Eng.wael Hosny

Object Oriented Programming using C#

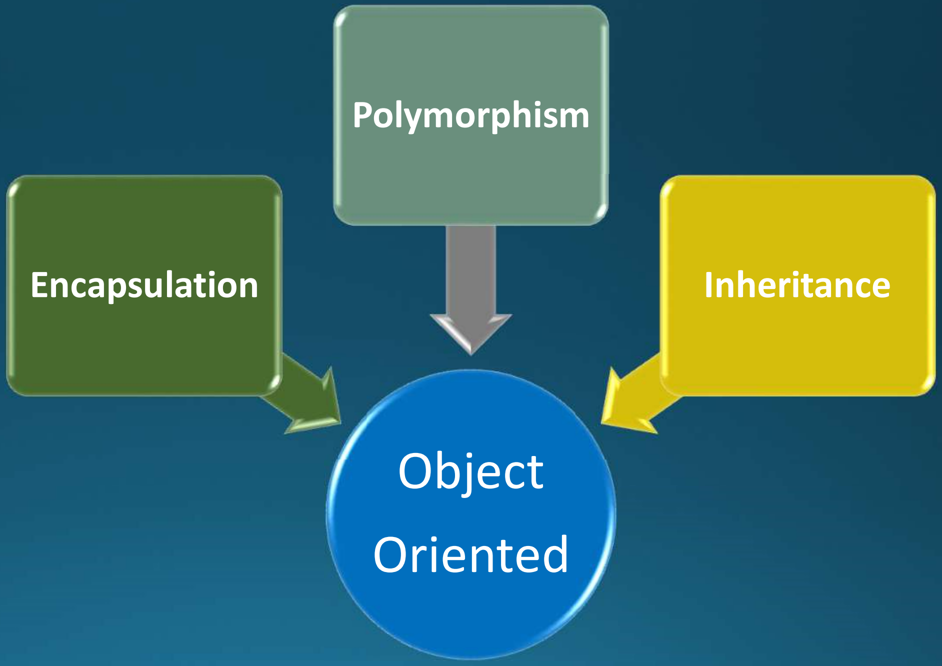
Introduction to programming

- What is programming
  - Programming is a way to “instruct the computer to perform various tasks”.
- Programming Techniques
  - Structural (modular) Programming (ex : C , Basic , Fortran)
  - Object Oriented Programming (ex C++ , Java , C#)

