CS151 Fall 2012 Lecture 11

Stephanie R Taylor

September 28, 2012

1 Administrative Topics

- We take Quiz 3
- Do many people have the textbook? If you have it, is it useful?

2 Assignment/Project 4

Today, I will go over the basic outline of task 3 in assignment 4 and relate it to memory. This way we practice cloning.

The assignment asks you to:

I Create a Pixmap from a .ppm file. Then make four copies of it.

II Apply a different transformation to each of the four copies.

III Create an empty Pixmap large enough to display all four copies. Put the four (transformed) copies into the large Pixmap in a 2 by 2 grid.

Well, this isn't exactly the order of operations specified in the assignment, because Bruce asks you to interleave each pixmap copy with each pixmap transformation. But this order will produce the same results and it allows me to more cleanly describe what is happening in memory.

2.1 Phase 0

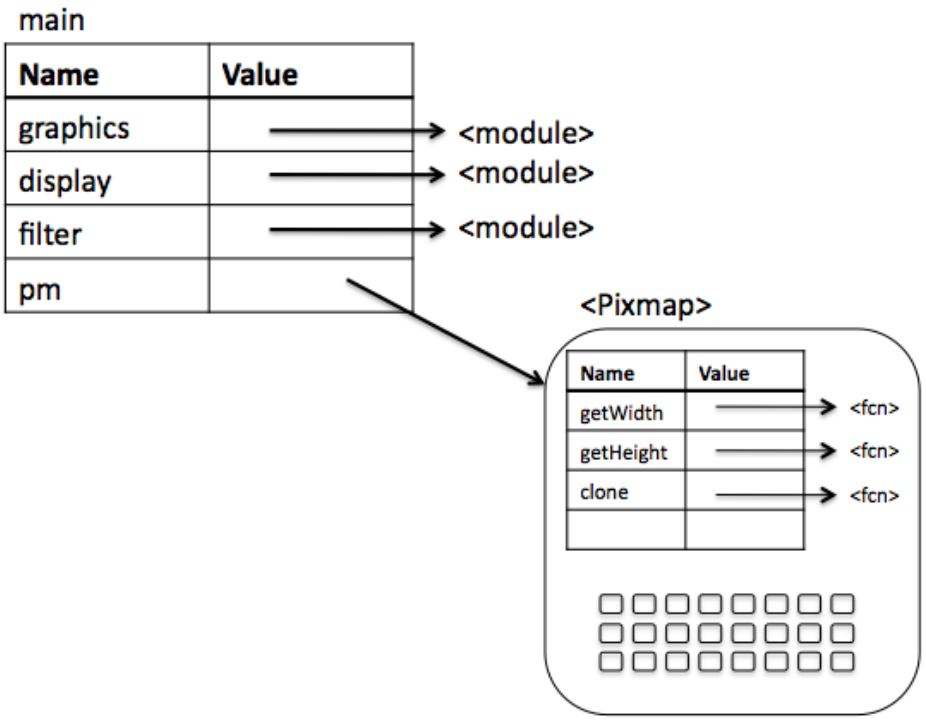
I will step through a complete program, that uses all of our graphics-related modules. In Phase 0, we import all them – graphics, display, and filter. The main symbol table is updated, but we don't need to show that here - it is old hat now.

2.2 Phase I

In phase I, we will load a Pixmap into a variable pm and then make four copies of it, storing them in variables map1, map2, map3, and map4. Here is the code:

```
1 pm = graphics.Pixmap('my_pic.ppm')
2 map1 = pm.clone()
3 map2 = pm.clone()
4 map3 = pm.clone()
5 map4 = pm.clone()
```

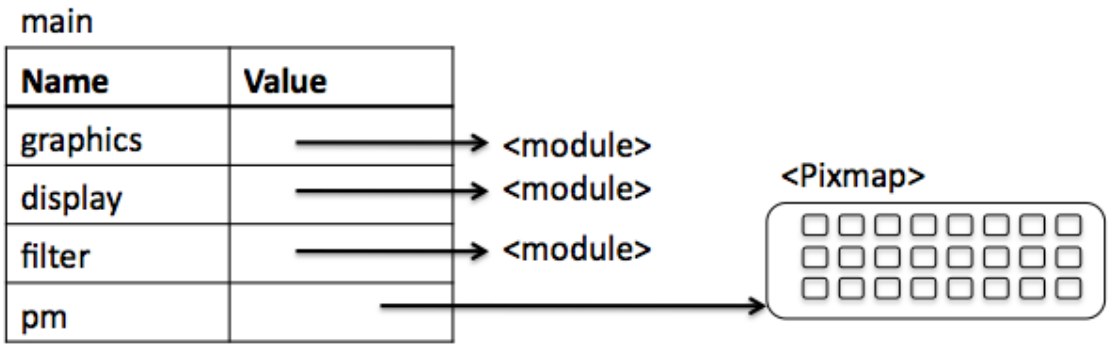
In line 1, the Pixmap object is created from the file named my_pic.ppm. The main symbol table is updated thus:



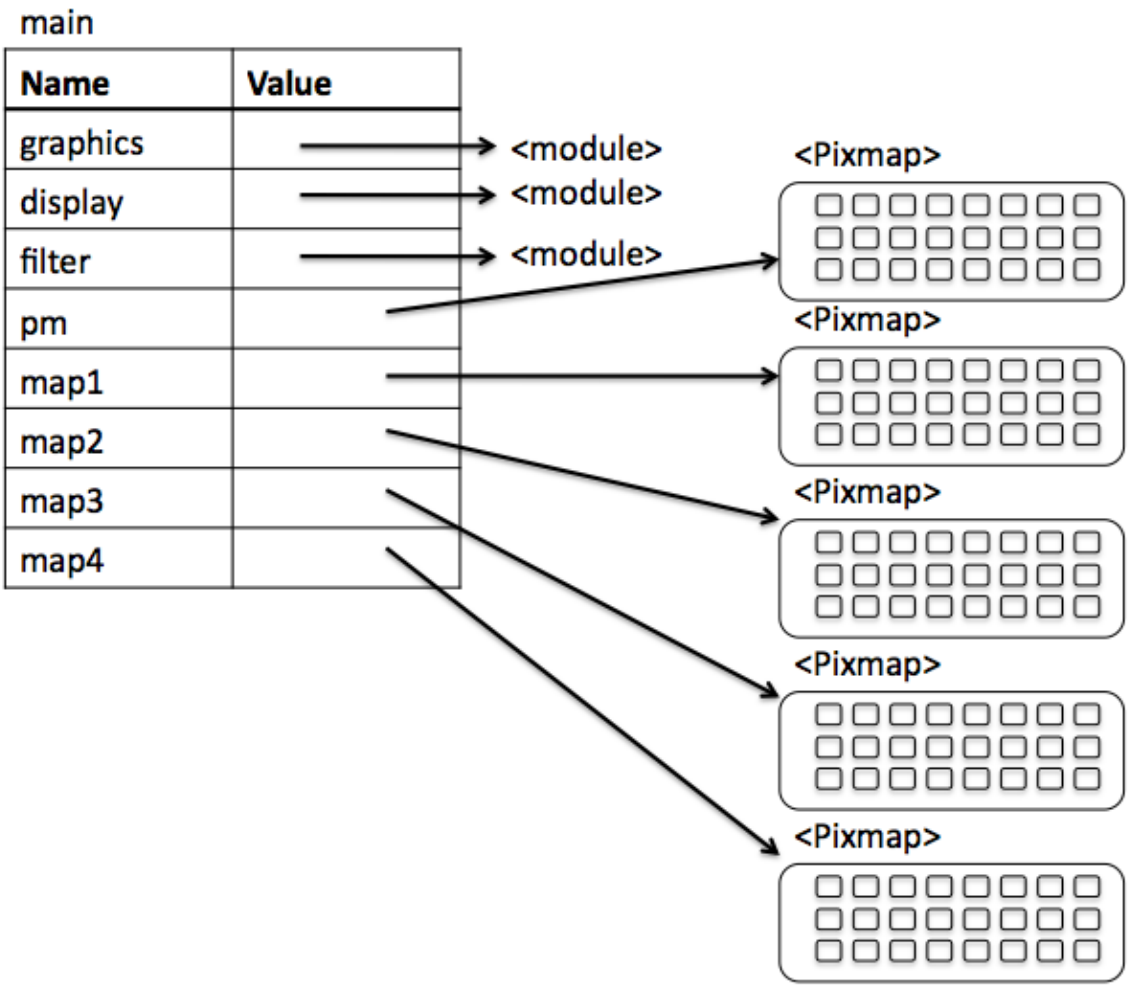
Notice that the Pixmap object has quite a complicated drawing. This is because I want to demonstrate a very cool feature of objects. Objects contain symbol tables! In an object's symbol table are entries for its data (not shown, but if it were, it would involve arrows to the pixels I drew above it) and entries for each of this methods (three are shown here). When John Zelle was writing the Graphics module, he wrote the code that defined the Pixmap type. As part of that code, he included these methods. To access these methods, we must use the dot notation (e.g. pm.clone()). The dot allows us to travel from the main table to the table inside the Pixmap to the entry for the clone method. This is a convenient notation for calling a function associated with an object. When we execute a method (such as clone), Python treats the object is itself as one of the inputs, even though it doesn't appear in the parentheses. So, in that sense pm.clone() is a call to a clone function that takes pm as input. This is quite a change from previous code. I find it helpful to think about the memory model. By placing the functions inside the object, the organization is immediately clear. And the dot notation is consistent with the dot notation we have already seen (i.e. it allows us to

travel from symbol table to symbol table). So it makes sense to call the clone method with the pm.clone() syntax.

