

INTERFACE IDICTIONARY

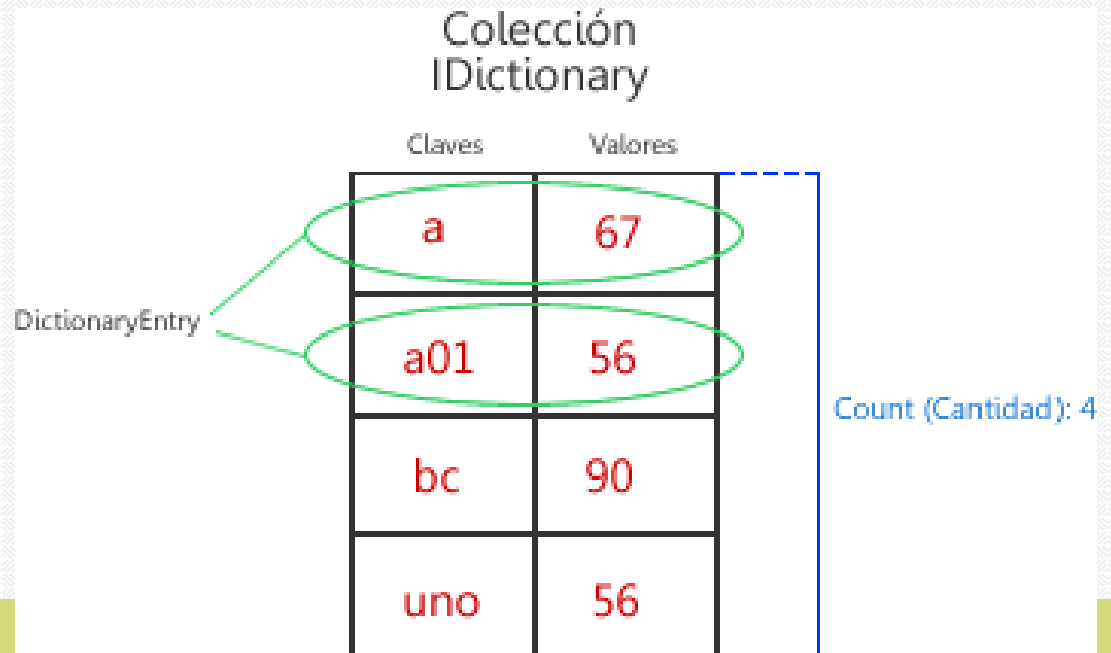
- Representa una colección de pares de clave y valor.
- Cada elemento es un par de clave y valor almacenado en un objeto DictionaryEntry.
 - La clave debe ser única.
 - El valor puede ser nulo y estar duplicado.
- Ejemplos:
 - Clase SortedList
 - Clase Hashtable

Colección IDictionary

Claves	Valores
a	67
a01	56
bc	90
uno	56

DictionaryEntry

Count (Cantidad): 4



DICTIONARYENTRY



- Objeto que define un par de clave y valor de diccionario que se puede establecer o recuperar.
- Propiedades:
 - Key: Obtiene o establece la **clave** del par de clave y valor.
 - Value: Obtiene o establece el **valor** del par de clave y valor.
- Por ejemplo:

```
Hashtable tablaHash = new Hashtable(); // Se crea una nueva tabla hash
tablaHash.Add("txt", "notepad.exe");
tablaHash.Add("bmp", "paint.exe");

foreach (DictionaryEntry elemento in tablaHash)
    Console.WriteLine("Clave = {0}, Valor = {1}", elemento.Key,
        elemento.Value);
```

Interface IDictionary

RECORRER ELEMENTOS

- **Foreach**

- La instrucción **foreach** requiere el tipo de cada elemento de la colección. Como los elementos del objeto **IDictionary** son pares de clave y valor, el tipo del elemento no se corresponde con el tipo de la clave ni con el del valor. En su lugar, el tipo del elemento es **DictionaryEntry**. Por ejemplo

```
foreach (DictionaryEntry elemento in tablaHash)
{
    Console.WriteLine("Clave = {0}, Valor = {1}", elemento.Key,
        elemento.Value);
}
```

SORTEDLIST



- Representa una colección de pares de clave y valor ordenados por claves (implementando IComparer)
- Se puede acceder a los elementos por su clave (IDictionary) o su índice (IList).
 - Índices basados en 0.
- Cada elemento es un par de clave y valor al que se puede obtener acceso como un objeto DictionaryEntry.
- La capacidad es el número de elementos que **SortedList** puede contener. Count es la cantidad que realmente contiene.

