**Collections Interview Questions**

**1) What is the difference between ArrayList and Vector?**

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **Vector** |
| 1) | ArrayList is not synchronized. | Vector is synchronized. |
| 2) | ArrayList is not a legacy class. | Vector is a legacy class. |
| 3) | ArrayList increases its size by 50% of the array size. | Vector increases its size by doubling the array size. |

**2) What is the difference between ArrayList and LinkedList?**

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **LinkedList** |
| 1) | ArrayList uses a dynamic array. | LinkedList uses doubly linked list. |
| 2) | ArrayList is not efficient for manipulation because a lot of shifting is required. | LinkedList is efficient for manipulation. |
| 3) | ArrayList is better to store and fetch data. | LinkedList is better to manipulate data. |

**3) What is the difference between Iterator and ListIterator?**

Iterator traverses the elements in forward direction only whereas ListIterator traverses the elements in forward and backward direction.

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | Iterator traverses the elements in forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| 2) | Iterator can be used in List, Set and Queue. | ListIterator can be used in List only. |

**4) What is the difference between Iterator and Enumeration?**

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **Enumeration** |
| 1) | Iterator can traverse legacy and non-legacy elements. | Enumeration can traverse only legacy elements. |
| 2) | Iterator is fail-fast. | Enumeration is not fail-fast. |
| 3) | Iterator is slower than Enumeration. | Enumeration is faster than Iterator. |

**5) What is the difference between List and Set?**

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

**6) What is the difference between HashSet and TreeSet?**

HashSet maintains **no order** whereas TreeSet maintains **ascending order**.

**7) What is the difference between Set and Map?**

Set contains values only whereas Map contains key and values both.

**8) What is the difference between HashSet and HashMap?**

HashSet contains only values whereas HashMap contains entry(key,value). HashSet can be iterated but HashMap need to convert into Set to be iterated.

**9) What is the difference between HashMap and TreeMap?**

HashMap maintains **no order** but TreeMap maintains **ascending order**.

**10) What is the difference between HashMap and Hashtable?**

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |

**11) What is the difference between Collection and Collections?**

Collection is an interface whereas Collections is a class. Collection interface provides normal functionality of data structure to List, Set and Queue. But, Collections class is to sort and synchronize collection elements.

**12) What is the difference between Comparable and Comparator?**

|  |  |  |
| --- | --- | --- |
| **No.** | **Comparable** | **Comparator** |
| 1) | Comparable provides only one sort of sequence. | Comparator provides multiple sort of sequences. |
| 2) | It provides one method named compareTo(). | It provides one method named compare(). |
| 3) | It is found in java.lang package. | it is found in java.util package. |
| 4) | If we implement Comparable interface, actual class is modified. | Actual class is not modified. |

**13) What is the advantage of Properties file?**

If you change the value in properties file, you don't need to recompile the java class. So, it makes the application easy to manage.

**14) What does the hashCode() method?**

The hashCode() method returns a hash code value (an integer number).

The hashCode() method returns the same integer number, if two keys (by calling equals() method) are same.

But, it is possible that two hash code numbers can have different or same keys.

**15) Why we override equals() method?**

The equals method is used to check whether two objects are same or not. It needs to be overridden if we want to check the objects based on property.

For example, Employee is a class that has 3 data members: id, name and salary. But, we want to check the equality of employee object on the basis of salary. Then, we need to override the equals() method.

**16) How to synchronize List, Set and Map elements?**

Yes, Collections class provides methods to make List, Set or Map elements as synchronized:

|  |
| --- |
| public static List synchronizedList(List l){} |
| public static Set synchronizedSet(Set s){} |
| public static SortedSet synchronizedSortedSet(SortedSet s){} |
| public static Map synchronizedMap(Map m){} |
| public static SortedMap synchronizedSortedMap(SortedMap m){} |

**17) What is the advantage of generic collection?**

If we use generic class, we don't need typecasting. It is typesafe and checked at compile time.

**18) What is hash-collision in Hashtable and how it is handled in Java?**

Two different keys with the same hash value is known as hash-collision. Two different entries will be kept in a single hash bucket to avoid the collision.

**19) What is the Dictionary class?**

The Dictionary class provides the capability to store key-value pairs.

**20) What is the default size of load factor in hashing based collection?**

The default size of load factor is **0.75**. The default capacity is computed as initial capacity \* load factor. For example, 16 \* 0.75 = 12. So, 12 is the default capacity of Map.

**21)**[**What is the difference between Array and ArrayList?**](https://www.javatpoint.com/array-vs-arraylist-in-java)

Array is a fixed length data structure whereas ArrayList is a variable length Collection class. We cannot change length of array once created in Java but ArrayList can be changed.

We cannot store primitives in ArrayList, it can only store objects. But array can contain both primitives and objects in Java. Since Java 5, primitives are automatically converted in objects which is known as auto-boxing.

1. **import** java.util.\*;
2. **public** **class** ListExample {
3. **public** **static** **void** main(String[] args) {
4. List<Integer> list=**new** ArrayList<>();
5. list.add(Integer.valueOf(10));//storing Integer object
6. list.add(20);//Now compiler converts it into Integer.valueOf(20) which is object
7. list.add(30);
9. System.out.println("Traversing List...");
10. **for**(Integer i:list){
11. System.out.println(i);
12. }
13. }
14. }

**Output:**

Traversing List...

10

20

30

**22) What is the difference between length of Array and size of ArrayList?**

ArrayList doesn't have length() method, the size() method of ArrayList provides the number of objects available in the collection.

Array has length property which provides the length or capacity of the Array. It is the total space allocated during the intialization of the array.

1. **import** java.util.ArrayList;
2. **public** **class** LengthVsSize {
3. **public** **static** **void** main(String[] args) {
4. //creating array of 10 elements
5. **int** arr[]=**new** **int**[10];
6. //storing 2 elements
7. arr[0]=10;
8. arr[1]=12;
9. //printing length of array
10. System.out.println(arr.length);//10
12. //Creating ArrayList
13. ArrayList<String> list=**new** ArrayList<String>();
14. //storing 2 elements
15. list.add("ankit");
16. list.add("nippun");
17. //printing size of ArrayList
18. System.out.println(list.size());//2
19. }
20. }

Output:

10

2

**23) How to convert ArrayList to Array and Array to ArrayList?**

Let's see a simple example to convert ArrayList to Array and Array to ArrayList in Java:

1. **public** **class** LengthVsSizeArrayList {
2. **public** **static** **void** main(String[] args) {
3. //creating Arraylist
4. List<String> fruitList = **new** ArrayList<>();
5. //adding String Objects to fruitsList ArrayList
6. fruitList.add("Mango");
7. fruitList.add("Banana");
8. fruitList.add("Apple");
9. fruitList.add("Strawberry");
10. fruitList.add("Pineapple");
11. System.out.println("Converting ArrayList to Array" );
12. String[] item = fruitList.toArray(**new** String[fruitList.size()]);
13. **for**(String s : item){
14. System.out.println(s);
15. }
16. System.out.println("Converting Array to ArrayList" );
17. List<String>l2 = **new** ArrayList<>();
18. l2 =  Arrays.asList(item);
19. System.out.println(l2);
20. }
21. }

Output:

Converting ArrayList to Array

Mango

Banana

Apple

Strawberry

Pineapple

Converting Array to ArrayList

[Mango, Banana, Apple, Strawberry, Pineapple]

**24) How to make Java ArrayList Read-Only?**

The read-only means unmodifiable view of Collection in which we can not perform any operation which will change the collection through add(), remove() or set() method. We can obtain read-only collection from the existing collection by calling Collections.unmodifiableCollection() method.

1. **public** **class** UnmodifiableArrayList {
2. **public** **static** **void** main(String[] args) {
3. List<String>fruitList = **new** ArrayList<String>();
5. fruitList.add("Mango");
6. fruitList.add("Banana");
7. fruitList.add("Apple");
8. fruitList.add("Strawberry");
9. fruitList.add("Pineapple");
11. List<String>unmodifiableList= Collections.unmodifiableList(fruitList);
12. unmodifiableList.add("INDIA");
13. System.out.println(fruitList);
14. }
15. }

Output:

Exception in thread "main"java.lang.UnsupportedOperationException

at java.util.Collections$UnmodifiableCollection.add(Collections.java:1055)

at collectionInterview.list.UnmodifiableArrayList.main(UnmodifiableArrayList.java:20)

Here we have converted the existing list fruitList to unmodifiable List. If we alter the "unmodifiableList", it will cause UnsupportedOperationException. We can still change the list with fruitList reference. like calling:

1. fruitList.add("INDIA");

**25) How to remove duplicates from ArrayList?**

To remove dupliates from ArrayList, we can convert it into Set. Since Set doesn't contain duplicate elements, it will have only unique elements.

