CS100J 18 September 2007

Prelim: Tues 25 Sep. 7:30 to 9:00
Review: Sun. 23 Sep 1–3, Phillips 101

More on Methods. Developing methods.

Also: The inside-out rule; and the use of **this** and **super.**

Read sec. 2.5 on stepwise refinement

Listen to PLive activities 2.5.1 -- 2.5.4!

Quotes that relate to specifying a method before writing it.

A verbal contract isn't worth the paper it's written on.

What is not on paper has not been said.

If you don't know where you are going, any road will take you there.

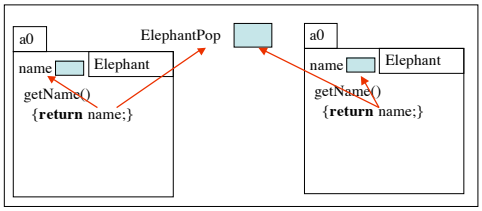
If you fail to plan you are planning to fail.

Don't try to solve a problem until you know what the problem is.

Inside-out rule

Inside-out rule in most programming languages (see p. 83):

Code in a construct can reference any of the names declared or defined in that construct, as well as names that appear in enclosing constructs (unless a name is declared twice, in which case the closer one prevails).

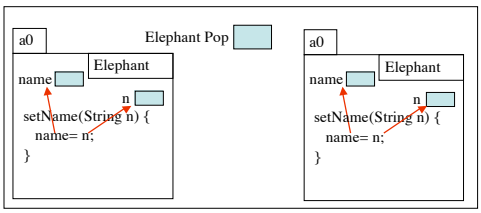


File drawer for class Elephant 2

Inside-out rule

Inside-out rule in most programming languages (see p. 83):

Code in a construct can reference any of the names declared or defined in that construct, as well as names that appear in enclosing constructs (unless a name is declared twice, in which case the closer one prevails).

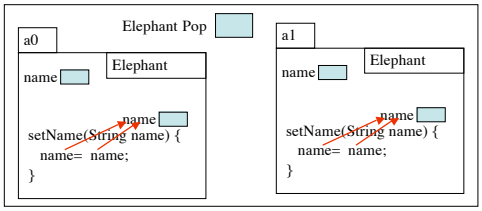


File drawer for class Elephant 3

Inside-out rule

Inside-out rule in most programming languages (see p. 83):

Code in a construct can reference any of the names declared or defined in that construct, as well as names that appear in enclosing constructs (unless a name is declared twice, in which case the closer one prevails).



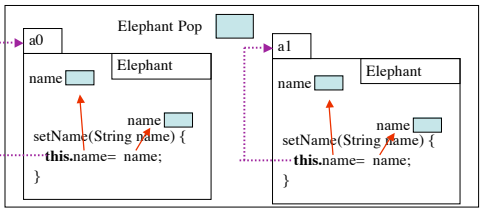
File drawer for class Elephant 4

About using **this** and **super**

Within an object, this refers to the name of the object itself,

In folder a0, **this** refers to folder a0.

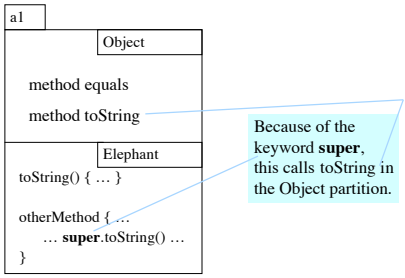
In folder a1, **this** refers to folder a1.



File drawer for class Elephant 5

About using **this** and **super**

Within a subclass object, super refers to the object —except the lowest partition.



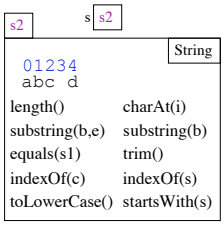
Because of the keyword **super**, this calls toString in the Object partition.

6

Strings String s= "abc d";

Text, pp. 175–181, discusses Strings
Look in CD ProgramLive
Look at API specs for String

s.length() is 5 (number of chars)
s.char(3) is 'c' (char at position 3)
s.substring(2,4) is "c "
s.substring(2) is "c 4"
" bcd ".trim() is "bcd" (trim beginning and ending blanks)



DO NOT USE == TO TEST STRING EQUALITY!

s1 == s2 tests whether s1 and s2 contain the name of the same object, not whether the objects contain the same string.

Use s1.equals(s2)

Developing another string function

```
/** Precondition: s contains at least one integer (without sign), and they are
    separated by commas (blanks are permissable after a comma). There are
    no blanks at the beginning and end of s.
    If the first integer is 0, remove it and its following comma if there is one
    (and following blanks).
    E.g. s = "52,     0, 76385"     Don't change s,
         s = "000,    11"           Change s to "11".
         s = "00"                   Change s to "".*/
public static String fix(String n)
```


Anglicizing an integer

```
/** = the English equivalent of n, for 1 <= n < 1,000
    e.g. ang(3) is "three"
         ang(412) is "four hundred twelve"
         ang(762) = "seven hundred sixty two" */
public static String ang(int n)
```

Hint at how to do it. When we add 5 + 3 + 2 + 8, we start with x = 0 and add one value to x at a time, step by step:

x= x + 5; x= x + 3; x= x + 2; x= x + 8;

Definition of x: *x is the sum of the values added so far.*

Can we start with String variable s = "" and step by step catenate pieces on to the end of it?

What's the definition of s?

Anglicizing an integer

```
/** = the English equivalent of n, for 1 <= n < 1,000
    e.g. ang(3) is "three"
         ang(641) is "six hundred forty one" */
public static String ang(int n)
```

Start with String variable s = "" and step by step catenate pieces on to the end of it? Use two local variables, s and k.

	s	k
start:	""	641
	"six hundred"	41
	"six hundred forty"	1
end	"six hundred forty one"	0

Definition of s and k:

ang(n) is s + ang(k)

To find ang(n), anglicize k and append the result to s.

Anglicizing an integer

```
/** = the English equivalent of n, for 1 <= n < 1,000
    e.g. ang(3) is "three"
         ang(641) is "six hundred forty one" */
public static String ang(int n) {
  // ang(n) is  s + ang(k)
  String s= "";
  int k= n;

}
```

You are expected to study section 13.4!

This definition of s and k is *very important*. This definition will drive the development. Whenever we append something to s, we have to change k to keep the definition true. The definition helps us develop the method body and helps the reader understand it.

Whenever you declare a local variable whose value will change often over execution of the method, **write a comment near its declaration to define it**!!

You will use the definition often as you develop the method. Without the definition, you will forget what the variable means and will make mistakes.

Anglicizing an integer

```
/** = the English equivalent of n, for 1 <= n < 1,000
    e.g. ang(3) is "three"
         ang(641) is "six hundred forty one" */
public static String ang(int n) {
  // ang(n) is  s + ang(k)
  String s= "";
  int k= n;

}
```

The rest of this lecture is devoted to the development of a different algorithm for anglicizing an integer —or rather the same algorithm but expressed entirely differently, using DrJava.

The final program will be on the course website.