**Java collection general question**

**Q1)Collection :** a collection is an object that represents a group of objects. A collection (also called as container) is an object that groups multiple elements into a single unit.  
**Collections Framework :** Collections framework provides unified/united/combined architecture for manipulating(using /deploying) and representing collections.  
**Benefits of Collections Framework :** 1. Improves program quality and speed, 2. Increases the chances of reusability of software. 3. Decreases programming effort(energy).

* **Benefits of java collections**

Reduced programming effort due to ready to use code

Increased performance because of high-performance implementations of data structures and algorithms.

Easy to learn APIs by learning only some top level interfaces and supported operations

**Q 2) What is the root interface in collection hierarchy ?**

* The root interface of Collection hierarchy is Collection interface.

But the Collection interface extends Iterable interface. Due to this some people consider Iterable interface as the root interface.

Iterable interface is present in java.lang package but Collection interface is present in java.util package. Oracle Java API docs mention that Collection interface is a member of the Java Collections framework.

Whereas, Iterable interface is not stated as a part of Java Collections framework in Java docs.

Due to this Collection interface is the root of Collections Framework.

**Q3) What is the difference between Collection and Collections ? Which collection classes are synchronized or thread-safe ?**

* **Collection** is  an interface while **Collections** is a java class , both are present in java.util package and  part of java collections framework.
* Stack, Properties , Vector and Hash table can be used in multi threaded environment because they are synchronized classes (or thread-safe).

**Q4) Why Map interface does not extend Collection interface?**

A good answer to this interview question is “because they are incompatible“. Collection has a method add(Object o). Map can not have such method because it need key-value pair. There are other reasons also such as Map supports keyset, value Set etc. Collection classes does not have such views.

Due to such big differences, Collection interface was not used in Map interface, and it was build in separate hierarchy.

**Q5) Why Collection interface does not extend Cloneable and Serializable interface?**

* The Collection interface specifies groups of objects known as elements. Each concrete implementation of a collection can choose its own way of how to maintain and order its elements.
* The semantics and the Implementations of either cloning or serialization come into play when dealing with actual implementations.
* Thus, the concrete implementations of collections should decide how they can be cloned or serialized.

**6) How to reverse the list?**

This question is just like above to test your knowledge of Collections utility class. Use it reverse()method to reverse the list.

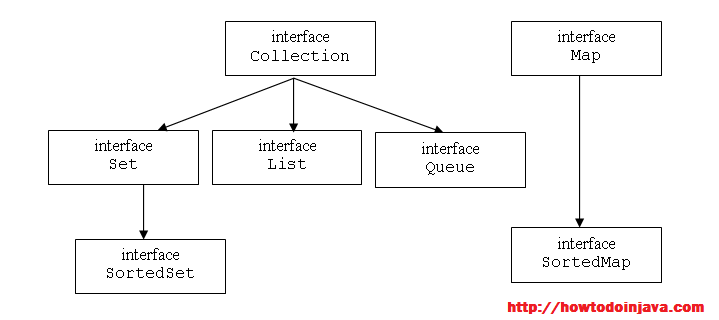
|  |
| --- |
| Collections.reverse(list); |
|  |

**Q7. Explain Collection’s hierarchy?**

* Java **Collection Hierarchy**
* As shown in above image, collection framework has one interface at top i.e. **Collection**. It is **extended by Set, List and Queue interfaces**. Then there are loads of other classes in these 3 branches which we will learn in following questions.
* Remember the signature of Collection interface. It will help you in many question.

|  |
| --- |
| * public interface Collection extends Iterable {//method definitions} |

* Framework also consist of Map interface, which is part of collection framework. but it does not extend Collection interface. We will see the reason in 4th question in this question bank.



**Java collection – List interface questions**

**Q 1) Why we use List interface? What are main classes implementing List interface?**

* A java list is a **“ordered” collection of elements**. This ordering is a **zero based index**. It does not care about duplicates.
* Apart from methods defined in Collection interface, it does **have its own methods** also which are largely to manipulate the collection **based on index location of element**. These methods can be grouped as search, get, iteration and range view. All above operations support index locations.
* The main classes implementing List interface are: **Stack, Vector, ArrayList and LinkedList**.

