Iterando por los grupos

Los elementos de un conjunto pueden ser agrupados por algún tipo de característica. Por ejemplo el conjunto de los números naturales { 2, 4, 5, 7, 9, 10, 14, 132 } puede ser agrupado según su paridad como pares: { 2, 4, 10, 14, 132 } e impares: { 5, 7, 9 }; o puede ser agrupado por cantidad de dígitos de la forma:

1 dígito: { 2, 4, 5, 7, 9 }

2 dígitos: { 10, 14 }

3 dígitos: { 132 }

De igual forma se podría hacer con un conjunto de personas y agruparlos por edad, o sexo, o por la primera letra del nombre.

La manera en la que se extrae una característica puede quedar planteada por la interfaz `ISelector<T,K>`, con una definición como la que se muestra.

```
publicinterfaceISelector<T, K>
{
    K Select(T obj);
}
```

Esta interfaz es genérica en el tipo de objeto que está analizando (parámetro genérico `T`) y el tipo de la característica extraída (parámetro genérico `K`). Por ejemplo, si se quisiera diseñar un selector que permitiese distinguir si un número entero es par o no se podría implementar de la siguiente forma:

```
publicclassParitySelector : ISelector<int, bool>
{
publicbool Select(intobj)
{
returnobj % 2 == 0;
    }
}
```

Dado un objeto de tipo `ISelector<T, K>` se pueden agrupar los elementos de un conjunto (expresado mediante un objeto de tipo `IEnumerable<T>`) en grupos donde los elementos tengan en común el valor de la característica extraída utilizando el selector.**No debe asumir que K será un tipo IComparable**.

La forma en que se pueden recorrer los elementos de un conjunto de manera agrupada será definida mediante la interfaz `IGroupEnumerator<T, K>`.

```
///<summary>
/// Define funcionalidades para un iterador de grupos.
///</summary>
///<typeparamname="T">El tipo de los elementos enumerados.</typeparam>
///<typeparamname="K">El tipo de la característicacomún para cada grupo.</typeparam>
publicinterfaceIGroupEnumerator<T, K>
{
///<summary>
/// Se mueve hacia el próximo elemento en el grupo actual.
/// Si no hay más elementos en el grupo devuelve false, de lo contrario devuelve true.
/// Lanza InvalidOperationExcepcion si se invoca antes de MoveNextGroup.
///</summary>
///<returns>Un valor booleano indicando si hubo un próximo elemento.</returns>
boolMoveNext();

///<summary>
/// Se mueve hacia el próximo grupo y devuelve true; si no hay mas grupos devuelve false.
///</summary>
///<returns>Devuelve un valor bool indicando si se pudo mover al próximo grupo.</returns>
boolMoveNextGroup();

///<summary>
/// Devuelve el valor de la característica que identifica al grupo actual.
/// Lanza InvalidOperationExcepcion si se invoca antes de MoveNextGroup.
///</summary>
    K CurrentKey{ get; }

///<summary>
/// Devuelve el valor del elemento actual del grupo que se está iterando.
/// Lanza InvalidOperationExcepcion si se invoca antes de MoveNext.
///</summary>
    T Current { get; }

///<summary>
/// Reinicia la iteración de los grupos.
///</summary>
voidResetGroups();

///<summary>
/// Reinicia la iteración de los elementos del grupo actual.
/// Lanza InvalidOperationExcepcion si se invoca antes de MoveNextGroup.
///</summary>
voidReset();

///<summary>
/// Devuelve el selector que está siendo usado para agrupar los elementos.
///</summary>
ISelector<T, K> Selector { get; }
}
```

Esta interfaz (muy parecida a `IEnumerator<T>`) permite movernos dentro de los elementos de un grupo utilizando el método `MoveNext`, referirse al elemento actual con la propiedad `Current`, movernos al próximo grupo utilizando `MoveNextGroup` y referirnos a la característica común del grupo actual mediante la propiedad `CurrentKey`. Además permite reiniciar la enumeración de los elementos del grupo actual mediante `Reset` y reiniciar la enumeración de todos los grupos mediante `ResetGroups`.

