C#

Diseño y Programación Avanzada de Aplicaciones

Curso 2002-2003

INDICE

Introducción de C++ a C#

- C++ fue diseñado para la POO de propósito general.
  - Interfaz de usuario basada en línea de comandos
- C# diseñado para trabajar en la plataforma .Net
  - Interfaz Windows
  - Redes e Internet

Concepto de P.O.O.

- Método de implementación en que los programas se organizan como colecciones cooperativas de **objetos,** cada uno de los cuales representan una instancia de alguna **clase**, y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de **herencia**. (Grady Booch)

Concepto de POO (II)

- La POO es una evolución de los lenguajes procedurales en la que se trata de modelar mediante código los objetos reales con los que se van a trabajar.
- “El control lo tienen los objetos no las funciones”

Beneficios de la POO

- Reusabilidad
  - Los nuevos sistemas O.O. pueden ser creados utilizando S.O.O. anteriormente creados.
- Extensibilidad
  - Los nuevos sistemas O.O. así obtenidos son fácilmente ampliables sin tener que 'retocar' los módulos, S.O.O., empleados en su construcción.

Concepto de P.O.O. (II)

- Cambia…
  - El modo de organización del programa:
    - En clases (datos + operaciones sobre datos)
  - Quiénes se encargan de trabajar con la información
    - Las funciones miembro de las clases
  - El concepto de ejecución de programa
    - Paso de mensajes

Clases

- “Esencia” de un objeto: abstracción de los atributos (características) y operaciones comunes a un conjunto de objetos
- Así, una clase representa al conjunto de objetos que comparten una estructura y un comportamiento comunes
- Otras estructuras = miembros de datos
- Clases = miembros de datos + acciones

Clases (II)

- Tanto métodos como lenguajes OO tienen el concepto de clase como elemento central.
  - Deben ser los únicos módulos
  - NO HAY NOCIÓN DE MAIN!!!!
- Todos los tipos de datos deben ser clases

Objeto

- **Objeto**: Entidad que contiene los atributos que describen su **estado** y las acciones (**comportamiento**) que se asocian con el objeto del mundo real.
- Se **instancia** a partir de una clase
- Todo objeto es instancia de alguna clase

Objeto (II)

- Un objeto es un elemento tangible (ocupa memoria) generado a partir de una definición de clase.
- Se puede definir como una entidad (real o abstracta) con un papel bien definido en el dominio del problema
- Conceptos:
  - Identidad,
  - Estado
  - Comportamiento

Objeto (III)

- Estado: conjunto de propiedades (estáticas) y valores actuales de esas propiedades (dinámicos)
- Comportamiento: modo en que éste actúa y reacciona (cambios de su estado y paso de mensajes)
- Identidad: propiedad que distingue a unos objetos de otros (nombre único de variable)

Definición de una Clase

- Identificador de Clase: nombre
- Atributos o variables: datos necesarios para describir los objetos creados a partir de la clase (sus INSTANCIAS)
- Servicios, operaciones, métodos o funciones miembro: acciones que un objeto conoce cómo ha de ejecutar

Conceptos de OO

- Abstracción: Diseño de una solución abstrayéndose de los detalles de implementación.
- Encapsulación: Oculta la información, mediante partes independientes, para la especificación y la implementación.
- Herencia: Permite la reutilización de código.
- Polimorfismo: permitir que existan operaciones con igual nombre que se utilicen para manejar objetos de tipo diferente.

Características generales del C#

- Lenguaje orientado a objetos
- Diseñado para la construcción de componentes
- Gran parte de la plataforma .Net está escrita en C# (Es un lenguaje muy probado)
- Lenguaje abierto (definida en ECMA)

Características generales del C# (II)

- Lenguaje orientado a objetos
- Diseñado para la construcción de componentes
- Lenguaje .Net de primer nivel
  - Genera código MSIL
  - Utiliza el sistema de tipos común de la plataforma .Net
  - Tiene acceso a la biblioteca de .Net
  - Permite recolección de basura, seguridad e interoperabilidad con otros lenguajes.

