

Section 21 : Arrays

17 July 2024 10:12



Harsha Vardhan

- › UI Expert
- › .NET Expert
- › Module Lead
- › Corporate Instructor

Arrays

4:32 3,

Agenda



Understanding Arrays

What is array and how to create.



Arrays - For loop

Iterate through array based on index.



Arrays - Foreach loop

Iterate through array based on sequence.



System.Array class

Properties and methods of Array Class



Multi-Dim Arrays

Two-Dim or more than that.

Agenda

6

Jagged Arrays

Array of arrays.

Introducing Arrays

Next: Array 'for' loop

What

- › Array is a group of multiple values of same type.
- › Arrays are stored in continuous-memory-locations in 'heap'.

Array
[0]
[1]
[2]
[3]
[4]
[5]
[6]

How

```
type[ ] arrayReferenceVariableName = new type[ size ];
```



- › Each value of array is called "element".
- › All the elements are stored in continuous memory locations.
- › The address of first element of the array will be stored in the "array reference variable".
- › The "Length" property stores count of elements of array. Index starts from 0 (zero).
- › Arrays are treated as objects of "System.Array" class; so arrays are stored in heap; its address (first element's address) is stored in reference variable at stack.

Agenda

1

Understanding Arrays

What is array and how to create.

2

Arrays - For loop

Iterate through array based on index.

3

Arrays - Foreach loop

Iterate through array based on sequence.

4

System.Array class

Properties and methods of Array Class

5

Multi-Dim Arrays

Two-Dim or more than that.

What

- › For loop starts with an "initialization"; checks the condition; executes the loop body and then performs incrementation or decrementation;
- › Use "for loop" to read elements from an array, based on index.

Array	
i = 0	a[0]
i = 1	a[1]
i = 2	a[2]
i = 3	a[3]
i = 4	a[4]
i = 5	a[5]
i = 6	a[6]

Array - For Loop

```
for (int i = 0; i < arrayRefVariable.Length; i++)  
{  
    arrayRefVariable[i]  
}
```



3

Arrays - Foreach loop

Iterate through array based on sequence.

Pros

- › You can read any part of the array (all elements, start from specific element, end with specific element).
- › You can read array elements in reverse.

Cons

- › A bit complex syntax.

What

- › Fforeach loop starts contains a "iteration variable"; reads each value of an array or collection and assigns to the "iteration variable", till end of the array / collection.
- › "Fforeach loop" is not based on index; it is based on sequence.

Array	
i = 0	a[0]
i = 1	a[1]
i = 2	a[2]
i = 3	a[3]
i = 4	a[4]
i = 5	a[5]
i = 6	a[6]

Array - ForEach Loop

```
foreach (DataType iterationVariable in arrayVariable)
{
    iterationVariable
}
```

Pros

- › Simplified Syntax
- › Easy to use with arrays and collections.
- › It internally uses "Iterators".

Cons

- › Slower performance, due to it treats everything as a collection.
- › It can't be used to execute repeatedly, without arrays or collections.
- › It can't read part of array / collection, or read array / collection reverse.

Agenda

1

Understanding Arrays

What is array and how to create.

2

Arrays - For loop

Iterate through array based on index.

3

Arrays - Foreach loop

Iterate through array based on sequence.

4

System.Array class

Properties and methods of Array Class



5

Multi-Dim Arrays

Two-Dim or more than that.

What

- > Every array is treated as an object for System.Array class.
- > The System.Array class provides a set of properties and methods for every array.

Array	
i = 0	a[0]
i = 1	a[1]
i = 2	a[2]
i = 3	a[3]
i = 4	a[4]
i = 5	a[5]
i = 6	a[6]



Array

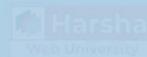
```
type[ ] arrayReferenceVariableName = new type[ size ];
```

Properties

- > Length

Methods

- > IndexOf
- > BinarySearch
- > Clear
- > Resize
- > Sort
- > Reverse
- > CopyTo
- > Clone



Properties

- > Length

Methods

- > IndexOf
- > BinarySearch
- > Clear
- > Resize
- > Sort
- > Reverse
- > CopyTo
- > Clone

[Array - IndexOf\(\) method](#)Next: [BinarySearch\(\)](#)

Harsha

IndexOf()

- > This method searches the array for the given value.
- > If the value is found, it returns its index.
- > If the value is not found, it returns -1.

