Top 70 Java Collections Interview Questions With Answers

(HTTPS://JAVACONCEPTOFTHEDAY.COM/AUTHOR/PRAMODBABLAD/) PRAMODBABLAD (HTTPS://JAVACONCEPTOFTHEDAY.COM/AUTHOR/PRAMODBABLAD/) /
JUNE 24, 2022 /
COLLECTION FRAMEWORK (HTTPS://JAVACONCEPTOFTHEDAY.COM/CATEGORY/JAVA-COLLECTION-FRAMEWORK-TUTORIAL/), JAVA INTERVIEW QUESTIONS (HTTPS://JAVACONCEPTOFTHEDAY.COM/CATEGORY/JAVA-INTERVIEW-QUESTIONS/)

1) What is the Java Collection Framework? Why it is introduced?

Java Collection Framework is a centralized and unified theme to store and manipulate the group of objects. Java Collection Framework provides some predefined classes and interfaces to handle the group of objects. Using this collection framework, you can store the objects as a list or as a set or as a queue or as a map and perform operations like adding or removing or retrieving the objects without much hard work.

Java Collection Framework or simply collections are nothing but the group of objects stored in well defined manner. Earlier, arrays are used to store these group of objects. But, arrays are not re-sizable. They are of fixed size. Size of the arrays can not be changed once they are defined. This causes lots of problem while handling the group of objects. To overcome this drawback of arrays, Java collection framework is introduced in Java from JDK 1.2.

Although, there were classes like `Dictionary`, `Vector`, `Stack` and `Properties` which handle group of objects better than the arrays. But, each of them handle the objects differently. The way you use `Dictionary` class is totally different from the way you use `Stack` class and the way you use `Vector` class is different from the way you use `Properties` class. Hence, there needed a central and unifying theme to handle the group of objects. The collection framework is the answer to that.

2) What is the root level interface of the Java collection framework?

`java.util.Collection` is the root level interface of the Java collection framework.

3) What are the four main core interfaces of the Java collection framework?

The whole Java collection framework is divided into four interfaces – `List` , `Queue` , `Set` and `Map` . In which all except `Map` are inherited from `java.util.Collection` interface.

List : It handles the sequential list of objects. `ArrayList` , `Vector` and `LinkedList` are the major implementation of this interface.

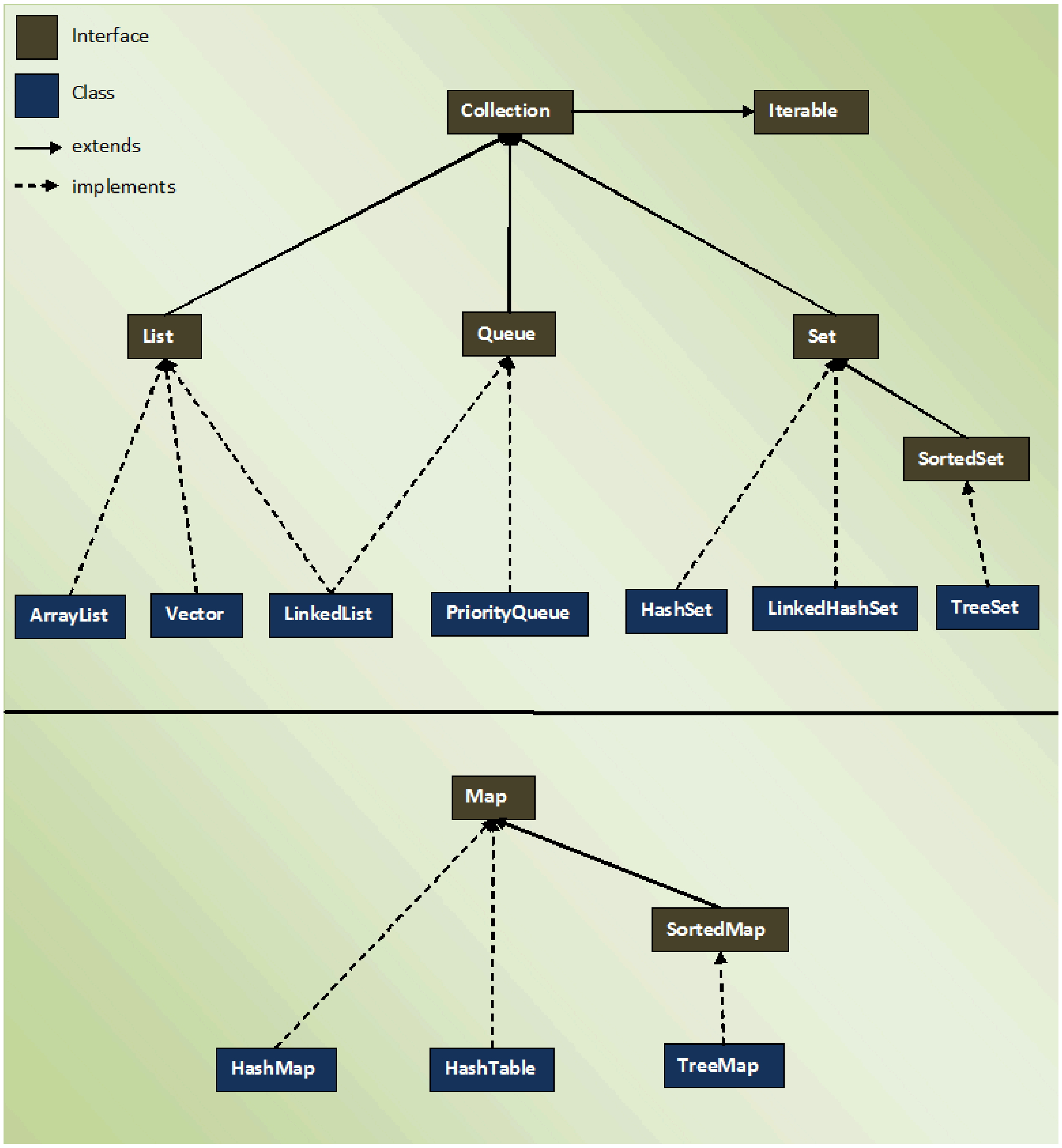
Queue : It handles the special group of objects in which elements are added from one end and removed from another end. `LinkedList` and `PriorityQueue` classes implement this interface.

Set : It handles the group of objects which must contain only unique elements. The major implementations of this interface are `HashSet` , `LinkedHashSet` and `TreeSet` .

Map : This is the one interface in Java Collection Framework which is not inherited from `Collection` interface. It handles the group of objects as key-value pairs. It is implemented by `HashMap` , `LinkedHashMap` and `TreeMap` .

4) Explain the class hierarchy of Java collection framework?

Below diagram shows the class hierarchy of collection framework.



(https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2014/11/CollectionHierarchy.png?ssl=1)

5) Why Map is not inherited from Collection interface although it is a part of Java collection framework?

Map is a collection of key-value pairs where as other collection types like List, Set and Queue are the collection of values. Collection interface has the methods which support only the collection of values but not the collection of key-value pairs. That's why Map doesn't inherit Collection interface.

6) What is Iterable interface?

`Iterable` interface is a member of `java.lang` package which is extended by `java.util.Collection` interface which is nothing but the root level interface of the Java collection framework. `Iterable` interface has only one method called `iterator()` which returns an `Iterator` object, using that object you can iterate over the elements of Collection. (`forEach()` and `spliterator()` methods are added to this interface from Java 8). That means these methods will be available in all collection types which are inherited from `Collection` interface.

7) What are the characteristics of List?

- List Interface represents an ordered or sequential collection of objects.
- Elements of the lists are ordered using Zero based index.
- Elements of the lists can be randomly accessed. i.e elements can be inserted at or removed from or retrieved from a specific position using integer index.
- A list may contain duplicate elements.
- A list may have multiple null elements.

8) What are the major implementations of List interface?

- ArrayList
- Vector
- LinkedList

9) What are the characteristics of ArrayList?

