LABORATORIO 3

Recorridos iterativos en árboles binarios

Jordi Blanco Lopez

Jesus Fernandez Cid De Rivera

Curso: 2016/2017

Informe

En este proyecto se nos pedía que implementáramos los tres recorridos fundamentales sobre árboles binarios (pre-, in- y post-orden) pero en sus versiones iterativas, es decir, sin usar recursividad. La clave de ello consiste en usar una pila, de manera semejante a lo que hace la máquina virtual de Java cuando ejecuta la versión recursiva.

Primera parte: implementación de los árboles binarios

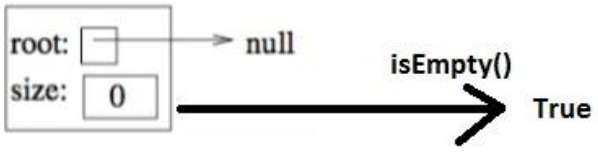
Como solamente estamos interesados en resolver el problema de los recorridos, implementamos, usando una estructura de nodos y referencias, una versión mínima de los árboles.

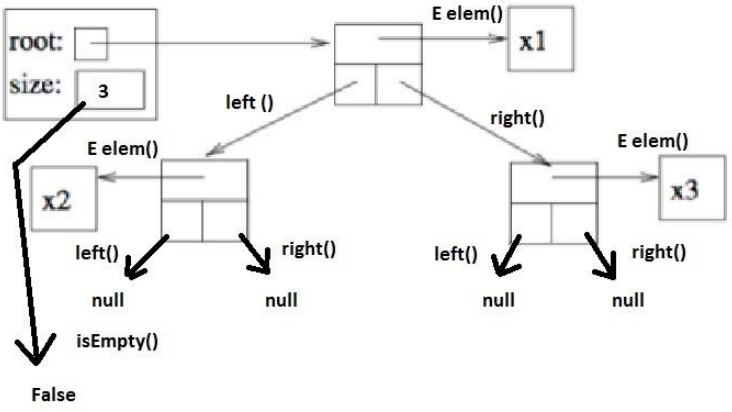
Interfaz BinaryTree<E>

Esta será la interfaz que declarará las operaciones que podremos realizar sobre los árboles binarios:

Las operaciones de la interfaz se comportan de la siguiente manera:

- elem(), left(), right() devuelven el valor almacenado en el nodo raíz del árbol, el subárbol izquierdo y el subárbol derecho, respectivamente. Todas ellas lanzan NoSuchElementException en caso de ser aplicadas sobre un árbol vacío.
- isEmpty() devuelve true si el árbol está vacío y false en caso contrario





También añadimos una función extra la cual nos facilitara el trabajo. Dicha función tiene el nombre de giveRoot() la cual transforma un árbol LinkedBinaryTree en un árbol Node.

Clase LinkedBinaryTree<E>

Esta será la clase que implementará la interfaz anterior (cumpliendo las especificaciones de las operaciones), usando nodos enlazados. Obviamente, además de las operaciones de la interfaz, tendrá diferentes constructores, para crear diferentes tipos de árboles:

- public LinkedBinaryTree()
- public LinkedBinaryTree(E elem)
- public LinkedBinaryTree(E elem, LinkedBinaryTree<E> left, LinkedBinaryTree<E> right)

Respectivamente, crean un árbol vacío, uno consistente en una hoja que contiene elem y, finalmente, un árbol cuya raíz contiene el elemento elem y cuyos árboles izquierdo y derecho son left y right.

- Tambien hemos añadido un constructor privado para construir un árbol a partir de un nodo, es decir:
    - private LinkedBinaryTree(Node<E> root)

Y hemos implementado la función extra declarada en la interfaz anterior, giveRoot().

