What is collections

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

**A Collection represents a single unit of objects, i.e., a group.**

**All the operations that you perform on a data such as**

1. searching,
2. sorting,
3. insertion,
4. manipulation,

**What is a framework in Java**

1. It provides readymade architecture.
2. It represents a set of classes and interface.
3. It is optional.
4. deletion, etc. can be achieved by Java Collections.

**What is Collection framework**

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

# **What are the main differences between Collection and Collections in Java?**

**difference between collections and collection in java**

* Major difference between Collection and Collections is **Collection is an interface** and **Collections is a class**.
* Both are belongs to java.util package
* Collection is base interface for list set and queue.
* Collections is a class and it is called utility class.
* Collections utility class contains some predefined methods so that we can use while working with Collection type of classes(treeset, arraylist, linkedlist etc.)

**Why we need collections**

**Collections** are used almost in every programming language.

**Collections** are used in situations where data is dynamic. **Collections** allow adding an element, deleting an element and host of other operations.

Types of collections

**Why collection is called framework in Java?**

The **Java collections framework** (JCF) is a set of classes and interfaces that implement commonly reusable **collection** data structures. Although referred to as a **framework**, it works in a manner of a library. The JCF provides both interfaces that define various **collections** and classes that implement them.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(Object element) | is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection c) | is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | is used to delete an element from this collection. |
| 4 | public boolean removeAll(Collection c) | is used to delete all the elements of specified collection from the invoking collection. |
| 5 | public boolean retainAll(Collection c) | is used to delete all the elements of invoking collection except the specified collection. |
| 6 | public int size() | return the total number of elements in the collection. |
| 7 | public void clear() | removes the total no. of elements from the collection. |
| 8 | public boolean contains(Object element) | is used to search an element. |
| 9 | public boolean containsAll(Collection c) | is used to search the specified collection in this collection. |
| 10 | public Iterator iterator() | returns an iterator. |
| 11 | public Object[] toArray() | converts collection into array. |
| 12 | public boolean isEmpty() | checks if collection is empty. |
| 13 | public boolean equals(Object element) | matches two collections. |
| 14 | public int hashCode() | returns the hash code number of the collection. |

### **Methods of Collection interface**

### **Hierarchy of Collection Framework**

Collection: in collection we have three interfaces

1. **List**
2. **Set**
3. **Map**

**List:**

List is extending

1. ArrayList and
2. LinkedList

**List follow the order and it will allow duplicates.**

# **Java ArrayList class**

**Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.**

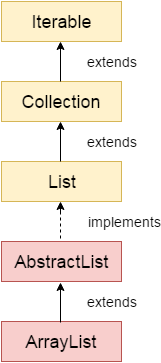
**Array list is used to get the data from particular position using index.**

The important points about Java ArrayList class are:

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list. **That’s why this is called non- synchronized**

### Hierarchy of ArrayList class

As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order.



### **Constructors of Java ArrayList**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| ArrayList() | It is used to build an empty array list. |
| ArrayList(Collection c) | It is used to build an array list that is initialized with the elements of the collection c. |
| ArrayList(int capacity) | It is used to build an array list that has the specified initial capacity. |

### Methods of Java ArrayList

|  |  |
| --- | --- |
| **Method** | **Description** |
| void add(int index, Object element) | It is used to insert the specified element at the specified position index in a list. |
| boolean addAll(Collection c) | It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. |
| void clear() | It is used to remove all of the elements from this list. |
| int lastIndexOf(Object o) | It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. |

**Void get(int index) get the element in specified index**

**Void remove(int index) remove the element in specified index**

**void contains we can search with some particular name in one element**

**Void clear clear the list**

**Void isEmpty tell us whether list is empty or not**

# **Generics in Java**

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects.

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

#### Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

**1) Type-safety :** We can hold only a single type of objects in generics. It doesn’t allow to store other objects.

**2) Type casting is not required:** There is no need to typecast the object.

**Before Generics, we need to type cast.**

1. **List list = new ArrayList();**
2. **list.add("hello");**
3. **String s = (String) list.get(0);//typecasting**

**After Generics, we don't need to typecast the object.**

1. **List<String> list = new ArrayList<String>();**
2. **list.add("hello");**
3. **String s = list.get(0);**

**3) Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

1. List<String> list = new ArrayList<String>();
2. list.add("hello");
3. list.add(32);//Compile Time Error

### **Java Non-generic Vs Generic Collection**

J**ava collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.**

**J**ava new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

**Let's see the old non-generic example of creating java collection.**

1. **ArrayList al=new ArrayList();//creating old non-generic arraylist**

**Let's see the new generic example of creating java collection.**

1. **ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist**

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives compile time error.

