Sobrecarga de operadores en C#

Concepto de redefinición de operador

Un **operador** en C# no es más que un símbolo formado por uno o más caracteres que permite realizar una determinada operación entre uno o más datos y produce un resultado.

En C# viene predefinido el comportamiento de sus operadores cuando se aplican a ciertos tipos de datos. Por ejemplo, si se aplica el operador **+** entre dos objetos **int** devuelve su suma, y si se aplica entre dos objetos **string** devuelve su concatenación. Sin embargo, también se permite que el programador pueda definir el significado la mayoría de estos operadores cuando se apliquen a objetos de tipos que él haya definido, y esto es a lo que se le conoce como **redefinición de operador**.

Nótese que en realidad la posibilidad de redefinir un operador no aporta ninguna nueva funcionalidad al lenguaje y sólo se ha incluido en C# para facilitar la legibilidad del código. Por ejemplo, si tenemos una clase Complejo que representa números complejos podríamos definir una función Sumar() para sus objetos de modo que a través de ella se pudiese conseguir la suma de dos objetos de esta clase como muestra este ejemplo:

```
Complejo c1 = new Complejo(3,2); // c1 = 3 + 2i
Complejo c2 = new Complejo(5,2); // c2 = 5 + 2i
Complejo c3 = c1.Sumar(c2); // c3 = 8 + 4i
```

Sin embargo, el código sería mucho más legible e intuitivo si en vez de tenerse que usar el método Sumar() se redefiniese el significado del operador + para que al aplicarlo entre objetos Complejo devolviese su suma. Con ello, el código anterior quedaría así:

```
Complejo c1 = new Complejo(3,2); // c1 = 3 + 2i
Complejo c2 = new Complejo(5,2); // c2 = 5 + 2i
Complejo c3 = c1 + c2; // c3 = 8 + 4i
```

Ésta es precisamente la utilidad de la redefinición de operadores: hacer más claro y legible el código, no hacerlo más corto. Por tanto, cuando se redefina un operador es importante que se le dé un significado intuitivo ya que si no se iría contra de la filosofía de la redefinición de operadores. Por ejemplo, aunque sería posible redefinir el operador ***** para que cuando se aplicase entre objetos de tipo Complejo devuelva su suma o imprimiese los valores de sus operandos en la ventana de consola, sería absurdo hacerlo ya que más que clarificar el código lo que haría sería dificultar su comprensión.

De todas formas, suele ser buena idea que cada vez que se redefina un operador en un tipo de dato también se dé una definición de un método que funcione de forma equivalente al operador. Así desde lenguajes que no soporten la redefinición de operadores también podrá realizarse la operación y el tipo será más reutilizable.

Definición de redefiniciones de operadores

Sintaxis general de redefinición de operador

La forma en que se redefine un operador depende del tipo de operador del que se trate, ya que no es lo mismo definir un operador unario que uno binario. Sin embargo, como regla general podemos

considerar que se hace definiendo un método público y estático cuyo nombre sea el símbolo del operador a redefinir y venga precedido de la palabra reservada **operator**. Es decir, se sigue una sintaxis de la forma:

public static *<tipoDevuelto>* ***operator*** *<símbolo>****(****<operandos>****)***
{
    <cuerpo>
}

Los modificadores **public** y **static** pueden permutarse si se desea, lo que es importante es que siempre aparezcan en toda redefinición de operador. Se pueden redefinir tanto operadores unarios como binarios, y en <operandos> se ha de incluir tantos parámetros como operandos pueda tomar el operador a redefinir, ya que cada uno representará a uno de sus operandos. Por último, en <cuerpo> se han de escribir las instrucciones a ejecutar cada vez que se aplique la operación cuyo operador es <símbolo> a operandos de los tipos indicados en <operandos>.

