HowToDoInJava

Java Collections Interview Questions

Last Updated: January 25, 2022 | By: Lokesh Gupta | Interview Questions | Interview Questions

Java Collections is one of the most important areas where you will be tested in junior or senior positions. The scope of questions is so broad that it is almost impossible to cover all the questions. Yet based on my previous interviews, I am attempting to put as many as possible good *interview questions*.

I am aiming for *beginners and senior-level interviews*, so bear with me if you find some questions too basic because they might be helpful for other junior developers.

Table of Contents

Java Collections Framework Questions

1. What is Java Collections framework? What are its advantages?

By definition, a Collection is **an object that can store a group of objects.** Like in set theory, a set is a group of elements. Easy enough !!

Before JDK 1.2, JDK had some utility classes such as *Vector* and *HashTable*, and there was no concept of the Collection framework. Later from JDK 1.2 onwards, JDK developers felt the need to have consistent support for reusable data structures. Finally, the collections framework was designed and **developed primarily by Joshua Bloch**, and was **introduced in JDK 1.2**.

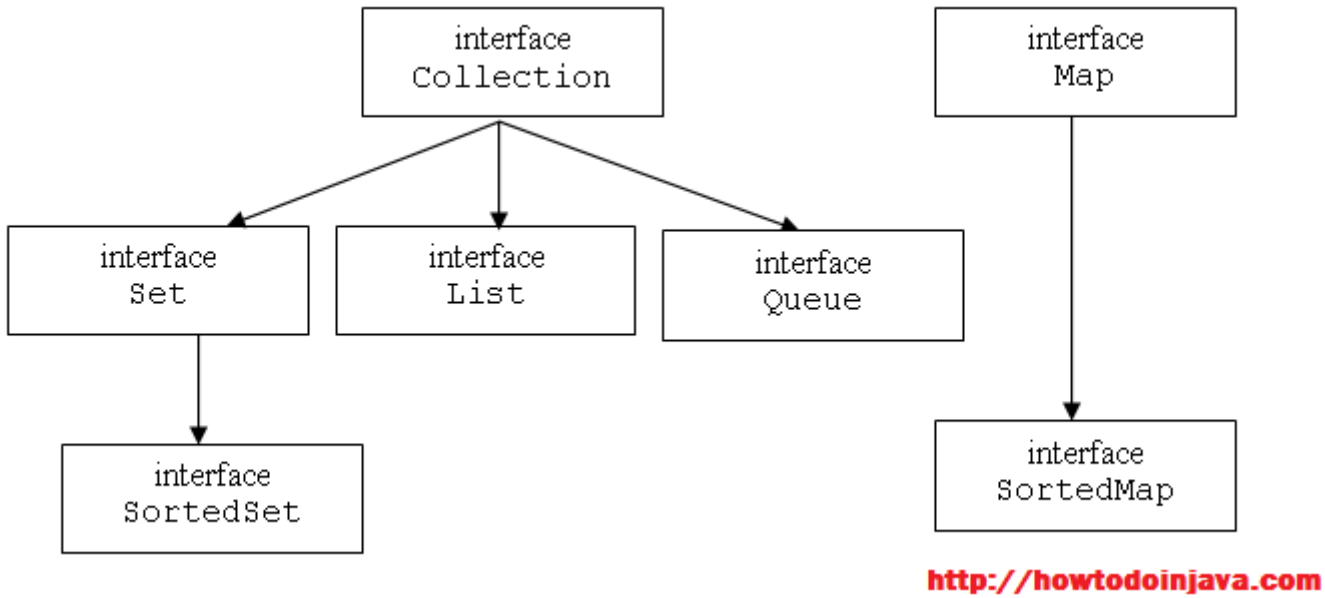
The ***Collections framework is a unified architecture for representing and manipulating collections***, manipulating collections independently of implementation details.

The framework provides classes and interfaces for representing different types of collections, such as *Set*, *List*, and *Map*. Besides providing basic implementations, it also provides other special-purpose implementations as well, for example, concurrent collections (*ConcurrentMap*, *BlockingQueue*, etc) and convenience implementations (*Arrays.asList()*, *Collections.emptySet()*, *Collections.singleton()*, etc)

We can list the most noticeable benefits of Java collections as:

- *Reduced programming effort* due to ready to use code
- *Increased performance* because of high-performance implementations of data structures and algorithms
- Provides *interoperability between unrelated APIs* by establishing a common language to pass collections back and forth
- *Easy to learn* APIs by learning only some top level interfaces and supported operations

2. What is Collections Hierarchy?



Java Collection Hierarchy

As shown in the above image, the collection framework has one interface at the top i.e. **Collection**. *Set*, *List* and *Queue* interfaces extend it. Then there are many other classes in these three branches.

Remember the signature of `Collection` interface. It will help you with many questions.

```
public interface Collection extends Iterable {
//method definitions
}
```

The framework also consists of *Map* interface, which is part of the collection framework but does not extend the *Collection* interface.

3. Why `Collection` does not extend Cloneable and Serializable interface?

Well, the most straightforward answer is “there is no need to do it”. Extending an interface means that you are creating a subtype of the interface, in other words, a more specialized behavior and *Collection* classes are not expected to do what *Cloneable* and *Serializable* interfaces do.

Another reason is that not everybody will have a reason to create *Cloneable* collections because if it may contain huge data, then every **unnecessary clone operation will consume a significant memory**. Beginners might use it without knowing the consequences.

Another reason is that **Cloneable and Serializable are very specialized behavior** and so should be implemented only when required. For example, many concrete classes in Collection implement these interfaces. So if you want this feature, use these collection classes; otherwise, use their alternative classes.

