Sesión 5. Árboles ordenados

Crea un proyecto Java de nombre `sesión-05`, descarga el material que se proporciona para la práctica y copia los archivos a los *packages app* y *estDatos*.

Material proporcionado para la práctica:

1. El archivo *Tree.java* que proporciona el TDA *Tree<E>* para árboles ordenados.
2. Un programa cliente *App.java*.

Se pide:

1. Proporcionar el tipo de dato mutable *TreeList<E>* de árboles ordenados. La información que se mantendrá para un árbol (su representación) ha de ser la siguiente:
   a. La etiqueta de la raíz del árbol: *labelRoot*
   b. La secuencia de subárboles (*List<Tree<E>>*) hijos de la raíz del árbol: *children*

   El tipo de dato deberá incluir los siguientes constructores:

public TreeList(E e);	// crea un árbol que sólo tiene raíz de etiqueta *e*
public TreeList(E e, Tree<E>…trees);	// crea un árbol cuya raíz tiene la etiqueta *e* y cuyos nodos // hijos son las raíces de los árboles dados
public TreeList(E e, List<Tree<E>> children);	// crea un árbol cuya raíz tiene la etiqueta *e* y cuyos nodos // hijos son las raíces de los árboles de la lista dada
public TreeList(Tree<E> t);	// constructor de conversión

2. Utilizar en el programa cliente las funciones de recorrido de árboles que se proporcionan en la interfaz *Tree<E>* para mostrar en consola todos los recorridos del árbol *ta* dado, indicando, en cada caso, el recorrido que se muestra.
3. Utiliza la función *preorder* dada en el cliente para definir la función *replace(t, label, change)*. Esta función cambia todas las etiquetas de los nodos del árbol *t* que coincidan con *label* a *change*. Comprueba que se han realizado bien los cambios mostrando de nuevo los recorridos.

   Nota: Obsérvese que el iterador interno *preorder* dado en el cliente trabaja con subárboles o con los nodos del árbol que se itera (que son las raíces de estos subárboles). En cambio, el iterador interno *preorder* definido en la interfaz *Tree<E>* trabaja con las etiquetas de los nodos del árbol.

4. (Complementario)Modifica el tipo de dato *TreeList<E>* para que incluya la operación *parent()*. Esta operación deberá retornar, en tiempo constante, el árbol cuya raíz es padre de la raíz del árbol.