158.212 Application Software Development

Suriadi Suriadi

Staff

Lecturer:

Suriadi Suriadi

Consultation time:

Tuesday 14:00 – 16:00 (throughout summer semester, excluding Christmas/New Year holiday)

Room 19a, Building 106, OR

Tutors:

Indu Sofat

Tong Liu

Timetable

Weekly:

Monday	- Lecture	12:00-14:00	(AT6)
	- Lab	15:00-17:00	(CLQB4)
Thursday	- Lecture	09:00-11:00	(AT6)

Teaching Semester:

24th November– 15th December (roughly 3 weeks)

and

5th January – 2nd January (roughly 3 weeks)

Key Learning Outcomes

1. Understand and apply fundamental software engineering principles, e.g. loose coupling and high cohesion between and within modules
2. Apply event-driven solutions using a combination of object and procedural paradigms and visual programming in integrated development environment.
3. Learn the use of structured exception handling and debugging for developing more robust programs. (e.g. null pointer exception)
4. Understand and apply basic object-oriented principles, e.g. encapsulation, polymorphism, and inheritances.
5. Gain practical skills by using a visual programming language.

Course Plan

1. .NET Framework and Basic programming – variables, types, conversion
2. Basic programming – flow control, conditions, loops
3. Basic programming – functions, parameters, scope, debugging
4. Object oriented programming (Part 1) and Visual Studio IDE
5. Refactoring and Controls/Events – adding controls
6. Test Driven Development and Controls/Events
7. Input validation, exception handling, dialog boxes
8. Basic data structures – array, list, dictionary
9. Object oriented programming (Part 2)
10. Multiple Forms – interaction and navigation
11. Object oriented programming (Part 3)
12. Object oriented programming (Part 4)
13. Course Review

Check the lecture and tutorial schedule in Stream

Assessment

The course is assessed in two parts:

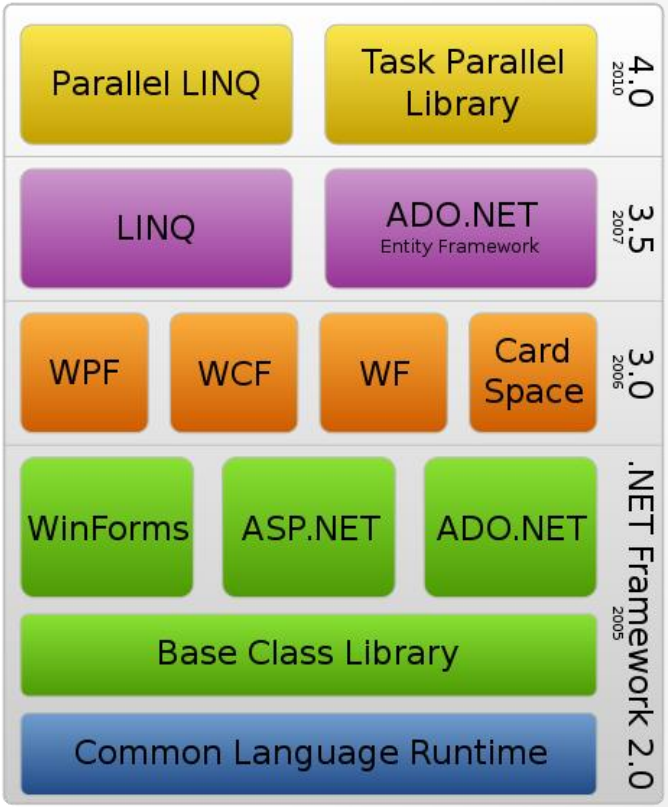
Five assignments due Weeks 2-6	40%
Final exam	60%

Lecture 1

.NET Framework

Proprietary technology

Implementation of the Common Language Infrastructure (CLI) specification



The .NET Framework Stack

.NET Framework

- .NET applications do not execute directly on the hardware,

- Instead, in a software environment known as the Common Language Runtime (CLR).

