DigiPen
INSTITUTE OF TECHNOLOGY

CS 116 – Action Script
More On Arrays

Elie Abi Chahine

Introduction

- So I'm assuming by now, we all know what an Array object is and how to access methods and properties inside an object.
- Having said that, in this chapter I will cover some extra methods found in the Array class that we never used before, or at least I never covered.
- First topic will be on advanced ways to work with the ***sort()*** function by customizing it.
- Second, will be covering how to use the functions that require a callback function as parameter ***(every, filter, forEach, map, some)***.

More On Sorting

Customizing the sort function

- Previously, we covered the ***sort()*** function specifying to it if we wanted to sort in a Descending/Ascending/Numberic… way.
- Now how about if we want to customize our sort function!!??
- In addition to the basic sorting that's available for an Array object, you can also define a custom sorting rule.
- To define a custom sort, you write a custom sort function and pass it as an argument to the ***sort()*** method.

Example

```
function orderLastName(a_ , b_):int        /* This function will specify the rule */
{
   var aName1:Array = a_.split(" ");
   var aName2:Array = b_.split(" ");
   if (aName1[1] < aName2[1])
   {
      return -1;
   }
   else if (aName1[1] > aName2[1])
   {
      return 1;
   }
   else
   {
      return 0;
   }
}

var aNames:Array = new Array("John Smith", "Jane Doe", "Mike Jones");
trace(aNames);       /* output: John Smith,Jane Doe,Mike Jones */
names.sort(orderLastName);
trace(aNames);       /* output: Jane Doe,Mike Jones,John Smith */
```

Example Explanation

- The custom sort function ***orderLastName()*** uses the ***split()*** function as a way to access the last name from each element and use it for the comparison.
- The function identifier **orderLastName** is used as parameter when calling the ***sort()*** method on the names array.
- The sort function accepts two parameters, a and b , because it works on two array elements at a time.
- The sort function's return value indicates how the elements should be sorted:
  - A return value of -1 indicates that the first parameter, a , precedes the second parameter, b .
  - A return value of 1 indicates that the second parameter, b , precedes the first, a .
  - A return value of 0 indicates that the elements have equal sorting precedence.

Methods & Callback Functions

Methods & Callback Functions

- In the Array class, we have a lot of methods that take a callback function as parameter in order to work.

every(callback:Function, thisObject:* = null):Boolean

filter(callback:Function, thisObject:* = null):Array

forEach(callback:Function, thisObject:* = null):void

map(callback:Function, thisObject:* = null):Array

some(callback:Function, thisObject:* = null):Boolean

Callback Functions

- The callback function is the function that will run on each item in the array.
- This function can contain a simple comparison (for example, item < 20) or a more complex operation.
  - ***function callbackName(item_:*, index_:int, array_:Array):ReturnType;***
- The callback function is invoked with three arguments:
  - The value of an item.
  - The index of an item, which is the index of the object you are currently working with in the array.
  - The Array object, which is the array you are working with and that contains all the elements.
- The return type for callback functions will vary depending on the method you are using.

every

every(callback:Function, thisObject:* = null):Boolean

- Executes a test function on each item in the array until an item is reached that returns false for the specified callback function.
- So basically, the callback function is checking if each element inside the array satisfies the condition inside it. It will return true if it does, false if it doesn't
- Of course the callback function has to return a Boolean value.
- At the end, ***every()*** will return to us true (if all the elements in the array passed the test) or false (if at least one element didn't pass the test).

Example

```
/* This is our callback function */
function HasGradeBetweenRange(element_:* , index_:int, arr_:Array):Boolean
{
    if(element_>=0 && element_<=100)
    {
       return true;
    }
    else
    {
       return false;
    }
}

var aGradesSectionA:Array = [50, 85, 57, 100];
trace(aGradesSectionA.every(HasGradeBetweenRange));   /* true */

var aGradesSectionB:Array = [50, 101, 57, 12];
trace(aGradesSectionB.every(HasGradeBetweenRange));   /* false */
```

filter

filter(callback:Function, thisObject:* = null):Array

- Executes a test function on each item in the array and constructs a new array for all items that return ***true*** for the specified callback function.
- So basically, the callback function is checking if each element inside the array satisfies the condition inside it. It will return true if it does, false if it doesn't
- Of course the callback function has to return a Boolean value.
- At the end, ***filter()*** will return to us an Array filled with the elements that passed the test.

Example

```
/* This is the callback function */
function IsFailingGrade(element_:* , index_:int, arr_:Array):Boolean
{
   if(element_ < 60)
   {
      return true;
   }

   return false;
}

var aNumbersSectionA:Array = [50, 85, 57, 100];
var aFailing:Array = aNumbersSectionA.filter(IsFailingGrade);
trace(aFailing);    /* output: 50, 57 */
```

forEach

forEach(callback:Function, thisObject:* = null):void

- Executes a function on each item in the array.
- The callback function is modifying each element by applying some sort of instructions on them.
- Of course the callback function shouldn't return anything.
- At the end, ***forEach()*** won't return to us anything, but the original Array will contain the modified elements.

Example

```
/* This is our callback function */
function TraceEvenNumbers(element_:* , index_:int, arr_:Array)
{
   if(element_ % 2 == 0)
   {
      trace(element_);
   }
}

var aNumbers:Array = [1, 2, 3, 4, 5];
aNumbers.forEach(TraceEvenNumbers);
/*
      output:
       2
       4
*/
```

NOTE!!!

NB: *if you want to change the values of the elements inside the array using the forEach function you need to access the elements using the arr_:Array and index_:int parameter and not the element_:* parameter.*

```
/* This is our callback function */
function MultiplyByTwo(element_:* , index_:int, arr_:Array)
{
   element_ *= 2;   /* WONT WORK!!!!! */
   arr_ [ index_ ] *= 2;
}

var aNumbers:Array = [1, 2, 3, 4, 5];
aNumbers.forEach(MultiplyByTwo);
trace(aNumbers); /* 2 , 4 , 6 , 8 , 10 */
```

map

map(callback:Function, thisObject:* = null):Array

- Executes a function on each item in an array, and constructs a new array of items corresponding to the results of the function on each item in the original array.
- The callback function is applying instructions on each element and returning the resulting value.
- Of course the callback function should return the same type as the element inside the original array.
- At the end, ***map()*** will return to us a new array containing the new values.

Example

```
/* This is our callback function */
function toUpper(element_:* , index_:int, arr_:Array):String
{
   return element_.toUpperCase();
}

var aNames:Array = ["john", "jack", "david"];
var aNamesUpdated:Array = aNames.map(toUpper);
trace(aNamesUpdated);  /* JOHN,JACK,DAVID */
trace(aNames);  /* john,jack,david */
```

VERY IMPORTANT!!!!

When using the map() function, if your original array contains complex types, you will be changing in both, the original and the newly created, arrays. (Example on the next slide)

My advice is to ONLY use the map() function when your array contains primitive types.

!!!Example!!!

```
function Reverse(element_:* , index_:int, arr_:Array):Array
{
   return element_.reverse();
}

var aNumbers:Array = new Array(1);
aNumbers[0] = new Array(1,2,3,4,5);
trace(aNumbers);   /* 1 , 2 , 3 , 4 , 5 */

var aNumbersUpdated:Array = aNumbers.map(Reverse);
trace(aNumbersUpdated);   /* 5 , 4 , 3 , 2, 1 */
trace(aNumbers);          /* 5 , 4 , 3 , 2, 1 */
```

some

some(callback:Function, thisObject:* = null):Boolean

- Executes a test function on each item in the array until an item is reached that returns true.
- So basically, the callback function is checking if each element inside the array satisfies the condition inside it. It will return true if it does, false if it doesn't.
- Of course the callback function has to return a Boolean value.
- At the end, ***some()*** will return to us true (if it found an element that satisfied the callback function's test) or false (if no elements satisfied the callback function's test).

Example

```
function IsAnUndefinedValue(element_:* , index_:int, arr_:Array):Boolean
{
    if(element_ == undefined)
    {
        return true;
    }
    return false;
}

var aNumbers:Array = new Array(5);
aNumbers[0] = 0;       aNumbers[1] = 1;       aNumbers[2] = 2;       aNumbers[3] = 3;

if(aNumbers.some(IsAnUndefinedValue) == true)  /* true */
{
    trace("Found an undefined value in the array");
}
else
{
    trace("Didn't find an undefined value in the array");
}
```

The End ☺