COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50

*You are here : **Home** / Core Java Tutorials / Collection framework Tutorial in java / Java Interview Questions and answers*

This is set of most complete collection interview questions and answers consisting of 100 questions. It's time to impress interviewer, crack collection interview questions in java. Best set of collection interview questions, I have tried to cover almost all the possible collection questions which could be framed in an interview by interviewer.

Collection interview Question 1. What is Collection framework in java?

Answer. It's the basic Collection framework interview question. Freshers must know about this. ***java.util.Collection*** *is the* root interface in the *hierarchy of Java Collection framework in java*.

The JDK does not provide any classes which directly implements this interface, but it provides classes which are implementations of more specific subinterfaces like Set and List in java.

java.util.Set extends java.util.Collection interface in java.
HashSet, CopyOnWriteArraySet, LinkedHashSet, TreeSet, ConcurrentSkipListSet, EnumSet classes implements Set interface.

java.util.List extends java.util.Collection interface in java.
ArrayList, LinkedList, Vector, CopyOnWriteArrayList classes implements List interface.

Also read >

COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75

COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-3 > Q75- Q100

Collection interview Question 2. Which interfaces and classes are most frequently used in Collection framework in java?

Answer. This collection framework interview question will test your practical knowledge. Freshers may get away by answering few interface and classes but experienced developers must answer this question in detail.

Most frequently used interface in Collection framework are >
List, Set and Map.

Most frequently used classes in Collection framework are >
HashSet, LinkedHashSet, TreeSet, ConcurrentSkipListSet classes implements Set interface.

ArrayList, LinkedList, Vector, CopyOnWriteArrayList classes implements List interface.

HashMap, Hashtable, ConcurrentHashMap, LinkedHashMap, TreeMap, ConcurrentSkipListMap classes implements **Map** interface.

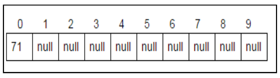
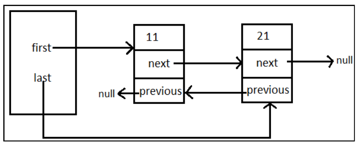
Collection interview Question 3. What are subinterfaces of Collection interface in java? Is Map interface also a subinterface of Collection interface in java?

Answer. List and Set are subinterfaces of java.util.Collection in java.

It's important to note Map interface is a member of the Java Collections Framework, but it does not implement Collection interface in java.

Collection interview Question 4. What are differences between ArrayList and LinkedList in java?

Answer. This is very important collection framework interview question in java.

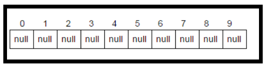
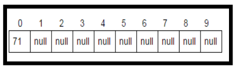
	Property	***java.util.ArrayList***	**java.util.LinkedList**
1	Structure	java.util.ArrayList is index based structure in java.  0 1 2 3 4 5 6 7 8 9 71 null null null null null null null null null	A java.util.**LinkedList** is a data structure consisting of a group of **nodes** which together represent a sequence. node is composed of a data and a reference (in other words, a **link**) to the next node in the sequence in java.  first last 11 next previous 21 next previous null null
2	**Resizable**	**ArrayList is Resizable-array in java.**	New node is created for storing new element in LinkedList in java.
3	**Initial capacity**	java.util.ArrayList is created with initial capacity of 10 in java.	For storing every element node is created in LinkedList, so linkedList's initial capacity is 0 in java.
4	Ensuring **Capacity**/ resizing.	ArrayList is created with initial capacity of 10. ArrayList's size is **increased by 50%** i.e. after resizing it's size become 15 in java.	For storing every element node is created, so linkedList's initial capacity is 0, it's size grow with addition of each and every element in java.
5	RandomAccess interface	ArrayList implements RandomAccess(Marker interface) to indicate that they support fast random access (i.e. index based access) in java.	LinkedList does not implement RandomAccess interface in java.
6	AbstractList and AbstractSequentialList	ArrayList extends AbstractList (abstract class) which provides implementation to List interface to minimize the effort required to implement this interface backed by RandomAccess interface.	LinkedList extends AbstractSequentialList (abstract class), AbstractSequentialList extends AbstractList. In LinkedList, data is accessed sequentially, so for obtaining data at specific index, iteration is done on nodes sequentially in java.
7	How **get(index)** method works? (Though difference has been discussed briefly in above 2	Get method of ArrayList directly gets element on specified index. Hence, offering O(1) complexity in java.	Get method of LinkedList iterates on nodes sequentially to get element on specified index. Hence, offering O(n) complexity in java.

	points but in this in point we will figure difference in detail.)		
8	**When to use**	**Use ArrayList when get operations is more frequent than add and remove operations in java.**	**Use LinkedList when add and remove operations are more frequent than get operations in java.**

For more detail like complexity comparison of method please read : **ArrayList vs LinkedList in java**

Collection interview Question 5. What are differences between ArrayList and Vector in java?

Answer. Another very important collection framework interview question to differentiate between ArrayList and Vector in java.

	Property	***java.util.ArrayList***	***java.util.Vector***
1	synchronization	java.util.ArrayList is **not synchronized** (because 2 threads on same ArrayList object can access it at same time). I have created **program** to show consequence of using ArrayList in multithreading environment. In the program we will implement our own arrayList in java.	java.util.Vector is **synchronized** (because 2 threads on same Vector object cannot access it at same time). I have created **program** to show advantage of using Vector in multithreading environment. In the program we will implement our own vector in java.
2	Performance	ArrayList is not synchronized, hence its operations are **faster** as compared to Vector in java.	Vector is synchronized, hence its operations are **slower** as compared to ArrayList in java. If we are working not working in multithreading environment jdk recommends us to use ArrayList.
3	Enumeration	**Enumeration** is **fail-fast**, means any modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException in java.	**Enumeration** is **fail-safe**, means any modification made to Vector during iteration using Enumeration don't throw any exception in java.
4	Introduced in which java version	ArrayList was introduced in second version of java i.e. **JDK 2.0**	Vector was introduced in first version of java i.e. **JDK 1.0** But it was refactored in java 2 i.e. JDK 1.2 to implement the List interface, hence making it a member of member of the Java Collections Framework.
5	Ensuring Capacity/ resizing.	ArrayList is created with initial capacity of 10. When its full size is **increased by 50%** i.e. after resizing it's size become 15 in java.	Vector is created with initial capacity of 10. Vector's size is **increased by 100%** i.e. after resizing it's size become 20 in java.
6	Custom implementation	 0 1 2 3 4 5 6 7 8 9 null null null null null null null null null null Read : ArrayList custom implementation	 0 1 2 3 4 5 6 7 8 9 71 null null null null null null null null null Read : Vector custom implementation

For more detail like complexity comparison of method please read: **ArrayList vs Vector- Similarity and Differences in java**

Collection interview Question 6. What are differences between List and Set interface in java?

Answer. Another very very important collection framework interview question to differentiate between **List and Set** in java.

	Property	***java.util.List***	***java.util.Set***
1	Insertion	java.util.List is ordered collection it	*Most of the java.util.Set implementation*

	order	**maintain insertion order** in java.	does not **maintain insertion order**. HashSet does not maintains insertion order in java. Thought LinkedHashSet maintains insertion order in java. TreeSet is sorted by natural order in java.
2	Duplicate elements	List **allows to store duplicate elements** in java.	*Set does **not allow to store duplicate elements** in java.*
3	Null keys	List allows to store **many null keys** in java.	Most of the Set implementations allow to add only **one null** in java**.** TreeSet does not allow to add null in java.
4	Getting element on specific **index**	List implementations provide get method to get element on specific index in java. ArrayList, Vector, copyOnWriteArrayList and LinkedList provides - *get(int index)* Method returns element on specified *index*. **Get method directly gets element on specified index. Hence, offering O(1) complexity.**	Set implementations does not provide any such get method to get element on specified index in java.
5	Implementing classes	**ArrayList**, **LinkedList**, **Vector**, **CopyOnWriteArrayList** classes implements **List** interface in java.	**HashSet**, **CopyOnWriteArraySet**, **LinkedHashSet**, **TreeSet**, **ConcurrentSkipListSet**, **EnumSet** classes implements **Set** interface in java.
6	listIterator	**listIterator** method returns listIterator to iterate over elements in List in java. **listIterator provides** additional methods as compared to iterator like **hasPrevious(), previous(), nextIndex(), previousIndex(), add(E element), set(E element)**	Set does not provide anything like listIterator. It simply return Iterator in java.
7	Structure and resizable	**List** are Resizable-array implementation of the java.util.**List** interface in java.	Set uses **Map** for their implementation. Hence, structure is map based and resizing depends on Map implementation. *Example > **HashSet** internally uses HashMap.*
8	Index based structure /RandomAccess	As **ArrayList** uses array for implementation it is index based structure, hence provides random access to elements. But **LinkedList** is not indexed based structure in java.	Set is not index based structure at all in java.

