

Chapter 3 - Introduction to C# Programming

1

- 3.1 Introduction
- 3.2 Simple Program: Printing a Line of Text
- 3.3 Another Simple Program: Adding Integers
- 3.4 Memory Concepts
- 3.5 Arithmetic
- 3.6 Decision Making: Equality and Relational Operators

Lecture by Fan Wu, MIS, CCU

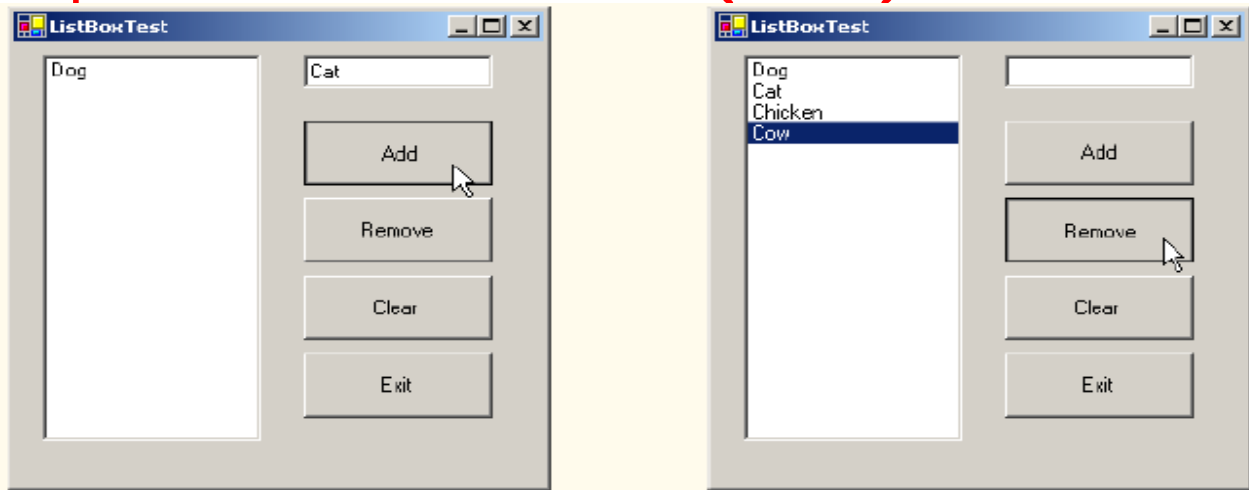


3.1 Introduction

- Console applications
 - No visual components
 - Only text output



- Windows applications
 - Forms in a familiar window style
 - Contain Graphical User Interfaces (GUIs)



3.2 Simple Program: Printing a line of text

3

A two-line comment, starting with double slash (//)

```
// Fig. 3.1: Welcome1.cs  
// A first program in C#.
```

being ignored by the compiler; only used for remark

```
using System;
```

```
class Welcome1
```

```
{  
    static void Main( string[] args )
```

```
{
```

```
    Console.WriteLine( "Welcome to C# Programming!" );
```

```
}
```

```
}
```

Using directive to let the compiler know what it should include (import) from the library (here is **System** class)

Adding space lines at will (for clearness only)



```
// Fig. 3.1: Welcomel.cs  
// A first program in C#.
```

- Why we need comments in a program?
 - Since people are forgetful
- How to differentiate the comments and instructions?
 - We use special symbol, like "//", to denote the comments
 - rules for comments if using "//"
 - Two back slashes must be adjacent, no space between them
 - Comments starting from the symbol "//" , extending until to the end of the line
 - All the contents, no matter what it is, after "//" are skipped by compiler
 - The above are two single-line comments
- How many ways to write a comment?
 1. Single-line comment
 2. Multiple-line comment

Multiple-line comment

- Multiple-line comment (not encouraged to use it)
 - The comments can across several lines
 - comment starts from special symbol `/*` and end until encouting `*/`
 - No space in the two characters
- Example of multiple-line comment

```
/* This is a multiple-line  
comment. It can be  
split over many lines */
```

- Pros and cons of multiple-line comment
 - Pros: a pair of symbols can express a big comment
 - Cons: some unexpected condition may occur if
 - the comment has the ending symbol in it
 - We forget an ending symbol
 - The code has the staring symbol (since of miss)



Cons of Multiple-line comment

- Example 1:

a = 5;

.....

