Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-four)

4.12: Nested Classes

Java allows classes to be defined not only as separate entities within a package, but also as "nested classes" defined inside of other classes. These nested classes are called Inner Classes. They have some interesting properties, and are different in several important respects from nested classes in C++. We will only briefly touch on Inner Classes here. Note that although I am introducing Inner Classes here for completeness, they should be considered a somewhat advanced topic. Don't worry if the subtleties of the material below escape you for now.

Any class can contain, nested inside of it, the definition of another class:

```
public class Outer
{
    private int x = 1;
    public class Inner     // See Note #1 below
    {
       private int y;
       Inner()
       {
          y = x + 1;       // Note #2
       }
    }

    public Outer()
    {
       Inner i = new Inner();
       // x = i.y;         // Note #3
    }
}
```

Note #1: Class Inner could be marked "private" because it is a member of class Outer. ("Top-level" classes still can't be marked private!) Thus no class other than Outer could know about the existence of Inner.

Note #2: Two things are surprising about this line. One is that Inner can read the private variable x of class Outer. Classes nested inside other classes have full access to all variables declared in any of their enclosing classes.

The second surprising thing (especially to any C++ programmers) is that there is an x variable to refer to at all! Every instance of an inner object *must* be associated with one and only one outer object; this is true in general for all nested classes in Java. The x member belonging to this so-called *enclosing instance* is the one being referred to here. When an inner is created from within the outer class, the enclosing instance is the one that created the object. If code outside of outer wants to create an inner, it must do so like this:

```
Outer o = new Outer();
Outer.Inner i = o.new Inner();
```

What's going on is that you must supply an enclosing instance for the Inner by using o.new instead of just new.

Note #3: Note that Outer can access Inner's private variables as well.