

ITU

Me

M.Sc. IT, ITU

Thesis: Forecalc – Developing a core spreadsheet implementation in F#

Senior Consultant, Microsoft
Cloud Development, Infrastructure, and Security

Associate Professor, ITU
Object-Oriented Programming, C#, F#, .NET Core

Captain, Danish Army (Reserve)
Acting Battalion Chief of Staff, Battalion S3

Wife: Katrine, Children: Laura (2) and Alma (9)
Origin: Aarhus, Current whereabouts: Vanløse, Copenhagen

Hobbies



Agenda

TDD

.NET (Core)

C#

Visual Studio 2017

Test-Driven Development

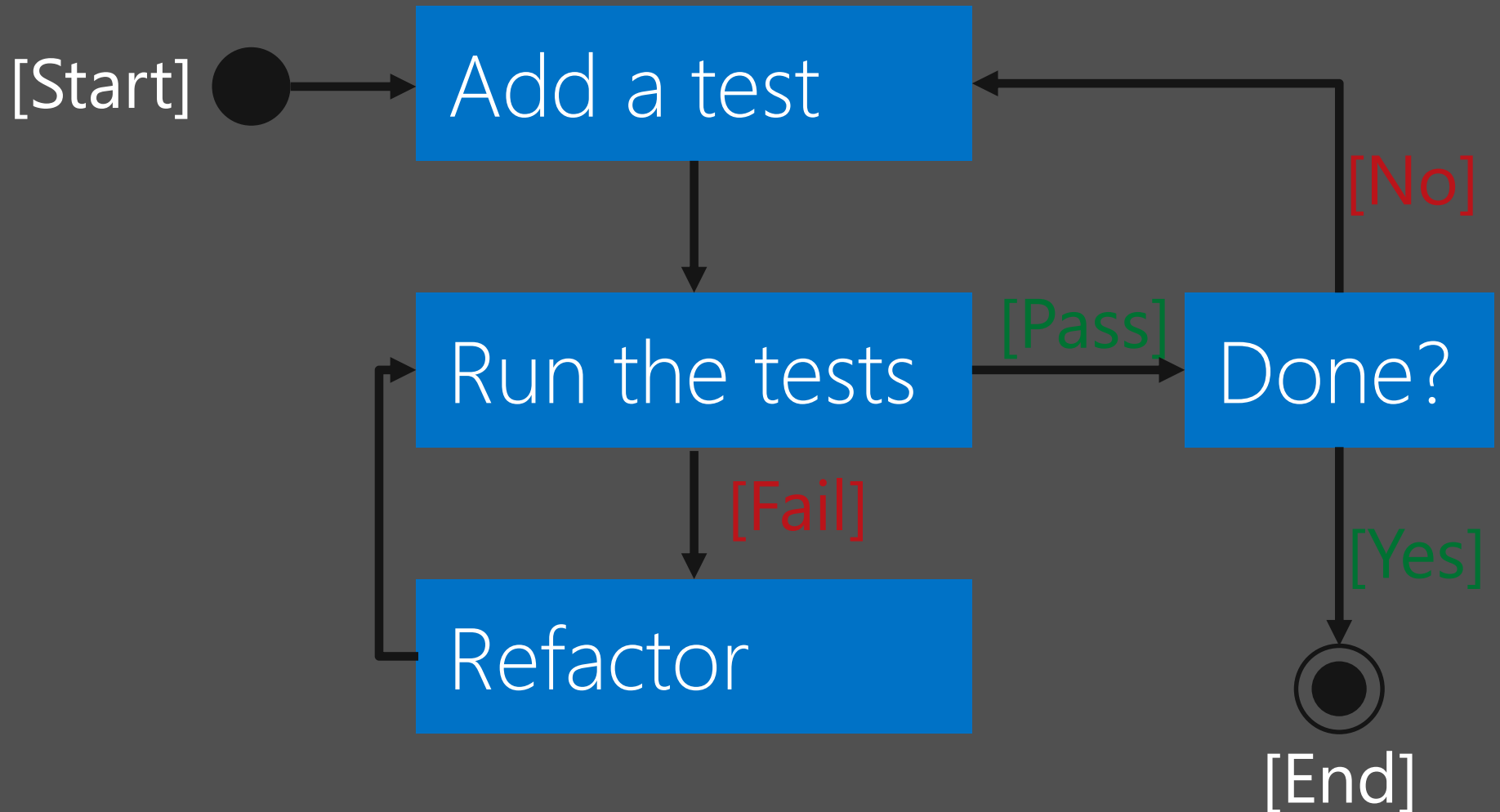
What?

