

Министерство науки и высшего образования Российской Федерации

Томский государственный университет  
систем управления и радиоэлектроники

В.Г. Резник

# **РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ**

Учебно-методическое пособие

**Контрольный лабораторный практикум студентов ФДО**

Томск  
2021

УДК 004.75(075.8)

ББК 32.973.202я73

Р 344

**Резник, Виталий Григорьевич**

Р 344 Распределенные вычислительные системы. Учебно-методическое пособие. Контрольный лабораторный практикум студентов ФДО / В.Г. Резник. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2021. – 22 с.

В учебно-методическое пособие предназначено для информационной поддержки контрольного лабораторного практикума студентов бакалавриата ФДО по направлению 09.03.01 «Информатика и вычислительная техника» при изучении дисциплины «Распределенные вычислительные системы».

Одобрено на заседании каф. АСУ протокол №\_\_\_\_\_ от \_\_\_\_\_

УДК 004.75

© Резник В. Г., 2021  
© Томск. гос. ун-т систем упр. и  
радиоэлектроники, 2021

# Оглавление

|                                                                   |           |
|-------------------------------------------------------------------|-----------|
| <b>Введение.....</b>                                              | <b>4</b>  |
| <b>1 Лабораторное задание №1.....</b>                             | <b>5</b>  |
| 1.1 Формулировка задания на выполнение работы №1.....             | 5         |
| 1.1.1 Описание функциональности RMI-сервера.....                  | 6         |
| 1.1.2 Описание интерфейса удалённого RMI-объекта на языке Java. 6 |           |
| 1.1.3 Описание реализации удалённого RMI-объекта на языке Java..7 |           |
| 1.2 Реализация RMI-сервера.....                                   | 8         |
| 1.2.1 Подготовка среды реализации RMI-сервера.....                | 8         |
| 1.2.2 Компиляция исходного текста RMI-сервера.....                | 9         |
| 1.2.3 Запуск и тестирование работы RMI-сервера.....               | 10        |
| 1.3 Запуск RMI-сервер в среде ОС MS Windows 10.....               | 12        |
| 1.3.1 Инсталляция JDK версии 17.02.....                           | 12        |
| 1.3.2 Настройка переменных среды.....                             | 13        |
| 1.3.3 Компиляция и старт RMI-сервера.....                         | 14        |
| <b>2 Лабораторное задание №2.....</b>                             | <b>16</b> |
| 2.1 Формулировка задания на выполнение работы №2.....             | 16        |
| 2.1.1 Функциональное описание конкретного RMI-клиента.....        | 16        |
| 2.1.2 Описание интерфейса RMI-клиента.....                        | 17        |
| 2.2 Реализация RMI-клиента.....                                   | 17        |
| 2.2.1 Компиляция исходного текста RMI-клиента.....                | 17        |
| 2.2.2 Запуск и тестирование работы RMI-клиента.....               | 19        |
| 2.3 Запуск RMI-клиента в среде ОС MS Windows 10.....              | 20        |
| 2.4 Контроль выполнения лабораторных работ.....                   | 21        |
| <b>Список использованных источников.....</b>                      | <b>22</b> |

## Введение

Данное учебно-методическое пособие (УМП) предназначено для информационной поддержки лабораторного практикума студентов бакалавриата ФДО по направлению 09.03.01 «Информатика и вычислительная техника» при изучении дисциплины «Распределенные вычислительные системы» (РВС).

УМП содержит учебный и методический материал для самостоятельного выполнения студентами ФДО двух лабораторных заданий, которые проводятся удалённо (в режиме ЭИОС), в период контрольного лабораторного практикума (КЛП), в период и объёме определенным учебным планом ФДО ТУСУР.

Учебный материал данного УМП представлен в виде двух глав, каждая из которых содержит требование к контрольному заданию студента и описание простейшего примера, реализующего последовательность выполнения студентом необходимых работ.

Для студента основными необходимыми условиями успешного прохождения лабораторного практикума являются:

- 1) знание теоретического материала изучаемой дисциплины в объёме базового учебного пособия [1];
- 2) наличие на личной ЭВМ студента инструментального программного обеспечения Java JDK (Java Development Kit) версии не ниже 1.8.

Прохождение студентом контрольного лабораторного практикума (КЛП) предполагает проведение двух видов занятий:

- 1) лекционная часть занятий по дисциплине РВС в объёме **8 часов**;
- 2) дополнительная лекционная часть занятий по дисциплине «Сети и телекоммуникации» (СиТ) в объёме **4 часа**, которая содержит базовый учебный материал по инструментальным средствам языка Java;
- 3) практическая часть по дисциплине РВС в объёме **8 часов**, которая предполагает выполнение двух лабораторных работ, описанных в данном УМП, и оформление единого отчёта, который в последующем оценивается преподавателем.

В целом, теоретическая часть лабораторного практикума опирается на содержание третьей главы учебного пособия [1]: «Объектные распределенные системы».

### Общее задание студенту:

**Реализовать собственное произвольное** распределенное приложение, содержащее программы **RMI-клиента** и **RMI-сервера**, используя теоретическую часть и примеры, изложенные в подразделе «3.3 Технология RMI» учебного пособия [1].

**Изложить** результаты работы в едином отчёте, шаблон и структура которого представлены в файле **Отчет\_РВС\_КЛП.doc**.

# 1 Лабораторное задание №1

**Лабораторная работа №1** содержит описание и реализацию примера *серверной части распределенного приложения*, демонстрирующего применение технологии RMI, которая в настоящее время входит в состав пакета Java SE: JDK SE (инструментального пакета языка Java уровня Standard Edition).

