**Main: Line 20:** Initialization of a new Monster List that represents a linked list, the empty constructor is used which initializes the first node to an empty node

**Main: Line 26:** We proceed to add some items to the List, we do that by adding a node, which contains the data, and points to the data node, then said node is placed where the previous data node was, essentially inserting the new node and element in the front of the list

**Main: Line 28:** Here we call a 'recursive' method, that essentially counts the depth of the list, as each non empty node is adding one to the result of the call of the referred node as per the definition of the base class, this stops once an empty node is encountered as it simply returns zero, and backtracking begins to evaluate the actual value

**Main: Line 29:** Count2 uses almost the same approach as Count1, being 'recursive' it adds one to the argument and passes that to the next node each time, the final empty node just returns that argument, and this argument gets returned by each subsequent method.

**Main: Line 33:** Yet another 'recursive' method, that akin to Count2, through the argument, keeps track of the current position, and once the index matches, returns the element, or null if we encounter the empty node

**Main: Line 39:** This gets a little more interesting, this 'recursive' method, based on the predicate, builds another linked list, the predicate dictates either the method makes a new node and returns it, alongside the rest of the call, or just passes the call to essentially skip this element

**Main: Line 43:** This line implicitly invokes the toString method of the list, which is similar to the Count1 method, where we add the string of the current element, then recursively add the rest, until an empty node is encountered that stops this recursion, and all the rest of the strings are finally getting collapsed into one

**How is printing the entire list accomplished?**

Akin to Count1 as explained, first we navigate until the deepest node, and from there, we start returning the strings and concatenating them together.

**What is the difference in count1() and count2() (Be specific)?**

One prepends the result, but can't be evaluated yet until the empty node has been met which breaks the recursion and allows for evaluation by backtracking, while the second approach already does the incrementation, by keeping track of the data through the argument in the method

**What is the purpose of the MonsterTest interface?**

It’s meant to be a predicate, that based on some condition, tells if it should be true or not, depending on how its used, it might essentially (and commonly does) act as a filter for list items and various other things.

**What would you need to do if you wanted to create a list of only Goblins?**

Switching to generics would be a better approach than making the same list over but for goblins, so that you're instantly making the list work with any type