4. Why `Map` interface does not extend `Collection` interface?

An excellent answer to this interview question is "**because they are incompatible**".

The *Collection* interface has a method `add(Object o)`. The *Map* can not have such a method because it needs key-value pair. There are other reasons also such as *Map* supports *EntrySet etc.* Collection classes do not have such views.

Due to such big differences, the *Map* interface was not included in the Collection framework hierarchy, and it was built in a separate hierarchy.

List Interface Questions

5. What is a *List*? What are Core Classes implementing *List*?

- *List* is **index-based** and **an ordered collection of elements**. The ordering is a zero-based index.
- *List* **allows the duplicate items**.
- Apart from methods defined in *Collection* interface, it does have its own methods also which are to manipulate the List based on the index location of the item. These methods can be grouped as *search, get, iteration and range view*. All above operations support index locations.

The main classes implementing the *List* interface are *Stack*, *Vector*, *ArrayList* and *LinkedList*.

6. How to Convert from Array to List?

This is a *coding-related question* that can be asked at the beginner level. The question intends to check the knowledge of applicants in *Collection utility classes*.

For now, let us learn that there are two utility classes in the Collection framework which are mostly seen in interviews:

- `Collections`
- `Arrays`

Collections class provides static functions to perform specific operations on various 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 note that this function is not specific to the *String* class. This can return a *List* of items of any class type.

7. How to Reverse a List?

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

```
Collections.reverse(list);
```

Set Interface Questions

8. What is a Set? What are Core Classes Implementing Set Interface?

A *Set* models the mathematical set in set theory. The *Set* interface is similar to *List* interface but with some differences.

- Unlike *List*, A *Set* **does not have index-based operations**. It only has methods that are inherited by the *Collection* interface.
- A *Set* is **not an ordered collection**. So no ordering is preserved while adding or removing elements.
- A *Set* guarantees the "uniqueness of elements". It **does not store duplicate elements**.

Set also adds a stronger contract on the behavior of the equals() and hashCode() operations, allowing two *Set* instances to be compared meaningfully even if their implementation types differ. *Two Set instances are equal if they contain the same elements.*

Main classes implementing Set interface are : *EnumSet*, *HashSet*, *LinkedHashSet*, *TreeSet*.

9. How HashSet Store Unique Items?

We must know that a *HashMap* stores key-value pairs, with one condition that keys will be unique. **HashSet uses unique keys feature of the Map to ensure the uniqueness** of items it stores. 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 we store an item in HashSet, item is stored as *key* in Map and the existing object as the *value***.

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

I will highly suggest you read this post: **How HashMap works?** This post will help you answer all the *HashMap* related questions very comfortably.

10. Can We Add a Null to a TreeSet or HashSet?

As you see, *there is no null check in add() method* in the previous question. And *HashMap* also allows one *null* key, so **HashSet allows one "null" item**.

TreeSet uses the same concept as *HashSet* for internal logic but uses *NavigableMap* for storing the elements.

```
private transient NavigableMap<E,Object> m;

// Dummy value to associate with an Object in the backing Map
private static final Object PRESENT = new Object();
```

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

Map Interface Questions

11. What is a Map? What are Core Classes Implementing Map Interface?

- A *Map* interface is a special type of collection that is **used to store key-value pairs**.
- *Map* **does not extend Collection interface**.
- *Map* interface provides methods to add, remove, search or iterate over various views of Map.

The main classes implementing Map interface are *HashMap*, *Hashtable*, *EnumMap*, *IdentityHashMap*, *LinkedHashMap* and *Properties*.

12. What are IdentityHashMap and WeakHashMap?

IdentityHashMap is similar to HashMap except that **it uses reference equality when comparing elements**.

IdentityHashMap class is not a widely used Map implementation. While this class implements the Map interface, it intentionally violates Map's general contract, which mandates using the *equals()* method when comparing objects. IdentityHashMap is designed only in rare cases wherein reference-equality semantics are required.

WeakHashMap implements 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. This class is primarily used with *key* objects whose *equals()* methods test for object identity using the == operator. Once such a key is discarded, it can never be recreated, so it is impossible to look up that key in a *WeakHashMap* at some later time and be surprised that its entry has been removed.

13. What is ConcurrentHashMap? How Does It Maintain Concurrency?

ConcurrentHashMap is an alternative to *HashMap* class to be safely used in the concurrent environment.

A *ConcurrentHashMap* is a *Hashtable* supporting full concurrency of retrievals and adjustable expected concurrency for updates.

ConcurrentHashMap 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.

ConcurrentHashMap class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details.

> Read More: ConcurrentHashMap Interview Questions.

14. How Hashmap Works?

This question is the **most important** and is most likely to be asked at every job interview level. You must be very clear on this topic., not only because it is the most asked question but also it will open up your mind to further questions related to Collection APIs.

The answer to this question is very large, and you should read my post: How HashMap works? For now, let us remember that **HashMap works on the principle of Hashing**.

A map, by definition is: "An object that maps keys to values". To store such a 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. The whole *Entry* object is stored in an array. The index of the array is calculated based on the hashcode of *Key* object.

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

15. How to Design a Good Key for HashMap?

Another good question that is usually followed up after answering how hashmap works. The most important constraint in a Map is that **we must be able to fetch the '*Value*' object back in the future using the '*Key*' object**. Otherwise, there is no use of having such a data structure.

If we understand the working of *HashMap*, we will find it largely depends on *hashCode() and equals()* methods of the *Key* instances.

- So a good *key object must provide the same hashCode() again and again*, no matter how many times it is fetched.