Любое распределенное приложение базируется общей модели «Клиент — Сервер», что обычно предполагает размещение минимум двух приложений на разных ЭВМ, которые могут взаимодействовать между собой посредством сетевого программного обеспечения:

- **приложение-сервер** концентрирует в себе некоторую функциональность, которая отсутствует у **приложения-клиента**, но необходима ему для реализации своей функциональности;
- **приложение-клиент** обычно реализует некоторую функциональность, которая необходима конечному пользователю, но, в силу ограниченности своих функциональных ресурсов, вынуждена обращаться за соответствующими ресурсами к одному или многим серверам.

Реализуя свое собственное распределенное приложение, студент должен чётко себе представлять, какую функциональность будет выполнять приложение-клиент, а какую приложение-сервер.

В любом случае, работу **необходимо начинать с реализации приложения-сервера**, что и демонстрируется в данном учебно-методическом пособии.

Следуя общему заданию лабораторного практикума данная глава пособия описывает процесс разработки приложения-сервера, который основан на объектном подходе реализации распределенных систем. Более того, учитывая многие технологические сложности инструментальных средств технологии CORBA (см. содержание главы 3 учебного пособия [1]), студенту предоставляется возможность использования технологии RMI, которая максимально упрощена в плане своей реализации.

В методическом плане, данная глава разделена на две части:

- 1) описание функциональности примера приложения-сервера;
- 2) пример реализации RMI-сервера.

## 1.1 Формулировка задания на выполнение работы №1

В этом подразделе студенту следует:

- 1) дать формальное описание функциональных возможностей RMI-сервера;
- 2) описать интерфейс удалённого RMI-объекта, которым в дальнейшем могут воспользоваться приложения-клиенты;
- 3) описать реализацию RMI-объекта, обеспечивающего функциональность

заданного интерфейса.

Далее приводится демонстрация конкретного упрощённого примера, на основе которого студент должен придумать и реализовать свой пример.

### 1.1.1 Описание функциональности RMI-сервера

Общая функциональность примера RMI-сервера определяется следующими свойствами:

- 1) сервер предоставляет приложениям-клиентам единственный метод с именем ***getMessage***, с помощью которого клиент может передать серверу произвольное текстовое сообщение;
- 2) сервер сохраняет принятое сообщение ***как последнее принятое***;
- 3) в общем случае, сервер сохраняет следующие данные: ***количество*** принятых сообщений, ***последнее*** и ***предпоследнее*** принятые сообщения;
- 4) с помощью метода ***getMessage***, сервер возвращает приложению-клиента массив из трёх строк: ***количество*** принятых сообщений, ***последнее*** и ***предпоследнее*** принятые сообщения.

### 1.1.2 Описание интерфейса удалённого RMI-объекта на языке Java

Проводим описание интерфейса удалённого RMI-объекта, показанного на листинге 1.1, при следующих условиях:

- 1) интерфейс имеет имя ***Klp2Rmi***;
- 2) интерфейс определяется в пакете ***klp2.rmi***;
- 3) интерфейс расширяет интерфейс ***java.rmi.Remote***.

Листинг 1.1 – Исходный текст интерфейса *Klp2Rmi*

```
package klp2.rmi;

import java.rmi.Remote;

public interface Klp2Rmi extends Remote {
    /**
     * Объявление метода GetMessage
     */
    public String[] getMessage(String str) throws
        java.rmi.RemoteException;
}
```

Обратите внимание, что в дальнейшем данный интерфейс будет использоваться как приложением RMI-сервера, так и приложением RMI-клиента.

Напоминаю, что студент должен сделать своё описание интерфейса, согласно собственному описанию функциональности RMI-сервера.

### 1.1.3 Описание реализации удалённого RMI-объекта на языке Java

В данном пункте приводится описание удалённого объекта, реализующего интерфейс **Rlp2Rmi**, который в последующем будет регистрироваться на общем сервере **rmiregistry**. Исходный текст описания представлен на листинге 1.2, при следующих дополнительно заданных условиях:

- 1) удалённый объект имеет имя **Klp2RmiImpl**;
- 2) расширяет класс **java.rmi.server.UnicastRemoteObject**;
- 3) сохраняет массив строковых объектов с именем **msgs**, содержащий три элемента: **количество** полученных сообщений, **последнее** сообщение и **предыдущее** полученное сообщение.

Листинг 1.2 – Исходный текст класса Klp2RmiImpl

```
package klp2.rmi;

import java.rmi.RemoteException;

public class Klp2RmiImpl extends
    java.rmi.server.UnicastRemoteObject
    implements Klp2Rmi
{
    /**
     * Версия реализации
     */
    private static final long serialVersionUID = 1L;
    /**
     * Сохраняемые классом данные
     */
    private int N = 0; // Количество принятых сообщений
    private String[] msgs = {"0", "", ""}; // Сохраняемые сообщения

    // Конструктор
    protected Klp2RmiImpl()
        throws RemoteException
    {
        super();
    }
    /**
     * Реализация заявленного интерфейса
     */
    public String[] getMessage(String str)
        throws RemoteException
    {
        msgs[0] = Integer.toString(++N);
        msgs[2] = new String(msgs[1]);
        msgs[1] = new String(str);
        System.out.println(msgs[0] + ":" + msgs[1] + ":" + msgs[2]);

        return msgs;
    }
}
```

## 1.2 Реализация RMI-сервера

Вторая часть лабораторного задания №1 содержит описание Java-класса, который программно ассоциируется с реализацией RMI-сервера.

