P4 - DSP Beat Box

4th Semester Medialogy Project

29-05-2013
Aalborg University

Andrei Vlad Constantin

Dennis Jensen

Simon Kaarfast

Christian Strauss

AALBORG UNIVERSITY
STUDENTS STUDYREPORT

Fourth Semester
Medialogi
Sofiendalsvej 9
9000 Aalborg

Title: DSP Beat Box

Theme: Sonic Interactions

Project period: 01-02-2013 to 29-05-2013

Project group: 430

Members:

Andrei Vlad Constantin

_____

Dennis Jensen

_____

Christian Strauss

_____

Simon kaarfast

_____

Supervisor(s):

Lance Putnam

Copies: 3

Pages: 94

Annex number and kind: 1 Data CD-ROM

Finished: 27-05-2013

Synopsis:

This report investigates a potential possibility of developing an assisting technology to aid students in the comprehension of the course Sound and Music computing. The course was very technical and heavy with mathematical formulas and numbers which many students found confusing and hard to relate to.

The project attempts to analyze the problem and solve it by constructing a tool which a teacher can utilize for their course in order to help teach their students gain a better understanding of the different aspects of the course by giving them a tool that provides visual and audible feedback if needed.

While it is estimated that the idea for a solution presented in this project could potentially help solve the problem, it was unable to prove a viable solution in its current state of development.

Foreword

This 4th semester project at Medialogy, Aalborg University has been centered on the course 'Sound & Music Computing' and has specifically focused on attempting to improve the course. It is important to note, however, that the purpose of this project is not to substitute the course's teacher or in any way criticize the course or insinuate that either the course structure or the teacher is insufficient. Rather, it is an attempt to come up with an easy-to-use method for the course teacher to illustrate the different theories to the course's students during the lectures, as well as a tool for course's students to use while self-studying.

Essentially, this project is not about whether - or how the 'Sound & Music Computing' course is flawed or otherwise lacking; it is about how the course teacher's tools for teaching his or her students can be improved.

Thanks go to the 4th semester students at Medialogy for aiding in the Lo-Fi test of this project and for giving valuable input on design and functionality as well as to the supervisor of the project, Lance Putnam, for his guidance and input throughout the project period. A note of thanks also goes to the course teacher of 'Sound & Music Computing' for his suggestions on how to test the efficiency of the project's end-product.

Index

Initial Problem

In the course Sound and Music Computing most of the explanation is done in the form of writing with chalk on a blackboard. While this may be sufficient for some students, others may prefer having a lesson explained through different medias, for example: aural, or visual.
While the course already makes great effort to include as many learning styles as possible, having an external learning source might help the students in understanding the course better, while also relieving some of the pressure put upon the teacher.

Problem analysis

Digital processing, where one can learn how to process data from analog to digital either with images or sound. It can be mind boggling to handle and process the information without getting confused, or just have a very hard time understanding the principle and concept of Audio digital processing. While in Visual/image digital processing (IDP), there were images to work with and somewhat more understandable in what was happening when linking images to the mathematical algorithm behind it. The same, however, cannot be said about audio digital processing (ADP) when trying to link mathematical formulas to sound.
As teaching has its various methods and techniques of learning, ADP was the most difficult to learn.

Although sound comes various forms of wave length from high to low frequencies, it is still difficult to comprehend what is happening to the sound wave when it gets manipulated with either through sound effects and/or an equalizers. Besides all the math behind the *sin wave,* a mathematician will have the advantage in learning and understanding the background behind sound waves and the how it is manipulated.
So in order to assist those who are less likely to gain an understanding of a subject from simply studying a mathematical formula, it would be relevant to help them visualize how the theory-aspect works in practical application and put it into context of the formulas.

The following description will be looking at 2 software programs that we have been introduced and learning through the fourth semester of Medialogy: *PureData* and Audacity.
Although the programs such as Pure Data (PD) and Audacity which both are very capable softwares for educational purposes, only pure data allows for real time visuals when programming sound, while audacity has a much broader easy, accessible functionality and selection of options when it comes to manipulation of sound. In audacity you have to apply the effect first and then see its updated state.

Explaining PureData

Pure data is a visual based programming language with all its functionality already preset for the user. This visual GUI software allows drag'n'drop, click and place functionality users to quickly get into the program without having to fully commit to learning a new programming language. Although some understanding of the functions is required of the user to learn to fully understand the meaning and functions of certain parameters.

What this means is that although the user may quickly learn how sound works, the more complicated algorithms and layouts for creating various effects and how these effects work requires more research and self knowledge.

"*Pd is certainly powerful and it has massive potential, but it is so extremely frustrating to use. The interface is horrid, the documentation is incomplete, and help is difficult to find. I've wasted massive amounts of time trying to debug problems only to find out that the system itself was screwing up, not my own logic, after closing and reopening my patch. If unseen state can mess up a patch, why is there no reset button? Pd almost seems to wear its spartan design as a badge of honor. I've used Pd for two major projects, but never again. I value my time and effort too much. Looks like it's time to front up and buy Max or learn SuperCollider.* "

- Alan[1]

Taking the negative review into perspective it does come to mind that PD does lack in the interface intuitiveness as well as its obsolete compared to its competitors in terms of function.

Despite the negative review, we must equally look at the positive review of this software as its important to note it has a better score of positive reviews contra negative ones.

"*It's creatively freeing and runs real time dsp on a machine that was be considered obsolete years ago. Starting out may be more difficult then ready-made music software .. but in the end you learn much more about your music and yourself through crafting your own tools.* "

- Dan Wilcox from robotcowboy.com[2]

[1] *http://sourceforge.net/projects/pure-data/reviews/ - 27th of May 2013*
[2] http://sourceforge.net/projects/pure-data/reviews/ *, - 27th may 2013*

So never the less, PD does bring a formality of a learning tool to understanding about our own sound we create in the program.

Audacity, however, is a free, multilingual audio editor and recorder for all operating systems.
Through its basic looking GUI layout, users can find it rather intimidating at first look with no tutorials for first time users so users will find themselves looking at something they might not fully understand.
However, Audacity does feature a lot of different functionality, as it was created for the purpose of serving as simple audio/editing software.
Through simple button layout you can record or import audio files. The sound wave data will then be displayed showing the data in a sound spectrum which can be zoomed in on for closer analysis of the sound wave if desired. Through drop down boxes for the menu, it is also simple to apply various effects to the audio clip such as compressors, echoes, or even to remove certain background noise through the whole sound clip. However the effects seem to only be applied destructively meaning, that once applied, it is applied permanently through the whole sound clip and cannot go back on the latter. So tweaking an equalizer or toggling a compressor at certain points are not available like some other advance packages. The edited effects on the sound wave are instantly displayed in the visual wave spectrum.

"*I can use Audacity to bring music beds in, create intros, and use sound effects, and then convert my finished project to MP3 format. Sometimes, I also import audio files that are giving me trouble, and view their waveforms to see if there are any visual clues as to what the problem might be.* "
- Brian Bertucci from About.com[3]

It is important to keep in mind that both serve a purpose for its users. One is as a learning tool for understanding the mechanisms behind audio processing while the other serves as a audio manipulator/recorder for more practical projects but the one thing they both have in common is they are both tools created to be simplistic and straightforward to use.

[3] http://podcasting.about.com/od/recordingequipment/fr/deepaud.htm , - *27th may 2013*

The trick is to combine the functionality of both into a single straightforward application. That has the learning mechanism for the user to understand and learn how audio effects affects the waveform of a sound while having the real time visual aspects of the sound wave and editable sound effects. At the same time, making the application a more attractive tool and preferable tool instead of pure data and Audacity.

Background Research

Given that the goal was to make a solution to an educational problem, a lot of research needed to be done in fields such as Learning, Inspiration, and Motivation, in order to progress any further with the project. Every group member chose one of the aforementioned fields, and began their research.

Learning Styles

Learning styles refers to how a person best seems to gain new knowledge. Different people require different approaches to better understand the material given to them. The seven main learning styles are as follows:

- **Visual** (spatial): You prefer using pictures, images, and spatial understanding.
- **Aural** (auditory-musical): You prefer using sound and music.
- **Verbal** (linguistic): You prefer using words, both in speech and writing.
- **Physical** (kinaesthetic): You prefer using your body, hands and sense of touch.
- **Logical** (mathematical): You prefer using logic, reasoning and systems.
- **Social** (interpersonal): You prefer to learn in groups or with other people.
- **Solitary** (intrapersonal): You prefer to work alone and use self-study.[4]

David Kolb, who was the one to start the learning style movement back in the early seventies, believed that:

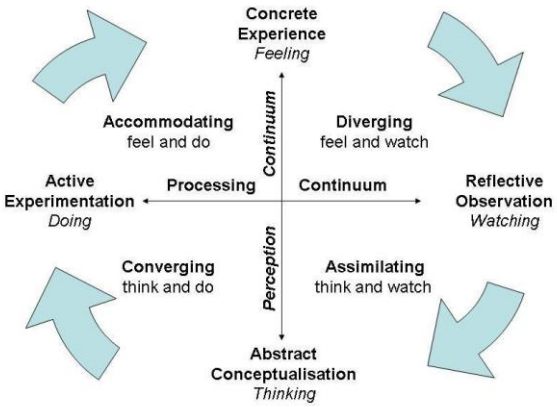
"...learning styles are not fixed personality traits, but relatively stable patterns of behavior that is based on their background and experiences. Thus, they can be thought of more as learning preferences, rather than styles."

Kolb believed that the act of learning required one to go though four processes which he named: Diverging, Assimilating, Converging, and Accommodating. He made a learning model based on this, which differs from others in that it does not break up the learning styles into individual parts; Instead, he made what he called

[4] http://www.learning-styles-online.com/overview/

The Learning Cycle that cycles though the four stages, so that one could try them all and later make the decision as to which method suited them the best.

The model is based on two lines, one going horizontally the other vertically, coming together to form a plus sign. The horizontal line is referred to as the "Processing Continuum", and it has "Doing" on the leftmost point, and "watching" in the opposite end.
The vertical line is referred to as the "Perception Continuum", and it has "Feeling" at the top, and "Thinking" at the bottom.



The idea behind it is that two adjacent point would come together to make one of the four processes mentioned above, for instance, the one on the left and the one on the top would make Accommodating, a mixture of learning by doing and feeling.[5]

Another model in this field goes by the name of VAK, and is a very popular model due to its simplicity in only revolving around the three learning styles: Visual, Auditory, and Kinaesthetic. According to VAK everyone uses all three, but one or two of them will always be dominant and these are the ones that determine the best way for you to gain new knowledge. [6]

Motivation and Learning

Attending a lecture where one is unable to utilize ones preferred learning style, will most likely result in the person having to do significantly more work in order to achieve the same results as in other lectures, and this can have quite an impact on motivation. To make the learning experience more rewarding for students, it is important to make them aware of the significance of internal control. Without this kind of control, a student being introduced to a sufficiently difficult material will quickly develop a decreased interest in the subject[7].

Potential Platforms

When looking into which platform for this program to work on, there are a few to choose from, namely Mac computers, iOS devices, PCs, and Android Smartphones. Making it available on mobile devices would

[5] http://www.nwlink.com/~donclark/hrd/styles/kolb.html (Source of text and image)
[6] http://www.nwlink.com/~donclark/hrd/styles/vakt.html
[7] http://www.edpsycinteractive.org/topics/motivation/motivate.html (Cognitive - Right after the table)

allow the user to use the program at any time, while also allowing for great recordings without having to buy a new microphone. Making it for computers would have many of the same benefits as the Smartphone, but it would be less practical to use outside of the house. One thing the PC and the android devices have in common is the option to write the code in the Java language. Since Java is a part of this course, it would be ideal to create something using this.

Target group analysis

One of the important factors that need to be taken into consideration is our target group. It is necessary to determine the needs and general preferences of the people in our target group.
To solve this, the selected target group which the application is intended for, needs to be analyzed.

