# Содержание.

1.Задание………………………………………………………………….3

2.Введение.

## 2.1.Wave-файлы…………………………………………………………..3

# 2.1.1.Структура файла……………………………………………………3

2.1.2 Чтение wave—файла………………………………………………...5

2.1.3. Представление данных…………………………………………….6

3. Эффекты………………………………………………………………..7

3.1. Дилэй………………………………………………………………….7

3.2. Овердрайв…………………………………………………………….8

4. Эквалайзер………………………………………………………………9

5. Графический интерфэйс………………………………………………13

6. Принципиальная схема работа программы…………………………14

7. Многопоточность…………………………………………………… 16

8. Листинг программы……………………………………………………18

9. Список литературы…………………………………………………….37

# 1.Задание

1. Разработать программу проигрывателя \*.wav файлов на языке Java с поддержкой заданной цепи из двух эффектов(эффект 1 и эффект 2) и графическим N-полосовым эквалайзером. Эффекты(эффект 1- “Делей”, эффект 2 – “Овердрайв”), число полос эквалайзера(N = 7):
   1. Количество полос, N = 7шт ;
   2. Диапазон частот D = 1 Гц – 44,1КГц;
   3. Тип используемого фильтра – КИХ, Оконный.
   4. Метод проектирования, см. таблицу.
   5. Ослабление сигнала в полосе заграждения, As = -40дБ;
   6. Apass = 3 ±Δ дБ;
   7. Графический интерфейс.
   8. Открытие и воспроизведение \*.wav файлов.
   9. Регулирование громкости.
   10. Включение и отключение эффектов эквалайзера.
   11. Регулирование параметров эффектов(не менее 1 параметра).
   12. Совместная работа эффектов и эквалайзера.
2. **Введение.**

## **Wave-файлы.**

# Структура файла.

Структура файла, если говорить грубо, представляет собой две области: заголовок файла и сами данные. Заголовок расположен в начале файла и содержит метаданные.

*Таблица 1. Структура wav файла.*

|  |  |  |
| --- | --- | --- |
| Местоположение | Поле | Описание |
| 0..3 (4 байта) | **chunkId** | Содержит символы "RIFF" в ASCII кодировке |
| 4..7 (4 байта) | **chunkSize** |  |
| 8..11 (4 байта) | **format** | Содержит символы "WAVE" |
| 12..15 (4 байта) | **subchunk1Id** | Содержит символы "fmt " |
| 16..19 (4 байта) | **subchunk1Size** |  |
| 20..21 (2 байта) | **audioFormat** | Формат аудио |
| 22..23 (2 байта) | **numChannels** | Количество каналов. Моно = 1, Стерео = 2 |
| 24..27 (4 байта) | **sampleRate** | Частота дискретизации. 8000 Гц, 44100 Гц и т.д. |
| 28..31 (4 байта) | **byteRate** | Количество байт, передаваемых за секунду воспроизведения. |
| 32..33 (2 байта) | **blockAlign** | Количество байт для одной выборки, включая все каналы. |
| 34..35 (2 байта) | **bitsPerSample** | Количество бит в выборке. Так называемая "глубина" или точность звучания. 8 бит, 16 бит и т.д. |
| 36..39 (4 байта) | **subchunk2Id** | Содержит символы "data" |
| 40..43 (4 байта) | **subchunk2Size** | Количество байт в области данных. |
| 44.. | **data** | Непосредственно WAV-данные. |

Вот и весь заголовок, длина которого составляет 44 байта, далее следует блок данных.

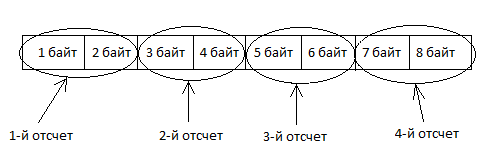
Рассмотрим более подробно данную таблицу. WAV-файл использует стандартную **RIFF**-структуру, которая группирует содержимое файла из отдельных секций (chunks) - формат выборок аудиоданных, аудиоданные, и т. п. Каждая секция имеет свой отдельный заголовок секции и отдельные данные секции. Заголовок секции указывает на тип секции и количество содержащихся в секции байт. Такой принцип организации позволяет программам анализировать только необходимые секции, пропуская остальные секции, которые не известны или которые не требуют обработки. Некоторые определенные секции могут иметь в своем составе подсекции (sub-chunks).

# Чтение wave-файла

Аудиовыборки многоканального цифрового аудио сохраняются как чередуемые данные, которые просто означают последовательные аудиовыборки нескольких каналов. Выборки каналов сохранены последовательно друг за другом, перед тем как произойдет переход к следующему времени выборки. Это сделано с целью возможности последовательного проигрывания файла даже тогда, когда еще не весь файл прочитан целиком. Это удобно, когда проигрывается большой файл с диска (который не может быть размещен целиком в памяти) или файл передается в последовательном потоке данных через сетевое соединение. На рисунке 3 показано как хранятся данные двухканального звука.

*Рисунок 1. Блок данных wav-файла*

При чтении файла, реализация, возможно, будет обеспечивать побайтное чтение в буфер. Нужно помнить, что элементы такого буфера не будут являться отсчетами, т.к. размер отсчета («битность») у разных файлов разная. В 16-битных wav-файлах на один отсчет одного канала приходится два расположенных друг за другом байта, в 24-битных – 3 байта.



*Рисунок 2. Расположение отсчетов в wav-файле*

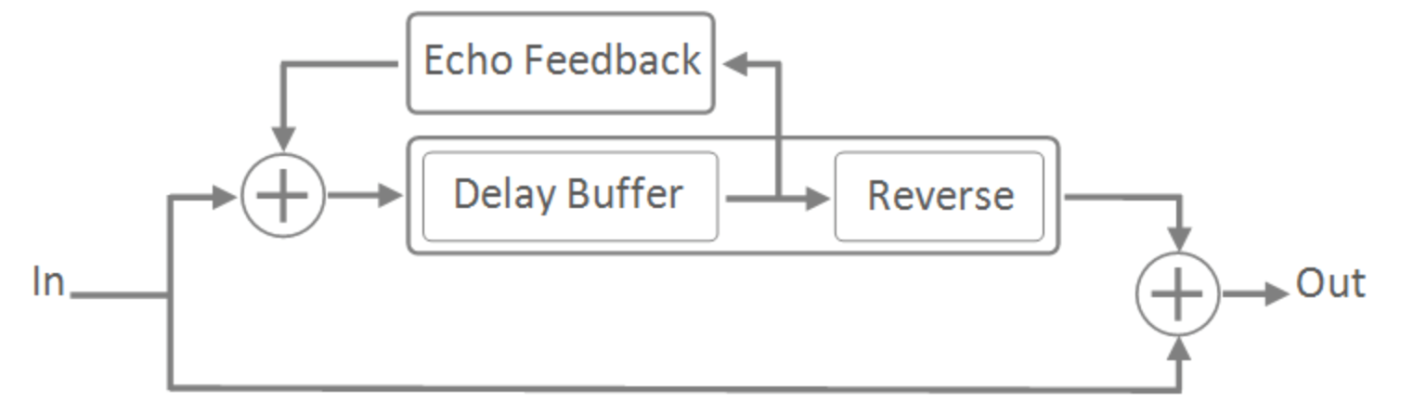
# Представление данных

Мы выяснили что на один отсчет может приходиться несколько байт. Рассмотрим вариант 16-битного файла. На один отсчет приходится 2 байта. Для обеспечения наиболее качественного звука было бы правильно реализовать структуру, содержащую все 16 бит и основные арифметические операции для нее. Поэтому в данной курсовой работе для облегчения и уменьшения объема работы будем использовать тип short int, размер которого как раз и равен 16 разрядам.

При преобразовании всего буфера, содержащего последовательность байтов, в буфер, содержащий отсчеты типа short int, а затем обратного преобразования никаких изменений в качестве звука при проигрывании не произойдет. Это значит что данный тип можно использовать для обработки, но следует внимательно следить за значениями буфера. Если вы попытаетесь умножить значение выборки на слишком большой коэффициент, результат может превысить 2 байта. На практике вы будете слышать помехи, щелчки и т.д. Эти явления допускаются, но для их минимизации следует проверять результат и при превышении 2-х байт уменьшать либо вообще не умножать отсчет.

1. **Эффекты.**
   1. **Дилэй.**

Дилэй ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *delay*) или эхо ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *echo*) — [звуковой эффект](https://ru.wikipedia.org/wiki/%D0%97%D0%B2%D1%83%D0%BA%D0%BE%D0%B2%D0%BE%D0%B9_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82) или соответствующее [устройство](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D0%B4%D0%B0%D0%BB%D1%8C_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82%D0%BE%D0%B2), имитирующее чёткие затухающие повторы ([эхо](https://ru.wikipedia.org/wiki/%D0%AD%D1%85%D0%BE)) исходного сигнала. Эффект реализуется добавлением к исходному сигналу его копии или нескольких копий, задержанных по времени. Под дилэем обычно подразумевается однократная задержка сигнала, в то время как эффект «эхо» — многократные повторы. По принципу действия схож с [ревербератором](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B2%D0%B5%D1%80%D0%B1%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F) и отличается от него лишь временем задержки, которое должно быть не менее 50-60 мс.

