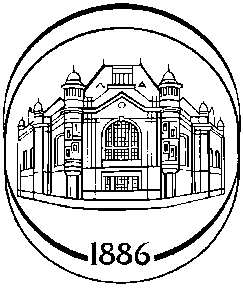
Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский Государственный Электротехнический Университет «ЛЭТИ» им. В.И. Ульянова (Ленина)»



Факультет компьютерных технологий и информатики

Кафедра автоматики и процессов управления

**ОТЧЕТ**

по лабораторной работе №3

по дисциплине «СМиСПИС»

Вариант №3

|  |  |  |
| --- | --- | --- |
| Студент гр. 5371 |  | Есенбаев Ч. |
| Студент гр. 5371 |  | Нургазы Б. |
| Студент гр. 5371 |  | Ильясов Е. |
| Преподаватель |  | Кораблев Ю.А. |

Санкт-Петербург

2020

1. **Цель работы**

Изучить методы разработки клиент-серверных приложений, научиться создавать приложение-сервер и приложение-клиент, обменивающиеся данными через стандартный интерфейс.

1. **Задание на лабораторную работу №3. Вариант №3.**

Изучить классы пакета **java.net**, которые отвечают за различные аспекты сетевого взаимодействия, технологию работы с сокетами. Разработать приложение «клиент-сервер». Сервер должен заменять квадратные скобки на круглые и записывать результат в файл.

1. **Выполнение лабораторной работы**

Для разработки клиента и сервера были использованы классы из пакета **java.net** и **java.io**.

На рис. 1 изображена UML-диаграмма разработанных классов, описание которых находится в табл. 2.

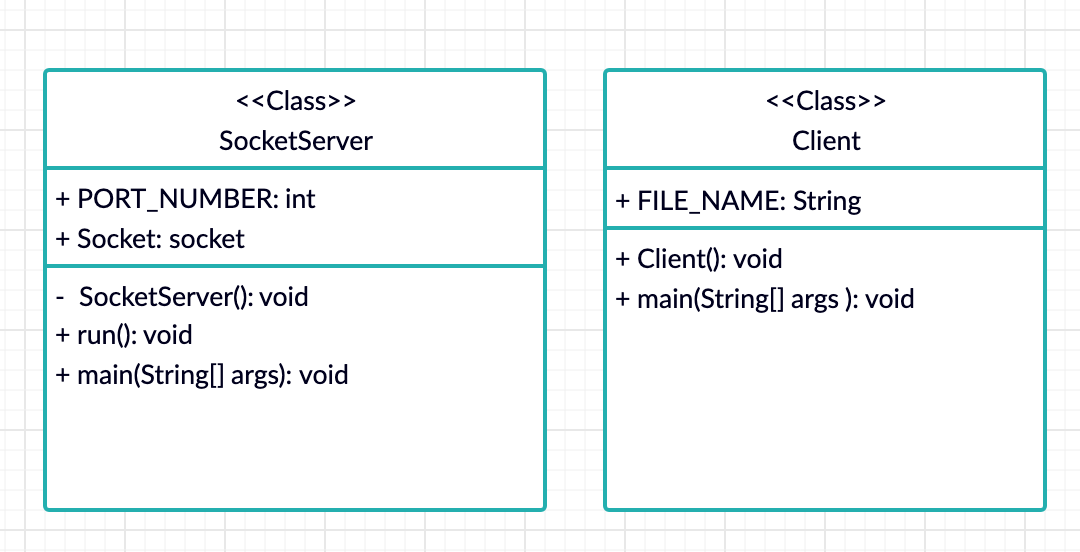


Рисунок 1. UML-диаграмма классов, разработанных в процессе выполнение лабораторной работы

Таблица 1. Описание UML-классов из рис.1

|  |  |
| --- | --- |
| **Название класса** | **Описание** |
| SocketServer | Содержит в себе функцию main() для приложения сервера. Содержит логику по старту сервера и порт, который будет прослушиваться сервером. |
| Client | Содержит в себе функцию main() для клиента. Содержит логику для старта приложения клиента. Содержит логику по отправке сообщений и записи ответов от сервера в файл. |

**Пример работы программы**

В процессе лабораторной работы были разработаны две программы – клиент *client.jar* и сервер *socketServer.jar.* Обе программы запускаются на 8083 порту. Обработанные сообщения записываются в output.txt, который создается в директории нахождения *client.jar*.

Чтобы запустить сервер приложение нужно выполнить команду:

**java -jar SocketServer.jar**

Чтобы запустить клиент приложение нужно выполнить команду:

**java -jar Client.jar**

На рис.2 представлен скриншот работы сервера.

