Collection Framework

Q1: What are the limitations of Arrays?

1. Arrays are fixed in size i.e., once we create an array, there is no chance of increasing or decreasing. Suppose if we don’t know the size of our requirement in advance, then Arrays are not recommended to use.
2. Arrays can hold only homogeneous data elements.
3. There is no underlying data structure for Arrays. Hence readymade method support is not available. So for every requirement compulsory we have to write code.

For ex: to sort the elements in array,

Arrays.sort(a);

But the elements which will be added next will be inserted based on index only but not is sorting order.

In array, it is not possible to add elements in an order(for ex sorting).

To resolve the above three problems Sun people introduced Collections concept.

Q2: What are the advantages of Collections?

1. Basically Collections are grow able in nature i.e., based on our requirement we can increase or decrease the capacity.
2. For every Collection, an underlying data structure is there. So for every requirement, readymade method support is available.

For ex: isEmpty(), contains(), remove(object), clear()

1. So being a programmer, just we have to know how to use that functionality and we are not responsible to define that functionality.

The main limitation of collection is:

It effects the performance of the system.

Collection is grow able in nature doesn’t mean collection object will be grown.

Collection object will be created with some initial capacity, if elements are added more than capacity then a new collection object will be created with new capacity and elements will be copied from old object to new object which effects the performance of the system.

Q3: What are the differences between Arrays and Collections?

|  |  |
| --- | --- |
| Arrays | Collections |
| 1.Arrays are fixed in size. | 1.growable in nature. |
| 2.memory point of view, Arrays are not recommended to use. | 2.memory point of view, Collections are  recommended to use. |
| 3.performance point of view, Arrays are recommended to use. | 3.performance point of view, Collections are not recommended to use. |
| 4.holds homogeneous elements only. | 4.holds both homogeneous and heterogeneous elements. |
| 5.for every requirement, readymade methods are not available because there is no underlying data structure available in this. | 5.for every requirement, readymade methods are available because there is an underlying data structure available for every collection. |
| 6.Arrays can hold both primitives and objects. | 6.Collections can hold only objects, but not primitives. |

Q4:What is Collection Framework?

It defines a group of classes and interfaces which are used to represent a group of objects as a single entity.

Q5:What about Collection Interface?

1. Generally this interface acts as a root interface of Collection Framework.
2. This is used to represent a group of individual objects as a single entity.
3. This interface has several utility abstract methods which can be apply on any kind of collection object.

Q6:What about Collections class?

Generally Collections is a concrete class, has some utility methods which are useful for the Collection implemented objects.

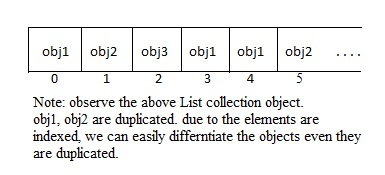
Whereas Collection is an interface which is used to represent a group of individual objects as a single entity.

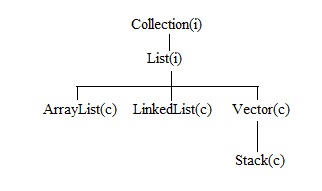
Q7:What about the List?

It is the child interface of Collection interface and it can be used for representing a group of individual objects as a single entity, where elements(here objects) insertion order is preserved and duplicate objects are allowed.

In this, generally elements(i.e objects) are indexed. That’s the reason it can differentiate the duplicate objects.

Its structure is like the following:

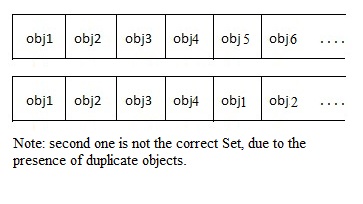


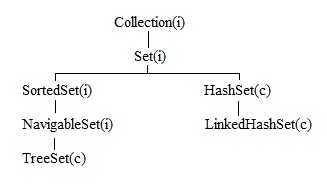


Q8:What about the Set?

It is the child interface of Collection interface and it can be used for representing a group of individual objects as a single entity, where elements(here objects) insertion order is not preserved and duplicate objects are not allowed.

