Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Пояснительная записка**

к курсовой работе

по курсу «Программирование на языке Java»

на тему «Разработка многомодульного приложения на языке Java»

Выполнил:

Студент группы 21ВВП1  
Зайкин С.А

Приняли:

к.т.н., доцент Юрова О. В.

к.т.н., доцент Карамышева Н. С.

Пенза 2024

Содержание

[Введение 3](#_Toc169349024)

[1 Постановказадачи 4](#_Toc169349025)

[2 Выбор решения 5](#_Toc169349026)

[3 Описание программы 6](#_Toc169349027)

[3.1 Серверная часть 6](#_Toc169349028)

[3.2 Клиентская часть 6](#_Toc169349029)

[4 Описание способа организации пользовательского интерфейса 8](#_Toc169349030)

[5 Описание результатов работы программы 9](#_Toc169349031)

[5.1 Запуск программы и работа с плейлистами 9](#_Toc169349032)

[5.2 Проигрывание композиций 9](#_Toc169349033)

[Заключение 11](#_Toc169349034)

[Список используемой литературы 12](#_Toc169349035)

[Приложение А. Исходный код 13](#_Toc169349036)

[ПриложениеА.1 –Исходный код Сервера 13](#_Toc169349037)

[Приложение А.2–Исходный код Клиента 14](#_Toc169349038)

[Приложение В. UML-диаграммы приложения 17](#_Toc169349039)

[ПриложениеВ.1 – UML-диаграмма вариантов использования 17](#_Toc169349040)

[Приложение В.2 – UML-диаграмма классов клиента 18](#_Toc169349041)

[Приложение В.3 – UML-диаграмма классов сервера 19](#_Toc169349042)

[Приложение В.4 – UML-диаграмма деятельности 20](#_Toc169349043)

[Приложение В.5 – UML-диаграмма развёртывания 21](#_Toc169349044)

[Приложение В.6 – UML-диаграмма последовательности 22](#_Toc169349045)

Введение

Сегодня Java является одним из самых распространенных и популярных языков программирования, который поддерживает объектно-ориентированный подход, полиморфизм, наследование и статическую типизацию. Это позволяет разработчикам создавать гибкие, масштабируемые и расширяемые приложения. Кроме того, приложения на Java не зависят от платформы, благодаря существованию виртуальной машины Java (JVM) на каждой поддерживаемой платформе.

Представленная в данном курсовом проекте программа является клиент-серверным приложением, которое позволяет прослушивать mp3 аудиофайлы, хранящиеся на сервере.

1. Постановказадачи

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 IntelliJ IDEA.

Программа должна работать так, чтобы пользователь мог прослушивать песни при этом не хранит все аудиофайлы у себя на устройстве. В проекте предусмотрен графический интерфейс и используются следующие технологии:

1. Java Collections Framework

2. Механизм обработки исключительных ситуаций

3. JavaStreamAPI

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 IntelliJ IDEA 1.3;
* Библиотека BasicPlayer;

Основной задачей курсового проекта являлась реализация взаимодействия клиента и сервера. Для выполнения данной задачи использовались как UDPтак и TCPсокеты.

Взаимодействие между клиентом и сервером должно происходить по принципу “Запрос-Ответ”. При этом для коротких и простых запросов используются более быстрый и простой протокол UDP, а для передаче с сервера больших объёмов информации (например файлов), был использован протокол TCP, способный точно передавать большие объёмы данных.

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

* Поддержка работы с плейлистами;
* Воспроизведение / Пауза проигрывания;
* Авто воспроизведение следующий композиции в плейлисте;
* Выбор композиции для исполнения;

1. Описание программы

Программа, состоит из клиентской и серверной части. Сначала запускается сервер, который входит в состояние ожидания запроса клиента.

После чего устройствах запускается клиентская часть, которая по нажатию определенных кнопок, посылает определённый запросы к серверу. Более подробно указано в UML - диаграмме вариантов использования (Приложение B1).

* 1. Серверная часть

MP3Server.java – главный файл серверного процесса, котором находится точка входа в программу, функция Main(Приложение B3).

После входа в программу сервер сканирует готовые к использованию и отправке клиентам аудиофайлы и сохраняет основные данные о них в список, регистрирует сокеты и встаёт в ожидание запросов от клиентов.

