# 4.3.1 Conversiones Boxing

Una conversión boxing permite que un tipo de valor (*value-type*) se convierta implícitamente en un tipo de referencia (*reference-type*). Existen los siguientes tipos de conversiones boxing:

- De cualquier tipo de valor (incluidos los tipos enum) al tipo `object`.
- De cualquier tipo de valor (incluidos los tipos enum) al tipo `System.ValueType`.
- De cualquier tipo de valor a cualquier tipo de interfaz (*interface-type*) implementada por el tipo de valor.
- De cualquier tipo enum al tipo `System.Enum`.

La conversión boxing de un valor a un tipo de valor consiste en asignar una instancia del objeto y después copiar en ella el valor del tipo de valor.

El proceso real de conversión boxing del valor de un tipo de valor se entiende mejor si uno se imagina la existencia de una clase boxing para el tipo. Para cualquier tipo de valor `T`, la clase boxing se comporta como si se declarara de la manera siguiente:

```
sealed class T_Box: System.ValueType
{
   T value;
   public T_Box(T t) {
      value = t;
   }
}
```

La conversión boxing de un valor `v` de tipo `T` consiste ahora en ejecutar la expresión `new T_Box(v)` y devolver la instancia resultante como un valor de tipo `object`. Por lo tanto, las instrucciones

```
int i = 123;
object box = i;
```

conceptualmente se corresponden con

```
int i = 123;
object box = new int_Box(i);
```

La conversión boxing de clases como `T_Box` e `int_Box` mencionadas anteriormente no existe en realidad y el tipo dinámico de un valor al que se ha aplicado la conversión boxing no es realmente un tipo de clase. En lugar de ello, un valor convertido mediante boxing de tipo `T` tiene el tipo dinámico `T`, y una comprobación tipo dinámica que usa el operador `is` sencillamente puede hacer referencia al tipo `T`. Con el siguiente ejemplo:

```
int i = 123;
object box = i;
if (box is int) {
   Console.Write("Box contains an int");
}
```

se devuelve la cadena "`Box contains an int`" en la consola.

Una conversión boxing implica la creación de una copia del valor al que se aplica la conversión. Esto es distinto de la conversión de un tipo de referencia a un tipo `object`, en la cual el valor sigue haciendo referencia a la misma instancia y sencillamente se considera como el tipo `object` menos derivado. Por ejemplo, dada la declaración

```
struct Point
{
   public int x, y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
}
```

las siguientes instrucciones

```
Point p = new Point(10, 10);
object box = p;
p.x = 20;
Console.Write(((Point)box).x);
```

muestran en la consola el valor 10, porque la operación boxing implícita que ocurre en la asignación de `p` a `box` causa la copia del valor de `p`. En cambio, si se hubiera declarado `Point` como `class`, el resultado sería 20, puesto que `p` y `box` harían referencia al mismo objeto.

---



Especificación del lenguaje C#

# 4.3.2 Conversiones Unboxing

Una conversión unboxing permite que un tipo de referencia (*reference-type*) se convierta explícitamente en un tipo de valor (*value-type)*. Existen los siguientes tipos de conversiones unboxing:

- Del tipo `object` a cualquier tipo de valor (*value-type*), incluidos los tipos enum (*enum-type*).
- Del tipo `System.ValueType` a cualquier tipo de valor (*value-type*), incluidos los tipos enum (*enum-type*).
- De cualquier tipo de interfaz (*interface-type*) a cualquier tipo de valor que implemente el tipo de interfaz.
- Del tipo `System.Enum` a cualquier tipo enum (*enum-type*).

Una operación unboxing consiste en comprobar primero que la instancia del objeto es un valor al que se ha aplicado la conversión boxing del tipo de valor dado, y copiar después el valor fuera de la instancia.

En cuanto a la clase boxing imaginaria descrita en la sección anterior, una conversión unboxing de un objeto `box` a un tipo de valor `T` consiste en ejecutar la expresión `((T_Box)box).value`. Por lo tanto, las instrucciones

```
object box = 123;
int i = (int)box;
```

conceptualmente se corresponden con

```
object box = new int_Box(123);
int i = ((int_Box)box).value;
```

Para que una conversión unboxing a un tipo de valor dado se ejecute correctamente en tiempo de ejecución, el valor del operando de origen debe ser una referencia a un objeto que se creó anteriormente mediante una conversión boxing de un valor de ese tipo de valor. Si el operando de origen tiene valor `null`, se producirá una excepción `System.NullReferenceException`. Si el operando de origen es una referencia a un objeto incompatible, se producirá una excepción `System.InvalidCastException`.

---





# 6.4 Conversiones definidas por el usuario

C# permite la ampliación de las conversiones explícitas e implícitas predefinidas mediante conversiones definidas por el usuario. Las conversiones definidas por el usuario se introducen mediante la declaración de operadores de conversión (Sección 10.9.3) en tipos de clase y struct.

