**Restricciones de Tipos Genéricos en C#**

1. **¿Qué son las restricciones de tipos genéricos?**

Son reglas que limitan los tipos de datos que se pueden usar como argumentos de tipo en una clase, interfaz, método o delegado genérico.

1. **¿Para qué sirven?**

* Garantizan que el parámetro genérico cumpla ciertas condiciones.
* Permiten usar en el código operaciones seguras (por ejemplo, crear instancias con new T(), acceder a métodos de una clase base o interfaz, etc.).
* Evitan errores en tiempo de compilación asegurando que el tipo sea válido.

**Tabla 1.**

*Resumen de las restricciones de tipos genéricos en C#*

| Restricción | Sintaxis | Descripción | Ejemplo de uso |
| --- | --- | --- | --- |
| Referencia | where T : class | El tipo debe ser una clase (tipo por referencia). | public class Repositorio<T> where T : class { } |
| Valor | where T : struct | El tipo debe ser un tipo valor (int, bool, DateTime, etc.), no nullable. | public class Contenedor<T> where T : struct { } |
| Constructor sin parámetros | where T : new() | El tipo debe tener un constructor público sin parámetros. | public T CrearInstancia<T>() where T : new() { return new T(); } |
| Clase base | where T : MiClaseBase | El tipo debe heredar de una clase específica. | public class Servicio<T> where T : EntidadBase { } |
| Interfaz | where T : IInterfaz | El tipo debe implementar una interfaz específica. | public class Validador<T> where T : IValidable { } |
| Múltiples restricciones | where T : ClaseBase, IInterfaz, new() | Combinación de varias restricciones: clase base, interfaz y constructor. | public class Manager<T> where T : EntidadBase, IValidable, new() { } |
| Entre genéricos | where T : U | El tipo T debe ser igual o derivado de otro parámetro genérico U. | public class Comparador<T, U> where T : U { } |

1. **Ejemplos:**

public class Servicio<T> where T : EntidadBase, IValidable, new()

{

public T CrearInstancia()

{

return new T();

}

}

Aquí T debe:

* Heredar de EntidadBase.
* Implementar la interfaz IValidable.
* Tener un constructor sin parámetros (Llamado también constructor implícito).
  1. **Ejemplo completo:**

**Crea un proyecto con las siguientes clases:**

namespace \_04\_Generic

{

// Interfaz para calcular bono

public interface ICalculateBonus

{

public decimal CalculateBonus();

}

}

namespace \_04\_Generic

{

// Clase base para todos los empleados

public abstract class EmployeeBase

{

public int Id { get; set; }

public string? Name { get; set; }

public abstract void DisplayInfo();

}

}

namespace \_04\_Generic

{

// Empleado de tiempo completo

public class FullTimeEmployee : EmployeeBase, ICalculateBonus

{

public decimal Salary { get; set; }

public override void DisplayInfo()

{

Console.WriteLine($"[Empleado de tiempo Completo] ID: {Id}, Nombre: {Name}, Salario: {Salary:C}");

}

public decimal CalculateBonus()

{

return Salary \* 0.10m;

}

}

}

namespace \_04\_Generic

{

// Empleado de medio tiempo

public class PartTimeEmployee : EmployeeBase, ICalculateBonus

{

public decimal HourlyRate { get; set; }

public int HoursWorked { get; set; }

public override void DisplayInfo()

{

Console.WriteLine($"[Empleado de medio tiempo] ID: {Id}, Nombre: {Name}, Horas: {HoursWorked}, Tarifa: {HourlyRate:C}");

}

public decimal CalculateBonus()

{

return HourlyRate \* HoursWorked \* 0.05m;

}

}

}

namespace \_04\_Generic

{

/\*

\* Repositorio genérico con restricciones

\*

\*/

// EmployeeBase: T debe heredar(o ser) de la clase base EmployeeBase.

// Esto garantiza que T tenga al menos las propiedades y métodos definidos en EmployeeBase(por ejemplo, Id, Name).

// ICalculateBonus: T debe implementar la interfaz ICalculateBonus.

// Esto asegura que T tenga el método CalculateBonus().

// new (): T debe tener un constructor público sin parámetros.

// Esto permite que dentro del repositorio se pueda hacer new T() para crear instancias dinámicamente.

public class EmployeeRepository<T> where T : EmployeeBase, ICalculateBonus, new()

{

private List<T> employees = new List<T>();

public void Add(T employee)

{

employees.Add(employee);

Console.WriteLine($"Empleado {employee.Name} agregado al repositorio.");

}

public void DisplayAll()

{

Console.WriteLine("\n--- Lista de Empleados ---");

foreach (var emp in employees)

{

emp.DisplayInfo();

Console.WriteLine($"Bono: {emp.CalculateBonus():C}");

}

}

}

}

namespace \_04\_Generic

{

class Program

{

static void Main()

{

var repo = new EmployeeRepository<FullTimeEmployee>();

var fullTime = new FullTimeEmployee

{

Id = 1,

Name = "Juan Perez",

Salary = 3000m

};

var partTime = new PartTimeEmployee

{

Id = 2,

Name = "Maria Lopez",

HourlyRate = 50000m,

HoursWorked = 80

};

// Repositorio de tiempo completo

repo.Add(fullTime);

// Repositorio de medio tiempo

var repo2 = new EmployeeRepository<PartTimeEmployee>();

repo2.Add(partTime);

// Mostrar

repo.DisplayAll();

repo2.DisplayAll();

}

}

}

La clase genérica EmployeeRespository declarada como:

public class EmployeeRepository<T> where T

: EmployeeBase, ICalculateBonus, new()

Impone la siguiente restricción:

1. public class EmployeeRepository<T>  
   Declara una **clase genérica** llamada EmployeeRepository que trabaja con un **parámetro de tipo genérico** T.
2. where T : EmployeeBase, ICalculateBonus, new()  
   Esto es una **restricción genérica**, y significa que el tipo T que se use debe cumplir estas condiciones:
   * EmployeeBase: T debe heredar (o ser) de la clase base EmployeeBase.  
     Esto garantiza que T tenga al menos las propiedades y métodos definidos en EmployeeBase (por ejemplo, Id, Name).
   * ICalculateBonus: T debe implementar la interfaz ICalculateBonus.  
     Esto asegura que T tenga el método CalculateBonus().
   * new(): T debe tener un constructor público sin parámetros.  
     Esto permite que dentro del repositorio se pueda hacer new T() para crear instancias dinámicamente.