Assignment 4

COMP 250 Fall 2018

posted:	Tuesday, Nov. 20, 2018
due:	Tuesday, Dec. 4, 2018 at 23:59

The Teaching Assistants handling this assignment are Tabish Syed (email), Johannes Brustle (email), Matt Grenander (email), Qing Tian (email), Anand Kamat (email).
Their office hours will be posted on mycourses Announcements.

General Instructions

- **Submission instructions**
  - Late assignments will be accepted up to 2 days late and will be penalized by 10 points per day. Note that submitting one minute late is the same as submitting 23 hours late. We will deduct 10 points for any student who has to resubmit after the due date (i.e. late) irrespective of the reason, be it wrong file submitted, wrong file format was submitted or any other reason. This policy will hold regardless of whether or not the student can provide proof that the assignment was indeed "done" on time.
  - Don't worry if you realize that you made a mistake after you submitted : you can submit multiple times. We encourage you to submit a first version a few days before the deadline (computer crashes do happen and myCourses may be overloaded during rush hours).
  - Please store all your files in a folder called "Assignment4", zip the folder and submit it to myCourses. Inside your zipped folder, there must be the following files.
    - `HashPair.java`
    - `MyHashTable.java`
    - `Song.java`
    - `MusicStore.java`

    Do not submit any other files, especially .class files. Any deviation from these requirements may lead to lost marks
- The starter code for all the above classes is provided. You must **not** modify the `HashPair` and the `Song` class. Note that for this assignment, you are NOT allowed to import any other class besides the ones already imported for you. **Any failure to comply with these rules will give you an automatic 0.**
- We have included with these instruction a tester class, which is a mini version of the final tester used to evaluate your assignment. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors.

We will test your code on a much more challenging set of examples. We therefore highly encourage you to modify the tester class and expand it.

- You will automatically get 0 if your code does not compile.
- Failure to comply with any of these rules will be penalized. If anything is unclear, it is up to you to clarify it by asking either directly a TA during office hours, or on the discussion board on myCourses.

Your task

For this assignment you will write several classes to simulate an online Music store. Make sure you implement all required methods according to the instructions given below. In addition to the required methods, you are free to add as many other **private** methods as you want. Note that `null` keys are not allowed in the hash table. Remember that in most of scenarios objects comparison does not use '=='.

[**70 points**] The class `MyHashTable` has the following fields:

- An `int` for the number of entries stored inside the table.
- An `int` for the number of buckets the table has (Note that this value is initialized by the constructor, but could change later on if the number of entries increases).
- A `final double` representing the load factor for the hash table.
- An `ArrayList` of buckets used to store the entries to the table. Where each bucket is a LinkedList of HashPairs.

Implement the following `public` methods in the `MyHashTable` class:

- The constructor `MyHashTable()` which takes an `int` as input representing the initial capacity of the table.[1] Using the input, the constructor initializes all the fields.
- A `put()` method that takes a *key* and a *value* as input. The method adds a `HashPair` of the *key* and *value* to the hash table. If a `HashPair` with the *key* already exists in the hash table, then you should overwrite the old *value* associated with the *key* with the new one. This method should be $O(1)$. If in this hash table there was a previous value associated to the given key, then the method overwrites it with the new value and returns the old one. If there was no value associated to the given key, than the method returns `null`.
- A `get()` method which takes a *key* as input and returns the *value* associated with it. If there is no such key in the hash table, then the method should return `null`. This method should run in $O(1)$ on average.
- A `remove()` method that takes a *key* as input and removes from the table the entry (i.e. the `HashPair`) associated to this *key*. The method should return the *value* associated to the *key*. If the *key* is not found, then the method returns `null`. This method should run in $O(1)$ on average.
- A `rehash()` method which takes no input and modifies the table so that it contains double the number of buckets. This method should be $O(n)$ where n the number of entries in the table.
- A `keys()` method which takes no input and returns an `ArrayList` containing all the keys in the table. The `keys` in the returned `ArrayList` may be in any order. This method

[1] The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created.

should be $O(n)$ where n the number of entries in the table.

- A `values()` method which takes no input and returns an `ArrayList` containing all the ***unique*** values in the table. The returned `ArrayList` of *unique* `values` may be in any order. This method should be $O(n)$ where n the number of entries in the table.

Inside this class you should also implement the following methods from the `private` nested class `MyHashIterator`.

- The constructor which should be $O(n)$.
- A `hasNext()` method which should be $O(1)$ and return `true` if the hash table has a next `HashPair`.
- A `next()` method which is also $O(1)$ and return the next `HashPair`.

[30 points] Implement the following `public` methods inside the `MusicStore` class. Note that you are allowed to add as many `private` methods and attributes as you see fit.

- The constructor `MusicStore()` which takes as input an `ArrayList` of `Song`s to initialize the `MusicStore`.
- A method `addSong()` which takes a `Song` as input and adds it to the MusicStore. This method should be $O(1)$.
- A method `searchByTitle()` which takes a `String` as input and returns the `Song` with the provided title. If there are multiple songs with the same title, you may return *any* one of them. Note that the input should match exactly the `Song` title. This method should be $O(1)$.
- A method `searchByArtist()` which takes a `String` as input representing an artist and returns an `ArrayList` of `Song`s containing all the songs in the MusicStore performed by the given artist. The `Song`s in the returned `ArrayList` may be in any order. This method should be $O(1)$.
- A method `searchByYear()` which takes an `Integer` as input representing a year and returns an `ArrayList` of `Song`s containing all the songs in the MusicStore produced in the given year. The `Song`s in the returned `ArrayList` may be in any order. This method should be $O(1)$