
C# Programming

CMPS203 Programming Languages
Project Presentation

Ismail Ari
ari@cse.ucsc.edu
University of California Santa Cruz



1

Roadmap

- Definitions (C#, .NET, CLR, IL)
- C# Syntax (*HelloWorld.cs*)

- Advanced C# concepts
- Summary: C# compared to C++ and Java

- Web Services
- Proposed Research Questions

2

C#

- C# is a new language designed specifically for Microsoft's .NET framework
- Combines best features of VB, Java and C++
- C# is intrinsically OO and web-enabled
- C# is *component oriented* & type-safe
- C# standard has been submitted to ECMA
 - European Computer Manufacturers Association
 - Java still not submitted to an independent standards body

3

What .NET does ...

- .NET framework is a computing platform
 - provides the infrastructure necessary to build and operate Internet based services and other applications.
- .NET framework promises to revolutionize programming (*on Windows platforms*)
 - Makes Internet development (Distributed Computing) easier
 - People develop software and components using different languages
 - Interoperability between components required
- .NET framework is a package of
 - Protocols (XML, SOAP)
 - Programming Languages (C#,VB, C++, Java,Perl etc.)
 - Tools

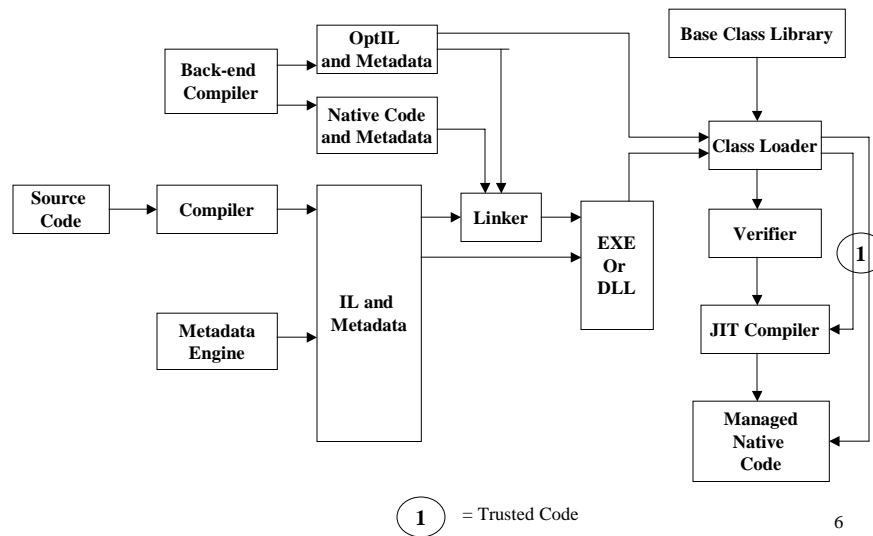
4

How .NET does it ..

- B.C. (Before C#, actually .NET)
 - Different languages required different runtime
 - VB runtime required to develop ActiveX controls
 - Web surfer had to have the runtime installed
 - There were versioning problems
 - DISTRIBUTED NIGHTMARE! (*for windows based systems !!*)
- .NET framework introduced a single, common runtime:
 - *Common Language Runtime (CLR)*
 - similar to JVM (automatic garbage collection, type-safety checking)
- All .NET programs are compiled to an *intermediate language (IL)* and IL is later compiled to native code.

5

Common Language Runtime



6

CLR Operation

- Compilation
 - Source *compiled* to IL by compiler
 - Metadata info created by metadata engine
 - Optionally linked with other code compiled by different compilers
 - Result is an EXE or DLL containing IL code
- Execution
 - IL code and any functionality from base class library is brought together by class loader
 - Combined code tested for type-safety by Verifier
 - JIT *compiler* processes IL and creates managed native code
 - Native code passed on to the .NET runtime manager (execution engine)
- .NET programs are effectively compiled *twice* !

7

CLR (Common Language Runtime)

- CLR services:
 - conversion of IL to native code,
 - type checking,
- Other CLR services
 - memory management and garbage collection,
 - exception handling, security,
 - profiling, debugging.

