Fundamentos de C#

TÓPICOS SELECTOS DE COMPUTACIÓN I

SEPTIEMBRE DE 2014

Agenda

Introducción

Tipos

Estructuras de control

Definición de clases, métodos, sobrecarga

Propiedades

Herencia

Clases abstractas

Miembros virtuales, interfaces

C Sharp

Es un lenguaje de programación orientado a objetos.

Se basa en los lenguajes C, C++, Java y Visual Basic.

Se trata de un lenguaje que combina todas las cualidades que se pueden esperar de un lenguaje moderno orientado a objetos.

C Sharp

Desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

C# (C Sharp) es un lenguaje simple, seguro, moderno, de alto rendimiento y con especial énfasis en internet y sus estándares (como XML).

Es también la principal herramienta para programar en la plataforma .NET.

C Sharp

Incorpora gestión automática de memoria, etc.) a la vez que proporciona un gran rendimiento.

Fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

*ECMA es una organización internacional basada en membresías de estándares para la comunicación y la información.

C Sharp

C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

El nombre C Sharp fue inspirado por la notación musical, donde ♯ (sostenido, en inglés sharp) indica que la nota (C es la nota do en inglés) es un semitono más alto, sugiriendo que C# es superior a C/C++.

El signo '#' viene de cuatro '+' pegados.

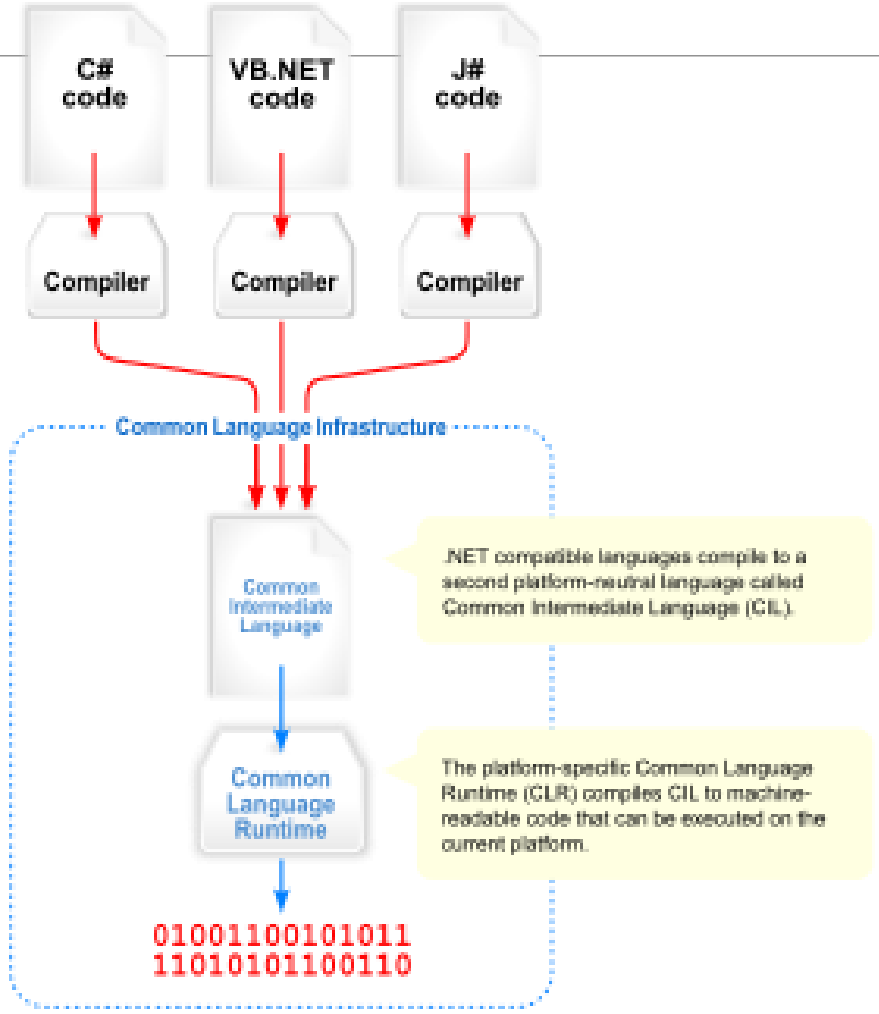
C Sharp

C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común (CLI o *Common Language Infrastructure).*

CLI es una especificación estandarizada que describe un entorno virtual para la ejecución de aplicaciones.