But, our picture will be very cluttered if we continue to draw the symbol table in each object. So, I will leave that out and use a picture more like this:



We execute lines 2 through 5 and 4 copies are made. Note that we could have said map2=map1.clone() instead of map2=pm.clone() as long as map1 hasn't been transformed. But it is probably safest to use pm. Note that we could not have said map2=map1 and achieved the same results.

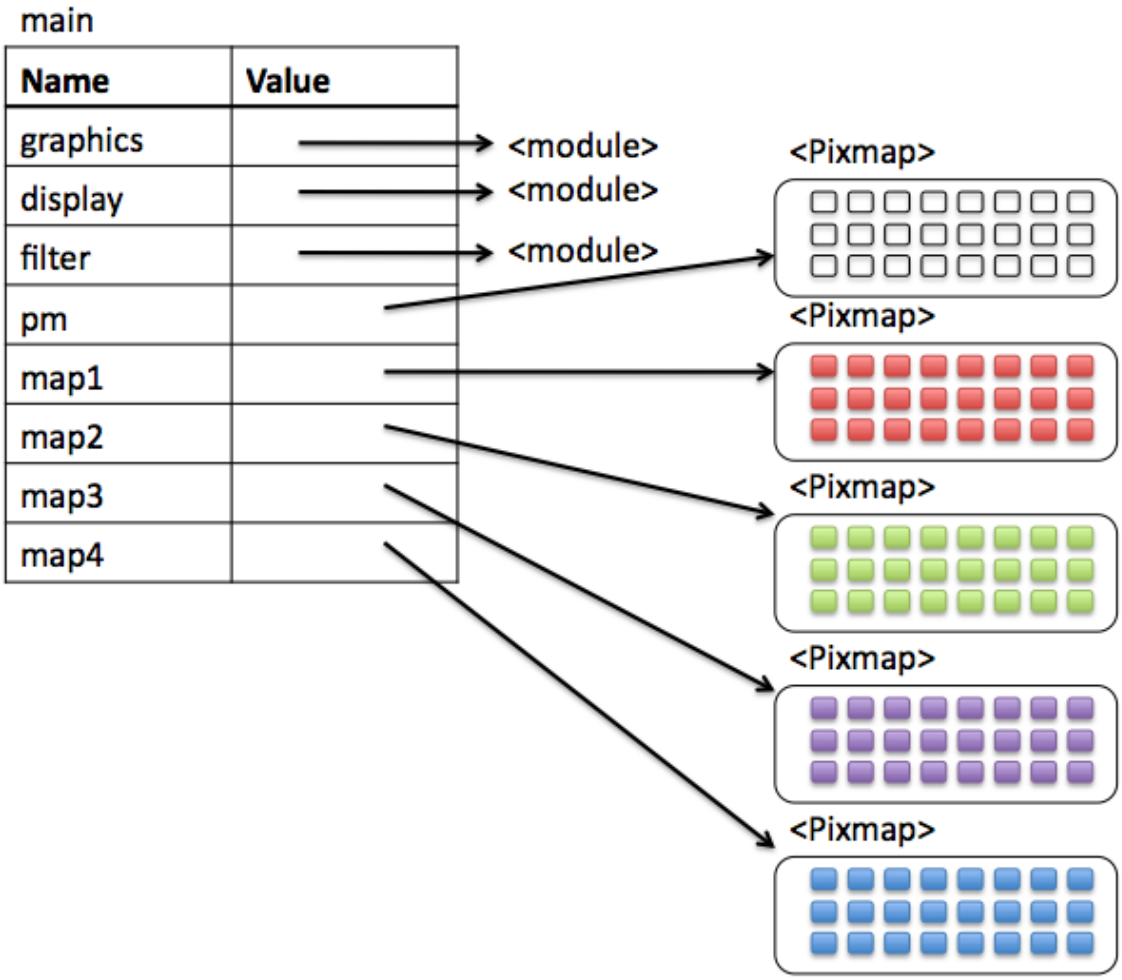

main
Name
Value
graphics
display
filter
pm
map1
map2
map3
map4
<module>
<module>
<module>
<Pixmap>
<Pixmap>
<Pixmap>
<Pixmap>
<Pixmap>

2.3 Phase II

Next we apply a different transformation to each pixmap, e.g.

```
negatePixmap(map1)
swapRedBluePixmap(map2)
grayPixmap(map3)
enhanceBluePixmap(map4)
```

and the result is four different Pixmaps:



2.4 Phase III

Finally, we make an empty Pixmap large enough to accommodate all four transformed Pixmaps. Then, we copy each pixmap into the large one.

