# CIS 181 - Lab 3

## Ackermann's Function

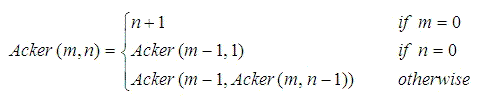
## Objectives

* To gain experience with using recursion by implementing the Ackermann's Function.
* To gain experience with using static methods to implement the AckerFunction class.
* To gain experience with using static variables to record method invocation history.

## Description

The Ackermann's function is of interest because it grows rapidly with respect to the size of m and n. It is the simplest example of a well defined total function which is computable but not [primitive recursive](http://mathworld.wolfram.com/PrimitiveRecursiveFunction.html). This means it cannot be implemented using only **for-loops**. Notice that for-loops (which have a fixed iteration limit) are a special case of while-loops.

The Ackermann's function is defined recursively for nonnegative integers as follows:



Implement the function as a method in Java and trace the history of method invocations as follows (e.g., m=1, n=2):

Input two intergers separated by a space character (enter "q" to quit):

1 2

Enter method acker: m = 1, n = 2

Enter method acker: m = 1, n = 1

Enter method acker: m = 1, n = 0

Enter method acker: m = 0, n = 1

Leave method acker: acker(0, 1) = 2

Leave method acker: acker(1, 0) = 2

Enter method acker: m = 0, n = 2

Leave method acker: acker(0, 2) = 3

Leave method acker: acker(1, 1) = 3

Enter method acker: m = 0, n = 3

Leave method acker: acker(0, 3) = 4

Leave method acker: acker(1, 2) = 4

Total number of invocations = 6, result = 4

Notice that to visually easy to follow the flow of execution, instead of just adding "Enter" and "Leave" messages, your program should indent the trace messages according to how "deep" the current recursive call is.

**Static Methods and Static Variables**

Within a class definition, elements that are declared static are called class elements - class methods or class variables - and are associated with the class itself, not with its instances. Such elements exist even if no objects of that class exist. They are referenced by qualifying their name with the name of class. For example, to use the square root method in the java.lang.Math class, you would refer to it as Math.sqrt(). Note that a static method cannot refer to a (nonstatic) instance variable of the class, nor can it invoke a nonstatic method of the class (unless it creates a new object of the class and uses this object as the calling object).

## A Sample Solution

Download the following jar file, and run it using the command "java -jar AckerFunction.jar" in a command console (you can open a command console by clicking on Start -> Run ..., and type in cmd). If you prefer GUI and do not like command console, you can also run a jar file in Eclipse.

* [AckerFunction.jar](file:///C:\Users\Alan\Work\Teaching\CIS181\reference\haiping\CIS181-F08-XU\labs\lab4\AckerFunction.jar)

If the file name is automatically changed into "AckerFunction.zip", you need to change it back to "AckerFunction.jar".

**Caution:** Even for modest values of m and n, Ackermann's function requires many recursive calls.

## Exercises

1. Type the program given below in the "AckerFunction.java" file. Notice that the private class variable spaces is used to record how "deep" the current recursive call is. It should not be accessible from outside of the class, so there is no getter/setter defined for this variable.
2. When you have finished entering the program, compile and run it. Make sure it works correctly (outputs nothing).
3. Modify the main method to allow user to input two nonnegative integers m and n, call the static method acker, and output the trace messages as well as the total number of method invocations. For example, when the user inputs "1 2 ", the program shall output the result as shown above (in **Description**).
4. Make your program fail-safe for user inputs. In other words, when the user input is invalid, the system should not break.
5. Submit your completed program "AckerFunction.java" to the myCourses!