Teaching Programming, Not Programs

Computer Science, Cornell

Teaching Programming, Not Programs

Computer Science, Cornell

Teaching Programming, Not Programs
Computer Science, Cornell

Past 5-6 years:

- Teach computing using Java
- 130-200 students each semester
- Mostly Engineering, but also Arts & Sciences, Human Ecol, Arch, Agr & Life Sciences
- 1/2 of them have never programmed before
- 2 50-minute lectures per week
- 1 50-minute closed, required lab per week

Past history of research and education on the formal development of programs.
1970’s, 1980’s 1990’s ...

Starting my 88th semester of teaching

Past history of research and education on the formal development of programs.
1970's, 1980's 1990's ...

Teach program design, program methodology: strategies, principles for developing programs

Starting my 88th semester of teaching

No experience teaching secondary school

Past history of research and education on the formal development of programs.
1970's, 1980's 1990's ...

Teach program design, program methodology: strategies, principles for developing programs

Starting my 88th semester of teaching

What I propose to do

Make this talk interesting whether you are teaching Java or C++ (ugh!) or Python or Ruby or whatever.

What I propose to do

Make this talk interesting whether you are teaching Java or C++ (ugh!) or Python or Ruby or whatever.

1. Discuss some pedagogical principles

2. Talk about how I teach programming (not programs)
   Get across concepts simply
   Teach methodology

3. Discuss other aspects —recursion before loops, use of loop invariants, ...

Goal: Reveal Programming Process and Teach Skills

Michael Caspersen discusses this in his PhD thesis, done at Aarhus, June 2007. Based on cognitive science, educational psychology, cognitive skill acquisition, research in programming methodology.

Mathias Felleissen has had great success using his “design recipe” and TeachScheme!. Has reached out to secondary education.

Goal: Reveal Programming Process and Teach Skills

Michael Caspersen discusses this in his PhD thesis, done at Aarhus, June 2007. Based on cognitive science, educational psychology, cognitive skill acquisition, research in programming methodology.

Mathias Felleissen has had great success using his “design recipe” and TeachScheme!. Has reached out to secondary education.

identify forms of data
write examples of these forms
identify the desired black-box behavior
write examples (test cases) of behavior
derive template for program from data
use template to complete program logic

Present material at the appropriate level of abstraction

Need a good model of the variable.
Need a good model of execution of proc/function calls.
Need a good model of classes and objects.

None of these should be in terms of
the computer and memory.

Present concepts at appropriate level of abstraction

The computer itself is not the right level of abstraction for beginners. Give them a model they can understand without mentioning computer and memory.

Present concepts at appropriate level of abstraction

The computer itself is not the right level of abstraction for beginners. Give them a model they can understand without mentioning computer and memory.

~~The computer must always know the type of value to be stored in the memory location associated with a variable.~~

~~An object reference variable actually stores the address where the object is stored in memory.~~

~~An object has its own unique identity, which distinguishes it from all other objects in the computer's memory …. An object's identity is handled behind the scenes by the Java virtual machine and should not be confused with the variables that might refer to that object.~~

(1) gives impression that only computers can execute programs.
(2) confuses people who have little idea of memory, virtual machines, and how computers work.

Present concepts at the appropriate level of abstraction

Algol 60 language definition does not mention the computer.

Present concepts at the appropriate level of abstraction

Algol 60 language definition does not mention the computer.

“The purpose of the algorithmic language is to describe computational processes. …

A variable is a designation given to a single value.

Assignment statements serve for assigning the value of an expression to a variable …. The process will … be understood to take place in three steps as follows:

4.2.3.1. Any subscript expressions occurring in the left part variable are evaluated in sequence from left to right.

4.2.3.2. The expression of the statement is evaluated.

4.2.3.3. The value of the expression is assigned to the left part variable, with any subscript expressions having values as evaluated in step 4.2.3.1.

Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

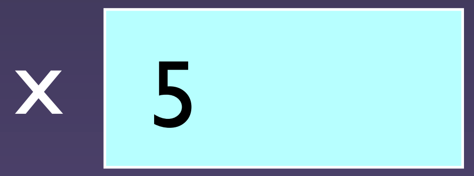
Computer: wrong level of abstraction for beginners. Give them a model they can understand —don’t mention computer and memory.

Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

Computer: wrong level of abstraction for beginners. Give them a model they can understand —don't mention computer and memory.

Problem: students don't know how to execute the assignment statement.

```
x=  x+2;
```



Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

Computer: wrong level of abstraction for beginners. Give them a model they can understand —don't mention computer and memory.

Problem: students don't know how to execute the assignment statement.

Variable:

1. A name associated with a value.
2. A named box with a value inside it.

x= x+2;

$x \ \boxed{5}$

Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

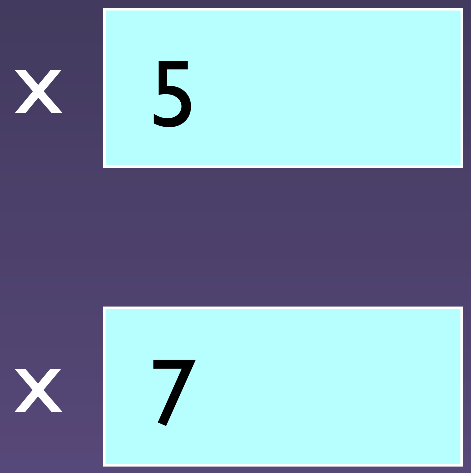
Computer: wrong level of abstraction for beginners. Give them a model they can understand —don't mention computer and memory.

Problem: students don't know how to execute the assignment statement.

Variable:

1. A name associated with a value.
2. A named box with a value inside it.

```
x=  x+2;
```



Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

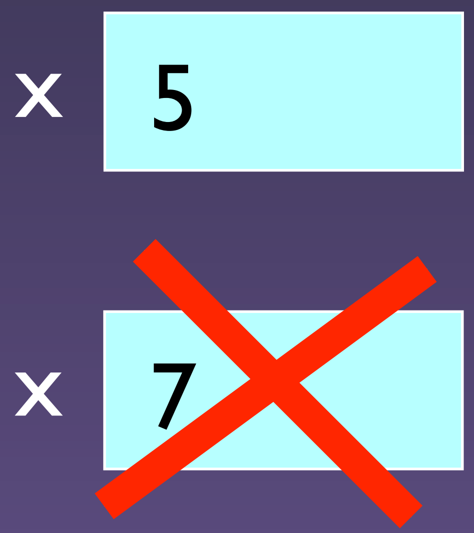
Computer: wrong level of abstraction for beginners. Give them a model they can understand —don't mention computer and memory.

Problem: students don't know how to execute the assignment statement.

Variable:

1. A name associated with a value.
2. A named box with a value inside it.

```
x=  x+2;
```



Don't draw variable again!

Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

Computer: wrong level of abstraction for beginners. Give them a model they can understand —don't mention computer and memory.

Problem: students don't know how to execute the assignment statement.

Variable:

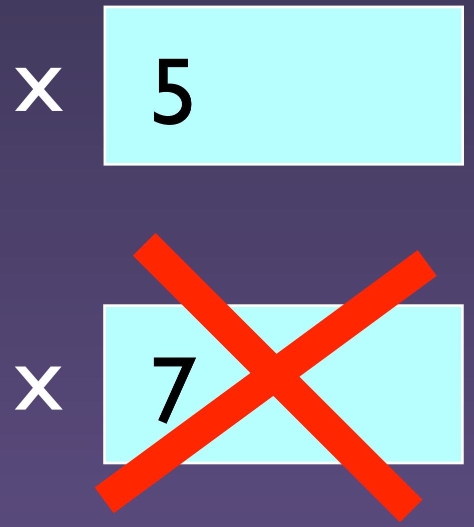
1. A name associated with a value.
2. A named box with a value inside it.

x= x+2;

To execute the assignment:

(1) evaluate the expression and

(2) store its value in the variable.



Don't draw variable again!

Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

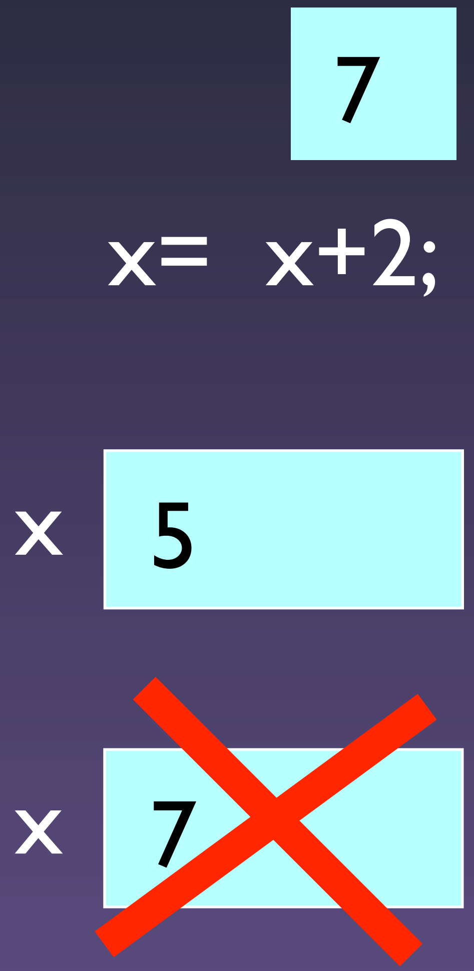
Computer: wrong level of abstraction for beginners. Give them a model they can understand —don't mention computer and memory.

Problem: students don't know how to execute the assignment statement.

Variable:

1. A name associated with a value.
2. A named box with a value inside it.

To execute the assignment:
(1) evaluate the expression and
(2) store its value in the variable.



Don't draw variable again!

Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

Computer: wrong level of abstraction for beginners. Give them a model they can understand —don't mention computer and memory.

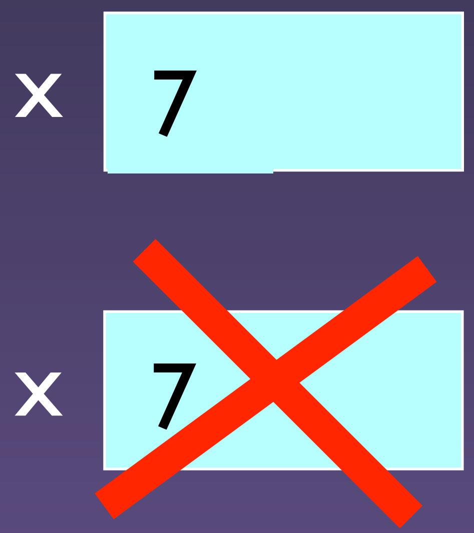
Problem: students don't know how to execute the assignment statement.

Variable:

1. A name associated with a value.
2. A named box with a value inside it.

```
x=  x+2;
```

To execute the assignment:
(1) evaluate the expression and
(2) store its value in the variable.



Don't draw variable again!

Present concepts at the appropriate level of abstraction —and provide precise, clear, definitions

Computer: wrong level of abstraction for beginners. Give them a model they can understand —don't mention computer and memory.

To execute the assignment:
(1) evaluate the expression and
(2) store its value in the variable.

To execute procedure call p(args)
(1) draw a frame for the call
(2) assign arg values to pars
(3) execute method body
(4) erase the frame for the call

To evaluate new C(args)
(1) create an object of class C
(2) execute constructor call C(args)
(3) yield as the value of the expression the "name" of the new object

Name the things you want to talk about

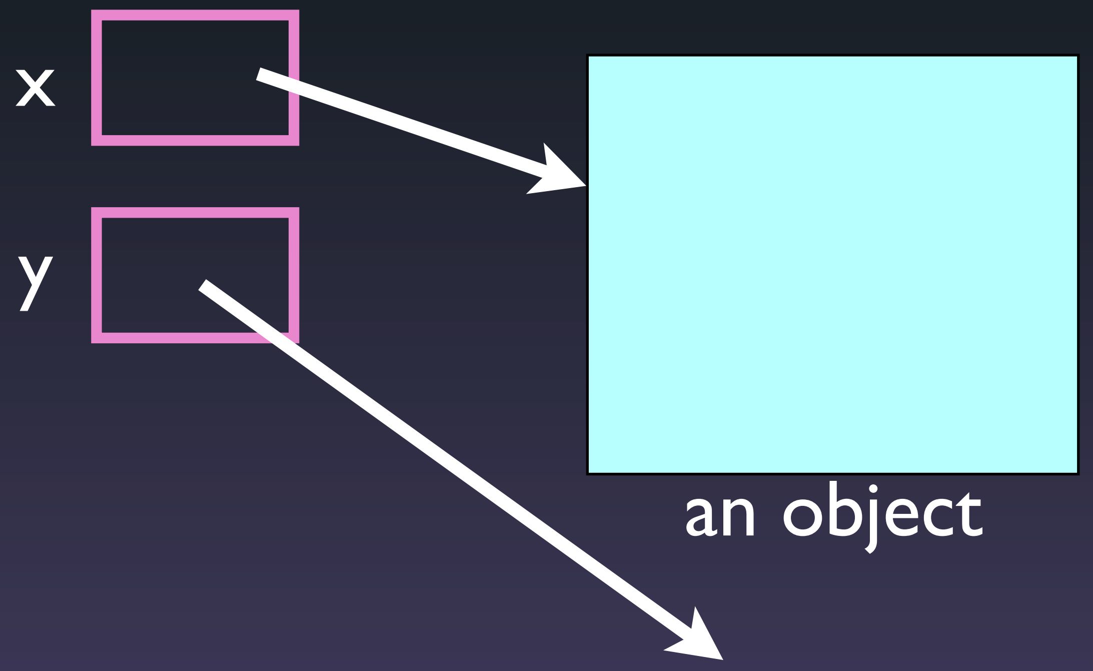
Name the things you want to talk about

Pointer-reference

Name the things you want to talk about

Pointer-reference

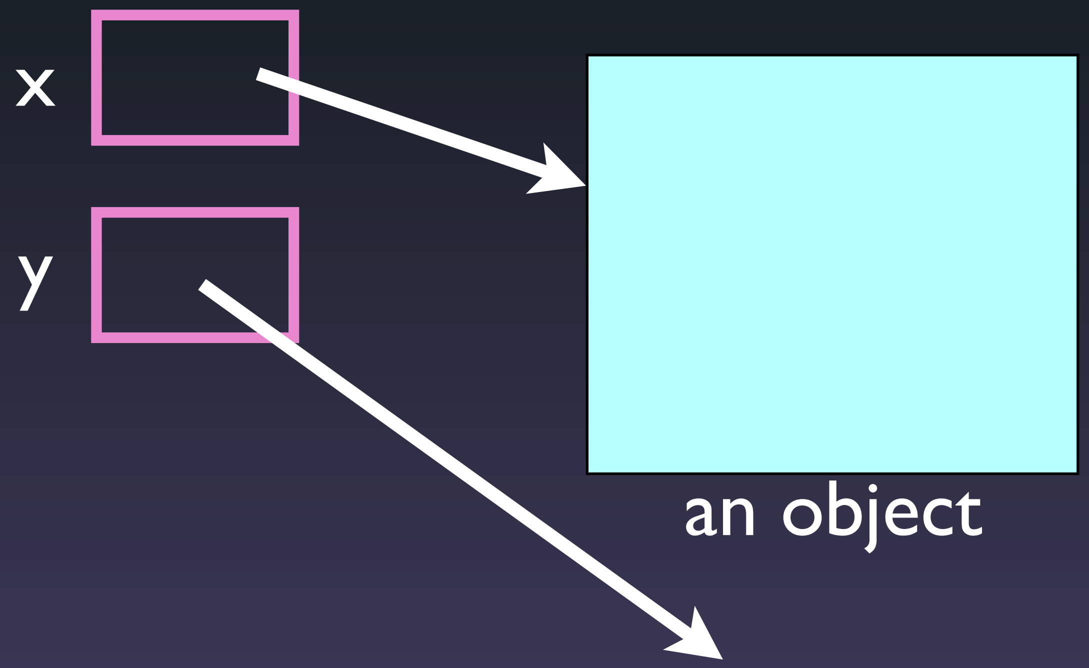
```
y=  x;
```



Name the things you want to talk about

~~Pointer-reference~~

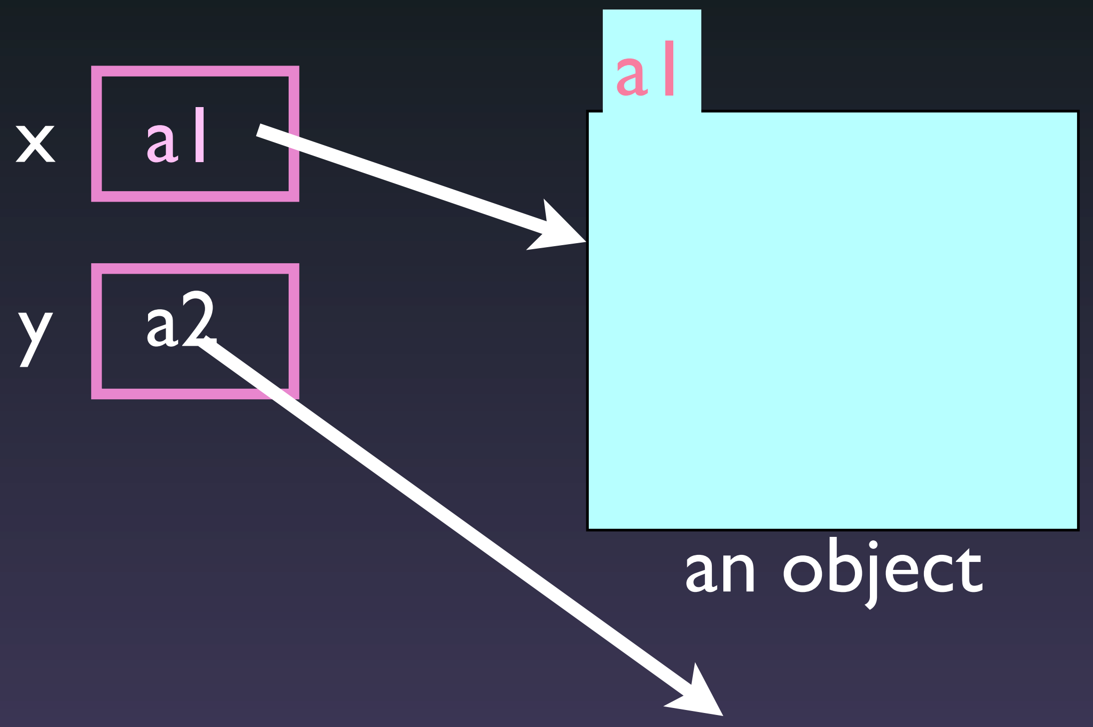
y= x;



