Collection:

# Collections in Java

A Collection is a group of individual objects represented as a single unit. Java provides Collection Framework which defines several classes and interfaces to represent a group of objects as a single unit.

The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are two main root interfaces of Java collection classes.

**Need for Collection Framework :**

Before Collection Framework (or before JDK 1.2) was introduced, the standard methods for grouping Java objects (or collections) were array or Vector or Hashtable. All three of these collections had no common interface.  
For example, if we want to access elements of array, vector or Hashtable. All these three have different methods and syntax for accessing members:

Collection Map

/ / \ \ |

/ / \ \ |

Set List Queue Dequeue SortedMap

/

/

SortedSet

**Core Interfaces in Collections**

Note that this diagram shows only core interfaces.

**Collection :** Root interface with basic methods like add(), remove(),

contains(), isEmpty(), addAll(), ... etc.

**Set :** Doesn't allow duplicates. Example implementations of Set

interface are HashSet (Hashing based) and TreeSet (balanced

BST based). Note that TreeSet implements **SortedSet**.

**List :** Can contain duplicates and elements are ordered. Example

implementations are LinkedList (linked list based) and

[ArrayList](http://www.geeksforgeeks.org/array-vs-arraylist-in-java/) (dynamic array based)

**Queue :** Typically order elements in FIFO order except exceptions

like PriorityQueue.

**Deque :** Elements can be inserted and removed at both ends. Allows

both LIFO and FIFO.

**Map :** Contains Key value pairs. Doesn't allow duplicates. Example

implementation are [HashMap and TreeMap](http://www.geeksforgeeks.org/hashmap-treemap-java/).

TreeMap implements **SortedMap**.

The difference between Set and Map interface is, in Set we have only

keys, but in Map, we have key value pairs.

Performance Analysis LIST VS LINKED LIST

Here if the list l is an ArrayList then we can access it in O(1) time since it is allocated contiguous memory blocks (just like an array) i.e random access is possible. But if the collection is LinkedList, then random access is not possible since it is not allocated contiguous memory blocks, so in order to access a element we will have to traverse the link list till you get to the required index, thus the time taken in worst case to access an element will be O(n).

Set in Java

* Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.
* Basically, Set is implemented by HashSet, LinkedSet or TreeSet (sorted representation).
* Set has various methods to add, remove clear, size, etc to enhance the usage of this interface

HashSet in Java

**HashSet:**

* Implements [Set Interface](http://quiz.geeksforgeeks.org/set-in-java/).
* Underlying data structure for HashSet is hashtable.
* As it implements the Set Interface, duplicate values are not allowed.
* Objects that you insert in HashSet are not guaranteed to be inserted in same order. Objects are inserted based on their hash code.
* NULL elements are allowed in HashSet.
* HashSet also implements Searlizable and Cloneable interfaces.

**What is initial capacity and load factor?**

The initial capacity means the number of buckets when hashtable (HashSet internally uses hashtable data structure) is created. Number of buckets will be automatically increased if the current size gets full.  
The load factor is a measure of how full the HashSet is allowed to get before its capacity is automatically increased. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is rehashed (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

Number of stored elements in the table

load factor = -----------------------------------------

                        Size of the hash table

**E.g.** If internal capacity is 16 and load factor is 0.75 then, number of buckets will automatically get increased when table has 12 elements in it.

**Effect on performance:**  
Load factor and initial capacity are two main factors that affect the performance of HashSet operations. Load factor of 0.75 provides very effective performance as respect to time and space complexity. If we increase the load factor value more than that then memory overhead will be reduced (because it will decrease internal rebuilding operation) but, it will affect the add and search operation in hashtable. To reduce the rehashing operation we should choose initial capacity wisely. If initial capacity is greater than the maximum number of entries divided by the load factor, no rehash operation will ever occur.

**Important Methods in HashSet:**

* **boolean add(E e)** : add the specified element if it is not present, if it is present then return false.
* **void clear()** : removes all the elements from set.
* **boolean contains(Object o)** : return true if element is present in set.
* **boolean remove(Object o)** : remove the element if it is present in set.
* **Iterator iterator()** : return an iterator over the element in the set.
* **How HashSet internally work?**  
  All the classes of Set interface internally backed up by Map. HashSet uses HashMap for storing its object internally. You must be wondering that to enter a value in HashMap we need a key-value pair, but in HashSet we are passing only one value.
* **Then how is it storing in HashMap?**  
  Actually the value we insert in HashSet acts as key to the map Object and for its value java uses a constant variable. So in key-value pair all the keys will have same value.

# Stack Class in Java

Java provides an inbuilt object type called **Stack**. It is a collection that is based on the last in first out (LIFO) principle. On Creation, a stack is empty.

It extends **Vector** class with five methods that allow a vector to be treated as a stack. The five methods are:  
  
1. **Object push(Object element)** : Pushes an element on the top of the stack.  
  
