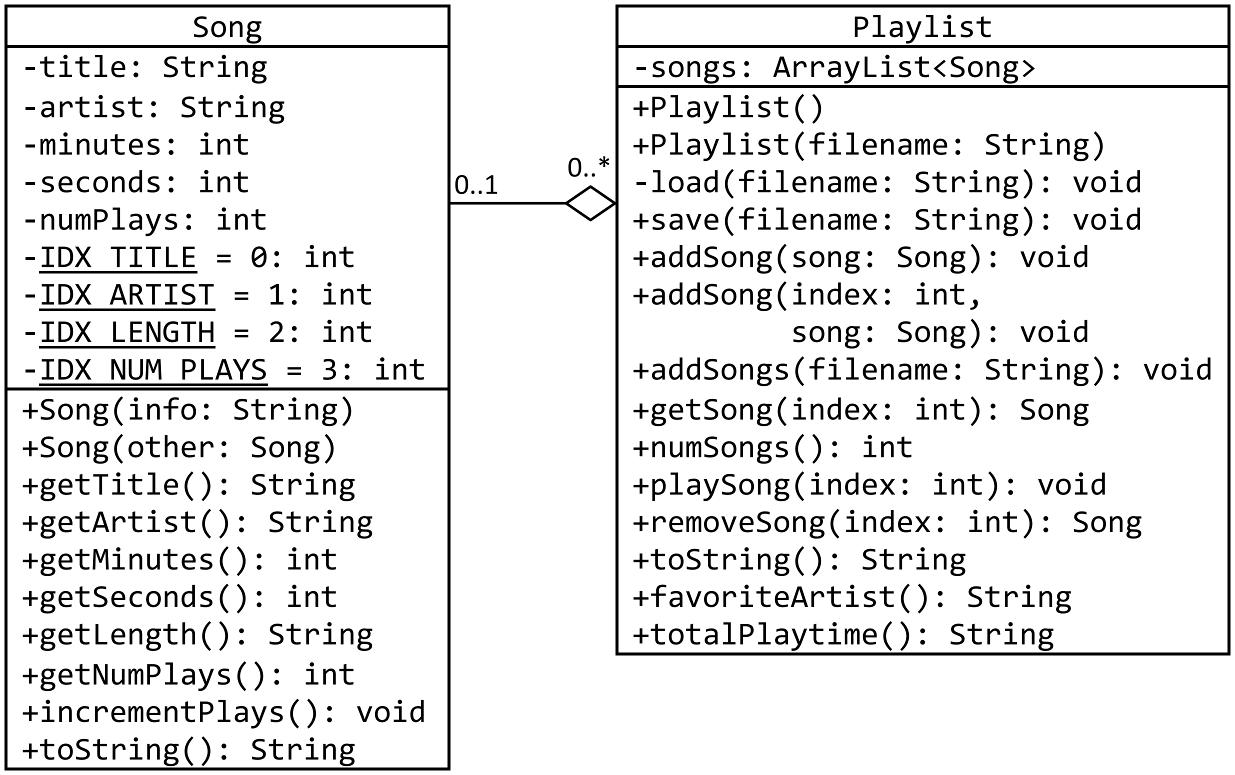
Lab 3: Reading and Writing Playlists

Due: Tuesday, Feb. 4 at 11:59 PM

Description: In our earlier programs, the states of the objects we constructed at runtime were lost when the program stopped. For many applications, this is an unacceptable limitation. A word processor, for instance, wouldn't be very useful if we lost our documents when we turned off our computer.

For our third lab, we will write a pair of classes that do not have this limitation: Song and Playlist. Playlist objects store a list of Song objects that can be modified in various ways. Most importantly, the state of a Playlist can be saved and loaded from a text file, so we can keep track of our music collection even after our program stops!

Class Diagram:[1]



Song Class: Each Song has a title, artist, length, and play count. The length is stored in two fields: minutes and seconds.

Songs can be constructed from either an info String or another Song object. Info Strings have the following format:

```
"<title>,<artist>,<minutes>:<seconds>"
```

[1] The numbers "0..1" on the aggregation line indicate that each Song belongs to either 0 or 1 Playlist. Likewise, "0..*" indicates that each Playlist has 0 or more Songs.

The values are separated by commas with no spaces. The index of each value is stored in a static variable. For instance, IDX_ARTIST stores the index of “<artist>”, which is the value of the artist field.

Optionally, an info String may have a fourth value that specifies the number of times a song has been played:

```
"<title>,<artist>,<minutes>:<seconds>,<numPlays>"
```

If this value is absent, initialize numPlays to 0.

