Notes 4: Using Arrays to manage lots-o-objects and MovieClips

So now you have two balls moving across the screen. What if you want 20 of these little bad-boys making tracks across the screen? I guess you could have 20 variables and 20 lines of code each time you want to check or modify something, but what if you want 100, or 200? Say you want to make it snow/rain on your screen and you want hundreds of snow flakes at once. You of course can practically hear the words “for loop” screaming at you, right? But even so, how do you name and then access all those variables from within a for loop? An array will do it. You can create an array, fill it with MovieClip objects (say the objects are balls, or rain drops), and then use a for loop to loop through the objects and update the (x,y) locations as need be.

Before going further, whenever you are in doubt, google to find the answer. For example, if you google “actionscript array class” you will get all sorts of good info.

Okay, first we need to understand how actionscript arrays work. First, you can use them pretty much the same as in C/C++. Consider the following code.

```actionscript
// Some simple array examples
var maxElements:Number = 10;
var theArray:Array;
theArray = new Array(maxElements);

theArray[0] = 0;
theArray[1] = 2;
theArray[2] = 4;
theArray[3] = 6;
theArray[4] = 8;
theArray[5] = 10;
theArray[6] = 12;
theArray[7] = 14;
theArray[8] = 16;
theArray[9] = 18;

trace("theArray[8] = " + theArray[8]);

trace("contents of array:");
for (var i:Number = 0 ; i < maxElements ; i++) {
    trace(theArray[i]);
}

for (var i:Number = 0 ; i < maxElements ; i++) {
    theArray[i] = 0;
}

trace("contents of array:");
for (var i:Number = 0 ; i < maxElements ; i++) {
    trace(theArray[i]);
}
```

```
for (var i:Number = 0 ; i < maxElements ; i++) {
    theArray[i] = i*4 ;
}

trace("\ncontents of array:"");
for (var i:Number = 0 ; i < maxElements ; i++) {
    trace(theArray[i]) ;
}

The above code (a3_1.fla) shows how to create an array of ten elements, assign values to the array one element at a time, print out the contents in a for loop, and assign contents within a for loop. So far so good, looks like C/C++.

But now it gets weird. Check out the code below.

a3_2.fla :

// Here are several ways to create arrays

// use the OO constructor notation, create an array with 20 empty elements
var a1:Array = new Array(20) ;
var a2:Array = [20] ;  // create and array with 20 elements?  NO, that is not what this does!
var a3:Array = ["dog","cat","bat","pig"] ;   // create and initialize with 4 string elements
var a4:Array = [0,2,4,6,8,10] ;   // create and initialize with 6 numeric elements
var a5:Array = [] ;  // create but don’t specify how many elements
var a6:Array = new Array() ;   // create using constructor notation not specifying num elements

// An array is an AS object, it has the member length
trace("a1.length = " + a1.length) ;
trace("a2.length = " + a2.length) ;
trace("a3.length = " + a3.length) ;
trace("a4.length = " + a4.length) ;
trace("a5.length = " + a5.length) ;
trace("a6.length = " + a6.length) ;

// When looping through an entire array it usually is a good idea to use array.length
trace("\nContents of array a1 are:"");
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":t" + a1[i]) ; } 
trace("\nContents of array a2 are:"");
for (var i:Number = 0 ; i < a2.length ; i++) { trace(i + ":t" + a2[i]) ; }
trace("\nContents of array a3 are:"");
for (var i:Number = 0 ; i < a3.length ; i++) { trace(i + ":t" + a3[i]) ; }
trace("\nContents of array a4 are:"");
for (var i:Number = 0 ; i < a4.length ; i++) { trace(i + ":t" + a4[i]) ; }
trace("\nContents of array a5 are:"");
for (var i:Number = 0 ; i < a5.length ; i++) { trace(i + ":t" + a5[i]) ; }
trace("\nContents of array a6 are:"");
for (var i:Number = 0 ; i < a6.length ; i++) { trace(i + ":t" + a6[i]) ; } 

trace("Again, lengths of a3,a4 before adding new elements:");
trace("a3.length = " + a3.length) ;
trace("a4.length = " + a4.length) ;

// We can add elements anyplace in an array, it automatically resizes:
a3[8] = "penguin" ;
trace("After adding elements beyond the end of arrays a3 and a4:");
trace("\nContents of array a3, whose length now is " + a3.length + ", are:");
for (var i:Number = 0 ; i < a3.length ; i++) { trace(i + ":" + a3[i]) ; }
trace("\nContents of array a4, whose length now is " + a4.length + ", are:");
for (var i:Number = 0 ; i < a4.length ; i++) { trace(i + ":" + a4[i]) ; }

The strange thing in the code above, compared to C/C++, is you can add elements beyond the end of the array it ActionScript is perfectly happy. It just creates space for the new element and also space between the previous last element and the new last element, filling the spaces in with “undefined”. Run the code above, understand it, play with it. Note, when looping over arrays it is often best to use arrayName.length as the loop upper bound.

In ActionScript arrays are a pre-defined class. They have members (the elements of course, but also “length” which is the number of elements currently in the array) and methods including:

- sort: sort the array
- push: push an element onto the end (top) of the array
- pop: pop (remove) an element from the end (top) of the array
- shift: shift (add) an element onto the front (bottom) of the array
- unshift: remove an element from the front (bottom) of the array
- reverse: reverse the order of the elements within the array
- concat: returns an array that is the concatenation of the arguments
- toString
- join
- slice
- splice

We first talk about sort since sorting arrays could well be something you need to do. Consider the following code. The comments are self-explanatory.

**a3_3.fla**

