«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет

По лабораторной работе №6

Вариант 16741

Студент:

*Ильин Н. С.*

*Р3110 поток 2.9*

Преподаватель:

*Бойко В.А.*

Санкт-Петербург, 2024 г.

Оглавление

[**Задание:** 3](#_Toc160583382)

[**Выполнение работы:** 5](#_Toc160583383)

[**Выводы**: 16](#_Toc160583384)

# **Задание:**

Разделить программу из [лабораторной работы №5](https://se.ifmo.ru/courses/programming#lab5) 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 размеру
* Клиент должен корректно обрабатывать временную недоступность сервера.
* Обмен данными между клиентом и сервером должен осуществляться по протоколу TCP
* Для обмена данными на сервере необходимо использовать **потоки ввода-вывода**
* 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**

**Отчёт по работе должен содержать:**

1. Текст задания.
2. Диаграмма классов разработанной программы (как клиентского, так и серверного приложения).
3. Исходный код программы.
4. Выводы по работе.

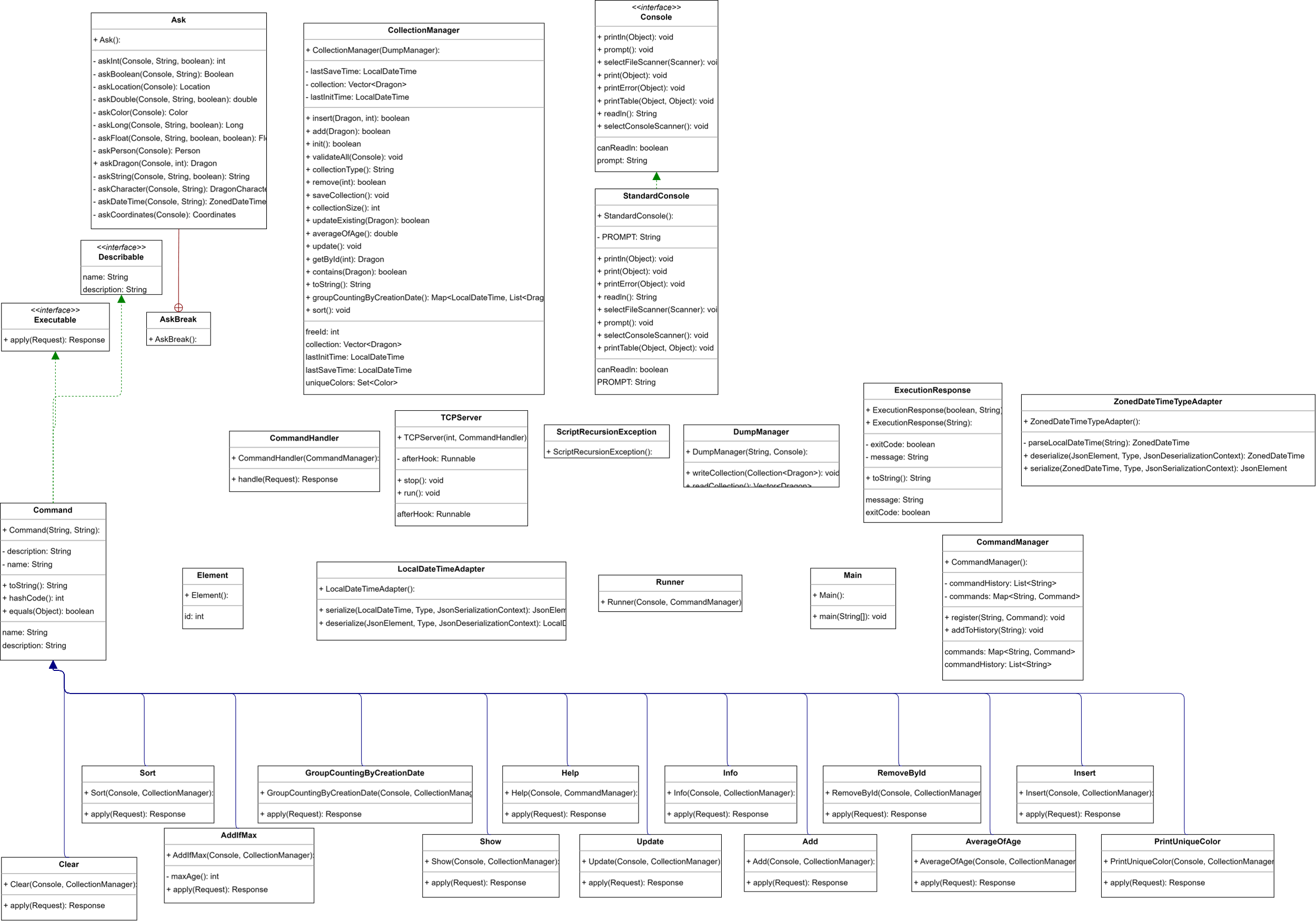
**Вопросы к защите лабораторной работы:**

1. Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия.
2. Протокол TCP. Классы Socket и ServerSocket.
3. Протокол UDP. Классы DatagramSocket и DatagramPacket.
4. Отличия блокирующего и неблокирующего ввода-вывода, их преимущества и недостатки. Работа с сетевыми каналами.
5. Классы SocketChannel и DatagramChannel.
6. Передача данных по сети. Сериализация объектов.
7. Интерфейс Serializable. Объектный граф, сериализация и десериализация полей и методов.
8. Java Stream API. Создание конвейеров. Промежуточные и терминальные операции.
9. Шаблоны проектирования: Decorator, Iterator, Factory method, Command, Flyweight, Interpreter, Singleton, Strategy, Adapter, Facade, Proxy

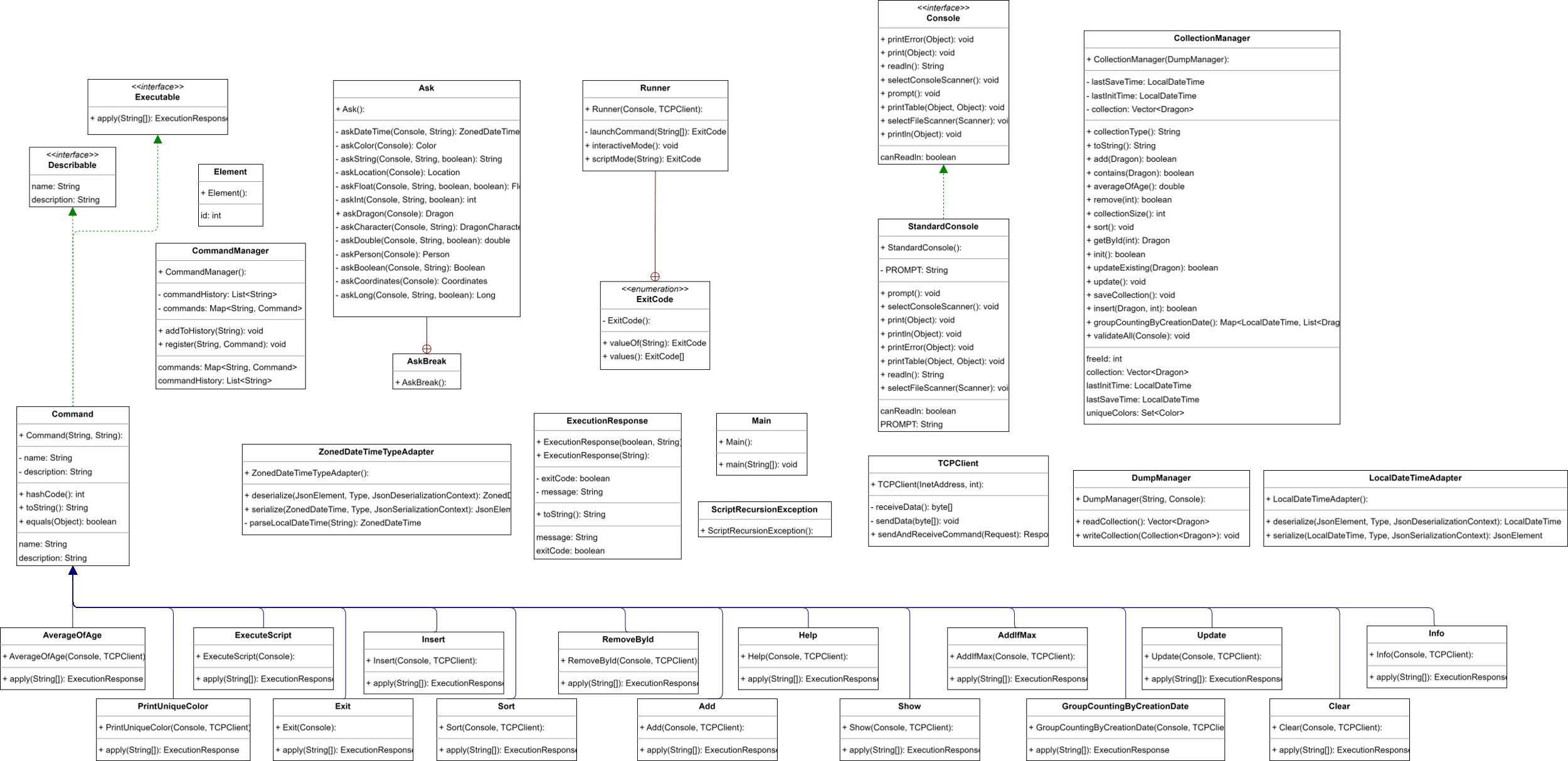
# **Выполнение работы:**

**UML-диаграмма:**

Сервер**:**



Клиент:



**Github:**  
<https://github.com/MrTheFall/study/tree/master/2sem/lab6>

**Наиболее важные части кода:**

Client Main.java

package org.lab6;

import org.lab6.commands.\*;

import org.lab6.managers.CollectionManager;

import org.lab6.managers.CommandManager;

import org.lab6.managers.DumpManager;

import org.lab6.network.TCPClient;

import org.lab6.utils.Runner;

import org.lab6.utils.console.StandardConsole;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import java.io.IOException;

import java.net.ConnectException;

import java.net.InetAddress;

public class Main {

private static final int PORT = 25565;

public static final Logger logger = LogManager.getLogger("ClientLogger");

public static void main(String[] args) {

var console = new StandardConsole();

try {

var client = new TCPClient(InetAddress.getLocalHost(), PORT);

new Runner(console, client).interactiveMode();

}catch (ConnectException e){

logger.error("Сервер недоступен");

} catch (IOException | RuntimeException e) {

