Slide 1

Administrivia

- Reminder(?): Quiz 5 Tuesday. Likely topic is loops.
- Reminder: Homework 6 due today.
- Okay to skim long examples in chapters 9 and 10.

Slide 2

Minute Essay From Last Lecture

- (What interesting things can you do given the ability to work with files in programs?)
- Many interesting answers. I will say more another time and/or respond individually.
- Two key/common ideas — “persistent data” (save results from one execution for the next) and “big data” (more than you would type in interactively, possibly from some source such as a Web site).

Slide 3

Files — Review/Recap

- Scala makes it fairly easy to get text input from a file, with `scala.io.Source.fromFile`.
- Creating text files is somewhat more complicated and uses the underlying Java libraries, but still very doable.
- Simple examples — "sum integers from a file" (from last time, simplified a bit, showing input only), "translate text to upper case" (on sample programs page, showing input/output).

Slide 4

Sidebar: Command-Line Arguments in Scala

- You've noticed that many commands (`vi`, `scala`, etc.) can be followed by additional text? this is one more way of getting input to a program — *command-line arguments*.
- In Scala, you can access these arguments via an array (of `String`s) called `args`.
- Often a good way to specify things such as filenames.

Slide 5

A Little About Objects and Classes

- A possible source of confusion at this point — some things (e.g., description of packages from last time, Scala API) only make sense if you know a little about objects and classes. Key ideas of "object-oriented programming", a major topic in follow-on course CSCI 1321.
- History of object-oriented programming goes back to (relatively!) early work on simulating physical systems with programs.
- If writing such a simulation, useful to have a nice way in the program to represent physical objects being simulated (e.g., a car in a simulation of traffic) and groups of similar objects (e.g., cars — all distinct but have common features).
- "Object-oriented programming" — generalize this idea to more abstract things (e.g., rational numbers, components of a GUI). Key idea is that objects somehow encapsulate both data (variables) and functionality (code).

Slide 6

A Little About Objects and Classes in Scala

- Scala classes represent different types — simple ones such as `Int` and `String`, more complex ones such as `io.Source` and `PrintWriter`. Class definitions (somewhat analogous to function definitions) include variables and "methods" (functions that work on instances of the class).
- Scala objects are either instances of a class (e.g., a single `Int` or `String`) or singleton objects such as `math`. Definitions of singleton objects can include variables and methods.
- Keyword `new` creates an instance of a class.
- Syntax for accessing/invoking data and methods uses dot/period (e.g., `math.Pi` or `math.sqrt(2)`.
- API shows objects and classes (sometimes have both with the same name — e.g., `Int`).

Slide 7

Case Classes — Motivation

- Arrays, lists, and loops were all introduced with the comment that even without them you can compute anything computable, but all of these constructs make it easier to do some things, or to do them in a way that may be easier for to understand.
- Case classes similarly don't really add any new functionality, but they do give us a better way to group related pieces of information — we can do that with tuples, sort of, but tuples give us no way to indicate what their elements mean. (For example, a tuple of two `Int`s could represent a rational number or a point in 2D space.)

Slide 8

Case Classes

- Case classes are a very simple example of a *user-defined type* (analogous to predefined types such as `Int`, `String`, `List`, etc.). (In object-oriented terms, they're a simple kind of class, with data/variables only.)
- What they give you is a way to define a named type (e.g., `Rational`) that represents a collection of related objects (e.g., the numerator and denominator) and give the parts names:

```
case class Rational(numerator : Int, denominator : Int)
val r1 = Rational(1, 4)
println(r1.numerator + "/" + r1.denominator)
```

Slide 9

Case Classes, Continued

- You don't get the ability to define, within the class, operations for the type, but you could do that separately in functions:

```
def rationalToString(r : Rational) : String = {
  r.numerator + "/" + r.denominator
}
```

- (If you also want to define operations, you need full-fledged classes. Next semester, for those continuing!)
- (Example(s)?)

Slide 10

Minute Essay

- We have only a few classes left. Topics I want to address include what to do about errors (such as what happens when you try to use `readInt` and the text isn't a number), a little about sorting and searching (for those students continuing to CSCI 1321), and little about GUIs in Scala.
- Any thoughts about which of these would help/interest you most?