Homework 8

Did you ever wonder how Spotify and YouTube manage their playlists? In this homework you will build a simple object-oriented architecture for playlist management.

Usage

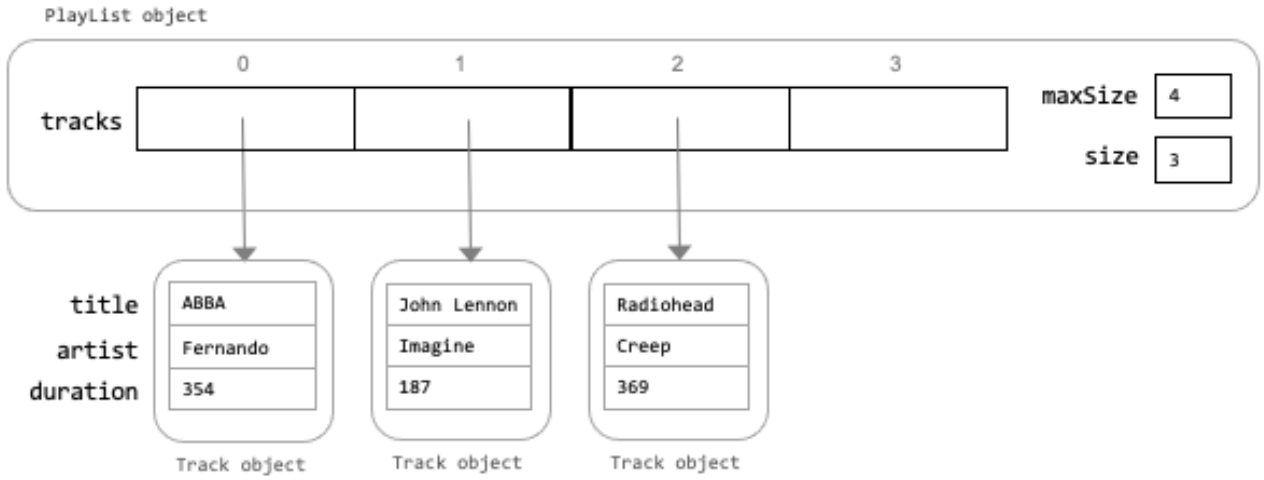
Our playlist management system consists of three classes. The testing (client) class is named `RuniFi`. The main method in this class creates a playlist and a few tracks, adds the tracks to the playlist, and then runs various tests. Here is a typical session (some **bold** formatting was added, to improve readability):

```
% java RuniFi

Adding tracks...

My list:
ABBA, Fernando, 354
John Lennon, Imagine, 187
Radiohead, Creep, 369
Michael Jackson, Thriller, 222

Total time duration of my list (in seconds):1132

After adding Yesterday at location 1:
ABBA, Fernando, 354
The Beatles, Yesterday, 125
John Lennon, Imagine, 187
Radiohead, Creep, 369
Michael Jackson, Thriller, 222

Index of Creep: 3

Index of Shake It Off: -1

After removing the track in location 2:
ABBA, Fernando, 354
The Beatles, Yesterday, 125
Radiohead, Creep, 369
Michael Jackson, Thriller, 222

After removing the first track:
The Beatles, Yesterday, 125
Radiohead, Creep, 369
Michael Jackson, Thriller, 222

After removing the last track:
The Beatles, Yesterday, 125
Radiohead, Creep, 369

New list:
Cher, Believe, 240
```

```
Coldplay, Yellow, 269
Lady Gaga, Shallow, 217
Doja Cat, Woman, 172

New list after removing Yellow:
Cher, Believe, 240
Lady Gaga, Shallow, 217
Doja Cat, Woman, 172

My list after merging with new list:
The Beatles, Yesterday, 125
Radiohead, Creep, 369
Cher, Believe, 240
Lady Gaga, Shallow, 217
Doja Cat, Woman, 172

The shortest track in my list is Yesterday

My list after sorting by increasing duration:
The Beatles, Yesterday, 125
Doja Cat, Woman, 172
Lady Gaga, Shallow, 217
Cher, Believe, 240
Radiohead, Creep, 369
(base) shimonschocken@Shimons-MacBook-Pro solution % a
```

