**Lab 14.1 Beginning to build an ArrayList recursively**

In this sequence of problems we practice recursion by abandoning our reliance on iteration. We resolve to solve a sequence of problems without using while or for loops. Instead we will think recursively and look at the world through a different lens.

Recursion is all about solving a large problem by using the solution to a similar smaller problem. The brilliant thing about recursion is that you can assume you already know how to solve the smaller problem. The goal is to demonstrate how the smaller solution relates to the larger problem at hand. For example, suppose you want to print all binary strings of length 3 and you already know how to print all binary strings of length 2. Here they are:

             00  
             01  
             10  
             11

How can we solve the larger problem with a list of strings of length 2? Add a “0” or “1”, right? So here is the solution to the larger problem:

             00 + 0 = 000  
             01 + 0 = 010  
             10 + 0 = 100  
             11 + 0 = 110  
      and  
             00 + 1 = 001  
             01 + 1 = 011  
             10 + 1 = 101  
             11 + 1 = 111    Voila!

Not every binary string problem can be solved in terms of a smaller one. At some point we have to declare the smallest problem we are interested in and just solve it. In the case of binary strings, we might agree that the smallest problem we will solve is the 1 digit problem and that the solution is simply 0 and 1. When we write solutions to problems using recursion we will always solve the smallest problem first, so our programs will usually have a similar structure:

if this is the smallest problem  
 Just solve it by hand.  
 else  
 Solve the large problem by using a solution to a smaller version of the same problem.

Now let's tackle a real problem. Suppose our goal is to produce an ArrayList with Integer objects 1, 2, 3, …, n by building a method called makeList(int n). We want to end up with a stand-alone method so we will package the method as a static method. Remember we are refusing to use while and for loops – think recursively!

Here is the skeleton code:

import java.util.\*;  
  
 public class ListMethods  
 {  
 public static ArrayList<Integer> makeList(int n)  
 {  
 ArrayList<Integer> tempList = null;  
 if (n <= 0) // The smallest list we can make  
 {  
  
  
  
 }  
 else // All other size lists are created here  
 {  
  
  
  
 }  
 return tempList;  
 }  
 }

The smallest problem we will solve is building and returning an ArrayList when n <= 0. In that case we want to return an empty ArrayList. Leave the else block empty for the moment and write the code that will solve this smallest problem. Test it using the test harness listed below:

import java.util.ArrayList;  
  
 public class ListMethodsRunner  
 {  
 public static void main(String[] args)  
 {  
 ArrayList<Integer> tempList = ListMethods.makeList(0);  
 if (tempList.size() == 0)  
 {  
 System.out.println("The list is empty");  
 }  
 else  
 {  
 for (Integer i : tempList)  
 {  
 System.out.println(i);  
 }  
 }  
 }  
 }

**Lab 14.2 Adding recursion to our ArrayList solution**

We left the else block empty in the last problem. Let’s fill it in now.  
  
This is where we need to show how we can solve the large problem (building an ArrayList with 1, 2, 3, …, n) by using a smaller version of the same problem. What is the next smallest problem? Yes, building an ArrayList with 1, 2, 3, …, n - 1 . How can we do that? Here’s the brilliant part: We can solve that problem by calling makeList(n – 1) – the very same method we are trying to write! That’s what makes the recursion happen. Whenever a method calls itself, the method is recursive. By calling the same method again, each instance of a problem is solved using the solution to a smaller instance. Logically, this process has to stop somewhere. In the case of our ArrayList problem, it stops when we call makeList(0). This is the problem we solved by hand in Lab 13.1.1.  
  
Whenever we call makeList(n - 1) we receive an ArrayList with 1, 2, 3, …, n - 1. How can we use that to produce an ArrayList with 1, 2, 3, …, n?  
  
Add that code to the else block and complete the makeList method. Test your code with the test harness by changing the argument that is passed to makeList. Can you make a list with 100 items?