Кафедра Вычислительной техники



Лабораторная работа №6

по дисциплине «Разработка клиент-серверных приложений»

на тему: «Сетевое соединение на сокетах»

Группа: АВТ-918

Студент: Ванин К.Е

Преподаватель: Токарев В.Г.

Содержание

[Цель работы 3](#_Toc55463600)

[Проектирование программы 3](#_Toc55463602)

[«Составные части» программы 4](#_Toc55463604)

Описание класса Protocol…………………………………………………………8

[Пример работы программы 10](#_Toc55463606)

[Вывод 1](#_Toc55463607)1

# Цель работы

Задание в л.р.5 реализовать с использованием класса дейтаграмм протокола UDP без установления соединения (класс DatagramSocket). Флажком выделяется «первое/второе» приложение, отличающиеся только номером порта, на котором они прослушивают прием. Например, первое принимает данные на порте 7001, а передает на 7002, второе – наоборот. Для конвертации XML-строки, содержащей передаваемый объект, в массив байтов для передачи в дейтаграммный сокет использовать классы ByteArrayOutputStream/ByteArrayOutputStream в качестве класса-источника текстового потока.

# Проектирование программы

* **public class MainJFrame extends javax.swing.JFrame – Основной класс запуска программы**
* **public class Server extends Thread – Создание сервера**
* **public class Log – Обработчик обишок**

# «Составные части» программы

**Класс MainJFrame**

package java\_laba\_2;  
  
public class MainJFrame extends javax.swing.JFrame {  
  
 Screen s;  
 Client c;  
  
 public MainJFrame() {  
 initComponents();  
  
 s = new Screen();  
 panel.add(s);  
 s.setSize(panel.getSize());  
 panel.addMouseListener(s);  
 panel.addKeyListener(s);  
 addWindowListener(new HelpListener());  
 s.setVisible(true);  
 pack();  
  
 }  
  
 @SuppressWarnings("unchecked")  
 private void initComponents() {  
  
 panel = new javax.swing.JPanel();  
 hostLabel = new javax.swing.JLabel();  
 hostEdit = new javax.swing.JTextField();  
 portLabel = new javax.swing.JLabel();  
 portInputEdit = new javax.swing.JTextField();  
 startClientButton = new javax.swing.JButton();  
 startServerButton = new javax.swing.JButton();  
 statusLabel = new javax.swing.JLabel();  
 portOutputEdit = new javax.swing.JTextField();  
 portOutput = new javax.swing.JLabel();  
 swapPortsButton = new javax.swing.JButton();  
  
 setDefaultCloseOperation(javax.swing.WindowConstants.*EXIT\_ON\_CLOSE*);  
 setBackground(new java.awt.Color(0, 0, 0));  
 setLocationByPlatform(true);  
 setResizable(false);  
 setSize(new java.awt.Dimension(666, 367));  
  
 panel.setBackground(new java.awt.Color(255, 255, 255));  
 panel.setBorder(javax.swing.BorderFactory.*createEmptyBorder*(1, 1, 1, 1));  
  
 javax.swing.GroupLayout panelLayout = new javax.swing.GroupLayout(panel);  
 panel.setLayout(panelLayout);  
 panelLayout.setHorizontalGroup(  
 panelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addGap(0, 458, Short.*MAX\_VALUE*)  
 );  
 panelLayout.setVerticalGroup(  
 panelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addGap(0, 0, Short.*MAX\_VALUE*)  
 );  
  
 hostLabel.setText("host");  
  
 hostEdit.setText("localhost");  
  
 portLabel.setText("port (input)");  
  
 portInputEdit.setText("8000");  
 portInputEdit.setToolTipText("");  
  
 startClientButton.setText("Start client");  
 startClientButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 startClientButtonActionPerformed(evt);  
 }  
 });  
  
 startServerButton.setText("Start server");  
 startServerButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 startServerButtonActionPerformed(evt);  
 }  
 });  
  
 statusLabel.setFont(new java.awt.Font("Source Sans Pro", 1, 18)); // NOI18N  
 statusLabel.setText("OFFLINE");  
  
 portOutputEdit.setText("8001");  
 portOutputEdit.setToolTipText("");  
  
 portOutput.setText("port (output)");  
  
 swapPortsButton.setText("Swap ports");  
 swapPortsButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 swapPortsButtonActionPerformed(evt);  
 }  
 });  
  
