Clase 12

Martes, 3 Octubre 2023

Indice

Inicializadores de Listas

- Podemos declarar e inicializar las Listas/Arrays de diferentes maneras:

1. Declaración e inicialización por separado:

```
'
public static void CreateList()
{
    List<int> list;
    list = new List<int>();

    list.Add(1);
    list.Add(5);
    list.Add(10);
}
'
```

2. Declaración en una línea:

```
'
public static void CreateList()
{
    List<int> list = new List<int>();

    list.Add(1);
    list.Add(5);
    list.Add(10);
}
'
```

3. Declaración e inicialización 'al vuelo':

```
'
public static void CreateList()
{
    List<int> list = new List<int>
    {
        1,
        5,
        10
    };
}
'
```

4. Declaración e inicialización en la misma linea:

```
'
public static void CreateList()
{
    List<int> list = new List<int>{1, 5, 10};
}
'
```

Inicializadores de Arrays

- Los arrays se crean de manera semejante a las listas

1. Declaración e inicialización por separado:

```
'
public static void CreateArray()
{
    int[] list;
    list = new int[3];
```

```
    list[0] = 1;
    list[1] = 5;
    list[2] = 10;
}
'
```

2. Declaración en una línea:

```
'
public static void CreateArray()
{
    int[] list = new int[3];

    list[0] = 1;
    list[1] = 5;
    list[2] = 10;
}
'
```

3. Declaración e inicialización 'al vuelo':

```
'
public static void CreateArray()
{
    int[] list = new int[]
    {
        1,
        5,
        10
    };
}
'
```

4. Declaración e inicialización en la misma linea:

```
'
public static void CreateArray()
{
    int[] list = new int[]{1, 5, 10};
}
'
```

Funciones que se utilizan habitualmente en Listas y Arrays

- **FUNCIÓN:** Se le pase una lista de enteros y un valor. Devuelve 'true' o 'false' si el valor está dentro de la lista.

```
'
public class Functions
{
    public static bool ContainsNumber(List<int> list, int number)
    {
        if( list == null || list.Count == 0)
            return false;

        for(int i = 0; i < list.Count; i++)
        {
            if (list[i] == number)
                return true;
        }
        return false;
    }
}
'
```

- Es buena idea utilizar el nombre de función ***'Contains'*** para evaluar si una lista contiene un elemento.
- Es aconsejable **validar los parámetros de entrada** de la función. Hay que comprobar que la lista no apunte a null y que no esté vacía.
- Queda ***prohibido*** el uso de la función Remove(). Será motivo de suspenso su uso.

- Podemos hacer *'folding'* con snippets de código siempre que queramos simplificar el código escrito. Para ello empleamos #region y #endregion.

- **FUNCIÓN:** Se le pasa una lista de enteros y te devuelve el valor mayor.

```
'
public class ListExample
{
    public static int GetMajor(List<int> list)
    {
        if( list == null || list.Count == 0)
            return int.MinValue;

        int result = list[0];
        for(int i = 0; i < list.Count; i++)
        {
            if(list[i] > result)
                result = list[i];
        }
        return result;
    }
}
'
```

- **FUNCIÓN:** Se le pasa una lista de enteros y te devuelve la posición del valor mayor.

```
'
public class ListExample
{
    public static int GetMajor(List<int> list)
    {
        if( list == null || list.Count == 0)
            return -1;

        int aux = list[0];
        int index = 0;

        for(int i = 0; i <& list.Count; i++)
        {
            if(list[i] > aux)
            {
                index = i;
                aux = list[i];
            }
        }
        return index;
    }
}
'
```

- Los valores de índice se establecen por convención como ***'index'***.
- Podemos llamar a la función anterior para buscar el valor mayor, pero tendríamos el problema de recorrer dos *for* distintos, con el doble de carga para el programa. Es preferible crear un único for que resuelva este problema.
- Debemos validar dos entradas en una lista: que no apunte a null, y que la lista no esté vacía.

```
'
public static int Validate()
{
    if(list == null)            // La lista apunta a null
        return 0;

    if(list.Count == null)      // La lista no contiene elementos.
        return 0;

    if(list == null)
        return -1;              //  Si se trata de índices devuelve -1.
}
'
```

- **FUNCIÓN:** Devuelve 'true' o 'false' si una lista que se le pasa por parámetros está ordenada.

```
'
public class ListExample
{
    public static bool IsOrdered(List<int> list)
    {
        if (list == null || list.Count == 0)
            return false;

        bool result = true;

        for(int i = 0; i < list.Count - 1; i++)
        {
            if (list[i] > list[i + 1])
                return = false;
        }

        if(list[list.Count - 1] > list[list.Count])
            return = false;

        return result;
    }
}
'
```

- **FUNCIÓN:** Ordena los valores de una lista de manera *ascendente*.

```
'
public class ListExample
{
    public static void SortAscendent(List<int> list)
    {
        if (list == null || list.Count == 0)
            return null;

        int aux;

        for(int i = 0; i < list.Count - 1; i++)
        {
            for(int j = list.Count - 1; j > 0; j--)
            {
                if (list[j] < list[j - 1])
                    {
                        aux = list[j - 1];
                        list[j - 1] = list[j];
                        list[j] = aux;
                    }
            }
        }
    }
}
'
```