LAS COLECCIONES

LAS QUEUE

Las Queue, algunas veces también denominada colas son estructuras de tipo **FIFO**. Se caracterizan por ser una secuencia de elementos en la que la operación de inserción **push** se realiza por un extremo y la operación de extracción **pop** por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir.
Esto nos indica que cuando insertamos elementos en el Queue, éstos se colocan atrás del último elemento insertado o en la parte posterior de éste. Al momento de leer elementos, se toma el que se encuentre en la parte superior del Queue.

El proceso por medio del cual nosotros insertamos un elemento nuevo al Queue se conoce como **Enqueue** y no debemos olvidar que lo hace al final. El acto de extraer un elemento del Queue se llama **Dequeue** y siempre toma el elemento que se encuentra en la parte superior.

DECLARACIÓN DEL QUEUE

```
Queue fila = new Queue();
```

EJEMPLO

```
Queue fila = new Queue(15);
```

ADICIÓN DE INFORMACIÓN

Método **Enqueue()**

Para poder hacer esto debemos utilizar el método **Enqueue()**, el cual pertenece a la clase Queue de C#. Este método es muy sencillo, ya que solamente requiere de un parámetro. En el parámetro colocamos el elemento que deseamos añadir, este método no regresa ningún valor.

EJEMPLO

```
fila.Enqueue(7);
datos.Enqueue(n);
palabras.Enqueue("Hola");
```

EXTRACCIÓN DE INFORMACIÓN

Método **Dequeue()**

Al igual que podemos colocar un elemento, también es posible extraerlo. Sin embargo, la extracción se lleva a cabo de acuerdo a las reglas del **Queue**. Cuando extraemos un elemento, el elemento que recibimos es el que se encuentra en la parte superior o al inicio del **Queue**. Debemos recordar que no es posible la extracción de los elementos que se encuentren en otras posiciones.
Para llevar a cabo la extracción tenemos que usar un método de la clase **Queue** que se llama **Dequeue()**. El método no necesita de ningún parámetro y regresa el elemento correspondiente. Es importante tener una variable que reciba al elemento o una expresión que haga uso de él.

EJEMPLO

```
int valor = 0;
…
…
valor = fila.Dequeue();
```

OBSERVAR UN ELEMENTO EN EL QUEUE

Método **Peek()**

Cuando hacemos uso del método **Dequeue()**, el elemento es extraído y deja de encontrarse adentro del **Queue** después de esa operación. Sin embargo, podemos observar el elemento. En este caso recibimos el valor del elemento, pero éste no es eliminado del **Queue** después de ser leído.
Para llevar a cabo la observación hacemos uso del método **Peek()**. Este método no necesita de ningún parámetro y regresa el elemento observado. Al igual que con **Dequeue()** debe de existir una variable o una expresión que reciba este valor.

EJEMPLO

```
int valor =    0;
…
…
valor = fila.Peek();
```

PARA SABER SI EL QUEUE TIENE DETERMINADO ELEMENTO

Método **Contains()**

Algunas veces podemos necesitar saber si en el interior del **Queue** se guarda un elemento en particular. Esto es posible hacerlo gracias a un método conocido como **Contains()**. Para usar este método, necesitamos pasar como parámetro el elemento que queremos determinar. El método regresará un valor de tipo **bool**. Si el elemento existe adentro del **Queue**, el valor regresado será **true**. En el caso de que el elemento no exista en el interior del **Queue** el valor regresado será **false**. Necesitamos tener una variable o una expresión que reciba el valor del método **Contains()**.

EJEMPLO

```
if(miFila.Contains(7) == true)
    Console.WriteLine("El elemento si existe");
```

PARA BORRAR LOS CONTENIDOS DEL QUEUE

Método **Clear()**

Si necesitamos eliminar todos los contenidos del **Queue**, es sencillo de hacer por medio del método **Clear()**. Este método ya es conocido de colecciones previamente estudiadas. Su utilización es muy sencilla ya que no necesita de ningún parámetro. Solamente debemos tener

cuidado, ya que si lo utilizamos en un momento inadecuado, perderemos información que todavía puede ser útil.

EJEMPLO

```
miFila.Clear();
```

PARA CONOCER LA CANTIDAD DE ELEMENTOS QUE TIENE EL QUEUE

Método **Count()**

Es conveniente saber la cantidad de elementos que contiene el **Queue**, especialmente antes de llevar a cabo algunas operaciones sobre éste. Cuando deseamos saber la cantidad de elementos existentes, debemos utilizar la propiedad de **Count**. Esta propiedad corresponde a un valor de tipo entero.

EJEMPLO

```
int cantidad = 0;
…
…
cantidad = miFila.Count;
```

PARA RECORRER LOS ELEMENTOS DEL QUEUE

Si necesitamos recorrer los elementos del **Queue**, por ejemplo, para mostrarlos o imprimirlos lo que nos conviene usar es un ciclo **foreach**. Su utilización es muy sencilla y similar a lo que hemos usado con otras colecciones.

EJEMPLO

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

LAS COLECCIONES

EL HASHTABLE

El **Hashtable** es una colección indexada. Es decir que vamos a tener un índice y un valor referenciado a ese índice. Sin embargo, la indexación no se lleva a cabo como en el arreglo o el **ArrayList**. El lugar adentro del **Hashtable** donde se coloca el elemento va a depender de un valor conocido como **key** o llave. El valor contenido en **key** es usado para calcular la posición del elemento en el **Hashtable**. El elemento que vamos a colocar se conoce como **value**. En el **Hashtable** siempre utilizaremos una pareja de **key** y **value** para trabajar con él.

