

Informe Seminario 11 LEnguajes de Programacion

Dinamismo en C#



* Yasmin Cisneros Cimadevila C-311
* Miguel Angel González Calles C-311
* Alejandro Klever Clemente C-311
* Damian O`hallorans Toledo C-312

# Pregunta 1:

Implemente el tipo Prototype de forma tal que el siguiente código compile y ejecute (C# 4.0) con la salida que se muestra a continuación:

internal static class Program

{

private static void Main(string[] args)

{

// Característica A

dynamic parte1 = new Prototype();

parte1.MetodoA = (Action<dynamic>)((self) => {

Console.WriteLine("MétodoA dice '{0}'", self.frase);

});

// Característica B

dynamic parte2 = new Prototype();

parte2.MetodoB = (Action<dynamic>)((self) => {

Console.WriteLine("MétodoB dice '{0}'", self.frase);

});

// ‘Fundiendo’ ambos objetos

var obj = parte1.BlendWith(parte2);

// Usando el objeto 'frase'

obj.frase = "Hello World!";

obj.MetodoA();

obj.MetodoB();

}

}

## Salida:

MétodoA dice 'Hello World!'

MétodoB dice 'Hello World!'

# Solución:

public class Prototype: DynamicObject

{

        private Dictionary<string, dynamic> \_dictionary = new Dictionary<string, dynamic>();

        public override bool TryGetMember(GetMemberBinder binder, out dynamic result)

        {

            return \_dictionary.TryGetValue(binder.Name, out result);

        }

        public override bool TrySetMember(SetMemberBinder binder, dynamic value)

        {

            \_dictionary[binder.Name] = value;

            return true;

        }

        public Prototype BlendWith(Prototype other)

        {

            var prototype = new Prototype();

            foreach (var (key, value) in \_dictionary)

                prototype.\_dictionary[key] = (Action) (() => value(prototype));

            foreach (var (key, value) in other.\_dictionary)

                prototype.\_dictionary[key] = (Action) (() => value(prototype));

            return prototype;

        }

    }

Como podemos apreciar para dar una funcionalidad satisfactoria a nuestro tipo **Prototype** hereda de **DynamicObject** la cual es una clase que nos permite definir qué tipo de operaciones pueden realizarse sobre un objeto. Como por ejemplo intentar poner u obtener alguna propiedad, invocar un método o definir las operaciones aritméticas básicas. Esta clase no puede ser directamente instanciada, para obtener este comportamiento dinámico se debe heredar de **DynamicObject**. En C# para activar el comportamiento dinámico de las clases derivadas de **DynamicObject** se debe usar la palabra clave *dynamic*. Como en este caso solo queremos acceder y poner los valores de propiedades sobrescribimos los métodos TryGetMember y TrySetMember.

Ya C# 4 nos introduce un nuevo tipo *dynamic*, un tipo estático pero que es libre del chequeo de tipos en tiempo de compilación. En la mayoría de los casos es adquiere la función de un objeto de tipo object. En compilación aún objeto tipado con *dynamic* se asume que soporta cualquier operación. En tiempo de ejecución este objeto tipado como dynamic adquiere tipo específico en cada momento especifico de la ejecución, es decir un objeto tipado con *dynamic* puede comenzar siendo un objeto de tipo *int* y luego volverse de tipo *string*. Este comportamiento lo podemos obtener simplemente declarando dicho objeto como *object*, pero habría que realizar un casteo al tipo del que se desea acceder a la propiedad o el método, en este caso la principal ventaja de usar el tipo *dynamic* en estos casos es hacer el código menos verboso, de forma tal que cuando deseemos acceder a las propiedades o métodos de dicho objeto no se haga un casteo explícito de un tipo *object* al tipo del que se quiere obtener la propiedad. Si se usa dynamic como parámetro de genericidad como por ejemplo una **List**< *dynamic* > este objeto será declarado en tiempo de ejecución como **List**<*object*>

## Inciso 1:

¿Por qué los métodos extensores no funcionan con un tipo dinámico (*dynamic*)?

Los métodos de extensión permiten "agregar" métodos a los tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original. Los métodos de extensión son una clase especial de método estático, pero se les llama como si fueran métodos de instancia en el tipo extendido. No existe ninguna diferencia aparente entre llamar a un método de extensión y llamar a los métodos realmente definidos en un tipo. Por ejemplo, cualquier tipo que implemente **IEnumerable**<T> parecerá tener métodos de instancia como **GroupBy**, **OrderBy**, **Average**, etc.

Los métodos extensores son referenciados en el código IL cuando se compila la instancia de la clase. O sea al compilar la instancia de la clase, el compilador detecta que esta implementa cierta interfaz, y le agrega la referencia a los métodos extensores correspondientes.

Esto no es posible con el tipo *dynamic* pues de la forma en que es inicializada es llamando directamente al constructor de una clase, y no es posible para el compilador en tiempo de ejecución detectar cual y agregar los métodos extensores correspondientes.

## Inciso 2:

Implemente también para **Prototype** un método Clone de manera que los miembros de tipo función que fuesen adquiridos por el original o alguna de sus copias sean compartidos entre todos.

public Prototype Clone()

        {

            return new Prototype{\_dictionary = \_dictionary};

        }

# Referencias

Microsoft. (20 de 07 de 2015). *docs.microsoft.com*. Obtenido de docs.microsoft.com: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/extension-methods

Valle, M. d. (25 de 03 de 2011). *Tipado dinamico en C# 4.0*. Obtenido de Tipado dinamico en C# 4.0: https://desarrolloweb.com/articulos/tipado-dinamico-c-dotnet.html