Announcements

- Test 4 is ???????
  - covers composition and simple inheritance

Recursion

notes Chapter ???

Printing n of Something

- suppose you want to implement a method that prints out n copies of a string

```
public static void printIt(String s, int n) {
  for(int i = 0; i < n; i++) {
    System.out.print(s);
  }
}
```

A Different Solution

- alternatively we can use the following algorithm:
  1. if n == 0 done, otherwise
     I. print the string once
     II. print the string (n – 1) more times

```
public static void printItToo(String s, int n) {
  if (n == 0) {
    return;
  }
  else {
    System.out.print(s);
    printItToo(s, n - 1);    // method invokes itself
  }
}
```

Recursion

- a method that calls itself is called a *recursive* method
- a recursive method solves a problem by repeatedly reducing the problem so that a base case can be reached

```
printItToo("*", 5)
*printItToo ("*", 4)
**printItToo ("*", 3)
***printItToo ("*", 2)
****printItToo ("*", 1)
*****printItToo ("*", 0) base case
*****
```

Notice that the number of times the string is printed decreases after each recursive call to printIt

Notice that the base case is eventually reached.

Infinite Recursion

- if the base case(s) is missing, or never reached, a recursive method will run forever (or until the computer runs out of resources)

```
public static void printItForever(String s, int n) {
  // missing base case; infinite recursion
  System.out.print(s);
  printItForever(s, n - 1);
}
```

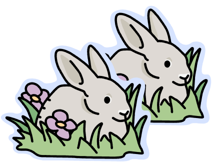
```
printItForever("*", 1)
* printItForever("*", 0)
** printItForever("*", -1)
*** printItForever("*", -2) ...........
```

Climbing a Flight of n Stairs

- not Java

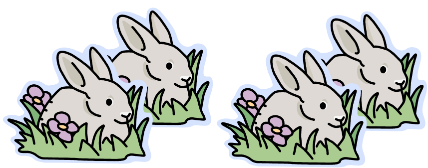
```
/**
  * method to climb n stairs
  */
climb(n) :
if n == 0
  done
else
  step up 1 stair
  climb(n – 1);
end
```

Rabbits



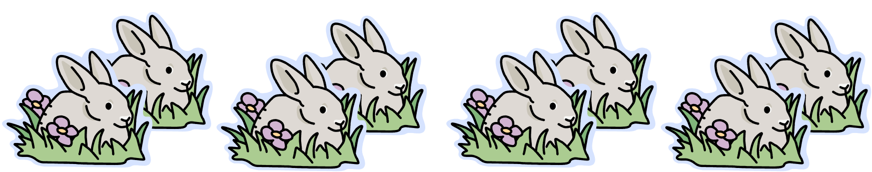
Month 0: 1 pair

0 additional pairs



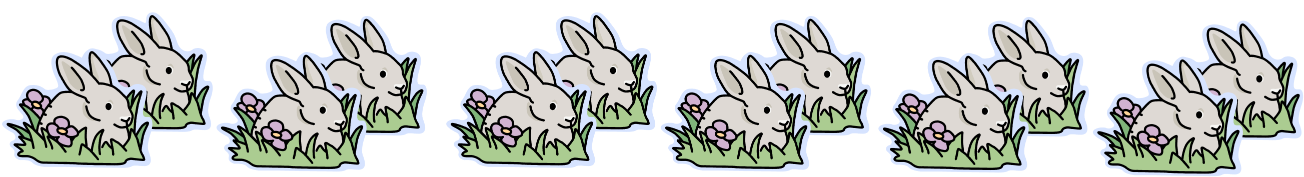
Month 1: first pair makes another pair

1 additional pair



Month 2: each pair makes another pair; oldest pair dies

1 additional pair



2 additional pairs

Month 3: each pair makes another pair; oldest pair dies

Fibonacci Numbers

- the sequence of additional pairs
  - `0, 1, 1, 2, 3, 5, 8, 13, ...`

  are called Fibonacci numbers

- base cases
  - `F(0) = 0`
  - `F(1) = 1`
- recursive definition
  - `F(n) = F(n - 1) +  F(n - 2)`

Recursive Methods & Return Values

- a recursive method can return a value
- example: compute the nth Fibonacci number

```
public static int fibonacci(int n) {
  if (n == 0) {
    return 0;
  }
  else if (n == 1) {
    return 1;
  }
  else {
   int f = fibonacci(n - 1) + fibonacci(n - 2);
   return f;
  }
}
```

Recursive Methods & Return Values

- write a recursive method that multiplies two positive integer values (i.e., both values are strictly greater than zero)

- observation: $m \times n$ means add m n's together
  - in other words, you can view multiplication as recursive addition

Recursive Methods & Return Values

- not Java:

```
/**
 * Computes m * n
 */
multiply(m, n) :
if m == 1
    return n
else
    return n + multiply(m - 1, n)
```

```
public static int multiply(int m, int n) {
  if (m == 1) {
    return n;
  }
  return n + multiply(m - 1, n);
}
```

Recursive Methods & Return Values

- example: write a recursive method **`countZeros`** that counts the number of zeros in an integer number **`n`**
  - **`10305060700002L`** has 8 zeros

- trick: examine the following sequence of numbers

1. **`10305060700002`**
2. **`1030506070000`**
3. **`103050607000`**
4. **`10305060700`**
5. **`103050607`**
6. **`1030506 ...`**

Recursive Methods & Return Values

- not Java:

```
/**
 * Counts the number of zeros in an integer n
 */
countZeros(n) :
if the last digit in n is a zero
    return 1 + countZeros(n / 10)
else
    return countZeros(n / 10)
```

