LAS COLECCIONES

Las colecciones son **estructuras de datos** que nos permiten guardar en su interior cualquier tipo de información. Existen diferentes tipos de colecciones y la forma como se guarda, se accede y se elimina la información en cada una de ellas es distinta. En los arreglos nosotros teníamos que indicar la cantidad de elementos que el arreglo debía tener. En las colecciones esto no es necesario, ya que es posible agregar elementos **dinámicamente**. Esto quiere decir que cuando el programa se está ejecutando podemos adicionar o borrar sus elementos.
En otros lenguajes de programación cuando se desea tener este tipo de estructuras generalmente es necesario programarlas antes de poder usarlas. Sin embargo, C# nos provee de las colecciones más importantes y podemos utilizarlas directamente sin tener que hacer ningún tipo de desarrollo previo. Las coleccione son: **ArrayList, Hashtable, Queue, y Stack**.

EL ARRAYLIST

La primera colección que aprenderemos se conoce como **ArrayList**, que guarda la información como si fuera una lista. Y sobre esta lista es posible realizar diferentes actividades con los elementos almacenados. Entendemos al **ArrayList** como un arreglo que puede cambiar su tamaño según lo necesitemos. Puede guardar cualquier tipo de dato, por lo que lo podemos usar para enteros, cadenas, flotantes o incluso para tipos definidos por nosotros mismos. **ArrayList** es una clase en C#, por lo que va a tener métodos o funciones que nos permitirán trabajar con los datos. El **ArrayList** tiene una propiedad que llamamos **capacidad**, que indica el tamaño que ocupa la lista. También tenemos el **conteo**, el cual nos dice cuántos elementos está guardando en su interior.

DECLARACIÓN DE UN ARRAYLIST

```
ArrayList datos = new ArrayList();
```

EJEMPLO

```
ArrayList datos = new ArrayList(32);
```

ADICIÓN DE INFORMACIÓN

Método **Add()**.

Siempre que se adiciona un elemento al **ArrayList**, este nuevo elemento se agrega al final. Si incorporamos otro elemento, se colocará después del anterior. La adición siempre se lleva a cabo después del último elemento que se encuentre en el **ArrayList**.

El método **Add()** va a necesitar de un único parámetro y este es el dato que queremos guardar. Por ejemplo, para guardar un dato de tipo entero podemos hacer lo siguiente

EJEMPLO

```
datos.Add(7);
datos.Add(n);
palabras.Add("Hola");
```

EL USO DE FOREACH

En la programación orientada a objetos existe un patrón conocido como iterador o iterator, según su sintaxis en el idioma inglés. La principal característica del iterador es permitirnos realizar un recorrido por todos los elementos que existen en una estructura de datos. El recorrido lo hace de forma secuencial, uno por uno. Esto resulta muy útil cuando sabemos que debemos recorrer todos los elementos de la estructura, como un ArrayList, y hacer algo con ellos.
En C# encontramos un iterador en el foreach. Con él es posible que recorramos los elementos, luego ejecutamos alguna acción con ellos y finalizamos cuando la colección ya no tiene más elementos que entregarnos.

```
foreach (tipo identificador in expresión)
    sentencia
```

EJEMPLO

```
foreach(int valor in costo)
{
    Console.WriteLine("El valor es {0}", valor);
}
```

CÓMO ACCEDER A LA INFORMACIÓN DE UN ARRAYLIST

La colección **ArrayList** nos permite acceder a sus elementos por medio de un índice, algunas colecciones no lo permiten, pero **ArrayList** sí. Esto es bueno, pues podemos trabajarla de forma similar a un arreglo, pero con la flexibilidad de ser dinámica.

EJEMPLO

Si tenemos un **ArrayList** llamado datos y deseamos imprimir el elemento **2**, lo podemos hacer de la siguiente manera:

```
Console.WriteLine("El dato es {0}",datos[2]);
```

De igual forma podemos utilizar el valor del elemento en una expresión:

```
impuesto = datos[2] * 0.15f;
```

Y podemos también asignar un valor determinado:

```
datos[2] = 5;
```

CÓMO OBTENER LA CANTIDAD DE ELEMENTOS EN UN ARRAYLIST

Método **Count()**.

En muchas ocasiones es útil saber cuántos elementos tenemos en el **ArrayList**. A diferencia del arreglo tradicional, éste puede cambiar su tamaño durante la ejecución de la aplicación y conocer el tamaño nos evita problemas con los valores de los índices al acceder el arreglo.

Afortunadamente es muy sencillo hacerlo, simplemente tenemos que leer el valor de la propiedad **count** del **ArrayList**. Esta propiedad es un valor entero.

```
elementos = datos.Count;
```

INSERTAR ELEMENTOS

Método **Insert()**

La inserción permite colocar un elemento nuevo en cualquier posición válida del arreglo. Para lograr esto usamos el método **Insert()**. Este método necesita de dos parámetros, el primer parámetro es el índice donde deseamos insertar el elemento y el segundo parámetro es el elemento a insertar.