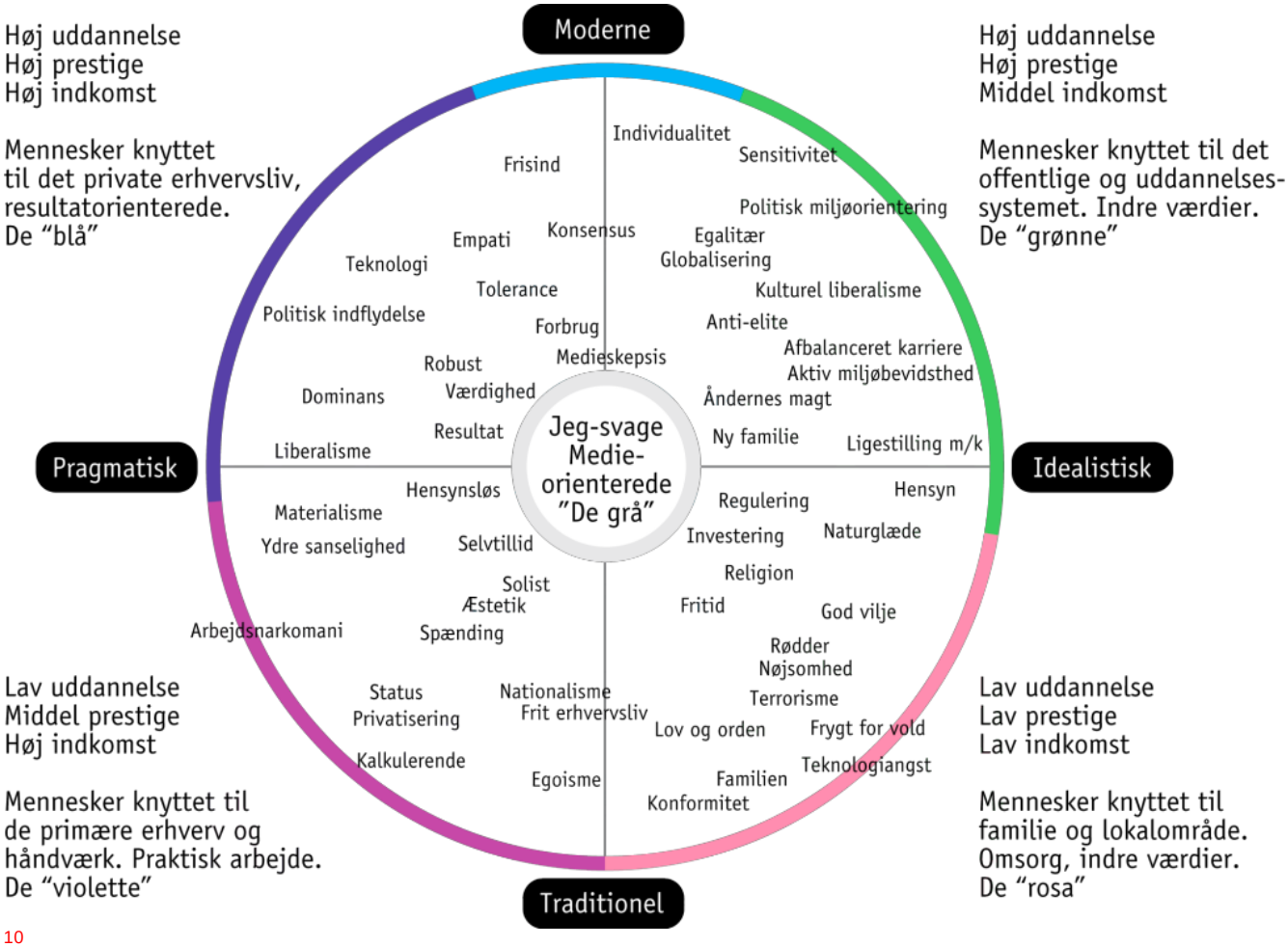
Looking at the Minerva model it is possible to start by taking an archetype group and use it as a guideline to further understand our target group.

As students at a university can range from ages of 18 and above (around 40 although uncommon, it is not unheard of), one must also carefully reflect on this matter as an age difference could have potential unknown factors in terms for the project's goal.
According to the article by "dkuni.dk"[8] on page 11, in Denmark an average age between 25 to 34 years were little over 35% that were part of a medium or longer higher education of 2009 (University or similar). While only 20% of the Danish population in the range of 55 to 64 years old were part of a higher education system.
From this data, one can take the presumption that our target group is more likely to be young and/or of same age.

To further backup this hypothesis, according to the University of Copenhagen's (KU)[9] own numbers, in 2012 the median age of students was 21.1

<image: http://nightlifescience.files.wordpress.com/2011/10/minerva-model.png?w=1024&h=729>

8 http://www.dkuni.dk/~/media/Files/Publikationer/Tal%20om%20de%20danske%20universiteter%202011.ashx , - 27th may 2013
9 http://tal.ku.dk/studerende/F_studerende/ , - 27th may 2013



10

By looking at the Minerva model, a model that characterizes the traits of the consumer in four distinctive groups, the targeted audience would be classified by its attributes to the blue segment as well as green segment. The blue segment characterize Individualistic, wealthy, materialistic, technology-experienced users, while the culture-rich, idealistic, middle-class individuals, and modern-minded characterize of the green segment.

[10] Minerva Image source: http://nightlifescience.files.wordpress.com/2011/10/minerva-model.png?w=1024&h=729 , - 27th May 2013

Problem Statement

The intention is to investigate to which degree it is possible to create a fun and helpful learning tool which can be used to aid students in better understanding and learning the Sound and Music Computing courses, while making it less technical and more visually expressive.

Sound Processing Methods

As the software is intended as an assisting technology in understanding Sound and Music Computing, the software has to contain sound processing methods related to the course. Optimally, every single aspect of Sound and Music Computing would be featured in the software, but due to a limited timeframe for research and development the amount of applicable effects featured in the software had to be restricted to one per member of the project group /developer team.

This way each group member could focus on researching his own individual subject and write an algorithm for the effect to be implemented in the software code

A list containing the core subjects of Sound and Music Computing was narrowed down to the most suitable subjects to incorporate into the software and these were then delegated between the members of the group. Primarily the applicable effects were chosen because the software's intended function is to help the user understand how the respective effects affect the sounds they are applied to. And so it would make less sense to focus on implementing several types of equalizers, for instance.

It *was* decided to implement a graphic equalizer, however, as it serves the purpose of giving the user visual feedback on the change in characteristics of the sound. The remaining subjects are applicable effects.

As mentioned before, the optimal solution would incorporate every possible effect related to Sound and Music Computing, but for the sake of developing a prototype within the given timeframe, the featured effects are limited to:

- Dynamic Range Compression
- Flanger
- Echo/Delay

Graphical Equalizer

In sound processing, equalization is the method used to alter the frequency response of an audio system using linear filters. Like most media equipment, this project uses relatively simple filters to make bass, middle and treble adjustments.

The equalizer implemented in this application is based on three frequency thresholds and three volumes, corresponding to the thresholds. If the sound sample belongs to a specific range, it is adjusted by its respective volume, being either amplified or reduced.

DRC

Dynamic range compression (DRC), or simply compression, reduces the amplitude of loud sound samples or amplifies quiet sound samples by processing the dynamic range of an audio signal. Compression is commonly used in sound recording, audio reproduction and broadcasting.

Compression is the process of bringing peaks and valleys of sounds closer together, which in effect reduces the very loud sounds and makes the less loud sounds more audible. It creates a much smoother sound because it reduces those loud peaking sounds, allowing for the sound volume to be increased without the loud sound peaks causing damage to either hearing or speakers.

$$\text{Output} = \frac{\text{Input-Threshold}}{\text{Rate}} + \text{Threshold}$$

The Rate determines the rate of compression. Typically, for compression, the practical rates are $\frac{1}{4}$ to $\frac{1}{2}$ .The use of compression rates of $\frac{1}{10}$ and above are considers limiters, which are merely extreme forms of compression.

However, if the Threshold exceeds the Input, the sound is no longer being compressed, but expanded. This happens because the threshold is added to the Input volume in the above formula. So if the threshold exceeds the Input volume, the volume is increased on any signal values that reach the threshold, increasing the dynamic range between the loud and quiet sounds which is called expansion.

Flanger

The flanger, or flanging, is an audio effect produced by mixing two identical signals, one signal delayed by a small and varying period. This results in an output characterized by peaks and notches being produced in the resultant frequency spectrum, related to each other in a linear harmony. Varying the time delay causes these to sweep up and down the frequency spectrum.

This effect was designed according to the definition, but was not implemented. As such, mentioning further implementation characteristics of the design is irrelevant.

Echo/Delay

Delay is an audio effect that consists of recording an input signal, storing it and then playing it after a period of time (delay). The delayed sound may either be played back multiple times, or played back into the recording, thus creating the impression of a decaying echo.

This effect was designed according to the definition, but was not implemented. As such, mentioning further implementation characteristics of the design is irrelevant.

Visualization design

Initially, it was intended to offer two types of visual output to the user: waveform graph and frequency spectrum graph. A waveform graph for an audio signal shows information about the amplitude of each recorded sample, while the spectrum graph provides information about frequency values for each recorded sample.

Due to the implemented effects being only DRC and an equalizer, it was decided that it is more relevant and important to provide the user with the possibility to view how the digital processing affects the frequency spectrum. Furthermore, the visualization implementation is capable of displaying both type of graphs, but, due to lack of space on the interface, it was decided to only show the spectrum graph in this version of the program.

Design

KISS – Keep It Simple, Stupid

Keep It Simple, Stupid – abbreviated *"KISS"* - was a design principle originally thought up by Clarence Leonard Johnson in the 70's. The principles is based on avoiding unnecessarily complex designs [11], which allows for an easier understanding of how to operate an interface, allowing for a less steep learning curve and helping to avoid errors, not only by the user, but in production and maintenance as well.

The design of this application will follow the same principle of simplicity in order to achieve the aforementioned benefits in development as well as usage.

Grouping of related items

Another design principle used in designing the interface, is the principle of grouping UI-elements that have relation to each other.[12] An example of this, is that the power button and reset button on a PC tower is usually located roughly in the same place, because their functions have relation to one another – one powers the computer on and off, the other interrupts the power for a short moment, causing the computer to reboot. This design approach makes for an intuitive interface and aids the user in figuring out the individual functions of elements of the interface.

[11] http://www.nap.edu/html/biomems/cjohnson.pdf - May 7th, 2013
[12] Lars Bo Larsen

Design Criteria

When designing the User Interface, it is important to make a set of design criteria that help ensure the interface is tailored to the purpose of the software as well as the needs of the user.

Intuitive controls

Since the idea is for the users to use the application to aid them in learning Sound and Music Computing, it's important that they are not required to spent a lot of time dealing with complex controls and a steep learning curve, but instead focus on the theories and methods of the course. For that reason, the user interface need to have intuitive control mechanics that make sense and allow the users to start using the application for its intended use right away.

Ease of use

Because the user is meant to be able to use this application during a studying session – and possibly a lecture as well – it is better if the application does not require any preparation before it is ready for use; it should be usable On-The-Fly.

Comprehensiveness

As the sole purpose with this application – as previously explained – is to help the user learn Sound and Music Computing, it is absolutely crucial that the feedback provided by the application is comprehensible to the user, or it will not serve its purpose.

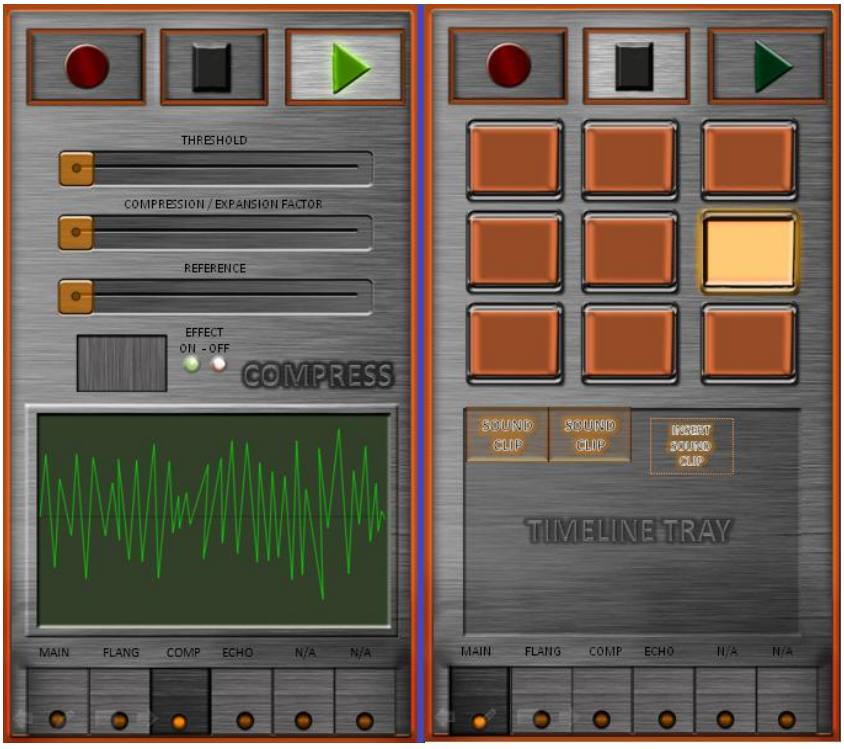
It's important to ensure that the application provides the user with audible and visual feedback that will enhance their learning rate considerably. Considering that if the user would spend any amount of time using the application, that time must be well spent – meaning it must yield results – or usage of the application would be redundant.

