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

- Working with Date and string data types
- Arrays and Collections
- Methods and Event handlers
- Object oriented programming
  - ✓ Classes
  - ✓ Indexer, delegates, events and operators
  - ✓ Inheritance
  - ✓ Interface and generics

Date and string data types

Working with Date and string data types

Dates and Times

- Data processing on date and time values
- DateTime structure of .Net framework provides a variety of properties and methods for
  - ✓ Getting information about dates and times
  - ✓ Formatting date and time values
  - ✓ Performing operations on dates and times
- To create a DateTime value
  - ✓ Use new key word
  - ✓ Specify the date and time values

Syntax:

```
DateTime variable = new DateTime(year, month, day[, hour, minute, second[, millisecond]]);
```

- ✓ If time is not specified, it set to 12:00 AM.
- ✓ Use also static Parse and TryParse method to create DateTime value from a string

```
DateTime variable = DateTime.Parse(string);
DateTime variable;
DateTime.TryParse(string, out variable);
```

Dates and Times > Example

```
DateTime startDate = new DateTime(2017, 01, 30); //
DateTime startDateTime = new DateTime(2017, 01, 30, 13, 30, 0); //
DateTime startDate = DateTime.Parse("03/08/17"); //
DateTime startDate = DateTime.Parse ("Feb 08, 2017 1:30 PM"); //
DateTime dateOfBirth= DateTime.Parse (Console.ReadLine()); //
DateTime expDate;
DateTime.TryParse (Console.ReadLine(), out expDate); //
DateTime deadlineDate;
Console.WriteLine("Enter deadline Date (mm/dd/yyyy) :");
deadlineDate = Convert.ToDateTime(Console.ReadLine());
```

Dates and Times > Date and Time writing format

- Valid date format includes:
  - ✓ 01/30/2017
  - ✓ 1/30/17
  - ✓ 01-30-2017
  - ✓ 1-30-17
  - ✓ 2017-30-1
  - ✓ Jan 30 2017
  - ✓ January 30, 2017
- Valid time format includes:-
  - ✓ 2:15 PM
  - ✓ 14:15
  - ✓ 02:15:30 AM

Dates and Times > Current Date and Time

- Use two static properties of DateTime structure to get the current date/time
  - ✓ Now
    - > Returns the current date and time
  - ✓ Today
    - > Returns the current date
- Example

```
DateTime currDateTime = DateTime.Now; //
DateTime currDate = DateTime.Today; //
DateTime regDateTime = DateTime.Toady; //
 DateTime modifiedDate = DateTime.Now; //
```

Dates and Times > Date and Time formatting

- The format depends on the computer regional setting
- Use the methods of DateTime structure to format the date/time in the way you want
- ToLongDateString()
  - ✓ Name for the day of the week, the name for the month, the date of the month, and the year
- ToShortDateString()
  - ✓ The numbers for the month, day, and year
- ToLongTimeString()
  - ✓ The hours, minutes and seconds
- ToShortTimeString()
  - ✓ The hours and minutes

Dates and Times > Date and Time formatting >Example

- Statements that format dates and times data

```
DateTime currDateTime = DateTime.Now;
string longDate = currDateTime.ToLongDateString(); //
string shortDate = currDateTime.ToShortDateString(); //
string longTime = currDateTime.ToLongTimeString(); //
string shortDate = currDateTime.ToShortTimeString(); //
```

Dates and Times > Getting information about Dates and Times

- The DateTime structure provides a variety of properties and methods for getting information about dates and times
- Property
  - ✓ Date – Returns the DateTime value with the time period portion se to 12:00 AM.
  - ✓ Month – Returns an integer for the month portion of the DateTime value.
  - ✓ Day – Returns an integer for the day portion of the DateTime value.
  - ✓ Year – Returns an integer for the year portion of the DateTime value.
  - ✓ Hour – Returns an integer for the hour portion of the DateTime value.
  - ✓ Minute – Returns an integer for the minute portion of the DateTime value.
  - ✓ Second – Returns an integer for the second portion of the DateTime value.
  - ✓ TimeOfDay – Returns a TimeSpan value that represents the amount of time has elapsed since 12:00 AM.
  - ✓ DayOfWeek – Returns a number of the DayOfWeek enumeration that represents the day of the week a DateTime value
  - ✓ DayOfYear - – Returns an integer for the numeric day of the year.
- Method
  - ✓ DaysInMonth(year, month) – Returns the number of days in a specified month and year.
  - ✓ IsLeapYear(year) – Returns a Boolean value that indicates whether or no a specified year is a leap year.