**Основная задача RMI-сервера** — регистрация на сервере *rmiregistry* всех удалённых объектов, обслуживаемых приложением-сервером.

В общем случае, RMI-сервер может регистрировать множество удалённых объектов, определённых разными интерфейсами.

В нашем случае, RMI-сервер будет регистрировать на сервере *rmiregistry* удалённый объект заданный классом *Klp2RmiImpl* и интерфейсом *Klp2Rmi*.

Чтобы выполнить эту часть задания, студент должен:

- 1) **подготовить** инструментальную среду языка Java, включая каталоги размещения исходных текстов приложений сервера и клиента;
- 2) **написать** и правильно разместить исходные тексты всех частей серверной части приложения, включая последующую их компиляцию;
- 3) **произвести** последовательный запуск *rmiregistry* и RMI-сервера, убедившись в отсутствии ошибочных сообщений программного обеспечения.

Далее в следующих трёх пунктах показана реализация всех указанных действий, применительно к рассматриваемому в данном пособии примеру.

### 1.2.1 Подготовка среды реализации RMI-сервера

Подготовку среды реализации приложений сервера и клиента студент должен осуществлять при следующих ограничивающих требованиях, что демонстрируется для примера операционной среды **ОС Linux** и от имени пользователя **vgr**:

- 1) сервер *rmiregistry* будет запускаться по умолчанию: по сетевому адресу *localhost* и порту **1099**;
- 2) исходные тексты всех компонент приложения RMI-сервера размещаются в каталоге */home/vgr/rmiserver*;
- 3) исходные тексты всех компонент приложения RMI-клиента размещаются в каталоге */home/vgr/rmiclient*;
- 4) все действия студента по компиляции и запуску программного обеспечения будут осуществляться в виртуальных терминалах (в среде командной строки).

Исходя из указанных ограничений и теоретического материала, изложенного в главе 2 учебного пособия [1], настройка инструментальной среды языка Java сводится к заданию двух ключевых переменных среды ОС: **JAVA\_HOME** и **CLASSPATH**.

Рекомендуется разместить определение этих переменных в базовом файле конфигурации **.bashrc**, например в виде:

```
export JAVA_HOME=/usr/lib/jvm/default
export CLASSPATH=$HOME/rmiserver:$CLASSPATH
export CLASSPATH=$HOME/rmiclient:$CLASSPATH
```

При использовании операционной среды ОС MS Windows, студенту следует соответствующим образом модифицировать экспорт указанных переменных.

### **1.2.2 Компиляция исходного текста RMI-сервера**

Исходный текст приложения RMI-сервера представлен на листинге 1.3. В представленном тексте использованы следующие дополнительные обозначения:

- 1) класс RMI-сервера представлен именем **Klp2RmiServer**;
- 2) удалённый объект регистрируется под именем **KLP2RMI**.

*Листинг 1.3 – Исходный текст класса Klp2RmiServer*

```
package klp2.rmi;

import java.rmi.Naming;

public class Klp2RmiServer
{
    public static void main(String[] args) {
        System.out.println("Klp2RmiServer: стартует... ");

        try {
            /**
             * Создается экземпляр удаленного объекта класса Klp2RmiImpl.
             */
            Klp2RmiImpl impl =
                new Klp2RmiImpl();

            /**
             * Регистрация экземпляра удаленного объекта
             * на сервере rmiregistry.
             */
            Naming.rebind("//localhost:1099/KLP2RMI", (Klp2Rmi)impl);

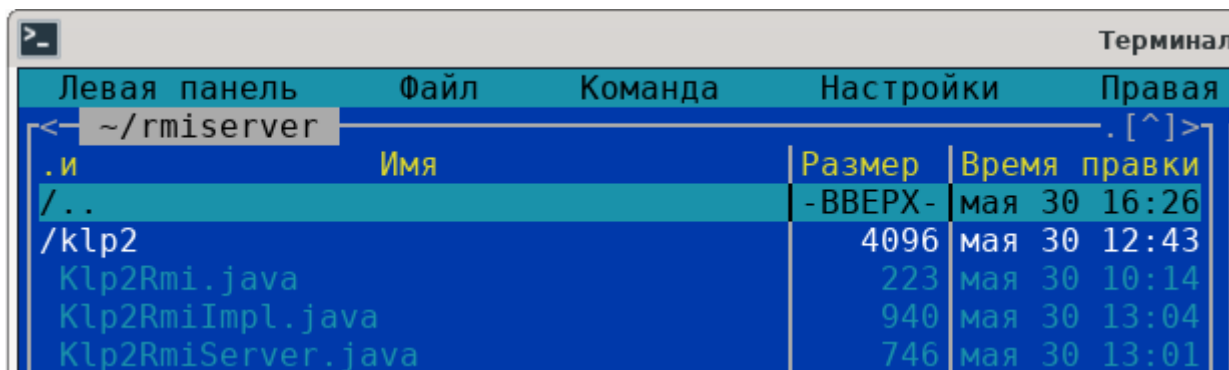
            System.out.println("Klp2RmiServer: стартовал... ");
        }
        catch (Exception e)
```

```

    {
        System.out.println("Исключение: " + e);
    }
}
}

```

Согласно заданным условиям, все компоненты RMI-сервера, определённые листингами 1.1, 1.2, 1.3 и включающие файлы **Klp2Rmi.java**, **Klp2RmiImpl.java**, **Klp2RmiServer.java** помещаются в каталог **/home/vgr/rmiserver**, как это показано на рисунке 1.1.



