Software Design Specification for MusicMake

The Software Design Specification Outline

1. Introduction

1.1 Purpose of this document

Comprehensive and detailed description of the design and specifications of the MusicMaker Software.

1.2 Scope of the development project

This Music Maker software will consist of 5 major functions. Playing music based on the instrument of choice and the notes that are played, record the notes played by the user, playback the tune, save the recorded tune into a text file and load a saved tune.

1.3 Definitions, acronyms, and abbreviations

The vocabulary mainly used for this software mainly includes musical vocabulary. They are tempo-indicates speed of music, voice- Channel and Instrument- the type of instrument.

1.4 References

This document refers many documents and web pages which has been used throughout this software project in it's implementation. The implementation of the Jfugue package has been referred from the documents SunJava Jfugue and Jfugue.org/book. Online reference sources have been used for the UI implementation. The main source has been stackoverflow.com and Java documentation on Jframe, and Swing package.This document refers many documents and web pages which has been used throughout this software project in it's implementation. The implementation of the Jfugue package has been referred from the documents SunJava Jfugue and Jfugue.org/book. Online reference sources have been used for the UI implementation. The main source has been stackoverflow.com and Java documentation on Jframe, and Swing package.

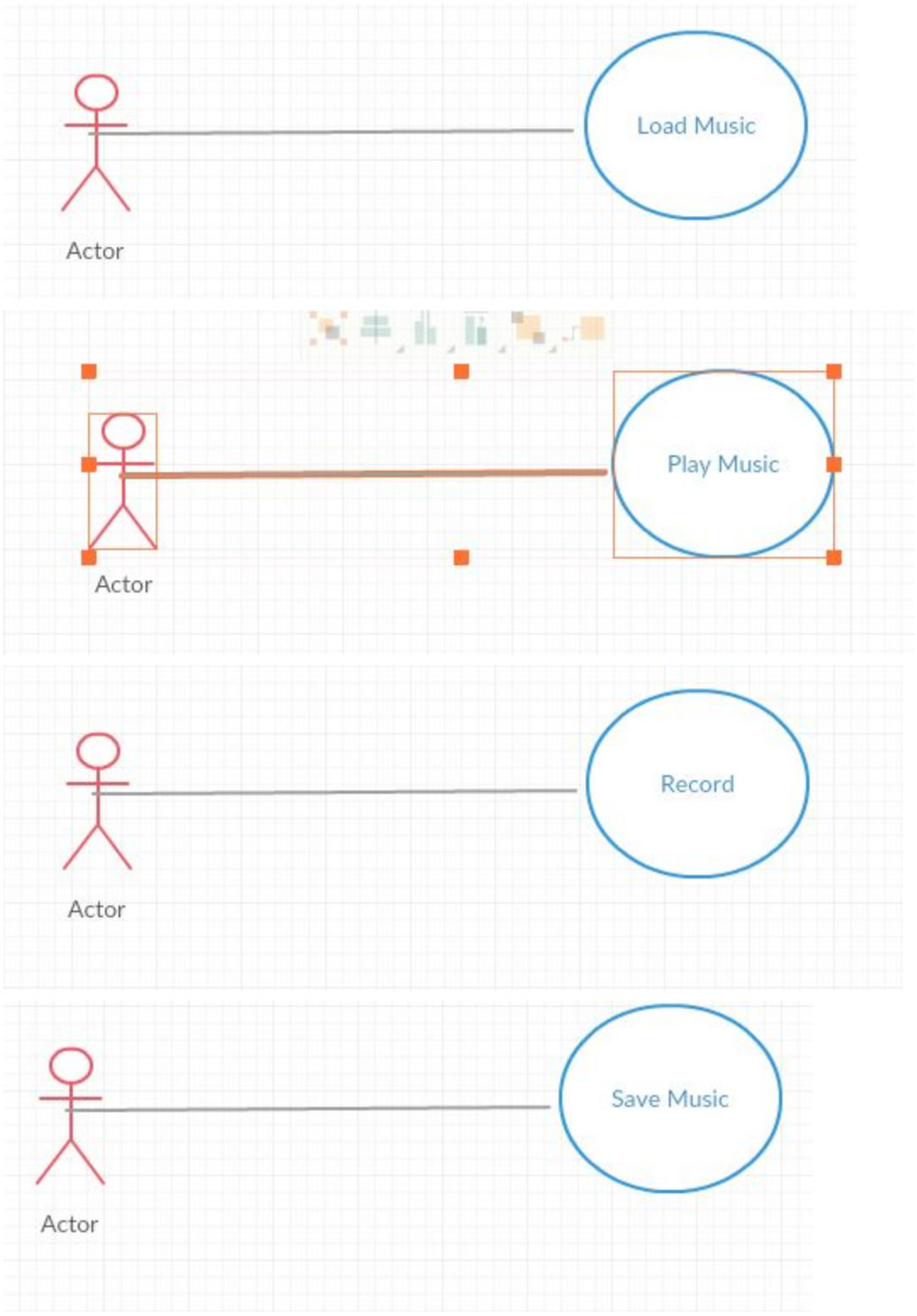
1.5 Overview of document

This first section is mainly an introduction, next we will look further into the system architecture, the detailed description of components, reuse and the relationship to other products, design decision and tradeoffs, the pseudocode for various components and finally the appendix.

2. System architecture description

2.1 Overview of modules / components

The User interface which provides 2 frames, the introduction frame and the actual instrument playing frame. The other components would be the actual classes which are a main class, instrument class and the actual instrument classes such as guitar, flute etc.

