Classes #2

These notes discuss the class code cc15. You should have that code in the Class1 folder open as you read through this.

In our last exciting episode of class notes, we left our hero on a dark and stormy night… er… sorry .. wrong story… We were exploring classes and we discussed this code:

```
// Class Code Set15
// Classes in Processing  #1

Square s;

void setup( )
{
   size( 600, 600 );

   s = new Square( );
   s.setup( );

   background( 0 );
}

void draw( )
{
   s.move( );
   s.draw( );
}
```

```
class Square
{
  // fields or variables
  float x, y, edge, dX, dY;
  color col;

  void setup( )
  {
     x = random( 10, 500 );
     y = random( 10, 500 );
     edge = random( 50, 200 );
     dX = random( 2, 10 );
     dY = random( 2, 10 );
     col = color( random( 255),
                  random( 255),
                  random( 255 ) );
  }
  void move( )
  {
     x += dX;
     if (x + edge > width || x < 0 )
     {
        dX = -dX ;
     }
     y += dY;
     if (y + edge > height || y < 0 )
     {
       dY = -dY ;
     }
  }
  void draw( )
  {
    fill( col );
    rect( x, y, edge, edge );
  }
}
```

The idea here is to capture, or encapsulate, or code everything we need to draw a square within a single programming entity that we call a class. In this example the class is named, Square. Now we can declare references to objects of this

Square class. This is the main idea of Object Oriented Programming. Let's pursue this idea and create a Circle class.

```
class Circle
{
  // fields (variables)
  float x, y, diameter, dX, dY;
  color col;
  // functions
  void setup( )
  {
     x = random( 10, width/3 );
     y = random( 10, height/3 );
     diameter= random( 50, width/10 );
     dX = random( 2, 10 );
     dY = random( 2, 10 );
     col = color( random( 255), random( 255), random( 255 ) );
  }
  void move( )
  {
     x += dX;
     if (x + diameter> width || x < 0 )
     {
        dX = -dX ;
     }

     y += dY;
     if (y + diameter> height || y < 0 )
     {
       dY = -dY ;
     }
  }
  void draw( )
  {
     fill( col );
     ellipse( x, y, diameter, diameter);
   }
}
```

This will work just like the Square class… but … wait a mo…
Let's look at these two classes side-by-side:

```
class Square
{
  // fields or variables
  float x, y, edge, dX, dY;
  color col;
```

```
class Circle
{
  // fields (variables)
  float x, y, diameter, dX, dY;
  color col;
  void setup( )
  {
     x = random( 10, 500 );
     y = random( 10, 500 );
     edge = random( 50, 200 );
     dX = random( 2, 10 );
     dY = random( 2, 10 );
     col = color( random( 255),
                  random( 255),
                  random( 255 ) );
  }
  void move( )
  {
     x += dX;
     if (x + edge > width || x < 0 )
     {
        dX = -dX ;
     }
     y += dY;
     if (y + edge > height || y < 0 )
     {
       dY = -dY ;
     }
  }
  void draw( )
  {
    fill( col );
    rect( x, y, edge, edge );
  }
}
```

```
  void setup( )
  {
     x = random( 10, width/3 );
     y = random( 10, height/3 );
     diameter= random( 50, width/10 );
     dX = random( 2, 10 );
     dY = random( 2, 10 );
     col = color( random( 255),
                  random( 255),
                  random( 255 ) );
  }
  void move( )
  {
     x += dX;
     if (x + diameter> width || x < 0 )
     {
        dX = -dX ;
     }
     y += dY;
     if (y + diameter> height || y < 0 )
     {
       dY = -dY ;
     }
  }
  void draw( )
  {
     fill( col );
     ellipse( x, y, diameter, diameter);
   }
}
```

There are only two real differences between the two classes. One is a variable name: edge vs diameter and the other is one function call: rect vs ellipse. Even though there are some minor differences in the initilization of the class variables., this is a massive duplication of code and effort. Even with cutting and pasting, the program is longer than it needs to be.

We can solve this duplication of code and effort by using a feature of OOP called **inheritance**. With the correct syntax, we can create new classes that "**inherit**" everything in their parent's class (without the parent having to die).

Here is how we can do this. First, we have to make a compromise concerning the variable names. We could use edge for the circle or diameter for the square. Since both are dimensions, we will replace the variable names edge and diameter with the name dimension.

