

Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий



Программирование

Отчет по курсовой работе
Приложение "Sun Radio"

Работу выполнила:

Кремнева В.Э.

Группа: 23501/4

Преподаватель:

Вылегжанина К.Д.

Санкт-Петербург
2017

Содержание

1	Проектирование приложения	2
1.1	Задание	2
1.2	Концепция	2
1.3	Минимально работоспособный продукт	2
1.4	Решаемые задачи	2
1.5	Выводы	2
2	Реализация приложения	2
2.1	Среда разработки	2
2.2	Выделенные классы	3
2.3	Выводы	3
3	Процесс обеспечения качества и тестирование	3
3.1	Тестирование	3
3.2	Выводы	3
4	Выводы	3
5	Приложение 1	3

1 Проектирование приложения

В современном мире музыка является неотъемлемой частью жизни многих и многих людей. Музыка часто сопровождает нас на работе, в дороге, во время досуга. Порой музыка может играть долгие часы. Настолько долгие, что за время прослушивания солнце может изменить свое положение относительно прослушивающего. Со временем суток так же меняется и настроение, и запас сил. И как было бы здорово, если бы музыка подстраивалась под наше самочувствие, дополняла и обогащала его.

1.1 Задание

Разработать приложение, позволяющее пользователям автоматически изменять тональность и громкость воспроизводимой музыки в соответствии с уровнем освещенности.

1.2 Концепция

Изменения в музыкальные файлы вносятся с помощью преобразования Фурье. Информация об освещенности поступает с фоторезистра.

1.3 Минимально работоспособный продукт

Консольное приложение, получающее на вход музыкальный файл и воспроизводящее его в соответствии с текущим уровнем освещенности.

1.4 Решаемые задачи

В процессе проектирования приложения было выделено три основных задачи.

- **Получение спектра файла**

Для удобства работы используются .wav файлы. Удобство в том, что в каждом фрейме хранятся значения амплитуд, то есть имеем зависимость амплитуды от времени, что является входными данными для преобразования Фурье. С помощью этого преобразования получим спектр файла – зависимость амплитуды от частоты. Но не всё так просто. При последовательном чтении фреймов накапливаются ошибки, поэтому необходимо использовать оконную функцию и читать данные с перекрытием. Эмпирически получено наилучшее перекрытие – в одну шестнадцатую от количества читаемых фреймов при условии, что читаем по 2048 фреймов. Число фреймов, равное степени двойки, выбрано не случайно: это задел на будущее. Для быстрого преобразования Фурье необходимо число фреймов, равное степени двойки. Используемая оконная функция – функция Блэкмана-Наталла. Выбрана она за минимальный размер боковых лепестков и удовлетворительное "растяжение" спектра. При использовании входного фильтра необходим выходной фильтр. Выходной фильтр должен быть таким, чтобы сумма произведений входного и выходного фильтров в каждой точке была равна единице. Для этого здесь я просто нормализую входную оконную функцию.

- **Корректные преобразования**

Необходимо изменить тон и громкость воспроизведения. Для изменения тона необходимо растянуть файл в N раз, интерполируя значения амплитуды и фазы между точками, а затем ускорить воспроизведение в N раз. Таким образом получим изменение тона при сохранении скорости воспроизведения.

- **Воспроизведение файла**

Воспроизведение файла производится средствами `javax.sound.sampled.*`

1.5 Выводы

В данном разделе рассмотрен процесс проектирования приложения для модуляции звука в зависимости от уровня освещенности. Выделены основные задачи и предложены варианты их решения.

2 Реализация приложения

2.1 Среда разработки

Операционная система: Windows 8.1
Среда разработки: IntelliJ IDEA 2016.3.4
Компилятор: javac, JDK 1.8.0_102

2.2 Выделенные классы

В приложении были выделены следующие классы:

- **SunRadio** - главный класс. Осуществляет основную работу приложения.
- **Complex** - класс для работы с комплексными числами. Взят из открытого источника.
- **WavFile** - класс для работы с .wav файлами. Взят из открытого источника.
- **WavFileException** - исключения для класса WavFile. Взят из открытого источника.
- **LightLevel** - класс, с помощью которого можно получить текущий уровень освещенности.
- **AM** - класс, содержащий средства для амплитудной модуляции.
- **DFTStraight** - реализует прямое дискретное преобразование Фурье.
- **DFTInverse** - реализует обратное дискретное преобразование Фурье.
- **Filter** - реализует оконную функцию.
- **Interpolation** - реализует линейную интерполяцию по двум точкам.
- **Scale** - реализует масштабирование в заданных пределах.
- **ToneModulation** - реализует модуляцию тона.
- **ToneModulationException** - исключения для класса ToneModulation.

2.3 Выводы

В данном разделе были описаны все классы, выделенные в процессе работы над проектом.

3 Процесс обеспечения качества и тестирование

3.1 Тестирование

Для проверки работы библиотеки использовались автоматические тесты, покрывающие основную функциональность ядра. Также в процессе разработки приложения проводилось ручное тестирование программы.

3.2 Выводы

В данном разделе описан процесс тестирования программы.

4 Выводы

В результате работы над курсовым проектом было реализовано приложение, предназначенное для изменения музыки в соответствии с уровнем освещенности. Изучено: преобразование Фурье, оконные функции, амплитудная модуляция, линейная интерполяция. Так же приобретены навыки разработки приложения на языке Java, а также навыки разработки тестов на языке Groovy.

5 Приложение 1

Листинг 1: Complex.java

```
1 /*
2  * This work by W. Patrick Hooper is free of known copyright restrictions.
3  * The work is in the public domain.
4  *
5  * Author's website: <a href="http://wphooper.com">http://wphooper.com</a>.
6  */
7
8 package com.external;
9
```

```

10 /**
11  * This class stores a complex number, and allows the user to do arithmetic
12  * with these numbers.
13  *
14  * Note that our complex numbers are immutable. That is, once they are
15  * constructed, they will not change. In particular, all our algebraic
16  * operations create a new complex number rather than updating the current one.
17  *
18  * @author W. Patrick Hooper
19  */
20 public final class Complex {
21     // The number stored is x+I*y.
22     final private double x, y;
23     // I don't want to allow anyone to access these numbers so I've labeled
24     // them private.
25
26     /** Construct a point from real and imaginary parts. */
27     public Complex(double real_part, double imaginary_part) {
28         x=real_part;
29         y=imaginary_part;
30     }
31
32     /** Construct a real number. */
33     public Complex(double real_part) {
34         x=real_part;
35         y=0;
36     }
37
38     // A static constructor.
39
40     /** Construct a complex number from the given polar coordinates. */
41     public static Complex fromPolar(double r, double theta) {
42         return new Complex(r*Math.cos(theta), r*Math.sin(theta));
43     }
44
45     // Basic operations on Complex numbers.
46
47     /** Return the real part. */
48     public double re(){
49         return x;
50     }
51
52     /** Return the imaginary part. */
53     public double im(){
54         return y;
55     }
56
57     /** Return the complex conjugate */
58     public Complex conj() {
59         return new Complex(x,-y);
60     }
61
62     /** Return the square of the absolute value. */
63     public double absSquared() {
64         return x*x+y*y;
65     }
66
67     /** Return the absolute value. */
68     public double abs() {
69         // The java.lang.Math package contains many useful mathematical functions,
70         // including the square root function.
71         return Math.sqrt(absSquared());
72     }
73
74     // ARITHMETIC
75
76     /** Add a complex number to this one.
77     *
78     * @param z The complex number to be added.
79     * @return A new complex number which is the sum.
80     */
81     public Complex add(Complex z) {
82         return new Complex(x+z.x, y+z.y);
83     }
84
85     /** Subtract a complex number from this one.

```

```

86     *
87     * @param z The complex number to be subtracted.
88     * @return A new complex number which is the sum.
89     */
90     public Complex minus(Complex z) {
91         return new Complex(x-z.x, y-z.y);
92     }
93
94     /** Negate this complex number.
95     *
96     * @return The negation.
97     */
98     public Complex neg() {
99         return new Complex(-x, -y);
100     }
101
102     /** Compute the product of two complex numbers
103     *
104     * @param z The complex number to be multiplied.
105     * @return A new complex number which is the product.
106     */
107     public Complex mult(Complex z) {
108         return new Complex(x*z.x-y*z.y, x*z.y+z.x*y);
109     }
110
111     /** Divide this complex number by a real number.
112     *
113     * @param q The number to divide by.
114     * @return A new complex number representing the quotient.
115     */
116     public Complex div(double q) {
117         return new Complex(x/q, y/q);
118     }
119
120     /** Return the multiplicative inverse. */
121     public Complex inv() {
122         // find the square of the absolute value of this complex number.
123         double abs_squared=absSquared();
124         return new Complex(x/abs_squared, -y/abs_squared);
125     }
126
127     /** Compute the quotient of two complex numbers.
128     *
129     * @param z The complex number to divide this one by.
130     * @return A new complex number which is the quotient.
131     */
132     public Complex div(Complex z) {
133         return mult(z.inv());
134     }
135
136     /** Return the complex exponential of this complex number. */
137     public Complex exp() {
138         return new Complex(Math.exp(x)*Math.cos(y), Math.exp(x)*Math.sin(y));
139     }
140
141
142     // FUNCTIONS WHICH KEEP JAVA HAPPY:
143
144     /** Returns this point as a string.
145     * The main purpose of this function is for printing the string out,
146     * so we return a string in a (fairly) human readable format.
147     */
148     // The _optional_ override directive "@Override" below just says we are
149     // overriding a function defined in a parent class. In this case, the
150     // parent is java.lang.Object. All classes in Java have the Object class
151     // as a superclass.
152     @Override
153     public String toString() {
154         // Comments:
155         // 1) "" represents the empty string.
156         // 2) If you add something to a string, it converts the thing you
157         // are adding to a string, and then concatenates it with the string.
158
159         // We do some voodoo to make sure the number is displayed reasonably.
160         if (y==0) {
161             return ""+x;

