

Introduction to .NET, C#, and Visual Studio

C# Programming

January 8

Part I

Administrivia

Administrivia

- When: Wednesdays 10–11am (and a few Mondays as needed)
- Where: Moore 100B
- This lab has Windows machines, but feel free to bring laptops

- Office Hours: to be announced

- Course webpage:
<http://www.seas.upenn.edu/~cse39905>

Course Structure

- No quizzes, no exams
- Roughly 6 projects
- Roughly 2 weeks per project
- The final project will be slightly longer and more open-ended
- Projects will be due at midnight on the night of the deadline
- All assignments should be submitted through the Blackboard Digital Dropbox
- Late policy: 15% off each day, up to 3 days late

Final Project

- Your chance to choose your own project
- Brainstorming and planning will begin after spring break
- Top projects will be entered into the Xtreme.NET Challenge – hopefully there will be 20 top projects :-)
- First prize: Xbox 360!
- Judges will include someone from Microsoft recruiting, maybe someone from the C# team
- More details to come at

<http://www.seas.upenn.edu/~cse39905/xtreme>

Part II

What is .NET?

The Microsoft .NET Framework

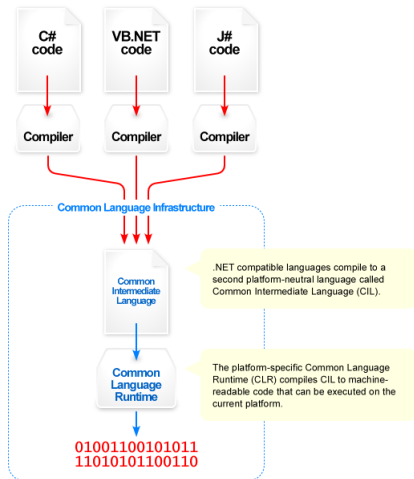
- .NET is a development platform that launched in 2000
- Goals include language independence, language integration, **web services**
- Technologies to promote rapid development of secure, connected applications
- .NET components include:
 - Languages (C#, VB, Visual C++, Visual J#, ...)
 - Common Language Runtime (CLR)
 - Framework Class Library (FCL)

Common Language Runtime

- A single runtime environment to execute programs written in any .NET language
- Includes a virtual machine
- Activates objects, manages memory, performs security checks, collects garbage

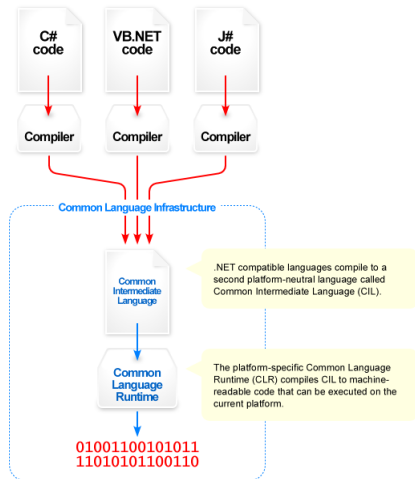
- To run on the CLR, a language must adhere to a Common Language Specification (CLS)
- A language must also build upon base types specified in the Common Type System (CTS)

Languages



- Language compilers translate source into Microsoft Intermediate Language (MSIL). The particular syntax of a language is unrestricted. (Language independence)

Languages



- Since each language targets the same IL with the same base type system, objects written in any .NET language can extend, implement, and interact with any other .NET objects.
(Language integration)

List of .NET Languages

(<http://www.dotnetpowered.com/languages.aspx>)

| | | | | |
|-------|---------|------------|------------|-----------|
| Ada | CAT | IL/MSIL | Nemerle | Scheme |
| APL | Cobol | Java | Pan | Smalltalk |
| AsmL | CULE | JavaScript | Perl | SML |
| Basic | E# | Lexico | Pascal | Spry |
| BETA | Eiffel | LISP | PHP | Synergy |
| BF | F# | LOGO | Processing | Tcl/Tk |
| Boo | Flash | Lua | Prolog | Zonnon |
| C | Forth | Mercury | Python | |
| C# | Fortran | Modula | Ruby | |
| C++ | G# | Mondrian | RPG | |
| Caml | Haskell | Oberon | Scala | |

