Languages and Computer Systems Department

Unit 7 .net Platform
First programs using C#

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Escuela Politécnica Superior
Universidad de Alicante

Programming in C#. First programs using C#

Objectives

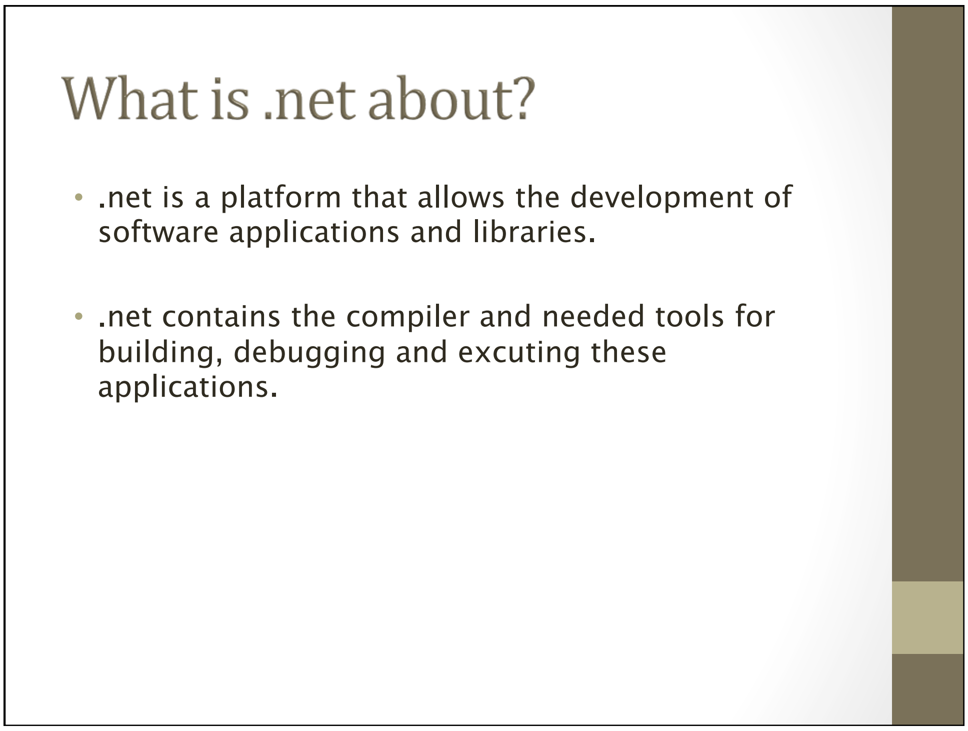
1. Introduction to the .NET platform
2. Knowing the C# origins
3. Creating a console-based application
4. Knowing the basic structure of a program in C#
5. Knowing the exception management
6. Knowing the Ilist data collection

INDEX

Introduction to .Net

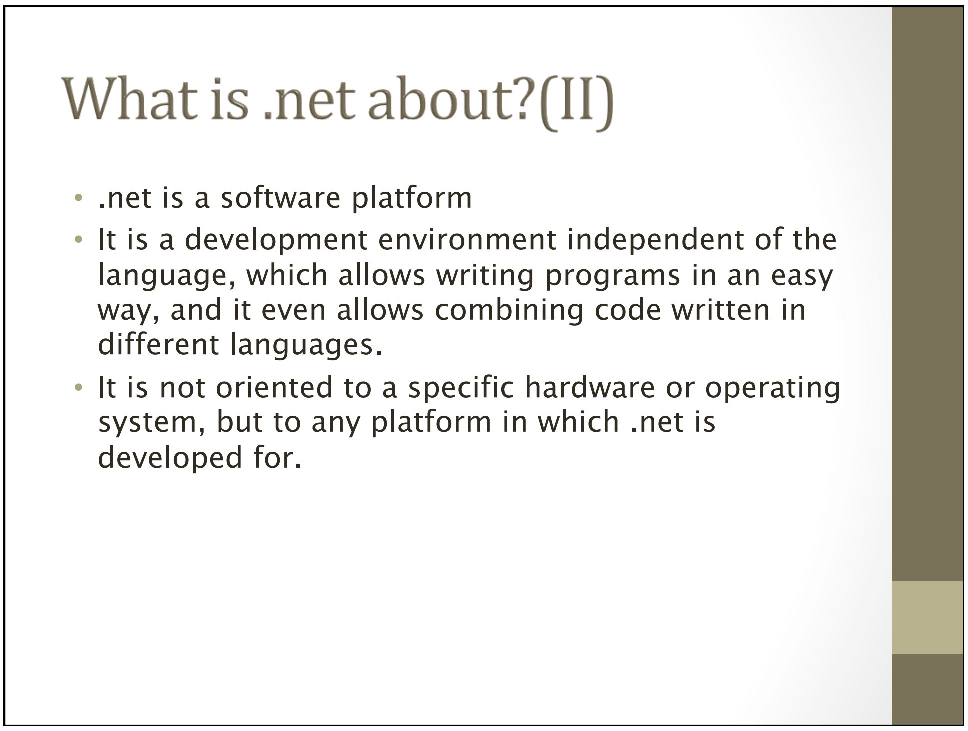
1

What .Net is about?



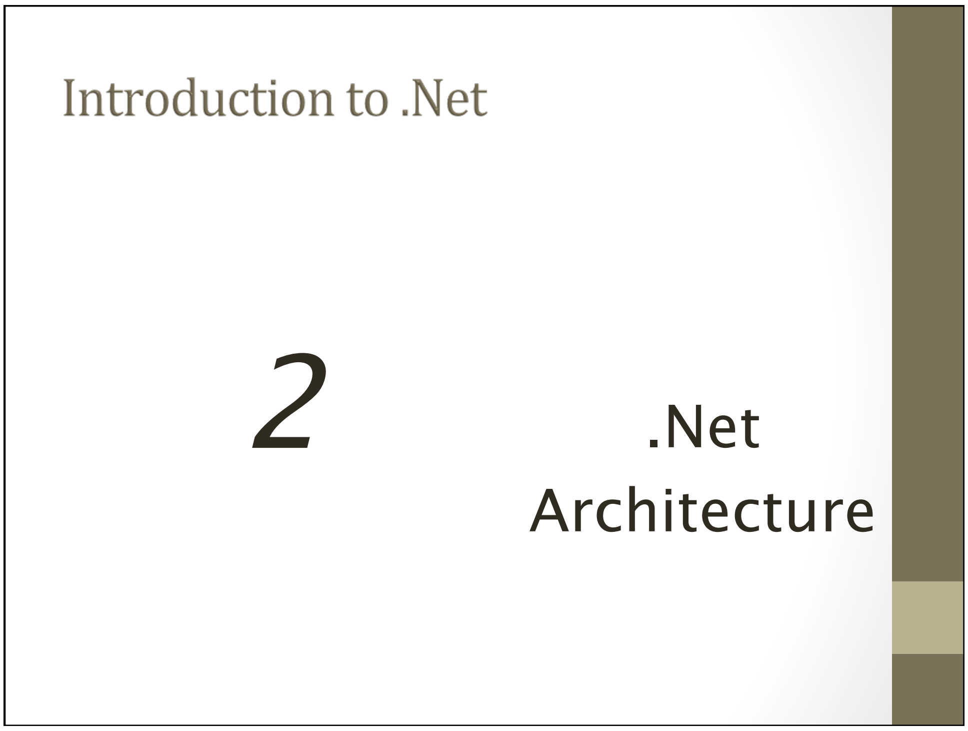
What is .net about?

- .net is a platform that allows the development of software applications and libraries.
- .net contains the compiler and needed tools for building, debugging and excuting these applications.



What is .net about?(II)

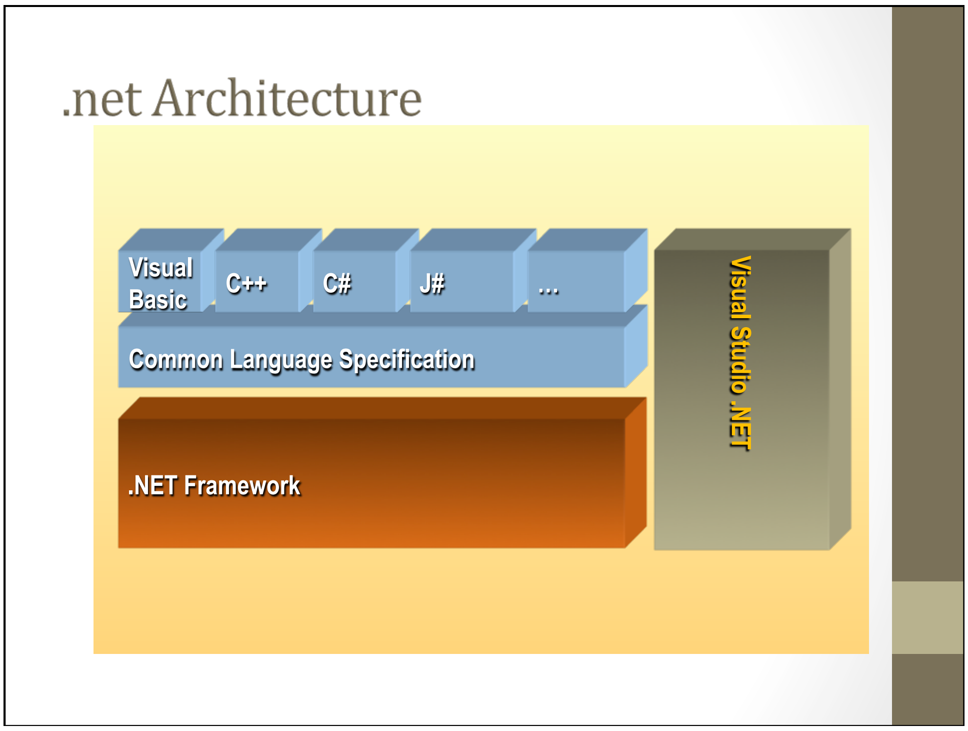
- .net is a software platform
- It is a development environment independent of the language, which allows writing programs in an easy way, and it even allows combining code written in different languages.
- It is not oriented to a specific hardware or operating system, but to any platform in which .net is developed for.



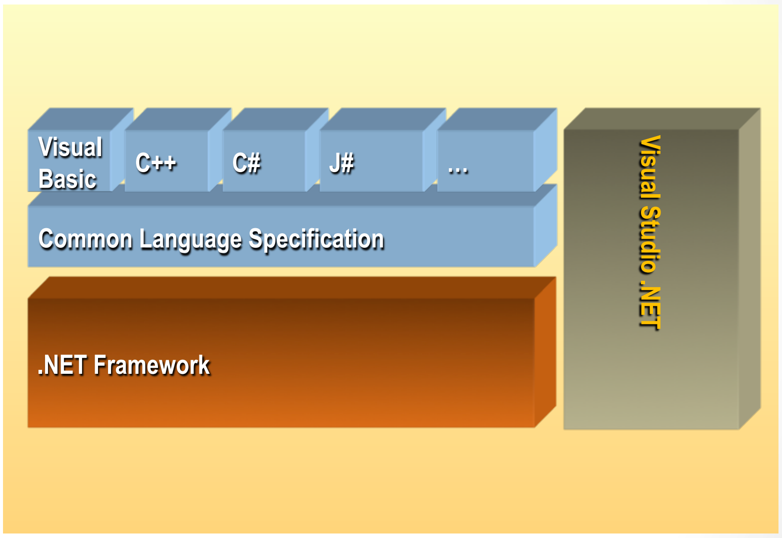
Introduction to .Net

2

.Net Architecture



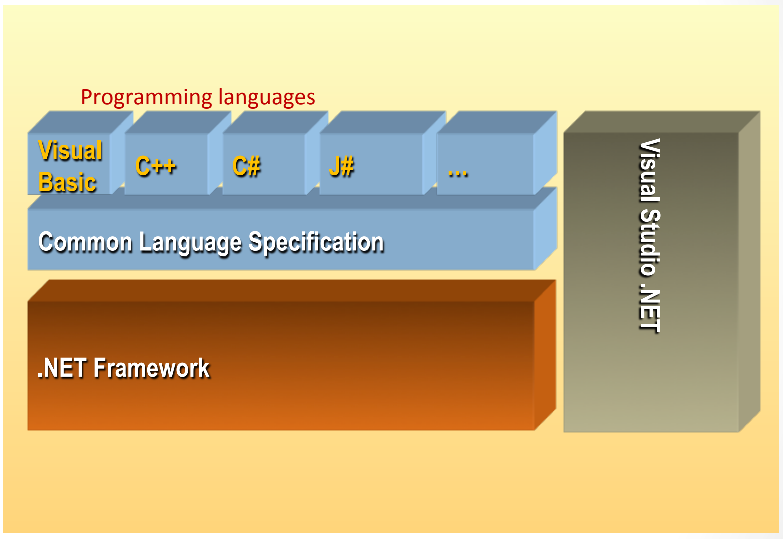
.net Architecture



Visual Studio .net

- Visual Studio .NET offers a development environment for developing applications to be run on the .NET Framework.
- It provides the fundamental technologies for simplifying the creation of:
  - Web applications and Web services
  - Windows-based applications

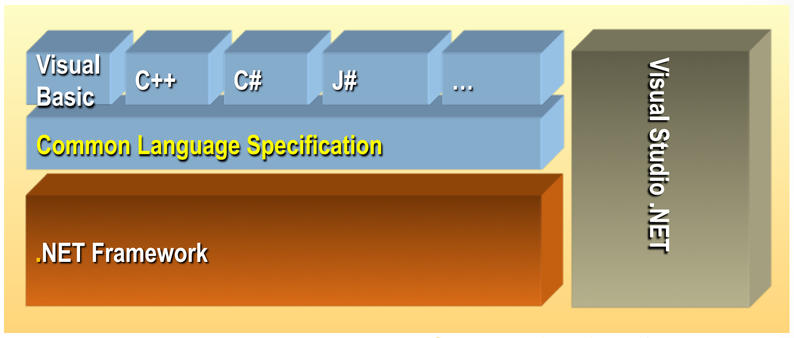
.net Architecture



Programming languages

- C#
  - C# has been specifically designed for the .NET platform and it is the first modern language component oriented from the family of C and C++–
  - It can be incrusted into ASP.net pages.
  - Some of the main features of this language include classes, interfaces, delegates, namespaces, properties, indexes, events, override of operators, versionning, attributes, unsecure code, creation of documentation in XML format…
- Visual Basic .net
- C++
- J#
- Other languages (phyton, etc)

CLS

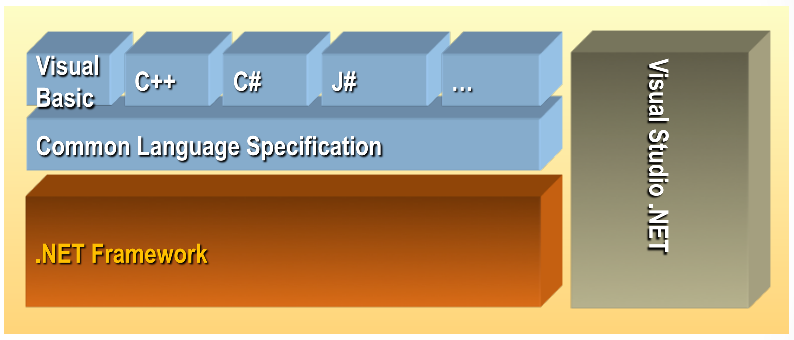


- The Common Language Specification (CLS) defines the minimal standards that have to be satisfied by the languages and the programmers if they want that their components and applications are widely used by other languages compatible with .NET.

Common Language Specification (II)

- The CLS specification allows the .NET developers creating applications as a team using multiple languages being sure that there will be no problems with the integration of those languages.
- The CLS specificacion also allows to the .NET developers inherit from classes developed in different languages.

.net framework



- It is the execution engine
- It provides a set of common services for the projects generated in .net, independently of the language used.

.net Framework (II)

- **Extensible**
  - The hierarchy of the .NET framework is not hidden to the developer. We can access and extend the .NET classs (excepting the ones that are sealed) by using inheritance. We can also implement multilanguage inheritance.
- **Easy to use by the developers**
  - In the .NET framework, the code is organized by hierarchical namespaces and classes.
  - The framework provides a common type system called unified type system, that is used by any language compatible with .NET. In the unified type system everything is an object.

Introduction to .Net

3

.net Framework Architecture

Parts of the .NET Framework.

The **Common Language Runtime** (**CLR**) is the nutshell of the .NET platform. This is the engine that has to manage the execution of the applications developed and it provides to them several services to simplify its development increasing their security and reliability.

Common Language Runtime (CLR)

- **Multiplatform execution**: The CLR acts as a virtual machine, having to run the applications designed for the .NET platform. This means, each platform for which exists a new version of the CLR can execute any .NET application.

By now there are developed CLR for all versions of Windows, but exists the possibility of developing a version for systems such Unix or Linux because the CLR architecture is open.

Mono Project: http://go-mono.com

mono::

Common Language Runtime (CLR)

- **Integration of languages:**

We can use code generated for the .NET framework from any .NET language using any other .NET language.

The integration of languages allows writing a class using C# that inherits from another written using Visual Basic.NET which, can also inherit from another class written in C++ *with managed extensions*.

Microsoft Intermediate Language (MSIL)

- The compilers that generate code for the .NET platform generate code written in the intermediate language known as Microsoft Intermediate Lenguage (MSIL)

Common Language Runtime (CLR)

- **Memory management:**

The CLR includes a **garbage collector** which avoids that the programmer has to know when destroying the objects that are unuseful.

Thanks to this collector we avoid common programming errors such as:
• Attempts of deletion of objects already deleted.
• Exhaustion of the memory caused by forgetting the deletion of unuseful objects or
• Request for accesing members of objects already deleted.