```

```

162     }
163     if (y>0) {
164         return ""+x+" "+y+"*I";
165     }
166     // otherwise y<0.
167     return ""+x+" "+(-y)+"*I";
168 }
169
170 /** Return true if the object is a complex number which is equal to this complex number.
171 → */
172 @Override
173 public boolean equals(Object obj) {
174     // Return false if the object is null
175     if (obj == null) {
176         return false;
177     }
178     // Return false if the object is not a Complex number
179     if (!(obj instanceof Complex)) {
180         return false;
181     }
182     // Now the object must be a Complex number, so we can convert it to a
183     // Complex number.
184     Complex other = (Complex) obj;
185
186     // If the x-coordinates are not equal, then return false.
187     if (x != other.x) {
188         return false;
189     }
190     // If the y-coordinates are not equal, then return false.
191     if (y != other.y) {
192         return false;
193     }
194     // Both parts are equal, so return true.
195     return true;
196 }
197
198 // Remark: In Java, we should really override the hashCode function
199 // whenever we override the equals function. But, I don't want to
200 // get into this for a light introduction to programming in java.
201 // Hash codes are necessary for various of Java's collections. See HashSet for instance.
202 // The following was generated by Netbeans.
203 @Override
204 public int hashCode() {
205     int hash = 3;
206     hash = 83 * hash + (int) (Double.doubleToLongBits(this.x) ^ (Double.doubleToLongBits(
207 → this.x) >>> 32));
208     hash = 83 * hash + (int) (Double.doubleToLongBits(this.y) ^ (Double.doubleToLongBits(
209 → this.y) >>> 32));
210     return hash;
211 }

```

Листинг 2: WavFile.java

```

1 package com.external;
2
3 import java.io.*;
4
5 import javax.sound.sampled.AudioFormat;
6 import javax.sound.sampled.AudioSystem;
7
8
9 /**
10  * Wav file IO class
11  * http://www.labbookpages.co.uk
12  *
13  * File format is based on the information from
14  * http://www.sonicspot.com/guide/wavefiles.html
15  * http://www.blitter.com/~russtopia/MIDI/~jglatt/tech/wave.htm
16  *
17  * @author A.Greensted
18  * @version 1.0
19  */
20 public class WavFile
21 {

```

```

22 private enum IOState {READING, WRITING, CLOSED};
23 private final static int BUFFER_SIZE = 4096;
24
25 private final static int FMT_CHUNK_ID = 0x20746D66;
26 private final static int DATA_CHUNK_ID = 0x61746164;
27 private final static int RIFF_CHUNK_ID = 0x46464952;
28 private final static int RIFF_TYPE_ID = 0x45564157;
29
30 private File file; // File that will be read from or written to
31 private IOState ioState; // Specifies the IO State of the Wav File (used for snaity
    ↪ checking)
32 private int bytesPerSample; // Number of bytes required to store a single sample
33 private long numFrames; // Number of frames within the data section
34 private FileOutputStream oStream; // Output stream used for writting data
35 private FileInputStream iStream; // Input stream used for reading data
36 private double floatScale; // Scaling factor used for int <=> float conversion
37 private double floatOffset; // Offset factor used for int <=> float conversion
38 private boolean wordAlignAdjust; // Specify if an extra byte at the end of the data chunk
    ↪ is required for word alignment
39
40 // Wav Header
41 private int numChannels; // 2 bytes unsigned, 0x0001 (1) to 0xFFFF (65,535)
42 private long sampleRate; // 4 bytes unsigned, 0x00000001 (1) to 0xFFFFFFFF
    ↪ (4,294,967,295)
43 // Although a java int is 4 bytes, it is signed, so need to use a
    ↪ long
44 private int blockAlign; // 2 bytes unsigned, 0x0001 (1) to 0xFFFF (65,535)
45 private int validBits; // 2 bytes unsigned, 0x0002 (2) to 0xFFFF (65,535)
46
47 // Buffering
48 private byte[] buffer; // Local buffer used for IO
49 private int bufferPointer; // Points to the current position in local buffer
50 private int bytesRead; // Bytes read after last read into local buffer
51 private long frameCounter; // Current number of frames read or written
52
53 // Cannot instantiate WavFile directly, must either use newWavFile() or openWavFile()
54 private WavFile()
55 {
56     buffer = new byte[BUFFER_SIZE];
57 }
58
59 public int getNumChannels()
60 {
61     return numChannels;
62 }
63
64 public long getNumFrames()
65 {
66     return numFrames;
67 }
68
69 public long getFramesRemaining()
70 {
71     return numFrames - frameCounter;
72 }
73
74 public long getFrameCounter() {
75     return frameCounter;
76 }
77
78 public long getSampleRate()
79 {
80     return sampleRate;
81 }
82
83 public int getValidBits()
84 {
85     return validBits;
86 }
87 public int getblockAlign()
88 {
89     return blockAlign;
90 }
91
92 public AudioFormat getAudioFormat() throws Exception
93 {

```



```

94     return AudioSystem.getAudioInputStream(file).getFormat();
95 }
96
97 public static WavFile newWavFile(File file, int numChannels, long numFrames, int validBits,
98     ↪ long sampleRate) throws IOException, WavFileException
99 {
100     // Instantiate new Wavfile and initialise
101     WavFile wavFile = new WavFile();
102     wavFile.file = file;
103     wavFile.numChannels = numChannels;
104     wavFile.numFrames = numFrames;
105     wavFile.sampleRate = sampleRate;
106     wavFile.bytesPerSample = (validBits + 7) / 8;
107     wavFile.blockAlign = wavFile.bytesPerSample * numChannels;
108     wavFile.validBits = validBits;
109
110     // Sanity check arguments
111     if (numChannels < 1 || numChannels > 65535) throw new WavFileException("Illegal_number_of_
112     ↪ channels,_valid_range_1_to_65536");
113     if (numFrames < 0) throw new WavFileException("Number_of_frames_must_be_positive");
114     if (validBits < 2 || validBits > 65535) throw new WavFileException("Illegal_number_of_
115     ↪ valid_bits,_valid_range_2_to_65536");
116     if (sampleRate < 0) throw new WavFileException("Sample_rate_must_be_positive");
117
118     // Create output stream for writing data
119     wavFile.oStream = new FileOutputStream(file);
120
121     // Calculate the chunk sizes
122     long dataChunkSize = wavFile.blockAlign * numFrames;
123     long mainChunkSize = 4 + // Riff Type
124         8 + // Format ID and size
125         16 + // Format data
126         8 + // Data ID and size
127         dataChunkSize;
128
129     // Chunks must be word aligned, so if odd number of audio data bytes
130     // adjust the main chunk size
131     if (dataChunkSize % 2 == 1) {
132         mainChunkSize += 1;
133         wavFile.wordAlignAdjust = true;
134     }
135     else {
136         wavFile.wordAlignAdjust = false;
137     }
138
139     // Set the main chunk size
140     putLE(RIFF_CHUNK_ID, wavFile.buffer, 0, 4);
141     putLE(mainChunkSize, wavFile.buffer, 4, 4);
142     putLE(RIFF_TYPE_ID, wavFile.buffer, 8, 4);
143
144     // Write out the header
145     wavFile.oStream.write(wavFile.buffer, 0, 12);
146
147     // Put format data in buffer
148     long averageBytesPerSecond = sampleRate * wavFile.blockAlign;
149
150     putLE(FMT_CHUNK_ID, wavFile.buffer, 0, 4); // Chunk ID
151     putLE(16, wavFile.buffer, 4, 4); // Chunk Data Size
152     putLE(1, wavFile.buffer, 8, 2); // Compression Code (Uncompressed)
153     putLE(numChannels, wavFile.buffer, 10, 2); // Number of channels
154     putLE(sampleRate, wavFile.buffer, 12, 4); // Sample Rate
155     putLE(averageBytesPerSecond, wavFile.buffer, 16, 4); // Average Bytes Per Second
156     putLE(wavFile.blockAlign, wavFile.buffer, 20, 2); // Block Align
157     putLE(validBits, wavFile.buffer, 22, 2); // Valid Bits
158
159     // Write Format Chunk
160     wavFile.oStream.write(wavFile.buffer, 0, 24);
161
162     // Start Data Chunk
163     putLE(DATA_CHUNK_ID, wavFile.buffer, 0, 4); // Chunk ID
164     putLE(dataChunkSize, wavFile.buffer, 4, 4); // Chunk Data Size
165
166     // Write Format Chunk
167     wavFile.oStream.write(wavFile.buffer, 0, 8);
168
169     // Calculate the scaling factor for converting to a normalised double

```

```

167     if (wavFile.validBits > 8)
168     {
169         // If more than 8 validBits, data is signed
170         // Conversion required multiplying by magnitude of max positive value
171         wavFile.floatOffset = 0;
172         wavFile.floatScale = Long.MAX_VALUE >> (64 - wavFile.validBits);
173     }
174     else
175     {
176         // Else if 8 or less validBits, data is unsigned
177         // Conversion required dividing by max positive value
178         wavFile.floatOffset = 1;
179         wavFile.floatScale = 0.5 * ((1 << wavFile.validBits) - 1);
180     }
181
182     // Finally, set the IO State
183     wavFile.bufferPointer = 0;
184     wavFile.bytesRead = 0;
185     wavFile.frameCounter = 0;
186     wavFile.ioState = IOState.WRITING;
187
188     return wavFile;
189 }
190
191 public static WavFile openWavFile(File file) throws IOException, WavFileException
192 {
193     // Instantiate new Wavfile and store the file reference
194     WavFile wavFile = new WavFile();
195     wavFile.file = file;
196
197     // Create a new file input stream for reading file data
198     wavFile.iStream = new FileInputStream(file);
199
200     // Read the first 12 bytes of the file
201     int bytesRead = wavFile.iStream.read(wavFile.buffer, 0, 12);
202     if (bytesRead != 12) throw new WavFileException("Not_enough_wav_file_bytes_for_header");
203
204     // Extract parts from the header
205     long riffChunkID = getLE(wavFile.buffer, 0, 4);
206     long chunkSize = getLE(wavFile.buffer, 4, 4);
207     long riffTypeID = getLE(wavFile.buffer, 8, 4);
208
209     // Check the header bytes contains the correct signature
210     if (riffChunkID != RIFF_CHUNK_ID) throw new WavFileException("Invalid_Wav_Header_data, ↪
incorrect_riff_chunk_ID");
211     if (riffTypeID != RIFF_TYPE_ID) throw new WavFileException("Invalid_Wav_Header_data, ↪
incorrect_riff_type_ID");
212
213     // Check that the file size matches the number of bytes listed in header
214     if (file.length() != chunkSize+8) {
215         throw new WavFileException("Header_chunk_size_(" + chunkSize + ")_does_not_match_file_ ↪
size_(" + file.length() + ")");
216     }
217
218     boolean foundFormat = false;
219     boolean foundData = false;
220
221     // Search for the Format and Data Chunks
222     while (true)
223     {
224         // Read the first 8 bytes of the chunk (ID and chunk size)
225         bytesRead = wavFile.iStream.read(wavFile.buffer, 0, 8);
226         if (bytesRead == -1) throw new WavFileException("Reached_end_of_file_without_finding_ ↪
format_chunk");
227         if (bytesRead != 8) throw new WavFileException("Could_not_read_chunk_header");
228
229         // Extract the chunk ID and Size
230         long chunkID = getLE(wavFile.buffer, 0, 4);
231         chunkSize = getLE(wavFile.buffer, 4, 4);
232
233         // Word align the chunk size
234         // chunkSize specifies the number of bytes holding data. However,
235         // the data should be word aligned (2 bytes) so we need to calculate
236         // the actual number of bytes in the chunk
237         long numChunkBytes = (chunkSize%2 == 1) ? chunkSize+1 : chunkSize;
238

