CS 116 – Action Script Arrays

Elie Abi Chahine

Introduction

- In the previous chapter, we covered the String object, now it is time to cover the Array object including most of it’s attributes.
- Often in programming you’ll need to work with a set of items rather than a single object. For example, in a music player application, you might want to have a list of songs waiting to be played. You wouldn’t want to have to create a separate variable for each song on that list. It would be preferable to have all the Song objects together in a bundle, and be able to work with them as a group.
- That would be a great time to use an Array.

Introduction

- An array is a programming element that acts as a container for a set of items, such as a list of songs.
- Most commonly all the items in an array are instances of the same class, but that is not a requirement in ActionScript.
- The individual items in an array are known as the array's elements .
- You can think of an array as a file drawer for variables.
  - Variables can be added as elements in the array, which is like placing a folder into the file drawer.
  - You can work with the array as a single variable (like carrying the whole drawer to a different location).
  - You can work with the variables as a group (like flipping through the folders one by one searching for a piece of information).
  - You can also access them individually (like opening the drawer and selecting a single folder).

Indexed Array

- The Array class is also known as Indexed Array class.
- Indexed arrays store a series of one or more values organized such that each value can be accessed using an index (unsigned integer value).
- The first index is always the number 0, and the index increments by 1 for each subsequent element added to the array.
- Indexed arrays use an unsigned 32-bit integer for the index number. The maximum size of an indexed array is 4,294,967,295. An attempt to create an array that is larger than the maximum size results in a run-time error.
- To access an individual element of an indexed array, you use the array access operator [ ] to specify the index position of the element you wish to access

Creating Arrays

- You have multiple ways to create an array:

```
var arr1:Array = [1,2,3,4,5];

var arr2:Array = ["John", 12 , "Jane", 15 , "David“ , 17];

var arr3:Array = arr2;
```

- Now using the constructor:

```
var arr1:Array = new Array (); /* creates an empty array, length = 0 */

var arr2:Array = new Array (3); /* creates an empty array, length = 3 */

var arr3:Array = new Array("John", "Jane", "David");
```

Inserting Elements

- The most basic way to add an element to an indexed array is to use the array access operator [ ].
- To set the value of an indexed array element, use the Array object name and index number on the left side of an assignment statement:

```
var arr1:Array = [1,2,3,4,5];

trace(arr1);   /* 1,2,3,4,5 */

arr1[3] = 17;

trace(arr1);   /* 1,2,3,17,5 */
```

- If the Array doesn't already have an element at that index, the index is created and the value is stored there. If a value exists at that index, the new value replaces the existing one.

Inserting Elements

- We also have methods inside the Array class that allow you to insert elements into an indexed array: ***push()*** , ***unshift()*** , and ***splice()***
- The ***push()*** method appends one or more elements to the end of an array.
- The ***unshift()*** method inserts one or more elements at the beginning of an array
- The ***splice()*** method will insert any number of items at a specified index in the array.

```
var aPlanets:Array = new Array();

aPlanets.push("Mars");   /* array contents: Mars */

aPlanets.unshift("Mercury");   /* array contents: Mercury , Mars */

aPlanets.splice(1, 0, "Venus", "Earth");
trace(aPlanets);   /* array contents: Mercury , Venus , Earth , Mars */
```

Retrieving Elements

- The simplest way to retrieve the value of an element from an indexed array is to use the array access operator [ ].

- To retrieve the value of an indexed array element, use the Array object name and index number on the right side of an assignment statement:

var aSongList:Array = new Array("Comfortably Numb" , "Kite" , "Marooned");
var sFavoriteSong:String = aSongList [2];

Removing Elements

- Three methods of the Array class allow you to remove elements: ***pop()*** , ***shift()*** and ***splice()***
- The ***pop()*** method removes the last element from the array.
- The ***shift()*** method removes the first element from the array.
- The **splice()** method, which can also be used to insert elements, removes an arbitrary number (specified by the user) of elements starting at the index number specified by the first argument sent to the method

```
var aOceans:Array = ["Victoria", "Pacific", "Aral", "Superior" , "Huron"];

aOceans.splice(2, 2); /* removing Aral and Superior */

aOceans.pop(); /* removes Huron */

aOceans.shift(); /* removes Victoria */

trace(aOceans); /* Pacific */
```

Replacing Elements

- How can we use the ***splice()*** method to replace data in our array?

```
var aOceans:Array = ["Victoria", "Pacific", "Aral", "Superior" , "Huron"];

aOceans.splice(2, 2, "Arctic", "Atlantic");  /* replaces Aral and Superior */

trace(aOceans);  /* Victoria,Pacific,Arctic,Atlantic,Huron */
```

Truncating Elements

- You can truncate an Array using an array’s length property.
- If you set the length property of an indexed array to a length that is less than the current length of the array, the array is truncated, removing any elements stored at index numbers higher than the new value of length minus 1.
- For example:

```
var aOceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];

aOceans.length = 2;

trace(aOceans);    /* Arctic,Pacific */
```

Sorting

- There are three methods that allow you to change the order of an indexed array: ***reverse()*** , ***sort()*** , and ***sortOn()***.
  - ***reverse()***:
    Changes the order of the elements so that the last element becomes the first element, the penultimate element becomes the second, and so on…
  - ***sort()***:
    Allows you to sort the Array's elements in a variety of predefined ways, such as alphabetical or numeric order. You can also specify a custom sorting algorithm.
  - ***sortOn()***:
    Allows you to sort objects that have one or more common properties, specifying the property or properties to use as the sort keys

```
var aOceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];

aOceans.length = 2;

trace(aOceans);    /* Arctic,Pacific */
```

Sorting

- The ***reverse()*** method:

```
var aOceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];

aOceans.reverse();

trace(aOceans);   /* Pacific,Indian,Atlantic,Arctic */
```

Sorting

- How about the ***sort()*** function?

<u>Small exercise:</u>

- ***Create an array containing the following values: 1 27 13 10 2 17 54 5***
- ***Try to sort it and output the result.***

Sorting

- Basic sorting with the ***sort()*** method:
- The ***sort()*** method rearranges the elements in an array using the default sort order .
- The default sort order has the following characteristics:
  - The sort is case-sensitive, which means that uppercase characters precede lowercase characters. For example, the letter D precedes the letter b.
  - The sort is ascending, which means that lower character codes (such as A) precede higher character codes (such as B).
  - The sort places identical values adjacent to each other but in no particular order.
  - The sort is string-based, which means that elements are converted to strings before they are compared (for example, 10 precedes 3 because the string "1" has a lower character code than the string "3" has).

Sorting

- You may find that you need to sort your Array without regard to case, or in descending order, or perhaps your array contains numbers that you want to sort numerically instead of alphabetically.

- The Array class's sort() method has an options parameter that allows you to alter each characteristic of the default sort order. The options are defined by a set of static constants in the Array class, as shown in the following list:

  - **Array.CASEINSENSITIVE :** This option makes the sort disregard case. For example, the lowercase letter b precedes the uppercase letter D.

  - **Array.DESCENDING :** This reverses the default ascending sort. For example, the letter B precedes the letter A.

  - **Array.UNIQUESORT :** This causes the sort to abort if two identical values are found.

  - **Array.NUMERIC:** This causes numerical sorting, so that 3 precedes 10.

Sorting

- The following example highlights some of these options. An Array named poets is created that is sorted using several different options.

```
var aPoets:Array = ["Blake", "collins", "Angelou", "Dante"];
aPoets.sort();  /* default sort */
trace(aPoets);  /* Angelou,Blake,Dante,collins */

aPoets.sort(Array.CASEINSENSITIVE);
trace(aPoets);   /* Angelou,Blake,collins,Dante */

aPoets.sort(Array.DESCENDING);
trace(aPoets);   /* collins,Dante,Blake,Angelou */

/* use two options */
aPoets.sort(Array.DESCENDING | Array.CASEINSENSITIVE);
trace(aPoets);   /* Dante,collins,Blake,Angelou */
```

Sorting

How can I sort without modifying the original array ?

- **Array.RETURNINDEXEDARRAY :**
  - This option directs the methods to return a new Array that reflects the sort and leaves the original Array unmodified. The Array returned by the methods is a simple Array of index numbers that reflects the new sort order and does not contain any elements from the original Array

```
var aLetters:Array = ["A", "Z" , "M" , "W"];

var aIndices:Array = aLetters.sort(Array.RETURNINDEXEDARRAY);
trace(aIndices); /* 0,2,3,1 */
trace(aLetters); /* A,Z,M,W */

for (var i:int = 0; i < aIndices.length; ++i)
{
   var index:int = aIndices[i];
   trace(aLetters[index]);
}
/* A M W Z */  /* Of course each one on a separate line */
```

Querying

Four methods of the Array class all query the array for information, but do not modify the array: ***concat()*** , ***join()*** , ***slice()*** , and ***toString()***

- ***concat()***:
  Takes a new array or list of elements as arguments and combines it with the existing array to create a new array

- ***slice():***
  Has two parameters, aptly named startIndex and an endIndex , and returns a new array containing a copy of the elements “sliced” from the existing array. The slice begins with the element at startIndex and ends with the element just before endIndex .
  That bears repeating: the element at endIndex is not included in the return value.

- ***join() & toString():***
  You can use them to query the array and return its contents as a string. If no parameters are used for the ***join()***method, the two methods behave identically.

  The ***join()*** method, unlike the ***toString()*** method, accepts a parameter named delimiter , which allows you to choose the symbol to use as a separator between each element in the returned string.

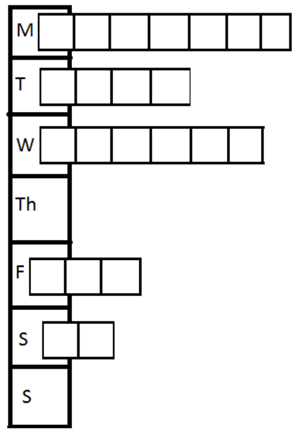
I’m sure at this point you can figure out how they work.

Multidimensional Arrays

- When we say multidimensional array, we just mean arrays inside arrays ☺ …
  Should be fun !!!

- Why would we ever want that??!!!

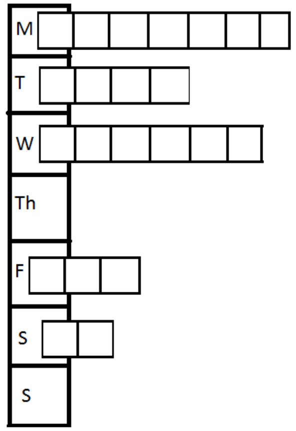
- Well imagine we want to store a separate list of tasks for each day of the week, you can create a multidimensional array with one element for each day of the week. Each element contains an array, which is the tasks array, that stores the list of tasks.



Multidimensional Arrays

Two dimensional Array:

- When you use two dimensional arrays, you can visualize the result as a table or spreadsheet. The elements of the first array represent the rows of the table, while the elements of the second array represent the columns



```
var aMasterTaskList:Array = new Array(7);

aMasterTaskList[0] = ["wash dishes", "take out trash"];
aMasterTaskList[1] = ["wash dishes", "pay bills"];
aMasterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
aMasterTaskList[3] = ["wash dishes"];
aMasterTaskList[4] = ["wash dishes", "clean house"];
aMasterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
aMasterTaskList[6] = ["mow lawn", "fix chair"];
```

Multidimensional Arrays

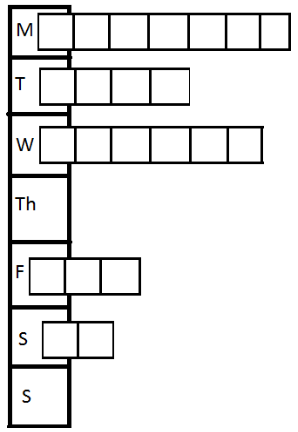
<u>Two dimensional Array:</u>

- You can use any Array constructor when creating an array inside another array

```
var aMasterTaskList:Array = new Array(7);

aMasterTaskList[0] = new Array();
aMasterTaskList[0].push("wash dishes");
aMasterTaskList[0].push("take out trash");

aMasterTaskList[1] = new Array("wash dishes", "pay bills");
    .
    .
    .
```

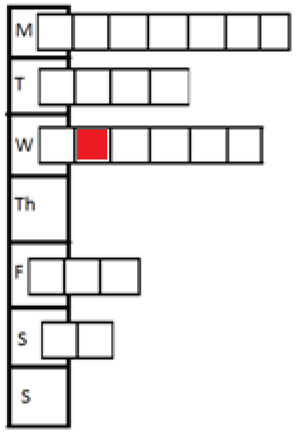


Multidimensional Arrays

Two dimensional Array:

- You can access individual items on any of the task lists using the array access operator [ ].
- The first set of brackets represents the day of the week, and the second set of brackets represents the task list for that day.
- For example, to retrieve the second task from Wednesday's list, first use index 2 for Wednesday, and then use index 1 for the second task in the list.

trace(aMasterTaskList[2][1]);



The End ☺