ORDENACIÓN:



- Cada nuevo elemento insertado a la colección, se inserta en el orden correcto.
 - Se actualizan los índices, al igual que cuando se quita un elemento.
- Es decir, el índice de un par de clave y valor específico puede cambiar a medida que se agregan o quitan elementos de la colección.
- Comparación con Hashtable:
 - Más lento: se debe ordenar a cada paso.
 - Más flexible: se accede a los elementos por índice o clave.

SortedList

EJEMPLO

// Se crea e inicializa una nueva SortedList.

```
SortedList mySL = new SortedList();
```



// Se agregan elementos

```
mySL.Add("First", "Hola");
```

```
mySL.Add("Second", "Mundo");
```

```
mySL.Add("Third", "!");
```

```
mySL.Add("Antes", "!");
```

// Se muestran sus propiedades:

```
Console.WriteLine("mySL");
```

```
Console.WriteLine(" Count: {0}", mySL.Count); // Salida: 4
```

```
Console.WriteLine(" Capacity: {0}", mySL.Capacity); // Salida: 16
```

```
Console.WriteLine(" Claves y valores:");
```

// Se la recorre y muestra el contenido

```
for (int i = 0; i < mySL.Count; i++)
```

```
{
```

```
    Console.WriteLine("\t{0}:\t{1}", mySL.GetKey(i), mySL.GetByIndex(i));
```

```
}
```

//Antes: ¡

//First: Hola

//Second: Mundo

//Third: !

SortedList

AGREGAR ELEMENTOS

- Add

- Agrega un elemento con la clave y el valor especificados a SortedList. Por ejemplo:

```
SortedList mySL = new SortedList();  
mySL.Add("First", "Hola");  
mySL.Add("Second", "Mundo");  
mySL.Add("Third", "!");  
mySL.Add("Antes", "¡");
```

- También puede utilizar la propiedad Item para agregar nuevos elementos estableciendo el valor de una clave que no existe en **SortedList**; por ejemplo,

```
miColeccion["claveNoExistente"] = nuevoValor
```

- ✦ Sin embargo, si la clave especificada ya existe en la colección SortedList, al establecer la propiedad Item se sobrescribe el valor anterior.

SortedList

RECUPERAR ELEMENTOS



- **GetByIndex**

- Obtiene el valor que se encuentra en el índice especificado.

```
Console.WriteLine(mySL.GetByIndex(2));
```

// Accedemos al elemento cuyo índice es 2, es decir "el". Importante: no olvidar que los elementos se ordenan al agregarlos a la colección.

```
Console.WriteLine(mySL[2]);
```

// Accedemos al elemento cuya CLAVE es 2, es decir: "es"

- **GetKey**

- Se obtiene la clave en el índice especificado.

```
Console.WriteLine(mySL.GetKey(2)); // Salida: 3
```


EJERCICIOS



- Ejercicio 1:
 - Se parte de la siguiente frase:
 - ✦ "Erase una vez un bosque inmenso."
 - Se debe crear un programa que separe cada palabra de la frase y lo almacene en una colección de tipo SortedList. La clave debe ser generada automáticamente y estar conformada por números sucesivos a partir del número que se desee.
 - Lugo se debe mostrar el contenido de la colección de manera **ordenada**. Deben listarse tanto las claves como los valores.

SortedList

QUITAR ELEMENTOS

- Clear



- Quita todos los elementos de la colección. La propiedad Count se establece en cero. Por ejemplo:

```
mySL.Add(1, "Hola");  
mySL.Add(2, "Mundo");  
mySL.Add(3, "!");  
mySL.Add(4, ";");  
mySL.Clear();
```

- Remove y RemoveAt

- Ambos métodos permiten quitar elementos dentro de una SortedList. El método Remove quita el elemento de la colección que coincide con la clave pasada como parámetro. RemoveAt en cambio, recibe como parámetro el índice del elemento que se desea eliminar. Por ejemplo:

```
mySL.Remove(2); // Se elimina el elemento cuya clave es 2  
mySL.Remove(15); // No ocurre nada  
mySL.RemoveAt(0); // Se elimina el elemento cuya clave es 0
```

EJERCICIOS



- Ejercicio 2:
 - Al ejercicio anterior agregarle luego de mostrar los elementos, lo siguiente:
 - ✦ Preguntar al usuario si desea eliminar algún elemento, en caso de ser así pedirle que lo ingrese, quitarlo de la colección y mostrar cómo queda luego de esta acción. Dar al usuario la opción de repetir esta operación o bien de eliminar todos los elementos de una sola vez.

