C# for Java Programmers

by
Venkat Subramaniam
venkats@agiledeveloper.com

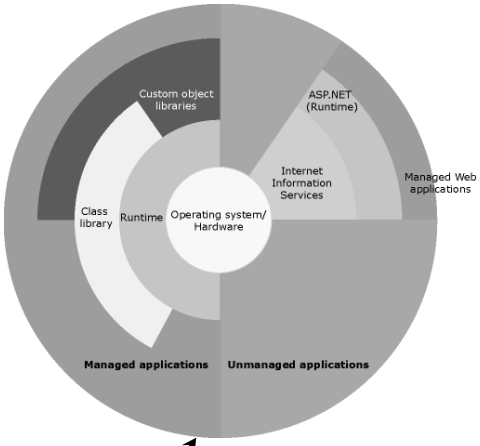
May 2003

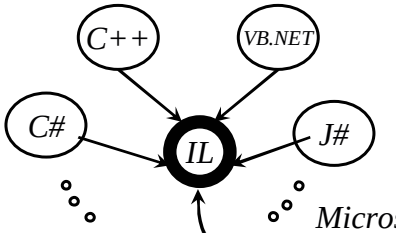
Presentation and examples can be downloaded from
http://www.agiledeveloper.com/download.aspx

Abstract

- Java and C#, as we know have a lot of similarities. Yet there are some differences as well. This presentation is intended for developers who are very familiar with Java and want to gain enough understanding to start programming in C#. A short overview of .NET architecture is first presented. A quick comparison of some of the subtle differences in the syntax and constructs are discussed. Finally, the features that are different in C# are presented. The topics covered include C# syntax, Delegates, Properties, Attributes, Garbage Collection, Code based security, pointer access, operator overloading, WinForm and Server side programming. Several code examples will be presented to develop code using Visual Studio .NET and the freely downloadable .NET framework as well.
- Dr. Venkat Subramaniam is an agile developer who teaches and mentors software developers. He has significant experience in architecture, design and development of distributed object systems. Venkat has trained more than 2500 software professionals around the world. He is also an adjunct professor at University of Houston and teaches the Professional Software Developer Series at Rice University's Technology Education Center.
- Each page with a  has an attached example

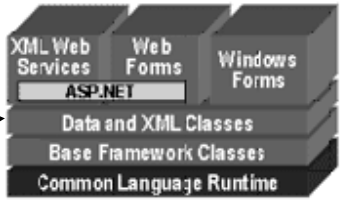
.NET Framework





Microsoft Intermediate Language (MSIL or IL)

Source: http://msdn.microsoft.com



Technology Comparison

Java	.NET
JVM	CLR
Byte Code	MSIL
Swing	WinForms
Servlet, JSP, Struts	ASP.NET, WebForms
Micro Edition	Compact Edition
JDBC	ADO.NET
Web Services(JAXRPC)	WebServices
J2EE*	Enterprise Services*
RMI*	.NET Remoting*
JMS*	MSMQ*
JNI	PInvoke/COM Interop
Java Predominates	Multi language support
Multi Platform	Windows Predominates
* Not quite equivalent - discussed later	

Java vs. C#

- Is this Java or C#?

```
public class HelloWorld
{
    public static void Main(String[] args)
    {

    }
}
```

main in Java

Syntax Similar, Yet Different

- Java
  - C++, Smalltalk, Objective-C root
  - Removes number of gotcha in C++
  - Restrictive when it comes to
    - pointers
    - methods are virtual by default
      - Positive: No way to hide a method and shoot in the foot like C++
- C#
  - Tries to be a union of C++ and Java
  - at times irritating in this regard
    - Discussed later
    - Use your judgment in using some of these features

Syntax Similarity

Java

```
public class HelloWorld
{
      public static void main(String[] args)
      {
            System.out.println("Hello World!");
      }
}
```

C#

```
public class HelloWorld
{
      public static void Main(String[] args)
      {
            Console.WriteLine("Hello World!");
      }
}
```

Optional

Syntax Differences

- Access Controls
  - Java
    - private, public, protected, (package friendly)
  - C#
    - private, public, internal, protected, protected internal
- Inheritance
  - Java
    - extends, implements
  - C#
    - : (for both one base class and interfaces)

