**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**отчет**

**по лабораторной работе №1**

**по дисциплине «Архитектура распределенных вычислительных систем»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 4303 |  | Осмоловский П.Д.  Полушина Ю.И. |
| Преподаватель |  | Томша А.Э. |

Санкт-Петербург

2016

**Содержание**

[**1.** **Формулировка задания** 3](#_Toc467253844)

[**2.** **Пример выполнения задания и его анализ** 3](#_Toc467253845)

[**2.1.** **Пример выполнения** 3](#_Toc467253846)

[**3.** **Формальная постановка задачи** 3](#_Toc467253847)

[**3.1.** **Исходные данные** 3](#_Toc467253848)

[**3.2.** **Результирующие данные** 3](#_Toc467253849)

[**4.** **Спецификация программы** 5](#_Toc467253850)

[**4.1.** **Исходные данные** 5](#_Toc467253851)

[**4.1.1.** **Ограничения на исходные данные** 5](#_Toc467253852)

[**4.2.** **Выходные данные** 5](#_Toc467253853)

[**4.2.1.** **Состав выходных данных** 5](#_Toc467253854)

[**4.2.2.** **Место и форма представления выходных данных** 5](#_Toc467253855)

[**4.3.** **Алгоритм работы программы** 5](#_Toc467253856)

[**5.** **Тестирование программы** 5](#_Toc467253857)

[**6.** **Вывод** 8](#_Toc467253858)

# Постановка задачи

Реализовать сервис получения текущей погоды с какого-либо погодного сервиса в сети Internet. Клиент, обращается к сервису и отправляет свой город, а в ответ получает данные

о текущей погоде в этом городе. Сервер должен реализовывать технологию кэширования состояния погоды в городах, с обновлением в 1 час.

# Пример выполнения задания и его анализ

## **Пример выполнения**

Входные данные:

[название города и отправка строки серверу]

Выходные данные:

[получение клиентом значения погоды в текущем городе]

# Формальная постановка задачи

## **Исходные данные**

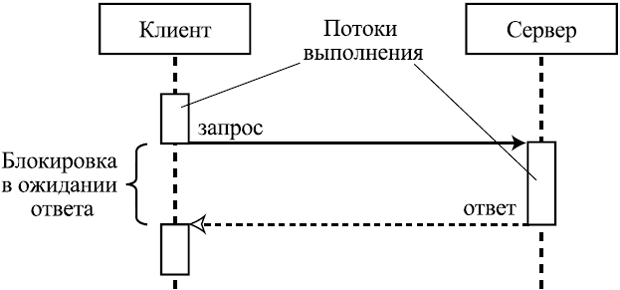
Исходных данные представляют собой строку, она не должна содержать кириллицу.

## **Результирующие данные**

Результатом выполнения работы является значение погоды, которое выводится на панель.

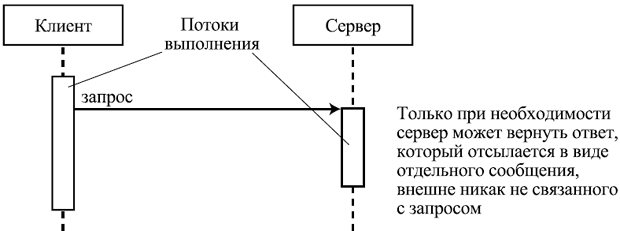
**Реализация**

Взаимодействие по сети было реализовано с помощью сокетов. Был разработан сервер и клиент. Для первого этапа организовано синхронное взаимодействие компонентов, где сервер имеет возможность обработать лишь запрос одного клиента в единицу времени, это представлено на следующей схеме:



Сервер реализован следующим образом: при запросе клиента проверяется наличие прогноза погоды запрашиваемого города в кэше, в случае его отсутствия, или неактуальности (подразумевается «просрочка» более чем на 2 часа), сервер обращается к сервису YahooWeather, откуда получает актуальные данные по метеообстановке в запрошенном городе, данные отправляются клиенту, а также записываются в кэш. При последующих обращениях клиентов данная процедура повторяется.

Далее был реализован принцип асинхронного взаимодействия сервера и клиентов. В этом случае меняется количество клиентов, запросы которых может обработать сервер в единицу времени. Таким образом сервер доступен для работы сразу нескольким клиентам, что существенно увеличивает его производительность. Для этого при обращении каждого клиента создается отдельный объект Сервер, который работает с этим клиентом.