Proportions suited for a Smartphone application

When designing the visual elements of the user interface, it's important to keep in mind the intended platform for this software – a Smartphone. That means the software will obviously be displayed on a phone display and not a 24 inch monitor and so elements of the user interface must not be too small, or it will quickly become frustrating and more time-consuming for the user to operate.

Elements of the User Interface

Preview



Design Theme

The design is inspired by a retro tape recorder and classic jukebox theme. Both apparatuses are sound oriented and matches the purpose of the application itself. It is a desired feature that the design theme communicates to the user, what the application is about.

Figure 1 - A classic jukebox - Source: http://www.radfordpl.org/wp-content/uploads/2011/05/jukebox22.jpg

Figure 2 - A retro tape recorder -
Source: http://i.istockimg.com/file_thumbview_approve/9916609/2/stock-photo-9916609-retro-tape-recorder.jpg

Tab system

As with many other applications and browsers, this design uses a tab-system to allow for several "pages" of user-interface, while maintaining easy navigation in the application.

RECORD-, STOP- and PLAY-buttons

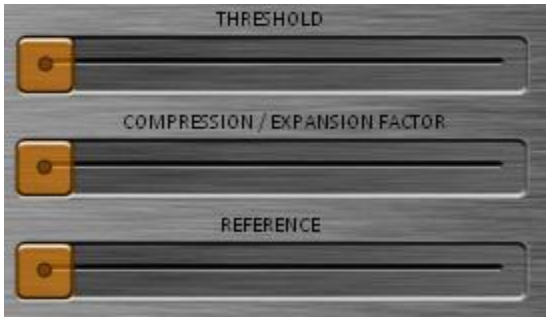
The Record- Stop- and Play-buttons are located at the very top of the application and in every tab to allow access to them at all times.

Effect-button

The effect button is used to toggle the respective effect on/off – which effect will depend on which tab is activated. To clearly indicate to the user which state the effect button is in, a green and a red LED will light up respectively

Sliders

Because the applicable effects have changeable parameters, those parameters need to be adjustable by the user, but in a quick and efficient way. Sliders were chosen over manual value-input boxes because it is believed to be aesthetically appealing as well as user-friendly, since the user won't need to type in values manually, but simply slide between pre-determined values.



Sound buttons

The sound buttons that are meant to store sounds recorded by the user, are designed so that they light up when pressed. They buttons are individually toggled on/off to control which button to record to – or from which button to load sound into the Timeline Tray.

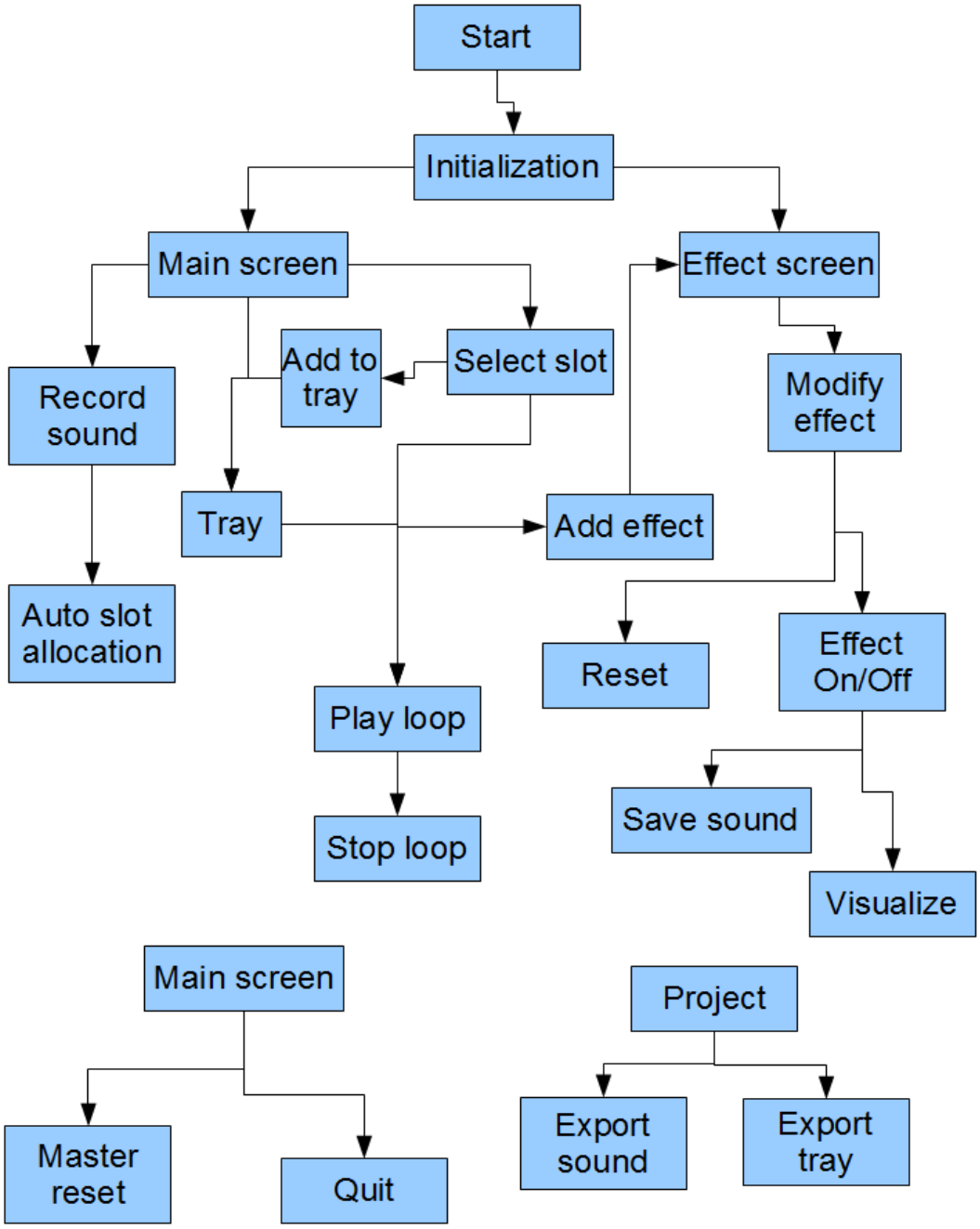
Graphic Display

The graphic display gives the user with a visual tool to observe the characteristics of a played sound and how the different applied sound-computing effects affects that sound

Timeline Tray

The Timeline Tray is where the user can queue up pre-recorded sounds from the sound buttons and later apply effects to them via the respective effect tabs.

Workflow Diagram



Guide to the work-flow diagram

The first step in the process of running the application is to launch it. While the application is in the initialization phase, an animated loading screen will keep the user entertained. After the initialization process is complete, the user will be greeted by a simplified help window, explaining the basic functionality of the program. This window only appears automatically when the program was launched for the first time on a new device, after which it becomes available to the user only by pressing the help button.

At this point, after having closed the informative note, the user is viewing the Main window of the program. This is the window associated with most of the basic sound management tasks, such as Record, Play Loop, Stop Loop and all Sound Slot and Tray manipulation.

The foremost action the user should accomplish is to record one or more sounds to utilize as input for the application. Upon pressing the Record button, the program automatically starts to register the sound input, storing it in the next free Sound Slot.

When the user has recorded a satisfying amount of sounds, he has several options to manipulate them. The most basic of these actions is to select a Sound Slot and start playing it in a loop. While the sound is in playback, the user has the opportunity to play around with the various effects/filters available. Also, pressing the Stop Loop button will terminate the playback.

A more advanced method of managing the sounds is to add them to the Tray. In order to add sounds to the Tray, the user must select a Sound Slot and then click on a free spot, anywhere in the Tray area. The Tray works as a queue, meaning elements can only be added at the end, and then, can only be played from the beginning. Also, unlike traditional queues, the user can remove an element from any position within the queue. From an audio perspective, the Tray and the Sound Slots function on the same principles, meaning that, once selected, they can be played, stopped and modified with effects/filters in the same way.

Choosing to add one or more effects/filters to the playback of a sound or the Tray, the user has to navigate to one of the effect/filter tabs. Each individual tab has its own fields and sliders based on the modifiable parameters of each effect/filter. The tabs also offer other functionality than customization, such as turning the effect/filter on or off, saving the modified sound, resetting the parameters and the possibility to visualize amplitude/time comparison and frequency/time comparison graphs between the original and the modified sounds.

The application also presents two methods of saving the output. The user can export an individual sound and/or the whole Tray, as WAVE audio files.

Other than the sound manipulation, the application offers the possibility to perform a Master reset, which clears the Sound Slots, the Tray and resets the effect/filter parameter values to their default. Also, since the application is intended to be used on a smartphone, it offers a Quit button to make sure no device memory remains associated with the program.

Lo-Fi Test

A Lo-Fi Test was performed on two separate groups; the target group and a group of people that didn't fit the target group.

The purpose of performing the Lo-Fi Test on the target group was to see if our design was comprehensive and appealing to the group of people to whom the use of this software is intended. The test was designed as an interactive PowerPoint with every button and control mechanism animated - although not functional - to allow the testers to get a better feel of the software's control surface.

Idea

All of the testers got the general idea of the software's intended usage and liked the concept of an application that assists in understanding the different methods of the Sound and Music Computing course.

Sound Buttons

The sound slot buttons which store individual sound bites caused some confusion for several of our testers and while a few of the testers figured it out, some mistook them for being part of a sequencer-feature, while others simply had no idea what the purpose of those buttons were; the buttons lacked an indication of what purpose they served.

Timeline Tray

The Timeline Tray was a few times mistaken for an Equalizer and was mostly quite confusing to the testers, meaning that they had little to no idea how it was supposed to work.

Figure 1 - Screenshot of the Lo-Fi PowerPoint in Tab 1 – The orange buttons are the sound buttons in which sounds are stored. Below that is the Timeline Tray.

Effect Button

The Effect ON/OFF button, which toggles the respective sound effects on or off, was criticized for its lack of contrast. Due to having a hue too similar to the UI base (background), it essentially wasn't eye-catching enough, which made it take longer for the testers to notice it.

As a supplemental indication of whether the given effect is ON or OFF, two LEDs located next to the button will light up respectively to indicate to the user when the effect is applied and when it is not. This feature was generally liked by the testers, but it was pointed out that the 'glass glare' effect was too bright on the inactive LEDs, making it harder to distinguish between the active and inactive LEDs.

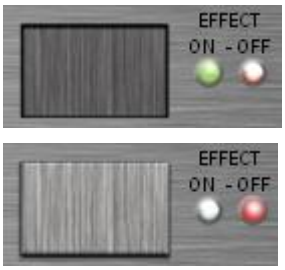


Figure 1 - Two separate examples of the EFFECT BUTTON switched ON/OFF, respectively.

Sliders

The Lo-Fi's sliders were meant to illustrate that they would be featured in the actual software, but were only roughly animated in the PowerPoint, meaning they were binary sliders that went from left to right when clicked on by the testers. Fortunately, they recognized them as sliders and intuitively used them as such; it was observed that all testers clicked and held the left mouse button while attempting to drag the sliders. While this approach didn't work in the Lo-Fi (due to aforementioned reasons) this is how the sliders are supposed to work in the actual software and because the testers recognized them as sliders and knew how they were supposed to work, they could easily abstract from the crude animations of the Lo-Fi version.

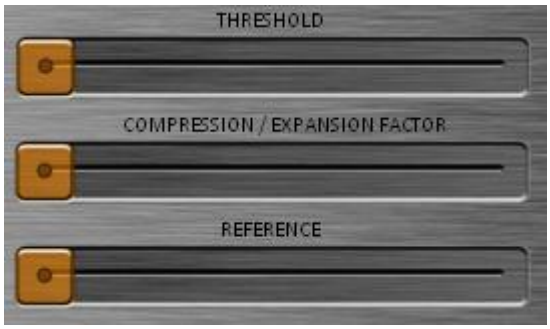


Figure 2 - An example of some of the sliders in the Lo-Fi.