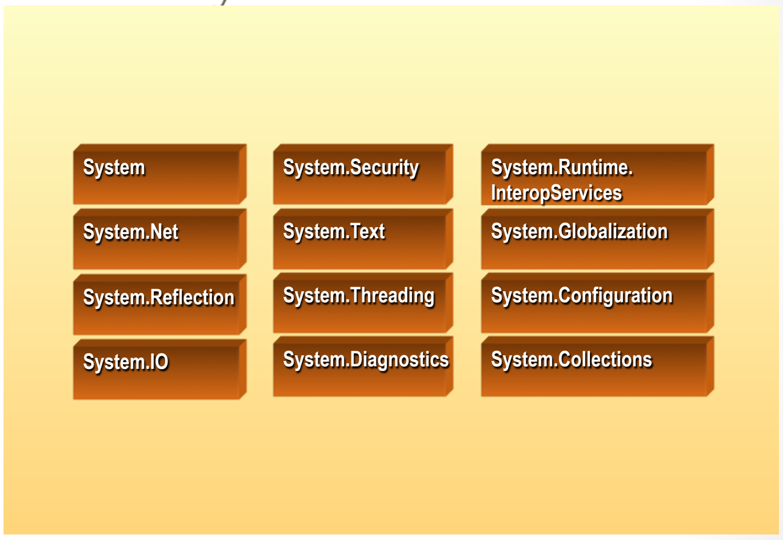
Common Language Runtime (CLR)

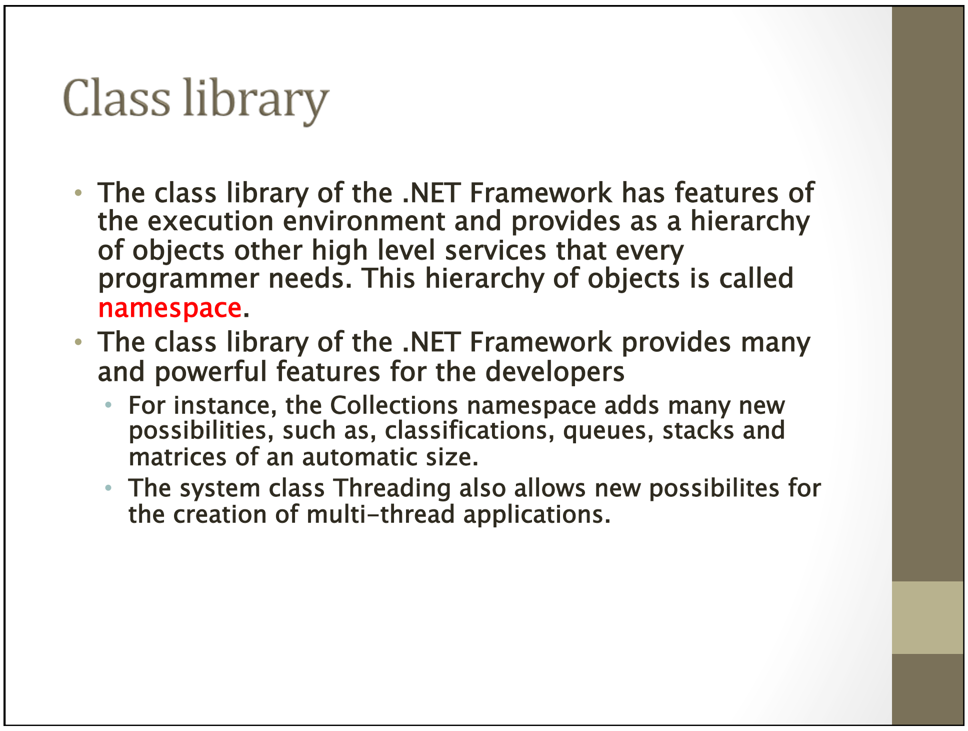
- **Security of types:** The CLR facilitates the detection of programming errors difficult to be detected by checking that all the type conversion done during the execution of a .NET application is done in a way that the source and target types are compatible.

Common Language Runtime (CLR)

- **Management of exceptions:** In the CLR all the errors that can be generated during the execution of an application are propagated in the same way: using *exceptions*.

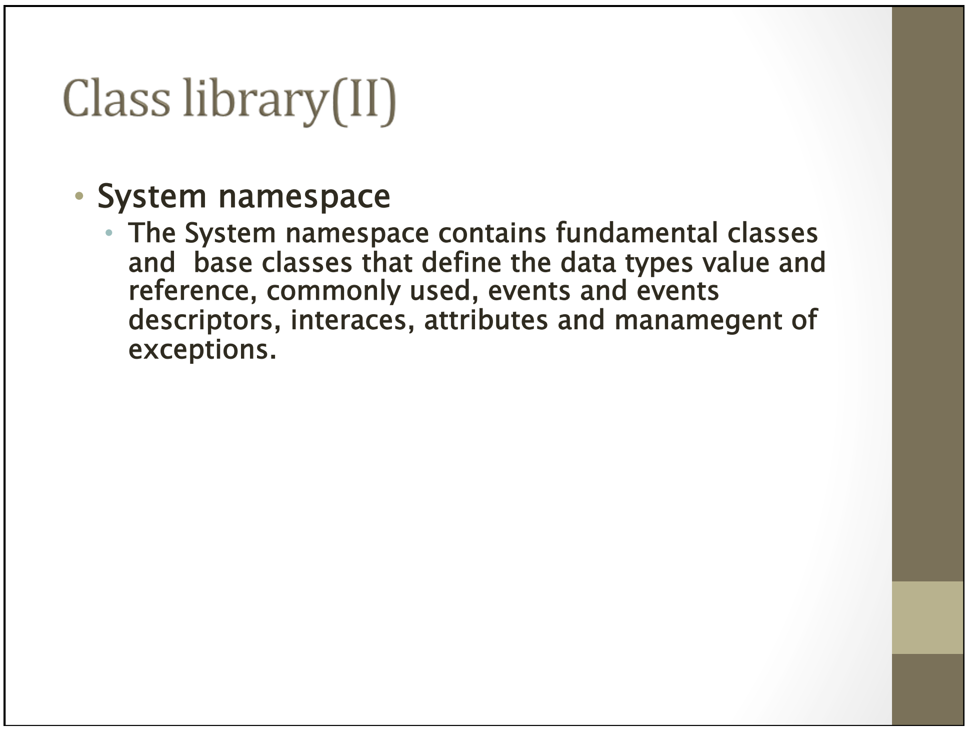
Class library





Class library

- The class library of the .NET Framework has features of the execution environment and provides as a hierarchy of objects other high level services that every programmer needs. This hierarchy of objects is called namespace.
- The class library of the .NET Framework provides many and powerful features for the developers
  - For instance, the Collections namespace adds many new possibilities, such as, classifications, queues, stacks and matrices of an automatic size.
  - The system class Threading also allows new possibilites for the creation of multi-thread applications.



Class library(II)

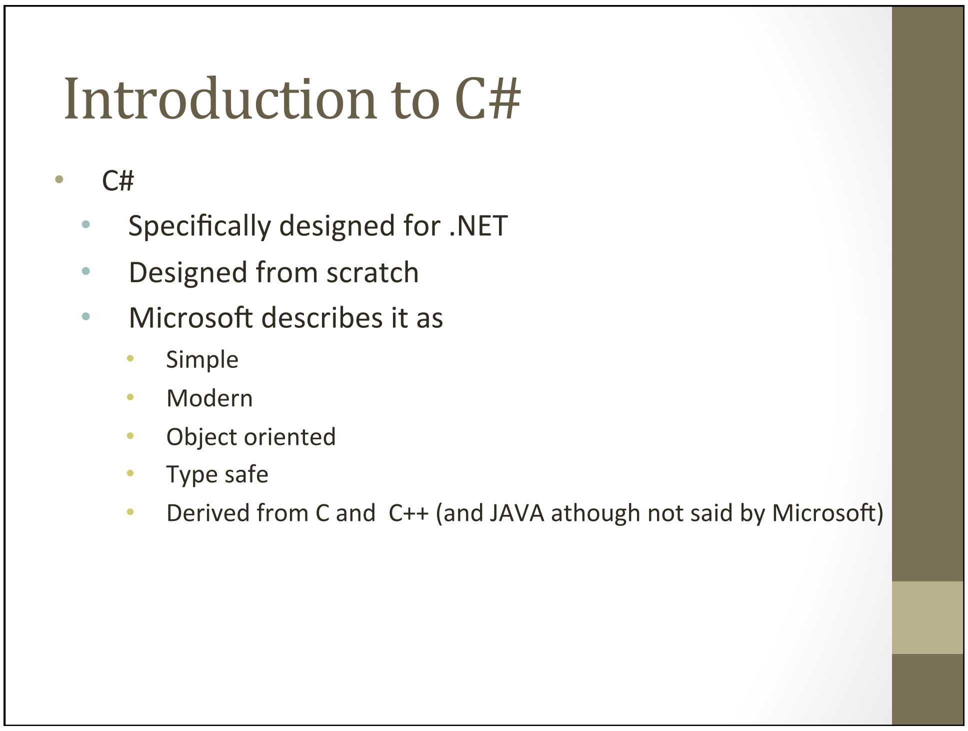
- System namespace
  - The System namespace contains fundamental classes and base classes that define the data types value and reference, commonly used, events and events descriptors, interaces, attributes and manamegent of exceptions.

Namespace	Utility of the data contained
System	Very frequently used types, such as basic types, tables, exceptions, dates, random numbers, garbage collector, console input/output, etc.
System.Collections	Colections of data commonly used such as stacks, queues, lists, dictionaries, etc.
System.Data	Manipulation of databases. They form the ADO.NET architecture.
System.IO	Manipulation of files and other data flows.
System.Net	Performing of network communications.
System.Reflection	Access to the metadata that go with the modules of code.
System.Runtime.Remoting	Access to remote objects,
System.Security	Access to the security policy in which CLR is based on.
System.Threading	Manipulation of threads.
System.Web.UI.WebControls	Creation of user interfaces windows-based for Web applications
System.Winforms	Creation of user interfaces windows-based for standard applications
System.XML	Access to data in XML format.

Programming in C#

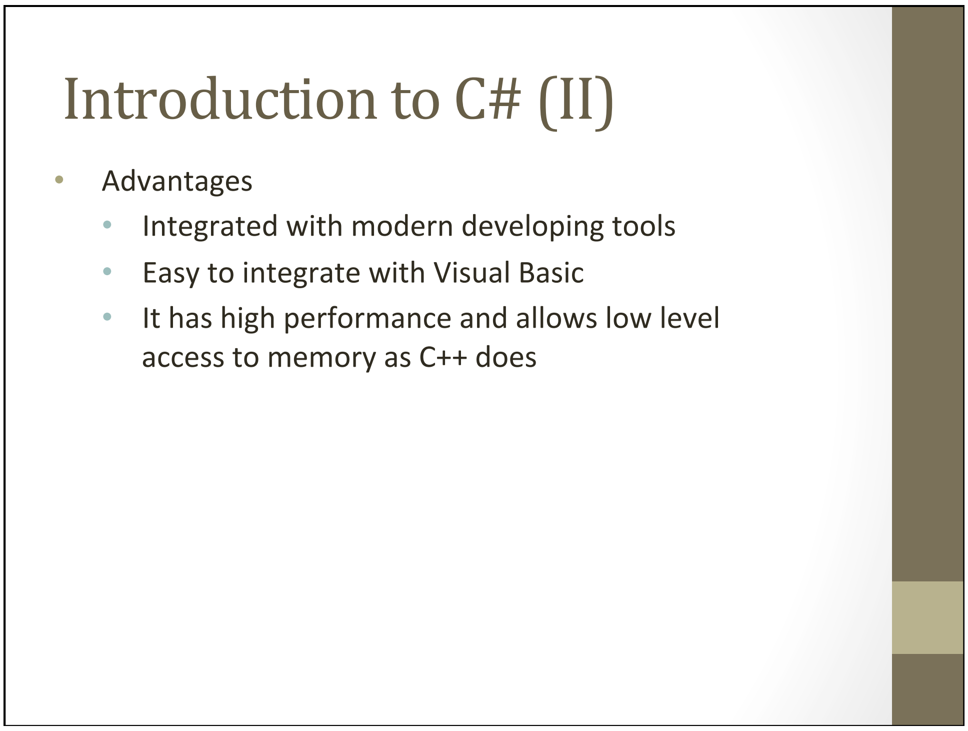
4

Introduction to C#



Introduction to C#

- C#
  - Specifically designed for .NET
  - Designed from scratch
  - Microsoft describes it as
    - Simple
    - Modern
    - Object oriented
    - Type safe
    - Derived from C and C++ (and JAVA athough not said by Microsoft)



Introduction to C# (II)

- Advantages
  - Integrated with modern developing tools
  - Easy to integrate with Visual Basic
  - It has high performance and allows low level access to memory as C++ does

Programming in C#

5

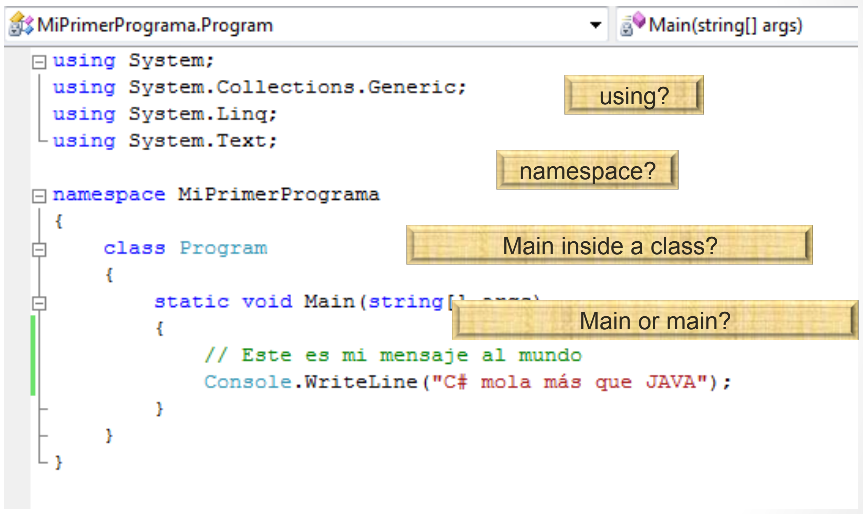
My first C# program

Exercise1

First Program

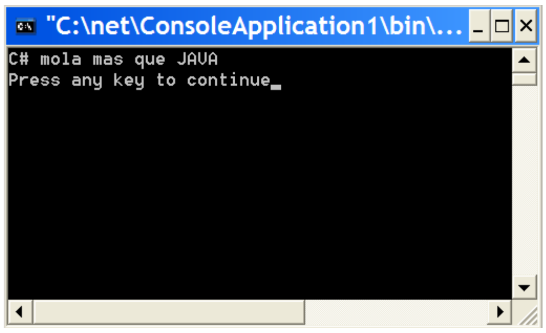
- Write a program in C# which shows a message on the screen

Demo : MiFirstProgram



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MiPrimerPrograma
{
    class Program
    {
        static void Main(string[] args)
        {
            // Este es mi mensaje al mundo
            Console.WriteLine("C# mola más que JAVA");
        }
    }
}
```

Demo : MiFirstProgram



```
C# mola mas que JAVA
Press any key to continue_
```

The main program: *Main*

- When writing it we have to:
  - Use a capital "M" like in "Main"
  - Designate a **Main** as the entry point to the program
  - Declare it as **static void Main**
- A Main can belong to multiple classes
  - In this case we have to specify which is the main entry point
- The application ends when Main ends or a when a return is performed

The class: *class*

- A C# application is a collection of classes, structures and types
- A class is a set of data and methods
- A C# application can include many classes
- Syntax :

```
class name
{
   ...
}
```

The namespaces :*namespace*

- .NET Framework offers many utility classes
  - Organized in namespaces
- System is the more used namespace
- We can refer to classes by mean of their namespace
- The using sentence

```
System.Console.WriteLine("Hola, mundo");
```

```
using System;
…
Console.WriteLine("Hola, mundo");
```

The namespaces: *using*

- The word using refers to the used namespaces.
- In the case of not referring to a namespace, when using its methods we have to specify the complete path.

The comments

- The comments are important
  - An application with the proper comments allows the designer to completely understand the application structure
- Single line comments

```
// Obtener el nombre del usuario
Console.WriteLine("¿Cómo se llama? ");
name = Console.ReadLine( );
```

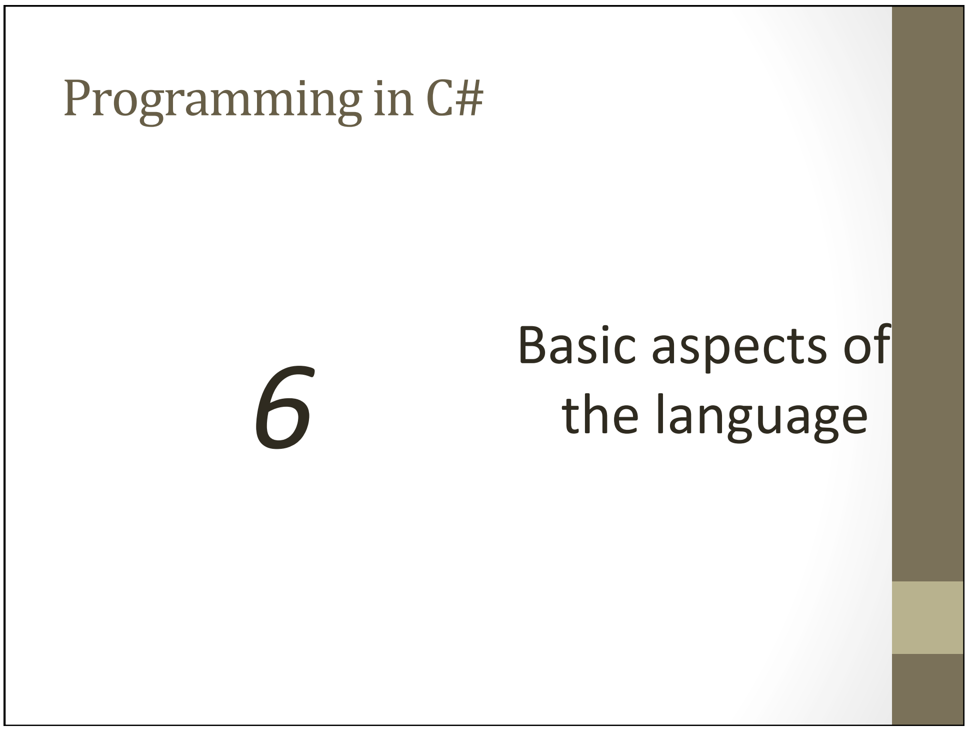
- Multiple lines comments

