Computer Science 202
Introduction to Programming
The College of Saint Rose
Fall 2012

Topic Notes: Classes

So far, our programs have operated on data (*i.e.*, variables and parameters) that we can categorize in just two ways:

- primitive types, such as `int`, `double`, `char`
- object types, such as `String`, `Scanner`, `Random`, `DecimalFormat`

It is these object type we will focus on a bit more. In particular, we will see how to introduce our own object types into our programs.

Every object in a Java program is an entity that can contain both *fields* – which are like the variables we've been using in our programs, and *methods*, which operate on the data in those fields.

The idea of an object is central to the *object-oriented programming* paradigm, which has been very popular since being introduced a few decades ago.

The idea is that we write program components, called *classes*, which represent the "objects" we wish to represent in our program. For each object, we include fields that are used to represent the state of the object and methods that allow that state to be queried or modified.

The text includes an example of an alarm clock. They came up with a list of fields that can be used to describe the state of an alarm clock. It is similar to this list:

- the current hour (0-23)
- the current minute (0-59)
- the current second (0-59)
- the alarm hour (0-23)
- the alarm minute (0-59)
- the alarm status (on or off)

And some methods that can be used to modify the state of the alarm clock:

- set current time

- set alarm time
- disable alarm
- enable alarm
- stop currently sounding alarm

We might also consider some methods to query the current state of the alarm:

- get current time
- get alarm time
- get alarm status (on or off)

And then the alarm clock might have some other things it does "on its own" – its state changes as the time proceeds:

- increment time by 1 second
- start sounding alarm

In Java, the functionality of an object is described in a *class*. We have been writing classes all semester as containers for our `main`, and recently, a few other methods. It turns out that this is just a small fraction of what a Java class can be used to do.

We look at a mechanism to achieve this by an example. Consider this example, which we could have written much earlier in the semester:

See Example: RatiosNoClass

This program maintains information about ratios of integer values. We create two ratios, each represented by 2 `int` variables, and print them, modify them, and compute their decimal equivalents.

But with a class that represents a `Ratio` object, we can *encapsulate* the numbers (the numerator and denominator) into fields, and provide methods to construct, access, and modify the fields.

See

See Example: Ratios

and the extensive comments within for details.