Getting information about Dates and Times > Example

```
DateTime currDateTime = DateTime.Now;
int month = currDateTime.Month;
int hour = currDateTime.Hour;
int dayOfYear = currDateTime.DayOfYear;
int daysInMonth = DateTime.DaysInMonth(currDateTime.Year, 2);
bool isLeapYear = currDateTime.IsLeapYear();
DayOfWeek  dayOfWeek = currDateTime.DayOfWeek;
string message=“”:
if(dayOfWeek == DayOfWeek.Saturday || dayOfWeek == DayOfWeek.Sunday )
{
      message = “Weekend”;
}
else {
      message = “Week day”;
}
```

- Methods for performing operations on dates and times
- Method
  - ✓ AddDays(days) - returns a DateTime value
  - ✓ AddMonths(months) - returns a DateTime value
  - ✓ AddYears(years) - returns a DateTime value
  - ✓ AddHours(hours) - returns a DateTime value
  - ✓ AddMinutes(minutes) - returns a DateTime value
  - ✓ AddSeconds(seconds) - returns a DateTime value
  - ✓ Add(timespan) - returns a DateTime value
  - ✓ Subtract(datetime) – returns a TimeSpan value
  - ✓ In addtion, possible to use operators like +, -, ==, !=, >, <, >=,<=
- TimeSpan – a time interval (dd:hh:mm:ss)
  - ✓ Represents a period of time stored as a tick
  - ✓ Use Days, Hours, Minutes and Seconds properties, to get portion of that TimeSpan value

Perform Operations on Dates and Times > Example

```
DateTime dateTime = DateTime.Parse("1/28/2017 13:30");  //
DateTime dueDate = dateTime.AddMonths(2); //
dueDate = dateTime.AddDays(60); //
DateTime runTime = dateTime.AddMinutes(30); //
runTime = dateTime.AddHours(10); //
DateTime currentDate = DateTime.Today; //
DateTime deadLine = DateTime.Parse("3/15/2017"); //
TimeSpan timeTillDeadline = deadLine.Subtract(currentDate); //
int daysTillDeadline = timeTillDeadline.Days; //
TimeSpan timeTillDeadline = deadLine – currentDate; //
double totalDaysTillDeadline = timeTillDeadline.TotalDays; //
int hoursTillDeadline= timeTillDeadline.Hours; //
int minutesTillDeadline= timeTillDeadline.Minutes; //
int secondsTillDeadline= timeTillDeadline.Seconds; //
bool passedDeadline = false;
if(currentDate > deadLine ){
                        passedDeadline = true;
}
```

Format dates and times

- Use Format methods of the String class to format dates and times
- Standard DateTime formatting
  - ✓ d – Short date
  - ✓ D – Long date
  - ✓ t – Short time
  - ✓ T – Long Time
  - ✓ f – Lang date, short time
  - ✓ F – Long date, long time
  - ✓ g – Short date, short time
  - ✓ G – Short date, long time

Format dates and times …

Custom DateTime formatting

- d – Day of the month without leading zero
- dd – Day of the month with leading zero
- ddd – Abbreviated day name
- dddd – Full day name
- M – Month without leading zeros
- MM – Month with leading zero
- MMM – Abbreviated month name
- MMMM – Full month name
- y – two digit year without leading zero
- yy – two digit year with leading zero
- yyyy – four digit year
- / - Date separator
- h – Hour without leading zeros
- hh – Hour with leading zeros
- H – Hour on a 24-hour clock without leading zeros
- HH – Hour on a 24-hour with leading zeros
- m – Minutes without leading zeros
- mm – Minutes with leading zeros
- s – Seconds without leading zeros
- ss – Seconds without leading zeros
- f – Fractions of seconds (one f for each decimal place)
- t – First character of AM/PM
- tt – Full AM/PM
- : - Time separator

