Java Hash Code and Hash Map

Raydhitya Yoseph 13509092
Informatics Engineering
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
raydhitya.yoseph@gmail.com

***Abstract*—*This paper contains explanation about how Java Object hashCode() and equals() work, and how both methods implemented in few Java classes, lastly how hash code used in HashMap. HashMap need good hashCode() implementation with evenly distributed integers, further work needs to be done on HashMap performance test, and hash function in Java and cryptography terms are different in computational complexity to produce its result, but retain the same concept of representing arbitrary data into fixed length data.*

***Index Terms*—*hash function, hash code, Java, hash map.*

I. Introduction

Hash function in cryptography is a function which map arbitrary string into fixed length string. Hash value produced by hash function is very sensitive. One character modification to the string produces totally different hash value.

The main function of hash function is to represent arbitrary data into fixed length data. In Java programming every objects has a hash function which produce the objects hash value. The hash function in Java different in term of complexity with hash function in cryptography. The hash function only do simple calculation to produce its value.

This paper explains how hash function implemented for few popular Java classes and the function impact on hash map.

II. Java Object

All printed material, including text, illustrations, and charts, must be kept within a print area of 17 cm wide by 25 cm high. Do not write or print anything outside the print area. All *text* must be in a two-column format. Columns are to be 8.25 cm wide, with a 0.5 cm space between them. Text must be fully justified.

Java is object oriented programming language. It has root object in which every other objects inherits from. The root object class name is Object. The Object class provides two methods which are hashCode() and equals() respectively.

The hashCode() function returns an integer which represents the object's hash value. The function is marked with native keyword which means its implementation is written in native language. The function implementation is dependents on Java Virtual Machine implementation.

The equals() function accept an Object and returns true if given object equals with the caller object and return false otherwise. How to determine two objects equality? The answer is depends on the programmer. For example one class has two integer attributes. Programmer can implements equals() return true if other object's two attributes have the same value. Programmer also can implements equals() return true if only other object's one attribute has the same value.

There are for rules which need to be followed regarding equals for non-null object references.

1. Reflexive: x.equals(x) should return true;
2. Symmetric: x.equals(y) should return true if and only if y.equals(x) returns true;
3. Transitive: if x.equals(y) true and y.equals(z) true then x.equals(z) should return true;
4. Consistent: multiple invocations of x.equals(y) consistently return true or consistently return false;
5. x.equals(null) should return true.

Next there are two rules related to equals() and hashCode().

1. If two objects equal according to equals(), then calling hashCode() on both objects must produce the same result;

2. If two objects not equals according to equals(), it is not required calling hashCode() on both objects produce different result. However, producing different result for unequal objects may improve the performance of hash tables.

III. Number and String

This part discuss about hashCode() and equals() implementation on Java primitive wrapper classes and String.

A. Number

Each Java primitives has wrapper class for them. For example integer has Integer class, float has Float class, and double has Double class. Wrapper classes provide utility functions such as transforming one type into another type.

Instance of the wrapper class contains the given value. For example instantiating new Integer class with 0 will make the object has 0 as its value. As seen on figure 1, all the classes inherit from Number class which provide methods to retrieve the object value to all number primitives.

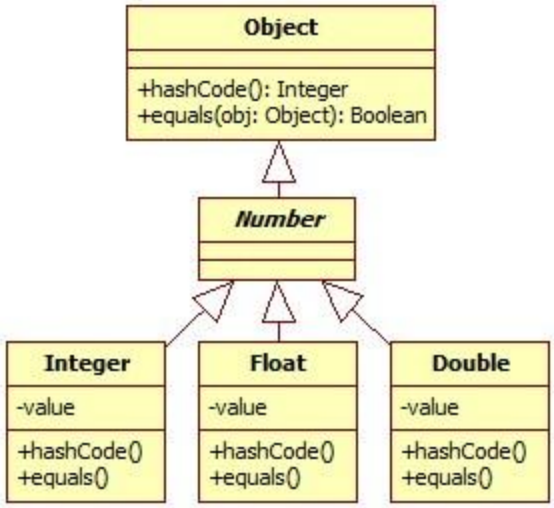

Figure 1 Number Class Diagram

The hashCode() and equals() implementation for all the wrapper classes are very simple. All hashCode() implementations transform the object value into integer representation. That means Integer class hashCode() implementation simply return its value. Float hashCode() implementation transform its float value into integer.

