Instructions

Read all the instructions before you start coding.

1. Read the statement, requirements and assumptions carefully.
2. Enter the following link and provide your name and email.
3. Click on “start the interview” button, once you do the timer will start recording all the typing you do, we want to see your thought process rather than only the end product, so it’s important that you design your own solution instead of copying from the internet.
4. Do all the coding within the tool (interviewzen).
5. Don’t use any IDE.
6. Try to code in such a way that your would compile although you don’t need to check in a real compiler that it does.
7. You are free to use any JDK 8 collection.
8. If you can/want to use streams that’s great but not a hard requirement.
9. Manually test your code and document your tests in the form of comments at the bottom of the exercise area.

Statement

Suppose you are in charge of designing a service that keeps track of the top N most liked music videos, each time that someone likes a video an operation of the service will be invoked with three parameters, the id of the video, the current number of likes and its genre (Rock, Pop, Country, Metal, Reggaeton, Quebradita, etc.)

We will have multiple clients, e.g. Web page, Android app that will call another operation on the service returns the N most liked videos sorted from more liked to less liked.

Also it’s necessary to support another operation that will receive one argument, the genre and will return the total number of likes received for that genre.

Requirements

1. Design the service as a Java class, service must implement the following interface:

```
package org.todito.music;

import java.math.BigInteger;

public interface LikeTrackingService {
  void recordLikes(String videoId, BigInteger currentLikes, String genre);
  <propose a data type> getTopLiked();
  BigInteger getNumberOfLikesForGenre(String genre);
}
```

Note that the return type of getTopLiked is not specified, it's open for you to propose in order to avoid influencing your design / implementation.

2. Use Java Collections and Java data types only to store your data.
3. Design the three operations described in the statement as methods from the class with the best time complexities that you can design for, if trades off are taken document them atop the page as comments.
4. Document any other assumption that you are taking not provided in the assumptions section in atop the class as comments, for example:

```
// I assume currentLikes is a positive number.
```

5. Provide the time complexity of each operation in a JavaDoc on each method.
6. Provide the space complexity in a JavaDoc at the class level.
7. Hit submit when done (It will let you playback your response to see how you did).
8. **For extra points! (optional)** If you were able to finish the previous problem we would like to see if you can then change the interface and implementation to take advantage of CompletabeFuture.
    a. Copy your submitted response (make sure you submitted the previous response) into a new instance of the test (open again link) with your same name and email.
    b. Change your implementation so you return CompletableFutures wrappers (see https://www.baeldung.com/java-completablefuture for more information)
    c. The interface now will look like the below.

```
package org.todito.music;

import java.math.BigInteger;
import java.util.concurrent.CompletableFuture;

public interface LikeTrackingService {
  CompletableFuture<Void> recordLikes(String videoId, BigInteger currentLikes,
String genre);
  CompletableFuture<propose a data type> getTopLiked();
  CompletableFuture<BigInteger> getNumberOfLikesForGenre(String genre);
```

```
}
```

Again note that the return type of getTopLiked is not specified, it's open for you to propose in order to avoid influencing your design / implementation.

d. Create example client code that calls the service and is able to consume the CompletableFuture.
e. Explain in your own words what is the usage of using completable future in the form of comments.

Assumptions

- There are V videos where V can be in the order of millions meaning that storing all video ids in memory won't be possible.
- The number of genres although not determined is small, suppose that less than 100.
- The number of concurrent calls to the three operations is high pretty high for this reason we want to keep consistency in the data, i.e. methods will be called from different threads concurrently so keep that in mind.
- Top-N number is much smaller than the number of genres, and we want it to be configurable when service is initialized / built.
- Assume that your program will run in a single JVM.