

# PRÁCTICO 2 - ÁRBOLES

Distintas formas de implementar árboles binarios.

# Árboles con nodos O sin nodos

- Un árbol se puede pensar como un **conjunto de nodos conectados entre sí.**

En este caso, el árbol tiene un puntero al nodo “raíz” (similar a la Lista que tenía un puntero al “first”). En lugar de tener tan solo un “next”, los nodos del árbol tienen un “hijoIzquierda” y un “hijoDerecha”

- Un árbol se puede pensar de manera recursiva, donde cada árbol **posee un subárbol izquierdo y un subárbol derecho.**

En este caso no existen nodos en nuestra implementación del árbol. Cada árbol almacena un valor, un puntero a su subárbol izquierdo y a su subárbol derecho. Los subárboles son simplemente otros árboles.

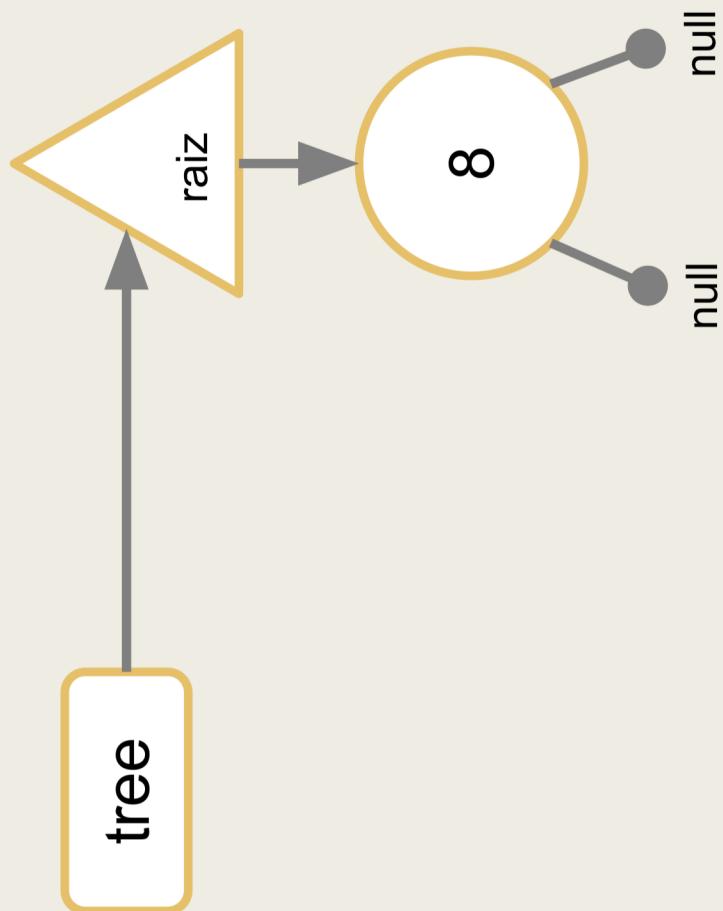
# Árbol con nodos

```
TreeWithNode tree = new TreeWithNode();
```