For more detail read : List vs Set - Similarity and Differences in java

Collection interview Question 7. What are differences between Iterator and ListIterator? in java

Answer. This collection framework interview question is tests your knowledge of iterating over different collection framework classes in java.

	java.util.ListIterator	*java.util.Iterator*
1	**hasPrevious()** method returns true if this listIterator has more elements when traversing the list in the reverse direction.	**No such method** in java.util.Iterator.

2	**previous()** returns previous element in iteration (traversing in backward direction). if the iteration has no previous elements than NoSuchElementException is thrown.	**No such method** in java.util.Iterator.
3	**nextIndex()** method returns the index of the element that would be returned by a subsequent call to next() method. If listIterator is at the end of the list than method returns size of list.	**No such method** in java.util.Iterator.
4	**previousIndex()** method returns the index of the element that would be returned by a subsequent call to previous() method. If listIterator is at the start of the list than method returns -1.	**No such method** in java.util.Iterator.
5	**add(E element)** Method inserts the specified **element** into the list. The element is inserted immediately before the element that would be returned by next (So, subsequent call to next would be unaffected), if any, and after the element that would be returned by previous (So,subsequent call to previous would return the new **element**), if any. If the list does not contain any element than new **element** will be the sole element in the list.	**No such method** in java.util.Iterator.
6	**set(E element)** Method replaces the last element returned by next() or previous() method with the specified **element**. This call can be made only if neither remove nor add have been called after the last call to next or previous. If call to set() method is followed up by any call made to remove() or add() method after next() or previous() than UnsupportedOperationException is thrown.	**No such method** in java.util.Iterator.
7	All the implementations of **List** interface like **ArrayList**, **LinkedList**, **Vector**, **CopyOnWriteArrayList** classes returns listIterator.	All Implementation classes of **Collection** interface's subinterfaces like Set and List return iterator.

For more detail read : **Iterator vs ListIterator - Similarity and Differences in java**

Collection interview Question 8. What are differences between Collection and Collections in java?

Answer. This is another very important collection framework interview question.In real projects you must have used both Collection and Collections but what is the difference between two of them in java?

java.util.***Collection*** *is the* root **interface** in the *hierarchy of Java Collection framework*.

The JDK does not provide any classes which directly implements java.util.Collection interface, but it provides classes such as **ArrayList**, **LinkedList**, **vector**, **HashSet**, **EnumSet**, **LinkedHashSet**, **TreeSet**, CopyOnWriteArrayList, CopyOnWriteArraySet, ConcurrentSkipListSet which implements more specific subinterfaces like Set and List in java.

java.util.**Collections** is a utility **class** which **consists** of **static methods** that **operate on** or return **Collection** in java.

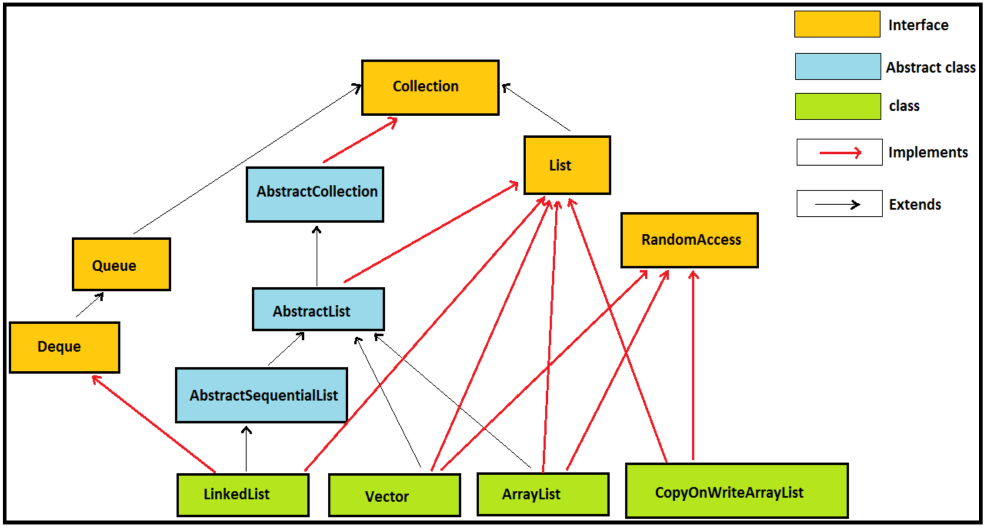
java.util.Collections provides method like >

- **reverse** method for reversing **List** in java.
- **shuffle** method for shuffling elements of **List** in java.
- **unmodifiableCollection**, **unmodifiableSet**, **unmodifiableList**, **unmodifiableMap** methods for making **List**, **Set** and **Map** unmodifiable in java.
- **min** method to return smallest element in **Collection** in java.
- **max** method to return smallest element in **Collection**.
- **sort** method for sorting **List**.
- **synchronizedCollection**, **synchronizedSet**, **synchronizedList**, **synchronizedMap** methods for synchronizing **List**, **Set** and **Map** respectively in java.

Additionally you must know that *java.util.Collection and java.util.Collections both were introduced in* ***second version of java i.e. in JDK 2.0.***

Collection interview Question 9. What are core classes and interfaces in java.util.List hierarchy in java?

Answer. Freshers must know core classes in List hierarchy but experienced developers must be able to explain this java.util.List hierarchy in detail.



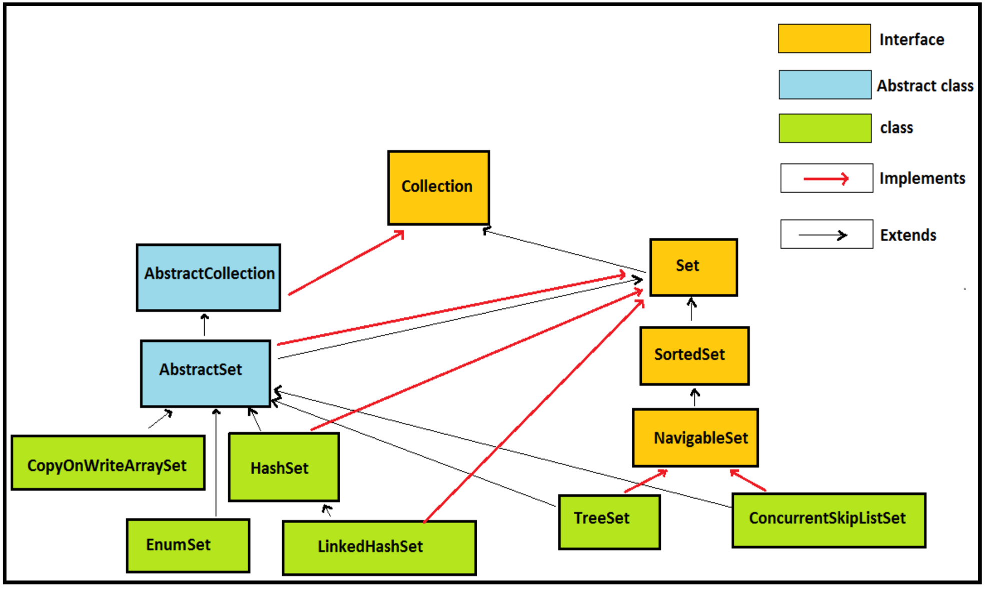
java.util.List interface extends java.util.Collection interface.

java.util.ArrayList, ***java.util.*LinkedList, *java.util.*Vector,** ***java.util.concurrent.*CopyOnWriteArrayList** classes implements java.util.List interface.

Also some abstract classes like java.util.**AbstractCollection**, java.util.**AbstractList** and java.util.**AbstractSequentialList** have been mentioned in hierarchy.

Collection interview Question 10. What are core classes and interfaces in java.util.Set hierarchy?

Answer. Freshers must know core classes in Set hierarchy but experienced developers must be able to explain this java.util.Set hierarchy in detail.



