**Collections Speed Demo**

A common design question is whether to use an ArrayList, LinkedList, or Map. Sometimes, Big-O considerations are important. Let's consider the Big-O differences between ArrayList and LinkedList.

Consider this driver program:

6 **public static void** main(String[] args)  
 7 {  
 8 ArrayList<Integer> al = new ArrayList<Integer>();  
 9 LinkedList<Integer> ll = new LinkedList<Integer>();  
10 makeValues(al, ll);  
11   
12 System.out.println("get each in ArrayList = " + timeGetEach(al));  
13 System.out.println("get each in LinkedList = " + timeGetEach(ll));  
14 System.out.println("\nadd at 0 in ArrayList = " + timeAddFirst(al));  
15 System.out.println("add at 0 in LinkedList = " + timeAddFirst(ll));  
16 System.out.println("\nadd at list.size() in ArrayList = "+timeAddLast(al));  
17 System.out.println("add at list.size() in LinkedList = "+timeAddLast(ll));  
18 System.out.println("addLast in LinkedList = " + timeAddLastLL(ll));  
19 }

If N = 10000, predict which one of each group is faster. **your prediction actual ms**

get each in ArrayList = 3000000 2596729  
get each in LinkedList = 2000000 59887481

add at 0 in ArrayList = 20000 29120   
add at 0 in LinkedList = 15000 19002  
  
add at list.size() in ArrayList = 5000 6661  
add at list.size() in LinkedList = 4000 5440  
addLast in LinkedList = 6000 6535

Then code the methods, run them, and record your actual output. In order to get time in milliseconds, use the static method System.currentTimeMillis(). It returns an int. Use it before and after the method call, and subtract the end value from the beginning value. To get the time in nanoseconds use System.nanoTime(). Now draw some conclusions.

1. A(n) \_\_arraylist\_\_ is faster than a(n) \_linkedlist\_ when used for only indexing such as get(i). But a(n) \_linkedlist\_ outperforms a(n) \_arraylist\_ when used for modifying the data structure, such as remove(obj), contains(obj), addFirst(obj), addLast(obj).

2. Never use any of the xxxxx(i) methods in a(n) \_linkedlist\_. A(n) \_linkedlist\_ points to nodes, not to indexes.

3. Never use any of the xxxxx(obj) methods in a(n) \_arraylist\_, with the exception of a few like add(obj). A(n) \_arraylist\_ points to indexes, not to nodes.

4. If quick access to the data is important, us a(n) \_arraylist\_.

5. If quick insertion and deletion is important, use a(n) \_linkedlist\_\_.

6. If the amount of data is relatively unchanging, use a(n) \_arrayllist\_.

7. If the amount of data changes rapidly and widely, use a(n) \_linkedlist\_.