7SENG001 Enterprise Application Development

Week 4 – Programming

Introduction to C#

Introduction to C# Part 1

Object Oriented Development

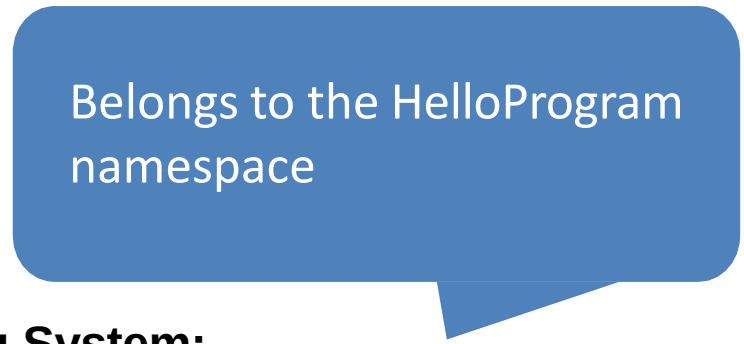
Outline

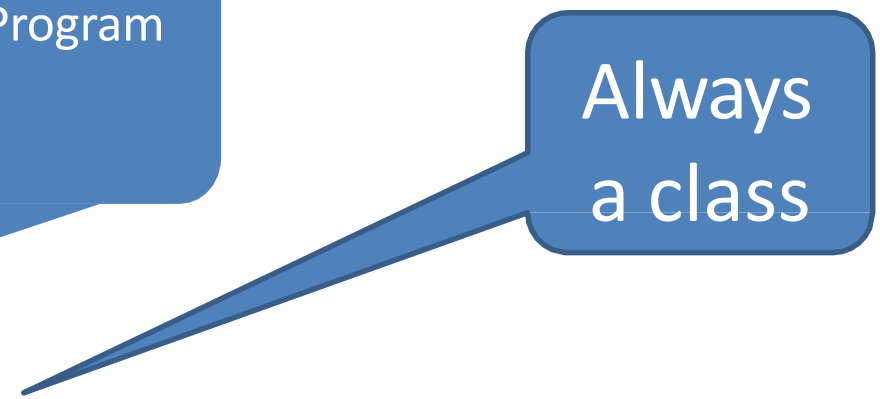
- Brief History
- Compilation and Runtime
- Basic Constructs and types
- Classes and Instances

Brief History

- First appeared 2001
- Developed as part of .NET
- Not a proprietary language (technically an open source specification)
- Microsoft’s version is called Visual C# .NET

