Difference between collection and collections in java

Collection is an interface in the java collection framework. It is divided into two parts –

- Java util collection - It contains classes such as Set , queue , List and etc.
- Java util map - It contains classes such as Map , sortedMap and etc.

On the other hand , Collections is the one the utility class. Main purpose of this class is to provide convenience method to the developers

Sr. No.	**Key**	**Collection**	**Collections**
1	Basic	It is an interface in Java collection framework	It is a utility class in Collection framework
2	Static Methods	It doesn't has all static methods	It has all static method
3	Operation	It is used to store list of object in a single object	It is used to operate on collection.

Example of Collection and Collections

```
import java.util.ArrayList;
```

```java
import java.util.Collections;
import java.util.List;
public class Main {
   public static void main(String[] args) {
      // List
      List list = new ArrayList();
      list.add("HCL");
      list.add("DELL");
      // Sorting List in ascending order according
to the natural ordering
      Collections.sort(list);
      list.forEach(System.out::println);
   }
}
```

```java
import java.util.*;
public class CollectionsExample {
   public static void main(String a[]){
      List<Integer> list = new ArrayList<Integer>();
      list.add(46);
      list.add(67);
      list.add(24);
      list.add(16);
      list.add(8);
      list.add(12);
      System.out.println("Value of maximum element from the collec
tion: "+Collections.max(list));
System.out.println("Value of minimum element from the collection:
"+Collections.min(list));
   }
}
```

Java ArrayList

```
import java.util.*;
 public class ArrayListExample1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
//Creating arraylist
      list.add("Mango");//Adding object in arraylist
      list.add("Apple");
      list.add("Banana");
      list.add("Grapes");
      //Printing the arraylist object
      System.out.println(list);
 }
}
```

FileName: ArrayListExample2.java

```
import java.util.*;
public class ArrayListExample2{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Mango");//Adding object in arraylist
  list.add("Apple");
  list.add("Banana");
  list.add("Grapes");
  //Traversing list through Iterator
  Iterator itr=list.iterator();//getting the Iterator
  while(itr.hasNext()){//check if iterator has the elements
System.out.println(itr.next());
//printing the element and move to next
  }
 }
}
```

ArrayListExample3.java

```
import java.util.*;
public class ArrayListExample3{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Mango");//Adding object in arraylist
  list.add("Apple");
  list.add("Banana");
  list.add("Grapes");
  //Traversing list through for-each loop
  for(String fruit:list)
    System.out.println(fruit);

 }
}
```

ArrayListExample4.java

```
import java.util.*;
public class ArrayListExample4{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Mango");
  al.add("Apple");
  al.add("Banana");
  al.add("Grapes");
  //accessing the element
      System.out.println("Returning element: "+al.get(1));//
it will return the 2nd element, because index starts from 0
  //changing the element
  al.set(1,"Dates");
  //Traversing list
  for(String fruit:al)
    System.out.println(fruit);

 }
}
```

SortArrayList.java

```
import java.util.*;
class SortArrayList{
 public static void main(String args[]){
  //Creating a list of fruits
  List<String> list1=new ArrayList<String>();
  list1.add("Mango");
  list1.add("Apple");
  list1.add("Banana");
  list1.add("Grapes");
  //Sorting the list
  Collections.sort(list1);
   //Traversing list through the for-each loop
  for(String fruit:list1)
    System.out.println(fruit);

 System.out.println("Sorting numbers...");
  //Creating a list of numbers
  List<Integer> list2=new ArrayList<Integer>();
  list2.add(21);
  list2.add(11);
  list2.add(51);
  list2.add(1);
  //Sorting the list
  Collections.sort(list2);
   //Traversing list through the for-each loop
  for(Integer number:list2)
    System.out.println(number);
 }
}
```