Syntax Differences Overriding

- Overriding
  - Java
    - final vs. non-final methods
    - You may accidentally override methods
      - Not really an issue most of the time!
  - C#
    - methods not marked as virtual can not be overridden
    - Overriding method needs to be marked with override
      - if you do not, it only gives you warning & assumes you are hiding
        - You got to be kidding!
        - Use the option to treat warnings as errors PLEASE
    - You can mark a method as new with intent to hide
      - I feel sick in my stomach

Java vs. C# Some Differences

Java	C#
package	namespace
import	using
Pass by value only	By value, ref and out
instanceof	is
extends, implements	:
super(…)	: base(…)
super.method()	base.method()
final (class)	sealed
final (field)	readonly
Class	Type
Serialization (binary)	Serialization (XML too)

Some More Differences

- Only in Java
  - Checked Exceptions and throws clause
  - Anonymous Inner classes (coming in future version of C#)
- Only in .NET
  - Attributes
  - Properties
  - Indexer
  - IDisposable
  - foreach
  - Delegate
  - Code based security
  - Pointer access - allows through “unsafe” construct
  - operator overloading

Attributes

- Defines characteristics on various subjects
  - assemblies, classes, methods, properties, etc.
- Similar to the attributes in Microsoft IDL
- Appears within [] before the subject

```
[System.Serializable]
public class Sample {
      [System.NonSerialized] private int aField1;
      private int aField2;

      [System.Obsolete("Use increment instead", true)]
      public void inc() {…}

      public void increment() { aField2++; }
```

Properties

- An attribute or field represents some characteristics of the object
- Making it public is undesirable
  - uncontrolled access
- You may make them private and provide access methods
- If intent is to access field, why not expose it as a property
  - intent is clear
  - tools can help us identify properties
  - change is very much controlled – still encapsulated!
- Compiler translates property p into get_p and set_p methods! and provides an illusion to the user

Writing Properties

```
public class Car {
      private int yearOfMake;
      private string bodyColor;
      public Car() {…}
      public int year
      {
            get { return yearOfMake; }
      }
      public virtual string color
      {
            get { return bodyColor; }
            set {
                  if (value.CompareTo("Orange") == 0)
                     throw new ApplicationException(…);
                  bodyColor = value;
            }
      }
}
```

Read-only property

Read/Write property

Well Encapsulated

Indexer

```
public class Vector {
      private int[] values;
      private int size;
      public Vector(int rSize) {…}
      public virtual int this[int index]
      {
            get
            {
                  if (index >= size)
                  throw new IndexOutOfRangeException(…);
                        return values[index];
            }
            set
            {…
                   values[index] = value;
            }
      }
}
```

Gives an illusion of being indexed

Translates into get_item and set_item methods

Problem with Finalize in Java

- Java has automatic Garbage collection
  - No need to worry about memory cleanup
  - Still resource cleanup is a concern
- Finalize called when Garbage Collector returns object to heap
- Garbage Collector may be lazy - Finalize will be called sometime in the future
- If program exits fast - Finalized may never be called

Do not depend on the Finalize() method

IDisposable

- .NET has a better handle on this
- Dispose the object by calling Dispose

```
public class Garbage : IDisposable {
      private bool disposed = false;
      public void Dispose() {
            if (disposed == true)
            throw new ObjectDisposedException(…);
            disposed = true;
            // What ever cleanup
            Console.WriteLine("Dispose called");
            GC.SuppressFinalize(this);
      }
      ~Garbage() { Dispose();
            Console.WriteLine("Finalize called");
      }
}
```

```
using (Garbage obj = new Garbage())
{
      // code to use obj
}//obj.Dispose() called automatically here!
```

