Recursion

Lecture 7

Object-Oriented Programming

Agenda

- Why Recursion?
- Definition
- Examples
- The Three Rules of Recursion
- Recursive Methods
- Activation Records
- Tree
- Equilateral Triangle
- Koch's Snowflake
- Towers of Hanoi
- Recursive Factorial Demo
- Fibonacci Series

Russian Dolls

Each doll contains a smaller doll which in turn contains a smaller doll…

Why Recursion?

- Why learn recursion?
  - New mode of thinking.
  - Powerful programming paradigm.
- Many computations are naturally self-referential.
  - Many Sorting Techniques.
  - A folder contains files and other folders.
  - Factorial
  - Trees and Spirals

Recursion Defined

- Models problems that are *self-similar*
  - Decompose whole task into smaller, simpler sub-tasks that are similar
  - Thus, each sub-task can be solved by applying similar technique
- Whole task solved by combining solutions to sub-tasks
  - Special form of *divide and conquer*
- Task is defined in terms of itself
  - In Java, modeled by method that calls itself
  - Requires *base case* (case simple enough to be solved directly, without recursion) to end recursion; otherwise infinite recursion

Recursion

- Recursion solves a problem by solving a smaller instance of the same problem.
- Think divide, conquer, and glue when all the subproblems have the same "shape" as the original problem.

Russian Dolls

- Take Example of the Russian Dolls
  - Each doll contains a smaller doll which in turn contains a smaller doll…
- How do you know that there are no more dolls?
- Could we apply the same procedure to count the participants of this class?

Counting the Number of Students

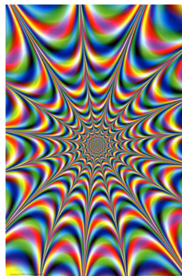
- Lets try it out….

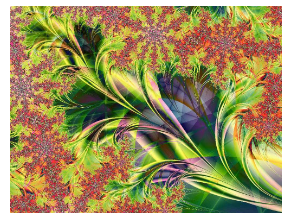
```
count(C)
    if (C == 1)
    {
        return 1
    }
    else
    {
        count(C-1)
        n = n + 1
    }
```

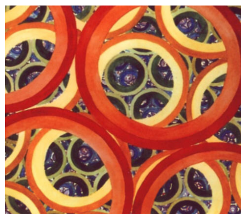
Remember Turtles

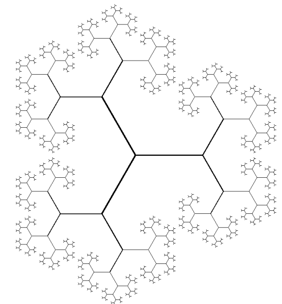
- Use turtle to draw *fractal* shapes:
  - complex shapes that are composed of smaller, simpler copies of some pattern
  - branch of mathematics developed by Benoit Mandelbrot: characteristic is self-similarity
  - natural for recursion!

Some Example of Fractals

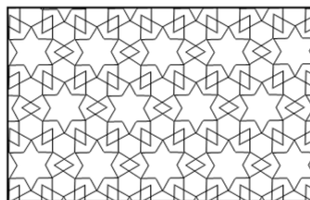








Some Islamic Art is Fractals Also



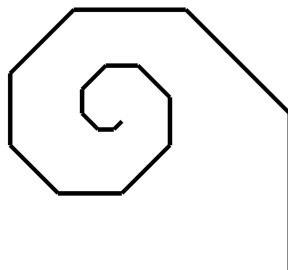
Drawing a Spiral

- Some Simple Fractal examples:
  - spiral, tree, and snowflake

- Let's start with the simplest: a spiral

  The spiral starts at a particular point. It is made of successively shorter lines, each line at an angle to the previous one. The length of the first side (the longest one), spiral's angle, and amount by which to decrement spiral's sides can be specified by the user.

Drawing a Spiral



Which angle are we turning?

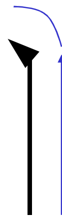
The Three Rules of Recursion

- Approach every recursive problem as if it were a journey; if you follow these rules, you will complete the journey successfully.

- RULE 1: Find out how to take just one step.
- RULE 2: Break each journey down into one step plus a smaller journey.
- RULE 3: Know when to stop.

