

(420-PS4-AB)

C# Language Introduction

Aref Mourtada

Fall 2017

Outline

- About
- Data Types & Variables
- Arithmetic Operators
- Escape Characters
- Comparison Operators
- Control Flow Statements
- Loops
- Methods
- Namespaces
- Classes

Getting Started with C#

- C# is a modern Object oriented programming language.
- Object oriented programming (OOP) is a programming paradigm using "objects" - data structures consisting of **Data fields** & **Methods**
- Programming techniques may include many features.
 - Data abstraction
 - Encapsulation
 - Messaging
 - Modularity
 - Polymorphism
 - Inheritance.

Data Types and Variables (1)

- Boolean :

```
bool flag = true;
```

- Numeric

```
int, long, float, double, decimal
```

- Characters

```
char, string
```

- Arrays

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

```
int[] myOtherNumbers = {10, 20, 30};
```

Data Types and Variables (2)

- Lists

```
List<string> roles = new List<string>();  
roles.Add("Administrators");  
roles.Add("ContentManagers");  
roles.Add("Members");
```

```
List<int> myN = new List<int>() {10, 20, 30};
```

- Container

```
var
```

Converting and Casting Data Types

- Convert class

Example : `Convert.ToBoolean("True") ;`

- Parse function

Example : `Int32.Parse(numberTxt.Text()) ;`

- For String: ToString function

Example : `string str = count.ToString() ;`



Arithmetic Operators

Operator	Function
+	Adds two values to each other
-	Subtracts one value from another
*	Multiplies two values
/	Divides two values
%	Divides two whole numbers and returns the remainder

Escape Characters

Escape Sequence	Meaning
\'	Single Quote
\"	Double Quote
\\	Backslash
\0	Null, not the same as the C# <i>null</i> value
\a	Bell
\b	Backspace
\f	form Feed
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab

Comparison Operators

Operator	Symbol
Equal	==
Not Equal	!=
Greater than	>
Equal or greater than	>=
Less than	<
Equal or less than	<=

- User only with primitive data types.
- No need for operators with boolean data types variables.
- To compare objects use methods.
 - Strings

Control Flow: 'if' Statements

- `if (condition)`
 `action;`
- `if (condition)`
 `action1;`
 `else`
 `action2;`

- **Nested:**
 `if (condition1)`
 `action1;`
 `else if (condition2)`
 `action2;`
 `else`
 `action3;`

Control Flow: 'switch' Statements

```
switch (myTest)
{
    case 1:
        MessageBox.Show("Hello1");
        break;

    case 2:
        MessageBox.Show("Hello2");
        break;

    default:
        MessageBox.Show("Hello3");
        break;
}
```

Loops: while

- Pretest loop.


```
while (condition)
{
    action;
    ...
}

statement;
```

Loops: Do while

- Posttest loop.

```
do  
{  
    action;  
    ...  
}  
while (condition) ;  
statement;
```



Loops: for

- Pretest loop.

```
for (initialization; condition; update statement;)
{
    loop statements;
}
statement;
```

Loops: foreach

- Iterator loop: iterates through the items in a list.
- It operates on arrays or collections.

```
string[] names ={"tom", "alex", "hello world"};  
foreach(string person in names)  
{  
    MessageBox.show(person) ;  
}  
statement;
```

Methods

- Methods are commonly used to break a problem down into small manageable pieces.

This is called *divide and conquer*.

- Methods simplify programs.
 - If a specific task is performed in several places in the program
 - A method can be written once to perform that task
 - And then be executed anytime it is needed.

This is known as *code reuse*

Method Structure

- To create a method, you must write a definition, which consists of:

1. Header

- Appears at the beginning of a method definition
- Lists several important things about the method
Including the method's name.

