CST 2550
Software Engineering Management and Development
Coursework 2 Report
Karaoke Application

Abstract

This report aims to describe the implementation of a karaoke application using Java and JavaFX. The program consists of three main components – a song library, a playlist and a media player. The first section of the report explains the design phase of the project. HashMap and LinkedList were chosen as data structures for this project. The pseudo codes were written, and their time complexity was analyzed. To support my choice of data structures, I recorded the time taken for each operation to execute and described my findings in the report below. The GUI was then created based on wireframe initially designed and the karaoke functionalities were added to the interface. The next section describes the testing approach used. Unit testing has been used to test the functionalities. All tests are successful, and the findings are recorded in a table below.

Table of Contents

Abstract....1
1 Introduction....2
2 Design....2
  2.1 Use case diagram....2
  2.2 Pseudo-code and time complexity analysis....3
  2.3 GUI Wireframes....6
3 Testing....6
  3.1 Unit Testing....6
4 Conclusion....8
5 References....9
6 Appendices....9
  6.1 Search song library....9
  6.2 Add Song to library....9
  6.3 Add song to playlist....10
  6.4 Display playlist....10
  6.5 Delete song from playlist....10
  6.6 Add song to library test....11
  6.7 Add non-existent song to playlist test....11
  6.8 Display playlist test....11
  6.9 Delete non-existent song from playlist test....12

1 Introduction

The purpose of this report is to describe the implementation of a karaoke application written in Java. The karaoke software consists of a song library, a media player interface and a playlist. The user can add new songs to the library and search for existing songs. Songs can be inserted at the end of the playlist from the library. When the user presses the play button, the first song from the library is played alongside its video file which is loaded onto the media player interface. Additionally, the song can be paused or skipped. Finally, a song can be deleted from the playlist by specifying its song number.

The report is further broken down into five sections. The design section consists of the use case diagram, the pseudo code and time complexity analysis and the wireframe of the GUI window created. The fifth section demonstrates the tests done using JUnit to verify the functionalities of the Karaoke application. The sixth section comprises of a summary and reflections of the project. The seventh section is a list of references used and the final section is a list of classes appended.

2 Design

2.1 Use case diagram

The use case diagram describes the functional requirements of the karaoke application.

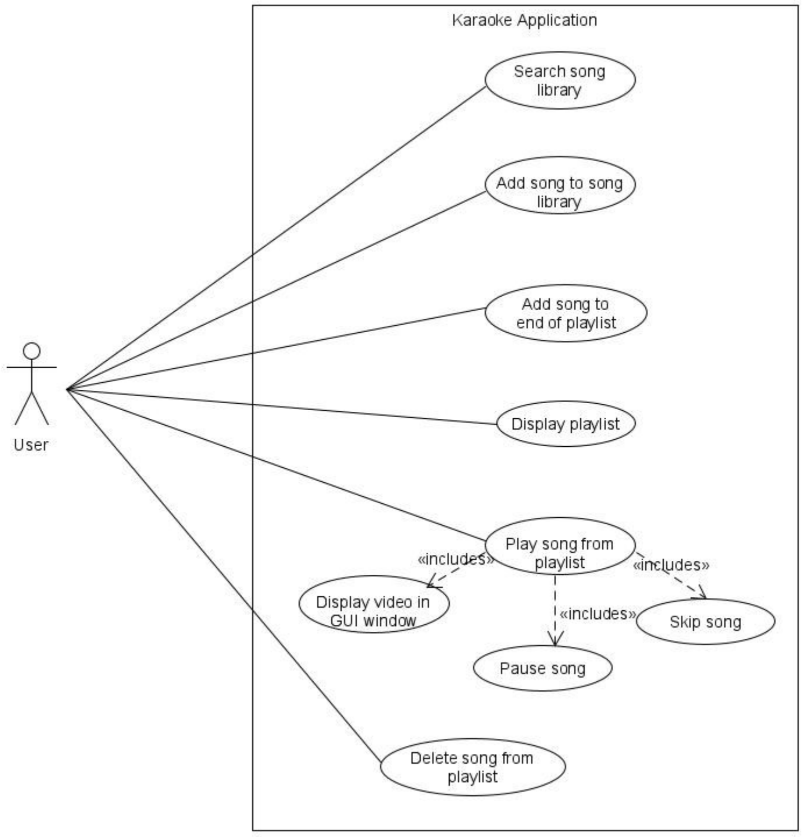


Figure 1 Use case diagram

2.2 Pseudo-code and time complexity analysis

I used a HashMap to store and search songs in the library. It is a data structure that works on the principle of hashing (Paul, 2017). It uses the put and get methods to store and retrieve objects as a key-value pair. Assuming that the key-value pairs are well distributed across the buckets(element of HashMap array used to store nodes (GeeksforGeeks, 2017)), that is, the hashcode implemented is good, then the put and get operations have an amortized complexity of O(1) (Patel, 2015).

