Campamento Militar

NOTA: SI USTED ESTÁ LEYENDO ESTE DOCUMENTO SI HABERLO DESCOMPRIMIDO, CIÉRRELO INMEDIATAMENTE Y DESCOMPRIMA EL CONTENIDO DEL ARCHIVO ZIP QUE RECIBIÓ EN EL ESCRITORIO. UNA VEZ HECHO ESTO, REANUDE SU TRABAJO DESDE ALLÍ. TRABAJAR ANTES DE DESCOMPRIMIR LE HARÁ PERDER SUS CAMBIOS, Y NO TENDRÁ OPORTUNIDAD DE RECLAMAR.

Se define la estructura de un campamento militar organizado jerárquicamente, en el que cada integrante debe acatar las órdenes de sus superiores. Se define como superior de A al jefe inmediato de A y recursivamente a los superiores del jefe de A.

Como puede fácilmente demostrarse la superioridad es transitiva, es decir si A es superior de B y B es superior de C entonces A es superior de C.

Originalmente, ningún integrante de la estructura tiene asignada tarea alguna. En dicha jerarquía se cumple que un integrante X sólo puede acatar órdenes de algún superior.

Cada integrante solo puede tener a lo sumo una orden asignada. Si se le pretende asignar una orden a alguien que no es un subordinado esto no tendrá efecto alguno. Si un superior asigna una orden a un subordinado que ya tiene una orden asignada se le cambiará de orden si esta había sido asignada por un superior de menor jerarquía, de lo contrario el intento de asignar una orden no tendrá efecto. Es decir si un soldado tenía la orden de `avanzar` asignada por un teniente superior suyo un capitán superior suyo le podrá cambiar la orden pero un sargento no.

Problema

Usted debe diseñar una estructura de datos que implemente correctamente la interface `IJerarquia<T>` que se da más abajo. El tipo genérico T indica el tipo de orden a ejecutar. Note que para las pruebas podrá ser instanciado con cualquier tipo genérico.

```
public interface IJerarquia<T>
{
    /// <summary>
    /// Establece a <see cref="jefe"/> como superior inmediato de
    /// <see cref="subordinado"/>. Cada integrante puede tener
    /// un solo jefe inmediato, y la operación es irreversible, por
    /// lo que este método debe lanzar una excepción de
    /// tipo <see cref="InvalidOperationException"/> si
    /// <see cref="subordinado"/> ya tiene asignado un jefe.
    /// La acción debe realizarse también jerárquicamente: sólo debe
    /// establecerse un jefe para alguien, si ya a dicho jefe se le
    /// asignó previamente algún jefe superior.
    /// </summary>
    void AsignaJefe(string jefe, string subordinado);
```

```
    /// <summary>
    /// Comprueba si <see cref="superior"/> es superior (inmediato o no)
    /// de <see cref="integrante"/>.  Si alguno de los elementos
    /// (<see cref="superior"/> o <see cref="integrante"/>) no existe,
    /// se debe lanzar excepción del tipo <see cref="InvalidOperationException"/>.
    /// </summary>
    bool EsSuperior(string superior, string integrante);

    /// <summary>
    /// Ordena a <see cref="integrante"/> ejecutar la <see cref="orden"/>,
    /// dictado por <see cref="superior"/>. Esto solamente tendrá efecto si
    /// <see cref="superior"/> es un superior de <see cref="integrante"/>
    /// y si la orden anterior fue dada a <see cref="integrante"/> por un
    /// superior de menor rango (o no hay orden dada).
    /// </summary>
    void Ordena(string superior, string integrante, T orden);

    /// <summary>
    /// Determina qué tarea está siendo llevada a cabo por <see cref="integrante"/>.
    /// Como parámetro de salida (<see cref="superior"/>) se devuelve además
    /// al superior que dio esa orden.
    /// Si <see cref="integrante"/> no existe, o no está cumpliendo ninguna orden,
    /// se debe lanzar una excepción de tipo <see cref="InvalidOperationException"/>.
    /// </summary>
    T Orden(string integrante, out string superior);

    /// <summary>
    /// Devuelve true si <see cref="integrante"/> tiene una
    /// orden asignada. De lo contrario devuelve false.
    /// Si <see cref="integrante"/> no existe, lanza una excepción
    /// de tipo <see cref="InvalidOperationException"/>.
    /// </summary>
    bool TieneOrden(string integrante);

    /// <summary>
    /// Devuelve true si <see cref="integrante"/> existe.
    /// De lo contrario devuelve false.
    /// </summary>
    bool Existe(string integrante);

    /// <summary>
    /// Devuelve todas las órdenes que están siendo ejecutadas,
    /// y quién está ejecutando cada una.
    /// </summary>
    IEnumerable<OrdenDada<T>> OrdenesPorCumplir();
}
```