- Similarly, the *same keys must return true when compared with equals() method* and different keys must return false.

For this reason, **immutable classes are considered the best candidates for HashMap keys.**

> Read more: How to design a good key for HashMap?

16. What are Different Views provided by Map Interface?

Map interface provides 3 views of key-value pairs stored in it:

- *KeySet* view – A Set of all the keys stored in the *Map*.
- *ValueSet* view – A Set of all the values stored in the *Map*.
- *EntrySet* view – A Set of all the key-value pairs stored in the *Map*.

All these views can be navigated using iterators.

17. When to Use HashMap and TreeMap?

***HashMap* stores unordered key-value pairs** and allows to perform many get/put operations on such key-value pairs.

TreeMap is a particular form of *HashMap*. **TreeMap maintains the ordering of keys**. This ordering is by default "natural ordering". We can override the default order by providing an instance of the Comparator class, whose *compare()* method will be used to maintain the ordering of keys.

Please note that all keys inserted into the *TreeMap* must implement the Comparable interface (this is necessary to decide the ordering).

Furthermore, all *TreeMap* keys must be mutually comparable i.e. *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 *TreeMap* that violates this constraint (for example, the user attempts to put a *String* key into a map whose other keys are *Integer* types), the *put()* call will throw a *ClassCastException*.

'Tell the Difference' Questions

18. 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.

19. Difference between List and Map?

A *List* is a collection of elements, whereas a Map is a collection of key-value pairs.

List and Map, both. have separate top-level interfaces, a separate set of generic methods, different supported methods and different views of the Collection.

20. Difference between HashMap and HashTable?

There are several differences between HashMap and Hashtable in Java:

- 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.

21. Difference between Vector and ArrayList?

Let us note down the differences:

- 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.

22. Difference between Iterator and Enumeration?

Iterators differ from enumerations in three ways:

- 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.

23. Difference between HashMap and HashSet?

HashMap is a collection of key-value pairs, whereas HashSet is an unordered collection of unique elements.

That's it. No need to describe further.

24. Difference between Iterator and ListIterator?

There are three Differences are there:

- 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.
- ListIterator allows you modify the list using add() remove() methods. Using Iterator you can not add, only remove the elements.

25. Difference between TreeSet and SortedSet?

SortedSet is an interface that *TreeSet* implements.

TreeSet stores elements that are ordered using their natural ordering.

SortedSet provides a *total ordering* on its elements. The elements are ordered using their natural arrangement, or by a `Comparator` typically provided at sorted set creation time.

26. Difference between ArrayList and LinkedList?

- LinkedList store elements within a doubly-linked list data structure. ArrayList 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.

Expert Level Collections Interview Questions

27. How to Create Read-only Collections?

Use the following methods:

- Collections.unmodifiableList(list);
- Collections.unmodifiableSet(set);
- Collections.unmodifiableMap(map);

These methods take a collection parameter and return a new read-only collection with the same elements as the original Collection.

28. How to Create Thread-safe Collections?

Use the below methods:

- Collections.synchronizedList(list);
- Collections.synchronizedSet(set);
- Collections.synchronizedMap(map);

The above methods take the Collection as a parameter and return the same type of Collection that is **synchronized** and **thread-safe**.

The other solution is to use concurrent collections. For example, *ConcurrentHashMap* or *BlockingQueue*.

29. Why Iterator doesn't have add() Method?

The sole purpose of an *Iterator* is to enumerate through a collection.

All collections contain the *add()* method to serve the purpose. There would be no point in adding to an Iterator because the Collection may or may not be ordered. And ***Iterator.add()* method can not have the same implementation for ordered and unordered collections**.

30. What are Different Ways to Iterate a List?

You can iterate over a list using the following ways:

- Iterator
- For loop
- For-each loop
- While loop
- Stream.forEach()

31. What is a Fail-fast Iterator?

Fail-fast Iterators **fail as soon as they detect that structure of the Collection has been changed** since iteration has begun.

Structural changes mean adding, removing or updating items from the Collection while one thread is Iterating over that Collection.

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

32. Difference between Fail-fast and Fail-safe Iterators?

We have understood the fail-fast iterators in the previous question.

Fail-safe iterators are just the opposite of fail-fast. A **Fail-safe iterator does not fail if we modify the underlying Collection** on which they are iterating because **they work on a clone of Collection instead of the** original Collection and that's why they are called a fail-safe iterator.

*Iterator of CopyOnWriteArrayList is an example of a **fail-safe Iterator***. Also, the iterator returned by *ConcurrentHashMap* keySet is a fail-safe iterator and never throw *ConcurrentModificationException*.

33. How to Avoid ConcurrentModificationException while Iterating a Collection?

We should first **try to find another alternative iterator that is fail-safe**. For example, if you are using *List* and you can use *ListIterator.* If it is a legacy collection, you can use *enumeration*.

If the above solutions 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 the last two approaches will cause a performance hit.

34. What is UnsupportedOperationException?

This exception is thrown **on invoked methods that are not supported by the actual collection type**.

For example, if you make a read-only list using "*Collections.unmodifiableList(list)*" and then call *add()* or *remove()* method, what should happen. It should clearly throw *UnsupportedOperationException*.

35. Which Collections Provide Random Access to the Items?

Index-based List classes and Map classes provide such access.

ArrayList, *HashMap*, *TreeMap*, *Hashtable* classes provide random access to its items.

36. What is BlockingQueue?

A *BlockingQueue* is 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 overrides:

- 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.

> Read more: How to use BlockingQueue?

37. What is Queue and Stack. Tell the Differences?

A Queue is a collection designed for holding elements before 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. Use a queue if you want to process a stream of incoming items in the order received. Good for work lists and handling requests.