A continuación se detallan estas funcionalidades y el comportamiento que deberá tener una implementación de esta interfaz.

`Selector`: devuelve el selector (objeto de tipo `ISelector<T, K>`) que está siendo usado por el iterador para agrupar los elementos.

`MoveNextGroup`: se posiciona en el siguiente grupo pero no en el primer elemento de dicho grupo. Habría que hacer `MoveNext` para posicionarnos en el primer elemento como tal. Si se terminaron los grupos, este método devolverá `false`, de lo contrario devuelve `true`. Para poder hacer `MoveNextGroup` no es necesario haber recorrido los elementos del grupo.

`MoveNext`: posiciona el iterador en el próximo elemento de un grupo. Si el grupo no tiene más elementos, este método devolverá `false` (aunque no se hayan acabado todos los elementos del conjunto original). Si no se ha hecho `MoveNextGroup` este método deberá lanzar `InvalidOperationException`.

`CurrentKey`: Esta propiedad devuelve el valor de la característica que es común a todos los elementos del grupo actual que se está recorriendo. Si no se ha hecho `MoveNextGroup` esta propiedad deberá lanzar `InvalidOperationException`.

`Current`: Esta propiedad devuelve el valor del elemento actual del grupo que se está recorriendo. Si no se ha hecho `MoveNext` esta propiedad deberá lanzar `InvalidOperationException`.

`ResetGroups`: Reinicia la iteración de los grupos.

`Reset`: Reinicia la iteración de los elementos del grupo actual. Si la iteración por los grupos aún no ha comenzado este método deberá lanzar `InvalidOperationException`.

El siguiente código permitiría mostrar todos los elementos de un conjunto agrupados por característica.

```
publicstaticvoid Print<T, K> (IGroupEnumerator<T, K> e)
{
while (e.MoveNextGroup())
    {
Console.WriteLine(e.CurrentKey);
Console.WriteLine("-----------");
while (e.MoveNext())
Console.WriteLine("  " + e.CurrentKey);
    }
}
```

Usted deberá implementar un tipo denominado `Grouper`dentro de una biblioteca de clases (**Class Library**) denominada `Grouping`. El tipo `Grouper` debe definirse como sigue:

```
namespace Grouping
{
publicclassGrouper
    {
publicstaticIGroupEnumerator<T, K>GroupBy<T, K>(IEnumerable<T> elements, ISelector<T, K>
selector)
{
/// Reemplace esta línea por la implementación de su método.
thrownewNotImplementedException();
        }
    }
}
```

Como se ilustra en el código el método `GroupBy` es el que permite obtener un iterador de grupos (objeto de tipo `IGroupEnumerator<T, K>`) a partir de un conjunto de elementos (expresado a través de un objeto de tipo `IEnumerable<T>`) y un objeto selector que permite extraer determinada característica para cada elemento.

Algunos ejemplos

Se le proveerá de un proyecto de tipo Aplicación de Consola (**ConsoleApplication**) con un conjunto de selectores definidos tanto para enteros como para personas (tipo `Person` provisto también en el proyecto). Este ejemplo crea par de conjuntos y los agrupa usando varios criterios. El resultado en consola debe ser:

```
Agrupando pares e impares...-------------------------
True
-----------
  2
  4
  10
  14
  132
False
-----------
  5
  7
  9

Agrupando por cantidad de dígitos...-------------------------
1
-----------
  2
  4
  5
  7
  9
2
-----------
  10
  14
```

```
3
-----------
  132

Agrupando por primera letra...-------------------------
J
-----------
  Juan (20 años)
A
-----------
  Ana (19 años)
  Alberto (23 años)
C
-----------
  Carlos (20 años)
  Chucho (19 años)
M
-----------
María (18 años)

Agrupando por edad...-------------------------
20
-----------
  Juan (20 años)
  Carlos (20 años)
19
-----------
  Ana (19 años)
  Chucho (19 años)
23
-----------
  Alberto (23 años)
18
-----------
María (18 años)

Press any key to continue . . .
```

Nota: No debe asumir que con los selectores de ejemplo es suficiente para probar su solución.