

**Instituto Técnico Nacional de Comercio**

**INCOS 3 QUILLACOLLO**

TEMA 3

RESTRICCIONES DE TIPOS DE PARÁMETROS

**Alumnos:**

Vallejos Santa Maria Roberto Rafael

Villca Condori Elmer

**Curso:**

Tercer año de sistemas

**Materia:**

Programación III

**Docente:**

Lic. Condori Oscar

## Abril 2024.

**Restricciones de tipos de parámetros**

1. **Introduccioón**

Las restricciones de tipos de parámetros son reglas que le dicen al compilador qué características debe tener un argumento de tipo específico cuando se utiliza en clases o métodos genéricos. Sin estas restricciones, el argumento de tipo puede ser cualquier tipo de datos, lo que puede llevar a comportamientos impredecibles o errores en el código. Sin restricciones, el compilador solo puede suponer los miembros definidos en la clase base fundamental de todos los tipos en .NET, que es System.Object. Esto limita la capacidad del compilador para inferir o garantizar características específicas del tipo de datos utilizado como argumento en el código genérico.

1. **Antecedentes**

Antes de la introducción de las restricciones de tipos de parámetros, en versiones anteriores de C# 6.0 (Visual Studio 2015), la sintaxis y las capacidades de C# para definir restricciones sobre los tipos de parámetros en clases y métodos genéricos eran más limitadas.

Aunque C# ya permitía la programación genérica mediante el uso de clases y métodos genéricos, no había una manera directa y efectiva de especificar qué tipos podían ser utilizados como argumentos en esas clases y métodos genéricos.

Sin las restricciones de tipos de parámetros, la seguridad y la claridad del código podían verse comprometidas. Los desarrolladores tenían que confiar en comentarios o documentación para indicar qué tipos eran aceptables como argumentos en clases y métodos genéricos, lo que dificultaba la comprensión del código y podía conducir a errores sutiles.

1. **Definición / Concepto**

Las restricciones de tipos de parámetros en C# son reglas que se pueden aplicar a los tipos de datos que se pueden usar como argumentos en clases y métodos genéricos. Estas reglas permiten especificar qué tipos de datos son válidos, lo que ayuda a garantizar un comportamiento predecible y seguro en el código genérico.

1. **Tipos de restricciones**

|  |  |  |
| --- | --- | --- |
| **Restricción** | **Descripción** | **Version C#** |
| **where T : class** | El argumento de tipo debe ser un tipo de referencia. Esta restricción se aplica también a cualquier clase, interfaz, delegado o tipo de matriz. En un contexto que acepta valores NULL, “T” debe ser un tipo de referencia que no acepta valores NULL. | C# 6.0  V.S 2015 |
| **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. La restricción “new()” no se puede combinar con las restricciones “struct” ni “unmanaged”. | C# 6.0  V.S 2015 |
| **where T : <nombre de clase base>** | El argumento de tipo debe ser o derivarse de la clase base especificada. En un contexto que acepta valores NULL, “T” debe ser un tipo de referencia que no acepta valores NULL derivado de la clase base especificada. | C# 6.0  V.S 2015 |
| **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. En un contexto que acepta valores NULL, “T” debe ser un tipo que no acepta valores NULL que implementa la interfaz especificada. | C# 6.0  V.S. 2015 |
| **where T : U** | El argumento de tipo proporcionado por T debe ser o se debe derivar del argumento proporcionado para U. En un contexto que acepta valores NULL, si U es un tipo de referencia que no acepta valores NULL, T debe ser un tipo de referencia que no acepta valores NULL. Si U es un tipo de referencia que acepta valores NULL, T puede ser que acepta valores NULL o que no aceptan valores NULL. | C# 7.0  V.S. 2017 |
| **where T : default** | Esta restricción resuelve la ambigüedad cuando es necesario especificar un parámetro de tipo sin restricciones al invalidar un método o proporcionar una implementación de interfaz explícita. La restricción default implica el método base sin la restricción class o struct. | C# 7.0  V.S. 2017 |
| **where T : struct** | El argumento type debe ser un tipo de valor que no acepta valores NULL, que incluye record struct tipos. Dado que todos los tipos de valor tienen un constructor sin parámetros accesible, declarado o implícito, la restricción struct implica la restricción new() y no se puede combinar con la restricción new(). No puede combinar la restricción struct con la restricción unmanaged. | C# 7.3  V.S. 2017 |
| **where T : unmanaged** | El argumento de tipo debe ser un tipo no administrado que no acepta valores NULL. La restricción unmanaged implica la restricción struct y no se puede combinar con las restricciones struct ni new(). | C# 7.3  V.S. 2017 |
| **where T : class?** | El argumento de tipo debe ser un tipo de referencia, que acepte o no valores NULL. Esta restricción también se aplica a cualquier clase, interfaz, delegado o tipo de matriz, incluidos los registros. | C# 8.0  V.S. 2019 |
| **where T : notnull** | El argumento de tipo debe ser un tipo que no acepta valores NULL. El argumento puede ser un tipo de referencia que no acepta valores NULL, o bien un tipo de valor que no acepta valores NULL. | C# 8.0  V.S. 2019 |

