



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	_____	Мартынов М.
Студентка гр. 5371	_____	Козлова С.
Студент гр. 5371	_____	Аверкиев В.
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.

## **2. 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 цепочки пробелов на один пробел.

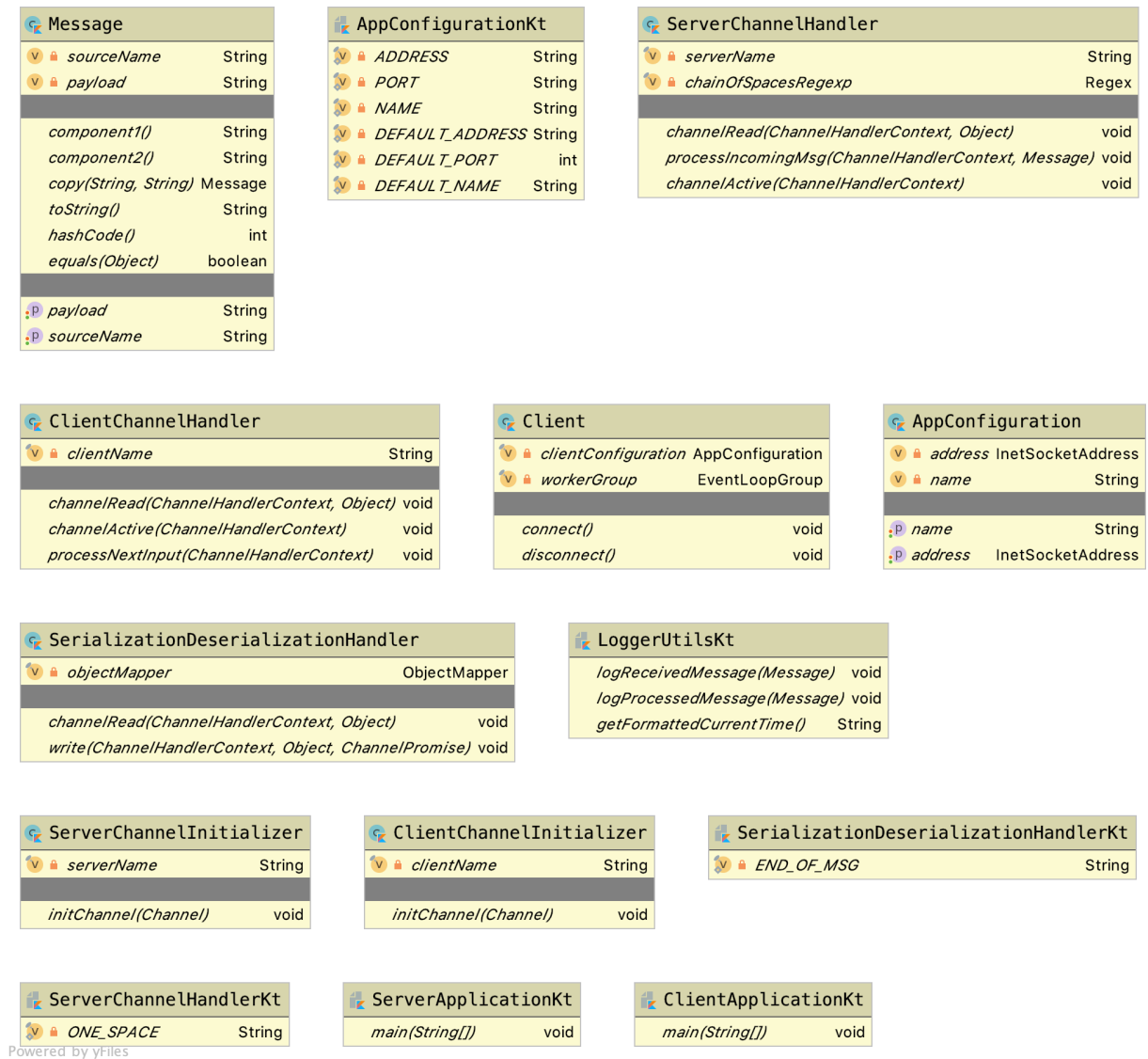
## **3. Выполнение лабораторной работы**

Для разработки клиента и сервера был выбран легковесный java-фреймворк “Netty” и язык JVM “Kotlin”.

Netty-это платформа NIO клиент сервер фреймворк, которая позволяет быстро и легко разрабатывать сетевые приложения, такие как серверы протоколов и клиенты. Это значительно упрощает и упрощает сетевое программирование, такое как TCP и UDP сокет-сервер.

"Быстро и легко" не означает, что результирующее приложение будет страдать от проблем с ремонтопригодностью или производительностью. Netty была тщательно разработана с учетом опыта, полученного в результате реализации множества протоколов, таких как FTP, SMTP, HTTP, а также различных бинарных и текстовых устаревших протоколов. В результате Netty удалось найти способ добиться простоты разработки, производительности, стабильности и гибкости без компромиссов.

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



Powered by yFiles

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

Таблица 1. Описание UML-классов из рис.1

Название класса	Описание
AppConfiguration	Класс, содержащий в себе информацию о конфигурации приложения. Хранит адрес, порт и название приложения.
Message	DTO (Data Transfer Object) класс, содержащий в себе имя приложения,

	отправляющего сообщения и непосредственно само сообщение. Такими сообщениями общается клиент и сервер.
ServerApplicationKt	Содержит функцию main() для приложения сервера.
ClientApplicationKt	Содержит функцию main() для приложения клиента.
Client	Содержит логику по старту клиента.
Server	Содержит логику по старту сервера.
SerializationDeserializationHandler	Сериализует класс Message в строку для передачи по TCP IP соединению и десериализует при получении строки. Сериализация и десериализация происходит с использованием библиотеки Jackson.
ServerChannelInitializer	Инициализирует Netty-pipeline обработчиками сообщений для сервера.
ClientChannelInitializer	Инициализирует Netty-pipeline обработчиками сообщений для клиента.
ClientChannelHandler	Обрабатывает сообщений с консоли, введенные пользователем, посылает их на сервер и получает ответ.
ServerChannelHandler	Обрабатывает сообщения от клиентов и отправляет им ответ.
LoggerUtils	Выводит сообщения на консоль

## Пример работы программы

В процессе лабораторной работы были разработаны две программы – клиент *client.jar* и сервер *server.jar*. Обе программы запускаются с ключами:

- *-name* – имя приложения. Дефолтное значение “unknown”.
- *-address* – IP-адресс. Для приложения клиента этот параметр задает IP-адресс сервера, для приложения сервера этот параметр задает локальный IP-адресс сервера. Дефолтное значение – “localhost”.
- *-port* – порт. Для приложения клиента этот параметр задает порт для подключения к серверу, для приложения сервера этот параметр задает порт, на котором сервер будет слушать входящие соединения. Дефолтное значение – “0”.

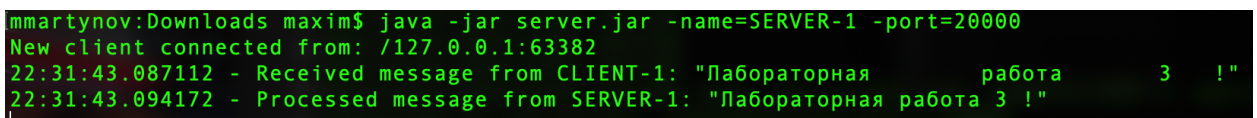
Чтобы запустить сервер приложение нужно выполнить команду:

```
java -jar server.jar -name=SERVER-1 -address=localhost -port=20000
```

Чтобы запустить клиент приложение нужно выполнить команду:

```
java -jar client.jar -name=CLIENT-1 -address=localhost -port=20000
```

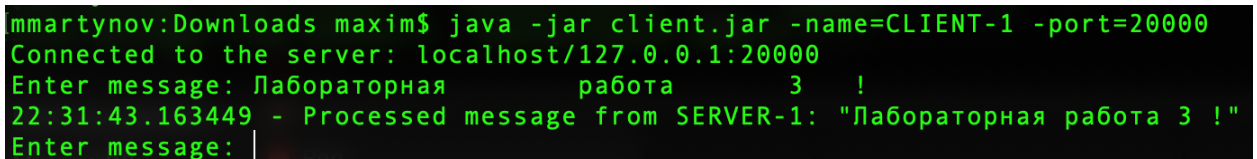
На рис.2 представлен скриншот работы сервера.



```
mmartynov:Downloads maxim$ java -jar server.jar -name=SERVER-1 -port=20000
New client connected from: /127.0.0.1:63382
22:31:43.087112 - Received message from CLIENT-1: "Лабораторная работа 3 !"
22:31:43.094172 - Processed message from SERVER-1: "Лабораторная работа 3 !"
```

Рисунок 2. Пример работы приложения server.jar

На рис.3 представлен скриншот работы клиента.



```
mmartynov:Downloads maxim$ java -jar client.jar -name=CLIENT-1 -port=20000
Connected to the server: localhost/127.0.0.1:20000
Enter message: Лабораторная работа 3 !
22:31:43.163449 - Processed message from SERVER-1: "Лабораторная работа 3 !"
Enter message: |
```

Рисунок 3. Пример работы приложения client.jar

## 4. Исходный код программы

Класс Message

```
package com.github.sindicat.lab3.dto

data class Message(
    var sourceName: String = "",
    var payload: String = ""
)
```

Класс AppConfiguration

```
package com.github.sindicat.lab3.config

import java.net.InetSocketAddress

private const val ADDRESS = "-address"
private const val PORT = "-port"
private const val NAME = "-name"
private const val DEFAULT_ADDRESS = "localhost"
private const val DEFAULT_PORT = 0
private const val DEFAULT_NAME = "unknown"

class AppConfiguration(commandLineArgs: Array<String>) {

    var address: InetSocketAddress? = null

    var name: String

    init {
        val argumentValueByKey: Map<String, String> = commandLineArgs
            .map {
                val splitedValue = it.split("=")
                splitedValue[0] to splitedValue[1]
            }.toMap()
        val host = (argumentValueByKey[ADDRESS] ?: DEFAULT_ADDRESS)
        val port = (argumentValueByKey[PORT]?.toInt() ?: DEFAULT_PORT)
        name = argumentValueByKey[NAME] ?: DEFAULT_NAME
        address = InetSocketAddress(host, port)
    }
}
```

## Класс SerializationDeserializationHandler

```
package com.github.sindicat.lab3.handler

import com.fasterxml.jackson.databind.ObjectMapper
import com.fasterxml.jackson.module.kotlin.KotlinModule
import com.github.sindicat.lab3.dto.Message
import io.netty.channel.ChannelDuplexHandler
import io.netty.channel.ChannelHandlerContext
import io.netty.channel.ChannelPromise

private const val END_OF_MSG = "\n"

class SerializationDeserializationHandler : ChannelDuplexHandler() {

    private val objectMapper: ObjectMapper =
        ObjectMapper().registerModule(KotlinModule())

    override fun channelRead(ctx: ChannelHandlerContext, msg: Any) {
        ctx.fireChannelRead(objectMapper.readValue(msg as String,
            Message::class.java))
    }

    override fun write(ctx: ChannelHandlerContext, msg: Any, promise:
        ChannelPromise?) {
        super.write(ctx, objectMapper.writeValueAsString(msg) +
            END_OF_MSG, promise)
    }
}
```

## Класс ClientChannelInitializer

```
package com.github.sindicat.lab3.client.handler.init
import com.github.sindicat.lab3.client.handler.ClientChannelHandler
import
com.github.sindicat.lab3.handler.SerializationDeserializationHandler
import io.netty.channel.Channel
import io.netty.channel.ChannelInitializer
import io.netty.handler.codec.LineBasedFrameDecoder
import io.netty.handler.codec.string.StringDecoder
import io.netty.handler.codec.string.StringEncoder
import io.netty.util.CharsetUtil

class ClientChannelInitializer(private val clientName: String) :
    ChannelInitializer<Channel>() {
    override fun initChannel(ch: Channel) {
        ch.pipeline()
            .addLast("frameDecoder", LineBasedFrameDecoder(100))
            .addLast("stringDecoder",
                StringDecoder(CharsetUtil.UTF_8))
            .addLast("stringEncoder",
                StringEncoder(CharsetUtil.UTF_8))
            .addLast("serializationDeserializationHandler",
                SerializationDeserializationHandler())
            .addLast("clientHandler",
                ClientChannelHandler(clientName))
    }
}
```

### Класс ClientChannelHandler

```
package com.github.sindicat.lab3.client.handler

import com.github.sindicat.lab3.dto.Message
import com.github.sindicat.lab3.utils.logProcessedMessage
import io.netty.channel.ChannelDuplexHandler
import io.netty.channel.ChannelHandlerContext

class ClientChannelHandler(private val clientName: String) :
    ChannelDuplexHandler() {

    override fun channelRead(ctx: ChannelHandlerContext, msg: Any) {
        val processedMsg = msg as Message
        logProcessedMessage(processedMsg)
        processNextInput(ctx)
    }

    override fun channelActive(ctx: ChannelHandlerContext?) {
        val remoteAddress = ctx!!.channel().remoteAddress()
        println("Connected to the server: $remoteAddress")
        processNextInput(ctx)
        super.channelActive(ctx)
    }

    fun processNextInput(ctx: ChannelHandlerContext) {
        print("Enter message: ")
        val inputMsg = readLine() ?: ""
        ctx.writeAndFlush(Message(clientName, inputMsg))
    }
}
```



### Класс Client

```
package com.github.sindicat.lab3.client

import com.github.sindicat.lab3.client.handler.init.ClientChannelInitializer
import com.github.sindicat.lab3.config.AppConfiguration
import io.netty.bootstrap.Bootstrap
import io.netty.channel.ChannelOption
import io.netty.channel.EventLoopGroup
import io.netty.channel.nio.NioEventLoopGroup
import io.netty.channel.socket.nio.NioSocketChannel

class Client(private val clientConfiguration: AppConfiguration) {

    private val workerGroup: EventLoopGroup = NioEventLoopGroup(1)

    fun connect() {
        try {
            val clientBootstrap: Bootstrap = Bootstrap()
                .group(workerGroup)
                .channel(NioSocketChannel::class.java)

            .handler(ClientChannelInitializer(clientConfiguration.name))
            val future = clientBootstrap
                .connect(clientConfiguration.address)
                .sync()
            future.channel().closeFuture().sync()
        } finally {
            disconnect()
        }
    }

    fun disconnect() {
        workerGroup.shutdownGracefully()
    }
}
```

### Класс Server

```
package com.github.sindicat.lab3.client

import com.github.sindicat.lab3.config.AppConfiguration

fun main(commandLineArgs: Array<String>) {
    val client = Client(AppConfiguration(commandLineArgs))
    Runtime.getRuntime().addShutdownHook(object : Thread() {
        override fun run() {
            client.disconnect()
        }
    })
    client.connect()
}
```

## Класс ServerChannelInitialier

```
package com.github.sindicat.lab3.server.handler.init

import
com.github.sindicat.lab3.handler.SerializationDeserializationHandler
import com.github.sindicat.lab3.server.handler.ServerChannelHandler
import io.netty.channel.Channel
import io.netty.channel.ChannelInitializer
import io.netty.handler.codec.LineBasedFrameDecoder
import io.netty.handler.codec.string.StringDecoder
import io.netty.handler.codec.string.StringEncoder
import io.netty.util.CharsetUtil

class ServerChannelInitializer(private val serverName: String) :
ChannelInitializer<Channel>() {

    override fun initChannel(ch: Channel) {
        ch.pipeline()
            .addLast("frameDecoder", LineBasedFrameDecoder(100))
            .addLast("stringDecoder",
StringDecoder(CharsetUtil.UTF_8))
            .addLast("stringEncoder",
StringEncoder(CharsetUtil.UTF_8))
            .addLast("serializationDeserializationHandler",
SerializationDeserializationHandler())
            .addLast("serverHandler",
ServerChannelHandler(serverName))
    }
}
```

## Класс ServerChannelHandler

```
package com.github.sindicat.lab3.server.handler

import com.github.sindicat.lab3.dto.Message
import com.github.sindicat.lab3.utils.logProcessedMessage
import com.github.sindicat.lab3.utils.logReceivedMessage
import io.netty.channel.ChannelDuplexHandler
import io.netty.channel.ChannelHandlerContext

private const val ONE_SPACE = " "

class ServerChannelHandler(private val serverName: String) :
ChannelDuplexHandler() {

    private val chainOfSpacesRegex = Regex("\\s+")

    override fun channelRead(ctx: ChannelHandlerContext, msg: Any?) {
        processIncomingMsg(ctx, msg as Message)
    }

    private fun processIncomingMsg(ctx: ChannelHandlerContext, msg:
Message) {
        logReceivedMessage(msg)
        val responsePayload: String =
msg.payload.replace(chainOfSpacesRegex, ONE_SPACE)
        val responseMsg = Message(sourceName = serverName, payload =
responsePayload)
        logProcessedMessage(responseMsg)
        ctx.writeAndFlush(responseMsg)
    }

    override fun channelActive(ctx: ChannelHandlerContext?) {
        val remoteAddress = ctx!!.channel().remoteAddress()
        println("New client connected from: $remoteAddress")
        super.channelActive(ctx)
    }
}
```

## Класс Server

```
import com.github.sindicat.lab3.config.AppConfiguration
import com.github.sindicat.lab3.server.handler.init.ServerChannelInitializer
import io.netty.bootstrap.ServerBootstrap
import io.netty.channel.ChannelOption
import io.netty.channel.EventLoopGroup
import io.netty.channel.nio.NioEventLoopGroup
import io.netty.channel.socket.nio.NioServerSocketChannel
import io.netty.util.concurrent.DefaultThreadFactory

class Server(private val serverConfiguration: AppConfiguration) {

    private val bossGroup: EventLoopGroup = NioEventLoopGroup(1,
        DefaultThreadFactory("Netty-Boss-Pool"))

    private val workerGroup: EventLoopGroup = NioEventLoopGroup(1,
        DefaultThreadFactory("Netty-Worker-Pool"))

    fun start() {
        try {
            val serverBootstrap = ServerBootstrap()
                .group(bossGroup, workerGroup)
                .channel(NioServerSocketChannel::class.java)

            .childHandler(ServerChannelInitializer(serverConfiguration.name))
                .localAddress(serverConfiguration.address)
            val future = serverBootstrap
                .bind()
                .sync()
            future.channel().closeFuture().sync()
        } finally {
            stop()
        }
    }

    fun stop() {
        workerGroup.shutdownGracefully()
        bossGroup.shutdownGracefully()
    }
}
```

## Класс Server

```
package com.github.sindicat.lab3.server

import Server
import com.github.sindicat.lab3.config.AppConfiguration

fun main(commandLineArgs: Array<String>) {
    val server = Server(AppConfiguration(commandLineArgs))
    Runtime.getRuntime().addShutdownHook( object : Thread() {
        override fun run() {
            server.stop()
        }
    })
    server.start()
}
```

## Файл LoggerUtils.kt

```
package com.github.sindicat.lab3.utils

import com.github.sindicat.lab3.dto.Message
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter

fun logReceivedMessage(msg: Message) {
    println("${getFormattedCurrentTime()} - Received message from  
${msg.sourceName}: \"${msg.payload}\"")
}

fun logProcessedMessage(msg: Message) {
    println("${getFormattedCurrentTime()} - Processed message from  
${msg.sourceName}: \"${msg.payload}\"")
}

private fun getFormattedCurrentTime() =
    LocalDateTime.now().format(DateTimeFormatter.ISO_LOCAL_TIME)
```

## **5. Вывод**

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