Colecciones

Las colecccciones son agrupaciones de objetos. Se utilizan para podes manipular un conjunto de instancias agrupadas en una entidad.
Existen varios tipos de colecciones en .NET. A continuación veremos ejemplos de ellas.

Arrays multidimensionales (vectores, matrices, cubos, etc.)

Es uno de los tipos de colecciones más simples.
Los arrays se instancian declarando su tipo de dato, dimensiones y cantidad de items.
Una vez declarado un array, la cantidad de items no puede ser modificada.

C#

```
int[] Vector = new int[x];

int[,] Matriz = new int[x,y];

int[,,] Cubo = new int[x, y,z];
```

VB.NET

```
Dim Vector As Integer() = New Integer(x) {}

Dim Matriz As Integer(,) = New Integer(x, y) {}

Dim Cubo As Integer(,,) = New Integer(x, y, z) {}
```

Los arrays pueden ser de cualquier tipo de objeto.

C#

```
Persona[] Personas = new Persona[3];
```

VB.NET

```
Dim Personas As Persona() = New Persona(2) {}
```

Clase ArrayList

El ArrayList es una colección del tipo lista.
A diferencia de los arrays multidimensionales y otra colecciones, el ArrayList no puede definirse de un tipo de datos especifo, o sea, soporta elementos de cualquier tipo de objeto.
Tampoco es necesario declarar la cantidad de elementos, ya que se redimensionan automáticamente.
Sin embargo, si se declaran con dimensión el rendimiento es mejor ya que no debe redimensionarse dinámicamente cada vez que se agrega un item.

Declaración

C#

```
ArrayList ListaA = new ArrayList();
ArrayList ListaB = new ArrayList(10);
```

VB.NET

```
Dim ListaA As New ArrayList
Dim ListaB As New ArrayList(10)
```

Recorrer sus items

C#

```
foreach (object item in ListaA)
{
    //Se muestra el tipo de objeto
    Console.Write("Tipo de objeto: {0}. ",item.GetType().FullName);

    //Se decide como mostrar el item segun sea el tipo de objeto
    if (item.GetType() == typeof(System.Int32))
    {
        Console.WriteLine("Valor: {0}", (int) item);
    }
    if (item.GetType() == typeof(string))
    {
        Console.WriteLine("Valor: {0}", item.ToString());
    }
    if (item.GetType() == typeof(Objetos.Persona))
    {
        Persona objPersona = (Persona) item;
        Console.WriteLine("Valor: {0}", objPersona.Nombre + " " + objPersona.Apellido );
    }
}
```

.NET

```
For Each item As Object In ListaA
    'Se muestra el tipo de objeto
    Console.Write("Tipo de objeto: {0}. ", item.GetType.FullName)

    'Se decide como mostrar el item segun sea el tipo de objeto
    If item.GetType Is GetType(System.Int32) Then
        Console.WriteLine("Valor: {0}", CInt(item))
    End If
    If item.GetType Is GetType(String) Then
        Console.WriteLine("Valor: {0}", item.ToString)
    End If
    If item.GetType Is GetType(Objetos.Persona) Then
        Dim objPersona As Persona = CType(item, Persona)
        Console.WriteLine("Valor: {0}", objPersona.Nombre & " " & objPersona.Apellido)
    End If
Next
```

Agregar items

C#

```
ListaA.Add(1);
ListaA.Add("A");
ListaA.Add(obj);
```

VB.NET

```
ListaA.Add(1)
ListaA.Add("A")
ListaA.Add(obj)
```

Insertar items

C#

```
ListaA.Insert(1, 2);
ListaA.Insert(1, "B");
ListaA.Insert(1, obj);
```

VB.NET

```
ListaA.Insert(1, 2)
ListaA.Insert(1, "B")
ListaA.Insert(1, obj)
```

Eliminar items

C#

```
ListaA.RemoveAt(0);
ListaA.Remove("A");
```

VB.NET

```
ListaA.RemoveAt(0)
ListaA.Remove("A")
```

Saber si un item está contenido en la lista

C#

```
ListaA.Contains("A")
```

VB.NET

```
ListaA.Contains("A")
```

Ordenar en forma ascendente

Si el tipo de datos del ArrayList no es un tipo primitivo, hay que implementar una interfaz para indicar al método cómo ordenar la colección.

C#

```
ListaC.Sort();
```

.NET

```
ListaC.Sort()
```

Buscar y obtener el índice de un elemento en la lista

Para utilizar el método BinarySearch() la lista debe estar ordenada.

C#

```
ListaC.BinarySearch("Ana")
```

VB.NET

```
ListaC.BinarySearch("Ana")
```

List<t>/List(Of t)

Esta colección muy similar ArrayList. La mayor diferencia es que las listas deben ser declaradas de un tipo específico.
Su uso es preferible al ArrayList, ya que es posible detectar en tiempo de compilación que no se agreguen por error elementos de un tipo diferente al que ha sido declarado.
Son además muy eficientes cuando se accede a ellas secuencialmente. Recorrer sus items
Para acceder a cada elemento se utiliza el índice.