# 6.4.1 Conversiones permitidas definidas por el usuario

C# sólo permite la declaración de algunas conversiones definidas por el usuario. En concreto, no es posible redefinir una conversión explícita o implícita ya existente. Una clase o estructura tiene permitido declarar una conversión de un tipo de origen `S` a un tipo de destino `T` solamente si son verdaderos todos los siguientes:

- `S` y `T` son tipos diferentes
- `S` o `T` es el tipo de clase o estructura en el que tiene lugar la declaración del operador.
- Ni `S` ni `T` son de tipo `object` o de tipo de interfaz.
- `T` no es una clase base de `S`, y `S` tampoco lo es de `T`.

Las restricciones aplicables a las conversiones definidas por el usuario se explican en la Sección 10.9.3.





# 6.4.2 Evaluación de conversiones definidas por el usuario

Una conversión definida por el usuario convierte un valor de su tipo, denominado tipo de origen, a otro tipo, denominado tipo de destino. La evaluación de una conversión definida por el usuario se centra en descubrir el operador de conversión definido por el usuario más

específico para los tipos de origen y de destino concretos. Esta determinación se divide en varios pasos:

- Buscar el conjunto de clases y estructuras a partir del cual se consideran los operadores de conversión definida por el usuario. Este conjunto consta del tipo de origen y sus clases base y el tipo de destino y sus clases base (con los supuestos implícitos de que sólo las clases y estructuras pueden declarar operadores definidos por el usuario y de que los tipos no de clase no tienen clases base).
- A partir del conjunto de tipos, determinar qué operadores de conversión definida por el usuario son aplicables. Para que un operador de conversión sea aplicable, debe ser posible realizar una conversión estándar (Sección 6.3) del tipo de origen al tipo de operando del operador, y debe ser posible realizar una conversión estándar del tipo del resultado del operador al tipo de destino.
- A partir del conjunto de operadores definidos por el usuario que puedan aplicarse, determinar qué operador es el más específico sin ninguna ambigüedad. En términos generales, el operador más específico es aquél cuyo tipo de operando es el "más próximo" al tipo de origen y cuyo tipo de resultado es el "más próximo" al tipo de destino. En las próximas secciones se definen las reglas exactas para establecer el operador de conversión definido por el usuario más específico.

Una vez identificado un operador de conversión definido por el usuario más específico, la ejecución de la conversión definida por el usuario implica hasta tres pasos:

- Primero, si se requiere, una conversión estándar del tipo de origen al tipo de operando del operador de conversión definido por el usuario.
- Después, invocar al operador de conversión definido por el usuario para que realice la conversión.
- Por último, si se requiere, realizar una conversión estándar del tipo del resultado del operador de conversión definido por el usuario al tipo de destino.

La evaluación de una conversión definida por el usuario nunca necesita más de un operador de conversión definido por el usuario. Esto es, una conversión del tipo `S` al tipo `T` nunca ejecuta en primer lugar una conversión definida por el usuario de `S` a `X` y después una conversión definida por el usuario de `X` a `T`.

En las próximas secciones se ofrecen las definiciones exactas de la evaluación de conversiones implícitas o explícitas definidas por el usuario. En las definiciones se usan los siguientes términos:

- Si existe una conversión implícita estándar (Sección 6.3.1) de un tipo `A` a un tipo `B`, y si ni `A` ni `B` son tipos de interfaz, entonces se dice que `A` está abarcado por `B`, y que `B` abarca a `A`.
- El tipo que más abarca de un conjunto de tipos es aquél que abarca todos los demás tipos del conjunto. Si ninguno de los tipos abarca a todos los demás, entonces el conjunto no tiene tipo que más abarca. En términos más intuitivos, el tipo que más

abarca es el "más grande" del conjunto, el tipo al que pueden convertirse implícitamente todos los demás tipos.
- El tipo más abarcado de un conjunto de tipos es aquél al que abarcan todos los demás tipos del conjunto. Si ninguno de los tipos es abarcado por todos los demás, entonces el conjunto no tiene un tipo más abarcado. En términos más intuitivos, el tipo más abarcado es el "más pequeño" del conjunto, aquél que puede convertirse implícitamente a todos los demás tipos.

