Colecciones

C#

TECNICATURA SUPERIOR EN PROGRAMACIÓN
LABORATORIO DE COMPUTACIÓN II

UTN
Facultad Regional
San Francisco

Colecciones

- Es un conjunto de objetos con tipos ***similares*** que se agrupan juntos.

- Características básicas:
  - **Capacidad:** número de elementos que puede contener.
  - **Extensión:** número de elementos que realmente contiene.
  - Expanden automáticamente su capacidad cuando ésta alcanza su máximo actual.
  - **Límite inferior:** es el índice de su primer elemento, cero.

Formas de Colección

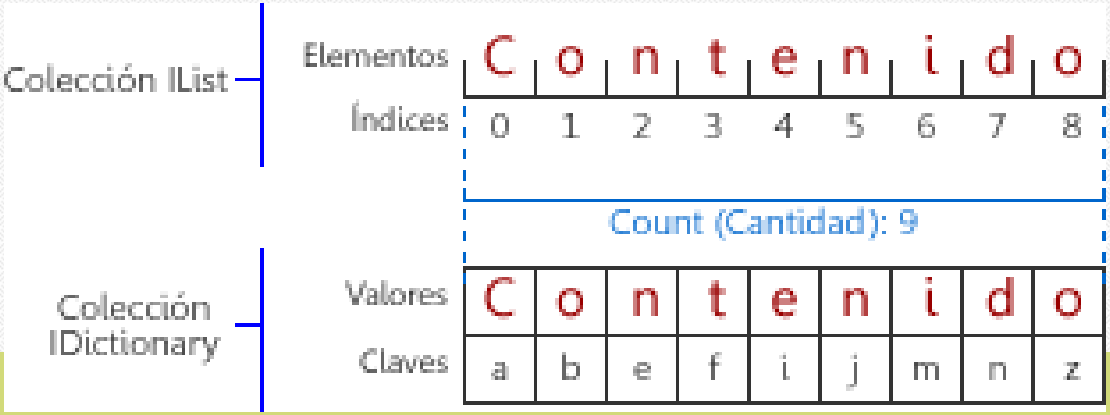
- La forma de una clase Colección hace referencia a la manera en que ésta organiza y almacena los objetos:
  - **Lista**: proporciona una lista de elementos ordenados sin indizar.
  - **Matriz**: proporciona una matriz de objetos indizada por números enteros, ordenada y con el tamaño ajustado dinámicamente.
  - **Mapa**: es una colección que asocia un objeto clave a un objeto de valor.

Ordenada: el orden en que se insertan o eliminan elementos determina su orden en la colección.

Indizada: los elementos de la colección pueden recuperarse por medio de un índice entero, de forma parecida a los elementos de una matriz típica.

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



INTERFACE ICOLLECTION E ILIST

- ICollection define métodos para manipular colecciones genéricas. Por ejemplo:
  - Add
  - Clear
  - Contains
  - Remove
  - CopyTo
- Una implementación de **IList** es una colección de valores a cuyos miembros se puede tener acceso por el índice.
  - Count
  - Item: proporciona acceso a un elemento mediante la sintaxis nombreColeccion[indice]

ARRAYLIST

- Implementa la interfaz IList mediante una matriz cuyo tamaño aumenta dinámicamente según se requiera.
- Capacidad: número de elementos que puede contener el objeto **ArrayList**.
  - La capacidad inicial predeterminada es 0.
- Se puede acceder a los elementos de la colección utilizando un índice entero.
  - Los índices están basados en cero.
- Acepta distintos tipos de objetos en la colección.

EJEMPLO:

```
ArrayList arrayNum = new ArrayList();
arrayNum.Add(1);
arrayNum.Add("algo");
arrayNum.Add('s');
arrayNum.Add(1000);
arrayNum.Add("numero");

    for (int i = 0; i < arrayNum.Count; i++)
        Console.Write(" - {0} - ",arrayNum[i]);

    // Salida: - 1 - - algo - - s - - 1000 - - numero -
```

EJERCICIO EN PARTES

- Parte 1:
  - Declare un objeto del tipo ArrayList. Asigne al mismo un nombre que haga referencia a temperaturas medias registradas en el mes de octubre.
  - Cree un programa donde se le muestre al usuario un menú con las opciones:
    - Agregar.
    - Modificar.
    - Eliminar.
    - Listar elementos.
    - Reiniciar los valores.
    - Salir.

