Test Case

## Method Overview

The method selected for the included test case is addSong(Request r), which has the following function signature:

**public synchronized boolean** addSong(Request r);

When excuted, this method attempts to add the song in the given Request object to the current list of songs playing. Any Request object may be received. Request objects consist of a Song and an optional specified time in hours and minutes. Songs consist of minutes and seconds, and the method returns a boolean indicating whether the song was successfully inserted. The complete set of test cases involving this method are detailed in Section V of the Design Document.

## Selected Test Case

Cases 1-3 in Section V of the Design Document are selected for this logging.

*Let T=0 be the current time, and Tf be the latest time for which there is a song scheduled to be played (on the queue). Let Tr be the requested time of the Request r if specified.*

Set-up

@Before

**public** **void** setUp() **throws** Exception {

MusicManager man = **new** MusicManager(**false**);

mQueue1 = **new** ActiveQueue(MIN, man);

mQueue2 = **new** ActiveQueue(MIN, man);

// Block any client requests

mQueue1.terminateWatcher();

mQueue2.terminateWatcher();

// Insert songs onto the list

Song s1 = **new** Song("files/02 We're In The Club Now.mp3");

Song s2 = **new** Song("files/03 Married Life.mp3");

Song s3 = **new** Song("files/04 Carl Goes Up.mp3");

Song s4 = **new** Song("files/05 52 Chachki Pickup.mp3");

LinkedList<Song> list = **new** LinkedList<Song>();

list.add(s1);

list.add(s2);

list.add(s3);

list.add(s4);

mQueue1.setSongs(list);

mQueue2.setSongs(list);

}

Boundaries Testing

/\*\*

\* Test method for {@link MusicManager.ActiveQueue#addSong(MusicManager.Request)}.

\*/

@Test

**public** **void** testAddSongBoundaries() {

Song testSong1 = **new** Song("files/01 Up With Titles.mp3");

Song testSong2 = **new** Song("files/22 Up With End Credits.mp3");

File session = **new** File("session.ser");

// Destroy any saved playlist/active queue settings

**if**(session.exists()) {

session.delete();

}

DateTime date = **new** DateTime();

**int** hour = date.getHourOfDay();

**int** min = date.getMinuteOfDay();

// Boundary case 1: Request r = null; *assertFalse*(mQueue1.addSong(**null**));

// Boundary case 2: r != null; r.min < 0, r.hour >= 0

*assertTrue*(mQueue1.addSong(**new** Request(testSong1, -1, hour)));

*assertTrue*(mQueue1.getSongs().size() == mQueue2.getSongs().size());

// Boundary case 3: r != null; r.min >= 0, r.hour < 0

*assertTrue*(mQueue1.addSong(**new** Request(testSong2, min, -1)));

*assertTrue*(mQueue1.getSongs().size() == mQueue2.getSongs().size());

// Boundary case 4: r != null; r.song = null;

*assertFalse*(mQueue2.addSong(**new** Request(**null**, 2, 2)));

}

Case 1: Tr != null, Tf > Tr.

/\*\*

\* Test method for {@link MusicManager.ActiveQueue#addSong(MusicManager.Request)}.

\*/

@Test

**public** **void** testAddSongCase1() {

Song testSong1 = **new** Song("files/Sleep Away.mp3");

DateTime date = **new** DateTime();

**int** hour = date.getHourOfDay();

**int** min = date.getMinuteOfHour();

// Find time the latest existing song will be played

**int** eHour = hour;

**int** eMin = min;

LinkedList<Song> list = mQueue1.getSongs();

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

eMin += list.get(i).getSongLength();

}

**while**(eMin >= 60) {

eHour++;

eMin -= 60;

}

// Insert at a position smaller than that time

*assertTrue*(mQueue1.addSong(**new** Request(testSong1, eHour, eMin - 10)));

// There should have been a swap assuming normal range of song length

*assertTrue*(mQueue1.getSongs().size() == list.size());

*assertTrue*(mQueue1.getSongs().contains(testSong1));

}

Case 2: Tr != null, Tf < Tr.

