**pointSetter** = **new** JButton(**"Set"**);

**pointSetter**.addActionListener(**new** ActionListener() {

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

**chart**.addPoint(**new** Point(**x**,**y**));

}

});

*labelAndParameterSetter4*.add(**languageChanger**);

JLabel delimiter = **new** JLabel(**" "**);

*labelAndParameterSetter4*.add(**pointSetter**);

*labelAndParameterSetter4*.add(delimiter);

JLabel zeroPoint = **new** JLabel(**"{0;0}"**);

*labelAndParameterSetter4*.add(zeroPoint);

*// Adding panels to chart controls panel.*

**allLabelsAndParameterSetters**.add(**labelAndParameterSetter1**);

**allLabelsAndParameterSetters**.add(**labelAndParameterSetter2**);

**allLabelsAndParameterSetters**.add(**labelAndParameterSetter3**);

**allLabelsAndParameterSetters**.add(*labelAndParameterSetter4*);

}

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

String ac = e.getActionCommand();

**if** (ac == **"xChangingValuesComboBox"**) reactToComboBox(e);

**else if** (ac == **"yPlus5"**) reactToRadioButtonYplus5(e);

**else if** (ac == **"yPlus1"**) reactToRadioButtonYplus1(e);

**else if** (ac == **"yMinus1"**) reactToRadioButtonYminus1(e);

**else if** (ac == **"yMinus5"**) reactToRadioButtonYminus5(e);

**xLabel**.setText(**"X = "** + **x** + **" "**);

**yLabel**.setText(**"Y = "** + **y** + **" "**);

}

**void** reactToComboBox(ActionEvent e){

JComboBox cb = (JComboBox) e.getSource();

String sValue = (String) cb.getSelectedItem();

**int** iValue = Integer.*parseInt*(sValue);

**switch** (iValue) {

**case** -5: {

**x** -= 5;

**break**;

}

**case** -1: {

**x**--;

**break**;

}

**case** 1: {

**x**++;

**break**;

}

**case** 5: {

**x** += 5;

**break**;

}

}

}

**void** reactToRadioButtonYplus5(ActionEvent e){

**y** += 5;

}

**void** reactToRadioButtonYplus1(ActionEvent e){

**y**++;

}

**void** reactToRadioButtonYminus1(ActionEvent e){

**y**--;

}

**void** reactToRadioButtonYminus5(ActionEvent e){

**y** -= 5;

}

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

JSlider source = (JSlider)e.getSource();

**if** (!source.getValueIsAdjusting()) {

**int** oldR = *R*;

*R* = (**int**)source.getValue();

**rLabel**.setText(**"R = "** + *R* + **" "**);

**chart**.setR(oldR);

}

}

**static class** Point {

**private int x**;

**private int y**;

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

**this**.**x** = x;

**this**.**y** = y;

}

**private boolean verifiedByServer** = **false**;

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

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

}

**interface** ChartListener {

**void** chartRefreshed();

}

**static class** InZoneChecker {

**public boolean** containsPoint(Point point) {

**if** (point.getX() < 0)

{

**if** ((point.getX() >= -*R*) && (point.getY() >= 0) && (point.getY() <= 0.5 \* (point.getX() + *R*)))

**return true**;

}

**else if** ((point.getX() <= *R*) && (point.getY() >= -0.5 \* *R*) && (point.getY() <= Math.*sqrt*(Math.*pow*(*R*, 2) - Math.*pow*(point.getX(), 2))))

**return true**;

**return false**;

}

**public boolean** containsPoint(Point point, **int** r) {

**if** (point.getX() < 0)

{

**if** ((point.getX() >= -r) && (point.getY() >= 0) && (point.getY() <= 0.5 \* (point.getX() + r)))

**return true**;

}

**else if** ((point.getX() <= r) && (point.getY() >= -0.5 \* r) && (point.getY() <= Math.*sqrt*(Math.*pow*(r, 2) - Math.*pow*(point.getX(), 2))))