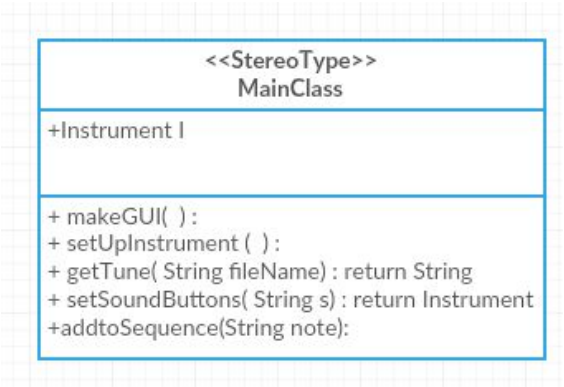


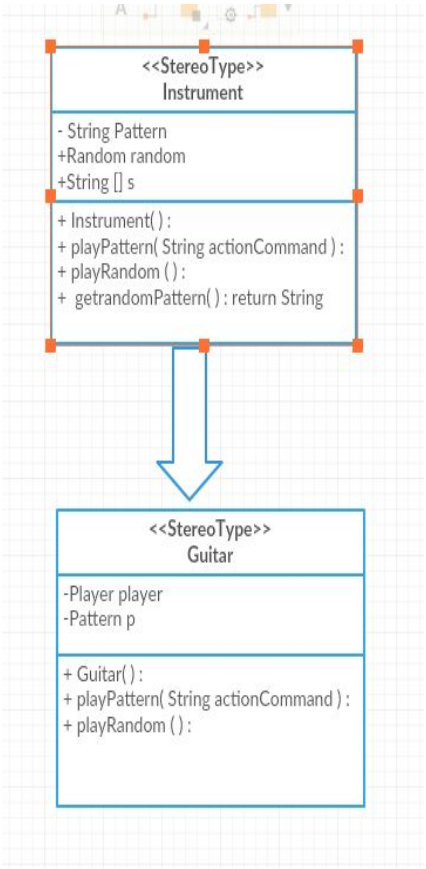
Various Use Cases for this Software

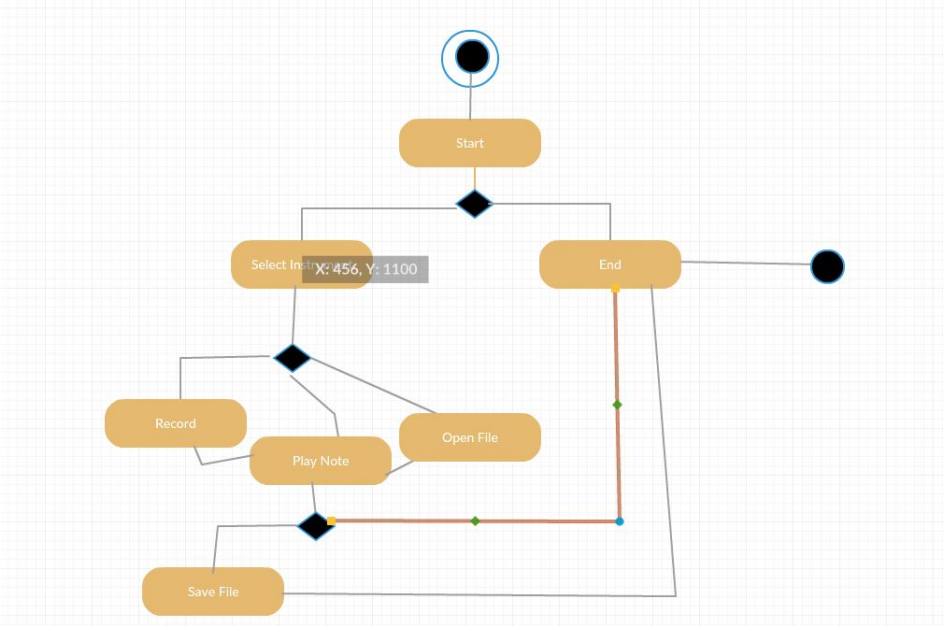
2.2 Structure and relationships

There is a main instrument class which is inherited by all the other specific instruments such as guitar, piano etc.. From