Let's see an example to remove duplicates from ArrayList:

1. **public** **class** RemoveDuplicateArrayList {
2. **public** **static** **void** main(String[] args) {
3. List<String> l = **new** ArrayList<String>();
4. l.add("Mango");
5. l.add("Banana");
6. l.add("Mango");
7. l.add("Apple");
8. System.out.println(l.toString());
9. Set<String> s = **new** LinkedHashSet<String>(l);
10. System.out.println(s);
11. }
12. }

Output:

Before converting to set

[Mango, Banana, Mango, Apple]

After converting to set

[Mango, Banana, Apple]

**26) How to reverse ArrayList?**

The reverse method of Collections class can be used to reverse any collection. It is a static method. Let's see the signature of reverse method:

1. **public** **static** **void** reverse(Collection c)

Let's see a simple example to reverse ArrayList in Java:

1. **public** **class** ReverseArrayList {
2. **public** **static** **void** main(String[] args) {
3. List<String> l = **new** ArrayList<String>();
4. l.add("Mango");
5. l.add("Banana");
6. l.add("Mango");
7. l.add("Apple");
8. System.out.println("Before Reversing");
9. System.out.println(l.toString());
11. Collections.reverse(l);
12. System.out.println("After Reversing");
13. System.out.println(l);
14. }
15. }

Output:

Before Reversing

[Mango, Banana, Mango, Apple]

After Reversing

[Apple, Mango, Banana, Mango]

**27) How to sort ArrayList in descending order?**

By using Collections.reverseOrder(Comparator<T>cmp) method, we can sort the collection in reverse order. The reverseOrder() method does the reversing on the basis of given Comparator. In case of null, it will reverse collection in natural ordering.

Let's see a simple example to sort the ArrayList in descending order.

***SmartPhone.java***

1. **import** java.util.Comparator;
2. **public** **class** SmartPhone {
3. String brand;
4. String model;
5. intprice;
6. intrating;
7. SmartPhone(String brand,String model,intprice, intrating){
8. **this**.brand = brand;
9. **this**.model = model;
10. **this**.price = price;
11. **this**.rating = rating;
13. }
14. **public** String getBrand() {
15. returnbrand;
16. }
17. **public** **void** setBrand(String brand) {
18. **this**.brand = brand;
19. }
20. **public** String getModel() {
21. returnmodel;
22. }
23. **public** **void** setModel(String model) {
24. **this**.model = model;
25. }
26. **public** **int** getPrice() {
27. returnprice;
28. }
29. **public** **void** setPrice(intprice) {
30. **this**.price = price;
31. }
32. **public** **int** getRating() {
33. returnrating;
34. }
35. **public** **void** setRating(intrating) {
36. **this**.rating = rating;
37. }
38. **public** String toString() {
39. **return**"SmartPhone [brand=" + brand + ", model=" + model + ", price=" + price + ", rating=" + rating + "]";
40. }
41. **public** **int** compareTo(SmartPhone sp) {
42. returnthis.price - sp.price;
44. }
45. }
46. **class** RatingComparator **implements** Comparator<SmartPhone> {
47. @Override
48. **public** **int** compare(SmartPhone obj1, SmartPhone obj2) {
49. **return** (obj1.rating<obj2.rating) ? -1 : (obj1.rating>obj2.rating) ? 1 : 0;
50. }
51. }

***ArrayListLearning.java***

1. **public** **class** ArrayListLearning {
2. @SuppressWarnings("unchecked")
3. **public** **static** **void** main(String[] args) {
5. List<SmartPhone> phoneList = **new** ArrayList<>();
6. SmartPhone ph1 = **new** SmartPhone("Apple", "6s", 50000, 10);
7. SmartPhone ph2 = **new** SmartPhone("lg", "pro2", 40000, 9);
8. SmartPhone ph3 = **new** SmartPhone("MI", "3s", 10000, 6);
9. SmartPhone ph4 = **new** SmartPhone("Letv", "le2", 12000, 7);
11. phoneList.add(ph1);
12. phoneList.add(ph2);
13. phoneList.add(ph3);
14. phoneList.add(ph4);
15. System.out.println("Actual List");
16. System.out.println(phoneList);
17. System.out.println("Sorting the list as comparator");
18. Collections.sort(phoneList, **new** RatingComparator());
20. System.out.println(phoneList);
21. System.out.println("Reversing the Comparator sorting");
22. Comparator<SmartPhone> cmp = Collections.reverseOrder(**new** RatingComparator());
24. Collections.sort(phoneList, cmp);
25. System.out.println("Printing the reverse list");
26. System.out.println(phoneList);
27. }
29. }

Output:

Actual List

[SmartPhone [brand=Apple, model=6s, price=50000, rating=10],

SmartPhone [brand=lg, model=pro2, price=40000, rating=9],

SmartPhone [brand=MI, model=3s, price=10000, rating=6],

SmartPhone [brand=Letv, model=le2, price=12000, rating=7]]

Sorting the list as comparator

[SmartPhone [brand=MI, model=3s, price=10000, rating=6],

SmartPhone [brand=Letv, model=le2, price=12000, rating=7],

SmartPhone [brand=lg, model=pro2, price=40000, rating=9],

SmartPhone [brand=Apple, model=6s, price=50000, rating=10]]

Reversing the Comparator sorting

Printing the reverse list

[SmartPhone [brand=Apple, model=6s, price=50000, rating=10],

SmartPhone [brand=lg, model=pro2, price=40000, rating=9],

SmartPhone [brand=Letv, model=le2, price=12000, rating=7],

SmartPhone [brand=MI, model=3s, price=10000, rating=6]]

**28) How to synchronize ArrayList?**

We can use Collections.synchronizedList(List<T>) method to synchronize collections in java. The synchronizedList(List<T>) method is used to return a synchronized (thread-safe) list backed by the specified list.

1. **import** java.util.\*;
2. **public** **class** SyncronizeArrayList {
3. **public** **static** **void** main(String args[]) {
4. // Non Synchronized ArrayList
5. List<String> fruitList = **new** ArrayList<String>();
7. fruitList.add("Mango");
8. fruitList.add("Banana");
9. fruitList.add("Apple");
10. fruitList.add("Strawberry");
11. fruitList.add("Pineapple");
13. // Synchronizing ArrayList in Java
14. furitList = Collections.synchronizedList(fruitList);
16. // we must use synchronize block to avoid non-deterministic behavior
17. **synchronized** (fruitList) {
18. Iterator<String> itr = fruitList.iterator();
19. **while** (itr.hasNext()) {
20. System.out.println(itr.next());
21. }
22. }
23. }
24. }

Output:

Mango

Banana

Apple

Strawberry

Pineapple

**29) When to use ArrayList and LinkedList?**

When to use ArrayList and LinkedList in Java

ArrayList provides constant time for search operation, so it is better to use ArrayList if searching is more frequent operation than add and remove operation. The LinkedList provides constant time for add and remove operations. So it is better to use LinkedList for manipulation.

ArrayList has O(1) time complexity to access elements via the get and set methods.

LinkedList has O(n/2) time complexity to access the elements.

LinkedLinked class implements Deque interface also, so you can get the functionality of double ended queue in LinkedList. The ArrayList class doesn't implement Deque interface.

In sort, ArrayList is better to access data wherease LinkedList is better to manipulate data. Both classes implements List interface.