logger.error(e);

}

}

}

Client TCPClient.java  
**package** org.lab6.network;

**import** org.lab6.Main;

**import** common.network.responses.Response;

**import** common.network.requests.Request;

**import** org.apache.commons.lang3.SerializationUtils;

**import** org.apache.logging.log4j.Logger;

**import** java.io.IOException;

**import** java.net.InetAddress;

**import** java.net.InetSocketAddress;

**import** java.nio.ByteBuffer;

**import** java.nio.channels.SocketChannel;

**public** **class** TCPClient {

**private** **final** SocketChannel client;

**private** **final** InetSocketAddress serverAddress;

**private** **final** Logger logger = Main.logger;

**public** TCPClient(InetAddress hostname, **int** port) **throws** IOException {

**this**.serverAddress = **new** InetSocketAddress(hostname, port);

**this**.client = SocketChannel.open();

**this**.client.connect(serverAddress);

**this**.client.finishConnect();

**this**.client.configureBlocking(**false**);

logger.info("SocketChannel opened connection to " + serverAddress + client.isConnected());

}

**public** Response sendAndReceiveCommand(Request request) **throws** IOException {

var requestData = SerializationUtils.serialize(request);

sendData(requestData);

var responseData = receiveData();

Response response = SerializationUtils.deserialize(responseData);

logger.info("Received response from server: " + response);

**return** response;

}

**private** **void** sendData(**byte**[] data) **throws** IOException {

ByteBuffer sizeBuffer = ByteBuffer.allocate(Integer.BYTES);

sizeBuffer.putInt(data.length);

sizeBuffer.flip();

client.write(sizeBuffer);

ByteBuffer dataBuffer = ByteBuffer.wrap(data);

**while** (dataBuffer.hasRemaining()) {

client.write(dataBuffer);

}

logger.info("Data sent to server.");

