```java
/*
 * Sources:
 * openjdk-jdk8u/jdk/src/share/classes/java/util/ArrayList.java
 * openjdk-jdk8u/jdk/src/share/classes/java/util/AbstractList.java
 */

public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneable, java.io.Serializable
{
    private static final int DEFAULT_CAPACITY = 10;
    private static final Object[] EMPTY_ELEMENTDATA = {};
    private static final Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA = {};
    transient Object[] elementData;
    private int size;

    public ArrayList(int initialCapacity) {
        if (initialCapacity > 0) {
            this.elementData = new Object[initialCapacity];
        } else if (initialCapacity == 0) {
            this.elementData = EMPTY_ELEMENTDATA;
        } else {
            throw new IllegalArgumentException("Illegal Capacity: "+
                                               initialCapacity);
        }
    }

    public ArrayList() {
        this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA;
    }

    public void ensureCapacity(int minCapacity) {
        int minExpand = (elementData != DEFAULTCAPACITY_EMPTY_ELEMENTDATA)
            ? 0
            : DEFAULT_CAPACITY;

        if (minCapacity > minExpand) {
            ensureExplicitCapacity(minCapacity);
        }
    }

    private static int calculateCapacity(Object[] elementData, int minCapacity) {
        if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
            return Math.max(DEFAULT_CAPACITY, minCapacity);
        }
        return minCapacity;
    }

    private void ensureCapacityInternal(int minCapacity) {
        ensureExplicitCapacity(calculateCapacity(elementData, minCapacity));
    }

    private void ensureExplicitCapacity(int minCapacity) {
        modCount++;

        if (minCapacity - elementData.length > 0)
            grow(minCapacity);
    }

    private static final int MAX_ARRAY_SIZE = Integer.MAX_VALUE - 8;

    private void grow(int minCapacity) {
        int oldCapacity = elementData.length;
        int newCapacity = oldCapacity + (oldCapacity >> 1);
        if (newCapacity - minCapacity < 0)
            newCapacity = minCapacity;
        if (newCapacity - MAX_ARRAY_SIZE > 0)
            newCapacity = hugeCapacity(minCapacity);
        elementData = Arrays.copyOf(elementData, newCapacity);
    }
```

```
74      private static int hugeCapacity(int minCapacity) {
75          if (minCapacity < 0)
76              throw new OutOfMemoryError();
77          return (minCapacity > MAX_ARRAY_SIZE) ?
78              Integer.MAX_VALUE :
79              MAX_ARRAY_SIZE;
80      }
81
82      public boolean add(E e) {
83          ensureCapacityInternal(size + 1);
84          elementData[size++] = e;
85          return true;
86      }
87
88      public void add(int index, E element) {
89          rangeCheckForAdd(index);
90
91          ensureCapacityInternal(size + 1);
92          System.arraycopy(elementData, index, elementData, index + 1,
93                           size - index);
94          elementData[index] = element;
95          size++;
96      }
97
98      public boolean addAll(Collection<? extends E> c) {
99          Object[] a = c.toArray();
100         int numNew = a.length;
101         ensureCapacityInternal(size + numNew);
102         System.arraycopy(a, 0, elementData, size, numNew);
103         size += numNew;
104         return numNew != 0;
105     }
106
107     public boolean addAll(int index, Collection<? extends E> c) {
108         rangeCheckForAdd(index);
109
110         Object[] a = c.toArray();
111         int numNew = a.length;
112         ensureCapacityInternal(size + numNew);
113
114         int numMoved = size - index;
115         if (numMoved > 0)
116             System.arraycopy(elementData, index, elementData, index + numNew,
117                              numMoved);
118
119         System.arraycopy(a, 0, elementData, index, numNew);
120         size += numNew;
121         return numNew != 0;
122     }
123
124     private void rangeCheckForAdd(int index) {
125         if (index > size || index < 0)
126             throw new IndexOutOfBoundsException(outOfBoundsMsg(index));
127     }
128 }
129
130 public abstract class AbstractList<E> extends AbstractCollection<E> implements List<E> {
131     protected transient int modCount = 0;
132 }
```