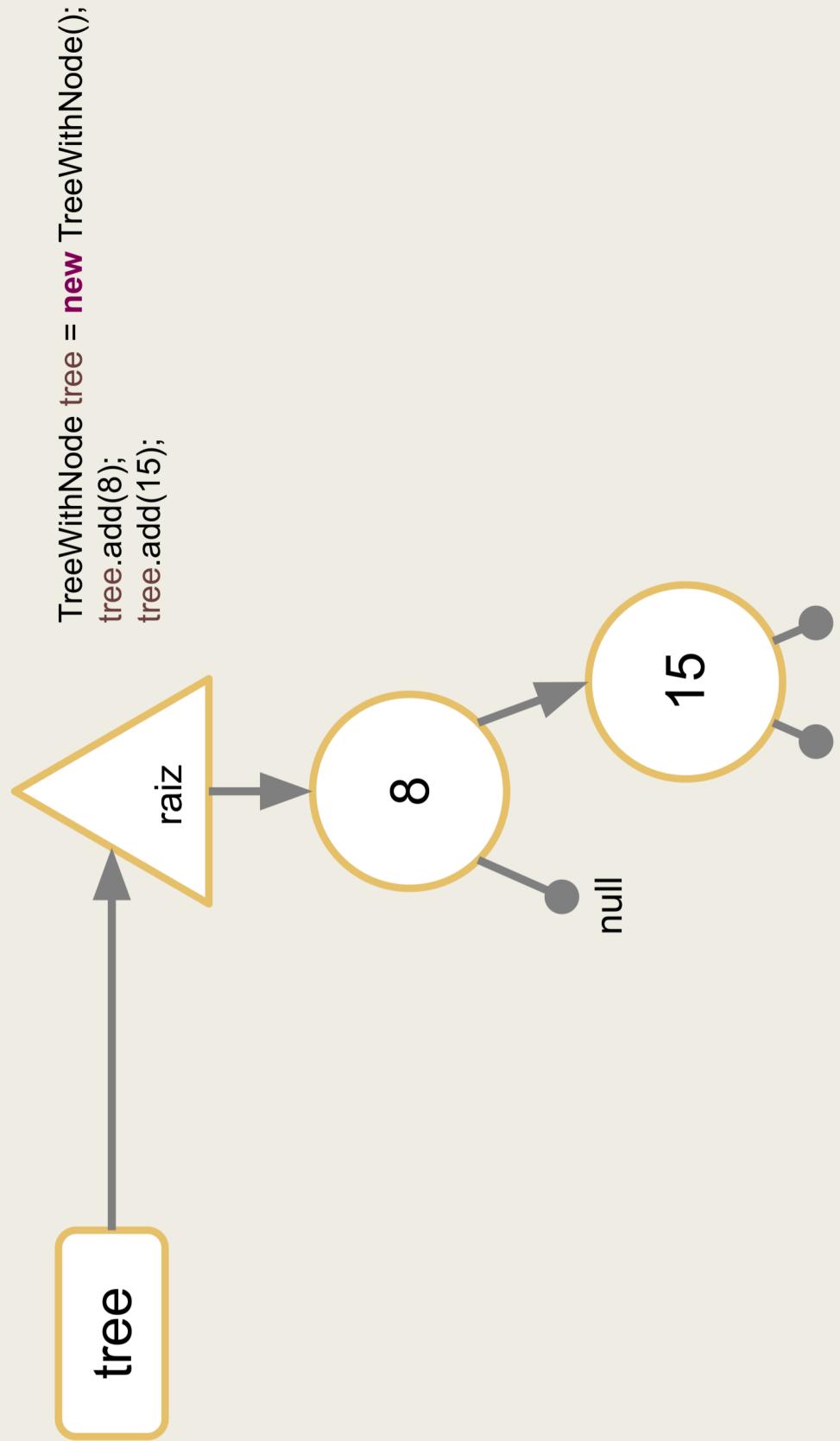


# Árbol con nodos

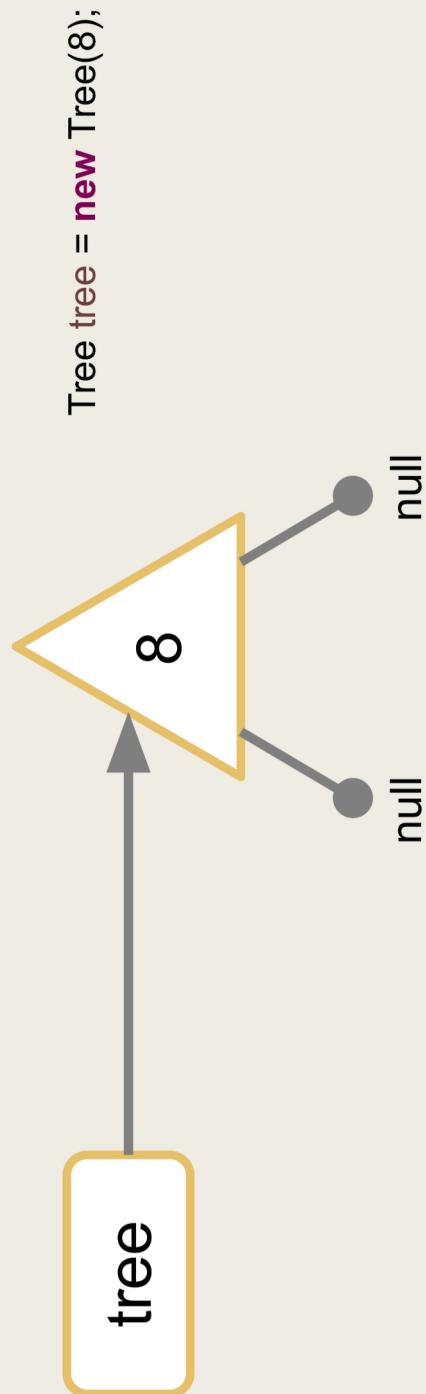
```
TreeWithNode tree = new TreeWithNode();  
tree.add(8);
```



