Announcements

- Program 5 has been assigned
    - Due one week from today: 11/25 at 11:59 PM
- Will not assign a Program 7

Program 5

For Program 5 you will be creating your own data structure called: MyVector.

Notice MyList.java and MyAbstractList.java are given to you. Though MyList.java is not completely implemented.

You must read the documentation in MyList.java and correctly implement the method signatures.

Let’s take a look at Zybooks.

Recursion

A method that calls itself is known as a **recursive method**.

For example:

```
public void countDown(int x) {
        If (x == 0) {
                System.out.println(“ZERO!”);
        } else {
                System.out.println(x);
                countDown(x - 1);                --- Notice it calls itself
        }
}
```

Recursion (cont.)

To further understand recursion we must understand what is going on in memory.

More specifically we need to understand Stack Memory and the role it plays in your programs.

Zybooks has a good example of this, let’s take a look....

Recursion (cont.)

Knowing what we know now about the Stack, what is the infamous error: **Stack Overflow**?

Stack Overflow refers to filling the Stack Memory region beyond its allocated capacity.

Zybooks has a good example of this…

Recursion (cont.)

Let us revisit this problematic code:

```
public void countDown(int x) {
        if (x == 0) {
                System.out.println(“ZERO!”);
        } else {
                System.out.println(x);
                countDown(x - 1);
                System.out.println(x);
        }
}
```

What is printed if countDown(5) were called? Why is it printed that way?

Recursion (cont.)

There are two steps required in creating your own recursive method:

- Write the **base case**: Every recursive method must have a case that returns a value without performing a recursive call. That case is called the **base case**. A programmer may write that part of the method first, and then test. There may be multiple base cases.
- Write the **recursive case**: The programmer then adds the **recursive case** to the method. This is where the recursive method calls itself. There may be multiple recursive cases.

Recursion (cont.)

```
public void countDown(int x) {
    if (x == 0) {
        System.out.println(“ZERO!”);    ← This is the base case
    } else {
        System.out.println(x);
        countDown(x - 1);               ← This is the recursive case
        System.out.println(x);
    }
}
```

Recursion (cont.)

A common error is to not cover all possible base cases in a recursive method.

Another common error is to write a recursive method that doesn't always reach a base case.

Both errors may lead to infinite recursion, causing the program to fail. (i.e. Causes stack overflow)

Recursion (cont.)

Common examples of recursion:

Fibonacci Sequence

Greatest Common Divisor (GCD)

Let’s take a look at each...

Searching Algorithms

An **algorithm** is a sequence of steps for accomplishing a task.

Linear search is a search algorithm that starts from the beginning of a list, and checks each element until the search key is found or the end of the list is reached.

A common example of the Linear Search algorithm is searching through the elements in an array.

Let’s look at an example...

Searching Algorithms (cont.)

An algorithm's **runtime** is the time the algorithm takes to execute.

Not to be confused with terms like: Runtime Polymorphism, those are two **different** uses of the word: *runtime*

Searching Algorithms (cont.)

Algorithm **runtime** example:

If each comparison takes **1 µs** (1 microsecond), a linear search algorithms runtime is **1 s** to search a list with 1,000,000 elements, **10 s** for 10,000,000 elements, and so on.

Ex: Searching Amazon's online store, which has more than 200 million items, could require more than 3 minutes.

Searching Algorithms (cont.)

An algorithm typically uses a number of steps proportional to the size of the input.

For a list with 32 elements, linear search requires at most 32 comparisons:

1 comparison if the search key is found at index 0, 2 if found at index 1, and so on, up to 32 comparisons if the the search key is **not found**.

Searching Algorithms (cont.)

For a list with **N elements**, linear search thus requires at most N comparisons.

The algorithm is said to require "on the order" of N comparisons.