Introduction to programming

- Object Oriented
  - Encapsulation
  - Polymorphism
  - Inheritance

  - Abstraction

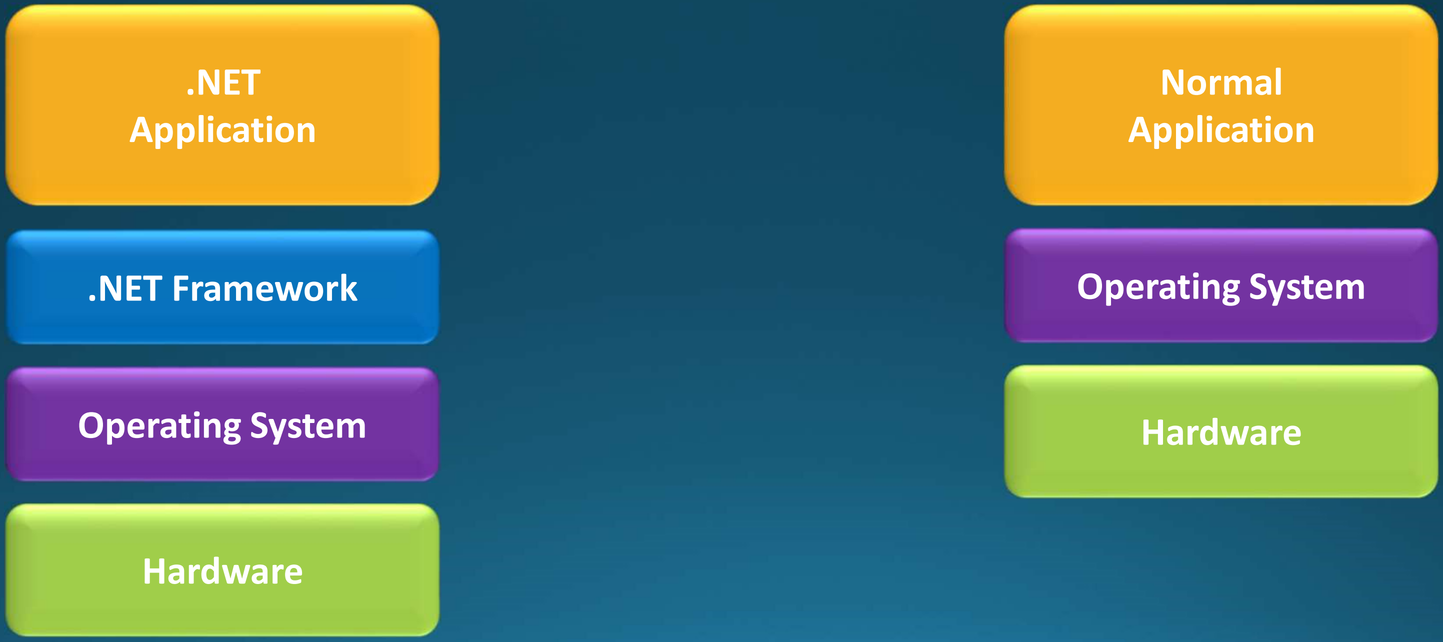


Object oriented using C# language

- .NET framework

.NET Framework

- What is .NET Framework



Overview of .NET Framework

- .NET Framework
  - .NET Framework applications
  - .NET Languages
  - .NET Framework Components

.NET Framework application

- Console applications
- Windows forms
- ASP.NET applications
- Web services
- Windows services
- SQL Server applications
- Small device applications (mobile application)

.NET Languages

- C#.NET
- V.B .NET
- C++ .NET
- J# .NET

.NET Framework Components

- CLR (Common Language Runtime)
- Class library (Resides as DLL on the H/D)

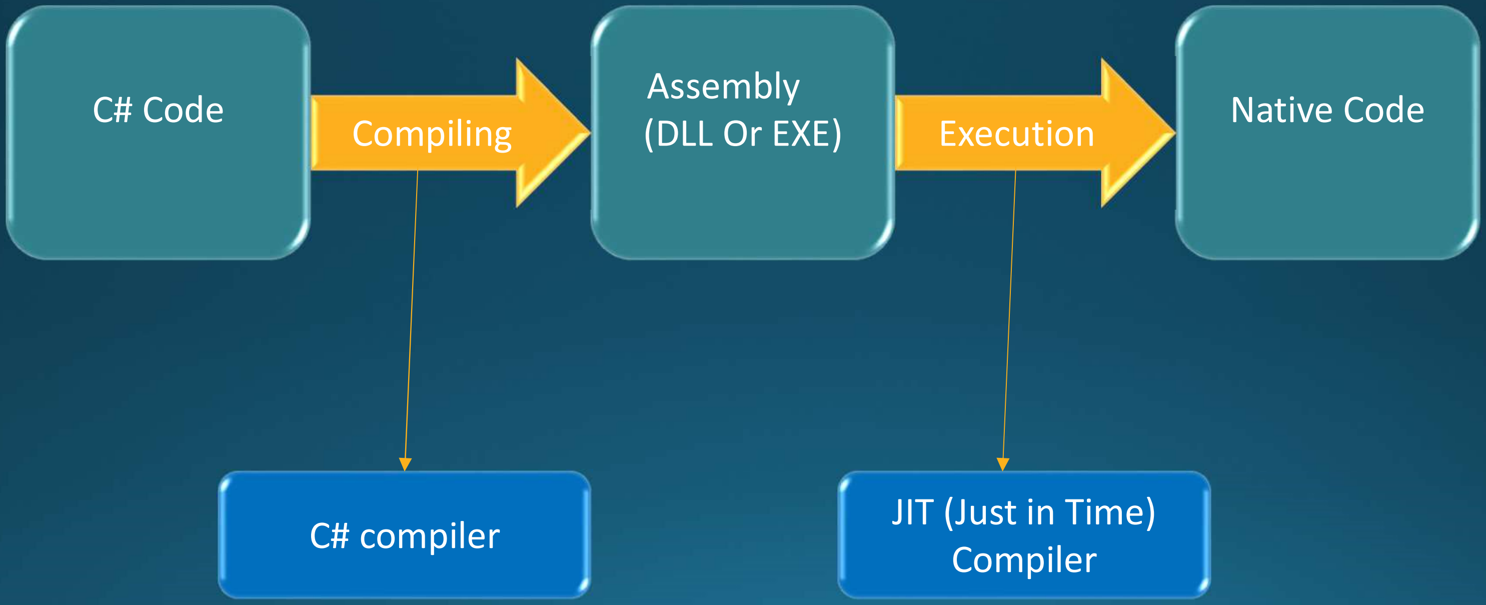
Common Language Runtime (CLR)

- Responsible
  - Executing application
  - Memory Management
  - Security enforcement
  - Language Integration
  - Thread Execution
- Managed Code
- Unmanaged Code

Common Language Runtime (CLR) cont.