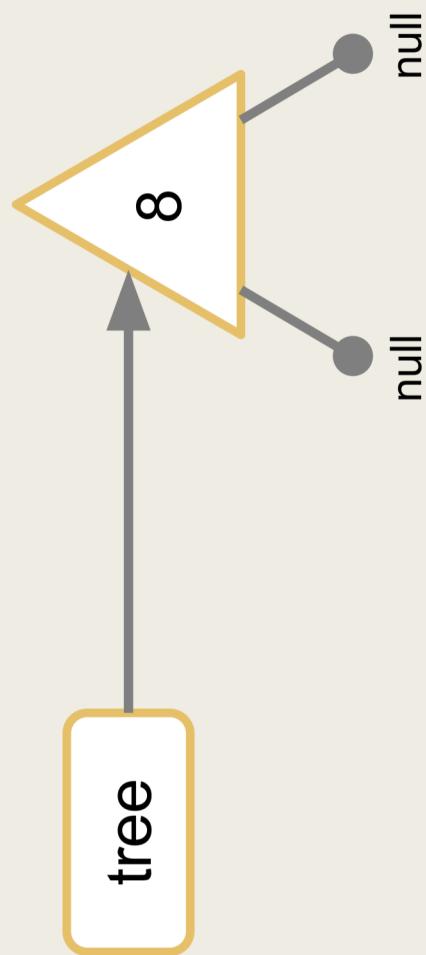
# Árbol con nodos



# Árbol con subárboles

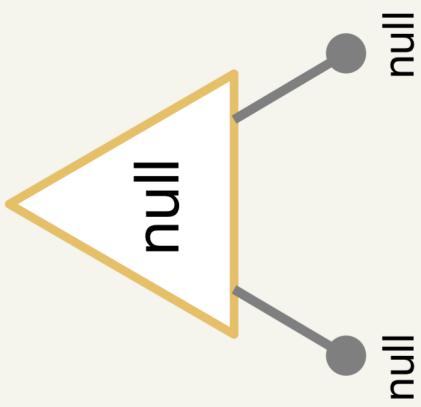


# Árbol con subárboles

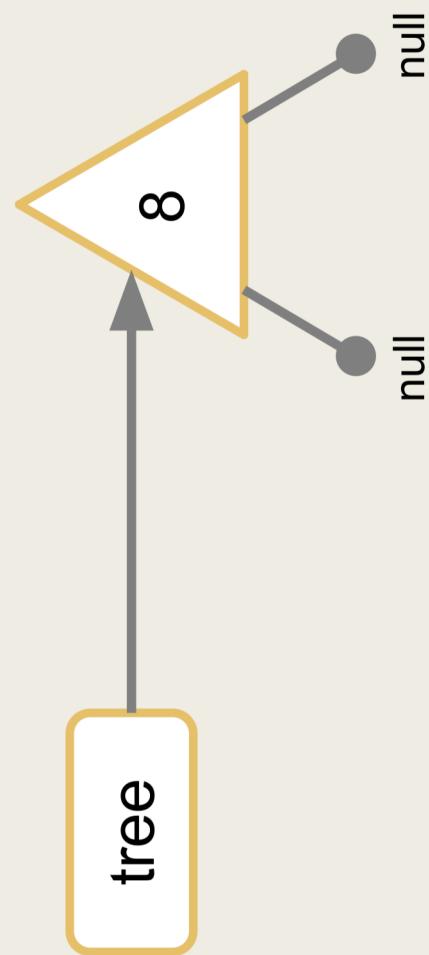


```
Tree tree = new Tree(8);
```

```
Tree tree = new Tree();
```

