**Collection Framework – Part\_04**

* **ArrayList:**

Usually we can use collection to hold and transfer objects from one location to another location (container). To provide support for this requirement every collection class by default implements Serializable and Cloneable interfaces.

ArrayList and Vector classes implements RandomAccess so that we can access any random elements with the same speed.

**RandomAccess:**

RandomAccess interface present in java.util package and it doesn’t contain any methods. It is a marker interface. Where the required ability will be provided automatically by the JVM.

Example:

ArrayList l1 = new ArrayList();

LinkedList l2 = new LinkedList();

System.out.println(l1 instanceof Serializable); //true

System.out.println(l2 instanceof Cloneable); //true

System.out.println(l1 instanceof RandonAccess); //true

System.out.println(l2 instanceof RandomAccess); //false

ArrayList is the best choice if our frequent operation is retrieval operation. (Because, ArrayList implements RandomAccess interface).

ArrayList is the worst choice, if our frequent operation is insertion or deletion in the middle.

Example:

If a list contains 1 lakh element, and if we want to add an item to the 1st index, so it will place our new value in the first element and the existing value is shifted to the next element, this operation needs to be performed for all the remaining indexes. So, this will impact the performance.

The same case will happen if we go for delete operation. When we delete an item the element next to the deleted index will occupy the deleted index, the same operation needs to be performed for all the remaining indexes.

* **Difference between ArrayList and Vector:**

|  |  |  |
| --- | --- | --- |
| S.No | ArrayList | Vector |
| 1 | Every method present in the ArrayList is non-synchronized. | Every method present in a Vector is synchronized. |
| 2 | At a time, multiple threads are allowed to operate on ArrayList object and hence it is not thread safe. | At a time, only one thread is allowed to operate on Vector object and hence it is thread safe. |
| 3 | Relatively performance is high because Threads are not required to wait to operate on ArrayList object. | Relatively performance is low because Threads are required to wait to operate on Vector object. |
| 4 | Introduced in 1.2 V and it is non-legacy. | Introduced in 1.0 V and it is legacy. |

* **How to get synchronized version of ArrayList object:**

By default ArrayList is non-synchronized but we can get synchronized version of ArrayList object sychronizedList() method of Collections class.

public static List synchronizedList(List l);

Eaxmple:

ArrayList l = new ArrayList();

List l1 = Collections.synchronizedList(l);

l 🡪 Non-synchronized

l1 🡪 synchronized

Similarly, we can get synchronized version of Set and Map objects using the following methods of Collections class.

public static Set synchronizedSet(Set s);

public static Map synchronizedMap(Map m);

* **LinkedList:**

The underlying data structure is “Double/Doubly Linked List”.

Insertion order is preserved.

Duplicated objects are allowed.

Heterogeneous objects are allowed.

null insertion is possible.

LinkedList implements Serializable and Cloneable interfaces, but not RandomAccess.

LinkedList is the best choice if our frequent operation, insertion or deletion in the middle.

LinkedList is the worst choice if our frequent operation is retrieval operation.

**Constructors:**

LinkedList l = new LinkedList();

Creates an empty LinkedList object.

LinkedList l = new LinkedList(Collection c);

Creates an equivalent LinkedList object for the given Collection.

**LinkedList class specific methods:**

Usually we can use LinkedList to develop Stacks and Queues. To provide support for this requirement LinkedList class defines the following specific methods.

void addFirst(Object o);

void addLast(Object o);

Object getFirst();

Object getLast();

Object removeFirst();

Object removeLast();

Example:

import java.util.\*;

class LinkedListDemo{

public static void main(String[] args){

LinkedList l = new LinkedList();

l.add(“durga”);

l.add(30);

l.add(null);

l.add(“durga”); //[durga,30,null,durga]

l.set(0,”Software”); //[Software,30,null,durga]

l.add(0, “venky”);// [venky, Software,30,null, durga]

l.removeLast(); [venky,Software,30,null]

l.addFirst(“CCC”);//[CCC,venky,Software,30,null]

System.out.println(l); [CCC,venky,Software,30,null]

}

}