# Спецификация программы

## **Исходные данные**

## **Ограничения на исходные данные**

## Входные данные представляют собой строку, которая может представлять собой идентификатор города или его название. Строка не должна содержать символы кириллицы, в противном случае мы получим уведомление об ошибке.

## **Выходные данные**

## **Состав выходных данных**

## Выходные данные это строка текста, в которой содержится значение погоды в текущем городе. В случае непредвиденной ошибки мы получим строку «error».

## **Место и форма представления выходных данных**

## Данные выводят выполнения работы выводятся в визуальную панель.

## 

## **Алгоритм работы программы**

**Программа выполняет следующую последовательность действий:**

1.Подключение сервера.

2.Подключение клиента.

3.Соединение клиента и сервера.

4.Отправка названия города в поток вывода (в отдельном потоке или в главном).

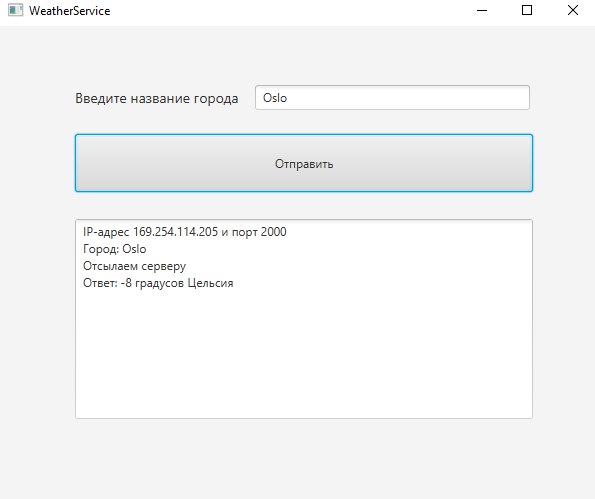
5.Получение данных из потока вывода.

6.Отсоединение от клиента.

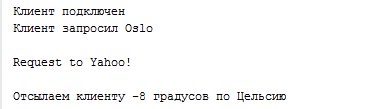
**Демонстрация работы**

1. Запросим прогноз города, которого нет в кэше сервера.

Клиент:

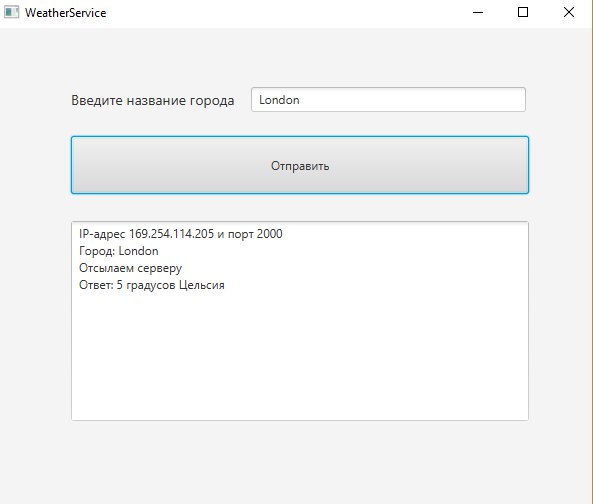


Лог сервера:

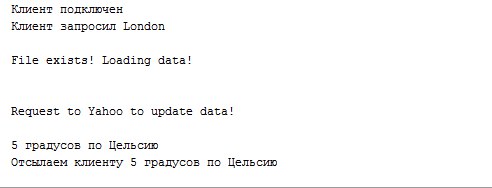


1. Запросим прогноз в городе, для которого ранее были кэшированы данные, но они устарели.

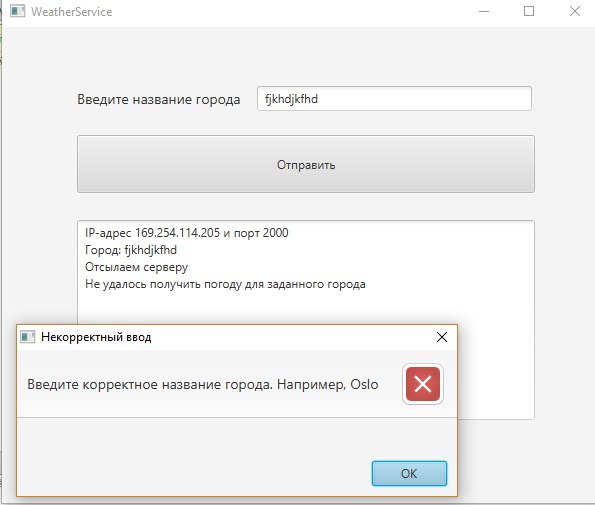
Клиент:



Лог сервера:



Проверим, как сервер отреагирует на запрос, содержащий некорректные данные:



**Вывод**

В ходе выполнения лабораторной работы мы изучили различные методы реализации взаимодействия сервера и клиента, основанных на технологии сокетов.

**Приложение**

Файл ServerS.java

package weather;  
  
import com.github.fedy2.weather.data.Condition;  
  
import javax.xml.bind.JAXBException;  
import java.io.\*;  
import java.net.InetAddress;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.nio.charset.Charset;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
import java.util.Date;  
import java.util.List;  
import java.util.Locale;  
  
import static java.lang.Thread.*NORM\_PRIORITY*;  
public class Server extends Thread {  
 private Socket socket;  
 private int num;  
  
 public static void main(String[] args) {  
 int port = 2000;  
 try {  
 int i = 0; // счётчик подключений  
  
 ServerSocket ss = new ServerSocket(port); // создаем сокет сервера и привязываем его к вышеуказанному порту  
 System.*out*.println("Ожидаем клиентов");  
  
 // слушаем порт  
 while (true) {  
 // ждём нового подключения, после чего запускаем обработку клиента  
 // в новый вычислительный поток и увеличиваем счётчик на единичку  
 new Server(i, ss.accept());  
 System.*out*.println("Клиент подключен");  
 i++;  
 }  
 } catch (Exception e) {  
 System.*out*.println("Ошибка: " + e); // вывод исключений  
 e.printStackTrace();  
 }  
 }  
  
 public Server(int num, Socket s) {  
 // копируем данные  
 this.num = num;  
 this.socket = s;  
  
 // и запускаем новый вычислительный поток  
 setDaemon(true);  
 setPriority(*NORM\_PRIORITY*);  
 start();  
 }  
  
 public void run() {  
 try {  
 String responseToClient = null;  
 String request = null;  
  
 // Берем входной и выходной потоки сокета, теперь можем получать и отсылать данные клиенту.  
 InputStream sin = socket.getInputStream();  
 OutputStream sout = socket.getOutputStream();  
  
 // Конвертируем потоки в другой тип, чтоб легче обрабатывать текстовые сообщения.  
 DataInputStream in = new DataInputStream(sin);  
 DataOutputStream out = new DataOutputStream(sout);  
  
 request = in.readUTF(); // ожидаем пока клиент пришлет строку текста.  
 System.*out*.println("Клиент запросил " + request);  
 // check if cache/line.txt exist  
  
 String filePathString = "src/main/cache/" + request + ".txt";  
  
 List<String> resIfE = null; // response if file exists  
  
  
 if (new File(filePathString).isFile() && new File(filePathString).exists() && !new File(filePathString).isDirectory()) {  
 System.*out*.println("\nFile exists! Loading data!\n");  
 try {  
 resIfE = Files.*readAllLines*(Paths.*get*(filePathString), Charset.*defaultCharset*());  
  
 // date  
 String stringDate = resIfE.get(0);  
 String stringDateFormat = "EEE MMM dd HH:mm:ss z yyyy";  
 SimpleDateFormat format = new SimpleDateFormat(stringDateFormat, Locale.*US*);  
 Date fileDate = format.parse(stringDate);  
  
  
 DateFormat dateFormat = new SimpleDateFormat("EEE MMM dd HH:mm:ss z yyyy", Locale.*US*);  
 Calendar cal = Calendar.*getInstance*();  
 cal.add(Calendar.*HOUR*, -1);  
  
 boolean needToUpdate = cal.getTime().after(fileDate);  
  
 if (needToUpdate) {  
  
 System.*out*.println("\nRequest to Yahoo to update data!\n");  
 responseToClient = logWeather(filePathString, request);  
 } else {  
 responseToClient = resIfE.get(1);  
 }  
  