- Size of the ArrayList is not fixed. It can increase and decrease dynamically as we add or delete the elements.
- Elements are placed according to Zero-based index. That means, first element will be placed at index 0 and last element at index n-1, where 'n' is the size of the ArrayList.
- ArrayList can have any number of null elements.
- ArrayList can have duplicate elements.
- As ArrayList implements `RandomAccess` , you can get, set, insert and remove elements of the ArrayList from any arbitrary position.
- ArrayList is not synchronized. That means, multiple threads can use same ArrayList simultaneously.

10) What are the three marker interfaces implemented by ArrayList?

RandomAccess, Cloneable and Serializable.

11) What is the default initial capacity of ArrayList?

Default initial capacity of an ArrayList is 10. This capacity increases automatically as we add more elements to ArrayList. You can also specify initial capacity of an ArrayList while creating it.

12) What is the main drawback of ArrayList?

When you insert an element in the middle of the ArrayList, the elements at the right side of that position are shifted one position right and when you delete an element, they will be shifted one position left. This feature of the ArrayList causes some performance issues as shifting of elements is time consuming if ArrayList has lots of elements.

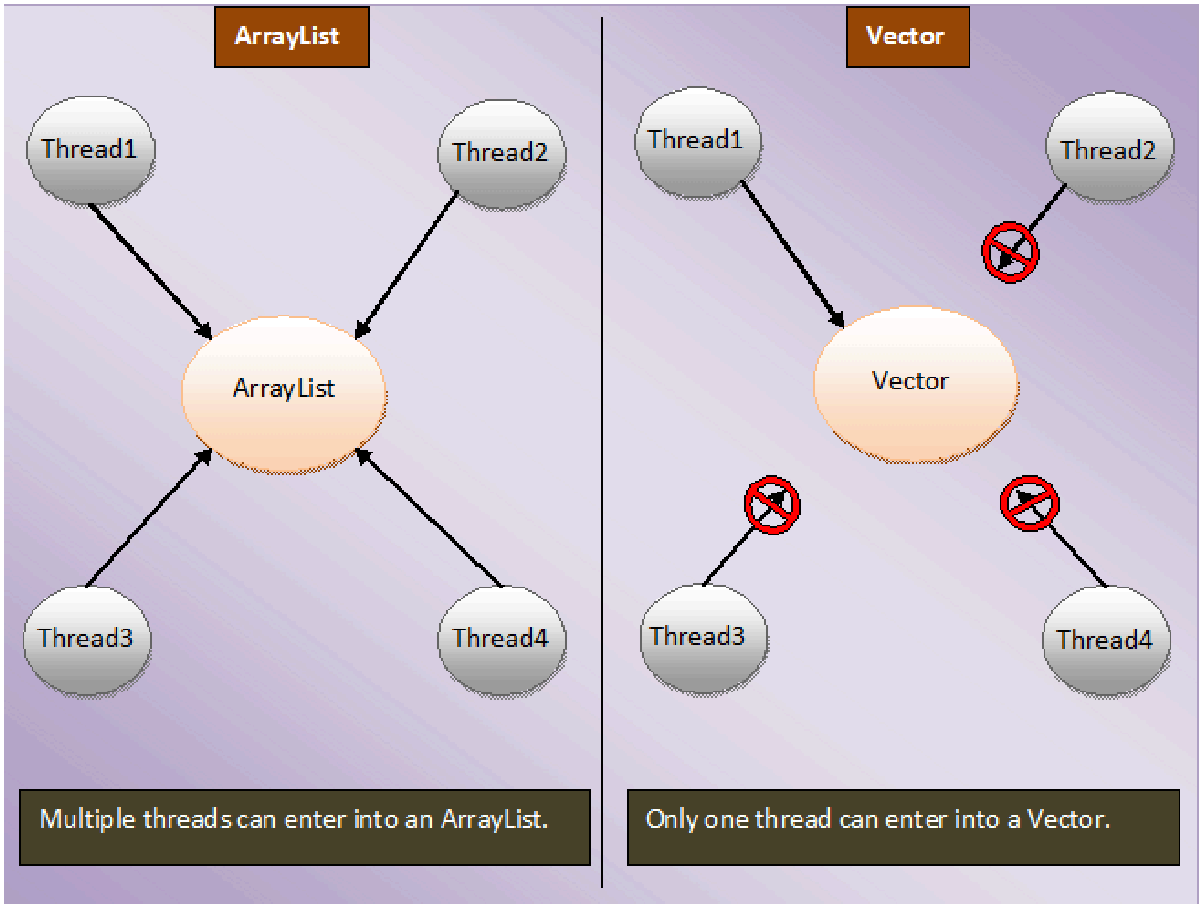
13) What are the differences between array and ArrayList?

Array	**ArrayList**
Arrays are static in nature. Arrays are fixed length data structures. You can't change their size once they are created.	ArrayList is dynamic in nature. Its size is automatically increased if you add elements beyond its capacity.
Arrays can hold both primitives as well as objects.	ArrayList can hold only objects.
Arrays can be iterated only through *for* loop or *for-each* loop.	ArrayList provides iterators to iterate through their elements.
The size of an array is checked using *length* attribute.	The size of an ArrayList can be checked using *size()* method.
Array gives constant time performance for both add and get operations.	ArrayList also gives constant time performance for both add and get operations provided adding an element doesn't trigger resize.
Arrays don't support generics.	ArrayList supports generics.
Arrays are not type safe.	ArrayList are type safe.
Arrays can be multi-dimensional.	ArrayList can't be multi-dimensional.
Elements are added using assignment operator.	Elements are added using add() method.

See More : Array Vs ArrayList (https://javaconceptoftheday.com/differences-between-array-vs-arraylist-in-java/)

14) How Vector is different from ArrayList?

The Vector Class is also dynamically grow-able and shrink-able collection of objects like an ArrayList class. But, the main difference between ArrayList and Vector is that Vector class is synchronized. That means, Vector is thread safe. only one thread can enter into vector object at any moment of time.



(https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2014/12/ArrayListVsVector.png?ssl=1)

15) Why it is recommended not to use Vector class in your code?

Vector class is preferred over ArrayList class when you are developing a multi threaded application. But, precautions need to be taken because vector may reduce the performance of your application as it is thread safe and only one thread is allowed to have object lock at any moment of time and remaining threads have to wait until a thread releases the object lock. So, it is always recommended that if you don't need thread safe environment, it is better to use ArrayList class than the Vector class.

And also Vector class is often considered as obsolete or "Due for Deprecation" by many experienced Java developers. They always recommend and advise not to use Vector class in your code. They prefer using ArrayList over Vector class.

16) What are the differences between ArrayList and Vector?

ArrayList	**Vector**
ArrayList is not thread safe.	Vector is thread safe.
As ArrayList is not synchronized, it gives better performance than Vector.	As Vector is synchronized, it is slightly slower than ArrayList.

ArrayList is not a legacy code.	Vector class is considered as legacy, due for deprecation.

See More : ArrayList Vs Vector (https://javaconceptoftheday.com/difference-between-arraylist-and-vector-class/)

17) What are the characteristics of Queue?

- Queue is a data structure in which elements are added from one end called tail and removed from another end called head.
- Queue is first-in-first-out type of data structure. That means an element which is inserted first will be the first element to be removed from the queue.
- null elements are not allowed in the queue.
- Queue can have duplicate elements.
- Queue is not random access. i.e you can’t set or insert or get elements at an arbitrary positions.

18) Mention the important methods of Queue?

Operation	Throws An Exception If operation is not possible	Returns null or false if operation is not possible
Add an element to the queue.	add()	offer()
Retrieve an element from the head of the queue.	element()	peek()
Retrieve And Remove an element from the head of the queue.	remove()	poll()

19) How Queue differs from List?