| Имя                | Размер  | Время правки |
|--------------------|---------|--------------|
| ./                 | -BBERX- | мая 30 16:26 |
| /klp2              | 4096    | мая 30 12:43 |
| Klp2Rmi.java       | 223     | мая 30 10:14 |
| Klp2RmiImpl.java   | 940     | мая 30 13:04 |
| Klp2RmiServer.java | 746     | мая 30 13:01 |

Рисунок 1.1 — Размещение исходных текстов компонент RMI-сервера

### 1.2.3 Запуск и тестирование работы RMI-сервера

Исходные тексты компонент RMI-сервера требуют правильной последовательности компиляции исходных текстов и правильного размещения полученных результатов.

Пошаговые действия правильной компиляции и размещения файлов, следующие.

**Шаг 1.** Создаются каталоги **~/rmiserver/klp2** и **~/rmiserver/klp2/rmi**.

**Шаг 2.** Находясь в каталоге **~/rmiserver**, проводится компиляция файла интерфейса командой **javac Klp2Rmi.java**, после чего полученный после компиляции файл **Klp2Rmi.class** перемещается в каталог **~/rmiserver/klp2/rmi**.

**Шаг 3.** Находясь в каталоге **~/rmiserver**, проводится компиляция файла класса командой **javac Klp2RmiImpl.java**, после чего полученный после компиляции файл **Klp2RmiImpl.class** перемещается в каталог **~/rmiserver/klp2/rmi**.

**Шаг 4.** Находясь в каталоге **~/rmiserver**, проводится компиляция файла класса командой **javac Klp2RmiServer.java**, после чего полученный после компиляции файл **Klp2RmiServer.class** перемещается в каталог **~/rmiserver/klp2/rmi**.

В результате указанных пошаговых действий в каталоге `~/rmiserver/klp2/rmi` правильно размещаются все компоненты приложения RMI-сервера.

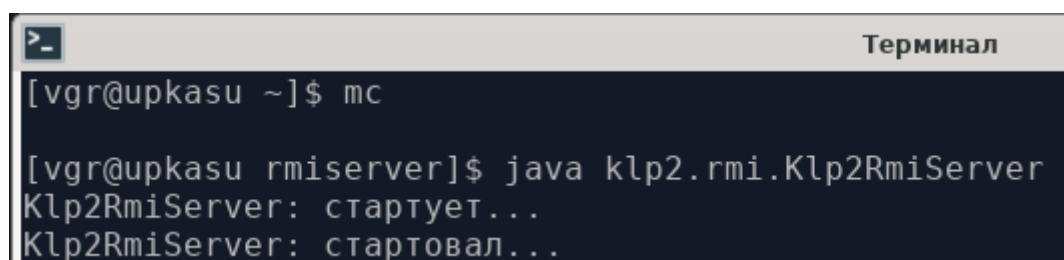
Рабочий запуск RMI-сервера осуществляется в два этапа.

**Этап 1.** Запускается виртуальный терминал, в котором выполняется команда ***rmiregistry***. В результате:

- 1) сервер ***rmiregistry*** запускается по умолчанию ***localhost:1099***;
- 2) сервер ничего не выводит;
- 3) терминал блокируется от ввода новых команд.

**Этап 2.** Запускается виртуальный терминал, в котором выполняется команда ***java klp2.mir.Klp2RmiServer***. В результате:

- 1) RMI-сервер проводит регистрацию удалённого объекта класса ***KlpRmiImpl*** по адресу ***localhost:1099/KLP2RMI***;
- 2) терминал блокируется от ввода новых команд;
- 3) при правильном запуске, сервер выводит в терминале информацию, показанную на рисунке 1.2.



```
>_ Терминал
[vgr@upkasu ~]$ mc

[vgr@upkasu rmiserver]$ java klp2.rmi.Klp2RmiServer
Klp2RmiServer: стартует...
Klp2RmiServer: стартовал...
```

Рисунок 1.2 — Нормальный запуск RMI-сервера в виртуальном терминале

На этом этапе, все работы по лабораторному заданию №1 считаются успешно выполненными.

Студенту следует отразить собственные полученные результаты аналогичного исследования в собственном едином отчёте.

## 1.3 Запуск RMI-сервер в среде ОС MS Windows 10

Как было отмечено ранее, при старте RMI-сервер регистрирует свой интерфейс на сервере *rmiregistry*. При этом, *rmiregistry* должен быть доступен интерфейс приложения. В нашем случае — это файл *Klp2Rmi.class*.

Чтобы всё работало правильно, нужно правильно настроить переменную среды **CLASSPATH**. Обычно неправильная настройка этой переменной является источником ошибок начинающих программистов на языке Java.

Данный подраздел ориентирован на пользователей ОС MS Windows 10 и изложен в трех пунктах:

- а) инсталляция JDK версии 17.02;
- б) настройка переменных среды;
- в) компиляция и старт RMI-сервера.

### 1.3.1 Инсталляция JDK версии 17.02

С сайта [www.oracle.com](http://www.oracle.com) можно скачать «Установщик x64», как это показано на рисунке 1.3.

