ACLARACIÓN PRÁCTICA 4 : Implementación del método isOrdered()

En el método isOrdered() se deben comparar los elementos de la lista.
El interface **Comparable** tiene un método que sirve para comparar elementos: **compareTo(obj)**.

En el documento "Ejercicios de listas con recursividad":
https://agora.unileon.es/pluginfile.php/275333/mod_resource/content/0/EjerciciosRecursivosListas.pdf
hay un ejercicio en el que tiene que utilizar el método **compareTo**, pero no ha indicado en el T que es comparable. **La solución es hacer un cast a Comparable<T>**

```
if ( ((Comparable<T>) aux).compareTo(elem) >0){
```

Es decir, en la clase **AbstractLinkedListImpl<T>**, como en la declaración de la clase no ha indicado que el T tiene que ser comparable, es necesario hacer una conversión del elemento que está en el nodo, a comparable, para poder compararlo. Esta conversión fallaría en tiempo de ejecución si la clase que sustituye al T no es comparable, porque no podrá ejecutar el método compareTo.

Sin embargo, en la implementación de las operaciones de la clase de lista ordenada (OrderedLinkedListImpl) como ya se ha indicado que el T es un tipo comparable (debe ser una clase que implementa el interface comparable o tener en su jerarquía alguna clase que lo implemente) se puede utilizar directamente el método compareTo sin tener que castear el elemento.

```
public class OrderedLinkedListImpl<T extends Comparable<? super T>>
```

No ponemos isOrdered() en la clase de lista ordenada, ya que entonces (si hubiésemos implementado correctamente el método add) siempre devolvería true. En la clase AbstractLinkedListImpl<T> y en la no ordenada, puede devolver true o false dependiendo del contenido de la lista.

En el **método isOrdered()** si el cast falla porque el tipo utilizado en tiempo de ejecución en lugar de T, no implementa el interface Comparable, disparará la excepción **java.lang.ClassCastException**.

En el caso de que no implemente Comparable no hay forma de comparación y entonces dispara la excepción.

TODO: El método isOrdered **debería capturar esa excepción** **java.lang.ClassCastException** mediante un bloque try/catch y **disparar una excepción propia de la colección** que indique la razón del problema ("El tipo de los elementos de la colección no es comparable").

Es decir:
- Si dispara esta excepción java.lang.ClassCastException, será porque el T no es comparable. En ese caso, e**l método isOrdered() debe capturar esta excepción y disparar una excepción propia** (que hay que crear en el paquete ule.edi.exceptions) llamada **TypeIsNotComparableException** que heredará de **RunTimeException** (por tanto no hay que modificar el interface ya que no habría que avisar en la signatura del método que la puede disparar).

- Para **crear la nueva excepción TypeIsNotComparableException**, seleccionamos el paquete donde la queremos crear (ule.edi.exceptions) -> Botón derecho -> New Class -> Se abre un cuadro de diálogo:
  - Name: TypeIsNotComparableException
  - SuperClass: RuntimeException (en botón Browse la seleccionamos)

Se crea la clase para la excepción. El nombre de la clase aparece subrayado en amarillo. Si ponemos el cursor encima del nombre, sale un menú desplegable donde podemos seleccionar "Add generated serial version ID" para que genere un número de serie para la excepción.

Al igual que en la excepción EmptyColectionException, en el constructor llamaremos al constructor de la clase padre, que tiene un parámetro, que será el mensaje que se verá cuando dispare la excepción.

```
public TypeÏsNotComparableException() {
        super("El tipo de los elementos no es comparable.");
    }
```