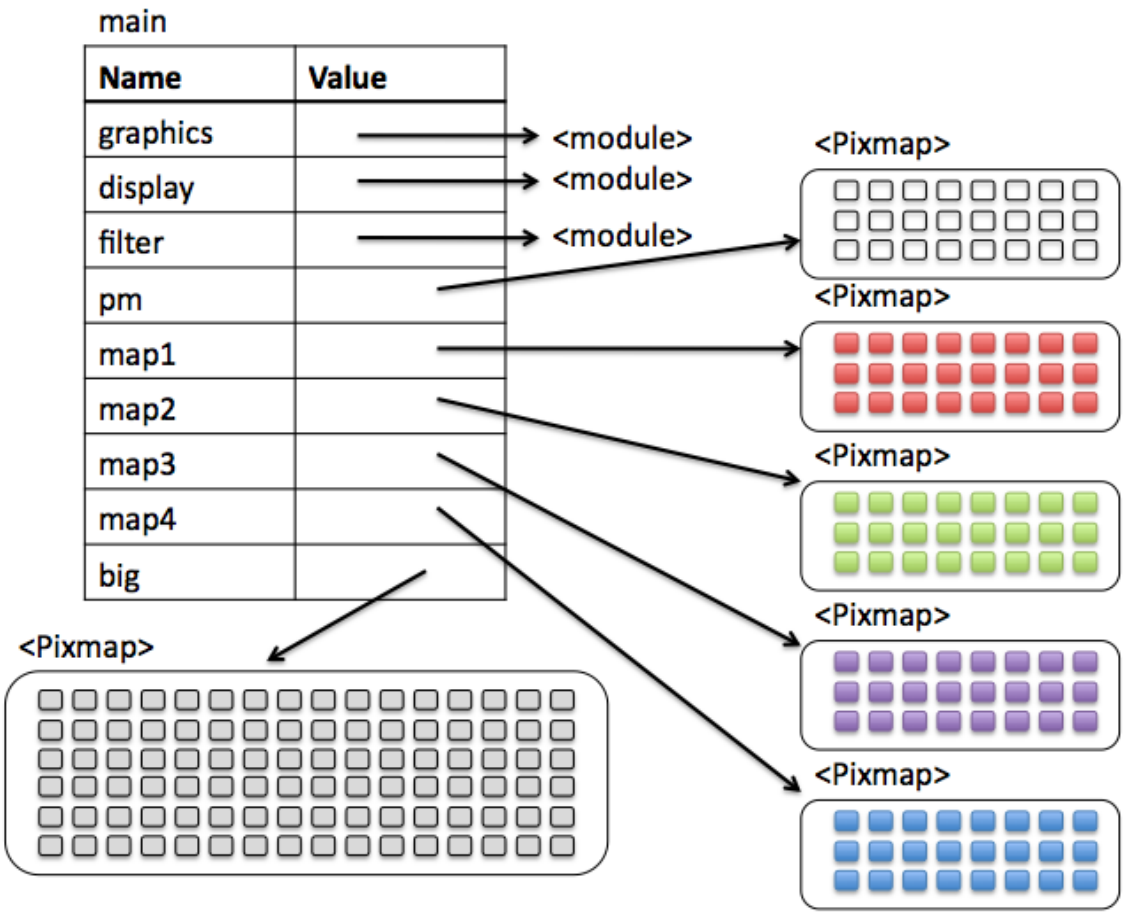
The copy function we will use (which is to be written by you) has the following declaration:

```
# place src into dst with upper left at x, y in dst
def putPixmap( dst, src, x, y ):
```

In other words, we specify where the upper left corner of the small pixmap (called src here) should go in the large pixmap (called dst here).

The main code, then needs to make the Pixmap and call putPixmap four times. Let's call the large Pixmap big. Then, we place map1 in the upper left corner, map2 in the lower left, map3 in the upper right, and map4 in the lower right.

```
1 big = graphics.Pixmap(2*pm.getWidth(), 2*pm.getHeight())
2 putPixmap(big, map1, 0, 0)
3 putPixmap(big, map2, 0, map1.getHeight())
4 putPixmap(big, map3, map1.getWidth(), 0)
5 putPixmap(big, map4, map2.getWidth(), map3.getHeight())
```


main
Name
Value
graphics
display
filter
pm
map1
map2
map3
map4
big
<module>
<module>
<module>
<Pixmap>
<Pixmap>
<Pixmap>
<Pixmap>
<Pixmap>
<Pixmap>

Then, after lines 2 through 5 have executed, we have the four pixmaps copied into it: