

**Содержание:**

[Введение 5](#_Toc136931358)

[1. Постановка задачи 7](#_Toc136931359)

[2. Выбор решения 8](#_Toc136931360)

[3. Описание программы 11](#_Toc136931361)

[4. Описание способа организации пользовательского интерфейса 13](#_Toc136931362)

[5. Описание результатов работы программы 16](#_Toc136931363)

[Заключение 27](#_Toc136931364)

[Список используемой литературы 28](#_Toc136931365)

[Приложение А. Листинг программы 29](#_Toc136931366)

[Клиент: 29](#_Toc136931367)

[Сервер: 52](#_Toc136931368)

[Приложение Б. UML-диаграммы 74](#_Toc136931369)

[Диаграмма вариантов использования 74](#_Toc136931370)

[Диаграмма классов 75](#_Toc136931371)

[Клиент: 75](#_Toc136931372)

[Сервер: 76](#_Toc136931373)

[Диаграмма деятельности 77](#_Toc136931374)

[Диаграмма развёртывания 78](#_Toc136931375)

[Диаграмма последовательности 79](#_Toc136931376)

# **Введение**

Современные технологии позволяют людям обмениваться различными видами информации, такими как графика, звук и другие данные, с легкостью и по всему миру.

В данной работе используется язык программирования Java, который является широко используемым, а в качестве среды разработки - IntelliJ IDEA.

Java является одним из самых популярных языков программирования, который имеет множество достоинств. Язык Java является объектно-ориентированным языком, что позволяет создавать модульные программы, исходный код который может использоваться многократно. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. Кроме того, Java имеет богатый API, который позволяет разработчикам создавать различные приложения, включая игры, работу с графикой и звуком.

Одной из особенностей языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. Затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine). Подобная архитектура обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, благодаря чему подобные программы без перекомпиляции могут выполняться на различных платформах - Windows, Linux, Mac OS и т.д. Для каждой из платформ может быть своя реализация виртуальной машины JVM, но каждая из них может выполнять один и тот же код.

Java также имеет встроенную систему сборки мусора (garbage collector), которая автоматически освобождает память, занятую объектами, которые больше не используются в программе. Это упрощает процесс разработки и уменьшает вероятность ошибок, связанных с управлением памятью.

В целом, Java является мощным и гибким языком программирования, который широко используется для разработки различных приложений, включая веб-приложения, мобильные приложения, игры и многое другое.

В данном курсовом проекте используется клиент-серверная архитектура, которая является распространенной моделью в компьютерных сетях, предусматривающая разделение процессов предоставления услуг и отправки запросов на них на разных компьютерах в сети. Каждый компьютер выполняет свои задачи независимо от других, что позволяет улучшить производительность и обеспечить более эффективное использование ресурсов.

Целью курсового проекта является закрепление знаний и приобретение практических навыков разработки многомодульных приложений на языке Java, также требуется умение самостоятельно работать со справочной и учебной литературой.

Для закрепления изученного материала в ходе курса программирования на Java в качестве курсового проекта была выбрана тема «MP3-проигрыватель».

# **1. Постановка задачи**

Нужно разработать клиент-серверное многомодульное приложение «MP3-проигрыватель». ОС – Windows. Язык программирования: Java. Среда разработки IntelliJ IDEA. Графическая библиотека - Java Swing.

Приложение должно обладать графическим интерфейсом и использовать следующие технологии:

1) Java Collections Framework

2) Механизм обработки исключительных ситуаций

3) Java Stream API

4) Java Multithreading

5) Сетевое взаимодействие

6) Java Swing

Функции сервера:

1) Обработка данных от пользователей.

2) Отправка информации в соответствии с принятыми данными.

3) Своевременное окончание сессии.

Функции клиента:

1) Реализация интерфейса, и его изменение в соответствии с ответами сервера.

2) Отправка данных серверу.

# **2. Выбор решения**

Работу над приложением клиент-серверного проигрывателя было решено начать с изучения и организации клиент-серверного взаимодействия.

Изучение данной организации позволило выявить оптимальный вариант разработки. Элементы Socket и ServerSocket были помещены в отдельную структуру (SocketPair), представляющую собой контейнер с описанием соединения сокетов. В данном контейнере, кроме элементов Socket и ServerSocket, должны присутствовать: два потока, отвечающие за обмен данными, флаг занятости контейнера, информация об IP клиента и порте, на котором располагается соединение.

Далее, необходимо было решить проблему с подключением нескольких клиентов. Было решено выделить отдельный порт (8000) для прослушивания входящих заявок от клиентов. Данный механизм был назван «гостевым сокетом». Клиент, при первом подключении, обращается, именно, к данному сокету и получает от него в ответ номер порта, на котором сервер сможет установить соединение. Также, в данном сокете есть механизм «обратной связи», позволяющий отсекать клиентов, которые занимают его слишком долго и не дают получить порты другим клиентам.

После организации сетевых пар, необходимо было решить вопрос с воспроизведением полученных файлов. Необходимо было взять самое лёгкое и компактное решение. Данным решением выступила библиотека vlcj, используемая в очень распространённом проигрывателе VLC. Выбор был обусловлен простотой использования данной библиотеки, а также её способностью к быстрой замене. Даже если программа потеряет исходники библиотеки vlcj, она всегда сможет подключить данные исходники из проигрывателя в системе и продолжить стабильную работу.

Следующим вопросом стала регистрация пользователя. Допустим, администратор сервера хочет создать ограниченный круг доступа. Для этого, ему необходимо создать ключи (папки), которые являются директориями доступа. Если пользователь при входе на сервер, введёт данные одной из этих папок, он спокойно подключится к серверу. В ином случае, доступ будет закрыт.

Теоритический путь решения был выбран. Теперь, необходимо перейти к практической организации.

Интерфейс пользователя был реализован с помощью Java Swing.

Java Swing - это набор библиотек и классов, предоставляющих разработчикам возможность создания графических интерфейсов пользователя (GUI) на языке Java.

Java Swing был разработан как альтернатива ранее использовавшимся в Java AWT (Abstract Window Toolkit) для создания приложений с графическим интерфейсом пользователя. Один из основных преимуществ Swing - это множество компонентов интерфейса пользователя, которые могут быть созданы и настроены графически на экране.

Некоторые из компонентов Swing включают:

- Кнопки (Button), поля ввода (TextField), переключатели (Checkbox) и другие элементы управления формами.

- Панели (Panel), которые могут быть использованы для создания композитных компонентов, таких как таблицы и списки.

- Метки (Label) для отображения текста, значков и изображений.

- Списки (List), таблицы (Table) и деревья (Tree) для отображения набора данных.

- Диалоги и окна (Dialogs) для отображения сообщений пользователю, запрашивающих информацию или подтверждения.

Swing также предоставляет систему менеджеров макета, позволяющую легко управлять компонентами интерфейса пользователя и гибко управлять расположением элементов при изменении размера или ориентации окна.Программное обеспечение JetBrains IntelliJ IDEA – это ведущая среда быстрой разработки на языке Java.

В качестве среды разработки была выбрана IntelliJ IDEA.

Программное обеспечение JetBrains IntelliJ IDEA – это ведущая среда быстрой разработки на языке Java.

IntelliJ IDEA имеет множество возможностей и инструментов для упрощения и ускорения процесса программирования. Она предлагает широкий набор функций, таких как:

- Редактор кода с подсветкой синтаксиса, автодополнением, проверкой ошибок и другими возможностями.

- Встроенный дебаггер, который позволяет отслеживать выполнение программы и выявлять ошибки.

- Система контроля версий, в том числе Git, SVN и Mercurial.

- Интегрированные инструменты для создания графических пользовательских интерфейсов.

- Поддержка автоматической генерации кода, объединяющая в себе высокоуровневые операции с применением шаблонов кода.

- Возможность расширения функциональности плагинами из множества доступных.

В IntelliJ IDEA также имеются функции, которые ускоряют написание кода, такие как Live templates, которые позволяют генерировать шаблоны кода для быстрого написания часто используемых конструкций на языке программирования.

Помимо этого, IntelliJ IDEA предоставляет возможности для автоматической проверки качества кода, поиска ошибок, оптимизации кода и многое другое.

IntelliJ IDEA является одним из самых популярных и мощных инструментов для разработки на языке программирования Java, известной своей оптимизацией и интуитивно понятным интерфейсом.

В ходе разработки приложения использовался следующий набор технологий:

1) Java - строго типизированный ООП язык общего назначения

2) IntelliJ IDEA – интегрированная среда разработки

3) Java Collections Framework

4) Механизм обработки исключительных ситуаций

5) Java Stream API

6) Java Multithreading

7) Сетевое взаимодействие

# **3. Описание программы**

Данная программа состоит из двух модулей: клиента и сервера, которые взаимодействуют между собой. Взаимодействие программ заключается в передаче данных с сервера к клиенту. Интерфейс программы интуитивно понятен и требует лишь небольшого знания английского языка.

Последующее описание модулей будет сопровождаться подписями **Server:** и **Client:**, для лучшего понимания принадлежности модуля.

**Server: Main.java –** класс, в которомначинается работа сервера. Вызывается программный и внутренний интерфейс, а также осуществляется контроль за работой сервера.

**Server: Audio.java –** класс, вызывающий окно воспроизведения аудио. Содержит одну функцию PlayingMp4File(), которая осуществляет вызов данного окна.

**Server: SContainer.java** – класс, содержащий в себе контейнерные формы (Название списка, список аудио, добавить аудио, удалить аудио). Осуществляет и поддерживает работу данных форм.

**Server: SerSocket.java** – класс, содержащий в себе реализацию связанной пары сокетов (отдельная структура **SocketPair**) и контейнеров под данные сокеты. Осуществляет автоматизированное функционирование контейнеров с сокетами.

**Server: ServerWindow.java –** класс, содержащий в себе реализацию графического интерфейса окна сервера.

**Server: Folder.java –** класс, позволяющий быстро осуществлять работу с файлами и обладающий для этого необходимыми функциями (создание файла, перемещение файла, удаление файла).

**Client: Main.java –** класс, в которомначинается работа клиента. Вызывается программный и внутренний интерфейс, а также осуществляется контроль за работой клиента.

**Client: Audio.java –** класс, вызывающий окно воспроизведения аудио. Содержит одну функцию PlayingMp4File(), которая осуществляет вызов данного окна.

**Client: SContainer.java** – класс, содержащий в себе контейнерные формы (Название списка, список аудио, добавить аудио, удалить аудио). Осуществляет и поддерживает работу данных форм.

**Client: CliSocket.java** – класс, содержащий в себе реализацию архитектуры, отвечающей за связь с сервером. Осуществляет автоматизированное функционирование клиентского сокета, связывание с сервером, получение файлов.

**Client: ClientWindow.java –** класс, содержащий в себе реализацию графического интерфейса окна клиента.

**Client: LoginWindow.java –** класс, содержащий в себе реализацию графического интерфейса окна запроса логина при входе на сервер.

**Client: Folder.java –** класс, позволяющий быстро осуществлять работу с файлами и обладающий для этого необходимыми функциями (создание файла, перемещение файла, удаление файла).

# **4. Описание способа организации пользовательского интерфейса**

В качестве среды разработки была выбрана программа IntelliJ IDEA. Программа  быстро приобрела популярность, как первая среда для Java с широким набором интегрированных инструментов для [рефакторинга](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3), которые позволяли программистам быстро реорганизовывать исходные тексты программ.

К преимуществам среды можно отнести:

* Эргономика разработчика.

При каждом принятии решения о проектировании и реализации мы учитываем риск прерывания потока разработчиков и делаем все возможное, чтобы устранить или минимизировать его.

IDE следует вашему контексту и автоматически вызывает соответствующие инструменты.

* Глубокий интеллект.

После того, как IntelliJ IDEA проиндексировала ваш исходный код, он предлагает быстрый и интеллектуальный опыт, предоставляя соответствующие предложения в каждом контексте: мгновенное и умное завершение кода, анализ кода на лету и надежные инструменты рефакторинга.

* Готовый опыт.

Критически важные инструменты, такие как интегрированные системы контроля версий и широкий спектр поддерживаемых языков и фреймворков, все под рукой - никаких хлопот с плагинами.

* Помощь для конкретной платформы.

Хотя IntelliJ IDEA является IDE для Java, она также понимает и обеспечивает интеллектуальную помощь в кодировании для большого разнообразия других языков, таких как SQL, JPQL, HTML, JavaScript и т. Д., Даже если языковое выражение вводится в строковый литерал в вашем Java-коде.

**Окно клиента**:

Основное окно клиента имеет 3 кнопки, 2 таблицы и 3 текстовых поля, в качестве графической оснастки.

Кнопки в окне имеют разные назначения, но интуитивно понятные надписи позволяют понять пользователю, какие функции они выполняют: **Add Music** – добавить аудио в локальный репозиторий, **Delete Music** – удалить аудио из локального репозитория, **Connect** – построить соединение с сервером.

Таблицы также имеют разные выводимые значения. Их назначение можно понять из текстовых полей, расположенных над ними:

1.**Local Music**. Таблица выводит список аудио, расположенных в локальном репозитории. При двойном нажатии на элемент списка, его можно воспроизвести.

2.**Server Music**. Таблица выводит список аудио, расположенных в серверном репозитории. При двойном нажатии на элемент списка, его можно загрузить с сервера в локальный репозиторий. Однако, если нет соединения с сервером, список выведен не будет

Последнее текстовое поле отвечает за информирование пользователя о состоянии работы клиента.

**Окно сервера**:

Основное окно сервера имеет 2 кнопки, 2 таблицы и 3 текстовых поля, в качестве графической оснастки.

Кнопки в окне имеют разные назначения, но интуитивно понятные надписи позволяют понять пользователю, какие функции они выполняют: **Add Music** – добавить аудио в сетевой репозиторий, **Delete Music** – удалить аудио из сетевого репозитория.

Таблицы также имеют разные выводимые значения. Их назначения можно понять из текстовых полей, расположенных над ними:

1.**Local Music**. Таблица выводит список аудио, расположенных в сетевом репозитории. При двойном нажатии на элемент списка, его можно воспроизвести.

2.**Clients**. Таблица выводит список клиентов, подключенных к серверу.

Последнее текстовое поле отвечает за информирование пользователя о состоянии работы сервера.

**Окно регистрации:**

Данное окно появляется при попытке подключится к серверу. Оно имеет 1 кнопку, 1 текстовое поле и 1 поле ввода в качестве графической оснастки.

Кнопка Login в окне отвечает за подтверждение ввода пользователя и отправку запроса на подключение серверу.