A Stack is also a form of Queue, but it is *LIFO (last-in-first-out)*. Use a stack if you want to push and pop from the top of the stack only. Good for recursive algorithms.

Whatever the ordering used, the head of the queue is that element that would be removed by a call to remove() or poll().

Also, note that *Stack and Vector are both synchronized*.

38. What is Comparable and Comparator interfaces?

In Java, all collection classes that have features of automatic sorting use *compare()* methods to ensure the correct sorting of elements. For example,*TreeSet*, *TreeMap* etc.

To support sorting of its instances, a class needs to implement *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 sorting pattern.

The instance of *Comparator* is passed usually as the *sort()* method argument in the supporting collections.

39. What are Collections and Arrays Classes?

Collections and *Arrays* classes are **special utility classes** to support collection framework core classes.

Collections provide utility functions to get read-only/synchronized collections, sort the Collection in various ways etc.

Arrays help to convert arrays into collection objects. Arrays also have some functions which help in copying or working in part of array objects.

40. Recommended Reading List

Read the following posts to understand the Collections framework in-depth.

- Java Collections Framework
- Sorting in Java
- Java Comparator
- Java Comparable
- Java ArrayList
- Java Array
- Java HashMap

Happy Learning !!

Recommended Reading:

1. Java String Interview Questions with Answers
2. Core Java Interview Questions
3. Java HashMap Interview Questions
4. Real Java Interview Questions asked in Oracle
5. Spring Interview Questions with Answers
6. Spring AOP Interview Questions
7. Spring WebMVC Interview Questions
8. Spring Boot Interview Questions
9. Immutable Collections with Factory Methods in Java 9
0. Java Collections sort()

Subscribe

* We do not spam !!

92 thoughts on “Java Collections Interview Questions”

Chaman jain

January 5, 2020 at 2:11 pm

Hi,
In Q 33 you said Use ListIterator to avoid ConcurrentModificationException
But ListListIterator is also fail-fast

Reply

Lokesh Gupta

January 5, 2020 at 8:19 pm

Please refer to Java docs

Reply

AshBhu

July 11, 2019 at 7:04 pm

I used Collections.unmodifiableList(list); and tried to add or remove an element from the list. But it didn't throw me UnsupportedOperationException.

List list = new ArrayList() ;

Reply

Mo Beigi

September 21, 2019 at 6:50 pm

Collections.unmodifiableList(list) returns a list. You are ignoring the result.

Try:

List list = Collections.unmodifiableList(new ArrayList());

Reply

Ganesh

November 26, 2019 at 3:06 pm

```
package arraylist;

import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

public class UnmodifiableList {
        public static void main(String[] args) {
                try {
                        List l1 = new LinkedList();
                        l1.add("A");
                        l1.add("Z");
                        System.out.println("L1: " + l1);
                        List immutablelist = Collections.unmo
                        System.out.println("\nTrying to modif
                        immutablelist.add("B");

                        System.out.println("immutablelist: "
                } catch (UnsupportedOperationException e) {
                        System.out.println(e);
                }

        }
}
```

Reply

sahab

February 22, 2019 at 3:06 pm

you are shareing useful information for me

Reply

Yaeko England

October 6, 2016 at 6:21 pm

Excellent commentary – I Appreciate the insight . Does someone know if my company could get a fillable IRS 943 copy to work with ?

Reply

laltu

August 5, 2016 at 3:02 pm

Hi,
Please chk Q.35, Random Access is only implemented by ArrayList and Vector in whole collection framework….but u mentioned
HashMap, TreeMap, Hashtable . please chk and update…

Thanks
Laltu

Reply

Lokesh Gupta

August 5, 2016 at 3:59 pm

I believe you understood the question incorrectly. It's not about `RandomAccess` interface.

Reply

Himansu

July 21, 2016 at 1:05 am

Difference between ArrayList and LinkedList?

Just to add few things here

On a LinkedList
Finding the point of insertion/deletion is O(n) but Performing the insertion/deletion is O(1)

On a ArrayList its reverse i.e.
Finding the point of insertion/deletion is O(1) but Performing the insertion/deletion is O(n)

Reply

Himansu

July 21, 2016 at 12:36 am

What are different Collection views provided by Map interface?

why is it called “value set view”, It should be “values” as they can be duplicate on the map it cannot be “value set”

Reply

Lokesh Gupta

July 21, 2016 at 1:53 am

Good observation 🙂 I will try to research on it.

Reply

Rajesh

February 7, 2020 at 12:38 am

Map deals with key-value pairs – Values may be duplicate but key for each value must be unique i.e why

Reply

srikanth

February 5, 2016 at 12:23 pm

Thanks. I haven't seen such a pretty good explained blog on collections. Just Awesome.

Reply

sankar balaji

August 10, 2015 at 5:10 pm

Awesome article. Read your post about how HashMap works for too good .

Reply

Lokesh Gupta

August 10, 2015 at 5:51 pm

Thanks. Much appreciated.

Reply

Paranjothi

April 29, 2015 at 6:50 pm

thanks for awsm article. looking forward for more interesting topics

Reply

shishir

April 15, 2015 at 1:43 pm

We can add null as a key to HashMap.

Question is how to retrieve assigned object to this key?

Reply

Lokesh Gupta

April 15, 2015 at 2:11 pm

```
HashMap<String, Integer> map = new HashMap<String, Integer>()
map.put(null, 1000);
System.out.println(map.get(null)); //Prints 1000
```

Reply

Bhaskar suthar

February 2, 2016 at 6:40 am

@Shisher : In case of hashmap only one null key is allowed and that null key is always stored at zero index.

