Classic Recursion Examples

Plus Class Methods and Applications

Tuesday, October 23, 2007

CS111 Computer Programming

Department of Computer Science
Wellesley College

Goals for Today

1. Present some classic recursion examples that every computer scientist is expected to know: factorial, fibonacci, and Towers of Hanoi.
2. Introduce tail recursion = recursion without pending operations.
3. Introduce some aspects of Java along the way that are helpful for the above:
   - Class (static) methods
   - Strings and concatenation
   - Applications with command-line input
   - Using System.out.println to trace method calls.

Class (`static`) Methods

A class method (marked by the keyword **static**) is a method whose behavior does not depend on a receiver instance.

Here are some examples from the Math class (which does not have any constructor methods or instances):

```
public static int abs (int a); Returns the absolute value of a
public static int max (int a, int b); Returns the greater of a and b.
public static double max (double a, double b); Returns the greater of a and b.
// Similar for min
public static int round (double a); Returns the integer closest to a.
public static double sqrt (double a); Returns the square root of a.
```

Class methods invoked using a class name to the left of the dot:

```
Math.abs(-3) // evaluates to 3
Math.max(4,7) // evaluates to 7
Math.max(5.1,4.2) // evaluates to 5.1
Math.round(5.1) // evaluates to 5
Math.sqrt(2.0) // evaluates to 1.4142135623730951
```


Defining your Own Class Methods

```
public class IntFunctions {

  public static int square (int x) {
    return x*x;
  }

  public static int sumOfSquares (int a, int b) {
    return IntFunctions.square(a) + IntFunctions.square(b);
    // could write square(a) instead of IntFunctions.square(a)
  }

  // E.g. :
  // IntFunctions.square(3) evaluates to 9
  // IntFunctions.sumOfSquares(3,4) evaluates to 25

}
```

Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:

Execution Land

```
System.out.println(IntFunctions.sumOfSquares(3,4));
```


Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:

Execution Land

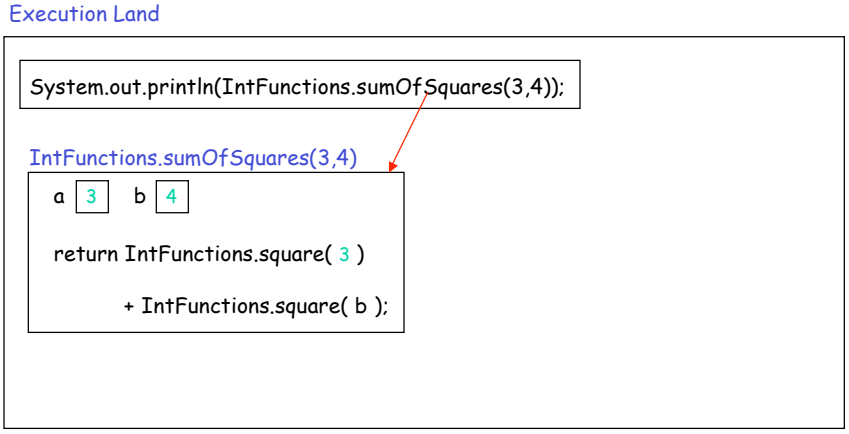
```
System.out.println(IntFunctions.sumOfSquares(3,4));
```

IntFunctions.sumOfSquares(3,4)

```
a 3   b 4

return IntFunctions.square( a )
       + IntFunctions.square( b );
```

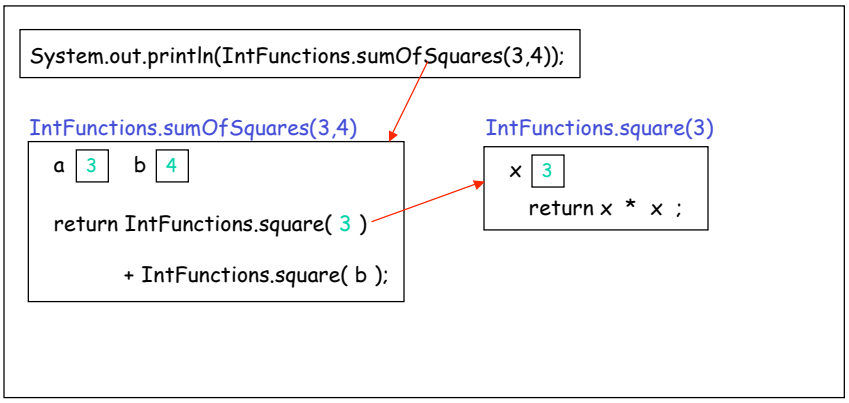
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



