1. add() :

**transient** Object[] elementData;

**public** **boolean** add(E e) {

ensureCapacityInternal(size + 1);

elementData[size++] = e;

**return** **true**;

}

1. set():

**public** E set(**int** index, E element) {

rangeCheck(index);

E oldValue = elementData(index);

elementData[index] = element;

**return** oldValue;

}

1. remove():

**public** E remove(**int** index) {

rangeCheck(index);

modCount++;

E oldValue = elementData(index);

**int** numMoved = size - index - 1;

**if** (numMoved > 0)

System.*arraycopy*(elementData, index+1, elementData, index,

numMoved);

elementData[--size] = **null**; // clear to let GC do its work

**return** oldValue;

}

NB : modCount acts as a flag to ensure whether any CUD operation happened or not concurrently. It is assumed that modCount is incremented by 1 when any CRUD operation happens

1. **for** (String s : stringList) :

**Implicitely Iterator works**

* while iteratimg if we add or remove elements, ConcurrentModificationException is thrown.

Reason :

**int** expectedModCount = modCount; -- During initialisation of iterator, modCount value iis assigned to expectdModCount.

**public** E next() {

checkForComodification(); -- This method causes the ConcurrentModificationException

----

----

}

**final** **void** checkForComodification() {

**if** (modCount != expectedModCount)

**throw** **new** ConcurrentModificationException();

} – As we all know, modCount will be increased for any CUD operation, so after initialization of Iterator , if any CUD operation happens, the exception will be thrown

1. **Internal working of Iterator :**

**private** **class** Itr **implements** Iterator<E> {

**int** cursor; // index of next element to return

**int** lastRet = -1; // index of last element returned; -1 if no such

**int** expectedModCount = modCount;

Itr() {}

**public** **boolean** hasNext() {

**return** cursor != size;

}

@SuppressWarnings("unchecked")

**public** E next() {

checkForComodification();

**int** i = cursor;

**if** (i >= size)

**throw** **new** NoSuchElementException();

Object[] elementData = ArrayList.**this**.elementData;

**if** (i >= elementData.length)

**throw** **new** ConcurrentModificationException();

cursor = i + 1;

**return** (E) elementData[lastRet = i];

}

**public** **void** remove() {

**if** (lastRet < 0)

**throw** **new** IllegalStateException();

checkForComodification();

**try** {

ArrayList.**this**.remove(lastRet);

cursor = lastRet;

lastRet = -1;

expectedModCount = modCount;

} **catch** (IndexOutOfBoundsException ex) {

**throw** **new** ConcurrentModificationException();

}

}

@Override

@SuppressWarnings("unchecked")

**public** **void** forEachRemaining(Consumer<? **super** E> consumer) {

Objects.*requireNonNull*(consumer);

**final** **int** size = ArrayList.**this**.size;

**int** i = cursor;

**if** (i >= size) {

**return**;

}

**final** Object[] elementData = ArrayList.**this**.elementData;

**if** (i >= elementData.length) {

**throw** **new** ConcurrentModificationException();

}

**while** (i != size && modCount == expectedModCount) {

consumer.accept((E) elementData[i++]);

}

// update once at end of iteration to reduce heap write traffic

cursor = i;

lastRet = i - 1;

checkForComodification();

}

**final** **void** checkForComodification() {

**if** (modCount != expectedModCount)

**throw** **new** ConcurrentModificationException();

}

}

1. Internal working of forEach() :

**public** **void** forEach(Consumer<? **super** E> action) {

Objects.*requireNonNull*(action);

**final** **int** expectedModCount = modCount;

@SuppressWarnings("unchecked")

**final** E[] elementData = (E[]) **this**.elementData;

**final** **int** size = **this**.size;

**for** (**int** i=0; modCount == expectedModCount && i < size; i++) {

action.accept(elementData[i]);

}

**if** (modCount != expectedModCount) {

**throw** **new** ConcurrentModificationException();

}

}