Reply

Krishna

February 8, 2015 at 11:01 am

Hi Lokesh,

This is Krishna.
It is very nice to interact with you.
I have a doubt on very basics. You may think it's silly.
But i'm confusing a little at that.

The point is ArrayList class is inheriting the properties of both List interfase and AbstractList
(As per api
public class ArrayList
extends AbstractList
implements List).
Why both?

I think, Inheriting AbstractList is sufficient.
I have gone through your Interfaces and Abstract Classes article. but i didn't get it.

Thanks & Regards,
Krishna.

Reply

Atul

May 1, 2015 at 11:47 am

Hi Krishna,

Nice Observation!!

ArrayList could have done with either implmenting List or extending AbstractList but that would have prevented polmorphism.

for eg: List list = new ArrayList();
AbstractList list = new ArrayList();

This would not have been possible otherwise.

Thnx

Reply

Raja

December 3, 2015 at 1:19 pm

AbstractList is already implementing List interface. I dont think it is needed.

Reply

Krishna Kumar

April 2, 2016 at 3:45 pm

Nice Observation, I feel by explicitly implementing the List interface ArrayList announces that it is an implementation of List. In case this was not done then it would have been a simple abstraction to the AbstractList and then an extra code dig in would have been needed to know that ArrayList is an implementation of the List interface. Moreover I feel ArrayList provides its own implementation of all the methods declared in the List interface and do not uses the List methods implemented by parent class AbstractList.

Reply

Nikhil Bharne

April 11, 2018 at 5:18 pm

Collection classes contain interface abstract classes and concrete classes so List is a interface extending from AbstractList and it have implementation some more concrete classes like Arraylist linklist vector stack .....

So abstract classes are fully unimplemented classes we can't use it that's why they they are given concrete classes

Hope you understand..

Reply

k c bhardwaj

January 22, 2015 at 4:45 am

public class TreeSet extends AbstractSet
implements NavigableSet, Cloneable, java.io.Serializable

Reply

Pramod Bablad

October 28, 2014 at 6:05 pm

Very nice article… Very much useful Q's & A's…

Reply

Vishwas

July 22, 2014 at 3:36 am

Que no :10
TreeSet allows one null element…you are saying it does not support null value… actully if wetry to add second null vlaue then only it gives null pinter exception….adding one null value works perfectly fine

Reply

Lokesh Gupta

July 22, 2014 at 7:03 am

Run below simple program:

```
public static void main(String[] args)
   {
      TreeSet<String> set = new TreeSet<String>();
      set.add(null);
   }
```

It gives error as : Exception in thread “main” java.lang.NullPointerException
at java.util.TreeMap.compare(Unknown Source)
at java.util.TreeMap.put(Unknown Source)
at java.util.TreeSet.add(Unknown Source)

I have made bold the TreeMap.compare() which is internally used inside TreeSet for maintaining the ordering. Null simply do not fit in any sorting order, so they are restricted.

Reply

irshad kashmiri

June 30, 2016 at 3:37 am

upto java6 we were able to insert one null value but java7 onwards we can not insert any null value.

Reply

AMRENDRA

November 19, 2014 at 3:46 pm

Tree map doesn't allow any null value as key whereas hash map allow one null value as key because tree map use sorted set to maintain order of key

Reply

Anand Jain

June 28, 2014 at 1:11 pm

Hi Lokesh,

I really like your posts.I want you to post one more interview question with answer which i faced recently in an interview :

How Collections.sort works? means which sorting algorithm does it implement?

A

Reply

Lokesh Gupta

June 28, 2014 at 1:12 pm

Thanks for the suggestion. I will cover it in separate post.

Reply

lavish

May 21, 2014 at 4:30 pm

Hi Lokesh,

Can you send me the latest interview question related to String/String Buffer.

Reply

Ban Ăn Chơi

April 2, 2014 at 8:58 am

Thanks, it helps me a lot.

Reply

Raja

March 12, 2014 at 3:00 am

Hi Gupta,

I am afraid there is a correction needed at 7th question. I guess, sorting list in reverse order and reversing the list order are different.

reverse() will just reverse the order. It will not sort.

The answer for this question should be,

Collections.sort(list, Collections.reverseOrder());

If you have meant only to reverse the order, you have mislead me 😉 Let me know what you think.

Thanks for the post! It is really good. Keep going!

– Raja

Reply

Lokesh Gupta

March 12, 2014 at 6:07 am

you are absolutely right. I will update the post.

Reply

mayuri

March 9, 2014 at 11:49 am

nice Lokesh..keep it up….really helpful blog….

Reply

Surjeet

February 10, 2014 at 7:07 pm

Hi Lokesh i solved above problem, running perfectly!

```
package com.intuit;

class Shared{
private boolean done=false;
protected int[] even={2,4,6,8,10};
protected int[] odd={1,3,5,7,9};
private static int i=0;
private static int j=0;
public synchronized void printOdd(){
while(done){
try{
wait();
}catch(InterruptedException ie){

}
}
if(j!=odd.length)
System.out.println(odd[j++]);
done=true;
notify();
}
public synchronized void printEven(){
while(!done){
try{
wait();
}catch(InterruptedException ie){

}
}
if(i!=even.length)
System.out.println(even[i++]);
```

```
done=false;
notify();
}
}
class OddThread implements Runnable{
private Shared shared;
public OddThread(Shared shared){
this.shared=shared;
}
public void run(){
for(int i=0;i<=shared.odd.length;i++){
shared.printOdd();
}
}
}
class EvenThread implements Runnable{
private Shared shared;
public EvenThread(Shared shared){
this.shared=shared;
}
public void run(){
for(int i=0;i<shared.even.length;i++){
shared.printEven();
}
}
}
public class OddEvenThread {

public static void main(String[] args) {
Shared shared = new Shared();
Thread t1 = new Thread(new OddThread(shared));
Thread t2 = new Thread(new EvenThread(shared));
t1.start();
t2.start();
```

```
try{
Thread.sleep(1000);
}catch(InterruptedException ie){

}
}
}
```

Reply

Lokesh Gupta

February 10, 2014 at 7:27 pm

Thanks for sharing.