Поле ввода позволяет ввести пользователю логин.

Последнее текстовое поле отвечает за информирование пользователя о корректности ввода и ответе от сервера.

**Окно воспроизведения:**

Данное окно появляется при попытке проиграть файл. Оно имеет 1 экран проигрывания и 12 кнопок в качестве графической оснастки.

Экран проигрывания осуществляет вывод на экран проигрываемой композиции.

Кнопки осуществляют полный функционал кнопок проигрывателя. Ниже идёт пояснение к их функционалу:

* Pause – остановить (возобновить) проигрывание
* Символ «минус» – уменьшить громкость
* Символ «плюс» - увеличить громкость
* Previous – проиграть предыдущий файл
* Next – проиграть следующий файл
* <<10 – перемотка на 10 секунд назад
* Play – возобновление проигрывания
* >>10 – перемотка на 10 секунд вперёд
* Stop – прерывание проигрывания
* << - уменьшить скорость проигрывания в 2 раза
* >> - увеличить скорость проигрывания в 2 раза
* Reset – начать проигрывание заново

# **5. Описание результатов работы программы**

Для полной демонстрации работы программы необходимо описать весь спектр действий, производимых в ней.

Начнём описание с запуска клиента программы (рис.1).

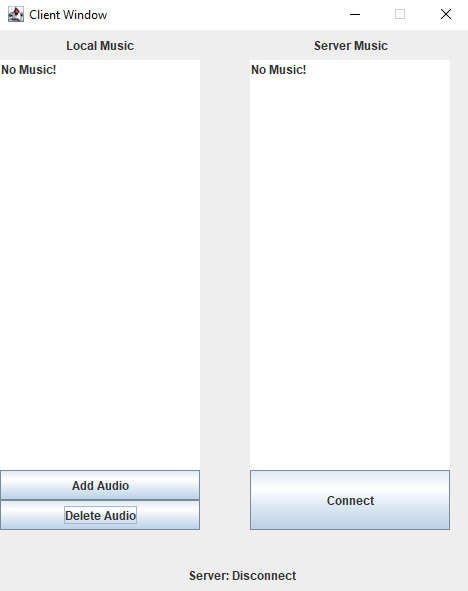


Рисунок 1 – Запуск клиента

Клиент представляет собой полнофункциональное приложение даже без участия сервера. Он позволяет проигрывать файлы в отдельном окне. Для демонстрации нам необходимо добавить запускаемый файл в локальный репозиторий. Добавление осуществляется при нажатии на кнопку Add Audio (рис. 2).

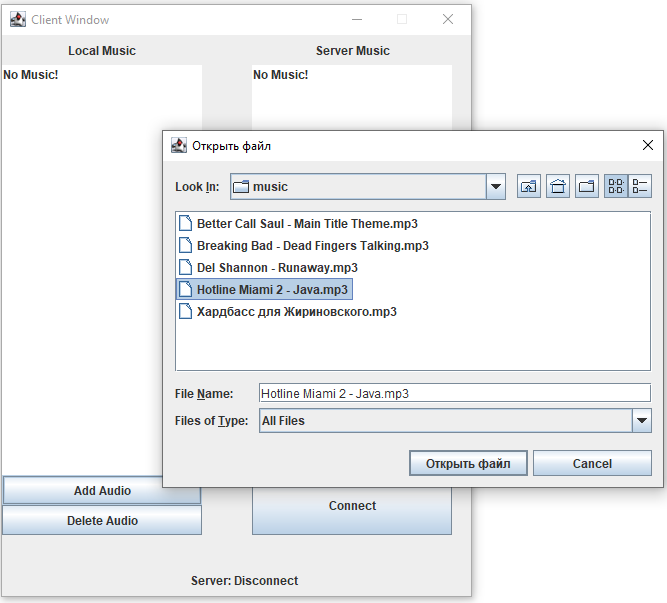


Рисунок 2 – Добавление элемента в локальный репозиторий

После добавления элемента он отобразится в локальном репозитории и с ним можно будет взаимодействовать (рис. 3).

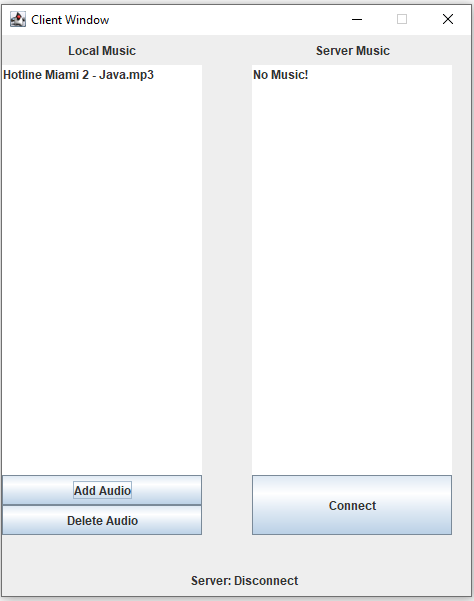


Рисунок 3 – Отображение элемента в локальном репозитории

Взаимодействие с элементом происходит при двойном нажатии на него. При этом действии, начнётся проигрывание (рис. 4).

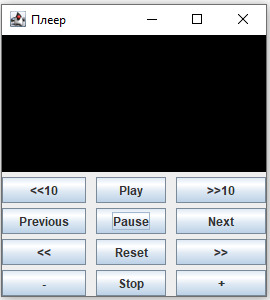


Рисунок 4 – Проигрывание файла

Возможно удалить файл из локальный директории при помощи нажатия на кнопку Delete Audio (рис. 5, рис. 6).

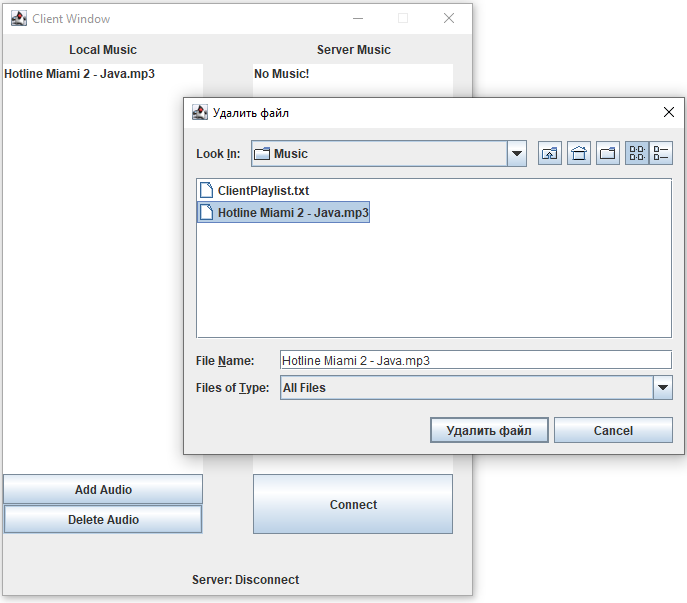


Рисунок 5 – Попытка удаления файла из локальной директории

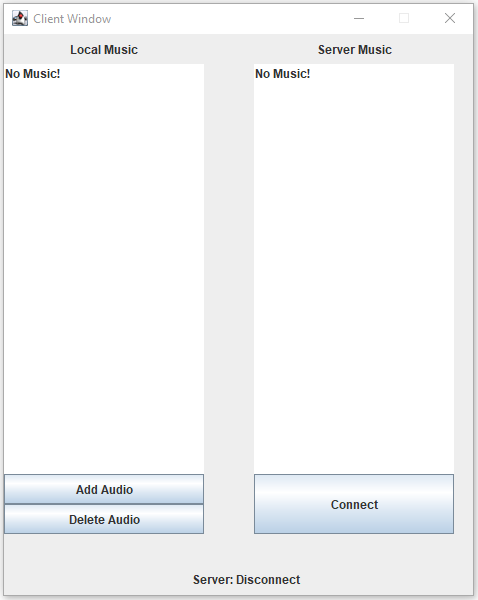


Рисунок 6 – Результат удаления из локального репозитория

Однако это не весь функционал программы. Полноценный спектр функций раскрывается при запуске сервервера (рис. 7).

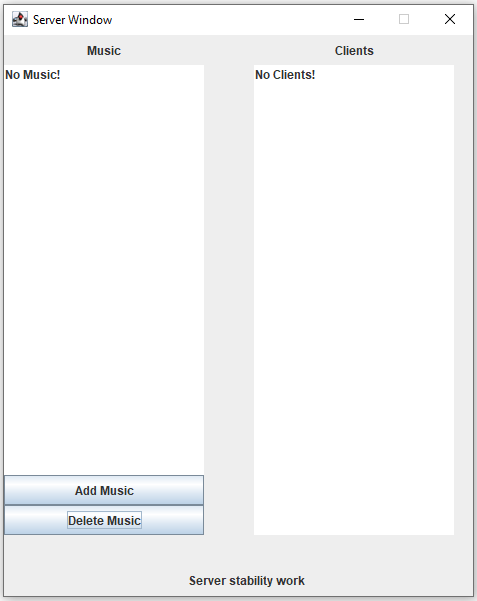


Рисунок 7 – Запуск сервера

Исходя из полученных данных, на сервере в данный момент нет данных. Для работы сервера необходимо их загрузить. Загрузка и удаление файлов с сервера происходит аналогично клиенту поэтому, данные шаги не будут рассмотрены детально (рис. 8).

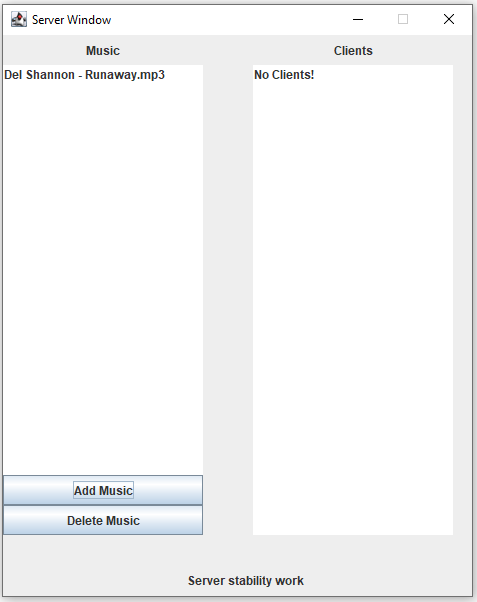


Рисунок 8 – Загрузка данных на сервер

Теперь необходимо подключиться клиенту для получения данных сервера. Чтобы выполнить подключение, нужно нажать кнопку Connect в клиенте. После этого будет предложено ввести логин (рис. 9).

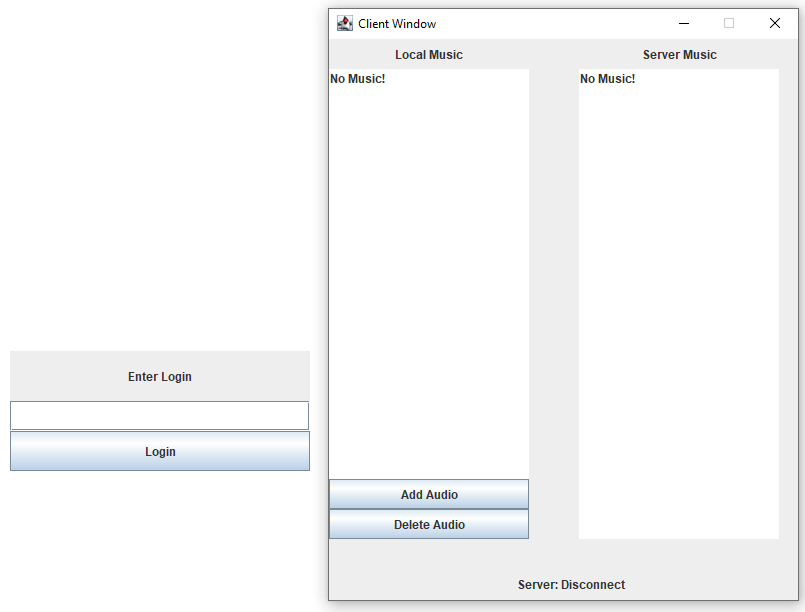


Рисунок 9 – Подключение к серверу

Необходимо ввести логин, который был выдан администратором сервера, иначе сервер не позволит к нему подключиться (рис. 10).

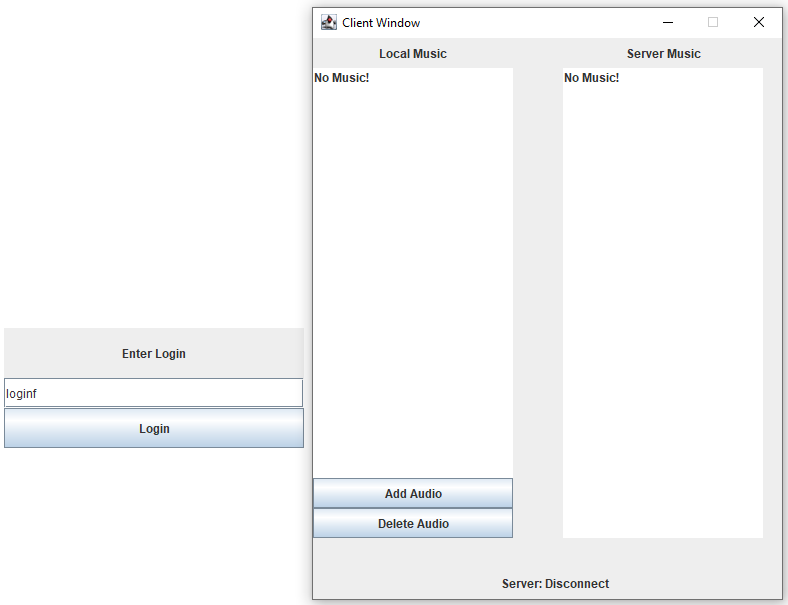


Рисунок 10 – Неудачное подключение к серверу

В случае ввода правильного логина сервер будет автоматически связан с клиентом на протяжении всей работы клиента (рис. 11).