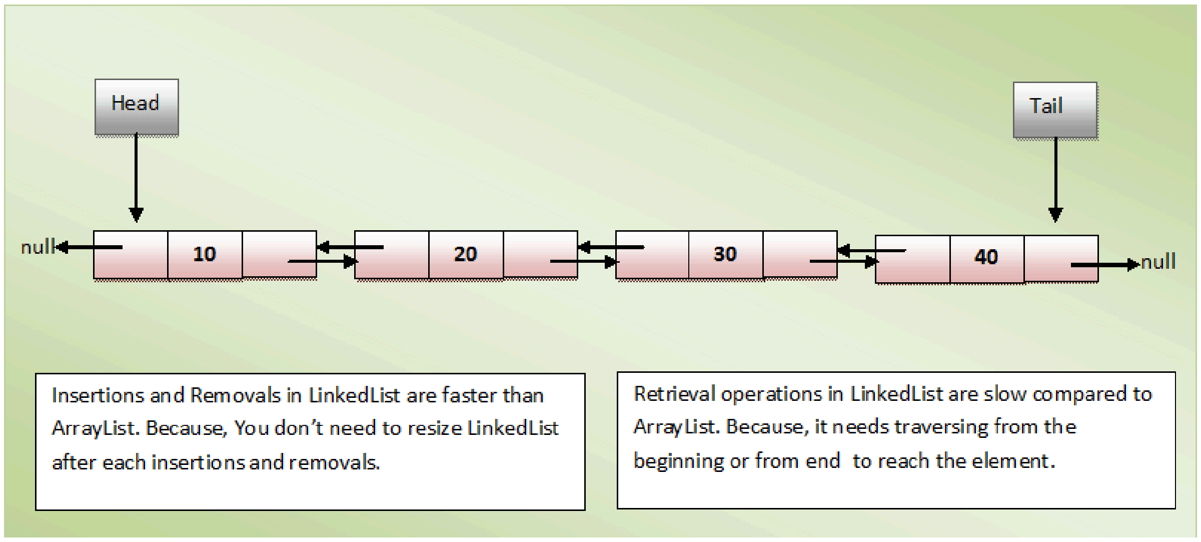
List	Queue
Random Access. i.e you can set, get, add or remove elements from an arbitrary position.	No random access. i.e you can’t set, get, add or remove elements from an arbitrary position.
Can have null elements.	No null elements.
It is an ordered collection of objects where elements are added or removed or retrieved randomly using an integer based index.	It is also an ordered collection of objects where elements are added from one end and removed or retrieved from another end.

20) Which popular collection type implements both List and Queue?

LinkedList

21) What are the Characteristics of LinkedList?

- Elements in the LinkedList are called as Nodes. Where each node consist of three parts – Reference To Previous Element, Value Of The Element and Reference To Next Element. Below diagram shows how LinkedList looks like.


Head
Tail
null
10
20
30
40
null
Insertions and Removals in LinkedList are faster than ArrayList. Because, You don't need to resize LinkedList after each insertions and removals.
Retrieval operations in LinkedList are slow compared to ArrayList. Because, it needs traversing from the beginning or from end to reach the element.

(https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2014/12/HowLinkedListWorks.png?ssl=1)

- Reference To Previous Element of first node and Reference To Next Element of last node are null as there will be no elements before the first node and after the last node.
- You can add or remove or retrieve the elements at both the ends and also in the middle of the LinkedList.
- Insertion and removal operations in LinkedList are faster than the ArrayList. Because in LinkedList, there is no need to shift the elements after each insertion and removal. only references of next and previous elements need to be changed.
- Retrieval of the elements is very slow in LinkedList as compared to ArrayList. Because in LinkedList, you have to traverse from beginning or end (whichever is closer to the element) to reach the element.
- The LinkedList can be used as stack. It has the methods pop() and push() which make it to function as Stack.
- The LinkedList can also be used as ArrayList, Queue, Single linked list and doubly linked list.
- LinkedList can have multiple null elements.
- LinkedList can have duplicate elements.
- LinkedList class in Java is not of type Random Access. i.e the elements can not be accessed randomly. To access the given element, you have to traverse the LinkedList from beginning or from end (whichever is closer to the element) to reach the given element.

22) What are the differences between ArrayList and LinkedList?

ArrayList	**LinkedList**
ArrayList is an index based data structure where each element is associated with an index.	Elements in the LinkedList are called as nodes, where each node consists of three things – Reference to previous element, Actual value of the element and Reference to next element.
Insertions and Removals in the middle of the ArrayList are very slow. Because after each insertion and removal, elements need to be shifted.	Insertions and Removals from any position in the LinkedList are faster than the ArrayList. Because there is no need to shift the elements after every insertion and removal. Only references of previous and next elements are to be changed.
Insertion and removal operations in ArrayList are of order O(n).	Insertion and removal in LinkedList are of order O(1).
Retrieval of elements in the ArrayList is faster than the LinkedList . Because all elements in ArrayList are index based.	Retrieval of elements in LinkedList is very slow compared to ArrayList. Because to retrieve an element, you have to traverse from beginning or end (Whichever is closer to that element) to reach that element.
Retrieval operation in ArrayList is of order of O(1).	Retrieval operation in LinkedList is of order of O(n).
ArrayList is of type Random Access. i.e elements can be accessed randomly.	LinkedList is not of type Random Access. i.e elements can not be accessed randomly. you have to traverse from beginning or end to reach a particular element.
ArrayList can not be used as a Stack or Queue.	LinkedList, once defined, can be used as ArrayList, Stack, Queue, Singly Linked List and Doubly Linked List.
ArrayList requires less memory compared to LinkedList. Because ArrayList holds only actual data and it's index.	LinkedList requires more memory compared to ArrayList. Because, each node in LinkedList holds data and reference to next and previous elements.
If your application does more retrieval than the insertions and deletions, then use ArrayList.	If your application does more insertions and deletions than the retrieval, then use LinkedList.

See More : ArrayList Vs LinkedList (https://javaconceptoftheday.com/arraylist-vs-linkedlist-java/)

23) What is the PriorityQueue?

PriorityQueue is a class in Java collection framework which implements Queue interface.

The PriorityQueue is a queue in which elements are ordered according to specified Comparator. You have to specify this Comparator while creating a PriorityQueue itself. If no Comparator is specified, elements will be placed in their natural order.

The PriorityQueue is a special type of queue because it is not a First-In-First-Out (FIFO) as in the normal queues. But, elements are placed according to supplied Comaparator.

The PriorityQueue does not allow null elements. Elements in the PriorityQueue must be of Comparable type, If you insert the elements which are not Comparable, you will get ClassCastException at run time.

The head element of the PriorityQueue is always the least element and tail element is always the largest element according to specified Comparator.

24) What are Deque and ArrayDeque? When they are introduced in Java?

Deque is an interface which extends the Queue interface and ArrayDeque is the class which implements Deque interface. Both are introduced from Java 6.

The Deque is the short name for “Double Ended Queue“. As the name suggest, Deque is a linear collection of objects which supports insertion and removal of elements from both the ends. The Deque interface defines the methods needed to insert, retrieve and remove the elements from both the ends.

The main advantage of Deque is that you can use it as both **Queue** (FIFO) as well as **Stack** (LIFO). The Deque interface has all those methods required for FIFO and LIFO operations. ArrayDeque class provides implementations for all these methods.

25) What are the characteristics of sets?

- Set contains only unique elements. It does not allow duplicates.
- Set can have maximum one null element.
- Random access of elements is not possible.
- Order of elements in a set is implementation dependent. HashSet maintains no order. TreeSet elements are ordered according to supplied Comparator (If no Comparator is supplied, elements will be placed in their natural order) and LinkedHashSet maintains insertion order.
- Set interface contains only methods inherited from Collection interface. It does not have it’s own methods. But, applies restriction on methods so that duplicate elements are always avoided.
- One more good thing about Set interface is that the stronger contract between equals() and hashCode() methods. According to this contract, you can compare two Set instances of different implementation types (HashSet, TreeSet and LinkedHashSet).
- Two set instances, irrespective of their implementation types, are said to be equal if they contain same elements.

26) What are the major implementations of Set interface?

