Кафедра Вычислительной техники



Лабораторная работа №5

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

Разработать приложение с клиентской и серверной частью. Серверная часть реализуется в потоке и запускается/останавливается при помощи флажка. Серверная компонента ждет запроса на соединение на фиксированном порте. Клиентская компонента в другой программе запрашивает соединение с серверной. Протокол является симметричным (двунаправленным), т.е. каждое приложение может посылать один тот же набор команд: закрытие соединения, очистка вектора объектов, передача сериализованного объекта, запрос размера вектора объектов, запрос объекта с заданным номером из вектора. Принимаемые объекты отображаются в клиенте и запоминаются в векторе.

# Проектирование программы

* **public class MainJFrame extends javax.swing.JFrame – Основной класс запуска программы**
* **public class Server extends Thread – Создание сервера**
* **public class Protocol – Соединение между сервером и клиентом**
* **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();  
  
 clearButton.setEnabled(false);  
 closeConnectionButton.setEnabled(false);  
 getAllObjectsButton.setEnabled(false);  
 getObjectButton.setEnabled(false);  
 getObjectCountButton.setEnabled(false);  
 objectCountEdit.setEnabled(false);  
 }  
  
 @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();  
 portEdit = new javax.swing.JTextField();  
 startConnectionButton = new javax.swing.JButton();  
 closeConnectionButton = new javax.swing.JButton();  
 getObjectButton = new javax.swing.JButton();  
 objectCountEdit = new javax.swing.JTextField();  
 getObjectCountButton = new javax.swing.JButton();  
 launchServerButton = new javax.swing.JButton();  
 statusLabel = new javax.swing.JLabel();  
 getAllObjectsButton = new javax.swing.JButton();  
 clearButton = 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");  
  
 portEdit.setText("8888");  
 portEdit.setToolTipText("");  
  
 startConnectionButton.setText("Start connection");  
 startConnectionButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 startConnectionButtonActionPerformed(evt);  
 }  
 });  
  
 closeConnectionButton.setText("Close connection");  
 closeConnectionButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 closeConnectionButtonActionPerformed(evt);  
 }  
 });  
  
 getObjectButton.setText("Get object");  
 getObjectButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 getObjectButtonActionPerformed(evt);  
 }  
 });  
  
 objectCountEdit.setText("0");  
  
 getObjectCountButton.setText("Get objects count");  
 getObjectCountButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 getObjectCountButtonActionPerformed(evt);  
 }  
 });  
  
 launchServerButton.setText("Launch server");  
 launchServerButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 launchServerButtonActionPerformed(evt);  
 }  
 });  
  
 statusLabel.setFont(new java.awt.Font("Source Sans Pro", 1, 18)); // NOI18N  
 statusLabel.setText("CLIENT");  
  
 getAllObjectsButton.setText("Get all objects");  
 getAllObjectsButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 getAllObjectsButtonActionPerformed(evt);  
 }  
 });  
  
 clearButton.setActionCommand("cvc");  
 clearButton.setLabel("Clear");  
 clearButton.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 clearButtonActionPerformed(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*, false)  
 .addComponent(getAllObjectsButton, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*)  
 .addGroup(layout.createSequentialGroup()  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addComponent(hostLabel)  
 .addComponent(portLabel))  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*UNRELATED*)  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*, false)  
 .addComponent(hostEdit, javax.swing.GroupLayout.*DEFAULT\_SIZE*, 150, Short.*MAX\_VALUE*)  
 .addComponent(portEdit)))  
 .addComponent(closeConnectionButton, javax.swing.GroupLayout.*DEFAULT\_SIZE*, 189, Short.*MAX\_VALUE*)  
 .addComponent(startConnectionButton, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*)  
 .addGroup(layout.createSequentialGroup()  
 .addComponent(getObjectButton)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addComponent(objectCountEdit))  
 .addComponent(getObjectCountButton, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*)  
 .addComponent(statusLabel)  
 .addComponent(launchServerButton, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*)  
 .addComponent(clearButton, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*))  
 .addGap(0, 11, 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)  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addComponent(portLabel)  
 .addComponent(portEdit, javax.swing.GroupLayout.*PREFERRED\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*PREFERRED\_SIZE*))  
 .addGap(18, 18, 18)  
 .addComponent(startConnectionButton)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addComponent(closeConnectionButton)  
 .addGap(18, 18, 18)  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*BASELINE*)  
 .addComponent(getObjectButton)  
 .addComponent(objectCountEdit, javax.swing.GroupLayout.*PREFERRED\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*PREFERRED\_SIZE*))  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addComponent(getObjectCountButton)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addComponent(getAllObjectsButton)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*)  
 .addComponent(clearButton)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*, 9, Short.*MAX\_VALUE*)  
 .addComponent(statusLabel)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*UNRELATED*)  
 .addComponent(launchServerButton)  
 .addGap(12, 12, 12))  
 );  
  