Name the things you want to talk about

~~Pointer-reference~~

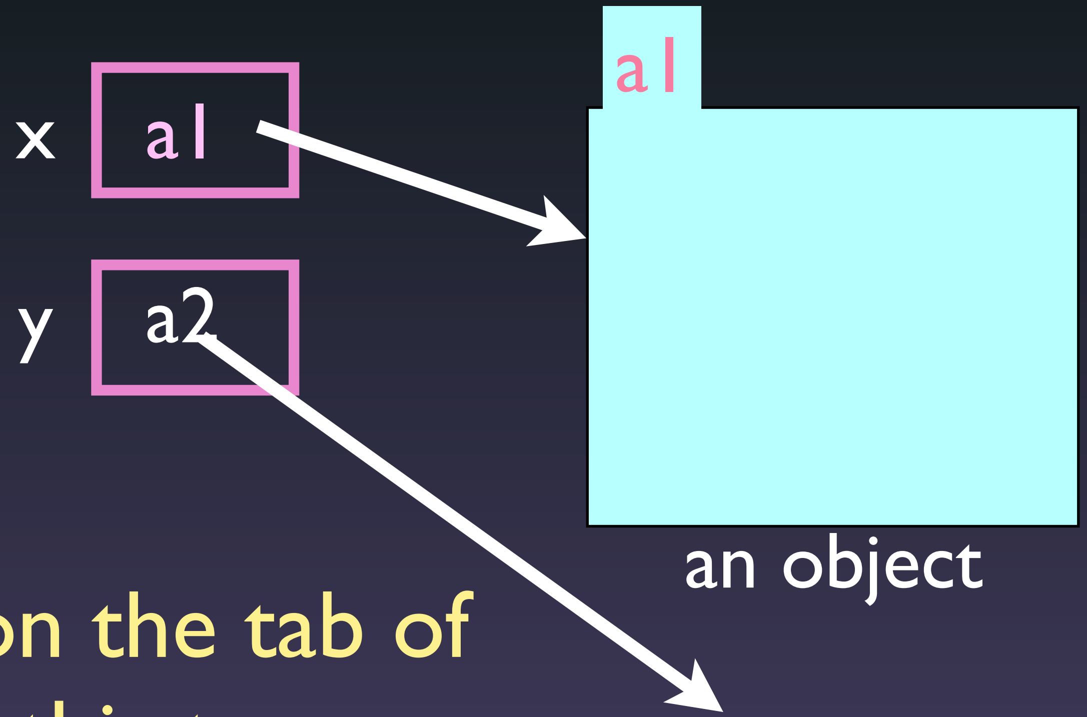
```
y=  x;
```



Name the things you want to talk about

~~Pointer-reference~~

y= x;



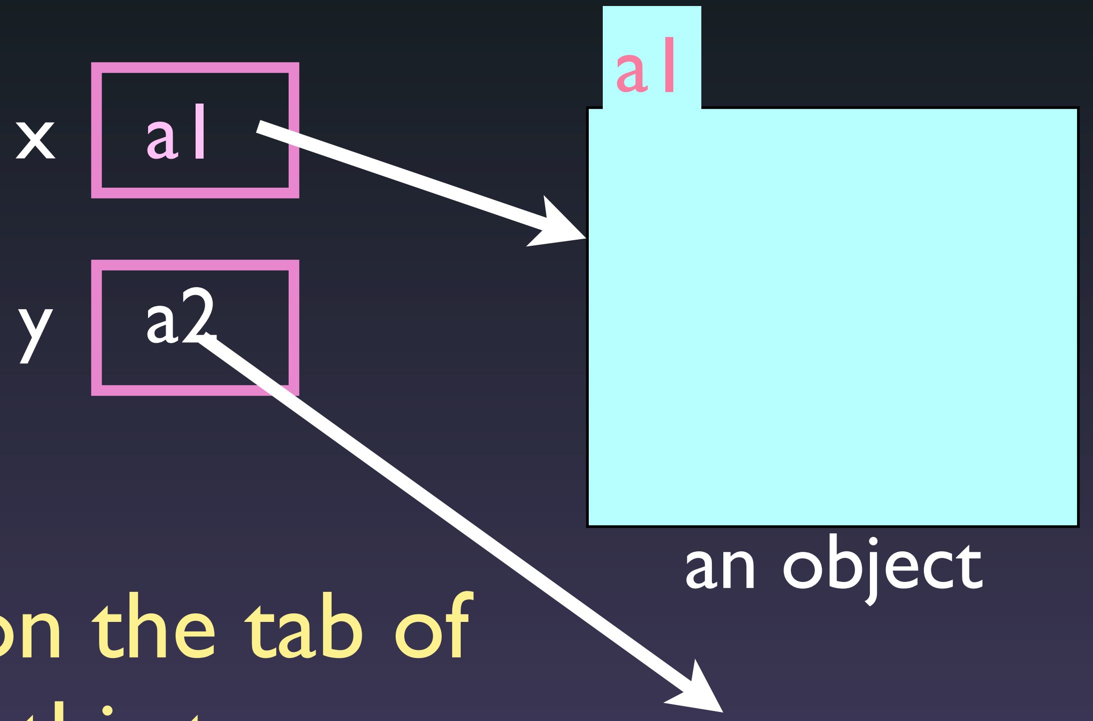
The names that can be placed on the tab of an object form a type. Values of this type can be placed in a variable of the associated class.

Name the things you want to talk about

~~Pointer-reference~~

y= x;

Expression x evaluates to a1



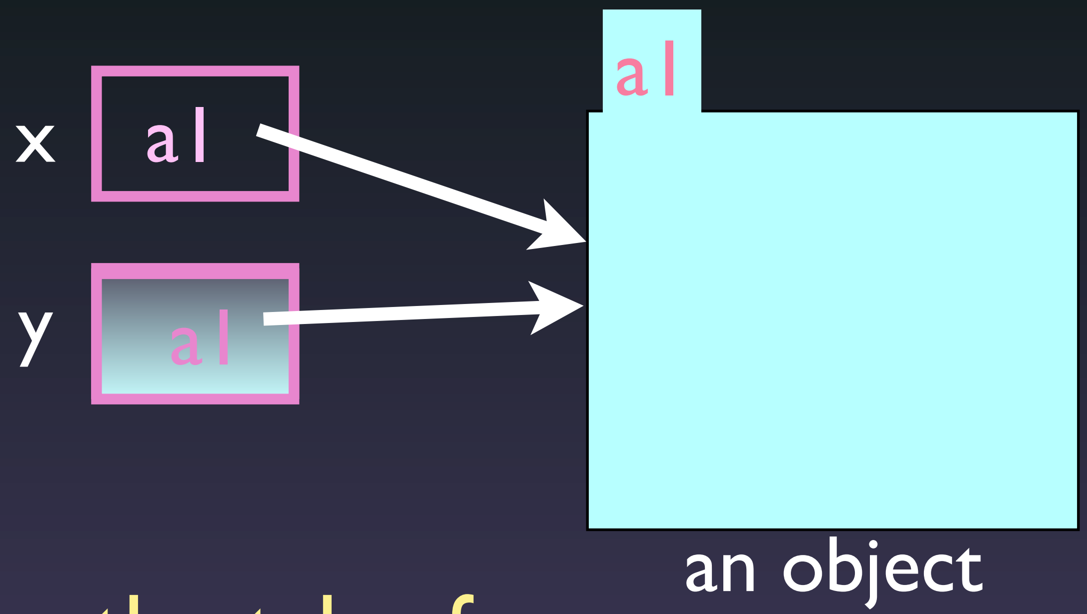
The names that can be placed on the tab of an object form a type. Values of this type can be placed in a variable of the associated class.

Name the things you want to talk about

~~Pointer-reference~~

y= x;

Expression x evaluates to a1



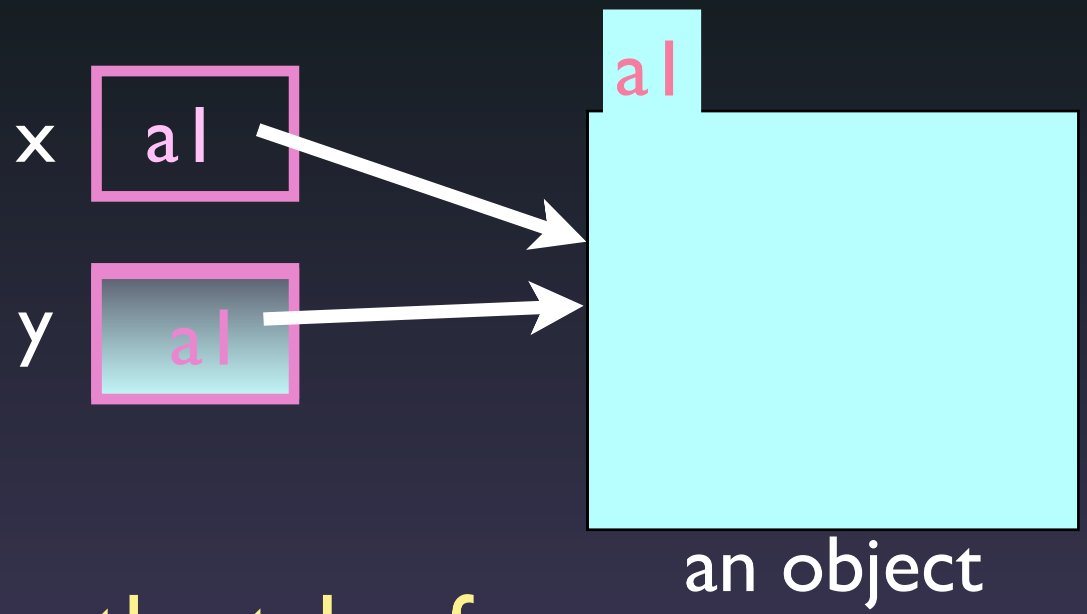
The names that can be placed on the tab of an object form a type. Values of this type can be placed in a variable of the associated class.

Name the things you want to talk about

~~Pointer-reference~~

y= x;

Expression x evaluates to a1



The names that can be placed on the tab of an object form a type. Values of this type can be placed in a variable of the associated class.

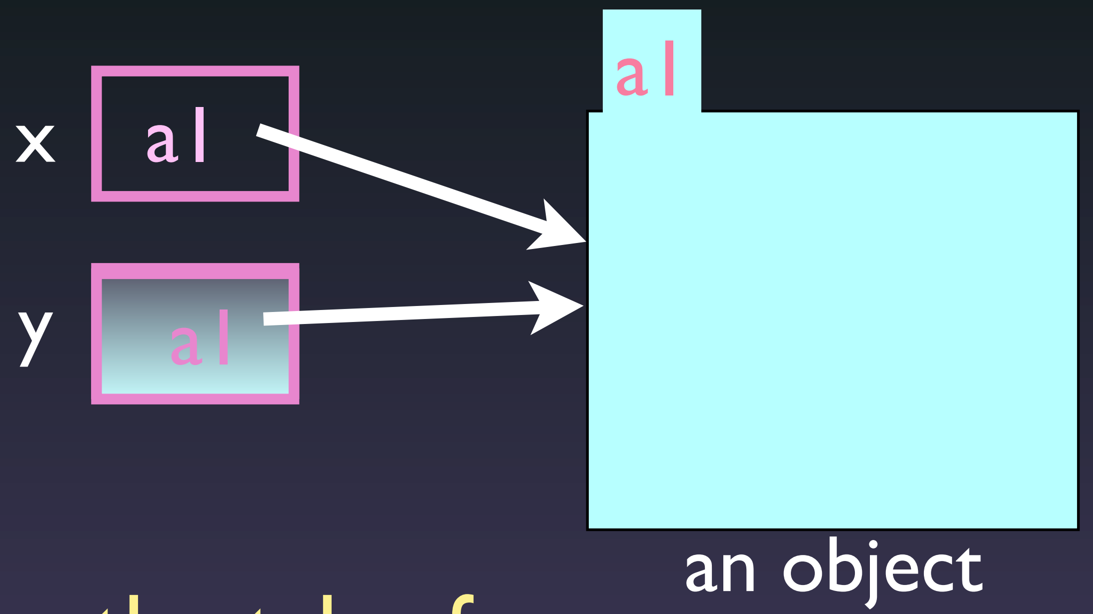
An object has its own unique identity, which distinguishes it from all other objects in the computer's memory …. An object's identity is handled behind the scenes by the Java virtual machine and should not be confused with the variables that might refer to that object.

Name the things you want to talk about

~~Pointer-reference~~

y= x;

Expression x evaluates to al



The names that can be placed on the tab of an object form a type. Values of this type can be placed in a variable of the associated class.

~~An object has its own unique identity, which distinguishes it from all other objects in the computer's memory …. An object's identity is handled behind the scenes by the Java virtual machine and should not be confused with the variables that might refer to that object.~~

Order material to minimize introduction of terms/topics without explanation —as much as possible, define a term when first used

Order material to minimize introduction of terms/topics without explanation —as much as possible, define a term when first used

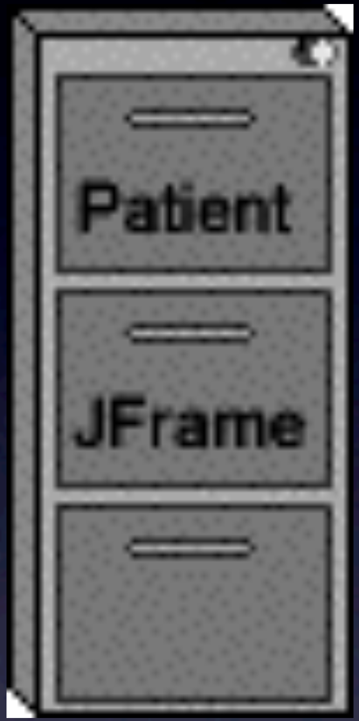
```
/** Print "Hello World" */
public class FirstClass {
    public static void main(String[] pars) {
        System.out.println("Hello World);
    }
}
```

Almost every line of a Java program deals with a class or object!

So the language Java dictates an OO-first approach to teaching programming

Model for objects and classes

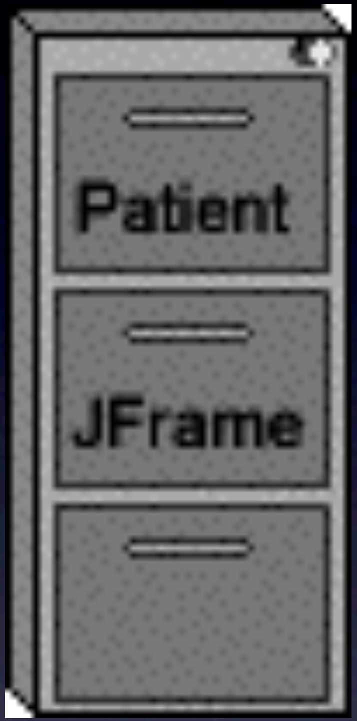
A class is a file drawer. All the manila folders in it have the same kind of information

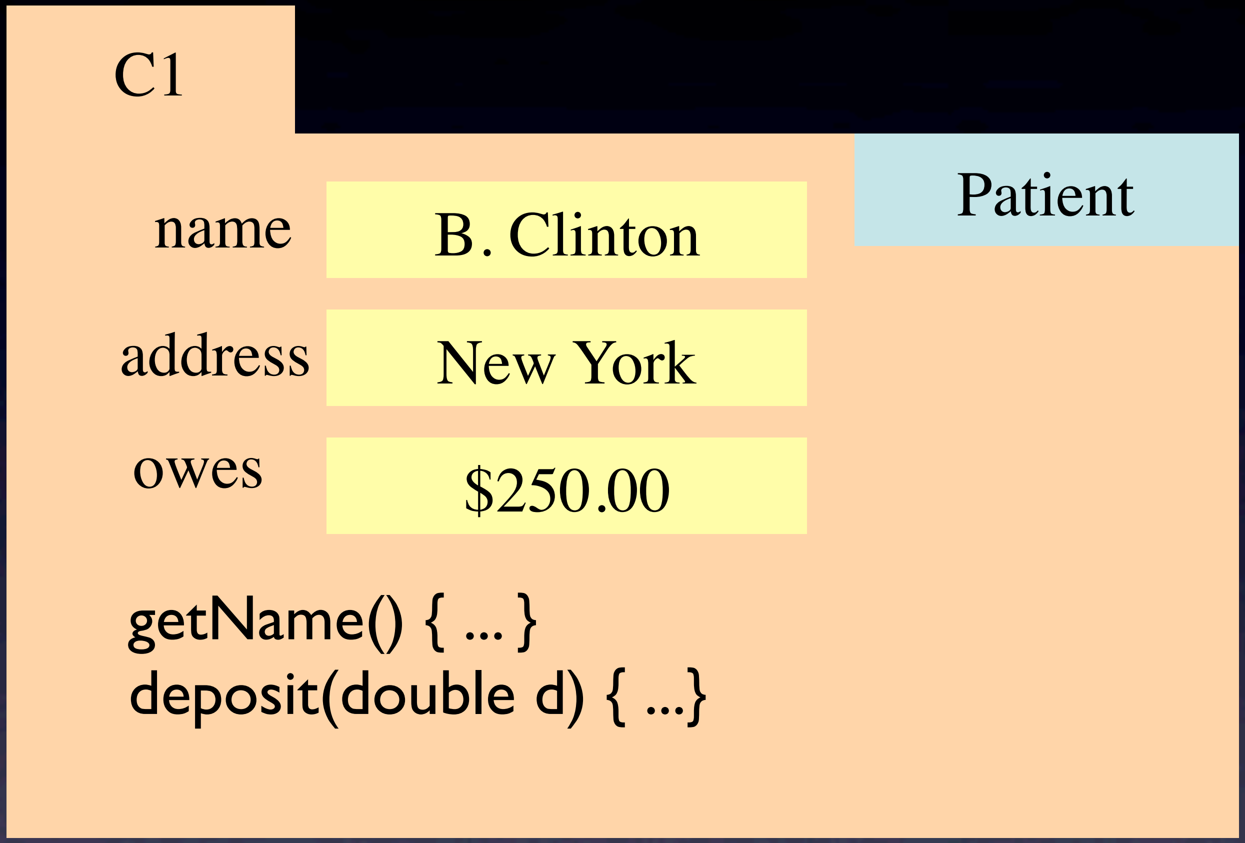


Manilla folder: an object or instance of the class

Model for objects and classes

A class is a file drawer. All the manila folders in it have the same kind of information