Reply

Surjeet

February 14, 2014 at 8:47 am

Hi , In which scenario we use Singly Linked list data structure and double linked list? Under what circumstance Singly linked list data structure will be better then double linked list?

Reply

Lokesh Gupta

February 14, 2014 at 4:23 pm

A real good question. Surjeet, I never came across any situation where I was forced to use (or even suggested to use) doubly linked list. In my whole career, I always managed to do my work with simple data structures like stack, linked list and maps.
I will really like to get some good answers from all those who visited and read this thread.

Reply

Surjeet

February 10, 2014 at 5:08 am

Hi Lokesh, recently i hv faced 1 interview in Intuit company, they asked "There are two threads, first thread contains even number(2,4,6,8,10) and second thread contains odd number(1,3,5,7,9), and i want to print in natural order or sequence like 1,2,3,4,5,6,7,8,9,10" , i answered we could use wait and notify method, after first thread put odd number and will notify to second thread that will print even number, and it will continue to till it both threads prints in natural order. but he was not satisfied, let me know if u could provide solution

Reply

Lokesh Gupta

February 10, 2014 at 8:37 am

You answered the question correctly. Either you didn't explain it well or interviewer has already any specific answer in his mind and it wanted you to read his mind. Nobody can satisfy the interviewer in second case. Leave it considering a bad day for you.

More discussion can be found here:
https://stackoverflow.com/questions/18799591/print-natural-sequence-with-help-of-2-threads1-is-printing-even-and-2nd-is-pri

Reply

Surjeet

February 13, 2014 at 6:34 pm

Another Question i hv faced::::

Say there are 3 array lists l1, l2 & l3 of same length.
Three threads accessing three lists. Say T1 -> l1, T2 ->l2 & T3 ->l3.
It should print in the order say first element of 1st then first element of 2nd list and
then first element of 3rd list.
Then second element of 1st then second element of 2nd list and then second element of 3rd list?

well i knw here i hv to use wait and notify…How to approach?
any suggestion

Reply

Lokesh Gupta

February 14, 2014 at 6:53 pm

A good solution can be using Semaphore. Read more about Semaphore here:

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html

A working example is below:

```
package com.howtodoinjava.demo.rest;

import java.util.Arrays;
import java.util.List;
import java.util.concurrent.Semaphore;

public class PrintList {
private List list1 = Arrays.asList(new Integer[]
{1,2,3,4,5});
private List list2 = Arrays.asList(new Integer[]
{1,2,3,4,5});
private List list3 = Arrays.asList(new Integer[]
{1,2,3,4,5});
private int listSize = 5;

private Semaphore semaphore = null;

public PrintList() {
semaphore = new Semaphore(3);
}

public void print() throws InterruptedException{
semaphore.acquire();
for(int i = 0;i
```

Reply

Surjeet

February 14, 2014 at 9:18 pm

```
semaphore.acquire();
for(int i = 0;i<listSize;i++){
System.out.println(list1.get(i));
System.out.println(list2.get(i));
System.out.println(list3.get(i));
}
semaphore.release();
```

But here the above code:each of the thread is accessing three list???

Venkat J

March 26, 2017 at 7:47 am

Hi Surjeet,

Thanks for sharing your interview experience.:)

venkata

March 16, 2014 at 5:45 am

Hi Surjeet, Could you please share the full code in both the approaches like using wait() and notify() methods and also using Semaphore.

Reply

Raja

December 3, 2015 at 1:30 pm

You could have mentioned the usage of boolean to tell which thread's turn to wait and notify

Reply

Sangeeta

February 9, 2014 at 12:36 pm

Hi Lokesh,
For reversing the list you have suggested to use:

List reversedList = Collections.reverse(list);

As return type of Collections.reverse(list) is a void we will not be able to compile the above piece of code.

Reply

Lokesh Gupta

February 9, 2014 at 4:05 pm

OOPS !! It is typo mistake. Corrected it.

Reply

Dinesh Reddy

January 21, 2014 at 2:18 am

Hi Lokesh..

good work.. Thanks

Reply

Manish

January 11, 2014 at 9:17 pm

Great Work Lokesh 🙂
Keep it up!

Reply

pankajmalviya04

December 15, 2013 at 9:47 am

Hi Lokesh,

Is it correct?
18) Difference between Set and List?

3rd point states that “List does not prevent inserting null elements (as many you like), but Set will allow only one null element.”

TreeSet does not allow to add one null element.
It throws NullPointerException.

Sample Code:
Set s1 = new TreeSet();
s1.add(null);

Output:
Exception in thread “main” java.lang.NullPointerException
at java.util.TreeMap.compare(Unknown Source)
at java.util.TreeMap.put(Unknown Source)
at java.util.TreeSet.add(Unknown Source)

Reply

Lokesh Gupta
December 15, 2013 at 11:51 am

I believe Yes.

