CS150 Lab 8 Report

Oanh Doan

December 3, 2017

1 Introduction

The goal of this lab is to create a binary search tree to store my own type of data. In this case, the data type is song, and the tree resembles a playlist. Each song stores information about the artist, title, year, and genre. The songs in the playlist (or the tree) are organized according to the following order: artist, title, year, and genre. Another part of the lab is to get to know and understand how to use the debugger in BlueJ.

2 Approach

2.1 Design of program

The program contains the following classes:

- Tree interface
- Generic Binary tree abstract class: implements methods to traverse the tree
- Generic Binary Search Tree class: implement methods to add, remove, find max, min
- Song: stores information of a song (title, artist, year, genre), and implement a method to compare song
- PlayList: stores a binary search tree of type Song.
- Controller: create instances of PlayList and call methods such as add songs from an input file and generate an output file of songs in sorted order.

Because the tree classes were tested in lab 6, in this lab, I only created unit tests for *Song* and *PlayList*. Thus, besides the classes above, there are two additional test classes:

- SongTest
- PlayListTest

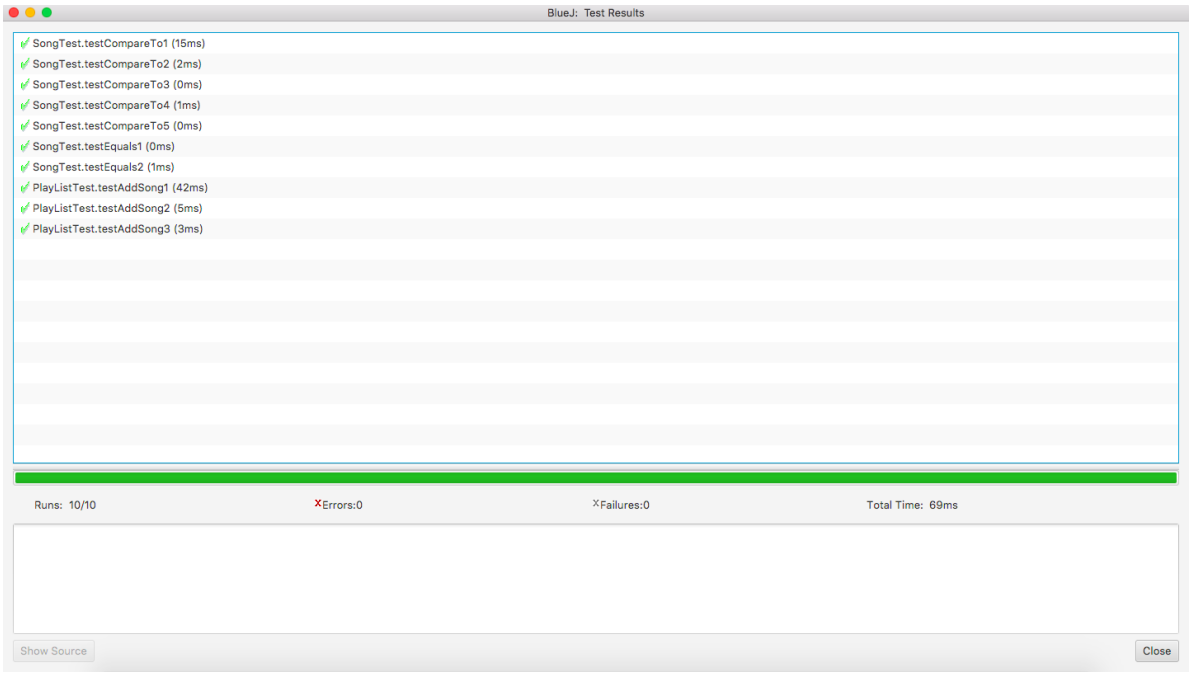

Figure 1. Results of unit tests

2.2 Choice of algorithms and data structures

- ArrayList [5], Array [4]: ArrayList and Array are used to convert the songs in the playlist into a format that can be used by the unit test.
- Comparable [1]: this interface is implemented to make the song comparable, which is necessary to determine the order of a given list of songs.
- Interface [2]: is used as we implement the Tree and Comparable interface.
- Override equals method [3]

3 Methods

In the controller class, I read in an input file then write the songs in sorted order to an output file.

