Java Collection

144. What is the difference between Collection and Collections Framework in Java?

In Java, a Collection is an object that contains multiple elements of same type in a single unit. These multiple elements can be accessed through one Collection object.

In Java Collections Framework is a library that provides common architecture for creating, updating and accessing different types of collections. In Collections framework there are common methods that are frequently used by developers for working on a Collection object.

145. What are the main benefits of Collections Framework in Java?

Main benefits of Collections Framework in Java are as follows:

1. Reusability: Java Collections Framework provides common classes and utility methods than can be used with different types of collections. This promotes the reusability of the code. A developer does not have to re-invent the wheel by writing the same method again.

2. Quality: Using Java Collection Framework improves the program quality, since the code is already tested and used by thousands of developers.

3. Speed: Most of programmers report that their development speed increased since they can focus on core logic and use the generic collections provided by Java framework.

4. Maintenance: Since most of the Java Collections framework code is open source and API documents is widely available, it is easy to maintain the code written with the help of Java Collections framework. One developer can easily pick the code of previous developer.

146. What is the root interface of Collection hierarchy in Java?

The root interface of Collection hierarchy in Java is Collection interface.

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

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

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

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

147. What are the main differences between Collection and Collections?

Main differences between Collection and Collections are as follows:

1. Collection is an interface in Java. But Collections is a class in Java.
2. Collection is a base interface. Collections is a utility class in Java.
3. Collection defines methods that are used for data structures that contain the objects. Collections defines the methods that are used for operations like access, find etc. on a Collection.

148. What are the Thread-safe classes in Java Collections framework?

The Thread-safe classes in Java Collections framework are:

- Stack
- Properties
- Vector
- Hashtable
- BlockingQueue
- ConcurrentMap
- ConcurrentNavigableMap

149. How will you efficiently remove elements while iterating a Collection?

The right way to remove elements from a collection while iterating is by using ListIterator.remove() method.

E.g.

```
ListIterator<Integer> iter = myList.iterator();
while(iter.hasNext()) { itr.remove();
}
```

Some developers use following code to remove an element which is incorrect:

```
Iterator<Integer> iter = myList.iterator();
while(iter.hasNext()) { itr.remove();
}
```

By doing so we get ConcurrentModificationException.

An iterator is first created to traverse the list. But at the same time the list is changed by remove() method.

In Java, it is not allowed for a thread to modify a collection while another thread is iterating it. ListIterator provides the capability of removing an object during traversal.

150. How will you convert a List into an array of integers like- int[]?

We can use ArrayUtils class in Apache Commons Lang library.

Sample code is:

```
int[]intArray = ArrayUtils.toPrimitive(myList.toArray(new Integer[0]));
```

If we use List.toArray(), it will convert List to Integer[].

Another option is:

```
int[] intArray = new int[myList.size()];
for (int i=0; i < myList.size(); i++) {
  intArray [i] = myList.get(i);
}
```

151. How will you convert an array of primitive integers int[] to a List collection?

We can use ArrayUtils in Apache Commons Lang library for this purpose.

Sample code is:

```
List intList = Arrays.asList(ArrayUtils.toObject(intArray));
```

The other option would be to use a for loop and explicitly adding integers to a List.

Sample code is:

```
int[]intArray = {10,20,30};
List<Integer> intList = new ArrayList<Integer>(); for (int i:
intArray) {
  intList.add(i);
}
```

152. How will you run a filter on a Collection?

We can use CollectionUtils of Apache for this purpose. We will have to create a Predicate that will define the condition for our filter. Then we can apply this Predicate in filter() method.

Sample code is:

In this example we filter any names that are less than 5 characters long.

```
List<String> namesList = asList( "Red", "Blue", "Green" );

List<String> shortNamesList = new ArrayList<String>();
shortNamesList.addAll( namesList );

    CollectionUtils.filter( shortNamesList, new Predicate(){ public
        boolean evaluate( Object input ) {
            return ((String) input).length() < 5;
        }
      } );
```

We can also use Google Guava library for this.

In Java 8, we can use Predicate to filter a Collection through Stream.

153. How will you convert a List to a Set?

There are two ways to convert a List to a Set in Java.

Option 1: Use HashSet

```
Set<Integer> mySet = new HashSet<Integer>(myList);
```

In this case we put a list into a HashSet. Internally hashCode() method is used to identify duplicate elements.

Option 2: Use TreeSet
In this case we use our own comparator to find duplicate objects.

```
Set<Integer> mySet = new TreeSet<Integer>(myComparator);
mySet.addAll(myList);
```

154. How will you remove duplicate elements from an ArrayList?

The trick in this question is to use a collection that does not allow duplicate elements. So we use a Set for this purpose.

Option 1: Use Set
If ordering of elements is not important then we just put the elements of ArrayList in a HashSet and then add them back to the ArrayList.

Sample Code is:

```
ArrayList myList = // ArrayList with duplicate elements
Set<Integer> mySet = new HashSet<Integer>(myList);
myList.clear();
myList.addAll(mySet);
```

Option 2: Use LinkedHashSet
If ordering of elements is important then we put the elements of ArrayList in a LinkedHashSet and then add them back to the ArrayList.

Sample Code is:

```
ArrayList myList = // ArrayList with duplicate elements
Set<Integer> mySet = new LinkedHashSet<Integer>(myList);
myList.clear();
myList.addAll(mySet);
```

155. How can you maintain a Collection with elements in Sorted order?

In Java, there are many ways to maintain a Collection with elements in sorted order.

Some collections like TreeSet store elements in the natural ordering. In case of natural ordering we have to implement Comparable interface for comparing the elements.

We can also maintain custom ordering by providing a custom Comparator to a Collection.

Another option is to use the utility method Collections.sort() to sort a List. This sorting gives nlog(n) order of performance. But if we have to use this method multiple times then it will be costly on performance.

Another option is to use a PriorityQueue that provides an ordered queue. The main difference between PriorityQueue and Collections.sort() is that PriorityQueue maintains a queue in Order all the time, but we can only retrieve head element from queue. We cannot access the elements of PriorityQueue in Random order.

We can use TreeSet to maintain sorted order of elements in collection if there are no duplicate elements in collection.

