C# : Types & Assemblies

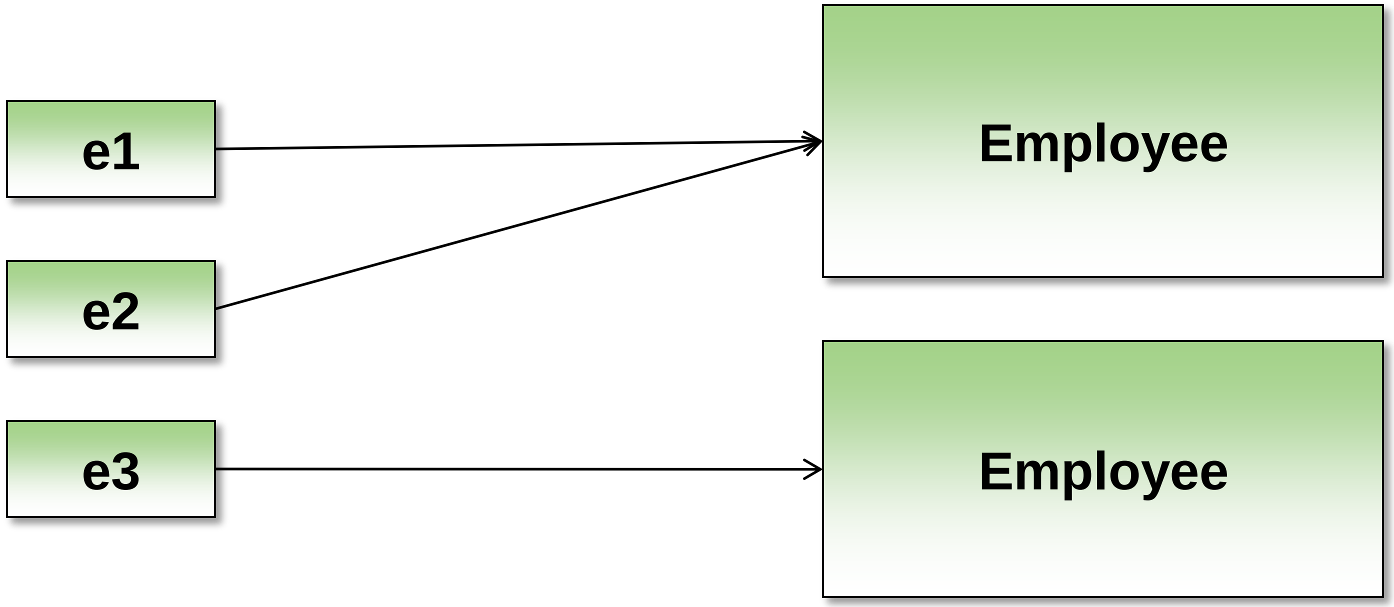
Interfacing with C#

Overview

- Value types and reference types
- Enumerations
- Structs
- Interfaces
- Arrays
- Assemblies
- Assembly references

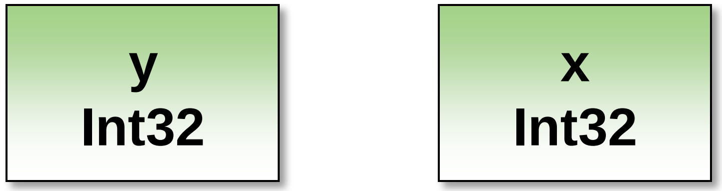
Reference Types

- **Variables store a reference to an object**
  - Multiple variables can point to the same object
  - Single variable can point to multiple objects over it’s lifetime
  - Objects allocated on the heap by new operator



Value Types

- **Variables hold the value**
  - No pointers or references
  - No object allocated on the heap – lightweight
  - Should be immutable
- **Many built-in primitives are value types**
  - Int32, DateTime, Double



Creating Value Types

- **struct definitions create value types**
    - Cannot inherit from a struct (implicitly sealed)
    - Rule of thumb: should be less than 16 bytes

```
public struct Complex
{
    public int Real;
    public int Imaginary;
}
```

