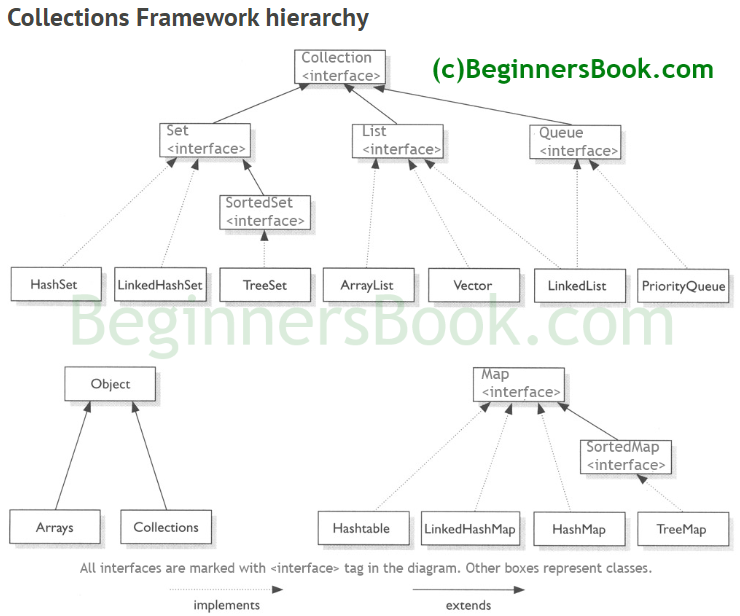
<https://www.java67.com/2013/01/difference-between-set-list-and-map-in-java.html>

<https://beginnersbook.com/2015/01/difference-between-list-set-and-map-in-java/>

The Java Collections Framework is a collection of interfaces and classes which helps in storing and processing the data efficiently.



### When to use LinkedList and when to use ArrayList?

1) As explained above the insert and remove operations give good performance (O(1)) in LinkedList compared to ArrayList(O(n)). Hence if there is a requirement of frequent addition and deletion in application then LinkedList is a best choice.

2) Search (get method) operations are fast in Arraylist (O(1)) but not in LinkedList (O(n)) so If there are less add and remove operations and more search operations requirement, ArrayList would be your best bet.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |

## 2. Performance  :  Performance of ArrayList and LinkedList depends on the type of operation a. get(int index) or search operation :  ArrayList get(int index) operation runs in constant time i.e O(1)  while LinkedList get(int index) operation run time is O(n) . The reason behind ArrayList being faster than LinkedList is that ArrayList uses index based system for its elements as it internally uses array data structure , on the other hand , LinkedList does not provide index based access for its elements as it iterates either from the beginning or end (whichever is closer) to retrieve the node at the specified element index.

## LinkedList vs ArrayList in Java

[Difference between LinkedList and ArrayList in Java](http://javarevisited.blogspot.com/2011/10/google-dart-program-example-tutorial.html)All the differences between LinkedList and ArrayList has there root on difference between [Array](http://javarevisited.blogspot.com/2012/01/anonymous-array-example-java-create.html) and LinkedList data-structure. If you are familiar with Array and LinkedList data structure you will most likely derive following differences between them:

1) Since Array is an index based data-structure searching or getting element from Array with index is pretty fast. Array provides O(1) performance for get(index) method but remove is costly in ArrayList as you need to rearrange all elements. On the Other hand LinkedList doesn't provide Random or index-based access and you need to iterate over linked list to retrieve any element which is of order O(n).

2) Insertions  are easy and fast in LinkedList as compared to ArrayList because there is no risk of resizing array

and copying content to new array if array gets full which makes adding into ArrayList of O(n) in worst case, while adding is O(1) operation in LinkedList in Java. ArrayList also needs to update its index if you insert something anywhere except at the end of array.

3) Removal is like insertions better in LinkedList than ArrayList.

4) LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next  and previous node.

Read more: <https://javarevisited.blogspot.com/2012/02/difference-between-linkedlist-vs.html#ixzz6OY3WJInK>

# Difference between list set and map in java?

BY CHAITANYA SINGH | FILED UNDER: [JAVA COLLECTIONS](https://beginnersbook.com/category/java-collections/)

[List, Set and Map](https://beginnersbook.com/java-collections-tutorials/) are the interfaces which implements Collection interface. Here we will discuss difference between List Set and Map in Java.

## List Vs Set Vs Map

1) **Duplicity:** List allows duplicate elements. Any number of duplicate elements can be inserted into the list without affecting the same existing values and their indexes.  
Set doesn’t allow duplicates. Set and all of the classes which implements Set interface should have unique elements.  
Map stored the elements as key & value pair. Map doesn’t allow duplicate keys while it allows duplicate values.

2)**Null values:** List allows any number of null values.  
Set allows single null value at most.  
Map can have single null key at most and any number of null values.

3) **Order:** List and all of its implementation classes maintains the insertion order.  
Set doesn’t maintain any order; still few of its classes sort the elements in an order such as TreeSet sorts the elements in ascending order, LinkedHashSet maintains the elements in insertion order.  
Similar to Set, Map also doesn’t stores the elements in an order, however few of its classes does the same. For e.g. TreeMap sorts the map in the ascending order of keys and LinkedHashMap sorts the elements in the insertion order, the order in which the elements got added to the LinkedHashMap.

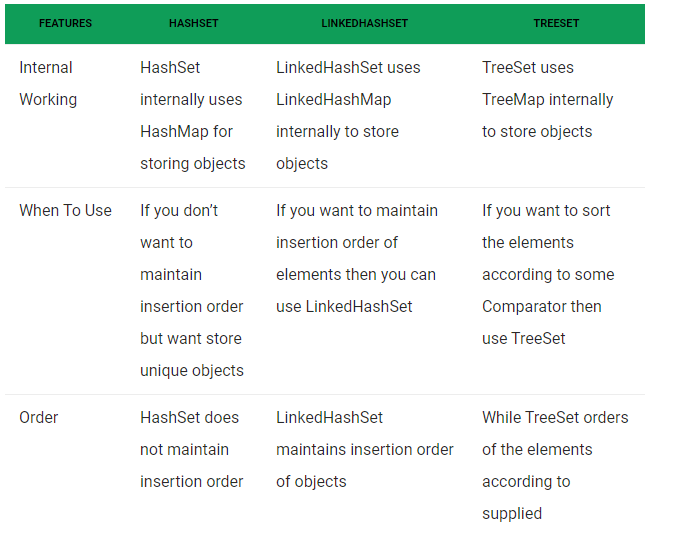
4) **Commonly used classes:**  
List: [ArrayList](https://beginnersbook.com/2013/12/java-arraylist/), [LinkedList](https://beginnersbook.com/2014/08/java-linkedlist-class/) etc.

Set: [HashSet](https://beginnersbook.com/2013/12/hashset-class-in-java-with-example/), [LinkedHashSet](https://beginnersbook.com/2013/12/linkedhashset-class-in-java-with-example/), [TreeSet](https://beginnersbook.com/2013/12/treeset-class-in-java-with-example/), SortedSet etc.

1. HashSet doesn’t maintain any kind of order of its elements.
2. TreeSet sorts the elements in ascending order.
3. LinkedHashSet maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.

Map: [HashMap](https://beginnersbook.com/2014/08/java-hashmap-class/), [TreeMap](https://beginnersbook.com/2013/12/treemap-in-java-with-example/), WeakHashMap, [LinkedHashMap](https://beginnersbook.com/2013/12/linkedhashmap-in-java/), IdentityHashMap etc.