}

**private** **byte**[] receiveData() **throws** IOException {

ByteBuffer sizeBuffer = ByteBuffer.allocate(Integer.BYTES);

**while** (sizeBuffer.hasRemaining()) {

client.read(sizeBuffer);

}

sizeBuffer.flip();

**int** size = sizeBuffer.getInt();

ByteBuffer dataBuffer = ByteBuffer.allocate(size);

**while** (dataBuffer.hasRemaining()) {

client.read(dataBuffer);

}

**return** dataBuffer.array();

}

}

Client Runner.java

**package** org.lab6.utils;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.NoSuchElementException;

**import** java.util.Scanner;

**import** org.lab6.commands.\*;

**import** org.lab6.exceptions.ScriptRecursionException;

**import** org.lab6.managers.CommandManager;

**import** org.lab6.network.TCPClient;

**import** org.lab6.utils.console.Console;

***/\*\****

***\* Класс для выполнения команд в различных режимах.***

***\*/***

**public** **class** Runner {

**private** **final** TCPClient client;

**public** **enum** ExitCode {

OK,

ERROR,

EXIT,

}

**private** **final** Console console;

**private** **final** CommandManager commandManager;

**private** **final** List<String> scriptStack = **new** ArrayList<>();

***/\*\****

***\* Конструктор класса Runner.***

***\* @param console***

***\* @param client***

***\*/***

**public** Runner(Console console, TCPClient client) {

**this**.client = client;

**this**.console = console;

**this**.commandManager = **new** CommandManager() {{

register("help", **new** Help(console, client));

register("info", **new** Info(console, client));

register("add", **new** Add(console, client));

register("update", **new** Update(console, client));

register("remove\_by\_id", **new** RemoveById(console, client));

register("clear", **new** Clear(console, client));

register("show", **new** Show(console, client));

register("execute\_script", **new** ExecuteScript(console));

register("exit", **new** Exit(console));

register("add\_if\_max", **new** AddIfMax(console, client));

register("insert", **new** Insert(console, client));

register("sort", **new** Sort(console, client));

register("average\_of\_age", **new** AverageOfAge(console, client));

register("group\_counting\_by\_creation\_date", **new** GroupCountingByCreationDate(console, client));

register("print\_unique\_color", **new** PrintUniqueColor(console, client));

}};

}

***/\*\****

***\* Интерактивный режим***

***\*/***

**public** **void** interactiveMode() {

**try** {

ExitCode commandStatus;

String[] userCommand = {"", ""};

**do** {

console.prompt();

userCommand = (console.readln().trim() + " ").split(" ", 2);

userCommand[1] = userCommand[1].trim();

commandManager.addToHistory(userCommand[0]);

commandStatus = launchCommand(userCommand);

} **while** (commandStatus != ExitCode.EXIT);

} **catch** (NoSuchElementException exception) {

console.printError("Пользовательский ввод не обнаружен!");

} **catch** (IllegalStateException exception) {

console.printError("Непредвиденная ошибка!");

}

}

***/\*\****

***\* Режим для запуска скрипта.***

***\* @param argument Аргумент скрипта***

***\* @return Код завершения.***

***\*/***

**public** ExitCode scriptMode(String argument) {

String[] userCommand = {"", ""};

ExitCode commandStatus;

scriptStack.add(argument);

**try** (Scanner scriptScanner = **new** Scanner(**new** File(argument))) {

**if** (!scriptScanner.hasNext()) **throw** **new** NoSuchElementException();

console.selectFileScanner(scriptScanner);

**do** {

userCommand = (scriptScanner.nextLine().trim() + " ").split(" ", 2);

userCommand[1] = userCommand[1].trim();

**while** (scriptScanner.hasNextLine() && userCommand[0].isEmpty()) {

userCommand = (scriptScanner.nextLine().trim() + " ").split(" ", 2);

userCommand[1] = userCommand[1].trim();

}

console.println(console.getPrompt() + String.join(" ", userCommand));

**if** (userCommand[0].equals("execute\_script")) {

**for** (String script : scriptStack) {

**if** (userCommand[1].equals(script))

{

console.selectConsoleScanner();

**throw** **new** ScriptRecursionException();

}

}

}

commandStatus = launchCommand(userCommand);