4 Data and analysis

Figure 2 and Figure 3 present the input and output files.

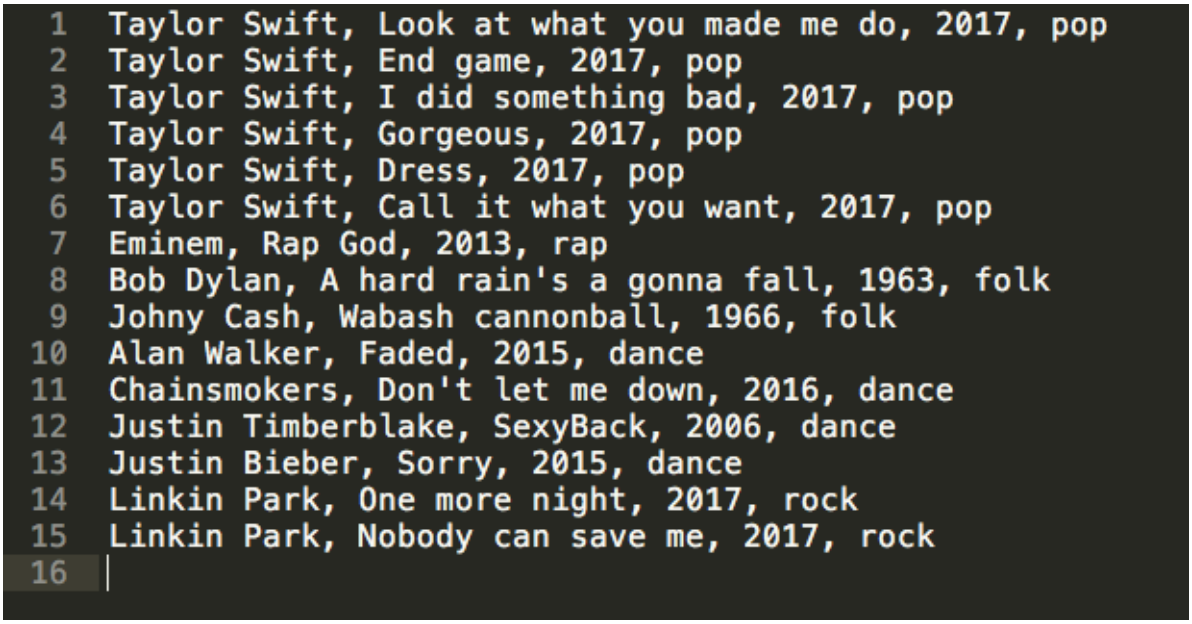

Figure 2. Input

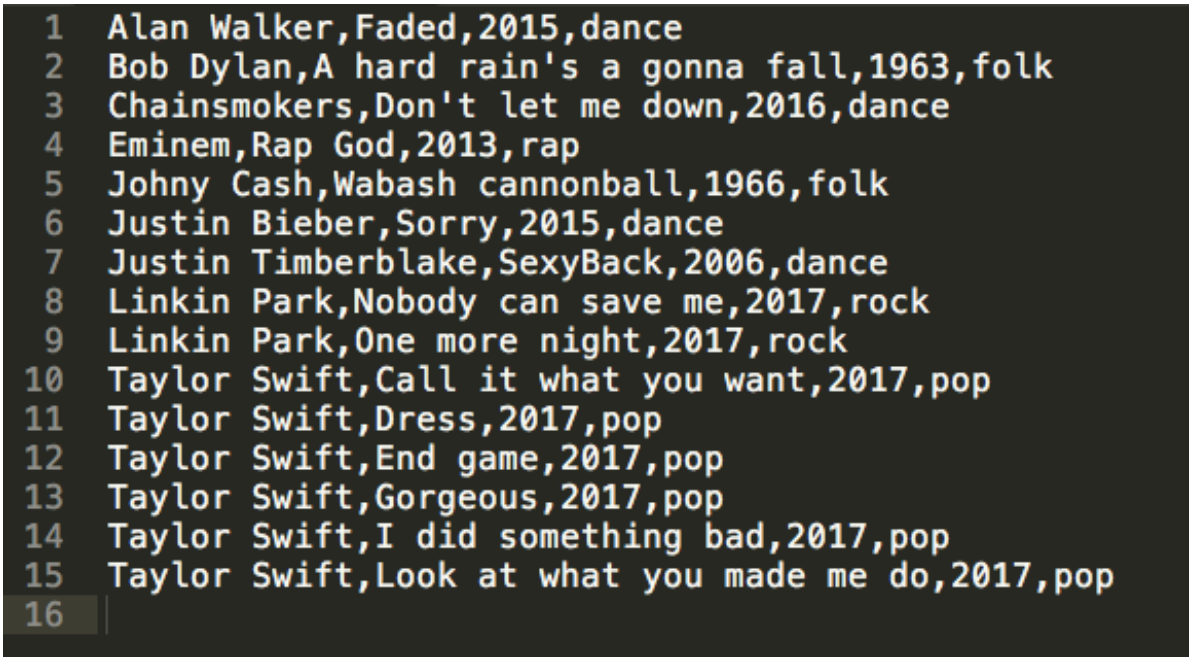

Figure 3. Output

In figure 3, the songs are sorted in the right order. First, the songs are sorted by artists' names. We also notice that the songs by Taylor Swift are very similar to each other. The only difference is the title, which determines the order of the songs in the playlist.

The following figures illustrate how a debugger works.

