Trabajo Práctico 1:

Colecciones en .NET

Integrantes: Arévalo Sabrina Ayelén, Caneva Ezequiel

Profesor: José A. Fernandez

Materia: Laboratorio III

Tema: Colecciones en .NET. CollectionUtil

Fecha de entrega: 25.04.23

Introducción

El presente trabajo grupal de investigación tiene como objetivo el buscar y presentar información sobre el tema asignado, justo con una serie de programas en los cuales se utilizarán dicha colección para justificar lo escrito en el trabajo de investigación.

Se presentará la colección CollectionUtil, cuya principal función es el de no distinguir el uso de mayúsculas y minúsculas. Se mostrará cómo implementarla, qué usos se le puede dar, qué métodos utiliza y en dónde se debe evitar su uso, esto junto con pruebas unitarias y su aplicación en un programa winform.

Clase CollectionUtil

CollectionUtil es una clase heredada de Objet. Tiene como función crear colecciones que omitan las mayúsculas y las minúsculas en las cadenas.

Su namespace es System.Collections.Specialized, este contiene colecciones para usos específicos y están fuertemente tipadas (diccionario de lista enlazada, vector de bits, colecciones que contienen solo cadenas). Además, usa el ensamblado System.Collections.NonGeneric.dll, que se encarga de proporcionar clases que definen colecciones de objetos no genéricas como listas, hashtables, diccionarios, etc.

Usos

Un ejemplo de los usos que se le puede dar es el de un programa que almacene valores de una población en una tabla hash o en una lista ordenada. A cada valor se le asigna un nombre llamada clave para que se tenga un acceso más rápido a los valores.

En la siguiente imagen sacada del sitio web de Microsoft se podrá mostrar dicho uso con mayor claridad:

```
using System;
using System.Collections;
using System.Collections.Specialized;

class TestCollectionsUtils
{
    public static void Main()
    {
        Hashtable population1 = CollectionsUtil.CreateCaseInsensitiveHashtable();

        population1["Trapperville"] = 15;
        population1["Doggerton"] = 230;
        population1["New Hollow"] = 1234;
        population1["McHenry"] = 185;

        // Select cities from the table using mixed case.
        Console.WriteLine("Case insensitive hashtable results:\n");
        Console.WriteLine("{0}'s population is: {1}", "Trapperville", population1["trapperville"]);
        Console.WriteLine("{0}'s population is: {1}", "Doggerton", population1["DOGGERTON"]);
        Console.WriteLine("{0}'s population is: {1}", "New Hollow", population1["New hoLLow"]);
        Console.WriteLine("{0}'s population is: {1}", "McHenry", population1["MchenrY"]);

        SortedList population2 = CollectionsUtil.CreateCaseInsensitiveSortedList();

        foreach (string city in population1.Keys)
        {
           population2.Add(city, population1[city]);
        }

        // Select cities from the sorted list using mixed case.
        Console.WriteLine("\nCase insensitive sorted list results:\n");
        Console.WriteLine("{0}'s population is: {1}", "Trapperville", population2["trapPeRVille"]);
        Console.WriteLine("{0}'s population is: {1}", "Doggerton", population2["dOGGeRtON"]);
        Console.WriteLine("{0}'s population is: {1}", "New Hollow", population2["nEw hOLLOW"]);
        Console.WriteLine("{0}'s population is: {1}", "McHenry", population2["MchEnrY"]);
    }
}

// This program displays the following output to the console
//
// Case insensitive hashtable results:
//
// Trapperville's population is: 15
// Doggerton's population is: 230
// New Hollow's population is: 1234
// McHenry's population is: 185
//
// Case insensitive sorted list results:
//
// Trapperville's population is: 15
// Doggerton's population is: 230
// New Hollow's population is: 1234
// McHenry's population is: 185
```

Nota: Microsoft (https://learn.microsoft.com/es-es/dotnet/api/system.collections.specialized.collectionsutil?view=net-7.0)

Métodos

[…] ” Los métodos generan una instancia que no distingue mayúsculas de minúsculas de la colección mediante implementaciones del proveedor de código hash y el comparador. La instancia resultante se puede usar como cualquier otra instancia de esa clase, aunque puede comportarse de forma diferente.”