---



Especificación del lenguaje C#

# 6.4.3 Conversiones implícitas definidas por el usuario

Una conversión implícita definida por el usuario del tipo `S` al tipo `T` se procesa como sigue:

- Se busca el conjunto de tipos, `D`, a partir del cual se consideran los operadores de conversión definida por el usuario. Este conjunto está formado por `S` (si `S` es una clase o una estructura), las clases base de `S` (si `S` es una clase) y `T` (si `T` es una clase o una estructura).
- Se busca el conjunto de operadores de conversión definida por el usuario aplicables, `U`. Este conjunto consta de los operadores de conversión implícita definida por el usuario declarados por las clases o estructuras de `D` que convierten de un tipo incluyente `S` a un tipo abarcado por `T`. Si `U` está vacía, la conversión no estará definida y se producirá un error en tiempo de compilación.
- Se busca el tipo de origen más específico, `SX`, de los operadores de `U`:
  - Si uno de los operadores de `U` se convierte desde `S`, entonces `SX` es `S`.
  - De lo contrario, `SX` es el tipo más abarcado del conjunto combinado de tipos de destino de los operadores de `U`. Si no se encuentra un tipo más abarcado, la conversión será ambigua y se producirá un error en tiempo de compilación.
- Se busca el tipo de destino más específico, `TX`, de los operadores de `U`:
  - Si uno de los operadores de `U` se convierte a `T`, entonces `TX` es `T`.
  - De lo contrario, `TX` es el tipo más incluyente del conjunto combinado de tipos de destino de los operadores de `U`. Si no se encuentra un tipo más incluyente, la conversión será ambigua y se producirá un error en tiempo de compilación.

- Si `U` contiene exactamente un operador de conversión definido por el usuario que convierte de `SX` a `TX`, éste es el operador de conversión más específico. Si no existe tal operador o si existen varios, la conversión será ambigua y se producirá un error en tiempo de compilación. De lo contrario, se aplica la conversión definida por el usuario:
    - Si `S` no es `SX`, se realiza una conversión implícita estándar de `S` a `SX`.
    - Se llama al operador de conversión definido por el usuario más específico para convertir de `SX` a `TX`.
    - Si `TX` no es `T`, se realiza una conversión implícita estándar de `TX` a `T`.

---



Especificación del lenguaje C#

# 6.4.4 Conversiones explícitas definidas por el usuario

Una conversión explícita definida por el usuario del tipo `S` al tipo `T` se procesa como sigue:

- Se busca el conjunto de tipos, `D`, a partir del cual se consideran los operadores de conversión definida por el usuario. Este conjunto consta de `S` (si `S` es una clase o estructura), las clases base de `S` (si `S` es una clase), `T` (si `T` es una clase o estructura) y las clases base de `T` (si `T` es una clase).
- Se busca el conjunto de operadores de conversión definida por el usuario aplicables, `U`. Este conjunto consta de los operadores de conversión implícita o explícita definidos por el usuario, declarados por las clases o estructuras de `D` que convierten de un tipo incluyente o abarcado por `S` a un tipo incluyente o abarcado por `S`. Si `U` está vacía, la conversión no estará definida y se producirá un error en tiempo de compilación.
- Se busca el tipo de origen más específico, `SX`, de los operadores de `U`:
    - Si uno de los operadores de `U` se convierte desde `S`, entonces `SX` es `S`.
    - De lo contrario, si uno de los operadores de `U` convierte de los tipos que abarca `S`, entonces `SX` es el tipo más abarcado del conjunto combinado de tipos de origen de estos operadores. Si no se encuentra un tipo más abarcado, la conversión será ambigua y se producirá un error en tiempo de compilación.
    - De lo contrario, `SX` es el tipo más incluyente del conjunto combinado de tipos de origen de los operadores de `U`. Si no se encuentra un tipo más

incluyente, la conversión será ambigua y se producirá un error en tiempo de compilación.

- Se busca el tipo de destino más específico, `TX`, de los operadores de `U`:
  - Si uno de los operadores de `U` se convierte a `T`, entonces `TX` es `T`.
  - De lo contrario, si uno de los operadores de `U` convierte a los tipos abarcados por `T`, entonces `TX` es el tipo más incluyente del conjunto combinado de tipos de origen de estos operadores. Si no se encuentra un tipo más incluyente, la conversión será ambigua y se producirá un error en tiempo de compilación.
  - De lo contrario, `TX` es el tipo más abarcado del conjunto combinado de tipos de destino de los operadores de `U`. Si no se encuentra un tipo más abarcado, la conversión será ambigua y se producirá un error en tiempo de compilación.
- Si `U` contiene exactamente un operador de conversión definido por el usuario que convierte de `SX` a `TX`, éste es el operador de conversión más específico. Si no existe tal operador o si existen varios, la conversión será ambigua y se producirá un error en tiempo de compilación. De lo contrario, se aplica la conversión definida por el usuario:
  - Si `S` no es `SX`, se realiza una conversión explícita estándar de `S` a `SX`.
  - Se llama al operador de conversión definido por el usuario más específico para convertir de `SX` a `TX`.
  - Si `TX` no es `T`, se realiza una conversión explícita estándar de `TX` a `T`.

---