- **CTS** (Common Type System)
  - It describe a set of data types that can be used in different .NET languages (C#, vb.NET,…)
  - Ex in C# : data type ***int***   in vb.net data type ***integer*** =>BCL ***Int32***
  - It support 2 types categories
    - Reference type
    - Value type
- **CLS** (Common Language Specification)
  - A set of rules and specification that any .NET language must comply to Run under .NET and so to achieve language integration
- **Garbage Collector**
  - Which responsible for ensure of clearing the memory after the application exit

Application Life Cycle



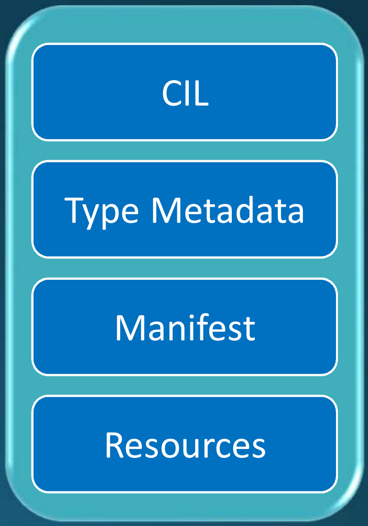
Assembly

generated from the first compiling phase which contain

- CIL (Common *Intermediate Language*) Now
  - Instruction not specific to certain processor
- Type Metadata
  - Data about the datatypes within the assembly (name , access levels,….)
- Manifest (Assembly Metadata)

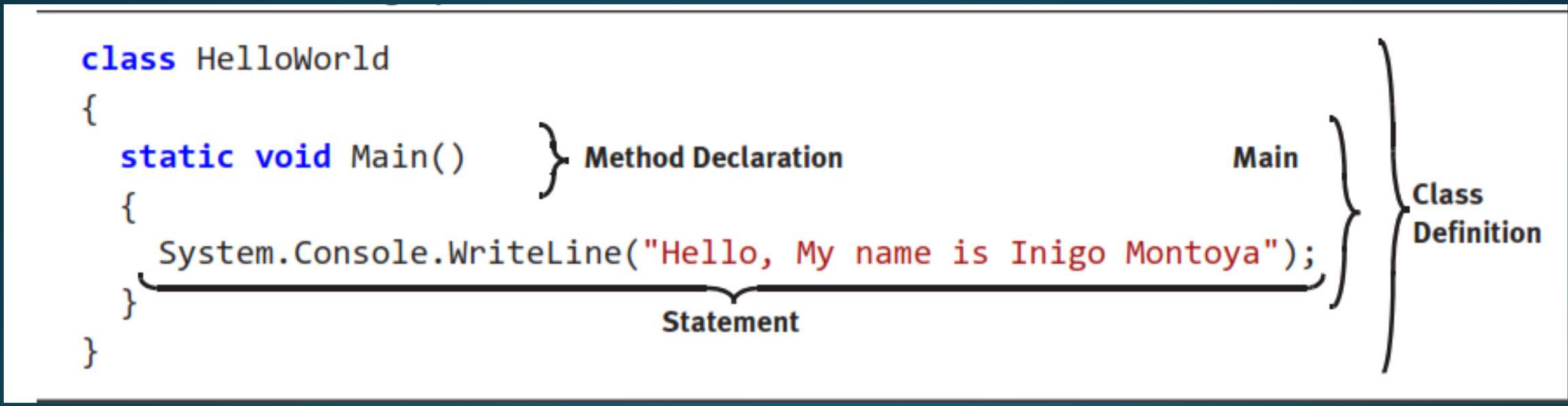
  Which contain the metadata describes
  - Version of the assembly
  - Security Information
  - External assemblies references
  - Exported types
- Resources

- ildasm .exe
- Dotnet peek



Assembly
(EXE or DLL)

First Program



```
class HelloWorld
{
  static void Main()
  {
   System.Console.WriteLine("Hello, My name is Inigo Montoya");
  }
}
```

