Comencemos a programar con

VBA - Access

Entrega 07

Colecciones y Objetos

Eduardo Olaz

Introducción a las Colecciones

Una colección, siguiendo la definición de la ayuda de Access, es un conjunto ordenado de elementos, a los que se puede hacer referencia como una unidad.

El objeto **Colección**, recibe el nombre de **`Collection`**.

Una colección se parece a una matriz, en el sentido de que podemos acceder a sus elementos mediante un índice, pero aquí acaban prácticamente sus semejanzas.

Algunas de las ventajas de las colecciones frente a las matrices son:

- Utilizan menos memoria que las matrices. Sólo la que necesitan en cada caso.
- El acceso a los elementos de la colección es más flexible. Como veremos hay varios métodos para hacerlo. Eso no significa que sea más rápido.
- Las colecciones tienen métodos para añadir nuevos elementos y eliminar los que ya contiene.
- El tamaño de la colección se ajusta de forma automática tras añadir ó quitar algún miembro.
- Pueden contener simultáneamente elementos de diferentes tipos

Son como un saco en los que podemos meter “casi todo”. Una de las pocas cosas que no se pueden añadir a una colección son las estructuras **Registro** definidas por el usuario, que vimos en la entrega anterior.

Para poder meter en una matriz simultáneamente diferentes tipos de datos, habría que declararla como **`Variant`**, o lo que es lo mismo, no especificar su tipo en el momento de declararla.

En la declaración de una variable del tipo **`Collection`**, no hay que declarar ni el número de elementos que va a contener, ni su tipo.

Aunque más adelante hablaremos de los Objetos, para declarar una colección, como objeto que es, hay que dar 2 pasos.

1. Se declara una variable como del tipo **`Collection`**.
2. Se crea una **instancia** del objeto **`Collection`** utilizando la palabra clave **`New`** y la palabra clave **`Set`** que sirve para la Asignación de objetos.
3. Hay una alternativa que es declararla y realizar la instancia en un único paso, que sería: `Public NombreDeLaColección As New Collection`

Para **añadir elementos** a una colección, se utiliza el método **`Add`**.

La forma es `NombreDeLaColección.Add Elemento`

Podemos añadir antes ó después de un elemento dado.

La forma es `NombreDeLaColección.Add Elemento, NumeroAntes`

`NombreDeLaColección.Add Elemento,, NumeroDespués`

Podemos incluso definir una palabra clave para acceder después a un elemento dado de la colección.

```
NombreDeLaColección.Add Elemento, "Clave"
```

Si utilizamos una clave, podemos también colocarlo antes ó después de un elemento dado.

Ejemplo:

Supongamos que tenemos la colección **Productos**.

Para añadir elemento "Llave fija de de 7-8 mm." podemos hacer:

```
Productos.Add "Llave fija de de 7-8 mm."
```

Si ahora queremos añadir "Destornillador Philips de 9 mm." con la clave "DestPhi009"

```
Productos.Add "Destornillador Philips de 9 mm.", "DestPhi009"
```

Vamos ahora a añadir un nuevo producto en la posición Nº 2

```
Productos.Add "Martillo Carpintero 4", "MrtCrp004", 2
```

Vamos ahora a añadir un nuevo producto después de la posición Nº 2

```
Productos.Add "Martillo Carpintero 6", "MrtCrp006", , 2
```

Para saber cuántos elementos hay en una colección tenemos el método **Count**.

```
lngElementos = Productos.Count
```

Podemos obtener un elemento de la colección mediante su índice ó su Clave.

Para eliminar un elemento de una colección se utiliza el método **Remove**, indicando su índice ó su clave.

```
Productos.Remove (3)
Productos.Remove ("DestPhi009")
```

Vamos ahora a probar todo.