156. What are the differences between the two data structures: a Vector and an ArrayList?

An ArrayList is a newer class than a Vector. A Vector is considered a legacy class in Java. The differences are:

1. Synchronization: Vector is synchronized, but the ArrayList is not synchronized. So an ArrayList has faster operations than a Vector.
2. Data Growth: Internally both an ArrayList and Vector use an array to store data. When an ArrayList is almost full it increases its size by 50% of the array size. Whereas a Vector increases it by doubling the underlying array size.

157. What are the differences between Collection and Collections in Java?

Main differences between Collection and Collections are:

1. Type: Collection is an interface in Java. Collections is a class.
2. Features: Collection interface provides basic features of data structure to List, Set and Queue interfaces. Collections is a utility class to sort and synchronize collection elements. It has polymorphic algorithms to operate on collections.
3. Method Type: Most of the methods in Collection are at instance level. Collections class has mainly static methods that can work on an instance of Collection.

158. In which scenario, LinkedList is better than ArrayList in Java?

ArrayList is more popular than LinkedList in Java due to its ease of use and random access to elements feature.

But LinkedList is better in the scenario when we do not need random access to elements or there are a lot of insertion, deletion of elements.

159. What are the differences between a List and Set collection in Java?

Main differences between a List and a Set are:

1. Order: List collection is an ordered sequence of elements. A Set is just a distinct collection of elements that is unordered.

2. Positional Access: When we use a List, we can specify where exactly we want to insert an element. In a Set there is no order, so we can insert element anywhere without worrying about order.

3. Duplicate: In a List we can store duplicate elements. A Set can hold only unique elements.

160. What are the differences between a HashSet and TreeSet collection in Java?

Main differences between a HashSet and TreeSet are:

1. Ordering: In a HashSet elements are stored in a random order. In a TreeSet, elements are stored according to natural ordering.

2. Null Value Element: We can store null value object in a HashSet. A TreeSet does not allow to add a null value object.

3. Performance: HashSet performs basic operations like add(), remove(), contains(), size() etc in a constant size time. A TreeSet performs these operations at the order of log(n) time.

4. Speed: A HashSet is better than a TreeSet in performance for most of operations like add(), remove(), contains(), size() etc .

5. Internal Structure: a HashMap in Java internally backs a HashSet. A NavigableMap backs a TreeSet internally.

6. Features: A TreeSet has more features compared to a HashSet. It has methods like pollFirst(), pollLast(), first(), last(), ceiling(), lower() etc.

7. Element Comparison: A HashSet uses equals() method for comparison. A TreeSet uses compareTo() method for comparison to maintain ordering of elements.

161. In Java, how will you decide when to use a List, Set or a Map collection?

1. If we want a Collection that does not store duplicate values, then we use a Set based collection.
2. If we want to frequently access elements operations based on an index value then we use a List based collection. E.g. ArrayList
3. If we want to maintain the insertion order of elements in a collection then we use a List based collection.
4. For fast search operation based on a key, value pair, we use a HashMap based collection.
5. If we want to maintain the elements in a sorted order, then we use a TreeSet based collection.

162. What are the differences between a HashMap and a Hashtable in Java?

Main differences between a HashMap and a Hashtable are:

1. Synchronization: HashMap is not a synchronized collection. If it is used in multi-thread environment, it may not provide thread safety. A Hashtable is a synchronized collection. Not more than one thread can access a Hashtable at a given moment of time. The thread that works on Hashtable acquires a lock on it and it makes other threads wait till its work is completed.
2. Null values: A HashMap allows only one null key and any number of null values. A Hashtable does not allow null keys and null values.
3. Ordering: A HashMap implementation by LinkedHashMap maintains the insertion order of elements. A TreeMap sorts the mappings based on the ascending order of keys. On the other hand, a Hashtable does not provide guarantee of any kind of order of elements. It does not maintain the mappings of key values in any specific order.
4. Legacy: Hashtable was not the initial part of collection framework in Java. It has been made a collection framework member, after being retrofitted to implement the Map interface. A HashMap implements Map interface and is a part of collection framework since the beginning.

5. Iterator: The Iterator of HashMap is a fail-fast and it throws ConcurrentModificationException if any other Thread modifies the map by inserting or removing any element except iterator's own remove() method.
   Enumerator of the Hashtable is not fail-fast.

163. What are the differences between a HashMap and a TreeMap?

Main differences between a HashMap and a TreeMap in Java are:

1. Order: A HashMap does not maintain any order of its keys. In a HashMap there is no guarantee that the element inserted first will be retrieved first.
2. In a TreeMap elements are stored according to natural ordering of elements. A TreeMap uses compareTo() method to store elements in a natural order.
3. Internal Implementation: A HashMap uses Hashing internally. A TreeMap internally uses Red-Black tree implementation.
4. Parent Interfaces: A HashMap implements Map interface. TreeMap implements NavigableMap interface.
5. Null values: A HashMap can store one null key and multiple null values. A TreeMap can not contain null key but it may contain multiple null values.
6. Performance: A HashMap gives constant time performance for operations like get() and put(). A TreeMap gives order of log(n) time performance for get() and put() methods.
7. Comparison: A HashMap uses equals() method to compare keys. A TreeMap uses compareTo() method for maintaining natural ordering.
8. Features: A TreeMap has more features than a HashMap. It has methods like pollFirstEntry() , pollLastEntry() , tailMap() , firstKey() , lastKey() etc. that are not provided by a HashMap.

164. What are the differences between Comparable and Comparator?

Main differences between Comparable and Comparator are:

1. Type: Comparable<T> is an interface in Java where T is the type of objects that this object may be compared to.

2. Comparator<T> is also an interface where T is the type of objects that may be compared by this comparator.

3. Sorting: In Comparable, we can only create one sort sequence. In Comparator we can create multiple sort sequences.

4. Method Used: Comparator<T> interface in Java has method public int compare (Object o1, Object o2) that returns a negative integer, zero, or a positive integer when the object o1 is less than, equal to, or greater than the object o2. A Comparable<T> interface has method public int compareTo(Object o) that returns a negative integer, zero, or a positive integer when this object is less than, equal to, or greater than the object o.

5. Objects for Comparison: The Comparator compares two objects given to it as input. Comparable interface compares "this" reference with the object given as input.

6. Package location: Comparable interface in Java is defined in java.lang package. Comparator interface in Java is defined in java.util package.