Layout

Overall, the testers felt the layout made sense in regards to the location of the different control-features and had no problem figuring out how to use the Record-, Stop- and Play buttons. A few of the testers, however, would have preferred them to be located near the bottom of the screen, above the tabs.

Figure 3 - The Record, Stop and Play buttons from the Lo-Fi.

Lo-Fi Test Outside of Target Group

This part of the Lo-Fi Test was carried out with a test subject that was not in the designated target group, which served the purpose of indicating if the software would be suitable for use by people outside of the target group it was specifically designed for as well as giving additional feedback on the design and user-friendliness of the user-interface.

The testers expressed confusion on how exactly to operate the recording system, but showed great insight in how to apply sounds stored in the Slot Buttons to the Timeline Tray.

It was suggested to include a Show-On-First-Startup Help Screen that explains the basics on how to record and store sounds, apply them to the Timeline Tray and then apply effects to them - this help screen should be available at all times via a button (for instance). It was emphasized that such a help screen would go a long way in clearing up any doubts about functionality in the user interface.

It was expressed that the way the tab-system works felt naturally and familiar, as it is seen in various other applications. In addition, there was no confusion, whatsoever, regarding how to handle the application and tweaking of effects.

Regarding the Slot Buttons, it was suggested to use some form of indicator of whether or not a given Slot Button had sound data stored in them - for instance by differentiating them by color - in such a way that the user would always know exactly which buttons were free to record onto and which already had sound clips ready for use.

Aesthetically, the design received great praise - everything was sized correctly and had the right amount of space in between them - and the overall RETRO-oriented theme was recognized.

Implementation

In this chapter, the group will be presenting the structure and algorithms that enable the application to fulfill its purpose. In the sub-chapters, individual components of the application will be described in greater detail as to facilitate a better understanding of how the group managed to create a physical high fidelity prototype of the desired program.

The programming language used for this application is Java. This has been utilized in association with the Eclipse IDE for Java Developers, version Juno service release 1, running with Java SDK7 and Java SRE7.

Java is a medium-level, general-purpose, class-based, object-oriented programming language and it is designed to have a very low dependency on external frameworks or applications. Furthermore, this allows developers to implement the WORA principle ("write once, run anywhere") that enables an application written in Java to be compiled only once for any given deployment of Java on a platform running the Java virtual machine.[13]

Due to the project being primarily destined for Smart-phone platforms, this great feature of Java allowed the group to write and test the application on a computer and be sure it would perform similarly on the intended platform. Also, because of the same reason of being oriented towards Smart-phones, the group decided to optimize the application to be more processor intensive, rather than memory intensive. Smart-phone platforms tend to be better at using more processing cycles to accomplish tasks, rather than dealing with clogged up memory space that could be better spent on the personal background programs that the users have installed.

Some of the components used have been implemented with inspiration from similar stand-alone programs found on the Internet. The group utilized resources, such as the Java Trail and the Java Programmers Guide, provided by Oracle, in order to find relevant helpful implementations for methods specific to Java. No external libraries have been utilized during the implementation of this application, as all the necessary components for building the program are available with the basic Java development package.

[13] http://www.java.com/en/download/whatis_java.jsp , 28th of May 2013

Program structure

Due to the nature of Java programming language of being class-based and object-oriented, the application code is structured into classes, based on categories of variables and methods that need to be available for specific objects. As such, there are 7 classes that form up the implementation of this program:

- maybeMain.java
- soundClass.java
- temporary.java
- simpleFFT.java
- Compressor.java
- graphicEqual.java
- WaveformGraph.java

The maybeMain.java class holds the interface code of the application. This class contains all the action listeners that command the program and it was created using Swing elements, which is one of the interface implementation packages for Java.

The soundClass.java class is responsible for creating and remembering the paths for the various sounds used by the application. It also has the important role of implementing various basic sound manipulation methods, such as play, record and stop.

The temporary.java is an auxiliary class that contains methods responsible for sound byte manipulations and file reading/writing. In short, it is the class that all the other classes are using to some extent.

The simpleFFT.java class implements the fast Fourier transform algorithm and method, which is necessary for obtaining frequency values out of the sound samples acquired by the application.

The Compressor.java is a one-method class responsible for applying the DRC (dynamic range control) effect on a specific sound. It contains the variables required for the algorithm to function, as well as the algorithm itself.

The graphicEqual.java is similar to the DRC effect class. It contains variables required and the algorithm responsible for the graphic equalizer component of the application.

The WaveformGraph.java is a class responsible for the visualization component of the included effects. This class contains the graph computing algorithm, as well as the methods for drawing the graphs inside each of the effect windows.

The code of the application can be found in the Appendix. It has been organized by classes, such as the ones presented above, and each class has been split into sections. Each of the sections are further described, in more detail, in the following sub-chapters, ordered based on the class they belong to, their importance and functionality.

Interface implementation

The GUI component of the application is the most important and most meticulous part. Due to the fact that the program is oriented towards interaction with humans, the user interface takes priority in front of other components, in order to facilitate the intended usability of the application.

As presented in the Program structure sub-chapter, the interface is implemented inside the maybeMain.java class. The interface code mainly consists of Swing components declarations and properties initializations.

As can be seen in the *declare interface components* section of this class, a combination of buttons, labels, sliders and panels have been utilized to put together the GUI of the program.

- buttons are used to enable the user interaction
- labels are used for the clarification of certain functionality that the users have found to not be intuitive
- sliders are used for picking up certain variable values that are required for controlling sound effect parameters
- panels are used to organize the interface and position various elements in a controllable environment

Furthermore, in order to perform tasks, such as automatic assignment of sound slots to recorded sounds, some flag and counter variables have been used, located in the *declare flag and counter variables* section.

As an example, the counter variable 'trayIndex' is used to track the position of the element that is being added to the Tray. Similarly, the flag variable 'compEff' indicates whether the DRC effect is enabled or disabled. An argument can be made that flag variables should be of the type boolean. However, this limits the possibility of further development of the functionality, due to the boolean type having only two values. If an integer type variable is used instead, this allows for the further development of other cases in which the same variable would be used as a flag, but with a different value.

Proceeding deeper into the implementation of the GUI, it can be seen that the next phase after declaring the interface elements is to initialize them and certain properties, such as position, color, layout, label text and others. The *method that initializes the GUI components and sets their properties* is responsible for this phase of the process. In this section, many 'set' and 'add' methods, specific to the individual Swing components, are called in order to actually draw the application.

However, in order to command the application using the interface, a special type of methods must be utilized. These methods are action listeners that check when interaction has occurred with one of the GUI components, thus triggering a task to be performed. The action listeners methods are located in the *action listeners* section.

An important Swing issue to take note of is that all the action listeners operate on the same processing thread. This causes time intensive events to block the interface from responding to interaction until the task is completed. In order to solve this problem, the group implemented multiple processing sub-threads to be launched respectively, whenever one of these time consuming events are triggered. The sub-threads are a special class named SwingWorker. As the name suggests, the purpose of this class is to perform background work, while the application is free to run in parallel with these time intensive tasks.

Lastly in order of importance, the maybeMain.java class contains two more methods. One of them is setting the text on a label that indicates which sound is selected on the effect windows (*clarification method that shows the user what sound is selected*) and the other one is responsible for drawing labels in the Tray according to the sounds that are added (*method that operates the display of sound names in the Tray*).

As a final note, it is worth mentioning that there is a reason for which the interface is implemented in a single class. This is due to the practical decision of encasing the GUI in a tabbed panel, in order to make it easier for the user to access the multiple functions of the application. Using a tabbed panel involves having the entire interface code in the same class, due to the way this special type of Swing panel is built.

Input implementation

The application is intended to be a sound processing tool and as such, the only input to be taken is audio. This is implemented partially in the soundClass.java and the temporary.java classes, with the latter containing some auxiliary methods for dealing with the audio data, which will be presented in a separate sub-chapter.

The input aspect of the program has been coded with consideration towards three types of functionality:

- automatic assignment of sound slots
- limited recording time
- possibility to stop the recording process

The automatization of sound slot assignment became a requirement after the Lo-Fi test. Users felt it is more intuitive if the program allocates the sound slots by itself, rather than the user selecting which sound slot to use.

This feature is implemented simply by making a counter variable (soundSlot) to track how many sounds have been recorded. This variable is incremented each time the recording method is called and is reset if all the sound slots are filled, as presented in the *method that helps the automatization of the record method* section of the soundClass.java class.

The limited recording time functionality has been implemented in order to avoid unwanted memory issues. This can occur due to not stopping the recording process properly or overloading the limited memory possibilities of the platform on which the application is running.

The group chose an elegant and simple algorithm for implementing this feature. This consists of starting a background processing thread, that runs for five seconds and then sends an interrupt error message to the recording thread. The code for this small algorithm can be found in the *record input method* section of the soundClass.java class and is identifiable by the object 'stopper' of the type Thread.

The final functionality coded into the recording algorithm is the possibility to stop the process at any given time. This is a common feature found on many media player type applications, being implemented due to usability (quality-of-life) considerations. The actual method is shared with the audio output aspect of the program, in the sense that both types of functionality require it. The code for this algorithm simply closes the input/output line allocated by Java and thus interrupts the respective process, as shown by the *input/output stop method* section of the soundClass.java class.

Output implementation

As the application is destined to be a sound processing edutainment tool, the group chose to make two types of output, in order to cater to multiple learning styles. The output types are:

- audio - implemented in the soundClass.java class
  - normal
  - with effect
- visual - implemented in the WaveformGraph.java class.

As presented, there are two types of methods for audio output. In order to simplify the implementation process, the group decided to code three similar algorithms for playing the audio files. As such, there is a normal play functionality and two functions for playing with an effect enabled.

The *output play method* section of the soundClass.java class showcases the code for the untempered play method. The algorithm in this section performs audio playback for a selected sound or for the sound Tray, based on the value of the soundSlot variable.

If the Tray is selected, it generates a outputStream type object that contains the byte arrays corresponding to the sounds included in the Tray. It then dumps this outputStream into a lineOut object which Java utilizes to recreate the sound.

If a sound slot is selected instead of the Tray, the algorithm simply reads the audio byte data from the sound file and renders it using the same type of lineOut object.

In case one of the effects provided is active, the program will use the respective alternative method for audio output. The reason behind providing these two alternative playback methods was to not create additional flag variables that would have to be sent to the normal play method, which would have complicated the code. The group is aware that this approach breaks the "write once" principle of object-oriented programming, but felt that the clarity in code gained in the trade-off was worth more.

As such, the two output methods are located in the *output play method with DRC effect* and *output play method with equalizer effect* sections of the soundClass.java class. The only difference between these output algorithms and the normal one is that, in the alternative methods, the sound data is processed in an effect object before being rendered. The effect objects in question are presented in a separated sub-chapter.

The second type of output featured in the application is the visual output. This consists of a frequency spectrum graph generated for the unaltered sound, as well as the effect processed sound, displayed in the effect window, based on whether the respective effect is enabled or disabled.

The visual output is implemented in the WaveformGraph.java class. The object is generated using one of the three constructor methods available, each being responsible for initialing different parameters based on which type of sound the graph is computed for.

Furthermore, the algorithm responsible for actually generating the graph can be found in the *graph computing and drawing algorithm* section of the WaveformGraph.java class. This method is one of the weaknesses of the application due to the way the graph is being displayed. Upon calculating the frequency values using the FFT algorithm, the program paints lines inside a Swing panel object. The x-axis coordinate of the line is computed for each frequency value, but due to the limited space for drawing the whole graph, this value is the same for 129 different consecutive values. The y-axis coordinate of the line is similarly computed for each frequency value, but for the same reason of lack of space, is scaled down by division with 10000000000.

Therefore, it can be deducted that the graph is not truly accurate, but has the functionality of offering the user an idea of what the actual audio graph would look like.

FFT and Auxiliary methods

The application uses a large amount of auxiliary methods and algorithms, in order to reach its desired expectations. There are two categories of auxiliary methods:

- FFT algorithm - used for extracting audio frequency values out of audio sample data
- miscellaneous auxiliary methods - used for converting to and from different data types and support the overall functionality of the program

The FFT algorithm was coded using inspiration from various open-source projects available on the internet. It was put together from various sources, based on the desired functionality that the group wanted it to perform. The simpleFFT.java contains the code for this method and, as the name suggests, it is a simplified implementation. This means that the algorithm maybe not be as efficient as other, more complex, implementations.