ArrayList

Agregar elementos

- Add
  - Agrega un objeto al final de **ArrayList**. Por ejemplo:

```
// Se crea e inicializa un nuevo ArrayList
ArrayList miArrayList = new ArrayList();
miArrayList.Add("¡");
miArrayList.Add("Hola");
miArrayList.Add("Mundo");
miArrayList.Add("!");
```

- Insert
  - Inserta un elemento en el índice especificado. Por ejemplo:

```
ArrayList miArray2 = new ArrayList();
miArray2.Insert(0, "Había ");
// Insert(Índice en el que se insertará, elemento a insertarse)
miArray2.Insert(1, "una ");
miArray2.Insert(2, "vez ");
miArray2.Insert(3, "un ");
miArray2.Insert(4, "perro");
```

EJERCICIO EN PARTES

- Parte 2:
  - Realice un método que se encargue de pedirle al usuario cuál es la temperatura media que desea agregar y lo añada a la colección.
  - Tenga en cuenta:
    - Lo ingresado debe ser un número válido.
    - Debe corroborar que la lista no esté completa.
    - Debe indicar cuántas temperaturas lleva ya cargadas y cuántas quedarían por cargar.
    - Cada temperatura se carga secuencialmente. Es decir, la primera corresponde al día 1, la segunda al día 2, la tercera al día 3, etc.

ArrayList

Quitar elementos

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

```
ArrayList miArrayQuitar = new ArrayList();
miArrayQuitar.Add("Este ");
miArrayQuitar.Add("es ");
miArrayQuitar.Add("un ");

miArrayQuitar.Clear();
```

- Remove y RemoveAt
  - Ambos métodos permiten quitar elementos dentro de un ArrayList. 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. Por ejemplo:

```
miArrayQuitar.Remove("es ");
miArrayQuitar2.RemoveAt(2);
```

EJERCICIO EN PARTES

- Parte 3:
  - Realice un método encargado de eliminar un elemento particular del ArrayList que contiene las temperaturas.
  - Tenga en cuenta:
    - Debe darle al usuario la opción de eliminar una temperatura indicando el día (por ejemplo, eliminar la temperatura del día 4).
    - Debe corroborar que el listado no esté vacío.
    - Si la eliminación se hace por día, debe controlar que no se intente borrar la temperatura de un día que aún no fue cargado.

ArrayList

Buscar elementos

- IndexOf y LastIndexOf
  - Buscan el **Object** especificado y devuelven el índice de base cero de la primera (o última) aparición en la **ArrayList** completa. Por ejemplo:

```
ArrayList miArray3 = new ArrayList();
miArray3.Add("Había");
miArray3.Add("una");
miArray3.Add("vez");
miArray3.Add("un");
miArray3.Add("perro");
miArray3.Add("y");
miArray3.Add("un");
miArray3.Add("gato");
miArray3.Add(".");

Console.WriteLine(miArray3.IndexOf("perro")); // Salida: 4
Console.WriteLine(miArray3.IndexOf("un")); // Salida: 3
Console.WriteLine(miArray3.LastIndexOf("un")); // Salida: 6
Console.WriteLine(miArray3.IndexOf("gato")); // Salida: 7
```

EJERCICIO EN PARTES

- Parte 4:
  - Añadir la siguiente funcionalidad en el método encargado de borrar un elemento:
    - Se debe dar al usuario la posibilidad de borrar un elemento ingresando cuál es la temperatura que desea quitar (por ejemplo, "15,5").
    - Se debe controlar que la temperatura que desea eliminar exista en el listado.
  - Agregar un método que le permita al usuario ingresar cuál es la temperatura que desea modificar. Tener en cuenta:
    - Al igual que al eliminar, el usuario podrá elegir el día del que desea modificar la temperatura, o ingresar la temperatura en sí.
    - Se debe controlar que la temperatura buscada exista. Se debe controlar que lo ingresado para reemplazar a la temperatura anterior, sea válido.

EJERCICIO EN PARTES

- Parte 5:
  - Si no lo hubiera hecho ya, escriba un método que permita mostrar por pantalla cada uno de los elementos actualmente existentes en el listado. Utilícelo cada vez que sea necesario en los demás métodos.
  - Para la opción restante, utilizar un método que le pregunte al usuario si realmente desea reiniciar los valores ya que esto eliminará toda la información cargada. Si responde afirmativamente, deberá limpiar el listado. Caso contrario, no deberá verse afectado.

