```
'Code for Initializing objects and variables
        End Sub


'Other class members
End Class
```

You can call a constructor only from within the constructor of either the same class or a derived class. In derived classes, the first line in the constructor calls the constructor of the base class to initialize the inherited objects. After calling the constructor of the base class, you can add code to initialize the objects and variables created in the derived class.

In Visual Basic.NET, constructors can also take arguments. In other words, you can create parameterized constructors in Visual Basic.NET. Consider the following example:

```
Public Sub New(Optional ByVal iempcode As Integer = 0)
```

The preceding example defines a parameterized constructor for the `Employee` class. The constructor accepts the employee code, a unique identification number assigned to each employee, as an argument. Therefore, each time you create an object of the `Employee` class, you can assign a unique identification number to the employee. If you do not provide an identification number to the employee when creating the employee object, a default value of 0 is assigned.

You can use either of the following statements to create an object of the `Employee` class:

```
Dim Emp1 As New Employee(1001)
```

or

```
Dim Emp2 As Employee = New Employee(1001)
```

Constructors are optional procedures. You can also create classes without constructors. However, it is recommended that you initialize objects created in a class by using a constructor.

Now that you know how to create constructors in Visual Basic.NET, you will take a look at how to declare destructors in Visual Basic.NET.

Destructors

Visual Basic.NET also provides the `Sub Finalize` procedure, which acts as a destructor. The function of a destructor is opposite that of a constructor. A destructor releases the memory and resources used by a destroyed object. The `Sub Finalize` procedure is a protected method of the `Object` class. You can override the `Sub Finalize` procedure in the classes you create. Take a look at the following example:

```
Protected Overrides Sub Finalize()
         MyBase.Finalize()
```