**Q2)Benefit of List interfaces**

* A [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html) is an ordered [Collection](https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html) (sometimes called a sequence). Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes operations for the following:
* Positional access —This includes methods such as get, set, add, addAll, and remove.
* Search —It methods include indexOf and lastIndexOf.
* Iteration — extends Iterator semantics to take advantage of the list's sequential nature. The listIterator methods provide this behavior.
* Range-view — The sublist method performs arbitrary range operations on the list.

**Q 3) How to convert an array of String to array list?**

There are two utility classes in Collection framework which are--**Collections and Arrays.**

* Collections class provides some static functions to perform specific operations on collection types. And Arrays provide utility functions to be performed on array types.

|  |
| --- |
| * //String array * String[] words = {"ace", "boom", "crew", "dog", "eon"}; * //Use Arrays utility class * List wordList = Arrays.asList(words); * //Now you can iterate over the list |

* Please not that this function is not specific to String class, it will return List of element of any type, of which the array is. e.g.

|  |
| --- |
| * //String array * Integer[] nums = {1,2,3,4}; * //Use Arrays utility class * List numsList = Arrays.asList(nums); |

**Java collection – Set interface questions**

**Q1 Why we use Set interface? What are main classes implementing Set interface?**

Set interface is like List interface but with some differences. First, it is not ordered collection. So no ordering is preserved while adding or removing elements. The main feature it does provide is “uniqueness of elements“. It does not support duplicate elements.

* Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ. Two Set instances are equal if they contain the same elements.
* Based on above reasons, it **does not have operations based on indexes of elements like List**. It only has methods which are inherited by Collection interface.
* Main classes implementing Set interface are :**EnumSet, HashSet, LinkedHashSet, TreeSet**. Read more on related java documentation.

**Q2) How HashSet store elements?**

* You must know that HashMap store key-value pairs, with one condition i.e. keys will be unique. HashSet uses Map’s this feature to ensure uniqueness of elements. In HashSet class, a map declaration is as below:

|  |
| --- |
| * private transient HashMap<E,Object> map; * //This is added as value for each key * private static final Object PRESENT = new Object(); |

* So **when you store a element in HashSet, it stores the element as key in map and “PRESENT” object as value**. (See declaration above).

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

* I will highly suggest you to read this post: [**How HashMap works in java?**](https://howtodoinjava.com/java/collections/how-hashmap-works-in-java/) This post will help you in answering all the HashMap related questions very easily.

**Q3) Can a null element added to a TreeSet or HashSet?**

* HashMap allows one null key, so **one “null” is allowed in HashSet**.
* TreeSet uses the same concept as HashSet for internal logic, but uses Navigable Map for storing the elements.

|  |
| --- |
| * private transient Navigable Map<E,Object> m; * // Dummy value to associate with an Object in the backing Map * private static final Object PRESENT = new Object(); |

* NavigableMap is subtype of SortedMap which does not allow null keys. So essentially,**TreeSet also does not support null keys**. It will throw NullPointerException if you try to add null element in TreeSet.

**Java collection – Map interface questions**

**Q1) Why we use Map interface? What are main classes implementing Map interface?**

* Map interface is a special type of collection which is **used to store key-value pairs**. It does not extend Collection interface for this reason. This interface provides methods to add, remove, search or iterate over various views of Map.
* Main classes implementing Map interface are:**HashMap, Hashtable, EnumMap, IdentityHashMap, LinkedHashMap and Properties.**

**Q2) What are IdentityHashMap and WeakHashMap?**

* **IdentityHashMap** is a special implementation of Map interface much like EnumMap but it also violates general contract of Map interface which mandates using equals method for comparing Object.

IdentityHashMap is designed for use only in the rare cases wherein reference-equality semantics are required.