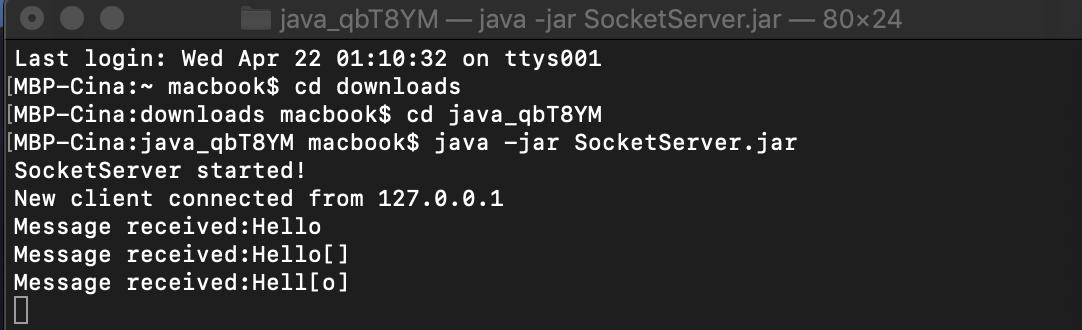


Рисунок 2. Пример работы приложения server.jar

На рис.3 представлен скриншот работы клиента.



Рисунок 3. Пример работы приложения client.jar

1. **Зачетное задание**

Формулировка: Реализуйте возможность выбора типа скобок для замены квадратных - круглые или фигурные.