Next, we have to define the parent class of the Square and Circle class. We will call this the Shape class. Here is the definition:

```
class Shape
{
  float x, y, dimension, dX, dY;
  color col;
  void setup( )
  {
     x = random( 10, width/3 );
     y = random( 10, height/3 );
     dimension= random( 50, width/10 );
     dX = random( 2, 10 );
     dY = random( 2, 10 );
     col = color( random( 255), random( 255), random( 255 ) );
  }
  void move( )
  {
     x += dX;
     if (x > width || x < 0 )
     {
        dX = -dX ;
     }
     y += dY;
     if (y  > height || y < 0 )
     {
       dY = -dY ;
     }
  }
  void draw( )
  {    // this function is intentionally empty
  }
}
```

Notice that the draw() function is empty. That is because there is no shape to draw. The idea of a shape is somewhat abstract. So we will make no effort to draw shape.

Now we bring on the inheritance. Here are the definitions of the classes Square and Circle. In the code below, they are defined as children of the parent class, Shape:

```
class Square extends Shape
{
  void draw( )
  {
    fill( col );
    rect( x, y, dimension, dimension);
  }
}
```

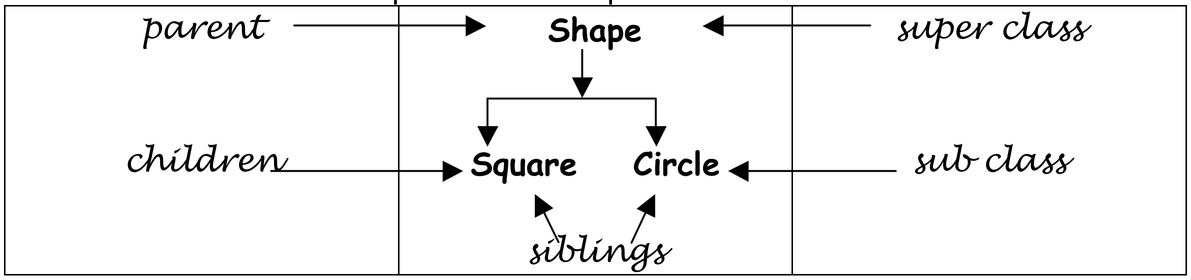
```
class Circle extends Shape
{
  void draw( )
 {
    fill( col );
    ellipse( x, y, dimension, dimension);
 }
}
```

The correct syntax is shown in blue in both class definitions. The word extends tells Processing that everything in the class Shape can be used here. The variables x, y, dimension and col are declared and initialized in the parent class, Shape. The definitions of setup( ) and move( ) are also located in the parent class, Shape. When our code tells a Circle object or a Square object to do something, Processing looks at the definitions of Circle or Shape classes to see what to do. If it does not find the answer there, it looks in the parent class, Shape for instructions. The only thing needed in the Circle and Square classes are instructions on how to draw the actual circle or square. This is done in the definition of the draw( ) functions.

Inheritance allows us to take advantage of code already written and that is (hopefully) tested and trusted to be free of errors.

There is some jargon associated with OOP and this inheritance stuff.

One is the diagram we use to show the inheritance relationships. It is usually in the form of a tree with the parent at the top of the tree.\



Terms used in describing the use of inheritance are shown above. The use of inheritance is often stated in terms of "**extending**" the Shape class to define the Square and Circle classes. Another phrase is that a programmer is "**sub-classing**" the Shape class to define the Square and Circle classes.

This works fine with single references or an array of references. The code on the next page works properly:

```
Shape [ ] array;
void setup( )
{
   size( 600, 600 );
   array = new Shape[ 100 ];
   initArray( );
   background( 0 );
}
void draw( )
{
   fill( 0 );
   rect( 0, 0, width, height );
   moveAndDrawAll( );
}
void moveAndDrawAll( )
{
  for ( int i = 0; i < array.length; i++)
  {
     array[i].move();
     array[i].draw( );
   }
}
```

```
void initArray( )
{
   for ( int i = 0; i < array.length; i++)
  {
      float randomNumber = random( 2 );
      if ( randomNumber < 1 )
      {
       array[i] = new Square( );
      }
      else
      {
       array[i] = new Circle( );
      }

      array[i].setup();
    }
}
```

All elements in an array in Processing have to be the same type. We cannot mix int and float values in the same array. This means that we cannot mix Square and Circle object references in the same array. BUT, we can declare an array of references to their parent type, Shape. This is perfectly legal. However, we do not declare any actual Shape references. When it is time to new the references in the array (shown in purple), this code uses a random value to choose between a Square or Circle object reference.