### **Iterating Collection through for-each loop**

import java.util.\*;

class TestCollection2{

 public static void main(String args[]){

  ArrayList<String> al=new ArrayList<String>();

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ravi");

  al.add("Ajay");

  for(String obj:al)

    System.out.println(obj);

 }

}

**Output**

Ravi

Vijay

Ravi

Ajay

### Example of addAll(Collection c) method

**ArrayList<String> al=new ArrayList<String>();**

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ajay");

**ArrayList<String> al2=new ArrayList<String>();**

  al2.add("Sonoo");

  al2.add("Hanumat");

**al.addAll(al2);//adding second list in first list**

**ouTput:**

Ravi

Vijay

Ajay

Sonoo

Hanumat

### Example of removeAll(Collection c) method

**ArrayList<String> al=new ArrayList<String>();**

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ajay");

**ArrayList<String> al2=new ArrayList<String>();**

  al2.add("Ravi");

  al2.add("Hanumat");

**al.removeAll(al2);//adding second list in first list**

**ouTput:**

Vijay

Ajay

### Example of retainAll(Collection c) method

**ArrayList<String> al=new ArrayList<String>();**

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ajay");

**ArrayList<String> al2=new ArrayList<String>();**

  al2.add("Ravi");

  al2.add("Hanumat");

**al.retainAll(al2);//adding second list in first list**

**ouTput:**

Ravi

### **Java ArrayList Example: BAG**

**package** List\_Set;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

**public** **class** List\_FindMaxPriceWithDuplicates {

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

List<Integer> bagprice=**new** ArrayList<Integer>();

bagprice.add(999);

bagprice.add(666);

bagprice.add(545);

bagprice.add(999);

bagprice.add(666);

bagprice.add(345);

bagprice.add(666);

//System.out.println("list of bagPrice is"+bagprice);

//ouput will be as follows

//list of bagPrice is[999, 666, 545, 999, 666, 345, 666]

//if you want to get output in proper oredr use for loop

**for** (Integer eachBagPrice : bagprice) {

System.out.println("list of bagPrice is "+eachBagPrice);

}

**Output:**

list of bagPrice is 999

list of bagPrice is 666

list of bagPrice is 545

list of bagPrice is 999

list of bagPrice is 666

list of bagPrice is 345

# **Java LinkedList class**

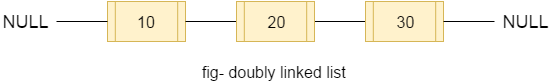
Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
* Java LinkedList class can be used as list, stack or queue.

### **Doubly Linked List**

In case of doubly linked list, we can add or remove elements from both side.



### Constructors of Java LinkedList

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| LinkedList() | It is used to construct an empty list. |
| LinkedList(Collection c) | It is used to construct a list containing the elements of the specified collection, in the order they are returned by the collection's iterator. |

### Methods of Java LinkedList

|  |  |
| --- | --- |
| **Method** | **Description** |
| void add(int index, Object element) | It is used to insert the specified element at the specified position index in a list. |
| void addFirst(Object o) | It is used to insert the given element at the beginning of a list. |
| void addLast(Object o) | It is used to append the given element to the end of a list. |
| int size() | It is used to return the number of elements in a list |
| boolean add(Object o) | It is used to append the specified element to the end of a list. |
| boolean contains(Object o) | It is used to return true if the list contains a specified element. |
| boolean remove(Object o) | It is used to remove the first occurence of the specified element in a list. |
| Object getFirst() | It is used to return the first element in a list. |
| Object getLast() | It is used to return the last element in a list. |
| int indexOf(Object o) | It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element. |
| int lastIndexOf(Object o) | It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element. |

### Java LinkedList Example

1. import java.util.\*;
2. public class TestCollection7{
3. public static void main(String args[]){
4. LinkedList<String> al=new LinkedList<String>();
5. al.add("Ravi");
6. al.add("Vijay");
7. al.add("Ravi");
8. al.add("Ajay");
10. Iterator<String> itr=al.iterator();
11. while(itr.hasNext()){
12. System.out.println(itr.next());
13. }
14. }
15. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection7)

Output:Ravi

Vijay

Ravi

Ajay

# **Difference between ArrayList and LinkedList**

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

But there are many differences between ArrayList and LinkedList classes that are given below.