При получении запроса сервер создаёт нить для его обработки. Метод run(), находящийся в нити, в котором происходит определение запроса, на основе которого нить запускает метод:

* ClientHandler() – отправляет клиенту названия всех композиций, находящихся на сервере и отправляет клиенту данные о запрошенной композиции (длительность и размер данных) и сами данные с аудио файлом;
  1. Клиентская часть

MP3Client.java – главный файл пользовательского процесса, содержащий функции работы программы. Также реализует интерфейс главного меню программы(Приложение B2).

Так же клиентская часть содержит следующие методы:

* Private void loadFileList() - содержит функционал получения списка имеющихся композиций.
* Private void next(ActionEvent e) – содержит функционал выбора следующей песни;
* Private void reset(ActionEvent e)- содержит функционал возобновления играющей композиции;
* Private void stop(ActionEvent e)- содержит функционал остановки играющей композиции;
* Private void PlayFile (String File name)– содержит функционал начала проигрывания новой композиции (отправка запроса на сервер, получение данных и первый запуск).

1. Описание способа организации пользовательского интерфейса

В качестве среды разработки была выбрана программа IntelliJ IDEA. 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 воспроизведение композиции, остановки композиции, переключение на следующую композицию, возобновление играющей композиции были использованы *jButton*.

Отображение статуса исполнения *jSlider*.

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

1. Описание результатов работы программы

Среда разработки IntelliJ IDEA 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 программы открывается главное меню(рис. 1).

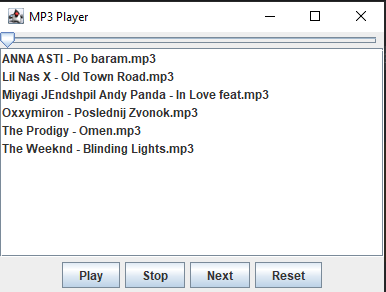


Рисунок 1 – Главное меню приложения

* 1. Проигрывание композиций

Запустим одну из композиций из нашего плейлиста (рис. 2).

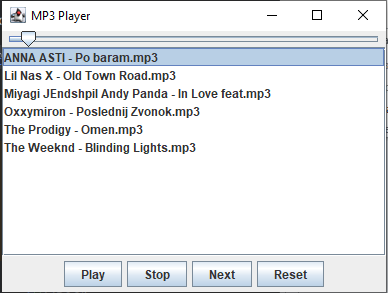


Рисунок 2 –Запуск проигрывателя

Пропустим песню и запустим следующую по порядку (рис.3).

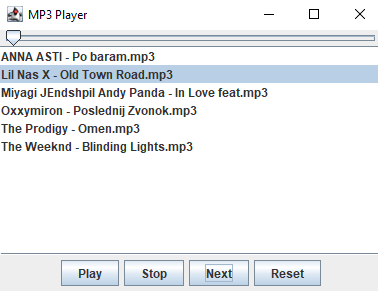


Рисунок 3 – Запуск следующей композиции

Как можно заметить следующая, идущая по порядку в плейлисте композиция была выбрана.

Заключение

Были реализованы следующие функции:

1. Прослушивание mp3 аудиофайлов, хранящихся на сервере.

2. Отображение списка доступных аудиофайлов на сервере.

3. Возможность выбора аудиофайла из списка и его прослушивание.

В процессе разработки были использованы следующие технологии и инструменты:

1. Язык программирования Java.

2. Библиотека Swing для создания графического интерфейса.

3. Среда разработки IntelliJ IDEA.

Таким образом, выполнение данной курсовой работы позволило углубиться в изучение языка программирования Java и его возможностей, а также получить практические навыки разработки клиент-серверных приложений.

Список используемой литературы

1. Портянкин, Java Swing: Эффектные пользовательские интерфейсы - Издание второе, 2011, 200 с.
2. Packagejavax.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

ПриложениеА.1 –Исходный код Сервера

import java.io.\*;

import java.net.\*;

public class MP3Server {

private static final int PORT = 12345;

private static final String MUSIC\_DIR = "music"; // Укажите путь к каталогу с MP3-файлами

public static void main(String[] args) throws IOException {

ServerSocket serverSocket = new ServerSocket(PORT);

System.out.println("Server started on port " + PORT);

while (true) {

Socket clientSocket = serverSocket.accept();

new Thread(new ClientHandler(clientSocket)).start();

}

}

private static class ClientHandler implements Runnable {

private Socket socket;

public ClientHandler(Socket socket) {

this.socket = socket;

}

@Override

public void run() {

try (

DataInputStream in = new DataInputStream(socket.getInputStream());

DataOutputStream out = new DataOutputStream(socket.getOutputStream())

) {

String command = in.readUTF();

if (command.equals("LIST")) {

File dir = new File(MUSIC\_DIR);