DECLARACIÓN DEL HASHTABLE

La declaración del **Hashtable** es muy sencilla y similar a la declaración de las otras colecciones. Para hacerlo, simplemente creamos una instancia de la clase **Hashtable** y después podemos hacer uso de las operaciones de la colección por medio de los métodos que nos provee.

EJEMPLO

```
Hashtable   miTabla = new Hashtable();
```

ADICIÓN DE ELEMENTOS AL HASHTABLE

Método **Add()**

Este método a diferencia de los de las otras colecciones, va a necesitar de dos parámetros. En el primer parámetro colocamos el **key** que será usado para indexar al elemento. Este **key** puede ser de cualquier tipo, pero generalmente se utilizan cadenas. El segundo parámetro será el valor o elemento a insertar, también puede ser de cualquier tipo. Hay que recordar que cuando hagamos uso de esta colección siempre trabajamos la información en parejas **key-value**.

EJEMPLO

```
miTabla.Add("Pan",   5.77);
miTabla.Add("Soda",   10.85);
miTabla.Add("Atun",   15.50);
```

RECORRIENDO EL HASHTABLE

Ciclo **foreach**

Para poder recorrer el **Hashtable**, haremos uso del ciclo **foreach**. Si queremos obtener la pareja **key-value**, nos apoyaremos en una clase conocida como **DictionaryEntry**. El diccionario también guarda parejas de datos.

Por ejemplo, si deseamos proseguir con la impresión de los contenidos del **Hashtable**, podemos colocar el código que se muestra a continuación:

```
foreach(DictionaryEntry  datos  in  miTabla)
    Console.WriteLine("Key  –  {0}, Value  –{1}",  datos.Key, datos.Value);
```

Si lo deseamos podemos extraer solamente los valores y colocar una copia de ellos en una colección. Esto nos permitiría trabajar con los valores de una forma más pa- recida a lo que hemos aprendido anteriormente.

```
ICollection valores = miTabla.Values;
…
…
foreach(double valor in valores) Console.WriteLine("El valor es   {0}",valor);
```

ICollection es una interfase usada para implementar las colecciones, de tal forma que valores puede actuar como cualquier colección válida que tengamos, en este caso la colección que representa los valores extraídos del **Hashtable.**

PARA OBTENER UN ELEMENTO DEL HASHTABLE

Si deseamos leer un elemento en particular del **Hashtable**, es posible que lo hagamos por medio de su propiedad **Item**. Para utilizarla, simplemente tenemos que colocar como índice el **key** que corresponde al valor que queremos leer en el momento. Debemos tener en cuenta que es posible que necesitemos hacer un **type cast** para dejar el valor correspondiente en el tipo necesario.

```
float  valor;
…


…
valor =  (float)miTabla.Item["Pan];
```

PARA BORRAR LOS CONTENIDOS DEL HASHHTABLE

Método **Clear()**

Todos los elementos guardados adentro del **Hashtable** pueden ser borrados al utilizar el método **Clear()**. Este método no necesita de ningún parámetro.

```
miTabla.Clear();
```

PARA ENCONTRAR SI YA EXISTE UN KEY

Método **Contains()**

Como no podemos repetir el mismo **key** para dos elementos, es necesario poder saber si ya existe determinado **key** en el **Hashtable**. Poder hacerlo es fácil, pues C# nos provee del método **Contains()**. Este método necesita de un único parámetro que es el **key** a buscar y regresa un valor de tipo **bool**. Si el valor regresado es **true** significa que el **key** ya existe, y si es **false** que no se encuentra en el **Hashtable.**

```
bool   exsite;
…
…
existe =  miTabla.Contains("Pan");
```

PARA ENCONTRAR SI YA EXISTE UN VALUE

Método **ContainsValue()**

Igualmente podemos buscar adentro del **Hashtable** por un **value** en particular. En este caso, usaremos el método **ContainsValue()**. Como parámetro colocaremos el valor a buscar y el método regresará un **bool**. Si el valor regresado es **true**, el **value** existe en el **Hashtable** y el valor de **false** nos indica que no se encuentra.

```
bool   existe;
…
…
existe =  miTabla.ContainsValue(17.50);
```

PARA CONOCER LA CANTIDAD DE PAREJAS EN EL HASHTABLE

Propiedad **Count**

Si deseamos saber cuántas parejas **key-value** existen dentro de nuestro Hashtable, podemos usar la propiedad de **Count**. No hay que olvidar que el valor regresado es un entero que dice el número de parejas.

```
int cantidad;
…
…
cantidad = miTabla.Count;
```

PARA ELIMINAR UN ELEMENTO DEL HASHTABLE

Método **Remove()**

En el **Hashtable** no solamente podemos adicionar elementos, también podemos eliminarlos. Al eliminarlos removemos la pareja **key-value** del **Hashtable**. Para poder llevar a cabo la eliminación, necesitamos conocer el **key** del valor a eliminar y utilizar el método **Remove()**. Este método necesita de un solo parámetro que es el **key** del elemento a borrar.

```
miTabla.Remove(“Pan”);
```