Design and Implementation of Java Music Player

ENGI 9874 Project

Zhenxin Zhao (201388915)

Haoyu Wang (201369592)

Table of contents

1. Design Target

The progress of the speed of today's desktop PCs coupled with breakthroughs in Java Virtual Machine (VM) optimization technology opens the door to Java-based, real-time processing, which is required by digital music players. Our project is the design and construction of a digital music player based on Java Media Framework.

2. Background

2.1 JMF

Java Media Framework (JMF) is an application programming interface (API) for incorporating time-based media into Java applications and applets. JMF contains classes that provide support for real-time Transport Protocol (RTP). RTP enables the transmission and reception of real-time media streams across the network. It also can be used for media-on-demand applications as well as interactive services, such as Internet telephony. [1]

2.2 Player

Player is a Media Handler for rendering and controlling time based media data. Player extends the Controller interface. Player provides methods for obtaining AWT components, media processing controls, and ways to manage other Controllers.

The life cycle of the player consists of three states. UnRealized state: Player has no knowledge of the media it is supposed to handle. Realizing state: Player identifies and acquires the resources it needs to playback the media. Prefetching state: Player perfected the media data. [1]

2.3 Manager

JMF uses intermediary objects called managers to integrate new implementations of key interfaces that define the behavior and interaction of objects used to capture, process, and present time-based media.

There are four kinds managers. Manager: handles the construction of Players, Processors, Data Sources, and Data Sinks. Package Manager: maintains a registry of packages that contain JMF classes, such as custom Players, Processors, Data Sources, and Data Sinks. Capture Device Manager: maintains a registry of available capture devices. Plug in Manager: maintains a registry of available JMF plug-in processing components, such as Multiplexers, Demultiplexers, Codes, Effects, and Renders. [1]

3. Design Specification

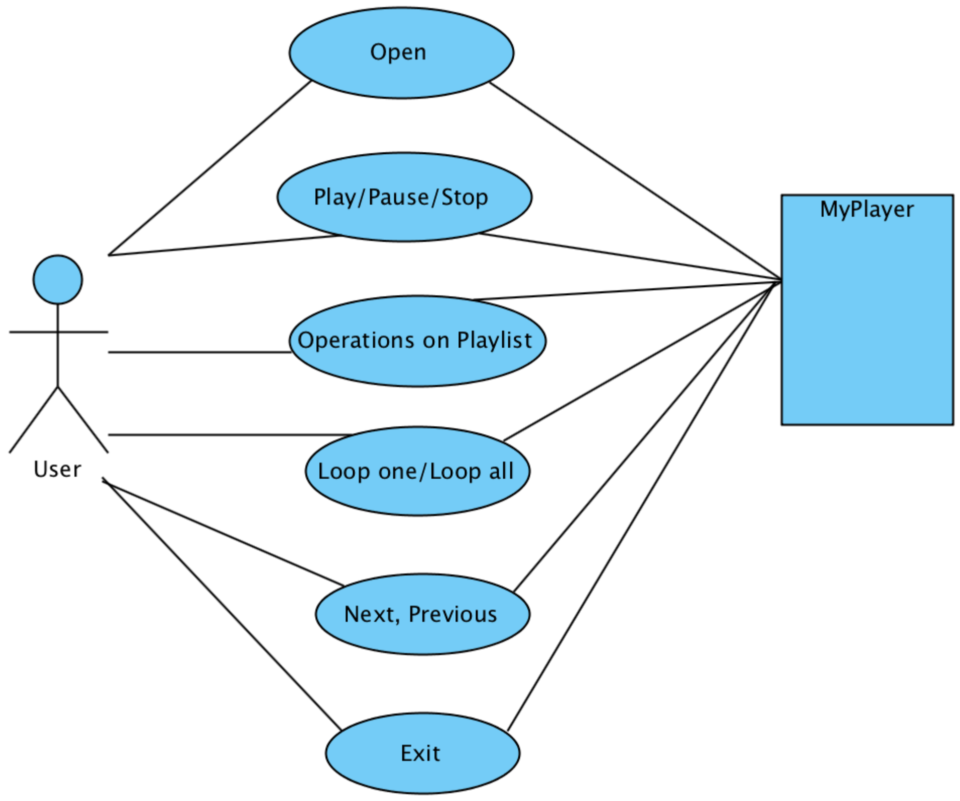


Figure 1. Use case diagram

The functionalities that our music player supposed to achieve are show in figure 1. Specifically, the functionalities include:

- Open the music file to play.
- Can pause and stop the currently playing music. When the music is paused or stopped, can play it again.
- Can achieve some operations on the playlist
  - Double clicks on the mouse can play the music that is being chosed on the playlist.
  - Click the button “Remove” can remove the music that has been

selected from the playlist

- ◆ Has two loop modes
  - Loop one: when the playing of the music stopped, the music will be automatically played again and again.
  - Loop all: loop all the music on the playlist, starting from the music that is currently being played.
- ◆ Can choose the next or last music displayed on the playlist.
- ◆ When click "Exit", the music player closed.

4. Design Implementation

4.1 Interface design

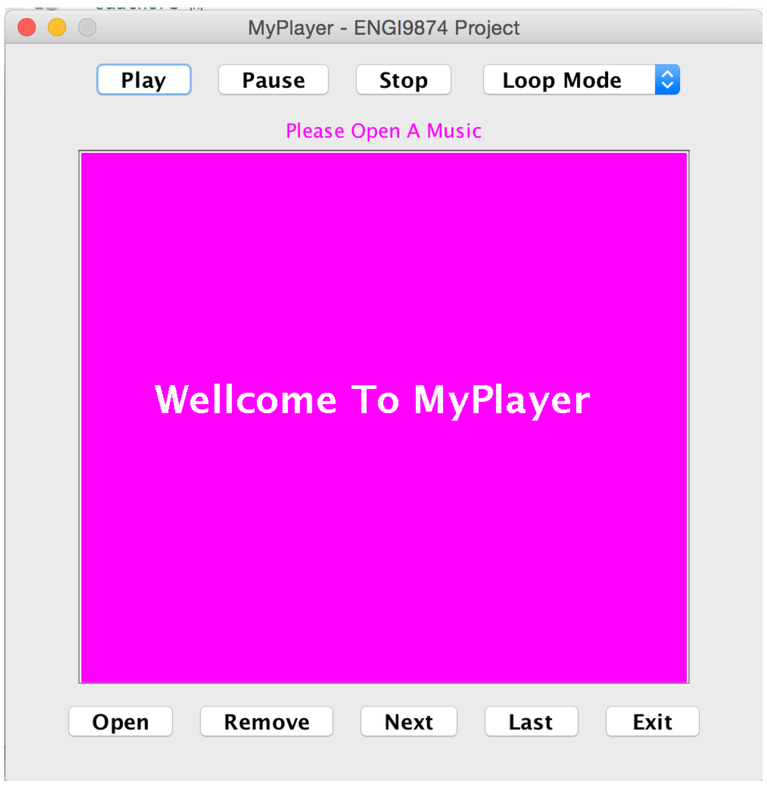


Figure 2. Interface

Figure2 shows the interface of the music player. It includes eight JButtons, one ComboBox, and one JList. We design a welcome graph here by drawing it on the JList. When the music player start working, the graph will disappear. Each button realizes the corresponding functionality of the music player.