<tipoDevuelto> no puede ser **void**, pues por definición toda operación tiene un resultado, por lo que todo operador ha de devolver algo. Además, permitirlo complicaría innecesariamente el compilador y éste tendría que admitir instrucciones poco intuitivas (como a+b; si el **+** estuviese redefinido con valor de retorno **void** para los tipos de a y b)

Además, los operadores no pueden redefinirse con total libertad ya que ello también dificultaría sin necesidad la legibilidad del código, por lo que se han introducido las siguientes restricciones al redefinirlos:

- Al menos uno de los operandos ha de ser del mismo tipo de dato del que sea miembro la redefinición del operador. Como puede deducirse, ello implica que aunque puedan sobrecargarse los operadores binarios nunca podrá hacerse lo mismo con los unarios ya que su único parámetro sólo puede ser de un único tipo (el tipo dentro del que se defina) Además, ello también provoca que no pueden redefinirse las conversiones ya incluidas en la BCL porque al menos uno de los operandos siempre habrá de ser de algún nuevo tipo definido por el usuario.

- No puede alterarse sus reglas de precedencia, asociatividad, ubicación y número de operandos, pues si ya de por sí es difícil para muchos recordarlas cuando son fijas, mucho más lo sería si pudiesen modificarse según los tipos de sus operandos.

- No puede definirse nuevos operadores ni combinaciones de los ya existentes con nuevos significados (por ejemplo ** para representar exponenciación), pues ello complicaría innecesariamente el compilador, el lenguaje y la legibilidad del código cuando en realidad es algo que puede simularse definiendo métodos.

- No todos los operadores incluidos en el lenguaje pueden redefinirse, pues muchos de ellos (como ., new, =, etc.) son básicos para el lenguaje y su redefinición es inviable, poco útil o dificultaría innecesariamente la legibilidad del código. Además, no todos los redefinibles se redefinen usando la sintaxis general hasta ahora vista, aunque en su momento se irán explicando cuáles son los redefinibles y cuáles son las peculiaridades de aquellos que requieran una redefinición especial.

A continuación se muestra cómo se redefiniría el significado del operador **+** para los objetos Complejo del ejemplo anterior:

```
class Complejo;
{
        public float ParteReal;
        public float ParteImaginaria;

        public Complejo (float parteReal, float parteImaginaria)
        {
                this.ParteReal = parteReal;
                this.ParteImaginaria = parteImaginaria;
        }
        public static Complejo operator +(Complejo op1, Complejo op2)
        {
                Complejo resultado = new Complejo();
                resultado.ParteReal = op1.ParteReal + op2.ParteReal;
                resultado.ParteImaginaria = op1.ParteImaginaria + op2.ParteImaginaria;
                return resultado;
        }
}
```

Es fácil ver que lo que en el ejemplo se ha redefinido es el significado del operador **+** para que cuando se aplique entre dos objetos de clase Complejo devuelva un nuevo objeto Complejo cuyas partes real e imaginaria sea la suma de las de sus operandos.

Redefinición de operadores unarios

Los únicos operadores unarios redefinibles son: **!**, **+**, **-**, **~**, **++**, **--**, **true** y **false**, y toda redefinición de un operador unario ha de tomar un único parámetro que ha de ser del mismo tipo que el tipo de dato al que pertenezca la redefinición.

Los operadores ++ y -- siempre ha de redefinirse de manera que el tipo de dato del objeto devuelto sea el mismo que el tipo de dato donde se definen. Cuando se usen de forma prefija se devolverá ese objeto, y cuando se usen de forma postifja el compilador lo que hará será devolver el objeto original que se les pasó como parámetro en lugar del indicado en el return. Por ello es importante no modificar dicho parámetro si es de un tipo referencia y queremos que estos operadores tengan su significado tradicional. Un ejemplo de cómo hacerlo es la siguiente redefinición de ++ para el tipo Complejo:

```
public static Complejo operator ++ (Complejo op)
{
        Complejo resultado = new Complejo(op.ParteReal + 1, op.ParteImaginaria);
        return resultado;
}
```

Nótese que si hubiésemos redefinido el **++** de esta otra forma:

```
public static Complejo operator ++ (Complejo op)
{
        op.ParteReal++;
        Return;                                                          op;
}
```