Format dates and times > Example

```
DateTime currDate = DateTime.Now;
string formattedDate = "";
formattedDate = String.Format("{0:d}", currDate);
formattedDate = String.Format("{0:D}", currDate);
formattedDate = String.Format("{0:t}", currDate);
formattedDate = String.Format("{0:T}", currDate);
formattedDate = String.Format("{0:ddd, MMM d, yyyy}", currDate);
formattedDate = String.Format("{0:M/d/yy}", currDate);
formattedDate = String.Format("{0:HH:mm:ss}", currDate);
```

Working with Strings

Processing and Manipulating Text Information

Working with Strings

- Working with characters in a string
- Working with string – creating string object from String class
- Use properties and methods of String class to work with string object
- StringBuilder class also provides another properties and methods to work with string

Properties and methods of String class

- Common properties and methods of String class
  - ✓ [index] – gets the character at the specified position
  - ✓ Length – gets the number of characters
  - ✓ StartsWith(string)
  - ✓ EndsWidth(string)
  - ✓ IndexOf(string[, startIndex])
  - ✓ LastIndexOf(string[, startIndex])
  - ✓ Insert(startIndex, string)
  - ✓ PadLeft(totalWidth) - Returns a new string that right-aligns the characters
  - ✓ PadRight(totalWidth) - Returns a new string that left-aligns the characters
  - ✓ Remove(startIndex, count)
  - ✓ Replace(oldString, newString)
  - ✓ Substring(startIndex[, length])
  - ✓ ToLower()
  - ✓ ToUpper()
  - ✓ Trim()
  - ✓ Split(splitCharacters)
  - ✓ Use an index to access each character in a string where 0 is the index for the first character, 1 the index for the second character, …

Working with String > Examples

```
string chars = “abcdef”;
char a = chars[0];
char b = chars[b];
string charAndSpace = “”;
for(int i = 0; i < chars.Length; i++)
        charAndSpace += chars[i] + “ ”;
Console.WriteLine(charAndSpace );
foreach(char c in chars)
        charAndSpace += c + “ ”;
Console.WriteLine(charAndSpace );
bool startsWith = chars.StartsWith(“abc”);
bool startsWith = chars.EndsWith(“abc”);
//
string univName= “Debre Berhan University”;
int index1 = univName.ToUpper();
```

Using StringBuilder class

- StringBuilder objects are mutable, means
  - ✓ they can be changed, deleted, replaced, modified
- Syntax
  - ✓ StringBuilder var = new StringBuilder([value] [,] [capacity]); // default capacity is 16 characters
- Indexer
  - ✓ [index] – Gets the character at the specified position
- Property
  - ✓ Length – Gets the number of characters in the string
  - ✓ Capacity – Gets or sets the number of characters the string can hold
- Method
  - ✓ Append(string) – Adds the specified strings to the end of the string
  - ✓ Insert(string, index) - Inserts the specified string at the specified index in the string
  - ✓ Remove(startIndex, count) – Removes the specified number of characters from the string starting at the specified index
  - ✓ Replace(oldString, newString) – Replaces all occurrences of the old string with the new string
  - ✓ ToString() - Converts the stringBuilder object to a string

StringBuilder > Example

- StringBuilder address1 = new StringBuilder();
- StringBuilder address2 = new StringBuilder(10);
- StringBuilder phone1= new StringBuilder(“0912345678”);
- StringBuilder phone2 = new StringBuilder(“0912345678”, 10);
- StringBuilder phoneNumber = new StringBuilder(10);
- phoneNumber.Append(“0912345678”);
- phoneNumber.Insert(0, “(+251)”);
- phoneNumber.Insert(4, “.”);
- phoneNumber.Replace(“.”, “-”);
- phoneNumber.Remove(0, 4);