4.2 System Design

4.2.1 Class diagram

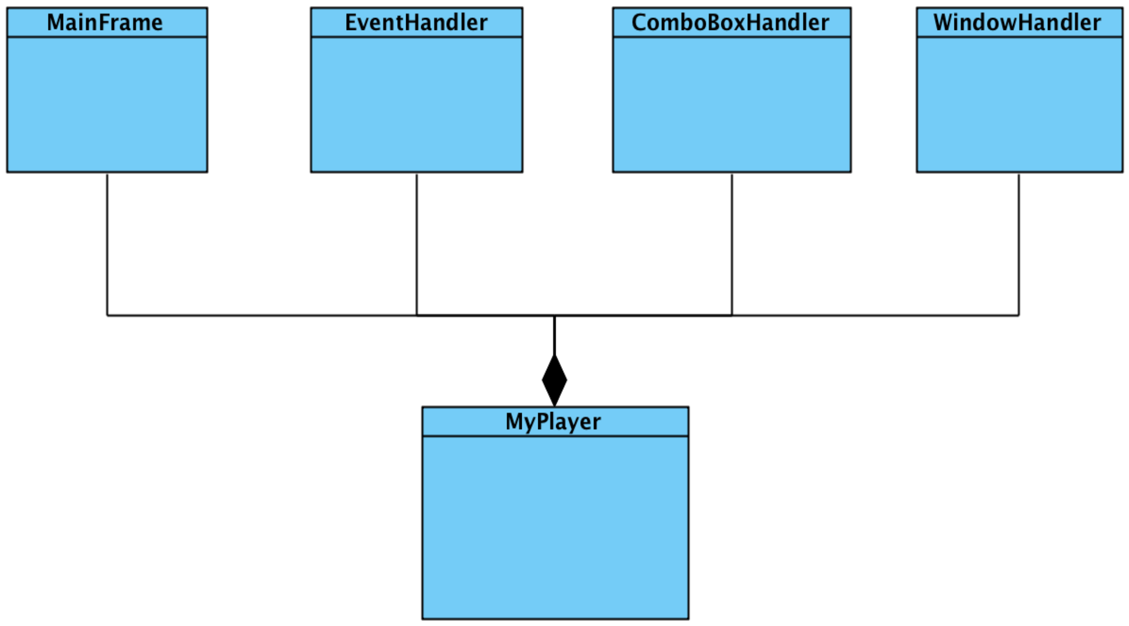


Figure 3. Class diagram system

Actually, there are many classes. Here, we just represent the main ones. Class MainFrame is used to handle the design of the interface. Class EventHandler is used to handle the click of the buttons on the MainFrame to realize the functionalities of the music player. Class ComboBoxHandler is used to handle the choice on the ComboBox, which is loop one or loop all. Class WindowHandler is used to handle the closing of the window. Class MyPlayer link contains one object of each class, linking them together to realize the system of the music Player.

4.2.2 Sequence Diagram

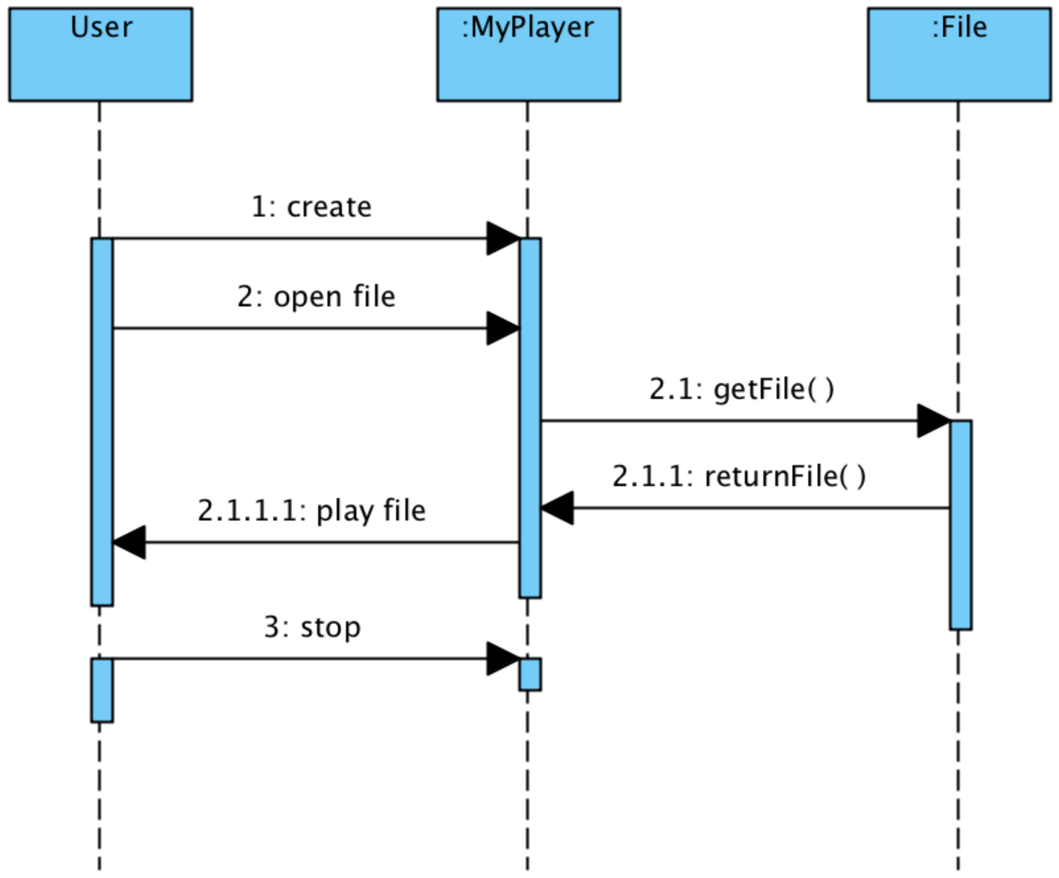


Figure 4. Sequence diagram of the system

Sequence Diagram shows the sequence of interaction process between User and Myplayer instance base on their event, the dashed lines hanging from the boxes are called object lifelines, representing the life span of the object during the scenario being modeled. The long, thin boxes on the lifelines are activation boxes, also called method-invocation boxes.