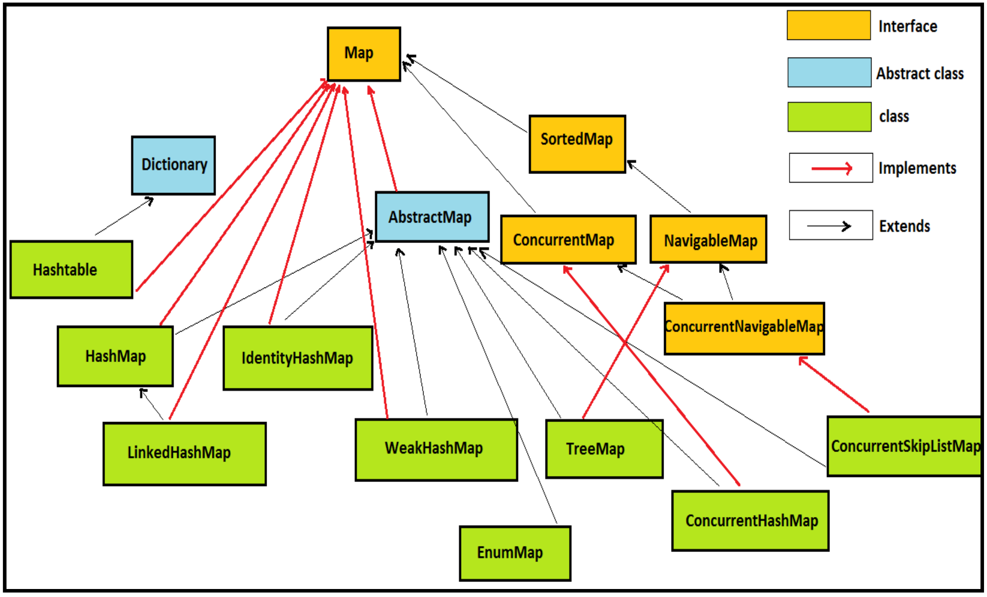
java.util.Set interface extends java.util.Collection interface.

java.util.HashSet, ***java.util.concurrent.*CopyOnWriteArraySet,** ***java.util.*LinkedHashSet, *java.util.*TreeSet**, java.util.concurrent.ConcurrentSkipListSet, java.util.EnumSet classes implements java.util.Set interface.

Also some abstract classes like java.util.**Dictionary** and java.util.**AbstractSet** and java.util.**AbstractCollection** have been mentioned in hierarchy.

Collection interview Question 11. What are core classes and interfaces in java.util.Map hierarchy?

Answer. Freshers must know core classes in Map hierarchy but experienced developers must be able to explain this java.util.Map hierarchy in detail.



java.util.Map interface extends java.util.Collection interface.

java.util.HashMap, ***java.util.*Hashtable*, java.util.concurrent.*ConcurrentHashMap*,***
***java.util.*LinkedHashMap**, java.util.TreeMap, java.util.concurrent.ConcurrentSkipListMap, java.util.IdentityHashMap, java.util.WeakHashMap, java.util.EnumMap classes implements java.util.**Map** interface.

Also some abstract classes like java.util.**Dictionary** and java.util.**AbstractMap** have been mentioned in hierarchy.

Collection interview Question 12. What are differences between Iterator and Enumeration in java?

Answer. Experienced developers must be well versed to answer this collection framework interview question in java.

***Differences** between java.util.**Iterator** and java.util.**Enumeration** in java >*

	Property	*java.util.Enumeration*	*java.util.Iterator*
1	Remove elements during iteration	java.util.Enumeration **doesn't allows** to remove elements from collection during iteration in java.	java.util.Iterator **allows** to remove elements from collection during iteration by using **remove()** method in java.

2	Improved naming conventions in Iterator	**nextElement()**
		Method Returns the next element of this enumeration if this enumeration object has at least one more element to provide.
		hasMoreElements()
		returns true if enumeration contains more elements.
			nextElement() has been changed to **next()** in Iterator.
			And
			hasMoreElements() has been changed to **hasNext()** in Iterator.

3	Introduced in which java version	Enumeration was introduced in first version of java i.e. **JDK 1.0**	Iterator was introduced in second version of java i.e. **JDK 2.0** Iterator was introduced to replace Enumeration in the Java Collections Framework.
4	Recommendation	**Java docs** recommends iterator over enumeration.	**Java docs** recommends iterator over enumeration.

5	Enumeration and Iterator over Vector	**Enumeration** returned by Vector is fail-safe, means any modification made to Vector during iteration using Enumeration don't throw any exception in java.	**Iterator** returned by Vector are fail-fast, means any structural modification made to ArrayList during iteration will throw ConcurrentModificationException in java.

For more detail read : Iterator vs Enumeration - Differences and similarities in java

Collection interview Question 13. How do we override equals and hashcode method in java, write a code to use Employee as key in HashMap in java? (Important)

Answer. This is one of the most important collection framework interview question in java. Prepare for this question properly. Freshers must know the concept how to override equals and hashcode method but experienced developers must be able to write the java code to override equals and hashcode neatly. We will override equals() and hashCode() like this -

By overriding equals() and hashCode() method we could use custom object as key in HashMap.

1) Check whether obj is null or not.

```
   if(obj==null) //If obj is null, return without comparing obj & Employee class.
```

2) check whether obj is instance of Employee class or not.

```
 if(this.getClass()!=obj.getClass()) //identifies whether obj is instance of
Employee class or not.
```

3) Then, type cast obj into employee instance.

```
  Employee emp=(Employee)obj;  //type cast obj into employee instance.
```

```
    @Override
    public boolean equals(Object obj){

           if(obj==null)
                  return false;

           if(this.getClass()!=obj.getClass())
                  return false;

           Employee emp=(Employee)obj;
           return (emp.id==this.id || emp.id.equals(this.id))
                               && (emp.name==this.name ||
emp.name.equals(this.name));
    }

    @Override
    public int hashCode(){
           int hash=(this.id==null ? 0: this.id.hashCode() ) +
                        (this.name==null ? 0: this.name.hashCode() );
           return hash;
    }
```

Let's say in an organisation there exists a employee with **id=1 and name='sam'** and **some data** is stored corresponding to him, but if modifications have to be made in data, **previous data must be overridden**.

DETAILED DESCRIPTION : Override equals() and hashCode() method.

Must read : Overriding equals and hashcode method - Top 18 Interview questions in java

Collection interview Question 14. What classes should i prefer to use a key in HashMap in java? (Important)

Answer. This collection framework interview question will check your in depth knowledge of Java’s Collection Api’s. we should prefer **String, Integer, Long, Double, Float, Short and any other wrapper class.** Reason behind using them as a key is that they override equals() and hashCode() method, we need not to write any explicit code for overriding equals() and hashCode() method in java.

Let’s use Integer class as key in HashMap(Example) -

```
import java.util.HashMap;
import java.util.Map;

public class StringInMapExample {
    public static void main(String...a){

          //HashMap's key=Integer class  (Integer’s api has already overridden
hashCode() and equals() method for us )
           Map<Integer, String> hm=new HashMap<Integer, String>();
           hm.put(1, "data");
           hm.put(1, "data OVERRIDDEN");

           System.out.println(hm.get(1));

    }
}
/*OUTPUT

data OVERRIDDEN

*/
```

If, we note above program, what we will see is we didn’t override equals() and hashCode() method, but still we were able to store data in HashMap, override data and retrieve data using get method.

>Let’s check in **Integer’s API**, how Integer class has overridden equals() and hashCode() method :

```
     public int hashCode() {
      return value;
  }

     public boolean equals(Object obj) {
      if (obj instanceof Integer) {
       return value == ((Integer)obj).intValue();
      }
      return false;
  }
```

Collection interview Question 15. What are differences between HashMap and Hashtable in java?

Answer. Fresher and Experienced developers must answer this important collection framework interview question in detail in java.

***Differences** between java.util.**HashMap** and java.util.**Hashtable** in java >*

	Property	***java.util.HashMap***	***java.util.Hashtable***
1	synchronization	java.util.HashMap is **not synchronized** (because 2 threads on same HashMap object can access it at same time) in java.	java.util.Hashtable is **synchronized** (because 2 threads on same Hashtable object cannot access it at same time) in java.
2	Performance	HashMap is not synchronized, hence its operations are **faster** as compared to Hashtable in java.	Hashtable is synchronized, hence its operations are **slower** as compared to HashMap in java. If we are working not working in multithreading environment jdk recommends us to use HashMap.
3	Null keys and	HashMap allows to store **one null**	Hashtable does **not allow to store**

	values	**key** and **many null values** i.e. many keys can have null value in java.	**null key or null value**. Any attempt to store null key or value throws runtimeException (NullPointerException) in java.
4	Introduced in which java version	HashMap was introduced in second version of java i.e. **JDK 2.0**	Hashtable was introduced in first version of java i.e. **JDK 1.0** But it was refactored in java 2 i.e. JDK 1.2 to implement the Map interface, hence making it a member of member of the Java Collections Framework.
5	Recommendation	In non-multithreading environment it is recommended to use HashMap than using Hashtable in java.	**In java 5 i.e. JDK 1.5**, it is **recommended** to use ConcurrentHashMap than using Hashtable.
6	Extends Dictionary (Abstract class, which is obsolete)	HashMap does not extends Dictionary in java.	Hashtable extends Dictionary (which maps non-null keys to values. In a given Dictionary we can look up value corresponding to key) in java.

For more detail read : **HashMap and Hashtable - Similarity and Differences in java**

Collection interview Question 16. when to use HashSet vs LinkedHashSet vs TreeSet in java?

Answer. Another very important collection framework interview question to differentiate between **following Set implementations** in java.