Iterating Collection through remaining ways

```
import java.util.*;
class ArrayList4{
 public static void main(String args[]){
                ArrayList<String>  list=new  ArrayList<String>();//
Creating arraylist
           list.add("Ravi");//Adding object in arraylist
           list.add("Vijay");
           list.add("Ravi");
           list.add("Ajay");
```

```
         System.out.println("Traversing list through List Iterator:");
         //Here, element iterates in reverse order
            ListIterator<String> list1=list.listIterator(list.size());
            while(list1.hasPrevious())
            {
                String str=list1.previous();
                System.out.println(str);
            }
       System.out.println("Traversing list through for loop:");
         for(int i=0;i<list.size();i++)
         {
          System.out.println(list.get(i));
         }

       System.out.println("Traversing list through forEach() method:");

                                                                                  //
The forEach() method is a new feature, introduced in Java 8.
          list.forEach(a->{ //Here, we are using lambda expression
              System.out.println(a);
            });

           System.out.println("Traversing list through forEachRemainin
g() method:");
            Iterator<String> itr=list.iterator();
                                                    itr.forEachRemaining(a->   //
Here, we are using lambda expression
            {
          System.out.println(a);
            });
 }
}
```

User-defined class objects in Java ArrayList

FileName: ArrayList5.java

```
class Student{
  int rollno;
  String name;
  int age;
  Student(int rollno,String name,int age){
   this.rollno=rollno;
   this.name=name;
   this.age=age;
  }
}

import java.util.*;
 class ArrayList5{
 public static void main(String args[]){
  //Creating user-defined class objects
  Student s1=new Student(101,"Sonoo",23);
  Student s2=new Student(102,"Ravi",21);
  Student s2=new Student(103,"Hanumat",25);
  //creating arraylist
  ArrayList<Student> al=new ArrayList<Student>();
  al.add(s1);//adding Student class object
  al.add(s2);
  al.add(s3);
  //Getting Iterator
  Iterator itr=al.iterator();
  //traversing elements of ArrayList object
  while(itr.hasNext()){
    Student st=(Student)itr.next();
    System.out.println(st.rollno+" "+st.name+" "+st.age);
  }
 }
}
```

Java ArrayList example to remove elements

```
import java.util.*;
 class ArrayList8 {
        public static void main(String [] args)
        {
          ArrayList<String> al=new ArrayList<String>();
          al.add("Ravi");
          al.add("Vijay");
          al.add("Ajay");
          al.add("Anuj");
          al.add("Gaurav");
          System.out.println("An initial list of elements: "+al);
          //Removing specific element from arraylist
          al.remove("Vijay");
           System.out.println("After invoking remove(object) method: "+
al);
          //Removing element on the basis of specific position
          al.remove(0);
          System.out.println("After invoking remove(index) method: "+a
l);

          //Creating another arraylist
          ArrayList<String> al2=new ArrayList<String>();
          al2.add("Ravi");
          al2.add("Hanumat");
          //Adding new elements to arraylist
          al.addAll(al2);
          System.out.println("Updated list : "+al);
          //Removing all the new elements from arraylist
          al.removeAll(al2);
            System.out.println("After invoking removeAll() method: "+al);

          //Removing elements on the basis of specified condition
                              al.removeIf(str  ->  str.contains("Ajay"));         //
Here, we are using Lambda expression
          System.out.println("After invoking removeIf() method: "+al);
          //Removing all the elements available in the list
          al.clear();
          System.out.println("After invoking clear() method: "+al);
       }
    }
```

Java ArrayList Example: Book

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int
quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class ArrayListExample20 {
public static void main(String[] args) {
    //Creating list of Books
    List<Book> list=new ArrayList<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",
8);
    Book b2=new Book(102,"Data Communications and Networking
","Forouzan","Mc Graw Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);

    //Adding Books to list
    list.add(b1);
    list.add(b2);
    list.add(b3);
    //Traversing list
    for(Book b:list){
        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publis
her+" "+b.quantity);
    }
}
}
```

Java LinkedList Example

```
import java.util.*;
public class LinkedList1{
 public static void main(String args[]){
  LinkedList<String> al=new LinkedList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");

