Computer Laboratory 11

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Introduction

This laboratory assignment involves implementing our first data structure (in lecture), called a *map*. A map is a data structure very similar to python's dictionary (some would argue they are two versions of the same abstract data type). Both data structures associate key objects with their associated value objects, and allow the value associated with a key to be set or retrieved. You will implement map by writing a Java class that uses two related arrays. As more key-value pairs are added more of the array will be used.

From an educational standpoint, in this lab you will:

- Apply what you've learned about array-based data structures.
- Build a useful, generic datastructure
- Gain more experience working with Java's Generics.

2 DEADLINES FOR THIS LAB

This lab will be due on April 22 at Noon.

3 Files

This lab will involve the following files:

- `Map.java` – **you write this** – This class implements a Map using two arrays.
- `MapTest.java` – **provided** – This is a test class for the Map
- `Point.java` – **provided** – This class implements the basic features needed to make MapTest work.

4 Map ADT

A map is a set of *key-value* pairs. Each key is said to be *associated* with its corresponding value, so there is at most one pair in the set with a given key. 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.

Using java generics – you can use any java class as the type for keys and any java class as the type for objects. For example, the keys might be `String`'s that are the English words for numbers. The values might be `Integer`'s[1] 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 for this assignment will be implemented using two **private** array variables. The first will be a keys array - it will store the keys. The second will be the values array - it will store the values. These two arrays will be the same length, and the key at a position in the keys array has its associated value at the same position in the values array. For example `keys[3]` might be "Hundred" and `values[3]` might be the `Integer 100`.

The key and value arrays will not be sorted, therefore many of the Map operations you need to implement will require a linear scan. If there are n key/value pairs in the Map, then most operations on this implementation of Map should require $O(n)$ comparisons. Later in this class we will see better ways to implement these methods, which will lead to $O(\log n)$ performance, or even $O(1)$ performance.

5 Requirements

You must write a Java class called `Map`. To simplify grading, you must use the same class and function names as given in this document. Your class must have two java generics type parameters `Key` and `Value` (where `Key` represents the type of the keys and `Value` represents the type of the map's values). By having these two type parameters your object will be much more generic and generally useful. The programmer using your Map class will be able to use any java Object as a key or value (making that choice as necessary for their program)

Here is what your class declaration should look like

```
public class Map<Key, Value> {
  // Code goes here.
}
```

[1] note, `int` is a primitive type and cannot be used with java generics

Within the class `Map`, you must have two **private** arrays called `keys` and `values`. The `keys` array must be an array whose base type is the class parameter `Key`. The `values` array must be an array whose base type is the class parameter `Value`. If a given key k is found at index i in the `keys` array, then the associated value can be found in the `values` array at index i. Do not try to use one array. Without a lot of unnecessary programming, this will not work.

You must also have a **private** integer variable called `count` that records how many elements of the arrays are being used. Generally this will equal the number of distinct keys in the Map. You may also need other private variables that we have not mentioned, you are free to add any such variables that you need. Note, however, that unnecessary private variables (those whose value can be easily computed from other variables, or those who could be completely local to one method) will be treated as a code style violation.

Your class must have the following methods. Most of them will use `count`, `keys`, and `values` in some way. Some of these methods are public, and directly implement the Map behaviors described in the previous section. Others are private, these are included as useful helper methods that will make the public methods easier to write.

One final note, both keys and values may be the special java value `null`. Your code should handle this case. You may need to handle this case separately from others to have the correct behavior.

- `public Map(int length)` Constructor. This constructor should initialize the `keys` and `values` arrays to have `length` elements. (Remember that you must actually create an Object array and then cast to the appropriate types in java) Don't forget to initialize any and all other instance variables as necessary. If a length less-than or equal-to 0 is given, initialize the arrays to have 1 element.

- `private boolean isEqual(Key leftKey, Key rightKey)` Test if `leftKey` is equal to `rightKey`. Either or both may be null. The major purpose of this method is to help you handle the different ways of checking equality. If one of the keys is null then you must use `==` because null has no methods. However, if neither key is null, you must use the `equals` method.

- `private int getIndex(Key key)` Search the array `keys` for an element that is equal to `key`. Return the index of that element. If there is no element equal to `key` in `keys`, then return -1.

- `public Value get(Key key)` Search the `keys` array for an element that is equal to `key`. If that element is at some index in `keys`, then return the element at that same index in the `values` array. If the key is not in the `keys` array then return null. (Remember that equality checking between two Object types in Java **should not** use the `==` operator.)

- `public boolean containsKey(Key key)` Test if there is an element in `keys` that is equal to `key`.

- `public void put(Key key, Value value)` Search the array `keys` for an element that is equal to `key`. If that element is at some index in `keys`, then change the element at the same index in `values` (effectively re-associating the key with a new value). If there is no element in `keys` that is equal to `key` then add `key` to `keys` and `value` to `values` at the same index. If both arrays are full, then you should expand the arrays following the procedure we saw in lecture:

  1. Make a new keys and values arrays – these should be twice as big as the old arrays
  2. copy data from the old arrays to the new arrays
  3. re-assign the keys/values array-variables to store the new keys/values arrays.
  4. add the new key/value pair to the arrays as normal.

- `public int size()` This should return the number of distinct keys in the Map.

When writing your program Intellij will warn you about "unchecked or unsafe operations" this is likely due to the object array cast in your constructor and can be safely ignored. This is a side-effect of how java actually implements generics.

6 Java Autograder Notes

As before - for the autograder to run, your code must 1) compile and 2) not have a package statement.

7 Submission

For this lab we expect only one file to be submitted: `Map.java`. Your submission must be entered into gradescope within 7 days of the start of this lab, although it can be up to 24 hours late for a 10% deduction.

Grading will be on the following general rubric:

- 50 points autograder tests and are primarily based on the provided test files (or rather the provided test files are based on the autograder tests...)
- 10 points code style
  - We expect a javadoc for each function longer than one line (yes, even if it's overriding another function)
  - You should have a comment at the top of the file with you/your partners name
  - Other than that – the code should be generally readable, well tabbed, etc.
- 40 points manual code review. The following is an approximate point breakdown for this code review.

- 5 points Constructor, private variables, etc.
- 4 points `public Value get(Key key)`
- 3 points `public boolean containsKey(Key key)`
- 15 points `public void put(Key key, Value value)`
- 3 points `public int size()`
- 10 points private methods (2 are expected. since they are private you can adjust the names/parameters a bit if you want.)