Hashing Lab

When you use one of the hash table classes in the Java library (example: HashSet, HashMap, HashTable), the hash table class stores your data based on hashing some key. These hash table classes rely heavily on two methods that should be implemented in the class for the key. These two methods are .equals and .hashCode, and they have the signatures below:

public int hashCode() // returns an integer

public boolean equals(Object obj) // returns true if this object and the parameter obj are equal

**BACKGROUND:**

The hash table classes insert an item by calling the hashCode() method on the key. The hashCode() method may return any integer. The hash table class uses that integer, along with the size of the table, to determine the index where the item should go. It searches a linked list of all the items that have hashed to that same index to see if the item is already there, using the .equals method to compare the new key to each key in the list at that index. Java’s hash tables do not allow duplicate keys. If the item is not there, it is inserted in the linked list at that same calculated index.

The hash table classes search for items by calling the hashCode() method on the key. They use the integer returned by hashCode() along with the size of the table to determine the index where the key was previously stored (the same way it determined where to insert the item.) It searches the linked list at that index, traversing the list and calling the .equals method to find a matching key, if it is there.

The hashCode() and equals() methods are included in the Object class, but they should be overridden by a class whose objects will be keys in a hash table.

Many times when hash tables are used, the key will be an Integer or String or Double. Those classes in the Java library have both equals methods and hashCode methods already written.

If you want to store keys that are objects of your own classes, you need to make sure that you implement these methods. They are usually fairly simple to implement.

**IMPLEMENTING AN EQUALS METHOD IN YourClass:**

**public boolean equals (Object obj) // make sure your parameter is of type Object**

**{**

**if (obj instanceof YourClass) { // make sure object is the right type; otherwise return false**

**YourClass data = (YourClass) obj; // cast obj to some variable of the right type**

**// check data’s fields with your fields,**

**// returning true if they match, false if not.**

**}**

**return false; // obj wasn’t of type YourClass**

**}**

example: suppose I have a class Course with the following two fields. Look at the .equals method.

public class Course

{

private int courseCode; // a 5-digit integer

private String semesterCode; // a 4-digit String

public boolean equals (Object obj)

{

if (obj instanceof Course) {

Course course = (Course) obj;

return course.courseCode == courseCode && course.semesterCode.equals(semesterCode);

}

return false;

}

**IMPLEMENTING A HASHCODE METHOD IN YourClass:**

**The goal of a good hashCode() method is to return different integers for different objects. Let your integer calculations span the integers MIN\_VALUE to MAX\_VALUE, if necessary, to achieve this. You perform some function on the fields in your class to construct the integer. Your goal is to minimize collisions, and to especially keep the lengths of the linked lists minimized (you don’t want lots of keys hashing to the same index).**

public int hashCode() // make sure your method signature matches

{

// use the String’s hashCode() method to give an integer for the semesterCode and put it together

// with the int courseCode

return semesterCode.hashCode() \* 100000 + courseCode;

}

**Assignment**

1. **On the student machine, copy the files from /u/css/classes/3460/Fall2013/HashingLab to your account.**

**cp /u/css/classes/3460/Spring2015/HashingLab/\* .**

1. **Add equals and hashCode methods to three classes: MyString, Fraction, and EightState**
2. **Run the TestHashing class and see how your methods do.**
3. MyString Class:

Fields: MyString just has one field – a String called MyString.

1. equals method: You may call the String’s equal method as part of yours, if you choose.
2. hashCode() Your job is to see if you can find an even better hashCode() method than the Java library uses. Do not call the String’s hashCode method. Use the ASCII values in myString to create your integer. The book’s code can provide some hints, but use the correct hashCode() signature that has no parameters. Make sure that your hashCode() method is spelled with a capital C for Code.
3. Fraction Class:

There are three fields: int numerator and denominator and boolean defined. **The fractions are always kept in reduced form, so you don’t have to worry about that**. If the fraction is not defined, then its numerator and denominator could be anything.

1. .equals: equal fractions have equal numerators and denominators. Two undefined Fractions are equal. If one fraction is undefined and the other is defined, then they are not equal.
2. .hashCode(). Find some way to turn the fields into a single int. Equal fractions should have equal hashCodes. Make sure that all undefined fractions get the same hashCode.
3. EightState: See this site that describes the game of eight sliding puzzle: <http://mypuzzle.org/sliding>

There is just one field – board: a two-dimensional 3x3 array of ints. The digits 0..8 will be in the field. Zero stands for the blank tile.

* 1. .equals: two EightStates are equivalent if the array values are the same in each location
  2. .hashCode(): see if you can turn a 3x3 array of ints into one int. Hint: a really bad idea would be to add all the entries together – the sum would always be the same.

**Testing your code:** Run the program TestHashing and choose from the menu. The file for testing MyString is hash.dat. It has over 20000 words in it.

This lab is due on Friday, April 10th.