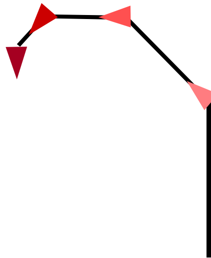
Drawing a Spiral Using the Turtle

•First Step: Move turtle forward to draw line and turn some degrees. What's next?



Drawing a Spiral Using the Turtle

Draw smaller line and turn! Then another, and another...



Recursive Methods

- Recursive methods
  - So far we are used to a method containing different message sends to *this*, but now we send the same message to *this*
  - Method must handle successively smaller versions of original task
- Method's variable(s):
  - As with separate methods, each invocation (message send) has its own copy of parameters and local variables, and shares access to instance variables
  - Record of code, all parameters and local variables is called *activation record*

Activation Records

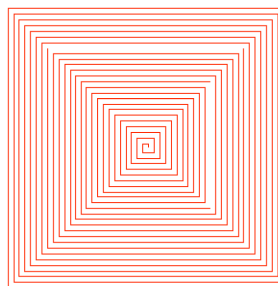
- With recursion,
  - Many activations of single method may exist at once
  - At base case, as many exist as depth of recursion
  - Each has its own copy of local variables, parameters
  - Each activation of a method is stored on the "activation stack"

Very Important Concept

Activation Record of the Spiral

- `public void spiral(steps, angle, length, increment)`
- e.g. spiral(100 , 90 , 0 , 3)

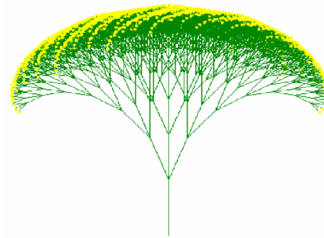
After 100 Steps



Tree

The tree is composed of a trunk that splits into two, smaller branches that sprout in opposite directions at the same angle. Each branch then splits as the trunk did until the sub-branch can no longer be seen, then it is drawn as a leaf. The length of a tree's main trunk, angle at which branches sprout, and amount to decrement each branch can be specified by user.

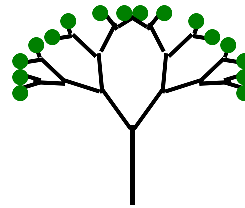
Or a Broccoli or a Cauliflower



Drawing a Tree using Turtles

- Broccoli or Cauliflower are essentially trees
- Compare each left branch to corresponding right branch:
  - Right branch is simply rotated copy of left branch
- Branches are themselves smaller trees!
  - branches are themselves smaller trees!
    - Branches are themselves smaller trees!

      …
- Our tree is recursive!
  - Base case is leaf

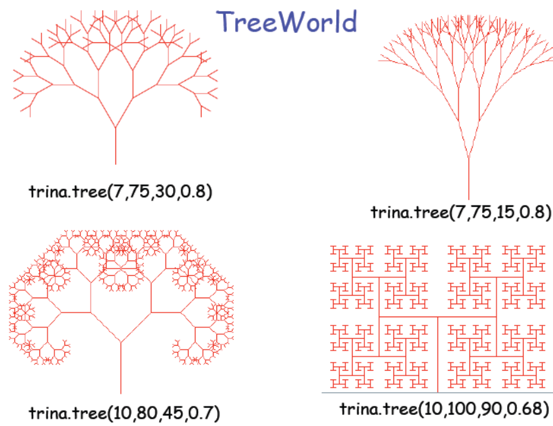


Drawing a Tree

```
public void tree(int levels, double
     length, double angle, double shrink)
```

- *Levels* states the number of branches the tree is going to have.
- *Length* states the length of the initial branch.
- *Angle* states the relatives angle of the two sibling branches.
- *Shrink* is the factor with which the branch length is going to shrink.

Types of Trees



Equilateral Triangle

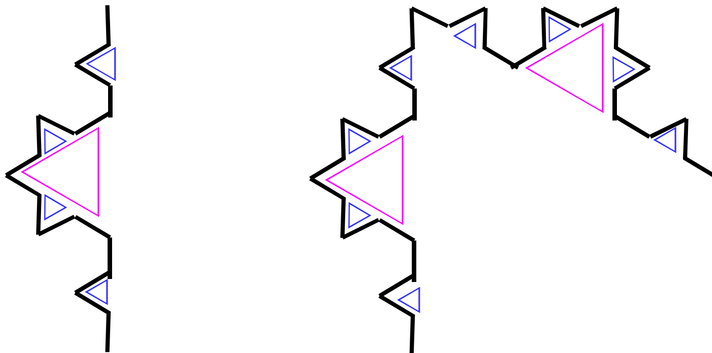
- What is an equilateral triangle?
- How can you draw an equilateral triangle using a turtle?