```

```

239     if (chunkID == FMT_CHUNK_ID)
240     {
241         // Flag that the format chunk has been found
242         foundFormat = true;
243
244         // Read in the header info
245         bytesRead = wavFile.iStream.read(wavFile.buffer, 0, 16);
246
247         // Check this is uncompressed data
248         int compressionCode = (int) getLE(wavFile.buffer, 0, 2);
249         if (compressionCode != 1) throw new WavFileException("Compression_Code_" +
    ↪ compressionCode + "_not_supported");
250
251         // Extract the format information
252         wavFile.numChannels = (int) getLE(wavFile.buffer, 2, 2);
253         wavFile.sampleRate = getLE(wavFile.buffer, 4, 4);
254         wavFile.blockAlign = (int) getLE(wavFile.buffer, 12, 2);
255         wavFile.validBits = (int) getLE(wavFile.buffer, 14, 2);
256
257         if (wavFile.numChannels == 0) throw new WavFileException("Number_of_channels_specified
    ↪ in_header_is_equal_to_zero");
258         if (wavFile.blockAlign == 0) throw new WavFileException("Block_Align_specified_in_
    ↪ header_is_equal_to_zero");
259         if (wavFile.validBits < 2) throw new WavFileException("Valid_Bits_specified_in_header_
    ↪ is_less_than_2");
260         if (wavFile.validBits > 64) throw new WavFileException("Valid_Bits_specified_in_header
    ↪ is_greater_than_64,_this_is_greater_than_a_long_can_hold");
261
262         // Calculate the number of bytes required to hold 1 sample
263         wavFile.bytesPerSample = (wavFile.validBits + 7) / 8;
264         if (wavFile.bytesPerSample * wavFile.numChannels != wavFile.blockAlign)
265         ↪ throw new WavFileException("Block_Align_does_not_agree_with_bytes_required_for_
    ↪ validBits_and_number_of_channels");
266
267         // Account for number of format bytes and then skip over
268         // any extra format bytes
269         numChunkBytes -= 16;
270         if (numChunkBytes > 0) wavFile.iStream.skip(numChunkBytes);
271     }
272     else if (chunkID == DATA_CHUNK_ID)
273     {
274         // Check if we've found the format chunk,
275         // If not, throw an exception as we need the format information
276         // before we can read the data chunk
277         if (foundFormat == false) throw new WavFileException("Data_chunk_found_before_Format_
    ↪ chunk");
278
279         // Check that the chunkSize (wav data length) is a multiple of the
280         // block align (bytes per frame)
281         if (chunkSize % wavFile.blockAlign != 0) throw new WavFileException("Data_Chunk_size_
    ↪ is_not_multiple_of_Block_Align");
282
283         // Calculate the number of frames
284         wavFile.numFrames = chunkSize / wavFile.blockAlign;
285
286         // Flag that we've found the wave data chunk
287         foundData = true;
288
289         break;
290     }
291     else
292     {
293         // If an unknown chunk ID is found, just skip over the chunk data
294         wavFile.iStream.skip(numChunkBytes);
295     }
296 }
297
298 // Throw an exception if no data chunk has been found
299 if (foundData == false) throw new WavFileException("Did_not_find_a_data_chunk");
300
301 // Calculate the scaling factor for converting to a normalised double
302 if (wavFile.validBits > 8)
303 {
304     // If more than 8 validBits, data is signed
305     // Conversion required dividing by magnitude of max negative value
306     wavFile.floatOffset = 0;

```

```

307     wavFile.floatScale = 1 << (wavFile.validBits - 1);
308 }
309 else
310 {
311     // Else if 8 or less validBits, data is unsigned
312     // Conversion required dividing by max positive value
313     wavFile.floatOffset = -1;
314     wavFile.floatScale = 0.5 * ((1 << wavFile.validBits) - 1);
315 }
316
317 wavFile.bufferPointer = 0;
318 wavFile.bytesRead = 0;
319 wavFile.frameCounter = 0;
320 wavFile.ioState = IOState.READING;
321
322 return wavFile;
323 }
324
325 // Get and Put little endian data from local buffer
326 // -----
327 private static long getLE(byte[] buffer, int pos, int numBytes)
328 {
329     numBytes --;
330     pos += numBytes;
331
332     long val = buffer[pos] & 0xFF;
333     for (int b=0 ; b<numBytes ; b++) val = (val << 8) + (buffer[--pos] & 0xFF);
334
335     return val;
336 }
337
338 private static void putLE(long val, byte[] buffer, int pos, int numBytes)
339 {
340     for (int b=0 ; b<numBytes ; b++)
341     {
342         buffer[pos] = (byte) (val & 0xFF);
343         val >>= 8;
344         pos ++;
345     }
346 }
347
348 // Sample Writing and Reading
349 // -----
350 private void writeSample(long val) throws IOException
351 {
352     for (int b=0 ; b<bytesPerSample ; b++)
353     {
354         if (bufferPointer == BUFFER_SIZE)
355         {
356             oStream.write(buffer, 0, BUFFER_SIZE);
357             bufferPointer = 0;
358         }
359
360         buffer[bufferPointer] = (byte) (val & 0xFF);
361         val >>= 8;
362         bufferPointer ++;
363     }
364 }
365
366 private long readSample() throws IOException, WavFileException
367 {
368     long val = 0;
369
370     for (int b=0 ; b<bytesPerSample ; b++)
371     {
372         if (bufferPointer == bytesRead)
373         {
374             int read = iStream.read(buffer, 0, BUFFER_SIZE);
375             if (read == -1) throw new WavFileException("Not enough data available");
376             bytesRead = read;
377             bufferPointer = 0;
378         }
379
380         int v = buffer[bufferPointer];
381         if (b < bytesPerSample-1 || bytesPerSample == 1) v &= 0xFF;
382         val += v << (b * 8);

```

```

383     bufferPointer++;
384 }
385
386 return val;
387 }
388
389 // Integer
390 // -----
391 public int readFrames(int[] sampleBuffer, int numFramesToRead) throws IOException,
392     ↳ WavFileException
393 {
394     return readFrames(sampleBuffer, 0, numFramesToRead);
395 }
396
397 public int readFrames(int[] sampleBuffer, int offset, int numFramesToRead) throws
398     ↳ IOException, WavFileException
399 {
400     if (ioState != IOState.READING) throw new IOException("Cannot_read_from_WavFile_instance")
401     ↳ ;
402
403     for (int f=0 ; f<numFramesToRead ; f++)
404     {
405         if (frameCounter == numFrames) return f;
406
407         for (int c=0 ; c<numChannels ; c++)
408         {
409             sampleBuffer[offset] = (int) readSample();
410             offset++;
411         }
412
413         frameCounter++;
414     }
415
416     return numFramesToRead;
417 }
418
419 public int readFrames(int[][] sampleBuffer, int numFramesToRead) throws IOException,
420     ↳ WavFileException
421 {
422     return readFrames(sampleBuffer, 0, numFramesToRead);
423 }
424
425 public int readFrames(int[][] sampleBuffer, int offset, int numFramesToRead) throws
426     ↳ IOException, WavFileException
427 {
428     if (ioState != IOState.READING) throw new IOException("Cannot_read_from_WavFile_instance")
429     ↳ ;
430
431     for (int f=0 ; f<numFramesToRead ; f++)
432     {
433         if (frameCounter == numFrames) return f;
434
435         for (int c=0 ; c<numChannels ; c++) sampleBuffer[c][offset] = (int) readSample();
436
437         offset++;
438         frameCounter++;
439     }
440
441     return numFramesToRead;
442 }
443
444 public int writeFrames(int[] sampleBuffer, int numFramesToWrite) throws IOException,
445     ↳ WavFileException
446 {
447     return writeFrames(sampleBuffer, 0, numFramesToWrite);
448 }
449
450 public int writeFrames(int[] sampleBuffer, int offset, int numFramesToWrite) throws
451     ↳ IOException, WavFileException
452 {
453     if (ioState != IOState.WRITING) throw new IOException("Cannot_write_to_WavFile_instance");
454
455     for (int f=0 ; f<numFramesToWrite ; f++)
456     {
457         if (frameCounter == numFrames) return f;

```

```

451     for (int c=0 ; c<numChannels ; c++)
452     {
453         writeSample(sampleBuffer[ offset ]);
454         offset ++;
455     }
456
457     frameCounter ++;
458 }
459
460
461 return numFramesToWrite;
462 }
463
464 public int writeFrames(int [][] sampleBuffer , int numFramesToWrite) throws IOException ,
465     ↪ WavFileException
466 {
467     return writeFrames(sampleBuffer , 0 , numFramesToWrite);
468 }
469
470 public int writeFrames(int [][] sampleBuffer , int offset , int numFramesToWrite) throws
471     ↪ IOException , WavFileException
472 {
473     if (ioState != IOState.WRITING) throw new IOException("Cannot_write_to_WavFile_instance");
474
475     for (int f=0 ; f<numFramesToWrite ; f++)
476     {
477         if (frameCounter == numFrames) return f;
478
479         for (int c=0 ; c<numChannels ; c++) writeSample(sampleBuffer[c][ offset ]);
480
481         offset ++;
482         frameCounter ++;
483     }
484
485     return numFramesToWrite;
486 }
487
488 // Long
489 // -----
490 public int readFrames(long[] sampleBuffer , int numFramesToRead) throws IOException ,
491     ↪ WavFileException
492 {
493     return readFrames(sampleBuffer , 0 , numFramesToRead);
494 }
495
496 public int readFrames(long[] sampleBuffer , int offset , int numFramesToRead) throws
497     ↪ IOException , WavFileException
498 {
499     if (ioState != IOState.READING) throw new IOException("Cannot_read_from_WavFile_instance")
500     ↪ ;
501
502     for (int f=0 ; f<numFramesToRead ; f++)
503     {
504         if (frameCounter == numFrames) return f;
505
506         for (int c=0 ; c<numChannels ; c++)
507         {
508             sampleBuffer[ offset ] = readSample();
509             offset ++;
510         }
511
512         frameCounter ++;
513     }
514
515     return numFramesToRead;
516 }
517
518 public int readFrames(long [][] sampleBuffer , int numFramesToRead) throws IOException ,
519     ↪ WavFileException
520 {
521     return readFrames(sampleBuffer , 0 , numFramesToRead);
522 }
523
524 public int readFrames(long [][] sampleBuffer , int offset , int numFramesToRead) throws
525     ↪ IOException , WavFileException
526 {