} **while** (commandStatus == ExitCode.OK && scriptScanner.hasNextLine());

console.selectConsoleScanner();

**if** (commandStatus == ExitCode.ERROR && !(userCommand[0].equals("execute\_script") && !userCommand[1].isEmpty())) {

console.println("Проверьте скрипт на корректность введенных данных!");

}

**return** commandStatus;

} **catch** (FileNotFoundException exception) {

console.printError("Файл со скриптом не найден!");

} **catch** (NoSuchElementException exception) {

console.printError("Файл со скриптом пуст!");

} **catch** (ScriptRecursionException exception) {

console.printError("Скрипты не могут вызываться рекурсивно!");

} **catch** (IllegalStateException exception) {

console.printError("Непредвиденная ошибка!");

System.exit(0);

} **finally** {

scriptStack.remove(scriptStack.size() - 1);

}

**return** ExitCode.ERROR;

}

***/\*\****

***\* Запускает команду.***

***\* @param userCommand команда для запуска.***

***\* @return код завершения.***

***\*/***

**private** ExitCode launchCommand(String[] userCommand) {

**if** (userCommand[0].isEmpty()) **return** ExitCode.OK;

var command = commandManager.getCommands().get(userCommand[0]);

**if** (command == **null**) {

console.printError("Команда '" + userCommand[0] + "' не найдена. Наберите 'help' для справки");

**return** ExitCode.ERROR;

}

**switch** (userCommand[0]) {

**case** "exit" -> {

**if** (!commandManager.getCommands().get("exit").apply(userCommand).getExitCode()) **return** ExitCode.ERROR;

**else** **return** ExitCode.EXIT;

}

**case** "execute\_script" -> {

**if** (!commandManager.getCommands().get("execute\_script").apply(userCommand).getExitCode()) **return** ExitCode.ERROR;

**else** **return** scriptMode(userCommand[1]);

}

**default** -> { **if** (!command.apply(userCommand).getExitCode()) **return** ExitCode.ERROR; }

};

**return** ExitCode.OK;

}

}

Server Main.java

**package** org.lab6;

**import** org.apache.logging.log4j.LogManager;

**import** org.apache.logging.log4j.Logger;

**import** org.lab6.commands.\*;

**import** org.lab6.handlers.CommandHandler;

**import** org.lab6.managers.CollectionManager;

**import** org.lab6.managers.CommandManager;

**import** org.lab6.managers.DumpManager;

**import** org.lab6.managers.TCPServer;

**import** org.lab6.utils.console.StandardConsole;

**import** java.io.IOException;

**import** java.net.SocketException;

**import** java.net.UnknownHostException;

**public** **class** Main {

**public** **static** **final** **int** PORT = 25565;

**public** **static** Logger logger = LogManager.getLogger("ServerLogger");

**public** **static** **void** main(String[] args) {

var console = **new** StandardConsole();

String filename = System.getenv("FILENAME");

**if** (filename == **null** || filename.isEmpty()) {

System.out.println("Задайте имя загружаемого файла с помощью переменной среды FILENAME");

System.exit(1);

}

System.out.println(filename);

var dumpManager = **new** DumpManager(filename, console);

var collectionManager = **new** CollectionManager(dumpManager);

**if** (!collectionManager.init()) { System.exit(1); }

collectionManager.validateAll(console);

Runtime.getRuntime().addShutdownHook(**new** Thread(collectionManager::saveCollection));

var commandManager = **new** CommandManager() {{

register("help", **new** Help(console, **this**));

register("info", **new** Info(console, collectionManager));

register("add", **new** Add(console, collectionManager));

register("update", **new** Update(console, collectionManager));

register("remove\_by\_id", **new** RemoveById(console, collectionManager));

register("clear", **new** Clear(console, collectionManager));

register("show", **new** Show(console, collectionManager));

register("add\_if\_max", **new** AddIfMax(console, collectionManager));

register("insert", **new** Insert(console, collectionManager));

register("sort", **new** Sort(console, collectionManager));

register("average\_of\_age", **new** AverageOfAge(console, collectionManager));

register("group\_counting\_by\_creation\_date", **new** GroupCountingByCreationDate(console, collectionManager));

register("print\_unique\_color", **new** PrintUniqueColor(console, collectionManager));

