Computer Laboratory 7
CSci 1913: Introduction to Algorithms, Data Structures, and Program Development
March 7/8, 2017

0. Introduction.

This laboratory assignment involves implementing a data structure called a *map*. A map acts something like a Python dictionary, in that it associates key objects with their corresponding value objects. However, it is implemented as a Java class, and it uses Java arrays internally.

1. Theory.

A map is a set of *key-value* pairs. Each key object is said to be *associated* with its corresponding value object, so there is at most one pair in the set with a given key object. You can perform the following operations on maps.

- You can test if a key has a value in the map.
- You can add a new key-value pair to the map.
- You can get the value that is associated with a given key.
- You can change the value that is associated with a given key.

For example, the key objects might be `String`'s that are the English words for numbers. The value objects might be `Integer`'s that are the numbers corresponding to those words. If you give the map an English word, then you can get back its corresponding number.

The maps implemented here use arrays. They work by doing linear search on those arrays. As a result, if a map has n pairs, then its operations may need $O(n)$ key comparisons. However, there are better ways to implement maps, using data structures not yet discussed in this course. These will require only $O(\log n)$ or even $O(1)$ comparisons.

2. Implementation.

You must write a Java class called `Map` that implements a map. To simplify grading, your class must use the same names for things that are given here. Your class `Map` must have two class parameters, `Key` and `Value`, so it looks like this. Here `Key` is the type of the map's key objects, and `Value` is the type of the map's value objects.

```
class Map<Key, Value>
{
  ⋮
}
```

Within the class `Map`, you must have two `private` arrays called `keys` and `values`. The array `keys` must be an array whose base type is the class parameter `Key`. The array `values` must be an array whose base type is the class parameter `Value`. Suppose that a key object *k* is found at the index *i* in the array `keys`. Then the value object associated with *k* is found at the same index *i* in the array `values`. Do not try to use only one array, because that will not work.

You must also have a `private` integer variable called `count` that records how many elements of the arrays are in use. You may also need other `private` variables that are not mentioned here.

Your class must have the following methods. Most of them use `count`, `keys`, and `values` somehow. Some methods are `public` and others are `private`. The `private` methods are helpers for the `public` methods; they will make your code easier to write. Also, in all methods, both key and value objects may be `null`. This may affect how you test if key objects are equal.

`public Map(int length)`

> Constructor. If `length` is less than 0 then you must throw an `IllegalArgumentException`. Otherwise, set `count` to 0, and make a new empty `Map` whose `keys` and `values` arrays have `length` elements. (Recall that you must make arrays of `Object`'s, then cast them to the appropriate types.)

`public Value get(Key key)`

> Return the value that is associated with `key`. Search the array `keys` for an object that is equal to `key`. If that object is at some index in `keys`, then return the object at the same index in the array `values`. If there is no object equal to `key` in `keys`, then throw an `IllegalArgumentException`.

`private boolean isEqual(Key leftKey, Key rightKey)`

> Test if `leftKey` is equal to `rightKey`. Either or both may be `null`. This method is necessary because you must use == when `leftKey` or `rightKey` are `null`, but you must use the `equals` method when both are not `null`. (Recall that `null` has no methods.)

`public boolean isIn(Key key)`

> Test if there is an object in the array `keys` that is equal to `key`.

`public void put(Key key, Value value)`

> Associate `key` with `value`. Search the array `keys` for an object that is equal to `key`. If that object is at some index in `keys`, then change the object at the same index in `values` to `value`. If there is no object in `keys` that is equal to `keys`, then add `key` to `keys`, and add `value` at the same index in `values`. If `keys` and `values` are full, so you cannot add `key` and `value`, then throw an `IllegalStateException`.

`private int where(Key key)`

> Search the array `keys` for an object that is equal to `key`. Return the index of that object. If there is no object equal to `key` in `keys`, then return −1.

The file **`tests.java`** on Moodle contains Java code that performs a series of tests. Each test calls a method from your class `Map`, and prints what the method returns. Each test is also followed by a comment that tells how many points it is worth, and what must be printed if it works correctly.

3. Deliverables.

Run the tests, then turn in the Java source code for the class `Map`. Your lab TA will tell you how and where to turn it in. Your work will be due after Spring Break. If your lab is on **Tuesday, March 7**, then your work must be turned in by **11:55 PM** on **Tuesday, March 21**. If your lab is on **Wednesday, March 8**, then your work must be turned in by **11:55 PM** on **Wednesday, March 22**. You are not allowed to work on this assignment, or even think about it, during Spring Break (☺).