/* this comment uses “ /* ”, “ */ ” to
embrace it */

- Compiler error
 - the comment is

/* this comment uses “ /* ” and “ */

- Example 2:

a = 5;

.....

/* this comment is

.....

b = 6;

.....

/* new comment ... */

- No error, but
 - the comment is

/* this comment is

.....

b = 6;

.....

... new comment ... */



Cons of Multiple-line comment

- Example 3:

```
a = 5;
```

```
.....
```

```
b = 6 /* 5;
```

```
.....
```

```
/* this comment .....
```

```
..... */
```

```
b = 7;
```

- Compile Error,
 - but the comment is

```
/* 5;
```

```
.....
```

```
/* this comment .....
```

```
..... */
```

2.2 A Simple Program: Printing a Line of Text

8

- **Example:**

- **Expected:**

```
j=j+4; /* comment 1 start
comment 1 will miss the right end of comment
j= k+ j; /* comment 2 start
comment 2 */
```

- **But Executed:**

```
j=j+4; /* comment 1 start
comment 1 will miss the right end of comment
j= k+ j; /* comment 2 start
comment 2 */
```

- **Suggested:**

```
j=j+4; // comment 1 start
//comment 1 will miss the right end of comment
j= k+ j; // comment 2 start
//comment 2
```


Using

- Some methods, like i/o methods, are provided by others
 - You just need to use (i.e., using) it, instead of developing it by yourself

```
using System;  
  
class Welcome1  
{  
    static void Main( string[] args )  
    {  
        Console.WriteLine( "Welcome to C# Programming!" );  
    }  
}
```

- If using a package,
 - You can utilize the static method of the classes in the package (discussed later)
 - Ex: "System" package has a console class that has write(), writeline(), ..., static methods to be called

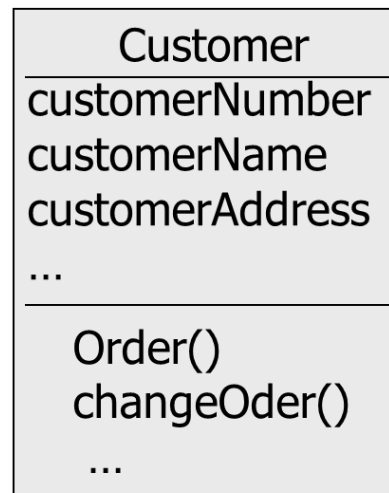
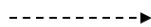


What is class?

- Modern programming groups the **related** codes and data in a unit (i.e., class) for easily processing
- A class simulates the real entity of the world with
 - Attributes to simulate entity's properties,**
 - like CustomerNumber, name, address of a customer's entity
 - Methods to simulate entity's actions,**
 - like order(), changeOrder()



Physical reality -
Person who is a
customer



a class Customer
(DB term "Entity")