4.2.3 State Diagram

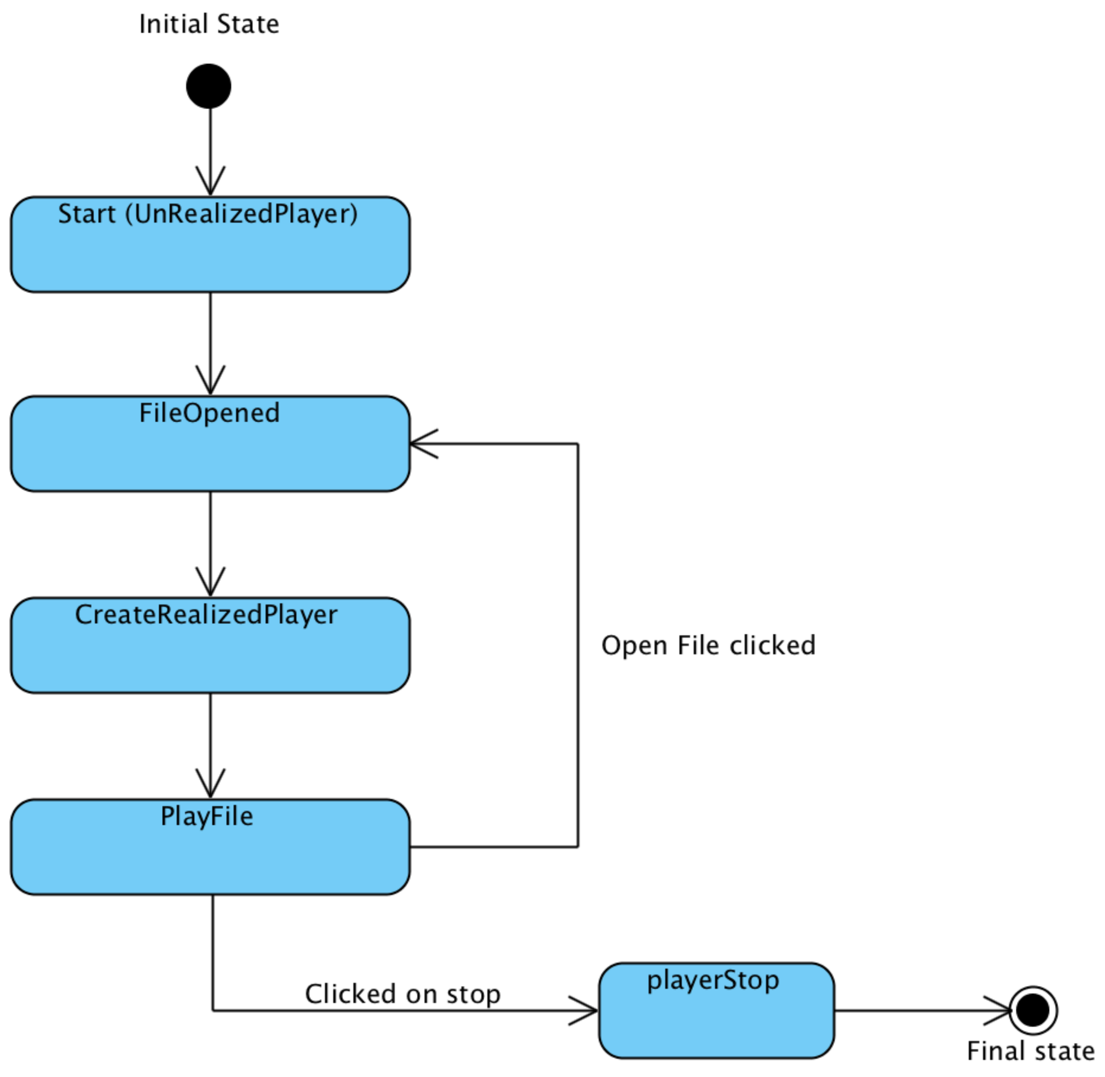


Figure 5. State diagram of the system

State Diagram demonstrate the behavior of an object through many use cases of the system which describe all of the possible states of an object as events occur. Rounded boxes representing the state of the object and arrows indicting the transition of the next state.

4.3 Design Patterns

In this part, we will show the design patterns employed in the system.

4.3.1 Singleton Pattern

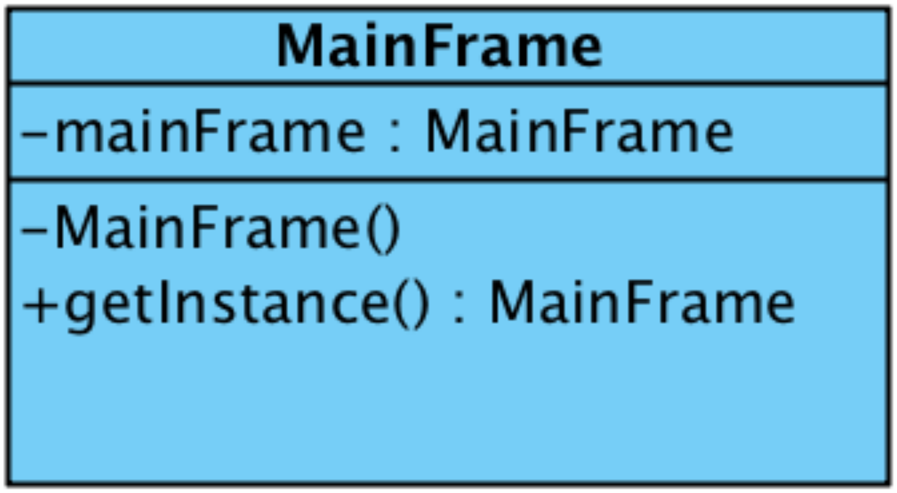
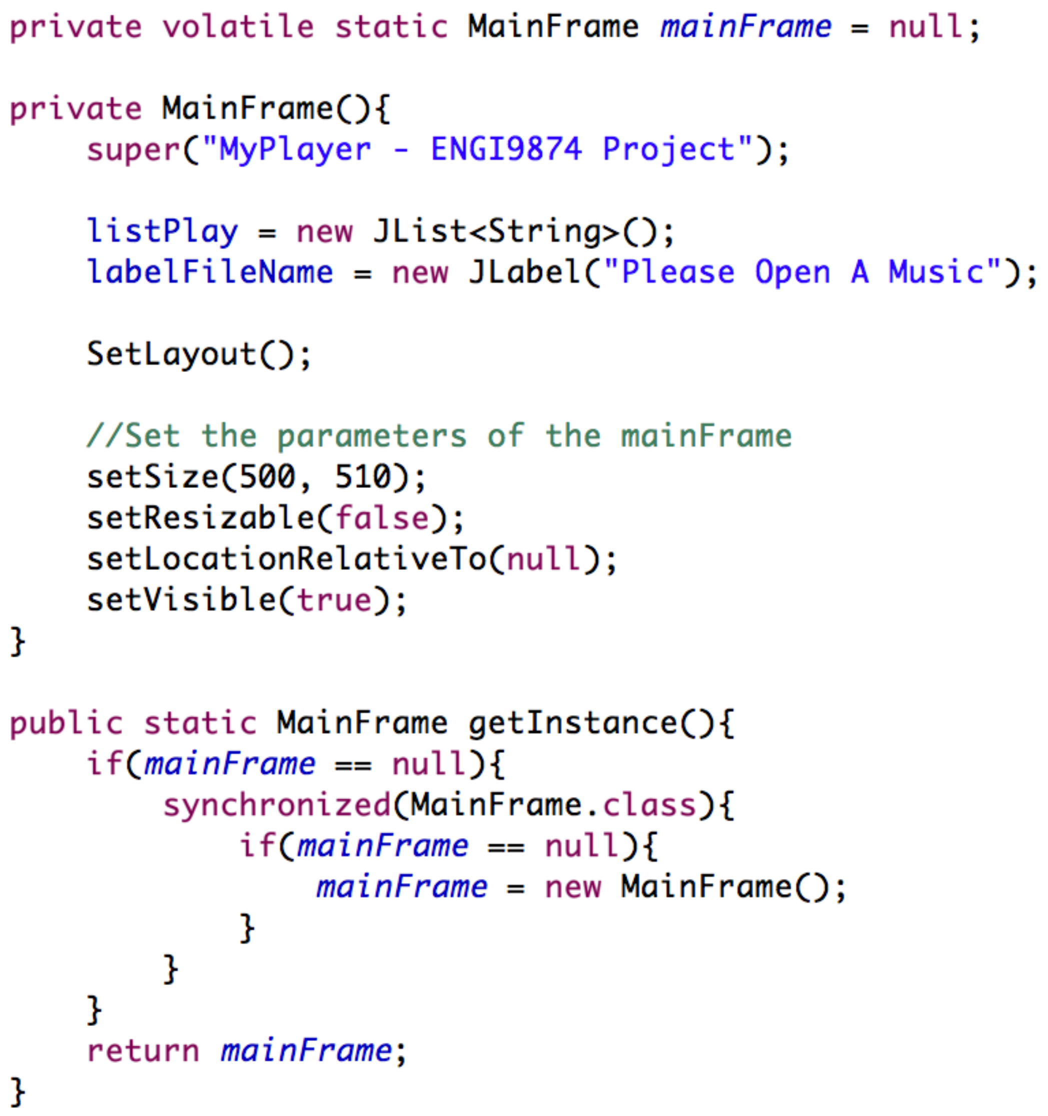


