Programming Skills (Design Patterns and C#)
CSCI-541 / CSCI-641

08/29/2013

Lab 1: Introduction (Hash Table)

Due: Monday, September 9, 2013 at 11:59pm

1 Goals

Familiarize yourself with the basic object-oriented features of the C# language by working on a simple project that you should already be familiar with.

2 Overview

Write a class that implements a chained hash table. The class must conform to the Table interface provided. In addition you must modify the TableFactory class in the test program to use your implementation. Finally you must provide your own unit test method for your hash table.

3 Design

Hash tables are one way of implementing a simple associative storage table. Associative means that each value stored is associated with a key – in our case, a unique key. Specifically, hash tables accomplish this by providing an array, and then using a hash function on the key to decide at what location the value should be. We are calling that location a bucket. Collisions happen when more than one needed key hashes to the same bucket. In your case you are to handle collisions by attaching some kind of collection data structure to each bucket (chaining) rather than searching for an alternative bucket (open addressing) in which to store the value.

Assume that the key class has implemented Equals and GetHashCode appropriately. See "Notes To Implementors" in System.Object.GetHashCode. If you do not see the Note to Implementors, be sure to click "Other Versions" and Select .Net Framework 4.0. If you read the documentation for Table.put you'll see that the method always succeeds. To realize this property and still maintain good performance, you are expected to be able to rehash, i.e., expand your table and reassign the keys to new buckets when the load gets above a certain fraction. Load is defined as the ratio of the number of entries in the table (not number of buckets) to the size of the array. It is supposed to indicate the likelihood of a collision. The more likely a collision, the harder it is to claim that table operations occur in O(1) time.

For the sake of this exercise, expansion is fixed at +50% for each time rehashing is done.

4 Testing

The **Table** interface, along with an exception definition, a table factory, and a simple test program is provided in Main.cs. More extensive testing will be applied during grading, so the test program you write is very important. You are allowed to share test programs with other members of the class, and in fact, many people may submit the same test program as long they are credited in the comments.

5 Implementation

Call your implementing class **LinkedHashTable**. Write the test code in a class called **TestTable** that has a method

```
public static void test()
```

You should not need to include any other assemblies for this project, other than the 2 provided in the main file. Additionally, for full credit, you must make use of generics.

All of your code should be in a single file called **Solution.cs**.

6 Submission

Submit your file **Solution.cs** to the dropbox in myCourses. There is a rubric available for this assigment, which you should review.