**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**отчет**

**по лабораторной работе №2**

**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** 4](#_Toc467253850)

[**4.1.** **Исходные данные** 4](#_Toc467253851)

[**4.1.1.** **Ограничения на исходные данные** 4](#_Toc467253852)

[**4.2.** **Выходные данные** 4](#_Toc467253853)

[**4.2.1.** **Состав выходных данных** 4](#_Toc467253854)

[**4.2.2.** **Место и форма представления выходных данных** 4](#_Toc467253855)

[**4.3.** **Алгоритм работы программы** 4](#_Toc467253856)

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

[**6.** **Вывод** 7](#_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**

Нужно реализовать: класс Client, интерфейс RemoteGetWeather и класс Server, класс Weather взят из предыдущей лабораторной работы.

Remote method Invocation — механизм, который позволяет вызывать метод удалённого объекта. Согласно ему, все операции по подготовке и передаче данных инкапсулируются в вызываемом методе клиентского объекта-заглушки (stub). Сам же вызов метода ничем не отличается от вызова метода обычного локального объекта, за небольшим исключением: локальные объекты передаются по значению (копии)

* при передаче удалённого (Remote) объекта, если он экспортирован, передаётся stub этого объекта
* передаваемые объекты должны быть Serializable
* кроме всех прочих исключительных ситуаций, при вызове удалённого метода может возбуждаться исключение RemoteException.

Класс сервера реализует удаленный интерфейс RemoteGetWeather.

Remote stub = UnicastRemoteObject.*exportObject*(service, 0);

Экспортируем удалённый объект и получаем stub, посредством которого клиент будет вызывать методы нашего объекта. Второй параметр exportObject — порт, который будет использоваться для соеденения с удалённым объектом, 0 — выбор любого свободного порта. stub нужно передать клиенту.

Клиенту нужно получить серверный stub, чтобы зарегистрироваться.

Registry registry = LocateRegistry.*getRegistry*(**"localhost"**, 2099);  
RemoteGetWeather service = (RemoteGetWeather) registry.lookup(**"sample/WeatherService"**);

Находим удалённый регистратор и запрашиваем у него stub связанный с именем «sample/WeatherService». Первый параметр LocateRegistry.getRegistry("localhost", 2099);— хост (localhost), второй — порт.  
Далее экспортируем клиентский удалённый объект и передадим серверу stub (уже клиентский) — зарегистрируемся. Сервер начнёт передавать ему нужное значение.

# 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).

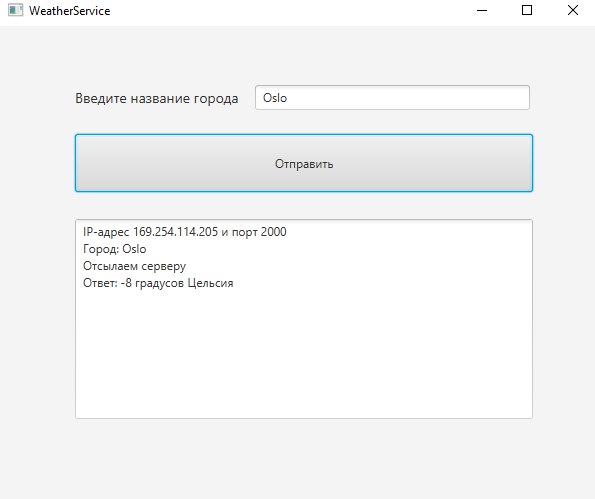
5.Получение данных из потока вывода.

6.Отсоединение от клиента.

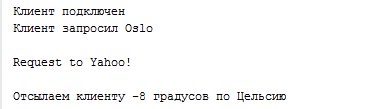
**Демонстрация работы**

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

Клиент:

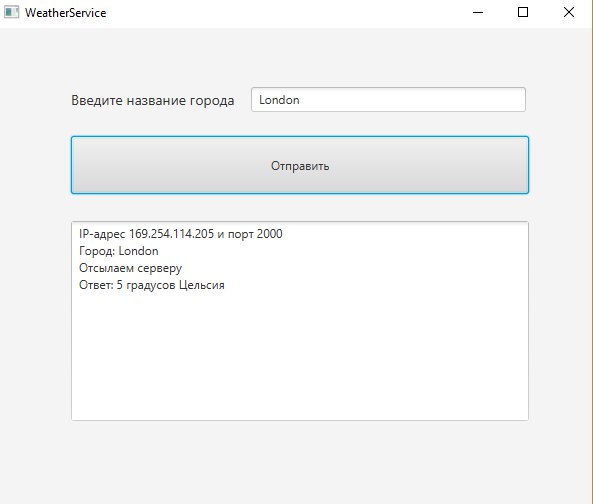


Лог сервера:

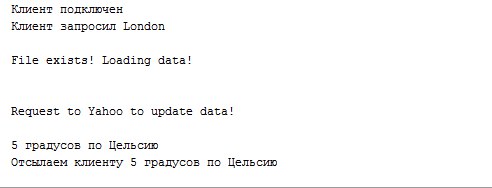


1. Запросим прогноз в городе, для которого ранее были кэшированы данные, но они устарели.