165. In Java, what is the purpose of Properties file?

A Properties file in Java is a list of key-value pairs that can be parsed by java.util.Properties class.
Generally a Properties file has extension .properties e.g. myapp.properties.
Properties files are used for many purposes in all kinds of Java applications. Some of the uses are to store configuration, initial data, application options etc.
When we change the value of a key in a properties file, there is no need to recompile the Java application. So it provides benefit of changing values at runtime.

166. What is the reason for overriding equals() method?

The equals() method in Object class is used to check whether two objects are same or not. If we want a custom implementation we can override this method.

For example, a Person class has first name, last name and age. If we want two Person objects to be equal based on name and age, then we can override equals() method to compare the first name, last name and age of Person objects.

Generally in HashMap implementation, if we want to use an object as key, then we override equals() method.

167. How does hashCode() method work in Java?

Object class in Java has hashCode() method. This method returns a hash code value, which is an integer.

The hashCode() is a native method and its implementation is not pure Java.

Java doesn't generate hashCode(). However, Object generates a HashCode based on the memory address of the instance of the object.

If two objects are same then their hashCode() is also same.

168. Is it a good idea to use Generics in collections?

Yes. A collection is a group of elements put together in an order or based on a property. Often the type of element can vary. But the properties and behavior of a Collection remains same. Therefore it is good to create a Collection with Generics so that it is type-safe and it can be used with wide variety of elements.

169. What is the difference between Collections.emptyList() and creating new instance of Collection?

In both the approaches, we get an empty list. But Collections.emptyList() returns an Immutable list. We cannot add new elements to an Immutable empty list.

Collections.emptyList() works like Singleton pattern. It does not create a new instance of List. It reuses an existing empty list instance.

Therefore, Collections.emptylist() gives better performance if we need to get an emptyList multiple times.

170. How will you copy elements from a Source List to another list?

There are two options to copy a Source List to another list.

Option 1: Use ArrayList constructor

```
ArrayList<Integer> newList = new ArrayList<Integer>(sourceList);
```

Option 2: Use Collection.copy()

To use Collections.copy() destination list should be of same or larger size than source list.

```
ArrayList<Integer> newList = new ArrayList<Integer>
(sourceList.size());
Collections.copy(newList, sourceList);
```

Collections.copy() does not reallocate the capacity of destination List if it does not have enough space to contain all elements of source List. It throws IndexOutOfBoundsException.

The benefit of Collection.copy() is that it guarantees that the copy will happen in linear time. It is also good for the scenario when we want to reuse an array instead of allocating more memory in the constructor of ArrayList.

One limitation of Collections.copy() is that it can accept only List as source and destination parameters.

171. What are the Java Collection classes that implement List interface?

Java classes that implement List interface are:

- AbstractList
- AbstractSequentialList
- ArrayList
- AttributeList
- CopyOnWriteArrayList
- LinkedList
- RoleList
- RoleUnresolvedList
- Stack
- Vector

172. What are the Java Collection classes that implement Set interface?

Java classes that implement Set interface are:

- AbstractSet
- ConcurrentSkipListSet
- CopyOnWriteArraySet
- EnumSet
- HashSet
- JobStateReasons
- LinkedHashSet
  TreeSet

173. What is the difference between an Iterator and ListIterator in Java?

Iterator and ListIterator are two interfaces in Java to traverse data structures. The differences between these two are:

1. ListIterator can be used to traverse only a List. But Iterator can be used to traverse List, Set, and Queue etc.

2. An Iterator traverses the elements in one direction only. It just goes. ListIterator can traverse the elements in two directions i.e. backward as well as forward directions.

3. Iterator cannot provide us index of an element in the Data Structure. ListIterator provides us methods like nextIndex() and previousIndex() to get the index of an element during traversal.

4. Iterator does not allow us to add an element to collection
   while traversing it. It throws ConcurrentModificationException. ListIterator allows use to add an element at any point of time while traversing a list.

5. An existing element's value cannot be replaced by using Iterator. ListIterator provides the method set(e) to replace the value of last element returned by next() or previous() methods.

174. What is the difference between Iterator and Enumeration?

Both Iterator and Enumeration are interfaces in Java to access Data Structures. The main differences between these are:

1. Enumeration is an older interface. Iterator is a newer interface.
2. Enumeration can only traverse legacy collections. Iterator can traverse both legacy as well as newer collections.
3. Enumeration does not provide remove() method. So we cannot remove any element during traversal. Iterator provides remove() method.
4. Iterator is a fail-fast interface, it gives ConcurrentModificationException if any thread tries to modify an element in the collection being iterated. Enumeration is not fail-fast.
5. Method names in Iterator are shorter than in an Enumeration.

175. What is the difference between an ArrayList and a LinkedList data structure?

Main differences between ArrayList and LinkedList data structures are:

1. **Data Structure**: An ArrayList is an indexed based dynamic array. A LinkedList is a Doubly Linked List data structure.
2. **Insertion**: It is easier to insert new elements in a LinkedList, since there is no need to resize an array. Insertion in ArrayList is O(n), since it may require resizing of array and copying its contents to new array.
3. **Remove elements**: LinkedList has better performance in removal of elements than ArrayList.
4. **Memory Usage**: LinkedList uses more memory than ArrayList, since it has to maintain links for next and previous nodes as well.
5. **Access**: LinkedList is slower in accessing an element, since we have to traverse the list one by one to access the right location.

176. What is the difference between a Set and a Map in Java?

Main differences between a Set and a Map in Java are:

1. **Duplicate Elements**: A Set does not allow inserting duplicate elements. A Map does not allow using duplicate keys, but it allows inserting duplicate values for unique keys.

2. **Null values**: A Set allows inserting maximum one null value. In a Map we can have single null key at most and any number of null values.

3. **Ordering**: A Set does not maintain any order of elements. Some of sub-classes of a Set can sort the elements in an order like LinkedHashSet. A Map does not maintain any order of its elements. Some of its sub-classes like TreeMap store elements of the map in ascending order of keys.

177. What is the use of a Dictionary class?

The Dictionary class in Java is used to store key-value pairs. Any non-null object can be used for key or value. But we cannot insert a null key or null object in Dictionary.

Dictionary class is deprecated now. So it should not be used in newer implementations.

178. What is the default size of load factor in a HashMap collection in Java?

Default value of load factor in a HashMap is 0.75.

