Action and Func delegates in C#

In this article, i will be summarizing Action and Func delegates in C#; what Action and Func delegates are, the difference between Action and Func delegates, creating and implementing these delegate types in simple use cases.

In conclusion of this article, I will also give reasons why Action and Func delegates are preferred to custom delegates.

**Overview**

Action delegates are delegate methods that can accept parameter values and returns no value –the return type is usually a *void*.

Func delegates are delegate methods that can accept parameter values and also *returns value* –the return type can be of any type.

Action and Func delegates are ready-made generic delegate in .net framework (available from .NET 3.5 onward). They are type-safe functions pointers that can reference a method with the same method signature.

**Func Delegate Example**

//Creating Func delegate that calculates 15% interest rate

static Func<double, double > getInterestRate = amount => amount \* 0.15;

//Calling the Func delegate

double amount = 1000;

double rateAmount = getInterestRate(amount);

Console.WriteLine($"15% interest of {amount} is {rateAmount}");

**Action Delegate Example**

//creating method with parameters

static void Addition(int a, int b) { Console.WriteLine(a + b); }

//creating method without parameter

static void Addition() { Console.WriteLine( "Hello addition"); }

//using the Action delegate for the Addition methods

Action aditionDelegate = new Action(Addition);

aditionDelegate.Invoke();

Action<int, int> aditionDelegateWithParameters =

new Action<int, int>(Addition);

aditionDelegateWithParameters.Invoke(4,4); //accept 2 parameters

**Invoking Action Delegate Options**

In the Action delegate example, *aditionDelegate* delegate is executed by calling the *Invoke()* There is an option of invoking [or executing] the delegate by name only.

**Example**

static void ExecuteDelegateByName()

{

Action aditionDelegate = new Action(Addition);

aditionDelegate();

}

static void ExecuteDelegateByInvoke()

{

Action aditionDelegate = new Action(Addition);

aditionDelegate.Invoke();

}

**Creating Action and Func Delegates in a Context**

Action and Func can be defined or created in any context. That is, the definition and implementation can be anywhere within a class. They can be defined and implemented within a method or outside a method in a class.

**Eaxmple**

class Program

{

static void Main(string[] args)

{

}

//in a method context. Defined and implemented within a method

static void ExecuteDelegateAndFunc()

{

Action aditionDelegate = new Action(Addition);

aditionDelegate.Invoke();

Func<double, double> getInterestRate = amount => amount \* 0.15;

double amountForInterestRateCalculation = 1000;

Console.WriteLine(getInterestRate(amountForInterestRateCalculation));

}

//in a class context:

//can be invoked anywhere within this class

static Action AditionDelegate = new Action(Addition);

//accessible anywhere within this class

static Func<double, double> GetInterestRate = amount => amount \* 0.15;

//methods for delegate

static void Addition(int a, int b) { Console.WriteLine(a + b); }

static void Addition() { Console.WriteLine("Hello addition"); }

}

**Important Information**

* Func return type is the last parameter type. For example, the return type of this Func delegate *Func<double, string>* is string. It takes double type value and returns a string.
* Both Action and Func (Action<> and Func<>) has 16 generic overloads –they can accept 16 parameters.

**Custom Delegate example above Action Delegate Example compared**

//define custom delegate

delegate double GetInterestRateCustomDelegate (double amount);

//create instance of custom delegate and implement delegate method option

static GetInterestRateCustomDelegate getInterestRateCustomDelegate

= new GetInterestRateCustomDelegate(

delegate (double amount)

{

return amount \* 0.15;

});

//call or execute created instance of custom delegate

static void ExecuteInstantiatedCustomDelegate()

{

double amount = 1000;

double rateAmount = getInterestRateCustomDelegate(amount);

Console.WriteLine($" 15% interest of {amount} from custom delegate instantiated is {rateAmount}");

}

**Important Information**

If you are a fan of using pointers (lambda expressions), and also, for the sake of making custom delegate friendlier; you can use pointers to simplify your custom delegate and make your codes cleaner.

The following example shows how to use a pointer for your custom delegate;

**Example**

//define custom delegate

delegate double GetInterestRateCustomDelegate (double amount);

//using pointer (lambda expression option instead of

//creating instance of custom delegate

static GetInterestRateCustomDelegate getInterestRateCustomDelegatePointer = amount => amount \* 0.15;

//call or execute custom delegate pointer

static void ExecuteCustomDelegatePointer()

{

double amount = 1000;

double rateAmount = getInterestRateCustomDelegatePointer(amount);

Console.WriteLine($" 15% interest of {amount} from custom delegate pointer is {rateAmount}");

}

**Why use Delegate?**

Why use delegates if all of the above examples are just executing methods via defined delegates? Delegates are very useful and designed for *representation of objects and for asynchronous callbacks.*

**Conclusion**

It is recommended to use .net built-in generic delegates like Action and Func delegates for less code, code simplicity and code readability.

In the next article, I will explain with demonstrations how to use delegates for asynchronous callbacks.