Figure 6. Singleton Pattern

Singleton Pattern used here is to ensure that only exist one object of class MainFrame in the whole system. Because all other classes have methods operate on the object of MainFrame, Singleton Pattern is important here to ensure all these methods operates on the same object of MainFrame. If every other class create a new object of MainFrame and operates on it, the system will be destroyed.

In order to solve multi-thread problems, we use double check here. The corresponding code is show in Figure 7.



```java
private volatile static MainFrame mainFrame = null;

private MainFrame(){
    super("MyPlayer - ENGI9874 Project");

    listPlay = new JList<String>();
    labelFileName = new JLabel("Please Open A Music");

    SetLayout();

    //Set the parameters of the mainFrame
    setSize(500, 510);
    setResizable(false);
    setLocationRelativeTo(null);
    setVisible(true);
}

public static MainFrame getInstance(){
    if(mainFrame == null){
        synchronized(MainFrame.class){
            if(mainFrame == null){
                mainFrame = new MainFrame();
            }
        }
    }
    return mainFrame;
}
```

Figure 7. Code of Singleton Pattern

4.3.2 Abstract Factory Pattern

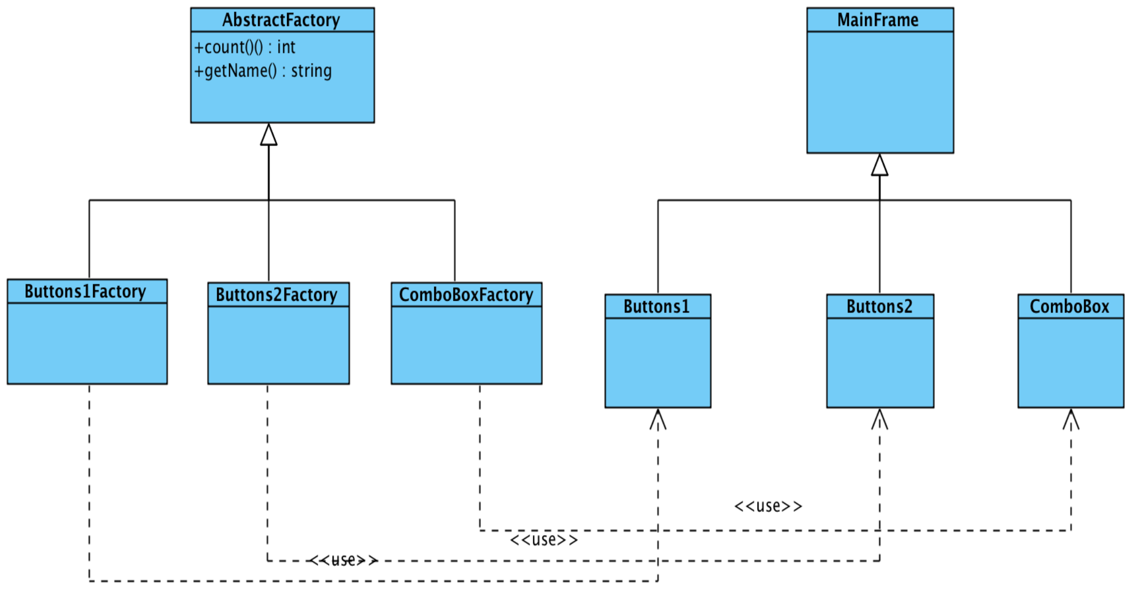


Figure 8. Abstract Factory Pattern

In out design, the components of the main frame are created by the corresponding factories. Abstract Factory Pattern is employed here. The reason to use it is the convenience of creating and removing the components in the main frame. For example, if we want to add or remove a button on the main frame, we just need to change the corresponding factory rather than change the code of the main frame. Therefore, by employing factory method, the extension of creating the main frame has been enhanced.

4.3.3 Proxy Pattern

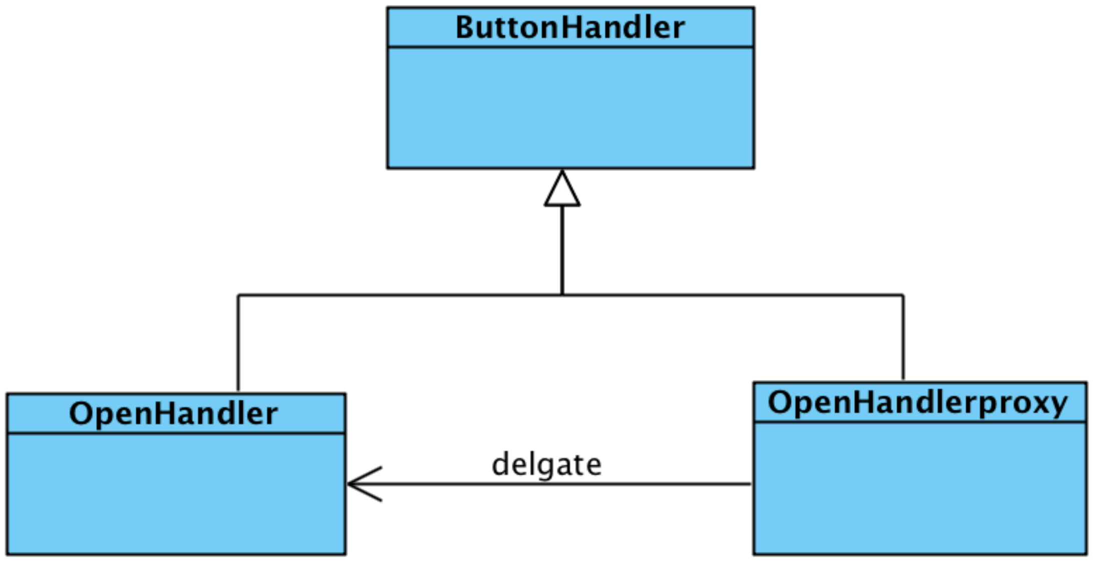
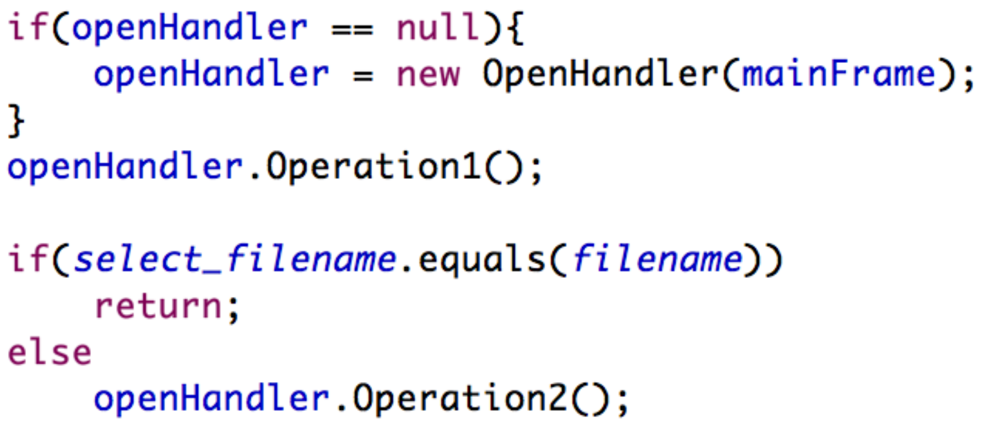