179. What is the significance of load factor in a HashMap in Java?

A HashMap in Java has default initial capacity 16 and the load factor is 0.75f (i.e. 75% of current map size). The load factor of a HashMap is the level at which its capacity should be doubled.

For example, in a HashMap of capacity 16 and load factor .75. The capacity will become 32 when the HashMap is 75% full. Therefore, after storing the 12th key–value pair (16 * .75 = 12) into HashMap, its capacity becomes 32.

180. What are the major differences between a HashSet and a HashMap?

The main difference between a HashSet and a HashMap are:

1. **Base class**: A HashSet class implements the Set interface. Whereas a HashMap class implements the Map interface.

2. **Storage**: A HashSet is used to store distinct objects. A HashMap is used for storing key & value pairs, so that these can be retrieved by key later on.

3. **Duplicate Elements**: A HashSet does not allow storing duplicate elements. A HashMap also does not allow duplicate keys. But we can store duplicate values in a HashMap.

4. **Null Elements**: In a HashSet we can store a single null value. In a HashMap we can store single null key, but any number of null values.

5. **Element Type**: A HashSet contains only values of objects as its elements. Whereas a HashMap contains entries(key value pairs).

6. **Iteration**: By using an Iterator we can iterate a HashSet. But a HashMap has to be converted into Set for iteration.

181. What are the similarities between a HashSet and a HashMap in Java?

As the name suggests, HashSet and HashMap are Hashing based collections. Similarities between HashSet and HashMap are:

1. **Thread Safety**: Both HashMap and HashSet are not synchronized collections. Therefore they are not good for thread-safe operations. To make these thread-safe we need to explicitly use synchronized versions.

2. **Order of Elements**: None of these classes guarantee the order of elements. These are unordered collections.

3. **Internal Implementation**: A HashMap backs up a HashSet internally. So HashSet uses a HashMap for performing its operations.

4. **Performance**: Both provide constant time performance for basic operations such as insertion and removal of elements.

182. What is the reason for overriding equals() method?

The equals() method in Object class is used to check whether two objects are same or not. If we want a custom implementation we can override this method.

For example, a Person class has first name, last name and age. If we want two Person objects to be equal based on name and age, then we can override equals() method to compare the first name, last name and age of Person objects.

Generally in HashMap implementation, if we want to use an object as key, then we override equals() method.

183. How can we synchronize the elements of a List, a Set or a Map?

Sometimes we need to make collections Thread-safe for use in Multi-threading environment. In Java, Collections class provides useful static methods to make a List, Set or Map as synchronized collections. Some of these methods are:

static <T> Collection<T> synchronizedCollection(Collection<T> c)
Returns a synchronized (thread-safe) collection backed by the specified collection.

static <T> List<T> synchronizedList(List<T> list)
Returns a synchronized (thread-safe) list backed by the specified list.

static <K,V> Map<K,V>synchronizedMap(Map<K,V> m)
Returns a synchronized (thread-safe) map backed by the specified map.

static <T> Set<T> synchronizedSet(Set<T> s)
Returns a synchronized (thread-safe) set backed by the specified set.

static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m)
Returns a synchronized (thread-safe) sorted map backed by the specified sorted map.

static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s) Returns a synchronized (thread-safe) sorted set backed by the specified sorted set.

184. What is Hash Collision? How Java handles hash-collision in HashMap?

In a Hashing scenario, at times two different objects may have same HashCode but they may not be equal. Therefore, Java will face issue while storing the two different objects with same HashCode in a HashMap. This kind of situation is Hash Collision.

There are different techniques of resolving or avoiding Hash Collision. But in HashMap, Java simply replaces the Object at old Key with new Object in case of Hash Collision.

185. What are the Hash Collision resolution techniques?

To resolve a Hash Collision we can use one of the following techniques:

- Separate Chaining with Linked List
- Separate Chaining with List Head Cells
- Open Addressing with Coalesced Hashing
- Open Addressing with Cuckoo Hashing
- Hopscotch Hashing
  Robinhood Hashing

186. What is the difference between Queue and Stack data structures?

Queue is a FIFO data structure. FIFO stands for First In First Out. It means the element added first will be removed first from the queue. A real world example of Queue is a line for buying tickets at a station. The person entering first in the Queue is served first.

Stack is a LIFO data structure. LIFO stands for Last In First Out.
The element that is added last is removed first from the collection.
In a Stack elements are added or removed from the top of stack.

A real world example of Stack is back button in browser. We can go back one by one only and it works in the reverse order of adding
webpages to history .

187. What is an Iterator in Java?

Iterator is an interface in Java to access the elements in a collection.

It is in java.util package.
It provides methods to iterate over a Collection class in Java.
Iterator interface in Java is based on Iterator design pattern. By using an Iterator one can traverse a container of objects and can also access the objects in the container. A container of objects is a Collection class in Java.

188. What is the difference between Iterator and Enumeration in Java?

Main differences between Iterator and Enumeration in Java are:

1. **Version**: Enumeration interface is in Java since JDK 1.0. Iterator interface was introduced in Java 1.2.

2. **remove() method**: The main difference between Enumeration and Iterator interface is remove() method. Enumeration can just traverse a Collection object. If we use Enumeration, we cannot do any modifications to a Collection while traversing the collection. Iterator interface provides remove() method to remove an element while traversing the Collection. There is not remove() method in Enumeration interface.

3. **Method names**: Names of methods in Iterator interface are hasNext(), next(), remove(). Names of methods in
Enumeration interface are hasMoreElements(), nextElement().

4. **Legacy Interface**: Enumeration is considered as a legacy interface. It is used to traverse legacy classes like Vector, Stack and HashTable. Iterator is a newer interface that is used to traverse almost all of the classes in Java Collections framework.

5. **Fail-fast vs. Fail-safe**: Iterator is based on fail-fast principle. It throws ConcurrentModificationException if a collection is modified during iteration over that collection. An Enumeration is based on fail-safe principle. It doesn't throw any exception if a collection is modified during traversal.

6. **Safety**: Since Iterator is fail-fast and does not allow modification of a collection by other threads, it is considered safer than Enumeration.

189. What is the design pattern used in the implementation of Enumeration in Java?

Enumeration is based on Iterator design pattern. Iterator design pattern provides a common interface with methods to traverse the collection of objects. It hides the underlying implementation details of the collection.