 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  
 getContentPane().setLayout(layout);  
 layout.setHorizontalGroup(  
 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addGroup(layout.createSequentialGroup()  
 .addComponent(panel, javax.swing.GroupLayout.*PREFERRED\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*PREFERRED\_SIZE*)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addGroup(layout.createSequentialGroup()  
 .addComponent(portOutput)  
 .addGap(18, 18, 18)  
 .addComponent(swapPortsButton)  
 .addContainerGap(javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*))  
 .addGroup(layout.createSequentialGroup()  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addComponent(startClientButton, javax.swing.GroupLayout.*PREFERRED\_SIZE*, 189, javax.swing.GroupLayout.*PREFERRED\_SIZE*)  
 .addComponent(statusLabel)  
 .addComponent(startServerButton, javax.swing.GroupLayout.*PREFERRED\_SIZE*, 189, javax.swing.GroupLayout.*PREFERRED\_SIZE*)  
 .addGroup(layout.createSequentialGroup()  
 .addComponent(hostLabel)  
 .addGap(18, 18, 18)  
 .addComponent(hostEdit, javax.swing.GroupLayout.*PREFERRED\_SIZE*, 150, javax.swing.GroupLayout.*PREFERRED\_SIZE*))  
 .addComponent(portLabel)  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*TRAILING*, false)  
 .addComponent(portInputEdit, javax.swing.GroupLayout.Alignment.*LEADING*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, 189, Short.*MAX\_VALUE*)  
 .addComponent(portOutputEdit, javax.swing.GroupLayout.Alignment.*LEADING*)))  
 .addGap(0, 8, Short.*MAX\_VALUE*))))  
 );  
 layout.setVerticalGroup(  
 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addComponent(panel, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*)  
 .addGroup(layout.createSequentialGroup()  
 .addContainerGap()  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*BASELINE*)  
 .addComponent(hostLabel)  
 .addComponent(hostEdit, javax.swing.GroupLayout.*PREFERRED\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*PREFERRED\_SIZE*))  
 .addGap(17, 17, 17)  
 .addComponent(portLabel)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addComponent(portInputEdit, javax.swing.GroupLayout.*PREFERRED\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*PREFERRED\_SIZE*)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*BASELINE*)  
 .addComponent(portOutput)  
 .addComponent(swapPortsButton))  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addComponent(portOutputEdit, javax.swing.GroupLayout.*PREFERRED\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*PREFERRED\_SIZE*)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*, 113, Short.*MAX\_VALUE*)  
 .addComponent(startClientButton)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addComponent(startServerButton)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
 .addComponent(statusLabel)  
 .addGap(12, 12, 12))  
 );  
  
 pack();  
 }// </editor-fold>//GEN-END:initComponents  
  
 private void startServerButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_startServerButtonActionPerformed  
 String host = hostEdit.getText();  
 int port\_in = Integer.parseInt(portInputEdit.getText());  
 int port\_out = Integer.parseInt(portOutputEdit.getText());  
 Server server = new Server(s, host, port\_out, port\_in);  
 server.start();  
  
 statusLabel.setText("SERVER RUNNING");  
 hostEdit.setEnabled(false);  
 portInputEdit.setEnabled(false);  
 portOutputEdit.setEnabled(false);  
  
 new Thread(new Runnable() {  
 public void run() {  
 while (server.isAlive()) {  
 }  
 statusLabel.setText("OFFLINE");  
 portInputEdit.setEnabled(true);  
 portOutputEdit.setEnabled(true);  
 }  
 }).start();  
 }//GEN-LAST:event\_startServerButtonActionPerformed  
  
 private void startClientButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_startClientButtonActionPerformed  
 String host = hostEdit.getText();  
 int port\_in = Integer.parseInt(portInputEdit.getText());  
 int port\_out = Integer.parseInt(portOutputEdit.getText());  
 c = new Client(s, host, port\_out, port\_in);  
 c.start();  
  