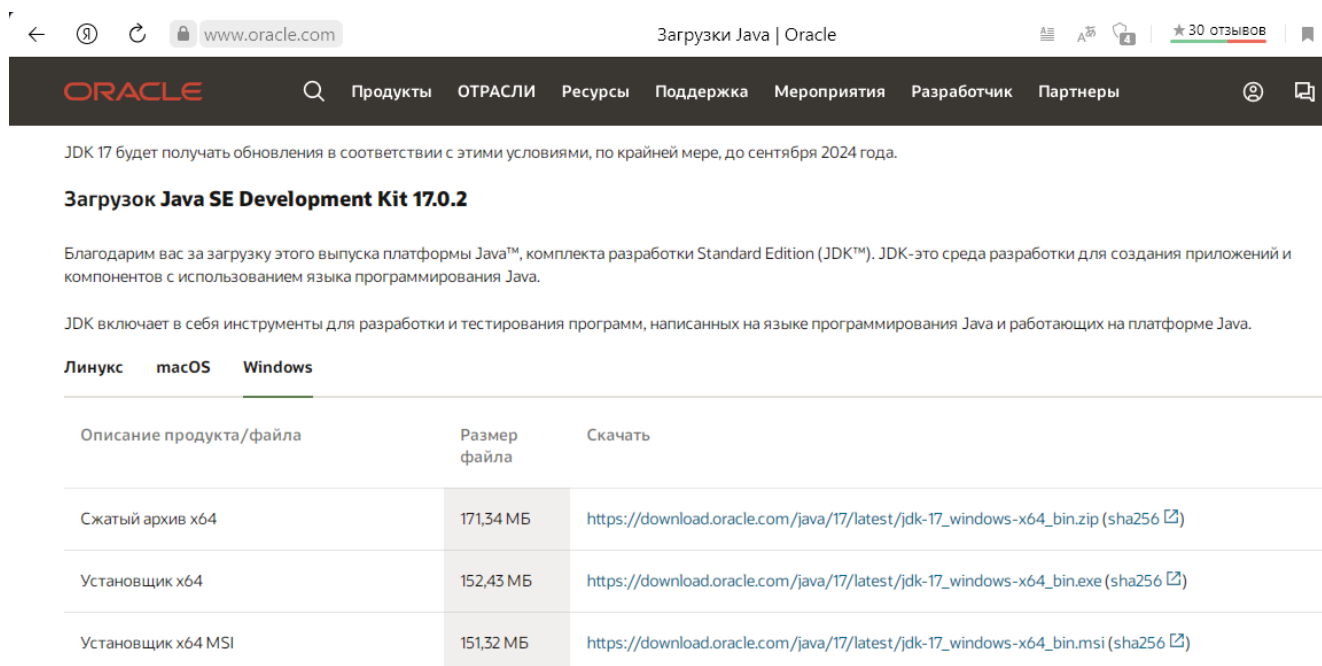


Рисунок 1.3 — Сайт с дистрибутивом JDK версии 17.02

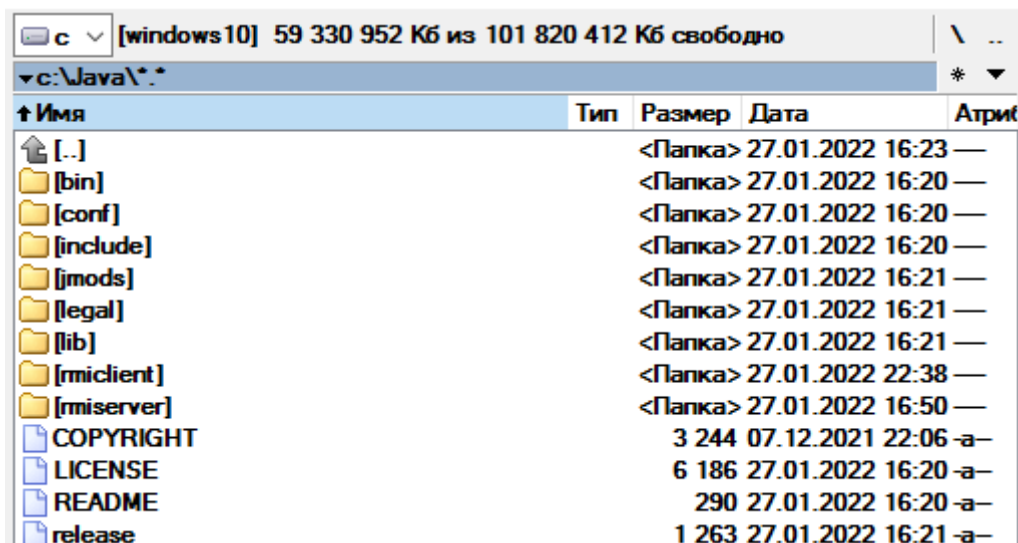
Запустив установщик, следует выбрать каталог дистрибутива или установить JDK по умолчанию.

Далее, исключительно для учебных целей, дистрибутив установлен в каталог **C:\Java\**.