/\*\*

\* Test method for {@link MusicManager.ActiveQueue#addSong(MusicManager.Request)}.

\* **@throws** SQLException

\* **@throws** ClassNotFoundException

\*/

@SuppressWarnings("static-access")

@Test

**public** **void** testAddSongCase2() **throws** ClassNotFoundException, SQLException {

Song testSong1 = **new** Song("files/Bleed.mp3");

DateTime date = **new** DateTime();

**int** hour = date.getHourOfDay();

**int** min = date.getMinuteOfHour();

// Load a test playlist for generating filler

**int** pid = Global.*getSRVInstance*().*PlaylistService*().GetPlaylistIdByName("rUp");

Collection<Song> coll = Global.*getSRVInstance*().*SongService*().getSongsByPlaylistId(pid);

LinkedList<Song> link = **new** LinkedList<Song>();

link.addAll(coll);

Playlist playlist = **new** Playlist();

playlist.setSongs(link);

mMan.changePlaylist(playlist);

mQueue1 = **new** ActiveQueue(MIN, mMan);

mQueue1.setSongs(mList);

**int** origLength = mQueue1.getSongs().size();

*assertTrue*(mQueue1.addSong(**new** Request(testSong1, hour, min + 40)));

**int** newLength = mQueue1.getSongs().size();

// The last element should be the added song

*assertTrue*(mQueue1.getSongs().get(newLength - 1).equals(testSong1));

// Additions; the length of the song list should be longer by > 1

*assertFalse*(mQueue1.getSongs().size() == origLength + 1);

// Check that insertion was +-3 minutes

**int** sum = 0;

**int** target = hour\*60 + min;

Song temp;

LinkedList<Song> songs = mQueue1.getSongs();

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

temp = songs.get(i);

**if**(temp == testSong1) {

System.*out*.println("Found it.");

**break**;

}

sum += temp.getSongLength();

}

System.*out*.println("Sum: " + sum);

System.*out*.println("Target: " + target);

*assertTrue*(Math.*abs*(sum - target) <= 3);

// Depending on sample songs used, this may fail.

// With this set it always passes. Failing occasionally is OK.

}

Case 3: Tr == null.

/\*\*

\* Test method for {@link MusicManager.ActiveQueue#addSong(MusicManager.Request)}.

\*/

@Test

**public** **void** testAddSongCase3() {

Song testSong1 = **new** Song("files/Bird Song.mp3");

mQueue1 = **new** ActiveQueue(MIN, mMan);

mQueue1.setSongs(mQueue2.getSongs());

// The length of the song list should be equal after a successful insert

*assertTrue*(mQueue1.addSong(**new** Request(testSong1, -1, -1)));

*assertTrue*(mQueue1.getSongs().size() == mQueue2.getSongs().size());

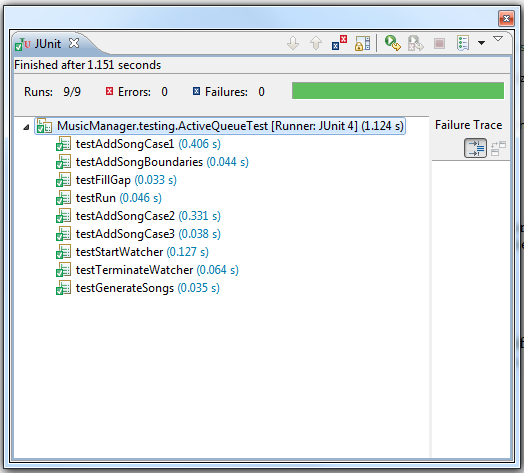
*assertTrue*(mQueue1.getSongs().contains(testSong1));

// It should be the third song on the list  *assertTrue*(mQueue1.getSongs().get(2).getTitle().equals(testSong1.getTitle()));

}

## Results

The results for all test cases were as expected, after a few failures due to errors in the test case assertions. These were not in the actual code, so are not logged here. With these corrected, the results were:



Note that as stated in the Design Document, since all tests are based in specific instances of Song objects (with distinct lengths), extended testing using varying playlists, songs, and times of day is also being done to thoroughly test this method.