Why?

How?

How?

RED-GREEN-REFACTOR



C#

Show me the CODE!!!

C# is intended to be a simple, modern, general-purpose, object-oriented programming language.

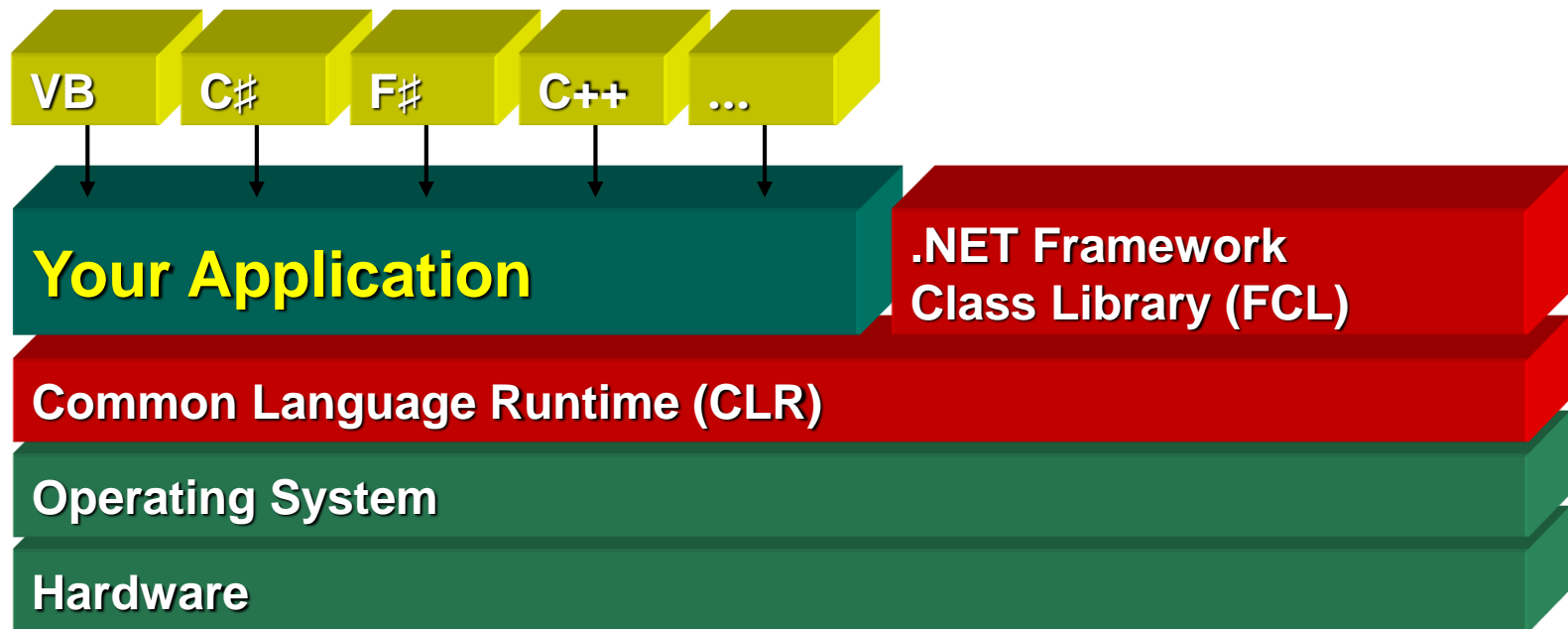
Ecma International (2006)