Differences *between java.util.**HashSet** vs java.util.**LinkedHashSet** vs java.util.**TreeSet** in java>*

	Property	*java.util.HashSet*	*java.util.LinkedHashSet*	*java.util.TreeSet*

1	Insertion order	java.util.HashSet does not maintains insertion order in java.
		Example in java >
		`set.add("b");`
		`set.add("c");`
		`set.add("a");`
		Output >
		`No specific order`
			java.util.LinkedHashSet maintains insertion order in java.
			Example in java >
			`set.add("b");`
			`set.add("c");`
			`set.add("a");`
			Output >
			`b`
			`c`
			`a`
				java.util.TreeSet is sorted by natural order in java.
				Example in java >
				`set.add("b");`
				`set.add("c");`
				`set.add("a");`
				Output >
				`a`
				`b`
				`c`

2	Null elements	HashSet allows to store **one null** in java.	LinkedHashSet allows to store **one null** in java.	TreeSet does **not** allows to store **any null** in java. Any attempt to add null throws runtimeException (NullPointerException).
3	Data structure internally used for storing data	For storing elements HashSet internally uses HashMap.	For storing elements LinkedHashSet internally uses LinkedHashMap.	For storing elements TreeSet internally uses TreeMap.
4	Introduced in which java version	java.util.HashSet was introduced in second version of java (1.2) i.e. **JDK 2.0**	java.util.LinkedHashSet was introduced in second version of java (1.4) i.e. **JDK 4.0**	java.util.TreeSet was introduced in second version of java (1.2) i.e. **JDK 2.0**
5	Implements which interface	HashSet implements **java.util.Set** interface.	LinkedHashSet implements **java.util.Set** interface.	TreeSet implements **java.util.Set** **java.util.SortedSet** **java.util.NavigableSet** interface.

For more detail read : **HashSet vs LinkedHashSet vs TreeSet in java**

Collection interview Question 17. What are differences between HashMap and ConcurrentHashMap in java?

Answer. Take my words java developers won't be able to get away from this very important collection framework interview question.

***Differences** between java.util.**HashMap** and java.util.concurrent.ConcurrentHashMap in java >*

Property	*java.util.**HashMap***	*java.util.concurrent.**ConcurrentHashMap***
synchronization	HashMap is **not synchronized.**	ConcurrentHashMap is **synchronized**.
2 threads on same Map object can access it at concurrently?	Yes, because HashMap is not synchronized**.**	Yes. But how despite of being synchronized, 2 threads on same *ConcurrentHashMap* object can access it at same time? *ConcurrentHashMap* is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently.
Performance	We will **synchronize HashMap and then compare its performance with ConcurrentHashMap**. *We can synchronize hashMap by using Collections's class **synchronizedMap** method.* *Map synchronizedMap = Collections.**synchronizedMap**(hashMap);* *Now, no 2 threads can access same instance of map concurrently.* **Hence synchronized HashMap's performance is slower as compared to ConcurrentHashMap.** But why we didn't compared HashMap (unSynchronized) with ConcurrentHashMap? Because performance of unSynchronized collection is always better than some synchronized collection. As, default (unSynchronized) hashMap didn't cause any locking.	**ConcurrentHashMap's performance is faster as compared to HashMap (**because it is divided into segments, as discussed in above point**).** *Read this post for performance comparison between HashMap and ConcurrentHashMap.*
Null keys and values	HashMap allows to store **one null key** and **many null values** i.e. any key can have null value.	ConcurrentHashMap does **not allow to store null key or null value**. Any attempt to store null key or value throws runtimeException (NullPointerException).

iterators	The iterators returned by the iterator() method of HashMap are ***fail-fast*** >
	hashMap.keySet().iterator()
	hashMap.values().iterator()
	hashMap.entrySet().iterator()
	all three iterators are ***fail-fast***
		iterators are ***fail-safe***.
		concurrentHashMap.keySet().iterator()
		concurrentHashMap.values().iterator()
		concurrentHashMap.entrySet().iterator()
		all three iterators are ***fail-safe.***

putIfAbsent	HashMap does not contain putIfAbsent method.	If map does not contain specified `key`, put specified `key-value` pair in map and return null.

	***putIfAbsent** method is equivalent to writing following code >*
	`synchronized (map){`
	`    if (!map.containsKey(key))`
	`    return map.put(key, value);`
	`    else`
	`       return map.get(key);`
	`}`
	Program to create method that provides functionality similar to putIfAbsent method of ConcurrentHashMap and to be used with HashMap
		If map already contains specified `key`, return value corresponding to specified `key`.
		Program to use ConcurrentHashMap's putIfAbsent method

Introduced in which java version	HashMap was introduced in **java 2 i.e. JDK 1.2**,	ConcurrentHashMap was introduced in **java 5** i.e. **JDK 1.5**, since its introduction Hashtable has become obsolete, because of concurrency level its performance is better than Hashtable.
Implements which interface	HashMap implements **java.util.Map**	ConcurrentHashMap implements **java.util.Map** and **java.util.concurrent.ConcurrentMap**
Package	HashMap is in **java.util** package	ConcurrentHashMap is in **java.util.concurrent** package.

For more detail read : HashMap and ConcurrentHashMap in java

Collection interview Question 18. When to use HashMap vs Hashtable vs LinkedHashMap vs TreeMap in java?

Answer. Another important collection framework interview question to differentiate between **following Map implementations** in java.

***Differences** between java.util.**HashMap** vs java.util.**Hashtable** vs java.util.**LinkedHashMap** vs java.util.**TreeMap** >*

	Property	*HashMap*	*Hashtable*	*LinkedHashMap*	*TreeMap*
1	Insertion order	HashMap does not maintains insertion order in java.	Hashtable does not maintains insertion order in java.	LinkedHashMap maintains insertion order in java.	TreeMap is sorted by natural order of keys in java.
2	Performance	HashMap is not synchronized, hence its operations are **faster** as compared to Hashtable.	Hashtable is synchronized, hence its operations are **slower** as compared HashMap. If we are working not working in multithreading environment jdk recommends us to use HashMap.	LinkedHashMap must be used only when we want to maintain insertion order. **Time and space overhead** is there because for maintaining order it internally uses **Doubly Linked list**.	TreeMap must be used only when we want sorting based on natural order. Otherwise sorting operations cost performance. (Comparator is called for sorting purpose)
3	Null keys and values	HashMap allows to store **one null key** and	Hashtable does **not allow to store**	LinkedHashMap allows to store **one null key** and **many null values** i.e. any	TreeMap does **not allow to store null key but allow many null values**.

		many null values i.e. many keys can have null value in java.	**null key or null value**. Any attempt to store null key or value throws runtimeException (NullPointerException) in java.	key can have null value in java.	Any attempt to store null key throws runtimeException (NullPointerException) in java.
4	Implements which interface	HashMap implements **java.util.Map**	Hashtable implements **java.util.Map**	LinkedHashMap implements **java.util.Map**	TreeMap implements **java.util.Map** **java.util.SortedMap** **java.util.NavigableMap**
5	Implementation uses?	HashMap use **buckets**	Hashtable use **buckets**	LinkedHashMap uses **doubly linked lists**	TreeMap uses **Red black tree**
6	Complexity of put, get and remove methods	O(1)	O(1)	O(1) **overhead** of updating **Doubly Linked list** for maintaining order it internally uses.	O(log(n))
7	Extends java.util.**Dictionary** (Abstract class, which is obsolete)	HashMap **doesn't** extends Dictionary.	Hashtable **extends** Dictionary (which maps non-null keys to values. In a given Dictionary we can look up value corresponding to key)	LinkedHashMap **doesn't** extends Dictionary.	TreeMap **doesn't** extends Dictionary.
8	Introduced in which java version?	HashMap was introduced in second version of java i.e. **JDK 2.0**	Hashtable was introduced in first version of java i.e. **JDK 1.0** But it was refactored in java 2 i.e. JDK 1.2 to implement the Map interface, hence making it a member of member of the Java Collections Framework.	LinkedHashMap was introduced in fourth version of java i.e. **JDK 4.0**	TreeMap was introduced in second version of java i.e. **JDK 2.0**

For more detail read : HashMap vs Hashtable vs LinkedHashMap vs TreeMap in java

Collection interview Question 19. What are differences between HashMap vs IdentityHashMap in java?

Answer. This is tricky and complex collection framework interview question for experienced developers in java.