* **WeakHashMap** is an implementation of the Map interface that stores only weak references to its keys. Storing only weak references allows a key-value pair to be garbage-collected when its key is no longer referenced outside of the**WeakHashMap**.
* **The main difference is that** IdentityHashMap uses == operator when comparing keys and values, instead of equals method used by W**eakHashMap**, or any other Map implementation for that matter.Also **IdentityHashMap** uses System.identityHashCode(object) instead of hashCode() method of other Map implementations.

**Q3) Explain ConcurrentHashMap? How it works?**

Taking from java docs:

A hash table supporting full concurrency of retrievals and adjustable expected concurrency for updates. This class obeys the same functional specification as Hashtable, and includes versions of methods corresponding to each method of Hashtable.

However, even though all operations are thread-safe, retrieval operations do not entail locking, and there is not any support for locking the entire table in a way that prevents all access. This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details.

**Q4) How hashmap works?**

* The **most important question** which is most likely to be seen in every level of job interviews. You must be very clear on this topic., not only because it is most asked question but also it will open up your mind in further questions related to collection APIs.
* Answer to this question is very large and you should read it my post: [**How HashMap works?**](https://howtodoinjava.com/java/collections/how-hashmap-works-in-java/) For now, lets remember that HashMap works **on principle of Hashing**. A map by definition is : “An object that maps keys to values”. To store such structure, **it uses an inner class Entry**:

|  |
| --- |
| static class Entry implements Map.Entry  {  final K key;  V value;  Entry next;  final int hash;  ...//More code goes here} |

Here key and value variables are used to store key-value pairs. Whole entry object is stored in an array.

|  |
| --- |
| /\*\*  The table, re-sized as necessary. Length MUST Always be a power of two.  \*/  transient Entry[] table; |

The index of array is calculated on basis on hashcode of Key object. Read more of linked topic.

**Q5) How to design a good key for hashmap?**

* Another good question usually followed up after answering how hashmap works. Well, the most important constraint is **you must be able to fetch the value object back in future**. Otherwise, there is no use of having such a data structure. If you understand the working of hashmap, you will find it largely depends on hashCode() and equals() method of Key objects.
* So a good key object**must provide same hashCode() again and again**, no matter how many times it is fetched. Similarly, same keys**must return true when compare with equals() method and different keys must return false**.
* For this reason,**immutable classes are considered best candidate for HashMap keys**.
* Read more : [**How to design a good key for HashMap?**](https://howtodoinjava.com/java/collections/how-to-design-a-good-key-for-hashmap/)

**Q6) What are different Collection views provided by Map interface?**

* Map interface provides 3 views of key-values pairs stored in it:
* key set view
* value set view
* entry set view
* All the views can be navigated using iterators.

Q**7) When to use HashMap or TreeMap?**

* HashMap is well known class and all of us know that. So, I will leave this part by saying that it is used to store key-value pairs and allows to perform many operations on such collection of pairs.
* TreeMap is special form of HashMap. **It maintains the ordering of keys** which is missing in HashMap class. This ordering is **by default “natural ordering”**. The default ordering can be override by providing an instance of Comparator class, whose compare method will be used to maintain ordering of keys.
* Please note that **all keys inserted into the map must implement the Comparable interface** (this is necessary to decide the ordering). Furthermore, all such keys must be mutually comparable: k1.compareTo(k2) must not throw a ClassCastException for any keys k1 and k2 in the map. If the user attempts to put a key into the map that violates this constraint (for example, the user attempts to put a string key into a map whose keys are integers), the put(Object key, Object value) call will throw a ClassCastException.

**Java collection – the difference questions**

Q1**) Difference between Set and List?**

The most noticeable differences are :

* Set is unordered collection where List is ordered collection based on zero based index.
* List allow duplicate elements but Set does not allow duplicates.
* List does not prevent inserting null elements (as many you like), but Set will allow only one null element.

**Q2) Difference between List and Map?**

List is collection of elements where as map is collection of key-value pairs.

**Q3) Difference between HashMap and HashTable?**

* Hashtable is synchronized, whereas HashMap is not.
* Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
* The third 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.