- This virtual machine provides a number of services such as security, memory management and exception handling.

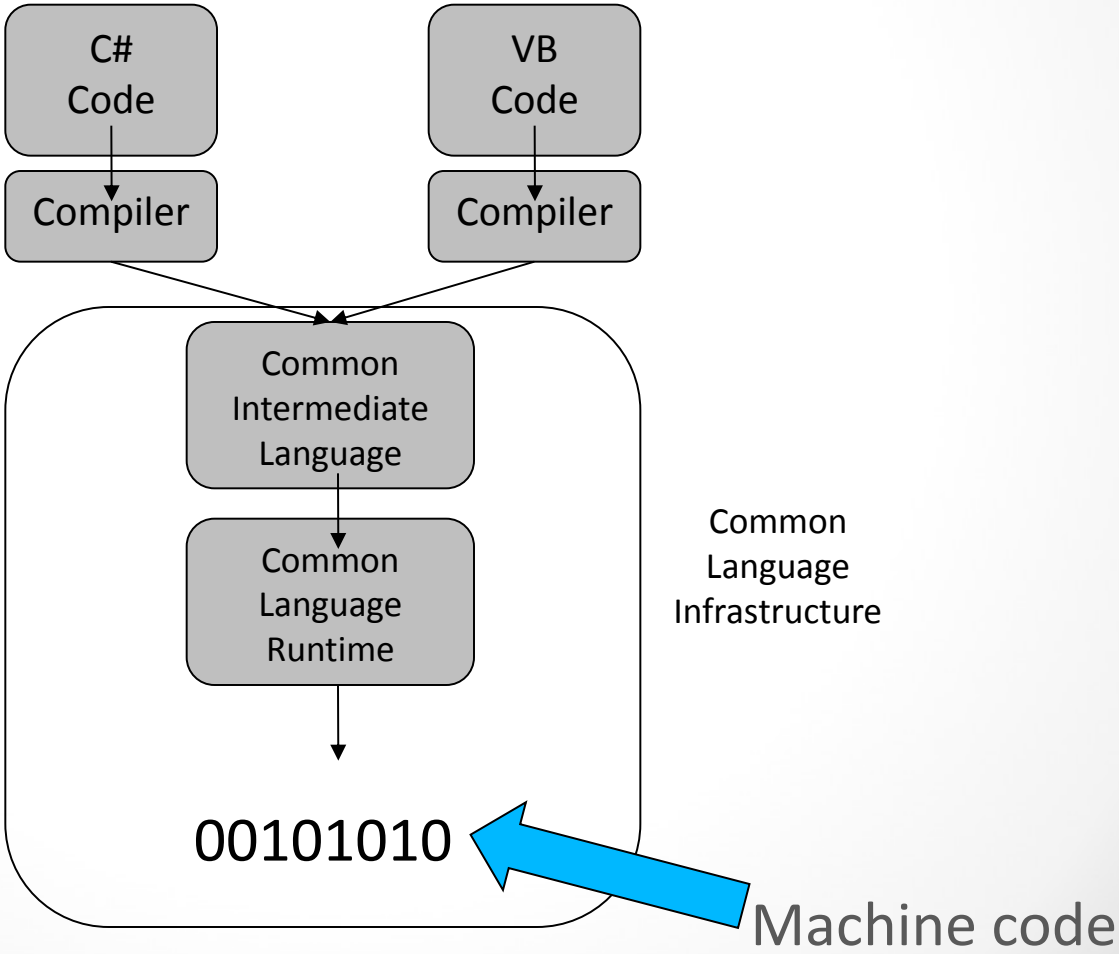
.NET Framework

- Consequently, when an application is compiled, it is
  - not compiled to machine code, but
  - to something known as Common Intermediate Language(CIL).

- CLR then executes CIL-compiled applications which
  - convert the CIL into machine code, and only then
  - executes on the hardware.

