Group Members
Eliot Cole
Kelvin Wilch
Bill Ngoga
Liam McGibbon

Writeup for Music Collection and Artist Search System

This document describes a Java program for managing a collection of songs and searching for artists based on a prefix. The program consists of the following classes:

- **CmpCnt.java:** This class provides a mechanism for counting comparisons made by a comparator object. It has a protected field `cmpCnt` to store the count, a constructor to initialize it to zero, a `resetCmpCnt` method to reset the count, and a `getCmpCnt` method to access the current count.
- **Song.java:** This class represents a song with three private fields: `artist`, `title`, and `lyrics`. It provides accessors for these fields, a `toString` method to format the song information, and a `compareTo` method to compare songs based on artist (primary key) and then title (secondary key). Additionally, it includes a nested class `CmpArtist` that implements `Comparator<Song>` and overrides the `compare` method to compare songs by artist only, incrementing the `cmpCnt` inherited from `CmpCnt` in the process.
- **SearchByArtistPrefix.java:** This class performs a search for songs whose artist names begin with a specified prefix. It uses a `SongCollection` object to access the song data and a `Song.CmpArtist` comparator for comparisons. The `search` method takes an artist prefix as input and performs a binary search on the sorted song array. It then iterates forward and backward from the binary search location to find all matching songs. The method outputs various statistics, including the number of comparisons made during the binary search and list building phases.
- **SongCollection.java:** This class reads a song data file and builds an array of `Song` objects. It uses a `try-catch` block to handle the case where the file is not found. The `SongCollection` constructor reads the file line by line using a `Scanner` and parses the artist, title, and lyrics. It then creates `Song` objects and adds them to an `ArrayList`. Finally, it converts the `ArrayList` to a `Song` array and sorts the array using `Arrays.sort`. The class also provides a `getAllSongs` method to access the song array.

Overall Functionality:

1. The `SongCollection` class reads a song data file and builds a sorted array of `Song` objects.
2. The `SearchByArtistPrefix` class takes an artist prefix as input and uses a `Song.CmpArtist` comparator to perform a binary search on the song array.
3. It then iterates forward and backward from the binary search location to find all matching songs.
4. The class outputs various statistics related to the search complexity, including the number of comparisons made during different phases.

Outputs: (NO Errors)

Group Members
Eliot Cole
Kelvin Wilch
Bill Ngoga
Liam McGibbon

Beatles

```
run:
GUI: Building Song array from ../allSongs.txt
0.445 seconds
GUI: Seaching for plug-ins:
GUI: Found SearchByArtistPrefix, building
 0.004 seconds
Index from binary search is 672
Binary search comparisons: 13
Front found at 672
Comparisons to build the list: 0
Actual complexity is: 13
k: 335
Log n: 13
Theoretical complexity: 348
```

Santana

```
GUI: Building Song array from ../allSongs.txt
0.382 seconds
GUI: Seaching for plug-ins:
GUI: Found SearchByArtistPrefix, building
 0.004 seconds
Index from binary search is 9199
Binary search comparisons: 3
Front found at 9158
Comparisons to build the list: 0
Actual complexity is: 3
k: 71
Log n: 13
Theoretical complexity: 84
```

Arlo

```
GUI: Building Song array from ../allSongs.txt
0.385 seconds
GUI: Seaching for plug-ins:
GUI: Found SearchByArtistPrefix, building
 0.002 seconds
Index from binary search is 577
Binary search comparisons: 14
Front found at 577
Comparisons to build the list: 0
Actual complexity is: 14
k: 4
Log n: 13
Theoretical complexity: 17
```

A

```
GUI: Building Song array from ../allSongs.txt
0.409 seconds
GUI: Seaching for plug-ins:
GUI: Found SearchByArtistPrefix, building
 0.004 seconds
Index from binary search is 0
Binary search comparisons: 13
Front found at 0
Comparisons to build the list: 0
Actual complexity is: 13
k: 581
Log n: 13
Theoretical complexity: 594
```

Group Members
Eliot Cole
Kelvin Wilch
Bill Ngoga
Liam McGibbon
Z

GUI: Building Song array from ../allSongs.txt
0.396 seconds
GUI: Seaching for plug-ins:
GUI: Found SearchByArtistPrefix, building
0.004 seconds
Index from binary search is 10374
Binary search comparisons: 14
Front found at 10374
Comparisons to build the list: 0
Actual complexity is: 14
k: 140
Log n: 13
Theoretical complexity: 153

X
GUI: Building Song array from ../allSongs.txt
0.385 seconds
GUI: Seaching for plug-ins:
GUI: Found SearchByArtistPrefix, building
0.003 seconds
Index from binary search is 10312
Binary search comparisons: 14
Front found at 10312
Comparisons to build the list: 0
Actual complexity is: 14
k: 0
Log n: 13
Theoretical complexity: 13

- Does your search meet the O(K +log2N) time goal? Yes Explain?

the search meets the expected time complexity of **$O(K + \log_2 N)$** as the total number of operations performed is consistent with the theoretical goal for every single search.