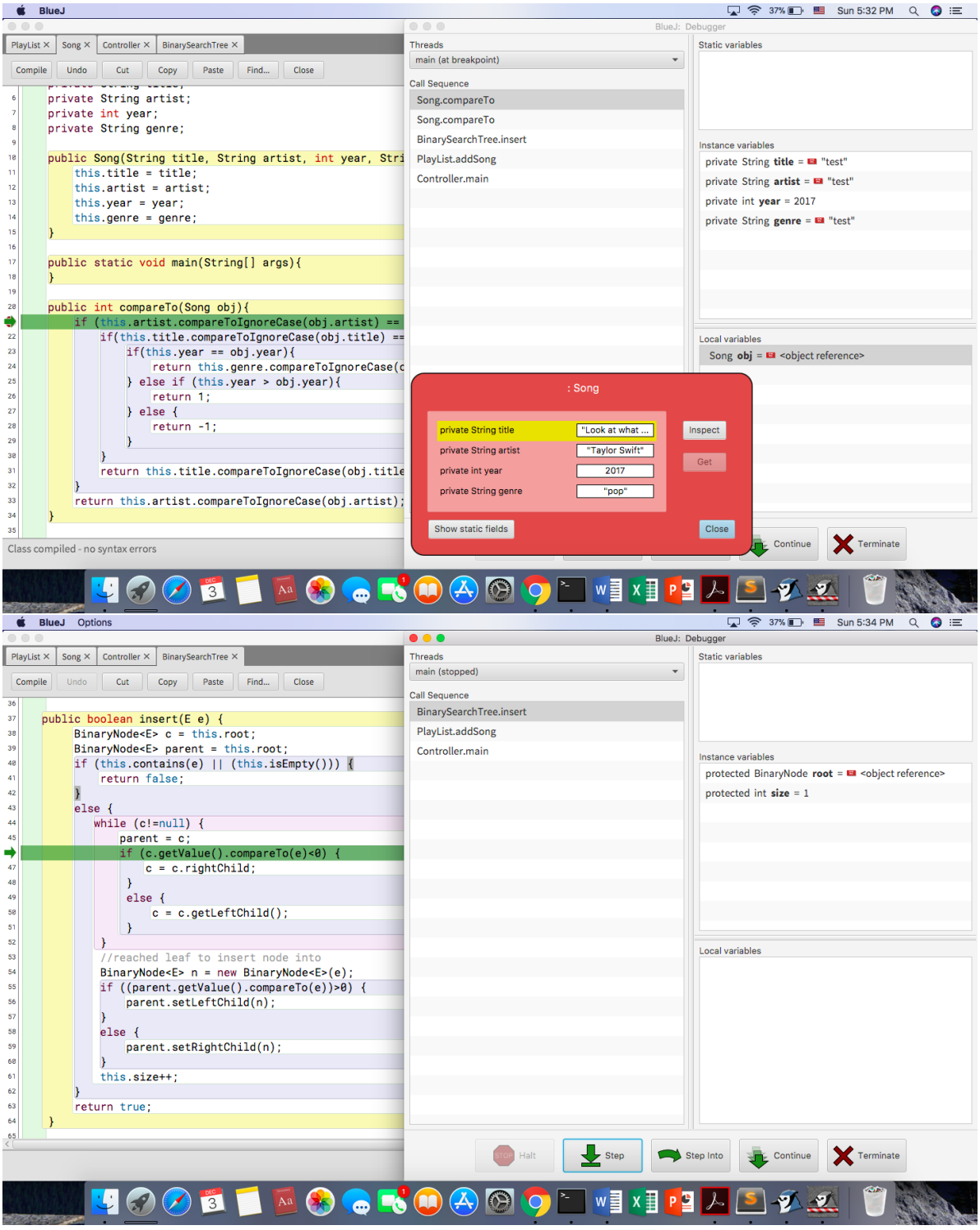

Figure 4. Add first song

When a tree is first created, a song named "test" is automatically set as root of the tree. This song will be deleted after we have finished adding other songs. When the first song is added to the playlist, the following methods are called:

- main() in Controller

- addSong() in PlayList
- insert() in BinarySearchTree
- contains() in BinarySearchTree
- compareTo() in Song

The idea is that before adding a song, we need to check if the song is already in the