**How 𝑯𝒂𝒔𝒉𝑺𝒆𝒕 works internally in Java**

HashSet is a class in Java which implements Set Interface and extends AbstractSet. It allows you to store only 𝒖𝒏𝒊𝒒𝒖𝒆 𝒆𝒍𝒆𝒎𝒆𝒏𝒕𝒔 that means no duplicates are allowed.

HashSet is a member of 𝑱𝒂𝒗𝒂 𝑪𝒐𝒍𝒍𝒆𝒄𝒕𝒊𝒐𝒏𝒔 𝑭𝒓𝒂𝒎𝒆𝒘𝒐𝒓𝒌.

Timeline

Description automatically generated

𝑷𝒐𝒊𝒏𝒕𝒔 𝒕𝒐 𝒃𝒆 𝒏𝒐𝒕𝒆𝒅 𝒃𝒆𝒇𝒐𝒓𝒆 𝒖𝒏𝒅𝒆𝒓𝒔𝒕𝒂𝒏𝒅𝒊𝒏𝒈 𝒊𝒏𝒕𝒆𝒓𝒏𝒂𝒍 𝒃𝒆𝒉𝒂𝒗𝒊𝒐𝒖𝒓

➡️ HashSet uses HashMap internally to store data in the form of Key-Value pair.

➡️ Elements are stored using 𝒉𝒂𝒔𝒉𝒊𝒏𝒈 𝒕𝒆𝒄𝒉𝒏𝒊𝒒𝒖𝒆, therefore elements are not stored in an ordered fashion and the elements will be returned in random order.

➡️ Elements are stored in the form of key-value pair (internally by HashMap) where key will the actual element value and value will be the 𝑷𝑹𝑬𝑺𝑬𝑵𝑻 Constant.

Diagram

Description automatically generated

pic references : javaconceptoftheday.com

=>private static final Object PRESENT = new Object();

➡️ This class permits at most 𝒐𝒏𝒆 𝒏𝒖𝒍𝒍 𝒆𝒍𝒆𝒎𝒆𝒏𝒕 because HashMap allows only one null key.

➡️ HashSet has default initial capacity of 16.

➡️ HashSet has default load factor of 0.75 or 75%.

➡️ HashSet does not have any method(like get(key) in HashMap) to retrieve the object directly.

➡️ The only way to retrieve objects from the HashSet is through iterating the entire HashSet. This can be achieved by using iterator, for, for-each ,etc.

➡️ The iterators returned by HashSet class iterator method are fail-fast: if the set is modified at any time after the iterator is created, in any way except through the iterator’s own remove method, the Iterator throws a 𝑪𝒐𝒏𝒄𝒖𝒓𝒓𝒆𝒏𝒕𝑴𝒐𝒅𝒊𝒇𝒊𝒄𝒂𝒕𝒊𝒐𝒏𝑬𝒙𝒄𝒆𝒑𝒕𝒊𝒐𝒏.

Diagram

Description automatically generated

𝑯𝒐𝒘 𝑯𝒂𝒔𝒉𝑺𝒆𝒕 𝒘𝒐𝒓𝒌𝒔 𝑰𝒏𝒕𝒆𝒓𝒏𝒂𝒍𝒍𝒚?  
In order to understand HashSet internal behaviour , one should understand the HashMap internals. This is because HashSet internally uses the HashMap behaviour.

The moment you create a HashSet the default constructor will get called and internally a backing HashMap object gets created automatically.

Set<String> hs = new HashSet<String>();

In HashSet class, a map instance can be created at class level like below

private transient HashMap<E,Object> map;

𝑪𝒐𝒏𝒔𝒕𝒓𝒖𝒄𝒕𝒐𝒓𝒔 𝒐𝒇 𝑯𝒂𝒔𝒉𝑺𝒆𝒕:

➡️ 𝑫𝒆𝒇𝒂𝒖𝒍𝒕 𝑪𝒐𝒏𝒔𝒕𝒓𝒖𝒄𝒕𝒐𝒓: Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).

➡️ 𝑷𝒂𝒓𝒂𝒎𝒆𝒕𝒆𝒓𝒊𝒛𝒆𝒅 𝒄𝒐𝒏𝒔𝒕𝒓𝒖𝒄𝒕𝒐𝒓 𝒘𝒊𝒕𝒉 𝒊𝒏𝒊𝒕𝒊𝒂𝒍 𝑪𝒂𝒑𝒂𝒄𝒊𝒕𝒚: Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).

➡️ 𝑷𝒂𝒓𝒂𝒎𝒆𝒕𝒆𝒓𝒊𝒛𝒆𝒅 𝒄𝒐𝒏𝒔𝒕𝒓𝒖𝒄𝒕𝒐𝒓 𝒘𝒊𝒕𝒉 𝒊𝒏𝒊𝒕𝒊𝒂𝒍 𝑪𝒂𝒑𝒂𝒄𝒊𝒕𝒚 & 𝒍𝒐𝒂𝒅 𝑭𝒂𝒄𝒕𝒐𝒓: Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor.

➡️ 𝑷𝒂𝒓𝒂𝒎𝒆𝒕𝒆𝒓𝒊𝒛𝒆𝒅 𝒄𝒐𝒏𝒔𝒕𝒓𝒖𝒄𝒕𝒐𝒓 𝒘𝒊𝒕𝒉 𝑪𝒐𝒍𝒍𝒆𝒄𝒕𝒊𝒐𝒏: Constructs a new set containing the elements in the specified collection.

𝑳𝒆𝒕𝒔 𝒔𝒆𝒆 𝒉𝒐𝒘 𝒂𝒅𝒅(𝑬 𝒆) 𝒘𝒐𝒓𝒌𝒔:

Adds the specified element to the set if it is not already present and then it returns true. If this set already contains the specified element, no operation would be performed and it returns false.

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

Whenever we try to add an element using HashSet, it stores internally as a key-value pair.

Here Key is an element which we are trying to add using HashSet and Value is a CONSTANT called PRESENT (Same for all the HashSet elements).

Ex: add(“Java”)

In background “Java” will behave like a “key” for the map and PRESENT will be it’s value. So, add(“Java”) is similar to map.put(“Java”,PRESENT).

return map.put(e, PRESENT)==null ===> returns true if the map doesn’t contains the specified element(e) , else false will be returned which means element already exists.

𝑳𝒆𝒕𝒔 𝒔𝒆𝒆 𝒉𝒐𝒘 𝒓𝒆𝒎𝒐𝒗𝒆(𝑶𝒃𝒋𝒆𝒄𝒕 𝒐) 𝒘𝒐𝒓𝒌𝒔:

Removes the specified element from this set if it is present. This method returns true if this set changed as a result of the call.

Once this method returns true the element will not be present in Set.

*Remember to override equals() and hashCode() for any object you are going to store in HashSet, since your object is used as a key in backup Map, it must override those methods. Make your object Immutable or effective immutable if possible.*

package collections;import java.util.HashSet;  
import java.util.Set;  
import java.util.stream.Collectors;  
public class SetDemo {public static void main(String[] args) {  
 Set<Integer> s= new HashSet<>();   
 s.add(1);  
 s.add(8);  
 s.add(1);  
 s.add(11);  
 s.add(10);  
 s.add(101);  
 s.add(10);  
 s.add(102);  
 System.out.println(s.toString());   
 // Filter the data value >10   
 System.out.println( s.parallelStream()  
 .filter(value->value >10 )  
 .collect(Collectors.toSet()));  
 }}

content references : shivaprasadguram(linked in)