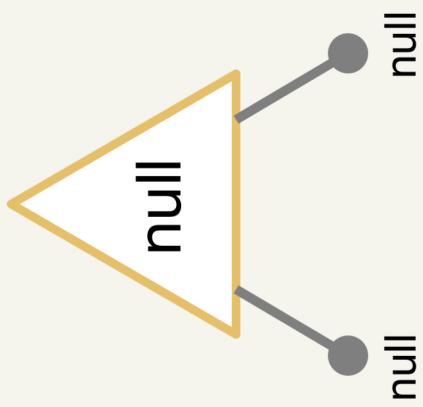


# Árbol con subárboles



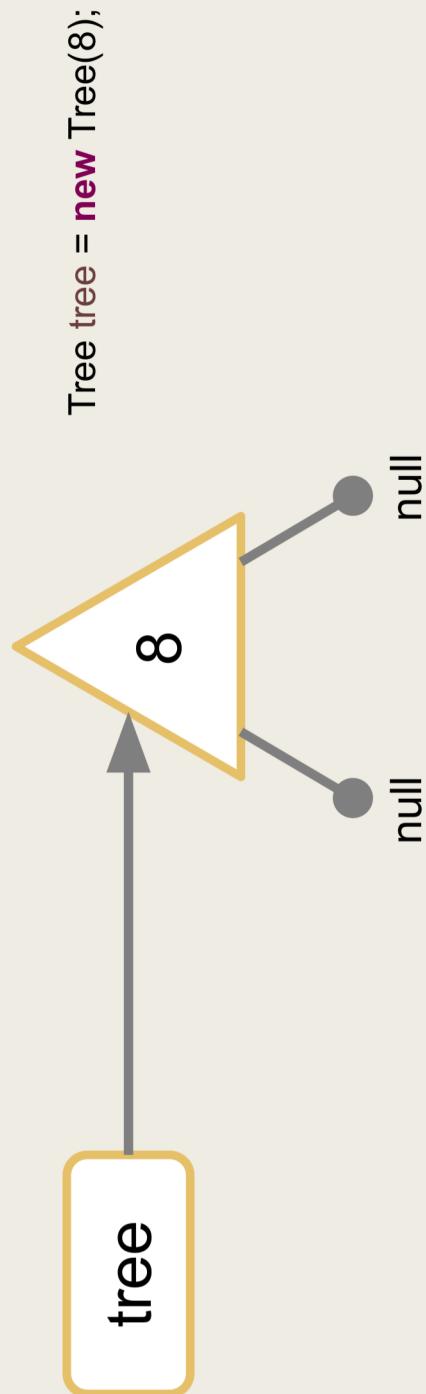
`Tree tree = new Tree(8);`

`Tree tree = new Tree();`

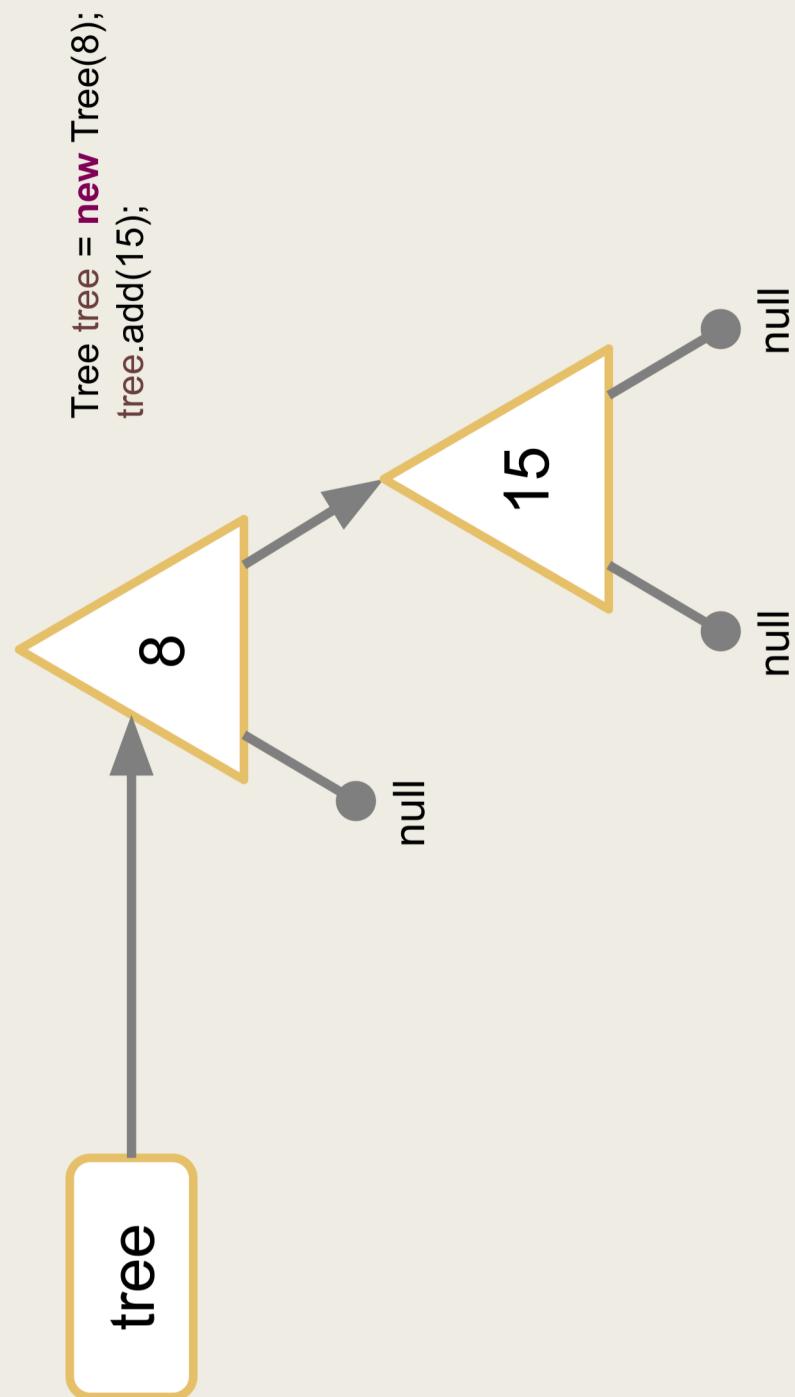


Tengo que andar chequeando siempre por null

# Árbol con subárboles

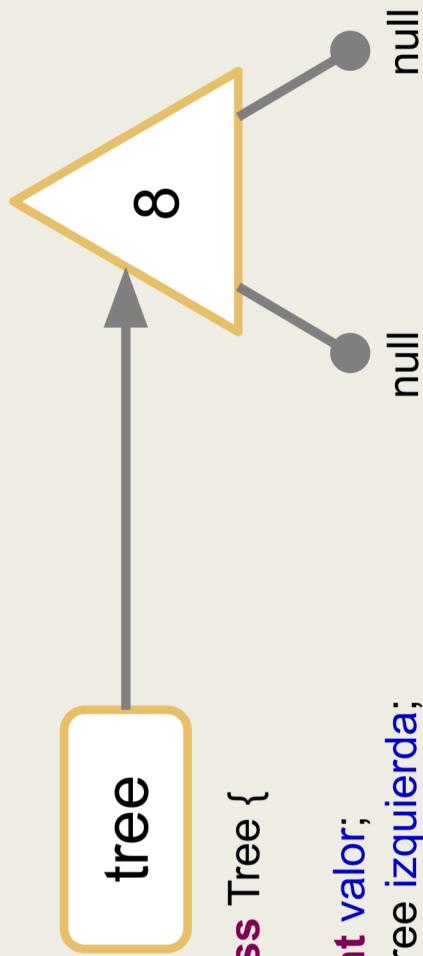


# Árbol con subárboles



# Implementación

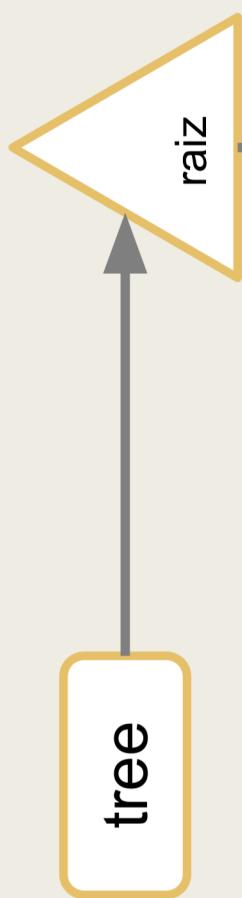
¿Cómo se implementa cada una de estas estructuras?



```
public class Tree {  
    private int valor;  
    private Tree izquierda;  
    private Tree derecha;  
  
    public Tree(int valor) {  
        this.valor = valor;  
        this.izquierda = null;  
        this.derecha = null;  
    }  
    ...  
}
```

# Implementación

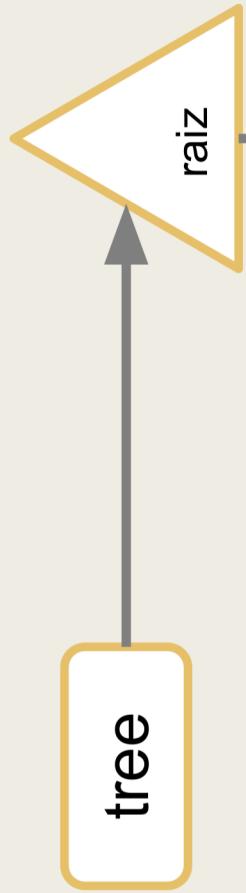
¿Cómo se implementa cada una de estas estructuras?



```
public class TreeWithNode {  
    private TreeNode raiz;  
  
    public TreeWithNode() {  
        this.raiz = null;  
    }  
    ...  
}
```

# Implementación

¿Cómo se implementa cada una de estas estructuras?

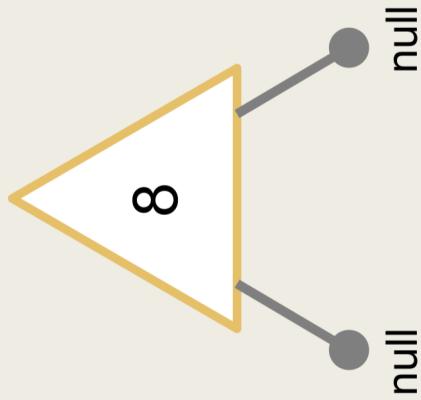