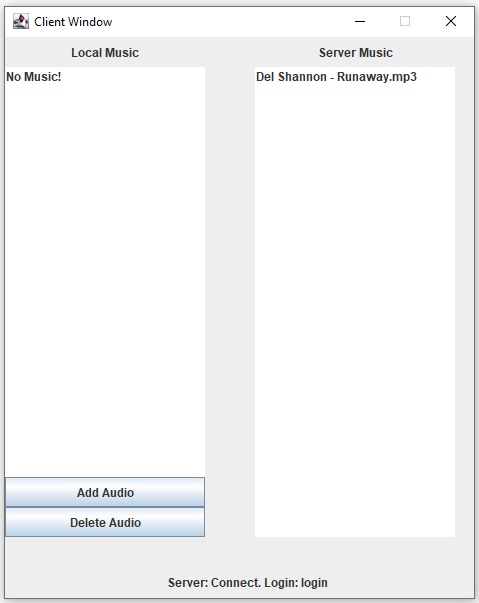


Рисунок 11 – Удачное подключение к серверу

После подключения к серверу на самом сервере стал отображаться подключенный клиент и какой логин он использует (рис. 12).

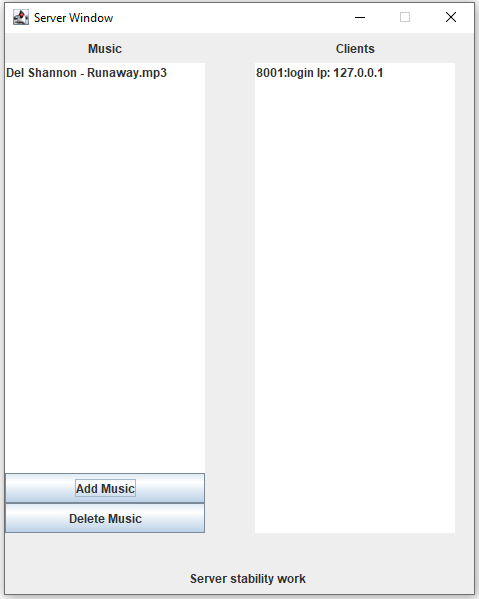


Рисунок 12 – Отображение клиента, подключенного к серверу

Также, после подключения клиента, мы можем выбрать любой элемент, лежащий на сервере и загрузить его. Для этого, необходимо дважды нажать на интересущий нас элемент и он переместится в локальный репозиторий после загрузки (рис. 13).

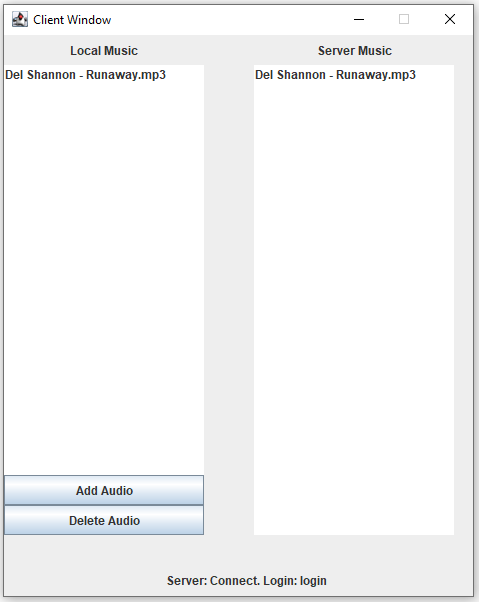


Рисунок 13 – Результат загрузки с сервера

Для проверки работоспособности, необходимо запустить файл, скачанный с сервера (рис. 14).

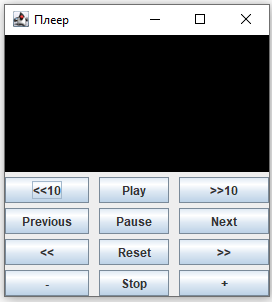


Рисунок 14 – Результат проверки скачанного файла

По итогам описания работы программы, было показано, что программа работает корректно и понятно.

# **Заключение**

В ходе данной курсовой работы было разработано клиент-серверное многомодульное приложение «MP3-проигрыватель» с пользовательским интерфейсом в среде IntelliJ IDEA. Использованы такие технологии как: Java Collections Framework, механизм обработки исключительных ситуаций, Java Stream API, Java Multithreading, Java Socket и Java ServerSocket, а также Java Swing.

Данные обрабатываются корректно, возникающие ошибки отлавливаются при помощи конструкции try catch. Для избегания возникновения ожидания при подключении к серверу нескольких клиентов используется многопоточность.

При выполнении данной курсовой работы были получены навыки разработки программ на языке Java.

Были освоены навыки разработки графического интерфейса с использованием Java Swing. Изучены технологии Socket. Получены навыки разработки клиент-серверного приложения.

# **Список используемой литературы**

1. "Java. Руководство для начинающих", Герберт Шилдт – ООО «Диалектика», 2019 – 816 с.

2. "Java. Полное руководство, десятое издание", Герберт Шилдт — ООО «Диалектика», 2018 — 1488 с.

3. "Swing: Эффектные пользовательские интерфейсы", Иван Портянкин — Издательство «Лори», 2011 — 591 с.

4. "Java. Эффективное программирование. 3-е изд.", Джошуа Блох — ООО «Диалектика», 2019 — 464 с.

5. "Алгоритмы на Java 4 — е издание", Роберт Сэдгевик, Кевин Уэйн - ООО «Диалектика», 2019 – 848 с.

# **Приложение А. Листинг программы**

# **Клиент:**

Audio.java

import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import javax.swing.\*;  
  
import uk.co.caprica.vlcj.component.EmbeddedMediaPlayerComponent;  
  
//Класс окна с воспроизводимым внутри аудио  
public class Audio {  
  
 private JFrame audioFrame; //Экран для вывода аудио (проигрыватель)  
 private EmbeddedMediaPlayerComponent mediaPlayerComponent; //Проигрыватель  
 private String currentPath; //Текущий путь к файлу  
 private String currentFile; //Текущий проигрываемый файл  
 private JButton playButton;  
 private JButton pauseButton;  
 private JButton nextButton;  
 private JButton previousButton;  
 private JPanel panel;  
 private JPanel panelMedia;  
 private JButton stopButton;  
 private JButton resetButton;  
 private JButton fastForwardButton;  
 private JButton rewindButton;  
 private JButton volumeUpButton;  
 private JButton volumeDownButton;  
 private JButton skipForwardButton;  
 private JButton skipBackwardButton;  
  
 Audio(DefaultListModel<String> musiclist)  
 {  
 audioFrame = new JFrame("Плеер");  
 audioFrame.setVisible(true);  
 audioFrame.setSize(280, 300);  
 audioFrame.setLocation(120, 100);  
 audioFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);  
 audioFrame.setContentPane(panel);//задаёт содержимое окна с рамкой  
  
 panelMedia = new JPanel();  
 mediaPlayerComponent = new EmbeddedMediaPlayerComponent();  
 panelMedia.add(mediaPlayerComponent);  
  
 audioFrame.setContentPane(mediaPlayerComponent);  
 audioFrame.setContentPane(new JPanel(new BorderLayout()));  
 audioFrame.getContentPane().add(mediaPlayerComponent, BorderLayout.CENTER);  
 audioFrame.getContentPane().add(panel, BorderLayout.SOUTH);  
 audioFrame.setVisible(false);  
  
  
 //Листенеры//  
  
 //Кнопка закрытия окна  
 audioFrame.addWindowListener(new WindowAdapter() {  
 @Override  
 public void windowClosing(WindowEvent e) {  
 mediaPlayerComponent.release();  
 audioFrame.setVisible(false);  
 }  
 });  
  
 //Кнопка Play  
 playButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().play();  
 }  
 });  
  
 //Кнопка Pause  
 pauseButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().pause();  
 }  
 });  
  
 //Кнопка Next  
 nextButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
  
 int curindex = 0;  
 String data = "";  
  
 for(;curindex < musiclist.size(); curindex++)  
 {  
 data = musiclist.getElementAt(curindex);  
 if(data.compareTo(audioFrame.getName()) == 0)  
 {  
 if(curindex == musiclist.size()-1)  
 {  
 data = audioFrame.getName();  
 break;  
 }  
 else  
 {  
 data = musiclist.getElementAt(curindex+1);  
 break;  
 }  
 }  
 }  
  
 audioFrame.setName(data);  
 mediaPlayerComponent.getMediaPlayer().stop();  
 mediaPlayerComponent.getMediaPlayer().playMedia("Client/Music/" + data);  
 }  
 });  
  
 //Кнопка Previous  
 previousButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
  
 int curindex = 0;  
 String data = "";  
  
 for(;curindex < musiclist.size(); curindex++)  
 {  
 data = musiclist.getElementAt(curindex);  
 if(data.compareTo(audioFrame.getName()) == 0)  
 {  
 if(curindex == 0)  
 {  
 data = audioFrame.getName();  
 break;  
 }  
 else  
 {  
 data = musiclist.getElementAt(curindex-1);  
 break;  
 }  
 }  
 }  
  
 audioFrame.setName(data);  
 mediaPlayerComponent.getMediaPlayer().stop();  
 mediaPlayerComponent.getMediaPlayer().playMedia("Client/Music/" + data);  
 }  
 });  
  
 //Кнопка Stop  
 stopButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().stop();  
 }  
 });  
  
 //Кнопка Reset  
 resetButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setTime(0);  
 }  
 });  
  
 //Кнопка >> (ускорение в 2 раза)  
 fastForwardButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setRate(mediaPlayerComponent.getMediaPlayer().getRate()\*2);  
 }  
 });  
  
 //Кнопка << (замеделние в 2 раза)  
 rewindButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setRate(mediaPlayerComponent.getMediaPlayer().getRate()/2);  
 }  
 });  
  
 //Кнопка + (увеличение громкости)  
 volumeUpButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setVolume(mediaPlayerComponent.getMediaPlayer().getVolume() + 10);  
 }  
 });  
  
 //Кнопка - (уменьшение грокости)  
 volumeDownButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setVolume(mediaPlayerComponent.getMediaPlayer().getVolume() - 10);  
 }  
 });  
  
 //Кнопка >>10 (перемотка на 10 секунд вперёд)  
 skipForwardButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().skip(10000);  
 }  
 });  
  
 //Кнопка 10<< (перемотка на 10 секунд назад)  
 skipBackwardButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().skip(-10000);  
 }  
 });  
 }  
  
 //Интерфейс для проигрывания файлв в формате mp3  
//Путь к папке с файлом  
//Проигрываемый файл  
 public void PlayingMp3File(String path, String fileName)  
 {  
 //Показ окна проигрывателя и проигрывание аудио  
 audioFrame.setVisible(true);  
 audioFrame.setName(fileName);  
 currentPath = path;  
 currentFile = fileName;  
 mediaPlayerComponent.getMediaPlayer().playMedia(path + fileName);  
 }  
  
}

ClientWindow.java

import javax.swing.\*;  
import java.awt.event.\*;  
import java.io.\*;  
  
//Класс окна клиента  
public class ClientWindow {  
  
 private JFrame clientFrame; //Форма для окна клиента  
 private JLabel clientState; //Строка о состоянии клиента  
 public SContainer allOurMusic; //Контейнер форм для хранения списка локальных аудио  
 public SContainer allServerMusic; //Контейнер форм для хранения списка серверных аудио  
  
 // Публичные флаги // Связь с интерфейсом сокетов  
 public Boolean flagStop = false;//Установка флага остановки  
 public Boolean flagAuth = false;//Установка флага подключения  
 public Boolean flagDsAt = false;//Установка флага отключения  
 public Boolean flagDown = false;//Установка флага закачки (скачивания файла)  
 public String downFile = ""; //Имя скачиваемого файла  
  
  
 ClientWindow()  
 {  
 //Создание и настройка окна клиента  
 clientFrame = new JFrame("Сlient Window");  
 clientFrame.setSize(485, 600);  
 clientFrame.setLayout(null);  
 clientFrame.setVisible(false);  
 clientFrame.setResizable(false);  
  
 //Настройка контейнера с локальными аудио  
 allOurMusic = new SContainer(clientFrame);  
 allOurMusic.SetContainerPosition(0, 0);  
 allOurMusic.SetContainerWidth(200);  
 allOurMusic.SetContainerName("Local Music");  
 allOurMusic.AddNewElement("No Music!");  
  
 //Настройка контейнера с серверными аудио  
 allServerMusic = new SContainer(clientFrame);  
 allServerMusic.SetContainerPosition(250, 0);  
 allServerMusic.SetContainerWidth(200);  
 allServerMusic.SetContainerName("Server Music");  
 allServerMusic.AddNewElement("No Music!");  
 allServerMusic.SetButtonsNames("Connect", "Disconnect");  
 allServerMusic.delButton.setVisible(false);  
 allServerMusic.addButton.setSize(200, 60);  
  
 //Установка листенеров  
  
 //Реакция на закрытие окна через крестик  
 clientFrame.addWindowListener(new WindowListener() {  
 public void windowOpened(WindowEvent e) {}  
 public void windowClosing(WindowEvent e) {  
 flagStop = true;  
 }  
 public void windowClosed(WindowEvent e) {}  
 public void windowIconified(WindowEvent e) {}  
 public void windowDeiconified(WindowEvent e) {}  
 public void windowActivated(WindowEvent e) {}  
 public void windowDeactivated(WindowEvent e) {}  
 });  
  
 //Реакция на двойное нажатие по списку аудио с сервера (скачать аудио)  
 allServerMusic.table.addMouseListener(new MouseListener() {  
 public void mouseClicked(MouseEvent evt) {  
 JList mList = (JList) evt.getSource();  
 if (evt.getClickCount() == 2) {  
 flagDown = true;  
 downFile = allServerMusic.table.getSelectedValue().toString();  
 }  
 }  
  
 public void mousePressed(MouseEvent evt) {}  
 public void mouseReleased(MouseEvent evt) {}  
 public void mouseEntered(MouseEvent evt) {}  
 public void mouseExited(MouseEvent evt) {}  
 });  
  
 //Реакция на двойное нажатие по списку аудио с локали (Воспроизвести аудио)  
 allOurMusic.table.addMouseListener(new MouseListener() {  
 public void mouseClicked(MouseEvent evt) {  
 JList mList = (JList) evt.getSource();  
 if (evt.getClickCount() == 2) {  
 Audio playingAudio = new Audio(allOurMusic.allList);  
 playingAudio.PlayingMp3File("Client/Music/", allOurMusic.table.getSelectedValue().toString());  
 }  
 }  
  
 public void mousePressed(MouseEvent evt) {}  
 public void mouseReleased(MouseEvent evt) {}  
 public void mouseEntered(MouseEvent evt) {}  
 public void mouseExited(MouseEvent evt) {}  
 });  
  
  
 //Add  
 allOurMusic.addButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 Folder folder = new Folder("Client/Music/");  
 folder.CopyFileInDirectory("Client/Music/");  
 }  
 });  
  