Array	
i = 0	a[0]
	value0
i = 1	a[1]
	value1
i = 2	a[2]
	value2
i = 3	a[3]
	value3
i = 4	a[4]
	value4
i = 5	a[5]
	value5
i = 6	a[6]
	value6

[Array - IndexOf\(\) method](#)[Array - IndexOf\(\) - Example](#)**static int System.IndexOf(System.Array array, int value)****Array.IndexOf(value3) = 3**

This method searches the array for the given value.



- › The "IndexOf" method performs linear search. That means it searches all the elements of an array, until the search value is found. When the search value is found in the array, it stops searching and returns its index.
- › The linear search has good performance, if the array is small. But if the array is larger, Binary search is recommended to improve the performance

Parameters

- › **array:** This parameter represents the array, in which you want to search.
- › **value:** This parameter represents the actual value that is to be searched.
- › **startIndex:** This parameter represents the start index, from where the search should be started.

BinarySearch()

- › This method searches the array for the given value.
- › If the value is found, it returns its index.
- › If the value is not found, it returns -1.

		Array	
i = 0	a[0]	value0	
i = 1	a[1]	value1	
i = 2	a[2]	value2	
i = 3	a[3]	value3	
i = 4	a[4]	value4	
i = 5	a[5]	value5	
i = 6	a[6]	value6	

```
static int Array.BinarySearch( System.Array array, int value)
```

```
Array.BinarySearch(value3) = 3
```



- › The "Binary Search" requires an array, which is already sorted.
- › On unsorted arrays, binary search is not possible.
- › It directly goes to the middle of the array (array size / 2), and checks that item is less than / greater than the search value.

Parameters

- › **array:** This parameter represents the array, in which you want to search.
- › **value:** This parameter represents the actual value that is to be searched



- › The "Binary Search" requires an array, which is already sorted.
- › On unsorted arrays, binary search is not possible.
- › It directly goes to the middle of the array (array size / 2), and checks that item is less than / greater than the search value.
- › If that item is greater than the search value, it searches only in the first half of the array.
- › If that item is less than the search value, it searches only in the second half of the array.
- › Thus it searches only half of the array. So in this way, it saves performance

Parameters

- › **array:** This parameter represents the array, in which you want to search.
- › **value:** This parameter represents the actual value that is to be searched

Properties

- › Length

Methods

- › IndexOf
- › BinarySearch
- › Clear**
- › Resize
- › Sort
- › Reverse
- › CopyTo
- › Clone

Clear()

- › This method starts with the given index and sets all the "length" no. of elements to zero (0).

Array		
i = 0	a[0]	value0
i = 1	a[1]	value1
i = 2	a[2]	0
i = 3	a[3]	0
i = 4	a[4]	0
i = 5	a[5]	value5
i = 6	a[6]	value6

Array - Clear() method

```
static void Array.Clear( System.Array array, int value, int length)
```

Clear() - Example

```
Array.Clear(a, 2, 3)
```

Parameters

- › **array:** This parameter represents the array, in which you want to clear the elements.
- › **index:** This parameter represents the index, from which clearing process is to be started.
- › **length:** This parameter represents the no. of elements that are to be cleared.



Resize()

- › This method increases / decreases size of the array.

Array		
i = 0	a[0]	value0
i = 1	a[1]	value1
i = 2	a[2]	value2
i = 3	a[3]	value3
i = 4	a[4]	value4



Array - Resize() method

```
static void Array.Resize(ref System.Array array, int newSize)
```

Array - Resize() method

Next: Sort()

Resize()

- › This method increases / decreases size of the array.



Array - Resize() method

Array - Resize() - Example

```
static void Array.Resize(ref System.Array array, int newSize)
```

```
Array.Resize(a, 7)
```

45

Array.Resize() method

Next: Sort()

Parameters

- › **array:** This parameter represents the array, which you want to resize.
- › **newSize:** This parameter represents the new size of the array, how many elements you want to store in the array. It can be less than or greater than the current size.

Internally it creates a new array and copy from the old array.

Sort()

> This method sorts the array in ascending order.