8

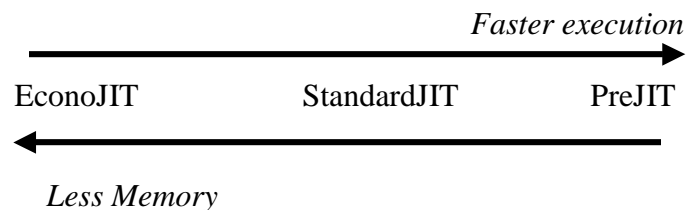
IL (Intermediate Language)

- IL is not interpreted (??)
 - Java byte codes are interpreted
 - Interpretation has negative effect on the performance
 - IL format is specified in Common Lang. Specification (CLS)
 - COM specification vs. CLS
- One can think of IL as an assembly language with extra commands for objects
 - There is close correlation between IL commands and their machine code equivalents
 - ILDASM.EXE: IL Disassembler could be used to extract IL

9

Just In Time (JIT) Compilers

- .NET will ship 3 JIT Compilers (*actually 2!*)
- They balance between
 - Compilation speed
 - Code execution speed
 - Memory Usage



10

.NET Base Classes

- “System” is the root of .NET base class namespace
- Basic Data Types defined in System
 - bool, char, int, double, .. ,
 - .., object, struct, uint ..
 - object is the type from which all other classes are derived.
- Some other namespaces within System
 - System.IO, System.Exception
 - System.Math, System.Random
 - System.Collections, System.Threading

11

C# Syntax (HelloWorld.cs)

```
// Allow easy reference to classes in the System namespace
using System;

// This "class" exists only to house the
// application's entry-point function
class MainApp {
    // Static method called "Main" is
    // application's entry point function
    public static void Main() {
        // Write text to the console
        Console.WriteLine("Hello World using C#!");
    }
}
```

12

ADVANCED C# CONCEPTS

13

Call by value, call by reference

- As we already know, with:
 - *Call by value*: local changes, made to the value passed to a method, do not affect original copy
 - *Call by reference*: modifications will persist.
- What if you want to call by value and be able to modify it locally? (use keyword: **ref**)

<pre>void refMethod(ref int n){ n += 3; } ... int p =3; // initialized refMethod(ref p); //p changed</pre>	<pre>void outMethod(out int n){ n = 3; } ... int p; // uninitialized outMethod(out p); // p set</pre>
--	---

14

Overloading

- Both method overloading and operator overloading are same as in C++ (and Java? Check syntax)

15

Exception Handling (1)

- Very much the same way as C++ and Java
- Unlike C++ you can't **throw** any type in C#
 - Has to be **System.Exception** type
- Built in exception handlers
 - e.g. Divide-by-zero-exception
 - `Console.WriteLine("Result is {0}", 15.0/0.0);`
 - `Result is Infinity`
- Order of catch blocks is important
 - Derived exception handlers must come before their base exception classes

16

Exception Handling (2)

- General `catch{ }` is C#'s equivalent of `catch(...)` in C++
- `CoreException` is C#'s equivalent of `Error` in Java
- Keywords: `checked/unchecked`
 - `checked(a*b), checked{ }`
 - Can be applied to arithmetic operations, statements, casts

17

Console.IO

- `Console.In.Read();`
- `Console.In.ReadLine();`
- `Console.Out.Write();`
- `Console.Out.WriteLine();`
 - `Console.WriteLine()` is shortcut for this
 - Markers: `{0}`, `{1:E4}` used for argument substitution
 - Formatting specifications (C,D,E,F,G,N,X)
- `Console.Error`

18

C# Namespaces (1)

- *Namespaces* provide a way to group classes
- In Java, the *package* names are forced into a hierarchical directory structure
- Namespaces in C# maintain a separation between the logical names and the physical packaging.
 - Namespaces may be called anything,
 - They just provide a level of scoping for classes.
- C# namespaces more flexible
 - components in logical packages easily distributed

19

C# Namespaces (2)

- Fully qualified names
 - `Finance.Bank.Account ba = new Finance.Bank.Account();`
- Keyword **“using”**
 - `Using Finance.Bank;`
 - `Account ba = new Account();`
- You can't import a single class in C#
 - Whole package gets included (!@#\$)
- *Assembly*: Basic element of packing in C#
 - Contains IL code and metadata describing what is inside the assembly
 - Physically housed in EXE/DLL or split across more than one physical file.