Common Language Runtime

- The result of compilation – ie, MSIL – is not machine code
- At runtime, this intermediate code is Just In Time (JIT) compiled into machine instructions to execute
- Methods are *JITed* as they are needed, and cached for future use
- JITing allows optimizations based on the environment of the running program

Framework Class Library

- Bottom layer contains classes that support:
 - input and output
 - string manipulation
 - security management
 - network communication
 - thread management
 - text manipulation
 - reflection
 - collections functionality
 - etc.
- Next layer contains classes that support management of SQL and XML data
- Top layer supports application development
 - Windows Forms
 - Web Forms
 - Web Services

Benefits of .NET

From the ground up...

- The CLR abstracts away low-level details of the processor architecture and of Windows system calls
- It also manages memory automatically
- The FCL provides a rich collection of classes to build upon, including convenient ways of connecting to remote data stores
- The Windows Forms, Web Forms, and Web Service libraries facilitate rapid development of rich interface applications, both desktop and web
- Applications written in any .NET language can interoperate with each other

Part III

What is C#?

Language Features

- C# is *the* .NET language of choice for serious development
- (VB.NET is also very expressive, but gets a bad rap)
- A modern object-oriented language with its roots in C, C++, and Java
- C# 2.0 (and .NET 2.0) released in November 2005
- New features included generics and anonymous methods
- C# 3.0 (and .NET 3.0) planned for 2007
- New language features will include anonymous types, lambda expressions, and Language Integrated Query (LINQ)

Language Features

- Built-in value types (`byte`, `char`, `bool`, `int`, `float`, `double`, `decimal`, `long`, `struct`, `enum`, etc.)
- Classes, partial class definitions
- Single inheritance
- Interfaces
- Structs
- Delegates (“type-safe function pointers”)
- Properties (in place of getters and setters)
- Operator overloading
- Preprocessor directives
- Unsafe manipulation of pointers directly
- Commenting that generates documentation

Part IV

Hello World

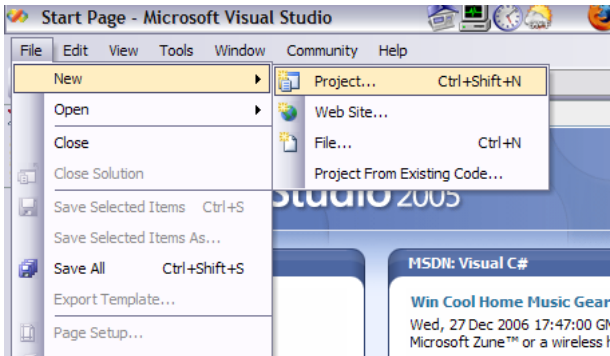
Visual Studio

Before we can write the simplest of programs, we need to become familiar with the Visual Studio development environment.

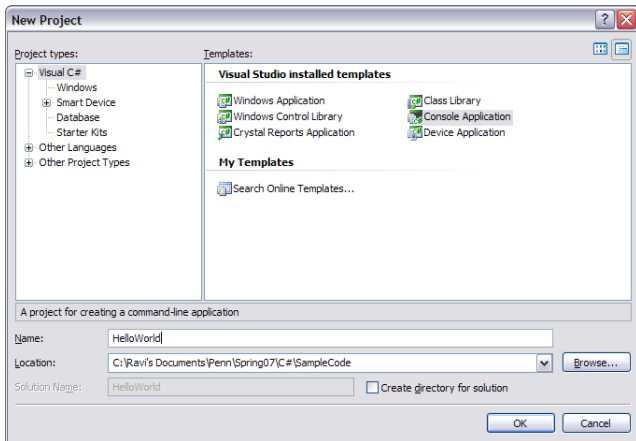
- Visual Studio facilitates rapid development with its Form Designer
- This allows the programmer to literally drag-and-drop GUI elements onto their window (or form)
- Double-clicking a form element creates a function (or event handler) that responds to that control's particular events
- Visual Studio's Form Designer has long been around for Visual Basic development
- It now offers this rich GUI development environment to all .NET languages

More on this next time...

Creating a Console App



Creating a Console App



If you choose to create a directory for solution, then all projects that are created within the solution will reside in the same folder

Project Organization

- A solution is a collection of projects
- Projects can be added and removed from a solution freely – you can organize a solution to contain related projects for simplicity
- To prevent naming conflicts, the `namespace` keyword restricts when classes are in scope
- You can think of a namespace as a Java package
- To refer to a class `Foo` in namespace `Moo`, you would write `Moo.Foo`
- If you write `using Moo;` outside of the namespace and class declarations, you can use `Foo` without qualification

Hello World

```
using System;

namespace HelloWorld
{
    class Hello
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello world!");
        }
    }
}
```

Compiling and Running

- To build the entire solution, F6
- To build the current project, Shift + F6
- To run the main project without debugging, Shift + F5
- To run the main project with debugging, F5

Compiling and Running

- To build the entire solution, F6
- To build the current project, Shift + F6
- To run the main project without debugging, Shift + F5
- To run the main project with debugging, F5

Part V

Some Language Features

Types

The following keywords, representing built-in C# types, are aliases for corresponding .NET types.

| | | | |
|----------------------|-----------------------------|---------------------|----------------------------|
| <code>bool</code> | <code>System.Boolean</code> | <code>uint</code> | <code>System.UInt32</code> |
| <code>byte</code> | <code>System.Byte</code> | <code>long</code> | <code>System.Int64</code> |
| <code>sbyte</code> | <code>System.SByte</code> | <code>ulong</code> | <code>System.UInt64</code> |
| <code>char</code> | <code>System.Char</code> | <code>object</code> | <code>System.Object</code> |
| <code>decimal</code> | <code>System.Decimal</code> | <code>short</code> | <code>System.Int16</code> |
| <code>double</code> | <code>System.Double</code> | <code>ushort</code> | <code>System.UInt16</code> |
| <code>float</code> | <code>System.Single</code> | <code>string</code> | <code>System.String</code> |
| <code>int</code> | <code>System.Int32</code> | | |

These keywords and .NET types (actually defined as structs) are **interchangeable**.

Access Modifiers

Each instance variable and method of a class has a visibility level.

- `public` – Accessible anywhere
- `private` – Accessible only within the class
- `protected` – Accessible within the class and within its descendants
- `internal` – Accessible within the class and within all classes in the same assembly
- `protected internal` – Accessible within the class, within its descendants, and within all classes in the same assembly

The default visibility (if none is explicitly written), is `internal`.

Properties

- In good style, instance variable are declared `private`
- In the event that these need to be read or modified from the outside, getter and/or setter methods need to be defined
- **Properties** in C# achieve this behavior without sacrificing the encapsulation of instance variables and without the the untidiness of getters and setters everywhere

Setting the Console's Title

The `Console` class, which we used in our first example, is a class defined somewhere in `System` namespace. We imagine that it has some `string` instance variable that holds the value of the window's title. Part of the class might look like:

```
public class Console
{
    ...
    private static string title;
    ...
    public static void getTitle() { return this.title; }
    public static void setTitle(string title)
        { this.title = title; }
    ...
}
```

Setting the Console's Title

Defining a property instead of a getter and setter, the class might look like:

```
public class Console
{
    ...
    private static string title;
    ...
    public static string Title
    {
        get { return title; }
        set { title = value; }
    }
    ...
}
```

Setting the Console's Title

- Now from the outside, the window's title can be changed by `Console.Title = "C# Saves the Day!";`
- Either the `get` or the `set` definition can be omitted as needed
- By convention, properties begin with a capital letter (and instance variables with a lowercase letter)

Comments

- `//` Single line comment
- `/*` Multi-line comments
can span multiple lines `*/`
- `///` `<summary>`
`///` These comments construct XML documentation
`///` for the code.
`///` All comments within these tags get extracted
`///` when the documentation generator is run.
`///` These also appear in the IntelliSense.
`///` `</summary>`