MARS

a music mixing language
Fall 2006

by Aaron Fernandes
Ritika Virmani
Swapneel Sheth
Michael Sorvillo

Table of Contents

1. Introduction

1.1 Overview

Computer scientists love listening to music. Many are even musicians themselves. But when it comes to creating their own music with computers, it is rather irrational to spend upwards of $600 on software that they will not use professionally. In addition, because of the complexity of these programs, there is a very steep learning curve. But because we all possess programming skills, MARS enables all computer scientists to mix and match different tracks of music, thereby creating their own loop-based musical composition.

MARS is an object oriented programming language with scripting features. The aim of MARS was to create a user-friendly domain specific language that is portable across multiple platforms. It will allow a user who has had any programming experience the ability to mix and match different audio files on top of one another, loop them, and add effects to these tracks.

It gives users a simple way to define their composition logically, so they can easily see how the parts they have defined are interconnected. Because much of the composition process is iterative, MARS allows you to define your parts and then iterate on certain sections of a composition. MARS will allow many computer scientists the ability to create music, even if they don't have much of a "classical" music background. Because MARS is based on predefined loops, users do not need to know music theory or how to read music; all they need is the ability to count and some creativity.

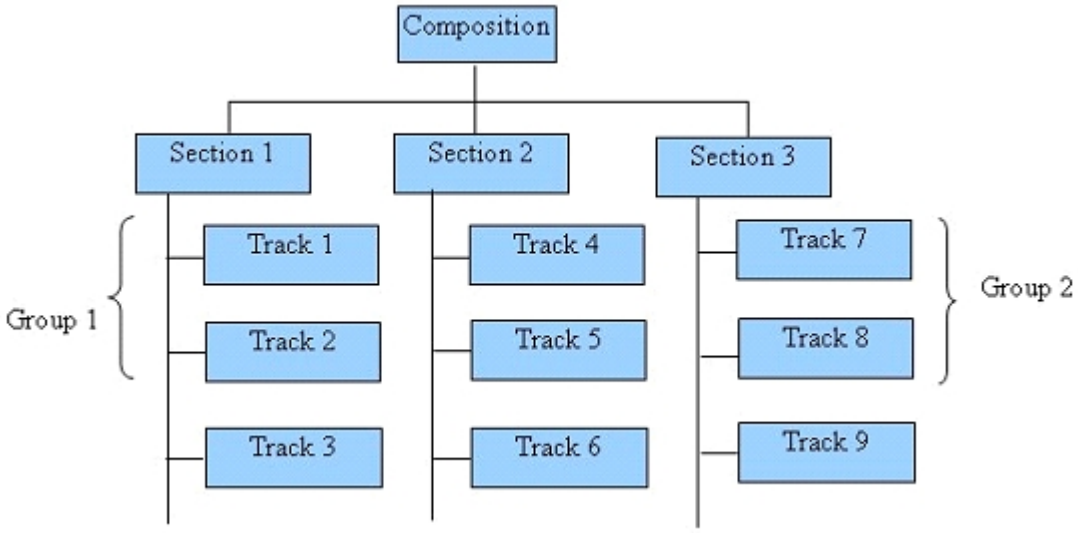
1.2 Features

1.2.1 - Users define their own compositions

MARS allows the user to mix different sound files into a single composition. Files must be of the wave (*.wav) sound format. One can specify the way these files interact with one another. They can be looped, played sequentially, faded, super-imposed, played at half the volume or a number of other features. The user has full control over what to do with these tracks, thereby leaving all the creativity up to him/her.

1.2.2 - Hierarchical Structure of Compositions

Like many applications in computer science, music compositions are nothing but a collections of different entities that are executed (played) in a certain order, or concurrently. A MARS musical composition consists of Sections, Tracks and Groups. Musical compositions can be represented like the tree below:



1.2.3 – Define the behavior of your entities

The composition is always the root of the tree that contains the sections, groups and tracks. Tracks are the building block of a composition. They are the individual wave files that the user wants to use. Groups are logical groupings of these tracks. For example, the user may want to have a "rhythm" group that will consist of tracks called "bass_drum", "snare_drum", and "cymbal". The user can define groups if they find themselves reusing the same groups of tracks at various points throughout the composition.

When you listen to pop music or even Mozart, you will hear repeated sections. In a pop song, this may be referred to as the "chorus" or the "bridge". Our sections are meant to serve this purpose. Users will define Sections that logically represent a self-contained part of the composition. These sections can be either repeated later on, or just used once.

Within sections or groups, users can choose to play, loop, or add some advanced functionality to their tracks. These behaviors are what will give the composition substance and make the combination of wave files come together to form a coherent composition.

1.2.4 - Advanced Functionality

For each entity, there exists advanced functionality for that entity. Users have the ability to only play a certain duration of a track, delay the playing of a track, or mix tracks together. They also have the ability to explicitly set the volume on certain tracks to be very high and set the volume on other tracks to be very low, similar to dynamics alteration in an actual musical composition. Users can also fade tracks in or out over a given duration of time.

1.3 Short sample composition

```
def composition MyOpus
            def section RhythmSection
                        track kick = “kick.wav”
                        track bass = “bass.wav”

                        for(i: 1 to 8 step 1) do
                                    bass.play()
                                    if(i % 2 == 0) then
                                                kick.play()
                                    end
                        end
            end
            playOrder(RhythmSection)
end
```

1.4 Conclusion

We believe that the MARS language makes it very simple for virtually anyone with programming experience to create their own composition in a very short amount of time. There is no need to buy any expensive software, all that is needed is a free Java compiler. We do all the gruesome details of working with an external sound API so the programmer can just worry about what making music is all about: creativity.

2. Tutorial

2.1 Your first composition - "HelloOpus"

Before we get into the complex things you can do with the MARS language, lets first start with how to define a composition and several entities. Whenever we define nested MARS specific entities (compositions, sections, groups), we use the "def" keyword, to let the compiler know that we are defining a new nested entity. In addition, whenever we are finished defining a nested entity, we wrap it up with an "end" keyword:

```
def composition HelloOpus
        …
end
```

Great, we now have a composition that we can work with called "HelloOpus". Each composition should always be in its own MARS (.mars) text file. Now lets say that we have two wave files "HelloWorld.wav" which is a wave file of Swapneel saying "Hello World!" and "Nifty.wav", which is a wave file of Ritkia saying "This is Nifty!". Each wave file will be a separate track in the composition:

```
def composition HelloOpus
        track swapneel = "HelloWorld.wav"
        track ritika = "Nifty.wav"
end
```

Now we have a composition with two tracks defined, but nothing will happen with them. We now have to define a section of the composition. All compositions must have at least one section. You can define tracks outside of a section, but you are not allowed to do anything with those tracks. In this particular section, we will play the track named swapneel, followed by the track named ritika, followed by the both tracks played on top of each other. The playOrder system command tells the compiler to play the sections of a composition in that particular order. Even though there is only one section in this composition, playOrder is mandatory.

```
def composition HelloOpus
        track swapneel = "HelloWorld.wav"
        track ritika = "Nifty.wav"

        def section MainSection

                swapneel.play()
                ritika.play()
                mix(swapneel.play(), ritika.play())
        end

        playOrder(MainSection)
end
```

Now that we have this composition, save it as "HelloOpus.mars" and exit your text editor. The next thing we can do it compile our composition. There are three steps to compile a MARS composition. Also, where ever it says "prompt>" below just means a

general command prompt, you should not be typing that. The first step is to compile your file using the mars compiler"

```
prompt> java Mars HelloOpus.mars
```

You now have a java file called "HelloOpus.java". To compile this file, type:

```
prompt> javac HelloOpus.java
```

You will now have a final executable MARS composition. To run it, type:

```
prompt> java HelloOpus
```

Congratulations, you have finished your first MARS composition. You should be hearing Swapneel saying "Hello World", followed by Ritika saying "This is nifty!", followed by both of them saying their sayings at the same time.

2.2 Advanced features

2.2.1 Looping

The Hello Opus was a great first composition, but there is really no depth to it. MARS allows the user to loop tracks, delay the playing of tracks, and do many other advanced features. We will talk about some of those advanced features in this section. For the following code snippets, assume that we are using the two tracks "swapneel" and "ritika" from the previous section.

There are two different ways to loop tracks in MARS. Since looping in our language is used very often, the loop command provides an easy way to loop several tracks. To do so, call loop on a track and let the track know how many times to loop:

```
swapneel.loop(5)
```

This will loop the swapneel track 5 times in a row. The same effect can be used with for loops in our language. For loops are meant for when the user wants to change something specific every iteration of a loop. For loops take the syntax for(variable:x to y step z). Variable will be the loop counter, x will be what number to start from, y will be the number to end on, and step will be the number that variable will be increased or decreased by each iteration. The following code does the same thing as the code above:

```
for(i:1 to 5 step 1) do
            swapneel.play()
end
```

2.2.2 Delays

Delays are used to delay playing a track for a certain amount of time. A user might want to delay playing a track because musically, a few more tracks might be playing at that time. To do this, just call the delay command, and pass the amount of time to delay into it. The following code will play the track named ritika, pause for 5 seconds, then play the track named swapneel.

```
ritika.play()
delay(5000)
swapneel.play()
```

2.2.3 Mixing

The real power in the MARS language comes from the users ability to mix tracks together. This allows the user to play more than one track on top of another at the same time, bringing depth to the composition. Most MARS compositions are going to be many tracks mixed on top of one another. To mix multiple tracks, you can use the mix system command. The following code plays the ritika track on top of the swapneel track:

```
mix(ritika.play(), swapneel.play())
```

2.2.4 Primitives

MARS allows users to do normal arithmetic and use conditionals, similar to any other high level programming language. The two primitive numeric types are int and double. Users may want use arithmetic for primitives because if they know the specific tempo of a set of tracks, they may find it easier to work in measures or beats instead of milliseconds.

2.2.5 More advanced features

There are more advanced features in MARS such as setting the volume, fading a track and creating groups of tracks. For a complete listing of advanced features, please reference the System Commands Section (3.3.4) of the Language Reference Manual.

2.3 Unique syntactical conventions

2.3.1 Scoping

In MARS, scope is not defined with braces like most high level languages. Instead, scope is created when you create a nested MARS entity, or create a nested control flow statement such as if or for. Scope is created within the def/end keywords. MARS uses static scoping similar to C++, and all any entities or variables created inside a scope are not accessible outside the scope.

2.3.2 Line Delimiters

MARS code is not terminated using a semi-colon. Instead, lines are terminated using the new line delimiter. Every line of code is its own statement, and you are not allowed to write multiple statements on the same line of code.

3. Language Reference Manual

3.1 Overview

MARS is an Object-Oriented Programming Language with Scripting features. It will allow a user the ability to mix and match different tracks (audio files) on top of one another and add effects to these tracks. It will give users a simple way to define their composition logically and give them the ability to listen to and iterate on certain sections of their composition.

Using MARS, a user can define a song, as a logical structure as follows.

1. Composition
2. Sections
3. Groups
4. Tracks

A track will be an individual audio file (wave). They can be defined anywhere, but functions on them can only be called inside sections or groups.

A group will be a logical collection of tracks and their behavior (play, loop). The default behavior of a group is to play each track one after the other.

A section is a physical grouping and is analogous to sections of a song, like intro, chorus, etc. A Section may or may not contain Groups, but it must contain Tracks.

A composition is a collection of Sections, which make up a Song. At the end of the composition definition, the user will define a play order, where they will tell the compiler what order the play the sections in.

3.2 Lexical Conventions

3.2.1 Keywords

MARS defines the following set of keywords. These keywords are reserved and cannot be used as identifiers.

composition	if
def	int
double	section
else	then
end	to
for	track
group	step

Table 1: Keywords

3.2.2 Arithmetic and Comparison Operators

MARS supports all Arithmetic and Comparison Operators similar to C and Java. All operators are left-associative and have the precedence levels shown below.

** is the Exponentiation operator. Hence, to calculate, 2^3 , we would write 2**3.

( )
**
!
*, /, %
+, -
>, >=, <, <=, ==, !=
&&, ||

Table 2: Operators

3.2.3 Numbers

MARS supports 2 types of numbers:

1. Integer (1 or more digits)
2. Double (1 or more digits followed by a decimal place followed by one or more digits)

3.2.4 Block Declarations

MARS supports standard blocks.

3.2.4.1 Standard Blocks

There are 2 kinds of Standard blocks. They are Groups and Sections. They can be defined as follows.

```
def composition C1
            def section S1
                        ...
                        def group G1
                                    ...
                                    ...
                        end
                        ...
            end
end
```

3.2.5 Scoping Rules

MARS is a statically typed language. Variable defined in a particular block has the scope of that block. Blocks are defined by using the `def – end` construct, as shown below

```
def section A
            track foo = “foo.wav”
            t1.loop(5)
end
foo.loop(5) //Illegal Access for 'foo'
```

3.2.6 Strings

MARS supports Strings only when defining a track. The Strings are of the form

```
“ path to file name . file type “
```

3.2.7 Comments

MARS supports single -line and multi-line comments like C, C++, Java, etc. Single Line Comments are indicated by a double forward slash (//) at the start of a comment. Multi Line Comments are delimited by '/*' and '*/'.

```
//This is a valid MARS comment.
/* So is
this */
```

3.2.8 New Line Delimiters

Each line of code in MARS is delimited by a newline character (\n). Semi colons are not allowed to terminate a statement.

3.2.9 Identifiers

Identifiers in MARS start with a letter and can be followed by any number of letters or digits.

3.3 Data Types/Attributes

3.3.1 Fundamentals

The fundamental type of the MARS language is a composition. The composition is a set of tracks, (optionally) groups, and sections.

3.3.1.1 composition

The composition is the foundation of the mars language. It acts as a container for the definition of all tracks, groups, sections, delays, and custom effects.

3.3.1.2 group

Groups are user defined grouping of tracks. The main purpose for the definition of groups are so users can group certain tracks in different categories such as "rhythm" and "melody" for ease of use later in the composition.

Groups will remain local to the scope in which they are defined (similar to C and Java). If they are not defined in a section and just defined in the composition, they are considered global to the whole composition.

Example:

```
def group rhythm
            track bass = "crazyBass.wav"
            track snare = "snareDrum.wav"
end
```

3.3.1.3 section

Sections are user defined subdivisions of a composition. Their main purpose is to allow the user to define a "mini composition" within their entire composition. Because most compositions follow a form where certain musical sections are repeated (i.e. A B A), these keywords will allow the user to emulate a real composition.

The behavior for a section is to play its tracks and groups with the defined delays and effects. Sections will not have the ability to be superimposed over each other as they are meant to be sequential entities.

All tracks and groups must be defined within a section and will remain local for that particular section.

Example:

```
def section A
            track melody = "happyMelody.wav"

            def group rhythm
                        track bass = "crazyBass.wav"
                        track snare = "snareDrum.wav"
            end

            melody.play();

end
```

3.3.2 Identifiers

All the fundamentals are made up of identifiers in the composition. These identifiers are defined by the user and must be a letter followed by letters or digits and unique to the scope in which they are defined.

Example:

```
track mike...
group rhythm...
section chorus...
```

3.3.3 Primitives

Primitives are used in MARS as an easy way for user to do simple arithmetic when they define their looping and delays.

3.3.3.1 int

Integers are mainly used when users define how many times to loop a given track. They are also used to help the user define delays for a given track or group. Users can do arithmetic on integers to make defining loops easier.

3.3.3.2 double

Doubles are mainly used for operations like manipulation of track lengths. Users can do arithmetic on doubles. Both integers and doubles can not be assigned a previously defined integer or double. This is an unnecessary task that should not need to be done for a MARS composition.

3.3.3.3 track

The track is the main building structure of a composition. Tracks are files that users specify to manipulate, add effects to, and superimpose over each other.

Tracks will remain local to the scope in which they are defined (similar to C and Java). If they are not defined in a section and just defined in the composition, they are considered global to the whole composition.

Example:

```
track bass = “crazyBass.wav”
track snare = “snareDrum.wav”
```

3.3.4 System Commands

Certain entities have the following operators that the user can manipulate to give their composition structure, superimpose tracks at specific times, and add custom effects.

3.3.4.1 play(), play(double)

Users can call the play command on a track or group to signal that they want it to begin to play at this point. Play commands can only be applied to a group or a track and they can only occur within a section. For tracks only, users can also only play it for a given amount of time by passing in a double. If the duration they specify is greater than the duration of the track, the difference is ignored.

3.3.4.2 loop(int)

This command will loop a specified track or group a number of times. This command will take an integer that will define how many times to loop. Loop is with respect to the calling track/group. Loops can only be applied to a group or a track and they can only happen within a section.

3.3.4.3 fadeIn(double)

This command allows a track to fade the volume in. It will take an double representing the number of milliseconds to fade in. The volume will start at 0 and it will go to full volume. Fade is with respect to the calling track, can only be applied to a track and it can only happen within a section.

3.3.4.4 fadeOut()

This command allows a track to fade the volume out. The volume will start at its current volume and will fade to 0 over the remaining duration of the track. FadeOut is with respect to the calling track, can only be applied to a track and it can only happen within a section.

3.3.4.5 setVolume(double)

This command allows the users to set the volume of a particular track. It takes a double between 0 and 10 representing the new volume. Users are not permitted to set the volume while the track is playing, they must call setVolume between loop iterations or before they explicitily play a track or group. SetVolume can be applied to a track or a group and can only be applied within a section.

3.3.4.6 getLength()

The command gets the length of a track. It returns an int that represents the length of a given track in milliseconds.

3.3.4.7 delay(double)

This command delays playing a certain track. It will take an integer representing the amount of time to delay in milliseconds.

3.3.4.8 mix(track.play(), track.play()...)

This command allows users to mix the playing of two or more tracks together. Its arguments will be a play command on two or more tracks. These tracks will be played on top of each other. If one of the tracks ends before another one does, it will play for the length of the longest track before moving on to the next command.

3.3.4.9 playOrder(section...)

PlayOrder is the main command for the entire composition that allows the user to specify the order in which the defined sections should be played. This command will take one or more arguments (sections) that will represent the sequential order of the compositions. PlayOrder is a required command as it is the main entry point into the composition.

Examples:

```
//play section A, B, then A
playOrder(A,B,A)
//play section A, B, A, A
playOrder(A,B,A,A)
```

3.4 Control Flow

3.4.1 Conditional

MARS supports conditional statements similar to other languages like 'C' and 'Java'. These conditionals allow the user to define certain scenarios of what to do with a particular track or group at a certain time or iteration. The following example will test to see if the length of a track is 2, and if so it will do something different.

Example:

```
if(t1.getLength() == 2) then
            t1.fadeIn(10)
else
            t1.setVolume(8.5);
end
```

The 'if' statements will take a boolean expression and evaluate it. If the boolean expression evaluates to true, the code inside the if will be executed. If it evaluates to false, the code will not be executed. The operators that are supported by a boolean expression can be found in section 3: Lexical Conventions. Unlike other languages like Java and C, conditional statements do not allow curly braces and everything is enclosed in if and end statements.