To add songs to the playlist, display and delete them, I used a Linked List. It is a linear data structure that allows dynamic memory allocation. The size of the Linked List does not need to be declared since memory is allocated during the run time by the compiler. (Singh, 2016). Elements inserted are stored using objects of type Node. Additionally, the performance of insertion and deletion in a Linked List is constant, O(1). Hence it is less costly in terms of performance (Singh, 2016). When invoking the add method, the linkLast method is internally called (UshaK, 2018). Hence, the added element is stored at the end of the list. The method pollFirst() retrieves and removes the first element from a list. Consequently, both Linked List and HashMap are suitable data structures as they satisfy the coursework requirements.

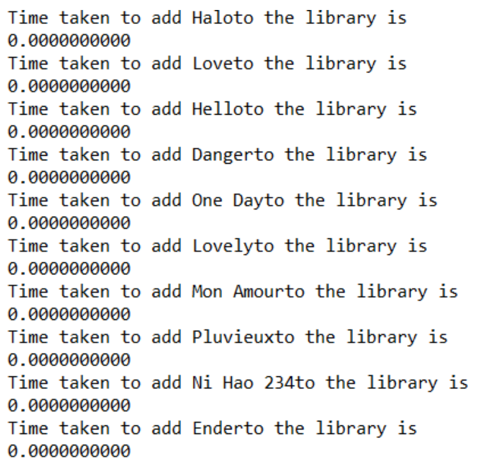
The pseudo codes of the most commonly used functionalities have been designed as shown below. The average time complexity analysis was then determined and confirmed by recording the time taken each operation takes.

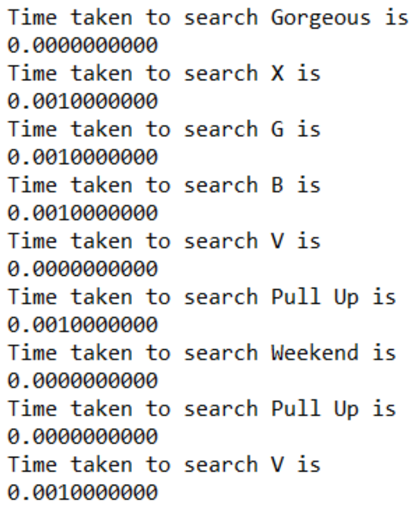
The variables are assigned as shown below:

H <- HashMap
L <- Linked List
A <- Song object
a <- song variables
b <- key
n <- node

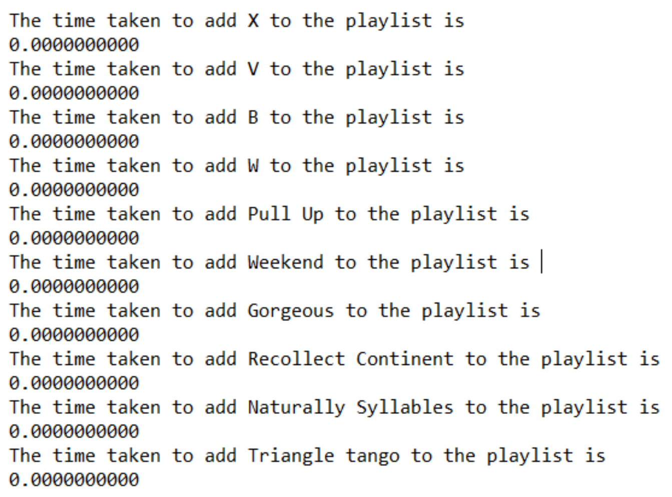
Pseudo Code	**Time Complexity Analysis**

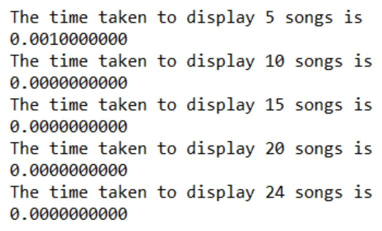
function ADDSONG(a){
    A <- a **C1**
    if H.containsKey(A.getTitle()) then **C2**
        > Song already exists in library **C3**
    else
        H.put(A.getTitle(), A) **C4**
        > Song successfully added to library **C5**
        Print A **C6**
    endif
endfunction
}
	T(n) = C1*1 + C2*1 + C3*1 + C4*1 + C5*1 = 1
	
	Only one song can be added at a time in the library. The time taken each time by the algorithm is always 0.0s as shown below, confirming that the time complexity is O(1).