190. Which methods do we need to override to use an object as key in a HashMap?

If we want to use an object as a key in a HashMap in Java, then we have to make sure that it has the implementation of equals() and hashCode() methods.

191. How will you reverse a List in Java?

In Collections class, Java provides a method reverse(List list) that can be used to reverse a List.

E.g.
Collections.reverse(myList);

192. How will you convert an array of String objects into a List?

Java provides Arrays class in java.util package. Arrays class has a method asList() that accepts an Array as input and returns a List as output.

public static <T> List<T> asList(T... a)

String[] myArray = {"George" , "Jack" , "Ryan"}; List myList
= Arrays.asList(myArray);

193. What is the difference between peek(), poll() and remove() methods of Queue interface in java?

In a Java Queue, poll() and remove() methods can be used for removing the head object of Queue. The main difference arises in the case when Queue is empty().

If Queue is empty then poll() method returns null value. If Queue is empty then remove() method throws NoSuchElementException.

In a Java Queue, peek() method retrieves the head of Queue but it does not remove it. If queue is empty then peek() method returns null value.

194. What is the difference between Array and ArrayList in Java?

The main differences between Array and ArrayList in Java are:

1. **Size**: Array in Java is fixed in size. We cannot change the size of array after creating it. ArrayList is dynamic in size. When we add elements to an ArrayList, its capacity increases automatically.

2. **Performance**: In Java Array and ArrayList give different performance for different operations.

3. **add() or get():** Adding an element to or retrieving an element from an array or ArrayList object has similar performance. These are constant time operations.

4. **resize():** Automatic resize of ArrayList slows down the performance. ArrayList is internally backed by an Array. In resize() a temporary array is used to copy elements from old array to new array.

5. **Primitives**: Array can contain both primitive data types as well as objects. But ArrayList cannot contain primitive data types. It contains only objects.

6. **Iterator**: In an ArrayList we use an Iterator object to traverse the elements. We use for loop for iterating elements in an array.

7. **Type Safety**: Java helps in ensuring Type Safety of elements in an ArrayList by using Generics. An Array can
contain objects of same type of class. If we try to store a different data type object in an Array then it throws ArrayStoreException.

8. **Length**: Size of ArrayList can be obtained by using size() method. Every array object has length variable that is same as the length/size of the array.

9. **Adding elements**: In an ArrayList we can use add() method to add objects. In an Array assignment operator is used for adding elements.

10. Multi-dimension: An Array can be multi-dimensional. An ArrayList is always of single dimension.

195. How will you insert, delete and retrieve elements from a HashMap collection in Java?

We use following methods to insert, delete and retrieve elements in a HashMap.

1. **Retrieve**: We use get() method to retrieve elements from a HashMap.
   Value get(Object key)

2. **Insert**: We use put() method to insert a key value pair in a HashMap.
   Value put(Key k, Value v)

3. **Delete**: We use remove() method to delete key-value pair from the HashMap.
   Value remove(Object key)

196. What are the main differences between HashMap and ConcurrentHashMap in Java?

Main differences between HashMap and ConcurrentHashMap are:

1. **Synchronization**: A HashMap is not synchronized. But a ConcurrentHashMap is a synchronized object.
2. **Null Key**: A HashMap can have one null key and any number of null values. A ConcurrentHashMap cannot have null keys or null values.
3. **Multi-threading**: A ConcurrentHashMap works well in a multi-threading environment.

197. What is the increasing order of performance for following collection classes in Java?

The increasing order of performance is:

- Hashtable
- Collections.SynchronizedMap
- ConcurrentHashMap
- HashMap

Hashtable has the worst performance and HashMap has the best performance.

198. Why does Map interface not extend Collection interface in Java?

A Map is a collection objects. But Map interface is not compatible with Collection interface in Java.

A Map requires key as well as a value. So it requires two parameters to add an element to a HashMap.

But Collection interface provides add(Object o) method with only one parameter.

Map collection has to provide methods like valueSet, keySet etc. These methods are specific to Map collection. Where as methods in Collection interface can be reused by a List, Set, Queue etc.

199. What are the different ways to iterate elements of a list in Java?

There are mainly two ways to iterate the elements of list in Java:

1. **Iterator**: We can get an Iterator for list and use it to iterate the objects of the list.

2. **For-each loop**: We can use for-each loop to traverse all the elements of a list.

200. What is CopyOnWriteArrayList? How it is different from ArrayList in Java?

CopyOnWriteArrayList was introduced in Java 5 version. It is a thread-safe collection. It is similar to an ArrayList.

In CopyOnWriteArrayList, all mutative operations (add, set etc.) are implemented by making a fresh copy of the underlying array.

Iterator of CopyOnWriteArrayList is guaranteed to not throw ConcurrentModificationException. But Iterator also does not reflect any additions, removals that happened to list after the Iterator was created.

All elements including null are permitted in CopyOnWriteArrayList.

201. How remove() method is implemented in a HashMap?

Remove() method in HashMap uses logic similar to the one used in get() method. First we locate the correct bucket in HashMap for an entry. Then within that bucket we remove the element e. It is similar to removing a node from a single-linked list.

If e is the first element in the bucket we set the corresponding element of Hash to e.next. Else we set the next field of the element just before e to e.next.

202. What is BlockingQueue in Java Collections?

BlockingQueue was introduced in Java 1.5. It extends Queue interface in Java.

BlockingQueue supports operations that wait for the queue to become non-empty when retrieving an element. Also it supports the operations that wait for space to become available in the queue while storing an element.

Some of the features of BlockingQueue are:

- It does not accept null elements.
- Its main use is in producer-consumer problems.
- BlockingQueue implementation is thread-safe.
- It can be used in inter-thread communications.
- It does not support any kind of "close" or "shutdown" operation to indicate that no more items will be added.

203. How is TreeMap class implemented in Java?

Internally, a TreeMap class in Java uses Red-Black tree.

It is a NavigableMap. The map sorts the keys in natural order or it can use a Comparator supplied at the creation time.

The implementation of TreeMap is not synchronized in Java.

204. What is the difference between Fail-fast and Fail-safe iterator in Java?

Differences between Fail-fast and Fail-safe iterators are as follows:

Fail-fast iterator throws ConcurrentModificationException. But Fail-safe iterator does not throw this exception.

