

Departament de Llenguatges i Sistemes Informàtics

Tema 7. Plataforma .net

Primers programes amb C#

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Escola Politècnica Superior
Universitat d'Alacant

Programació en C#. Parteix I: Primers programes amb C#

Objectius

1. Introducció a la plataforma .NET
2. Conèixer els orígens de C#
3. Crear una aplicació de consola
4. Conèixer l'estructura bàsica d'un programa C#
5. Conèixer la gestió d'excepcions
6. Conèixer les col·leccions de dades de llist

INDICE

1. Què és .net?
2. Arquitectura .net
3. Arquitectura .net Framework
4. Introducció a C#
5. El meu primer programa amb C#
6. Aspectes bàsics del llenguatge
7. Gestió d'excepcions
8. Col·leccions de dades

Introducció a .Net

1

Què és .Net?

Què és .net?

- .net és una plataforma que permet el desenvolupament d'aplicacions programari i llibreries.
- .net conté el compilador i les eines necessàries per construir, depurar i executar aquestes aplicacions.

Què és .net? (II)

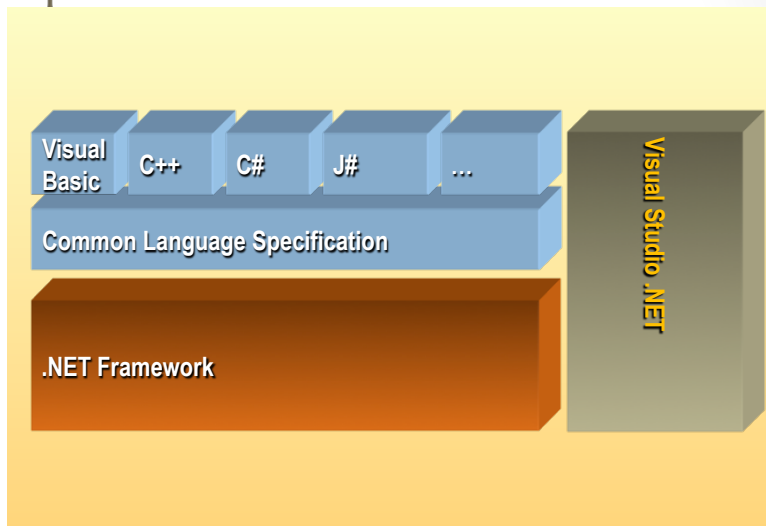
- .net és una plataforma programari
- És un entorn de desenvolupament independent del llenguatge, que permet escriure programes de forma senzilla, i fins i tot permet combinar codi escrit en diferents llenguatges.
- No està orientat a un Maquinari/Sistema Operatiu concret, sinó a qualsevol plataforma per la qual .net estigui desenvolupat.

Introducció a .Net

2

Arquitectura .Net?

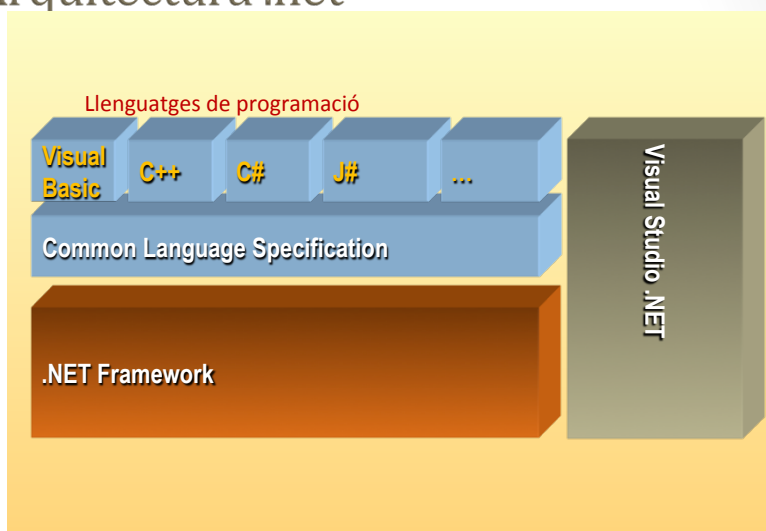
Arquitectura .net



Visual Studio .net

- **Visual Studio .NET** ofereix un entorn de desenvolupament per desenvolupar aplicacions que s'executen sobre el .NET Framework.
- Proporciona les tecnologies fonamentals per simplificar la creació i implantació de
 - Aplicacions i Serveis Web
 - Aplicacions basades en Windows

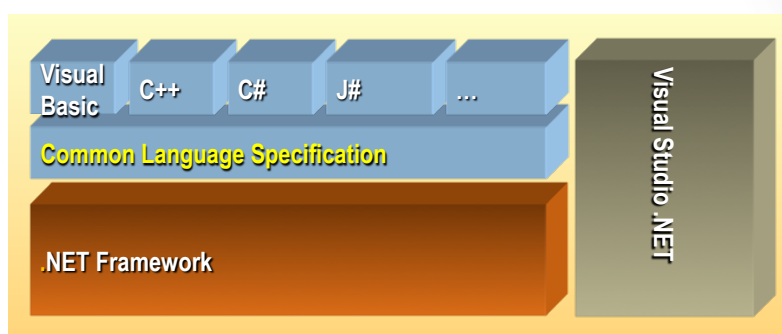
Arquitectura .net



Llenguatges de programació

- **C#**
 - C# ha estat dissenyat específicament per a la plataforma .NET i és el primer llenguatge modern orientat a components de la família de C i C++.
 - Pot incrustar-se en pàgines ASP.NET.
 - Algunes de les principals característiques d'aquest llenguatge inclouen classes, interfícies, delegats, espais de noms, propietats, indexadores, esdeveniments, sobrecàrrega d'operadors, versionado, atributs, codi insegur, creació de documentació en format XML
- Visual Basic .net
- C++
- J#
- Llenguatges de tercers