2. Body

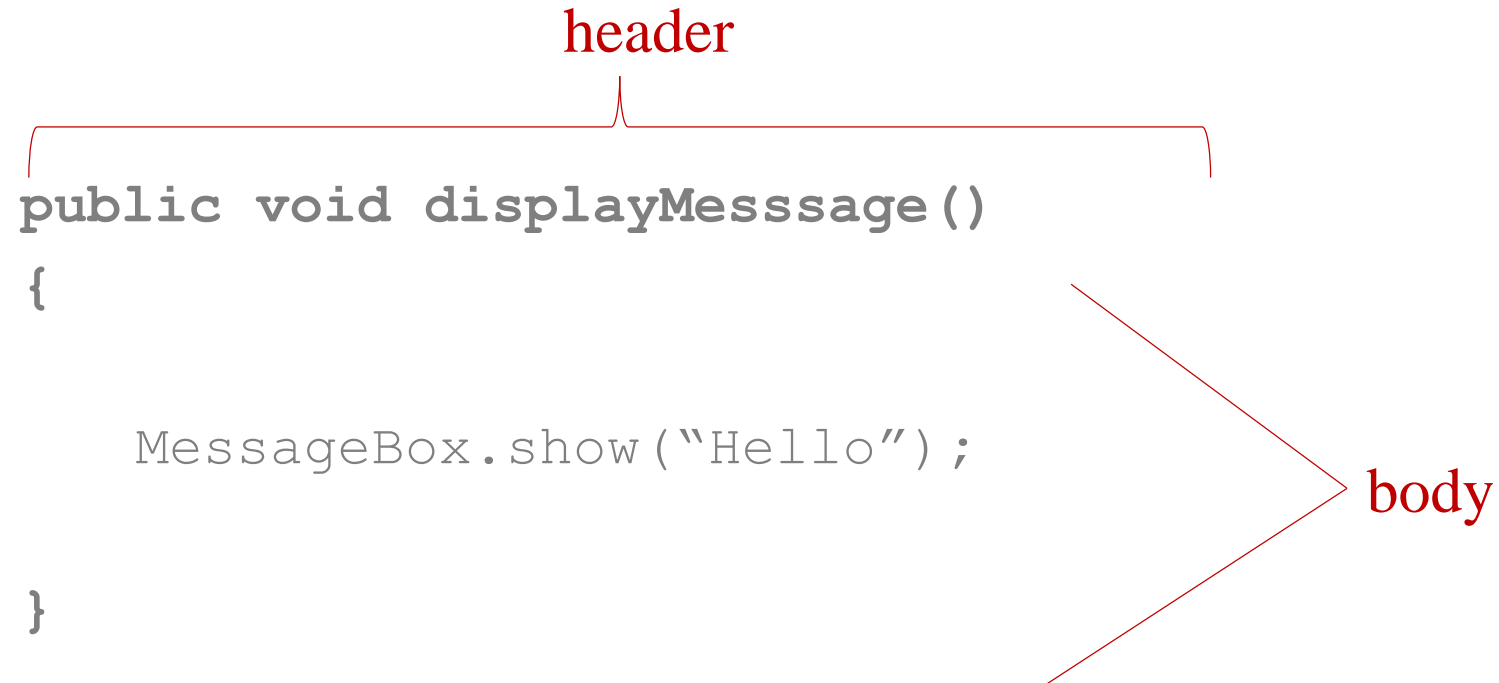
- The method body is a collection of statements that are performed when the method is executed.

Two Parts of Method Declaration

header

```
public void displayMesssage()  
{  
    MessageBox.show("Hello");  
}
```

body



Parts of a Method Header

Method Modifiers	Return Type	Method Name	Parentheses
<code>public</code>	<code>void</code>	<code>displayMessage</code>	<code>()</code>

```
{  
  
    MessageBox.show("Hello") ;  
  
}
```

Parts of a Method Header

- Method modifiers
 - **public**—method is publicly available to code outside the class
- Return type
 - void or the data type from a value-returning method
- Method name
 - name that is descriptive of what the method does
- Parentheses
 - contain nothing or a list of one or more variable declarations if the method is capable of receiving arguments.

Calling a Method

- A method executes when it is called.
- The Page_Load method is automatically called when an Asp .NET page starts, but other methods are executed by method call statements.

```
displayMessage ( ) ;
```

- Notice that the method modifiers and the void return type are not written in the method call statement. Those are only written in the method header.

Documenting Methods

- A method should always be documented by writing comments that appear just before the method's definition.
- The comments should provide a brief explanation of the method's purpose.
- The documentation comments begin with `/**` and end with `*/`.

Passing Arguments to a Method

- Values that are sent into a method are called arguments.

```
MessageBox.show("Hello");  
number = Int32.parse("123");
```

- The data type of an argument in a method call must correspond to the variable declaration in the parentheses of the method declaration. The parameter is the variable that holds the value being passed into a method.
- By using parameter variables in your method declarations, you can design your own methods that accept data this way.

Arguments are Passed by Value

- All arguments of the primitive data types are passed by value, which means that only a copy of an argument's value is passed into a parameter variable.
- A method's parameter variables are separate and distinct from the arguments that are listed inside the parentheses of a method call.
- If a parameter variable is changed inside a method, it has no affect on the original argument.
- A class type variable does not hold the actual data item that is associated with it, but holds the memory address of the object.
 - A variable associated with an object is called a reference variable.

Returning a Value from a Method

- Data can be passed into a method by way of the parameter variables. Data may also be returned from a method, back to the statement that called it.

```
int num = Int32.parse("700");
```

- The string "700" is passed into the parseInt method.
- The `int` value 700 is returned from the method and stored into the `num` variable.

Namespaces

- Namespaces are C# program elements designed to help you organize your programs.
- They also provide assistance in avoiding name clashes between two sets of code.
- Implementing Namespaces in your own code is a good habit because it is likely to save you from problems later when you want to reuse some of your code.
- You specify the Namespaces you want to use in the top of your code.

Namespaces

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;
```

Classes

- The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling.
- A real instance of a class is called an “object” or an “instance of a class”.

Classes: Constructor

- The purpose of constructors is to initialize class members when an instance of the class is created.

```
class Car
{
    public string color;    //Field

    //Constructor - Used to initialize the Class
    public Car()
    {
        color="green";
    }
    //Constructor - with parameter
    public Car(string initColor)
    {
        color=initColor;
    }
}

..
Car myCar = new();
Car myCar2 = new("red");
```

Classes: Properties

- Properties provide the opportunity to protect a field in a class by reading and writing to it through the property.

```
class Car
{
    public string Name{get;set;}
    public string color{get;set;}

    public void ShowCarProperties()
    {
        color="green";
    }
}

...
Car myCar = new Car ();
myCar.Name="Volvo";
myCar.Color="Blue";
```

Class Members

- Constructors
- Destructors (opposite of Constructors)
- Fields
- Methods
- Properties
- Indexers
- Delegates
- Events
- Nested Classes

Q & A