Common Language Infrastructure .NET Framework

Java vs. .NET ?



.NET Framework

There are a few advantages of this approach.

- Interoperability
  - Compiled into common CIL,
  - Access to methods written in different languages

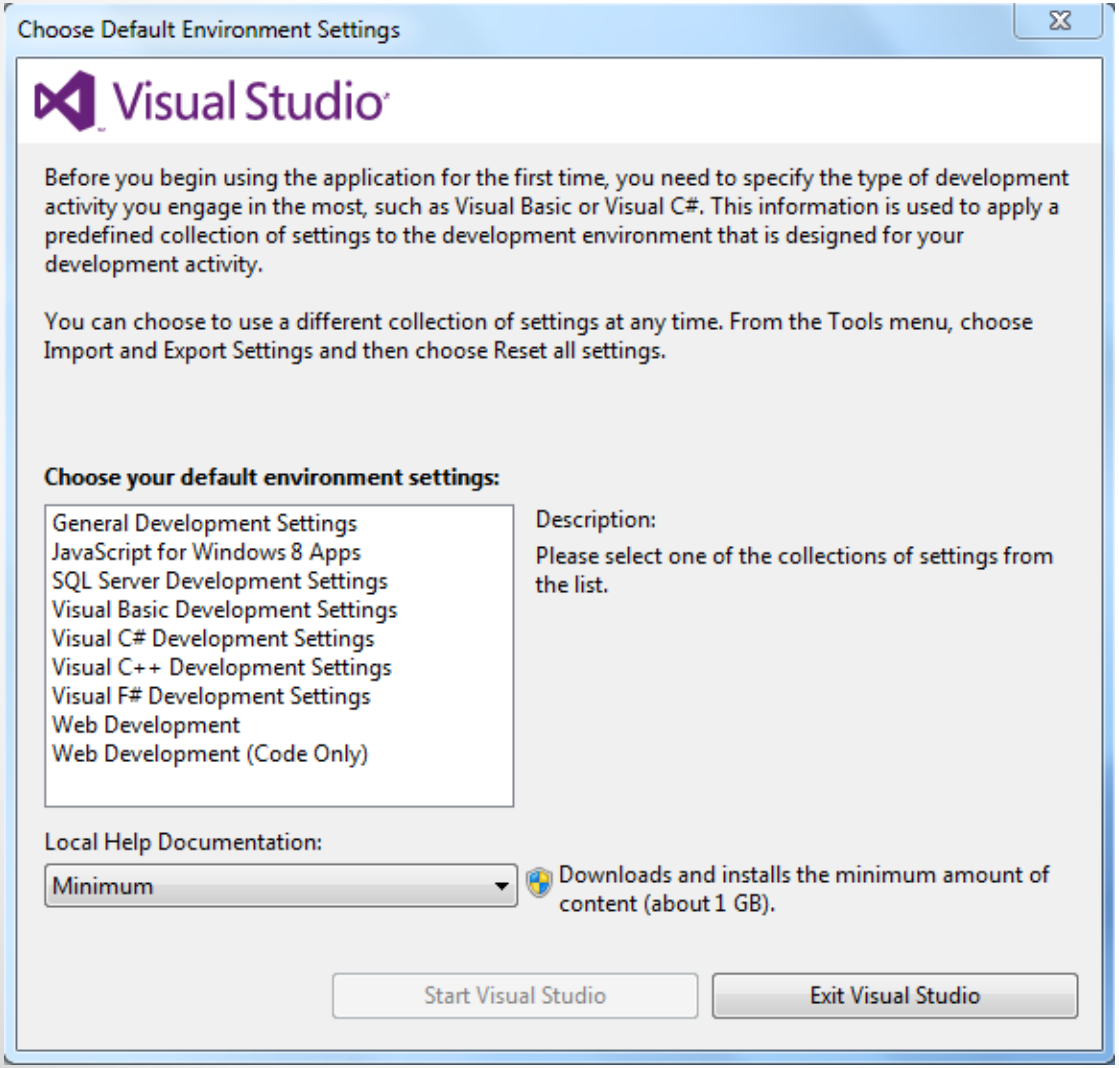
- Common Language Runtime –
  - Common runtime services, e.g. memory management, security, exception handling, and type safety (syntactically and semantically acceptable).
    - e.g. Chomsky's "*Colorless green ideas sleep furiously*"

.NET Framework

There are a few major advantages of this approach.