All equals() implementation also very simple. All the implementations go as follows first check whether given object is an instance of corresponding class or not. If it is, cast the object to the corresponding class and get its value. Two objects equal if both have the same value for its value attribute.

B. String

String is widely used data type in many programming languages. Java String class provide utility methods and store string value in character array.

String has an attribute called hash initialized with 0. String hashCode() implementation iterate its character array and compute h = 32 * h + array element, where h initialized with hash attribute. String equals() implementations check given String character array are the same in length and the same in characters.

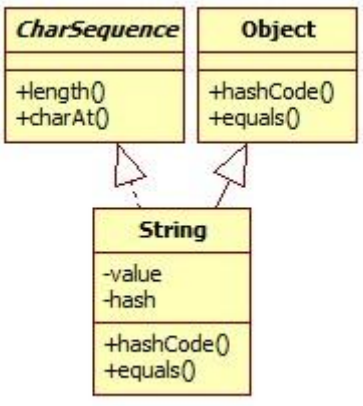

Figure 2 String Class Diagram

IV. COLLECTIONS

Java provides collection classes which contains more than one other objects and have methods manipulating its elements. The collection classes is widely known as Java Collection Framework.

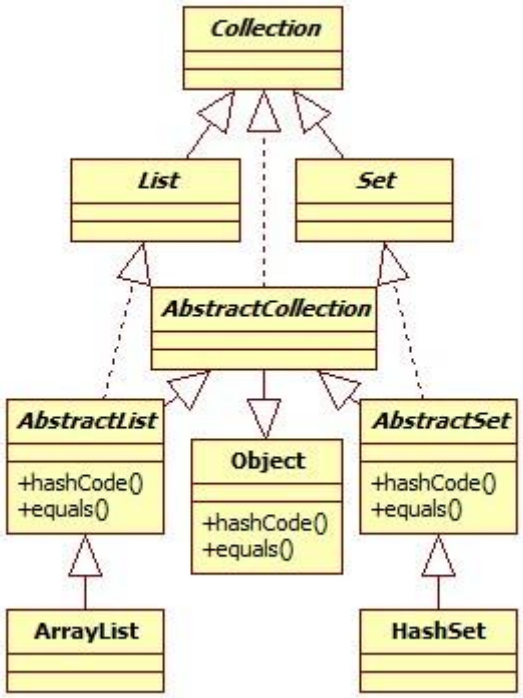

Figure 3 Collection Class Diagram

A. Collection Interfaces

Collection Framework designed hierarchically with Collection as root interface as seen in figure 3. Collection interface is simply collection of objects which is not widely used interface. List and Set which inherits from Collection are more often used.

List is an ordered collection. The user of List interface has precise control over where in the list each element is inserted. The user can access elements by their integer index and search for elements in the list. Set is a collection that contains no duplicate elements. As implied by its name, Set interface models the mathematical set abstraction.

B. Collection Implementation

AbstractCollection class provides a skeletal implementation of the Collection interface, to minimize the effort required to implement this interface. AbstractCollection inherits from Object and does not override hashCode() and equals().

AbstractList inherits from AbstractCollection to provide skeletal List implementation. AbstractSet inherits from AbstractCollection to provide skeletal Set implementation. Both classes override hashCode() and equals().

Both classes implements hashCode() in a simple manner. The implementation iterate all the collection elements and sum their respective hashCode(). The sum result is the collection hashCode().

For equals() implementation the two classes implement it differently. The main idea is to check if two objects of same collection have same elements. AbstractList iterate all its elements using iterator and check whether current element of both AbstractLists equal or not. If all elements equals, two objects of AbstractList are equal.

AbstractSet cannot iterate its elements in the same manner as AbstractList because its elements are unordered. AbstractSet iterate all elements of one object and for each element it search that element in the second AbstractSet object. So, that means AbstractSet equals() method complexity is $O(n^2)$ and AbstracList equals() method complexity is O(n).

V. HASHMAP

Hash map or hash table is a data structure that map keys to values. Hash table uses hash function to compute an index an array of buckets. In many situations, hash table turn out to be more efficient that search trees or any table lookup structure. Java implements hash table in a class called HashMap.

A. Map Interfaces

Java provides Map as root interface, as seen in figure 4, for classes that map keys to values. Map interface has Entry interface that represent key value mapping stored in the Map.

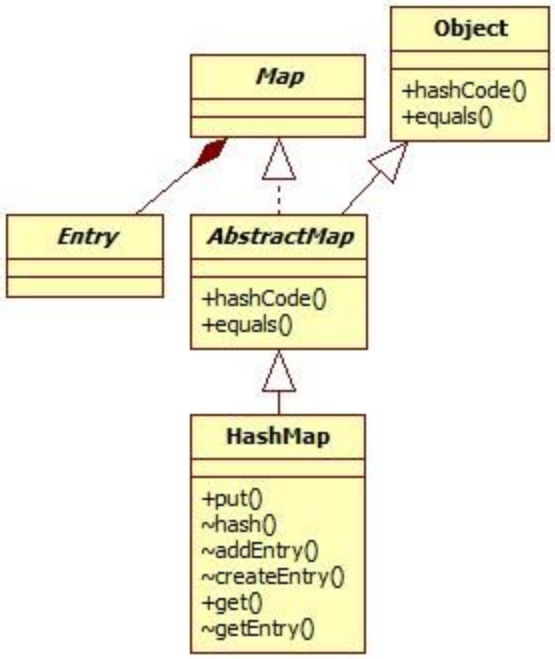

Figure 4 Map Class Diagram

B. Map Implementation

AbstractMap class provides a skeletal implementation of the Map interface. Like AbstractList and AbstractSet, AbstractMap overrides hashCode() and equals() method. Also like both previous collections, AbstractMap iterate all its elements for equals() and hashCode() implementations.

AbstractMap use Set view of its elements to iterate its elements. AbstractMap has entrySet() method which returns its elements viewed as Set. The Set view then used for iterating Map's elements and used for equals() and hashCode() implementation. The hashCode() implementation simply sum all the Map's elements' hash code. The equals() implementation same as AbstractSet implementation which uses one AbstractMap object as iteration base and search for current element in the other AbstractMap object thus make its complexity $O(n^2)$.

C. Hash Code and Hash Map

What interesting is how HashMap used hashCode() to store its elements. HashMap stores its elements in array of buckets. The buckets use key hash codes as its index and is a linked list of Entry.

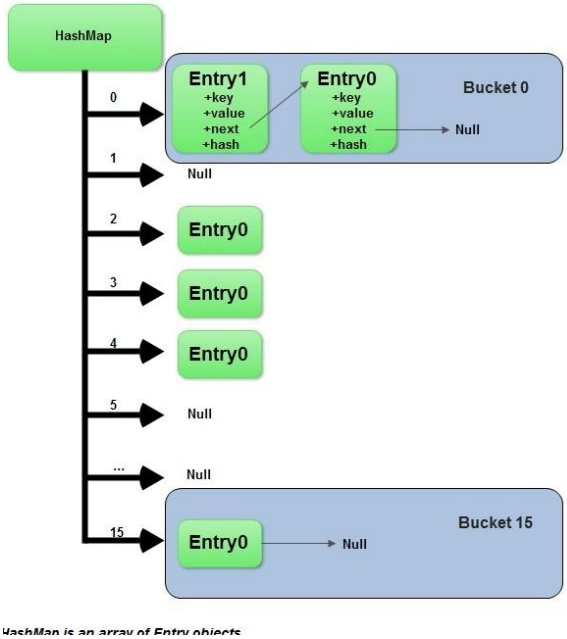

Figure 5 HashMap Implementation

HashMap has array of Entry attribute as the bucket representation. When HashMap put() method called, it hashes the key with its internal hash function to defend against poor quality hash function. Next, it calls indexFor() function to find the bucket index. The indexFor() method performs logical and with current buckets length minus one.

