CIT 483/583 Lab – Collections

Fall 2018

**Instructions**

Open and save this file in any MS word-compatible format as Lab03\_*Firstname*\_*Lastname*.<ext> and place your answers in that document. Do a **Save-As** and retain all of my content. Keep the document safe in case your submission fails, or you discover an error prior to the due date and wish to re-submit. Submit your document to the Lab03 dropbox in Canvas. The due date and any other pertinent information are noted in the Canvas item.

This lab should be completed on students.cs.nku.edu as a reference implementation and as a means of testing your answers. It can be done using IRB only, or you may create .rb files to test some or all of your code, but do not submit .rb files (programs).

**Place the results of the command or other answers in or immediately following each question and make sure your answers stand out from the questions by using a different font color.**

1. The *hash* method is implemented in class Object and is used to generate a Fixnum value that uniquely places the object in a 'bucket'. It does not necessarily mean it's the same object, so it's not always a unique identifier.

>> obj1 = Object.new

>> obj1.hash

>> obj2 = Object.new

>> obj2.hash

1. As stated in the documentation for the hash method, a.eql?(b) implies a.hash == b.hash.

In other words, the hash method for any two built-in objects that are considered equivalent should produce the same Fixnum value. Our two Object instances don't have any properties per se that would make them equivalent, and we would not expect them to be so.

>> obj1 == obj2

>> obj1.eql?(obj2)

>> obj1.hash == obj2.hash

1. On the other hand, many built-in types have this property, including Strings.

>> s1 = "test"

>> s2 = "test"

>> s1 == s2

>> s1.eql?(s2)

>> s1.hash

>> s2.hash

>> s1.hash == s2.hash

1. Arrays, for example, that "look alike" also have this property.

>> ary1 = [1, 2, 3, 4]

>> ary2 = [1, 2, 3, 4]

>> ary1 == ary2

>> ary1.eql? ary2

>> ary1.hash

>> ary2.hash

1. Now we can revisit the idea proposed in the lecture slides as to "what does it mean to be a good Hash key?" We can create a Hash with obj1 and obj2 as keys, and see that each slot in the Hash is distinct.

>> obj1 = Object.new

>> obj2 = Object.new

>> h = Hash.new

>> h[obj1] = "Value 1"

>> h[obj2] = "Value 2"

>> h

1. We can surmise that s1 and s2 would behave differently, since it is the case that s1.eql?(s2) is true and s1.hash == s2.hash. The same would be true of ary1 and ary2 as defined above.

>> s1 = "test"

>> s2 = "test"

>> h1 = Hash.new

>> h1[s1] = "Value 1"

>> h1[s2] = "Value 2"

>> h1

>> ary1 = [1, 2, 3, 4]

>> ary2 = [1, 2, 3, 4]

>> h2 = Hash.new

>> h2[ary1] = "value 1"

>> h2[ary2] = "value 2"

>> h2

In both cases, above, notice that the second time we use keys that "look the same" causes the value to be overwritten. **In other words, Hashes are designed to maintain unique keys.**

1. We can now develop the idea of a custom object that is a good (or bad) Hash key. The first attempt turns out to be a "bad" Hash key, since the keys "look the same" and are eql?, but yet refer to different slots in the Hash.

>> class HashKeyCandidate1

attr\_accessor :value

def initialize(value)

self.value = value

end

def eql?(other)

return other.value.eql?(self.value)

end

def ==(other)

return other.value == (self.value)

end

end

>> hk1 = HashKeyCandidate1.new(5)

>> hk2 = HashKeyCandidate1.new(5)

>> hk1 == hk2

>> hk1.eql?(hk2)

>> h = Hash.new

>> h[hk1] = 'value1'

>> h[hk2] = 'value2'

Use an iterator method of your choice and a block to show many key-value pairs h contains at this point.

1. The second attempt implements a hash function that is based on the *value* field, which is a simple Fixnum when the objects are constructed, and thus is itself a "good" Hash key

>> class HashKeyCandidate2

attr\_accessor :value

def initialize(value)

self.value = value

end

def eql?(other)

return other.value.eql?(self.value)

end

def ==(other)

return other.value == (self.value)

end

**def hash**

**return value.hash**

**end**

end

>> hk3 = HashKeyCandidate2.new(5)

>> hk4 = HashKeyCandidate2.new(5)

>> hk3 == hk4

>> hk3.eql? hk4

>> hk3.hash == hk4.hash

>> h = Hash.new

>> h[hk3] = "Value 1"

>> h[hk4] = "Value 2"

>> h.size

1. The last minor issue with this class is its rather ugly printed format when we use it as a Hash key. Let's create one more version that will display its value in a more succinct format.

>> class HashKeyCandidate3

attr\_accessor :value

def initialize(value)

self.value = value

end

def eql?(other)

return other.value.eql?(self.value)

end

def ==(other)

return other.value == (self.value)

end

def hash

return value.hash

end

def inspect

return value

end

end

>> hk5 = HashKeyCandidate3.new(5)

>> hk6 = HashKeyCandidate3.new(5)

>> h = Hash.new

>> h[hk5] = "Value 1"

>> h[hk6] = "Value 2"

>> h.size

>> h