После установки дистрибутива, переходим в каталог **C:\Java\** и создаем там

два каталога, как показано на рисунке 1.4:

- а) **rmiserver** для запуска сервера;
- б) **rmiclient** для запуска программы клиента.



| Имя         | Тип     | Размер | Дата             | Атриб |
|-------------|---------|--------|------------------|-------|
| [..]        | <Папка> |        | 27.01.2022 16:23 | —     |
| [bin]       | <Папка> |        | 27.01.2022 16:20 | —     |
| [conf]      | <Папка> |        | 27.01.2022 16:20 | —     |
| [include]   | <Папка> |        | 27.01.2022 16:20 | —     |
| [jmods]     | <Папка> |        | 27.01.2022 16:21 | —     |
| [legal]     | <Папка> |        | 27.01.2022 16:21 | —     |
| [lib]       | <Папка> |        | 27.01.2022 16:21 | —     |
| [rmiclient] | <Папка> |        | 27.01.2022 22:38 | —     |
| [rmiserver] | <Папка> |        | 27.01.2022 16:50 | —     |
| COPYRIGHT   |         | 3 244  | 07.12.2021 22:06 | -a-   |
| LICENSE     |         | 6 186  | 27.01.2022 16:20 | -a-   |
| README      |         | 290    | 27.01.2022 16:20 | -a-   |
| release     |         | 1 263  | 27.01.2022 16:21 | -a-   |

Рисунок 1.4 — Содержимое каталога C:\Java\

Обычно разработку программного обеспечения не ведут в каталогах дистрибутивов, но для учебных целей это — приемлемо.

### 1.3.2 Настройка переменных среды

Для настройки переменных среды следует воспользоваться соответствующими инструментальными средствами ОС:

- а) на нижней панели рабочего стола следует активировать кнопку главного меню правой кнопкой мыши и, в появившемся контекстном меню, выбрать пункт «Система»;
- б) в появившемся окне «Система» следует активировать пункт меню «Дополнительные параметры системы»;
- в) в появившемся окне «Свойства системы» необходимо активировать кнопку «Переменные среды...»; в результате появится окно «Переменные среды», которое показано на рисунке 1.5.

Редактированию подвергаются две переменные CLASSPATH и Path, что отмечено стрелками на рисунке 1.5:

- а) если переменная CLASSPATH – отсутствует, то её нужно создать с помощью кнопки «Создать...»;
- б) в переменную Path необходимо добавить каталог **C:\Java\bin**; тогда все утилиты языка Java будут запускаться без указания абсолютного пути; в частности, это касается сервера **rmiregistry**;
- в) после завершения установки переменных среды, все окна терминалов необходимо закрыть и запустить снова.



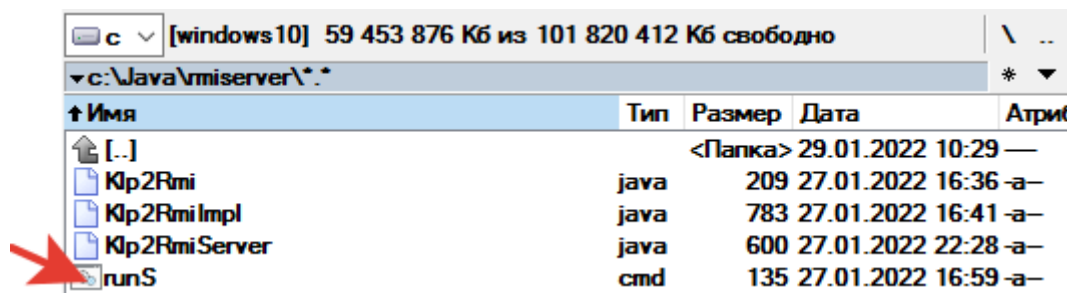


Рисунок 1.6 — Каталог для запуска RMI-сервера

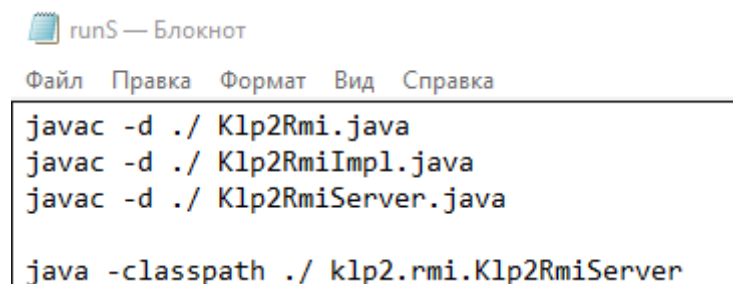


Рисунок 1.7 — Содержимое файла сценария RunS.cmd

Обратите внимание, что три команды **javac** последовательно компилируют исходные тексты RMI-сервера и размещают их в нужных каталогах. Это достигается с помощью опции «**-d**», после чего указывается каталог, относительно которого будет рассматриваться оператор **package** в исходных текстах программ RMI-сервера.

Последняя команда **java** производит запуск RMI-сервера, причем используемая опция также указывает каталог, относительно которого будут искаться классы запускаемой программы.

Для нашего случая опцию «**-classpath**» можно не использовать, поскольку сам запуск проводится в пределах проведенных ранее настроек.

Если не настроить переменные среды, как указано в предыдущем пункте, то RMI-сервер не запустится, поскольку **rmiregistry** не найдет класс **Rlp2Rmi.class** и откажет серверу в регистрации.

## 2 Лабораторное задание №2

Лабораторная работа №2 содержит описание и реализацию примера **клиентской части распределенного приложения**, демонстрирующего применение технологии RMI, входящей в состав пакета Java SE: JDK SE (инструментального пакета языка Java уровня Standard Edition).

**Методически**, лабораторное задание №2 является продолжением разработки распределенного приложения, серверная часть которого уже должна быть реализована в лабораторной работе №1.

Структура и последовательность работ, описанных в данной главе, следует структуре предыдущей главы и также разделена на две части:

- 1) описание функциональности примера приложения-клиента;
- 2) пример реализации RMI-клиента.