  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Java LinkedList example to remove elements

```
import java.util.*;
public class LinkedList3 {

        public static void main(String [] args)
        {
           LinkedList<String> ll=new LinkedList<String>();
           ll.add("Ravi");
           ll.add("Vijay");
           ll.add("Ajay");
           ll.add("Anuj");
           ll.add("Gaurav");
           ll.add("Harsh");
           ll.add("Virat");
           ll.add("Gaurav");
           ll.add("Harsh");
           ll.add("Amit");
           System.out.println("Initial list of elements: "+ll);
        //Removing specific element from arraylist
     ll.remove("Vijay");
   System.out.println("After invoking remove(object) method: "+ll);
        //Removing element on the basis of specific position
     ll.remove(0);
     System.out.println("After invoking remove(index) method: "+ll);
     LinkedList<String> ll2=new LinkedList<String>();
    ll2.add("Ravi");
     ll2.add("Hanumat");
        // Adding new elements to arraylist
      ll.addAll(ll2);
      System.out.println("Updated list : "+ll);
        //Removing all the new elements from arraylist
      ll.removeAll(ll2);
      System.out.println("After invoking removeAll() method: "+ll);
        //Removing first element from the list
      ll.removeFirst();
     System.out.println("After invoking removeFirst() method: "+ll);
         //Removing first element from the list
      ll.removeLast();
      System.out.println("After invoking removeLast() method: "+ll);
         //Removing first occurrence of element from the list
      ll.removeFirstOccurrence("Gaurav");
  System.out.println("After invoking removeFirstOccurrence() metho
d: "+ll);
         //Removing last occurrence of element from the list
      ll.removeLastOccurrence("Harsh");
    System.out.println("After invoking removeLastOccurrence() meth
od: "+ll);

             //Removing all the elements available in the list
     ll.clear();
      System.out.println("After invoking clear() method: "+ll);
       }
    }
```

Java LinkedList Example: Book

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int
quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class LinkedListExample {
public static void main(String[] args) {
    //Creating list of Books
    List<Book> list=new LinkedList<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",
8);
    Book b2=new Book(102,"Data Communications & Networking","
Forouzan","Mc Graw Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);

    //Adding Books to list
    list.add(b1);
    list.add(b2);
    list.add(b3);
    //Traversing list
    for(Book b:list){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publishe
r+" "+b.quantity);
    }
}
}
```

How to convert Array to List

```
import java.util.*;
public class ArrayToListExample{
public static void main(String args[]){
//Creating Array
String[] array={"Java","Python","PHP","C++"};
System.out.println("Printing Array: "+Arrays.toString(array));
//Converting Array to List
List<String> list=new ArrayList<String>();
for(String lang:array){
list.add(lang);
}
System.out.println("Printing List: "+list);
}
}
```

How to convert List to Array

```
import java.util.*;
public class ListToArrayExample{
public static void main(String args[]){
 List<String> fruitList = new ArrayList<>();
 fruitList.add("Mango");
 fruitList.add("Banana");
 fruitList.add("Apple");
 fruitList.add("Strawberry");
 //Converting ArrayList to Array
 String[] array = fruitList.toArray(new String[fruitList.size()]);
 System.out.println("Printing Array: "+q);
 System.out.println("Printing List: "+fruitList);
}
}
```

Get and Set Element in List

```
import java.util.*;
public class ListExample2{
 public static void main(String args[]){
 //Creating a List
 List<String> list=new ArrayList<String>();
 //Adding elements in the List
 list.add("Mango");
 list.add("Apple");
 list.add("Banana");
 list.add("Grapes");
 //accessing the element
  System.out.println("Returning element: "+list.get(1));//
it will return the 2nd element, because index starts from 0
 //changing the element
 list.set(1,"Dates");
 //Iterating the List element using for-each loop
 for(String fruit:list)
  System.out.println(fruit);

 }
}
```

How to Sort List

```
import java.util.*;
class SortArrayList{
 public static void main(String args[]){
  //Creating a list of fruits
  List<String> list1=new ArrayList<String>();
  list1.add("Mango");
  list1.add("Apple");
  list1.add("Banana");
  list1.add("Grapes");
  //Sorting the list
  Collections.sort(list1);
   //Traversing list through the for-each loop
  for(String fruit:list1)
    System.out.println(fruit);