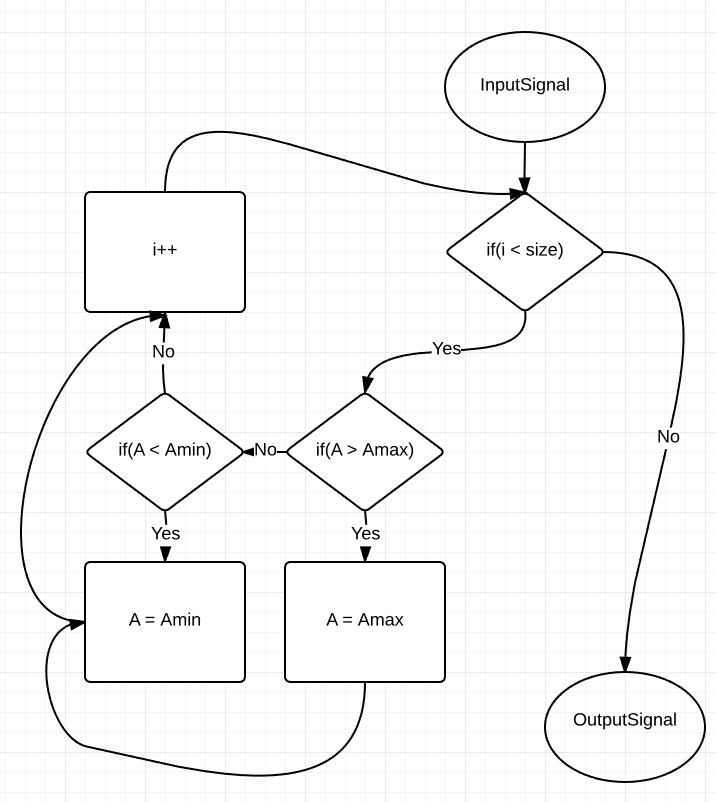


*Рис.3 Принцип вычисления дилэя.*

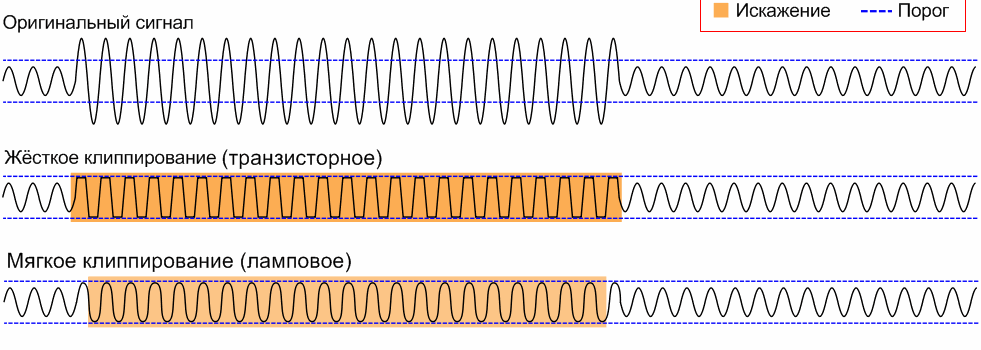
**3.2 Овердрайв**

Принцип действия.

Если амплитуда входного сигнала больше максимальной или минимальной заданной амплитуды, то входной сигнал обрезается до уровня максимальной и минимальной из заданных амплитуд соответственно.



*Рис. 4 Блок-схема обсчета эффекта “Овердрайв”.*



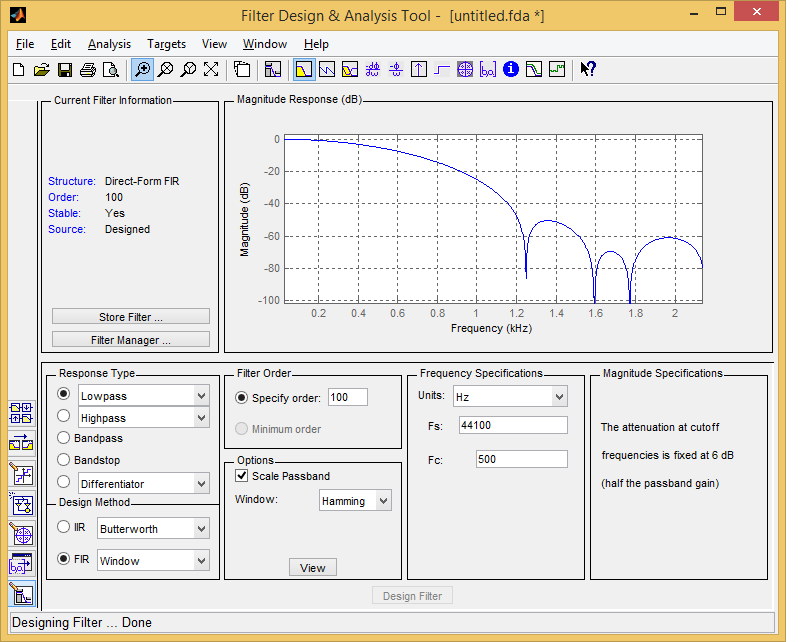
*Рис. 5 Способы обрезания амплитуды.*

1. **Эквалайзер.**

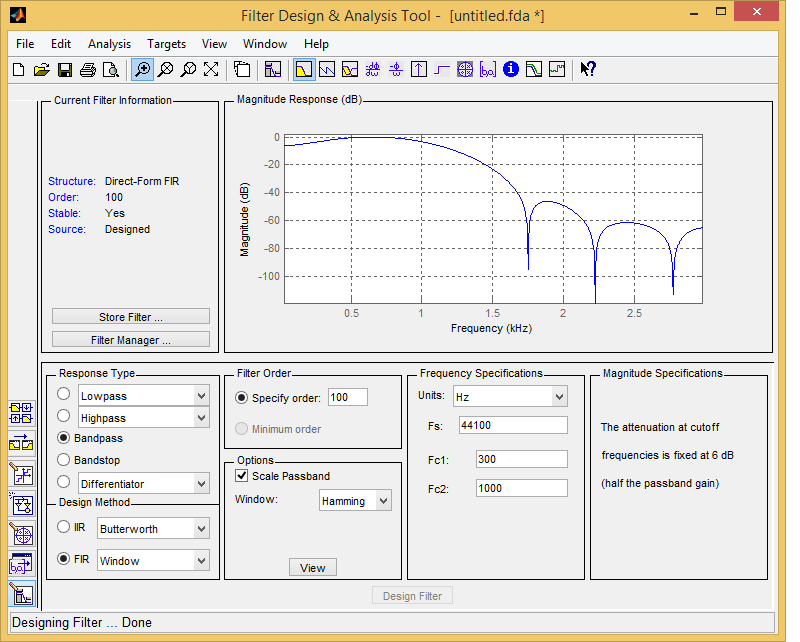
Порядок фильтров выбран так, чтобы обеспечить спад амплитуды на Fs не менее -40 дб на декаду.

|  |  |  |
| --- | --- | --- |
| **№ полосы** | **Частоты (Гц)** | **Тип фильтра** |
| 1 | < 400 | ФНЧ |
| 2 | 300-1000 | ПФ |
| 3 | 900-2400 | ПФ |
| 4 | 2200-5100 | ПФ |
| 5 | 4500-6500 | ПФ |
| 6 | 6000-11500 | ПФ |
| 7 | 11000-11500 | ПФ |

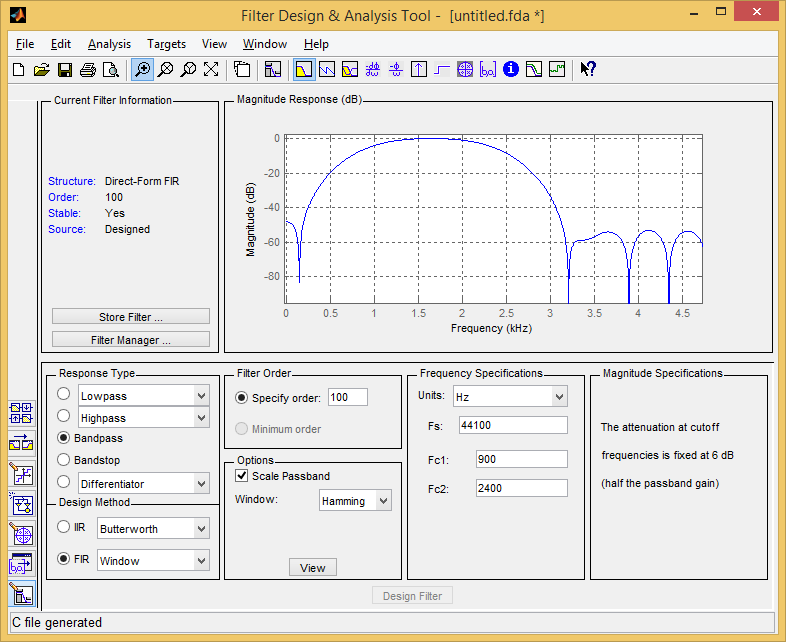
Таблица 1. Разбиение на полосы.



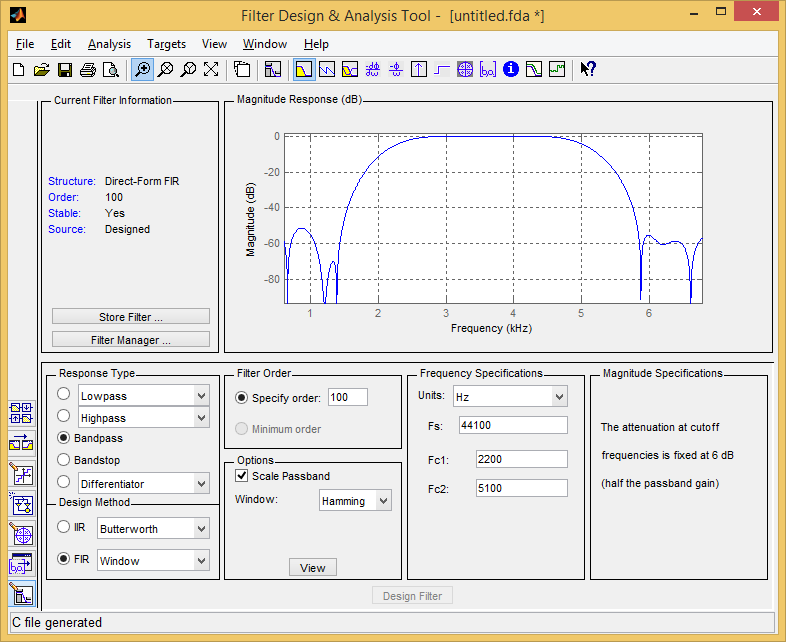
*Рис. 6 Проектирование ФНЧ.*



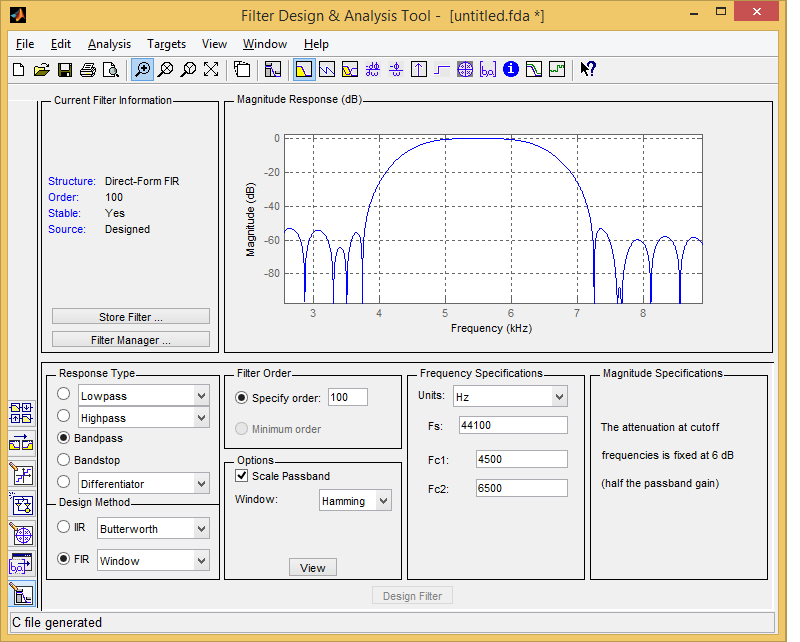
*Рис. 7 Проектирование полосового фильтра ПФ №1*