playlist. To do that, we have to compare it with other songs in the list. Thus, the first two songs that are compared are the first song from our input file and the root of the tree.

After we have successfully added the first song, we continue to add the second song.

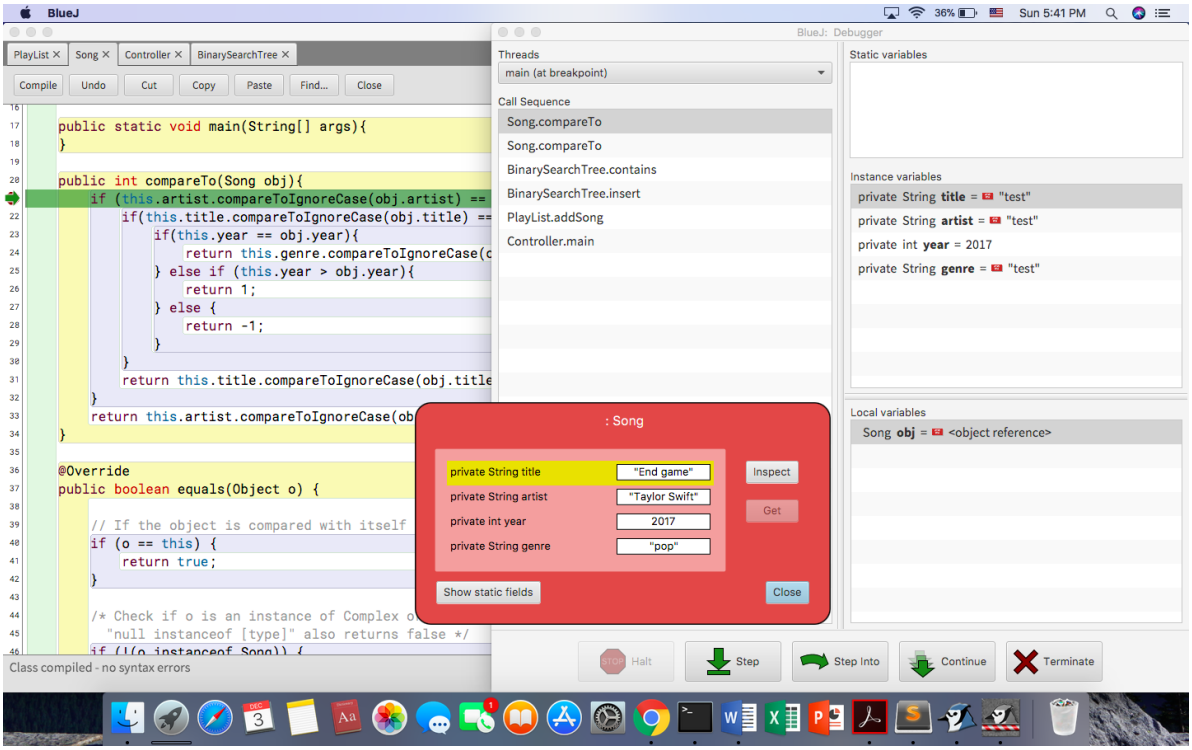

Figure 5. Compare 2^{nd} song with the root

Following the same procedure, to add the second song, we need to compare it with each of the existing songs in the playlist. The first one to be compared with is the root.