**Q4) Difference between Vector and ArrayList?**

* All the methods of Vector is synchronized. But, the methods of ArrayList is not synchronized.
* Vector is a Legacy class added in first release of JDK. ArrayList was part of JDK 1.2, when collection framework was introduced in java.
* By default, Vector doubles the size of its array when it is re-sized internally. But, ArrayList increases by half of its size when it is re-sized.

**Q5)Difference between Iterator and Enumeration?**

Iterators allow the caller to remove elements from the underlying collection during the iteration with its remove() method. You can not add/remove elements from a collection when using enumerator.

* Enumeration is available in legacy classes i.e Vector/Stack etc. whereas Iterator is available in all modern collection classes.
* Another minor difference is that Iterator has improved method names e.g. Enumeration.hasMoreElement() has become Iterator.hasNext(), Enumeration.nextElement() has become Iterator.next() etc.

**Q 6) Difference between HashMap and HashSet?**

HashMap is collection of key-value pairs whereas HashSet is un-ordered collection of unique elements. That’s it. No need to describe further.

**Q7) Difference between Iterator and ListIterator?**

We can use Iterator to traverse Set and List and also Map type of Objects. But List Iterator can be used to traverse for List type Objects, but not for Set type of Objects.

* By using Iterator we can retrieve the elements from Collection Object in forward direction only whereas List Iterator, which allows you to traverse in either directions using hasPrevious() and previous() methods.
* List Iterator allows you modify the list using add() remove() methods. Using Iterator you can not add, only remove the elements.

**Q8) Difference between Tree Set and Sorted Set?**

Sorted Set is an interface which TreeSet implements. That’ it !!

**Q9) Difference between Array List and LinkedList?**

LinkedList store elements within a doubly-linked list data structure. Array List store elements within a dynamically resizing array.

* LinkedList allows for constant-time insertions or removals, but only sequential access of elements. In other words, you can walk the list forwards or backwards, but grabbing an element in the middle takes time proportional to the size of the list. ArrayLists, on the other hand, allow random access, so you can grab any element in constant time. But adding or removing from anywhere but the end requires shifting all the latter elements over, either to make an opening or fill the gap.
* 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.

**More collection interview**

**1) How to make a collection read only?**

Use following methods:

* Collections.unmodifiableList(list);
* Collections.unmodifiableSet(set);
* Collections.unmodifiableMap(map);
* These methods takes collection parameter and return a new read-only collection with same elements as in original collection.

**2) How to make a collection thread safe?**

Use below methods:

Collections.synchronizedList(list);

* Collections.synchronizedSet(set);
* Collections.synchronizedMap(map);
* Above methods take collection as parameter and return same type of collection which are synchronized and thread safe.

**3) Why there is not method like Iterator.add() to add elements to the collection?**

The sole purpose of an Iterator is to enumerate through a collection. All collections contain the add() method to serve your purpose. There would be no point in adding to an Iterator because the collection may or may not be ordered. And add() method can not have same implementation for ordered and unordered collections.

**4) What are different ways to iterate over a list?**

You can iterate over a list using following ways:

* Iterator loop
* For loop
* For loop (Advance)
* While loop

**5) What do you understand by iterator fail-fast property?**

Fail-fast Iterators fail as soon as they realized that structure of Collection has been changed since iteration has begun. Structural changes means adding, removing or updating any element from collection while one thread is Iterating over that collection.

Fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws ConcurrentModificationException.

**6) What is difference between fail-fast and fail-safe?**

Fail-safe iterators are just opposite to fail-fast. They never fail if you modify the underlying collection on which they are iterating, because they work on clone of Collection instead of original collection and that’s why they are called as fail-safe iterator

* Iterator of Copy On Write Array List is an example of fail-safe Iterator also iterator written by Concurrent HashMap keySet is also fail-safe iterator and never throw Concurrent Modification Exception.