First Program

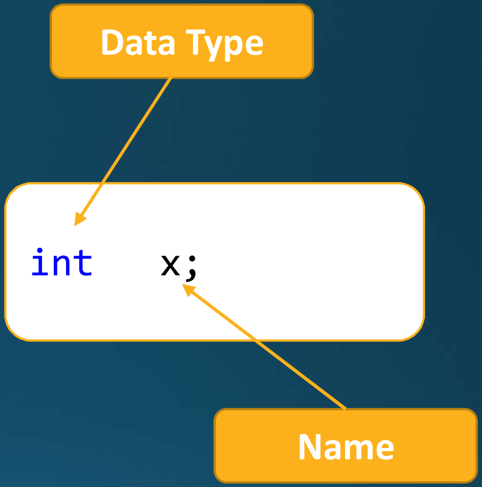
- Console Application
  - Main method
    - Static modifier
  - Intro to IDE
    - Solution, Project
    - Solution explorer(References) ,toolbox, breakpoint , watch, run, run without debug ,build
    - Help(F1)

Console Application

- Console Methods
  - Console.Write, Console.WriteLine
  - Console.Read, Console.ReadLine, Console.Readkey
  - Console.ResetColor
  - Console.clear
  - SetCursorPosition
- Properties
  - Console.BackgroundColor
  - Console.ForegroundColor

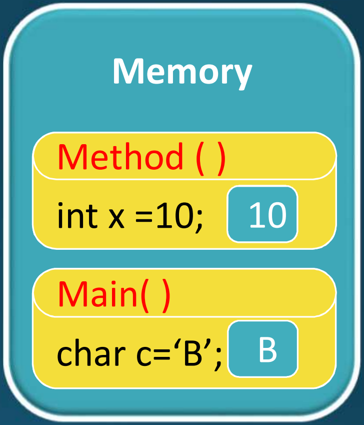
Variables and Data Types

- What is variable ?
  - A place in memory that has
    - Name
    - Size (number of bytes)
  - Declare variable (type, name)
  - Variable name
    - Must begin with a letter
    - Can’t be a digit
    - Can’t contain space or symbol like ? , / , - ,*, @
    - Can start or contain _



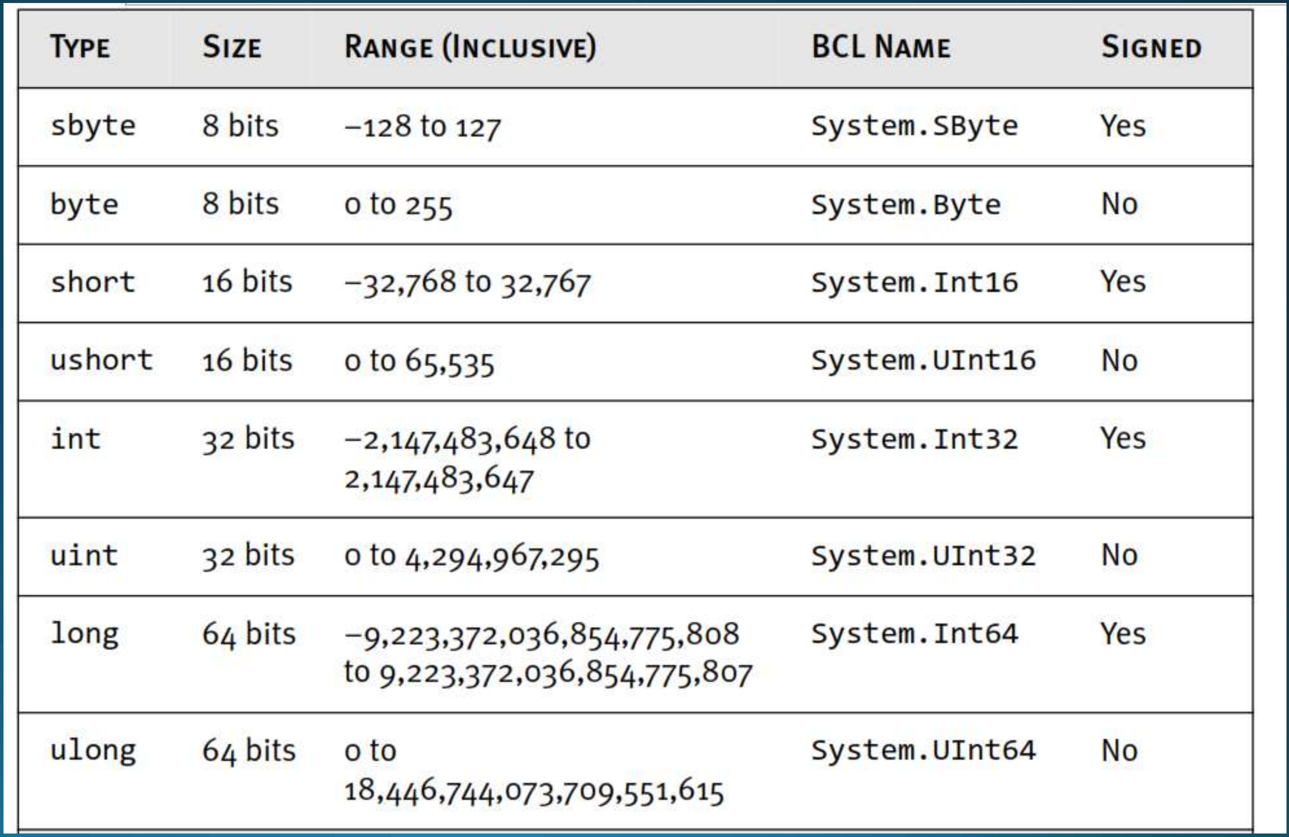
Data Types cont.