Fail-fast iterator does not clone the original collection. Fail-safe iterator creates a copy of the original collection of objects.

A Fail-fast iterator tries to immediately throw Exception when it encounters failure. A Fail-safe Iterator works on a copy of collection instead of original collection.

205. How does ConcurrentHashMap work in Java?

ConcurrentHashMap extends AbstractMap in Java. It was introduced in Java 1.5. It provides concurrency in a collection based on a HashMap.

All methods are thread-safe in ConcurrentHashMap.

Internally there is a Hashtable backing a ConcurrentHashMap. This Hashtable supports the concurrent methods for retrieval of data as well as updates on ConcurrentHashMap.

It has same functional specification as a Hashtable.

It also supports a set of sequential and bulk operations. These operations accept parallelismThreshold argument.

206. What is the importance of hashCode() and equals() methods?

In a HashMap collection it is very important for a key object to implement hashCode() method and equals() method. If hashCode() method returns same hashcode for all key objects then the hash collision will be high in HashMap. Also with same hashcode, we will get same equals method that will make our HashMap inefficient.

The problem arises when HashMap treats both outputs same instead of different. It will overwrite the most recent key-value pair with the previous key-value pair.

So it is important to implement hashCode() and equals() methods correctly for an efficient HashMap collection.

207. What is the contract of hashCode() and equals() methods in Java?

Contract of hashCode() and equals() methods is as follows in Java:

If object1.equals(object2), then object1.hashCode() == object2.hashCode() should always be true. It means if two objects are equal then their hashCode should be same.

If object1.hashCode() == object2.hashCode() is true, it does not guarantee that object1.equals(object2). It means if two objects have same hashCode, then can still have different values so that may not be equal objects.

208. What is an EnumSet in Java?

Set: EnumSet is a specialized implementation of Set.

1. **Use**: It is mainly used with enum types.
2. **Single enum type**: All the elements in an EnumSet must come from a single enum type when the set is created.
3. **Bit vector**: Internally, EnumSet is represented as bit vector.
4. **Iterator**: The iterator of EnumSet traverses the elements in their natural order. (It is the order in which the enum constants are declared).
5. **Null**: In an EnumSet, null elements are not permitted. If we try to insert a null element it throws NullPointerException.
6. **Thread-safe**: EnumSet is not a synchronized collection. For use in multi-threading scenarios, EnumSet should be synchronized.
7. **Bit flags**: EnumSet is a very good alternative to int based “bit flags” implementation.

209. What are the main Concurrent Collection classes in Java?

Java 1.5 has provided new package java.util.concurrent. This package contains thread-safe collection classed. These collection classes can be modified while iterating. The iterator of these classes is fail-safe.

Main Concurrent Collection classes in Java 8 are:

- ArrayBlockingQueue
- CopyOnWriteArrayList
- CopyOnWriteArraySet
- ConcurrentHashMap
- ConcurrentLinkedDeque
- ConcurrentLinkedQueue
- LinkedBlockingQueue
- LinkedBlockingDeque
- PriorityBlockingQueue

210. How will you convert a Collection to SynchronizedCollection in Java?

Java provides an easy method in java.utils.Collections class to create a ThreadSafe collection from a regular collection.

We can use the method synchronizedCollection() for this purpose.

For any class of type T we can use following method:

static <T> Collection<T> synchronizedCollection(Collection<T> c)

211. How IdentityHashMap is different from a regular Map in Java?

IndentityHashMap in Java implements Map interface. But it is not a general purpose implementation. It violates the general contract of Map interface by a different implementation of equals() method.

In an IdentityHashMap, two keys k1 and k2 are equal if and only if (k1==k2). (In a normal Map implementation (like HashMap) two keys k1 and k2 are considered equal if and only if (k1==null ? k2==null : k1.equals(k2)).)

It implements the Map interface with a hash table, using reference-equality in place of object-equality when comparing keys (and values).

212. What is the main use of IdentityHashMap?

Main uses of IdentityHashMap are:

1. Topology Preservation: The typical use of IdentityHashMap class is topology-preserving object graph transformations, such as serialization or deep-copying. In such a scenario, a program must maintain a "node table" to keep track of all the object references that have already been processed.

2. The node table should not considered distinct objects as equal even if they happen to be equal.

3. Proxy objects: Another use of this class is to maintain proxy objects. A debugging program has to maintain a proxy object for each object in the program being debugged.

213. How can we improve the performance of IdentityHashMap?

IdentityHashMap class has one tuning parameter for performance improvement: expectedMaxSize.

This parameter is the maximum number of key-value mappings that the map is expected to hold.

We can use this parameter is used to determine the number of buckets initially in the hash table. The precise relationship between the expected maximum size and the number of buckets is unspecified.

If the number of key-value mappings exceeds the expected maximum size, the number of buckets is increased.

Increasing the number of buckets is also known as rehashing. Rehashing may be fairly expensive. So it is better to create identity hash maps with a sufficiently large expected maximum size.

But iteration over a Map collection requires time proportional to the number of buckets in the hash table. So iteration may take extra time due to large number of buckets.

Therefore the value of expectedMaxSize should be set in consideration with both of these aspects.

214. Is IdentityHashMap thread-safe?

The implementation of IdentityHashMap is not thread-safe, since its methods are not synchronized.

The iterators returned by the iterator method of IdentityHashMap are fail-fast. But the fail-fast behavior of an iterator cannot be guaranteed.

Since the Iterator is fail-fast, it throws ConcurrentModificationException.

215. What is a WeakHashMap in Java?

WeakHashMap is a class similar to IdentityHashMap.
Internally, it is represented by a Hashtable.
It is not a synchronized class. We can make a WeakHashMap thread safe by using Collections.synchronizedMap() method.
An entry in WeakHashMap is automatically removed when it is no longer in ordinary use.
The presence of a mapping for a given key does not prevent the key from being discarded by the garbage collector.
WeakHashMap also permits null keys and null values.

216. How can you make a Collection class read Only in Java?

In Java, there are useful methods to make a Collection class read Only. We can make the Collection read Only by using one of the following methods:

- Collections.unmodifiableMap(Map m)
- Collections.unmodifiableList(List l)
- Collections.unmodifiableSet(Set s)
  Collections.unmodifiableCollection(Collection c)

217. When is UnsupportedOperationException thrown in Java?

In a Java collection UnsupportedOperationException is thrown when the requested operation is not supported by the collection.
It is an unchecked exception that is thrown on optional operations.
If there is an optional add() or remove() methods in a read only collection, then this exception can be thrown.