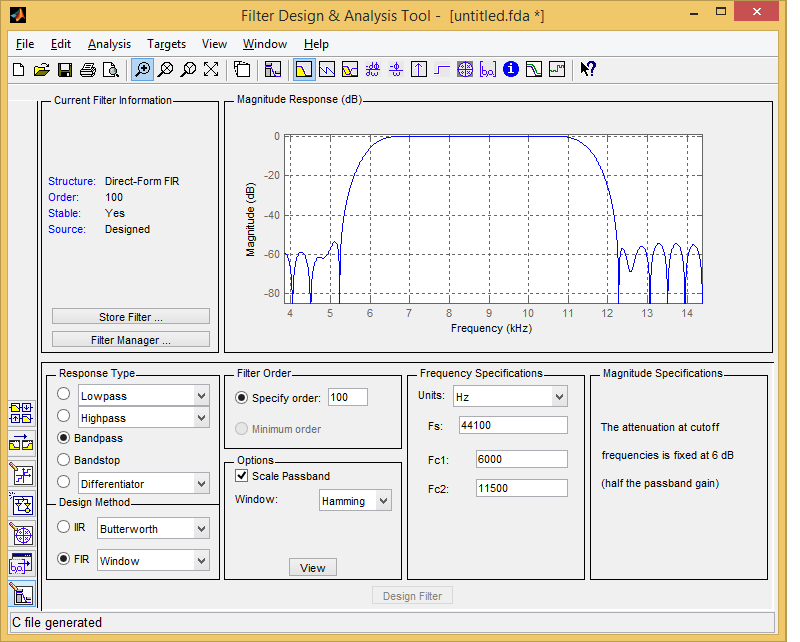
*Рис. 8 Проектирование полосового фильтра ПФ №2*



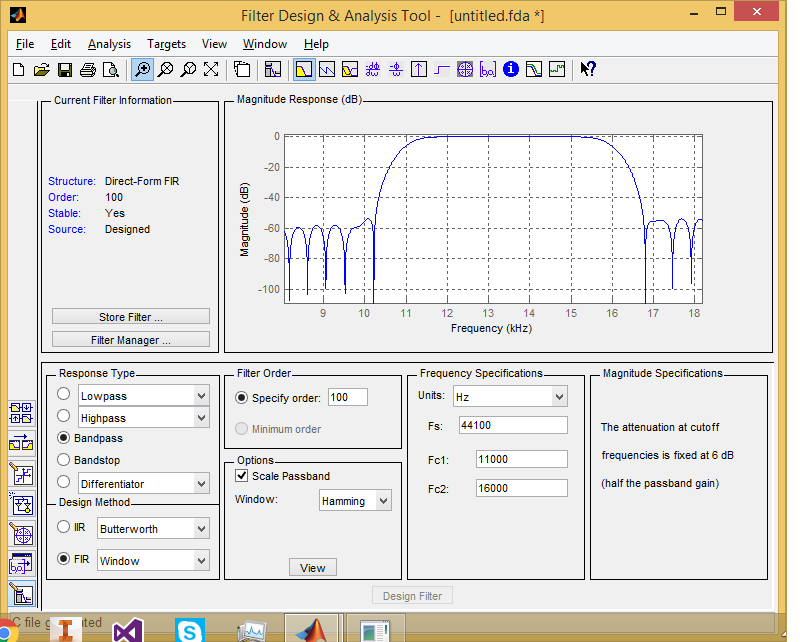
*Рис. 9 Проектирование полосового фильтра ПФ №3*



*Рис. 10 Проектирование полосового фильтра ПФ №4*



*Рис. 11 Проектирование полосового фильтра ПФ №5*



*Рис. 12 Проектирование ПФ № 6.*

1. **Графический интерфейс.**

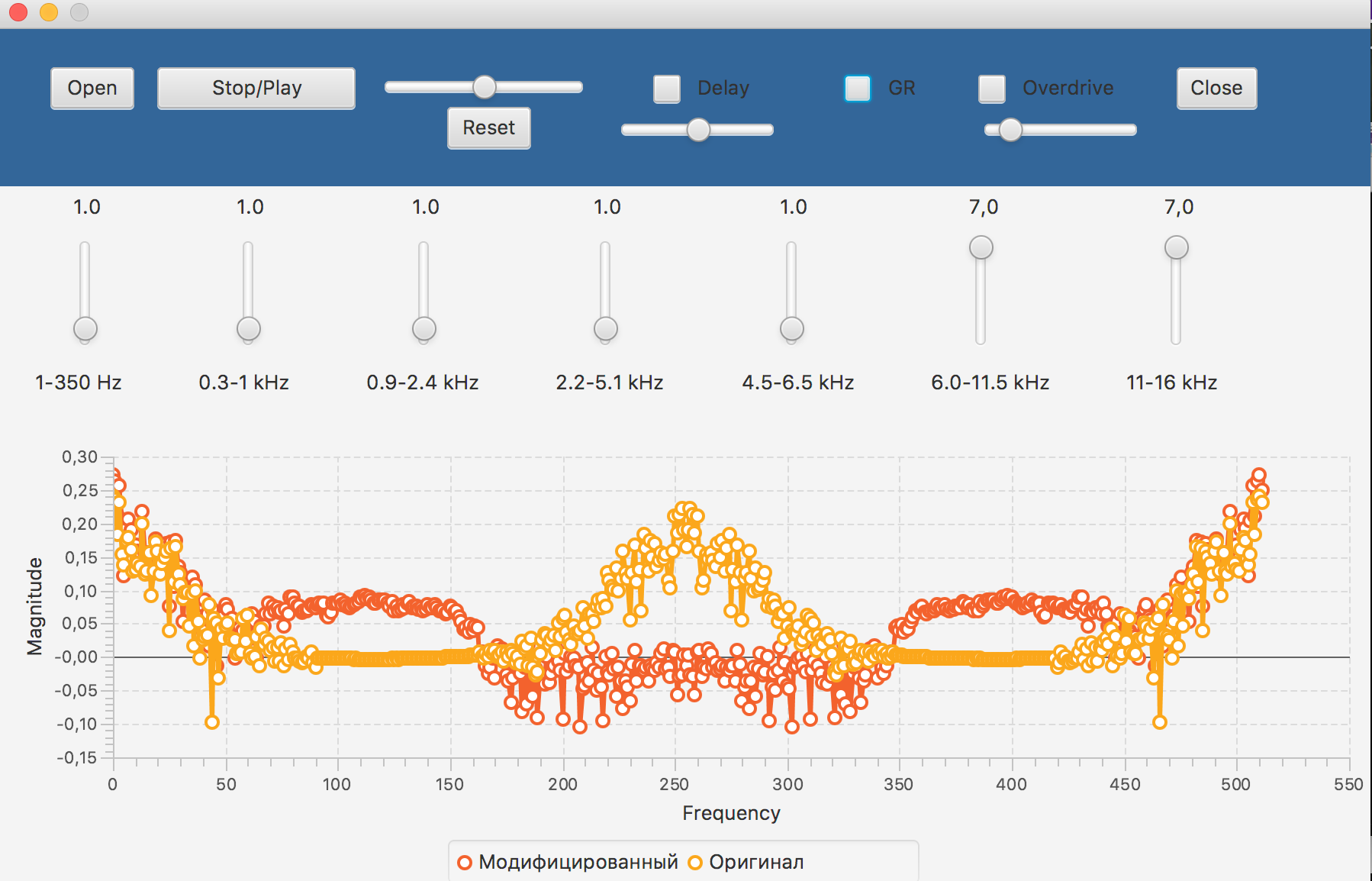
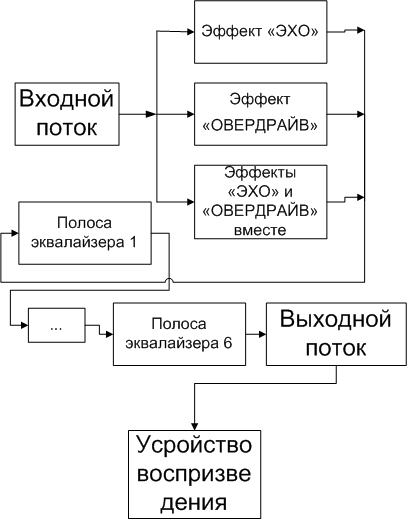


Рис. 13 Графический интерфейс.

1. **Принципиальная схема работы программы.**



*Рисунок 14. Схема обработки аудио файла*

../../../Downloads/PROGRAM.pdf

Рис. 15 Распределение данных и задач между потоками.

1. **Многопоточность.**

Т.к. не все машины обладают достаточно мощным процессором для того, чтобы вытянуть все вычисления производимые в данной программе в одном потоке, было решено разбить вычисления производимые в классе эквалайзера на потоки. Количество потоков равно количеству логических ядер ЦП. Если количество ядер ЦП будет равно одному, то программа разобьет вычисления на два потока.