- Types of Data
  - Value Types
    - The variable contain the data
  - Reference Types



Value Types

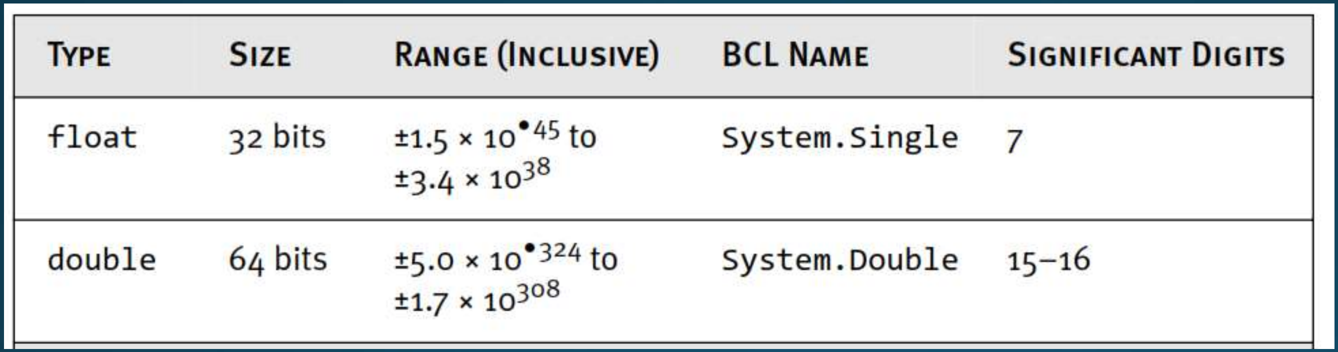
- Integer



Type	Size	Range (Inclusive)	BCL Name	Signed
`sbyte`	8 bits	−128 to 127	`System.SByte`	Yes
`byte`	8 bits	0 to 255	`System.Byte`	No
`short`	16 bits	−32,768 to 32,767	`System.Int16`	Yes
`ushort`	16 bits	0 to 65,535	`System.UInt16`	No
`int`	32 bits	−2,147,483,648 to 2,147,483,647	`System.Int32`	Yes
`uint`	32 bits	0 to 4,294,967,295	`System.UInt32`	No
`long`	64 bits	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	`System.Int64`	Yes
`ulong`	64 bits	0 to 18,446,744,073,709,551,615	`System.UInt64`	No

Value Types

- Floating point



TYPE	SIZE	RANGE (INCLUSIVE)	BCL NAME	SIGNIFICANT DIGITS
float	32 bits	$\pm1.5 \times 10^{\bullet 45}$ to $\pm3.4 \times 10^{38}$	System.Single	7
double	64 bits	$\pm5.0 \times 10^{\bullet 324}$ to $\pm1.7 \times 10^{308}$	System.Double	15–16

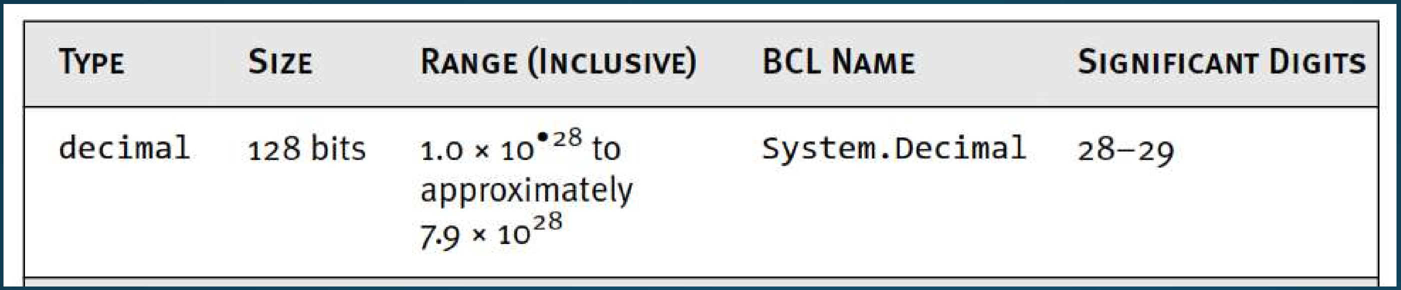
  - Literal Error

```
float f = 10.0;  // error  to correct it   float f=10.0f;
```

- Boolean
  - true , false

Value Types

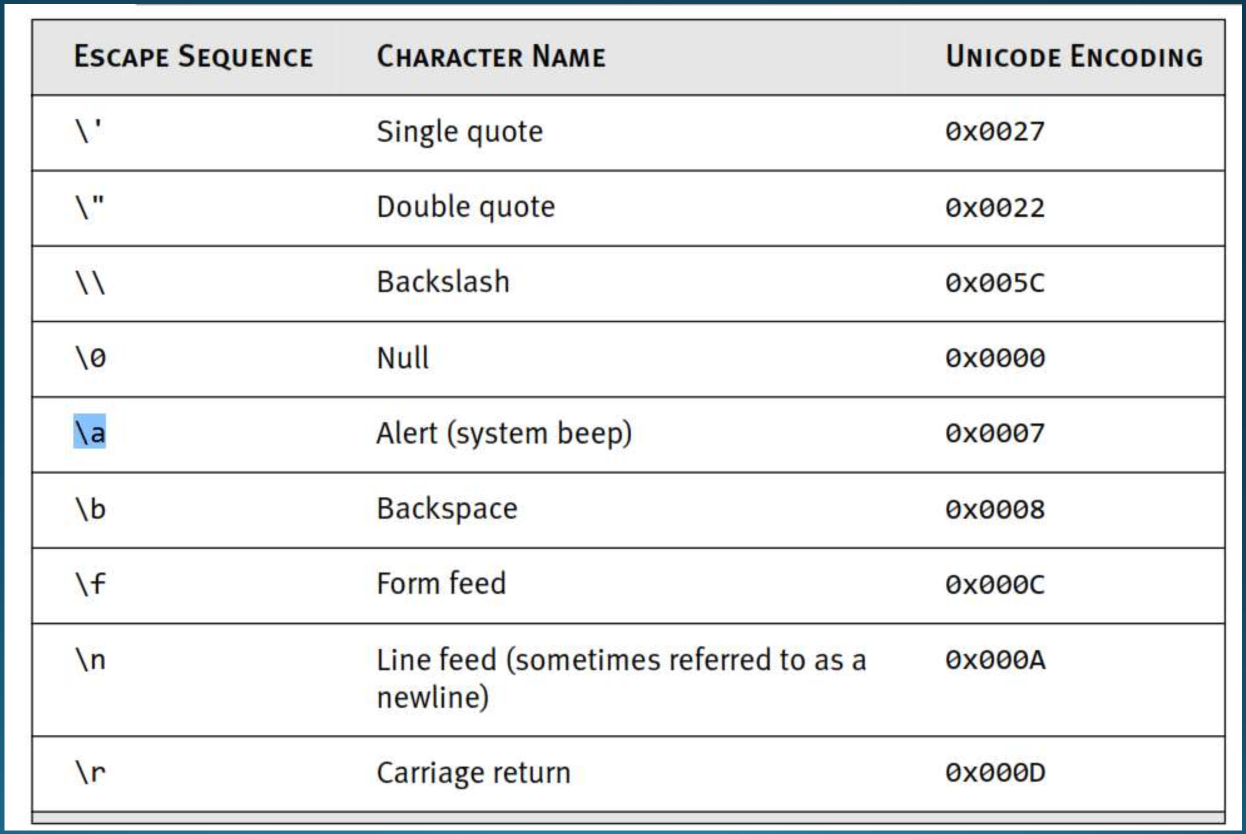
- Decimal



Type	Size	Range (Inclusive)	BCL Name	Significant Digits
`decimal`	128 bits	$1.0 \times 10^{\bullet 28}$ to approximately 7.9×10^{28}	`System.Decimal`	28–29