```
/* Encontrar la  mayor raíz
de la ecuación cuadrática */
x = (…);
```

The comments(II)

- /// Comments which allow generating automatic documentation of the project.

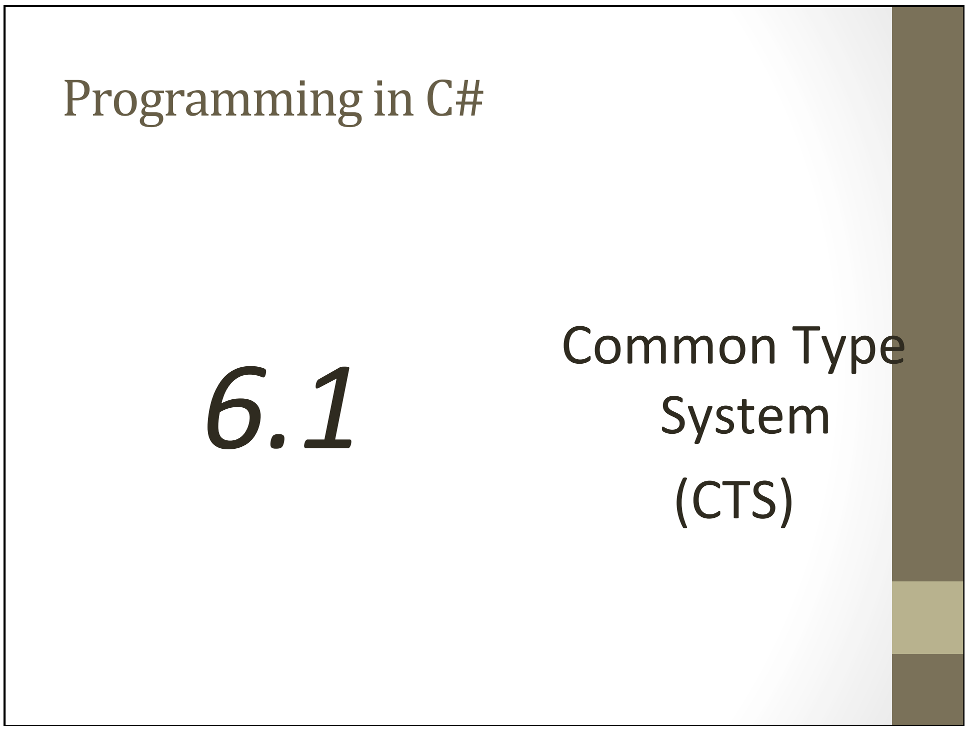
```
namespace ConsoleApplication11
{
  /// <summary>
  /// La clase 1 permite mostrar mensajes
  /// </summary>
  class Class1
  {
   /// <summary>
   /// Punto de entrada principal de la aplicación.
   /// </summary>
```



Programming in C#

6

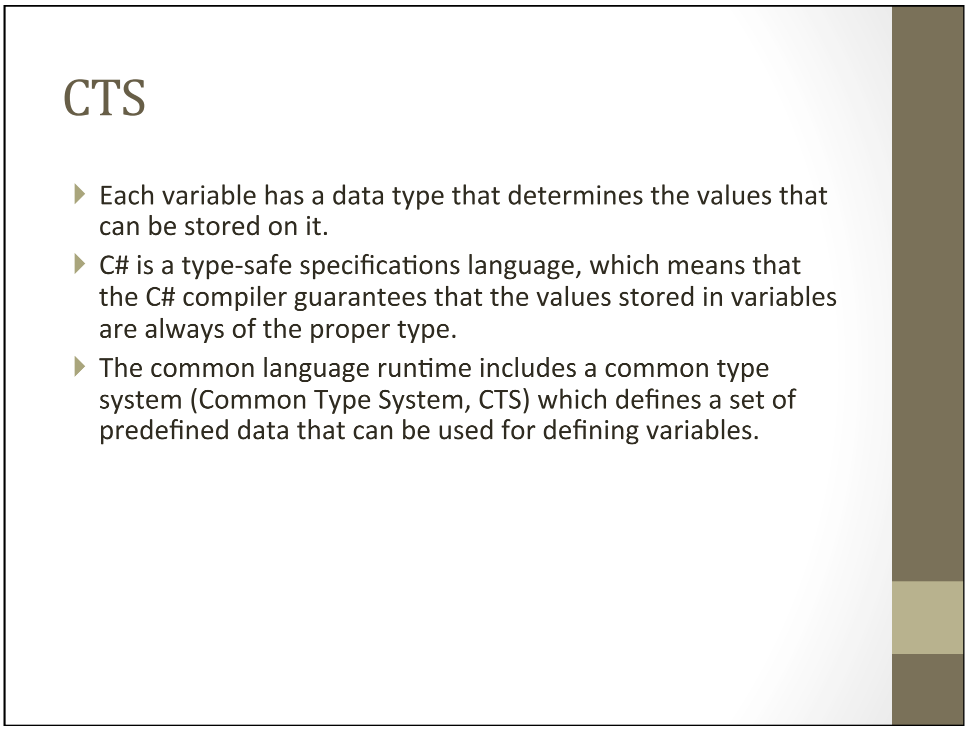
Basic aspects of the language



Programming in C#

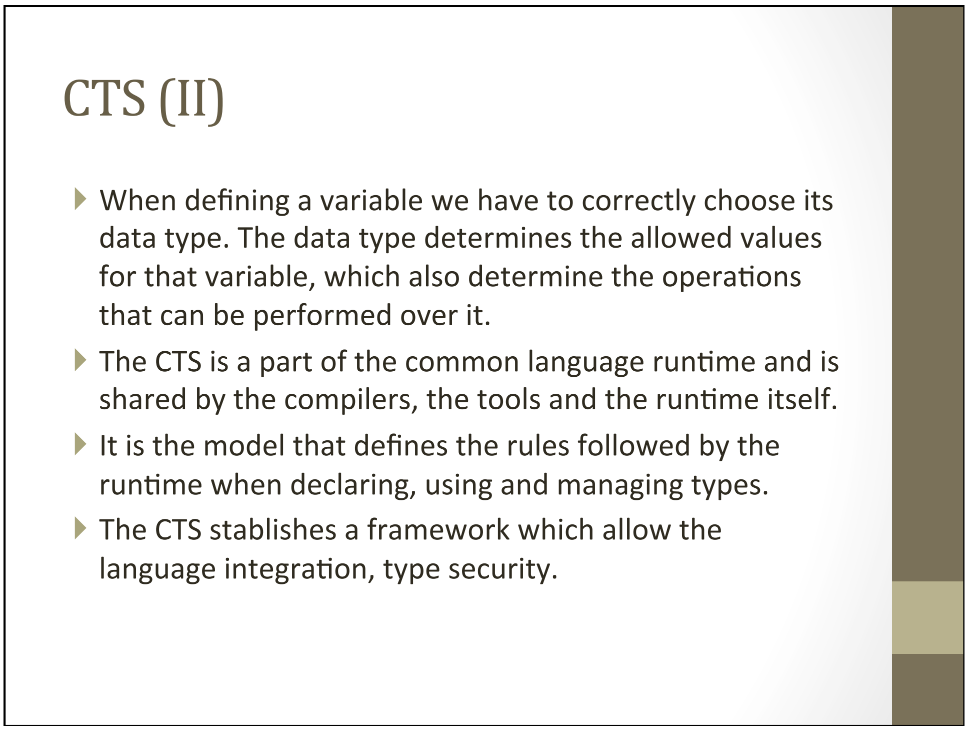
6.1

Common Type System (CTS)



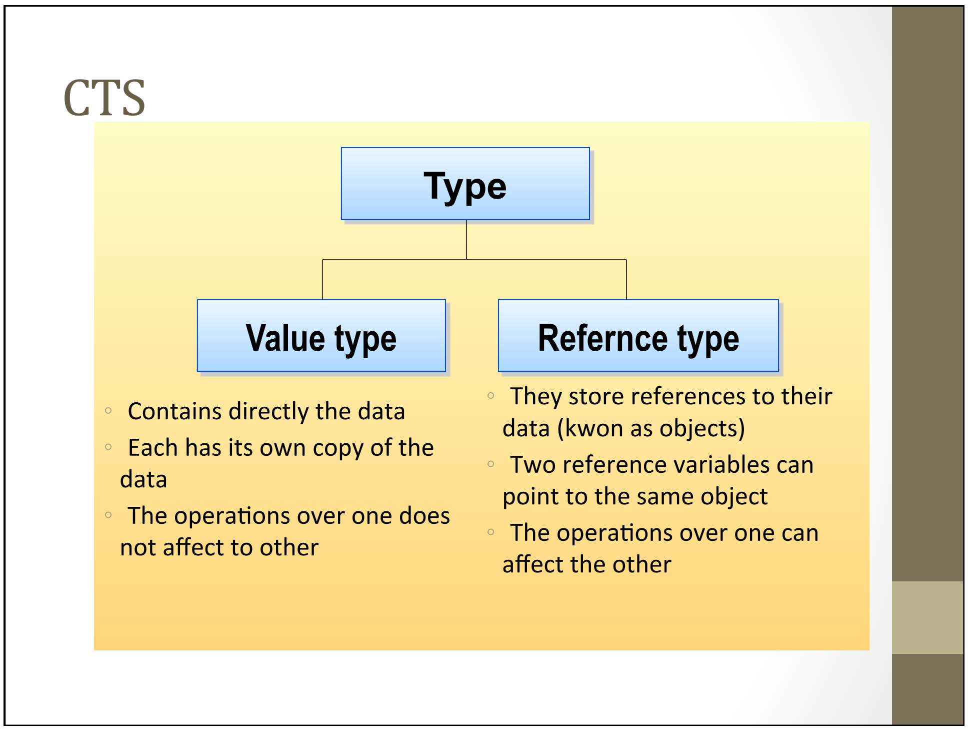
CTS

- Each variable has a data type that determines the values that can be stored on it.
- C# is a type-safe specifications language, which means that the C# compiler guarantees that the values stored in variables are always of the proper type.
- The common language runtime includes a common type system (Common Type System, CTS) which defines a set of predefined data that can be used for defining variables.

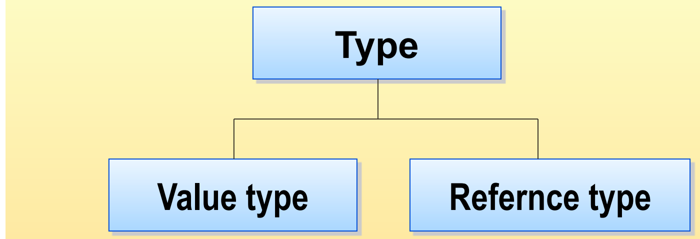


CTS (II)

- When defining a variable we have to correctly choose its data type. The data type determines the allowed values for that variable, which also determine the operations that can be performed over it.
- The CTS is a part of the common language runtime and is shared by the compilers, the tools and the runtime itself.
- It is the model that defines the rules followed by the runtime when declaring, using and managing types.
- The CTS stablishes a framework which allow the language integration, type security.

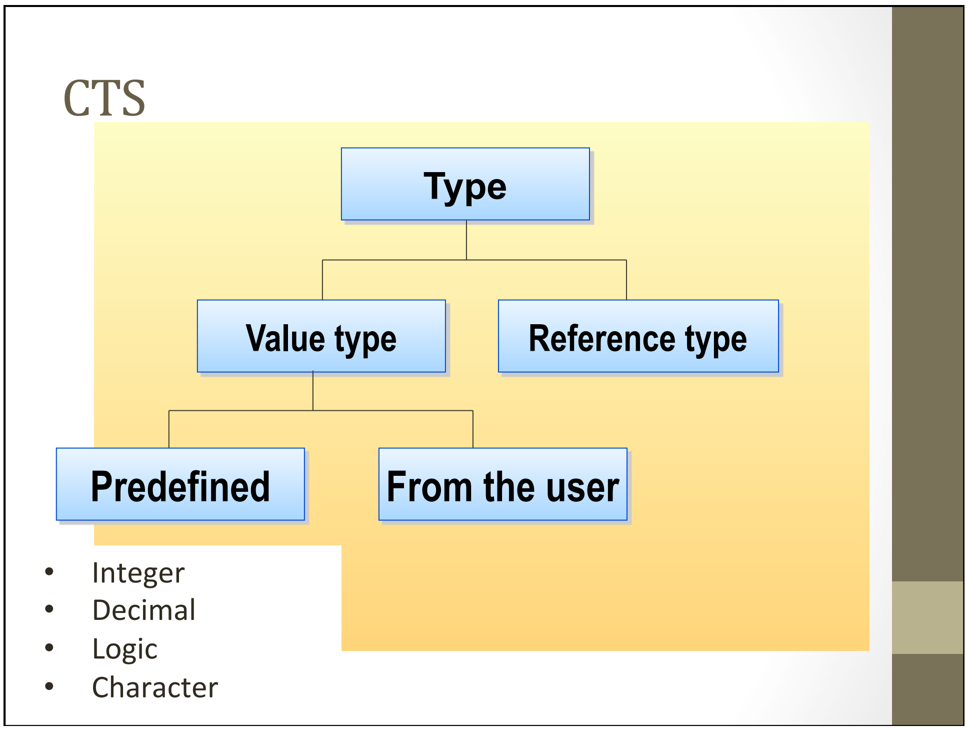


CTS

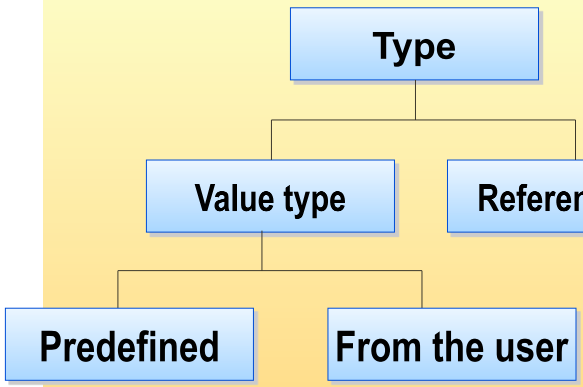


- Contains directly the data
- Each has its own copy of the data
- The operations over one does not affect to other

- They store references to their data (kwon as objects)
- Two reference variables can point to the same object
- The operations over one can affect the other



CTS



- Integer
- Decimal
- Logic
- Character

Integer

sbyte	SByte	8 bits signed
short	Int16	16 bits signed
int	Int32	32 bits signed
long	Int64	64 bits signed
byte	Byte	8 bits unsigned
ushort	UInt16	16 bits unsigned
uint	UInt32	32 bits unsigned
ulong	Uint64	64 bits unsigned

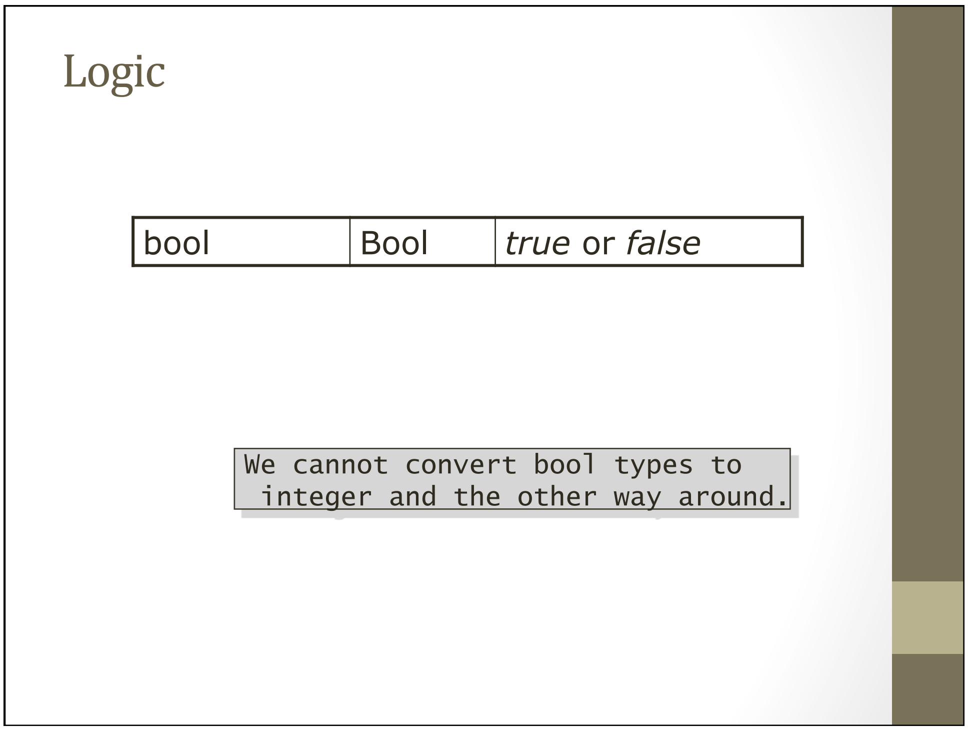
```
The byte type is not equivalent to the char type
```

Decimal

float	Single	32 bits. Simple precision (7 significant digits)
double	Double	64 bits. Double precision (15 significant digits)
decimal	Decimal	High precision (28 significant digits)

```
By default a non integer number is double
```

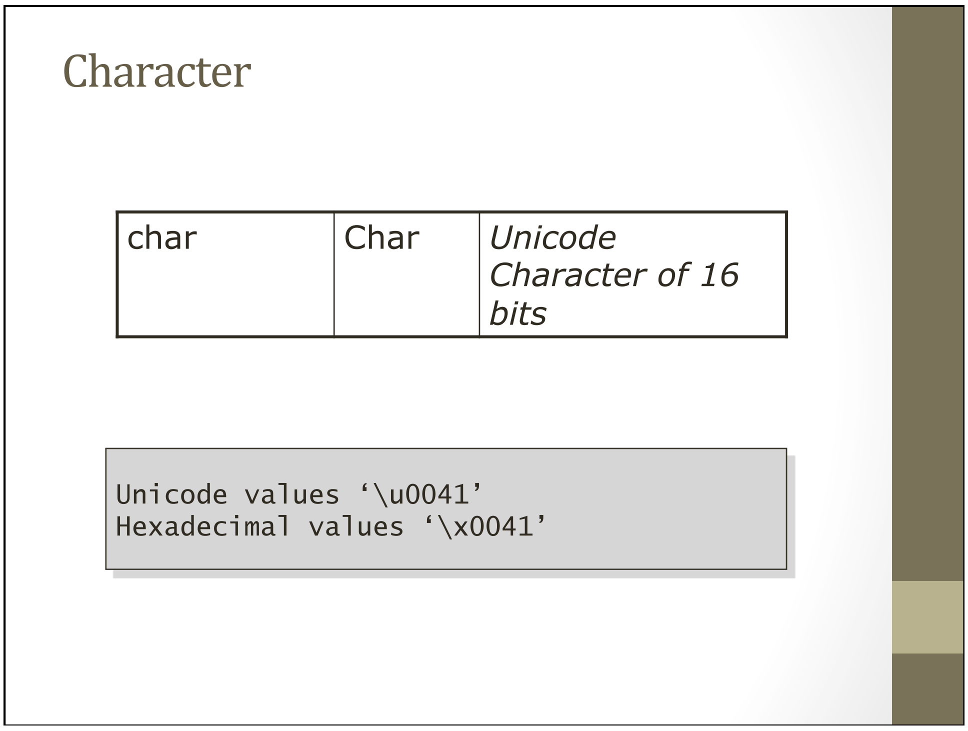
```
Explicitly
34.5F (float)
34.5M (decimal)
```



Logic

bool	Bool	*true* or *false*

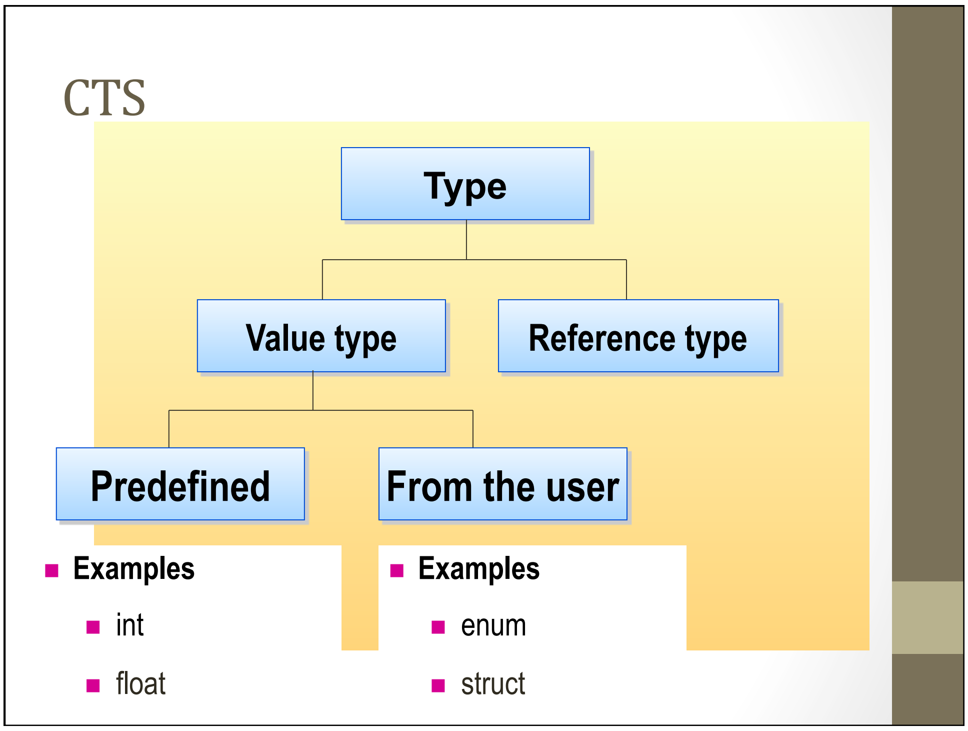
```
We cannot convert bool types to
 integer and the other way around.
```



Character

char	Char	*Unicode Character of 16 bits*

```
Unicode values ‘\u0041’
Hexadecimal values ‘\x0041’
```

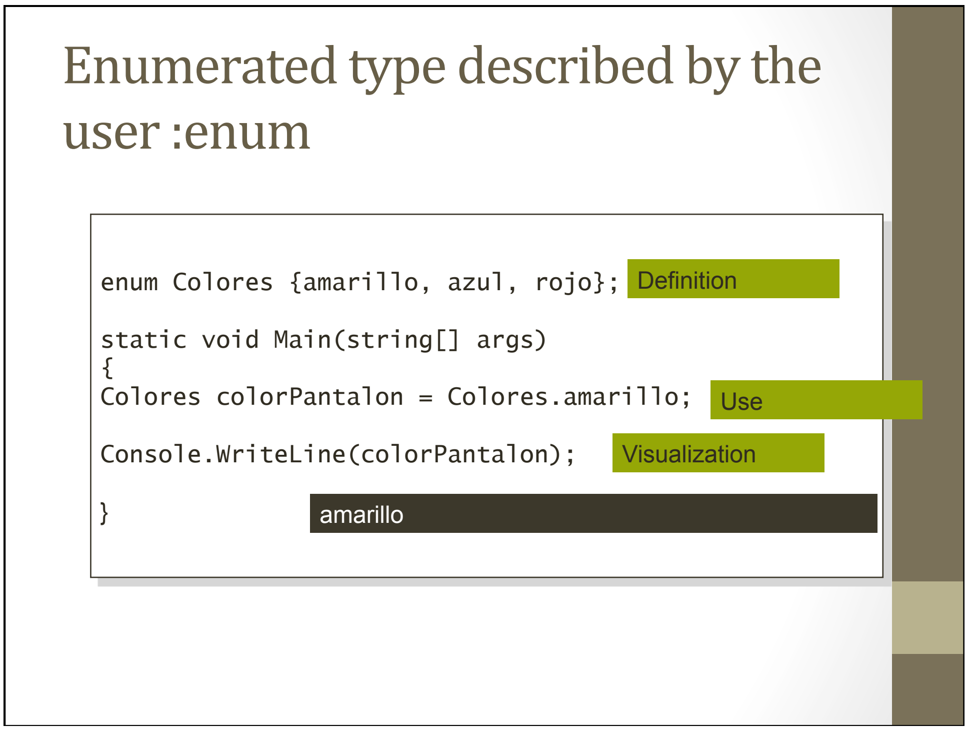


CTS

- Type
  - Value type
    - Predefined
    - From the user
  - Reference type

- Examples
  - int
  - float

- Examples
  - enum
  - struct



Enumerated type described by the user :enum

```
enum Colores {amarillo, azul, rojo};
static void Main(string[] args)
{
Colores colorPantalon = Colores.amarillo;
Console.WriteLine(colorPantalon);
}
```

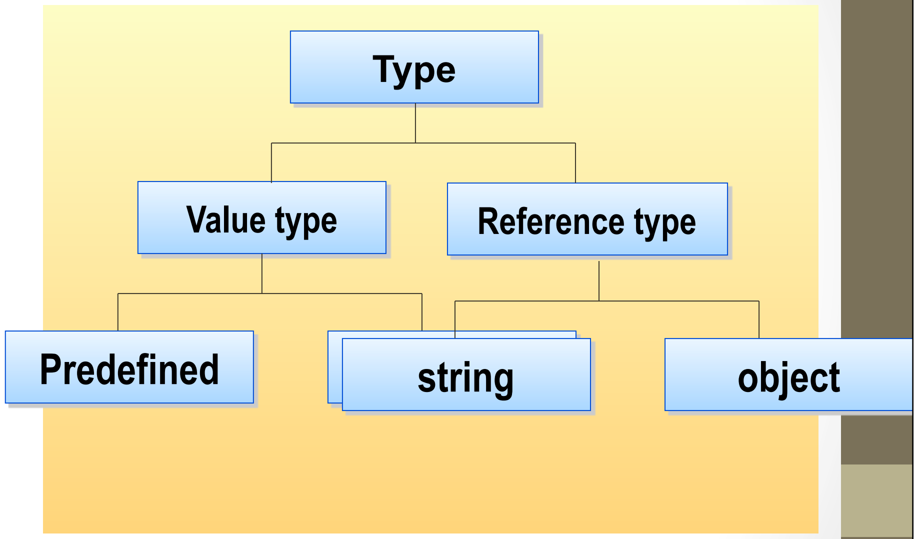
Definition

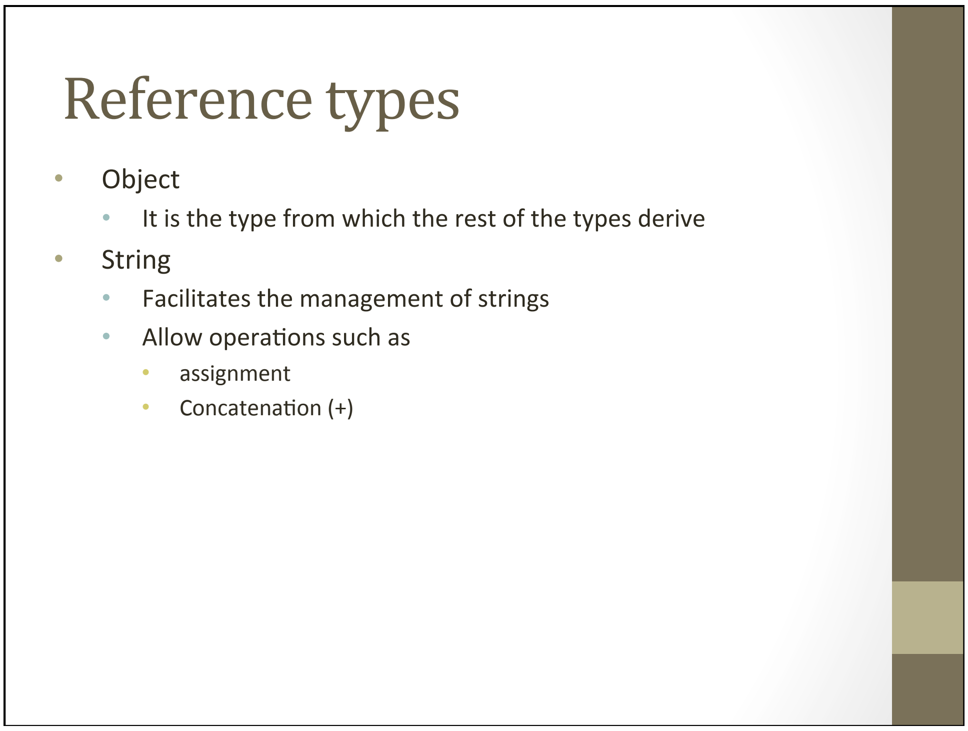
Use

Visualization

```
amarillo
```

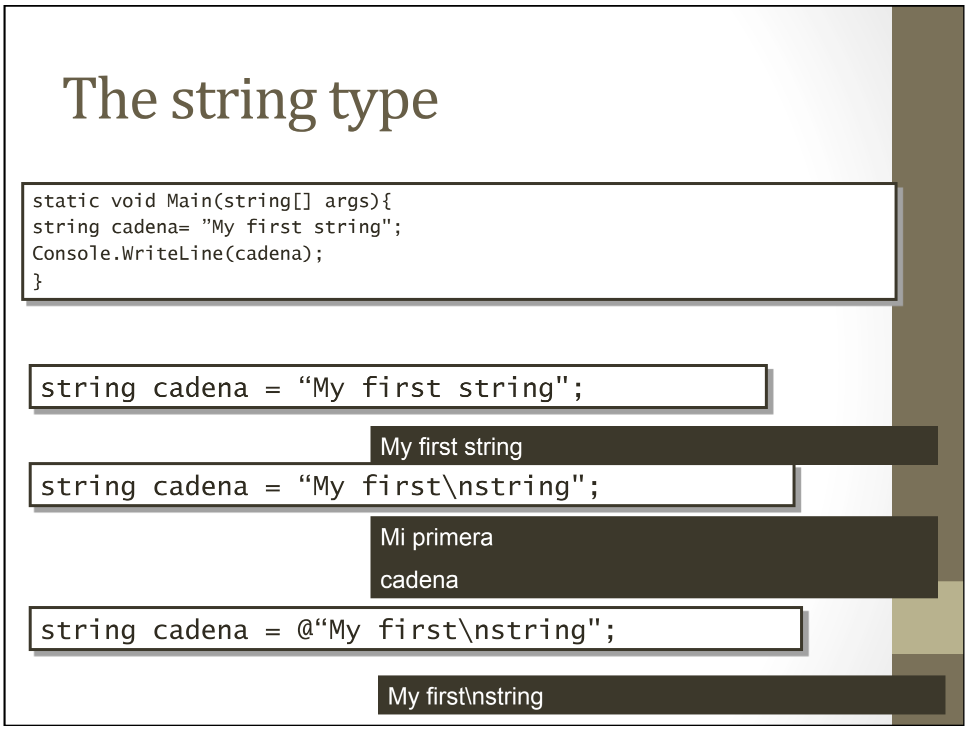
CTS





Reference types

- Object
  - It is the type from which the rest of the types derive
- String
  - Facilitates the management of strings
  - Allow operations such as
    - assignment
    - Concatenation (+)



The string type

```
static void Main(string[] args){
string cadena= ”My first string";
Console.WriteLine(cadena);
}
```

```
string cadena = “My first string";
```

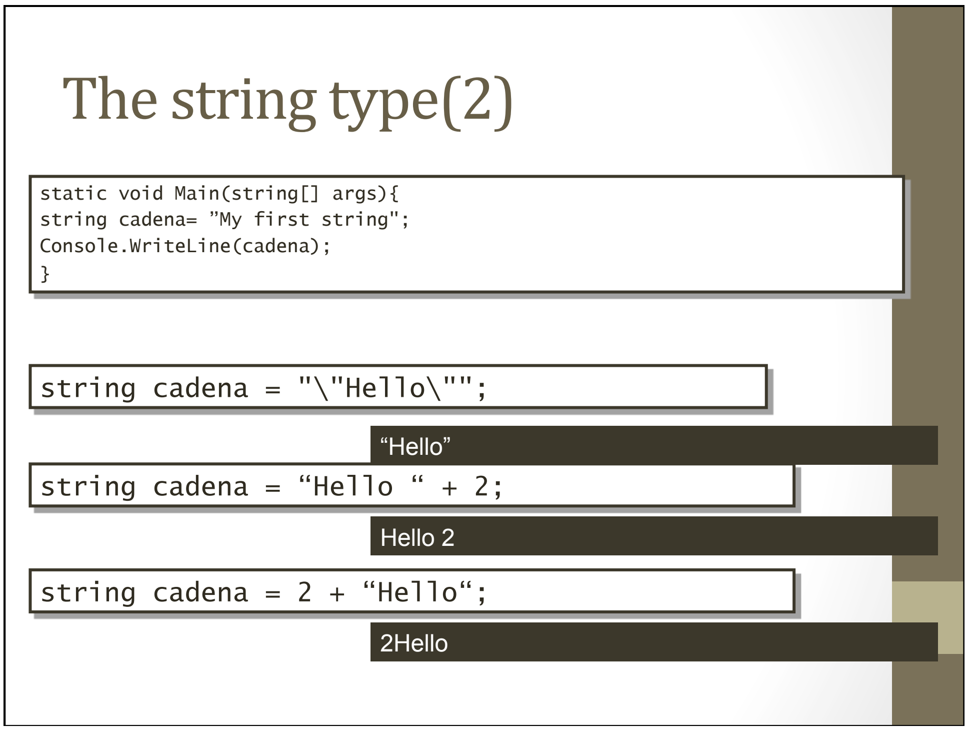
```
My first string
```

```
string cadena = “My first\nstring";
```

```
Mi primera
cadena
```

```
string cadena = @“My first\nstring";
```

```
My first\nstring
```



The string type(2)

```
static void Main(string[] args){
string cadena= ”My first string";
Console.WriteLine(cadena);
}
```

```
string cadena = "\"Hello\"";
```

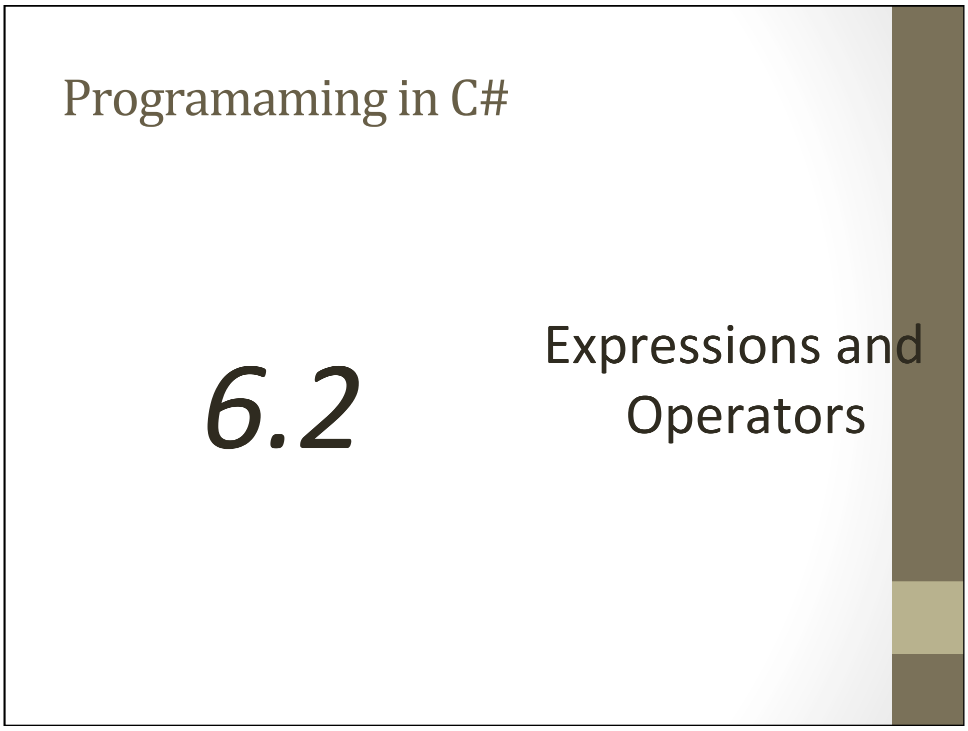
```
“Hello”
```

```
string cadena = “Hello “ + 2;
```

```
Hello 2
```

```
string cadena = 2 + “Hello“;
```

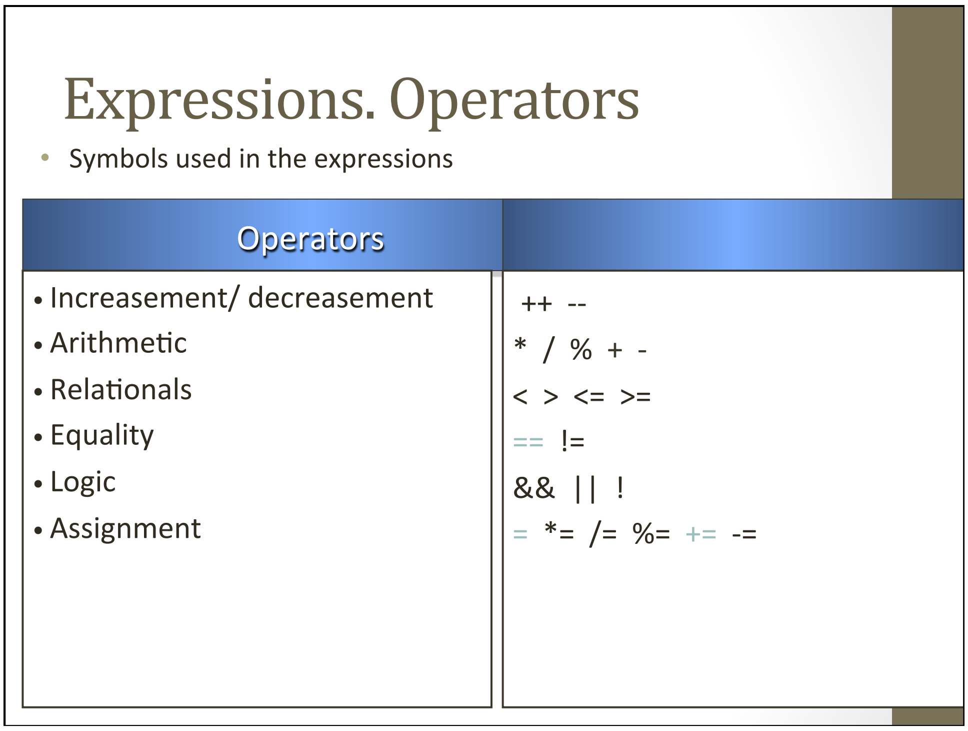
```
2Hello
```



Programaming in C#

6.2

Expressions and Operators



Expressions. Operators

- Symbols used in the expressions

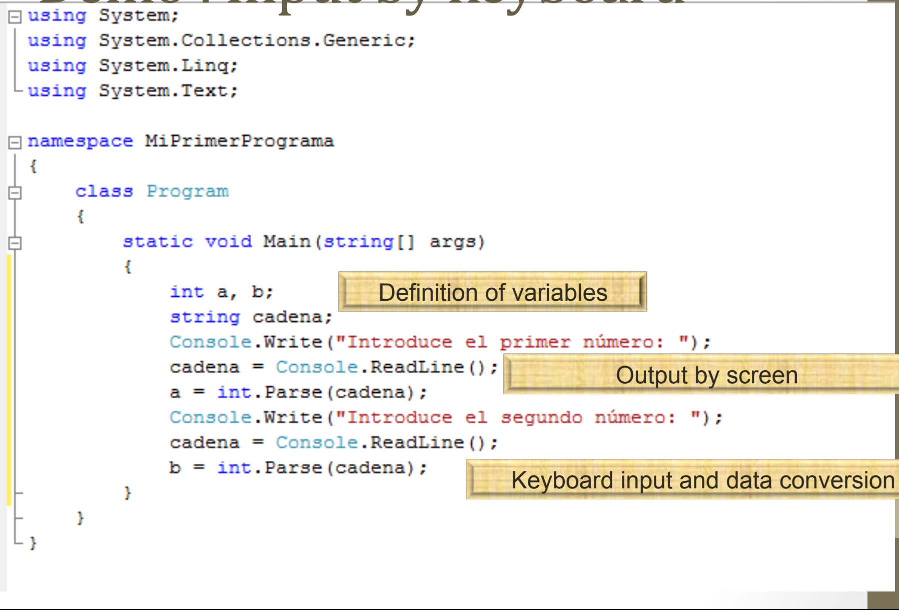
Operators	
• Increasement/ decreasement	++ --
• Arithmetic	* / % + -
• Relationals	< > <= >=
• Equality	== !=
• Logic	&& || !
• Assignment	= *= /= %= += -=

Exercise 2

Input by keyboard

- Write a program in C# which reads two numbers by keyboard.

Demo : Input by keyboard



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MiPrimerPrograma
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b;
            string cadena;
            Console.Write("Introduce el primer número: ");
            cadena = Console.ReadLine();
            a = int.Parse(cadena);
            Console.Write("Introduce el segundo número: ");
            cadena = Console.ReadLine();
            b = int.Parse(cadena);
        }
    }
}
```

