Recursion

Recursion

Any method that calls itself, either directly or indirectly

Idea, take a problem,
break that problem down into a slightly simpler problem,
ask yourself to solve that slightly simpler problem,
repeat

Example: What is 5+4
"But I only how how to add and subtract 1"

```
so: 5+4
        5+(4-1)  + 1
                5+((3-1)+1) +1
                        5+((2-1+1) +1 + 1
                                5+1+1+1+1
                        6+1+1+1
                7+1+1
        8+1
    9
```

Implementing add1

```
public class AddOne {
    public int adder(int base, int adder) {
        if (adder == 1)
            return base + adder;
        return adder(base, adder - 1) + 1;
    }

    public static void main(String[] args) {
        AddOne addr = new AddOne();
        System.out.println("5+1=" + addr.adder(5, 1));
        System.out.println("5+3=" + addr.adder(5, 3));
        System.out.println("5+5=" + addr.adder(5, 5));
    }
}
```

STOPPING Recursion

Importantly, need a way to stop

```
public void badRecurse(int c) {
  System.out.println("A" + c);
  badRecurse(c-1);
}

public void okRecurse(int c){
  System.out.println("OK" + c);
  if (c==0) return;
  okRecurse(c-1);
}

public void goodRecurse(int c) {
  System.out.println("B" + c);
  if (c<=0) return;
  goodRecurse(c-1);
}
```

```
public void loop(int c) {
    for (int i = c; i >= 0;
             i--) {
        System.out.println(c);
    }
}
```

add1, again

```java
public class AddOne {
    public int adder(int base, int adder) {
        if (adder == 1)
            return base + adder;
        if (adder == 0)
            return base;
        if (adder < 0)
            return -999;
        return adder(base, adder - 1) + 1;
    }

    public static void main(String[] args) {
        AddOne addr = new AddOne();
        System.out.println("5+1=" + addr.adder(5, 1));
        System.out.println("5+3=" + addr.adder(5, 3));
        System.out.println("5+5=" + addr.adder(5, 5));
    }
}
```

Recursive Method

- Base case(s):
  - no recursive calls are performed
  - every chain of recursive calls must reach a base case
- Recursive calls:
  - Calls to the same method in a way that progress is made towards a base case

The Factorial

- Recursive definition: $f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$

Recursive Factorial (pt 1)

```
public class Factorial {
    public static void main(String[] args) {
        String[] defaultArgs = { "5" };
        if (args.length == 0)
            args = defaultArgs;
        try {
            int fb = Integer.parseInt(args[0]);
            Factorial f = new Factorial();
            int b = f.factorial(fb);
            System.out.println(b);
        } catch (NumberFormatException nfe) {
            System.err.println("<<" + args[0] + ">> must be an integer")
        }

    }
}
```

Recursive Factorial (pt 2)

```
/**
 * Compute factorial recursively
 * @param n the number whose factorial you want to compute
 * @return the factorial of the given number
 */
public int factorial(int n) {
    if (n == 1)
        return 1;
    if (n < 1)
        return 0;
    int f = factorial(n - 1);
    return f*n;
}
```

Show step by step in VSC

Recursion — return values

These methods do the same thing

```
public int rAdder(int num1, int num2) {
     if (num2<=0)
         return num1;
     return rAdder(num1+1, num2-1);
 }

public int rAdderB(int num1, int num2) {
     if (num2<=0)
         return num1;
     return 1+rAdderB(num1, num2-1);
 }
```

Practice

```
/** Print the given char the number of times given by num consecutively on
     * the same line. After the last, print a newline.
     * @param ch the char to print
     * @param num the number of times to print the char
     */
    public void rowOfChars(char ch, int num)

    /** Compute the base 2 log of a number.
    * The integer part only.  So base2log(7)==2
    * @param num the number to compute for
    * @return the base 2 log, integer part.
    */
    public int base2log(int num) {
```

so

```
  Recurser r = new Recurser();
  r.rowOfChars('d', 17);
  r.rowOfChars('X', 15);
  System.out.println(r.base2log(7));
  System.out.println(r.base2log(16));
  System.out.println(r.base2log(23));
```

```
ddddddddddddddddd
XXXXXXXXXXXXXXX
2
4
4
```