Array	
i = 0	a[0]
i = 1	a[1]
i = 2	a[2]
i = 3	a[3]
i = 4	a[4]
i = 5	a[5]
i = 6	a[6]
	100
	950
	345
	778
	20
	800
	80

Sort

Array	
i = 0	a[0]
i = 1	a[1]
i = 2	a[2]
i = 3	a[3]
i = 4	a[4]
i = 5	a[5]
i = 6	a[6]
	20
	80
	100
	345
	778
	800
	950

Array - Sort() method

Array - Sort() - Example

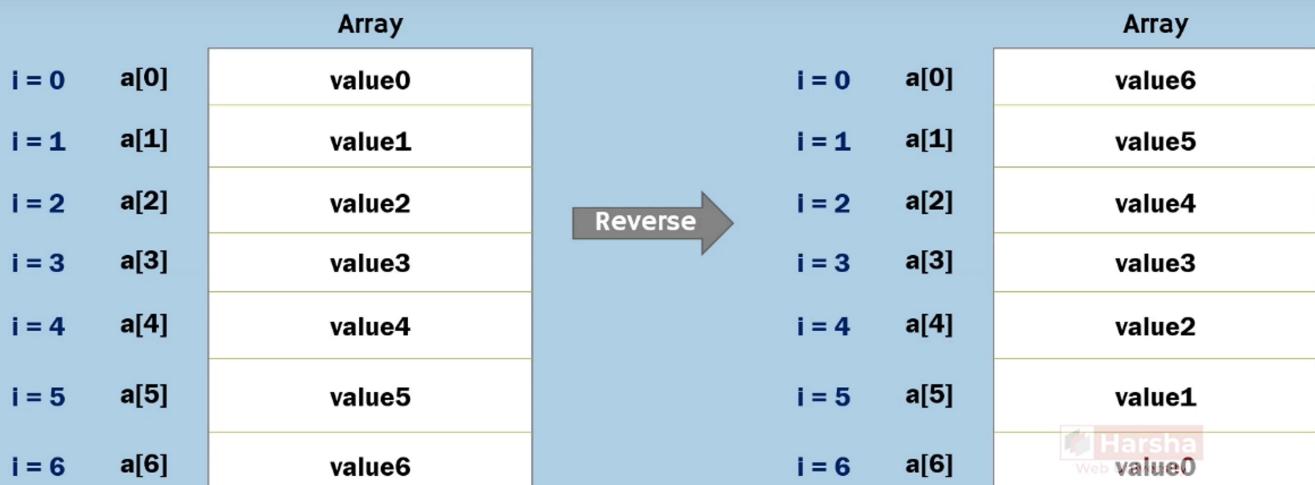
~~static void Array.Sort(System.Array array)~~

Array.Sort(a)

By default, it internally uses QuickSort algorithm to perform sort operation.

Reverse()

→ This method reverses the array.

**Array - Reverse() method****Array - Reverse() - Example**

```
static void Array.Reverse( System.Array array )
```

```
Array.Reverse(a)
```

Properties and methods of Array Class

Web University

5

Multi-Dim Arrays

Two-Dim or more than that.

1x 0:00 / 9:10

Single-Dim Arrays

- Group of multiple rows; each row contains a single value.

array		
i = 0	a[0]	value0
i = 1	a[1]	value1
i = 2	a[2]	value2
i = 3	a[3]	value3
i = 4	a[4]	value4

Multi-Dim Arrays

- Group of multiple rows; each row contains a group of values.

[Row 0]	{ value1, value2 }	[Column 1]
[Row 1]	{ value1, value2 }	
[Row 2]	{ value1, value2 }	
[Row 3]	{ value1, value2 }	
[Row 4]	{ value1, value2 }	

Multi-Dim Arrays

Next: Jagged Arrays

What

- Stores elements in rows & columns format.
- Every row contains a series of elements.

[Row 0]	{ value1, value2 }	[Column 1]
[Row 1]	{ value1, value2 }	
[Row 2]	{ value1, value2 }	
[Row 3]	{ value1, value2 }	
[Row 4]	{ value1, value2 }	

Multi-Dim Array

```
type[ , ] arrayReferenceVariable = new type[ rowSize, columnSize ];
```

What

- > Stores elements in rows & columns format.
- > Every row contains a series of elements.
- > You can create arrays with two or dimensions, by increasing the no. of commas (,).

	[Column 0]	[Column 1]
[Row 0]	{ value1,	value2 }
[Row 1]	{ value1,	value2 }
[Row 2]	{ value1,	value2 }
[Row 3]	{ value1,	value2 }
[Row 4]	{ value1,	value2 }

Multi-Dim Array

```
type[ , ] arrayReferenceVariable = new type[ rowSize, columnSize ];
```

6

Jagged Arrays

Array of arrays.