```

```

520     if (ioState != IOState.READING) throw new IOException("Cannot_read_from_WavFile_instance")
521     ↪ ;
522     for (int f=0 ; f<numFramesToRead ; f++)
523     {
524         if (frameCounter == numFrames) return f;
525
526         for (int c=0 ; c<numChannels ; c++) sampleBuffer[c][offset] = readSample();
527
528         offset ++;
529         frameCounter ++;
530     }
531
532     return numFramesToRead;
533 }
534
535 public int writeFrames(long [] sampleBuffer, int numFramesToWrite) throws IOException,
536     ↪ WavFileException
537 {
538     return writeFrames(sampleBuffer, 0, numFramesToWrite);
539 }
540
541 public int writeFrames(long [] sampleBuffer, int offset, int numFramesToWrite) throws
542     ↪ IOException, WavFileException
543 {
544     if (ioState != IOState.WRITING) throw new IOException("Cannot_write_to_WavFile_instance");
545
546     for (int f=0 ; f<numFramesToWrite ; f++)
547     {
548         if (frameCounter == numFrames) return f;
549
550         for (int c=0 ; c<numChannels ; c++)
551         {
552             writeSample(sampleBuffer[ offset ]);
553             offset ++;
554         }
555
556         frameCounter ++;
557     }
558
559     return numFramesToWrite;
560 }
561
562 public int writeFrames(long [][] sampleBuffer, int numFramesToWrite) throws IOException,
563     ↪ WavFileException
564 {
565     return writeFrames(sampleBuffer, 0, numFramesToWrite);
566 }
567
568 public int writeFrames(long [][] sampleBuffer, int offset, int numFramesToWrite) throws
569     ↪ IOException, WavFileException
570 {
571     if (ioState != IOState.WRITING) throw new IOException("Cannot_write_to_WavFile_instance");
572
573     for (int f=0 ; f<numFramesToWrite ; f++)
574     {
575         if (frameCounter == numFrames) return f;
576
577         for (int c=0 ; c<numChannels ; c++) writeSample(sampleBuffer[c][ offset ]);
578
579         offset ++;
580         frameCounter ++;
581     }
582
583     return numFramesToWrite;
584 }
585
586 // Double
587 // -----
588 public int readFrames(double [] sampleBuffer, int numFramesToRead) throws IOException,
589     ↪ WavFileException
590 {
591     return readFrames(sampleBuffer, 0, numFramesToRead);
592 }
593
594 public int readFrames(double [] sampleBuffer, int offset, int numFramesToRead) throws

```

```

590     ↪ IOException, WavFileException
591 {
592     if (ioState != IOState.READING) throw new IOException("Cannot_read_from_WavFile_instance")
593     ↪ ;
594
595     for (int f=0 ; f<numFramesToRead ; f++)
596     {
597         if (frameCounter == numFrames) return f;
598
599         for (int c=0 ; c<numChannels ; c++)
600         {
601             sampleBuffer[offset] = floatOffset + (double) readSample() / floatScale;
602             offset ++;
603         }
604
605         frameCounter ++;
606     }
607
608     return numFramesToRead;
609 }
610
611 public int readFrames(double [][] sampleBuffer, int numFramesToRead) throws IOException,
612     ↪ WavFileException
613 {
614     return readFrames(sampleBuffer, 0, numFramesToRead);
615 }
616
617 public int readFrames(double [][] sampleBuffer, int offset, int numFramesToRead) throws
618     ↪ IOException, WavFileException
619 {
620     if (ioState != IOState.READING) throw new IOException("Cannot_read_from_WavFile_instance")
621     ↪ ;
622
623     for (int f=0 ; f<numFramesToRead ; f++)
624     {
625         if (frameCounter == numFrames) return f;
626
627         for (int c=0 ; c<numChannels ; c++) sampleBuffer[c][offset] = floatOffset + (double)
628     ↪ readSample() / floatScale;
629
630         offset ++;
631         frameCounter ++;
632     }
633
634     return numFramesToRead;
635 }
636
637 public int readFramesWithOverlap(double [] sampleBuffer, int numFramesToRead, int overlap)
638     ↪ throws IOException, WavFileException
639 {
640     numFramesToRead = readFrames(sampleBuffer, 0, numFramesToRead);
641     long coeff = frameCounter * overlap / numFramesToRead - overlap + 1;
642     frameCounter = coeff * numFramesToRead / overlap;
643
644     return numFramesToRead;
645 }
646
647 public int writeFrames(double [] sampleBuffer, int numFramesToWrite) throws IOException,
648     ↪ WavFileException
649 {
650     return writeFrames(sampleBuffer, 0, numFramesToWrite);
651 }
652
653 public int writeFrames(double [] sampleBuffer, int offset, int numFramesToWrite) throws
654     ↪ IOException, WavFileException
655 {
656     if (ioState != IOState.WRITING) throw new IOException("Cannot_write_to_WavFile_instance");
657
658     for (int f=0 ; f<numFramesToWrite ; f++)
659     {
660         if (frameCounter == numFrames) return f;
661
662         for (int c=0 ; c<numChannels ; c++)
663         {
664             writeSample(((long) (floatScale * (floatOffset + sampleBuffer[offset]))));

```



```

657         offset ++;
658     }
659
660     frameCounter ++;
661 }
662
663 return numFramesToWrite;
664 }
665
666 public int writeFrames(double [][] sampleBuffer, int numFramesToWrite) throws IOException,
667     ↪ WavFileException
668 {
669     return writeFrames(sampleBuffer, 0, numFramesToWrite);
670 }
671
672 public int writeFrames(double [][] sampleBuffer, int offset, int numFramesToWrite) throws
673     ↪ IOException, WavFileException
674 {
675     if (ioState != IOState.WRITING) throw new IOException("Cannot_write_to_WavFile_instance");
676
677     for (int f=0 ; f<numFramesToWrite ; f++)
678     {
679         if (frameCounter == numFrames) return f;
680
681         for (int c=0 ; c<numChannels ; c++) writeSample((long) (floatScale * (floatOffset +
682     ↪ sampleBuffer[c][offset])));
683
684         offset ++;
685         frameCounter ++;
686     }
687
688     return numFramesToWrite;
689 }
690
691 public void close() throws IOException
692 {
693     // Close the input stream and set to null
694     if (iStream != null)
695     {
696         iStream.close();
697         iStream = null;
698     }
699
700     if (oStream != null)
701     {
702         // Write out anything still in the local buffer
703         if (bufferPointer > 0) oStream.write(buffer, 0, bufferPointer);
704
705         // If an extra byte is required for word alignment, add it to the end
706         if (wordAlignAdjust) oStream.write(0);
707
708         // Close the stream and set to null
709         oStream.close();
710         oStream = null;
711     }
712
713     // Flag that the stream is closed
714     ioState = IOState.CLOSED;
715 }
716
717 public void display()
718 {
719     display(System.out);
720 }
721
722 public void display(PrintStream out)
723 {
724     out.printf("File:_%s\n", file);
725     out.printf("Channels:_%d, Frames:_%d\n", numChannels, numFrames);
726     out.printf("IO_State:_%s\n", ioState);
727     out.printf("Sample_Rate:_%d, Block_Align:_%d\n", sampleRate, blockAlign);
728     out.printf("Valid_Bits:_%d, Bytes_per_sample:_%d\n", validBits, bytesPerSample);
729 }
730
731 public static void main(String[] args)

```

```

730 {
731     if (args.length < 1)
732     {
733         System.err.println("Must supply filename");
734         System.exit(1);
735     }
736
737     try
738     {
739         for (String filename : args)
740         {
741             WavFile readWavFile = openWavFile(new File(filename));
742             readWavFile.display();
743
744             long numFrames = readWavFile.getNumFrames();
745             int numChannels = readWavFile.getNumChannels();
746             int validBits = readWavFile.getValidBits();
747             long sampleRate = readWavFile.getSampleRate();
748
749             WavFile writeWavFile = newWavFile(new File("out.wav"), numChannels, numFrames,
750             ↪ validBits, sampleRate);
751
752             final int BUF_SIZE = 5001;
753
754             // int[] buffer = new int[BUF_SIZE * numChannels];
755             // long[] buffer = new long[BUF_SIZE * numChannels];
756             double[] buffer = new double[BUF_SIZE * numChannels];
757
758             int framesRead = 0;
759             int framesWritten = 0;
760
761             do
762             {
763                 framesRead = readWavFile.readFrames(buffer, BUF_SIZE);
764                 framesWritten = writeWavFile.writeFrames(buffer, BUF_SIZE);
765                 System.out.printf("%d_%d\n", framesRead, framesWritten);
766             }
767             while (framesRead != 0);
768
769             readWavFile.close();
770             writeWavFile.close();
771         }
772
773         WavFile writeWavFile = newWavFile(new File("out2.wav"), 1, 10, 23, 44100);
774         double[] buffer = new double[10];
775         writeWavFile.writeFrames(buffer, 10);
776         writeWavFile.close();
777     }
778     catch (Exception e)
779     {
780         System.err.println(e);
781         e.printStackTrace();
782     }
783 }

```

Листинг 3: WavFileException.java

```

1 package com.external;
2
3 /**
4  * Exception for WavFile Class
5  * http://www.labbookpages.co.uk
6  *
7  * @author A.Greensted
8  */
9
10 public class WavFileException extends Exception
11 {
12     public WavFileException()
13     {
14         super();
15     }
16
17     public WavFileException(String message)
18     {

```

```

19     super(message);
20 }
21
22 public WavFileException(String message, Throwable cause)
23 {
24     super(message, cause);
25 }
26
27 public WavFileException(Throwable cause)
28 {
29     super(cause);
30 }
31 }

