# Object composition: Circle Class with location

**The goal for this exercise** is to make sure that you can create and use classes to solve a problem, and that you can compose classes out of other classes.

Let’s consider how we might create a class that represents a circle that is located at a particular place in our (X,Y) coordinate system. Let us furthermore assume that we already have a Point class, which contains data and code that represents a 2D, (x,y) style point, like so:

**class** Point{

**double** x;

**double** y;

};

Now that we know what problem we will (eventually) solve, let’s step back, and examine a similar (but different) problem – this way we can see how to solve this type of problem, but this explanation won’t give away the answer to this exercise. The similar-but-different problem will be how to create a Line class.

One way we could define the Line class is to create a class that consists of two end points, like so:

**class** Line {

**public**:

**double** x1; // first endpoint

**double** y1;

**double** x2; // second endpoint

**double** y2;

};

However, if we do this, we’ll be duplicating code that we’ve already implemented in our Point class. Furthermore, if we wanted to do something like find out how long the line is (i.e., the distance between the two points), instead of writing all the code here in the Line class, we’d rather do something like ask the two Point objects (each of which represents the center of one of the circles) how far apart they are. This way, we can reuse the ‘Distance’ method on the Point class.

Thus, a good way to implement the Line class is by COMPOSING our Line class out of two instances of the Point class (plus any other information we might want to store about the line – perhaps a color, or maybe a label/caption). Therefore, two Points must be added as a data fields to the Line, as shown below. Remember that we’ll need to initialize the new fields in the Line's constructors (also, you may want to add more constructors to initialize the new location).

**class** Point {

An interesting choice you'll have is whether to make the Point a public Property, or whether you want to give the Circle the individual getter/setter methods for the (X, Y) attributes of a Point. It will be valuable for you to think about this (if you're working with someone, please discuss this with your partner), and to weigh the pros and cons of each approach, before deciding on one or the other.

**class** Point {

**public**:

**double** x;

**double** y;

**Point**() { // Default Constructor

y = 0;

x = 0;

}

**Point**(**double**, **double**);

};

**Point::Point**(**double** \_x, **double** \_y){

x = \_x;

y = \_y;

}

**class** Line {

**public**:

Point\* pt1;

Point\* pt2;

**Line**() { // Default Constructor

pt1 = 0;

pt2 = 0;

}

**Line**(**double**, **double**, **double**, **double**);

};

**Line::Line**(**double** x1, **double** y1, **double** x2, **double** y2){

pt1 = **new** Point(x1,y1);

pt2 = **new** Point(x2,y2);

}

**NOTE:** You are NOT allowed to solve this problem using inheritance. You must specifically use an instance variable for the Point class.

**What you need to do for this exercise:**  In the starter project, add code near the the Object\_Composition\_Circle class, so that the following is done:

1. Create a Point class . It should have a constructor that allows you to instantiate (to create) Point objects, it should have a Print method, and it should have getter/setter methods for the X & Y attributes.   
     
   Once you've done this, uncomment the code in the Starter Project and compile.  
   1. You do NOT have to create Line class. That was just an example that was being used to illustrate the above concepts.
2. Create a Circle class, that is composed of the radius (a double), and the center (a Point object). In the Circle’s constructor, create the Point object that will be used to track where the Circle’s center is.
   1. The Circle class should also have a Print method. Make sure that you re-use the Point.Print method (instead of getting the X & Y fields individually)
3. Create several Circle objects, and print them out.  
     
   Once you've done this, uncomment the code in the “RunExcercise” method in the Object\_Composition\_Circle class and your code should compile.
4. For all of the above, don’t worry if your output is not EXACTLY like the comment – an extra newline here or there is fine.