 System.out.println("Sorting numbers...");
  //Creating a list of numbers
  List<Integer> list2=new ArrayList<Integer>();
  list2.add(21);
  list2.add(11);
  list2.add(51);
  list2.add(1);
  //Sorting the list
  Collections.sort(list2);
   //Traversing list through the for-each loop
  for(Integer number:list2)
    System.out.println(number);
 }

}
```

Java ListIterator Interface

ListIterator Interface is used to traverse the element in a backward and forward direction.

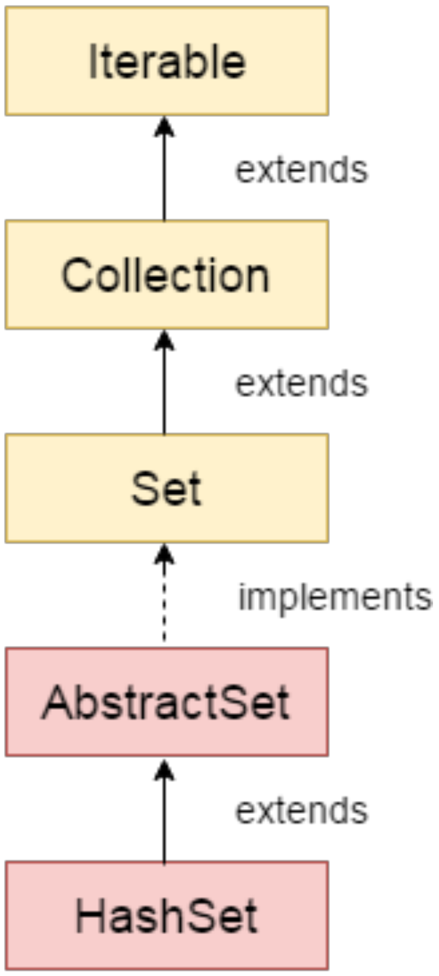
```
import java.util.*;
public class ListIteratorExample1{
public static void main(String args[]){
List<String> al=new ArrayList<String>();
        al.add("Amit");
        al.add("Vijay");
        al.add("Kumar");
        al.add(1,"Sachin");
        ListIterator<String> itr=al.listIterator();
         System.out.println("Traversing elements in forward direction");

        while(itr.hasNext()){

                                      System.out.println("index:"+itr.nextIndex()
+" value:"+itr.next());
        }
         System.out.println("Traversing elements in backward direction
");
        while(itr.hasPrevious()){
```

```
                          System.out.println("index:"+itr.previousIndex()
+" value:"+itr.previous());
        }
}
}
```

Java HashSet



Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing.**
- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.

- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.
- The initial default capacity of HashSet is 16, and the load factor is 0.75.

```
import java.util.*;
class HashSet1{
 public static void main(String args[]){
  //Creating HashSet and adding elements
    HashSet<String> set=new HashSet();
           set.add("One");
           set.add("Two");
           set.add("Three");
           set.add("Four");
           set.add("Five");
           set.add(“Two”);
           Iterator<String> i=set.iterator();
           while(i.hasNext())
           {
           System.out.println(i.next());
           }
 }
}
```

Java HashSet example ignoring duplicate elements

```
import java.util.*;
class HashSet2{
 public static void main(String args[]){
  //Creating HashSet and adding elements
  HashSet<String> set=new HashSet<String>();
  set.add("Ravi");
  set.add("Vijay");
  set.add("Ravi");
  set.add("Ajay");
  //Traversing elements
  Iterator<String> itr=set.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Java HashSet Example: Book

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int
quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class HashSetExample {
public static void main(String[] args) {
HashSet<Book> set=new HashSet<Book>();
    //Creating Books
Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
Book b2=new Book(102,"Data Communications & Networking","F
orouzan","Mc Graw Hill",4);
 Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to HashSet
    set.add(b1);
    set.add(b2);
    set.add(b3);
    //Traversing HashSet
    for(Book b:set){
     System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publishe
r+" "+b.quantity);
```

```
    }
}
```

Java LinkedHashSet Class

Java LinkedHashSet class is a Hashtable and Linked list implementation of the Set interface. It inherits the HashSet class and implements the Set interface.

The important points about the Java LinkedHashSet class are:

- Java LinkedHashSet class contains unique elements only like HashSet.
- Java LinkedHashSet class provides all optional set operations and permits null elements.
- Java LinkedHashSet class is non-synchronized.
- Java LinkedHashSet class maintains insertion order.