 clearButton.getAccessibleContext().setAccessibleName("clearButton");  
  
 pack();  
 }  
  
 private void launchServerButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_launchServerButtonActionPerformed  
 String host = hostEdit.getText();  
 int port = Integer.*parseInt*(portEdit.getText());  
  
 Server server = new Server(s,host,port);  
 server.start();  
  
 statusLabel.setText("SERVER RUNNING");  
 hostEdit.setEnabled(false);  
 portEdit.setEnabled(false);  
  
 new Thread(new Runnable() {  
 public void run() {  
 while(server.isAlive()) {}  
 statusLabel.setText("CLIENT");  
 hostEdit.setEnabled(true);  
 portEdit.setEnabled(true);  
 }  
 }).start();  
 }  
  
 private void startConnectionButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 String host = hostEdit.getText();  
 int port = Integer.*parseInt*(portEdit.getText());  
 c = new Client(s,host,port);  
 c.start();  
  
 clearButton.setEnabled(true);  
 startConnectionButton.setEnabled(false);  
 closeConnectionButton.setEnabled(true);  
 getAllObjectsButton.setEnabled(true);  
 getObjectButton.setEnabled(true);  
 getObjectCountButton.setEnabled(true);  
 objectCountEdit.setEnabled(true);  
  
 hostEdit.setEnabled(false);  
 portEdit.setEnabled(false);  
 }  
  
 private void getAllObjectsButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 try {  
 Protocol.*QueryObjects*(c.socket);  
 } catch (Exception e) { e.printStackTrace(); }  
 }  
  
 private void closeConnectionButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 try {  
 Protocol.*CloseConnection*(c.socket);  
 } catch (Exception e) { e.printStackTrace(); }  
  
 clearButton.setEnabled(false);  
 startConnectionButton.setEnabled(true);  
 closeConnectionButton.setEnabled(false);  
 getAllObjectsButton.setEnabled(false);  
 getObjectButton.setEnabled(false);  
 getObjectCountButton.setEnabled(false);  
 objectCountEdit.setEnabled(false);  
  
 hostEdit.setEnabled(true);  
 portEdit.setEnabled(true);  
 }  
  
 private void getObjectButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 int count = Integer.*parseInt*(objectCountEdit.getText());  
 try {  
 Protocol.*QueryObject*(c.socket, count);  
 } catch (Exception e) { e.printStackTrace(); }  
  
 }  
  
 private void getObjectCountButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 try {  
 Protocol.*QueryListSize*(c.socket);  
 } catch (Exception e) { e.printStackTrace(); }  
  
 }  
  
 private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 try {  
 Protocol.*ClearList*(c.socket);  
 } catch (Exception e) { e.printStackTrace(); }  
 }  
  
 public static void main(String args[]) {  
 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);  
 }  
  
  
  
 java.awt.EventQueue.*invokeLater*(new Runnable() {  
 public void run() {  
 MainJFrame f = new MainJFrame();  
 f.setVisible(true);  
 }  
 });  
 }  
  
 private javax.swing.JButton clearButton;  
 private javax.swing.JButton closeConnectionButton;  
 private javax.swing.JButton getAllObjectsButton;  
 private javax.swing.JButton getObjectButton;  
 private javax.swing.JButton getObjectCountButton;  
 private javax.swing.JTextField hostEdit;  
 private javax.swing.JLabel hostLabel;  
 private javax.swing.JButton launchServerButton;  
 private javax.swing.JTextField objectCountEdit;  
 private javax.swing.JPanel panel;  
 private javax.swing.JTextField portEdit;  
 private javax.swing.JLabel portLabel;  
 private javax.swing.JButton startConnectionButton;  
 private javax.swing.JLabel statusLabel;  
  
