Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерных технологий

Лабораторная работа №6

Вариант 213071

Выполнила:

Павличенко Софья Алексеевна, P3115

Проверил:

Вербовой Александр Александрович

Санкт-Петербург 2024г.

Оглавление

Задание

Разделить программу из лабораторной работы №5 на клиентский и серверный модули. Серверный модуль должен осуществлять выполнение команд по управлению коллекцией. Клиентский модуль должен в интерактивном режиме считывать команды, передавать их для выполнения на сервер и выводить результаты выполнения.

Необходимо выполнить следующие требования:

- Операции обработки объектов коллекции должны быть реализованы с помощью Stream API с использованием лямбда-выражений.
- Объекты между клиентом и сервером должны передаваться в сериализованном виде.
- Объекты в коллекции, передаваемой клиенту, должны быть отсортированы по умолчанию
- Клиент должен корректно обрабатывать временную недоступность сервера.
- Обмен данными между клиентом и сервером должен осуществляться по протоколу UDP
- Для обмена данными на сервере необходимо использовать **датаграммы**
- Для обмена данными на клиенте необходимо использовать **сетевой канал**
- Сетевые каналы должны использоваться в неблокирующем режиме.

Обязанности серверного приложения:

- Работа с файлом, хранящим коллекцию.
- Управление коллекцией объектов.
- Назначение автоматически генерируемых полей объектов в коллекции.
- Ожидание подключений и запросов от клиента.
- Обработка полученных запросов (команд).
- Сохранение коллекции в файл при завершении работы приложения.
- Сохранение коллекции в файл при исполнении специальной команды, доступной только серверу (клиент такую команду отправить не может).

Серверное приложение должно состоять из следующих модулей (реализованных в виде одного или нескольких классов):

- Модуль приёма подключений.
- Модуль чтения запроса.
- Модуль обработки полученных команд.
- Модуль отправки ответов клиенту.

Сервер должен работать в **однопоточном** режиме.

Обязанности клиентского приложения:

- Чтение команд из консоли.
- Валидация вводимых данных.
- Сериализация введённой команды и её аргументов.
- Отправка полученной команды и её аргументов на сервер.
- Обработка ответа от сервера (вывод результата исполнения команды в консоль).
- Команду save из клиентского приложения необходимо убрать.

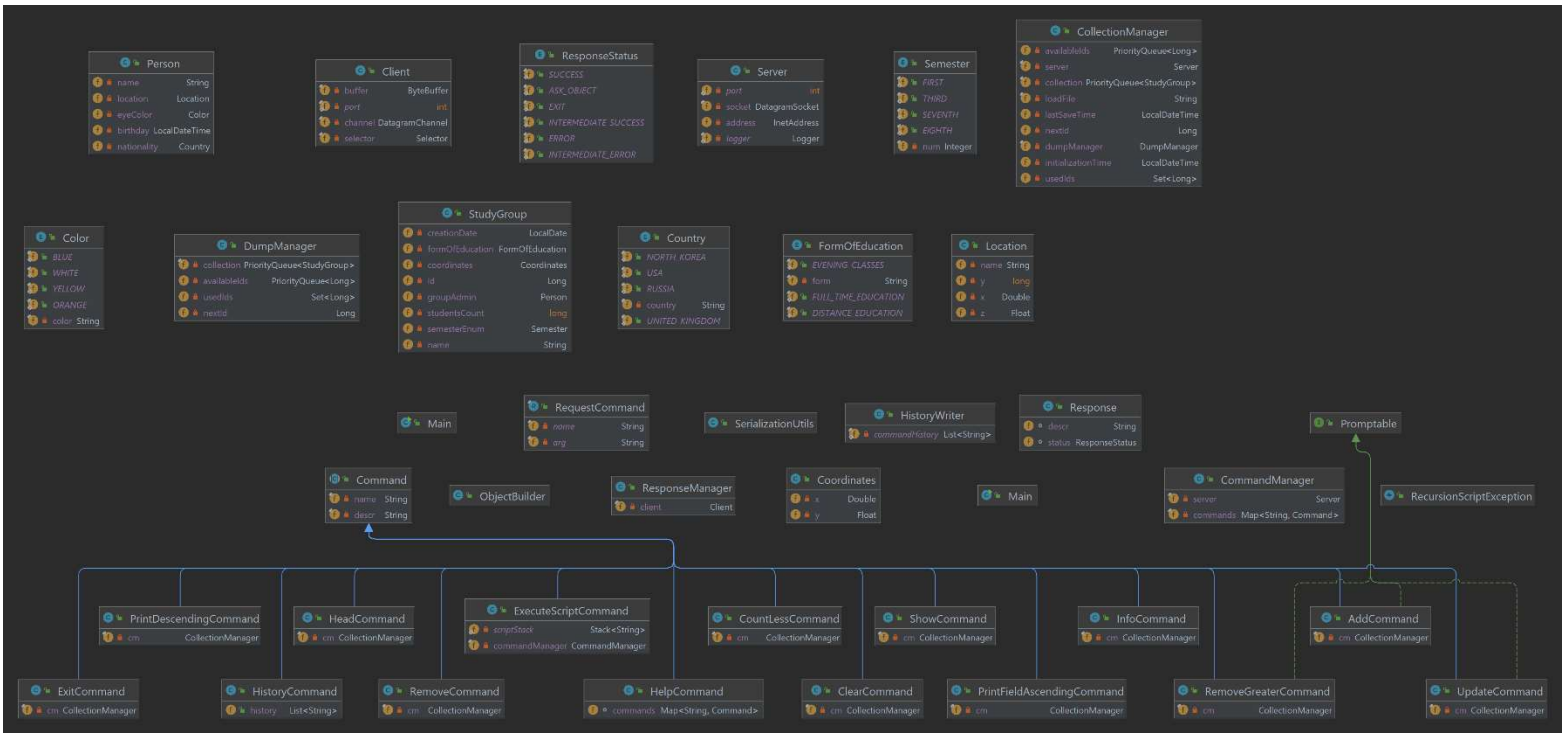
- Команда exit завершает работу клиентского приложения.

Важно! Команды и их аргументы должны представлять из себя объекты классов. Недопустим обмен "простыми" строками. Так, для команды add или её аналога необходимо сформировать объект, содержащий тип команды и объект, который должен храниться в вашей коллекции.

Дополнительное задание:

Реализовать логирование различных этапов работы сервера (начало работы, получение нового подключения, получение нового запроса, отправка ответа и т.п.) с помощью **Log4J2**

Диаграмма классов реализованной объектной модели



Решение

Исходный код программы

server.Main.java

```
package server;

import common.communication.RequestCommand;
import common.communication.Response;
import server.network.Server;
import common.models.StudyGroup;
import server.managers.*;

import java.util.PriorityQueue;

/**
 * Основной класс приложения, отвечающий за запуск серверной части программы.
 */
public class Main {
    public static void main(String[] args) {
        PriorityQueue<StudyGroup> collection = new PriorityQueue<>();
        String loadFile = null;
        if (args.length > 0) loadFile = args[0];

        Server server = new Server();

        CollectionManager collectionManager = new
CollectionManager(collection, loadFile, server);
        CommandManager commandManager = new CommandManager(collectionManager,
server);

        while (true) {
            RequestCommand requestCommand = (RequestCommand)
server.readRequest();
            Response response = commandManager.execute(requestCommand);
            server.sendResponse(response);
        }
    }
}
```

server.network.Server.java

```
package server.network;

import common.communication.*;
import common.utils.SerializationUtils;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import server.utils.HistoryWriter;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

/**
 * Серверное соединение для взаимодействия с клиентом по протоколу UDP.
 */
```

```
public class  Server {
    private static final Logger logger = LogManager.getLogger(Server.class);
    private static int port = 3130;
    private final DatagramSocket socket;
    private InetAddress address = null;

    /**
     * Конструктор класса Server, который создает серверный сокет и запускает
сервер.
     */
    public Server() {
        try {
            socket = new DatagramSocket(3130);
            logger.info("Сервер запущен...");
        } catch (SocketException e) {
            logger.error("Ошибка при запуске сервера: " + e.getMessage());
            throw new RuntimeException("Произошла ошибка при запуске сервера:
" + e);
        }
    }

    /**
     * Читает и возвращает запрос от клиента.
     */
    public Object readRequest() {
        try {
            byte[] requestData = new byte[2048];
            DatagramPacket packet = new DatagramPacket(requestData,
requestData.length);
            socket.receive(packet);
            if (!(packet.getAddress() == address && packet.getPort() ==
port)) HistoryWriter.clear();
            address = packet.getAddress();
            port = packet.getPort();
            System.out.println(address.toString() + port);

            logger.debug("Получен новый запрос от клиента");
            return SerializationUtils.deserialize(packet.getData());
        } catch (IOException e) {
            logger.error("Ошибка при получении данных от клиента: " +
e.getMessage());
            throw new RuntimeException("Произошла ошибка при получении данных
от клиента: " + e.getMessage());
        }
    }

    /**
     * Отправляет ответ клиенту.
     * @param response ответ сервера клиенту.
     */
    public void sendResponse(Response response) {
        try {
            byte[] responseData = SerializationUtils.serialize(response);
            DatagramPacket packet = new DatagramPacket(responseData,
responseData.length, address, port);
            socket.send(packet);
            logger.debug("Отправлен ответ на запрос клиента");
        } catch (IOException e) {
            logger.error("Ошибка при отправке данных клиенту: " +
e.getMessage());
            throw new RuntimeException("Произошла ошибка при отправке данных
клиенту: " + e);
        }
```

```
    }
}
```

