CMPSCI 187 / Spring 2015
URL Table

Due on never!

Marc Liberatore and John Ridgway

Morrill I N375
Section 01 @ 10:00
Section 02 @ 08:30

Contents

Overview

In this assignment, you will implement a simple data structure to hold URLs and related data for use in a web crawler. Your implementation will support a meaningful `hashCode` method. You will then implement a simple hash table with support for adding and removing entries, as well as iteration. Your hash table will be able to support a load factor of greater than one.

Learning Goals

- Demonstrate understanding of `hashCode` by implementing it for a multi-attribute class.
- Demonstrate understanding of hash tables.
- Demonstrate understanding of the chaining method of collision resolution in hash tables.

General Information

Read this entire document. If, after a careful reading, something seems ambiguous or unclear to you, then communicate to the course staff immediately.

Start this assignment as soon as possible. Do not wait until 5pm the night before the assignment is due to tell us you don't understand something, as our ability to help you will be minimal.

Reminder: Copying partial or whole solutions, obtained from other students or elsewhere, is academic dishonesty. Do not share your code with your classmates, and do not use your classmates' code. If you are confused about what constitutes academic dishonesty you should re-read the course syllabus and policies. We assume you have read the course information in detail and by submitting this assignment you have provided your virtual signature in agreement with these policies.

You are responsible for submitting project assignments that compile and are configured correctly. If your project submission does not follow these policies exactly you may receive a grade of zero for this assignment.

Policies

- For some assignments, it will be useful for you to write additional java files. Any java file you write that is used by your solution MUST be in the provided `src` directory you export.
- The course staff are here to help you figure out errors (not solve them for you), but we won't do so for you after you submit your solution. When you submit your solution, **be sure to remove all compilation errors from your project**. Any compilation errors in your project will cause the autograder to fail, and you will receive a zero for your submission. **No Exceptions!**

Test Files

In the `test` directory, we provide several JUnit test cases that will help you keep on track while completing the assignment. We recommend you run the tests often and use them to help create a checklist of things to do next. *But you should be aware that we deliberately do not provide you the full test suite we use when grading.*

We recommend that you think about possible cases and add new `@Test` cases to these files as part of your programming discipline. Simple tests to add will consider questions such as:

- Do your methods handle edge cases such as integer arguments that may be positive, negative, or zero. Many methods only accept arguments that are in a particular range.
- Does your code handle unusual cases, such as empty or maximally-sized data structures?

More complex tests will be assignment-specific. To build good test cases, think about ways to exercise methods. Work out the correct result for a call of a method with a given set of parameters by hand, then add it as a test case. Note that we will not be looking at your test cases (unless otherwise specified by the assignment documentation), they are just for your use.

Before submitting, make sure that your program compiles with and passes all of the original tests. If you have errors in these files, it means the structure of the files found in the `src` directory have been altered in a way that will cause your submission to lose some (or all) points.

Problem 1

Note: You may not use any of the classes from the Java Platform API that implement the Collection interface (or its sub-interfaces, etc.). The list of these classes is included in the API documentation, available at: `http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html`. Doing so will be viewed as an attempt to cheat.

Import Project into Eclipse

Begin by downloading the starter project and importing it into your workspace. It is very important that you **do not rename** this project as its name is used during the autograding process. If the project is renamed, your assignment will not be graded, and you will receive a zero.

The imported project may have some errors, but these should not prevent you from getting started. Specifically, we may provide JUnit tests for classes that do not yet exist in your code. You can still run the other JUnit tests.

The project should normally contain the following root items:

`src` This is the source folder where all code you are submitting must go. You can change anything you want in this folder (unless otherwise specified in the problem description and in the code we provide), you can add new files, etc.

Problem 1 continued on next page...

`support` This folder contains support code that we encourage you to use (and must be used to pass certain tests). You must not change or add anything in this folder. To help ensure that, we suggest that you set the support folder to be read-only. You can do this by right-clicking on it in the package explorer, choosing Properties from the menu, choosing Resource from the list on the left of the pop-up Properties window, unchecking the Permissions check-box for Owner-Write, and clicking the OK button. A dialog box will show with the title "Confirm recursive changes", and you should click on the "Yes" button.

`test` The test folder where all of the public unit tests are available.

`JUnit 4` A library that is used to run the test programs.

`JRE System Library` This is what allows Java to run; it is the location of the Java System Libraries.

If you are missing any of the above or if errors are present in the project (other than as specifically described below), seek help immediately so you can get started on the project right away.

Why Hash Tables?

Often we need fast access to a large set of otherwise unstructured data. Sometimes, we require keyed access to a set of values. That is, we have a set of key/value pairs that we wish to be able to access quickly using the keys. For example, personnel records are often keyed by social security number. Search engines crawl the web, cataloguing page contents as the first step in generating an index that users can then search. Perhaps you've heard of Google? The pages it retrieves are keyed by URL, and the associated contents are the value that might be stored. This general abstraction (storing key/value pairs in a way that provides fast lookup by key) is called a `Map` in Java.

We've learned about two general ways to provide fast access to data in such cases. One, trees, requires that a total order exist upon the keys. Another, hash tables, requires that keys be "hashable" — in Java, this means it must implement meaningful `equals` and `hashCode` methods. In this assignment we'll implement a hash code for a simple URL record, similar to what a web crawler would use, and we'll implement a hash table based on a toy version of `Map`.

The idea of a `Map` is an extremely useful one; many languages provide some version of a hash table as either part of the standard library or built into the syntax of the language itself.

Examining the code

In this assignment, you will fill out the `URLRecord` and `HashTable` classes. `URLRecord` is largely standalone, but you may want to look over the files in `support` before implementing `HashTable`.

In the `structures` package, we provide the `Map` interface you'll be implementing in `HashTable`. We also provide the `Entry` class, which represents a key/value pair and is the unit upon which a `Map` operates. Finally, we provide the `Node` class, which you will need when implementing chaining.

What to do

Implement each method of `URLRecord` and `HashTable` (including constructors) marked as TODO. Below are some hints and observations to get you started.

The two classes are independent, which should come as no surprise to you at this point: just because a web crawler might use the `HashTable` to store URLs and pages does not mean that `HashTable` depends upon `URLRecord`. `HashTable` is a generic collection.

You can use Eclipse to generate the `equals` and `hashCode` methods of `URLRecord` for you. Be careful to understand what it's doing if you choose to do so — blindly clicking "OK" in dialog boxes will not produce a correct implementation.

When implementing the various methods of `HashTable`, you'll need to think carefully about the various cases that can come up: Is a bucket empty or full? Is an element at the head of a chain, or deeper in? Is the underlying data structure you're using properly initialized in your constructor?

Good luck! This assignment combines many of the concepts and details of Java we've covered this semester.

Export and Submit

When you have completed the changes to your code, you should export an archive file containing the entire Java project. To do this, click on the `url-table-student` project in the package explorer. Then choose "File → Export" from the menu. In the window that appears, under "General" choose "Archive File". Then choose "Next" and enter a destination for the output file. Be sure that the project is named **`url-table-student`**. Save the exported file with the `zip` extension (any name is fine). Log into Moodle and submit the exported zip file.