To illustrate this format, consider the following info String:

```
"Ten Years Gone,Led Zeppelin,6:31"
```

A Song constructed from this String stores “Ten Years Gone” in the title field, “Led Zeppelin” in the artist field, 6 in the minutes field, and 31 in the seconds field. The info String does not have a fourth value, so the object stores 0 in the numPlays field.

Below are descriptions of some methods of the Song class:

- Song(Song other): This is known as a “copy constructor.” It creates a new Song that is a copy of an existing Song. That is, it initializes the fields of the new Song to the values stored in the given Song. The utility of this method will become clear when writing the Playlist class.
- getLength(): Return the song length in the format of an info String:

  ```
  "<minutes>:<seconds>"
  ```

  If the number of seconds is less than 10, pad the value with a leading zero so it has two digits. The easiest way to do this is with the format method of the String class. Use the format String “%02d”, which indicates that the value should be a decimal integer with at least two digits.
- incrementPlays(): Increase numPlays by 1. Note that this is a mutator, which implies that Songs are mutable.
- toString(): Return an info String with the values of the fields. If numPlays is 0, do not include it in the output. (Hint: Use getLength() to avoid duplicating code.)

Playlist Class: Each Playlist stores a reference to an ArrayList of Songs. We want to protect these Songs so they can only be modified by methods of the Playlist class. In order to accomplish this, any Song must be copied before it is added or returned from a Playlist. Otherwise, the calling method will have a reference to private data.[2]

Below are descriptions of the methods of the Playlist class:

- Playlist(): Construct an empty Playlist.
- Playlist(String filename): Construct a Playlist from a file of info Strings. The Playlist should contain a Song for every line of the file, and the order of the Songs should match the file.

[2] This was not a concern in Lab 2 because, unlike Songs, Fish are immutable.

- Load(String filename): Read a file of info Strings with the given name. For each line of the file, create a Song and add it to the end of the Playlist. This method is intended to be a helper method for Playlist(String filename) and addSongs(String filename).
- save(String filename): Save the output of toString() to a file with the given name. (Overwrite the contents of the file if it already exists.)
- addSong(Song song): Add a copy of the given Song to the end of the Playlist.
- addSong(int index, Song song): Add a copy of the given Song to the Playlist at the given index.
- addSongs(String filename): A public version of the method Load(String filename).
- getSong(int index): Return a copy of the Song with the given index.
- numSongs(): Return the number of Songs in the Playlist.
- playSong(int index): Increment the number of plays of the Song with the given index.
- removeSong(int index): Remove and return the Song with the given index. (It is not necessary to return a copy.)
- toString(): Return a String with the values of the fields of every Song in the Playlist. Create the String by calling toString() on each Song and joining the output with newline characters. (Do not terminate the String with a newline character.)

  For example, if a Playlist contains Songs constructed with these info Strings:

  ```
  "So What,Miles Davis,9:22"
  "Firth of Fifth,Genesis,9:35"
  ```

  Then calling toString() on the Playlist should return this String:

  ```
  "So What,Miles Davis,9:22\nFirth of Fifth,Genesis,9:35"
  ```

- favoriteArtist(): Return the artist that appears most frequently in the Playlist. If the Playlist is empty, return null.
- totalPlaytime(): Return the total length of all the Songs in the following format:

  ```
  "<hours>:<minutes>:<seconds>"
  ```

  The number of minutes and seconds should each be a number between 0 and 59. If the number of hours is 0, use this format instead:

  ```
  "<minutes>:<seconds>"
  ```

  Pad the number of seconds with a leading zero if it is less than 10. If the number of hours is greater than 0, pad the number of minutes as well.

Reading and Writing: In CS 1323 and 1324, we used the Scanner and PrintWriter classes to read and write files. You are welcome to use those classes here, but the BufferedReader and BufferedWriter classes are lightweight alternatives.[3]

Objects of these classes can be constructed with the following statements:

```
BufferedReader br = new BufferedReader(new FileReader(filename));
BufferedWriter bw = new BufferedWriter(new FileWriter(filename));
```

Call readLine() on the BufferedReader to read the next line of the file. The method returns null after it reads the last line. Call write(String str) on the BufferedWriter to write the given String to the file. After you are done reading or writing, call the close() method.

Any method that uses a BufferedReader or BufferedWriter must be able to throw an input-output exception. Import java.io.IOException and add "throws IOException" after the parameter list.

Collaboration: If you collaborate with other students or use online resources for help, cite the names of your collaborators or the URLs of your sources in comments in your code.

Submission: Submit your code to the Zyante website for automated testing. We highly recommend that you write some tests of your own before doing this.

[3] The Scanner class, in particular, is overkill for this program because the info Strings are parsed by Song(String info), rather than by Load(String filename). The latter method simply reads each line of a file.