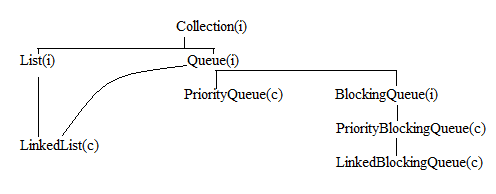
In this, generally elements(i.e objects) are not indexed. That’s the reason it can’t differentiate the duplicate objects.





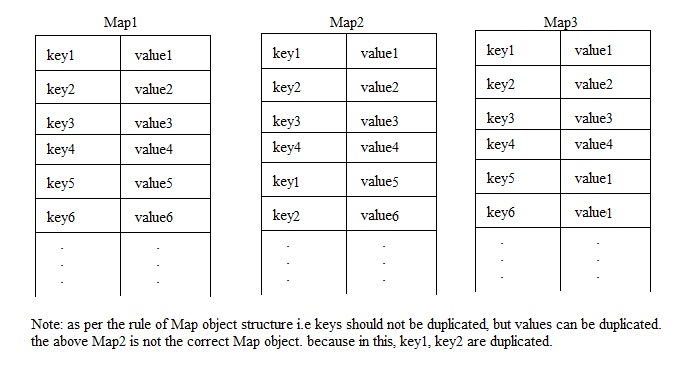
Q9:What about the Queue?

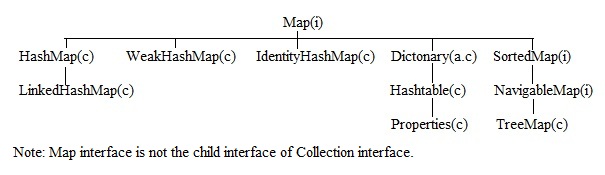
It is the child interface of Collection interface and it can be used for representing a group of individual objects as a single entity, where we want to represent group of objects prior to processing. For this, it follows FIFO or we can customize based on our requirement.



Q10:What about the Map?

It is not the child interface of Collection interface and it can be used for representing a group of individual objects as key & value pairs. Both key and value should be objects. Duplicate keys are not allowed, but values can be duplicated. In Map, each key-value pair is considered as an entry. Hence a map is considered as set of entries.





Q11:tell me about the ArrayList?

1. It is a class implemented from List interface.
2. The underlying data structure is resizable Array and grow able Array.
3. Insertion order is preserved.
4. Duplicate objects are allowed.
5. Heterogeneous objects are allowed.
6. Null insertion is possible.
7. This class is implemented from the Serializable, Cloneable and RandomAccess marker interfaces.
8. ArrayList is not recommended to use, if frequent operation is insertion or deletion in the middle.(for this, LinkedList is recommended).
9. ArrayList is recommended to use, if frequent operation is retrieval. (because it is implemented the RandomAccess interface).

Note:

ArrayList is recommended if the frequent operation is retrieval.

LinkedList is recommended if the frequent operation is insertion or deletion.

Q12:What is the RandomAccess interface and its purpose?

It is a marker interface, present in java.util package, doesn’t contain any methods. ArrayList, Vector classes implement this interface, hence we can access any random element with the same speed, hence if frequent operation is retrieval then it is recommended to use the ArrayList.

Q13:tell me about the LinkedList?

1. It is a class implemented from List interface.
2. The underlying data structure is double linked list.
3. Insertion order is preserved.
4. Duplicate objects are allowed.
5. Heterogeneous objects are allowed.
6. Null insertion is possible.
7. This class is implemented from the Serializable, Cloneable interfaces.but not the RandomAccess interface.
8. LinkedList is not recommended to use, if frequent operation is retrieval.(for this, ArrayList is recommended).
9. LinkedList is recommended to use, if frequent operation is insertion or deletion in the middle.

Note:

ArrayList is recommended if the frequent operation is retrieval.

LinkedList is recommended if the frequent operation is insertion or deletion.

Q14:What are the differences between ArrayList and LinkedList?