}

**Класс Server**

package java\_laba\_2;  
  
import java.net.InetAddress;  
import java.net.ServerSocket;  
import java.net.Socket;  
  
public class Server extends Thread{  
 private ServerSocket ssocket;  
 Socket s;  
 public Screen surface;  
 Server(Screen sf, String ip, int port)  
 {  
 surface = sf;  
  
 try {  
 ssocket = new ServerSocket(port,0,InetAddress.*getByName*(ip));  
 Log.*ln*("### SERVER STARTED: " + ssocket.getInetAddress().toString() + ':' + ssocket.getLocalPort());  
 } catch (Exception e) { e.printStackTrace(); }  
 setDaemon(true);  
 setPriority(*NORM\_PRIORITY*);  
 try {  
 s = ssocket.accept();  
 } catch (Exception e) { e.printStackTrace();}  
 }  
 @Override  
 public void run()  
 {  
 while (true)  
 try {  
 if (!s.isClosed())  
 {  
 Protocol.QueryData q = Protocol.*ListenClient*(s);  
  
 switch (q.code)  
 {  
 case (Protocol.*CLEAR\_LIST*):  
 {  
 surface.objects.clear();  
 Log.*ln*("SERVER: list cleared");  
 } break;  
 case (Protocol.*GET\_OBJECT*):  
 {  
 int i = (Integer)q.data;  
 Protocol.*SendObject*(s, surface.objects.get(i),i);  
 Log.*ln*("SERVER: object " + i + " sended");  
 } break;  
 case (Protocol.*GET\_ALL\_OBJECTS*):  
 {  
 Protocol.*SendAllObjects*(s, surface.objects);  
 Log.*ln*("SERVER: all objects sended");  
 } break;  
 case(Protocol.*GET\_LIST\_SIZE*):  
 {  
 Protocol.*SendListSize*(s, surface.objects.size());  
 Log.*ln*("SERVER: objects count sended");  
 } break;  
 case(Protocol.*CLOSE\_CONNECTION*) :  
 {  
 Protocol.*CloseConnection*(s);  
 Log.*ln*("SERVER: connection closed");  
 } break;  
 default: Log.*ln*("SERVER: unknown query code"); break;  
 }  
 } else {ssocket.close(); break; }  
 }  
 catch (Exception e) { }  
  
 Log.*ln*("SERVER: stop running");  
 }  
}

**Класс Protocol**

package java\_laba\_2;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.io.OutputStream;  
import java.net.Socket;  
import java.util.ArrayList;  
  
public class Protocol {  
  
 final static public int *UNKNOWN* = -1;  
  
 // CLIENT  
 final static public int *CLOSE\_CONNECTION* = 0;  
 final static public int *CLEAR\_LIST* = 1;  
 final static public int *GET\_LIST\_SIZE* = 2;  
 final static public int *GET\_OBJECT* = 3;  
 final static public int *GET\_ALL\_OBJECTS* =4;  
  
 // SERVER  
 final static public int *SEND\_LIST\_SIZE* = 5;  
 final static public int *SEND\_OBJECT* = 6;  
 final static public int *SEND\_ALL\_OBJECTS* = 7;  
  
 static public class QueryData  
 {  
 public int code;  
 public Object data;  
 }  
  
 static void CloseConnection(Socket s) throws IOException  
 {  
 s.getOutputStream().write(*CLOSE\_CONNECTION*);  
 s.getOutputStream().flush();  
 s.close();  
 Log.*ln*("CLIENT: connection closed");  
 }  
 static void ClearList(Socket s) throws IOException  
 {  
 s.getOutputStream().write(*CLEAR\_LIST*);  
 s.getOutputStream().flush();  
 Log.*ln*("CLIENT: clear list requested");  
 }  
  
 static void QueryListSize(Socket s) throws IOException  
 {  
 s.getOutputStream().write(*GET\_LIST\_SIZE*);  
 s.getOutputStream().flush();  
 Log.*ln*("CLIENT: objects count requested");  
  
 }  
 static void QueryObject(Socket s, int index) throws IOException  
 {  
 s.getOutputStream().write(*GET\_OBJECT*);  
 s.getOutputStream().write(index);  
 s.getOutputStream().flush();  
 Log.*ln*("CLIENT: object " + index + " requested");  
  