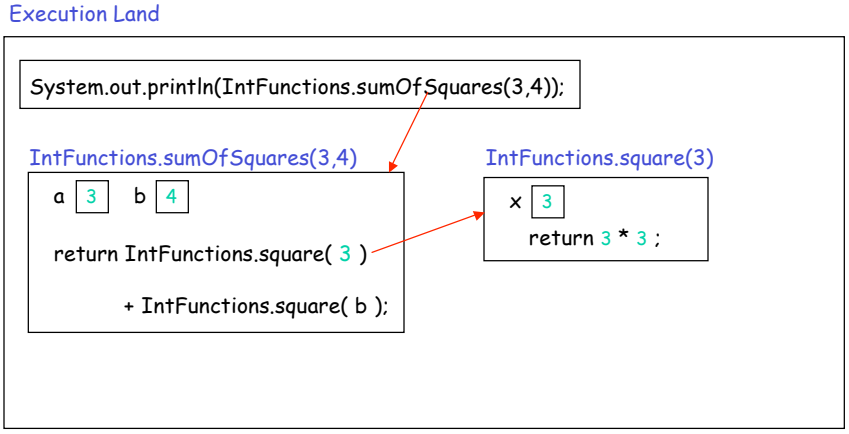
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



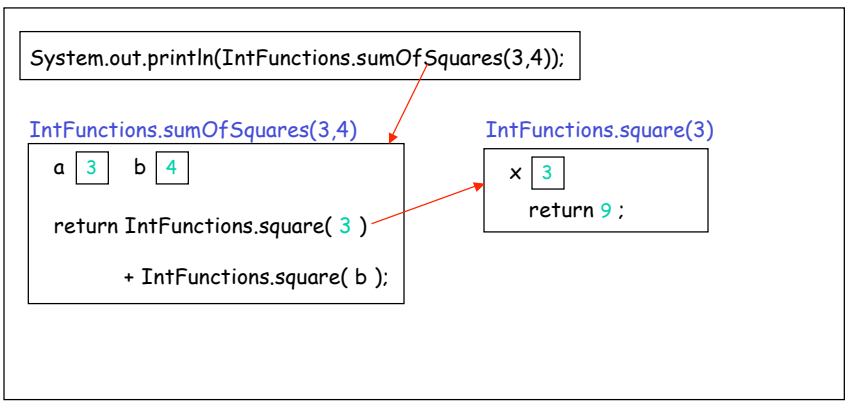
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



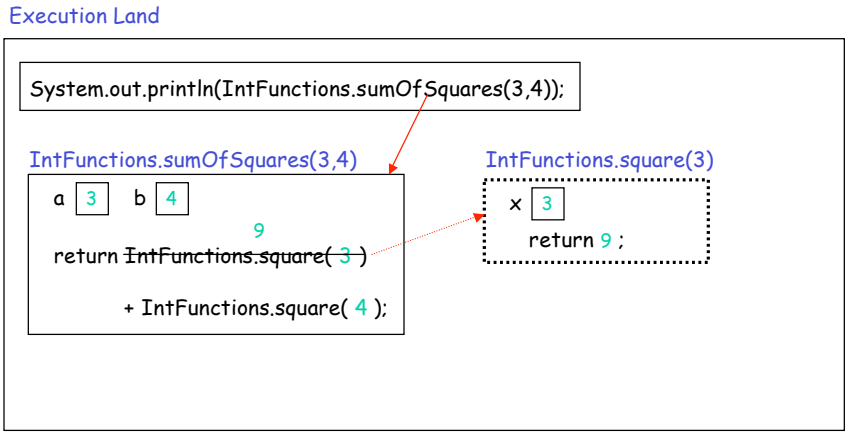
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



