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

Problem Scenario

The real estate office was very impressed with your work from Assn 3. Nevertheless, the IT director at the office 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 real estate office would like help their realtors identify specific listings as either sold or not sold. Therefore, they would like to replace your prior reports with a report on this.

Program Requirements

You will replace your **RealtorLogImpl** and **PropertyLogImpl** classes from Assn 3 to work with hash tables, instead of linked lists.

The **RealtorLogImpl** will use a Hash Set to store Realtor objects. The hash table will be implemented as an array of Realtor objects (initialized to null when the table is created). The hash table will use the ***open addressing method*** via linear probing to resolve collisions. The array size for this implementation will be 23 (a prime number).

The **PropertyLogImpl** will use a hashmap to store Property objects. You will need to create a **MapEntry** class to implement the hashmap. Each MapEntry will be comprised of a key and a Property node. The hashmap will be an array of MapEntry objects (initialized to null when the table is created).

Since the MapEntry value is a PropertyNode, it can be used as a reference to a linked list. So if there are multiple entries that map to the same index, the collisions will be resolved by using the ***chaining method***, adding each entry to the top of the linked list at that index. This means that each Property object that maps to the same index (via the hashcode) will be stored in a linked list at that index. The array size for this implementation will be 17 (another prime number).

See the last page of this requirements document for diagrams of the hash tables.

You will create your own hashing algorithms for this assignment.

> Create a hashcode for a Realtor object in the **RealtorLogImpl** hash table by adding the ASCII/Unicode values of each character in the Realtor license number. After adding them, use the modulus operator, so that the license numbers will map correctly to array indexes. For example, if the license number were CC1122333, then the hashcode will be:
>
> 67 + 67 + 49 + 49 + 50 + 50 + 51 + 51 + 51 = 485 mod 23 = 2 (maps to index 2)
>
> Create a hashcode for a Property object in the **PropertyLogImpl** hash table by simply using the modulus operator directly on the MLS number, so that the MLS numbers will map correctly to array indexes. For example, if the MLS number were 1234567, then the hashcode will be:
>
> 1234567 mod 17 = 10 (maps to index 10)

The input file of Realtors and Property listings will remain the same as used in the previous assignments, except that **you can now assume all data will be valid**. So there will no longer be any need to validate and remove incorrect data.

Read the input file and use the correct hashCode to place each Realtor/Property object into the correct location in one of the hash tables. Then display the contents of each hash table as follows:

Add a **displayHash()** method to both **RealtorLogImpl** and **PropertyLogImpl**.

These methods will first display a header:

```
Realtor Hash Table:          OR    Property Hash Table:
```

and will then display where each Realtor/Property object is stored in the hash table.

For example:

```
Realtor Set:
      Index 0 is empty
      Index 1 is empty
      Index 2 contains Realtor CC1122333, Saul Peterson
      Index 3 contains Realtor WK8888889, Maria Gonzales
            :
      Index 21 contains Realtor BF9988777, Ashley Somerset
      Index 22 is empty

Property Map:
      Index 0 is empty
      Index 1 is empty
            :
      Index 5 contains Properties: 3399291 3428599 1231111
            :
      Index 8 contains Properties: 3388448 4569999
      Index 9 contains Properties: 3456789
      Index 10 is empty
      Index 11 contains Properties: 1232222
            :
      Index 16 is empty
```

Note that your **RealtorLogImpl** and **PropertyLogImpl** will no longer use **add** and **remove** methods (because no data will be removed from the hash tables). Instead, they will use **add** and **find** methods.

The **add** methods will be used to place the Realtor and Property objects into their hash tables.

The **find** method in the **RealtorLogImpl** class will search for a Realtor based on the realtor license number. It will return a reference to the found Realtor object, or will return ***null*** if the realtor is not in the hash table.

The **find** method in the **PropertyLogImpl** class will search for a Property based on the MLS number. It will return a reference to the found Property object, or will return ***null*** if the property is not in the hash table.

You will not need to create any of the reports that you created in the past assignments. So you can remove the calls that created those reports from your **main** method.

Instead, the real estate office wants you use the hash tables to create a new sales report of sold/unsold listings by realtor for them. The office will provide a simple flat file of realtor license numbers and property MLS numbers, for all of the realtors in the office who requested a report on their sold/unsold properties.

Each line of the file will contain a realtor license number and a list of property MLS numbers that the realtor would like to check the sold/unsold status on.

For example:

```
CC1122333 1231111 1232222
WK8888889 4569999
RT8375938 9582842
BF9988777 3428599 3388449 3399291
```

This file, called **`realtorRequests.txt`**, will be located in the **input** folder.

When reading the data in this file:

Use a hash code to find the Realtor license number in your **RealtorLogImpl** hash table

Use a hash code find each Property MLS number in your **PropertyLogImpl** hash table and for each, determine if the Property is sold or not.

Replace the old create report method in **PrintImpl** with a method that will generate a new report to be written to an output file named **`assn5salesReport.txt`**, which will be located in the **output** folder of the project. See sample report output below.

```
Realtor CC1122333, Saul Peterson
      Property 1231111 is available for $233999
      Property 1232222 is SOLD
Realtor WK8888889, Maria Gonzales
      Property 4569999 is available for $432999
Realtor RT8375938 does not exist
Realtor BF9988777, Ashley Somerset
      Property 3428599 is SOLD
      Property 3388449 does not exist
      Property 3399291 is available for $315999
```

Note that when the Realtor does not exist, you should not search for any of the Property listings on that line.

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

Additional Requirements

- 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 Realtor and Property 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:

`realtorRequests1.txt`
`realtorRequests2.txt`
(i.e. number each data file after the filename of **`realtorRequests.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.

- Add screen shots of **clean compile** of your classes to the documentation folder.
    - Be sure to clear all compile errors before capturing the screen shot.

WARNING: Submittals without the clean compile screenshots will **not** be accepted.
(This means that programs that do not 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 2** Submission Folder (located under **Assignments** tab in online course).

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

Grading

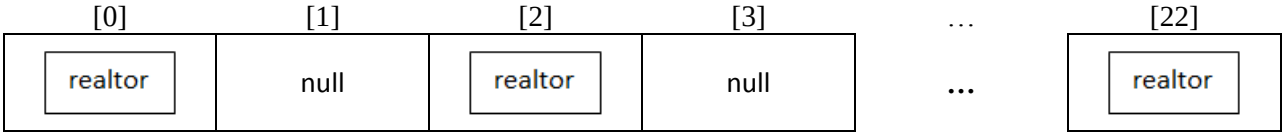
This program will be graded using the **rubric** that is linked under **Student Resources** page.

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

(see next page for diagrams)

HashSet Diagram

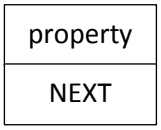
The hash set is an array of Realtor objects:



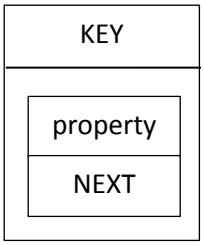
Each realtor maps to one index.

HashMap Diagrams

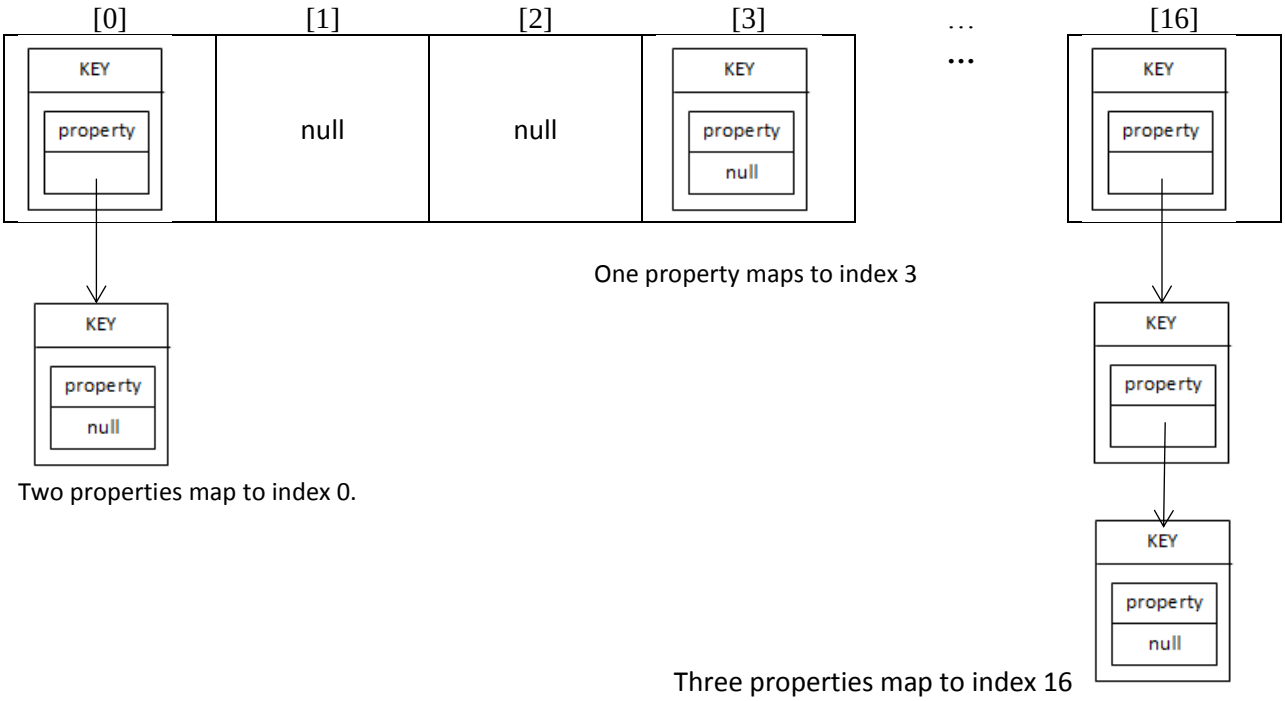
PropertyNode



MapEntry
(the **key** is used to determine the hashcode,
and the value is a PropertyNode object)



The hash map is an array of MapEntry objects:



One property maps to index 3

Two properties map to index 0.

Three properties map to index 16