Method Parameters

- **Parameters pass “by value”**
  - Reference types pass a copy of the reference
  - Value types pass a copy of the value
  - Changes to value don’t propagate to caller
- **Parameter keywords**
  - ref and out keywords allow pass “by reference”
  - ref parameters requires initialized variable

```
public bool Work(ref string text, out int age)
{
    return Int32.TryParse(text, out age);
}
```

The Magical String Type

- **Strings are reference types**
  - But behave like value types
  - Immutable
  - Checking for equality performs a string comparison

```
string s1 = "Vitamin";
string s2 = "Vitamin";

bool result = s1 == s2;

result = s1.Equals(s2,
         StringComparison.InvariantCultureIgnoreCase);
```

Boxing & Unboxing

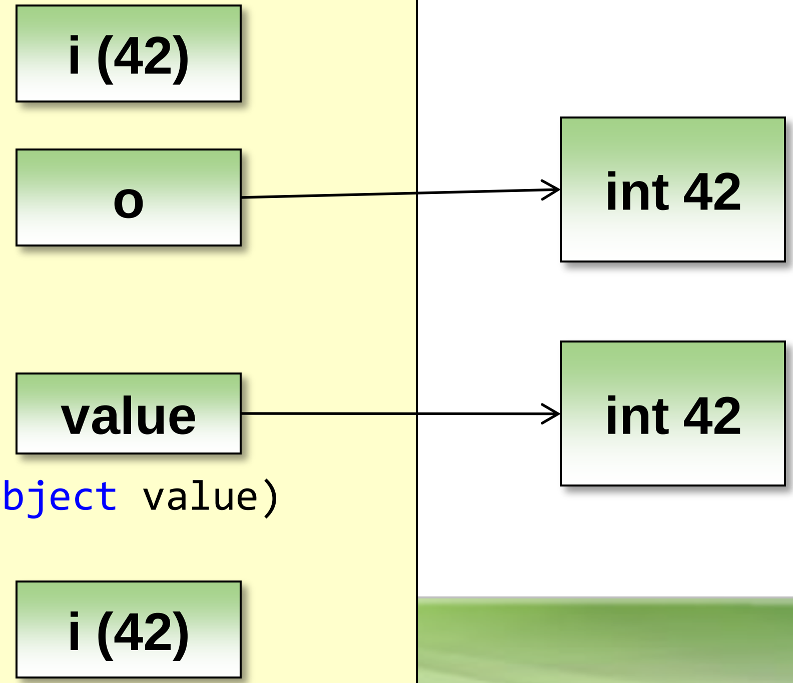
- **Boxing converts a value type to an object**
  - Copies value into allocated memory on the heap
  - Can lead to performance and memory consumption problems
- **Unboxing converts an object to a value type**

```
public static void Main()
{
    int i = 42;

    object o = i; // box

    DoWork(i); // box
}

private static void DoWork(object value)
{
    int i = (int) value;
}
```



Enumerations

- **An enum creates a value type**
  - A set of named constants
  - Underlying data type is int by default

```
public enum PayrollType
{
    Contractor = 1,
    Salaried,
    Executive,
    Hourly
}
```

```
if(e.Role == PayrollType.Hourly)
{
    // ...
}
```

What Makes a Value Type & Reference Type?

- **Value Type**
  - struct
  - enum
- **Reference Type**
  - class
  - interface
  - delegate
  - array

Interfaces

- **An interface defines a group of related methods, properties, and events.**
  - No implementation defined in interface (very abstract)
  - All members are public
  - Classes and structs can inherit from an interface and provide an implementation
  - Classes and structs can inherit from multiple interfaces

```
interface IMessageLogger
{
    void LogMessage(string message);
}
```

```
class FileSystemLogger : IMessageLogger
{
    public void LogMessage(string message)
    {
        // ....
    }
}
```

Arrays

- **Simple data structure for managing a collection of variables**
    - Everything inside has the same type
    - Always 0 indexed
    - Always derive from abstract base type Array
    - Single-dimensional, multi-dimensional, and jagged

