CS360

Windows Programming

.NET Framework

- Components of the .NET framework
  - Framework Class Library (FCL)
  - Common Language Runtime (CLR)
  - Common Intermediate Language (CIL)

Managed Modules

- An executable designed to be run by the CLR is a *managed module*
- One of the elements of the managed module is the *metadata* that describes the contents of the module

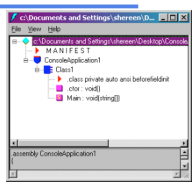
ILDASM

- The metadata includes
  - List of all types (i.e. classes)
  - List of methods (functions)
  - List of fields
  - List of properties
- Visual Studio provides a tool that enables us to view the metadata
  - ILDASM

ILDASM

```
using System;

namespace ConsoleApplication1
{
   class Class1
   {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine( "Hello World!" );
        }
   }
}
```



Common Intermediate Language

- CIL is a pseudo-assembly language
- Includes around 100 instructions
- Uses a stack-based execution model
- *Loading* copies from memory to the stack
- *Storing* copies from the stack to memory

Example

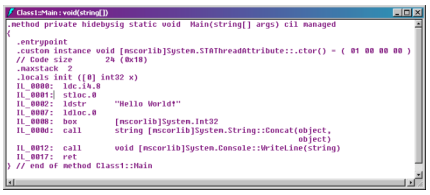
```
int a = 3;
int b = 7;
int c = a + b;

ldc.i4.3  // Load a 32-bit (i4) 3 onto the stack
stloc.0   // Store it in local variable 0 (a)
ldc.i4.7  // Load a 32-bit (i4) 7 onto the stack
stloc.1   // Store it in local variable 1 (b)
ldloc.0   // Load local variable 0 onto the stack
ldloc.1   // Load local variable 1 onto the stack
add       // Add the two and leave the sum on the stack
stloc.2   // Store the sum in local variable 2 (c)
```


Example

```
using System;

namespace ConsoleApplication1
{
 class Class1
 {
  [STAThread]
  static void Main(string[] args)
  {
   int x = 8;
   Console.WriteLine( "Hello World!" + x );
  }
 }
}
```



Intellectual Property Issues

- Is it a problem that we can disassemble the programs from their executables?
- Web services are never exposed to users
- Code distributed to end users can be scrambled

.NET Framework Class Library

- C Programmers
  - Use Windows API
- C++ Programmers
  - Use MFC
- Visual Basic Programmers
  - Visual Basic API
- .NET Programmers
  - .NET Framework Class Library

OOP

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Why is C# Important?

- Designed specifically for writing .NET code
- Includes modern language features
- An improvement over existing languages

C#: Design Goals

- A simple, modern, general-purpose, object-oriented language
- Support for software engineering principles such as
  - type checking
  - array bounds checking
  - detection of use of uninitialized variables
  - garbage collection
  - multi-threading
  - exception handling
  - explicit pointers only in "unsafe" code

C#: Design Goals (continued)

- Suitable for applications deployed in a distributed environment
- Familiar syntax for C/C++ programmers
- Economical in use of memory and processing requirements, but not competing directly with C or assembly language on size or performance

C#: Language Constructs

- Operators and Expressions: same as C++
- Arrays: 1-D, multi-D, jagged
- Flow Control:
  - if/else
  - switch
  - goto
  - continue
  - break
  - return
- Iteration
  - for
  - while
  - do/while

C#: Classes

- Single inheritance
- Can implement multiple interfaces
- Members
  - Fields, methods (including constructors),
- properties, indexers, events
  - Access control: public, protected, internal, private
  - Static and instance members
  - Abstract methods (for polymorphism)
- Nested types

Let's Do Something Practical

```
using System;
class Hello
{
 static void Main()
 {
  Console.WriteLine("Hello, world!");
 }
}
```

Type: types include classes, structs, enumerations, etc

Method (Another name for function)

Another Example

- Write the definition for a class Rational that stores the numerator and denominator
- The class should contain the necessary constructors
- The class should also have the following functions
  - Add
  - Operator+
  - ToString
  - ReduceToLowestTerms
  - Gcd

Rational

```
class Rational
{
 private int num, den;
 public Rational(int numerator, int denominator)
 {
  this.num = numerator;
  this.den = denominator;
  ReduceToLowestTerms();
 }
```


Rational

```
public Rational(int num) : this(num, 1)
{
}

public Rational() : this(0, 1)
{
}
```


Rational

```
public Rational Add(Rational r)
{
 Rational sum = new Rational(r.num * this.den
                           + r.den * this.num,
                             r.den * this.den);
 return sum;
}
```


Rational

```
public static Rational operator +(Rational
  r1,
                                  Rational r2)
{
 return r1.Add(r2);
}
```


Rational

```
public override string ToString()
{
 return "[" + num + " / " + den + "]";
}
private void ReduceToLowestTerms()
{
 int g = Gcd(num, den);
 num /= g;
 den /= g;
}
private int Gcd(int x, int y)
{
 if (y == 0)
  return x;
 else
  return Gcd(y, x % y);
}
```


Rational

- All the previous create the Rational class
- How can we test the Rational class?

Summary

- We have completed chapter 1
  - P. 3 - 26