Classes #1

These notes discuss the class code cc14. You should have that code open as you read through this.

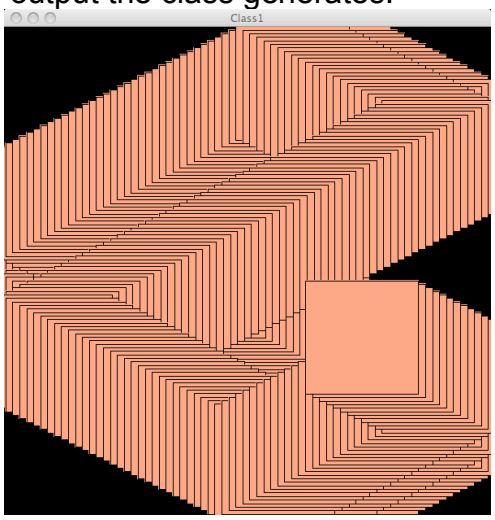
This set and the next two sets of notes concern the idea of classes, objects, and OOP. You need to have a reasonably clear idea what these three terms mean. You are responsible for the ideas presented in these notes. We will look at a lot of code in the notes and in class but the use of this code is voluntary on your part. You should explore these ideas if you feel they will be useful in your work.

We will concentrate these notes on looking at a class – how it is coded and how it is used. Open the files in the folder Class1.

To use a class in our code we need two files:

- the "client" file that "hires" the class to do some work. This is same the file we have been writing for 11 weeks. It has the setup() and draw() functions as well as keyPressed() and mousePressed() if needed.
- the "server" file that provides the needed service. For this code example the server has the variables and functions to draw and move a colored square.

Here is a sample of the output the class generates:



The next page has the code for the Client file, Class1.pde:

// Class1	This file/program is the client. It "hires" the Square class
Square s;	To use a class we must declare a reference to an object of the class in the way we declare a reference to an array. Since s is a reference but no memory has been allocated, its value at this time is null. *See Figure A below.*

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

s = new Square( );	Also similar to arrays, we have to "new" the class. This "newing" allocates memory for the variables needed by the class. This "newing" makes what we call the object, or in this case the Square object. This Square object is referenced by **s**. All of the variables in the Square object are initialized by Java to zero or false or the null character or null depending on the type of the variable. *See Figure B below.*
s.setup( );	Since all of the variables in the Square object must have values that are useful, we must call a function to do that. For now, we are going to use a setup( ) function. [1] *See Figure C below.*
background(0); }	
void draw( ) {	

s.move( );
s.draw( );
}
	In order to use the object we must call its functions. Since the functions are in another file (or class) we have to use the possessive syntax just as we did with arrays when we needed the value of the length of the array. The **.** is the syntax for possession. so we call the **move( )** and **draw( )** function "owned" by the Square object that **s** is referencing.

[1] This is not the "proper" way to initialize the variables in the object. The "proper" way is to use a special kind of function called a constructor. Shiffman explains this in the book and we will look at constructors soon.

Here are the figure referenced in the code on the previous page;

figure #A **The Square reference, s is null**

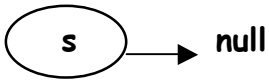


figure #B **The Square reference, s has been newed. All of the variables are initialized to zero.**

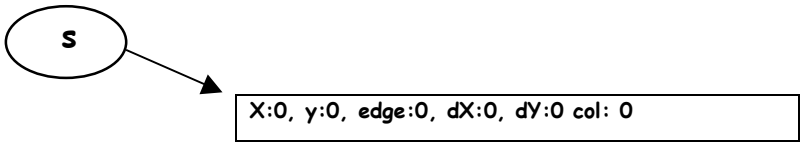
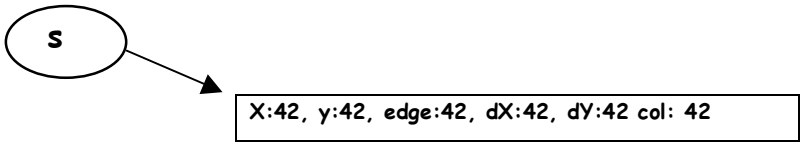


figure #C **The setup( ) function has been called. All of the variables now have random values. These random values are shown below as 42.**



We have to provide a definition of the class Square just like we have to provide definitions of functions that we need but but are not defined in the API.

The syntax for defining a class is much simpler than the syntax for defining a function. The next page shows the definition of the class, Square which is in the file, Square.pde.

```
class Square
{

// fields or variables
  float x, y, edge, dX, dY;
  color col;


 // functions
  void setup( )
  {
     x = random( 10, 600 );
     y = random( 10, 600 );
     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 ;
     }
  }
```

To define a class, we start with the word, class followed by the name of the class. The name of the file containing the class must be the same as the class name. The name of this file is Square.pde.