SortedList

BUSCAR ELEMENTOS



- **Contains y ContainsKey**

- Determina si la colección **SortedList** contiene una clave específica, devolviendo true en ese caso o false en otro. Por ejemplo:

```
mySL.Add(1, "Hola");  
mySL.Add(2, "Mundo");  
mySL.Add(3, "!");  
mySL.Add(4, "¡");  
Console.WriteLine(mySL.Contains(2)); // Salida: true  
Console.WriteLine(mySL.ContainsKey(5)); // Salida: false
```

- **ContainsValue**

- Determina si **SortedList** contiene un valor específico. Devuelve **true** en este caso.

```
mySL.Add(1, "Hola");  
mySL.Add(2, "Mundo");  
mySL.Add(3, "!");  
mySL.Add(4, "¡");  
Console.WriteLine(mySL.ContainsValue("hola")); // Salida: false  
Console.WriteLine(mySL.ContainsValue("Hola")); // Salida: true
```

SortedList

BUSCAR ELEMENTOS



- IndexOfKey

- Devuelve el índice de base cero de la clave especificada en **SortedList**.

- IndexOfValue

- Devuelve el índice de base cero de la primera aparición del valor especificado en **SortedList**.

```
mySL.Add(5, "correcto ");
```

```
mySL.Add(2, "es ");
```

```
mySL.Add(4, "orden ");
```

```
mySL.Add(3, "el ");
```

```
mySL.Add(1, "Este ");
```

```
Console.WriteLine(mySL.IndexOfKey(3)); // Salida: 2
```

```
Console.WriteLine(mySL.IndexOfKey(9)); // Salida: -1
```

```
Console.WriteLine(mySL.IndexOfValue("orden ")); // Salida: 3
```

```
Console.WriteLine(mySL.IndexOfValue("Palabras")); // Salida: -1
```

EJERCICIOS



- Ejercicio 3:
 - Al ejercicio anterior agregarle la siguiente funcionalidad:
 - ✦ Se deben eliminar elementos según una clave que el usuario ingrese, según un índice o bien según el elemento. En cada caso el programa deberá controlar si lo que se desea eliminar existe antes de hacerlo, y en caso de que no se encuentre, notificar al usuario de dicho hecho. Si lo encuentra, continúa como corresponde (se elimina el elemento)

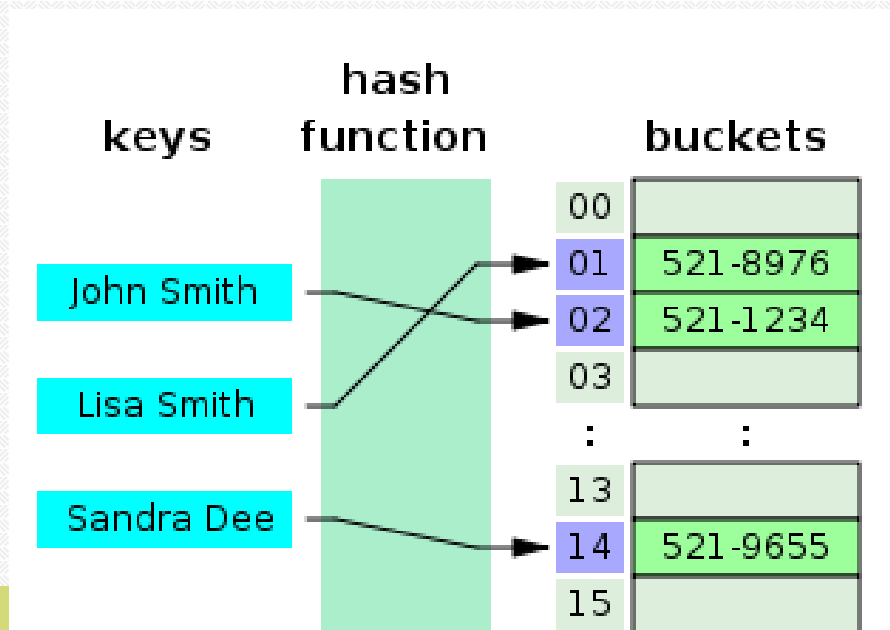
HASHTABLE



- Almacena un par (**clave, valor**) y utiliza la **clave** hash para obtener la ubicación de almacenamiento.
- Permite indexar/localizar un conjunto de elementos a través de una función (función hash).
 - La función Hash es una función que a partir de una clave devuelve la posición en la tabla que le corresponde al elemento.
- Cada elemento es un par de clave y valor al que se puede obtener acceso como un objeto DictionaryEntry

HASHTABLE

- Cuando se agrega un elemento, éste se coloca en un sector de almacenamiento en función del código hash de la clave.
- Las búsquedas utilizarán su código hash para buscar en un sector de almacenamiento determinado solamente.
 - Se reducen considerablemente la cantidad de comparaciones de clave necesarias para encontrar un elemento particular.



