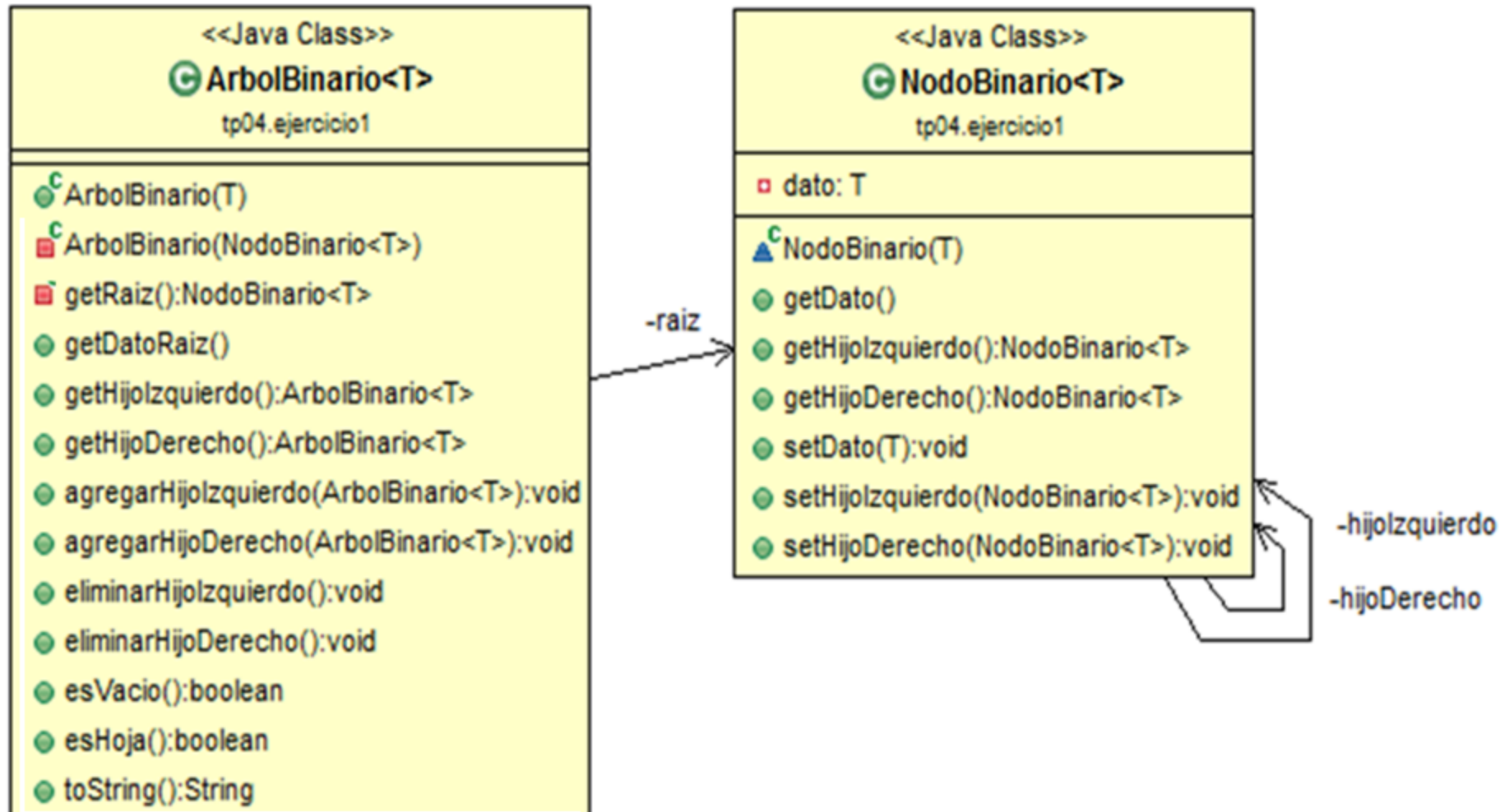


Arboles Binarios

Estructura



Arboles Binarios

Código Fuente

```
package tp03;

public class ArbolBinario<T> {

    private NodoBinario<T> raiz;

    public ArbolBinario(T dato) {
        this.raiz = new NodoBinario<T>(dato);
    }

    private ArbolBinario(NodoBinario<T> nodo) {
        this.raiz = nodo;
    }

    private NodoBinario<T> getRaiz() {
        return this.raiz;
    }

    public T getDatoRaiz() {
        return (this.raiz==null) ? null:this.raiz.getDato();
    }

    public ArbolBinario<T> getHijoIzquierdo() {
        return
            new ArbolBinario<T>(this.raiz.getHijoIzquierdo());
    }

    public void agregarHijoIzquierdo(ArbolBinario<T> hijo){
        this.raiz.setHijoIzquierdo(hijo.getRaiz());
    }
    . . .
}
```

```
package tp03;

public class NodoBinario<T> {
    private T dato;