CLS

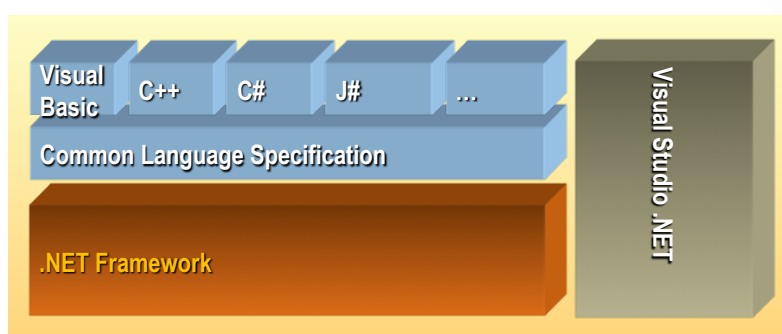


- L'especificació **Common Language Specification (CLS)** defineix els mínims estàndards que han de satisfer els llenguatges i desenvolupadors si desitgen que els seus components i aplicacions siguin àmpliament utilitzats per altres llenguatges compatibles amb .NET.

Common Language Specification (II)

- L'especificació **CLS** permet als desenvolupadors de .NET crear aplicacions com a part d'un equip que utilitza múltiples llenguatges amb la seguretat que no hi haurà problemes amb la integració dels diferents llenguatges.
- L'especificació **CLS** també permet als desenvolupadors de .NET heretar de classes desenvolupades en llenguatges diferents.

.net framework



- És el motor d'execució
- Proporciona un conjunt de serveis comuns per als projectes generats en .net, amb independència del llenguatge.

.net Framework (II)

- **Extensible**

- La jerarquia del .NET Framework no queda oculta al desenvolupador. Podem accedir i estendre classes .NET (tret que estiguin segellades) utilitzant herència. També podem implementar herència multilenguaje.

- **Fàcil d'usar pels desenvolupadors**

- En el .NET Framework, el codi està organitzat en espais de noms jeràrquics i classes. El Framework proporciona un sistema de tipus comú, denominat sistema de tipus unificat, que utilitza qualsevol llenguatge compatible amb .NET. En el sistema de tipus unificat, tot és un objecte.

Introducció a .Net

3

Arquitectura
.net
Framework

Parts del .NET Framework.

El **Common Language Runtime (CLR)** és el nucli de la plataforma .NET. És el motor encarregat de gestionar l'execució de les aplicacions per a ella desenvolupades i a les quals ofereix nombrosos serveis que simplifiquen el seu desenvolupament i afavoreixen la seva fiabilitat i seguretat.

Common Language Runtime (CLR)

- **Execució multiplataforma:** El CLR actua com una màquina virtual, encarregant-se d'executar les aplicacions dissenyades per a la plataforma .NET. És a dir, qualsevol plataforma per la qual existeixi una versió del CLR podrà executar qualsevol aplicació .NET.

Fins ara solament s'han desenvolupats CLR para totes les versions de Windows, existeix la possibilitat de desenvolupar una versió per a sistemes com Unix o Linux a causa que l'arquitectura del CLR és oberta.

Projecto Mono: <http://go-mono.com>



Common Language Runtime (CLR)

- **Integració de llenguatges:**

Des de qualsevol llenguatge pel qual existeixi un compilador que generi codi per a la plataforma .NET és possible utilitzar codi generat per a la mateixa usant qualsevol altre llenguatge tal com si de codi escrit usant el primer es tractés.

La integració de llenguatges proveeix que és possible escriure una classe en C# que hereti d'una altra escrita en Visual BASIC.NET que, al seu torn, hereti d'una altra escrita en C++ *amb extensions gestionades*.

Microsoft Intermediate Language (MSIL)

- Els compiladors que generen codi per a la plataforma .NET generen codi escrit en el llenguatge intermedi conegut com **Microsoft Intermediate Language (MSIL)**

Common Language Runtime (CLR)

- **Gestió de memòria:**

El CLR inclou un **recol·lector d'escombraries** que evita que el programador hagi de tenir en compte quan ha de destruir els objectes que deixin de ser-li útils.

Gràcies a aquest recol·lector s'eviten errors de programació molt comunes com:

- Intents d'esborrat d'objectes ja esborrats.
- Esgotament de memòria per oblit d'eliminació d'objectes inútils o
- Sol·licitud d'accés a membres d'objectes ja destruïts.

Common Language Runtime (CLR)

- **Seguretat de tipus:** El CLR facilita la detecció d'errors de programació difícils de localitzar comprovant que tota conversió de tipus que es realitzi durant l'execució d'una aplicació .NET es faci de manera que els tipus origen i destinació siguin compatibles.

Common Language Runtime (CLR)

- **Tractament d'excepcions:** En el CLR tots els errors que es puguin produir durant l'execució d'una aplicació es propaguen d'igual manera: mitjançant *excepcions*.

Biblioteca de classes



System	System.Security	System.Runtime.InteropServices
System.Net	System.Text	System.Globalization
System.Reflection	System.Threading	System.Configuration
System.IO	System.Diagnostics	System.Collections

Biblioteca de classes

- La biblioteca de classes del .NET Framework exposa característiques de l'entorn d'execució i proporciona en una jerarquia d'objectes altres serveis d'alt nivell que tot programador necessita. Aquesta jerarquia d'objectes es denomina **espai de noms**.
- La biblioteca de classes del .NET Framework proporciona nombroses i potents característiques noves per als desenvolupadors
 - Per exemple, l'espai de noms Collections afegeix nombroses possibilitats noves, com a classificació, cues, piles i matrius de grandària automàtica.
 - La classe de sistema Threading també ofereix noves possibilitats per crear veritables aplicacions multi-fil.