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| The underlying data structure is resizable Array and grow able Array. | The underlying data structure is double linked list. |
| it is recommended if the frequent operation is retrieval. | it is recommended if the frequent operation is insertion or deletion. |
| it is implemented from Serializable, Cloneable and RandomAccess marker interfaces. | it is implemented from Serializable, Cloneable marker interfaces but not RandomAccess marker interface. |

Q15:What are the differences between ArrayList and Vector?

|  |  |
| --- | --- |
| ArrayList | Vector |
| 1. No method is synchronized. | 1. All methods are synchronized. |
| 1. ArrayList object is not thread safe. | 1. Vector object is always thread safe. |
| 1. Performance is high. | 1. Performance is low. |
| 1. Introduced in 1.2 version and hence it is non legacy. | 1. Introduced in 1.0 version and hence it is legacy. |

Q16:how can we get synchronized version of ArrayList?

Collections class contains synchronizedList(List l).

Ex: ArrayList l1=new ArrayList();

List l2=Collections.synchronizedList(l1);

Similarly, we can get synchronized version of Set & Map by using the following methods.

i.e synchronizedSet(Set s), synchronizedMap(Map m).

Q17.What is the difference between size and capacity?

Size represent the no of objects present in a Collection object.

Capacity represents the no of objects that a Collection object can hold.

Q18.What is a cursor and how many cursors are there in java?

To retrieve object by object from any collection object, we go for cursors.

There are three cursors in java. They are Enumeration, Iterator, ListIterator.

Q19.What are the differences between Enumeration, Iterator, ListIterator?

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Enumeration | Iterator | ListIterator |
| It is legacy. | Yes | no | No |
| It is applicable for | Only legacy collection classes i.e Vector, Stack. | Any collection object. Hence it is an universal cursor. | Only List implemented class objects. |
| movements | Single direction cursor(forward direction) | Single direction cursor(forward direction) | Bidirection |
| How to get | By calling elements() method. | By calling iterator() method. | By calling listIterator() method. |
| Accessability | Only read. | Read and remove. | Read, remove, add, replace. |
| methods | Two methods.  hasMoreElements(), nextElement(). | Three methods.  hasNext(), next(), remove(). | Nine methods. |

Q20: tell me about HashSet?

1. It is a class implemented from Set interface. The underlying data structure is Hashtable.
2. Insertion order is not preserve. insertion order is based on hashcode of the object.
3. Duplicate objects are not allowed.

Note: if we are trying to insert duplicate object to the HashSet, we won’t get any compile time or runtime error. The add() method simply returns null i.e the object is not added.

1. Null insertion is possible.
2. This class implements Serializable, Cloneable interfaces.
3. HashSet is recommended to use if our frequent operation is search operation.
4. Heterogenious objects are allowed.

Q21:What is load factor or fill ratio?

Generally in ArrayList,

ArrayList al=new ArrayList(); it creates an empty ArrayList object with default initial capacity 10.

if ArrayList reaches its max capacity then a new ArrayList object will be created with new capacity.

Here newcapacity=(al.capacity\*(3/2))+1 i.e 16.

Similarly in HashSet, the new capacity roll will be played by load factor or fill ratio.

Here default initial capacity is 16 and default fill ratio or load factor is 0.75

Q22:if we are trying to insert duplicate objects in Set, What will happen?

Basically Set doesn’t allow duplicate objects. Even though you are trying to insert duplicate object in to Set, then it won’t raise any compile time error or runtime error. The add() method simply returns null. i.e the object is not added.

Q23:What is the difference between HashSet and LinkedHashSet?

|  |  |
| --- | --- |
| HashSet | LinkedHashSet |
| The underlying data structure is hashtable. | The underlying data structure is hashtable and linked list. |
| Insertion order is not preserved. | Insertion order is preserved. |
| Introduced in java 1.2 | Introduced in java 1.4 |

Note: with LinkedHashSet, we can maintain uniqness and insertion order. That’s why LinkedHashSet, LinkedHashMap are the best suitable to implement cache applications.

Q24:What are the major enhancements of Collection Framework in java 1.4 version?

LinkedHashSet, LinkedHashMap & IdentityHashMap.

Q25:tell me about SortedSet interface?

1. It is the child interface of Set
2. If we want to represent a group of objects without duplicates, according to some sorting order then we should go for SortedSet interface.
3. The sorting order can be natural sorting order or customised sorting order.