Reply

shishir
December 11, 2013 at 10:29 am

Good collection of Collection!!!! 🙂

Reply

Lokesh Gupta

December 11, 2013 at 10:39 am

🙂

Reply

Surjeet Mohanty

December 10, 2013 at 11:28 am

Hi Lokesh, i hv one doubt, why Hashmap allows null key, whats the main purpose and its benefit

Reply

Lokesh Gupta

December 10, 2013 at 10:51 pm

Really a very good question. I can't think of any (good) reason for wanting to store null as a key, and in general I will advise against using null as a key. Taking from JDK 1.2 Java Collections API Change Summary:

> **Added null-key support to HashMap**. This was done for consistency with TreeMap and the late, unlamented ArrayMap, and because

> customers requested it. Now all of our general-purpose collection implementations accept null keys, values and elements.

Later Doug Lea also agreed that it is not a good design.
http://cs.oswego.edu/pipermail/concurrency-interest/2006-May/002485.html

Further digressing: I personally think that allowing nulls in Maps (also Sets) is an open invitation for programs to contain errors that remain undetected until they break at just the wrong time.

Reply

H Singh

December 7, 2013 at 12:21 pm

Hi Lokesh,

How I can get the bucket location with highest number of Map.Entry objects?

Thanks in Advance.

Reply

Lokesh Gupta

December 7, 2013 at 10:13 pm

I think HashMap does not provide any such way. But, you can extend the HashMap and override relevant methods and maintain counters for added

entries.

Reply

H Singh

December 8, 2013 at 12:22 am

I want to get the highest occurring entry data element. I want to store my data in HashMap so that I can get the element with maximum occurrence.

Can you suggest some alternative approach.

Thanks in Advance.

Reply

Lokesh Gupta

December 8, 2013 at 11:04 am

You need to extend HashMap class and add modified behavior.

Reply

Snehal Masne

November 30, 2013 at 6:26 pm

Excellent collection on Collection(s).
Keep it up ! 🙂

Reply

nrvmodi

November 20, 2013 at 8:33 pm

I have One Question related to List :
How to remove particular value from the List ?
Suppose I have List and it contain [0,1,2,3,4,5] then i want to remove 1 value from the list ?
How it possible

Reply

Lokesh Gupta

November 20, 2013 at 11:05 pm

You have **remove()** method. What's problem in this?

Reply

Naresh

November 5, 2013 at 4:09 pm

Hi Lokesh,
Does the JDBC-ODBC Bridge support multiple concurrent open statements per connection?
Explain me with an example

Reply

Venki

October 24, 2013 at 5:33 pm

Hi Lokesh,
how can i implement this example
class Contact{
String name;
String email;
String mobile no;
}
static HashMap convertToMap(List list{
//how to write logic in here
}

Thank's in advance

Reply

Lokesh Gupta

October 25, 2013 at 11:14 am

I would like to encourage you to write this program. Let me know (with code) what stops you.

Reply

Venki

October 26, 2013 at 7:44 pm

I can't understand this code,i have no idea,please help me,

Reply

Venki

October 27, 2013 at 9:28 am

Hi Lokesh,
Thanks for encourage me, i tried this program but this is correct or not

```
package com.pvr;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Contact {
private int id;
private String name;
private String email;
private String address;
```

```
public int getId() {
return id;
}

public void setId(int id) {
this.id = id;
}

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public String getEmail() {
return email;
}

public void setEmail(String email) {
this.email = email;
}

public String getAddress() {
return address;
}

public void setAddress(String address) {
this.address = address;
}

public Contact() {
}
public Contact(int id,String name,String email,String address){
```

this.id=id;
this.name=name;
this.email=email;
this.address=address;
}
static HashMap convertToMap(List list){
Map map=new HashMap();
for(Contact c:list){
map.put(c.toString(),c);
}
return (HashMap) map;

}
public static void main(String[] args){
List list=new ArrayList();
list.add(new Contact(101,"pvr", "pvr@email.com","gnt"));
list.add(new Contact(102,"ven","ven@gmail.com","hyd"));
HashMap m=new Contact().convertToMap(list);
System.out.println(m);
}
}

Reply

Lokesh Gupta

October 27, 2013 at 12:43 pm

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

```
public class Contact
{
private int id;
private String name;
private String email;
private String address;

public int getId() {
return id;
}

public void setId(int id) {
this.id = id;
}

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public String getEmail() {
return email;
}

public void setEmail(String email) {
this.email = email;
}

public String getAddress() {
return address;
}
```

```
public void setAddress(String address) {
this.address = address;
}

@Override
public String toString() {
return "Contact#" + this.getId();
}
public Contact(int id, String name, String email, String
address) {
this.id = id;
this.name = name;
this.email = email;
this.address = address;
}

private static Map convertToMap(List list) {
Map map = new HashMap();
for (Contact c : list) {
map.put(c.getId(), c);
}
return map;
}

public static void main(String[] args){
List list=new ArrayList<>();
list.add(new Contact(101,"pvr", "pvr@email.com","gnt"));
list.add(new Contact(102,"ven","ven@gmail.com","hyd"));
Map map = convertToMap(list);
System.out.println(map);

System.out.println(map.get(101));
System.out.println(map.get(102));
```

```
    }
}
```

Ramesh

October 15, 2013 at 12:05 pm

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 Iterator.

Just noticed something in the above statement and would like to clarify, you meant Enumerator in the second statement?

Reply

Lokesh Gupta

October 15, 2013 at 11:18 pm

Ramesh, Can you please be more specific about context? Not sure where you are pointing to with “second statement”?

Reply

Debarshi

October 18, 2013 at 5:23 pm

What Ramesh is saying, instead of this line – “You can not add/remove elements from a collection when using Iterator”, it should be “You can not add/remove elements from a collection when using Enumerator”.

Nice article by the way..keep it up..

