**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра Веб-технологий и компьютерного моделирования**

**РАЗРАБОТКА ЧАТА СРЕДСТВАМИ ЯЗЫКА JAVA**

Курсовая работа

|  |  |
| --- | --- |
|  | Лукьяновича Александра Сергеевича  студента 2 курса, специальность «веб-программирование и интернет-технологии»  Научный руководитель:  кандидат техн. наук,  доцент Люлькин А. Е. |

|  |
| --- |
|  |

Минск, 2018г.

**Оглавление**

[**Введение** 3](#_Toc514314415)

[**1. Разработка приложений типа Клиент-Сервер средствами Java** 4](#_Toc514314416)

[**1.1 Платформа JavaFX** 4](#_Toc514314417)

[**1.2 Пакет java.net** 5](#_Toc514314418)

[**1.3 Класс Thread** 6](#_Toc514314419)

[**2.** **Реализация чата** 8](#_Toc514314420)

[**2.1 Выбор режима работы** 8](#_Toc514314421)

[**2.2 Режим клиента** 8](#_Toc514314422)

[**2.3 Режим сервера** 12](#_Toc514314423)

[**3. Реализация** 14](#_Toc514314424)

[**3.1 Файлы и структура пакетов** 14](#_Toc514314425)

[**3.2 Руководство пользователя** 15](#_Toc514314426)

[**Заключение** 18](#_Toc514314427)

[**Список использованных источников** 19](#_Toc514314428)

[**Приложение: листинги программы** 20](#_Toc514314429)

# **Введение**

Средства Интернет играют важную роль в жизни общества и бизнеса. Курсовая работа посвящена сетевой архитектуре «клиент-сервер».

Приложения типа клиент-сервер созданы для связи пользователей через всемирную сеть. Логика таких приложений проста: программа-сервер ожидает от клиентских программ запросы и предоставляет им свои ресурсы в виде данных, определяет правила взаимодействия пользователей друг с другом. Поскольку программа-сервер выполняет запросы от многих программ-клиентов, её размещают на специальной вычислительной машине с высокой производительностью. Клиентская программа приводит полученные от сервера данные в вид, приятный для пользователя. Преимущества такого подхода:

* Отсутствие дублирования кода программы-сервера программами-клиентами
* Так как все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются
* Все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов. На сервере проще организовать контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа

**Постановка задач**

1. Усовершенствовать навыки программирования с использованием технологий Java. В частности, ООП, многопоточность, использование пакета java.net. Освоить современный фреймворк JavaFX, а также язык разметки XML
2. Создать приложение «чат» с использованием технологии «клиент-сервер» средствами Java: клиенты обмениваются сообщениями через сервер, который реализует многопоточность, чтобы своевременно обрабатывать запросы пользователей

# **1. Разработка приложений типа Клиент-Сервер средствами Java**

## **1.1 Платформа JavaFX**

JavaFX – платформа для создания приложений с насыщенным графическим интерфейсом, постепенно пришедшая на замену всем известной библиотеке Swing. Она позволяет грамотно отделять бизнес-логику от графического интерфейса, взаимодействующего с пользователем. Можно как создавать окна с помощью SceneBuilder (используется декларативный язык описания FXML), так и в старом стиле: настраивая и устанавливая компоненты вручную. Библиотека JavaFX подключена к JDK начиная с восьмой версии. Предназначена для создания приложений для мобильных устройств и ПК, а также интернет-приложений, работающих в браузерах.

Приложения JavaFX создаются с помощью декларативного языка программирования JavaFX Script. Для разработки приложений на языке JavaFX Script необходимо установить JavaFX SDK, который входит в поставку с Java SE 7. Из кода, написанного на языке JavaFX Script, можно обращаться к любым библиотекам Java. Поэтому совместное использование языков Java и JavaFX Script позволяет решать разнообразные задачи, например, логика бизнес-приложения может быть написана на Java, а графический интерфейс пользователя — на JavaFX Script. Последняя версия JavaFX 8 предоставляет возможность 3D-графики, сенсорного ввода, а также поддержку печати.

Средства включают класс, наследуемый от javafx.application.Application. С помощью данного класса производится изначальный запуск приложения. Для каждого окна имеется свой .fxml файл, в котором описана его графическая составляющая. Кроме того, приложение включает в себя класс-контроллер, в котором реализована логика. Для придания торгового вида каждый элемент графической составляющей можно настроить с помощью каскадных таблиц стилей (CSS), добавив к привычным селекторам префикс –fx-: «-fx-text-fill: red;»

## **1.2 Пакет java.net**

Пакет java.net содержит удобные базовые средства для передачи данных посредством любого Internet-протокола.

**Socket**

Программы-серверы, прослушивающие свои порты, работают под управлением операционной системы. У машин-серверов могут быть самые разные операционные системы, особенности которых передаются программам-серверам.

Чтобы сгладить различия в реализациях разных серверов, между сервером и портом введён промежуточный программный слой, названный *сокетом* (socket). Различаются ***клиентские*** сокеты (*Socket*) и ***серверные*** сокеты (*ServerSocket*).

Клиентские сокеты посылают запрос серверу через порт с указанным номером, серверные сокеты слушают порт с указанным номером.

Соединение с помощью сокетов устанавливается так:

1. Сервер создаёт сокет, прослушивающий порт сервера.

2. Клиент тоже создаёт сокет, через который связывается с сервером, сервер начинает устанавливать (accept) связь с клиентом.

3. Устанавливая связь, сервер создаёт новый сокет, прослушивающий порт с другим, новым номером, и сообщает этот номер клиенту.

4. Клиент посылает запрос на сервер через порт с новым номером.

После этого соединение становится совершенно симметричным — два сокета обмениваются информацией, а сервер через старый сокет продолжает прослушивать прежний порт, ожидая следующего клиента.

В Java сокет — это объект класса socket из пакета java.io. В классе шесть конструкторов, в которые разными способами заносится адрес хоста и номер порта. Чаще всего применяется конструктор



**Методы класса Socket**

Перечислим наиболее интересные, на наш взгляд, методы класса Socket.

Прежде всего, это методы getInputStream() и getOutputStream(), предназначенные для создания входного и выходного потока, соответственно:



Эти потоки связаны с сокетом и должны быть использованы для передачи данных по каналу связи.

После того как работа с сокетом завершена, его необходимо закрыть методом close():



## **1.3 Класс Thread**

Когда операционная система запускает виртуальную машину Java для выполнения приложения, она создает один процесс с несколькими подпроцессами. *Главный*(main) подпроцесс выполняет байт-коды программы, а именно: он сразу же обращается к методу main() приложения. Этот подпроцесс может породить новые подпроцессы, которые, в свою очередь, способны породить подпроцессы и т. д. Главным подпроцессом апплета является один из подпроцессов браузера, в котором апплет выполняется. Главный подпроцесс не играет никакой особой роли, просто он создается первым.

Подпроцесс в Java создается и управляется методами класса Thread. После создания объекта этого класса одним из его конструкторов новый подпроцесс запускается методом start().



Класс Thread реализует интерфейс Runnable. Этот интерфейс описывает только один метод run(). Новый подпроцесс будет выполнять то, что записано в этом методе. Впрочем, класс Thread содержит только пустую реализацию метода run (), поэтому класс Thread не используется сам по себе, он всегда расширяется. При его расширении метод run() переопределяется.





Метод run() не содержит аргументов, т. к. некому передавать их значения в метод. Он не возвращает значения, его некуда передавать. К методу run() нельзя обратиться из программы, это всегда делается автоматически исполняющей системой Java при запуске нового подпроцесса методом start().

Итак, задать действия создаваемого подпроцесса можно двумя способами: расширить класс Thread или реализовать интерфейс Runnable. Первый способ позволяет использовать методы класса Thread для управления подпроцессом. Второй способ применяется в тех случаях, когда надо только реализовать метод run(), или класс, создающий подпроцесс, уже расширяет какой-то другой класс.

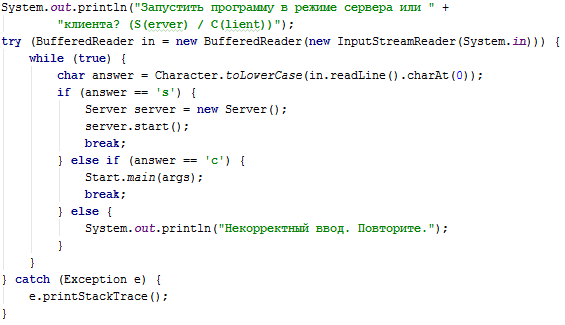
Выполняющийся подпроцесс можно приостановить статическим методом sleep(long ms) на ms миллисекунд. Если вычислительная система может отсчитывать наносекунды, то можно приостановить подпроцесс с точностью до наносекунд методом sleep(long ms, int nanosec).

# **Реализация чата**

Разработка приложения производилась в IDE IntelliJ IDEA Ultimate 2018.1, созданной российскими разработчиками, обладающей большим количеством возможностей, в том числе и для взаимодействия с FXML-файлом и SceneBuilder.

## **2.1 Выбор режима работы**

Для начала нужно выбрать, в каком режиме запускать программу – сервер или клиент. Это нам и нужно первым делом узнать у пользователя, поэтому в метод main(…) пишем следующее:



Здесь всё достаточно просто – спрашиваем, как запускать программу, считываем букву ответа и запускаем соответствующий класс. Стоит пояснить только по поводу класса BufferedReader – это класс стандартной библиотеки, который облегчает работу с вводом данных из консоли. Он инициализируется стандартным потоком ввода.

## **2.2 Режим клиента**

Прежде всего создаём окно авторизации с помощью SceneBuilder, который автоматически изменяет наш .fxml файл

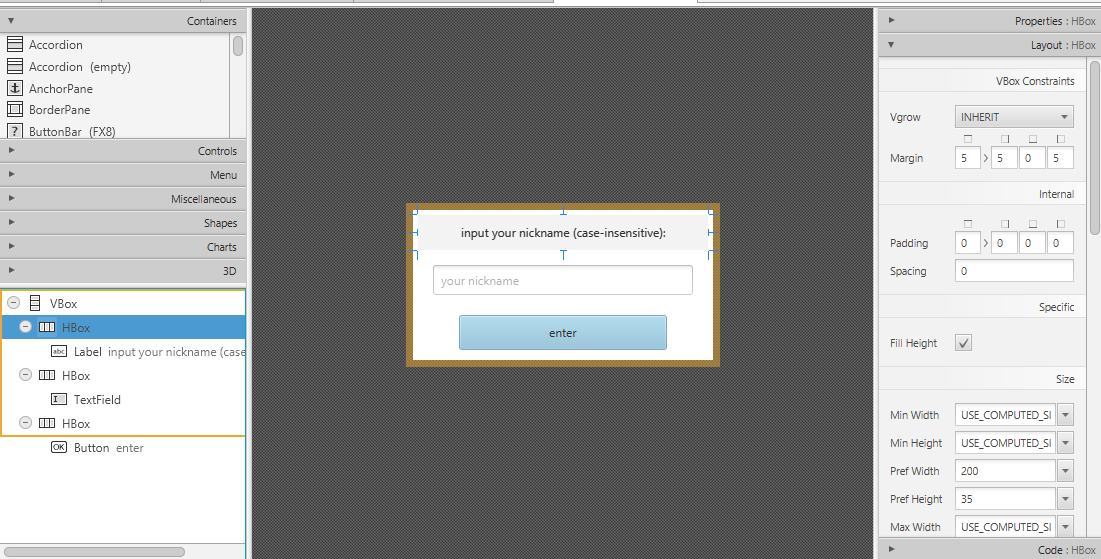
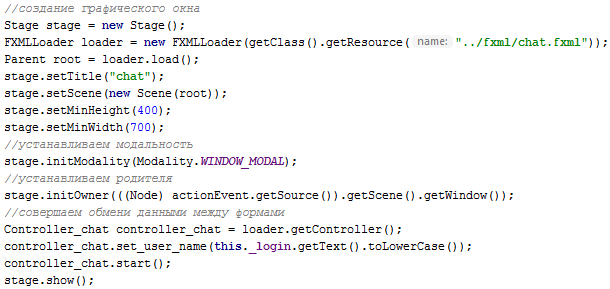


Рис. 1: работа в SceneBuilder

Информация, введённая в этом окне, передаётся в контроллер дочернего окна, где клиент подключается к серверу.



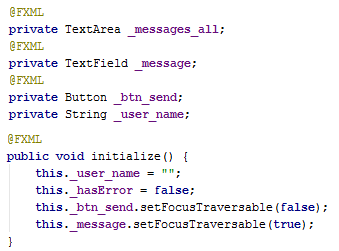
Клиенту нужно подключиться к серверу. Для этого нужно знать IP сервера и порт подключения.



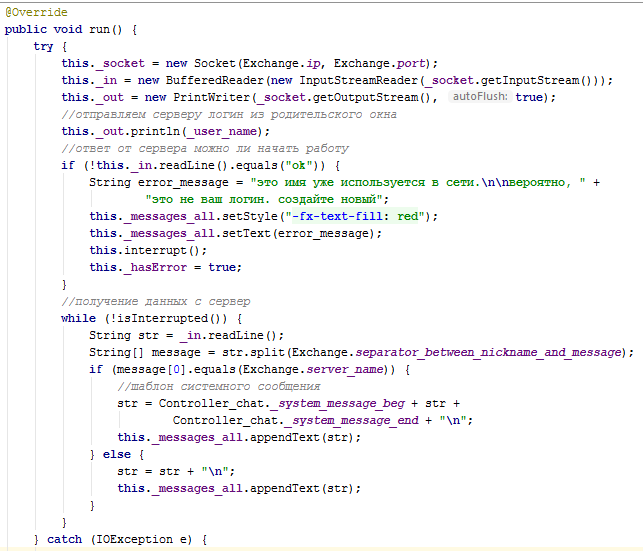
Exchange объявлен как abstract, так как содержит исключительно статические данные и создавать его экземпляр нам ни к чему.

Тестирование проводится на одной машине, потому IP тоже неизменный.

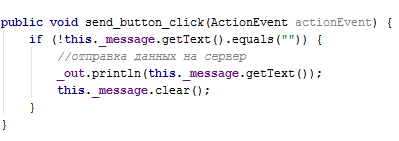
Конструктор контроллера вызывается раньше, чем заполнятся любые аннотированные поля @FXML, затем вызывается метод initialize(). Таким образом, конструктор не имеет доступа к полям @FXML, ссылающимся на компоненты, определенные в файл .fxml, в то время как initialize() имеет к ним доступ.



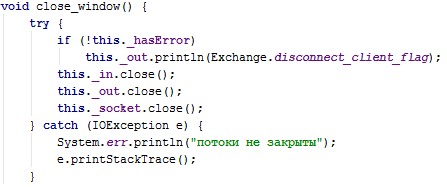
В главном потоке выполняется слушать событий элементов формы, поэтому для постоянного вывода информации, полученной от сервера, мы реализовываем метод run():



Отправка сообщения серверу происходит по нажатию кнопки «send». Её обработчик событий:



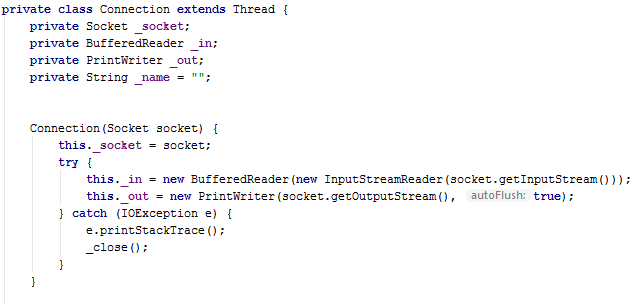
При закрытии окна вызывается метод close\_window():

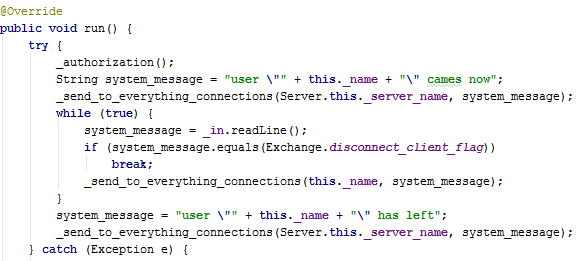


## **2.3 Режим сервера**

Сервер, в отличие от клиента, работает не с классом Socket, а с ServerSocket. При создании его объекта программа никуда не подключается, а просто создаётся сервер на порту, переданном в конструктор.

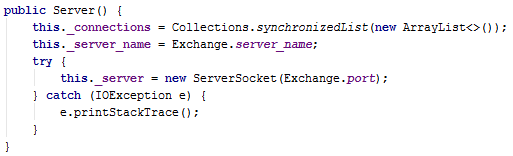
Вся логика работы с конкретным пользователем будет находиться во внутреннем классе Connection, а Server будет только принимать новые подключения и оперировать существующими. Cоздадим класс Connection, который должен в отдельной нити принимать от пользователя сообщения и рассылать их остальным клиентам:



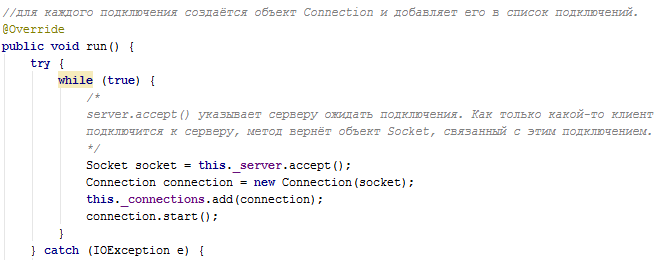


\_connections - это массив со всеми соединениями пользователей. Когда необходимо отправить какое-то сообщение всем, мы перебираем этот массив и обращаемся к каждому клиенту.

Теперь осталось только создать сервер, который будет принимать подключения, создавать объекты Connection и добавлять их в массив. В конструкторе класса Server пишем:



Метод, в котором создаются подключения:



# **3. Реализация**

## **3.1 Файлы и структура пакетов**

Вы, конечно, знаете, что любая программа на Java начинается с метода main(String[] args). Для большей наглядности не будем добавлять его ко всем остальным классам, а создадим отдельный класс Main и пакет для него – main. В программе имеются константы. Я предпочитаю выносить их в отдельный файл в виде public static полей, поэтому создам класс Exchange и также добавим его в пакет main.

Как мы помним, программа должна работать в режиме клиента или сервера. Создадим два соответствующих пакета Client и Server.

В итоге дерево пакетов выглядит так:

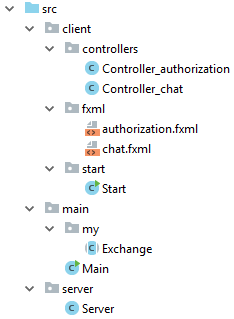


Рис. 2: структура пакетов

## **3.2 Руководство пользователя**

Запускаем метод main(…) из класса Main в режиме сервера, после два раза в режиме клиента.

Если не запустить сервер, мы видим ошибки:

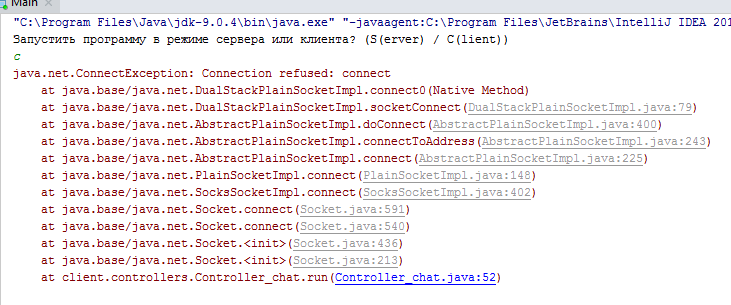


Рис. 3: ошибка: сервер не запущен

Окно слева – это пользователь, который пришёл раньше. Когда подключается второй – пользователю, который уже в сети, приходит уведомление от сервера:

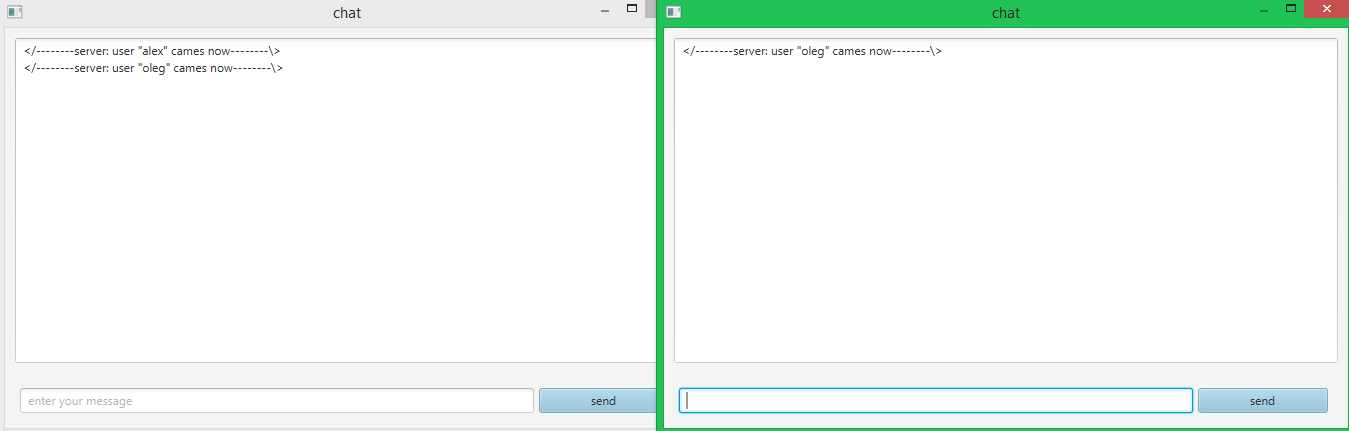


Рис. 4: присоединение пользователя

Если пользователь с таким именем уже в сети, выводится сообщение

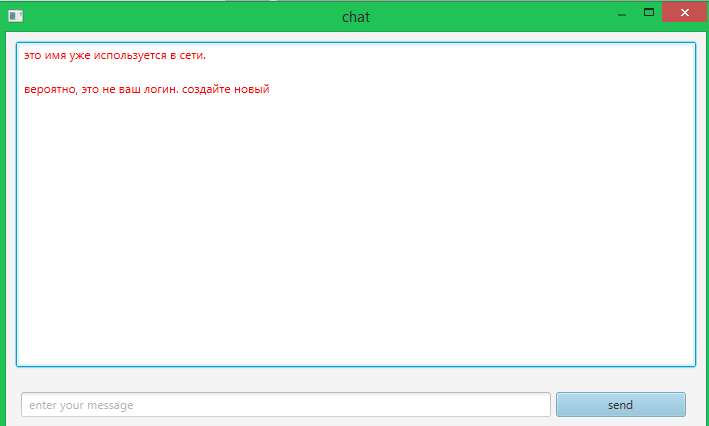


Рис. 5: вход не разрешён сервером

Далее мы наблюдаем как осуществляется обмен сообщениями:

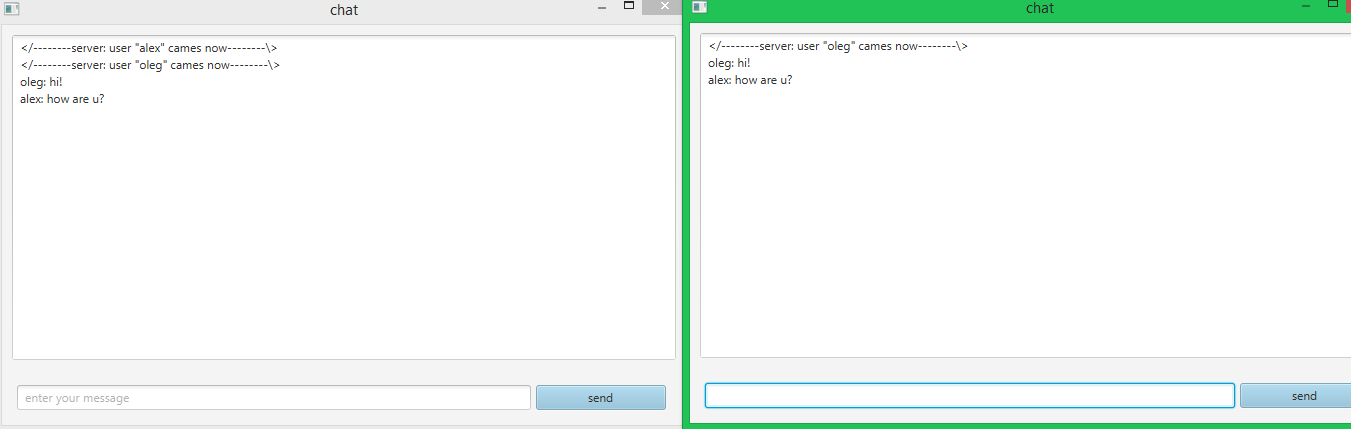


Рис. 6: обмен сообщениями

Когда пользователь выходит из чата, остальные пользователи получают уведомление:

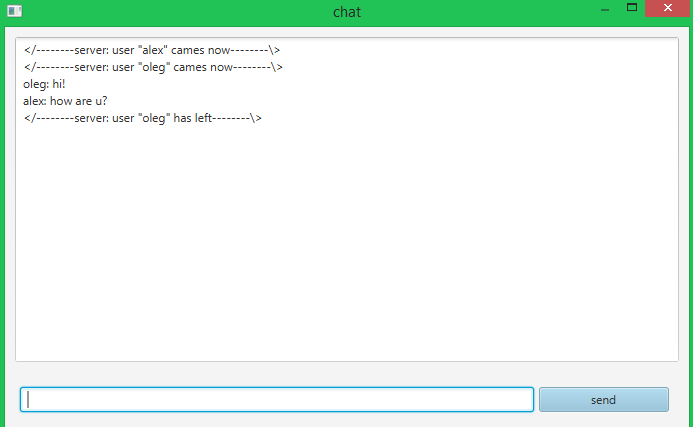


Рис. 7: выход пользователя

# **Заключение**

В рамках курсовой работы были изучены возможности программирования графических интерфейсов с использованием технологий Java. Изучены методы построения архитектуры приложений. Было разработано клиент-серверное приложение «чат», предназначенное для общения пользователей в сети.

Приложение довольно простое по архитектуре, оно служит для быстрого получения информации от иных пользователей, которые находятся в сети. Для входа необходимо ввести login. Если никто не авторизирован в данный момент под этим логином, система пускает Вас в чат и оповещает всех пользователей об этом. Когда Вы покидаете чат – система также оповещает всех пользователей об этом, а Вас переносит обратно на окно регистрации.

Так как приложение разработано для быстрого получения ответа на определённый вопрос(-ы), не реализовано хранилище зарегистрированных пользователей и истории сообщений. Ввиду этого, если Вы вышли из чата и в окне регистрации вновь введёте тот же логин, то для системы вы будете абсолютно новым пользователем.

Среди задач ветки, стоит выделить класс Exchange, в котором находится общая информация для клиентской и серверной части приложения, например, номер порта на сервере.

# **Список использованных источников**

1. Альтман Е.А. Разработка приложений на платформе JavaFX/ Альтман Е.А., Ананьева Н.Г.
2. Тимур Машнин. JavaFX 2.0. Разработка RIA-приложений, Санкт-Петербург: Корона-принт, 2012.
3. Using JavaFX UI Controls [Электронный ресурс] / Режим доступа: http://docs.oracle.com/javafx/2/ui\_controls/jfxpub-ui\_controls.htm
4. И.Н. Блинов, В.С. Романчик. Java. Промышленное программирование, Мн., УниверсалПресс, 2007.
5. М. Морган. Java 2. Руководство разработчика, Москва, Вильямс, 2000
6. Ильдар Хабибулин Java 3-е издание, Санкт-Петербург 2008
7. БХВ-Петербург, 2003. — 400 с: ил.
8. Б. Эккель. Философия Java, СПб: Питер, 2001.- 880с.

# **Приложение: листинги программы**

Main.java

**public class** Main {  
 **public static void** main(String[] args) {  
 System.***out***.println(**"Запустить программу в режиме сервера или клиента? (S(erver) / C(lient))"**);  
 **try** (BufferedReader in = **new** BufferedReader(**new** InputStreamReader(System.***in***))) {  
 **while** (**true**) {  
 **char** answer = Character.*toLowerCase*(in.readLine().charAt(0));  
 **if** (answer == **'s'**) {  
 Server server = **new** Server();  
 server.start();  
 **break**;  
 } **else if** (answer == **'c'**) {  
 Start.*main*(args);  
 **break**;  
 } **else** {  
 System.***out***.println(**"Некорректный ввод. Повторите."**);  
 }  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Server.java

**public class** Server **extends** Thread {  
 **private** ServerSocket **\_server**;  
 **private** List<Connection> **\_connections**;  
 **private** String **\_server\_name**;  
  
  
 **public** Server() {  
 **this**.**\_connections** = Collections.*synchronizedList*(**new** ArrayList<>());  
 **this**.**\_server\_name** = Exchange.***server\_name***;  
 **try** {  
 **this**.**\_server** = **new** ServerSocket(Exchange.***port***);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 *//для каждого подключения создаётся объект Connection и добавляет его в список подключений.* @Override  
 **public void** run() {  
 **try** {  
 **while** (**true**) {  
 */\*  
 server.accept() указывает серверу ожидать подключения. Как только какой-то клиент  
 подключится к серверу, метод вернёт объект Socket, связанный с этим подключением.  
 \*/* Socket socket = **this**.**\_server**.accept();  
 Connection connection = **new** Connection(socket);  
 **this**.**\_connections**.add(connection);  
 connection.start();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally** {  
 \_close\_all();  
 }  
 }  
  
  
 **private void** \_close\_all() {  
 **try** {  
 **this**.**\_server**.close();  
 **synchronized** (**this**.**\_connections**) {  
 **for** (Connection connection : **this**.**\_connections**) {  
 connection.\_close();  
 }  
 }  
 } **catch** (IOException e) {  
 System.***err***.println(**"потоки не закрыты"**);  
 e.printStackTrace();  
 }  
 }  
  
  
 **private class** Connection **extends** Thread {  
 **private** Socket **\_socket**;  
 **private** BufferedReader **\_in**;  
 **private** PrintWriter **\_out**;  
 **private** String **\_name** = **""**;  
  
  
 Connection(Socket socket) {  
 **this**.**\_socket** = socket;  
 **try** {  
 **this**.**\_in** = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  
 **this**.**\_out** = **new** PrintWriter(socket.getOutputStream(), **true**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 \_close();  
 }  
 }  
  
  
 **private void** \_send\_to\_everything\_connections(String login, String message) {  
 *//шаблон сообщения* String str = login + Exchange.***separator\_between\_nickname\_and\_message*** + message;  
 **synchronized** (**\_connections**) {  
 **for** (Connection connection : **\_connections**) {  
 connection.**\_out**.println(str);  
 }  
 }  
 }  
  
  
 **private void** \_authorization() **throws** Exception {  
 **this**.**\_name** = **this**.**\_in**.readLine();  
 **for** (Connection connection : **\_connections**) {  
 *//если имя уже занято, обрываем соединение* **if** (connection.**\_name**.equals(**this**.**\_name**) && **this**!=connection) {  
 **this**.**\_out**.println(**"close"**);  
 **throw new** Exception(**"логин занят"**);  
 }  
 }  
 **this**.**\_out**.println(**"ok"**);  
 }  
  
  
 @Override  
 **public void** run() {  
 **try** {  
 \_authorization();  
 String system\_message = **"user \""** + **this**.**\_name** + **"\" cames now"**;  
 \_send\_to\_everything\_connections(Server.**this**.**\_server\_name**, system\_message);  
 **while** (**true**) {  
 system\_message = **\_in**.readLine();  
 **if** (system\_message.equals(Exchange.***disconnect\_client\_flag***))  
 **break**;  
 \_send\_to\_everything\_connections(**this**.**\_name**, system\_message);  
 }  
 system\_message = **"user \""** + **this**.**\_name** + **"\" has left"**;  
 \_send\_to\_everything\_connections(Server.**this**.**\_server\_name**, system\_message);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 \_close();  
 }  
 }  
  
  
 **private void** \_close() {  
 **try** {  
 **this**.**\_in**.close();  
 **this**.**\_out**.close();  
 **this**.**\_socket**.close();  
 **\_connections**.remove(**this**);  
 **if** (**\_connections**.size() == 0) {  
 Server.**this**.\_close\_all();  
 }  
 } **catch** (IOException e) {  
 System.***err***.println(**"подключение не закрыто"**);  
 e.printStackTrace();  
 }  
 }  
 }  
}

Exchange.java

**public abstract class** Exchange {  
 **public static final int *port*** = 8290;  
 **public static final** String ***ip*** = **"localhost"**;  
 **public static final** String ***server\_name*** = **"server"**;  
 **public static final** String ***separator\_between\_nickname\_and\_message*** = **": "**;  
 **public static final** String ***disconnect\_client\_flag*** = **"$$$///111"**;  
}

Start.java

**public class** Start **extends** Application {  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception {  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"../fxml/authorization.fxml"**));  
 primaryStage.setTitle(**"authorization"**);  
 primaryStage.setScene(**new** Scene(root, 300, 150));  
 primaryStage.show();  
 }  
  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

Controller\_chat.java

**public class** Controller\_chat **extends** Thread {  
 @FXML  
 **private** TextArea **\_messages\_all**;  
 @FXML  
 **private** TextField **\_message**;  
 @FXML  
 **private** Button **\_btn\_send**;  
 **private** String **\_user\_name**;  
 **private** BufferedReader **\_in**;  
 **private** PrintWriter **\_out**;  
 **private** Socket **\_socket**;  
  
 **private static final** String ***\_system\_message\_beg*** = **"</--------"**;  
 **private static final** String ***\_system\_message\_end*** = **"--------\\>"**;  
 **private boolean \_hasError**;  
  
  
 @FXML  
 **public void** initialize() {  
 **this**.**\_user\_name** = **""**;  
 **this**.**\_hasError** = **false**;  
 **this**.**\_btn\_send**.setFocusTraversable(**false**);  
 **this**.**\_message**.setFocusTraversable(**true**);  
 }  
  
  
 **void** set\_user\_name(String login) {  
 **this**.**\_user\_name** = login;  
 }  
  
  
 @Override  
 **public void** run() {  
 **try** {  
 **this**.**\_socket** = **new** Socket(Exchange.***ip***, Exchange.***port***);  
 **this**.**\_in** = **new** BufferedReader(**new** InputStreamReader(**\_socket**.getInputStream()));  
 **this**.**\_out** = **new** PrintWriter(**\_socket**.getOutputStream(), **true**);  
 *//отправляем серверу логин из родительского окна* **this**.**\_out**.println(**\_user\_name**);  
 *//ответ от сервера можно ли начать работу* **if** (!**this**.**\_in**.readLine().equals(**"ok"**)) {  
 String error\_message = **"это имя уже используется в сети.\n\nвероятно, это не ваш логин. создайте новый"**;  
 **this**.**\_messages\_all**.setStyle(**"-fx-text-fill: red"**);  
 **this**.**\_messages\_all**.setText(error\_message);  
 **this**.interrupt();  
 **this**.**\_hasError** = **true**;  
 }  
 *//получение данных с сервер* **while** (!isInterrupted()) {  
 String str = **\_in**.readLine();  
 String[] message = str.split(Exchange.***separator\_between\_nickname\_and\_message***);  
 **if** (message[0].equals(Exchange.***server\_name***)) {  
 *//шаблон системного сообщения* str = Controller\_chat.***\_system\_message\_beg*** + str + Controller\_chat.***\_system\_message\_end*** + **"\n"**;  
 **this**.**\_messages\_all**.appendText(str);  
 } **else** {  
 str = str + **"\n"**;  
 **this**.**\_messages\_all**.appendText(str);  
 }  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 **public void** send\_button\_click(ActionEvent actionEvent) {  
 **if** (!**this**.**\_message**.getText().equals(**""**)) {  
 *//отправка данных на сервер* **\_out**.println(**this**.**\_message**.getText());  
 **this**.**\_message**.clear();  
 }  
 }  
  
  
 **void** close\_window() {  
 **try** {  
 **if** (!**this**.**\_hasError**)  
 **this**.**\_out**.println(Exchange.***disconnect\_client\_flag***);  
 **this**.**\_in**.close();  
 **this**.**\_out**.close();  
 **this**.**\_socket**.close();  
 } **catch** (IOException e) {  
 System.***err***.println(**"потоки не закрыты"**);  
 e.printStackTrace();  
 }  
 }  
}

Controller\_authorization.java

**public class** Controller\_authorization {  
 @FXML  
 **private** TextField **\_login**;  
  
  
 **public void** submit\_login(ActionEvent actionEvent) {  
 **if** (!**this**.**\_login**.getText().equals(**""**)) {  
 **try** {  
 *//создание графического окна* Stage stage = **new** Stage();  
 FXMLLoader loader = **new** FXMLLoader(getClass().getResource(**"../fxml/chat.fxml"**));  
 Parent root = loader.load();  
 stage.setTitle(**"chat"**);  
 stage.setScene(**new** Scene(root));  
 stage.setMinHeight(400);  
 stage.setMinWidth(700);  
 *//устанавливаем модальность* stage.initModality(Modality.***WINDOW\_MODAL***);  
 *//устанавливаем родителя* stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());  
 *//совершаем обмени данными между формами* Controller\_chat controller\_chat = loader.getController();  
 controller\_chat.set\_user\_name(**this**.**\_login**.getText().toLowerCase());  
 controller\_chat.start();  
 stage.show();  
 *//событие нажатия на крестик* stage.setOnCloseRequest(**new** EventHandler<WindowEvent>() {  
 @Override  
 **public void** handle(WindowEvent event) {  
 controller\_chat.interrupt();  
 controller\_chat.close\_window();  
 }  
 });  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

chat.fxml

*<?***xml version="1.0" encoding="UTF-8"***?>*

<**VBox maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="700.0" xmlns="http://javafx.com/javafx/8.0.121" xmlns:fx="http://javafx.com/fxml/1" fx:controller="client.controllers.Controller\_chat"**>  
 <**children**>  
 <**AnchorPane prefHeight="350.0" VBox.vgrow="ALWAYS"**>  
 <**children**>  
 <**TextArea fx:id="\_messages\_all" editable="false" prefHeight="330.0" prefWidth="680.0" promptText="the message history will displayed here." wrapText="true" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0"** />  
 </**children**>  
 <**VBox.margin**>  
 <**Insets bottom="10.0" left="10.0" right="10.0" top="10.0"** />  
 </**VBox.margin**>  
 </**AnchorPane**>  
 <**AnchorPane prefHeight="50.0" prefWidth="700.0" VBox.vgrow="ALWAYS"**>  
 <**children**>  
 <**Button fx:id="\_btn\_send" contentDisplay="CENTER" defaultButton="true" layoutX="540.0" mnemonicParsing="false" onAction="#send\_button\_click" prefHeight="70.0" prefWidth="130.0" text="send" AnchorPane.bottomAnchor="5.0" AnchorPane.rightAnchor="10.0" AnchorPane.topAnchor="5.0"** />  
 <**TextField fx:id="\_message" prefHeight="34.0" prefWidth="500.0" promptText="enter your message" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="5.0" AnchorPane.rightAnchor="145.0" AnchorPane.topAnchor="5.0"** />  
 </**children**>  
 <**VBox.margin**>  
 <**Insets bottom="10.0" left="10.0" right="10.0" top="10.0"** />  
 </**VBox.margin**>  
 </**AnchorPane**>  
 </**children**>  
</**VBox**>

authorization.fxml

*<?***xml version="1.0" encoding="UTF-8"***?>*

<**VBox alignment="CENTER" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="150.0" prefWidth="300.0" xmlns="http://javafx.com/javafx/8.0.121" xmlns:fx="http://javafx.com/fxml/1" fx:controller="client.controllers.Controller\_authorization"**>  
 <**children**>  
 <**HBox alignment="CENTER" prefHeight="35.0" prefWidth="200.0"**>  
 <**children**>  
 <**Label alignment="CENTER" contentDisplay="CENTER" text="input your nickname (case-insensitive):"** />  
 </**children**>  
 <**VBox.margin**>  
 <**Insets left="5.0" right="5.0" top="5.0"** />  
 </**VBox.margin**>  
 </**HBox**>  
 <**HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0"**>  
 <**children**>  
 <**TextField fx:id="\_login" prefHeight="30.0" prefWidth="260.0" promptText="your nickname"** />  
 </**children**>  
 <**VBox.margin**>  
 <**Insets bottom="5.0" left="5.0" right="5.0" top="5.0"** />  
 </**VBox.margin**>  
 </**HBox**>  
 <**HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0"**>  
 <**children**>  
 <**Button defaultButton="true" mnemonicParsing="false" onAction="#submit\_login" prefHeight="40.0" prefWidth="208.0" text="enter"**>  
 <**HBox.margin**>  
 <**Insets bottom="10.0" left="5.0" right="5.0" top="5.0"** />  
 </**HBox.margin**>  
 </**Button**>  
 </**children**>  
 </**HBox**>  
 </**children**>  
</**VBox**>