- Portability
  - Applications execute on the CLR (downloadable), and not the hardware.
  - Theoretically, on any platforms,
    - but only on Windows so far.
  - Other platforms: `adapter' needed (e.g. *Wine* or Mono on Linux)
- Other advantages?
- Disadvantages?

Creating a Command Line Program in Visual Studio

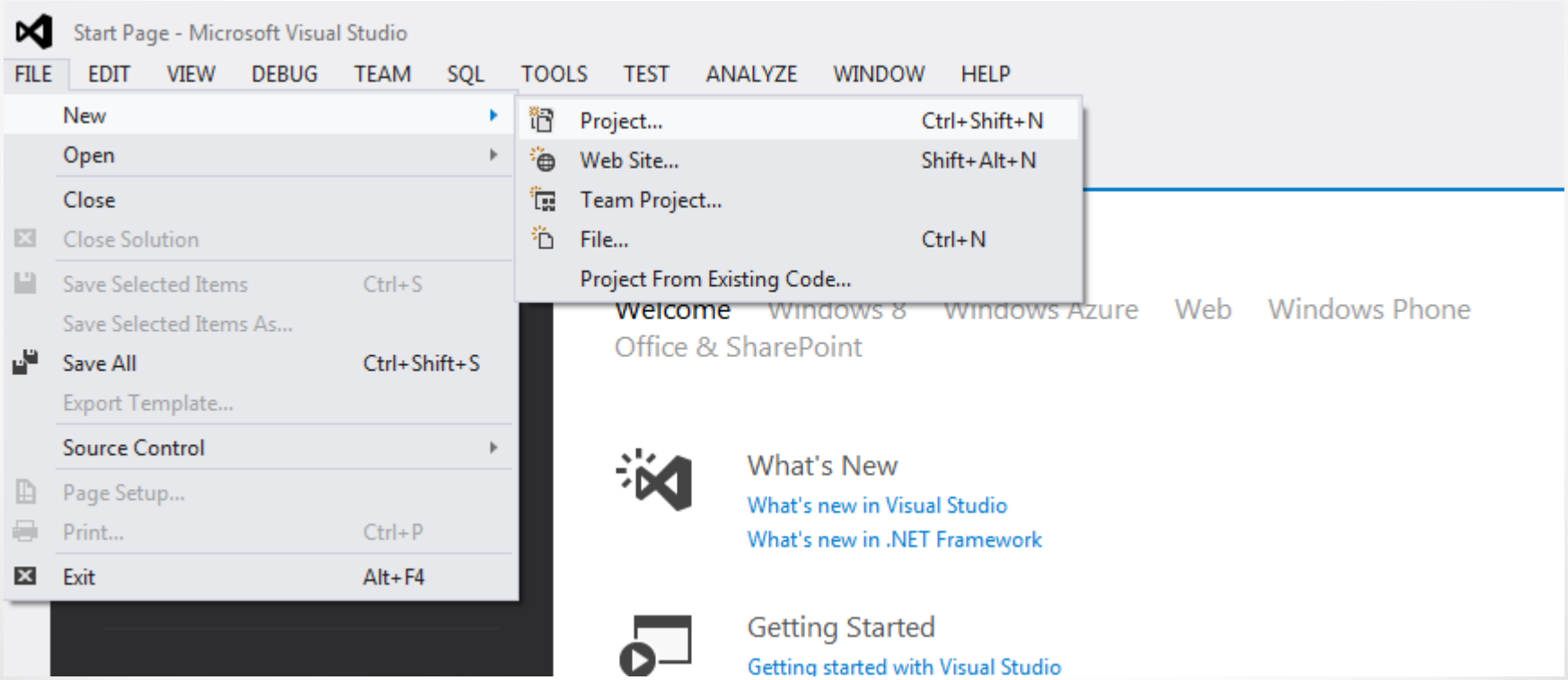


Set your default environment
(for convenience only)

Let's choose "Visual C# Development Settings"

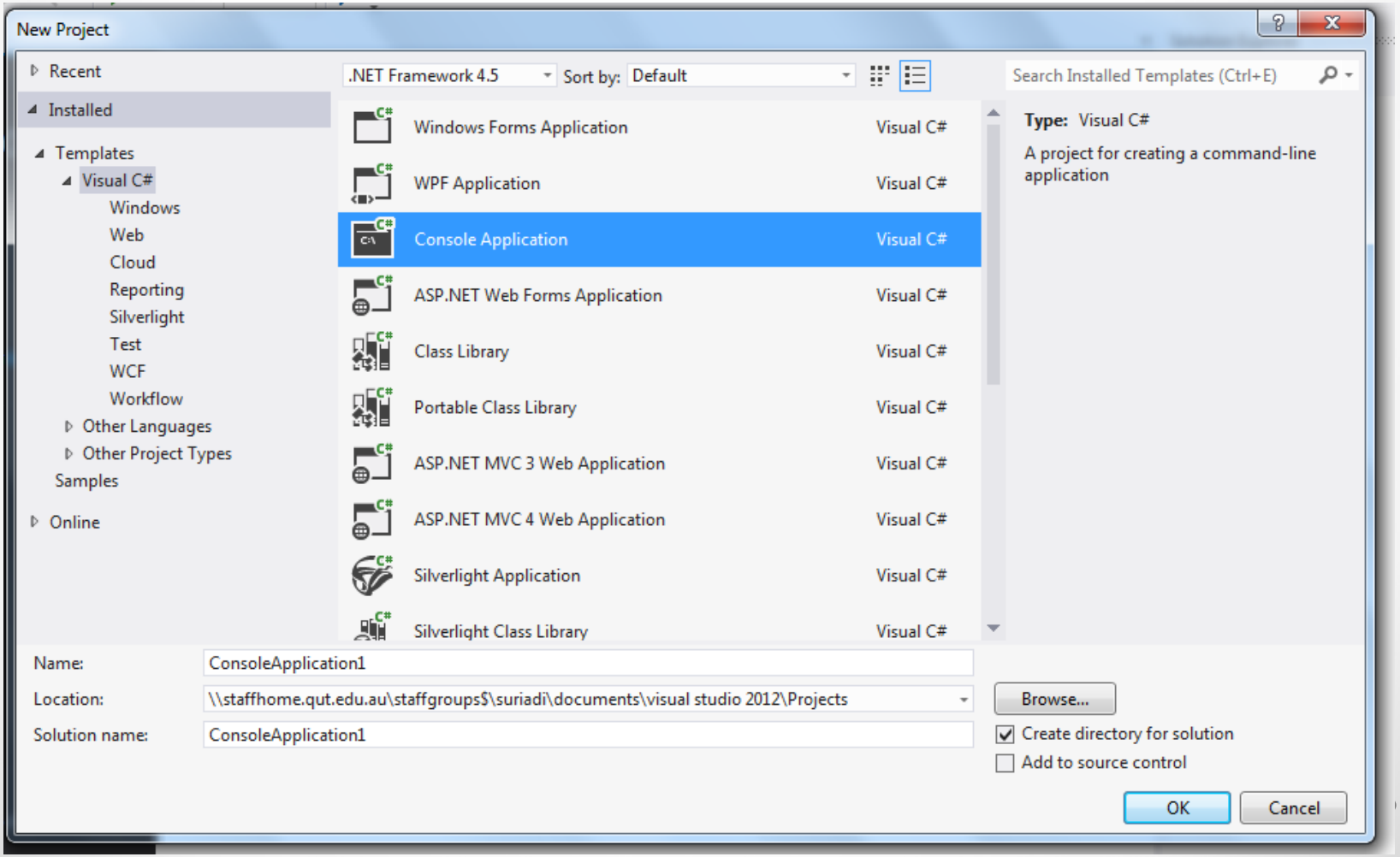
Creating a Command Line Program in Visual Studio

