CS100J   12 February 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: Thurs Feb 21, 7:30 to 9:00
Review: Sun. 17 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.
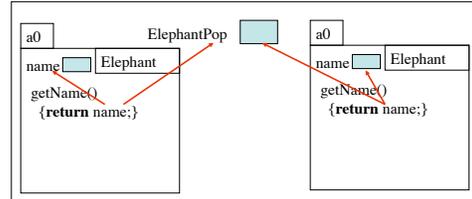
If you fail to plan you are planning to fail.

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

1

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).

a0   ElephantPop

name   Elephant
getName()
    {**return** name;}

a0

name   Elephant
getName()
    {**return** name;}

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).
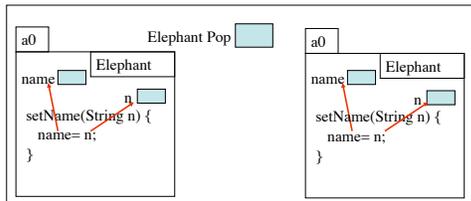

a0   Elephant Pop

name   Elephant
    n
setName(String n) {
    name= n;
}

a0

name   Elephant
    n
setName(String n) {
    name= n;
}

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).
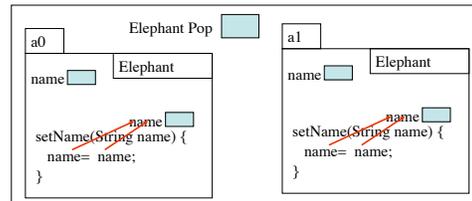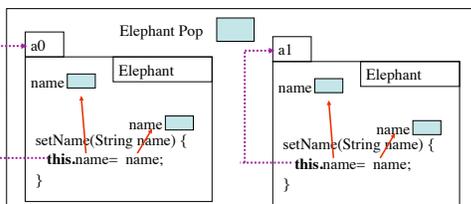

a0   Elephant Pop

name   Elephant
    name
setName(String name) {
    name= name;
}

a1

name   Elephant
    name
setName(String name) {
    name= name;
}

File drawer for class Elephant   4

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

Within an object, this refers to the name of the object itself,
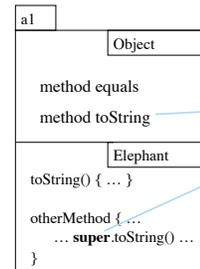
In folder a0, **this** refers to folder a0.

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

Elephant Pop

a0

name   Elephant
    name
setName(String name) {
    this.name= name;
}

a1

name   Elephant
    name
setName(String name) {
    this.name= name;
}

File drawer for class Elephant   5

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

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

a1

Object

method equals

method toString

Elephant

toString() { … }

otherMethod { …
    … **super**.toString() …
}

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

6

1

Slide 7

Strings   String s= "abc d";

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

s2   s s2

```
01234
abc d
```
String

length()   charAt(i)
substring(b,e) substring(b)
equals(s1)   trim()
indexOf(c)   indexOf(s)
toLowerCase()   startsWith(s)

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)

7

Slide 8

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)

8

Slide 9

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)

9

Slide 10

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.

10

Slide 11

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.

11