### 2.1 Формулировка задания на выполнение работы №2

В общем случае **приложение-клиент** может быть слабо связано с функциональностью **приложения-сервер**. Это подразумевает, что приложение клиента может:

- 1) **использовать** множество удалённых объектов, размещённых на множестве серверов приложений, которые функционально и организационно никак не связаны между собой;
- 2) **включать** очень маленькую функциональность разных отдельных удалённых объектов по сравнению с собственной функциональностью;
- 3) **учитывать** только синтаксическое описание удалённых объектов, которое предоставлено в виде их интерфейсов.

В рамках выполнения лабораторной работы, студенту не следует сильно расширять функциональность своего приложения-клиента. Достаточно продемонстрировать адекватное использование методов, уже описанных во время выполнения задания №1.

#### 2.1.1 Функциональное описание конкретного RMI-клиента

Для учебного примера функциональность приложения-клиента задано в виде двух условий:

- 1) клиент **посылает** серверу текстовую строку, используя метод **getMessage(...)** удалённого объекта;
- 2) клиент выводит на печать результат обращения к методу **getMessage(...)**.

## 2.1.2 Описание интерфейса RMI-клиента

Считается, что описание удалённого интерфейса, содержащего синтаксис метода ***getMessage(...)***, соответствует исходному тексту языка Java, представленному на листинге 1.1, пункта 1.1.2, главы 1 данного пособия.

Поскольку разработчику приложения-клиента обычно доступен только исходный текст интерфейса удалённого объекта, то допустимо модифицировать оператор ***package*** для согласования размещения программного обеспечения клиента.

## 2.2 Реализация RMI-клиента

Вторая часть лабораторного задания №2 содержит описание Java-класса, который программно ассоциируется с реализацией приложения RMI-клиента.

**Основная задача RMI-клиента** — подключение ко всем нужным серверам ***rmiregistry*** с целью поиска всех нужных точек подключения к удалённым объектам и выполнения обращений к этим объектам.

В нашем случае, на уже запущенном сервере ***rmiregistry***, должен быть зарегистрирован удалённый объект заданный классом ***Klp2RmiImpl*** и интерфейсом ***Klp2Rmi***. Дополнительно, должна быть подготовлена среда реализации RMI-клиента, что уже было выполнено в предыдущем задании и подробно описано в пункте 1.2.1 данного пособия.

**Более конкретно**, выполняя эту часть своего задания, студент ориентируется на реализацию своего серверного приложения и организацию его среды разработки и запуска. Далее, - необходимо:

- 1) **написать** и правильно разместить исходные тексты всех частей **клиентской** части приложения, включая последующую их компиляцию;
- 2) **произвести** необходимое количество запусков приложения RMI-клиента, убедившись в отсутствии ошибочных сообщений программного обеспечения, и демонстрируя работоспособность всего распределенного приложения.

Далее, на учебном примере, демонстрируются все необходимые действия.

### 2.2.1 Компиляция исходного текста RMI-клиента

**Исходный текст** учебного примера приложения RMI-клиента, удовлетворяющего перечисленным выше ограничениям, представлен на листинге

2.1.

**Дополнительные ограничения** на реализацию этого текста представлены следующими условиями:

- 1) RMI-клиент реализован как класс **Klp2RmiClient**;
- 2) удалённый объект клиента доступен по ссылке: **//localhost:1099/KLP2RMI**;
- 3) передаваемая на сервер строка задаётся аргументом приложения.

*Листинг 2.1 – Исходный текст класса Klp2RmiClient*

```
package klp2.rmi;

import java.rmi.Naming;

public class Klp2RmiClient
{
    public static void main(String[] args)
    {
        /**
         * Читаем и устанавливаем значение аргумента
         */
        String str = ""; // Строка аргумента
        if(args.length > 0) str = new String(args[0]);
        System.out.println("Клиент: число аргументов " + args.length);
        /**
         * Результат обращения к серверу
         */
        String[] msgs;

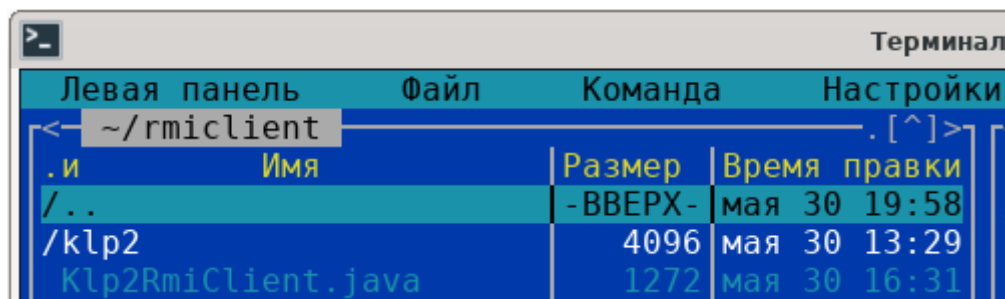
        try{
            /**
             * Получаю объектную ссылку на удаленный объект
             */
            Klp2Rmi rmiobj =
                (Klp2Rmi)Naming.lookup("//localhost:1099/KLP2RMI");

            System.out.println("Посылаю: " + str);
            /**
             * Вызываю удалённый метод
             */
            msgs =
                rmiobj.getMessage(str);

            /**
             * Печатаю результат работы приложения-клиента
             */
            System.out.println("Клиент: число сообщений сервера: " + msgs[0]);
            System.out.println("Последнее сообщение сервера: " + msgs[1]);
            System.out.println("Предпоследнее сообщение сервера: " + msgs[2] +
                "\n");
        }
        catch (Exception e)
        {
            System.out.println("Client-ERROR : " + e) ;
        }
    }
}
```