	 Time taken to add Haloto the library is 0.0000000000 Time taken to add Loveto the library is 0.0000000000 Time taken to add Helloto the library is 0.0000000000 Time taken to add Dangerto the library is 0.0000000000 Time taken to add One Dayto the library is 0.0000000000 Time taken to add Lovelyto the library is 0.0000000000 Time taken to add Mon Amourto the library is 0.0000000000 Time taken to add Pluvieuxto the library is 0.0000000000 Time taken to add Ni Hao 234to the library is 0.0000000000 Time taken to add Enderto the library is 0.0000000000 *Figure 2 Time to add a song*

function SEARCHSONGLIBRARY(b){
if H.containsKey(b) then **C1**
H.get(b).getArtist() **C2**
H.get(b).getRunningTime() **C2**
H.get(b).getTitle() **C3**
else
> Song does not exist **C4**
endif
return H.get(b) **C5**
endfunction
}
	T(n) = C1*1 + C2*1 + C3*1 + C4*1 + C5*1 = 1
	
	Only one song can be searched at a time. The time taken for the search algorithm is almost always constant, taking either 0.000s or 0.001s. The average time taken is 0.001s, showing that the time complexity is O(1).
	
	 Time taken to search Gorgeous is
	0.0000000000
	Time taken to search X is
	0.0010000000
	Time taken to search G is
	0.0010000000
	Time taken to search B is
	0.0010000000
	Time taken to search V is
	0.0000000000
	Time taken to search Pull Up is
	0.0010000000
	Time taken to search Weekend is
	0.0000000000
	Time taken to search Pull Up is
	0.0000000000
	Time taken to search V is
	0.0010000000
	Figure 3 Time taken to search for a song

function ADDSONGTOPLAYLIST(b){
A <- null **C1**
if(H.containsKey(b)) then **C2**
A <- H.get(b) **C3**
if (L.contains(A))then **C4**
> Print song already exists in playlist **C5**
else
L.add(A) **C6**
> Print song added to playlist **C7**
endif
	T(n) = C1*1 + C2*1 + C3*1 + C4*1 + C5*1 + C6*1 + C7*1 + C8*1 = 1
	
	Only one song can be added at a time to the playlist. The time taken is always 0.000s, proving that the algorithm has a time complexity of O(1).

else
    Print Song does not exist in library **C8**
endif
endfunction
}
	
	The time taken to add X to the playlist is
	0.0000000000
	The time taken to add V to the playlist is
	0.0000000000
	The time taken to add B to the playlist is
	0.0000000000
	The time taken to add W to the playlist is
	0.0000000000
	The time taken to add Pull Up to the playlist is
	0.0000000000
	The time taken to add Weekend to the playlist is
	0.0000000000
	The time taken to add Gorgeous to the playlist is
	0.0000000000
	The time taken to add Recollect Continent to the playlist is
	0.0000000000
	The time taken to add Naturally Syllables to the playlist is
	0.0000000000
	The time taken to add Triangle tango to the playlist is
	0.0000000000
	Figure 4 Time taken to add song to playlist

function <u>DISPLAYPLAYLIST</u>(){
    c <- " " **C1**
    int i <- 0 **C2**
    for each A in L do **C3**
        c <- c + i + A.getTitle() + A.getRunningTime() + A.getArtist() **C4**
        i++ **C5**
    endfor
    if (c.length() == 0) then **C6**
        Print Playlist is empty **C7**
    endif
    Print c on GUI window **C8**
endfunction
}
	$T(n) = C1*1 + C2*1 + C3*(n+1) + C4*n + C5*1 + C6*1 + C7*1 + C8*1 = n$
	
	The time taken to display n number of songs has been recorded. As n increases, the time taken stays constant. showing that the algorithm has, in fact, an average time complexity of O(1).
	
