

Содержание

[Введение 5](#_Toc166486252)

[1 Постановка задачи 6](#_Toc166486253)

[2 Выбор решения 7](#_Toc166486254)

[3 Описание программы 8](#_Toc166486255)

[3.1 Серверная часть 8](#_Toc166486256)

[3.2 Клиентская часть 10](#_Toc166486257)

[4 Описание способа организации пользовательского интерфейса 11](#_Toc166486258)

[5 Описание результатов работы программы 12](#_Toc166486259)

[5.1 Запуск программы и работа с плейлистами 12](#_Toc166486260)

[5.2 Добавление, удаление и поиск композиций 14](#_Toc166486261)

[5.3 Проигрывание композиций 16](#_Toc166486262)

[5.4 Обработка исключений 18](#_Toc166486263)

[Заключение 20](#_Toc166486264)

[Список используемой литературы 21](#_Toc166486265)

[Приложение А. Исходный код 22](#_Toc166486266)

[Приложение А.1 – Исходный код Сервера 22](#_Toc166486267)

[Приложение А.2 – Исходный код Клиента 25](#_Toc166486268)

[Приложение В. UML-диаграммы приложения 35](#_Toc166486269)

[Приложение В.1 – UML-диаграмма вариантов использования 35](#_Toc166486270)

[Приложение В.2 – UML-диаграмма классов клиента 36](#_Toc166486271)

[Приложение В.3 – UML-диаграмма классов сервера 37](#_Toc166486272)

[Приложение В.4 – UML-диаграмма деятельности 38](#_Toc166486273)

[Приложение В.5 – UML-диаграмма развёртывания 39](#_Toc166486274)

[Приложение В.6 – UML-диаграмма последовательности 40](#_Toc166486275)

Введение

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Этот язык ориентирован на объекты, поддерживает ключевые концепции объектно-ориентированного программирования, такие как полиморфизм, наследование и статическая типизация. Его объектно-ориентированный подход отлично подходит для создания крупных, гибких, масштабируемых и расширяемых приложений. Одно из преимуществ Java заключается в том, что приложения, написанные на нем, не зависят от платформы. Это достигается благодаря наличию виртуальной машины Java (JVM), которая обеспечивает платформонезависимость. Хотя JVM может отличаться на разных платформах, программы на Java могут работать на любой из них.

Представленное в данном курсовом проекте приложение является простой клиент-серверной системой, которая позволяет пользователю прослушивать mp3 аудиофайлы, находящиеся на едином сервере.

1. Постановка задачи

Основной задачей данного курсового проекта является закрепление знаний и приобретение практических навыков разработки клиент-серверных приложений на языке Java. Для этого необходимо выполнить программную реализацию модульного приложения «Mp3 плеер» на языке Java в среде разработки NetBeans.

Программа должна работать так, чтобы пользователь мог прослушивать песни с сервера, при этом не хранить их у себя на машине. В проекте предусмотрен графический интерфейс и используются следующие технологии:

1. Java Collections Framework

2. Механизм обработки исключительных ситуаций

3. Java Stream API

4. Java Multithreading

5. Java Socket и Java Server Socket

1. Выбор решения

Для реализации проекта используются:

* Объектно-ориентированный язык Java;
* Библиотека Swing;
* Пакеты, содержащие файлы ресурсов, пользовательский интерфейс;
* Java Stream API;
* IDE NetBeans 20;
* Библиотека Jlayer;

Основной задачей курсового проекта являлась реализация взаимодействия клиента и сервера. Для выполнения данной задачи использовались как UDP так и TCP сокеты.

UDP сокеты используются для быстрой и простой передачи коротких и простых запросов между клиентом и сервером. Они обеспечивают быстрое взаимодействие, что идеально подходит для операций, которые не требуют высокой надежности или гарантированной доставки.

С другой стороны, TCP сокеты применяются для передачи больших объемов информации, таких как файлы, между клиентом и сервером. Протокол TCP обеспечивает надежную и точную передачу данных.

Интерфейс пользователя реализован с помощью библиотеки Java Swing и содержит в себе стандартный функционал музыкального проигрывателя:

* Поддержка работы с плейлистами;
* Воспроизведение / Пауза проигрывания;
* Регулировка громкости;
* Перемотка композиций;
* Авто воспроизведение следующий композиции в плейлисте;
* Воспроизведение композиций в случайном порядке;

1. Описание программы

Программа, состоит из клиентской и серверной части. Сначала запускается сервер, который входит в состояние ожидания запроса клиента, это можно увидеть на 4 UML диаграмме.

После чего устройствах запускается клиентская часть, которая по нажатию определенных кнопок, посылает определённый запросы к серверу, что так же отображается на 4 UML диаграмме.

* 1. Серверная часть

Server.java – файл серверного процесса, котором находится точка входа в программу, функция Main.

После входа в программу сервер сообщает о своем запуске и начинает сканирование готовых к использованию и отправке клиентам аудиофайлы и сохраняет основные данные о них в список, а так же выводит список песен в консоль, регистрирует сокеты и встаёт в ожидание запросов от клиентов.

При получении запроса сервер создает отдельную нить для его обработки. Внутри нити определяется тип запроса, и в зависимости от этого выбирается один из двух методов:

1. **SendSongListName()**: отправляет клиенту список названий всех композиций, доступных на сервере.
2. **SendPlaySong(String SongPath, long Lendth, long time)**: отправляет клиенту данные о запрошенной композиции, включая её длительность, размер данных и сам аудиофайл.

Сервер использует технологию DatagramSocket для обмена сообщениями с клиентами. При этом порты RECEIVE\_PORT и SEND\_PORT используются для приема и отправки текстовых сообщений соответственно, а порт FILE\_PORT используется для передачи аудиофайлов клиенту.

Информация о песнях загружается из папки "Songs" на сервере при его запуске. Каждая песня представлена объектом Song, содержащим имя файла, путь к файлу, размер и длительность.

Этот сервер обеспечивает эффективную передачу аудиофайлов клиентам и предоставляет им доступ к полному списку доступных композиций.

* 1. Клиентская часть

Frame.java – главный файл пользовательского процесса, содержащий функции работы программы. Также реализует интерфейс главного меню программы.

SongSelecterFrame.java – интерфейс и реализацию дополнительного окна приложения, с помощью которого производится добавление новых композиций с сервера в плейлист, а так же поиск песни из списка.

Так же клиентская часть содержит следующие методы:

* void PlayNextSong() – содержит функционал выбора следующей песни для автоматического проигрывания;
* void PlaySong(String SongName) – содержит функционал начала проигрывания новой композиции (отправка запроса на сервер, получение данных и первый запуск);
* private void AddSongActionPerformed() – функция обработки нажатия на кнопку добавить песню, отправляет запрос на сервер и открывает дополнительное окно со списком выбора композиций;
* private void SavePlaylistActionPerformed() и private void OpenPlaylistActionPerformed() – выполняют сохранение и открытие плейлиста соответственно;
* private void GainSliderStateChanged() – обработка событий слайдера громкости;
* private void SongTimelineStateChanged() – обработка событий слайдера перемотки композиции;
* private void RemoveSongActionPerformed() – обработка удаления песни из плейлиста;
* private MuteBntActionPerformed() – обработка события включения и выключения звука;
* private void PreviousSongActionPerformed и NextSongActionPerfofmed – функции переключения песен.

