Java Collection Framework HashSet

Java HashSet class is used to store unique elements. It uses hash table internally to store the elements. It implements Set interface and extends the AbstractSet class. Declaration of the is given below.

Java HashSet class declaration

```
public class HashSet<E>extends AbstractSet<E>implements Set<E>,
Cloneable, Serializable
```

Copy

Important Points:

1. It creates a collection that uses hash table for storage. A hash table stores information by using a mechanism called hashing.
2. HashSet does not maintain any order of elements.
3. HashSet contains only unique elements.
4. It allows to store null value.
5. It is non synchronized.
6. It is the best approach for search operations.
7. The initial default capacity of HashSet is 16.

HashSet Constructors

HashSet class has three constructors that can be used to create HashSet accordingly.

```
HashSet()  //This creates an empty HashSet



HashSet( Collection C )  //This creates a HashSet that is initialized
with the elements of the Collection C



HashSet( int capacity )  //This creates a HashSet that has the
specified initial capacity
```

Copy

Example of HashSet class

In this example, we are creating a HashSet which is initially empty. Later on we will add items to this collection.

```
import java.util.*;

class Demo

{
  public static void main(String args[])

  {

    // Creating HashSet

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

    // Displaying HashSet

    System.out.println(hs);

  }

}
```

Copy

```
[]
```

HashSet Method

Method	Description
add(E e)	It adds the specified element to this set if it is not already present.
clear()	It removes all of the elements from the set.
clone()	It returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
contains(Object o)	It returns true if this set contains the specified element.
isEmpty()	It returns true if this set contains no elements.
iterator()	It returns an iterator over the elements in this set.
remove(Object o)	It removes the specified element from this set if it is present.
size()	It returns the number of elements in the set.

spliterator()	It creates a late-binding and fail-fast Spliterator over the elements in the set.

Add Elements to HashSet

In this example, we are creating a HashSet that store string values. Since HashSet does not store duplicate elements, we tried to add a duplicate elements but the output contains only unique elements.

```
import java.util.*;

class Demo

{

  public static void main(String args[])

  {

    // Creating HashSet

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

    // Adding elements

    hs.add("Mohan");

    hs.add("Rohan");

    hs.add("Sohan");

    hs.add("Mohan");

    // Displaying HashSet

    System.out.println(hs);

  }

}
```

Copy

```
[Mohan, Sohan, Rohan]
```

Remove Elements from HashSet

To remove elements from the hashset, we are using remove() method that remove the specified elements.

```
import java.util.*;

class Demo

{

  public static void main(String args[])

  {

    // Creating HashSet

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

    // Adding elements

    hs.add("Mohan");

    hs.add("Rohan");

    hs.add("Sohan");

    hs.add("Mohan");

    // Displaying HashSet

    System.out.println(hs);

    // Removing elements

    hs.remove("Mohan");

    System.out.println("After removing elements: \n"+hs);

  }

}
```

Copy

```
[Mohan, Sohan, Rohan]

After removing elements:

[Sohan, Rohan]
```

Traversing Elements of HashSet

Since HashSet is a collection then we can use loop to iterate its elements. In this example we are traversing elements using for loop. See the below example.

```
import java.util.*;

class Demo

{
  public static void main(String args[])

  {

    // Creating HashSet

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

    // Adding elements

    hs.add("Mohan");

    hs.add("Rohan");

    hs.add("Sohan");

    hs.add("Mohan");

    // Traversing ArrayList

    for(String element : hs) {

      System.out.println(element);

    }

  }

}
```

Copy

```
Mohan

Sohan

Rohan
```

Get size of HashSet

Sometimes we want to know number of elements an HashSet holds. In that case we use size() then returns size of HashSet which is equal to number of elements present in the list.

```
import java.util.*;

class Demo

{

  public static void main(String args[])

  {

    // Creating HashSet

    HashSet<String> hs = new HashSet<String>();
    // Adding elements

    hs.add("Mohan");

    hs.add("Rohan");

    hs.add("Sohan");

    hs.add("Mohan");

    // Traversing ArrayList

    for(String element : hs) {

      System.out.println(element);

    }

    System.out.println("Total elements : "+hs.size());

  }

}
```

Copy

```
OUTPUT-

Mohan

Sohan

Rohan

Total elements : 3
```