Then, create a new project:



Creating a Command Line Program in Visual Studio

Select Console Application (note the default environment)



Creating a Command Line Program in Visual Studio

Should get a file that looks like this:

Program.cs

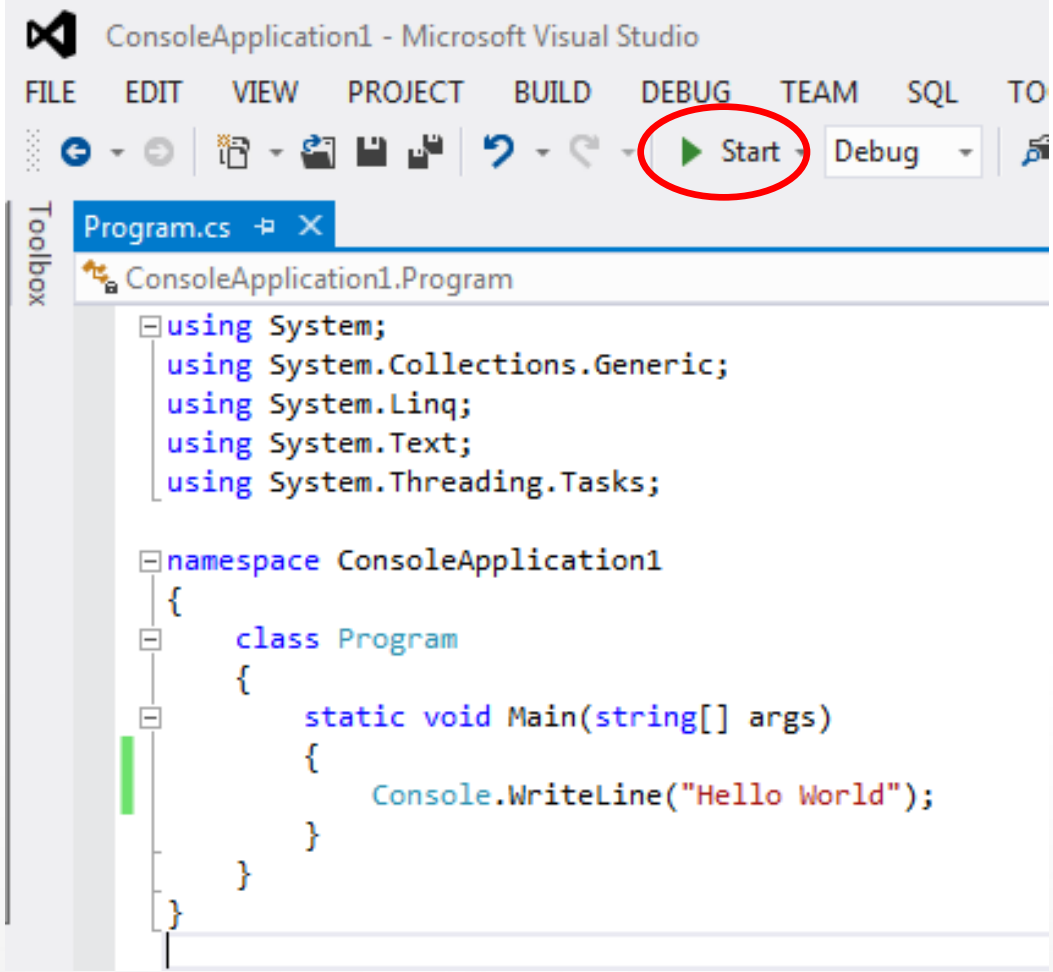
ConsoleApplication1.Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Creating a Command Line Program in Visual Studio

Hello World Program



Basic Programming

To create any computer-based applications, we must first understand the fundamentals of programming.

