Федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский государственный университет

информационных технологий, механики и оптики

Факультет прикладной информатики и компьютерных технологий

ITMO UNIVERSITY

Программирование Интернет Приложений

Лабораторная работа №5

Вариант: 2017

Группа: P3218

Студент: Петкевич Константин Вячеславович

Преподаватель: Гаврилов Антон Валерьевич

г. Санкт-Петербург

Текст Задания: *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 TCP.*
- *Для данных в сообщении клиента должен использоваться тип double.*
- *Для данных в ответном сообщении сервера должен использоваться тип int.*
- *Каждое сообщение на сервере должно обрабатываться в отдельном потоке. Класс потока должен быть унаследован от класса Thread.*
- *Приложение должно быть локализовано на 2 языка - русский и испанский.*
- *Строки локализации должны храниться в текстовом файле.*
- *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.*

Листинг программы

```
public class Client {
    public static Socket get() throws IOException {
            InetAddress ipAddress = InetAddress.getByName(App.CLIENT_ADDR);
            Socket socket = new Socket(ipAddress, App.SERVER_PORT);
            return socket;
    }
}

public class
ServApp extends Thread {
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(App.SERVER_PORT);
        Socket socket = ss.accept();
        System.out.println("fsdfsd");
        new Thread(new Server(socket)).start();
    }
}
public class Server extends Thread {
Socket socket;
Server(Socket socket) {
    this.socket = socket;
}
    private server.Shape recountShape(int r) {
        return new server.Shape(new Area[] {
                new Area(point -> point.Y() >= -r // rect
                        && point.Y() <= 0,
                        0, r),
                new Area(point -> point.Y() <= -point.X() + r/2 // triangle
                        && point.Y() >= 0,
                        0, r/2),
                new Area(point -> point.Y() <= Math.pow(Math.pow(-r/2, 2) - Math.pow(point.X(), 2),
0.5) // circ
```

```
                && point.Y() >= 0,
                -r/2, 0)
    });
}
public String receiveMessage(int bufferSize) throws IOException {
        byte[] receiveData = new byte[bufferSize];
        InputStream sin = socket.getInputStream();
        DataInputStream in = new DataInputStream(sin);
        in.read(receiveData);
        return new String(receiveData);
}
public void sendMessage(String message) throws IOException {
    byte[] sendData = message.getBytes();
    OutputStream sout = socket.getOutputStream();
    DataOutputStream out = new DataOutputStream(sout);
    out.write(sendData);
    out.flush();
}
public void run() {
    float[] data = new float[3]; //x,y,r
    while (true) {
        String message = null;
        try {
            message = receiveMessage(App.BUFFER_SIZE).trim();
        } catch (IOException e) {
            e.printStackTrace();
        }
            Pattern pattern = Pattern.compile("-?\\d+[.]\\d+");
            Matcher matcher = pattern.matcher(message);
            for (int i = 0; i < 3; i++) {
                if (!matcher.find()) {
                    throw new IllegalArgumentException();
                } else {
```

```
                    data[i] = new Float(matcher.group());
                }
            }
            Boolean fits = recountShape((int) data[2]).contains(new GraphPoint(data[0], data[1]));
            String result = String.format("%s %s", fits, (int) data[2]);
        try {
            sendMessage(result);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
  }
}
public final class App extends Thread {
    public static final int SERVER_PORT = 4444;
    public static final String CLIENT_ADDR = "localhost";
    public static final int BUFFER_SIZE = 1024;
    private JFrame frame;
    private JButton setButton;
    private JButton localeButton;
    private ResourceBundle labelBundle;
    private ArrayList<Locale> locales = new ArrayList<Locale>();
    private Locale currentLocale;
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new App());
    }
    public void run() {
        locales.add(new Locale("ru"));
        locales.add(new Locale("esp"));
        Locale.setDefault(locales.get(1));
        this.currentLocale = locales.get(1);
        labelBundle = ResourceBundle.getBundle("Language_ru");
        Graph graph = null;
```

```
        try {
            graph = new Graph();
        } catch (IOException e) {
            e.printStackTrace();
        }
        CoordinatesPanel coordsPanel = new CoordinatesPanel(graph);
        //          ServApp Frame           //
        this.frame = new JFrame(labelBundle.getString("Title"));
        frame.setLayout(new GridLayout(1, 2));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.setMinimumSize(new Dimension(800, 400));
        frame.setPreferredSize(new Dimension(800, 400));
        frame.setVisible(true);
        //          client.Graph            //
        frame.add(graph);
        //          Bottom Panel            //
        JPanel bottomPanel = new JPanel();
        bottomPanel.setLayout(new BoxLayout(bottomPanel, BoxLayout.PAGE_AXIS));
        this.setButton = new
JButton(ResourceBundle.getBundle("Language_ru").getString("SetKey"));
        setButton.setPreferredSize(new Dimension(200,40));
        coordsPanel.setAlignmentX(Component.CENTER_ALIGNMENT);
        bottomPanel.add(coordsPanel);
        setButton.setAlignmentX(Component.CENTER_ALIGNMENT);
        coordsPanel.setLayout(new BoxLayout(coordsPanel, BoxLayout.PAGE_AXIS));
        bottomPanel.add(setButton);
        this.localeButton = new
JButton(ResourceBundle.getBundle("Language_ru").getString("Language"));
        localeButton.setPreferredSize(new Dimension(200,40));
        localeButton.setAlignmentX(Component.CENTER_ALIGNMENT);
        localeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
```

```
            changeLocale();
        }
    });
    bottomPanel.add(localeButton);
    frame.add(bottomPanel);
    PointController pointController = new PointController(
            coordsPanel.getXField(),
            coordsPanel.getYField(),
            graph
    );
    graph.addMouseListener(pointController);
    setButton.addActionListener(pointController);
  }
  public void changeLocale() {
    currentLocale = locales.get((locales.indexOf(currentLocale) + 1)  % locales.size());
    Locale.setDefault(currentLocale);
    ResourceBundle.clearCache();
    this.localeButton.setText(ResourceBundle.getBundle("Language",
currentLocale).getString("Language"));
    this.setButton.setText(ResourceBundle.getBundle("Language",
currentLocale).getString("SetKey"));
    this.frame.setTitle(ResourceBundle.getBundle("Language",
currentLocale).getString("Title"));
  }
}
```

Вывод: При написании лабораторный работы я изучил два протокола TCP и UDP. Их разница в так называемой “гарантии доставки”. TCP требует отклика от клиента, которому доставлен пакет данных, подтверждения доставки, и для этого ему необходимо установленное заранее соединение. Также протокол TCP считается надежным. TCP исключает потери данных, дублирование и перемешивание пакетов, задержки. UDP все это допускает, и соединение для работы ему не требуется. Процессы, которым данные передаются по UDP, должны обходиться полученным, даже и с потерями. TCP контролирует загруженность соединения, UDP не контролирует ничего, кроме целостности полученных датаграмм.