Entonces el resultado devuelto al aplicárselo a un objeto siempre sería el mismo tanto si fue aplicado de forma prefija como si lo fue de forma postifija, ya que en ambos casos el objeto devuelto sería el mismo. Sin embargo, eso no ocurriría si Complejo fuese una estructura, ya que entonces op no sería el objeto original sino una copia de éste y los cambios que se le hiciesen en el cuerpo de la redefinición de **++** no afectarían al objeto original, que es el que se devuelve cuando se usa **++** de manera postfija.

Respecto a los operadores **true** y **false**, estos indican respectivamente, cuando se ha de considerar que un objeto representa el valor lógico cierto y cuando se ha de considerar que representa el valor lógico falso, por lo que sus redefiniciones siempre han de devolver un objeto de tipo **bool** que indique dicha situación. Además, si se redefine uno es obligatorio redefinir también el otro, pues siempre es posible usar indistintamente uno u otro para determinar el valor lógico que un objeto de ese tipo represente.

En realidad los operadores **true** y **false** no pueden usarse directamente en el código fuente, sino que redefinirlos para un tipo de dato es útil porque permiten utilizar objetos de ese tipo en expresiones condicionales tal y como si de un valor lógico se tratase. Por ejemplo, podemos redefinir estos operadores en el tipo Complejo de modo que consideren cierto a todo complejo distinto de 0 + 0i y falso a 0 + 0i:

```
public static bool operator true(Complejo op)
{
        return (op.ParteReal != 0 op.ParteImaginaria != 0);
}

public static bool operator false(Complejo op)
{
        return (op.ParteReal == 0 && op.ParteImaginaria == 0);
}
```

Con estas redefiniciones, un código como el que sigue mostraría por pantalla el mensaje Es cierto:

```
Complejo c1 = new Complejo(1, 0); // c1 = 1 + 0i

        if (c1)
                System.Console.WriteLine("Es cierto");
```

Redefinición de operadores binarios

Los operadores binarios redefinibles son **+**, **-**, *****, **/**, **%**, **&**, , **^**, **<<**, **>>**, **==**, **!=**, **>**, **<**, **>=** y **<=** Toda redefinición que se haga de ellos ha de tomar dos parámetros tales que al menos uno sea del mismo tipo que el tipo de dato del que es miembro la redefinición.

Hay que tener en cuenta que aquellos de estos operadores que tengan complementario siempre han de redefinirse junto con éste. Es decir, siempre que se redefina en un tipo el operador **>** también ha de redefinirse en él el operador **<**, siempre que se redefina **>=** ha de redefinirse **<=**, y siempre que se redefina **==** ha de redefinirse **!=**.

También hay que señalar que, como puede deducirse de la lista de operadores binarios redefinibles dada, no es redefinir directamente ni el operador de asignación = ni los operadores compuestos (**+=**, **-=**, etc.) Sin embargo, en el caso de estos últimos dicha redefinición ocurre de manera automática al redefinir su parte "no **=**" Es decir, al redefinir **+** quedará redefinido consecuentemente **+=**, al redefinir ***** lo hará ***=**, etc.

Por otra parte, también cabe señalar que no es posible redefinir directamente los operadores && y . Esto se debe a que el compilador los trata de una manera especial que consiste en evaluarlos perezosamente. Sin embargo, es posible simular su redefinición redefiniendo los operadores unarios true y false, los operadores binarios & y y teniendo en cuenta que && y se evalúan así:

- **&&**: Si tenemos una expresión de la forma x && y, se aplica primero el operador **false** a x. Si devuelve **false**, entonces x && y devuelve el resultado de evaluar x; y si no, entonces devuelve el resultado de evaluar x & y.
- : Si tenemos una expresión de la forma x y, se aplica primero el operador **true** a x. Si devuelve **true**, se devuelve el resultado de evaluar x; y si no, se devuelve el de evaluar x y.