EJERCICIO EN PARTES

- Parte 6:
  - Agregue una opción más al menú:
    - Calcular estadísticas.
  - Para ello deberá: mostrar la mayor y menor temperatura. El promedio de todas las temperaturas. La cantidad de temperaturas que estuvieron debajo de la media y la cantidad de temperaturas que estuvieron sobre la media.

StringCollection

- Representa una colección de cadenas.
- Las comparaciones distinguen entre mayúsculas y minúsculas.
- Se puede obtener acceso a los elementos usando un índice de tipo entero.
  - Los índices están basados en cero.
- Para conocer el número de cadenas incluidas se utiliza la propiedad **Count.**
- Para tener acceso a un elemento específico de la colección se usa la sintaxis siguiente:
  - nombreColeccion[indice].

EJEMPLO:

```
StringCollection colCadenas = new StringCollection();
colCadenas.Add("Se pueden agregar cadenas de texto");
colCadenas.Add("Pero no se admiten otros tipos");
colCadenas.Add("Ya que es una colección especializada");
colCadenas.Add(1); // Esto NO es correcto
```

- Esta colección no admite otro tipo de datos que no sea string, ya que se trata de una colección especializada.
- Se debe utilizar el espacio de nombres

System.Collections.Specialized

Agregar elementos

- Add
  - Agrega un objeto al final del StringCollection. Por ejemplo:

```
colCadenas.Add("Primer texto");
colCadenas.Add("Segundo texto");
colCadenas.Add("Tercer texto");
```

- Insert
  - Inserta un elemento en el índice especificado. Por ejemplo:

```
colCadenas.Insert(0, "Nuevo primer texto");
colCadenas.Insert(2, "Texto en la posición 2");
```

StringCollection

Recorrer elementos

- Foreach
  - Repite un grupo de instrucciones incluidas en el bucle para **cada elemento** de una matriz o de un objeto collection. 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.

```
StringCollection coleccion = new StringCollection();
coleccion.Add("Primer texto");
coleccion.Add("Segundo texto");
coleccion.Add("Tercer texto");
foreach (string cadena in coleccion)
{
    Console.WriteLine(cadena);
}
```

EJERCICIO EN PARTES

- Parte 1:
  - Se requiere preparar un programa que gestione los títulos de las películas de las que dispone un pequeño video club. Se deben utilizar métodos para implementar las funcionalidades. Desarrolle por el momento, los siguientes:
    - public static void mostrarColeccion(StringCollection coleccion)
    - public static void agregarTitulo(string nuevaPelicula)
  - Y desde el método main realice las acciones necesarias para su uso. Declare una colección de string, y llame al método tantas veces como necesite para colmar la colección de al menos, 5 títulos de películas. Luego muestre el contenido de la colección.

StringCollection

Buscar elementos

- IndexOf
  - Buscan el **Object** especificado y devuelven el índice de base cero de la primera aparición en la colección completa. Por ejemplo:

```
colCadenas.Add("Primer texto");
colCadenas.Add("Segundo texto");
colCadenas.Add("Tercer texto");

Console.WriteLine(colCadenas.IndexOf("Tercer Texto")); // Salida: -1
Console.WriteLine(colCadenas.IndexOf("Tercer texto")); // Salida: 2
```

StringCollection

Quitar elementos

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

```
colCadenas.Add("Primer texto");
colCadenas.Add("Segundo texto");
colCadenas.Insert(0, "Nuevo primer texto");
colCadenas.Insert(2, "Texto en la posición 2");

colCadenas.Clear();
```

- Remove y RemoveAt
  - Ambos métodos permiten quitar elementos dentro de una StringCollection. 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. Por ejemplo:

```
colCadenas.Add("Primer texto");
colCadenas.Add("Segundo texto");
colCadenas.Add("Tercer texto");
colCadenas.Remove("Tercer Texto"); // Diferencia entre mayúsculas y minúsculas
colCadenas.Remove("Tercer texto");
```

EJERCICIO EN PARTES

- Parte 2:
  - Implemente ahora:
    - public static void eliminarElemento(string titulo, StringCollection coleccion)
    - public static void eliminarTodo(StringCollection coleccion)
  - Tenga en cuenta:
    - Debe corroborar que la colección no esté vacía.
    - Cuando se elimine todo, primero se debe pedir confirmación del usuario.
  - Desde el método main realice las llamadas a estos métodos para corroborar que funcionen correctamente. Muestre el contenido de la colección luego de cada acción.