ArrayList Example

1. **import** java.util.\*;
2. **public** **class** ListExample {
3. **public** **static** **void** main(String[] args) {
4. //ArrayList is better to store and view data
5. List<String> list=**new** ArrayList<>();
6. list.add("ankit");
7. list.add("peter");
8. list.add("mayank");
10. System.out.println("Traversing ArrayList...");
11. **for**(String s:list){
12. System.out.println(s);
13. }
14. }
15. }

Output:

Traversing ArrayList...

ankit

peter

mayank

LinkedList Example

1. **import** java.util.\*;
2. **public** **class** ListExample2 {
3. **public** **static** **void** main(String[] args) {
4. //LinkedList is better to manipulate data
5. List<String> list=**new** LinkedList<>();
6. list.add("ankit");
7. list.add("peter");
8. list.add("mayank");
9. System.out.println("After adding: "+list);
10. list.remove("peter");
11. System.out.println("After removing: "+list);
12. list.set(1,"vivek");
13. System.out.println("After changing: "+list);
14. }
15. }

Output:

After adding: [ankit, peter, mayank]

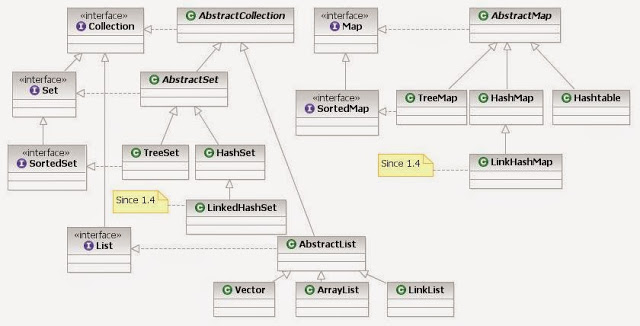
After removing: [ankit, mayank]

After changing: [ankit, vivek]

***Beginner Level (0-1 yr): Java Collections Interview Questions  and Answers***  
  
**Q1  What is Collection ? What is a Collections Framework ? What are the benefits of** **Java Collections Framework ?**  
**Collection :** A collection (also called as container) is an object  that groups multiple elements into a single unit.  
  
**Collections Framework :**Collections framework provides unified architecture for manipulating and representing collections.  
  
**Benefits of Collections Framework :**  
1. Improves program quality and speed  
2. Increases the chances of reusability of software  
3. Decreases programming effort.  
  
**Q2 What is the root interface in collection hierarchy ?**  
Root interface in collection hierarchy is **Collection interface .**Few interviewer may argue that   
Collection interface extends **Iterable interface**. So iterable should be the root interface. But you should reply iterable interface present in java.lang package not in java.util package .It is clearly mentioned in [Oracle Collection  docs](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) , that Collection interface is a member of the Java Collections framework.  For [Iterable interface Oracle doc](https://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html) , iterable interface is not mentioned as a part of the Java Collections framework .So if the question includes  collection hierarchy , then you should answer the question as Collection interface (which is found in java.util package).  
 **Q3 What is the difference between Collection and Collections ?**  
Collection is  an interface while Collections is a java class , both are present in java.util package and  part of java collections framework.

**Q4 Which collection classes are synchronized or thread-safe ?**  
Stack, Properties , Vector and Hashtable can be used in multi threaded environment because they are synchronized classes (or thread-safe).

**Q5 Name the core Collection  interfaces ?**

[](http://1.bp.blogspot.com/-ifC30f-ZZ7M/VU7wWRwqo3I/AAAAAAAAAmQ/b1CIhOP5NU8/s1600/Java_collection_framework.jpg)The list of core collection interfaces are : just mention the important ones  
  
Important : Collection , Set , Queue , List , Map  
  
Other interface also in the list :  SortedSet, SortedMap , Deque, ListIterator etc.  
 **Q6 What is the difference between List and Set ?**  
Set contain only unique elements while List can contain duplicate elements.  
Set is unordered while List is ordered . List maintains the order in which the objects are added .  
  
**Q7 What is the difference between Map and Set ?**  
Map object has unique keys each containing some value, while Set contain only unique values.  
  
**Q8 What are the classes implementing List and Set interface ?**  
***Class implementing List interface :***  ArrayList , Vector , LinkedList ,  
  
***Class implementing Set interface :***HashSet , TreeSet

**Q9 What is an iterator ?**  
Iterator is an interface . It is found in java.util package. It provides methods to iterate over any Collection.  
  
  
**Q10 What is the difference between Iterator and Enumeration ?**  
The main difference between Iterator and Enumeration is that Iterator has remove() method while Enumeration doesn't.  
Hence , using Iterator we can manipulate objects by adding and removing the objects from the collections. Enumeration behaves like a read only interface as it can only traverse the objects and fetch it .  
  
**Q11 Which design pattern followed by Iterator ?**  
It follows iterator design pattern. Iterator design pattern provides us to navigate through the collection of objects by using a common interface without letting us know about the underlying implementation.  
  
Enumeration is an example of Iterator design pattern.  
 **Q12 Which methods you need to override to use any object as key in HashMap ?**  
To use any object as key in HashMap , it needs to implement equals() and hashCode() method .  
  
**Q13  What is the difference between Queue and Stack ?**  
Queue is a data structure which is based on FIFO ( first in first out ) property . An example of Queue in real world is buying movie tickets in the multiplex or cinema theaters.  
  
Stack is a data structure which is based on LIFO (last in first out) property . An example of Stack in real world is  insertion or removal of CD  from the CD case.  
  
**Q14 How to reverse the List in Collections ?**  
There is a built in reverse method in Collections class . reverse(List list) accepts list as parameter.  
  
**Collections.reverse(listobject);**  
  
**Q15 How to convert the array of strings into the list ?**  
Arrays class of java.util package contains the method asList() which accepts the array as parameter.  
So,  
  
**String[]  wordArray =  {"Love Yourself"  , "Alive is Awesome" , "Be in present"};**  
**List wordList =  Arrays.asList(wordArray);**  
  
  
***Intermediate Level (1-3 yrs): Java Collections Interview Questions  and Answers***

**Q16 What is the difference between ArrayList and Vector ?**

It is one of the frequently asked collection interview question , the main differences are  
Vector is synchronized while ArrayList is not . Vector is slow while ArrayList is fast . Every time when needed, Vector increases the capacity twice of its initial size while ArrayList increases its ArraySize by 50%. find detailed explanation   [ArrayList vs Vector](http://javahungry.blogspot.co.uk/2013/12/difference-between-arraylist-and-vector-in-java-collection-interview-question.html)  .

**Q17 What is the difference between HashMap and Hashtable ?**  
It is one of the most popular collections interview question for java developer . Make sure you go through this once before appearing for the interview .  
Main differences between HashMap and Hashtable are :  
  
a. HashMap allows one null key and any number of null values while Hashtable does not allow null keys and null values.  
b. HashMap is not synchronized or thread-safe while Hashtable is synchronized or thread-safe .  
find detailed explanation here [Hashtable vs HashMap in Java](http://javahungry.blogspot.co.uk/2014/03/hashmap-vs-hashtable-difference-with-example-java-interview-questions.html)  
  
**Q18 What is the difference between peek(),poll() and remove() method of the Queue** **interface ?**  
Both poll() and remove() method is used to remove head object of the Queue. The main difference lies when the Queue is empty().  
If Queue is empty then poll() method will return null . While in similar case , remove() method will throw NoSuchElementException .  
peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

**Q19 What is the difference between Iterator and ListIterator.**  
Using Iterator we can traverse the list of objects in forward direction . But ListIterator can traverse the collection in both directions that is forward as well as backward.  
  
**Q20 What is the difference between Array and ArrayList in Java ?**  
This question checks whether student understand the concept of static and dynamic array. Some main differences between Array and ArrayList are :  
a. Array is static in size while ArrayList is dynamic in size.  
b. Array can contain primitive data types while ArrayList can not contain primitive data types.

find detailed explanation [ArrayList vs Array in Java](http://javahungry.blogspot.ca/2015/03/difference-between-array-and-arraylist-in-java-example.html)

**Q21 What is the difference between HashSet and TreeSet ?**  
Main differences between HashSet and TreeSet are :  
a.  HashSet maintains the inserted elements in random order while TreeSet maintains elements in the sorted order  
b. HashSet can store null object while TreeSet can not store null object.  
find detailed explanation here [TreeSet vs HashSet in Java](http://javahungry.blogspot.co.uk/2014/03/difference-between-hashset-and-treeset-similarities-and-example.html)  
  
  
**Q22 Write java code showing insertion,deletion and retrieval of HashMap object ?**  
Do it yourself (DIY) , if found any difficulty or doubts then please mention in the comments.  
  
**Q23 What is the difference between HashMap and ConcurrentHashMap ?**  
This is also one of the most popular java collections interview question . Make sure this question is in your to do list before appearing for the interview .  
Main differences between HashMap and ConcurrentHashMap are :  
a. HashMap is not synchronized while ConcurrentHashMap is synchronized.  
b. HashMap can have one null key and any number of null values while ConcurrentHashMap does not allow null keys and null values .  
find detailed explanation here [ConcurrentHashMap vs HashMap in Java](http://javahungry.blogspot.co.uk/2014/02/hashmap-vs-concurrenthashmap-java-collections-interview-question.html)  
  
**Q24 Arrange the following in the ascending order (performance):  
HashMap , Hashtable , ConcurrentHashMap and Collections.SynchronizedMap**  
Hashtable  <  Collections.SynchronizedMap  <  ConcurrentHashMap  <  HashMap

**Q25 How HashMap works in Java ?**  
This is one of the most important question for java developers. HashMap  works on the principle of Hashing . Find detailed information here to understand [what is hashing and how hashmap works in java](http://javahungry.blogspot.co.uk/2013/08/hashing-how-hash-map-works-in-java-or.html) .  
  