## When to use List, Set and Map in Java?

1. If you do not want to have duplicate values in the database then Set should be your first choice as all of its classes do not allow duplicates.  
   2) If there is a need of frequent search operations based on the index values then List (ArrayList) is a better choice.  
   3) If there is a need of maintaining the insertion order then also the List is a preferred collection interface.  
   4) If the requirement is to have the key & value mappings in the database then Map is your best bet.
2. 

# HashSet Class in Java with example

BY CHAITANYA SINGH | FILED UNDER: [JAVA COLLECTIONS](https://beginnersbook.com/category/java-collections/)

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class is not synchronized. However it can be synchronized explicitly like this: Set s = Collections.synchronizedSet(new HashSet(...));

**Points to Note about HashSet:**

1. HashSet doesn’t maintain any order, the elements would be returned in any random order.
2. HashSet doesn’t allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.
3. HashSet allows null values however if you insert more than one nulls it would still return only one null value.
4. HashSet is non-synchronized.
5. The iterator returned by this class is fail-fast which means iterator would throw ConcurrentModificationException if HashSet has been modified after creation of iterator, by any means except iterator’s own remove method.

## HashSet Example

import java.util.HashSet;

public class HashSetExample {

public static void main(String args[]) {

// HashSet declaration

HashSet<String> hset =

new HashSet<String>();

// Adding elements to the HashSet

hset.add("Apple");

hset.add("Mango");

hset.add("Grapes");

hset.add("Orange");

hset.add("Fig");

//Addition of duplicate elements

hset.add("Apple");

hset.add("Mango");

//Addition of null values

hset.add(null);

hset.add(null);

//Displaying HashSet elements

System.out.println(hset);

}

}

Output:

[null, Mango, Grapes, Apple, Orange, Fig]

As you can see there all the duplicate values are not present in the output including the duplicate null value. No insertion order is maintained

## Example of LinkedHashSet:

import java.util.LinkedHashSet;

public class LinkedHashSetExample {

public static void main(String args[]) {

// LinkedHashSet of String Type

LinkedHashSet<String> lhset = new LinkedHashSet<String>();

// Adding elements to the LinkedHashSet

lhset.add("Z");

lhset.add("PQ");

lhset.add("PQ");

lhset.add("N");

lhset.add("O");

lhset.add("KK");

lhset.add("FGH");

System.out.println(lhset);

// LinkedHashSet of Integer Type

LinkedHashSet<Integer> lhset2 = new LinkedHashSet<Integer>();

// Adding elements

lhset2.add(99);

lhset2.add(7);

lhset2.add(7);

lhset2.add(0);

lhset2.add(67);

lhset2.add(89);

lhset2.add(66);

System.out.println(lhset2);

}

}

**Output:**

[Z, PQ, N, O, KK, FGH]

[99, 7, 0, 67, 89, 66]

Observe the output: Both types of LinkedHashSet have preserved the insertion order and also no duplicates are allowed.

# TreeSet in Java

TreeSet is one of the most important implementations of the SortedSet interface in Java that uses a Tree for storage. The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided. This must be consistent with equals if it is to correctly implement the Set interface. It can also be ordered by a Comparator provided at set creation time, depending on which constructor is used. The TreeSet implements a NavigableSet interface by inheriting AbstractSet class.  
Few important features of TreeSet are as follows:

1. TreeSet implements the [SortedSet](https://www.geeksforgeeks.org/sortedset-java-examples/) interface so duplicate values are not allowed.
2. Objects in a TreeSet are stored in a sorted and ascending order.
3. TreeSet does not preserve the insertion order of elements but elements are sorted by keys.
4. TreeSet does not allow to insert Heterogeneous objects. It will throw classCastException at Runtime if trying to add hetrogeneous objects.
5. TreeSet serves as an excellent choice for storing large amounts of sorted information which are supposed to be accessed quickly because of its faster access and retrieval time.
6. TreeSet is basically implementation of a self-balancing binary search tree like [Red-Black Tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/). Therefore operations like add, remove and search take O(Log n) time. And operations like printing n elements in sorted order takes O(n) time.

**Constructors of TreeSet class:**

1. **TreeSet t = new TreeSet();**  
   This will create empty TreeSet object in which elements will get stored in default natural sorting order.
2. **TreeSet t = new TreeSet(Comparator comp);**  
   This constructor is used when external specification of sorting order of elements is needed.
3. **TreeSet t = new TreeSet(Collection col);**  
   This constructor is used when any conversion is needed from any Collection object to TreeSet object.
4. **TreeSet t = new TreeSet(SortedSet s)**;  
   This constructor is used to convert SortedSet object to TreeSet Object.

**Synchronized TreeSet:**  
The implementation in a TreeSet is not synchronized in a sense that if multiple threads access a tree set concurrently, and at least one of the threads modifies the set, it must be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the set. If no such object exists, the set should be “wrapped” using the Collections.synchronizedSortedSet method. This is best done at creation time, to prevent accidental unsynchronized access to the set:

TreeSet ts = new TreeSet();

Set syncSet = Collections.synchronziedSet(ts);

Below program illustrates the basic opearation of a TreeSet:

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate insertions in TreeSet  import java.util.\*;    class TreeSetDemo {      public static void main(String[] args)      {          TreeSet<String> ts1 = new TreeSet<String>();            // Elements are added using add() method          ts1.add("A");          ts1.add("B");          ts1.add("C");            // Duplicates will not get insert          ts1.add("C");            // Elements get stored in default natural          // Sorting Order(Ascending)          System.out.println(ts1);      }  } |

**Output:**

[A, B, C]

Two things must be kept in mind while creating and adding elements into a TreeSet:

* Firstly, insertion of null into a TreeSet throws [NullPointerException](https://www.geeksforgeeks.org/null-pointer-exception-in-java/) because while insertion of null, it gets compared to the existing elements and null cannot be compared to any value.
* Secondly, if we are depending on default natural sorting order, compulsory the object should be **homogeneous**and **comparable** otherwise we will get **RuntimeException:**ClassCastException

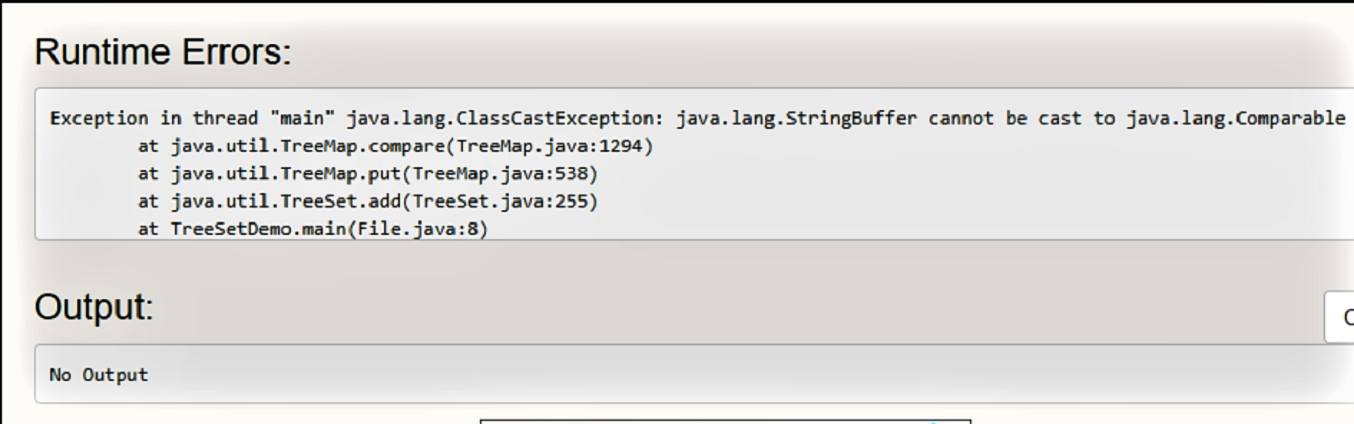
filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java code to illustrate StringBuffer  // class does not implements  // Comparable interface.  import java.util.\*;  class TreeSetDemo {        public static void main(String[] args)      {          TreeSet<StringBuffer> ts = new TreeSet<StringBuffer>();            // Elements are added using add() method          ts.add(new StringBuffer("A"));          ts.add(new StringBuffer("Z"));          ts.add(new StringBuffer("L"));          ts.add(new StringBuffer("B"));          ts.add(new StringBuffer("O"));            // We will get RunTimeException :ClassCastException          // As StringBuffer does not implements Comparable interface          System.out.println(ts);      }  } |