Interfaz Traversals

Esta interfaz declarará los métodos que implementarán cada uno de los recorridos. Se trata de una interfaz no genérica que contiene tres métodos que sí lo son.

```
public interface Traversals {
    <E> List<E> preOrder(BinaryTree<E> tree);
    <E> List<E> inOrder(BinaryTree<E> tree);
    <E> List<E> postOrder(BinaryTree<E> tree);
}
```

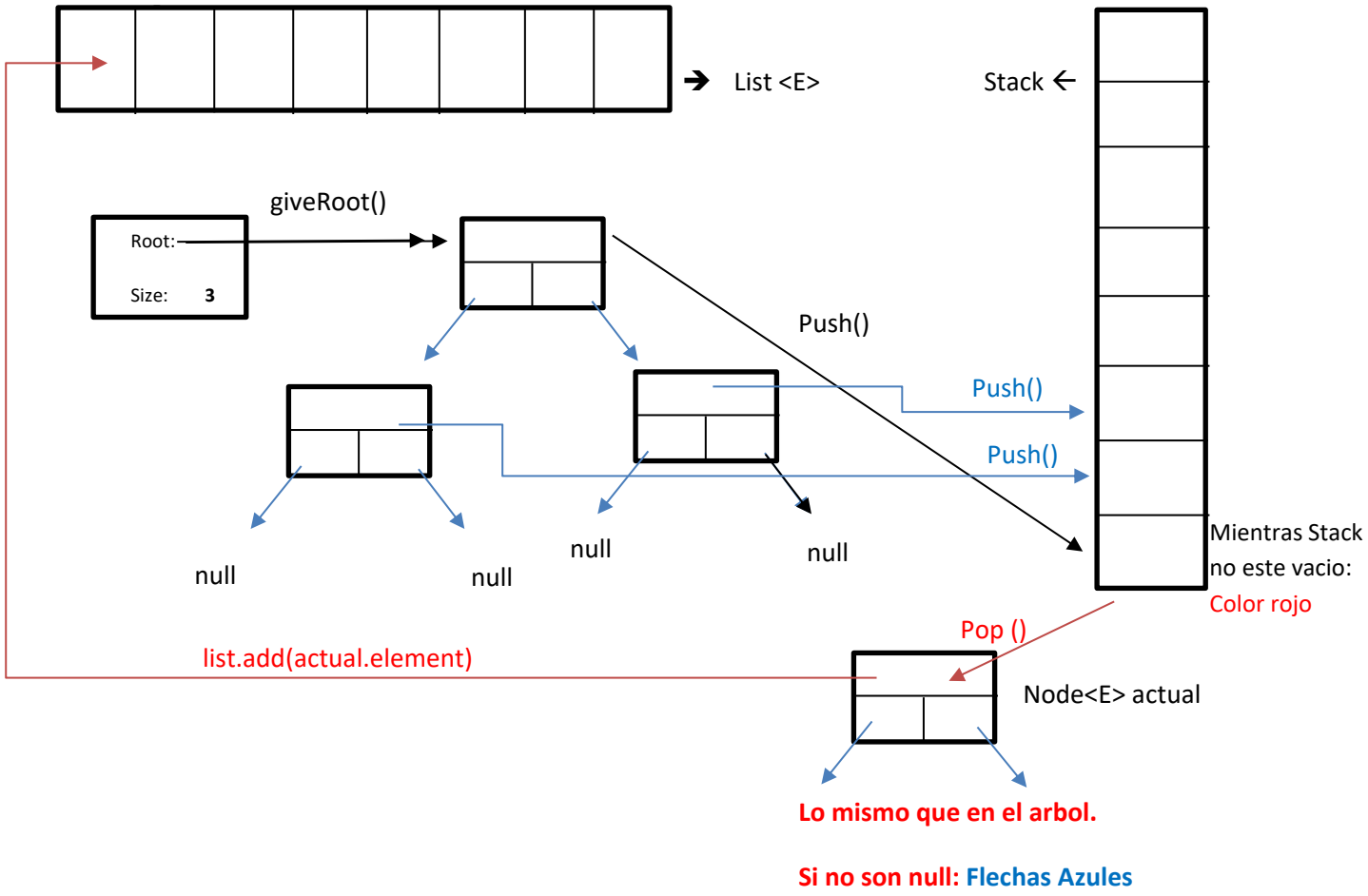
Pregunta: **¿Qué diferencias provocaría que la interfaz fuera genérica y los métodos no?**

Que por muchos tipos de Objetos que puedan entrar en la interfaz no podrán emplear dichos métodos a no ser que sea el objeto especificado para el método en sí.

