Create Account on CodingBat.com and share progress with j.brucker@ku.th

1. Go to codingbat.com and create an account. Input your **real name** (in English) on the account.

2. Go to the "prefs" section ("prefs" link in upper right corner of page). In the "Teacher Share" section, enter **`j.brucker@ku.th.`**

`NOTE:` Input your real and share results with j.brucker@ku.th so I can see what problems you solved and give you a score. If you omit either of these (your name or not sharing) then no credit.

1. On **codingbat.com** solve (at least) these problems in the **Recusion-1** group:
factorial
triangle
sumDigits
changeXY
stringClean

Solve these problems in the **Recursion-2** group:
groupSum - we will use something like this in a future lab
groupSum6
groupNoAdj
splitArray

3-4. Write a class named **`Recursion`** in the default package containing the solution to problems 3 and 4. Commit your code to Github as a project named **`homework2`**.

3. Write a **`unique(List)`** method as described below. Use recursion and no loops. To get rid of warning messages, you can replace `List` with `List<?>`.

```
/**
 * Remove duplicate consecutive items from a list, changing the list parameter.
 * For example, if list = { a, b, b, b, c, b, c, c} then after calling unique the list
 * will contain {a, b, c, b, c}.  Only consecutive duplicates are removed.
 * Objects are compared using their own equals method.
 * @param list of any kind of object.  The elements are not null.
 * @return reference to the list parameter with consecutive duplicates removed.
 */
public static List unique(List list)
```

Requirements: Don't create any new lists. Modify the original list. Use the subList() method of the List interface. subList() returns a view of part of the existing list (not a copy), so any changes you make to subList affect the original list. Using subList, unique() can all itself (recursion) to process part of the list.

For example:

```
// create a List containing {"a", "b", "c", "d", "e"}
List<String> list = new ArrayList<>();
list.add("a");
list.add("b");
list.add("c");
...
// create a view (sublist) containing element 1, 2, 3 (but not 4)
```

```
List sub = list.subList(1, 4);
// sub contains {"b", "c", "d"}
sub.get(0);           // returns "b"
sub.get(1);            // returns "c"
sub.set(2, "hahaha");  // change element 2 to "hahaha"
sub.remove(1);         // remove element #1 ("c")
// print the original List.  You'll see that subList changed it
for(Object o: list) System.out.print(o+", ");
// output:  a, b, hahaha, e
```

4. (*Recursion Gone Wrong*) A classic example of recursion is the Fibonacci numbers, defined by the sequence:

F(0) = 1
F(1) = 1
F(n) = F(n-1) + F(n-2) for n > 1.

A simple recursive function for this is:

```
public static long fibonacci(int n) {
    // the base case
    if (n < 2) return 1;
    // the recursive step
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Try this on your computer. As the value of n gets larger the computation becomes very slow. Even `fibonacci(48)` takes a *long* time on a typical notebook computer.

4.1 Explain why this function is so slow, even for fairly small values of n (like n=45 or n=50). Write your explanation in `README.md` in your repository.

4.2 In the Recursion class, write a fibonacci(n) method that is fast. It must use recursion, no loops, and no external variables (that is, no attributes of a class). You probably need to write a *helper method* for this, and call the helper method to perform recursion. A hint is that the simple fibonacci method returns only *one* Fibonacci number, but the formula for Fibonacci's needs *two* numbers.

As with all assignments, do this yourself. Try to solve it yourself without asking anyone for help, including Google. In class next week, I will ask someone to explain why the simple method is slow and how to make it fast. Extra credit to the fastest method, using the least amount of memory.

For Your Knowledge

In Java, Fibonacci numbers will overflow a *long* value at Fibonacci(92). If you write it Python, you can compute arbitrarily large values.