Declaración

C#

```
List<string> ListaA = new List<string>();
List<string> ListaB = new List<string>(10);
```

VB.NET

```
Dim ListaA As New List(Of String)
Dim ListaB As New List(Of String)(10)
```

Recorrer sus items

Cómo se conoce el tipo de dato no es necesario determinar el tipo de objeto cómo en el caso de un ArrayList:

C#

```
foreach (Persona item in ListaC)
{
    Console.WriteLine("Apellido: {0}, {1}",item.Apellido, item.Nombre);
}
```

.NET

```
For Each item As Persona In ListaC
    Console.WriteLine("Apellido: {0}, {1}", item.Apellido, item.Nombre)
Next
```

El resto de la operaciones (agregar, insertar, etc.) se realiza en forma análoga a la clase ArrayList.

SortedList<k, v>/

La principal diferencia respecto a la lista es que se accede a sus elementos a través de una clave. Asimismo, la lista se ordena independientemente del orden en el que se agregan los items.

Declaración

C#

```
SortedList<string, int> ListaOrdenada = new SortedList<string, int>();
```

VB.NET

```
Dim ListaOrdenada As New SortedList(Of String, Integer)
```

Agregar items

C#

```
ListaOrdenada.Add("C", 300);
ListaOrdenada.Add("A", 100);
ListaOrdenada.Add("B", 200);
```

VB.NET

```
ListaOrdenada.Add("C", 300)
ListaOrdenada.Add("A", 100)
ListaOrdenada.Add("B", 200)
```

Acceder a un valor través de su clave

C#

```
Console.WriteLine(ListaOrdenada["A"]);
Console.WriteLine(ListaOrdenada["B"]);
Console.WriteLine(ListaOrdenada["C"]);
```

VB.NET

```
Console.WriteLine(ListaOrdenada("A"))
Console.WriteLine(ListaOrdenada("B"))
Console.WriteLine(ListaOrdenada("C"))
```

Recorrer sus items

C#

```
foreach (KeyValuePair<string, int> item in ListaOrdenada)
{
    Console.WriteLine("Clave: {0}. Valor: {1}",item.Key, item.Value);
}
```

VB.NET

```
For Each item As KeyValuePair(Of String, Integer) In ListaOrdenada
    Console.WriteLine("Clave: {0}. Valor: {1}", item.Key, item.Value)
Next
```

Queue<t>

Esta clase implementa una cola.
El primer elemento en entrar es el primero en salir (FIFO).
A medida que se accede a sus elementos estos van siendo eliminados.

Declaración

C#

```
Queue<Persona> Cola = new Queue<Persona>();
```

VB.NET

```
Dim Cola As New Queue(Of Persona)
```

Agregar items

C#

```
Cola.Enqueue(new Persona("Ana", "González"));
Cola.Enqueue(new Persona("Pedro", "Benítez"));
Cola.Enqueue(new Persona("Isabel", "Pérez"));
```

VB.NET

```
Cola.Enqueue(New Persona("Ana", "González"))
Cola.Enqueue(New Persona("Pedro", "Benítez"))
Cola.Enqueue(New Persona("Isabel", "Pérez"))
```

Recorrer y vaciar sus items

C#

```
while (Cola.Count > 0)
{
    Persona item = Cola.Dequeue();
    Console.WriteLine("{0}, {1}", item.Apellido, item.Nombre);
}
```

VB.NET

```
While Cola.Count > 0
    Dim item As Persona = Cola.Dequeue
    Console.WriteLine("{0}, {1}", item.Apellido, item.Nombre)
End While
```

Stack<t>

Esta clase implementa una pila.
El último elemento en entrar es el primero en salir (LIFO).
A medida que se accede a sus elementos estos van siendo eliminados.

Declaración

C#

```
Stack<Persona> Pila = new Stack<Persona>();
```

VB.NET

```
Dim Pila As New Stack(Of Persona)
```

Agregar items

C#

```
Pila.Push(new Persona("Ana", "González"));
Pila.Push(new Persona("Pedro", "Benítez"));
Pila.Push(new Persona("Isabel", "Pérez"));
```

VB.NET

```
Pila.Push(New Persona("Ana", "González"))
Pila.Push(New Persona("Pedro", "Benítez"))
Pila.Push(New Persona("Isabel", "Pérez"))
```

Recorrer y vaciar sus items

C#

```
while (Pila.Count > 0)
{
    Persona item = Pila.Pop();
    Console.WriteLine("{0}, {1}", item.Apellido, item.Nombre);
}
```

VB.NET

```
While Pila.Count > 0
    Dim item As Persona = Pila.Pop
    Console.WriteLine("{0}, {1}", item.Apellido, item.Nombre)
End While
```