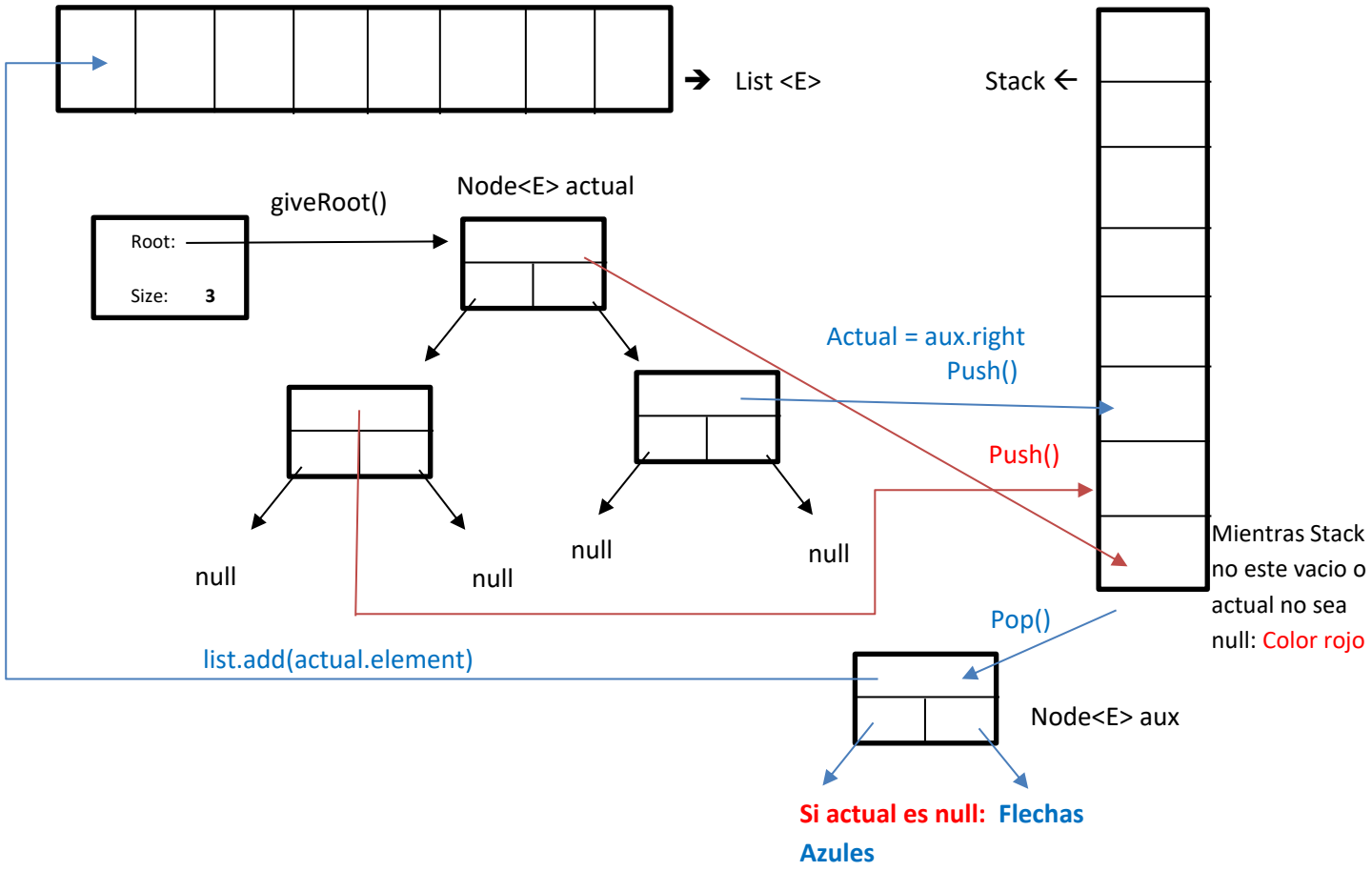
Clase IterativeTraversals

Esta es la clase que contendrá la implementación iterativa de los tres recorridos:

- **public <E> List<E> preOrder(BinaryTree<E> tree)**



- **public <E> List<E> InOrder(BinaryTree<E> tree)**



- **public <E> List<E> postOrder(BinaryTree<E> tree)**