	
	The time taken to display 5 songs is
	0.0010000000
	The time taken to display 10 songs is
	0.0000000000
	The time taken to display 15 songs is
	0.0000000000
	The time taken to display 20 songs is
	0.0000000000
	The time taken to display 24 songs is
	0.0000000000
	Figure 5 Time taken to display n songs in playlist

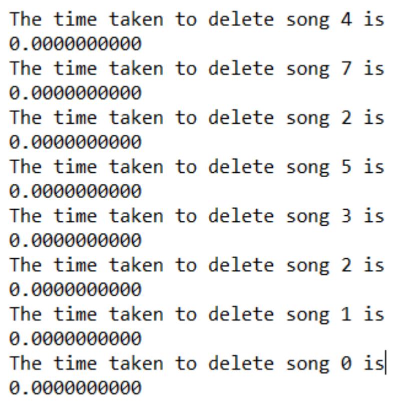
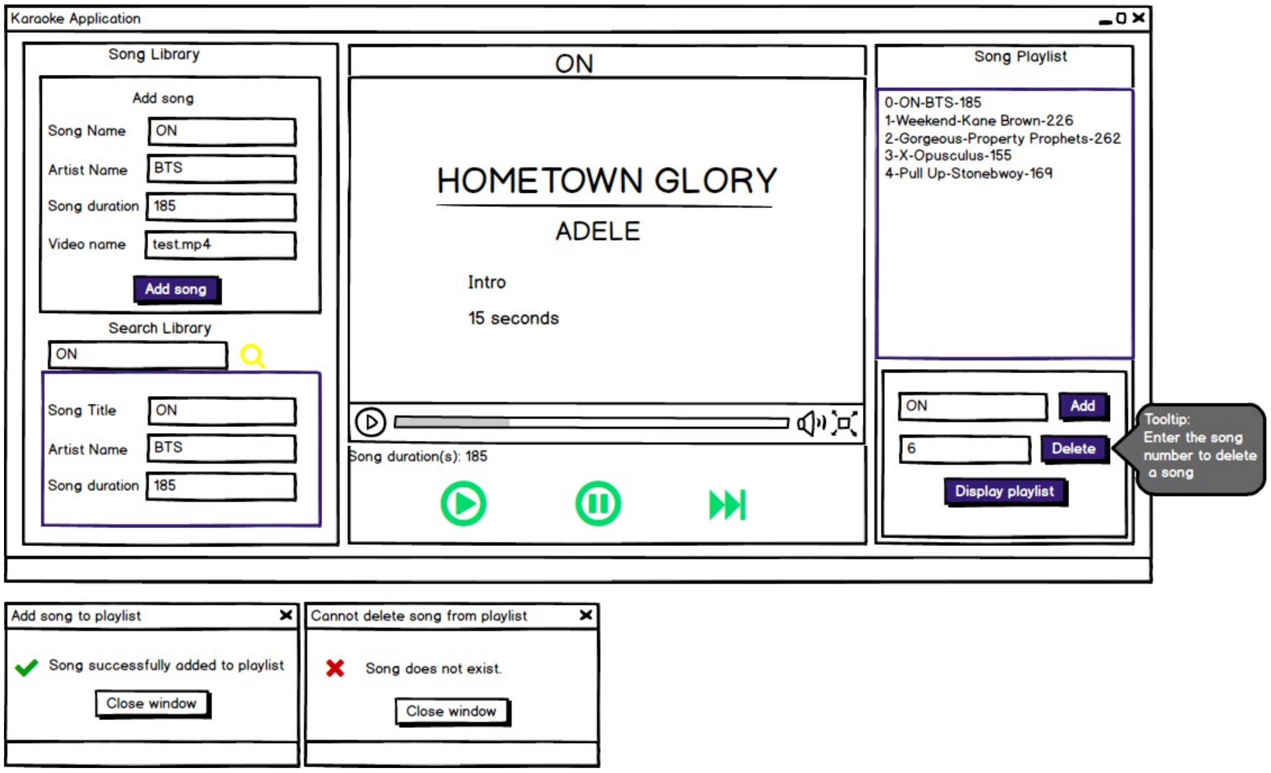
function <u>DELETESONGFROMPLAYLIST</u>(n){
    L.remove(n) **C1**
endfunction
}
	The delete method consists of a single operation. Hence $T(n) = C1*1 = 1$
	
	The time taken to delete a song is always 0.000s, showing that the algorithm has a time complexity of O(1).
	