Users can also define an optional else to execute if the boolean expression after the if statement evaluates to false. If no else statement appears, the user can just close the if statement with an "end" keyword. One addition thing to note is that the user can not evaluate arithmetic expressions inside of boolean expressions for an if statement. The following example below is invalid:

```
if(2+3 < 5-4) then                  //invalid!
            ...
end
```

3.4.2 Iteration

MARS supports iteration through built in functionality for many of its data types. The user can specify that they want to loop a particular track, group, section, any number of times. This shorthand is designed because looping tracks and groups are such common tasks in the MARS language. The example below will define a track and loop it 6 times.

Example:

```
track t1 = “mike.wav”
int loopTrack1 = 6
t1.loop(loopTrack1)
```

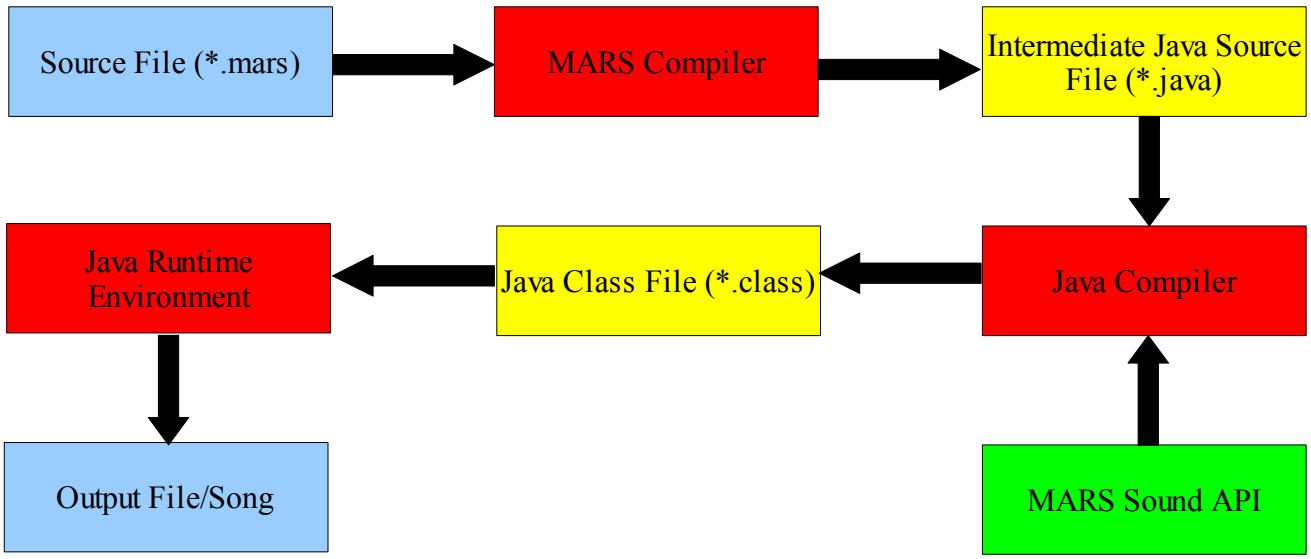
MARS also supports traditional looping similar to C and Java. The user can define a for loop and carry on a certain number of tasks within that loop. MARS supports iteration in this fashion as well because users will want to accomplish certain tasks on each iteration that they can not do using the built in “loop” function. The for statement takes a starting value, a terminating value, and a step value. Please note that the step value must be a positive integer indicating how much to increase/decrease the loop counter each iteration. In the example below the tracks inside the loop have been predefined.

Example:

```
for(i: 1 to 10 step 1) do
            track1.play();
            track1.setVolume( i );  //increase volume

            //if its the last iteration, fade out!
            if(i == 10)
                        track1.fadeOut()
            end
end
```

3.5 Compilation & Execution



Drawing 1: Compilation - Interpretation - Execution Flowchart

The steps to compile and execute a MARS program are as follows:

- The user writes a program in the MARS programming language.
- The MARS Compiler converts the .mars file into a .java file. This can be done as shown below:
  - java Mars sample_program.mars
- The above step will generate sample_program.java. The java source file can be compiled and executed like any other java program.
  - javac sample_program.java
  - java sample_program

3.6 Code Samples

3.6.1 Full Composition

```
def composition sampleOpus

            //define some global tracks!
            track kick = “kick.wav”
            track bass = “bass.wav”
            track melody = “melody.wav”

            //define our first section
            def section A
                        for(i: 1 to 8 step 1) do
                                    kick.play();

                                    //slowly add more tracks
                                    if(i >=2) then
                                            mix(kick.play(), bass.play())
                                    end

                                    if(i >= 4) then
                                    mix(kick.play(), bass.play(),melody.play())
                                    end

                                    //if its the last iteration, fade out
                                    if(i == 8) then
                                                kick.fadeOut()
                                                bass.fadeOut()
                                                melody.fadeOut()
                                    end
                        end
            end

            def section B
                        //define some new tracks
                        track bass2 = “bass2.wav”
                        track melody = “melodyNew.wav”
                        track counterMelody = “counterMelody.wav”

                        bass2.play()
                        melody.play(bass2.getLength() * 4)
                        counterMelody.play(bass2.getLength() * 8)

                        bass2.loop(12)
                        melody.loop(8)
                        counterMelody.loop(4)

            end

            def section C
                        track cymbal = cymbal.wav

                        for(i: 1 to 8 step 1) do
                                    kick.play();

                                    //play cymbals every other iteration
                                    if(i % 2 == 0) then
                                                mix(kick.play(),cymbal.play())
                                    end


                                    if(i >=2) then
                                                mix(bass.play(),kick.play())
                                    end

                                    if(i >= 4) then
                                                mix(bass.play(),melody.play())
                                    end
                        end
            end

            //play this composition using sections A,B,C,B,A
            playOrder(A,B,C,B,A)
end
```

4. Project Plan

4.1 Planning, specification, development and testing

Once team MARS was formed and we had the idea for the MARS language, we all brainstormed to talk about various things we wanted our language to accomplish. MARS was first supposed to be a DJ language, but once we all talked about capabilities of what can be done with wave sound files, we decided it was going to be more of a music mixing language.

Our planning model was to all research anything related to playing sound in Java, and to try to get a sample application of several wave files playing on top of one another. We all had a pretty good idea about what we wanted this language to do and we all kept an open mind about new features to add.

Following the white paper, we decided to investigate all technical details about the sound framework we would work with. We first investigated the Java Sound API, but could not find anything in the documentation about super-imposing songs on top of one another, which was critical for our language. So we decided to work with the Java Media Framework, a multi-media framework for Java.

For the Language Reference Manual, our plan was to have all details about how the back-end would be implemented figured out, and to detail the specific features of our language. Following the Language Reference Manual, our team split up who would be working on what specific tasks and then used BaseCamp, an on line project management website.

Half way through the semester, we came to the realization that the Java Media Framework would not work because it took too long to load a small sound file, and we would need to load and play them instantaneously. Aaron and Ritika researched the Java Sound API more and decided that it does have the capability to do what our language needs.

Once we came up with the specifications of our language and posted the tasks of development phase, we started having smaller meetings every week instead of larger meetings. Swapneel and Mike would meet to discuss and work on the Lexer and Parser and Aaron and Ritika would meet to discuss and work on building a functional back-end included the MARS sound API that would interface with the Java Sound API. Once the Lexer and Parser were done, Swapneel worked with Aaron and Ritika to talk about how the walker would interface with the back-end. Towards the end of semester, Mike wrote several test cases for different aspects of the language, and the group ran though these test cases.

4.2 Programming style

The MARS team followed several programming guidelines for both the ANTLR code and Java Code. For all the ANTLR code, we broke up the Mars.g file into three different portions: the lexer, parser and tree walker. Each section broken up by comments and within each section, all related items were grouped together for easy reading (i.e. operators for the Lexer or arithmetic expressions for the Parser).

For the Java code, our team used several guidelines. These guidelines helped each member of our team work independently and easily understand one another's code. The guidelines were as follows:

- The top of every Java file should have brief comment telling about what this class does.

- All member variables in a class should be referenced with a specific "this" pointer in front of it. Although it is unnecessary for compilation, this helped our team know which variables were member variables when referencing them later on.

- All Vectors or collections must explicitly declare a "type" of objects that will be stored in that collection.

- Javadocs should be written for all methods within a class.

- Longer, more descriptive variables names are preferred to shorter names.

- All open braces should be placed on the same line as a statement or method declaration. Example:

```
if (2 == 3) {
...
}
```

- Comments should be used for all other code at the programmers discretion.

- To get another programmers attention, use "@" with their name, so they know it is an important comment to them. Example:
  // @swapneel: please finish this programming language

- Java code should follow the Java coding standard set by Sun Microsystems. The URL can be found here: http://java.sun.com/docs/books/jls/index.html

- All primitive MARS types should implement the Type interface.

4.3 Project Timeline

Task	Date Completed
White Paper	9-26-2006
Language Details Defined	10-12-2006
Language Reference Manual	10-26-2006
Grammar defined	11-10-2006
Functional back-end implemented	11-10-2006
Walker implemented	11-27-2006
Integration Complete	12-01-2006
Final Report Complete	12-18-2006
Testing Complete	12-19-2006

4.4 Roles and Responsibilities

Person	Role
Swapneel Sheth	Lexer, Parser, Walker
Ritika Virmani	MARS sound API, Code generation
Mike Sorvillo	Lexer, Testing, Documentation
Aaron Fernandes	Parser, Walker, Code generation

4.5 Software development environment and tools

All compiler code was written using Java 2 SE 5. All the lexer, parser and walker code was generated using ANTLR. Our integrated development environment of choice was Eclipse, mainly because of its integration with third-part ANTLR software and version control, ease of use, and convenient features like auto-complete.

Our version control system of choice was Subversion. We chose subversion over CVS because of its nicer features such as all commits being truly atomic, directory versioning, and the ability to rename files and directories. Ritika and Aaron used Subclipse, the eclipse subversion plug, Swapneel used the old-fashioned command line ssh with custom shell scripts, while Mike used TortoiseSVN, a windows based subversion client that is integrated into the Windows shell.

Basecamp was used as our project management website. Basecamp allows a project to be set up with multiple team members and allows deadlines, to-do lists, and tasks to be assigned to specific users. In addition, Basecamp has a white board feature,

where users can post notes about a project. One of our white boards was to keep track of bugs, and the other white board was called "Doodles" for miscellaneous notes to each other.

A few more features of Basecamp that we took advantage of was the calendar feature and the message board. Instead of flooding each others email boxes, we would use the Basecamp message board where a team member can post a message and other team members can view and write back to these messages. Basecamp was a critical tool in keeping this project organized.

4.6 Project Log

Below is a more detailed account of what occurred throughout the semester. We typically had meeting on Thursday before class, but that varied depending on everybody's workload for that given week.

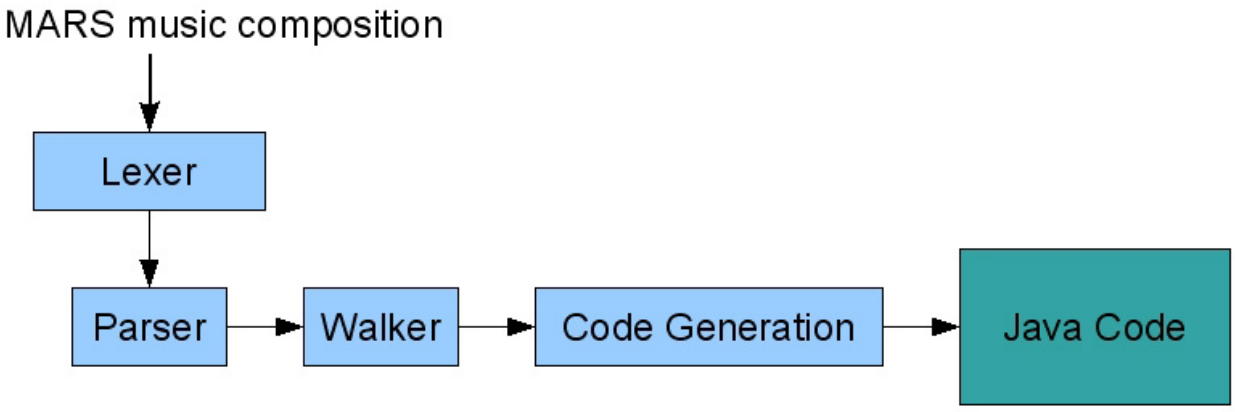
Task	Date
Brainstorming for ideas	9-14-2006
Exploring music language idea	9-21-2006
Finished White paper	9-26-2006
Discuss Language Details	10-5-2006
Finalize Language Details	10-12-2006
Finished Language Reference Manual	10-19-2006
** NO MEETING – MIDTERMS! **	10-26-2006
Work on Lexer and Parser	11-02-2006
Work on Functional back-end	11-02-2006
Start on Walker	11-15-2006
** NO MEETING – THANKSGIVING! **	11-23-2006
Begin integration	11-29-2006
Begin Testing	12-09-2006
Begin Final Report	12-09-2006
Complete Final Report	12-18-2006
Complete Testing	12-19-2006

5. Architectural Design

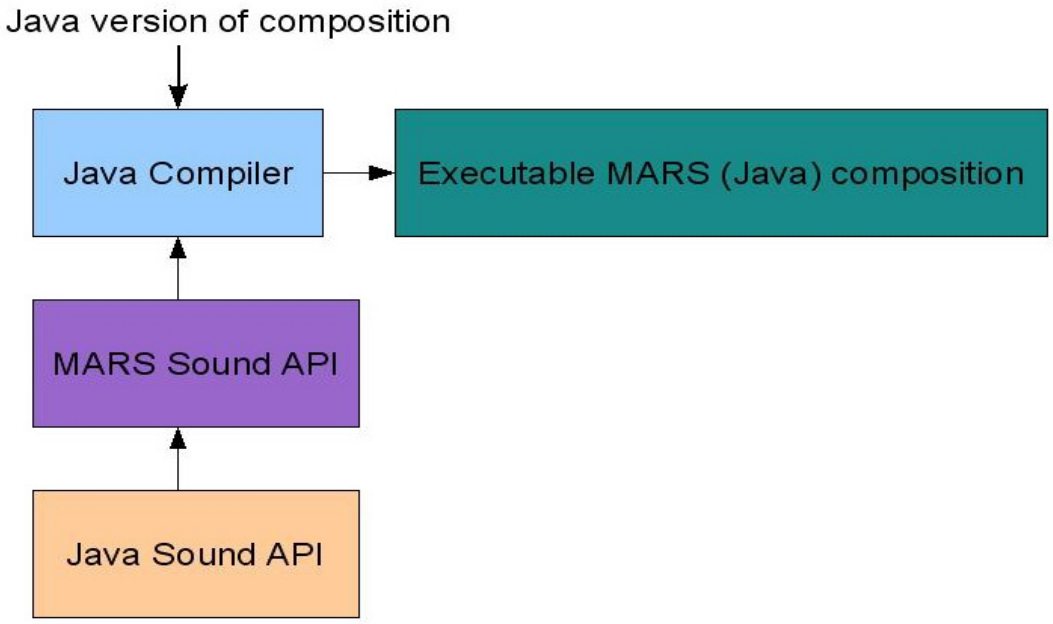
5.1 Block Diagrams

The MARS language starts with a MARS music composition (*.mars) text file. The first phase is lexical analysis, then parsing. The parser will generate an abstract syntax tree that the MarsWalker (MarsWalker.java) can walk and check for various things that the parser can not. Some examples of that are:

- checking if operands are defined
- checking that variables are defined
- making sure operands are consistent with the operator
- making sure booleans are the only thing being tested in "if" statements
- other syntax errors the parser can not avoid
- making sure function call arguments match arguments of function definitions
- making sure a variable has not been redefined
- tracks don't contain groups, groups don't contain sections



At this point, we will have a MARS java file. This will be the Java representation of the MARS music composition. This file will then get recompiled using the Java compiler. In its compilation process, it will reference the MARS sound API, which references the Java Sound API. The final product will be a Java binary file that can be executed like any other Java program.



5.2 Interfaces between components

The Lexer goes through the MARS composition (*.mars) file, and divides it up into tokens that that parser can use. If there are any invalid tokens, the lexer throws an error. The final output of the Lexer is a set of tokens for the parser. The parser takes the tokens from the lexer and makes sure that they appear in the correct order. If the programmer does not follow the correct syntax, an error is throw. The final output of the Parser is an abstract syntax tree for the Walker.

The Walker traverses the AST and at each node it evaluates the action at that particular node. Some main actions it will do is create Java representations of the types defined by the user (i.e. Int, Track, Composition, etc...). It will also perform static semantic analysis to resolve symbols and check for things that the parser can not. Examples of those actions can be found in Section 5.1. The output of the Walker is a MARS composition data structure.

The Code Generator takes a MARS Composition data structure and generates Java code based on the elements of the composition. The output will be a Java representation of the original MARS composition. At this point, the user can compile that Java file which will interface with the MARS Sound API.

The MARS Sound API is an abstracted version of the Java Sound API. It contains only the functions necessary for this project, and not complete Java Sound functionality. After compilation, the user is left with a final Java executable file. To hear their composition, they can use the Java runtime environment to run their composition.

5.3 Implementation responsibilities

In the beginning of the project, Mike and Swapneel were to implement the lexer and the parser and Aaron and Ritika were to implement the code generation and music functionality. Over the course of the project, our responsibilities varied. Please reference Section 4.4 for implementation roles and responsibilities.

6. Test Plan

Each person who coded a feature was responsible for testing that feature. After a feature was deemed workable on its own, we would run the formal tests for the new feature and all the old features. This regression testing method was used to make sure that no new features broke older, working features of the language. The following section explains a little more about the formal testing that we did as a group.

6.1 Translator testing

Each member that worked on the translator was responsible for testing the part that they coded after coding it. Once they established that work, it was checked off the list and it was to be tested more throughly as we integrated more and more features. The way we decided to test the lexer, parser and walker of our language was to implement very small sample programs that used specific features of our language. Examples of this are number and arithmetic, for loops, if statements, MARS entity definition and usage, and system command usage. These tests were then used again every time a new feature was added to the MARS language for regression purposes.