***Differences** between java.util.**HashMap** and java.util.**IdentityHashMap** in java >*

	Property	*java.util.HashMap*	*java.util.IdentityHashMap*
1	**Keys comparison**	**HashMap** when comparing	**IdentityHashMap** when comparing

	object-equality vs reference-equality	keys (and values) performs object-equality not reference-equality. In an HashMap, two keys k1 and k2 are equal if and only if (k1==null ? k2==null : k1.equals(k2))	keys (and values) performs reference-equality in place of object-equality. In an IdentityHashMap, two keys k1 and k2 are equal if and only if (k1==k2)
2	Initial size	Constructs a new HashMap, Its initial capacity is 16 in java. `new HashMap();`	Constructs a new IdentityHashMap, with maximum size of 21 in java. `new IdentityHashMap();`
3	Introduced in which java version	HashMap was introduced in second version of java i.e. **JDK 2.0**	IdentityHashMap was introduced in fourth version of java i.e. **JDK 4.0**
4	*Program*	Program 1 shows > *comparing keys (and values) performs object-equality in place of reference-equality . In an HashMap, two keys k1 and k2 are equal if and only if* **(k1==null ? k2==null : k1.equals(k2)).**	Program 2 shows > *comparing keys (and values) performs reference-equality in place of object-equality. In an IdentityHashMap, two keys k1 and k2 are equal if and only if* **(k1==k2).**
5	overridden equals() and hashCode() method call?	*overridden equals() and hashCode() method* are called when put, get methods are called in ***HashMap***. As shown in Program 3.	*overridden equals() and hashCode() method* are not called when put, get methods are called in ***IdentityHashMap***. *Because IdentityHashMap implements equals() and hashCode() method by itself and checks for reference-equality of keys.* As shown in Program 4.
6	Application - can maintain *proxy object*	HashMap cannot be used to maintain *proxy object.*	IdentityHashMap can be used to maintain *proxy objects*. For example, we might need to maintain proxy object for each object debugged in the program.

For more detail read : HashMap vs IdentityHashMap - Similarity and Differences with program in java

Collection interview Question 20. What is WeakHashMap in java?

Answer. Another tricky collection framework interview question for experienced developers in java.

java.util.WeakHashMap is hash table based implementation of the Map interface, with *weak keys*. An entry in a WeakHashMap will be automatically removed by garbage collector when its key is no longer in ordinary use. Mapping for a given key will not prevent the key from being discarded by the garbage collector, (i.e. made finalizable, finalized, and then reclaimed). When a key has been discarded its entry is removed from the map in java.

java.util.**WeakHashMap** is implementation of the java.util.Map interface in java.

The behavior of the java.util.WeakHashMap class depends upon garbage collector

The behavior of the WeakHashMap class depends upon garbage collector in java. Because the garbage collector may discard keys at any time, in WeakHashMap it may look like some unknown thread is silently removing entries. Even if you synchronize WeakHashMap instance and invoke none of its methods,

- it is possible for the **size** method to return smaller values over time,
- for **isEmpty** method to return false and then true,
- for **containsKey** method to return true and later false for a given key,
- for **get** method to return a value for a given key but later return null,
- for **put** method to return null, and
- for **remove** method to return false for a key that previously existed in the WeakHashMap.

Each key object in a WeakHashMap is stored indirectly as the referent of a weak reference. Therefore a key will be removed automatically only after the weak references to it, both inside and outside of the map, have been cleared by the garbage collector.

Collection interview Question 21. What is EnumSet in java?

Answer. Freshers must know about EnumMap in java.

A java.util.EnumSet is specialized **Set** implementation for use with enum types in java.

EnumSet all elements comes from a single enum type that is specified when the set is created in java.

Order of elements in EnumSet in java

The java.util.EnumSet maintains ***natural order*** (the order in which the enum constants are declared) of elements in java.

Iterator on EnumSet in java

The iterator returned by the iterator method traverses the elements in their ***natural order*** (the order in which the enum constants are declared).

iterator never throw ConcurrentModificationException and it may or may not show the effects of any modifications to the set that occur while the iteration is in progress.

Null elements in EnumSet in java

Null elements are not allowed in EnumSet in java. Attempts to insert a null element will throw NullPointerException in java.

Collection interview Question 22. What is EnumMap in java?

Answer. Freshers must be able to answer this collection framework interview question in java. A java.util.EnumMap is specialized **Map** implementation for use with enum type keys.

EnumMap all keys comes from a single enum type that is specified when the set is created in java.

Order of keys in EnumMap in java

The EnumMap maintains ***natural order*** (the order in which the enum constants are declared) of keys in java.

Iterator on EnumMap in java

The iterator returned by the iterator method in EnumMap traverses the elements in their ***natural order* of keys**(the order in which the enum constants are declared).

iterator never throw ConcurrentModificationException and it may or may not show the effects of any modifications to the map that occur while the iteration is in progress in java.

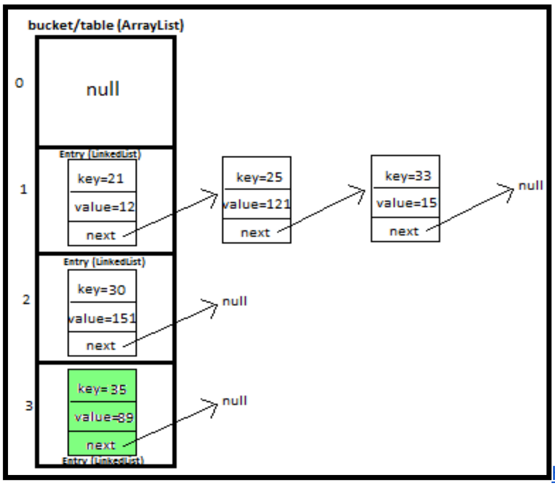
Null allowed in EnumMap in java?

Null keys are not allowed in EnumMap. Attempts to insert a null key will throw NullPointerException.

But, Null values are allowed in EnumMap in java.

Collection interview Question 23. How to implement own/custom HashMap in java? Or How HashMap works in java?

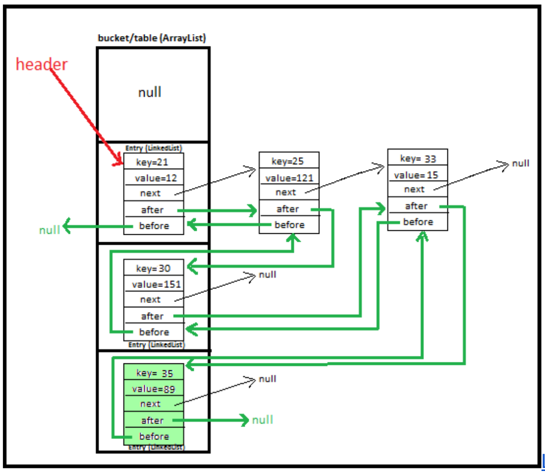
Answer.



HashMap Custom implementation/ HashMap works in java

Collection interview Question 24. How to implement own LinkedHashMap in java? Or LinkedHashMap works in java?

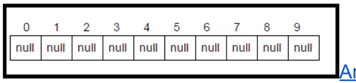
Answer.



LinkedHashMap Custom implementation/How LinkedHashMap works in java

Collection interview Question 25. How to implement own ArrayList in java?Or How ArrayList works in java ?

Answer.

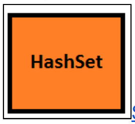


0	1	2	3	4	5	6	7	8	9
null	null	null	null	null	null	null	null	null	null

ArrayList custom implementation / How ArrayList works in java

Collection interview Question 26. How to implement own HashSet in java? Or How HashSet works in java ?

Answer.



Set Custom implementation/ Or How HashSet works in java

Collection interview Question 27. How to implement own LinkedHashSet in java? Or How LinkedHashSet works in java ?

Answer.

LinkedHashSet Custom implementation/ How LinkedHashSet works in java

Collection interview Question 28. What do you mean by fail-fast and fast-safe? What is ConcurrentModificationException?

Answer.

Iterator returned by few Collection framework Classes are **fail-fast,** means any structural modification made to these classes during iteration will throw ConcurrentModificationException.

Some important classes whose returned iterator is **fail-fast >**

- ArrayList
- LinkedList
- vector
- HashSet

Iterator returned by few Collection framework Classes are **fail-safe,** means any structural modification made to these classes during iteration won’t throw any Exception.

Some important classes whose returned iterator is **fail-safe >**

- CopyOnWriteArrayList
- CopyOnWriteArraySet
- ConcurrentSkipListSet

For more detail read : ConcurrentModificationException, Fail-fast and Fail-safe in detail in java

Collection interview Question 29. What are different ways of iterating over elements in List?

Answer.

Creating ArrayList and add element.

```
List<String> arrayList=new ArrayList<String>();
arrayList.add("javaMadeSoEasy");
```

*1. Iterate over elements in ArrayList using **iterator()***

iterator() method returns iterator to iterate over elements in ArrayList.

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

iterator returned by ArrayList is fail-fast.

*2. Iterate over elements in ArrayList using **listIterator()***

```
      ListIterator<String> listIterator=arrayList.listIterator();
```

ListIterator returned by ArrayList is also fail fast.