1. Описание способа организации пользовательского интерфейса

В качестве среды разработки была выбрана программа NetBeans. Программа предоставляет все средства, необходимые при разработке графического приложения.

Для реализации пользовательского интерфейса была использована библиотека Swing. Эта библиотека содержит более богатый и удобный набор элементов пользовательского интерфейса, обеспечивает одинаковое восприятие конечными пользователями приложений на разных платформах.

Основные кнопки переключения песни, сохранения и загрузки плейлиста, добавления песни, удаления песни были использованы *jButton*.

Кнопки проигрывания / паузы и случайного проигрывания, а так же полного выключения звука были выбраны кнопки типа *jToggleButton*, которые имеют активное и неактивное состояния, что позволяет уменьшить количество лишних элементов управления.

Выбор громкости и перемотки композиции выполнены на основе компонента *jSlider*.

Большую часть окна приложения занимает текущий плейлист, содержащий множество композиций, для за его отображение отвечает компонент *jList*.

Так же для реализации поиска по песням был использован компонент *jTextField*.

1. Описание результатов работы программы

Среда разработки NetBeans предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна клиентской части, алгоритмом программы.

Ниже продемонстрирован результат тестирования функционала программы.

* 1. Запуск программы и работа с плейлистами

При запуске сервера в консоль выводится информация о том, что сервер запущен, а так же сервер отображает все песни хранящиеся на нем (рис. 1).

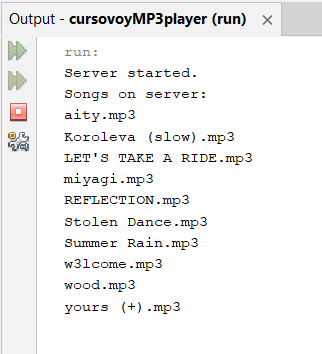


Рисунок – 1 Запуск сервера

При запуске программы клиента открывается главное меню (рис. 2).

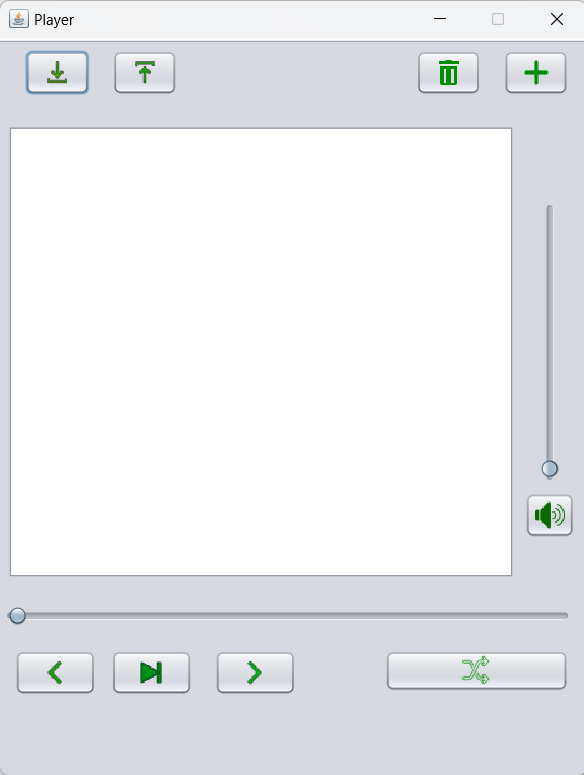


Рисунок – 2 Главное меню

При открыть или сохранить плейлист можно при помощи одноименных кнопок. Откроем готовый плейлист и посмотрим на результат (рис. 3).

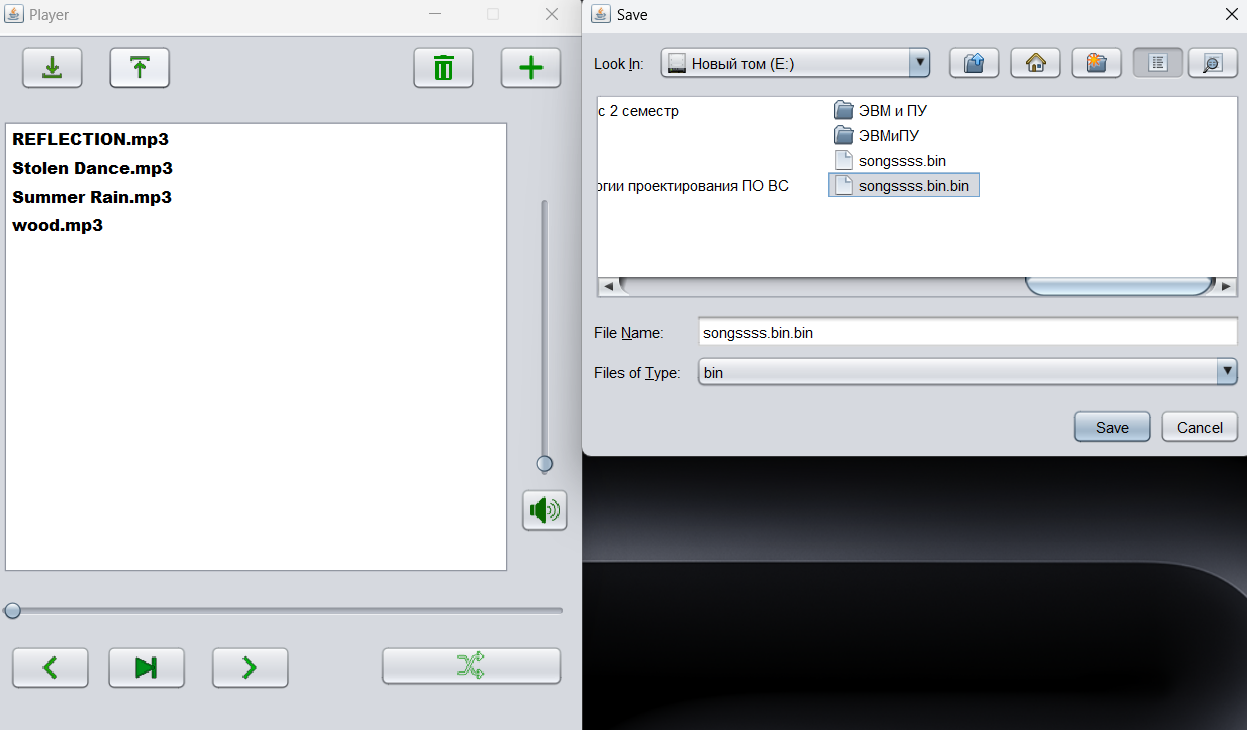


Рисунок – 3 Открытие плейлиста

* 1. Добавление, удаление и поиск композиций

Для добавления композиции используются кнопки «Добавить». Добавим в плейлист одну из всех песен находящихся на сервере (рис. 4).

