CSCI-290 *Project Part 1*

Parsing the Data File

Objectives: Writing Java code. Using Java library functions for ArrayList. Implementing CompareTo() from the Java Interface Comparable. Sorting.

The 10 programming projects for this class will build upon each other to produce an application for efficiently performing a variety of searches on a large database of song lyrics. In this first assignment you will be reading in and storing the song database. Later assignments will build and evaluate other data structures and perform a variety of searches.

Data Files and starting code for each project will be provided on Brightspace.

Setting up your project

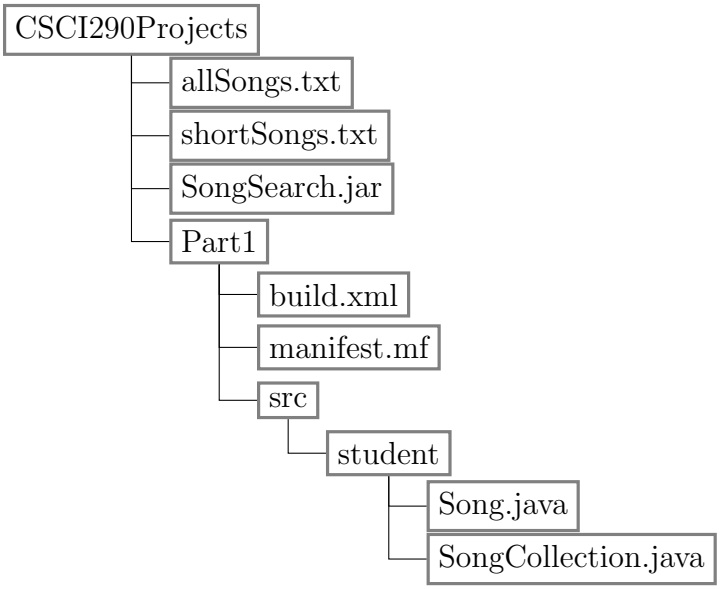
1. Create a folder called CSCI290Projects in Windows Explorer or Finder.
2. Create a project called Part1 using Netbeans inside the CSCI290Projects folder. Make sure that **Java with Ant** is selected. Uncheck the Create Main Class checkbox. We will build a new project for each part so that you will always have the last one that worked!
3. This will not create a package so you will have to do that manually. Right-click on Source Packages. Choose New – Java Package – name the package **student**. Notice the lowercase package name!!

For part 1, there is a zip archive with the following files:

- shortSongs.txt - 6 short songs, use while debugging all parts of the project
- allSongs.txt - 10514 songs used for all parts of the project
- Song.java - starting code for a song class
- SongCollection.java - starting code for reading in the song file
- SongSearch.jar - a Graphical User Interface used for all parts of the project

The two data files and the GUI will be used for all parts so it makes sense to put them where we can access them for all 10 parts. I will show you how to "path" them. It's very important that everyone follow these instructions as you will be sending bare files to me... I will be placing them into a project to test them so your project MUST be formatted like mine. Please DO NOT ZIP anything you send this semester until Part 10!

These files should be distributed as follows:



The song files are formatted as follows:

```
ARTIST="Mildred Hill and Patty Hill"
TITLE="Happy Birthday To You"
LYRICS=" Happy birthday to you
Happy birthday to you
Happy birthday dear _____
Happy birthday to you
"
ARTIST="unknown"
TITLE="Pop Goes The Weasel"
LYRICS="All around the mulberry bush
The monkey chased the weasel.
The monkey thought 'twas all in good fun
Pop! goes the weasel

A penny for a spool of thread,
A penny for a needle-
```

...

Task 1: Finish the class Song which holds three strings (artist, title, and lyrics) for a song. Part of the documentation for each Method MUST include the author(s). Any changes made later must also be documented with a revised by: line that also explains when the change was made, what was changed, and why.

