САНКТ-ПЕТЕРБУРЖСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Кафедра Вычислительной техники

Лабораторная работа №5

Выполнил:
студент II курса группы 2125
Припадчев Артём

Проверит:
Харитонова А.Е.

Санкт-Петербург
2013

Задание: 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 UDP.
- Каждое сообщение на сервере должно обрабатываться в отдельном потоке. Класс потока должен реализовывать интерфейс Runnable.
- Приложение должно быть локализовано на 2 языка - русский и греческий.
- Строки локализации должны храниться в отдельном классе.
- 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.

Код программы

Локализация:

```
String language = new String("en");
String country = new String("US");

Locale currentLocale;
ResourceBundle messages;
currentLocale = new Locale(language, country);
messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
```

Клиент:

```
import java.io.*;
import java.net.*;

public class Client implements Runnable {
   private DatagramSocket datagramSocket = null;
   private final int port = 8001;
   private final String host = "127.0.0.1";
   public static boolean serverIsAvailable;
   public static boolean isBusy = false;

   public void runClient() throws IOException {
      datagramSocket = new DatagramSocket();
      System.out.println("UDPClient: Started");
   }

   public void sendData(byte[] data) throws IOException{
      try {
         DatagramPacket sendPacket = new DatagramPacket(data, data.length, InetAddress.getByName(host), port);
         datagramSocket.send(sendPacket);
      } catch (UnknownHostException ex) {
         System.out.print("Error! Unknown host!");
      }
   }

   public byte[] receiveData() throws IOException {
      byte[] buf = new byte[512];
      DatagramPacket recvPacket = new DatagramPacket(buf, buf.length);
         datagramSocket.setSoTimeout(50);
         datagramSocket.receive(recvPacket);
      buf = recvPacket.getData();
      return buf;
   }

   public void Close() {
```

```
        if (datagramSocket != null) {
            datagramSocket.close();
        }
    }

    @Override
    public void run() {
        serverIsAvailable = false;
        isBusy = true;
        String test = "1.0 1.0 1.0";
        while (!serverIsAvailable) {
            try {
            Lab4.client.sendData(test.getBytes());
            int testAnsw =  Integer.parseInt(new String(Lab4.client.receiveData()).trim());
            }
            catch (Exception ex)
            {
                serverIsAvailable = false;
                continue;
            }
            serverIsAvailable = true;
                if (!Contour.listEmergingPointDraw.isEmpty())
                    for (int i=0; i<Contour.listEmergingPointDraw.size(); i++) {
                        String s = Contour.listEmergingPointDraw.get(i).getX() + " " + Contour.listEmergingPointDraw.get(i).getY() +
" " + Contour.contourRadius;
                        try {
                            Lab4.client.sendData(s.getBytes());
                            Contour.listEmergingPointDraw.get(i).setHit(Integer.parseInt(new String(Lab4.client.receiveData()).trim()));
                        } catch (Exception e) {

                        }
                    }
                if (!Contour.points.isEmpty())
                    for (int i = 0; i < Contour.points.size(); i++) {
                        String s = Contour.points.get(i).getX() + " " + Contour.points.get(i).getY() + " " + Contour.contourRadius;
                        try {
                            Lab4.client.sendData(s.getBytes());
                            Contour.points.get(i).setHit(Integer.parseInt(new String(Lab4.client.receiveData()).trim()));
                        } catch (Exception e) {
                        }
                    }
                Lab4.contourRepaint();
        }
        isBusy = false;
    }
}
```

Сервер:

```
import java.io.*;
public class ServerApp {
    public static void main(String[] args) {
        try {
            Server server = new Server(8001);
            new Thread(server).start();
            System.out.println("Server is starting.");
            BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
            String str=input.readLine();
            if(str.toUpperCase() == "CLOSE")
                server.CloseServer();
            System.out.println("Server is closed.");

        }
        catch (Exception e)
```

```
        {
            System.out.print("Error! Server is not starting:" + e.toString());
        }
    }
}
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class Server implements Runnable{
    DatagramSocket socket;

    public Server(int port) throws Exception{
        socket = new DatagramSocket(port);
    }

    @Override
    public void run(){
        while(true)
        {
            try
            {
                StartServer();
            }
            catch (Exception e)
            {
                System.out.print("Server is dead in start method" + e.toString());
                if(socket!=null)
                    socket.close();
                break;
            }
        }
    }

    private void StartServer() throws Exception
    {
        byte[] receiveData = new byte[512];
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        socket.receive(receivePacket);
        new Thread(new Responder(socket,receivePacket)).start();
    }
    public void CloseServer()
    {
        if(socket!=null)
            socket.close();
    }
}
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.util.LinkedList;
import java.util.Scanner;

public class Responder implements Runnable {
    DatagramSocket socket = null;
    DatagramPacket packet = null;

    public Responder(DatagramSocket socket, DatagramPacket packet)
    {
        this.socket = socket;
        this.packet = packet;

    }

    @Override
```

```
    public void run() {
        String  data = new String(packet.getData()).trim();
        Scanner scanner = new Scanner(data);
        int inside = hitInArea(new Mark(Double.parseDouble(scanner.next()),Double.parseDouble(scanner.next())),
Double.parseDouble(scanner.next()));
        String sendData = Integer.toString(inside);
        DatagramPacket response = new DatagramPacket(sendData.getBytes(), sendData.getBytes().length,packet.getAddress(),
packet.getPort());
        try
        {
        socket.send(response);
        }
        catch (Exception e)
        {
            System.out.print("Server crashed in send data" + e.toString());
            if(socket!=null)
                socket.close();
        }
    }

    public int hitInArea(Mark mark, double contourRadius)
    {
        LinkedList<Figure> listFigure;
        listFigure = new LinkedList<Figure>();
        listFigure.add(new FRectangle(0, 0, contourRadius, -contourRadius));
        listFigure.add(new FQuarterOfCircle(contourRadius, Quarter.Third));
        listFigure.add(new FTriangle(new Mark(0, 0), new Mark(-contourRadius, 0), new Mark(0, contourRadius / 2)));

        int inside = 0;
        int countHitOnBorder = 0;
        for (Figure figure : listFigure) {
            if (figure.hitInArea(mark)) {
                inside = 1;

            }
            if (figure.hitOnBorder(mark))
                countHitOnBorder++;
        }
        if (countHitOnBorder == 1) inside = 0;
        else if (countHitOnBorder > 1)
            inside = 1;
        return inside;
    }
}
```

Вывод: в ходе работы были рассмотрены базовые принципы локализации приложений, а также базовые принципы взаимодействия по сети.