Value Types

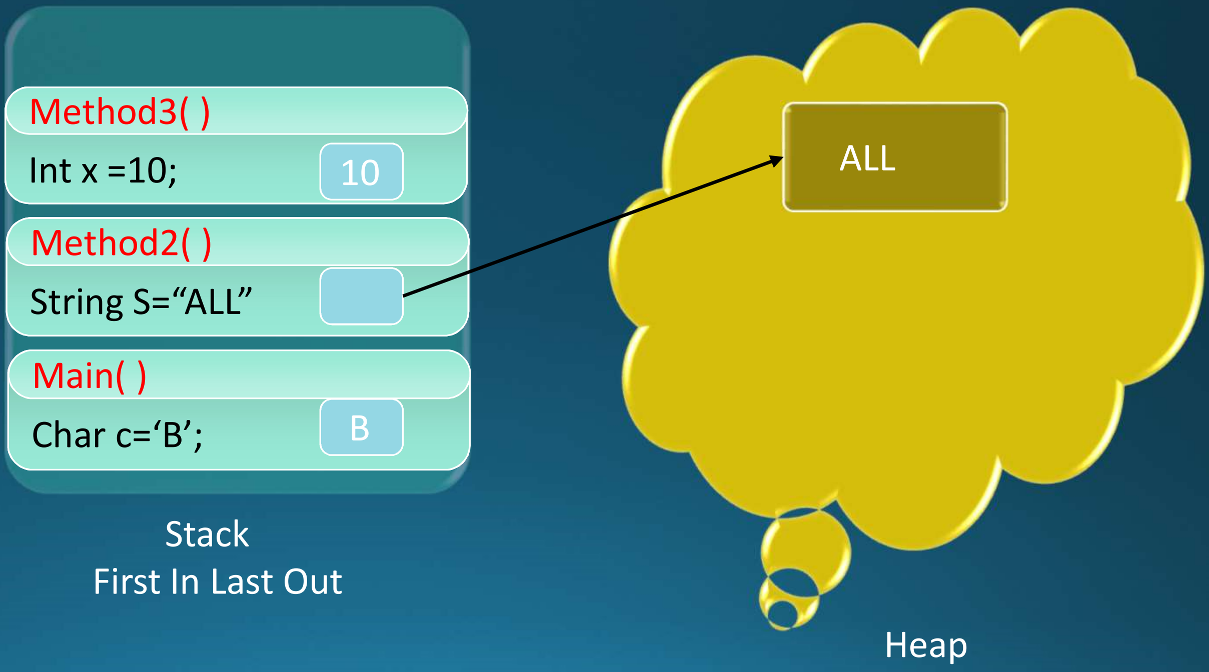
- Character



Escape Sequence	Character Name	Unicode Encoding
`\'`	Single quote	0x0027
`\"`	Double quote	0x0022
`\\`	Backslash	0x005C
`\0`	Null	0x0000
`\a`	Alert (system beep)	0x0007
`\b`	Backspace	0x0008
`\f`	Form feed	0x000C
`\n`	Line feed (sometimes referred to as a newline)	0x000A
`\r`	Carriage return	0x000D

Reference Types

- Value Type vs. Reference Type
  - Stack vs. Heap
- Arrays
- String
- Classes



String

- String Methods
  - Static
    - Format (Full Name Example)
    - Concat (Full Name Example)
    - Compare two versions
  - Instance method
    - StratWith
    - EndWith
    - ToLower
    - ToUpper
    - Trim
    - Replace
    - TocharArray()
    - PadLeft() PadRight()
- String is immutable
  - StringBuilder

Nullable Modifier

- Assign null to value types
  - Ex :

```
int? x = null;
```

  - *Useful for database programming*

Conversions between Data Types

- Implicit Cast

```
float f1 = 10.0f;
double d = f;
```

- Explicit Cast

```
double d = 10.5;
float f1 =(float) d;
```

- Conversion without cast
  - int.parse (String)
  - bool int.Tryparse(String , out int)
  - ToString ()
- Boxing and Unboxing
  - Object class
- as operator

Assignment

- Install Visual Studio
- First program
- Get sum , average for 2 numbers
  - Watch ,breakpoint debug
- Email validation
- calculator

Operators

- C# supports:
  - Unary operators:
    - Requires one operand such as x++
  - Binary operators:
    - Require two operands in the expression such as x+2
  - Ternary operators:
    - Requires three operands such as Conditional (?   :) operator.

Operators (Cont.)

- Arithmetic Operators:

Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6
--	Decrement	x=--y	x=4

- + Operators with string
- Characters in arithmetic operators (unicode)
  - Result must be taken in int variable

```
string s1 = "Ahmed";
string s2 = "ali";
string s3 = s1 + s2;
```

```
char c1 = 'A';
char c2 = 'b';
char c3=c1 + c2; //error → int c3=c1+c2;
```

Operators (Cont.)