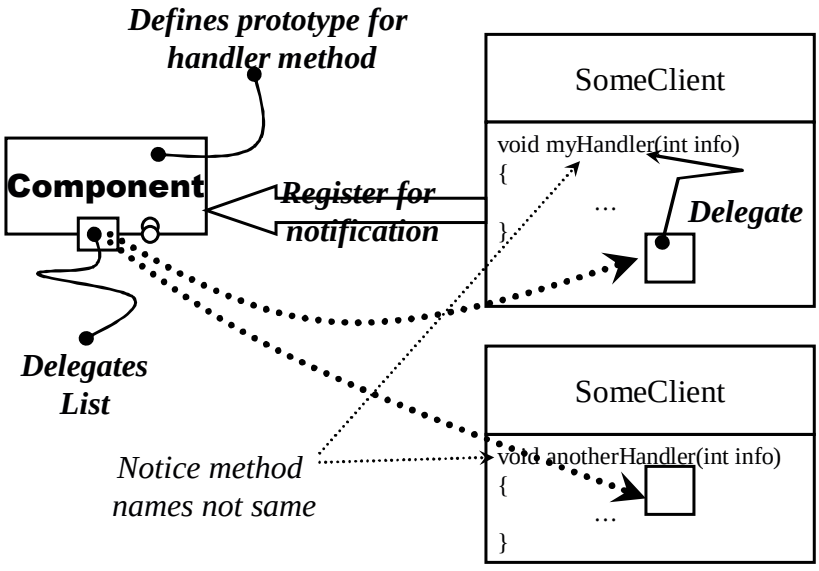
foreach

- Any object that implements the IEnumerable can be traversed using a foreach

```
foreach(DataRow row in table.Rows)
{
     int someField = row[“some_field”];
     // row here represents each row in
     // the collection in the respective
     // iteration through the loop
}
```

Quiz Time!

Event Handling with Delegate



Delegate

- Event handlers traditionally were global or static functions
- In .NET, delegates allow a lot more
- Delegates are data structures (objects) that hold
  - either a pointer to global or static function
  - or a pointer to an object's method and the object itself
  - are object-oriented, type safe and secure
- All Delegates derive from the Delegate class

Writing A Delegate

- Delegate classes are written with special syntax
- Compiler does special processing
  - Writes a constructor and Invoke method when compiled
- You can maintain a list of delegates by simply adding and subtracting
  - myDelegate += anotherDelegate;
    // Adds the delegate
  - myDelegate -= anotherDelegate;
    // Removes the delegate

Writing A Delegate…

```
public delegate void StockQuoteDelegate(double amount);
public class StockQuote {
      public StockQuoteDelegate highDelegate = null;
      public StockQuoteDelegate lowDelegate = null;

      private void newHighReached(double amount)
            {// one way to invoke the handlers
                  Object[] args = new Object[1];
                  args[0] = amount;
                  highDelegate.DynamicInvoke(args);
            }
      private void newLowReached(double amount) {
            //another way of achieving the same result
                  lowDelegate(amount);
            }
}
```

Using A Delegate

```
public class MyClass {
public static void highReport(double amt) { … }
public static void lowReport (double amt) { … }
public void beepReport(double amt) { … } // Non-static
static void Main(string[] args) {
      MyClass obj = new MyClass();
      StockQuote stkQuote = new StockQuote();
      stkQuote.highDelegate +=
            new StockQuoteDelegate(User.highReport);
      stkQuote.lowDelegate +=
            new StockQuoteDelegate(User.lowReport);
      stkQuote.highDelegate +=
            new StockQuoteDelegate(obj.beepReport);
      stkQuote.lowDelegate +=
            new StockQuoteDelegate(obj.beepReport);
      …
      stkQuote.highDelegate -=
            new StockQuoteDelegate(obj.beepReport);
}
```

Code Access Security

- Permission granted based on trust level
- Security Demand
  - Your code (class library) demands that other classes calling your methods or accessing objects of your classes have a certain set of permissions (specified by you)
  - All the callers in the Call Stack are checked to see if ***any*** of the callers lack the required permission
    - SecurityException is thrown if that is the case
- Security Overrides
  - allows you to override code permission explicitly
  - you can further restrict your permission before calling a third party code – a way to use other's untrustworthy code

Code Access Security…

```
public class MyClass {
      public void foo() {
            TextReader reader =
             new StreamReader("myfile.dat");
            …
}
// Security demand may be specified using attributes
[FileIOPermission(SecurityAction.Demand,  Unrestricted=true)]
public void f2() {…}
public void f3() {
// This method applies security overrides.
// It demands that the code being called does
// not access any files.
FileIOPermission perm =
      new FileIOPermission(
            PermissionState.Unrestricted);
perm.Deny();
MyClass obj = new MyClass();
obj.foo();
FileIOPermission.RevertDeny();
```

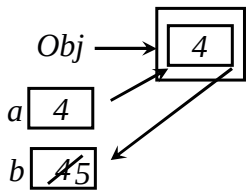
Value Types vs. Reference Types

- Value Types are created on stack by default
- Reference Types are created only on heap
- User defined
  - value types defined using **struct**
  - reference types defined using **class**
- It is hard to see if you are using value type or reference type
- Behavior on copy quite different