Escribimos en un módulo estándar lo siguiente

```
Public Sub PruebaColeccion1()
    Dim Productos As Collection

    Set Productos = New Collection

    Productos.Add "Llave fija de de 7-8 mm."
    Productos.Add "Destornillador Philips de 9 mm.", "DestPhi009"
    Productos.Add "Martillo Carpintero 4", "MrtCrp004", 2
    Productos.Add "Martillo Carpintero 6", "MrtCrp006", , 2

    Debug.Print Productos.Count & " elementos"
    Debug.Print Productos(1)
    Debug.Print Productos(2)
    Debug.Print Productos("MrtCrp006")
    Debug.Print Productos("DestPhi009")

    Productos.Remove (3)
    Productos.Remove ("DestPhi009")

    Debug.Print Productos.Count & " elementos"
End Sub
```

Al ejecutarlo nos mostrará en la ventana **Inmediato**.:

```
4 elementos
Llave fija de de 7-8 mm.
Martillo Carpintero 4
Martillo Carpintero 6
Destornillador Philips de 9 mm.
2 elementos
```

Si utilizamos la opción de añadir un elemento antes ó después de uno dado, deberíamos cerciorarnos de que ese elemento existe.

Si en el ejemplo anterior la línea 6 hubiera sido

```
Productos.Add "Martillo Carpintero 4", "MrtCrp004", 6
```

Nos hubiera dado un error de “Subíndice fuera de intervalo”, ya que en ese momento la colección **Productos** no contenía 6 elementos.

Para averiguar si existe ese nº de elemento usaremos el método **Count** del objeto **Productos**.

Antes de seguir: For Each - - - Next

Como veremos más adelante, en VBA existen unas estructuras de código que permiten realizar bucles ó iteraciones.

Una de ellas es la estructura

```
For Each Elemento in Colección
     (Tras este Each, Elemento pasa a ser uno de los
     elementos de esa colección y podemos usarlo)
Next Elemento
```

Elemento debe ser del tipo **variant** ó un objeto del tipo de los que están en la colección

Por ejemplo, vamos a crear un nuevo procedimiento:

```
Public Sub PruebaColeccion2()
    Dim Clientes As New Collection
    Dim Cliente As Variant

    With Clientes
        .Add "Antonio Urrutia Garastegui", "Urrutia"
        .Add "Ibón Arregui Viana", "Arregui"
        .Add "Pedro Martínez Vergara", "Martínez"
    End With

    For Each Cliente In Clientes
        Debug.Print Cliente
    Next Cliente

End Sub
```

Al ejecutarlo nos muestra:

```
Antonio Urrutia Garastegui
Ibón Arregui Viana
Pedro Martínez Vergara
```

Fijaros que Cliente que es del tipo Variant, automáticamente se convierte en un tipo String, al pasar por el bucle `For Each Cliente In Clientes`
Esto es porque la colección está conteniendo datos del tipo String.

Colecciones de Controles

Las colecciones las podemos encontrar en muchas partes de Access.

Por ejemplo los formularios e informe tienen una serie de controles; pues bien, también cada formulario, o informe, posee una colección que es la colección **Controls**.

Esta colección guarda las referencias de los controles del formulario, o informe.

Vamos a ver cómo podemos leer datos de los Objetos de esta colección.

Para ello vamos a utilizar la estructura

```
For Each Objeto in Me.Controls
Next Objeto
```

Manos a la obra:

Los objetos del formulario (controles) los almacenaremos en la colección **Objetos**.

Vamos a crear un nuevo formulario, y habiendo desactivado el asistente de controles, como se explicaba en la entrega 01, vamos a colocar varios controles. Da igual cuáles sean.

He colocado una Etiqueta, un Cuadro de Texto, un Botón de Opción, un Cuadro Combinado, un Cuadro de Lista y un Botón de Comando.

Los cinco últimos, excepto el botón de comando, llevan a su vez asociada una etiqueta.

El módulo de clase del formulario es el que se abre desde el modo diseño del formulario, al presionar en el menú **Ver** y a continuación la opción **Código**.

