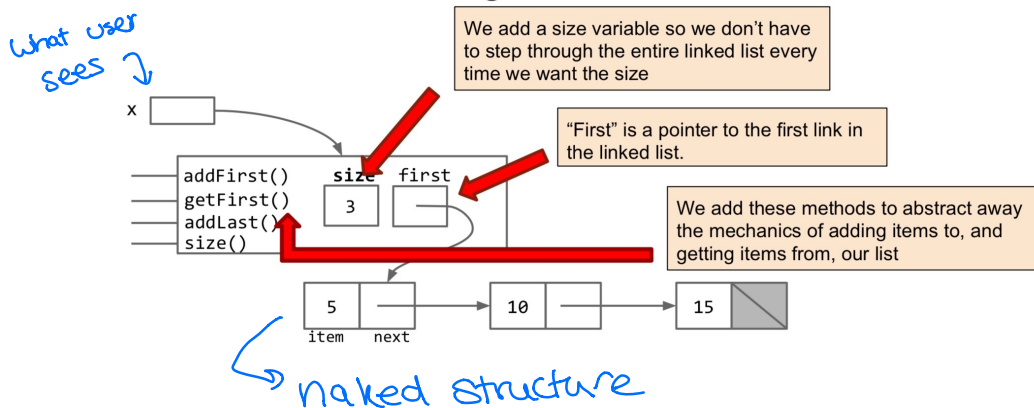


SLL List (Singly Linked List)

- basically regular linked list combined with "encapsulation"
- hides away inner workings of the linked list and allows us to store LL meta info.
- Analogy: SLL → train
LL → individual cars

Review: SLList Diagram

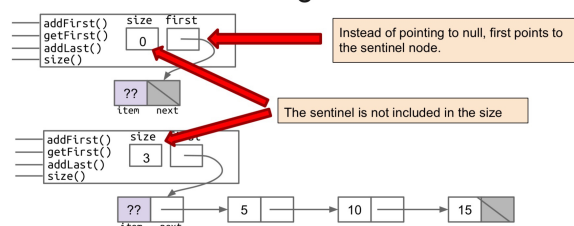


Something is missing here!

Sentinel Nodes!

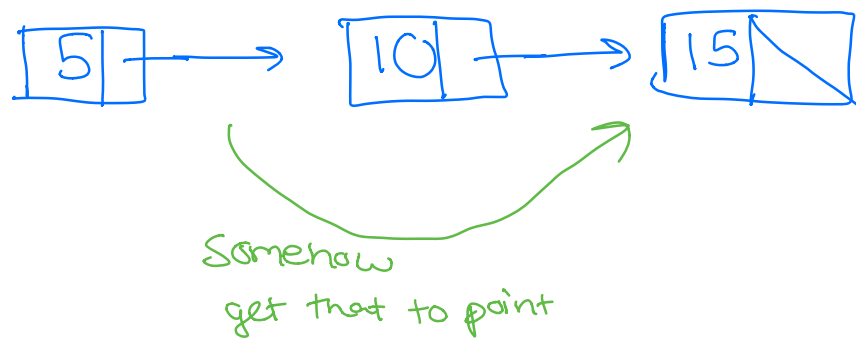
- Goal: make code as simple and generic as possible
- placeholder that helps avoid pesky null checks

Review: Sentinel Node Diagram



DLL (Doubly Linked List)

Imagine we want to remove(10) on this list:



A List

- arrays have benefit of constant time access.
- faster access than DLL, SLL
- however array size can't change!
- Solution: resize when too full

More Practice with Linked Lists

```
1 public class SLList {
2     private class IntNode {
3         public int item;
4         public IntNode next;
5         public IntNode(int item, IntNode next) {
6             this.item = item;
7             this.next = next;
8         }
9     }
10
11     private IntNode first;
12
13     public void addFirst(int x) {
14         first = new IntNode(x, first);
15     }
16 }
```

- 1.1 Implement `SLList.insert` which takes in an integer `x` and an integer `position`. It inserts `x` at the given `position`. If `position` is after the end of the list, insert the new node at the end.

For example, if the `SLList` is $5 \rightarrow 6 \rightarrow 2$, `insert(10, 1)` results in $5 \rightarrow 10 \rightarrow 6 \rightarrow 2$ and if the `SLList` is $5 \rightarrow 6 \rightarrow 2$, `insert(10, 7)` results in $5 \rightarrow 6 \rightarrow 2 \rightarrow 10$. Additionally, for this problem assume that `position` is a non-negative integer.

```
1 public void insert(int item, int position) {
```

- 1.2 Add another method to the `SLList` class that reverses the elements. Do this using the existing `IntNode` objects (you should not use **new**).

```
1 public void reverse() {
```

- 1.3 *Extra:* If you wrote `reverse` iteratively, write a second version that uses recursion (you may need a helper method). If you wrote it recursively, write it iteratively.

Arrays

- 2.1 Consider a method that inserts an **int** `item` into an **int[]** `arr` at the given position. The method should return the resulting array. For example, if `x = [5, 9, 14, 15]`, `item = 6`, and `position = 2`, then the method should return `[5, 9, 6, 14, 15]`. If `position` is past the end of the array, insert `item` at the end of the array.

Is it possible to write a version of this method that returns void and changes `arr` in place (i.e., destructively)? *Hint:* These arrays are filled meaning an array containing `n` elements will have length `n`.

Extra: Fill in the below according to the method signature:

```
1 public static int[] insert(int[] arr, int item, int position) {
```

- 2.2 Consider a method that destructively reverses the items in `arr`. For example calling `reverse` on an array `[1, 2, 3]` should change the array to be `[3, 2, 1]`. Write the `reverse` method:

```
1 public static void reverse(int[] arr) {
```

- 2.3 *Extra:* Write a non-destructive method `replicate(int[] arr)` that replaces the number at index `i` with `arr[i]` copies of itself. For example, `replicate([3, 2, 1])` would return `[3, 3, 3, 2, 2, 1]`. For this question assume that all elements of the array are positive.

```
1 public static int[] replicate(int[] arr) {
```