Data structure

Our system consists of three classes: `Track`, `PlayList`, and `RuniFi`. Each musical track is represented by a `Track` object (an instance of a class named `Track`), and each playlist is represented by a `PlayList` object (an instance of a class named `PlayList`). Inside the `PlayList` class, a playlist is implemented as an array whose elements point to `Track` objects, as follows:



Note that clients who create and use playlists (programs like `RuniFi`) use the `PlayList` class API without knowing anything about this internal architecture. That's one example of a good object-oriented design: Clients can use objects without worrying about how they are implemented.

Implementation plan

1. Read the code of `RuniFi`, and learn how it uses the services of the `PlayList` and `Track` classes. As we said above, `RuniFi`, which is a client program, knows absolutely nothing about the internal implementation of playlists. It creates and manipulates playlists like black box objects, using the class API (which is given to you as the skeletal `PlayList.java` file).

2. Implement and test the helper `formattedDuration` method in the `Track` class. You can use Java's `String.format` method (which is documented in the `String` class API). Or – better – flex your programming muscles and write a few lines of code that implements this formatting logic directly. You will have to do some simple computations, and convert `int` values to characters and strings – a good little programming exercise.

3. Implement the `PlayList` methods, in the order in which they appear in the `PlayList.java` file. This will allow you to unit-test these methods in the order in which they are called and tested in `RunFi.java`.

Implementation tips

1. The mother of all tips in this particular program: Don't forget to modify the `size` field when adding or deleting tracks.

2. When implementing the `toString` method of `PlayList`, remember that `Track` objects know how to display themselves. Use this ability.

3. When implementing the `add(int i, Track track)` method, note that there are two cases. If you are adding a track to the end of the list, that's easy. Otherwise, you have to make room for the new track. To do so, you have to write code that shifts all the elements of the `tracks` array one position to the right.

4. Note that the playlist *is ordered*. The order is simply the order by which the tracks were added to the list by the user. Therefore, when removing a track: (i) you are not allowed to change the order of the remaining tracks in the list, and (ii) you have to "close the gap" in the array, by shifting all the tracks on the right of the deleted track one step to the left.

5. When implementing the `add(PlayList other)` method, remember that `PlayList` objects know how to add tracks to themselves. Use this ability.

6. The `indexOf(String title)` method makes no assumptions about the lower-case / upper-case format of the given `title` input. In other words, inputs like `Imagine`, `imagine`, `IMAGINE`, or `ImaGine` should all cause the method to focus on the track whose title is `"Imagine"`. When implementing this method, you can leave this implementation detail to the end. Start by

assuming that the input is correct, i.e. `"Imagine"`, and make sure that the method works correctly. Then take care of the upper/lower case detail.

7. When implementing the remove(String title) method, remember that we already have a method that knows how to return the index of a track that has some title.

8. To test your implementation of the minIndex(int start) method, use the `titleOfShortestTrack` method, whose implementation is given. Notice that the latter method provides only a limited test of `minIndex`, since it starts the search only at index 0. You must write some test code that calls `minIndex` with various start values. Put this code in the main method of `RuniFi`.

9. The `sortedInPlace` method operates directly on the `tracks` array. It does not create or return a new array. To swap two array elements, you have to put the value of one of the two elements in some temporary storage. For example, to save the value of `tracks[i]`, use a statement like `Track temp = tracks[i]`. This statement declares an object variable named `temp`, and makes it point to the object that `tracks[i]` points at.

Submission

Submit only one Java class: `PlayList.java`. There is no need to submit `Track` and `Runifi`, even though you wrote some code in these classes. Before submitting your work for grading, make sure that your code is written according to our Java Coding Style Guidelines. **Submission deadline**: February 15, 2024, 23:55.