Consultas sobre colecciones

El tipo **IEnumerable<T>** representa una secuencia de elementos. Como Ud conoce los array (como **int[]**, **string[]**), las listas (**List<T>**), las pilas (**Stack<T>**) y las colas (**Queue<T>**) son implementaciones de **IEnumerable<T>**

A partir de un **IEnumerable<T>** se pide poder crear objetos de tipo **Consulta<T>** que debe ser una implementación de **IEnumerable<T>** con métodos para realizar determinadas acciones de selección y búsqueda.

Usted deberá implementar el tipo **Consulta<T>** donde el parámetro genérico **T** representa el tipo de los elementos sobre los que trabaja la consulta.

```
public class Consulta <T> : IEnumerable<T> { … }
```

Como el tipo `Consulta<T>` es `IEnumerable<T>` debe implementar el método `GetEnumerator` que devuelve un `IEnumerable<T>`.
El tipo debe tener otros métodos que a su vez devuelven un objeto consulta como resultado de iterar sobre los elementos de la consulta y hacer determinadas operaciones que se mencionan a continuación.

CONSTRUCTOR DEL TIPO CONSULTA

```
public Consulta(IEnumerable<T> e)
{
    …
}
```

Permite crear una instancia del tipo **Consulta<T>** a partir de un objeto **IEnumerable<T>**.

ITERADOR DEL TIPO CONSULTA

```
public IEnumerator<T> GetEnumerator()
{
    …
}
```
Por ejemplo la sentencia:

```
foreach (int x in new Consulta<int>(new int[] { 1, 2, 3, 4, 5 }))
  Console.WriteLine(x);
```
Imprime los números del array en la consola ya que el objeto `Consulta` inicial es el propio array.

FILTRAR

```
public Consulta<T> Filtra(Func<T, bool> predicado)
{
    …
}
```

Este método devuelve una nueva consulta con los elementos que cumplan el predicado expresado mediante el delegado **`Func<T, bool>`**. Este delegado representa funciones que reciben un parámetro de tipo **`T`** y devuelven un valor **`bool`**. Este parámetro toma el valor del elemento que se desea saber si cumple o no con el filtro. El método almacenado en **`predicado`** devolverá valor **`true`** si el elemento pasa el filtro o valor **`false`** en caso contrario.

Por ejemplo si se desean obtener los números que sean pares a partir de un `IEnumerable<int>` puede escribirse:

```
Consulta<int> numeros = new Consulta<int>(new int[] { 1, 2, 3, 4, 5 });
Consulta<int> pares = numeros.Filtra(e => e % 2 == 0);
foreach (int x in pares)
    Console.WriteLine(x);
```

Note el uso de una expresión lambda `e => e % 2 == 0` para plantear el predicado "es par". Recuerde que las expresiones lambda son formas sintácticas más cómodas para expresar delegados. Otras formas de llamar al método **`Filtra`** podrían haber sido:

```
…
static bool EsPar(int e) { return e % 2 == 0; }
…

foreach (int x in new Consulta<int>(new int[] { 1, 2, 3, 4, 5 }).Filtra(EsPar))
   Console.WriteLine(x);
```

o

```
foreach (int x in new Consulta<int>(new int[] { 1, 2, 3, 4, 5 })
  .Filtra(delegate(int e) { return e % 2 == 0; }))
     Console.WriteLine(x);
```

TRASFORMA.

```
public Consulta<R> Transforma<R>(Func<T, R> transformacion)
{
    …
}
```

El método `Transforma` se aplica a una consulta y devuelve una nueva consulta con los elementos resultantes de transformar cada elemento de la consulta actual.
Por ejemplo

```
string[] palabras = new string[] { "Hola", "Mundo", "Programación" };

Consulta<int> longitudes =
  new Consulta<string>(palabras).Transforma<int>(p=>p.Length);

foreach (int n in longitudes)
    Console.WriteLine(n);
```

Da como resultado las longitudes de las cadenas (es decir a formado una nueva `Consulta` con elementos de tipo `int` que son las longitudes de las cadenas originales)

```
5
12
```

EXISTE

```
public bool Existe(Func<T, bool> predicado)
{
    …
}
```

El método `Existe` devuelve un `bool` que indica si todos los elementos de la consulta cumplen con el predicado.

Por ejemplo la ejecución del código siguiente
```
new Consulta<string>(new string[] { "A", "B", "C" }).Existe (x=>x.ToLower() == "c")
```

devuelve **`true`** porque hay una cadena que convertida a minúscula es igual a `"c"`

PARA TODOS

```
public bool ParaTodos(Func<T, bool> predicado)
{
    …
}
```
Devuelve true si todos los elementos de la consulta cumplen con el predicado

CANTIDAD

```
public int Cantidad
{
    get
    {
        …
    }
}
```

Devuelve la cantidad de elementos de la consulta actual.

Por ejemplo:

```
string[] colores = new string[] { "azul", "blanco", "rojo", "amarillo"};
Console.WriteLine ("Cantidad de cadenas de longitud mayor de 5 {0}",
                    colores.Filtrar(s => s.Length>5).Cantidad);
debe escribir como resultado 2
```

SALTA

El método

```
public Consulta<T> Salta(int n)
{
    …
}
```

devuelve una consulta con los elementos de la consulta original luego de haber obviado los primeros `n` elementos, si `n` es mayor o igual que la cantidad de elementos de la consulta original entonces la consulta que devuelve es vacía (es decir sin elementos).