Generic Constraints

We have already seen generic constraints in the previous article. Now let's see how to use them in here. We know that there are six constraints. They are,

***where T : class***

[Class](https://www.dotnetfunda.com/articles/show/2959/class-object-and-class-inheritance-in-csharp) is a [reference type](https://www.dotnetfunda.com/articles/show/2933/conversions-cts-value-and-reference-types-in-csharp). So it is a reference type constraint. it can be applied to any [class](https://www.dotnetfunda.com/articles/show/2959/class-object-and-class-inheritance-in-csharp), [interface](https://www.dotnetfunda.com/articles/show/2976/interface-explicit-interface-implementations-in-csharp), delegate and [array type](https://www.dotnetfunda.com/articles/show/2947/arrays-declaration-and-default-initialization-of-arrays-in-csharp).

***Example***,

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Example  
{  
    class Program  
    {  
        class MyClass  
        {  
            public void Test<T>()  
                where T : class     // Generic Constraint  
            {  
                Console.WriteLine("Hello"); // Prints Hello  
            }  
        }  
          
        static void Main()  
        {  
            MyClass my = new MyClass();  
            my.Test<string>();  
        }  
    }  
}

In the above example, we have used generic constraint ***where T : class***. If you run this code, you might see the following word in your console,



***where T : struct***

[Struct](https://www.dotnetfunda.com/articles/show/2973/struct-properties-of-structs-in-csharp) is a [value type](https://www.dotnetfunda.com/articles/show/2933/conversions-cts-value-and-reference-types-in-csharp). So this is a value type constraint. It can be applied to any types except to a **nullable** type.

***Example***,

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Example  
{  
    class Program  
    {  
        class Exam<T>  
            where T : struct      // Generic Constraint struct  
        {  
            public void Test()  
            {  
                Console.WriteLine("Public Exam"); // Public Exam  
            }  
        }  
          
        static void Main()  
        {  
            Exam<int> exam = new Exam<int>();  
            exam.Test();  
        }  
    }  
}

In the above example, we have a generic constraint ***where T : struct***. It can be applied to any type except nullable.

And we have created an instance in the **Main()** method and calling the generic constraint from there. If you run this one, you will see the following output in your console,



***where T : interface***

It is an [interface](https://www.dotnetfunda.com/articles/show/2976/interface-explicit-interface-implementations-in-csharp) constraint. It can be applied to a type  that must implement the specified interface.

***Example***,

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Data;  
  
namespace Example  
{  
    class Program  
    {  
        class Data<T>  
            where T : IDisposable     // Generic Constraint interface  
        {  
            public void Test()  
            {  
                Console.WriteLine("Data Table"); // Data Table  
            }  
        }  
          
        static void Main()  
        {  
            Data<DataTable> d = new Data<DataTable>();  
            d.Test();  
        }  
    }  
}

In the above code, we are using a generic interface constraint. **DataTable** (derived from **System.Data**) is implementing the **IDisposable** (interface).

Now, press Ctrl + F5 to see the following output in your console,



We can also specify multiple generic interface constraints.

***where T : base class***

The name itself tells that it can be applied to the types that must be in the base class or derived from the base class.

***Example***,

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Example  
{  
    class Program  
    {  
        public class Base { }  
  
        public class Derived<T>  
            where T : Base          // base class generic constraint  
        {  
            public void Test()  
            {  
                Console.WriteLine("Derived Class"); // Derived Class  
            }  
        }  
          
        static void Main()  
        {  
            Derived<Base> s = new Derived<Base>();  // creating instance  
            s.Test();  
        }  
    }  
}

In the above code, we have generic base class constraint. In the **Main()** method, we have created instance with base class support.

Run this code in your console to get printed like below,



***where T : new()***

It is applied to the type which have a public parameter-less constructor. Hence it is known as the parameter-less constructor constraint.

***Example***,

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Example  
{  
    class Program  
    {  
        // default parameter-less constructor  
        class MyClass<T>  
            where T : new()       // Generic parameter-less constructor constraint  
        {  
            public void Test()  
            {  
                Console.WriteLine("This is MyClass method");  
            }  
        }  
          
        static void Main()  
        {  
            MyClass<Program> my = new MyClass<Program>();  
            my.Test();  
        }  
    }  
}

In the above code, we are using **new()** constraint. In order to use this generic constraint, we need a public parameter-less constructor.

In the **Main()** method, we have an instance **MyClass<Program>**. As we know that **Program** is a class with parameter-less constructor. Hence it is used with **MyClass**.

The output of the above code will be,



**new()** constraint should be specified last when the type is used together with other constraints. That is,

class MyClass<T>  
    where : Base, new() {