There are three major implementations of Set interface.

- HashSet
- LinkedHashSet
- TreeSet

27) What are the differences between List and Set?

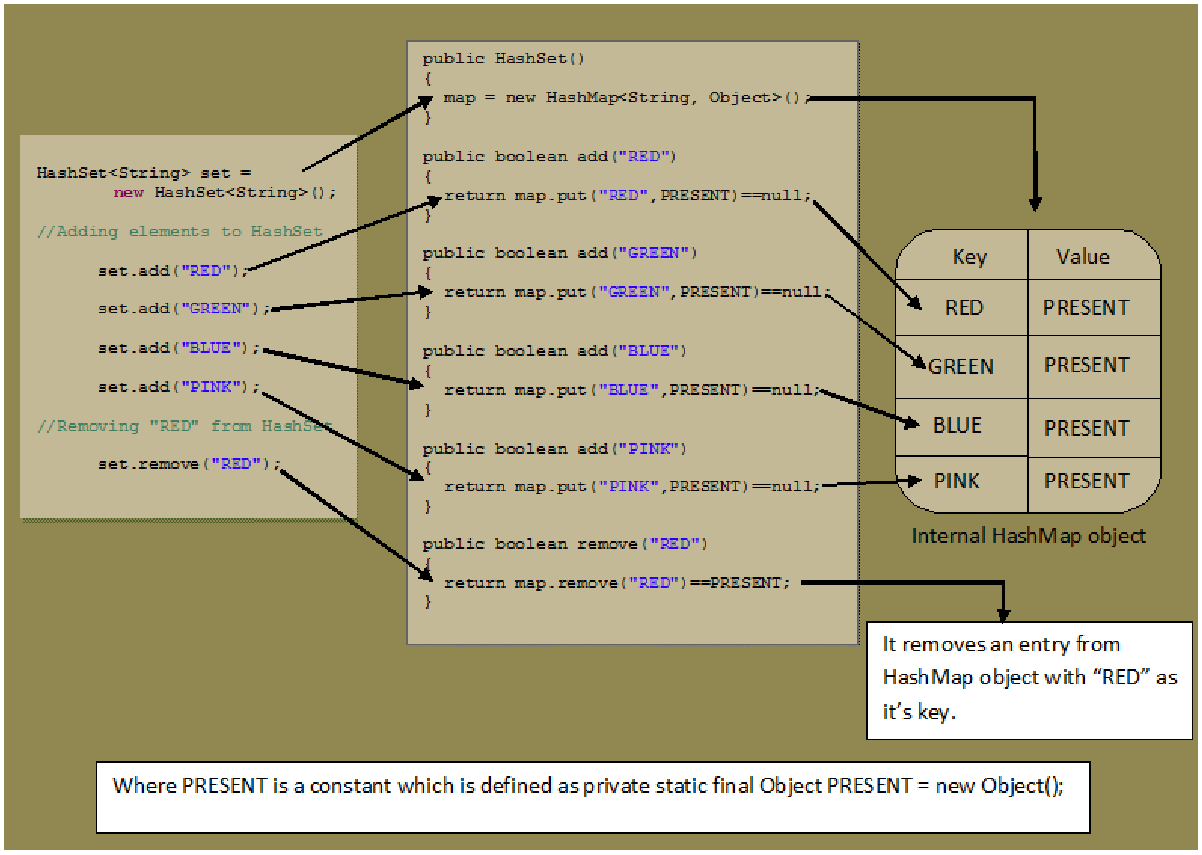
List	**Set**
List can have duplicate elements.	Set doesn't allow duplicate elements. It allows only unique elements.
List elements are ordered according zero-based index.	Order of elements in a set is implementation dependent. HashSet maintains no order. TreeSet elements are ordered according to supplied Comparator (If no Comparator is supplied, elements will be placed in their natural ascending order) and LinkedHashSet maintains insertion order.
List can have any number of null elements.	Set can have maximum one null element.
List elements can be accessed randomly.	Set elements can't be accessed randomly.
Ex : ArrayList, LinkedList	Ex : HashSet, LinkedHashSet, TreeSet

28) What are the characteristics of HashSet?

- HashSet implements Set interface.
- It is a collection of objects which contains only unique elements. It does not allow duplicate elements. If you try to insert a duplicate element, older element will be overwritten.
- HashSet class internally uses HashMap to store the objects. The elements you enter into HashSet will be stored as keys of HashMap and their values will be a constant.
- HashSet can have maximum one null element.
- HashSet doesn't maintain any order. The order of the elements will be largely unpredictable. And it also doesn't guarantee that order will remain constant over time.
- HashSet offers constant time performance for insertion, removal and retrieval operations.
- HashSet is not synchronized. If you want synchronized HashSet, use Collections.synchronizedSet() method.

29) How HashSet works internally in Java?

HashSet internally uses HashMap to store it's elements. Whenever you create a HashSet object, one **HashMap** object associated with it is also created. This HashMap object is used to store the elements you enter in the HashSet. The elements you add into HashSet are stored as keys of this HashMap object. The value associated with those keys will be a constant called PRESENT.



```
HashSet<String> set =
        new HashSet<String>();

//Adding elements to HashSet

        set.add("RED");

        set.add("GREEN");

        set.add("BLUE");

        set.add("PINK");

//Removing "RED" from HashSet

        set.remove("RED");
```

```
public HashSet()
{
  map = new HashMap<String, Object>();
}

public boolean add("RED")
{
  return map.put("RED",PRESENT)==null;
}

public boolean add("GREEN")
{
  return map.put("GREEN",PRESENT)==null;
}

public boolean add("BLUE")
{
  return map.put("BLUE",PRESENT)==null;
}

public boolean add("PINK")
{
  return map.put("PINK",PRESENT)==null;
}

public boolean remove("RED")
{
  return map.remove("RED")==PRESENT;
}
```

Key	Value
RED	PRESENT
GREEN	PRESENT
BLUE	PRESENT
PINK	PRESENT

Internal HashMap object
It removes an entry from HashMap object with "RED" as it's key.
Where PRESENT is a constant which is defined as private static final Object PRESENT = new Object();

(https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2015/01/HowHashSetWorks.png?ssl=1)

See More : How HashSet works internally in Java? (https://javaconceptoftheday.com/how-hashset-works-internally-in-java/)

30) What are the characteristics of LinkedHashSet?