C# - A simple program





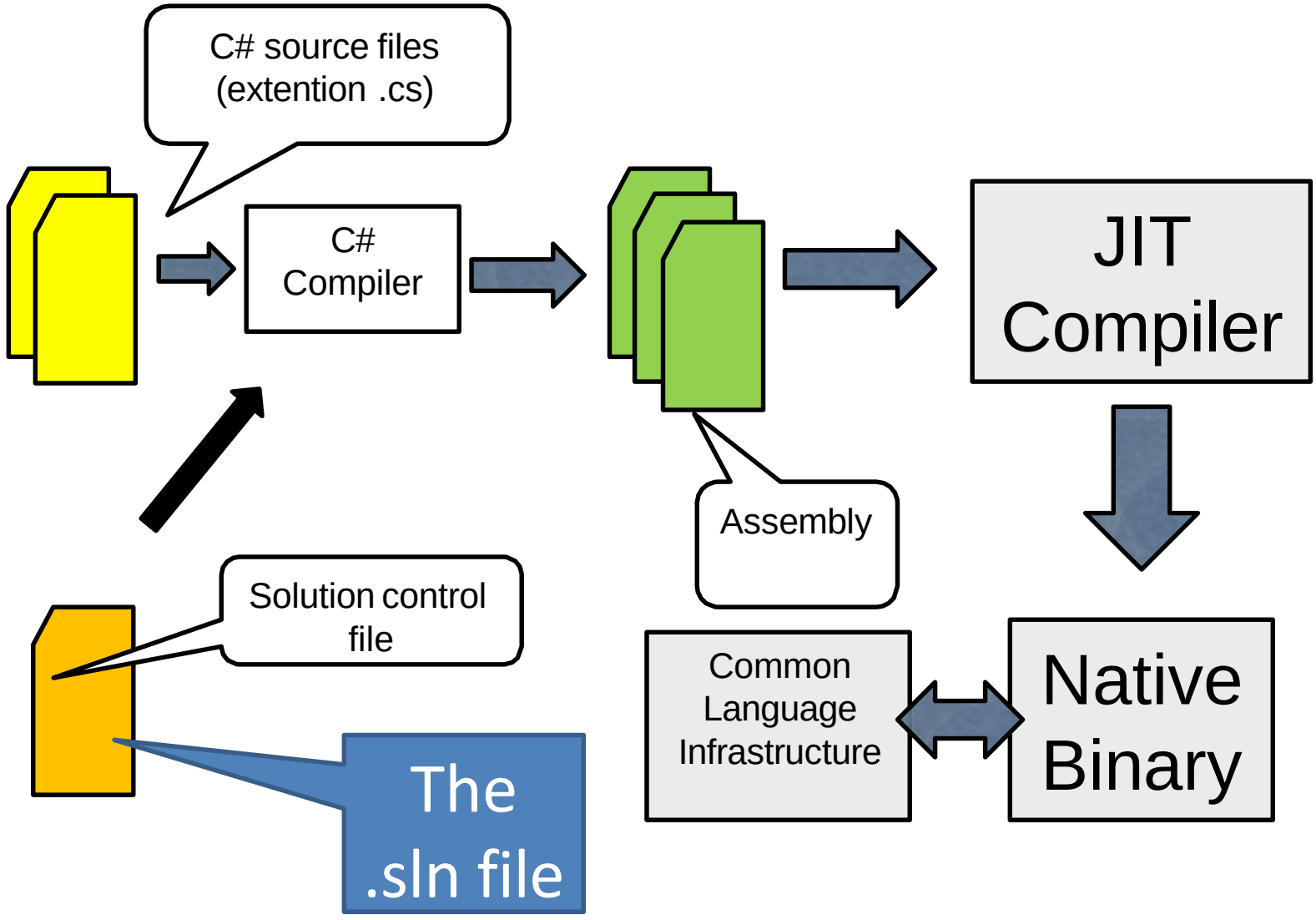

```
using System;
namespace HelloProgram
{
   public class HelloWorld
   {
            public static int Main()
            {
                     Console.Write("Hello World\n");
                     return 0;
            }
   }
}
```

Note that this code is similar to C++ and Java

Compilation Process

- .NET compilers produce **Intermediate Language** (IL) and metadata
  - Just in Time (JIT) compilers converts IL to **native binary code**
  - Programs are run in a **managed** environment
  - Managed means
    - Type safe
    - Automatic Garbage Collection

Compilation Process



Target Platforms

- IL Programs can technically be run on:
  - Windows 98, ME, NT3.1, NT4.0, XP, Vista, Windows 7
  - Linux, Apple OSX (mono project), UNIX
  - **Mobile Devices (PDA, Smart Phone, embedded)**
  - Other systems (**microframework**)
- Requires that a CLI is installed on the target
  - E.g. Microsoft CLR on Windows
  - Compact Framework for embedded
  - For Mac OSX (see mono-project.com)

Recall the Hello Program

```
using System;
namespace HelloProgram
{
   public class HelloWorld
   {
            public static int Main()
            {
                     Console.Write("H
                     ello World\n");
              return 0;
            }
   }
}
```

Namespace

- Overall layout is divided into layers
- Outmost layer begins with **namespace** HelloProgram
- Namespaces are useful for grouping programs together (much like **package**s in Java)
- Code in different source files can belong to the same namespace
- Use is optional but **good practice**

What **can** be in the **namespace** scope

- Classes
- Structures (used extensively in C and eleswhere)
- Enumerations
- Delegates – more on these later
- only **one** Main class

What **can** be in the **class** scope?