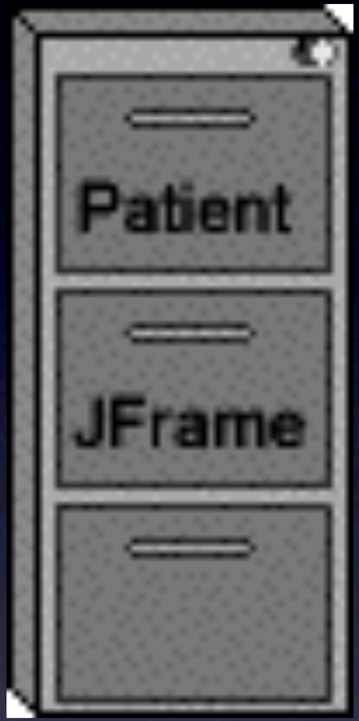




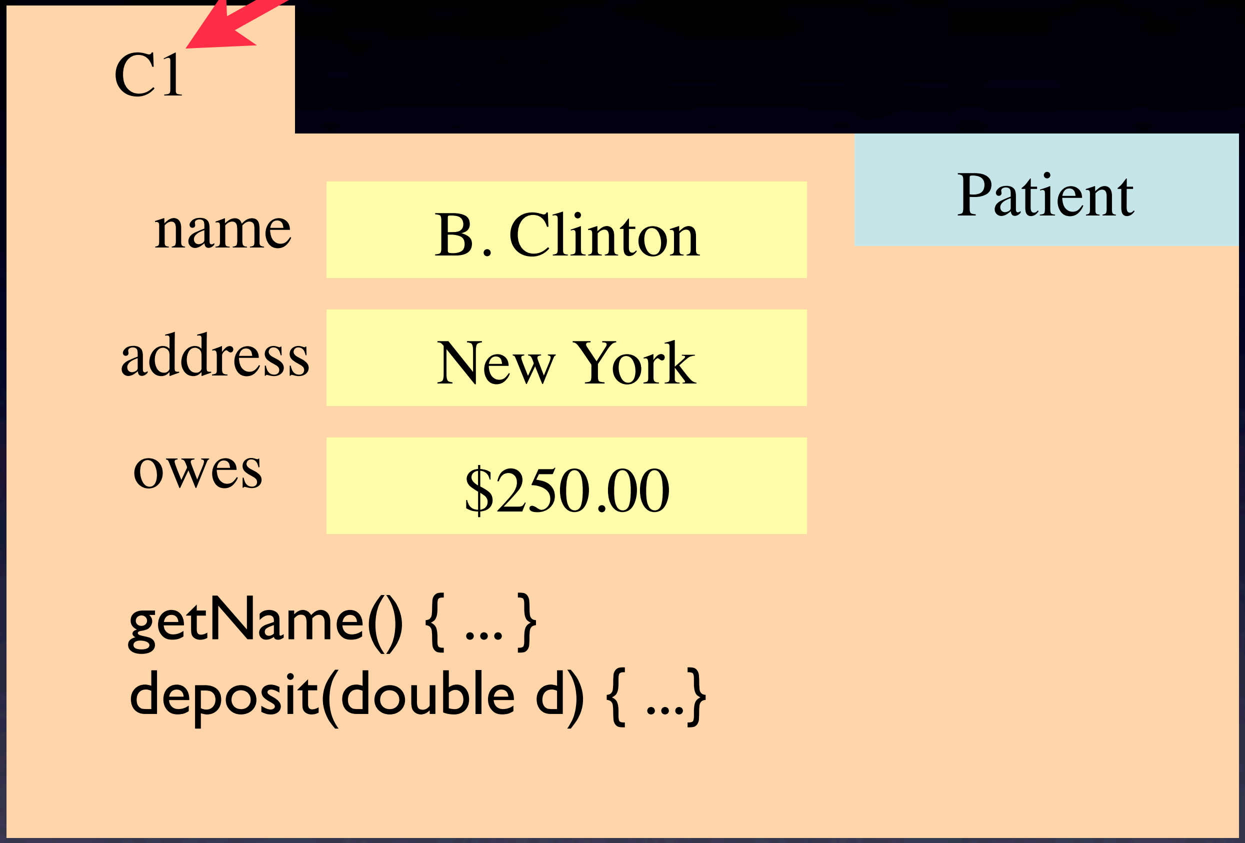
Manilla folder: an object or instance of the class

Model for objects and classes

A class is a file drawer. All the manila folders in it have the same kind of information



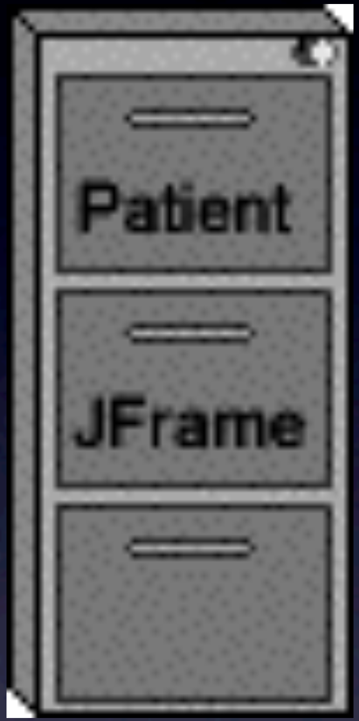
Whoever creates the folder gets to choose the name on the tab. Must be unique.



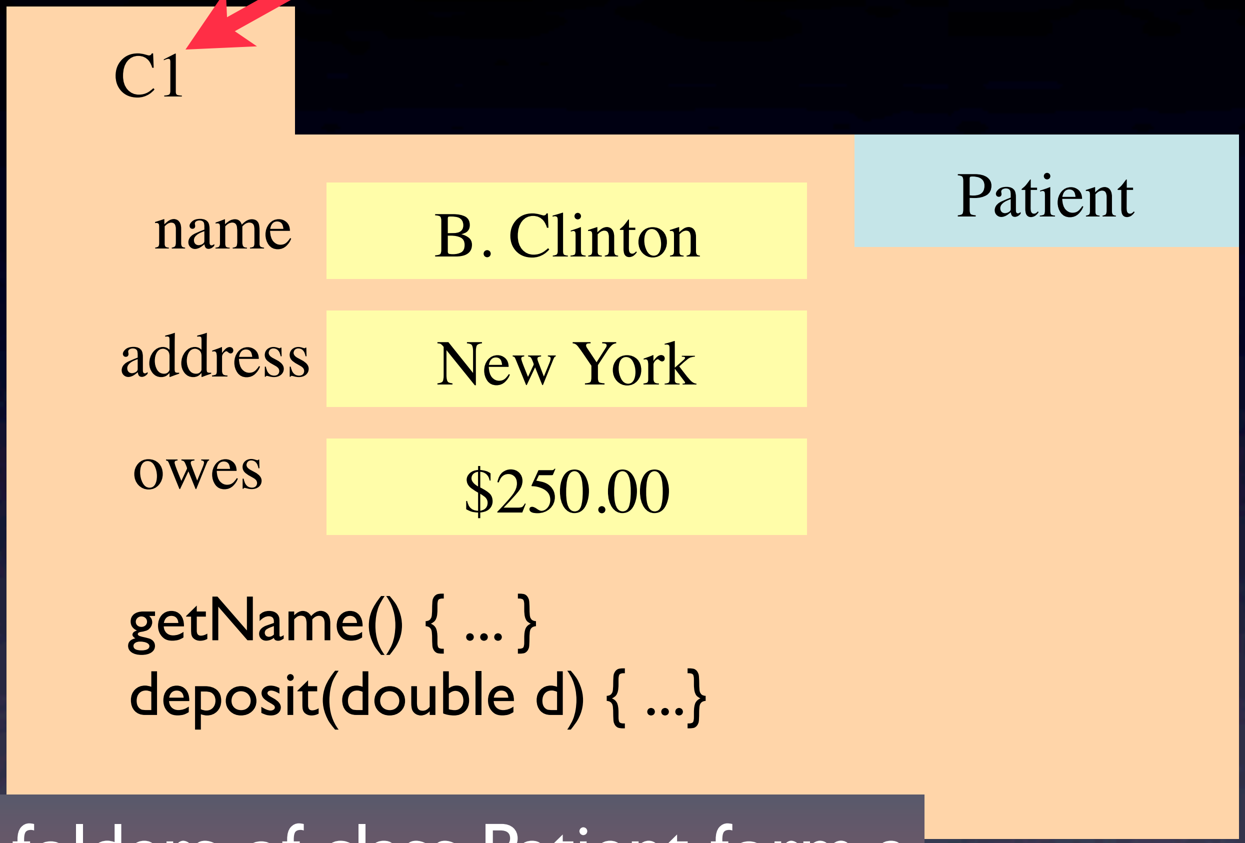
Manilla folder: an object or instance of the class

Model for objects and classes

A class is a file drawer. All the manila folders in it have the same kind of information



Whoever creates the folder gets to choose the name on the tab. Must be unique.



The names on folders of class Patient form a type of value.
Importance cannot be overestimated.
Allows us to eliminate terms like pointer and reference and provide a single consistent view of assignment.

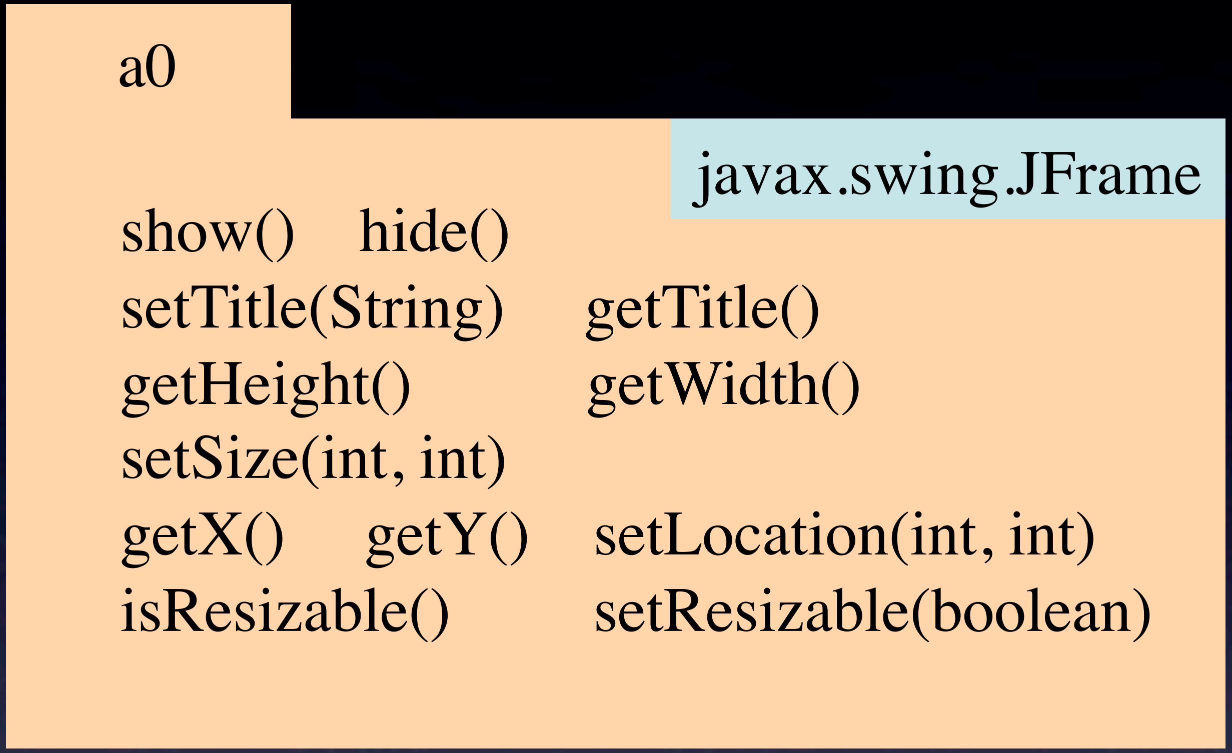
Manilla folder: an object or instance of the class

Model for objects and classes

First class definition: subclass of JFrame.

Reasons:

1. Inheritance comes naturally, right in the beginning.
2. Never have to show something that can't be explained.
3. See right away how things are reused.


a0
javax.swing.JFrame
show() hide()
setTitle(String) getTitle()
getHeight() getWidth()
setSize(int, int)
getX() getY() setLocation(int, int)
isResizable() setResizable(boolean)

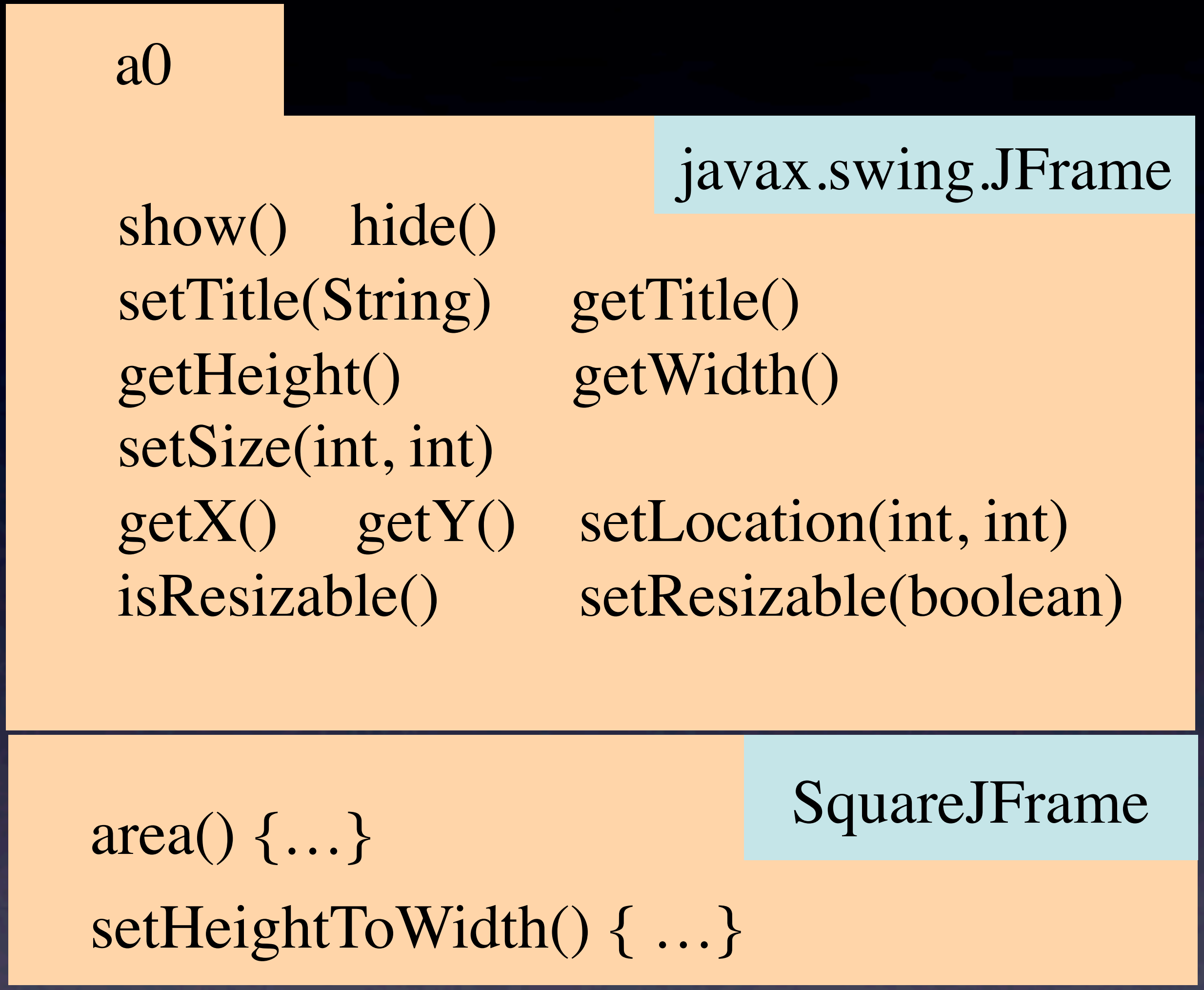
Manilla folder: an object or instance of the class

Model for objects and classes

First class definition: subclass of JFrame.

Reasons:

1. Inheritance comes naturally, right in the beginning.
2. Never have to show something that can't be explained.
3. See right away how things are reused.



Manilla folder: an object or instance of the class

Model for objects and classes

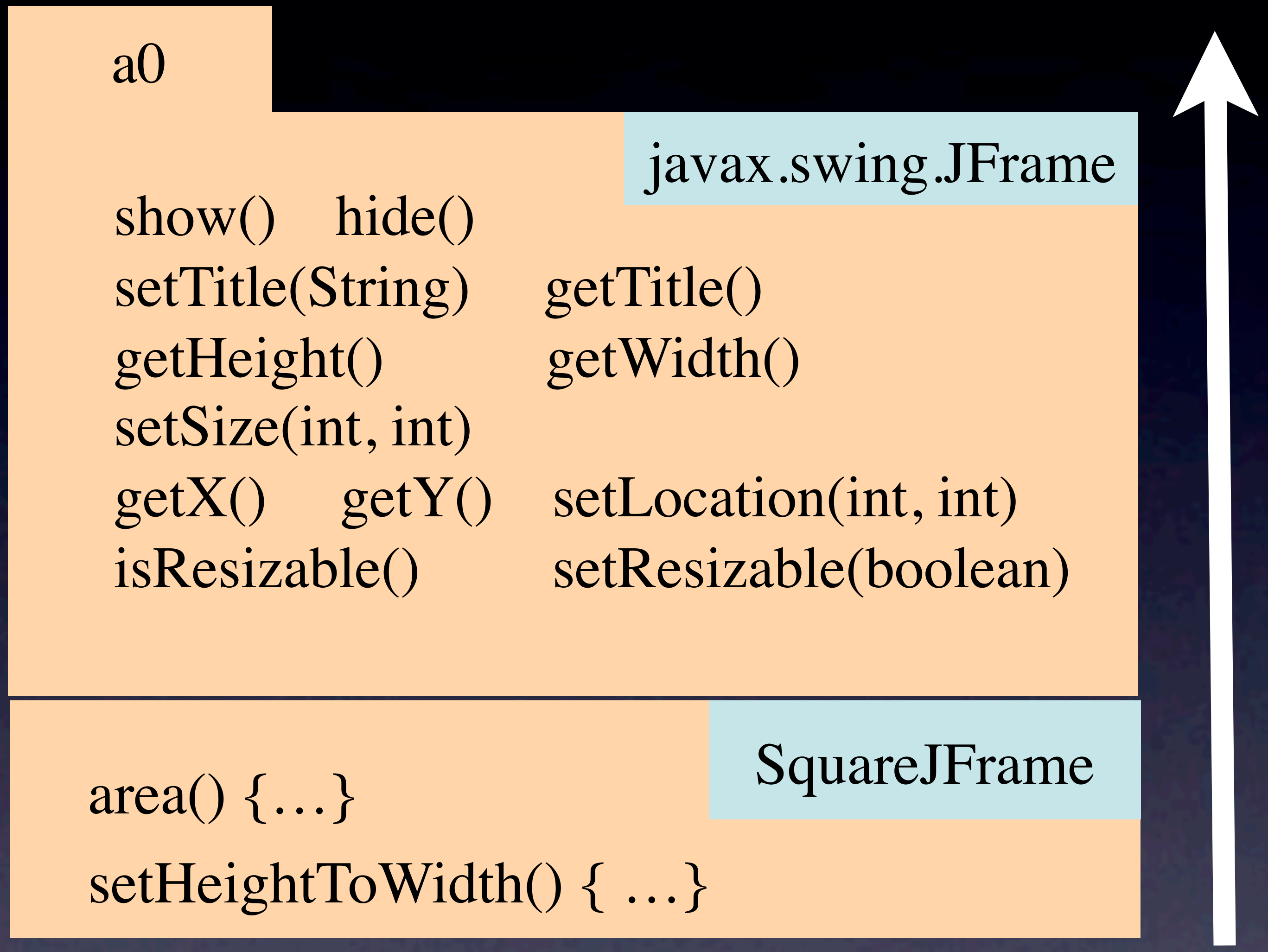
First class definition: subclass of JFrame.

Reasons:

1. Inheritance comes naturally, right in the beginning.

2. Never have to show something that can't be explained.

3. See right away how things are reused.



Bottom-up rule says to search for a component from bottom up. Gets overriding method, naturally.