}

**Согласно условиям компиляции и запуска** компонент распределенного приложения, исходный текст листинга 2.1 должен быть помещён в каталог указанный в системной переменной **CLASSPATH**, котором также должны быть подготовлены подкаталоги, соответствующие оператору **package** (см. рисунок 2.1).



| Имя                | Размер  | Время правки |
|--------------------|---------|--------------|
| ..                 | -BBEPX- | мая 30 19:58 |
| /klp2              | 4096    | мая 30 13:29 |
| Klp2RmiClient.java | 1272    | мая 30 16:31 |

Рисунок 2.1 - Размещение исходного текста приложения RMI-клиента

Компиляция исходного текста RMI-приложения осуществляется в этом же каталоге, командой: **javac Klp2RmiClient.java**

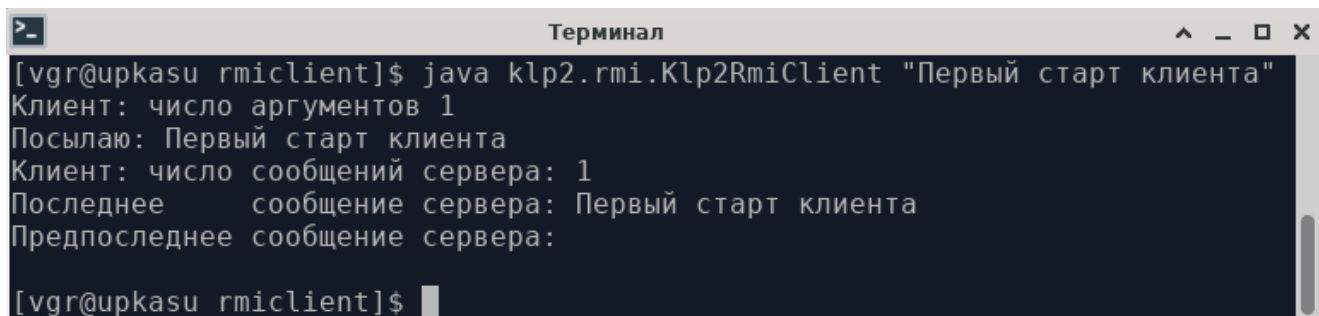
Полученный результат, в виде файла **Klp2RmiClient.class**, помещается в каталог: **~/rmiclient/klp2/rmi**.

В случае, когда RMI-сервер запускается на другой ЭВМ, необходимо дополнительно в каталог **~/rmiclient** поместить исходный текст файла-интерфейса приложения: **Klp2Rmi.java**. Тогда, сначала необходимо провести компиляцию и размещение файла-интерфейса, как это описано в пункте 1.2.3 настоящего пособия, а затем — переходить к компиляции приложения-клиента.

### 2.2.2 Запуск и тестирование работы RMI-клиента

Запуск приложения-клиента осуществляется из каталога **~/rmiclient** командой общего вида: **java klp2.rmi.Klp2RmiClient Сообщение**

Пример такого запуска приведён на рисунке 2.2.



```
[vgr@upkasu rmiclient]$ java klp2.rmi.Klp2RmiClient "Первый старт клиента"
Клиент: число аргументов 1
Посылаю: Первый старт клиента
Клиент: число сообщений сервера: 1
Последнее сообщение сервера: Первый старт клиента
Предпоследнее сообщение сервера:
[vgr@upkasu rmiclient]$
```

Рисунок 2.2 — Пример запуска приложения RMI-клиента

## 2.3 Запуск RMI-клиента в среде ОС MS Windows 10

Поскольку все необходимые переменные среды уже были настроены в предыдущем разделе, то здесь мы ограничимся следующими подсказками:

- а) исходные тексты программы клиента размещены в каталоге **C:\Java\rmiclient**, как это показано на рисунке 2.3;
- б) сценарий запуска RMI-клиента показан на рисунке 2.4.



Рисунок 2.3 — Каталог для запуска RMI-клиента

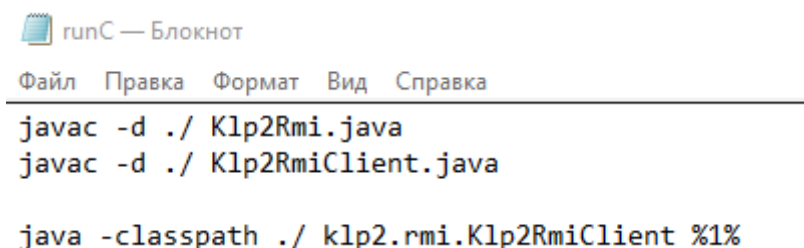


Рисунок 2.4 — Содержимое файла сценария RunC.cmd

Поскольку программа RMI-клиента запускается много раз, то, после первого запуска, команды `javac` можно закомментировать оператором `REM`.

## 2.4 Контроль выполнения лабораторных работ

### **Общее задание:**

Каждый студент должен реализовать своё собственное распределённое приложение, провести его отладку и тестирование.

По результатам проделанных лабораторных работ №1 и №2 студент должен подготовить единый письменный отчёт для оценки лабораторного практикума преподавателем. Отчёт размещается на сетевом диске СДО ФДО, в месте указанном преподавателем.

## **Список использованных источников**

- 1 Резник В.Г. Распределенные вычислительные сети : учебное пособие / В. Г. Резник. – Томск : ФДО, ТУСУР, 2020. – 254 с.