Boxing and Unboxing

- Boxing is the conversion or copying of a value type to a reference type
  - The value types is wrapped into a reference intance
- Unboxing is the conversion or copying of a boxed reference to a value type

```
int a = 4;
Object obj = a; // boxing
Int32 b = (int)obj;//unboxing
b = 5;
```



Unsafe vs. Unmanaged Code

- Unmanaged Code:
  - this is not executed under the tight supervision of CLR
    - no garbage collection
    - limited debugging capabilities
  - Useful to call Platform Specific functions
- Unsafe Code:
  - this is managed code!
  - it simply uses some constructs (like pointer usage) that C# does not encourage

Unsafe Code

- CLR manages memory
- Java does not allow pointer manipulation
- C# derived from C++, wants to allow it, however with caution
- Code that manipulates pointers may lead to memory leaks, etc.
- C# declares the section of code that manipulates pointers as unsafe!
  - your take care of memory management when within this block of code – allows you to use pointers
- To prevent GC use the fixed keyword to ***pin***

Usage of unsafe

```
unsafe public static void usePointers(int[] array)
{
        fixed(int* ptrArray = array) {
                for (int i = 0; i < array.Length; i++)
                        Console.Write(*(ptrArray + i) + " ");
        }
}
…
{       ValTypeX instOfX;
        ValTypeX[] myXArray = new ValTypeX[2];
        RefTypeY objY = new RefTypeY();
        …
        unsafe
        {
                ValTypeX* ptrX = &instOfX;
                // No need to use fixed for Value type (on the stack, remember!)
                Console.WriteLine(ptrX->val);
                fixed(ValTypeX* ptrX2 = myXArray)  {
                        Console.WriteLine(ptrX2->val);
                }
                fixed(int* pVal = &objY.val)  {
                        Console.WriteLine(*pVal);
                }
                …
```


Operator Overloading

- C++ has operator overloading
  - one can debate if this is a feature or a flaw
- Java smartly avoided this for good reasons
- It is disappointing to see that C# took this up!
- Note operator overloading is not supported across .NET languages
  - If you want your code to be CLI compliant, you must provide a regular method for each overloaded one
- C# has gone overboard with overloading
  - May overload true, false, &&, || (gets pretty messy)
  - This is surely a *feature* to be avoided

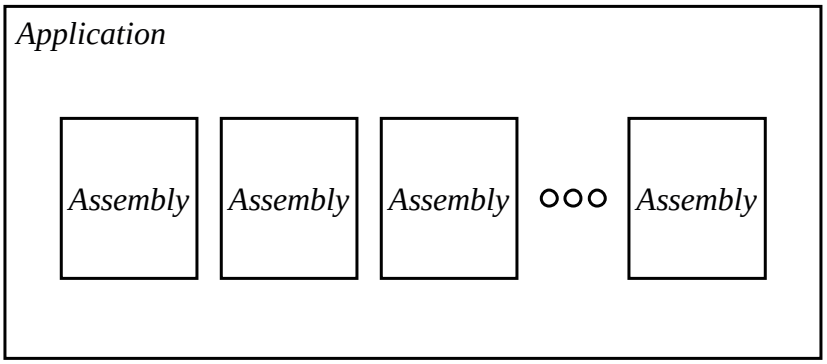
Quiz Time!