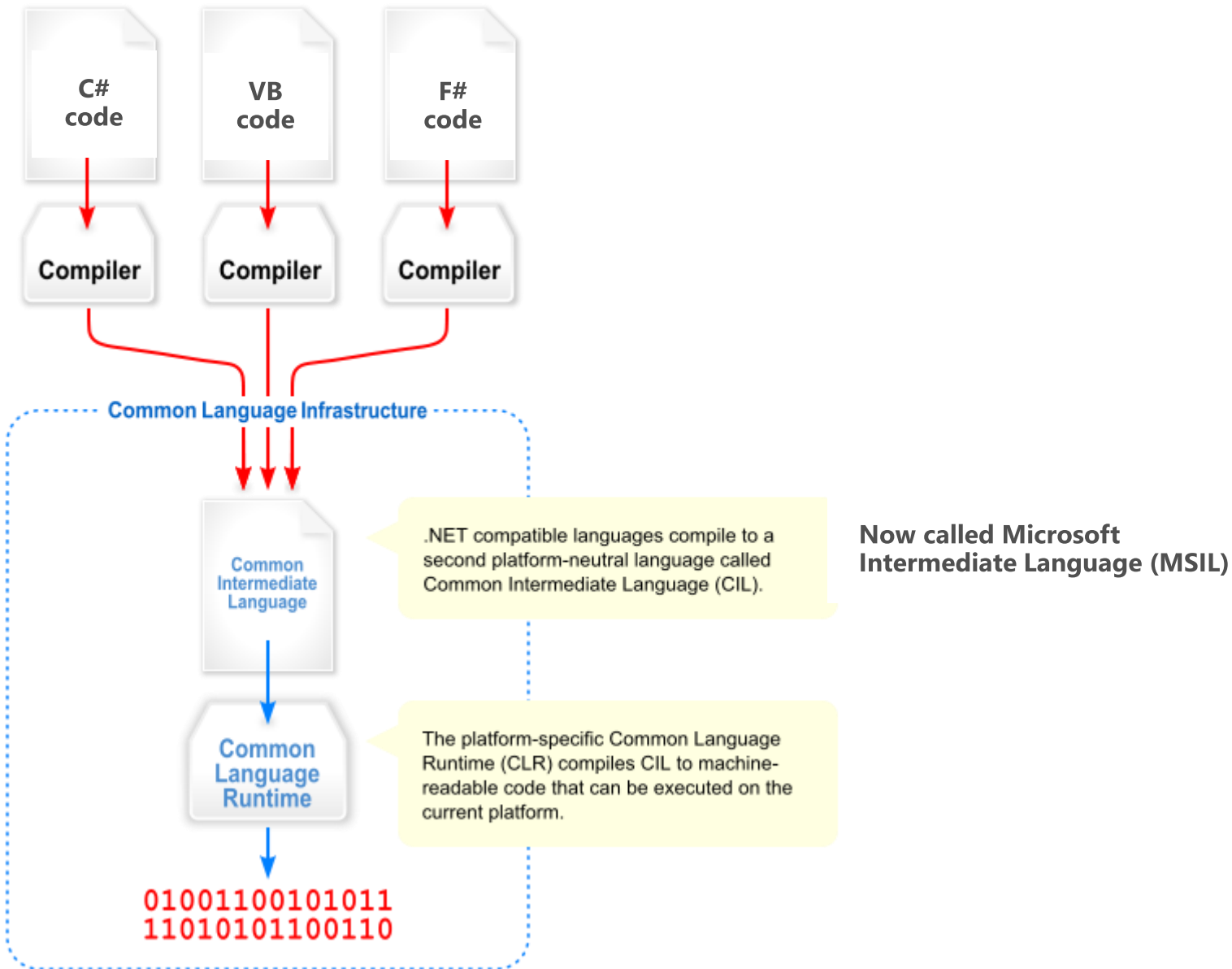
A brief introduction

.NET Framework

.NET offers multiple languages, a large class library, driven by a virtual machine



.NET Framework Compilation



.NET Framework

Versions

1.0 Visual Studio .NET (2002)

1.1 Visual Studio .NET 2003

2.0 Visual Studio 2005

3.0 (2006)