Tipos básicos de C#.Net

- **bool (Boolean)** Valores *true* o *false*
- **byte** (byte) Entero sin signo de 8 bits
- **char** (char) Carácter de 16 bits
- **int (int32)** Entero sin signo de 32 bits
- **uint (uint32)** Entero con signo de 32 bits
- **long (int64)** Entero sin signo de 64 bits
- **ulong (uint64)** Entero con signo de 64 bits
- **float** (s**ingle)** Valor punto flotante de precisión de 32 bits
- **double (double)** Valor real de doble precisión
- **string** (string) Cadena de carácteres

De C++ a C#

- C# compila a código intermedio (MSIL), C++ compila a código nativo.
- Gestión de memoria. C# libera al programador de las tareas relacionadas con la gestión de memoria. (Recolección de basura)
- Punteros No son tan necesarios como en C++
- Sobrecarga de operadores Más limitada en C#

Primer programa en C#

```
using System;

namespace ConsoleApplication5
{
        class Class1
        {
                static void Main(string[] args)
                {
                Console.WriteLine("Hola alumnos de DPAA");
                }
        }
}
```

¿Dónde están los includes?

¿namespace?

¿Main o main?

¿Main dentro de una clase?

¿Console.WriteLine?

String[] args

¿Falta el ; al final de la clase?


```
Hola alumnos de DPAA
Press any key to continue
```

Donde están los includes?

- Donde están los *includes*
  - En C# no son necesarios
- ¿Main o main?
  - Main
- ¿Main dentro de una clase?
  - El main se debe definir dentro de una clase
- ¿Console.WriteLine?
  - Mas intuitivo que el “<<“
- ¿Falta el ; al final de la clase?
  - No

Ámbitos con nombre o *namespaces*

- Un *namespace* es un ámbito delimitado explícitamente al que se le ha asignado un identificador.
- En el interior de un *namespace* se pueden definir nuevos *namespaces*.
- Los *namespaces* solucionan el problema de la nomenclatura de elementos en aplicaciones grandes.

Ámbitos con nombre o *namespaces (II)*

- Sintaxis

```
namespace Nombre {
// definiciones
}
```

Ámbitos con nombre o *namespaces (III)*

```
using System;

namespace Fernando.ProgramacionVSNET
{
  class HolaMundo
  {
    static void Main()
    {
      System.Console.WriteLine("Hola desde Fernando.ProgramacionVSNET");
    }
  }
}
```



```
Hola desde Fernando.ProgramacionVSNET
Press any key to continue
```

Ámbitos con nombre o *namespaces (IV)*

```
using System;

namespace ClasesFernando
{

  namespace ProgramacionVSNET
  {
//
  /// Clase HolaMundo de ProgramacionVSNET
  public class HolaMundo
    { // con una clase llamada HolaMundo
      public System.String Saludo()
      {
        return  "Hola desde el ámbito Fernando.ProgramacionVSNET";
      }
    }
  }
```

ClasesFernando es el espacio superior

ProgramacionVSNET es un espacio interior

Ámbitos con nombre o *namespaces (V)*

```
using System;

namespace ClasesFernando
{                                   // ProgramacionVBNET es otro espacio interior

  namespace ProgramacionVBNET
  {
/// Clase HolaMundo de ProgramacionVBNET
 public class HolaMundo             // también tiene una clase HolaMundo
    {/// Método que devuelve el mensaje de saludo
   /// <returns>Cadena de caracteres con el saludo</returns>
      public System.String Saludo()
      {
        return
          "Hola desde el ámbito Fernando.ProgramacionVBNET";
      }
    }
  }
  }
```

Ámbitos con nombre o *namespaces (VI)*

```
using System;
using ClasesFernando.ProgramacionVSNET;
using ClasesFernando.ProgramacionVBNET;
class UsaHolaMundo
{
  static void Main()
  {
    HolaMundo MiHolaMundo = new HolaMundo();

    Console.WriteLine(MiHolaMundo.Saludo());
  }
}
```


Ambigüedad. ¿ Que Hola Mundo?