**return true**;

**return false**;

}

}

**static class** Chart **extends** JPanel {

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

**public static final** Color ***COLOR\_BEIGE*** = **new** Color(245, 245, 220);

**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\_LIGHTGREEN*** = **new** Color(61,255, 35);

**public static final** Color ***GRAY*** = **new** Color(175, 173, 179);

**private int centerX**;

**private int centerY**;

*//private int R = 5;*

**private** InZoneChecker **inZoneChecker**;

**public** Chart() {

**inZoneChecker** = **new** InZoneChecker();

}

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

InZoneChecker newInZoneChecker = **new** InZoneChecker();

java.util.List<Point> pointsToBeAnimated = **new** ArrayList<Point>();

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

**if**(!newInZoneChecker.containsPoint(point) && **inZoneChecker**.containsPoint(point, oldR)) {

**if** (**finisheAnimation**) startAnimation(pointsToBeAnimated);

**break**;

}

}

**inZoneChecker** = newInZoneChecker;

refresh();

}

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

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

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

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

**listeners**.add(listener);

}

**public** java.util.List<Point> getPoints() {

**return** *points*;

}

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

*points*.add(point);

refresh();

}

**public boolean finisheAnimation** = **true**;

**public void** startAnimation(**final** java.util.List<Point> pointsAnima) {

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

**public void** run() {

**finisheAnimation** = **false**;

**try** {

**int** R0 = ***COLOR\_BROWN***.getRed();

**int** G0 = ***COLOR\_BROWN***.getGreen();

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

**final int** N = 100;

**int** R = R0;

**int** G = G0;

**int** B = B0;

**while** (B < 255) {

**if** (R < 255)

R++;

**if** (G < 255)

G++;

**if** (B < 255)

B++;

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

repaint();

Thread.*sleep*(8);

}

**while** (B > B0) {

**if** (R > R0)

R--;

**if** (G > G0)

G--;

**if** (B > B0)

B--;

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

repaint();

Thread.*sleep*(8);

}

} **catch**(InterruptedException e) {

e.printStackTrace();

**finisheAnimation** = **true**;

}

**finisheAnimation** = **true**;

}

}).start();

}

**public void** refresh() {

repaint();

**for**(ChartListener listener : **listeners**)

listener.chartRefreshed();

}

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

paintTriangle(g, color);

paintCircle(g, color);

paintRectangle(g, color);

}

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

g.setColor(color);

g.fillPolygon(**new** Polygon(**new int**[] {**centerX** , **centerX**, **centerX** - *R*, **centerX**}, **new int**[] {**centerY**, **centerY** - *R*/2, **centerY**, **centerY**}, 4));

}

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

g.setColor(color);

g.fillArc(**centerX** - *R*, **centerY** - *R*,*R*<<1,*R*<<1,90,-90);

}

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

g.setColor(color);

g.fillRect(**centerX**, **centerY**, *R*, *R*/2);

}

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

**super**.paintComponent(graphics);

graphics.setColor(***COLOR\_BEIGE***);

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

**centerX** = getWidth() / 2;

**centerY** = getHeight() / 2;

paintArea(graphics, *areaColor*);

graphics.setColor(***COLOR\_BLACK***);

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

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

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

**if** (point.**verifiedByServer**){

graphics.setColor(**inZoneChecker**.containsPoint(point) ? ***COLOR\_LIGHTGREEN*** : ***COLOR\_BLACK***);

}

**else**{

**switch** (*lab4*.sendPoint(point)){

**case** 0:

graphics.setColor(***COLOR\_BLACK***);

point.**verifiedByServer** = **true**;

**break**;

**case** 1:

graphics.setColor(***COLOR\_LIGHTGREEN***);

point.**verifiedByServer** = **true**;

**break**;

**default**:

graphics.setColor(***GRAY***);

**break**;

}

}

graphics.fillOval(**centerX** + point.getX()-(***POINT\_RADIUS*** / 2), **centerY** - point.getY()-(***POINT\_RADIUS*** / 2),***POINT\_RADIUS***,***POINT\_RADIUS***);