Biblioteca de classes (II)

- **Espais de noms System**
 - L'espai de noms System conté classes fonamentals i classes bàsiques que defineixen tipus de dades valor i referència comunament utilitzats, esdeveniments i descriptors d'esdeveniments, interfícies, atributs i processament d'excepcions.

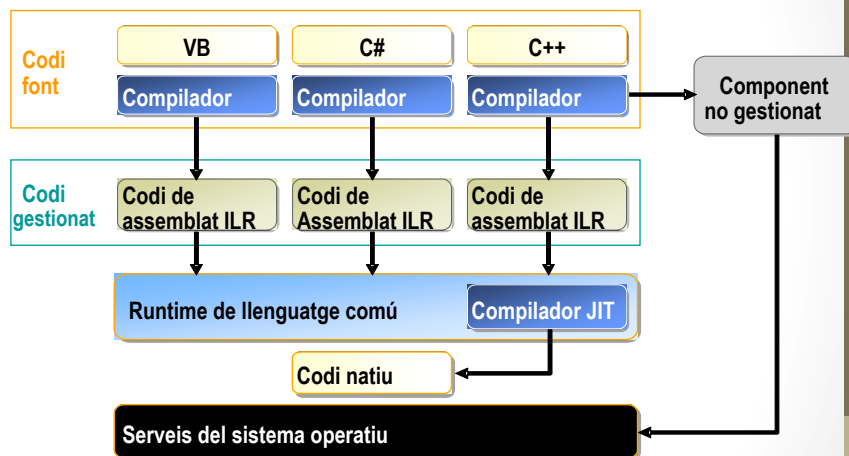
Espai de noms	Utilitat dels tipus de dades que conté
System	Tipus molt freqüentment usats, com els tipus bàsics, taules, excepcions, dates, nombres aleatoris, recol·lector d'escombraries, entrada/sortida en consola, etc.
System.Collections	Col·leccions de dades d'ús comú com a piles, cues, llistes, diccionaris, etc.
System.Data	Manipulació de bases de dades. Formen la denominada arquitectura ADO.NET.
System.IO	Manipulació de fitxers i altres fluxos de dades.
System.Net	Realització de comunicacions en xarxa.
System.Reflection	Accés a les metadades que acompanyen als mòduls de codi.
System.Runtime.Remoting	Accés a objectes remots.
System.Security	Accés a la política de seguretat en què es basa el CLR.
System.Threading	Manipulació de fils.
System.Web.UI.WebControls	Creació d'interfícies d'usuari basades en finestres per a aplicacions Web.
System.Windows.Forms	Creació d'interfícies d'usuari basades en finestres per a aplicacions estàndard.
System.Xml	Accés a dades en format XML.

Introducció a .Net

4

El model d'execució

El model d'execució



Programació en C#

4

Introducció a C#

Introduccion a C#

- C#
 - Llenguatge dissenyat específicament para .NET
 - Dissenyat des de zero sense cap condicionament
 - Microsoft ho descriu com
 - Senzill
 - Modern
 - Orientat a objectes
 - Segur quant a tipus
 - Derivat de C i C++ (i a JAVA encara que Microsoft no ho digui)

Introducció a C# (II)

- Avantatges
 - Integrat amb modernes eines de desenvolupament
 - Fàcil d'integrar amb Visual Basic
 - Té l'alt rendiment i permet l'accés a memòria de baix nivell de C++

Programació en C#

5

El meu primer programa amb C#

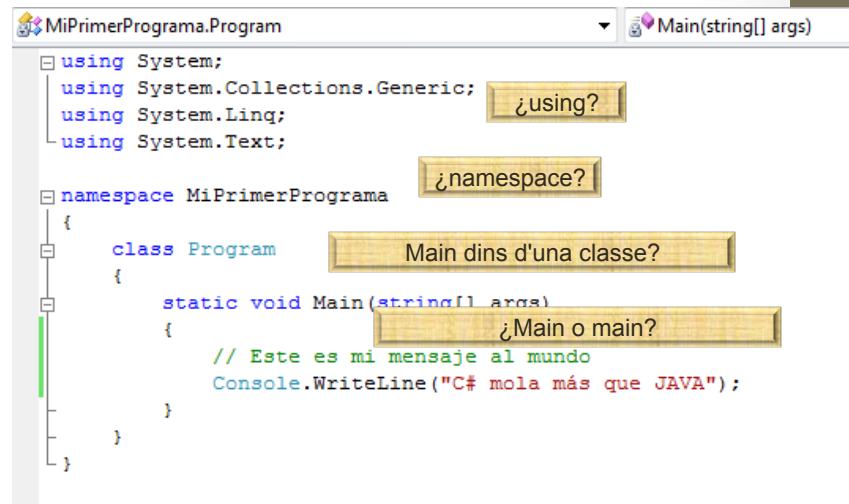


Exercici 1

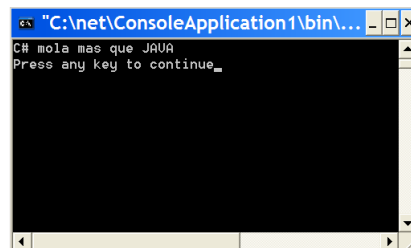
Primer Programa

- Escriure un programa en C# que mostri un missatge en pantalla.

