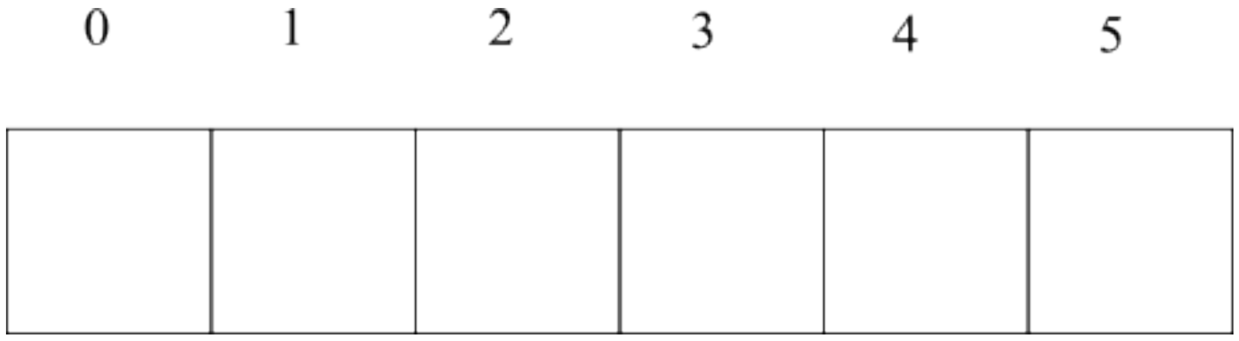
Processing Notes

Chapter 14: Arrays

So far when ever we want to manage more than a few objects things get messy. Even when we started using classes, handling a lot of objects is messy. Consider the examples of Chapter 13: if we want to handle 50 MovingRects or 50 MovingSprites we need to write a lot of code! Much better than not using classes, but still a lot of code. Programming languages allow us to bundle many objects together in a **data structure** to more effeciently store and manipulate objects. There are many types of data structures, and as you become a computer scientist you will learn how to create them and use them effeciently. On of the most elementary data structures is the **array.** An array can be thought of as a long box with cubby-holes to store objects inside:



This is a pictorial representation of an array of 6 elements. In most programming languages, including Processing and Java, arrays start at location 0. Thus, and array with 6 elements has element0 .. element5. An array can be declared to hold elements of any given basic type (int, char, float, boolean, etc) or objects of any defined class type. Thus, we could create an array of MovingSprites if we want to.

Consider the following code:

```
int NumElements = 5 ;
int[] theNums= new int[NumElements] ;

theNums[0] = 1 ;
theNums[1] = 2 ;
theNums[2] = 3 ;
theNums[3] = 4 ;
```

```
theNums[4] = 5 ;

for (int i = 0 ; i < NumElements ; i++)
  println(theNums[i]) ;

for (int i = 0 ; i < NumElements ; i++)
{
  theNums[i] = theNums[i] * 2 ;
}

for (int i = 0 ; i < NumElements ; i++)
  println(theNums[i]) ;
```

The second line declares and creates an array name “theNums” that holds 5 integers. The expression “int[]” is declaring the type of theNums to be an array of integers. The “new” command say create space for a new array, and the “int[NumElements]” specifies it should be space to hold 5 integers.

The next 5 lines of code put the values 1..5 into array elements 0..4.

The first for-loops prints out the elements. Note, to access element 2 we can simple say:

```
theNums[2]
```

In the for loop we say “theNums[i]” where the “i” is replace with the current value for “i”, hence, the loop executes 5 times and prints out theNums[0], theNums[1], .. theNums[4].

In the next for-loop the current value of each element is replaced with the current value times 2 (i.e. the contents of each array element are doubled). In the last loop the elements are printed out again.