}

repaint();

}

}

**public void** showPointsLabels(){

**allLabelsAndParameterSetters**.add(*labelAndParameterSetter4*);

**chart**.subscribe(**new** ChartListener() {

**public void** chartRefreshed() {

**int** lastPointIndex = *points*.size() - 1;

*pointsLabels*.add(**new** JLabel(**"{"** + *points*.get(lastPointIndex).getX() + **";"** +

*points*.get(lastPointIndex).getY() + **"}\n"**));

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

*labelAndParameterSetter4*.add(*pointsLabels*.get(i));

}

**x** = *points*.get(lastPointIndex).getX();

**y** = *points*.get(lastPointIndex).getY();

**xLabel**.setText(**"X = "** + **x** + **" "**);

**yLabel**.setText(**"Y = "** + **y** + **" "**);

}

});

}

**public int** sendPoint(Point point){

**int** result = -1;

**try**{

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

**output**.writeFloat(point.getX());

**output**.writeFloat(point.getY());

**output**.writeFloat(*R*);

result = **input**.readInt();

} **catch** (Exception e){

result = -1;

}

**return** result;

}

**public void** connectToServer() {

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

**public void** run() {

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

**try** {

**clientSocket** = **new** Socket();

**clientSocket**.connect(**new** InetSocketAddress(**"localhost"**, 8652), 0);

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

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

} **catch** (IOException e) {

}

**try** {

Thread.*sleep*(5000);

} **catch** (Exception e) {

}

}

}

}).start();

}

**public static boolean** hostAvailabilityCheck() {

**try** (Socket s = **new** Socket(**"localhost"**, 8652)) {

**return true**;

} **catch** (IOException ex) {

}

**return false**;

}

**public void** addChart() {

**chart** = **new** Chart();

**chart**.addPoint(**guiPoint**);

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

@Override

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

Point newPoint = **new** Point(e.getX() - **chart**.getCenterX(), **chart**.getCenterY() - e.getY());

**chart**.addPoint(newPoint);

}

@Override

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

}

});

add(**chart**);

}

}

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**public class** Server **implements** Runnable {

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

**volatile static int** *i* = 0;

**public** Server(Socket socket) {

**this**.**socket** = socket;

}

**public void** run() {

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

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

{

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

**float** x = input.readFloat();

**float** y = input.readFloat();

**float** r = input.readFloat();

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

}

} **catch**(Exception e) {

}

}

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

**int** portNumber = 8652;

**boolean** listening = **true**;

**try** (ServerSocket serverSocket = **new** ServerSocket(portNumber)) {

**while** (listening) {

**new** Thread(**new** Server(serverSocket.accept())).start();

}

} **catch** (IOException e) {

System.***err***.println(**"Could not listen on port "** + portNumber);

System.*exit*(-1);

}

}

}

**class** InZoneChecker {

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

**if** (x < 0)

{

**if** ((x >= -r) && (y >= 0) && (y <= 0.5 \* (x + r)))

**return true**;

}

**else if** ((x <= r) && (y >= -0.5 \* r) && (y <= Math.*sqrt*(Math.*pow*(r, 2) - Math.*pow*(x, 2))))

**return true**;

**return false**;

}

}

**import** java.util.ListResourceBundle;

**public class** LocBundle\_en\_US **extends** ListResourceBundle {

**public** Object[][] getContents() {

**return contents**;

}

**private** Object[][] **contents** = {

{**"Set"**, **new** String(**"Set"**) },

};

}

**import** java.util.ListResourceBundle;

**public class** LocBundle\_sr\_RS **extends** ListResourceBundle{

**public** Object[][] getContents() {

**return contents**;

}

**private** Object[][] **contents** = {

{**"Set"**, **new** String(**"Комплет"**) },

};

}

**Вывод:**

В результате выполнения лабораторной работы мною были изучены способы локализации, средства, необходимые для создания клиент-серверного приложения, изучены протоколы передачи данных TCP и UDP.

Полученные знания можно использовать для создания графических клиент-серверных приложений языке Java.