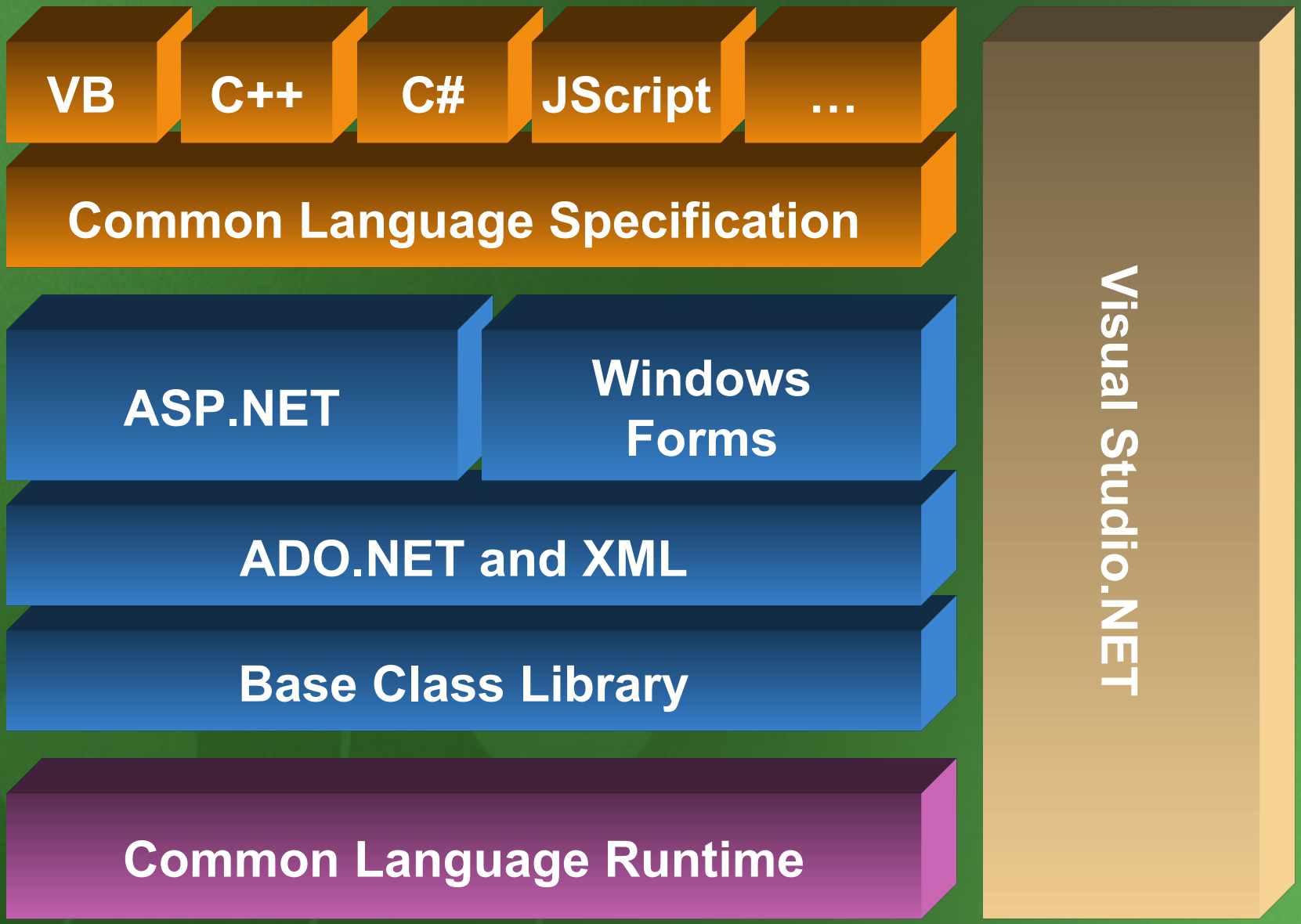
# .NET Framework and C#

Anders Hejlsberg
Distinguished Engineer
Microsoft

Zurich, May 28, 2001

# The .NET Framework



# The .NET Framework

- Simplifies application development
  - No COM plumbing, OOP, interoperability
- Based on web standards / practices
- Robust execution environment
  - GC, exceptions, type-safety, security
- Multiple programming languages
- Easy deployment and management
  - Zero impact install, side-by-side
- Standards work in progress