Note: for customised sorting order, we need to use comparator() method.

Q26:tell me about TreeSet?

1. It is a class implemented from SortedSet interface. The underlying data structure is balanced tree.
2. Duplicate objects are not allowed.
3. All elements will be inserted according to some sorting order hence insertion order is not preserved.
4. TreeSet t1=new TreeSet() it creates an empty TreeSet object where all the objects will be inserted according to natural sorting order.
5. TreeSet t2=new TreeSet(Comparator c1) it creates an empty TreeSet object where all the objects will be inserted according to customized sorting order described by Comparator object.

Q27:tell me about Comparable interface?

1. If we are depending on natural sorting order, compulsary objects should be homogenious and comparable. Otherwise we will get class cast exception.
2. An object is said to be comparable if and only if the corresponding class implements Comparable interface. Only String class, wrapper classes are comparable.
3. Comparable interface is present in the java.lang package, contains only one method compareTo().
4. When we are depending on natural sorting order, jvm internally calls compareTo() method to places objects according to sorting order. Hence the objects should be comparable(i.e should be homogenious) otherwise we will get ClassCastException.

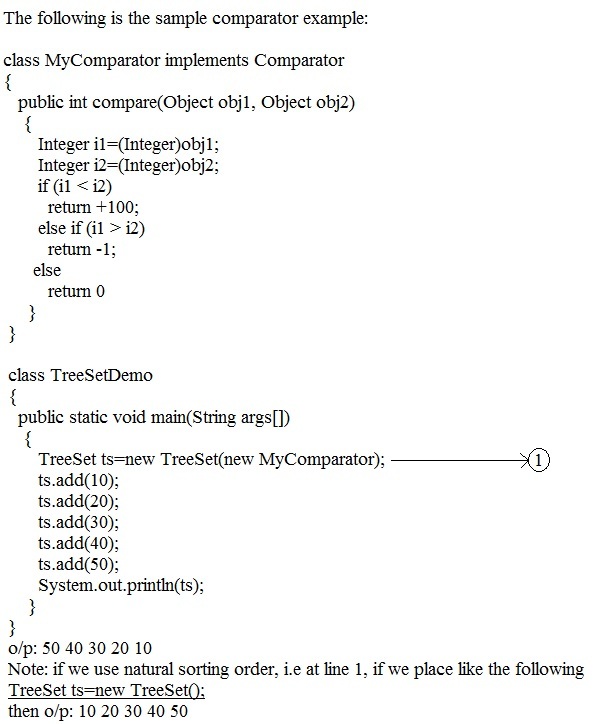
Q28:tell me about Comparator interface?

1. Sometimes we have to implement our own sorting order instead of natural sorting order. We can handle this kind of requirement by using Comparator interface.
2. This interface present in java.util package, contains two methods.
3. Public int compare(Object obj1, Object obj2). We have to define this method.

Here obj1, obj2 need not to be homogeneous(i.e comparable), can be heterogeneous.

1. Public boolean equals(Object obj). this method is already available in Object class.so defining this method is optional.

Note: Comparable meant for default natural sorting order where as Comparator meant for customized sorting order.



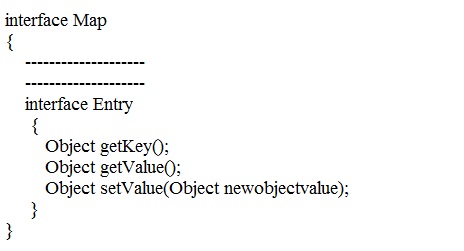
Q29:What is the difference between Comparable and Comparator?

|  |  |
| --- | --- |
| Comparable | Comparator |
| Present in java.lang package. | Present in java.util package. |
| Contains only one method compareTo(Object obj1). | Contains two methods.(i) Compare(Object obj1, Object obj2) (ii) equals(Object obj) |
| It is to define natural sorting order. | It is to define customized sorting order. |
| String class and wrapper classes implemented this interface. | None |
| It is a marker interface. | It is not a marker interface. |

Q30:comparision between Set implemented classes.

