Relaciones de equivalencia en .NET

Una **relación** sobre un conjunto *C* es un subconjunto de pares del conjunto *CxC*. Por ejemplo, el conjunto {(1,1), (2,3), (2,2), (3,3)} es una relación válida sobre el conjunto {1, 2, 3}.

Una **relación de equivalencia** cumple las siguientes propiedades:

- Todo elemento *x* perteneciente a C está relacionado consigo mismo. **Reflexividad**.
- Si *x*ϵC está relacionado con *y*ϵC entonces el elemento *y* está relacionado con el elemento *x*. **Simetría**.
- Si *a*ϵC y *b*ϵC están relacionados, y a su vez, *b* y *c*ϵC están relacionados, entonces los elementos *a* y *c* están relacionados. **Transitividad**.

Ejemplo, supongamos que relacionamos los números (1, 2), (3, 2), (4, 4) y (1, 5), se cumplen entre otras cosas: 1 está relacionado con 1; 1 está relacionado con 3; 2 está relacionado con 5; 5 está relacionado con 2; etc.

Dado un elemento *x* perteneciente a C, se puede obtener la **clase de equivalencia** *[x]* que no es más que el conjunto de elementos que están relacionados con *x* directa o indirectamente.

Para el ejemplo anterior: [1] = {1, 2, 3, 5} mientras que [4] = {4}.

Definiremos la mezcla M de dos relaciones R y Q como una relación en la que se han relacionado todos los pares de elementos relacionados en R y los pares de elementos relacionados en Q. (Ver ejemplo 3)

Se desea implementar una estructura de datos en .NET que permita manejar conjuntos de este tipo.

La estructura deberá llamarse `RelacionEq` y ser genérica con un parámetro `T` que determine el tipo de elementos relacionados. Esta estructura debe implementar la interfaz `IRelacionEq<T>` la cual se describe a continuación y es provista en la biblioteca `Weboo.Utiles.dll`.

```
public interface IRelacionEq<T>
{
    /// <summary>
    /// Relaciona un nuevo par de elementos.
    /// </summary>
    /// <param name="p1">Primer elemento del par</param>
    /// <param name="p2">Segundo elemento del par</param>
    void Relaciona(T p1, T p2);

    /// <summary>
    /// Devuelve un enumerable con todos los elementos que son equivalentes a cierto elemento.
    /// </summary>
    /// <param name="s">Elemento que identifica la clase de equivalencia.</param>
    /// <returns>
    /// Un objeto IEnumerable que permite iterar por los elementos equivalentes.
    /// </returns>
    IEnumerable<T> ClaseDeEquivalencia(T s);
```

```
    /// <summary>
    /// Devuelve la cantidad de elementos del conjunto base de la relacion de equivalencia.
    /// </summary>
    int CantidadDeElementos { get; }

    /// <summary>
    /// Devuelve los elementos del conjunto base de la relacion de equivalencia.
    /// </summary>
    IEnumerable<T> Elementos { get; }

    /// <summary>
    /// Determina si dos elementos son equivalentes para la relación de equivalencia actual.
    /// </summary>
    /// <param name="e1">Elemento primero para determinar equivalencia</param>
    /// <param name="e2">Elemento segundo para determinar equivalencia</param>
    /// <returns>True si los elementos son equivalentes, Falso en otro caso.</returns>
    bool SonEquivalentes(T e1, T e2);

    /// <summary>
    /// Devuelve la mezcla entre esta relacion de equivalencia y una relacion especifica.
    /// </summary>
    /// <param name="relacion">Relacion para realizar la mezcla.</param>
    /// <returns>Una relacion nueva resultante de la mezcla.</returns>
    IRelacionEq<T> Mezcla(IRelacionEq<T> relacion);
}
```

Note:

Que la forma de adicionar elementos en el **conjunto base** de la relación es a partir del método `Relaciona`. Si se desea agregar un elemento únicamente puede hacerse `Relaciona (x, x)`. Si un par de elementos que ya están relacionados se vuelven a relacionar este método no tiene efecto (no lanza error).

Que los enumeradores `ClaseDeEquivalencia` y `Elementos` son conjuntos por lo que no pueden iterar elementos repetidos.

Los métodos `ClaseDeEquivalencia` y `SonEquivalentes` deben lanzar la excepción `ArgumentException` en caso de que algún elemento pasado por parámetro no haya sido agregado a la estructura previamente usando el método `Relaciona`.

Usted deberá entregar su clase implementada en una biblioteca de clases `Relaciones.dll`, dentro del **namespace** `Relaciones`.

Ejemplos

Ejemplo 1.

```
RelacionEq<string> r = new RelacionEq<string>();
r.Relaciona("a", "b");
r.Relaciona("b", "c");
r.Relaciona("b", "c");
r.Relaciona("b", "d");
r.Relaciona("c", "d");
r.Relaciona("h", "k");

foreach (string s in r.ClaseDeEquivalencia("a"))
    Console.Write(s+" ");
```

Imprime: a b c d

Ejemplo 2.

```
RelacionEq<int> r = new RelacionEq<int>();
r.Relaciona(3, 4);
r.Relaciona(5, 10);
r.Relaciona(1, 2);
r.Relaciona(2, 4);
r.Relaciona(5, 6);
foreach (int x in r.Elementos)
    Console.Write(x + " ");
```

Imprime: 3 4 5 10 1 2 6

```
RelacionEq<int> r1 = new RelacionEq<int>();
r1.Relaciona(1, 2);
r1.Relaciona(2, 3);
r1.Relaciona(4, 5);
r1.Relaciona(6,7);
r1.Relaciona(8,9);

RelacionEq<int> r2 = new RelacionEq<int>();
r2.Relaciona(4, 1);
r2.Relaciona(5,8);

IRelacionEq<int> r = r1.Mezcla(r2);
foreach (int x in r.ClaseDeEquivalencia(1))
    Console.Write(x + " ");
```

Imprime: 1 2 3 4 5 8 9