the main class calls are made to create instrument objects which are used.

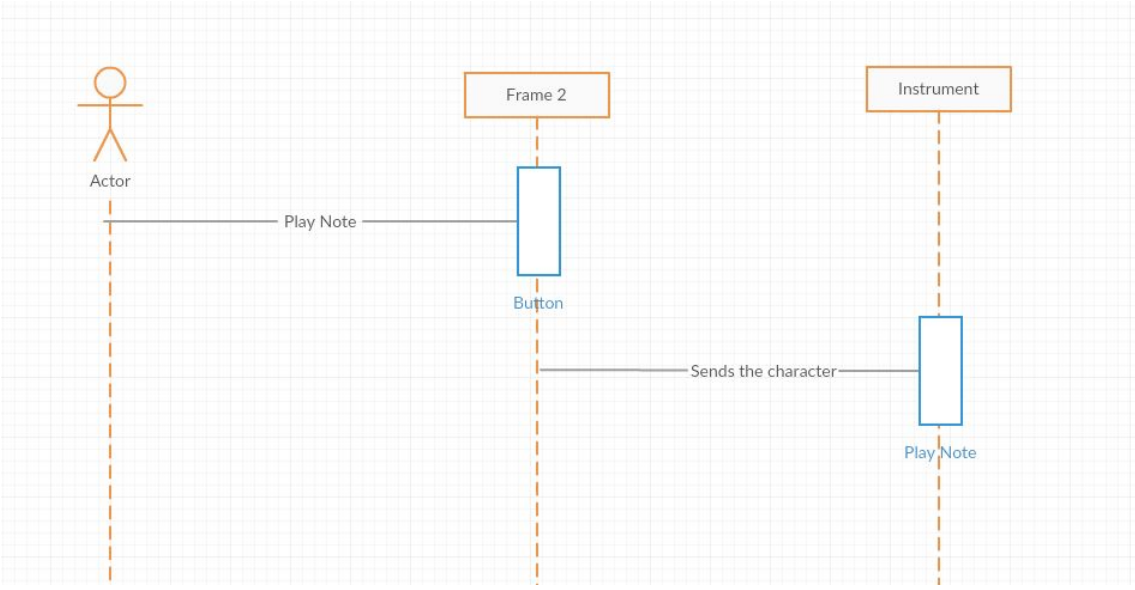
Then there is the main class that creates the 2 frames and also creates the instruments such that when the user selects the instrument and plays the note and also records it if the user chooses so.



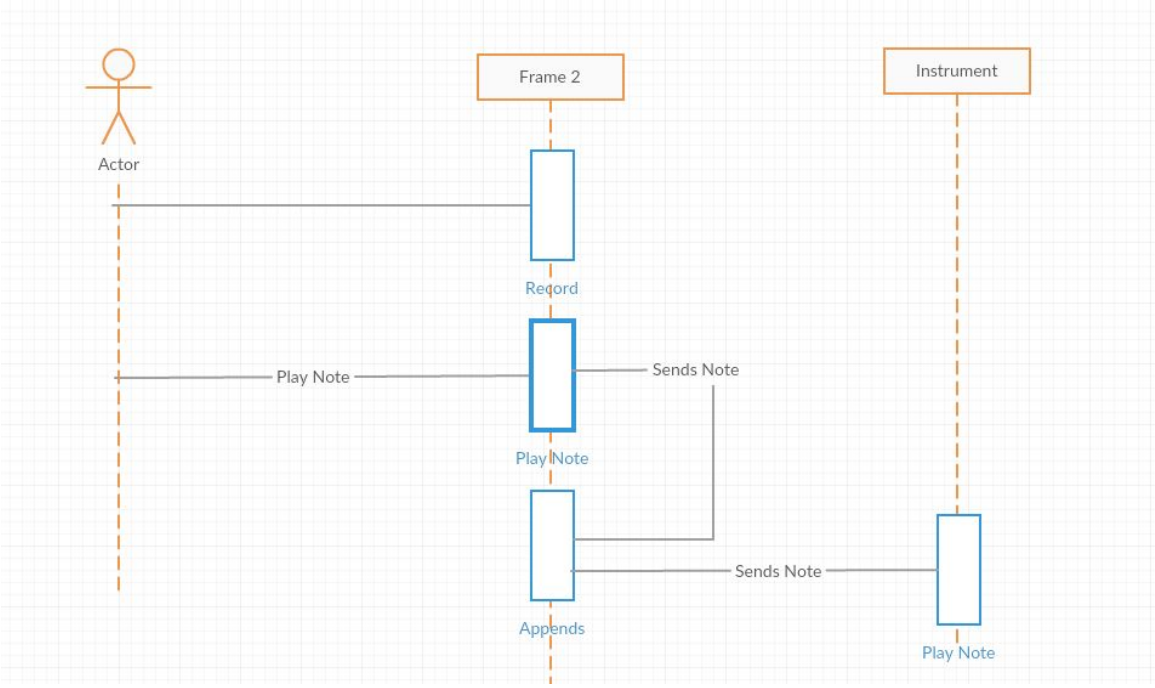




Activity Flow for the Software



Sequence Diagram for Play Music



Sequence Diagram for Record Music

2.3 User interface issues

The size of the buttons may not be according to the user's preference. The user does not know the tune that he has played so far in the recorded sequence. The selection of the instruments are in the middle instead of the leftmost corner.

3. Detailed description of components

3.1 Component template description

The following will give a description of each Class in this software system.

3.2 MainClass

Use exactly the template you define in 3.2. If a part of the template is not applicable, then mark it N/A rather than omitting it. Has the functions
addtoSequence - adds the tune selected by the user to the total recorded sequence
getTune - gets the tune to play from the selected file.
makeGUI - opens up the first frame which starts the application.
setSoundButtons - this function will tune the sound buttons to the selected instrument.
setUpInstrument - launches the 2nd frame where the user will have many options to choose from including the instrument of choice, record, play recording, load and save a tune.

3.3 Instrument Class

This class is the parent class to all the instrument classes that have been build in this software. It's functions
Instrument()- Constructor
playRandom()- generates a random pattern from a list of patterns
getrandomPattern()- returns a pattern generated from the playRandom()
playPattern()- does nothing

3.4 The actual instruments(Piano,Guitar,Harmonica,Violin,Flute,Gunshot, Bird_Tweet,Clarinet,Sitar,Trumpet,Tuba,Whistle)

These classes have the same structure as follows
Constructor() - generates the object.
playPattern()- plays the pattern that has been passed to it.
playRandom()- calls the parent playRandom() and plays that tune.