Definition of variables

Output by screen

Keyboard input and data conversion

Programming in C#

6.3

Variables and Constants

Declaration of variables

- They are declared by data type and variable name:

```
int objectAccount;
```

- --o--

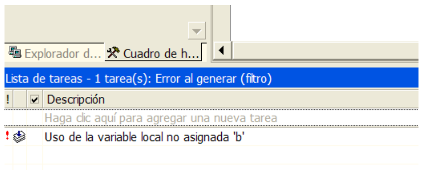
```
int objectAccount, employeeNumber;
```

```
int objectAccount,
    employeeNumber;
```

Declaration of variables (II)

Question. What does it happens if we try to access a non initialized variable?

```
static void Main(string[] args)
    {
        int a,b;
        a = b + 2;
    }
```



Lista de tareas - 1 tarea(s): Error al generar (filtro)

! | ✓ | Descripción

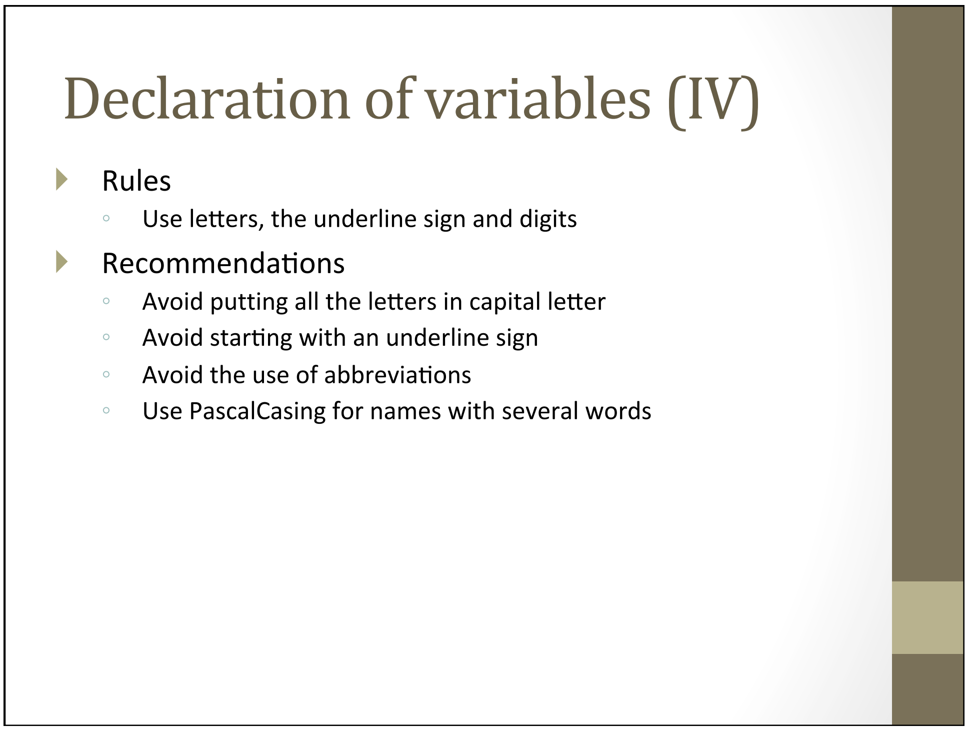
Haga clic aquí para agregar una nueva tarea

! | Uso de la variable local no asignada 'b'

!!!! The C# compiler demands that every variable is initialized before being used

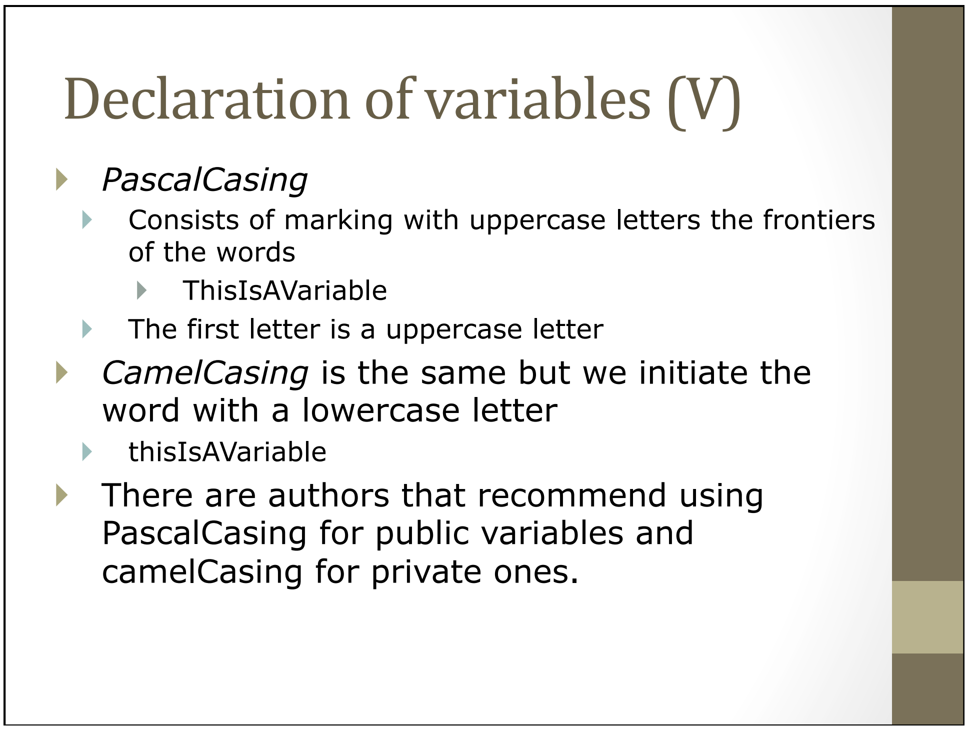
Declaration of variables (III)

- C# is much more safe than
  - C++ Leaves the programmer the task of guaranteeing that the variables are initialized. (Modern compilers emit warning messsages but allow to generate the executable file).
  - VB initializes to 0 by default the variables.



Declaration of variables (IV)

- Rules
  - Use letters, the underline sign and digits
- Recommendations
  - Avoid putting all the letters in capital letter
  - Avoid starting with an underline sign
  - Avoid the use of abbreviations
  - Use PascalCasing for names with several words



Declaration of variables (V)

- *PascalCasing*
  - Consists of marking with uppercase letters the frontiers of the words
    - ThisIsAVariable
  - The first letter is a uppercase letter
- *CamelCasing* is the same but we initiate the word with a lowercase letter
  - thisIsAVariable
- There are authors that recommend using PascalCasing for public variables and camelCasing for private ones.

Declaration of variables (VI)

- Assigning values to already declared variables:

```
int employeeNumber;
employeeNumber = 23;
```

- Initialize a variable when is declared:

```
int employeeNumber = 23;
```

- It is also possible initialize values of characters:

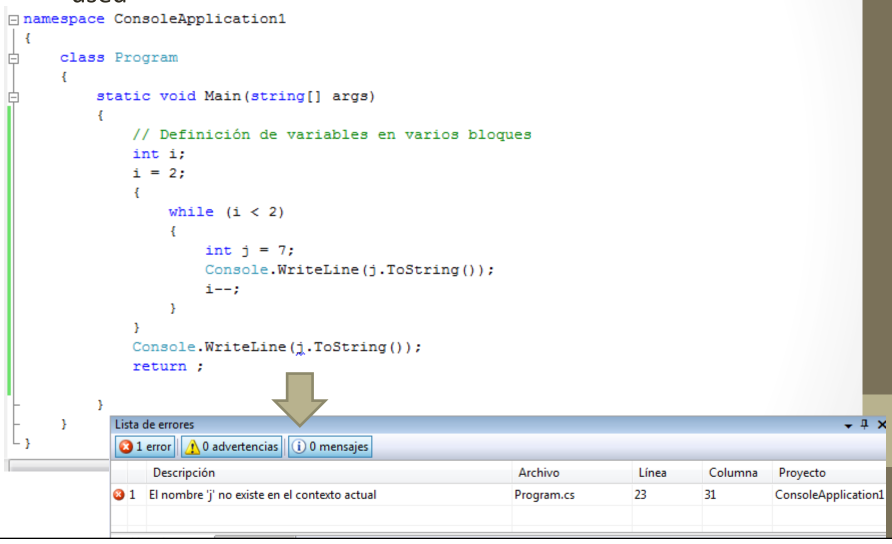
```
char initialName = 'J';
```

Variables scope

- The scope of a variable coincides with the one of its container class
- The scope of a variable ends with the curled bracket (}) which closes the block or method in which the variable is declared.
- The variables can be declared inside a loop, being only visible inside it.
  - This is compliant with the ANSI standard of C++
  - Previous versions of Microsoft C++ were not compliant.

Variables scope (II)

- The C# compiler requires that any variable is initialized before being used



```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Definición de variables en varios bloques
            int i;
            i = 2;
            {
                while (i < 2)
                {
                    int j = 7;
                    Console.WriteLine(j.ToString());
                    i--;
                }
            }
            Console.WriteLine(j.ToString());
            return ;

        }
    }
}
```

Lista de errores

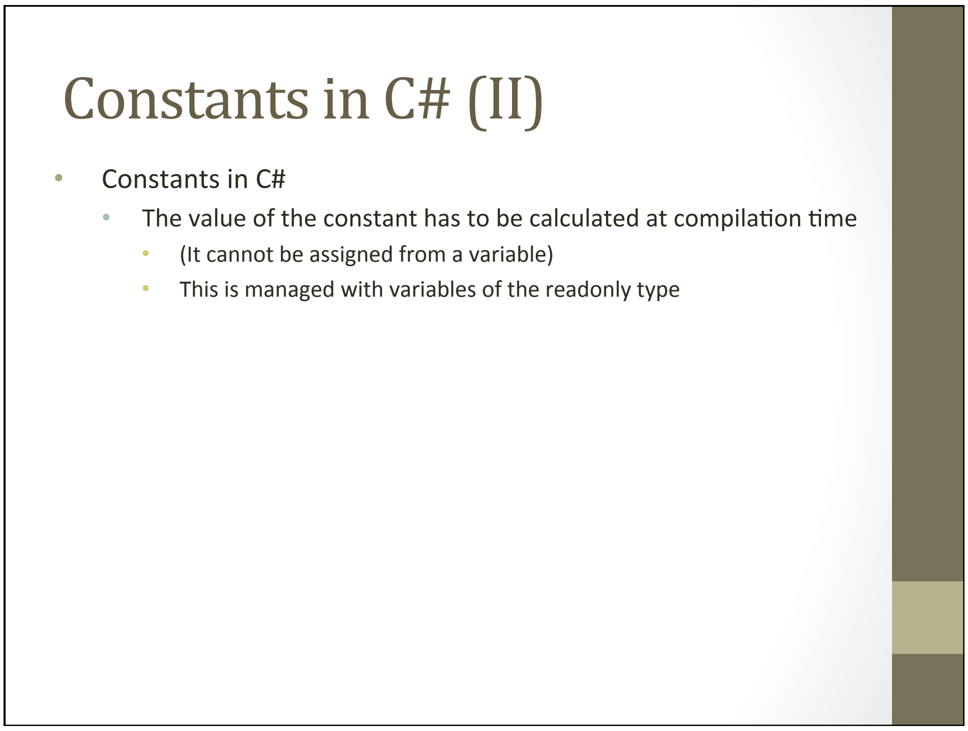
1 error | 0 advertencias | 0 mensajes

		Descripción	Archivo	Línea	Columna	Proyecto
✖	1	El nombre 'j' no existe en el contexto actual	Program.cs	23	31	ConsoleApplication1

Constants

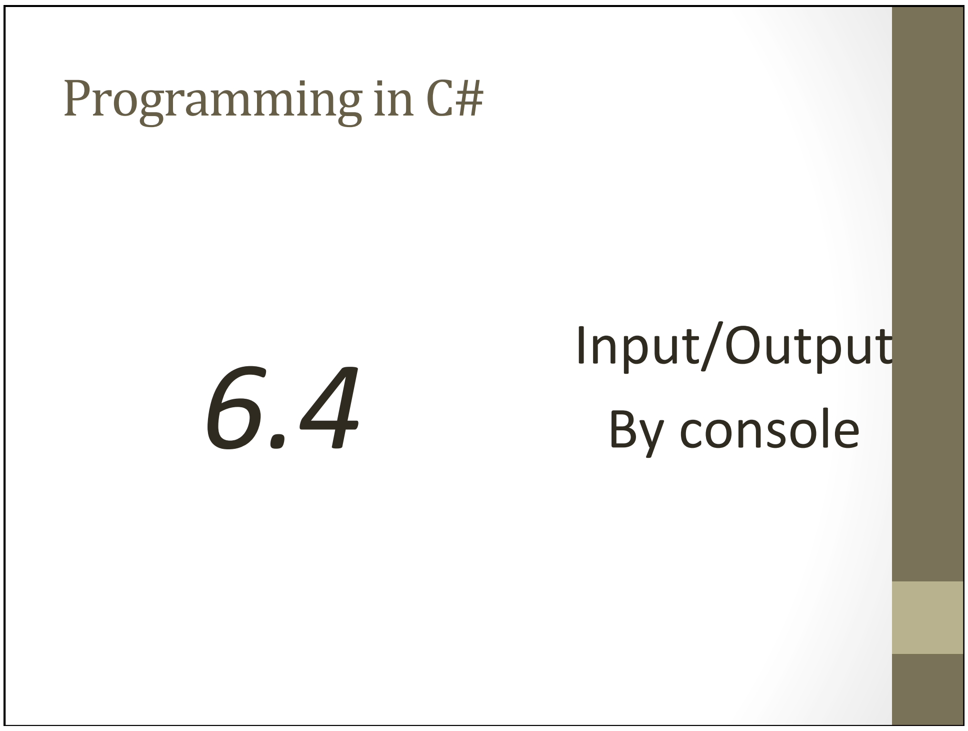
- A constant is a variable that cannot change its value
- They are defined in the same way as the variables (adding the keyword *const*)
- They have to be mandatory initialized when declared.

```
const int valor = 6378;
```



Constants in C# (II)

- Constants in C#
  - The value of the constant has to be calculated at compilation time
    - (It cannot be assigned from a variable)
    - This is managed with variables of the readonly type

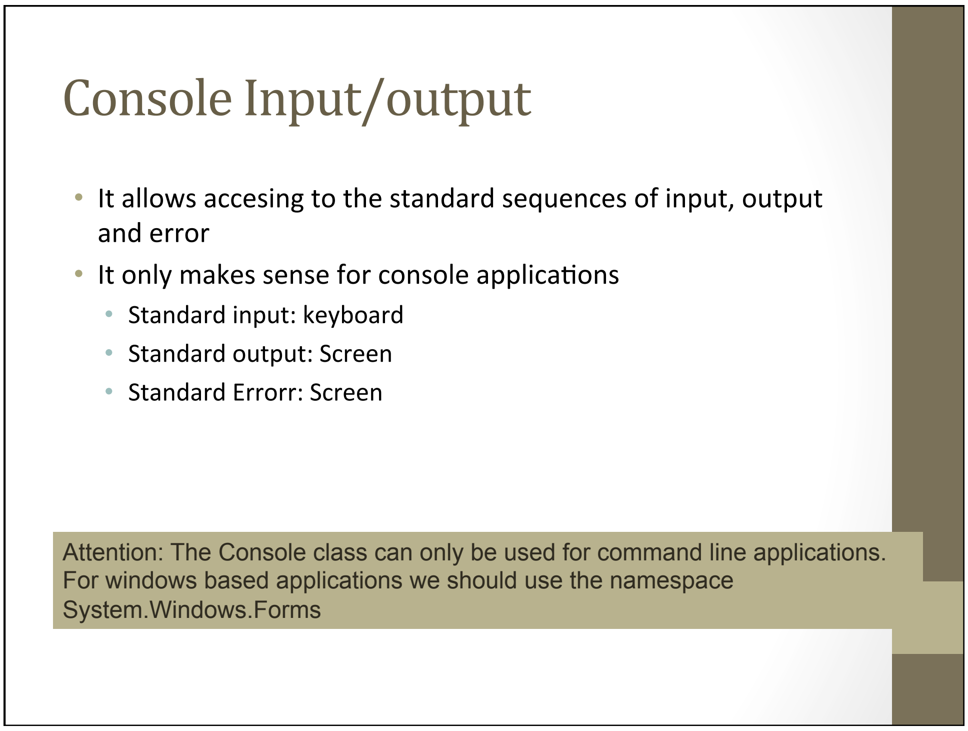


Programming in C#

6.4

Input/Output

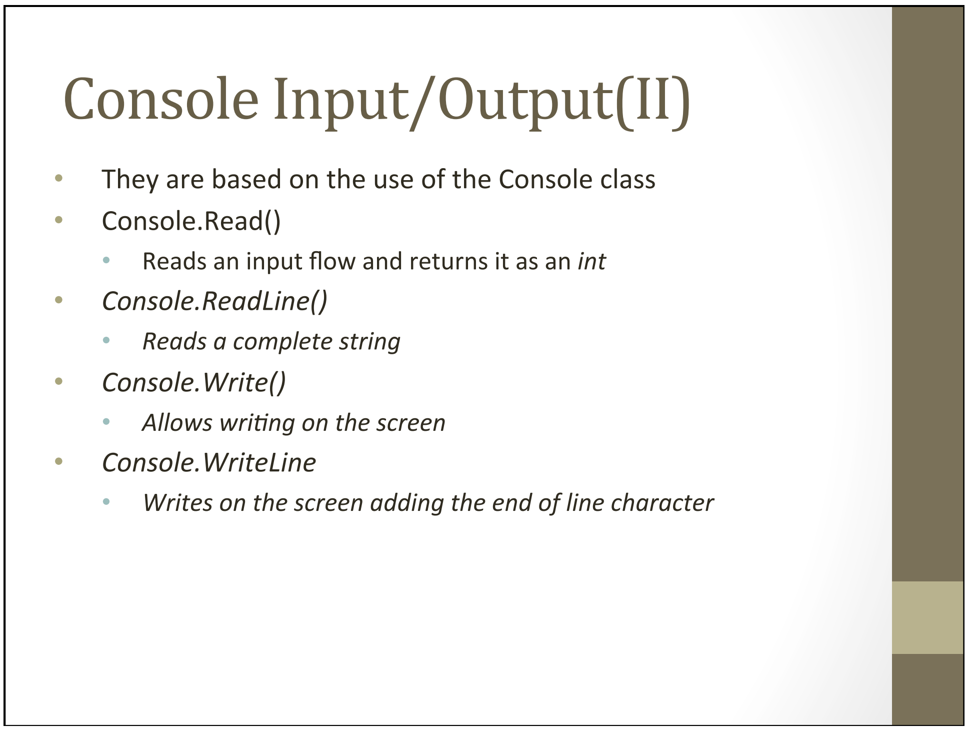
By console



Console Input/output

- It allows accesing to the standard sequences of input, output and error
- It only makes sense for console applications
  - Standard input: keyboard
  - Standard output: Screen
  - Standard Errorr: Screen

