Clicker Question

Given the following code:

```
public class RecursiveMath {

    public int recursiveAddition(int n) {
        if (n<=1) {
            return 1;
        } else {
            return recursiveAddition(n-1);
        }
    }

}
```

What is the output of `this.recursiveAddition(4)`?

A.1
B.9
C.10
D.StackOverflow!

Answer: A
When `this.recursiveAddition(4)` is called we check if n is less than or equal to 1, and because it is not we return the result of `this.recursiveAddition(3)`. This results in a call to `this.recursiveAddition(2)`, which results in a call to `this.recursiveAddition(1)` , which returns 1.

Clicker Question

Given the following code:

```
public class RecursiveMath {

    public int funkyFactorial(int n) {
        if (n == 0) {
            return 1;
        } else {
            return n * this.funkyFactorial(n-2);
        }
    }

}
```

What is the output of `this.funkyFactorial(5)`?

A.1
B.5
C.15
D.StackOverflow!

Answer: D
We wind up with calls to `funkyFactorial(3), funkyFactorial(1), funkyFactorial(-1)`, etc. and get a StackOverflow exception because our recursion never bottoms out.

Clicker Question

Given the following code:

```
public class RecursiveMath{
    private int _count;
    //constructor elided
    public int collatzCounter(int n) {
        _count += 1;
        if (n == 1) {
            return _count;
        } else {
            if (n % 2 == 0) { //if n is even
                return collatzCounter(n / 2);
            } else {
                return collatzCounter(3 * n + 1);
            }
        }
    }
    public int myCounter(int n) {
        _count = 0;
        return collatzCounter(n);
    }
}
```

What is the output of `myCounter(5)`?

A.4
B.5
C.6
D.StackOverflow

Answer: C

When we call `myCounter(5)`, `_count` is set to zero. We can follow the value of `n` and the value of `_count` at the start of the `collatzCounter()` method:

n	_count
5	0
16	1
8	2
4	3
2	4
1	5

and `_count` is incremented once more before we return it, so it winds up with a value of 6.

Clicker Question

Given the following code:

```
private void draw(int trunkLen) {

   if (trunkLen <= 0) {
      this.addLeaf(); // creates a leaf
                      // at the current location
   } else {
      _turtle.forward(trunkLen);
      _turtle.left(_branchAngle);
      this.draw(trunkLen - _trunkDecrememnt);
      _turtle.right(2*_branchAngle);
      this.draw(trunkLen - _trunkDecrement);
      _turtle.left(_branchAngle);
      _turtle.back(trunkLen);
   }
}
```

What would happen if you got rid of the first call `this.draw(trunkLen-_trunkDecrement)`?

A. We will only draw the right half of the tree
B. We will draw a spiral that terminates in a leaf
C. Stack Overflow!
D. None of the above

We accidently misplaced this slide in the lecture (we had meant to ask it once you had already seen the draw method for the full tree) so it's not a bad idea to take a second to read it over again.

.
.
.
.
.
.

Answer: B
Remember that our turtle leaves a trail as it moves forward. When we call draw, it moves forward, turns left, then immediately turns all the way to the right. We then enter the next call to draw, and the process continues with shorter and shorter `trunkLen`s until we draw a leaf. After we draw the leaf, the turtle will move backwards out the spiral to its original position. This is because when a method call contains another method call inside of it, we don't forget about the rest of the method, we just wait until the inner call is executed. This is true even in the case of recursion.