Demo : MiPrimerPrograma



Demo : MiPrimerPrograma



El programa principal : *Main*

- En escriure Main cal:
 - Utilitzar una “M” majúscula, com en “Main”
 - Designar un **Main** com el punt d'entrada al programa
 - Declarar **Main** com **static void Main**
- Un Main pot pertànyer a múltiples classes
 - En aquest cas s'ha d'especificar com és el primer punt d'entrada
- L'aplicació acaba quan Main acaba o executa un return

La classe: *class*

- Una aplicació C# és una col·lecció de classes, estructures i tipus
- Una classe és un conjunt de dades i mètodes
- Una aplicació C# pot incloure moltes classes
- Sintaxi :

```
class nombre
{
    . . .
}
```

Els espais de noms :*namespace*

- .NET Framework ofereix moltes classes d'utilitat
 - Organitzades en espais de noms (namespace)
- System és l'espai de noms més utilitzat
- Es fa referència a classes pel seu espai de noms
- La sentència using

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

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

Els espais de noms :*using*

- La paraula using referència els espais de noms utilitzats.
- En cas de no referenciar un espai de noms, en utilitzar els seus mètodes s'ha d'especificar el path complet.
- La sentència using

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

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

Els comentaris

- Els comentaris són importants
 - Una aplicació amb els comentaris adequats permet a un desenvolupador comprendre perfectament l'estructura de l'aplicació
- Comentaris d'una sola línia

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

```
/* Trobar la major arrel
de l'equació quadràtica /
x = (...);
```

Els comentaris (II)

- ▣ **///** Comentaris que permeten generar documentació automàtica del projecte.

```
namespace ConsoleApplication11
{
    /// <summary>
    /// La classe 1 permet mostrar missatges
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punt d'entrada principal de la aplicaciOn.
        /// </summary>
```

Programació en C#

6

Aspectes bàsics
del llenguatge

Programació en C#

6.1

Sistema de Tipus
Comuns
(CTS)

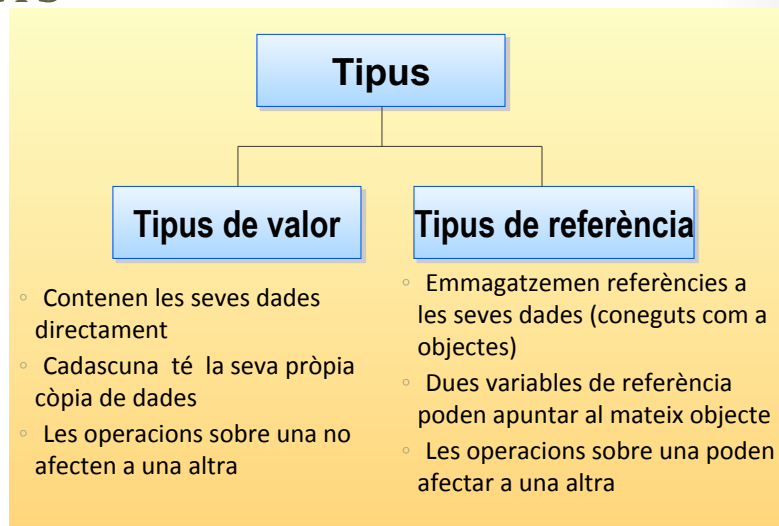
CTS

- ▶ Cada variable té un tipus de dades que determina els valors que es poden emmagatzemar en ella.
- ▶ C# és un llenguatge d'especificacions segures (type-safe), la qual cosa significa que el compilador de C# garanteix que els valors emmagatzemats en variables són sempre del tipus adequat.
- ▶ El runtime de llenguatge comú inclou un sistema de tipus comuns (Common Type System, CTS) que defineix un conjunt de tipus de dades predefinides que es poden utilitzar per definir variables.

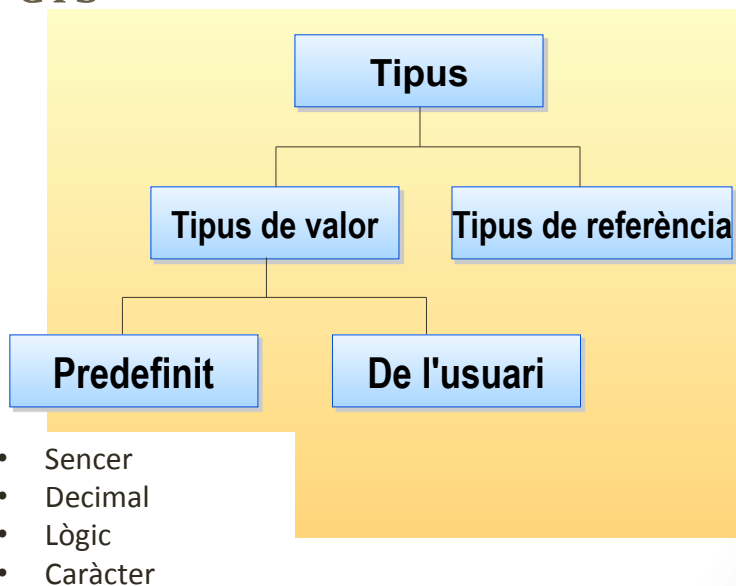
CTS (II)

- ▶ En definir una variable és necessari triar el tipus de dades correcte per a ella. El tipus de dades determina els valors permesos per a aquesta variable, els quals al seu torn determinen les operacions que es poden efectuar sobre ella.
- ▶ El CTS és una part integral del runtime de llenguatge comú i és compartit pels compiladors, les eines i el propi runtime.
- ▶ És el model que defineix les regles que segueix el runtime a l'hora de declarar, usar i gestionar tipus.
- ▶ El CTS estableix un marc que permet la integració entre llenguatges, la seguretat de tipus i l'execució de codi amb altes prestacions.

CTS



CTS



Sencer

sbyte	SByte	8 bits amb signe
short	Int16	16 bits con signo
int	Int32	32 bits amb signe
long	Int64	64 bits amb signe
byte	Byte	8 bits sense signe
ushort	UInt16	16 bits sense signe
uint	UInt32	32 bits sense signe
ulong	UInt64	64 bits sense signe

El tipus byte no és equivalent al char.

