**Java HashSet**

**Introduction**

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. HashSet extends AbstractSet and implements the Set interface. It creates a collection that uses a hash table for storage. A hash table stores information by using a mechanism called hashing. In hashing, the informational content of a key is used to determine a unique value, called its hash code. The hash code is then used as the index at which the data associated with the key is stored. The transformation of the key into its hash code is performed automatically.

When you put an object into a Hashset, it uses the object's hashcode value to determine where to put the object in the Set. But it also compares the object's hashcode to the hashcode of all the other objects in the Hash Set, and if there's no matching hashcode,the HashSet assumes that this new object is not a duplicate. HashSet finds a matching hashcode for two objects. one you're inserting and one already in the set-the HashSet will then call one of the object's equals() methods to see if these hashcode-matched objects really are equal. And if they're equal, the HashSet knows that the object you're attempting to add is a duplicate of something in the Set, so the add doesn't happen.

A HashSet is an unsorted, unordered Set. It uses the hashcode of the object being inserted, so the more efficient your hashCode() implementation the better access performance you'll get. Use this class when you want a collection with no duplicates and you don't care about order when you iterate through it.

HashSet() // Default constructor

HashSet(Collection c) // It creates HashSet from collection c

HashSet( int capacity) // it creates HashSet with initial capacity mentioned

HashSet(int capacity, float loadFactor) // This creates HashSet with capacity and load factor

Choosing an initial capacity that's too high can waste both space and time. On the other hand, choosing an initial capacity that's too low wastes time by copying the data structure each time it's forced to increase its capacity. If you don't specify an initial capacity, the default is 16.

The HashSet class has one other tuning parameter called the load factor. HashSet size grows as per load factor defined or default double size.

**HashSet Methods**

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| public boolean add(Object o) | Adds an object to a HashSet if already not present in HashSet. |
| public boolean remove(Object o) | Removes an object from a HashSet if found in HashSet. |
| public boolean contains(Object o) | Returns true if object found else return false |
| public boolean isEmpty() | Returns true if HashSet is empty else return false |
| public int size() | Returns number of elements in the HashSet |

**Java Code**

[view plaincopy to clipboardprint?](http://www.w3resource.com/java-tutorial/java-hashset.php)

1. **package** hashset;


5. **import** java.util.HashSet;


9. **public** **class** HashSetDemo {

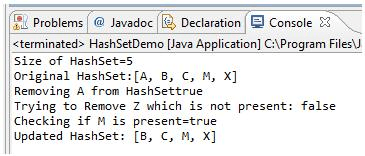

13. **public** **static** **void** main(String[] args) {
15. HashSet<string> hs = **new** HashSet<string>();
17. // Adding element to HashSet
19. hs.add("M");
21. hs.add("B");
23. hs.add("C");
25. hs.add("A");
27. hs.add("M");
29. hs.add("X");


33. System.out.println("Size of HashSet=" + hs.size());
35. System.out.println("Original HashSet:" + hs);


39. System.out.println("Removing A from HashSet: " + hs.remove("A"));
41. System.out.println("Trying to Remove Z which is not present: "
43. + hs.remove("Z"));
45. System.out.println("Checking if M is present=" + hs.contains("M"));
47. System.out.println("Updated HashSet: " + hs);


51. }
53. }
55. </string></string>

**Output**



**Summary**

* HashSet provides collection of unique objects
* HashSet is unsorted, unordered and non-indexed based collection class
* HashSet can have only one Null element