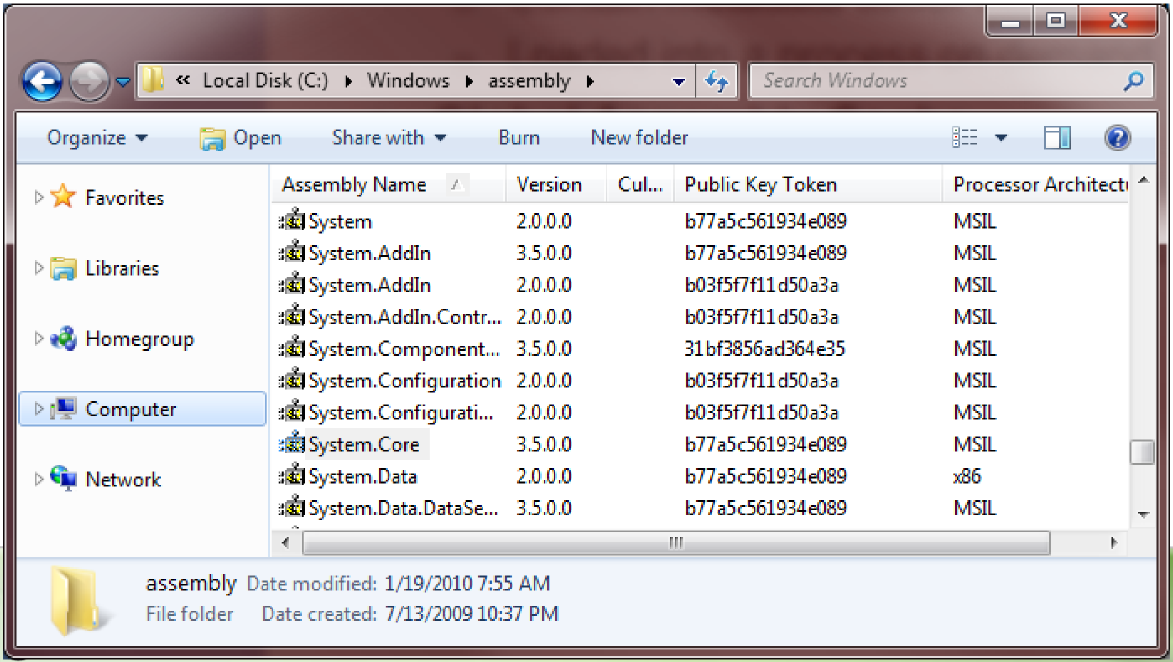
```
const int numberOfBowlers = 4;
int[] scores = new int[numberOfBowlers];

int totalScore = 0;
foreach(int score in scores)
{
    totalScore += score;
}

double averageScore = (double)totalScore / scores.Length;
```

Assemblies

- **Fundamental building blocks**
  - Implemented as .exe or .dll files
  - Contain metadata about version and all types inside
- **Global Assembly Cache**
  - A central location to store assemblies for a machine
  - Assembly in the GAC requires a strong name