Los elementos de la estructura jerárquica se representarán con cadenas de texto (`string`). El valor de la cadena en sí no tiene mayor significación, actúa simplemente un identificador para el elemento.

La operación `AsignaJefe(string jefe, string subordinado)` define una relación entre un jefe y su subordinado inmediato. Este método requiere que ya anteriormente `jefe` **tenga** a su vez un jefe inmediato asignado, y que `subordinado` **no tenga** ningún jefe asignado; de lo contrario debe lanzarse una excepción de tipo `InvalidOperationException`. Note que un integrante no puede tener más de un jefe inmediato pero sí más de un superior en general.

La operación `bool EsSuperior(string superior, string integrante)` determina si `superior` es un superior (inmediato o no) de `integrante`. En caso de que alguno de los dos no exista se debe lanzar una excepción de tipo `InvalidOperationException`.

La operación `Ordena(string superior, string integrante, T orden)` indica que `superior` le está mandando a `integrante` a cumplir la `orden` que tendrá efecto o no según lo explicado anteriormente.

La operación `T Orden(string integrante, out string superior)` devuelve la orden que está cumpliendo `integrante` y devuelve a través del parámetro `superior` quién fue el superior que le dio dicha orden. En caso de no existir integrante, o no estar cumpliendo ninguna orden, se debe lanzar una excepción de tipo `InvalidOperationException`.

La operación `bool TieneOrden(string integrante)` determina si `integrante` está cumpliendo alguna orden, dada anteriormente por algún superior suyo. En caso de no existir `integrante`, se debe lanzar una excepción de tipo `InvalidOperationException`.

La operación `bool Existe(string integrante)` determina si `integrante` existe, lo que es equivalente a decir que ya se le ha asignado un superior.

La operación `IEnumerable<OrdenDada<T>> OrdenesPorCumplir()` devuelve todas las órdenes que están asignadas así como el responsable de ejecutar cada una. La clase `OrdenDada<T>` representa una orden asignada a un integrante particular, y tiene al siguiente forma:

```
public class OrdenDada<T>
{
    private readonly string integrante;
    private readonly T orden;

    public OrdenDada(string integrante, T orden)
    {
        this.integrante = integrante;
        this.orden = orden;
    }

    public string Integrante
    {
        get { return integrante; }
    }

    public T Orden
    {
        get { return orden; }
    }
}
```

La propiedad `Orden` almacena la orden dada, mientras que la propiedad `Integrante` almacena quién recibió esta orden, es decir, el valor del parámetro `subordinado` en la correspondiente llamada a `Ordena(...)`.

Para su implementación, usted se basará en una clase `Jerarquía<T>`, que ya implementa la interfaz `IJerarquía<T>` y define un constructor. El parámetro de ese constructor representa al jefe supremo de la jerarquía, que es el único que no tiene superiores. Es imprescindible que usted trabaje sobre esta clase, y no modifique el constructor, para que el probador funcione correctamente.