6.1.1 BasicTest.mars

Below is the test we did for testing the lexer, numbers and arithmetic. It tests thing such as making sure the lexer is only letting in the proper tokens, adding, subtracting, defining variables, etc...

```
/*
  BasicTest.mars
  will test out lexer, numbers, arithmetic, math functionality
 */

 def composition BasicTest

            //must wrap this in section for parser
            def section A

                        //make sure odd multiline comments work
                        /****/

                        //create some variables, only assign some
                        int a = 6
                        int b = 9
                        double e = 15.5

                        //assign the other two later on
                        b = 9
                        e = 15.5

                        //reassign a variable
                        a = 14

                        //test associativity & precedence
                        int f
                        f = 2 + 4 * 3 + 2

                        //test parentheis
                        int s
                        s = (2 + 4) * (3 + 2)

                        //subtract two doubles
```

```
                    double q
                    q = 20.5 - 10.5


                    //test mod
                    int wpd
                    wpd = 4 % 2

                    //test power
                    int pow
                    pow = 2**3

                    //create new variables.  assign first to second
                    int foo = 90
                    int bar = 99

                    //test a valid filename and get its length
                    track t1 = "mike.wav"
                    int t1Length
                    t1Length = t1.getLength()


                    /*
                    //test an invalid filename (*LEXER ERROR*)
                    track t2 = "john.mp3"

                    //make sure we cant assign booleans (*WALKER ERROR*)
                    int ggg = true
                    int ddd = 2 < 3

                    //test one that is not declared (*WALKER ERROR*)
                    c = 5

                    //type random garbage to make sure lexer gets it (*LEXER ERROR*)
                    garbage!
                    int xx = "hello!"

                    //cant declare and assign to an expression
                    int fd = 2 + 4
                    */

        end

        //needed for parser!
        playOrder(A)
end
```

6.1.2 BasicTest.java

Below is the Java output that our MARS compiler generated.

```
package Tests;
import mars_util.*;
public class BasicTest{
public static void main(String[] args) {
A();
}
static void A() {
Track t1= new Track("file:/mike.wav");

int a = 6;
int b = 9;
double e = 15.5;
int f = 0;
int s = 0;
double q = 0.0;
int wpd = 0;
int pow = 0;
int foo = 90;
int bar = 99;
int t1Length = 0;
b = (int)9;
e = 15.5;
a = (int)14;
f = (int)((2 + (4 * 3)) + 2);
s = (int)((2 + 4) * (3 + 2));
q = (20.5 - 10.5);
wpd = (int)(4 % 2);
```

```
pow = (int)(Math.pow(2, 3));
t1Length = (int)t1.getLength();
}
}
```

6.1.3 LoopTest.mars

Below is the test we did for testing loops in mars. Both for loops and system call loops are being tested to make sure there is no issue with them.

```
/*
  LoopTest.mars
  will test out looping: for loops and system call loops
 */

 def composition LoopTest

            //we have to create a section for this to work
            def section A

                        //create two tracks to test with
                        track t1 = "C:/sounds/verse_guitar.wav"
                        track t2 = "C:/sounds/verse_orchestra.wav"


                        //loop the first one five times
                        t1.loop(5)

                        //define a variable, lets step it up each iteration
                        int a = 0

                        //test incremental loops
                        for(i: 1 to 3 step 1) do

                                    //play the second track
                                    t2.play()
                        end


            end

            //needed for parser
            playOrder(A)
 end
```

6.1.4 LoopTest.java

Below is the Java output that our MARS compiler generated.

```
package Tests;
import mars_util.*;
public class LoopTest{
public static void main(String[] args) {
A();
}
static void A() {
Track t1= new Track("file:/C:/sounds/verse_guitar.wav");

Track t2= new Track("file:/C:/sounds/verse_orchestra.wav");

int a = 0;
t1.loop(5);
for (int i=1;i<=3;i+=1) {
t2.play();
}
}
}
```

6.1.5 ConditionalTest.mars

Below is the test we did for conditionals. If statements, if/else statements, and combinations are tested.

```
/*
  ConditionalTest.mars
  will test out conditionals: make sure if statements work in all instances
 */

 def composition ConditionalTest

            //we have to create a section for this to work
            def section A

                        //create two tracks to test with
                        track t1 = "C:/sounds/verse_guitar.wav"
                        track t2 = "C:/sounds/verse_orchestra.wav"

                        //create two test integers
                        int a = 2
                        int b = 3

                        //we should only be hearing the orchestra
                        if(a == b) then
                                    t1.play()
                        else
                                    t2.play()
                        end

                        //lets create a complex boolean...should always play
                        if(2 < 3 &&  3 > 1 || a == a && !(a == b)) then
                                    t2.play()
                        end

                        //create another composite boolean with precedence
                        if(((a != b || 2 == 7) && 2 >= 2) && 3 <= 3) then
                                    t1.play()
                        else
                                    t2.play()
                        end

                        //this should fail.  can not evaluate arith inside booleans (*ERROR*)
                        /*if(2 + 3 == 5) then
                                    t1.play()
                        end*/
            end

            //needed for parser
            playOrder(A)
 end
```

6.1.6 ConditionalTest.java

Below is the Java output that our MARS compiler generated.

```
package Tests;
import mars_util.*;
public class ConditionalTest{
public static void main(String[] args) {
A();
}
static void A() {
Track t1= new Track("file:/C:/sounds/verse_guitar.wav");

Track t2= new Track("file:/C:/sounds/verse_orchestra.wav");

int a = 2;
int b = 3;
if (a == b) {
t1.play();
}
else {
```

```
t2.play();
}
if (((2 < 3) && (3 > 1)) || ((a == a) && ( ! ( (a == b)) ))) {
t2.play();
}
else {
}
if ((((a != b) || (2 == 7)) && (2 >= 2)) && (3 <= 3)) {
t1.play();
}
else {
t2.play();
}
}
}
```

6.1.5 EntityTest.mars

Below is the test we did for MARS entities. It will test various things like creating and manipulating groups and tracks in different scopes, and playing sections more than once

```
/*
  EntityTest.mars
  will test out entities: make sure all mars entities can not be nested improperly
                                    test scoping for entities and primitives
 */

 def composition EntityTest

            //create some global tracks to play with
            track t1 = "C:/sounds/verse_guitar.wav"
            track t2 = "C:/sounds/verse_orchestra.wav"

            //create some ints to test with
            int a = 3
            int c = 3

            //define an invalid group with a nested section (*ERROR*)
            /*def group gInvalid
                        def section gInvalid
                                    a = 5
                        end
            end*/

            //we have to create a section for this to work
            def section A

                        //define a group called g1
                        def group g1
                                    t1.loop(2)
                                    t2.play()
                        end

                        //play that group
                        g1.play()

                        //check variables in parent scope
                        if(a == 3) then
                                    t1.play()
                        end
            end

            //define another section to test with
            def section B

                        //define some local tracks
                        track t3 = "C:/sounds/bridge_bass.wav"
                        track t4 = "C:/sounds/bridge_guitar.wav"

                        //define another group that works with all 4 tracks
                        def group g2
                                    t2.play()
                                    t3.play()
                                    t1.play()
                                    t4.play()
                        end
```

```
                        //play that group
                        g2.play()

                        //should not work...not in scope (*ERROR*)
                        //g1.play()

            end


            //needed for parser
            playOrder(A,B,A)
 end
```

6.1.6 EntityTest.java

Below is the Java output that our MARS compiler generated.

```
package Tests;
import mars_util.*;
public class EntityTest{
static Track t1= new Track("file:/C:/sounds/verse_guitar.wav");
static Track t2= new Track("file:/C:/sounds/verse_orchestra.wav");
static int a = 3;
static int c = 3;
public static void main(String[] args) {
A();
B();
A();
C();
A();
}
static void A() {
t1.loop(2);
t2.play();
if (a == 3) {
t1.play();
}
else {
}
}
static void B() {
Track t3= new Track("file:/C:/sounds/bridge_bass.wav");

Track t4= new Track("file:/C:/sounds/bridge_guitar.wav");

t2.play();
t3.play();
t1.play();
t4.play();
}
static void C() {
}
}
```

6.1.5 CommandTest.mars

Below is the test we did for MARS system commands. We just defined several tracks and groups to work with, and tested each of the system commands in various orders

```
/*
  CommandTest.mars
  will test out system commands: make sure you can only call commands on correct things.  make sure all
commands are working correctly
 */

 def composition CommandTest

            //define a section to work in
```

```
        def section A

                //define some tracks to work with
                track t1 = "C:/sounds/rap_keys.wav"
                track t2 = "C:/sounds/rap_guitar.wav"
                track t3 = "C:/sounds/bridge_bass.wav"

                //play the first then the second one
                t1.play()
                t2.play()

                //mix em!!!
                mix(t1.play(), t2.play())

                //pause for 5 seconds
                delay(5000)

                //lower the volume of track 1 and loop it 3 times
                t1.setVolume(2)
                t1.loop(2)

                //only play 1000 ms of track 2
                t2.play(1000)

                //reset the volume, play the first track
                t1.setVolume(10)
                t1.play()
                t1.fadeOut()
                t1.setVolume(10)

                //define a group to work with (just plays t1 and t2)
                def group g1
                        t1.play()
                        t2.play()
                end


                //loop that group twice
                g1.loop(2)

                //play the third track and fade it in
                t3.play()
                t3.fadeIn(2000)

                //play t2 only if it is longer than 10 seconds (it isnt)
                if(t2.getLength() > 10000) then
                        t2.play()
                else
                        t1.play()
                end

                //cant fade groups! (*ERROR*)
                //g1.fadeOut()

        end

        //needed for parser
        playOrder(A)
end
```

6.1.6 CommandTest.java

Below is the Java output that our MARS compiler generated.

```
package Tests;
import mars_util.*;
public class CommandTest{
public static void main(String[] args) {
A();
}
static void A() {
Track t1= new Track("file:/C:/sounds/rap_keys.wav");

Track t2= new Track("file:/C:/sounds/rap_guitar.wav");

Track t3= new Track("file:/C:/sounds/bridge_bass.wav");

t1.play();
t2.play();
```

```
{
Track[] tracksToMix=new Track[2];
double[] innerParams=new double[2];
tracksToMix[0] = t1;
innerParams[0] = -1.0;
tracksToMix[1] = t2;
innerParams[1] = -1.0;
Track.mix(tracksToMix,innerParams);
}
try {
Thread.sleep((long)5000);
}
catch (Exception e) {
System.out.println("Error while delaying");
}
t1.setVolume(2);
t1.loop(2);
t2.play(1000);
t1.setVolume(10);
t1.play();
t1.fadeOut();
t1.setVolume(10);
for (int g1=0;g1<2;g1++) {
t1.play();
t2.play();
}
t3.play();
t3.fadeIn(2000);
if (t2.getLength() > 10000) {
t2.play();
}
else {
t1.play();
}
}
}
```

6.2 Automated Testing

Because our language involved music, it is very hard to use automated testing such as JUnit or another testing suite. In other languages, you can usually generate a visual output (either graphically or by text), but with music it really all depends on the users ability to hear if something is wrong.

The solution we came up with to solve this problem is taking a song that has already been made, chopping it up into different tracks and sections, and then mixing and matching them in an attempt to recreate portions of the song. This way, if something sounds different than the original song, we know something is either wrong with the MARS composition or with the language itself. Mike's friend from his undergraduate was nice enough to give us different parts of his song and Mike chopped those parts up into smaller sections that we more usable to mix and match.

6.2.1 OldTimer.mars

Below is the MARS composition that we created using the testing loops. It is a full composition that makes use of almost every aspect of the MARS language including: looping, mixing, playing, setting volume, conditionals, and defining and using entities.

```
/*
  OldTimer.mars
  is a full blown composition. an attempt to recreate the song
  "What's an Old-Timer Like You Want" by Team Goldie
 */

 def composition OldTimer
```

```
            //define some global verse tracks
            track verseBass = "C:/sounds/verse_bass.wav"
            track verseGuitar = "C:/sounds/verse_guitar.wav"
            track verseOrchestra = "C:/sounds/verse_orchestra.wav"

            //define some global stinger tracks
            track stingBass = "C:/sounds/stinger_bass.wav"
            track stingGuitar = "C:/sounds/stinger_guitar.wav"
            track stingOrchestra = "C:/sounds/stinger_orchestra.wav"

            //define the verse
            def section verse

                        //play this mix 4 times

                        for(i: 1 to 4 step 1) do
                                    mix(verseBass.play(), verseGuitar.play(), verseOrchestra.play())
                        end

                        //alternate a bunch of times!
                        for(i: 1 to 3 step 1) do

                                    mix(stingBass.play(), stingGuitar.play(), stingOrchestra.play())
                                    mix(verseBass.play(), verseGuitar.play(), verseOrchestra.play())
                        end

                        //play this stinger one more time
                        mix(stingBass.play(), stingGuitar.play(), stingOrchestra.play())
            end

            //define our bridge
            def section bridge

                        //define some bridge tracks
                        track bridgeBass = "C:/sounds/bridge_bass.wav"
                        track bridgeGuitar = "C:/sounds/bridge_guitar.wav"
                        track bridgeOrchestra = "C:/sounds/bridge_orchestra.wav"

                        //just mix these three for the bridge
                        mix(bridgeBass.play(), bridgeGuitar.play(), bridgeOrchestra.play())
            end

            //define our chorus
            def section chorus

                        //define some chorus tracks
                        track chorusBass = "C:/sounds/chorus_bass.wav"
                        track chorusGuitar = "C:/sounds/chorus_guitar.wav"
                        track chorusGuitar2 = "C:/sounds/chorus_guitar2.wav"
                        track chorusOrchestra = "C:/sounds/chorus_orchestra.wav"
                        track chorusKeys = "C:/sounds/chorus_keys.wav"

                        //mix these to get the main chorus
                        //note, guitar 2 only loops twice!
                        //mix(chorusGuitar2.loop(2), chorusBass.loop(4), chorusGuitar.loop(4),
chorusOrchestra.loop(4), chorusKeys.loop(4))

                        for(i: 1 to 2 step 1) do
                                    mix(chorusGuitar2.play(), chorusBass.play(), chorusGuitar.play(),
chorusOrchestra.play(), chorusKeys.play())
                        end
            end

            //define the guitar solo section
            def section solo

                        //define guitar solo track
                        track soloGuitar = "C:/sounds/solo_guitarSolo.wav"

                        //now we have the guitar solo...only loop that twice
                        //but mix it with the verse
                        //mix(soloGuitar.loop(2), verseBass.loop(4), verseGuitar.loop(4),
verseOrchestra.loop(4))

                        //just play the solo for now
                        //soloGuitar.play()

                        mix(soloGuitar.play(), verseBass.play(), verseGuitar.play(),
verseOrchestra.play())
            end

            //define the rap section
            def section rap
```

```
                    //define some stinger tracks
                    track chorusStingGuitar2 = "C:/sounds/chorusStinger_guitar2.wav"
                    track chorusStingKeys = "C:/sounds/chorusStinger_keys.wav"


                    //define our rap tracks
                    track rapIntro = "C:/sounds/rapIntro_guitar.wav"
                    track rapGuitar = "C:/sounds/rap_guitar.wav"
                    track rapBass = "C:/sounds/rap_bass.wav"
                    track rapKeys = "C:/sounds/rap_keys.wav"
                    track rapOutroBass = "C:/sounds/rapOutro_bass.wav"
                    track rapOutroGuitar = "C:/sounds/rapOutro_guitar.wav"
                    track rapOutroKeys = "C:/sounds/rapOutro_keys.wav"

                    //play the rap intro
                    mix(chorusStingGuitar2.play(), chorusStingKeys.play(), rapIntro.play())

                    //loop this rap chorus, increase volume each iteration
                    for(i: 1 to 3 step 1) do

                              //set the volume according to the iteration
                              if(i == 1) then
                                        rapBass.setVolume(4)
                                        rapGuitar.setVolume(4)
                                        rapKeys.setVolume(4)
                              end

                              if(i == 2) then
                                        rapBass.setVolume(7)
                                        rapGuitar.setVolume(7)
                                        rapKeys.setVolume(7)
                              end

                              if(1 == 3) then
                                        rapBass.setVolume(10)
                                        rapGuitar.setVolume(10)
                                        rapKeys.setVolume(10)
                              end

                              //mix the rap guitar and keys for the chorus
                              mix(rapBass.play(), rapGuitar.play(), rapKeys.play())
                    end


                    //mix the outro to end it!
                    mix(rapOutroBass.play(), rapOutroGuitar.play(), rapOutroKeys.play())
          end


          //play this sucker!
          playOrder(verse, bridge, chorus, solo, verse, bridge, chorus, rap)
end
```

6.2.2 OldTimer.java

Below is the Java composition that our MARS compiler generated. It can be compiled used the Java compiler.

```
package Tests;
import mars_util.*;
public class OldTimer{
static Track verseBass= new Track("file:/C:/sounds/verse_bass.wav");
static Track verseGuitar= new Track("file:/C:/sounds/verse_guitar.wav");
static Track verseOrchestra= new Track("file:/C:/sounds/verse_orchestra.wav");
static Track stingBass= new Track("file:/C:/sounds/stinger_bass.wav");
static Track stingGuitar= new Track("file:/C:/sounds/stinger_guitar.wav");
static Track stingOrchestra= new Track("file:/C:/sounds/stinger_orchestra.wav");
public static void main(String[] args) {
verse();
bridge();
chorus();
solo();
verse();
bridge();
chorus();
rap();
```

```
}
static void verse() {
for (int i=1;i<=4;i+=1) {
{
Track[] tracksToMix=new Track[3];
double[] innerParams=new double[3];
tracksToMix[0] = verseBass;
innerParams[0] = -1.0;
tracksToMix[1] = verseGuitar;
innerParams[1] = -1.0;
tracksToMix[2] = verseOrchestra;
innerParams[2] = -1.0;
Track.mix(tracksToMix,innerParams);
}
}
for (int i=1;i<=3;i+=1) {
{
Track[] tracksToMix=new Track[3];
double[] innerParams=new double[3];
tracksToMix[0] = stingBass;
innerParams[0] = -1.0;
tracksToMix[1] = stingGuitar;
innerParams[1] = -1.0;
tracksToMix[2] = stingOrchestra;
innerParams[2] = -1.0;
Track.mix(tracksToMix,innerParams);
}
{
Track[] tracksToMix=new Track[3];
double[] innerParams=new double[3];
tracksToMix[0] = verseBass;
innerParams[0] = -1.0;
tracksToMix[1] = verseGuitar;
innerParams[1] = -1.0;
tracksToMix[2] = verseOrchestra;
innerParams[2] = -1.0;
Track.mix(tracksToMix,innerParams);
}
}
{
Track[] tracksToMix=new Track[3];
double[] innerParams=new double[3];
tracksToMix[0] = stingBass;
innerParams[0] = -1.0;
tracksToMix[1] = stingGuitar;
innerParams[1] = -1.0;
tracksToMix[2] = stingOrchestra;
innerParams[2] = -1.0;
Track.mix(tracksToMix,innerParams);
}
}
static void bridge() {
Track bridgeBass= new Track("file:/C:/sounds/bridge_bass.wav");

Track bridgeGuitar= new Track("file:/C:/sounds/bridge_guitar.wav");

Track bridgeOrchestra= new Track("file:/C:/sounds/bridge_orchestra.wav");

{
Track[] tracksToMix=new Track[3];
double[] innerParams=new double[3];
tracksToMix[0] = bridgeBass;
innerParams[0] = -1.0;
tracksToMix[1] = bridgeGuitar;
innerParams[1] = -1.0;
tracksToMix[2] = bridgeOrchestra;
innerParams[2] = -1.0;
Track.mix(tracksToMix,innerParams);
}
}
static void chorus() {
Track chorusBass= new Track("file:/C:/sounds/chorus_bass.wav");

Track chorusGuitar= new Track("file:/C:/sounds/chorus_guitar.wav");

Track chorusGuitar2= new Track("file:/C:/sounds/chorus_guitar2.wav");

Track chorusOrchestra= new Track("file:/C:/sounds/chorus_orchestra.wav");

Track chorusKeys= new Track("file:/C:/sounds/chorus_keys.wav");

for (int i=1;i<=2;i+=1) {
{
Track[] tracksToMix=new Track[5];
```

```
double[] innerParams=new double[5];
tracksToMix[0] = chorusGuitar2;
innerParams[0] = -1.0;
tracksToMix[1] = chorusBass;
innerParams[1] = -1.0;
tracksToMix[2] = chorusGuitar;
innerParams[2] = -1.0;
tracksToMix[3] = chorusOrchestra;
innerParams[3] = -1.0;
tracksToMix[4] = chorusKeys;
innerParams[4] = -1.0;
Track.mix(tracksToMix,innerParams);
}
}
}
static void solo() {
Track soloGuitar= new Track("file:/C:/sounds/solo_guitarSolo.wav");

{
Track[] tracksToMix=new Track[4];
double[] innerParams=new double[4];
tracksToMix[0] = soloGuitar;
innerParams[0] = -1.0;
tracksToMix[1] = verseBass;
innerParams[1] = -1.0;
tracksToMix[2] = verseGuitar;
innerParams[2] = -1.0;
tracksToMix[3] = verseOrchestra;
innerParams[3] = -1.0;
Track.mix(tracksToMix,innerParams);
}
}
static void rap() {
Track chorusStingGuitar2= new Track("file:/C:/sounds/chorusStinger_guitar2.wav");

Track chorusStingKeys= new Track("file:/C:/sounds/chorusStinger_keys.wav");

Track rapIntro= new Track("file:/C:/sounds/rapIntro_guitar.wav");

Track rapGuitar= new Track("file:/C:/sounds/rap_guitar.wav");

Track rapBass= new Track("file:/C:/sounds/rap_bass.wav");

Track rapKeys= new Track("file:/C:/sounds/rap_keys.wav");

Track rapOutroBass= new Track("file:/C:/sounds/rapOutro_bass.wav");

Track rapOutroGuitar= new Track("file:/C:/sounds/rapOutro_guitar.wav");

Track rapOutroKeys= new Track("file:/C:/sounds/rapOutro_keys.wav");

{
Track[] tracksToMix=new Track[3];
double[] innerParams=new double[3];
tracksToMix[0] = chorusStingGuitar2;
innerParams[0] = -1.0;
tracksToMix[1] = chorusStingKeys;
innerParams[1] = -1.0;
tracksToMix[2] = rapIntro;
innerParams[2] = -1.0;
Track.mix(tracksToMix,innerParams);
}
for (int i=1;i<=3;i+=1) {
if (i == 1) {
rapBass.setVolume(4);
rapGuitar.setVolume(4);
rapKeys.setVolume(4);
}
else {
}
if (i == 2) {
rapBass.setVolume(7);
rapGuitar.setVolume(7);
rapKeys.setVolume(7);
}
else {
}
if (1 == 3) {
rapBass.setVolume(10);
rapGuitar.setVolume(10);
rapKeys.setVolume(10);
}
else {
}
```

```
{
Track[] tracksToMix=new Track[3];
double[] innerParams=new double[3];
tracksToMix[0] = rapBass;
innerParams[0] = -1.0;
tracksToMix[1] = rapGuitar;
innerParams[1] = -1.0;
tracksToMix[2] = rapKeys;
innerParams[2] = -1.0;
Track.mix(tracksToMix,innerParams);
}
}
{
Track[] tracksToMix=new Track[3];
double[] innerParams=new double[3];
tracksToMix[0] = rapOutroBass;
innerParams[0] = -1.0;
tracksToMix[1] = rapOutroGuitar;
innerParams[1] = -1.0;
tracksToMix[2] = rapOutroKeys;
innerParams[2] = -1.0;
Track.mix(tracksToMix,innerParams);
}
}
}
```

