Slide 1

Administrivia

- Homework 4 due date extended to Thursday.
- Next quiz Tuesday (a week from today).

Slide 2

Minute Essay From Last Lecture

- (What can you do with these methods?)
- At this point, not a lot of inspired/inspiring ideas.
- A problem right now is that we don't seem to have a good way to get input, other than typing it all in (yuck!). We could use input redirection (to get input from a file), but then we have to get *all* input from the file.

Slide 3

Homework 4 Hints

- (Review problems.)
- (Hints as needed? continue at end of class)

Slide 4

Collection Methods — Review/Recap

- Many, many methods for operating on elements of a collection, more than we have time to look at. Simple example is revised versions of array/list demo programs. A few more today, more examples in textbook. Higher-order methods may seem strange at first. Practice helps!
- Textbook also, in passing, describes “curried” functions, needed in order to understand “fold” methods. Also not an easy topic to understand, but should make some sense with a bit of practice. For now okay to skim, just taking note of syntax. Also okay to skim sections on types and variable argument lists.

Slide 5

Mutability and Aliasing

- Up to now we've taken a fairly abstract view of what variables are and how things are stored in the computer's memory. Need to know a bit more in order for some things to make sense, though.
- So . . . In Scala all variables are what in Java are known as *references* — pointers to other memory areas. Some of these pointed-to things (*objects*) can be changed (*mutable*) and some can't (*immutable*).
- It's possible for two variables to point to the same object. If the object is mutable, things can get interesting — changes made via one variable are reflected when you access via the other. Sometimes this is useful; sometimes it's a source of trouble.

Slide 6

Argument Passing — Pass-By-Value

- (Terminology: I'll use "argument" and "parameter" interchangeably. Some writers make a distinction between the thing in the function and the thing in the calling program. I'll use the terms *formal* and *actual* to make that distinction.)
- When you call a function as we've done so far, Scala passes all arguments *by value*, into `val` variables. So you can't change the variables themselves. However, if the object being pointed to is mutable, it can be changed. Again — sometimes useful, sometimes a source of trouble.

Slide 7

Argument Passing — Pass-By-Name

- Scala offers an additional mechanism for argument-passing: *pass-by-name*. Not so easy to get the full picture of how it works, but used in some useful standard methods and so worth mentioning now.
- Basic idea is that rather than passing a value to the called function/method you pass a function, and the function is called *every time the argument is referenced* (rather than only once).

Slide 8

Pass-By-Name and Collection Methods

- Arrays and lists have two methods that use pass-by-name: `fill` and `tabulate`.
- Simple examples:

```
val a1 = Array.fill(4)(10)
val a2 = Array.tabulate(4){i => i+1)
val a3 = Array.fill(10)(readInt)
```

(Notice(?) that the syntax is that of a “curried” function.)

Slide 9

Minute Essay

- None — sign in.