```
public class TreeWithNode {  
    private TreeNode raiz;  
  
    public TreeWithNode() {  
        this.raiz = null;  
    }  
    ...  
}  
  
public class TreeNode {  
    private int valor;  
    private TreeNode izquierda;  
    private TreeNode derecha;  
  
    public TreeNode(int value) {  
        this.valor = value;  
        this.izquierda = null;  
        this.derecha = null;  
    }  
    ...  
}
```

# Implementación

¿Cómo afecta a la hora de implementar los métodos (add, delete, etc.)?

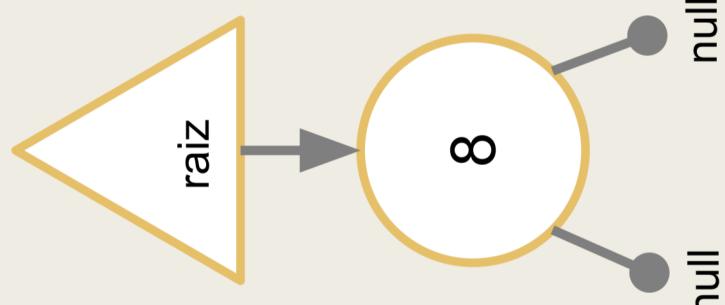
```
public void add(int newValue) {  
    if (this.valor > newValue) {  
        if (this.izquierda == null)  
            this.izquierda = new Tree(newValue);  
        else  
            this.izquierda.add(newValue);  
    } else if (this.valor < newValue) {  
        if (this.derecha == null)  
            this.derecha = new Tree(newValue);  
        else  
            this.derecha.add(newValue);  
    }  
}
```



# Implementación

¿Cómo afecta a la hora de implementar los métodos (add, delete, etc.)?

