Recursion II

Methods that call themselves

Reminder

- 1st Exam
  - Tomorrow
  - Will cover
    - Inheritance
    - Exceptions
  - 10-20 questions
  - Variety of question types
    - Short answer
    - Fill in the code
    - Step through the code
    - Perhaps some multiple choice

Recursive Methods

- A recursive method is one that can call itself

Non-recursive

```
methodA() {
…
   methodB ();
…
}
```

Recursive

```
methodB() {
…
   methodB ();
…
}
```

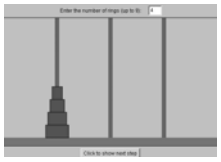
Components of a recursive methods

- Three necessary components for a recursive method:
  1. A test to stop or continue the recursion
  2. An end case that stops the recursion
  3. A recursive call that continues the recursion.

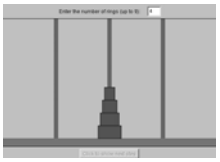
Towers of Hanoi

- 3 pegs and N disks (of different sizes)
- Starting with all N disks on one peg
  - Move all N disks to another peg
    - Can only move one peg at a time
    - You can never place a larger peg on top of a smaller peg.

Towers of Hanoi



Start



Goal

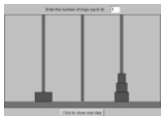
Towers of Hanoi

- Let's see the game in action
  - Two of many Tower of Hanoi applets on the Web
  - http://www.cut-the-knot.com/recurrence/hanoi.html
  - http://www.stlukes.new-canaan.ct.us/faculty/kress/hanoi.html

Towers of Hanoi

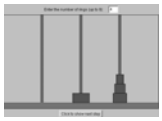
- Recursive solution
  - Forget for a moment that you can only move 1 disk at a time
    - Define a source, destination, and "spare" peg
    - Move the top N-1 disks from your source peg to your "spare" peg
    - Move the Nth disk from your source to your destination peg
    - Move the top N-1 disks from your spare to your destination

Towers of Hanoi



Step 1

Move N-1 disks from source to spare



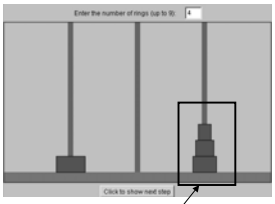
Step 2

Move Nth disk from source to destination

Step 3

Move N-1 disks from spare to destination

Towers of Hanoi



Note that this is just a smaller version of the Tower of Hanoi puzzle with N-1 disks (RECURSION!)

Towers of Hanoi

```
public void hanoi (int N,     // number disks
                   int src,   // source
                   int dest,  // destination
                   int spare ) // spare
{
     // Step 1: move N-1 disks to spare
     hanoi (N-1, src, spare, dest);

     // Step 2: move Nth disk to destination
     moveOne (src, dest);

     // Step 3:move N-1 disks from spare to destination
     hanoi (N-1, spare, dest, src)
}
```

Tower of Hanoi

- But can't recursion go on forever?

Towers of Hanoi

- The recursion stops when n = 1
  - Then there's only one disk to move,
  - So simply move it.

Tower of Hanoi

```
public void hanoi (int N,     // number disks
                   int src,   // source
                   int dest,  // destination
                   int spare ) // spare
{
     if ( N == 1 ) moveOne (src, dest)
     else {
         // Step 1: move N-1 disks to spare
         hanoi (N-1, src, spare, dest);

         // Step 2: move Nth disk to destination
         moveOne (src, dest);

         // Step 3: move N-1 disks from spare to destination
         hanoi (N-1, spare, dest, src)
     }
}
```

Tower of Hanoi

```
public void hanoi (int N,     // number disks
                   int src,   // source
                   int dest,  // destination
                   int spare ) // spare
{
     if ( N == 1 ) moveOne (src, dest)
     else {
         // move N-1 disks to spare
         hanoi (N-1, src, spare, dest);

         // move Nth disk to destination
         moveOne (src, dest);

         // move N-1 disks from spare to destination
         hanoi (N-1, spare, dest, src)
     }
}
```

Test

Stop

Continue

Towers of Hanoi

```
public class Hanoi {
    // some arrays that keep track of which
    // disks are on which pegs
   ...
   public Hanoi () {}
   public void hanoi (int n, int src, int dest, int
  spare) { ... }
   public void moveOne (int src, int dest) { ... }

   public static void main (String args[])
   {
      Hanoi H = new Hanoi();
      H.hanoi (nDisks, 0, 1, 2);
   }
}
```

Recursion and Stacks

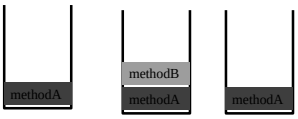
- When Java calls a method, it places info related to that method on a call stack
  - When a method is called, it's info is pushed onto the call stack
  - When a method is done, it's info is popped off the call stack
  - The top of the stack holds info for the current method being executed.