Decimals

float	Single	32 bits. Precisió simple (7 dígits significatius)
double	Double	64 bits. Precisió doble (15 dígits significatius)
decimal	Decimal	Alta precisió (28 dígits significatius)

Per defecte un nombre no sencer és double

Explícitament
34.5F (float)
34.5M (decimal)

Lògic

bool	Bool	<i>true o false</i>
------	------	---------------------

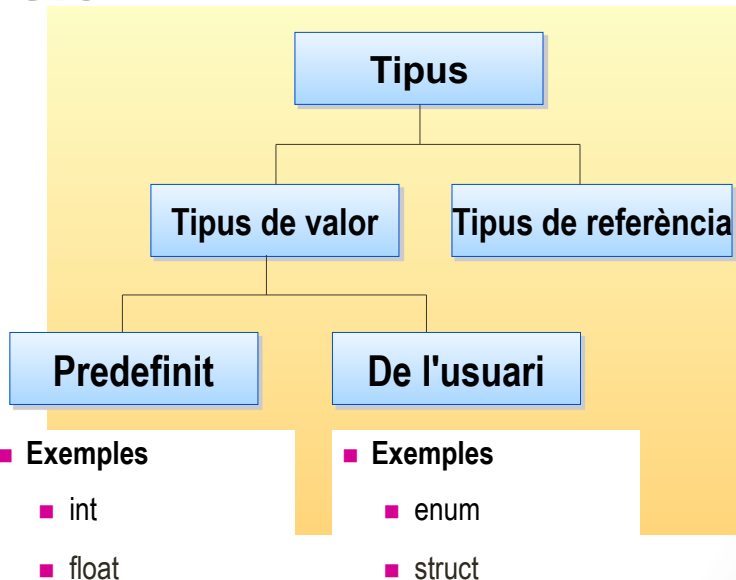
No es poden convertir tipus bool
a sencer o viceversa.

Caràcter

char	Char	<i>Caràcter Unicode de 16 bits</i>
------	------	--

Valores Unicode '\u0041'
Valores hexadecimales '\x0041'

CTS



Tipus enumerats definits per l'usuari :enum

```
enum Colores {amarillo, azul, rojo};
```

Definició

```
static void Main(string[] args)
```

```
{
```

```
Colores colorPantalon = Colores.amarillo;
```

Ús

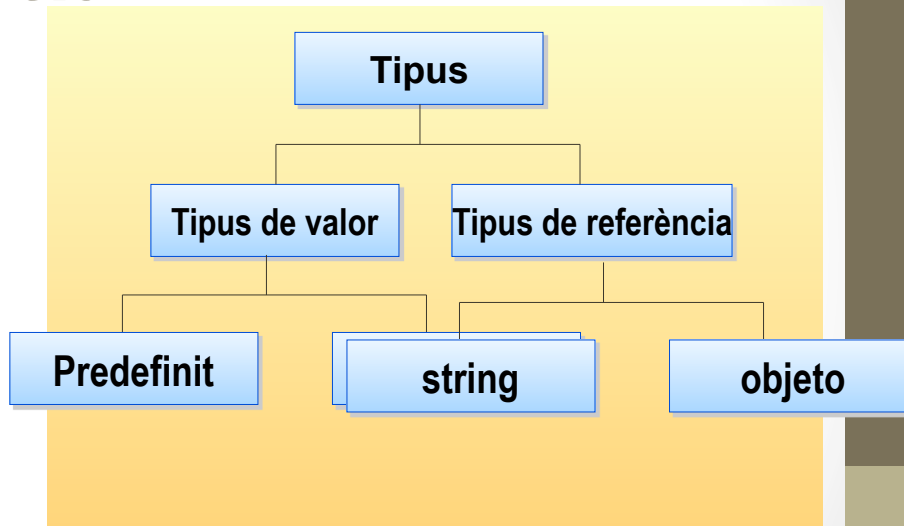
```
Console.WriteLine(colorPantalon);
```

Visualització

```
}
```

```
amarillo
```

CTS



Tipus referencia

- Object
 - És un tipus del que deriven la resta de tipus
- String
 - Facilita la gestió de cadenes
 - Permet operacions del tipus
 - Assignació directa
 - Concatenació (+)

El tipus string

```
static void Main(string[] args){
    string cadena= "Mi primera cadena";
    Console.WriteLine(cadena);
}
```

```
string cadena = "Mi primera cadena";
```

Mi primera cadena

```
string cadena = "Mi primera\ncadena";
```

Mi primera
cadena

```
string cadena = @"Mi primera\ncadena";
```

Mi primera\ncadena

El tipo string (2)

```
static void Main(string[] args){
    string cadena= "Mi primera cadena";
    Console.WriteLine(cadena);
}
```

```
string cadena = "\"Hola\"";
```

"Hola"

```
string cadena = "Hola " + 2;
```

Hola 2

```
string cadena = 2 + "Hola";
```

2Hola

Programació en C#

6.2

Expressions i Operadors

Expressions. Operadors

- Símbols utilitzats en les expressions

Operadors	
<ul style="list-style-type: none"> • Increment / decremento • Aritmètics • Relacionals • Igualtat • Lògics • Assignació 	<pre> ++ -- * / % + - < > <= >= == != && ! = *= /= %= += -= </pre>



Exercici 2

Entrada per teclat

- Escriure un programa en C# que llegeixi dos nombres per teclat.