4.0 Reuse and relationships to other products

The main reuse of code is seen in between the classes itself as the different instrument classes. The base code is similar except for the different constructor calls. No external software has been used for reuse of the code.

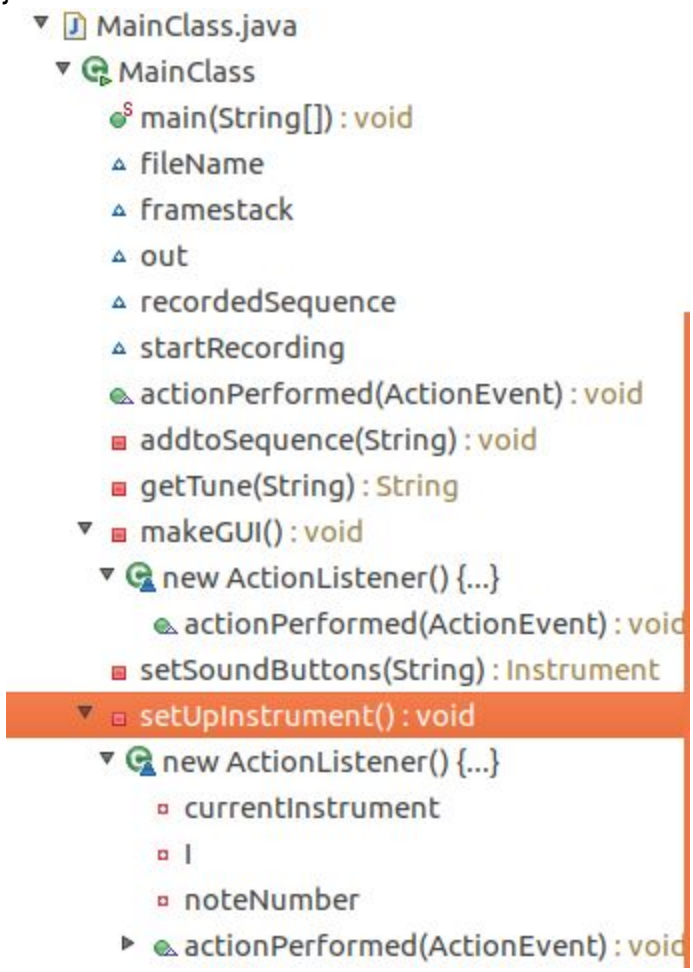
5.0 Design decisions and tradeoffs

JFrames are very limited in the way that they can be designed and hence this software had to compromise for not such an intuitive UI for better and easier code. Also incorporation of more functionality such as adding numerous tunes and also increasing the complexity of each tune, because of the time constraints.

6.0 Pseudo Code for components

MainClass-

makeGUI(){
Create the intro frame;
call setupInstrument()
}
setUpInstrument(){
Set up the UI for the second frame.
Take user i/p on the kind of instrument that they want and setup the instrument.
Assign buttons to perform actions such as play the note and save, record, replay etc.
}



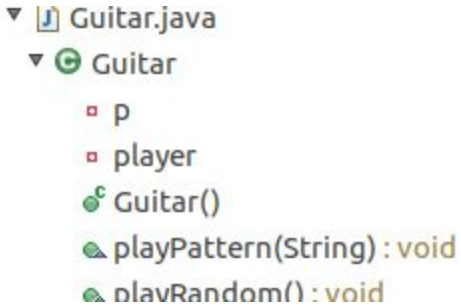
Instrument Class:-

Constructor call(){}
playPatter(){ Play pattern that was passed onto the function}

playRandomPattern(){Play a random pattern that is passed onto the the parent}

Genetic Instrument Class:-

Constructo(){}
PlayPattern(){ Get the button that the user pressed and }
PlayRandom(){Call the parent random function}



7.0 Appendices (if any)

“Not Applicable”