CS1110 10 February 2009

Inside-out rule; use of **this** and **super**
Developing methods (using String ops).
Read sec. 2.5 on stepwise refinement
Listen to PLive lectures 2.5.1–2.5.4.

Reading for next lecture: the same

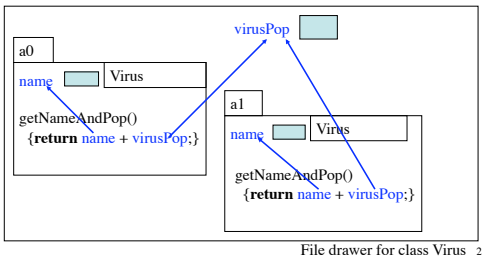
Prelim: Thu 19 Feb, 7:30 to 9:00, Baker 200
Review: Sun 15 Feb, 1–3, Phillips 101

For today:
Turn in Assignment 2.
Pick up three handouts:
- Assignment 3, due next Monday. "Group" early on CMS and watch the course "Announcements" webpage.
- Prelim review notes
- Today's slides

(Suggestion: get a 3-ring binder and a 3-hole punch. Take notes on 3-hole-punched looseleaf paper).

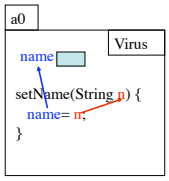
Remember frame boxes and figuring out variable references?
The *inside-out rule* (see p. 83)

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. (If a name is declared twice, the closer one prevails.)

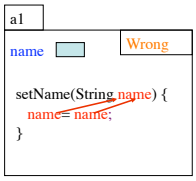


File drawer for class Virus

Method parameters participate in the inside-out rule: remember the frame.



Parameter n would be found in the frame for the method call.



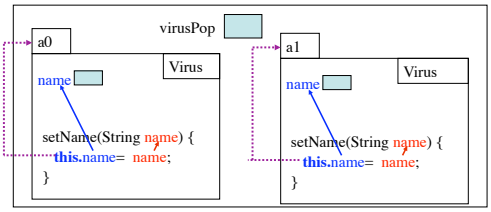
Parameter name "blocks" the reference to the field name.

A solution: **this** and **super**

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

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

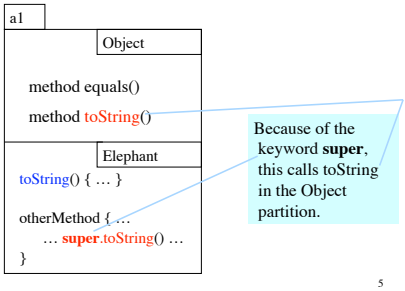
In folder a1, **this** refers to a1.



File drawer for class Virus

A solution: using **this** and **super**

Within a subclass object, super refers to the superclass.



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

Strings are (important) objects that come with useful methods.

String s= "abc d";

Note the "index (number) from 0" scheme:

```
01234
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.charAt(2) is 'c' (char at index 2)
s.substring(2,4) is "c " (NOT "c d")
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 a string function

```
/** Precondition: s contains a sequence of unsigned integers separated by
    commas (each possibly followed by blanks). There are
    no blanks or commas 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 a string function

```
/** Precondition: s contains a sequence of unsigned integers separated by
    commas (each possibly followed by blanks). There are
    no blanks or commas at the beginning and end of s.
    = s IF the first integer isn't 0
    =s but with its first integer and delimiting comma and blanks removed  if
 the first integer is originally 0
    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)
```


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.