1. Add private fields and implement the constructor.

2. Implement methods: getArtist(), getTitle(), and getLyrics()
3. Implement a toString() method that returns a string of the form: artist, "title"
4. Implement the compareTo() method that will compare songs alphabetically by artist and title. This comparison should be case insensitive. Hint: there is a String method: compareToIgnoreCase().
5. **Unit Testing:** I expect you to follow a testing methodology whereby each class that you create will contain a main() method that can be used to test the methods of that class. These can be relatively simple, but they should ensure you that this class works correctly. I have already written a testing method for the Song class. Just run the Song class to test it. This can be done through the Project Properties on the Run dialog. Set Song.java as the main class or right-click the Song.java source code window and choose Run File from the menu.

Task 2: Complete the class SongCollection.
It will read in a file of song lyrics and create an array of Song objects. It is actually easiest to create a local ArrayList and when you are finished reading in the file, create the array songs (already declared!!) the size of the ArrayList and then use toArray to create the array.

1. It should have a constructor that takes as an argument the name of a data file. It should read the file, build an array of Songs, and sort the array.
2. I suggest reading the file line by line and then just using string functions to verify and extract the pieces of the songs. (I used string methods: startsWith(), indexOf(), lastIndexOf(), substring(). I also use a StringBuilder to concatenate the lyrics into a single string. Don't forget to append the newline characters between lines. It's also possible to read in the entire file using the double quotes as the delimiters and then parse the resulting array of strings. The latter is simpler than the former since the lyrics end up as a single string that includes all of the line feeds that should be part of it.
3. You should handle any exceptions, don't just re-throw them because there is no one else to handle them!
4. Store the input as a simple java array of Songs. This is already declared.
5. I suggest that you use an ArrayList<Song>while reading in the songs, and then create the array of the right size (length of the list) and convert it to a plain array using the toArray(Song[] a) method.
6. You have been given a method: public Song[] getAllSongs() that simply returns the sorted song array. As you implement the various searches in future assignments, they will call this method to get this array of songs as a data source for building more powerful data structures. The file itself should only be read once!
7. Testing: Finish the main() testing method. It takes in a song file name as a command line argument and then creates an instance of the SongCollection class. Add code to print the total number of songs as well as the Artist and Title of the lesser of the first 10 songs or the length of the array. This single line will do the trick for an array called list:

```
Stream.of(list).limit(10).forEach(System.out::println);
```

```
Total songs =  10514, first songs:
Aerosmith   "Adam's Apple"
Aerosmith   "Ain't Enough"
Aerosmith   "Ain't that a Bitch"
Aerosmith   "All Your Love"
Aerosmith   "Amazing"
Aerosmith   "Angel"
Aerosmith   "Angel's Eye"
Aerosmith   "Animal Crackers"
Aerosmith   "Attitude Adjustment"
Aerosmith   "Avant Garden"
```

8. To run a program in NetBeans using different main classes and different data files, you will need to set up a run configuration:
   - Select: Project in the Projects Explorer pane of NetBeans
   - Choose File - Project Properties (Project)
   - Click run in the Categories pane.
   - Click the [Browse] button next to the text box for Main Class and choose SongCollection.java.
   - Click in the Arguments text box and put the name of the data file that you want to use for this run (like ../shortSongs.txt or ../allSongs.txt). The path (../) is necessary for the program to find the files.
   - Now when you click the green Run arrow those choices will apply. Change the arguments as needed.

Setting up NetBeans to run the GUI from the jar file:

- Select: Project in the Projects Explorer pane of NetBeans
- Choose File - Project Properties (Project)
- Click Libraries in the Categories pane.
- Click the Run tab
- In Version 8.x
  - Click the [Add Jar/Folder] button on the right
- In Version 12.x
  - Click the + sign next to Classpath
  - Choose Add JAR/Folder
- Browse to the SongSearch.jar file and select it
- Click the radio button next to Relative Path
- Click [Open] button at the bottom.
- Now Click Run in the Categories pane.

