Enumerador de árboles

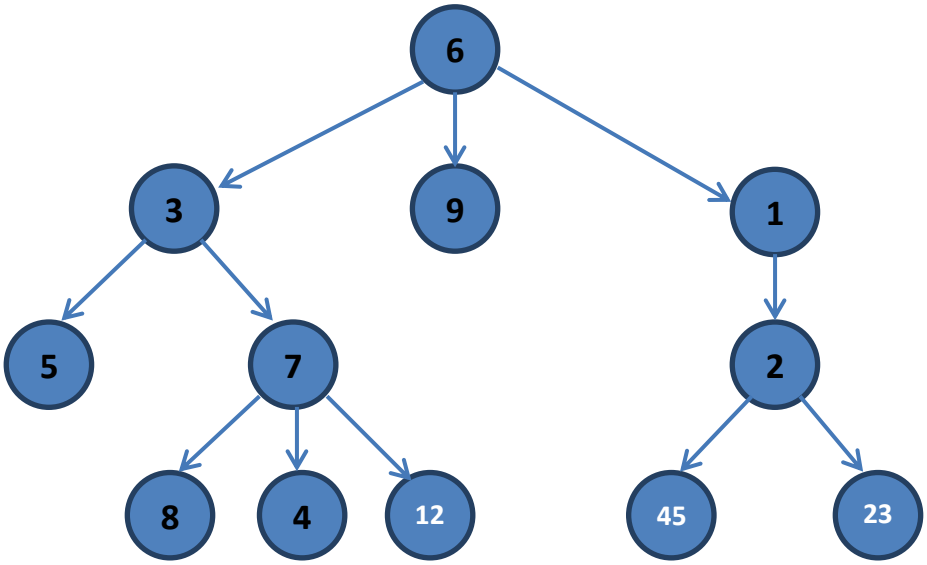
Implemente una clase **ArbolLevelEnumerator<T>** que implemente la interfaz **ILevelEnumerator<T>** que a su vez implementa la interfaz **IEnumerator<T>**. Esta clase deberá permitir enumerar los elementos de un **Arbol<T>**, por niveles, y permitirá "saltar" al siguiente nivel, en cualquier punto de la iteración, al invocar el método **JumpNextLevel**, para interrumpir la iteración del nivel actual y retomarla en el próximo nivel.

Tanto la clase **Arbol<T>** como la interfaz **ILevelEnumerator<T>** se incluyen en el ensamblado **Extraordinario.dll**. Usted deberá implementar la clase **ArbolLevelEnumerator<T>** que se muestra a continuación:

```
public class ArbolLevelEnumerator<T> : ILevelEnumerator<T>
{
    public ArbolLevelEnumerator(Arbol<T> arbol) {  }

    public T Current { get {  } }

    object System.Collections.IEnumerator.Current { get {  } }

    public bool MoveNext() { }

    public void Reset() { }

    public bool JumpNextLevel() { }

    public void Dispose() { }
}
```

El constructor de la clase **ArbolLevelEnumerator<T>** recibe como parámetro el árbol sobre el cual deberá iterar. Se garantizará que dicho árbol no sea null. El comportamiento dado por la interfaz **IEnumerator<T>** debe permitir enumerar los elementos del árbol por niveles. La implementación del método **JumpNextLevel** debe saltar al siguiente nivel. Si se invoca a **JumpNextLevel** y este devuelve **true**, el elemento que devuelva **Current**, debe ser el valor del primer nodo del proximo nivel del árbol. De manera análoga al **MoveNext** este método devuelve **true** si es posible saltar y **false** en caso contrario.

Por ejemplo, para el siguiente árbol:



El siguiente código:

```
Arbol<int> arbol = GenerarArbol(); // Crear árbol con los valores del ejemplo.

ArbolLevelEnumerator<int> levelEnumerator = new ArbolLevelEnumerator<int>(arbol);

while (levelEnumerator.MoveNext())
{
    Console.WriteLine(levelEnumerator.Current);
}
```

Debería imprimir la secuencia 6, 3, 9, 1, 5, 7, 2, 8, 4, 12, 45, 23

Por su parte, el siguiente código:

```
  while (levelEnumerator.MoveNext())
  {
      Console.WriteLine(levelEnumerator.Current);

      if (current == 7)
      {
          levelEnumerator.JumpNextLevel();
          break;
      }
  }

  Console.WriteLine(levelEnumerator.Current);
```

Debería imprimir la secuencia 6, 3, 9, 1, 5, 7, 8

Véase que se salta del 7 al 8, obviando el 2, pues se llamo a **JumpNextLevel** estando en 7 y 8 es el primer valor del siguiente nivel.

Tenga en cuenta que si se está enumerando el último nivel y se invoca el método **JumpNextLevel**, la iteración debe concluir y el método debe devolver **false**. Preguntar por la propiedad **Current**, luego que **JumpNextLevel** retornó **false**, debe lanzar **InvalidOperationException**.

Por ejemplo:

```
while (levelEnumerator.MoveNext())
{
    Console.WriteLine(levelEnumerator.Current);

    if (levelEnumerator.Current == 12)
        levelEnumerator.JumpNextLevel();

    Console.WriteLine(levelEnumerator.Current); // Lanza excepción
}
```

El código anterior, debería lanzar excepción al tratar de imprimir **levelEnumerator.Current** pues **levelEnumerator.JumpNextLevel()** debió devolver **false** y terminar la iteración.

Igualmente, si se invoca el método **JumpNextLevel** antes de haber invocado el método **MoveNext**, deberá lanzarse una excepción del tipo **InvalidOperationException**. Por otra parte, invocar el método **JumpNextLevel** después que **MoveNext** devolvió **false**, también deberá retornar **false**, y viceversa.

Si estando en el último elemento de la iteración, en el caso del ejemplo el valor 23, invocar el método **JumpNextLevel** tendría el mismo efecto que invocar **MoveNext**. En este caso ambos deberían retornar **false**.

El método **Reset** deberá devolver al iterador a su estado inicial, o sea, como si nunca se hubiera invocado a **MoveNext**.

La clase **Arbol<T>** tendra los siguientes miembros:

```
public class Arbol<T>
    {
        public Arbol(T valor);

        public Arbol(T valor, IEnumerable<Arbol<T>> hijos);

        public List<Arbol<T>> Hijos { get; }

        public T Valor { get; }

        public void AgregarHijo(Arbol<T> nuevoHijo);
    }
```

Se garantiza que el campo *Hijos* estará inicializado despues de la invocación de cualquiera de los construtores de la clase, o sea, *Hijos* nunca será null una vez creada una instancia de `Arbol<T>`.

Nota: Usted no tendrá que darle código al método Dispose. Debe dejarlo vacio tal como se le entregó en la plantilla de solución.