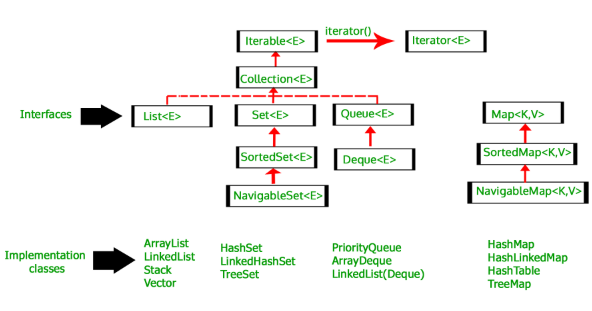


.  
**NOTE:**

1. An object is said to be comparable if and only if the corresponding class implements **Comparable interface**.
2. **String** class and all **Wrapper** classes already implements Comparable interface but StringBuffer class doesn’t implements Comparable interface.Hence we got ClassCastException in the above example.
3. For an empty tree-set, when trying to insert null as first value, one will get NPE from JDK 7.From 1.7 onwards null is not at all accepted by TreeSet. However upto JDK 6, null will be accepted as first value, but any if insertion of any more values in the TreeSet, will also throw NullPointerException.  
   Hence it was considered as bug and thus removed in JDK 7.

**Methods of TreeSet class:**  
TreeSet implements [SortedSet](https://www.geeksforgeeks.org/sortedset-java-examples/) so it has availability of all methods in Collection, [Set](http://quiz.geeksforgeeks.org/set-in-java/) and SortedSet interfaces. Following are the methods in Treeset interface.

* 1. [void add(Object o):](https://www.geeksforgeeks.org/treeset-add-method-in-java/) This method will add specified element according to some sorting order in TreeSet. Duplicate entires will not get added.
  2. [boolean addAll(Collection c):](https://www.geeksforgeeks.org/treeset-addall-method-in-java/) This method will add all elements of specified Collection to the set. Elements in Collection should be homogeneous otherwise ClassCastException will be thrown. Duplicate Entries of Collection will not be added to TreeSet.
  3. [void clear():](https://www.geeksforgeeks.org/treeset-clear-method-in-java/) This method will remove all the elements.
  4. [boolean contains(Object o):](https://www.geeksforgeeks.org/treeset-contains-method-in-java/) This method will return true if given element is present in TreeSet else it will return false.
  5. [Object first():](https://www.geeksforgeeks.org/treeset-first-method-in-java/) This method will return first element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
  6. [Object last():](https://www.geeksforgeeks.org/treeset-last-method-in-java/) This method will return last element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
  7. [SortedSet headSet(Object toElement):](https://www.geeksforgeeks.org/treeset-headset-method-in-java/) This method will return elements of TreeSet which are less than the specified element.
  8. [SortedSet tailSet(Object fromElement):](https://www.geeksforgeeks.org/treeset-tailset-method-in-java/) This method will return elements of TreeSet which are greater than or equal to the specified element.
  9. [SortedSet subSet(Object fromElement, Object toElement):](https://www.geeksforgeeks.org/treeset-subset-method-in-java/) This method will return elements ranging from fromElement to toElement. fromElement is inclusive and toElement is exclusive.
  10. [boolean isEmpty():](https://www.geeksforgeeks.org/treeset-isempty-method-in-java/) This method is used to return true if this set contains no elements or is empty and false for the opposite case.
  11. [Object clone():](https://www.geeksforgeeks.org/treeset-clone-method-in-java/) The method is used to return a shallow copy of the set, which is just a simple copied set.
  12. [int size():](https://www.geeksforgeeks.org/treeset-size-method-in-java/) This method is used to return the size of the set or the number of elements present in the set.
  13. [boolean remove(Object o):](https://www.geeksforgeeks.org/treeset-remove-method-in-java/) This method is used to return a specific element from the set.
  14. [Iterator iterator():](https://www.geeksforgeeks.org/treeset-iterator-method-in-java/) Returns an iterator for iterating over the elements of the set.
  15. [Comparator comparator()](https://www.geeksforgeeks.org/treeset-comparator-method-in-java/): This method will return Comparator used to sort elements in TreeSet or it will return null if default natural sorting order is used.
  16. [**ceiling​(E e):**](https://www.geeksforgeeks.org/treeset-ceiling-method-in-java-with-examples/) This method returns the least element in this set greater than or equal to the given element, or null if there is no such element.
  17. **descendingIterator​():** This method returns an iterator over the elements in this set in descending order.
  18. **descendingSet​():** This method returns a reverse order view of the elements contained in this set.
  19. **floor​(E e):** This method returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
  20. **higher​(E e):** This method returns the least element in this set strictly greater than the given element, or null if there is no such element.
  21. **lower​(E e):** This method returns the greatest element in this set strictly less than the given element, or null if there is no such element.
  22. **pollFirst​():** This method retrieves and removes the first (lowest) element, or returns null if this set is empty.
  23. **pollLast​():** This method retrieves and removes the last (highest) element, or returns null if this set is empty.
  24. **spliterator​():** This method creates a late-binding and fail-fast Spliterator over the elements in this set.



|  |
| --- |
| public class GFG {      public static void main(String[] args)      {            HashMap<String, Integer> map              = new HashMap<>();            print(map);          map.put("vishal", 10);          map.put("sachin", 30);          map.put("vaibhav", 20);            System.out.println("Size of map is:- "                             + map.size());            print(map);          if (map.containsKey("vishal")) {              Integer a = map.get("vishal");              System.out.println("value for key"                                 + " \"vishal\" is:- "                                 + a);          }            map.clear();          print(map);      }        public static void print(Map<String, Integer> map)      {          if (map.isEmpty()) {              System.out.println("map is empty");          }            else {              System.out.println(map);          }      }  } |

**Output:**

map is empty

Size of map is:- 3

{vaibhav=20, vishal=10, sachin=30}

value for key "vishal" is:- 10

map is empty

-------------------------33333333333333333============3=33333333333333333333333==============

**Difference between Set, List and Map in Java - Interview question**

Set, List and Map are three important interfaces of Java collection framework and Difference between Set, List, and Map in Java is one of the most frequently asked [Java Collection interview question](http://java67.blogspot.com/2012/09/java-collection-interview-questions.html). Sometimes this question is asked as When to use List, Set and Map in Java. Clearly, the interviewer is looking to know that whether you are familiar with fundamentals of Java collection framework or not. In order to decide when to use List, Set or Map, you need to know what are these interfaces and what functionality they provide. [List in Java](http://java67.blogspot.com/2012/07/sort-list-ascending-descending-order-set-arraylist.html) provides ordered and indexed collection which may contain duplicates.   
  