También se puede presionar el botón [**Código**]

Ahora, en el módulo de clase del formulario vamos a escribir lo siguiente

```
Private Sub Form_Load()
     Caption = " Controles en " & Me.Name
     MostrarControles
End Sub
```

Y a continuación escribiremos en el procedimiento `MostrarControles`, con lo que quedará así:

```
Option Compare Database
Option Explicit

Private Sub Form_Load()
     Caption = " Controles en el formulario"
```

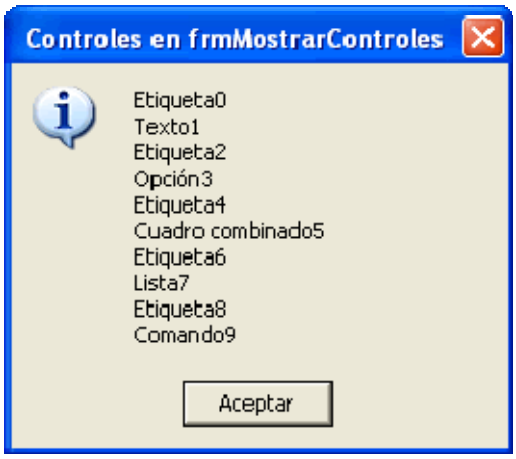
```
        MostrarControles
    End Sub

    Private Sub MostrarControles ()
        Dim ctrl As Control
        Dim strMensaje As String
        For Each ctrl In Me.Controls
           strMensaje = strMensaje & ctrl.Name & vbCrLf
        Next ctrl
        MsgBox strMensaje, _
                vbInformation + vbOKOnly, _
               " Controles en " _
               & Me.Name
    End Sub
```

Al arrancar el formulario, y antes de mostrarse aparecerá este Cuadro de mensaje.

Como inciso, quiero indicar que un Cuadro de Mensaje se muestra mediante la función **MsgBox**.

Permite mostrar mensajes con diversos tipos de iconos, botones y configurar el título de la barra superior.



El evento `Form_Load()` (**al cargar**) del formulario realiza 2 acciones

- Primero pone título al formulario Access.
  En este caso "**Controles en el formulario**". Para ello utiliza la línea
  `Caption = " Controles en el formulario"`
  La propiedad **Caption**, en este caso se refiere al título del propio formulario.
- A continuación llama al procedimiento `MostrarControles` que está definido en el mismo formulario como un procedimiento `private`.

Vamos a fijarnos detenidamente en el procedimiento **MostrarControles**.

En él definimos dos variables, la primera de nombre **ctrl**, de tipo `Control`.

La segunda es la cadena de texto **strMensaje** de tipo `string`.

A continuación utiliza la estructura de código.

```
    For Each ctrl In Me.Controls
```

Como hemos comentado antes, se podría traducir en algo así como:

Toma, uno a uno, todos los objetos **Control** que encuentres en la colección **Controls** del formulario actual y los vas asignando a la variable **ctrl**.

Con esto se consigue que `ctrl` vaya adquiriendo sucesivamente la posibilidad de manejar cada uno de los controles de la colección **Controls**, lo que equivale a poder manejar todos los controles del formulario.

En este código, cuando `ctrl` apunta a un control, puede leer ó modificar algunas de las propiedades del control al que apunta. Por ejemplo puede leer su propiedad **Name** y escribirla en la cadena `strMensaje`. A continuación llega a la siguiente línea que le hace buscar el siguiente control de la colección **Controls**.

Estas dos líneas son:

```
            strMensaje = strMensaje & ctrl.Name & vbCrLf
    Next ctrl
```

Como elemento nuevo puedo mencionar `vbCrLf` .

vbCrLf es una constante definida por VBA. Es una constante de texto que contiene dos caracteres, en concreto el carácter que produce un **Retorno de Carro** (ASCII 13) y el carácter que genera un **Salto de Línea** (ASCII 10).

Forma parte de las llamadas **Constantes Intrínsecas**.

En resumen va asignando uno a uno los controles de **Controls** a la variable **ctrl**, leemos su propiedad **Name** que escribimos en la variable **strMensaje** y vamos repitiendo el proceso y escribiendo al principio de la línea siguiente.

A continuación mostramos el valor de la variable **strMensaje** en un cuadro de mensaje.