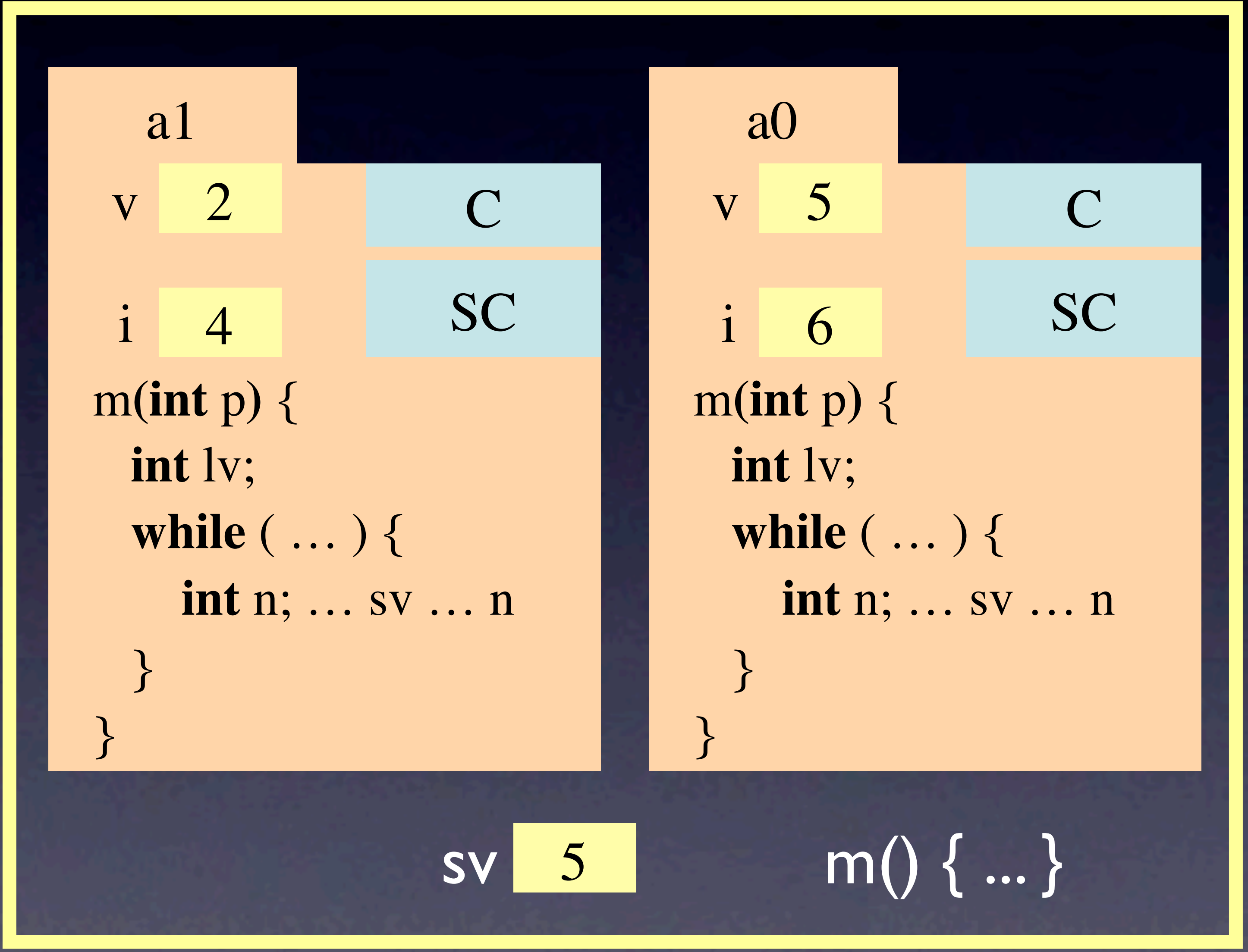
Manilla folder: an object or instance of the class

The model for objects/classes allows simple explanation of language concepts

Inside-out rule

To determine the declaration to which a variable name refers, look in the current construct, then the surrounding one, then the surrounding one, etc., until it is found.

Similar rule for method calls



SC's file drawer

Inside-out rule is used, with minor differences, in most languages, including predicate logic

Have first assignment require mastery:

Use a submission-feedback loop until everything is right

Allow you to establish some important ground rules with code that is short and straightforward. No one gets penalized for misunderstanding.

1. Beginning programmers: not penalized for being confused.
2. “Experienced” programmers: not penalized for their bad habits.

- Precise, clear, complete specs on procedures/functions/methods
- Precise, clear, complete class invariant
- Appropriate test cases
- Proper indentation
- Correct program

Executing method calls

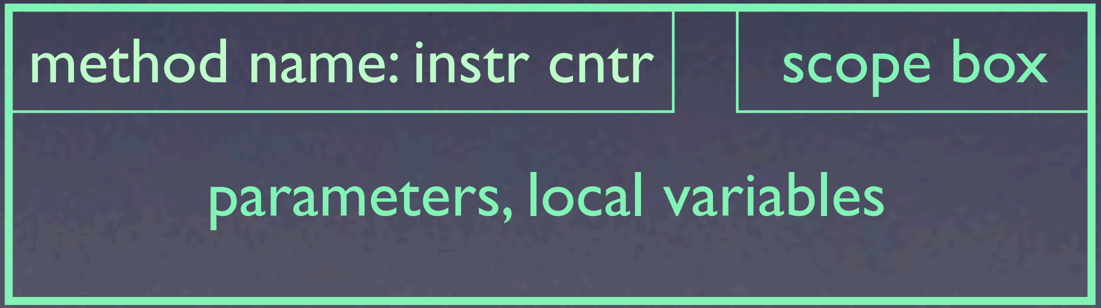
Force students to learn to execute a method call by hand

- Will give them a concrete understanding that they can't get otherwise.
- Later, it will be easy to see that recursion actually works.

Executing method calls

Force students to learn to execute a method call by hand

- Will give them a concrete understanding that they can't get otherwise.
- Later, it will be easy to see that recursion actually works.



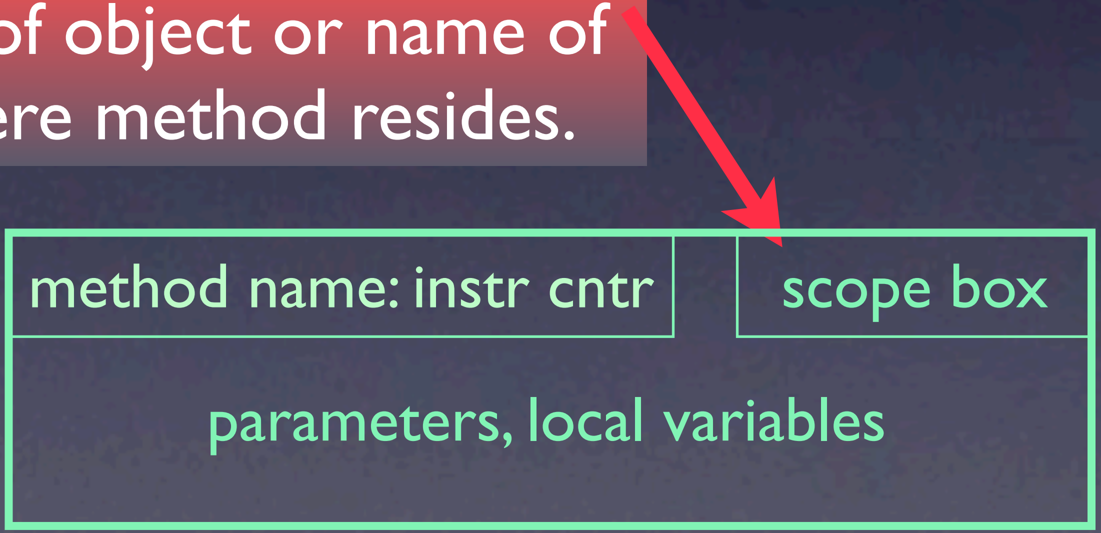
template for frame for a call

Executing method calls

Force students to learn to execute a method call by hand

- Will give them a concrete understanding that they can't get otherwise.
- Later, it will be easy to see that recursion actually works.

Contains name of object or name of file drawer where method resides.



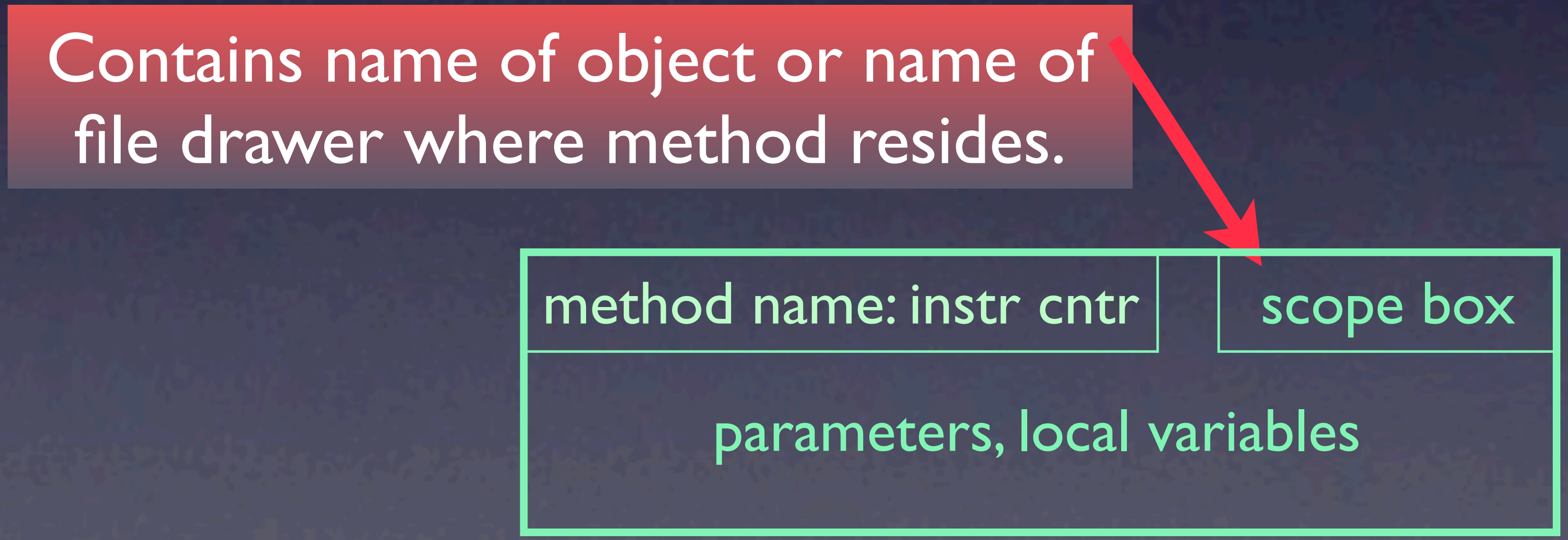
template for frame for a call

Executing method calls

Force students to learn to execute a method call by hand

- Will give them a concrete understanding that they can't get otherwise.
- Later, it will be easy to see that recursion actually works.

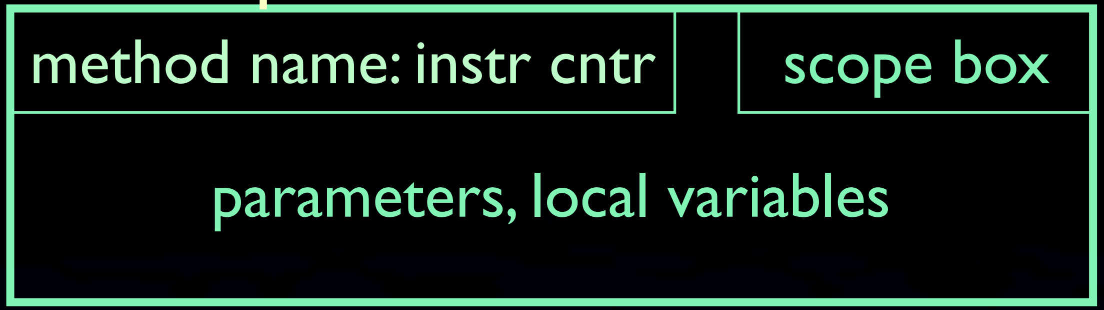
When you first introduce the topic, don't **you** do the execution. Show the students the template for a frame for a call, ask them to get out a piece of paper, and have them do it, step by step, in groups of two.


Contains name of object or name of file drawer where method resides.
method name: instr cntr
scope box
parameters, local variables

template for frame for a call

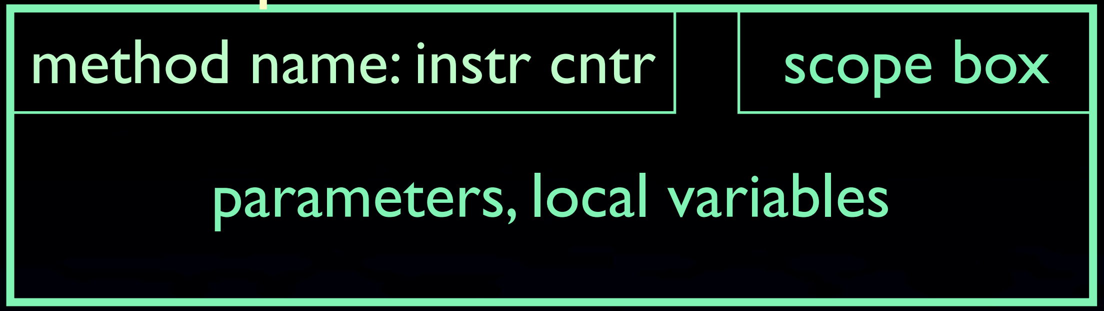
Executing method calls

template for frame for a call



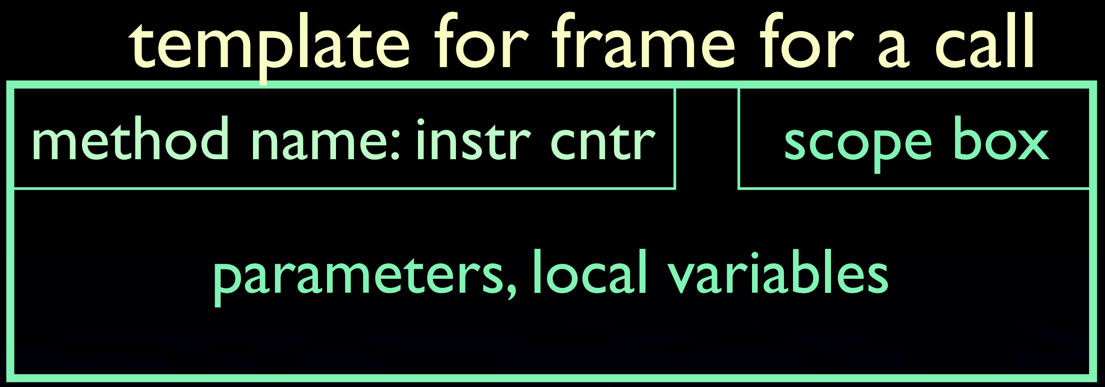
Executing method calls

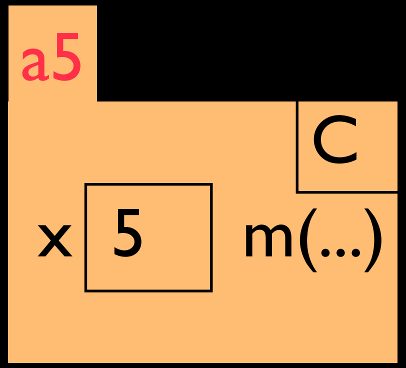
template for frame for a call



```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

Executing method calls

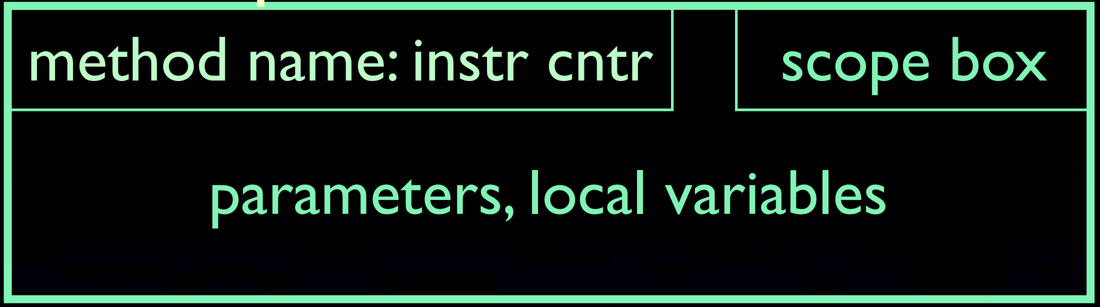




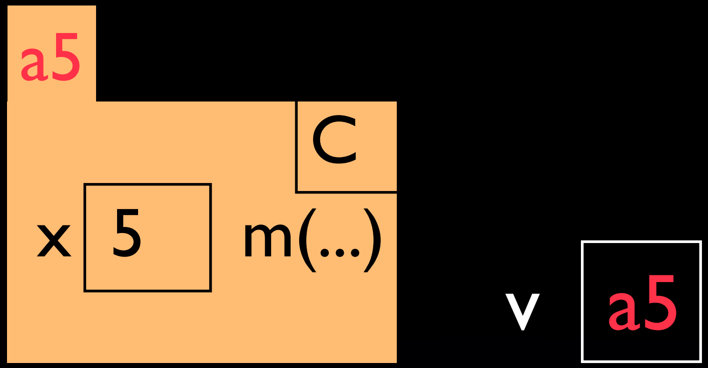
```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