- Fields: data members
- Methods : member functions
- Properties: alternative syntax for getting and setting fields (see later)
- Operators: alternative symbolic representation for a method
- Events: runtime function determination and dispatching
- Delegate: function address specification

Methods

- For methods the name in combination with the list of parameters is called the **signature**
- Identification of a method is done by examining the signature (not just the name)
  - Different signature means it is a different method
    - For example foo(int x) and foo(double d) are different methods with the same name
  - This is known as **name overloading**

Alternate specifications of Main

- static void Main()
- static int Main()
- static void Main (string[] args)
  - Used to pass in parameters often from a command line
- static int Main(string[] args)
- Note that there must be **only one** Main (the program entry point)

The using keyword

```
namespace HelloProgram
// using System;
public class HelloWorld
{
        public static int Main()
        {
                System.Console.Write("Hello World\n");
                return 0;
        }
}
```

Commented out

Therefore need to specify the namespace'

Comments

```
/// <summary>
/// This will be included in the
    documentation
///</summary>
/* this is a multiline
 comment */
// single line comment
```

Conditionals: if

- Conditions must be boolean
  - Must always evaluate to **true** or **false**

```
if (i < 5)
{
        //statement block
}
else
{
        //statement block
}
```

Switch

- All but the last case must break or goto
  - Unless no code inside
- The order is now Important!!

C# 7 adds pattern matching

```
switch (myString)
{
        case “Ford”:
                statement block
                break;

        case “Prefect”:
                statement block
                goto case “ford”;

        case “Arthur”:
                statement block
                break;

        default:
                statement block
}
```

C# generalizes the switch statement

- You can switch on any type (not just primitive types)
- Patterns can be used in case clauses
- Case clauses can have additional conditions on them

```
switch(shape)
{
    case Circle c:
        WriteLine($"circle with radius {c.Radius}");
        break;
    case Rectangle s when (s.Length == s.Height):
        WriteLine($"{s.Length} x {s.Height} square");
        break;
    case Rectangle r:
        WriteLine($"{r.Length} x {r.Height} rectangle");
        break;
    default:
        WriteLine("<unknown shape>");
        break;
    case null:
        throw new
ArgumentNullException(nameof(shape));
}
```

Loops: while and do

```
while (i < 5)
{
        statement block
}

  do {
            statement block
     } while (i < 5);
```

Loops: for

```
for (int  i = 0;  i < 5;  i ++)
{
        if (i == 3)
                continue;        // skip this case

        statement block
}
```

String Formating

The escape character

string s = “C:\\Program Files\\”;

string s = “ the “ character delimits strings”;

string interpolation with $

```
return string.Format("{0} {1}", FirstName, LastName);          //Old way

string FullName => $"{FirstName} {LastName}"; //Interpolation
```

Variable and Constants

```
int     i;                  // variable

const int i = 5;            // constant

readonly int    i; /* variable within
constructor, elsewhere constant */
```

Class and Instance Methods

```
public class Person
{
    private static int classVal;     // Number of people ever existed
    private int instValAge;             // Used to hold person‟s age

    public Person()                  // Instance Constructor
    {
        ++classVal;                  // Another person has existed
    }

    public static int People()       // A class method because made „static‟
    {
        return classVal;
    }
```

```
. . .

Person Fred = new Person();
Console.WriteLine(Person.People());


Person Jim = new Person();
Console.WriteLine(Person.People());

. . . .
```

Class and Instance Methods

```
public class Person
{
    private static int classVal;    // Number of people ever existed
    private int instValAge;             // Used to hold person"s age

    public Person()                 // Instance Constructor
    {
        ++classVal;                 // Add another person to the count
    }

    public static int People()      // Class method due to "static"
    {
        return classVal;
    }



    public void SetAge(int age)     // Instance method
    {
        instValAge = age;
    }



    public int GetAge()    // Instance method
    {
        return instValAge;
    }
}
```

```
. . .


Person Ford = new Person();

Ford.SetAge(25);

Console.WriteLine("Ford is {0}",  Fred.GetAge());

. . .
```

The System.Object Class

- Basis for all other objects

- System.Object (object)
  - And therefore every object supports:
    - Equals()
    - GetHashCode()
    - GetType()
    - ToString()

C# Keywords

abstract	do	implicit	private	this
as	double	in	protected	throw
base	else	int	public	true
bool	enum	interface	readonly	try
break	event	internal	ref	typeof
byte	explicit	is	return	uint
case	extern	lock	sbyte	ulong
catch	false	long	sealed	unchecked
char	finally	namespace	set	unsafe
checked	fixed	new	short	ushort
class	float	null	sizeof	using
const	for	object	stackalloc	value
continue	foreach	operator	static	virtual
decimal	get	out	string	void
default	goto	override	struct	@
delegate	if	params	switch	