2. **Object pop()** : Removes and returns the top element of the stack. An ‘EmptyStackException’ exception is thrown if we call pop() when the invoking stack is empty.  
  
3. **Object peek( )** : Returns the element on the top of the stack, but does not remove it.  
  
4. **boolean empty()** : It returns true if nothing is on the top of the stack. Else, returns false.  
  
5. **int search(Object element)** : It determines whether an object exists in the stack. If the element is found, it returns the position of the element from the top of the stack. Else, it returns -1.

Stack<Integer> stack = new Stack<Integer>();

//push

for(int i = 0; i < 5; i++)

{

stack.push(i);

}

//pop

for(int i = 0; i < 5; i++)

{

Integer y = (Integer) stack.pop();

System.out.println(y);

}

**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.

**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.

**Q) Why is it preferred to declare: List<String> list = new ArrayList<String>(); instead of ArrayList<String> = new ArrayList<String>();**

Ans) It is preferred because:

1. If later on code needs to be changed from ArrayList to Vector then only at the declaration place we can do that.
2. The most important one – If a function is declared such that it takes list. E.g void showDetails(List list);  
   When the parameter is declared as List to the function it can be called by passing any subclass of List like ArrayList, Vector, LinkedList making the function more flexible.

**Q) Which data structure HashSet implements ?**

Ans) HashSet implements hashmap internally to store the data. The data passed to hashset is stored as key in hashmap with null as value.

**Q) What is a ConcurrentHashMap ?**

Ans) A concurrentHashMap is thread-safe implementation of Map interface. In this class put and remove method are synchronized but not get method. This class is different from Hashtable in terms of locking; it means that hashtable use object level lock but this class uses bucket level lock thus having better performance. The allowed concurrency among update operations is guided by the optional concurrencyLevel constructor argument (default 16), which is used as a hint for internal sizing. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. Because placement in hash tables is essentially random, the actual concurrency will vary. Ideally, you should choose a value to accommodate as many threads as will ever concurrently modify the table. Using a significantly higher value than you need can waste space and time, and a significantly lower value can lead to thread contention. But overestimates and underestimates within an order of magnitude do not usually have much noticeable impact. A value of one is appropriate when it is known that only one thread will modify and all others will only read. Also, resizing this or any other kind of hash table is a relatively slow operation, so, when possible, it is a good idea to provide estimates of expected table sizes in constructors.

**Q) What is the difference between iterator access and index access?**?

Ans) Index based access allow access of the element directly on the basis of index. The cursor of the datastructure can directly goto the 'n' location and get the element. It doesnot traverse through n-1 elements.

In Iterator based access, the cursor has to traverse through each element to get the desired element.So to reach the 'n'th element it need to traverse through n-1 elements.

Insertion,updation or deletion will be faster for iterator based access if the operations are performed on elements present in between the datastructure.

Insertion,updation or deletion will be faster for index based access if the operations are performed on elements present at last of the datastructure.

Traversal or search in index based datastructure is faster.

ArrayList is index access and LinkedList is iterator access.

**Q) How to sort a list in reverse order?**

Ans) To sort the elements in the reverse natural order of the strings, get a reverse Comparator from the Collections class with reverseOrder(). Then, pass the reverse Comparator to the sort() method.

List list = new ArrayList();

Comparator comp = Collections.reverseOrder();

Collections.sort(list, comp)

**Q) Can a null element be added to a Treeset or HashSet ?**

Ans) A null element can be added only if the set is of size 1 because when a second element is added then as per set defination a check is made to check duplicate value and comparison with null element will throw NullPointerException.  
HashSet is based on hashMap and can contain null element.

**Q) How to sort list of strings - case insensitive ?**

Ans) using Collections.sort(list, String.CASE\_INSENSITIVE\_ORDER);

**Q) How to make a List (ArrayList,Vector,LinkedList) read only ?**

Ans) A list implemenation can be made read only using**Collections.unmodifiableList(list)**. This method returns a new list. If a user tries to perform add operation on the new list; UnSupportedOperationException is thrown.

**Q) Which is faster to iterate LinkedHashSet or LinkedList?**

Ans) LinkedList.

**Q) Arrange in the order of speed - HashMap,HashTable, Collections.synchronizedMap,concurrentHashmap**

Ans) HashMap is fastest, ConcurrentHashMap,Collections.synchronizedMap,HashTable.

**Q23) What is identityHashMap?**

Ans) The IdentityHashMap uses == for equality checking instead of equals(). This can be used for both performance reasons, if you know that two different elements will never be equals and for preventing spoofing, where an object tries to imitate another.

**Q) What is WeakHashMap?**

Ans) A hashtable-based Map implementation with weak keys. An entry in a 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, that is, made finalizable, finalized, and then reclaimed. When a key has been discarded its entry is effectively removed from the map, so this class behaves somewhat differently than other Map implementations.