```javascript
var a1:Array = ["pig","cat","monkey","donkey","tiger","bat"];
trace("\nbefore calling a1.sort, a1 contains: \n" + a1) ;

// now sort and print out a1
a1.sort();
trace("\nAfter calling a1.sort, a1 contains: \n" + a1) ;

// The above worked great, how about trying it out on numbers
var a3:Array = [13, 52, 33, 2, 25, 14, 3, 77, 8];
trace("\nbefore calling a3.sort, a3 contains: " + newline + a3) ;

// sort and print out a3:
a3.sort();
trace("\nafter calling a3.sort, a3 contains: " + newline + a3) ;
```
// It was alphabetical, NOT numeric!
// To get numerical we need to define a sort function and specify it in the sort() call

function sortByNumber(a, b) {
    return (a > b);
}

// now lets call sort with sortByNumber as an argument
a3.sort(sortByNumber);
trace("after calling a3.sort(sortByNumber)" + newline + a3);

The next chunk of code contains example usage of many of the other methods. The comments are self-explanatory.

a3_4.fla:

var a1:Array = ["dog","cat","bat","pig"] ;  // create and initialize with 4 string elements

// first print out the contents of a1
trace("a1.length = " + a1.length + ", contents are:");
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":" + a1[i]) ; }

// push the string "cow" on the array and print out the array
a1.push("cow");
trace("After a1.push("cow"), a1.length = " + a1.length + ", contents are:");
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":" + a1[i]) ; }

// pop the last element from the array and print out the array
var temp:Object = a1.pop();
trace("After temp = a1.pop(), temp = " + temp);

trace("After a1.pop(), a1.length = " + a1.length + ", contents are:");
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":" + a1[i]) ; }

// reverse the array and print it out
a1.reverse();
trace("After a1.reverse(), a1.length = " + a1.length + ", contents are:");
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":" + a1[i]) ; }

// the following splices in (inserts) the three strings start at element 2, it moves
// the elements at 2 and greater down.  The zero is the number of elements
// that are REPLACED by the new string
a1.splice(2,0,"coyote","moose","duck");
trace("After a1.splice(2,0,"coyote","moose","duck") ; a1.length = " + a1.length + ", contents are:");
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":" + a1[i]) ; }

// Say I hate ducks.  One can use splice to insert nothing
// in place of an existing entry.  This allows you to delete from
// the middle of an array.
// The following bit of code will remove ducks from the array:
var i:Number = 0 ;
while ( (i < a1.length) && (a1[i] != "duck") )
    i++
if (a1[i] == "duck")
    trace("duck at " + i);
a1.splice(i,1) ;
trace("\nAfter splicing out the duck, a1.length = "+a1.length + ", contents are:"+ ) ;
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":" + a1[i]) ; }

// Lets reinitialize our arrays and print them out
a1 = ["a","b","c","d"] ;
a2 = [3,6,9] ;
trace("After a1 = ["a","b","c","d"] ; a2 = [3,6,9] ; : ") ;
trace("a1.length = "+ a1.length + ", contents are:"+ ) ;
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":" + a1[i]) ; }
trace("a1.length = "+ a2.length + ", contents are:"+ ) ;
for (var i:Number = 0 ; i < a2.length ; i++) { trace(i + ":" + a2[i]) ; }

// The concat() method returns the array with the argument arrays concatenated onto the end
a1 = a1.concat(a2) ;
trace("\nAfter a1.concat(a2), a1.length = "+a1.length + ", contents are:"+ ) ;
for (var i:Number = 0 ; i < a1.length ; i++) { trace(i + ":" + a1[i]) ; }

a2 = a2.concat(a2,a1) ;
trace("\nAfter a2.concat(a2,a1), a2.length = "+a2.length + ", contents are:"+ ) ;
for (var i:Number = 0 ; i < a2.length ; i++) { trace(i + ":" + a2[i]) ; }

// The toString() method ties together all the elements into one long string where
// each element is comma seperated.
var temp2:String = a1.toString() ;
trace("After temp2 = a1.toString(), temp2 = " + temp2) ;

// The join() method allows you to specify what char to join the elements