The FFT class contains some supportive methods that enable the Fourier transform to be performed, such as a conversion method between and integer array and a double array, the latter being required as input for the FFT implementation. Initially, the group wanted to incorporate a multiple window functions feature, but due to complexity considerations, this was let go, opting to only have the basic rectangle window function. Furthermore, there are several types of Fourier transforms that the method can perform. This happens in order to allow for further development possibilities, such as increasing the number of types of graphs available to the user.

The main FFT method, available in the *FFT algorithm* section of the simpleFFT.java class, is the direct implementation of the mathematical Fourier transform formula. The output of the transform is customized by some output methods that further refine the result, such as the *FFT algorithm for obtaining magnitude values* which is currently used to generate audio frequency values.

The auxiliary methods mentioned before are implemented in the temporary.java class. The algorithms involved deal with extracting audio data from files and byte processing.

An example implementation for extracting audio data can be found in the *method for obtaining sound bytes from file* section of the temporary.java class. This method extracts the audio bytes in a audioInputStream type object, which is specific to the internal Java sound library. Furthermore, it converts these values from byte to integer type, in order to enable the rest of the application to work with them.

Similarly, the byte manipulation algorithms are located in the sections: *method for converting an integer array to a byte array*, *method for converting a byte array to an integer array*, *method for converting an integer number to a byte array*, *method for converting a byte array into an integer number*, *method for converting a short number to a byte array* and *method for converting a byte array into a short number*. They are responsible for various conversions that the program needs to do in order to fulfill the other described functionality.

Effects and Filters

Initially, the group wanted to implement as many effect and filters as possible into the application. However, it was later decided to create a small selection out of these and allow for the further development of the other effects to be an extra feature, to be done over time.

The code for the implementation of these effects can be found in the Compressor.java and the graphicalEq.java classes.

Dynamic Range Control (DRC) - Compressor.java

The DRC algorithm is relatively simple in its implementation. It requires two parameters to be given to the method: the compression threshold (compThresh) and the compression ratio (ratio).

The algorithm itself consists of comparing audio sample values to the threshold and based on whether they are lower or higher, process them accordingly. If a sample value is higher, then the method reduces the difference between the threshold and the value by the ratio. If a sample value is lower, then the method amplifies the sample by the ratio.

The result of the algorithm is an array of processed audio byte values.

Graphical Equalizer (EQ) - graphicalEq.java

The equalizer works very similar to the DRC algorithm, the main difference being the number of threshold values. The EQ has three threshold values: treble (hiThreshold), middle (miThreshold) and bass (loThreshold). The values for these variables are not changeable by the user, as they need to be within a specific range, based on how the program processes the sample values.

Furthermore, the EQ has three ratios, or volumes, one for each of the thresholds. The values for these variables are picked up from the sliders present on this specific effect's window and are changeable by the user.

The algorithm itself works by detecting in which frequency range a sample is and then processing it by multiplication with the respective ratio (volume). This method usually amplifies the sound, but in select cases, it can also reduce the overall volume.

Conclusion

The testing of the software did not prove it to be able to meet the criteria of success. Although it *did* indeed improve the test scores over the control group, the difference was marginal.

While the concept of the product might be able to yield the desired result with the sufficient amount of improvement and further development, in its current state of development, it has not been able it make a strong enough impact on the test groups to be considered at viable solution to the projects problem.

Further Development

In its current state of development, the software is more a Hi-Fidelity prototype than a finished, distributable product. In order for the product to meet its originally design specifications, several features have yet to be implemented.

Designed Textures

In accordance with its design, the textures made for the software need to be applied to give it a more appealing appearance and make it less boring to look at. The textures are also meant to communicate professionalism and reliability to the user.

More effects implemented

The software currently doesn't have all the intended mechanics incorporated. There are several sound processing methods associated with the Sound and Music Computing course and ultimately all of these should be implemented to give the software practical usage. But at this point only Dynamic Range Compression has been implemented.

Dynamic Theory-display

It might also be easier for the user to understand the theory of the different effects if it were possible to see the variables of the mathematical formulas change in accordance with the user input. Taking Dynamic Range Compression as an example, imagine the mathematical formula for compression displayed on the screen with the separate variables changing as the user moves the respective sliders.

This way, they would have both a visual illustration (the equalizer graph) of how the sound is affected by the compression formula's variables – threshold and rate of compression – and quite possibly gain a better understanding of how the process works.

Quiz-mode

But at the moment, the software does not have a way of testing the user's knowledge of Sound and Music Computing, which would be a useful feature for examination preparations, for instance.

This should take the form of a quiz-mode that generates different kinds of exercises for the user to complete. The quiz feature should be able to generate these randomly to avoid exhausting a limited pool of pre-made exercises.

Smartphone-port

Originally the software has been designed for Smartphones, but since it was both developed and tested on PC, it still needs to be ported to Smartphone. But since the software is coded in Java – the same language also running Android-operated Smartphones – this port should be relatively easy.

Also, since the software in its entirety is developed for Smartphones, everything from layout to control mechanics of the User Interface is designed and suited for Smarthphone-usage.

Tutorial

In order for the user to gain a clear understanding of how to operate the software, it will necessary to implement a tutorial to guide them through the different features. The tutorial screen should be displayed the first time the software starts up, after which it should only be displayed when the user presses a 'help' button to call the tutorial forth.

Bibliography

Dkuni. (2013, may 27th). Retrieved from http://www.dkuni.dk/~/media/Files/Publikationer/Tal%20om%20de%20danske%20universiteter%202011.ashx

Edpsycinteractive. (2013, may 27th). Retrieved from http://www.edpsycinteractive.org/topics/motivation/motivate.html

Larsen, L. B. (2013, may 27th). Aalborg.

learning-styles-online. (2013, may 27th). Retrieved from http://www.learning-styles-online.com/overview/

Nap. (2013, may 27th). Retrieved from http://www.nap.edu/html/biomems/cjohnson.pdf

Nightlifescience. (2013, may 27th). Retrieved from http://nightlifescience.files.wordpress.com/2011/10/minerva-model.png?w=1024&h=729

Nwlink. (2013, may 27th). Retrieved from http://www.nwlink.com/~donclark/hrd/styles/kolb.html

Nwlink. (2013, may 27th). Retrieved from http://www.nwlink.com/~donclark/hrd/styles/vakt.html

Podcasting. (2013, may 27th). Retrieved from http://podcasting.about.com/od/recordingequipment/fr/deepaud.htm

Sourceforge. (2013, may 27th). Retrieved from http://sourceforge.net/projects/pure-data/reviews/

Sourceforge. (2013, may 27th). Retrieved from http://sourceforge.net/projects/pure-data/reviews/

Tal. (2013, may 27th). Retrieved from http://tal.ku.dk/studerende/F_studerende/

Java. (2013, may 28th). Retrieved from http://www.java.com/en/download/whatis_java.jsp

Appendix

The Code

maybeMain.java

```
*import packages*
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JSlider;
import javax.swing.JTabbedPane;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.SwingConstants;
import javax.swing.SwingWorker;
import javax.swing.border.LineBorder;
import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JLabel;
import javax.swing.BoxLayout;


*class body*
public class maybeMain {
	*create soundClass object*
	soundClass mWindow = new soundClass();


	*declare interface components*
```

```
private JFrame frmDspBeatbox;

JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.BOTTOM);

JPanel mainWindow = new JPanel();

JButton btnRecord = new JButton("Record");

JButton btnStop = new JButton("Stop");

JButton btnPlay = new JButton("Play");

JButton btnSound1 = new JButton("Sound1");

JButton btnSound2 = new JButton("Sound2");

JButton btnSound3 = new JButton("Sound3");

JButton btnSound4 = new JButton("Sound4");

JButton btnSound5 = new JButton("Sound5");

JButton btnSound6 = new JButton("Sound6");

JButton btnSound7 = new JButton("Sound7");

JButton btnSound8 = new JButton("Sound8");

JButton btnSound9 = new JButton("Sound9");

JPanel panelTray = new JPanel();

JLabel lblSound1 = new JLabel("");

JLabel lblSound2 = new JLabel("");

JLabel lblSound3 = new JLabel("");

JLabel lblSound4 = new JLabel("");

JLabel lblSound5 = new JLabel("");

JLabel lblSound6 = new JLabel("");

JLabel lblSound7 = new JLabel("");

JLabel lblSound8 = new JLabel("");

JLabel lblSound9 = new JLabel("");

JPanel effectWindow1 = new JPanel();

JButton btnStop1 = new JButton("Stop");

JButton btnPlay1 = new JButton("Play");
```

```
JSlider compThresh = new JSlider();

JSlider compRatio = new JSlider();

JLabel lblSoundSlotSel = new JLabel("Sound slot label");

JLabel lblCompThresh = new JLabel("Threshold");

JLabel lblCompRatio = new JLabel("Ratio");

JButton btnOn = new JButton("On");

JButton btnOff = new JButton("Off");

JLabel lblStatus = new JLabel("Effect is Off");

JPanel vizPanel = new JPanel();

JPanel effectWindow2 = new JPanel();

JButton btnStop2 = new JButton("Stop");

JButton btnPlay2 = new JButton("Play");

JButton btnOn1 = new JButton("On");

JButton btnOff1 = new JButton("Off");

JLabel lblSoundSlotSel1 = new JLabel("Sound slot label");

JLabel lblStatus1 = new JLabel("Effect is Off");

JSlider gainLow = new JSlider();

JSlider gainMid = new JSlider();

JSlider gainHi = new JSlider();

JLabel lblBass = new JLabel("Low");

JLabel lblMiddle = new JLabel("Middle");

JLabel lblTreble = new JLabel("High");

JPanel vizPanel1 = new JPanel();

JPanel effectWindow3 = new JPanel();

JButton btnStop3 = new JButton("Stop");

JButton btnPlay3 = new JButton("Play");

JButton btnOn2 = new JButton("On");

JButton btnOff2 = new JButton("Off");
```

```
JLabel lblSoundSlotSel2 = new JLabel("Sound slot label");

JLabel lblStatus2 = new JLabel("Effect is Off");

JPanel vizPanel2 = new JPanel();

JPanel effectWindow4 = new JPanel();

JButton btnStop4 = new JButton("Stop");

JButton btnPlay4 = new JButton("Play");

JButton btnOn3 = new JButton("On");

JButton btnOff3 = new JButton("Off");

JLabel lblSoundSlotSel3 = new JLabel("Sound slot label");

JLabel lblStatus3 = new JLabel("Effect is Off");

JPanel vizPanel3 = new JPanel();


*declare flag and counter variables*

int soundSlot = 0;

int trayIndex = 1;

int compEff = 0;

int eqEff = 0;


*method that runs the application*

public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

                public void run() {

                        try {

                                maybeMain window = new maybeMain();

                                window.frmDspBeatbox.setVisible(true);

                        } catch (Exception e) {

                                e.printStackTrace();

                        }
```

```
                }
        });
}

*constructor method that contains the interaction control methods*
public maybeMain() {
        initialize();

        *action listeners*
        btnPlay.addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent e) {
                        SwingWorker worker = new SwingWorker<String, Void>() {
                          @Override
                          public String doInBackground() {
                            mWindow.play(soundSlot);
                            return "";
                          }
                        };
                        worker.execute();
                }
        });

        btnStop.addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent e) {
                        SwingWorker worker = new SwingWorker<String, Void>() {
                          @Override
```

```
                    public String doInBackground() {
                        mWindow.stop();
                        return "";
                    }
                };
                worker.execute();
            }
        });

        btnRecord.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                SwingWorker worker = new SwingWorker<String, Void>() {
                    @Override
                    public String doInBackground() {
                        mWindow.record();
                        return "";
                    }
                };
                worker.execute();
            }
        });

        btnSound1.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                soundSlot = 1;
                setLabelSoundSel(soundSlot);
```

```
            }
        });

        btnSound2.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                soundSlot = 2;
                setLabelSoundSel(soundSlot);
            }
        });

        btnSound3.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                soundSlot = 3;
                setLabelSoundSel(soundSlot);
            }
        });

        btnSound4.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                soundSlot = 4;
                setLabelSoundSel(soundSlot);
            }
        });

        btnSound5.addMouseListener(new MouseAdapter() {
```

```
		@Override
		public void mouseClicked(MouseEvent e) {
			soundSlot = 5;
			setLabelSoundSel(soundSlot);
		}
	});

	btnSound6.addMouseListener(new MouseAdapter() {
		@Override
		public void mouseClicked(MouseEvent e) {
			soundSlot = 6;
			setLabelSoundSel(soundSlot);
		}
	});

	btnSound7.addMouseListener(new MouseAdapter() {
		@Override
		public void mouseClicked(MouseEvent e) {
			soundSlot = 7;
			setLabelSoundSel(soundSlot);
		}
	});

	btnSound8.addMouseListener(new MouseAdapter() {
		@Override
		public void mouseClicked(MouseEvent e) {
			soundSlot = 8;
			setLabelSoundSel(soundSlot);
```

```
		}
	});


	btnSound9.addMouseListener(new MouseAdapter() {
		@Override
		public void mouseClicked(MouseEvent e) {
			soundSlot = 9;
			setLabelSoundSel(soundSlot);
		}
	});



	panelTray.addMouseListener(new MouseAdapter() {
		@Override
		public void mouseClicked(MouseEvent arg0) {
			if(soundSlot > 0 && soundSlot < 10)
				{mWindow.addToTray(soundSlot);
				 if (trayIndex == 10) trayIndex = 1;
				 drawOnTray(trayIndex++).setText("Sound" + soundSlot);
				 soundSlot = 0;
				}
			else {soundSlot = 10; setLabelSoundSel(soundSlot);}
		}
	});


	btnPlay1.addMouseListener(new MouseAdapter() {
		@Override
		public void mouseClicked(MouseEvent e) {
```

```
                    SwingWorker worker = new SwingWorker<String, Void>() {
                      @Override
                      public String doInBackground() {
                        if (compEff == 0) mWindow.play(soundSlot);
                        else mWindow.playWithCompression(mWindow.selSoundSlot(soundSlot),
                            compThresh.getValue(), compRatio.getValue());
                        return "";
                      }
                    };
                    worker.execute();
            }
        });

        btnStop1.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                    SwingWorker worker = new SwingWorker<String, Void>() {
                      @Override
                      public String doInBackground() {
                        mWindow.stop();
                        return "";
                      }
                    };
                    worker.execute();
            }
        });

        btnOn.addMouseListener(new MouseAdapter() {
```

```
                @Override
                public void mouseClicked(MouseEvent e) {
                        compEff = 1;
                        lblStatus.setText("Effect is On");
                        vizPanel.removeAll();
                        WaveformGraph graph = new
WaveformGraph(mWindow.selSoundSlot(soundSlot), 1,
        compThresh.getValue(), compRatio.getValue());
                        graph.setBounds(15, 232, 258, 141);
                        vizPanel.add(graph);
                        vizPanel.revalidate();
                }
        });

        btnOff.addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent e) {
                        compEff = 0;
                        lblStatus.setText("Effect is Off");
                        vizPanel.removeAll();
                        WaveformGraph graph = new
WaveformGraph(mWindow.selSoundSlot(soundSlot), 0);
                        graph.setBounds(15, 232, 258, 141);
                        vizPanel.add(graph);
                        vizPanel.revalidate();
                }
        });

        btnPlay2.addMouseListener(new MouseAdapter() {
```

```
            @Override
            public void mouseClicked(MouseEvent e) {
                SwingWorker worker = new SwingWorker<String, Void>() {
                  @Override
                  public String doInBackground() {
                      if (eqEff == 0) mWindow.play(soundSlot);
                    else mWindow.playWithEq(mWindow.selSoundSlot(soundSlot),
                            gainLow.getValue(), gainMid.getValue(), gainHi.getValue());
                    return "";
                  }
                };
                worker.execute();
            }
        });

        btnStop2.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                SwingWorker worker = new SwingWorker<String, Void>() {
                  @Override
                  public String doInBackground() {
                    mWindow.stop();
                    return "";
                  }
                };
                worker.execute();
            }
        });
```

```
		btnOn1.addMouseListener(new MouseAdapter() {
			@Override
			public void mouseClicked(MouseEvent e) {
				eqEff = 1;
				lblStatus1.setText("Effect is On");
				vizPanel1.removeAll();
				WaveformGraph graph = new
WaveformGraph(mWindow.selSoundSlot(soundSlot), 2,
	gainLow.getValue(), gainMid.getValue(), gainHi.getValue());
				graph.setBounds(15, 232, 258, 141);
				vizPanel1.add(graph);
				vizPanel1.revalidate();
			}
		});

		btnOff1.addMouseListener(new MouseAdapter() {
			@Override
			public void mouseClicked(MouseEvent e) {
				eqEff = 0;
				lblStatus1.setText("Effect is Off");
				vizPanel1.removeAll();
				WaveformGraph graph = new
WaveformGraph(mWindow.selSoundSlot(soundSlot), 0);
				graph.setBounds(15, 232, 258, 141);
				vizPanel1.add(graph);
				vizPanel1.revalidate();
			}
		});
```

```
btnPlay3.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
                SwingWorker worker = new SwingWorker<String, Void>() {
                  @Override
                  public String doInBackground() {
                    mWindow.play(soundSlot);
                    return "";
                  }
                };
                worker.execute();
        }
});

btnStop3.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
                SwingWorker worker = new SwingWorker<String, Void>() {
                  @Override
                  public String doInBackground() {
                    mWindow.stop();
                    return "";
                  }
                };
                worker.execute();
        }
});
```

```
btnPlay4.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
                SwingWorker worker = new SwingWorker<String, Void>() {
                  @Override
                  public String doInBackground() {
                    mWindow.play(soundSlot);
                    return "";
                  }
                };
                worker.execute();
        }
});

btnStop4.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
                SwingWorker worker = new SwingWorker<String, Void>() {
                  @Override
                  public String doInBackground() {
                    mWindow.stop();
                    return "";
                  }
                };
                worker.execute();
        }
});
```

```
}

*clarification method that shows the user what sound is selected*
private void setLabelSoundSel(int soundSlot)
{
	if (soundSlot != 10)
		{lblSoundSlotSel.setText("Sound " + soundSlot + " selected");
		 lblSoundSlotSel1.setText("Sound " + soundSlot + " selected");
		 lblSoundSlotSel2.setText("Sound " + soundSlot + " selected");
		 lblSoundSlotSel3.setText("Sound " + soundSlot + " selected");
		}
	else
		{lblSoundSlotSel.setText("Sound tray selected");
		 lblSoundSlotSel1.setText("Sound tray selected");
		 lblSoundSlotSel2.setText("Sound tray selected");
		 lblSoundSlotSel3.setText("Sound tray selected");
		}
}

*method that operates the display of sound names in the Tray*
private JLabel drawOnTray(int soundSlot)
{
	switch(soundSlot) {
	case 1: return lblSound1;
	case 2: return lblSound2;
	case 3: return lblSound3;
	case 4: return lblSound4;
	case 5: return lblSound5;
```

```
            case 6: return lblSound6;
            case 7: return lblSound7;
            case 8: return lblSound8;
            case 9: return lblSound9;
            default: return lblSound1;
            }
    }

    *method that initializes the GUI components and sets their properties*
    private void initialize() {
            frmDspBeatbox = new JFrame();
            frmDspBeatbox.setTitle("DSP Beat Box");
            frmDspBeatbox.setResizable(false);
            frmDspBeatbox.setBounds(100, 100, 300, 450);
            frmDspBeatbox.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frmDspBeatbox.getContentPane().setLayout(null);

            tabbedPane.setBounds(0, 0, 294, 412);
            frmDspBeatbox.getContentPane().add(tabbedPane);
            tabbedPane.addTab("Main", null, mainWindow, null);
            mainWindow.setLayout(null);

            btnRecord.setBounds(16, 11, 75, 37);
            mainWindow.add(btnRecord);
            btnStop.setBounds(198, 11, 75, 37);
            mainWindow.add(btnStop);
            btnPlay.setBounds(107, 11, 75, 37);
            mainWindow.add(btnPlay);
```

```
btnSound1.setBounds(16, 71, 75, 37);
mainWindow.add(btnSound1);
btnSound2.setBounds(107, 71, 75, 37);
mainWindow.add(btnSound2);
btnSound3.setBounds(198, 71, 75, 37);
mainWindow.add(btnSound3);
btnSound4.setBounds(16, 119, 75, 37);
mainWindow.add(btnSound4);
btnSound5.setBounds(107, 119, 75, 37);
mainWindow.add(btnSound5);
btnSound6.setBounds(198, 119, 75, 37);
mainWindow.add(btnSound6);
btnSound7.setBounds(16, 167, 75, 37);
mainWindow.add(btnSound7);
btnSound8.setBounds(107, 167, 75, 37);
mainWindow.add(btnSound8);
btnSound9.setBounds(198, 167, 75, 37);
mainWindow.add(btnSound9);

panelTray.setBorder(new LineBorder(new Color(0, 0, 0), 2));
panelTray.setBounds(12, 215, 264, 136);
mainWindow.add(panelTray);
panelTray.setLayout(null);
panelTray.setBackground(Color.WHITE);

lblSound1.setBounds(10, 7, 46, 35);
panelTray.add(lblSound1);
```

```
lblSound2.setBounds(108, 7, 46, 35);
panelTray.add(lblSound2);
lblSound3.setBounds(208, 7, 46, 35);
panelTray.add(lblSound3);
lblSound4.setBounds(10, 49, 46, 35);
panelTray.add(lblSound4);
lblSound5.setBounds(109, 49, 46, 35);
panelTray.add(lblSound5);
lblSound6.setBounds(208, 49, 46, 35);
panelTray.add(lblSound6);
lblSound7.setBounds(10, 91, 46, 35);
panelTray.add(lblSound7);
lblSound8.setBounds(109, 91, 46, 35);
panelTray.add(lblSound8);
lblSound9.setBounds(208, 91, 46, 35);
panelTray.add(lblSound9);

tabbedPane.addTab("DRC", null, effectWindow1, null);
effectWindow1.setLayout(null);
compThresh.setMinimum(1);
compThresh.setMaximum(1000000000);

compThresh.setBounds(15, 90, 175, 23);
effectWindow1.add(compThresh);
compRatio.setMaximum(100000000);
compRatio.setMinimum(1);
compRatio.setBounds(15, 141, 175, 23);
effectWindow1.add(compRatio);
```

```
btnPlay1.setBounds(41, 11, 83, 43);
effectWindow1.add(btnPlay1);
btnStop1.setBounds(165, 11, 83, 43);
effectWindow1.add(btnStop1);
btnOn.setBounds(15, 187, 47, 34);
effectWindow1.add(btnOn);
btnOff.setBounds(70, 187, 49, 34);
effectWindow1.add(btnOff);

lblSoundSlotSel.setHorizontalAlignment(SwingConstants.CENTER);
lblSoundSlotSel.setBounds(65, 65, 159, 14);
effectWindow1.add(lblSoundSlotSel);
lblCompThresh.setHorizontalAlignment(SwingConstants.LEFT);
lblCompThresh.setBounds(200, 79, 73, 34);
effectWindow1.add(lblCompThresh);
lblCompRatio.setHorizontalAlignment(SwingConstants.LEFT);
lblCompRatio.setBounds(200, 141, 73, 34);
effectWindow1.add(lblCompRatio);
lblStatus.setHorizontalAlignment(SwingConstants.CENTER);
lblStatus.setBounds(184, 197, 89, 14);
effectWindow1.add(lblStatus);

vizPanel.setBackground(Color.WHITE);
vizPanel.setBorder(new LineBorder(new Color(0, 0, 0), 2));
vizPanel.setBounds(15, 232, 258, 141);
effectWindow1.add(vizPanel);
vizPanel.setLayout(new BoxLayout(vizPanel, BoxLayout.X_AXIS));
```

```
tabbedPane.addTab("EQ", null, effectWindow2, null);
effectWindow2.setLayout(null);

btnPlay2.setBounds(41, 11, 83, 43);
effectWindow2.add(btnPlay2);
btnStop2.setBounds(165, 11, 83, 43);
effectWindow2.add(btnStop2);
btnOn1.setBounds(15, 187, 47, 34);
effectWindow2.add(btnOn1);
btnOff1.setBounds(70, 187, 49, 34);
effectWindow2.add(btnOff1);

lblSoundSlotSel1.setHorizontalAlignment(SwingConstants.CENTER);
lblSoundSlotSel1.setBounds(74, 65, 141, 14);
effectWindow2.add(lblSoundSlotSel1);
lblStatus1.setHorizontalAlignment(SwingConstants.CENTER);
lblStatus1.setBounds(184, 197, 89, 14);
effectWindow2.add(lblStatus1);
lblBass.setHorizontalAlignment(SwingConstants.CENTER);
lblBass.setBounds(227, 90, 46, 23);
effectWindow2.add(lblBass);
lblMiddle.setHorizontalAlignment(SwingConstants.CENTER);
lblMiddle.setBounds(227, 120, 46, 23);
effectWindow2.add(lblMiddle);
lblTreble.setHorizontalAlignment(SwingConstants.CENTER);
lblTreble.setBounds(225, 150, 46, 23);
effectWindow2.add(lblTreble);
```

```
gainLow.setMaximum(10000);
gainLow.setMinimum(1);

gainLow.setBounds(15, 90, 200, 23);
effectWindow2.add(gainLow);
gainMid.setMaximum(10000);
gainMid.setMinimum(1);
gainMid.setBounds(15, 120, 200, 23);
effectWindow2.add(gainMid);
gainHi.setMaximum(10000);
gainHi.setMinimum(1);
gainHi.setBounds(15, 150, 200, 23);
effectWindow2.add(gainHi);

vizPanel1.setBackground(Color.WHITE);
vizPanel1.setBorder(new LineBorder(new Color(0, 0, 0), 2));
vizPanel1.setBounds(15, 232, 258, 141);
effectWindow2.add(vizPanel1);
vizPanel1.setLayout(new BoxLayout(vizPanel1, BoxLayout.X_AXIS));

tabbedPane.addTab("Echo", null, effectWindow3, null);
effectWindow3.setLayout(null);

btnPlay3.setBounds(41, 11, 83, 43);
effectWindow3.add(btnPlay3);
btnStop3.setBounds(165, 11, 83, 43);
effectWindow3.add(btnStop3);
btnOn2.setBounds(15, 187, 47, 34);
```

```
effectWindow3.add(btnOn2);
btnOff2.setBounds(70, 187, 49, 34);
effectWindow3.add(btnOff2);

lblSoundSlotSel2.setHorizontalAlignment(SwingConstants.CENTER);
lblSoundSlotSel2.setBounds(74, 65, 141, 14);
effectWindow3.add(lblSoundSlotSel2);
lblStatus2.setHorizontalAlignment(SwingConstants.CENTER);
lblStatus2.setBounds(184, 197, 89, 14);
effectWindow3.add(lblStatus2);

vizPanel2.setBackground(Color.WHITE);
vizPanel2.setBorder(new LineBorder(new Color(0, 0, 0), 2));
vizPanel2.setBounds(15, 232, 258, 141);
effectWindow3.add(vizPanel2);

tabbedPane.addTab("Flanger", null, effectWindow4, null);
effectWindow4.setLayout(null);

btnPlay4.setBounds(41, 11, 83, 43);
effectWindow4.add(btnPlay4);
btnStop4.setBounds(165, 11, 83, 43);
effectWindow4.add(btnStop4);
btnOn3.setBounds(15, 187, 47, 34);
effectWindow4.add(btnOn3);
btnOff3.setBounds(70, 187, 49, 34);
effectWindow4.add(btnOff3);
```

```
                lblSoundSlotSel3.setHorizontalAlignment(SwingConstants.CENTER);
                lblSoundSlotSel3.setBounds(74, 65, 141, 14);
                effectWindow4.add(lblSoundSlotSel3);
                lblStatus3.setHorizontalAlignment(SwingConstants.CENTER);
                lblStatus3.setBounds(184, 197, 89, 14);
                effectWindow4.add(lblStatus3);

                vizPanel3.setBackground(Color.WHITE);
                vizPanel3.setBorder(new LineBorder(new Color(0, 0, 0), 2));
                vizPanel3.setBounds(15, 232, 258, 141);
                effectWindow4.add(vizPanel3);
        }

}
```