Recursion and Stacks

Non-recursive

```
methodA() {
…
   methodB ();
…
}

methodB () {
 …
}
```



Before call to MethodB

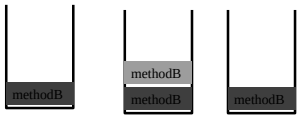
During call to MethodB

After call to MethodB

Recursion and Stacks

Recursive

```
methodB() {
…
   methodB ();
…
}
```

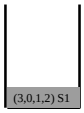


Before recursive call | During recursive call | After recursive call

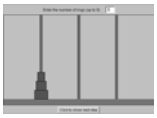
Back to Hanoi

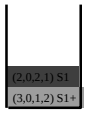
- Let's map out what happens when solving the Tower of Hanoi with 3 disks.
  - Disks will start on peg 0
  - Disks will end up on peg 1
  - Peg 3 will be our spare
  - hanoi (3, 0, 1,2)

Tower of Hanoi

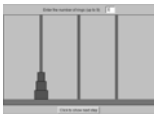


hanoi(3,0,1,2)
src: 0 dest:1 spare:2
Step 1: call hanoi(2,0,2,1)

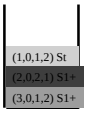




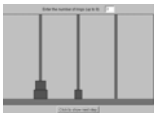
hanoi(2,0,2,1)
src: 0 dest:2 spare:1
Step 1: call hanoi(1,0,1,2)

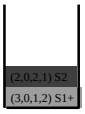


Tower of Hanoi

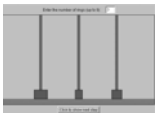


hanoi(1,0,1,2)
src: 0 dest:1 spare:2
Stop: move(0,1)

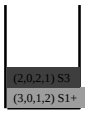




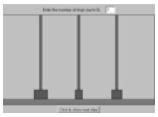
hanoi(2,0,2,1)
src: 0 dest:2 spare:1
Step 2: move (0,2)

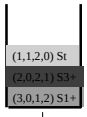


Tower of Hanoi

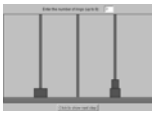


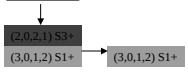
hanoi(2,0,2,1)
src: 0 dest:2 spare:1
Step 3: hanoi(1,1,2,0)



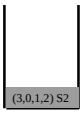


hanoi(1,1,2,0)
src:1 dest:2 spare:0
Stop: move(1,2)

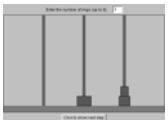


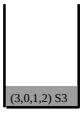


Tower of Hanoi

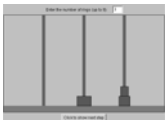


hanoi(3,0,1,2)
src: 0 dest:1 spare:2
Step 2: move(0,1)

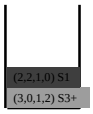




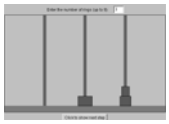
hanoi(3,0,1,2)
src: 0 dest:1 spare:2
Step 3: hanoi(2,1,0,2)

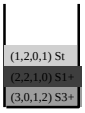


Tower of Hanoi

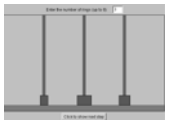


hanoi(2,2,1,0)
src: 2 dest:1 spare:0
Step 1: hanoi(1,2,0,1)

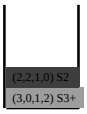




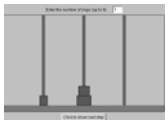
hanoi(1,2,0,1)
src: 2 dest:0 spare:1
Stop: move(2,0)

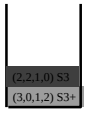


Tower of Hanoi

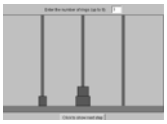


hanoi(2,2,1,0)
src: 2 dest:1 spare:0
Step 2: move(2,1)

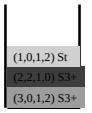




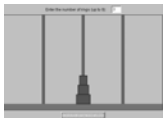
hanoi(2,2,1,0)
src: 2 dest:1 spare:0
Step 3: hanoi(1,0,1,2)

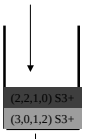


Tower of Hanoi

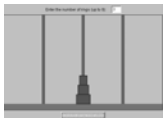


hanoi(1,0,1,2)
src: 0 dest:1 spare:2
Stop: move(0,1)





We are done!



→empty

Tower of Hanoi

- Things to note
  - Recursive functions can be `void`
  - Recursive functions can all operate on the same underlying data structure.
  - Iterative solution to Tower of Hanoi problem would be difficult to describe.

Tower of Hanoi

- Questions?

Summary

- Recursion – When a method calls itself
- Components of a recursive method
  - Test
  - Stop
  - Continue
- Recursion, Method calls, and stacks
- Tower of Hanoi.

Next time

- Exam tomorrow
- Questions?