CLI permite que aplicaciones escritas en distintos lenguajes de alto nivel puedan ejecutarse en múltiples plataformas tanto de hardware como de software sin necesidad de reescribir o recompilar su código fuente.

CLI


C# code
VB.NET code
J# code
Compiler
Compiler
Compiler
Common Language Infrastructure
Common Intermediate Language
Common Language Runtime
.NET compatible languages compile to a second platform-neutral language called Common Intermediate Language (CIL).
The platform-specific Common Language Runtime (CLR) compiles CIL to machine-readable code that can be executed on the current platform.
01001100101011
11010101100110

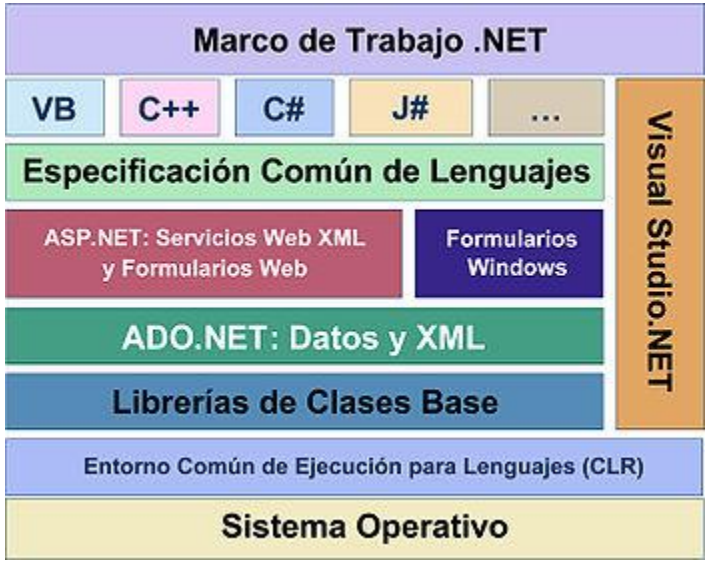
C Sharp

C# no es la plataforma .NET.

La plataforma .NET es la API para desarrollo.

C# es el lenguaje utilizado para hacer programas sobre la plataforma .NET.

Plataforma .NET



C Sharp

C# contiene las herramientas para definir nuevas clases, sus métodos y propiedades.

Ofrece la habilidad para implementar encapsulación, herencia y polimorfismo, que son los tres pilares de la programación orientada a objetos.

C# tiene un nuevo estilo de documentación XML que se incorpora a lo largo de la aplicación.

C# soporta también interfaces, una forma de estipular los servicios requeridos de una clase.

Las clases en C# pueden heredar de un padre pero puede implementar varias interfaces.

C# también provee soporte para estructuras.

C# vs Java

C# es mucho más cercano a C++ que a Java en cuanto a diseño.

C# toma casi todos sus operadores, palabras reservadas y expresiones directamente de C++.

Ofrece la posibilidad de trabajar directamente con direcciones de memoria.

C# vs C++

C# se ejecuta en una máquina virtual, la cual se hace cargo de la gestión de memoria y por lo tanto el uso de punteros es menos importante en C# que en C++.

C# también es mucho más orientado a objetos, los tipos usados derivan en última instancia el tipo 'object'.

Los métodos de las clases son accedidos mediante '.' en lugar de '::'.

Requisitos para iniciar

El conjunto de utilidades "Microsoft .NET Framework" y el ".NET Framework SDK" para Windows y el proyecto MONO o dotGNU para Linux o MacOS proporcionan estas herramientas.