*3. Iterate over elements in list using **enumeration***

```
   Enumeration<String> listEnum=Collections.enumeration(arrayList);
    while(listEnum.hasMoreElements()){
       System.out.println(listEnum.nextElement());
    }
```

enumeration is also fail-fast.

*4. Iterate over elements in list using **enhanced for loop***

```
           for (String string : arrayList) {
               System.out.println(string);
        }
```

enhanced for loop is also fail-fast.

Collection interview Question 30. What are different ways of iterating over elements in Set?

Answer. *Creating HashSet and add element.*

```
Set<String> hashSet=new HashSet<String>();
hashSet.add("javaMadeSoEasy");
```

1. Iterate over elements in HashSet using iterator()

iterator() method returns iterator to iterate over elements in HashSet.

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

iterator returned by HashSet is fail-fast.

2. Iterate over elements in Set using enumeration

```
     Enumeration<String> listEnum=Collections.enumeration(set);
     while(listEnum.hasMoreElements()){
        System.out.println(listEnum.nextElement());
     }
```

enumeration is also fail-fast.

*3. Iterate over elements in Set using **enhanced for loop***

```
           for (String string : set) {
               System.out.println(string);
        }
```

enhanced for loop is also fail-fast.

Collection interview Question 31. What are different ways of iterating over keys, values and entry in Map?

Answer. *Create and put key-value pairs in HashMap >*

```
        Map<Integer,String> hashMap=new HashMap<Integer,String>();
              hashMap.put(11, "javaMadeSoEasy");
        hashMap.put(21, "bmw");
        hashMap.put(31, "ferrari");
```

1. Iterate over keys -

hashMap.keySet().iterator() method returns iterator to iterate over keys in HashMap.

```
Iterator<Integer> keyIterator=hashMap.keySet().iterator();
while(keyIterator.hasNext()){
  System.out.println(keyIterator.next());
}

/*OUTPUT
21
11
31
*/
```

2. Iterate over values -

hashMap.values().iterator() method returns iterator to iterate over keys in HashMap.

```
Iterator<String> valueIterator=hashMap.values().iterator();
while(valueIterator.hasNext()){
  System.out.println(valueIterator.next());
}

/*OUTPUT
javaMadeSoEasy
audi
ferrari
*/
```

iterator returned is fail-fast..

3. Iterate over Entry-

hashMap.entrySet().iterator() method returns iterator to iterate over keys in HashMap.

```
Iterator<Entry<Integer, String>>
entryIterator=hashMap.entrySet().iterator();
while(entryIterator.hasNext()){
    System.out.println(entryIterator.next());
}

/*OUTPUT
21=javaMadeSoEasy
11=audi
31=ferrari
*/
```

iterator returned is fail-fast..

Collection interview Question 32. What is difference between Comparable and Comparator? How can you sort List?

Answer.

	Property	***Comparable***	***Comparator***
1	Comparing instances of class	Comparable is used to compare instances of same class	Comparator can be used to compare instances of same or different classes.
2	**sorting order**	Comparable can be implemented by class which need to define a **natural ordering for its objects.**	Comparator is implemented when one wants a **different sorting order** and define custom way of comparing two instances.

		Example - String, Integer, Long , Date and all other wrapper classes implements Comparable.	

3	Changes to class	For using Comparable, original Class must implement it.
		
		Example-
		
		`class Employee implements Comparable<Employee>`
		
		For using Comparable, Employee Class must implement it, no other class can implement it.
		
		As used in **Program 1**
			Class itself can implement Comparator or
			any other class can implement Comparator. Hence avoiding modification to original class.
			
			Example-
			
			`class ComparatorName implements Comparator<Employee>`
			
			`class ComparatorId implements Comparator<Employee>`
			
			In above example modifications were made to `ComparatorName` and `ComparatorId.` Hence avoiding modification to Employee class.
			
			As used in **Program 4**

4	Sorting on basis on one or many criteria	Provides sorting only on **one** criteria, **because** Comparable can be implemented by original class only.	We can use Comparator to sort class on **many** criterias **because** class itself or any other class can implement Comparator.

5	Method	compareTo method
		
		`@Override`
		`public int compareTo(Employee obj) {`
		` //sort Employee on basis of name(ascending order)`
		`  return this.name.compareTo(obj.name);`
		`}`
		
		Method compares **this** with **obj** object and returns a integer.
		
		• positive – **this** is **greater** than **obj**
		• zero – **this** is **equal** to **obj**
		• negative – **this** is **less** than **obj**
		
		As used in **Program 1**
			compare method
			
			`@Override`
			`public int compare(Employee obj1, Employee obj2) {`
			` //sort Employee on basis of name(ascending order)`
			`    return obj1.name.compareTo(obj2.name);`
			`}`
			
			Method compares **obj1** with **obj2** object and returns a integer.
			
			• positive – **obj1** is **greater** than **obj2**
			• zero – **obj1** is **equal** to **obj2**
			• negative – **obj1** is **less** than **obj2**
			
			As used in **Program 3**

6	Package	**java.lang** **java.lang** package is automatically imported by every program in java. Hence, we need to write explicit statement for importing java.lang.Comparable.	**java.util** We need to write explicit import statement - `import java.util.Comparator`
7	Using **Collections.sort**	Let's say we wanna sort list of Employee, **Collections.sort(**list**)** uses Comparable interface for sorting class. As used in Program 1	Let's say we wanna sort list of Employee, `Collections.sort(list,new ComparatorName());` uses Comparator interface for sorting class. As used in Program 5

Read more : [Comparable vs Comparator - differences and sorting list by implementing Comparable and Comparator in classes and inner classes]

Collection interview Question 33. How sort method of Collections class works internally?

Answer. *Collections.sort internally calls Arrays.sort,*

Arrays.Sort() internally uses Merge Sort.

If number of elements is less than 7 then Insertion Sort is used rather than *Merge Sort*. (because in case elements are less than 7 it offers better time complexity)

Collection interview Question 34. How can you sort given HashMap on basis of keys?

Answer.

Please Read : Sort Map by key in Ascending and descending order by implementing Comparator interface and overriding its compare method and using TreeMap

Collection interview Question 35. How can you sort given HashMap on basis of values?

Answer.

Please Read : Sort Map by value in Ascending and descending order by implementing Comparator interface and overriding its compare method

Collection interview Question 36. In what all possible ways you can sort a given Set?

Answer.

Please Read : Sort Set by using TreeSet and by implementing Comparator and Comparable interface

Collection interview Question 37. How you can sort arrays? And how Comparator of superclass can be used by subclasses?

Answer.

Please Read : Arrays.sort to sort arrays by implementing Comparator and how Comparator of superclass can be used by subclasses

Collection interview Question 38. What are differences between ArrayList vs CopyOnWriteArrayList?

Answer.

***Differences** between java.util.**ArrayList** and java.util.concurrent.**CopyOnWriteArrayList** in java >*

	Property	***java.util.ArrayList***	**java.util.concurrent. CopyOnWriteArrayList**
1	synchronization	ArrayList is not **synchronized** (because 2 threads on same ArrayList object can access it at same time). I have created **program** to show see consequence of using ArrayList in multithreading environment. In the program i will implement our own arrayList.	**CopyOnWriteArrayList** is **synchronized** (because 2 threads on same CopyOnWriteArrayList object cannot access it at same time).
2	Iterator and listIterator	Iterator and listIterator returned by ArrayList are **Fail-fast**, means any structural modification made to ArrayList during iteration using Iterator or listIterator will throw ConcurrentModificationException in java.	Iterator and listIterator returned by CopyOnWriteArrayList are **Fail-safe** in java. As shown in Program 2 below.

		As shown in Program 1 below.	
3	Enumeration is fail-fast	**Enumeration** returned by ArrayList is **fail-fast**, means any structural modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException. As shown in Program 1 below.	**Enumeration** returned by CopyOnWriteArrayList is **fail-safe.** As shown in Program 2 below.
4	Iterate using **enhanced for loop**	Iteration done on ArrayList using **enhanced for loop** is **Fail-fast,** means any structural modification made to ArrayList during iteration using **enhanced for loop** will throw ConcurrentModificationException. As shown in Program 1 below.	Iteration done on CopyOnWriteArrayList using **enhanced for loop** is **Fail-safe.** As shown in Program 2 below.
5	Performance	ArrayList is not synchronized, hence its operations are **faster** as compared to CopyOnWriteArrayList.	CopyOnWriteArrayList is synchronized, hence its operations are **slower** as compared to ArrayList.
6	AbstractList	ArrayList extends AbstractList (abstract class) which provides implementation to List interface to minimize the effort required to implement this interface backed by RandomAccess interface.	CopyOnWriteArrayList does not extends AbstractList, though CopyOnWriteArrayList also implements RandomAccess interface.
7	Introduced in which java version	ArrayList was introduced in second version of java (1.2) i.e. **JDK 2.0**	CopyOnWriteArrayList was introduced in fifth version of java (1.5) i.e. **JDK 5.0**
8	Package	java.util	java.util.**concurrent**

For more detail read : ArrayList vs CopyOnWriteArrayList - Similarity and Differences with program

Collection interview Question 39. What are differences between HashSet vs CopyOnWriteArraySet?