 //Delete  
 allOurMusic.delButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 Folder folder = new Folder("Client/Music/");  
 folder.DeleteFileFromDirectory("Client/Music/");  
 }  
 });  
  
 //Connect  
 allServerMusic.addButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 flagAuth = true;  
 allServerMusic.addButton.setVisible(false);  
 allServerMusic.table.setSize(200, 470);  
 }  
 });  
  
 //Disconnect  
 allServerMusic.delButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 flagDsAt = true;  
 }  
 });  
  
 //Добавление строки состояния  
 clientState = new JLabel();  
 clientState.setText("Change settings...");  
 clientState.setBounds(0, 530, 485, 30);  
 clientState.setVerticalAlignment(SwingConstants.CENTER);  
 clientState.setHorizontalAlignment(SwingConstants.CENTER);  
 clientFrame.add(clientState);  
 }  
  
 //Интерфейс для показа / сокрытия окна  
 //true - показать окно / false - скрыть окно  
 public void ViewServerWindow(Boolean viewMode)  
 {  
 clientFrame.setVisible(viewMode);  
 }  
  
 //Интерфейс для изменения строки состояния клиента  
 //Новая строка состояния  
 public void SetStringState(String enterState)  
 {  
 clientState.setText(enterState);  
 }  
  
 //Функция обновления списка аудио  
 //Необходимо занести в цикл для корректной работы  
 //Обновляемый контейнер  
 //Путь папки, в которой ищем файл  
 //Имя файла  
 public void UpdateMusicList(SContainer enterContainer, String enterFolderPath, String fileName)  
 {  
 Boolean control = false;  
  
 //Убираем все элементы из списка в контейнере  
 enterContainer.RemoveAllElements();  
  
 try  
 {  
 File file = new File(enterFolderPath + "/" + fileName);  
 FileReader reader = new FileReader(file);  
 BufferedReader bufferedReader = new BufferedReader(reader);  
  
 String readLine = bufferedReader.readLine();  
  
 while(readLine != null)  
 {  
 enterContainer.AddNewElement(readLine);  
 readLine = bufferedReader.readLine();  
 control = true;  
 }  
  
 //Елси нет элементов в списке  
 if(control == false)  
 {  
 enterContainer.AddNewElement("No Music!");  
 }  
  
 }  
 catch(FileNotFoundException fe)  
 {  
 System.out.println(fe);  
 return;  
 }  
 catch(IOException IOE)  
 {  
 System.out.println(IOE);  
 return;  
 }  
 }  
}

CliSocket.java

import java.net.\*;  
import java.io.\*;  
import java.util.concurrent.TimeUnit;  
  
//Архитектура, отвечающая за связь с сервером  
public class CliSocket  
{  
 private Socket socket; //Сокет для соединения с сервером  
 private DataInputStream input; //Поток для чтения из сокета  
 private DataOutputStream output; //Поток для записи в сокет  
 private int portNumber; //Номер порта, на котором мы будем работать с сервером  
 private String fileName; //Имя файла для отправки  
  
 //Публичные переменные для запросов  
 public String clientLogin = "No"; //Логин клиента  
 public Boolean authorization = false; //Флаг авторизации. Если она пройдена, начинае работу.  
 public Boolean get = false; //Флаг на получение трека с сервера  
 public Boolean stop = false; //Флаг на завершение работы  
  
 //Сборка и вычисления происходят в отдельном потоке!  
 public CliSocket()  
 {  
 //Функция проверки рабочих папок  
 StartUp();  
 }  
  
 //Функция начала проверки необходмых папок (Подготовка)  
 //Функция создаст необходимые папки, если они отсутствуют  
 private void StartUp()  
 {  
 //Проверка на наличие папки Server  
 Folder CheckedFolder = new Folder("Client");  
 CheckedFolder.CreateFolder();  
  
 //Проверка на наличие папки Client/Tracks  
 CheckedFolder.folderPath = "Client\\Music";  
 CheckedFolder.CreateFolder();  
  
 //Проверка на наличие папки Client/Playlists  
 CheckedFolder.folderPath = "Client\\Playlists";  
 CheckedFolder.CreateFolder();  
 }  
  
 //Основная функция работы с сервером  
 private void WorkWithServer()  
 {  
 try {  
 //Получение основного порта для работы  
 GetServerPort();  
  
 //Цепляемся на отправленный сервером порт и начинаем работу  
 socket = new Socket("127.0.0.1", portNumber);  
  
 //Настройка потока для прослушки, отправки сообщений и ввода с консоли  
 input = new DataInputStream(socket.getInputStream());  
 output = new DataOutputStream(socket.getOutputStream());  
  
 //Авторизация на сервере  
 ServerAuthorization();  
 authorization = true;  
  
 //Работа с сервером  
 while(stop == false)  
 {  
 MessageListener();  
 }  
  
 //Закрытие клиентского сокета  
 socket.close();  
 input.close();  
 output.close();  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Функция, получающая порт для работы с сервером  
 //Порт записывается в portNumber  
 private void GetServerPort()  
 {  
 try  
 {  
 //Цепляемся на 8000 порт и ждём от сервера информации о другом доступном порте  
 socket = new Socket("127.0.0.1", 8000);  
  
 //Организуем поток для получения ответа  
 input = new DataInputStream(socket.getInputStream());  
  
 //Начинаем слушать до тех пор, пока у нас не появится ответ  
 portNumber = input.readInt();  
  
 //После получения ответа - закрываем сокет и поток  
 socket.close();  
 input.close();  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Функция-запрос к серверу получения файла  
 private Boolean GetFileMessage()  
 {  
 FileOutputStream fileStream; //Поток для файла  
 Long sizeFile; //Размер файла, который мы отправляем  
 byte[] bytesFile = new byte[16\*1024]; //Буффер для хранения файла  
 int count; //Переменная для контроля передачи  
  
 try  
 {  
 //Запрос на получение файла  
 output.writeUTF(fileName);  
  
 //Выборка директории получения в зависимости от типа файла  
 if(fileName.endsWith(".mp3")){  
 fileStream = new FileOutputStream("Client\\Music\\" + fileName);  
 }  
 else if(fileName.endsWith(".txt"))  
 {  
 fileStream = new FileOutputStream("Client\\Playlists\\" + fileName);  
 }  
 else  
 {  
 return false;  
 }  
  
 //Ответ сервера на наличие файла  
 if(input.readBoolean() == true) //Файл есть и он придёт  
 {  
 //Ожидание отправки файла  
 sizeFile = input.readLong();  
  
 while (sizeFile > 0 && (count = input.read(bytesFile, 0, (int) Math.min(bytesFile.length, sizeFile))) != -1)  
 {  
 fileStream.write(bytesFile, 0, count);  
 sizeFile -= count;  
 }  
  
 //Окончание получения файла. Закрытие файлового потока.  
 fileStream.close();  
 System.out.println("File " + fileName + " " + " get!");  
 return true;  
 }  
 else //Файла нет и он не придёт  
 {  
 return false;  
 }  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return false;  
 }  
 }  
  
 //Функция-ответ на клиентский запрос файла  
 private Boolean SendFile()  
 {  
 Folder clientFolder; //Папка для извлечения файла  
 FileInputStream fileStream; //Поток для файла  
 byte[] bytesFile = new byte[16\*1024]; //Размер файла, который мы отправляем  
 Long sizeFile; //Размер файла, который мы отправляем  
 String needFile; //Строка, хранящее значение нужного клиенту файла  
 int count; //Переменная для контроля передачи  
  
 try {  
 //Забираем название файла  
 needFile = input.readUTF();  
  
 //Выбор директории. Откуда будем брать файл!?  
 if(needFile.endsWith(".mp3")){  
 clientFolder = new Folder("Client\\Music\\");  
 }  
 else  
 {  
 clientFolder = new Folder("Client\\Playlists\\");  
 }  
  
 //Задаём начальные признаки отправки  
 File sendFile = new File(clientFolder.folderPath + needFile);  
 sizeFile = sendFile.length();  
 fileStream = new FileInputStream(sendFile);  
  
 //Если файл существует  
 if (clientFolder.FindFile(needFile)) {  
 output.writeBoolean(true);  
 output.writeLong(sizeFile);  
  
 while ((count = fileStream.read(bytesFile)) != -1) {  
 output.write(bytesFile, 0, count);  
 output.flush();  
 }  
  
 fileStream.close();  
 return true;  
 }  
 //Если файла не существует  
 else  
 {  
 output.writeBoolean(false);  
 return false;  
 }  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return false;  
 }  
 }  
  
 //Функция, выполняющая авторизацию на сервере  
 private void ServerAuthorization()  
 {  
 //Создание окна авторизации  
 LoginWindow log = new LoginWindow();  
 log.ViewLoginWindow(true);  
  
 String userAnswer = "1";  
  
 try  
 {  
 //Цикл отправки логина  
 while(userAnswer.compareTo("1") == 0)  
 {  
 //Ожидание ввода пользователя  
 //Нужна небольшая задержка, чтобы цикл осуществил срабатывание условия  
 while(log.flagResult == false)  
 {  
 try  
 {  
 Thread.sleep(100);  
 }  
 catch(InterruptedException i)  
 {  
 System.out.println("Except");  
 return;  
 }  
 }  
  
 //Уведомление пользователя, отправка логина на сервер  
 log.SetStringState("Start Authorization...");  
 clientLogin = log.GetLoginEnter();  
 output.writeUTF(clientLogin);  
  
 //Ожидание ответа о корректности логина  
 //Если логин корректен  
 if(input.readBoolean() == true)  
 {  
 log.SetStringState("Login found!");  
 log.SetStringState("Server entering...");  
 log.ViewLoginWindow(false);  
 return;  
 }  
 //Если логин - некорректен, повторяем ввод  
 else  
 {  
 log.SetStringState("Login not found! Input other login!");  
 output.writeUTF("1");  
 log.flagResult = false;  
 }  
 }  
 }  
 catch(EOFException eof)  
 {  
 System.out.println(eof);  
 return;  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Функция аудита сообщений сервера  
 //Должна быть в цикле  
 private void MessageListener()  
 {  
 String messageIdentifier; //Переменная, хранящая идентификатор сообщения  
  
 try  
 {  
 //Получение кода сообщения  
 messageIdentifier = input.readUTF();  
  
 //Отключение соединения  
 if(messageIdentifier.compareTo("0") == 0)  
 {  
 stop = true;  
 }  
 //Отправить файл  
 else if(messageIdentifier.compareTo("1") == 0)  
 {  
 output.writeUTF("2");  
 SendFile();  
 }  
 //Запросить файл  
 else if(messageIdentifier.compareTo("2") == 0)  
 {  
 GetFileMessage();  
 }  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Интерфейс запуска авторизации  
 public void StartClientWork()  
 {  
 //Создание отдельного потока для функции работы сокета  
 Thread qui = new Thread(() -> WorkWithServer());  
 qui.start();  
 }  
  
 //Функция получения определённого трека с сервера  
 //Имя трека, который необходимо получить с сервера (\*\*\*.mp3)  
 public void GetCli(String trackName)  
 {  
 try  
 {  
 if(authorization == true){  
 fileName = trackName;  
 output.writeUTF("1");  
 }  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Функция остановки работы клиентского сокета  
 public void StopCli()  
 {  
 try  
 {  
 output.writeUTF("0");  
 }  
 catch(IOException n)  
 {  
 System.out.println(n);  
 return;  
 }  
 }  
}

Folder.java

import javax.swing.\*;  
import java.io.\*;  
  
//Класс папки для работы с файлами  
public class Folder  
{  
 public String folderPath; //Путь до папки  
  
 public Folder(String enteredFolderName)  
 {  
 folderPath = enteredFolderName;  
 }  
  
 //Функция проверки существования папки  
 public Boolean CheckFolder()  
 {  
 File CheckFolder = new File(folderPath);  
  
 //Условие проверки существования папки  
 if(CheckFolder.exists())  
 {  
 return true;  
 }  
 else  
 {  
 return false;  
 }  
 }  
  
 //Функция проверки существования файла в данной папке  
 public Boolean CheckFile(String fileName)  
 {  
 File CheckFile = new File(folderPath + "\\" + fileName);  
  
 //Условие проверки существования файла  
 if(CheckFile.exists() && CheckFile.isFile())  
 {  
 return true;  
 }  
 else  
 {  
 return false;  
 }  
 }  
  
 //Функция создания папки  
 //Есть проверка на существовние папки  
 public Boolean CreateFolder()  
 {  
 //Проверка существования папки перед созданием  
 if(CheckFolder())  
 {  
 return false;  
 }  
  
 File CreateFolder = new File(folderPath);  
 CreateFolder.mkdirs();  
 return true;  
 }  
  
 //Функция обнаружения всех MP4-файлов в папке и их запись в плейлист  
 //Есть проверка на существование папки  
 //Есть вывод о ходе проверки  
 public void DetectAllMp3(String playlistName)  
 {  
 try {  
 //Проверка существовования папки  
 if (!CheckFolder()) {  
 return;  
 }  
  
 //Проверка существования файла плейлиста  
 if (!CheckFile(playlistName)) {  
 File CheckFile = new File(folderPath + "\\" + playlistName);  
 CheckFile.createNewFile();  
 }  
  
 //Получение списка всех файлов из папки  
 File CheckFolder = new File(folderPath);  
 File AllFiles[] = CheckFolder.listFiles();  
  
 PrintWriter FileWriter = new PrintWriter(folderPath + "\\" + playlistName);  
  
 for (File f : AllFiles) {  
 if (f.isFile() && f.exists() && f.getName().endsWith(".mp3")) {  
 FileWriter.println(f.getName());  
 }  
 }  
  
 FileWriter.close();  
 }  
 catch(IOException IOE)  
 {  
 System.out.println(IOE);  
 }  
 }  
  
 //Функция поиска файла в текущей папке  
 //Вводить файлы только с приставкой  
 public Boolean FindFile(String fileName)  
 {  
 //Проверка существовования папки  
 if(!CheckFolder())  
 {  
 return false;  
 }  
  
 //Получение списка всех файлов из папки  
 File CheckFolder = new File(folderPath);  
 File AllFiles[] = CheckFolder.listFiles();  
  
 for (File f : AllFiles)  
 {  
 //Условие поиска: одинак. имя, это файл, он существует  
 if((fileName.compareTo(f.getName()) == 0) && f.isFile()  
 && f.exists())  
 {  
 return true;  
 }  
 }  
  
 return false;  
 }  
  
 //Функция выбора и копирования файла в папку  
 //Папка, куда будет скопирована информация  
 public Boolean CopyFileInDirectory(String enterFolderPath)  
 {  
 //Открытие окна выбора файла  
 JFileChooser fileChoose = new JFileChooser();  
  