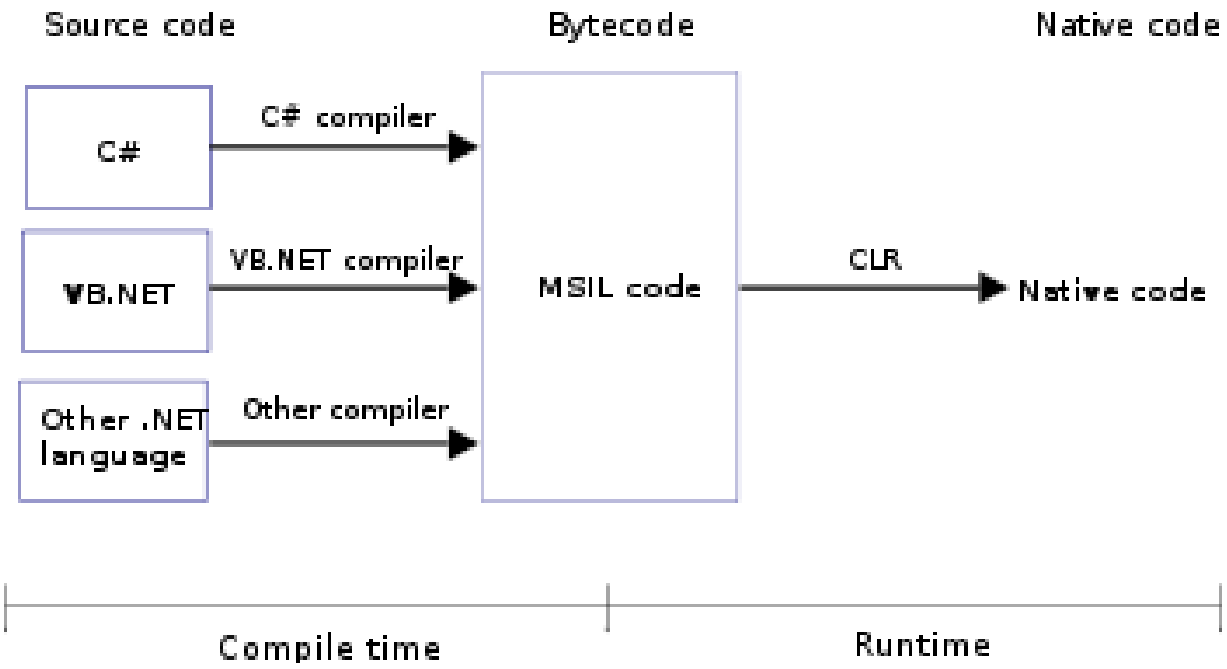
CLR
Common Language Runtime

- El Common Language Runtime o CLR es un entorno de ejecución para los códigos de los programas que corren sobre la plataforma Microsoft .NET.

- El CLR es el encargado de compilar una forma de código intermedio llamada Common Intermediate Language (CIL, anteriormente conocido como MSIL, por Microsoft Intermediate Language), al código de máquina nativo, mediante un compilador en tiempo de ejecución.

- Es una implementación del estándar Common Language Infrastructure (CLI).

CLR
Common Language Runtime



Agenda

Introducción

Tipos

Estructuras de control

Definición de clases, métodos, sobrecarga

Propiedades

Herencia

Clases abstractas

Miembros virtuales, interfaces

Tipos

Tipos de valor

- Almacenan datos

Tipos de referencia

- Almacenan referencias a los datos reales.

Tipo de punteros

- Los tipos de referencia también se denominan objetos. Los tipos de puntero sólo se pueden utilizar en modo unsafe.

Tipos de valor

Los tipos de valor consisten en dos categorías principales:

- Estructuras
- Enumeraciones

Las estructuras se dividen en las siguientes categorías:

- Tipos numéricos
  - Tipos integrales
  - Tipos de punto flotante
  - Decimal (mayor precisión)

bool

Estructuras definidas por el usuario.

http://msdn.microsoft.com/es-es/library/exx3b86w.aspx

Tipos que aceptan valores NULL

Los tipos que aceptan valores NULL son instancias de la estructura System.Nullable. Un tipo que acepta valores NULL puede representar el rango normal de valores de su tipo de valor subyacente, además de un valor null adicional.

```
int? num = null;

 if (num.HasValue == true) {
            System.Console.WriteLine("num = " + num.Value);

} else {

             System.Console.WriteLine("num = Null");

}
```

Tipos que aceptan valores NULL

La sintaxis **T?** es la forma abreviada de System.Nullable < T >, donde **T** es un tipo de valor. Las dos formas son intercambiables.