Figure 9. Proxy Pattern

Proxy Pattern is employed here to handle the case when open the music that is currently being played, the music will not be started playing again. We divide the operation of openHandler into two operations. Operation1 get the filename when open a file. Operation2 play the file. In the proxy class, we add some additional rules to Operation2 to realize the target. The detailed code is showing below.



```
if(openHandler == null){
    openHandler = new OpenHandler(mainFrame);
}
openHandler.Operation1();

if(select_filename.equals(filename))
    return;
else
    openHandler.Operation2();
```

Figure 10. Part code of Proxy Pattern

4.3.4 Template Pattern

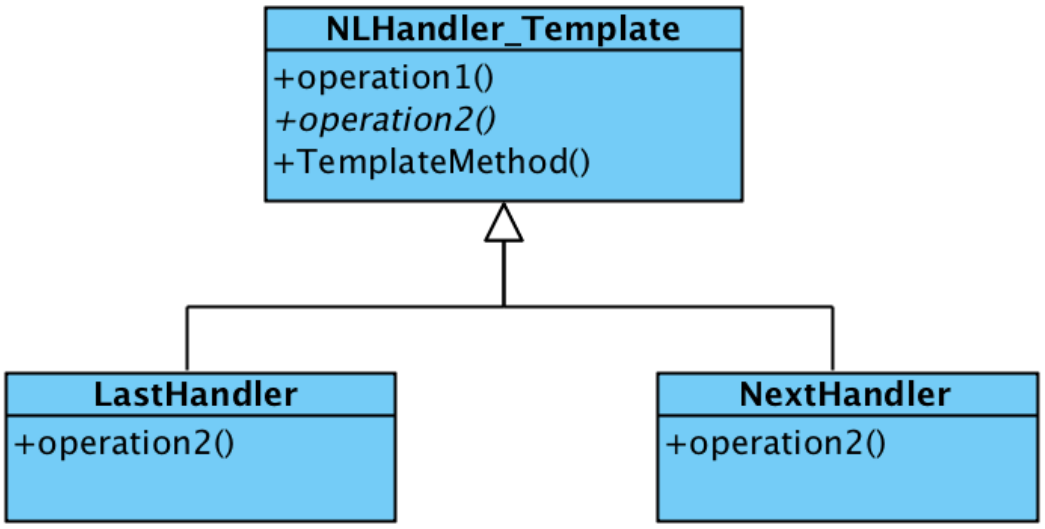
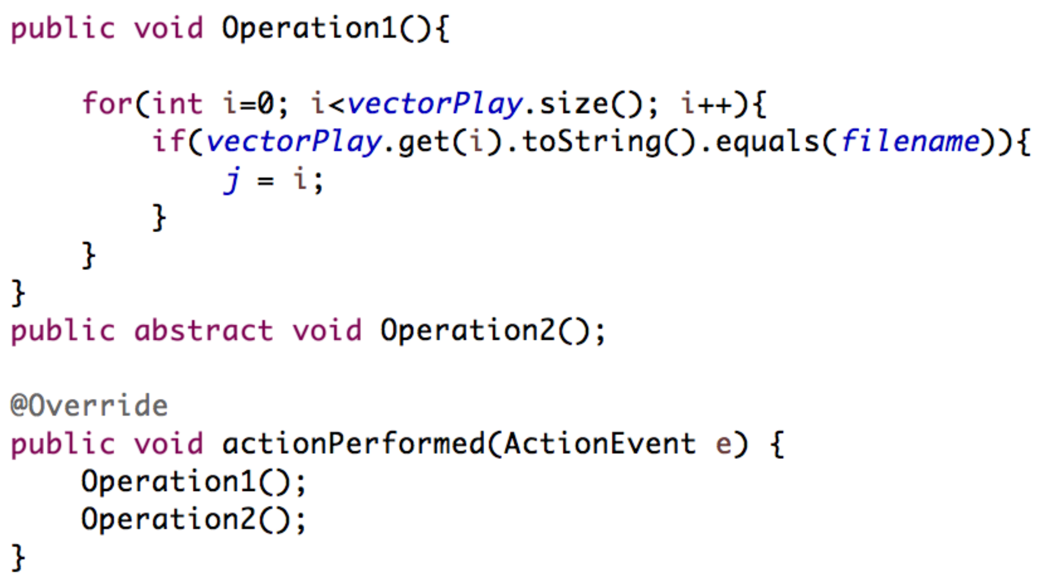


Figure 11. Template Pattern

Since the operation of class LastHandler and class NextHandler is similar, template pattern is employed here. We put the common operation of LastHandler and NextHandler in the super class, and make it a concrete method operation1 in the super class. The different operations of them are implemented in each concrete operation2, which realize the abstract method operation2 in the class NLHandler_Template. In this way, we reduce the code of class LastHandler and class NextHandler, also make the system achieve a better extension.

Part code of the class NLHandler_Template is shown in the following figure.



```java
public void Operation1(){

    for(int i=0; i<vectorPlay.size(); i++){
        if(vectorPlay.get(i).toString().equals(filename)){
            j = i;
        }
    }
}
public abstract void Operation2();

@Override
public void actionPerformed(ActionEvent e) {
    Operation1();
    Operation2();
}
```

Figure 12. Part code of Template Pattern

4.3.5 Command Pattern

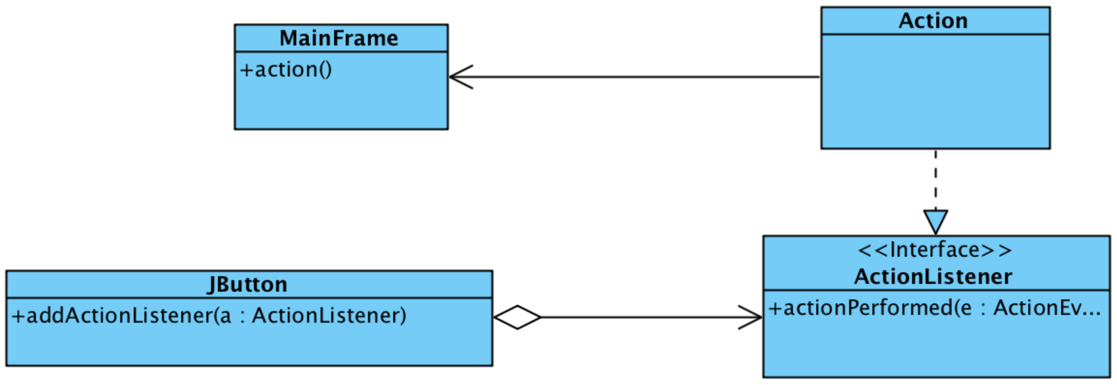

MainFrame
+action()
Action
<<Interface>>
ActionListener
+actionPerformed(e : ActionEv...
JButton
+addActionListener(a : ActionListener)

Figure 13. Command Pattern

Command Pattern is employed here, because our system contains multiple invoker classes (each button as well as a combobox is a invoker) and multiple command classes. Through decoupling the invocation and action, these invoker classes can be mixed and matched with the command classes. Every class is just responsible for one obligation.