Листинг данного класса Equalizer.java

**package** ru.bmstu.www.equalizer;

**import** java.util.concurrent.ExecutionException;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.Future;

**import** ru.bmstu.www.filter.Filter;

**import** ru.bmstu.www.filter.coefs.FilterInfo;

**public** **class** **Equalizer** {

**private** **short**[] inputSignal;

**private** **short**[]outputSignal;

**private** **Filter**[] filters;

**private** **final** **static** **int** ***COUNT\_OF\_BANDS*** = 7;

**private** **static** **int** *COUNT\_OF\_THREADS*;

**private** **final** **int** lenghtOfInputSignal;

**private** ExecutorService pool;

**public** **Equalizer**(**final** **int** lenghtOfInputSignal) {

pool = **Executors**.*newFixedThreadPool*(*COUNT\_OF\_THREADS*);

**this**.lenghtOfInputSignal = lenghtOfInputSignal;

**this**.createFilters();

*COUNT\_OF\_THREADS* = **Runtime**.*getRuntime*().availableProcessors();

**if**(**Equalizer**.*COUNT\_OF\_THREADS* == 1)

**Equalizer**.*COUNT\_OF\_THREADS* = 2;

}

**public** **void** **setInputSignal**(**short**[] inputSignal) {

**this**.inputSignal = inputSignal;

**this**.outputSignal = **new** **short**[**this**.lenghtOfInputSignal];

**this**.filters[0].settings(**FilterInfo**.***CoefsOfBand\_0***,

**FilterInfo**.***COUNT\_OF\_COEFS***, **this**.inputSignal);

**this**.filters[1].settings(**FilterInfo**.***CoefsOfBand\_1***,

**FilterInfo**.***COUNT\_OF\_COEFS***, **this**.inputSignal);

**this**.filters[2].settings(**FilterInfo**.***CoefsOfBand\_2***,

**FilterInfo**.***COUNT\_OF\_COEFS***, **this**.inputSignal);

**this**.filters[3].settings(**FilterInfo**.***CoefsOfBand\_3***,

**FilterInfo**.***COUNT\_OF\_COEFS***, **this**.inputSignal);

**this**.filters[4].settings(**FilterInfo**.***CoefsOfBand\_4***,

**FilterInfo**.***COUNT\_OF\_COEFS***, **this**.inputSignal);

**this**.filters[5].settings(**FilterInfo**.***CoefsOfBand\_5***,

**FilterInfo**.***COUNT\_OF\_COEFS***, **this**.inputSignal);

**this**.filters[6].settings(**FilterInfo**.***CoefsOfBand\_6***,

**FilterInfo**.***COUNT\_OF\_COEFS***, **this**.inputSignal);

}

**private** **void** **createFilters**() {

**this**.filters = **new** **Filter** [**Equalizer**.***COUNT\_OF\_BANDS***] ;

**this**.filters[0] = **new** Filter(**this**.lenghtOfInputSignal);

**this**.filters[1] = **new** Filter(**this**.lenghtOfInputSignal);

**this**.filters[2] = **new** Filter(**this**.lenghtOfInputSignal);

**this**.filters[3] = **new** Filter(**this**.lenghtOfInputSignal);

**this**.filters[4] = **new** Filter(**this**.lenghtOfInputSignal);

**this**.filters[5] = **new** Filter(**this**.lenghtOfInputSignal);

**this**.filters[6] = **new** Filter(**this**.lenghtOfInputSignal);

}

**public** **void** **equalization**( ) **throws** **InterruptedException**, **ExecutionException** {

Future<**short**[]>[] **fs** = **new** Future[**Equalizer**.***COUNT\_OF\_BANDS***];

**for**(**int** **i** = 0; i < **Equalizer**.***COUNT\_OF\_BANDS***; i++) {

fs[i] = pool.submit(**this**.filters[i]);

}

**for**(**int** **i** = 0; i < **this**.outputSignal.length; i++) {

**this**.outputSignal[i] += fs[0].get()[i] +

fs[1].get()[i] +

fs[2].get()[i] +

fs[3].get()[i] +

fs[4].get()[i] +

fs[5].get()[i] +

fs[6].get()[i] ;

}

}

**public** **Filter** **getFilter**(**short** nF) {

**return** **this**.filters[nF];

}

**public** **short**[] **getOutputSignal**() {

**try** {

**Thread**.*sleep*(0);

} **catch** (**InterruptedException** **e**) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** **this**.outputSignal;

}

**public** **void** **close**() {

**if**(**this**.pool != **null**) {

**this**.pool.shutdown();

}

}

}

1. **Листинг программы.**

Программа написана на языке Java и является полностью портируемым программным продуктом на все машины, где установлена JRE не ниже версии 1.8.0. При написании программы использовались следующие библиотеки: JavaFX 2.0, java.sound.\* . При написании программы использовался

паттерн проектирования MVC, что делает программу легко модифицируемой при ее дальнейшем развитии.

JavaFX – современная графическая библиотека, поддерживающая OpenGL и DirectX, что позволяет избавиться от присущих Java тормозов при отрисовке интерфейса(UI) на машинах, которые имеют в своем составе GPU, поддерживающие OpenGL или DirectX. Данная библиотека также позволяет снять нагрузку с CPU по отрисовке UI, если выполнено условие, обговоренное выше.

Пакеты:

Пакет ru.bmstu.www.player содержит в своем составе класс AudioPlayer.java, который непосредственно занимается обработкой данных, запуском потока воспроизведения.

**package** ru.bmstu.www.player;

**import** java.io.File;

**import** java.io.IOException;

**import** java.nio.ByteBuffer;

**import** javax.sound.sampled.AudioFormat;

**import** javax.sound.sampled.AudioInputStream;

**import** javax.sound.sampled.LineEvent;

**import** javax.sound.sampled.LineListener;

**import** javax.sound.sampled.LineUnavailableException;

**import** javax.sound.sampled.SourceDataLine;

**import** javax.sound.sampled.UnsupportedAudioFileException;

**import** ru.bmstu.www.effects.Delay;

**import** ru.bmstu.www.effects.Overdrive;

**import** ru.bmstu.www.equalizer.Equalizer;

**import** ru.bmstu.www.filter.coefs.FilterInfo;

**import** ru.bmstu.www.input.AudioFileFormat;

**import** ru.bmstu.www.input.ReadMusicFile;

**public** **class** **AudioPlayer** **implements** LineListener{

**private** **double** volume;

**private** SourceDataLine sourceDataLine;

**private** **AudioInputStream** ais;

**private** **byte**[] buff;

**private** **final** **int** BUFF\_SIZE = 65536 \* 2;

**private** **short**[] sampleBuff;

**private** **Delay** delay;

**private** **boolean** isDelay;

**private** **Overdrive** overdrive;

**private** **double** overdriveCoef;

**private** **boolean** isOverdrive;

**private** **Equalizer** equalizer;

**private** **boolean** pause;

**private** **AudioFormat** format;

**public** **AudioPlayer**(**File** musicFile) **throws** **UnsupportedAudioFileException**,

**IOException**, **InterruptedException**, **LineUnavailableException** {

**ReadMusicFile** **readFile** = **new** ReadMusicFile(musicFile);

**this**.sourceDataLine = readFile.getSourceDataLine();

**this**.ais = readFile.getAudioInputStream();

**this**.sampleBuff = **new** **short**[BUFF\_SIZE / 2];

**this**.delay = **new** Delay();

**this**.overdrive = **new** Overdrive();

**this**.isDelay = **false**;

**this**.isOverdrive = **false**;

**this**.overdriveCoef = 1.0;

**this**.equalizer = **new** Equalizer();

**AudioFileFormat** **aff** = **new** AudioFileFormat();

format = **new** AudioFormat((**float**)aff.getSampleRate(),

aff.getBits(), aff.getChannels(),

aff.isSigned(), aff.isBigEndian());

**this**.volume = 1.0;

}

**public** **void** **play**() {

**try**{

**this**.sourceDataLine.open(**this**.format);

**this**.sourceDataLine.start();

**int** **bytesRead** = -1;

**this**.buff = **new** **byte**[**this**.BUFF\_SIZE +

ru.bmstu.www.filter.coefs.**FilterInfo**.***COUNT\_OF\_COEFS***];

**this**.pause = **false**;

**while** ((bytesRead = ais.read(**this**.buff, 0, **this**.BUFF\_SIZE)) != -1) {

**this**.ByteArrayToSamplesArray();

**if**(**this**.pause) {**this**.stop();}

**if**(**this**.isDelay)

**this**.delay(**this**.sampleBuff);

**if**(**this**.isOverdrive) {

**this**.overdrive(sampleBuff);

}

equalizer.setInputSignal(**this**.sampleBuff, **FilterInfo**.***COUNT\_OF\_COEFS***);

**this**.equalizer.equalization();

**this**.sampleBuff = equalizer.getOutputSignal();

**this**.SampleArrayByteArray();

sourceDataLine.write(**this**.buff, 0, **this**.buff.length -3 );

} **System**.***out***.println("END");

**this**.sourceDataLine.drain();

**this**.sourceDataLine.close();

}**catch** (**Exception** **e**) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**private** **void** **delay**(**short**[] inputSamples) {

**this**.delay.setInputSampleStream(inputSamples);

**this**.delay.createEffect();

}

**public** **boolean** **delayIsActive**() {

**return** **this**.isDelay;

}

**public** **void** **setDelay**(**boolean** b) {

**this**.isDelay = b;

}

**private** **void** **overdrive**(**short**[] inputSamples) {

**this**.overdrive.setOverdriveCoef(**this**.overdriveCoef);

**this**.overdrive.setInputSampleStream(inputSamples);

**this**.overdrive.createEffect();

}

**public** **boolean** **overdriveIsActive**() {

**return** **this**.isOverdrive;

}

**public** **void** **setOverdrive**(**boolean** b) {

**this**.isOverdrive = b;

}

**public** **void** **setOverdriveCoef**(**double** c) {

**this**.overdriveCoef = c;

}

**private** **void** **stop**() {

**if**(pause) {

**for**(;;) {

**try** {

**if**(!pause) **break**;

**Thread**.*sleep*(100);

} **catch** (**InterruptedException** **e**) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**public** **void** **setPause**(**boolean** pause) {

**this**.pause = pause;

}

**public** **boolean** **getPause**() {

**return** **this**.pause;

}

**public** **void** **setVolume**(**double** volume) {

**this**.volume = volume;

}

**public** **double** **getVolume**() {

**return** **this**.volume;

}

***@Override***

**public** **void** **update**(**LineEvent** event) {

**LineEvent**.**Type** **type** = event.getType();

}

**public** **short**[] **getSamples**() {

**return** **this**.sampleBuff;

}

**private** **void** **ByteArrayToSamplesArray**() {

**for**(**int** **i** = 0, **j** = 0; i < **this**.BUFF\_SIZE; i += 2 , j++) {

**this**.sampleBuff[j] = (**short**) (0.6 \* (**ByteBuffer**.*wrap*(**this**.buff, i, 2).order(

java.nio.**ByteOrder**.***LITTLE\_ENDIAN***).getShort()) \* **this**.getVolume());

