CSCI 1933 Project 5

Hash Maps

Instructions

Please read and understand these expectations thoroughly. Failure to follow these instructions could negatively impact your grade. Rules detailed in the course syllabus also apply but will not necessarily be repeated here.

- UPDATE **Due:** The project is due on **Friday, May 5th** by **11:55 PM**. Any late submissions received after this time will receive an automatic zero.
- **Identification:** Include your x500 in a comment in all files you submit. If you worked with a partner, you must include their x500 as well. Example: `//Written by shino012 and hoang159`.
- **Submission:** Please submit a `.zip` or `.tar.gz` file on Moodle containing your `src/` folder with all the `.java` files. You are allowed to modify your submission, so submit early and often.

  Verify that all the required and/or necessary files are included in your submission. Failure to submit the correct files will result in a score of zero for all missing parts. Additionally, late submissions, as well as submissions made in an incorrect format, are likely to receive a penalty.
- **Partners:** You may work alone or with *one* partner. **Failure to identify your partner in your submission as outlined above is indistinguishable from cheating and you will both receive a zero.** Ensure all code shared with your partner is private. Only one project submission is required per group.
- **Java Version:** The TAs will grade your code using Java 8. Please ensure your code works in Java 8, which is the version installed on the CSELabs machines. Java 6 and Java 7 will work with Java 8 (at least for the purposes of this course).
- **Libraries:** You are permitted to import any libraries needed to read input from a file (for example, java.util.Scanner and java.io.FileReader). All other imports are disallowed. This means `System` is fine, but something from `java.util` is not.
- **Grading:** You must use the *exact* class and method signatures we ask for. This is because we may use unit tests to grade parts of your code. If your code uses other names, you may receive a penalty.

  Grading will be done by the TAs, so please address grading problems to them **privately**.
- **Questions:** Please post questions related to this project in the Projects forum on Moodle. **Do not e-mail the TAs or the class email unless you have private questions.** All

questions about how to code Java or questions about understanding this instruction document belong on Moodle. However, please avoid posting your answer code on Moodle, even if it is not working.

Introduction

In this project you will program a HashMap, which implements the `Map<K,V>` interface. A `Map` is an Object that assigns keys to values, with each key having at most one value.

A simplified `Map<K,V>` interface will be provided along with this writeup. **Do not modify this interface or use another interface with the same name.** The `Map` interface requires the methods below:

- `public void clear()`: This method removes all mappings from the `Map`. This method should be more efficient that calling `remove` on every element.
- `public boolean containsKey(K key)`: This method returns `true` if the `Map` contains a mapping for the specified key, and `false` otherwise.
- `public boolean containsValue(V value)`: This method returns `true` if the `Map` contains a mapping for the specified value, and `false` otherwise.
- `public V get(K key)`: This method returns the value to which the specified key is mapped. If the `Map` does not contain the specified key, the method instead returns `null`.
- `public boolean isEmpty()`: This method returns `true` if the `Map` contains no key-value mappings, and `false` otherwise.
- `public V put(K key, V value)`: This method associates the specified key with the specified value in the `Map` and returns the newly associated value, **if** the specified key is not already in the `HashMap`. If the specified key is already in the `HashMap`, the association is left unchanged, and the current value is returned instead.
- `public V remove(K key)`: This method removes and returns the mapping for the specified key in the `Map` if it is present, and returns `null` otherwise.
- `public V replace(K key, V value)`: If the specified key is already mapped to a value, this method will return the previous value associated with the key and replace it will the input value. If the specified key is not mapped to a value, this method will simply return null, altering nothing in the `HashMap`.
- `public int size()`: This method returns the number of key-value mappings in the `Map`.

1 HashMap

The `HashMap<K,V>` data structure will be implemented using the `Map<K,V>` interface outlined above, using an array of linked lists as the underlying data structure. The individual nodes of the linked lists will be the key-value mappings, stored as entries. The `Entry<K,V>` objects should contain the following attributes:

- `K key`;
- `V value`;
- `Entry<K,V> next`;

The `Entry<K,V>` class is not provided, you must implement it yourself.

Details

In addition to the implementing the interface methods outlined above, the `HashMap` class should include two constructors:

- `public HashMap(int buckets)`: This constructor will create anarray of the specified length.
- `public HashMap()`: This constructor will create a default base array of length 64.

As well as the following two methods:

- `public String toString()`: This method returns a `String` representation of the `HashMap` in the following format:
  - If the `Map` is empty, return `"{}"`
  - If the `Map` has one key-value mapping (for example - Key: 1, Value: A), return `"{1: A}"`.
  - If the `Map` contains multiple elements (for example - Key: 1, Value: A; Key: 2, Value: B; Key: 3, Value: C), return `"{3: C, 1: A, 2: B}"`. Note that the items are out of order because the `HashMap` does not guarantee any specific order. This is not a problem, as long as every key-value mapping in the `Map` is present once (and only once).
  - This method should work regardless of the generic types `K` and `V`.
  - Make sure there is no trailing comma. For example, `"{1: A, 2: B,}"` is not correct.

- `private int hash(K key)`: This method returns an int, determining which "bucket" (index in the array) an `Entry` should be placed into. This should be calculated by taking the remainder of the key divided by the number of buckets.

It should be noted that the `containsKey`, `get`, `put`, `remove`, and `replace` methods should be more efficient that traversing the entire array and each linked list it contains. The `hash` method is critical to this requirement. Note, however, that this does not require the key be an `Integer` - objects in Java have a function `hashCode()` which generates a hash value (32-bit integer). This value may be used to determine in which bucket in the array an `Entry` should be stored.

Additionally, keep in mind that `null` values are allowed in the `HashMap`. At most one key may be `null`, and any number of values may be `null`.

2 Weather Data

Available with this lab is the file `weather_10000.txt`. Opening this file, each line of data may be read as a weather record, containing the following information: *city, country, longitude, latitude, temperature, minimum, maximum, description.*

This data was obtained from the OpenWeatherMap API (`api.openweathermap.org`), through a process known as fetching - obtaining data from somewhere to be used in your program. When fetching data, you will usually go in with some parameters, which specify or limit the data intended to be retrieved. Once the data has been fetched, it then needs to be parsed into a format that is easier to be read and/or processed, which is what you see in the `weather_10000.txt` file.

The ability to fetch data is useful when dealing with dynamic data - that is, data that changes asynchronously over time. For example, as the weather continues to change over time, the data available at OpenWeatherMap will update to show these changes.

The final section of this project goes into fetching data from APIs in more detail, and is optional. Here, your task is simply to create a `WeatherRecord` object with these attributes, parse the data in `weather_10000.txt` into instances of `WeatherRecord`, and store these instances as values in your HashMap.

This is fairly open-ended, you are welcome to do this whatever way makes sense to you.

3 Runtime Analysis

Finally, analyze the runtime of all the `Map` methods implemented in `HashMap` (*ignore* ***`toString`****, but include* ***`hash`***). Analyze both the *average case runtime* and the *worst case runtime* using Big-O notation. Please include explanations of your runtimes.

Please submit your analysis file along with your source code. The analysis should be in `.txt` or `.pdf` format. Other formats are not acceptable and we will apply a penalty.

4 Fetching Data from the OpenWeatherMap API

> **NOTE:** This step is optional.

For an earlier section of this project, we provided you with data from the OpenWeatherMap API to store in the `HashMap`. For this section, we are asking you to fetch the data yourself.

The Oracle documentation on URLConnections (https://docs.oracle.com/javase/tutorial/networking/urls/readingWriting.html), as well as the documentation available on OpenWeatherMap API (api.openweathermap.org) should be enough to get you started.

Finally, for this section, you are allowed to reference external libraries for parsing JSON/XML. As a starting point, we recommend using org.json (https://github.com/stleary/JSON-java) for JSON and SAXParser (`javax.xml.parsers.SAXParser`) for XML, but you are welcome to use any other libraries available for this purpose that you are comfortable with.