Koch's Snowflake

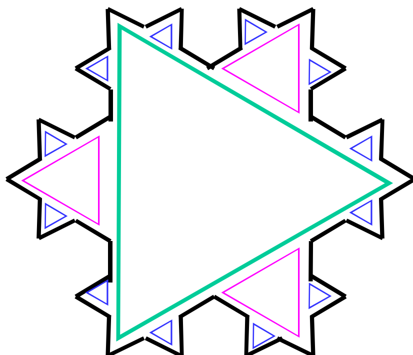
- Invented by Swedish mathematician, Helge von Koch, in 1904; also known as *Koch Island*

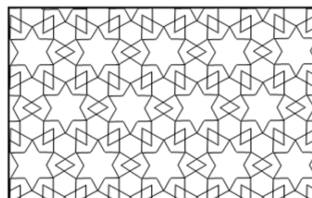
The snowflake is created by taking an equilateral triangle and partitioning each side into three equal parts. Each side's middle part is then replaced by another equilateral triangle (with no base) whose sides are one third as long as the original. This process is repeated for each remaining line segment. The length of the initial equilateral triangle's side can be given by the user.

Koch's Snowflake



Koch's Snowflakes



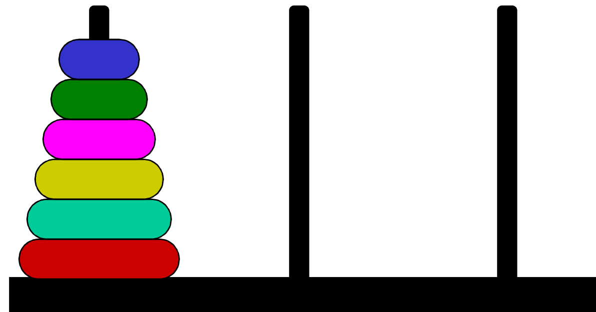


Any Similarity?

Towers of Hanoi

- Game invented by French mathematician Edouard Lucas in 1883
- **Goal:** move tower of *n* disks, each of a different size, from left-most peg to right-most peg
- **Rule 1:** no disk can be placed on top of smaller disk
- **Rule 2:** only one disk can be moved at once

Towers of Hanoi



Towers of Hanoi

- One disk:
  - move disk to final pole
- Two disks:
  - use one disk solution to move top disk to intermediate pole
  - use one disk solution to move bottom disk to final pole
  - use one disk solution to move top disk to final pole
- Three disks
  - use two disk solution to move top disks to intermediate pole
  - use one disk solution to move bottom disk to final pole
  - use two disk solution to move top disks to final pole
- In general (for n disks)
  - use $n - 1$ disk solution to move top disks to intermediate pole
  - use one disk solution to move bottom disk to final pole
  - use $n - 1$ disk solution to move top disks to final pole

Recursive Factorial Demo

by Robert Sedgewick and Kevin Wayne

```
pubic 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) {
      System.out.println(fact(3));
   }
}
```

n = 3

environment

fact(3)

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

n = 3

environment

fact(3)

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


n = 3

environment

fact(3)

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

n = 3
environment

fact(3)

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

n = 2
environment

fact(2)

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


n = 3
environment

fact(3)

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

n = 2
environment

fact(2)

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

n = 3

environment

fact(3)

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

n = 2

environment

fact(2)

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


n = 3

environment

fact(3)

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

n = 2

environment

fact(2)

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

n = 1

environment

fact(1)

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

n = 3
environment

fact(3)

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

n = 2
environment

fact(2)

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

n = 1
environment

fact(1)

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


n = 3
environment

fact(3)

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

n = 2
environment

fact(2)

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

n = 1
environment

fact(1)

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

n = 3
environment

fact(3)

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

n = 2
environment

fact(2)

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

n = 1
environment

fact(1)

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

n = 0
environment

fact(0)