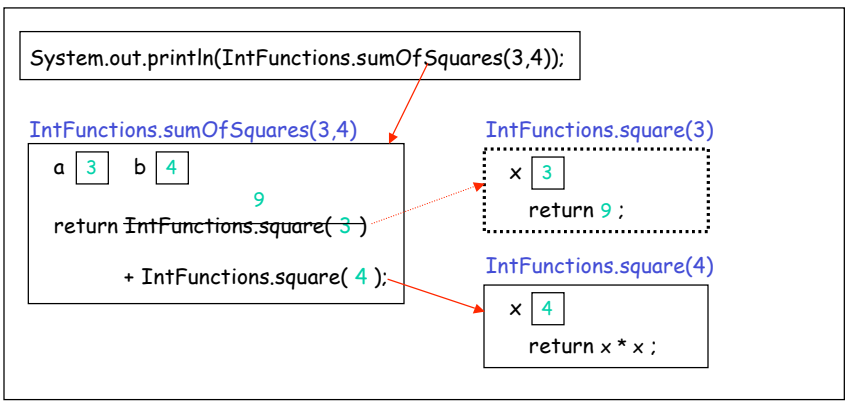
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



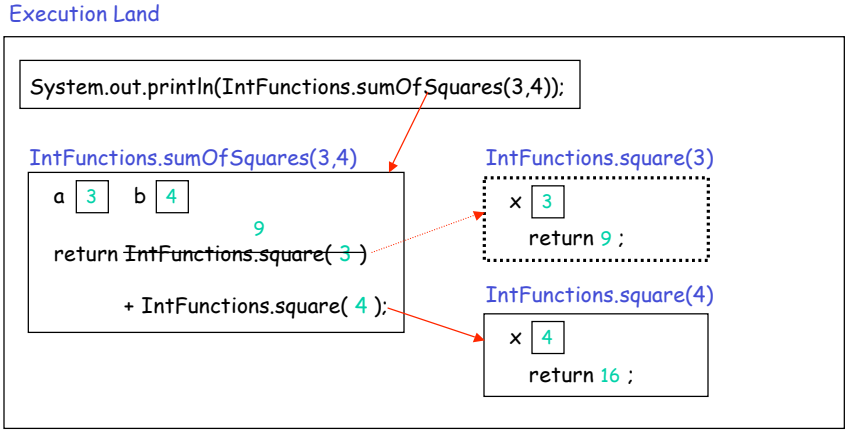
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



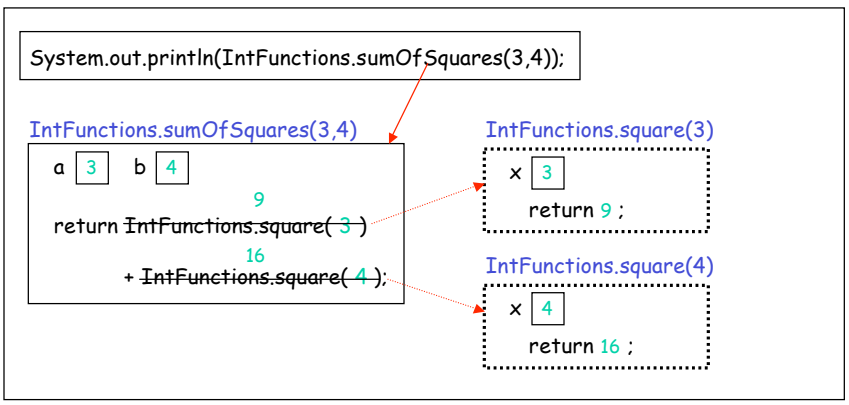
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



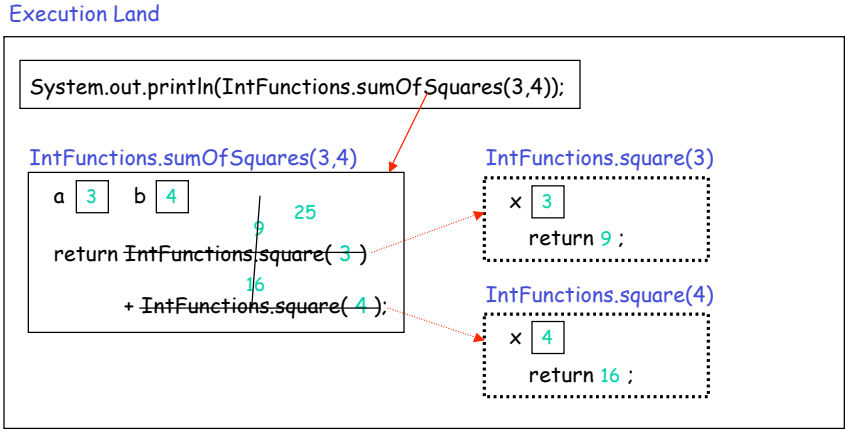
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



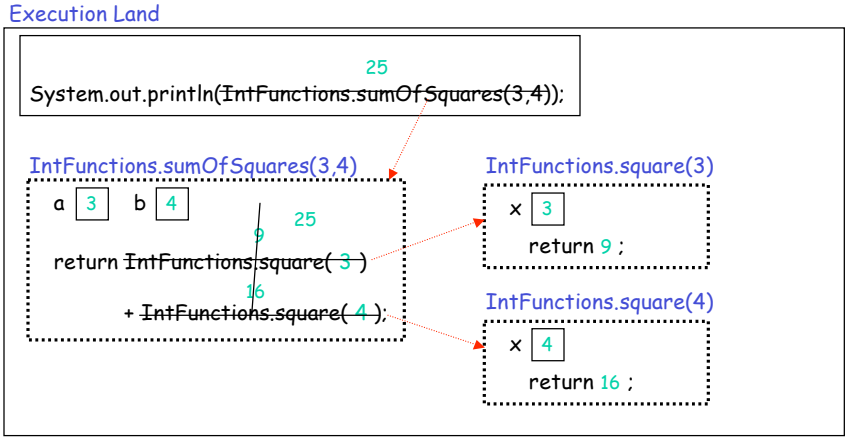
Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



Class Methods in a JEM

In a JEM, a class method has no receiver object or **this** variable. E.g.:



Strings and String Concatenation

A string is a sequence of characters.

Java strings can be written as characters delimited by double quotes:

```
"cat"  "Computer Science"  "CS111 rocks!"
```

In Java, strings are concatenated with + :

```
"CS" + "111" + " " + "rocks!" // evaluates to "CS111 rocks!"
```

Strings can be displayed in the console using System.out.println:

```
System.out.println("cat"); // displays cat
System.out.println("CS" + "111" + " " + "rocks!" ) // displays CS111 rocks!
```


Strings of Digits vs. Numbers

Strings of digits are different from numbers:

```
111 + 234 // evaluates to 345
"111" + "234" // evaluates to "111234"
```

Can convert between strings of digits and numbers:

```
Integer.toString(111) + Integer.toString(234) // evaluates to "111234"
Integer.parseInt("111") + Integer.parseInt("234") // evaluates to 345
Integer.parseInt("12c") // aborts the program with an exception
```

If one argument of + is a string, Java will automatically convert the other:

```
"111" + 234 // evaluates like "111" + Integer.toString(234) -> "111234"
111 + "234" // evaluates like Integer.toString(111) + "234" -> "111234"
```

This facilitates displaying information from a program. E.g.:

```
int a = 5;
int b = 3;
System.out.println( "a = " + a
                    + "; b = " + b
                    + "; Math.min(a,b) = " + Math.min(a,b) + ";")
// displays a = 5; b = 3; Math.min(a,b) = 3;
```

A Simple Java Application

An application is a Java class that can be invoked from the top-level system, such as the Dr. Java Interaction Pane, by typing **java *<className>***.

This invokes the class method named main() within the class.

```
public class IntTest1 {
  public static void main (String[] args) {
      int a = 5;
      int b = 3;
      System.out.println("Math.min(" + a + ", " + b + ") = " + Math.min(a,b));
      System.out.println("IntFunctions.square(" + a + ") = " + IntFunctions.square(a));
      System.out.println("IntFunctions.sumOfSquares(" + a + ", " + b + ") = "
                         + IntFunctions.sumOfSquares(a,b));
  }
}
```

boilerplate header for main()

Dr. Java Interaction Pane

```
> java IntTest1
Math.min(5, 3) = 3
IntFunctions.square(5) = 25
IntFunctions.sumOfSquares(5, 3) = 34
```


A Java Application with Arguments

The *args* parameter in the main() method denotes an array of string arguments passed in from the top-level system: args[0] is the first string argument, args[1] is the second string argument, and so on.