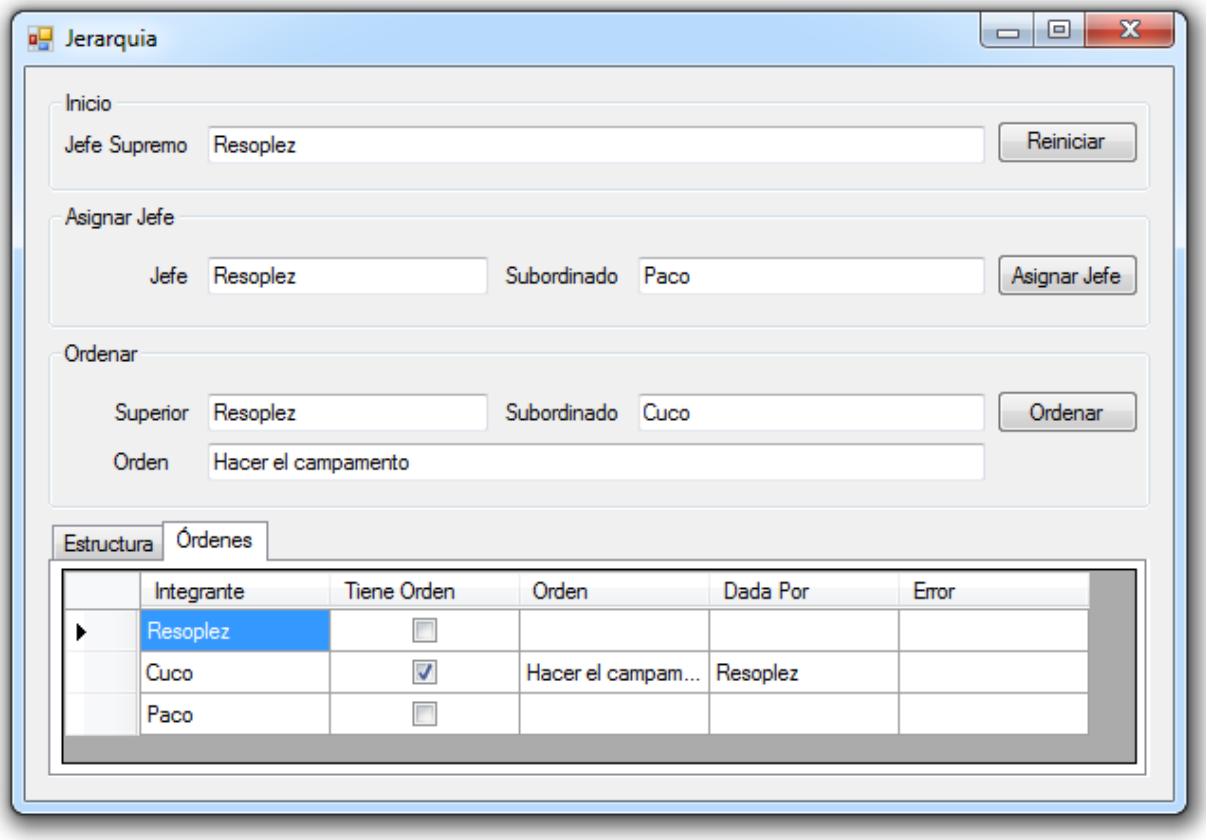
```
public class Jerarquia<T> : IJerarquia<T>
{
    private readonly string jefeSupremo;

    public Jerarquia(string jefeSupremo)
    {
        this.jefeSupremo = jefeSupremo;
    }

    public void AsignaJefe(string jefe, string subordinado)
    {
        throw new NotImplementedException();
    }

    public bool EsSuperior(string superior, string integrante)
    {
        throw new NotImplementedException();
    }

    public void Ordena(string superior, string integrante, T orden)
    {
        throw new NotImplementedException();
    }

    public T Orden(string integrante, out string superior)
    {
        throw new NotImplementedException();
    }

    public bool TieneOrden(string integrante)
    {
        throw new NotImplementedException();
    }

    public bool Existe(string integrante)
    {
        throw new NotImplementedException();
    }

    public IEnumerable<OrdenDada<T>> OrdenesPorCumplir()
    {
        throw new NotImplementedException();
    }
}
```

Solución

Como es costumbre, usted debe haber recibido un compactado con este documento y una solución de Visual Studio. En esta solución encontrará una biblioteca de clases donde debe realizar su implementación, así como una aplicación de *Windows Forms* que le servirá de probador. Esta aplicación le permite de forma visual construir su jerarquía, asignar jefes, dar órdenes y comprobar el estado de la estructura de datos.

NOTA: Esta aplicación de pruebas simplemente se limita a interactuar con **su** implementación y mostrar los resultados obtenidos del código que **usted** implementó. En ningún momento indica si el resultado obtenido es correcto o no. Es su responsabilidad comparar estos resultados con el resultado esperado (mediante una solución "a mano" de cada caso de prueba).