Compiling from Command Line

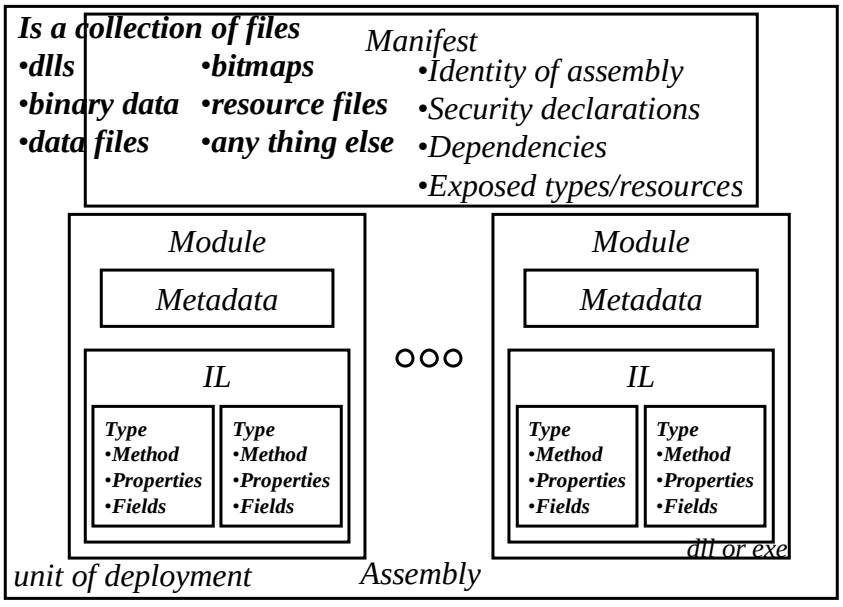
- csc is the C# compiler
- You can create two kinds of assemblies
  - dll
    - csc /target:library Car.cs
  - exe
    - csc /out:prog.exe User.cs
- In order to reference another assembly
  - csc /out:prog.exe /lib:c:\release
    /reference:Car.dll User.cs

  Remember to set path properly or bring up .NET command prompt

What's an Assembly?

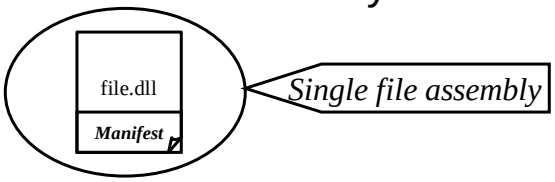


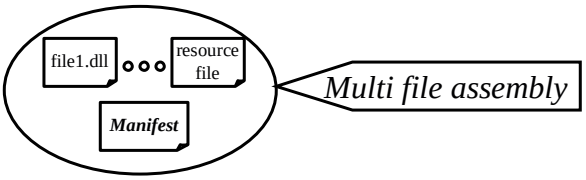
Assemblies



Manifest

- Describes details about the assembly
  - version
  - security
  - scope
  - resources
  - classes
  - types
  - dependencies

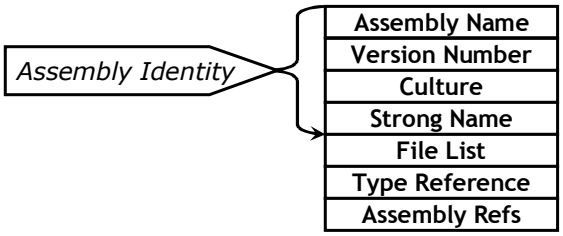




- Stored either
  - as a Portable Executable (PE) file along with MISL
  - or as stand along PE file with only the manifest info

Manifest Info

- Provides enumeration of all files in assembly
- This makes the assembly self-describing
- Maps references to types, resources & files with their declarations/implementations
- Provides details of assemblies that containing assembly depends on



- Additional information may be provided in manifest
  - company, version, copyright, trademark, etc.

Assembly Types

- Private Assemblies
  - the default
  - only intended for use by one application
  - if several applications use, each will have a copy
    - (not the intent)
- Shared Assemblies
  - this is intended for use by multiple applications
  - name must be unique so it does not collide
  - place in the global assembly cache

Building and Deploying Assembly

- Creating a Library Project
  - Make sure your class implements IComponent or derives from System.ComponentModel.Component
- Introspect the assembly using ildasm (covered next)
- Generating shared names from keys
  - a public-private key pair is generated for the assembly
    - **`sn –k myKeyFile.snk`**
  - a hash value is created for names and contents of files
  - the hash is then signed using the private key
  - public key is then placed into the name of the assembly
- Signing the assembly
  - modify an entry in the AssemblyInfo to indicate the key file
  - **[assembly:AssemblyKeyFile(“myKeyFile.snk”)]**
  - you may also replace the key at a later time by running sn with –R option
- Placing the assembly in cache using the gacutil tool
  - **`gacutil /i:`*`MyComponent`*`.dll`**

Assembly Versioning

- An assembly user generally ***binds*** to a specific version of an assembly it uses
- Runtime performs checks to identify assembly:
  - checks original assembly for version to bind to
  - checks configuration files to determine version
  - looks in **Global Assembly Cache** or local dir for that version
- This deals with only Assembly Version Number
- Informational Version
  - another number for information may be provided
  - not used by the run time, though

Quiz Time!