 }  
 static void QueryObjects(Socket s) throws IOException  
 {  
 s.getOutputStream().write(*GET\_ALL\_OBJECTS*);  
 s.getOutputStream().flush();  
 Log.*ln*("CLIENT: all objects requested");  
  
 }  
  
 static void SendListSize(Socket s, int size) throws IOException  
 {  
 s.getOutputStream().write(*SEND\_LIST\_SIZE*);  
 s.getOutputStream().write(size);  
 s.getOutputStream().flush();  
 }  
  
  
 static void SendObject(Socket s, Smile o, int index) throws IOException  
 {  
 OutputStream os = s.getOutputStream();  
 os.write(*SEND\_OBJECT*);  
 os.write(index);  
 ObjectOutputStream oos = new ObjectOutputStream(os);  
 oos.writeObject(o);  
 os.flush();  
 }  
  
 static void SendAllObjects(Socket s, ArrayList<Smile> list) throws IOException  
 {  
 OutputStream os = s.getOutputStream();  
 os.write(*SEND\_ALL\_OBJECTS*);  
 ObjectOutputStream oos = new ObjectOutputStream(os);  
 oos.writeObject(list);  
 os.flush();  
 }  
  
 static QueryData ListenServer(Socket s) throws IOException, ClassNotFoundException  
 {  
 InputStream is = s.getInputStream();  
  
 Log.*ln*("CLIENT: InputStream size = " + is.available());  
 int code = is.read();  
 QueryData qd = new QueryData();  
 qd.code = code;  
 switch (code)  
 {  
 case(*SEND\_LIST\_SIZE*):  
 qd.data = is.read();  
 break;  
 case(*SEND\_OBJECT*):  
 {  
 QueryData obj = new QueryData();  
 // object index  
 obj.code = is.read();  
  
 ObjectInputStream ois = new ObjectInputStream(is);  
 obj.data = (Smile)ois.readObject();  
 qd.data = obj;  
 }  
 break;  
 case(*SEND\_ALL\_OBJECTS*):  
 {  
 ObjectInputStream ois = new ObjectInputStream(is);  
 qd.data = (ArrayList<Smile>)ois.readObject();  
 }  
 break;  
 default: { qd.code = *UNKNOWN*; qd.data = null; } break;  
 }  
 return qd;  
 }  
  
 static QueryData ListenClient(Socket s) throws IOException, ClassNotFoundException  
 {  
 InputStream is = s.getInputStream();  
 int code = is.read();  
 QueryData q = new QueryData();  
  
 q.code = code;  
  
 switch(code){  
 case(*GET\_LIST\_SIZE*): q.data = is.read(); break;  
 case(*GET\_OBJECT*) : q.data = is.read(); break;  
 case(*CLOSE\_CONNECTION*):  
 case(*CLEAR\_LIST*):  
 case(*GET\_ALL\_OBJECTS*): break;  
 default: { q.code = *UNKNOWN*; q.data = null; }  
 }  
 return q;  
 }  
}

**Класс Log**

package java\_laba\_2;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.io.OutputStream;  
import java.net.Socket;  
import java.util.ArrayList;  
  
public class Protocol {  
  
 final static public int *UNKNOWN* = -1;  
  
 // CLIENT  
 final static public int *CLOSE\_CONNECTION* = 0;  
 final static public int *CLEAR\_LIST* = 1;  
 final static public int *GET\_LIST\_SIZE* = 2;  
 final static public int *GET\_OBJECT* = 3;  
 final static public int *GET\_ALL\_OBJECTS* =4;  
  
 // SERVER  
 final static public int *SEND\_LIST\_SIZE* = 5;  
 final static public int *SEND\_OBJECT* = 6;  
 final static public int *SEND\_ALL\_OBJECTS* = 7;  
  
 static public class QueryData  
 {  
 public int code;  
 public Object data;  
 }  
  
 static void CloseConnection(Socket s) throws IOException  
 {  
 s.getOutputStream().write(*CLOSE\_CONNECTION*);  
 s.getOutputStream().flush();  
 s.close();  
 Log.*ln*("CLIENT: connection closed");  
 }  
 static void ClearList(Socket s) throws IOException  
 {  
 s.getOutputStream().write(*CLEAR\_LIST*);  
 s.getOutputStream().flush();  
 Log.*ln*("CLIENT: clear list requested");  
 }  
  
 static void QueryListSize(Socket s) throws IOException  
 {  
 s.getOutputStream().write(*GET\_LIST\_SIZE*);  
 s.getOutputStream().flush();  
 Log.*ln*("CLIENT: objects count requested");  
  