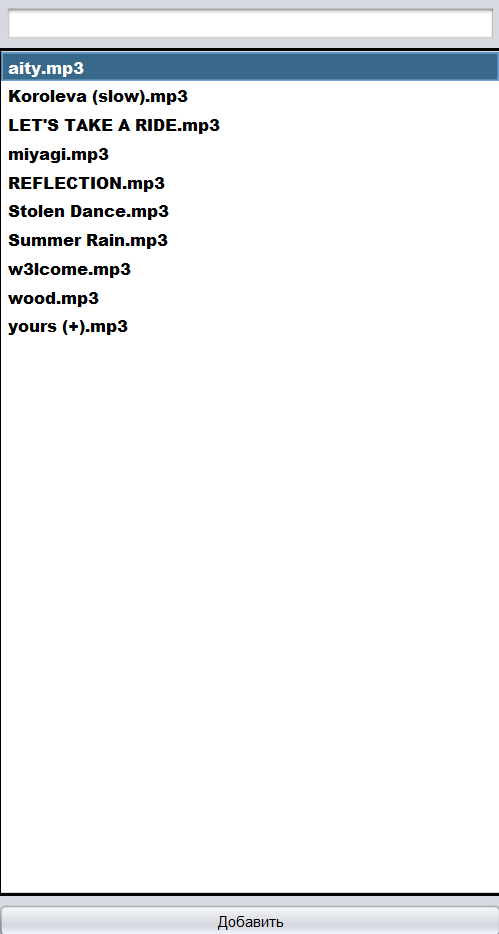
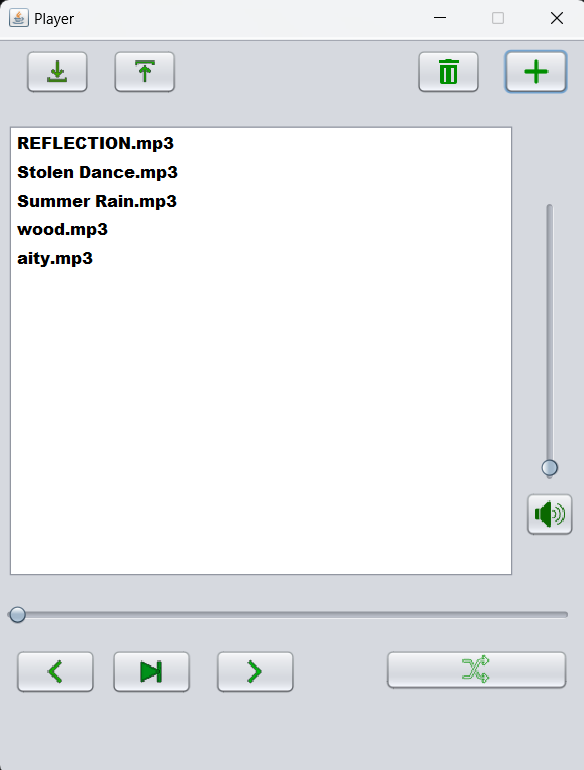
 

Рисунок – 4 Добавление песни

Для поиска нужной песни используем поле для ввода, куда вводим название искомого аудиофайла и нажимаем клавишу “Enter” (рис. 5).

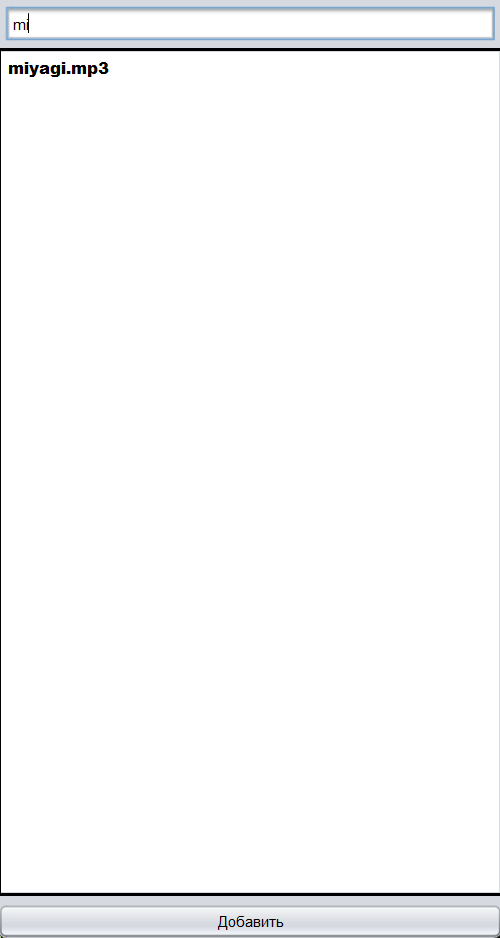


Рисунок – 5 Поиск песни

Далее удалим песню под названием “aity” (рис. 6).

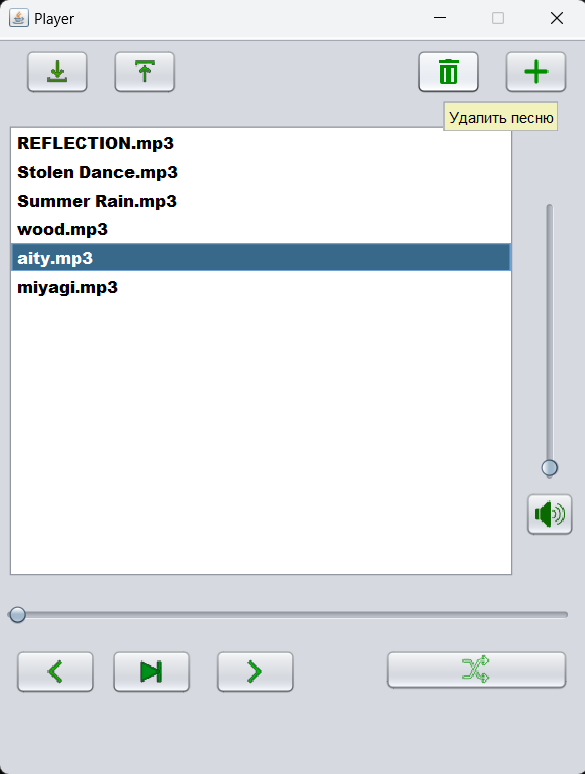
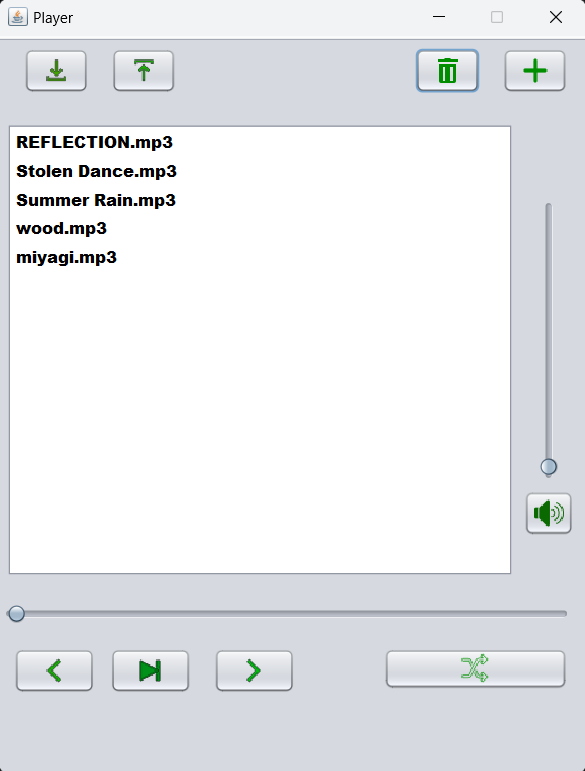
 

