ФЕДЕРАЛЬОНЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗАВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ “НАЦИОНАЛЬНЫЙ ИСЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Факультет Программной Инженерии и Компьютерной Техники
Дисциплина: «Программирование»

ОТЧЁТ

по лабораторной работе №6
Вариант №59101

Выполнил:

Студент группы P3111

Дорохин Сергей Константинович

Проверил:

Бойко Владислав Алексеевич

Санкт-Петербург

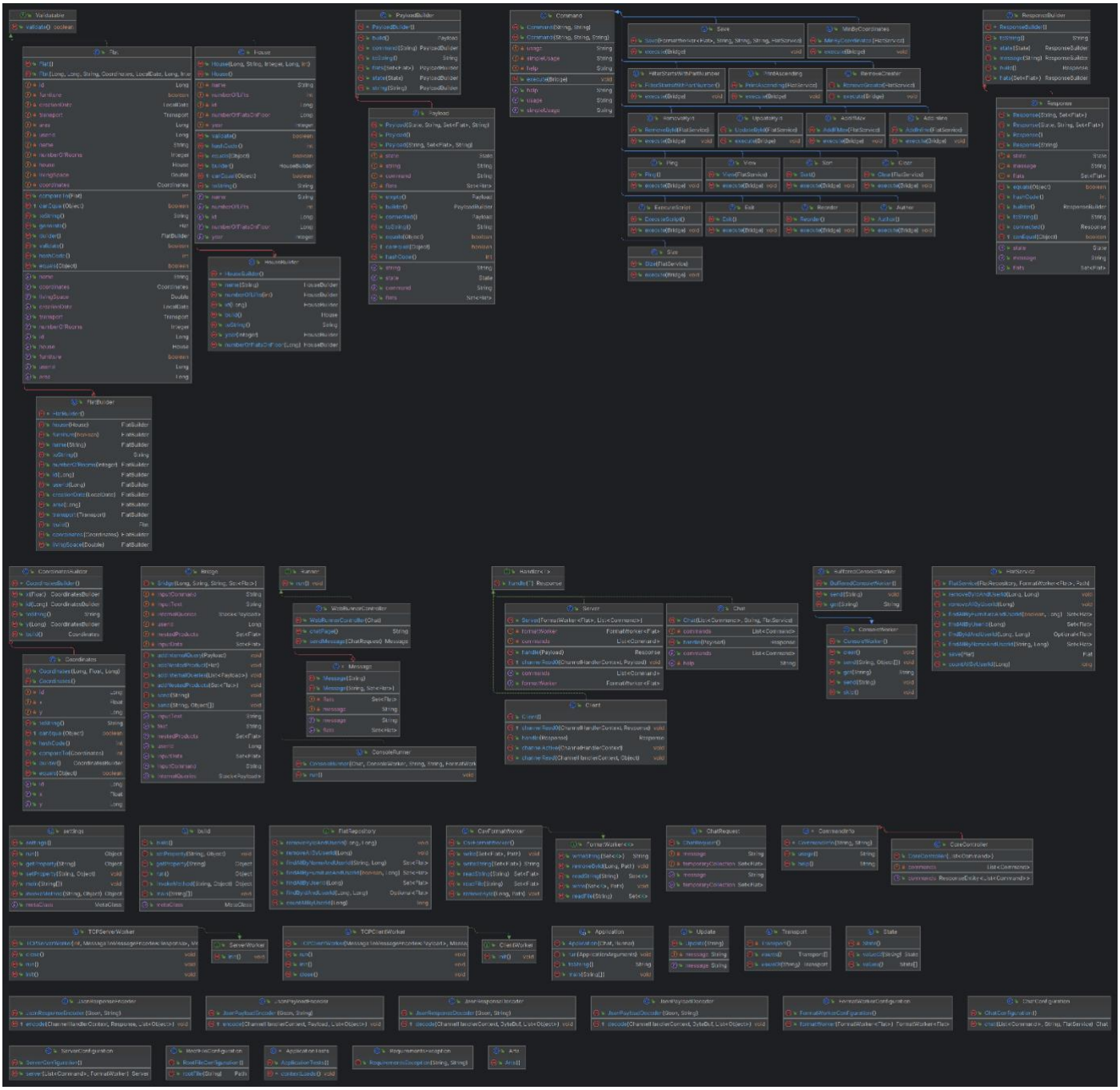
2024 г.

Задание

Вариант 59101

Код Программы

Диаграмма



Самые важные компоненты:

Server.java

```
package com.serezka.lab.lab6.server.handler;

import com.serezka.lab.core.command.Bridge;
import com.serezka.lab.core.command.Command;
import com.serezka.lab.core.database.model.User;
```

```java
import com.serezka.lab.core.handler.Handler;
import com.serezka.lab.core.io.socket.objects.Payload;
import com.serezka.lab.core.io.socket.objects.Response;
import com.serezka.lab.core.io.socket.objects.State;
import io.netty.channel.ChannelHandler;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.SimpleChannelInboundHandler;
import lombok.AccessLevel;
import lombok.Getter;
import lombok.experimental.FieldDefaults;
import lombok.extern.log4j.Log4j2;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.stream.Collectors;

@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
@Component("lab6handler")
@Log4j2(topic = "Server")
@PropertySource("classpath:chat.properties")
@ChannelHandler.Sharable
public class Server extends SimpleChannelInboundHandler<Payload> implements
Handler<Response, Payload> {
    public static final long USER_ID = -6;

    String helpPattern;

    @Getter
    List<Command> commands;

    public Server(@Qualifier("commands") List<Command> commands,
@Value("${chat.help.pattern}") String helpPattern) {
        this.commands = commands;
        this.helpPattern = helpPattern;
    }

    @Override
    protected void channelRead0(ChannelHandlerContext chx, Payload payload) {
        if (payload == null) {
            log.warn("payload can't be null!");
            return;
        }

        if (payload.getState() == null) {
            log.warn("payload's field 'state' can't be null!");
            return;
        }

        if (payload.getState() == State.CONNECTED) {
            chx.writeAndFlush(Response.connected());
            return;
        }

        log.info("new payload from client: {}", payload.toString());
        Response handledResponse = handle(payload);
        log.info("answer for client: {}", handledResponse.toString());

        chx.writeAndFlush(handledResponse);

        log.info("answer for client sent");
    }

    @Override
    public Response handle(Payload payload) {
```

```
            if (payload.getCommand() == null)
                return new Response("command can't be null!");

            if (payload.getFlats() != null)
                payload.getFlats().forEach(flat -> flat.setUserId(USER_ID));

            if (payload.getCommand().equalsIgnoreCase("help"))
                return new Response(getHelp());

            List<Command> suitableCommands = commands.stream()
                    .filter(command -> payload.getCommand().matches(command.getUsage()))
                    .toList();

            if (suitableCommands.isEmpty())
                return new Response("введена некорректная команда, help - все команды");

            if (suitableCommands.size() > 1) log.warn("suitable commands size > 1 ! {}",
    suitableCommands.toString());

            // create bridge
            Bridge commandBridge = new Bridge(USER_ID, payload.getCommand(),
    payload.getString(), payload.getFlats());
            suitableCommands.getFirst().execute(commandBridge);

            // check internal stack
            commandBridge.getInternalQueries().forEach(this::handle);

            return new Response(commandBridge.getText(),
    commandBridge.getNestedProducts());
        }

        private String getHelp() {
            return "Все доступные команды: \n" + commands.stream()
                    .map(command -> String.format("%n" + helpPattern,
    command.getSimpleUsage(), command.getHelp()))
                    .collect(Collectors.joining());
        }
    }
```

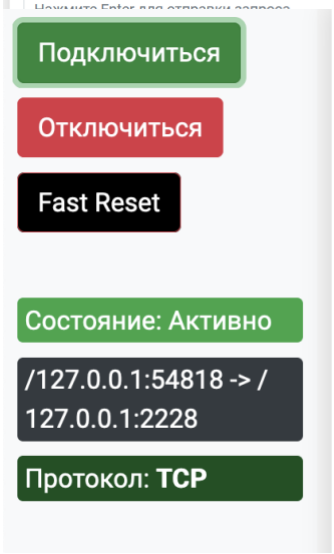
Lab6ClientHandler.java

```
package com.serezka.lab.lab6.client.handler;

import com.serezka.lab.core.command.Command;
import com.serezka.lab.core.handler.Handler;
import com.serezka.lab.core.io.socket.objects.Payload;
import com.serezka.lab.core.io.socket.objects.Response;
import com.serezka.lab.core.io.socket.objects.State;
import io.netty.channel.ChannelHandler;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.SimpleChannelInboundHandler;
import lombok.AccessLevel;
import lombok.Getter;
import lombok.NonNull;
import lombok.experimental.FieldDefaults;
import lombok.experimental.NonFinal;
import lombok.extern.log4j.Log4j2;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.List;
import java.util.Stack;

@Component("lab6client")
@PropertySource("classpath:client.properties")
```

```java
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
@Log4j2(topic = "client6")
@ChannelHandler.Sharable
public class Lab6ClientHandler extends SimpleChannelInboundHandler<Response> implements
Handler<Response, Payload> {
    @NonFinal Deque<Response> responses = new ArrayDeque<>();

    public Response getResponse() {return responses.isEmpty() ? null :
responses.pop();}

    @Getter List<Command> commands;

    @NonFinal ChannelHandlerContext context = null;

    public Lab6ClientHandler(@Qualifier("commands") List<Command> commands) {
        this.commands = commands;
    }

    @Override
    public void channelActive(@NonNull ChannelHandlerContext context) {
        this.context = context;
    }

    @Override
    protected void channelRead0(ChannelHandlerContext channelHandlerContext, Response
response) throws Exception {
        responses.add(response);
    }

    @Override
    public Response handle(Payload input) {
        context.writeAndFlush(input);
        return Response.builder().state(State.OK).message("waiting...").build();
    }
}
```

Весь остальной код: [лабораторная работа №6]

Результат работы



Вывод

Помимо интересного опыта, который я получил во время выполнения 5-ой лабораторной работы, дополнительно я познакомился с клиент-серверной разработкой и работой протокола TCP. Это был так же очень занимательный и интересный опыт, который мне пригодится в будущем.