	
	The time taken to delete song 4 is
	0.0000000000
	The time taken to delete song 7 is
	0.0000000000
	The time taken to delete song 2 is
	0.0000000000
	The time taken to delete song 5 is
	0.0000000000
	The time taken to delete song 3 is
	0.0000000000
	The time taken to delete song 2 is
	0.0000000000
	The time taken to delete song 1 is
	0.0000000000
	The time taken to delete song 0 is
	0.0000000000
	Figure 6 Time taken to delete song from playlist

Table 1 Pseudo Code

2.3 GUI Wireframes

The karaoke interface is a single-window program consisting of three components: the song library, the media player and the song playlist. The interface has been designed by applying Nielsen's heuristics (Nielsen, 2017). The user can easily navigate through the application. All components are clearly labelled. An alert box has been designed to display warnings and successful or error messages. When the user hovers over each button, a tooltip pops up, describing the functionality of each one of them.



Song Name: ON
Artist Name: BTS
Song duration: 185
Video name: test.mp4

Search Library: ON
Song Title: ON
Artist Name: BTS
Song duration: 185

ON
HOMETOWN GLORY
ADELE
Intro
15 seconds
Song duration(s): 185

0-ON-BTS-185
1-Weekend-Kane Brown-226
2-Gorgeous-Property Prophets-262
3-X-Opusculus-155
4-Pull Up-Stonebwoy-169

ON
6

Tooltip: Enter the song number to delete a song

Add song to playlist: Song successfully added to playlist

Cannot delete song from playlist: Song does not exist.

Figure 7 GUI Mockup

3 Testing

For this project, Unit testing is used. Unit testing is a type of software testing that verifies that each unit of the program works as expected (Software Testing Fundamentals, 2018b). Junit is an open-sourced framework which has been used in this project to perform unit testing. The table below demonstrates the tests done for most functionalities. Some test methods written are appended below.

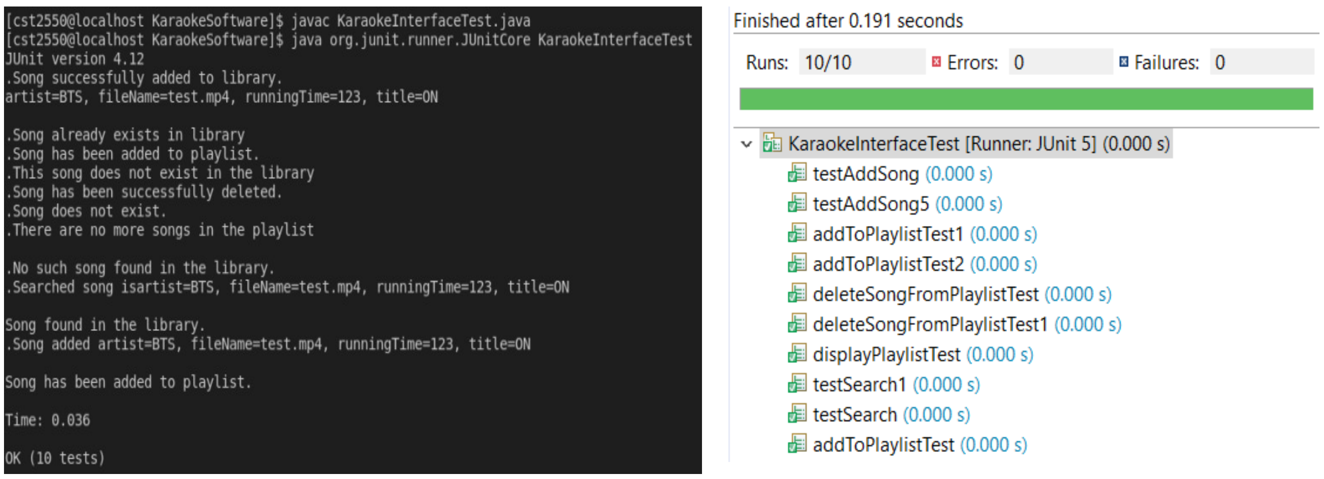
3.1 Unit Testing

Test ID	Test Description	Input	Expected Output	Actual Output	Success(S) / Failed(F)
1.0	Check song title	assertEquals("ON", newSong.getTitle());	"ON"	"ON"	S
1.1	Check artist	assertEquals("BTS", newSong.getArtist());	"BTS"	"BTS"	S
1.2	Check	assertEquals(123,	123	123	S