Note, the way we assigned the values to the array was not very efficient. If we had an array with 100 elements we would need 100 lines of code. The following would be more effecient:

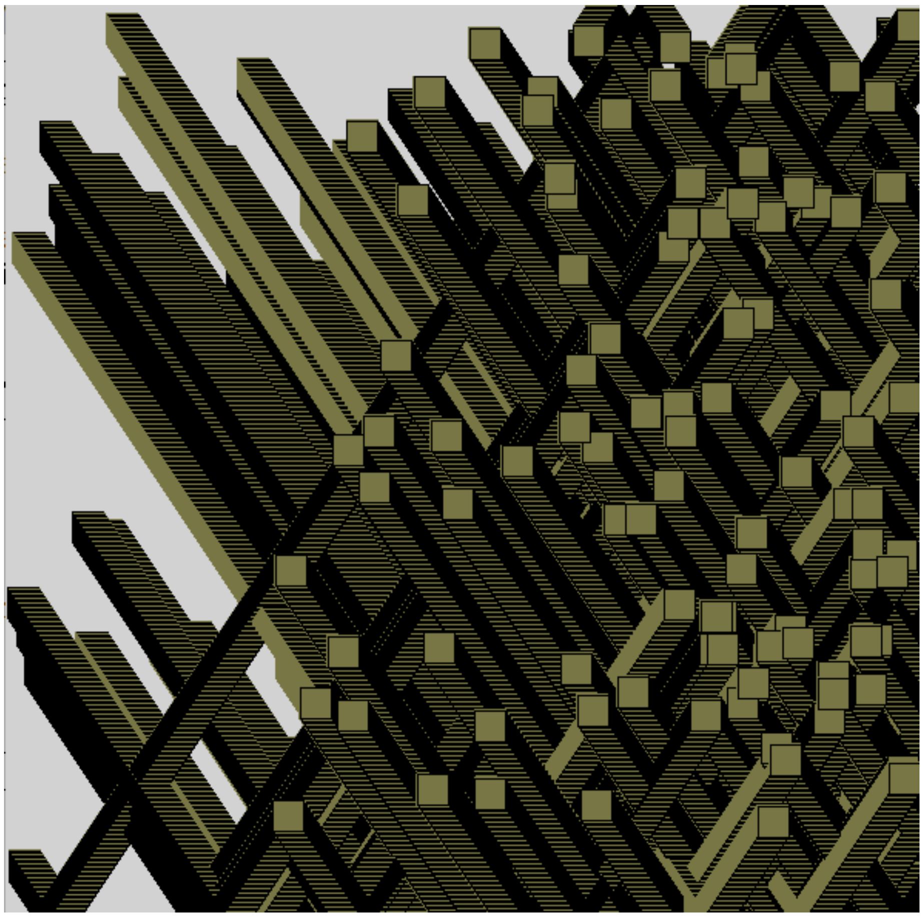
```
for (int i = 0 ; i < NumElements ; i++)
{
  theNums[i] = i ;
}
```

In the following example an array of MovingRectangle objects is created and use:

```
int NumRects = 100 ;
MovingRectangle[] theRects = new MovingRectangle[NumRects] ;

void setup()
{
  size(600,600) ;
  MovingRectangle aMR ;
  for (int i = 0 ; i < NumRects ; i++)
  {
    aMR = new MovingRectangle( random(width-20), random(height-20),
                        20,20, 1, 1.5, 100,100,50) ;
    theRects[i] = aMR ;

  }
}

void draw()
{
  for (int i = 0 ; i < NumRects ; i++)
  {
    theRects[i].updateLocation() ;
    theRects[i].drawRect() ;
  }
}
```