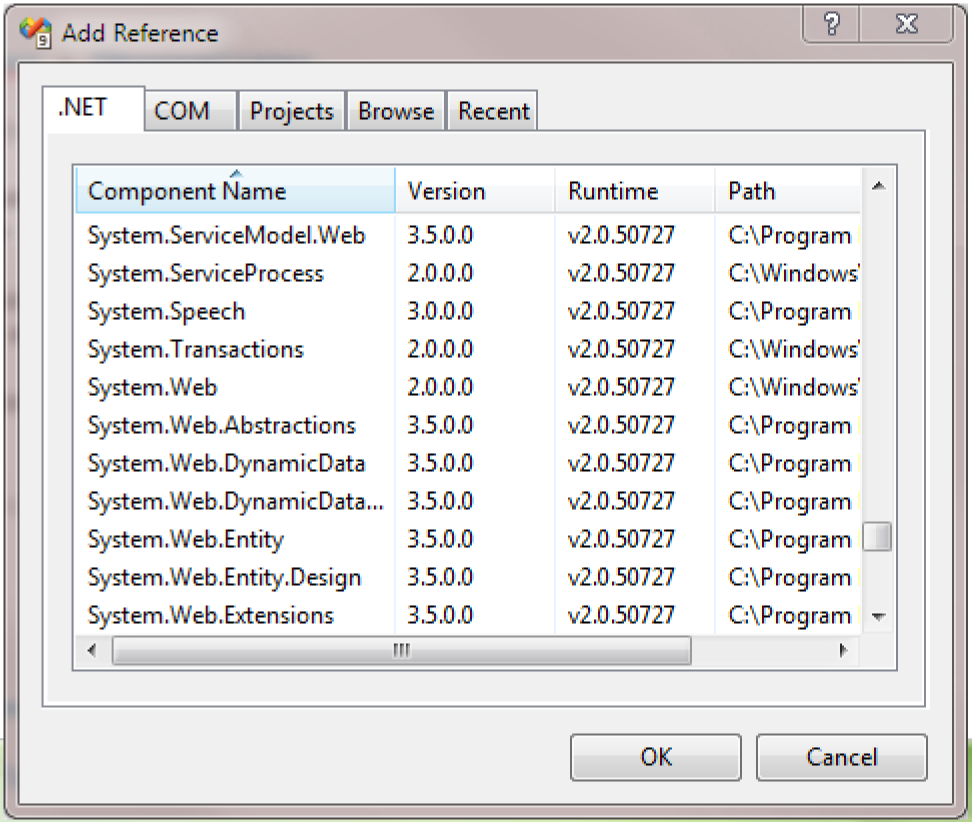


Assembly Name	Version	Cul...	Public Key Token	Processor Architectu
System	2.0.0.0		b77a5c561934e089	MSIL
System.AddIn	3.5.0.0		b77a5c561934e089	MSIL
System.AddIn	2.0.0.0		b03f5f7f11d50a3a	MSIL
System.AddIn.Contr...	2.0.0.0		b03f5f7f11d50a3a	MSIL
System.Component...	3.5.0.0		31bf3856ad364e35	MSIL
System.Configuration	2.0.0.0		b03f5f7f11d50a3a	MSIL
System.Configurati...	2.0.0.0		b03f5f7f11d50a3a	MSIL
System.Core	3.5.0.0		b77a5c561934e089	MSIL
System.Data	2.0.0.0		b77a5c561934e089	x86
System.Data.DataSe...	3.5.0.0		b77a5c561934e089	MSIL

assembly Date modified: 1/19/2010 7:55 AM
File folder Date created: 7/13/2009 10:37 PM

References

- **Must load assembly into a process before using types inside**
  - Easy approach – reference the assembly in Visual Studio
  - Assemblies loaded on demand at runtime



Component Name	Version	Runtime	Path
System.ServiceModel.Web	3.5.0.0	v2.0.50727	C:\Program
System.ServiceProcess	2.0.0.0	v2.0.50727	C:\Windows'
System.Speech	3.0.0.0	v2.0.50727	C:\Program
System.Transactions	2.0.0.0	v2.0.50727	C:\Windows'
System.Web	2.0.0.0	v2.0.50727	C:\Windows'
System.Web.Abstractions	3.5.0.0	v2.0.50727	C:\Program
System.Web.DynamicData	3.5.0.0	v2.0.50727	C:\Program
System.Web.DynamicData...	3.5.0.0	v2.0.50727	C:\Program
System.Web.Entity	3.5.0.0	v2.0.50727	C:\Program
System.Web.Entity.Design	3.5.0.0	v2.0.50727	C:\Program
System.Web.Extensions	3.5.0.0	v2.0.50727	C:\Program

Summary

- **Every type is a value type or reference type**
  - Use struct to create a value type
  - Use class to create a reference type
- **Arrays and strings are reference types**
  - Strings behave like a value type