Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

Курсовая работа
По дисциплине разработка кроссплатформенных приложения
На тему: «Чат»

Выполнили:
Ст. гр. ПМИ-118
Зиняков Е.М.
Ванцын Д.А.
Принял:
Ст. преподаватель
Воронова Н.М.

Владимир 2021 г.

Оглавление

Постановка задачи

Разработать кроссплатформенное клиент серверное приложение в среде intellij idea на языке программирования Java – «Чат». С использованием библиотека для создания графического интерфейса (SWING).

Приложение реализовывает общий чат для одной группы людей. У пользователей изначально есть готовые логины и пароли. Логин и пароль должен храниться на сервере в отдельном файле, системный администратор должен выдать каждому пользователю личный логин и пароль, по которому пользователь сможет зайти. Так же пользователь может отправлять как общедоступные сообщения, так и приватные (доступные только двоим)

Функционал приложения

Клиет:

1. Получить список подключённых пользователей.
2. Подключиться к серверу.
   2.1. Задать port
   2.2. Задать login
   2.3. Задать password
   2.4. Пройти проверку авторизации
3. Отключится от сервера
4. Отправить сообщение
   4.1. Выбрать получателя
      4.1.1. Всем
      4.1.2. Личное (т.е. выбрать кому отправить сообщение)
   4.2. Задать текст сообщения
   4.3. Отправить сообщение
5. Просмотреть историю сообщений чата.

Сервер:

1. Включить сервер.
   a. Задать port серверу или оставить стандартный
2. Выключить сервер.

Распределение задач

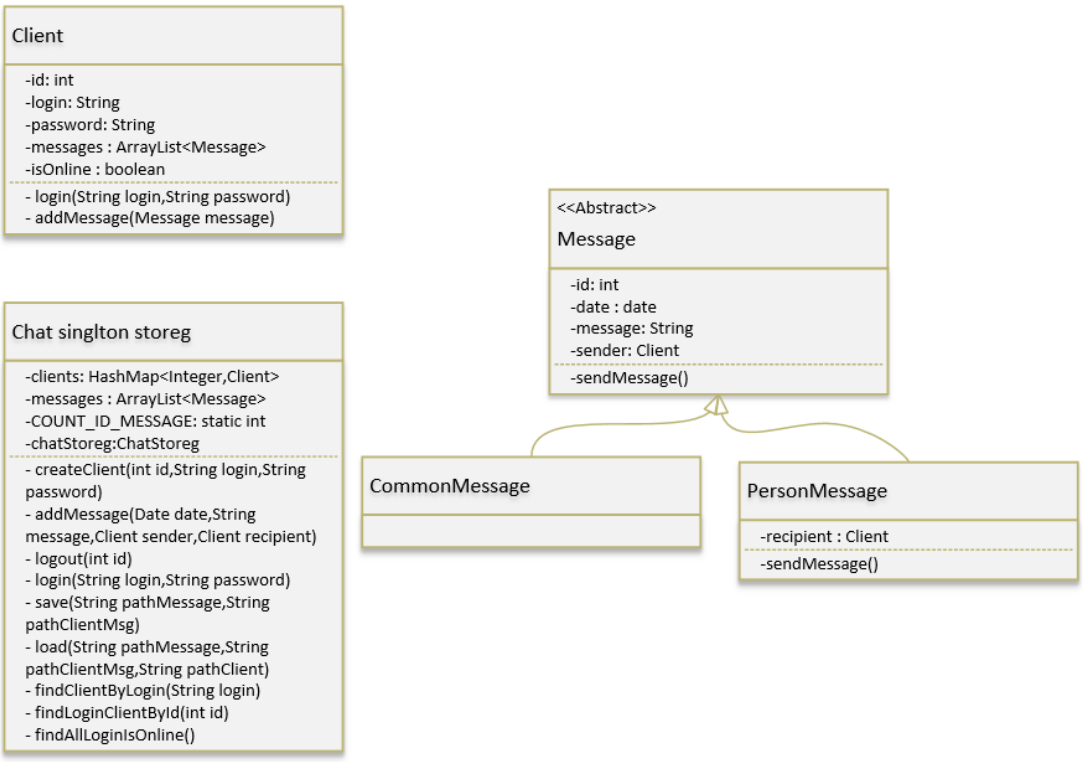
Зиняков Ефим:

1. Серверная часть приложение.
   1.1. Разработка бизнес логики серверной части
2. Клиент-Серверное взаимодействие на клиенте и сервере.
3. Разработка JavaDoc.
4. Разработка отчета

Ванцын Дмитрий:

1. Клиентская часть приложения.
   1.1. Разработка бизнес логики клиентской части
2. Разработка пользовательского интерфейса на стороне клиента и сервера
   2.1.Разработка дизайна программы
   2.2.Разработка обработчика форм
3. Разработка JavaDoc.
4. Разработка отчета

Объектная модель



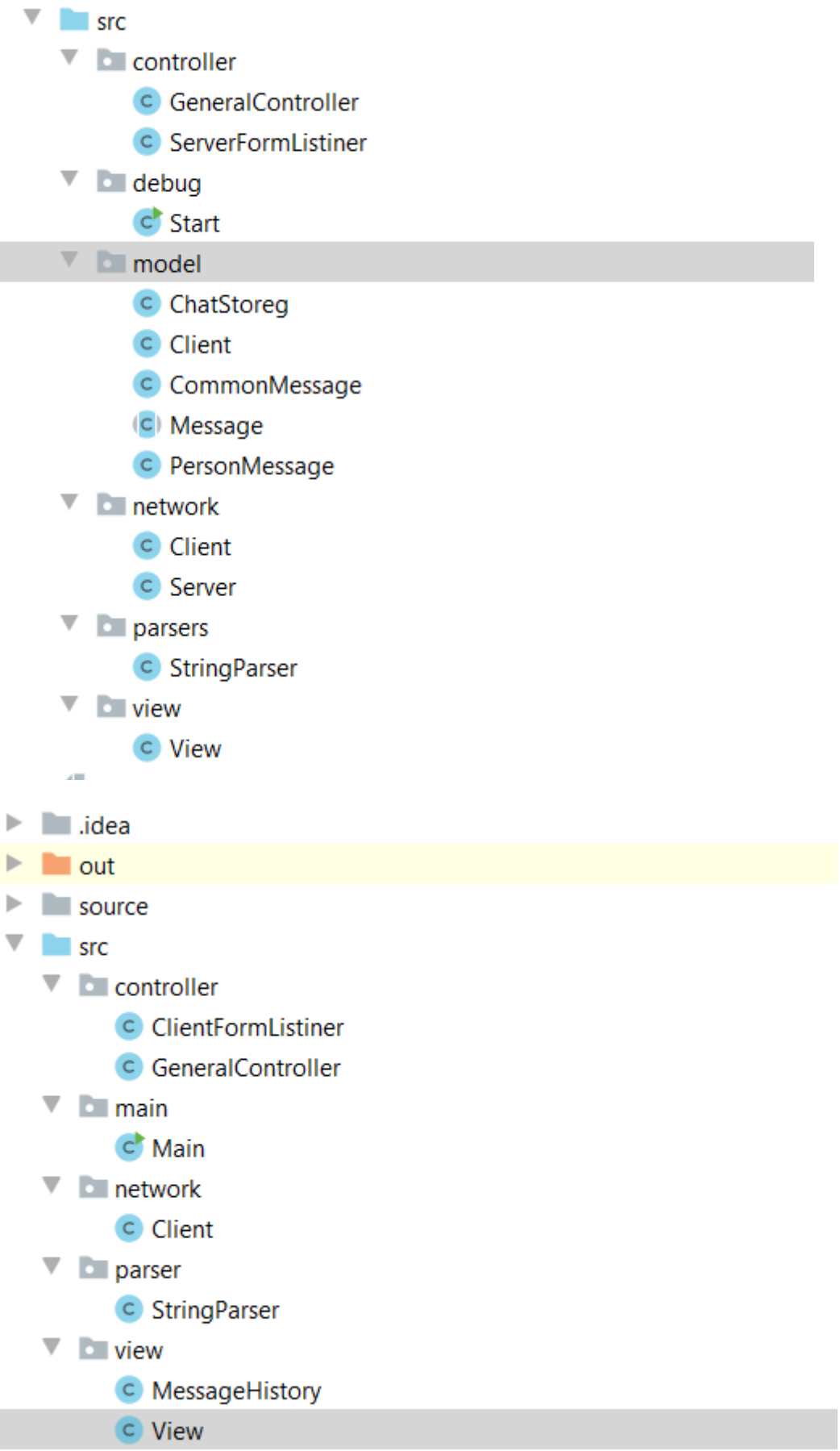
Chat – хранит все сообщения которые были отправленные сообщения также для связи с классом Message. Также для связи с классом Client хранится все пользователи.

В классе Client описанные данные о клиенте. Также храниться все отправленные сообщения этого пользователя.

Класс Message описывает сообщение его характеристик. Поле sender связывает его с классом Client.

Класс PersonMessage и CommonMessage нужны для определения тип сообщений личные или обще доступные.

Описание классов



Класс GeneralController

Package controller

Class GeneralController

java.lang.Object
    controller.GeneralController

```
public class GeneralController
extends java.lang.Object
```

Главный контроллер приложения. Связывает все части проекта, и обрабатывает их.

Version:

1.0

Author:

Зиняков Ефим

Field Summary

Fields

Modifier and Type	Field	Description
static ChatStoreg	chatStoreg	Хранилище
static java.lang.String	FILE_SETTINGS_NAME	Путь где хранится файл с настройками
static java.util.Properties	properties	Коллекция настроек
static Server	server	Сервер
static ServerFormListiner	sfl	Обработчик формы
static View	view	Форма

Constructor Summary

Constructors

Constructor	Description
GeneralController()	

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method	Description
static void	connect(int port)	Обаботчки для кнопки 'Старт' из формы.
static void	disconnect(java.lang.String client, int id)	Отлючение пользователя от сервера
static java.util.ArrayList<java.lang.String>	findAllLoginIsOnline()	Поиск в хранилище всех пользоватей которые в сети
static java.lang.String	findLoginClientById(int id)	Поиск в хранилище логин клиента по id
static void	getMessages(Client client)	Получение истории сообщений из хранилища
static void	login(Client client)	Авторизация пользователя.
static void	startApp()	Запуск всего приложение.
static void	stopServer()	Отключение сервера.
static void	submitMessage(int senderID, java.lang.String msg, java.lang.String whom)	Отправка сообщений на клиент.

Класс ServerFormListiner

Package controller

Class ServerFormListiner

java.lang.Object
    controller.ServerFormListiner

All Implemented Interfaces:

`java.awt.event.ActionListener, java.util.EventListener`

```
public class ServerFormListiner
extends java.lang.Object
implements java.awt.event.ActionListener
```

Класс является слушателем компонентов формы. В себе содержит методы, которые будут срабатывать во время действий на форме.

Version:

1.0

Author:

Ванцын Дмитрий

Field Summary

Fields

Modifier and Type	Field	Description
`private static View`	`view`	Интерфейс программы

Constructor Summary

Constructors

Constructor	Description
`ServerFormListiner(View view)`	Конструктор, который создаёт слушателя компонентов формы, а так же добавляющий этогос лушателя компонентам формы

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`void`	`actionPerformed(java.awt.event.ActionEvent e)`	Унаследованный метод, который реагирует на события на форме
`static void`	`errorMes(java.lang.String str)`	Метод, который выводит ошибки в диалоговые окна

Класс Start

Package debug

Class Start

java.lang.Object
    debug.Start

```
public class Start
extends java.lang.Object
```

Constructor Summary

Constructors

Constructor	Description
`Start()`	

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`static void`	`main(java.lang.String[] arg)`	

Класс ChatStorage

Package model

Class ChatStoreg

java.lang.Object
    model.ChatStoreg

All Implemented Interfaces:

`java.io.Serializable`

```
public class ChatStoreg
extends java.lang.Object
implements java.io.Serializable
```

Класс являет хранилещем всего чата. Реализующий паттерн проектирование SINGLETON В себе содержит коллекцию всех клиентов `clients` (клиенты загружаются с файла после старта сервера) Коллекцию сообщений `message`, заполнение данной коллекции идет из файла (старые) и когда с сервера приходит сообщение(новые) Статическое поле `COUNT_ID_MESSAGE` - хранит в себе id сообщений.

Version:

1.0

Author:

Зиняков Ефим

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
`private static ChatStoreg`	`chatStoreg`	Поле для реализации паттерна SINGLETON
`private java.util.HashMap<java.lang.Integer,Client>`	`clients`	Коллекция которая хранит всех пользователей загружаемых из файла
`static int`	`COUNT_ID_MESSAGE`	Хранит в себе id сообщений
`private java.util.ArrayList<Message>`	`message`	Коллекция которая все сообщения

Constructor Summary

Constructors

Modifier	Constructor	Description
private	`ChatStoreg()`	Приватный конструктор для реализции SINGLETON.

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`Message`	`addMessage(java.util.Date date, java.lang.String message, Client sender, Client recipient)`	Создание сообщение `CommonMessage`,`PersonMessage` и добавление его в коллекцию.
private void	`createClient(int id, java.lang.String login, java.lang.String password)`	Добавление клиента в коллекцию клиентов.
`java.util.ArrayList<java.lang.String>`	`findAllLoginIsOnline()`	Ищет клиентов типа `Client` которые онлайн в коллекции.
`Client`	`findClientByLogin(java.lang.String login)`	Ищет клиента типа `Client` по его логину в коллекции.
java.lang.String	`findLoginClientById(int id)`	Ищет в коллекции логин клиента по его id
static ChatStoreg	`getChatStore()`	Метод нужен для создание объекта данного класса.
`Client`	`getClient(int id)`	Достать клиента по его id
`java.util.ArrayList<Message>`	`getMessage()`	Получить коллекцию всех сообщений.
void	`load(java.lang.String pathMessage, java.lang.String pathClientMsg, java.lang.String pathClient)`	Метод загружает хранилище из файла.
private void	`loadClientMsg(java.lang.String path)`	Загрузка сообщений определенного пользователя (только то что он отправлял) из файла.
private void	`loadClients(java.lang.String path)`	Загрузка пользователь в коллекцию `clients` из файла.
private void	`loadMessage(java.lang.String path)`	Загрузка сообщений в коллекцию `message` из файла.
int	`login(java.lang.String login, java.lang.String password)`	Авторизация пользователя.
void	`logout(int id)`	Меняет пользователю(`Client`) поле isOnline = false (т.е не в сети).
void	`save(java.lang.String pathMessage, java.lang.String pathClientMsg)`	Метод сохраняет хранилище в файл.
void	`saveClientMsg(java.lang.String path)`	Сохранение сообщений определенного пользователя (только то что он отправлял) в файл Сохранение происходит с помощью сериализации.
private void	`saveMessage(java.lang.String path)`	Сохранение сообщений из коллекци `message` в файл Сохранение происходит с помощью сериализации.

Класс Client

Package model

Class Client

java.lang.Object
    model.Client

All Implemented Interfaces:

java.io.Serializable

```
public class Client
extends java.lang.Object
implements java.io.Serializable
```

Класс реализуеший модель клиента.

Version:

1.0

Author:

Зиняков Ефим

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
private int	id	Уникальный идентификатор клиента
private boolean	isOnline	Поле показывающие в сети ли клиент
private java.lang.String	login	Уникальный логин клиента
private java.util.ArrayList<Message>	messages	Хранилище сообщений определнного клиента
private java.lang.String	password	Пароль клиента

Constructor Summary

Constructors

Constructor	Description
Client(int id, java.lang.String login, java.lang.String password)	Конструктор - создание нового объекта клиента.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	addMessage(Message message)	Добавление сообщение в коллекцию сообщений клиента
int	getId()	Получение значение поля id
java.lang.String	getInfo()	Вывести информацию о полях клиента
java.lang.String	getLogin()	Получение значение поля login
java.util.ArrayList<Message>	getMessages()	Получение значение поля messages
boolean	isOnline()	Получение значение поля isOnline
int	login(java.lang.String login, java.lang.String password)	Проверка для авторизации пользователя.
void	setMessages(java.util.ArrayList<Message> messages)	Заменяет поля messages
void	setOnline(boolean online)	Замена поля isOnline

Класс CommonMessage

Package model

Class CommonMessage

java.lang.Object
    model.Message
        model.CommonMessage

All Implemented Interfaces:

java.io.Serializable

```
public class CommonMessage
extends Message
implements java.io.Serializable
```

Класс который определяет сообщение для всех. Наследует класс Message Message "

Version:

1.0

Author:

Зиняков Ефим

See Also:

Serialized Form

Method Summary

Methods inherited from class model.Message
getInfo, getSender, sendMessage

Класс Message

Package model

Class Message

java.lang.Object
    model.Message

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

CommonMessage, PersonMessage

```
public abstract class Message
extends java.lang.Object
implements java.io.Serializable
```

Класс, который отвечает за сообщение. В себе он хранит всю информацию о сообщение, такие как уникальны id, рремя когда было отправленно сообщение, само сообщение, и кто откравил (отправителем считается клиент Client) Являтся абстрактным.

Version:

1.0

Author:

Зиняков Ефим

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
protected java.util.Date	date	Время когда было отправленно сообщение
protected int	id	Уникальный идентификатор клиента
protected java.lang.String	message	Сообщение
protected Client	sender	Отправитель.

Constructor Summary

Constructors

Constructor	Description
Message(int id, java.lang.String message, Client sender, java.util.Date date)	Конструктор - создает объект данного класса.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
java.lang.String	getInfo()	Мутод, который дает информацию о сообщение (оформленную)
Client	getSender()	Вернет поле отправителя sender
java.lang.String	sendMessage()	Метод который формирует строку для отправки ее на клиент

Класс PersonMessage

Package model

Class PersonMessage

java.lang.Object
  model.Message
    model.PersonMessage

All Implemented Interfaces:

`java.io.Serializable`

```
public class PersonMessage
extends Message
implements java.io.Serializable
```

Класс который определяет личные сообщения. Наследует класс Message `Message` Отличие от других классов Message тут есть поле которые хранит в себе получателя.

Version:

1.0

Author:

Зиняков Ефим

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
`protected Client`	`recipient`	Поле получатель.

Fields inherited from class model.Message

`date, id, message, sender`

Constructor Summary

Constructors

Constructor	Description
`PersonMessage(int id, java.lang.String message, Client sender, Client recipient, java.util.Date date)`	Конструктор - создание объекта данного класса Вызывает конструкот базового класса `Message(int, String, Client, Date)`, а также заполняет поле `recipient` *

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`java.lang.String`	`getInfo()`	Формирует строку для вывода информации об объекте.
`Client`	`getRecipient()`	Получить поле `recipient`
`java.lang.String`	`sendMessage()`	Формирует строку для отправки ее на клиент.

Methods inherited from class model.Message

`getSender`

Класс Client

Package network

Class Client

java.lang.Object
    java.lang.Thread
        network.Client

All Implemented Interfaces:

`java.lang.Runnable`

```
public class Client
extends java.lang.Thread
```

Серверны клиент. Принимает сообщение которые пришли с клиента и отпраляет `GeneralController` для обработки.

Version:

1.0

Author:

Зиняков Ефим

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields

Modifier and Type	Field	Description
`private int`	**`clientId`**	id пользователя подключеного к этому сокету
`private boolean`	**`exit`**	Флаг для выхода из из потока клиента
`private java.io.BufferedReader`	**`in`**	Поток получение данных с клиента
`private java.io.BufferedWriter`	**`out`**	Поток отправки данных на клиента
`private java.net.Socket`	**`s`**	Сокет

Fields inherited from class java.lang.Thread

`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary

Constructors

Constructor	Description
`Client``(java.net.Socket s)`	Конструктор создание объекта `Client`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`void`	**`connect`**`()`	Подключение отдельного потока для пользователя
`void`	**`disconnect`**`()`	Отключение сокета.
`int`	**`getClientId`**`()`	Получение поля `clientId`
`void`	**`run`**`()`	
`void`	**`sendMsg`**`(java.lang.String str)`	Отправить сообщение по сокету.
`void`	**`setClientId`**`(int clientId)`	Заменять или добавить в поле `clientId`
`void`	**`setExit`**`(boolean exit)`	Изменяет флаг для выхода из потока

Класс Server

Package network

Class Server

java.lang.Object
    java.lang.Thread
        network.Server

All Implemented Interfaces:

`java.lang.Runnable`

```
public class Server
extends java.lang.Thread
```

Многопоточный сервер Сервер работает на синхронном взаимодействие Хранит сессию клиента.

Version:

1.0

Author:

Зиняков Ефим

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields

Modifier and Type	Field	Description
java.util.ArrayList<Client>	clients	Все сервер сокеты
java.lang.String	port	Порт
private java.net.ServerSocket	ss	Сервер сокет

Fields inherited from class java.lang.Thread

MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

Constructors

Constructor	Description
Server(int port)	Конструктор - создание объекта сокет.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	connect()	Запуск отдельного потока для сервера
void	run()	
void	sendAllMsg(java.lang.String str)	Отправка сообщения всем
void	sendPersonMsg(java.lang.String str, int senderId, int whomId)	Отправка личного сообщения.
void	stopServer()	Остановка сервера и отключение пользователей от сервера

Класс StringParser

Package parsers

Class StringParser

java.lang.Object
  parsers.StringParser

public class **StringParser**
extends java.lang.Object

Парсер строки которая приходит с клиента. Парсер разбивает строку на тип задачи и пременную по типу ключ значение. Строку он заносить в HashMap, где ключ типа String и его значение тоже String. В роли значение может быть массив. Чтобы получить значение переменной нужно вызвать метод `getProperty(String)` куда стоит передать ключ(т.е название переменной). Если переменная является массивом то нужно вызвать медот `getPropertys(String)` куда также стоит передать ключ.

Version:

1.0

Author:

Зиняков Ефим

Nested Class Summary

Nested Classes

Modifier and Type	Class	Description
static class	StringParser.type	Enum, который хранит тип задачи LOGIN - проверка при подключение DISCONNECT - отключение пользователя SUBMIT_MESSAGE - отправка сообщений на клиент GET_MESSAGE - отправка истории сообщения

Field Summary

Fields

Modifier and Type	Field	Description
`private static java.util.HashMap<java.lang.String,java.util.ArrayList<java.lang.String>>`	**`properties`**	Коллекция которая хранит все переменные и ее значения

Constructor Summary

Constructors

Constructor	Description
`StringParser`()	

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
`static void`	**`flush`**()	Очищение хранилища переменных
`static java.lang.String`	**`getProperty`**(java.lang.String key)	Метод который по ключи вернет значение
`static java.util.ArrayList<java.lang.String>`	**`getPropertys`**(java.lang.String key)	Метод который по ключу достает массивы значений
`static void`	**`info`**()	Выводит в консоль читаемую информацию всех ключей и их значений
`static void`	**`parsingString`**(java.lang.String str)	Разбите строки и занесение ключей и значений в `properties` Спец символы: "#;" - разделитель для переменных или типа задачи. "#:" - разделяет переменную и ее значение "#,"- разделить для значения типа массива, перечесление элементов массива Как должна выглядить передоваемся строка: [тип задачи]#;[имя переменной#:значение переменной]#; перменных может быть не ограниченое кол-во. Пример: LOGIN#;msg#:LOGINFAIL IS_ONLINE#;msg#:Пользователи онлайн#;client#: Efim#,Dima#,Lexa

Класс StringParser.type

Package parsers

Enum StringParser.type

java.lang.Object
  java.lang.Enum<StringParser.type>
    parsers.StringParser.type

All Implemented Interfaces:

`java.io.Serializable, java.lang.Comparable<StringParser.type>, java.lang.constant.Constable`

Enclosing class:

StringParser

```
public static enum StringParser.type
extends java.lang.Enum<StringParser.type>
```

Enum, который хранит тип задачи
LOGIN - проверка при подключение
DISCONNECT - отключение пользователя
SUBMIT_MESSAGE - отправка сообщений на клиент
GET_MESSAGE - отправка истории сообщения

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Enum

`java.lang.Enum.EnumDesc<E extends java.lang.Enum<E>>`

Enum Constant Summary

Enum Constants

Enum Constant	Description
DISCONNECT	
GET_MESSAGES	
LOGIN	
SUBMIT_MESSAGE	

Constructor Summary

Constructors

Modifier	Constructor	Description
private	type()	

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method	Description
static StringParser.type	valueOf(java.lang.String name)	Returns the enum constant of this type with the specified name.
static StringParser.type[]	values()	Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum

clone, compareTo, describeConstable, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

Класс View

Package view

Class View

java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
                        view.View

All Implemented Interfaces:

`java.awt.image.ImageObserver`, `java.awt.MenuContainer`, `java.io.Serializable`, `javax.accessibility.Accessible`, `javax.swing.RootPaneContainer`, `javax.swing.WindowConstants`

```
public class View
extends javax.swing.JFrame
```

Класс является интерфейсом программы. В себе содержит компонеты интерфейса

Version:

1.0

Author:

Ванцын Дмитрий

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JFrame

`javax.swing.JFrame.AccessibleJFrame`

Nested classes/interfaces inherited from class java.awt.Frame

`java.awt.Frame.AccessibleAWTFrame`

Nested classes/interfaces inherited from class java.awt.Window

`java.awt.Window.AccessibleAWTWindow`, `java.awt.Window.Type`

Nested classes/interfaces inherited from class java.awt.Container

`java.awt.Container.AccessibleAWTContainer`

Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.AccessibleAWTComponent`, `java.awt.Component.BaselineResizeBehavior`, `java.awt.Component.BltBufferStrategy`, `java.awt.Component.FlipBufferStrategy`

Field Summary

Fields

Modifier and Type	Field	Description
private javax.swing.JButton	buttonExit	Кнопка выхода из прилодения
private javax.swing.JButton	buttonStart	Кнопка старта сервера
private javax.swing.JButton	buttonStop	Кнопка остановки сервера
private javax.swing.JLabel	labelPort	Надпись, показывающая, куда вводить порт
private javax.swing.JPanel	loginPanel	Панель, содержащая в себе компонеты для запуска севера
private javax.swing.JPanel	logPanel	Панель, содержащая в себе тексовую обоасть, для вывода логов
private javax.swing.JPanel	mainPanel	Панель, содержащая в себе панели формы
private javax.swing.JScrollPane	scroll	Панель, для добавления на текстовую область, для вывода логов скролл
private javax.swing.JTextField	textFieldPort	Текстовая панель, для ввода порта
private javax.swing.JTextArea	textLog	Текстовая область, для вывода логов

Fields inherited from class javax.swing.JFrame

accessibleContext, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame

CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Constructor Summary

Constructors

Constructor	Description
View(java.lang.String title)	Конструктор, создающий интерфес серверной части программы.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	connect()	Переклчючает доступности кнопок на форме, если клиент подключился к серверу
void	disconnect()	Переключает доступность кнопок на форме, если клиент отключился от сервера
javax.swing.JButton	getButtonExit()	Получить доступ к кнопке подключиться
javax.swing.JButton	getButtonStart()	Получить доступ к кнопке подключиться
javax.swing.JButton	getButtonStop()	Получить доступ к кнопке подключиться
javax.swing.JTextField	getTextFieldPort()	Получить доступ к текстовому полю, в котором указан порт для подключения к серверу
void	init()	Задаёт положение всех компонентов, располагающихся на форме
void	sendMes(java.lang.String message)	Функция, которая выводит заданное сообщение в форму, указывая время, в которое оно было отправлено

Клиентская форма:

Класс View

Package view

Class View

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame
            view.View

All Implemented Interfaces:

`java.awt.image.ImageObserver`, `java.awt.MenuContainer`, `java.io.Serializable`, `javax.accessibility.Accessible`, `javax.swing.RootPaneContainer`, `javax.swing.WindowConstants`

```
public class View
extends javax.swing.JFrame
```

Класс является интерфейсом программы. Реализующий паттерн проектирование SINGLETON В себе содержит компоненты интерфейса

Version:

1.0

Author:

Ванцын Дмитрий

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JFrame

`javax.swing.JFrame.AccessibleJFrame`

Nested classes/interfaces inherited from class java.awt.Frame

`java.awt.Frame.AccessibleAWTFrame`

Nested classes/interfaces inherited from class java.awt.Window

`java.awt.Window.AccessibleAWTWindow, java.awt.Window.Type`

Nested classes/interfaces inherited from class java.awt.Container

`java.awt.Container.AccessibleAWTContainer`

Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy`

Field Summary

Fields

Modifier and Type	Field	Description
`private javax.swing.JButton`	`buttonConnect`	Кнопка подключиться к серверу
`private javax.swing.JButton`	`buttonDisconnect`	Кнопка отключиться от сервера
`private javax.swing.JButton`	`buttonEnter`	Кнопка отправить сообщение
`private javax.swing.JButton`	`buttonExit`	Кнопка выход из приложения
`private javax.swing.JButton`	`buttonGetAllMessage`	Кнорка, для просмотра истории переписки
`private javax.swing.JButton`	`buttonIsOnline`	Кнопка для просмотра пользователей, которые подключены к сереверу
`private javax.swing.JPanel`	`buttonPanel`	Панель, содержащая в себе кнопки подключиться, отключиться и выход
`private javax.swing.JTextArea`	`chat`	Текстовая область, в которую будут выводиться сообщения
`private javax.swing.JPanel`	`chatPanel`	Панель, содержащая в себе текстовую область, в которую будут выводиться сообщения
`private javax.swing.JPanel`	`connectPanel`	Панель, содержащая в себе компонеты для подключения к северу

private javax.swing.JPanel	enterPanel	Панель, содержащая в себе компоненты для отправки сообщений, а так же для просмотра истории сообщений и пользователей онлайн
private javax.swing.JLabel	labelHost	Надпись, показывающая, куда вводить хост
private javax.swing.JLabel	labelLogin	Надпись, показывающая, куда вводить логин
private javax.swing.JLabel	labelPassword	Надпись, показывающая, куда вводить пароль
private javax.swing.JLabel	labelPort	Надпись, показывающая, куда вводить порт
private javax.swing.JPanel	mainPanel	Панель, содержащая в себе панели формы
private javax.swing.JScrollPane	scroll	Поле, прикрепляющие к текстовой области скрол
private javax.swing.JComboBox<java.lang.String>	selectUsers	Поле для выбора, кому отправить сообщение
private javax.swing.JTextField	textFieldEnter	Поле, в которое надо ввести сообщение
private javax.swing.JTextField	textFieldHost	Текстовая панель, для ввода хоста
private javax.swing.JTextField	textFieldLogin	Текстовая панель, для ввода логина
private javax.swing.JTextField	textFieldPassword	Текстовая панель, для ввода пароля
private javax.swing.JTextField	textFieldPort	Текстовая панель, для ввода порта
(package private) java.lang.String[]	users	Поле для начальной инициализации поля выбора, кому отправить сообщение
private static View	view	Поле для реализации паттерна SINGLETON

Constructor Summary

Constructors

Modifier	Constructor	Description
private	View(java.lang.String title)	Приватный конструктор для реализции SINGLETON.

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	connect()	Переклчючает доступности кнопок на форме, если клиент подключился к серверу
void	disconnect()	Переключает доступность кнопок на форме, если клиент отключился от сервера
javax.swing.JButton	getButtonConnect()	Получить доступ к кнопке подключиться
javax.swing.JButton	getButtonDisconnect()	Получить доступ к кнопке отключится
javax.swing.JButton	getButtonEnter()	Получить доступ к кнопке отправить сообщение
javax.swing.JButton	getButtonExit()	Получить доступ к кнопке выход
javax.swing.JButton	getButtonGetAllMessage()	Получить доступ к кнопке История сообщений
javax.swing.JButton	getButtonIsOnline()	Получить доступ к кнопке посмотрть всех пользователей онлайн
javax.swing.JComboBox<java.lang.String>	getSelectUsers()	Получить доступ к меню, в котором можно выбрать, кому отправить сообщение
javax.swing.JTextField	getTextFieldEnter()	Получить доступ к текстовому полю, в котором указано сообщение
javax.swing.JTextField	getTextFieldHost()	Получить доступ к текстовому полю, в котором указан хост для подключения к серверу
javax.swing.JTextField	getTextFieldLogin()	Получить доступ к текстовому полю, в котором указан логин для подключения к серверу
javax.swing.JTextField	getTextFieldPassword()	Получить доступ к текстовому полю, в котором указан пароль для подключения к серверу
javax.swing.JTextField	getTextFieldPort()	Получить доступ к текстовому полю, в котором указан порт для подключения к серверу
static View	getView()	Метод нужен для создание объекта данного класса.
void	init()	Задаёт положение всех компонентов, располагающихся на форме
void	sendMes(java.lang.String message)	Функция, которая выводит заданное сообщение в форму, указывая время, в которое оно было отправлено
void	sendMesNoTime(java.lang.String message)	Функция, которая выводит заданное сообщение в форму, не указывая время, в которое оно было отправлено

Класс MessageHistory

Package view

Class MessageHistory

java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
                        view.MessageHistory

All Implemented Interfaces:

`java.awt.image.ImageObserver`, `java.awt.MenuContainer`, `java.io.Serializable`, `javax.accessibility.Accessible`, `javax.swing.RootPaneContainer`, `javax.swing.WindowConstants`

```
public class MessageHistory
extends javax.swing.JFrame
```

Класс является интерфейсом диалогового окна, показывающего историю сообщений. В себе содержит компонеты интерфейса диалогового окна, показывающего историю сообщений.

Version:

1.0

Author:

Ванцын Дмитрий

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
`private javax.swing.JButton`	`buttonClose`	Кнопка, для выхода из интерфеса диалогового окна
`private static javax.swing.JTextArea`	`history`	Поле, содержащая в себе текстовую область, для вывода сообщений
`private javax.swing.JPanel`	`mainPanel`	Панель, содержащая в себе все компоненты формы
`private javax.swing.JScrollPane`	`scroll`	Поле, добавляющееe к текстовой области, для вывода сообщений скролл

Constructor Summary

Constructors

Constructor	Description
`MessageHistory(java.lang.String title)`	Конструктор, создающий форму, для просмотра истории сообщений

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`void`	`init()`	Задаёт положение всех компонентов, располагающихся на форме
`static void`	`setHistory(java.lang.String str)`	Изменяет историю сообщений

Класс ClientFormListiner

Package controller

Class ClientFormListiner

java.lang.Object
    controller.ClientFormListiner

All Implemented Interfaces:

`java.awt.event.ActionListener`, `java.awt.event.MouseListener`, `java.util.EventListener`

```
public class ClientFormListiner
extends java.lang.Object
implements java.awt.event.ActionListener, java.awt.event.MouseListener
```

Класс является слушателем компонентов формы. В себе содержит методы, которые будут срабатывать во время действий на форме.

Version:

1.0

Author:

Ванцын Дмитрий

Field Summary

Fields

Modifier and Type	Field	Description
`private static java.util.ArrayList<java.lang.String>`	`userIsOnline`	Коллекция, содержащая в себе всех пользователей, находящихся онлайн
`private static View`	`view`	Интерфейс программы

Constructor Summary

Constructors

Constructor	Description
`ClientFormListiner(View view)`	Конструктор, который создаёт слушателя компонентов формы, а так же добавляющий этогос лушателя компонентам формы

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`void`	`actionPerformed(java.awt.event.ActionEvent e)`	Унаследованный метод, который реагирует на события на форме
`static void`	`addToUser(java.lang.String user)`	Добавляет пользователей онлайн, в меню выбора кому отправить сообщения
`static void`	`errorMes(java.lang.String str)`	Метод, который выводит ошибки в диалоговые окна
`void`	`mouseClicked(java.awt.event.MouseEvent e)`	Унаследованные методы, которые реагируют на дейтсвия мыши на форме
`void`	`mouseEntered(java.awt.event.MouseEvent e)`	
`void`	`mouseExited(java.awt.event.MouseEvent e)`	
`void`	`mousePressed(java.awt.event.MouseEvent e)`	
`void`	`mouseReleased(java.awt.event.MouseEvent e)`	
`static void`	`removeUser(java.lang.String user)`	Удаляет пользователей, которые отключились от сервера, из меню выбора кому отправить сообщения

Класс GeneralController

Package controller

Class GeneralController

java.lang.Object
    controller.GeneralController

```
public class GeneralController
extends java.lang.Object
```

Главный контроллер приложения. Связывает все части проекта, и обрабатывает их.

Version:

1.0

Author:

Ванцын Дмитрий

Field Summary

Fields

Modifier and Type	Field	Description
static Client	client	Клиент
static java.lang.String	FILE_SETTINGS_NAME	Путь где хранится файл с настройками
static ClientFormListiner	kFL	Слушатель интерфеса программы
static java.util.Properties	properties	Коллекция настроек
static View	view	Интерфейс программы

Constructor Summary

Constructors

Constructor	Description
GeneralController()	

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method	Description
static void	checkConnect(java.lang.String str)	Проверка подключения к серверу
static void	connect(java.lang.String host, int port)	Обаботчки для кнопки 'Старт' из формы.
static void	disconnect()	Отлючение пользователя от сервера
static void	enter(java.lang.String msg, java.lang.String whom)	Отправка сообщений на сервер
static void	getMessageHistory()	Получение пользователей онлайн
static void	setMessageHistory()	Изменение истории сообщений
static void	setOnline()	Изменение пользователей онлайн
static void	startApp()	Запуск всего приложение.
static void	stopServer(Client client)	Остановка сервера

Класс Main

Package main

Class Main

java.lang.Object
    main.Main

```
public class Main
extends java.lang.Object
```

Constructor Summary

Constructors

Constructor	Description
`Main()`	

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method
`static void`	`main(java.lang.String[] str)`

Package network

Class Client

java.lang.Object
    java.lang.Thread
        network.Client

All Implemented Interfaces:

`java.lang.Runnable`

```
public class Client
extends java.lang.Thread
```

Клиент. Принимает сообщение которые пришли с сервера и отпраляет в `GeneralController` для обработки.

Version:

1.0

Author:

Зиняков Ефим

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields

Modifier and Type	Field	Description
private boolean	exit	Флаг для выхода из из потока клиента
private java.lang.String	host	
private java.io.BufferedReader	in	Поток получение данных с клиента
private java.io.BufferedWriter	out	Поток отправки данных на клиента
private int	port	
private java.net.Socket	s	Сокет

Fields inherited from class java.lang.Thread

MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

Constructors

Constructor	Description
Client(java.lang.String host, int port)	Конструктор создание объекта

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	connect()	Подключение отдельного потока для пользователя
void	run()	
void	sendMsg(java.lang.String str)	Отправить сообщение по сокету.
void	setExit(boolean exit)	Изменяет флаг для выхода из потока

Класс StringParser

Package parser

Class StringParser

java.lang.Object
    parser.StringParser

```
public class StringParser
extends java.lang.Object
```

Парсер строки которая приходит с клиента. Парсер разбивает строку на тип задачи и пременную по типу ключ значение. Строку он заносить в HashMap, где ключ типа String и его значение тоже String. В роли значение может быть массив. Чтобы получить значение переменной нужно вызвать метод `getProperty(String)` куда стоит передать ключ(т.е название переменной). Если переменная являеться массивом то нужно вызвать медот `getPropertys(String)` куда также стоит передать ключ.

Version:

1.0

Author:

Зиняков Ефим

Nested Class Summary

Nested Classes

Modifier and Type	Class	Description
static class	**StringParser.type**	Enum, который хранит тип задачи LOGIN - проверка при подключение DISCONNECT - отключение пользователя SUBMIT_MESSAGE - отправка сообщений на сервер MESSAGE - истории сообщения STOP_SERVER - отключение сервера BASE - вывод сообщение в лог IS_ONLINE - пользователи онлайн

Field Summary

Fields

Modifier and Type	Field	Description
private static java.util.HashMap<java.lang.String,java.util.ArrayList<java.lang.String>>	**properties**	Коллекция которая хранит все переменные и ее значения

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method	Description
static void	**flush**()	Очищение хранилища переменных
static java.lang.String	**getProperty**(java.lang.String key)	Метод который по ключи вернет значение
static java.util.ArrayList<java.lang.String>	**getPropertys**(java.lang.String key)	Метод который по ключу достает массивы значений
static void	**info**()	Выводит в консоль читаемую информацию всех ключей и их значений
static void	**parsingString**(java.lang.String str)	Разбите строки и занесение ключей и значений в `properties` Спец символы: "#;" - разделитель для переменных или типа задачи. "#:" - разделяет переменную и ее значение "#,"- разделить для значения типа массива, перечесление элементов массива Как должна выглядить передоваемся строка: [тип задачи]#;[имя переменной#:значение переменной]#; перменных может быть не ограниченое кол-во. Пример: LOGIN#;msg#:LOGINFAIL IS_ONLINE#;msg#:Пользователи онлайн#;client#: Efim#,Dima#,Lexa

Класс StringParser.Enum

Package parser

Enum StringParser.type

java.lang.Object
    java.lang.Enum<StringParser.type>
        parser.StringParser.type

All Implemented Interfaces:

`java.io.Serializable, java.lang.Comparable<StringParser.type>, java.lang.constant.Constable`

Enclosing class:

StringParser

```
public static enum StringParser.type
extends java.lang.Enum<StringParser.type>
```

Enum, который хранит тип задачи
LOGIN - проверка при подключение
DISCONNECT - отключение пользователя
SUBMIT_MESSAGE - отправка сообщений на сервер
MESSAGE - истории сообщения
STOP_SERVER - отключение сервера BASE - вывод сообщение в лог IS_ONLINE - пользователи онлайн

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Enum

`java.lang.Enum.EnumDesc<E extends java.lang.Enum<E>>`

Enum Constant Summary

Enum Constants

Enum Constant
BASE
DISCONNECT
IS_ONLINE
LOGIN
MESSAGE
STOP_SERVER
SUBMIT_MESSAGE

Листинг приложения.

Сервер:

GeneralController

```
package controller;

import model.*;
import network.Server;
import parsers.StringParser;
import view.View;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.Properties;

/**
 * Главный контроллер приложения.
 * Связывает все части проекта, и обрабатывает их.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class GeneralController {
    /** Путь где хранится файл с настройками */
    public static String FILE_SETTINGS_NAME = "source/config.properties";
    /** Коллекция настроек */
    public static Properties properties = new Properties();
    /** Хранилище  */
    public static ChatStoreg chatStoreg = null;
    /** Сервер  */
    public static Server server = null;
    /** Форма  */
    public static View view = null;
    /** Обработчик формы  */
    public static ServerFormListiner sfl = null;

    /**
     * Запуск всего приложение. Инициализация полей
     */
    public static void startApp(){
        try{
            properties.load(new FileInputStream(FILE_SETTINGS_NAME));
            chatStoreg = ChatStoreg.getChatStore();
            view = new View("Сервер");
            sfl = new ServerFormListiner(view);
        } catch (IOException exception) {
            ServerFormListiner.errorMes(exception.getMessage());
        }
    }

    /**
     * Обаботчки для кнопки 'Старт' из формы.
     * Задание сервера, загрузка в хранилище из файла.
     * @param port Порт, по которому будет подключаться сервер.
     */
    public static void connect(int port){
        try {
            server = new Server(port);
```

```
chatStoreg.load(properties.getProperty("pathOldMessage"),properties.getProperty("pathClientMsg"),properti
es.getProperty("pathClients"));
        }catch (ClassNotFoundException e){
            ServerFormListiner.errorMes(e.getMessage());
        } catch (IOException exception) {
            ServerFormListiner.errorMes(exception.getMessage());
        }
    }

    /**
     * Отправка сообщений на клиент. Добавление сообщения в хранилище
     * @param senderID отправитель
     * @param msg сообщение
     * @param whom кому
     */
    public static void submitMessage(int senderID,String msg,String whom){
        if (whom.equals("Всем")){
            Message message = chatStoreg.addMessage(new Date(),msg, chatStoreg.getClient(senderID),null);
            GeneralController.server.sendAllMsg("SUBMIT_MESSAGE#;msg#:" + message.sendMessage());
        }
        else{
            Message message = chatStoreg.addMessage(new Date(),msg,chatStoreg.getClient(senderID),
chatStoreg.findClientByLogin(whom));

server.sendPersonMsg("SUBMIT_MESSAGE#;msg#:"+message.sendMessage(),senderID,chatStoreg.findClie
ntByLogin(whom).getId());
        }
    }

    /**
     * Авторизация пользователя.
     * @param client клиент, который пытается подключиться
     */
    public static void login(network.Client client){

client.setClientId(chatStoreg.login(StringParser.getProperty("login"),StringParser.getProperty("password")));
        if (client.getClientId() != -1) {
            client.sendMsg("LOGIN#;msg#:LOGINSUCCESS");
            String clientName = findLoginClientById(client.getClientId());
            String str = "IS_ONLINE#;msg#:Пользователь " + clientName + "
подключился#;clientIsOnline#:";
            for (String s : findAllLoginIsOnline()){
                str += s+"#,";
            }
            server.sendAllMsg(str);
            view.sendMes("Пользователь "+ clientName + " подключился!");
        } else{
            view.sendMes("Совершена неудачная попытка зайти в аккаунт " +
                    StringParser.getProperty("login"));
            client.sendMsg("LOGIN#;msg#:LOGINFAIL");
            client.setExit(false);
        }
    }

    /**
     * Отлючение пользователя от сервера
     * @param client пользователь который отключается
     * @param id его id
     */
    public static void disconnect(String client, int id){
        server.sendAllMsg("DISCONNECT#;client#:"+ client);
        view.sendMes("Пользователь "+ client + " отключился!");
        chatStoreg.logout(id);
```

```
    }

    /**
     * Отключение сервера. (т.е закрытие всех сокетов, и самого сервера)
     * Сохранение данных из хранилища в файл
     */
    public static void stopServer() {
        try {
            chatStoreg.save(properties.getProperty("pathOldMessage"),
properties.getProperty("pathClientMsg"));
            server.stopServer();
            view.disconnect();
        } catch (IOException exception) {
            exception.printStackTrace();
        }
    }

    /**
     * Поиск в хранилище логин клиента по id
     * @param id id, по которому нужно искать
     * @return Вернет логин
     */
    public static String findLoginClientById(int id){
        return chatStoreg.findLoginClientById(id);
    }

    /**
     * Поиск в хранилище всех пользователей которые в сети
     * @return Вернет коллекцию всех пользователей онлайн
     */
    public static ArrayList<String> findAllLoginIsOnline(){
        return chatStoreg.findAllLoginIsOnline();
    }

    /**
     * Получение истории сообщений из хранилища
     * @param client клиент который запросил историю.
     */
    public static void getMessages(network.Client client) {
        String str = "";
        for (Message message : chatStoreg.getMessage()) {
            if(message instanceof CommonMessage)
                str += message.sendMessage() + ("#n");
            else if (message instanceof PersonMessage){
                if (message.getSender().getId() == client.getClientId() ||
                        ((PersonMessage) message).getRecipient().getId() == client.getClientId())
                    str += message.sendMessage() + ("#n");
            }
        }
        client.sendMsg("MESSAGE#;msg#:" + (str.equals("")?"История сообщений пуста!":str));
    }
}
```

ServerFormListiner

```
package controller;

import view.View;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```java
import static java.lang.System.exit;
/**
 * Класс является слушателем компонентов формы.
 * В себе содержит методы, которые будут срабатывать во время действий на форме.
 * @author Ванцын Дмитрий
 * @version 1.0
 */
public class ServerFormListiner implements ActionListener {
    /**
     * Интерфейс программы
     */
    private static View view = null;
    /**
     * Конструктор, который создаёт слушателя компонентов формы, а так же добавляющий этогос
лушателя компонентам формы
     * @param view Интерфейс программы
     */
    public ServerFormListiner(View view) {
        this.view = view;
        view.getButtonExit().addActionListener(this);
        view.getButtonStart().addActionListener(this);
        view.getButtonStop().addActionListener(this);
    }
    /**
     * Унаследованный метод, который реагирует на события на форме
     * @param e События на форме
     */
    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == view.getButtonExit()){
            exit(0);
        }
        if(e.getSource() == view.getButtonStop()){
            GeneralController.stopServer();
            view.sendMes("Сервер выключен!");
        }
        if(e.getSource() == view.getButtonStart()){
            String portStr = view.getTextFieldPort().getText().replaceAll("\\s+","");
            if(portStr.isEmpty()){
                errorMes("Введите порт");
                return;
            }
            if(!portStr.matches("\\d+")){
                errorMes("В поле порт введены не числа");
                return;
            }
            int port = Integer.parseInt(portStr);
            if(port<=0||port>65536) {
                errorMes("Значение в поле порт не подходит для подключения");
                return;
            }
            GeneralController.connect(port);
            view.connect();
            view.sendMes("Сервер запущен");
        }
    }
    /**
     * Метод, который выводит ошибки в диалоговые окна
     * @param str Строка с ошибкой
     */
    public static void errorMes (String str){
        JOptionPane.showMessageDialog(view, str,
```

```
            "Ошибка", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Server

```
package network;

import controller.ServerFormListiner;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.util.ArrayList;

/**
 * Многопоточный сервер
 * Сервер работает на синхронном взаимодействие
 * Хранит сессию клиента.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class Server extends Thread{
    /** Сервер сокет  */
    private ServerSocket ss;
    /** Все сервер сокеты */
    public ArrayList<Client> clients = new ArrayList<>();
    /** Порт */
    public String port;

    /**
     * Конструктор - создание объекта сокет.
     * @param port порт для задния ServerSocket
     * @throws IOException Вызываеться если не получилось создать ServerSocket
     */
    public Server(int port) throws  IOException{
        ss = new ServerSocket(port);
        this.port = String.valueOf(port);
        connect();
    }

    @Override
    /**
     * Выполнение потока. Ожидание подлючение и создание слиента.
     */
    public void run() {
        Socket s;
        while (true) {
            try {
                s = ss.accept();
                clients.add(new Client(s));
            } catch (SocketException e){
                break;
            } catch (IOException e) {
                ServerFormListiner.errorMes(e.getMessage());
            }
        }
    }
    /** Запуск отдельного потока для сервера */
    public void connect(){
        start();
```

```
    }

    /**
     * Остановка сервера и отключение пользователей от сервера
     * @throws IOException Вызываеться если произошел сбой в отключение сокета
     */
    public void stopServer() throws IOException {
        sendAllMsg("STOP_SERVER");
        for (Client c : clients){
            c.disconnect();
        }
        clients.clear();
        ss.close();
    }

    /**
     * Отправка личного сообщения.
     * @param str сообщение
     * @param senderId отправитель
     * @param whomId получатель
     */
    public void sendPersonMsg(String str,int senderId,int whomId){
        for (Client c: clients){
            if (c.getClientId() == senderId || c.getClientId() == whomId)
                c.sendMsg(str);
        }
    }

    /**
     * Отправка сообщения всем
     * @param str сообщение
     */
    public void sendAllMsg(String str){
        for(Client i : clients){
            if(i.isAlive()){
                i.sendMsg(str);
            }
        }
    }
}
```

View

```
package view;

import controller.GeneralController;

import javax.swing.*;
import java.awt.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

/**
 * Класс является интерфейсом программы.
 * В себе содержит компонеты интерфейса
 * @author Ванцын Дмитрий
 * @version 1.0
 */
public class View extends JFrame {
    /** Панель, содержащая в себе панели формы */
```

```
    private JPanel mainPanel = new JPanel();
    /** Панель, содержащая в себе тексовую обоасть, для вывода логов */
    private JPanel logPanel = new JPanel();
    /** Панель, содержащая в себе компонеты для запуска севера */
    private JPanel loginPanel = new JPanel();
    /** Надпись, показывающая, куда вводить порт */
    private JLabel labelPort = new JLabel("Порт:");
    /** Текстовая панель, для ввода порта */
    private JTextField textFieldPort = new JTextField();
    /** Кнопка старта сервера */
    private JButton buttonStart = new JButton("Старт");
    /** Кнопка остановки сервера */
    private JButton buttonStop = new JButton("Стоп");
    /** Кнопка выхода из прилодения */
    private JButton buttonExit = new JButton("Выход");
    /** Текстовая область, для вывода логов */
    private JTextArea textLog = new JTextArea();
    /** Панель, для добавления на текстовую область, для вывода логов скролл*/
    private JScrollPane scroll = new JScrollPane(textLog, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
    /**
     * Конструктор, создающий интерфес серверной части программы.
     * @param title задаёт имя формы
     * @exception HeadlessException Вызывается, когда код, зависящий от клавиатуры, дисплея или мыши,
вызывается в среде, не поддерживающей клавиатуру, дисплей или мышь.
     */
    public View(String title) throws HeadlessException {
        super(title);
        disconnect();
        init();
    }
    /** Задаёт положение всех компонентов, располагающихся на форме */
    public void init (){
        setSize(600, 450);
        setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
        loginPanel.setLayout(new GridLayout(1,4,10,10));
        labelPort.setFont(new Font(null, Font.BOLD, 13));
        labelPort.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
        loginPanel.add(labelPort);
        loginPanel.add(textFieldPort);
        loginPanel.add(buttonStart);
        loginPanel.add(buttonStop);
        loginPanel.add(buttonExit);
        loginPanel.setBorder(BorderFactory.createEmptyBorder(0,0,10,0 ));
        logPanel.setLayout(new GridLayout());
        textLog.setEditable(false);
        logPanel.add(scroll);
        mainPanel.setLayout(new BorderLayout());
        mainPanel.add(loginPanel, BorderLayout.NORTH);
        mainPanel.add(logPanel, BorderLayout.CENTER);
        mainPanel.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        add(mainPanel);
        textFieldPort.setText(GeneralController.properties.getProperty("port"));
        setLocationRelativeTo(null);
        setVisible(true);
    }
    /**
     * Переключает доступности кнопок на форме, если клиент подключился к серверу
     */
    public void connect(){
        buttonExit.setEnabled(false);
        buttonStart.setEnabled(false);
        buttonStop.setEnabled(true);
```

```
        textFieldPort.setEnabled(false);
    }
    /**
     *Переключает доступность кнопок на форме, если клиент отключился от сервера
     */
    public void disconnect(){
        buttonExit.setEnabled(true);
        buttonStart.setEnabled(true);
        buttonStop.setEnabled(false);
        textFieldPort.setEnabled(true);

    }
    /**
     * Функция, которая выводит заданное сообщение в форму, указывая время, в которое оно было
отправлено
     * @param message сообщение, которое надо вывыести в форму
     */
    public void sendMes (String message){
        Date currentDate = new Date();
        DateFormat timeFormat = new SimpleDateFormat("HH:mm:ss", Locale.getDefault());
        String timeText = timeFormat.format(currentDate);
        textLog.append(timeText + " " + message + "\n");
    }
    /**
     * Получить доступ к текстовому полю, в котором указан порт для подключения к серверу
     * @return вернёт компонент(текстовое поле с портом подключения к сереверу)
     */
    public JTextField getTextFieldPort() {
        return textFieldPort;
    }
    /**
     * Получить доступ к кнопке подключиться
     * @return вернёт компонент(кнопку старт)
     */
    public JButton getButtonStart() {
        return buttonStart;
    }
    /**
     * Получить доступ к кнопке подключиться
     * @return вернёт компонент(кнопку стоп)
     */
    public JButton getButtonStop() {
        return buttonStop;
    }
    /**
     * Получить доступ к кнопке подключиться
     * @return вернёт компонент(кнопку выход)
     */
    public JButton getButtonExit() {
        return buttonExit;
    }

}
```

ChatStoreg

```
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
```

```java
/**
 * Класс являет хранилещем всего чата. Реализующий паттерн проектирование SINGLETON
 * В себе содержит коллекцию всех клиентов {@link ChatStoreg#clients} (клиенты загружаются с файла после старта сервера)
 * Коллекцию сообщений  {@link ChatStoreg#message}, заполнение данной коллекции идет из файла (старые) и когда с сервера приходит сообщение(новые)
 * Статическое поле {@link ChatStoreg#COUNT_ID_MESSAGE} - хранит в себе id сообщений.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class ChatStoreg implements Serializable {
    /** Коллекция которая хранит всех пользователей загружаемых из файла */
    private HashMap<Integer,Client> clients;        // все пользователи
    /** Коллекция которая все сообщения */
    private ArrayList<Message> message;         // история всех сообщений
    /** Хранит в себе id сообщений */
    public static int COUNT_ID_MESSAGE = 0;
    /** Поле для реализации паттерна SINGLETON */
    private static ChatStoreg chatStoreg = null;

    /**
     * Приватный конструктор для реализции SINGLETON.
     * Для создание объект нужно вызывать метод {@link ChatStoreg#getChatStore()}
     */
    private ChatStoreg() {
        clients = new HashMap<>();
        message = new ArrayList<>();
    }
    /**
     * Метод нужен для создание объекта данного класса.
     * Или же если объект создан верене ссылку на него
     * @return Вернет созданный объект {@link ChatStoreg}
     */
    public static ChatStoreg getChatStore(){
        if (chatStoreg == null)
            chatStoreg = new ChatStoreg();
        return chatStoreg;
    }

    /**
     * Добавление клиента в коллекцию клиентов.
     * @param id id клиента
     * @param login логин клиента
     * @param password пароль клиента
     */
    private void createClient(int id,String login,String password){
        clients.put(id,new Client(id,login,password));
    }

    /**
     * Создание сообщение {@link CommonMessage},{@link PersonMessage} и добавление его в коллекцию.
     * Если сообщение предназначено для всех передаваемое значение recipient должно быть null.
     * @param date Время отправление сообщения
     * @param message Сообщение
     * @param sender Отправитель
     * @param recipient Получатель, если получателя нет нужно отправиться передать null
     * @return Вернет созданое сообщение
     */
    public Message addMessage(Date date, String message, Client sender, Client recipient){
        if (message == null) throw new IllegalArgumentException("Нужно передать сообщение!");
        if (!clients.containsValue(sender)) throw new IllegalArgumentException("Такого пользователя нет!");
        Message msg;
```

```
        if (recipient == null)
            msg = new CommonMessage(COUNT_ID_MESSAGE++,message,sender,date);
        else
            msg = new PersonMessage(COUNT_ID_MESSAGE++,message,sender,recipient,date);
        clients.get(sender.getId()).addMessage(msg);
        this.message.add(msg);
        return msg;
    }

    /**
     * Меняет пользователю({@link Client}) поле isOnline = false (т.е не в сети).
     * @param id id пользователя которому нужно изменить онлайн.
     */
    public void logout(int id){
        clients.get(id).setOnline(false);
    }

    /**
     * Авторизация пользователя. Проверяте не в сети ли пользователь и правильно ли передан пароль и
логин.
     * @param login логин для проверки
     * @param password пароль для проверки
     * @return Если авторизация успешна вернет id пользователя который прошел проверку, иначе вернет
-1
     */
    public int login(String login, String password){
        for (Client c : clients.values()) {
            int clientId = c.login(login, password);
            if (clientId != -1){
                return clientId;
            }
        }
        return -1;
    }

    /**
     * Загрузка пользователь в коллекцию {@link ChatStoreg#clients} из файла.
     * @param path путь к файлу где храняться пользователи. Формат записи в файле id:login:password
     * @throws IOException Вызывает если не получилось открыть поток с файлом или не удалось
считать данные с потока
     */
    private void loadClients(String path) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(path));
        while(true){
            String str = br.readLine();
            if (str == null) break;
            String[] buff;
            buff = str.split(":");
            createClient(Integer.parseInt(buff[0]),buff[1],buff[2]);
        }
    }

    /**
     * Сохранение сообщений определенного пользователя (только то что он отправлял) в файл
     * Сохранение происходит с помощью сериализации.
     * @param path путь к файлу.
     * @throws IOException Вызывает если не получилось открыть поток с файлом или не удалось
записать данные в файл.
     */
    public void saveClientMsg(String path) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(path));
        for (int i = 0; i < clients.size(); i++)
            oos.writeObject(clients.get(i).getMessages());
```

```
        oos.close();
    }
    /**
     * Загрузка сообщений определенного пользователя (только то что он отправлял) из файла.
     * Загрузка происходит с помощью сериализации.
     * @param path путь к файлу.
     * @throws IOException Вызывает если не получилось открыть поток с файлом или не удалось
считать данные в файл.
     * @throws ClassNotFoundException Если не удалось считать данные с файла.
     */
    private void loadClientMsg(String path) throws IOException, ClassNotFoundException {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(path));
        for (int i = 0; i<clients.size(); i++){
            ArrayList<Message> mes = (ArrayList<Message>) ois.readObject();
            if (!mes.isEmpty())
                clients.get(i).setMessages(mes);
        }
        ois.close();
    }
    /**
     * Сохранение сообщений из коллекци {@link ChatStoreg#message} в файл
     * Сохранение происходит с помощью сериализации.
     * @param path путь к файлу.
     * @throws IOException Вызывает если не получилось открыть поток с файлом или не удалось
записать данные в файл.
     */
    private void saveMessage(String path) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(path));
        oos.writeObject(message);
        oos.writeInt(COUNT_ID_MESSAGE);
        oos.close();
    }
    /**
     * Загрузка сообщений в коллекцию {@link ChatStoreg#message} из файла.
     * Загрузка происходит с помощью сериализации.
     * @param path путь к файлу.
     * @throws IOException Вызывает если не получилось открыть поток с файлом или не удалось
считать данные в файл.
     * @throws ClassNotFoundException Если не удалось считать данные с файла.
     */
    private void loadMessage(String path) throws IOException, ClassNotFoundException {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(path));
        message = (ArrayList<Message>) ois.readObject();
        COUNT_ID_MESSAGE = ois.readInt();
        ois.close();
    }

    /**
     * Метод сохраняет хранилище в файл. В методе вызываются методы
     * saveMessage - сохраняет всю историю сообщений в файл.
     * saveClientMsg - сохраняет сообщение каждого пользователя(т.е. каждое сообщение отправленное
определенным пользователем)
     * Сохранение происходить с помощью сериализации.
     * @param pathMessage Путь к файлу где будут храниться все сообщения
     * @param pathClientMsg Путь к файлу где будут храниться сообщение каждого пользователя
     * @throws IOException Вызывает если не получилось открыть поток с файлом или не удалось
записать данные в файл.
     */
    public void save(String pathMessage,String pathClientMsg) throws IOException {
        saveMessage(pathMessage);
        saveClientMsg(pathClientMsg);
    }
    /**
```

```
    * Метод загружает хранилище из файла. В методе вызываются методы
    * loadMessage - загружает всю историю сообщений из файл.
    * loadClientMsg - загружает сообщение каждого пользователя(т.е. каждое сообщение отправленное
определенным пользователем) из файла
    * loadClient - загружает клиентов из файла в коллекцию {@link ChatStoreg#clients}. Формат записи в
файле id:login:password
    * Загрузка происходить с помощью сериализации.
    * @param pathMessage Путь к файлу где будут храняться все сообщения
    * @param pathClientMsg Путь к файлу где будут храняться сообщение каждого пользователя
    * @param pathClient Путь к файлу где будут храняться клиенты.
    * @throws IOException Вызывает если не получилось открыть поток с файлом или не удалось
записать данные в файл.
    * @throws ClassNotFoundException вызываеться если не удалось считать данные с файла.
    */
   public void load(String pathMessage,String pathClientMsg,String pathClient) throws IOException,
ClassNotFoundException {
       loadClients(pathClient);
       loadMessage(pathMessage);
       loadClientMsg(pathClientMsg);
   }

   /**
    * Ищет клиента типа {@link Client} по его логину в коллекции.
    * @param login логин по которому нужно искать клиента
    * @return Если клиент найден вернет клиента, иначе null
    */
   public Client findClientByLogin(String login){
       for (Client c : clients.values()){
           if (c.getLogin().equals(login)) return c;
       }
       return null;
   }
   /**
    * Ищет в коллекции логин клиента по его id
    * @param id id по которому нужно искать
    * @return Вернет логин
    */
   public String findLoginClientById(int id){
       return clients.get(id).getLogin();
   }
   /**
    * Ищет клиентов типа {@link Client} которые онлайн в коллекции.
    * @return Вернет ArrayList клиентов которые в сети.
    */
   public ArrayList<String> findAllLoginIsOnline(){
       ArrayList<String> clientIsOnline = new ArrayList<>();
       for (Client c: clients.values())
           if (c.isOnline()) clientIsOnline.add(c.getLogin());
       return clientIsOnline;
   }

   /**
    * Достать клиента по его id
    * @param id id по которому нужно достать
    * @return Вернет клиента типа {@link Client}
    */
   public Client getClient(int id){
       return clients.get(id);
   }

   /**
    * Получить коллекцию всех сообщений.
    * @return Вернет коллекцию всех сообщений.
```

```
    */
   public ArrayList<Message> getMessage(){
      return message;
   }

}
```

PersoneMessage

```
package model;

import java.io.Serializable;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Класс который определяет личные сообщения. Наследует класс Message {@link Message}
 * Отличие от других классов Message тут есть поле которые хранит в себе получателя.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class PersonMessage extends Message implements Serializable {
   /** Поле получатель. Тип {@link Client}*/
   protected Client recipient;
   /**
    * Конструктор - создание объекта данного класса
    * Вызывает конструкот базового класса {@link Message#Message(int, String, Client, Date)},
    * а также заполняет поле {@link PersonMessage#recipient}     *
    * @param id уникальный идентификатор
    * @param message сообщение
    * @param sender отправитель
    * @param recipient получатель
    * @param date время отправление
    */
   public PersonMessage(int id, String message, Client sender, Client recipient, Date date){
      super(id,message,sender,date);
      this.recipient = recipient;
   }

   /**
    * Формирует строку для вывода информации об объекте.
    * Вызывает мето {@link Message#getInfo()} и добавляет к нему поле получателя {@link
PersonMessage#recipient}
    * @return Вернет оформленную строку.
    */
   public String getInfo(){
      return super.getInfo()+ " recipient: " + recipient;
   }

   /**
    * Формирует строку для отправки ее на клиент.
    * Переопределяет метод {@link Message#sendMessage()}
    * @return Вернет оформленную строку
    */
   public String sendMessage(){
      DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
      return  dateFormat.format(date) +" " + sender.getLogin() + " -> "+recipient.getLogin()+": " + message;
   }
   /**
    * Получить поле {@link PersonMessage#recipient}
    * @return Вернет поле {@link PersonMessage#recipient}. Тип {@link Client}
```

```
     */
    public Client getRecipient() {
        return recipient;
    }
}
```

StringParser

```
package parsers;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;

/**
 * Парсер строки которая приходит с клиента.
 * Парсер разбивает строку на тип задачи и пременную по типу ключ значение.
 * Строку он заносить в HashMap, где ключ типа String и его значение тоже String.
 * В роли значение может быть массив.
 * Чтобы получить значение переменной нужно вызвать метод {@link StringParser#getProperty(String)}
куда стоит
 * передать ключ(т.е название переменной).
 * Если переменная являеться массивом то нужно вызвать медот {@link
StringParser#getPropertys(String)} куда также стоит
 * передать ключ.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class StringParser {
    /** Коллекция которая хранит все переменные и ее значения */
    private static HashMap<String, ArrayList<String>> properties;

    /**
     * Enum, который хранит тип задачи <br>
     * LOGIN - проверка при подключение <br>
     * DISCONNECT - отключение пользователя <br>
     * SUBMIT_MESSAGE - отправка сообщений на клиент <br>
     * GET_MESSAGE - отправка истории сообщения <br>
     */
    public enum type {LOGIN,DISCONNECT,SUBMIT_MESSAGE, GET_MESSAGES}

    /**
     * Очищение хранилища переменных
     */
    public static void flush(){
        properties.clear();
    }

    /**
     * Метод который по ключу достает массивы значений
     * @param key ключ
     * @return Вернет ArrayList значений
     */
    public static ArrayList<String> getPropertys(String key){
        if (key == null) throw new IllegalArgumentException("Чтобы достать свойсво нужно передать
строку!");
        return properties.get(key);
    }

    /**
     * Метод который по ключи вернет значение
     * @param key ключ
```

```
     * @return Вернет значение тип String
     */
    public static String getProperty(String key) {
        if (key == null) throw new IllegalArgumentException("Чтобы достать свойсво нужно передать строку!");
        if(properties.get(key) == null) return null;
        if (properties.get(key).size() > 1) throw new IllegalArgumentException("Это массив элементов");
        return (properties.get(key)).get(0);
    }

    /**
     * Разбите строки и занесение ключей и значений в {@link StringParser#properties}<br>
     * Спец символы:<br>
     *  "#;" - разделитель для переменных или типа задачи.<br>
     *  "#:" - разделяет переменную и ее значение<br>
     *  "#,"- разделить для значения типа массива, перечесление элементов массива<br>
     * Как должна выглядить передоваемся строка:<br>
     *  [тип задачи]#;[имя переменной#:значение переменной]#; перменных может быть не ограниченое кол-во.<br>
     * Пример:<br>
     *  LOGIN#;msg#:LOGINFAIL<br>
     *  IS_ONLINE#;msg#:Пользователи онлайн#;client#: Efim#,Dima#,Lexa<br>
     * @param str Строка которую нужно запарсить.
     */
    public static void parsingString(String str){
        if (str == null) throw new IllegalArgumentException("В парсер стоит передать строку!");
        properties = new HashMap<>();
        String[] buff = str.split("#;");
        for (String s : buff){
            ArrayList<String> arr = new ArrayList<>();
            if (!s.contains("#:")){
                arr.add(s);
                properties.put("type",arr);
            }
            else {
                String[] buff2 = s.split("#:");
                for (int i = 0; i < buff2.length; i+=2) {
                    if (buff2[i+1].contains("#,"))
                        arr.addAll(Arrays.asList(buff2[i+1].split("#,")));
                    else
                        arr.add(buff2[i+1]);
                    properties.put(buff2[i],arr);
                }
            }
        }
    }

    /**
     * Выводит в консоль читаемую информацию всех ключей и их значений
     */
    public static void info(){
        for (String s: properties.keySet()){
            System.out.print(s+": ");
            for (String s1 : properties.get(s))
                System.out.print(" "+s1);
            System.out.print("\n");
        }
    }
}
```

network\Client

```java
package network;


import controller.GeneralController;
import controller.ServerFormListiner;
import parsers.StringParser;

import java.io.*;
import java.net.Socket;
import java.net.SocketException;

/**
 * Серверны клиент. Принимает сообщение которые пришли с клиента и отпраляет {@link
GeneralController} для обработки.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class Client extends Thread{
    /** Сокет */
    private Socket s;
    /** id пользователя подключеного к этому сокету */
    private int clientId;
    /** Поток получение данных с клиента */
    private BufferedReader in;
    /** Поток отправки данных на клиента */
    private BufferedWriter out;
    /** Флаг для выхода из из потока клиента */
    private boolean exit = true;

    /**
     * Конструктор создание объекта {@link Client}
     * @param s сокет
     * @throws IOException Вызывается если что-то случилось с потоком передачи данных
     */
    public Client(Socket s) throws IOException {
        this.s = s;
        in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
        connect();
    }
    /** Подключение отдельного потока для пользователя */
    public void connect(){
        start();
    }
    @Override
    /**
     * Выполнение потока. Обработка данных которые приходят с клиента.
     */
    public void run() {
        try {
            String msg;
            while (exit) {
                msg = in.readLine();
                if (msg == null){
                    exit = false;
                    break;
                }
                else
                    StringParser.parsingString(msg);
                switch (StringParser.type.valueOf(StringParser.getProperty("type"))){
                    case LOGIN:
                        GeneralController.login(this);
                        break;
```

```
                case SUBMIT_MESSAGE:
                    GeneralController.submitMessage(clientId,StringParser.getProperty("msg"),
StringParser.getProperty("whom"));
                    break;
                case GET_MESSAGES:
                    GeneralController.getMessages(this);
                    break;
                case DISCONNECT:
                    exit = false;
                    GeneralController.disconnect(GeneralController.findLoginClientById(clientId),clientId);
                    break;
            }
            StringParser.flush();
        }
        s.close();
    } catch (SocketException e) {
        StringParser.flush();
    } catch (IOException exception) {
        ServerFormListiner.errorMes(exception.getMessage());
    }
}

/**
 * Отключение сокета.
 * @throws IOException Срабатывает если не получилось отключить сокет.
 */
public void disconnect() throws IOException {
    s.close();
}

/**
 * Отправить сообщение по сокету.
 * @param str сообщение
 */
public void sendMsg(String str){
    try {
        out.write(str+"\n");
        out.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Получение поля {@link Client#clientId}
 * @return Вернет id {@link Client}
 */
public int getClientId() {
    return clientId;
}

/**
 * Заменять или добавить в поле {@link Client#clientId}
 * @param clientId id клиента, кому пренадлежит этот сокет
 */
public void setClientId(int clientId) {
    this.clientId = clientId;
}

/**
 * Изменяет флаг для выхода из потока
 * @param exit занчение которое нужно проставить
 */
```

```
    public void setExit(boolean exit) {
        this.exit = exit;
    }
}
```

model\Client

```
package model;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * Класс реализуеший модель клиента.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class Client implements Serializable {
    /** Уникальный идентификатор клиента */
    private int id;
    /** Уникальный логин клиента */
    private String login;
    /** Пароль клиента */
    private String password;
    /** Хранилище сообщений определнного клиента */
    private ArrayList<Message> messages;
    /** Поле показывающие в сети ли клиент */
    private boolean isOnline = false;

    /**
     *  Конструктор - создание нового объекта клиента.
     *  @param id id клиента
     *  @param login логин клиента
     *  @param password пароль клиента
     */
    public Client (int id, String login, String password){
        this.id = id;
        this.login = login;
        this.password = password;
        messages = new ArrayList<>();
    }
    /**
     * Проверка для авторизации пользователя. Проверяет совпадение логина и пароля.
     * @param login логин клиента
     * @param password пароль клиента
     * @return Вернет id клиента, если пользователь не в сети и пароль и логин верны. Иначе вернет -1.
     */
    public int login(String login,String password){
        if(!isOnline)
            if (this.login.equals(login))
                if (this.password.equals(password)){
                    isOnline = true;
                    return id;
                }
        return -1;
    }

    /**
     * Добавление сообщение в коллекцию сообщений клиента
     * @param message сообщение которое будет добавленно в коллекцию
```

```
     */
    public void addMessage(Message message){
        messages.add(message);
    }

    /**
     * Вывести информацию о полях клиента
     * @return Вернет строку со всеме полями (оформленную)
     */
    public String getInfo(){
        return "id: "+ id +" login: " + login + " password: "+ password;
    }

    /**
     * Получение значение поля {@link Client#id}
     * @return вернет id клиента
     */
    public int getId() {
        return id;
    }

    /**
     * Получение значение поля {@link Client#isOnline}
     * @return вернет true если пользователь в сети, иначе false
     */
    public boolean isOnline() {
        return isOnline;
    }

    /**
     * Замена поля {@link Client#isOnline}
     * @param online значение на которое будет заменяться
     */
    public void setOnline(boolean online) {
        isOnline = online;
    }
    /**
     * Получение значение поля {@link Client#login}
     * @return вернет логин пользователя
     */
    public String getLogin() {
        return login;
    }
    /**
     * Получение значение поля {@link Client#messages}
     * @return вернет коллекцию всех сообщений пользователя
     */
    public ArrayList<Message> getMessages() {
        return messages;
    }

    /**
     * Заменяет поля {@link Client#messages}
     * @param messages коллекция на которую нужно заменять
     */
    public void setMessages(ArrayList<Message> messages){
        this.messages = messages;
    }
}
```

Message

```java
package model;

import java.io.Serializable;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Класс, который отвечает за сообщение. В себе он хранит всю информацию о сообщение, такие как
 * уникальны id, рремя когда было отправленно сообщение, само сообщение, и кто откравил
(отправителем считается клиент
 * {@link Client})
 * Являтся абстрактным.
 * @author Зиняков Ефим
 * @version 1.0
 */
public abstract class Message implements Serializable {
    /** Уникальный идентификатор клиента */
    protected int id;
    /** Время когда было отправленно сообщение */
    protected Date date;
    /** Сообщение */
    protected String message;
    /** Отправитель. Объект класса  */
    protected Client sender;
    /**
     * Конструктор - создает объект данного класса.
     * @param id уникальный идентификатор
     * @param message сообщение
     * @param sender отправитель
     * @param date время отправление
     */
    public Message(int id, String message, Client sender,Date date){
        this.id = id;
        this.message = message;
        this.sender = sender;
        this.date = date;
    }
    /**
     * Мутод, который дает информацию о сообщение (оформленную)
     * @return Вернет строку оформленную строку
     */
    public String getInfo(){
        DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
        return "id: " + id + " date: " + dateFormat.format(date) + " message: " + message + " sender: " +
sender;
    }
    /**
     * Метод который формирует строку для отправки ее на клиент
     * @return Вернет оформленную строку
     */
    public String sendMessage(){
        DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
        return dateFormat.format(date) +" " + sender.getLogin() + ": " + message;
    }
    /**
     * Вернет поле отправителя {@link Message#sender}
     * @return Вернет поле отправителя {@link Message#sender}
     */
    public Client getSender() {
        return sender;
    }
}
```

PersoneMessage

```java
package model;

import java.io.Serializable;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Класс который определяет личные сообщения. Наследует класс Message {@link Message}
 * Отличие от других классов Message тут есть поле которые хранит в себе получателя.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class PersonMessage extends Message implements Serializable {
    /** Поле получатель. Тип {@link Client}*/
    protected Client recipient;
    /**
     * Конструктор - создание объекта данного класса
     * Вызывает конструкот базового класса {@link Message#Message(int, String, Client, Date)},
     * а также заполняет поле {@link PersonMessage#recipient}     *
     * @param id уникальный идентификатор
     * @param message сообщение
     * @param sender отправитель
     * @param recipient получатель
     * @param date время отправление
     */
    public PersonMessage(int id, String message, Client sender, Client recipient, Date date){
        super(id,message,sender,date);
        this.recipient = recipient;
    }

    /**
     * Формирует строку для вывода информации об объекте.
     * Вызывает мето {@link Message#getInfo()} и добавляет к нему поле получателя {@link
PersonMessage#recipient}
     * @return Вернет оформленную строку.
     */
    public String getInfo(){
        return super.getInfo()+ " recipient: " + recipient;
    }

    /**
     * Формирует строку для отправки ее на клиент.
     * Переопределяет метод {@link Message#sendMessage()}
     * @return Вернет оформленную строку
     */
    public String sendMessage(){
        DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
        return  dateFormat.format(date) +" " + sender.getLogin() + " -> "+recipient.getLogin()+": " + message;
    }
    /**
     * Получить поле {@link PersonMessage#recipient}
     * @return Вернет поле {@link PersonMessage#recipient}. Тип {@link Client}
     */
    public Client getRecipient() {
        return recipient;
    }
}
```

CommonMessage

```
package model;

import java.io.Serializable;
import java.util.Date;

/**
 * Класс который определяет сообщение для всех. Наследует класс Message {@link Message} *
 * @author Зиняков Ефим
 * @version 1.0
 */
public class CommonMessage extends Message implements Serializable {
    /**
     * Конструктор - создание объекта данного класса
     * Вызывает конструкот базового класса {@link Message#Message(int, String, Client, Date)}
     * @param id уникальный идентификатор
     * @param message сообщение
     * @param sender отправитель
     * @param date время отправление
     */
    public CommonMessage(int id, String message, Client sender, Date date){
        super(id,message,sender,date);
    }

}
```

Start

```
package debug;

import controller.GeneralController;

public class Start {

    public static void main(String[] arg){
        GeneralController.startApp();
    }
}
```

Клиент:

```
package network;


import controller.GeneralController;
import controller.ClientFormListiner;
import parser.StringParser;

import java.io.*;
import java.net.Socket;
/**
 * Клиент. Принимает сообщение которые пришли с сервера и отпраляет в {@link GeneralController}
для обработки.
 * @author Зиняков Ефим
 * @version 1.0
 */
```

```
public class Client extends Thread{
    private String host ;
    private int port ;
    /** Сокет */
    private Socket s;
    /** Поток получение данных с клиента */
    private BufferedReader in;
    /** Поток отправки данных на клиента */
    private BufferedWriter out;
    /** Флаг для выхода из из потока клиента */
    private boolean exit = true;
    /**
     * Конструктор создание объекта
     * @param host host по которому нужно подключиться
     * @param port port по которому нужно подлючиться
     * @throws IOException Вызывается если что-то случилось с потоком передачи данных
     */
    public Client(String host,int port) throws IOException {
        this.host = host;
        this.port = port;
        s = new Socket(host,port);
        in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
        connect();
    }
    @Override
    /**
     * Выполнение потока. Обработка данных которые приходят с клиента.
     */
    public void run() {
        sendMsg("LOGIN#;login#:"+ GeneralController.view.getTextFieldLogin().getText()+"#;password#:"+
                GeneralController.view.getTextFieldPassword().getText());
        while (exit){
            try {
                String msg;
                while (exit) {
                    msg = in.readLine();
                    if (msg == null){
                        exit = false;
                        break;
                    }
                    else
                        StringParser.parsingString(msg);
                    switch (StringParser.type.valueOf(StringParser.getProperty("type"))){
                        case LOGIN:
                            GeneralController.checkConnect(StringParser.getProperty("msg"));
                            break;
                        case STOP_SERVER:
                            GeneralController.stopServer(this);
                            break;
                        case DISCONNECT:
                            ClientFormListiner.removeUser(StringParser.getProperty("client"));
                            break;
                        case IS_ONLINE:
                            GeneralController.view.sendMes(StringParser.getProperty("msg"));
                            GeneralController.setOnline();
                            break;
                        case SUBMIT_MESSAGE:
                            GeneralController.view.sendMesNoTime(StringParser.getProperty("msg"));
                            break;
                        case MESSAGE:
                            GeneralController.setMessageHistory();
                            break;
```

```
                    case BASE:
                        GeneralController.view.sendMes(StringParser.getProperty("msg"));
                        break;
                }
                StringParser.flush();
            }
            s.close();
        } catch (IOException exception) {
            ClientFormListiner.errorMes(exception.getMessage());
        }
    }
}
/** Подключение отдельного потока для пользователя */
public void connect() throws IOException{
    start();
}
/**
 * Отправить сообщение по сокету.
 * @param str сообщение
 */
public void sendMsg(String str){
    try {
        out.write(str+"\n");
        out.flush();
    } catch (IOException e) {
        ClientFormListiner.errorMes(e.getMessage());
    }
}
/**
 * Изменяет флаг для выхода из потока
 * @param exit занчение которое нужно проставить
 */
public void setExit(boolean exit) {
    this.exit = exit;
}
}
```

View

```
package view;

import controller.GeneralController;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

/**
 * Класс является интерфейсом программы. Реализующий паттерн проектирование SINGLETON
```

```
 * В себе содержит компонеты интерфейса
 * @author Ванцын Дмитрий
 * @version 1.0
 */
public class View extends JFrame {
    /** Панель, содержащая в себе панели формы */
    private JPanel mainPanel = new JPanel();
    /** Панель, содержащая в себе компонеты для подключения к северу */
    private JPanel connectPanel = new JPanel();
    /** Панель, содержащая в себе кнопки подключиться, отключиться и выход */
    private JPanel buttonPanel = new JPanel();
    /** Панель, содержащая в себе текстовую область, в которую будут выводиться сообщения */
    private JPanel chatPanel = new JPanel();
    /** Панель, содержащая в себе компоненты для отправки сообщений, а так же для просмотра
истории сообщений и пользователей онлайн */
    private JPanel enterPanel = new JPanel();
    /** Надпись, показывающая, куда вводить порт */
    private JLabel labelPort = new JLabel("Порт:");
    /** Текстовая панель, для ввода порта */
    private JTextField textFieldPort = new JTextField();
    /** Надпись, показывающая, куда вводить хост */
    private JLabel labelHost = new JLabel("Хост:");
    /** Текстовая панель, для ввода хоста */
    private JTextField textFieldHost = new JTextField();
    /** Надпись, показывающая, куда вводить логин */
    private JLabel labelLogin = new JLabel("Логин:");
    /** Текстовая панель, для ввода логина */
    private JTextField textFieldLogin = new JTextField();
    /** Надпись, показывающая, куда вводить пароль */
    private JLabel labelPassword = new JLabel("Пароль:");
    /** Текстовая панель, для ввода пароля */
    private JTextField textFieldPassword= new JTextField();
    /** Кнопка подключиться к серверу */
    private JButton buttonConnect = new JButton("Подключиться");
    /** Кнопка отключиться от сервера */
    private JButton buttonDisconnect = new JButton("Отключиться");
    /** Кнопка выход из приложения */
    private JButton buttonExit = new JButton("Выход");
    /** Текстовая область, в которую будут выводиться сообщения */
    private JTextArea chat = new JTextArea();
    /** Поле, прикрепляющие к текстовой области скрол */
    private JScrollPane scroll = new JScrollPane(chat, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
    /** Поле, в которое надо ввести сообщение */
    private JTextField textFieldEnter = new JTextField("Введите сообщение");
    /** Кнопка отправить сообщение */
    private JButton buttonEnter = new JButton("Отправить");
    /** Поле для начальной инициализации поля выбора, кому отправить сообщение */
    String[] users = {"Всем"};
    /** Поле для выбора, кому отправить сообщение */
    private JComboBox <String> selectUsers = new JComboBox<String>(users);
    /** Кнорка, для просмотра истории переписки */
    private JButton buttonGetAllMessage = new JButton("История переписки");
    /** Кнопка для просмотра пользователей, которые подключены к сереверу */
    private JButton buttonIsOnline = new JButton("Пользователи онлайн");
    /** Поле для реализации паттерна SINGLETON */
    private static View view = null;

    /**
     * Приватный конструктор для реализции SINGLETON.
     * @param title задаёт имя формы
     * @exception HeadlessException Вызывается, когда код, зависящий от клавиатуры, дисплея или мыши,
вызывается в среде, не поддерживающей клавиатуру, дисплей или мышь.
```

```
     * Для создание объект нужно вызывать метод {@link View#getView()}
     */
    private View(String title) throws HeadlessException {
        super(title);
        setSize(1000, 720);
        setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
        getRootPane().setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));
        init();
    }

    /**
     * Метод нужен для создание объекта данного класса.
     * Или же если объект создан вернёт ссылку на него
     * @return Вернет созданный объект {@link View}
     */
    public static View getView (){
        if(view == null)
            view = new View("Чат");
        return view;
    }

    /** Задаёт положение всех компонентов, располагающихся на форме */
    public void init (){
        textFieldHost.setText(GeneralController.properties.getProperty("host"));
        textFieldPort.setText(GeneralController.properties.getProperty("port"));
        mainPanel.setLayout(new BorderLayout());
        connectPanel.setLayout(new BoxLayout(connectPanel, BoxLayout.X_AXIS));
        labelHost.setFont(new Font(null, Font.BOLD, 13));
        connectPanel.add(Box.createHorizontalStrut(7));
        connectPanel.add(labelHost);
        connectPanel.add(Box.createHorizontalStrut(7));
        textFieldHost.setPreferredSize(new Dimension(70,10));
        connectPanel.add(textFieldHost);
        connectPanel.add(Box.createHorizontalStrut(7));
        labelPort.setFont(new Font(null, Font.BOLD, 13));
        connectPanel.add(labelPort);
        connectPanel.add(Box.createHorizontalStrut(7));
        textFieldPort.setPreferredSize(new Dimension(40,10));
        connectPanel.add(textFieldPort);
        connectPanel.add(Box.createHorizontalStrut(7));
        labelLogin.setFont(new Font(null, Font.BOLD, 13));
        connectPanel.add(labelLogin);
        connectPanel.add(Box.createHorizontalStrut(7));
        textFieldLogin.setPreferredSize(new Dimension(70,10));
        connectPanel.add(textFieldLogin);
        labelPassword.setFont(new Font(null, Font.BOLD, 13));
        labelPassword.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
        connectPanel.add(labelPassword);
        textFieldPassword.setPreferredSize(new Dimension(70,10));
        connectPanel.add(textFieldPassword);
        buttonPanel.setLayout(new GridLayout(1,3,10,10));
        buttonPanel.setBorder(new EmptyBorder(0,10,0,0));
        buttonPanel.add(buttonConnect);
        buttonPanel.add(buttonDisconnect);
        buttonPanel.add(buttonExit);
        connectPanel.add(buttonPanel);
        connectPanel.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        mainPanel.add(connectPanel, BorderLayout.NORTH);

        chatPanel.setLayout(new GridLayout());
        mainPanel.add(chatPanel, BorderLayout.CENTER);

        chat.setEditable(false);
```

```
        chatPanel.add(scroll);
        textFieldEnter.setFont(new Font(null, Font.BOLD, 14));
        enterPanel.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.HORIZONTAL;
        c.weightx = 10;
        c.gridx = 0;
        c.gridy = 0;
        c.insets = new Insets(5,7,0,7);
        enterPanel.add(textFieldEnter, c);

        c.fill = GridBagConstraints.CENTER;
        c.weightx = 1;
        c.insets = new Insets(10,0,5,5);
        c.gridx = 1;
        c.gridy = 0;
        enterPanel.add(buttonEnter, c);

        c.fill = GridBagConstraints.PAGE_END;
        c.weightx = 1;
        c.insets = new Insets(10,0,5,5);
        c.gridx = 2;
        c.gridy = 0;
        enterPanel.add(buttonIsOnline, c);

        selectUsers.setBackground(Color.CYAN);
        selectUsers.setFont(new Font(null, Font.BOLD, 13));
        c.fill = GridBagConstraints.HORIZONTAL;
        c.insets = new Insets(5,7,10,7);
        c.gridx = 0;

        c.gridy = 1;
        enterPanel.add(selectUsers, c);
        c.fill = GridBagConstraints.CENTER;

        c.insets = new Insets(10,0,10,0);
        c.gridx = 1;
        c.gridwidth = 2;
        c.gridy = 1;
        enterPanel.add(buttonGetAllMessage, c);

        mainPanel.add(enterPanel, BorderLayout.SOUTH);

        add(mainPanel);
        disconnect();
        setLocationRelativeTo(null);
        setVisible(true);
    }

    /**
     * Переключает доступности кнопок на форме, если клиент подключился к серверу
     */
    public void connect(){
        buttonExit.setEnabled(false);
        buttonConnect.setEnabled(false);
        buttonDisconnect.setEnabled(true);
        buttonEnter.setEnabled(true);
        textFieldEnter.setEnabled(true);
        textFieldHost.setEnabled(false);
        textFieldLogin.setEnabled(false);
        textFieldPassword.setEnabled(false);
        textFieldPort.setEnabled(false);
```

```
        selectUsers.setEnabled(true);
        buttonGetAllMessage.setEnabled(true);
        buttonIsOnline.setEnabled(true);

    }

    /**
     *Переключает доступность кнопок на форме, если клиент отключился от сервера
     */
    public void disconnect(){
        buttonExit.setEnabled(true);
        buttonConnect.setEnabled(true);
        buttonDisconnect.setEnabled(false);
        buttonEnter.setEnabled(false);
        textFieldEnter.setEnabled(false);
        textFieldHost.setEnabled(true);
        textFieldLogin.setEnabled(true);
        textFieldPassword.setEnabled(true);
        textFieldPort.setEnabled(true);
        selectUsers.setEnabled(false);
        buttonGetAllMessage.setEnabled(false);
        buttonIsOnline.setEnabled(false);
    }

    /**
     * Функция, которая выводит заданное сообщение в форму, указывая время, в которое оно было
отправлено
     * @param message сообщение, которое надо вывыести в форму
     */
    public void sendMes (String message){
        Date currentDate = new Date();
        DateFormat timeFormat = new SimpleDateFormat("HH:mm:ss", Locale.getDefault());
        String time = timeFormat.format(currentDate);
        chat.append(time + " " + message + "\n");
    }

    /**
     * Функция, которая выводит заданное сообщение в форму, не указывая время, в которое оно было
отправлено
     * @param message сообщение, которое надо вывыести в форму
     */
    public void sendMesNoTime(String message){
        message = message.replaceAll("#n", "\n");
        chat.append(message + "\n");
    }

    /**
     * Получить доступ к кнопке История сообщений
     * @return вернёт компонент(кнопку История сообщений)
     */
    public JButton getButtonGetAllMessage() {
        return buttonGetAllMessage;
    }

    /**
     * Получить доступ к текстовому полю, в котором указан порт для подключения к серверу
     * @return вернёт компонент(текстовое поле с портом подключения к сереверу)
     */
    public JTextField getTextFieldPort() {
        return textFieldPort;
    }

    /**
```

```
     * Получить доступ к текстовому полю, в котором указан хост для подключения к серверу
     * @return вернёт компонент(текстовое поле с хостом подключения к сереверу)
     */
    public JTextField getTextFieldHost() {
        return textFieldHost;
    }
    /**
     * Получить доступ к текстовому полю, в котором указан логин для подключения к серверу
     * @return вернёт компонент(текстовое поле с логином подключения к сереверу)
     */
    public JTextField getTextFieldLogin() {
        return textFieldLogin;
    }
    /**
     * Получить доступ к текстовому полю, в котором указан пароль для подключения к серверу
     * @return вернёт компонент(текстовое поле с паролем подключения к сереверу)
     */
    public JTextField getTextFieldPassword() {
        return textFieldPassword;
    }
    /**
     * Получить доступ к кнопке подключиться
     * @return вернёт компонент(кнопку подключиться)
     */
    public JButton getButtonConnect() {
        return buttonConnect;
    }
    /**
     * Получить доступ к кнопке отключится
     * @return вернёт компонент(кнопку отключиться)
     */
    public JButton getButtonDisconnect() {
        return buttonDisconnect;
    }
    /**
     * Получить доступ к кнопке выход
     * @return вернёт компонент(кнопку выход)
     */
    public JButton getButtonExit() {
        return buttonExit;
    }
    /**
     * Получить доступ к текстовому полю, в котором указано сообщение
     * @return вернёт компонент(текстовое поле с сообщением)
     */
    public JTextField getTextFieldEnter() {
        return textFieldEnter;
    }
    /**
     * Получить доступ к кнопке отправить сообщение
     * @return вернёт компонент(кнопку отправить сообщение)
     */
    public JButton getButtonEnter() {
        return buttonEnter;
    }
    /**
     * Получить доступ к меню, в котором можно выбрать, кому отправить сообщение
     * @return вернёт компонент(меню, в котором можно выбрать, кому отправить сообщение)
     */
    public JComboBox<String> getSelectUsers() {
        return selectUsers;
    }
    /**
```

```
    * Получить доступ к кнопке посмотрть всех пользователей онлайн
    * @return вернёт компонент(кнопку посмотрть всех пользователей онлайн)
    */
   public JButton getButtonIsOnline() {
      return buttonIsOnline;
   }
}
```

ClientFormListiner

```
package controller;

import view.MessageHistory;
import view.View;

import javax.swing.*;
import java.awt.event.*;
import java.util.ArrayList;

import static java.lang.System.exit;
/**
 * Класс является слушателем компонентов формы.
 * В себе содержит методы, которые будут срабатывать во время действий на форме.
 * @author Ванцын Дмитрий
 * @version 1.0
 */
public class ClientFormListiner implements ActionListener, MouseListener {
   /**
    * Интерфейс программы
    */
   private static View view = null;
   /**
    * Коллекция, содержащая в себе всех пользователей, находящихся онлайн
    */
   private static ArrayList<String> userIsOnline = new ArrayList<>();

   /**
    * Конструктор, который создаёт слушателя компонентов формы, а так же добавляющий этогос
лушателя компонентам формы
    * @param view Интерфейс программы
    */
   public ClientFormListiner(View view) {
      this.view = view;
      view.getButtonExit().addActionListener(this);
      view.getButtonConnect().addActionListener(this);
      view.getButtonDisconnect().addActionListener(this);
      view.getButtonEnter().addActionListener(this);
      view.getButtonGetAllMessage().addActionListener(this);
      view.getTextFieldEnter().addMouseListener(this);
      view.getButtonIsOnline().addActionListener(this);
   }

   /**
    * Унаследованный метод, который реагирует на события на форме
    * @param e События на форме
    */
   @Override
   public void actionPerformed(ActionEvent e) {
      if(e.getSource() == view.getButtonExit()){
         exit(0);
      }
      else if(e.getSource() == view.getButtonDisconnect()){
```

```
            GeneralController.disconnect();
        }
        else if(e.getSource() == view.getButtonConnect()){
            String portStr = view.getTextFieldPort().getText().replaceAll("\\s+","");
            String hostStr = view.getTextFieldHost().getText().replaceAll("\\s+","");
            String login = view.getTextFieldLogin().getText().replaceAll("\\s+","");
            String password = view.getTextFieldPassword().getText().replaceAll("\\s+","");
            if(portStr.isEmpty()){
                view.sendMes("Введите порт");
                return;
            }
            if(hostStr.isEmpty()){
                view.sendMes("Введите хост");
                return;
            }
            if(!portStr.matches("\\d+")){
                view.sendMes("В поле порт записаны не числа");
                return;
            }
            int port = Integer.parseInt(portStr);
            if(port<=0||port>65536) {
                view.sendMes("Значение в поле порт не подходит для подключению к серверу");
                return;
            }
            if(login.isEmpty()){
                view.sendMes("Введите логин");
                return;
            }
            if(password.isEmpty()){
                view.sendMes("Введите пароль");
                return;
            }
            GeneralController.connect(hostStr,port);
        }
        else if(e.getSource() == view.getButtonEnter()){
            String message = view.getTextFieldEnter().getText();
            if(message.isEmpty() || message.equals("Введите сообщение")){
                view.sendMes("Сообщение не введено");
                return;
            }
            int lengthStr = 60;
            int count = message.length()/lengthStr;
            for (int i = 1; i <= count; i++)
                message = new StringBuilder(message).insert(lengthStr*i,"#n").toString();
            GeneralController.enter(message,view.getSelectUsers().getSelectedItem().toString());
            view.getTextFieldEnter().setText("");
        }
        else if(e.getSource() == view.getButtonGetAllMessage()){
            GeneralController.getMessageHistory();
            new MessageHistory("История переписки");
        }
        else if(e.getSource() == view.getButtonIsOnline()){
            if(userIsOnline.size() != 0)
                JOptionPane.showMessageDialog(view, userIsOnline.toArray(new String[0]), "Пользователи
онлайн",
                        JOptionPane.INFORMATION_MESSAGE);
            else JOptionPane.showMessageDialog(view, "Подключённых пользователей нет",
                    "Пользователи онлайн", JOptionPane.INFORMATION_MESSAGE);
        }
    }
    /**
     * Метод, который выводит ошибки в диалоговые окна
     * @param str Строка с ошибкой
```

```
     */
    public static void errorMes (String str){
        JOptionPane.showMessageDialog(view, str,
                "Ошибка", JOptionPane.INFORMATION_MESSAGE);
    }
    /**
     * Добавляет пользователей онлайн, в меню выбора кому отправить сообщения
     * @param user Пользователь, которого надо добавить
     */
    public static void addToUser(String user){
        for (int i = 0; i<view.getSelectUsers().getModel().getSize();i++){
            if(view.getSelectUsers().getModel().getElementAt(i).equals(user))
                return;
        }
        userIsOnline.add(user);
        view.getSelectUsers().addItem(user);
    }
    /**
     * Удаляет пользователей, которые отключились от сервера, из меню выбора кому отправить
сообщения
     * @param user Пользователь, которого надо удалить
     */
    public static void removeUser(String user){
        view.sendMes("Пользователь" + user +"отключился");
        userIsOnline.remove(user);
        view.getSelectUsers().removeItem(user);
    }
    /**
     * Унаследованные методы, которые реагируют на дейтсвия мыши на форме
     * @param e Действия мыши на форме
     */
    @Override
    public void mouseClicked(MouseEvent e) {
        if(e.getComponent()==view.getTextFieldEnter())
            view.getTextFieldEnter().setText("");
    }

    @Override
    public void mousePressed(MouseEvent e) {

    }

    @Override
    public void mouseReleased(MouseEvent e) {

    }

    @Override
    public void mouseEntered(MouseEvent e) {

    }

    @Override
    public void mouseExited(MouseEvent e) {

    }

}
```

MessageHistory

```java
package view;

import parser.StringParser;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Класс является интерфейсом диалогового окна, показывающего историю сообщений.
 * В себе содержит компонеты интерфейса диалогового окна, показывающего историю сообщений.
 * @author Ванцын Дмитрий
 * @version 1.0
 */
public class MessageHistory extends JFrame {
    /** Панель, содержащая в себе все компоненты формы */
    private JPanel mainPanel = new JPanel();
    /** Поле, содержащая в себе текстовую область, для вывода сообщений */
    private static JTextArea history = new JTextArea();
    /** Поле, добавляющеее к текстовой области, для вывода сообщений скролл */
    private JScrollPane scroll = new JScrollPane(history, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
    /** Кнопка, для выхода из интерфеса диалогового окна */
    private JButton buttonClose = new JButton("Закрыть");

    /**
     * Конструктор, создающий форму, для просмотра истории сообщений
     * @param title задаёт имя формы
     * @exception HeadlessException Вызывается, когда код, зависящий от клавиатуры, дисплея или мыши,
вызывается в среде, не поддерживающей клавиатуру, дисплей или мышь.
     */
    public MessageHistory(String title) throws HeadlessException {
        super(title);
        init();
    }
    /** Задаёт положение всех компонентов, располагающихся на форме */
    public void init(){
        setSize(400, 400);
        mainPanel.setLayout(new BorderLayout());
        mainPanel.add(scroll, BorderLayout.CENTER);
        history.setEditable(false);
        mainPanel.add(buttonClose, BorderLayout.SOUTH);
        buttonClose.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose();
            }
        });
        scroll.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        add(mainPanel);
        setLocationRelativeTo(null);
        setVisible(true);
    }

    /**
     * Изменяет историю сообщений
     * @param str Текст, содержащий историю сообщений
     */
    public static void setHistory(String str) {
        history.setText(str);
    }
}
```

GeneralConroller

```
package controller;

import network.Client;
import parser.StringParser;
import view.MessageHistory;
import view.View;

import javax.swing.*;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
/**
 * Главный контроллер приложения.
 * Связывает все части проекта, и обрабатывает их.
 * @author Ванцын Дмитрий
 * @version 1.0
 */
public class GeneralController {
    /** Путь где хранится файл с настройками */
    public static String FILE_SETTINGS_NAME = "source/config.properties";
    /** Коллекция настроек */
    public static Properties properties = new Properties();
    /** Интерфейс программы */
    public static View view = null;
    /** Слушатель интерфеса программы */
    public static ClientFormListiner kFL = null;
    /** Клиент */
    public static Client client = null;
    /**
     * Запуск всего приложение. Инициализация полей
     */
    public static void startApp (){
        try {
            properties.load(new FileInputStream(FILE_SETTINGS_NAME));
            view = View.getView();
            kFL = new ClientFormListiner(view);
        } catch (IOException exception) {
            ClientFormListiner.errorMes(exception.getMessage());
        }
    }
    /**
     * Обаботчки для кнопки 'Старт' из формы.
     * Задание сервера, загрузка в хранилище из файла.
     * @param port Порт, по которому будет подключаться сервер.
     * @param host Хост, по которому будет подключаться сервер.
     */
    public static void connect(String host,int port){
        try {
            client = new Client(host,port);
        } catch (IOException e) {
            ClientFormListiner.errorMes(e.getMessage());
        }
    }
    /**
     * Отлючение пользователя от сервера
     */
    public static void disconnect(){
        client.sendMsg("DISCONNECT");;
        view.disconnect();
```

```
    }
    /**
     * Отправка сообщений на сервер
     * @param msg сообщение
     * @param whom кому предназначено сообщение
     */
    public static void enter(String msg,String whom){
        client.sendMsg("SUBMIT_MESSAGE#;msg#:"+msg+"#;whom#:" +whom);
    }
    /**
     * Проверка подключения к серверу
     * @param str Строка, показывающая удачность или провальности подключения к серверу
     */
    public static void checkConnect(String str){
        if(str.equals("LOGINSUCCESS")){
            view.connect();
            JOptionPane.showMessageDialog(GeneralController.view, new String[]{
                        "Дражайше приветствуем тебя друже в нашем уютном чатике",
                        "Надеемся тебе понравится у нас", "Чувствуй себя как дома"},
                    "Добро пожаловать", JOptionPane.INFORMATION_MESSAGE);
        }
        else if (str.equals("LOGINFAIL")){
            JOptionPane.showMessageDialog(GeneralController.view,
                    "Не правильный логин или пароль!");
            client.setExit(false);
        }
    }
    /**
     * Изменение пользователей онлайн
     */
    public static void setOnline() {
        for (String s : StringParser.getPropertys("clientIsOnline"))
            ClientFormListiner.addToUser(s);
    }
    /**
     * Изменение истории сообщений
     */
    public static void setMessageHistory(){
        String history = StringParser.getProperty("msg");
        MessageHistory.setHistory(history.replaceAll("#n", "\n"));
    }
    /**
     * Получение пользователей онлайн
     */
    public static void getMessageHistory() {
        client.sendMsg("GET_MESSAGES#;");
    }
    /**
     * Остановка сервера
     */
    public static void stopServer(Client client) {
        client.setExit(false);
        view.disconnect();
        ClientFormListiner.removeUser(GeneralController.view.getTextFieldLogin().getText());
    }
}
```

Main

```
package main;

import controller.GeneralController;
```

```
public class Main {
    public static void main(String[] str){
        GeneralController.startApp();
    }
}
```

StringParser

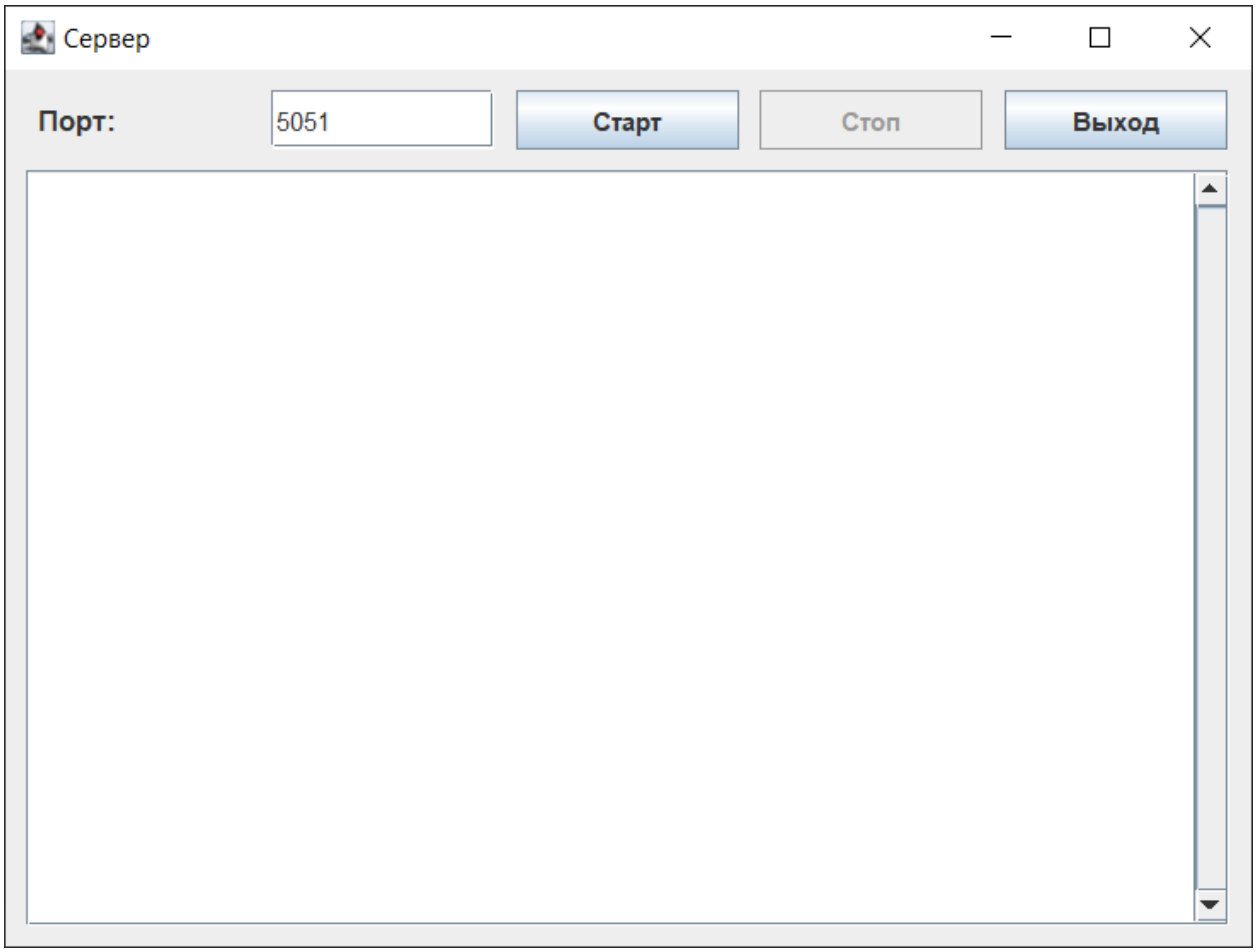
```
package parser;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
/**
 * Парсер строки которая приходит с клиента.
 * Парсер разбивает строку на тип задачи и пременную по типу ключ значение.
 * Строку он заносить в HashMap, где ключ типа String и его значение тоже String.
 * В роли значение может быть массив.
 * Чтобы получить значение переменной нужно вызвать метод {@link StringParser#getProperty(String)}
куда стоит
 * передать ключ(т.е название переменной).
 * Если переменная являеться массивом то нужно вызвать медот {@link
StringParser#getPropertys(String)} куда также стоит
 * передать ключ.
 * @author Зиняков Ефим
 * @version 1.0
 */
public class StringParser {
    /** Коллекция которая хранит все переменные и ее значения */
    private static HashMap<String, ArrayList<String>> properties;
    /**
     * Enum, который хранит тип задачи <br>
     * LOGIN - проверка при подключение <br>
     * DISCONNECT - отключение пользователя <br>
     * SUBMIT_MESSAGE - отправка сообщений на сервер <br>
     * MESSAGE - истории сообщения <br>
     * STOP_SERVER - отключение сервера
     * BASE - вывод сообщение в лог
     * IS_ONLINE - пользователи онлайн
     */
    public enum type {LOGIN,DISCONNECT,IS_ONLINE,STOP_SERVER,SUBMIT_MESSAGE,
MESSAGE,BASE}
    /**
     * Очищение хранилища переменных
     */
    public static void flush(){
        properties.clear();
    }
    /**
     * Метод который по ключу достает массивы значений
     * @param key ключ
     * @return Вернет ArrayList значений
     */
    public static ArrayList<String> getPropertys(String key){
        if (key == null) throw new IllegalArgumentException("Чтобы достать свойсво нужно передать
строку!");
        return properties.get(key);
    }
    /**
     * Метод который по ключи вернет значение
     * @param key ключ
```

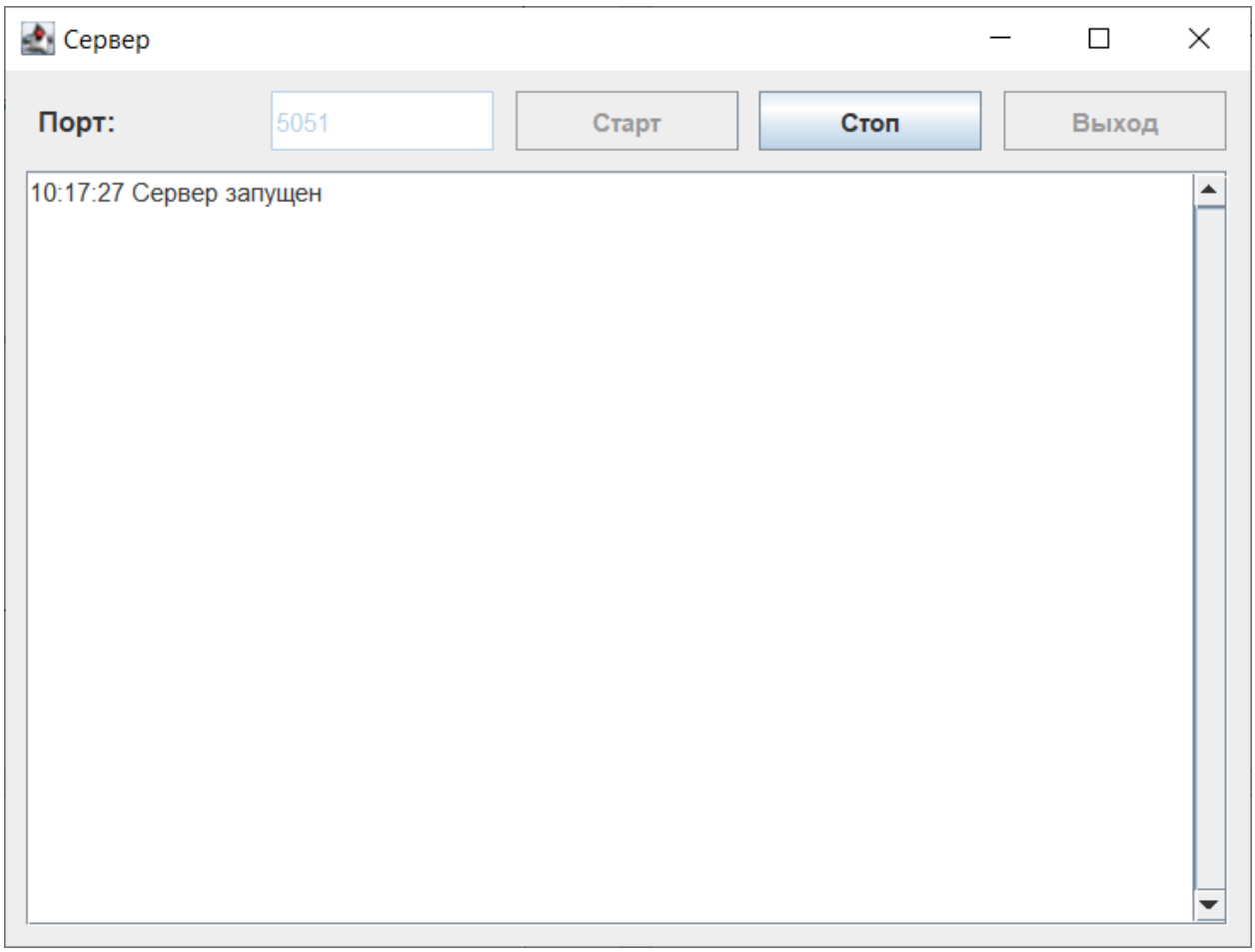
```
     * @return Вернет значение тип String
     */
    public static String getProperty(String key) {
        // Может вернуть просто Стринг.
        // А может вернуть массив стрингов, если под ключем лежит массив.
        if (key == null) throw new IllegalArgumentException("Чтобы достать свойсво нужно передать
строку!");
        if(properties.get(key) == null) return null;
        if (properties.get(key).size() > 1) throw new IllegalArgumentException("Это массив элементов");
        return (properties.get(key)).get(0);
    }
    /**
     * Разбите строки и занесение ключей и значений в {@link StringParser#properties}<br>
     * Спец символы:<br>
     *  "#;" - разделитель для переменных или типа задачи.<br>
     *  "#:" - разделяет переменную и ее значение<br>
     *  "#,"- разделить для значения типа массива, перечесление элементов массива<br>
     * Как должна выглядить передоваемся строка:<br>
     *  [тип задачи]#;[имя переменной#:значение переменной]#; перменных может быть не ограниченое
кол-во.<br>
     * Пример:<br>
     *  LOGIN#;msg#:LOGINFAIL<br>
     *  IS_ONLINE#;msg#:Пользователи онлайн#;client#: Efim#,Dima#,Lexa<br>
     * @param str Строка которую нужно запарсить.
     */
    public static void parsingString(String str){
        //      "type"#;msg#:dfsfs
        //      "type"#;msg#:sdad#;client#: Efim#,Dima#,Lexa
        if (str == null) throw new IllegalArgumentException("В парсер стоит передать строку!");
        properties = new HashMap<>();
        System.out.println(str);
        String[] buff = str.split("#;");
        for (String s : buff){
            ArrayList<String> arr = new ArrayList<>();
            if (!s.contains("#:")){
                arr.add(s);
                properties.put("type",arr);
            }
            else {
                String[] buff2 = s.split("#:");
                for (int i = 0; i < buff2.length; i+=2) {
                    if (buff2[i+1].contains("#,"))
                        arr.addAll(Arrays.asList(buff2[i+1].split("#,")));
                    else
                        arr.add(buff2[i+1]);
                    properties.put(buff2[i],arr);
                }
            }
        }
    }
    /**
     * Выводит в консоль читаемую информацию всех ключей и их значений
     */
    public static void info(){
        for (String s: properties.keySet()){
            System.out.print(s+": ");
            for (String s1 : properties.get(s))
                System.out.print(" "+s1);
            System.out.print("\n");
        }
    }
}
```

Результаты работы программы

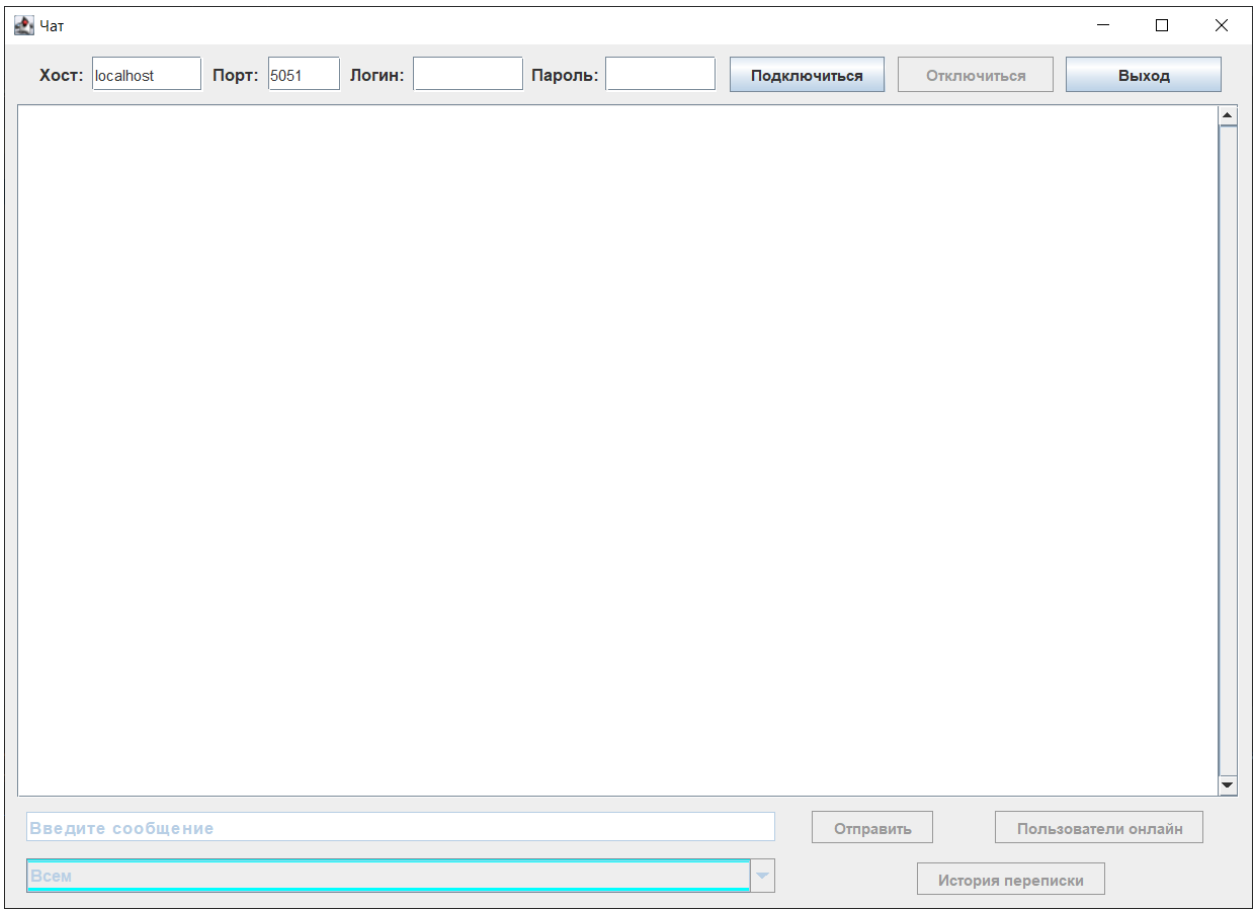
Чтобы пользователь запустил сервер, он должнен задать порт (или оставить стандартный) и нажать кнопку старт



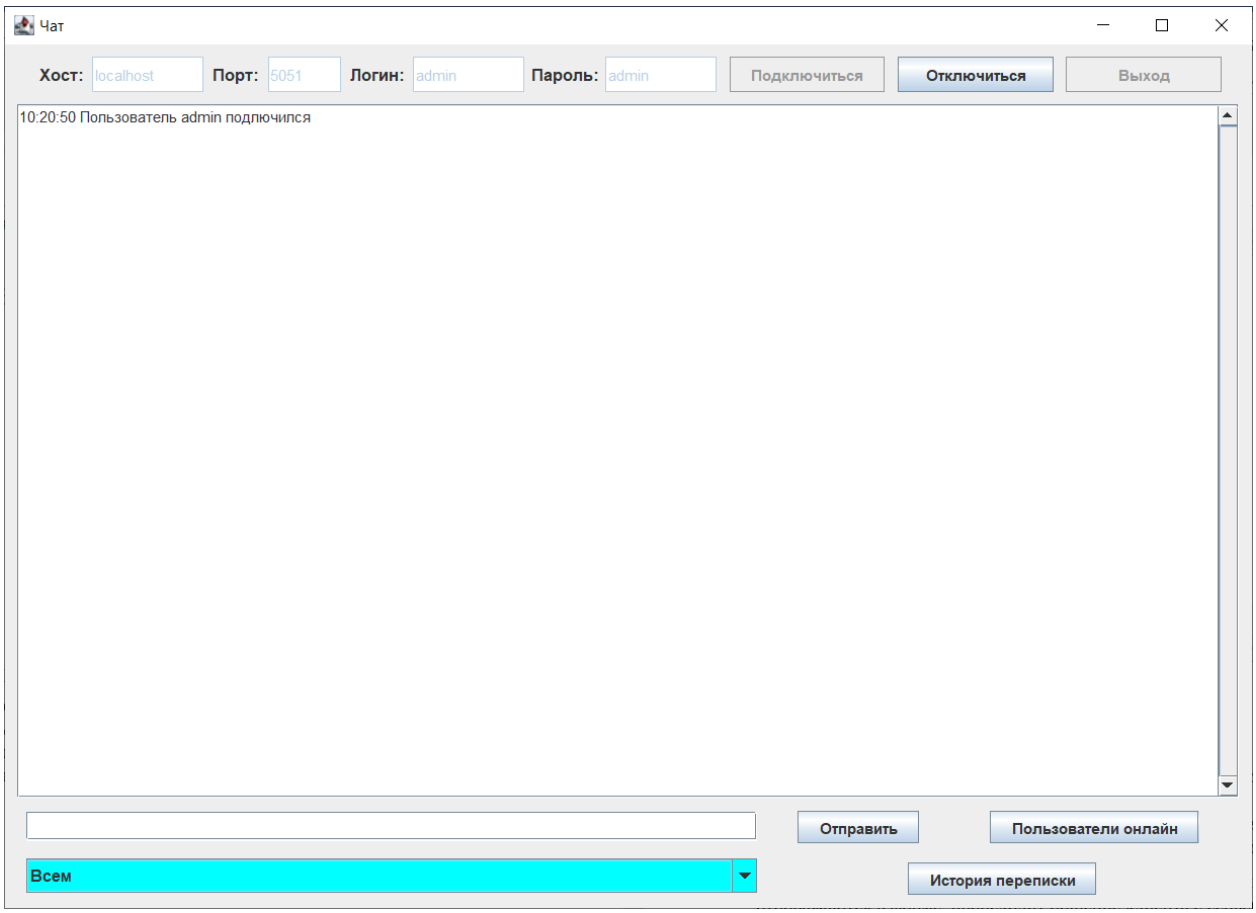
Во включенном состоянии сервер можно остановить и отключить всех клиентов от сервера



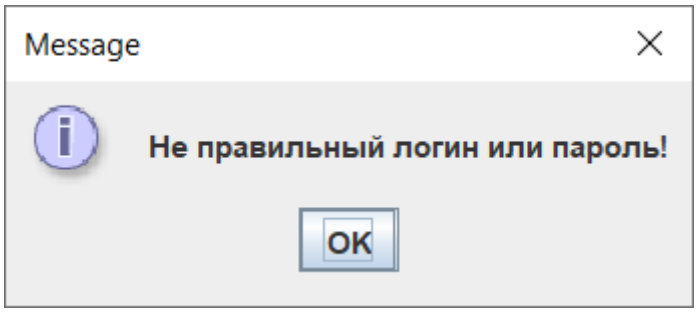
Для подключения к серверу пользователю нужно ввести хост и порт (или оставить их стандартными), а так же логин и пароль, которые должны совпадать с уже имеющимися на сервере. Изначально доступны кнопки подключиться и выход.



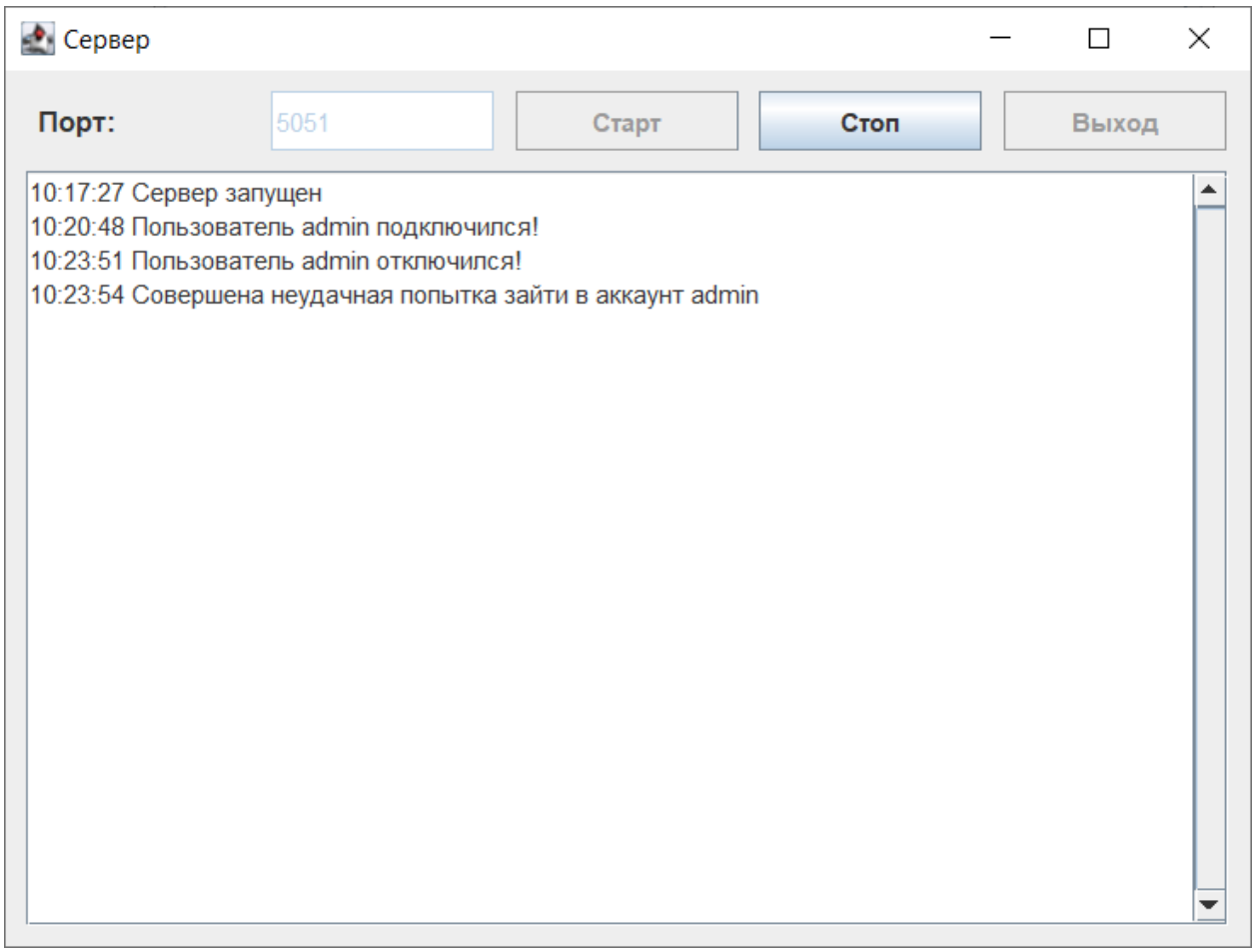
Если все параметры оказались верны, то при нажатии на кнопку подключиться открывается доступ к функционалу чата. При нажатии кнопки выход приложение закрывается.



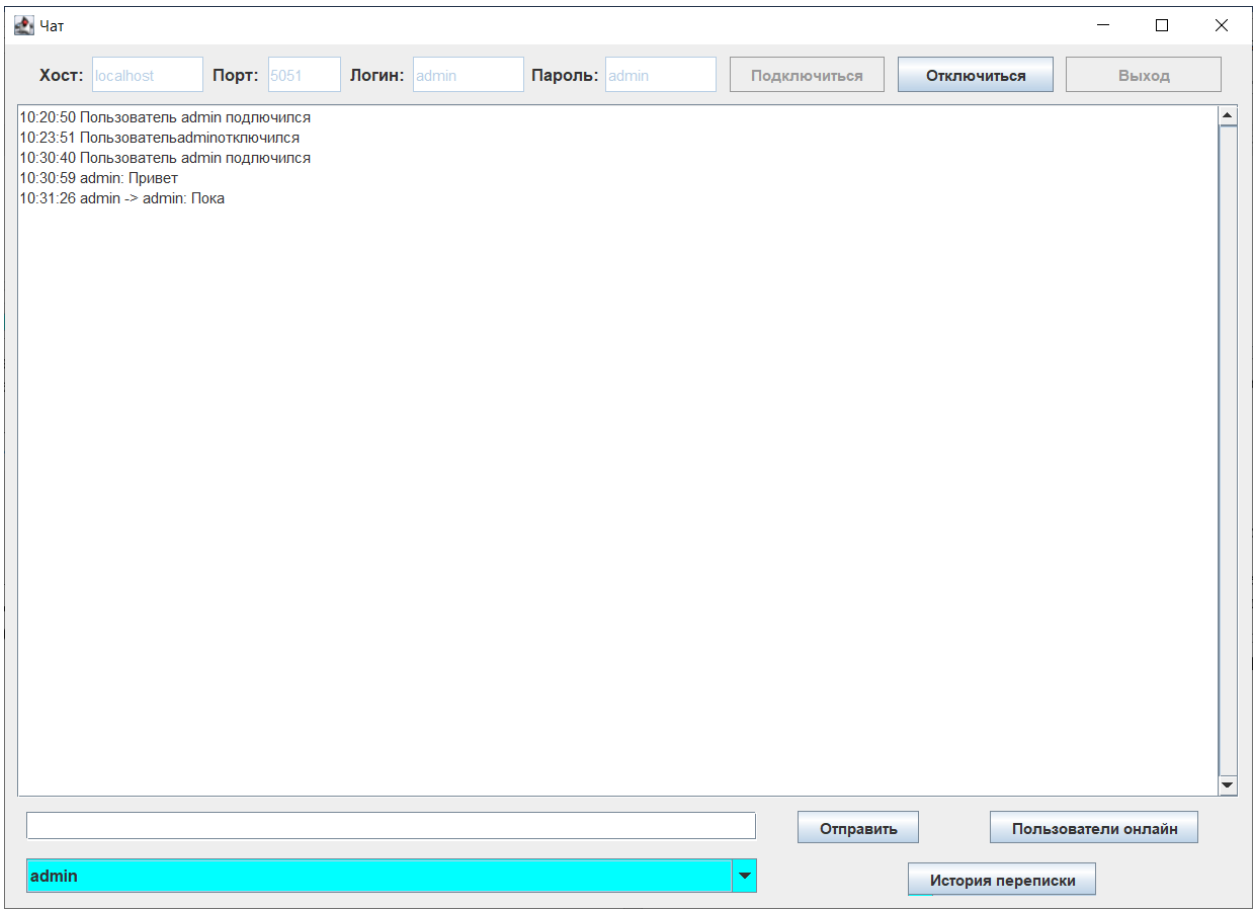
Если данные введены неверно, то выводится диалоговое окно об ошибке. Если возникает любая другая ошибка, то она так же будет отражена в диалоговом окне.



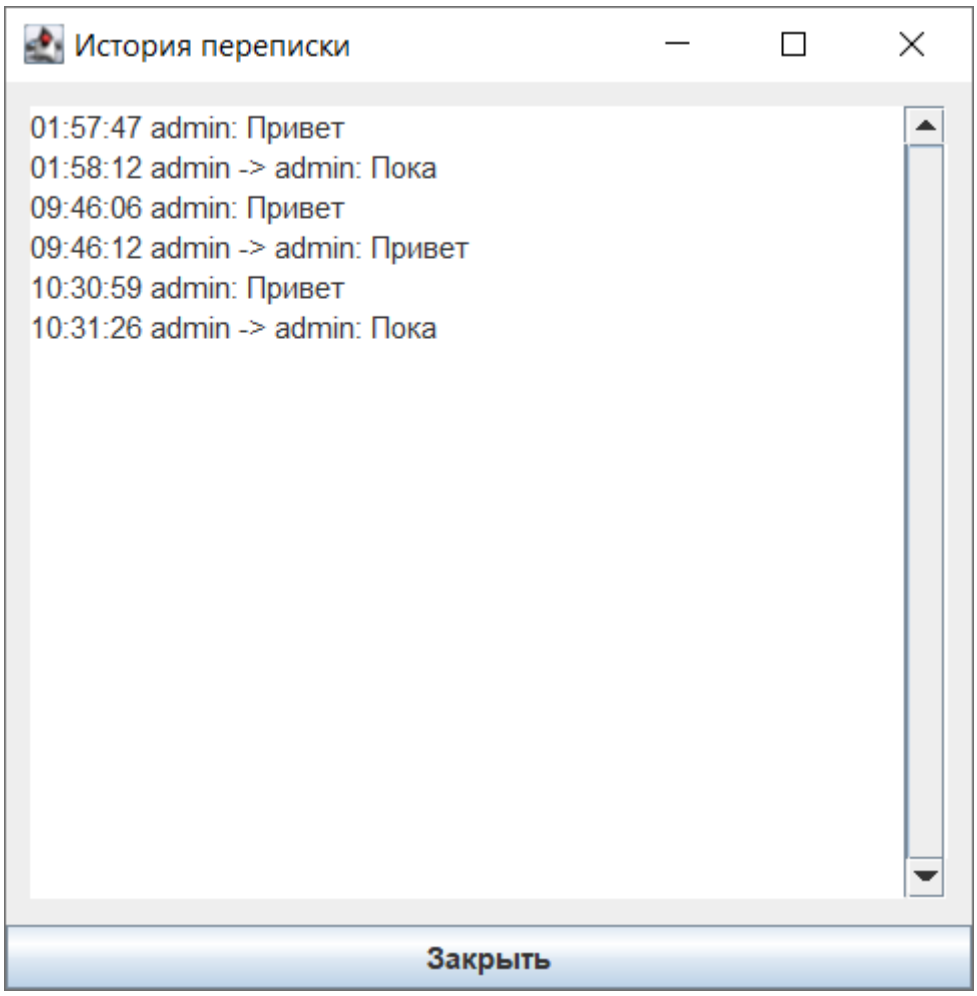
При удачном или неудачном подключении на сервере выводиться сообщение об этом (в логи)



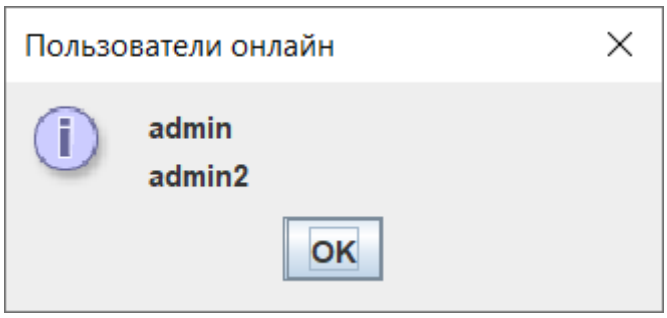
При отправке сообщения, оно будет выводится в чат (если сообщение личное будет показываться отправителю и получателю).



При нажатии на кнопку история сообщений можно просмотреть историю сообщений в отдельном окне. Кнопка закрыть закрывает окно истории переписки.



При нажатии кнопки Пользователи онлайн можно просмотреть всех пользователей, которые в данный момент находятся онлайн.



Заключение

В процессе разработки курсовой работы был создано кроссплатформенное клиент-серверное приложение. Которое задействовало большинство функционала языка Java, такие как коллекции, потоки, работа с файлами, передача данных через TPC/IP соединение и другие.

В курсовой выполнен весь заявленный функционал. Данные отображаются в форме, происходит общение клиента с сервером, вывод сообщений в чат, просмотр истории сообщений и пользователей онлайн, сохранение и загрузка данных сервера.