What does WinForms provide?

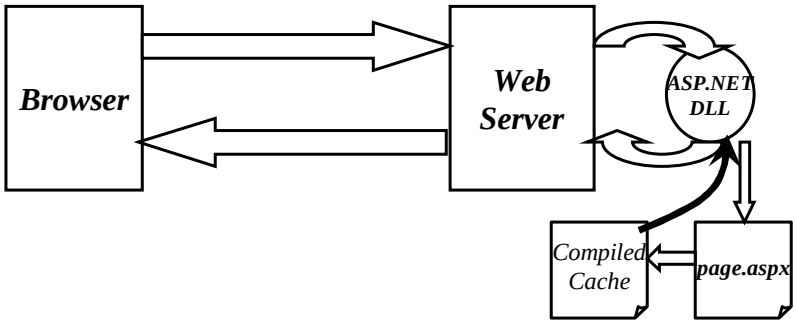
- Windows Forms or WinForms are classes that provide User Interfaces
- These bring rich VB UI to all .NET languages
- Set of class library to develop efficient UI on .NET
- Provides visual inheritance
  - If you inherit from a class, you inherit its UI as well
  - You can say inherit from a dialog and simply add a button

Form Class

- Representation of any window in the application
- You can create an object of this, set it to be modal, create as a dialog, create and attach MDI windows

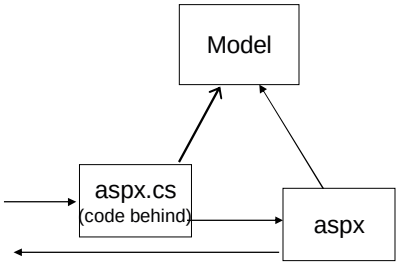
```
public class Form : ContainerControl {
      public static Form ActiveForm {get; }
      public IButtonControl AcceptButton {get; set; }
      …
      public Cursor Cursor {get; set; }
      public bool MaximizeBox {get; set; }
      …
      public void Activate();
      public void Close();
      public void Refresh();
      …
}
```

ASP.NET and Web Server



Code-Behind Page

- Separates the UI from the control
- Pay layout can be done using drag and drop
- Event handling can be done using C# code
- Gives exceptional reuse and extensibility



ASP.NET provides events

- In ASP, page executed top to bottom, sequentially
- ASP.NET provides event handles
  - Page_Init
  - Page_Load
  - Events referred by control on web page

```
private void Page_Load(object sender, System.EventArgs e)
      {
            Response.Write("Page_Load called<BR>");
      }
private void Page_Init(object sender, EventArgs e)
      {
            InitializeComponent();
            Response.Write("Page_Init called<BR>");
      }
```


HTML Server Controls

- Think of these as macros
  - they expand to HTML code
- They appear like HTML tags
  - But much shorter and replace several lines of HTML
- Smart enough to remember state between calls

```
<FORM RUNAT="SERVER">
State:
<ASP:ListBox ID="StateSelect" RUNAT="SERVER">
      <ASP:ListItem>CA</ASP:ListItem>
      <ASP:ListItem>GA</ASP:ListItem>
      <ASP:ListItem>MA</ASP:ListItem>
      <ASP:ListItem>TX</ASP:ListItem>
      <ASP:ListItem>VA</ASP:ListItem>
</ASP:ListBox>
<INPUT type="SUBMIT" value="Send">
</FORM>
```

Server-side Event Handling

- Traditionally in ASP, browser handles client events
- Some times we want to perform some thing on server
  - like accessing additional data
- Still good idea to handle client events on client side
- Allows you to handle events on server side as well
- Browser handles event and sends postback to server

Validation Controls

- How do you check the validity of input data
- Requires writing JavaScript or VBScrtipt
- Difficult to update and maintain
  - especially if need to check with data on backend

Quiz Time!

Summary

- Visited syntax similarities
- Visited syntax differences
- Visited Attributes, Delegates, IDisposable, Properties, foreach, unsafe code, operator overloading, code based security
- Discussed Assemblies, assembly types, versioning
- WinForm features
- ASP.NET Code behind page concept
- Easier for Java developers to benefit

Thank you!

- Please fill your evaluations!
- http://www.agiledeveloper.com/download.aspx
- You may download this and other presentations from above URL