La aplicación tiene controles para crear una nueva jerarquía y adicionar jefes y órdenes. En la parte inferior se muestran dos componentes que visualizan en todo momento el estado de la estructura de datos. En la primera ("*Estructura*") se muestra toda la jerarquía, es decir, todos los pares Jefe-Subordinado que su estructura reporta. En la segunda ("*Órdenes*") se muestra para cada integrante, si está ejecutando una orden, y por quién fue dada. En ambos casos, si se produce una excepción, se muestra en la línea correspondiente el texto de la excepción.



Ejemplo

A continuación se muestra un ejemplo de uso de la estructura, y el resultado de cada operación realizada.

Usted tiene la responsabilidad de diseñar y resolver a mano tantos ejemplos adicionales como considere necesario para comprobar la validez de su implementación.

Acción	Estructura	Órdenes
`new Jerarquía<string>("Resoplez");` Se crea una jerarquía nueva con un jefe supremo pasado en el constructor.		**Resoplez** – *Nada*
`AsignaJefe("Resoplez", "Cuco");` Se asigna el primer jefe.	Resoplez -> Cuco	**Resoplez** – *Nada*

`Ordena("Resoplez", "Cuco",`
`"Hacer el campamento");`
 	
Se da una orden por el jefe inmediato, por lo tanto es efectiva.
	Resoplez -> Cuco	**Resoplez** – *Nada*
		Cuco – *Hacer el campamento* (Resoplez)

`AsignaJefe("Resoplez", "Paco");` Se asigna un nuevo jefe, aparece un nuevo subordinado.	Resoplez -> Cuco Resoplez -> Paco	**Resoplez** – *Nada* **Cuco** – *Hacer el campamento* (Resoplez) **Paco** – Nada

`Ordena("Cuco", "Paco",`
`"Ayudar");`
 	
La orden no proviene de un superior, así que no tiene efecto.
	Resoplez -> Cuco
	Resoplez -> Paco
		Resoplez – *Nada*
		Cuco – *Hacer el campamento* (Resoplez)
		Paco – Nada

`AsignaJefe("Paco", "Pepe");` Se asigna un nuevo jefe. Note que como superior aparecen además los superiores de dicho jefe.	Resoplez -> Cuco Resoplez -> Paco Paco -> Pepe Resoplez -> Pepe	**Resoplez** – *Nada* **Cuco** – *Hacer el campamento* (Resoplez) **Paco** – Nada **Pepe** – Nada

`Ordena("Paco", "Pepe",`
`"Ensillar las yeguas");`
 	
La orden proviene de un superior, así que se acata sin problemas.
	Resoplez -> Cuco
	Resoplez -> Paco
	Paco -> Pepe
	Resoplez -> Pepe
		Resoplez – *Nada*
		Cuco – *Hacer el campamento* (Resoplez)
		Paco – Nada
		Pepe – *Ensillar las yeguas* (Paco)

`Ordena("Cuco", "Pepe",`
`"Hacer café");`
 	
La orden no proviene de un superior, por lo que nada sucede.
	Resoplez -> Cuco
	Resoplez -> Paco
	Paco -> Pepe
	Resoplez -> Pepe
		Resoplez – *Nada*
		Cuco – *Hacer el campamento* (Resoplez)
		Paco – Nada
		Pepe – *Ensillar las yeguas* (Paco)

`Ordena("Resoplez", "Pepe",`
`"Ayudar a Cuco");`
 	
La orden proviene de un superior con mayor rango que quién dio la orden anterior, por lo que tanto, se acata.
	Resoplez -> Cuco
	Resoplez -> Paco
	Paco -> Pepe
	Resoplez -> Pepe
		Resoplez – *Nada*
		Cuco – *Hacer el campamento* (Resoplez)
		Paco – Nada
		Pepe – *Hacer café* (Resoplez)

`Ordena("Paco", "Pepe",`
`"Volver");`
 	
La orden proviene de un superior, pero tiene menor rango que quién dio la orden anterior, por lo que se ignora.
	Resoplez -> Cuco
	Resoplez -> Paco
	Paco -> Pepe
	Resoplez -> Pepe
		Resoplez – *Nada*
		Cuco – *Hacer el campamento* (Resoplez)
		Paco – Nada
		Pepe – *Hacer café* (Resoplez)