Рисунок – 6 Удаление песни из текущего плейлиста

* 1. Проигрывание композиций

Запустим одну из композиций из нашего плейлиста (рис. 7).

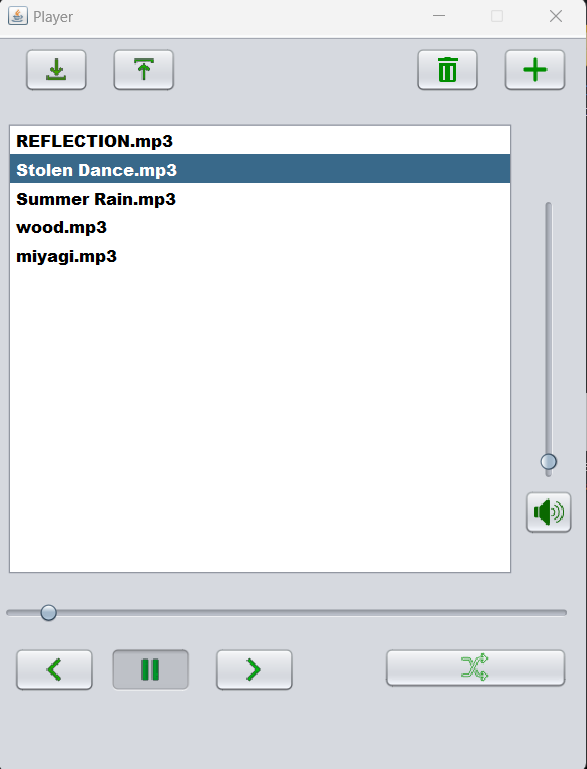


Рисунок – 7 Запуск проигрывателя

Далее перемотаем песню на середину и увеличим громкость до 40% (рис. 8).

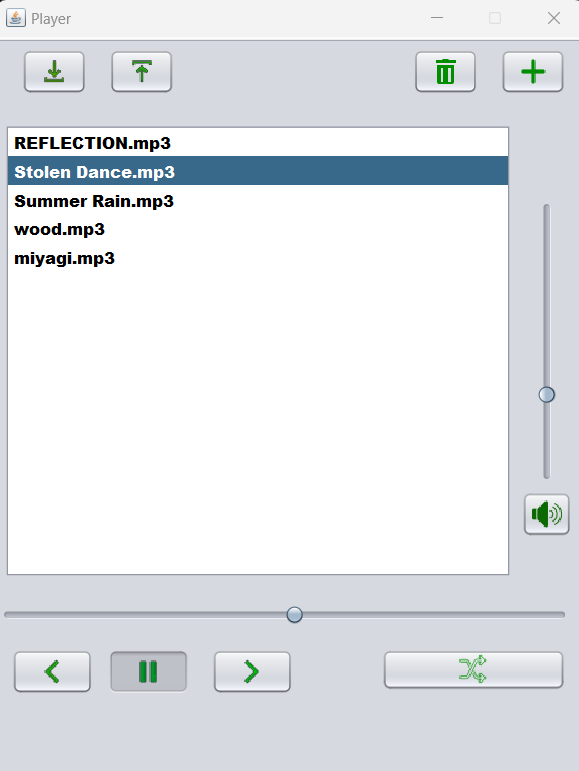


Рисунок – 8 Перемотка и регулировка звука

Нажмем кнопку случайного воспроизведения и переключим текущий трек (рис.9).

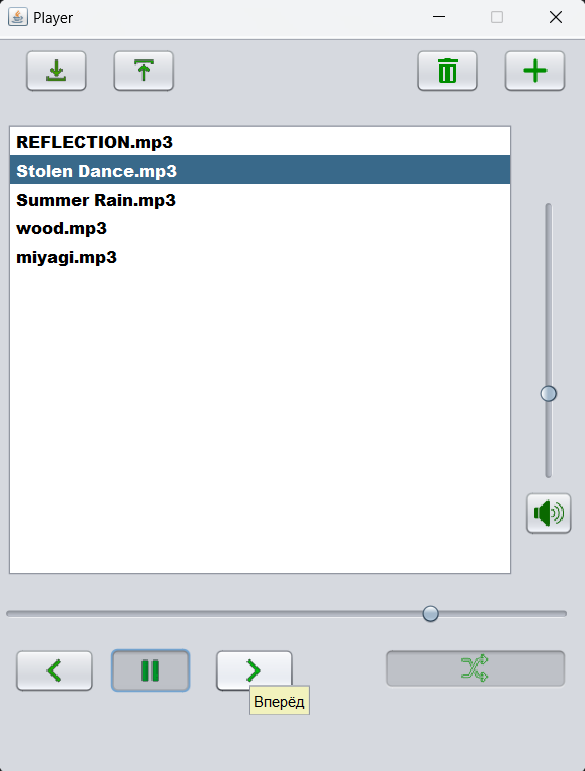
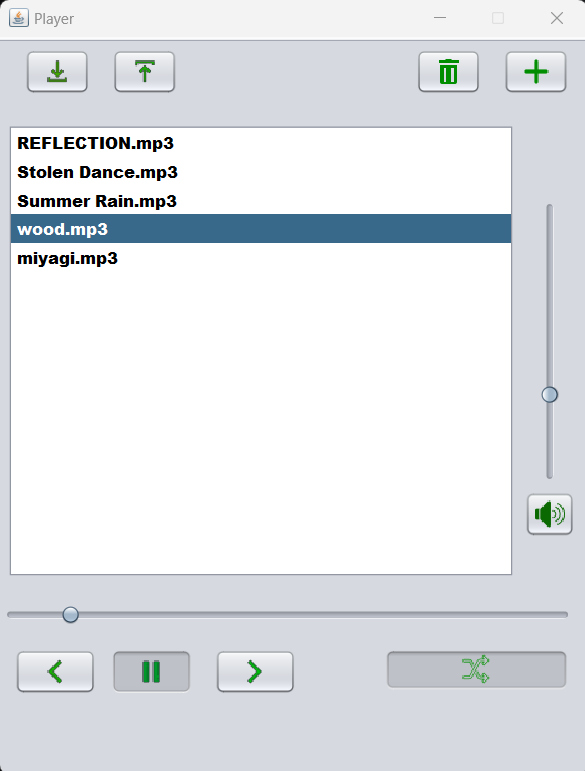
 

Рисунок 9 – Случайный выбор следующей композиции

Сервер все это время выводит в консоль информацию о том, какую песню он отсылает (рис. 10).

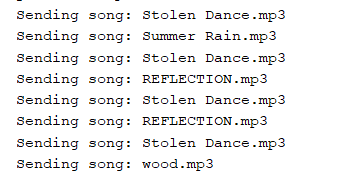


Рисунок 10 – Вывод информации в консоль

* 1. Обработка исключений

При нажатии на клавишу воспроизведения песни, учитывая, что сама песня не выбрана, появится окно с предупреждением (рис. 11).

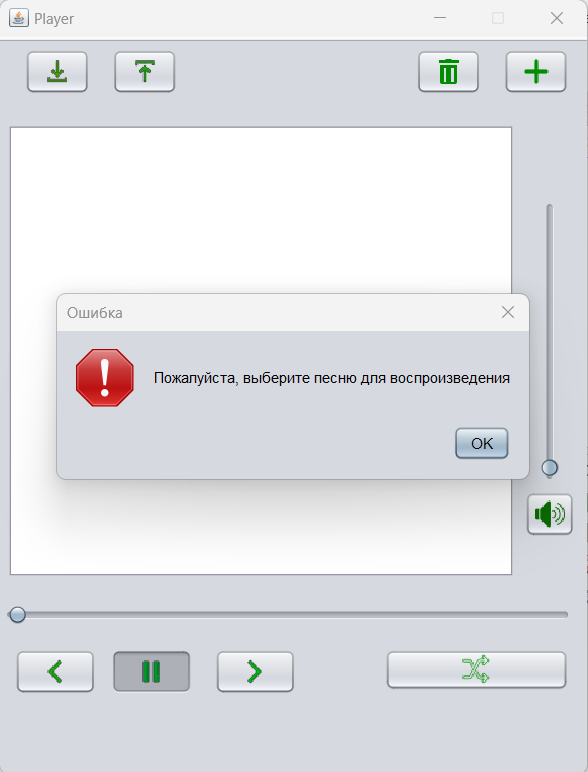


Рисунок 11 – Предупреждение о необходимости выбора песни

При попытке передвинуть ползунок громкости, учитывая, что изначально песня не была выбрана и запущена, появится окно с предупреждением (рис.12).

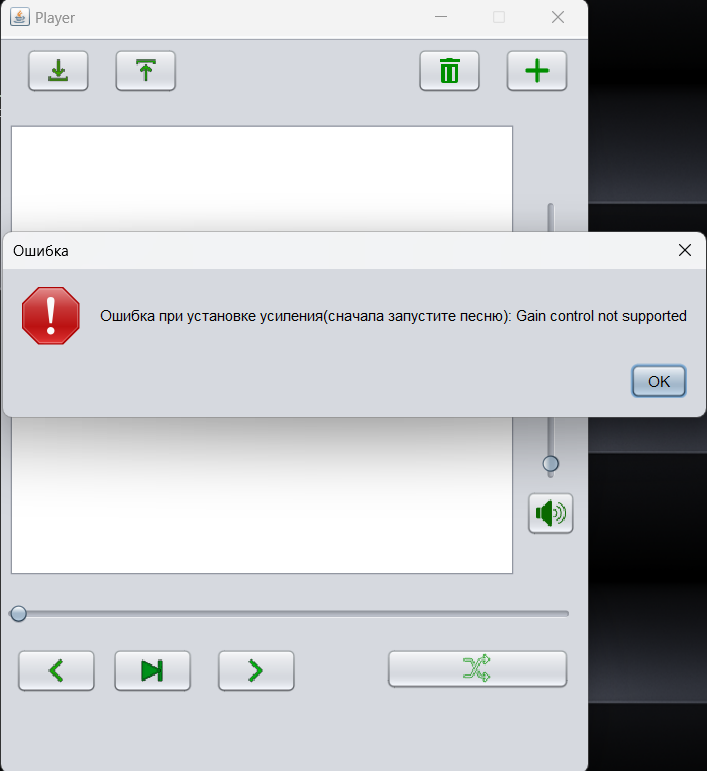


Рисунок 12 – Предупреждение о необходимости запуска песни

Заключение

При выполнении данной курсовой работы были получены навыки разработки программ на языке высокого уровня Java. Были освоены навыки разработки клиент серверных приложений на сокетах и графического интерфейса с помощью библиотеки Swing. Изучены возможности среды разработки NetBeans.

В рамках данной работы был разработан MP3 - проигрыватель, обладающий базовым функционалом, необходимым любому подобному приложению в наше время.

Список используемой литературы

1. Портянкин, Java Swing: Эффектные пользовательские интерфейсы - Издание второе, 2011, 200 с.
2. Package javax.swing // docs.oracle.com : [сайт]. – 2023. – URL: https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html (дата обращения: 01.04.2023).
3. Берд, Барри Java для чайников / Барри Берд. - М.: Диалектика / Вильямс, 2013. - 521 c.
4. Дубаков А.А. Сетевое программирование: учебное пособие / А.А. Дубаков – СП: НИУ ИТМО, 2013. – 248 с.

Приложение А. Исходный код

Приложение А.1 – Исходный код Сервера

package server;

import java.io.DataOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.Socket;

import java.util.ArrayList;

import java.util.Map;

import javax.sound.sampled.AudioFileFormat;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.UnsupportedAudioFileException;

public class Server {

private static final int RECEIVE\_PORT = 17;// Порт для приема сообщений от клиента

private static final int SEND\_PORT = 26;// Порт для отправки сообщений клиенту

private static final int FILE\_PORT = 28;// Порт для передачи файлов клиенту

private static ArrayList<Song> mainArray;

public static class Song {

public String fileName;

public String filePath;

public long fileSize;

public long duration;

public Song(String fileName, String filePath, long fileSize, long duration) {

this.fileName = fileName;

this.filePath = filePath;

this.fileSize = fileSize;

this.duration = duration;

}

}

public static void main(String[] args) {

mainArray = new ArrayList<>();

loadSongs();// Загрузка информации о песнях

System.out.println("Server started.");

System.out.println("Songs on server:");

for (Song song : mainArray) {

System.out.println(song.fileName);

}

try (DatagramSocket receiveSocket = new DatagramSocket(RECEIVE\_PORT);

DatagramSocket sendSocket = new DatagramSocket()) {

byte[] buffer = new byte[256];