Jagged Arrays

What

- > Jagged Array is an “array of arrays”.
- > The member arrays can be of any size.

Jagged Array

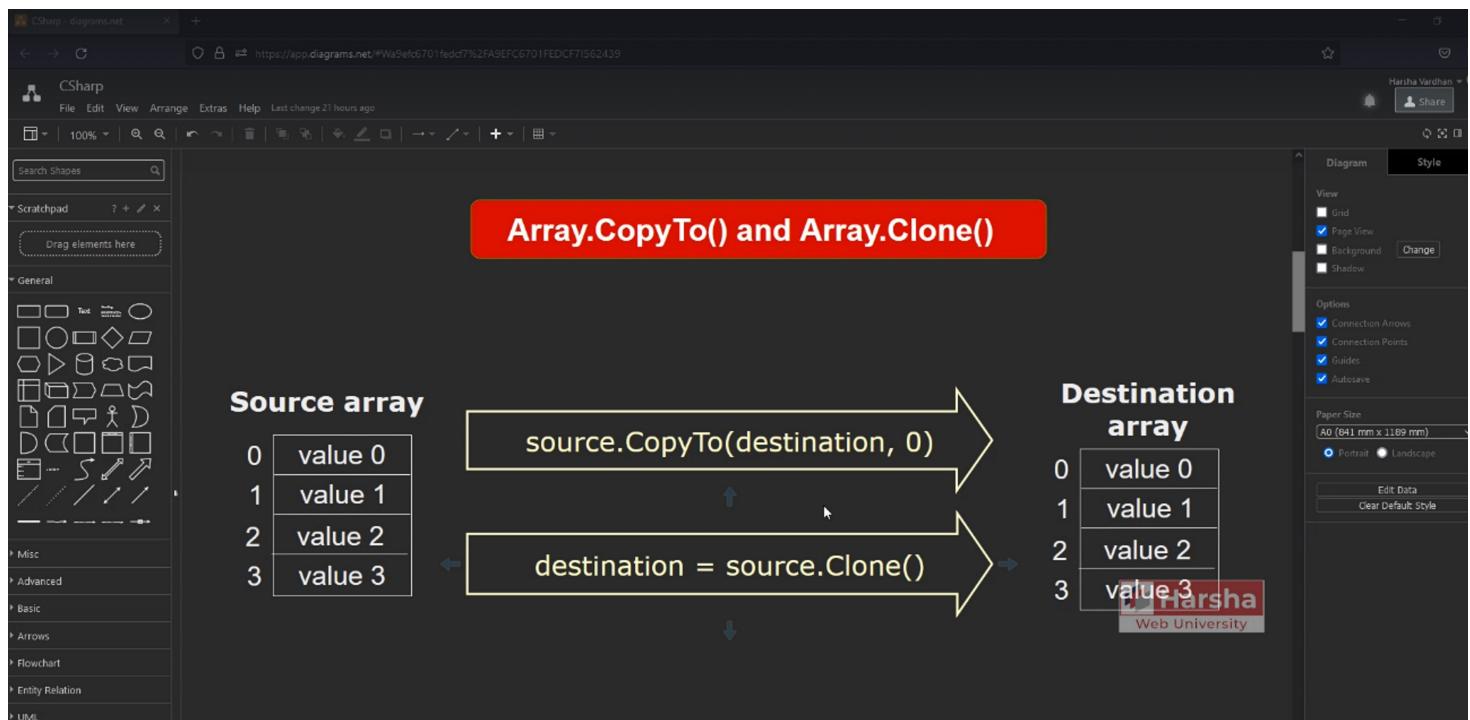
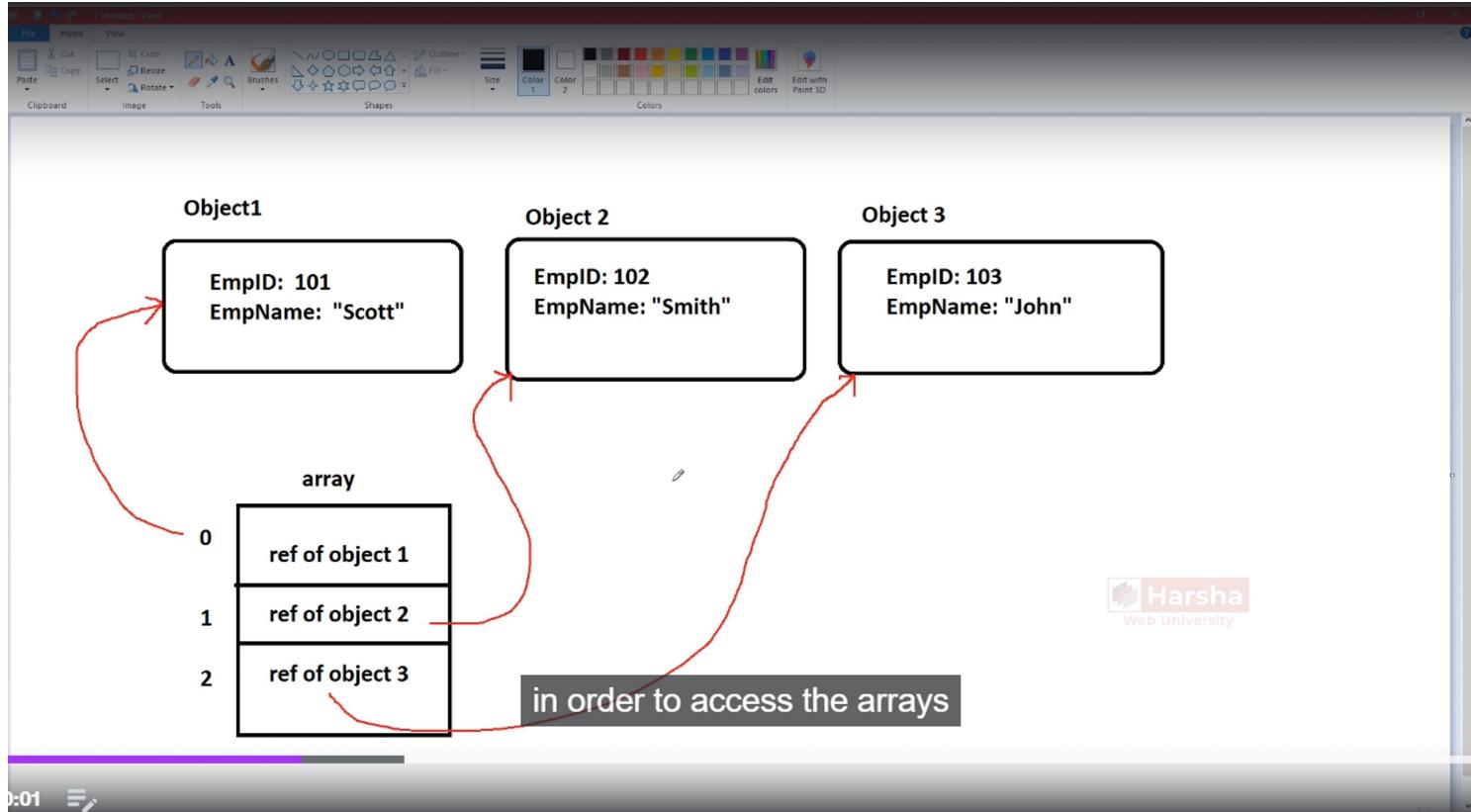
[Row 0]	{ value1, value2 }
[Row 1]	{ value1, value2, value3 }
[Row 3]	{ value1, value2, value3, value4, value5 }
[Row 4]	{ value1 }
[Row 5]	{ value1, value2, value3, value4, value5, value6, value7 }

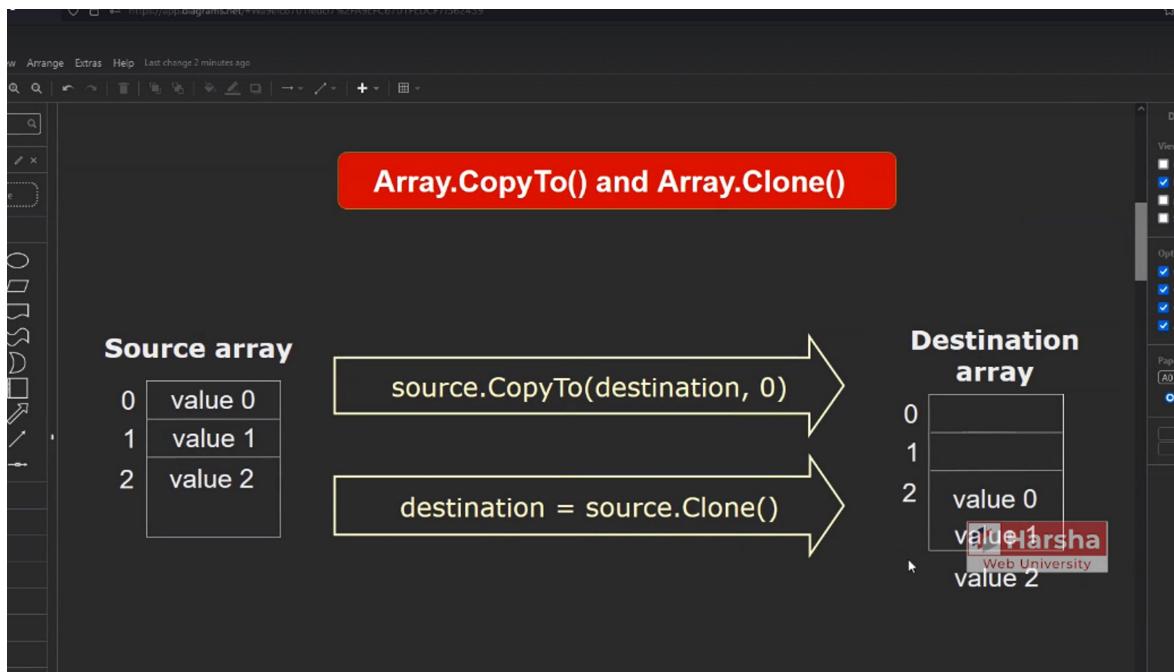
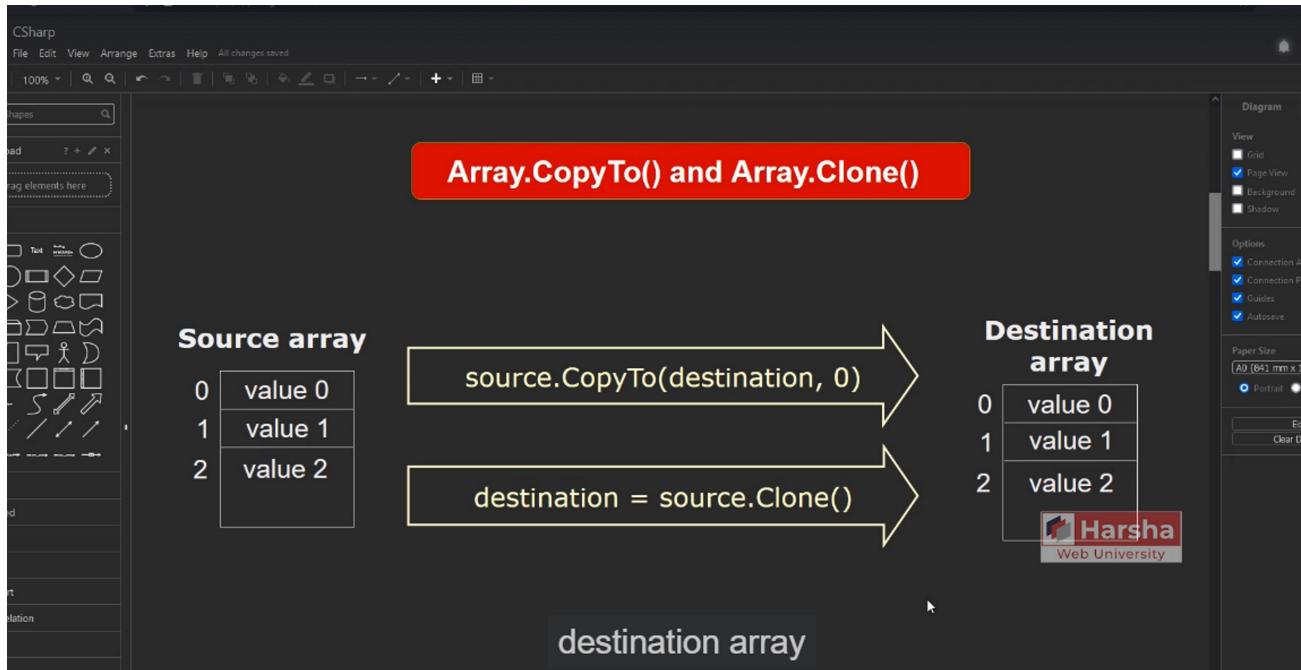
Jagged Array

How

```
type[ ] [ ] arrayReferenceVariable = new type[ rowSize ] [ ];  
arrayReferenceVariable[index] = new type[ size ];
```

Arrays of Objects





Array.CopyTo()

CopyTo() requires to have an existing destination array; and the destination array should be large enough to hold all elements from the source array, starting from the specified startIndex.

CopyTo() allows you to specify the startIndex at destination array.

The result array need not be type-casted explicitly.

Array.Clone()

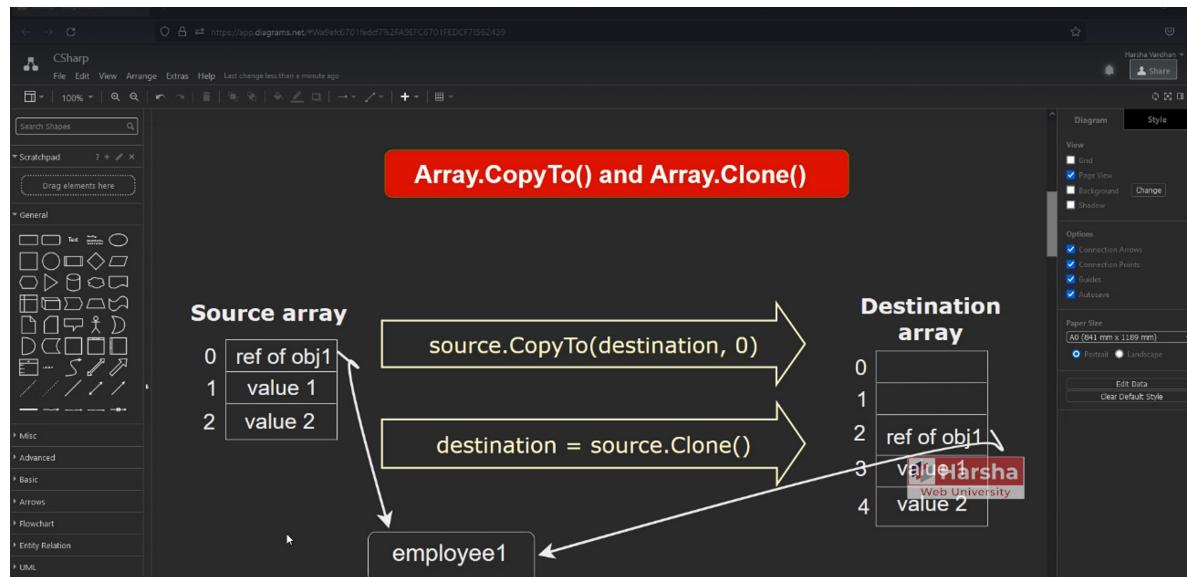
Clone() creates a new destination array; you need not have an existing array.

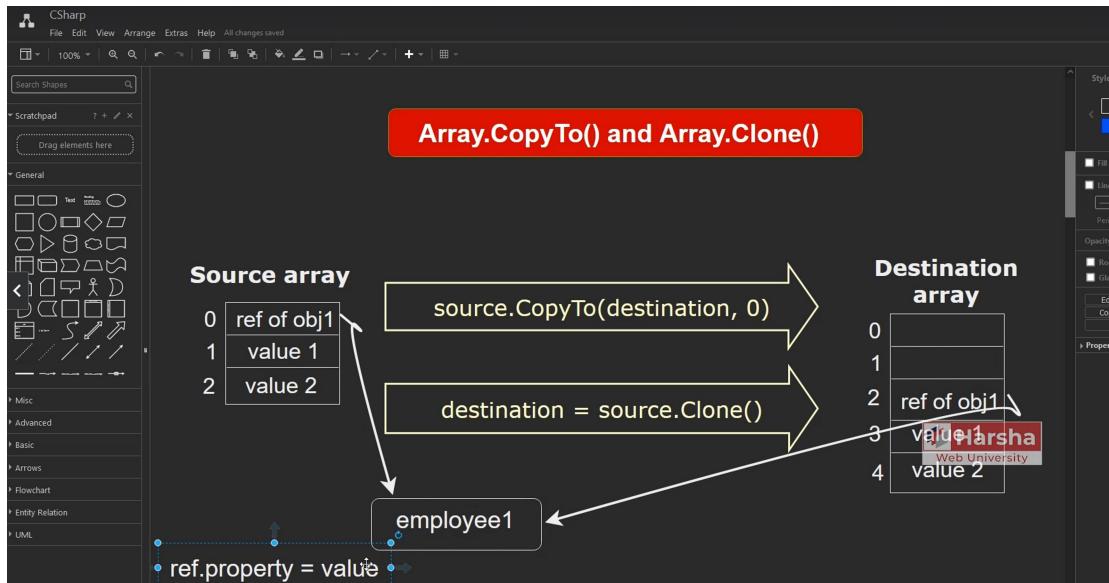


Clone() doesn't allow you to specify the startIndex at destination array.

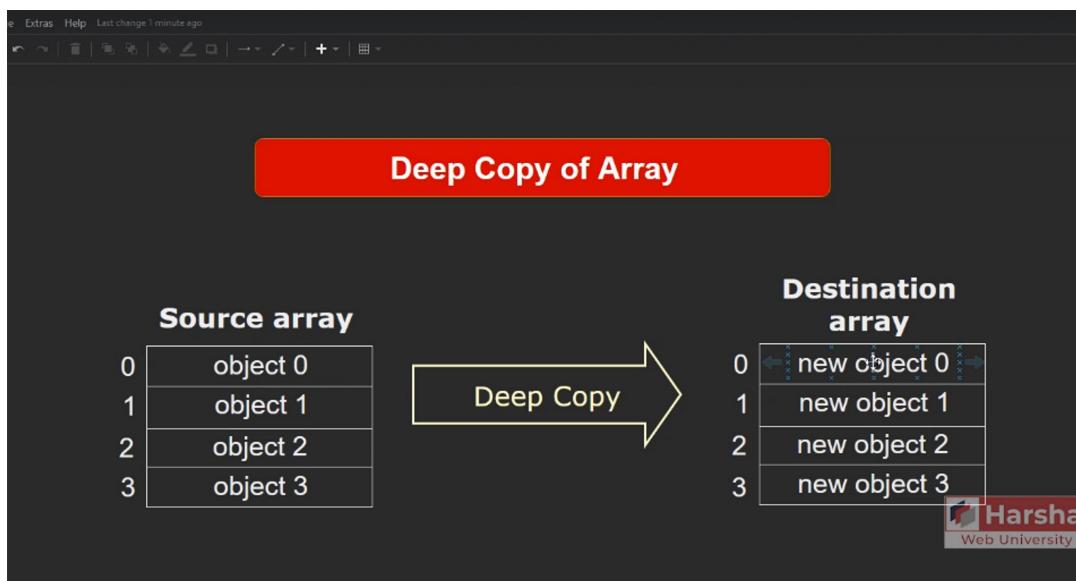
The result array will be returned as 'object' type; so need to be type-casted to array type.

Both performs Shallow Copy. Not Deep Copy





I don't want to copy the reference but actual value of source array. we can do it by deep clone.



The diagram shows a screenshot of a C# code editor with annotations. The code defines a `Program` class with a `Main` method. Inside `Main`, an array `employees` is created and populated with three `Employee` objects. The first object is then passed to a `CreateNewEmployee` method. Red arrows and boxes highlight the `this` keyword, the `new Employee()` call, and the parameter being passed. A callout box labeled "copy" points to the arrow between the original object and the new object. Another callout box labeled "new object" points to the receiving box. A note at the bottom states: "employees[0].CreateNewEmployee() into the new object".

```
11  public void CreateNewEmployee()
12  {
13      this
14      new Employee()
15  }
16
17  class Program
18  {
19      static void Main()
20      {
21          Employee[] employees = new Employee[]
22          {
23              new Employee() { EmployeeName = "Joseph", Role = "Developer" },
24              new Employee() { EmployeeName = "Jack", Role = "Designer" },
25              new Employee() { EmployeeName = "Alexa", Role = "Analyst" }
26          };
27          employees[0].CreateNewEmployee() into the new object
    }
```

In order to instruct all C# developer to give the same method name (Clone), there is a predefined interface in C#,

System.ICloneable