}};

**try** {

var server = **new** TCPServer(PORT, **new** CommandHandler(commandManager));

server.setAfterHook(collectionManager::saveCollection);

server.run();

} **catch** (SocketException e) {

logger.fatal("Случилась ошибка сокета", e);

} **catch** (UnknownHostException e) {

logger.fatal("Неизвестный хост", e);

} **catch** (IOException e) {

**throw** **new** RuntimeException(e);

}

}

}

Server TCPServer.java

**package** org.lab6.managers;

**import** common.models.Dragon;

**import** common.network.requests.Request;

**import** common.network.responses.Response;

**import** common.network.responses.ShowResponse;

**import** org.apache.commons.lang3.SerializationException;

**import** org.apache.logging.log4j.Logger;

**import** org.apache.commons.lang3.SerializationUtils;

**import** org.lab6.Main;

**import** common.network.responses.NoSuchCommandResponse;

**import** org.lab6.handlers.CommandHandler;

**import** java.io.\*;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.net.SocketException;

**import** java.util.Collections;

**import** java.util.Comparator;

***/\*\****

***\* TCP обработчик запросов***

***\*/***

**public** **class** TCPServer {

**private** **final** ServerSocket serverSocket;

**private** **final** CommandHandler commandHandler;

**private** Runnable afterHook;

**private** **final** Logger logger = Main.logger; *// Ensure this logger is initialized properly*

**private** **boolean** running = **true**;

**public** TCPServer(**int** port, CommandHandler commandHandler) **throws** IOException {

**this**.serverSocket = **new** ServerSocket(port);

**this**.commandHandler = commandHandler;

}

**public** **void** run() {

logger.info("TCP Server started on port " + serverSocket.getLocalPort());

**while** (running) {

Socket clientSocket = **null**;

**try** {

clientSocket = serverSocket.accept();

logger.info("Client connected from " + clientSocket.getRemoteSocketAddress());

DataInputStream input = **new** DataInputStream(clientSocket.getInputStream());

DataOutputStream output = **new** DataOutputStream(clientSocket.getOutputStream());

*// Keep the connection alive to handle multiple requests*

**while** (running && !clientSocket.isClosed()) {

*// Read the length of the incoming message*

**int** length;

**try** {

length = input.readInt();

} **catch** (EOFException e) {

logger.info("Client disconnected gracefully.");

**break**;

} **catch** (SocketException e) {

logger.info("Client connection was reset.");

**break**;

}

*// If length is greater than 0, process the request*

**if** (length > 0) {

**byte**[] message = **new** **byte**[length];

input.readFully(message, 0, message.length);

Request request = **null**;

Response response = **null**;

*// Deserialize the request and process it*

**try** {

request = SerializationUtils.deserialize(message);

logger.info("Processing request: " + request);

response = commandHandler.handle(request);

} **catch** (SerializationException e) {

logger.error("Error while handling the request: " + e.toString(), e);

response = **new** NoSuchCommandResponse(request != **null** ? request.getName() : "unknown");

} **finally** {

**if** (afterHook != **null**) afterHook.run();

}

*// Serialize and send the response to client*

**byte**[] responseData;

**try** {

**if** (response **instanceof** ShowResponse showResponse) {

logger.error(showResponse.dragons);

Collections.sort(showResponse.dragons, **new** Comparator<Dragon>() {

@Override

**public** **int** compare(Dragon d1, Dragon d2) {

**int** size1 = SerializationUtils.serialize(d1).length;

**int** size2 = SerializationUtils.serialize(d2).length;

**return** Integer.compare(size1, size2);

}

});

}

responseData = SerializationUtils.serialize(response);

output.writeInt(responseData.length);

output.write(responseData);

output.flush();

logger.info("Response sent to client");

} **catch** (IOException e) {

logger.error("Error sending response to client: " + e.toString(), e);

}

} **else** {

logger.error("No data or incorrect data received from client.");

}

}

} **catch** (IOException e) {

logger.error("Error handling client connection: " + e.toString(), e);

} **finally** {

*// Disconnection moved to the while-loop break condition*

**if** (clientSocket != **null** && !clientSocket.isClosed()) {

**try** {

clientSocket.close();