- Class name

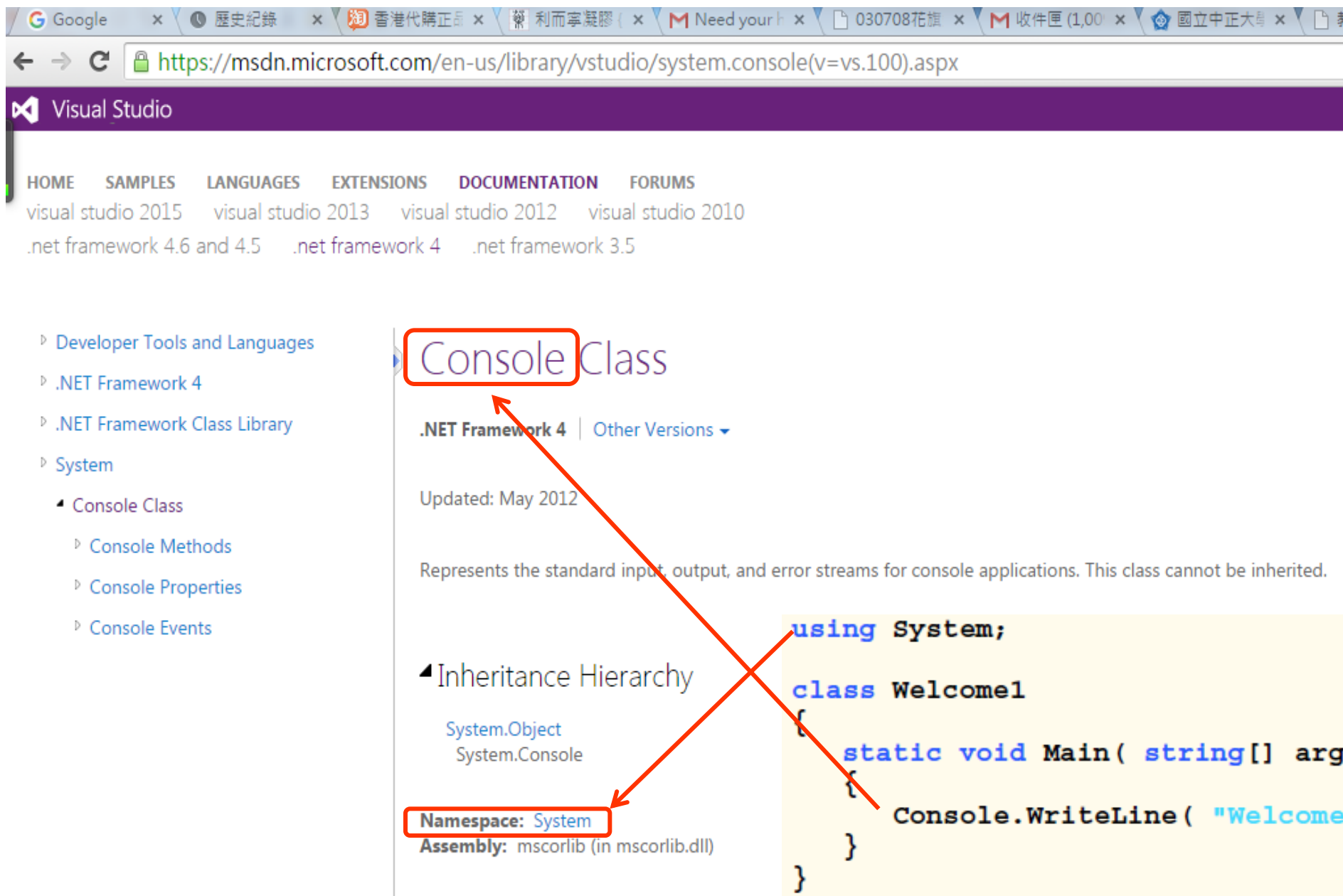
- Class attributes

- Class behaviors
(methods,
processes/functions
working with
data)

We can define a
class, define its
attributes, and
define its methods

We can use the
attributes of a class,
and call the
methods of a class

MSDN usage for console class (1/2)



Visual Studio

HOME SAMPLES LANGUAGES EXTENSIONS DOCUMENTATION FORUMS

visual studio 2015 visual studio 2013 visual studio 2012 visual studio 2010

.net framework 4.6 and 4.5 .net framework 4 .net framework 3.5

- Developer Tools and Languages
- .NET Framework 4
- .NET Framework Class Library
- System
 - Console Class
 - Console Methods
 - Console Properties
 - Console Events

Console Class

.NET Framework 4 | Other Versions ▾

Updated: May 2012

Represents the standard input, output, and error streams for console applications. This class cannot be inherited.

Inheritance Hierarchy

- System.Object
 - System.Console

Namespace: System










Assembly: mscorlib (in mscorlib.dll)

```
using System;

class Welcome1
{
    static void Main( string[] arg
    {
        Console.WriteLine( "Welcome
    }
}
```

MSDN usage for console class (2/2)

Methods

	Name	Description
	WriteLine(String)	Writes the specified string value, followed by the current line
	WriteLine(UInt32)	Writes the text representation of the specified 32-bit unsigned integer
	WriteLine(UInt64)	Writes the text representation of the specified 64-bit unsigned integer
	WriteLine(String, Object)	Writes the text representation of the specified object, followed by the specified format information.
	WriteLine(String, Object[])	Writes the text representation of the specified array of objects, followed by the specified format information.
	WriteLine(Char[], Int32, Int32)	
	WriteLine(String, Object, Object)	
	WriteLine(String, Object, Object, Object)	
	WriteLine(String, Object, Object, Object, Object)	writes the text representation of the specified objects and the specified format information to the standard output stream using the specified format information.