**7) How to avoid Concurrent Modification Exception while iterating a collection?**

* You should first try to **find another alternative iterator which are fail-safe**. For example if you are using List and you can use ListIterator. If it is legacy collection, you can use enumeration.
* If above options are not possible then you can use one of three changes:
* If you are using JDK1.5 or higher then you can use ConcurrentHashMap and CopyOnWriteArrayList classes. It is the recommended approach.
* You can convert the list to an array and then iterate on the array.
* You can lock the list while iterating by putting it in a synchronized block.
* Please note that last two approaches will cause a performance hit.

**8) What is UnsupportedOperationException?**

* This exception is thrown **on invoked methods which are not supported by actual collection type**.
* For example, if you make a read-only list list using “Collections.unmodifiableList(list)” and then call add() or remove() method, what should happen. It should clearly throw UnsupportedOperationException.

**9) Which collection classes provide random access of it’s elements?**

ArrayList, HashMap, TreeMap, Hashtable classes provide random access to it’s elements.

**10)What is BlockingQueue?**

* **A Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.**
* BlockingQueue methods come in four forms: one throws an exception, the second returns a special value (either null or false, depending on the operation), the third blocks the current thread indefinitely until the operation can succeed, and the fourth blocks for only a given maximum time limit before giving up.

**11) What is Queue and Stack, list down their differences?**

A collection designed for holding elements prior to processing. Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations.  
Queues typically, but do not necessarily, order elements in a FIFO (first-in-first-out) manner.

* **Stack is also a form of Queue but one difference, it is LIFO (last-in-first-out).**
* Whatever the ordering used, the head of the queue is that element which would be removed by a call to remove() or poll(). Also note that Stack and Vector are both synchronized.
* **Usage:** Use a queue if you want to process a stream of incoming items in the order that they are received.Good for work lists and handling requests.  
  Use a stack if you want to push and pop from the top of the stack only. Good for recursive algorithms.

**12) What is Comparable and Comparator interface?**

In java. all collection which have feature of automatic sorting, uses compare methods to ensure the correct sorting of elements. For example classes which use sorting are TreeSet, TreeMap etc.

* **To sort the data elements a class needs to implement Comparator or Comparable interface**. That’s why all Wrapper classes like Integer,Double and String class implements Comparable interface.
* **Comparable helps in preserving default natural sorting, whereas Comparator helps in sorting the elements in some special required sorting pattern.** The instance of comparator if passed usually as collection’s constructor argument in supporting collections.

**13) What are Collections and Arrays classes?**

* **Collections and Arrays classes are special utility classes to support collection framework core classes.** They provide utility functions to get read-only/ synchronized collections, sort the collection on various ways etc.
* Arrays also helps array of objects to convert in collection objects. Arrays also have some functions which helps in copying or working in part of array objects.

# **Java - Data Structures**

* The data structures provided by the Java utility package are very powerful and perform a wide range of functions. These data structures consist of the following interface and classes −
* **1)Enumeration,2)BitSet,3)Vector,4) Stack,5)Dictionary,6)Hashtable,7)Properties**
* **The Enumeration**
* The Enumeration interface isn't itself a data structure, but it is very important within the context of other data structures. The Enumeration interface defines a means to retrieve successive elements from a data structure.
* **For example,** Enumeration defines a method called next Element that is used to get the next element in a data structure that contains multiple elements.
* **The BitSet**
* The BitSet class implements a group of bits or flags that can be set and cleared individually.
* **The Vector**
* The Vector class is similar to a traditional Java array, except that it can grow as necessary to accommodate new elements.
* Like an array, elements of a Vector object can be accessed via an index into the vector.
* **The Stack**
* The Stack class implements a last-in-first-out (LIFO) stack of elements.
* **The Dictionary**
* The Dictionary class is an abstract class that defines a data structure for mapping keys to values.
* **The Hashtable**
* The Hashtable class provides a means of organizing data based on some user-defined key structure.
* **The Properties**
* Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String.
* For example, it is the type of object returned by System.getProperties( ) when obtaining environmental values.