```

ЛИСТИНГ 4: LightLevel.java

```

1 package com.sunradio.core;
2
3 import com.sunradio.math.Scale;
4
5 import static java.lang.Math.*;
6 import java.util.Random;
7
8 /**
9  * LightLevel describes how we get an array of data with current light level
10  * @author V.Kremneva
11  */
12 public class LightLevel {
13
14     /**
15      * Get randomised light level
16      *
17      * @param amount of how many measurement of light level we want
18      * @return scaled to [0;1] array of light values
19      */
20     private static double[] getFakeLightLevel(int amount, Double minAmplitude, Double
21 ↪ maxAmplitude) {
22
23         //fake it 'till you make it
24         Integer[] lightLevel = new Integer[amount];
25         for (int i = 0; i < amount; i++)
26             lightLevel[i] = (int)(sin(2 * PI * i / 100)*1000); //for smoothness
27
28         return Scale.run(lightLevel, minAmplitude, maxAmplitude);
29     }
30
31     /**
32      * Get light level from Arduino //TODO: specify
33      *
34      * @param amount of how many measurement of light level we want
35      * @return scaled to [0;1] array of light values
36      */
37     static double[] getLightLevel(int amount, Double minAmplitude, Double maxAmplitude) {
38         return getFakeLightLevel(amount, minAmplitude, maxAmplitude);
39     }
40
41     /**
42      * Get light level
43      * @param values an array to whom get light level
44      * @return light level for 'values'
45      */
46     public static double[] getLightLevel(double[] values) {
47         double maxVal, minVal;
48         maxVal = Double.MIN_VALUE; minVal = Double.MAX_VALUE;
49         for (double val: values) {
50             if (maxVal > val) maxVal = val;
51             if (maxVal < val) minVal = val;
52         }
53
54         return getLightLevel(values.length, minVal, maxVal);
55     }
56
57     private static int getAverageFakeLevel() {
58         Random random = new Random();
59         return random.nextInt();

```

```

60     }
61
62     static int getAverageLightLevel(double[] values) {
63         return getAverageFakeLevel();
64     }
65 }

```

Листинг 5: SunRadio.java

```

1 package com.sunradio.core;
2
3 import com.external.WavFile;
4 import com.sunradio.math.DFTInverse;
5 import com.sunradio.math.DFTStraight;
6 import com.sunradio.math.Filter;
7 import com.sunradio.math.ToneModulation;
8
9 import java.io.File;
10 /*
11 import javax.sound.sampled.AudioInputStream;
12 import javax.sound.sampled.AudioSystem;
13 import javax.sound.sampled.Clip;
14 */
15
16 /**
17  * @author V.Kremneva
18  */
19 public class SunRadio {
20     private final int FRAMES = 2048; //amount of frames to read
21     private final int OVERLAP = 16; //coefficient of overlap
22
23     private WavFile wavInput; //input file
24     private WavFile wavOutput; //output file
25
26     private int bufferIndAmount; //amount of indexes in 'buffer' array needed to read data to
27     private int overlapIndAmount; //amount of indexes to work with overlap
28     private int offset; //amount of new frames read in each step of cycle
29     private int outputBufferIndAmount; //amount of indexes in am array to write
30     private long wholeIndAmount; //amount of pieces to read in whole file
31
32     /**
33      * Open file and set some fields depending on it
34      *
35      * @param inputPath path to file to open
36      */
37     private void openWavFile(String inputPath) {
38         try {
39
40             wavInput = WavFile.openWavFile(new File(inputPath));
41
42         } catch (Exception e) {
43             System.err.println(e.toString());
44         }
45
46         int numChannels = wavInput.getNumChannels();
47         bufferIndAmount = FRAMES * numChannels;
48         overlapIndAmount = OVERLAP * numChannels;
49         offset = FRAMES / OVERLAP;
50         outputBufferIndAmount = bufferIndAmount + overlapIndAmount;
51         wholeIndAmount = wavInput.getNumFrames() * numChannels;
52     }
53
54     /**
55      * Create empty output file like input file but stretched
56      *
57      * @param outputPath path where to create output file
58      */
59     private void createStretchedOutputFile(String outputPath, int stretch) {
60         try {
61
62             wavOutput = WavFile.newWavFile(new File(outputPath),
63                 wavInput.getNumChannels(), wavInput.getNumFrames() * stretch,
64                 wavInput.getValidBits(), wavInput.getSampleRate());
65
66         } catch (Exception e) {
67             System.err.println(e.toString());

```

```

68     }
69 }
70
71 private void run() {
72     double[] buffer = new double[bufferIndAmount];
73     double[] outputBuffer = new double[outputBufferIndAmount];
74     double[] outputWindowFunction;
75     int lightLevel, frames_read;
76     DFTStraight transformable;
77     ToneModulation toneModulation;
78
79     transformable = new DFTStraight();
80     toneModulation = new ToneModulation(bufferIndAmount);
81
82     int counter = 0;
83     try {
84         do {
85             //read next 'FRAMES' into buffer — amplitudes(t)
86             frames_read = wavInput.readFramesWithOverlap(buffer, FRAMES, OVERLAP);
87
88             //get current level of light
89             lightLevel = LightLevel.getAverageLightLevel(buffer);
90
91             for (int i = 0; i < overlapIndAmount; i++) {
92                 //apply window filter. first and last 'offset' goes without filter
93                 if (!(wavInput.getFrameCounter() == offset) && !(wavInput.getFrameCounter
94 → () == (wholeIndAmount - offset))) {
95                     buffer = Filter.apply(buffer, Filter.BlackmanNuttall(bufferIndAmount))
96 → ;
97                 }
98
99                 //run Fourier transform
100                transformable.run(buffer);
101
102                //stretch in 'light level' times
103                toneModulation.setCurrentData(transformable);
104                transformable.setData(toneModulation.stretch(lightLevel));
105
106                //run inverse Fourier transform
107                buffer = DFTInverse.run(transformable.getData());
108
109                //apply output window function
110                outputWindowFunction = Filter.getOutputWindowFunc(Filter.BlackmanNuttall(
111 → bufferIndAmount));
112                buffer = Filter.apply(buffer, outputWindowFunction);
113
114                for (int j = 0; j < bufferIndAmount; j++)
115                    outputBuffer[j + i] += buffer[j];
116
117                //read next 'FRAMES' into buffer — amplitudes(t)
118                frames_read = wavInput.readFramesWithOverlap(buffer, FRAMES, OVERLAP);
119            }
120
121            //write data to new .wav file
122            wavOutput.writeFrames(outputBuffer, FRAMES);
123
124            //move data for overlap
125            outputBuffer = move(outputBuffer, overlapIndAmount);
126
127            toneModulation.setPreviousData(transformable);
128            counter++;
129        } while (frames_read != 0);
130
131        //todo: adjust volume
132        //todo: fasten velocity of playback
133        //play(outputPath);
134
135    } catch (Exception e) {
136        System.err.println(e.toString());
137    }
138 }
139
140 private void closeFiles() {
141     try {

```

```

141         wavInput.close();
142         wavOutput.close();
143
144     } catch (Exception e) {
145         System.err.println(e.toString());
146     }
147 }
148
149 /**
150  * Move data to the left with filling with 0
151  * @param data data to move
152  * @param offset amount of steps to move
153  * @return array with nulls in the end and 'data' values moved on offset
154  */
155 public static double[] move(double[] data, int offset) {
156     double[] result = new double[data.length];
157
158     System.arraycopy(data, offset, result, 0, data.length - offset);
159
160     return result;
161 }
162
163 /*private static void play(String pathname) {
164     try {
165         Clip c = AudioSystem.getClip();
166         AudioInputStream ais = AudioSystem.getAudioInputStream(new File(pathname));
167
168         c.open(ais);
169         c.loop(0);
170
171         Thread.sleep(1000);
172     } catch (Exception e) {
173         System.err.println(e.toString());
174     }
175 }*/
176
177 public static void main(String[] args) {
178
179     if (args.length < 2) throw new IllegalArgumentException("As the arguments of the
180     ↪ program " +
181         "input path and output path are needed.");
182
183     SunRadio radio = new SunRadio();
184
185     radio.openWavFile(args[0]);
186
187     radio.run();
188
189     radio.closeFiles();
190 }

```

Листинг 6: AM.java

```

1 package com.sunradio.math;
2
3 /**
4  * Amplitude modulation
5  * @author V.Kremneva
6  */
7 public class AM {
8
9     /**
10     * Modulate values according to conditions with the coefficient of modulation -0.65.
11     * Assume 'values' as amplitude values therefore perform an Amplitude Modulation.
12     *
13     * @param values an array with amplitude values to modulate
14     * @param conditions an array of modulation conditions for each amplitude value
15     * @return a double array of modulated values of amplitudes
16     * @throws IllegalArgumentException if amount of conditions is less than amount of values
17     */
18     public static double[] modulate(double[] values, double[] conditions) {
19         return modulate(values, conditions, -0.65);
20     }
21
22     /**

```

```

23     * Modulate values according to conditions.
24     * Assume 'values' as amplitude values therefore perform an Amplitude Modulation.
25     *
26     * @param values an array with amplitude values to modulate
27     * @param conditions an array of modulation conditions for each amplitude value
28     * @param modulationCoeff modulation coefficient which picks up by trial and error
29     * @return a double array of modulated values of amplitudes
30     * @throws IllegalArgumentException if amount of conditions is less than amount of values
31     */
32     private static double[] modulate(double[] values, double[] conditions, double
    ↪ modulationCoeff)
33         throws IllegalArgumentException {
34
35         if (values.length > conditions.length) {
36             throw new IllegalArgumentException("A_size_of_an_array_with_conditions_" +
37                 "should_be_equal_or_more_than_size_of_an_array_of_values.\n" +
38                 "Conditions_size_" + conditions.length + ".Values_size_" + values.
    ↪ length);
39         }
40
41         double[] modulated;
42         modulated = new double[values.length];
43
44         double maxCond;
45         maxCond = conditions[0];
46         for (int i = 1; i < values.length; i++)
47             if (conditions[i] > maxCond) maxCond = conditions[i];
48
49         for (int i = 0; i < values.length; i++)
50             modulated[i] = values[i] * (1 + modulationCoeff * conditions[i] / Math.abs(maxCond
    ↪ ));
51
52         return modulated;
53     }
54 }

```

Листинг 7: DFTStraight.java

```

1 package com.sunradio.math;
2
3 import com.external.Complex;
4 import java.util.Arrays;
5 import static java.lang.Math.*;
6
7 /**
8  * Straight discrete Fourier transform.
9  *
10  * @author V.Kremneva
11  */
12 public class DFTStraight {
13
14     private Complex[] data; //Complex data
15     private double maxAmplitude; //maximum value of real amplitude in 'data' array
16     private double minAmplitude; //minimum value of real amplitude in 'data' array
17     private int size; //size of 'data' array
18     private boolean isTransformed; //flag whether was 'data' transformed by DFT or not
19
20     public DFTStraight() {
21         maxAmplitude = 0.0;
22         minAmplitude = 0.0;
23         size = 0;
24         isTransformed = false;
25     }
26
27     public double getMaxAmplitude() {
28         return maxAmplitude;
29     }
30
31     public double getMinAmplitude() {
32         return minAmplitude;
33     }
34
35     public boolean isTransformed() {
36         return isTransformed;
37     }
38 }