**Примеры выполнения**

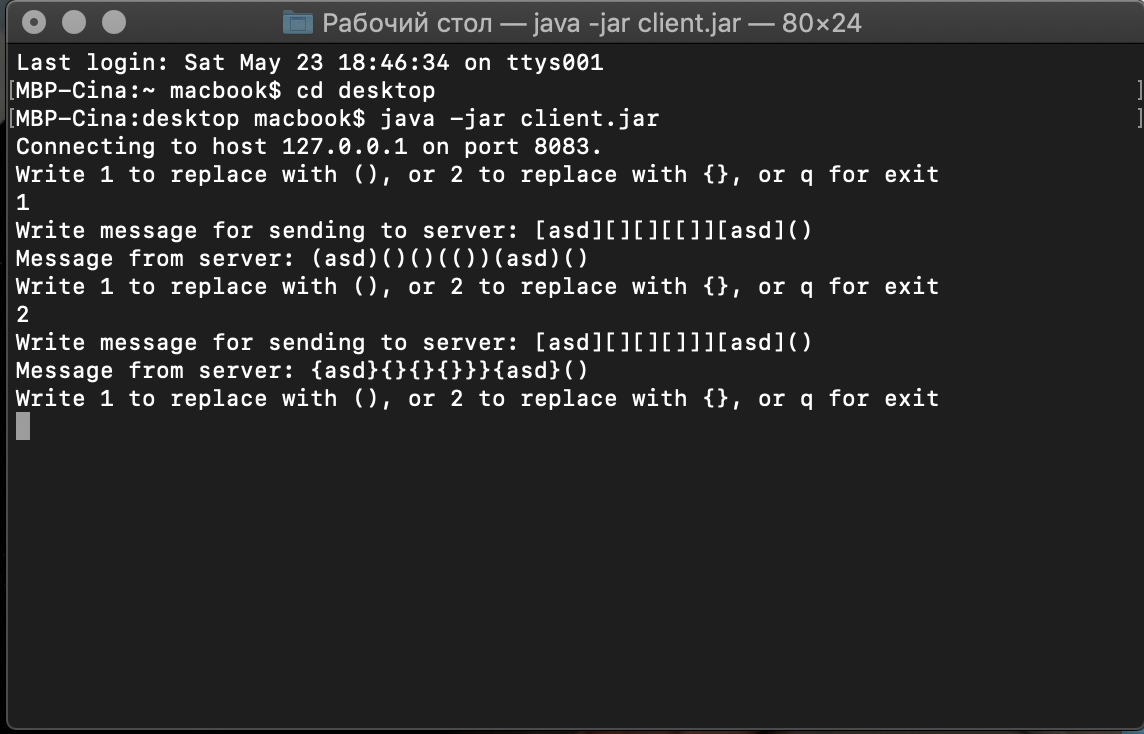
****

Рисунок 4. Пример работы client.jar

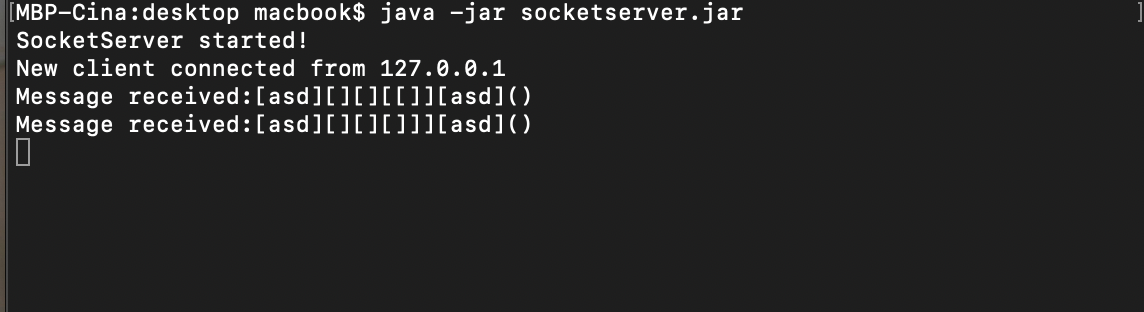


Рисунок 5. Пример работы sockerserver.jar

На рисунках 4 и 5 видна добавленная логика по выбору вида скобок для замены квадратных. Стоит заметить, что отосланные круглые скобки () не заменяются ни на что, т.к. по заданию нужно заменить только квадратные.

Дополнительный функционал реализован с помощью ключа, который отправляется на сервер и сервер принимает решение о замене квадратных на нужные скобки. Исходных код в 5 пункте обновлен.

1. **Исходный код программы**

Класс SocketServer

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

public class SocketServer extends Thread {

public static final int PORT\_NUMBER = 8083;

public Socket socket;

private SocketServer(Socket socket) {

this.socket = socket;

System.out.println("New client connected from " + socket.getInetAddress().getHostAddress());

start();

}

public void run() {

InputStream in = null;

OutputStream out = null;

String leftReplacement;

String rightReplacement;

try {

in = socket.getInputStream();

out = socket.getOutputStream();

BufferedReader br = new BufferedReader(new InputStreamReader(in));

String incomeMessage;

while ((incomeMessage = br.readLine()) != null) {

System.out.println("Message received:" + incomeMessage.substring(1));

if (incomeMessage.charAt(0) == '2') {

leftReplacement = "{";

rightReplacement = "}";

} else {

leftReplacement = "(";

rightReplacement = ")";

}

incomeMessage = incomeMessage.substring(1);

incomeMessage = incomeMessage.replaceAll("\\[", leftReplacement).replaceAll("\\]", rightReplacement)

.replaceAll("\\{", leftReplacement).replaceAll("\\}", rightReplacement);

incomeMessage += '\n';

out.write(incomeMessage.getBytes());

}

} catch (IOException ex) {

System.out.println("Unable to get streams from client");

} finally {

try {

in.close();

out.close();

socket.close();

} catch (IOException ex) {

ex.printStackTrace();

}

}

}

public static void main(String[] args) {

ServerSocket server = null;

try {

server = new ServerSocket(PORT\_NUMBER);

System.out.println("SocketServer started!");

while (true) {

/\*\*

\* create a new {@link SocketServer} object for each connection

\* this will allow multiple client connections

\*/

new SocketServer(server.accept());

}

} catch (IOException ex) {

System.out.println("Unable to start server.");

} finally {

try {

if (server != null)

server.close();

} catch (IOException ex) {

ex.printStackTrace();

}

}

}

}

Класс Client

import java.io.\*;

import java.net.Socket;

import java.net.UnknownHostException;

public class Client {

public static final String FILE\_NAME = "output";

public static void main(String args[]) {

int port = 8083;

new Client(port);

}

public Client(int port) {

try {

String serverHostname = "127.0.0.1";

System.out.println("Connecting to host " + serverHostname + " on port " + port + ".");

Socket echoSocket = null;

PrintWriter out = null;

BufferedReader in = null;

try {

echoSocket = new Socket(serverHostname, 8083);

out = new PrintWriter(echoSocket.getOutputStream(), true);

in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));

} catch (UnknownHostException e) {

System.err.println("Unknown host: " + serverHostname);

System.exit(1);

} catch (IOException e) {

System.err.println("Unable to get streams from server");

System.exit(1);

}

/\*\* {@link UnknownHost} object used to read from console \*/

BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

int messageCount = 0;

while (true) {

String ClientMessage;

String swapper;

System.out.println("Write 1 to replace with (), or 2 to replace with {}, or q for exit");

swapper = stdIn.readLine();

/\*\* Exit on 'q' char sent \*/

if ("q".equals(swapper)) {

break;

}

System.out.print("Write message for sending to server: ");

ClientMessage = stdIn.readLine();

out.println(swapper + ClientMessage);

String ServerMessage = in.readLine();

System.out.println("Message from server: " + ServerMessage);

BufferedWriter writer = new BufferedWriter(new FileWriter(FILE\_NAME, true));

writer.write("Message from client: " + ClientMessage + "\n");

writer.write("Answer from server: " + ServerMessage + "\n");

writer.close();

}

/\*\* Closing all the resources \*/

out.close();

in.close();

stdIn.close();

echoSocket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

1. **Вывод**

В процессе выполнения лабораторной работыбылиизучить методы разработки клиент-серверных приложений, создано приложение-сервер и приложение-клиент, которые обмениваются данными через стандартный интерфейс.