**Generic**:

Use generic types to maximize code reuse, type safety, and performance.

The most common use of generics is to create collection classes.

List<T> (In most cases), Dictionary<TKey,TValue>, Queue<T>, ICollection<T>,

++++++ ---

**Delegates** allow methods to be passed as parameters.

Delegates can be used to define callback methods.

Delegates can be chained together; for example, multiple methods can be called on a single event.

We need following 4 steps for a delegate

//**1)** Declare

delegate int NumberChanger(int n);

//**2)** some method/s for the delegate (should match the signature of declared delegate)

public static int AddNum(int p) {

num += p;

return num; }

//**3)**create delegate instances

NumberChanger nc1 = new NumberChanger(MethosForDelegate.AddNum);

//**4)** calling the methods using the delegate objects

nc1(25);

//**Step 3 if we use Func<>**) create delegate instances

Func<string, int, bool> FuncDelegateXyz = new NumberChanger(MethosForDelegate.AddNum);

//**Step 4** if we use Func<>) **)** calling the methods using the delegate objects

FuncDelegateXyz (27);

**Func**: Func is logically similar to base delegate implementation.

we need to provide the signature parameter & its return type.

First two parameters are the method input parameters. 3rd parameter (always the last parameter) is the out parameter which should be the output return type of the method.

Func is always used when you have return object or type from method.

If you have void method, you should be using **Action**.