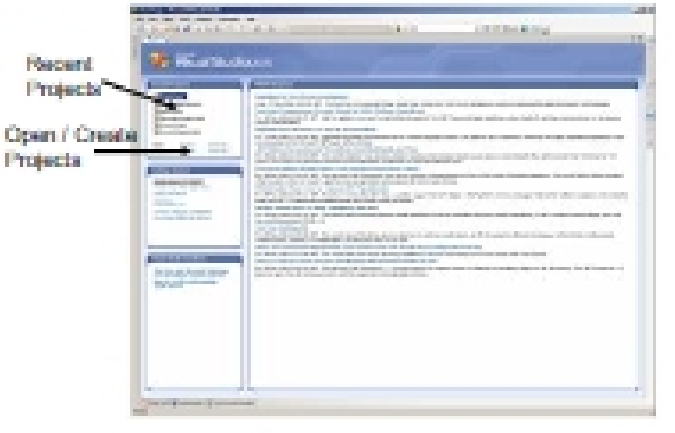
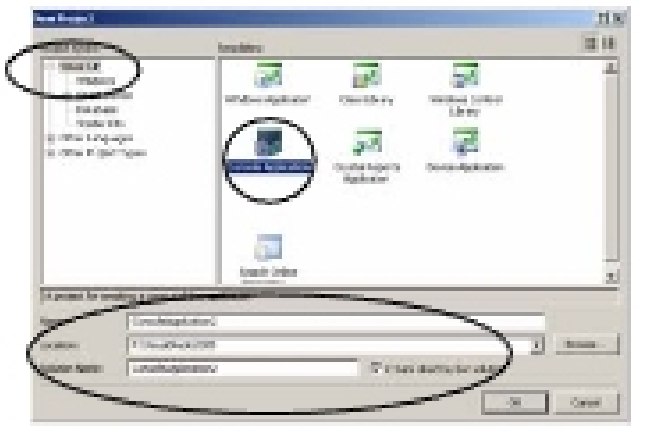
C# in 75 Minutes

Perry Kivolowitz

Launching / Start Page

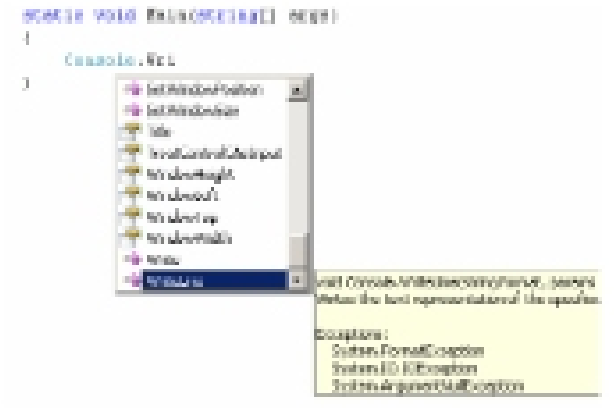


New console application



Intellisense

Intellisense



Intellisense

```
static void Main(string[] args)
{
    Console.WriteLine("Hello World
}
```

11 of 19 void Console.WriteLine(string value)
value: The value to write.

Hello World

```
using System;
using System.Collections.Generic;
using System.Text;

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

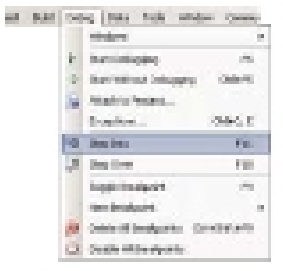
Hello World

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            Console.WriteLine("The time now is " + DateTime.Now.ToLongTimeString());
            if (DateTime.Now.DayOfWeek == DayOfWeek.Friday)
            {
                Console.WriteLine("TGIF !");
            }
            else
            {
                Console.WriteLine("It is a weekday :(");
            }
            Console.ReadLine();
        }
    }
}
```

Breakpoints / single stepping

- F9 toggles breakpoints
- F5 start debugging
- F10 step over
- F11 step into
- Shift F5 kills debugging



Watching / rerunning

Types

- object — *ALL types derive from object*
  - int — *int i = 0;*
  - string — *string s = "hello world";*
  - char — *char c = 'A';*
  - float — *float f = 3.0f;*
  - bool — *bool b = true;*
  - enums — eg: *DayOfWeek.Friday*
  - user defined *classes and structs*
  - many more

Object: base class of all types

- Even value types like int and float derive from Object:
  - int x = 1;
    x.ToString() → "1"
    1.ToString() → "1"
- Object implements other methods like:
  - object.Equals(other)
  - object.GetType()
  - Not used in this course

Strong typing

- Pro
  - Catch / prevent errors at compile time
- Con
  - Verbosity

Strong typing

```
class Class1
{
    public void Foo()
    {
        int i = 0;
        float f = i;                        // implicit - works
        float g = ((float) i) / 1.0f;       // explicit - works
        double h = i;                       // implicit - works
        decimal k = 1m;                     // explicit - works
        double l = 1.ToDouble();            // all types have ToDouble()
        decimal m = System.Convert.ToDecimal(1);  // generalized type conversion
        char a = 'a';                       // works
        char b = 'b';                       // works
        float c = 1.0f;                     // works
        float d = 1.0f;                     // works
    }
```

Type conversion

- Implicit
  - Obvious relationship exists
  - No loss of information
- Explicit
  - Like a cast in C or C++
- Type conversion
  - System.Convert.To_____()
  - Use this extensively for ADO.NET work.

Strings

- string s = "some string";
- string s += " and some other string";
- s.Length – read only attribute
- s.Trim() – returns string without leading / trailing white space
- Many other members to the string class
  - Split(), SubString(), etc.

Strings

- Can be indexed: char c = s[2];
- Usual escape sequences
  - \" \n \\ etc.
- Precede with @ to make a literal string
  - @"C:\temp\foo" is the same as
  - "C:\\temp\\foo"
- Well defined logical operators like =, >, etc.
- See also StringBuilder class

Equivalent of (s)printf

- string System.Format()
- Uses positional notation:
  - System.Format("Hi {0} {1},", fName, lName);
- Formatting for specific types available
  - Left for the reader