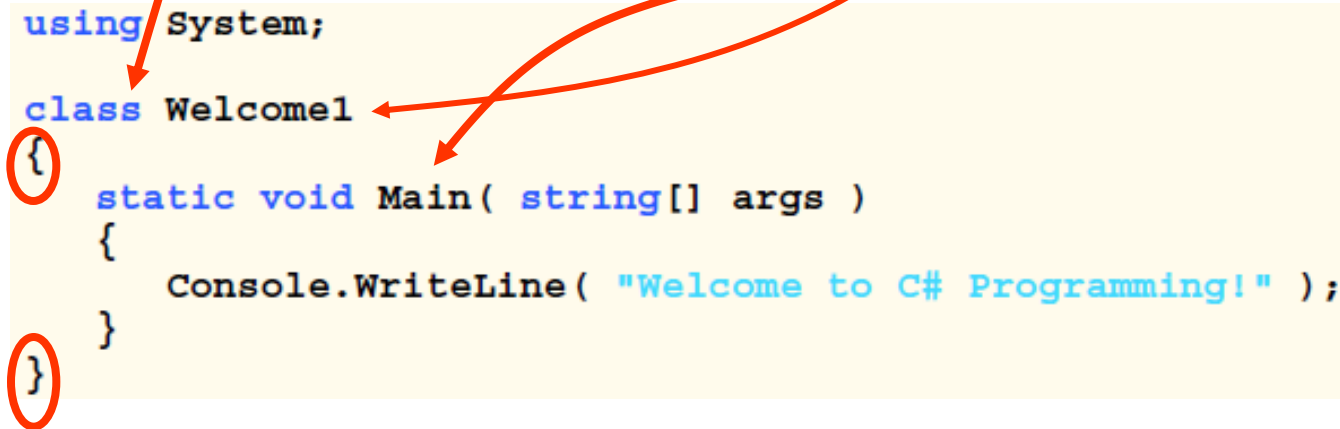
```
using System;

class Welcome1
{
    static void Main( string[] args )
    {
        Console.WriteLine( "Welcome to C# Program" );
    }
}
```

Class definition & its example

- Classes definition
 - Defined using keyword **class** and the **class name** (named by user)
 - Have a body delimited with curly braces (**{** and **}**)
- A program can have more than one class definitions
 - But, what class the program will start to execute?
 - From the class with method `main()`, and the start of execution is from method `main()`

```
using System;  
  
class Welcome1  
{  
    static void Main( string[] args )  
    {  
        Console.WriteLine( "Welcome to C# Programming!" );  
    }  
}
```

