F# - GENERICS

Generics allow you to delay the specification of the data type of programming elements in a class or a method, until it is actually used in the program. In other words, generics allow you to write a class or method that can work with any data type.

You write the specifications for the class or the method, with substitute parameters for data types. When the compiler encounters a constructor for the class or a function call for the method, it generates code to handle the specific data type.

In F#, function values, methods, properties, and aggregate types such as classes, records, and discriminated unions can be generic.

Generic constructs contain at least one type parameter. Generic functions and types enable you to write code that works with a variety of types without repeating the code for each type.

Syntax

Syntax for writing a generic construct is as follows −

```
// Explicitly generic function.
let function-name<type-parameters> parameter-list =
   function-body

// Explicitly generic method.
[ static ] member object-identifer.method-name<type-parameters> parameter-list [ return-
type ] =
   method-body

// Explicitly generic class, record, interface, structure,
// or discriminated union.
type type-name<type-parameters> type-definition
```

Examples

```
(* Generic Function *)
let printFunc<'T> x y =
   printfn "%A, %A" x y

printFunc<float> 10.0 20.0
```

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

```
10.0, 20.0
```

You can also make a function generic by using the single quotation mark syntax −

```
(* Generic Function *)
let printFunction (x: 'a) (y: 'a) =
   printfn "%A %A" x y

printFunction 10.0 20.0
```

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

```
10.0 20.0
```

Please note that when you use generic functions or methods, you might not have to specify the type arguments. However, in case of an ambiguity, you can provide type arguments in angle brackets as we did in the first example.

If you have more than one type, then you separate multiple type arguments with commas.

Generic Class

Like generic functions, you can also write generic classes. The following example demonstrates this −

```
type genericClass<'a> (x: 'a) =
   do printfn "%A" x

let gr = new genericClass<string>("zara")
let gs = genericClass( seq { for i in 1 .. 10 -> (i, i*i) } )
```

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

```
"zara"
seq [(1, 1); (2, 4); (3, 9); (4, 16); ...]
```