**Q26 What is the difference between LinkedList and ArrayList in Java ?**  
Main differences between LinkedList and ArrayList are :  
a. LinkedList is the doubly linked list implementation of list interface , while , ArrayList is the resizable array implementation of list interface.  
b. LinkedList can be traversed in the reverse direction using descendingIterator() method  provided by the Java Api developers , while , we need to implement our own method to traverse ArrayList in the reverse direction . find the detailed explanation here [ArrayList vs LinkedList in java](http://javahungry.blogspot.co.uk/2015/04/difference-between-arraylist-and-linkedlist-in-java-example.html).  
  
  
  
**Q27 What are Comparable and Comparator interfaces ? List the difference between** **them ?**

We already explained what is comparable and comparator interface in detail along with examples here,  [Comparable vs Comparator in Java](http://javahungry.blogspot.com/2013/08/difference-between-comparable-and.html)

**Q28 Why Map interface does not extend the Collection interface in Java Collections** **Framework ?**  
One liner answer : **Map interface is not compatible with the Collection interface.**  
Explanation : Since Map requires key as well as value , for example , if we want to add key-value pair then we will use put(Object key , Object value) . So there are two parameters required to add element to the HashMap object  . In Collection interface add(Object o) has only one parameter.   
The other reasons are Map supports valueSet , keySet as well as other appropriate methods which have just different views from the Collection interface.  
  
**Q29 When to use ArrayList and when to use LinkedList in application?**  
ArrayList has constant time search operation O(1) .Hence, ArrayList is preferred when there are more get() or search operation .  
  
Insertion , Deletion operations take constant time O(1) for LinkedList. Hence, LinkedList is preferred when there are more insertions or deletions involved in the application.  
  
**Q30 Write the code for iterating the list in different ways in java ?**  
There are two ways to iterate over the list in java :  
a. using Iterator  
b. using for-each loop  
  
Coding part : Do it  yourself (DIY) , in case of any doubts or difficulty please mention in the comments .

***Advance Level (3+ yrs): Java Collections Interview Questions  and Answers*  
Q31 How HashSet works internally in java ?**  
This is one of the popular interview question . HashSet internally uses HashMap to maintain the uniqueness of elements. We have already discussed in detail [hashset internal working in java](http://javahungry.blogspot.co.uk/2013/08/how-sets-are-implemented-internally-in.html).  
  
**Q32 What is CopyOnWriteArrayList ?  How it is different from  ArrayList in Java?**  
[CopyOnWriteArrayList](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) is a thread safe variant of ArrayList   in which all mutative operations like add , set are implemented by creating a fresh copy of the underlying array.  
It guaranteed not to throw ConcurrentModificationException.  
It permits all elements including null. It is introduced in jdk 1.5 .

**Q33  How HashMap works in Java ?**  
We are repeating this question , as it is one of the most important question for java developer.HashMap works on the principle of Hashing . please find the detailed answer here [hashmap internal working in java](http://javahungry.blogspot.co.uk/2013/08/hashing-how-hash-map-works-in-java-or.html) .  
  
**Q34 How remove(key) method works in HashMap ?**  
This is the new question which is getting popular among java interviewers . We have shared the detailed explanation here [how remove method works internally in java](http://javahungry.blogspot.co.uk/2015/03/how-remove-method-internally-works-in-hashmap-java.html).  
  
**Q35 What is BlockingQueue in Java Collections Framework?**  
[BlockingQueue](https://docs.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html) implements the java.util.Queue interface . BlockingQueue 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 .  
It does not accept null elements.  
Blocking queues are primarily designed for the producer-consumer problems.  
BlockingQueue implementations are thread-safe and can also be used in inter-thread communications.  
This concurrent Collection class was added in jdk 1.5  
**Q36 How TreeMap works in Java ?**  
TreeMap internally uses Red-Black tree to sort the elements in natural order. Please find the detailed answers here [internal implementation of TreeMap in java](http://javahungry.blogspot.co.uk/2014/06/how-treemap-works-ten-treemap-java-interview-questions.html) .  
  
**Q37 All the questions related to HashSet class can be found here ,**[**frequently asked HashSet interview questions**](http://javahungry.blogspot.co.uk/2014/04/top-10-hashset-java-interview-questions-collection.html)  
  
**Q38 What is the difference between Fail- fast iterator and Fail-safe iterator ?**This is one  of the most popular interview question for the higher experienced java developers .  
Main differences between Fail-fast and Fail-safe iterators are :  
a. Fail-fast throw ConcurrentModificationException while Fail-safe does not.  
b. Fail-fast does not clone the original collection list of objects while Fail-safe creates a copy of the original collection list of objects.  
The difference is explained in detail here [fail-safe vs fail-fast iterator in java](http://javahungry.blogspot.co.uk/2014/04/fail-fast-iterator-vs-fail-safe-iterator-difference-with-example-in-java.html).  
  
**Q39 How ConcurrentHashMap works internally in Java?**  
The detailed answer is already explained here  [internal implementation of concurrenthashmap](http://javahungry.blogspot.co.uk/2015/02/how-concurrenthashmap-works-in-java-internal-implementation.html)  
  
**Q40 How do you use a custom object as key in Collection  classes like HashMap ?**If one is using the custom object as key then one needs to override equals() and hashCode() method  
and one also need to fulfill the contract.  
If you want to store the custom object in the SortedCollections like SortedMap then one needs to make sure that equals() method is consistent to the compareTo() method. If inconsistent , then collection will not follow their contracts ,that is , Sets may allow duplicate elements.

**Q41 What is hash-collision in Hashtable ? How it was handled in Java?**  
In Hashtable , if two different keys have the same hash value then it lead to hash -collision. A bucket of type linkedlist used to hold the different keys of same hash value.  
  
**Q42 Explain the importance of hashCode() and equals() method ? Explain the contract** **also ?**HashMap object uses Key object hashCode() method and equals() method to find out the index to put the key-value pair. If we want to get value from the HashMap same both methods are used . Somehow, if both methods are not implemented correctly , it will result in two keys producing the same hashCode() and equals() output. The problem will arise that HashMap will treat both output same instead of different and overwrite the most recent key-value pair with the previous key-value pair.  
Similarly all the collection classes that does not allow the duplicate values use hashCode() and equals() method to find the duplicate elements.So it is very important to implement them correctly.  
  
**Contract of hashCode() and equals() method**  
a.If  object1.equals(object2) , then  object1.hashCode() == object2.hashCode() should always be true.  
  
b. If object1.hashCode() == object2.hashCode() is true does not guarantee object1.equals(object2)  
 **Q45 How do you convert a given Collection to SynchronizedCollection ?**One line code :    Collections.synchronizedCollection(Collection collectionObj) will convert a given collection to synchronized collection.  
  
**Q46  What is IdentityHashMap ?  
  
IdentityHashMap**  
[IdentityHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/IdentityHashMap.html) is a class present in java.util package. It implements the Map interface with a hash table , using [reference equality instead of object equality](http://javahungry.blogspot.co.uk/2013/06/difference-between-equals-and-double-equals-method-with-example-java-collections-interview-question.html) when comparing keys and values.In other words , in IdentityHashMap two keys k1 and k2 are considered equal if only if (k1==k2).  
IdentityHashMap is not synchronized.  
Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.   
**Q47 What is  WeakHashMap ?   
WeakHashMap :**[WeakHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/WeakHashMap.html) is a class present in java.util package similar to IdentityHashMap. It is a Hashtable based implementation of Map interface with weak keys. An entry in WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector.  
It permits null keys and null values.  
Like most collection classes this class is not synchronized.A synchronized WeakHashMap may be constructed using the Collections.synchronizedMap() method.  
Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

**Q48 How will you make Collections readOnly ?**

We can make the Collection readOnly by using the following lines code:

General : Collections.unmodifiableCollection(Collection c)  
  
Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)

**Q49  What is UnsupportedOperationException?**This exception is thrown to indicate that the requested operation is not supported.  
Example of UnsupportedOperationException:  
In other words, if you call add() or remove() method on the readOnly collection . We know readOnly collection can not be modified . Hence , UnsupportedOperationException will be thrown.  
 **Q50 Suppose there is an Employee class. We add Employee class objects to the** **ArrayList. Mention the steps need to be taken , if I want to sort the objects in ArrayList** **using the employeeId attribute present  in Employee class.**a. Implement the Comparable interface for the Employee class and now to compare the objects by employeeId we will override the emp1.compareTo(emp2)  
b. We will now call Collections class sort method and pass the list as argument , that is ,  
     Collections.sort(empList)  

**If an Employee class is present and its objects are added in an arrayList. Now I want the** **list to be sorted on the basis of the employeeID of Employee class. What are the steps?**

Ans)

* Implement Comparable interface for the Employee class and override the compareTo(Object obj) method in which compare the employeeID
* Now call Collections.sort() method and pass the list as an argument.

