**Estructuras de Datos**
**Grado en Ingeniería Informática en Sistemas de Información**

# Enseñanzas Prácticas y de Desarrollo
# EPD-6: Árboles Binarios

## Objetivos

- Conocer el concepto de árbol.
- Implementar y utilizar árboles.

## Conceptos

### *1. Concepto de Árbol*

Un árbol es una estructura de datos no lineal, dinámica, jerárquica y homogénea que emula la forma de un árbol mediante el uso de nodos conectados entre sí, siendo éstos las unidades elementales de dicha estructura.

Un nodo es la unidad sobre la que se construye el árbol y puede tener cero o más nodos hijos conectados a él. Se dice que un nodo ***A*** es padre de un nodo ***B*** si existe un enlace directo desde ***A*** hasta ***B***. Al mismo tiempo, podemos decir que ***B*** es hijo de ***A***. Sólo puede haber un único nodo sin padres, que llamaremos raíz. Un nodo que no tiene hijos se conoce como hoja o externo. Otros términos relacionados con los árboles son: antecesor, sucesor, grados de un nodo, nivel de un nodo, orden, altura y anchura.

La ADT del árbol se compone de los siguientes métodos:

- *size*: devuelve el número de nodos en el árbol.
- *isEmpty*: indica si el árbol está o no vacío.
- *elements:* devuelve un iterador con todos los elementos del árbol.
- *nodes:* devuelve un iterador con todos los nodos del árbol.
- *root:* devuelve el nodo raíz del árbol.
- *parent:* .devuelve el nodo padre de un nodo dado.
- *children:* devuelve una colección iterable con los nodos hijos de un nodo dado.
- *isInternal:* devuelve si el nodo dado es interno.
- *isExternal:* devuelve si el nodo dado es externo.
- *isRoot:* devuelve si el nodo dado es el nodo raíz del árbol.

### *2. Árboles binarios*

Un árbol binario es una estructura de datos en la cual cada nodo tiene como máximo dos nodos hijos. Típicamente los nodos hijos son llamados izquierdo y derecho. La ADT de un árbol binario incluye los siguientes métodos:

- *left:* devuelve el nodo hijo izquierda de un nodo dado.
- *right:* devuelve el nodo hijo derecha de un nodo dado.
- *hasLeft:* si el nodo dado tiene o no nodo hijo izquierdo.
- *hasRight:* si el nodo dado tiene o no nodo hijo derecho.
- *elementsInOrder:* devuelve un iterador con todos los elementos del árbol recorrido en inorden.
- *elementsPreOrder:* devuelve un iterador con todos los elementos del árbol recorrido en preorden.
- *elementsPostOrder:* devuelve un iterador con todos los elementos del árbol recorrido en postorden.
- *nodesInOrder:* devuelve un iterador con todos los nodos del árbol recorrido en inorden.
- *nodesPreOrder:* devuelve un iterador con todos los nodos del árbol recorrido en preorden.
- *nodesPostOrder:* devuelve un iterador con todos los nodos del árbol recorrido en postorden.

Como con cualquier tipo de estructuras de datos necesitamos contar con algoritmos para recorrer su contenido. Existen una serie de recorridos que llevaremos a cabo frecuentemente y son aquellos que pasan por todos los nodos de un árbol. Este tipo de recorridos se pueden clasificar en: recorridos en profundidad y recorridos en anchura o por niveles. En cuanto a los recorridos en profundidad, podemos distinguir:

1. *Recorrido en **preorden***: consiste en visitar el nodo actual y después visitar el subárbol izquierdo y después el subárbol derecho.

2. *Recorrido en **inorden***: primero se visita el subárbol izquierdo, después el nodo actual y seguidamente el subárbol derecho.
3. R*ecorrido en **postorden***: se visita en primer lugar el subárbol izquierdo, después el subárbol derecho y finalmente el nodo actual.

## Bibliografía Básica

Documentación de la API de Java:

- *Introduction to Algorithms*. Thomas H. Cormen. Capítulo 12, Página 253.