 }  
 static void QueryObject(Socket s, int index) throws IOException  
 {  
 s.getOutputStream().write(*GET\_OBJECT*);  
 s.getOutputStream().write(index);  
 s.getOutputStream().flush();  
 Log.*ln*("CLIENT: object " + index + " requested");  
  
 }  
 static void QueryObjects(Socket s) throws IOException  
 {  
 s.getOutputStream().write(*GET\_ALL\_OBJECTS*);  
 s.getOutputStream().flush();  
 Log.*ln*("CLIENT: all objects requested");  
  
 }  
  
 static void SendListSize(Socket s, int size) throws IOException  
 {  
 s.getOutputStream().write(*SEND\_LIST\_SIZE*);  
 s.getOutputStream().write(size);  
 s.getOutputStream().flush();  
 }  
  
  
 static void SendObject(Socket s, Smile o, int index) throws IOException  
 {  
 OutputStream os = s.getOutputStream();  
 os.write(*SEND\_OBJECT*);  
 os.write(index);  
 ObjectOutputStream oos = new ObjectOutputStream(os);  
 oos.writeObject(o);  
 os.flush();  
 }  
  
 static void SendAllObjects(Socket s, ArrayList<Smile> list) throws IOException  
 {  
 os.flush();  
 }  
  
 static QueryData ListenServer(Socket s) throws IOException, ClassNotFoundException  
 {  
 InputStream is = s.getInputStream();  
  
 Log.*ln*("CLIENT: InputStream size = " + is.available());  
 int code = is.read();  
 QueryData qd = new QueryData();  
 qd.code = code;  
 switch (code)  
 {  
 case(*SEND\_LIST\_SIZE*):  
 qd.data = is.read();  
 break;  
 case(*SEND\_OBJECT*):  
 {  
 QueryData obj = new QueryData();  
 // object index  
 obj.code = is.read();  
  
 ObjectInputStream ois = new ObjectInputStream(is);  
 obj.data = (Smile)ois.readObject();  
 qd.data = obj;  
 }  
 break;  
 case(*SEND\_ALL\_OBJECTS*):  
 {  
 ObjectInputStream ois = new ObjectInputStream(is);  
 qd.data = (ArrayList<Smile>)ois.readObject();  
 }  
 break;  
 default: { qd.code = *UNKNOWN*; qd.data = null; } break;  
 }  
 return qd;  
 }  
  
 static QueryData ListenClient(Socket s) throws IOException, ClassNotFoundException  
 {  
 InputStream is = s.getInputStream();  
  
 q.code = code;  
  
 switch(code){  
 case(*GET\_LIST\_SIZE*): q.data = is.read(); break;  
 case(*GET\_OBJECT*) : q.data = is.read(); break;  
 case(*CLOSE\_CONNECTION*):  
 case(*CLEAR\_LIST*):  
 case(*GET\_ALL\_OBJECTS*): break;  
 default: { q.code = *UNKNOWN*; q.data = null; }  
 }  
 return q;  
 }  
}

**Спецификация протокола взаимодействия**

Описание байта на клиентской стороне:

CLOSE\_CONNECTION = 0; - Закрытие соединения

CLEAR\_LIST = 1; - Очистка списка объектов

GET\_LIST\_SIZE = 2; - Получение размера списка объектов

GET\_OBJECT = 3; - Получение одного объекта из списка (по индексу)