*Now consider that Employee class is a jar file.*

* 1) Since Comparable interface cannot be implemented, create Comparator and override the compare(Object obj, Object obj1) method .
* 2) Call Collections.sort() on the list and pass comparator as an argument.

**Q4)What is difference between HashMap and HashTable?**

Ans) Both collections implements Map. Both collections store value as key-value pairs. The key differences between the two are:

1. Hashmap is not synchronized in nature but hashtable is(thread-safe). This means that in a multithreaded application, only one thread can gets access to a hashtable object and do an operation on it. Hashmap doesn't gurantee such behavior and is not used in multithreaded environment.
2. Hashmap is traveresed using an iterator, hashtable can be traversed by enumerator or iterator.
3. Iterator in hashmap is fail-fast, enumerator in hashtable is not fail-fast
4. HashMap permits null values and only one null key, while Hashtable doesn't allow key or value as null.
5. Since hashtable is synchronized, it is relatively slower in performance than hashmap

**Q5) What are the classes implementing the List interface?**

Ans) There are three implementation of List interface:

1. **ArrayList** : It is a resizable array implementation. The size of the ArrayList can be increased dynamically also operations like add,remove and get can be formed once the object is created. It also ensures that the data is retrieved in the manner it was stored. The ArrayList is not thread-safe.
2. **Vector**: It is thread-safe implementation of ArrayList. The methods are wrapped around a synchronized block.
3. **LinkedList**: the LinkedList implements Queue interface too and provide FIFO (First In First Out) operation for add operation. It is faster than ArrayList if its mainly used forinsertion and deletion of elements.

**Q6) Which all classes implement Set interface ?**

Ans) A Set is a collection that contains no duplicate elements. More formally, a set contains no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. **HashSet,SortedSet and TreeSet** are the commonly used class which implements Set interface.

* **SortedSet** - It is an interface which extends Set. A the name suggest, the interface allows the data to be iterated in the ascending order or sorted on the basis of Comparator or Comparable interface. All elements inserted into the interface must implement Comparable or Comparator interface.
* **TreeSet** - It is the implementation of SortedSet interface. This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains). The class is not synchronized. The class uses Red-Black tree data structure.
* **HashSet:** This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets

**Q7) What is difference between List and a Set?**

Ans)

1. List can contain duplicate values but Set doesn't allow.
2. List allows retrieval of data to be in same order in the way it is inserted but Set doesnt ensures the sequence in which data can be retrieved.(Except HashSet)

**Q8) What is difference between Arrays and ArrayList ?**

Ans)

* Arrays are created of fix size whereas ArrayList is dynamic in nature and can vary its length. Also the size of array cannot be incremented or decremented. But with arrayList the size is variable.
* Once the array is created elements cannot be added or deleted from it. But with ArrayList the elements can be added and deleted at runtime.
* List list = new ArrayList();
* list.add(1);
* list.add(3);

list.remove(0) // will remove the element from the 1st location.

* ArrayList is one dimensional but array can be multidimensional.

int[][][] intArray= new int[3][2][1]; // 3 dimensional array

* Array can contain objects of a single data type or class. ArrayList if not used with generic can contain objects of different classes

**Q9) When to use ArrayList or LinkedList ?**

Ans)

1. Adding new elements is pretty fast for either type of list. Inserting element to nth location in arraylist and to first location in linkedlist takes O(1).
2. For the ArrayList, doing random lookup using "get" is faster O(1), but for LinkedList O(n), it's slow. It's slow because there's no efficient way to index into the middle of a linked list. Linkedlist lookup always start from 1st location.
3. When removing elements, using ArrayList is slow. This is because all remaining elements in the underlying array of Object instances must be shifted down for each remove operation. But LinkedList is fast, because deletion can be done simply by changing a couple of links.

So an ArrayList works best for cases where you're doing random access on the list and a LinkedList works better if you're doing a lot of editing in the middle of the list.

Source : [Read More - from java.sun](http://java.sun.com/developer/TechTips/1999/tt0809.html)

**Q11) What are advantages of iterating a collection using iterator?**

Ans) For loop does not allow updating the colection(add or remove) whereas Iterator does. Also Iterator can be used where there is no clue what type of collections will be used because all collections implement Iterator interface.

**Q12) Which design pattern Iterator follows?**

Ans) It follows Iterator design pattern. Iterator Pattern is a type of behavioral pattern. The Iterator pattern is one, which allows you to navigate through a collection of data using a common interface without knowing about the underlying implementation. Iterator should be implemented as an interface. This allows the user to implement it anyway its easier for him/her to return data. The benefits of Iterator are about their strength to provide a common interface for iterating through collections without bothering about underlying implementation.

Example of Iteration design pattern - Enumeration The class java.util.Enumeration is an example of the Iterator pattern. It represents and abstract means of iterating over a collection of elements in some sequential order without the client having to know the representation of the collection being iterated over. It can be used to provide a uniform interface for traversing collections of all kinds.

**When to use List, Set and Map in Java**

Based upon our understanding of *difference between Set, List and Map* we can now decide when to use List, Set or Map in Java.

1) If you need to access elements frequently by using index, than List is a way to go. Its implementation e.g. [ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) provides faster access if you know index.

2) If you want to store elements and want them to maintain an **order** on which they are inserted into collection then go for List again, as List is an ordered collection and maintain insertion order.

3) If you want to create collection of unique elements and **don't want any duplicate** than choose any Set implementation e.g. HashSet, LinkedHashSet or TreeSet. All Set implementation follow there general contract e.g. uniqueness but also add addition feature e.g. TreeSet is a SortedSet and elements stored on TreeSet can be sorted by using [Comparator or Comparable in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html). LinkedHashSet also maintains insertion order.

4) If you store data in form of key and value than Map is the way to go. You can choose from Hashtable, HashMap, TreeMap based upon your subsequent need. In order to choose between first

**Q) What is the difference between ArrayList and Vector?**A[ArrayList](http://beginnersbook.com/2013/12/java-arraylist/) and [Vector](http://beginnersbook.com/2013/12/vector-in-java/) both use Array as a data structure internally. However there are few differences in the way they store and process the data. In this post we will discuss the difference and similarities between ArrayList and Vector.

**ArrayList Vs Vector:**

1) **Synchronization**: ArrayList is non-synchronized which means multiple threads can work on ArrayList at the same time. For e.g. if one thread is performing an add operation on ArrayList, there can be an another thread performing remove operation on ArrayList at the same time in a multithreaded environment

while Vector is synchronized. This means if one thread is working on Vector, no other thread can get a hold of it. Unlike ArrayList, only one thread can perform an operation on vector at a time.

2) **Resize:** Both ArrayList and Vector can grow and shrink dynamically to maintain the optimal use of storage, however the way they resized is different. ArrayList grow by half of its size when resized while Vector doubles the size of itself by default when grows.

3) **Performance**: ArrayList gives better performance as it is non-synchronized. Vector operations gives poor performance as they are thread-safe, the thread which works on Vector gets a lock on it which makes other thread wait till the lock is released.

4) **fail-fast**: First let me explain what is fail-fast: If the collection (ArrayList, vector etc) gets structurally modified by any means, except the **add or remove methods** of iterator, after creation of iterator then the iterator will throw [ConcurrentModificationException](http://docs.oracle.com/javase/6/docs/api/java/util/ConcurrentModificationException.html). Structural modification refers to the addition or deletion of elements from the collection.

As per the [Vector javadoc](http://docs.oracle.com/javase/7/docs/api/java/util/Vector.html) the Enumeration returned by Vector is not fail-fast. On the other side the iterator and listIterator returned by ArrayList are fail-fast.