3.5 Visual Studio 2008 (2007)

4.0 Visual Studio 2010

4.5 Visual Studio 2012

4.5.1 Visual Studio 2013

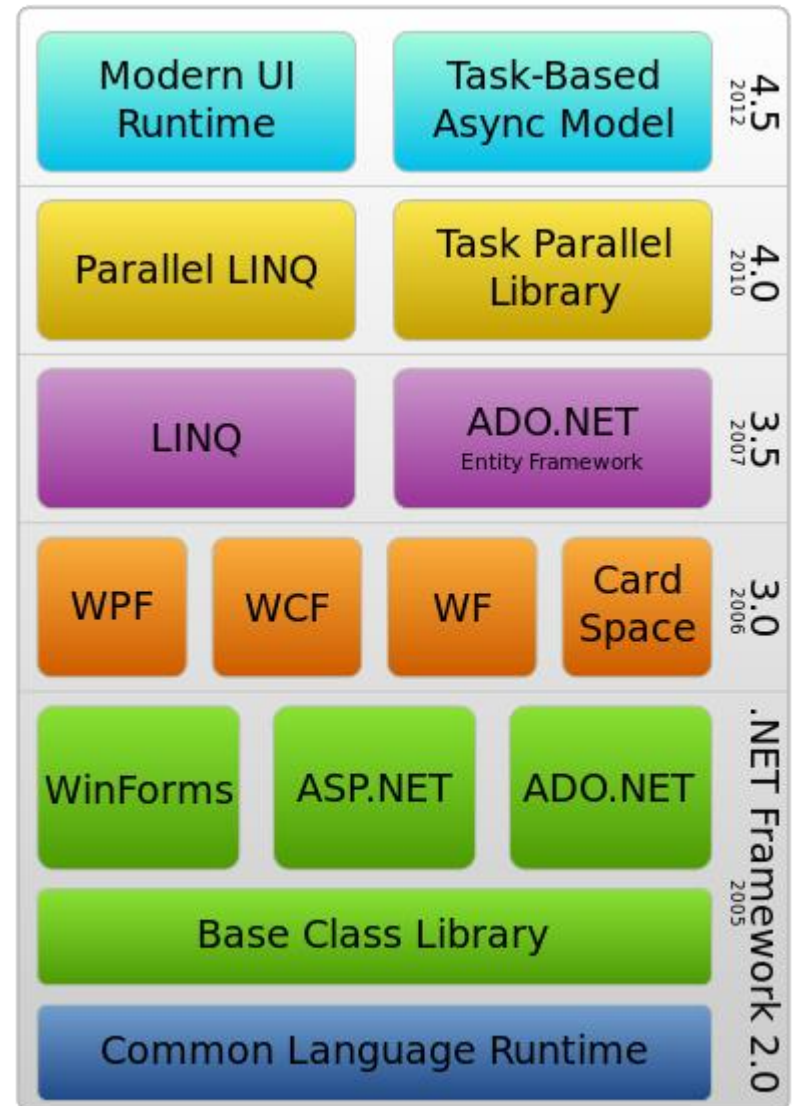
4.5.2 (2014)

4.6 Visual Studio 2015

4.6.1 (2015)

4.6.2 (2016)

.NET Core 1.0 (2016)



The .NET Framework Stack

C#

Language Basics

Naming Conventions

Composed names

currentLayout, CurrentLayout

Properties

Pi, Name, Size

Variables and fields

vehicle, leftElement

Classes

MyClass, List<T>

Private fields

_vehicle, _leftElement

Interfaces

IException, IObservable

Methods

CurrentVehicle(), Size()

[http://msdn.microsoft.com/en-us/library/ms229002\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/ms229002(v=vs.110).aspx)

Value Types can never be null!