Demo : Entrada per teclat

```
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.WriteLine("Introduce el primer número: ");
            cadena = Console.ReadLine();
            a = int.Parse(cadena);
            Console.WriteLine("Introduce el segundo número: ");
            cadena = Console.ReadLine();
            b = int.Parse(cadena);
        }
    }
}
```

Definició de variables

Sortida per pantalla

Entrada per teclat i conversió de dades

Programació en C#

6.3

Variables i Constants

Declaració de variables

- Se solen declarar per tipus de dada i nom de variable:

```
int objetoCuenta;
```

- --0--

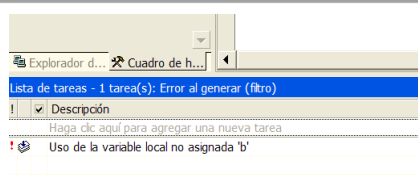
```
int objetoCuenta, empleadoNúmero;
```

```
int objetoCuenta,  
    empleadoNúmero;
```


Declaració de variables (II)

Pregunta. Què ocorre si s'intenta accedir a una variable no inicialitzada?

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



!!!! El compilador C# exigeix que qualsevol variable estigui inicialitzada abans de ser usada

Declaració de variables (III)

- C# és molt més segur que
 - C++ Deixa al programador la tasca de garantir que usa variables inicialitzades. (Els compiladors moderns emeten missatges d'avís però permeten generar l'executable.
 - VB inicialitza a 0 per defecte les variables.

Declaració de variables (IV)

- ▶ Regles
 - Usa lletres, el signe de subratllat i dígit
- ▶ Recomanacions
 - Evita posar totes les lletres en majúscules
 - Evita començar amb un signe de subratllat
 - Evita l'ús d'abreviatures.
 - Usi PascalCasing per a noms amb diverses paraules

Declaració de variables (V)

- ▶ *PascalCasing*
 - ▶ Consisteix a marcar amb majúscules les fronteres de les paraules
 - ▶ EstaEsUnaVariable
 - ▶ La primera lletra és majúscula
- ▶ *CamelCasing* és el mateix però iniciant la paraula en minúscula
 - ▶ estaEsUnaVariable
- ▶ Hi ha autors que recomanen utilitzar PascalCasing variables públiques i camelCasing per a privades.

Declaració de variables (VI)

- Assignar valors a variables ja declarades:

```
int empleadoNumero;  
empleadoNumero = 23;
```

- Inicialitzar una variable quan es declara:

```
int empleadoNumero = 23;
```

- També és possible inicialitzar valors de caràcters:

```
char inicialNombre = 'J';
```

Àmbit de les variables

- L'àmbit d'una variable coincideix amb la seva classe contenidora
- L'àmbit d'una variable finalitza amb la clau que tanca el bloc o mètode en el qual la variable ha estat declarada.
- Les variables poden ser declarades dins d'un bucle, sent sol visibles en el mateix.
 - Això compleix l'estàndard ANSI de C++
 - Les versions anteriors de Microsoft C++ no ho complien

Àmbit de les variables (II)

- El compilador C# exigeix que qualsevol variable estigui inicialitzada abans de ser usada

```

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 ;
            }
        }
    }
}
  
```

The error message box shows:

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

Constants

- Un constant és una variable que no pot canviar el seu valor
- Es defineixen com les variables (amb la paraula *const*)
- S'han d'inicialitzar obligatòriament en la inicialització

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

Constants en C# (II)

- Les constants en C#
 - El valor de la constant ha de ser calculat en temps de compilació
 - (No es pot assignar a partir d'una variable)
 - Això es gestiona amb variables del tipus readonly

Programació en C#

6.4

Entrada/Sortida Per Consola

Entrada/sortida de consola

- Permet accedir a les seqüències estàndard d'entrada, sortida i error
- Només té sentit per a aplicacions de consola
 - Entrada estàndard: teclado
 - Sortida estàndard: Pantalla
 - Error estàndard: Pantalla

Ull: La classe Console solament s'ha d'utilitzar per a aplicacions en línia de comandos. Para aplicacions windows s'ha d'utilitzar l'espai de nom System.Windows.Forms

Entrada/sortida de consola (II)

- Es basen en l'ús de la classe Console
- `Console.Read()`
 - Llegeix un flux d'entrada i ho retorna com *un int*
- `Console.ReadLine()`
 - Llegeix una cadena de text assabenta
- `Console.Write()`
 - Permet escriure en la pantalla
- `Console.WriteLine`
 - Escriu en pantalla afegint caràcter retorno de carro o fi de línia

Entrada/sortida de consola

(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);
}
```

```
C:\windows\system32\cmd.exe
Introduce el primer número: 5
Introduce el segundo número: 6_
```

Paràmetres del writeline (II)

- Cadena que conté entre claus els paràmetres que s'indiquen a continuació separats per comes

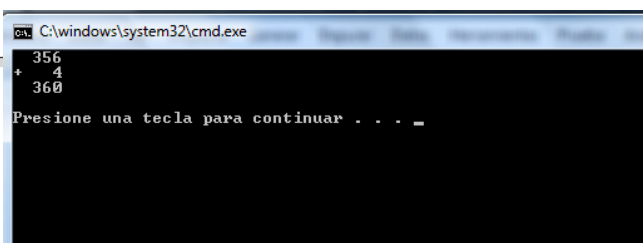
```
static void Main(string[] args)
{
    int a, b;
    a = 3;
    b = 4;
    Console.WriteLine("La suma de {0} y {1} es {2} ",a,b, a+b);
    Console.WriteLine("La suma de " + a + " y " + b + " es " + (a+b));
}
```

```
C:\windows\system32\cmd.exe
La suma de 3 y 4 es 7
La suma de 3 y 4 es 7
Presione una tecla para continuar . . . _
```

Formato de sortida

- La sortida es pot formatar utilitzant el format {n.m} on n és l'índex del paràmetre i m l'ample de la sortida