5) **Who belongs to collection framework really?**The vector was not the part of collection framework, it has been included in collections later. It can be considered as Legacy code. There is nothing about Vector which List collection cannot do. Therefore Vector should be avoided. If there is a need of thread-safe operation make ArrayList synchronized as discussed in the next section of this post or use [CopyOnWriteArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) which is a thread-safe variant of ArrayList.

There are few **similarities between** these classes which are as follows:

1. Both Vector and ArrayList use growable array data structure.
2. The iterator and listIterator returned by these classes (Vector and ArrayList) are fail-fast.
3. They both are ordered collection classes as they maintain the elements insertion order.
4. Vector & ArrayList both allows duplicate and null values.
5. They both grows and shrinks automatically when overflow and deletion happens.

**When to use ArrayList and when to use vector?**

It totally depends on the requirement. If there is a need to perform “thread-safe” operation the vector is your best bet as it ensures that only one thread access the collection at a time.

**Performance:** Synchronized operations consumes more time compared to non-synchronized ones so if there is no need for thread safe operation, ArrayList is a better choice as performance will be improved because of the concurrent processes.

**How to make ArrayList synchronized?**  
As I stated above ArrayList methods are non-synchronized but still if there is a need you can make them synchronized like this –

//Use Collecions.synzhonizedList method

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

...

//If you wanna use iterator on the synchronized list, use it

//like this. It should be in synchronized block.

synchronized (list) {

Iterator iterator = list.iterator();

while (iterator.hasNext())

...

iterator.next();

...

}

**Q) What is the difference between ArrayList and LinkedList?**A) [ArrayList](http://beginnersbook.com/2013/12/java-arraylist/) and [LinkedList](http://beginnersbook.com/2013/12/linkedlist-in-java-with-example/) both implements List interface and their methods and results are almost identical. However there are few differences between them which make one better over another depending on the requirement.

**ArrayList Vs LinkedList**

1) **Search**: ArrayList search operation is pretty fast compared to the LinkedList search operation. get(int index) in ArrayList gives the performance of O(1) while LinkedList performance is O(n).

Reason: ArrayList maintains index based system for its elements as it uses array data structure implicitly which makes it faster for searching an element in the list. On the other side LinkedList implements **doubly linked list** which requires the traversal through all the elements for searching an element.

2) **Deletion**: LinkedList remove operation gives O(1) performance while ArrayList gives variable performance: O(n) in worst case (while removing first element) and O(1) in best case (While removing last element).

Conclusion: LinkedList element deletion is faster compared to ArrayList.

Reason: LinkedList’s each element maintains two pointers (addresses) which points to the both neighbor elements in the list. Hence removal only requires change in the pointer location in the two neighbor nodes (elements) of the node which is going to be removed. While In ArrayList all the elements need to be shifted to fill out the space created by removed element.

3) **Inserts Performance**: LinkedList add method gives O(1) performance while ArrayList gives O(n) in worst case. Reason is same as explained for remove.

4) **Memory Overhead**: ArrayList maintains indexes and element data while LinkedList maintains element data and two pointers for neighbor nodes hence the memory consumption is high in LinkedList comparatively.

There are few **similarities between** these classes which are as follows:

1. Both ArrayList and LinkedList are implementation of List interface.
2. They both maintain the elements insertion order which means while displaying ArrayList and LinkedList elements the result set would be having the same order in which the elements got inserted into the List.
3. Both these classes are non-synchronized and can be made synchronized explicitly by using [Collections.synchronizedList](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#synchronizedList(java.util.List)) method.
4. The iterator and listIterator returned by these classes are fail-fast (if list is structurally modified at any time after the iterator is created, in any way except through the iterator’s own remove or add methods, the iterator will throw a [ConcurrentModificationException](http://docs.oracle.com/javase/6/docs/api/java/util/ConcurrentModificationException.html)).

**When to use LinkedList and when to use ArrayList?**

1) As explained above the insert and remove operations give good performance (O(1)) in LinkedList compared to ArrayList(O(n)). Hence if there is a requirement of frequent addition and deletion in application then LinkedList is a best choice.

2) Search (get method) operations are fast in Arraylist (O(1)) but not in LinkedList (O(n)) so If there are less add and remove operations and more search operations requirement, ArrayList would be your best bet.

**Q) What is the difference between HashMap and Hashtable?**A**HashMap** and **Hashtable** both classes implements **java.util.Map** interface, however there are differences in the way they work and their usage. Here we will discuss the differences between these classes.

**HashMap vs Hashtable**

1) [HashMap](http://beginnersbook.com/2013/12/hashmap-in-java-with-example/) is non-synchronized. This means if it’s used in multithread environment then more than one thread can access and process the HashMap simultaneously.

[Hashtable](http://docs.oracle.com/javase/6/docs/api/java/util/Hashtable.html) is synchronized. It ensures that no more than one thread can access the Hashtable at a given moment of time. The thread which works on Hashtable acquires a lock on it to make the other threads wait till its work gets completed.

2) HashMap allows one null key and any number of null values.

Hashtable doesn’t allow null keys and null values.

3) HashMap implementation [LinkedHashMap](http://beginnersbook.com/2013/12/linkedhashmap-in-java/) maintains the insertion order and [TreeMap](http://beginnersbook.com/2013/12/treemap-in-java-with-example/) sorts the mappings based on the ascending order of keys.

Hashtable doesn’t guarantee any kind of order. It doesn’t maintain the mappings in any particular order.

4) Initially Hashtable was not the part of [collection framework](http://beginnersbook.com/java-collections-tutorials/) it has been made a collection framework member later after being retrofitted to implement the Map interface.

HashMap implements Map interface and is a part of collection framework since the beginning.

5) Another difference between these classes is that the Iterator of the HashMap is a fail-fast and it throws [ConcurrentModificationException](http://docs.oracle.com/javase/6/docs/api/java/util/ConcurrentModificationException.html) if any other Thread modifies the map structurally by adding or removing any element except iterator’s own remove() method. In Simple words fail-fast means: When calling iterator.next(), if any modification has been made between the moment the iterator was created and the moment next() is called, a ConcurrentModificationException is immediately thrown.

Enumerator for the Hashtable is not fail-fast.

For e.g.

**HashMap:**

HashMap hm= new HashMap();

....

....

Set keys = hm.keySet();

for (Object key : keys) {

//it will throw the ConcurrentModificationException here

hm.put(object & value pair here);

}

**Hashtable:**

Hashtable ht= new Hashtable();

....

.....

Enumeration keys = ht.keys();

for (Enumeration en = ht.elements() ; en.hasMoreElements() ; en.nextElement()) {

//No exception would be thrown here

ht.put(key & value pair here);

}

**When to use HashMap and Hashtable?**

1) As stated above the main difference between HashMap & Hashtable is synchronization. If there is a need of thread-safe operation then Hashtable can be used as all its methods are synchronized but it’s a legacy class and should be avoided as there is nothing about it, which cannot be done by HashMap. For [multi-thread](http://beginnersbook.com/2013/03/multithreading-in-java/) environment I would recommend you to use ConcurrentHashMap (Almost similar to Hashtable) or even you can make the HashMap synchronized explicitly ([Read here](http://beginnersbook.com/2013/12/how-to-synchronize-hashmap-in-java-with-example/)..).

2) Synchronized operation gives poor performance so it should be avoided until unless required. Hence for non-thread environment HashMap should be used without any doubt.

**Q) What is the difference between HashSet and TreeSet?**A**HashSet vs TreeSet**

1) [HashSet](http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/) gives better performance (faster) than [TreeSet](http://beginnersbook.com/2013/12/treeset-class-in-java-with-example/) for the operations like add, remove, contains, size etc. HashSet offers constant time cost while TreeSet offers log(n) time cost for such operations.

2) HashSet does not maintain any order of elements while TreeSet elements are sorted in ascending order by default.

**Similarities**:

1) Both HashSet and TreeSet does not hold duplicate elements, which means both of these are duplicate free.

2) If you want a sorted Set then it is better to add elements to HashSet and then [convert it into TreeSet](http://beginnersbook.com/2014/08/how-to-convert-a-hashset-to-a-treeset/) rather than creating a TreeSet and adding elements to it.

3) Both of these classes are non-synchronized that means they are not thread-safe and should be synchronized explicitly when there is a need of thread-safe operations.