Este es un ejemplo que nos muestra cómo podemos ir asignando los elementos de una colección de controles a una variable del tipo control.

Las colecciones **Controls** las crea implícitamente Access en los formularios e informes.

Introducción a las Clases

Puede sorprender a más de uno que en el comienzo de un curso de introducción a VBA empecemos ya a hablar de las clases.

¿Y por qué no?

En el fondo, desde el principio estamos usando clases.

Un formulario es un objeto, pero tiene su módulo de clase asociado.

Cada vez que creamos un objeto, estamos usando su módulo de clase, la mayor parte de las veces sin saberlo.

¿Pero qué es una clase?

En VBA de Access, una clase es algo tan sencillo como un conjunto de código, que contiene además de datos procedimientos para manejarlos, y con el que se pueden crear Objetos.

La clase es el código, y el Objeto creado es el Ejemplar de la clase.

Repasemos el Tipo Persona que habíamos definido en la entrega anterior

```
Public Type Persona
    Nombre As String
    Apellido1 As String
```

```
        Apellido2 As String
        FechaNacimiento As Date
        Telefono As String
    End Type
```

Está bien; pero estaría mejor si por ejemplo, tras introducir la fecha de nacimiento, pudiéramos obtener además la Edad de la persona de forma automática.

Para ello el tipo Persona, debería tener un procedimiento que nos devolviera la edad, en función de la fecha de nacimiento.

Esto no se puede hacer con los tipos **Registro**.

Para poder hacerlo podemos utilizar una clase.

Para crear una clase primero debemos crearnos un módulo de clase.

Para ello, desde la ventana del editor de código hacemos lo siguiente.

Desde la opción de menú **Insertar** seleccionamos la opción **Módulo de clase**.

También podríamos haber presionado la flecha Hacia abajo del botón [**Agregar módulo**] y seleccionar **Módulo de clase**.

Se nos abre el editor en la ventana del nuevo módulo, y aparece éste en la ventana del **Explorador del proyecto**. Esta ventana normalmente aparecerá en la parte izquierda de la pantalla.

Fijaros que aparece un icono con el nombre **Clase1** a su derecha.

Este es un icono especial que llevan los módulos de clase.

En VBA de Access, y también en el de Visual Basic, cada clase debe tener su propio módulo. Lo que equivale a decir que **en un módulo de clase sólo puede haber una clase**.

Vamos a cambiar el nombre de la clase y llamarla **CPersona**.

Para ello podemos, por ejemplo presionar el botón de [**Guardar**].

En esta entrega vamos a utilizar una parte muy pequeña de las posibilidades de las clases, pero es una forma de empezar.

Empezando:

Vamos a introducir este código en el módulo de clase:

```
Option Compare Database
Option Explicit

Public Nombre As String
Public Apellido1 As String
Public Apellido2 As String
Public FechaNacimiento As Date
Public Telefono As String

Public Function Edad() As Long
    ' Que conste que esta función no es exacta
    If FechaNacimiento > 0 Then
        Edad = (Date - FechaNacimiento) / 365.2425
```

```
        End If
    End Function

    Public Function NombreCompleto() As String
        NombreCompleto = Nombre _
                & " " & Apellido1 _
                & " " & Apellido2
    End Function
```

Tampoco nos ha pasado nada por escribir esto.

Prácticamente la única parte que todavía no hemos visto es

```
If FechaNacimiento > 0 Then
        Edad = (Date - FechaNacimiento) / 365.2425
End If
```

De todas formas es fácil de entender:

Si `FechaNacimiento` es mayor que 0, entonces haz que la función devuelva la diferencia entre la fecha de hoy `Date`, y la `FechaNacimiento`, dividida entre `365.25`.

Veremos las instrucciones **`If...Then...Else`** en próximas entregas.

Otra puntualización es la comilla simple

```
' Que conste que esta función no es exacta
```

La comilla simple permite escribir comentarios en el texto, para darnos información, o al que vaya a leer el código con posterioridad.

Los comentarios son ignorados durante la ejecución del código.

Ya tenemos la clase CPersona. ¿Cómo la podremos usar?.