- int? num = null; //OK
- System.Nullable<int> num = null; //OK

Tipos de referencia

Las variables de tipos de referencia, conocidas como objetos, almacenan referencias a los datos reales. Esta sección presenta las palabras clave siguientes utilizadas para declarar tipos de referencia:

- clase
- interfaz
- delegado

Tipos de referencia integrados:

- dynamic
- objeto
- string

http://msdn.microsoft.com/es-es/library/490f96s2.aspx

Boxing/Unboxing

Las conversiones boxing y unboxing permiten tratar los tipos de valor como objetos. La aplicación de la conversión boxing a un tipo de valor empaqueta el tipo en una instancia del tipo de referencia Object.

```
-----------------------------------
int i = 123;
object o = (object)i; // boxing
-----------------------------------
o = 123;
i = (int)o; // unboxing
```

Var

A partir de Visual C# 3.0, las variables que se declaran en el ámbito de método pueden tener un tipo var implícito. A una variable local tipificada implícitamente se le asigna establecimiento inflexible de tipos como si hubiera declarado el tipo el propio usuario, pero es el compilador el que determina el tipo. Las dos declaraciones siguientes de i tienen una funcionalidad equivalente:

```
var i = 10; // implicitly typed

int i = 10; //explicitly typed
```

Palabras clave

Son identificadores predefinidos reservados que tienen un significado especial para el compilador. No se pueden utilizar como identificadores en un programa a menos que incluyan el carácter @ como prefijo. Por ejemplo, @if es un identificador válido pero if no, por ser una palabra clave.

http://msdn.microsoft.com/es-es/library/x53a06bb.aspx

Agenda

Introducción

Tipos

Estructuras de control

Definición de clases, métodos, sobrecarga

Propiedades

Herencia

Clases abstractas

Miembros virtuales, interfaces

Estructuras de control

while

do..while

for, foreach

if

switch

Agenda

Introducción

Tipos

Estructuras de control

Definición de clases, métodos, sobrecarga

Propiedades

Herencia

Clases abstractas

Miembros virtuales, interfaces

Estructuras

Las estructuras son tipos de valor; cuando un objeto se crea a partir de una estructura y se asigna a una variable, la variable contiene el valor completo de la estructura.

```
public struct PostalAddress {
        // Fields, properties, methods and events      go here...
}
```

Clases

En C#, una clase es un tipo de datos muy eficaz. Como las estructuras, las clases definen los datos y el comportamiento del tipo de datos. Los programadores pueden crear objetos que son instancias de una clase. A diferencia de las estructuras, las clases admiten herencia, que es una parte fundamental de la programación orientada a objetos.

Clases

La esencia de la programación orientada a objetos es la creación de nuevos tipos.

Un tipo puede ser la representación de un "componente". En programación un componente es algo que cumple una función.

Por ejemplo, un componente puede ser un botón, un panel, etc. En una aplicación podemos tener varios botones del mismo tipo como por ejemplo botones de Aceptar, Cancelar, etc. Estos botones tienen propiedades y comportamientos similares.

Los botones pueden tener propiedades como cambiar de "tamaño", "posición", etc. Las propiedades son las mismas pero los valores almacenados en sus atributos pueden ser diferentes.

Por ejemplo, un botón puede tener tamaño X y otro tamaño Y. Podemos decir entonces que tenemos varias instancias del mismo componente (o varios botones del mismo tipo) con diferentes valores en sus atributos.

Clases

```
public class Persona{

}
```

Métodos

Un *método* es un bloque de código que contiene una serie de instrucciones. Los programas hacen que las instrucciones se ejecuten mediante una *llamada* al método y la especificación de los argumentos de método necesarios. En C#, cada instrucción se ejecuta en el contexto de un método. El método Main es el punto de entrada de cada aplicación C# al que llama Common Language Runtime (CLR) cuando se inicia el programa.

Las funciones normalmente llevan nombre que definen su función. Por ejemplo, la función WriteLine() de la clase Console "Escribe una Línea en la consola".

De forma similar se pueden declarar clases a las que se le añaden un número ilimitado de métodos.

Main() es una función especial que indica la entrada principal de ejecución de un programa. Cada programa en C# debe tener una función Main().

http://msdn.microsoft.com/es-es/library/ms173114(VS.10).aspx

Métodos

Los métodos se declaran en una clase o struct mediante la especificación del nivel de acceso como public o private, modificadores opcionales como abstract, el valor devuelto, el nombre del método y cualquier parámetro de método. Todos esos elementos constituyen la firma del método.

http://msdn.microsoft.com/es-es/library/ms173114(VS.10).aspx

Declaración de métodos

[entorno] tipo_a_retornar Nombre_de_la_Función ([tipo Argumento1, tipo Argumento2,...])

{

    //lineas de código

}

Ejemplo:

static *void* Main() {

    //lineas de código

}

Sobrecargas en C#

De métodos

- public void Comer() { }
- public void Comer(int x) { }
- public void Comer(int x, int y) { }
- public void Comer(int x, int y, int z) { }
- public void Comer(int x, int y, int z, int w) { }

De operadores

- Al igual que C++, C# permite sobrecargar operadores para utilizarlos en clases propias. Esto hace posible que utilizar un tipo de datos definido por el usuario parezca tan natural y lógico como utilizar un tipo de datos fundamental.

Name Space

System.- Representa un Espacio de nombres (namespace en inglés).

Los espacios de nombres (namespaces) se crearon principalmente para dar más organización a las clases y elementos.

Este ámbito permite organizar el código y proporciona una forma de crear tipos globalmente únicos.

La plataforma .NET tiene incorporados muchos componentes y sería una tarea imposible tratar de memorizar todos los nombres de ellos para no repetirlos.

Agenda

Introducción

Tipos

Estructuras de control

Definición de clases, métodos, sobrecarga

Propiedades

Herencia

Clases abstractas

Miembros virtuales, interfaces

Propiedades

En C#, una propiedad es un miembro con nombre de una clase, estructura o interfaz que ofrece una forma ordenada de tener acceso a campos privados mediante lo que se denomina métodos de descriptor de acceso get y set.

```
private string apellido;

     public string Apellido
     {
        set {
             apellido = value;
         }
         get {
             return apellido;
```

Propiedades

También es válido:

```
public string Apellido{get;set;};
```

Si se quisiera restringir el acceso a get o set se establecer el nivel de acceso con private:

```
public string Apellido{get;private set;};
```

Agenda

Introducción

Tipos

Estructuras de control

Definición de clases, métodos, sobrecarga

Propiedades

Herencia

Clases abstractas

Miembros virtuales, interfaces

Herencia

Las clases pueden heredar de otra clase. Para conseguir esto, se coloca un signo de dos puntos después del nombre de la clase al declarar la clase y se denomina la clase de la cual se hereda (la clase base) después del signo de dos puntos.

```
public class A {
    public A() {
    }
}
 public class B : A {
        public B() {
        }
 }
```

Agenda

Introducción

Tipos

Estructuras de control

Definición de clases, métodos, sobrecarga

Propiedades

Herencia

Clases abstractas

Miembros virtuales, interfaces

Clases abstractas

La palabra clave abstract permite crear clases y miembros de clase únicamente con propósitos de herencia: para definir características de clases derivadas, no abstractas. La palabra clave sealed permite impedir la herencia de una clase o de ciertos miembros de clase marcados previamente como virtuales.

Agenda

Introducción

Tipos

Estructuras de control

Definición de clases, métodos, sobrecarga

Propiedades

Herencia

Clases abstractas

Miembros virtuales, interfaces

Métodos virtuales

La palabra clave **virtual** se utiliza para modificar un método, propiedad, indizador o declaración de evento y permite reemplazar a cualquiera de estos en una clase derivada. En el siguiente ejemplo, cualquier clase que hereda este método puede reemplazarlo:

```
public virtual double Area() {
                    return x * y;
}
```

Métodos virtuales

De forma predeterminada, los métodos son no virtuales. No se puede reemplazar un método no virtual.

No puede utilizar el modificador **virtual** con los modificadores **static**, **abstract, private** u **override**.

Delegados

- Un delegado es un tipo que hace referencia a un método. Cuando se asigna un método a un delegado, éste se comporta exactamente como el método. El método delegado se puede utilizar como cualquier otro método, con parámetros y un valor devuelto, como en este ejemplo:

```
public delegate int Calcular(int x, int y);
```

Delegados

- Los delegados son similares a los punteros a función de C++. Los delegados permiten pasar los métodos como parámetros.

- Los delegados pueden utilizarse para definir métodos de devolución de llamada.

- Los delegados pueden encadenarse; por ejemplo, se puede llamar a varios métodos en un solo evento.

- La versión 2.0 de C# introduce el concepto de métodos anónimos, que permiten pasar bloques de código como parámetros en lugar de utilizar métodos definidos independientemente.

Expresiones Lambda

Una expresión lambda es una función anónima que puede contener expresiones e instrucciones y se puede utilizar para crear delegados o tipos de árboles de expresión.

Todas las expresiones lambda utilizan el operador lambda

= >, que se lee como "va a".

El lado izquierdo del operador lambda especifica los parámetros de entrada (si existe alguno), mientras que el lado derecho contiene el bloque de expresiones o instrucciones. La expresión lambda x => x * x se lee "x va a x veces x".

Lambdas de expresión

Una expresión lambda con una expresión en el lado derecho se denomina lambda de expresión. Las lambdas de expresión se utilizan ampliamente en la construcción de Árboles de expresión (C# y Visual Basic).

Devuelven el resultado de la expresión y presenta la siguiente forma básica:

```
(input parameters) => expression

(x, y) => x == y

(int x, string s) => s.Length > x

() => SomeMethod()
```

Lambdas de instrucciones

Una lambda de instrucciones es similar a un lambda de expresión, salvo que las instrucciones se encierran entre llaves:

(input parameters) => {statement;}

El cuerpo de una lambda de instrucciones puede estar compuesto de cualquier número de instrucciones; sin embargo, en la práctica, generalmente no hay más de dos o tres.

```
delegate void TestDelegate(string s);
...
TestDelegate myDel = n => { string s = n + " " + "World";
                            Console.WriteLine(s);
                          };
myDel("Hello");
```

Lambdas con los operadores de consulta estándar

Muchos operadores de consulta estándar tienen un parámetro de entrada cuyo tipo es uno de la familia de delegados genéricos.

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

int newNumbers = numbers.Count(n => n % 2 == 0);
```

Regresa el número de pares en el arreglo.

Métodos de extensión

- Los métodos de extensión permiten "agregar" métodos a los tipos existentes sin necesidad de crear un nuevo tipo derivado y volver a compilar o sin necesidad de modificar el tipo original.

- Los métodos de extensión más comunes son los operadores de consulta estándar de LINQ que agregan funcionalidad de consulta a los tipos System.Collections.IEnumerable y System.Collections.Generic.IEnumerable(Of T) existentes. Para usar los operadores de consulta estándar, primero inclúyalos en el ámbito con una directiva using System.Linq.

Métodos de extensión

```
namespace ExtensionMetodo {
    public static class Extensions{
        public static int WordCount(this String str) {
                        return str.Split(new char[] { ' ', '.', '?' },
                        StringSplitOptions.RemoveEmptyEntries).Length;
            }
        }
 }
//Para usar
using ExtensionMetodo;
```

Indizadores

- Los indizadores suponen una comodidad sintáctica al permitir crear una clase, estructura o interfaz a las que las aplicaciones cliente pueden tener acceso como si se tratara de una matriz. Por lo general, los indizadores se implementan en tipos cuya finalidad principal es encapsular una matriz o colección interna.

Indizadores

```
class SampleCollection<T> {
        private T[] arr = new T[100];
        public T this[int i] {
              get { return arr[i]; }
             set { arr[i] = value; }
         }
  }
  // Esta clase muestra como usar el indexer
  class Program {
       static void Main(string[] args) {
         SampleCollection<string> stringCollection = new SampleCollection<string>();
         stringCollection[0] = "Hello, World";
         System.Console.WriteLine(stringCollection[0]);
       }
   }
```