Generics in C#

In c#, **generic** is a type used to define a class, structure, interface, or method with **placeholders** (type parameters) to indicate that they can store or use one or more of the types. In c#, the compiler will replace placeholders with the specified type at compile time.

In c#, mostly, we will use generics with collections and the methods that operate on them to specify a type of objects to store in a collection. The generics were introduced in .NET Framework 2.0 with a new namespace called `System.Collections.Generic`.

In c#, generics are useful for improving code reusability, type safety, and performance compared with non-generic types such as arraylist.

C# Generics Declaration

To define a class or method as generic, we need to use a type parameter as a placeholder with angle (**<>**) brackets.

Following is the example of defining a generic class with type parameter (**T**) as a placeholder with an angle (**<>**) brackets.

```
public class GenericClass<T>
{
   public T msg;
   public void genericMethod(T name, T location)
   {
      Console.WriteLine("{0}", msg);
      Console.WriteLine("Name: {0}", name);
      Console.WriteLine("Location: {0}", location);
   }
}
```

If you observe the above class, we created a class (**GenericClass**) with one parameter (**msg**) and method (genericMethod) using type parameter (**T**) as a placeholder with an angle (**<>**) brackets.

Here, the angle (**<>**) brackets will indicate a **GenericClass** is generic, and type parameter (**T**) is used to accept a requested type. The type parameter name can be anything like **X** or **U**, etc., based on our requirements.

Generally, while creating an instance of the class, we need to specify an actual type, then the compiler will replace all the type parameters such as **T** or **U** or **X**, etc., with a specified actual type. In c#, the following is the example of creating an instance of a generic class.

```
// Instantiate Generic Class, string is the type argument
GenericClass<string> gclass = new GenericClass<string>();
gclass.msg = "Welcome to Tutlane";
gclass.genericMethod("Suresh Dasari", "Hyderabad");
```

If you observe the above code, we are sending a type as “**string**” so the compiler will substitute all the type parameters (**T**) with defined type “**string**” and our class (**GenericClass**) will be like as shown below.

```
public class GenericClass
{
   public string msg;
   public void genericMethod(string name, string location)
   {
      Console.WriteLine("{0}", msg);
      Console.WriteLine("Name: {0}", name);
      Console.WriteLine("Location: {0}", location);
   }
}
```

In c#, we can also create our custom generic types and methods to provide our generalized solutions that are type-safe and efficient.

C# Generic Class Example

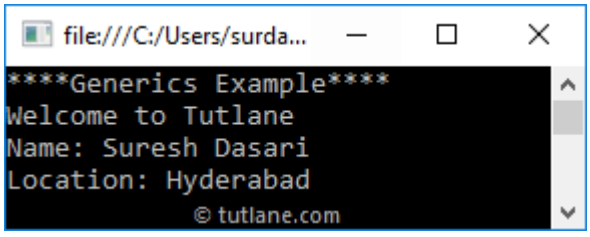
Following is the example of creating a generic class using type parameter (**T**) with angle (**<>**) brackets in c# programming language.

```
using System;

namespace Tutlane
{
   public class GenericClass<T>
   {
      public T msg;
      public void genericMethod(T name, T location)
      {
         Console.WriteLine("{0}", msg);
         Console.WriteLine("Name: {0}", name);
         Console.WriteLine("Location: {0}", location);
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("****Generics Example****");
         // Instantiate Generic Class, string is the type argument
```

```
            GenericClass<string> gclass = new GenericClass<string>();
            gclass.msg = "Welcome to Tutlane";
            gclass.genericMethod("Suresh Dasari", "Hyderabad");
            Console.ReadLine();
        }
    }
}
```

When you execute the above c# program, you will get the result below.



This is how we can use generics in c# to create generic classes or methods based on our requirements.

C# Generic Class as Base / Derived Class

In c#, you can use the generic class as a base class, but we need to provide a type instead of a type parameter for the base class because there is no way to send a required type argument to instantiate a base class at run time.

Following is the example of using a generic class as a base class in the c# programming language.

```
// No Error
class DClass1 : GenericClass<string> {
// implementation
}
// Compile Time Error
//class DClass2 : GenericClass<T> {
// implementation
//}
```

If the derived class is generic, we don't need to specify a type for the generic base class. Instead, we can use the type parameter (**T**).