The next syntax requirement is an open brace. We must also put a corresponding closing brace at the end of the file as the last character in the file.

The rest looks like the code we have been writing since day 2.

We list the global variables (often called fields).

And we define any functions we need. Here is our definition of the setup( ) function which initializes the variables. In this example, they are initialized to random values.

If we wanted to designate the values of some or all of the variables, we would use parameters/argument in the call and definition of the setup( ) function.

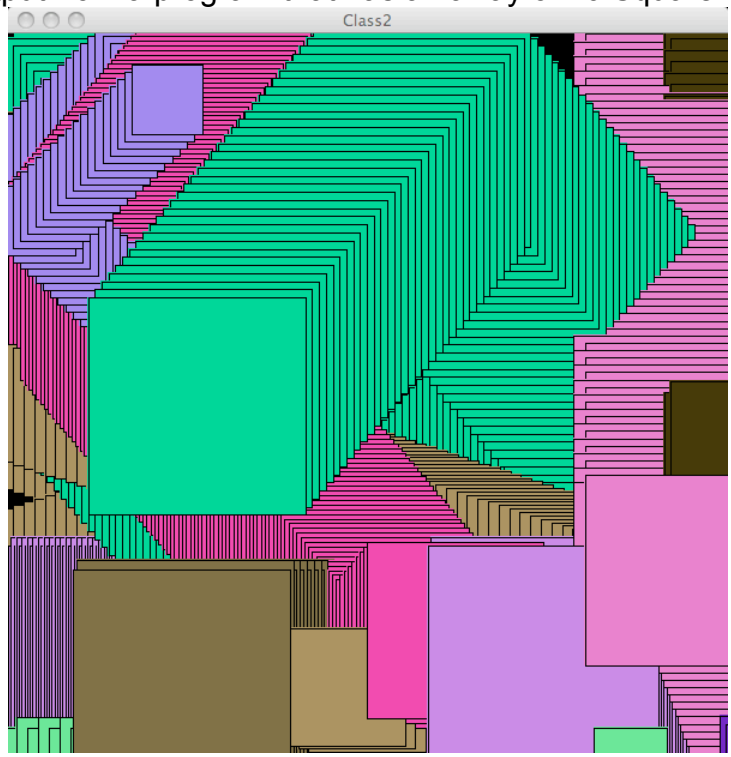
Here is the definition of the move function. There is nothing new here.

void draw( )
{
fill(col);
rect(x, y, edge, edge);
}
}
	Here is the definition of the function draw. Again, nothing is new here.
	
	This is the closing brace of the class.

That is all there is to defining and using a class. Syntactically is it much simpler than defining a function. The main problem for most novice programmers is recognizing the advantages of using a class and then designing the class. By designing the class we mean to determine what variables are needed and what functions must be defined. In this example, it is reasonably straightforward. But, that is not always the situation.

An Array of Object References

You should open the files in the folder Class2. We can build an array of object references just as we would build an array of int or an array of String references. Here is the output from a program that has an array of 10 Square references:



The next page shows the code in the file Class2.pde :

`final int MAX = 10;`	This is a constant because of the word, final. We use this to set the size or dimension the array.
`Square [ ] array;`	This declares a reference to the array of Square references. Right now the reference is null. *See figure #1 below*

`void setup( )`
`{`
`  size( 600, 600 );`
	This is the usual setup() function that Processing calls.

`  array = new Square[ MAX ];`	This line allocates enough memory for 10 Square objects and assigns s to reference the array. Each reference in the array is null. *See figure #2 below*
`  initArray( );`	Since the 10 individual Square references are null, we must new each one of them. We do this in a function called initArray( ).
`  background( 0 );` `}`	
`void initArray( )` `{`	This function initializes the Square objects referenced by the array's elements.
`  for ( int i = 0; i < array.length; i++)` `  {`	It traverses the array and
`    array[i] = new Square( );`	- news each element of the array creating a Square object. *See figure #3 below*

`    array[i].setup();`
`  }`
`}`
	- tells each element to execute its setup() function to initialize its variables. *See Figure #4 below*

`void draw( )`
`{`
`  moveAndDrawAll( );`
`}`
	To keep the draw( ) function "clean" we call a function to move and draw the squares.

void moveAndDrawAll( )
{
for (int i = 0; i < array.length; i++)
{
array[i].move();
array[i].draw( );
}
}
 	
void keyPressed( )
{
initArray( );
}
	This function :
	
	traverses the array and
	
	- tells each Square object to move
	- tells each Square object to draw itself
	
	A key press will initialize the array of Square objects.