7. Lessons Learned

7.1 Aaron's lessons

"By far, the most important thing I learned from working on our project was that the analysis and design phases of a project are crucial - although we "studied" it in school, spending time on good design made the rest of the project (coding and testing) easier to handle. It is my opinion that trying to make the code as modular as possible is the best way to avoid future problems. Moreover, it helps when features need to be added or removed."

7.2 Ritika's lessons

"The lesson I learned was that as soon as possible we should define the semantics of the language clearly. This is hard to do as one is not clear about exactly how the back-end part will be implemented, but maybe touching base with all the project members regularly will help. Also, it is very important that one person is aware of both the back-end and front-end and how they are integrated."

7.3 Swapneel's lessons

"When you don't know what the code will look like and need to do later, try to make it as modular and clean as possible. Also make sure to use version control. It will save a lot of time. Using Eclipse makes life a lot easier. Always try to design your code to be as generic as possible, especially the ancillary files."

7.4 Mike's lessons

"Well of course the first thing is to start everything earlier. The end of the semester is busy enough for everyone, not including this project. Live close to campus so it is easy to meet up with your group anytime. Realize that not everything is going to go as planned, and just adapt to the way things go."

7.5 Advice for future teams

The main advice we have for future teams is to start early, set your deadlines and try to meet them regardless of how busy you are with other things. Because there are no hard deadlines set forth by the class aside from the White Paper and Language Reference Manual, its very easy to get into the groove of putting PLT as a last priority...until the end of the semester.

Another point of advice is that this is a technical and creative project, and like any creative project, it is very easy to be overambitious of what you can accomplish in a single semester. Make sure you set realistic goals of what your language can do. Start with a small number of features, and then if time allows it, slowly add features to your language. If you start with too many features, there may be dependencies; however, cutting those features may disrupt other aspects of your language.

8. Appendix: Code Listing

8.1 mars package

The mars package contains all front end source code files such as the Mars.g ANTLR code used to generate the Lexer, Parser, and Walker. Also contains the Symbol Table and MARS primitive types.

8.1.1 Mars.g – ANTLR code for Lexer

```
//.g file for Mars

header {
package mars;
import antlr.*;
import antlr.CommonAST;
import java.util.*;
}

//**************Lexer****************/
//@author swapneelsheth, mike

class MarsLexer extends Lexer;
            options{
                        k=2;
            //Look-Ahead for String Literal....Need to check
                        testLiterals = false;
            }

            protected DIGIT : '0'..'9';
            protected LETTER : ('a'..'z') | ('A'..'Z');
            protected FILETYPE : (SND | AU | WAV);
            protected WAV : ("WAV" | "wav");
            protected SND : ("SND" | "snd");
            protected AU : ("AU" | "au");
            protected DOUBLE : ;

            INT : (DIGIT)+ ("." (DIGIT)+ {$setType(DOUBLE);})?;

            ID
            options {
                        testLiterals = true;    //ID cannot be a keyword
            }

                        : LETTER (LETTER | DIGIT)* ;

            FILENAME : '"'! ((LETTER | DIGIT | '_' | '-' | '/' | '\\' | ':')+ PERIOD FILETYPE) '"'!;

            WHITESPACE : (' ' | '\t' | '\n' {newline();} | '\r' '\n' {newline();})+
{$setType(Token.SKIP);};

            //Multi Line Comments
            COMMENT : "/*" (options {greedy=false;}: .)* "*/" {$setType(Token.SKIP);}          ;

            COMMENT_LINE : "//" (~('\n' | '\r'))* {$setType(Token.SKIP);} ;

            //Operators, etc.
            PLUS : '+';
            MINUS : '-';
            MUL : '*';
            DIV : '/';
            MOD : '%';
            POW : "**";
            LPAREN : '(';
            RPAREN : ')';
            NOT: '!';
            EQ : "==";
            NEQ : "!=";
            GT : '>';
            GE : ">=";
            LT : '<';
            LE : "<=";
```

```
ASSIGN : '=';
AND : "&&";
OR : "||";
COMMA : ',';
COLON : ':';
PERIOD : '.';
```

8.1.2 Mars.g – ANTLR code for Parser

```
//*************** PARSER ***************************//
//@author swapneelsheth, mike

            class MarsParser extends Parser;
            options {
                        ASTLabelType = "mars.MarsAST";
                        buildAST = true;
                        k = 2;
            }

            tokens{
                        ARGLIST;
                        PARAMLIST;
            }

            //a whole file
            file : decls_comp EOF!;

            //declarations of comps, secions, groups, primitives
            decls_comp: "def"! "composition"^ ID (decls)* play_order "end"!;
            decls: decls_section | decls_group | decls_custom | decls_others;
            decls_section: "def"! "section"^ ID (decls_group | decls_custom | decls_others | stmt)*
"end"!;
            decls_group: "def"! "group"^ ID (decls_custom | decls_others | stmt)* "end"!;
            decls_custom: "def"^ ID LPAREN! (arglist)? RPAREN! (decls_others | stmt)* "end"!;
            decls_others: "track"^ ID ASSIGN! FILENAME
                        | ("int"^) ID (ASSIGN! INT)?
                        | ("double"^) ID (ASSIGN! DOUBLE)?;
            play_order: "playOrder"^ LPAREN! paramlist RPAREN!;

            //a statement
            stmt: ctrl
                        | loop
                        | assign_expr
                        | func_call;

            //things that make up a statement
            ctrl: "if"^ LPAREN! bool_expr RPAREN! "then" (decls_others | stmt)*
                                    (options { greedy = true; }: "else" (decls_others | stmt)*)? "end"!;
            loop: "for"^ LPAREN! ID COLON! (INT) "to"! (INT) ("step" (INT))? RPAREN! "do"! (decls_others
| stmt)* "end"!;

            assign_expr: ID ASSIGN^ arith_expr;

            func_call: (ID PERIOD!)? ID^ LPAREN! (paramlist)? RPAREN!;

            //boolean expressions cannot be arithmetic expressions
            bool_expr: logic_term (OR^ logic_term )*;
            logic_term: logic_factor (AND^ logic_factor)*;
            logic_factor: (NOT^)? equality;
            equality: relational ((EQ^ | NEQ^) relational)*;
            relational: bool_base ((GT^ | GE^ | LT^ | LE^) bool_base)*;
            bool_base: (func_call | ID | INT | DOUBLE | (LPAREN! bool_expr RPAREN!));

            //arithmetic expressions can not be boolean expressions
            arith_expr: arith_term ((PLUS^ | MINUS^) arith_term)*;
            arith_term: arith_pow ((MUL^ | DIV^ | MOD^) arith_pow)*;
            arith_pow : arith_base (POW^ arith_base)*;
            arith_base: (func_call | ID | INT | DOUBLE | (LPAREN! arith_expr RPAREN!));

            //Various "Types"
            type : "section" | "group" | "track" | "int" | "double";

            //For Custom Behavior
            arg: type ID;
            arglist: arg (COMMA! arg)* {#arglist = #([ARGLIST, "arglist"], arglist);} ;
            paramlist: (arith_expr)  (COMMA! arith_expr)* {#paramlist = #([PARAMLIST, "paramlist"],
paramlist);};
```

8.1.3 Mars.g – ANTLR code for Tree Walker

```
//*************** TREE WALKER ***************************//
//@author swapneelsheth, aaron

            class MarsWalker extends TreeParser;
            {SymTab s0 = new SymTab(null);
            SymTab current = s0;
            String sign = null;
            String currentGroup = null;
            boolean playOrderFlag = false;}

            file returns [Composition c]
            {c=null;}

             : c=decls_comp;

            decls_comp returns [Composition c]
                                                    {SymTab sInner = new SymTab(current);
                                                    current = sInner;
                                                    FunctionCall f=null;
                                                    c = null;
                                                    }

                                        :       #("composition" a:ID

                                                                {c = new
Composition(a.getText());

                                                                        sInner.put(c);


                                                                        }


                                                (

                                                        {Section d;}

                                                        (d = decls_section)
{c.addSection(d);sInner.put(d);}


                                                        | {Group g;}

                                                        (g = decls_group)
{c.addGroup(g);sInner.put(g);}


                                                        | {Track t;}

                                                        (t = decls_track)
{c.addTrack(t);sInner.put(t);}


                                                        | {Type type;}

                                                        (type = decls_primitive)
{c.addPrimitive(type);}


                                                        | {CustomBehaviour cb;}

                                                        (cb = decls_custom) {sInner.put(cb);}


                                                        |

                                                        (f = play_order)        {c.addStatement(f);}


                                                )*
```

```
                                                                                {current =
sInner.getOuter();

                                                                                }

                                                                                );


            play_order returns [FunctionCall f]

            {           f = null;
                        Vector v = null;}

            : #(a:"playOrder" v=paramlist                                       {


                                                                    if (v.size() == 0) throw new
SemanticException ("Line "+a.getLine()+": Atleast 1 Argument needed for Function Call playOrder");

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

                                                                                String param =
v.get(i).toString();

                                                                                if (!(current.get(param)
instanceof mars.Section)) throw new SemanticException ("Line "+a.getLine()+": Invalid Argument "+param+"
for Function Call playOrder");

                                                                    }

                                                                    f = new FunctionCall("playOrder");

                                                                    f.addParam(v);

                                                        }

                                                );


            stmt returns [Statement s]
                                                                    {s = null;}

                                                        : s = func_call
                                                        | s = loop
                                                        | s = ctrl
                                                        | s = assign_expr;


            decls_section returns [Section s]
                                                                    {SymTab sInner = new SymTab(current);
                                                                    current = sInner;
                                                                    s = null;
                                                                    Statement st = null;}

                                                        :           #("section" name:ID

                                                                                {if
(current.getOuter().isDefined(name.getText())) throw new SemanticException("Line "+name.getLine()+":
"+name.getText()+" is already defined");

                                                                                s = new
Section(name.getText());

                                                                                sInner.put(s);

                                                                                }



                                                        (

                                                                    {Group g;}

                                                                    (g = decls_group)
{s.addGroup(g);sInner.put(g);}



                                                                    | {Track t;}

                                                                    (t = decls_track)
{s.addTrack(t);sInner.put(t);}
```

```
                                                        | {CustomBehaviour cb;}
                                                        (cb = decls_custom) {sInner.put(cb);}

                                                        | {Type type;}
                                                        (type = decls_primitive)
{s.addPrimitive(type);}

                                                        |           st = stmt
                                                                    {s.addStatement(st);}
                                                )*

                                                                                {current =
sInner.getOuter();
                                                                                }
                                                                                );

            decls_group returns [Group g]
                                                        {SymTab sInner = new SymTab(current);
                                                        current = sInner;
                                                        g = null;
                                                        Statement s = null;
                                                        }
                                                :           #("group" name:ID
                                                {currentGroup = name.getText();
                                                                                if
(current.getOuter().isDefined(name.getText())) throw new SemanticException("Line " + name.getLine() + ":
"+name.getText()+" is already defined");
                                                                                g = new
Group(name.getText());
                                                                                sInner.put(g);
                                                                                }

                                                (
                                                        {Track t;}
                                                        (t = decls_track)
{g.addTrack(t);sInner.put(g);}

                                                        | {CustomBehaviour cb;}
                                                        (cb = decls_custom) {sInner.put(cb);}

                                                        | {Type type;}
                                                        (type = decls_primitive)
{g.addPrimitive(type);}

                                                        |           s = stmt
                                                                    {g.addStatement(s);}
                                                )*
```

```
                                                                                {current =
sInner.getOuter();

                                                                                currentGroup = null;}

                                                                                );


            decls_custom returns [CustomBehaviour cb]
                                                        {SymTab sInner = new SymTab(current);
                                                        current = sInner;
                                                        Vector v = new Vector<Object>();
                                                        cb = null;
                                                        Statement s = null;
                                                        }

                                                :       #("def" name:ID (v=arglist)?
                        {if (current.getOuter().isDefined(name.getText())) throw new
SemanticException("Line " + name.getLine() + ": "+name.getText()+" is already defined");

                                                                                cb = new
CustomBehaviour(name.getText());

                                                                                cb.addArg(v);

                                                                                sInner.put(cb);}


                                        (               {Type type;}

                                                        (type = decls_primitive)
{cb.addPrimitive(type);}


                                                        | s = stmt

                                                        {cb.addStatement(s);}

                                        )*

                                                                                {

                                                                                current =
sInner.getOuter();

                                                                                });


            arglist returns [Vector v]
                                                        {v = new Vector<String>();}

                                                :  #(ARGLIST
                                                                        (argType:type argName:ID
                        {v.add(argType.getText());


            v.add(argName.getText());

                                                                                }
                                                                        )+);



            decls_primitive returns [Type t]

            {t = null;}

             : (t=decls_int) | (t=decls_dbl);

            type : "section" | "group" | "track" | "int" | "double";


            decls_track returns [Track t]
                                                        {t = null;}

                                                :   #("track" name:ID fname:FILENAME
            {if (current.get(name.getText()) != null) throw new SemanticException("Line " +
name.getLine() + ": "+name.getText()+" is alreadydefined");

                                                                                t = new
```

```
Track(name.getText(), fname.getText());
                                                                    current.put(t);
                                                                    });
            decls_int returns [Int i]
            {i = null;}
                                          :          #("int" name:ID (value:INT)?
                        {if (current.get(name.getText()) != null) throw new SemanticException("Line " +
name.getLine() + ": "+name.getText()+" is alreadydefined");
                                                                    if (value == null) i
= new Int(name.getText());
                                                                    else i = new
Int(name.getText(), Integer.parseInt(value.getText()));
                                                                    current.put(i);
                                                                    });
            decls_dbl returns [Dbl di]
            {di = null;}
                                          :          #("double" name:ID (value:DOUBLE)?
            {if (current.get(name.getText()) != null) throw new SemanticException("Line " +
name.getLine() + ": "+name.getText()+" is alreadydefined");
                                                                    if (value == null) di
= new Dbl(name.getText());
                                                                    else di = new
Dbl(name.getText(), Double.parseDouble(value.getText()));
                                                                    current.put(di);
                                                                    });
            param : INT | ID | DOUBLE;
            func_call returns [FunctionCall f]
                                                    {Vector v = new Vector<String>();
                                                    f = null;
                                                    String classType = null;
                                                    Type typeFunction=null;}
                                          :          #(methodName:ID (className:ID)?
(v=paramlist)?
                                                                    {typeFunction =
current.get(methodName.getText());
                                                                    if (typeFunction ==
null || (!(typeFunction instanceof mars.CustomBehaviour) && !(typeFunction instanceof mars.SystemCall)))
throw new SemanticException("Line " + methodName.getLine() + ": Function "+methodName.getText()+" is not
defined");
                                                                    if (className !=
null) {
                                                                                Type t =
current.get(className.getText());
                                                                                if (t ==
null) throw new SemanticException("Line "+className.getLine()+": "+className.getText()+"is not
defined");
                                                                                else {
            SystemCall sc = (SystemCall) current.get(methodName.getText());
            if (!(sc.getCalledBy()).contains((t.getClass()).getName())) throw new SemanticException("Line
"+className.getLine()+": "+className.getText()+" cannot call method "+methodName.getText());
            else classType = (t.getClass()).getName();
```

```
                                                                                }
                                                                        }
                                                                        else {
                                                                                if
((current.get(methodName.getText()) instanceof mars.SystemCall) &&
(!(methodName.getText()).equals("playOrder")) && (!(methodName.getText()).equals("delay")) &&
(!(methodName.getText()).equals("mix"))) throw new SemanticException("Line "+methodName.getLine()+":
Incorrect call of method "+methodName.getText());
                                                                        }
                                                                        Vector<String>
argList = null;
                                                                        if (typeFunction
instanceof mars.CustomBehaviour) {
                                                                                argList =
((CustomBehaviour) typeFunction).getArgList();
                                                                        }


                                                                        if (typeFunction
instanceof mars.SystemCall) {
                                                                                argList =
((SystemCall) typeFunction).getArgList();
                                                                        }
                                                                        int size =
argList.size();
                                                                        if
(methodName.getText().equals("play") && v.size() == 0) {



                                                                        }
                                                                        else if
(methodName.getText().equals("play") && (current.get(className.getText()) instanceof mars.Group)) {
                                                                                if
(v.size() != 0) throw new SemanticException ("Line "+methodName.getLine()+": Group.play() cannot take any
arguments");
                                                                        }
                                                                        else if
(methodName.getText().equals("mix")) {
                                                                                if
(v.size() < 2) throw new SemanticException ("Line "+methodName.getLine()+": Atleast 2 Arguments needed
for Function Call "+methodName.getText());
                                                                                for (int
i=0; i<v.size(); i++) {

            String param = v.get(i).toString();

            String[] funcCall = param.split("::");

            if (funcCall.length != 2) throw new SemanticException ("Line "+methodName.getLine()+":
Invalid Argument "+param+" for Function Call mix");

            Type object = current.get(funcCall[0]);

            if (object == null) throw new SemanticException ("Line "+methodName.getLine()+":
"+funcCall[0]+" is not defined");

            if (!(object instanceof mars.Track)) throw new SemanticException ("Line
```

```
"+methodName.getLine()+": "+funcCall[0]+" has to be a Track");

            if (funcCall[1].contains(":")) {

                        String[] funcs = funcCall[1].split(":");

                        if (!(funcs[0].equals("play"))) throw new SemanticException ("Line
"+methodName.getLine()+": Function "+funcs[0]+" not allowed here");

                        Type number = current.get(funcs[1]);

                        if (number != null) {

                                    if (!(number instanceof mars.Int)) throw new SemanticException ("Line
"+methodName.getLine()+": Invalid Argument "+funcs[1]+" for Function Call "+funcs[0]);


                        }

                        else {

                                    try {

                                                Integer integer = new Integer(funcs[1]);

                                    }

                                    catch (Exception e) {

                                                throw new SemanticException ("Line
"+methodName.getLine()+": Invalid Argument "+funcs[1]+" for Function Call "+funcs[0]);

                                    }

                        }

            }

            else {

                        if (!(funcCall[1].equals("play"))) throw new SemanticException ("Line
"+methodName.getLine()+": Function "+funcCall[1]+" not allowed here");

            }
                                                                                                    }
                                                                                        }
                                                                                        else {
                                                                                                    if
(v.size() != (size/2)) throw new SemanticException ("Line "+methodName.getLine()+": Invalid Number of
Arguments for "+methodName.getText());
                                                                                                    for (int
i=0; i<size; i=i+2) {

            String arg = argList.get(i);

            if (arg.equals("track")) {
```

```
                        String param = v.get(i/2).toString();

                        if (!(current.get(param) instanceof mars.Track)) throw new SemanticException
("Line "+methodName.getLine()+": Invalid Argument "+param+" for Function Call "+methodName.getText());

            }

            else if (arg.equals("group")) {

                        String param = v.get(i/2).toString();

                        if (!(current.get(param) instanceof mars.Group)) throw new SemanticException
("Line "+methodName.getLine()+": Invalid Argument "+param+" for Function Call "+methodName.getText());

            }

            else if (arg.equals("section")) {

                        String param = v.get(i/2).toString();

                        if (!(current.get(param) instanceof mars.Section)) throw new SemanticException
("Line "+methodName.getLine()+": Invalid Argument "+param+" for Function Call "+methodName.getText());

            }

            else if (arg.equals("int")) {

                        String param = v.get(i/2).toString();

                        Type t = current.get(param);

                        if (t != null) {

                                    if (!(t instanceof mars.Int)) throw new SemanticException ("Line
"+methodName.getLine()+": Invalid Argument "+param+" for Function Call "+methodName.getText());

                        }

                        else {

                                    if (param.contains("::")) {

                                                String[] funcCall = param.split("::");

                                                Type object = current.get(funcCall[0]);

                                                Type function = current.get(funcCall[1]);

                                                if (object == null) throw new SemanticException ("Line
"+methodName.getLine()+": "+funcCall[0]+" is not defined");

                                                if (function == null) throw new SemanticException ("Line
"+methodName.getLine()+": Function "+funcCall[1]+" is not defined");

                                                if (!(function instanceof mars.SystemCall)) throw new
SemanticException ("Line "+methodName.getLine()+": Cannot Call Function "+funcCall[1]);

                                                Type returnType = ((SystemCall)
function).getReturnType();
```

```
                                        SystemCall sc = (SystemCall) function;

                                        if (!(returnType instanceof mars.Int)) throw new
SemanticException ("Line "+methodName.getLine()+": Invalid Argument "+param+" for Function Call
"+methodName.getText());

                                        if
(!(sc.getCalledBy()).contains((object.getClass()).getName())) throw new SemanticException("Line
"+methodName.getLine()+": "+funcCall[0]+" cannot call method "+funcCall[1]);

                                }

                                else {

                                        try {

                                                Integer integer = new Integer(param);

                                        }

                                        catch (Exception e) {

                                                throw new SemanticException ("Line
"+methodName.getLine()+": Invalid Argument "+param+" for Function Call "+methodName.getText());

                                        }

                                }

                        }

            }

            else if (arg.equals("double")) {

                        String param = v.get(i/2).toString();

                        Type t = current.get(param);

                        if (t != null) {

                                if (!(t instanceof mars.Dbl)) throw new SemanticException ("Line
"+methodName.getLine()+": Invalid Argument "+param+" for Function Call "+methodName.getText());

                        }

                        else {

                                if (param.contains("::")) {

                                        String[] funcCall = param.split("::");

                                        Type object = current.get(funcCall[0]);

                                        Type function = current.get(funcCall[1]);

                                        if (object == null) throw new SemanticException ("Line
```

```
"+methodName.getLine()+": "+funcCall[0]+" is not defined");

                                        if (function == null) throw new SemanticException ("Line
"+methodName.getLine()+": Function "+funcCall[1]+" is not defined");

                                        if (!(function instanceof mars.SystemCall)) throw new
SemanticException ("Line "+methodName.getLine()+": Cannot Call Function "+funcCall[1]);

                                        Type returnType = ((SystemCall)
function).getReturnType();

                                        SystemCall sc = (SystemCall) function;

                                        if (!(returnType instanceof mars.Dbl)) throw new
SemanticException ("Line "+methodName.getLine()+": Invalid Argument "+param+" for Function Call
"+methodName.getText());

                                        if
(!(sc.getCalledBy()).contains((object.getClass()).getName())) throw new SemanticException("Line
"+methodName.getLine()+": "+funcCall[0]+" cannot call method "+funcCall[1]);

                                }

                                else {

                                        try {

                                                Double dbl = new Double(param);

                                                if (methodName.getText().equals("setVolume")
&& (dbl < 0 || dbl > 10)) System.out.println("WARNING: For Function Call setVolume, argument has to be
between 0 and 10. Rounding Up/Down as obvious");

                                        }

                                        catch (Exception e) {

                                                throw new SemanticException ("Line
"+methodName.getLine()+": Invalid Argument "+param+" for Function Call "+methodName.getText());

                                        }

                                }

                        }

                }
                                                                                        }
                                                                                }


                                                                                if (className ==
null) f = new FunctionCall(methodName.getText());

                                                                                else f = new
FunctionCall(methodName.getText(), className.getText(), classType);

                                                                                f.addParam(v);

                                                                                });


                paramlist returns [Vector v]
```

```
                                                {v = new Vector<String>();}
                                        :  #(PARAMLIST
                                                        (argName:param
                                                {if (argName.getType() == ID) {
                                                                                                Type t =
current.get(argName.getText());
                                                                                                String s
= argName.getText();
                                                                                                if (t
instanceof mars.SystemCall) {
            String child = (argName.getFirstChild()).getText();
            s = child + "::" + s;
            if (argName.getNumberOfChildren() > 1) {
                        s = s + ":" + argName.getFirstChild().getNextSibling().getFirstChild().getText();
            }
                                                                                                }
                                                                                                v.add(s);
                                                                                        }
                                                                                        else
v.add(argName.getText());
                                                                                        }
                                                                                )+);
            number : INT | DOUBLE;
            loop returns [For f]
                                                {SymTab sInner = new SymTab(current);
                                                current = sInner;
                                                int stepValue = 1;
                                                f = null;
                                                Vector<Statement> body = new
Vector<Statement>();
                                                Vector<Type> primitives = new Vector<Type>();
                                                Statement s = null;
                                                boolean nameDefined = false;}
                                        :       #("for" name:ID {if
(current.get(name.getText()) !=null) nameDefined=true;}
            lowerBound:number upperBound:number ("step" step:number {stepValue =
Integer.parseInt(step.getText());})?
                                    {sInner.put(new Int(name.getText()));}
                                    (
                                                {Type type;}
                                                (type = decls_primitive) {primitives.add(type);}
                                                |       s = stmt
                                                {body.add(s);}
```

```
                                    )*

                                                                                                {f = new
For(name.getText(), Integer.parseInt(lowerBound.getText()), Integer.parseInt(upperBound.getText()),
stepValue, body, primitives);


            f.setVariableDefined(nameDefined);

                                                                                                current =
sInner.getOuter();

                                                                                                });

            if_check returns [BoolExpr bool]
            {int lValue = 0;
            int rValue = 0;
            int INTVALUE = 1;
            int DOUBLEVALUE = 2;
            bool = null;
            String aText = null;
            String bText = null;
            boolean sysCallFlagA = false;
            boolean sysCallFlagB = false;}


            :           (
                                    a:param b:param        {


                                                                                    if (a.getType() == INT) lValue =
INTVALUE;

                                                                                    else if (a.getType() == DOUBLE)
lValue = DOUBLEVALUE;

                                                                                    else {

                                                                                                Type t =
current.get(a.getText());

                                                                                                if (t != null) {

                                                                                                            if (t
instanceof mars.Int) lValue = INTVALUE;

                                                                                                            else if
(t instanceof mars.Dbl) lValue = DOUBLEVALUE;

                                                                                                            else if
(t instanceof mars.SystemCall) {


            sysCallFlagA = true;


            if (((SystemCall)t).getReturnType() instanceof mars.Int) lValue = INTVALUE;


            else if (((SystemCall)t).getReturnType() instanceofmars.Dbl) lValue = DOUBLEVALUE;


            else throw new SemanticException ("Line "+a.getLine()+": "+a.getText()+" returns NULLwhich
is not valid");

                                                                                                            }

                                                                                                            else
throw new SemanticException ("Line "+a.getLine()+": "+a.getText()+" is not a Valid Type for Comparison");

                                                                                                }

                                                                                                else throw new
SemanticException ("Line "+a.getLine()+": "+a.getText()+" is notdefined");

                                                                                    }

                                                                                    if (b.getType() == INT) rValue =
INTVALUE;
```

```
                                                                        else if (b.getType() == DOUBLE)
rValue = DOUBLEVALUE;
                                                                        else {
                                                                                Type t =
current.get(b.getText());
                                                                                if (t != null) {
                                                                                        if (t
instanceof mars.Int) rValue = INTVALUE;
                                                                                        else if
(t instanceof mars.Dbl) rValue = DOUBLEVALUE;
                                                                                        else if
(t instanceof mars.SystemCall) {
            sysCallFlagB = true;
            if (((SystemCall)t).getReturnType() instanceof mars.Int) rValue = INTVALUE;
            else if (((SystemCall)t).getReturnType() instanceofmars.Dbl) rValue = DOUBLEVALUE;
            else throw new SemanticException ("Line "+b.getLine()+": "+b.getText()+" returns NULLwhich
is not valid");
                                                                                        }
                                                                                        else
throw new SemanticException ("Line "+b.getLine()+": "+b.getText()+" is not a Valid Type for Comparison");
                                                                                }
                                                                                else throw new
SemanticException ("Line "+b.getLine()+": "+b.getText()+" is notdefined");
                                                                        }
                                                                        aText = a.getText();
                                                                        bText = b.getText();
                                                                        if (sysCallFlagA) aText =
a.getFirstChild().getText() + "." + aText+"()";
                                                                        if (sysCallFlagB) bText =
b.getFirstChild().getText() + "." + bText+"()";
                                                                        bool = new BoolExpr("(" + aText +
sign + bText + ")");
                        );
                                                                        }
            if_cond returns [BoolExpr bool]
            {bool = null;
            BoolExpr b1 = null, b2 = null;}
            : (
                                                                                #(GT
{sign = " > ";}(bool=if_check))
                                                                                | #(GE
{sign = " >= ";}(bool=if_check))
                                                                                | #(LT
{sign = " < ";}(bool=if_check))
                                                                                | #(LE
{sign = " <= ";}(bool=if_check))
                                                                                | #(EQ
{sign = " == ";}(bool=if_check))
                                                                                | #(NEQ
{sign = " != ";}(bool=if_check))
                                                                                | #(NOT
(b1 = if_cond) {bool = new BoolExpr("( ! ( "+b1.getCondition()+") )");})
                                                                                | #(AND
            (b1=if_cond) (b2=if_cond) {bool = new BoolExpr("(" + b1.getCondition() + " && "
+b2.getCondition() + ")");})
```

```
                                                                                                    |#(OR
            (b1=if_cond) (b2=if_cond) {bool = new BoolExpr("(" + b1.getCondition() + " || "
+b2.getCondition() + ")");})
                                                                                                    );

            ctrl returns [If i]
                                                            {SymTab sInner = new SymTab(current);
                                                            current = sInner;
                                                            i = null;
                                                            Statement s = null;
                                                            BoolExpr bool = null;
                                                            Vector<Statement> thenPart = new
Vector<Statement>();
                                                            Vector<Statement> elsePart = new
Vector<Statement>();
                                                            Vector<Type> thenPrimitives = new
Vector<Type>();
                                                            Vector<Type> elsePrimitives = new
Vector<Type>();
                                                            }
                                                  :         #(a:"if"  (bool=if_cond) {
                                                            if (bool == null) throw new SemanticException
("Line " +a.getLine()+": Invalid Boolean Condition for IF Statement");
                                                            }
                                                                                     "then"    (
                                        {Type type;}
                                        (type = decls_primitive) {thenPrimitives.add(type);}

                                        |           s = stmt
                                                    {thenPart.add(s);}

                              )*
                              ("else" (
                                                            {Type type;}
                                                            (type = decls_primitive)
{elsePrimitives.add(type);}

                                                            | s = stmt
                                                            {elsePart.add(s);}
                                                  )*
                              )?
                                                                                {i = new If(bool,
thenPart, thenPrimitives, elsePart, elsePrimitives);
                                                                                current =
sInner.getOuter();
                                                                                });

            assign_expr returns [AssignExpr ae]
            {ae = null;
            AssignExpr ae1 = null, ae2 = null;
            String s = null;
            Type t = null;}
            :  #(ASSIGN a:ID                                          {t = current.get(a.getText());
            if (t == null) throw new SemanticException ("Line "+a.getLine()+": "+a.getText()+" is not
defined");
            else {
```

```
                        if (!((t instanceof mars.Int) || (t instanceof mars.Dbl))) throw new
SemanticException ("Line "+a.getLine()+": Illegal lValue for "+a.getText());

              }

            }

            (s=assign_check)

            {if (s == null) throw new SemanticException ("Line "+a.getLine()+": Illegal Assignment
Expression");

                        if (t instanceof mars.Int) s = "(int)" + s;

                        ae = new AssignExpr(t, s);}

                        );

            assign_check returns [String s]
            {String s1 = null, s2 = null;
            s = null;
            FunctionCall f = null;}

            :           (

                                                                                                    (#(PLUS
(s1=assign_check) (s2=assign_check) {s = "(" + s1 +" + "+ s2 + ")";})
                                                                                                    | #(MINUS
(s1=assign_check) (s2=assign_check) {s = "(" + s1 +" - "+ s2 + ")";})
                                                                                                    | #(MUL
(s1=assign_check) (s2=assign_check) {s = "(" + s1 +" * "+ s2 + ")";})
                                                                                                    | #(DIV
(s1=assign_check) (s2=assign_check) {s = "(" + s1 +" / "+ s2 + ")";})
                                                                                                    | #(MOD
(s1=assign_check) (s2=assign_check) {s = "(" + s1 +" % "+ s2 + ")";})
                                                                                                    | #(POW
(s1=assign_check) (s2=assign_check) {s = "(Math.pow(" + s1 +", "+ s2 + "))";})
                                                                                                    | f =
b:func_call {

            if (f != null) {

                        Type t = current.get(f.getName());

                        if (t instanceof mars.SystemCall) {

                                    Type returnType = ((SystemCall) t).getReturnType();

                                    if (!((returnType instanceof mars.Int) || (returnType
instanceof mars.Dbl))) throw new SemanticException ("Line "+b.getLine()+": "+f.getName() + " does not
return an Integer or Double");

                              }

                        else throw new SemanticException ("Line "+b.getLine()+": Method "+f.getName() + "
not allowed here");

                        if ((f.getName()).equals("playOrder")) throw new SemanticException ("Line
"+b.getLine()+": " + f.getName() + " does not return an Integer or Double");

                        s = f.getClassName()+"."+f.getName()+"()";

            }
                                                                                                    }
                                                                                                    |
a:number {

            if ( a.getType() == ID) {

                        Type t = current.get(a.getText());

                        if (t == null) throw new SemanticException ("Line "+a.getLine()+":
"+a.getText()+" is not defined");

                        else {

                                    if (t instanceof mars.SystemCall) {

                                                Type returnType = ((SystemCall) t).getReturnType();

                                                if (!((returnType instanceof mars.Int) || (returnType
instanceof mars.Dbl))) throw new SemanticException ("Line "+a.getLine()+": "+a.getText() + " does not
return an Integer or Double");
```

```
                                    }
                                    else if (!((t instanceof mars.Int) || (t instanceof mars.Dbl))) throw
new SemanticException ("Line "+a.getLine()+": "+a.getText() + " not allowed in rValue for an Assignment
Expression");
                            }
            }
            s = a.getText();}
                                                                                                    )
                            );
```

8.1.4 Main.java – Main entry point for the compiler

```
/**
 * Main Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

import antlr.debug.misc.*;
import java.io.*;
import antlr.SemanticException;

public class Main extends Thread{

            /**
             * @param args
             */
            public static void main(String[] args) {
                        try {
                                    if (args.length < 1) throw new IllegalArgumentException("File Name
must be specified");
                                    MarsLexer l = new MarsLexer(new FileInputStream(new File(args[0])));
                                    MarsParser p = new MarsParser(l);
                                    p.setASTNodeClass("mars.MarsAST");
                                    p.file();
                                    System.out.println(p.getAST().toStringList());
                                    ASTFrame f = new ASTFrame("Test Tree View", (mars.MarsAST)
p.getAST());
                                    f.setVisible(true);
                                    f.setSize(1024, 768);
                                    MarsWalker w = new MarsWalker();
                                    Composition theComp = w.file((mars.MarsAST) p.getAST());
                                    if (args[0].endsWith(".mars")) {
                                                String compName = args[0].substring(0,args[0].length()-
5);
                                                String[] splitted = compName.split("\\\\");
                                                if (!(splitted[splitted.length-
1].equals(theComp.getName()))) throw new Exception ("File Name should match Composition Name");
                                    }
                                    else throw new Exception ("File Extensions should be .mars");
                                    new backEnd.CodeGenerator(theComp, args[0]);
                                    sleep(1000);
                        }
                        catch(SemanticException e) {
                                    System.out.println(e.getLine()+e.getMessage());
                        }
                        catch (Exception e) {
                                    e.printStackTrace();

                        }
            }
}
```