 //Выбранный файл  
 File origFile, copyFile;  
 String choseFile = new String("\_ESTEB\_");  
  
 int ret = fileChoose.showDialog(null, "Открыть файл");  
 if(ret == JFileChooser.APPROVE\_OPTION)  
 {  
 origFile = fileChoose.getSelectedFile();  
 }  
 else  
 {  
 return false;  
 }  
  
 InputStream is = null;  
 OutputStream os = null;  
  
 copyFile = new File(enterFolderPath + origFile.getName());  
  
 try  
 {  
 is = new FileInputStream(origFile);  
 os = new FileOutputStream(copyFile);  
 byte[] buffer = new byte[1024];  
 int length;  
  
 while((length = is.read(buffer)) > 0)  
 {  
 os.write(buffer, 0, length);  
 }  
  
 is.close();  
 os.close();  
 }  
 catch(IOException e)  
 {  
 System.out.println(e);  
 }  
  
 return true;  
 }  
  
 //Функция удаления файла из папки  
 //Путь к папке, в которой лежит файл  
 public Boolean DeleteFileFromDirectory(String enterFolderPath)  
 {  
 //Открытие окна выбора файла  
 JFileChooser fileChoose = new JFileChooser(enterFolderPath);  
  
 //Выбранный файл  
 File origFile, copyFile;  
 String choseFile = new String("\_ESTEB\_");  
  
 //Открыть окно с выбором файла  
 int ret = fileChoose.showDialog(null, "Удалить файл");  
 if(ret == JFileChooser.APPROVE\_OPTION)  
 {  
 origFile = fileChoose.getSelectedFile();  
 }  
 else  
 {  
 return false;  
 }  
  
 //Удаление файла  
 origFile.delete();  
 return true;  
 }  
}

LoginWindow.java

import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
//Класс окна ввода логина  
public class LoginWindow {  
  
 private JFrame loginFrame; //Форма для окна логина  
 private JTextField loginEnter; //Строка для ввода логина  
 private JButton loginButton; //Кнопка "Login"  
 private JLabel stateString; //Строка состояния окна логина  
  
 public Boolean flagResult = false; //Флаг фиксации результата  
  
 LoginWindow()  
 {  
 loginFrame = new JFrame("Login on Server");  
 loginEnter = new JTextField("");  
 loginButton = new JButton("Login");  
 stateString = new JLabel();  
  
 //Настройка окна ввода логина  
 loginFrame.setBounds(500, 500, 300, 120);  
 loginFrame.setLayout(null);  
 loginFrame.setVisible(false);  
 loginFrame.setResizable(false);  
 loginFrame.setUndecorated(true);  
  
 //Настройка элементов окна логина  
 stateString.setBounds(0, 0, 300, 50);  
 stateString.setText("Enter Login");  
 stateString.setHorizontalAlignment(JTextField.CENTER);  
 stateString.setVerticalAlignment(JTextField.CENTER);  
  
 loginEnter.setBounds(0, 50, 300, 30);  
 loginButton.setBounds(0, 80, 300, 40);  
  
 //Добавление элементов в окно  
 loginFrame.add(loginEnter);  
 loginFrame.add(loginButton);  
 loginFrame.add(stateString);  
  
 //Установка листенеров  
 loginButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 flagResult = true;  
 }  
 });  
 }  
  
 //Интерфейс для показа / сокрытия окна  
 //true - показать окно / false - скрыть окно  
 public void ViewLoginWindow(Boolean viewMode)  
 {  
 loginFrame.setVisible(viewMode);  
 }  
  
 //Интерфейс для изменения строки состояния  
 //Новое значение строки состояния  
 public void SetStringState(String enterState)  
 {  
 stateString.setText(enterState);  
 }  
  
 //Интерфейс для получения введённого логина  
 public String GetLoginEnter()  
 {  
 return loginEnter.getText();  
 }  
}

Main.java

import com.sun.jna.NativeLibrary;  
import uk.co.caprica.vlcj.discovery.NativeDiscovery;  
import uk.co.caprica.vlcj.runtime.RuntimeUtil;  
  
public class Main {  
 public static void main(String[] args)  
 {  
 //Привязка библиотеки vlc  
 NativeLibrary.addSearchPath(RuntimeUtil.getLibVlcLibraryName(), "C:\\Program Files\\AudioLAN\\VLC");  
 new NativeDiscovery().discover();  
  
 //Вывод окна клиента  
 ClientWindow cw = new ClientWindow();  
 cw.ViewServerWindow(true);  
  
 //Создание объекта клиента  
 CliSocket nSocket = new CliSocket();  
  
 //Создание объекта "папка клиента" для отслеживания новых файлов  
 Folder detectFolder = new Folder("Client/Music");  
  
 //Пока запущено основное окно  
 while(cw.flagStop == false)  
 {  
 //Обработка подключения клиента к серверу  
 if(cw.flagAuth == true)  
 {  
 nSocket.StartClientWork();  
 cw.flagAuth = false;  
 }  
  
 //Обработка отключения клиента от сервера  
 if(cw.flagDsAt == true)  
 {  
 nSocket.StopCli();  
 cw.flagDsAt = false;  
 nSocket.authorization = false;  
 nSocket.stop = false;  
 nSocket.clientLogin = "null";  
 }  
  
 //Анализ резульата авторизации  
 if(nSocket.authorization != true)  
 {  
 cw.SetStringState("Server: Disconnect");  
 }  
 else  
 {  
 //Получить список всех аудио с сервера  
 nSocket.GetCli("AllMusic.txt");  
  
 //Задержка для устранения конфликта передачи  
 try  
 {  
 Thread.sleep(1000);  
 }  
 catch(InterruptedException i)  
 {  
 System.out.println("Except");  
 return;  
 }  
  
 //Скачать нужный файл  
 if(cw.flagDown == true)  
 {  
 nSocket.GetCli(cw.downFile);  
 cw.flagDown = false;  
 }  
  
 //Обновление строки состояния клиента  
 cw.SetStringState("Server: Connect. Login: " + nSocket.clientLogin);  
  
 //Обновление списка серверных треков  
 cw.UpdateMusicList(cw.allServerMusic, "Client/Playlists", "AllMusic.txt");  
 }  
  
 //Обновление списка локальных треков  
 detectFolder.DetectAllMp3("ClientPlaylist.txt");  
 cw.UpdateMusicList(cw.allOurMusic, "Client/Music", "ClientPlaylist.txt");  
  
 //Задержка между обновлением харатеристик клиента  
 try  
 {  
 Thread.sleep(2000);  
 }  
 catch(InterruptedException i)  
 {  
 System.out.println("Except");  
 return;  
 }  
  
 //Реакция на остановку программы (Находится здесь, т.к необходимо вручную вызвать остановку)  
 if(cw.flagStop == true)  
 {  
 //Обрыв соединения с сервером  
 nSocket.StopCli();  
 System.out.println("End");  
 }  
 }  
 }  
}

SContainer.java

import javax.swing.\*;  
  
//Класс серверных контейнеров форм (Список - Граф. представление - Кнопка)  
public class SContainer  
{  
 private DefaultListModel<String> allList; //Список элементов в таблице  
 public JButton addButton; //Кнопка добавления в список  
 public JButton delButton; //Кнопка удаления из списка  
 public JList <String> table; //Выводимая таблица  
 private JLabel name; //Подпись контейнера  
 private JScrollPane scroll; //Скроллбар контейнера  
  
 private int xPos = 0; //Позиции контейнера  
 private int yPos = 0;  
  
 private int width = 0; //Ширина контейнера  
  
 private int high = 410;//Высота списка в контейнере  
  
 //Конструктор контейнера  
 //Окно, куда будут приставлены формы  
 SContainer(JFrame baseFrame) {  
 allList = new DefaultListModel<String>();  
 table = new JList<>(allList);  
 addButton = new JButton();  
 delButton = new JButton();  
 name = new JLabel();  
 scroll = new JScrollPane();  
  
 //Настройка кнопки добавления  
 baseFrame.add(addButton);  
 addButton.setText("Add Audio");  
  
 //Настройка кнопки удаления  
 baseFrame.add(delButton);  
 delButton.setText("Delete Audio");  
  
 //Настройка таблицы  
 baseFrame.add(table);  
  
 //Настройка надписи над контейнером  
 name.setHorizontalAlignment(SwingConstants.CENTER);  
 name.setVerticalAlignment(SwingConstants.CENTER);  
 baseFrame.add(name);  
 }  
  
 //Функция фиксации изменения размеров контейнера  
 private void ChangeContainerPosition()  
 {  
 name.setBounds(xPos, yPos, width, 30);  
 table.setBounds(xPos, yPos + 30, width, high);  
 addButton.setBounds(xPos, yPos + 440, width, 30);  
 delButton.setBounds(xPos, yPos + 470, width, 30);  
 scroll.setBounds(xPos + width, yPos + 30, 20, 500);  
 }  
  
 //Интерфейс для задания координат контейнера  
 public void SetContainerPosition(int x, int y)  
 {  
 xPos = x;  
 yPos = y;  
 ChangeContainerPosition();  
 }  
  
 //Интерфейс для задания ширины контейнера  
 public void SetContainerWidth(int enterWidth)  
 {  
 width = enterWidth;  
 ChangeContainerPosition();  
 }  
  
 //Интерфейс для задания названия кнопок  
 public void SetButtonsNames(String firstButton, String secondButton)  
 {  
 addButton.setText(firstButton);  
 delButton.setText(secondButton);  
 }  
  
 //Интерфейс для задания имени контейнера  
 public void SetContainerName(String enterName)  
 {  
 name.setText(enterName);  
 }  
  
 //Интерфейс для добавления нового элемента в список  
 public void AddNewElement(String newElement)  
 {  
 allList.addElement(newElement);  
 }  
  
 //Интерфейс для удаления элемента из списка  
 public void RemoveAllElements()  
 {  
 allList.removeAllElements();  
 }  
}

# **Сервер:**

Audio.java

import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import javax.swing.\*;  
  
import uk.co.caprica.vlcj.component.EmbeddedMediaPlayerComponent;  
  
//Класс окна с воспроизводимым внутри аудио  
public class Audio {  
  
 private JFrame audioFrame; //Экран для вывода аудио (проигрыватель)  
 private EmbeddedMediaPlayerComponent mediaPlayerComponent; //Проигрыватель  
 private String currentPath; //Текущий путь к файлу  
 private String currentFile; //Текущий проигрываемый файл  
 private JButton playButton;  
 private JButton pauseButton;  
 private JButton nextButton;  
 private JButton previousButton;  
 private JPanel panel;  
 private JPanel panelMedia;  
 private JButton stopButton;  
 private JButton resetButton;  
 private JButton fastForwardButton;  
 private JButton rewindButton;  
 private JButton volumeUpButton;  
 private JButton volumeDownButton;  
 private JButton skipForwardButton;  
 private JButton skipBackwardButton;  
  
 Audio(DefaultListModel<String> musiclist)  
 {  
 audioFrame = new JFrame("Плеер");  
 audioFrame.setVisible(true);  
 audioFrame.setSize(280, 300);  
 audioFrame.setLocation(120, 100);  
 audioFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);  
 audioFrame.setContentPane(panel);//задаёт содержимое окна с рамкой  
  
 panelMedia = new JPanel();  
 mediaPlayerComponent = new EmbeddedMediaPlayerComponent();  
 panelMedia.add(mediaPlayerComponent);  
  
 audioFrame.setContentPane(mediaPlayerComponent);  
 audioFrame.setContentPane(new JPanel(new BorderLayout()));  
 audioFrame.getContentPane().add(mediaPlayerComponent, BorderLayout.CENTER);  
 audioFrame.getContentPane().add(panel, BorderLayout.SOUTH);  
 audioFrame.setVisible(false);  
  
  
 //Листенеры//  
  
 //Кнопка закрытия окна  
 audioFrame.addWindowListener(new WindowAdapter() {  
 @Override  
 public void windowClosing(WindowEvent e) {  
 mediaPlayerComponent.release();  
 audioFrame.setVisible(false);  
 }  
 });  
  
 //Кнопка Play  
 playButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().play();  
 }  
 });  
  
 //Кнопка Pause  
 pauseButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().pause();  
 }  
 });  
  
 //Кнопка Next  
 nextButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
  
 int curindex = 0;  
 String data = "";  
  
 for(;curindex < musiclist.size(); curindex++)  
 {  
 data = musiclist.getElementAt(curindex);  
 if(data.compareTo(audioFrame.getName()) == 0)  
 {  
 if(curindex == musiclist.size()-1)  
 {  
 data = audioFrame.getName();  
 break;  
 }  
 else  
 {  
 data = musiclist.getElementAt(curindex+1);  
 break;  
 }  
 }  
 }  
  
 audioFrame.setName(data);  
 mediaPlayerComponent.getMediaPlayer().stop();  
 mediaPlayerComponent.getMediaPlayer().playMedia("Server/AllMusic/" + data);  
 }  
 });  
  
 //Кнопка Previous  
 previousButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
  
 int curindex = 0;  
 String data = "";  
  
 for(;curindex < musiclist.size(); curindex++)  
 {  
 data = musiclist.getElementAt(curindex);  
 if(data.compareTo(audioFrame.getName()) == 0)  
 {  
 if(curindex == 0)  
 {  
 data = audioFrame.getName();  
 break;  
 }  
 else  
 {  
 data = musiclist.getElementAt(curindex-1);  
 break;  
 }  
 }  
 }  
  
 audioFrame.setName(data);  
 mediaPlayerComponent.getMediaPlayer().stop();  
 mediaPlayerComponent.getMediaPlayer().playMedia("Server/AllMusic/" + data);  
 }  
 });  
  
 //Кнопка Stop  
 stopButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().stop();  
 }  
 });  
  
 //Кнопка Reset  
 resetButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setTime(0);  
 }  
 });  
  
 //Кнопка >> (ускорение в 2 раза)  
 fastForwardButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setRate(mediaPlayerComponent.getMediaPlayer().getRate()\*2);  
 }  
 });  
  
 //Кнопка << (замеделние в 2 раза)  
 rewindButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setRate(mediaPlayerComponent.getMediaPlayer().getRate()/2);  
 }  
 });  
  
 //Кнопка + (увеличение громкости)  
 volumeUpButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setVolume(mediaPlayerComponent.getMediaPlayer().getVolume() + 10);  
 }  
 });  
  
 //Кнопка - (уменьшение грокости)  
 volumeDownButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().setVolume(mediaPlayerComponent.getMediaPlayer().getVolume() - 10);  
 }  
 });  
  
 //Кнопка >>10 (перемотка на 10 секунд вперёд)  
 skipForwardButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().skip(10000);  
 }  
 });  
  
 //Кнопка 10<< (перемотка на 10 секунд назад)  
 skipBackwardButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 mediaPlayerComponent.getMediaPlayer().skip(-10000);  
 }  
 });  
 }  
  
 //Интерфейс для проигрывания файлов в формате mp3  
 //Путь к папке с файлом  
 //Проигрываемый файл  
 public void PlayingMp3File(String path, String fileName)  
 {  
 //Показ окна проигрывателя и проигрывание аудио  
 audioFrame.setVisible(true);  
 audioFrame.setName(fileName);  
 mediaPlayerComponent.getMediaPlayer().playMedia(path + fileName);  
 }  
}

Folder.java

import javax.swing.\*;  
import java.io.\*;  
  
  
public class Folder  
{  
  
 public String folderPath;  
  
 public Folder(String EnteredFolderName){  
 folderPath = EnteredFolderName;  
 }  
  
 //Функция проверки существования папки  
 public Boolean CheckFolder()  
 {  
 File CheckFolder = new File(folderPath);  
  
 //Условие проверки существования папки  
 if(CheckFolder.exists())  
 {  
 return true;  
 }  
 else  
 {  
 return false;  
 }  
 }  
  
 //Функция проверки существования файла  
 public Boolean CheckFile(String FileName)  
 {  
 File CheckFile = new File(folderPath + "\\" + FileName);  
  
 //Условие проверки существования файла  
 if(CheckFile.exists() && CheckFile.isFile())  
 {  
 return true;  
 }  
 else  
 {  
 return false;  
 }  
 }  
  
 //Функция создания папки  
 //Есть проверка на существовние папки  
 public Boolean CreateFolder()  
 {  
 //Проверка существования папки перед созданием  
 if(CheckFolder())  
 {  
 return false;  
 }  
  
 File CreateFolder = new File(folderPath);  
 CreateFolder.mkdirs();  
 return true;  
 }  
  
 //Функция обнаружения всех MP3-файлов в папке и их запись в плейлист  
 //Есть проверка на существование папки  
 //Есть вывод о ходе проверки  
 public void DetectAllMp3(String PlaylistName) throws IOException  
 {  
 //Проверка существовования папки  
 if(!CheckFolder())  
 {  
 return;  
 }  
  
 //Проверка существования файла плейлиста  
 if(!CheckFile(PlaylistName))  
 {  
 File CheckFile = new File(folderPath + "\\" + PlaylistName);  
 CheckFile.createNewFile();  
 }  
  
 //Получение списка всех файлов из папки  
 File CheckFolder = new File(folderPath);  
 File AllFiles[] = CheckFolder.listFiles();  
  
 PrintWriter FileWriter = new PrintWriter(folderPath + "\\" + PlaylistName);  
  
 for (File f : AllFiles)  
 {  
 if(f.isFile() && f.exists() && f.getName().endsWith(".mp3"))  
 {  
 FileWriter.println(f.getName());  
 }  
 }  
 FileWriter.close();  
 }  
  
 //Функция поиска файла в текущей папке  
 //Вводить файлы только с приставкой  
 public Boolean FindFile(String fileName)  
 {  
 //Проверка существовования папки  
 if(!CheckFolder())  
 {  
 return false;  
 }  
  
 //Получение списка всех файлов из папки  
 File CheckFolder = new File(folderPath);  
 File AllFiles[] = CheckFolder.listFiles();  
  
 for (File f : AllFiles)  
 {  
 //Условие поиска: одинак. имя, это файл, он существует, оканчивается .mp3  
 if((fileName.compareTo(f.getName()) == 0) && f.isFile()  
 && f.exists())  
 {  
 return true;  
 }  
 }  
  
 return false;  
 }  
  
 //Функция выбора и копирования файла в папку Server/AllMusic  
 public Boolean GetFileInDirectory()  
 {  
 //Открытие окна выбора файла  
 JFileChooser fileChoose = new JFileChooser();  
  
 //Выбранный файл  
 File origFile, copyFile;  
 String choseFile = new String("\_ESTEB\_");  
  
 int ret = fileChoose.showDialog(null, "Открыть файл");  
 if(ret == JFileChooser.APPROVE\_OPTION)  
 {  
 origFile = fileChoose.getSelectedFile();  
 }  
 else  
 {  
 return false;  
 }  
  
 InputStream is = null;  
 OutputStream os = null;  
  
 copyFile = new File("Server/AllMusic/" + origFile.getName());  
  
 try  
 {  
 is = new FileInputStream(origFile);  
 os = new FileOutputStream(copyFile);  
 byte[] buffer = new byte[1024];  
 int length;  
  
 while((length = is.read(buffer)) > 0)  
 {  
 os.write(buffer, 0, length);  
 }  
  
 is.close();  
 os.close();  
 }  
 catch(IOException e)  
 {  
 System.out.println(e);  
 }  
  
 return true;  
 }  
  
 //Функция удаления файла из папки  
 public Boolean DeleteFileOutDirectory()  
 {  
 //Открытие окна выбора файла  
 JFileChooser fileChoose = new JFileChooser("Server/AllMusic");  
  
 //Выбранный файл  
 File origFile, copyFile;  
 String choseFile = new String("\_ESTEB\_");  
  
 //Открыть окно с выбором файла  
 int ret = fileChoose.showDialog(null, "Удалить файл");  
 if(ret == JFileChooser.APPROVE\_OPTION)  
 {  
 origFile = fileChoose.getSelectedFile();  
 }  
 else  
 {  
 return false;  
 }  
  
 //Удаление файла  
 origFile.delete();  
 return true;  
 }  
}

Main.java

import java.lang.Thread;  
import java.io.IOException;  
import com.sun.jna.NativeLibrary;  
import uk.co.caprica.vlcj.runtime.RuntimeUtil;  
import uk.co.caprica.vlcj.discovery.NativeDiscovery;  
  
//Основной класс  
public class Main  
{  
 public static void main(String[] args) throws IOException  
 {  
  
 //Привязка библиотеки vlc  
 NativeLibrary.addSearchPath(RuntimeUtil.getLibVlcLibraryName(), "C:\\Program Files\\AudioLAN\\VLC");  
 new NativeDiscovery().discover();  
  
 /\*Предподготовка\*/  
  
 //Проверка на наличие папки Server  
 Folder CheckedFolder = new Folder("Server");  
 CheckedFolder.CreateFolder();  
  
 //Проверка на наличие папки Server/AllMusic  
 CheckedFolder.folderPath = "Server\\AllMusic";  
 CheckedFolder.CreateFolder();  
  
 //Найти все MP3 файлы в папке Server\AllMusic  
 CheckedFolder.DetectAllMp3("AllMusic.txt");  
  
 /\*\_\_\_\_\_\_\_\_\_\_\_\_\_\*/  
  
 //Создание объекта сервера  
 SerSocket OurServer = new SerSocket();  
  
 //Вывод окна сервера  
 ServerWindow sw = new ServerWindow();  
 sw.ViewServerWindow(true);  
 sw.SetStringState("Server stability work");  
  
 while(true)  
 {  
 //Уведомление об обновлении сервера  
 sw.SetStringState("Data update");  
  
 //Обновление в списке клиентов  
 sw.UpdateClientList(OurServer.GetSocketList(), OurServer.GetContainerMax());  
  
 //Пересоздание списка музыки  
 CheckedFolder.DetectAllMp3("AllMusic.txt");  
  
 //Обновление в списке музыки  
 sw.UpdateMusicList();  
  
 try {  
 sw.SetStringState("Server stability work");  
 Thread.sleep(5000);  
 }  
 catch(InterruptedException i)  
 {  
 System.out.println("Except");  
 return;  
 }  
 }  
 }  
}

SContainer.java

import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
//Класс серверных контейнеров форм (Список - Графю представление - Кнопка)  
public class SContainer  
{  
 private DefaultListModel<String> allList; //Список элементов в таблице  
 private JButton addButton; //Кнопка добавления в список  
 private JButton delButton; //Кнопка удаления из списка  
 private JLabel name; //Подпись контейнера  
 private JScrollPane scroll; //Скроллбар контейнера  
 public JList <String> table; //Выводимая таблица  
  
 private int xPos = 0; //Позиция контейнера  
 private int yPos = 0;  
 private int width = 0; //Ширина контейнера  
  
 private int high = 410; //Высота листа в контейнере  
  
 //Конструктор контейнера  
 //Окно, куда будут приставлены формы  
 SContainer(JFrame baseFrame) {  
 allList = new DefaultListModel<String>();  
 table = new JList<>(allList);  
 addButton = new JButton();  
 delButton = new JButton();  
 name = new JLabel();  
 scroll = new JScrollPane();  
  
 //Настройка кнопки добавления  
 baseFrame.add(addButton);  
 addButton.setText("Add Music");  
  
 //Настройка кнопки удаления  
 baseFrame.add(delButton);  
 delButton.setText("Delete Music");  
  
 //Настройка таблицы  
 baseFrame.add(table);  
  
 //Настройка надписи  
 name.setHorizontalAlignment(SwingConstants.CENTER);  
 name.setVerticalAlignment(SwingConstants.CENTER);  
 baseFrame.add(name);  
  
  
 //Установка листенеров  
 addButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 Folder folder = new Folder("Server/AllMusic");  
 folder.GetFileInDirectory();  
 }  
 });  
  
 delButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 Folder folder = new Folder("Server/AllMusic");  
 folder.DeleteFileOutDirectory();  
 }  
 });  
 }  
  
 private void ChangeContainerPosition()  
 {  
 name.setBounds(xPos, yPos, width, 30);  
 table.setBounds(xPos, yPos + 30, width, high);  
 addButton.setBounds(xPos, yPos + 440, width, 30);  
 delButton.setBounds(xPos, yPos + 470, width, 30);  
 scroll.setBounds(xPos + width, yPos + 30, 20, 500);  
 }  
  
 //Интерфейс для задания координат контейнера  
 public void SetContainerPosition(int X, int Y)  
 {  
 xPos = X;  
 yPos = Y;  
 ChangeContainerPosition();  
 }  
  
 //Интерфейс для задания ширины контейнера  
 public void SetContainerWidth(int enterWidth)  
 {  
 width = enterWidth;  
 ChangeContainerPosition();  
 }  
  
 //Интерфейс для сокрытия кнопок (необходимо в списке клиентов)  
 //true - сделать видимыми / false - сделать невидимыми  
 public void HideButtons(Boolean enterState)  
 {  
 addButton.setVisible(enterState);  
 delButton.setVisible(enterState);  
 }  
  
 //Интерфейс для задания новой высоты листа в контейнере  
 public void SetContainerListHigh(int enterHigh)  
 {  
 high = enterHigh;  
 ChangeContainerPosition();  
 }  
  
 //Интерфейс для задания имени контейнера  
 public void SetContainerName(String enterName)  
 {  
 name.setText(enterName);  
 }  
  
 //Интерфейс для добавления нового элемента в список  
 public void AddNewElement(String newElement)  
 {  
 allList.addElement(newElement);  
 }  
  
 //Интерфейс для удаления элемента из списка  
 public void RemoveAllElements()  
 {  
 allList.removeAllElements();  
 }  
}

SerSocket.java

import java.net.\*;  
import java.io.\*;  
  
//Контейнер описывающий пару: Сетевой - Клиентский сокет  
class SocketPair  
{  
 public ServerSocket serverSocket; //Сервеный сокет для соединения  
 public Socket clientSocket; //Клиентский сокет для соединения  
 public DataInputStream inputThread; //Поток чтения из сокета  
 public DataOutputStream outputThread; //Поток записи в сокет  
 public int currentPort; //Порт, на котором находится пара  
 public String currentIp; //Ip клиента  
 public String userLogin; //Логин, под которым зашёл клиент  
 public Boolean stateBlock = false; //Состояние блока (t-используется / f-не используется)  
}  
  
//Класс, описывающий объект сервера  
public class SerSocket  
{  
 private ServerSocket guestSocket; //Гостевой сокет для выдачи нужного порта  
 private SocketPair[] socketMatrix; //Матрица контейнеров с сокетами  
 private int maxBlocks = 10; //Максимально кол-во контейнеров в матрице (одновременных клиентов на сервере)  
 private Boolean flagClose = false; //Флаг остановки работы сервера  
 private String fileName; //Название запрашиваемого файла  
  
 //Основной конструктор для создания серверного сокета  
 public SerSocket()  
 {  
 try  
 {  
 //Создание гостевого сокета, выдающего номера портов  
 guestSocket = new ServerSocket(8000);  
  
 //Создание контейнеров для сокетов  
 socketMatrix = new SocketPair[maxBlocks];  
  
 for(int i = 0; i < maxBlocks; ++i){  
 socketMatrix[i] = new SocketPair();  
 }  
  
 //Создание отдельного потока для функции обработки заявок  
 Thread qui = new Thread(() -> TakeNeedPort());  
 qui.start();  
 }  
 catch(UnknownHostException u)  
 {  
 System.out.println(u);  
 return;  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Поточная функция. Организует раздачу портов (контейнеров) для сокетов, ожидающих сервер.  
 //Высоконагруженная функция!!!  
 private void TakeNeedPort()  
 {  
 Socket copySocket; //Доп. сокет для отлавливания соединения  
 DataOutputStream output; //Поток записи в сокет  
  
 try  
 {  
 //Пока не выставлен флаг на закрытие сервера, обрабатываем заявки  
 while(true)  
 {  
 try {  
 Thread.sleep(100);  
 }  
 catch(InterruptedException i)  
 {  
 System.out.println("Except");  
 return;  
 }  
  
 //Ждём соединения в порт заявок.  
 //Затем, уведомлояем о соединении и открываем поток для отправки сообщений клиенту  
 copySocket = guestSocket.accept();  
 output = new DataOutputStream(copySocket.getOutputStream());  
 System.out.println("\nNew Guest! Taking him new port...");  
  
 //Ищем пустой контейнер под соединение  
 for(int i = 1; i < maxBlocks; ++i)  
 {  
 //Если такой блок найден - заполняем его актуальными данными  
 if(socketMatrix[i].stateBlock != true)  
 {  
 socketMatrix[i].stateBlock = true; //Бронируем контейнер  
 socketMatrix[i].currentPort = 8000 + i; //Расчёт порта  
 socketMatrix[i].serverSocket =  
 new ServerSocket(socketMatrix[i].currentPort); //Создаём сетевой сокет в забронированном блоке  
  
 //Сообщаем клиенту, что контейнер готов для работы, высылая номер порта  
 output.writeInt(socketMatrix[i].currentPort);  
  
 //Создание отдельного потока для каждого контейнера / сетевого сокета  
 ServerThreadCreate(socketMatrix[i]);  
 break;  
 }  
 }  
 }  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Создаёт потоки для независимой работы сетевых сокетов.  
 //Нужна только для корректного создания потока с функцией ServerSockWork()!  
 private void ServerThreadCreate(SocketPair currentPair)  
 {  
 Thread qui = new Thread(() -> WorkWithClient(currentPair));  
 qui.start();  
 }  
  
 //Поточная функция. Осуществляет работу сетевого сокета на отдельном потоке / в отдельном контейнере.  
 //Высоконагруженная функция!  
 private void WorkWithClient(SocketPair currentPair)  
 {  
  
 try  
 {  
 currentPair.clientSocket = currentPair.serverSocket.accept(); //Ожидание подключение клиента через новый порт  
 GetClientIp(currentPair); //Получение IP клиента  
 System.out.println("New Client on port: " + currentPair.currentPort); //Сообщение о подключении  
 System.out.println("Client IP: " + currentPair.currentIp);  
  
 //Создание потоков обмена данными  
 currentPair.inputThread = new DataInputStream(currentPair.clientSocket.getInputStream());  
 currentPair.outputThread = new DataOutputStream(currentPair.clientSocket.getOutputStream());  
  
 //Авторизация клиента  
 currentPair.userLogin = ClientAuthorization(currentPair);  
  
 //Цикл обработки сообщений  
 while(flagClose != true)  
 {  
 MessageListener(currentPair);  
 }  
  
 //Завершение работы с контейнером  
 currentPair.serverSocket.close();  
 currentPair.stateBlock = false;  
 currentPair.userLogin = "null";  
  
 currentPair.inputThread.close();  
 currentPair.outputThread.close();  
  
 flagClose = false;  
  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Функция аудита сообщений сервера  
 private void MessageListener(SocketPair currentPair)  
 {  
 String messageIdentifier; //Переменная, хранящая идентификатор сервера  
  
 try  
 {  
 //Получение кода сообщения  
 messageIdentifier = currentPair.inputThread.readUTF();  
  
 //Отключение соединения  
 if(messageIdentifier.compareTo("0") == 0)  
 {  
 flagClose = true;  
 }  
 //Отправить файл  
 else if(messageIdentifier.compareTo("1") == 0)  
 {  
 currentPair.outputThread.writeUTF("2");  
 SendFile(currentPair);  
 }  
 //Запросить файл  
 else if(messageIdentifier.compareTo("2") == 0)  
 {  
 currentPair.outputThread.writeUTF("1");  
 GetFileMessage(currentPair);  
 }  
 //Переадресовать запрос файла на сервер  
 else if(messageIdentifier.compareTo("3") == 0)  
 {  
 currentPair.outputThread.writeUTF("2");  
 }  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return;  
 }  
 }  
  
 //Функция-запрос на получение файла  
 private Boolean GetFileMessage(SocketPair currentPair)  
 {  
 FileOutputStream fileStream; //Поток для файла  
 Long sizeFile; //Размер файла, который мы отправляем  
 byte[] bytesFile = new byte[16\*1024]; //Буффер для хранения файла  
 int count; //Переменная для контроля передачи  
  
 try  
 {  
 //Оповещение о начале получения файла  
 System.out.println("Start getting file..." + fileName);  
  
 //Запрос на получение файла  
 currentPair.outputThread.writeUTF(fileName);  
  
 //Выборка директории получения в зависимости от типа файла  
 if(fileName.endsWith(".mp3")){  
 fileStream = new FileOutputStream("Server\\AllMusic\\" + fileName);  
 }  
 else if(fileName.endsWith(".txt"))  
 {  
 fileStream = new FileOutputStream("Server\\" + currentPair.userLogin +"\\" + fileName);  
 }  
 else  
 {  
 return false;  
 }  
  
 //Ответ сервера на наличие файла  
 if(currentPair.inputThread.readBoolean() == true) //Файл есть и он придёт  
 {  
 //Ожидание отправки файла  
 sizeFile = currentPair.inputThread.readLong();  
  
 while (sizeFile > 0 && (count = currentPair.inputThread.read(bytesFile, 0, (int) Math.min(bytesFile.length, sizeFile))) != -1)  
 {  
 fileStream.write(bytesFile, 0, count);  
 sizeFile -= count;  
 }  
  
 //Окончание получения файла. Закрытие файлового потока.  
 fileStream.close();  
 System.out.println("File " + fileName + " " + " get!");  
 return true;  
 }  
 else //Файла нет и он не придёт  
 {  
 System.out.println("Client don't have file: " + fileName);  
 return false;  
 }  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return false;  
 }  
 }  
  
 //Функция-ответ на запрос файла от клиента  
 private Boolean SendFile(SocketPair currentPair)  
 {  
 Folder serverFolder; //Папка для извлечения файла  
 FileInputStream fileStream; //Поток для файла  
 byte[] bytesFile = new byte[16\*1024]; //Размер файла, который мы отправляем  
 Long sizeFile; //Размер файла, который мы отправляем  
 String needFile; //Строка, хранящее значение нужного клиенту файла  
 int count; //Переменная для контроля передачи  
  
 try {  
 //Забираем название файла  
 needFile = currentPair.inputThread.readUTF();  
  
 //Выбор директории. Откуда будем брать файл!?  
 serverFolder = new Folder("Server\\AllMusic\\");  
  
 //Оповешение о начале отправки файла  
 System.out.println("Client " + currentPair.currentIp + " " + currentPair.currentPort + " need file " + needFile);  
 System.out.println("Start Sending...");  
  
 //Задаём начальные признаки отправки  
 File sendFile = new File(serverFolder.folderPath + needFile);  
 sizeFile = sendFile.length();  
 fileStream = new FileInputStream(sendFile);  
  
 //Если файл существует  
 if (serverFolder.FindFile(needFile)) {  
 currentPair.outputThread.writeBoolean(true);  
 System.out.println("File exists. Sending...");  
 currentPair.outputThread.writeLong(sizeFile);  
  
 while ((count = fileStream.read(bytesFile)) != -1) {  
 currentPair.outputThread.write(bytesFile, 0, count);  
 currentPair.outputThread.flush();  
 }  
 System.out.println("File send");  
 fileStream.close();  
 return true;  
 }  
 //Если файла не существует  
 else  
 {  
 System.out.println("File doesn't exists");  
 currentPair.outputThread.writeBoolean(false);  
 return false;  
 }  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return false;  
 }  
 }  
  
 //Функция, выполняющая авторизацию пользователя  
 private String ClientAuthorization(SocketPair currentPair)  
 {  
 String userLogin = ""; //Переменная для хранения пользовательского логина  
 String userChoose = "1"; //Переменная для хранения выбора пользовтаеля  
  
 try  
 {  
 //Цикл обработки логина пользователя  
 while(userChoose.compareTo("1") == 0)  
 {  
 //Считывание логина пользователя  
 userLogin = currentPair.inputThread.readUTF();  
  
 //Создание объекта папка с нужным логином  
 Folder baseFolder = new Folder("Server\\" + userLogin);  
  
 //Проверка существования папки с логином  
 if (baseFolder.CheckFolder())  
 {  
 //Отправка метки о принятии и сопроводительного сообщения  
 currentPair.outputThread.writeBoolean(true);  
 return userLogin;  
 }  
 else  
 {  
 //Отправка метки об отказе и сопроводительного сообщения  
 currentPair.outputThread.writeBoolean(false);  
  
 //Считывание ответа пользователя  
 userChoose = currentPair.inputThread.readUTF();  
  
 //Если он предпочёл создать новый логин...  
 if(userChoose.compareTo("2") == 0)  
 {  
 baseFolder.CreateFolder(); //Создать логин по введённым данным  
 return userLogin;  
 }  
 }  
 }  
  
 return userLogin;  
 }  
 catch(IOException i)  
 {  
 System.out.println(i);  
 return "ExceptionsThrowAttic!";  
 }  
 }  
  
 //Функция, подсчитывающая клиентский IP и помещающая его в currentIP  
 private void GetClientIp(SocketPair currentPair)  
 {  
 InetSocketAddress clientAddress = (InetSocketAddress)currentPair.clientSocket.getRemoteSocketAddress();  
 currentPair.currentIp = clientAddress.getAddress().getHostAddress();  
 }  
  
 //Функция получения списка контейнеров  
 //Необходимо для работы с списком клиентов в интерфейсе  
 public SocketPair[] GetSocketList()  
 {  
 return socketMatrix;  
 }  
  
 //Функция получения максимального кол-ва контейнеров  
 public int GetContainerMax()  
 {  
 return maxBlocks;  
 }  
}

ServerWindow.java

import javax.swing.\*;  
import java.awt.event.MouseEvent;  
import java.awt.event.MouseListener;  
import java.io.\*;  
  
//Класс окна сервера  
public class ServerWindow {  
  
 private JFrame ServerFrame; //Форма для окна сервера  
 private SContainer allMusic; //Контейнер форм для хранения списка музыки  
 private SContainer allClients; //Контейнер форм для хранения списка клиентов  
 private JLabel serverState; //Строка о состоянии сервера  
  
 ServerWindow()  
 {  
 //Создание и настройка окна сервера  
 ServerFrame = new JFrame("Server Window");  
 ServerFrame.setSize(485, 600);  
 ServerFrame.setLayout(null);  
 ServerFrame.setVisible(false);  
 ServerFrame.setResizable(false);  
  
 //Настройка контейнера с аудио  
 allMusic = new SContainer(ServerFrame);  
 allMusic.SetContainerPosition(0, 0);  
 allMusic.SetContainerWidth(200);  
 allMusic.SetContainerName("Music");  
 allMusic.AddNewElement("No Music!");  
  
 //Настройка контейнера с клиентами  
 allClients = new SContainer(ServerFrame);  
 allClients.SetContainerPosition(250, 0);  
 allClients.SetContainerWidth(200);  
 allClients.SetContainerName("Clients");  
 allClients.HideButtons(false);  
 allClients.AddNewElement("No Clients!");  
 allClients.SetContainerListHigh(470);  
  
 //Добавление строки состояния  
 serverState = new JLabel();  
 serverState.setText("Change settings...");  
 serverState.setBounds(0, 530, 485, 30);  
 serverState.setVerticalAlignment(SwingConstants.CENTER);  
 serverState.setHorizontalAlignment(SwingConstants.CENTER);  
 ServerFrame.add(serverState);  
  
  
 allMusic.table.addMouseListener(new MouseListener() {  
 @Override  
 public void mouseClicked(MouseEvent evt) {  
 JList mList = (JList) evt.getSource();  
 if (evt.getClickCount() == 2) {  
 Audio playingAudio = new Audio(allMusic.allList);  
 playingAudio.PlayingMp3File("Server/AllMusic/", allMusic.table.getSelectedValue().toString());  
 }  
 }  
  
 public void mousePressed(MouseEvent evt) {}  
 public void mouseReleased(MouseEvent evt) {}  
 public void mouseEntered(MouseEvent evt) {}  
 public void mouseExited(MouseEvent evt) {}  
 });  
  
 }  
  
 //Интерфейс для показа / сокрытия окна  
 //true - показать окно / false - скрыть окно  
 public void ViewServerWindow(Boolean viewMode)  
 {  
 ServerFrame.setVisible(viewMode);  
 }  
  
 //Интерфейс для изменения строки состояния сервера  
 //Новая строка состояния  
 public void SetStringState(String enterState)  
 {  
 serverState.setText(enterState);  
 }  
  
 //Функция обновления списка клиентов на сервере  
 //Список контейнеров для работы  
 //Максимальное кол-во контейнеров  
 public void UpdateClientList(SocketPair[] actualPair, int actualSize)  
 {  
 Boolean control = false;  
  
 //Убираем все элементы из списка  
 allClients.RemoveAllElements();  
  
 //Добавляет актуальные элементы  
 for (int i = 0; i < actualSize; i++)  
 {  
 if(actualPair[i].stateBlock == true)  
 {  
 allClients.AddNewElement  
 (actualPair[i].currentPort + ":" + actualPair[i].userLogin + " Ip: " + actualPair[i].currentIp);  
  
 control = true;  
 }  
 }  
  
 //Если нет элементов в списке  
 if(control == false){  
 allClients.AddNewElement("No Clients!");  
 }  
 }  
  
 //Функция обновления списка музыки на сервере  
 //Необходимо занести в цикл для корректной работы  
 public void UpdateMusicList()  
 {  
 Boolean control = false;  
  
 //Убираем все элементы из списка  
 allMusic.RemoveAllElements();  
  
 try  
 {  
 File file = new File("Server/AllMusic/AllMusic.txt");  
 FileReader reader = new FileReader(file);  
 BufferedReader bufferedReader = new BufferedReader(reader);  
  
 String readLine = bufferedReader.readLine();  
  
 while(readLine != null)  
 {  
 allMusic.AddNewElement(readLine);  
 readLine = bufferedReader.readLine();  
 control = true;  
 }  
  
 //Елси нет элементов в списке  
 if(control == false)  
 {  
 allMusic.AddNewElement("No Music!");  
 }  
  
 }  
 catch(FileNotFoundException fe)  
 {  
 System.out.println(fe);  
 return;  
 }  
 catch(IOException IOE)  
 {  
 System.out.println(IOE);  
 return;  
 }  
 }  
}