# Base Class Library

- Data types, conversions, formatting
- Collections: ArrayList, Hashtable, etc.
- Globalization: Cultures, sorting, etc.
- I/O: Binary and text streams, files, etc.
- Net: HTTP, TCP/IP sockets, etc.
- Reflection: Metadata and IL emit
- Security: Permissions, cryptography
- Text: Encodings, regular expressions

# Windows Forms

- Combines VB forms and MFC
  - Delegation as well as subclassing
- Advanced features
  - Visual forms inheritance, automatic layout
  - Advanced graphics support – GDI+
  - Easy access to Win32 ® API
- Controls can be hosted in IE 5.x
  - No installation, registration or GUIDs
- Code access security

# ADO.NET and XML

- Consumes all types of data
  - XML (hierarchical), Relational
- Powerful in-memory data cache
  - Lightweight, stateless, disconnected
  - Supports both relational and XML access
  - High-perf, low overhead stream access
- Great XML support including:
  - W3C DOM, XSL/T, XPath, and Schema

# ASP.NET

- Rich page architecture
  - Web Forms, Web Controls
- Great Web Services support
- Compiled languages
- Easier to deploy
- Enhanced reliability and availability
- Improved performance and scalability
- Automatic multiple client support
  - DHTML, HTML 3.2, WML, small devices

# Multi-Language Platform

- The .NET Platform is Language Neutral
  - All .NET languages are first class players
  - You can leverage your existing skills
- CLR = Union of language features
- CLS = Intersection of language features
- Microsoft is providing
  - VB, C++, C#, Jscript
- Industry and academia
  - APL, COBOL, Eiffel, Fortran, Haskell, ML, Perl, Python, Scheme, Smalltalk, …

# Standardization

- CLI and C# submitted to ECMA
  - Proposal adopted at ECMA TC39 meeting in September 2000
  - Co-sponsored by Intel, Hewlett-Packard
- Common Language Infrastructure
  - Based on .NET Common Language Runtime and Class Libraries
  - Layered into increasing levels of functionality

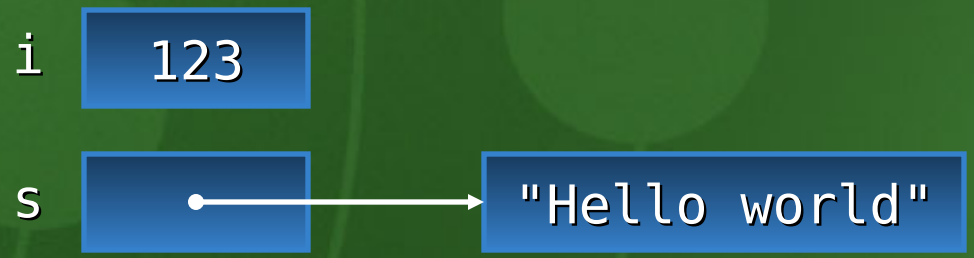
# C# – The Big Ideas

- The first component oriented language in the C and C++ family
- Everything really is an object
- Robustness and durability
- Preserving your investment

# C# Type System

- Value types
  - Directly contain data
  - Cannot be null
- Reference types
  - Contain references to objects
  - May be null

```
int i = 123;
string s = "Hello world";
```



# C# Type System

- **Value types**
  - **Primitives** `int i; double d;`
  - **Enums** `enum State { Off, On }`
  - **Structs** `struct Point { int x, y; }`
- **Reference types**
  - **Classes** `class Foo: Bar, IFoo {...}`
  - **Interfaces** `interface IFoo: IBar {...}`
  - **Arrays** `string[] a = new string[10];`
  - **Delegates** `delegate void Empty();`