```
import java.util.*;
class LinkedHashSet1{
 public static void main(String args[]){
 //Creating HashSet and adding elements
        LinkedHashSet<String> set=new LinkedHashSet();
               set.add("One");
               set.add("Two");
               set.add("Three");
               set.add("Four");
               set.add("Five");
               Iterator<String> i=set.iterator();
               while(i.hasNext())
               {
               System.out.println(i.next());
               }
 }
}
```

Remove Elements Using LinkeHashSet Class

```
import java.util.*;
```

```
public class LinkedHashSet3
{

// main method
public static void main(String argvs[])
{

// Creating an empty LinekdhashSet of string type
LinkedHashSet<String> lhs = new LinkedHashSet<String>();

// Adding elements to the above Set
// by invoking the add() method
lhs.add("Java");
lhs.add("T");
lhs.add("Point");
lhs.add("Good");
lhs.add("Website");

// displaying all the elements on the console
System.out.println("The hash set is: " + lhs);

// Removing an element from the above linked Set

System.out.println(lhs.remove("Good"));

// After removing the element
System.out.println("After removing the element, the hash set is: " +
lhs);

// returns false
System.out.println(lhs.remove("For"));

}
}
```

Java TreeSet class

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

The important points about the Java TreeSet class are:

- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quiet fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.

```
import java.util.*;
class TreeSet1{
 public static void main(String args[]){
  //Creating and adding elements
  TreeSet<String> al=new TreeSet<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
  //Traversing elements
  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

TreeSet2.java

```
import java.util.*;
class TreeSet2{
 public static void main(String args[]){
 TreeSet<String> set=new TreeSet<String>();
         set.add("Ravi");
         set.add("Vijay");
         set.add("Ajay");
         System.out.println("Traversing element through Iterator in des
cending order");
        Iterator i=set.descendingIterator();
        while(i.hasNext())
        {
            System.out.println(i.next());
        }

 }
}
```

Java TreeSet Example 4:

```
import java.util.*;
class TreeSet4{
 public static void main(String args[]){
  TreeSet<String> set=new TreeSet<String>();
        set.add("A");
        set.add("B");
        set.add("C");
        set.add("D");
        set.add("E");
        System.out.println("Initial Set: "+set);

        System.out.println("Reverse Set: "+set.descendingSet());

        System.out.println("Head Set: "+set.headSet("C", true));

        System.out.println("SubSet: "+set.subSet("A", false, "E", true
));

        System.out.println("TailSet: "+set.tailSet("C", false));
 }
}
```

Java PriorityQueue Example

```
import java.util.*;
```

```
class TestCollection12{
public static void main(String args[]){
PriorityQueue<String> queue=new PriorityQueue<String>();
queue.add("Amit");
queue.add("Vijay");
queue.add("Karan");
queue.add("Jai");
queue.add("Rahul");
System.out.println("head:"+queue.element());
System.out.println("head:"+queue.peek());
System.out.println("iterating the queue elements:");
Iterator itr=queue.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
queue.remove();
queue.poll();
System.out.println("after removing two elements:");
Iterator<String> itr2=queue.iterator();
while(itr2.hasNext()){
System.out.println(itr2.next());
}
}
}
```

Java PriorityQueue Example: Book

```
import java.util.*;
class Book implements Comparable<Book>{
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int
quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
public int compareTo(Book b) {
    if(id>b.id){
        return 1;
    }else if(id<b.id){
        return -1;
    }else{
    return 0;
    }
}
}
public class LinkedListExample {
public static void main(String[] args) {
    Queue<Book> queue=new PriorityQueue<Book>();
    //Creating Books
    Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",
8);
    Book b2=new Book(233,"Operating System","Galvin","Wiley",6);

    Book b3=new Book(101,"Data Communications & Networking","
Forouzan","Mc Graw Hill",4);
    //Adding Books to the queue
    queue.add(b1);
    queue.add(b2);
    queue.add(b3);
    System.out.println("Traversing the queue elements:");
    //Traversing queue elements
    for(Book b:queue){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publishe
r+" "+b.quantity);
    }
    queue.remove();
    System.out.println("After removing one book record:");
    for(Book b:queue){
        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publis
her+" "+b.quantity);
        }
}
}
```

Java Map Example: Non-Generic (Old Style)

```
import java.util.*;
public class MapExample1 {
public static void main(String[] args) {
    Map map=new HashMap();
    //Adding elements to map
    map.put(1,"Amit");
    map.put(5,"Rahul");
    map.put(2,"Jai");
    map.put(6,"Amit");
    //Traversing Map
Set set=map.entrySet();
//Converting to Set so that we can traverse
    Iterator itr=set.iterator();
    while(itr.hasNext()){

Converting to Map.Entry so that we can get key and value separate
ly
        Map.Entry entry=(Map.Entry)itr.next();
        System.out.println(entry.getKey()+" "+entry.getValue());
    }
}
}
```

Java HashMap

Java **HashMap** class implements the Map interface which allows us *to store key and value pair*, where keys should be unique.

Points to remember

- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.

- Java HashMap may have one null key and multiple null values.
- Java HashMap is non synchronized.
- Java HashMap maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

```
import java.util.*;
public class HashMapExample1{
 public static void main(String args[]){
   HashMap<Integer,String> map=new HashMap<Integer,String>();
//Creating HashMap
   map.put(1,"Mango");  //Put elements in Map
   map.put(2,"Apple");
   map.put(3,"Banana");
   map.put(4,"Grapes");

   System.out.println("Iterating Hashmap...");
   for(Map.Entry m : map.entrySet()){
    System.out.println(m.getKey()+" "+m.getValue());
   }
}
}
```

No Duplicate Key on HashMap

You cannot store duplicate keys in HashMap. However, if you try to store duplicate key with another value, it will replace the value.

```
import java.util.*;
public class HashMapExample2{
 public static void main(String args[]){
   HashMap<Integer,String> map=new HashMap<Integer,String>();
//Creating HashMap
   map.put(1,"Mango");  //Put elements in Map
   map.put(2,"Apple");
   map.put(3,"Banana");
   map.put(1,"Grapes"); //trying duplicate key
   System.out.println("Iterating Hashmap...");
   for(Map.Entry m : map.entrySet()){
    System.out.println(m.getKey()+" "+m.getValue());
   }
}
}
```

Java HashMap example to add() elements

```
import java.util.*;
class HashMap1{
 public static void main(String args[]){
     HashMap<Integer,String>  hm=new  HashMap<Integer,String>();

    System.out.println("Initial list of elements: "+hm);
      hm.put(100,"Amit");
      hm.put(101,"Vijay");
      hm.put(102,"Rahul");

      System.out.println("After invoking put() method ");
      for(Map.Entry m:hm.entrySet()){
       System.out.println(m.getKey()+" "+m.getValue());
      }

      hm.putIfAbsent(103, "Gaurav");
      System.out.println("After invoking putIfAbsent() method ");
      for(Map.Entry m:hm.entrySet()){
           System.out.println(m.getKey()+" "+m.getValue());
          }
  HashMap<Integer,String> map=new HashMap<Integer,String>();
      map.put(104,"Ravi");
      map.putAll(hm);
      System.out.println("After invoking putAll() method ");
      for(Map.Entry m:map.entrySet()){
           System.out.println(m.getKey()+" "+m.getValue());
          }
 }
}
```

Java HashMap example to remove() elements

```
import java.util.*;
public class HashMap2 {
   public static void main(String args[]) {
  HashMap<Integer,String>  map=new  HashMap<Integer,String>();

      map.put(100,"Amit");
      map.put(101,"Vijay");
      map.put(102,"Rahul");
      map.put(103, "Gaurav");
    System.out.println("Initial list of elements: "+map);
    //key-based removal
    map.remove(100);
    System.out.println("Updated list of elements: "+map);
    //value-based removal
    map.remove(101);
    System.out.println("Updated list of elements: "+map);
    //key-value pair based removal
    map.remove(102, "Rahul");
    System.out.println("Updated list of elements: "+map);
   }
}
```