Attention: The Console class can only be used for command line applications. For windows based applications we should use the namespace System.Windows.Forms

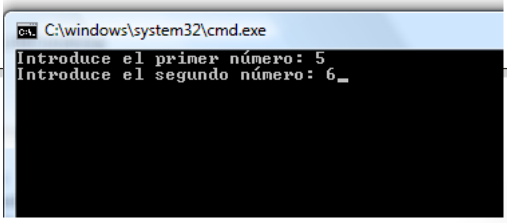


Console Input/Output(II)

- They are based on the use of the Console class
- Console.Read()
  - Reads an input flow and returns it as an *int*
- *Console.ReadLine()*
  - *Reads a complete string*
- *Console.Write()*
  - *Allows writing on the screen*
- *Console.WriteLine*
  - *Writes on the screen adding the end of line character*

Console Input/Output(III)

```
static void Main(string[] args)
    {
    int a, b;
    string cadena;
    Console.Write ("Introduce el primer número: ");
    cadena = Console.ReadLine();
    a = int.Parse(cadena);
    Console.Write("Introduce el segundo número: ");
    cadena = Console.ReadLine();
    b = int.Parse(cadena);
    }
```

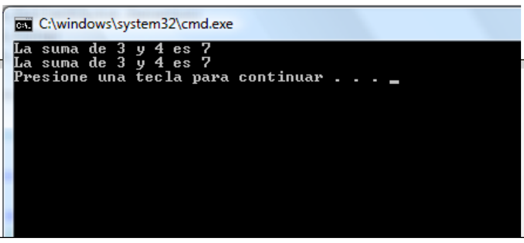


```
Introduce el primer número: 5
Introduce el segundo número: 6_
```

Parameters of writeline

- Chain that contains between curled brackets the parameters that are indicated next separated by commas

```
static void Main(string[] args)
  {
  int a, b;
  a = 3;
  b = 4;
  Console.WriteLine("La suma de {0} y {1} es {2} ",a,b, a+b);
  Console.WriteLine("La suma de " + a + " y " + b + " es " + (a+b));
}
```

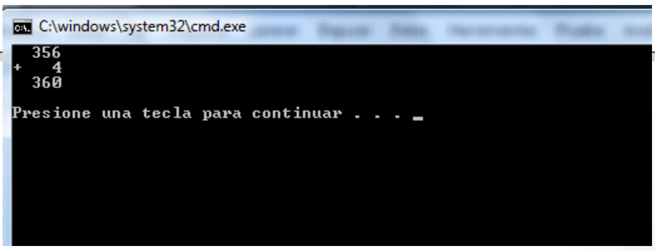


```
La suma de 3 y 4 es 7
La suma de 3 y 4 es 7
Presione una tecla para continuar . . . _
```

Formatting of the output

- The output can be formatted using the format {n.m} where n is the index of the parameter and w the width of the output

```
static void Main(string[] args)
  {
  int a, b;
  a = 356;
  b = 4;
  Console.WriteLine(" {0,4}\n+{1,4}\n {2,4}\n",a,b, a+b);
  }
```

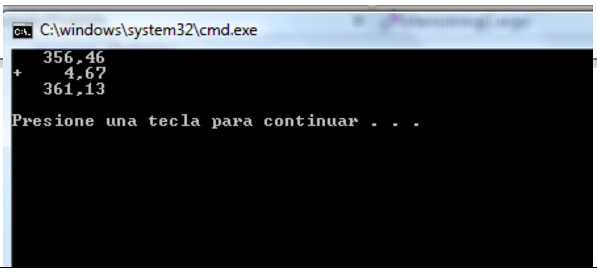


```
  356
+   4
  360

Presione una tecla para continuar . . . _
```

Precision

- We can indicate the number of decimals using F and a number

```
static void Main(string[] args)
 {
double a, b;
a = 356.459;
b = 4.67;
Console.WriteLine(" {0,8:F2}\n+{1,8:F2}\n {2,8:F2}\n",a,b, a+b);
 }
```



```
   356,46
+    4,67
   361,13

Presione una tecla para continuar . . .
```

Exercise 3

Addition and Square of two numbers

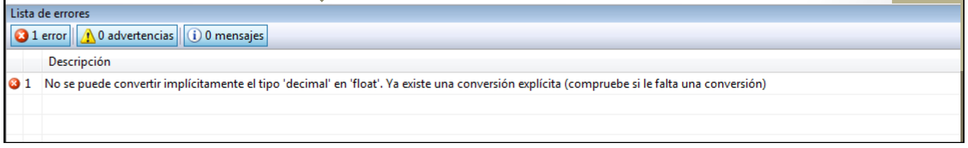
- Write a program in C# which reads two numbers by keyboard. Then it will show the result of adding both numbers and the square of this addition.

Demo : Input by keyboard

```
static void Main(string[] args)
{
    int a, b;
    decimal suma;
    float cuadrado;
    string cadena;
    Console.Write("Introduce el primer número: ");
    cadena = Console.ReadLine();
    a = int.Parse(cadena);
    Console.Write("Introduce el segundo número: ");
    cadena = Console.ReadLine();
    b = int.Parse(cadena);
    suma = a + b;
    cuadrado = suma * suma;
}
```

Data conversion without lost of information

Data conversion with lost of information



```
Lista de errores
1 error | 0 advertencias | 0 mensajes
   Descripción
1  No se puede convertir implícitamente el tipo 'decimal' en 'float'. Ya existe una conversión explícita (compruebe si le falta una conversión)
```

Programming in C#

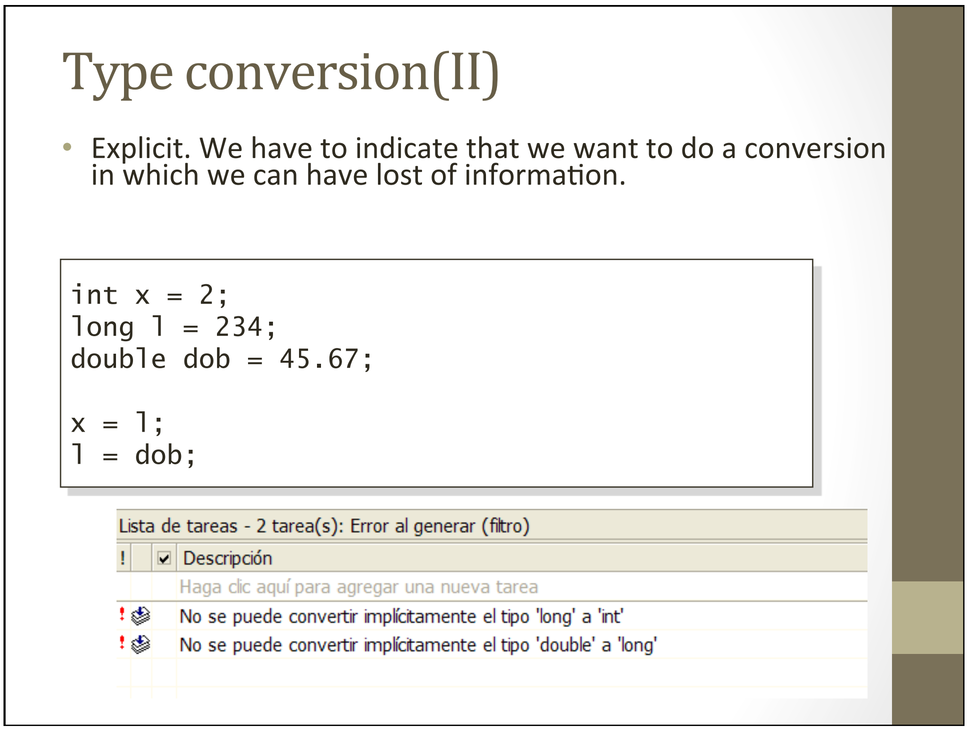
6.5

Type conversion

Type conversion

- Implicit
  - Automatic, when it is not possible to lose information

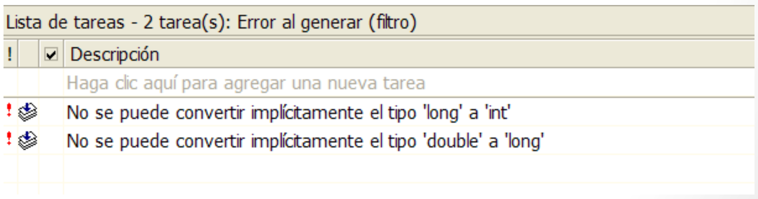
```
int x = 2;
long l = 234;
double dob = 45.67;

dob = x;
l = x;
```



Type conversion(II)

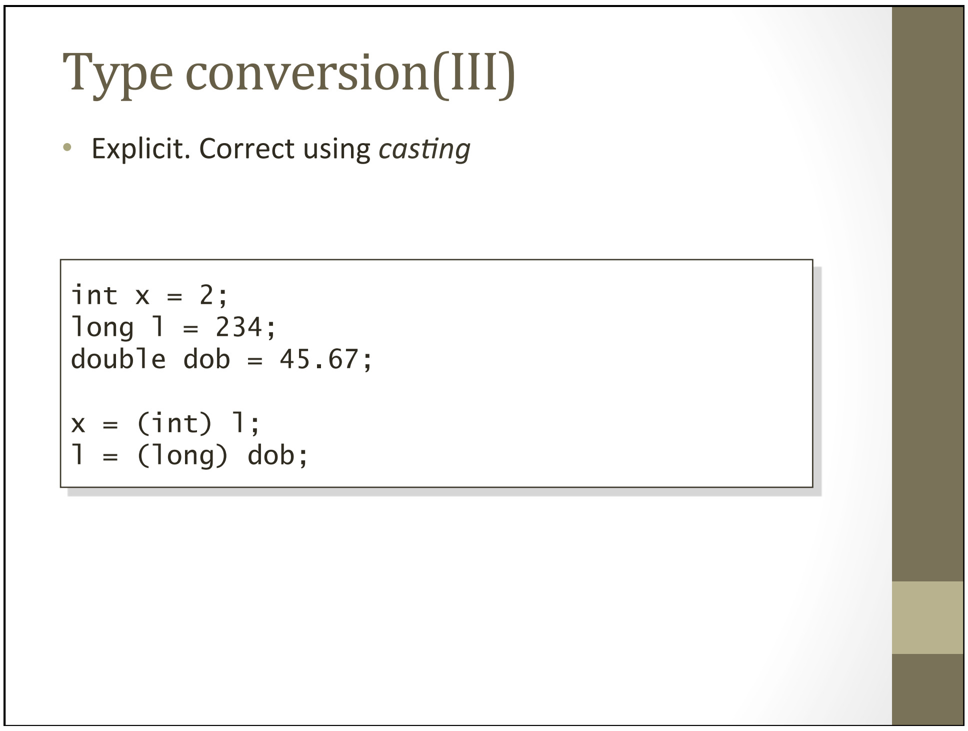
- Explicit. We have to indicate that we want to do a conversion in which we can have lost of information.

```
int x = 2;
long l = 234;
double dob = 45.67;

x = l;
l = dob;
```



Lista de tareas - 2 tarea(s): Error al generar (filtro)

!	☑	Descripción
		Haga clic aquí para agregar una nueva tarea
!		No se puede convertir implícitamente el tipo 'long' a 'int'
!		No se puede convertir implícitamente el tipo 'double' a 'long'



Type conversion(III)

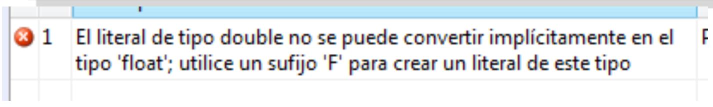
- Explicit. Correct using *casting*

```
int x = 2;
long l = 234;
double dob = 45.67;

x = (int) l;
l = (long) dob;
```

Type conversion(IV)

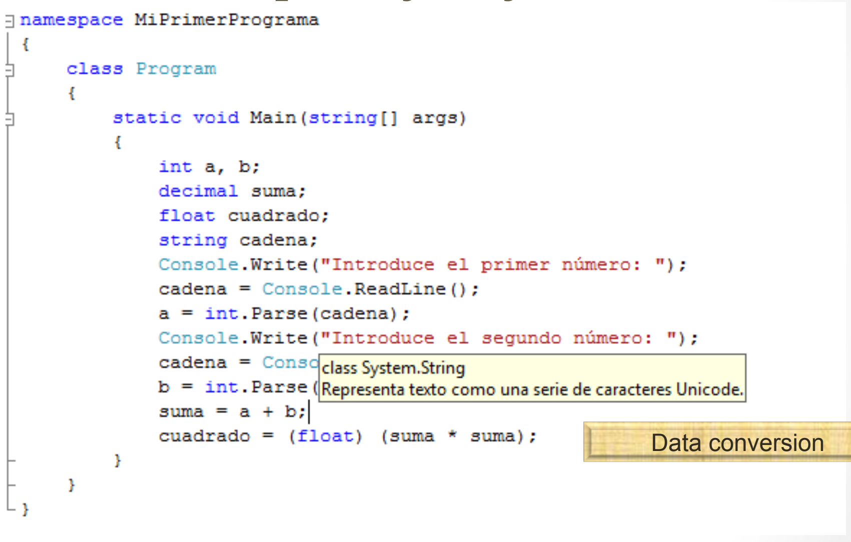
- Is the next program correct?

```
static void Main(string[] args)
      {
      float numero;
      numero = 28.67;
      }
```

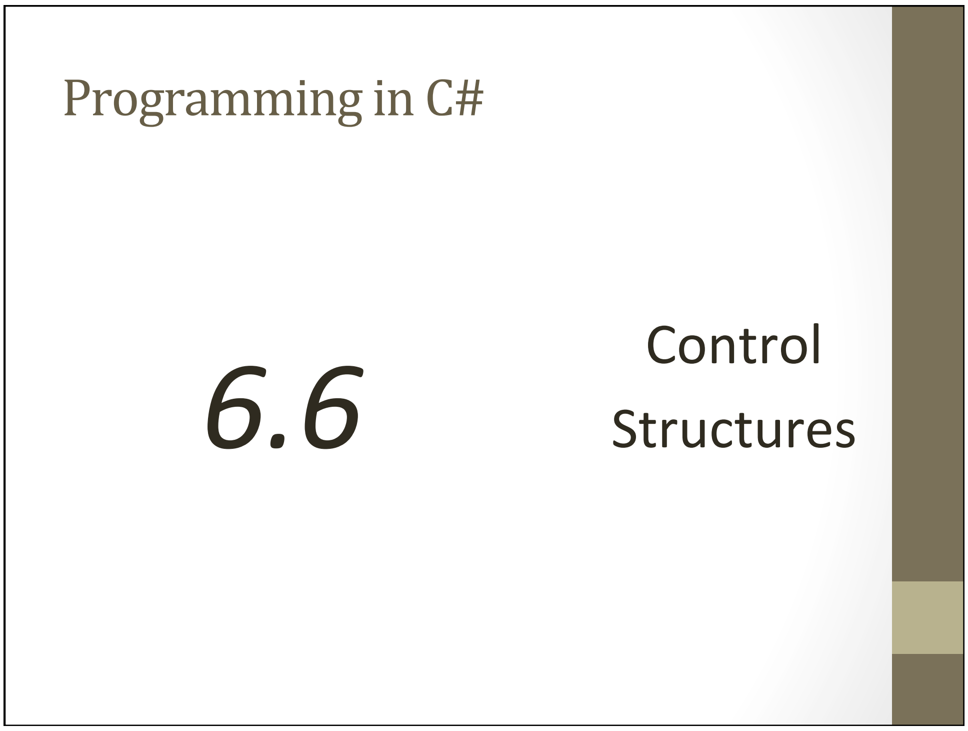

1 El literal de tipo double no se puede convertir implícitamente en el tipo 'float'; utilice un sufijo 'F' para crear un literal de este tipo

```
static void Main(string[] args)
      {
      float numero;
      numero = 28.67F;
      }
```

Demo : Input by keyboard


```
namespace MiPrimerPrograma
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b;
            decimal suma;
            float cuadrado;
            string cadena;
            Console.Write("Introduce el primer número: ");
            cadena = Console.ReadLine();
            a = int.Parse(cadena);
            Console.Write("Introduce el segundo número: ");
            cadena = Conso
            b = int.Parse(
            suma = a + b;
            cuadrado = (float) (suma * suma);
        }
    }
}
```
class System.String
Representa texto como una serie de caracteres Unicode.

