F# - CLASSES

Classes are types that represent objects that can have properties, methods, and events. 'They are used to model actions, processes, and any conceptual entities in applications'.

Syntax

Syntax for defining a class type is as follows −

```
// Class definition:
type [access-modifier] type-name [type-params] [access-modifier] ( parameter-list ) [ as
identifier ] =
   [ class ]
      [ inherit base-type-name(base-constructor-args) ]
      [ let-bindings ]
      [ do-bindings ]
      member-list
      ...
   [ end ]

// Mutually recursive class definitions:
type [access-modifier] type-name1 ...
and [access-modifier] type-name2 ...
...
```

Where,

- The **type-name** is any valid identifier. Default access modifier for this is **public**.
- The **type-params** describes optional generic type parameters.
- The **parameter-list** describes constructor parameters. Default access modifier for primary constructor is **public**.
- The **identifier** used with the optional **as** keyword gives a name to the instance variable, or **self-identifier,** which can be used in the type definition to refer to the instance of the type.
- The **inherit** keyword allows you to specify the base class for a class.
- The **let** bindings allow you to declare fields or function values local to the class.
- The **do-bindings** section includes code to be executed upon object construction.
- The **member-list** consists of additional constructors, instance and static method declarations, interface declarations, abstract bindings, and property and event declarations.
- The keywords **class** and **end** that mark the start and end of the definition are optional.

Constructor of a Class

The constructor is code that creates an instance of the class type.

In F#, constructors work little differently than other .Net languages. In the class definition, the arguments of the primary constructor are described as parameter-list.

The body of the constructor consists of the **let** and **do** bindings.

You can add additional constructors by using the new keyword to add a member −

```
new (argument-list) = constructor-body
```

The following example illustrates the concept −

Example

The following program creates a line class along with a constructor that calculates the length of the line while an object of the class is created −

```
type Line = class
   val X1 : float
   val Y1 : float
   val X2 : float
   val Y2 : float

   new (x1, y1, x2, y2) as this =
      { X1 = x1; Y1 = y1; X2 = x2; Y2 = y2;}
      then
         printfn " Creating Line: {(%g, %g), (%g, %g)}\nLength: %g"
            this.X1 this.Y1 this.X2 this.Y2 this.Length

   member x.Length =
      let sqr x = x * x
      sqrt(sqr(x.X1 - x.X2) + sqr(x.Y1 - x.Y2) )
end
let aLine = new Line(1.0, 1.0, 4.0, 5.0)
```

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

```
Creating Line: {(1, 1), (4, 5)}
Length: 5
```

Let Bindings

The let bindings in a class definition allow you to define private fields and private functions for F# classes.

```
type Greetings(name) as gr =
   let data = name
   do
      gr.PrintMessage()
   member this.PrintMessage() =
      printf "Hello %s\n" data
let gtr = new Greetings("Zara")
```

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

```
Hello Zara
```

Please note the use of self-identifier *gr* for the *Greetings* class.