CS 158 Lab 15 March 30, 2010
Recursion

1. Create a new Java project called Recursion. This class will not have any objects, so make sure any methods you create are static. In this project create a new class called **GreatestCommonDivisor**. Add a method called **gcd** that returns an integer and takes two integer arguments. The greatest common divisor of two numbers is the largest integer that divides both numbers. There is a simple, iterative way to do this, but there is also a nice recursive solution. First, create a JUnit test class and add some test cases. Then code and test the following recursive algorithm (which is called the Euclidean algorithm):
    - If the second argument is a divisor of the first argument, then the greatest common divisor is the second argument
    - Otherwise the greatest common divisor can be found by getting the greatest common divisor of the second argument and the remainder that you get by dividing the first argument by the second argument

2. Create a new class called **Binary** that has a main method. Add a method called **printBinary** that takes an integer argument. Here's an easy recursive algorithm for printing the binary equivalent of a nonnegative integer. If the number is less than 2, you know what the answer is (the number itself). This is the base case. Otherwise, the last digit of the answer will be the remainder of the argument divided by 2. You can print that after printing the binary version of half the argument. For example, to print the binary version of the decimal number 17, since 17 is not less than two, you first print (recursively) the binary version of half of 17 (namely the binary version of 8, which is 1000) followed by the remainder of 17 divided by 2, which is 1. The result is 10001. Code this method and test it in the main method by printing the binary versions of a bunch of numbers.

3. Create a class with a **main** method called **Permutations**. Create a method called **getPermutations** that returns an **ArrayList** of **String**s and takes a **String** argument. Here is the algorithm:
    - Create an **ArrayList** that will contain all of the permutations
    - If the length of the input string is 1, just add that string to the **ArrayList**
    - Otherwise
        - separate the first character from the rest of the input string
        - get all permutations of the rest of the input string
        - for each permutation, add the first character in **all** possible positions and put each result into the **ArrayList** to be returned
    - Return the **ArrayList**

    It will be helpful for you to create another method first called **insertCharacterAt** that returns a **String** and takes as arguments the character to be inserted, the **String** into which it is to be inserted, and the position at which it is to be inserted. Create a JUnit test class for **Permutations** and include a test method for **insertCharacterAt**. Write good tests for this method, and then fill in the method and make sure it works. Be sure to include cases involving the end of the word, and when the word itself has no characters in it. Then work on implementing **getPermutations**. Test it by calling it in the main method with a 3 letter word and printing all the elements of the **ArrayList** returned.