	running time	newSong.getRunningTime());			
1.3	Check file name	assertEquals("test.mp4", newSong.getFileName());	"test.mp4"	"test.mp4"	S
1.4	Check successful message when adding song to library	assertEquals(expectedMessage, displayMessage);	"Song successfully added to library."	"Song successfully added to library."	S
1.5	Verify song details	assertTrue(KaraokeFunctionalities .songLibrary.containsValue(newS ong));	artist=BTS, fileName=test. mp4, runningTime=1 23, title=ON	artist=BTS, fileName=te st.mp4, runningTim e=123, title=ON	S
1.6	Check that song already exists	assertNotEquals(expectedMessage , displayMessage1);	"Song successfully added to library."	"Song already exists in library"	S
2.0	Search for existing song	assertTrue(KaraokeFunctionalities .songLibrary.containsKey("ON") && KaraokeFunctionalities.songLibra ry.get("ON") != null);	artist=BTS, fileName=test. mp4, runningTime=1 23, title=ON	artist=BTS, fileName=te st.mp4, runningTim e=123, title=ON	S
2.1	Check that song does not exist	assertNotEquals(word, searchFalse);	"Song found in the library."	"No such song found in the library."	S
3.0	Check successful message when adding song to playlist	assertEquals(expectedMsg,output True);	"Song has been added to playlist."	"Song has been added to playlist."	S
3.1	Check song exists in playlist	assertTrue(KaraokeFunctionalities .songPlaylist.contains(sameSong)) ;	artist=BTS, fileName=test. mp4, runningTime=1 23, title=ON	artist=BTS, fileName=te st.mp4, runningTim e=123, title=ON	S
3.2	Check that song already exists in playlist	assertEquals(alreadyExpected, alreadyTrue);	"This song already exists in the playlist"	"This song already exists in the playlist"	S
3.3	Check that	assertNotEquals(outputTrue,	"Song has been	"This song	S

	song does not exist in library	outputFalse);	added to playlist."	does not exist in the library"	
3.4	Check that playlist is not empty	assertFalse(KaraokeFunctionalities.songPlaylist.isEmpty());	0-ON-BTS-123	0-ON-BTS-123	S
4.0	Delete song successfully from playlist	assertEquals(expectedDelete, deleteSong);	"Song has been successfully deleted."	"Song has been successfully deleted."	S
4.1	Verify that non-existent song cannot be deleted	assertNotEquals(expectedDelete, deleteFalse);	"Song has been successfully deleted."	"Song does not exist."	S

Table 2 Unit Testing



```
[cst2550@localhost KaraokeSoftware]$ javac KaraokeInterfaceTest.java
[cst2550@localhost KaraokeSoftware]$ java org.junit.runner.JUnitCore KaraokeInterfaceTest
JUnit version 4.12
.Song successfully added to library.
artist=BTS, fileName=test.mp4, runningTime=123, title=ON

.Song already exists in library
.Song has been added to playlist.
.This song does not exist in the library
.Song has been successfully deleted.
.Song does not exist.
.There are no more songs in the playlist

.No such song found in the library.
.Searched song isartist=BTS, fileName=test.mp4, runningTime=123, title=ON

Song found in the library.
.Song added artist=BTS, fileName=test.mp4, runningTime=123, title=ON

Song has been added to playlist.

Time: 0.036

OK (10 tests)
```

```
Finished after 0.191 seconds
Runs: 10/10    Errors: 0    Failures: 0

KaraokeInterfaceTest [Runner: JUnit 5] (0.000 s)
    testAddSong (0.000 s)
    testAddSong5 (0.000 s)
    addToPlaylistTest1 (0.000 s)
    addToPlaylistTest2 (0.000 s)
    deleteSongFromPlaylistTest (0.000 s)
    deleteSongFromPlaylistTest1 (0.000 s)
    displayPlaylistTest (0.000 s)
    testSearch1 (0.000 s)
    testSearch (0.000 s)
    addToPlaylistTest (0.000 s)
```

Figure 8 Successful unit testing

4 Conclusion

In order to design a karaoke application, I first designed a Song class implementing the Comparable interface. After analyzing the coursework requirements, I created a use case diagram. Afterwards, I designed the pseudo code of all functionalities and recorded their execution time. The data structures used for this project are HashMap and Linked List. The media player was created using the MediaPlayer class' methods. The wireframe of the karaoke application was then designed and finally all functionalities were implemented. To validate the functions, unit testing was used. All tests were successful as shown in the table.