# **Java - Collections Framework**

* The collections framework was designed to meet several goals, such as −
* The framework had to be high-performance. The implementations for the fundamental collections (dynamic arrays, linked lists, trees, and hashtables) were to be highly efficient.
* The framework had to allow different types of collections to work in a similar manner and with a high degree of interoperability.
* The framework had to extend and/or adapt a collection easily.
* Towards this end, the entire collections framework is designed around a set of standard interfaces. Several standard implementations such as **LinkedList, HashSet,** and **TreeSet**, of these interfaces are provided that you may use as-is and you may also implement your own collection, if you choose.
* A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following −
* **Interfaces** − These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
* **Implementations, i.e., Classes** − These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
* **Algorithms** − These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface.

## The Collection Interfaces

* The collections framework defines several interfaces. This section provides an overview of each interface −
* [**The Collection Interface**](https://www.tutorialspoint.com/java/java_collection_interface.htm)
* This is enables to work with groups of objects; it is at the top of the collections hierarchy.
* [**The List Interface**](https://www.tutorialspoint.com/java/java_list_interface.htm)
* This extends **Collection** and an instance of List stores an ordered collection of elements.
* [**The Set**](https://www.tutorialspoint.com/java/java_set_interface.htm)
* This extends Collection to handle sets, which must contain unique elements.
* [**The SortedSet**](https://www.tutorialspoint.com/java/java_sortedset_interface.htm)
* This extends Set to handle sorted sets.
* [**The Map**](https://www.tutorialspoint.com/java/java_map_interface.htm)
* This maps unique keys to values.
* [**The Map.Entry**](https://www.tutorialspoint.com/java/java_mapentry_interface.htm)
* This describes an element (a key/value pair) in a map. This is an inner class of Map.
* [**The SortedMap**](https://www.tutorialspoint.com/java/java_sortedmap_interface.htm)
* This extends Map so that the keys are maintained in an ascending order.
* [**The Enumeration**](https://www.tutorialspoint.com/java/java_enumeration_interface.htm)
* This is legacy interface defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects. This legacy interface has been superceded by Iterator.

## The Collection Classes

* Java provides a set of standard collection classes that implement Collection interfaces.
* The standard collection classes are summarized in the following table −
* **AbstractCollection**
* Implements most of the Collection interface.
* **AbstractList**
* Extends AbstractCollection and implements most of the List interface.
* **AbstractSequentialList**
* Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.
* [LinkedList](https://www.tutorialspoint.com/java/java_linkedlist_class.htm)
* Implements a linked list by extending AbstractSequentialList
* [ArrayList](https://www.tutorialspoint.com/java/java_arraylist_class.htm)
* Implements a dynamic array by extending AbstractList.
* **AbstractSet**
* Extends AbstractCollection and implements most of the Set interface.
* [**HashSet**](https://www.tutorialspoint.com/java/java_hashset_class.htm)
* Extends AbstractSet for use with a hash table
* [**LinkedHashSet**](https://www.tutorialspoint.com/java/java_linkedhashset_class.htm)
* Extends HashSet to allow insertion-order iterations.
* [**TreeSet**](https://www.tutorialspoint.com/java/java_treeset_class.htm)
* Implements a set stored in a tree. Extends AbstractSet.
* **AbstractMap**
* Implements most of the Map interface.
* [**HashMap**](https://www.tutorialspoint.com/java/java_hashmap_class.htm)
* Extends AbstractMap to use a hash table.
* [**TreeMap**](https://www.tutorialspoint.com/java/java_treemap_class.htm)
* Extends AbstractMap to use a tree.
* [**WeakHashMap**](https://www.tutorialspoint.com/java/java_weakhashmap_class.htm)

Extends AbstractMap to use a hash table with weak keys.

* [**LinkedHashMap**](https://www.tutorialspoint.com/java/java_linkedhashmap_class.htm)

Extends HashMap to allow insertion-order iterations.

[**IdentityHashMap**](https://www.tutorialspoint.com/java/java_identityhashmap_class.htm)

Extends AbstractMap and uses reference equality when comparing documents.