GET\_ALL\_OBJECTS =4; - Получение всех объектов из списка

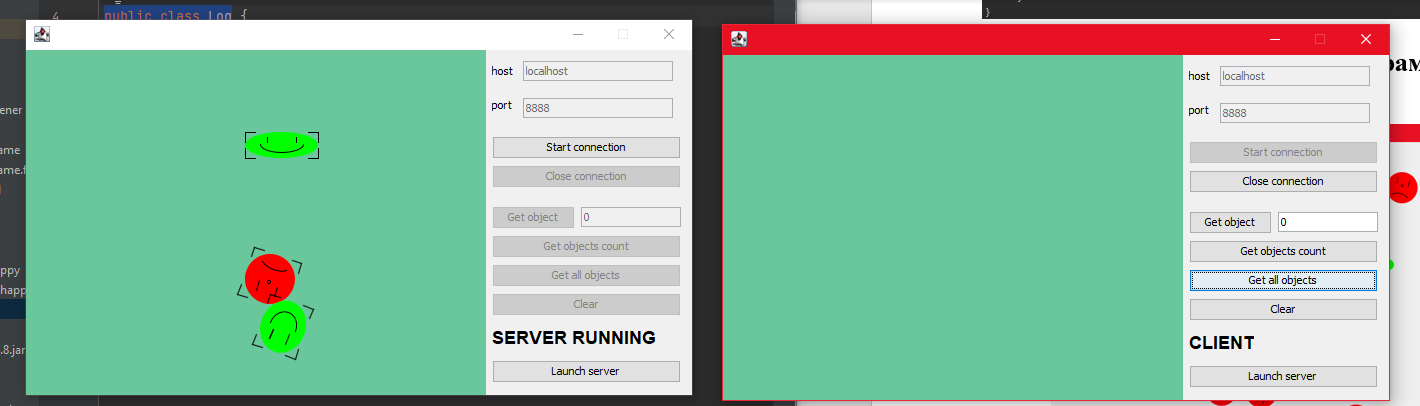
Описание байта на серверной стороне:

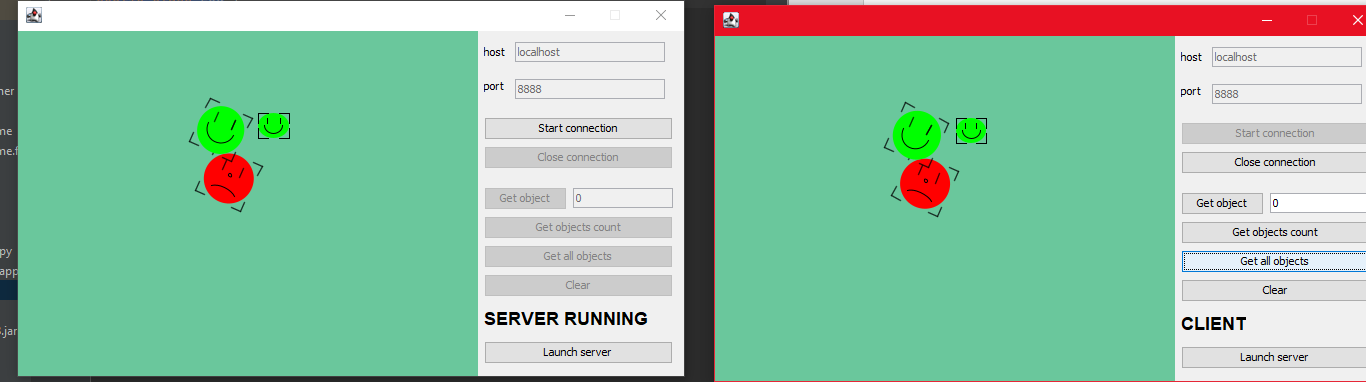
SEND\_LIST\_SIZE = 5; - Отправление полного размера списка объектов

SEND\_OBJECT = 6; - Отправление объекта (по индексу)

SEND\_ALL\_OBJECTS = 7; - Отправление всех объектов

**Пример работы программы**





# Вывод

В ходе выполнения лабораторной работы было разработано приложение с клиентской и серверной частью.

Серверная часть реализуется в потоке и запускается/останавливается при помощи флажка.

Серверная компонента ждет запроса на соединение на фиксированном порте.

Клиентская компонента в другой программе запрашивает соединение с серверной.

Протокол является симметричным (двунаправленным), т.е. каждое приложение может посылать один тот же набор команд: закрытие соединения, очистка вектора объектов, передача сериализованного объекта, запрос размера вектора объектов, запрос объекта с заданным номером из вектора.

Принимаемые объекты отображаются в клиенте и запоминаются в векторе.