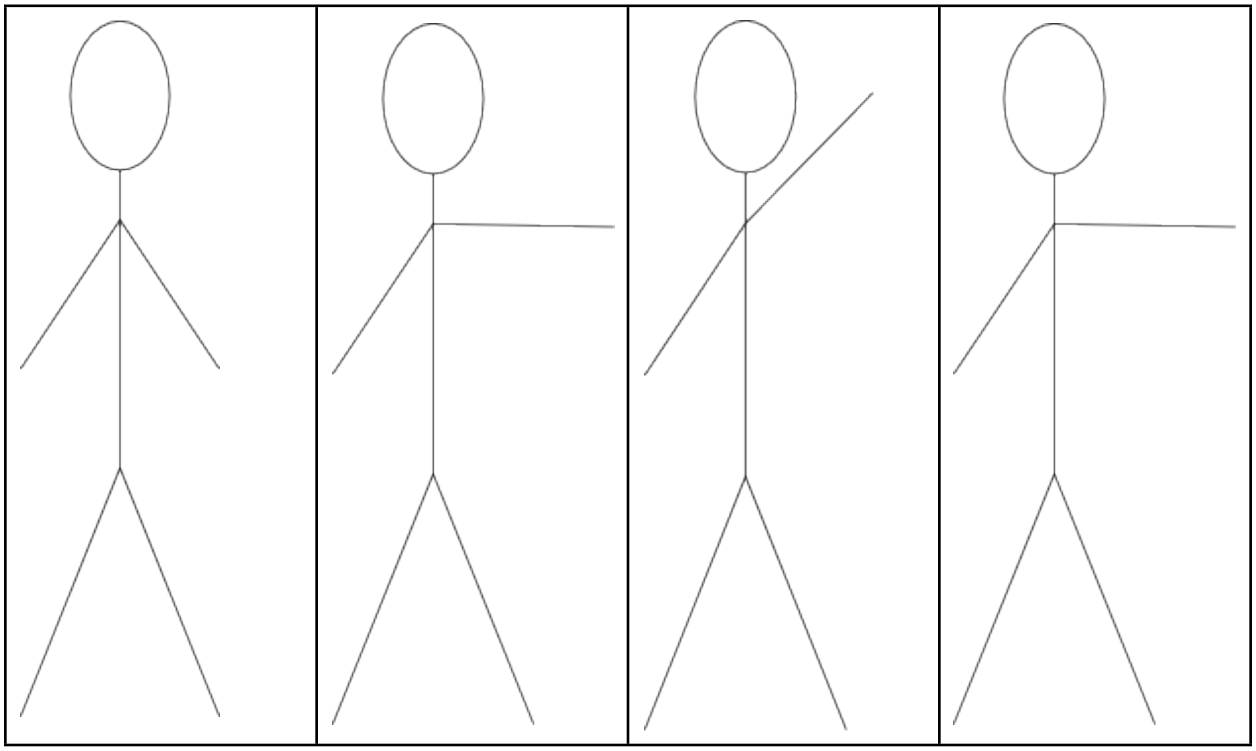
Inside the setup( ) function the for loop creates 100 MovingRectangle objects and puts one into each element of the array named “theRects”. In the draw method a for loop loops through each element and calls the updateLocation( ) and drawRect( ) method on each object in the array. Without using an array we would need 100 MovingRectangle variables and the draw method would need 200 lines of code.

Because there is no background( ) command in the draw( ) function we get the ghosting effect as the rectangles move about. After a short while the screen for the above example looks like:



If you run the code you will find it is much more interesting as an animation.

Arrays can be used for many purposes. Whenever you want to hold a bunch of objects in a container they come in handy. Another example is using an array to hold images for an animation. Lets assume you have four images named stick1.png .. stick4.png, where each image is a different part of an animation cycle. For example the following four images played in succession and then looped will make the stickman waive:



Code for doing this is:

```
int NumImages = 4 ;
int currentImageNum ;
PImage[] theImages = new PImage[NumImages] ;

void setup()
{
  size(600,600) ;
  theImages[0] = loadImage("stick1.png")  ;
  theImages[1] = loadImage("stick2.png")  ;
  theImages[2] = loadImage("stick3.png")  ;
  theImages[3] = loadImage("stick2.png")  ;
  currentImageNum = 0 ;
  frameRate(8 ) ;
}

void draw()
{
  background(255) ;
  image(theImages[currentImageNum],40,20) ;
  currentImageNum++ ;
  if (currentImageNum >= NumImages)
    currentImageNum = 0 ;
}
```

The array “theImages” is created to hold elements of type PImage. Then, in setup( ) the four images are loaded into elements 0, 1, 2, and 3. A global variable named “currentImageNum” is used to specify which frame of the animation is next to be shown. The draw method shows the current image bay calling:

```
image( theImages[currentImageNum], 40, 20) ;
```

and then increments the variable currentImageNum so that on the next call to draw( ) the next frame of the animation cycle will be shown.

EXERCISE 14A
Create an animation of a hundred of more small moving rectangles that start in the center and explode out towards the boundaries. When they make it to the boundary they are restarted in the center so we have a “fountain” of rectangles emitting from the center. See the quicktime video on the class website for an example.

EXERCISE 14B
Modify your AnimatedMovingSprite from exercise 13B. If you did not do 13B just do this exercise now. Use an array to hold the different images and make the drawSprite( ) method cycle through these images. Show your animate sprite both moving around the screen and being animate. You can use the stick figures above if you want.