Here is the array referenced in the code;

figure #1 array is null

figure #2 array has been newd with ten references to Squares. The ten references are null

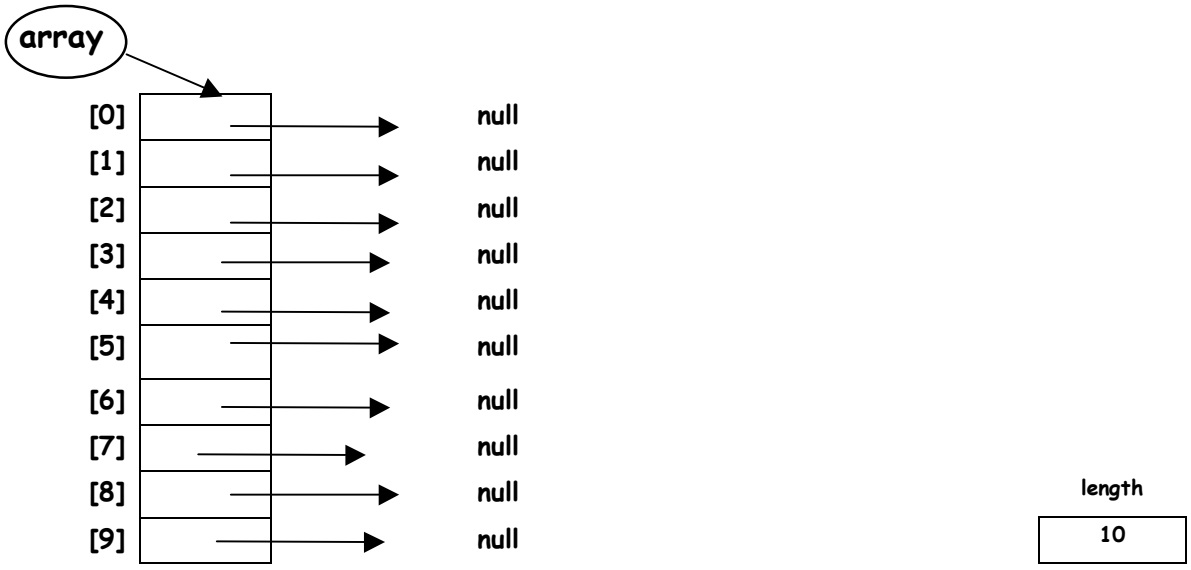


figure #3 The array has been traversed and each Square reference has been newd. The values of the variables in each Square object are set to zero.

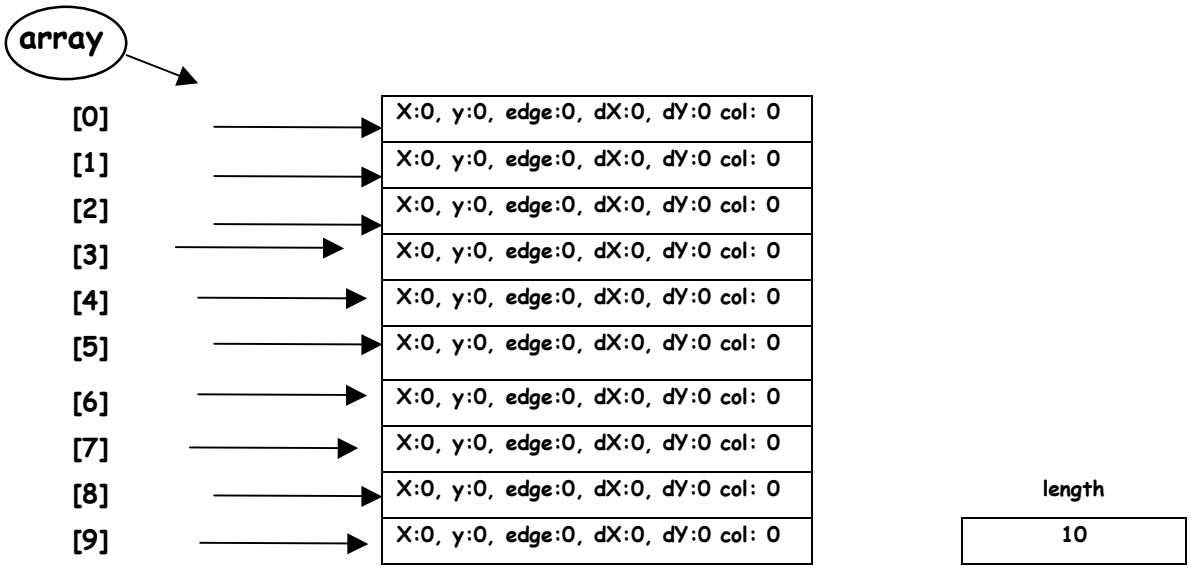


Figure #4 The array has been traversed and setup() called for each Square object. The variables now have random values. In this diagram these random values are all the same and shown as 42. In the program, these values would be different.