Executing method calls

template for frame for a call

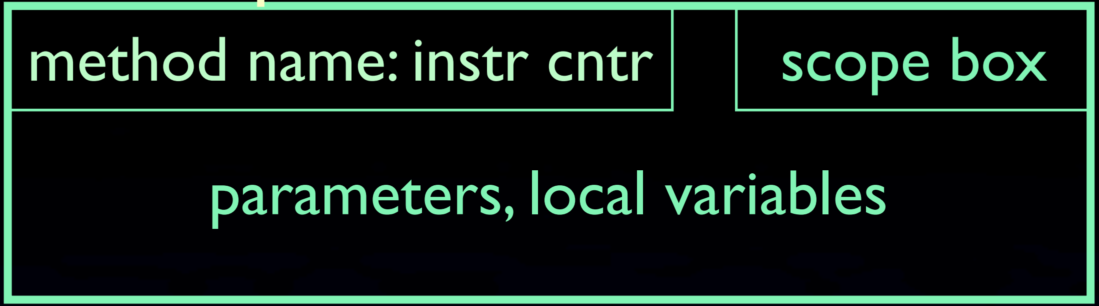


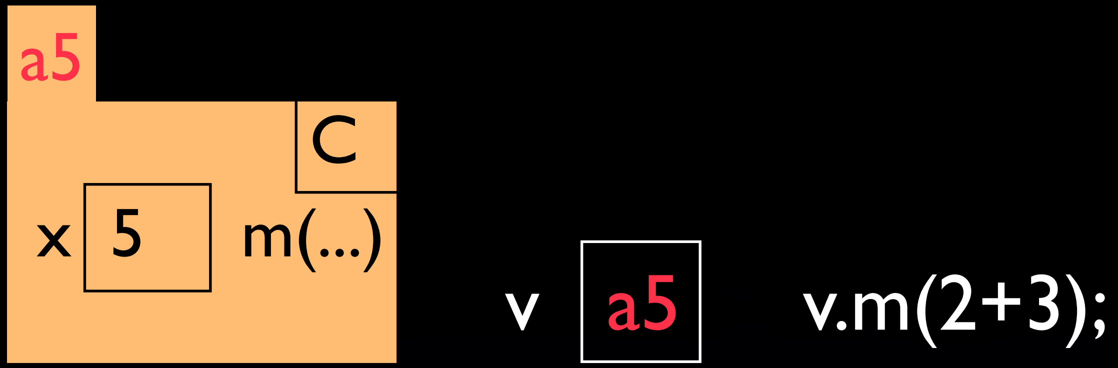
```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```



Executing method calls

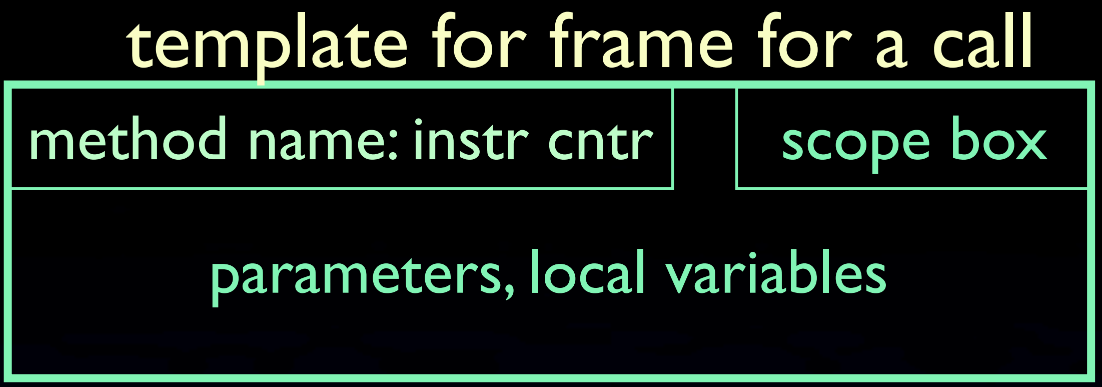
template for frame for a call

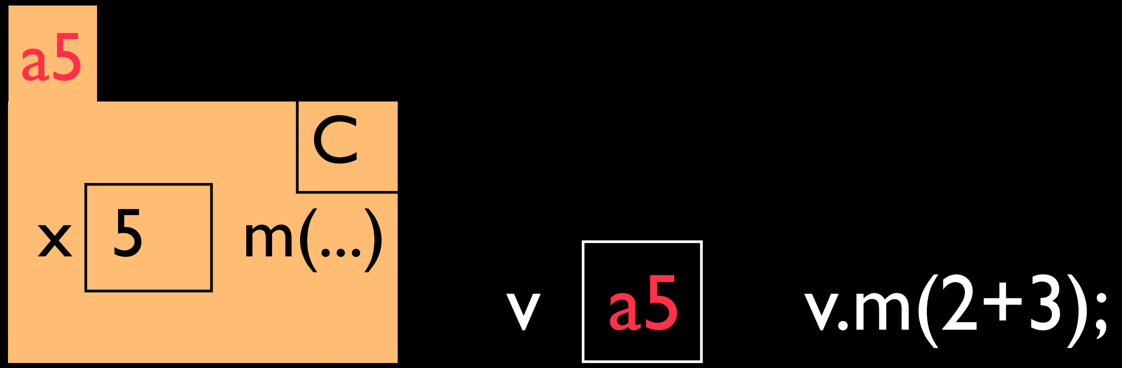




```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

Executing method calls

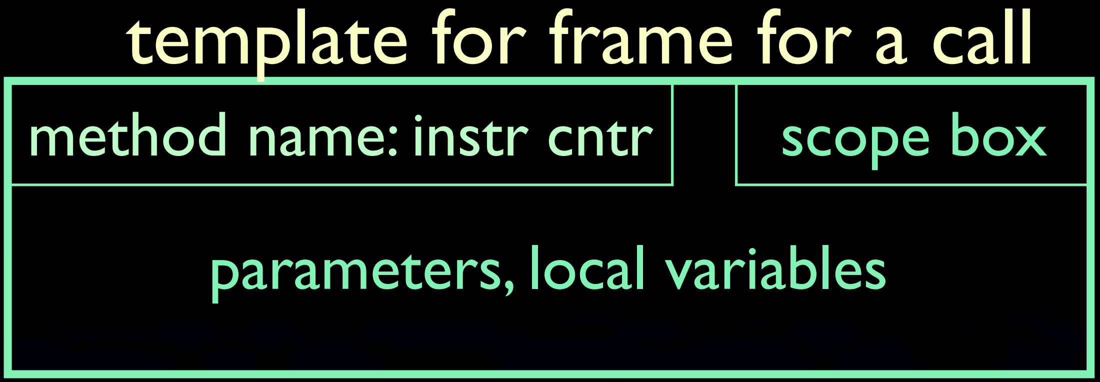


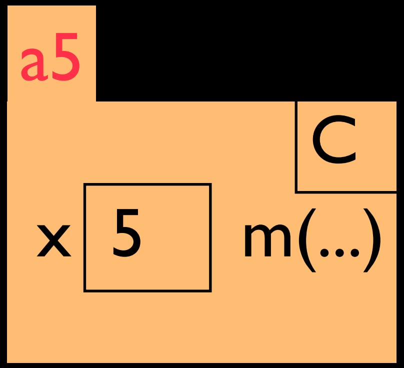


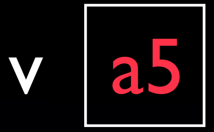
```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

1. Draw frame for call

Executing method calls



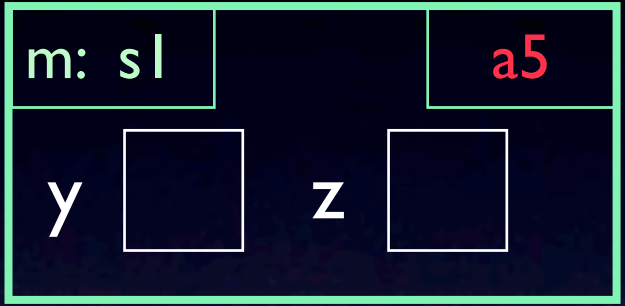




v.m(2+3);

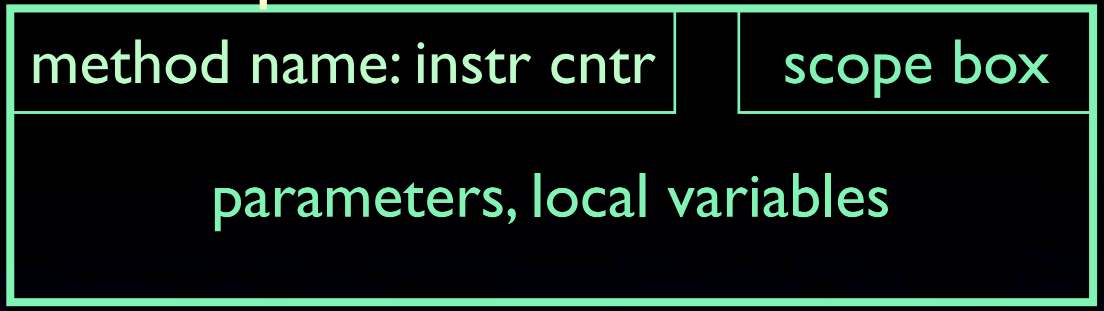
```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

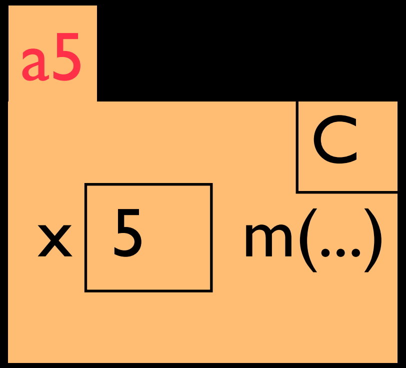
1. Draw frame for call

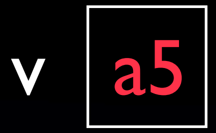


Executing method calls

template for frame for a call





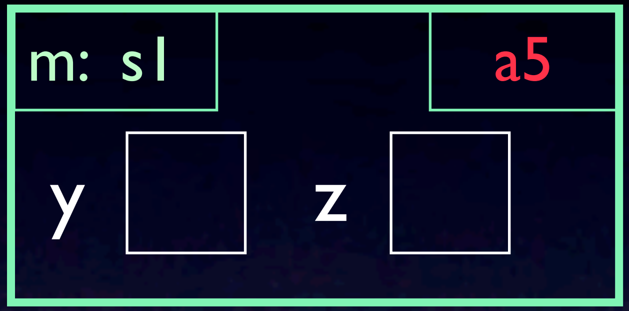


v.m(2+3);

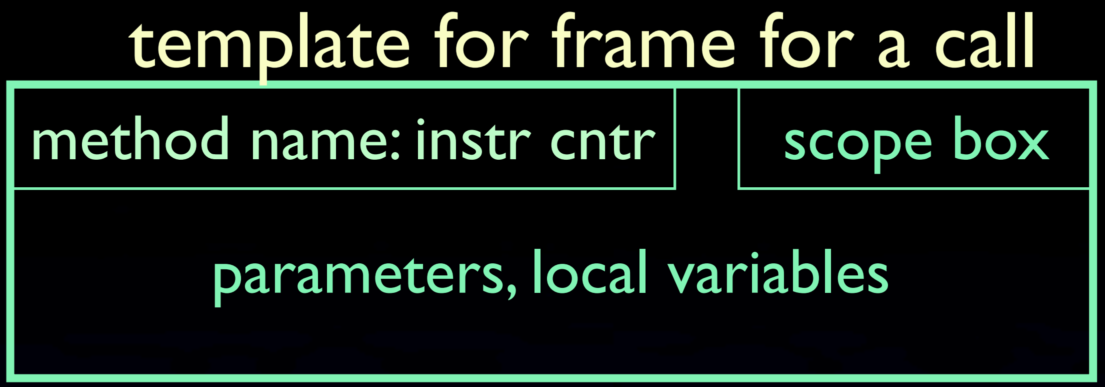
```
public class C {
   private int x;
   public void m(int y) {
      s1: if (y != 0) {
            s2: int z= y+1;
            s3: x= z;
      }
   }
}
```

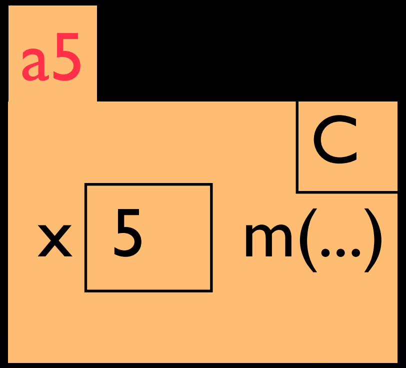
1. Draw frame for call

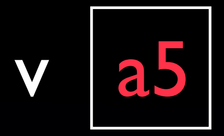
2. Store arg values in pars



Executing method calls



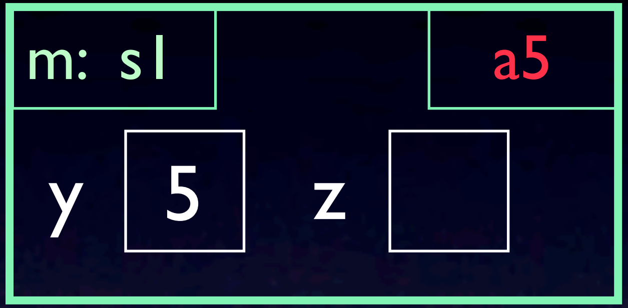




v.m(2+3);

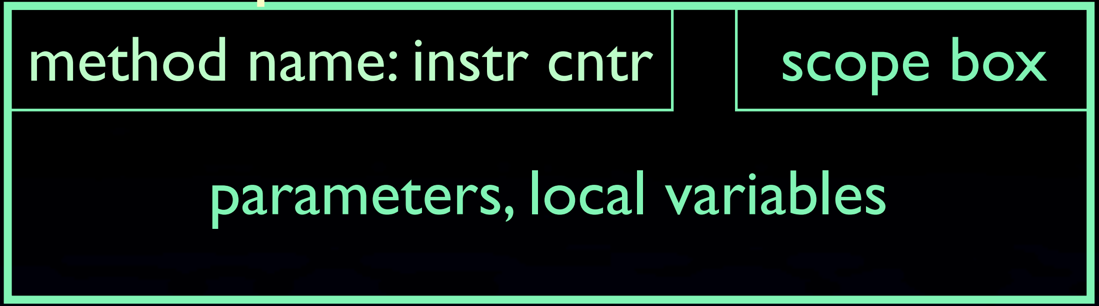
```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

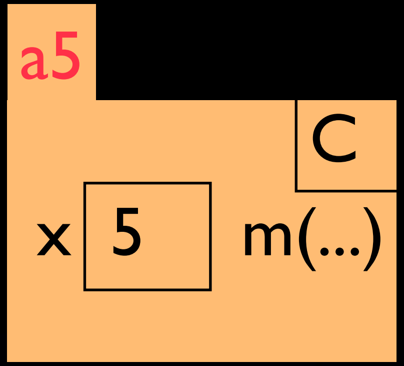
1. Draw frame for call
2. Store arg values in pars

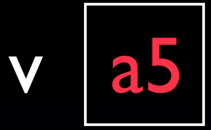


Executing method calls

template for frame for a call





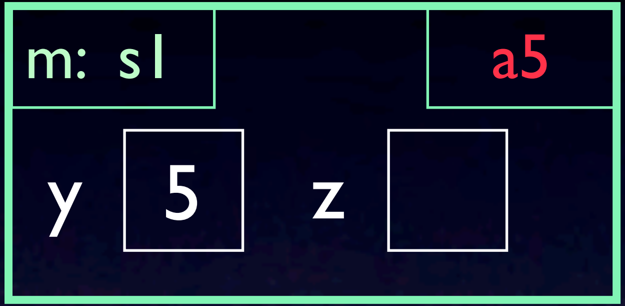


v.m(2+3);

```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

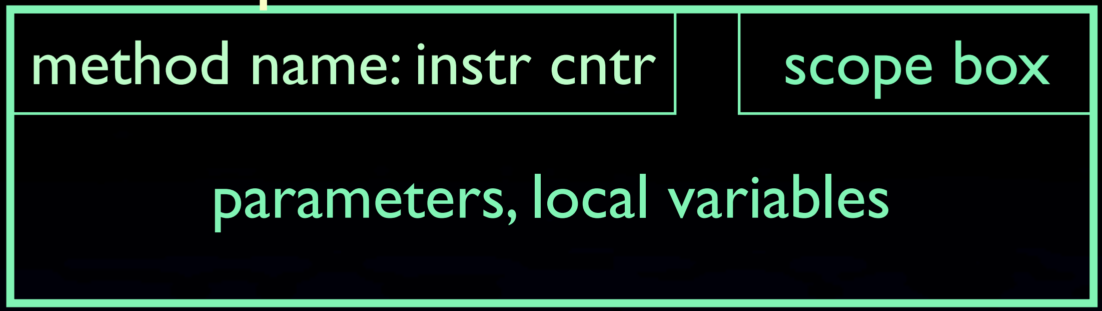
1. Draw frame for call
2. Store arg values in pars
3. Execute method body.

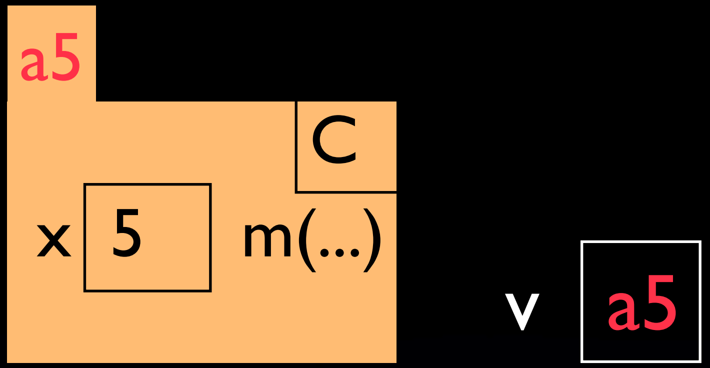
Look in frame for call for names; if not there, use scope box



Executing method calls

template for frame for a call



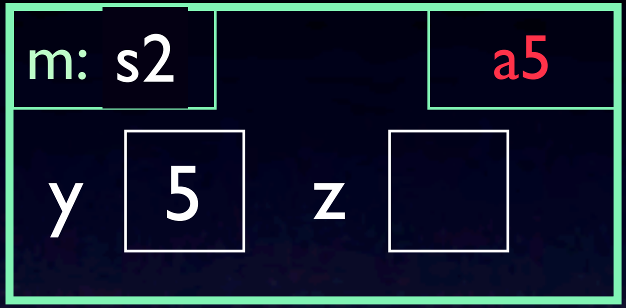


v.m(2+3);

```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

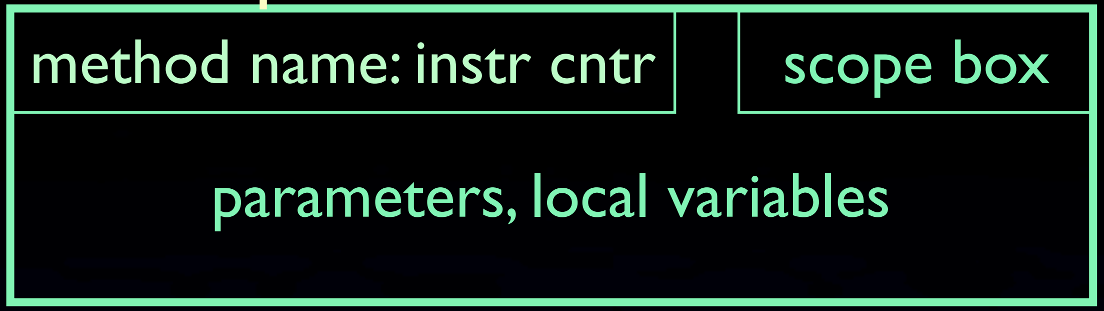
1. Draw frame for call
2. Store arg values in pars
3. Execute method body.

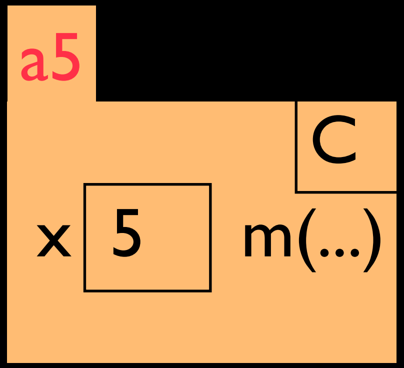
Look in frame for call for names; if not there, use scope box

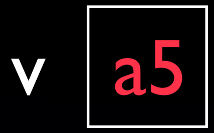


Executing method calls

template for frame for a call





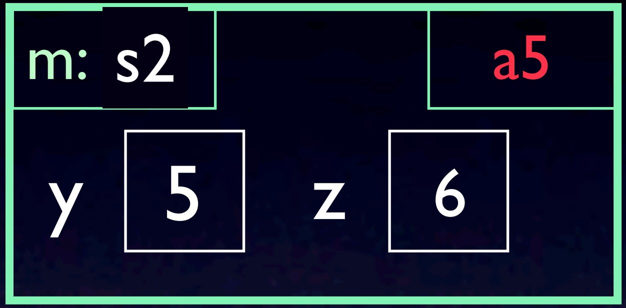


v.m(2+3);

```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

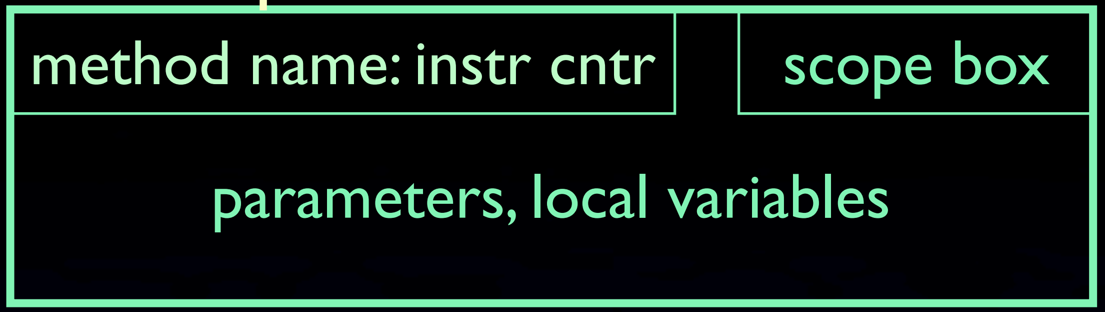
1. Draw frame for call
2. Store arg values in pars
3. Execute method body.