|  |  |  |  |
| --- | --- | --- | --- |
| Property | HashSet | LinkedHashSet | TreeSet |
| Data structure | Hash table | Hash table + linked list | Balanced tree |
| Insertion order | Not preserved | preserved | Not preserved |
| Sorting order | Not preserved | Not preserved | Preserved |
| Duplicate objects | No | No | No |
| Heterogeneous | allowed | allowed | Not allowed in case of natural sorting order. Allowed in case of customized sorting order. |
| Null acceptance | Allowed(once) | Allowed(once) | For empty TreeSet, as a first element, null is allowed.in other cases we will get NullPointerException. |

Q31:What is the Entry interface?

As we know, in Map, each key-value pair is considered as an entry. Hence Map is considered as set of entries. Without existing Map object, there is no chance of existing entry object. Hence Entry interface define inside the Map interface. See the following.

Q32: tell me about the HashMap?

1. The underlying data structure is hash table. This class is implemented from Map interface.
2. Insertion order is not preserve, because it is based on hashcode of keys.
3. Duplicate keys are not allowed. But values can be duplicated.
4. Heterogeneous objects allowed for both keys and values.
5. Null key is allowed. But only once.
6. Null values are allowed for any no of times.

Note:

Collection interface meant for individual objects where as Map interface meant for key-value pairs. Hence there is no relationship between Collection and Map.

Q33.tell me about Hashtable?

1. It is a class implemented from Map interface. The underlaying data structure is hash table.
2. Insertion order is not preserve, because it is based on hashcode of keys.
3. Duplicate keys are not allowed. But values can be duplicated.
4. Heterogeneous objects allowed for both keys and values.
5. Null insertion is not possible for both key & value. Violation leads to NullPointerException.
6. All methods are synchronized and hence Hashtable Object is thread safe.

Q34:What are the differences between HashMap and Hashtable?

|  |  |
| --- | --- |
| HashMap | Hashtable |
| Underlying data structure is hash table.  No method is synchronized | Here also, underlying data structure is hash table. Every method is synchronized |
| HashMap object is not thread safe | Hashtable object is thread safe. |
| Performance is high. | Performance is low. |
| Null is allowed for both keys and values | Null, we can not use for key and value. Violation leads to NullPointerException. |
| It is non legacy, introduced in java 1.2 | It is legacy, introduced in java 1.0 |

Q35: tell me about LinkedHashMap?

It is exactly same as HashMap except the following differences.

|  |  |
| --- | --- |
| HashMap | LinkedHashMap |
| The underlying data structure is hash table | The underlying data structure is hash table and linked list. |
| Insertion order is not preserve. | Insertion order is preserve. |
| Introduced in java 1.2 | Introduced in java 1.4 |

Q36: tell me about IdentityHashMap?

1. This is exactly same as HashMap, but the following difference. In case of HashMap, jvm always uses .equals() method to identify duplicate keys.
2. But in case of IdentityHashMap, jvm uses ==(equals) operator to identify duplicate keys. For clarification see the following example

HashMap X=new HashMap();

Integer i1=new Integer(10);

Integer i2=new Integer(10);

X.put(i1,”Gutta”);

X.put(i2,”Suresh”);

System.out.println(X);

o/p: {10=Gutta}

Note: here, if we take IdentityHashMap instead of HashMap, the we will get

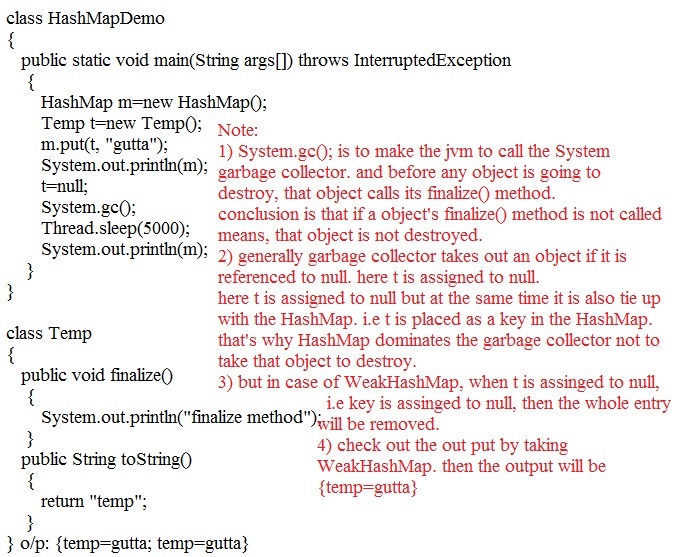
o/p: {10=Gutta, 10=Suresh}

Q37:tell me about WeakHashMap?

it is exactly same as the HashMap except the following difference.