```
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);
}
```

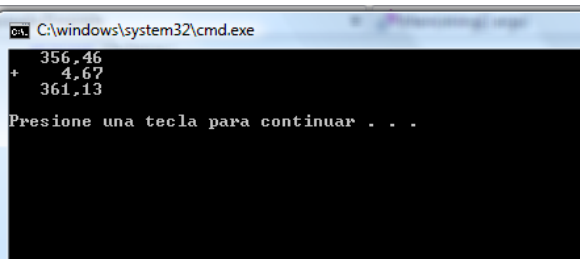


```
C:\windows\system32\cmd.exe
356
+ 4
360
Presione una tecla para continuar . . .
```

Precisió

- Es poden indicar els decimals utilitzant F i un nombre

```
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);
}
```



```
C:\windows\system32\cmd.exe
356.46
+ 4.67
361.13
Presione una tecla para continuar . . .
```




Exercici 3

Suma i Quadrat de dos nombres

- Escriure un programa en C# que llegeixi dos nombres per teclat. Després haurà de mostrar el resultat de la suma de tots dos i el quadrat d'aquesta suma.

Demo : Entrada per teclat

```
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;
}
```

Conversió de dades sense pèrdua d'informació

Conversió de dades amb pèrdua d'informació

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)

Programació en C#

6.5

Conversió de Tipus

Conversió de tipus

- Implicita
 - Automàtica, quan no hi ha possible pèrdua d'informació.

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

Conversió de tipus (II)

- Explícita. S'ha d'indicar que es desitja realitzar una conversió en la qual pugues haver-hi pèrdua d'informació.

```
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'

Conversió de tipus (III)

- Explícita. Correcta utilitzant *càsting*

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

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

Conversió de tipus (IV)

- És correcte el següent programa?

```
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 : Entrada per teclat

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

Conversió de dades

Programació en C#

6.6

Estructures de control

Estructures de control

- Condicional
 - if/else
 - switch/case
- Repetició
 - while
 - do/while
 - for
 - foreach
- Canvi de seqüència
 - return
 - break
 - continue

Condicional Simple

□ 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");
```

Condicional Simple (II)

Com funcionarà?.

```
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'

SOLUCIÓ

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

Condicional múltiple

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

BREAK. Gairebé obligatori.

Condicional múltiple (II)

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

Error

SOLUCIÓ

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

Condicional múltiple (III)

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

Correcte

Condicional múltiple (IV)

Permet utilitzar etiquetes tipus string

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


Estructures de repetició

□ 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);
```

Estructures de repetició (II)

□ foreach

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

Permet recórrer tots els elements d'un contenidor

És necessari definir una variable del tipus dels elements del contenidor

Canvi de seqüència:return

- Return retorna el control a la rutina llamante a l'actual

```
int valor, suma;
Console.Write("Introduce un número : ");
valor = int.Parse(Console.ReadLine());
suma = 0;
while (valor <= 10)
{
    if (valor == 3)
        return;
    suma += valor;
    valor++;
}

Console.WriteLine("El valor es " + suma);
```

Què fa el programa ???

Canvi de seqüència:break

- Break surt de l'actual estructura de bucle

```
int valor, suma;
Console.Write("Introduce un número : ");
valor = int.Parse(Console.ReadLine());
suma = 0;
while (valor <= 10)
{
    if (valor == 3)
        break;
    suma += valor;
    valor++;
}

Console.WriteLine("El valor es " + suma);
```

Què fa el programa ???

Canvi de seqüència:continue

- Continue: Obliga a executar la següent iteració del bucle

```
int valor, suma;
Console.Write("Introduce un número : ");
valor = int.Parse(Console.ReadLine());
suma = 0;
while (valor <= 10)
{
    valor++;
    if (valor == 3)
        continue;

    suma += valor;
}

Console.WriteLine("El valor es " + suma);
```

Què fa el programa ???

Programació en C#

7

Excepcions

Excepcions

- C# ofereix facilitats per a la gestió d'errors per mitjà del maneig d'excepcions.
- Una excepció és un objecte que es crea quan es produeix una situació d'error específica.
- A més l'objecte conté informació que permet resoldre el problema

Excepcions (II)

- Dues classes importants d'excepcions
 - `System.SystemException` Tenen naturalesa molt general i poden ser llançades per qualsevol aplicació.
 - `System.ApplicationException` Classe base de qualsevol classe d'excepció definida per tercers

Excepcions (III)

- La programació d'excepcions es defineix

```
try
{
    // Codigo d'execució normal
}
Catch
{
    // Gestió d'errors
}
Finally
{
    // Alliberament de recursos
}
```

Excepcions (IV)

- ▶ El bloc try conté el codi que forma part del funcionament normal del programa
- ▶ El bloc catch conté el codi que gestiona els diversos errors que es puguin produir
- ▶ El bloque finally contiene el código que libera los recursos. És un camp opcional.

Excepcions (V)

- Les excepcions funcionen de la següent forma
 1. Executar instruccions de bloc try
 1. Si hi ha error anar a bloc catch (2)
 2. Si no hi ha error anar a bloc finally (3)
 2. Executar instruccions del bloc catch
 3. Executar bloc finally
 4. Fi de programa

Programació en C#

8

Col·leccions de dades

Col·leccions de dades

- En .NET Framework existeixen tres tipus principals de col·leccions:
 - Les col·leccions basades en **ICollection**
 - Les col·leccions basades en la **interfície IList**
 - Les col·leccions basades en la **interfície IDictionary**
- La diferència bàsica entre aquests tipus de col·leccions és **com estan emmagatzemats els elements que contenen**, per exemple, les col·leccions de tipus IList (i les directament derivades de ICollection) solament emmagatzemen un valor, mentre que les col·leccions de tipus IDictionary guarden un valor i una clau relacionada amb aquest valor.