The Set interface provides an unordered collection of unique objects, i.e. Set doesn't allow duplicates, while Map provides a data structure based on key-value pair and hashing.   
  
All three List, Set, and Map are interfaces in Java and there are many concrete implementations of them are available in Collection API. ArrayList and LinkedList are two most popular used List implementation while [LinkedHashSet, TreeSet, and HashSet](http://javarevisited.blogspot.com/2012/11/difference-between-treeset-hashset-vs-linkedhashset-java.html) are frequently used Set implementation.

# [Difference between TreeSet, LinkedHashSet and HashSet in Java with Example](https://javarevisited.blogspot.com/2012/11/difference-between-treeset-hashset-vs-linkedhashset-java.html)

TreeSet, LinkedHashSet and HashSet all are implementation of Set interface and by virtue of that, they follows contract of Set interface i.e. they do not allow duplicate elements. Despite being from same type hierarchy,  there are lot of difference between them; which is important to understand, so that you can choose most appropriate [Set](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html) implementation based upon your requirement. By the way difference between TreeSet and HashSet or LinkedHashSet is also one of the [popular Java Collection interview question](http://javarevisited.blogspot.sg/2011/11/collection-interview-questions-answers.html), not as popular as [Hashtable vs HashMap](http://javarevisited.blogspot.sg/2010/10/difference-between-hashmap-and.html) or [ArrayList vs Vector](http://javarevisited.blogspot.sg/2011/09/difference-vector-vs-arraylist-in-java.html) but still appears in various Java interviews.  
  
  
In this article we will see *difference between HashSet, TreeSet and LinkedHashSet* on various points e.g. Ordering of elements, performance, allowing null etc and then we will see When to use TreeSet or LinkedHashSet or simply [HashSet in Java](http://javarevisited.blogspot.sg/2012/06/hashset-in-java-10-examples-programs.html).

## Difference between TreeSet, LinkedHashSet and HashSet in Java

[TreeSet vs HashSet vs LinkedHashSet in Java with Example](https://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)TreeSet, LinkedHashSet and HashSet in Java are three Set implementation in collection framework and like many others they are also used to store objects. Main feature of TreeSet is sorting,  LinkedHashSet is insertion order and HashSet is just general purpose collection for storing object.   
  
HashSet is implemented using [HashMap in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) while TreeSet is implemented using [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html).  TreeSet is a SortedSet implementation which allows it to keep elements in the sorted order defined by either [Comparable or Comparator interface](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html).   
  
Comparable is used for natural order sorting and Comparator for [custom order sorting](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html) of objects, which can be provided while creating instance of TreeSet.   
  
Anyway before seeing difference between TreeSet, LinkedHashSet and HashSet, let's see some similarities between them:

1) **Duplicates** : All three implements Set interface means they are not allowed to store duplicates.

2) **Thread safety**: HashSet, TreeSet and LinkedHashSet are not [thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), if you use them in multi-threading environment where at least one Thread  modifies Set you need to externally synchronize them.

3) **Fail-Fast Iterator** : Iterator returned by TreeSet, LinkedHashSet and HashSet are fail-fast Iterator. i.e. If Iterator is modified after its creation by any way other than Iterators remove() method, it will throw ConcurrentModificationException with best of effort. read more about [fail-fast vs fail-safe Iterator](http://javarevisited.blogspot.sg/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) here

Now let’s see the **difference between HashSet, LinkedHashSet and TreeSet in Java** :

### 1. Performance and Speed

First difference between them comes in terms of  speed.  HashSet is fastest, LinkedHashSet is second on performance or almost similar to HashSet but TreeSet is bit slower because of sorting operation it needs to perform on each insertion. TreeSet provides guaranteed O(log(n)) time for common operations like add, remove and contains, while HashSet and LinkedHashSet offer constant time performance e.g. O(1) for add, contains and remove given hash function uniformly distribute elements in bucket.

### 2. Ordering

HashSet does not maintain any order while LinkedHashSet maintains insertion order of elements much like List interface and TreeSet maintains sorting order or elements.

### 3. Internal Implementation

HashSet is backed by an HashMap instance, LinkedHashSet is implemented using HashSet and LinkedList while TreeSet is backed up by NavigableMap in Java and by default it uses TreeMap.

### 4. null

Both HashSet and LinkedHashSet allows null but TreeSet doesn't allow null but TreeSet doesn't allow null and throw [java.lang.NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html) when you will insert null into TreeSet. Since TreeSet uses [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) of respective elements to compare them  which throws NullPointerException while comparing with null, here is an example:

TreeSet cities

Exception in thread "main" java.lang.NullPointerException

        at java.lang.String.compareTo(String.java:1167)

        at java.lang.String.compareTo(String.java:92)

        at java.util.TreeMap.put(TreeMap.java:545)

        at java.util.TreeSet.add(TreeSet.java:238)

**5. Comparison**  
HashSet and LinkedHashSet uses [equals() method in Java](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) for comparison but TreeSet uses [compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) for maintaining ordering. That's why compareTo() should be consistent to equals in Java. failing to do so break general contact of Set interface i.e. it can permit duplicates.

## TreeSet vs HashSet vs LinkedHashSet - Example

Let’s compare all these Set implementation on some points by writing Java program. In this example we are demonstrating difference in ordering, time taking while inserting 1M records among [TreeSet](http://java67.blogspot.sg/2012/08/difference-between-treemap-and-treeset-java.html), HashSet and LinkedHashSet in Java. This will help to solidify some points which discussed in earlier section and help to decide when to use HashSet, LinkedHashSet or TreeSet in Java.

**import** java.util.Arrays;  
**import** java.util.HashSet;  
**import** java.util.LinkedHashSet;  
**import** java.util.Set;  
**import** java.util.TreeSet;  
  
/\*\*  
 \* Java program to demonstrate difference between TreeSet, HashSet and LinkedHashSet

 \* in Java Collection.  
 \* @author  
 \*/  
**public** **class** SetComparision {  
   
    **public** **static** **void** main(**String** args[]){              
        **HashSet**<**String**> fruitsStore = **new** **HashSet**<**String**>();  
        **LinkedHashSet**<**String**> fruitMarket = **new** **LinkedHashSet**<**String**>();  
        **TreeSet**<**String**> fruitBuzz = **new** **TreeSet**<**String**>();  
       
        for(**String** fruit: **Arrays**.asList("mango", "apple", "banana")){  
            fruitsStore.add(fruit);  
            fruitMarket.add(fruit);  
            fruitBuzz.add(fruit);  
        }

*//no ordering in HashSet – elements stored in random order*  
        **System**.out.println("Ordering in HashSet :" + fruitsStore);

*//insertion order or elements – LinkedHashSet storeds elements as insertion*  
        **System**.err.println("Order of element in LinkedHashSet :" + fruitMarket);

*//should be sorted order – TreeSet stores element in sorted order*  
        **System**.out.println("Order of objects in TreeSet :" + fruitBuzz);

*//Performance test to insert 10M elements in HashSet, LinkedHashSet and TreeSet*  
        **Set**<**Integer**> numbers = **new** **HashSet**<**Integer**>();  
        **long** startTime = **System**.nanoTime();  
        for(**int** i =0; i<10000000; i++){  
            numbers.add(i);  
        }

        **long** endTime = **System**.nanoTime();  
        **System**.out.println("Total time to insert 10M elements in HashSet in sec : "

                            + (endTime - startTime));  
       
       
        *// LinkedHashSet performance Test – inserting 10M objects*

        numbers = **new** **LinkedHashSet**<**Integer**>();  
        startTime = **System**.nanoTime();  
        for(**int** i =0; i<10000000; i++){  
            numbers.add(i);  
        }  
        endTime = **System**.nanoTime();  
        **System**.out.println("Total time to insert 10M elements in LinkedHashSet in sec : "