client.Main.java

```
package client;

import client.managers.ResponseManager;
import client.network.Client;
import java.util.Scanner;

import common.communication.*;

/**
 * Основной класс приложения, отвечающий за запуск клиентской части
программы.
 */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        Client client;
        client = new Client();
        ResponseManager responseManager = new ResponseManager(client);

        while (sc.hasNext()) {
            String line = sc.nextLine().trim();
            if (!line.isEmpty()) {
                String[] tokens = line.split(" ");
                String arg = null;
                if (tokens.length > 1) arg = tokens[1];
                RequestCommand request = new RequestCommand(tokens[0], arg);
                client.sendRequest(request);
                Response response = null;
                while (response == null || (response.getStatus() !=
ResponseStatus.SUCCESS && response.getStatus() != ResponseStatus.ERROR)) {
                    response = client.readResponse();
                    responseManager.handle(response);
                }
            }
        }
        sc.close();
        client.close();
    }
}
```

client.network.Client.java

```
package client.network;

import common.communication.Response;
import common.utils.SerializationUtils;

import java.io.*;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.DatagramChannel;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.util.Iterator;
```

```java
import java.util.Set;

/**
 * Клиентское соединение для взаимодействия с сервером по протоколу UDP.
 */
public class Client {
    private final DatagramChannel channel;
    private final Selector selector;
    private final ByteBuffer buffer;
    private static final int port = 3130;

    /**
     * Конструктор класса Client, который создает клиентское соединение и
настраивает канал для передачи данных.
     */
    public Client() {
        try {
            channel = DatagramChannel.open();
            channel.configureBlocking(false);
            selector = Selector.open();
            channel.register(selector, SelectionKey.OP_WRITE);
            buffer = ByteBuffer.allocate(2048);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Произошла ошибка при настройке канала
передачи данных: " + e);
        }
    }

    /**
     * Отправляет запрос на сервер.
     * @param object объект, который необходимо отправить на сервер.
     */
    public void sendRequest(Object object) {
        byte[] sendData = SerializationUtils.serialize(object);
        buffer.clear();
        buffer.put(sendData);
        buffer.flip();

        for (int attempt = 1; attempt <= 3; attempt += 1) {
            try {
                channel.send(buffer, new InetSocketAddress("localhost",
port));

                if (selector.select(5000) != 0) {
                    channel.register(selector, SelectionKey.OP_READ);
                    return;
                }
            } catch (IOException e) {
                e.printStackTrace();
                throw new RuntimeException("Произошла ошибка при передаче
данных на сервер: " + e);
            }
        }
        throw new RuntimeException("Превышено количество попыток отправки
запроса. Сервер временно недоступен!");
    }

    /**
     * Читает и возвращает ответ от сервера.
     */
    public Response readResponse() {
        for (int attempt = 1; attempt <= 3; attempt += 1) {
            try {
```

```
                selector.select();
                Set<SelectionKey> keys = selector.selectedKeys();
                Iterator<SelectionKey> iterator = keys.iterator();

                while (iterator.hasNext()) {
                    SelectionKey key = iterator.next();
                    iterator.remove();

                    if (key.isReadable()) {
                        buffer.clear();
                        channel.receive(buffer);
                        return (Response)
SerializationUtils.deserialize(buffer.array());
                    }
                }
                Thread.sleep(5000);
            } catch (IOException e) {
                e.printStackTrace();
                throw new RuntimeException("Произошла ошибка при получении
данных с сервера: " + e);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
        throw new RuntimeException("Превышено количество попыток получения
ответа. Сервер временно недоступен!");
    }

    /**
     * Закрывает клиентское соединение.
     */
    public void close() {
        try {
            if (selector != null) selector.close();
            if (channel != null) channel.close();
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Произошла ошибка при закрытии канала
передачи данных: " + e);
        }
    }
}
```

Заключение

В результате выполнения лабораторной работы я узнала о протоколах UDP и TSP, научилась реализовывать клиент-серверную архитектуру, работать с каналами и сокетами, сериализовывать объекты и передавать их по сети.