Lab: Linked Lists in JavaTown

We'll write all of our methods for working with linked lists in a class called `ListUtil`. The `ListUtil` class has no instance fields. Do not add any.

Exercise: Sizing Up the Situation

Add a method `size` to the class `ListUtil`, which should take a linked list as input and return the `size` of that list. For example, calling `size(primes)` using the `primes` list from the homework should return *3*. (Do *not* add any instance variables to `ListUtil` at any point in this lab!)

Exercise: It's What You Get

Add a method `get` to the class `ListUtil`, which should take in a linked list and an index, and should return the element at that index. Indices should start from 0. For example, calling `get(primes, 0)` should return *2*, and `get(primes, 2)` should return *5*.

Exercise: Making a List, Checking It Twice

Add a method `makeList` to the class `ListUtil`, which should take in a number and a value, and return a list containing that many copies of value. For example, `makeList(2, "It")` should return a list of 2 elements, both of which are the string "It".

Exercise: Giveth and Taketh Away

Add a method `add` to the class `ListUtil`, which should take in a list and a value, and add the value to the end of the list (without creating more than one `ListNode`). Be sure to include a comment describing any restrictions on the input list. (If done properly, there will be no restrictions.)

Now add a method `remove` to the class `ListUtil`, which should take in a list and a value, and remove that value from the list (without creating any more `ListNode`s). Your method should remove all occurrences of the given value. Again be sure to include a comment above your method describing any restrictions on the input. (If done properly, there will be no restrictions.)

Exercise: It Takes All Types

Match each of the following `ListUtil` methods with its type:

- `size` Number, Any → List
- `get` List →List
- `makeList` List, Any → List
- `add` List → Number
- `reverse` List, Number → Any

Your answer must appear in your notebook.

Exercise: Memorizer

Write a class `Memorizer` with a method called `seen`, which returns *`true`* when it's given a number it has already "seen", and *`false`* when it's given a new number. Here's an example usage of the `Memorizer` class. A helper method will make this task easier! Remember that a reference to an empty list is a null reference. You may not store a listnode of the form `ListNode(null,null)`!

```
a = new Memorizer();
b = new Memorizer();
a.seen(1)      //returns false
a.seen(2)      //returns false
a.seen(1)      //returns true
b.seen(2)      //returns false
b.seen(2)      //returns true
```

Exercise: reverse

Add a method `reverse` to the class `ListUtil`, which should take in a list, reverse it, and return a pointer to the front of the reversed list (without creating any more `ListNode`s).