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

Lecture 7 43

n = 3
environment

fact(3)

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

n = 2
environment

fact(2)

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

n = 1
environment

fact(1)

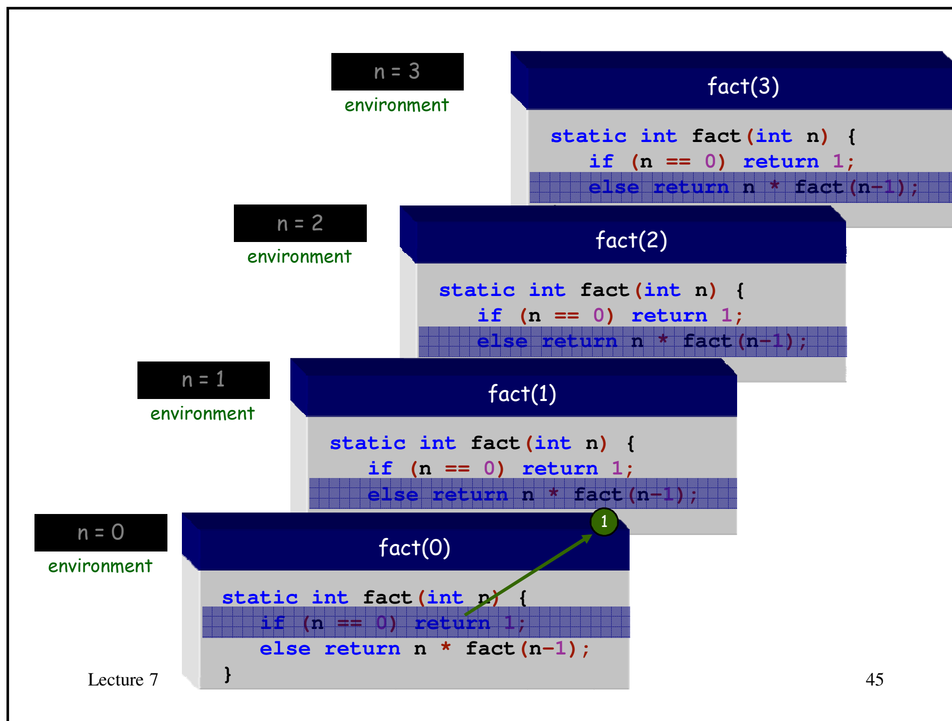
```
static int fact(int n) {
   if (n == 0) return 1;
   else return n * fact(n-1);
```

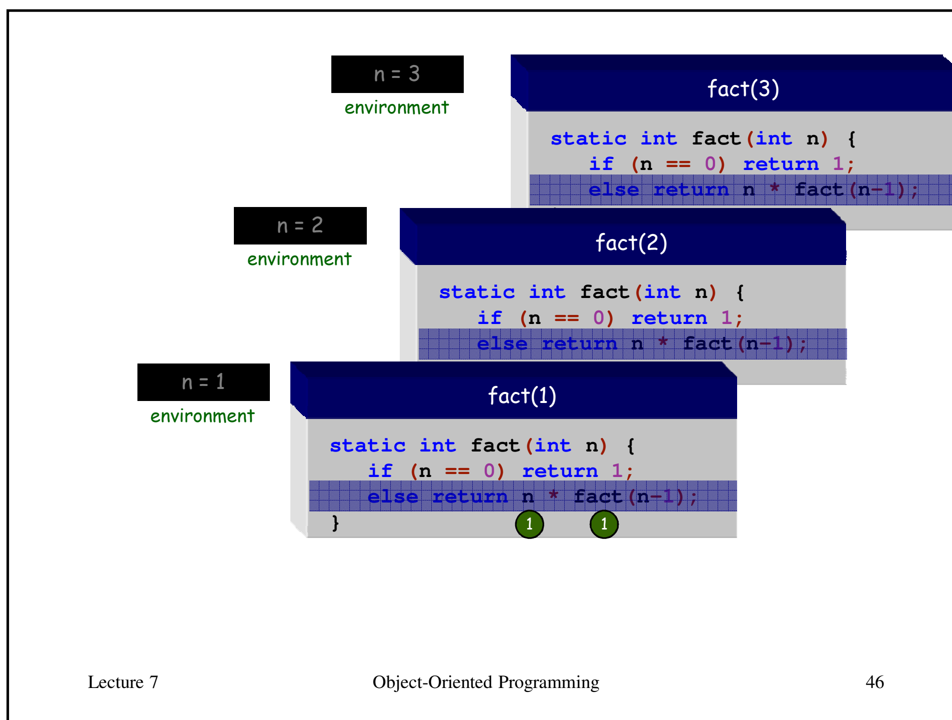
n = 0
environment

fact(0)