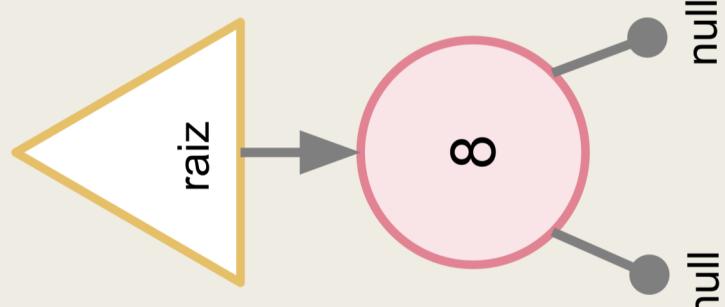
```
public void add(TreeNode nodo, int valor) {  
    if (nodo.getValor() > valor) {  
        if (nodo.getIzq() == null) {  
            TreeNode temp = new TreeNode(valor);  
            nodo.setIzq(temp);  
        } else {  
            add(nodo.getIzq(), valor);  
        }  
    } else if (nodo.getValor() < valor) {  
        if (nodo.getDer() == null) {  
            TreeNode temp = new TreeNode(valor);  
            nodo.setDer(temp);  
        } else {  
            add(nodo.getDer(), valor);  
        }  
    }  
}
```



# Implementación

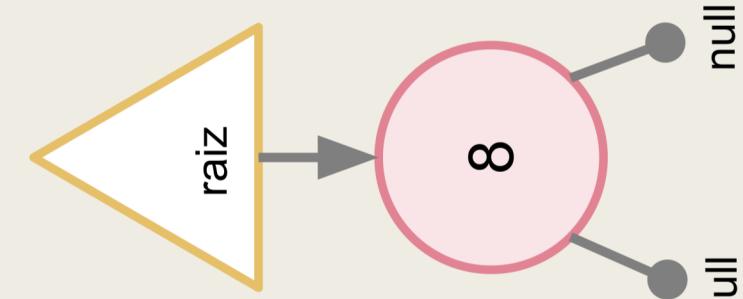
¿Cómo afecta a la hora de implementar los métodos (add, delete, etc.)?

```
public void add(TreeNode nodo, int valor) {
    if (nodo.getValor() > valor) {
        if (nodo.getIzq() == null) {
            TreeNode temp = new TreeNode(valor);
            nodo.setIzq(temp);
        } else {
            add(nodo.getIzq(), valor);
        }
    } else if (nodo.getValor() < valor) {
        if (nodo.getDer() == null) {
            TreeNode temp = new TreeNode(valor);
            nodo.setDer(temp);
        } else {
            add(nodo.getDer(), valor);
        }
    }
}
```



# Implementación

¿Cómo afecta a la hora de implementar los métodos (add, delete, etc.)?



```
public void add(TreeNode nodo, int valor) {
    if (nodo.getValor() > valor) {
        if (nodo.getIzq() == null) {
            TreeNode temp = new TreeNode(valor);
            nodo.setIzq(temp);
        } else {
            add(nodo.getIzq(), valor);
        }
    } else if (nodo.getValor() < valor) {
        if (nodo.getDer() == null) {
            TreeNode temp = new TreeNode(valor);
            nodo.setDer(temp);
        } else {
            add(nodo.getDer(), valor);
        }
    }
}
```

¡Rompo encapsulamiento!

# Implementación

¿Cómo afecta a la hora de implementar los métodos (add, delete, etc.)?

```
private void add(TreeNode nodo, int valor) {
    if (nodo.getValor() > valor) {
        if (nodo.getIzq() == null) {
            TreeNode temp = new TreeNode(valor);
            nodo.setIzq(temp);
        } else {
            add(nodo.getIzq(), valor);
        }
    } else if (nodo.getValor() < valor) {
        if (nodo.getDer() == null) {
            TreeNode temp = new TreeNode(valor);
            nodo.setDer(temp);
        } else {
            add(nodo.getDer(), valor);
        }
    }
}

public void add(int valor) {
    if (this.raiz == null)
        this.raiz = new TreeNode(valor);
    else
        this.add(this.raiz, valor);
}
```

# Implementación

¿Hay alguna implementación mejor que otra?