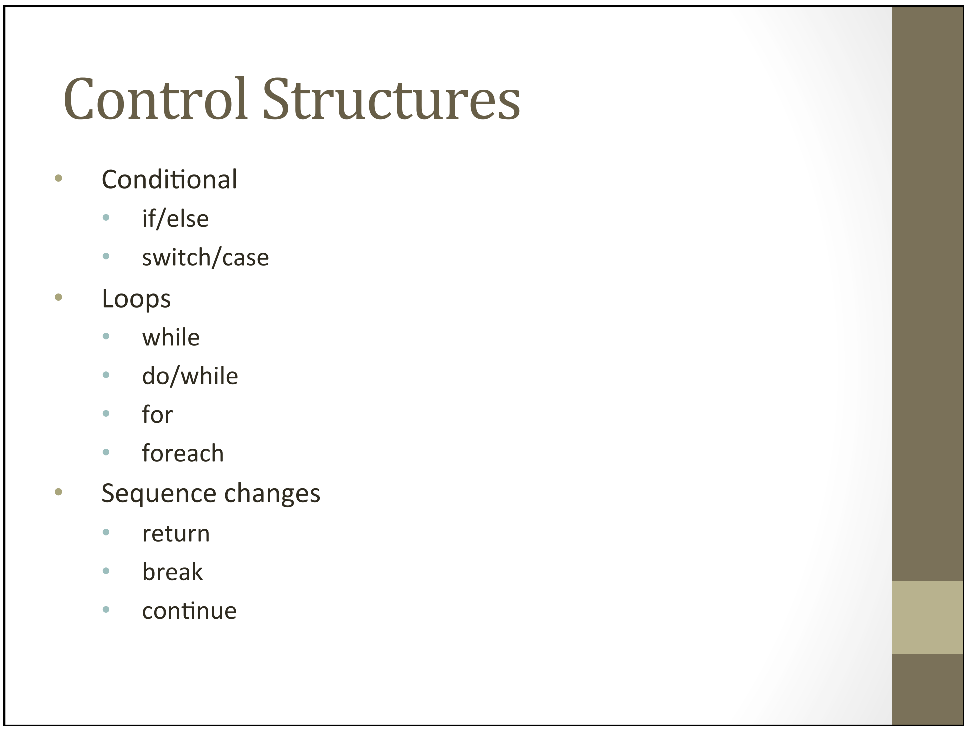
Data conversion



Programming in C#

6.6

Control Structures



Control Structures

- Conditional
  - if/else
  - switch/case
- Loops
  - while
  - do/while
  - for
  - foreach
- Sequence changes
  - return
  - break
  - continue

Simple conditional

- if

```
if (saldo > reintegro)
     Console.WriteLine("OK");
```

- if else

```
if (saldo > reintegro)
 Console.WriteLine("OK");
   else
 Console.WriteLine("No OK");
```

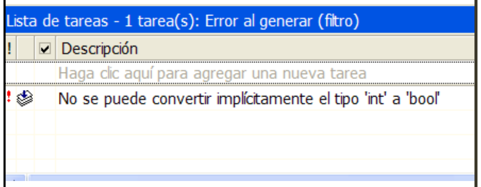
- if else if

```
if (saldo > reintegro)
  Console.WriteLine("OK");
else if (saldo < reintegro)
  Console.WriteLine("No OK");
 else
  Console.WriteLine("a 0");
```

Simple conditional(II)

How will it work?.

```
if (saldo = reintegro)
  Console.WriteLine(“a 0");
else if (saldo < reintegro)
  Console.WriteLine("No OK");
 else
  Console.WriteLine(“OK");
```


Lista de tareas - 1 tarea(s): Error al generar (filtro)

```
! ☑ Descripción
    Haga clic aquí para agregar una nueva tarea
!   No se puede convertir implícitamente el tipo 'int' a 'bool'
```

SOLUCIÓN

```
if (saldo == reintegro)
  Console.WriteLine(“a 0");
else if (saldo < reintegro)
  Console.WriteLine("No OK");
 else
  Console.WriteLine(“OK")
```

Multiple conditional

```
switch (saldo)
   {
   case 1 : saldo *= 2;
        break;
   case 2 : saldo *= 3;
        break;
   default : saldo = 0;
            break;

   }
```

BREAK. Almost mandatory.

Multiple conditional(II)

```
switch (saldo) {
   case 1 : saldo *= 2;

   case 2 : saldo *= 3;
        break;
   default : saldo = 0;
            break;

   }
```

Error

```
SOLUCIÓN
switch (saldo) {
   case 1 : saldo *= 2;
        goto case 2;
   case 2 : saldo *= 3;
        break;
   default : saldo = 0;
            break;

   }
```

Multiple conditional(III)

```
switch (saldo) {
   case 1 :

   case 2 : saldo *= 3;
        break;
   default : saldo = 0;
            break;

   }
```

Correct

Multiple conditional(IV)

Allows using labels of type string

```
string nombre= "Pedro";
switch (nombre)
   {
   case "Pedro" : Console.WriteLine("Hola Pedro");
        break;
   case "Juan" : Console.WriteLine("Hola Juan");
        break;
    }
```

Repetition structures

- for

```
for (i = 0; i <= 10 ; i++)
        suma+= i;
```

- while

```
while ( i <= 10)
   {
   suma+= i;
   i++;
   }
```

- do

```
i = 0;
do
  {
  suma += i;
  i++;
  }
  while (i < 10);
```

Repetition structures(II)

- foreach

```
int [ ] mivector  = {1,2,3};
foreach(int algo in mivector)
        Console.Write (algo);
```

It allows going trough all the elements of a collection

It is needed to define a variable of the type of the elements of the collection

Change of sequence:return

- Return gives the control back to the calling code

```
int number, addition;
        Console.Write(“Write a number : ");
        number = int.Parse(Console.ReadLine());
        addition = 0;
        while (number <= 10)
        {
           if (number == 3)
              return;
           addition += number;
           number++;
        }

        Console.WriteLine(“The value is " + addition);
```

What does this program do???

Change of sequence:break

- Break goes out from the current structure of the loop

```
int number, addition;
        Console.Write(“Write a number : ");
        number = int.Parse(Console.ReadLine());
        addition = 0;
        while (number <= 10)
        {
           if (number == 3)
              break;
           addition+= number;
           number++;
        }

        Console.WriteLine(“The value is " + addition);
```

What does this program do???

Change of sequence:continue

- Continue: Obligates to execute the next loop iteration

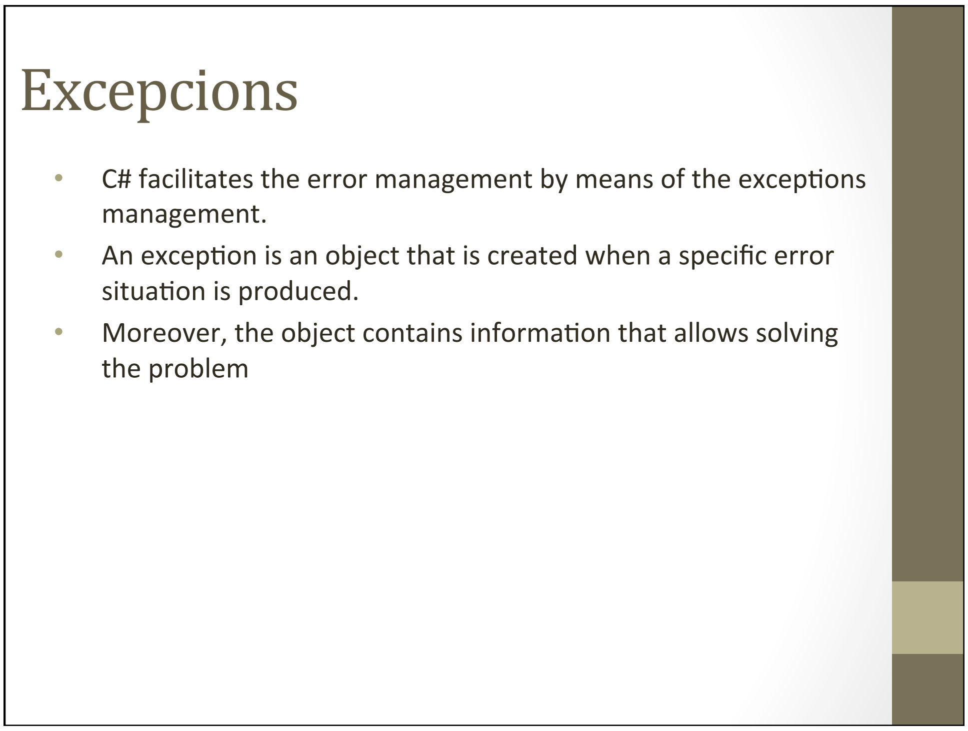
```
int number, addition;
        Console.Write(“Write a number : ");
        number = int.Parse(Console.ReadLine());
        addition = 0;
        while (number <= 10)
        {
           number++;
           if (number== 3)
                 continue;
            addition += number;
}

        Console.WriteLine(“The value is " + addition);
```

What does this program do???

Programming in C#

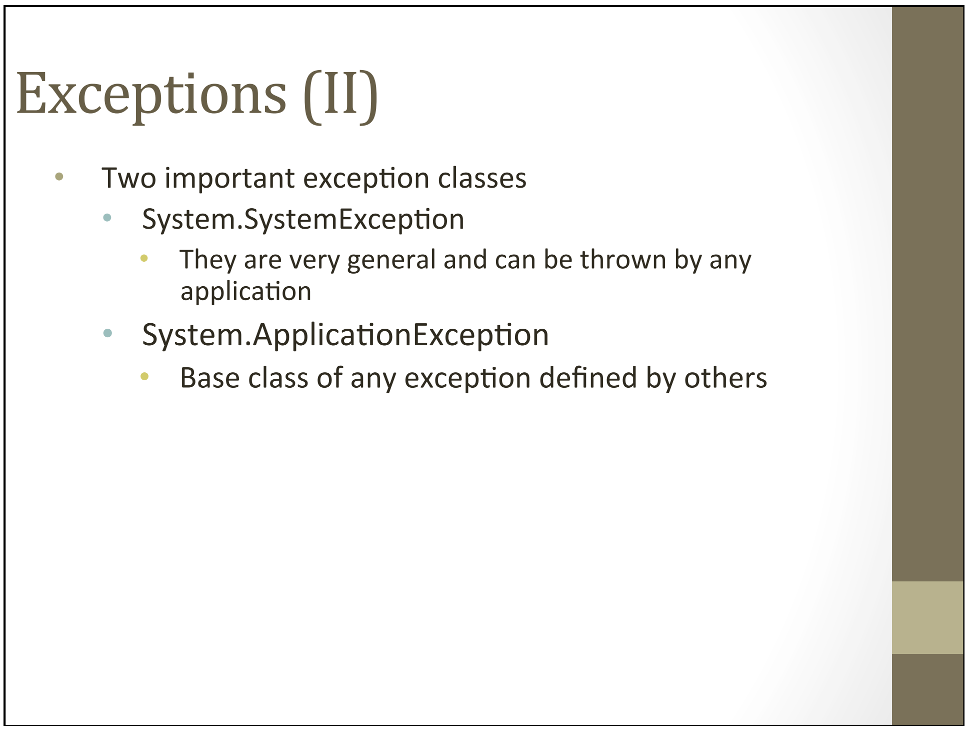
7

Exceptions



Excepcions

- C# facilitates the error management by means of the exceptions management.
- An exception is an object that is created when a specific error situation is produced.
- Moreover, the object contains information that allows solving the problem



Exceptions (II)

- Two important exception classes
  - System.SystemException
    - They are very general and can be thrown by any application
  - System.ApplicationException
    - Base class of any exception defined by others

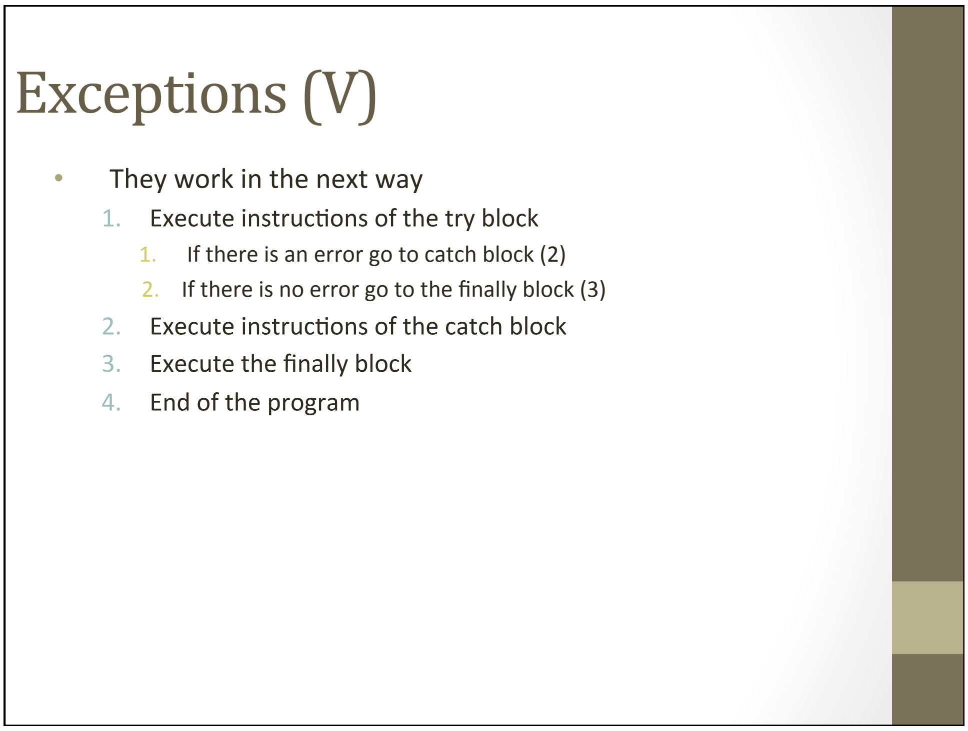
Exceptions (III)

- We define the following structure

```
try
{
 // normal execution code
}
Catch
{
// Errors management
}
Finally
{
// Resources release
}
```

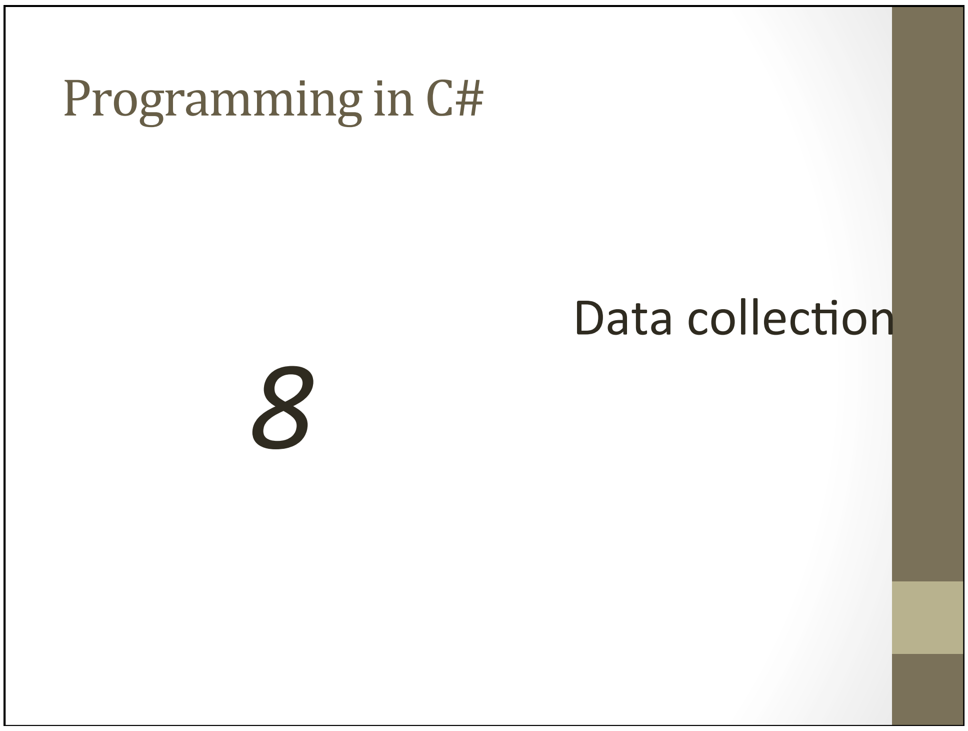
Exceptions (IV)

- The try block contains the code of normal behaviour of the program
- The catch block contains the code that manages the errors that can happen in the 'normal' code
- The finally block contains the code to release the resources. It is an optional block.



Exceptions (V)

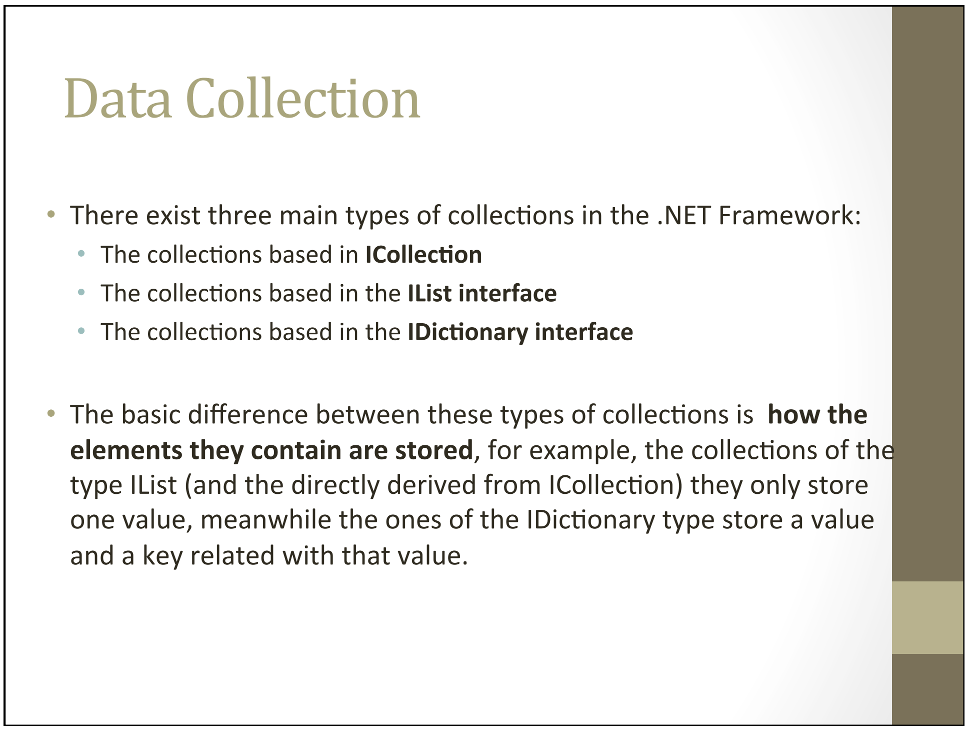
- They work in the next way
  1. Execute instructions of the try block
     1. If there is an error go to catch block (2)
     2. If there is no error go to the finally block (3)
  2. Execute instructions of the catch block
  3. Execute the finally block
  4. End of the program



Programming in C#

8

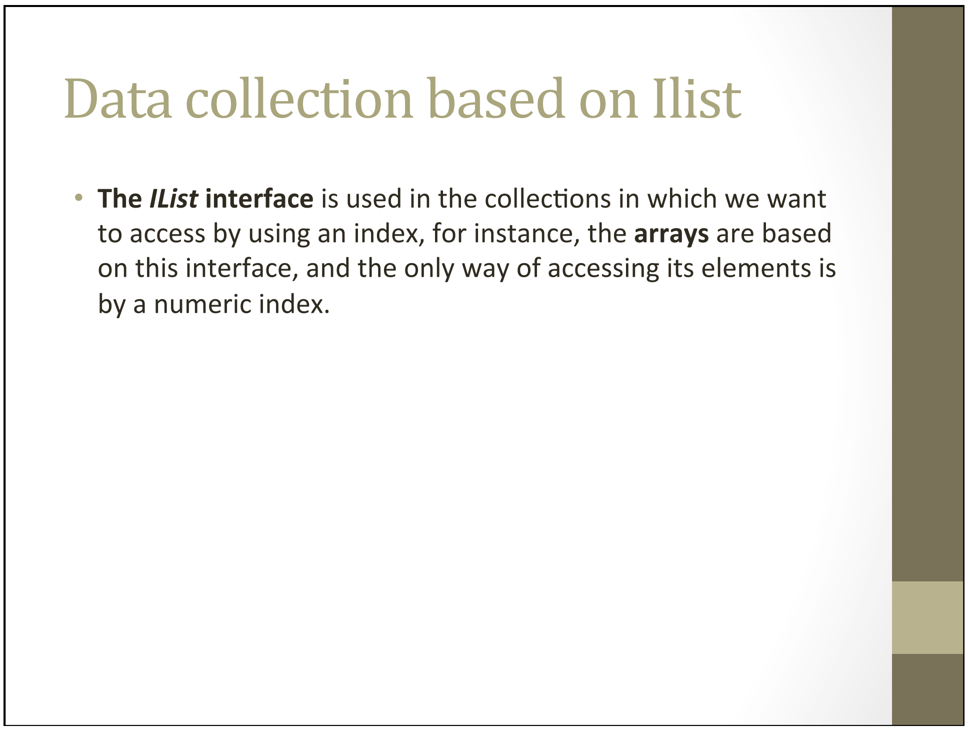
Data collection



Data Collection

- There exist three main types of collections in the .NET Framework:
  - The collections based in **ICollection**
  - The collections based in the **IList interface**
  - The collections based in the **IDictionary interface**

- The basic difference between these types of collections is **how the elements they contain are stored**, for example, the collections of the type IList (and the directly derived from ICollection) they only store one value, meanwhile the ones of the IDictionary type store a value and a key related with that value.