Recursion and Arrays

Arrays fit naturally with loops

- So, need to simulate a loop
    - private recursive function with a "index" variable

```
public void showArrayLoop(int[] arr) {
    for (int i=0; i<arr.length; i++) {
        System.out.println(i + ": " + arr[i])
    }
}
public void showArray(int[] a) {
    showArrayUtil(a, 0);
}
private void showArrayUtil(int[] a, int loc)
    if (loc >= a.length)
        return;
    System.out.println(loc + ": " + a[loc]);
    showArrayUtil(a, loc + 1);
}
```

Returning an ArrayList

```
    public ArrayList<Integer> makeIncrArray(int size, int maxgap) {
        return makeIncrArrayUtil(size, maxgap, 0, new Random());
    }

   private ArrayList<Integer> makeIncrArrayUtil(int size, int maxgap, int
loc, Random r) {
        if (loc >= size)
            return new ArrayList<Integer>();
        ArrayList<Integer> ai = makeIncrArrayUtil(size, maxgap, loc + 1,
r);
        if (ai.size()>0)
            ai.add(ai.get(ai.size() - 1) + r.nextInt(maxgap)+1);
        else
            ai.add(r.nextInt(maxgap));
        return ai;
    }
```

recursion practice

```
  /**
   * Compute the sum of the components of the array
   */
   public int addArray(int[] array);


   /**
     * Count the number of odd numbers in the
     * provided array
    **/
   public int numOdd(ArrayList<Integer> intArrLis)
```

Recursion — returning values & private recursive functions

```
private BigInteger fibonacciUtil(BigInteger fibNumA, BigInteger fibNumB, 
int counter)
    {
        System.out.println(counter + " " + fibNumA + " " + fibNumB);
        if (counter==1)
            return fibNumA.add(fibNumB);
        return fibonacciUtil(fibNumB, fibNumA.add(fibNumB), counter-1);
    }

   public BigInteger fibonacci(int n) {
        if (n<=0) return BigInteger.valueOf(0);
        if (n<3)  return BigInteger.valueOf(1);
        return fibonacciUtil(BigInteger.valueOf(1), 
BigInteger.valueOf(1), n-2);
    }
```

Count the number of occurrences of a letter in a string

```
public int numOccur1(char ch, String str) {
     if (str == null || str.equals("")) {
          return 0;
     }
     int count = 0;
     if (str.charAt(0) == ch) {
          count++;
     }
     numOccur1(ch, str.substring(1));
     return count;
}
```

What does this return on “a” , “abc”
Why?

Occurrence count v2

```
int acount = 0;

 public int numOccur2(char ch, String str) {
     if (str == null || str.equals("")) {
         return 0;
     }
     if (str.charAt(0) == ch) {
         acount++;
     }
     numOccur2(ch, str.substring(1));
     return acount;
 }
```

Correct, but a BAD solution

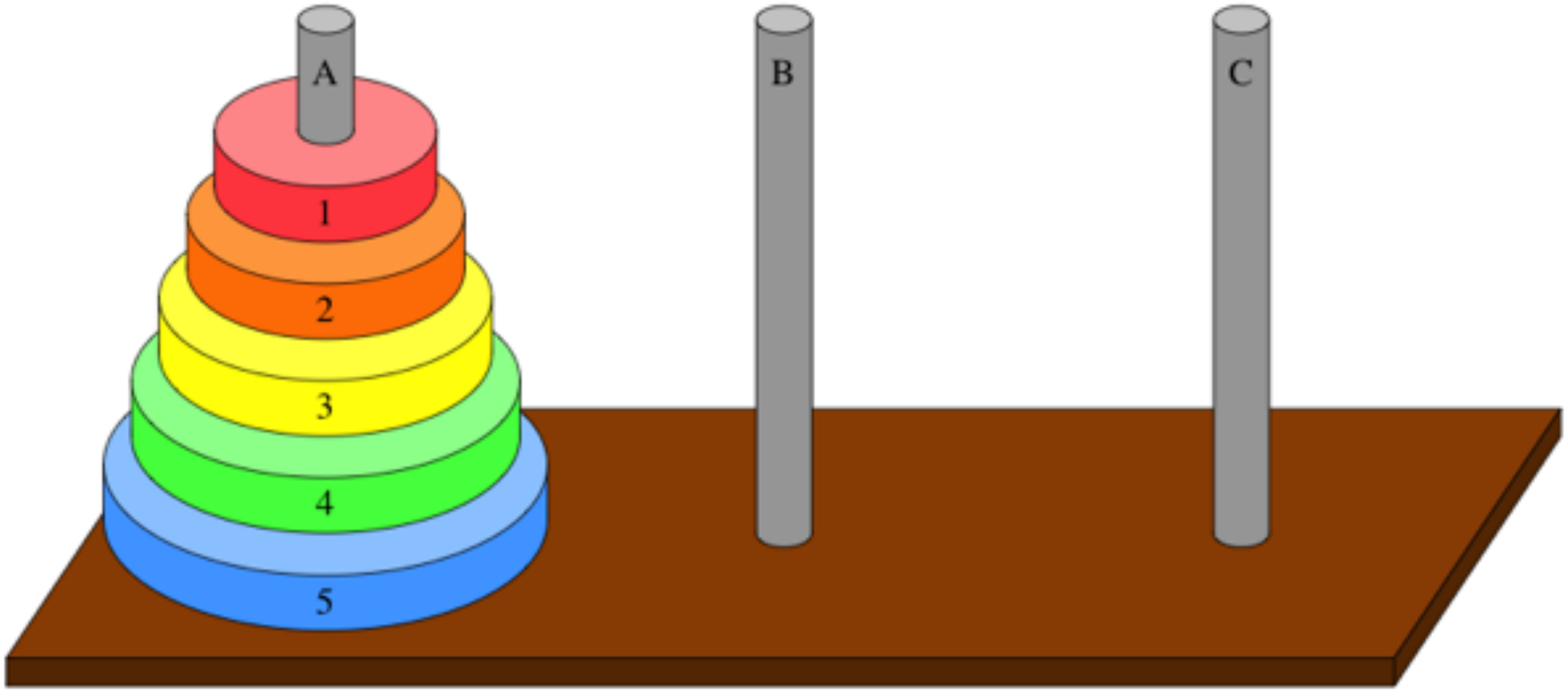
Occurrence count v3 and v4

```
public int numOccur3(char ch, String str) {
    if (str == null || str.equals("")) { return 0; }
    int count = 0;
    if (str.charAt(0) == ch) { count = 1; }
    return count + numOccur3(ch, str.substring(1));
}


public int numOccur4(char ch, String str) {
    return numOccur4Util(ch, str, 0);
}
private int numOccur4Util(char ch, String str, int count) {
    if (str == null || str.equals("")) { return count; }
    if (str.charAt(0) == ch) { count++; }
    return numOccur4Util(ch, str.substring(1), count);
}
```

v5 and v6

```
public int numOccur5(char ch, String str) {
        if (str == null || str.length()==0)
            return 0;
        return numOccur5Util(ch, str, 0, 0);
    }
    private int numOccur5Util(char ch, String str, int loc, int count) {
        if (loc >= str.length())
            return count;
        if (str.charAt(loc) == ch) { count++; }
        return numOccur5Util(ch, str, loc+1, count);
    }

public int numOccur6(char ch, String str) {
        if (str == null || str.length()==0)
            return 0;
        return numOccur6Util(ch, str, 0);
    }
    private int numOccur6Util(char ch, String str, int loc) {
        if (loc >= str.length())
            return 0;
        int cc = 0;
        if (str.charAt(loc) == ch) { cc=1; }
        return cc+numOccur6Util(ch, str, loc+1);
    }
```

Towers of Hanoi



Lab

- Get the code for RArray.java from the website.
  - Copy & paste it into a file in VSC
- Adjust the main function so it only creates fills an array list with numbers and then runs findUtilB
  - all you need to do is comment out a few lines
- Adjust findUtilB so it you know how many recursive calls it makes
  - Hint: Add print statement(s)
- Run code 10 times and record the number of recursive calls
- Adjust code to make an ArrayList of 20,000 rather than 200
- Run code 10 times and record the number of recursive calls
- send averages of the 10 runs to gtowell151@cs.brynmawr.edu