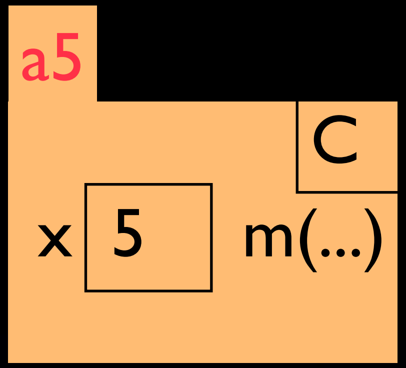
Look in frame for call for names; if not there, use scope box

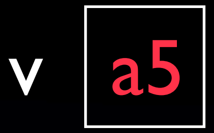


Executing method calls

template for frame for a call





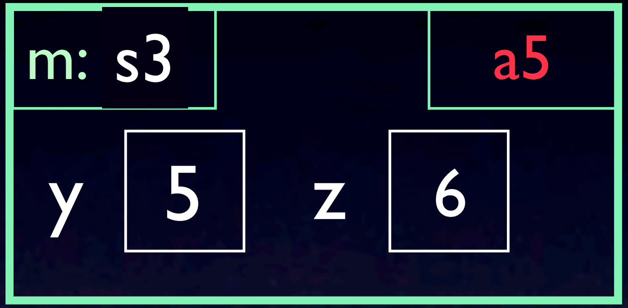


v.m(2+3);

```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

1. Draw frame for call
2. Store arg values in pars
3. Execute method body.

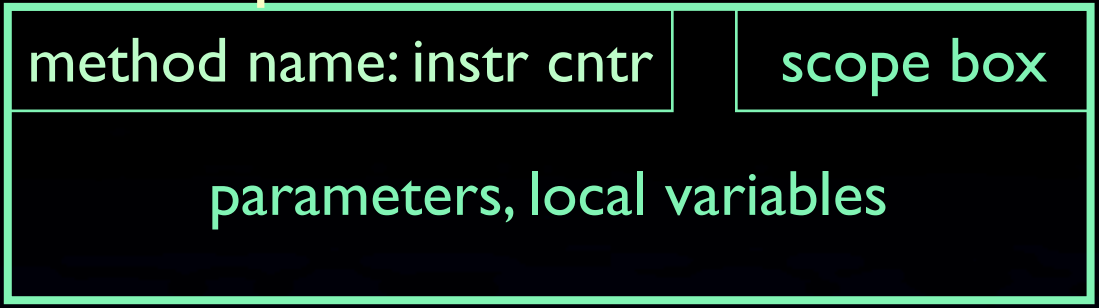
Look in frame for call for names; if not there, use scope box

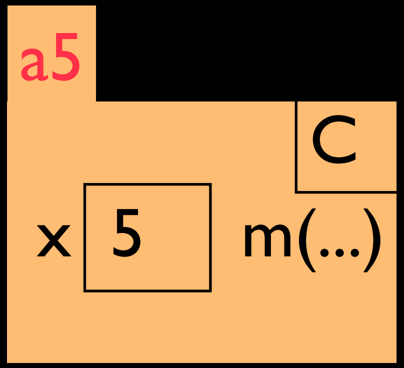


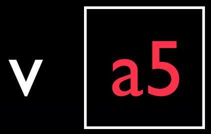
Friday, January 15, 2010

Executing method calls

template for frame for a call







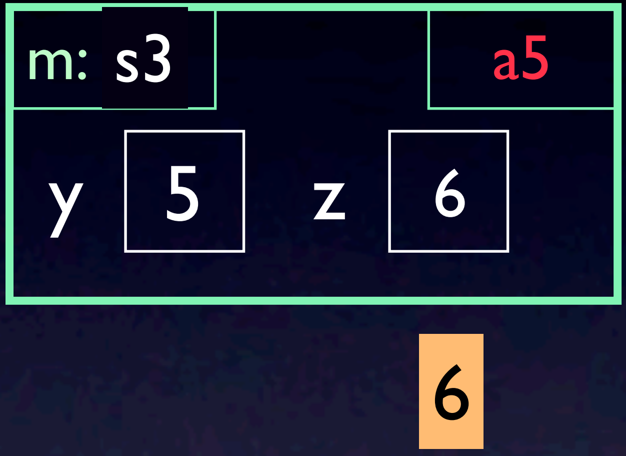
v.m(2+3);

```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

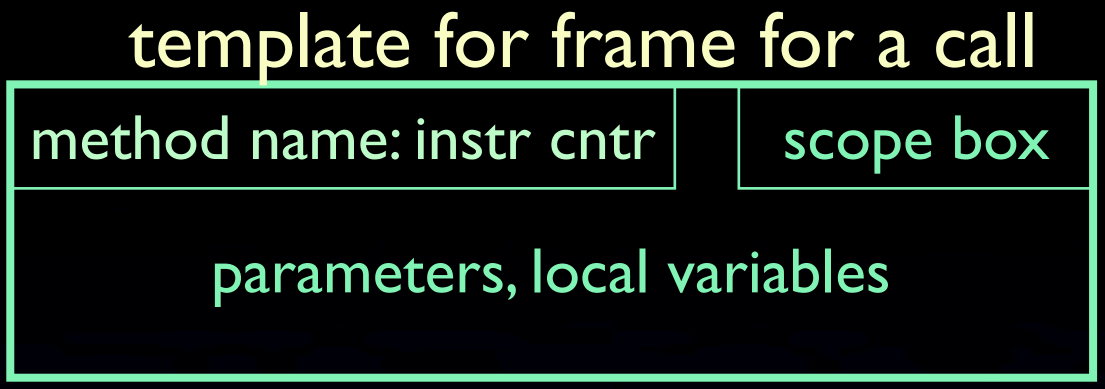
1. Draw frame for call

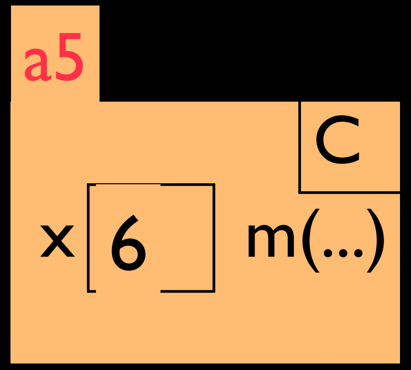
2. Store arg values in pars

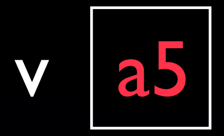
3. Execute method body.
Look in frame for call for names; if not there, use scope box



Executing method calls



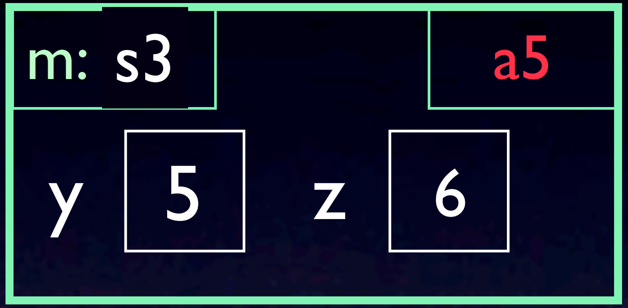




v.m(2+3);

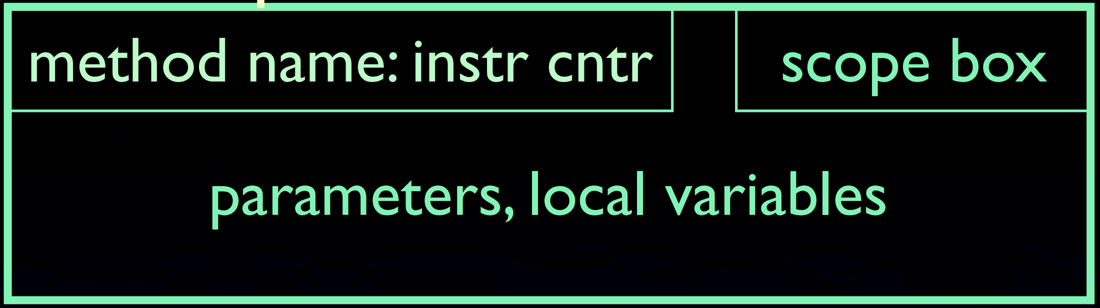
```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

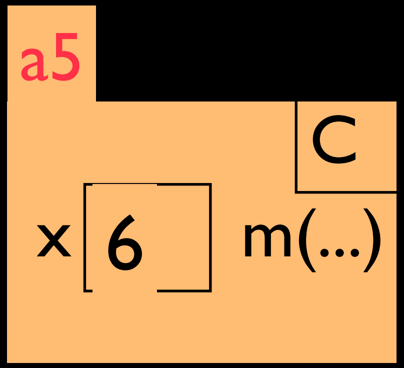
1. Draw frame for call
2. Store arg values in pars
3. Execute method body.
Look in frame for call for names; if not there, use scope box

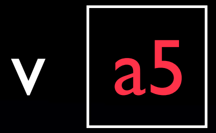


Executing method calls

template for frame for a call







v.m(2+3);

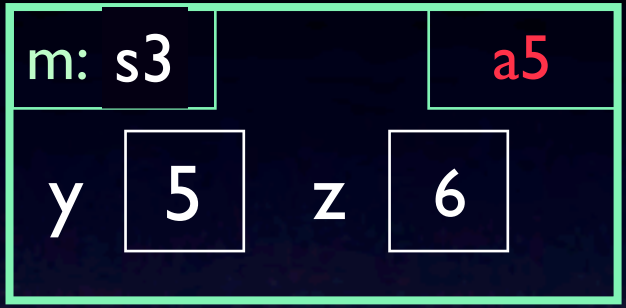
```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
              s2: int z= y+1;
              s3: x= z;
       }
   }
}
```

1. Draw frame for call

2. Store arg values in pars

3. Execute method body.
Look in frame for call for names; if not there, use scope box

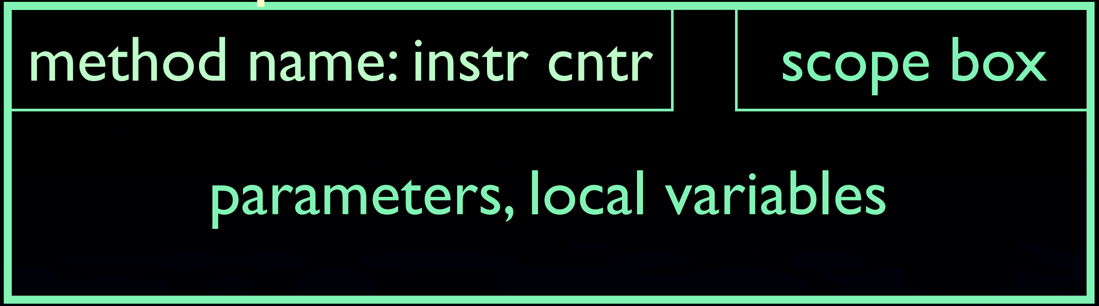
4. Erase the frame

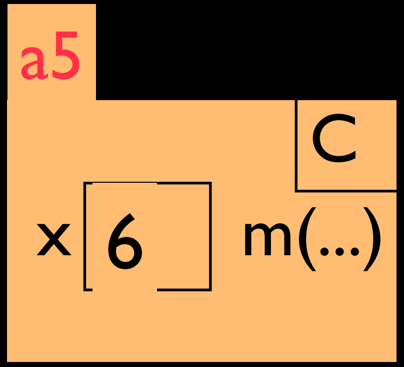


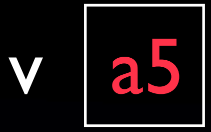
Friday, January 15, 2010

Executing method calls

template for frame for a call



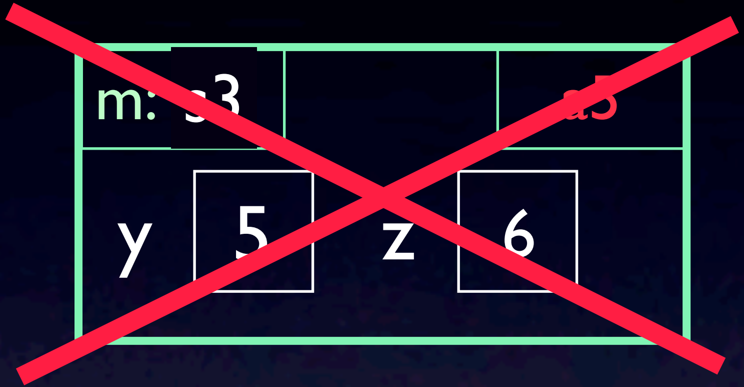




v.m(2+3);

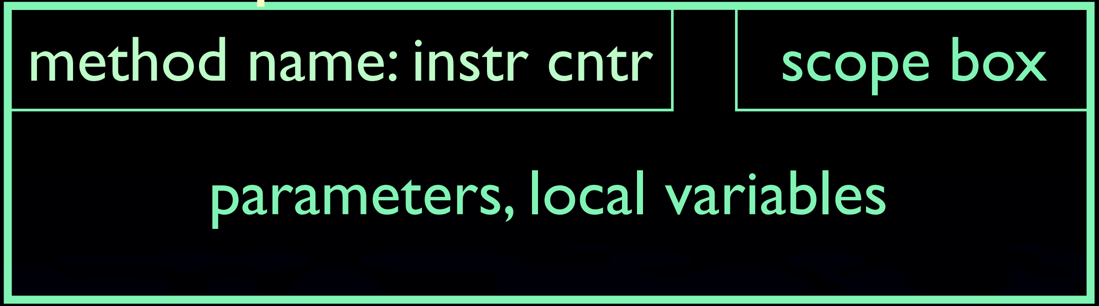
```
public class C {
   private int x;
   public void m(int y) {
      s1: if (y != 0) {
            s2: int z= y+1;
            s3: x= z;
      }
   }
}
```

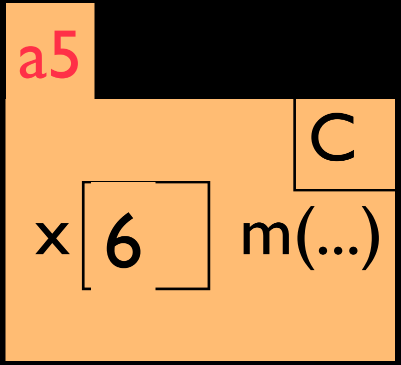
1. Draw frame for call
2. Store arg values in pars
3. Execute method body. Look in frame for call for names; if not there, use scope box
4. Erase the frame

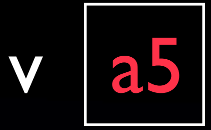


Executing method calls

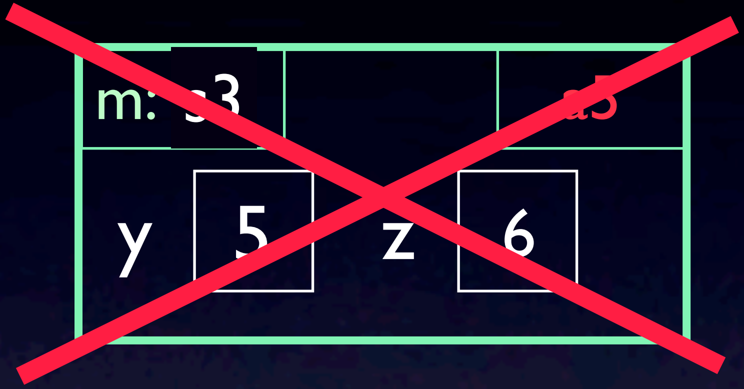
template for frame for a call







v.m(2+3);



```
public class C {
   private int x;
   public void m(int y) {
       s1: if (y != 0) {
               s2: int z= y+1;
               s3: x= z;
       }
   }
}
```

1. Draw frame for call
2. Store arg values in pars
3. Execute method body. Look in frame for call for names; if not there, use scope box
4. Erase the frame

Question: When is local variable z created?

Teaching programming skills

Many students have no idea how to go about developing a simple function.

Example: Student already had a 1-semester course in Matlab. Given clear spec for this method.

```
/** = a string representation of this Company ... (complete spec) */
public String toString() {
    return type + " company " + name +
            (id >= 0 ?". Id " + id : "") +
            ". Founded " + year + ". Has " + employee +
            " employees. Owns " + owns +
            (owns != 1 ? " companies." : " company.");
}
```

Can’t figure out how to write it without using nested if statement.
I was trying to write a series of conditional return statements.
What should the structure ultimately look like?

Teaching programming skills

Can’t figure out how to write it without using nested if statement.
I was trying to write a series of conditional return statements.
What should the structure ultimately look like?

My answer: Look at the structure of what is required: return:

> the type,
> the company name,
> the id, if it is there,
> the year it was founded,
> the number of employees it has,
> how many companies it has

Each piece may need some words around it, ...

Just write an expression for each piece, connect them with "+".
Build it up one piece at a time, testing to make sure each piece is right.

Teaching programming skills

Can’t figure out how to write it without using nested if statement.
I was trying to write a series of conditional return statements.
What should the structure ultimately look like?

My answer: Look at the structure of what is required: return:

the type,
the company name,
the id, if it is there,
the year it was founded,
the number of employees it has,
how many companies it has

Thanks! It seems so simple I can't believe I didn't think of it.

Each piece may need some words around it, ...

Just write an expression for each piece, connect them with "+".
Build it up one piece at a time, testing to make sure each piece is right.

Integrated development/testing of methods

Students need to see **you** develop and test functions, using stepwise refinement, interweaving coding and testing, and explaining your thought processes —or asking them to help.

