Resumen 2do parcial Laboratorio.

**TIPOS DE COLECCIONES**

Las colecciones se basan en las interfaces

**ICollection, IList e IDictionary**

|  |  |
| --- | --- |
| **IList** | **IDictionary** |
| Cada elemento contiene sólo un valor. | Cada elemento contiene una clave y un valor. |
| Por ejemplo: Array, ArrayList o List | Por ejemplo: Hashtable y SortedList |

**ArrayList**

**Implementa la interfaz IList mediante una matriz cuyo tamaño aumenta dinámicamente según se requiera.**

**Acepta distintos tipos de objetos en la colección.**

|  |  |  |
| --- | --- | --- |
| **Declarar** | ArrayList miArrayList = new ArrayList(); | Declara la Lista. |
| **AGREGAR ELEMENTOS** | | |
| **Add** | miArrayList.Add("Hola");  miArrayList.Add(variable a agregar); | Agrega un objeto al final de ArrayList. |
| **Insert** | miArrayList.Insert(0, "Había "); | Inserta un elemento en el índice especificado. |
| **QUITAR ELEMENTOS** | | |
| **Clear** | miArrayList. Clear(); | Quita todos los elementos de la clase ArrayList. |
| **Remove y RemoveAt** | miArrayList .Remove(“es ")  miArrayList .RemoveAt(2); | Ambos métodos permiten quitar elementos dentro de un ArrayList. \* |
| **BUSCAR ELEMENTOS** | | |
| **IndexOf y LastIndexOf** | (miArrayList.IndexOf("perro")); | Buscan el Object especificado y devuelven el índice de base cero de la primera (o última) aparición. |

\* En el caso de Remove, quita la primera aparición de un objeto concreto de ArrayList que se pasa como parámetro. En el caso de RemoveAt en cambio, el método recibe como parámetro el índice de base cero que se corresponde con el elemento que se desea quitar.

**STRINGCOLLECTION**

Representa una colección de cadenas.

Las comparaciones distinguen entre mayúsculas y minúsculas.

Para conocer el número de cadenas incluidas se utiliza la propiedad Count.

Esta colección no admite otro tipo de datos que no sea string, ya que se trata de una colección especializada.

**System.Collections.Specialized**

|  |  |  |
| --- | --- | --- |
| **Declarar** | StringCollection colCadenas = new StringCollection(); | Declara la Colección. |
| **AGREGAR ELEMENTOS** | | |
| **Add** | colCadenas.Add("Primer texto"); | Agrega un string al final de la colección. |
| **Insert** | colCadenas.Insert(0, "Nuevo primer texto"); | Inserta un elemento en el índice especificado. |
| **QUITAR ELEMENTOS** | | |
| **Clear** | colCadenas.Clear(); | Quita todos los elementos de la colección. |
| **Remove y RemoveAt** | colCadenas.Remove("Tercer Texto");  // Diferencia entre mayúsculas y minúsculas colCadenas.RemoveAt(1); | Ambos métodos permiten quitar elementos dentro de un StringCollection. \* |
| **BUSCAR ELEMENTOS** | | |
| **IndexOf** | Console.WriteLine(colCadenas.IndexOf("Tercer texto"));  // Salida: 2  **// Salida: -1 si no lo encuentra** | Buscan el Object especificado y devuelven el índice de base cero de la primera aparición. |
| **RECORRER ELEMENTOS** | | |
| **Foreach** | foreach (string cadena in coleccion)  {  Console.WriteLine(cadena);  } | Repite un grupo de instrucciones incluidas en el bucle para cada elemento de una matriz o de un objeto collection. \*\* |

**\***En el caso de Remove, quita la primera aparición de un objeto concreto que se pasa como parámetro. En el caso de RemoveAt en cambio, el método recibe como parámetro el índice de base cero que se corresponde con el elemento que se desea quitar.

**\*\*** Se utiliza para recorrer una colección de elementos y obtener la información deseada, pero no se debe utilizar para cambiar el contenido de la colección, ya que se pueden producir efectos secundarios imprevisibles.