Ámbitos con nombre o *namespaces (VII)*

```
using System;
using ClasesFernando.ProgramacionVSNET;
using ClasesFernando.ProgramacionVBNET;
class UsaHolaMundo
{
  static void Main()
  {
        ClasesFernando.ProgramacionVSNET.HolaMundo  MiHolaMundo =
              new  ClasesFernando.ProgramacionVSNET.HolaMundo();

    Console.WriteLine(MiHolaMundo.Saludo());
  }
}
```

```
// Aquí 'HolaMundo' no es una referencia ambigua
```


"Peñazo" tener que escribirlo todo

Ámbitos con nombre o *namespaces (VIII)*

```
using System;
//using VS = ClasesFernando.ProgramacionVSNET; // incluimos referencias
//using VB = ClasesFernando.ProgramacionVBNET; // a los dos ámbitos

class UsaHolaMundo
{
  static void Main()
  {
    // Aquí 'HolaMundo' no es una referencia ambigua
            VS.HolaMundo  MiHolaMundo =
                    new  VS.HolaMundo();

    Console.WriteLine(MiHolaMundo.Saludo());
  }
}
```

Variables en C#

- Las variables que son campos miembros se inicializan por defecto.
  - Numéricas a 0
  - Bool a false
  - String a Null
- Las variables locales no se inicializan por defecto, pero el compilador producirá un error si una variable se utiliza sin haberle asignado un valor

Variables en C# (II)

¿Qué visualiza?

```
using System;
class Class1
        {

                static void Main(string[] args)
                {
                        int a, b;
                        bool enc;

                        Console.WriteLine("{0} {1} {2}", a,b,enc);
                }
        }
```


Error, uso de variables locales no asignadas

Variables en C# (III)

¿Qué visualiza?

```
using System;
class Class1
        {

                static void Main(string[] args)
                {
                        int a, b;
                        bool enc;
                        a = 2;
                        b = 3 ;
                        enc = (a == b );
                        Console.WriteLine("{0} {1} {2}", a,b,enc);
                }
        }
```

2 3 False

Variables en C# (IV)

```
using System;
class Class1
        {

        static void Mai
                {
                int a, b
                bool e
                Console.Write ("Introduce el primer número: ");
                cadena = Console.ReadLine();
                a = Int32.Parse(cadena);
                Console.Write("Introduce el segundo número: ");
                cadena = Console.ReadLine();
                b = Int32.Parse(cadena);
        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));
                }
        }
```

¿Qué visualiza?

```
Introduce el primer número: 4
Introduce el segundo número: 5
La suma de 4 y 5 es 9
La suma de 4 y 5 es 9
```

Estructuras de control

- Funcionan exactamente igual que en C++
  - for
  - return
  - break
  - continue
- Cambian
  - if else
  - while do
  - do while
  - switch
- Nuevas estructuras del C# :
  - foreach

if .. else

```
using System;
class Class1
        {

        static int Main(string[] args)
                {
                        int x=1;
                        if (x)
                          Console.WriteLine("Hola Mundo");
                        return 0;
                }

        }
}
```

¿Qué visualiza?

Error. Correcto (if x!= 0)

Ojo C# no permite conversión de númerica a bool o viceversa

While y do…while

- Lo mismo que la estructura if..else.
  - Solo permiten expresiones lógicas en la condición de fin de bucle.

Switch

Ventaja. Permite utilizar cadenas en las variables de selección

```
string cad = "Hola";
        switch (cad)
        {
        case "Hola":
        Console.WriteLine("Hola Mundo");
         break
        case "Adios":
        Console.WriteLine("Hola Mundo");
        break;
        default :  Console.WriteLine("Nada");
        break;
        }
```

¿Qué visualiza?

Correcto

ERROR. Ojo en C# cada claúsula *case* debe garantizar una salida

Switch (II)

```
string cad = "Hola";
switch (cad)
        {
        case "Hola":
        Console.WriteLine("Hola Mundo");
        goto case "Adios";
        case "Adios":
        Console.WriteLine("Adios Mundo");
        break;
        default :  Console.WriteLine("Nada");
        break;
        }
```

¿Qué visualiza?

Cada clausula case debe garantizar una salida

Hola Mundo

Adios Mundo

foreach

Permite recorrer todos los elementos de un array o colección sin necesidad de un índice explícito

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


¿Qué visualiza?