- **CreateCaseInsensitiveHashtable():** método que crea una nueva instancia que no tiene en cuenta el uso de mayúsculas y minúsculas de la clase hashtable con la capacidad inicial por defecto.
- **CreateCaseInsensitiveHashtable(IDictionary):** se encarga de la copiar las entradas del diccionario especificado a una nueva instancia que no distingue entre mayúsculas y minúsculas de la clase Hashtable con la misma capacidad inicial que el número de entradas copiadas.

- **CreateCaseInsensitiveHashtable(Int32).**
- **CreateCaseInsensitiveSortedList():** Crea una nueva instancia de la clase SortedList, que omite el uso de mayúsculas y minúsculas en las cadenas.
- **Equals(Objet):** compara si un objeto especificado con anterioridad es igual que el objeto actual.
- **GetHashCode():** Sirve como función hash predeterminada.
- **GetType():** Obtiene el tipo de dato de la instancia actual.
- **MemberwiseClone():** Crea una copia superficial del objeto actual.
- **ToString():** Convierte un objeto en su representación de cadena para que sea adecuado para mostrarlo por pantalla.

Aplicación

Producto	Versiones
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8
.NET Standard	2.0, 2.1
UWP	10.0
Xamarin.iOS	10.8
Xamarin.Mac	3.0

Evitar su uso

A diferencia de un hashtable que puede admitir un escritos y varios lectores en simultaneo, pero para admitir escritores se lo hace por medio del contenedor devuelto por el método Synchronized(Hashtable). Un SortedList puede admitir varios lectores simultáneamente, siempre y cuando no se modifique la colección. Esto para poder garantizar la seguridad de subprocesos de SortedList, todas las operaciones deben realizarse a través del contenedor devuelto por el método Synchronized(SortedList).

Otro escenario en el que se debe tener cuidado es en el de la enumeración, ya que a través de una colección no es intrínsecamente un procedimiento seguro para subprocesos. Sin importar que la colección esté sincronizada, los subprocesos pueden modificar su valor, lo cual hace que ocurra una excepción.

Una posible solución para poder garantizar la seguridad para la ejecución de subprocesos durante la enumeración, es por medio del bloqueo de la colección durante toda la enumeración. Otra forma es detectando las excepciones resultantes de los cambios realizados por otros subprocesos.

Test

Los tests unitarios mostrarán con mayor detalle el uso de los métodos que nos permite usar la colección.

El primer test usa un Hashtable para almacenar los elementos, este implementa mediante el constructor de CollectionUtil el método CreateCaseInsensitiveHashtable() y almacena la cantidad de habitantes de cada ciudad, teniendo como nombre clave a la ciudad y como valor a la cantidad de personas. Para verificar que todo esté bien, se compara si el código con el que se fue asignada cada ciudad es el mismo que si se accede a el con el nombre clave asignado, variando el orden de mayúsculas y minúsculas.

```
[Fact]
0 referencias
public void Test1()
{
    Hashtable ciudades1 = CollectionsUtil.CreateCaseInsensitiveHashtable();

    ciudades1["Corrientes"] = 352646;
    ciudades1["Goya"] = 66129;
    ciudades1["Resistencia"] = 290723;
    ciudades1["Paso de la Patria"] = 5598;

    Assert.Equal(352646, ciudades1["cOrRIeNTES"]);
    Assert.Equal(66129, ciudades1["GOYA"]);
    Assert.Equal(290723, ciudades1["resisTENCIA"]);
    Assert.Equal(5598, ciudades1["Paso de LA PAtria"]);

}
```

Este test crea un diccionario que almacena strings que serán usados como nombre clave y también valores del tipo int. Se encarga de guardar nombres de libros con su respectiva cantidad de páginas.