Manipulating Strings

Comparing, Concatenating, Searching, Extracting Substrings, Splitting

Comparing Strings

- A number of ways exist to compare two strings:
  - ✓ Dictionary-based string comparison
    - > Case-insensitive
      - int result = string.Compare(str1, str2, true);
      - // result == 0 if str1 equals str2
      - // result < 0 if str1 is before str2
      - // result > 0 if str1 is after str2
    - > Case-sensitive
      - string.Compare(str1, str2, false);
- Equality checking by operator ==
  - ✓ Performs case-sensitive compare

```
> if (str1 == str2)
> {
>     …
> }
```

- Using the case-sensitive `Equals()` method
  - ✓ The same effect like the operator ==

```
> if (str1.Equals(str2))
> {
>     …
> }
```

Comparing Strings – Example

- Finding the first string in alphabetical order from a given list of strings:

```
string[] towns = {“Hawassa", “Dilla", “Adama",
     “Mekele", “Debre Berhan", “Dessie", “Gonder"};
string firstTown = towns[0];
for (int i=1; i<towns.Length; i++)
{
   string currentTown = towns[i];
   if (String.Compare(currentTown, firstTown) < 0)
   {
      firstTown = currentTown;
   }
}
Console.WriteLine("First town: {0}", firstTown);
```

Concatenating Strings

- There are two ways to combine strings:
  - ✓ Using the `Concat()` method
    - string str = String.Concat(str1, str2);
  - ✓ Using the + or the += operators
    - string str = str1 + str2 + str3;
    - string str += str1;
  - ✓ Any object can be appended to string
    - string name = “Lemma";
    - int age = 22;
    - string s = name + " " + age; // Lemma 22

Concatenating Strings – Example

```
string firstName = “Soliana";
string lastName = “Abesselom";

string fullName = firstName + " " + lastName;
Console.WriteLine(fullName);// Soliana Abesselom

int age = 25;
string nameAndAge ="Name: " + fullName + "\nAge: " + age;
Console.WriteLine(nameAndAge);
// Name: Soliana Abesselom
// Age: 25
```

Searching in Strings

- Finding a character or substring within given string
  - ✓ First occurrence
    - > IndexOf(string str);
  - ✓ First occurrence starting at given position
    - > IndexOf(string str, int startIndex)
  - ✓ Last occurrence
    - > LastIndexOf(string)
  - ✓ IndexOf is case-sensetive

Searching in Strings > Example

- ✓ string str = "C# Programming Course";
- ✓ int index = str.IndexOf("C#");
- ✓ index = str.IndexOf("Course");
- ✓ index = str.IndexOf("COURSE");
- ✓ index = str.IndexOf("ram");
- ✓ index = str.IndexOf("r");
- ✓ index = str.IndexOf("r", 5);
- ✓ index = str.IndexOf("r", 8);

Extracting Substrings

- Extracting substrings
  - ✓ `str.Substring(int startIndex, int length)`
    - string filename = @"C:\Pics\bird2009.jpg";
    - string name = filename.Substring(8, 8);
  - ✓ `str.Substring(int startIndex)`
    - string filename = @"C:\Pics\Summer2009.jpg";
    - string nameAndExtension = filename.Substring(8);
    - // nameAndExtension is Summer2009.jpg

Splitting Strings

- To split a string by given separator(s) use the following method:
  - ✓ string[] Split(params char[])

```
string listOfBeers = “Bedelle, Habesha Raya, Dashen Giorgis, Meta";
string[] beers = listOfBeers.Split(' ', ',', '.');
Console.WriteLine("Available beers are:");
foreach (string beer in beers)
{
    Console.WriteLine(beer);
}
```

Trimming White Space

- Using method `Trim()`
  - ✓ string s = " example of white space ";
  - ✓ string clean = s.Trim();
  - ✓ Console.WriteLine(clean);
- Using method `Trim(chars)`
  - ✓ string s = " \t\nHello!!! \n";
  - ✓ string clean = s.Trim(' ', ',' ,'!', '\n','\t');
  - ✓ Console.WriteLine(clean); //
- Using `TrimStart()` and `TrimEnd()`
  - ✓ string s = " C# ";
  - ✓ string clean = s.TrimStart(); //

For more information

- http://www.tutorialspoint.com/csharp/
- 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?