CS1110 23 September 2008

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!

Prelim: Tues 30 Sep, 7:30 to 9:00
Review: Sun 28 Sep, 1–3, Phillips 101

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.

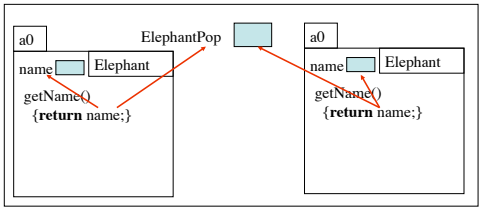
If you fail to plan you are planning to fail.

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).

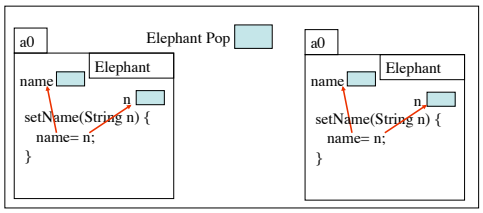


File drawer for class Elephant

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).

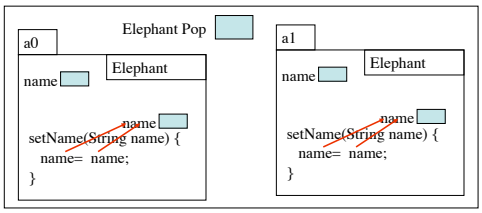


File drawer for class Elephant

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).



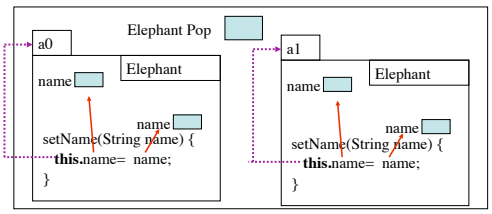
File drawer for class Elephant

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.

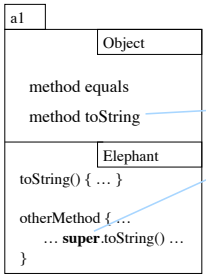
In folder a1, **this** refers to folder a1.



File drawer for class Elephant

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

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

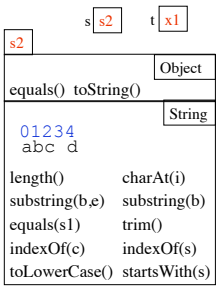


Because of the keyword **super**, this calls toString in the Object partition.

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!

s == t tests whether s and t contain the name of the same object, not whether the objects contain the same string.

Use s.equals(t)

Developing another string function

```
/** Precondition: s contains at least one integer (without sign), and they are
    separated by commas (blanks are permissible after a comma). There are
    no blanks at the beginning and end of s.
    = s but with its first integer removed (remove also following comma,
      if there is one, and following blanks).
    E.g.  s = "52,      0, 76385"      Return "0, 76385"
          s = "000,     11"            Return "11"
          s = "00"                     Return ""
*/
public static String removeInt(String s)
```


Developing another string function

```
/** Precondition: s contains at least one integer (without sign), and they are
    separated by commas (blanks are permissible after a comma). There are
    no blanks at the beginning and end of s.
    If the first integer is 0, return s but with its first integer and following ‘,’
    and blanks removed; otherwise, return s.
    E.g.  s = "52,      0, 76385"      Return s,
          s = "000,     11"            Change s to "11".
          s = "00"                     Change s to "".*/
public static String removeZero(String 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) {

}
```

The rest of this lecture is devoted to the beginning of the development of an algorithm for anglicizing an integer.

The final program will be on the course website after Thursday’s lecture.

Principles and strategies

Principle: Write a method spec before you write the body.

Compile often: Compiling often will help you catch syntax errors quickly and easily.

Mañana Principle: Write the specification of a method and “stub” it in, so that it can be compiled and produces something that allows further development. Put off its complete development until later. (Mañana means tomorrow, or an indefinite time in the future.)

Intersperse program development with testing: The worst thing you can do is to write a complete program and then begin testing. Because if there is an error, you have no idea where it is and how to find it. However, if you test each method as completely as possible after writing it, then any errors *should* be localized to that method.