There are several limitations for this project. A song file can be added only via the terminal and in the text file format. The list of songs stored in the library is displayed on the terminal instead of an interface. Additionally, manual entry of a new song via the interface is stored only in the HashMap and not in the text file. Hence the song is removed when the window is closed. There is no time progress bar to display the duration of each song. The display pane displays only the first 17 songs. One cannot scroll along the display plane to view all songs. A song can be deleted only by inserting its index instead of the song title. For a similar future project, I would implement my own data structure(s). Secondly, I would display list of data more neatly in tables and finally I would design a more user-friendly interface.

5 References

[1] GeeksforGeeks. (2017). *Internal Working of HashMap in Java*. [online] Available at: https://www.geeksforgeeks.org/internal-working-of-hashmap-java/ [Accessed 5 May 2020].

[2] Nielsen, J. (2017). *10 Heuristics for User Interface Design: Article by Jakob Nielsen*. [online] Nielsen Norman Group. Available at: https://www.nngroup.com/articles/ten-usability-heuristics/ [Accessed 24 Apr. 2020].

[3] Paul, J. (2013). *10 Examples of HashMap in Java - Programming Tutorial*. [online] Java67. Available at: https://www.java67.com/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html [Accessed 3 May 2020].

[4] Patel, J. (2015). *How time complexity of Hashmap get() and put() operation is O(1)? Is it O(1) in any condition? | JavaByPatel*. [online] javabypatel.blogspot.com. Available at: http://javabypatel.blogspot.com/2015/10/time-complexity-of-hashmap-get-and-put-operation.html [Accessed 18 Apr. 2020].

[5] Paul, J. (2017). *How HashMap works in Java*. [online] Javarevisited. Available at: https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html [Accessed 5 May 2020].

[6] Singh, C. (2016). *LinkedList in Java with Example*. [online] BeginnersBook. Available at: https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/ [Accessed 3 May 2020].

[7] Software Testing Fundamentals. (2018b). *Unit Testing - Software Testing Fundamentals*. [online] Available at: http://softwaretestingfundamentals.com/unit-testing/ [Accessed 2 May 2020].

[8] UshaK (2018). *LinkedList Internal Implementation in Java*. [online] KnpCode. Available at: https://knpcode.com/java/collections/linkedlist-internal-implementation-in-java/ [Accessed 5 May 2020].

6 Appendices

6.1 Search song library

```
static Song searchSongLibrary(String songSearched) {
        if (songLibrary.containsKey(songSearched)) {
                KaraokeInterface.searchSongOutput.setText(songLibrary.get(songSearched).getTitle());
                KaraokeInterface.searchArtistOutput.setText(songLibrary.get(songSearched).getArtist());
                KaraokeInterface.searchDurationOutput
                                .setText(String.valueOf(songLibrary.get(songSearched).getRunningTime()));
        } else {
                AlertBox.displayPopUp("Cannot find song", "No such song found in the library.");
        }
        return songLibrary.get(songSearched);
    } // end of searchSongLibrary
```

6.2 Add Song to library

```
    static void addSong(String songName, String artistName, int songDuration, String fileName) {
        Song newSong = new Song(songName, artistName, songDuration, fileName);
        if (songLibrary.containsKey(newSong.getTitle())) {
            AlertBox.displayPopUp("Song already exists", "Song already exists in library");
        } else {
            songLibrary.put(newSong.getTitle(), newSong);
            AlertBox.displayPopUp("Add song to library", "Song successfully added to library.");
            System.out.println("New song added to library: \n" + newSong);
        }
    }// end of addSong
```

6.3 Add song to playlist

```
    static void addSongToPlaylist(String songName) {
        Song sameSong = null;
        if (songLibrary.containsKey(songName)) {
            sameSong = songLibrary.get(songName);
            if (songPlaylist.contains(sameSong)) {
                AlertBox.displayPopUp("Song already exists", "This song already exists in the playlist");
            } else {
                songPlaylist.add(sameSong);
                displayPlaylist();
                AlertBox.displayPopUp("Song added to playlist", "Song has been added to playlist.");
            }
        } else {
            AlertBox.displayPopUp("Song cannot be added to playlist", "This song does not exist in the
library");
        }
    }// end of addSongToPlaylist
```