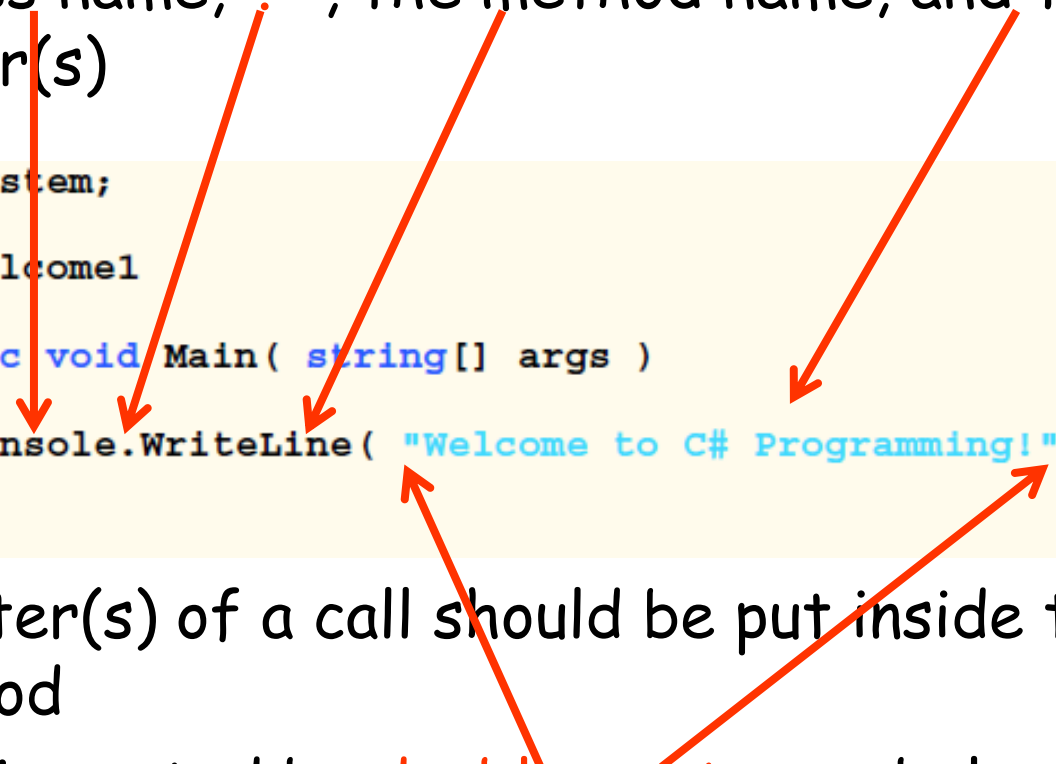


How to call static method (syntax)?

- When we **call** a static method of a class, we should add the class name, `'`, the method name, and the parameter(s)

```
using System;

class Welcome1
{
    static void Main( string[] args )
    {
        Console.WriteLine( "Welcome to C# Programming!" );
    }
}
```



- The parameter(s) of a call should be put inside the brace of the method
 - A string is quoted by **double-quote** symbol
 - Cf: **single-quote** is for a character, like 'w', 'c', etc



Static & public method in class

- For information hiding, not all methods in a class can be called by other classes
- If a method is defined as **static & public** method in a class,
 - the method can be called at any time, after using its package

```
using System;

class Welcome1
{
    static void Main( string[] args )
    {
        Console.WriteLine( "Welcome to C# Programming!" );
    }
}
```

Welcome to C# Programming!



3.2 Simple Program: Printing a line of text in a message box

- Graphical User Interface (GUI)
 - GUIs are used to make it easier to get data from the user as well as display data to the user
 - Message boxes
 - Within the **System.Windows.Forms** namespace
 - Used to prompt or display information to the user




```
1 // Fig. 3.7: Welcome4.cs
2 // Printing multiple lines in a dialog Box.
3
4 using System;
5 using System.Windows.Forms;
6
7 class Welcome4
8 {
9     static void Main( string[] args )
10    {
11        MessageBox.Show( "Welcome\nto\nC#\nprogramming!" );
12    }
13 }
```

System.Windows.Forms namespace allows the programmer to use the MessageBox class.



This will display the contents in a message box as opposed to in the console window.

Escape sequence	Description
<code>\n</code>	Newline. Position the screen cursor to the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the previous characters output on that line.
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\"</code>	Double quote. Used to print a double quote (") character.



```
4 using System;
6 class Addition {
8     static void Main( string[] args ) {
10         string firstNumber, secondNumber;
13         int number1,
14             number2, sum;
18         Console.Write( "Please enter the first integer: " );
19         firstNumber = Console.ReadLine();
22         Console.Write( "\nPlease enter the second integer: " );
23         secondNumber = Console.ReadLine();
26         number1 = Int32.Parse( firstNumber );
27         number2 = Int32.Parse( secondNumber );
30         sum = number1 + number2;
33         Console.WriteLine( "\nThe sum is {0}.", sum );
35     }
37 }
```

Please enter the first integer: 45

Please enter the second integer: 72

The sum is 117.