Using bucket index from indexFor() function HashMap check the bucket. There are two condition that will happen. First is if the bucket first element is not null and second is if the bucket first element is null. If the element is not null, HashMap iterate the bucket linked list. If it founds Entry element with same hash code and that Entry key equal per equals() method with provided key by the

put() caller, it will replace old value with same value. Remember that unequal objects per equals() method is not required to have different hash code. So, it is possible two not equal object have same hash code. That is what called as collision.

The second condition is the bucket first element is null or HashMap does not find satisfying key condition per the first condition. It will call addEntry() to check current HashMap size is equal or more to threshold. Threshold is defined by default load factor which is 0.75. If the HashMap need bigger size, it will resize twice as big as current bucket size. After resizing it will call createEntry() to really add the Entry to the linked list. The method will retrieve current first element of the bucket, create new Entry with given key and value, points the new Entry next element to the retrieved Entry, assign it as the bucket first element, and finally increment the HashMap's size.

Retrieving elements is easier. When get() method called, HashMap will hash the given key with its own hash function which is the same one used in put(), to find the bucket index. The getEntry() method will use the index it will search the bucket to find key with same hash code and equals. If it find key with satisfying condition, it will return the Entry and the get() method will return the value.

VI. Experiment and Result

As knowing HashMap operates using hashCode() it is interesting to find out what the hashCode() impact on the Map. By simple observation if all the keys being put into the HashMap have same hash code, the HashMap will degenerate to linked list. So, a good hash code is needed for good HashMap performance.

The experiment use Java Integer and String classes as keys for the HashMap. Two method used to provide the Integer object. First is using Java Random class and second is using Java SecureRandom class. Only one method to provide String object which is using UUID class.

The experiment will generate 1000000 random keys and put it into the map with the same value. To determine each random method effectiveness in providing distinct hash code this paper counts how many bucket really used. Number of entries divided by number of used bucket produces average entries per bucket. The closer the average to 1 means the less keys with same hash code produced.

Each method is tried 5 times and the result is shown in table 1.

	Random	SecureRandom	UUID
Average	1.256665	1.256693	1.257033

Table 1 Experiment Result

The result for all three methods are very similar. Using random integer from Random and SecureRandom will yields average 1.256 Entry per bucket. Using random string from UUID will yield average 1.257 Entry per bucket.

VII. Conclusion

Seven things can be concluded. First, Java Object provide hashCode() and equals() method which designed to be overridden by other classes. Second, Java Integer, Float, and Double classes provide simple hashCode() implementation which return their values in integer.

Third, Java AbstractList and AbstractSet use their elements hash codes for their hashCode ()implementation. They also check all their elements equality for their equals() implementation.

Fourth, Java HashMap is a data structure which use hashCode() heavily. Java HashMap implementation use Entry class for key value pair. Hash Map store its element using bucket. HashMap use indexFor() to determine where to store each entry in the bucket using their hashCode() function.

Fifth, HashMap needs good hashCode() implementation that produce evenly distributed integers for good performance. Sixth, further work needs to be done on performance test using heavy object for HashMap value.

Seventh, hash function in Java and cryptography terms are different in computational complexity to provide its result, but retain the same concept of representing arbitrary data with fixed length data.

References

[1] http://en.wikipedia.org/wiki/Hash_table retrieved at 18 May 2013.
[2] http://stackoverflow.com/questions/11810394/regarding-hashmap-implementation-in-java retrieved at 18 May 2013.
[3] http://stackoverflow.com/questions/1757363/java-hashmap-performance-optimization-alternative retrieved at 19 May 2013.
[4] http://stackoverflow.com/questions/41107/how-to-generate-a-random-alpha-numeric-string retrieved at 19 May 2013.
[5] Java Development Kit Source Code.

Declaration

I hereby declare the paper is my own writing, not an adaptation, nor translation from another person paper, and nor a form of plagiarism.

Bandung, 19th May 2013

Raydhtiya Yoseph 13509092