```
/** = English equivalent of n.
    Precondition: 0 < n < 1,000,000.
    Examples:
    3:      "three"
    45:     "forty five"
    100:    "one hundred"
    127:    "one hunded twenty seven"
    1001:   "one thousand one"
    999099: "nine hundred ninety thousand ninety nine*/
  public static String anglicize(int n)
```

Recursion is easier than loops

Three things needed:

1. Write a precise specification of function
2. Write the base case
3. Write the recursive case --try to express the answer to the problem in terms of the same problem on a smaller scale.

Hints:

Discuss executing recursive function vs understanding it

Don't do Towers of Hanoi or Factorial!

Do functions that process strings

Then ask students to do 5-6-10 similar ones. Practice makes perfect

Show them interesting problems:

- Sierpinski triangles, Koch snowflakes, ...
- Tiling Elaine's kitchen
- functions on integers
- Quicksort

Demonstrate the development of recursive functions

Recursion: If you get the point, stop; otherwise, see Recursion.

Infinite recursion: See Infinite recursion.

Examples of recursive functions

Count number of ‘e’s in a string

Remove blanks from a string

Remove adjacent equal chars

Duplicate each character

Reverse a string

Tell whether a string is a palindrome

Compress a string with many adjacent equal characters (no digits).
E.g. for "aaaaaaaaaaaabbaaaaaazzzz" produce "a12b2a6z4".

“Commafy” an integer, e.g. change int 35476934 to “35,476,943”

Add the digits of integer d together

Count the number of times digit 3 occurs in integer d

Reverse integer d, e.g. from 45637 produce 73654

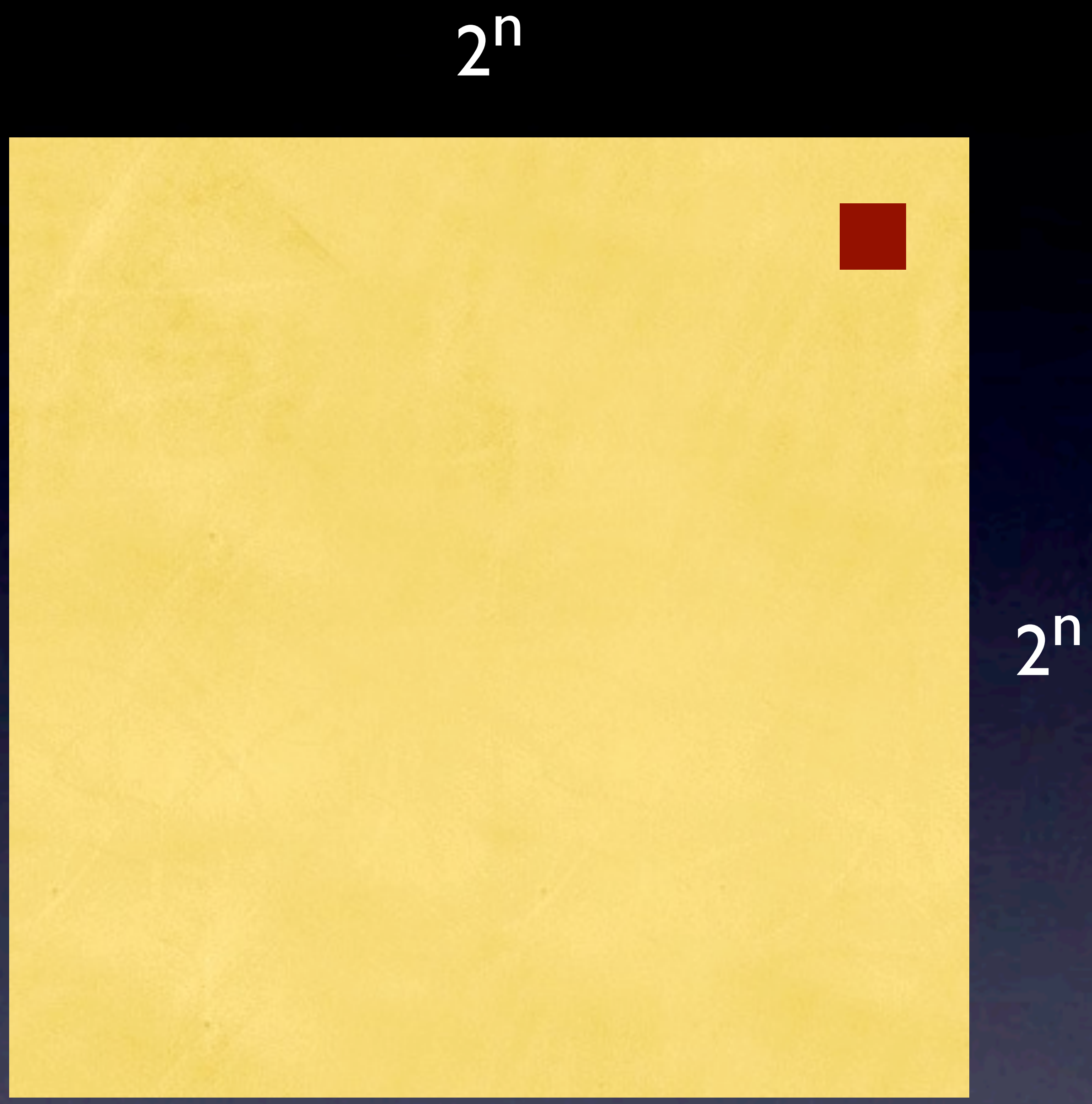
Complement integer d, e.g. from 93723 produce 17387

Count number of people in an ancestral tree (or, number of females, number of people with no descendants, etc.)

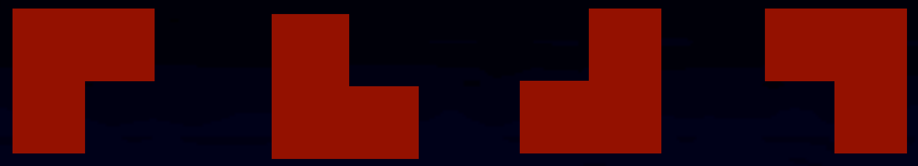
Examples of recursive functions

/** = sum of all integer values in obj.
 Precondition: obj is an object of one of the classes:
 Integer, Integer[], Integer[][], Integer[][][], etc.
 Examples: Below, a boldface integer like **4** represents an
 Integer object that contains that integer.
 For the argument **5**, the value 5 is returned.
 For the array {**1**, **2, 3**}, 6 is returned because 1+2+3 = 6.
 For the array {{**1**, **2**, **5**}, {**3**, **4**}}, 15 is returned because 1+2+5+3+4 = 15.
 For the array {{{**1**}, {**0**, **3**}, {}}, {{**1**,**2**,**3**}, {**3**}}}, 13 is returned because 1+0+3+0+1+2+3+3 = 13.
 */
public static int intDeepSum(Object obj) {

Tiling Elaine's 16 x 16 Kitchen

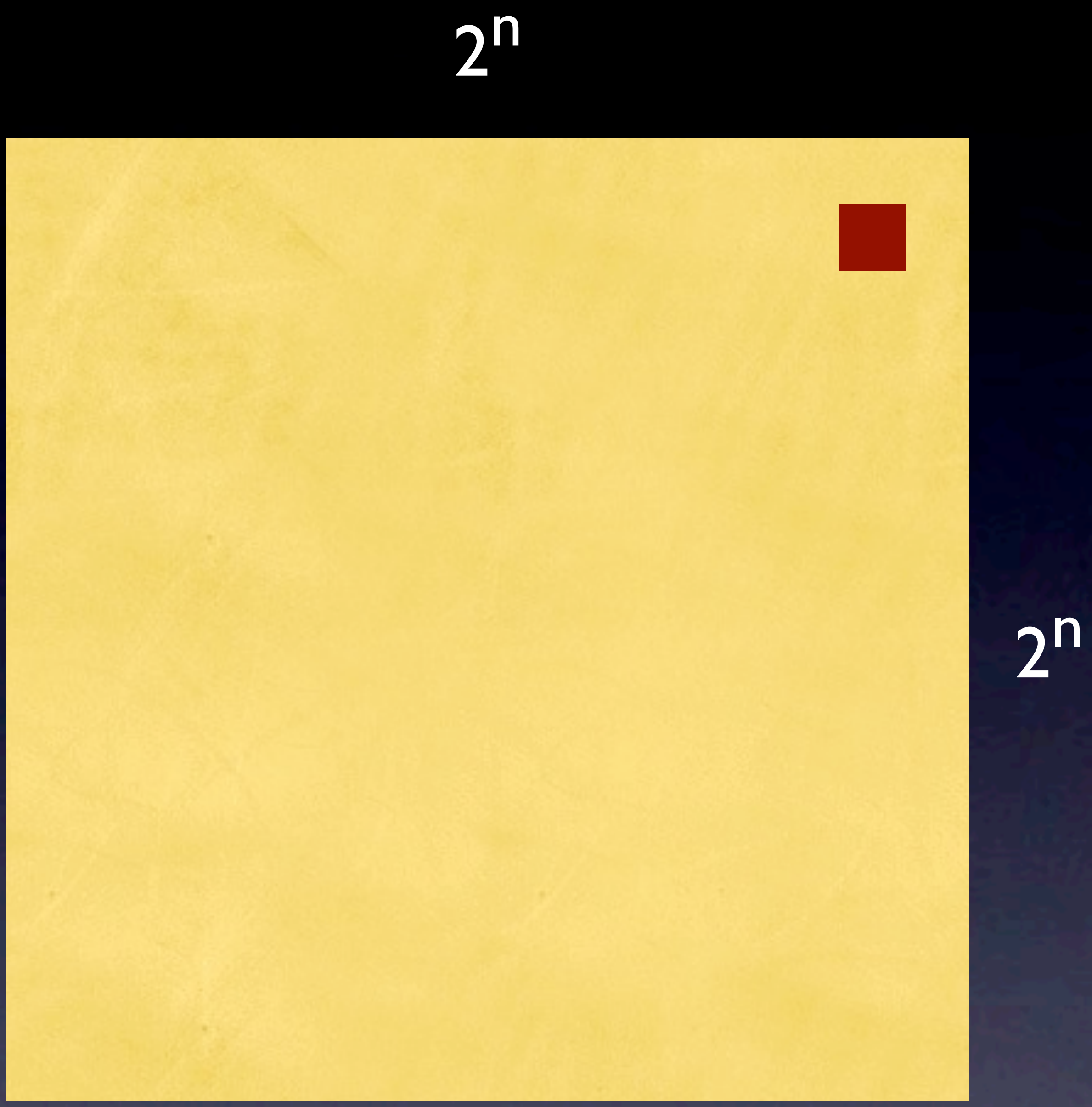


one 1 x 1 square of kitchen is covered by a refrigerator

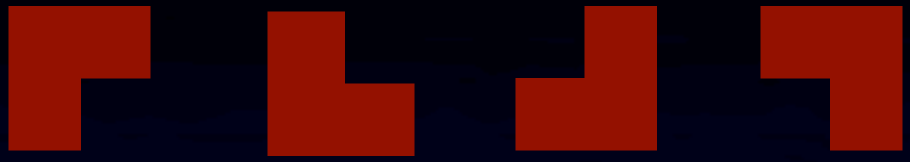


Tile the kithen with L-shaped tiles

Tiling Elaine's 16 x 16 Kitchen



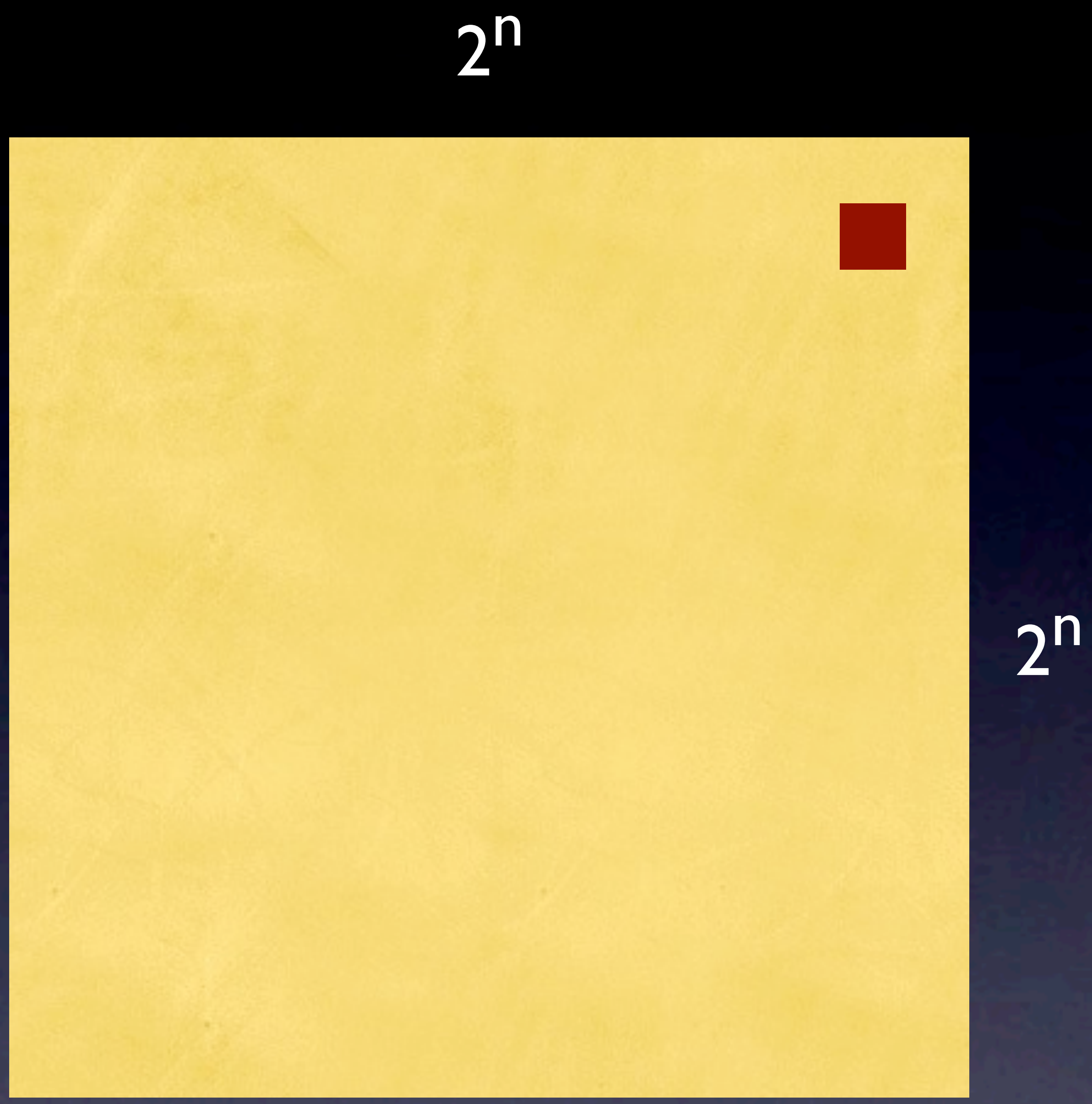
one 1 x 1 square of kitchen is covered by a refrigerator



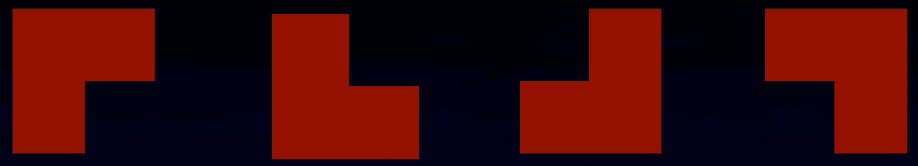
Tile the kithen with L-shaped tiles

Base case: 2^0 x 2^0 kitchen

Tiling Elaine's 16 x 16 Kitchen



one 1 x 1 square of kitchen is covered by a refrigerator



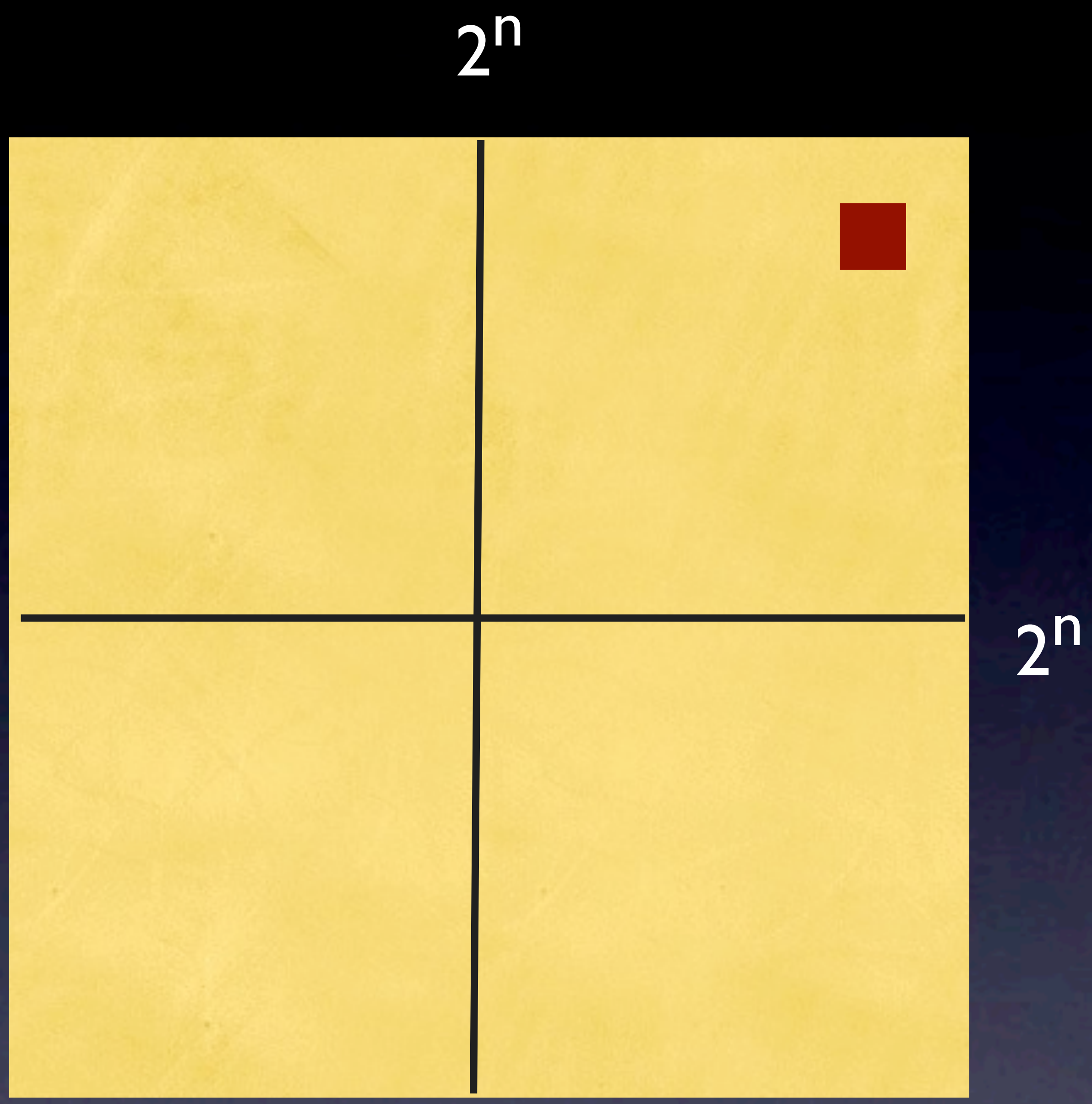
Tile the kithen with L-shaped tiles

Base case: 2^0 x 2^0 kitchen

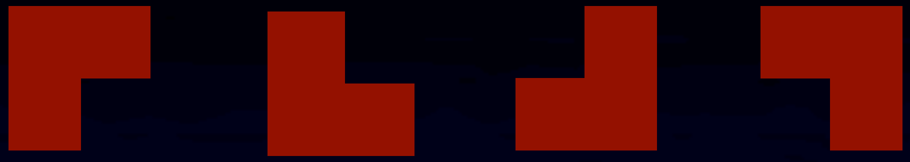
Recursive case: 2^n x 2^n kitchen:

How can we solve it in terms of 2^{n-1} x 2^{n-1} kitchens?

Tiling Elaine's 16 x 16 Kitchen



one 1 x 1 square of kitchen is covered by a refrigerator



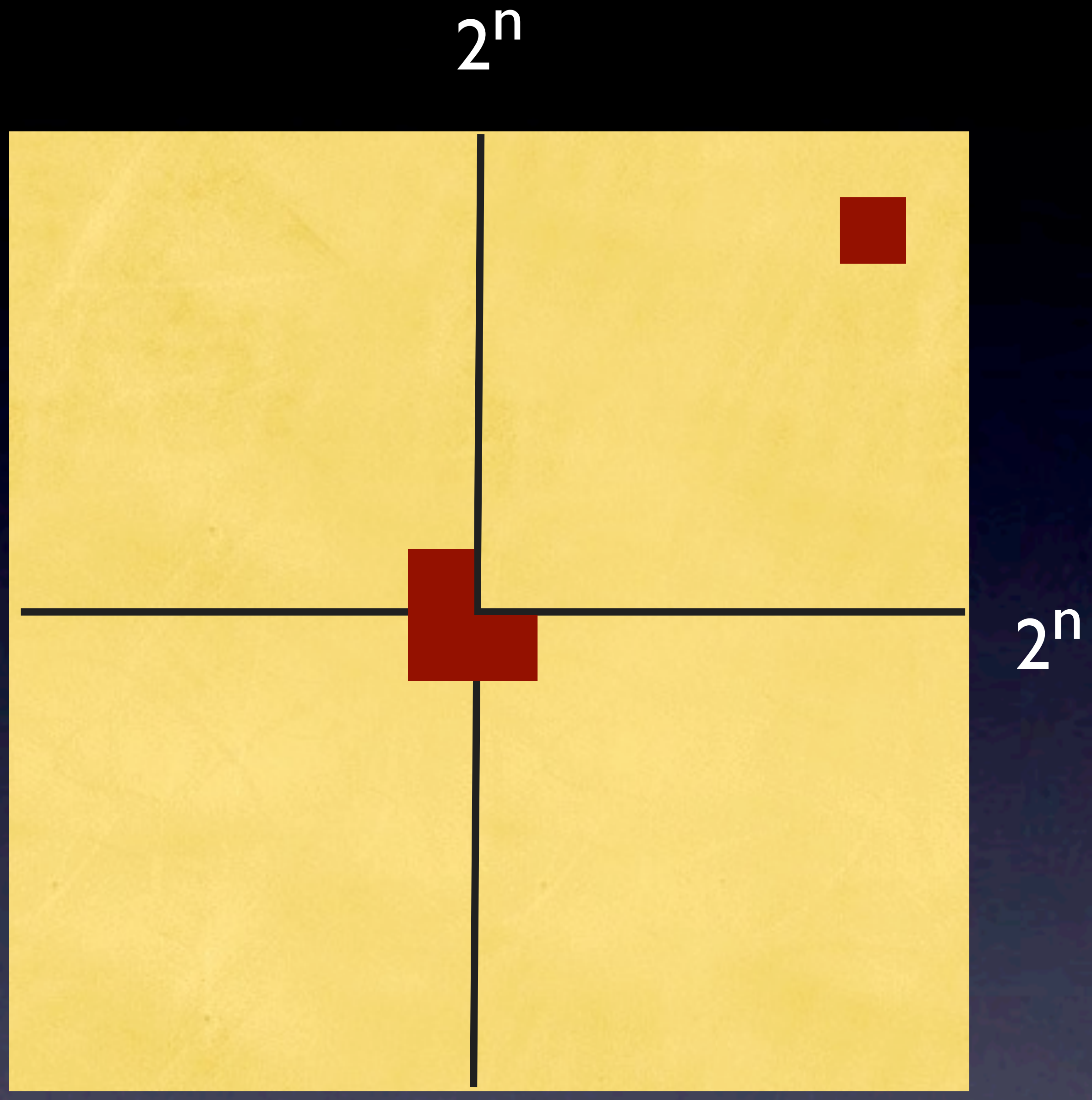
Tile the kithen with L-shaped tiles

Base case: 2^0 x 2^0 kitchen

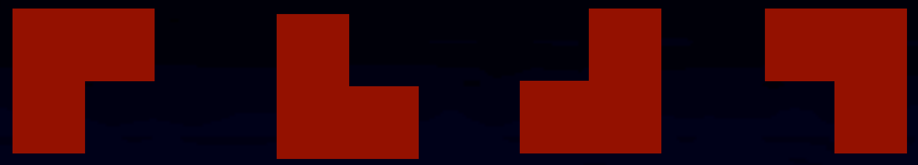
Recursive case: 2^n x 2^n kitchen:

How can we solve it in terms of 2^{n-1} x 2^{n-1} kitchens?

Tiling Elaine's 16 x 16 Kitchen



one 1 x 1 square of kitchen
is covered by a refrigerator



Tile the kithen
with L-shaped tiles

Base case: 2^0 x 2^0 kitchen

Recursive case: 2^n x 2^n kitchen:

How can we solve it in terms of 2^{n-1} x 2^{n-1} kitchens?

Let students know where the course is heading

We teach in basically a bottom-up style, introducing one new “feature” at a time.

Bad for “global learners”, who need to see the big picture.

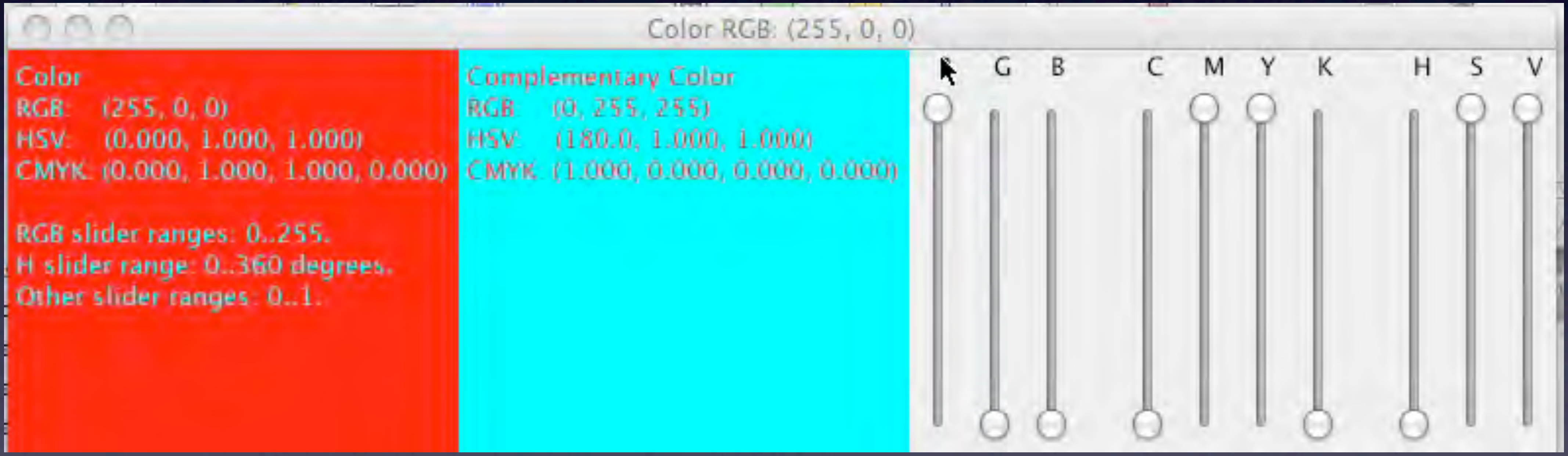
Compensate by giving overviews, show where course is going

Let students know where the course is heading

We teach in basically a bottom-up style, introducing one new "feature" at a time.

Bad for "global learners", who need to see the big picture.