- Assignment Operators:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

wr1

Operators (Con

- Bitwise Operators:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise Left Shift
>>	Bitwise Right Shift

```
int a = 60;/* 60 = 0011 1100 */
int b = 13;/* 13 = 0000 1101 */
int c = 0;

c = a & b; /* 12 = 0000 1100 */
Console.WriteLine("Line 1 - Value of c is {0}", c);

c = a | b; /* 61 = 0011 1101 */
Console.WriteLine("Line 2 - Value of c is {0}", c);

c = a ^ b; /* 49 = 0011 0001 */
Console.WriteLine("Line 3 - Value of c is {0}", c);

c = ~a;    /*-61 = 1100 0011 */
Console.WriteLine("Line 4 - Value of c is {0}", c);

c = a << 2;/* 240 = 1111 0000 */
Console.WriteLine("Line 5 - Value of c is {0}", c);

c = a >> 2;/* 15 = 0000 1111 */
Console.WriteLine("Line 6 - Value of c is {0}", c);
Console.ReadLine();
```

wr1 using System; namespace OperatorsAppl { class Program { static void Main(string[] args) { int a = 60; /* 60 = 0011 1100 */ int b = 13; /* 13 = 0000 1101 */ int c = 0; c = a & b; /* 12 = 0000 1100 */ Console.WriteLine("Line 1 - Value of c is {0}", c); c = a | b; /* 61 = 0011 1101 */ Console.WriteLine("Line 2 - Value of c is {0}", c); c = a ^ b; /* 49 = 0011 0001 */ Console.WriteLine("Line 3 - Value of c is {0}", c); c = ~a; /*-61 = 1100 0011 */ Console.WriteLine("Line 4 - Value of c is {0}", c); c = a << 2; /* 240 = 1111 0000 */ Console.WriteLine("Line 5 - Value of c is {0}", c); c = a >> 2; /* 15 = 0000 1111 */ Console.WriteLine("Line 6 - Value of c is {0}", c); Console.ReadLine(); } } }

wael radwan, 4/22/2019

Operators (Cont.)

- Comparison Operators:

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equality
!=	Inequality

Operators (Cont.)

- Logical Operators:

Operator	Description
&&	Logical "AND" – returns true when both operands are true; otherwise it returns false
||	Logical "OR" – returns true if either operand is true. It only returns false when both operands are false
!	Logical "NOT"—returns true if the operand is false and false if the operand is true. This is a unary operator and precedes the operand

Operators Precedence

- Operator precedence: Determines the order in which operators are evaluated. Operators with higher precedence are evaluated first.
- Operator Associativity: Determines the order in which operators of the same precedence are processed.
- The operators that you have learned are evaluated in the following order (from highest precedence to lowest):
  - Parentheses(())
  - Multiply/divide/modulus (*, /, %)
  - Addition/Subtraction (+, -)
  - Comparison (<, <=, >=, >)
  - Equality (==, !=)
  - Logical and (&&)
  - Logical or (||)
  - Conditional (?:)
  - Assignment operators (=, +=, -=, *=, /=, %=)
- Example:
  - 5 + 3 * 2 = 11 ⍰ 3*2=6 , then 6+5 = 11.
  - BUT (5 + 3) * 2 = 16   5+3 = 8 , then 8*2 = 16.

Controlling Program Flow

- Control Statements that can be used are:
  - Conditional Statements
    - if ….else
    - switch/case
  - Loop Statements
    - for
    - while
    - do…while
    - foreach

Controlling Statements (Cont.)

- Conditional Statements

if else

```
if (condition)
{
  do something;
}
else
{
 if (Condition)
 {
   do somethingelse;
 }
else
 {
  do somethingelse;
 }
}
```

Switch / case

```
switch (expression)
{
case value1:
    statements
    break;

case value2:
    statements
    break;
default :
     statements
}
```

Controlling Statements (Cont.)

- Loop Statements

for

```
for(int i=0;i<3;i++)
{
    …
}
```

while

```
while(x<5)
{
    …
}
```

do while

```
do
{
    …
} while (x < 6);
```

foreach

```
foreach(int i in arr) // arr is array of int
{
    …. // i is used for read only

}
```

Controlling Statements (Cont.)

- Breaking Loops :
  - break statement : The break statement will break the loop and continue executing the code that follows after the loop (if any).
  - continue statement: The continue statement will break the current loop and continue with the next value.