6.4 Display playlist

```
    static void displayPlaylist() {
        String outputSongNumber = "";
        int i = 0;
        for (Song song : songPlaylist) {
            outputSongNumber = outputSongNumber + i + " - " + song.getTitle() + " - " + song.getArtist() + "
- "
                    + song.getRunningTime() + "\n";
            i++;
        }
        if (outputSongNumber.length() == 0) {
            AlertBox.displayPopUp("Playlist empty", "There are currently no songs in the playlist");
        }
        KaraokeInterface.loadPlaylist.setText(outputSongNumber);
    }// end of displayPlaylist
```

6.5 Delete song from playlist

```
    static void deleteSongFromPlaylist(int index) {
        try {
            songPlaylist.remove(index);
            displayPlaylist();
				AlertBox.displayPopUp("Song deleted", "Song has been successfully deleted.");
			} catch (IndexOutOfBoundsException ex) {
				AlertBox.displayPopUp("Song cannot be deleted", "Song does not exist.");
			}
		}// end of deleteSongFromPlaylist
```

6.6 Add song to library test

```
		@Test
		public final void testAddSong() {
			String songName = "ON";
			String artistName = "BTS";
			int songDuration = 123;
			String fileName = "test.mp4";
			String expectedMessage = "Song successfully added to library.";
			String displayMessage = "Song successfully added to library.";
			Song newSong = new Song(songName, artistName, songDuration, fileName);
			if (KaraokeFunctionalities.songLibrary.containsKey(newSong.getTitle())) {
				System.out.println("Song already exists in library");
			} else {
				KaraokeFunctionalities.songLibrary.put(newSong.getTitle(), newSong);
				System.out.println(displayMessage);
				System.out.println(newSong);
			}

			assertEquals("ON", newSong.getTitle());
			assertEquals("BTS", newSong.getArtist());
			assertEquals(123, newSong.getRunningTime());
			assertEquals("test.mp4", newSong.getFileName());
			assertEquals(expectedMessage, displayMessage);
			assertTrue(KaraokeFunctionalities.songLibrary.containsKey("ON") &&
KaraokeFunctionalities.songLibrary.get("ON") != null);
			assertTrue(KaraokeFunctionalities.songLibrary.containsValue(newSong));

		}
```

6.7 Add non-existent song to playlist test

```
		@Test
		public final void addToPlaylistTest2() { //does not exist
			String songName = "Tester";
			Song sameSong = null;
			String outputTrue = "Song has been added to playlist.";
			String outputFalse = "This song does not exist in the library";
			String alreadyTrue = "This song already exists in the playlist";
			if (KaraokeFunctionalities.songLibrary.containsKey(songName)) {
				sameSong = KaraokeFunctionalities.songLibrary.get(songName);
				if (KaraokeFunctionalities.songPlaylist.contains(sameSong)) {
					System.out.println(alreadyTrue);
				} else {
					KaraokeFunctionalities.songPlaylist.add(sameSong);
					System.out.println(outputTrue);
				}
			} else {
				System.out.println(outputFalse);
			}

			assertNotEquals(outputTrue, outputFalse);
```

```
	}
```

6.8 Display playlist test

```
	@Test
	public final void displayPlaylistTest() {
		String outputSongNumber = "";
		int i = 0;
		for (Song song : KaraokeFunctionalities.songPlaylist) {
			outputSongNumber = outputSongNumber + i + " - " + song.getTitle() + " - " + song.getArtist() + "
- "
						+ song.getRunningTime() + "\n";
			i++;
		}
		if (outputSongNumber.length() == 0) {
			System.out.println("There are no more songs in the playlist");
		}
		System.out.println(outputSongNumber);

		assertFalse(KaraokeFunctionalities.songPlaylist.isEmpty());
	}
```

6.9 Delete non-existent song from playlist test

```
	@Test
	public final void deleteSongFromPlaylistTest1() { //song not exist
		int index = 0;
		String expectedDelete = "Song has been successfully deleted.";
		String deleteSong = "Song has been successfully deleted.";
		String deleteFalse = "Song does not exist.";
		try {
			KaraokeFunctionalities.songPlaylist.remove(index);
			System.out.println(deleteSong);
		} catch (IndexOutOfBoundsException ex) {
			System.out.println(deleteFalse);
		}

		assertNotEquals(expectedDelete, deleteFalse);

	}
```