}

}

**private** **void** **SampleArrayByteArray**() {

**for**(**int** **i** = 0, **j** = 0; i < **this**.sampleBuff.length && j < (**this**.BUFF\_SIZE +

ru.bmstu.www.filter.coefs.**FilterInfo**.***COUNT\_OF\_COEFS*** - 1 ); i++, j += 2 ) {

**this**.buff[j] = (**byte**)(**this**.sampleBuff[i]);

**this**.buff[j + 1] = (**byte**)(**this**.sampleBuff[i] >>> 8);

}

}

**public** **Equalizer** **getEqualizer**() {

**return** **this**.equalizer;

}

**public** **void** **close**() {

**if**(**this**.ais != **null**)

**try** {

**this**.ais.close();

} **catch** (**IOException** **e**) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**if**(**this**.sourceDataLine != **null**)

**this**.sourceDataLine.close();

}

}

}

}

Пакет ru.bmstu.www.view содержит точку входа – класс EqualizerAPP.java, контроллер UI – UserUnterfaceController.java и саму верстку UI – UserInterface.fxml

Класс EqualizerAPP.java

**package** ru.bmstu.www.view;

**import** java.io.IOException;

**import** javafx.application.Application;

**import** javafx.fxml.FXMLLoader;

**import** javafx.scene.Parent;

**import** javafx.scene.Scene;

**import** javafx.stage.Stage;

/\*\*

\*

\* **@author** admin

\*/

**public** **class** **EqualizerApp** **extends** **Application** {

***@Override***

**public** **void** **start**(**Stage** stage) **throws** **IOException** {

**Parent** **root** = **FXMLLoader**.*load*(getClass().getResource("UserInterface.fxml"));

**Scene** **scene** = **new** Scene(root);

stage.setScene(scene);

stage.setWidth(870);

stage.setHeight(270);

stage.setResizable(**false**);

stage.show();

}

/\*\*

\* **@param** args the command line arguments

\*/

**public** **static** **void** **main**(**String**[] args) {

*launch*(args);

}

}

Класс UserInterface.java

**package** ru.bmstu.www.view;

**import** java.io.File;

**import** java.io.IOException;

**import** java.net.URL;

**import** java.util.ResourceBundle;

**import** javax.sound.sampled.LineUnavailableException;

**import** javax.sound.sampled.UnsupportedAudioFileException;

**import** javafx.beans.value.ChangeListener;

**import** javafx.beans.value.ObservableValue;

**import** javafx.collections.FXCollections;

**import** javafx.collections.ObservableList;

**import** javafx.fxml.FXML;

**import** javafx.fxml.Initializable;

**import** javafx.scene.chart.LineChart;

**import** javafx.scene.chart.XYChart;

**import** javafx.scene.control.CheckBox;

**import** javafx.scene.control.Label;

**import** javafx.scene.control.Slider;

**import** javafx.scene.media.AudioSpectrumListener;

**import** javafx.scene.media.MediaPlayer;

**import** javafx.stage.FileChooser;

**import** javafx.stage.FileChooser.ExtensionFilter;

**import** javafx.stage.Stage;

**import** ru.bmstu.www.player.AudioPlayer;;

/\*\*

\* FXML Controller class

\*

\* **@author** admin

\*/

**public** **class** **UserInterfaceController** **implements** Initializable {

***@FXML***

**private** **Label** labelForSlider\_0, labelForSlider\_1, labelForSlider\_2, labelForSlider\_3,

labelForSlider\_4,labelForSlider\_5, labelForSlider\_6, timeLabel;

***@FXML***

**private** **Slider** Slider\_0, Slider\_1, Slider\_2, Slider\_3, Slider\_4, Slider\_5,

Slider\_6,soundSlider, timeSlider, overdriveSlider;

***@FXML***

**private** **LineChart** graph;

***@FXML***

**CheckBox** overdriveCheck, delayCheck;

**private** **MediaPlayer** player;

**private** **XYChart**.**Data**[] series1Data;

**private** **XYChart**.**Data**[] series2Data;

**private** **AudioPlayer** audioPlayer;

**private** **Thread** playThread;

/\*\*

\* Initializes the controller class.

\* **@param** url

\* **@param** rb

\*/

***@Override***

**public** **void** **initialize**(**URL** url, **ResourceBundle** rb) {

**this**.labelInitialize();

**XYChart**.**Series**<Integer,Number> **series1** = **new** **XYChart**.Series<>();

**XYChart**.**Series**<Integer,Number> **series2** = **new** **XYChart**.Series<>();

series1.setName("Модифицированный");

series2.setName("Оригинал");

series1Data = **new** **XYChart**.**Data**[128];

series2Data = **new** **XYChart**.**Data**[128];

**for** (**int** **i**=0; i<series1Data.length; i++) {

series1Data[i] = **new** **XYChart**.Data<>(i, 50);

series1.getData().add(series1Data[i]);

series2Data[i] = **new** **XYChart**.Data<>(i, 50);

series2.getData().add(series2Data[i]);

}

ObservableList<**XYChart**.**Series**<Integer, Number>> **lineChartData** = **FXCollections**.*observableArrayList*();

lineChartData.add(series1);

lineChartData.add(series2);

graph.setData(lineChartData);

graph.createSymbolsProperty();

graph.setAnimated(**false**);

**this**.checkBoxInnitial();

**this**.volumeFromSlider();

}

***@FXML***

**private** **void** **clickOpen**() **throws** **UnsupportedAudioFileException**, **IOException**, **InterruptedException**, **LineUnavailableException** {

**FileChooser** **fileChooser** = **new** FileChooser();

fileChooser.setTitle("Open Resource File");

fileChooser.getExtensionFilters().addAll(

**new** ExtensionFilter("Audio Files", "\*.wav"));

**File** **selectedFile** = fileChooser.showOpenDialog(**new** Stage());

**if**(selectedFile == **null**) **return**;

**this**.audioPlayer = **new** AudioPlayer(selectedFile);

playThread = **new** Thread(()->{

**this**.audioPlayer.play();

});

playThread.start();

**System**.***out***.println("PLAY");

}

***@FXML***

**private** **void** **clickStopPlay**() {

**if**(**this**.audioPlayer != **null**) {

**if**(!**this**.audioPlayer.getPause())

**this**.audioPlayer.setPause(**true**);

**else** **this**.audioPlayer.setPause(**false**);

}

}

***@FXML***

**private** **void** **clickReset**() {

**if** (**this**.audioPlayer == **null**) **return**;

Slider\_0.setValue(1.0);

Slider\_1.setValue(1.0);

Slider\_2.setValue(1.0);

Slider\_3.setValue(1.0);

Slider\_4.setValue(1.0);

Slider\_5.setValue(1.0);

Slider\_6.setValue(1.0);

soundSlider.setValue(1.0);

**this**.overdriveSlider.setValue(1.0);

}

// Listen for Slider value changes

**private** **void** **labelInitialize**() {

Slider\_0.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

**String** **str** = **String**.*format*("%.1f", (newValue.doubleValue()));

labelForSlider\_0.setText(str);

audioPlayer.getEqualizer().getFilter((**short**)0).setGain((**float**)newValue.doubleValue());

}

});

Slider\_1.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

**String** **str** = **String**.*format*("%.1f", (newValue.doubleValue()));

labelForSlider\_1.setText(str);

audioPlayer.getEqualizer().getFilter((**short**)1).setGain((**float**)newValue.doubleValue());

}

});

Slider\_2.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

**String** **str** = **String**.*format*("%.1f", (newValue.doubleValue()));

labelForSlider\_2.setText(str);

audioPlayer.getEqualizer().getFilter((**short**)2).setGain((**float**)newValue.doubleValue());

}

});

Slider\_3.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

**String** **str** = **String**.*format*("%.1f", (newValue.doubleValue()));

labelForSlider\_3.setText(str);

audioPlayer.getEqualizer().getFilter((**short**)3).setGain((**float**)newValue.doubleValue());

}

});

Slider\_4.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

**String** **str** = **String**.*format*("%.1f", (newValue.doubleValue()));

labelForSlider\_4.setText(str);

audioPlayer.getEqualizer().getFilter((**short**)4).setGain((**float**)newValue.doubleValue());

}

});

Slider\_5.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

**String** **str** = **String**.*format*("%.1f", (newValue.doubleValue()));

labelForSlider\_5.setText(str);

audioPlayer.getEqualizer().getFilter((**short**)5).setGain((**float**)newValue.doubleValue());

}

});

Slider\_6.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

**String** **str** = **String**.*format*("%.1f", (newValue.doubleValue()));

labelForSlider\_6.setText(str);

audioPlayer.getEqualizer().getFilter((**short**)6).setGain((**float**)newValue.doubleValue());

}

});

overdriveSlider.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

**String** **str** = **String**.*format*("%.1f", (newValue.doubleValue()));

audioPlayer.setOverdriveCoef(newValue.doubleValue());

}

});

}

**private** **void** **checkBoxInnitial**() {

**this**.delayCheck = **new** CheckBox();

**this**.delayCheck.selectedProperty().addListener(**new** ChangeListener<Boolean>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Boolean**> observable, **Boolean** oldValue, **Boolean** newValue) {

**System**.***out***.println("njj");

}

});

}

**private** **void** **volumeFromSlider**() {

soundSlider.valueProperty().addListener(**new** ChangeListener<Number>() {

***@Override***

**public** **void** **changed**(ObservableValue<? **extends** **Number**> observable,

**Number** oldValue, **Number** newValue) {

audioPlayer.setVolume(newValue.doubleValue());

}

});

}

**private** **void** **updateLineChartAndSoundSlider**() {

player.setAudioSpectrumListener(**new** AudioSpectrumListener() {

***@Override***

**public** **void** **spectrumDataUpdate**(**double** timestamp, **double** duration,

**float**[] magnitudes, **float**[] phases) {

**if**(player.getStatus() == **MediaPlayer**.*Status*.***PLAYING*** ) {

**for** (**int** **i** = 0; i < series1Data.length; i++) {

series1Data[i].setYValue(**Math**.*abs*( magnitudes[i]) /\*+ 60\*/);