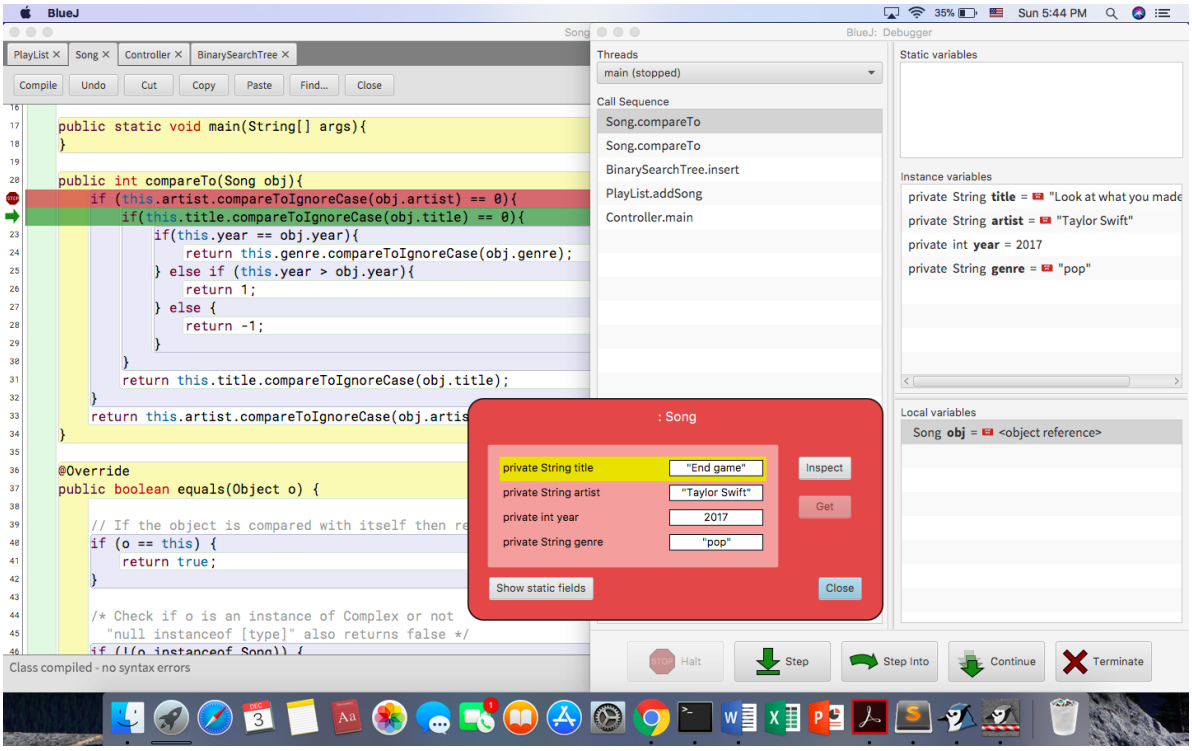

Figure 5. Compare 2^{nd} song with 1^{st} song

Next, we compare the second sond with the first one. We observe that since both songs have the same artists, the debugger jumps to the second if condition in CompareTo method. At this stage, it compares the titles of the two songs. Since the two titles differ, the method returns.

5 Conclusion

The debugger allows us to look into the flow of the program and view the current state, including static, local, and instance variables. This can be very hepful to figure out at which step our codes go wrong should any bugs occur.

References

[1] Oracble. Comparable interface. https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html.

[2] Oracble. Interface. https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html.

[3] Oracble. Overriding equals() method in java. https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html.

[4] Oracle. Arraylist (java platform se 7). https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html.

[5] Oracle. Arraylist (java platform se 8). https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html.