

ЛАБОРАТОРНАЯ РАБОТА №1

По дисциплине:	Распределенные информационно-аналитические системы.
Тема занятия:	Архитектура клиент-сервер как модель распределенной системы.
Цель занятия:	Изучить особенности построения клиент-серверной архитектуры, являющейся одной из моделей распределенной системы; выработать практические навыки работы с протоколами передачи данных UDP и TCP/IP.
Количество часов:	4-6.

Часть 1. Создание приложения UDP

Постановка задачи

Необходимо разработать клиент/серверное приложение, в котором сервер может распространять сообщения всем клиентам, зарегистрированным в группе 233.0.0.1, порт 1502. Пользователь сервера должен иметь возможность ввода и отправки текстовых сообщений, а пользователь-клиент просматривает полученные сообщения.

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать новый проект.
2. Реализовать класс сервера для ввода и отправки сообщений.
3. Реализовать класс клиента для получения и просмотра сообщений.
4. Протестировать приложение – запустить сервер и клиент, и отправить сообщение.

Подготовительный этап

Для реализации проекта необходимо установить и настроить интегрированную среду разработки, например, JetBrains IntelliJ IDEA. Все примеры в текущей лабораторной работе приведены с использованием указанной среды.

Создание нового проекта

1. Выберите пункт меню `File` → `New` → `Project`, в окне выбора типа проекта укажите `Java` и нажмите `Next`.
2. Укажите имя проекта `Lab1` и нажмите `Finish`.

Создание класса `Server`

Класс `Server` предназначен для отсылки сообщений всем клиентам, зарегистрированным в группе 233.0.0.1. Создание класса `Server` включает в себя следующие основные задачи:

1. Создание сокета с помощью класса `DatagramSocket`. Сокет сервера выполняет задачу отправки сообщения.
2. Создание объекта `InetAddress`, представляющего адрес сервера. Адреса для групповой (multicast) передачи сообщений выбираются из диапазона 224.0.0.0-239.255.255.255. В нашем приложении будет указан адрес 233.0.0.1.
3. Организация ввода строки сообщения с клавиатуры и создание объекта `packet` класса `DatagramPacket`, который хранит введенные данные и использует метод `send()` объекта класса `DatagramSocket`, для отсылки пакета всем клиентам группы.

Для создания класса сервера щелкните правой кнопкой мыши на каталог `src` в окне `Project` и выберите `New` → `Package`. В появившемся окне в качестве имени пакета укажите `com.company.lab1`. Затем сделайте те же операции и выберите `New` → `Java Class`, где в качестве имени класса (Name) задайте `Server`. Нажмите `OK`.

Код класса `Server` приведен ниже:

```

package com.company.lab1;

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

public class Server {

    private BufferedReader in = null;
    private String str = null;
    private byte[] buffer;
    private DatagramPacket packet;
    private InetAddress address;
    private DatagramSocket socket;

    public Server() throws IOException {
        System.out.println("Sending messages");

        // Создается объект DatagramSocket, чтобы
        // принимать запросы клиента
        socket = new DatagramSocket();

        // Вызов метода transmit(), чтобы передавать сообщение всем
        // клиентам, зарегистрированным в группе
        transmit();
    }

    public void transmit() {
        try {
            // Создается входной поток, чтобы принимать
            // данные с консоли
            in = new BufferedReader(new InputStreamReader(System.in));
            while (true) {
                System.out.println(
                    "Введите строку для передачи клиентам: ");
                str = in.readLine();
                buffer = str.getBytes();
                address = InetAddress.getByName("233.0.0.1");
                // Посылка пакета датаграмм на порт номер 1502
                packet = new DatagramPacket(
                    buffer,
                    buffer.length,
                    address,
                    1502);

                // Посылка сообщений всем клиентам в группе
                socket.send(packet);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                // Закрытие потока и сокета
                in.close();
                socket.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String arg[]) throws Exception {
        // Запуск сервера
        new Server();
    }
}

```

Создание класса Client

Класс Client позволяет клиенту присоединиться к группе 233.0.0.1 для получения сообщений от сервера. Создание класса Client включает в себя следующие основные задачи:

1. Создание сокета для просмотра групповых сообщений с помощью класса MulticastSocket. Сокет клиента выполняет задачу приема сообщения.
2. Создание объекта InetAddress, представляющего адрес сервера и присоединение к группе этого сервера с помощью метода сокета joinGroup.
3. Организация чтения пакетов датаграмм (DatagramPacket) из сокета и отображение полученных данных на экране.