}

timeSlider.setValue((player.getCurrentTime().toSeconds()) /

(player.getTotalDuration().toSeconds()));

timeLabel.setText(**String**.*format*("%02d.%d/%02d.%d",

(**int**)((player.getTotalDuration().toSeconds() % 3600) / 60),

(**int**)(player.getTotalDuration().toSeconds() % 60),

(**int**)((player.getCurrentTime().toSeconds() % 3600) / 60),

(**int**)(player.getCurrentTime().toSeconds() % 60)));

}

}

});

}

***@FXML***

**private** **void** **createDelay**() {

**System**.***out***.println("Delay");

**if**(!**this**.audioPlayer.delayIsActive())

**this**.audioPlayer.setDelay(**true**);

**else** **this**.audioPlayer.setDelay(**false**);

}

***@FXML***

**private** **void** **createOverdrive**() {

**System**.***out***.println("Overdrive");

**if**(!**this**.audioPlayer.overdriveIsActive())

**this**.audioPlayer.setOverdrive(**true**);

**else** **this**.audioPlayer.setOverdrive(**false**);

}

***@FXML***

**private** **void** **clickClose**() {

**if**(**this**.audioPlayer != **null**) {

**if**(**this**.playThread != **null**)

**this**.playThread.interrupt();

**this**.audioPlayer.getEqualizer().close();

**this**.audioPlayer.close();

}

**System**.*exit*(0);

}

}

Верстка UserInterface.fxml

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

<?**import** java.lang.\*?>

<?**import** java.util.\*?>

<?**import** javafx.scene.\*?>

<?**import** javafx.scene.control.\*?>

<?**import** javafx.scene.layout.\*?>

<?**import** javafx.scene.chart.LineChart?>

<?**import** javafx.scene.chart.NumberAxis?>

<**BorderPane** xmlns:fx=*"http://javafx.com/fxml/1"*

fx:id=*"root"* prefHeight=*"550.0"* prefWidth=*"650.0"*

fx:controller=*"ru.bmstu.www.view.UserInterfaceController"*>

<**top**>

<**HBox** spacing=*"15"* style=*"-fx-background-color: #336699;"* >

<**padding**>

<**javafx.geometry.Insets** top=*"25"* right=*"25"* bottom=*"25"* left=*"35"*/>

</**padding**>

<**Button** fx:id=*"buttonOpen"* text=*"Open"* minWidth=*"55"*

onAction=*"#clickOpen"* >

</**Button**>

<**Button** fx:id=*"buttonStopPlay"* text=*"Stop/Play"* minWidth=*"130"*

onAction=*"#clickStopPlay"* >

</**Button**>

<**VBox** spacing=*"5"* >

<**padding**>

<**javafx.geometry.Insets** top=*"5"* />

</**padding**>

<**Slider** fx:id=*"soundSlider"*

max=*"1.5"* min=*"0"* value=*"1.0"* rotate=*"0"*>

</**Slider**>

<**HBox** >

<**padding**>

<**javafx.geometry.Insets** left=*"45"*/>

</**padding**>

<**Button** fx:id=*"buttonReset"* text=*"Reset"* minWidth=*"55"*

onAction=*"#clickReset"* >

</**Button**>

</**HBox**>

</**VBox**>

<**HBox** spacing=*"5"* >

<**padding**>

<**javafx.geometry.Insets** top=*"5"* left=*"25"*/>

</**padding**>

<**CheckBox** fx:id=*"delayCheck"* onAction=*"#createDelay"* > </**CheckBox**>

<**Label** text=*"Делэй"* > </**Label**>

</**HBox**>

<**VBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**HBox** spacing=*"5"*>

<**padding**> <**javafx.geometry.Insets** top=*"5"* /> </**padding**>

<**CheckBox** fx:id=*"overdriveCheck"* onAction=*"#createOverdrive"*> </**CheckBox**>

<**Label** text=*"Овердрайв"* > </**Label**>

</**HBox**>

<**VBox**>

<**padding**> <**javafx.geometry.Insets** top=*"10"* /> </**padding**>

<**Slider** fx:id=*"overdriveSlider"*

max=*"7"* min=*"0"* value=*"1.0"* rotate=*"0"* maxWidth=*"110"*>

</**Slider**>

</**VBox**>

</**VBox**>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"130"* /> </**padding**>

<**Button** fx:id=*"buttonClose"* text=*"Close"* maxWidth=*"110"*

onAction=*"#clickClose"* >

</**Button**>

</**HBox**>

</**HBox**>

</**top**>

<**center**>

<**GridPane** hgap=*"50"*>

<**padding**>

<**javafx.geometry.Insets** top=*"5"* right=*"25"* bottom=*"25"* left=*"25"*/>

</**padding**>

<**VBox** spacing=*"10"* GridPane.rowIndex=*"0"* GridPane.columnIndex=*"0"*>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Label** text=*"1.0"* fx:id=*"labelForSlider\_0"* minWidth=*"5"*> </**Label**>

</**HBox**>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Slider** orientation=*"VERTICAL"* max=*"7"* value=*"1.0"* min=*"0"*

fx:id=*"Slider\_0"* minHeight=*"78"*>

</**Slider**>

</**HBox**>

<**Label** text=*"1-350 Hz"* > </**Label**>

</**VBox**>

<**VBox** spacing=*"10"* GridPane.rowIndex=*"0"* GridPane.columnIndex=*"1"*>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Label** text=*"1.0"* fx:id=*"labelForSlider\_1"* minWidth=*"5"*> </**Label**>

</**HBox**>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Slider** orientation=*"VERTICAL"* max=*"7"* value=*"1.0"* min=*"0"*

fx:id=*"Slider\_1"* minHeight=*"78"*>

</**Slider**>

</**HBox**>

<**Label** text=*"0.3-1 kHz"* > </**Label**>

</**VBox**>

<**VBox** spacing=*"10"* GridPane.rowIndex=*"0"* GridPane.columnIndex=*"2"*>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"30"* /> </**padding**>

<**Label** text=*"1.0"* fx:id=*"labelForSlider\_2"* minWidth=*"5"*> </**Label**>

</**HBox**>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"30"* /> </**padding**>

<**Slider** orientation=*"VERTICAL"* max=*"7"* value=*"1.0"* min=*"0"* fx:id=*"Slider\_2"*>

</**Slider**>

</**HBox**>

<**Label** text=*"0.9-2.4 kHz"* > </**Label**>

</**VBox**>

<**VBox** spacing=*"10"* GridPane.rowIndex=*"0"* GridPane.columnIndex=*"3"*>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Label** text=*"1.0"* fx:id=*"labelForSlider\_3"* minWidth=*"5"*> </**Label**>

</**HBox**>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Slider** orientation=*"VERTICAL"* max=*"7"* value=*"1.0"* min=*"0"*

fx:id=*"Slider\_3"* minHeight=*"78"*>

</**Slider**>

</**HBox**>

<**Label** text=*"2.2-5.1 kHz"* > </**Label**>

</**VBox**>

<**VBox** spacing=*"10"* GridPane.rowIndex=*"0"* GridPane.columnIndex=*"4"*>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Label** text=*"1.0"* fx:id=*"labelForSlider\_4"* minWidth=*"5"*> </**Label**>

</**HBox**>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Slider** orientation=*"VERTICAL"* max=*"7"* value=*"1.0"* min=*"0"*

fx:id=*"Slider\_4"* minHeight=*"78"*>

</**Slider**>

</**HBox**>

<**Label** text=*"4.9-10.6 kHz"* > </**Label**>

</**VBox**>

<**VBox** spacing=*"10"* GridPane.rowIndex=*"0"* GridPane.columnIndex=*"5"*>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Label** text=*"1.0"* fx:id=*"labelForSlider\_5"* minWidth=*"5"*> </**Label**>

</**HBox**>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Slider** orientation=*"VERTICAL"* max=*"7"* value=*"1.0"* min=*"0"*

fx:id=*"Slider\_5"* minHeight=*"78"*>

</**Slider**>

</**HBox**>

<**Label** text=*"9.5-21.2 kHz"* > </**Label**>

</**VBox**>

<**VBox** spacing=*"10"* GridPane.rowIndex=*"0"* GridPane.columnIndex=*"6"*>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Label** text=*"1.0"* fx:id=*"labelForSlider\_6"* minWidth=*"5"*> </**Label**>

</**HBox**>

<**HBox**>

<**padding**> <**javafx.geometry.Insets** left=*"25"* /> </**padding**>

<**Slider** orientation=*"VERTICAL"* max=*"7"* value=*"1.0"* min=*"0"*

fx:id=*"Slider\_6"* minHeight=*"78"*>

</**Slider**>

</**HBox**>

<**Label** text=*"21.2-48 kHz"* > </**Label**>

</**VBox**>

</**GridPane**>

</**center**>

<**bottom**>

<**LineChart** fx:id=*"graph"* maxHeight=*"300"* >

<**xAxis**>

<**NumberAxis** label=*"Frequency"* lowerBound=*"0"* upperBound=*"44100"* />

</**xAxis**>

<**yAxis**>

<**NumberAxis** label=*"Magnitude"* lowerBound=*"0"* upperBound=*"3"* />

</**yAxis**>

</**LineChart**>

</**bottom**>

</**BorderPane**>

Пакет ru.bmstu.www.input содержит в себе класс для чтения файла – ReadMusicFile.java и класс, опиcывающий структуру \*.wav файла – AudioFileFormat.java

Класс ReadMusicFile.java

**package** ru.bmstu.www.input;

**import** java.io.File;

**import** java.io.IOException;

**import** javax.sound.sampled.AudioFormat;

**import** javax.sound.sampled.AudioInputStream;

**import** javax.sound.sampled.AudioSystem;

**import** javax.sound.sampled.LineUnavailableException;

**import** javax.sound.sampled.SourceDataLine;

**import** javax.sound.sampled.UnsupportedAudioFileException;

**public** **class** **ReadMusicFile**{

**private** **AudioInputStream** ais;

**private** **byte**[] outputSignal;

**private** SourceDataLine sdl;

**public** **ReadMusicFile**(**File** filePath) **throws** **UnsupportedAudioFileException**, **IOException**, **InterruptedException**, **LineUnavailableException** {