Value Types

Holds a value – assignment copies the value

Struct

- Numeric types
 - Integral types
 - Floating point types
 - Decimal
- bool

Integral
byte, sbyte,
short, ushort,
Char,
int, uint,
long, ulong

Floating point
float
Double

Decimal
decimal

Enumeration

```
enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

User defined structs

System.Guid

System.Drawing.Point

System.DateTime

System.Numerics.BigInteger

System.Numerics.Complex

Value Types

```
int age;
```

```
System.Int32 age;
```

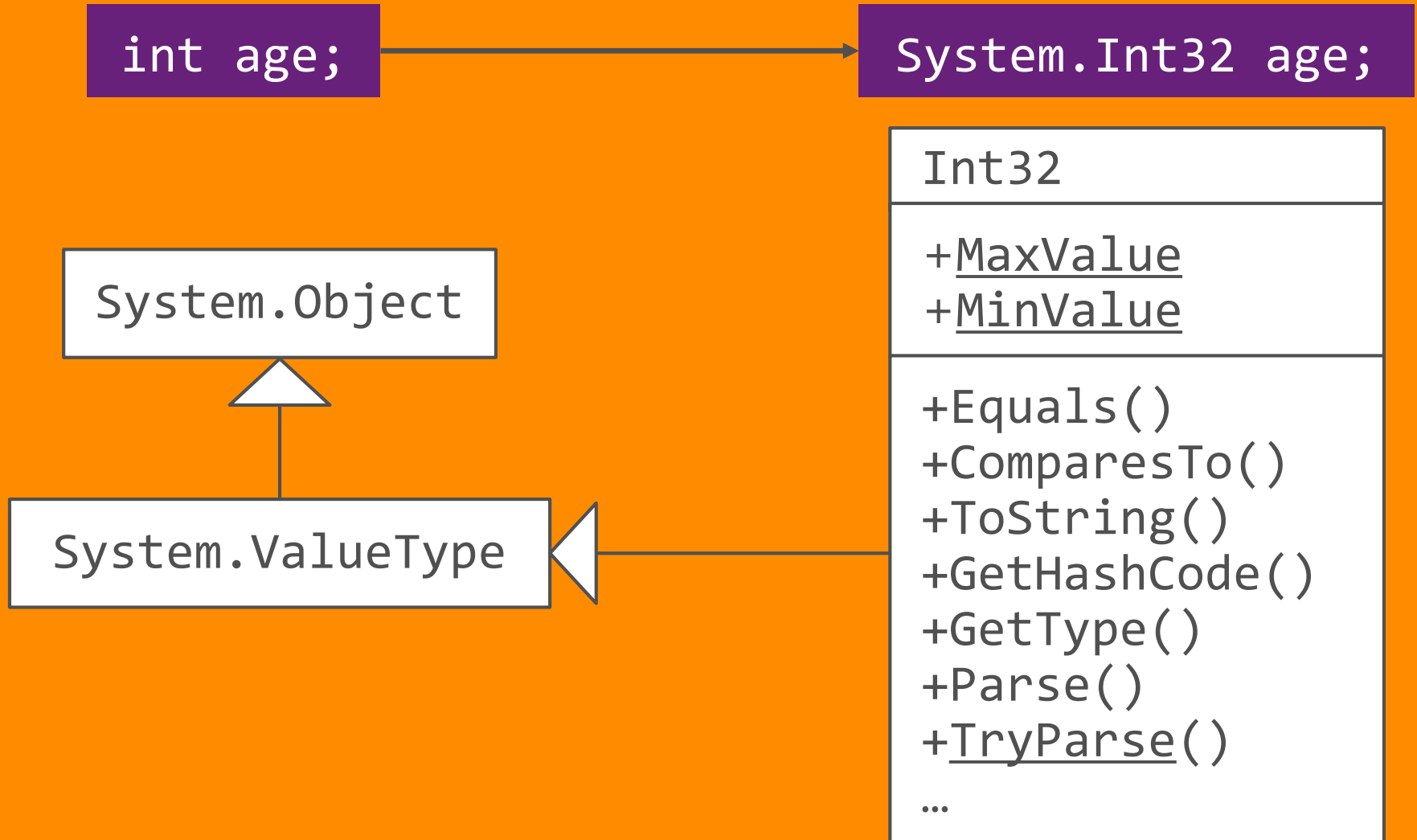
System.Object

System.ValueType

Int32

+MaxValue
+MinValue

+Equals()
+CompareTo()
+ToString()
+GetHashCode()
+GetType()
+Parse()
+TryParse()
...



Value Types