Для создания класса клиента щелкните правой кнопкой мыши на пакет com.company.lab1 в каталоге src окна Project и выберите New → Java Class. В появившемся окне в качестве имени класса (Name) задайте Client. Нажмите ОК.

Код класса Client приведен ниже:

```
package com.company.lab1;

import java.net.*;

public class Client {
    private static InetAddress address;
    private static byte[] buffer;
    private static DatagramPacket packet;
    private static String str;
    private static MulticastSocket socket;

    public static void main(String arg[]) throws Exception {
        System.out.println("Ожидание сообщения от сервера");
        try {

            // Создание объекта MulticastSocket, чтобы получать
            // данные от группы, используя номер порта 1502
            socket = new MulticastSocket(1502);

            address = InetAddress.getByName("233.0.0.1");

            // Регистрация клиента в группе
            socket.joinGroup(address);
            while (true) {
                buffer = new byte[256];
                packet = new DatagramPacket(
                    buffer, buffer.length);
                // Получение данных от сервера
                socket.receive(packet);
                str = new String(packet.getData());
                System.out.println(
                    "Получено сообщение: " + str.trim());
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                // Удаление клиента из группы
                socket.leaveGroup(address);

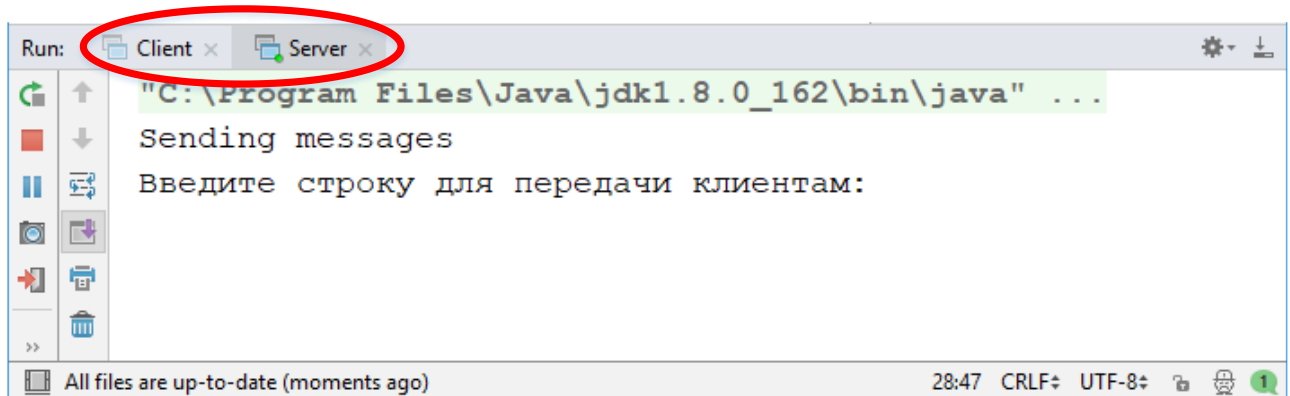
                // Закрытие сокета
                socket.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
}  
}
```

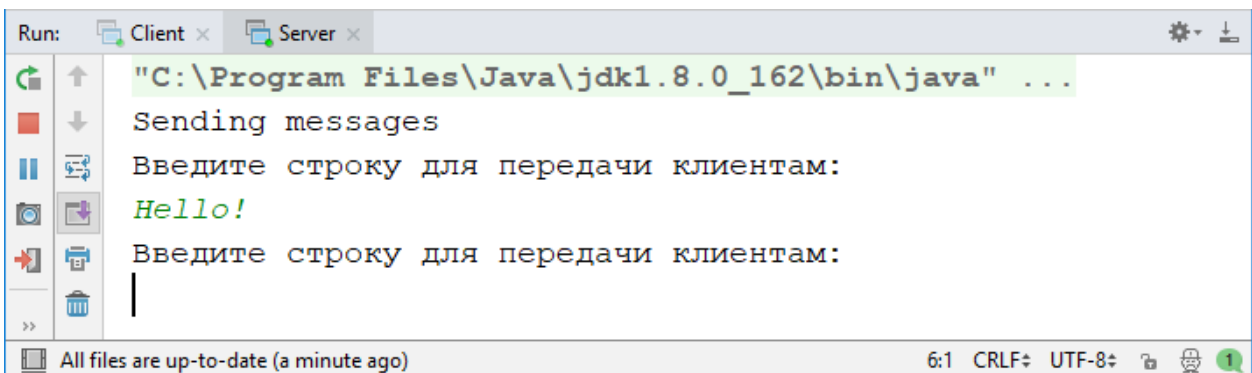
Запуск и тестирование

Каждый из построенных классов `Client` и `Server` содержит метод `main()` и является, по сути, отдельным приложением, которое может быть запущено на отдельной машине, подключенной к сети, при этом по умолчанию область видимости передачи групповых сообщений (multicasting scope) ограничивается подсетью сервера. В нашем случае роль клиента и сервера будет выполнять один и тот же компьютер.