 statusLabel.setText("CLIENT RUNNING");  
 hostEdit.setEnabled(false);  
 portInputEdit.setEnabled(false);  
 portOutputEdit.setEnabled(false);  
 new Thread(new Runnable() {  
 public void run() {  
 while (c.isAlive()) {  
 }  
 statusLabel.setText("OFFLINE");  
 portInputEdit.setEnabled(true);  
 portOutputEdit.setEnabled(true);  
 }  
 }).start();  
 }//GEN-LAST:event\_startClientButtonActionPerformed  
  
 private void swapPortsButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_swapPortsButtonActionPerformed  
 String temp = portInputEdit.getText();  
 portInputEdit.setText(portOutputEdit.getText());  
 portOutputEdit.setText(temp);  
 }//GEN-LAST:event\_swapPortsButtonActionPerformed  
  
 */\*\*  
 \** ***@param*** *args the command line arguments  
 \*/* public static void main(String args[]) {  
 /\* Set the Nimbus look and feel \*/  
 //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  
 /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  
 \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html  
 \*/  
 try {  
 for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.*getInstalledLookAndFeels*()) {  
 if ("Windows".equals(info.getName())) {  
 javax.swing.UIManager.setLookAndFeel(info.getClassName());  
 break;  
 }  
 }  
 } catch (ClassNotFoundException ex) {  
 java.util.logging.Logger.getLogger(MainJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
 } catch (InstantiationException ex) {  
 java.util.logging.Logger.getLogger(MainJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
 } catch (IllegalAccessException ex) {  
 java.util.logging.Logger.getLogger(MainJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
 } catch (javax.swing.UnsupportedLookAndFeelException ex) {  
 java.util.logging.Logger.getLogger(MainJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
 }  
 //</editor-fold>  
  
 /\* Create and display the form \*/  
 java.awt.EventQueue.invokeLater(new Runnable() {  
 public void run() {  
 MainJFrame f = new MainJFrame();  
 f.setVisible(true);  
 }  
 });  
 }  
  
 // Variables declaration - do not modify//GEN-BEGIN:variables  
 private javax.swing.JTextField hostEdit;  
 private javax.swing.JLabel hostLabel;  
 private javax.swing.JPanel panel;  
 private javax.swing.JTextField portInputEdit;  
 private javax.swing.JLabel portLabel;  
 private javax.swing.JLabel portOutput;  
 private javax.swing.JTextField portOutputEdit;  
 private javax.swing.JButton startClientButton;  
 private javax.swing.JButton startServerButton;  
 private javax.swing.JLabel statusLabel;  
 private javax.swing.JButton swapPortsButton;  
 // End of variables declaration//GEN-END:variables  
  
}

**Класс Server**

package java\_laba\_2;  
  
import java.io.\*;  
import java.net.\*;  
  
public class Server extends Thread {  
  
 DatagramSocket s;  
 public Screen surface;  
 private boolean stoped = false;  
 private int portIn, portOut;  
 private InetAddress addr;  
  
 private void Stop() {  
 stoped = true;  
 }  
  
 Server(Screen sf, String ip, int port\_out, int port\_in) {  
 try {  
 surface = sf;  
 addr = InetAddress.*getByName*(ip);  
 portIn = port\_in;  
 portOut = port\_out;  
 s = new DatagramSocket();  
 Log.*ln*("### SERVER STARTED ");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 setDaemon(true);  
 setPriority(*NORM\_PRIORITY*);  
 }  
  
 @Override  
 public void run() {  
 while (stoped != true) {  
 try {  
 Thread.sleep(100);  
 DatagramPacket pac;  
  
 ByteArrayOutputStream bos = new ByteArrayOutputStream();  
 ObjectOutput out = null;  
 try {  
 out = new ObjectOutputStream(bos);  
 out.writeObject(surface.objects);  
 byte[] blob = bos.toByteArray();  
 pac = new DatagramPacket(blob, blob.length, addr, portOut);  
 s.send(pac);  
 } finally {  
 try {  
 if (out != null) {  
 out.close();  
 }  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 try {  
 bos.close();  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 break;  
 }  
 }  
  
 Log.ln("SERVER: stop running");  
 }  
}

**Класс Log**

package java\_laba\_2;  
  
  
public class Log {  
 static void ln(String s)  
 {  
 System.*out*.println(s);  
 }  
}

