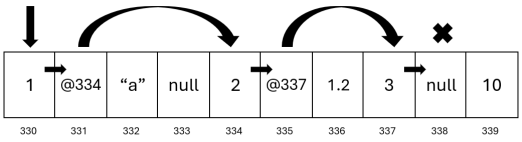
Assignment 4 - Linked Lists

In this assignment we will work with and define a custom data type called a *linked list*, an alternative to arrays. As we saw, arrays cannot expand because the form continuous blocks in the memory of the program, and thus expanding may overwrite memory being used for something else. Linked lists get around this issue by having each piece of the list consist of two parts: the actual *content* stored in the list, and the *memory address* of the next part of the list. In memory, a linked list containing the ints $1, 2, 3$ in that order may look like:



The head of the list is located at memory address 330, and its content is 1. The next memory address 331 stores the memory address of the next part of the list 334. At address 334 we find 2, the second int in the list, and 335 points us to the last part of the list. Notice the memory address for the last element of the list is `null`, indicating there is no next element in the list.

In this assignment we will implement the class IntegerLL, a LinkedList of integers. Java has a built-in linked list class `java.util.LinkedList` but recreating it will be instructive.

Problem 1. Our IntegerLL class has two fields:

```
private int data;
private IntegerLL next;
```

In Java, we do not work with memory addresses directly, so rather than an address, `next` is itself an `IntegerLL`. This is known as a *recursive* data structure. The final object constituting a specific linked list will have it's next field as *null*.
Define a Constructor that given an integer n, creates a single element linked list containing n.

Problem 2. Complete the following iterative code for the method length() that returns the length of this linked list.

```
public int length() {
    int counter = 1;
    IntegerLL current = this;

    while (current.next != null) {


    }

    return counter;
}
```

Write a recursive method to compute length(). Which do you feel is more intuitive for this data structure?

Problem 3. Write a method add(int n) that appends n to this list and returns nothing. Again, consider how you might do it iteratively and recursively.

Problem 4. Challenge: Write a method removeIndex(int n) removes the n-th element of the list (zero-indexed) for $n \geq 1$ (i.e. you cannot remove the first element). When you remove an item from the middle of the list, you will need re-link the remaining parts of the list together correctly.

Hint: Your code will probably include something like `current.next = current.next.next`.
You should throw an IndexOutOfBoundsException if n is out of bounds (less than 1 or greater than the length).