123

Foreach (II)

No permite realizar asignaciones

```
static void Main(string[] args)
                    {
                    int [ ] mivector = new int [10];
                    foreach(int algo in mivector)
                      algo = 2;
                    }
```


¿Qué visualiza?

ERROR. No se puede asignar nada a variables de lectura

Arrays

- Superficialmente el tratamiento de los arrays en C# es igual que en C++ pero:
  - En C# un array es una instancia de clase
  - Está bajo el control del recolector de basura
  - Su rendimiento es menor
  - Es más cómodo de manejar
    - Disponen de la propiedad Length.
    - Verifica la validez de acceso a un índice

Arrays (II)

¿Qué hace?

```
static void Main(string[] args)
                {
                int i;
                int [ ] mivector = new int [10];
                Console.WriteLine(mivector.Length);
                for ( i = 0; i < 11 ; i++)
                mivector[i] = i ;
                }
```


10
Excepción. IndexOutofRangeException

Arrays (III)

¿Qué visualiza?

```
static void Main(string[] args)
                {
                int i;
                int [ ] mivector = new int [10];
                Console.WriteLine(mivector.Length);
                for ( i = 0; i < 10 ; i++)
                        Console.Write(mivector[i]);
                }
```

```
10
0000000000
```

Si un array no es inicializado explícitamente el constructor por defecto se llamará para cada uno de los elementos. (Como si fueran miembros de una clase)

Arrays Multidimensionales

¿Qué visualiza?

```
1 0
3 6
9 12
```

```
static void Main(string[] args)
        {
        int i,j;
        int [,] mivectorMulti =  { {1, 0}, {3, 6}, {9, 12} };
        for ( i = 0; i < 3 ; i++)
                {
                        for ( j = 0; j <2 ; j++)
                        Console.Write(mivectorMulti[i,j] + " ");

                        Console.WriteLine();
                }
        }
```


Observar referencias e inicialización

Arrays Multidimensionales (II)

¿Qué visualiza?

```
static void Main(string[] args)
                {
                int [,] mivectorMulti =  { {1, 0}, {3, 6}, {9, 12} };

                        foreach ( int indice in mivectorMulti)
                                        Console.Write(indice);

                        }
                }
```

1036912

Los tipos como objetos

- Los tipos de datos básicos en C# pueden ser tratados como objetos

```
static int Main(string[] args)
{
        int I = 10;
        string Y = I.ToString();
        Console.WriteLine(Y);

        return 0;
}
```

Los tipos básicos como objetos (II)

- E incluso

```
static int Main(string[] args)
{
        string Y = 10.ToString();
        Console.WriteLine(Y);

        return 0;
}
```


Realmente C#compila todos los tipos básicos a las clases bases. Para C# todo es un objeto

Los tipos básicos como objetos (III)

- Todos los tipos tienen un método ToString(). Este método devuelve una representación en forma de cadena de caracteres de su valor.
- Char ofrece gran cantidad de propiedades que devuelven información sobre su contenido (IsLetter, IsNumber, etc) y métodos para efectuar conversiones (To Upper(), ToLower());

Los tipos básicos como objetos (IV)

- Ejemplos

```
static int Main(string[] args)
{
        char a = 'A';
        if (char.IsLetter(a))

          Console.WriteLine(char.ToUpper(a));


        return 0;
}
```

Los tipos básicos como objetos (IV)

- Los tipos enteros ofrecen MinValue y MaxValue para indicar valores que puede representar un tipo. Epsilon el menor valor mayor que cero en los double y float

```
static int Main(string[] args)
{
Console.WriteLine(Int16.MinValue);
Console.WriteLine(Int16.MaxValue);
Console.WriteLine(Int32.MinValue);
Console.WriteLine(Int32.MaxValue);
Console.WriteLine(double.Epsilon);
return 0;
}
```

Los tipos básicos como objetos (V)

- Para los tipos double y float se definen:
  - Nan (Not a Number, no definido).
  - PositiveInfinity
  - NegativeInfinity

```
static int Main(string[] args)
        {
        double a, b;
        a=?;
        b=?;
        Console.WriteLine(a/b);
        return 0;
        }
```