1. **Implementación**

**5.1. Restricción where T : class**: Esta restricción garantiza que el tipo de parámetro T sea una clase de referencia.

// Clase base abstracta

public abstract class Animal

{

public abstract void HacerSonido();

}

// Clase de implementación que hereda de la clase base

public class Perro : Animal

{

public override void HacerSonido()

{

Console.WriteLine("Guau guau");

}

}

// Clase genérica con restricción 'where T : class'

public class ContenedorClase<T> where T : class

{

private T \_elemento;

public ContenedorClase(T elemento)

{

\_elemento = elemento;

}

public void MostrarInformacion()

{

Console.WriteLine($"El tipo del elemento es: {typeof(T)}");

}

}

class Program

{

static void Main(string[] args)

{

// Crear una instancia de ContenedorClase con un tipo que cumpla la restricción 'where T : class'

ContenedorClase<Perro> contenedorPerro = new ContenedorClase<Perro>(new Perro());

contenedorPerro.MostrarInformacion();

}

}

**5.2. Restricción where T : new():** Esta restricción garantiza que el tipo de parámetro T tenga un constructor público sin parámetros.

// Clase sin constructor público sin parámetros

public class Persona

{

public string Nombre { get; set; }

public Persona(string nombre)

{

Nombre = nombre;

}

}

// Clase genérica con restricción 'where T : new()'

public class ContenedorNew<T> where T : new()

{

private T \_instancia;

public ContenedorNew()

{

\_instancia = new T();

}

public void MostrarNombre()

{

// Intentamos acceder a una propiedad específica del tipo T

// Solo funcionará si T tiene una propiedad llamada 'Nombre'

Console.WriteLine($"Nombre: {\_instancia.GetType().GetProperty("Nombre").GetValue(\_instancia)}");

}

}

class Program

{

static void Main(string[] args)

{

// Crear una instancia de ContenedorNew con un tipo que cumpla la restricción 'where T : new()'

ContenedorNew<Persona> contenedorPersona = new ContenedorNew<Persona>();

contenedorPersona.MostrarNombre();

}

}

**5.3. Restricción where T : <nombre de clase base>:** Esta restricción garantiza que el tipo de parámetro T sea o herede de una clase base específica.

// Clase base

public class Fruta

{

public virtual string ObtenerColor()

{

return "Desconocido";

}

}

// Clase de implementación que hereda de la clase base

public class Manzana : Fruta

{

public override string ObtenerColor()

{

return "Rojo";

}

}

// Clase genérica con restricción 'where T : <nombre de clase base>'

public class ContenedorBase<T> where T : Fruta

{

private T \_fruta;

public ContenedorBase(T fruta)

{

\_fruta = fruta;

}

public void MostrarColor()

{

Console.WriteLine($"El color de la fruta es: {\_fruta.ObtenerColor()}");

}

}

class Program

{

static void Main(string[] args)

{

// Crear una instancia de ContenedorBase con un tipo que cumpla la restricción 'where T : <nombre de clase base>'

ContenedorBase<Manzana> contenedorManzana = new ContenedorBase<Manzana>(new Manzana());

contenedorManzana.MostrarColor();

}

}

**5.4. Restricción where T : <nombre de interfaz>:** Esta restricción garantiza que el tipo de parámetro T implemente una interfaz específica.

// Interfaz

public interface IVehiculo

{

void Arrancar();

void Detener();

}

// Clase que implementa la interfaz

public class Automovil : IVehiculo

{

public void Arrancar()

{

Console.WriteLine("El automóvil ha arrancado.");

}

public void Detener()

{

Console.WriteLine("El automóvil se ha detenido.");

}

}

// Clase genérica con restricción 'where T : <nombre de interfaz>'

public class ContenedorInterfaz<T> where T : IVehiculo

{

private T \_vehiculo;

public ContenedorInterfaz(T vehiculo)

{

\_vehiculo = vehiculo;

}

public void RealizarOperaciones()

{

\_vehiculo.Arrancar();

// Aquí podrían realizarse más operaciones relacionadas con vehículos

\_vehiculo.Detener();

}

}

class Program

{

static void Main(string[] args)

{

// Crear una instancia de ContenedorInterfaz con un tipo que cumpla la restricción 'where T : <nombre de interfaz>'

ContenedorInterfaz<Automovil> contenedorAutomovil = new ContenedorInterfaz<Automovil>(new Automovil());

contenedorAutomovil.RealizarOperaciones();

}

}

1. **Conclusiones**

Las restricciones de tipos de parámetros en C# es una herramienta que permite escribir código más seguro, claro y reutilizable al limitar los tipos que pueden ser utilizados como argumentos en clases y métodos genéricos. Al imponer restricciones sobre los tipos de parámetros, los desarrolladores pueden garantizar ciertas propiedades o capacidades en los tipos utilizados, lo que reduce la posibilidad de errores y facilita la comprensión del código.

1. **Bibliografia**

BillWagner. (2024, March 11). Restricciones de tipos de parámetros: Guía de programación de C# - C#. Retrieved April 8, 2024, from Microsoft.com website: https://learn.microsoft.com/es-es/dotnet/csharp/programming-guide/generics/constraints-on-type-parameters