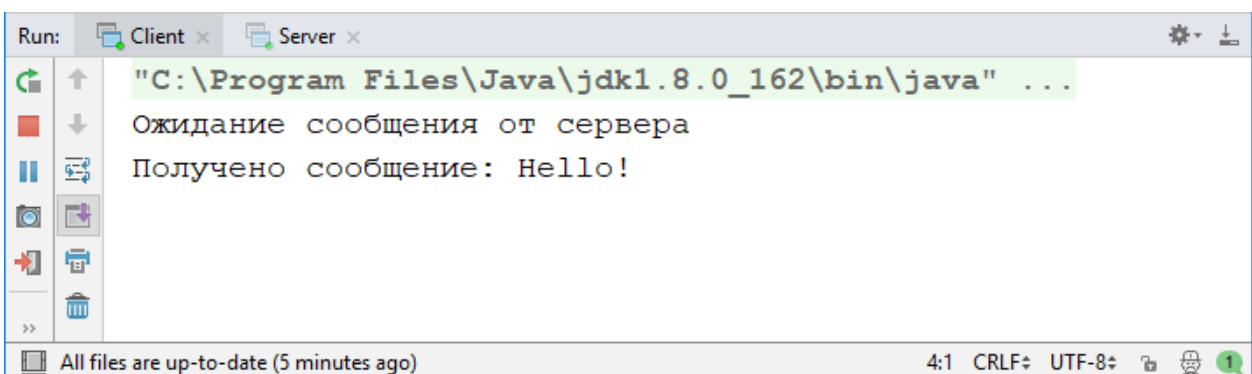
1. Щелкните правой кнопкой мыши на классе `Client` в окне `Project` и выберите команду `Run 'Client.main()'`. Прделайте то же самое с классом `Server`.
2. В результате будут запущены два приложения, переключаться между которыми можно с помощью вкладок представления `Run`:



3. Выберите консоль сервера, введите строку `"Hello!"` и нажмите `Enter` для подтверждения отправки.



4. Просмотрите консоль клиента и убедитесь, что клиент успешно принял сообщение.



5. Остановка приложения осуществляется с помощью кнопки Stop... представления Run.

Часть 2. Создание приложения TCP/IP

Постановка задачи

Необходимо разработать клиент/серверное приложение, в котором сервер слушает запросы клиентов на порт 1500 и отправляет объект-сообщение содержащий текущую дату/время сервера и строку сообщения. Пользователь-клиент должен иметь возможность просмотра полученного сообщения.

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать класс `DateMessage` с двумя полями: дата и строка – для хранения и передачи сообщения клиенту.
2. Реализовать класс сервера для прослушивания соединений на порту 1500 и отправки сообщений. Задача класса сервера должна выполняться в отдельном потоке.
3. Реализовать класс клиента для получения и просмотра сообщений
4. Протестировать приложение – запустить сервер и клиент, и проверить передачу и получение сообщения.

Подготовительный этап

Для реализации проекта необходимо установить и настроить интегрированную среду разработки, например, JetBrains IntelliJ IDEA. Все примеры в текущей лабораторной работе приведены с использованием указанной среды.

Создание класса `DateMessage`

1. Создайте новый Java-класс `DateMessage` в пакете `com.company.lab1`.
2. Вставьте следующее содержимое класса:

```
package com.company.lab1;

import java.io.Serializable;
import java.util.Date;

public class DateMessage implements Serializable {

    private Date date;
    private String message;

    public DateMessage(Date date, String message) {
        this.date = date;
        this.message = message;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

```
}  
}
```

Создание класса ServerTCP

Создание класса ServerTCP включает в себя следующие основные задачи:

1. Создание серверного сокета с помощью класса ServerSocket.
2. Ожидание запроса от клиента с помощью метода accept() серверного сокета.
3. Формирование объекта-сообщения и отправка его с помощью выходного потока клиентского сокета.

Создайте новый Java-класс ServerTCP в пакете com.company.lab1.

Код класса Server приведен ниже:

```
package com.company.lab1;  
  
import java.io.ObjectOutputStream;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.Calendar;  
  
/**  
 * Класс сервера (выполняется в отдельном процессе)  
 */  
public class ServerTCP extends Thread {  
  
    // Объявляется ссылка на объект - сокет сервера  
    ServerSocket serverSocket = null;  
  
    /**  
     * Конструктор по умолчанию  
     */  
    public ServerTCP() {  
        try {  
            // Создается объект ServerSocket, который получает  
            // запросы клиента на порт 1500  
            serverSocket = new ServerSocket(1500);  
            System.out.println("Starting the server ");  
            // Запускаем процесс  
            start();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
  
    /**  
     * Запуск процесса  
     */  
    public void run() {  
        try {  
            while (true) {  
                // Ожидание запросов соединения от клиентов  
                Socket clientSocket = serverSocket.accept();  
                System.out.println("Connection accepted from " +  
clientSocket.getInetAddress().getHostAddress());  
  
                // Получение выходного потока, связанного с объектом Socket  
                ObjectOutputStream out =  
                    new ObjectOutputStream(clientSocket.getOutputStream());  
  
                // Создание объекта для передачи клиентам  
                DateMessage dateMessage = new DateMessage(  
                    Calendar.getInstance().getTime(),  
                    "Текущая дата/время на сервере");  
            }  
        }  
    }  
}
```

```

        // Запись объекта в выходной поток
        out.writeObject(dateMessage);
        out.close();
    }
} catch (Exception e) {
    e.printStackTrace();
}
}

public static void main(String args[]) {
    // Запуск сервера
    new ServerTCP();
}
}

```

Создание класса ClientTCP

Класс ClientTCP позволяет клиенту присоединиться к серверу, используя его IP-адрес (в нашем случае localhost) и получить от него сообщение. Создание класса ClientTCP включает в себя следующие основные задачи:

1. Создание сокета для доступа к серверу localhost на порт 1500.
2. Получение входного потока сокета.
3. Чтение объекта-сообщения из потока и отображение полученных данных на экране.

Создайте новый Java-класс ClientTCP в пакете com.company.lab1.

Код класса Client приведен ниже:

```

package com.company.lab1;

import java.io.ObjectInputStream;
import java.net.Socket;

public class ClientTCP {

    public static void main(String args[]) {
        try {
            // Создается объект Socket для соединения с сервером
            Socket clientSocket = new Socket("localhost", 1500);

            // Получаем ссылку на поток, связанный с сокетом
            ObjectInputStream in =
                new ObjectInputStream(clientSocket.getInputStream());

            // Извлекаем объект из входного потока
            DateMessage dateMessage =
                (DateMessage) in.readObject();

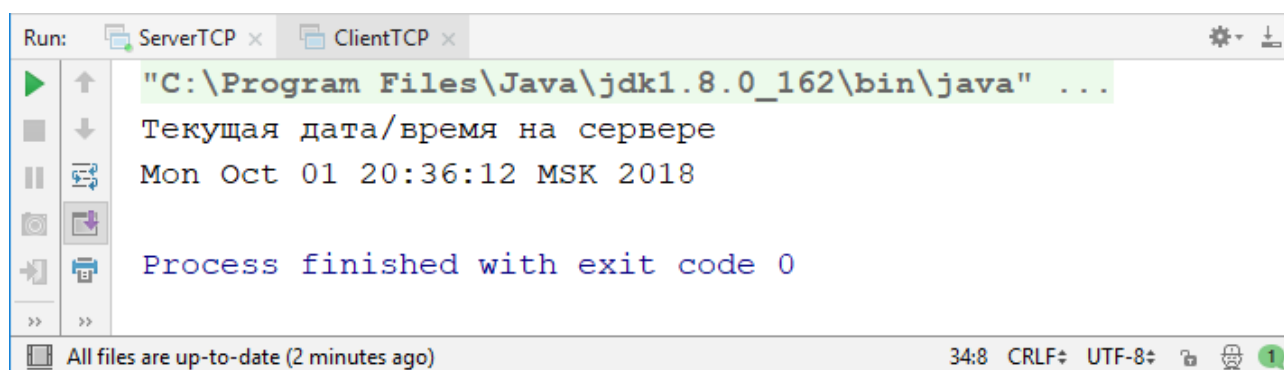
            // Выводим полученные данные в консоль
            System.out.println(dateMessage.getMessage());
            System.out.println(dateMessage.getDate());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Запуск и тестирование