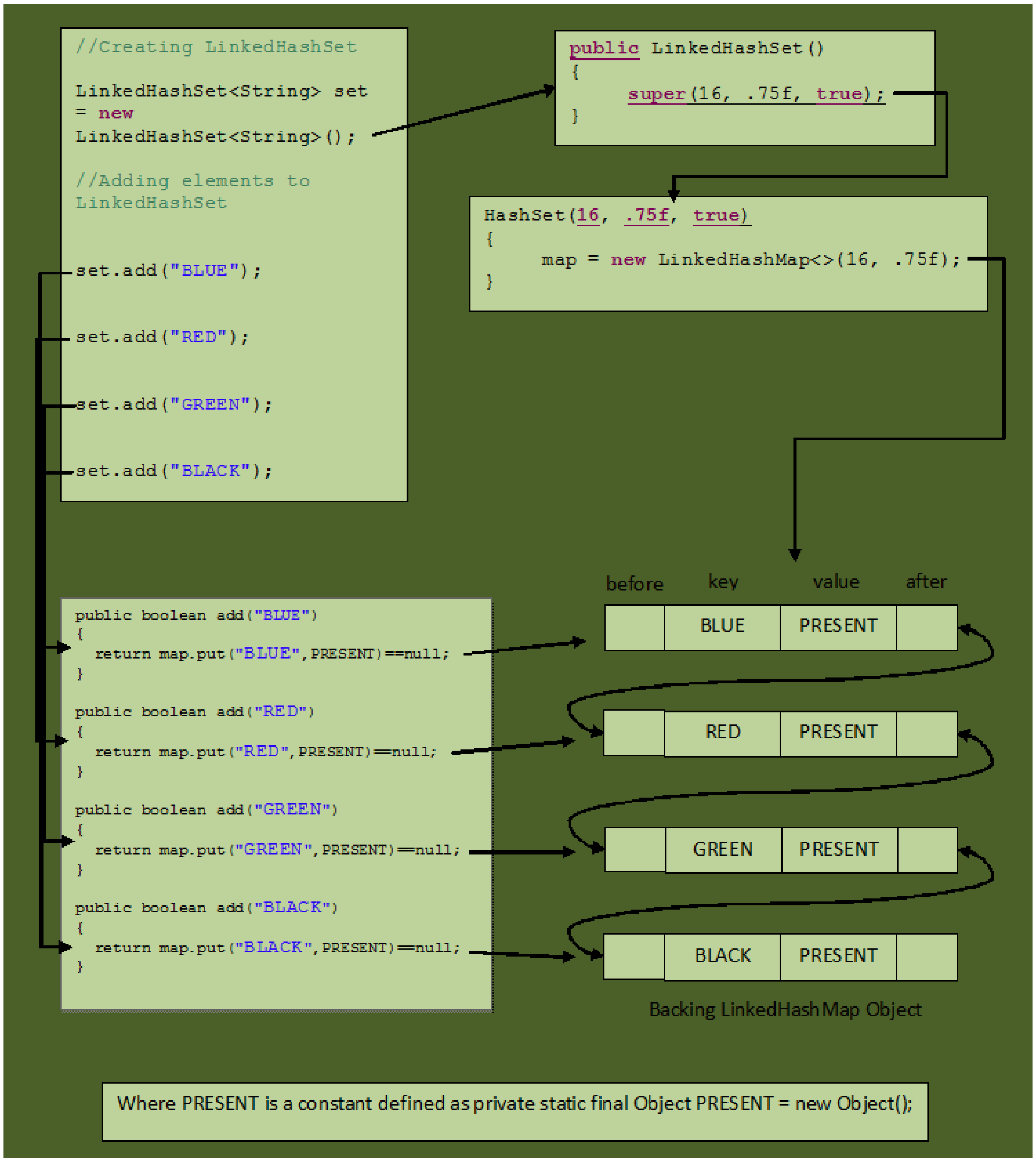
- LinkedHashSet internally uses LinkedHashMap to store it's elements just like HashSet which internally uses HashMap to store it's elements.
- LinkedHashSet maintains insertion order. This is the main difference between LinkedHashSet and HashSet.
- LinkedHashSet also gives constant time performance for insertion, removal and retrieval operations. The performance of LinkedHashSet is slightly less than the HashSet as it has to maintain linked list internally to order it's elements.
- LinkedHashSet doesn't allow duplicate elements and allows maximum one null element.
- Iterator returned by LinkedHashSet is fail-fast. i.e if the LinkedHashSet is modified at any time after the Iterator is created, it throws ConcurrentModificationException.
- LinkedHashSet is not synchronized. To get the synchronized LinkedHashSet, use Collections.synchronizedSet() method.

31) When you prefer LinkedHashSet over HashSet?

LinkedHashSet is preferred over HashSet if you want a unique collection of objects in an insertion order.

32) How LinkedHashSet works internally in Java?

LinkedHashSet is an extended version of HashSet. HashSet doesn't follow any order where as LinkedHashSet maintains insertion order. HashSet uses HashMap object internally to store it's elements where as LinkedHashSet uses LinkedHashMap object internally to store and process it's elements.



```
//Creating LinkedHashSet

LinkedHashSet<String> set
= new
LinkedHashSet<String>();

//Adding elements to
LinkedHashSet

set.add("BLUE");

set.add("RED");

set.add("GREEN");

set.add("BLACK");
```

```
public LinkedHashSet()
{
     super(16, .75f, true);
}
```

```
HashSet(16, .75f, true)
{
     map = new LinkedHashMap<>(16, .75f);
}
```

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

public boolean add("RED")
{
  return map.put("RED",PRESENT)==null;
}

public boolean add("GREEN")
{
  return map.put("GREEN",PRESENT)==null;
}

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

Where PRESENT is a constant defined as private static final Object PRESENT = new Object();

(https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2015/01/HowLinkedHashSetWorks.png?ssl=1)

See More : How LinkedHashSet works internally in Java? (https://javaconceptoftheday.com/how-linkedhashset-works-internally-in-java/)

33) What is SortedSet? Give one Example?

The SortedSet is an interface which extends Set interface. It’s elements are sorted, that’s why name SortedSet. The elements of the SortedSet are sorted according to supplied Comparator. This Comparator is supplied while creating a SortedSet. If you don’t supply Comparator, elements will be placed in their natural order.

TreeSet is the SortedSet.

34) What is NavigableSet? Give one example?

The NavigableSet is an interface which extends SortedSet interface which in turn extends Set interface.

The NavigableSet is a SortedSet with navigation facilities. The NavigableSet interface provides many methods through them you can easily find closest matches of any given element. It has the methods to find out less than, less than or equal to, greater than and greater than or equal of any element in a SortedSet.

TreeSet is also of type NavigableSet.

35) What are the characteristics of TreeSet?

- The elements in TreeSet are sorted according to specified Comparator. If no Comparator is specified, elements will be placed according to their natural ascending order.
- Elements inserted in the TreeSet must be of Comparable type and elements must be mutually comparable. If the elements are not mutually comparable, you will get ClassCastException at run time.
- TreeSet does not allow even a single null element.
- TreeSet is not synchronized. To get a synchronized TreeSet, use Collections.synchronizedSortedSet() method.
- TreeSet gives performance of order log(n) for insertion, removal and retrieval operations.
- Iterator returned by TreeSet is of fail-fast nature. That means, If TreeSet is modified after the creation of Iterator object, you will get ConcurrentModificationException.
- TreeSet internally uses TreeMap to store it’s elements just like HashSet and LinkedHashSet which use HashMap and LinkedHashMap respectively to store their elements.

36) How HashSet, LinkedHashSet and TreeSet differ from each other?

HashSet	**LinkedHashSet**	**TreeSet**

HashSet uses HashMap internally to store it's elements.	LinkedHashSet uses LinkedHashMap internally to store it's elements.	TreeSet uses TreeMap internally to store it's elements.
HashSet doesn't maintain any order of elements.	LinkedHashSet maintains insertion order of elements. i.e elements are placed as they are inserted.	TreeSet orders the elements according to supplied Comparator. If no Comparator is supplied, elements will be placed in their natural ascending order.
HashSet gives better performance than the LinkedHashSet and TreeSet.	The performance of LinkedHashSet is between HashSet and TreeSet. It's performance is almost similar to HashSet. But slightly in the slower side as it also maintains LinkedList internally to maintain the insertion order of elements.	TreeSet gives less performance than the HashSet and LinkedHashSet as it has to sort the elements after each insertion and removal operations.
HashSet gives performance of order O(1) for insertion, removal and retrieval operations.	LinkedHashSet also gives performance of order O(1) for insertion, removal and retrieval operations.	TreeSet gives performance of order O(log(n)) for insertion, removal and retrieval operations.
HashSet uses equals() and hashCode() methods to compare the elements and thus removing the possible duplicate elements.	LinkedHashSet also uses equals() and hashCode() methods to compare the elements.	TreeSet uses compare() or compareTo() methods to compare the elements and thus removing the possible duplicate elements. It doesn't use equals() and hashCode() methods for comparison of elements.
HashSet allows maximum one null element.	LinkedHashSet also allows maximum one null element.	TreeSet doesn't allow even a single null element. If you try to insert null element into TreeSet, it throws NullPointerException.
HashSet requires less memory than LinkedHashSet and TreeSet as it uses only HashMap internally to store its elements.	LinkedHashSet requires more memory than HashSet as it also maintains LinkedList along with HashMap to store its elements.	TreeSet also requires more memory than HashSet as it also maintains Comparator to sort the elements along with the TreeMap.

Use HashSet if you don't want to maintain any order of elements.	Use LinkedHashSet if you want to maintain insertion order of elements.	Use TreeSet if you want to sort the elements according to some Comparator.

See More : HashSet Vs LinkedHashSet Vs TreeSet (https://javaconceptoftheday.com/hashset-vs-linkedhashset-vs-treeset-in-java/)

37) What are the differences between Iterator and ListIterator?

Iterator	**ListIterator**
Using Iterator, you can traverse List, Set and Queue type of objects.	But using ListIterator, you can traverse only List objects.
Using Iterator, we can traverse the elements only in forward direction.	But, using ListIterator you can traverse the elements in both the directions – forward and backward.
Using Iterator you can only remove the elements from the collection.	But using ListIterator, you can perform modifications (insert, replace, remove) on the list.
You can't iterate a list from the specified index using Iterator.	But using ListIterator, you can iterate a list from the specified index.
Methods : hasNext(), next() and remove()	Methods : hasNext(), hasPrevious(), next(), previous(), nextIndex(), previousIndex(), remove(), set(), add()

See More : Iterator Vs ListIterator (https://javaconceptoftheday.com/difference-between-iterator-and-listiterator-in-java/)

38) How Map interface is different from other three primary interfaces of Java collection framework – List, Set and Queue?

The main difference between Map interface and other three top level interfaces is that it doesn't inherit from Collection interface. Instead it starts it's own interface hierarchy for maintaining the key-value associations.

Map stores the data as key-value pairs where each key is associated with a value where as other three interfaces – List, Set and Queue – store only values.

39) What are the popular implementations of Map interface?

- HashMap
- LinkedHashMap
- TreeMap

40) What are the characteristics of HashMap?

- HashMap holds the data in the form of key-value pairs where each key is associated with one value.
- HashMap doesn't allow duplicate keys. But it can have duplicate values.
- HashMap can have multiple null values and only one null key.
- HashMap is not synchronized. To get the synchronized *HashMap*, use *Collections.synchronizedMap()* method.
- HashMap maintains no order.
- HashMap gives constant time performance of O(1) for the operations like *get()* and *put()* methods.
- Default initial capacity of HashMap is 16.

41) How HashMap works internally in Java?

See here : How HashMap works internally in Java? (https://javaconceptoftheday.com/how-hashmap-works-internally-in-java/)

42) What is hashing?

The whole HashMap data structure is based on the principle of Hashing. Hashing is nothing but the function or algorithm or method which when applied on any object/variable returns an unique integer value representing that object/variable. This unique integer value is called *hash code*. Hash function or simply hash said to be the best if it returns the same hash code each time it is called on the same object.

43) What is the initial capacity of HashMap?

The capacity of an HashMap is the number of buckets in the hash table. The initial capacity is the capacity of an HashMap at the time of its creation. The default initial capacity of the HashMap is 2^4 i.e 16. The capacity of the HashMap is doubled each time it reaches the threshold. i.e the capacity is increased to 2^5=32, 2^6=64, 2^7=128….. when the threshold is reached.

44) What is the load factor of HashMap?

Load factor is the measure which decides when to increase the capacity of the HashMap. The default load factor is 0.75f.

45) What is the threshold of an HashMap? How it is calculated?

The threshold of an HashMap is the product of current capacity and load factor.

Threshold = (Current Capacity) * (Load Factor)

For example, if the HashMap is created with initial capacity of 16 and load factor of 0.75f, then threshold will be, Threshold = 16 * 0.75 = 12

That means, the capacity of the *HashMap* is increased from 16 to 32 after the 12th element (key-value pair) is added into the *HashMap*.

46) What is rehashing?

Rehashing is a process where new HashMap object with new capacity is created and all old elements (key-value pairs) are placed into new object after recalculating their hash code. Whenever HashMap reaches its threshold, rehashing takes place.

47) How initial capacity and load factor affect the performance of an HashMap?

Whenever *HashMap* reaches its threshold, rehashing takes place. This process of rehashing is both space and time consuming. So, you must choose the initial capacity, by keeping the number of expected elements (key-value pairs) in mind, so that rehashing process doesn't occur too frequently.

You also have to be very careful while choosing the load factor. According to *HashMap* doc, the default load factor of 0.75f always gives best performance in terms of both space and time. For example,

If you choose load factor as 1.0f, then rehashing takes place after filling 100% of the current capacity. This may save the space but it will increase the retrieval time of existing elements. Suppose if you choose load factor as 0.5f, then rehashing takes place after filling 50% of the current capacity. This will increase the number of rehashing operations. This will further degrade the HashMap in terms of both space and time.

So, you have to be very careful while choosing the initial capacity and load factor of an *HashMap* object. Choose the initial capacity and load factor such that they minimize the number of rehashing operations.

48) What are the differences between HashSet and HashMap?

HashSet	**HashMap**
HashSet implements Set interface.	HashMap implements Map interface.
HashSet stores the data as objects.	HashMap stores the data as key-value pairs.
HashSet internally uses HashMap.	HashMap internally uses an array of *Entry<K, V>* objects.
HashSet doesn't allow duplicate elements.	HashMap doesn't allow duplicate keys, but allows duplicate values.
HashSet allows only one null element.	HashMap allows one null key and multiple null values.

Insertion operation requires only one object.	Insertion operation requires two objects, key and value.
HashSet is slightly slower than HashMap.	HashMap is slightly faster than HashSet.

See More : HashMap Vs HashSet (https://javaconceptoftheday.com/differences-between-hashmap-vs-hashset-in-java/)

49) What are the differences between HashMap and HashTable?

HashMap	**HashTable**
HashMap is not synchronized and therefore it is not thread safe.	HashTable is internally synchronized and therefore it is thread safe.
HashMap allows maximum one null key and any number of null values.	HashTable doesn’t allow null keys and null values.
Iterators returned by the HashMap are fail-fast in nature.	Enumeration returned by the HashTable are fail-safe in nature.
HashMap extends AbstractMap class.	HashTable extends Dictionary class.
HashMap returns only iterators to traverse.	HashTable returns both Iterator as well as Enumeration for traversal.
HashMap is fast.	HashTable is slow.
HashMap is not a legacy class.	HashTable is a legacy class.
HashMap is preferred in single threaded applications. If you want to use HashMap in multi threaded application, wrap it using Collections.synchronizedMap() method.	Although HashTable is there to use in multi threaded applications, now a days it is not at all preferred. Because, ConcurrentHashMap is better option than HashTable.

See More : HashMap Vs HashTable (https://javaconceptoftheday.com/differences-between-hashmap-and-hashtable-in-java/)

50) How do you remove duplicate elements from an ArrayList in Java?

Removing Duplicate Elements From ArrayList Using HashSet :

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

public class MainClass
{
    public static void main(String[] args)
    {
        //Constructing An ArrayList

        ArrayList<String> listWithDuplicateElements = new ArrayList<Str

        listWithDuplicateElements.add("JAVA");

        listWithDuplicateElements.add("J2EE");

        listWithDuplicateElements.add("JSP");

        listWithDuplicateElements.add("SERVLETS");

        listWithDuplicateElements.add("JAVA");

        listWithDuplicateElements.add("STRUTS");

        listWithDuplicateElements.add("JSP");

        //Printing listWithDuplicateElements

        System.out.print("ArrayList With Duplicate Elements :");

        System.out.println(listWithDuplicateElements);

        //Constructing HashSet using listWithDuplicateElements

        HashSet<String> set = new HashSet<String>(listWithDuplicateElem

        //Constructing listWithoutDuplicateElements using set

        ArrayList<String> listWithoutDuplicateElements = new ArrayList<

        //Printing listWithoutDuplicateElements

        System.out.print("ArrayList After Removing Duplicate Elements :

        System.out.println(listWithoutDuplicateElements);
    }
}
```

See More : How To Remove Duplicate Elements From ArrayList In Java? (https://javaconceptoftheday.com/how-to-remove-duplicate-elements-from-arraylist-in-java/)

51) Which Collection type do you suggest me If I want a sorted collection of objects with no duplicates?

TreeSet is the best suitable for such scenarios where you want a collection of objects with no duplicates and also sorted based on a particular data field.

52) What are the differences between Fail-Fast Iterators and Fail-Safe Iterators?

Fail-Fast Iterators	**Fail-Safe Iterators**
Fail-Fast iterators doesn't allow modifications of a collection while iterating over it.	Fail-Safe iterators allow modifications of a collection while iterating over it.
These iterators throw `ConcurrentModificationException` if a collection is modified while iterating over it.	These iterators don't throw any exceptions if a collection is modified while iterating over it.
They use original collection to traverse over the elements of the collection.	They use copy of the original collection to traverse over the elements of the collection.
These iterators don't require extra memory.	These iterators require extra memory to clone the collection.
Ex : Iterators returned by *ArrayList*, *Vector*, *HashMap*.	Ex : Iterator returned by *ConcurrentHashMap.*

See More : Fail-Fast Vs Fail-Safe (https://javaconceptoftheday.com/fail-fast-and-fail-safe-iterators-in-java-with-examples/)

53) How do you convert an Array to ArrayList and an ArrayList to Array?

Array To ArrayList In Java :

a) Using `Arrays.asList()` Method :

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

public class ArrayToArrayListExample
{
    public static void main(String[] args)
    {
        String[] array = new String[] {"ANDROID", "JSP", "JAVA", "STRUT

        ArrayList<String> list = new ArrayList<String>(Arrays.asList(ar

        System.out.println(list);
    }
}
```

b) Using `Collections.addAll()` Method

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

public class ArrayToArrayListExample
{
    public static void main(String[] args)
    {
        String[] array = new String[] {"ANDROID", "JSP", "JAVA", "STRUT

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

        Collections.addAll(list, array);

        System.out.println(list);
    }
}
```

c) Using Java 8 Streams

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class ArrayToArrayListExample
{
    public static void main(String[] args)
    {
        String[] array = new String[] {"ANDROID", "JSP", "JAVA", "STRUT

        List<Object> list = Arrays.stream(array).collect(Collectors.toL

        System.out.println(list);
    }
}
```

ArrayList To Array In Java :

```java
import java.util.ArrayList;

public class ArrayListToArrayExample
{
    public static void main(String[] args)
    {
        ArrayList<String> list = new ArrayList<String>();

        list.add("JAVA");

        list.add("JSP");

        list.add("ANDROID");

        list.add("STRUTS");

        list.add("HADOOP");

        list.add("JSF");

        String[] array = new String[list.size()];

        list.toArray(array);

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

54) What is the difference between Collection and Collections?

This is one of the most confusing Java interview question asked many a times to Java freshers. Most of time, this question has been asked to Java freshers to check their basic knowledge about the Java Collection Framework. This question seems confusing because both `Collection` and `Collections` look similar. Both are part of Java collection framework, but both serve different purpose. `Collection` is a top level interface of Java collection framework where as `Collections` is an utility class. Below table shows the difference between them.

Collection	Collections
`Collection` is a root level interface of the Java Collection Framework. Most of the classes in Java Collection Framework inherit from this interface.	`Collections` is an utility class in `java.util` package. It consists of only static methods which are used to operate on objects of type Collection.
List, Set and Queue are main sub interfaces of this interface.	Collections.max(), Collections.min(), Collections.sort() are some methods of Collections class.

See More : Collection Vs Collections (https://javaconceptoftheday.com/difference-between-collection-and-collections-in-java/)

55) How collections are different from Java 8 streams?

Collections	Streams
Collections are mainly used to store and group the data.	Streams are mainly used to perform operations on data.
You can add or remove elements from collections.	You can't add or remove elements from streams.
Collections have to be iterated externally.	Streams are internally iterated.
Collections can be traversed multiple times.	Streams are traversable only once.
Collections are eagerly constructed.	Streams are lazily constructed.
Ex : List, Set, Map…	Ex : filtering, mapping, matching…

See More : Collections Vs Streams (https://javaconceptoftheday.com/collections-and-streams-in-java/)

56) How do you convert HashMap to ArrayList in Java?

See here : Convert HashMap To ArrayList In Java (https://javaconceptoftheday.com/convert-hashmap-to-arraylist-in-java/)

57) What keySet(), values() and entrySet() methods do?

keySet(), values() and entrySet() are the methods of Map interface. Hence, they are available in all the implementations of Map interface – HashMap, LinkedHashMap and TreeMap.

keySet() : This method returns a set of keys.

values() : This method returns a Collection of values.

entrySet() : This method returns a set of key-value pairs.

58) What is the difference between Iterator and Java 8 Spliterator?

Iterator	Spliterator
It performs only iteration.	It performs splitting as well as iteration.
Iterates elements one by one.	Iterates elements one by one or in bulk.
Most suitable for serial processing.	Most suitable for parallel processing.
Iterates only collection types.	Iterates collections, arrays and streams.

Iterator	**Spliterator**
Size is unknown.	You can get exact size or estimate of the size.
Introduced in JDK 1.2.	Introduced in JDK 1.8.
You can't extract properties of the iterating elements.	You can extract some properties of the iterating elements.
External iteration.	Internal iteration.

See More : Iterator Vs Spliterator (https://javaconceptoftheday.com/differences-between-iterator-vs-spliterator-in-java-8/)

59) How do you sort an ArrayList?

An ArrayList can be sorted using `sort()` method of `Collections` class.

Collections.sort() method has two overloaded forms. They are,

1) *sort(List<T> list)* : This method sorts the specified list according to natural ordering of its elements.

2) *sort(List<T> list, Comparator<? super T> c)* : This method sorts the specified list according to supplied Comparator.

These two methods are used not only just to sort the ArrayList, but also other list types like LinkedList and Vector.

See More : How To Sort An ArrayList In Java? (https://javaconceptoftheday.com/how-to-sort-an-arraylist-in-java/)

60) What are the differences between HashMap and ConcurrentHashMap?

HashMap	**ConcurrentHashMap**
HashMap is not synchronized internally and hence it is not thread safe.	ConcurrentHashMap is internally synchronized and hence it is thread safe.
HashMap is the part of Java collection framework since JDK 1.2.	ConcurrentHashMap is introduced in JDK 1.5 as an alternative to HashTable.
HashMap allows maximum one null key and any number of null values.	ConcurrentHashMap doesn't allow even a single null key and null value.
Iterators returned by HashMap are fail-fast in nature.	Iterators returned by ConcurrentHashMap are fail-safe in nature.

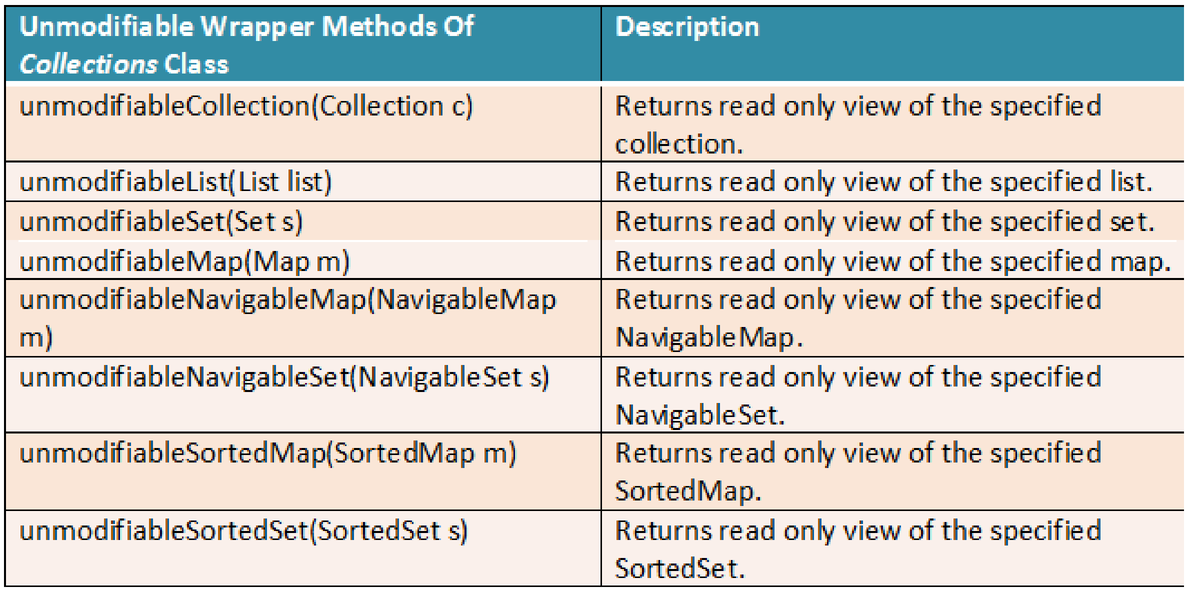
HashMap is faster.	ConcurrentHashMap is slower.
Most suitable for single threaded applications.	Most suitable for multi threaded applications.

See More : HashMap Vs ConcurrentHashMap (https://javaconceptoftheday.com/hashmap-vs-concurrenthashmap-in-java/)

61) How do you make collections read-only or unmodifiable?

java.util.Collections class provides some unmodifiable wrapper methods to create read only collections in Java. These methods take the *Collection* type as an argument and returns read only view of the specified collection. Any modification operations (like add, delete or edit an element) on the returned collection, direct or via its iterators, will result in *UnsupportedOperationException.* But, you can perform any modification operations on original collection and those modifications are reflected in the returned collection.

Below table shows complete list of all unmodifiable wrapper methods of *Collections* class which are used to create read only collections.



Unmodifiable Wrapper Methods Of *Collections* Class	Description
unmodifiableCollection(Collection c)	Returns read only view of the specified collection.
unmodifiableList(List list)	Returns read only view of the specified list.
unmodifiableSet(Set s)	Returns read only view of the specified set.
unmodifiableMap(Map m)	Returns read only view of the specified map.
unmodifiableNavigableMap(NavigableMap m)	Returns read only view of the specified NavigableMap.
unmodifiableNavigableSet(NavigableSet s)	Returns read only view of the specified NavigableSet.
unmodifiableSortedMap(SortedMap m)	Returns read only view of the specified SortedMap.
unmodifiableSortedSet(SortedSet s)	Returns read only view of the specified SortedSet.

(https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2018/07/ReadOnlyCollections.png?ssl=1)

62) How do you reverse an ArrayList in Java?

An *ArrayList* can be reversed using *Collections.reverse()* method. Below is the program.

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

public class ReverseArrayListExample
{
    public static void main(String[] args)
    {
        //Constructing an ArrayList

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

        list.add("Gold");

        list.add("Iron");

        list.add("Copper");

        list.add("Silver");

        list.add("Nickel");

        list.add("Cobalt");

        list.add("Zinc");

        //Printing list before reverse

        System.out.println("ArrayList Before Reverse :");

        System.out.println(list);

        //Reversing the list using Collections.reverse() method

        Collections.reverse(list);

        //Printing list after reverse

        System.out.println("ArrayList After Reverse :");

        System.out.println(list);
    }
}
```

63) What are the differences between synchronized HashMap, HashTable and ConcurrentHashMap?

	Synchronized HashMap	**HashTable**	**ConcurrentHashMap**
Locking Level	Object Level	Object Level	Segment Level
Synchronized operations	All operations are synchronized.	All operations are synchronized.	Only update operations are synchronized.

How many threads can enter into a map at a time?	Only one thread	Only one thread	By default, 16 threads can perform update operations and any number of threads can perform read operations at a time.
Null Keys And Null Values	Allows one null key and any number of null values.	Doesn't allow null keys and null values.	Doesn't allow null keys and null values.
Nature Of Iterators	Fail-Fast	Fail-Safe	Fail-Safe
Introduced In?	JDK 1.2	JDK 1.1	JDK 1.5
When To Use?	Use only when high level of data consistency is required in multi threaded environment.	Don't Use. Not recommended as it is a legacy class.	Use in all multi threaded environment except where high level of data consistency is required.

See More : Synchronized HashMap Vs HashTable Vs ConcurrentHashMap (https://javaconceptoftheday.com/synchronized-hashmap-vs-hashtable-vs-concurrenthashmap-in-java/)

64) How do you sort HashMap by keys?

See here : ***Solution (https://javaconceptoftheday.com/java-8-sort-hashmap-by-keys/)***

65) How do you sort HashMap by values?

See here : ***Solution (https://javaconceptoftheday.com/java-8-sort-hashmap-by-values/)***

66) How do you merge two maps with same keys?

See here : ***Solution (https://javaconceptoftheday.com/java-8-merge-two-maps-with-same-keys/)***

67) What do you know about Java 9 immutable collections? How they are different from unmodifiable collections returned by the Collections wrapper methods?

Immutable collections are the collections which can not be modified once they are created. Java 9 has introduced some static factory methods to easily create immutable collections. They are *List.of()*, *Set.of()* and *Map.of()*.

Before Java 9, `Collections.unmodifiableXXX()` methods are used to create unmodifiable collections. These methods just behave like wrapper methods which return unmodifiable view or read-only view of the original collection. i.e you can't perform modifying operations like add, remove, replace, clear etc through the references returned by these wrapper methods. But, you can modify original collection if you have other references to it and those modifications will be reflected in the view returned by these methods.

Java 9 Immutable collections and unmodifiable collections returned by the `Collections.unmodifiableXXX()` wrapper methods are not the same. Unmodifiable collections are just the read-only views of the original collection. You can perform modifying operations on the original collection and those modifications will be reflected in the collections returned by these methods. But, immutable collections returned by Java 9 static factory methods are 100% immutable. You can't modify them once they are created.

See More : Java 9 Immutable Collections (https://javaconceptoftheday.com/java-9-immutable-collections/)

68) What do you know about Java 10 List.copyOf(), Set.copyOf() and Map.copyOf() methods? Why they are introduced?

In Java 9, some static factory methods are introduced to easily create immutable collections. They are List.of(), Set.of() and Map.of(). These methods take individual elements as arguments and create immutable collections consisting of those elements. From Java 10, some more static factory methods are introduced to create immutable collections from existing collections. They are List.copyOf(), Set.copyOf() and Map.copyOf(). These methods take whole collection as an argument and create immutable copy of that collection.

69) What are the differences between Enumeration And Iterator?

Enumeration	**Iterator**
Using *Enumeration*, you can only traverse the collection. You can't do any modifications to collection while traversing it.	Using *Iterator*, you can remove an element of the collection while traversing it.
Enumeration is introduced in JDK 1.0	*Iterator* is introduced from JDK 1.2
Enumeration is used to traverse the legacy classes like *Vector*, *Stack* and *HashTable*.	*Iterator* is used to iterate most of the classes in the collection framework like *ArrayList*, *HashSet*, *HashMap*, *LinkedList* etc.
Methods : *hasMoreElements()* and *nextElement()*	Methods : *hasNext()*, *next()* and *remove()*
Enumeration is fail-safe in nature.	*Iterator* is fail-fast in nature.
Enumeration is not safe and secured due to it's fail-safe nature.	*Iterator* is safer and secured than *Enumeration*.

See More : Enumeration Vs Iterator In Java (https://javaconceptoftheday.com/differences-between-enumeration-vs-iterator-in-java/)

70) Which is of type RandomAccess – ArrayList, LinkedList, HashSet and HashMap?

ArrayList

Also Read : Java Collections Interview Questions – Baeldung (https://www.baeldung.com/java-collections-interview-questions)

PREVIOUS POST

Java 12 Switch Expressions (https://javaconceptoftheday.com/java-12-switch-expressions/)

NEXT POST

CSS Cheat Sheet (https://javaconceptoftheday.com/css-cheat-sheet/)

Related Posts