# Structs

- Like classes, except
  - Stored in-line, not heap allocated
  - Assignment copies data, not reference
  - No inheritance
- Ideal for light weight objects
  - Complex, point, rectangle, color
  - int, float, double, etc., are all structs
- Benefits
  - No heap allocation, less GC pressure
  - More efficient use of memory

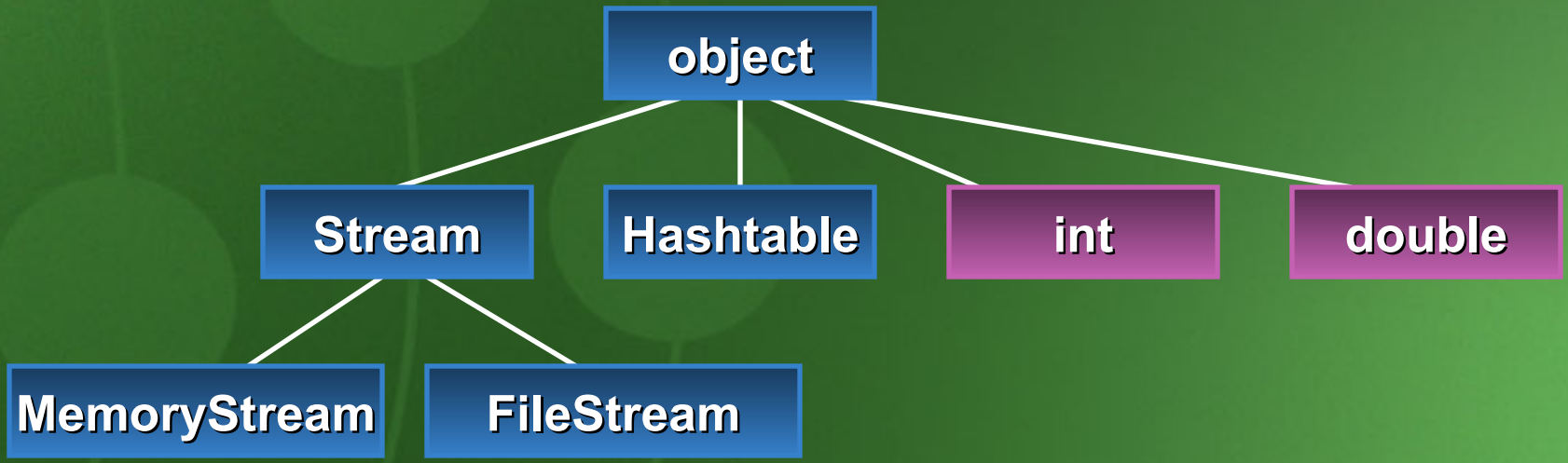
# Classes and Structs

```
 class CPoint { int x, y; ... }
struct SPoint { int x, y; ... }

CPoint cp = new CPoint(10, 20);
SPoint sp = new SPoint(10, 20);
```

# Unified Type System

- Everything is an object
  - All types ultimately inherit from object
  - Any piece of data can be stored, transported, and manipulated with no extra work



# Unified Type System

- Boxing
  - Allocates box, copies value into it
- Unboxing
  - Checks type of box, copies value out

```
int i = 123;
object o = i;
int j = (int)o;
```

# Unified Type System

- Benefits
  - Eliminates “wrapper classes”
  - Collection classes work with all types
  - Replaces OLE Automation's Variant
- Lots of examples in .NET Framework

```
string s = string.Format(
   "Your total was {0} on {1}", total, date);

Hashtable t = new Hashtable();
t.Add(0, "zero");
t.Add(1, "one");
t.Add(2, "two");
```