Compensate by giving overviews, show where course is going



Color
RGB: (255, 0, 0)
HSV: (0.000, 1.000, 1.000)
CMYK: (0.000, 1.000, 1.000, 0.000)

RGB slider ranges: 0..255.
H slider range: 0..360 degrees.
Other slider ranges: 0..1.

Complementary Color
RGB: (0, 255, 255)
HSV: (180.0, 1.000, 1.000)
CMYK: (1.000, 0.000, 0.000, 0.000)

Let students know where the course is heading

We teach in basically a bottom-up style, introducing one new "feature" at a time.

Bad for "global learners", who need to see the big picture.

Compensate by giving overviews, show where course is going

Let students know where the course is heading

We teach in basically a bottom-up style, introducing one new "feature" at a time.

Bad for "global learners", who need to see the big picture.

Compensate by giving overviews, show where course is going

Let students know where the course is heading

We teach in basically a bottom-up style, introducing one new "feature" at a time.

Bad for "global learners", who need to see the big picture.

Compensate by giving overviews, show where course is going

Let students know where the course is heading

We teach in basically a bottom-up style, introducing one new "feature" at a time.

Bad for "global learners", who need to see the big picture.

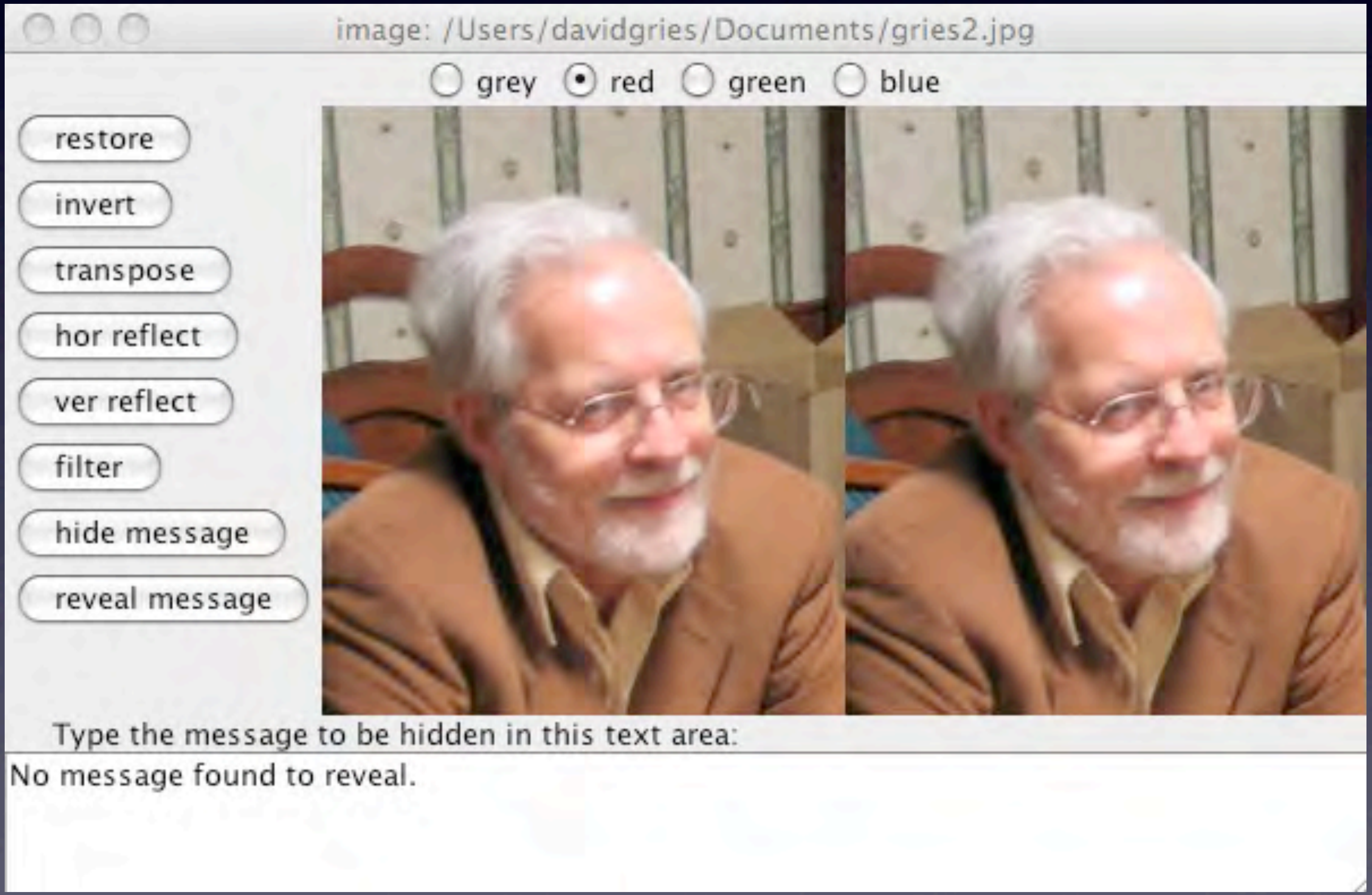
Compensate by giving overviews, show where course is going

Let students know where the course is heading

We teach in basically a bottom-up style, introducing one new "feature" at a time.

Bad for "global learners", who need to see the big picture.

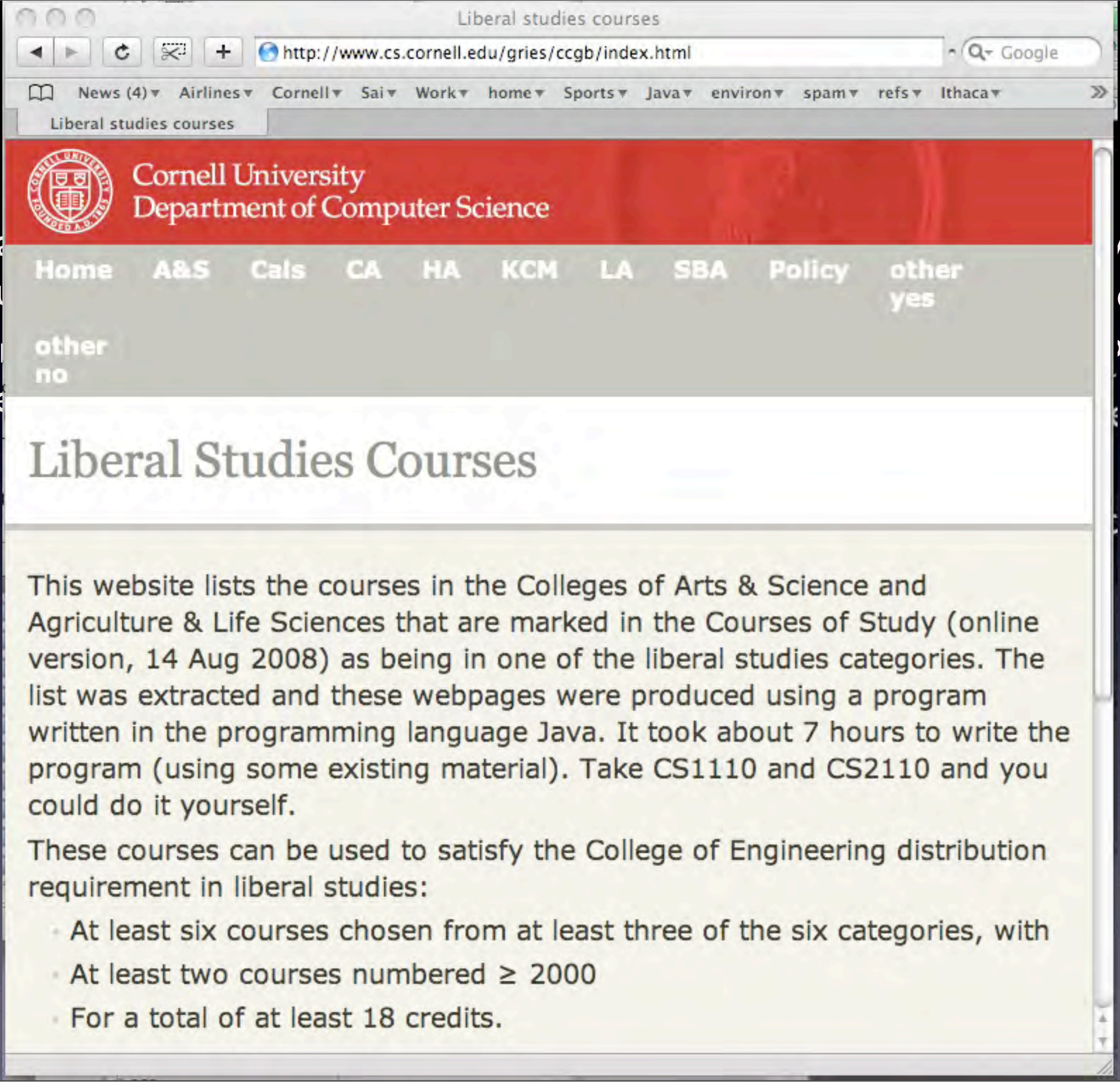
Compensate by giving overviews, show where course is going



Type the message to be hidden in this text area:

No message found to reveal.

We tea ving
introdu ere
Bad fo ping
see the



Liberal Studies Courses

This website lists the courses in the Colleges of Arts & Science and Agriculture & Life Sciences that are marked in the Courses of Study (online version, 14 Aug 2008) as being in one of the liberal studies categories. The list was extracted and these webpages were produced using a program written in the programming language Java. It took about 7 hours to write the program (using some existing material). Take CS1110 and CS2110 and you could do it yourself.

These courses can be used to satisfy the College of Engineering distribution requirement in liberal studies:

- At least six courses chosen from at least three of the six categories, with
- At least two courses numbered ≥ 2000
- For a total of at least 18 credits.

How to find material on what I talked about

David and Paul Gries. A Multimedia Introduction to Programming Using Java. Springer Verlag, NY 2005.

Comes with a CD that has 250-odd 2-5 minute lectures with synched animation. On the CD, we can really concentrate on program development.

Webpage for current course.
www.cs.cornell.edu/courses/cs1110/2009fa/

You can get slides of lectures and labs, assignments, etc.

Teach OO first

Two aspects to programming:

Structural/organizational

Procedural

Based on pedagogical principals, I teach structural/ organizational aspect first: OO

Teach OO first

Two aspects to programming:
Structural/organizational
Procedural

Based on pedagogical principals, I teach structural/ organizational aspect first: OO

1. Java expressions and assignment: int, double, boolean, String
2. Objects --method calls.
3. Class definition (a subclass), with a function decl. and a procedure decl.
4. Fields, constructors, getter/setters, JUnit testing
5. JUnit testing, class Object, static variables

Up to this point, the only statements they know are method call, assignment, and return.

Teach OO first

Two aspects to programming:
Structural/organizational
Procedural

Based on pedagogical principals, I teach structural/ organizational aspect first: OO

1. Java expressions and assignment: int, double, boolean, String
2. Objects --method calls.
3. Class definition (a subclass), with a function decl. and a procedure decl.
4. Fields, constructors, getter/setters, JUnit testing
5. JUnit testing, class Object, static variables

Up to this point, the only statements they know are method call, assignment, and return.

6. Methods: first look at if statements
7. super-this. Inside-out rule. Stepwise refinement
8. Constructors in subclasses. Stepwise refinement
9. Wrapper classes. Stepwise refinement
10. Recursion
11. Recursion
12. Casting, instanceof, function equals