Variable & its definition

```
string firstNumber, secondNumber;  
int number1,  
    number2, sum;
```

- Variable
 - For a variable, the computer will use **a memory** to store its value (discussed later)
 - A variable can be named by programmer, but **avoiding from using keywords**, like "if ", "else" (discussed later)
 - The name of a variable should begin with alphabet and then followed by no special characters, like ?, ^, +
 - the variable name is **case-sensitive**



Case sensitive

- What is case sensitive (capitalization matters) ?
 - **a1** and **A1** are different
- Why C# provides case-sensitive, but VB, SQL does not
 - Advantage:
 - It can provide more variable's naming space
 - Some implicit rules:
 - first-capital variable as global variable, like **Sum**,
 - small-letter variable as local variable, like **sum**
 - all-capital variable as constant variable, like **SUM**
 - Disadvantage:
 - a non-sense users may be frustrated while compiler replies a lot of variable undefined message since of the misuse of case.



Variable definition

- Variable declaration is needed in strongly-typed language, like C, Delphi, Java, ...
 - Advantages: Increase readability, reliability
 - Disadvantage: troublesome
- Variable declaration is not always needed in weakly-typed language, like Basic, VB, Fortran
 - Advantage: easy;
 - Disadvantage: may cause unexpected errors
- Ex: In Fortran (variables declaration is not necessary), if you add 1 to 10, and store the result to variable i. The following codes can pass the compiler

```
for (i = 1 to 10)
```

```
  j = i+1;
```

```
.....
```



Adding Integers

- `Console.ReadLine()`
 - Used to **get a value from the user input, which is seen as a string**
- `Int32.Parse()`
 - Used to **convert a string argument to an integer**
 - **This method is a static method of Class `Int32`, which (the class) exists in `System` package**

```
Console.Write( "Please enter the first integer: " );  
firstNumber = Console.ReadLine();  
Console.Write( "\nPlease enter the second integer: " );  
secondNumber = Console.ReadLine();  
number1 = Int32.Parse( firstNumber );  
number2 = Int32.Parse( secondNumber );
```

```
Please enter the first integer: 45
```

```
Please enter the second integer: 72
```

```
The sum is 117.
```

Adding Integers

```
sum = number1 + number2;
```

- Assignment (=) statement
 - Calculates sum of **number1** and **number2** (right hand side)
 - Uses assignment operator = to assign result to variable **sum**
 - **number1** and **number2** are operands
 - Read as:
 - **sum** is assigned by the value of **number1 + number2**
- '=' used in Java, C#, ..., it means assignment, not equal to if it is "equal to", then chaos.

Ex: ~~j~~ = ~~j~~ + 1;
=> 0 = 1;



Adding Integers

```
sum = number1 + number2;  
Console.WriteLine( "\nThe sum is {0}.", sum );
```

- Here, {0} used in a string to reserve the position of the later variable's value in a string
 - 0 is the first later variable
- Example

```
Console.WriteLine(  
    "The numbers entered are {0} and {1}", number1, number2 );
```

- Output

"The numbers entered are 45 and 72"



Arithmetic

C# operation	Arithmetic operator	Algebraic expression	C# expression
Addition	+	$f + 7$	<code>f + 7</code>
Subtraction	-	$p - c$	<code>p - c</code>
Multiplication	*	bm	<code>b * m</code>
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>
Modulus	%	$r \bmod s$	<code>r % s</code>

- **Modulus operator %** returns the remainder
 $7 \% 5$ evaluates to 2
- **Integer division truncates remainder**
 $7 / 5$ evaluates to 1
 But $7.0/5$ evaluates to 1.4



comparison

Standard algebraic equality operator or relational operator	C# equality or relational operator	Example of C# condition	Meaning of C# condition
---	------------------------------------	-------------------------	-------------------------

Equality operators

$==$	<code>==</code>	<code>x == y</code>	x is equal to y
\neq	<code>!=</code>	<code>x != y</code>	x is not equal to y

Relational operators

$>$	<code>></code>	<code>x > y</code>	x is greater than y
$<$	<code><</code>	<code>x < y</code>	x is less than y
\geq	<code>>=</code>	<code>x >= y</code>	x is greater than or equal to y
\leq	<code><=</code>	<code>x <= y</code>	x is less than or equal to y