// together with, the empty string "" means there are no seperating characters
var temp3:String = a1.join("\") ;
trace("\nAfter temp3 = a1.join("\") , temp3 = " + temp3) ;
var temp4:String = a1.join(\"\") ;
trace("\nAfter temp4 = a1.join(\"\") , temp4 = " + temp4) ;
var temp5:String = a1.join("PIG") ;
trace("\nAfter temp4 = a1.join("PIG") , temp5 = " + temp5) ;

Note, the toString() and join() methods should leave you wondering how to manipulate
the strings, google for "actionscript string class", the class has many methods to aid string
manipulation.

The above gave you a crash course in the actionscript array class. There is more you can
learn about arrays, in particular 2D arrays (which are just arrays of arrays), but the above
should be enough to get you going.

Now, how can we use arrays to create our 200 raindrops? Here is a snippet of code that
will do what you want.

a2_5.fla :
var balls:Array;
balls = new Array(20);

for (var i:Number = 0; i < 20; i++) {
    balls[i] = attachMovie("ball", "b"+i, i);
    // the above line is slick, each time executed the linkage var is the same, "ball",
    // but we want the internal identifier to be unique as well as the level of the
    // new instance, hence, we use ("b"+i) and (i) in the attach movie statement
}

for (var i:Number = 0; i < 20; i++) {
    // Set the initial (_x,_y) coordinates for each ball.
    // The following creates two columns of balls.
    if ((i%2) == 0)
        balls[i]._x = 10;
    else
        balls[i]._x = 100;
    balls[i]._y = 40*i;
}

onEnterFrame = function () {
    for (var i:Number = 0; i < 20; i++) {
        balls[i]._x += 3;
        balls[i]._y -= 3;
    }
}

The first two lines declare “balls” to be an array, and then create an array of 20 elements. Obviously you can put in any number of elements. Balls is now an array of 20 elements, but we have not specified the element types. Here we are using the flash style of just saying they are “objects”, and then we assign instances of movieClips to the elements with the attachMovieClip method in the for loop. In the second for loop with set the _x and _y members of each movieClip in the array. Then, in the onEnterFrame function we loop through and update the _x and _y members of each MovieClip.

The above shows how you can create arrays of MovieClips and then move them around. To make snow or rain you probably want to draw something other than a ball. It is easier to work with a larger object (the object you draw and put in the library) and then scale it down. You also might want to rotate the object on screen to give it a “swirling” look. Check out the following code:

**a3_6.fla**:

```actionscript
var numFlakes:Number = 40;
var snowflakes:Array;
snowflakes = new Array(numFlakes);

for (var i:Number = 0; i < numFlakes; i++) {
    snowflakes[i] = attachMovie("snowflake", "sf"+i, i);
    // The next two lines SCALE the object down to 5% of the size
```
snowflakes[i]._xscale = 5 ;
snowflakes[i]._yscale = 5 ;
}

for (var i:Number = 0 ; i < numFlakes ; i++) {
  if ((i%2) == 0)
    snowflakes[i]._x = 10 ;
  else
    snowflakes[i]._x = 100 ;
snowflakes[i]._y = 20*i ;
}

onEnterFrame = function () {
  for (var i:Number = 0 ; i < numFlakes ; i++) {
    snowflakes[i]._x += 3 ;
snowflakes[i]._y -= 3 ;
    // the following line ROTATES the object by 5 degrees
    snowflakes[i]._rotation += 5 ;
  }
}

There are few differences comparing the above code with the proceeding file. First, the use of _xscale and _yscale to make the image of the “snowflakes” smaller than the drawn library image. Second, the use of _rotation to make the snowflakes look like they are turning.

You now know enough flash and actionscript to create a scene with rain or snow. Go for it!

Stuff to come:

The Library
GoToAndPlay, GotoAndStop

Functions and Scope

Input and Output with dynamic text boxes and variables

Arrays

Objects/Classes

Arrays of Objects

Mouse Handlers (in each frame)

Keyboard Handlers (in each frame)

Listeners