HashSet in Java

HashSet:

- Implements Set Interface.
- Underlying data structure for HashSet is hashtable.
- As it implements the Set Interface, duplicate values are not allowed.
- Objects that you insert in HashSet are not guaranteed to be inserted in same order. Objects are inserted based on their hash code.
- NULL elements are allowed in HashSet.
- HashSet also implements Searlizable and Cloneable interfaces.

Constructors in HashSet:

```
HashSet h = new HashSet();
Default initial capacity is 16 and default load factor is 0.75.

HashSet h = new HashSet(int initialCapacity);
default loadFactor of 0.75

HashSet h = new HashSet(int initialCapacity, float loadFactor);

HashSet h = new HashSet(Collection C);
```

What is initial capacity and load factor?

The initial capacity means the number of buckets when hashtable (HashSet internally uses hashtable data structure) is created. Number of buckets will be automatically increased if the current size gets full. The load factor is a measure of how full the HashSet is allowed to get before its capacity is automatically increased. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is rehashed (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

```
                  Number of stored elements in the table
    load factor = --------------------------------------
                        Size of the hash table
```

E.g. If internal capacity is 16 and load factor is 0.75 then, number of buckets will automatically get increased when table has 12 elements in it.

Effect on performance:
Load factor and initial capacity are two main factors that affect the performance of HashSet operations. Load factor of 0.75 provides very effective performance as respect to time and space complexity. If we increase the load factor value more than that then memory overhead will be reduced (because it will decrease internal rebuilding operation) but, it will affect the add and search operation in hashtable. To reduce the rehashing operation we should choose initial capacity wisely. If initial capacity is greater than the maximum number of entries divided by the load factor, no rehash operation will ever occur.

Important Methods in HashSet:

1. **boolean add(E e)** : add the specified element if it is not present, if it is present then return false.
2. **void clear()** : removes all the elements from set.
3. **boolean contains(Object o)** : return true if element is present in set.
4. **boolean remove(Object o)** : remove the element if it is present in set.
5. **Iterator iterator()** : return an iterator over the element in the set.

Sample Program:

```
// Java program to demonstrate working of HashSet
import java.util.*;

class Test
{
    public static void main(String[]args)
    {
        HashSet<String> h = new HashSet<String>();

        // adding into HashSet
        h.add("India");
        h.add("Australia");
        h.add("South Africa");
        h.add("India");// adding duplicate elements

        // printing HashSet
        System.out.println(h);
        System.out.println("List contains India or not:" +
                           h.contains("India"));

        // Removing an item
        h.remove("Australia");
        System.out.println("List after removing Australia:"+h);

        // Iterating over hash set items
        System.out.println("Iterating over list:");
        Iterator<String> i = h.iterator();
        while (i.hasNext())
            System.out.println(i.next());
    }
```

```
}
```

Run on IDE

Output of the above program:

```
[Australia, South Africa, India]
List contains India or not:true
List after removing Australia:[South Africa, India]
Iterating over list:
South Africa
India
```

How HashSet internally work?

All the classes of Set interface internally backed up by Map. HashSet uses HashMap for storing its object internally. You must be wondering that to enter a value in HashMap we need a key-value pair, but in HashSet we are passing only one value.

Then how is it storing in HashMap?

Actually the value we insert in HashSet acts as key to the map Object and for its value java uses a constant variable. So in key-value pair all the keys will have same value.

If we look at the implementation of HashSet in java doc, it is something like this;

```
private transient HashMap map;

// Constructor - 1
// All the constructors are internally creating HashMap Object.
public HashSet()
{
    // Creating internally backing HashMap object
    map = new HashMap();
}

// Constructor - 2
public HashSet(int initialCapacity)
{
    // Creating internally backing HashMap object
    map = new HashMap(initialCapacity);
}

// Dummy value to associate with an Object in Map
private static final Object PRESENT = new Object();
```

If we look at add() method of HashSet class:

```
public boolean add(E e)
{
   return map.put(e, PRESENT) == null;
}
```

We can notice that, add() method of HashSet class internally calls put() method of backing HashMap object by passing the element you have specified as a key and constant “PRESENT” as its value.

remove() method also works in the same manner. It internally calls remove method of Map interface.

```
public boolean remove(Object o)
{
  return map.remove(o) == PRESENT;
}
```

Time Complexity of HashSet Operations:
The underlying data structure for HashSet is hashtable. So amortize (average or usual case) time complexity for add, remove and look-up (contains method) operation of HashSet takes **O(1)** time.

References:
https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html

This article is contributed by **Dharmesh Singh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Articles Java

Recommended Posts:

LinkedHashSet class in Java with Examples
TreeSet class in Java with Examples
Set in Java
Java.util.HashMap in Java
LinkedList in java
Matrix Introduction
Commonly asked questions in Flipkart Interviews
Commonly Asked Questions in Goldman Sachs Interviews
Functional Dependency and Attribute Closure
GATE CS 2016 Syllabus

(Login to Rate and Mark)

3.1 Average Difficulty : **3.1/5.0**
Based on **11** vote(s)

Add to TODO List

Mark as DONE