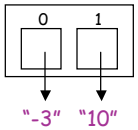
```
public class IntTest2 {
  public static void main (String[] args) {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      System.out.println("Math.min(" + a + ", " + b + ") = " + Math.min(a,b));
      System.out.println("IntFunctions.square(" + a + ") = " + IntFunctions.square(a));
      System.out.println("IntFunctions.sumOfSquares(" + a + ", " + b + ") = "
                         + IntFunctions.sumOfSquares(a,b));
  }
}
```

boilerplate header for main()

Dr. Java Interaction Pane

```
> java IntTest2 -3 10
Math.min(-3, 10) = -3
IntFunctions.square(-3) = 9
IntFunctions.sumOfSquares(-3, 10) = 109
```

In this case, args is the array:



Classic Recursion: Factorial

4! = 4 x (3 x 2 x 1)
= 4 x 3!

recursive cases

3! = 3 x (2 x 1)
= 3 x 2!

Likewise

2! = 2 x (1)
= 2 x 1!

1! = 1 X 0!

0! = 1

base case

Factorial in Java

```
public static int fact (int n) {




}
```

So, if n==4, fact(4)

24

4*fact(3)

6

3*fact(2)

2

2*fact(1)

1

1*fact(0)

1

1

A Factorial Application

```
public class Factorial {

  public static int fact (int n) {
    if (n == 0) {
      return 1;
    } else {
      return n * fact (n - 1);
    }
  }

  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    System.out.println("fact(" + n + ") = " + fact(n));
  }
}
```


A Factorial Application with Tracing

```
public class FactorialTrace {

public static int fact (int n) {
    System.out.println("Entering fact(" + n + ")");
    int result;
    if (n == 0) {
        result = 1;
    } else {
        result = n * fact (n - 1);
    }
    System.out.println("Exiting fact(" + n + ") with " + result);
    return result;
  }

  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    System.out.println("fact(" + n + ") = " + fact(n));
  }
}
```

What does `java FactorialTrace 3` Display?

```
> java FactorialTrace 3
```


Tail Recursion

A recursive method is **tail-recursive** when it has one recursive subproblem and no pending operations. In this case, conquering the subproblem solves the whole problem.

Which of the following Buggle methods is tail-recursive? Why?

```
public void bagelForward(int n) {
  if (n == 0) { // do nothing
  } else {
    dropBagel();
    forward();
    bagelForward(n-1);
  }
}
```

```
public void bagelLine(int n) {
  if (n == 0) { // do nothing
  } else {
    dropBagel();
    forward();
    bagelLine(n-1);
    backward();
  }
}
```

```
public void bagelsToWall() {
  if (isFacingWall()) {
    dropBagel();
  } else {
    dropBagel();
    forward();
    bagelsToWall();
  }
}
```

```
public void bagelsToWallAndBack() {
  if (isFacingWall()) {
    dropBagel();
  } else {
    dropBagel();
    forward();
    bagelsToWallAndBack();
    backward();
  }
}
```

