



# Introduction to C# procedural programming



# C# Program Structure

- Object oriented
- Everything belongs to a class
  - no global scope
- “Classical” complete C# program – the statements have to be within class, and method `Main()` is executed at startup
- “New” complete C# program – there can be a top global level statements, which are executed at the start

# C# Program Structure

- Namespaces
  - Contain types and other namespaces
- Type declarations
  - Classes, structs, interfaces, enums, and delegates
- Members
  - Constants, fields, methods, properties, events, operators, constructors, destructors
- Organization
  - No header files, code written “in-line”

# C# Program Structure

- “Classical” complete C# program:

```
namespace ConsoleTest
{
    class Class1
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello World!");
        }
    }
}
```

- “New” complete C# program:

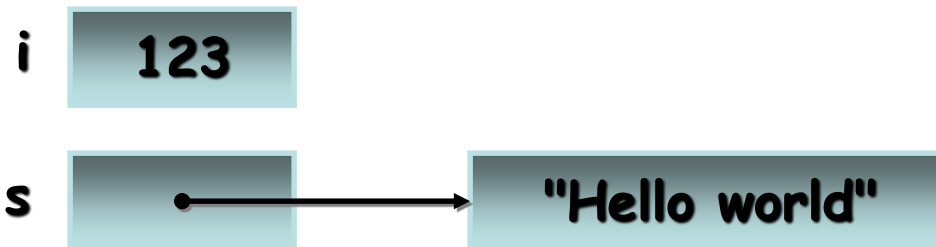
```
System.Console.WriteLine("Hello World!");
```

# Value and Reference Types

## Definition

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

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



# Value and Reference Types

## Example

- Value types

- Primitives
- Enums
- Structs

```
int i;
```

```
enum State { Off, On }
```

```
struct Point { int x, y; }
```

- Reference types

- Classes
- Interfaces
- Arrays
- Delegates

```
class Foo: Bar, IFoo {...}
```

```
interface IFoo: IBar {...}
```

```
string[] a = new string[10];
```

```
delegate void Empty();
```

# Simple Types

- Integer Types
  - **byte**, **sbyte** (8bit), **short**, **ushort** (16bit)
  - **int**, **uint** (32bit), **long**, **ulong** (64bit)
- Floating Point Types
  - **float** (precision of 7 digits)
  - **double** (precision of 15–16 digits)
- Exact Numeric Type
  - **decimal** (28 significant digits)
- Character Types
  - **char** (single character)
  - **string** (rich functionality, by-reference type)
- Boolean Type
  - **bool** (distinct type, **not** interchangeable with **int**)

# Statements and Comments

- Case sensitive (myVar != MyVar)
- Statement delimiter is semicolon ;
- Block delimiter is curly brackets { }
- Single line comment is //
- Block comment is /\* \*/
  - Save block comments for debugging!



# Data

- All data types derived from ***System.Object***
- Declarations:
  - datatype varname;*
  - datatype varname = initvalue;*
- C# does not automatically initialize local variables (but will warn you)!

# Value Data Types

- Directly contain their data:
  - int (numbers)
  - long (really big numbers)
  - bool (true or false)
  - char (unicode characters)
  - float (7-digit floating point numbers)
  - string (multiple characters together)

# Expressions

- Expressions are build of constants, variables and operators

# Data Manipulation

=	assignment
+	addition
-	subtraction
*	multiplication
/	division
%	modulus
++	increment by one
--	decrement by one

# Conditional Operators

== equals

!= not equals

< less than

<= less than or equal

> greater than

>= greater than or equal

&& and

|| or

# Branches and loops

- Expressions are build of constants, variables and operators

# If, Case Statements

```
if (expression)  
    { statements; }  
else if  
    { statements; }  
else  
    { statements; }
```

```
switch (i) {  
    case 1:  
        statements;  
        break;  
    case 2:  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```

# Loops

```
for (initialize-statement; condition; increment-statement);  
{  
    statements;  
}  
  
while (condition)  
{  
    statements;  
}
```

Note: can include *break* and *continue* statements



# strings

- Immutable sequence of Unicode characters (char)
- Creation:
  - `string s = "Bob";`
  - `string s = new String("Bob");`
- Backslash is an escape:
  - Newline: `"\n"`
  - Tab: `"\t"`

# string/int conversions

- string to numbers:
  - `int i = int.Parse("12345");`
  - `float f = float.Parse("123.45");`
- Numbers to strings:
  - `string msg = "Your number is " + 123;`
  - `string msg = "It costs " +  
string.Format("{0:C}", 1.23);`

# Arrays

- (page 21 of quickstart handout)
- Derived from `System.Array`
- Use square brackets `[]`
- Zero-based
- Static size
- Initialization:
  - `int [] nums;`
  - `int [] nums = new int[3]; // 3 items`
  - `int [] nums = new int[] {10, 20, 30};`

# Arrays

- Built on .NET **System.Array** class
- Declared with type and shape, but no bounds
  - `int [ ] SingleDim;`
  - `int [ , ] TwoDim;`
  - `int [ ][ ] Jagged;`
- Created using **new** with bounds or initializers
  - `SingleDim = new int[20];`
  - `TwoDim = new int[,]{ {1,2,3}, {4,5,6} };`
  - `Jagged = new int[1][ ];`  
`Jagged[0] = new int[ ]{1,2,3};`

# Arrays

- Multidimensional

// 3 rows, 2 columns

```
int [ , ] myMultiIntArray = new int[3,2]
```

```
for(int r=0; r<3; r++)
```

```
{
```

```
    myMultiIntArray[r][0] = 0;
```

```
    myMultiIntArray[r][1] = 0;
```

```
}
```

# Summary

- C# builds on the .NET Framework component model
- New language with familiar structure
  - Easy to adopt for developers of C, C++, Java, and Visual Basic applications
- Fully object oriented
- Optimized for the .NET Framework

Thanks for your attention!

