Seminario 10 de Lenguajes de Programación. *Dinamismo en C#*

Equipo 1:
Carmen Irene Cabrera Rodríguez
Enrique Martínez González
David Guaty Domínguez
Julio José Horta Vázquez
C-312

March 29, 2020

1 Dinamismo en C#

Con el lanzamiento de C# 4.0 en el año 2010, aparece la programación dinámica como feature de este lenguaje. La palabra clave *dynamic* actúa como declaración de tipo estática en el sistema de tipado de C#, de forma que, aunque obtiene características dinámicas, continúa siendo un lenguaje de tipado estático. El uso de esta palabra clave dice al compilador que el elemento de este tipo será definido en tiempo de ejecución y por tanto, en tiempo de compilación se asume que puede soportar cualquier operación. No obstante, si el código no es válido, los errores se obtienen en tiempo de ejecución.

Un ejemplo común del uso de *dynamic* se muestra en el siguiente código:

```
dynamic d = "test";
Console.WriteLine(d.GetType());
// Prints "System.String".

d = 100;
Console.WriteLine(d.GetType());
// Prints "System.Int32".
```

Note que se hace posible asignar objetos de diferentes tipos a una variable que ha sido declarada como *dynamic*. El código compila y el tipo de objeto es identificado en tiempo de ejecución. Sin embargo, observe el siguiente código:

```
dynamic d = "test";

// The following line throws an exception at run time.
d++;
```

Este igual compila, por la misma razón mencionada anteriormente: el compilador deconoce el tipo del objeto que se ejecutará y por tanto, es incapaz de señalar que la operación de incremento no está soportada en este caso. En tiempo de ejecución, este código lanza excepción.

Además de *dynamic* existen otras dos variantes para lograr un comportamiento dinámico en C#, ambas introducidas en *.Net Framework 4* en la DLR; estas son: **ExpandoObject** y **DynamicObject**. Estas clases permitirán crear tipos dinámicos y se encuentran detalladamente explicadas en el *Seminario 8, Semigrupo 2*.

1.1 *dynamic* vs *var*

La palabra clave *var* constituye solo una instrucción para que el compilador infiera el tipo a partir de la expresión de inicialización de la variable; en caso que no pueda hacerlo, se produce error de compilación. Además el tipo de esa variable, contrario a lo que sucede cuando se utiliza *dynamic*, no puede cambiar durante el tiempo de ejecución.

Otra diferencia, es que *dynamic* puede ser utilizado como tipo de retorno o como tipo de parámetros en la declaración de una función.

1.2 ¿Cómo C# maneja el tipo *dynamic* en tiempo de compilación?

Dynamic es un pseudotipo, lo que significa que en tiempo de compilación C# trata al tipo *dynamic* como tipo *Object*; pero además, C# no realiza chequeo de tipos sobre dicho objeto. Al usar *dynamic*, cualquier miembro de este objeto (métodos, propiedades, delegados, etc) puede ser invocado y el compilador no realizará comprobación alguna en tiempo de compilación, esto se pospone para el tiempo de ejecución. El compilador utiliza las técnicas de Boxing y Unboxing para realizar lo anterior.
Para realizar el chequeo de miembros de un objeto *dynamic* en tiempo de ejecución, C# hace uso de COM IDispatch para objetos COM, IDynamicMetaObjectProvider DLR interface para objetos que implementen la interface y reflection para los demás objetos.

2 Resolución del problema

Se deseaba obtener una implementación de la clase **Prototype** para que el código dado compilara y su salida fuera la requerida; dicha implementación puede encontrarse en los archivos adjuntos a este documento. Para ello, la clase *Prototype* hereda de *DynamicObject* y por consiguiente, brinda una implementación de los métodos: *TryGetMember*, *TrySetMember* y *TryInvokeMembertext*. Vea que en el último caso se tiene en cuenta el uso de la palabra self para referirse la clase a sí misma, en lugar de la palabra clave *this*; lo cual es un comportamiento similar al que se observa en Python con el uso de *self* como palabra clave para dicho fin.

Igualmente, se implementó el método *BlendWith* que devuelve un nuevo objeto de tipo Prototype, como resultado de la mezcla entre dos del mismo tipo.

Además, se asume la clase *Prototype* como un ICloneable con el objetivo de proveer una implementación al método Clone() también requerido por el ejercicio.

3 Métodos extensores

La declaración de un método extensor debe realizarse dentro de una clase estática de la siguiente forma:

public static <*tipo de retorno*> <*nombre del método*>(this <*tipo a extender*> name, <argumentos>)

Estos permiten "agregar" métodos a tipos existentes sin necesidad de crear una clase derivada, recompilar o modificar el tipo original. Constituyen un tipo especial de métodos estáticos, pero son llamados como si fueran métodos de instancia del tipo extendido.

Se pueden usar métodos de extensión para ampliar una clase o interfaz, pero no para invalidarlas. Nunca se llamará a un método de extensión con el mismo nombre y signatura que un método de interfaz o clase. En tiempo de compilación, los métodos de extensión siempre tienen menos prioridad que los métodos de instancia definidos en el propio tipo. En otras palabras, si un tipo tiene un método denominado Process(int i) y hay un método de extensión con la misma signatura, el compilador siempre se enlazará al método de instancia. Cuando el compilador encuentra una invocación de método, primero busca una coincidencia en los métodos de instancia del tipo. Si no la hay, buscará cualquier método de extensión definido para el tipo y se enlazará al primer método de extensión que encuentre.

Todos los métodos de instancia son métodos de extensores donde el primer argumento es la instancia, C# lo esconde, pero se ve explícito en los métodos extensores.

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

"...los métodos extensores en código no dinámico funcionan haciendo una búsqueda completa en todas las clases conocidas por el compilador hasta encontrar una clase estática que tenga el método extensor que coincida. La búsqueda va en orden, basada en la jerarquía de los *namespaces* y las directivas *using* en cada *namespace*.

Esto significa que para que la invocación de un método de extensión dinámico sea resuelto correctamente, el DLR tiene que conocer en tiempo de ejecución (*runtime*) cuál es la jerarquía de *namespaces* y directivas *using* estaban en el código fuente. Nosotros no tenemos un mecanismo manuable para codificar toda la información necesaria en el lugar de la llamada (*call site*). Nosotros consideramos inventar ese mecanismo, pero decidimos que era de muy alto costo y producía mucho riesgo en la planificación para que valiera la pena."[1]

References

[1] StackOverflow. Lippert, Eric. Desarrollador y MVP de C#. 15 de marzo de 2011