- don't forget to establish the base case(s)
  - when should the recursion stop? when you reach a single digit (not zero digits; you never reach zero digits!)
    - base case #1 : **`n == 0`**
      - **`return 1`**
    - base case #2 : **`n != 0 && n < 10`**
      - **`return 0`**

```
public static int countZeros(long n) {

  if(n == 0L) {  // base case 1
    return 1;
  }
  else if(n < 10L) {  // base case 2
    return 0;
  }

  boolean lastDigitIsZero = (n % 10L == 0);
  final long m = n / 10L;
  if(lastDigitIsZero) {
    return 1 + countZeros(m);
  }
  else {
    return countZeros(m);
  }
}
```

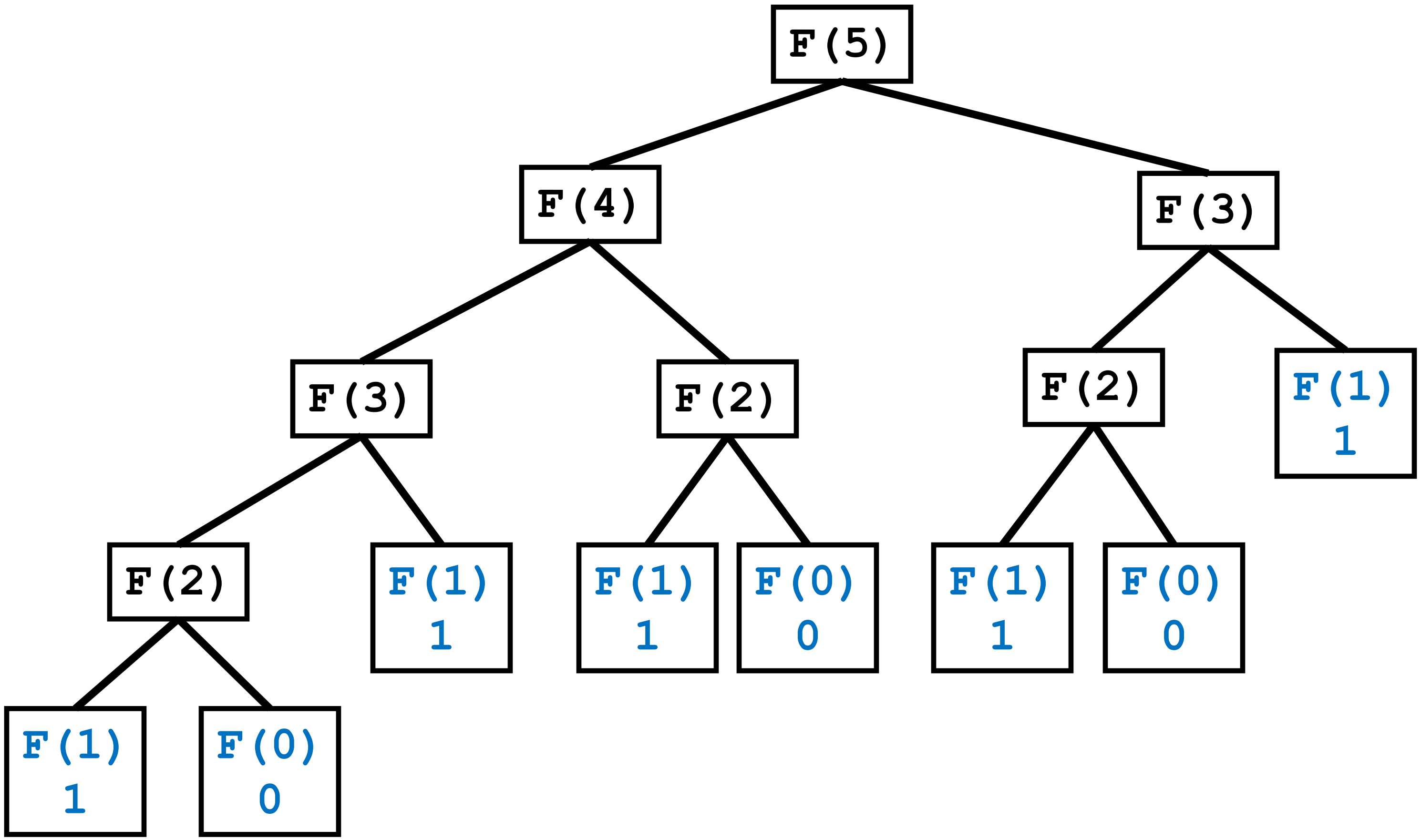
countZeros Call Stack

`callZeros( 800410L )`

last in first out

`callZeros( 8L )`	0
`callZeros( 80L )`	1 + 0
`callZeros( 800L )`	1 + 1 + 0
`callZeros( 8004L )`	0 + 1 + 1 + 0
`callZeros( 80041L )`	0 + 0 + 1 + 1 + 0
`callZeros( 800410L )`	1 + 0 + 0 + 1 + 1 + 0
	= 3

Fibonacci Call Tree



Compute Powers of 10

- write a recursive method that computes 10^n for any integer value **n**
- recall:
  - $10^0 = 1$
  - $10^n = 10 * 10^{n-1}$
  - $10^{-n} = 1 / 10^n$

```java
public static double powerOf10(int n) {
  if (n == 0) {
    // base case
    return 1.0;
  }
  else if (n > 0) {
    // recursive call for positive n
    return 10.0 * powerOf10(n - 1);
  }
  else {
    // recursive call for negative n
    return 1.0 / powerOf10(-n);
  }
}
```