## Experimentos

**E1**. Analice el siguiente fragmento de código correspondiente a un árbol binario:

```
public class BinaryNode {
	Object element;
	BinaryNode left;
	BinaryNode right;
	public BinaryNode(Object ele) {
		this.element = ele;
		this.left = null;
		this.right = null;
	}
	public BinaryNode(Object ele, BinaryNode l, BinaryNode r) {
		this.element = ele;
		this.left = l;
		this.right = r;
	}
}
```

¿Por qué el tipo de atributo element es `Object`? ¿Qué representan los campos `left` y `right`?

**E2**. Analice la siguiente interfaz correspondiente a un árbol.

```
public interface BinaryTree {
	int size();
	boolean isEmpty();
	Iterator elementsInOrder();
	Iterator elementsPreOrder();
	Iterator elementsPostOrder();
	Iterator nodesInOrder();
	Iterator nodesPreOrder();
	Iterator nodesPostOrder();
	BinaryNode root();
	BinaryNode parent(BinaryNode node);
	Collection children(BinaryNode node);
	boolean isInternal(BinaryNode node);
	boolean isExternal(BinaryNode node);
	boolean isRoot(BinaryNode node);
	BinaryNode left(BinaryNode node);
	BinaryNode right(BinaryNode node);
	boolean hasLeft(BinaryNode node);
	boolean hasRight(BinaryNode node);
}
```

**E3**. Analice el siguiente fragmento de código:

```
public class generateRootTree() {
       public static void main(String[] args) throws Exception {
              BinaryNode root  = new BinaryNode("A");
              BinaryNode node1 = new BinaryNode("B1");
              BinaryNode node2 = new BinaryNode("B2");
              BinaryNode node3 = new BinaryNode("C11");
              BinaryNode node4 = new BinaryNode("C21");
              BinaryNode node5 = new BinaryNode("C22");

              root.left = node1;
              root.right = node2;
              node1.left = node3;
              node2.left = node4;
              node2.right = node5;
       }
}
```

¿Qué hace? Represente cómo se disponen los nodos.

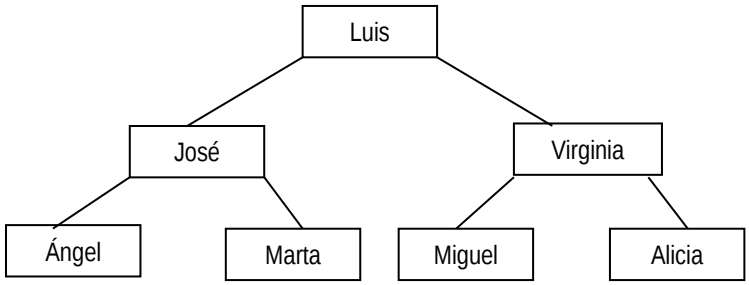
## Ejercicios

**EJ1**. (90 mins.) Implementar un árbol binario que implemente la interfaz ***BinaryTree***. Para implementar los métodos que devuelven un iterador puede, por simplicidad, añadir todos los elementos requeridos a una colección y luego devolver el iterador sobre esa colección.

## Problemas

**P1. (90 mins.)** Extender el ejercicio 1 para que se puedan insertar nodos en el árbol. Para esto se añadirán dos métodos ***insertLeft*** y ***insertRight*** que recibirán dos parámetros, un nodo del árbol ***N*** y el elemento que se insertará como hijo izquierdo/derecho del nodo ***N***.

**P2. (25 mins)** Realice una función que busque una persona por su nombre en el siguiente árbol genealógico. En caso de lo encuentre, devolver la dirección de dicho nodo.



**P3. (30 mins)** Implemente un procedimiento ***BorrarSubArb*** que dados un árbol binario de búsqueda de enteros ***A*** sin elementos repetidos y un entero ***x***, elimine el subárbol de ***A*** que tiene a ***x*** como su raíz. Si ***x*** no pertenece a ***A*** el procedimiento no tendrá efecto.
Por ejemplo, dado el siguiente árbol y ***x=7*** el resultado ha de ser: