

**Реферат**

Курсовой проект: 34 страницы, 11 рисунков, 6 источников, 2 приложения.

ЧАТ, JAVA, SOCKET, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, UML-ДИАГРАММЫ, ДВУХУРОВНЕВАЯ АРХИТЕКТУРА

Объектом исследования является процесс разработки клиент-серверного приложения на Java.

Предметом исследования являетсятехнология разработки клиент-серверного приложения с двухуровневой архитектурой.

Целью работы является проектирование и разработка клиент-серверного приложения на объектно-ориентированном языке Java.

Основные полученные результаты:

- проведен анализ предметной области;

- изучены основы разработки клиент-серверных приложений;

- спроектировано и разработано приложение с клиент-серверной архитектурой.

Работа направлена на проектирование и разработку приложения с клиент-серверной архитектурой по созданной технологии разработки клиент-серверного приложения. Рассматривается специфика разработки приложений на языке программирования Java. В ходе работы было создано клиент-серверное приложение «Чат». Исходные коды написаны на языке программирования Java.

Содержание

[Введение 6](#_Toc28081451)

[1 Анализ предметной области 8](#_Toc28081452)

[1.1 Общее описание предметной области 8](#_Toc28081453)

[1.2 Исследование существующих способов и средств построения сетевых приложений 8](#_Toc28081454)

[1.3 Описание механизмов, предназначенных для сетевого взаимодействия программ. 9](#_Toc28081455)

[1.4 Модели архитектуры разработки сетевого приложения 9](#_Toc28081456)

[2 Назначение проектируемого приложения 11](#_Toc28081457)

[3 Концептуальное проектирование сетевого приложения 12](#_Toc28081458)

[3.1 Описание функциональных особенностей и пользователей приложения 12](#_Toc28081459)

[3.2 Взаимодействие классов приложения 12](#_Toc28081460)

[3.3 Описание последовательности взаимодействия объектов приложения 13](#_Toc28081461)

[4 Обоснование выбора вида программного обеспечения. 14](#_Toc28081462)

[5 Реализация сетевого приложения 15](#_Toc28081463)

[5.1 Взаимодействие клиента и сервера 15](#_Toc28081464)

[6 Описание работы с приложением 21](#_Toc28081465)

[6.1 Описание основных форм приложения 21](#_Toc28081466)

[6.2 Руководство пользователя. 23](#_Toc28081467)

[Заключение 26](#_Toc28081468)

[Список использованных источников 27](#_Toc28081469)

[Приложение 1 28](#_Toc28081470)

[Приложение 2 35](#_Toc28081471)

**Нормативные ссылки**

В данном курсовом проекте использовались следующие нормативные ссылки:

1. ГОСТ Р 1.5-2004 Стандарты национальные Российской Федерации. Правила построения, изложения, оформления и обозначения

2. ГОСТ 2.104-68 ЕСКД. Основные надписи

3. ГОСТ 7.80-2000 СИБИД. Библиографическая запись. Заголовок. Общие требования и правила составления

4. ГОСТ 7.82-2001 СИБИД. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления

5. ГОСТ 7.9-95 СИБИД. Реферат и аннотация. Общие требования

6. ГОСТ 19.001-77 ЕСПД. Общие положения

7. ГОСТ 19.104-78 ЕСПД. Основные надписи

8. ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам

9. ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению

10. ГОСТ 19.402-78 ЕСПД. Описание программы

11. ГОСТ 19.404-79 ЕСПД. Пояснительная записка. Требования к содержанию и оформлению

# Введение

Возможность передачи информации на расстоянии интересовала людей всегда. Для того чтобы обмениваться информацией не только при личной встрече, но и на больших расстояниях, люди изобретали всё новые технические средства, протягивали структурированные кабельные сети по всему миру, организовывали почтовые системы, запускали спутники связи. С развитием информационных технологий стали возможным еще более глобальные коммуникации. Сегодня люди с легкостью по всему миру могут обмениваться не только сообщениями, но и графикой и звуковой и другими видами информации. Историческим «докомпьютерным» предшественником чатов, несомненно, являлся телефон. Ни телеграф, ни почта не могли позволить общаться в реальном времени и не были доступны в домашних условиях. Изобретение и распространение телефона вызвало настоящую революцию в способах и средствах общения. Возможность поговорить с собеседником в реальном времени на другом континенте казалась настоящим чудом.

Язык Java является основой практически для всевозможных типов сетевых приложений и всеобщим стандартом для разработки встроенных и мобильных приложений, веб-контента, игр, и корпоративного программного обеспечения. В мире насчитывается более девяти миллионов специалистов, разрабатывающих приложения на Java, которая позволяет эффективно разрабатывать, тестировать, внедрять и использовать программное обеспечение.

Объект исследования:процесс разработки клиент-серверного  
приложения на Java.

Предмет исследования:технология разработки клиент-серверного приложения с двухуровневой архитектурой.

Цель исследования:создать технологию разработки клиент-сервера и клиент-серверное приложение на объектно-ориентированном языке Java.

Задачи:

1. рассмотреть подходы и принципы создания клиент-серверных приложений;
2. проанализировать особенности разработки клиент-серверных приложений на языке Java;
3. обосновать технологию разработки клиент-серверных приложений;
4. спроектировать и разработать клиент-серверное приложение «Чат» на языке Java.

# 1 Анализ предметной области

## 1.1 Общее описание предметной области

Принято считать, что наиболее универсальный способ общения - при помощи служб мгновенного обмена сообщениями, таких, как ICQ, Skype, Qip и пр. Этот вариант не всегда удобен, ведь для использования вышеперечисленных клиентов необходимым условием является подключение к интернету. Локальные чаты могут работать без выхода в глобальную паутину. Для общения по локальной сети существует множество готовых решений.

## 1.2 Исследование существующих способов и средств построения сетевых приложений

Программа CommFort позволяет тонко организовать работу с большим количеством пользователей и предоставляет возможность общения посредством голоса. Если по каким-то соображениям не устраивает обязательное наличие сервера, можно попробовать установить Vypress Chat. Local Network Chat - это отличное решение для небольшой домашней сети, которая объединяет максимум три компьютера. LANcet Chat - хороший чат с поддержкой русского языка, рассчитанный на небольшую локальную сеть, с общим числом пользователей около двадцати человек. Основной недостаток в том, что они являются платными программами. Имеют 31-дневный испытательный срок без функциональных ограничений; после его завершения продолжают работать в режиме ограниченной функциональности, и при запуске отображается окно с предложением приобрести лицензию на постоянное использование. Решено разработать программу «Чат», которая бы могла оперативно отправлять сообщения и файлы пользователям в сети. Программа также должна работать на всех системах семейства Windows. Программное средство работает по схеме клиент-сервер и состоит из следующих функциональных частей:

1. Сервер. На его стороне должен работать в скрытом от пользователя режиме агент, который предоставляет возможность принимать информацию от одного клиента и рассылать ее остальным;
2. Клиент. Подключается к серверу и представляет собой приложение с UI (user interface) Windows.

## 1.3 Описание механизмов, предназначенных для сетевого взаимодействия программ

Процесс разработки программного обеспечения – набор правил, согласно которой построена разработка программного обеспечения. Приложение можно назвать клиент-серверным если оно включает в себя клиент-серверную архитектуру. Разработку клиент-серверного приложения необходимо начинать с выбора архитектуры клиент-сервера.

## 1.4 Модели архитектуры разработки сетевого приложения

Архитектура «клиент-сервер»(рисунок 1) характеризуется наличием по крайне мере двух взаимодействующих, самостоятельных процессов – клиента и сервера. Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодействия между клиентом и сервером, которые называются протоколом обмена (протоколом взаимодействия). Процессы, осуществляющие некоторую службу, например, службу базы данных или файловой системы, называются серверами, а процессы, запрашивающие службы у серверов посредством посылки запроса и последующего ожидания ответа от сервера, называются клиентами. Фактически эти процессы – программное обеспечение, которое установлено на разных вычислительных машинах и взаимодействующее между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине.

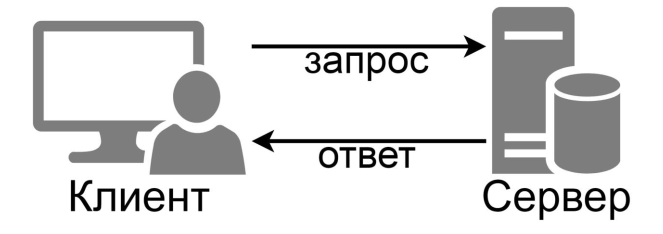


Рисунок 1 – Архитектура клиент-серверного приложения

# 2 Назначение проектируемого приложения

Основной целью создания сетевого чата является мгновенный обмен сообщениями. Использование данного программного продукта поможет сократить время выполнения какой-либо работы, или просто хорошо провести время, общаясь в кругу друзей.

Сетевой чат может быть использован как для малого круга людей, т.е. в домашних условиях, так и на производстве небольшого предприятия. Разрабатываемое программное обеспечение может применяться как для важных дел связанных непосредственно с работой, так и для развлекательных целей.

Разрабатываемое программное обеспечение должно обеспечивать следующие функции:

1. сетевой чат должен обеспечивать мгновенный обмен сообщениями, быть полезным в использовании и потреблять наименьшее количество ресурсов персонального компьютера;
2. программа должна отображать список подключенных пользователей, обеспечивать стабильный поток обмена информацией;
3. программный продукт должен иметь дружественный интерфейс, не требующий специальной подготовки пользователей, должен способствовать более эффективному распределению и экономии времени.

# 3 Концептуальное проектирование сетевого приложения

## 3.1 Описание функциональных особенностей и пользователей приложения

На рисунке 2 изображена диаграмма прецедентов. Эта диаграмма отображает отношения между пользователями, которые взаимодействуют с системой и сервисами, которые предоставляет спроектированная система.

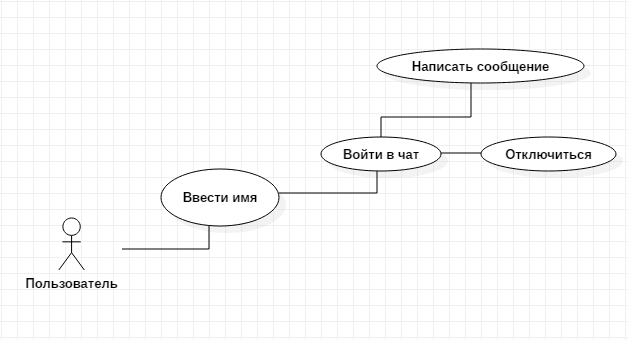


Рисунок 2 – Диаграмма прецедентов

## 3.2 Взаимодействие классов приложения

На рисунке 3 изображена диаграмма классов, входящих в состав программного обеспечения.

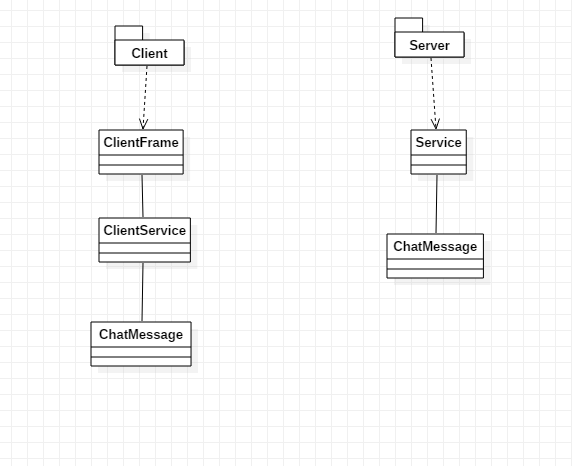


Рисунок 3 – Диаграмма классов

## 3.3 Описание последовательности взаимодействия объектов приложения

С помощью диаграммы последовательностей(рисунок 4) видно как происходит обмен информации между пользователем, клиентом и сервером.



Рисунок 4 – Диаграмма последовательностей

# 4 Обоснование выбора вида программного обеспечения

Для создания клиент-серверного приложения была выбрана двухуровневая архитектура, так как она легко реализуема и вся обработка данных происходит на сервере, а также она не требует никаких финансовых вложений.

Для разработки приложения были выбраны объектно-ориентированный язык Java. Java — это язык программирования общего назначения, который следует парадигме объектно-ориентированного программирования и подходу «Написать один раз и использовать везде». Java используется для настольных, сетевых, мобильных и корпоративных приложений. Этот язык идеально подходит для написания сетевых приложений.

Приложение будем разрабатывать в Intellij Idea, так как это наиболее удобная и современная среда разработки, которая обладает всем необходимым функционалом для создания сетевых приложений. Также в нее включен инструментарий для разработки интерфейса приложения, что тоже является большим плюсом. IntelliJ IDEA — это наиболее интеллектуальная среда. Она по ходу написания и выполнения может проанализировать код, выявить ошибки и предложить достойное решение. Основные преимущества этой среды разработки:

* поддержка широкого круга технологий;
* графический редактор для создания интерфейса;
* красивый и интуитивный интерфейс среды разработки.

# 5 Реализация сетевого приложения

Архитектура клиент-серверного приложения подразумевает наличие в проекте двух приложений – клиента и сервера. Покажем их взаимодействие.

## 5.1 Взаимодействие клиента и сервера

При запуске приложения сервера, программа должна открыть сетевое соединение на определенном порте. Метод, в котором сервер создает сокет, который «слушает» определенный порт и ждет подключения клиента:

**public** Service() {  
 **try** {  
 **serverSocket** = **new** ServerSocket(5678);  
  
 System.***out***.println(**"Сервер запущен!"**);  
  
 **while** (**true**) {  
 **socket** = **serverSocket**.accept();  
  
 **new** Thread(**new** ListenerSocket(**socket**)).start();  
 }  
  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
}

Соответственно при запуске клиентского приложения оно подключается к серверу. Метод, в котором описаны функция подключения к серверу, описан ниже:

**public** Socket connect() {  
 **try** {  
 **this**.**socket** = **new** Socket(**"localhost"**, 5678);  
 **this**.**output** = **new** ObjectOutputStream(**socket**.getOutputStream());  
 } **catch** (UnknownHostException ex) {  
 Logger.*getLogger*(ClientService.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(ClientService.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
  
 **return socket**;  
}

Как только соединение установлено, на сервере и клиенте должны обновиться списки пользователей онлайн. Метод со стороны клиента, который получает список пользователей в сети от сервера и обновляет его на клиенте, приведен ниже:

**private void** refreshOnlines(ChatMessage message) {  
 System.***out***.println(message.getSetOnlines().toString());  
   
 Set<String> names = message.getSetOnlines();  
   
 names.remove(message.getName());  
   
 String[] array = (String[]) names.toArray(**new** String[names.size()]);  
   
 **this**.**listOnlines**.setListData(array);  
 **this**.**listOnlines**.setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);  
 **this**.**listOnlines**.setLayoutOrientation(JList.***VERTICAL***);  
}

А с помощью данного метода сервер обновляет список пользователей и передает информацию всем клиентам:

**private void** sendOnlines() {  
 Set<String> setNames = **new** HashSet<String>();  
 **for** (Map.Entry<String, ObjectOutputStream> kv : **mapOnlines**.entrySet()) {  
 setNames.add(kv.getKey());  
 }  
  
 ChatMessage message = **new** ChatMessage();  
 message.setAction(Action.***USERS\_ONLINE***);  
 message.setSetOnlines(setNames);  
  
 **for** (Map.Entry<String, ObjectOutputStream> kv : **mapOnlines**.entrySet()) {  
 message.setName(kv.getKey());  
 **try** {  
 kv.getValue().writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
}

Передача сообщений между сервером и клиентами происходит следующим образом. Когда пользователь на клиентском приложении вводит сообщение и нажимает кнопку отправить, запускается метод, который отправляет сообщение на сервер:

**public void** send(ChatMessage message) {  
 **try** {  
 **output**.writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(ClientService.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
}

Сервер, в свою очередь, ожидает сообщение от клиента и как только оно приходит на сервер рассылает его с помощью кода описанного в методе run():

**public void** run() {  
 ChatMessage message = **null**;  
 **try** {  
 **while** ((message = (ChatMessage) **input**.readObject()) != **null**) {  
 Action action = message.getAction();  
  
 **if** (action.equals(Action.***CONNECT***)) {  
 **boolean** isConnect = connect(message, **output**);  
 **if** (isConnect) {  
 **mapOnlines**.put(message.getName(), **output**);  
 sendOnlines();  
 }  
 } **else if** (action.equals(Action.***DISCONNECT***)) {  
 disconnect(message, **output**);  
 sendOnlines();  
 **return**;  
 } **else if** (action.equals(Action.***SEND\_ONE***)) {  
 sendOne(message);  
 } **else if** (action.equals(Action.***SEND\_ALL***)) {  
 sendAll(message);  
 }  
 }  
 } **catch** (IOException ex) {  
 ChatMessage cm = **new** ChatMessage();  
 cm.setName(message.getName());  
 disconnect(cm, **output**);  
 sendOnlines();  
 System.***out***.println(message.getName() + **" покинул чат!"**);  
 } **catch** (ClassNotFoundException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
}

На сервере происходит обработка и отправка сообщений с помощью одного из данных методов:

**private void** send(ChatMessage message, ObjectOutputStream output) {  
 **try** {  
 output.writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
}  
  
**private void** sendOne(ChatMessage message) {  
 **for** (Map.Entry<String, ObjectOutputStream> kv : **mapOnlines**.entrySet()) {  
 **if** (kv.getKey().equals(message.getNameReserved())) {  
 **try** {  
 kv.getValue().writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
 }  
}  
  
**private void** sendAll(ChatMessage message) {  
 **for** (Map.Entry<String, ObjectOutputStream> kv : **mapOnlines**.entrySet()) {  
 **if** (!kv.getKey().equals(message.getName())) {  
 message.setAction(Action.***SEND\_ONE***);  
 **try** {  
 kv.getValue().writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
 }  
}

Когда пользователь завершает работу с сервером и отключается, сервер сообщает об этом в поле для сообщений с помощью метода disconnect():

**private void** disconnect(ChatMessage message, ObjectOutputStream output) {  
 **mapOnlines**.remove(message.getName());  
  
 message.setAction(Action.***SEND\_ONE***);  
  
 sendAll(message);  
  
 System.***out***.println(**"Пользователь "** + message.getName() + **" покинул сервер"**);  
}

Так проходит взаимодействие между сервером и клиентом в разработанном приложении.

# 6 Описание работы с приложением

## 6.1 Описание основных форм приложения

Основная форма приложение показана на рисунке 5.

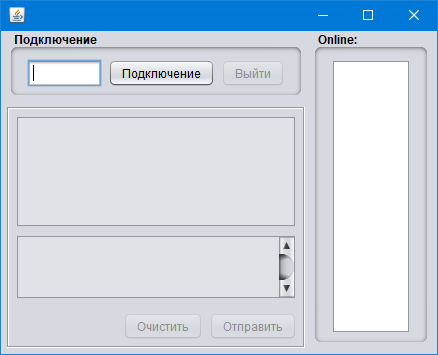


Рисунок 5 – Главная форма приложения

При успешном подключении клиентского приложения к серверу на экране отображается форма в которой отображается сообщение об успешном входе в чат(рисунок 6).

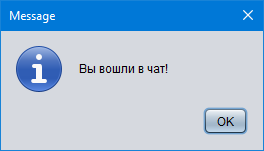


Рисунок 6 – Сообщение об успешном входе в чат

На рисунке 7 видно как происходит общение в чате.

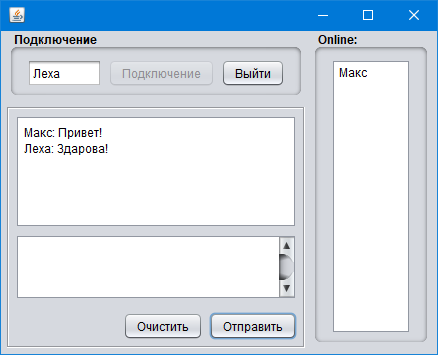


Рисунок 7 – Переписка пользователей

Когда пользователь отключается от сервера у него открывается информационное сообщение(рисунок 8).

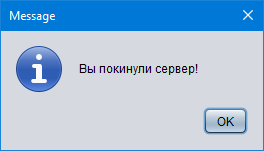


Рисунок 8 – Сообщение о выходе из чата

## 6.2 Руководство пользователя

Как только пользователь запускает клиентское приложение он попадает в главное окно приложения(рисунок 9).

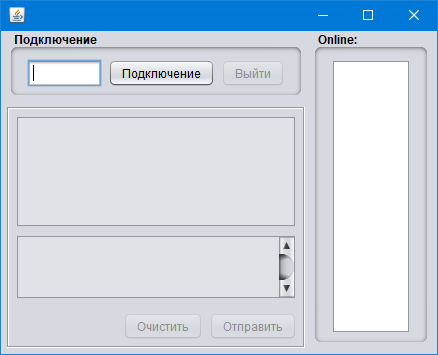


Рисунок 9 – Главное окно приложения

Чтобы подключиться к серверу пользователю необходимо указать свое имя в выделенном поле(рисунок 10) и нажать кнопку «Подключение».

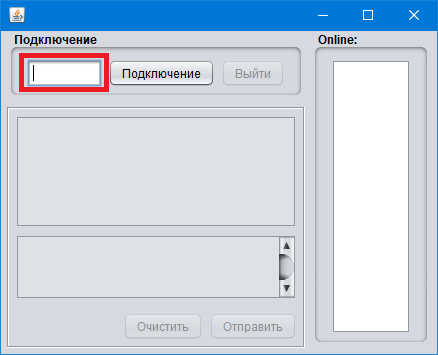
****

Рисунок 10 – Поле для имени пользователя

В поле 1(рисунок 11) отображается переписка пользователей.

Для того чтобы отправить сообщение необходимо в поле 2(рисунок 11) ввести сообщение и нажать на кнопку «Отправить». Кнопка «Очистить» очищает поле для ввода сообщений.

В поле 3(рисунок 11) можно увидеть пользователей, которые находятся в сети в данный момент.

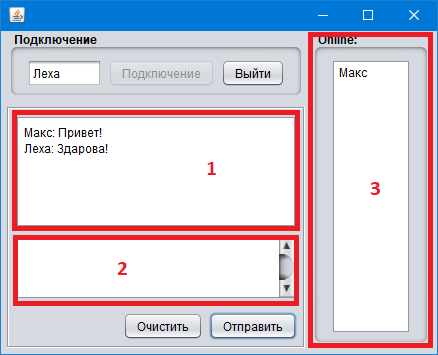


Рисунок 11 – Зоны главной формы

# Заключение

В данном курсовом проекте необходимо было спроектировать и разработать клиент-серверное приложение для групповой обмены сообщениями на объектно-ориентированном языке Java.

В результате выполнения курсового проектирования были выполнены следующие задачи:

* были рассмотрены подходы и принципы создания клиент-серверных приложений;
* изучены основы разработки клиент-серверных приложений;
* проанализированы особенности разработки клиент-серверных приложений на языке Java;
* спроектировано и разработано клиент-серверное приложение «Чат» на языке Java.

Разработанное сетевое приложение может быть использовано как для малого круга людей, в домашних условиях, так и на производстве небольшого предприятия. Программный продукт имеет дружественный интерфейс, не требующий специальной подготовки пользователей, и способствует более эффективному распределению и экономии времени.

# Список использованных источников

1 Вязовик, Н.А. Программирование на Java : учебное пособие / Н.А. Вязовик. — 2-е изд. — Москва : ИНТУИТ, 2016. — 603 с. — Текст : электронный // Электронно-библиотечная система «Лань» : [сайт]. — URL: https://e.lanbook.com/book/100405

2 Гуськова, О.И. Объектно ориентированное программирование в Java : учебное пособие / О. И. Гуськова. - Москва : МПГУ, 2018. - 240 с. - ISBN 978-5-4263-0648-6. - URL: <http://znanium.com/bookread2.php?book=1020593>

3 Дубаков, А.А. Введение в объектно-ориентированное программирование на Java : учебное пособие / А.А. Дубаков. — Санкт-Петербург : НИУ ИТМО, 2016. — 250 с. — Текст : электронный // Электронно-библиотечная система «Лань» : [сайт]. — URL: https://e.lanbook.com/book/110468

4 Интернет-программирование на Java: Пособие / Будилов В.А. - СПб:БХВ-Петербург, 2014. - 698 с. ISBN 978-5-9775-1931-1 - Режим доступа: <http://znanium.com/bookread2.php?book=940239>

5 Программирование на языке Java. Конспект лекций : учебное пособие / А.В. Гаврилов, С.В. Клименков, А.Е. Харитонова, Е.А. Цопа. — Санкт-Петербург : НИУ ИТМО, 2015. — 126 с. — Текст : электронный // Электронно-библиотечная система «Лань» : [сайт]. — URL: https://e.lanbook.com/book/91488

6 Хеффельфингер, Д. Разработка приложений Java ЕЕ 7 в NetBeans 8 / Дэвид Хеффельфингер ; пер. с англ. А.Н. Киселева. - Москва : ДМК Пресс, 2016. - 348 с. - ISBN 978-5-97060-329-1. - Текст : электронный. - URL: <https://new.znanium.com/read?id=341214>

# Приложение 1

Листинг программы

Главный класс сервера

**package** com.mron.app.service;  
  
**import** com.mron.app.bean.ChatMessage;  
**import** com.mron.app.bean.ChatMessage.Action;  
  
**import** java.io.IOException;  
**import** java.io.ObjectInputStream;  
**import** java.io.ObjectOutputStream;  
**import** java.net.ServerSocket;  
**import** java.net.Socket;  
**import** java.util.HashMap;  
**import** java.util.HashSet;  
**import** java.util.Map;  
**import** java.util.Set;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
**public class** Service {  
  
 **private** ServerSocket **serverSocket**;  
 **private** Socket **socket**;  
 **private** Map<String, ObjectOutputStream> **mapOnlines** = **new** HashMap<String, ObjectOutputStream>();  
  
 **public** Service() {  
 **try** {  
 **serverSocket** = **new** ServerSocket(5678);  
  
 System.***out***.println(**"Сервер запущен!"**);  
  
 **while** (**true**) {  
 **socket** = **serverSocket**.accept();  
  
 **new** Thread(**new** ListenerSocket(**socket**)).start();  
 }  
  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
  
 **private class** ListenerSocket **implements** Runnable {  
  
 **private** ObjectOutputStream **output**;  
 **private** ObjectInputStream **input**;  
  
 **public** ListenerSocket(Socket socket) {  
 **try** {  
 **this**.**output** = **new** ObjectOutputStream(socket.getOutputStream());  
 **this**.**input** = **new** ObjectInputStream (socket.getInputStream());  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
  
 @Override  
 **public void** run() {  
 ChatMessage message = **null**;  
 **try** {  
 **while** ((message = (ChatMessage) **input**.readObject()) != **null**) {  
 Action action = message.getAction();  
  
 **if** (action.equals(Action.***CONNECT***)) {  
 **boolean** isConnect = connect(message, **output**);  
 **if** (isConnect) {  
 **mapOnlines**.put(message.getName(), **output**);  
 sendOnlines();  
 }  
 } **else if** (action.equals(Action.***DISCONNECT***)) {  
 disconnect(message, **output**);  
 sendOnlines();  
 **return**;  
 } **else if** (action.equals(Action.***SEND\_ONE***)) {  
 sendOne(message);  
 } **else if** (action.equals(Action.***SEND\_ALL***)) {  
 sendAll(message);  
 }  
 }  
 } **catch** (IOException ex) {  
 ChatMessage cm = **new** ChatMessage();  
 cm.setName(message.getName());  
 disconnect(cm, **output**);  
 sendOnlines();  
 System.***out***.println(message.getName() + **" покинул чат!"**);  
 } **catch** (ClassNotFoundException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
 }  
  
 **private boolean** connect(ChatMessage message, ObjectOutputStream output) {  
 **if** (**mapOnlines**.size() == 0) {  
 message.setText(**"YES"**);  
 send(message, output);  
 **return true**;  
 }  
  
 **if** (**mapOnlines**.containsKey(message.getName())) {  
 message.setText(**"NO"**);  
 send(message, output);  
 **return false**;  
 } **else** {  
 message.setText(**"YES"**);  
 send(message, output);  
 **return true**;  
 }  
 }  
  
 **private void** disconnect(ChatMessage message, ObjectOutputStream output) {  
 **mapOnlines**.remove(message.getName());  
  
 message.setAction(Action.***SEND\_ONE***);  
  
 sendAll(message);  
  
 System.***out***.println(**"Пользователь "** + message.getName() + **" покинул сервер"**);  
 }  
  
 **private void** send(ChatMessage message, ObjectOutputStream output) {  
 **try** {  
 output.writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
  
 **private void** sendOne(ChatMessage message) {  
 **for** (Map.Entry<String, ObjectOutputStream> kv : **mapOnlines**.entrySet()) {  
 **if** (kv.getKey().equals(message.getNameReserved())) {  
 **try** {  
 kv.getValue().writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
 }  
 }  
  
 **private void** sendAll(ChatMessage message) {  
 **for** (Map.Entry<String, ObjectOutputStream> kv : **mapOnlines**.entrySet()) {  
 **if** (!kv.getKey().equals(message.getName())) {  
 message.setAction(Action.***SEND\_ONE***);  
 **try** {  
 kv.getValue().writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
 }  
 }  
  
 **private void** sendOnlines() {  
 Set<String> setNames = **new** HashSet<String>();  
 **for** (Map.Entry<String, ObjectOutputStream> kv : **mapOnlines**.entrySet()) {  
 setNames.add(kv.getKey());  
 }  
  
 ChatMessage message = **new** ChatMessage();  
 message.setAction(Action.***USERS\_ONLINE***);  
 message.setSetOnlines(setNames);  
  
 **for** (Map.Entry<String, ObjectOutputStream> kv : **mapOnlines**.entrySet()) {  
 message.setName(kv.getKey());  
 **try** {  
 kv.getValue().writeObject(message);  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(Service.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
 }  
}

Главный класс клиента

**package** com.mron.app.frame;  
  
**import** com.mron.app.bean.ChatMessage;  
**import** com.mron.app.bean.ChatMessage.Action;  
**import** com.mron.app.service.ClientService;  
**import** java.io.IOException;  
**import** java.io.ObjectInput;  
**import** java.io.ObjectInputStream;  
**import** java.io.ObjectOutputStream;  
**import** java.net.Socket;  
**import** java.util.Set;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
**import** javax.swing.\*;  
  
**public class** ClientFrame **extends** javax.swing.JFrame {  
  
 **private** Socket **socket**;  
 **private** ChatMessage **message**;  
 **private** ClientService **service**;  
  
 **public** ClientFrame() {  
 initComponents();  
 }  
  
 **private class** ListenerSocket **implements** Runnable {  
  
 **private** ObjectInputStream **input**;  
  
 **public** ListenerSocket(Socket socket) {  
 **try** {  
 **this**.**input** = **new** ObjectInputStream(socket.getInputStream());  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(ClientFrame.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
  
 @Override  
 **public void** run() {  
 ChatMessage message = **null**;  
 **try** {  
 **while** ((message = (ChatMessage) **input**.readObject()) != **null**) {  
 Action action = message.getAction();  
  
 **if** (action.equals(Action.***CONNECT***)) {  
 connected(message);  
 } **else if** (action.equals(Action.***DISCONNECT***)) {  
 disconnected();  
 **socket**.close();  
 } **else if** (action.equals(Action.***SEND\_ONE***)) {  
 System.***out***.println(**"::: "** + message.getText() + **" :::"**);  
 receive(message);  
 } **else if** (action.equals(Action.***USERS\_ONLINE***)) {  
 refreshOnlines(message);  
 }  
 }  
 } **catch** (IOException ex) {  
 Logger.*getLogger*(ClientFrame.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 } **catch** (ClassNotFoundException ex) {  
 Logger.*getLogger*(ClientFrame.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  
 }  
 }  
 }  
  
 **private void** connected(ChatMessage message) {  
 **if** (message.getText().equals(**"NO"**)) {  
 **this**.**txtName**.setText(**""**);  
 JOptionPane.*showMessageDialog*(**this**, **"Соединение не установлено!\nПопробуйте еще раз с новым именем."**);  
 **return**;  
 }  
  
 **this**.**message** = message;  
 **this**.**btnConnect**.setEnabled(**false**);  
 **this**.**txtName**.setEditable(**false**);  
  
 **this**.**btnDisconnect**.setEnabled(**true**);  
 **this**.**txtAreaSend**.setEnabled(**true**);  
 **this**.**txtAreaReceive**.setEnabled(**true**);  
 **this**.**btnSend**.setEnabled(**true**);  
 **this**.**btnClear**.setEnabled(**true**);  
  
 JOptionPane.*showMessageDialog*(**this**, **"Вы вошли в чат!"**);  
 }  
  
 **private void** disconnected() {  
  
 **this**.**btnConnect**.setEnabled(**true**);  
 **this**.**txtName**.setEditable(**true**);  
  
 **this**.**btnDisconnect**.setEnabled(**false**);  
 **this**.**txtAreaSend**.setEnabled(**false**);  
 **this**.**txtAreaReceive**.setEnabled(**false**);  
 **this**.**btnSend**.setEnabled(**false**);  
 **this**.**btnClear**.setEnabled(**false**);  
   
 **this**.**txtAreaReceive**.setText(**""**);  
 **this**.**txtAreaSend**.setText(**""**);  
  
 JOptionPane.*showMessageDialog*(**this**, **"Вы покинули сервер!"**);  
 }  
  
 **private void** receive(ChatMessage message) {  
 **this**.**txtAreaReceive**.append(message.getName() + **": "** + message.getText() + **"\n"**);  
 }  
  
 **private void** refreshOnlines(ChatMessage message) {  
 System.***out***.println(message.getSetOnlines().toString());  
   
 Set<String> names = message.getSetOnlines();  
   
 names.remove(message.getName());  
   
 String[] array = (String[]) names.toArray(**new** String[names.size()]);  
   
 **this**.**listOnlines**.setListData(array);  
 **this**.**listOnlines**.setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);  
 **this**.**listOnlines**.setLayoutOrientation(JList.***VERTICAL***);  
 }  
  
**private void** btnConnectActionPerformed(java.awt.event.ActionEvent evt) {*//GEN-FIRST:event\_btnConnectActionPerformed* String name = **this**.**txtName**.getText();  
  
 **if** (!name.isEmpty()) {  
 **this**.**message** = **new** ChatMessage();  
 **this**.**message**.setAction(Action.***CONNECT***);  
 **this**.**message**.setName(name);  
  
 **this**.**service** = **new** ClientService();  
 **this**.**socket** = **this**.**service**.connect();  
  
 **new** Thread(**new** ListenerSocket(**this**.**socket**)).start();  
  
 **this**.**service**.send(**message**);  
 }  
 }*//GEN-LAST:event\_btnConnectActionPerformed* **private void** btnDisconnectActionPerformed(java.awt.event.ActionEvent evt) {*//GEN-FIRST:event\_btnDisconnectActionPerformed* ChatMessage message = **new** ChatMessage();  
 message.setName(**this**.**message**.getName());  
 message.setAction(Action.***DISCONNECT***);  
 **this**.**service**.send(message);  
 disconnected();  
 }*//GEN-LAST:event\_btnDisconnectActionPerformed* **private void** btnClearActionPerformed(java.awt.event.ActionEvent evt) {*//GEN-FIRST:event\_btnClearActionPerformed* **this**.**txtAreaSend**.setText(**""**);  
 }*//GEN-LAST:event\_btnClearActionPerformed* **private void** btnSendActionPerformed(java.awt.event.ActionEvent evt) {*//GEN-FIRST:event\_btnSendActionPerformed* String text = **this**.**txtAreaSend**.getText();  
 String name = **this**.**message**.getName();  
   
 **this**.**message** = **new** ChatMessage();  
   
 **if** (**this**.**listOnlines**.getSelectedIndex() > -1) {  
 **this**.**message**.setNameReserved((String) **this**.**listOnlines**.getSelectedValue());  
 **this**.**message**.setAction(Action.***SEND\_ONE***);  
 **this**.**listOnlines**.clearSelection();  
 } **else** {  
 **this**.**message**.setAction(Action.***SEND\_ALL***);  
 }  
   
 **if** (!text.isEmpty()) {  
 **this**.**message**.setName(name);  
 **this**.**message**.setText(text);  
  
 **this**.**txtAreaReceive**.append(name + **": "** + text + **"\n"**);  
   
 **this**.**service**.send(**this**.**message**);  
 }  
   
 **this**.**txtAreaSend**.setText(**""**);  
 }*//GEN-LAST:event\_btnSendActionPerformed  
  
 // Variables declaration - do not modify//GEN-BEGIN:variables* **private** javax.swing.JButton **btnConnect**;  
 **private** javax.swing.JButton **btnSend**;  
 **private** javax.swing.JButton **btnClear**;  
 **private** javax.swing.JButton **btnDisconnect**;  
 **private** javax.swing.JPanel **jPanel1**;  
 **private** javax.swing.JPanel **jPanel2**;  
 **private** javax.swing.JPanel **jPanel3**;  
 **private** javax.swing.JScrollPane **jScrollPane1**;  
 **private** javax.swing.JScrollPane **jScrollPane2**;  
 **private** javax.swing.JScrollPane **jScrollPane3**;  
 **private** javax.swing.JList **listOnlines**;  
 **private** javax.swing.JTextArea **txtAreaReceive**;  
 **private** javax.swing.JTextArea **txtAreaSend**;  
 **private** javax.swing.JTextField **txtName**;  
 *// End of variables declaration//GEN-END:variables*}

# Приложение 2

Проверка на оригинальность