soundClass.java

```
*import packages*
import java.io.ByteArrayOutputStream;
import java.io.File;
import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.TargetDataLine;
```

```
*class body*

public class soundClass{


        *variable/object declarations*

        AudioFormat             audioFormat = new AudioFormat(AudioFormat.Encoding.PCM_SIGNED, 44100.0F,
                                        16, 2, 4, 44100.0F, false);

        TargetDataLine          lineIn = null;

        SourceDataLine          lineOut = null;

        DataLine.Info           infoIn = new DataLine.Info(TargetDataLine.class, audioFormat);

        DataLine.Info           infoOut = new DataLine.Info(SourceDataLine.class, audioFormat);


        temporary temp = new temporary();


        int[] Tray = new int[9];

        int index = 0;

        int soundSlot = 0;


        *method that sets the paths for the sounds*

        public String selSoundSlot(int soundSlot)

        {

                String path="";

                switch (soundSlot) {

                case 1: path = "\\sound1.wav"; break;

                case 2: path = "\\sound2.wav"; break;

                case 3: path = "\\sound3.wav"; break;

                case 4: path = "\\sound4.wav"; break;

                case 5: path = "\\sound5.wav"; break;

                case 6: path = "\\sound6.wav"; break;
```

```
            case 7: path = "\\sound7.wav"; break;
            case 8: path = "\\sound8.wav"; break;
            case 9: path = "\\sound9.wav"; break;
            default:{}
            }
            return path;
}

*method that helps the automatization of the record method*
private int recordCounter()
{
            if (soundSlot == 9)          soundSlot = 0;
            soundSlot++;
            return soundSlot;
}

*record input method*
public void record()
{
Thread stopper = new Thread(new Runnable() {
public void run() {
try {
            Thread.sleep(5000);
} catch (InterruptedException e) {}
lineIn.stop();
lineIn.close();
}
});
```

```
            try     {
                    lineIn = (TargetDataLine) AudioSystem.getLine(infoIn);
                    lineIn.open(audioFormat);
                    lineIn.start();
                    stopper.start();
                    AudioSystem.write(new AudioInputStream(lineIn), AudioFileFormat.Type.WAVE, new
                            File(selSoundSlot(recordCounter())));
            }       catch (Exception e)         {}
}

*input/output stop method*
public void stop()
{
            try     {
                    lineIn.stop();
                    lineIn.close();
                    lineOut.stop();
                    lineOut.close();
            }       catch (Exception e){};
}

*output play method*
public void play(int soundSlot)
{
            byte[] audioData;
            if (soundSlot == 10)
                    {
                            ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
```

```
                    for(int i=0;i<index;i++)
                        outputStream.write(temp.reader(selSoundSlot(Tray[i])), 0,
                        temp.reader(selSoundSlot(Tray[i])).length);
                    audioData =  outputStream.toByteArray();
                }
            else audioData = temp.reader(selSoundSlot(soundSlot));
            try {
                lineOut = (SourceDataLine) AudioSystem.getLine(infoOut);
                lineOut.open(audioFormat);
            }       catch (Exception e) {}
            lineOut.start();
            lineOut.write(audioData, 0, audioData.length);
            lineOut.drain();
            stop();
    }

    *method for adding elements to the Tray*
    public void addToTray(int soundSlot)
    {
            if(index == 9) index = 0;
            Tray[index++] = soundSlot;
    }

    *output play method with DRC effect*
    public void playWithCompression(String path, int compThresh, int ratio)
    {
            Compressor comp = new Compressor();
            int[] audioData = comp.compress(path, compThresh, ratio);
```

```
                byte[] toPlay = temp.intToByteAudioDataArray(audioData);
                try {
                        lineOut = (SourceDataLine) AudioSystem.getLine(infoOut);
                        lineOut.open(audioFormat);
                }       catch (Exception e) {}
                lineOut.start();
                lineOut.write(toPlay, 0, toPlay.length);
                lineOut.drain();
                stop();
        }

        *output play method with equalizer effect*
        public void playWithEq(String path, int gainLow, int gainMid, int gainHi)
        {
                graphicEqual eq = new graphicEqual(path, gainLow, gainMid, gainHi);
                int[] audioData = eq.applyEqual();
                byte[] toPlay = temp.intToByteAudioDataArray(audioData);
                try {
                        lineOut = (SourceDataLine) AudioSystem.getLine(infoOut);
                        lineOut.open(audioFormat);
                }       catch (Exception e) {}
                lineOut.start();
                lineOut.write(toPlay, 0, toPlay.length);
                lineOut.drain();
                stop();
        }
}
```

temporary.java

```
*import packages*
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;

*class body*
public class temporary {
        *variable declaration*
   private int myChunkSize;
   private int mySubChunk1Size;
   private short myFormat;
   private short myChannels;
   private int mySampleRate;
   private int myByteRate;
   private short myBlockAlign;
   private short myBitsPerSample;
   private int myDataSize;

   public byte[] myData;
```

```
        *method for obtaining WAV file header and sound bytes from a file*

        public void read(String path) {

DataInputStream inFile = null;

byte[] tmpInt = new byte[4];

byte[] tmpShort = new byte[2];


try {

   inFile = new DataInputStream(new FileInputStream(path));


   String chunkID = "" + (char) inFile.readByte() + (char) inFile.readByte() + (char) inFile.readByte() + (char)
      inFile.readByte();


   inFile.read(tmpInt);

   myChunkSize = byteArrayToInt(tmpInt);


   String format = "" + (char) inFile.readByte() + (char) inFile.readByte() + (char) inFile.readByte() + (char)
      inFile.readByte();


   String subChunk1ID = "" + (char) inFile.readByte() + (char) inFile.readByte() + (char) inFile.readByte() + (char)
      inFile.readByte();


   inFile.read(tmpInt);

   mySubChunk1Size = byteArrayToInt(tmpInt);


   inFile.read(tmpShort);

   myFormat = byteArrayToShort(tmpShort);


   inFile.read(tmpShort);

   myChannels = byteArrayToShort(tmpShort);
```

```
      inFile.read(tmpInt);
      mySampleRate = byteArrayToInt(tmpInt);

      inFile.read(tmpInt);
      myByteRate = byteArrayToInt(tmpInt);

      inFile.read(tmpShort);
      myBlockAlign = byteArrayToShort(tmpShort);

      inFile.read(tmpShort);
      myBitsPerSample = byteArrayToShort(tmpShort);

      String dataChunkID = "" + (char) inFile.readByte() + (char) inFile.readByte() + (char) inFile.readByte() + (char)
         inFile.readByte();

      inFile.close();

      myData = reader(path);
      myDataSize = myData.length;
    }
   catch (Exception e) {e.printStackTrace();}
}

         *method for writing a WAV file header and sound bytes to a file*
         public void save(String path) {
   try {
      DataOutputStream outFile = new DataOutputStream(new FileOutputStream(path));
```

```
      outFile.writeBytes("RIFF");
      outFile.write(intToByteArray((int) myChunkSize), 0, 4);

      outFile.writeBytes("WAVE");
      outFile.writeBytes("fmt ");
      outFile.write(intToByteArray((int) mySubChunk1Size), 0, 4);

      outFile.write(shortToByteArray((short) myFormat), 0, 2);

      outFile.write(shortToByteArray((short) myChannels), 0, 2);

      outFile.write(intToByteArray((int) mySampleRate), 0, 4);

      outFile.write(intToByteArray((int) myByteRate), 0, 4);

      outFile.write(shortToByteArray((short) myBlockAlign), 0, 2);

      outFile.write(shortToByteArray((short) myBitsPerSample), 0, 2);

      outFile.writeBytes("data");
      outFile.write(intToByteArray((int) myDataSize), 0, 4);

      outFile.write(myData);
      outFile.close();
    }
    catch (Exception e) {e.printStackTrace();}
  }
```

```
*method for obtaining sound bytes from file*

public byte[] reader(String path)

{

        AudioInputStream        audioInputStream = null;

        int     nBytesRead = 0;


        try             {audioInputStream = AudioSystem.getAudioInputStream(new File(path));}

        catch (Exception e)

        {

                e.printStackTrace();

                System.exit(1);

        }


        byte[]  abData = new byte[(int) (audioInputStream.getFrameLength()) *
                (audioInputStream.getFormat().getFrameSize())];


        while (nBytesRead != -1)

        {

                try                                                     {nBytesRead =
                        audioInputStream.read(abData, 0, abData.length); }

                catch (IOException e)       {e.printStackTrace();}

        }

        return abData;

}


*method for converting an integer array to a byte array*

public byte[] intToByteAudioDataArray (int[] inputData)

{
```

```
            byte[] toReturn = new byte[inputData.length*4];
            byte[] tmpInt = new byte[4];
            int audioByte = 0;
            for (int index = 0; index < inputData.length; index++)
            {
            tmpInt = intToByteArray(inputData[index]);
            toReturn[audioByte + 0] = tmpInt[0];
            toReturn[audioByte + 1] = tmpInt[1];
            toReturn[audioByte + 2] = tmpInt[2];
            toReturn[audioByte + 3] = tmpInt[3];
            audioByte += 4;
        }
    return toReturn;
}

*method for converting a byte array to an integer array*
public int[] byteToIntAudioDataArray(byte[] inputByteArray)
{
    int[] toReturn = new int[inputByteArray.length / 4];
    byte[] tmpInt = new byte[4];
    int index = 0;
    for (int audioByte = 0; audioByte < inputByteArray.length; audioByte += 4)
        {
                    tmpInt[0] = inputByteArray[audioByte + 0];
                    tmpInt[1] = inputByteArray[audioByte + 1];
                    tmpInt[2] = inputByteArray[audioByte + 2];
                    tmpInt[3] = inputByteArray[audioByte + 3];
                    toReturn[index++] = byteArrayToInt(tmpInt);
```

```
            }
            return toReturn;
        }

        *method for converting an integer number to a byte array*
public byte[] intToByteArray(int x) {
    ByteBuffer buffer = ByteBuffer.allocate(4);
    buffer.putInt(x);
    return buffer.array();
}

        *method for converting a byte array into an integer number*
public int byteArrayToInt(byte[] bytes) {
    ByteBuffer buffer = ByteBuffer.allocate(4);
    buffer.put(bytes);
    buffer.flip();
    return buffer.getInt();
}

        *method for converting a short number to a byte array*
public byte[] shortToByteArray(short x) {
    ByteBuffer buffer = ByteBuffer.allocate(2);
    buffer.putShort(x);
    return buffer.array();
}

        *method for converting a byte array into a short number*
public short byteArrayToShort(byte[] bytes) {
```

```
    ByteBuffer buffer = ByteBuffer.allocate(2);
    buffer.put(bytes);
    buffer.flip();
    return buffer.getShort();
  }

}
```

