Generics

Generics is a new concept in .net 2005 and this is similar to templates in C++. By using generics, you can create generic methods and generic classes.

Generic Methods

A generic method is a method that can accept any type of arguments. For most of the algorithms, logic is same whatever the type of data is. Hence instead of creating one method for every data type, you can create only one method that can accept any type of arguments, which is called as generic method. To create a method as generic method, syntax is as follows.

```
[Access Modifier] returntype MethodName<GenericTypes>([Parameters])
{

}
```

After creating a generic method, to call the generic method syntax is as follows.

MethodName<DataType>([Arguments])

Example : The following example creates a method Swap() that swaps two values as a generic method so that it can swap any type of values.

```
namespace Generics
{
    class Program
    {
        public static void Swap<MyType>(ref MyType X, ref MyType Y)
        {
            MyType T;
            T = X;
            X = Y;
            Y = T;
        }
        static void Main(string[] args)
        {
            int A = 10, B = 20;
            float F1 = 3.5F, F2 = 5.5F;
            string S1 = "Naresh", S2 = "Microsoft";
            Swap<int>(ref A, ref B);
            Swap<float>(ref F1, ref F2);
            Swap<string>(ref S1, ref S2);
            Console.WriteLine("After Swap ...");
            Console.WriteLine("A = {0}\tB = {1}", A, B);
            Console.WriteLine("F1 = {0}\tF2 = {1}", F1, F2);
            Console.WriteLine("S1 = {0}\tS2 = {1}", S1, S2);
        }
    }
}
```

Comparison of Generic Types

Generic type will be treated as a user defined type and hence operators like >,<,>= and <= will not work on generic types. When you have to perform comparison of generic types, then inherit the generic type from **Icomparable** interface that provides a method called **CompareTo().** This method can be used to compare generic types. For inheriting generic type from **Icomparable** interface use the keyword **where** at the end of the method declaration.

[Access Modifier] returntype MethodName<GT>([Parameters]) where GT : Icomparable

{

}

Example : The following example creates a method Sort() as generic method that can sort any type of array.

```
namespace Generics
{
    class GenSort
    {
        public static void Sort<MyType>(params MyType[] A) where MyType :
IComparable
        {
            MyType T;
            for (int i = 0; i < A.GetLength(0); i++)
            {
                for (int j = i + 1; j < A.GetLength(0); j++)
                {
                    if (A[i].CompareTo(A[j]) > 0)
                    {
                        T = A[i];
                        A[i] = A[j];
                        A[j] = T;
                    }
                }
            }
        }
        static void Main()
        {
            int[] A = { 23, 9, 17, 12, 3 };
            float[] F = { 27.32F, 56.66F, 32.21F, 17.76F, 21.21F };
            string[] S = { "Microsoft", "Sun", "Oracle", "Adobe", "Ibm" };
            Sort<int>(A);
            Sort<float>(F);
            Sort<string>(S);
            Console.WriteLine("After Sort Values In Arrays Are...");
            foreach (int n in A)
            {
                Console.Write("{0}\t", n);
            }
            Console.WriteLine();
            foreach (float n in F)
            {
                Console.Write("{0}\t", n);
            }
            Console.WriteLine();
            foreach (string n in S)
            {
                Console.WriteLine(n);
            }
        }
    }
}
```

Multiple Generic Types

There may be a situation where you have to take more than one parameter for a generic method that are of different type. In this case generic method can be created with more than one generic type. Syntax is as follows.

[Access Modifier] returntype MethodName<GT1,GT2,…>([Parameters])
{

}

After creating a generic method with multiple generic types, to call the generic method syntax is as follows.

MethodName<DataType1,DataType2,…>([Arguments])

Example : The following example creates a method with the name Print() as a generic method that accepts two parameters of two different types.

```
namespace Generics
{
    class MultiGen
    {
        public static void Print<MyType1, MyType2>(MyType1 X, MyType2 Y)
        {
            Console.WriteLine("X = {0}\tY = {1}", X, Y);
        }
        static void Main()
        {
            Print<int, float>(10, 14.5F);
            Print<float, string>(23.34F, "Naresh");
            Print<string, string>("Microsoft", "Naresh");
        }
    }
}
```

Generic Classes

Not only methods but classes can also be created as generic classes. A generic class is a class whose instances can work each on a different type of data. For creating a class as generic class syntax is as follows.

[Access Modifier] class classname<generictypes>
{

}

After creating generic class, to create an instance for the generic class, syntax is as follows.

Classname<Datatypes> ObjName = new classname<datatypes>();

Example : The following example creates a class with the name Test as generic class

```
namespace Generics
{
    class Test<MyType>
    {
        MyType A, B;
        public Test(MyType X, MyType Y)
        {
            A = X;
            B = Y;
        }
        public void Print()
        {
            Console.WriteLine("A = {0}\tB = {1}", A, B);
        }
    }
    class GenricClass
    {
        static void Main()
        {
            Test<int> T1 = new Test<int>(10, 20);
            Test<float> T2 = new Test<float>(3.5F, 4.5F);
            Test<string> T3 = new Test<string>("C#.Net", "VB.Net");
            T1.Print();
            T2.Print();
            T3.Print();
        }
    }
}
```

Within a generic class if you have to perform comparison of generic type, inherit the generic type from **Icomparable** interface with the help of **where** keyword at the end of class declaration.

```
[Access Modifier] class classname<GT> where GT : Icomparable
{

}
```

A generic class can also be created with more than one generic type same as a generic method and has the following syntax.

```
[Access Modifier] class classname<GT1,GT2,…>
{

}
```

When a generic class is created with more than one generic type then an instance is created for that class as follows.

```
Classname<DT1,DT2,…> Objname = new classname<DT1,DT2,…>();
```