Data collection based on Ilist

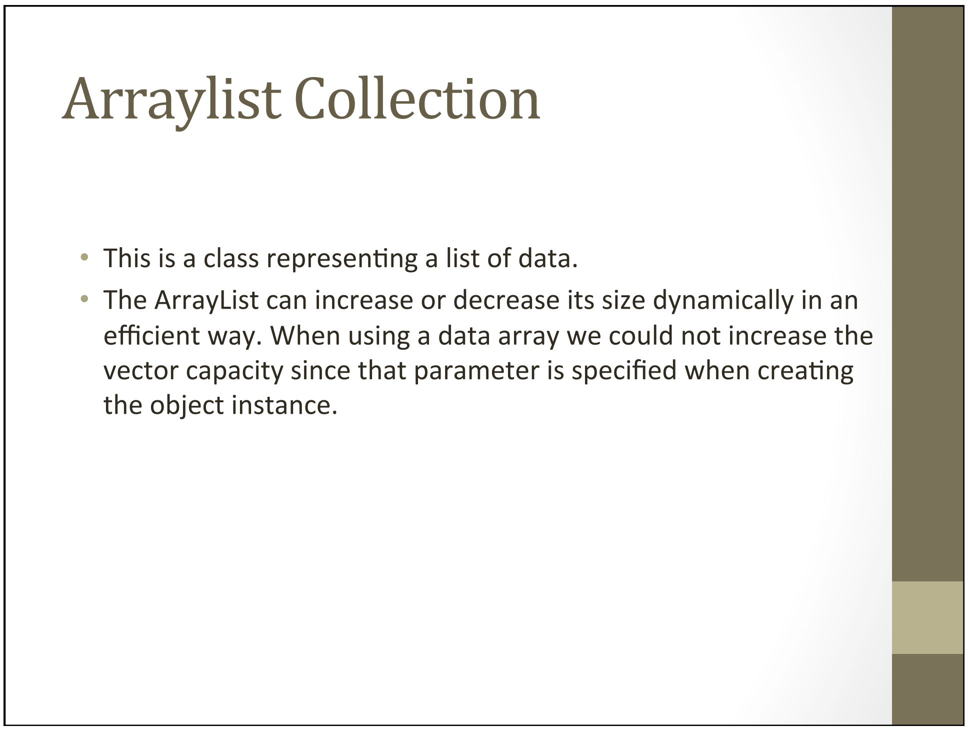
- **The *IList* interface** is used in the collections in which we want to access by using an index, for instance, the **arrays** are based on this interface, and the only way of accessing its elements is by a numeric index.

Data collection based on Ilist

- There exist three main types of collections implementing this interface:
  - **Read-only**, collections that cannot be modified. This type of collections are usually based on the abstract class*ReadOnlyCollectionBase*.
  - **Fixed size collections**, we cannot add or remove elements from them, but we can modify them. For instance, collections based on Array are fixed size collections.
  - **Variable size collections,** they allow any type of addition, removal and modification. Most of the collections belong to this type, they allow us to dynamically add or remove elements.

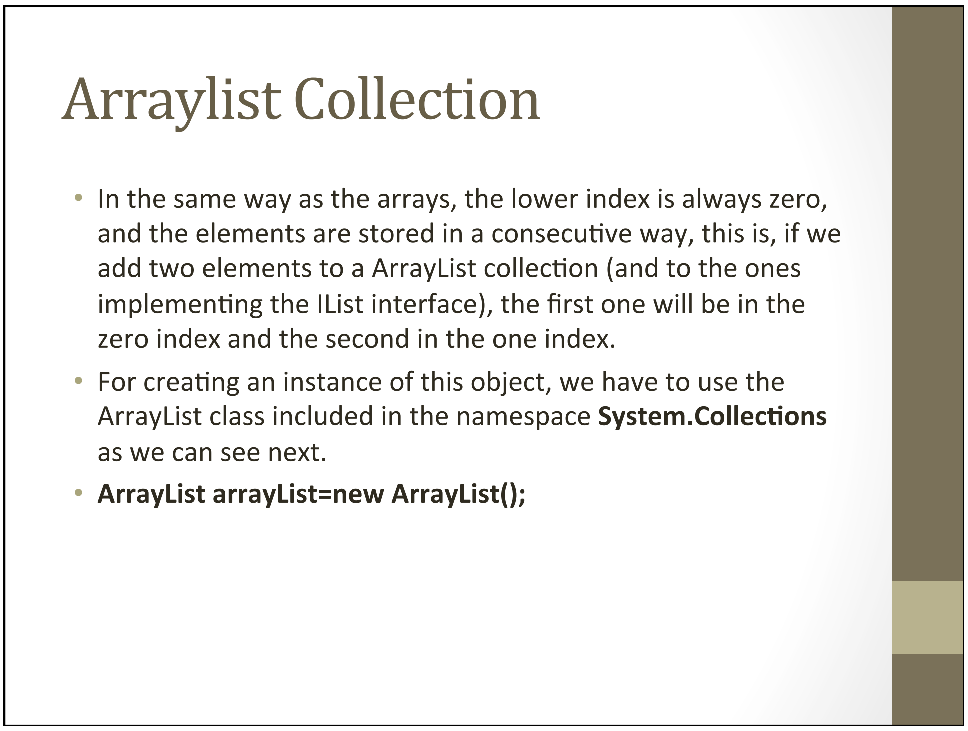
Data collection based on Ilist

- There exist a big number of collections in .NET implementing this interface, (many of them are collections based on controls), we can stress the following:
- ***ArrayList*,** the “classic” collection for this interface type. It contains all the usual members in this type of collections.
- ***CollectionBase***, an abstract class for creating our own collections based on *IList*.
- ***StringCollection***, a specialized collection that can only contain string values.



Arraylist Collection

- This is a class representing a list of data.
- The ArrayList can increase or decrease its size dynamically in an efficient way. When using a data array we could not increase the vector capacity since that parameter is specified when creating the object instance.



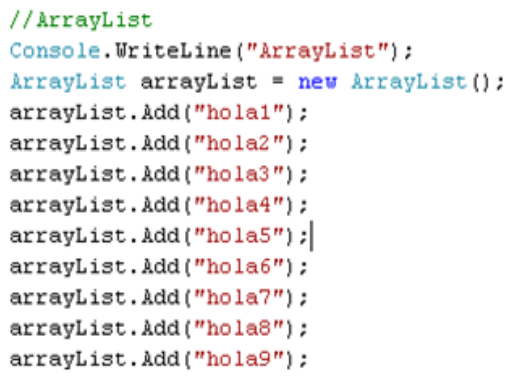
Arraylist Collection

- In the same way as the arrays, the lower index is always zero, and the elements are stored in a consecutive way, this is, if we add two elements to a ArrayList collection (and to the ones implementing the IList interface), the first one will be in the zero index and the second in the one index.
- For creating an instance of this object, we have to use the ArrayList class included in the namespace **System.Collections** as we can see next.
- **ArrayList arrayList=new ArrayList();**

Arraylist Collection

- The constructor of the ArrayList class also accepts an integer parameter that indicates the initial capacity of the object created.
- If is neeed adding a new object to the collection, we have to use the **Add** method**,** that inserts the new element in the last position, or the **Insert** method that inserts it in the indicated position.

Arraylist Collection



```
//ArrayList
Console.WriteLine("ArrayList");
ArrayList arrayList = new ArrayList();
arrayList.Add("hola1");
arrayList.Add("hola2");
arrayList.Add("hola3");
arrayList.Add("hola4");
arrayList.Add("hola5");
arrayList.Add("hola6");
arrayList.Add("hola7");
arrayList.Add("hola8");
arrayList.Add("hola9");
```

Arraylist Collection

- **All the objects stored in an Arraylist are treated as objects**, therefore it is possible to add any type of data, this means, we can add integers, strings, objects of our defined classes, etc.
- Moreover, on the opposite way from the arrays, **not all the elements have to be of the same data type**. In some cases this can be an advantage, because it allows us storing big variety of information in a single collection, however, for being efficient (cast, boxing, unboxing; convert a type by value to one by reference when we are going to store it in the collection (boxing), and the inverse process when we want to get it (unboxing)), there are cases in which is better using the generic collections.

Arraylist Collection

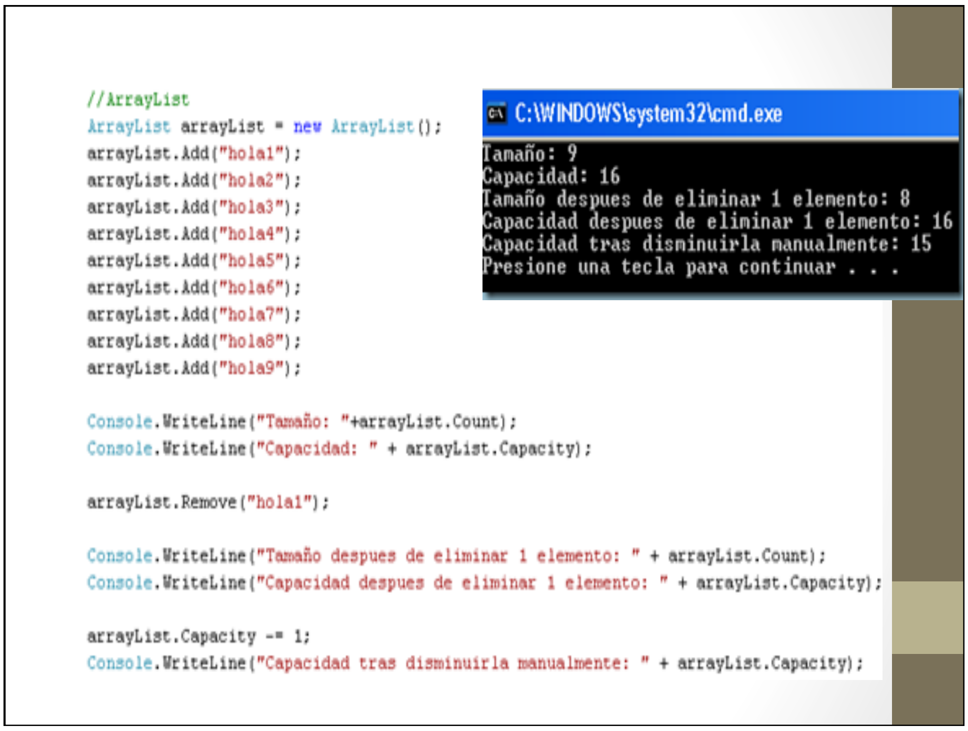
- If we need to remove elements from the collection, we can use the methods: **remove, removeAt or RemoveRange,** which remove the object passed as a parameter, or in a certain index, or a group of elements, respectively.

Arraylist Collection

- The most used properties of this collection are : **Count and Capacity**.
- The first one is used for knowing the current amount of elements of the collection.
- The second one indicates the maximum capacity of the collection for storing elements. It is necessary knowing that the capacity of the collection will be automatically increased if needed when inserting an element.

Arraylist Collection

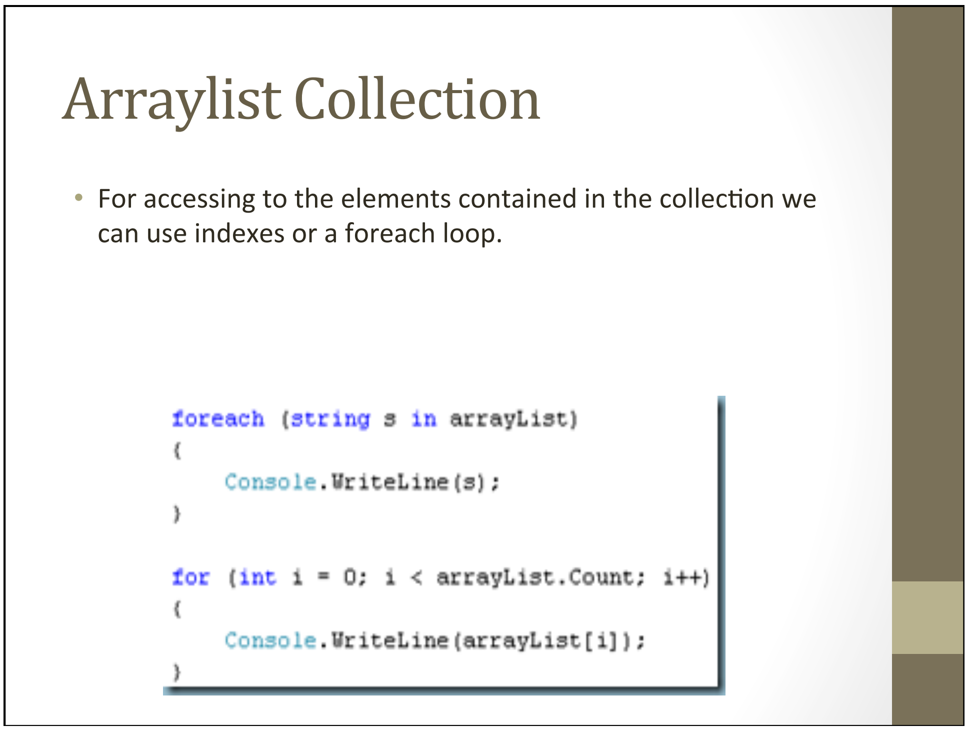
- The capacity of a collection can never be less than the total amount of elements contained, so if we manually modify the Capacity property and assign it a value less than the value returned by the Count property, we will get an exception of the type **ArgumentOutOfRangeException .**



```
//ArrayList
ArrayList arrayList = new ArrayList();
arrayList.Add("hola1");
arrayList.Add("hola2");
arrayList.Add("hola3");
arrayList.Add("hola4");
arrayList.Add("hola5");
arrayList.Add("hola6");
arrayList.Add("hola7");
arrayList.Add("hola8");
arrayList.Add("hola9");

Console.WriteLine("Tamaño: "+arrayList.Count);
Console.WriteLine("Capacidad: " + arrayList.Capacity);

arrayList.Remove("hola1");

Console.WriteLine("Tamaño despues de eliminar 1 elemento: " + arrayList.Count);
Console.WriteLine("Capacidad despues de eliminar 1 elemento: " + arrayList.Capacity);

arrayList.Capacity -= 1;
Console.WriteLine("Capacidad tras disminuirla manualmente: " + arrayList.Capacity);
```

C:\WINDOWS\system32\cmd.exe

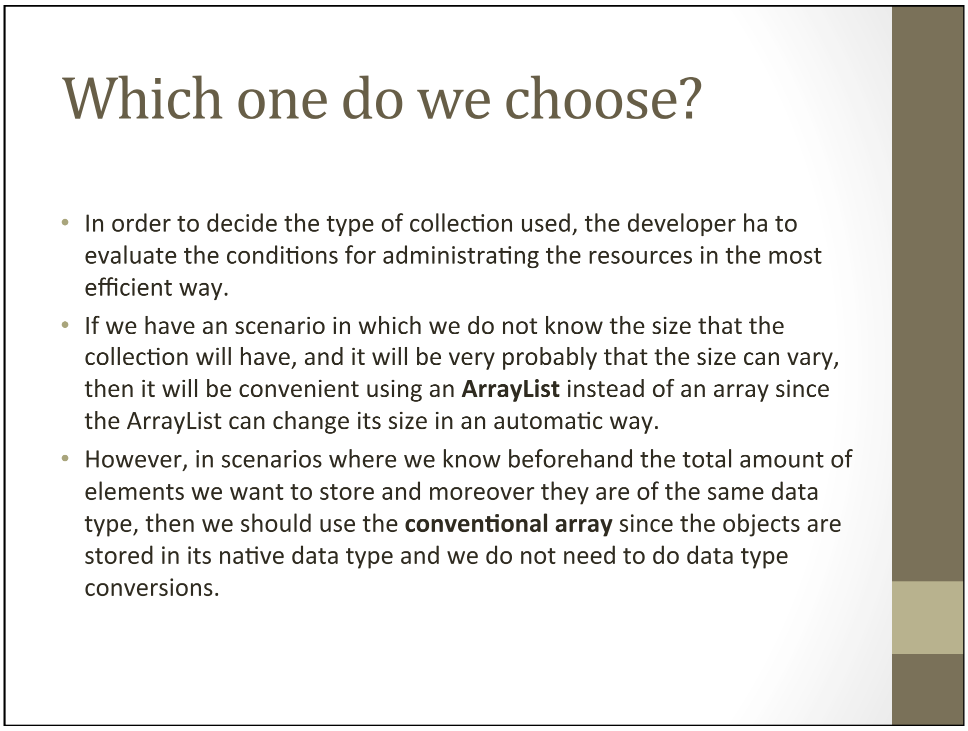
```
Tamaño: 9
Capacidad: 16
Tamaño despues de eliminar 1 elemento: 8
Capacidad despues de eliminar 1 elemento: 16
Capacidad tras disminuirla manualmente: 15
Presione una tecla para continuar . . .
```



Arraylist Collection

- For accessing to the elements contained in the collection we can use indexes or a foreach loop.

```
foreach (string s in arrayList)
{
    Console.WriteLine(s);
}

for (int i = 0; i < arrayList.Count; i++)
{
    Console.WriteLine(arrayList[i]);
}
```



Which one do we choose?

- In order to decide the type of collection used, the developer ha to evaluate the conditions for administrating the resources in the most efficient way.
- If we have an scenario in which we do not know the size that the collection will have, and it will be very probably that the size can vary, then it will be convenient using an **ArrayList** instead of an array since the ArrayList can change its size in an automatic way.
- However, in scenarios where we know beforehand the total amount of elements we want to store and moreover they are of the same data type, then we should use the **conventional array** since the objects are stored in its native data type and we do not need to do data type conversions.