What is a program?

- A series of instructions
    - read data (e.g. users' keyboard input, files),
    - process the data
    - produce some outputs (e.g. data analysis results).
- Not necessarily sequential instructions
- Crudely, a program thus consists of *a set of instructions that manipulate and transform data in a particular order to produce certain outputs.*

Basic Programming

Data and operations – basic concepts

- Data Types
- Variables
- Declarations
- Constants
- Assignments
- Calculations
- Conversions

Data

- Data is integral to any applications. Why?
- Applications work with various types of data, e.g.
  - names, money, addresses, dates etc.
  - data quality should not be underestimated
- Different data types
  - are stored differently and
  - have different operations that can be performed.
- Data can be separated into two main types:
  - numeric
  - non-numeric

Numeric Data Types

- Numeric data types can be used for mathematical operations, e.g. add, subtract, multiply, divide etc.

- There are different types of numeric data types.

- Visual Basic has seven numeric data types

Numeric Data Types

Type	Storage	Range	
Byte	1 byte	0	to 255
Short(Int16)	2 bytes	0	to 65535
Integer(Int32)	4 bytes	-2,147,483,648	to 2,147,483,647
Long(Int64)	8 bytes	-9,223,372,036,854,775,808	to 9,223,372,036,854,775,807
Single	4 bytes	-3.402823e+38	to 3.402823e+38
Double	8 bytes	-1.79769313486232e+308	to 1.79769313486232e+308
Decimal	16 bytes	+/- 79,228,162,514,264,337,593,543,950,335 +/- 7.9228162514264337593543950335 (28 decimal points)	

- Decimal is more precise (base-10 operation), but less efficient. Used in accounting/finance software.

C# Numeric Data Types

VB type	C# equivalent
Byte	byte
Short	short
Integer	int (Int32)
Long	long (Int64)
Single	float
Double	double
Decimal	decimal

Non-numeric Data Types

- Non-numeric data types
  - represent data that cannot be manipulated mathematically,
  - but other forms of data manipulation is still possible.

- Non-numeric data types include
  - Text-based data (e.g. character and string),
  - Date,
  - Booleans, and
  - Objects (more details later in the semester).

VB Non-numeric Data Types

Type	Storage	Range
Char	2 bytes	0 to 65535
String	N/A	0 to 2 billion unicode characters
Date	8 bytes	January, 1, 100 to December, 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes (32-bit plat.) 8 bytes (64-bit plat.)	Any Object

C# Non-numeric Data Types

VB type	C# equivalent
Char	char
String	string
Date	DateTime
Boolean	bool
Object	object

Variables

- Variables is like a container that stores a particular data item.
  - Data can be put into a variable and read from a variable.

- To use variable the program must be able to
  - **interpret** the data inside the variable and
  - **identify** the variables – why?

Declaring Variables

- To declare a variable, you must supply both
  - a **type** and
  - a **name**.

- The type determines the meaning of the data inside the variable (for correct interpretation)

- The name allows you to identify the variable.