Reply

Lokesh Gupta

October 18, 2013 at 10:27 pm

OK, but it’s already had been corrected. Ramesh and Debarshi, Thanks for clearing up.

Reply

venkata

March 16, 2014 at 5:54 am

Hi Lokesh,

While using the iterator, it is allowing to remove but not add the elements. As per my understanding , iterator is used for just retrieving the elements only.

My question is, why iterator allows remove the elements? what was the reason behind that?

Reply

Lokesh Gupta

March 16, 2014 at 6:21 am

iterators ae meant to iterate, thats why once the collection is modified mid iteration, iterator throws exception.

Reply

Manish

October 13, 2013 at 11:18 pm

Question 24
ListIterator allows you modify the list using add() remove() methods. Using Iterator you can not do this.

-using iterator we can remove but can't add.

-Manish

Reply

Lokesh Gupta

October 13, 2013 at 11:22 pm

Fair enough. I will make the update.

Reply

manu

August 22, 2013 at 11:06 am

Question 23.- has Hashset as ordered but they are unordered.

Reply

Babu

July 30, 2013 at 1:53 am

Treset allows null only once.
ex:
Set s = new TreeSet();
s.add(null);
s.add(null); // throws NULL pointer exception

Reply

subbareddy

July 11, 2013 at 10:58 am

hi lokesh,
i have probelm actually my boss said to me to create a website in java that is generally like a community block.

but i dont know how to implement that and which technogies is best just tell me.

Reply

Lokesh Gupta

July 15, 2013 at 12:46 pm

Sorry for late reply as I was on vacation in last week.

I believe its not technology what matters. Today, you can build any website with any set of technologies. The thing which matter, is what you want to build.

If I have been in your place, I will first understand the requirements (present as well as future). Then write a good and flexible design and get it reviewed with some brilliant mind and experienced colleagues.

Finally, implement it in any good MVC framework like struts or spring.

Reply

Siddu

July 11, 2013 at 7:03 am

TreeSet Wont support null as value.

Reply

Peter Birk Nielsen

July 9, 2013 at 6:59 pm

Hi. Spotted a minor error in question 7 about the Collections.reverse method. That method mutates the argument list, and does not return any value, so your example won't compile.

Reply

sunny

July 9, 2013 at 4:49 pm

TreeSet supports null. But a single entry should be there for multiple only it will throw NPE

Reply

Lokesh Gupta

July 9, 2013 at 5:11 pm

Hey sunny, thanks for comment. But I am really unable to gt what you are pointing to.
Can you please give any code sample, which will make things more clear??

Reply

Shankar Morwal

September 21, 2013 at 2:13 am

I guess he meant, that we can insert null in treeset too if we have 1 or more then one element in tree set. Tree set throw null pointer exception when first element added is null.
You said that tree set doesn't allow nulls.

By the way nice blog. i regularly follow your blog. I am fond of Java and wordpress.
My blog https://www.examsbook.com/

Reply

Lokesh Gupta

September 21, 2013 at 11:37 am

Hey Shankar, your site also looks good buddy. good going.

Reply

Shankar Morwal

September 23, 2013 at 9:20 pm

I more into Java. this is just hobby work.. 🙂

Leave a Comment

Name *

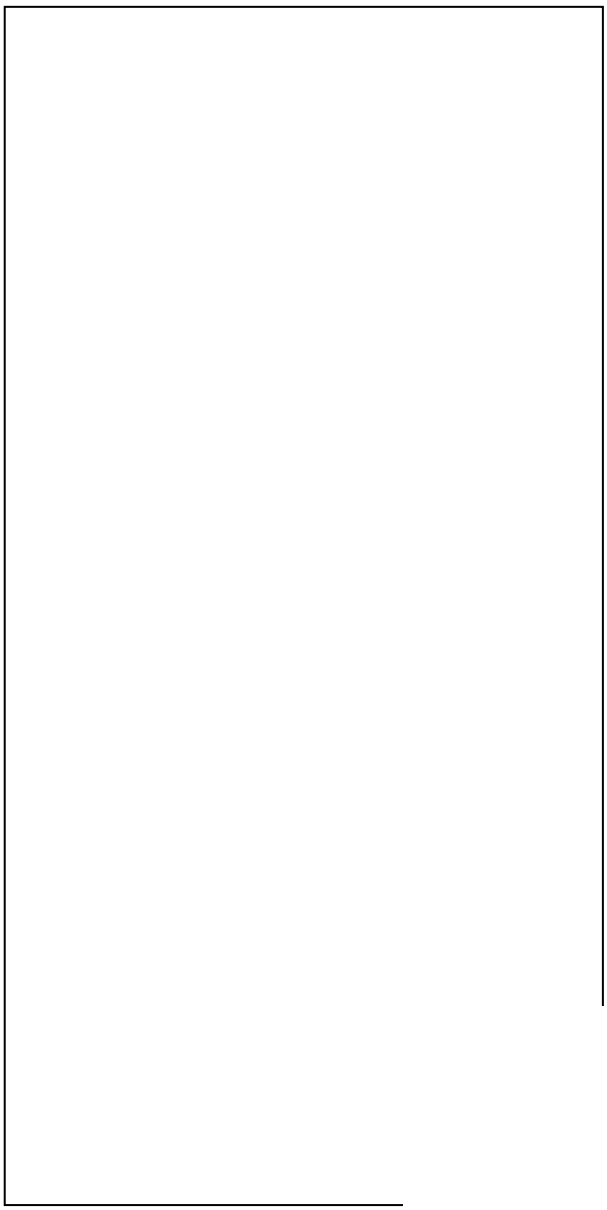
Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...



HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial