(C++/CLI IV): Constructores de copia y de asignación

Continuando con el tema de los constructores y toda su parafernalia vamos a ver por encima ciertos aspectos que no suelen ser muy tenidos en cuenta en los libros sobre C++.

Un constructor de copia es aquel que crea un objeto a partir de otro ya existente. En C++ el compilador se encarga de crear un constructor de copia si el usuario no lo ha hecho así, pero para C++/CLI es necesario y obligatorio que nosotros definamos uno si vamos a utilizar esta característica en nuestros objetos.

El motivo más importante por el cual esta característica es así se debe a que cualquier clase siempre hereda por defecto de *Object*, y esta clase no implementa dicho constructor, ya que el sistema podría no saber qué copiar y qué no copiar. Esto es así porque todos los objetos .NET son polimórficos ya que son accedidos mediante un manejador y entonces el compilador no sabe si implementar copia en superficie o en profundidad, y hasta qué nivel de la misma.

Por todo ello, si queremos crear un objeto a partir de otro, como en

```
Cosa ^c=gcnew Cosa(otraCosa);
```

tenemos que implementar dicho constructor. También tenemos que darnos cuenta de que al implementar este también debemos implementar el constructor por defecto ya que el compilador de C++/CLI no lo hará por nosotros al haber definido otro. Una forma de hacerlo sería mediante de la siguiente plantilla:

```
ref class Cosa
{
public:
      Cosa(void){}
      Cosa(const Cosa ^){…}
};
```

Ahora ya podemos hacer uso del código que hemos mostrado en primer lugar.

Pero C++/CLI también soporta semántica de pila para objetos referencia, por lo que si utilizamos el código siguiente veremos que no podemos compilarlo:

```
Cosa c;
Cosa ^otra=gcnew Cosa(c);
```

El compilador protestará con el error C3073. Una posible solución a ello es utilizar el código

```
Cosa c;
Cosa ^otra=gcnew Cosa(%c);
```

que mediante el uso del operador unitario % nos devuelve un manejador al objeto deseado. Aunque no es una solución elegante, es funcional y perfectamente válida.

Otra forma sería implementar un constructor de copia que tomara una referencia al objeto en cuestión:

```
ref class Cosa
{
public:
      Cosa(void){}
      Cosa(const Cosa ^){…}
      Cosa(const Cosa %){…}
};
```

De este modo ya no tenemos necesidad de utilizar el operador unitario y nuestro código queda mucho más elegante. Pero ahora caemos en dos nuevos problemas.

El primero de ellos es la duplicación de código; es decir, tenemos dos lugares diferentes en los cuales realizamos una construcción de copia, lo que en general viene desaconsejado por la propia idiosincrasia del C++. Una solución algo menos elegante sería el que ambos constructores llamaran a un método compartido por ambos (recordemos que un constructor no debe llamar a otro, por lo que la solución de llamar al de copia por referencia desde el de copia por valor no es válida). Otro tema es que mediante el uso de un método intermedio no podemos utilizar listas de inicialización.

El segundo es un poco más serio en lo que respecta a la filosofía .NET, y consiste en que el segundo constructor de copia no será visible para ningún otro lenguaje .NET. Aunque realmente no supone ningún problema ya que simplemente no estará visible –no romperá nada ni afectará en ningún modo a otro código), por lo que podremos utilizarlo desde cualquier otro código en C++/CLI pero no, por ejemplo, desde C#.

Cualquiera de las dos soluciones es perfectamente válida, y queda a decisión del lector utilizar la que quiera, sin olvidar que la primera de ellas (utilizar el operador unitario %) no significa ninguna penalización en el rendimiento del ejecutable.

Construcción mediante asignación

Veamos el siguiente código:

```
Cosa c1;
Cosa c2=c1;
```

Sí se compila con C++ estándar no es necesario que definamos nuestro propio operador de asignación, pero por los mismos motivos que en C++/CLI no se autoimplementa el constructor de copia, tampoco lo hace el operador de asignación.

Pero aquí debemos distinguir entre copiar un manejador y copiar un objeto, es decir, el código siguiente es perfectamente válido y no requiere la definición de un operador de asignación, ya que lo que se está copiando es el manejador y no el objeto en sí:

```
Cosa ^c1=gcnew Cosa();
Cosa ^c2=^c1;
```

Por lo tanto, la definición del operador de asignación se hace de la misma forma en C++/CLI que en C++:

```
Cosa %operator=(const Cosa %)
{
…
}
```

Y dentro del cuerpo del método realizaremos las asignaciones que deseemos dependiendo del grado de profundidad con el que queramos realizar la asignación/copia.

Dado que los otros lenguajes .NET no soportan la copia de objetos referencia situados en la pila, dicho operador no estará visible para nadie más que para aquellos programas realizados en C++ y en C++/CLI. En otras palabras: como no podemos tener objetos referencia realmente situados en la pila, lo que el compilador de C++/CLI hace es darnos la ilusión de que están ahí, y por eso no tiene mucho sentido que lenguajes como C# o VB.NET puedan disponer de dicho operador.

Una última consideración a la hora de implementar este operador: ojito con la copia sobre sí mismo, que dependiendo de cómo la hagamos podría dejar el objeto en un estado no válido. Lo mejor es comprobar si la referencia de seguimiento obtenida es el propio objeto y en ese caso devolverse a sí mismo:

```
Cosa %operator=(const Cosa %c)
{
   if(%c==this)
      return *this;
   …
}
```