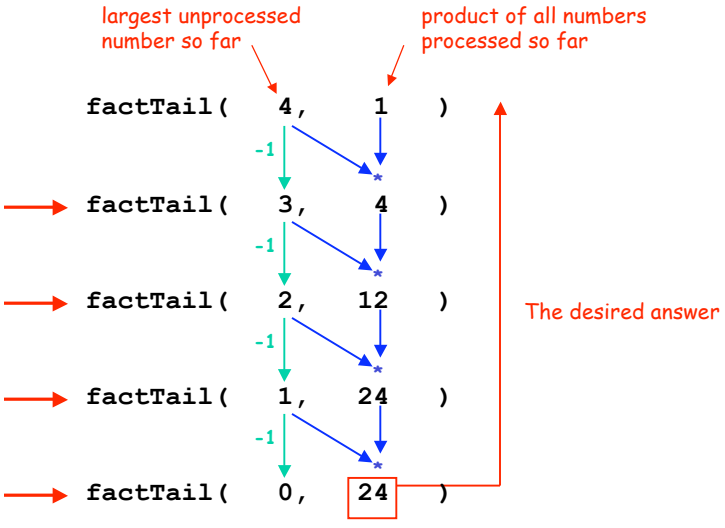
Tail Recursion in TurtleWorld

Which of the following Turtle methods is tail-recursive? Why?

```
public void spiral(int steps,
                   int angle,
                   int length,
                   int increment) {
  if (steps <= 0) {
    // do nothing
  } else {
    fd(length);
    lt(angle);
    spiral(steps-1,
           angle,
           length+increment,
           increment);
  }
}
```

```
public void tree(int levels, double length,
                 double angle, double shrink) {
  if (levels <= 0) {
    // do nothing
  } else {
    fd(length);
    rt(angle);
    tree(levels-1, length*shrink, angle, shrink);
    lt(2*angle);
    tree(levels-1, length*shrink, angle, shrink);
    rt(angle);
    bd(length);
  }
}
```


A Tail-Recursive Factorial



Defining factTail() in Java

serves as a place to pass answer along

```
// factorial using tail recursion
public static int factTail (int num, int ans) {




}
```


An Application with factTail()

```
public class FactorialTail {

  public static int fact (int n) {
    return factTail(n, 1);
    // seed = initial answer = empty product = 1.
  }

  public static int factTail (int num, int ans) {
    if (num == 0) {
      return ans;
     } else {
      return factTail(num - 1, num * ans);
    }
  }

  public static void main(String [] args) {
    int n = Integer.parseInt(args[0]);
    System.out.println("fact(" + n + ") = " + fact(n));
  }
}
```

An Application with factTail() and Tracing

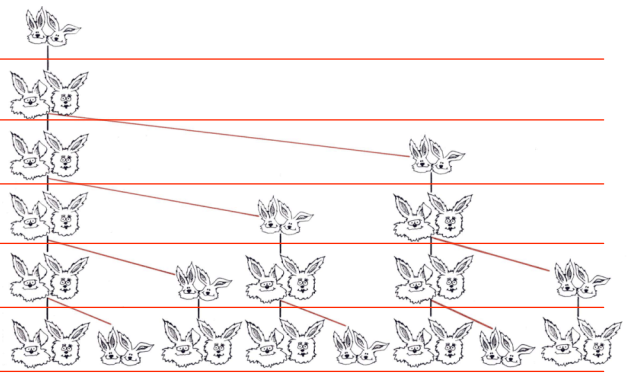
```
public class FactorialTailTrace {

  public static int fact (int n) {return factTail(n, 1);}

  public static int factTail (int num, int ans) {
    if (num == 0) {
      System.out.println(factTail(" + num + "," + ans + ")"
                         + " returns " + ans);
      return ans;
    } else {
      System.out.println("factTail(" + num + "," + ans + ")");
      return factTail(num - 1, num * ans);
    }
  }

  public static void main(String [] args) {
    int n = Integer.parseInt(args[0]);
    System.out.println("fact(" + n + ") = " + fact(n));
  }
}
```


java FactorialTailTrace 3

```
> java FactorialTailTrace 3
```

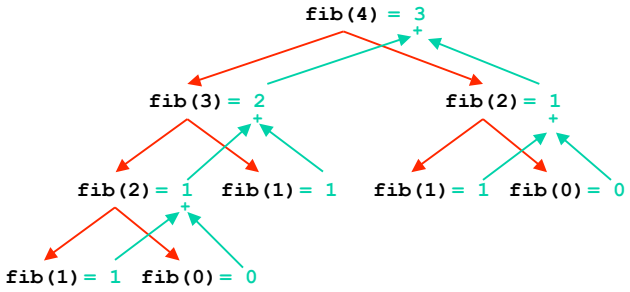
Leonardo Pisano Fibonacci

Time	# Pairs
0	0
1	1
2	1
3	2
4	3
5	5
6	8

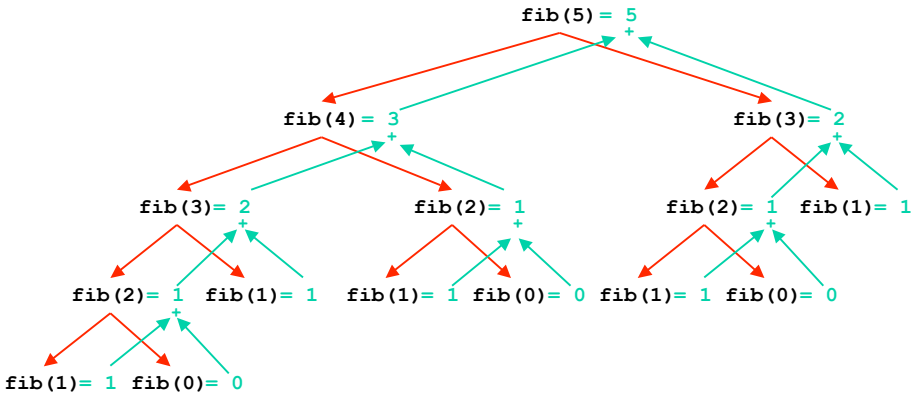


The fib() Method

```
public static int fib (int n) { // Returns pairs at time n
    if (n < 2) { // Assume n >= 0
        return n;
    } else {
        return fib(n-1)     // pairs alive last month
               + fib(n-2); // newborn pairs
    }
}
```

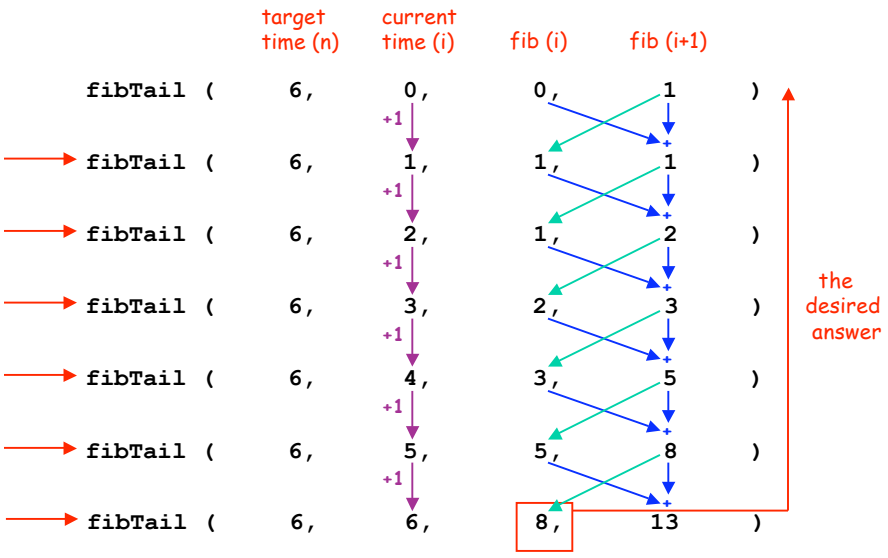


It gets worse



How long would it take to calculate fib(100)?

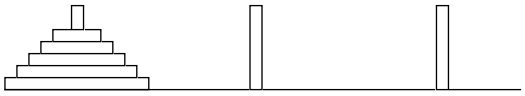
Is there a better way? Yes!



A Faster Fibonacci

```
public class FibonacciFast {

  public static int fib (int n) {
    return fibTail(n, 0, 0, 1);
  }

  public static int fibTail (int n, int i,
                             int fib_i, int fib_i_plus_1) {









}

  public static void main (String[] args) {
    int n = Integer.parseInt(args[0]);
    System.out.println("fib(" + n + ") = " + fib(n));
  }
}
```


Towers of Hanoi



Solving Hanoi in Java

```
public static void main (String[] args) {
  int n = Integer.parseInt(args[0]);
  System.out.println("-----------------------------------");
  System.out.println("Instructions for solving Hanoi("
                      + n + ", A, B, C):”);
  hanoi(n, "A”, "B”, "C”);
}

public static void moveDisk (int disk, String source, String dest {
 // Doesn't move a disk, but prints out an instruction to do so.
 System.out.println("Move disk " + disk + " from peg " + source
                    + " to peg " + dest);
}

public static void hanoi (int n, String source,
                          String dest,String spare) {




}
```