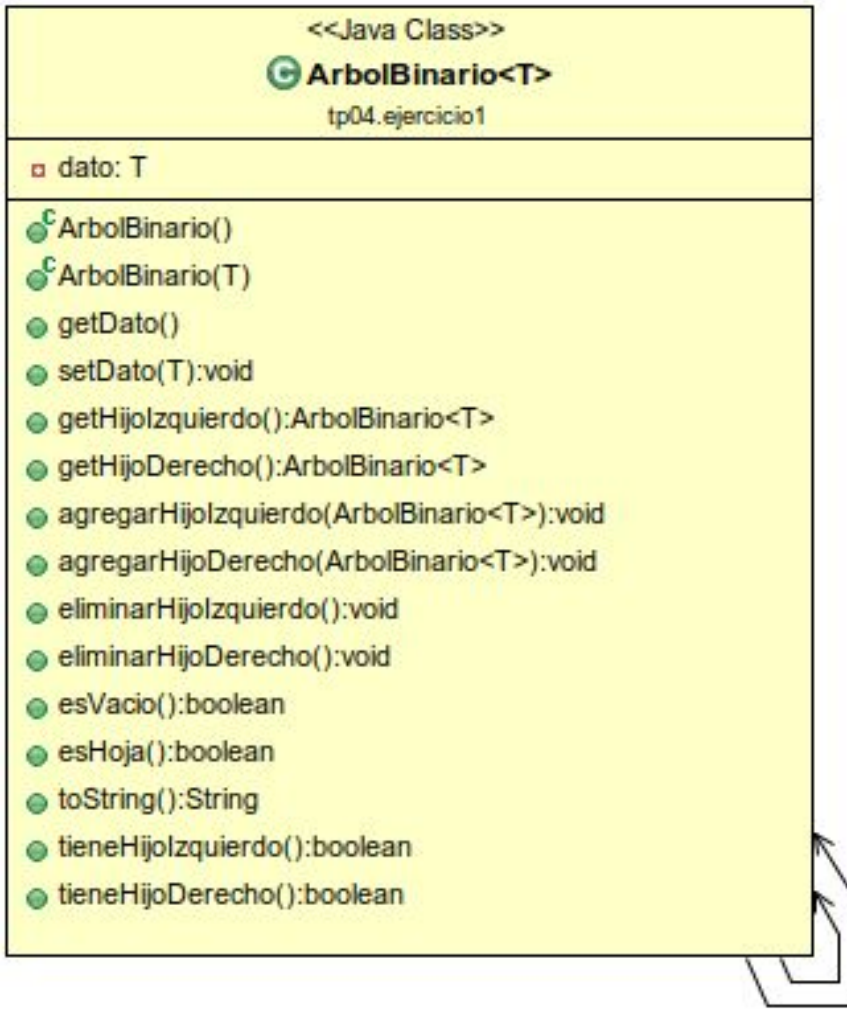
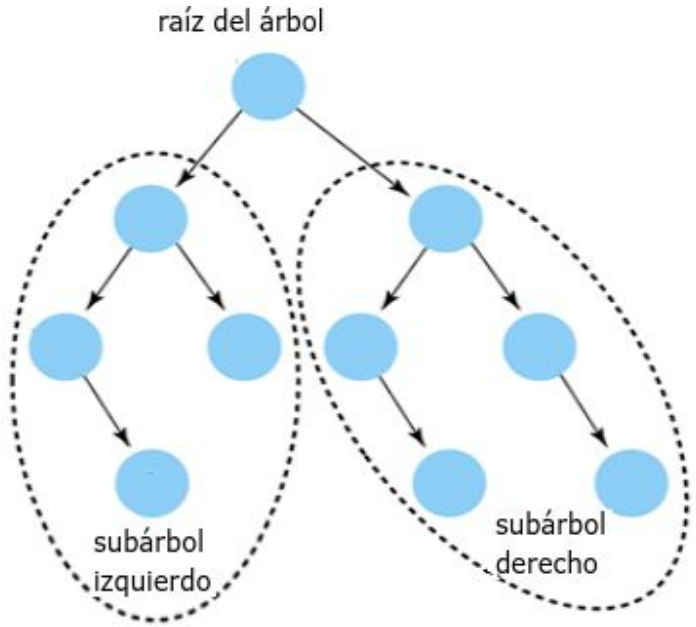
Arboles Binarios