Como se mencionó en el test anterior, se crea un hashtable, pero la diferencia está en que se accederán a los elementos del diccionario por medio del constructor y el uso del método CreateCaseInsensitiveHashtable(IDictionary).

Para finalizar se compara el valor con el nombre clave que se le dio a cada elemento.

```
[Fact]
0 referencias
public void Test2() {
    //
    Dictionary<string, int> libros_paginas = new Dictionary<string, int>();

    libros_paginas.Add("Libro1", 300);
    libros_paginas.Add("Libro2", 250);
    libros_paginas.Add("Libro3", 380);
    libros_paginas.Add("Libro4", 500);


    Hashtable libros = CollectionsUtil.CreateCaseInsensitiveHashtable(libros_paginas);

    Assert.Equal(300, libros["LIBRO1"]);
    Assert.Equal(250, libros["LIbro2"]);
    Assert.Equal(380, libros["liBRO3"]);
    Assert.Equal(500, libros["LiBrO4"]);

}
```

Como tercer y último test se creó una SortedList, que al igual que es Test1, almacena la cantidad de habitantes que tiene cada ciudad. Como anteriormente se hizo, por medio del constructor se implementa el método CreateCaseInsensitiveSortedList() para omitir las mayúsculas y minúsculas en la lista ordenada. Finalizando el test con la comparación que se hizo en cada una de las pruebas, dando como resultado un correcto funcionamiento.

```
[Fact]
0 referencias
public void test3() {

   SortedList ciudades2 = CollectionsUtil.CreateCaseInsensitiveSortedList();
    ciudades2.Add("Corrientes", 352646);
    ciudades2.Add("Goya", 66129);
    ciudades2.Add("Resistencia",290723);
    ciudades2.Add("Paso de la Patria", 5598);

    Assert.Equal(352646, ciudades2["cOrRIeNTES"]);
    Assert.Equal(66129, ciudades2["GOYA"]);
    Assert.Equal(290723, ciudades2["resisTENCIA"]);
    Assert.Equal(5598, ciudades2["Paso de LA PAtria"]);
}
```

Proyecto Winfrom

Opción 1

El proyecto busca generar botones en tiempo de ejecución implementando CollectioUtil. Estos botones se encuentran en un panel de control y deben ser removidos uno por uno por medio de un botón específico.

Los botones asignados a los operadores fueron los que se utilizaron para implementar dicha colección, estos fueron almacenados en una tabla hash y a las variables se les fue asignado un valor de tipo string y un nombre clave representativo de cada operador.

También se creo una lista de String en el cual se le asigno los mismos caracteres que en la tabla hash pero alternando entre mayúsculas y minúsculas.

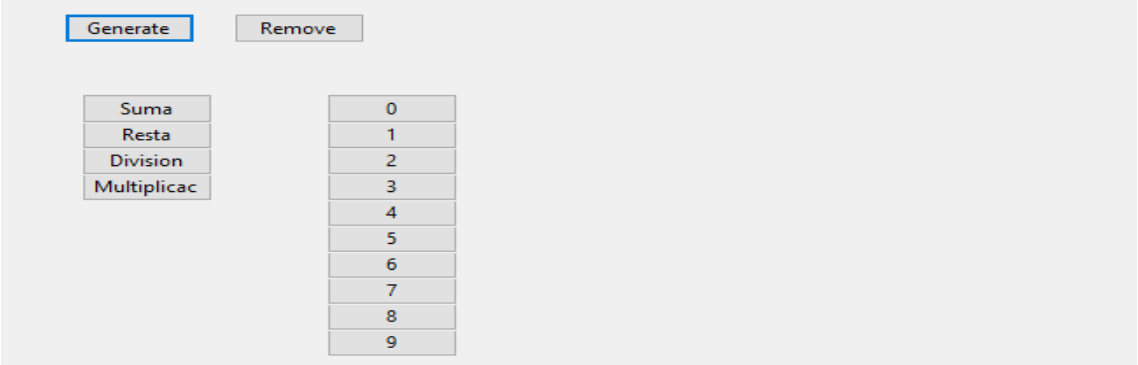
```
Hashtable botones = CollectionsUtil.CreateCaseInsensitiveHashtable();
botones["Sumar"] = "Suma";
botones["Restar"] = "Resta";
botones["Dividir"] = "Division";
botones["Multiplicar"] = "Multiplicacion";

List<String> otralist = new List<String>();

otralist.Add("SUMAR");
otralist.Add("rEsTar");
otralist.Add("DivIDir");
otralist.Add("MulTIPLIcar");
```