**Synchromized:**

**A synchronized collection implies that the class is thread safe.**

Collection classes are not synchronized by default. But if you want a synchronized collection, you can use static method java.util.Collections.synchronizedCollection(Collection<T> c). It will create wrapper over your collection object. So, actually, your collection object will not be synchronized, but you will access your object's method via synchronized methods in wrapper object.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses 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 doubly linked list so no bit shifting is required in memory. |
| 3) 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. |

### **Example of ArrayList and LinkedList in Java**

1. import java.util.\*;
2. class TestArrayLinked{
3. public static void main(String args[]){
5. List<String> al=new ArrayList<String>();//creating arraylist
6. al.add("Ravi");//adding object in arraylist
7. al.add("Vijay");
8. al.add("Ravi");
9. al.add("Ajay");
11. List<String> al2=new LinkedList<String>();//creating linkedlist
12. al2.add("James");//adding object in linkedlist
13. al2.add("Serena");
14. al2.add("Swati");
15. al2.add("Junaid");
17. System.out.println("arraylist: "+al);
18. System.out.println("linkedlist: "+al2);
19. }
20. }

**Output:**

arraylist: [Ravi,Vijay,Ravi,Ajay]

linkedlist: [James,Serena,Swati,Junaid]

# Java HashSet class

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.

## **Difference between List and Set**

List can contain duplicate elements whereas Set contains unique elements only.

## **Methods of Java HashSet class**

Various methods of Java HashSet class are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Modifier & Type** | **Method** | **Description** |
| 1) | Boolean | [add(Object o)](https://www.javatpoint.com/java-hashset-add-method) | It is used to adds the specified element to this set if it is not already present. |
| 2) | Void | [clear()](https://www.javatpoint.com/java-hashset-clear-method) | It is used to remove all of the elements from this set. |
| 3) | object | [clone()](https://www.javatpoint.com/java-hashset-clone-method) | It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned. |
| 4) | boolean | [contains(Object o)](https://www.javatpoint.com/java-hashset-contains-method) | It is used to return true if this set contains the specified element. |
| 5) | boolean | [isEmpty()](https://www.javatpoint.com/java-hashset-isempty-method) | It is used to return true if this set contains no elements. |
| 6) | Iterator<E> | [iterator()](https://www.javatpoint.com/java-hashset-iterator-method) | It is used to return an iterator over the elements in this set. |
| 7) | boolean | [remove(Object o)](https://www.javatpoint.com/java-hashset-remove-method) | It is used to remove the specified element from this set if it is present. |
| 8) | int | [size()](https://www.javatpoint.com/java-hashset-size-method) | It is used to return the number of elements in this set. |
| 9) | Spliterator<E> | [spliterator()](https://www.javatpoint.com/java-hashset-spliterator-method) | It is used to create a late-binding and fail-fast Spliterator over the elements in this set. |

# **Java LinkedHashSet class**

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

The important points about Java LinkedHashSet class are:

* Contains unique elements only like HashSet.
* Provides all optional set operations, and permits null elements.
* Maintains insertion order.

### **Example of LinkedHashSet class:**

1. import java.util.\*;
2. class TestCollection10{
3. public static void main(String args[]){
4. LinkedHashSet<String> al=new LinkedHashSet<String>();
5. al.add("Ravi");
6. al.add("Vijay");
7. al.add("Ravi");
8. al.add("Ajay");
9. Iterator<String> itr=al.iterator();
10. while(itr.hasNext()){
11. System.out.println(itr.next());
12. }
13. }
14. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection10)

Ravi

Vijay

Ajay

# **Java TreeSet class**

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

The important points about Java TreeSet class are:

* Contains unique elements only like HashSet.
* Access and retrieval times are quiet fast.
* Maintains ascending order.

### **Java TreeSet Example**

1. import java.util.\*;
2. class TestCollection11{
3. public static void main(String args[]){
4. //Creating and adding elements
5. TreeSet<String> al=new TreeSet<String>();
6. al.add("Ravi");
7. al.add("Vijay");
8. al.add("Ravi");
9. al.add("Ajay");
10. //Traversing elements
11. Iterator<String> itr=al.iterator();
12. while(itr.hasNext()){
13. System.out.println(itr.next());
14. }
15. }
16. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection11)

Output:

Ajay

Ravi

Vijay

# **Java Map Interface**

A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys.

### **Useful methods of Map interface**

|  |  |
| --- | --- |
| **Method** | **Description** |
| Object put(Object key, Object value) | It is used to insert an entry in this map. |
| void putAll(Map map) | It is used to insert the specified map in this map. |
| Object remove(Object key) | It is used to delete an entry for the specified key. |
| Object get(Object key) | It is used to return the value for the specified key. |
| boolean containsKey(Object key) | It is used to search the specified key from this map. |
| Set keySet() | It is used to return the Set view containing all the keys. |
| Set entrySet() | It is used to return the Set view containing all the keys and values. |

### Methods of Map.Entry interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| Object getKey() | It is used to obtain key. |
| Object getValue() | It is used to obtain value. |

### **Java Map Example: Generic (New Style)**

1. import java.util.\*;
2. class MapInterfaceExample{
3. public static void main(String args[]){
4. Map<Integer,String> map=new HashMap<Integer,String>();
5. map.put(100,"Amit");
6. map.put(101,"Vijay");
7. map.put(102,"Rahul");
8. for(Map.Entry m:map.entrySet()){
9. System.out.println(m.getKey()+" "+m.getValue());
10. }
11. }
12. }

Output:

102 Rahul

100 Amit

101 Vijay

### **Difference between HashSet and HashMap**

HashSet contains only values whereas HashMap contains entry(key and value).