In case of HashMap, an object is not eligible for gc() when the object is associated with HashMap, even though it doesn’t contain any external reference. It means HashMap dominates the garbage collector.

In case of WeakHashMap, in the above situation, garbage collector dominates WeakHashMap. To get clarity on this concept, see the following example.



Q38:tell me about the SortedMap?

1. It is the child interface of Map interface.
2. If we want to represent set of entries according to some sorting order of keys, then we should go for SortedMap.
3. i.e sorting order is based on keys, but not based on values.

Q39:tell me about the TreeMap?

1. It is a class implemented from SortedMap interface. The underlying data structure is RED-BLACK tree.
2. Sorting order always based on keys.
3. Duplicate keys are not allowed but values can be duplicated.
4. If we are depending on natural sorting order, compulsory keys should be homogeneous and comparable otherwise we will get ClassCastException.
5. If we are defining our own sorting order then the keys need not to be homogeneous and need not to be comparable. No rules on values.
6. Coming to null acceptance, as the first entry, the key as null is allowed. If we insert other than this case, we will get NullPointerException. In case of null acceptance, there are no rules for values.

Q40:tell me about Properties?

It is child class of Hashtable. In this case,both key and value should be String object only.

See the following sample program on this.

Import java.util.\*;

Import java.io.\*;

Class PropertiesDemo

{

Public static void main(String args[]) throws Exception

{

Properties p=new Properties();

FileInputStream fis=new FileInputStream(“Abc.properties”);

p.load(fis);

System.out.println(p);

String s=getProperty(“AAA”);

System.out.println(s);

p.setProperty(“BBB”,”99999”);

FileOutputStream fos=new FileOutputStream(“Abc.properties”);

p.store(fos,”updated by gutta”);

}

}

Properties file is

Abc.properties

BBB=99999

AAA=88888

CCC=77777

DDD=66666

EEE=55555

Q41:tell me about Queue?

1. It is an interface introduced in java 1.5 version
2. If we want to represent group of objects prior to processing, then we should go for Queue.
3. In general Queue follow FIFO, but we can customize the order based on our requirement.
4. From java 1.5 version, LinkedList is alos implented from this interface.
5. LinkedList implementation of Queue always follows FirstInFirstOut.
6. Methods in Queue interface
7. Boolean offer(Object o) it is for adding an object to the Queue.
8. Object peek() it returns head element of the Queue if the Queue is empty, this method returns null.
9. Object poll() it removes and returns head element of the Queue if the Queue is empty, this method returns null.
10. Object element() it returns head element of the Queue if Queue is empty, this raises NoSuchElementException.
11. Object remove() it remove and returns head element of the Queue if the Queue is empty, this method raises NoSuchElementException.

Q42:tell me about Priority Queue?

1. It represents a data structure to hold a group of individual objects prior to processing based on some priority.
2. It can be natural sorting order or customized sorting order which is defined by Comparator object.
3. If we are depending on natural sorting order then objects should be homogeneous and comparable, otherwise we will get ClassCastException.
4. If we are depending on customized sorting order then objects need not to be homogeneous and comparable.
5. Insertion order is not preserved.
6. Duplicate objects are not allowed.
7. Null insertion isnot possible even as the first element also.

Q43:tell me about Collections class?

1. It is an utility class present in java.util package.
2. It defines several utility methods for Collection implemented class object.
3. You just go through api once to know what methods they are.

Q44.tell me about Arrays class?

It is an utility class present in java.util package, defines several methods applicable for Arrays(i.e primitive array, object array). Just go through api once to know what methods they are.

T H E E N D