package messenger\_client**;**//№3  
import com.google.gson.Gson**;** //1  
import messenger\_network.TCPCOnnectionListener**;**import messenger\_network.TCPConnection**;** // The importation of a TCP socket in Java  
  
import javax.swing.\***;** //2 и т.д  
import java.awt.\***;**import java.awt.event.ActionEvent**;**import java.awt.event.ActionListener**;**import java.awt.event.MouseEvent**;**import java.io.IOException**;**import java.util.ArrayList**;**import java.util.List**;**import static java.awt.Color.*BLUE***;**// класс сообщения с инициализацией строковых переменных для  
class Message {  
 private String TYPE**;** //типа сообщения  
 private String USERNAME**;** //имени отправителя  
 private String USERNAME\_TO**;** //имени получателя  
 private String TEXT**;** //строки самого текста  
 private ArrayList<String> NAMELIST**;** // массив для хранения всех этих строковых данных  
  
 //объявление инициализирующего конструктора класса сообщений со всеми атрибутами, что были в классе выше  
 public Message(String TYPE**,** String USERNAME**,** String USERNAME\_TO**,**String TEXT**,** ArrayList<String> NAMELIST) { //1  
 this.TYPE = TYPE**;** this.USERNAME = USERNAME**;** this.USERNAME\_TO = USERNAME\_TO**;** this.TEXT = TEXT**;** this.NAMELIST = NAMELIST**;** }  
  
 //геттеры, возвращающие значение переменных и массива имен со всеми строковыми данными  
 // Encapsulation, is to make sure that "sensitive" data is hidden from users.  
 public String getTYPE() {  
 return TYPE**;** }  
 public String getUSERNAME() {  
 return USERNAME**;** }  
 public String getUSERNAME\_TO() {  
 return USERNAME\_TO**;** }  
 public String getTEXT() {  
 return TEXT**;** }  
 public ArrayList<String> getNAMELIST() {  
 return NAMELIST**;** }  
}  
  
public class ClientWindow extends JFrame implements ActionListener**,** TCPCOnnectionListener { //№2  
  
  
 static String *IP\_ADDR* = null/\*"192.168.1.220"\*/**;** // переменная для задания статического значения ip - address  
  
 //значения порта, его широта и долгота  
 private static final int *PORT* = **8882;** private static final int *WIDTH* = **400;** private static final int *HIGHT* = **300;** private static String *UserName\_To* = null**;** // по умолчанию строковое значение получателя = 0  
 String UserName **;**/\*System.getProperty("user.name");\*/ //инициализируем строку отправителя  
 private static String *type* = null**;** //инициализируем тип сообщения = 0 по умолчанию  
 private static ArrayList<String> *name\_list* = new ArrayList<String>()**;** //инициализируем массив для списка всех пользователей ICQ  
  
//инициализация конструкторов по умолчанию  
 final JFrame form2 = new JFrame()**;** //для окна логина  
 final JTextField name = new JTextField()**;** //текстового поля для адреса  
 final JTextField ip = new JTextField()**;** //имени пользователя  
 final JButton next = new JButton()**;** //кнопки залогинивания  
  
  
 //инициализация конструкторов по умолчанию  
 final JFrame form1 = new JFrame()**;**//для формы пользователя  
 final JTextArea log = new JTextArea()**;** // для отображения (логовых)принятых и отправленных сообщений в панеле  
 final JTextField fieldinput = new JTextField()**;** //для поля ввода текста сообщения  
 final DefaultListModel listModel = new DefaultListModel()**;**//для отображения списка всех сообщений в текущей сессии  
 final JList list = new JList(listModel)**;** //для отображения списка всех пользователей  
  
 TCPConnection connection**; //создаем объект для соединения** a TCP socket ClientWindow(String ipText**,**String nameText){ //функция клиентского окна с передаваемыми аргументами ранее введенных айпи адреса и имени пользователя  
  
 this.*IP\_ADDR* = ipText**; // сообщаем для машины статистическое значение Айпи адреса конкретного НТТР - заголовка, помещая его в переменную с помощью указателя на нее** this.UserName = nameText**; //то же самое с именем** form1.setDefaultCloseOperation(WindowConstants.*EXIT\_ON\_CLOSE*)**; //#4 set method what to do if we have closed the form1 – we are exited from the application** form1.setSize(*WIDTH***,***HIGHT*)**; //set the size of form #5** form1.setLocationRelativeTo(null)**; //#6**

log.setEditable(false)**; //#7 запрет на редакт поля** to prevent the user from entering text into the wrong text field.log.setLineWrap(true)**; // №7** fieldinput.addActionListener(this)**; //** поле ввода текста сообщения №7form1.add(log**,** BorderLayout.*CENTER*)**; //#8** JScrollPane scroll = new JScrollPane(log)**; //создаем новый объект скроллбара #9** scroll.setVerticalScrollBarPolicy(JScrollPane.*VERTICAL\_SCROLLBAR\_ALWAYS*)**;**

**//задаем поведение конкретно для вертикального скролбара (всегда)** form1.add(fieldinput**,** BorderLayout.*SOUTH*)**; // добавляем на форму поле для ввода распол строго снизу** listModel.addElement("To all")**; //добавляем в начало списка пользователей элемент «Написать всем»** list.setFocusable(false)**; //№10 получение фокуса для поля ввода текстового сообщения с клавы** JTextField fieldnickname = new JTextField(UserName)**; //созд текст поле для отображ текущ зареган юзеров** fieldnickname.setEditable(false)**; //другие юзеры не могут изменять имя конкретного юзера** form1.add(fieldnickname**,** BorderLayout.*NORTH*)**; //закрепляем имя текущего пользователя в самом топе формы** form1.add(list**,** BorderLayout.*EAST*)**; //справа закрепляем лист всех текущих юзеров и возможных получателей** form1.add(scroll)**; // вызов метода, обеспечивающий скролл** list.addMouseListener(new CustomListener())**; //add the listener of mouse`s order(send message while clicking of button)** try {  
 connection = new TCPConnection(this**,** *IP\_ADDR***,** *PORT*)**; //a new object for creating tcp connection, setting the value of ip –address and port,** String msg = null**; //the cur value of string mes is null(the field is empty)**

**//a new object of class message with arguments of name of registered user, the sender name, the receiver name, the beginner value of message(empty) + list of registered users “online” (it`s empty by default and only after initialization we add here other users** Message message = new Message("name\_registration"**,**UserName**,** *UserName\_To***,** msg**,** null)**;** String json = new Gson().toJson(message)**; //#11** connection.sedString(json)**; //#12 the convertion of string to gson object** System.*out*.println("name register "+json)**; //?** String name = "Admin"**; //the name of simulated admin** String msg1 = UserName + " connected"**; //the message of successfully registered name users**

**// new object of class with test admin message to somebody ??** Message message1 = new Message("admin\_message"**,**name**,** *UserName\_To***,** msg1**,** null)**;** String json1 = new Gson().toJson(message1)**; //#11** connection.sedString(json1)**; //#12** System.*out*.println("admin "+json)**; //?** } catch (IOException e) { //for catching different variables exceptions  
 e.printStackTrace()**; //#13** }

// setVisible(boolean visible) — когда окно создается, оно по умолчанию невидимо. Чтобы отобразить окно на экране, вызывается //данный метод с параметром **true**. Если вызвать его с параметром **false**, окно снова станет невидимым.

form1.setVisible(true)**;** Message message = new Message("refresh"**,** UserName**,**null**,**null**,**null)**; //the creation of a new potok for updating** Thread refresh = new Thread(new Runnable() { **//#14 the creation of a new potok with interface runnable**  
 @Override //look in explanation loginpage  
 public void run() { //the function of running of the thread

//пробовать выполнять пока все в норме  
 try {  
 while (true){  
 String json = new Gson().toJson(message)**; //создаем экземпляр класса для сообщениия** connection.sedString(json)**; //включаем наше сообщение в поток** Thread.*sleep*(**1000**)**;**} //усыпляем его на некоторое время   
 } catch (InterruptedException e) { // в случаи сиключений ловим их и выдаем стактрейс  
 e.printStackTrace()**;** }  
 }  
 })**;** refresh.start()**; //перед перезапуском обновляем поток** }  
  
 @Override //look in explanation loginpage  
 public void actionPerformed(ActionEvent e) {

if(*UserName\_To* == null) { //если имя получателя пусто  
 *type* = "all"**; //то тогда считать, что пишем сразу всем** String msg = fieldinput.getText()**; // получаем строковое значение нашего напечатанного сообщения из поля для отпраки** if (msg.equals("")) return**; //если поле пустое то не возвращаем ничего** fieldinput.setText(null)**; //по умолчание поле для отправки сообщений пусто** Message message = new Message(*type***,** UserName**,** *UserName\_To***,** msg**,** null)**;//новый объект класса месседж** String json = new Gson().toJson(message)**; //преобразуем его в объект джейсон библиотеки** connection.sedString(json)**; //включаем этот объект в поток** }  
 if(UserName\_To != null){ //если же имя получателя не пустое  
 if(UserName\_To == "To all") { // и оно выбрано как ко всем

//те же действия что в случаи выше  
 type = "all"**;** String msg = fieldinput.getText()**;** if (msg.equals("")) return**;** fieldinput.setText(null)**;** Message message = new Message(type**,** UserName**,** null**,** msg**,** null)**;** String json = new Gson().toJson(message)**; //сериализация джейсон объекта** connection.sedString(json)**;** } else { // в случаи если пишем конкретному получателю  
 type = "private"**; //тип сообщения – приватный итот же набор действий** String msg = fieldinput.getText()**;** if (msg.equals("")) return**;** fieldinput.setText(null)**;** Message message = new Message(type**,** UserName**,** UserName\_To**,** msg**,** null)**;** String json = new Gson().toJson(message)**;** connection.sedString(json)**;** }  
 }  
 //connection.sedString(fieldnickname.getText() + ": " + msg);  
  
 }  
   
 @Override //look in explanation loginpage  
 public void onConnectionReady(TCPConnection tcpConnection**,** String value) { //функция для отображения готовоности подключения (установлено ли оно)  
  
  
 System.out.println("OnConnectionReady" + value)**; //вывод спец сообщения**// printMeg("Connection ready...");  
 }  
  
 @Override /look in explanation loginpage  
 public void onReceiveString(TCPConnection tcpConnection**,** String value) { //ф – ция получения списка в виде строк с именами пользователей   
  
 Message message = new Gson().fromJson(value**,** Message.class)**; ///десериализация джава объектов** System.out.println("aaaaaaaaaaa"+value)**;** name\_list = message.getNAMELIST()**;** System.out.println("aaaaaaaaaa"+name\_list)**; //кол – во и имена авториз юзеров**  
  
 if(message.getTYPE() != "name\_list" && message.getTEXT() != null) printMeg(value)**;** if(message.getTYPE() == "name\_list") {  
 name\_list = message.getNAMELIST()**;** System.out.println("client name list "+name\_list)**;** int lenght = name\_list.size()**;** listModel.removeAllElements()**;** listModel.addElement("To all")**;** for(int i = **0;** i<lenght**;** i++){  
 listModel.addElement(name\_list.get(i))**;** }  
 }if(message.getUSERNAME() == null) {  
 if(UserName\_To == "To all")  
 name\_list = message.getNAMELIST()**;** System.out.println("client name list "+name\_list)**;** int lenght = name\_list.size()**;** listModel.removeAllElements()**;** listModel.addElement("To all")**;** for(int i = **0;** i<lenght**;** i++) {  
 listModel.addElement(name\_list.get(i))**;** }  
 }  
 }  
  
 @Override  
 public void onDisconnect(TCPConnection tcpConnection) {  
  
 // printMeg("Connection close");  
 }  
  
 @Override  
 public void onException(TCPConnection tcpConnection**,** Exception e) {  
  
 // printMeg("Connection exeprion" + e);  
 }  
  
  
 synchronized void printMeg (String mag){  
  
 Message message1 = new Gson().fromJson(mag**,** Message.class)**;** String name = message1.getUSERNAME()**;** String text = message1.getTEXT()**;** String name\_to = message1.getUSERNAME\_TO()**;** SwingUtilities.invokeLater(new Runnable() {  
 @Override  
 public void run() {  
  
  
 if(message1.getUSERNAME() != null) {  
 if (name\_to != null) {  
  
 log.append(name + " to " + name\_to + " : " + text + "\n")**;** log.setCaretPosition(log.getDocument().getLength())**;** }  
 if (name\_to == null) {  
 log.append(name + " : " + text + "\n")**;** log.setCaretPosition(log.getDocument().getLength())**;** }  
 }  
 }  
 })**;** }  
  
 private class CustomListener implements java.awt.event.MouseListener {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 String name\_to = (String)list.getSelectedValue()**;** if(name\_to != "abc"){  
 UserName\_To = name\_to**;** System.out.println("pereslat lichno "+name\_to)**;** }  
 if(name\_to == "abc") {  
 UserName\_To = null**;** System.out.println("pereslat v obchiy" + name\_to)**;** }  
  
 }  
  
 @Override  
 public void mousePressed(MouseEvent e) {  
  
 }  
  
 @Override  
 public void mouseReleased(MouseEvent e) {  
  
 }  
  
 @Override  
 public void mouseEntered(MouseEvent e) {  
  
 }  
  
 @Override  
 public void mouseExited(MouseEvent e) {  
  
 }  
 }  
}