8.1.5 SymTab.java – The MARS symbol table

```
/**
 * Symbol Table Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

import java.util.Hashtable;
import java.util.Vector;

public class SymTab {
            private Hashtable<String, Type> hash;
            private SymTab outer;

            /**
             * Constructor
             * @param outer is the Symbol Table of the Outer Scope
             */

            public SymTab(SymTab outer) {
                        this.outer = outer;
                        hash = new Hashtable<String, Type>();
                        if (outer == null) populate();
            }

            /**
             * Inserts a Type Object into the Symbol Table
             * The Symbol Table will be a Hash table mapping Name => Type Object
             * @param type is an Type Object. Type can be an int, double, track, group, section or
composition.
             */
            public void put(Type type) {
                        hash.put(type.getName(), type);
            }

            /**
             * Does a Lookup in the Symbol Tables
             * @param name is the Name of the Identifier to Search in the Symbol Table
             * @return A Type Object if found in the Symbol Tables, else NULL
             */
            public Type get(String name) {
                        for (SymTab t = this; t!=null; t=t.outer) {
                                    Type type = (Type)t.hash.get(name);
                                    if (type !=null) return type;
                        }
                        return null;
            }

            /**
             * @return the outer
             */
            public SymTab getOuter() {
                        return outer;
            }

            /**
             * Tests for Type Declaration in the current scope
             * @param name name of the Type to be checked for
             * @return true is a Type Object is defined in the same scope
             */
            public boolean isDefined(String name) {
                        Type type = (Type)this.hash.get(name);
                        if (type != null) return true;
                        else return false;
            }

            private void populate() {
                        SystemCall sc;
                        Vector<String> argList = new Vector<String>();

                        sc = new SystemCall("play", null);
                        sc.addCalledBy("mars.Track");
                        sc.addCalledBy("mars.Group");
                        argList.add("double");
                        argList.add("a");
                        sc.addArg(argList);
                        this.put(sc);

                        sc = new SystemCall("delay", null);
```

```
                        sc.addCalledBy("");
                        argList.clear();
                        argList.add("double");
                        argList.add("a");
                        sc.addArg(argList);
                        this.put(sc);

                        sc = new SystemCall("fadeIn", null);
                        sc.addCalledBy("mars.Track");
                        argList.clear();
                        argList.add("double");
                        argList.add("a");
                        sc.addArg(argList);
                        this.put(sc);

                        sc = new SystemCall("fadeOut", null);
                        sc.addCalledBy("mars.Track");
                        this.put(sc);

                        sc = new SystemCall("setVolume", null);
                        sc.addCalledBy("mars.Track");
                        sc.addCalledBy("mars.Group");
                        argList.clear();
                        argList.add("double");
                        argList.add("a");
                        sc.addArg(argList);
                        this.put(sc);

                        sc = new SystemCall("getLength", new Int("-1"));
                        sc.addCalledBy("mars.Track");
                        this.put(sc);

                        sc = new SystemCall("playOrder", null);
                        sc.addCalledBy("");
                        this.put(sc);

                        sc = new SystemCall("mix", null);
                        sc.addCalledBy("");
                        this.put(sc);

                        sc = new SystemCall("loop", null);
                        sc.addCalledBy("mars.Track");
                        sc.addCalledBy("mars.Group");
                        argList.clear();
                        argList.add("int");
                        argList.add("a");
                        sc.addArg(argList);
                        this.put(sc);

                }
}
```

8.1.6 MarsAST.java – A MARS abstract syntax tree

```
/**
 * MarsAST Class for adding line numbers to AST
 * @author swapneelsheth, aaron
 */

package mars;

import antlr.CommonAST;
import antlr.Token;

public class MarsAST extends CommonAST {
     private int line = 0;
     private int column = 0;

     /**
      * Constructor used to initialise the CommonAST Object
      */
     public void initialize(Token token) {
          super.initialize(token);
          line=token.getLine();
          column=token.getColumn();
     }

     /**
      * return the line
      */
     public int getLine(){
          return line;
     }

     /**
             * @return the column
             */
            public int getColumn() {
                        return column;
            }
}
```

8.1.7 AssignExpr.java – Assignment expressions for MARS

```
/**
 * Assignment Expression Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

public class AssignExpr implements Statement{
            private Type lValue;
            private String rValue;

            /**Constructor
             * @param lValue lValue of the Assignment Expression
             * @param rValue rValue of the Assignment Expression
             */
            public AssignExpr(Type lValue, String rValue) {
                        this.lValue = lValue;
                        this.rValue = rValue;
            }

            /**
             * @return the lValue
             */
            public Type getLValue() {
                        return lValue;
            }

            /**
             * @return the rValue
             */
            public String getRValue() {
                        return rValue;
            }
}
```

8.1.8 BoolExpr.java – Boolean expressions in MARS

```
/**
 * Boolean Expressions in MARS
 * @author swapneelsheth, aaron
 */

package mars;

public class BoolExpr implements Statement{
            private String condition;

            /**Constructor
             * @param condition boolean condition
             */
            public BoolExpr(String condition) {
                        this.condition = condition;
            }

            /**
             * @return the condition
             */
            public String getCondition() {
                        return condition;
            }
}
```

8.1.9 Composition.java – MARS representation of a composition

```
/**
 * Composition Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

import java.util.*;

public class Composition implements Type{
            private String name;
            private Vector<Section> sections;
            private Vector<Group> groups;
            private Vector<Track> tracks;
            private Vector<Statement> body;
            private Vector<Type> primitives;

            /**
             * @param name name of the composition
             * @param tempo tempo of the composition
             */
            public Composition(String name) {
                        this.name = name;
                        sections = new Vector<Section>();
                        groups = new Vector<Group>();
                        tracks = new Vector<Track>();
                        body = new Vector<Statement>();
                        primitives = new Vector<Type>();
            }

            /**
             * @return the name
             */
            public String getName() {
                        return name;
            }

            /**
             * @return the sections
             */
            public Vector<Section> getSections(){
                        return sections;
            }

            /**
             * @return the groups
             */
            public Vector<Group> getGroups() {
                        return groups;
            }

            /**
             * @return the tracks
             */
            public Vector<Track> getTracks() {
                        return tracks;
            }

            /**
             * Add a Section to a Composition
             * @param section name of the section to be added
             */
            public void addSection(Section section) {
                        sections.add(section);
            }

            /**
             * Add a Group to a Composition
             * @param group name of the group to be added
             */
            public void addGroup(Group group) {
                        groups.add(group);
            }

            /**
             * Add a Track to a Composition
             * @param track name of the track to be added
             */
            public void addTrack(Track track) {
```

```
                tracks.add(track);
        }

        /**
         * @return the body
         */
        public Vector<Statement> getBody() {
                return body;
        }

        /**
         * Add Statements to the Vector of Statements (body)
         * @param s statement to be added
         */
        public void addStatement(Statement s) {
                body.add(s);
        }

        /**
         * @return the primitives
         */
        public Vector<Type> getPrimitives() {
                return primitives;
        }

        /**
         * Add Primitive Declarations (mars.Int or mars.Dbl) to the primitives Vector
         * @param type object to be added
         */
        public void addPrimitive(Type type) {
                primitives.add(type);
        }
}
```

8.1.10 CustomBehaviour.java – Custom Behavior functionality in MARS

```
/**
 * Custom Behaviour Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

import java.util.*;


public class CustomBehaviour implements Type{
            private String name;
            private Vector<String> argList;
            private Vector<Statement> body;
            private Vector<Type> primitives;

            /**
             * Constructor
             * @param name
             */
            public CustomBehaviour(String name) {
                        this.name = name;
                        argList = new Vector<String>();
                        body = new Vector<Statement>();
                        primitives = new Vector<Type>();
            }

            /**
             * @return the arglist
             */
            public Vector<String> getArgList() {
                        return argList;
            }

            /**
             * @return the name
             */
            public String getName() {
                        return name;
            }

            /**
             * Adds the arguments to the Vector
             * @param args Adds the vector passed to argList
             */
            public void addArg(Vector args) {
                        for (int i=0; i<args.size(); i = i+2) {
                                    String type = (String)args.get(i);
                                    String name = (String)args.get(i+1);
                                    argList.add(type);
                                    argList.add(name);
                        }
            }

            /**
             * @return the body
             */
            public Vector<Statement> getBody() {
                        return body;
            }

            /**
             * Add Statements to the Vector of Statements (body)
             * @param s statement to be added
             */
            public void addStatement(Statement s) {
                        body.add(s);
            }


            /**
             * @return the primitives
             */
            public Vector<Type> getPrimitives() {
                        return primitives;
            }

            /**
             * Add Primitive Declarations (mars.Int or mars.Dbl) to the primitives Vector
```

```
             * @param type object to be added
             */
            public void addPrimitive(Type type) {
                        primitives.add(type);
            }
}
```

8.1.11 Dbl.java – MARS representation of a Double primitive

```
/**
 * Double Class for MARS for primitive type double
 * @author swapneelsheth, aaron
 */

package mars;

public class Dbl implements Type{
            private String name;
            private double value;

            /**
             * Constructor
             * @param name name of the double
             * @param value value of the double
             */
            public Dbl(String name, double value) {
                        this.name = name;
                        this.value = value;
            }

            /**
             * "Default" Constructor
             * @param name name of the Int
             */
            public Dbl(String name) {
                        this(name, 0);
            }

            /**
             * @return the value
             */
            public double getValue() {
                        return value;
            }

            /**
             * @param value the value to set
             */
            public void setValue(double value) {
                        this.value = value;
            }

            /**
             * @return the name
             */
            public String getName() {
                        return name;
            }
}
```

8.1.12 Int.java – MARS representation of an Integer primitive

```
/**
 * Int Class for Mars for primitive type int
 * @author swapneelsheth, aaron
 */

package mars;

public class Int implements Type{
            private String name;
            private int value;

            /**Constructor
             * @param name name of the integer
             * @param value value of the integer
             */
            public Int(String name, int value) {
                        this.name = name;
                        this.value = value;
            }

            /**
             * "Default" Constructor
             * @param name name of the Int
             */
            public Int(String name) {
                        this(name, 0);
            }

            /**
             * @return the value
             */
            public int getValue() {
                        return value;
            }

            /**
             * @param value the value to set
             */
            public void setValue(int value) {
                        this.value = value;
            }

            /**
             * @return the name
             */
            public String getName() {
                        return name;
            }
}
```

8.1.13 FunctionCall.java – Function class for MAR

```
/**
 * Function Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;
import java.util.*;

public class FunctionCall implements Statement{
            String name;
            String className;
            String classType;
            Vector <String> paramList;

            /**
             * Constructor
             * @param methodName name of the method called
             * @param className name of the Class
             */
            public FunctionCall(String methodName, String className, String classType) {
                        this.name = methodName;
                        this.className = className;
                        this.classType = classType;
                        paramList = new Vector<String>();
            }

            /**
             * "Default" Constructor
             * @param methodName name of the method called
             */
            public FunctionCall(String methodName) {
                        this(methodName, "", "");
            }

            /**
             * @return the className
             */
            public String getClassName() {
                        return className;
            }

            /**
             * @return the methodName
             */
            public String getName() {
                        return name;
            }

            /**
             * @return the paramList
             */
            public Vector<String> getParamList() {
                        return paramList;
            }

            /**
             * Add a Parameter List to the Function
             * @param params Adds the vector passed to paramList
             */
            public void addParam(Vector params) {
                        for (int i=0; i<params.size(); i = i+1) {
                                    String name = (String)params.get(i);
                                    paramList.add(name);
                        }
            }

            /**
             * Get the class type of the object on which the function was called.
             * @return the class type as a string
             */
            public String getClassType() {
                        return classType;
            }
}
```

8.1.14 Group.java – Representation of a MARS group

```
/**
 * Group Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

import java.util.*;

public class Group implements Type{
            private String name;
            private int volume;
            private Vector<Track> tracks;
            private Vector<Statement> body;
            private Vector<Type> primitives;

            /**
             * @param name name of the group
             */
            public Group(String name) {
                        this.name = name;
                        tracks = new Vector<Track>();
                        body = new Vector<Statement>();
                        primitives = new Vector<Type>();
            }

            /**
             * @return the volume
             */
            public int getVolume() {
                        return volume;
            }

            /**
             * @param volume the volume to set
             */
            public void setVolume(int volume) {
                        this.volume = volume;
            }

            /**
             * @return the name
             */
            public String getName() {
                        return name;
            }

            /**
             * @return the tracks
             */
            public Vector<Track> getTracks() {
                        return tracks;
            }

            /**
             * Add a Track to a Group
             * @param track name of the track to be added
             */
            public void addTrack(Track track) {
                        tracks.add(track);
            }

            /**
             * @return the body
             */
            public Vector<Statement> getBody() {
                        return body;
            }

            /**
             * Add Statements to the Vector of Statements (body)
             * @param s statement to be added
             */
            public void addStatement(Statement s) {
                        body.add(s);
            }

            /**
             * @return the primitives
```

```
	 */
	public Vector<Type> getPrimitives() {
		return primitives;
	}

	/**
	 * Add Primitive Declarations (mars.Int or mars.Dbl) to the primitives Vector
	 * @param type object to be added
	 */
	public void addPrimitive(Type type) {
		primitives.add(type);
	}
}
```

8.1.15 Track.java – Representation of a MARS track

```
/**
 * Track Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

public class Track implements Type{
            private String name;
            private String location;
            private int volume;

            /**
             * Constructor
             * @param name This is the name of the Track
             * @param location This is the location of the Track
             */
            public Track(String name, String location) {
                        this.name = name;
                        this.location = location;
            }

            /**
             * @return the volume
             */
            public int getVolume() {
                        return volume;
            }

            /**
             * @param volume the volume to set
             */
            public void setVolume(int volume) {
                        this.volume = volume;
            }

            /**
             * @return the location
             */
            public String getLocation() {
                        return location;
            }

            /**
             * @param location the location to set
             */
            public void setLocation(String location) {
                        this.location = location;
            }

            /**
             * @return the name
             */
            public String getName() {
                        return name;
            }
}
```

8.1.16 Section.java – Representation of a MARS section

```
/**
 * Section Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

import java.util.*;

public class Section implements Type{
            private String name;
            private Vector<Group> groups;
            private Vector<Track> tracks;
            private Vector<Statement> body;
            private Vector<Type> primitives;

            /**
             * @param name name of the section
             */
            public Section(String name) {
                        this.name = name;
                        groups = new Vector<Group>();
                        tracks = new Vector<Track>();
                        body = new Vector<Statement>();
                        primitives = new Vector<Type>();
            }

            /**
             * @return the name
             */
            public String getName() {
                        return name;
            }

            /**
             * @return the groups
             */
            public Vector<Group> getGroups() {
                        return groups;
            }

            /**
             * @return the tracks
             */
            public Vector<Track> getTracks() {
                        return tracks;
            }

            /**
             * Add a Group to a Section
             * @param group name of the group to be added
             */
            public void addGroup(Group group) {
                        groups.add(group);
            }

            /**
             * Add a Track to a Section
             * @param track name of the track to be added
             */
            public void addTrack(Track track) {
                        tracks.add(track);
            }

            /**
             * @return the body
             */
            public Vector<Statement> getBody() {
                        return body;
            }

            /**
             * Add Statements to the Vector of Statements (body)
             * @param s statement to be added
             */
            public void addStatement(Statement s) {
                        body.add(s);
            }
```

```
		/**
		 * @return the primitives
		 */
		public Vector<Type> getPrimitives() {
			return primitives;
		}

		/**
		 * Add Primitive Declarations (mars.Int or mars.Dbl) to the primitives Vector
		 * @param type object to be added
		 */
		public void addPrimitive(Type type) {
			primitives.add(type);
		}
}
```

8.1.17 If.java – Representation of a MARS "if" statement

```
/**
 * If Class for MARS
 * @author swapneelsheth, aaron
 */

package mars;

import java.util.Vector;

public class If implements Statement{
            private BoolExpr boolExpr;
            private Vector<Statement> thenPart;
            private Vector<Type> thenPrimitives;
            private Vector<Statement> elsePart;
            private Vector<Type> elsePrimitives;

            /**Constructor
             * @param boolExpr boolean expression for the conditional
             * @param thenPart the then part of the conditional
             * @param elsePart the else part of the conditional
             */
            public If(BoolExpr boolExpr, Vector<Statement> thenPart, Vector<Type> thenPrimitives,
Vector<Statement> elsePart, Vector<Type> elsePrimitives) {
                        this.boolExpr = boolExpr;
                        this.thenPart = thenPart;
                        this.thenPrimitives = thenPrimitives;
                        this.elsePart = elsePart;
                        this.elsePrimitives = elsePrimitives;
            }

            /**
             * @return the boolExpr
             */
            public BoolExpr getBoolExpr() {
                        return boolExpr;
            }

            /**
             * @return the elsePart
             */
            public Vector<Statement> getElsePart() {
                        return elsePart;
            }

            /**
             * @return the thenPart
             */
            public Vector<Statement> getThenPart() {
                        return thenPart;
            }

            /**
             * @return the elsePrimitives
             */
            public Vector<Type> getElsePrimitives() {
                        return elsePrimitives;
            }

            /**
             * @return the thenPrimitives
             */
            public Vector<Type> getThenPrimitives() {
                        return thenPrimitives;
            }
}
```

8.1.18 For.java – Representation of a MARS “for loop”

```
/**
 * Class for FOR loops
 * @author swapneelsheth, aaron
 */

package mars;

import java.util.Vector;

public class For implements Statement{
            private String variable;
            private int lowerBound;
            private int upperBound;
            private int step;
            private Vector<Statement> body;
            private Vector<Type> primitives;
            private boolean variableDefined = false;

            /**
             * Constructor
             * @param variable name of the variable used in the 'for' loop
             * @param lowerBound lower value of the iteration loop
             * @param upperBound higher value of the iteration loop
             */
            public For(String variable, int lowerBound, int upperBound, int step, Vector<Statement> body,
Vector<Type> primitives) {
                        this.variable = variable;
                        this.lowerBound = lowerBound;
                        this.upperBound = upperBound;
                        this.step = step;
                        this.body = body;
                        this.primitives = primitives;
            }

            /**
             * @return the lowerBound
             */
            public int getLowerBound() {
                        return lowerBound;
            }

            /**
             * @return the upperBound
             */
            public int getUpperBound() {
                        return upperBound;
            }

            /**
             * @return the variable
             */
            public String getVariable() {
                        return variable;
            }

            /**
             * @return the body
             */
            public Vector<Statement> getBody() {
                        return body;
            }

            /**
             * @return the step
             */
            public int getStep() {
                        return step;
            }

            /**
             * @return the primitives
             */
            public Vector<Type> getPrimitives() {
                        return primitives;
            }

            /**
             * @return the variableDefined
             */
```

```
	public boolean isVariableDefined() {
		return variableDefined;
	}

	/**
	 * @param variableDefined the variableDefined to set
	 */
	public void setVariableDefined(boolean variableDefined) {
		this.variableDefined = variableDefined;
	}
}
```

8.1.19 SystemCall.java – MARS representation of a system call

```
/**
 * Class for Built In Functions in MARS
 * @author swapneelsheth, aaron
 */

package mars;

import java.util.Vector;

public class SystemCall implements Type{
            private String name;
            private Vector<String> argList;
            private Type returnType;
            private Vector<String> calledBy;

            /**
             * Constructor
             * @param name
             * @param type
             */
            public SystemCall(String name, Type type) {
                        this.name = name;
                        returnType = type;
                        argList = new Vector<String>();
                        calledBy = new Vector<String>();
            }

            /**
             * @return the arglist
             */
            public Vector<String> getArgList() {
                        return argList;
            }

            /**
             * @return the name
             */
            public String getName() {
                        return name;
            }

            /**
             * Adds the arguments to the Vector
             * @param args Adds the vector passed to argList
             */
            public void addArg(Vector args) {
                        for (int i=0; i<args.size(); i = i+2) {
                                    String type = (String)args.get(i);
                                    String name = (String)args.get(i+1);
                                    argList.add(type);
                                    argList.add(name);
                        }
            }

            /**
             * @return the returnType
             */
            public Type getReturnType() {
                        return returnType;
            }

            /**
             * Add types that can call this SystemCall
             * @param type type that can call
             */
            public void addCalledBy(String type) {
                        calledBy.add(type);
            }

            /**
             * @return the calledBy
             */
            public Vector<String> getCalledBy() {
                        return calledBy;
            }
}
```

8.1.20 Statement.java – Interface for statements in MARS

```
/**
 * Null Interface for all Statements (Arithmetic, boolean, For, If) in MARS
 * @author swapneelsheth, aaron
 */

package mars;

public interface Statement{

}
```

8.1.21 Type.java – Interface for types in MARS

```
/**
 * Null Interface Type for all Types in MARS
 * @author swapneelsheth, aaron
 */

package mars;

public interface Type {

            public String getName();

}
```

8.2 backEnd package

The backEnd package contains things specific to the back-end of the compiler such as code generation.

