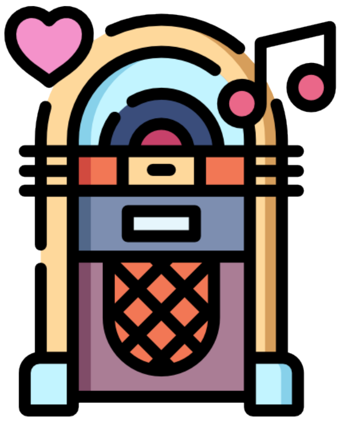
CSCI-605 Advanced Object-Oriented Programming Concepts

Homework 6: Jukebox Hero



1 Introduction

Hip teen hangout spot Arnold's is gearing up for the annual team dance marathon. The jukebox needs to be stocked with enough *45* singles such that no song will be repeated if they are randomly played. How many songs should the jukebox hold?

Resident "math whiz" Richie Cunningham whips out his vintage mechanical calculator.

- According to the Guiness Book of World Records, the longest team dance marathon lasted 24 hours (1440 minutes).
- According to DJ Venus Flytrap from radio station WKRP (in Cincinnati) the average radio song is just around 3 minutes.
- This means the jukebox needs a minimum of 480 songs (1440/3) to last the entire marathon.

Armed with this knowledge, Arnold reasons that if he acquires 1000 songs there should be no repeats because more than half the songs will never be played. Unfortunately his line of thinking couldn't be more wrong. The real number of songs required may surprise even you.

In this homework you will develop a simulation that finds for a given total number of songs the average number of songs played at random to reach a repeat. This is a famous probability problem known as The Birthday Paradox.

1.1 Goals

The goal of this lab is to apply what you have learned about the Java Collections Framework to a practical problem, make sound choices about which data structure works best for any particular set of needs, and use them correctly in your programs. Note that "works best" includes a number of factors including how the data is stored and retrieved, as well as performance.

1.2 Provided Files

1. The **data** folder includes several data files that contain songs (see "Program Operation" for more detail). These files should be used when implementing and testing your program.
2. The **sample** folder contains examples of the expected output from the program when it is implemented and executed.

2 Implementation

2.1 Command line arguments

The program runs with two required command line arguments, the file name followed by the random seed number:

```
$ java Jukebox filename seed
```

If the number of arguments is correct, they are guaranteed to be the correct format, a string followed by an integer value. If the number of arguments is incorrect, display a usage message and exit the program, i.e.:

```
Usage: java Jukebox filename seed
```

2.2 Program Operation

The jukebox will be loaded with the songs contained in the specified song data file. Each line on the file includes information regarding one single song, e.g:

```
TRMMMYQ128F932D901<SEP>SOQMMHC12AB0180CB8<SEP>Faster Pussy cat<SEP>Silent Night
```

Here we only care about the third (artist) and fourth (song) fields. You can use the split method with the string `"<SEP>"` as the argument to split the line into a native array of fields that are delimited by the separator. You should also specify the second optional argument, the limit, as 4. Otherwise reading in some songs will fail and throw an exception because they are improperly formatted in the song file/s.

When executed the program displays the following information:

- The total number of songs in the jukebox.
- The first five songs that are played when the simulation is run.
- The total time in second it takes to run the simulation.
- The number of times the simulation is run. For this assignment that number is 50,000.
- The total number of songs played throughout the entire simulation.
- The average number of songs played to get a duplicate, across the entire simulation.
- The song that was played the most. You only need to choose one if there is tie.
- For the song that was played the most, list all the songs by that artist in that are in the jukebox, alphabetically by song name along with the total number of times each song was played.

We have provided a sample run per each song data file using a seed of 0.

2.3 Design

While you are free to design and implement your program as you see fit (as long as it meets the requirements detailed in this document and the design is purely object oriented manner), below are some suggestions for the minimum classes that you should consider implementing:

- Song - represents a single song in the jukebox. Every song has a name and artist. This class should implement the behavior described below. Note that the following list is not comprehensive:
  - A `toString` method that returns a string representation of the song in the format "Artist: XXX, Song: XXX".
  - A `equals` method. Two songs are equal if both the artist and the song are the same.
  - A method that returns a hash code for the song. It should meet the requirements of the general contract for `equals` and `hashCode` (i.e. if `a.equals(b)` then `a.hashCode() == b.hashCode()`).
  - A song must define a natural ordering such that they are arranged in some order. You may choose the ordering based on the requirements.

- Jukebox - The main executable class. This class should define a `main` method that creates and uses an instance to perform the following tasks:
  - Reads in the song data file into the jukebox.
  - Runs the simulation *50,000* times while collecting information to determine the average number of plays to reach a duplicate, as well as the counts for each time a song was played.
  - Generates the statistics.
  - Displays the statistics.

You must use data structures that work efficiently with a large number of songs. A song in the jukebox should be accessible in constant time. Songs that are played in one "run" to determine a duplicate should also work in constant time. **Your program should be able to run the simulation on the million song data file, *tracks-1M.txt* in under one minute!**

2.4 Implementation Hints

- There are some songs in the larger data files that are duplicates (same exact artist and song name) that should not be re-added to the jukebox. Look at the sample runs for more detail about each individual song file.
- To seed a random number generator refer to Random's constructor. You should only create one instance of this generator and use it repeatedly to generate random numbers using the method nextInt that is bounded by the total number of songs in the jukebox. Please note, you should not be doing any shuffling of the songs in this simulation!
- To do timing, use the System.currentTimeMillis method. Before running the simulation call it once to get the start time. When the simulation ends call it again to get the end time. The elapsed time in seconds is the difference between the two.
- Your program should be able to produce the same results (ties notwithstanding and not with the exact times listed) as long as:

  - You seed the random number generator correctly.
  - You generate only one instance of the random generator and use it repeatedly.
  - You run the simulation the required number of times, each time stopping as soon as you get a duplicate and not including that duplicate in your calculations.
- Run with the smaller test files to confirm your results before trying the larger ones.

3 Submission

You will need to submit all of your code to the MyCourses assignment before the due date. You must submit your `src` folder as a ZIP archive named "hw6.zip" (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework).

Do not submit the data directory! You should not zip up your entire project folder! Go inside the project folder and zip only the src folder!

4 Grading

The following deductions will apply:

- Completeness/Implementation: up to 100% if a part of the writeup is not observed.
- Correctness/Testing: up to 50% or affected part if the solution is not correct.
- Quality/Style/Documentation: up to 50% if solution is poorly designed.
- Explanation: up to 100% if the student is not able to respond grader's questions.