218. Let say there is a Customer class. We add objects of Customer class to an ArrayList. How can we sort the Customer objects in ArrayList by using customer firstName attribute of Customer class?

There are two ways to handle this scenario. We can use these options:
Comparable: Implement the Comparable interface for Customer class and compare customer objects by firstName attribute.

Comparator: Implement Comparator for comparing two Customer objects on the basis of firstName attribute. Then use this comparator object in sort method of Collections class.

219. What is the difference between Synchronized Collection and Concurrent Collection?

In Java 1.5 many Concurrent collection classes were added in SDK.
These are ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc.

Java also provides utility methods to get a synchronized copy of collection like ArrayList, HashMap etc. by using
Collections.synchronizedList(), Collections.synchronizedMap() methods.

The main difference is in performance. Concurrent collection classes have better performance than synchronized collection classes because they lock only a portion of the class to achieve concurrency and thread-safety.

220. What is the scenario to use ConcurrentHashMap in Java?

ConcurrentHashMap is more suited for scenarios where we have multiple reader threads and one writer thread. In this case map is locked only during the write operation.

If we have an equal number of reader and writer threads then ConcurrentHashMap performance is similar to a Hashtable or a synchronized HashMap.

221. How will you create an empty Map in Java?

There are two ways to create an empty Map in Java.

1. **Immutable**: If we want an immutable empty Map, we can use following code:

```
myMap = Collections.emptyMap();
```

2. **Any map**: For all other scenarios, we can use following code by using new method:

```
myMap = new HashMap();
```

222. What is the difference between remove() method of Collection and remove() method of Iterator?

In Collection interface remove(Object o) method is used to remove objects from a Collection.

List interface also provides remove(int index) method to remove an object at a specific index.

These methods are used to remove an entry from Collection, while no thread is iterating over it.

When we are iterating over a Collection, then we have to remove() method of Iterator. This method removes current element from Iterator's point of view. If we use remove(0 method of Collection or List, then we will get ConcurrentModificationException.

Therefore, it is recommended to use remove() method of Iterator during the traversal of a Collection by an Iterator.

223. Between an Array and ArrayList, which one is the preferred collection for storing objects?

An ArrayList is backed up by array internally. There are many usability advantages of using an ArrayList over an array in Java.

Array has a fixed length at the time of creation. Once it is created we cannot change its length.

ArrayList is dynamic in size. Once it reaches a threshold, it automatically allocates a new array and copies contents of old array to new array.

Also ArrayList provides support of Generics. But Array does not support Generics.

E.g. If we store an Integer object in a String array at Runtime it will throw ArrayStoreException. Whereas, if we use ArrayList then as compile time we will get the error. This helps in preventing errors from happening at runtime.

If we know the size in advance and do not need re-sizing the collection then Array should be used in place of an ArrayList.

224. Is it possible to replace Hashtable with ConcurrentHashMap in Java?

Yes, a ConcurrentHashMap can be replaced with Hashtable in Java.

But it requires careful observation, since locking behavior of Hashtable is different than that of ConcurrentHashmap.

A Hashtable locks whole Map instead of a portion of Map. Compound operations like if(Hashtable.get(key) == null) put(key, value) work in Hashtable but not in ConcurrentHashMap.

In a ConcurrentHashMap we use putIfAbsent() method for such a scenario.

225. How CopyOnWriteArrayList class is different from ArrayList and Vector classes?

CopyOnWriteArrayList was introduced in Java 1.5. It implements List interface.

It provides better concurrent access methods than a Synchronized List.

In CopyOnWriteList, concurrency is achieved by copying ArrayList over each write and replace with original instead of locking.

CopyOnWriteArrayList also does not throw any ConcurrentModification Exception during Iteration.

It is a thread-safe list.

It is different from a Vector in terms of Concurrency. CopyOnWriteArrayList provides better Concurrency by reducing contention among readers and writers.

226. Why ListIterator has add() method but Iterator does not have?

ListIterator can iterate in the both directions of a Collection. It maintains two pointer for previous and next element. In ListIterator we can use add() method to add an element into the list immediately before the element returned by next() method.

So a subsequent call to next() method will not be affected. And the call to previous() method will return the newly added element.

In Iterator we can only traverse in one direction. So there is no purpose of add() method there.

227. Why do we sometime get ConcurrentModificationException during iteration?

When we remove an object by using remove() method of a Collection or List while an Iterator thread is traversing it, we get ConcurrentModificationException. If an Iterator detects any
structural change in Collection it can throw ConcurrentModificationException.

228. How will you convert a Map to a List in Java?

In Java, a Map has three collection sets:

key set
value set
key-value set

Each of these Sets can be converted to List by using a constructor.

Sample code is as follows:

List keyList = new ArrayList(map.keySet()); List
valueList = new ArrayList(map.values()); List entryList =
new ArrayList(map.entrySet());

229. How can we create a Map with reverse view and lookup in Java?

In a Map we can lookup for a value by using a distinct key. In a Map with reverse view and lookup, even the values are distinct. So there is one to one mapping between keys and values and vice version.

If we enable this constraint on a Map then we can look up a key by its value. Such data structure is called bi-directional map.

There is no built data structure similar to reverse lookup Map in JDK.

But Apache Common Collections and Guava libraries provide implementation of bidirectional map. It is called BidiMap and BiMap. Both of these data structure enforce the constraint of one to one mapping between keys and values.

230. How will you create a shallow copy of a Map?

In Java, most implementations of Map interface provide a constructor to create copy of another map. But the copy method is not synchronized.

Therefore, when a thread is copying the map, another thread can modify it.

To prevent such a scenario, we should use Collections.synchronizedMap() method to first create a thread-safe map.

Another way of to create a shallow copy is by using clone() method.
But it is not considered as a recommended approach.

231. Why we cannot create a generic array in Java?

Java does not allow creation of array with generics as elements.

In Java an array has to know the type information of its elements at runtime.

This information is used at runtime to throw ArrayStoreException if data type of an element to be inserted does not match the type of Array.

In case of Generics, the type information of a collection is erased at runtime by Type Erasure. Due to this array cannot use generics as elements.

