P08 MusicPlayer300

Overview

In this program, you'll be creating a queue-based music player that loads song files from your local machine and plays them using your computer's sound output (speakers or headphones). You'll interact with this music player using a text-based console menu that you will also create.

Familiarity with the Java library we'll use to accomplish this, `javax.sound.sampled`, isn't the point of this assignment, so we'll provide an interface to the library for you to use by way of a JAR file as we did with the GUI libraries earlier this semester. However, if you'd like to explore the library on your own (outside of this assignment), you're strongly encouraged to! It's very fun to add audio effects to your programs.

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **11/22**.
15 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. ⭐ As we're getting to the end of the semester, be aware that more and more of the immediate tests will focus on YOUR TESTER METHODS, rather than your implementation. This program will still include some non-tester class immediate feedback, but *many* of the immediate tests will be focused on your tester class. ⭐ Passing all immediate automated tests does **not** guarantee full credit for the assignment.
20 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.
10 points	**Manual Grading**: performed by TAs once all submissions are in. These tests focus on commenting and style but may also include algorithm checks.

Learning Objectives

After completing this assignment, you should be able to:

- **Describe** the functionality of a queue data structure and **explain** its operational details when implemented using a linked list.
- **Create** a simple menu-based text interface for your programs
- **List** the special situations (edge cases) that need to be verified for queue data structures

Additional Assignment Requirements and Notes

Keep in mind:

- Pair programming is **ALLOWED** for this assignment. You must either register your partnership by 11/22 using this form, or complete the assignment individually.

- The ONLY external libraries you **may** use in your program are:
  ```
  java.io.File
  java.util.Scanner
  ```
  and any relevant exceptions. Use of any other packages (including direct use of `javax.sound.sampled`) is NOT permitted for this assignment.

- You are allowed to define any **local** variables you may need to implement the methods in this specification (inside methods). You are NOT allowed to define any additional instance or static variables beyond those specified in the write-up.

- All test methods must be public static and return boolean values. You may define additional **private** tester methods, and additional **private** helper methods in other classes.

- All classes and methods must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide, and any methods longer than three lines should have at least one inline comment in the body of the method.

- Any source code provided in this specification may be included verbatim in your program without attribution.

- **Run your program locally before you submit to Gradescope**. If it doesn't work on your computer, *it will not work on Gradescope*.

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Academic Conduct Expectations and Advice, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - What do I do about hardware problems?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

1. Getting Started

1. Create a new project in Eclipse, called something like **P08 MusicPlayer300**.
   a. Ensure this project uses Java 17. Select "JavaSE-17" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.

2. Download the **QueueADT.java** file from the assignment page and add it to your project.

3. Create 5 Java source files within that project's src folder:
   a. Song.java
   b. SongNode.java
   c. Playlist.java (the linked queue)
   d. MusicPlayer300.java (MAY contain a main method)
   e. MusicPlayerTester.java (contains a main method)

Additionally, as with our GUI programs, you'll need a jar file.

1.1 Download the AudioUtility jar file

Download the p08core.jar file from the assignment page, which contains an instantiable class that interfaces with the `javax.sound.sampled` library, called **AudioUtility**. If you're interested in the source code, we're happy to provide it on request! The sound library is not the point of this assignment, so we'll do the work for now.

Copy the jar file into your project folder, and refresh the Package Explorer panel in Eclipse. You should see the jar file there. Add it to your build path as in P02 and P05.

1.2 Check your setup

In the Song class, add the `audioClip` data field of type AudioUtility. If you've got your jar file connected properly, this should not cause any errors and you're good to continue with the project.

If you DO see errors pop up at this point, please contact a TA, peer mentor, or instructor for assistance before you continue!

1.3 Download the audio files

Download the audio.zip file from the assignment page **and expand it**; it contains several small MIDI files you can use to test your program. The files named only with a number are very short (6-13 seconds); the files with actual names are MIDI versions of longer songs you may recognize.

Add this expanded folder to your project; it should be in the base project directory and not src or any other sub-folder.

2. Building the Queue

The first coding step is constructing the linked queue our music player is built around. This requires the Song, SongNode, and Playlist classes.

2.1 Song

Referencing the Song class javadocs, write a tester method called `testSongConstructor()` in your tester class:

- Test the constructor with an *invalid* file, like one that doesn't exist or one of the provided .txt files (this should throw an IllegalArgumentException)
- Test a valid file with `toString()` and `getTitle()` and `getArtist()` accessor methods.

Write the constructor and the other three methods in your Song class, and *verify that they work properly* before you continue – constructing a Song with an actual music file isn't something we've done before, and since that's the backbone of our project you'll want to make sure you can do it correctly. **Be aware** – if you're using one of the "real" songs we provided, it can take a couple of seconds to get everything initialized on the back end. Be patient!

The other methods in Song have to do with *playback* – making the audio file actually play or stop playing, and testing whether it is currently running. Write a tester called `testSongPlayback()` to verify that these methods work – and grab some headphones, maybe. **Be aware** – song files can take a second or so to actually start playing. You may want to use `Thread.sleep(1000)` to pause the test method long enough for the computer to actually begin playing the song (and for AudioUtility's `isRunning()` method to return true).

