

Clases genéricas (Guía de programación de C#)

19/07/2015 • Tiempo de lectura: 3 minutos • 

En este artículo

[Vea también](#)

Las clases genéricas encapsulan operaciones que no son específicas de un tipo de datos determinado. El uso más común de las clases genéricas es con colecciones como listas vinculadas, tablas hash, pilas, colas y árboles, entre otros. Las operaciones como la adición y eliminación de elementos de la colección se realizan básicamente de la misma manera independientemente del tipo de datos que se almacenan.

Para la mayoría de los escenarios que necesitan clases de colección, el enfoque recomendado es usar las que se proporcionan en la biblioteca de clases .NET. Para más información sobre el uso de estas clases, vea [Colecciones genéricas en .NET](#).

Normalmente, crea clases genéricas empezando con una clase concreta existente, y cambiando tipos en parámetros de tipo de uno en uno hasta que alcanza el equilibrio óptimo de generalización y facilidad de uso. Al crear sus propias clases genéricas, entre las consideraciones importantes se incluyen las siguientes:

- Los tipos que se van a generalizar en parámetros de tipo.

Como norma, cuantos más tipos pueda parametrizar, más flexible y reutilizable será su código. En cambio, demasiada generalización puede crear código que sea difícil de leer o entender para otros desarrolladores.

- Las restricciones, si existen, que se van a aplicar a los parámetros de tipo (Vea [Restricciones de parámetros de tipo](#)).

Una buena norma es aplicar el máximo número de restricciones posible que todavía le permitan tratar los tipos que debe controlar. Por ejemplo, si sabe que su clase genérica está diseñada para usarse solo con tipos de referencia, aplique la restricción de clase. Esto evitará el uso no previsto de su clase con tipos de valor, y le permitirá usar el operador `as` en `T`, y comprobar si hay valores NULL.

- Si separar el comportamiento genérico en clases base y subclases.

Como las clases genéricas pueden servir como clases base, las mismas consideraciones de diseño se aplican aquí con clases no genéricas. Vea las reglas sobre cómo heredar de clases base genéricas posteriormente en este tema.

- Si implementar una o más interfaces genéricas.

Por ejemplo, si está diseñando una clase que se usará para crear elementos en una colección basada en genéricos, puede que tenga que implementar una interfaz como [IComparable<T>](#) donde `T` es el tipo de su clase.

Para obtener un ejemplo de una clase genérica simple, vea [Introducción a los genéricos](#).

Las reglas para los parámetros de tipo y las restricciones tienen varias implicaciones para el comportamiento de clase genérico, especialmente respecto a la herencia y a la accesibilidad de miembros. Antes de continuar, debe entender algunos términos. Para una clase genérica `Node<T>`, el código de cliente puede hacer referencia a la clase especificando un argumento de tipo, para crear un tipo construido cerrado (`Node<int>`). De manera alternativa, puede dejar el parámetro de tipo sin especificar, por ejemplo cuando especifica una clase base genérica, para crear un tipo construido abierto (`Node<T>`). Las clases genéricas pueden heredar de determinadas clases base construidas abiertas o construidas cerradas:

C#

Copiar

```
class BaseNode { }
class BaseNodeGeneric<T> { }

// concrete type
class NodeConcrete<T> : BaseNode { }

//closed constructed type
class NodeClosed<T> : BaseNodeGeneric<int> { }

//open constructed type
class NodeOpen<T> : BaseNodeGeneric<T> { }
```

Las clases no genéricas, en otras palabras, concretas, pueden heredar de clases base construidas cerradas, pero no desde clases construidas abiertas ni desde parámetros de tipo porque no hay ninguna manera en tiempo de ejecución para que el código de cliente proporcione el argumento de tipo necesario para crear instancias de la clase base.

C#

Copiar

```
//No error
class Node1 : BaseNodeGeneric<int> { }

//Generates an error
//class Node2 : BaseNodeGeneric<T> {}

//Generates an error
//class Node3 : T {}
```

Las clases genéricas que heredan de tipos construidos abiertos deben proporcionar argumentos de tipo para cualquier parámetro de tipo de clase base que no se comparta mediante la clase heredada, como se demuestra en el código siguiente:

C#

Copiar


```
class BaseNodeMultiple<T, U> { }

//No error
class Node4<T> : BaseNodeMultiple<T, int> { }

//No error
class Node5<T, U> : BaseNodeMultiple<T, U> { }

//Generates an error
//class Node6<T> : BaseNodeMultiple<T, U> {}
```


Las clases genéricas que heredan de tipos construidos abiertos deben especificar restricciones que son un superconjunto de las restricciones del tipo base, o que las implican:

C#	
<pre>class NodeItem<T> where T : System.IComparable<T>, new() { } class SpecialNodeItem<T> : NodeItem<T> where T : System.IComparable<T>, new() { }</pre>	

Los tipos genéricos pueden usar varios parámetros de tipo y restricciones, de la manera siguiente:

C#	
<pre>class SuperKeyType<K, V, U> where U : System.IComparable<U> where V : new() { }</pre>	

Tipos construidos cerrados y construidos abiertos pueden usarse como parámetros de método:

C#	
<pre>void Swap<T>(List<T> list1, List<T> list2) { //code to swap items } void Swap(List<int> list1, List<int> list2) { //code to swap items }</pre>	

Si una clase genérica implementa una interfaz, todas las instancias de esa clase se pueden convertir en esa interfaz.

Las clases genéricas son invariables. En otras palabras, si un parámetro de entrada especifica un `List<BaseClass>`, obtendrá un error en tiempo de compilación si intenta proporcionar un `List<DerivedClass>`.

Vea también

- [System.Collections.Generic](#)
- [Guía de programación de C#](#)
- [Genéricos](#)
- [Guardar el estado de los enumeradores](#)
- [Un puzle de herencia, parte uno](#)

¿Le ha resultado útil esta página?

 Sí  No