```

```

39 public Complex[] getData() {
40     return data;
41 }
42
43 public int getSize() {
44     return size;
45 }
46
47 /** Get a double phases from 'data' array.
48  *
49  * @return a double array which contains phase values
50  */
51 public double[] getPhases() {
52     double[] phases = new double[size];
53     double allowance;
54     for (int i = 0; i < size; i++) {
55
56         if (data[i].re() > 0) allowance = 0;
57         else if (data[i].im() > 0) allowance = PI;
58         else allowance = -PI;
59
60         phases[i] = allowance + atan(data[i].im() / data[i].re());
61     }
62
63     return phases;
64 }
65
66 /** Get a phase value on specific harmonic
67  *
68  * @param n frequency value of harmonic
69  * @return double value of phase of harmonic
70  */
71 public double getPhase(int n) {
72     double allowance;
73     if (data[n].re() > 0) allowance = 0;
74     else if (data[n].im() > 0) allowance = PI;
75     else allowance = -PI;
76
77     return allowance + atan(data[n].im() / data[n].re());
78 }
79
80 public static double getPhase(Complex[] data, int n) {
81     double allowance;
82     if (data[n].re() > 0) allowance = 0;
83     else if (data[n].im() > 0) allowance = PI;
84     else allowance = -PI;
85
86     return allowance + atan(data[n].im() / data[n].re());
87 }
88
89 /** Get a double amplitudes from 'data' array.
90  *
91  * @return a double array which contains amplitude values
92  */
93 public double[] getAmplitudes() {
94     double[] amplitudes = new double[size];
95
96     for (int i = 0; i < size; i++)
97         amplitudes[i] = data[i].abs() / ((size - 1) * 2); // '-1)*2' due to cutting in
98 ↪ half
99
100     return amplitudes;
101 }
102
103 static double[] getAmplitudes(Complex[] data) {
104     double[] amplitudes = new double[data.length];
105
106     for (int i = 0; i < data.length; i++)
107         amplitudes[i] = data[i].abs() / ((data.length - 1) * 2); // '-1)*2' due to cutting
108 ↪ in half
109
110     return amplitudes;
111 }
112
113 static double[] getPhases(Complex[] data) {
114     double[] phases = new double[data.length];

```



```

113     double allowance;
114     for (int i = 0; i < phases.length; i++) {
115
116         if (data[i].re() > 0) allowance = 0;
117         else if (data[i].im() > 0) allowance = PI;
118         else allowance = -PI;
119
120         phases[i] = allowance + atan(data[i].im() / data[i].re());
121     }
122
123     return phases;
124 }
125
126 /** Get an amplitude value on specific harmonic
127  *
128  * @param n frequency value of harmonic
129  * @return double value of amplitude of harmonic
130  */
131 public double getAmplitude(int n) {
132     return data[n].abs() / ((size - 1) * 2);
133     // '-1)*2' due to cutting in half
134 }
135
136 public void setData(Complex[] newData) {
137     size = newData.length;
138     data = Arrays.copyOf(newData, size);
139
140     double max, min;
141     max = Double.MIN_VALUE; min = Double.MAX_VALUE;
142     for (int i = 0; i < size; i++) {
143         if (this.getAmplitude(i) > max) max = this.getAmplitude(i);
144         if (this.getAmplitude(i) < min) min = this.getAmplitude(i);
145     }
146     maxAmplitude = max; minAmplitude = min;
147 }
148
149 /**
150  * Cut Complex array in two pieces.
151  * We need this because the periods of the input data become split into "positive"
152  * and "negative" frequency complex components. As a result, only half of array
153  * contains data we are interested in and the rest of array is just a reflection with
154  * opposite sign.
155  *
156  * @param dataToCut Complex data we want to be cut
157  */
158 private void cutDataInHalf(Complex[] dataToCut) {
159     size = size / 2 + 1; //'+1' to include center value
160
161     data = new Complex[size];
162     data = Arrays.copyOf(dataToCut, size);
163 }
164
165 /** Run transform with search of max and min value of amplitude.
166  *
167  * @param buffer an array of the magnitudes
168  * @return a Complex array which contains amplitude and phase values
169  * @throws IllegalArgumentException if buffer is empty
170  */
171 public Complex[] run(double[] buffer) throws IllegalArgumentException {
172
173     size = buffer.length;
174     Double realAmplitude;
175     Complex cBuffer, expDegree, tempData[];
176     tempData = new Complex[size];
177
178     if (size == 0) throw new IllegalArgumentException("Size of a buffer cannot be < 1.\n↪ size=" + size);
179
180     maxAmplitude = Double.MIN_VALUE;
181     minAmplitude = Double.MAX_VALUE;
182     for (int i = 0; i < size; i++) {
183         tempData[i] = new Complex(0, 0);
184
185         for (int j = 0; j < size; j++) {
186             cBuffer = new Complex(buffer[j]);
187             expDegree = new Complex(0, -2 * PI * j * i / size);

```

```

188         tempData[i] = tempData[i].add(cBuffer.mult(expDegree.exp()));
189     }
190
191     realAmplitude = tempData[i].abs() / size;
192     if (realAmplitude > maxAmplitude) maxAmplitude = realAmplitude;
193     if (realAmplitude < minAmplitude) minAmplitude = realAmplitude;
194 }
195
196 isTransformed = true;
197
198 cutDataInHalf(tempData);
199
200 return data;
201 }
202
203 /**
204  * Change phase values in 'data' without changing amplitudes
205  *
206  * @param newPhases an array of phase values we want to apply
207  */
208 void applyNewPhases(double[] newPhases) {
209     double a, b, allowance;
210     for (int i = 0; i < size; i++) {
211         if (data[i].re() > 0) allowance = 0;
212         else if (data[i].im() > 0) allowance = -PI;
213         else allowance = PI;
214
215         a = data[i].abs() / sqrt(1 + pow(tan(newPhases[i] + allowance), 2.0));
216         b = a * tan(newPhases[i] + allowance);
217
218         //we get 'b' from equation for phase and 'a' from my condition:
219         //i want the real amplitudes be the same
220
221         data[i] = new Complex(a, b);
222     }
223 }
224
225 static Complex[] applyNewPhases(double[] newPhases, Complex[] oldData) {
226     double a, b, allowance;
227     Complex[] result = new Complex[newPhases.length];
228     for (int i = 0; i < newPhases.length; i++) {
229         if (oldData[i].re() > 0) allowance = 0;
230         else if (oldData[i].im() > 0) allowance = -PI;
231         else allowance = PI;
232
233         a = oldData[i].abs() / sqrt(1 + pow(tan(newPhases[i] + allowance), 2.0));
234         b = a * tan(newPhases[i] + allowance);
235
236         //we get 'b' from equation for phase and 'a' from my condition:
237         //i want the real amplitudes be the same
238
239         result[i] = new Complex(a, b);
240     }
241     return result;
242 }
243
244 /**
245  * Change amplitude values in 'data' without changing phases
246  *
247  * @param newAmplitudes an array of amplitude values we want to apply
248  */
249 void applyNewAmplitudes(double[] newAmplitudes) {
250     int sign;
251     double a, b;
252     for (int i = 0; i < size; i++) {
253         if (this.getPhase(i) < 0) sign = -1;
254         else sign = 1;
255         b = pow(data[i].im(), 2.0) * pow(newAmplitudes[i], 2.0) * pow(size, 2.0);
256         b = b / (pow(data[i].re(), 2.0) + pow(data[i].im(), 2.0));
257         b = sqrt(b);
258
259         a = sign * sqrt(pow(newAmplitudes[i], 2.0) * pow(size, 2.0) - pow(b, 2.0));
260
261         //we get 'a' from equation for real amplitude and 'b' from my condition:
262         //i want the real phase be the same
263

```

```

264         data[i] = new Complex(a, b);
265     }
266 }
267
268 static Complex[] applyNewAmplitudes(double[] newAmplitudes, Complex[] oldData) {
269     int sign;
270     double a, b;
271     Complex[] result = new Complex[newAmplitudes.length];
272     for (int i = 0; i < newAmplitudes.length; i++) {
273         if (getPhase(oldData, i) < 0) sign = -1;
274         else sign = 1;
275         b = pow(oldData[i].im(), 2.0) * pow(newAmplitudes[i], 2.0) * pow(newAmplitudes.
↪ length, 2.0);
276         b = b / (pow(oldData[i].re(), 2.0) + pow(oldData[i].im(), 2.0));
277         b = sqrt(b);
278
279         a = sign * sqrt(pow(newAmplitudes[i], 2.0) * pow(newAmplitudes.length, 2.0) - pow(
↪ b, 2.0));
280
281         //we get 'a' from equation for real amplitude and 'b' from my condition:
282         //i want the real phase be the same
283
284         result[i] = new Complex(a, b);
285     }
286
287     return result;
288 }
289 }

```

Листинг 8: DFTInverse.java

```

1 package com.sunradio.math;
2
3 import com.external.Complex;
4
5 import java.util.Arrays;
6
7 import static java.lang.Math.PI;
8
9 /**
10  * Inverse discrete Fourier transform.
11  * @author V.Kremneva
12  */
13 public class DFTInverse {
14
15     /**
16      * Restore frequency complex components from just one half.
17      *
18      * @param dataToRestore array of data to restore
19      * @return full spectrum Complex array
20      */
21     private static Complex[] restoreData(Complex[] dataToRestore) {
22
23         int oldSize = dataToRestore.length;
24         int newSize = oldSize * 2 - 2;
25         Complex[] result = new Complex[newSize];
26
27         System.arraycopy(dataToRestore, 0, result, 0, oldSize);
28
29         for (int i = oldSize; i < newSize; i++)
30             result[i] = dataToRestore[newSize - i].conj();
31
32         return result;
33     }
34
35     /** Run transform.
36      *
37      * @param transformed a Complex array which contains amplitude and phase values
38      * @return an array of amplitudes
39      */
40     public static double[] run(Complex[] transformed){
41
42         Complex[] data;
43         data = restoreData(transformed);
44
45         int size = data.length;

