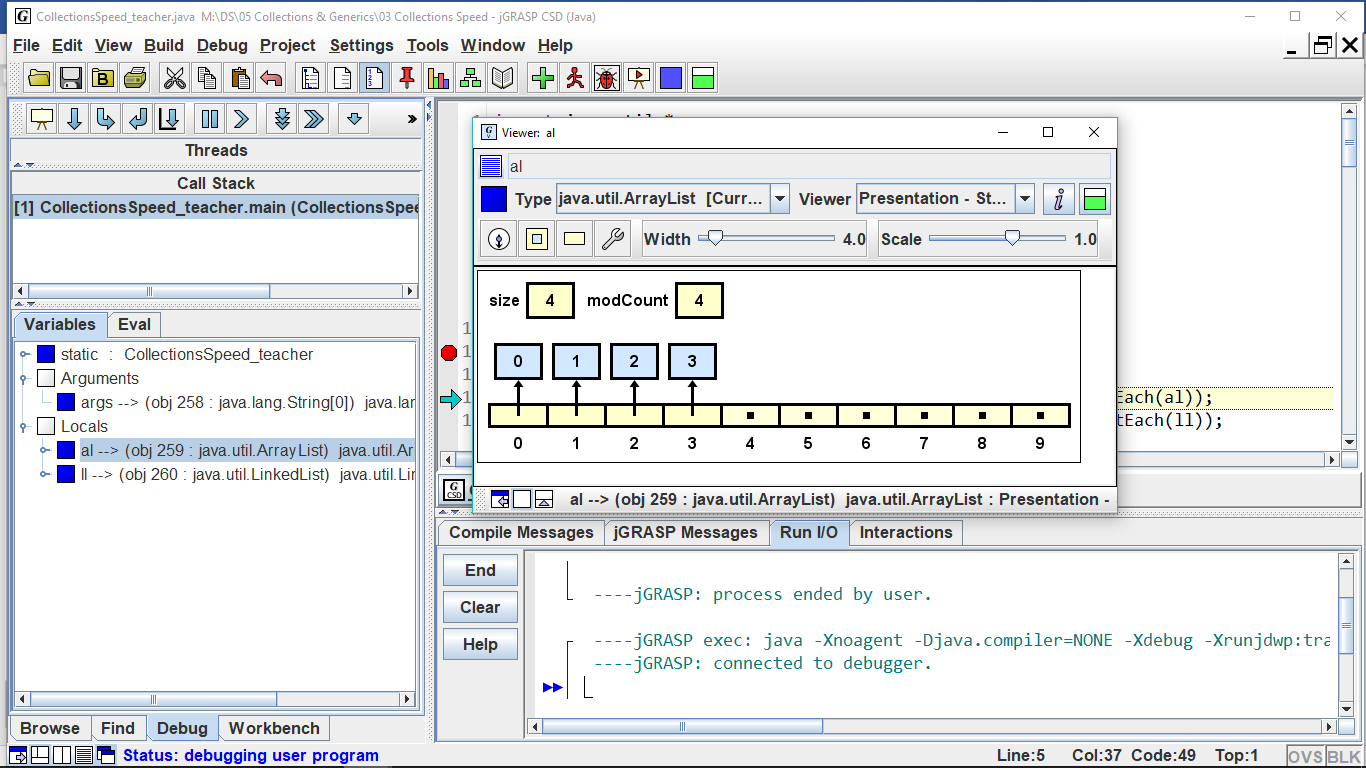
Big-O Values for LinkedList and ArrayList

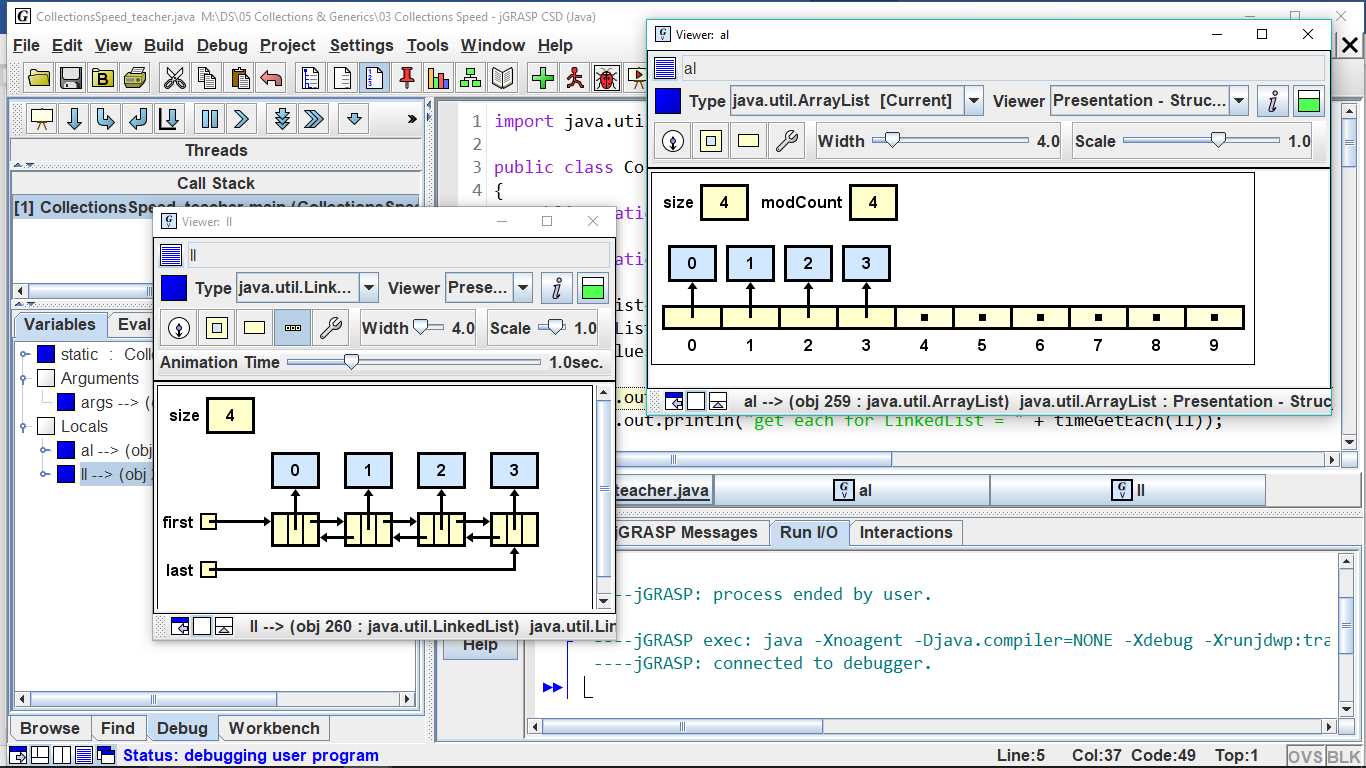
Preparation for CollectionsSpeed

As you have seen with TJArrayList, the java.util.ArrayList<E> maintains a backing "raw" array.



anArrayList

Similar to DLL, java.util.LinkedList<E> maintains a backing doubly linked list of list nodes with a pointer to the last node.



aLinkedList

|  |  |  |
| --- | --- | --- |
|  | **LinkedList** | **ArrayList** |
| size() | **O(1)** | **O(1)** |
| add(o) – adds at end of method | **O(1)** | **o(1), ammortized** |
| add(i, o) | **O(n)** | **O(n) shift** |
| get(i) | **O(n)** | **O(1)** |
| set(i, o ) | **O(n)** | **O(1)** |
| remove(i) | **O(1)** | **O(n)** |
| addFirst(o) | **O(1)** |  |
| addLast(o) | **O(1)** |  |
| getFirst() | **O(1)** |  |
| getLast() | **O(1)** |  |
| removeFirst | **O(1)** |  |
| removeLast() | **O(1)** |  |

Because of their differing structure, the two implementations of List<E> have different Big-O performances. As you fill in the Big-O table, think about when you need to access each element, and when you can jump directly to the element.

Some of these methods are *index-based* and others are *object-based.*

Predict the Big-O values and explain why:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | if list is an ArrayList | If list is a LinkedList |
| 1. | for(int i=0; i<list.size(); i++)  total += list.get(i); | **O(n) – outer for loop is o(n), and get method for ArrayList is o(1)** | **O(n2) – outer for loop is o(n), and get method for LinkedList is o(n)** |
| 2. | while( iter.hasNext() )  if(iter.next().equals("Bob"))  iter.remove(); | **O(n2) – while loop is o(n), and remove method is o(n) in ArrayList** | **O(n) – while loop is o(n), and remove method is o(1) in LinkedList** |

Let's think about the Big-O for *ordered* (or sorted) ArrayLists and LinkedLists. "Ordered" means that every time you insert or delete, you must maintain the sorted order of the list.

Predict the Big-O values and explain why:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | if list is an ArrayList | If list is a LinkedList |
| 3. | Searching the list | **O(log n) – ArrayList uses a binary search, with runs on o(log n). Get method for ArrayList is o(1).** | **O(n2) – Searching the list means going through each element, which is o(n). Get method for LinkedList is o(n).** |
| 4. | Inserting an item into the list | **O(n) – we are assuming here that we will not a resize. However, we would have to shift the elements, which is o(n).** | **O(n) – Shifting is o(n), but no resize is needed here. Wouldn’t matter anyways for LinkedLists.** |
| 5. | Deleting an item | **O(n) – deleting from an ArrayList forces a resize – remove method for ArrayList is o(n).** | **O(1) – deleting from a LinkedList does not force a resize – remove method for LinkedList is o(1). (probs not right, below is probs right)**  **o(n) – remove method for LinkedList is o(1), but you still need to traverse, which is o(n).** |
| 6. | Insert an item that forces a resize | **O(n) – Inserting in ArrayList is o(n), because we will need to shift the elements.** | **O(n) – Inserting in LinkedList is o(n), because the add function with an index as a parameter is o(n).** |

When do you choose to use an ArrayList and when to use a LinkedList? Sometimes Big-O efficiencies make the difference.

1. If quick access to the data is important, use a(n) **ArrayList**
2. If quick insertion and deletion is important, use a(n) **LinkedList**
3. If the amount of data is relatively unchanging, use a(n) **ArrayList**
4. If the amount of data changes rapidly and widely, use a(n) **LinkedList**
5. If data is to be inserted at the front, use a(n) **LinkedList** because the Big-O is **o(1)**
6. If data is to be inserted at the mid-point, use **both** because the Big-O is **o(n) for both!**
7. If data is to be inserted at the end, use **LinkedList** because the Big-O is **o(1)**
8. If you are accessing the data by its index, then a(n) **ArrayList**  gives you an access time of O(1).
9. If you are accessing the data by its index, then a(n) **LinkedList**  gives you an access time of O(n).