File[] files = dir.listFiles((d, name) -> name.endsWith(".mp3"));

assert files != null;

out.writeInt(files.length);

for (File file : files) {

out.writeUTF(file.getName());

}

} else if (command.startsWith("GET ")) {

String fileName = command.substring(4);

File file = new File(MUSIC\_DIR, fileName);

if (file.exists() && file.isFile()) {

out.writeLong(file.length());

try (FileInputStream fis = new FileInputStream(file)) {

byte[] buffer = new byte[4096];

int bytesRead;

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

out.write(buffer, 0, bytesRead);

}

}

} else {

out.writeLong(0);

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

Приложение А.2–Исходный код Клиента

import javazoom.jl.decoder.JavaLayerException;

import javazoom.jl.player.advanced.AdvancedPlayer;

import javazoom.jl.player.advanced.PlaybackEvent;

import javazoom.jl.player.advanced.PlaybackListener;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.io.\*;

import java.net.Socket;

import java.util.ArrayList;

import java.util.List;

public class MP3Client extends JFrame {

private static final String SERVER\_HOST = "localhost";

private static final int SERVER\_PORT = 12345;

private JList<String> fileList;

private DefaultListModel<String> listModel;

private JButton playButton;

private JButton stopButton;

private JButton nextButton;

private JButton resetButton;

private JSlider progressBar;

private AdvancedPlayer player;

private Thread playerThread;

private volatile boolean playing;

private volatile boolean stopped;

private int currentFileIndex = 0;

public MP3Client() {

super("MP3 Player");

listModel = new DefaultListModel<>();

fileList = new JList<>(listModel);

playButton = new JButton("Play");

stopButton = new JButton("Stop");

nextButton = new JButton("Next");

resetButton = new JButton("Reset");

progressBar = new JSlider(0, 100, 0);

progressBar.setValue(0);

JPanel panel = new JPanel(new BorderLayout());

panel.add(new JScrollPane(fileList), BorderLayout.CENTER);

JPanel controlPanel = new JPanel();

controlPanel.add(playButton);

controlPanel.add(stopButton);

controlPanel.add(nextButton);

controlPanel.add(resetButton);

panel.add(controlPanel, BorderLayout.SOUTH);

panel.add(progressBar, BorderLayout.NORTH);

add(panel);

setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

playButton.addActionListener(this::play);

stopButton.addActionListener(this::stop);

nextButton.addActionListener(this::next);

resetButton.addActionListener(this::reset);

loadFileList();

// Thread to update progress bar

new Thread(() -> {

while (true) {

if (playing) {

progressBar.setValue(progressBar.getValue() + 1);

}

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}).start();

}

private void loadFileList() {

try (Socket socket = new Socket(SERVER\_HOST, SERVER\_PORT);

DataInputStream in = new DataInputStream(socket.getInputStream());

DataOutputStream out = new DataOutputStream(socket.getOutputStream())) {

out.writeUTF("LIST");

int fileCount = in.readInt();

List<String> files = new ArrayList<>();

for (int i = 0; i < fileCount; i++) {

files.add(in.readUTF());

}

listModel.clear();

files.forEach(listModel::addElement);

} catch (IOException e) {

e.printStackTrace();

}

}

private void play(ActionEvent e) {

String selectedFile = fileList.getSelectedValue();

if (selectedFile != null) {

currentFileIndex = fileList.getSelectedIndex();

playFile(selectedFile);

}

}

private void playFile(String fileName) {

try (Socket socket = new Socket(SERVER\_HOST, SERVER\_PORT);

DataInputStream in = new DataInputStream(socket.getInputStream());

DataOutputStream out = new DataOutputStream(socket.getOutputStream())) {

out.writeUTF("GET " + fileName);

long fileSize = in.readLong();

if (fileSize > 0) {

byte[] buffer = new byte[4096];

ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

int bytesRead;

while (fileSize > 0 && (bytesRead = in.read(buffer, 0, (int) Math.min(buffer.length, fileSize))) != -1) {

byteArrayOutputStream.write(buffer, 0, bytesRead);

fileSize -= bytesRead;

}

InputStream inputStream = new ByteArrayInputStream(byteArrayOutputStream.toByteArray());

if (playerThread != null && playerThread.isAlive()) {

player.close();

}

player = new AdvancedPlayer(inputStream);

playerThread = new Thread(() -> {

try {

playing = true;

stopped = false;

progressBar.setValue(0);

player.setPlayBackListener(new PlaybackListener() {

@Override

public void playbackFinished(PlaybackEvent evt) {

playing = false;

if (!stopped && currentFileIndex < listModel.getSize() - 1) {

SwingUtilities.invokeLater(() -> next(null));