- In the text box for the Main Class type songSearch.SongSearchGUI (case is important here!!)
- Now when you click the green Run arrow the GUI will run with which ever file you set in the Arguments text box. At the moment, only the [Browse] button on the GUI will be active.

Task 3: Test that your code works with the graphical user interface in SongSearch.jar. The GUI starts by calling your SongCollection() method to read in the song file. As you complete each future assignment, the GUI will automatically recognize your new search classes and plug them into the interface. Dr Robert Boothe at USM wrote the GUI. If you run into a problem, let me see the bad behavior. So far the only search available will be "Browse".

If you have followed the specifications above (correct class names, method names, and return types) then your code will plug into the GUI without any changes on your part.

What to turn in:
For each assignment you will create a write up. This can be created in any format, but printed to a pdf to turn it in. I expect a well written and organized report using proper English sentences, grammar and spelling. Explain your answers to the questions. You will need to copy and paste output to the console from your program into some write up documents. DO NOT USE SCREEN CAPTURES when asked for the console output. Others will require screen captures pasted in. Use the Windows Snipping tool to grab just what is asked for. (I don't know what the Apple version is). You do not need to turn in a printed copy of anything. You will be handing in multiple single files nothing zipped!

Written part: (named Part1Writeup.pdf)

1. Turn in just one project report per group. All group members should be listed first.
2. The testing output from the Song class tests (copy & paste from console window).
3. The testing output from your SongCollection class using ../allSongs.txt (copy & paste from console window).
4. Run the SongSearch graphical user interface. Click Browse and then click on the first song to show its lyrics. Paste in a screen shot of the GUI.
5. It is your responsibility to test your code. Explain any incomplete parts, or any known bugs. I take off fewer points if you are aware of a bug rather than oblivious to it. If there are none say so!

Electronic submission:

- I will create groups and put you into them. Anyone in the group can upload files.
- I will run your unit testing code, so each class must implement the specified unit testing. Points will be deducted if I have to do any coding to unit test a class.
- Upload to Brightspace Part1Writeup.pdf, Song.java, SongCollection.java as separate files. Do not zip! Make sure that all group members are listed on the write up!!! You

are not allowed to work alone on any of the programming this semester.

Grading Criteria:

Written part. Clearly organized!	10%	Song class basics	20%
SongCollection: file reading & parsing,	30%	Works with the GUI	10%
Song compareTo() and sorting	20%	Song Collection testing routine	10%

I expect your programs to be properly commented and use good style. Points may be deducted for egregious disregard of these matters. Every method in every class must have complete documentation including authorship. A full an complete revision history should be at the top of every class file! Revision comments apply only to the file they are in. Do not reference work done in other files.

An acceptable write up is shown below. I used a very small font and sized the screen capture such that everything would fit on one page for display in this document. You should not do that. Make sure everything is easily readable without a magnifying glass.

Part 1 - Parsing the Data File

Group 3: Dot Matrix and Phil A Dendron

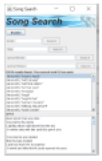
Task 1 output from unit test of Song

```
testing getArtist: Professor B
testing getTitle: Small Steps
testing getLyrics:
Write your programs in small steps
small steps, small steps
Write your programs in small steps
Test and debug every step of the way.

testing toString:

Song 1: Professor B, "Small Steps"
Song 2: Brian Dill, "Ode to Bobby B"
Song 3: Professor B, "Debugger Love"
testing compareTo:
Song1 vs Song2 = 14
Song2 vs Song1 = -14
Song1 vs Song3 = 15
Song3 vs Song1 = -15
Song1 vs Song1 = 0
```

Task 2 output from unit test of SongCollection

```
Total songs = 10514, first songs:
Aerosmith, "Adam's Apple"
Aerosmith, "Ain't Enough"
Aerosmith, "Ain't that a Bitch"
Aerosmith, "All Your Love"
Aerosmith, "Amazing"
Aerosmith, "Angel"
Aerosmith, "Angel's Eye"
Aerosmith, "Animal Crackers"
Aerosmith, "Attitude Adjustment"
Aerosmith, "Avant Garden"
```

Task 3 output (screen capture of the GUI as described)

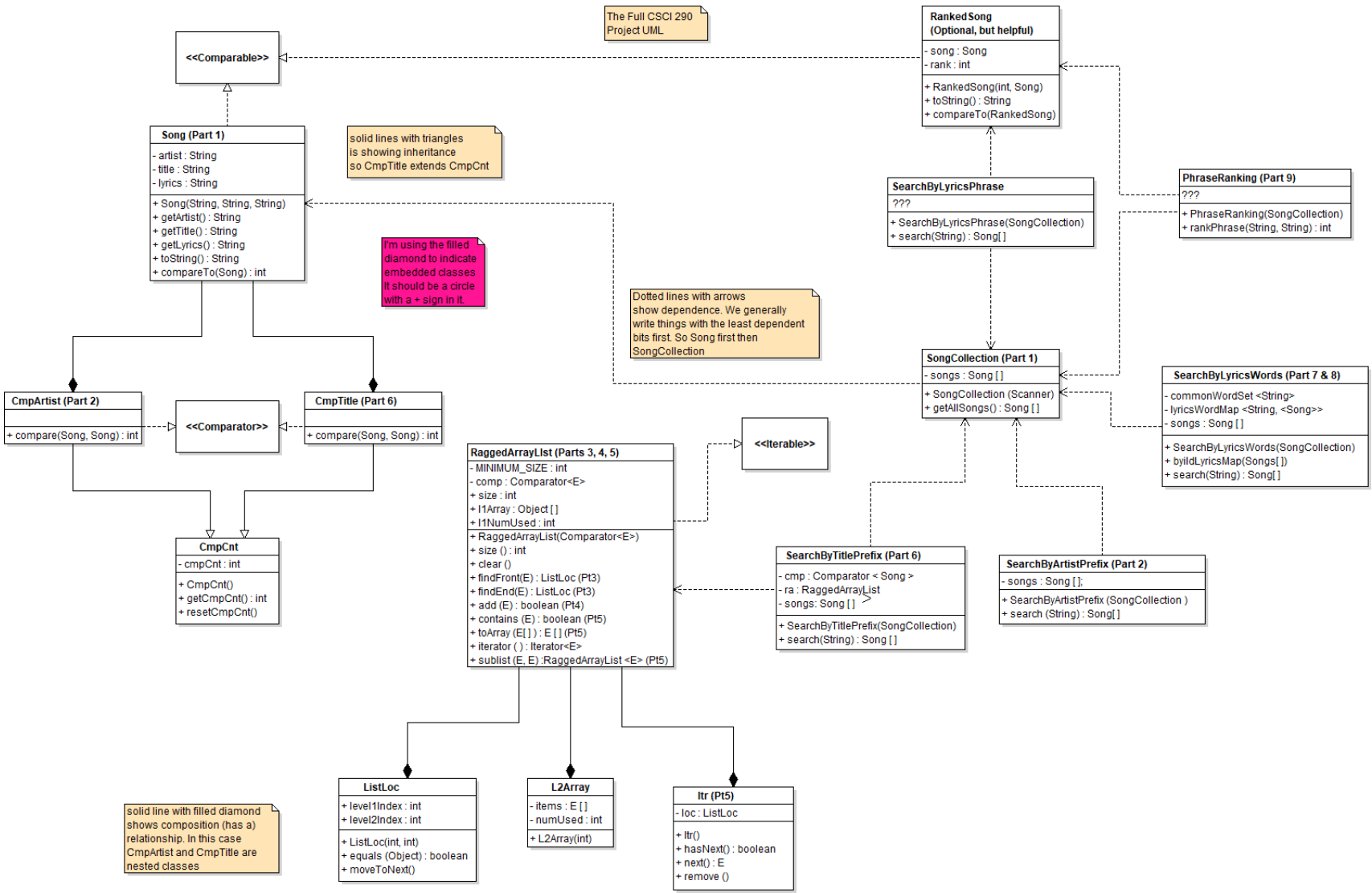


Incomplete Parts: None

Known Bugs: None

Do fill out the evaluation for yourself and your partner. Failure to submit an evaluation will cost you 10 points on the part of the project!

