# *Examples of Recursion*

Data Structures in Java with JUnit

*Recursion = Recursion( Again-1 );*

# *A Combinatorial method*

- This example of a recursive solution comes from the field of *Combinatorics*
- Problem: *A D.J. plays 10 songs each hour. There are 40 different songs. How many different one hour sets are possible?*
- Or in general given 40 different things, how many sets of size 10 can be chosen
- Or using the standard notation – *n choose k*

$$\boxed{\binom{n}{k}}$$

# *Recursive n choose k (con.)*

- In any given hour, we could play "Stairway"
- All the possible sets are those that have "Stairway"and those that don't ( sum below)
- After picking this one song--now a simpler problem--the DJ must pick 9 more from the remaining 39 songs (from 39 choose 9)
- Possible sets are those with "Stairway", and those without

$$\binom{40}{10} = \binom{39}{9} + \binom{39}{10}$$

# *From 5 choose 2*

- Consider a simpler problem, from 5 letters choose 2

$$\binom{5}{2} = \binom{4}{1} + \binom{4}{2}$$

A B C D E

- All the sets with A (from 4 choose 1):
  - AB AC AD AE
- Those without A (from 4 choose 2, no A):
  - BC BD BE CD CE DE

# *Recursive n choose k (con.)*

- Or, in general, we get

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

- Rewritten as a method named c that has two parameters n and k:

$$c(n,k) = c(n-1,k-1) + c(n-1,k)$$

- We're getting closer but where is the base case?

# *The base cases for n choose k*

- First, n must be at least as big as k:
  - We cannot choose a 10 song set from 9 songs
- When n == k, there is only one choice
  - only one possible 10 song set from 10 songs
- To be meaningful, k must be at least 1
  - We're not interested in sets with 0 songs
- When k is 1, there are n ways to choose
  - If we only play 1 song sets, and we have 10 songs to choose from, we get n, or 10, possible sets

# *Finally, here is the recursive definition of n choose k*

- The recursive definition of n choose k summarizes all of these points:

$$c(n,k)=\begin{cases} k=1 \Rightarrow n \\ n=k \Rightarrow 1 \\ n>k \;\&\&\; k>1 \Rightarrow c(n-1,k-1)+c(n-1,k) \end{cases}$$

What is c(5, 2)? _____

What is c(4, 1)? _____

What is c(4, 2)? _____

What is c(6, 3)? _____

# *Another Example*

- Next example does not need recursion either
  - but it does provide further insight into recursive solutions
    - and insights into a RecursionFun assignment

# *Converting Decimal Numbers to other bases*

- Problem: Convert a decimal (base 10) number into other bases

| *method Call* | *Return* |
|---|---|
| `convert(99, 2)` | `1100011` |
| `convert(99, 3)` | `10200` |
| `convert(99, 4)` | `1203` |
| `convert(99, 5)` | `344` |
| `convert(99, 6)` | `243` |
| `convert(99, 7)` | `201` |
| `convert(99, 8)` | `143` |
| `convert(99, 9)` | `120` |

# *Digits are multiplied by powers of the base* *10, 8, 2, or whatever*

- First: converting from other bases to decimal
  - Decimal numbers multiply digits by powers of 10

$9507_{10} = 9 \times 10^3 + 5 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$

  - Octal numbers *powers of 8*

$1567_8 = 1 \times 8^3 + 5 \times 8^2 + 6 \times 8^1 + 7 \times 8^0$

$= 512 + 320 + 48 + 7 = 887_{10}$

  - Binary numbers *powers of 2*

$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 8 + 4 + 0 + 1 = 13_{10}$

# *Converting base 10 to base 2*

1) divide number (5) by new base(2), write remainder (1)
2) divide quotient (2), write new remainder (0) to left
3) divide quotient (1), write new remainder (1) to left

```
   __2_
2 )  5     Remainder =  1
   ___1_
2 )  2     Remainder =  0
   ___0_
2 )  1     Remainder =  1
```

Stop when the quotient is 0

$5_{10} = 101_2$

# *Converting base 10 to base 8*

1) divide number by new base (8), write remainder (1)
2) divide quotient (2), write new remainder (0) to left
3) divide quotient (1), write new remainder (1) to left

```
   _12_
8 )99     Remainder =  3
   __1_
8 )12     Remainder =  4
   __0_
8 ) 1     Remainder =  1
```

Stop when the quotient is 0 $99_{10} = 143_8$

# *Possible Solutions*

- We could either
  - store remainders in an array and reverse it or
  - write out the remainders in reverse order
    - have to postpone the output until we get quotient = 0
  - store result as a String (like a Recursion assignment)
- Iterative Algorithm

```
while the decimal number > 0   {
    Divide the decimal number by the new base
    Set the decimal number = decimal number divided by the base
    Store the remainder to the left of any preceding remainders
}
```

# *Recursive algorithm*
# *Base Case -- Recursive Case*

- Base case
  - if decimal number being converted = 0
    - do nothing (or return "")
- Recursive case
  - if decimal number being converted > 0
    - solve a simpler version of the problem
      - use the quotient as the argument to the next call
    - store the current remainder (number % base) in the correct place

# *One solution*

```
assertEquals("14", rf.convert(14, 10));
assertEquals("1100011", rf.convert(99, 2));
assertEquals("143", rf.convert(99, 8));
assertEquals("98", rf.convert(98, 10)); // 9*10+8


public String convert(int num, int base) {
  if (num == 0)
    return "";
  else
    return convert(num/base, base) + (num%base);
}
```

# *Hexadecimal, something we see as Computer Scientists*

- Convert this algorithm to handle all base up through hexadecimal (base 16)
  - 10 = A
  - 11 = B
  - 12 = C
  - 13 = D
  - 14 = E
  - 15 = F

# *Array Processing with recursion*
# *Needs a private helper*

- Modify the given array so that all the even numbers come before all the odd numbers. Other than that, the numbers can be in any order. Use recursion. Do not use a loop

```
evensLeft({1, 0, 1, 0, 0, 1}) → {0, 0, 0, 1, 1, 1}

evensLeft({3, 3, 2}) → {2, 3, 3}

evenLeft({2, 2, 2}) → {2, 2, 2}
```

# *Private Helper*

- Like binaraySearch in the current project, evensLeft does not have enough information passed to the public method
- Solution: Call a recursive private helper method with additional arguments

```
public void evensLeft(int[] a) {
  evensLeft(a, 0, 0); // Maintain where to swap
}
public void evensLeft(int[] a,
                      int i, int swapIndex) {
  // ...
}
```