Col·leccions basades en IList

- La **interfície IList** s'utilitza en les col·leccions a les quals volem accedir mitjançant un índex, per exemple, els **arrays** realment està basats en aquesta interfície, i l'única forma que tenim d'accedir als elements d'un array, (i per extensió als elements de les col·leccions basades en *IList*), és mitjançant un índex numèric.

Col·leccions basades en llist

- Existeixen tres tipus principals de col·leccions que implementen aquesta interfície:
 - **Les de solament lectura**, col·leccions que no es poden modificar. Aquest tipus de col·leccions solen basar-se a la classe abstracta *ReadOnlyCollectionBase*.
 - **Les col·leccions de grandària fixa**, no es poden llevar ni afegir elements, però sí modificar-los. Per exemple, les col·leccions basades en *Array* són de grandària fixa.
 - **Les de grandària variable** permeten qualsevol tipus d'addició, eliminació i modificació. La majoria de les col·leccions solen ser d'aquest tipus, és a dir, ens permeten dinàmicament afegir o eliminar elements.

Col·leccions basades en llist

- Existeix un gran nombre de col·leccions en .NET que implementen aquesta interfície, (sobretot les col·leccions basades en controls), entre les quals podem destacar les següents:
 - ***ArrayList***, la col·lecció "clàssica" per a aquest tipus d'interfície. Conté tots els membres habituals en aquest tipus de col·leccions.
 - ***CollectionBase***, una classe abstracta per poder crear les nostres pròpies col·leccions basades en *ICollection*.
 - ***StringCollection***, una col·lecció especialitzada que solament pot contenir valors de tipus cadena.

Col·lecció ArrayList

- Aquesta és una classe que representa una llista de dades.
- El ArrayList pot augmentar o disminuir la seva grandària dinàmicament d'una manera eficient. Amb un array de dades no era possible augmentar la capacitat del vector ja que aquest paràmetre és especificat al moment de crear la instància de l'objecte.
- El ArrayList a diferència, brinda la possibilitat d'augmentar o disminuir la seva grandària dinàmicament segons sigui necessari.

Col·lecció ArrayList

- Igual que ocorre amb els arrays, l'índex inferior és sempre el zero i els elements s'emmagatzemen de forma consecutiva, és a dir, si afegim dos elements a una col·lecció de tipus ArrayList (i a les quals implementin la interfície IList), el primer ocuparà la posició zero i el segon la posició un.
- Per crear una instància d'aquest objecte, s'ha d'utilitzar la classe ArrayList inclosa a l'espai de nom **System.Collections** com es mostra a continuació.
- **ArrayList arrayList=new ArrayList();**

Col·lecció ArrayList

- El constructor de la classe ArrayList accepta també un paràmetre tipus sencer que indica la capacitat inicial de l'objecte que s'aquesta creant.
- Si és necessari agregar un objecte a la col·lecció, s'ha d'utilitzar el mètode **Add**, el qual insereix el nou element en l'última posició, o el mètode **Insert** el qual ho insereix en la posició indicada.

Col·lecció ArrayList

```
//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");
```

Col·lecció Arraylist

- **Tots els objectes emmagatzemats en un Arraylist són tractats com a objectes**, per tant és possible agregar tot tipus de dades, és a dir, es pot agregar enters, cadenes de text, objectes de classes pròpies, etc.
- I a diferència dels array, **no tots els elements han de ser del mateix tipus de dada**. Això en algunes ocasions pot ser un avantatge ja que permet emmagatzemar gran varietat d'informació en una sola col·lecció, no obstant això, per raons de rendiment (cast, boxing, unboxing; convertir un tipus per valor en un per referència quan va a guardar-ho en la col·lecció (boxing), i el procés invers quan el volem recuperar (unboxing)), hi ha ocasions en les quals és preferible utilitzar les col·leccions genèriques.

Col·lecció Arraylist

- Si és necessari llevar elements de la col·lecció, s'ha d'usar el mètode **remove**, **removeAt** o **RemoveRange**, els quals eliminen l'objecte passat com a paràmetre, o un element en una posició específica, o un grup d'elements respectivament.

Col·lecció Arraylist

- Les propietats més utilitzades d'aquesta col·lecció són : **Count** i **Capacity**.
- La primera serveix per conèixer la quantitat actual d'elements que conté la col·lecció.
- La segona indica la capacitat màxima actual de la col·lecció per emmagatzemar elements. És necessari tenir present que la capacitat de la col·lecció, augmenta en cas de ser necessari en inserir un element, amb el que es garanteix el redimensionamiento automàtic.

Col·lecció Arraylist

- La capacitat d'una col·lecció mai podrà ser menor a la quantitat total d'elements continguts, per la qual cosa si es modifica manualment la propietat Capacity i se li assigna un valor menor que el valor retornat per la propietat Count, obtindrem una excepció de tipus **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);
```

ex 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 . . .
```

Col·lecció ArrayList

- Per accedir als elements continguts per la col·lecció es pot fer mitjançant l'ús d'índexs o mitjançant la instrucció foreach.

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

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

Quin triar?

- Determinar què tipus de col·lecció usar en un cas específic, és tasca del desenvolupador i s'ha d'avaluar les condicions per determinar la manera més eficient d'administrar els recursos.
- Si és un escenari on no coneixem la grandària que tindrà la col·lecció i si a més serà molt probable que la grandària variï, llavors serà recomanable sota totes les altres circumstàncies usar un **ArrayList** en lloc d'un array a causa que el ArrayList brinda la possibilitat de redimensionarlo automàticament.
- No obstant això, per a escenaris on es coneix per endavant la quantitat total d'elements a emmagatzemar i si tots són del mateix tipus, s'ha d'usar el **array convencional** ja que els objectes són emmagatzemats en el seu tipus de dades natiu i no és necessari fer conversions.