**if**(filePath != **null**) {

**this**.ais = **AudioSystem**.*getAudioInputStream*(filePath);

**AudioFormat** **format** = ais.getFormat();

**this**.sdl = **AudioSystem**.*getSourceDataLine*(format);

**this**.sdl.flush();

}

}

**public** **byte**[] **getOutputSignal**() {

**return** **this**.outputSignal;

}

**public** **AudioInputStream** **getAudioInputStream**() {

**return** **this**.ais;

}

**public** **void** **closeAudioInputStream**() {

**try** {

**this**.ais.close();

} **catch** (**IOException** **e**) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**public** SourceDataLine **getSourceDataLine**() {

**return** **this**.sdl;

}

}

Класс AudioFileFormat.java

**package** ru.bmstu.www.input;

**public** **class** **AudioFileFormat** {

**private** **boolean** bigEndian;

**private** **boolean** signed;

**int** bits;

**int** channels;

**double** sampleRate;

**public** **AudioFileFormat**() {

**this**.bigEndian = **false**;

**this**.signed = **true**;

**this**.bits = 16;

**this**.channels = 2;

**this**.sampleRate= 44100.0;

}

**public** **boolean** **isBigEndian**() {

**return** **this**.bigEndian;

}

**public** **boolean** **isSigned**() {

**return** **this**.signed;

}

**public** **int** **getBits**() {

**return** **this**.bits;

}

**public** **int** **getChannels**() {

**return** **this**.channels;

}

**public** **double** **getSampleRate**() {

**return** **this**.sampleRate;

}

}

Пакет ru.bmstu.www.filter содержит в себе класс Filter.java, описывающий работу фильтра посредством свертки, пакет ru.bmstu.www.filter.coefs.Filter.Info.java содержит данные для кэширования, описывающие каждый КИХ фильтр.

Класс Filter.java

**package** ru.bmstu.www.filter;

**import** java.util.concurrent.Callable;

**public** **class** **Filter** **implements** Callable<**short**[]> {

**protected** **int** countOfCoefs;

**protected** **double**[] coefsFilter;

**protected** **short**[] inputSignal;

**protected** **short**[] outputSignal;

**protected** **double** gain;

**public** **Filter**(){

gain = 1;

}

**public** **Filter**(**double**[] coefsFilter, **int** countOfCoefs, **short**[] inputSignal) {

}

**public** **void** **settings**(**double**[] coefsFilter, **int** countOfCoefs, **short**[] inputSignal) {

**this**.inputSignal = inputSignal;

**this**.coefsFilter = coefsFilter;

**this**.countOfCoefs = countOfCoefs;

**this**.outputSignal = **new** **short**[inputSignal.length + countOfCoefs];

}

**private** **short**[] **svertka**() {

**int** **multiplication**;

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

**for**(**int** **j** = 0; j < **this**.countOfCoefs; j++) {

multiplication = (**int**) (**this**.inputSignal[i] \* **this**.coefsFilter[j]);

**if**(gain == 1.0)

**this**.outputSignal[i + j] += 0.145 \* (**short**)(multiplication );

**else**

**this**.outputSignal[i + j] += 0.13 \* gain \* (**short**)(multiplication );

}

}

**return** **this**.outputSignal;

}

**public** **void** **setGain**(**float** d) {

**this**.gain = d;

}

**public** **double** **getGain**() {

**return** **this**.gain;

}

**public** **short**[] **getOutputSignal**() {

**return** **this**.outputSignal;

}

**public** **long** **getCountOfSamples**() {

**return** **this**.inputSignal.length;

}

***@Override***

**public** **short**[] **call**() **throws** **Exception** {

**this**.svertka();

**return** **this**.outputSignal;

}

}

Класс ru.bmstu.www.filter.coefs.Filter.Info.java

**package** ru.bmstu.www.filter.coefs;

**public** **final** **class** **FilterInfo** {

**public** **final** **static** **short** ***COUNT\_OF\_COEFS*** = 71;

**public** **static** **final** **double**[] ***CoefsOfBand\_0*** = {

0.009727802235495, 0.01005372011768, 0.01037661723222, 0.01069601749223,

0.01101144643888, 0.01132243218444, 0.01162850635496, 0.01192920503026,

0.01222406967917, 0.01251264808775, 0.01279449527833, 0.01306917441716,

0.01333625770857, 0.01359532727351, 0.01384597601033, 0.01408780843588,

0.0143204415047, 0.01454350540458, 0.01475664432642, 0.01495951720665,

0.01515179844026, 0.015333178563, 0.01550336490076, 0.01566208218492,

0.01580907313185, 0.01594409898543, 0.0160669400211, 0.01617739601028,

0.01627528664408, 0.01636045191517, 0.01643275245696, 0.01649206983926,

0.01653830681965, 0.01657138755, 0.01659125773765, 0.01659788476075,

0.01659125773765, 0.01657138755, 0.01653830681965, 0.01649206983926,

0.01643275245696, 0.01636045191517, 0.01627528664408, 0.01617739601028,

0.0160669400211, 0.01594409898543, 0.01580907313185, 0.01566208218492,

0.01550336490076, 0.015333178563, 0.01515179844026, 0.01495951720665,

0.01475664432642, 0.01454350540458, 0.0143204415047, 0.01408780843588,

0.01384597601033, 0.01359532727351, 0.01333625770857, 0.01306917441716,

0.01279449527833, 0.01251264808775, 0.01222406967917, 0.01192920503026,

0.01162850635496, 0.01132243218444, 0.01101144643888, 0.01069601749223,

0.01037661723222, 0.01005372011768, 0.009727802235495

};

**public** **static** **final** **double**[] ***CoefsOfBand\_1*** = {

-0.01755756425183, -0.01771361079537, -0.01769510813836, -0.01749192351103,

-0.01709588211672, -0.01650093720183, -0.01570331426198, -0.01470162678113,

-0.01349696128617, -0.0120929299248, -0.01049568923314,-0.008713924244489,

-0.006758797594599,-0.004643863794393,-0.002384949359484,5.868619963598e-018,

0.00249110342483, 0.005066758002567, 0.007703878277482, 0.01037815912207,

0.01306435854441, 0.0157365966809, 0.01836866695348, 0.02093435515832,

0.02340776210385, 0.02576362533107, 0.02797763543167, 0.0300267425288,

0.03188944860207, 0.03354608151981, 0.03497904688676, 0.03617305411895,

0.03711531351648, 0.03779570151306, 0.03820689173242, 0.03834444996972,

0.03820689173242, 0.03779570151306, 0.03711531351648, 0.03617305411895,

0.03497904688676, 0.03354608151981, 0.03188944860207, 0.0300267425288,

0.02797763543167, 0.02576362533107, 0.02340776210385, 0.02093435515832,

0.01836866695348, 0.0157365966809, 0.01306435854441, 0.01037815912207,

0.007703878277482, 0.005066758002567, 0.00249110342483,5.868619963598e-018,

-0.002384949359484,-0.004643863794393,-0.006758797594599,-0.008713924244489,

-0.01049568923314, -0.0120929299248, -0.01349696128617, -0.01470162678113,

-0.01570331426198, -0.01650093720183, -0.01709588211672, -0.01749192351103,

-0.01769510813836, -0.01771361079537, -0.01755756425183

};

**public** **static** **final** **double**[] ***CoefsOfBand\_2*** = {

0.00057092896356,-0.001894326265919,-0.003802884235666,-0.004955419010411,

-0.005236532750047,-0.004633441162433,-0.003244315063616,-0.001274741278846,

0.0009780315666856, 0.003152100742915, 0.00485695349458, 0.005715250679225,

0.005406270726679, 0.003706544188376, 0.000523267305691,-0.004083398476294,

-0.009892091300092, -0.01652774246778, -0.02348495411561, -0.03016583019527,

-0.03592932568836, -0.04014791164658, -0.04226645858023, -0.04185781906677,

-0.03866969805174, -0.03265803997868, -0.02400328592949, -0.01310736219567,

-0.0005710145828236, 0.01284706624612, 0.02628615109546, 0.03884977007526,

0.04967671400511, 0.05801052559231, 0.06326131946772, 0.06505440229956,

0.06326131946772, 0.05801052559231, 0.04967671400511, 0.03884977007526,

0.02628615109546, 0.01284706624612,-0.0005710145828236, -0.01310736219567,

-0.02400328592949, -0.03265803997868, -0.03866969805174, -0.04185781906677,

-0.04226645858023, -0.04014791164658, -0.03592932568836, -0.03016583019527,

-0.02348495411561, -0.01652774246778,-0.009892091300092,-0.004083398476294,

0.000523267305691, 0.003706544188376, 0.005406270726679, 0.005715250679225,

0.00485695349458, 0.003152100742915,0.0009780315666856,-0.001274741278846,

-0.003244315063616,-0.004633441162433,-0.005236532750047,-0.004955419010411,

-0.003802884235666,-0.001894326265919, 0.00057092896356

};

**public** **static** **final** **double**[] ***CoefsOfBand\_3*** = {

-0.008759193689763, -0.01342666818475, -0.01381382821962, -0.01028487037879,

-0.004961766391355,-0.0007824911947999,-0.0001163208830641,-0.003560526438847,

-0.009500620572317, -0.01469953420739, -0.01571831286182, -0.01056664551305,

0.0001730285767443, 0.01331965049454, 0.02427443443558, 0.02896695964301,

0.02573341211249, 0.01625208541634, 0.005015682851624, -0.00255805675641,

-0.002644894974391, 0.004928678104298, 0.01608283215434, 0.02378249994513,

0.02088773841348, 0.003491252634957, -0.02664558896894, -0.06170838718399,

-0.08990450055042, -0.0994315084575, -0.0829569152048, -0.04090958868068,

0.01777668569574, 0.07793249745313, 0.122780205395, 0.1393512485004,

0.122780205395, 0.07793249745313, 0.01777668569574, -0.04090958868068,

-0.0829569152048, -0.0994315084575, -0.08990450055042, -0.06170838718399,

-0.02664558896894, 0.003491252634957, 0.02088773841348, 0.02378249994513,

0.01608283215434, 0.004928678104298,-0.002644894974391, -0.00255805675641,

0.005015682851624, 0.01625208541634, 0.02573341211249, 0.02896695964301,

0.02427443443558, 0.01331965049454,0.0001730285767443, -0.01056664551305,

-0.01571831286182, -0.01469953420739,-0.009500620572317,-0.003560526438847,

-0.0001163208830641,-0.0007824911947999,-0.004961766391355, -0.01028487037879,

-0.01381382821962, -0.01342666818475,-0.008759193689763

};

