Regis University CC&IS
CS310 Data Structures
Programming Assignment 5: Hashing with Sets and Maps

Problem Scenario

The charity was very impressed with your work from Assn 3. Nevertheless, the IT director at the charity just received the latest "Java Geek Weekly", and read all about hashing. Now, the IT director wants to replace the linked lists in your project with hash sets and maps.

In addition, the charity would like help donors out with their taxes. Therefore, they would like to print a report, identifying whether specific donations are tax deductible or not.

Program Requirements

You will replace your **DonorImpl** and **DonationImpl** classes from Assn 3 (to work with hash tables, instead of linked lists).

The **DonorImpl** will use a Hash Set to store the Donor objects. It will use an ***open addressing method*** of resolving collisions (linear probing, quadratic probing, or double hashing – your choice). The array size for this implementation is to be a size of 23 (a prime number).

The **DonationImpl** will use a "multi hashmap" to store Donation objects. It will to allow for multiple keys to be inserted into the hashmap. While the key for the map will be a hashcode, the value will be a reference to a linked list. Each donation that maps to this hashcode will be stored in a linked list. If there are multiple entries for the same hashcode, the collisions will be resolved by using the ***chaining method***. Note you will need to create a **MapEntry** class to implement the hashmap (see diagram on last page of this requirements document). Use the same size table for this implementation.

You will create your own hashing algorithms for this assignment.

First you will create a hashcode by adding the ASCII/Unicode values of each digit in ID (use the donor ID for the **DonorImpl** and the donation ID for the **DonationImpl**). Then use the modulus operator, so that your IDs will map correctly to the array indexes. For example, if the ID is 804, then the hashcode will be 56 + 48 + 52 = 156 mod 23 = 18.

The input will remain the same as the past assignments. However, you will not need to create the same reports as you have in the past assignments. **Remove** any code that created the previous report.

The charity will provide a simple flat file of donor IDs and donation IDs, for all donors who requested a tax report. Each line will contain a donor ID, and a list of donation IDs that the donor would like to check the tax deductible status on. For example:

```
3 302 303
1 105 102 107
2 204
```

The file, called **`donorRequests.txt`**, will be located in the **input** folder.

You will need to find the donor ID in your **DonorImpl**
and you will need to find each donation ID in your **DonationImpl**
and for each, determine if the donation is tax deductible

Modify the **PrintImpl** to generate a new report to be written to an output file named "assn5report.txt", which will be located in the output folder of the project.
Your report output will look like this:

```
Donor 3, Thomas Jefferson
      Donation 302 for $22.22 is tax deductible
      Donation 303 for $111.00 is NOT tax deductible
Donor 1, George Washington
      Donation 105 for $123.45 is NOT tax deductible
      Donation 102 for $200.00 is tax deductible
      Donation 107 for $123.45 is NOT tax deductible
Donor 2, Marie Antionette
      Donation 304 for $1000.00 is tax deductible
```

The program must follow the **CS310 Coding Standards** from Content section 1.9.

Deliverables

- Create **Javadoc headers**, and generate **Javadoc files** for each of your new ***implementation*** classes and for all new static methods in the main class. You are responsible for completing the comment headers created.
- Your original input data file (containing Donor and Donation data to build the hash tables from) will still be read from the **input** folder in your project.

  Place all test data files that you create to test your program in the **input** folder of your project, and name them as follows:

  `assn5input1.txt`
  `assn5input2.txt`
  (i.e. number each data file after the filename of **`assn5input.txt`**)
- Your new input data file will also be read from the **input** folder in your project.

  Place all test data files that you create to test your program in the **input** folder of your project, and name them as follows:

  `donorRequests1.txt`
  `donorRequests2.txt`
  (i.e. number each data file after the filename of **`donorRequests.txt`**)

  As a group, all of your test data files should demonstrate that you have tested every possible execution path within your code, including erroneous data which causes errors or exceptions.
- Your output report will be written to a **`taxReport.txt`** file in the **output** folder in your project.
- Add screen shots of **clean compile** of your classes to the documentation folder.
  - Remember, if you have compile errors, a red error symbol is placed on the file and folder name tabs. Be sure to clear all compile errors before capturing the screen shot.

  WARNING: Submittals without the clean compile will not be accepted.

Program Submission

This programming assignment is due by midnight of the date listed on the **Course Assignments by Week** page.

- Export your project from NetBeans using the same method as you did for previous assns.
    - Name your export file in the following format:
      `CS310<lastname>Assn<x>.zip`

      For example:
      `CS310SmithAssn5.zip`

- Submit your **.zip** file to the **Prog Assn 5** dropbox.

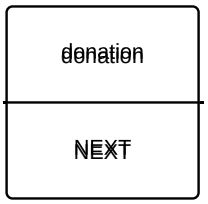
  Warning: Only NetBeans export files will be accepted.
  Do not use any other kind of archive or zip utility.

Grading

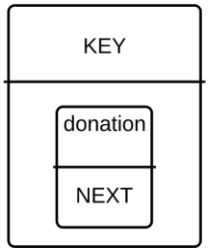
This program will be graded using the **rubric** that is linked under **Student Resources** page.

WARNING:
*Programs submitted more than 5 days past the due date will **not** be accepted, and will receive a grade of 0.*

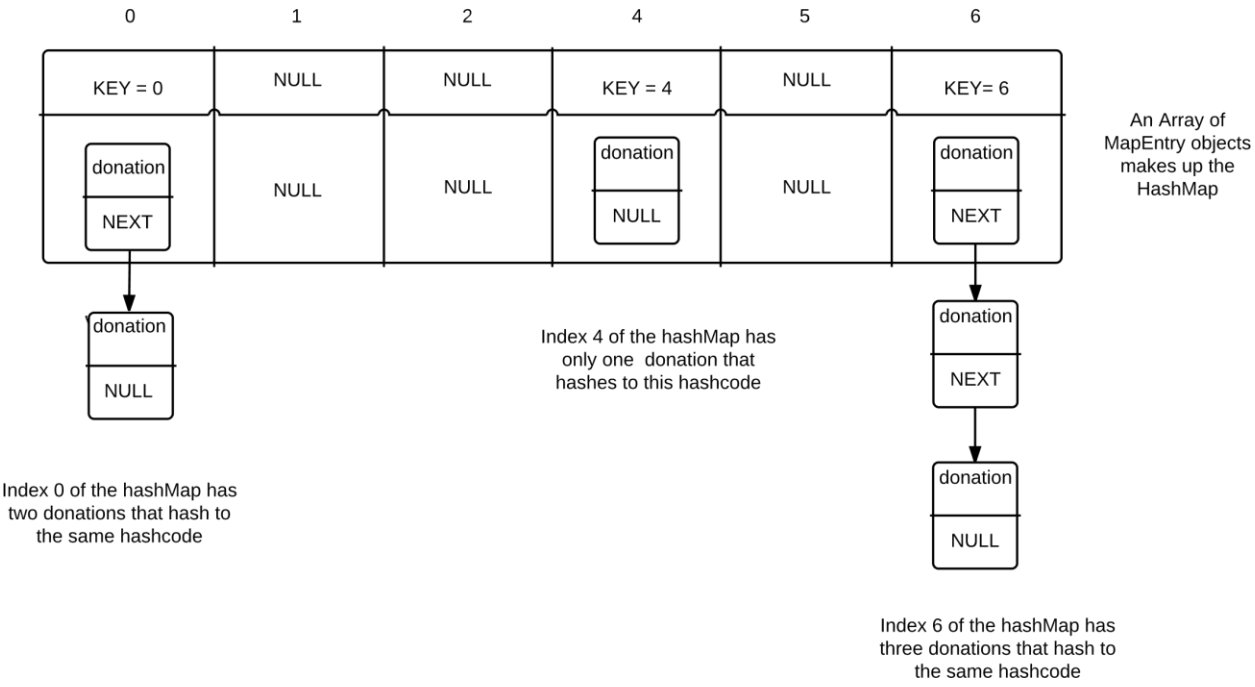
HashMap Diagrams



DonationNode



MapEntry - the key is the hashcode, while the value is a DonationNode object



An Array of MapEntry objects makes up the HashMap

Index 0 of the hashMap has two donations that hash to the same hashcode

Index 4 of the hashMap has only one donation that hashes to this hashcode

Index 6 of the hashMap has three donations that hash to the same hashcode