Si a y b = 0. Escribe Nan

Si a>0 y b = 0. Escribe Infinito

Si a<0 y b = 0. Escribe -Infinito

Los tipos básicos como objetos (VI)

- Que mostraría ?

```
static void Main(string[] args)
        {
         double a, b,c ;
                    a=7;
                    b=0;
                    c = a/b;
                    if (double.IsPositiveInfinity(c))
                              Console.WriteLine("Es negativo e infinito");

        }
```


Es negativo e infinito

Conversiones entre tipos

- C# es menos flexible que C++.
  - Implícitas Aquellas en las que no hay riesgo de pérdida de datos
  - Explícitas Aquellas en las que hay riesgo de pérdida de datos

```
static void Main(string[] args)
        {
         float f1 = 40.0 ;
         long l1 = f1 ; //
      short s1 = (short) l1; // Conversión explícita estilo C
      int i1 = s1;  //Conversión implícita
     short s1 = short l1; // Conversión explícita estilo C++
  }
```


Debiera ser long l1 =(long) f1

No permite estilo de C++. Debe ser (short)

Secuencias de Escape

- C# utiliza el mismo método que C++ para mostrar los caracteres de escape, utilizando la barra invertida.
- Diferencias
  - No existe el carácter ‘\0’
  - C# ofrece la secuencia \uxxxx, donde xxxx representa el código Unicode
  - Colocando un carácter @ muestra cadenas verbatim (tal cual).

Secuencias de Escape (II)

```
static void Main(string[] args)
        {
         //Console.WriteLine("c:\hola\mundo"); Errorea
       Console.WriteLine("c:\\hola\\mundo");
         Console.WriteLine(@"c:\hola\mundo");
  }
```



```
C:\hola\mundo

C:\hola\mundo
```

El operador new

- En C# el operador new simplemente indica que se está llamando al constructor de una variable.
- Es posible utilizar new para llamar al constructor de los tipos predefinidos
- Que diferencia hay en

```
int X = new int();
int X = 0;
int X;
```

La última deja sin inicializar a X si es una variable local.

Métodos

- En C# los métodos se definen que las funciones en C++, salvo que siempre deben ser métodos de clase y que la definición y la declaración se colocan juntas.

Métodos. Parámetros

- En C# los parámetros se pasan por valor
- Si se desea modificar esto
  - ref Parámetros por referencia (Deben estar inicializados antes de pasarlos)
  - out Parámetros de salida
- Se deben especificar tanto en la definición del método como en la llamada al mismo.

Métodos. Parámetros (II)

- Definición

```
public void Cuadrado( ref double valor, out double salida)
        {
                valor *= 2;
                salida = valor * valor;
        }
```


- Llamada

```
m.Cuadrado(ref a, out b);
```

Métodos. Sobrecarga.

- C# permite sobrecargar métodos, pero no permite valores por defecto-.
- En C++

```
public void Cuadrado(double valor, bool cond = true)
{
}
```


- En C#

```
public void Cuadrado(double valor)
{
Cuadrado(valor, true)
}
public void Cuadrado(double valor, bool cond)
{
}
```

Propiedades

- Las propiedades no tienen equivalente en C++.
- Es un método que se disfraza sintácticamente de forma que parece el tratamiento de un campo simple.

Propiedades(II)

LLAMADA

```
m.Lado = 2;
Console.WriteLine(m.Lado);
```

```
class Class2
    {
        public Class2() {}
        private double lado;
        public double Lado
        {
                get
                { return lado; }
                set
                {
                        lado = value; // Value es un parametro
    adicional implícito }

                }
        }       }
```

Propiedades (II)

- Las propiedades no tienen equivalente en C++.
- Es un método que se disfraza sintácticamente de forma que parece el tratamiento de un campo simple.

- En C#

```
public void Cuadrado(double valor)
        {
    Cuadrado(valor, true)
        }
public void Cuadrado(double valor, bool cond)
        {
        }
```

Definición de una clase

```
class MiClase : MiClaseBase
    {
        private string MiCampo;
        public int MiMetodo()
          {
          return 2;
          }
 }
```