**Класс Server**

package java\_laba\_2;  
  
import java.io.\*;  
import java.net.\*;  
  
public class Server extends Thread {  
  
 DatagramSocket s;  
 public Screen surface;  
 private boolean stoped = false;  
 private int portIn, portOut;  
 private InetAddress addr;  
  
 private void Stop() {  
 stoped = true;  
 }  
  
 Server(Screen sf, String ip, int port\_out, int port\_in) {  
 try {  
 surface = sf;  
 addr = InetAddress.*getByName*(ip);  
 portIn = port\_in;  
 portOut = port\_out;  
 s = new DatagramSocket();  
 Log.*ln*("### SERVER STARTED ");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 setDaemon(true);  
 setPriority(*NORM\_PRIORITY*);  
 }  
  
 @Override  
 public void run() {  
 while (stoped != true) {  
 try {  
 Thread.*sleep*(100);  
 DatagramPacket pac;  
  
 ByteArrayOutputStream bos = new ByteArrayOutputStream();  
 ObjectOutput out = null;  
 try {  
 out = new ObjectOutputStream(bos);  
 out.writeObject(surface.objects);  
 byte[] blob = bos.toByteArray();  
 pac = new DatagramPacket(blob, blob.length, addr, portOut);  
 s.send(pac);  
 } finally {  
 try {  
 if (out != null) {  
 out.close();  
 }  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 try {  
 bos.close();  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 break;  
 }  
 }  
  
 Log.*ln*("SERVER: stop running");  
 }  
}

**Спецификация протокола взаимодействия**

Описание байта на клиентской стороне:

stoped - закрытие сессии (по умолчанию false).

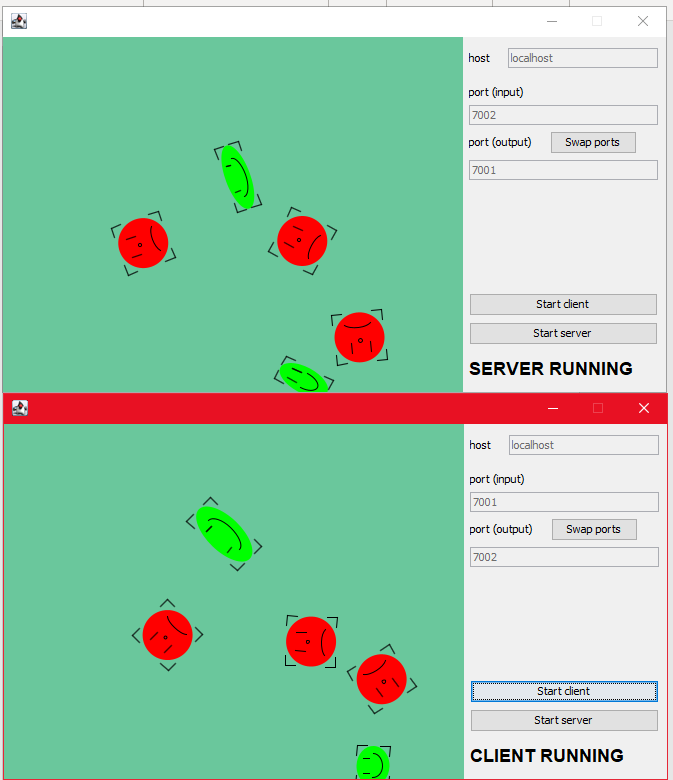
После вызова функции Stop() значение stoped становится true – передается сообщение “SERVER: stop running” – соединение закрыто;

portIn - вход порт.

portOut - выходной порт.

InetAddress addr – адресс.

**Пример работы программы**



# Вывод

В ходе выполнения лабораторной работы была сделана реализация с использованием класса дейтаграмм протокола UDP без установления соединения (класс DatagramSocket).

Флажком выделяется «первое/второе» приложение, отличающиеся только номером порта, на котором они прослушивают прием.

Первое принимает данные на порте 7001, а передает на 7002, второе – наоборот.

Для конвертации XML-строки, содержащей передаваемый объект, в массив байтов для передачи в дейтаграммный сокет использовать классы ByteArrayOutputStream/ByteArrayOutputStream в качестве класса-источника текстового потока.