# **Приложение Б. UML-диаграммы**

# **Диаграмма вариантов использования**

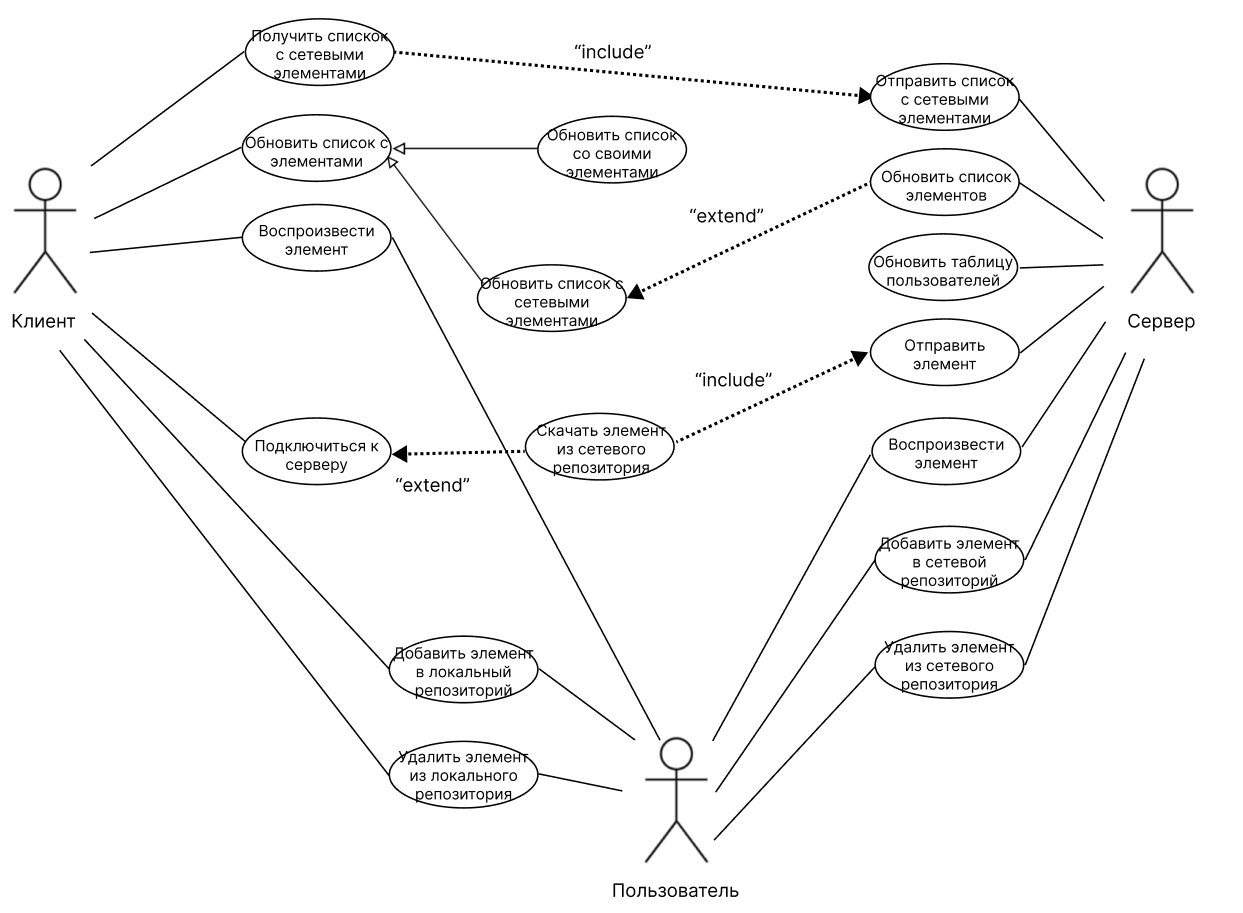


Рисунок 15 – Диаграмма вариантов использования

# **Диаграмма классов**

# **Клиент:**

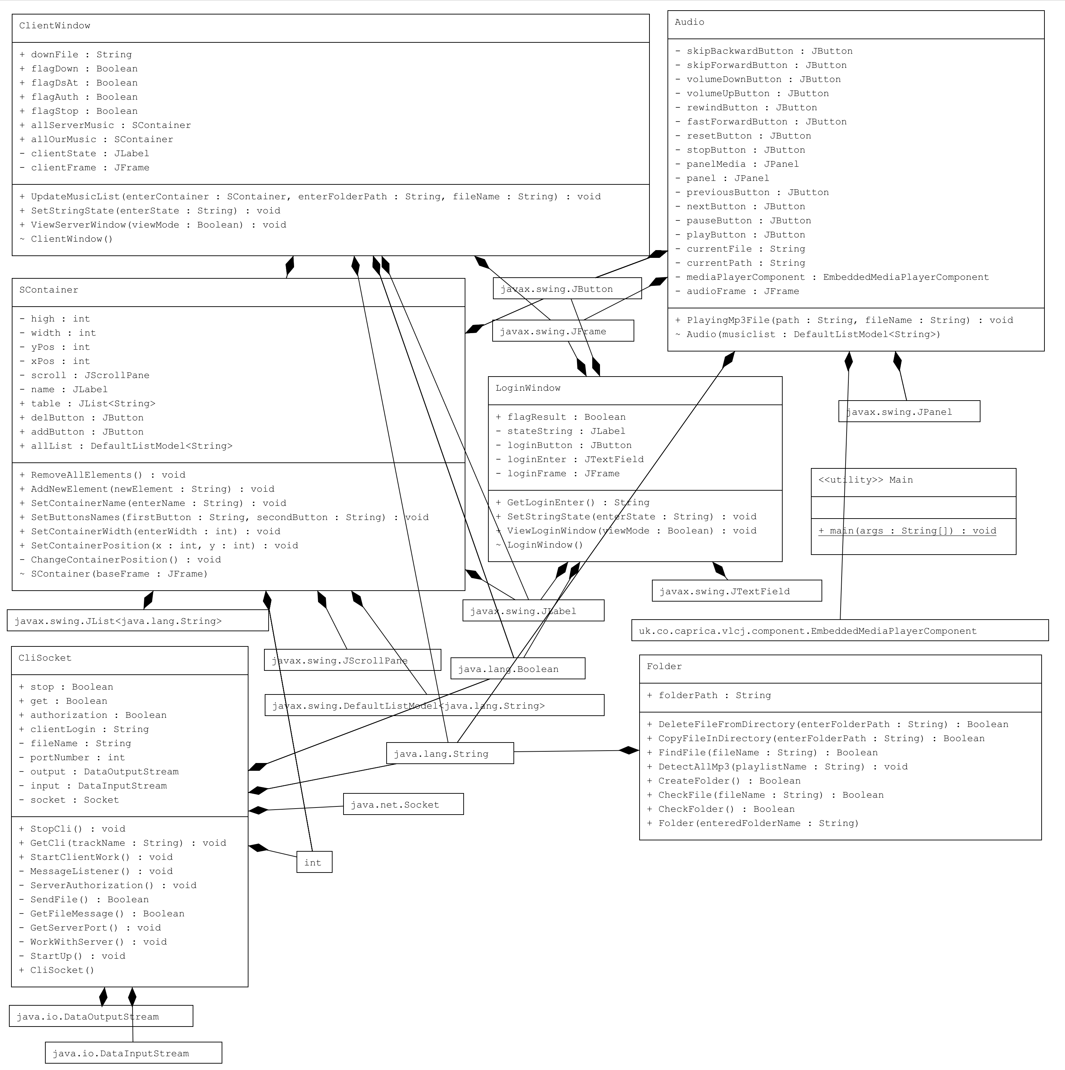


Рисунок 16 – Диаграмма классов клиента

# **Сервер:**

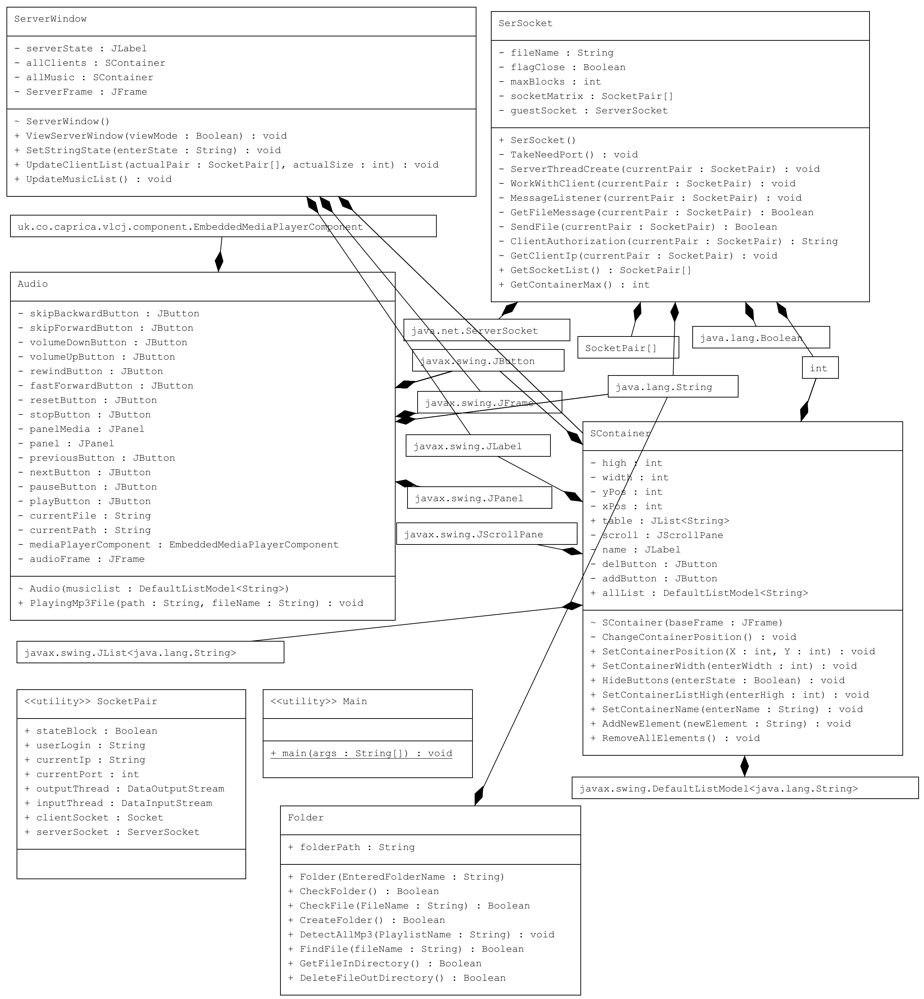


Рисунок 17 – Диаграмма классов сервера

# **Диаграмма деятельности**

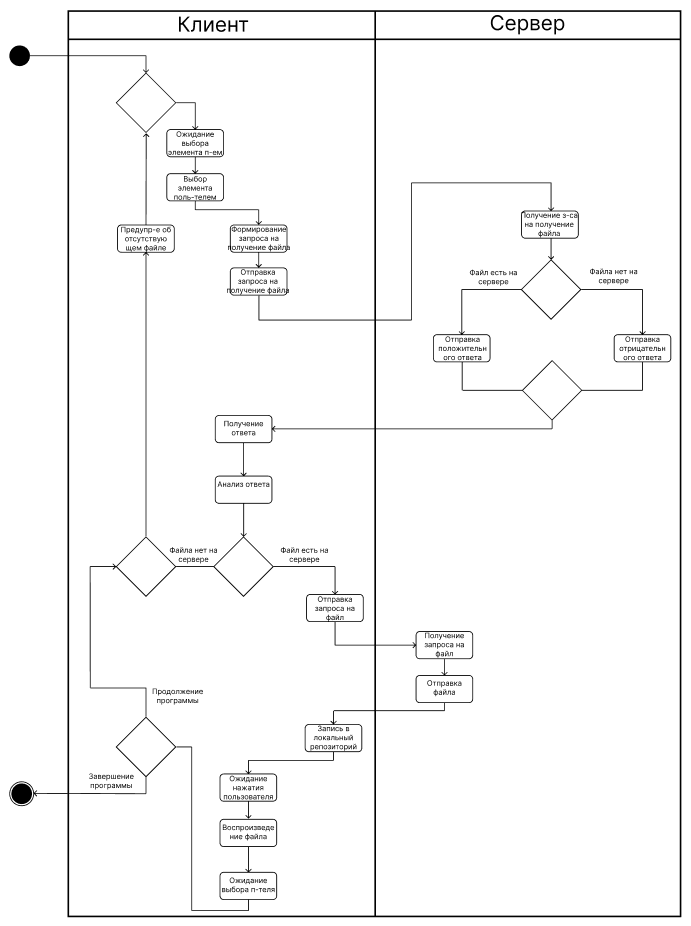


Рисунок 18 – Диаграмма деятельности

# **Диаграмма развёртывания**

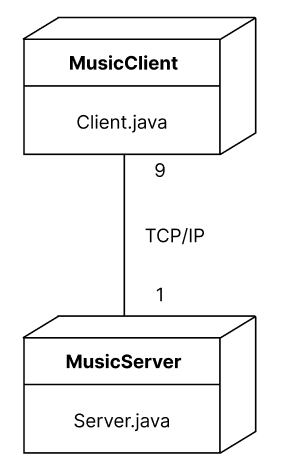


Рисунок 19 – Диаграмма развёртывания

# **Диаграмма последовательности**

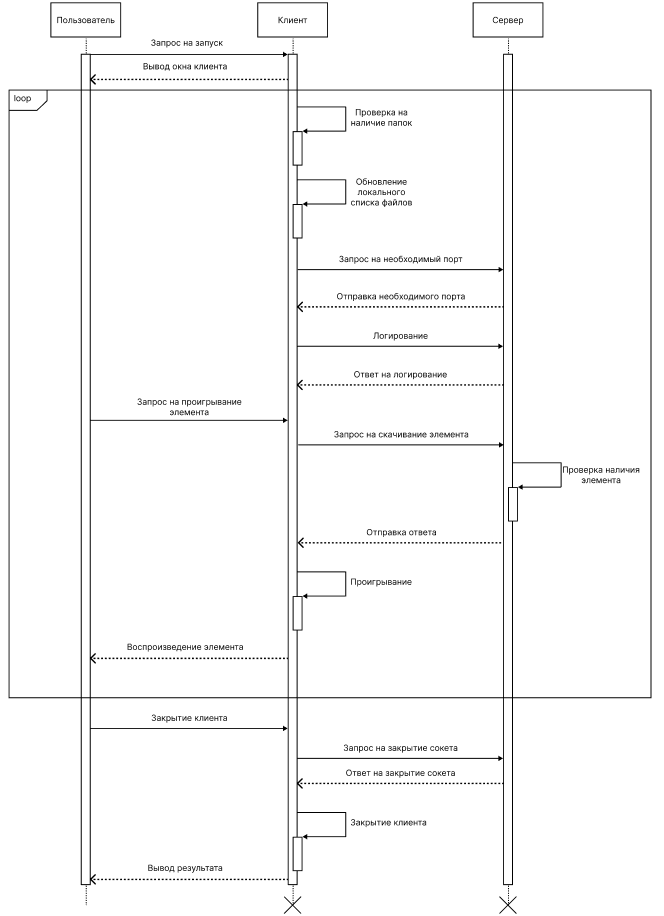


Рисунок 20 – Диаграмма последовательности