159 XTENDED

A WEEKLY REVIEW

RECURSION

Pull up **Socrative.com**

and join room **XTEND159**

IF YOU CAN SOLVE A PROBLEM WITH A LOOP, DO IT.

Computer Scientists Everywhere

Consider this code...

Pull up Socrative.com
and join room **XTEND159**

```
public static void countdown(int n)
{
    if (n == 0)
    {
        System.out.print(“Blastoff!”);
    } else {
        System.out.print(n + “ ”);
        countdown(n - 1);
    }
}
```

What is the output of countdown(4)?

WHAT IS RECURSION?

- A Recursive Method is a method that calls itself.
  That's it.

- It consists of two things:
  A Recursive Call (when the method calls itself)
  and a Base Case (a stopping point)

FROM OUR PREVIOUS EXAMPLE…

```
public static void countdown(int n)
{
    if (n == 0)
    {
        System.out.print("Blastoff!");

    } else {
        System.out.print(n + " ");
        countdown(n - 1);
    }
}
```

This is our base case.

This is our recursive call

DEFINING RECURSIVE CALLS

- A recursive call is a call to the current method.

- It should make progress towards a base case.

DEFINING BASE CASE

- A base case is the stopping point for a recursive method.
- For certain inputs, the method will not perform a recursive call.
- Any chain of recursive calls must reach a base case, otherwise we will get a stack overflow exception.

Consider this code...

Pull up Socrative.com
and join room **XTEND159**

```
1 public static void fibonacci(int n)
2 {
3     if (n == 1 || n == 2)
4     {
5         return 1;
6     }
7     return fibonacci(n - 1) + fibonacci(n - 2);
8 }
```

State the base case(s) and recursive call(s).

Introducing...

The Recursion Fairy

She'll Solve ALL Your Problems!*

*Computer Science Related

EXPLAINING THE RECURSION FAIRY

- Sometimes, we can break a problem down in to smaller versions of the same problem.

- Factorial: The product of an integer and all integers below it.

  $3! = 3 \bullet 2! = 3 \bullet 2 \bullet 1!$

  …the recursion fairy can help solve this…

EXPLAINING THE RECURSION FAIRY

```
public static int factorial(int n)
{
    if (n == 1)
    {
        return 1;
    }

    return n * factorial(n-1);
}
```

We handle the first part of the problem...

...and then hand the rest off to the recursion fairy.

ACTIVITY - COUNTX

Given a string, compute recursively (no loops) the number of lowercase 'x' chars in the string.

```
public int countX(String str)
{
    // TODO
}
```

HINT: You may find the methods chatAt(int index), substring(int startIndex), and length() from the String class helpful in solving this problem.

ACTIVITY - COUNTX

```
public int countX(String str)
{
    if (str.length == 0)
    {
        return 0;
    }

    if (str.charAt(0) == 'x')
    {
        return 1 + countX(str.substring(1));
    }

    return countX(str.substring(1));
}
```

ACTIVITY - ARRAY11

Given an array of ints, compute recursively the number of times that the value 11 appears in the array. We'll use the convention of considering only the part of the array that begins at the given index. In this way, a recursive call can pass index+1 to move down the array. The initial call will pass in index as 0.

```
public int array11(int[] nums, int index)
{
    // TODO
}
```

Example: array11([1, 2, 11], 0) → 1

ACTIVITY - ARRAY11

```
public int array11(int[] nums, int index)
{
    if (index == nums.length)
    {
        return 0;
    }

    if (nums[index] == 11)
    {
        return 1 + array11(nums, index + 1);
    }

    return array11(nums, index + 1);
}
```

ACTIVITY - DIVIDEBY2

Given a integer, return the number of times you need to divide by two before you reach the number 1.

```
public int divideByTwo(int n)
{
    // TODO
}
```

Example: divideByTwo(8) → 3

ACTIVITY - DIVIDEBY2

```
public int divideByTwo(int n)
{
    if (n <= 1)
    {
        return 0;
    }

    return 1 + divideByTwo(n / 2);
}
```