Answer.

***Differences** between java.util.**HashSet** and java.util.concurrent.**CopyOnWriteArraySet** in java >*

	Property	***java.util.HashSet***	**java.util.concurrent. CopyOnWriteArraySet**
1	synchronization	HashSet is not **synchronized** (because 2 threads on same HashSet object can access it at same time) in java.	**CopyOnWriteArraySet** is **synchronized** (because 2 threads on same CopyOnWriteArraySet object cannot access it at same time) in java.
2	Iterator	Iterator returned by HashSet is **Fail-fast,** means any structural modification made to HashSet during iteration using Iterator will throw ConcurrentModificationException in java. As shown in Program 1 below.	Iterator returned by **CopyOnWriteArraySet** is **Fail-safe** in java. As shown in Program 2 below.
3	Enumeration is fail-fast	**Enumeration** returned by HashSet is **fail-fast**, means any structural modification	**Enumeration** returned by CopyOnWriteArraySet is **fail-safe.**

		made to HashSet during iteration using Enumeration will throw ConcurrentModificationException. As shown in Program 1 below.	As shown in Program 2 below.
4	Iterate using **enhanced for loop**	Iteration done on HashSet using **enhanced for loop** is **Fail-fast,** means any structural modification made to HashSet during iteration using **enhanced for loop** will throw ConcurrentModificationException. As shown in Program 1 below.	Iteration done on CopyOnWriteArraySet using **enhanced for loop** is **Fail-safe.** As shown in Program 2 below.
5	Performance	HashSet is not synchronized, hence its operations are **faster** as compared to CopyOnWriteArraySet.	CopyOnWriteArraySet is synchronized, hence its operations are **slower** as compared to HashSet.
6	Introduced in which java version	HashSet was introduced in second version of java (1.2) i.e. **JDK 2.0**	CopyOnWriteArraySet was introduced in fifth version of java (1.5) i.e. **JDK 5.0**
7	Package	java.util	java.util.**concurrent**

For more detail read : HashSet vs CopyOnWriteArraySet - Similarity and Differences with program

Collection interview Question 40. What are differences between TreeSet vs ConcurrentSkipListSet?

Answer.

***Differences** between java.util.**TreeSet** and java.util.concurrent.**ConcurrentSkipListSet** in java >*

	Property	***java.util.TreeSet***	**java.util.concurrent. ConcurrentSkipListSet**
1	synchronization	TreeSet is not **synchronized** (because 2 threads on same TreeSet object can access it at same time) in java.	**ConcurrentSkipListSet** is **synchronized** (because 2 threads on same ConcurrentSkipListSet object cannot access it at same time) in java.
2	Iterator	Iterator returned by TreeSet is **Fail-fast**, means any structural modification made to TreeSet during iteration using Iterator will throw ConcurrentModificationException in java. As shown in Program 1 below.	Iterator returned by **ConcurrentSkipListSet** is **Fail-safe** in java. As shown in Program 2 below.
3	Enumeration is fail-fast	**Enumeration** returned by TreeSet is **fail-fast**, means any structural modification made to TreeSet during iteration using Enumeration will throw ConcurrentModificationException. As shown in Program 1 below.	**Enumeration** returned by ConcurrentSkipListSet is **fail-safe.** As shown in Program 2 below.
4	Iterate using **enhanced for loop**	Iteration done on TreeSet using **enhanced for loop** is **Fail-fast,** means any structural modification made	Iteration done on ConcurrentSkipListSet using **enhanced for loop** is **Fail-safe.**

		to TreeSet during iteration using **enhanced for loop** will throw ConcurrentModificationException. As shown in Program 1 below.	As shown in Program 2 below.
5	Performance	TreeSet is not synchronized, hence its operations are **faster** as compared to ConcurrentSkipListSet.	ConcurrentSkipListSet is synchronized, hence its operations are **slower** as compared to TreeSet.
6	Introduced in which java version	TreeSet was introduced in second version of java (1.2) i.e. **JDK 2.0**	ConcurrentSkipListSet was introduced in sixth version of java (1.6) i.e. **JDK 6.0**
7	Package	java.util	java.util.**concurrent**

For more detail read : TreeSet vs ConcurrentSkipListSet - Similarity and Differences with program

Collection interview Question 41. What are differences between TreeMap vs ConcurrentSkipListMap?

Answer.

***Differences** between java.util.**TreeMap** and java.util.concurrent.**ConcurrentSkipListMap** in java >*

	Property	***java.util.TreeMap***	***java.util.concurrent. ConcurrentSkipListMap***
1	synchronization	TreeMap is **not synchronized** (because 2 threads on same TreeMap object can access it at same time) in java.	ConcurrentSkipListMap is **synchronized** (because 2 threads on same ConcurrentSkipListMap object cannot access it at same time) in java.
2	Iterator	The iterators returned by the iterator() method of Map's “collection view methods" are *fail-fast*> - *map.keySet().iterator()* - *map.values().iterator()* - *map.entrySet().iterator()* all three iterators are ***fail-fast,*** means any structural modification made to TreeMap during iteration using any of 3 Iterator will throw ConcurrentModificationException. As shown in Program 1 below.	The iterators returned by the iterator() method of Map's “collection view methods" are *fail-safe* > - *map.keySet().iterator()* - *map.values().iterator()* - *map.entrySet().iterator()* all three iterators are ***fail-safe.*** As shown in Program 2 below.
3	Performance	TreeMap is not synchronized, hence its operations are **faster** as compared to ConcurrentSkipListMap.	ConcurrentSkipListMap is synchronized, hence its operations are **slower** as compared to TreeMap.
4	Introduced inin which java version	TreeMap was introduced in second version of java i.e. **JDK 2.0**	ConcurrentSkipListMap was introduced in sixth version of java i.e. **JDK 6.0**
5	Package	java.util	java.util.**concurrent**
6	Implements which interface	Map SortedMap NavigableMap	Map SortedMap NavigableMap ConcurrentNavigableMap

For more detail read : TreeMap vs ConcurrentSkipListMap - Similarity and Differences with program

Collection interview Question 43. Can we use null element in TreeSet? Give reason?

Answer. No, TreeSet does **not** allows to store **any null keys**.

Any attempt to add null throws runtimeException (NullPointerException).

TreeSet internally compares elements for sorting elements by natural order (comparator may be used for sorting, if defined at creation time) and null is not comparable, Any attempt to compare null with other object will throw NullPointerException.

Collection interview Question 44. Can we use null key in TreeMap? Give reason?

Answer. No, TreeMap **not allow to store null key.**

Any attempt to store null key throws runtimeException (NullPointerException).

TreeMap internally compares keys for sorting keys by natural order (comparator may be used for sorting, if defined at creation time) and null is not comparable, Any attempt to compare null with other object will throw NullPointerException.

Collection interview Question 45. How ConcurrentHashMap works? Can 2 threads on same ConcurrentHashMap object access it concurrently?

Answer. *ConcurrentHashMap* is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently.

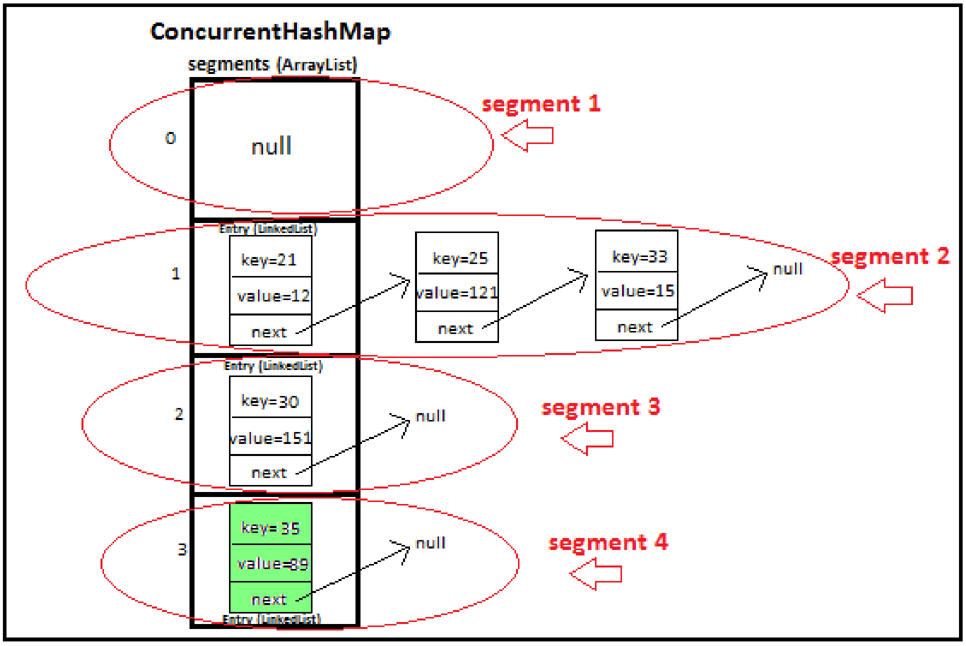
Can threads read the segment locked by some other thread?
Yes. When thread locks one segment for updation it does not block it for retrieval (done by get method) hence some other thread can read the segment (by get method), but it will be able to read the data before locking.