 System.*out*.println(responseToClient + " градусов по Цельсию");  
 } catch (Exception ex) {  
 responseToClient = "error";  
 ex.printStackTrace();  
 }  
 } else {  
 System.*out*.println("\nRequest to Yahoo!\n");  
 try {  
 responseToClient = logWeather(filePathString, request);  
 } catch (NullPointerException ex) {  
 responseToClient = "error";  
 ex.printStackTrace();  
 }  
 }  
  
 System.*out*.println("Отсылаем клиенту " + responseToClient + " градусов по Цельсию");  
 out.writeUTF(responseToClient); // отсылаем клиенту обратно ту самую строку текста.  
 out.flush(); // заставляем поток закончить передачу данных.  
 } catch (Exception x) {  
 x.printStackTrace();  
 }  
 }  
  
  
 public String logWeather(String filePathString, String request) throws java.io.FileNotFoundException {  
 File f = new File(filePathString);  
 PrintWriter outWriter = new PrintWriter(f);  
  
 Weather newWeather = null; // get weather  
 try {  
 newWeather = new Weather(request);  
 } catch (JAXBException e) {  
 e.printStackTrace();  
 System.*out*.println("JAXBException");  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*out*.println("IOException");  
 }  
  
 outWriter.println(newWeather.showDate().toString());  
 outWriter.println(Integer.*toString*(newWeather.showTemp()));  
 outWriter.close();  
 return Integer.*toString*(newWeather.showTemp());  
 }  
}

Файл Server.java

package weather;  
  
import javax.xml.bind.JAXBException;  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.nio.charset.Charset;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
import java.util.Date;  
import java.util.List;  
import java.util.Locale;  
public class ServerS {  
 public static void main(String[] ar) {  
 int port = 2000;  
 try {  
 String responseToClient = null;  
 String request = null;  
  
 ServerSocket ss = new ServerSocket(port); // создаем сокет сервера и привязываем его к вышеуказанному порту  
 System.*out*.println("Ожидаем клиентов");  
  
 while (true) {  
 Socket socket = ss.accept(); // заставляем сервер ждать подключений и выводим сообщение когда кто-то связался с сервером  
 System.*out*.println("Клиент подключен");  
  
 // Берем входной и выходной потоки сокета, теперь можем получать и отсылать данные клиенту.  
 InputStream sin = socket.getInputStream();  
 OutputStream sout = socket.getOutputStream();  
  
 // Конвертируем потоки в другой тип, чтоб легче обрабатывать текстовые сообщения.  
 DataInputStream in = new DataInputStream(sin);  
 DataOutputStream out = new DataOutputStream(sout);  
 request = in.readUTF(); // ожидаем пока клиент пришлет строку текста.  
 System.*out*.println("Клиент запросил " + request);  
 // check if cache/line.txt exist  
  
 String filePathString = "src/main/cache/" + request + ".txt";  
  
 List<String> resIfE = null; // response if file exists  
  
 if (new File(filePathString).isFile() && new File(filePathString).exists() && !new File(filePathString).isDirectory()) {  
 System.*out*.println("\nFile exists! Loading data!\n");  
 try {  
 resIfE = Files.*readAllLines*(Paths.*get*(filePathString), Charset.*defaultCharset*());  
  
 // date  
 String stringDate = resIfE.get(0);  
 String stringDateFormat = "EEE MMM dd HH:mm:ss z yyyy";  
 SimpleDateFormat format = new SimpleDateFormat(stringDateFormat, Locale.*US*);  
 Date fileDate = format.parse(stringDate);  
  
  
 DateFormat dateFormat = new SimpleDateFormat("EEE MMM dd HH:mm:ss z yyyy", Locale.*US*);  
 Calendar cal = Calendar.*getInstance*();  
 cal.add(Calendar.*HOUR*, -1);  
  
 boolean needToUpdate = cal.getTime().after(fileDate);  
  
 if (needToUpdate) {  
 System.*out*.println("\nRequest to Yahoo to update data!\n");  
 responseToClient = *logWeather*(filePathString, request);  
 } else {  
 responseToClient = resIfE.get(1);  
 }  
  