```
int i1 = 42;
```

Memory

i1: int

i2: int

```
int i2 = i2;
```

ReferenceEquals is always *false*

Reference Types

```
var car = new Vehicle();
```

```
Vehicle audi = null;  
audi = car;
```

Memory

Vehicle:
object



Reference Type Equality

ReferenceEquality:

```
Person p1 = new Person("Joe");
```

```
Person p2 = new Person("Joe");
```

```
Person p3 = p2;
```

```
ReferenceEquality(p1, p2) = false
```

```
ReferenceEquality(p2, p3) = true;
```

In C# the == operator is "equal" to reference equality. (Can be overridden)

Value Equality (for reference types)

```
p1.Equals(p2) = true; (Can be overridden)
```

Value Type Equality

Equals the same as for reference types

`object.ReferenceEquality` will always return false for value types

`==` operator is overridden so it does value equality

String Interning

```
string a = "Peter";  
string b = "Peter";
```

```
a.Equals(b);    ==> true
```

```
a == b;         ==> true
```

```
object.ReferenceEquals(a, b); ==> true
```

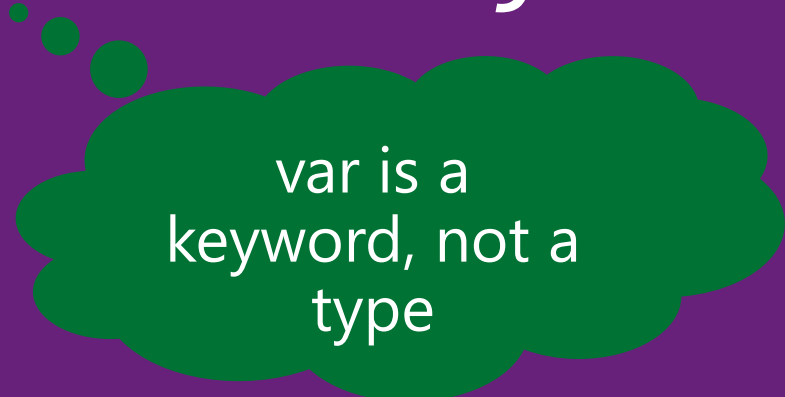


String
Interning

The String Type is Immutable – assigning creates a new value...

Local Variable Type Inference

var *identifier* = *expression*;



var is a
keyword, not a
type

Enumeration

```
public enum Day { Mon, Tue, Wed, Thu, Fri, Sat, Sun }
```

```
public enum Month : uint { Jan = 1, Feb, Mar, }
```

```
public enum Color : uint { Red = 0xFF0000,  
                           Green = 0x00FF00,  
                           Blue = 0x0000FF }
```

```
Vehicle car = new Vehicle();  
car.Color = Color.Red;
```

```
Console.Write(Day.Mon);
```

Array stuff

```
int[] intArray = new int[4];
```

```
double[,] doubleArray = new double[4, 5];
```

```
int[,] array1 = {{1,2},{3,4}};
```

```
int value1 = intArray[0];
```

```
double value2 = doubleArray[0,1];
```

```
Console.WriteLine(array1[1,1]);
```

Arrays are 0-based

String stuff

```
public static void Main(string[] args)
{
    var name = "Anders";
    var argument = args[0];

    Console.WriteLine(10 + " hello " + name + argument);
}
```



Automatic
conversion

Compile from Command Line

Developer Command Prompt for VS 2017:

```
C:\[program_dir]> csc
```

.NET Core:

```
$ dotnet run
```

Command Line Options

Compiler Option	Short Form	Description
<code>/target:exe</code> <code>/target:library</code>	<code>/t:exe</code> <code>/t:library</code>	Compile to an executable (.exe file) (default) Compile to a library (.dll file)
<code>/reference:Lib.dll</code>	<code>/r:Lib.dll</code>	Include reference to library Lib.dll
<code>/main:MyClass</code>	<code>/m:MyClass</code>	Method Main in MyClass is the entry point
<code>/debug</code>	<code>/d</code>	Add debugging information

```
C:\Program Files (x86)\Microsoft Visual Studio 14.0>csc /?
```