Following is the example of defining a generic derived class in the c# programming language.

```
// No Error
class DClass1 : GenericClass<string> {
// implementation
}
// No Error
class DClass2<T> : GenericClass<T> {
// implementation
}
```

C# Generic Methods

In c#, if we define a method with a type parameter, it is called a **generic method**. Following is the example of defining a generic method with a type parameter using angle (**<>**) brackets.

```
public void genericMethod<T>(T a, T b)
{
// Implementation
}
```

This generic method can be called either by specifying the type of argument or without an argument, as shown below.

```
genericMethod<int>(1, 2);

//or


genericMethod(1, 2);
```

C# Generic Method Example

In c#, you can call a generic method by passing any type of argument. Following is the example of defining a generic method in the c# programming language.

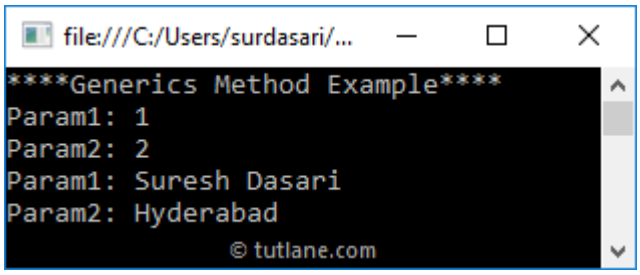
```
using System;

namespace Tutlane
{
    public class SampleClass
    {
        public void GMethod<T>(T a, T b)
        {
            Console.WriteLine("Param1: {0}", a);
            Console.WriteLine("Param2: {0}", b);
        }
    }
    class Program
    {
      static void Main(string[] args)
      {
          Console.WriteLine("****Generics Method Example****");
          SampleClass s = new SampleClass();
          s.GMethod<int>(1, 2);
          s.GMethod("Suresh Dasari", "Hyderabad");
          Console.ReadLine();
      }
```

```
    }
}
```

If you observe the above code, we call our generic method (**GMethod**) with or without type parameters and send different types of arguments based on our requirements.

When you execute the above c# program, you will get the result below.



This is how you can define generic methods in the c# programming language based on our requirements.

C# Generic Delegates

In c#, a generic delegate will be same as a normal delegate, but the only difference is a generic delegate will have a generic type with angle (<>) brackets.

Following is the example of defining a generic delegate in the c# programming language.

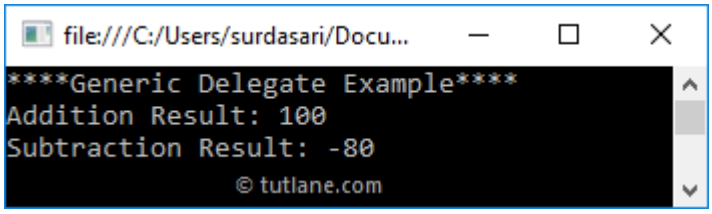
```
using System;

namespace Tutlane
{
   // Declare Generic Delegate
   public delegate T SampleDelegate<T>(T a, T b);
   class MathOperations
   {
      public int Add(int a, int b)
      {
         return a + b;
      }
      public int Subtract(int x, int y)
      {
         return x - y;
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("****Generic Delegate Example****");
         MathOperations m = new MathOperations();
         // Instantiate delegate with add method
```

```
        SampleDelegate<int> dlgt = new SampleDelegate<int>(m.Add);
        Console.WriteLine("Addition Result: " + dlgt(10, 90));
        // Instantiate delegate with subtract method
        dlgt = m.Subtract;
        Console.WriteLine("Subtraction Result: " + dlgt(10, 90));
        Console.ReadLine();
    }
  }
}
```

If you observe the above code, we defined a delegate (**SampleDelegate**) with generic type parameter (**T**) using angle (**<>**) brackets and accessing it by creating an instance of delegate with a required argument (**int**).

When you execute the above c# program, you will get a result, as shown below.



This is how we can use generics with delegates based on our requirements in the c# programming language.

C# Generics Overview

The following are the important properties of generics in the c# programming language.

- In c#, generics are represented by using angle bracket **<>**.
- To define a class or method as generic, we need to use a type parameter as a placeholder with angle (**<>**) brackets.
- The compiler will replace all the placeholders with the specified type at compile time.
- In c#, generics are useful for improving the code reusability, type safety, and performance compared with non-generic types such as arraylist.
- In c#, you can use generics with interfaces, classes, methods, properties, delegates, events, and operators.