# Implementación

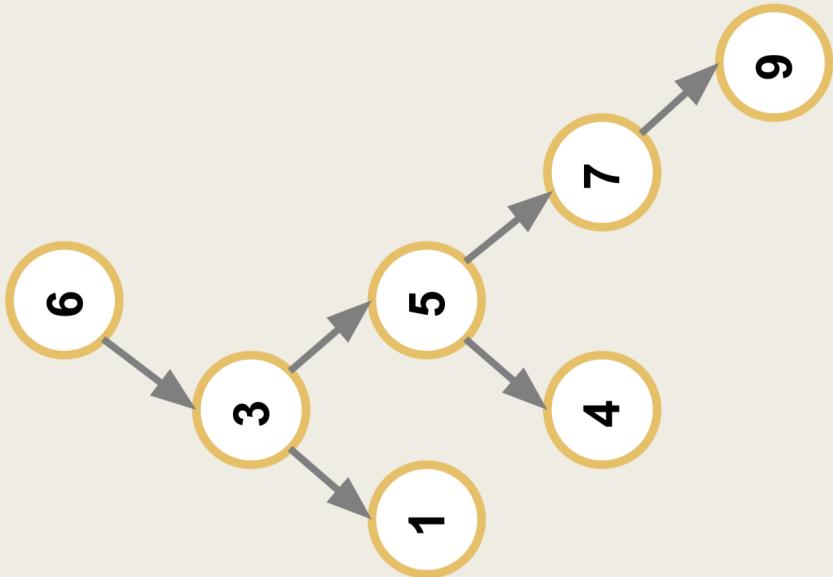
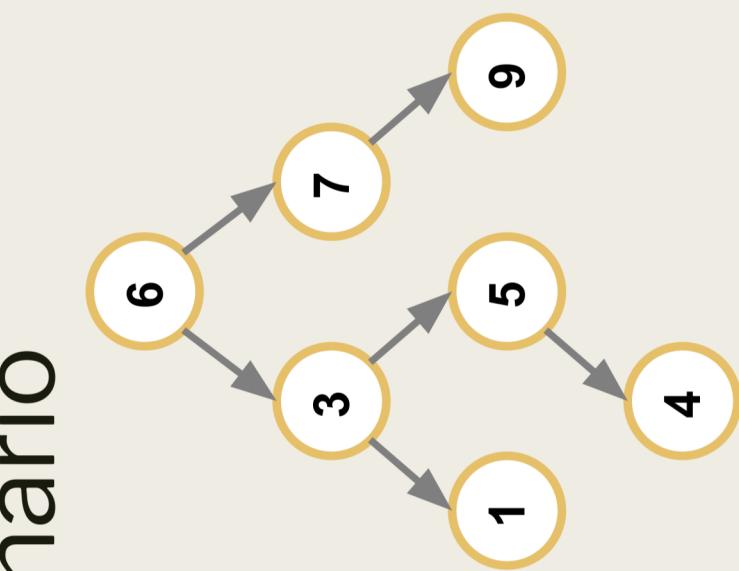
**¿Hay alguna implementación mejor que otra? NO**

La implementación de cada uno de los tipos de árboles puede presentar distintas complicaciones.

Para ciertas personas resulta más sencillo pensar y programar un árbol sin nodos, mientras que otros se sienten más cómodos manejando los nodos dentro del árbol.

**Implementen el que les resulte mas natural de pensar.**

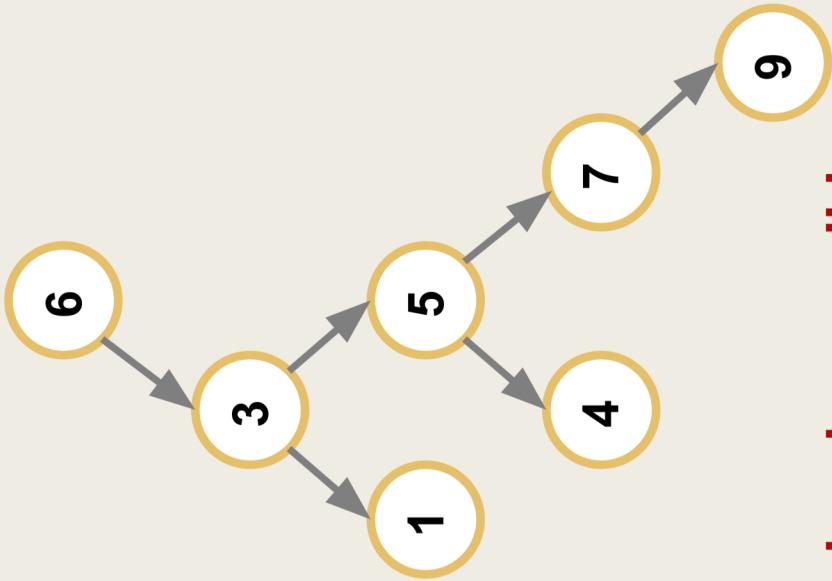
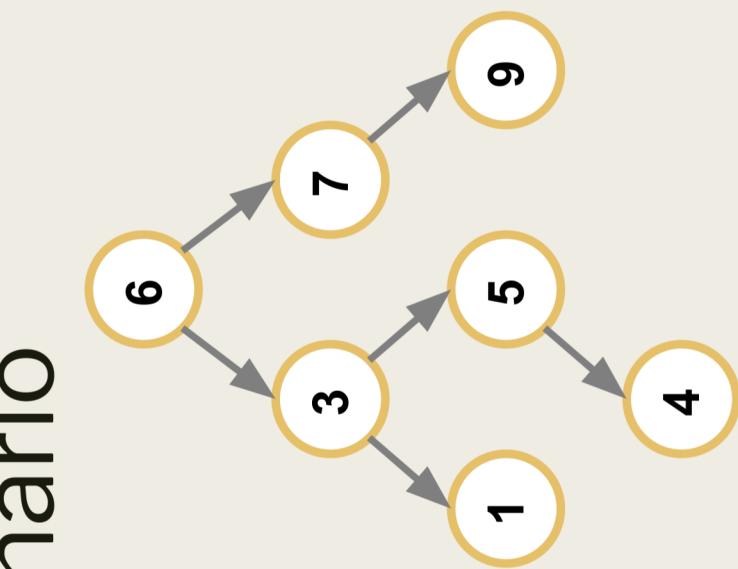
# Recorrido pre-orden en un árbol binario