# **Java LinkedHashMap class**

Java LinkedHashMap class is Hash table and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

The important points about Java LinkedHashMap class are:

* A LinkedHashMap contains values based on the key.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It is same as HashMap instead maintains insertion order.

### L**inkedHashMap class Parameters**

**L**et's see the Parameters for java.util.LinkedHashMap class.

* **K**: It is the type of keys maintained by this map.
* **V**: It is the type of mapped values.

### **Methods of Java LinkedHashMap class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| Object get(Object key) | It is used to return the value to which this map maps the specified key. |
| void clear() | It is used to remove all mappings from this map. |
| boolean containsKey(Object key) | It is used to return true if this map maps one or more keys to the specified value. |

### **Java LinkedHashMap Example**

1. import java.util.\*;
2. class TestCollection14{
3. public static void main(String args[]){
5. LinkedHashMap<Integer,String> hm=new LinkedHashMap<Integer,String>();
7. hm.put(100,"Amit");
8. hm.put(101,"Vijay");
9. hm.put(102,"Rahul");
11. for(Map.Entry m:hm.entrySet()){
12. System.out.println(m.getKey()+" "+m.getValue());
13. }
14. }
15. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection14)

Output:100 Amit

101 Vijay

102 Rahul

# **Java TreeMap class**

Java TreeMap class implements the Map interface by using a tree. It provides an efficient means of storing key/value pairs in sorted order.

The important points about Java TreeMap class are:

* A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* It contains only unique elements.
* It cannot have null key but can have multiple null values.
* It is same as HashMap instead maintains ascending order.

### **Methods of Java TreeMap class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean containsKey(Object key) | It is used to return true if this map contains a mapping for the specified key. |
| boolean containsValue(Object value) | It is used to return true if this map maps one or more keys to the specified value. |
| Object firstKey() | It is used to return the first (lowest) key currently in this sorted map. |
| Object get(Object key) | It is used to return the value to which this map maps the specified key. |
| Object lastKey() | It is used to return the last (highest) key currently in this sorted map. |
| Object remove(Object key) | It is used to remove the mapping for this key from this TreeMap if present. |
| void putAll(Map map) | It is used to copy all of the mappings from the specified map to this map. |
| Set entrySet() | It is used to return a set view of the mappings contained in this map. |
| int size() | It is used to return the number of key-value mappings in this map. |
| Collection values() | It is used to return a collection view of the values contained in this map. |

### **Java TreeMap Example:**

1. import java.util.\*;
2. class TestCollection15{
3. public static void main(String args[]){
4. TreeMap<Integer,String> hm=new TreeMap<Integer,String>();
5. hm.put(100,"Amit");
6. hm.put(102,"Ravi");
7. hm.put(101,"Vijay");
8. hm.put(103,"Rahul");
9. for(Map.Entry m:hm.entrySet()){
10. System.out.println(m.getKey()+" "+m.getValue());
11. }
12. }
13. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection15)

Output:100 Amit

101 Vijay

102 Ravi

103 Rahul

### **What is difference between HashMap and TreeMap?**

|  |  |
| --- | --- |
| **HashMap** | **TreeMap** |
| 1) HashMap can contain one null key. | TreeMap can not 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.

The important points about Java Hashtable class are:

* A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
* It contains only unique elements.
* It may have not have any null key or value.
* It is synchronized.

### **Java Hashtable Example**

1. import java.util.\*;
2. class TestCollection16{
3. public static void main(String args[]){
4. Hashtable<Integer,String> hm=new Hashtable<Integer,String>();
6. hm.put(100,"Amit");
7. hm.put(102,"Ravi");
8. hm.put(101,"Vijay");
9. hm.put(103,"Rahul");
11. for(Map.Entry m:hm.entrySet()){
12. System.out.println(m.getKey()+" "+m.getValue());
13. }
14. }
15. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection16)

Output:

103 Rahul

102 Ravi

101 Vijay

100 Amit

# **Difference between HashMap and Hashtable**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **HashMap** | **Hashtable** | | 1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is **synchronized**. It is thread-safe and can be shared with many threads. | | 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. | | 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. | | 4) HashMap is **fast**. | Hashtable is **slow**. | | 5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap); | Hashtable is internally synchronized and can't be unsynchronized. | | 6) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. | | 7) Iterator in HashMap is **fail-fast**. | Enumerator in Hashtable is **not fail-fast**. | | 8) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. | |