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

Lecture 7 44


n = 3
environment
fact(3)
static int fact(int n) {
if (n == 0) return 1;
else return n * fact(n-1);
n = 2
environment
fact(2)
static int fact(int n) {
if (n == 0) return 1;
else return n * fact(n-1);
n = 1
environment
fact(1)
static int fact(int n) {
if (n == 0) return 1;
else return n * fact(n-1);
n = 0
environment
fact(0)
1
static int fact(int n) {
if (n == 0) return 1;
else return n * fact(n-1);
}
Lecture 7
45


n = 3
environment
fact(3)
static int fact(int n) {
if (n == 0) return 1;
else return n * fact(n-1);
n = 2
environment
fact(2)
static int fact(int n) {
if (n == 0) return 1;
else return n * fact(n-1);
n = 1
environment
fact(1)
static int fact(int n) {
if (n == 0) return 1;
else return n * fact(n-1);
}
1
1
Lecture 7
Object-Oriented Programming
46

n = 3
environment

fact(3)

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

n = 2
environment

fact(2)

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

1

n = 1
environment

fact(1)

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

1 1

n = 3
environment

fact(3)

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

n = 2
environment

fact(2)

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

2 1

n = 3
environment

fact(3)

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

2

n = 2
environment

fact(2)

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

2 1

n = 3
environment

fact(3)

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

3 2

n = 3
environment

fact(3)

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

3 2

```
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) {
      System.out.println(fact(3));
   }
}
```

6

```
% java Factorial
6
```


Fibonacci Series

- Fibonacci series
  - 0, 1, 1, 2, 3, 5, 8, 13, 21…
  - fibonacci(0) = 0
    fibonacci(1) = 1
    fibonacci(n) = fibonacci(n - 1) + fibonacci(n – 2)
  - fibonacci(0) and fibonacci(1) are base cases
  - Each number in the series is sum of two previous numbers

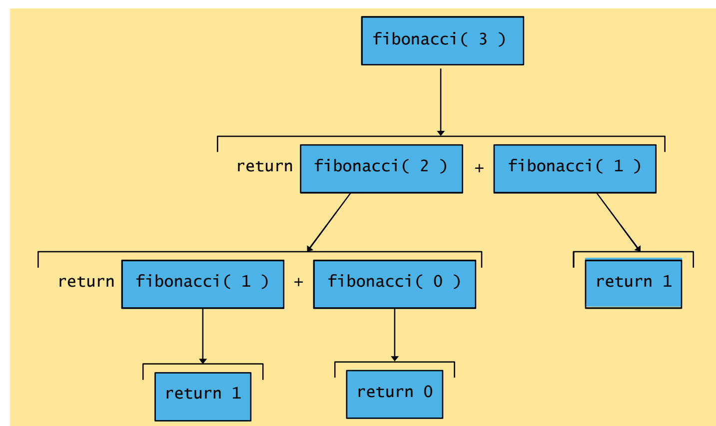
Fibonacci Code

```
public long fibonacci( long n )
    {
     // base case
     if ( n == 0 || n == 1 )
             return n;

     // recursive step
     else
     return fibonacci( n - 1 ) + fibonacci( n - 2 );

    } // end method fibonacci
```


Fibonacci Executed



Reading

Book Name: Object Oriented Programming in Java^{TM} - A Graphical Approach

Author: Kathryn E. Sanders & Andries van Dam

Content: Chapter # 12

Acknowledgements

- While preparing this course I have greatly benefited from the material developed by the following people:
  - Andy Van Dam (Brown University)
  - Mark Sheldon (Wellesley College)
  - Robert Sedgewick and Kevin Wayne (Princeton University)
  - Mark Guzdial and Barbara Ericsson (Georgia Tech)
  - Richard Halterman (Southern Adventist University)