CS 445: Data Structures
Fall 2017

Assignment 4

Assigned: Thursday, November 02 **Due:** Thursday, November 16 11:59 PM

---

1 Motivation

Imagine that you are starting a streaming radio business. Partnerships have been established to provide you with audio content, and you will stream that content to users. You plan to use your knowledge of what songs your users like/dislike to make effective suggestions for new songs that they will like.

In this assignment, you will consider such a streaming radio service. You will be designing an abstract data type (ADT) that stores the service's music library and users' ratings of songs. This ADT represents a data structure that will be used by both client applications and for administrative uses (e.g., adding new songs to the library).

Once you complete this ADT, you could provide it to the programmers of the client applications, and they will be able to develop applications based on this specification (much like you were tasked with doing in A1). On the other hand, you can also distribute the ADT to the engineers in charge of implementing the functionality of the data structure. Thus, these two parties can work in parallel on their respective components, which can work together by virtue of both following the "contract" that your ADT provides.

2 Provided files

First, carefully read the provided files. You can find these files on Pitt Box in a folder named `cs445-a4-abc123`, where `abc123` is your Pitt username.

- The `StreamingRadio` interface is a skeleton of your ADT. Many details will need to be filled in (see Tasks).
- The `User`, `Song`, and `Station` interfaces are provided as types for your methods' parameters and/or return types.
- The `doc/` directory contains Javadoc-style documentation of the provided code (open `index.html`).

3 Tasks

You must write an abstract data type as a Java interface. You will need to declare several abstract methods that the data structure would need to support. You should consider all possible corner cases in your specification, and describe what the class should do in all

cases. A client that reads your ADT should know all the things that could go wrong, and how they will be handled. For some of these corner cases, you will need to make a design decision regarding the best way to handle them.

> **Note: You will not be implementing this data structure.** You do not need to consider implementation-level details, since you are only writing the ADT. This means that you must specify the full set of details that a *client* needs to know in order to use the data structure, but *not* the details of how the operations will be carried out or how the data will be organized in memory. The implementation details should be left up to the implementers.

3.1 Helper types

When developing your ADT, you can assume the following types exist.

- `User`: Represents a user.
- `Song`: Represents a song.
- `Station`: Represents a collection of Songs, a playlist for a radio station.

You do not need to make any assumptions, nor write any code, regarding the details of these data structures. They are provided to help you specify (parameter and return) types correctly . Interfaces representing these types are provided, but do not declare any methods, since their details are not needed in this assignment. You can assume that these interfaces (and their implementations) would be developed separately.

3.2 Abstract methods to declare

You will need to declare each of the following abstract methods in your `StreamingRadio` interface.

- `addSong`: Adds a new song to the system.
- `removeSong`: Removes an existing song from the system.
- `addToStation`: Adds an existing song to the playlist for an existing radio station.
- `removeFromStation`: Removes a song from the playlist for a radio station.
- `rateSong`: Sets an existing user's rating for an existing song, as a number of stars from 1 to 5.
- `clearRating`: Clears an existing user's rating on a song.
- `predictRating`: Predicts the rating that a user will assign to a particular song that they have not yet rated. The predicted rating will be a number of stars from 1 to 5.
- `suggestSong`: Suggests a song for a user that they are predicted to like.

You can assume that the set of users and the set of radio stations are fixed (e.g., at initialization) and thus you do not need to include operations to change these collections.

You will be completing the interface `StreamingRadio`, a template of which is included in the provided code. We have completed `clearRating` for you as an example to help you get started. You are allowed to modify this method for consistency with the others, if you take a different approach than that presented. The other abstract methods are initially declared as `void` methods with no parameters; you need to expand each declaration to specify a return type and parameters, as necessary. You also need to include a detailed comment for each abstract method describing its effect, its input parameters, its return value, any corner cases that the client may need to consider, any exceptions the method may throw (including a description of the circumstances under which this will happen), and so on.

> **Note:** If you cannot find an existing exception that is appropriate for a particular corner case, you may create your own (perhaps by following the format of `ExpressionError` from A2).

You should include enough details that a client could use this data structure without ever being surprised or not knowing what will happen, even though they haven't seen the implementation. Finally, you should use Javadoc-style comments where appropriate, so that running `javadoc` on your code will produce a readable documentation page. You do not need to use advanced features of Javadoc, but you should include `@param`, `@return`, and `@throws` comments as in our example code from class and the textbook.

A documentation page for the initial `StreamingRadio` interface is included in the provided code. This was generated using the following command:

```
javadoc -d doc cs445/a4/*.java
```

(Note here that '`cs445/a4/*.java`' specifies that all Java files within `cs445/a4/` should be processed, and '`-d doc`' specifies that the Javadoc output should be created in a directory named `doc`.)

Finally, to test if your interface is well-formed, you should compile it using `javac` from the command line. You must include your code in the `cs445.a4` package. You will not be able to run your program, because you will not be writing any executable code, much less a `main` method. It should, however, compile successfully.

4 Grading

Your grade will be assigned based on whether the code is in the correct form and compiles successfully (10%); whether Javadoc generates documentation correctly (12%); and your specification of each of the required abstract methods: `addSong` (8%), `removeSong` (8%), `addToStation` (12%), `removeFromStation` (12%), `rateSong` (12%), `predictRating` (14%), and `suggestSong` (12%).

5 Submission

Upload your java files in the provided Box directory as edits or additions to the provided code.

All programs will be tested on the command line, so if you use an IDE to develop your program, you must export the java files from the IDE and ensure that they compile and generate Javadoc documentation from the command line. Do not submit the IDE's project files.

Your project is due at 11:59 PM on Thursday, November 16. You should upload your progress frequently, even far in advance of this deadline: **No late submissions will be accepted.**