Tipos Genéricos

17

Programación II y Laboratorio de Computación II

Edición 2018

Generics

- Generics es el mecanismo de implementación de clases parametrizadas introducido en la versión 2.0 del lenguaje C#.

- Una clase parametrizada es exactamente igual a una clase de las habituales, salvo por un pequeño detalle: su definición contiene algún elemento que depende de un parámetro que debe ser especificado en el momento de la declaración de un objeto de dicha clase.

Generics

- Esto puede resultar extremadamente útil a la hora de programar clases genéricas, capaces de implementar un tipado fuerte sin necesidad de conocer a priori los tipos para los que serán utilizadas.

- List es una clase parametrizada:

```
List<Parametro> l = new List<Parametro>();
```

- Dictonary es otro ejemplo, con dos parámetros:

```
Dictionary<int, string> m = new Dictionary<int, string>();
```

Generics – Uso simple

```
public class Mensaje<T>
{
    private T miAtributo;
}

// ...
Mensaje<string> tipoTexto = new Mensaje<string>();
Mensaje<MiClase> tipoMio = new Mensaje<MiClase>();
```

Generics – Uso menos simple

```
public class Mensajero<T, U>
{
    private T miAtr1;
    private U miAtr2;
    private Dictionary<T, U> miDiccionario;
}

Mensaje<string, int> tipoTexto = new Mensaje<string, int>();
Mensaje<char, MiClase> tipoMio = new Mensaje<char, MiClase>();
```

Restricciones

- Una buena regla consiste en aplicar el mayor número de restricciones posible que siga permitiendo manejar los tipos que se deben utilizar.

```
public class Mensajero<T> where T : Mensaje
{ }
class EjemploComplejo<K, V, U>
    // Implemente interfaz
    where U : System.IComparable<U>
    // V tenga constructor por defecto
    where V : new()
{ }
```

Restricciones

Restricción	Descripción
where T : struct	El argumento de tipo debe ser un tipo de valor.
where T : class	El argumento de tipo debe ser un tipo de referencia.
where T : unmanaged	El argumento de tipo no debe ser un tipo de referencia y no debe contener ningún miembro de tipo de referencia en ningún nivel de anidamiento.
where T : new()	El argumento de tipo debe tener un constructor sin parámetros público. Cuando se usa conjuntamente con otras restricciones, la restricción new() debe especificarse en último lugar.
where T : <nombre de clase base>	El argumento de tipo debe ser o derivarse de la clase base especificada.
where T : <nombre de interfaz>	El argumento de tipo debe ser o implementar la interfaz especificada. Pueden especificarse varias restricciones de interfaz. La interfaz de restricciones también puede ser genérica.
where T : U	El argumento de tipo proporcionado por T debe ser o derivarse del argumento proporcionado para U.

Restricciones

- Algunas de estas restricciones son mutuamente excluyentes.
- Todos los tipos de valor deben tener un constructor sin parámetros accesible.
- La restricción struct implica la restricción new() y la restricción new() no se puede combinar con la restricción struct.
- La restricción unmanaged implica la restricción struct.
- La restricción unmanaged no se puede combinar con las restricciones struct o new().

Métodos y Generics

- Para que un método sea genérico, no hace falta que la clase también lo sea:

```
class Prueba
{
    public static void OpTest<T>(T s, T t) where T : class
    {
        System.Console.WriteLine(s == t);
    }

    public void OpTest2<T>(T s, T t) where T : class
    {
        System.Console.WriteLine(s == t);
    }
}
```