Generics in C#

There are two ways in which we can perform programming as

1. Specific to the requirement.
2. Generic way of programming.

- Generics allow for designing a classes and methods whose types are specified only at the time of declaration and instantiation.
- This enables development of universal classes and methods that help in improving performance, productivity and type-safety.
- Generics are often used in creating collection classes for implementing concepts such as lists, hash tables, queues, etc.
- These classes manage a set of objects and encapsulate operations that are not specific to a particular data type.
- Generics are also known as parametric polymorphism.
- Generics were introduced in C# 2.0 as a part of Common Language Runtime of .NET to overcome the limitation in implementing generalization in earlier versions.
- Generalization was accomplished by casting types to and from universal base type, System.Object that was not type-safe and required casting, which results in a hit to performance.

- Some of the benefits in using generics include:
    - Casting is not required for accessing each element in the collection
    - Client code that uses generics is type-safe during execution by preventing the usage of data whose type is different from the one used in the declaration
    - Code is not duplicated for multiple types of data

- The List<T> collection class is an example for generic class provided in the .NET Framework Class library that can be used to add, delete and search an item of any type (T) that is passed as parameter to it.
- When the List class is instantiated with a type parameter in the client code, it would be similar to a concrete class implemented with same type.
- Generics are similar to C++ templates in concept but differ mainly in implementation.

There are two ways in which we can use the concept of generics as

1.Generic Method

2.Generic Class

Generic Method :

In this case only method is generic but class is normal class.

Example :

If we want to write application which is used to swap two elements then we have to write separate method for each data type as

// Specific Programming approach

```
using System;

public class Marvellous
{
	public void SwapInt(ref int no1, ref int no2)
	{
		int temp;
		temp = no1;
		no1 = no2;
		no2 = temp;
	}

	public void SwapFloat(ref float no1, ref float no2)
	{
		float temp;
		temp = no1;
		no1 = no2;
		no2 = temp;
	}

	public void SwapChar(ref char no1, ref char no2)
	{
		char temp;
		temp = no1;
		no1 = no2;
		no2 = temp;
	}
```

```
        public void SwapDouble(ref double no1, ref double no2)
        {
                double temp;
                temp = no1;
                no1 = no2;
                no2 = temp;
        }
}

public class Infosystems
{
        public static void Main(String[] arg)
        {
                int i1 = 11,i2 = 21;

                char ch1 = 'a', ch2 = 'b';

                float f1 = 3.14f,f2 = 6.14f;

                double d1 = 8.0,d2 = 9.0;

                Marvellous mobj = new Marvellous();

                Console.WriteLine("Before Swap integers {0} & {1}",i1,i2);
                mobj.SwapInt(ref i1,ref i2);
                Console.WriteLine("After Swap integers {0} & {1}",i1,i2);

                Console.WriteLine("Before Swap Characters {0} & {1}",ch1,ch2);
                mobj.SwapChar(ref ch1,ref ch2);
                Console.WriteLine("After Swap Characters {0} & {1}",ch1,ch2);

                Console.WriteLine("Before Swap Floats {0} & {1}",f1,f2);
                mobj.SwapFloat(ref f1,ref f2);
                Console.WriteLine("After Swap Floats {0} & {1}",f1,f2);

                Console.WriteLine("Before Swap doubles {0} & {1}",d1,d2);
                mobj.SwapDouble(ref d1,ref d2);
                Console.WriteLine("After Swap doubles {0} & {1}",d1,d2);
        }
}
```

In above application we observe that logic inside all Swap functions is same but we have to write multiple definitions for each data types separately.

We can avoid that overhead by using the concept of Generics as

// Generic Programming approach

```
using System;
using System.Threading;

public class Marvellous
{
	public void Swap <T>(ref T no1, ref T no2)
	{
		T temp;
		temp = no1;
		no1 = no2;
		no2 = temp;
	}
}

public class Infosystems
{
	public static void Main(String[] arg)
	{
		int i1 = 11, i2 = 21;

		char ch1 = 'a', ch2 = 'b';

		float f1 = 3.14f, f2 = 6.14f;

		double d1 = 8.0, d2 = 9.0;

		Marvellous mobj = new Marvellous();

		Console.WriteLine("Before Swap inntegers {0} & {1}",i1,i2);
		mobj.Swap(ref i1,ref i2);
		Console.WriteLine("After Swap integers {0} & {1}",i1,i2);

		Console.WriteLine("Before Swap Characters {0} & {1}",ch1,ch2);
		mobj.Swap(ref ch1,ref ch2);
		Console.WriteLine("After Swap Characters {0} & {1}",ch1,ch2);
```

```
            Console.WriteLine("Before Swap Floats {0} & {1}",f1,f2);
            mobj.Swap(ref f1,ref f2);
            Console.WriteLine("After Swap Floats {0} & {1}",f1,f2);

            Console.WriteLine("Before Swap doubles {0} & {1}",d1,d2);
            mobj.Swap(ref d1,ref d2);
            Console.WriteLine("After Swap doubles {0} & {1}",d1,d2);
        }
}
```

In above application we just change the changing data type with the template argument i.e. T.

Generic Class

We have to write class which contains two characteristics and one method which is used to perform its addition.

```
public class MarvellousInt
{
        public int no1,no2;

        public GenMarvellous(int value1, int value2)
        {
                no1 = value1;
                no2 = value2;
        }

        public int Add()
        {
                return no1 + no2;
        }
}

public class MarvellousFloat
{
        public float no1,no2;

        public GenMarvellous(float value1, float value2)
        {
                no1 = value1;
```

```
            no2 = value2;
        }

        public float Add()
        {
            return no1 + no2;
        }
}

public class Infosystems
{
        public static void Main(String[] arg)
        {
            MarvellousInt iobj = new MarvellousInt(10,11);
            int iRet = iobj.Add();
            Console.WriteLine("Addition of two integers {0}",iRet);

            MarvellousFloat fobj = new MarvellousFloat(10.4f,11.2f);
            float fRet = fobj.Add();
            Console.WriteLine("Addition of two floats {0}",fRet);
        }
}
```

Inn above application we have to write two different classes for integer and float. But in below generic application we can write only one class.

```
// Generic Programming approach

using System;

public class GenMarvellous<T>
{
        public T no1,no2;

        public GenMarvellous(T value1, T value2)
        {
            no1 = value1;
            no2 = value2;
        }
```

```
        public T Add()
        {
                dynamic a = no1;
                dynamic b = no2;

                return a + b;
        }
}

public class Infosystems
{
        public static void Main(String[] arg)
        {
                GenMarvellous <int> iobj = new GenMarvellous<int>(10,11);
                int iRet = iobj.Add();
                Console.WriteLine("Addition of two integers {0}",iRet);

                GenMarvellous <float> fobj = new GenMarvellous<float>(10.4f,11.2f);
                float fRet = fobj.Add();
                Console.WriteLine("Addition of two floats {0}",fRet);
        }
}
```