Difference between ArrayList and Vector In java

BY CHAITANYA SINGH | FILED UNDER: JAVA COLLECTIONS

ArrayList and Vector both use Array as a data structure internally. However there are few differences in the way they store and process the data. In this post we will discuss the difference and similarities between ArrayList and Vector.

ArrayList Vs Vector:

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

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

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

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

4) **fail-fast**: First let me explain what is fail-fast: If the collection (ArrayList, vector etc) gets structurally modified by any means, except the **add or remove methods** of iterator, after creation of iterator then the iterator will throw `ConcurrentModificationException`. Structural modification refers to the addition or deletion of elements from the collection.

As per the Vector javadoc the Enumeration returned by Vector is not fail-fast. On the other side the iterator and listIterator returned by ArrayList are fail-fast.

5) **Who belongs to collection framework really?** The vector was not the part of collection framework, it has been included in collections later. It can be

considered as Legacy code. There is nothing about Vector which List collection cannot do. Therefore Vector should be avoided. If there is a need of thread-safe operation make ArrayList synchronized as discussed in the next section of this post or use CopyOnWriteArrayList which is a thread-safe variant of ArrayList.

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

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

When to use ArrayList and when to use vector?

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

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

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

```
//Use Collecions.synzhonizedList method
List list = Collections.synchronizedList(new ArrayList());
...

//If you wanna use iterator on the synchronized list, use it
//like this. It should be in synchronized block.
synchronized (list) {
  Iterator iterator = list.iterator();
  while (iterator.hasNext())
       ...
       iterator.next();
       ...
}
```