```

```

46     Complex exp_degree, magnitude;
47     double[] result = new double[size];
48
49     for (int i = 0; i < size; i++) {
50         magnitude = new Complex(0, 0);
51
52         for (int j = 0; j < size; j++) {
53             exp_degree = new Complex(0, 2 * PI * j * i / size);
54             magnitude = magnitude.add(data[j].mult(exp_degree.exp()));
55         }
56
57         result[i] = magnitude.re() / size;
58     }
59
60     return result;
61 }
62 }

```

Листинг 9: Filter.java

```

1 package com.sunradio.math;
2
3 import static java.lang.Math.*;
4
5 /**
6  * Filter functions
7  * @author V.Kremneva
8  */
9 public class Filter {
10
11     /**
12      * Window function of Blackman–Nuttall
13      *
14      * @param size is the length of array to be windowed
15      * @return an array of values of this window
16      */
17     public static double[] BlackmanNuttall(int size) {
18         double[] result = new double[size];
19
20         //constants from formula of -BlackmanNuttall window
21         final double A0 = 0.3635819;
22         final double A1 = 0.4891775;
23         final double A2 = 0.1365995;
24         final double A3 = 0.0106411;
25
26         double firstCos, secondCos, thirdCos;
27         for (int i = 0; i < size; i++) {
28             firstCos = A1 * cos((2 * PI * i) / (size - 1));
29             secondCos = A2 * cos((4 * PI * i) / (size - 1));
30             thirdCos = A3 * cos((6 * PI * i) / (size - 1));
31
32             result[i] = A0 - firstCos + secondCos - thirdCos;
33         }
34
35         return result;
36     }
37
38     /**
39      * Window function of Blackman–Nuttall
40      *
41      * @param n point where value of this function is needed
42      * @param size is the length of array to be windowed
43      * @return value of this window function in 'n'
44      */
45     public static double BlackmanNuttall(int n, int size) {
46         //constants from formula of -BlackmanNuttall window
47         final double A0 = 0.3635819;
48         final double A1 = 0.4891775;
49         final double A2 = 0.1365995;
50         final double A3 = 0.0106411;
51
52         double firstCos, secondCos, thirdCos;
53         firstCos = A1 * cos((2 * PI * n) / (size - 1));
54         secondCos = A2 * cos((4 * PI * n) / (size - 1));
55         thirdCos = A3 * cos((6 * PI * n) / (size - 1));
56

```

```

57         return A0 - firstCos + secondCos - thirdCos;
58     }
59
60     /**
61     * Apply window function to an array of values
62     *
63     * @param amplitudes an array to be windowed
64     * @param windowFunc an array with values of window function
65     * @return an array with transformed 'amplitudes' according to 'windowFunc'
66     * @throws IllegalArgumentException if length of 'windowFunc' is less than length of '
    ↪ amplitudes'
67     */
68     public static double[] apply(double[] amplitudes, double[] windowFunc)
69         throws IllegalArgumentException {
70
71         int size = amplitudes.length;
72
73         if (size > windowFunc.length) throw new IllegalArgumentException("Length of window_
    ↪ function is too small");
74
75         double[] result = new double[size];
76         for (int i = 0; i < size; i++) {
77             result[i] = amplitudes[i] * windowFunc[i];
78         }
79         return result;
80     }
81
82     /**
83     * Remove noise from amplitudes array
84     *
85     * @param toDenoise array to remove noise from
86     * @return sort of clean array
87     */
88     public static double[] denoise(double[] toDenoise) {
89         final double DENOISE_COEFF = 0.1; //got it by trial and error
90         double maxAmplitude, eps;
91
92         maxAmplitude = toDenoise[0];
93         for (double val: toDenoise)
94             if (val > maxAmplitude) maxAmplitude = val;
95
96         eps = maxAmplitude * DENOISE_COEFF;
97         for (int i = 0; i < toDenoise.length; i++)
98             if (abs(toDenoise[i]) < eps) toDenoise[i] = 0.0;
99
100         return toDenoise;
101     }
102
103     /**
104     * Get function to filter output data before writing to file
105     *
106     * @param inputWindowFunc function used to filter input data
107     * @return function to filter output data
108     */
109     public static double[] getOutputWindowFunc(double[] inputWindowFunc) {
110         double[] outputWindowFunc = new double[inputWindowFunc.length];
111         double sum = 0.0;
112
113         for (double val: inputWindowFunc)
114             sum += val * val;
115
116         for (int i = 0; i < outputWindowFunc.length; i++)
117             outputWindowFunc[i] = inputWindowFunc[i] / sum;
118
119         return outputWindowFunc;
120     }
121 }

```

Листинг 10: Interpolation.java

```

1 package com.sunradio.math;
2
3 /**
4  * Class to interpolate.
5  *
6  * @author V.Kremneva

```

```

7  */
8  class Interpolation {
9      /**
10       * Perform linear interpolation by two points
11       * @param x0 x from the first point
12       * @param y0 f(x) from the first point
13       * @param x1 x from the second point
14       * @param y1 f(x) from the second point
15       * @param x x from the point we are interested in
16       * @return f(x) from the point we are interested in
17       */
18       static double linearByX(double x0, double y0, double x1, double y1, double x) {
19           return y0 + (x - x0) * (y1 - y0) / (x1 - x0);
20       }
21
22       /**
23       * Perform linear interpolation by two points
24       * @param x0 x from the first point
25       * @param y0 f(x) from the first point
26       * @param x1 x from the second point
27       * @param y1 f(x) from the second point
28       * @param y f(x) from the point we are interested in
29       * @return x from the point we are interested in
30       */
31       static double linearByY(double x0, double y0, double x1, double y1, double y) {
32           return x0 + (y - y0) * (x1 - x0) / (y1 - y0);
33       }
34 }

```

Листинг 11: Scale.java

```

1  package com.sunradio.math;
2
3  /**
4   * Helps to scale data
5   * @author V.Kremneva
6   */
7  public class Scale {
8
9      /**
10       * Scale numeric data
11       *
12       * @param arr is an array of data to scale
13       * @param from is the lower bound of scaling
14       * @param to is the upper bound of scaling
15       * @return scaled array of data
16       * @throws IllegalArgumentException if the data array is empty
17       */
18       public static<T extends java.lang.Number> double[] run(T[] arr, double from, double to)
19           throws IllegalArgumentException {
20
21           int size = arr.length;
22           if (size < 1) throw new IllegalArgumentException("Array_of_the_values_cannot_be_empty"
23 → );
24           if (from > to) {
25               double temp = from;
26               from = to; to = temp;
27           }
28
29           double maxLevel = arr[0].doubleValue();
30           double minLevel = arr[0].doubleValue();
31           double current;
32           for (int i = 1; i < size; i++) {
33               current = arr[i].doubleValue();
34
35               if (current > maxLevel) maxLevel = current;
36               else if (current < minLevel) minLevel = current;
37           }
38
39           double step = (to - from) / (maxLevel - minLevel);
40
41           double [] scaledLightLevel = new double [size];
42           for (int i = 0; i < size; i++)
43               scaledLightLevel[i] = (arr[i].doubleValue() - minLevel) * step + from;
44
45           return scaledLightLevel;
46 }

```

```

45     }
46 }

```

Листинг 12: ToneModulation.java

```

1 package com.sunradio.math;
2
3 import com.external.Complex;
4
5 /**
6  * Tone modulation.
7  *
8  * @author V.Kremneva
9  */
10 public class ToneModulation {
11     private double[] previousPhases; //values of previous phases
12     private double[] previousAmplitudes; //values of previous amplitudes
13     private double[] currentPhases; //values of current phases
14     private double[] currentAmplitudes; //values of current amplitudes
15
16     private int length; //length of all of the arrays
17
18     private boolean previousIsSet; //indicates whether previous data was set
19     private boolean currentIsSet; //indicates whether current data was set
20
21     public ToneModulation() {
22         previousIsSet = false;
23         currentIsSet = false;
24     }
25
26     public ToneModulation(int size) {
27         length = size;
28
29         previousPhases = new double[size];
30         previousAmplitudes = new double[size];
31
32         previousIsSet = true;
33         currentIsSet = false;
34     }
35
36     public void setPreviousData(DFTStraight prevData) {
37         previousAmplitudes = prevData.getAmplitudes();
38         previousPhases = prevData.getPhases();
39
40         previousIsSet = true;
41     }
42
43     public void setCurrentData(DFTStraight currentData) {
44         currentAmplitudes = currentData.getAmplitudes();
45         currentPhases = currentData.getPhases();
46
47         currentIsSet = true;
48     }
49
50     /**
51     * Stretch data in N times
52     *
53     * @param coefficient in how many times to stretch
54     * @return stretched data
55     */
56     public Complex[] stretch(int coefficient) throws ToneModulationException {
57         if (!previousIsSet || !currentIsSet)
58             throw new ToneModulationException("Previous_and_current_data_must_be_set");
59
60         int newSize = length * coefficient;
61         Complex[] result = new Complex[newSize];
62         double[] newPhases = new double[newSize];
63         double[] newAmplitudes = new double[newSize];
64         double[] velocity = new double[length];
65
66         for (int i = 0; i < length; i++)
67             velocity[i] = currentPhases[i] - previousPhases[i];
68
69         for (int i = 0; i < length; i++)
70             for (int j = 0; j < coefficient; j++)
71                 newPhases[i + j] = currentPhases[i] + velocity[i];

```

```

72
73     result = DFTStraight.applyNewPhases(newPhases, result);
74
75     for (int i = 0; i < length; i++)
76         for (int j = 0; j < coefficient; j++)
77             newAmplitudes[i + j] = Interpolation.linearByX(0, previousAmplitudes[i],
78 ↪ coefficient, currentAmplitudes[i], j);
79
80     result = DFTStraight.applyNewAmplitudes(newAmplitudes, result);
81
82     return result;
83 }

```

Листинг 13: ToneModulationException.java

```

1 package com.sunradio.math;
2
3 /**
4  * Exception for ToneModulation class
5  *
6  * @author V.Kremneva
7  */
8 public class ToneModulationException extends Exception {
9
10     public ToneModulationException(String message) {
11         super(message);
12     }
13 }