232. **What is a PriorityQueue in Java?**

A PriorityQueue is data structure based on Queue. Unlike Queue, the elements on PriorityQueue are not returned in FIFO order.
A PriorityQueue maintains the natural order of its elements or it uses a Comparator provided at initialization.
It is an unbounded queue based on a priority heap.
PriorityQueue does not allow null values. We cannot add any object that does not provide natural ordering to PriorityQueue.
PriorityQueue in Java is not thread-safe.
It gives O(log n) time for enqueing and dequeing operations.

233. What are the important points to remember while using Java Collections Framework?

Some of the important points to remember while using Java Collections Framework are:

1. **Interfaces**: For Collections, we should write code with generic interfaces instead of concrete implementation. Due to this we maintain the flexibility of changing the implementation at a later point of time.

2. **Generics**: We should use Generics for type-safety and to avoid ClassCastException at runtime.
3. **Collections**: It is recommended to use Collections utility class for algorithms and various other common methods for Collections.
4. **Right Type**: We have to choose the right type of Java collection based on our need. If size is fixed, we can use Array over ArrayList. If we do not want duplicate elements we use Set.
   If we need the ability to iterate the elements of a Map in the order of insertion then we use a TreeMap.
5. **Initial Size**: In some collection classes we can specify the initial size/capacity. Therefore we should have an estimate of number of elements in a Collection before deciding the right collection type. We can use it to avoid rehashing or resizing.
6. **Map**: We should use immutable classes provided by Java as key elements in a Map.

234. How can we pass a Collection as an argument to a method and ensure that method will not be able to modify it?

To ensure that a method is not able to modify a Collection passed as an argument, we have to make the Collection read only.

We can make a read only collection by using Collections.unmodifiableCollection(Collection c) method.

This will make sure that any operation to change the collection will throw UnsupportedOperationException.

235. Can you explain how HashMap works in Java?

In Java, a HashMap works on the concept of hashing.

A HashMap in Java stores both key and value objects, in a bucket. It is stored as an Entry object that implements Map.Entry interface.

The key object used in a HashMap has to provide implementation for hashCode() and equals() methods.

When put() method is used to store a key-value pair, the HashMap implementation calls hashCode() method on Key object to calculate a hash that is used to find a bucket where Entry object will be stored.

When get() method is used to retrieve a value stored against a key object, we first calculate a hash of Key object. Then we use this hash to find the bucket in which that particular key is stored.

Once Key object's location is found, it may happen that more than one Key is stored in same location. So now we use equals() method to find the exact Key object. Once the exact Key object is found we use it to get Value object.

236. Can you explain how HashSet is implemented in Java?

Internally, a HashSet uses a HashMap to store the elements and to maintain the uniqueness of elements.

When we create a HashSet object, a corresponding HashMap object is also created.

When we insert an element in HashSet, it inserts it into corresponding HashMap.

237. What is a NavigableMap in Java?

As the name suggests, NavigableMap provides the capability to navigate the keys of a Map in Java. A NavigableMap extends SortedMap interface.

Some of the interesting methods of a NavigableMap are descendingKeySet(), descendingMap(), headMap() and tailMap().

238. What is the difference between descendingKeySet() and descendingMap() methods of NavigableMap?

The descendingKeySet() method of NavigableMap returns a NavigableSet in which the elements are stored in reversed order as compared to the original key set.

The returned view is internally represented by the original KeySet of NavigableMap. Therefore any changes to the descending set also get reflected in the original set.

But it is not recommended to remove elements directly from the key set. We should use the Map.remove() method.

The descendingMap() method of NavigableMap returns a NavigableMap which is an inverse view of the original Map. The order of the elements in this view are in reverse order of the elements in original map. Any changes to this view are also reflected in the original map.

239. What is the advantage of NavigableMap over Map?

The main advantage of NavigableMap over Map is the Navigation capability.

It provides the capabilities of a Map, SortedMap and navigation in one collection.

It even returns the closest matches for given search targets.

Methods like lowerEntry, floorEntry, ceilingEntry, and higherEntry return Map.Entry objects associated with keys respectively less than, less than or equal, greater than or equal, and greater than a given key.

Methods like lowerKey, floorKey, ceilingKey, and higherKey return only the associated keys. All of these methods are designed for locating, not traversing entries.

240. What is the difference between headMap(), tailMap() and subMap() methods of NavigableMap?

The headMap() method returns a view of the original NavigableMap that contains the elements that are less than a given element.

```
NavigableMap original = new TreeMap();
original.put("1", "1"); original.put("2", "2");
original.put("3", "3");

//this headmap1 will contain elements "1" and "2"
SortedMap headmap1 = original.headMap("3");

//this headmap2 will contain elements "1", "2", and "3" because "inclusive"=true
        NavigableMap headmap2 = original.headMap("3", true);
```

The tailMap() method works similar to headMap() method, but it returns all elements that are higher than the given input element.

The subMap() method accepts two parameters demarcating the boundaries of the view map to return.

All the three methods return a subset of the original map in a view form.

241. How will you sort objects by Natural order in a Java List?

We can use Collections.sort method to sort the elements of a List in natural order. To use this method, we have to make sure that element objects implement compareTo() method.

We can also use a Comparator to define the natural ordering for elements of a List. Then we can use this Custom Comparator in sort method of Collections class.

242. How can we get a Stream from a List in Java?

From Java 8 onwards it is a very easy to get a Stream from a List. We can just use stream() method to get a stream from a list of elements.

243. Can we get a Map from a Stream in Java?

Yes, we can create a Map from the elements of a Stream. We can use map() method to get a Map.

```
E.g. items.stream()
     .map( item -> item.toLowerCase() )
```

In this example we are creating a map with each item object mapped to its LowerCase equivalent.

This is also used in Map-Reduce implementation on a Stream.

244. What are the popular implementations of Deque in Java?

The two most popular implementation of Deque interface in Java are:

1. **ArrayDeque**: It is a resizable array implementation of Deque. The capacity of ArrayDeque can increase based on the need of the program. It is not thread safe implementation. Also the iterator on ArrayDeque is fail-fast.

2. **LinkedList**: This is another popular implementation of Deque interface in Java. It is also not synchronized, so it is not thread-safe. It mainly provides functionality of a doubly linked list.