20

OO features of the C# language

- **struct** and **enum** are available for use in C#
- C# classes
 - C# doesn't separate class definition and implementation (no header files; like Java)
 - No semicolon (;) after class definition
 - Access modifiers: **public**, **private**, **protected(?)**, **readonly**, **internal**
 - **const** and **readonly** members
 - C# does not define default constructors
 - Destructors are very similar to Java's **finalize()** method

21

C# Inheritance and Polymorphism

- C# allows single inheritance
 - `public class CheckAccount : Account { .. }`
 - Interfaces used for multiple inheritance
 - Keyword **"base"** is used to access base class from the derived class
- Polymorphism (Dynamic Types)
 - Base class will use **"virtual"** keyword
 - Derived class has to use **"override"** keyword
 - **abstract** classes are cannot be instantiated
 - Keyword **"sealed"** avoids being used as a base class

22

Object Class

- **System.Object**
- All classes in C# are derived from Object
 - Including built in types like int, long
- Object has four methods
 - ToString(): returns name of the class
 - e.g. **Person** class overrides it to return “name” of the person
 - Equals(): compares the content of objects
 - This is different from “o1==o2” reference comparison
 - GetHashCode(): helps searching for objects easy
 - GetType()

23

Boxing & Unboxing

- C# confusingly has another class called ‘object’
 - object-with-a-small-O
- *Boxing*: is to use ‘object’ value type as a box for other value types.
- *Unboxing*
 - Recasting back
- C# can automatically do boxing
 - This works ...

```
int n=4;  
// create a box to hold n  
object o= n;  
// Unbox it  
int m = (int)obj;
```

```
string s = 10.ToString;
```

24

C# Interfaces

- **interface** contains the definition of methods and no implementation code
 - Works very much the same way as Java interfaces
 - Similar to C++ abstract base-classes
- C# doesn't allow multiple inheritance, but you can implement multiple interfaces

```
Interface ICame{
    void Come();
}
Interface ISaw{
    void Look();
}
public class Test:ICame, ISaw
{
    public void Come(){//implement it}
    public void Look(){//implement it}
}
```

25

Properties

- C# accessors(get/set methods) are built into the language, in the form of *properties*
- It looks like getting direct access to the data
 - ..or as if you overloaded '.' operator
- Get clause doesn't have to return value type
 - *Properties* can represent dynamic data

26

Properties (2)

```
public class Test {
    private int val;
    public int Val{
        get{ return val; }
        set{ val = value; }
    }
}
...
Test t = new Test();
t.Val = 10; // set
int n = t.Val; // get
```

27

Indexers

- Useful when the class is a container of objects
- Similar to C++ overloaded [] operator

```
public class IndexTest{
    double values[];
    public IndexTest(){..}

    public double this[int index]{
        get{ return values[index];}
        set{ values[index] = values;}
    }
}
```

28

Delegates (1)

- Function pointers (type-safe!)
 - A reference to a method rather than a data member
- A delegate provides a ‘template’ for a single method
 - Defines a function without implementing it

```
public class ArithTest{
    public delegate int ArithOp(int a, int b);
    public int DoOp(ArithOp ar){
        int n=4, m =3; // DoOp -> handle()
        return ar(n,m); // ar() - event_handler()
    }
}
```

29

Delegates (2)

```
public class DoTest{
    public static AddOp(int a, int b){ return a + b;}
    public static SubOp(int a, int b){ return a - b;}

    public static Main(String[] args){
        ArithTest.ArithOp add = new ArithTest.ArithOp(AddOp);
        ArithTest at = new ArithTest();
        int result = at.DoOp(add);
    }
}

Why not add them using AddOp?? DoOp could have done more.
```

30

Events (1)

- Some classes can *publish* set of events
- Others *subscribe* to be notified at run time
 - Only those who subscribed are notified
 - Avoids polling for changes/events.
- A class publishes *callback functions* as delegates
 - and the listening object implements them.
- Observer/Observable Design Pattern

31

Events (2)

```
// Main method of a test class
public static int Main(string args[]){
    StockWatcher sw = new StockWatcher();
    StockObserver obs = new StockObserver("One", sw);
    rw.Notify("base", 5.7);
    return 0;
}

class StockWatcher{
    public delegate void PriceChange(object sender, StockInfo
    info);
    public event StockChange OnChange;
    public void Notify(string name, double newval){
        StockInfo s = new StockInfo(name, newval);
        OnChange(this, s);
    }
}
```

32

Events (3)

```
class StockObserver{
    StockWatcher ws;
    // .. constructor to initialize ws & set up the
    delegate
    public void StockHasChanged(object sender, Stockinfo
    info)
    { /* Prints price change message & new value */ }
}

class StockInfo : EventArgs {
    public readonly string name;
    public readonly double price;
    // constructor to initialize the data
}
```

33

Attributes (1)

- Declarative information about a classes, methods or data members
 - conditional, obsolete, guid, in,out
 - serializable , unserializable

```
#define DEBUGGING
public class Test {
    [conditional("DEBUGGING")]
    public void TraceOutput(){
        Console.WriteLine("Some Trace Output...");
    }
}
```

34

Attributes (2)

```
public static int Main(string args){
    Test t = new Test();

    // If DEBUGGING is not defined, the
    // following line will be ignored,
    // and no exception will be thrown.
    t.TraceOutput();

    // conditional methods can only have void
    // return type to block selective discards.
    // Assume bar() was conditional..
    int result = t.foo() + t.bar();
}
```

35

Code Management

- Memory Leak
 - Programs failing to release allocated memory
 - eventually eat up all RAM and crash the system
- CLR's Garbage Collector
 - Periodically checks the memory heap for unreferenced objects
 - Frees memory used by unreferenced objects

36

Safe/Unsafe vs. Managed/Unmanaged

- Unsafe and unmanaged code confusion
- Unmanaged
 - Native code that does not execute in runtime context
 - C# code is managed
 - VB6 and C++ (without managed extensions) are unmanaged
- Unsafe blocks
 - Sidestep C#'s type-checking mechanism
 - e.g. C# allows pointer usage for direct memory access

37

C# Documentation

- `javadoc` processes special tags in Java source code and produces HTML pages as its output.
- The idea is to automatically generate the documentation from the source code.
- C# Documentation differs from javadoc in 2 ways
 - It is processed by the compiler, not by a separate tool
 - It generates XML rather than HTML
 - just apply an XSL stylesheet and get HTML if you want
- Syntax
 - `csc /doc:myclass.xml myclass.cs`
 - Tags: `///`, `<summary>`, `<c>`, `<example>`, `<see>`..

38

Microsoft Componentization History

- DLL (Dynamic Link Libraries)
- COM (Component Object Model)
 - DLLs from different prog. Languages can be shared
 - *COM+* = COM + DCOM (*RPC*)
- .NET Components
 - Web is transitioning away from a network for serving HTML documents towards globally-distributed applications consisting of millions of objects living on different platforms and negotiating with each other using SOAP (Simple Object Access Protocol).

39

Web Services

- Today usually, HTML pages are the results of our requests to servers
- Ultimate Distributed Application
 - Have components, whose methods are callable over the Internet.
- A fully customizable web page
 - Even more flexible than a dynamically generated page.
 - Clients can access data (news, quotes etc.) using their own programs, not through dynamic/static or customized server applications..

40

Client-Server using C#

41

Summary: C# and C++

- Same

- Different

42

Summary: C# and Java

- Same

- Different

43

Skipped Details

- Windows Applications (WinCV, Visual Studio)
- C# XML Documentation
- ADO.NET, ASP.NET, VB.NET
- COM, COM+ Interoperability
- Assemblies and Manifests

44

Relations to Design Patterns

- Events: Observer / Observable Pattern
- Delegates : Facade, adapter, proxy, decorator
- Attributes : ??
- Indexers: Iterator pattern

- Other Design Patterns:
 - Factory, state/strategy, singleton, command, CoR..

45

C# Research Possibilities

- Design Patterns in C#
 - Does C# make it easier to implement Design Patterns?

- Compare
 - CORBA, CORBA IDL, IIOP
 - Java RMI, Java IDL
 - C# .NET Components , IL & CLS
 - Microsoft COM, DCOM.

46

Some References

- C# Programming With the Public Beta
 - Harvey, Robinson, Templeman, Watson
- Overview of the .NET Platform and C#
 - Sinderson E., CMPS203 Project Report, UCSC
- <http://www.csharp-tutorial.com>
- <http://www.csharp-station.com>
- Deep Inside C# (*John Osborn*)
 - Interview with MS chief architect Anders Hejlsberg
- MS .NET vs J2EE (*O'Reilly News Article*)
 - Jim Farley (Author of *J2EE in a Nutshell* by O'Reilly)