8.2.1 CodeGenerator.java – Class that drives code generation

```
package backEnd;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Iterator;
import java.util.Vector;


import mars.*;

/**
 * Class that drives the code Generation
 * @author Aaron and Ritika
 */
public class CodeGenerator {

            static String newLine = System.getProperty("line.separator");
            static FileWriter fw=null;

            /**
             * Constructor
             * @param comp the composition to be generated.
             */
            public CodeGenerator(Composition comp, String fileName) {
                        try {
                                    /**
                                     * First, open the target file. Do the necessary imports.
                                     */
                                    String fileNameJava=fileName.substring(0,fileName.length()-4);
                                    fw = new FileWriter(new File(fileNameJava+"java"));
                                    writeLine("package Tests;");
                                    writeLine("import mars_util.*;");
                                    writeLine("public class "+comp.getName()+"{");

                                    Vector<mars.Track> compTracks=comp.getTracks();  //get all the Tracks
defined in the composition scope
                                    for(int i=0;i<compTracks.size();i++){
                                                writeLine("static Track "+compTracks.get(i).getName()+"=
new Track(\"file:/"+compTracks.get(i).getLocation()+"\");");
                                    }
                                    Vector<mars.Type> compPrimitives = comp.getPrimitives();    // get
all the primitives defined in composition

                                    for(int i=0;i<compPrimitives.size();i++){
                                                Type primitiveType = compPrimitives.get(i);
                                                if (primitiveType instanceof mars.Int) {
                                                            writeLine("static int " +
primitiveType.getName()+" = " + ((mars.Int)primitiveType).getValue() + ";" );
                                                }
                                                else if (primitiveType instanceof mars.Dbl) {
                                                            writeLine("static double " +
primitiveType.getName()+" = " + ((mars.Dbl)primitiveType).getValue() + ";" );
                                                }
                                    }

                                    Vector<mars.Section> compSections=comp.getSections(); //get all the
Sections defined in the composition scope

                                    Vector<String> playParams=null;
                                    Vector<mars.Statement> compBody= comp.getBody();
                                    for(int i=0;i<compBody.size();i++){
                                                mars.Statement stmt = compBody.get(i);

            if(stmt.getClass().getName().equals("mars.FunctionCall")){
                                                            mars.FunctionCall fc=
(mars.FunctionCall)stmt;
                                                            if(fc.getName().equals("playOrder")){
```

```
                                                playParams = fc.getParamList();
                                        }//end if(fc.equals("playOrder")){

                                        }//
if(stmt.getClass().getName().equals("mars.FunctionCall")){
                                }// for(int i=0;i<compBody.size();i++){

                                // Starting users psvm.

                                writeLine("public static void main(String[] args) {" );

                                Iterator<String> sectionNameItr=playParams.iterator();
                                while(sectionNameItr.hasNext()){
                                        String currentSectionName=sectionNameItr.next();
                                        writeLine(currentSectionName + "();" );
                                }
                                writeLine("}");//close PSVM

                                /*
                                 * For each section, define a new method.
                                 */

                                Iterator<mars.Section> sectionItr = compSections.iterator();

                                while(sectionItr.hasNext()){
                                        mars.Section currentSection = sectionItr.next();
                                        writeLine("static void " + currentSection.getName() + "()
{");

                                        backEnd.CodeGenSection aSection = new
backEnd.CodeGenSection(currentSection);
                                        aSection.expandYourself();

                                        writeLine("}");
                                }

                                writeLine("}");//close ClassBody
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                        finally{
                                try {
                                        fw.close();
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                }

                /**
                 * Writes a line to a file. Puts a newline after that.
                 * @param str the line to be written
                 * @throws IOException
                 */
                static void writeLine(String str) throws IOException {

                        fw.write(str + newLine);
                }

                /**
                 * Writes a line to a file. No newline at the end.
                 * @param str the line to be written
                 * @throws IOException
                 */
                static void write(String str) throws IOException {
                        fw.write(str);
                }

}
```

8.2.2 CodeGenBody.java – Class to generate the body of a composition

```
package backEnd;

import java.util.Vector;
import java.io.*;

import mars.Type;

/**
 * Class to Generate the body of a composition, group or section
 * @author Aaron and Ritika
 *
 */
public class CodeGenBody {

            /**
             * Expands the group
             * @param body the body to be expanded
             * @throws IOException
             */
            static void expand(Vector<mars.Statement> body) throws IOException
            {
                        for(int i=0;i<body.size();i++){
                                    mars.Statement stmt = body.get(i);
                                    System.out.println(stmt.getClass().getName());

                                    /**
                                     * Checking for various types of statements.
                                     */

                                    // If the statement is a function call.

                                    if(stmt.getClass().getName().equals("mars.FunctionCall")){
                                                mars.FunctionCall fc= (mars.FunctionCall)stmt;

                                                /*
                                                 * Now checking for the various function calls supported.
                                                 */

                                                // delay function

                                                if (fc.getName().equals("delay")) {
                                                            CodeGenerator.writeLine("try {");
                                                            CodeGenerator.writeLine("Thread.sleep((long)"
+ fc.getParamList().get(0) + ");");
                                                            CodeGenerator.writeLine("}");
                                                            CodeGenerator.writeLine("catch (Exception e)
{");

            CodeGenerator.writeLine("System.out.println(\"Error while delaying\");");
                                                            CodeGenerator.writeLine("}");
                                                } // end: if (fc.getName().equals("delay"))


                                                // play function

                                                else if(fc.getName().equals("play")){

                                                            // Is it a track that we have to play?

                                                            if (fc.getClassType().equals("mars.Track")) {

            CodeGenerator.write(fc.getClassName()+"."+fc.getName()+"(");

                                                                        if(fc.getParamList().size()!=0){
                                                                                    String param =
fc.getParamList().get(0);

                                                                                    if
(param.contains("::")) {

                                                                                                String
myString[] = param.split("::");

                                                                                                param =
myString[0] + "." + myString[1] + "()";

                                                                                    }

            CodeGenerator.write(param);

                                                                        }
                                                                        CodeGenerator.writeLine(");");
```

```
                                                        }// end: if
(fc.getClassType().equals("mars.Track"))

                                                        //           Is it a group that we have to
play?

                                                        else if
(fc.getClassType().equals("mars.Group")) {
                                                                backEnd.CodeGenGroup g = new
backEnd.CodeGenGroup(CodeGenSection.findMatchingGroup(fc.getClassName()));
                                                                g.expandYourself();
                                                        }
                                                }//end if(fc.getName().equals("play")){

                                                // loop function.

                                                else if(fc.getName().equals("loop")){

                                                        // Loop a track

                                                        if (fc.getClassType().equals("mars.Track")) {

            CodeGenerator.write(fc.getClassName()+"."+fc.getName()+"(");

                                                                if(fc.getParamList().size()!=0){
                                                                        String param =
fc.getParamList().get(0);

                                                                        if
(param.contains("::")) {
                                                                                String
myString[] = param.split("::");
                                                                                param =
myString[0] + "." + myString[1] + "()";

                                                                        }

            CodeGenerator.write(param);

                                                                }
                                                                CodeGenerator.writeLine(");");
                                                        }// end: if
(fc.getClassType().equals("mars.Track"))

                                                        //                       Loop a group

                                                        else if
(fc.getClassType().equals("mars.Group")) {
                                                                mars.Group mg =
CodeGenSection.findMatchingGroup(fc.getClassName());
                                                                backEnd.CodeGenGroup g = new
backEnd.CodeGenGroup(mg);
                                                                String ctr = mg.getName();
                                                                String param = null;

                                                                if(fc.getParamList().size()!=0){
                                                                        param =
fc.getParamList().get(0);

                                                                        if
(param.contains("::")) {
                                                                                String
myString[] = param.split("::");
                                                                                param =
myString[0] + "." + myString[1] + "()";

                                                                        }
                                                                }
                                                                CodeGenerator.writeLine("for (int
" + ctr + "=0;" + ctr + "<" + param + ";" + ctr + "++) {");
                                                                g.expandYourself();
                                                                CodeGenerator.writeLine("}");
                                                        }
                                                }//end if(fc.getName().equals("loop")){

                                                // fadeIn function.

                                                else if(fc.getName().equals("fadeIn")){

                                                        if (fc.getClassType().equals("mars.Track")) {

            CodeGenerator.write(fc.getClassName()+"."+fc.getName()+"(");

                                                                if(fc.getParamList().size()!=0){
                                                                        String param =
```

```
fc.getParamList().get(0);
                                                                        if
(param.contains("::")) {
                                                                            String
myString[] = param.split("::");
                                                                            param =
myString[0] + "." + myString[1] + "()";

                                                                        }
            CodeGenerator.write(param);

                                                            }
                                                            CodeGenerator.writeLine(");");
                                                    }// end: if
(fc.getClassType().equals("mars.Track"))
                                                }//end if(fc.getName().equals("fadeIn")){

                                                // FadeOut function.
                                                else if(fc.getName().equals("fadeOut")){
                                                    if (fc.getClassType().equals("mars.Track")) {
            CodeGenerator.writeLine(fc.getClassName()+"."+fc.getName()+"();");
                                                    }// end: if
(fc.getClassType().equals("mars.Track"))
                                                }//end if(fc.getName().equals("fadeOut")){
                                                // SetVolume function.
                                                else if(fc.getName().equals("setVolume")){
                                                    if (fc.getClassType().equals("mars.Track")) {
            CodeGenerator.write(fc.getClassName()+"."+fc.getName()+"(");
                                                            if(fc.getParamList().size()!=0){
            CodeGenerator.write(fc.getParamList().get(0));
                                                            }
                                                            CodeGenerator.writeLine(");");
                                                    }// end: if
(fc.getClassType().equals("mars.Track"))
                                                }//end if(fc.getName().equals("setVolume")){
                                                // getLength function.
                                                else if(fc.getName().equals("getLength")){
                                                    if (fc.getClassType().equals("mars.Track")) {
            CodeGenerator.writeLine(fc.getClassName()+"."+fc.getName()+"();");
                                                    }// end: if
(fc.getClassType().equals("mars.Track"))
                                                }//end if(fc.getName().equals("getLength")){
                                                // If the function is mix.
                                                else if (fc.getName().equals("mix")) {
                                                    if (fc.getParamList().size() != 0) {
                                                            Object[] params =
(fc.getParamList()).toArray();
                                                            String[] tracks = new
String[params.length];
                                                            double[] innerParam = new
double[params.length];

                                                            for (int j=0;j<params.length;j++)
{
                                                                String temp[] =
(params[j].toString()).split("::");
                                                                tracks[j] = temp[0];

                                                                if
(temp[1].contains(":")) {
                                                                    String
anotherTemp[] = temp[1].split(":");
            innerParam[j] = Integer.parseInt(anotherTemp[1]);
                                                                }// end: if
(temp[1].contains(":"))
```

```
                                                                else {
        innerParam[j] = -1;
                                                                }
                                                        }// end: for (int
i=0;i<params.length;i++)

                                                        new CodeGenMix(tracks,
innerParam);
                                                }// end: if (fc.getParamList().size() != 0)
                                        }//end: if (fc.getName().equals("mix"))

                                }// end: if(stmt.getClass().getName().equals("mars.FunctionCall"))
                                //          If the statement is a for loop.
                                else if(stmt.getClass().getName().equals("mars.For")){

                                        mars.For forLoop = (mars.For)stmt;
                                        String operator = null;
                                        int sign= 0;
                                        if (forLoop.getLowerBound() < forLoop.getUpperBound())
                                        {
                                                operator = "<=";
                                                sign = 1;
                                        }
                                        else {
                                                operator = ">=";
                                                sign = -1;
                                        }


                                        CodeGenerator.writeLine("for (int " +
forLoop.getVariable() + "=" + forLoop.getLowerBound() + ";" + forLoop.getVariable() + operator +
forLoop.getUpperBound() + ";" + forLoop.getVariable() + "+=" + (sign*forLoop.getStep() + ") {"));
                                        Vector<mars.Type> forLoopPrimitives =
forLoop.getPrimitives();// get all the primitives defined in for Loop

                                        for(int j=0;j<forLoopPrimitives.size();j++){
                                                Type primitiveType =
forLoopPrimitives.get(j);
                                                if (primitiveType instanceof mars.Int) {
                                                        CodeGenerator.writeLine("int " +
primitiveType.getName()+" = " + ((mars.Int)primitiveType).getValue() + ";" );
                                                }
                                                else if (primitiveType instanceof mars.Dbl) {
                                                        CodeGenerator.writeLine("double "
+ primitiveType.getName()+" = " + ((mars.Dbl)primitiveType).getValue() + ";" );
                                                }
                                        }
                                        CodeGenBody.expand(forLoop.getBody());
                                        CodeGenerator.writeLine("}");// closing the for loop
                                }// end: if(stmt.getClass().getName().equals("mars.For"))

                                // If the statement is an if.

                                else if(stmt.getClass().getName().equals("mars.If")){
                                        mars.If ifCondition = (mars.If)stmt;

                                        CodeGenerator.writeLine("if " +
ifCondition.getBoolExpr().getCondition() + " {");

                                        //Expanding then part.

                                        Vector<mars.Type> thenPrimitives =
ifCondition.getThenPrimitives();    // get all the primitives defined in for Loop

                                        for(int j=0;j<thenPrimitives.size();j++){
                                                Type primitiveType = thenPrimitives.get(j);
                                                if (primitiveType instanceof mars.Int) {
                                                        CodeGenerator.writeLine("int " +
primitiveType.getName()+" = " + ((mars.Int)primitiveType).getValue() + ";" );
                                                }
                                                else if (primitiveType instanceof mars.Dbl) {
                                                        CodeGenerator.writeLine("double "
+ primitiveType.getName()+" = " + ((mars.Dbl)primitiveType).getValue() + ";" );
                                                }
                                        }

                                        CodeGenBody.expand(ifCondition.getThenPart());

                                        CodeGenerator.writeLine("}");       // Closing then block

                                        //Expanding else block

                                        CodeGenerator.writeLine("else {");
```

```
                                                Vector<mars.Type> elsePrimitives =
ifCondition.getElsePrimitives();    // get all the primitives defined in for Loop

                                                for(int j=0;j<elsePrimitives.size();j++){
                                                            Type primitiveType = elsePrimitives.get(j);
                                                            if (primitiveType instanceof mars.Int) {
                                                                        CodeGenerator.writeLine("int " +
primitiveType.getName()+" = " + ((mars.Int)primitiveType).getValue() + ";" );
                                                            }
                                                            else if (primitiveType instanceof mars.Dbl) {
                                                                        CodeGenerator.writeLine("double "
+ primitiveType.getName()+" = " + ((mars.Dbl)primitiveType).getValue() + ";" );
                                                            }
                                                }

                                                CodeGenBody.expand(ifCondition.getElsePart());

                                                CodeGenerator.writeLine("}");       // Closing else block

                                    }// end: if(stmt.getClass().getName().equals("mars.If")){

                                    // If the statement is an assignment expression.

                                    else if(stmt.getClass().getName().equals("mars.AssignExpr")){
                                                mars.AssignExpr assEx = (mars.AssignExpr)stmt;

            CodeGenerator.writeLine(((mars.Type)assEx.getLValue()).getName() + " = " + assEx.getRValue()
+ ";");
                                    }
                        }// end: for(int i=0;i<theSectionBody.size();i++){
            }
}
```

8.2.3 CodeGenGroup.java – Class to expand a group definition

```
package backEnd;

import java.io.IOException;
import java.util.Vector;

import mars.Type;

/**
 * Expands a group definition.
 * @author Aaron and Ritika
 *
 */
class CodeGenGroup {


            private mars.Group theGroup;
            Vector<mars.Track> groupTracks;
            Vector<mars.Type> groupPrimitives;
            Vector<mars.Statement> groupStmts;

            CodeGenGroup(mars.Group theGroup){

                        this.theGroup=theGroup;
                        groupTracks=theGroup.getTracks();  //get all the Tracks defined in the group
scope
                        groupPrimitives = theGroup.getPrimitives(); // get all the primitives in the
group
                        groupStmts = theGroup.getBody();

            }

            /**
             * Expands a group.
             * @throws IOException
             */
            void expandYourself() throws IOException {

                        // Defining all tracks within the group
                        for(int i=0;i<groupTracks.size();i++){
                                    CodeGenerator.writeLine("Track "+groupTracks.get(i).getName()+"=new
Track(\"file:/"+groupTracks.get(i).getLocation()+"\");"+CodeGenerator.newLine);
                        }

                        // Defining all primitive types within the group.

                        for(int i=0;i<groupPrimitives.size();i++){
                                    Type primitiveType = groupPrimitives.get(i);
                                    if (primitiveType instanceof mars.Int) {
                                                CodeGenerator.writeLine("int " +
primitiveType.getName()+" = " + ((mars.Int)primitiveType).getValue() + ";" );
                                    }
                                    else if (primitiveType instanceof mars.Dbl) {
                                                CodeGenerator.writeLine("double " +
primitiveType.getName()+" = " + ((mars.Dbl)primitiveType).getValue() + ";" );
                                    }
                        }// end:for(int i=0;i<groupPrimitives.size();i++)

                        // Iterate through all the statements in the group body.

                        Vector<mars.Statement> theGroupBody= theGroup.getBody();
                        CodeGenBody.expand(theGroupBody);
            }// endFunc: void expandYourself() throws IOException
}
```

8.2.4 CodeGenSection.java – Class to expand a section in a composition

```
package backEnd;

import java.io.IOException;
import java.util.Iterator;
import java.util.Vector;

import mars.Type;

/**
 * Expands each section in a composition
 * @author Aaron and Ritika
 *
 */
public class CodeGenSection {


            private mars.Section theSection;
            Vector<mars.Track> sectionTracks;
            static Vector<mars.Group> sectionGroups;
            Vector<mars.Type> sectionPrimitives;
            Vector<mars.Statement> sectionStmts;

            CodeGenSection(mars.Section theSection){

                        this.theSection=theSection;
                        sectionTracks=theSection.getTracks();  //get all the Tracks defined in the
section scope
                        sectionGroups=theSection.getGroups(); //get all the Groups defined in the section
scope
                        sectionPrimitives = theSection.getPrimitives(); //get all the primitives in the
section
                        sectionStmts = theSection.getBody();

            }

            /**
             * Code that expands the section
             * @throws IOException
             */
            void expandYourself() throws IOException {

                        // Defining all tracks within the section
                        for(int i=0;i<sectionTracks.size();i++){
                                    CodeGenerator.writeLine("Track "+sectionTracks.get(i).getName()+"=
new Track(\"file:/"+sectionTracks.get(i).getLocation()+"\");"+CodeGenerator.newLine);
                        }

                        // Defining all primitive types within the section.

                        for(int i=0;i<sectionPrimitives.size();i++){
                                    Type primitiveType = sectionPrimitives.get(i);
                                    if (primitiveType instanceof mars.Int) {
                                                CodeGenerator.writeLine("int " +
primitiveType.getName()+" = " + ((mars.Int)primitiveType).getValue() + ";" );
                                    }
                                    else if (primitiveType instanceof mars.Dbl) {
                                                CodeGenerator.writeLine("double " +
primitiveType.getName()+" = " + ((mars.Dbl)primitiveType).getValue() + ";" );
                                    }
                        }// end:for(int i=0;i<sectionPrimitives.size();i++)

                        // Iterate through all the statements in the section body.

                        Vector<mars.Statement> theSectionBody= theSection.getBody();
                        CodeGenBody.expand(theSectionBody);
            }// endFunc: void expandYourself() throws IOException

            /**
             * finds a matching 'group' given the group name
             * @param currentGroupName the name of the group
             * @return the matching 'group'
             */
            static mars.Group findMatchingGroup(String currentGroupName) {
                        int index=-1;
                        Iterator<mars.Group> groupItr=sectionGroups.iterator();
                        while(groupItr.hasNext()){ //assigns the default playOrder of sections
                                    index++;
                                    if(groupItr.next().getName().equals(currentGroupName)){
```

```
                                        return sectionGroups.get(index);
                                }
                        }
                        return null;
                }
        }
```

8.2.5 CodeGenMix.java – Class that generates mixing tracks

```
package backEnd;

import java.io.*;

/**
 * Class that handles mixing tracks.
 * @author Aaron and Ritika
 *
 */
class CodeGenMix {

            CodeGenMix(String[] tracks,double[] innerParams) throws IOException {
                        CodeGenerator.writeLine("{");
                        CodeGenerator.writeLine("Track[] tracksToMix=new Track["+ tracks.length +"];");
                        CodeGenerator.writeLine("double[] innerParams=new double["+ tracks.length +"];");

                        for (int i=0;i<tracks.length;i++) {
                                    CodeGenerator.writeLine("tracksToMix["+i+"] = "+tracks[i]+";");
                                    CodeGenerator.writeLine("innerParams["+i+"] = "+innerParams[i]+";");
                        }//end: for (int i=0;i<tracks.length;i++)

                        CodeGenerator.writeLine("Track.mix(tracksToMix,innerParams);");
                        CodeGenerator.writeLine("}");


            }
}
```

8.3 mars_util package

The mars_util package is the sound API that powers the MARS language. It must be included with when you compile the Java representation of your MARS composition.

8.3.1 MixerInformation.java – Provides the software mixer information

```
package mars_util;


import javax.sound.sampled.*;
import java.net.URL;

/**
 * Provides the software mixer information
 * @author Ritika
 *
 */
public class MixerInformation {


            private static SourceDataLine line = null;

            public MixerInformation(){

                        try {

                                    String fnm = "file:/sound_engine.wav";


                                    AudioInputStream stream = AudioSystem.getAudioInputStream( new
URL(fnm) );
                                    AudioFormat format = stream.getFormat( );



                                    // gather information for line creation
                        DataLine.Info info = new DataLine.Info(SourceDataLine.class, format);
                        if (!AudioSystem.isLineSupported(info)) {
                          System.out.println("Line does not support: " + format);
                          System.exit(0);
                        }

                        // get a line of the required format
                        line = (SourceDataLine) AudioSystem.getLine(info);
                        line.open(format);
                        Control[] supportedControls = line.getControls();
                        for (int count=0; count<supportedControls.length; count++){
                        System.out.println("supported controls"+ supportedControls[count]);
                        }

                        Line[] lines = {line, line};


                                    Mixer.Info[]aMixerInfos = AudioSystem.getMixerInfo();
                                    System.out.println("no of mixers"+ aMixerInfos.length);
                                    Mixer m;
                                    for (int mixercount=0; mixercount<aMixerInfos.length;mixercount++){
                                                m = AudioSystem.getMixer(aMixerInfos[mixercount]);
                                                System.out.println("Mixer Info:" + " Name:" +
aMixerInfos[mixercount].getDescription() + " Vendor:" + aMixerInfos[mixercount].getVendor());

                                                System.out.println("Sync supported by"+ mixercount +
"==="+ m.isSynchronizationSupported(lines, false) );
                                                System.out.println("max lines on this mixer
are"+m.getMaxLines(info));
                                                System.out.println("----------");
                                    }

                        }
                        catch (Exception e)
                        {  System.out.println( e.getMessage( ));
```

```
                System.exit(0);
            }

        }


        public static void main(String[] args) {
                new MixerInformation();
        }
}
```

8.3.2 Track.java – Provides all the functionality for a track of music

```
package mars_util;

import java.net.URL;
import java.util.Timer;
import java.util.TimerTask;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.FloatControl;
import javax.sound.sampled.LineEvent;
import javax.sound.sampled.LineListener;
import javax.sound.sampled.UnsupportedAudioFileException;

/**
 * Defines a track and all that a track can do
 * @author Aaron and Ritika
 *
 */
public class Track implements LineListener{

	Clip trk;
	long position=0;
	boolean stillPlaying=false;

	/**
	 * Constructor
	 * @param clipName the 'filename' of the track
	 */
	public Track(String clipName){
		try {

			/*
			 * Get a line to which the track can be attached and controlled.
			 */
			AudioInputStream ais = AudioSystem.getAudioInputStream(new
URL(clipName));
			AudioFormat fmt = ais.getFormat();
			DataLine.Info info = new DataLine.Info(Clip.class, fmt);

			if (!AudioSystem.isLineSupported(info)) {
				System.out.println("Unsupported format " + fmt);
				System.exit(0);
			}

			/*
			 * Assign the 'clip' to the line obtained.
			 */
			trk = (Clip) AudioSystem.getLine(info);
			trk.addLineListener(this);
			trk.open(ais);
			ais.close();
			this.position=0;
			this.stillPlaying=false;
			long durn = trk.getMicrosecondLength()/1000;

			if (durn <= 1000.00) {
				throw new UnsupportedAudioFileException("Too small a
file");
			}


		}
		catch(Exception e) {
			e.printStackTrace();
			System.exit(0);
		}

	}

	/**
	 * This method is called when the 'line' on which the track is playing is updated
	 * eg: during start, stop, etc.
	 */
	public void update(LineEvent le) {
		if (le.getType() == LineEvent.Type.START) {
			stillPlaying=true;
```

```
                }
                if (le.getType() == LineEvent.Type.STOP) {
                        stillPlaying=false;
                }

        }

        /**
         * Plays the track.
         */
        public void play() {
                if (trk != null) {
                        trk.setMicrosecondPosition(position*1000);
                        trk.start();
                        long durn = trk.getMicrosecondLength()/1000;
                        try {

                                Thread.sleep(durn-position);
                                while(stillPlaying){

                                }
                        }
                        catch(InterruptedException ie) {
                                ie.printStackTrace();
                        }
                        position=0;
                        trk.setMicrosecondPosition(0);
                }
        }


        /**
         * play the track for a given duration of time.
         * @param dbldurn the duration for which the track must play
         */
        public void play(double dbldurn) {
                if (trk != null) {
                        trk.setMicrosecondPosition(position*1000);
                        trk.start();
                        long durn = (long)dbldurn;
                        try {

                                Thread.sleep(durn-position);
                                trk.stop();
                                while(stillPlaying){

                                }
                        }
                        catch(InterruptedException ie) {
                                ie.printStackTrace();
                        }
                        position+=durn;
                        trk.setMicrosecondPosition(position*1000);
                        if(position>=trk.getMicrosecondLength()/1000){
                                position=0;
                                trk.setMicrosecondPosition(0);
                        }
                }
        }

        /**
         * Loops a track
         * @param loop_number the number of times the track must loop.
         */
        public void loop(int loop_number) {
                if (trk != null) {
                        for(int i=0;i<loop_number;i++){
                                this.play();
                        }
                }

        }


        /**
         * To set the volume of the track.
         * @param volume the volume to be set.
         */
        public void setVolume(double volume)
        {
                if ((trk != null) ) {
                        if (trk.isControlSupported(FloatControl.Type.MASTER_GAIN)) {
                                FloatControl gainControl = (FloatControl)
trk.getControl(FloatControl.Type.MASTER_GAIN);
```

```
                                                volume=volume*9*0.1+1;
                                                volume=Math.log10(volume);
                                                double range = gainControl.getMaximum() -
gainControl.getMinimum();
                                                double gain = (range * volume) +
gainControl.getMinimum();
                                                gainControl.setValue((float)gain );
                                        }
                                        else
                                                System.out.println("No Volume controls available");
                                }
                        }

                        /**
                         * Fades a track in.
                         * @param dbl_no_of_millis the time over which the track must be faded in.
                         */
                        public void fadeIn(double dbl_no_of_millis) {
                                if (trk != null) {
                                        long no_of_millis=(long)dbl_no_of_millis;
                                        trk.setMicrosecondPosition(position*1000);
                                        try {

                                                int step=250;// lets fadeout with a step of  0.25seconds

                                                FloatControl gainControl =
(FloatControl)trk.getControl(FloatControl.Type.MASTER_GAIN);

                                                int noOfStepsNeeded= (int)no_of_millis/step; //calculate
the number of steps you will need to fadeout

                                                float gain=gainControl.getMinimum(); //start track with
minimum gain
                                                gainControl.setValue(gain);//this is for volume zero at
start of track
                                                float gainBase=gain;
                                                trk.start();
                                                float gainIncr;
                                                for (int count=1; count<=noOfStepsNeeded; count++){
                                                        gainIncr=(float) ((-gainBase)
*Math.log10((((float)count*9)/noOfStepsNeeded)+1));
                                                        gain =gainBase+gainIncr; //increment at each
step
                                                        if(gain>gainControl.getMaximum()){
                                                                gain=gainControl.getMaximum();
                                                        }
                                                        if(gain<gainControl.getMinimum()){
                                                                gain=gainControl.getMinimum();
                                                        }
                                                        gainControl.setValue(gain);

                                                        Thread.sleep(step); //let thread sleep for
250 milliseconds = step


                                                }
                                                trk.stop();
                                                while(stillPlaying){

                                                }
                                        }
                                        catch(InterruptedException ie) {
                                                ie.printStackTrace();
                                        }
                                        position =no_of_millis;
                                        trk.setMicrosecondPosition(position*1000);
                                        if(no_of_millis >=trk.getMicrosecondLength()){
                                                position=0;
                                                trk.setMicrosecondPosition(0);
                                        }
                                }
                        }

                        /**
                         * Fades a track out.
                         *
                         */
                        public void fadeOut() {
                                if (trk != null) {                              // Try moving this somewhere
earlier in the control flow
                                        trk.setMicrosecondPosition(position*1000);
                                        long no_of_millis=trk.getMicrosecondLength()/1000-position; //number
of milliseconds user wants to fade out
```

```
                                    trk.start();
                                    try {

                                                int step=250; //we want to fade out in steps of 250
milliseconds

                                                FloatControl gainControl =
(FloatControl)trk.getControl(FloatControl.Type.MASTER_GAIN);//get the control for gain

                                                long noOfStepsNeeded= no_of_millis/step;  //calculate
number of steps needed to fadeout
                                                float gain=gainControl.getValue();
                                                float gainBase=gain;
                                                float gainDecr;//=range/noOfStepsNeeded;//calculate the
step decrement value for the gain
                                                for (long count=noOfStepsNeeded; count>0; count--){
                                                            gainDecr=(-gainControl.getMinimum())+(float)
((gainControl.getMinimum())*Math.log10((((float)count*9)/noOfStepsNeeded)+1));
                                                            gain=gainBase-gainDecr;
                                                            if(gain>gainControl.getMaximum()){
                                                                        gain=gainControl.getMaximum();
                                                            }
                                                            if(gain<gainControl.getMinimum()){
                                                                        gain=gainControl.getMinimum();
                                                            }
                                                            gainControl.setValue(gain); //set the gain
                                                            Thread.sleep(step); //put to sleep the thread
for the duration of the step
                                                }
                                                trk.stop();
                                                while(stillPlaying){
                                                            /* Do nothing while atleast one track is
playing */
                                                }
                                    }
                                    catch(InterruptedException ie) {
                                                ie.printStackTrace();
                                    }
                                    position=0;
                                    trk.setMicrosecondPosition(0);
                        }
            }

            public double getLength(){
                        if(trk!=null){
                                    return trk.getMicrosecondLength()/1000;
                        }
                        return -1;
            }

            /**
             * Used to start a particular track.
             *
             */
            public void start() {
                        if (trk != null) {
                                    trk.start();
                        }
            }


            /**
             * Mixes a given set of tracks.
             * @param tracks the array of tracks to be mixed
             * @param innerParams how long to play each track
             */
            public static void mix(Track[] tracks, double[] innerParams){
                        boolean paramFound=false;            // has the user limited the playtime of a
track?
                        int timerCtr = 0;
                        for (int i=0;i<innerParams.length ;i++){
                                    if(innerParams[i]!=-1){
                                                paramFound=true;
                                                timerCtr++;
                                    }
                        }


                        if(paramFound){        // Set timers for the tracks that have to play for a
limited time
                                    Timer[] alarms = new Timer[timerCtr];

                                    class StopTrack extends TimerTask {
                                                Track t;
```

```
                                        StopTrack(Track t) {
                                                this.t = t;
                                        }
                                        public void run() {
                                                t.stop();
                                                this.cancel();
                                        }
                                }


                                for (int i=0,j=0;i< tracks.length;i++){
                                        tracks[i].setPosition(tracks[i].position);
                                        tracks[i].start();
                                        if (innerParams[i] != -1) {
                                                alarms[j] = new Timer();
                                                alarms[j].schedule(new StopTrack(tracks[i]),
(long)innerParams[i]);
                                                j++;
                                        }

                                }

                                double[] durations=new double[innerParams.length];

                                // Find the duration of all tracks to be played.

                                for (int i=0;i<tracks.length;i++){
                                        if (innerParams[i] == -1) {
                                                durations[i]=tracks[i].getLength();
                                        }
                                        else{
                                                durations[i]=innerParams[i];
                                        }

                                }
                                //find the max duration to sleep the thread for

                                double maxDuration=Double.MIN_VALUE;

                                for (int i=0;i<tracks.length;i++){
                                        maxDuration = Math.max(maxDuration, durations[i]);
                                }
                                try {
                                        Thread.sleep((long)maxDuration);

                                        boolean done = false;

                                        while(!done){
                                                /*Do nothing while atleast one track is still
playing*/
                                                for (int j=0;j<tracks.length;j++) {
                                                        if (tracks[j].stillPlaying ==
true) {
                                                                done = false;
                                                                break;
                                                        }
                                                        else
                                                                done = true;
                                                }//end:for (int j=0;j<tracks.length;j++)
                                        }
                                }
                                catch(InterruptedException ie) {
                                        ie.printStackTrace();
                                }

                                // Stop all timers.

                                for (int i=0;i< alarms.length;i++){
                                        if(alarms[i]!=null){
                                                alarms[i].cancel();
                                        }

                                }

                                //set the positions

                                for (int i=0;i< tracks.length;i++){
                                        if(innerParams[i]==-1){
                                                // For tracks that have played completely,
reset their positions.
                                                tracks[i].position=0;
                                        }
                                        else{       // Set the new position of the track.
                                                tracks[i].position+=innerParams[i];
```

```
                                                            if(tracks[i].position
>(long)tracks[i].getLength()){
                                                                        tracks[i].position=0;
                                                            }
                                                }
                                    }



                        }
                        else{    // If all the tracks to be mixed have to be played for their entire
duration
                                    //    there is no need for any timer.

                                    double maxDuration=Double.MIN_VALUE ;

                                    for (int i=0;i<tracks.length;i++){
                                                maxDuration=Math.max(maxDuration,tracks[i].getLength());
                                    }


                                    for (int i=0;i< tracks.length;i++){
                                                tracks[i].setPosition(tracks[i].position);
                                                tracks[i].start();
                                    }

                                    //    pause for the duration of the longest track.

                                    try{
                                                Thread.sleep((long)maxDuration);

                                                //you dont need to stop the tracks since the lineevent
STOP  is called on them


                                                boolean done = false;


                                                while(!done){
                                                            /*Do nothing while atleast one track is still
playing*/
                                                            for (int j=0;j<tracks.length;j++) {
                                                                        if (tracks[j].stillPlaying ==
true) {
                                                                                    done = false;
                                                                                    break;
                                                                        }
                                                                        else
                                                                                    done = true;
                                                            }//end:for (int j=0;j<tracks.length;j++)
                                                }
                                    }
                                    catch(InterruptedException e){
                                                e.printStackTrace();
                                    }

                                    //wrap the tracks around

                                    for (int i=0;i<tracks.length;i++){
                                                tracks[i].position=0;
                                    }
                        }

            }

            /**
             * Set the frame position of the track.
             * @param position the position in milliseconds.
             */
            void setPosition(long position) {
                        trk.setMicrosecondPosition(position*1000);

            }

            /**
             * Stops the track.
             *
             */
            void stop() {
                        trk.stop();
            }
}
```