**INTERFACE IDICTIONARY**

**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.**

**Clase SortedList - Clase Hashtable**

|  |  |
| --- | --- |
| **Propiedades** | |
| **Key** | **Value** |
| Obtiene o establece la clave del par de clave y valor. | Obtiene o establece el valor del par de clave y valor. |
| Console.WriteLine("Clave = {0}, Valor = {1}", elemento.Key, elemento.Value); | |

|  |  |
| --- | --- |
| **RECORRER ELEMENTOS** | |
| **Foreach** | **foreach (DictionaryEntry elemento in tablaHash)**  **{**  **Console.WriteLine("Clave = {0}, Valor = {1}", elemento.Key, elemento.Value);**  **}** |
| **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.** | |

**SORTEDLIST**

**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.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Muestran sus propiedades** | | | |
| **Console.WriteLine("Count: {0}",mySL.Count);**  **// Salida: 4** | | **Console.WriteLine("Capacity:{0}",mySL.Capacity);**  **// Salida: 16** | |
| **Declarar SortedList** | | | |
| SortedList mySL = new SortedList(); | | | |
| **AGREGAR ELEMENTOS** | | | |
| **Add** | mySL.Add("First", "Hola");  mySL.Add("Second", "Mundo");  mySL.Add("Third", "!"); | | Agrega un elemento con la clave y el valor especificados a SortedList. |
| **Item** | miColeccion["claveNoExistente"] = nuevoValor | | Agregar nuevos elementos estableciendo el valor de una clave que no existe en SortedList. |
| **RECUPERAR ELEMENTOS** | | | |
| **GetByIndex** | Console.WriteLine(mySL.GetByIndex(2));  **// Accedemos al elemento cuyo índice es 2**  Console.WriteLine(mySL[2]);  **// Accedemos al elemento cuya CLAVE es 2** | | Obtiene el valor que se encuentra en el índice especificado.  **Importante:** no olvidar que los elementos se ordenan al agregarlos a la colección. |
| **GetKey** | Console.WriteLine(mySL.GetKey(2));  **// Salida: 3** | | Se obtiene la clave en el índice especificado. |
| **QUITAR ELEMENTOS** | | | |
| **Clear** | mySL.Clear(); | | Quita todos los elementos de la colección. |
| **Remove y RemoveAt** | mySL.Remove(2);  **// Se elimina el elemento cuya clave es 2** | | Ambos métodos permiten quitar elementos dentro de una SortedList. \* |
| **BUSCAR ELEMENTOS** | | | |
| **Contains y ContainsKey** | Console.WriteLine(mySL.Contains(2));  **// Salida: true** Console.WriteLine(mySL.ContainsKey(5));  **// Salida: false** | | Determina si la colección SortedList contiene una clave específica, devolviendo true en ese caso o false en otro |
| **ContainsValue** | Console.WriteLine(mySL.ContainsValue("hola"));  **// Salida: false** Console.WriteLine(mySL.ContainsValue("Hola"));  **// Salida: true** | | Determina si SortedList contiene un valor específico. Devuelve true en este caso. |
| **IndexOfKey** | Console.WriteLine(mySL.IndexOfKey(3));  **// Salida: 2** | | Devuelve el índice de base cero de la clave especificada en SortedList. |
| **IndexOfValue** | Console.WriteLine(mySL.IndexOfValue(“orden”));  **// Salida: 3** Console.WriteLine(mySL.IndexOfValue(“Palabras“)); **// Salida: -1** | | Devuelve el índice de base cero de la primera aparición del valor especificado en SortedList. |

**HASHTABLE**

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.

Útiles para grandes cantidades de información.

Búsqueda, inserción y borrado muy rápidos.

|  |
| --- |
| **Declarar Hashtable** |
| Hashtable myHT = new Hashtable(); |

|  |  |  |
| --- | --- | --- |
| **Add** | myHT.Add("uno", "Una");  myHT.Add("dos", "palabra"); | Agrega un elemento con la clave y el valor especificados a Hashtable. Una clave no puede ser nula, pero un valor sí. |
| **Item** | miColeccion["claveNoExistente"] = nuevoValor | También puede utilizar la propiedad Item para agregar nuevos elementos estableciendo el valor de una clave que no existe en Hashtable; |
| **BUSCAR ELEMENTOS** | | |
| **Contains y ContainsKey** | myHT.Add("dos", "palabra");  Console.WriteLine(myHT.ContainsKey("dos"));  **// Salida: true** | Determina si la colección Hashtable contiene una clave específica, devolviendo true en ese caso o false en otro. |
| **ContainsValue** | myHT.Add("dos", "palabra");  Console.WriteLine(myHT.ContainsValue(“palabra"));  **// Salida: true** | Determina si Hashtable contiene un valor específico. Devuelve true en este caso. |
| **QUITAR ELEMENTOS** | | |
| **Clear** | myHT.Clear(); | Quita todos los elementos de Hashtable. |
| **Remove** | myHT.Add("cuatro", "otra");  myHT.Remove("cuatro");  **// Quita el elemento cuya clave es “cuatro”** | Quita el elemento con la clave especificada de Hashtable. Si no contiene un elemento con esa clave, Hashtable no sufre ningún cambio. |