Declaring Variables

In VB

```
Dim password As String
Dim yourName As String
Dim age As Integer
Dim birthday As Date
```

In C#

```
string password;
string yourName;
int age;
DateTime birthday;
```

Variable Names

Some rules about variable names:

- Must be less than 255 characters
- No spacing allowed
- Must not begin with a number
- May not contain special characters ‘.’ ‘&’ etc

Assigning to Variables

After variables have been created, we can assign values to them:

```
password = “secret123”
yourName = “John Smith”
age = 30
birthday = #10/30/1981#  (in VB)
birthday = new DateTime(1981, 10, 30) (in C#)
```

Strings must be surrounded by “ ”

Date or Time literals by # # (in VB only).

Constants

Constants are similar to variables but their contents cannot be changed during the program. Useful for mathematical constants etc.

Constants must be assigned to when they are declared.

In

```
Const PI as Double = 3.14159 (VB)
const double PI = 3.14159 (C#)
```

Calculations

In VB, numeric data types can easily be manipulated using mathematical operators.

Operator	Function	Example
+	Addition	a + b (3+2 = 5)
-	Subtraction	a – b (3-2 = 1)
^	Exponential	a^b (3^2 = 9)
*	Multiplication	a*b (3*2 = 6)
/	Division	a/b (3/2 = 1.5)
Mod	Modulus (remainder)	a Mod b (3 Mod 2 = 1)
\	Integer Division	a/b (3\2 = 1)

Calculations

Strings can be manipulated in a number of ways.

Both + and & can be used to concatenate strings

```
Dim text1, text2, text3 as String
text1 = “Visual”
text2 = “Basic”
text3 = text1 + text2
text3 = text1 & text2
```

Both result in “VisualBasic”

Calculations

What's the difference? The + operator is generally used to add values together whereas the & is intended solely for concatenation.

The & operator will convert other data types to strings and concatenate them rather than perform an addition. **(Only in VB)**.

Calculations

For example:

```
Dim i1 as Integer = 2
Dim i2 as Integer = 3
Console.Write(i1 + i2)      Displays     "5"
Console.Write(i1 & i2)      Displays     "23"
```

Conversions

Applications must often convert data of one type to another.

For an integer to be displayed on the screen, it must first be converted to a string.

Likewise if the user types in a number, it is actually a string that must be converted to an integer.

Explicit Conversions

Visual Basic

Numeric types can be converted to a String using

```
Dim i1 As Integer
Dim s As String
s = i1.ToString()
```

String can be converted to numeric types using

```
Dim i1 As Integer
Dim s As String
i1 = Integer.Parse(s)
```

Implicit Conversions

or simply

```
Dim i1 As Integer
Dim s As String
s = i1
```

VB Only

```
Dim i1 As Integer
Dim s As String
i1 = s
```

Implicit Conversions

Converting from one numeric type to another.

```
Dim d As Double
Dim i1 As Integer = 3
Dim i2 As Integer = 4

d = i1/i2            gives 0.75
i1 = d               gives 1
i1 = 0.25            gives 0
i1 = 0.5             gives 0
```

VB Only

Conversions

In VB and C#, the following functions are also important for converting between the common data types.

VB.NET	C#
CInt()	Convert.ToInt32()
CLng()	Convert.ToInt64()
CSng()	Convert.ToFloat()
CDbl()	Convert.ToDouble()
CChr()	Convert.ToChar()
CStr()	Convert.ToString()

How about this (allowed in C# ?):

```
int a = 1;
int b = 3;
int c = a/b;
```

Summary

.NET Framework

Basic Programming:

- Types, Variables, Declarations, and Constants
- Assignments and Calculations
- Conversions
- Visual Studio 2012 IDE and “Hello World” program

Next Steps

- Deitel and Deitel (2014). Visual C# 2012 How to Program. 5th Edition. Pearson.
  - Chapter 1 and Chapter 2
- Deitel and Deitel (2014). Visual Basic 2012 How to Program. 6th Edition. Pearson.
  - Chapter 1 and 2
- The computer lab is already booked for Monday 3-5 pm, so you can use it this week if you want to start
- MSDNAA – get VS 2012 free, ask your program coordinator
- Start practical questions for Week 1
- Start assignment 1