Java HashMap example to replace() elements

```
import java.util.*;
class HashMap3{
 public static void main(String args[]){
     HashMap<Integer,String> hm=new  HashMap<Integer,String>();

      hm.put(100,"Amit");
      hm.put(101,"Vijay");
      hm.put(102,"Rahul");
      System.out.println("Initial list of elements:");
     for(Map.Entry m:hm.entrySet())
     {
        System.out.println(m.getKey()+" "+m.getValue());
```

```
    }
    System.out.println("Updated list of elements:");
    hm.replace(102, "Gaurav");
    for(Map.Entry m:hm.entrySet())
    {
      System.out.println(m.getKey()+" "+m.getValue());
    }
    System.out.println("Updated list of elements:");
    hm.replace(101, "Vijay", "Ravi");
    for(Map.Entry m:hm.entrySet())
    {
      System.out.println(m.getKey()+" "+m.getValue());
    }
    System.out.println("Updated list of elements:");
    hm.replaceAll((k,v) -> "Ajay");
    for(Map.Entry m:hm.entrySet())
    {
      System.out.println(m.getKey()+" "+m.getValue());
    }
 }
}
```

Java LinkedHashMap class

```
import java.util.*;
class LinkedHashMap1{
 public static void main(String args[]){

   LinkedHashMap<Integer,String> hm=new LinkedHashMap<Integ
er,String>();

  hm.put(100,"Amit");
  hm.put(101,"Vijay");
  hm.put(102,"Rahul");

for(Map.Entry m:hm.entrySet()){
   System.out.println(m.getKey()+" "+m.getValue());
  }
 }
}
```

Java LinkedHashMap Example: Key-Value pair

```
import java.util.*;
class LinkedHashMap2{
 public static void main(String args[]){
   LinkedHashMap<Integer, String> map = new LinkedHashMap<In
teger, String>();
      map.put(100,"Amit");
     map.put(101,"Vijay");
     map.put(102,"Rahul");
       //Fetching key
       System.out.println("Keys: "+map.keySet());
       //Fetching value
       System.out.println("Values: "+map.values());
       //Fetching key-value pair
       System.out.println("Key-Value pairs: "+map.entrySet());
 }
}
```

Java TreeMap class

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

- Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- Java TreeMap contains only unique elements.
- Java TreeMap cannot have a null key but can have multiple null values.
- Java TreeMap is non synchronized.

- Java TreeMap maintains ascending order.

Java TreeMap Example

```
import java.util.*;
class TreeMap1{
 public static void main(String args[]){
   TreeMap<Integer,String> map=new TreeMap<Integer,String>();
      map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");

      for(Map.Entry m:map.entrySet()){
       System.out.println(m.getKey()+" "+m.getValue());
      }
 }
}
```

Java TreeMap Example: remove()

```
import java.util.*;
public class TreeMap2 {
   public static void main(String args[]) {
       TreeMap<Integer,String> map=new TreeMap<Integer,String>();

      map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");
      System.out.println("Before invoking remove() method");
      for(Map.Entry m:map.entrySet())
      {
          System.out.println(m.getKey()+" "+m.getValue());
      }
      map.remove(102);
      System.out.println("After invoking remove() method");
      for(Map.Entry m:map.entrySet())
      {
         System.out.println(m.getKey()+" "+m.getValue());
      }
      }
}
```

Java TreeMap Example: SortedMap

```
import java.util.*;
class TreeMap4{
 public static void main(String args[]){
   SortedMap<Integer,String> map=new TreeMap<Integer,String>();

      map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");
                                                    //Returns key-
value pairs whose keys are less than the specified key.
      System.out.println("headMap: "+map.headMap(102));
                                                    //Returns key-
value pairs whose keys are greater than or equal to the specified ke
y.
      System.out.println("tailMap: "+map.tailMap(102));
      //Returns key-value pairs exists in between the specified key.
      System.out.println("subMap: "+map.subMap(100, 102));
 }
}
```

What is difference between HashMap and TreeMap?

