CS 158 Lab 23 April 29, 2010

Playlist Similarity

1. Create a new project called Playlist Similarity. Add the following classes and protected fields:

Song	**Artist**	**Genre**
protected String title;	protected String artist;	protected String genre;
protected String artist;	protected int occurrences;	protected int occurrences;
protected String genre;	protected int playCount;	protected int playCount;
protected int occurrences;		
protected int playCount;		

2. For each class, create a constructor that initializes all of the fields with arguments that are in the same order as given above. You may want to add **print** methods that print on one line all of the fields for a **Song**, **Artist** or **Genre**, separated by commas.

3. In order to sort songs easily, we need to be able to compare **Song** objects. So have **Song** implement the **Comparable** interface. You will have to provide a correct implementation of the **compareTo** method. So if songA and songB are **Song** objects, **songA.compareTo(songB)** should return -1 if songA is less than songB, return 1 if songA is greater than songB, and 0 if songA is the same as songB. To be "less than" means that the title of songA is less than that of songB in lexicographic order, or if the titles are the same, then the artist of songA is less than the artist of songB. The songs are equal if and only if both the title and the artist match. Remember to use the **compareTo** method of the String class to do the String comparisons. Include a JUnit test for this method.

4. Add a **PlayList** class. It should have an **ArrayList** for each of the three classes above, called **songList**, **artistList**, and **genreList**. Add a string variable called **name**. Add a constructor that instantiates each of the lists, but puts nothing into them. Add a setter and getter for the **name** field. Add a method called **sortSongList** that takes no arguments and returns nothing, but sorts the **ArrayList** of songs by calling the **sort** method of the **Collections** class, supplying as an argument the **ArrayList** of **Song** objects. Create a good JUnit test to make sure the sorting is working right. Go back to the **Artist** and **Genre** classes and make them implement the **Comparable** interface. For those classes, the comparison is simpler: **Artist** objects only depend on the **artist** field, and **Genre** objects only depend on the **genre** field for ordering purposes. Then in the **PlayList** class add methods **sortArtistList** and **sortGenreList** that sort the lists.

5. On Blackboard under Content, you should find a **Reader** class that you can use to read from the xml version of a playlist. Copy it into your project as a new class. Notice that the constructor takes an argument that is the name of a file, and then does all the work of reading the entire playlist into a private **ArrayList** object. You can use the methods **hasNextSong** and **nextSong** to obtain **Song** objects that you can put into your **songList**. You should also find a folder that contains playlists in xml format that were contributed. You can download one or more of those into the project folder so they can be read easily. Create a **Main** class (if you don't already have one) that has a **main** method. In the **main** method, create a **PlayList** object, and use a **JOptionPane.showInputDialog** to obtain the name of a playlist file. Set the **name** field of the **PlayList** object to the name of the file.

Add a **Reader** object, and give its constructor the string that you stored the file name in. Then for each song in the **Reader**, add it to the **songList** of your **PlayList** object, setting the **occurrences** field to 1 for each song. Once all the songs are in your list, sort the song list. Print it out to see that it looks right.

6. In the **PlayList** class, create a method called **removeDuplicateSongs** that takes no arguments and returns nothing. Once the song list is sorted, it is easy to remove duplicate songs. Recall that two song objects are the same if their title and artist are the same, and our **compareTo** method already will tell us if they are the same by returning 0. So go through the song list comparing consecutive elements. If they are the same song, add the two occurrence numbers together, store that number in the first song, and remove the second song from the list. Be careful not to run off the end of the **ArrayList**. Write a JUnit test for this method. Remember that the data must be sorted b efore you run **removeDuplicateSongs**. (By the way, I noticed that at least one of the playlists contributed contains duplicate songs.)

7. In the **PlayList** class, add a method called **buildOtherLists**. For each song in the song list, create an **Artist** and **Genre** object (making sure to put the appropriate data into them), and add them to the appropriate lists. Add methods called **removeDuplicateArtists** and **removeDuplicateGenres** that work the same as what you did in step 6. The occurrence numbers will be much larger here. In the **Main** class **main** method, add the code to create the lists, sort them, and remove duplicates.

8. Modify the **main** method so that it creates an **ArrayList** of **PlayList** objects. Create a loop that asks the user if they have more playlists to enter, and if so, prompts the user for a file name, and reads the playlist into a new **PlayList** object's song list, and sorts and removes duplicates and creates the other lists as described above.

9. In the **Main** class, create a static method called **measure** that takes two **PlayList** arguments and returns an integer. This integer will be a measure of how similar the playlists are. The bigger the value, the more similarity. Use the following formula: (number of songs in common in the two playlists times 100) plus (the number of artists in common times 10) plus the number of genres in common. In the **main** method, create a nested loop that calls **measure** for every possible pair and prints the results.

Due Wednesday, May 5 for extra credit.