For operations such as putAll concurrent retrievals may reflect removal of only some entries.
For operations such as clear concurrent retrievals may reflect removal of only some entries.

***Segments** in ConcurrentHashMap with **diagram** >*

we have ConcurrentHashMap with **4 segments -**

(Diagram shows how **segments** are formed in ConcurrentHashMap)



Collection interview Question 46. Write a program to show consequence of using ArrayList in multithreading environment?

Answer. Program to show consequence of using ArrayList in multithreading environment in java

dri blog

Collection interview Question 47. Write a program to show advantage of using Vector in multithreading environment?

Answer. Program to show advantage of using Vector in multithreading environment in java

Collection interview Question 48. Mention properties of most frequently used Collection classes and Interfaces? Mention as many properties as much you can.

Answer. This question is real test for experienced developers, this will test your in depth awareness of Collection classes and Interfaces. Answering this question in detail will really ensure your selection.

List	**Duplicate elements**	**insertion order**	**Sorted by natural order**	**synchronized**	**null elements**	**Iterator**
ArrayList	Yes	Yes			Yes	Iterator & listIterator are Fail-fast
LinkedList	Yes	Yes			Yes	Iterator & listIterator are Fail-fast
CopyOnWriteArrayList	Yes	Yes		Yes	Yes	Iterator & listIterator are **Fail-safe**
Set	**Duplicate elements**	**insertion order**	**Sorted by natural order**	**synchronized**	**null elements**	**Iterator**
HashSet					Yes	Fail-fast
LinkedHashSet		Yes			Yes	Fail-fast
TreeSet			Yes		No	Fail-fast
ConcurrentSkipListSet			Yes	Yes	No	**Fail-safe**

Map	**Duplicate Keys**	**insertion order of keys**	**Sorted by natural order of keys**	**synchronized**	**null keys or null values**	**Iterator**
						Map implementations returns 3 iterators >
						map.keySet().iterator()
						map.values().iterator()
						map.entrySet().iterator()

HashMap					one null key and many null values	All are Fail-fast
Hashtable				Yes	No	All are Fail-fast
ConcurrentHashMap				Yes	No	All are **Fail-safe**
TreeMap			Yes		Null key not allowed,	All are Fail-fast

					Allow many null values	
ConcurrentSkipListMap			Yes	Yes	No	All are **Fail-safe**

Collection - List, Set and Map all properties in tabular form

Collection interview Question. 49 Which list class must be preferred in multithreading environment, considering performance constraint?

Answer. CopyOnWriteArrayList

Collection interview Question 50. Which Set class must be preferred in multithreading environment, considering performance constraint?

Answer. CopyOnWriteArraySet (allows null and elements aren't sorted in natural order) **or**

ConcurrentSkipListSet (doesn't allows null and elements are sorted in natural order)
Select one depending on your requirement.

Collection interview Question 51. Which Map class must be preferred in multithreading environment, considering performance constraint?

Answer. ConcurrentHashMap(keys aren't sorted in natural order) **or** ConcurrentSkipListMap(keys are sorted in natural order)
Select one depending on your requirement.

Collection interview Question 52. Let's say you have to build dictionary and multiple users can add data in that dictionary? And you can use 2 Collection classes? Which Collection classes you will prefer and WHY?

Answer. It's very **important question** which test your **logical** reasoning and your ability to create robust applications in multithreading environment.

We must use ConcurrentSkipListMap and TreeSet >

```
ConcurrentSkipListMap<String, TreeSet<String>> myDictionary =
                           new ConcurrentSkipListMap<String, TreeSet<String>>();
```

Store words in ConcurrentSkipListMap as key>

- keys are sorted in **natural order** (words will be sorted in natural order),
- **doesn't allow null** keys (words can't be null)
- **doesn't allow duplicate** keys (words can't be duplicate) and
- synchronized, so 2 threads won't create synchronization problems (will take care of different uses adding words concurrently)

for storing meaning of word in dictionary we must use TreeSet as value in ConcurrentSkipListMap **because one word can have many meanings** >

- elements are sorted in **natural order** (meaning of word are sorted in natural order),
- **doesn't allow null** elements (meaning of word can't be null),
- **doesn't allow duplicate** elements (meaning of word can't be duplicate)

Program for creating and using Java dictionary using Collection classes>

```
package com.ankit.dictionary;

import java.util.TreeSet;
import java.util.concurrent.ConcurrentSkipListMap;
```

```
/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class MyDictionary {
    public static void main(String[] args) {
            ConcurrentSkipListMap<String, TreeSet<String>> myDictionary =
                          new ConcurrentSkipListMap<String, TreeSet<String>>();

           TreeSet<String> innocentMeaning = new TreeSet<String>();
           innocentMeaning.add("not responsible for an event yet suffering its consequences");
           innocentMeaning.add("not guilty of a crime");

           myDictionary.put("innocent", innocentMeaning);

           TreeSet<String> appealingMeaning = new TreeSet<String>();
           appealingMeaning.add("attractive");
           appealingMeaning.add("expressing a desire for help");

           myDictionary.put("appealing", appealingMeaning);

           System.out.println(myDictionary);

    }
}

/* OUTPUT

{appealing=[attractive, expressing a desire for help], innocent=[not guilty of a crime, not
responsible for an event yet suffering its consequences]}

*/
```

Collection interview Question 53. Why to use java.util.WeakHashMap map which is so inconsistent and unpredictable in behaviour?

Answer. Let's say we have huge application which consists of lots n lots of object and may run short of memory at any time, we will like garbage collector to quickly discard less used key value pair to free up some memory. As, behavior of the WeakHashMap class depends upon garbage collector.
I believe discarding less used key-value is always going to a better option than running out of memory.

Related >>

COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75

COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-3 > Q75- Q100

Labels: Collection Framework Core Java Interview questions

Must read for you :

6 Comments www.javamadesoeasy.com 🔒Disqus' Privacy Policy 1 Login

♡ Recommend 3 Tweet f Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Bhargav Patel • 4 years ago

Hi Javamdessoeasy,

Hi Went through your tutorial. It's very nice explanation . I appreciated. But one small doubt i have. Please attache image. It confused me. You said like....

"Can threads read the segment locked by some other thread?
Yes, When thread locks one segment for updation it does not block it for retrieval (done by get method) hence some other thread can read the segment (by get method), but it will be able to read the data before locking."

So initially you say yes we can read the segment locked by some other thread and at the end you said " it will be able to read the data before locking"

Please reply what does it mean ???

Thanks,
Bhargav

1 ⌃ | ⌄ • Reply • Share ›

Ankit Mittal Mod • 4 years ago

Dear **@Bhargav Patel**

Yes, t2 thread will not be able to get updated value until t1 thread released lock on segment.

There is nothing wrong with that. That is how concurrency is designed to provide you with most relevant data.

In all real time real scenarios you will face this many times.

⌃ | ⌄ • Reply • Share ›

Bhargav Patel ➦ Ankit Mittal • 4 years ago

Thanks Ankit Mittal,
Nice tutorial. Keep it up.

⌃ | ⌄ • Reply • Share ›

Ankit Mittal Mod ➦ Bhargav Patel • 4 years ago

Yeah Thanks @bhargav

⌃ | ⌄ • Reply • Share ›

Ankit Mittal Mod • 4 years ago

Dear **@Bhargav Patel** ,

Let's take thread 1 (t1) and thread 2 (t2)

t1 locks segment 2 (in diagram) and t1 tries to update key=21, it update it's value to 88, but it not yet has released lock on segment 2,
Mean while t2 tries to access get value corresponding to key=21, then it will get value as 12 not 88.
But why?
That simply means thread will be able to read the data before locking (i.e. t2 "will be able to read the data before locking").

⌃ | ⌄ • Reply • Share ›

Bhargav Patel ➜ Ankit Mittal • 4 years ago

That means t2 thread will not be able to get updated value until t1 thread released lock on segment !!!! is it so ?

If yes then this is waste full... Because ConcurrentHashMap is invented to used in multi threading environment. please correct me if i am wrong...

Thanks,
Bhargav

⌃ | ⌄ • Reply • Share ›

Newer Post Home Older Post