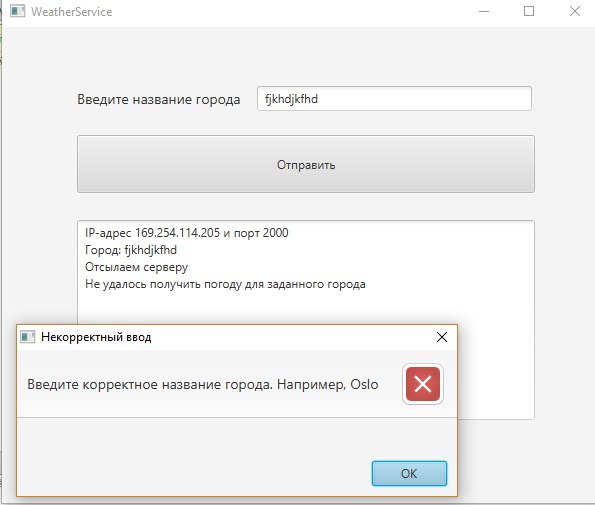
Клиент:



Лог сервера:



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



**Вывод**

В ходе выполнения лабораторной работы мы изучили метод реализации взаимодействия сервера и клиента, основанный на технологии Java RMI.

**Приложение**

Файл Server.java

**public class** Server **implements** RemoteGetWeather {  
  
 **public static void** main(String[] args) {  
 **try** {  
 Registry registry = LocateRegistry.*createRegistry*(2099);  
 RemoteGetWeather service = **new** Server();  
 Remote stub = UnicastRemoteObject.*exportObject*(service, 0);  
 registry.bind(**"sample/WeatherService"**, stub);  
 System.***out***.println(**"Сервер запущен успешно"**);  
 **while** (**true**) {  
 Thread.*sleep*(Integer.***MAX\_VALUE***);  
 }  
 } **catch** (Exception exception) {  
 System.***err***.println(**"Server: "** + exception.toString());  
 }  
 }  
  
 **public** Object getWeather(String city) {  
 String responseToClient = **null**;  
 String filePathString = **"src/main/cache/"** + city + **".txt"**;  
 List<String> resIfE = **null**; *// response if file exists* **if** (**new** File(filePathString).isFile() && **new** File(filePathString).exists() && !**new** File(filePathString).isDirectory()) {  
 ***out***.println(**"\nFile exists! Loading data!\n"**);  
 **try** {  
 resIfE = Files.*readAllLines*(Paths.*get*(filePathString), Charset.*defaultCharset*());  
  
 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);  
 Calendar cal = Calendar.*getInstance*();  
 cal.add(Calendar.***HOUR***, -2);  
  
 **boolean** needToUpdate = cal.getTime().after(fileDate);  
  
 **if** (needToUpdate) {  
 ***out***.println(**"\nRequest to Yahoo to update data!\n"**);  
 responseToClient = Weather.*logWeather*(filePathString, city);  
 } **else** {  
 responseToClient = resIfE.get(1);  
 }  
  
 ***out***.println(responseToClient + **" градусов по Цельсию"**);  
 } **catch** (Exception ex) {  
 responseToClient = **"error"**;  
 ex.printStackTrace();  
 }  
 } **else** {  
 ***out***.println(**"\nRequest to Yahoo!\n"**);  
 **try** {  
 responseToClient = Weather.*logWeather*(filePathString, city);  
 } **catch** (Exception ex) {  
 responseToClient = **"error"**;  
 ex.printStackTrace();  
 }  
 }  
 ***out***.println(**"Отсылаем клиенту "** + responseToClient + **" градусов по Цельсию"**);  
 **return** responseToClient;  
 }  
}

Файл Client.java

**public void** sendServer() {  
 **textResponse**.clear();  
 **try** {  
 Registry registry = LocateRegistry.*getRegistry*(**"localhost"**, 2099);  
 RemoteGetWeather service = (RemoteGetWeather) registry.lookup(**"sample/WeatherService"**);  
 **if** (**textCity**.getText() == **null** || **textCity**.getText().length() == 0) {  
 error(**"Пожалуйста, заполните поле"**);  
 } **else** {  
 **try** {  
 **while** (**true**) {  
 String line = **textCity**.getText();  
 **textResponse**.appendText(**"Город: "** + line + **"\n"**);  
 **textResponse**.appendText(**"Отсылаем серверу\n"**);  
 Object result = service.getWeather(line);  
 String response = result.toString();  
 **if** (response.equals(**"error"**)) {  
 **textResponse**.appendText(**"Не удалось получить погоду для заданного города\n"**);  
 error(**"Введите корректное название города. Например, Oslo"**);  
 } **else** {  
 **textResponse**.appendText(**"Ответ: "** + response + **" градусов Цельсия\n"**);  
 }  
 **break**;  
 }  
 } **catch** (Exception x) {  
 x.printStackTrace();  
 }  
 }  
 } **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();  
}

Интерфейс RemoteGetWeather.java

**public interface** RemoteGetWeather **extends** Remote {  
 Object getWeather(String name) **throws** RemoteException;  
}