Grabamos todo y nos abrimos un nuevo módulo, esta vez de los estándar.

Esta vez va a haber algo más de código:

```
Option Compare Database
Option Explicit

Public Empleados As Collection

Public Sub PruebaConClase()
    Set Empleados = New Collection
    Dim psnEmpleado As Cpersona
    Dim Empleado As New Cpersona
    Dim i As Long

    'Asignamos valores a la variable psnEmpleado
    Set psnEmpleado = New Cpersona
    With psnEmpleado
```

```
        .Nombre = "Antonio"
        .Apellido1 = "Urrutia"
        .Apellido2 = "Garastegui"
        .FechaNacimiento = #2/24/1965#
        .Telefono = "998111111"
    End With
    ' Añadimos el contenido de la variable a la colección
    Empleados.Add psnEmpleado, "Urrutia"

    'Asignamos valores del 2° empleado a psnEmpleado
    Set psnEmpleado = New Cpersona
    With psnEmpleado
        .Nombre = "Ibón"
        .Apellido1 = "Arregui"
        .Apellido2 = "Viana"
        .FechaNacimiento = #9/14/1985#
        .Telefono = "998222222"
    End With
    ' Añadimos el segundo empleado a la colección
    Empleados.Add psnEmpleado, "Arregui"

    'Asignamos valores de otro nuevo empleado
    Set psnEmpleado = New Cpersona
    With psnEmpleado
        .Nombre = "Pedro"
        .Apellido1 = "Martínez"
        .Apellido2 = "Vergara"
        .FechaNacimiento = #3/11/1979#
        .Telefono = "998333333"
    End With
    ' Añadimos el segundo empleado a la colección
    Empleados.Add psnEmpleado, "Martínez"

    For Each Empleado In Empleados
        With Empleado
            Debug.Print .Nombre
            Debug.Print .Apellido1
            Debug.Print .Apellido2
            Debug.Print .FechaNacimiento
            Debug.Print .Telefono
            Debug.Print .NombreCompleto _
                        & ", " & .Edad & " años"
```

```
        End With
    Next Empleado

    Set Empleado = Empleados(2)
    With Empleado
        Debug.Print .Nombre
        Debug.Print .Apellido1
        Debug.Print .Apellido2
        Debug.Print .FechaNacimiento
        Debug.Print .Telefono
        Debug.Print .NombreCompleto _
                    & ", " & .Edad & " años"
    End With

    Set Empleado = Empleados("Martínez")
    With Empleado
        Debug.Print .Nombre
        Debug.Print .Apellido1
        Debug.Print .Apellido2
        Debug.Print .FechaNacimiento
        Debug.Print .Telefono
        Debug.Print .NombreCompleto _
                    & ", " & .Edad & " años"
    End With

    VaciaColeccion Empleados

End Sub

Public Sub VaciaColeccion(ByRef Coleccion As Collection)
    Dim i As Long
    For i = Coleccion.Count To 1 Step -1
        Coleccion.Remove (i)
    Next i
End Sub
```

Otro tema nuevo que aparece aquí es el bucle **`For - - - Next`** en el procedimiento **`VaciaColección`**.

Os adelanto que en este caso i va tomando valores que van del número de elementos de la colección, hasta 1, disminuyendo de 1 en 1.

Analizad este código y en la próxima entrega lo comentaré más a fondo.

Si ejecutamos el procedimiento **PruebaConClase** nos imprime, en la ventana **Inmediato**:

```
Antonio
Urrutia
Garastegui
24/02/1965
998111111
Antonio Urrutia Garastegui, 40 años
Ibón
Arregui
Viana
14/09/1985
998222222
Ibón Arregui Viana, 19 años
Pedro
Martínez
Vergara
11/03/1979
998333333
Pedro Martínez Vergara, 26 años
Ibón
Arregui
Viana
14/09/1985
998222222
Ibón Arregui Viana, 19 años
Pedro
Martínez
Vergara
11/03/1979
998333333
Pedro Martínez Vergara, 26 años
```