The full UML for this semester's project is on the following page. It shows all inheritance, composition, implementation and dependence relationships.


The Full CSCI 290 Project UML
<<Comparable>>
RankedSong (Optional, but helpful)
- song : Song
- rank : int
+ RankedSong(int, Song)
+ toString() : String
+ compareTo(RankedSong)
Song (Part 1)
- artist : String
- title : String
- lyrics : String
+ Song(String, String, String)
+ getArtist() : String
+ getTitle() : String
+ getLyrics() : String
+ toString() : String
+ compareTo(Song) : int
solid lines with triangles is showing inheritance so CmpTitle extends CmpCnt
SearchByLyricsPhrase
???
+ SearchByLyricsPhrase(SongCollection)
+ search(String) : Song[ ]
PhraseRanking (Part 9)
???
+ PhraseRanking(SongCollection)
+ rankPhrase(String, String) : int
I'm using the filled diamond to indicate embedded classes It should be a circle with a + sign in it.
Dotted lines with arrows show dependence. We generally write things with the least dependent bits first. So Song first then SongCollection
SongCollection (Part 1)
- songs : Song [ ]
+ SongCollection (Scanner)
+ getAllSongs() : Song [ ]
SearchByLyricsWords (Part 7 & 8)
- commonWordSet <String>
- lyricsWordMap <String, <Song>>
- songs : Song [ ]
+ SearchByLyricsWords(SongCollection)
+ byildLyricsMap(Songs[ ])
+ search(String) : Song[ ]
CmpArtist (Part 2)
+ compare(Song, Song) : int
<<Comparator>>
CmpTitle (Part 6)
+ compare(Song, Song) : int
<<Iterable>>
RaggedArrayList (Parts 3, 4, 5)
- MINIMUM_SIZE : int
- comp : Comparator<E>
+ size : int
+ l1Array : Object [ ]
+ l1NumUsed : int
+ RaggedArrayList(Comparator<E>)
+ size () : int
+ clear ()
+ findFront(E) : ListLoc (Pt3)
+ findEnd(E) : ListLoc (Pt3)
+ add (E) : boolean (Pt4)
+ contains (E) : boolean (Pt5)
+ toArray (E[ ]) : E [ ] (Pt5)
+ iterator ( ) : Iterator<E>
+ sublist (E, E) :RaggedArrayList <E> (Pt5)
CmpCnt
- cmpCnt : int
+ CmpCnt()
+ getCmpCnt() : int
+ resetCmpCnt()
SearchByTitlePrefix (Part 6)
- cmp : Comparator < Song >
- ra : RaggedArrayList
- songs: Song [ ]
+ SearchByTitlePrefix(SongCollection)
+ search(String) : Song [ ]
SearchByArtistPrefix (Part 2)
- songs : Song [ ];
+ SearchByArtistPrefix (SongCollection)
+ search (String) : Song[ ]
solid line with filled diamond shows composition (has a) relationship. In this case CmpArtist and CmpTitle are nested classes
ListLoc
+ level1Index : int
+ level2Index : int
+ ListLoc(int, int)
+ equals (Object) : boolean
+ moveToNext()
L2Array
- items : E [ ]
- numUsed : int
+ L2Array(int)
Itr (Pt5)
- loc : ListLoc
+ Itr()
+ hasNext() : boolean
+ next() : E
+ remove ()