                            + (endTime - startTime));

*// TreeSet performance Test – inserting 10M objects*  
        numbers = **new** **TreeSet**<**Integer**>();  
        startTime = **System**.nanoTime();  
        for(**int** i =0; i<10000000; i++){  
            numbers.add(i);  
        }  
        endTime = **System**.nanoTime();  
        **System**.out.println("Total time to insert 10M elements in TreeSet in sec : "

                            + (endTime - startTime));  
    }  
}  
  
**Output**  
Ordering in **HashSet** :[banana, apple, mango]  
Order of element in **LinkedHashSet** :[mango, apple, banana]  
Order of objects in **TreeSet** :[apple, banana, mango]  
Total time to insert 10M elements in **HashSet** in sec : **3564570637**  
Total time to insert 10M elements in **LinkedHashSet** in sec : **3511277551**  
Total time to insert 10M elements in **TreeSet** in sec : **10968043705**

## When to use HashSet, TreeSet and LinkedHashSet in Java

Since all three implements Set [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) they can be used for common Set operations like not allowing duplicates but since HashSet, TreeSet and LinkedHashSet has there special feature which makes them appropriate in certain scenario.   
  
Because of sorting order provided by TreeSet, use TreeSet when you need a collection where elements are sorted without duplicates. [HashSet](http://javarevisited.blogspot.sg/2011/09/difference-hashmap-vs-hashset-java.html) are rather general purpose Set implementation, Use it as default Set implementation if you need a fast, duplicate free collection.  
  
While LinkedHashSet is extension of HashSet and its more suitable where you need to maintain **insertion order** of elements, similar to [List](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html) without compromising performance for costly TreeSet.   
  
Another use of LinkedHashSet is for creating copies of existing Set, Since LinkedHashSet preservers insertion order, it returns Set which contains same elements in same order like exact copy.   
  
In short,  although all three are Set interface implementation they offer distinctive feature, HashSet is a general purpose Set while LinkedHashSet provides insertion order guarantee and TreeSet is a SortedSet which stores elements in sorted order specified by [Comparator or Comparable in Java](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html).

## How to copy object from one Set to other

Here is code example of LinkedHashSet which demonstrate How LinkedHashSet can be used to copy objects from one Set to another without losing order. You will get exact replica of source Set, in terms of contents and order. Here static method copy(Set source) is written using [Generics](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html), This kind of [parameterized method](http://javarevisited.blogspot.sg/2012/08/how-to-write-parametrized-class-method-Generic-example.html) provides type-safety and help to avoid ClassCastException at runtime.

**import** java.util.Arrays;  
**import** java.util.HashSet;  
**import** java.util.LinkedHashSet;  
**import** java.util.Set;  
  
/\*\*  
 \* Java program to **copy object from one HashSet to another using LinkedHashSet**.  
 \* LinkedHashSet preserves order of element while copying elements.  
 \*  
 \* @author Javin  
 \*/  
**public** **class** SetUtils{  
     
    **public** **static** **void** main(**String** args[]) {  
         
        **HashSet**<**String**> source = **new** **HashSet**<**String**>(**Arrays**.asList("Set, List, Map"));  
        **System**.out.println("source : " + source);  
        **Set**<**String**> copy = SetUtils.copy(source);  
        **System**.out.println("copy of HashSet using LinkedHashSet: " + copy);  
    }  
     
    */\*  
     \* Static utility method to copy Set in Java  
     \*/*  
    **public** **static** <T> **Set**<T> copy(**Set**<T> source){  
           **return** **new** **LinkedHashSet**<T>(source);  
    }  
}  
**Output:**  
source : [**Set**, **List**, **Map**]  
copy of **HashSet** using **LinkedHashSet**: [**Set**, **List**, **Map**]

Always code for interface than implementation so that you can replace HashSet to LinkedHashSet or TreeSet when your requirement changes. That’s all on *difference between HashSet, LinkedHashSet and TreeSet in Java*.  If you know any other significant difference between TreeSet, LinkedHashSet and HashSet which is worth remembering than please add as comment.

Read more: <https://javarevisited.blogspot.com/2012/11/difference-between-treeset-hashset-vs-linkedhashset-java.html#ixzz6M1KayWzn>

In this Java article, we will see the *difference between Map, Set, and List in Java* and learn when to use List, Set or Map.

**Set vs List vs Map in Java**

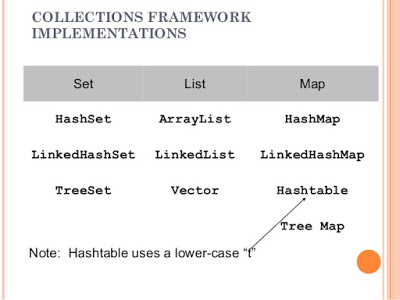
As I said Set, List and Map are interfaces, which defines core contract e.g. a Set contract says that it can not contain duplicates. Based on our knowledge of List, Set and Map let's compare them on different metrics.

**Duplicate Objects**

The main difference between List and Set interface in Java is that List **allows duplicates** while Set doesn't allow duplicates. All implementation of Set honor this contract.   
  
While a Map holds two objects per Entry e.g. a key and a value and It may contain duplicate values but keys are always unique. See [here](http://java67.blogspot.sg/2012/08/difference-between-list-and-set-in-java.html) for more difference between List and Set data structure in Java.

**Order**

Another key difference between List and Set is that List is an **ordered collection**, List's contract maintains insertion order or element. Set is an **unordered collection**, you get no guarantee on which order element will be stored.   
  
Though some of the Set implementation e.g. LinkedHashSet maintains order. Also SortedSet and SortedMap e.g. [TreeSet and TreeMap](http://java67.blogspot.com/2012/08/difference-between-treemap-and-treeset-java.html) maintain a sorting order, imposed by using Comparator or Comparable.

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https://www.pluralsight.com/courses/java-8-lambda-expressions-collections-streams)

Read more: <https://www.java67.com/2013/01/difference-between-set-list-and-map-in-java.html#ixzz6M1KruN5P>

**Difference between TreeMap and TreeSet in Java**

**Difference between TreeSet and TreeMap in Java**  
Main Difference between TreeMap and TreeSet is that TreeMap is an implementation of Map interface while TreeSet is an implementation of Set interface. There are some similarities between both TreeMap and TreeSet and few differences as well. In this Java tutorial, we will first see similarities between TreeMap and TreeSet and than you will learn some differences between TreeMap and TreeSet in Java. Key point to remember about TreeMap and TreeSet is that they use compareTo() or compare() method to compare object, So if uses puts a String object in TreeSet of Integers, add() method will throw ClassCastException at runtime prior to Java 5, with Java 5 you can use Generics to avoid this happening by declaring TreeMap and TreeSet with parametrized version. If you want to master Java Collection framework by heart, you can see [Java Generics and Collection](http://www.amazon.com/dp/0596527756/?tag=javamysqlanta-20) book by Maurice Naftaline, one of the best work on Java Collections framework.