Estructura



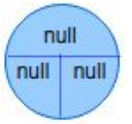


Arboles Binarios

Código Fuente

```
package tp03.ejercicio1;
public class ArbolBinario<T> {
    private T dato;
    private ArbolBinario<T> hijoIzquierdo;
    private ArbolBinario<T> hijoDerecho;

    public ArbolBinario() {
       super();
    }
    public ArbolBinario(T dato) {
       this.dato = dato;
    }

    public T getDato() {
       return dato;
    }

    public void setDato(T dato) {
       this.dato = dato;
    }
    public ArbolBinario<T> getHijoIzquierdo() {
       return this.hijoIzquierdo;
    }
    public ArbolBinario<T> getHijoDerecho() {
       return this.hijoDerecho;
    }
    public void agregarHijoIzquierdo(ArbolBinario<T> hijo) {
       this.hijoIzquierdo = hijo;
    }
    public void agregarHijoDerecho(ArbolBinario<T> hijo) {
       this.hijoDerecho = hijo;
    }
    public void eliminarHijoIzquierdo() {
       this.hijoIzquierdo = null;
    }
    public void eliminarHijoDerecho() {
       this.hijoDerecho = null;
    }
    public boolean esVacio() {
       return (this.esHoja() && this.getDato()==null);
    }
    public boolean esHoja() {
                return (!this.tieneHijoIzquierdo() &&
                        !this.tieneHijoDerecho());
    }
    public boolean tieneHijoIzquierdo() {
       return this.hijoIzquierdo!=null;
    }

    public boolean tieneHijoDerecho() {
       return this.hijoDerecho!=null;
    }
}
```

Constructores

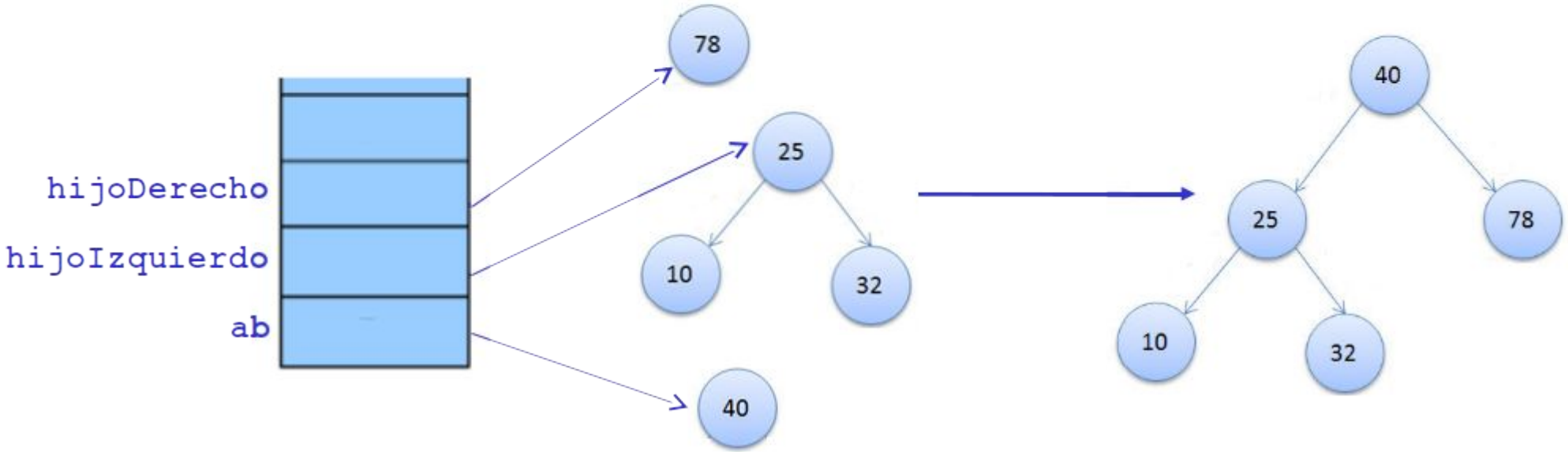


Arbol vacío

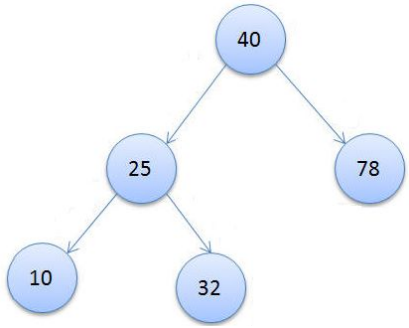
Arboles Binarios

Creación

```
ArbolBinario<Integer> ab = new ArbolBinario<Integer>(new Integer(40));
ArbolBinario<Integer> hijoIzquierdo = new ArbolBinario<Integer>(25);
hijoIzquierdo.agregarHijoIzquierdo(new ArbolBinario<Integer>(10));
hijoIzquierdo.agregarHijoDerecho(new ArbolBinario<Integer>(32));
ArbolBinario<Integer> hijoDerecho = new ArbolBinario<Integer>(78);
ab.agregarHijoIzquierdo(hijoIzquierdo);
ab.agregarHijoDerecho(hijoDerecho);
```