**Salida:** 6 3 1 5 4 7 9

**Salida:** 6 3 1 5 4 7 9

# Recorrido pre-Orden en un árbol binario



**Dos árboles distintos pueden dar la misma salida**

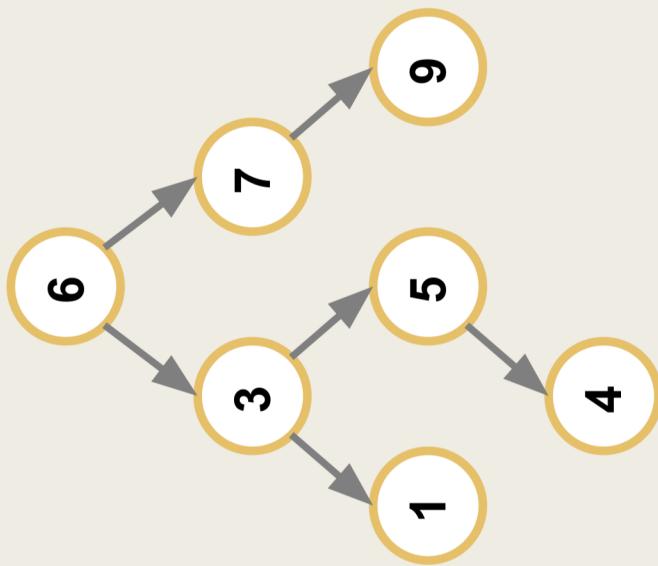
**Salida:** 6 3 1 5 4 7 9

**Salida:** 6 3 1 5 4 7 9

# Recorrido pre-orden en un árbol binario

Modificamos el recorrido pre-orden para que nos devuelva más información:

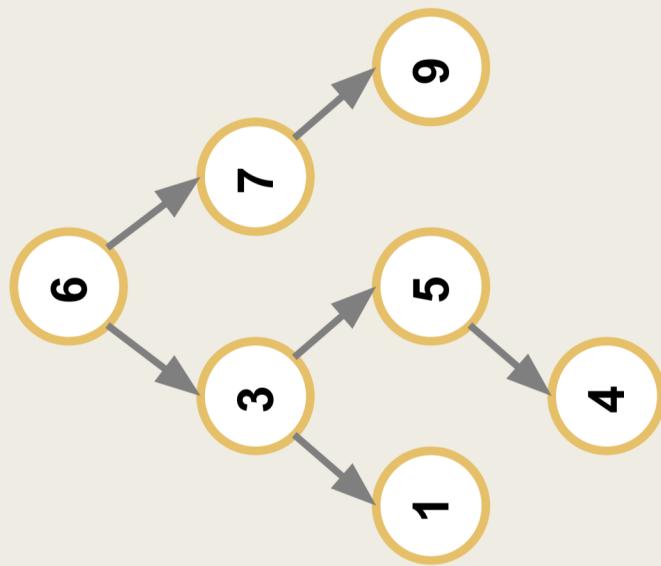
- Hacemos el recorrido normalmente mostrando los valores de los nodos.
- Cuando nos encontramos con un nodo vacío (es decir, un izquierda o derecha en null) mostramos un carácter especial (“-”).



# Recorrido pre-orden en un árbol binario

Modificamos el recorrido pre-orden para que nos devuelva más información:

- Hacemos el recorrido normalmente mostrando los valores de los nodos.
- Cuando nos encontramos con un nodo vacío (es decir, un izquierda o derecha en null) mostramos un carácter especial ("‐").



**Salida:** 6 3 1 - - 5 4 - - - 7 - 9 - -