**Examples:**

**HashSet example**

import java.util.HashSet;

class HashSetDemo{

public static void main(String[] args) {

// Create a HashSet

HashSet<String> hset = new HashSet<String>();

//add elements to HashSet

hset.add("Abhijeet");

hset.add("Ram");

hset.add("Kevin");

hset.add("Singh");

hset.add("Rick");

// Duplicate removed

hset.add("Ram");

// Displaying HashSet elements

System.out.println("HashSet contains: ");

for(String temp : hset){

System.out.println(temp);

}

}

}

**Output:**

HashSet contains:

Rick

Singh

Ram

Kevin

Abhijeet

**TreeSet example**

import java.util.TreeSet;

class TreeSetDemo{

public static void main(String[] args) {

// Create a TreeSet

TreeSet<String> tset = new TreeSet<String>();

//add elements to TreeSet

tset.add("Abhijeet");

tset.add("Ram");

tset.add("Kevin");

tset.add("Singh");

tset.add("Rick");

// Duplicate removed

tset.add("Ram");

// Displaying TreeSet elements

System.out.println("TreeSet contains: ");

for(String temp : tset){

System.out.println(temp);

}

}

}

**Output:** Elements are sorted in ascending order.

TreeSet contains:

Abhijeet

Kevin

Ram

Rick

Singh

**Q) What is the difference between ArrayList and HashMap?**A[ArrayList](http://beginnersbook.com/2013/12/java-arraylist/) and [HashMap](http://beginnersbook.com/2013/12/hashmap-in-java-with-example/) are two commonly used collection classes in Java. Even though both are the part of collection framework, the way they store and process the data is entirely different. In this post we will see the main differences between these two collections.

**ArrayList vs HashMap in Java**

1) **Implementation**: **ArrayList** implements List Interface while **HashMap** is an implementation of Map interface. List and Map are two entirely different collection interfaces.

2) **Memory consumption**: ArrayList stores the element’s value alone and internally maintains the indexes for each element.

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

//String value is stored in array list

arraylist.add("Test String");

HashMap stores key & value pair. For each value there must be a key associated in HashMap. That clearly shows that memory consumption is high in HashMap compared to the ArrayList.

HashMap<Integer, String> hmap= new HashMap<Integer, String>();

//String value stored along with the key value in hash map

hmap.put(123, "Test String");

3) **Order**: ArrayList maintains the insertion order while HashMap doesn’t. Which means ArrayList returns the list items in the same order in which they got inserted into the list. On the other side HashMap doesn’t maintain any order, the returned key-values pairs are not sorted in any kind of order.

4) **Duplicates**: ArrayList allows duplicate elements but HashMap doesn’t allow duplicate keys (It does allow duplicate values).

5) **Nulls**: ArrayList can have any number of null elements. HashMap allows one null key and any number of null values.

6) **get method**: In ArrayList we can **get** the element by specifying the index of it. In HashMap the elements is being fetched by specifying the corresponding key.

**Q) What is the difference among List, Set and Map interfaces?**A) [List, Set and Map](http://beginnersbook.com/java-collections-tutorials/) are the interfaces which implements Collection interface. Here we will discuss difference between List Set and Map in Java.

**List Vs Set Vs Map**

1) **Duplicity:** List allows duplicate elements. Any number of duplicate elements can be inserted into the list without affecting the same existing values and their indexes.  
Set doesn’t allow duplicates. Set and all of the classes which implements Set interface should have unique elements.  
Map stored the elements as key & value pair. Map doesn’t allow duplicate keys while it allows duplicate values.

2)**Null values:** List allows any number of null values.  
Set allows single null value at most.  
Map can have single null key at most and any number of null values.

3) **Order:** List and all of its implementation classes maintains the insertion order.  
Set doesn’t maintain any order; still few of its classes sort the elements in an order such as LinkedHashSet maintains the elements in insertion order.  
Similar to Set Map also doesn’t stores the elements in an order, however few of its classes does the same. For e.g. TreeMap sorts the map in the ascending order of keys and LinkedHashMap sorts the elements in the insertion order, the order in which the elements got added to the LinkedHashMap.

4) **Commonly used classes:**  
List: [ArrayList](http://beginnersbook.com/2013/12/java-arraylist/), [LinkedList](http://beginnersbook.com/2014/08/java-linkedlist-class/) etc.  
Set: [HashSet](http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/), [LinkedHashSet](http://beginnersbook.com/2013/12/linkedhashset-class-in-java-with-example/), [TreeSet](http://beginnersbook.com/2013/12/treeset-class-in-java-with-example/), SortedSet etc.  
Map: [HashMap](http://beginnersbook.com/2014/08/java-hashmap-class/), [TreeMap](http://beginnersbook.com/2013/12/treemap-in-java-with-example/), WeakHashMap, [LinkedHashMap](http://beginnersbook.com/2013/12/linkedhashmap-in-java/), IdentityHashMap etc.

**When to use List, Set and Map in Java?**

1) If you do not want to have duplicate values in the database then Set should be your first choice as all of its classes do not allow duplicates.  
2) If there is a need of frequent search operations based on the index values then List (ArrayList) is a better choice.  
3) If there is a need of maintaining the insertion order then also the List is a preferred collection interface.  
4) If the requirement is to have the key & value mappings in the database then Map is your best bet.

**Q) What is the difference between Iterator and ListIterator?**A) **Iterator vs ListIterator**

1) Iterator is used for traversing List and Set both.

We can use ListIterator to traverse List only, we cannot traverse Set using ListIterator.

2) We can traverse in only forward direction using Iterator.

Using ListIterator, we can traverse a List in both the directions (forward and Backward).

3) We cannot obtain indexes while using Iterator

We can obtain indexes at any point of time while traversing a list using ListIterator. The methods nextIndex() and previousIndex() are used for this purpose.

4) We cannot add element to collection while traversing it using Iterator, it throws ConcurrentModificationException when you try to do it.

We can add element at any point of time while traversing a list using ListIterator.

5) We cannot replace the existing element value when using Iterator.

By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods.

6) Methods of Iterator:

* hasNext()
* next()
* remove()

Methods of ListIterator:

* add(E e)
* hasNext()
* hasPrevious()
* next()
* nextIndex()
* previous()
* previousIndex()
* remove()
* set(E e)

**Q) What is the difference between Iterator and Enumeraton?**A) Here are the few differences between them:  
1) Iteator can be used on legacy(Vector etc.) classes as well as non-legacy classes (ArrayList, LinkedList etc.), whereas Enumeration is basically used for legacy classes such as Vector.  
2) Iterator is comparatively slower than Enumeration.  
3) Iterator is fail-fast wheras Enumeration is not.

**Q) How to synchronize HashMap?**  
A) [HashMap is a non-synchronized collection class](http://beginnersbook.com/2013/12/hashmap-in-java-with-example/). If we need to perform thread-safe operations on it then we must need to synchronize it explicitly. In this tutorial we will see how to synchronize HashMap.

**Example:**

In this example we have a HashMap<Integer, String> it is having integer keys and String type values. In order to synchronize it we are using [Collections.synchronizedMap(hashmap)](http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#synchronizedMap(java.util.Map))  it returns a thread-safe map backed up by the specified HashMap.

**Important point to note in the below example:**  
Iterator should be used in a synchronized block even if we have synchronized the HashMap explicitly (As we did in the below code).

**Syntax:**

Map map = Collections.synchronizedMap(new HashMap());

...

//This doesn't need to be in synchronized block

Set set = map.keySet();

// Synchronizing on map, not on set

synchronized (map) {

// Iterator must be in synchronized block

Iterator iterator = set.iterator();

while (iterator.hasNext()){

...

}

}

**Complete Code:**

package beginnersbook.com;

import java.util.Collections;

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

import java.util.Iterator;

public class HashMapSyncExample {

public static void main(String args[]) {

HashMap<Integer, String> hmap= new HashMap<Integer, String>();

hmap.put(2, "Anil");

hmap.put(44, "Ajit");

hmap.put(1, "Brad");

hmap.put(4, "Sachin");

hmap.put(88, "XYZ");

Map map= Collections.synchronizedMap(hmap);

Set set = map.entrySet();

synchronized(map){

Iterator i = set.iterator();

// Display elements

while(i.hasNext()) {

Map.Entry me = (Map.Entry)i.next();

System.out.print(me.getKey() + ": ");

System.out.println(me.getValue());

}

}

}

}

Output:

1: Brad

2: Anil

4: Sachin

88: XYZ

44: Ajit

**Q) How to synchronize ArrayList?**  
A) As we are aware that ArrayList is non-synchronized and should not be used in multi-thread environment without explicit synchronization. This post is to discuss how to synchronize ArrayList in Java.

