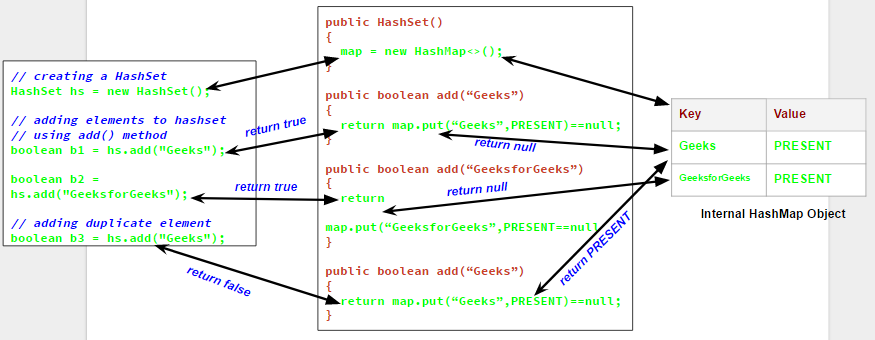
Internal working of Set/HashSet in Java

As we know that a set is a **well-defined** collection of distinct objects. Each member of a set is called an element of the set. So in other words, we can say that a **set will never contain duplicate elements**. But how in java [Set](https://www.geeksforgeeks.org/set-in-java/)interface implemented classes like [HashSet](https://www.geeksforgeeks.org/hashset-in-java/), [LinkedHashSet](https://www.geeksforgeeks.org/linkedhashset-class-in-java-with-examples/), [TreeSet](https://www.geeksforgeeks.org/treeset-class-java-examples/) etc. achieve this uniqueness. In this post, we will discuss the hidden truth behind this uniqueness.

**How HashSet works internally in Java?**

  
We will understand this with an example.Let us see the output of the following program which try to add duplicate elements in a HashSet.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate  // internal working of HashSet    import java.util.HashSet;    class Test  {      public static void main(String args[])      {          // creating a HashSet          HashSet hs = new HashSet();            // adding elements to hashset          // using add() method          boolean b1 = hs.add("Geeks");          boolean b2 = hs.add("GeeksforGeeks");            // adding duplicate element          boolean b3 = hs.add("Geeks");            // printing b1, b2, b3          System.out.println("b1 = "+b1);          System.out.println("b2 = "+b2);          System.out.println("b3 = "+b3);            // printing all elements of hashset          System.out.println(hs);        }  } |

Output:

b1 = true

b2 = true

b3 = false

[GeeksforGeeks, Geeks]

Now from the output, it is clear that when we try to add a duplicate element to a set using *add()* method, it returns *false*, and element is not added to hashset, as it is already present. Now the question comes, how *add()* method checks whether the set already contains the specified element or not. It will be more clear if we have a closer look on the *add()* method and default constructor in HashSet class.

// predefined HashSet class

public class HashSet

{

// A HashMap object

private transient HashMap map;

// A Dummy value(PRESENT) to associate with an Object in the Map

private static final Object PRESENT = new Object();

// default constructor of HashSet class

// It creates a HashMap by calling

// default constructor of HashMap class

public HashSet() {

map = new HashMap<>();

}

// add method

// it calls put() method on map object

// and then compares it's return value with null

public boolean add(E e) {

return map.put(e, PRESENT)==null;

}

// Other methods in Hash Set

}

Now as you can see that whenever we create a HashSet, it internally creates a [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) and if we insert an element into this HashSet using *add()* method, it actually call *put()* method on internally created HashMap object with element you have specified as it’s key and constant Object called **“PRESENT”** as it’s value. So we can say that **a Set achieves uniqueness internally through HashMap**. Now the whole story comes around [how a HashMap and *put()* method internally works](https://www.geeksforgeeks.org/internal-working-of-hashmap-java/).

As we know in a [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) each key is unique and when we call *put(Key, Value)* method, it returns the previous value associated with key, or *null* if there was no mapping for key. So in *add()* method we check the return value of map.put(key, value) method with *null* value.

1. If map.put(key, value) returns *null*, then the statement “map.put(e, PRESENT) == null” will return *true* and element is added to the HashSet(internally HashMap).
2. If map.put(key, value) returns old value of the key, then the statement “map.put(e, PRESENT) == null” will return *false* and element is not added to the HashSet(internally HashMap).

As LinkedHashSet extends HashSet, so it internally calls constructors of HashSet using [super()](https://www.geeksforgeeks.org/super-keyword/). Similarly creating an object of [TreeSet](https://www.geeksforgeeks.org/treeset-class-java-examples/) class internally creates object of [Navigable Map](https://www.geeksforgeeks.org/navigablemap-interface-in-java-with-example/)as backing map.