```

Листинг 14: SunRadioTest.groovy

```

1 package com.sunradio.core
2
3 import com.external.WavFile
4 import com.sunradio.math.AM
5 import com.sunradio.math.DFTInverse
6 import com.sunradio.math.DFTStraight
7 import com.sunradio.math.Filter
8
9 class SunRadioTest extends GroovyTestCase {
10     final FRAMES = 100
11     final EPS = 0.00001
12     DFTStraight transformable = new DFTStraight()
13
14     int numChannels, indAmount, framesRead
15     double[] buffer, lightLevel, modulated, amplitudes
16
17
18     void testRun() {
19         try {
20             // inputPath = "C:\\Users\\LEV\\IdeaProjects\\SunRadio\\launch.wav";
21             // outputPath = "C:\\Users\\LEV\\IdeaProjects\\SunRadio\\new1.wav";
22
23         } catch (Exception e) {
24             System.err.println(e.toString())
25         }
26     }
27
28     void testMove() {
29         int offset = 3
30         double[] before = [1, 2, 3, 4, 5, 6, 7]
31         double[] afterExpected = [4, 5, 6, 7, 0, 0, 0]
32         double[] afterActual
33
34         afterActual = SunRadio.move(before, offset)
35
36         for (int i = 0; i < before.length; i++)
37             assertEquals(afterExpected[i], afterActual[i])
38     }
39 }

```

Листинг 15: DFTStraightTest.groovy


```

1 package com.sunradio.math
2
3 import static java.lang.Math.*
4
5 class DFTStraightTest extends GroovyTestCase {
6
7     final EPS = 0.00001
8     final ITERATIONS = 100
9     final SPLIT = 100
10    final NUMBER = 80.0
11
12    double[] buffer = new double[ITERATIONS]
13    double[] result = new double[ITERATIONS]
14    DFTStraight dftStraight = new DFTStraight()
15
16    //f(t) = sin(t)
17    void testSin() {
18
19        //Split the sinus period in SPLIT pieces and take the sinus value in each of them
20        for (int i = 0; i < ITERATIONS; i++)
21            buffer[i] = sin(2 * PI * i / SPLIT)
22
23        dftStraight.run(buffer)
24        result = dftStraight.getAmplitudes()
25
26        int amount = 0, size = dftStraight.getSize()
27        for (int i = 0; i < size; i++)
28            if (result[i] > EPS)
29                amount++
30
31        assertEquals(1, amount)
32    }
33
34    //f(t) = NUMBER*sin(t)
35    //Test passes with any number
36    void testConstMultSin() {
37
38        //Split the sinus period in SPLIT pieces and take the sinus value in each of them
39        for (int i = 0; i < ITERATIONS; i++)
40            buffer[i] = NUMBER * sin(2 * PI * i / SPLIT)
41
42        dftStraight.run(buffer)
43        result = dftStraight.getAmplitudes()
44
45        int amount = 0, size = dftStraight.getSize()
46        for (int i = 0; i < size; i++)
47            if (result[i] > EPS)
48                amount++
49
50        assertEquals(1, amount)
51    }
52
53    //f(t) = sin(NUMBER*t)
54    void testSinMultConst() throws IllegalArgumentException {
55
56        if ((NUMBER > SPLIT) || (NUMBER < SPLIT / 2))
57            throw new IllegalArgumentException("Number_should_be_less_than_SPLIT_and_more_than
58    ↪ SPLIT/2_due_to_sinus_period")
59
60        //Split the sinus period in SPLIT pieces and take the sinus value in each of them
61        for (int i = 0; i < ITERATIONS; i++)
62            buffer[i] = sin(NUMBER * 2 * PI * i / SPLIT)
63
64        dftStraight.run(buffer)
65        result = dftStraight.getAmplitudes()
66
67        int amount = 0, size = dftStraight.getSize()
68        for (int i = 0; i < size; i++)
69            if (result[i] > EPS)
70                amount++
71
72        assertEquals(1, amount)
73    }
74
75    //f(t) = NUMBER*sin(t) + sin(Number*t)
76    void testTwoSinuses() {

```

```

76
77     if ((NUMBER > SPLIT) || (NUMBER < SPLIT / 2))
78         throw new IllegalArgumentException("Number_should_be_less_than_SPLIT_and_more_than
79         ↪ SPLIT/2_due_to_sinus_period")
80
81     //Split the sinus period in SPLIT pieces and take the sinus value in each of them
82     for (int i = 0; i < ITERATIONS; i++)
83         buffer[i] = NUMBER*sin(2 * PI * i / SPLIT) + sin(NUMBER * 2 * PI * i / SPLIT)
84
85     dftStraight.run(buffer)
86     result = dftStraight.getAmplitudes()
87
88     int amount = 0, size = dftStraight.getSize()
89     for (int i = 0; i < size; i++)
90         if (result[i] > EPS)
91             amount++
92
93     assertEquals(2, amount)
94 }
95
96 void testApplyNewAmplitudes() {
97     double[] modulation
98     DFTStraight dftStraightApplied = new DFTStraight()
99
100     for (int i = 0; i < ITERATIONS; i++)
101         buffer[i] = sin(2 * PI * i / SPLIT)
102
103     dftStraight.run(buffer)
104
105     modulation = new double[dftStraight.size]
106     Random random = new Random()
107     for (int i = 0; i < dftStraight.size; i++)
108         modulation[i] = abs(random.nextDouble())
109
110     dftStraightApplied.setData(dftStraight.getData())
111     dftStraightApplied.applyNewAmplitudes(modulation)
112
113     double oldPhase, newPhase
114     for (int i = 0; i < dftStraight.size; i++) {
115         oldPhase = dftStraight.getPhase(i)
116         newPhase = dftStraightApplied.getPhase(i)
117
118         assert abs(oldPhase - newPhase) < EPS
119     }
120 }
121
122 void testApplyNewPhases() {
123     double[] modulation
124     DFTStraight dftStraightApplied = new DFTStraight()
125
126     for (int i = 0; i < ITERATIONS; i++)
127         buffer[i] = sin(2 * PI * i / SPLIT)
128
129     dftStraight.run(buffer)
130
131     modulation = new double[dftStraight.size]
132     Random random = new Random()
133     for (int i = 0; i < dftStraight.size; i++)
134         modulation[i] = abs(random.nextDouble())
135
136     dftStraightApplied.setData(dftStraight.getData())
137     dftStraightApplied.applyNewPhases(modulation)
138
139     double oldAmplitude, newAmplitude
140     for (int i = 0; i < dftStraight.size; i++) {
141         oldAmplitude = dftStraight.getAmplitude(i)
142         newAmplitude = dftStraightApplied.getAmplitude(i)
143
144         assert abs(oldAmplitude - newAmplitude) < EPS
145     }
146 }

```

Листинг 16: DFTInverseTest.groovy

```
1 package com.sunradio.math
```

```

2
3 import static java.lang.Math.PI
4 import static java.lang.Math.sin
5
6 class DFTInverseTest extends GroovyTestCase {
7     final EPS = 0.00001
8     final ITERATIONS = 100
9     final SPLIT = 100
10    final NUMBER = 80.0
11
12    double[] income = new double[ITERATIONS]
13    DFTStraight outcome_straight = new DFTStraight()
14    double[] outcome_inverse = new double[ITERATIONS]
15
16    // f(t) = sin(t)
17    public void testSin() {
18        // Split the sinus period in SPLIT pieces and take the sinus value in each of them
19        for (int i = 0; i < ITERATIONS; i++)
20            income[i] = sin(2 * PI * i / SPLIT)
21
22        outcome_straight.run(income)
23        outcome_inverse = DFTInverse.run(outcome_straight.getData())
24
25        double difference;
26        int amount = 0;
27        for (int i = 0; i < ITERATIONS; i++) {
28            difference = outcome_inverse[i] - income[i]
29            if (difference > EPS) amount++
30        }
31
32        assertEquals(0, amount)
33    }
34
35    // f(t) = NUMBER*sin(t) + sin(Number*t)
36    public void testTwoSin() throws IllegalArgumentException {
37        if ((NUMBER > SPLIT) || (NUMBER < SPLIT / 2))
38            throw new IllegalArgumentException("Number_should_be_less_than_SPLIT_and_more_than
39            ↪ SPLIT/2_due_to_sinus_period")
40
41        // Split the sinus period in SPLIT pieces and take the sinus value in each of them
42        for (int i = 0; i < ITERATIONS; i++)
43            income[i] = NUMBER*sin(2 * PI * i / SPLIT) + sin(NUMBER * 2 * PI * i / SPLIT)
44
45        outcome_straight.run(income)
46        outcome_inverse = DFTInverse.run(outcome_straight.getData())
47
48        double difference;
49        int amount = 0;
50        for (int i = 0; i < ITERATIONS; i++) {
51            difference = outcome_inverse[i] - income[i]
52            if (difference > EPS) amount++
53        }
54
55        assertEquals(0, amount)
56    }
57 }

```

Листинг 17: FilterTest.groovy

```

1 package com.sunradio.math
2
3 import static java.lang.Math.PI
4 import static java.lang.Math.sin
5
6 class FilterTest extends GroovyTestCase {
7     final ITERATIONS = 100
8     final SPLIT = 80
9     final EPS = 0.00001
10
11    double[] buffer = new double[ITERATIONS]
12    double[] winFunc = new double[ITERATIONS]
13    double[] applied = new double[ITERATIONS]
14
15    void testApply() {
16
17        // Split the sinus period in SPLIT pieces and take the sinus value in each of them

```

```

18     for (int i = 0; i < ITERATIONS; i++)
19         buffer[i] = sin(2 * PI * i / SPLIT)
20
21     winFunc = Filter.BlackmanNuttall(ITERATIONS)
22     applied = Filter.apply(buffer, winFunc)
23
24     for (int i = 0; i < ITERATIONS; i++)
25         assert applied[i] - (buffer[i] * winFunc[i]) < EPS
26 }
27
28 void testGetOutputFilter() {
29     double[] inputWindowFunc = Filter.BlackmanNuttall(ITERATIONS)
30     double[] outputWindowFunc = Filter.getOutputWindowFunc(inputWindowFunc)
31     double sum = 0.0
32
33     for (int i = 0; i < ITERATIONS; i++)
34         sum += inputWindowFunc[i] * outputWindowFunc[i]
35
36     assertEquals(1.0, sum)
37 }
38 }

```

Листинг 18: ScaleTest.groovy

```

1 package com.sunradio.math
2
3 class ScaleTest extends GroovyTestCase {
4     void testRun() {
5         final EPS = 0.00000001
6         def arr = [5, 8, 7, 2] as Integer[]
7         def from = 0.0
8         def to = 1.0
9
10        def outputExpected = [0.5, 1.0, 5 / 6, 0.0] as double[]
11
12        double[] outputValue = Scale.run(arr, from, to)
13        for (int i = 0; i < 4; i++)
14            assertTrue((outputExpected[i] - outputValue[i]) < EPS)
15    }
16 }

```