**There are two ways to synchronize explicitly:**

1. Using Collections.synchronizedList() method
2. Using thread-safe variant of ArrayList: CopyOnWriteArrayList

**Example 1: Collections.synchronizedList() method for Synchronizing ArrayList**

In this example we are using [Collections.synchronizedList()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#synchronizedList(java.util.List)) method. The important point to note here is that iterator should be in synchronized block in this type of synchronization as shown in the below example.

package beginnersbook.com;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Collections;

public class Details {

public static void main(String a[]){

List<String> syncal =

Collections.synchronizedList(new ArrayList<String>());

//Adding elements to synchronized ArrayList

syncal.add("Pen");

syncal.add("NoteBook");

syncal.add("Ink");

System.out.println("Iterating synchronized ArrayList:");

synchronized(syncal) {

Iterator<String> iterator = syncal.iterator();

while (iterator.hasNext())

System.out.println(iterator.next());

}

}

}

Output:

Iterating synchronized ArrayList:

Pen

NoteBook

Ink

**Method 2: Using CopyOnWriteArrayList**

[CopyOnWriteArrayList](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) is a thread-safe variant of ArrayList.

package beginnersbook.com;

import java.util.concurrent.CopyOnWriteArrayList;

import java.util.Iterator;

public class Details {

public static void main(String a[]){

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

//Adding elements to synchronized ArrayList

al.add("Pen");

al.add("NoteBook");

al.add("Ink");

System.out.println("Displaying synchronized ArrayList Elements:");

//Synchronized block is not required in this method

Iterator<String> iterator = al.iterator();

while (iterator.hasNext())

System.out.println(iterator.next());

}

}

Output:

Displaying synchronized ArrayList Elements:

Pen

NoteBook

Ink

**Q) What is the difference between fail-fast and fail-safe?**  
A) fail-fast Iterators throw ConcurrentModificationException when there is a structural modification in underlying collection whereas fail-safe doesn’t throw any exception when they detect structural modification as fail-safe iterators works with clones of underlying collection.

**Q) How to convert array to ArrayList?**  
A) Here we are sharing three different ways to convert an Array to ArrayList. Basically we are converting an String Array to ArrayList of String type.

String array[] to ArrayList<String>

**Method 1: Conversion using Arrays.asList()**

Syntax:

ArrayList<T> arraylist= new ArrayList<T>(Arrays.asList(arrayname));

Example:

In this example we are using [Arrays.asList](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#asList(T...)) method to convert the Array to ArrayList.

import java.util.\*;

public class ArrayToArrayList {

public static void main(String[] args) {

/\* Array Declaration and initialization\*/

String citynames[]={"Agra", "Mysore", "Chandigarh", "Bhopal"};

/\*Array to ArrayList conversion\*/

ArrayList<String> citylist= new ArrayList<String>(Arrays.asList(citynames));

/\*Adding new elements to the converted List\*/

citylist.add("New City2");

citylist.add("New City3");

/\*Final ArrayList content display using for\*/

for (String str: citylist)

{

System.out.println(str);

}

}

}

Output:

Agra

Mysore

Chandigarh

Bhopal

New City2

New City3

**Method 2: Collections.addAll method**

[Collections.addAll](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#addAll(java.util.Collection, T...)) method all the array elements to the specified collection. This is how Collections.addAll method is being called. It does the same as Arrays.asList method however it is much faster than it so performance wise this is a best way to get the array converted to ArrayList.

String array[]={new Item(1), new Item(2), new Item(3), new Item(4)};  
ArrayList<T> arraylist = new ArrayList<T>();  
Collections.addAll(arraylist, array);

OR  
  
Collections.addAll(arraylist, new Item(1), new Item(2), new Item(3), new Item(4));  
  
Example

import java.util.\*;

public class Example2 {

public static void main(String[] args) {

/\* Array Declaration and initialization\*/

String array[]={"Hi", "Hello", "Howdy", "Bye"};

/\*ArrayList declaration\*/

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

/\*Conversion\*/

Collections.addAll(arraylist, array);

/\*Adding new elements to the converted List\*/

arraylist.add("String1");

arraylist.add("String2");

/\*Display array list\*/

for (String str: arraylist)

{

System.out.println(str);

}

}

}

Output

Hi

Hello

Howdy

Bye

String1

String2

**Method 3: Manual way of doing things**

We can also add all the array’s element to the array list manually. Below example shows the logic of manual conversion.

package beginnersbook.com;

import java.util.\*;

public class Details {

public static void main(String[] args) {

/\*ArrayList declaration\*/

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

/\*Initialized Array\*/

String array[] = {"Text1","Text2","Text3","Text4"};

/\*array.length returns the current number of

\* elements present in array\*/

for(int i =0;i<array.length;i++)

{

/\* We are adding each array's element to the ArrayList\*/

arraylist.add(array[i]);

}

/\*ArrayList content\*/

for(String str: arraylist)

{

System.out.println(str);

}

}

}

Output:

Text1

Text2

Text3

Text4

**Q) How to remove repeated elements from ArrayList?**  
A) The steps followed in the below example are:  
1) Copying all the elements of ArrayList to LinkedHashSet. Why we choose LinkedHashSet? Because it removes duplicates and maintains the insertion order.  
2) Emptying the ArrayList  
3) Copying all the elements of LinkedHashSet(non-duplicate elements) to the ArrayList. Please find below the complete code.

**Example: Removing duplicates from List**

import java.util.ArrayList;

import java.util.List;

import java.util.LinkedHashSet;

public class RemoveDuplicates {

public static void main(String[] args) {

/\* Creating ArrayList of Strings and adding

\* elements to it

\*/

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

al.add("Ajay");

al.add("Becky");

al.add("Chaitanya");

al.add("Ajay");

al.add("Rock");

al.add("Becky");

// Displaying ArrayList elements

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

System.out.println("ArrayList contains: "+al);

// Creating LinkedHashSet

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

/\* Adding ArrayList elements to the LinkedHashSet

\* in order to remove the duplicate elements and

\* to preserve the insertion order.

\*/

lhs.addAll(al);

// Removing ArrayList elements

al.clear();

// Adding LinkedHashSet elements to the ArrayList

al.addAll(lhs);

// Displaying ArrayList elements

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

System.out.println("ArrayList contains: "+al);

}

}

**Output:**

Before:

ArrayList contains: [Ajay, Becky, Chaitanya, Ajay, Rock, Becky]

After:

ArrayList contains: [Ajay, Becky, Chaitanya, Rock]

**Q) How to loop ArrayList?**  
A) There are four ways to loop ArrayList:

1. For Loop
2. Advanced for loop
3. While Loop
4. Iterator

Lets have a look at the below example – I have used all of the mentioned methods for iterating list.

import java.util.\*;

public class LoopExample {

public static void main(String[] args) {

ArrayList<Integer> arrlist = new ArrayList<Integer>();

arrlist.add(14);

arrlist.add(7);

arrlist.add(39);

arrlist.add(40);

/\* For Loop for iterating ArrayList \*/

System.out.println("For Loop");

for (int counter = 0; counter < arrlist.size(); counter++) {

System.out.println(arrlist.get(counter));

}

/\* Advanced For Loop\*/

System.out.println("Advanced For Loop");

for (Integer num : arrlist) {

System.out.println(num);

}

/\* While Loop for iterating ArrayList\*/

System.out.println("While Loop");

int count = 0;

while (arrlist.size() > count) {

System.out.println(arrlist.get(count));

count++;

}

/\*Looping Array List using Iterator\*/

System.out.println("Iterator");

Iterator iter = arrlist.iterator();

while (iter.hasNext()) {

System.out.println(iter.next());

}

}

}

Output:

For Loop

14

7

39

40

Advanced For Loop

14

7

39

40

While Loop

14

7

39

40

Iterator

14

7

39

40

.

**How to iterate arraylist elements using Enumeration interface**

import java.util.Enumeration;

import java.util.ArrayList;

import java.util.Collections;

public class EnumExample {

public static void main(String[] args) {

//create an ArrayList object

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

//Add elements to ArrayList

arrayList.add("C");

arrayList.add("C++");

arrayList.add("Java");

arrayList.add("DotNet");

arrayList.add("Perl");

// Get the Enumeration object

Enumeration<String> e = Collections.enumeration(arrayList);

// Enumerate through the ArrayList elements

System.out.println("ArrayList elements: ");

while(e.hasMoreElements())

System.out.println(e.nextElement());

}

}

Output:

ArrayList elements:

C

C++

Java

DotNet

Perl