Un bucket
contiene un par
clave-valor

HASHTABLE



- Útiles para grandes cantidades de información.
- Almacenan la información en posiciones pseudo-aleatorias, así que el acceso ordenado a su contenido es bastante lento.
- Búsqueda, inserción y borrado muy rápidos.
- Por ejemplo:

```
Hashtable myHT = new Hashtable();  
myHT.Add("uno", "Una");  
myHT.Add("dos", "palabra");
```

Hashtable

AGREGAR ELEMENTOS

- Add

- Agrega un elemento con la clave y el valor especificados a Hashtable. Una clave no puede ser nula, pero un valor sí. Por ejemplo:

```
Hashtable myHT = new Hashtable();  
myHT.Add("uno", "Una");  
myHT.Add("dos", "palabra");
```

- También puede utilizar la propiedad Item para agregar nuevos elementos estableciendo el valor de una clave que no existe en **Hashtable**; por ejemplo,

```
miColeccion["claveNoExistente"] = nuevoValor
```

- ✦ Sin embargo, si la clave especificada ya existe en la colección, al establecer la propiedad Item se sobrescribe el valor anterior.

Hashtable

BUSCAR ELEMENTOS

- Contains y ContainsKey



- Determina si la colección **Hashtable** contiene una clave específica, devolviendo true en ese caso o false en otro. Por ejemplo:

```
myHT.Add("dos", "palabra");  
myHT.Add("tres", "más");  
myHT.Add("cuatro", "otra");  
Console.WriteLine(myHT.ContainsKey("dos")); // Salida: true  
Console.WriteLine(myHT.ContainsKey("cinco")); // Salida: false
```

- ContainsValue

- Determina si **Hashtable** contiene un valor específico. Devuelve **true** en este caso.

```
myHT.Add("dos", "palabra");  
myHT.Add("tres", "más");  
myHT.Add("cuatro", "otra");  
Console.WriteLine(myHT.ContainsValue("Palabra")); // Salida: false  
Console.WriteLine(myHT.ContainsValue("palabra")); // Salida: true
```

EJERCICIOS



- Ejercicio 1:
 - Crear un programa que genere una colección automática de múltiplos. Para ello deberá pedirle al usuario que ingrese un número y el sistema deberá calcular y almacenar en una colección de tipo Hashtable los múltiplos de dicho número desde el 0 al 5000 (como valores de la colección). La clave de cada elemento será generada automáticamente y deberá ser un número, consecutivo uno del otro, del 0 al 5000. el usuario deberá ingresar por cuánto quiere multiplicar el número original y el sistema mostrará el resultado.
 - Por ejemplo, el usuario indica que desea ver la tabla del 6. Luego consulta cuánto es 6×174 (segundo número ingresado), y el sistema muestra el resultado: 1044.

Hashtable

QUITAR ELEMENTOS



- Clear

- Quita todos los elementos de Hashtable. Por ejemplo:

```
Hashtable myHT = new Hashtable();  
myHT.Add("uno", "Una");  
myHT.Add("dos", "palabra");  
myHT.Add("tres", "más");  
myHT.Clear();
```

- Remove

- Quita el elemento con la clave especificada de Hashtable. Si no contiene un elemento con esa clave, **Hashtable** no sufre ningún cambio. Por ejemplo:

```
myHT.Add("tres", "más");  
myHT.Add("cuatro", "otra");  
myHT.Remove("Cuatro"); // No ocurren cambios  
myHT.Remove("cuatro"); // Quita el elemento cuya clave es "cuatro"
```

EJERCICIOS



- Ejercicio 2:
 - Agregar al ejercicio anterior la posibilidad al usuario de indicar si desea limpiar toda la colección y volver a comenzar, para lo cual deberá ingresar nuevamente un número sobre el que desea consultar sus múltiplos y repetir el proceso las veces que el usuario lo desee.

EJERCICIOS



- Ejercicio 3:
 - Recuerde el siguiente ejercicio:
 - ✦ “Se necesita construir un pequeño diccionario que permita al usuario ingresar una palabra en inglés y se muestre por pantalla su traducción/significado en español. El usuario debe poder buscar tantas palabras como desee (se le pregunta si desea continuar). Las palabras a incluir son, por lo menos:
 - Protocol, index, all, user, machine, window, last, abstract, source, device, mouse.”
 - Y desarrolle una solución en la que se utilicen e implementen los nuevos conceptos vistos.