HashMap	TreeMap

1) HashMap can contain one null key.	TreeMap cannot contain any null key.
2) HashMap maintains no order.	TreeMap maintains ascending order.

Java Hashtable class

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

Points to remember

- A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
- Java Hashtable class contains unique elements.
- Java Hashtable class doesn't allow null key or value.
- Java Hashtable class is synchronized.

```
import java.util.*;
class Hashtable1{
 public static void main(String args[]){
  Hashtable<Integer,String> hm=new Hashtable<Integer,String>();

  hm.put(100,"Amit");
  hm.put(102,"Ravi");
  hm.put(101,"Vijay");
  hm.put(103,"Rahul");

  for(Map.Entry m:hm.entrySet()){
   System.out.println(m.getKey()+" "+m.getValue());
  }
 }
}
```

Java Hashtable Example: remove()

```
import java.util.*;
public class Hashtable2 {
   public static void main(String args[]) {
   Hashtable<Integer,String> map=new Hashtable<Integer,String>();

     map.put(100,"Amit");
     map.put(102,"Ravi");
     map.put(101,"Vijay");
     map.put(103,"Rahul");
     System.out.println("Before remove: "+ map);
       // Remove value for key 102
       map.remove(102);
       System.out.println("After remove: "+ map);
   }
}
```

Java Hashtable Example: putIfAbsent()

```
import java.util.*;
class Hashtable4{
 public static void main(String args[]){
    Hashtable<Integer,String> map=new Hashtable<Integer,String>()
;
     map.put(100,"Amit");
     map.put(102,"Ravi");
     map.put(101,"Vijay");
     map.put(103,"Rahul");
     System.out.println("Initial Map: "+map);
     //Inserts, as the specified pair is unique
     map.putIfAbsent(104,"Gaurav");
     System.out.println("Updated Map: "+map);
     //Returns the current value, as the specified pair already exist
     map.putIfAbsent(101,"Vijay");
     System.out.println("Updated Map: "+map);
 }
}
```

Sorting in Collection

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

Collections class provides static methods for sorting the elements of a collection. If collection elements are of a Set type, we can use TreeSet. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

Example to sort string objects

```
import java.util.*;
class TestSort1{
public static void main(String args[]){

ArrayList<String> al=new ArrayList<String>();
al.add("Viru");
al.add("Saurav");
al.add("Mukesh");
al.add("Tahir");

Collections.sort(al);
Iterator itr=al.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
 }
}
}
```

Example to sort string objects in reverse order

```
import java.util.*;
```

```
class TestSort2{
public static void main(String args[]){

ArrayList<String> al=new ArrayList<String>();
        al.add("Viru");
        al.add("Saurav");
        al.add("Mukesh");
        al.add("Tahir");

        Collections.sort(al,Collections.reverseOrder());
        Iterator i=al.iterator();
        while(i.hasNext())
        {
            System.out.println(i.next());
        }
}
}
```

Example to sort Wrapper class objects

```
import java.util.*;
class TestSort3{
public static void main(String args[]){

ArrayList al=new ArrayList();
al.add(Integer.valueOf(201));
al.add(Integer.valueOf(101));
al.add(230); //
internally will be converted into objects as Integer.valueOf(230)

Collections.sort(al);

Iterator itr=al.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
 }
}
}
```

Example to sort user-defined class objects

```
import java.util.*;

class Student implements Comparable<Student> {
    public String name;
  public Student(String name) {
    this.name = name;
  }
  public int compareTo(Student person) {
    return name.compareTo(person.name);

  }
}
public class TestSort4 {
  public static void main(String[] args) {
      ArrayList<Student> al=new ArrayList<Student>();
      al.add(new Student("Viru"));
      al.add(new Student("Saurav"));
      al.add(new Student("Mukesh"));
      al.add(new Student("Tahir"));

    Collections.sort(al);
    for (Student s : al) {
      System.out.println(s.name);
    }
  }
}
```