EJEMPLO

Por ejemplo, si deseamos insertar el valor de **5** en el índice **2**, hacemos lo siguiente:

```
datos.Insert(2, 5);
```

PARA ELIMINAR UN ELEMENTO

Método **RemoveAt()**

Es posible eliminar cualquier elemento del **ArrayList** y hacerlo de forma muy sencilla. Lo único que necesitamos es conocer el índice del elemento a eliminar. El índice es un valor entero y debe ser válido.

Este método solamente necesita de un parámetro, que es el índice del objeto que deseamos eliminar. Por ejemplo, si queremos eliminar el elemento que se encuentra en el índice **7**, podemos hacer lo siguiente:

```
datos.RemoveAt(7);
```

PARA ENCONTRAR UN ELEMENTO

Método **IndexOf()**.

Con los **ArrayList** es posible saber si un elemento en particular se encuentra adentro de él. Para lograr esto hacemos uso del método **IndexOf()**. Este método requiere de un solo parámetro que es el objeto a buscar adentro del **ArrayList**. El método nos regresa un valor entero.

Este valor es el índice donde se encuentra la primera ocurrencia del elemento, esto es debido a que podemos tener el elemento guardado en diferentes posiciones. Si el elemento no se encuentra en el **ArrayList**, entonces simplemente recibimos el valor de **-1**.

Si lo que deseamos es buscar el índice donde se encuentra el elemento **7** en nuestro **ArrayList**, hacemos lo siguiente:

```
índice = datos.IndexOf(7);
```

EL STACK

El **Stack** o **pila**, nos permite guardar elementos y cambiar su tamaño de forma dinámica, sin embargo, trabaja en forma diferente al arreglo y al **ArrayList**.

El **Stack** es una estructura de tipo **LIFO**. LIFO es el acrónimo en inglés para *Last- in-first-out*, es decir, el primero que entra, el último que sale.

El efecto de colocar nuevos elementos en la parte superior del **Stack** se conoce como **Push**. Cuando tomamos un elemento de la parte superior del **Stack** se conoce como **Pop**.

Debido a este comportamiento con **Push** y **Pop** podemos entender cómo el primer objeto que se coloca adentro del Stack es el último en poder salir.

COMO CREAR EL STACK

Como cualquier otra colección el **Stack** necesita ser instanciado para poderlo utilizar. C# nos provee de la clase **Stack** y adentro de esta clase encontramos todos los métodos necesarios para trabajar con él. La instanciación simplemente será la creación de un objeto de esta clase. Si deseamos crear un **Stack** hacemos lo siguiente:

```
Stack miPila = new Stack();
```

Hemos creado un **Stack** llamado **miPila**. Nosotros podemos usar cualquier nombre que sea válido. Una vez instanciado podemos empezar a colocar información en él.

CÓMO INTRODUCIR INFORMACIÓN AL STACK

Método **Push()**

Para introducir información al **Stack** usamos el método **Push()**. Este método coloca el nuevo elemento en la parte superior del **Stack**. El método necesita únicamente de un parámetro que es el elemento que deseamos insertar. Podemos utilizar el método **Push()** cuantas veces sea necesario para colocar la información.

```
miPila.Push(7);
miPila.Push(11);
miPila.Push(8);
```

CÓMO OBTENER INFORMACIÓN DEL STACK

Método **Pop()**

Si lo que necesitamos es obtener la información que está contenida en el **Stack** lo podemos hacer al tomar el elemento que se encuentra en la parte superior del **Stack**. Para lograr esto hacemos uso del método **Pop()**. Este método no necesita ningún parámetro y regresa el elemento correspondiente.

Por ejemplo, si deseamos tomar el elemento de nuestro **Stack**:

```
int valor = 0;
...
valor = (int)miPila.Pop();
```

USO DE FOREACH PARA RECORRER EL STACK

Si necesitamos recorrer el **Stack**, es posible hacerlo por medio del uso de **foreach**. El uso de esta alternativa es similar al del **ArrayList**.

```
foreach( int n in miPila)
Console.WriteLine("{0}",n);
```

Podemos usar el **foreach**, generalmente recorreremos el **Stack** por medio de **Pop()**.

PARA OBTENER LA CANTIDAD DE ELEMENTOS DEL STACK

Propiedad **Count()**

Es posible conocer la cantidad de elementos que tiene el **Stack** y hacerlo es muy sencillo. Debemos leer la propiedad **Count** del **Stack**. Esta propiedad nos regresa un valor entero con la cantidad de elementos. El uso es equivalente al del **ArrayList**.

```
cantidad = miPila.Count;
```

PARA LIMPIAR LOS CONTENIDOS DEL STACK

Método **Clear()**

Si deseamos eliminar todos los elementos del **Stack** de forma rápida lo podemos hacer al usar el método **Clear()**. Este método no necesita de ningún parámetro y solamente debe ser usado cuando sepamos que debemos borrar los elementos del **Stack**.

```
miPila.Clear();
```

PARA SABER SI EL STACK TIENE UN ELEMENTO

Método **Contains()**

Si deseamos saber si adentro del **Stack** se encuentra un elemento en particular, podemos hacer uso del método **Contains()**. Este método necesita de un parámetro que es el objeto a encontrar adentro del Stack. El método regresa un valor de tipo **bool**. Si el objeto se encuentra el valor es **true**, pero si no se encuentra es **false**.

```
bool enStack = false;
...
enStack = miPila.Contains(7);
```