}

}

});

player.play();

} catch (JavaLayerException ex) {

ex.printStackTrace();

}

});

playerThread.start();

}

} catch (IOException | JavaLayerException ex) {

ex.printStackTrace();

}

}

private void stop(ActionEvent e) {

if (player != null) {

stopped = true;

player.close();

playing = false;

progressBar.setValue(0);

}

}

private void next(ActionEvent e) {

if (currentFileIndex < listModel.getSize() - 1) {

currentFileIndex++;

fileList.setSelectedIndex(currentFileIndex);

playFile(fileList.getSelectedValue());

}

}

private void reset(ActionEvent e) {

stop(null);

if (currentFileIndex < listModel.getSize()) {

playFile(fileList.getSelectedValue());

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(MP3Client::new);

}

}

Приложение В. UML-диаграммы приложения

ПриложениеВ.1 – UML-диаграмма вариантов использования

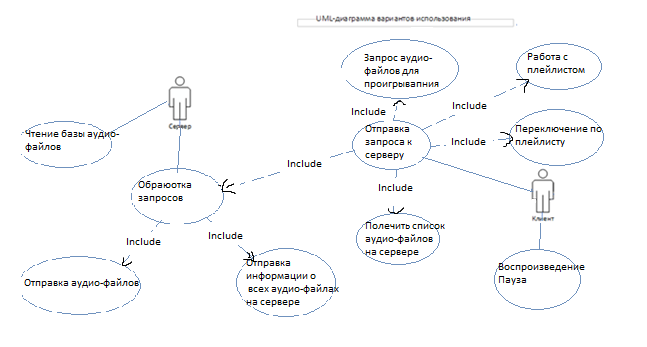


Рисунок19– UML-Диаграмма вариантов использования

Приложение В.2 – UML-диаграмма классов клиента

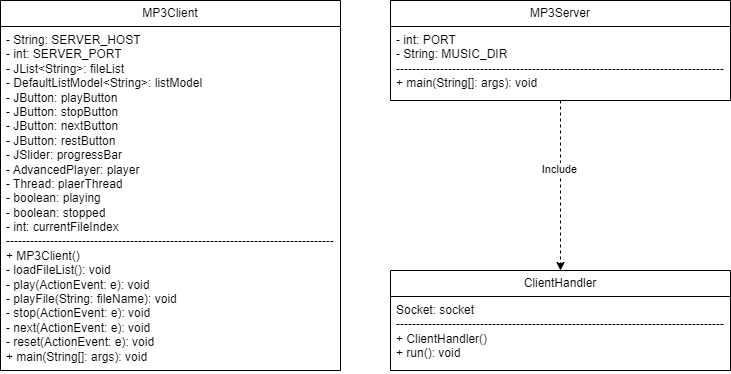


Рисунок20– UML-диаграмма классов клиента

Приложение В.3 – UML-диаграмма классов сервера

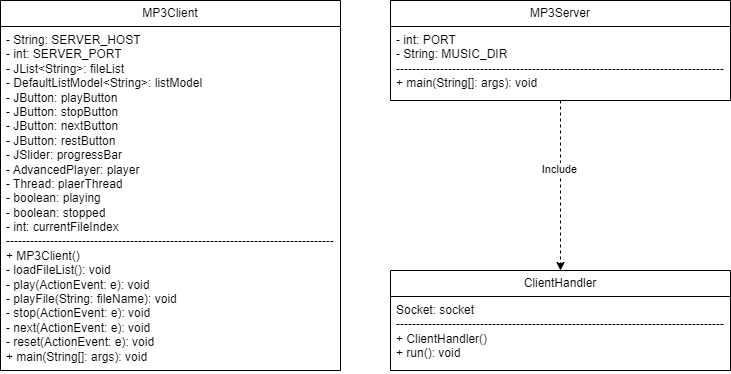


Рисунок22– UML-диаграмма классов сервера

Приложение В.4 – UML-диаграмма деятельности

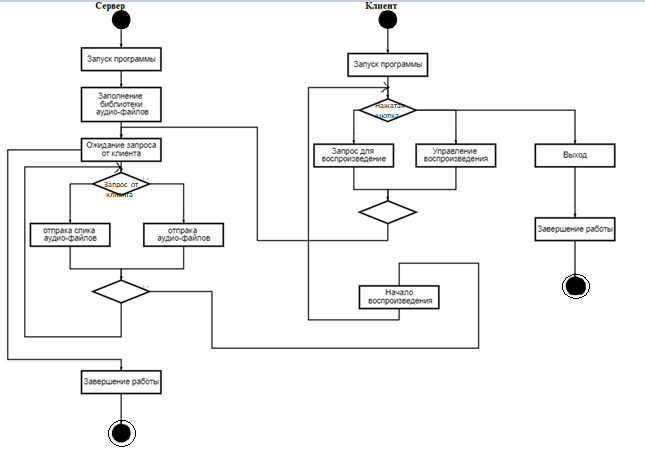


Рисунок 24– UML-диаграмма деятельности

Приложение В.5 – UML-диаграмма развёртывания

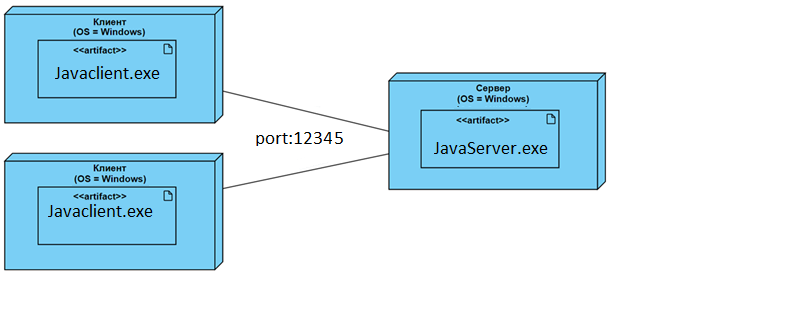


Рисунок 25 – UML-диаграмма развёртывания

Приложение В.6 – UML-диаграмма последовательности

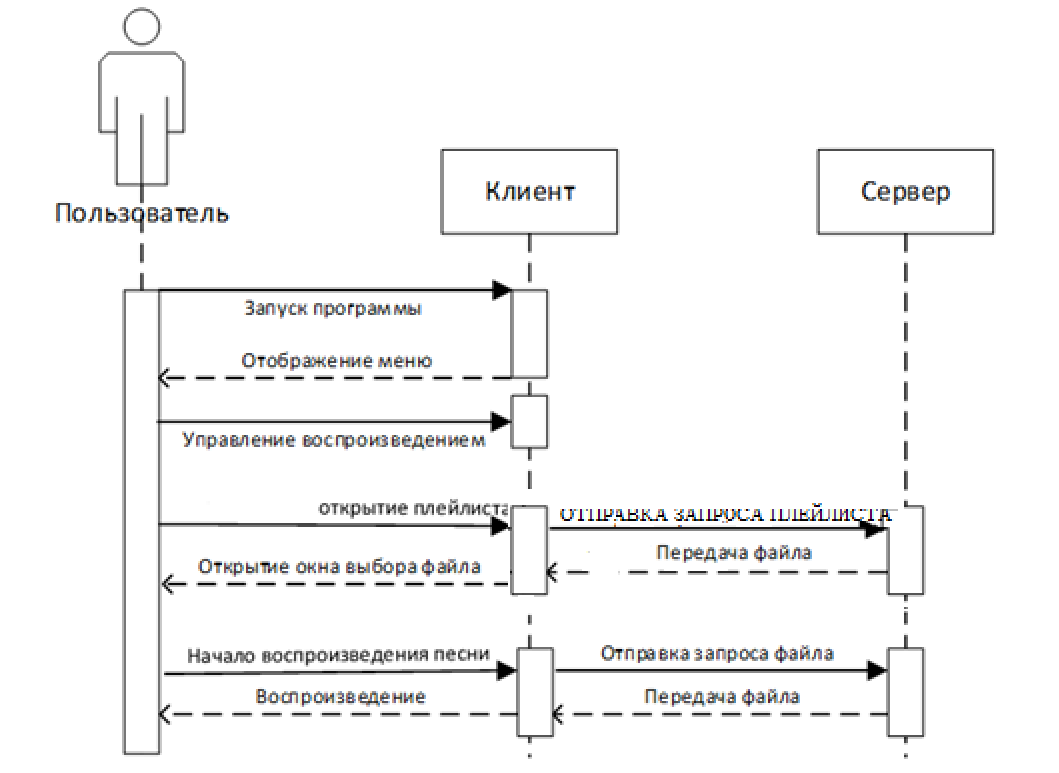


Рисунок 26– UML-диаграмма последовательности