Generics in C#

The Generics in C# are introduced as part of C# 2.0. The Generics in C# allows us to define classes and methods which are decoupled from the data type. In other words, we can say that the Generics allow us to create classes using angular brackets for the data type of its members. At compilation time, these angular brackets are going to be replaced with some specific data types.

The most common use of generics is to create collection classes. The .NET class library contains several generic collection classes in the System.Collections.Generic namespace. These should be used whenever possible instead of classes such as ArrayList in the System.Collections namespace. In C#, the Generics can be applied to: Class, Method, Interface, Abstract class, Static method, Property, Event, Delegates, and Operator. Generic classes may be constrained to enable access to methods on particular data types. Information on the types that are used in a generic data type may be obtained at run-time by using reflection.

Advantages of Generics in C#:

- It Increases the reusability of the code.
- The Generics are type-safe. We will get the compile-time error if we try to use a different type of data rather than the one we specified in the definition.
- We get better performance with Generics as it removes the possibilities of boxing and unboxing.

Example:

```
// type parameter T in angle brackets
public class GenericList<T>
{
    // The nested class is also generic on T.
    private class Node
    {
        // T used in non-generic constructor.
        public Node(T t)
        {
            next = null;
            data = t;
        }

        private Node next;
        public Node Next
        {
            get { return next; }
            set { next = value; }
        }

        // T as private member data type.
        private T data;

        // T as return type of property.
        public T Data
        {
            get { return data; }
            set { data = value; }
        }
    }

    private Node head;

    // constructor
    public GenericList()
    {
        head = null;
    }

    // T as method parameter type:
    public void AddHead(T t)
    {
        Node n = new Node(t);
        n.Next = head;
        head = n;
    }

    public IEnumerator<T> GetEnumerator()
    {
        Node current = head;

        while (current != null)
        {
            yield return current.Data;
            current = current.Next;
        }
    }
}
class TestGenericList
{
    static void Main()
    {
        // int is the type argument
        GenericList<int> list = new GenericList<int>();
        for (int x = 0; x < 10; x++)
        {
            list.AddHead(x);
        }
        foreach (int i in list)
        {
            System.Console.Write(i + " ");
        }
        System.Console.WriteLine("\nDone");
    }
}
// Output:
//    0 1 2 3 4 5 6 7 8 9
```