F# - DELEGATES

A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime. F# delegates are similar to pointers to functions, in C or C++.

Declaring Delegates

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which have the same signature as that of the delegate.

Syntax for delegate declaration is −

```
type delegate-typename = delegate of type1 -> type2
```

For example, consider the delegates −

```
// Delegate1 works with tuple arguments.
type Delegate1 = delegate of (int * int) -> int
// Delegate2 works with curried arguments.
type Delegate2 = delegate of int * int -> int
```

Both the delegates can be used to reference any method that has two *int* parameters and returns an *int* type variable.

In the syntax −

- ***type1*** represents the argument types.
- ***type2*** represents the return type.

Please note −

- The argument types are automatically curried.
- Delegates can be attached to function values, and static or instance methods.
- F# function values can be passed directly as arguments to delegate constructors.
- For a static method the delegate is called by using the name of the class and the method. For an instance method, the name of the object instance and method is used.
- The Invoke method on the delegate type calls the encapsulated function.
- Also, delegates can be passed as function values by referencing the Invoke method name without the parentheses.

The following example demonstrates the concept −

Example

```
type Myclass() =
   static member add(a : int, b : int) =
      a + b
   static member sub (a : int) (b : int) =
      a - b
   member x.Add(a : int, b : int) =
      a + b
   member x.Sub(a : int) (b : int) =
      a - b

// Delegate1 works with tuple arguments.
type Delegate1 = delegate of (int * int) -> int
// Delegate2 works with curried arguments.
```

```
type Delegate2 = delegate of int * int -> int

let InvokeDelegate1 (dlg : Delegate1) (a : int) (b: int) =
   dlg.Invoke(a, b)
let InvokeDelegate2 (dlg : Delegate2) (a : int) (b: int) =
   dlg.Invoke(a, b)

// For static methods, use the class name, the dot operator, and the
// name of the static method.
let del1 : Delegate1 = new Delegate1( Myclass.add )
let del2 : Delegate2 = new Delegate2( Myclass.sub )

let mc = Myclass()
// For instance methods, use the instance value name, the dot operator, and the instance
method name.

let del3 : Delegate1 = new Delegate1( mc.Add )
let del4 : Delegate2 = new Delegate2( mc.Sub )

for (a, b) in [ (400, 200); (100, 45) ] do
   printfn "%d + %d = %d" a b (InvokeDelegate1 del1 a b)
   printfn "%d - %d = %d" a b (InvokeDelegate2 del2 a b)
   printfn "%d + %d = %d" a b (InvokeDelegate1 del3 a b)
   printfn "%d - %d = %d" a b (InvokeDelegate2 del4 a b)
```

When you compile and execute the program, it yields the following output –

```
400 + 200 = 600
400 - 200 = 200
400 + 200 = 600
400 - 200 = 200
100 + 45 = 145
100 - 45 = 55
100 + 45 = 145
100 - 45 = 55
```