Práctica 2: Árboles genéricos

Ejercicio 1: Aumentar la funcionalidad de la clase `LinkedTree` implementando el método:

```
/** Moves a node and its corresponding subtree (rooted at pOrig) to make
it as a new children of pDest */
public Position<E> moveSubtree(Position<E> pOrig, Position<E> pDest)
                                    throws RuntimeException;
```

Además, se deberán definir pruebas unitarias (implementadas con JUnit) para asegurar su correcto funcionamiento.

Ejercicio 2: Implementar un árbol de tipo *Left-Child Right-Sibling* (LCRS) como estrategia alternativa de diseño de árboles genéricos para tener una funcionalidad equivalente a `LinkedTree`. Para que el ejercicio se considere válido las dos clases deben tener los mismos métodos y el mismo comportamiento. Es obligatorio utilizar el prototipo de clase disponible en el Aula Virtual.

Ejercicio 3: Incorporar los test unitarios facilitados en el Aula Virtual y aumentar su funcionalidad para asegurar el correcto funcionamiento de la nueva estrategia LCRS implementada. Se valorará la calidad y cobertura de los test implementados.

Ejercicio 4: Implementar los iteradores `FrontIterator`, `PreorderIterator` y `PostorderIterator`, cuyo prototipo está disponible en el Aula Virtual. El ejercicio se considerará que está correctamente implementado si SÓLO se utilizan los métodos de la interfaz `Tree` (de tal forma que puedan ser utilizados tanto para objetos de la clase `LinkedTree` como de `LCRSTree`). Desarrollar test unitarios para dichos iteradores que comprueben su correcto funcionamiento.

Ejercicio 5: Se desea desarrollar un programa en Java que permita gestionar los árboles genealógicos de las familias que aparecen en la serie **Juego de Tronos**. En concreto, se pide crear una clase denominada `GameOfThrones` con la siguiente funcionalidad:

1. Cargar de un fichero de texto los árboles genealógicos de la familia en el formato descrito al final de este documento. El prototipo de este método será:

   ```
   public void loadFile(String pathToFile)
   ```

2. Consultar los integrantes de una familia. Recibe un apellido de la familia a visualizar y devuelve el árbol genealógico correspondiente. El prototipo de este método será:

   ```
   public LinkedTree<FamilyMember> getFamily(String surname)
   ```

3. Consultar el heredero de una famila. Para ello, se proporciona un apellido y devuelve el primer hijo varón. En caso de que no exista, se devolvería la mayor de las hijas o el primer nieto barón (y así sucesivamente).

```
public String findHeir(String surname)
```

4. Puesto que se puede descubrir que una persona tiene el apellido que no le corresponde (por ser un descendiente ilegítimo), se debe poder cambiar de familia (por simplicidad, se mantiene el apellido original). Para ello, se debe solicitar al usuario el nombre del personaje correspondiente y a qué familia cambia, indicando el nuevo ascendiente. El cambio afectará a todos sus descendientes.

   ```
   public void changeFamily(String memberName, String newParent)
   ```

5. Consultar parentesco. Determina si dos personajes pertencen a la misma famila. Para ello se solicita al usuario los nombres de dos personajes y comprueba si ambos son parientes o no.

   ```
   public boolean areFamily(String name1, String name2)
   ```

FORMATO DEL FICHERO DE TEXTO

```
ID1 = nombre1 apellidos2 (género1)edad1
ID2 = nombre2 apellidos2 (género2)edad2
.
.
.
IDN = nombreN apellidosN (géneroN)edadN
-----------------------------------------------------------------------
m
raíz1
raíz2
.
.
.
raízm
ID_Padre -> ID_Hijo
ID_Padre -> ID_Hijo
.
.
.
```

Donde:

- `IDX` será un identificador único asociado a cada miembro.
- `nombreX apellidosX` serán los nombres y apellidos de cada miembro asociados al identificador.
- `géneroX` podrá ser `M` si es hombre (Male) o `F` si es mujer (Female).
- `edadX` indicará la edad del miembro asociado al identificador.
- `m` será el número de familias que se van a cargar del fichero.
- `raíz1` … `raízM` serán los cabez de cada familia, que funcionarán como raíz del árbol familiar.
- `ID_Padre -> ID_Hijo` indica una relación de parentesco donde el miembro identificado por `ID_Padre` es padre del miembro identificado por `ID_Hijo`.