**HASHTABLE**

**Una tabla hash o Hash Table**, es una colección que permite almacenar pares de objetos, donde el **primero es conocido como llave (key) y el segundo es conocido como valor (value).**

De esta manera, para agregar “un” elemento a la colección, siempre se debe pasar como argumento estos dos objetos.

[image](https://developmania.files.wordpress.com/2008/09/image.png)

Una de las *ventajas que presenta ésta colección* frente a las colecciones mencionadas anteriormente (Arraylist , [Stack y Queue](https://developmania.wordpress.com/2008/08/29/colecciones-de-datos-en-cnet-parte-iii-stack-queue/" \t "_blank)), es el poco tiempo que tarda en realizar la búsqueda de un elemento determinado, debido a que utiliza el hash de las llaves (keys) para ordenar los elementos, permitiendo así, tener una ubicación única para cada uno.

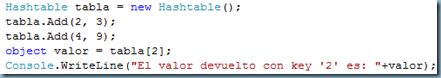
Un Hash *consiste en pocas palabras*, en un código único generado por una función Hash, la cual genera una salida única para entradas diferentes y además, produce la misma salida para entradas iguales.

Por estas características de una función Hash, no pueden haber llaves iguales en la tabla Hash ya que esto produciría el mismo Hash y no sería posible diferenciar unívocamente los elementos de la colección.

El comportamiento de la HashTable se puede comparar a una tabla de una base de datos relacional, donde cada tabla generalmente (en este caso obligatoriamente) tiene una llave primaria por medio de la cual se diferencia cada registro de los demás y que además su valor debe ser único en toda la tabla.

De la misma manera se comporta la HashTable ya que los elementos son ordenados según su Hash.

Cuando sea necesario obtener uno de los elementos contenidos en la HashTable, se debe utilizar la llave o key con la que fue almacenada, como se muestra a continuación:

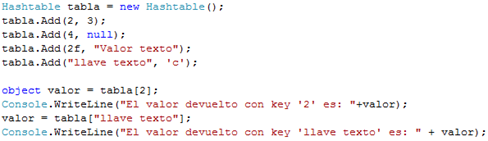
[](https://developmania.files.wordpress.com/2008/09/image1.png)

[image](https://developmania.files.wordpress.com/2008/09/image2.png)

Siempre que se referencia la HashTable utilizando una Key, será retornado el valor correspondiente a dicha clave en un objeto de tipo Object, por lo que es importante convertir el dato obtenido al tipo de dato adecuado para evitar generar excepciones.

Lo anterior, permite tener en la colección pares de llaves y valor que pertenezcan a tipos de datos diferentes, es decir, nada nos obliga a que todos los elementos del HashTable sean del mismo tipo. Incluso las llaves (keys) y los valores (values), pueden ser de tipos de datos diferentes entre sí.

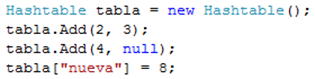
Sin embargo, no es muy recomendable utilizar la HashTable de esta manera, ya que eso implicaría conocer exactamente qué elementos están incluidos en la HashTable e implicaría tener mucho más cuidado a la hora de extraer los datos, lo cual impediría que la aplicación sea flexible y escalable.

[](https://developmania.files.wordpress.com/2008/09/image3.png)

[image](https://developmania.files.wordpress.com/2008/09/image4.png)

En el ejemplo anterior se ve como se puede ingresar llaves y valores de diferentes tipos de datos. Además se puede ver como es posible ingresar un valor de tipo null, lo cual no es posible hacerlo en una de las llaves ya que se generaría la excepción ArgumentNullException.

Utilizando la sintaxis “[key]” (por ejemplo: tabla[“llave texto”]) sobre una tabla Hash, es posible actualizar el valor contenido o en caso de que la llave utilizada no exista en la tabla, se inserta este nuevo elemento.

[](https://developmania.files.wordpress.com/2008/09/image5.png)

La clase HashTable pertenece al espacio de nombres System.Collections e implementa las colecciones IEnumerable, ICollection, entre otras.

Debido a esto, es posible utilizar la instrucción foreach para leer datos de una HashTable.

**Sin embargo, en cada iteracion de esta instrucción, se debe operar con un objeto de tipo DictionaryEntry que contendrá un par llave/valor**.

[image](https://developmania.files.wordpress.com/2008/09/image6.png)

Algunos de los **métodos más utilizados de la clase HashTable** son:

**Add**: permite agregar un nuevo par llave/valor a la colección.

**Remove**: permite quitar un par llave/valor de la colección

**ContainsKey:** permite saber si la colección contiene un par cuya clave sea la que se le pasa como parámetro.

**ContainsValue**: permite saber si la colección contiene un par cuyo valor sea el que se le pasa como parámetro.