} **catch** (IOException e) {

logger.error("Failed to close client socket: " + e.toString(), e);

}

}

}

}

*// Close server socket*

**try** {

serverSocket.close();

} **catch** (IOException e) {

logger.error("Failed to close server socket: " + e.toString(), e);

}

}

**public** **void** setAfterHook(Runnable afterHook) {

**this**.afterHook = afterHook;

}

**public** **void** stop() {

running = **false**;

**try** {

serverSocket.close();

} **catch** (IOException e) {

logger.error("Failed to close server socket when stopping: " + e.toString(), e);

}

}

}

Common Request.java

**package** common.network.requests;

**import** java.io.Serializable;

**import** java.util.Objects;

**public** **abstract** **class** Request **implements** Serializable {

**private** **final** String name;

**public** Request(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o) **return** **true**;

**if** (o == **null** || getClass() != o.getClass()) **return** **false**;

Request response = (Request) o;

**return** Objects.equals(name, response.name);

}

@Override

**public** **int** hashCode() {

**return** Objects.hash(name);

}

@Override

**public** String toString() {

**return** "Request{" +

"name='" + name + '**\'**' +

'}';

}

}

Common Response.java

**package** common.network.responses;

**import** java.io.Serializable;

**import** java.util.Objects;

**public** **abstract** **class** Response **implements** Serializable {

**private** **final** String name;

**private** **final** String error;

**public** Response(String name, String error) {

**this**.name = name;

**this**.error = error;

}

**public** String getName() {

**return** name;

}

**public** String getError() {

**return** error;

}

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o) **return** **true**;

**if** (o == **null** || getClass() != o.getClass()) **return** **false**;

Response response = (Response) o;

**return** Objects.equals(name, response.name) && Objects.equals(error, response.error);

}

@Override

**public** **int** hashCode() {

**return** Objects.hash(name, error);

}

@Override

**public** String toString() {

**return** "Response{" +

"name='" + name + '**\'**' +

", error='" + error + '**\'**' +

'}';

}

}

# **Выводы**:

В рамках данной лабораторной работы было выполнено разделение приложения на клиентский и серверный модули, что является важным шагом в изучении архитектуры клиент-сервер и разработки распределенных систем. Реализация серверного модуля, который отвечает за управление коллекцией, позволила осуществить централизованное управление данными. Клиентский модуль, в свою очередь, обеспечил отправку команд на сервер в интерактивном режиме и получение результатов обработки. Такое разделение обеспечило четкое разграничение логики работы с данными и пользовательским интерфейсом.

Сериализация объектов для передачи между клиентом и сервером представляет собой стандартный механизм обмена данными в распределенных приложениях, и его освоение было важным практическим навыком. Понимание принципов и особенностей сериализации несомненно окажется полезным в дальнейшей профессиональной деятельности.

Особое внимание было уделено устойчивости работы клиентского приложения при возникновении временной недоступности сервера, что улучшает общую надежность системы и повышает качество пользовательского опыта.

Использование Stream API и лямбда-выражений для работы с объектами коллекции показало значительно большую эффективность и лаконичность кода по сравнению с традиционными подходами. Это подчеркнуло значимость функционального программирования в современной разработке ПО.

Использование неблокирующего ввода-вывода при реализации обмена данными позволило повысить эффективность работы приложения, избегая ожидания операций ввода-вывода и тем самым используя ресурсы системы более рационально. Это не только увеличивает производительность, но и обеспечивает более высокое качество обслуживания пользователей.

Четко определенные обязанности серверного и клиентского приложений усиливают структурированность разработки и облегчают поддержку кода и его масштабирование.

Особенное значение в процессе работы приобрело логирование с помощью популярной библиотеки Log4J2, которое обеспечило детальный контроль за процессом выполнения серверного приложения и способствовало более эффективной отладке.

Это исследование позволило глубже понять принципы распределенных систем и важность четкого разделения функционала на клиентскую и серверную части приложения. Также было уделено внимание практике валидации данных и обработке исключений, что повышает надежность и безопасность приложения. Полученный опыт и знания станут основой для создания масштабируемых и устойчивых программных решений в будущем.