⭐ Your `play()` method should ALSO print "Playing …" with the song's information from `toString()`.

Before you continue, the tester methods you should have written for the Song class are:

- `testSongConstructor()`
- `testSongPlayback()`

2.2 SongNode

The SongNode class is a singly-linked node, similar to the LinkedNode from P07 (except it contains a Song instead of a MultipleChoiceQuestion). Implement it and test it in the same way.

Before you continue, the tester method you should have written for the SongNode class are:

- `testSongNode()`

2.3 Playlist

Playlist is a linked queue that implements the QueueADT interface – if you haven't downloaded QueueADT.java from the assignment page, do so now.

A linked queue is essentially a linked list where the only add operation is adding to the end, and the only remove operation is removing from the front. Additionally, aside from our `toString()` method, the only publicly-accessible node will be the one at the front of the queue (the next one to be removed).

As such, the only tester methods you'll need for your Playlist queue are:

- `testEnqueue()` which should also test the constructor and accessor methods
- `testDequeue()`

3. Writing the MusicPlayer300

Now comes the fun part – creating the actual music player application around your queue. Reference the javadocs as you go; most methods are explained there. The few that need additional details are listed out below!

3.1 getMenu()

The menu for your interactive application should look like this:

> Enter one of the following options:
> [A <filename>] to enqueue a new song file to the end of this playlist
> [F <filename>] to load a new playlist from the given file
> [L] to list all songs in the current playlist
> [P] to start playing ALL songs in the playlist from the beginning

[P -t <Title>] to play all songs in the playlist starting from <Title>
[P -a <Artist>] to start playing only the songs in the playlist by Artist
[N] to play the next song
[Q] to stop playing music and quit the program

3.2 runMusicPlayer300()

This should be the LAST method you write in MusicPlayer300.

If you choose to add a main method to MusicPlayer300, a call to this method will be its only contents. This is the method that begins the interactive menu loop and processes user input. Its only argument is a Scanner hooked up to `System.in`.

This method should loop continuously, and do the following, using other methods from MusicPlayer300 wherever possible:

- Display the menu
- Prompt the user for input (use a "> ")
- Save the next line of input and parse out the option the user has selected:
  - **A** – add a song to the end of the playlist. You will need to further prompt the user for the title and artist of this song, and then add the new song to the playlist.
  - **F** – load in a new playlist from the given file.
  - **L** – display all the songs remaining in the current playlist.
  - **P** – begin playing the songs in the playlist, but first! Check to see if there was a modifier.
    - -t – begin playback at the song with the given title.
    - -a – play only the songs by the given artist (hint: use the `filterPlay` and `filterArtist` data fields for assistance here)
  - **N** – stop the current song and move to the next song in the playlist. Songs should only be dequeued from the playlist when you're ready to move to the next song; otherwise, you won't be able to stop their audio!
  - **Q** – clear out the queue and end the method. Print a "Goodbye!" message.
  - Anything else – print out "I don't know how to do that." and go back to the beginning.

3.3 Sample Run

What follows is what your MusicPlayer300 might look like when you run it. User input is in **green**.

```
Enter one of the following options:
[A <filename>] to enqueue a new song file to the end of this playlist
[F <filename>] to load a new playlist from the given file
[L] to list all songs in the current playlist
[P] to start playing ALL songs in the playlist from the beginning
[P -t <Title>] to play all songs in the playlist starting from <Title>
[P -a <Artist>] to start playing only the songs in the playlist by Artist
```

```
[N] to play the next song
[Q] to stop playing music and quit the program
> P
No songs left :(
[menu]
> F test_playlist.txt
Loading "test1"
Loading "test2"
Loading "test4"
Loading "test3"
[menu]
> L
"test1" (0:6) by hobbes
"test2" (0:6) by hobbes
"test4" (0:13) by yoko kanno
"test3" (0:6) by hobbes
[menu]
> P -a hobbes
Playing "test1" (0:6) by hobbes...
[menu]
> N
Playing "test2" (0:6) by hobbes...
[menu]
> N
Playing "test3" (0:6) by hobbes...
[menu]
> N
No songs left :(
[menu]
> A waterloo.mid
Title: Waterloo
Artist: ABBA
Unable to load that song
[menu]
> A audio/waterloo.mid
Title: Waterloo
Artist: ABBA
[menu]
> P
Playing "Waterloo" (2:45) by ABBA...
[menu]
> Q
Goodbye!
```

* I'm abbreviating the menu here to save space. You should print the whole thing each time.

3.4 Test Methods

Happy Thanksgiving – you do NOT need to write separate test methods for this class. Test that it works by interacting with it! If you have your own audio files, try adding them to the `audio` directory and playing your own music instead of the silly MIDIs I found for you.

Note that all test methods you DO write must use the provided audio files ONLY – these will be the only files available on Gradescope, and we will run your test methods there.

Assignment Submission

Hooray, you've finished this CS 300 programming assignment!

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, submit your source code through Gradescope.

For full credit, please submit ONLY the following files (source code, *not* .class files):

- Song.java
- SongNode.java
- Playlist.java
- MusicPlayer300.java
- MusicPlayerTester.java

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

Copyright Notice