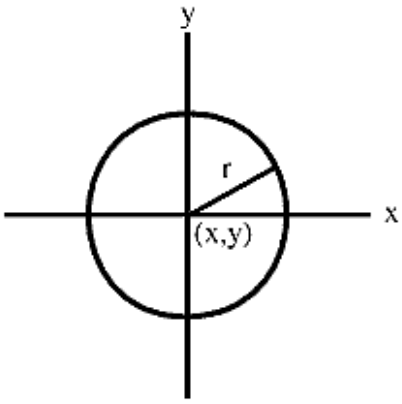
Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-four)

4.2: Constructors

Let's say you're writing a program that manipulates geometric shapes: a texture-mapped game, perhaps, or a theorem prover. You need a data structure to represent a Circle. Well, a Circle can be described with three numbers: an *x* and a *y* coordinate for the center, and a radius *r*.



We can define a Java class to hold these values:

```
public class Circle
{
   public double x, y, r;
}
```

and proceed to create circles as needed:

```
// Create a new Circle
Circle frisbee = new Circle();
// position at the origin
frisbee.x = frisbee.y = 0;
// make it 2 units across
frisbee.r = 1.0;
```

This is a lot of code just to set up what looks like a generic Circle object. Nevertheless, it's essentially the same sort of code you must write in C or Visual Basic when using user-defined data types such as Circle. If you use a lot of Circles, you'll probably need to type these same lines over and over again in your program. Java lets you put this kind of initialization code inside the class and arrange to have it called whenever you create a new Circle object:

```
public class Circle
{
    // member variable
    public double x, y, r;
    // Create a generic Circle
    public Circle()
    {
```

```
        x = y = 0.0;
        r = 1.0;
    }
}
```

Then when you type new Circle( ), the initialization function (called a constructor) gets called. Each new Circle you create is already positioned at the origin and has a radius of 1. You save two lines of typing every time you create a Circle.

A few things about the constructor Circle( ) are worth mentioning. First of all, notice that Circle( ) has no return type. Constructors never declare a return type, not even void. Because of this, you can never call a constructor directly; only "new" can call one. In other words:

```
Circle c = Circle();        // WRONG
Circle();                   // Also WRONG
Circle okay = new Circle(); // OK
```

Second, notice that in the constructor, we can refer to x, y, and r directly, without any qualification. When the constructor is called, the system has already set aside memory to hold a new Circle object. When code in a constructor refers to x, for example, it is referring to the x variable inside the memory set aside for the particular object being created. In general, code in a class can always refer to member variables of that class in this way. In Java all code (well, almost all; static methods have no current instance, as we'll see) executes in the context of a "current object." The current object can actually be referred to with a special variable called this:

```
// Create a generic Circle
public Circle()
{
  this.x = this.y = 0.0;
  this.r = 1.0;
}
```

Such a version of the constructor is exactly equivalent to the earlier version; the "this," if omitted, is implied.