4.3.6 Adapter Pattern

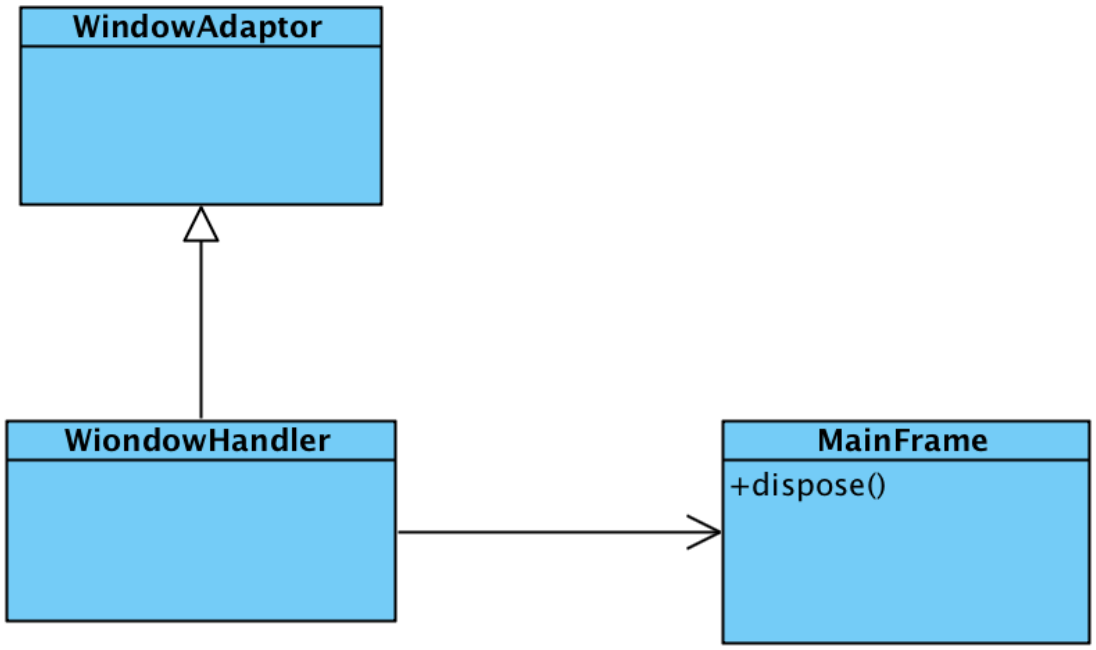


Figure 14. Adapter Pattern

The action of closing the window is realized by Adapter Pattern. Class WindowHandler realizes the abstract class java.awt.event.WindowAdaptor. It maintains a object of class MainFrame. When the method of windowClosing is called, the method of dispose of MainFrame will be called. In this way, we handle the closing of the main frame of the system.