// Бесконечный цикл для приема запросов от клиентов

while (true) {

DatagramPacket requestPacket = new DatagramPacket(buffer, buffer.length);

receiveSocket.receive(requestPacket);

String message = new String(requestPacket.getData(), 0, requestPacket.getLength());

InetAddress clientAddress = requestPacket.getAddress();

// Обработка запроса от клиента

handleRequest(message, clientAddress, sendSocket);

}

} catch (IOException e) {

e.printStackTrace();

}

}

// Метод для загрузки информации о песнях

private static void loadSongs() {

File directoryPath = new File("Songs");

File[] filesList = directoryPath.listFiles();

for (File file : filesList) {

try {

AudioFileFormat aff = AudioSystem.getAudioFileFormat(file.getAbsoluteFile());

Map<String, Object> param = aff.properties();

long paramTime = (long) param.get("duration");

long time = (long) Math.round(paramTime / 1000000);

// Добавление информации о песне в массив

mainArray.add(new Song(file.getName(), file.getAbsolutePath(), file.length(), time));

} catch (UnsupportedAudioFileException | IOException e) {

e.printStackTrace();

}

}

}

// Метод для обработки запроса от клиента

private static void handleRequest(String message, InetAddress clientAddress, DatagramSocket sendSocket) {

String[] parts = message.split("~");

switch (parts[0]) {

case "AddSong":

sendSongListName(clientAddress, sendSocket);

break;

case "PlaySong":

sendRequestedSong(parts[1], clientAddress, sendSocket);

break;

default:

break;

}

}

// Метод для отправки списка песен клиенту через sendMessage

private static void sendSongListName(InetAddress clientAddress, DatagramSocket sendSocket) {

StringBuilder message = new StringBuilder();

for (Song song : mainArray) {

message.append(song.fileName).append("~");

}

sendMessage(message.toString(), clientAddress, sendSocket);

}

// Метод для отправки запрошенной песни клиенту

private static void sendRequestedSong(String songName, InetAddress clientAddress, DatagramSocket sendSocket) {

for (Song song : mainArray) {

if (song.fileName.equals(songName)) {

try (Socket fileSocket = new Socket(clientAddress, FILE\_PORT);

DataOutputStream outFile = new DataOutputStream(fileSocket.getOutputStream());

FileInputStream fileInputStream = new FileInputStream(song.filePath)) {

System.out.println("Sending song: " + song.fileName);

// Отправляем размер файла и его продолжительность

outFile.writeInt((int) song.fileSize);

outFile.writeInt((int) song.duration);

// Отправляем содержимое файла

byte[] buffer = new byte[1024];

int bytesRead;

while ((bytesRead = fileInputStream.read(buffer)) != -1) {

outFile.write(buffer, 0, bytesRead);

}

} catch (IOException e) {

e.printStackTrace();

}

break;

}

}

}

private static void sendMessage(String message, InetAddress clientAddress, DatagramSocket sendSocket) {

try {

byte[] buffer = message.getBytes();

DatagramPacket packet = new DatagramPacket(buffer, buffer.length, clientAddress, SEND\_PORT);

sendSocket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

}

}

Приложение А.2 – Исходный код Клиента

**Файл Form.java**

import java.awt.event.InputEvent;

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

import java.io.DataInputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.ServerSocket;

import java.net.Socket;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.Map;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.DefaultListModel;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

import javax.swing.filechooser.FileNameExtensionFilter;

import javazoom.jlgui.basicplayer.BasicController;

import javazoom.jlgui.basicplayer.BasicPlayer;

import javazoom.jlgui.basicplayer.BasicPlayerEvent;

import javazoom.jlgui.basicplayer.BasicPlayerException;

import javazoom.jlgui.basicplayer.BasicPlayerListener;

public class Frame extends javax.swing.JFrame implements BasicPlayerListener {

public Frame() throws SocketException, UnknownHostException, IOException {

Song = new PlayableSonInfo();

Song.Player = new BasicPlayer();

Song.PlayerControl =(BasicController)Song.Player;

Song.Player.addBasicPlayerListener(this);

Song.Gain = 0.1;

socet = new DatagramSocket();

ResSocket = new DatagramSocket(26);

address = InetAddress.getByName("localhost");

FileInputSocket = new ServerSocket(28);

initComponents();

PlaylistModule = (DefaultListModel) jList1.getModel();

int GainInt = GainSlider.getValue();

double NewGain = (double)GainInt /(double)1000;

Song.Gain = NewGain;

}

private void SavePlaylistActionPerformed(java.awt.event.ActionEvent evt) {

OpenDialogBox odb = new OpenDialogBox();

String FileName = odb.DialogSave("bin") + ".bin";

ArrayList<String> arr = new ArrayList<String>();

try

{

FileOutputStream fos = new FileOutputStream(FileName);

BufferedOutputStream bis = new BufferedOutputStream(fos);

ObjectOutputStream oos = new ObjectOutputStream(bis);

for (int i = 0; i < PlaylistModule.size(); i++)

{

arr.add((String) PlaylistModule.getElementAt(i));

}

oos.writeObject(arr);

oos.close();

}

catch (FileNotFoundException ex)

{

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

catch (IOException ex)

{

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

void PlayNextSong() throws IOException{

int nextIndex = 0;

if(PlaylistModule.getSize()>0){

if(RandomPlaylistChoose.isSelected()){

String SongName = GetRandomSong();

PlaySong(SongName);

PlayPauseBnt.setSelected(true);

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/пауза.png")));

return;

}

for(int i = 0; i < PlaylistModule.getSize(); i++){

if(Song.Name.equals(PlaylistModule.get(i))){

nextIndex = i + 1;

break;

}

}

if(nextIndex <= PlaylistModule.getSize() -1){

jList1.setSelectedIndex(nextIndex);

try {

PlayPauseBnt.setSelected(true);

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/пауза.png")));

PlaySong(jList1.getSelectedValue());

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

}

private void NextSongActionPerformed(java.awt.event.ActionEvent evt) {

try {

PlayNextSong();

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void PlayPauseBntActionPerformed(java.awt.event.ActionEvent evt) {

if(PlayPauseBnt.isSelected()){

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/пауза.png")));

String SongName = jList1.getSelectedValue();

if(SongName == null && RandomPlaylistChoose.isSelected())

SongName = GetRandomSong();

if(SongName != null && !SongName.equals(Song.Name)){

try {

PlaySong(SongName);

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

if (SongName != null && SongName.equals(Song.Name)) {

try {

Song.PlayerControl.resume();

} catch (BasicPlayerException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

} else {

JOptionPane.showMessageDialog(this, "Пожалуйста, выберите песню для воспроизведения", "Ошибка", JOptionPane.ERROR\_MESSAGE);

PlayPauseBnt.setSelected(false); //сбрасываем состояние кнопки

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/Играть.png")));

try {

Song.PlayerControl.pause();

} catch (BasicPlayerException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

else{

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/Играть.png")));

try {

Song.PlayerControl.pause();

} catch (BasicPlayerException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

private void AddSongActionPerformed(java.awt.event.ActionEvent evt) {

String buf = "AddSong";

byte[] message = buf.getBytes();

DatagramPacket packet = new DatagramPacket(message, message.length, address, 17);

try {

socet.send(packet);

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

byte[] buffer = new byte[8196];

DatagramPacket request = new DatagramPacket(buffer, buffer.length);

String Message = "";

try {

ResSocket.receive(request);

Message = new String(request.getData(), 0, request.getLength());

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

String[] SongsList = Message.split("~");

SelectFrame AddSongs = new SelectFrame();

AddSongs.setParentFrame(this);

AddSongs.setSongsInList(SongsList);

AddSongs.setVisible(true);

AddSongs.isVisible();

this.setVisible(false);

}

private void MuteBntActionPerformed(java.awt.event.ActionEvent evt) {

if(MuteBnt.isSelected()){

Gain = GainSlider.getValue();

GainSlider.setValue(0);

MuteBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/Без\_звука.png")));

}

else{

MuteBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/Звук.png")));

GainSlider.setValue(Gain);

}

}

private void RemoveSongActionPerformed(java.awt.event.ActionEvent evt) {

int index = jList1.getSelectedIndex();

if(index != -1)

{

PlaylistModule.removeElementAt(index);

}

}

private void OpenPlaylistActionPerformed(java.awt.event.ActionEvent evt) {

OpenDialogBox odb = new OpenDialogBox();

String FileName = odb.DialogSave("bin");

if(FileName != null){

ArrayList<String> arr = new ArrayList<String>();

try

{

FileInputStream fis = new FileInputStream(FileName);

BufferedInputStream bis = new BufferedInputStream(fis);

ObjectInputStream ois = new ObjectInputStream(bis);

arr = (ArrayList<String>) ois.readObject();

PlaylistModule.clear();

for (int i = 0; i < arr.size(); i++)

PlaylistModule.addElement((String)arr.get(i));

} catch (FileNotFoundException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

} catch (ClassNotFoundException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

private void GainSliderStateChanged(javax.swing.event.ChangeEvent evt) {

int GainInt = GainSlider.getValue();

double NewGain = (double) GainInt / (double) 1000;

Song.Gain = NewGain;

try {

Song.PlayerControl.setGain(NewGain);

} catch (BasicPlayerException ex) {

// Обработка исключения

//Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

// Дополнительные действия, например, вывод сообщения об ошибке или возвращение к предыдущему значению усиления

JOptionPane.showMessageDialog(this, "Ошибка при установке усиления(сначала запустите песню): " + ex.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);

// Возвращаем ползунок к предыдущему значению

GainSlider.setValue((int) (Song.Gain \* 1000));

}

}

private void PreviousSongActionPerformed(java.awt.event.ActionEvent evt) {

int nextIndex = 0;

if(PlaylistModule.getSize()>0){

if(RandomPlaylistChoose.isSelected()){

String SongName = GetRandomSong();

try {

PlaySong(SongName);

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

PlayPauseBnt.setSelected(true);

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/пауза.png")));

return;

}

for(int i = 0; i < PlaylistModule.getSize(); i++){

if(Song.Name.equals(PlaylistModule.get(i))){

nextIndex = i - 1;

break;

}

}

if(nextIndex > -1){

jList1.setSelectedIndex(nextIndex);

try {

PlayPauseBnt.setSelected(true);

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/пауза.png")));

PlaySong(jList1.getSelectedValue());

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

}

private void SongTimelineStateChanged(javax.swing.event.ChangeEvent evt) {

if(SongTimeline.getValueIsAdjusting() == false){

if(AutoJump == true){

try {

AutoJump = false;

//DurationJump = true;

double sliderPos = SongTimeline.getValue()\*1.0 / 1000;

long bytesToskip = (long) Math.round(((long) Song.\_Size) \*sliderPos);

//CurSong.PlayerControl.pause();

Song.PlayerControl.seek(bytesToskip);

//CurSong.PlayerControl.resume();

Song.PlayerControl.setGain(Song.Gain);

} catch (BasicPlayerException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

else{

AutoJump = true;

\_Jump = true;

}

}

private void jList1MouseClicked(java.awt.event.MouseEvent evt) {

if(evt.getModifiers() == InputEvent.BUTTON1\_MASK && evt.getClickCount() == 2){

if(PlayPauseBnt.isSelected()){

try {

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/Играть.png")));

Song.PlayerControl.stop();

} catch (BasicPlayerException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

else{

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/пауза.png")));

PlayPauseBnt.setSelected(true);

}

String SongName = jList1.getSelectedValue();

try {

PlaySong(SongName);

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

private void RandomPlaylistChooseActionPerformed(java.awt.event.ActionEvent evt) {

}

public static void main(String args[]) {

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Frame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Frame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Frame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Frame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

try {

new Frame().setVisible(true);

} catch (SocketException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

} catch (UnknownHostException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

});

}

@Override

public void opened(Object o, Map map) {

Song.PlayStatus = true;

SongTimeline.setValue(0);

}

@Override

public void progress(int i, long l, byte[] bytes, Map map) {

float progress = 1.0f;

long bytesread = (long) map.get("mp3.position.byte");

if((bytesread>0) && (Song.duration > 0))

progress = bytesread\*1.0f/Song.\_Size\*1.0f;

long secondAmount = (long) (Song.duration \* progress);

if(Song.duration !=0){

if(\_Jump == false){

SongTimeline.setValue((int) Math.round((secondAmount)\*1000 / Song.duration));

}

}

}

@Override

public void stateUpdated(BasicPlayerEvent bpe) {

int state = bpe.getCode();

switch(state){

case BasicPlayerEvent.PLAYING:{

\_Jump = false;

break;

}

case BasicPlayerEvent.SEEKING:{

\_Jump = true;

break;

}

case BasicPlayerEvent.EOM:{

Song.PlayStatus = false;

PlayPauseBnt.setSelected(false);

PlayPauseBnt.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Icons/Играть.png")));

try {

PlayNextSong();

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

break;

}

}

}

@Override

public void setController(BasicController bc) {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

String GetRandomSong(){

while(true){

int SongNum = (int) (Math.random()\*PlaylistModule.getSize());

if(!PlaylistModule.get(SongNum).equals(Song.Name)){

jList1.setSelectedIndex(SongNum);

return (String) PlaylistModule.get(SongNum);

}

}

}

class PlayableSonInfo

{

String Name;

byte[] \_Data;

long \_Size;

long duration;

BasicController PlayerControl;

BasicPlayer Player;

double Gain;

boolean PlayStatus;

}

class OpenDialogBox

{

String FileName;

public String DialogSave(String NameFilter)

{

JFileChooser chooser = new JFileChooser();

FileNameExtensionFilter filter = new FileNameExtensionFilter(

"bin", NameFilter);

chooser.setFileFilter(filter);

int returnVal = chooser.showSaveDialog(null);

if(returnVal == JFileChooser.APPROVE\_OPTION)

{

FileName = chooser.getSelectedFile().getAbsolutePath();

}

return FileName;

}

}

void PlaySong(String SongName) throws IOException{

try {

Song.PlayerControl.stop();

String buf = "PlaySong" + "~" + SongName;

byte[] message = buf.getBytes();

DatagramPacket packet = new DatagramPacket(message, message.length, address, 17);

socet.send(packet);

try {

Socket InSocket = FileInputSocket.accept();

DataInputStream Data = new DataInputStream(InSocket.getInputStream());

int FileSize = Data.readInt();

int FileTime = Data.readInt();

if (FileSize > 0){

byte[] FileContent = new byte[FileSize];

Data.readFully(FileContent, 0, FileSize);

Song.Name = SongName;

Song.\_Data = FileContent;

Song.\_Size = FileSize;

Song.duration = FileTime;

InSocket.close();

File Buffer = new File("Song.mp3");

FileOutputStream fos = new FileOutputStream(Buffer);

fos.write(Song.\_Data);

fos.close();

InputStream a = new FileInputStream(Buffer);

Song.PlayerControl.open(Buffer);

Song.PlayerControl.play();

Song.PlayerControl.setGain(Song.Gain);

}

} catch (IOException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

} catch (BasicPlayerException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

} catch (BasicPlayerException ex) {

Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);

}

}

boolean AutoJump = false;

boolean \_Jump = false;

ServerSocket FileInputSocket;

DatagramSocket socet, ResSocket;

InetAddress address;

DefaultListModel PlaylistModule;

PlayableSonInfo Song;

int Gain;

// Variables declaration - do not modify

private javax.swing.JButton AddSong;

private javax.swing.JSlider GainSlider;

private javax.swing.JPanel MenuLowerPart;

private javax.swing.JPanel MenuUpperPart;

private javax.swing.JToggleButton MuteBnt;

private javax.swing.JButton NextSong;

private javax.swing.JButton OpenPlaylist;

private javax.swing.JToggleButton PlayPauseBnt;

private javax.swing.JPanel PlaylistFrame;

private javax.swing.JButton PreviousSong;

private javax.swing.JToggleButton RandomPlaylistChoose;

private javax.swing.JButton RemoveSong;

private javax.swing.JButton SavePlaylist;

private javax.swing.JSlider SongTimeline;

private javax.swing.JList<String> jList1;

private javax.swing.JMenuBar jMenuBar1;

private javax.swing.JScrollPane jScrollPane1;

// End of variables declaration

}

**Файл SelectFrame.java**

package Gui;

import java.util.ArrayList;

import javax.swing.DefaultListModel;

public class SelectFrame extends javax.swing.JFrame {

private ArrayList<String> songList = new ArrayList<>();

public void setParentFrame(Frame frame) {

\_aThis = frame;

}

public void setSongsInList(String[] AddList){

songList.clear();

DefaultListModel<String> SongListmodule = (DefaultListModel<String>) ListWindow.getModel();

SongListmodule.clear();

for (int i = 0; i < AddList.length; i++) {

songList.add(AddList[i]);

SongListmodule.addElement(AddList[i]);

}

}

Frame \_aThis;

public SelectFrame() {

initComponents();

}

@SuppressWarnings("unchecked")

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

DefaultListModel<String> SongListmodule = (DefaultListModel<String>) ListWindow.getModel();

int[] SelectedItems;

SelectedItems = ListWindow.getSelectedIndices();

for (int i = 0; i < SelectedItems.length; i++) {

String SongToAdd = (String) SongListmodule.getElementAt(SelectedItems[i]);

\_aThis.PlaylistModule.addElement(SongToAdd);

}

this.dispose();

\_aThis.setVisible(true);

}

private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {

String searchText = jTextField1.getText();

filterSongs(searchText);

}

private void filterSongs(String searchText) {

DefaultListModel<String> model = (DefaultListModel<String>) ListWindow.getModel();

model.clear();

for (String song : songList) {

if (song.toLowerCase().contains(searchText.toLowerCase())) {

model.addElement(song);

}

}

}

public static void main(String args[]) {

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(SelectFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(SelectFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(SelectFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(SelectFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new SelectFrame().setVisible(true);

}

});

}

private javax.swing.JList<String> ListWindow;

private javax.swing.JButton jButton1;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JTextField jTextField1;

}

Приложение В. UML-диаграммы приложения

Приложение В.1 – UML-диаграмма вариантов использования



Рисунок 13 – UML-Диаграмма вариантов использования

Приложение В.2 – UML-диаграмма классов клиента

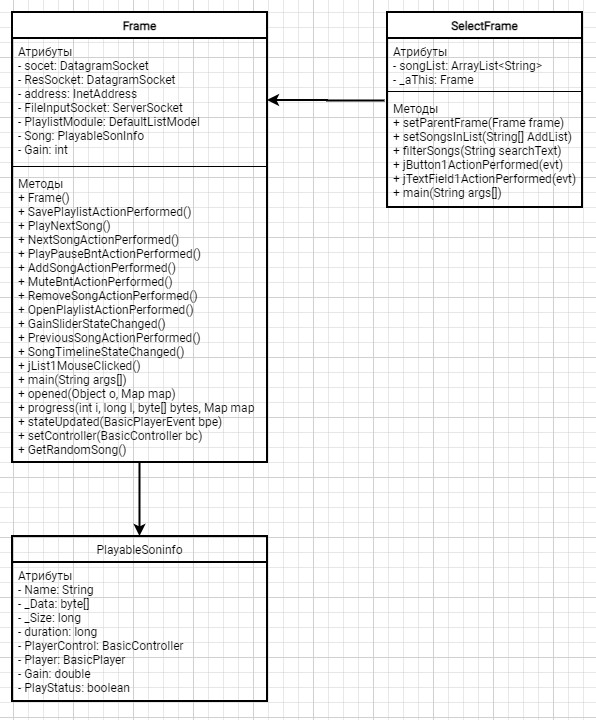


Рисунок 14 – UML-диаграмма классов клиента

Приложение В.3 – UML-диаграмма классов сервера

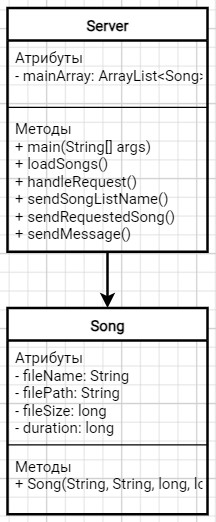


Рисунок 15 – UML-диаграмма классов сервера

Приложение В.4 – UML-диаграмма деятельности



Рисунок 16 – UML-диаграмма деятельности

Приложение В.5 – UML-диаграмма развёртывания



Рисунок 17 – UML-диаграмма развёртывания

Приложение В.6 – UML-диаграмма последовательности



Рисунок 18 – UML-диаграмма последовательности