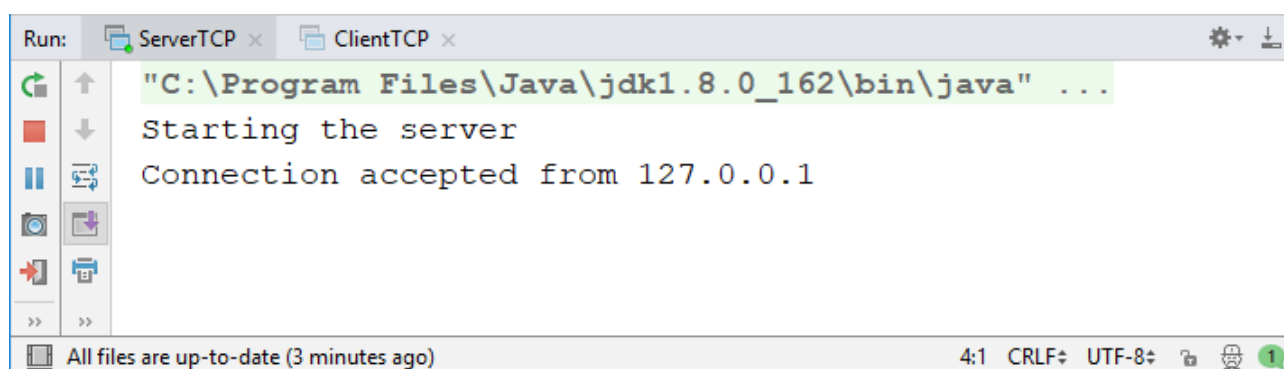
Запустите класс ServerTCP и в консоли отобразится сообщение "Starting the server".

Проделайте то же самое с классом ClientTCP. При запуске клиент пытается соединиться с сервером и обрабатывает полученное сообщение. В результате в консоли клиента выводится следующее:



```
Run: ServerTCP x ClientTCP x
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Текущая дата/время на сервере
Mon Oct 01 20:36:12 MSK 2018
Process finished with exit code 0
All files are up-to-date (2 minutes ago) 34:8 CRLF UTF-8
```

Выберите консоль сервера и просмотрите сообщение о приеме соединения от клиента:



```
Run: ServerTCP x ClientTCP x
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Starting the server
Connection accepted from 127.0.0.1
All files are up-to-date (3 minutes ago) 4:1 CRLF UTF-8
```

Самостоятельные задания:

Вариант задания выбирается согласно номеру подгруппы. Задание для всех подгрупп общее. Отличие состоит в том, что первая подгруппа использует для реализации проекта протокол передачи данных TCP, вторая – UDP.

В этом проекте вам нужно работать в парах. Вам предлагается разработать систему интернет-чата, основанную на архитектуре клиент-сервер, с использованием языка программирования Java и протокол передачи данных TCP (UDP):

- Ваша система должна позволять подключать несколько удаленных клиентов к одному центральному серверу.
- Когда пользователь вводит текстовое сообщение на своем клиенте, сообщение доставляется через сервер и отображается любым другим клиентом, который в данный момент подключен к серверу, включая первоначального отправляющего клиента.
- Пользователи могут присоединиться и выйти из чата в любое время, если сервер работает.
- При присоединении пользователи выбирают псевдонимы, которые будут отображаться вместе с их отдельными сообщениями. Адрес сервера также должен быть указан при запуске клиента.

Характеристики:

При реализации проекта необходимо соблюдать некоторые особенности:

- Когда пользователь печатает, входящие сообщения должны быть помещены в буфер, для избегания какого-либо дублирования. Сообщения будут отображены позже, как только пользователь введет свое сообщение.
- Вся система должна быть устойчива к сбоям клиента и/или заблокированным соединениям, то есть любая проблема с одним клиентом не должна затрагивать других пользователей.

Задачи:

- Напишите программные требования к системе, включая какие-либо необходимые схемы / диаграммы.
- Опишите ваш алгоритм, показывая, какие взаимодействия происходят между клиентами и сервером.
- Код на Java должен быть:
 - надежным (правильная обработка исключений и необычных случаев);
 - хорошо написан (четкая структура, содержательные имена идентификаторов и т.д.);
 - правильно прокомментирован (общая структура, назначение ваших классов и методов, параллелизм возможных потоков).
- Протестируйте свою реализацию и сообщите о возможных ошибках и/или неожиданных действиях, которые могут возникнуть.