Event Driven Programming

Programming with Event Driven in C#

Chalew Tesfaye
Department of Computer Science
Debre Berhan University
2017

Objectives

- After the end this lesson the student will be able to
  - ✓ Understand C# language fundamentals
    - > Data type, variables and constants, …
  - ✓ Write a C# program statement
  - ✓ Develop OO program with C#

Lesson Outline

- Arrays and Collections
- Methods and Event handlers
- Object oriented programming
  - ✓ Classes
  - ✓ Indexer, delegates, events and operators
  - ✓ Inheritance
  - ✓ Interface and generics

Collections

Processing Sequences of Elements and set of elements

Array

- An array is a sequence of elements
  - ✓ All elements are of the same type
  - ✓ The order of the elements is fixed
  - ✓ Has fixed size (`Array.Length`)
  - ✓ Zero based index
  - ✓ Is reference type
- Declaration - defines the type of the elements
  - ✓ Square brackets `[ ]` mean "array"
  - ✓ Examples:
    - > `int[] myIntArray;`
    - > `string[] myStringArray;`
  - ✓ Use the operator `new`
    - > Specify array length
    - > `myIntArray = new int[5];`
  - ✓ Creating and initializing can be done together:
    - > `myIntArray = {1, 2, 3, 4, 5};`
  - ✓ The `new` operator is not required when using curly brackets initialization
- C# provides automatic bounds checking for arrays

Creating Array > Example

- Creating an array that contains the names of the days of the week

```
string[] daysOfWeek =
{
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    "Sunday"
};
```

Accessing Array Elements

- Read and Modify Elements by Index
- Array elements are accessed using the square brackets operator `[]` (indexer)
  - ✓ Array indexer takes element's index as parameter
  - ✓ The first element has index `0`
  - ✓ The last element has index `Length-1`
- Array elements can be retrieved and changed by the `[]` operator

Accessing Array Elements > Example

- Reading Arrays From the Console
  - ✓ First, read from the console the length of the array

```
int n = int.Parse(Console.ReadLine());
```

  - ✓ Next, create the array of given size and read its elements in a `for` loop

```
int[] arr = new int[n];
for (int i=0; i<n; i++)
{
   arr[i] = int.Parse(Console.ReadLine());
}
```

- Print each element to the console

```
string[] array = {"one", "two", "three"};
// Process all elements of the array
for (int index = 0; index < array.Length; index++)
{
   // Print each element on a separate line
   Console.WriteLine("element[{0}] = {1}",
      index, array[index]);
}
```

Accessing Array Elements > Example

- Reversing an Array

```
int[] array = new int[] {1, 2, 3, 4, 5};
// Get array size
int length = array.Length;
// Declare and create the reversed array
int[] reversed = new int[length];
// Initialize the reversed array
for (int index = 0; index < length; index++)
{
   reversed[length-index-1] = array[index];
}
```

- Read int array from the console and check if it is symmetric:

```
bool isSymmetric = true;
for (int i=0; i<(array.Length+1)/2; i++)
{
   if (array[i] != array[n-i-1])
   {
      isSymmetric = false;
   }
}
```

Accessing Array Elements > Example

- Printing array of integers in reversed order:

```
int[] array ={12,52,83,14,55};
Console.WriteLine("Reversed: ");
for (int i = array.Length-1; i >= 0; i--)
{
    Console.Write(array[i] + " ");
}
```

- Print all elements of a `string[]` array using foreach:

```
string[] cities= { “Adama”, “Hawassa", “Debre Berhan", “Bahir Dar", “Mekelle"};
foreach (string city in cities)
{
    Console.WriteLine(city);
}
```

Multidimensional Arrays

- Using Array of Arrays, Matrices and Cubes
- Multidimensional arrays have more than one dimension (2, 3, …)
  - ✓ The most important multidimensional arrays are the 2-dimensional
    - > Known as matrices or tables
- Declaring multidimensional arrays:

```
int[,] intMatrix;
float[,] floatMatrix;
string[,,] strCube;
```

- Creating a multidimensional array
  - ✓ Use new keyword
  - ✓ Must specify the size of each dimension

```
int[,] intMatrix = new int[3, 4];
float[,] floatMatrix = new float[8, 2];
string[,,] stringCube = new string[5, 5, 5];
```

- Creating and initializing with values multidimensional array:

```
int[,] matrix =
{
   {1, 2, 3, 4}, // row 0 values
   {5, 6, 7, 8}, // row 1 values
}; // The matrix size is 2 x 4 (2 rows, 4 cols)
```

Multidimensional Arrays > Example

- Reading a matrix from the console

```
int rows = int.Parse(Console.ReadLine());
int cols= int.Parse(Console.ReadLine());
int[,] matrix = new int[rows, cols];
String inputNumber;
for (int row=0; row<rows; row++)
{
  for (int col=0; cols<cols; col++)
  {
    Console.Write("matrix[{0},{1}] = ", row, col);
    inputNumber = Console.ReadLine();
    matrix[row, col] = int.Parse(inputNumber);
  }
}
```

Multidimensional Arrays > Example

- Printing a matrix on the console:

```
for (int row=0; row<matrix.GetLength(0); row++)
{
  for (int col=0; col<matrix.GetLength(1); col++)
  {
    Console.Write("{0} ", matrix[row, col]);
  }
  Console.WriteLine();
}
```

Multidimensional Arrays > Example

- Finding a 2 x 2 platform in a matrix with a maximal sum of its elements

```
int[,] matrix = {
  {7, 1, 3, 3, 2, 1},
  {1, 3, 9, 8, 5, 6},
  {4, 6, 7, 9, 1, 0}
};
int bestSum = int.MinValue;
for (int row=0; row<matrix.GetLength(0)-1; row++)
  for (int col=0; col<matrix.GetLength(1)-1; col++)
  {
    int sum = matrix[row, col] + matrix[row, col+1]
      + matrix[row+1, col] + matrix[row+1, col+1];
    if (sum > bestSum)
      bestSum = sum;
  }
```

Array Functions

- Array is a class; hence provides properties and methods to work with it
- Property
  - ✓ Length – gets the number of elements in all the dimension of array
- Methods
  - ✓ GetLength(dimension) - gets the number of elements in the specified dimension of an array
  - ✓ GetUpperBound(dimension) – Gets the index of the last elements in the specified dimension of an array
  - ✓ Copy(array1, array2, length) – Copies some or all of the values in one array to another array.
  - ✓ BinarSearch(array, value) – Searches a one-dimensional array that's in ascending order and returns the index for a value
  - ✓ Sort(array) – Sorts the elements of a one dimensional array in to ascending order
  - ✓ Clear(array, start, end) – clears elements of an array
  - ✓ Reverse(array) - reverses the elements of an array
- The BinarySearch method only works on arrays whose elements are in ascending order

Array Function > Example

```
int [] number = new int [4] {1,2,3,4,5};
int len = numbers.GetLength(0);
int upperBound = numbers.GetUpperBound(0);
//
string[] names = {“Abebe”, “Kebede”, “Soliana”, “Fatuma”, “Makida”};
Array.Sort(names);
foreach(string name in names)
    Consolw.WriteLine(name);
//
decimal[] sales = {15463.12m, 25241.3m, 45623.45m, 41543.23m,
40521.23m};
int index = Array.BinarySearch(names, “Soliana”);
decimal  sale = sales[index];
```

Copying Arrays

- Sometimes we may need to copy the values from one array to another one
  - ✓ If we do it the intuitive way we would copy not only the values but the reference to the array
  - ✓ Changing some of the values in one array will affect the other
    - > int[] copyArray=array;
  - ✓ The way to avoid this is using Array.Copy()
    - > Array.Copy(sourceArray, copyArray);
  - ✓ This way only the values will be copied but not the reference
- Reading assignment : How to work with Jagged arrays

List<T>

- `Lists` are arrays that resize dynamically
  - ✓When adding or removing elements
    - > use indexers (like `Array`)
  - ✓`T` is the type that the List will hold
    - > E.g.
      - `List<int>` will hold integers
      - `List<object>` will hold objects
- Basic Methods and Properties
  - ✓`Add(T element)` – adds new element to the end
  - ✓`Remove(element)` – removes the element
  - ✓`Count` – returns the current size of the List

List > Example

```
List<int> intList=new List<int>();
for( int i=0; i<5; i++)
{
  intList.Add(i);
}
```

- ✓ Is the same as

```
int[] intArray=new int[5];
for( int i=0; i<5; i++)
{
  intArray[i] = i;
}
```

- The main difference
  - ✓ When using lists we don't have to know the exact number of elements

Lists vs. Arrays

- Lets have an array with capacity of 5 elements

```
int[] intArray=new int[5];
```

- If we want to add a sixth element (we have already added 5) we have to do

```
int[] copyArray = intArray;
intArray = new int[6];
for (int i = 0; i < 5; i++)
{
   intArray[i] = copyArray[i];
}
intArray[5]=newValue;
```

- With List we simply do

```
list.Add(newValue);
```

ArrayList

- It represents ordered collection of an object that can be **indexed** individually.
- It is basically an alternative to an array.
- However, unlike array you can add and remove items from a list at a specified position using an **index** and the array resizes itself automatically.

```
ArrayList arrayList = new ArrayList();
```

- It can contain a mixed content as object
- It also allows dynamic memory allocation, adding, searching and sorting items in the list.

ArrayList > Example

```
ArrayList al = new ArrayList();
Console.WriteLine("Adding some numbers:");
al.Add(45);
al.Add(78);
al.Add(33);
al.Add(56);
al.Add(12);
al.Add(23);
al.Add(9);
Console.WriteLine("Capacity: {0} ", al.Capacity);
Console.WriteLine("Count: {0}", al.Count);
Console.Write("Content: ");
foreach (int i in al)
{
  Console.Write(i + " ");
}
Console.WriteLine();
Console.Write("Sorted Content: ");
al.Sort();
foreach (int i in al)
{
  Console.Write(i + " ");
}
Console.WriteLine();
```

```
ArrayList arrrayList = new ArrayList();
arrrayList.Add(“One”);
arrrayList.Add(2);
arrrayList.Add(“Three”);
arrrayList.Add(4);
int number = 0;
foreach(object obj in araayList)
{
        If(obj is int)
        {
                number += (int) obj;
        }
}
```

Enumeration - enum

- Enumeration is a set of related constants that define a value type
- Each constant is a member of the enumeration
  - ✓ syntax

```
enum enumName [: type]
{
    ConstantName1 [= value1],
    ConstantName2 [=value2], …
}
```

- Example

```
enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
int WeekdayStart = (int)Days.Mon;
int WeekdayEnd = (int)Days.Fri;
Console.WriteLine("Monday: {0}", WeekdayStart);
Console.WriteLine("Friday: {0}", WeekdayEnd);
```

Dictionaries

- Dictionaries are used to associate a a particular key with a given value
- In C#, defined by Hashtable class
- It uses a **key** to access the elements in the collection.
- A hash table is used when you need to access elements by using key, and you can identify a useful key value.
- Each item in the hash table has a **key/value** pair.
- The key is used to access the items in the collection.
- Key must be unique
- Do not have any sense of order

Hashtable > Example

```
Hashtable ht = new Hashtable();
ht.Add("001", “Abe Kebe");
ht.Add("002", “Alem Selam");
ht.Add("003", “Fatuma Ahmed");
ht.Add("004", “Soliana Abesselom");
ht.Add("005", “Tigist Abrham");
ht.Add("006", “Selam Ahmed");
ht.Add("007", “Makida Birhanu");
if (ht.ContainsValue(" Selam Ahmed "))
{
  Console.WriteLine("This student name is already in the list");
}
else
{
  ht.Add("008", " Selam Ahmed ");
}
// Get a collection of the keys.
ICollection key = ht.Keys;
foreach (string k in key)
{
  Console.WriteLine(k + ": " + ht[k]);
}
```

Stacks

- Stack also maintain a list of items like array and ArrayList, but
  - ✓ Operates on a push-on and pop-off paradigm
- Stacks are LIFO – Last In First Out

```
Stack stack = new Stack();
stack.Push("item"); // insert item on the top
object obj = stack.Pop(); // gets top item
object obp = stack.Peek() // looks at top
int size = stack.Count; //gets the number of items in the stack
```

Queues

- Queues maintain a list of items like stack, but
  - ✓ Operate with a principle of FIFO – First In First Out

```
Queue queue = new Queue();
queue.Enqueue("item"); //insert an item to the last
object obj = queue.Dequeue(); // gets first item
int size = queue.Count; // gets number of items
```

Quiz-5pts

1. What is the difference between Array, List and ArrayList?
2. Write a program that accept two arrays of integer with n elements and display an array that contain the sum of each element of the first two arrays.
3. Write a program that accepts a two dimensional square matrix with nxn element and print the elements in formatted way.

For more information

- Brian Bagnall, et al. C# for Java Programmers. USA. Syngress Publishing, Inc.
- Svetlin Nakov *et al*. Fundamentals of Computer Programming With C#. Sofia, 2013
- Joel Murach, Anne Boehm. Murach C# 2012, Mike Murach & Associates Inc USA, 2013

QUESTION?