**Similarities between TreeMap and TreeSet in Java**

Here is a list of similarities between TreeMap and TreeSet in Java:

[Difference between TreeMap and TreeSet in Java](https://1.bp.blogspot.com/-_GCqP1vu06Q/UBaTOZM869I/AAAAAAAAAaw/ykubu9U9kK4/s1600/java_logo_50_50.jpg)1) Both TreeMap and TreeSet are sorted data structure, which means they keep there element in predefined Sorted order. Sorting order can be natural sorting order defined by [Comparable](http://java67.blogspot.sg/2013/08/difference-between-comparator-and-comparable-in-java-interface-sorting.html) interface or custom sorting Order defined by [Comparator](http://java67.blogspot.sg/2014/11/java-8-comparator-example-using-lambda-expression.html) interface. Both TreeMap and TreeSet has overloaded constructor which accept a Comparator, if provided all elements inside TreeSet or TreeMap will be compared and Sorted using this Comparator.

2) Both TreeSet and TreeMap implements base interfaces e.g. TreeSet implements Collection and Set interface so that they can be passed to method where a Collection is expected and TreeMap implements java.util.Map interface, which means you can pass it when a Map is expected.  
  
  
3) TreeSet is practically implemented using TreeMap instance, similar to HashSet which is internally backed by HashMap instance. See my post [Internal Implementation of HashSet](http://java67.blogspot.sg/2014/01/how-hashset-is-implemented-or-works-internally-java.html) to learn more.

4) Both TreeMap and TreeSet are non synchronized Collection, hence can not be shared between multiple threads. You can make both TreeSet and TreeMap synchronized by wrapping them into Synchronized collection by calling Collections.synchroinzedMap() method.

5) Iterator returned by TreeMap and TreeSet are fail-fast, means they will throw ConcurrentModificationException when TreeMap or TreeSet is modified structurally once Iterator is created. this [fail-fast behavior](http://java67.blogspot.sg/2015/06/what-is-fail-safe-and-fail-fast-iterator-in-java.html) is not guaranteed but works in best effort.

6) Both TreeMap and TreeSet are slower than there Hash counter part like HashSet and HashMap and instead of providing constant time performance for add, remove and get operation they provide performance in O(log(n)) order.

**TreeSet vs TreeMap in Java**

Now let's see some differences between TreeSet vs TreeMap in Java:

1) Major difference between TreeSet and TreeMap is that TreeSet implements Set interface while TreeMap implements Map interface in Java.

2) Second difference between TreeMap and TreeSet is the way they store objects. TreeSet stores only one object while TreeMap uses two objects called key and Value. Objects in TreeSet are sorted while keys in TreeMap remain in sorted Order.

3) Third difference between TreeSet and TreeMap is that, former implements NavigableSet while later implements NavigableMap in Java.

4) Fourth difference is that duplicate objects are not allowed in TreeSet but duplicates values are allowed in TreeMap.

That's all on **difference between TreeSet and TreeMap in Java**. If you find any other significant difference between TreeMap and TreeSet then please post as comment.

Read more: <https://www.java67.com/2012/08/difference-between-treemap-and-treeset-java.html#ixzz6M1L0lWDu>

**Null elements**

The list allows null elements and you can have many null objects in a List because it also allowed duplicates. Set just allow one null element as there is no duplicate permitted while in Map you can have null values and at most one null key.   
  
Worth noting is that [Hashtable doesn't allow null key or values](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) but HashMap allows null values and one null key.  This is also the main difference between these two popular implementations of Map interface, aka HashMap vs Hashtable.

**Useful implementations**  
Most popular implementations of List interface in Java are ArrayList, LinkedList, and Vector class. ArrayList is more general purpose and provides random access with index, while LinkedList is more suitable for frequently adding and removing elements from List.  
  
Vector is synchronized counterpart of ArrayList. On the other hand, most popular implementations of the Set interface are HashSet, LinkedHashSet, and TreeSet. First one is general purpose Set which is backed by HashMap, see [how HashSet works internally in Java](http://java67.blogspot.sg/2014/01/how-hashset-is-implemented-or-works-internally-java.html) for more details.  
  
It also doesn't provide any ordering guarantee but LinkedHashSet does provide ordering along with uniqueness offered by the Set interface.  
  
Third implementation TreeSet is also an implementation of SortedSet interface, hence it keeps elements in a sorted order specified by compare() or compareTo() method.  
  
Now the last one, most popular implementation of Map interface is HashMap, LinkedHashMap, Hashtable, and TreeMap.  
  
First one is the non-synchronized general purpose Map implementation while Hashtable is its synchronized counterpart, both doesn' provide any ordering guarantee which comes from LinkedHashMap. Just like TreeSet, TreeMap is also a sorted data structure and keeps keys in sorted order (see [Java Fundamentals: Collections](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fjava-fundamentals-collections))

Read more: <https://www.java67.com/2013/01/difference-between-set-list-and-map-in-java.html#ixzz6M1LDEDsE>

Read more: <https://www.java67.com/2013/01/difference-between-set-list-and-map-in-java.html#ixzz6M1KEzhHm>

**How HashSet Internally Works in Java**

Not many programmer know that HashSet is internally implemented using HashMap in Java, so if you know [How HashMap works internally in Java](http://java67.blogspot.sg/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html), more likely you can figure out *how HashSet works in Java*. But, now a curious Java developer can question that, how come HashSet uses HashMap, because you need a key value pair to use with Map, while in HashSet we only store one object. Good question, isn't it? If you remember some functionality of earlier class, then you know that [HashMap allows duplicate values](http://java67.blogspot.sg/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html) and this property is exploited while implementing HashSet in Java.  
  
  
Since HashSet implements Set interface it needs to guarantee uniqueness and this is achieved by storing elements as keys with same value always. Things gets clear by checking HashSet.java from JDK source code.  
  
All you need to look at is, how elements are stored in HashSet and how they are retrieved from HashSet. Since HashSet doesn't provide any direct method for retrieving object e.g. get(Key key) from HashMap or get(int index) from List, only way to get object from HashSet is via Iterator. See [here](http://java67.blogspot.sg/2012/10/how-to-iterate-over-hashset-in-java.html)for code example of iterating over HashSet in Java.  
  
When you create an object of HashSet in Java, it internally create instance of backup Map with default initial capacity 16 and default load factor 0.75 as shown below :  
  
/\*\*

\* Constructs a new, empty set; the backing <tt>HashMap</tt> instance has

\* default initial capacity (16) and load factor (0.75).

\*/

public **HashSet**() {

map **=** **new** **HashMap**<>();

}

Now let's see the code for add() and iterate() method from java.util.HashSet in Java to understand *how HashSet works internally in Java*.

**How Object is stored in HashSet**

As you can see below, a call to add(Object) is delegate to put(Key, Value) internally, where Key is the object you have passed and value is another object,  called PRESENT, which is a constant in java.util.HashSet as shown below :

**private** transient **HashMap**<E,**Object**> map;

// Dummy value to associate with an Object in the backing Map

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

public boolean add(E e) {

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

}

Since PRESENT is a constant, for all keys we have same value in backup HashMap called map.  
  
  
**How Object is retrieved from HashSet**  
Now let's see the code for getting iterator for traversing over HashSet in Java. iterator() method from java.util.HashSet class returns iterator for backup Map returned by map.keySet().iterator() method.  
  
       /\*\*

\* Returns an iterator over the elements in this set. The elements

\* are returned in no particular order.

\*

\* @return an Iterator over the elements in this set

\* @see ConcurrentModificationException

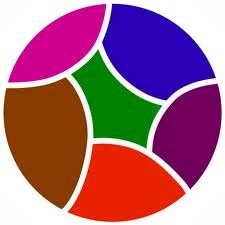
\*/

public **Iterator**<E> iterator() {

**return** map.keySet().iterator();

}

**How to use HashSet in Java**

[](http://www.shareasale.com/m-pr.cfm?merchantID=53701&userID=880419&productID=687369660)

Using HashSet in Java is very simple, don't think it is Map but think more like Collection i.e. add elements by using add() method, check its return value to see if object already existed in HashSet or not. Similarly use iterator for retrieving element from HashSet in Java. You can also use contains() method to check if any object already exists in HashSet or not. This method use [equals() method](http://java67.blogspot.sg/2012/11/difference-between-operator-and-equals-method-in.html) for comparing object for matching. You can also use remove() method to remove object from HashSet. Since element of HashSet is used as key in backup HashMap, they must implement [equals() and hashCode()](http://java67.blogspot.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html) method. Immutability is not requirement but if its immutable then you can assume that object will not be changed during its stay on set. Following example demonstrate basic usage of HashSet in Java, for more advanced example, you can check [this](http://javarevisited.blogspot.sg/2012/06/hashset-in-java-10-examples-programs.html)tutorial.

**import** **java.util.HashSet**;

**import** **java.util.Iterator**;

/\*\*

\* Java Program to demonstrate How HashSet works internally in Java.

\* @author http://java67.blogspot.com

\*/

**public** **class** **HashSetDemo**{

**public** **static** **void** **main**(String args[]) {

HashSet<String> supportedCurrencies = **new** HashSet<String>();

// adding object into HashSet, this will be translated to put() calls

supportedCurrencies.add("USD");

supportedCurrencies.add("EUR");

supportedCurrencies.add("JPY");

supportedCurrencies.add("GBP");

supportedCurrencies.add("INR");

supportedCurrencies.add("CAD");

// retrieving object from HashSet

Iterator<String> itr = supportedCurrencies.iterator();

**while**(itr.hasNext()){

System.out.println(itr.next());

}

}

}

Output

JPY

EUR

INR

USD

CAD

GBP

That's all about **How HashSet is implemented in Java** and **How HashSet works internally**. As I said, If you how HashMap internally in Java, you can explain working of HashSet provided,  you know it uses same values for all keys. Remember to override equals() and hashCode() for any object you are going to store in HashSet, since your object is used as key in backup Map, it must override those method. Make your object Immutable or effective immutable if possible.

Read more: <https://www.java67.com/2014/01/how-hashset-is-implemented-or-works-internally-java.html#ixzz6M1LXvHNp>

**When to use List, Set and Map in Java**

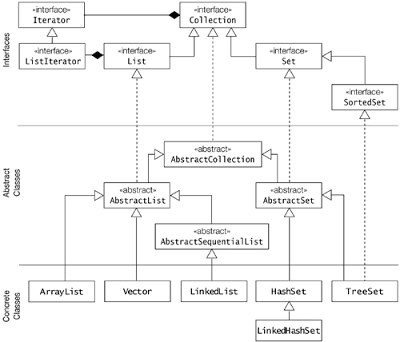
Based upon our understanding of the *difference between Set, List and Map* we can now decide when to use List, Set or Map in Java.

1) If you need to access elements frequently by using the index than List is a way to go. Its implementation e.g. [ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) provides faster access if you know index.

2) If you want to store elements and want them to maintain an **order** on which they are inserted into a collection then go for List again, as List is an ordered collection and maintain insertion order.

3) If you want to create a collection of unique elements and **don't want any duplicate** than choosing any Set implementation e.g. HashSet, LinkedHashSet or TreeSet.   
  
All Set implementation follow there general contract e.g. uniqueness but also add addition feature e.g. TreeSet is a SortedSet and elements stored on TreeSet can be sorted by using [Comparator or Comparable in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html). LinkedHashSet also maintains insertion order.

4) If you store data in form of key and value than Map is the way to go. You can choose from Hashtable, HashMap, TreeMap based upon your subsequent need. In order to choose between first two see the [difference between HashSet and HashMap in Java](http://javarevisited.blogspot.com/2011/09/difference-hashmap-vs-hashset-java.html).

[](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https://www.udemy.com/java-the-complete-java-developer-course/)

That's all on the **difference between Set, List, and Map in Java**. All three are the most fundamental interface of Java Collection framework and any Java developer should know there distinguish feature and given a situation should be able to pick right Collection class to use.  You can also further see, [Java Fundamentals: Collections](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fjava-fundamentals-collections), a free course form Pluralsight by Richard Warburton. Pluralsight offers a 10-day free trial period, in which you can access many more advanced Java courses.  
  
It's also good to remember the difference between there implementation e.g. When to use ArrayList and LinkedList, HashMap vs Hashtable or [When to use Vector or ArrayList](http://java67.blogspot.com/2012/09/arraylist-vs-vector-in-java-interview.html) etc. Collection API is huge and it's difficult to know every bits and piece but at the same time, there is no excuse for not knowing fundamentals like the difference between Set, List, and Map in Java.

Read more: <https://www.java67.com/2013/01/difference-between-set-list-and-map-in-java.html#ixzz6M1LzF2OV>

[**Difference between HashMap and HashSet in Java**](https://javarevisited.blogspot.com/2011/09/difference-hashmap-vs-hashset-java.html)

HashMap vs HashSet is the most frequently asked question during any [core java interview and](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html) the interview is not said completed until they will not cover the **Collection Framework** and[**multi-threading interview**](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html) and collections are uncompleted without Covering **Hash Set and HashMap.**

Both HashMap and HashSet are part of the collection framework which allows us to work with a collection of objects. Collection Framework has its own interfaces and implementation classes. Basically, a collection is divided as Set Interface, List, and Queue Interfaces.  
  
  
All these interfaces have their own property also apart from they get from a collection like Set allows Collection of objects but forbids duplicate value, List allows duplicate along with indexing. Queue works on the FCFS algorithm.

First, we have one look at **What HashMap and HashSet**are then will go for Differences between HashSet and HashMap

**What is HashSet in Java?**

[HashMap vs HashSet, difference between HashMap and Hashset in Java](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html)**HashSet**is the implementation of **Set** Interface which does not allow duplicate value all the methods which are in **Collection** **Framework** are also in Set Interface by default but when we are talking about Hash set the main thing is objects which are going to be stored in HashSet must [override **equals**()](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) and **hashCode**() method so that we can check for equality and no duplicate value is stored in our set.   
  
If we have created our own objects we need to implement **hashCode**() and **equal**() in such a manner that will be able to compare objects correctly when storing in a set so that duplicate objects are not stored, if we have not to override this method objects will take default implementation of this method.

**public boolean add(Object o)**method is used to add an element in a set which returns false if it’s a duplicate value in case of  HashSet otherwise returns true if added successfully.

**What is HashMap?**

HashMap is an implementation of Map Interface, which maps a key to value. Duplicate keys are not allowed in a map. Basically, map Interface has two implementation classes HashMap and TreeMap the main difference is TreeMap maintains the order of the objects but HashMap will not.HashMap allows null values and null keys.[HashMap is not synchronized](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html), but the collection framework provide methods so that we can make them synchronized if multiple threads are going to access our hashmap and one thread structurally changes our map.

**public Object put(Object key, Object value)**method is used to add an element in the map.

You can read more about HashMap in my article [How HashMap works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) and the [Difference between HashMap and hashtable in Java](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html)

**Difference between HashSet and HashMap in Java**

Following are some differences between HashMap and HashSet:

|  |  |
| --- | --- |
| **HashMap** | **Hash Set** |
| HashMap  is an implementation of Map interface | HashSet is an implementation of Set Interface |
| HashMap Stores data in form of  key-value pair | HashSet Store only objects |
| Put method is used to add element in map | Add method is used to add element is Set |
| In hash map hashcode value is calculated using key object | Here member object is used for calculating hashcode value which can be same for two objects so equal () method is used to check for equality if it returns false that means two objects are different. |
| HashMap is faster than HashSet because unique key is used to access object | HashSet is slower than Hashmap |

Please let me know if you need any other difference between HashSet and HashMap in Java and I will add them here.

Read more: <https://javarevisited.blogspot.com/2011/09/difference-hashmap-vs-hashset-java.html#ixzz6M1ME6aGu>

**Difference between List and Set in Java**

**List vs Set in Java**

Main Difference between List and Set is that List is an ordered Collection while Set is an unordered collection. Java collection framework offers several collection classes for various needs but all collection can be divided into broadly three categories : List, Set, and Map. All List, Set, and Map are defined as interfaces and then you have several implementation like ArrayList and Vector are the popular implementation of List interface, while HashSet is a popular implementation of Set interface. In this java tutorial, we will mainly see What are differences between List and Set collection at the top level and How to choose when to use List in java and when to use Set in Java. I believe once you are able to understand the fundamental difference between List and Set you are most likely be well versed on when to use List and when to choose Set.

**Difference between List and Set in Java**

As I said the main difference between Set and List is that List is an ordered Collection which means List preserves the order on which an element is inserted into List.   
  
  
  
So if you insert Object A before Object B then A will be stored at lower index than B. Since Set is an UN-ordered collection it doesn't maintain any inserting order of element,   
  
Though you can have SortedSet which offers to sort functionality on top of Set interface and you can impose either natural order or Object or any custom order by using Comparator and Comparable while storing object inside Set.

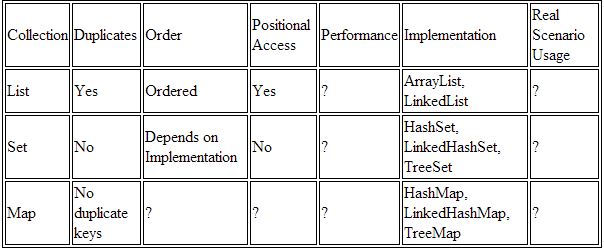
Another ***significant difference between List and Set*** is that List allows you to store duplicates in the collection while Set doesn't allow any duplicates. This is very significant as it clearly says that if you want a collection of unique object use Set. duplication of Object is detected using equals() method.   
  
So if two objects are equal using equals method, the later object will replace the former in Set if added using add() method, due to this reason only one null element is allowed inside Set.   
  
It's also worth noting that in the case of SortedSet like TreeSet, compareTo method is used to compare object and decide whether an object is duplicate or not. two objects will be duplicate if their compareTo() method returns zero and that's why it's said that compareTo should be consistent with equals method in java.

For the sake of clarity let's see **differences between List and Set interface in point format**:

1) List maintains insertion order of elements while Set doesn't maintain any ordering.

2) The list allows duplicate objects while Set doesn't allow any duplicates.

If you compare the implementation of List and Set interface like ArrayList vs HashSet you can see not only differences which are imposed by characteristics of List and Set but also implementation level differences like ArrayList uses an array as data structure while HashSet uses hashing mechanism.  
  
Here is a nice summary of difference between List, Set and Map in Java:

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https://www.pluralsight.com/courses/java-fundamentals-collections)

That's all on **differences between List and Set interface in Java**. bottom line is that List is an ordered collection while Set is unordered, List allows duplicates and Set don't allow duplicates. Let us know if you come across any other significant difference between List and Set in Java.

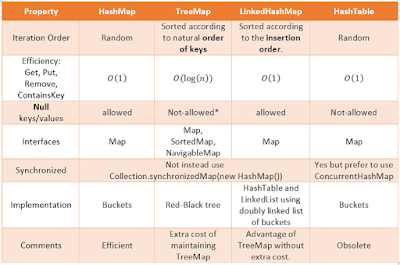
Read more: <https://www.java67.com/2012/08/difference-between-list-and-set-in-java.html#ixzz6M1Ivz4Ra>

## Difference between HashMap and Hashtable in Java

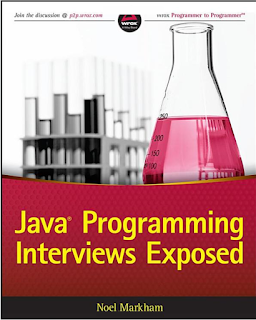
Both HashMap and Hashtable implements Map interface but there is some significant difference between them which is important to remember before deciding whether to use HashMap or Hashtable in Java. Some of them are thread-safety, synchronization, and speed. here are those differences :

1.The HashMap class is roughly equivalent to Hashtable, except that it is non-synchronized and permits nulls. (HashMap allows null values as key and value whereas [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html)doesn't allow nulls).

2. One of the major **differences between HashMap and Hashtable** is that HashMap is non-synchronized whereas Hashtable is synchronized, which means Hashtable is thread-safe and can be shared between multiple threads but HashMap can not be shared between multiple threads without proper synchronization. Java 5 introduces [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html)which is an alternative of Hashtable and provides better scalability than Hashtable in Java.

[](https://3.bp.blogspot.com/-iw2pMTU6pHo/VvP3HoW7XrI/AAAAAAAAFSQ/By53EkcfMGkoth857hnsc57XxYy8uwfAA/s1600/Difference+between+HashMap,+TreeMap,+LinkedHashMap+and+hashtable+in+Java.png)

3. Another significant difference between HashMap vs Hashtable is that Iterator in the HashMap is  a fail-fast iterator  while the enumerator for the Hashtable is not and throw ConcurrentModificationException if any other Thread modifies the map structurally  by adding or removing any element except Iterator's own remove() method. But this is not a guaranteed behavior and will be done by JVM on best effort. This is also an important [difference between Enumeration and Iterator in Java](http://javarevisited.blogspot.sg/2010/10/what-is-difference-between-enumeration.html).  
  
4. One more notable *difference between Hashtable and HashMap* is that because of thread-safety and synchronization Hashtable is much slower than HashMap if used in Single threaded environment. So if you don't need synchronization and HashMap are only used by one thread, it outperforms Hashtable in Java.  
  
5. HashMap does not guarantee that the order of the map will remain constant over time.  
  
If you are preparing this question as part of your Java interview preparation,  I suggest preparing every important topic as given in [Programming Interviews Exposed](http://www.amazon.com/dp/1118261364/?tag=javamysqlanta-20). It covers basics, core java, threads, a framework like Spring and Hibernate and many others key topics.

[](http://www.amazon.com/dp/1118261364/?tag=javamysqlanta-20)

### HashMap and Hashtable : note on Some Important Terms

1)Synchronized means only one Thread can modify a hash table at one point of time. Basically, it means that any thread before performing an update on a Hashtable will have to acquire a lock on the object while others will wait for the lock to be released.

2) Fail-safe is relevant from the context of iterators. If an [Iterator or ListIterator](http://javarevisited.blogspot.sg/2011/10/java-iterator-tutorial-example-list.html) has been created on a collection object and some other thread tries to modify the collection object "structurally", a concurrent modification exception will be thrown. It is possible for other threads though to invoke "set" method since it doesn't modify the collection "structurally". However, if prior to calling "set", the collection has been modified structurally, "IllegalArgumentException" will be thrown.

[Difference between HashMap and Hashtable in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html)

3) Structurally modification means deleting or inserting element which could effectively change the structure of the map.

HashMap can be synchronized by

Map m = Collections.synchronizeMap(hashMap);

In Summary, there are significant*differences between Hashtable and HashMap in Jav*a e.g. thread-safety and speed and based upon that only use Hashtable if you absolutely need thread-safety if you are running Java 5 consider using ConcurrentHashMap in Java.

Read more: <https://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html#ixzz6M23zWUBE>