# Component Development

- What defines a component?
  - Properties, methods, events
  - Integrated help and documentation
  - Design-time information
- C# has first class support
  - Not naming patterns, adapters, etc.
  - Not external files
- Components are easy to build and consume

# Properties

- Properties are “smart fields”
  - Natural syntax, accessors, inlining

```
public class Button: Control
{
   private string caption;

   public string Caption {
      get {
         return caption;
      }
      set {
         caption = value;
         Repaint();
      }
   }
}
```

```
Button b = new Button();
b.Caption = "OK";
String s = b.Caption;
```

# Indexers

- Indexers are “smart arrays”
  - Can be overloaded

```
public class ListBox: Control
{
   private string[] items;

   public string this[int index] {
      get {
         return items[index];
      }
      set {
         items[index] = value;
         Repaint();
      }
   }
}
```

```
ListBox listBox = new ListBox();
listBox[0] = "hello";
Console.WriteLine(listBox[0]);
```

# Events
## Sourcing

- Define the event signature

```
public delegate void EventHandler(object sender, EventArgs e);
```

- Define the event and firing logic

```
public class Button
{
   public event EventHandler Click;

   protected void OnClick(EventArgs e) {
      if (Click != null) Click(this, e);
   }
}
```

# Events
## Handling

- Define and register event handler

```
public class MyForm: Form
{
   Button okButton;

   public MyForm() {
      okButton = new Button(...);
      okButton.Caption = "OK";
      okButton.Click += new EventHandler(OkButtonClick);
   }

   void OkButtonClick(object sender, EventArgs e) {
      ShowMessage("You pressed the OK button");
   }
}
```

# Attributes

- How do you associate information with types and members?
  - Documentation URL for a class
  - Transaction context for a method
  - XML persistence mapping
- Traditional solutions
  - Add keywords or pragmas to language
  - Use external files, e.g., .IDL, .DEF
- C# solution:  Attributes

# Attributes

```
public class OrderProcessor
{
   [WebMethod]
   public void SubmitOrder(PurchaseOrder order) {...}
}

[XmlRoot("Order", Namespace="urn:acme.b2b-schema.v1")]
public class PurchaseOrder
{
   [XmlElement("shipTo")]  public Address ShipTo;
   [XmlElement("billTo")]  public Address BillTo;
   [XmlElement("comment")] public string Comment;
   [XmlElement("items")]   public Item[] Items;
   [XmlAttribute("date")]  public DateTime OrderDate;
}

public class Address {...}

public class Item {...}
```

# Attributes

- Attributes can be
  - Attached to types and members
  - Examined at run-time using reflection
- Completely extensible
  - Simply a class that inherits from System.Attribute
- Type-safe
  - Arguments checked at compile-time
- Extensive use in .NET Frameworks

# Unsafe Code

- Unsafe code
  - Low-level code without leaving the box
  - Enables unsafe casts, pointer arithmetic
- Declarative pinning
  - Fixed statement
- Basically “inline C”

```
unsafe void Foo() {
   char* buf = stackalloc char[256];
   for (char* p = buf; p < buf + 256; p++) *p = 0;
   ...
}
```

# Unsafe Code

```
class FileStream: Stream
{
   int handle;

   public unsafe int Read(byte[] buffer, int index, int count) {
      int n = 0;
      fixed (byte* p = buffer) {
         ReadFile(handle, p + index, count, &n, null);
      }
      return n;
   }

   [dllimport("kernel32", SetLastError=true)]
   static extern unsafe bool ReadFile(int hFile,
      void* lpBuffer, int nBytesToRead,
      int* nBytesRead, Overlapped* lpOverlapped);
}
```

# Other C# Features

- Versioning
- Multi-dimensional arrays
- Operator overloading
- User-defined conversions
- Variable parameter lists
- ref and out parameters
- Unsigned types (byte, ushort, uint, ulong)
- Decimal type (28 digits)
- Conditional compilation
- foreach statement
- Explicit interface member implementations
- XML documentation comments

Microsoft®