Arboles Binarios
Recorridos



Preorden
Se procesa primero la raíz y luego sus hijos, izquierdo y derecho.

```
40, 25, 10, 32, 78
```

Inorden
Se procesa el hijo izquierdo, luego la raíz y último el hijo derecho

```
10, 25, 32, 40, 78
```

Postorden
Se procesan primero los hijos, izquierdo y derecho, y luego la raíz

```
10, 32, 25, 78, 40
```

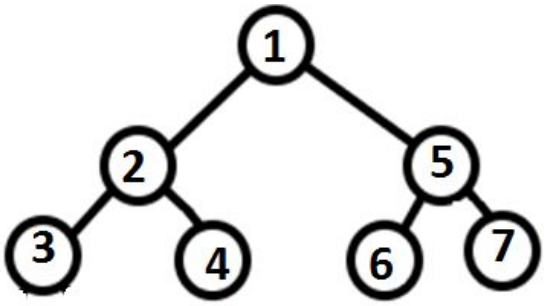
Por niveles
Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc.

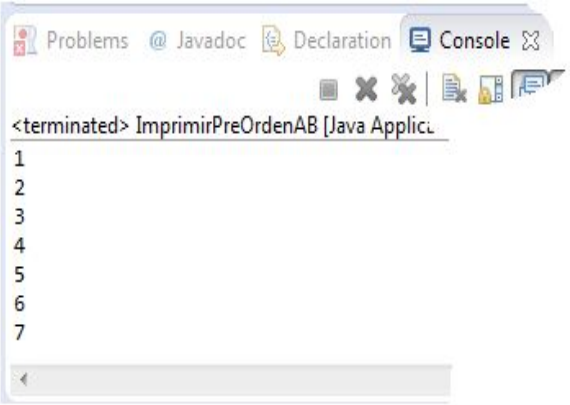
```
40, 25, 78, 10, 32
```

Arboles Binarios
Recorrido PreOrden

Se procesa primero la raíz y luego sus hijos, izquierdo y derecho

```
public class ArbolBinario<T> {
    private T dato;
    private ArbolBinario<T> hijoIzquierdo;
    private ArbolBinario<T> hijoDerecho;

    public void printPreorden() {
        if(!this.esVacio()) {
            System.out.println(this.getDato());

            if(this.tieneHijoIzquierdo())
                this.getHijoIzquierdo().printPreorden();

            if(this.tieneHijoDerecho())
                this.getHijoDerecho().printPreorden();
        }
    }
```

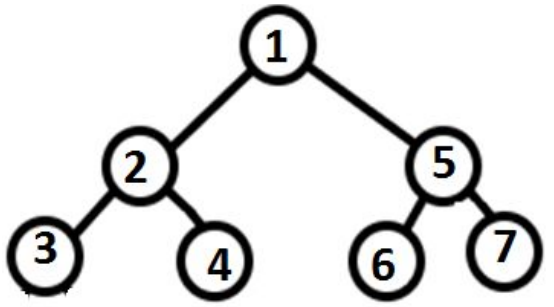


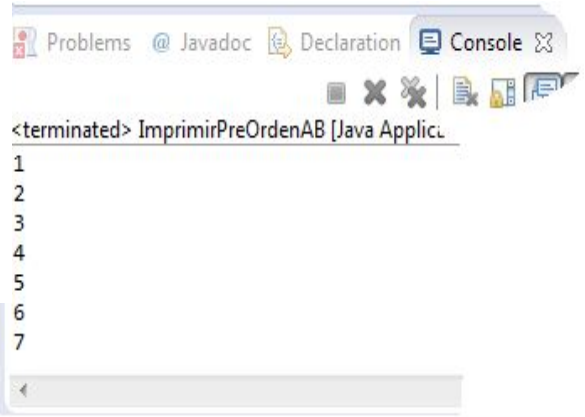


Arboles Binarios
Recorrido PreOrden

Qué cambio harias si el método preorden() debe definirse en otra clase diferente al ArbolBinario<T>?

```
public class ArbolBinarioExamples<T> {
    public void preorder(ArbolBinario<T> arbol) {
        if(!arbol.esVacio()) {
            System.out.println(arbol.getDato());

            if(arbol.tieneHijoIzquierdo())
                this.preorder(arbol.getHijoIzquierdo());

            if(arbol.tieneHijoDerecho())
                this.preorder(arbol.getHijoDerecho());
        }
    }
```





Arboles Binarios
Recorrido PreOrden

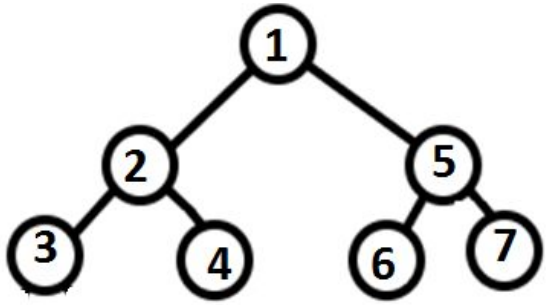
Qué cambio harías para devolver una lista con los elementos de un recorrido en preorden?

```
public class ArbolBinarioExamples<T> {

    public ListaGenerica<T> preorder(ArbolBinario<T> arbol){
        ListaGenerica<T> result = new ListaEnlazadaGenerica<T>();
        this.preorder_private(arbol, result);
        return result;
    }

    private void preorder_private(ArbolBinario<T> arbol, ListaGenerica<T> result) {
        if (!arbol.esVacio()) {
            result.agregarFinal(arbol.getDato());
            if (arbol.tieneHijoIzquierdo())
                this.preorder_private(arbol.getHijoIzquierdo(), result);
            if (arbol.tieneHijoDerecho())
                this.preorder_private(arbol.getHijoDerecho(), result);
        }
    }
```

Arboles Binarios

Recorrido por Niveles

Recorrido implementado en la clase **`ArbolBinario`**



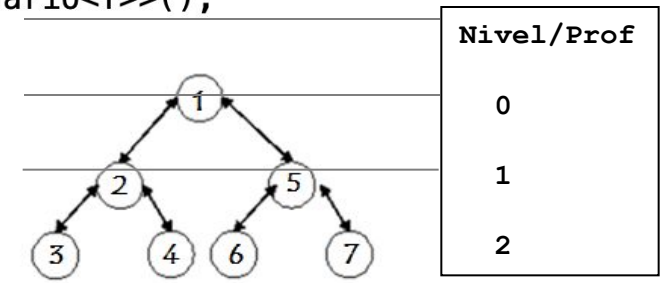

```
public class ArbolBinario<T> {
    private T dato;
    private ArbolBinario<T> hijoIzquierdo;
    private ArbolBinario<T> hijoDerecho;

    public void recorridoPorNiveles() {
        if (!this.esVacio()) {
            ArbolBinario<T> arbol_aux = null;
            ColaGenerica<ArbolBinario<T>> cola = new ColaGenerica<ArbolBinario<T>>();
            cola.encolar(this);
            cola.encolar(null);
            while(!cola.esVacia()) {
                arbol_aux = cola.desencolar();
                if (arbol_aux != null) {
                    System.out.println(arbol_aux.getDato());

                    if(arbol_aux.tieneHijoIzquierdo())
                        cola.encolar(arbol_aux.getHijoIzquierdo());

                    if(arbol_aux.tieneHijoDerecho())
                        cola.encolar(arbol_aux.getHijoDerecho());

                }else if (!cola.esVacia()) {
                    cola.encolar(null);
                }
            }
        }
    }
```

Arboles Binarios

Es árbol lleno?

Dado un árbol binario de altura h, diremos que es lleno si cada nodo interno tiene grado 2 y todas las hojas están en el mismo nivel (h). Implementar un método para determinar si un árbol binario es “lleno”

```
public boolean lleno() {
     if (this.esVacio()) return false;
     ArbolBinario<T> arbol = null;
     ColaGenerica<ArbolBinario<T>> cola = new  ColaGenerica<ArbolBinario<T>>();
     cola.encolar(this);
     int cant_nodos=0;
     cola.encolar(null);
     int nivel= 0;
     while (!cola.esVacia()) {
       arbol = cola.desencolar();
       if (arbol != null) {
          cant_nodos++;
          if (arbol.tieneHijoIzquierdo())
            cola.encolar(arbol.getHijoIzquierdo());

          if (arbol.tieneHijoDerecho())
            cola.encolar(arbol.getHijoDerecho());
       }else {
          if (cant_nodos != Math.pow(2, nivel))
              return false;
          if (!cola.esVacia()) {
                nivel++;
                cola.encolar(null);
               cant_nodos=0;
             }
       }
    }
    return true;
}
```



cola

1	null	2	5	