**public** **static** **final** **double**[] ***CoefsOfBand\_4*** = {

-0.00151744010927,0.0008486851154434, -0.01015359374533, -0.0158808687229,

-0.001593406265259, 0.01356402446208, 0.009140961417049,4.342770355485e-005,

0.006914211425178, 0.01444900693063,-0.0007474561128633, -0.02193805359582,

-0.01770780291669, 0.001221490744726, 0.001547076233313,-0.009162481488725,

0.003568126808258, 0.03039374570102, 0.0265753362143,-0.006391065219308,

-0.01831078875674, -0.00122481284559,-0.004990372365796, -0.03766790270863,

-0.03628452477496, 0.01931511491073, 0.05348195304853, 0.02191936839515,

-0.000648842695272, 0.04258838052879, 0.055377488357, -0.06438548922355,

-0.2000313008241, -0.1289830687142, 0.1297496039508, 0.2780569840058,

0.1297496039508, -0.1289830687142, -0.2000313008241, -0.06438548922355,

0.055377488357, 0.04258838052879,-0.000648842695272, 0.02191936839515,

0.05348195304853, 0.01931511491073, -0.03628452477496, -0.03766790270863,

-0.004990372365796, -0.00122481284559, -0.01831078875674,-0.006391065219308,

0.0265753362143, 0.03039374570102, 0.003568126808258,-0.009162481488725,

0.001547076233313, 0.001221490744726, -0.01770780291669, -0.02193805359582,

-0.0007474561128633, 0.01444900693063, 0.006914211425178,4.342770355485e-005,

0.009140961417049, 0.01356402446208,-0.001593406265259, -0.0158808687229,

-0.01015359374533,0.0008486851154434, -0.00151744010927

};

**public** **static** **final** **double**[] ***CoefsOfBand\_5*** = {

0.004043659620734,-0.002275278005292, 0.01676347588103, -0.01075296735844,

-0.001375526498289, -0.01085319839188, 0.01334364458755, 0.005684803989199,

-0.002153959592002,-0.009799727018984,-0.008532321687968, 0.01767466716265,

0.0005406209238993, 0.007727142919318, -0.02870547832728, 0.01135403120932,

-0.002046097788684, 0.02821544277939, -0.02093394644516, -0.0078445679965,

-0.01190453584359, 0.02288707454689, 0.01894735237998, -0.01973068893306,

-0.01307774487433, -0.02604935427137, 0.06089980125906, -0.01097901795289,

0.02162935923548, -0.1019151526582, 0.0535939770989, 0.009724823707764,

0.1320946581228, -0.1594219833303, -0.2255043123052, 0.5113800284953,

-0.2255043123052, -0.1594219833303, 0.1320946581228, 0.009724823707764,

0.0535939770989, -0.1019151526582, 0.02162935923548, -0.01097901795289,

0.06089980125906, -0.02604935427137, -0.01307774487433, -0.01973068893306,

0.01894735237998, 0.02288707454689, -0.01190453584359, -0.0078445679965,

-0.02093394644516, 0.02821544277939,-0.002046097788684, 0.01135403120932,

-0.02870547832728, 0.007727142919318,0.0005406209238993, 0.01767466716265,

-0.008532321687968,-0.009799727018984,-0.002153959592002, 0.005684803989199,

0.01334364458755, -0.01085319839188,-0.001375526498289, -0.01075296735844,

0.01676347588103,-0.002275278005292, 0.004043659620734

};

**public** **static** **final** **double**[] ***CoefsOfBand\_6*** = {

0.002662219701948, 0.001116757923598,-0.004914123080256, 0.0079256659083,

-0.009456932559705, 0.009075666089975,-0.006720402854152, 0.002740110920467,

0.002147861397437,-0.006978579961015, 0.0107200769577, -0.01248143858692,

0.01170854227435,-0.008327287347903, 0.00280175566567, 0.003910843055298,

-0.01050775325273, 0.01557022813419, -0.01783734710182, 0.01647535147916,

-0.01129124810821, 0.002846363599009, 0.007558915406074, -0.01803627187782,

0.02637935687915, -0.03041305786472, 0.02836642869383, -0.01920643821972,

0.002873360670959, 0.01962717362101, -0.0462978971025, 0.07438464149151,

-0.1007401676891, 0.1222627405118, -0.1363414111527, 0.1412375317396,

-0.1363414111527, 0.1222627405118, -0.1007401676891, 0.07438464149151,

-0.0462978971025, 0.01962717362101, 0.002873360670959, -0.01920643821972,

0.02836642869383, -0.03041305786472, 0.02637935687915, -0.01803627187782,

0.007558915406074, 0.002846363599009, -0.01129124810821, 0.01647535147916,

-0.01783734710182, 0.01557022813419, -0.01050775325273, 0.003910843055298,

0.00280175566567,-0.008327287347903, 0.01170854227435, -0.01248143858692,

0.0107200769577,-0.006978579961015, 0.002147861397437, 0.002740110920467,

-0.006720402854152, 0.009075666089975,-0.009456932559705, 0.0079256659083,

-0.004914123080256, 0.001116757923598, 0.002662219701948

};

**public** **static** **final** **String** ***InfoBand\_0*** ="LowPass 350 Hz";

**public** **static** **final** **String** ***InfoBand\_1*** ="BandPass 300-1000 Hz";

**public** **static** **final** **String** ***InfoBand\_2*** ="BandPass 950-2350 Hz";

**public** **static** **final** **String** ***InfoBand\_3*** ="BandPass 2300-5100 Hz";

**public** **static** **final** **String** ***InfoBand\_4*** ="BandPass 5000-10600 Hz";

**public** **static** **final** **String** ***InfoBand\_5*** ="BandPass 10000-21200 Hz";

**public** **static** **final** **String** ***InfoBand\_6*** ="HighPass 21000 Hz";

}

Пакет ru.bmstu.ru.www.effects содежит в себе абстрактный класс Effect.java, диктующий правила, по которым должны быть написаны конкретные классы, реализующие конкретные эффекты.

Delay.java – наследник Effect.java, реализующий эффект “задержка”.

Overdrive – наследник Effect.java, реализующий эффект “перегрузка”.

Класс Effect.java

**package** ru.bmstu.www.effects;

**public** **abstract** **class** **Effect** {

**protected** **short**[] inputAudioStream;

**public** **abstract** **short**[] **createEffect**();

**public** **abstract** **short**[] **getOutputAudioStream**();

}

Класс Delay.java

**package** ru.bmstu.www.effects;

**public** **class** **Delay** **extends** **Effect**{

**public** **Delay**() {

**super**();

}

**public** **void** **setInputSampleStream**(**short**[] inputAudioStream) {

**this**.inputAudioStream = inputAudioStream;

}

***@Override***

**public** **synchronized** **short**[] **createEffect**() {

**short** **amplitude** = 0;

**short** **delayAmplitude** = 0;

**int** **checkFlag** = 0;

**int** **delay** = 15000;

**int** **position** = 0;

**for**(**int** **i** = delay ; i < **this**.inputAudioStream.length - 256; i ++) {

amplitude = **this**.inputAudioStream[i];

delayAmplitude = **this**.inputAudioStream[position];

checkFlag = ( (( delayAmplitude) + (**int**)(0.9 \* amplitude)));

**if**(checkFlag < **Short**.***MAX\_VALUE*** && checkFlag > **Short**.***MIN\_VALUE*** ) {

delayAmplitude = (**short**)checkFlag;

**this**.inputAudioStream[position] = delayAmplitude;

position += 1;

}

}

**return** **this**.inputAudioStream;

}

***@Override***

**public** **synchronized** **short**[] **getOutputAudioStream**() {

**return** **this**.inputAudioStream;

}

}

Класс Override.java

**package** ru.bmstu.www.effects;

**public** **class** **Overdrive** **extends** ru.bmstu.www.effects.**Effect** {

**private** **short** maxAmplitude;

**private** **short** minAmplitude;

**private** **static** **final** **short** ***standartMax*** = 3500;

**private** **static** **final** **short** ***standartMin*** = -3500;

**private** **double** coef;

**public** **Overdrive**() {

**super**();

**this**.coef = 1.0;

}

**public** **void** **setInputSampleStream**(**short**[] inputAudioStream) {

**this**.inputAudioStream = inputAudioStream;

}

***@Override***

**public** **synchronized** **short**[] **createEffect**() {

**this**.setMaxAndMinAmpl();

**for**(**int** **i** = 0; i < **this**.inputAudioStream.length - 256; i ++) {

**if**(**this**.inputAudioStream[i] > **this**.maxAmplitude)

**this**.inputAudioStream[i] = (**this**.maxAmplitude);

**else** **if**(**this**.inputAudioStream[i] < **this**.minAmplitude)

**this**.inputAudioStream[i] = (**this**.minAmplitude);

}

**return** **this**.inputAudioStream;

}

**private** **void** **setMaxAndMinAmpl**() {

**this**.maxAmplitude = (**short**) (**Overdrive**.***standartMax*** \* **this**.coef);

**this**.minAmplitude = (**short**) (**Overdrive**.***standartMin*** \* **this**.coef);

}

**public** **void** **setOverdriveCoef**(**double** coef) {

**this**.coef = coef;

}

***@Override***

**public** **synchronized** **short**[] **getOutputAudioStream**() {

**return** **this**.inputAudioStream;

}

}

# Литературы:

1. Недашковский, В.М. Методические указания к выполнению домашнего задания «Проектирование графического цифрового эквалайзера»: учеб. пособие / В.М. Недашковский, В.С. Белолапотков. – М., 2014.
2. Конспекты лекций по курсу «Цифровая обработка сигналов».
3. <https://ru.wikipedia.org/wiki/Дилэй> - Электронный ресурс. Дата обращения – 15.03.2016
4. https://docs.oracle.com/javase/tutorial/sound/ Электронный ресурс. Дата обращения – 11.03.2016
5. <https://ru.wikibooks.org/wiki/Реализации_алгоритмов/Быстрое_преобразование_Фурье> - Электронный ресурс. Дата обращения - 13.04.2016
6. Брюс Эккель «Философия Java» . 4-е полное издание. Издательство «Питер, 2015»