Luego instanciamos una lista genérica de clase Button e implementamos un ciclo iterativo for para la creación de dichos botones, utilizando el método Text se le

añadió el nombre al botón colocando dentro de los corchetes del Hashtable los elementos de la lista de String junto al método ToString(). Luego se utilizó el método Location para dar la posición de cada botón, a su vez cada botón se adhirió a la lista Button creada y también al panel.

```
lista = new List<Button>();
for (int i = 0; i < 4; i++)
{
    var boton = new Button();
    boton.Text = botones[otralist[i]].ToString();
    boton.Location = new Point((10), (i * 20));
    panelCalc.Controls.Add(boton);
    lista.Add(boton);
}
```

Como resultado se obtuvo los 4 botones esperados sin importar que en la lista de String se hayan alternado las mayúsculas y minúsculas.



Para finalizar, se agregó un botón cuya función es la de remover uno por uno los botones del panel. Para que realice esa acción se usaron los métodos Remove(), encargado de remover todos los elementos que van posicionándose en la posición 0 de la lista, para especificar que se va a borrar los elementos en esa posición se usa el método ElementAt(0).

Para poder ver esos cambios en el panel, se usa el método RemoveAt(0) para remover los elementos en una lista a partir de la posición especificada.

Todo esto se encuentra en un if que controla que la lista no esté vacía usando Cout que va controlando la cantidad de elementos que contiene la lista.

```
1 referencia
private void Remove_Click(object sender, EventArgs e)
{

    if (lista.Count != 0)
    {

        lista.Remove(lista.ElementAt(0));
        panelCalc.Controls.RemoveAt(0);
    }

}
```

Opción 2

A diferencia de la primera forma de implementación, en este caso se crearon los botones de los operadores individualmente accediendo a su valor por medio de la variable botones, usada para Hastable y entre paréntesis el nombre con el que se accede al valor, en donde claramente se ve que no distingue el uso de mayúsculas y minúsculas.

Par finalizar y mostrar el valor que contiene, se utiliza el método ToString().

```
//Creación de los botones

var bAdd = new Button();
bAdd.Text = botones["SumAr"].ToString(); //Muestra el valor Suma

bAdd.Location = new Point(90, 0);
panelCalc.Controls.Add(bAdd);

lista.Add(bAdd);

var bResta = new Button();
bResta.Text = botones["RESTAr"].ToString(); //Muestra el valor Resta

bResta.Location = new Point(180, 0);
panelCalc.Controls.Add(bResta);

lista.Add(bResta);

var bMulti = new Button();
bMulti.Text = botones["MultIPlicacion"].ToString(); //Muestra el valor Multiplicar

bMulti.Location = new Point(180, 20);
panelCalc.Controls.Add(bMulti);

lista.Add(bMulti);

var bDiv = new Button();
bDiv.Text = botones["DivISion"].ToString(); //Muestra el valor Dividir

bDiv.Location = new Point(270, 40);
panelCalc.Controls.Add(bDiv);

lista.Add(bDiv);
```

Link al repositorio compartido de GitHub

https://github.com/Ezequiel-Caneva/CollectionUtil1

Fuentes

https://learn.microsoft.com/es-es/dotnet/api/system.collections.specialized.collectionsutil?view=net-7.0

https://learn.microsoft.com/es-es/dotnet/api/system.collections.specialized?view=net-7.0