4.4 Functionality Achievement

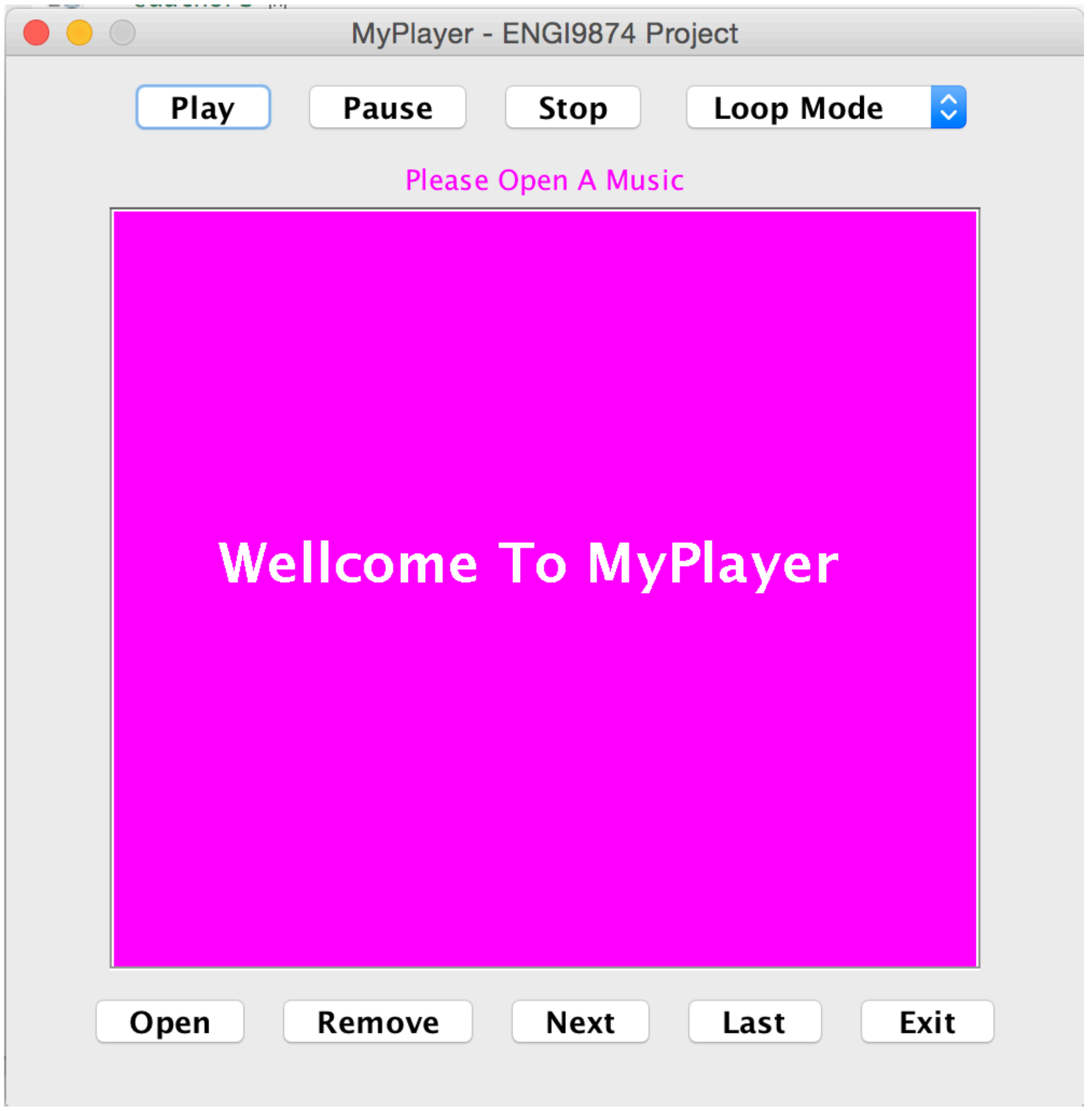
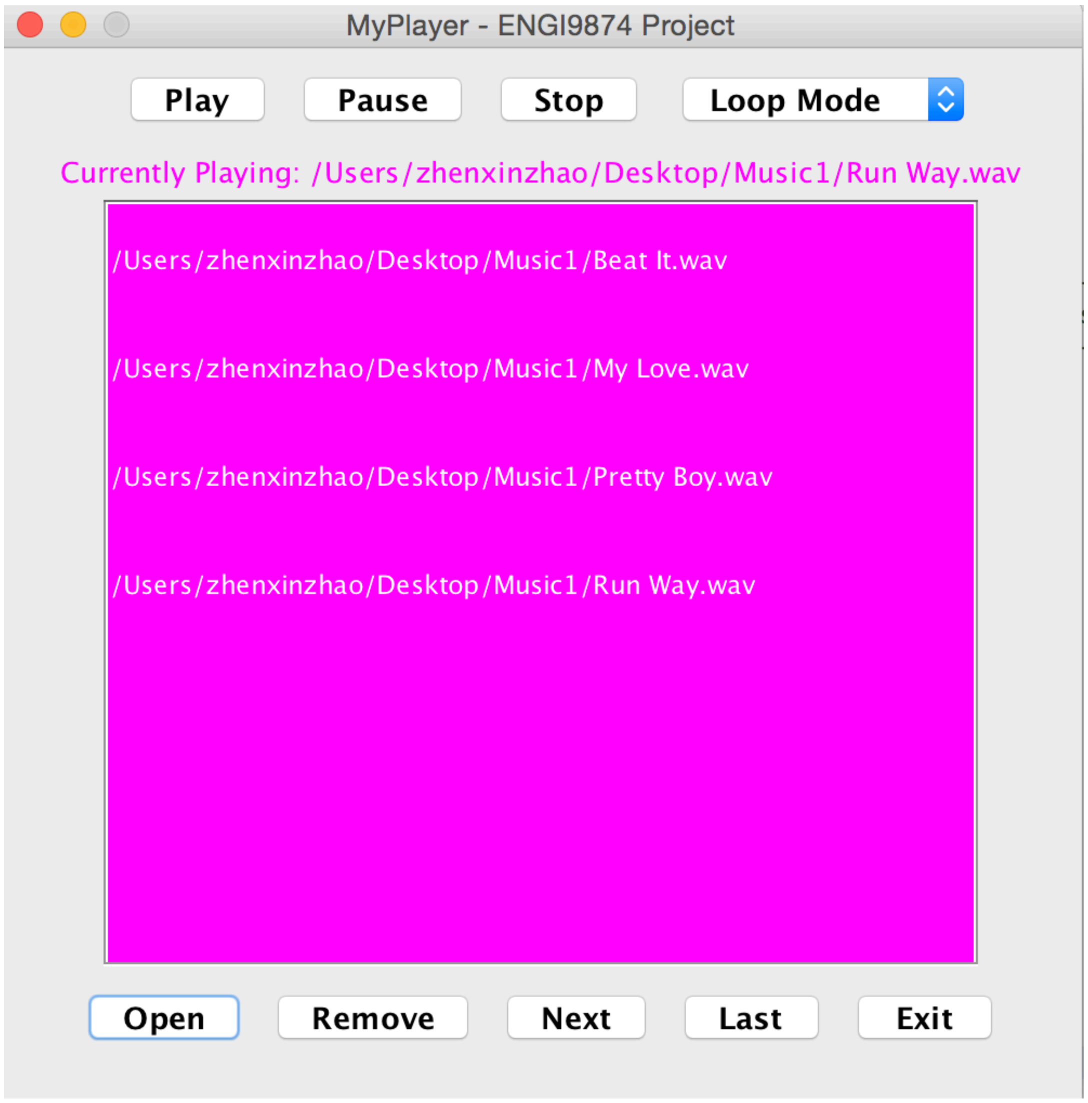


Figure 15. Welcome interface

In the welcome interface, we display a hint “Please Open A Music”, and display “Welcome To Myplayer” in the middle of the frame. The buttons and combobox are used to active the functions of the system.