 System.*out*.println(responseToClient + " градусов по Цельсию");  
 } catch (Exception ex) {  
 responseToClient = "error";  
 ex.printStackTrace();  
 }  
 } else {  
 System.*out*.println("\nRequest to Yahoo!\n");  
 try {  
 responseToClient = *logWeather*(filePathString, request);  
 } catch (NullPointerException ex) {  
 responseToClient = "error";  
 ex.printStackTrace();  
 }  
 }  
 System.*out*.println("Отсылаем клиенту " + responseToClient + " градусов по Цельсию");  
 out.writeUTF(responseToClient); // отсылаем клиенту обратно ту самую строку текста.  
 out.flush(); // заставляем поток закончить передачу данных.  
 }  
 } catch (Exception x) {  
 x.printStackTrace();  
 }  
 }  
  
 public static String logWeather(String filePathString, String request) throws java.io.FileNotFoundException {  
 File f = new File(filePathString);  
 PrintWriter outWriter = new PrintWriter(f);  
  
 Weather newWeather = null; // get weather  
 try {  
 newWeather = new Weather(request);  
 } catch (JAXBException e) {  
 e.printStackTrace();  
 System.*out*.println("JAXBException");  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*out*.println("IOException");  
 }  
  
 outWriter.println(newWeather.showDate().toString());  
 outWriter.println(Integer.*toString*(newWeather.showTemp()));  
 outWriter.close();  
 return Integer.*toString*(newWeather.showTemp());  
 }  
}

Файл Client.java

package sample;

import javafx.fxml.FXML;

import javafx.scene.control.Alert;

import javafx.scene.control.TextArea;

import javafx.scene.control.TextField;

import java.io.\*;

import java.net.InetAddress;

import java.net.Socket;

public class Controller {

@FXML

private TextField textCity;

@FXML

private TextArea textResponse;

private Alert alert;

@FXML

public void sendServer() {

textResponse.clear();

int serverPort = 2000; // здесь обязательно нужно указать порт к которому привязывается сервер.

String address = "169.254.114.205"; // это IP-адрес компьютера, где исполняется наша серверная программа.

// Здесь указан адрес того самого компьютера где будет исполняться и клиент.

if (textCity.getText() == null || textCity.getText().length() == 0) {

error("Пожалуйста, заполните поле");

} else {

try {

InetAddress ipAddress = InetAddress.getByName(address); // создаем объект который отображает вышеописанный IP-адрес.

textResponse.appendText("IP-адрес " + address + " и порт " + serverPort + "\n");

Socket socket = new Socket(ipAddress, serverPort); // создаем сокет используя IP-адрес и порт сервера.

// Берем входной и выходной потоки сокета, теперь можем получать и отсылать данные клиентом.

InputStream sin = socket.getInputStream();

OutputStream sout = socket.getOutputStream();

// Конвертируем потоки в другой тип, чтоб легче обрабатывать текстовые сообщения.

DataInputStream in = new DataInputStream(sin);

DataOutputStream out = new DataOutputStream(sout);

while (true) {

String line = textCity.getText();

textResponse.appendText("Город: "+line+"\n");

textResponse.appendText("Отсылаем серверу\n");

out.writeUTF(line); // отсылаем введенную строку текста серверу.

out.flush(); // заставляем поток закончить передачу данных.

line = in.readUTF(); // ждем пока сервер отошлет строку текста.

if(line.equals("error")) {

textResponse.appendText("Не удалось получить погоду для заданного города\n");

error("Введите корректное название города. Например, Oslo");

} else {

textResponse.appendText("Ответ: " + line + " градусов Цельсия\n");

}

break;

}

} catch (Exception x) {

x.printStackTrace();

}

}

}

public void error(String s) {

alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Некорректный ввод");

alert.setHeaderText(s);

alert.showAndWait();

}

}

Файл Weather.java

package weather;

import com.github.fedy2.weather.YahooWeatherService;

import com.github.fedy2.weather.data.Channel;

import com.github.fedy2.weather.data.Condition;

import com.github.fedy2.weather.data.unit.DegreeUnit;

import javax.xml.bind.JAXBException;

import java.io.IOException;

import java.util.Date;

import java.util.List;

public class Weather {

private Date date;

private int temp;

public Date showDate() {

return date;

}

public int showTemp() {

return temp;

}

public Weather() {

this.date = null;

this.temp = 0;

}

public Weather(String titleCity) throws JAXBException, IOException {

YahooWeatherService service = new YahooWeatherService();

List<Channel> channels = service.getForecastForLocation(titleCity, DegreeUnit.CELSIUS).first(1);

for (Channel channel : channels) {

this.date = channel.getItem().getCondition().getDate();

this.temp = channel.getItem().getCondition().getTemp();

}

}

}