simpleFFT.java

```
*class body*
public class simpleFFT {

        *variable declaration*
         public static final int FFT_MAGNITUDE = 1;
         public static final int FFT_POWER = 2;
         public static final int FFT_REVERSE = 3;
         private static final double twoPI = 2 * Math.PI;

         private int transformationType;
         private int windowSize;
         private int startWindowFunction;

        *constructor method*
         public simpleFFT(int transformationType, int startWindowFunction, int windowSize)
         {
                this.transformationType = transformationType;
                this.startWindowFunction = startWindowFunction;
```

```
            this.windowSize = windowSize;
    }


*method for converting a integer array to a double array*
    public double[] setFFTdata (int [] inputArray)
    {
            double [] fftData = new double [inputArray.length];
            for (int i=0; i<inputArray.length; i++)
                        fftData[i] = (double) inputArray[i];
            return fftData;
    }


*method for converting a double array to a integer array*
    public int[] setiFFTdata (double [] inputArray)
    {
            int [] fftData = new int [inputArray.length];
            for (int i=0; i<inputArray.length; i++)
                        fftData[i] = (int) inputArray[i];
            return fftData;
    }


*method that controls the FFT algorithm*
    public double [] transform(double[] FFTArray)
    {
            double [] toReturn = new double [FFTArray.length];
            switch(transformationType)
            {
                    case FFT_MAGNITUDE:
```

```
                        toReturn = magnitudeFFT(applyWindowFunction(FFTArray, startWindowFunction,
                                windowSize));

                        break;


                        case FFT_POWER:

                        toReturn = powerFFT(applyWindowFunction(FFTArray, startWindowFunction,
windowSize));

                        break;


                        case FFT_REVERSE:

                        toReturn = reverseFFT(FFTArray);

                        break;

                }

                return toReturn;

        }


        *method that applies a rectangle window function to the FFT algorithm*

        private double [] applyWindowFunction(double[] inputData, int startWindowFunction, int windowSize)

        {

                double [] outputData = new double [windowSize-startWindowFunction+1];

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

                        if (i > startWindowFunction && i < windowSize)        outputData[i] = inputData[i];

                return outputData;

        }


        *FFT algorithm for obtaining power values*

        private double [] powerFFT(double[] real)

        {

                double[] temp = new double [real.length];
```

```
            temp = real;
            double[] imag = new double [real.length];
            fft(temp, imag, -1);
            for (int i = 0; i < temp.length; i++)
                  temp[i] = temp[i] * temp[i] + imag[i] * imag[i];
            return temp;
 }

*FFT algorithm for obtaining magnitude values*
 private double [] magnitudeFFT(double[] real)
 {
            double[] temp = new double [real.length];
            temp = real;
            double[] imag = new double [real.length];
            fft(temp, imag, -1);
            for (int i = 0; i < temp.length; i++)
                  temp[i] = Math.sqrt(temp[i] * temp[i] + imag[i] * imag[i]);
            return temp;
 }

*reverse FFT algorithm*
 private double [] reverseFFT(double[] real)
 {
            double[] temp = new double [real.length];
            temp = real;
            double[] imag = new double [real.length];
            fft(temp, imag, 1);
            return temp;
```

```
}


*FFT algorithm*

private void fft(double[] re, double[] im, int direction)

{

                int n = re.length;

                int bits = (int)Math.rint(Math.log(n) / Math.log(2));


                if (n != (1 << bits)) throw new IllegalArgumentException("FFT data must be power of 2");


                int localN;

                int j = 0;

                for (int i = 0; i < n-1; i++)

                {

                        if (i < j)

                                {

                                double temp = re[j];

                                re[j] = re[i];

                                re[i] = temp;

                                temp = im[j];

                                im[j] = im[i];

                                im[i] = temp;

                                }

                        int k = n / 2;

                        while ((k >= 1) &&  (k - 1 < j))

                        {

                                j = j - k;

                                k = k / 2;
```

```
            }
            j = j + k;
        }

        for(int m = 1; m <= bits; m++)
        {
            localN = 1 << m;
            double Wjk_r = 1;
            double Wjk_i = 0;
            double theta = twoPI / localN;
            double Wj_r = Math.cos(theta);
            double Wj_i = direction * Math.sin(theta);
            int nby2 = localN / 2;
            for (j = 0; j < nby2; j++)
            {
                for (int k = j; k < n; k += localN)
                {
                    int id = k + nby2;
                    double tempr = Wjk_r * re[id] - Wjk_i * im[id];
                    double tempi = Wjk_r * im[id] + Wjk_i * re[id];
                    re[id] = re[k] - tempr;
                    im[id] = im[k] - tempi;
                    re[k] += tempr;
                    im[k] += tempi;
                }
                double wtemp = Wjk_r;
                Wjk_r = Wj_r * Wjk_r  - Wj_i * Wjk_i;
                Wjk_i = Wj_r * Wjk_i  + Wj_i * wtemp;
```

```
                                        }
                                }
                        }
}
```

Compressor.java

```
*class body*
public class Compressor
{
        *object declaration*
        temporary temp = new temporary();

        *DRC algorithm*
        public int[] compress(String path, int compThresh, int ratio)
        {
                int[] audioBytes = temp.byteToIntAudioDataArray(temp.reader(path));
                for (int i=0; i<audioBytes.length; i++)
                        if (audioBytes[i] > compThresh)
                                audioBytes[i] = (audioBytes[i] - compThresh) / (ratio * 100) + compThresh;
                        else if (audioBytes[i] < compThresh)
                                audioBytes[i] = audioBytes[i] / (ratio * 100) + audioBytes[i];
                return audioBytes;
        }
}
```

graphicEqual.java

```
*class body*
public class graphicEqual {
        *variable/object declaration*
        private int hiTreshold = 1000000000;
        private int miTreshold = 100000;
        private int loTreshold = 10;
        private int hiVolume;
        private int miVolume;
        private int loVolume;
        int[] audioBytes;

        temporary temp = new temporary();

        *constructor method*
  public graphicEqual(String path, int hiVolume, int miVolume, int loVolume)
  {
        this.hiVolume = hiVolume;
        this.miVolume = miVolume;
        this.loVolume = loVolume;
        audioBytes = temp.byteToIntAudioDataArray(temp.reader(path));
  }

        *equalizer algorithm*
  public int [] applyEqual()
        {
                int [] outputFrequency = new int [audioBytes.length];
                for(int i=0; i<audioBytes.length; i++)
                {
```

```
                    if (audioBytes[i]>hiTreshold) outputFrequency[i] = hiVolume * audioBytes[i] / 1000000;

                    else if (audioBytes[i]>miTreshold && audioBytes[i]<hiTreshold) outputFrequency[i]
=                   miVolume * audioBytes[i] / 1000000;

                    else if (audioBytes[i]>loTreshold && audioBytes[i]<miTreshold)
                    outputFrequency[i] = loVolume * audioBytes[i] /
1000000;

                    else outputFrequency[i] = audioBytes[i];

            }

            return outputFrequency;

        }

}
```

WaveformGraph.java

```
*import packages*

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import javax.swing.JPanel;


*class body*

public class WaveformGraph extends JPanel {


        *variable/object declaration*

        private String path;

        private int graphId;

        private int compThresh;

        private int ratio;

        private int gainLow;
```

```
        private int gainMid;
        private int gainHi;
        double [] FFTData;

        simpleFFT sfft = new simpleFFT(1, 0, 131072/4-1);
        temporary temp = new temporary();

        *constructor method*
public WaveformGraph(String path, int graphId)
{
        this.path = path;
        this.graphId = graphId;
}

        *constructor method for DRC*
public WaveformGraph(String path, int graphId, int compThresh, int ratio)
{
        this.path = path;
        this.graphId = graphId;
        this.compThresh = compThresh;
        this.ratio = ratio;
}

        *constructor method for equalizer*
public WaveformGraph(String path, int graphId, int gainLow, int gainMid, int gainHi)
{
        this.path = path;
        this.graphId = graphId;
```

```
        this.gainLow = gainLow;
        this.gainMid = gainMid;
        this.gainHi = gainHi;
}


        *graph computing and drawing algorithm*
        public void paintComponent(Graphics g)
        {
                super.paintComponent(g);
          Graphics2D g2d = (Graphics2D) g;
          g2d.setColor(Color.blue);
          int h =  141;
          if(graphId == 0)
          {
                FFTData = sfft.transform(sfft.setFFTdata(temp.byteToIntAudioDataArray(temp.reader(path))));
                for (int i=0; i<FFTData.length; i++)
                        g2d.drawLine(i/129, h, i/129, (int) (h-FFTData[i]/10000000000.0d));
          }
          else if (graphId == 1)
          {
                Compressor comp = new Compressor();
                FFTData = sfft.transform(sfft.setFFTdata(comp.compress(path, compThresh, ratio)));
                for (int i=0; i<FFTData.length; i++)
                        g2d.drawLine(i/129, h, i/129, (int) (h-FFTData[i]/10000000000.0d));
          }
          else if (graphId == 2)
          {
                graphicEqual eq = new graphicEqual(path, gainLow, gainMid, gainHi);
```

```
                        FFTData = sfft.transform(sfft.setFFTdata(eq.applyEqual()));

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

                        g2d.drawLine(i/129, h, i/129, (int) (h-FFTData[i]/10000000000.0d));

            }

          }

}
```

The Hi-Fi Test Questions

Sound & Music Computing

Form Description

A sound track has very loud peaks, making the quieter parts less audible. How would you solve this problem? *

- By compressing the Dynamic Range
- By limiting the audio
- By expanding the Dynamic Range
- By increasing the volume
- By lowering the volume
- Don't know.

Which topics are usually associated with Dynamic Range of audio? *

- [ ] - Compression
- [ ] - Expansion
- [ ] - Limiting
- [ ] - Noise Gates
- [ ] - All of the above
- [ ] - None of the above
- [ ] Don't know.

Which statements are correct? *

- [ ] - Compression decreases audio dynamic range
- [ ] - Limiting determines the maximum volume of the audio
- [ ] - Compression reduces the size of audio files.
- [ ] - Limiters are the same as compression, only with a higher rate of compression
- [ ] - Noise gates only work with analogue audio and must be applied before digital sampling.
- [ ] Don't know.

Let x be the sound input and x0 the threshold. Expansion is effective for: *

- x = x0
- x > x0
- x ≤ x0
- x + x0
- x < x0
- Option 6
- Don't know.

Let x be the sound input and x_0 the threshold. Compression is effective for: *

- x = x0
- x ≥ x0
- x ≤ x0
- x + x0
- x < x0
- Don't know.

A sound track has unwanted noise that needs to be reduced. How is this problem solved? *

- By compressing the Dynamic Range
- By limiting the audio
- By expanding the Dynamic Range
- By increasing the volume
- By lowering the volume
- Don't know

Add item