Currently Playing: /Users/zhenxinzhao/Desktop/Music1/Run Way.wav

/Users/zhenxinzhao/Desktop/Music1/Beat It.wav

/Users/zhenxinzhao/Desktop/Music1/My Love.wav

/Users/zhenxinzhao/Desktop/Music1/Pretty Boy.wav

/Users/zhenxinzhao/Desktop/Music1/Run Way.wav

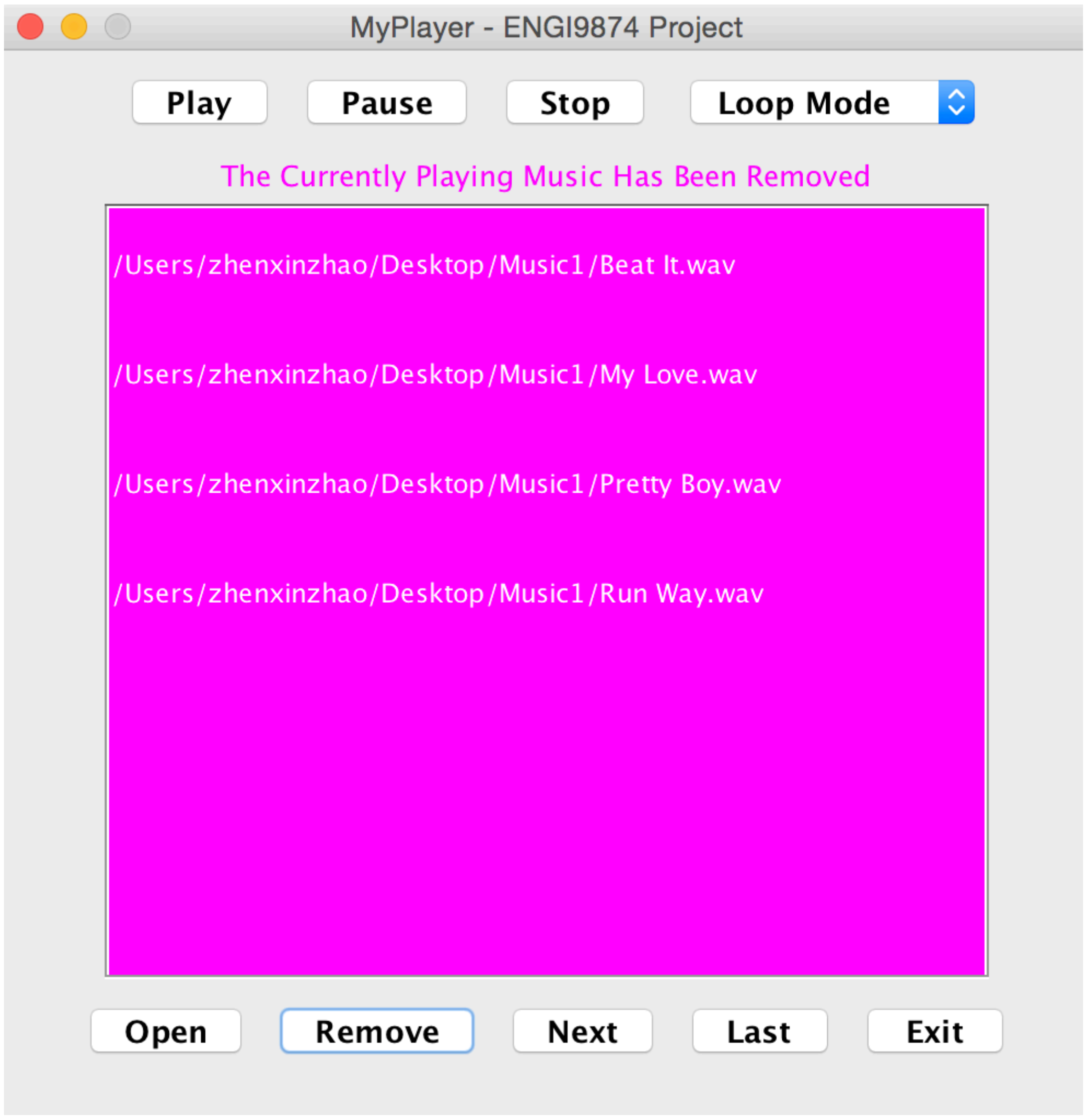
Figure 16. Open music

Open - open a file to play. The name of the music will be automatically added to the playlist.

Play – continue to play the music when it is paused or stopped. Display the filename in the label.

Pause – pause the playing of music. Display the hint “the playing is paused”.

Stop – stop the playing of music. Display the hint “the playing is stopped”.



The Currently Playing Music Has Been Removed

/Users/zhenxinzhao/Desktop/Music1/Beat It.wav

/Users/zhenxinzhao/Desktop/Music1/My Love.wav

/Users/zhenxinzhao/Desktop/Music1/Pretty Boy.wav

/Users/zhenxinzhao/Desktop/Music1/Run Way.wav

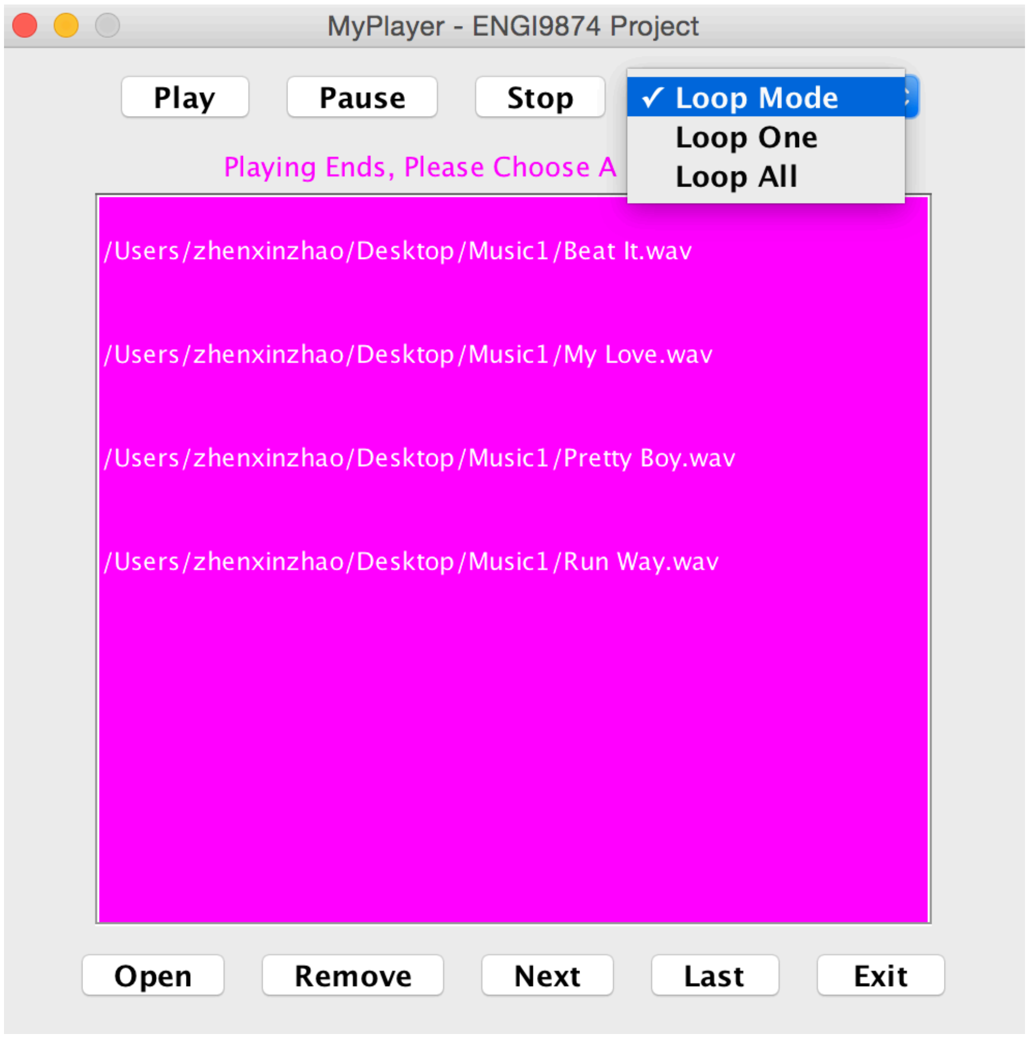
Figure 17. Remove a song

Remove – remove one song from the playlist. If the song is currently being played, the playing should be stopped. And give a hint "The Currently Playing Music Has Been Removed" as show in Figure 17.

Next – play the next song according to the position in the playlist.

Last – play the last song according to the position in the playlist.

Exit – exit the music player.



Playing Ends, Please Choose A

/Users/zhenxinzhao/Desktop/Music1/Beat It.wav

/Users/zhenxinzhao/Desktop/Music1/My Love.wav

/Users/zhenxinzhao/Desktop/Music1/Pretty Boy.wav

/Users/zhenxinzhao/Desktop/Music1/Run Way.wav

Figure 18. Two kinds of loop mode

The choice of two different loop modes is implemented in a combobox. When choose loop one, the currently playing music will be repeat again and again. When choose loop all, the music will be automatically played according to the playlist. When the last song in the playlist finishes, it will automatically play the first song.

5. Conclusion

We learnt much during this project. We learnt what is java media framework and how to use it; we learnt how to use design patterns in the real case; we learnt how to draw the UML diagrams of the system. In the process of implementation the design, we met a bunch of bugs. With great effort, we finally solved all of them.

The functionalities implemented in this system are not complicated. Actually, there are many useful and interesting functions can be add to a music player, such as downloading music from Internet, showing the lyric of the song, and supporting multiple formats of music. Due to time limit and the complication, we did not implement these interesting functionalities. Therefore, we think we could add more functions to the design to make the function more complete in the future. We also could design a more friendly and beautiful interface. In this way, we would use the music player designed by ourselves to listen music forever. We think it is awesome.

6. Reference

[1] http://www.oracle.com/technetwork/java/javase/documentation-138769.html

[2] http://www.ibm.com/developerworks/java/tutorials/j-jmf/j-jmf.html