    private NodoBinario<T> hijoIzquierdo;
    private NodoBinario<T> hijoDerecho;

    NodoBinario(T dato){
        this.dato = dato;
    }

    public T getDato(){
        return this.dato;
    }

    NodoBinario<T> getHijoIzquierdo(){
        return this.hijoIzquierdo;
    }

    NodoBinario<T> getHijoDerecho(){
        return this.hijoDerecho;
    }

    public void setDato(T dato){
        this.dato = dato;
    }

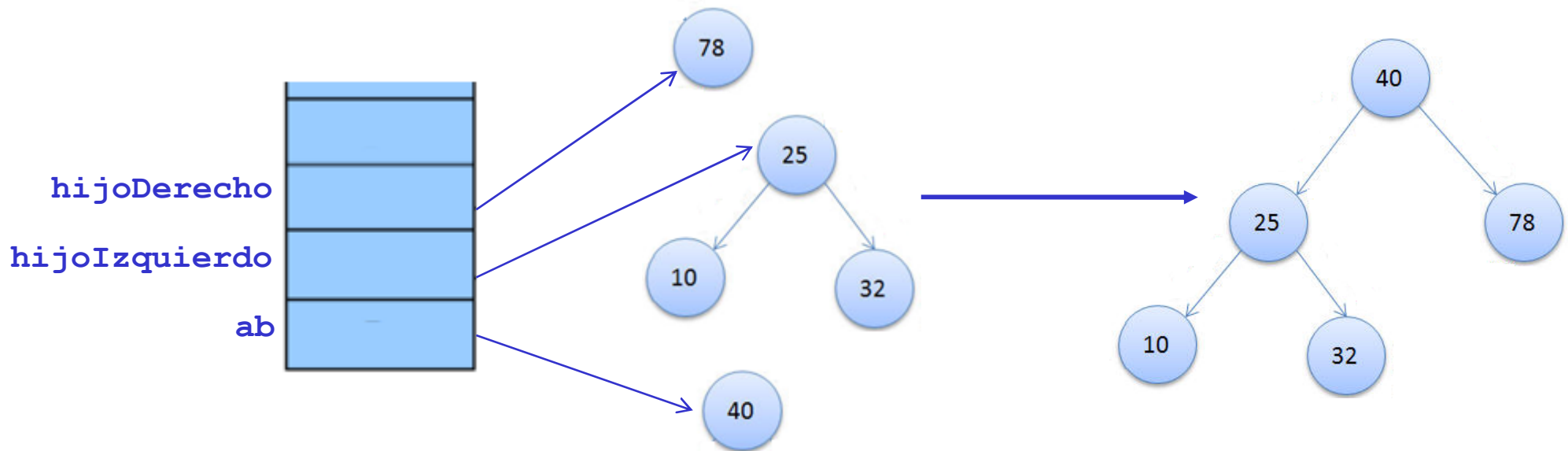
    public void setHijoIzquierdo(NodoBinario<T> hijoIzq){
        this.hijoIzquierdo = hijoIzq;
    }

    public void setHijoDerecho(NodoBinario<T> hijoDer){
        this.hijoDerecho = hijoDer;
    }
}
```

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

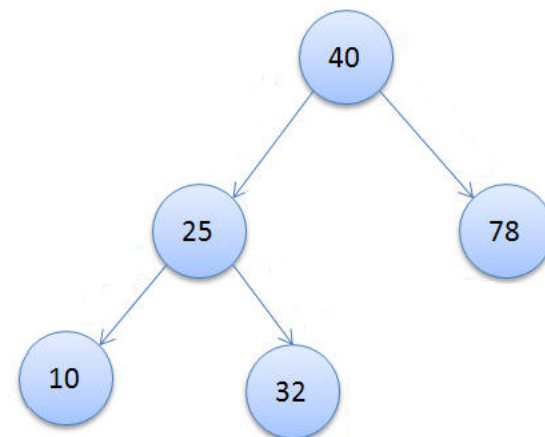
Se procesan primero los hijos, izquierdo y derecho, y luego la raíz

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

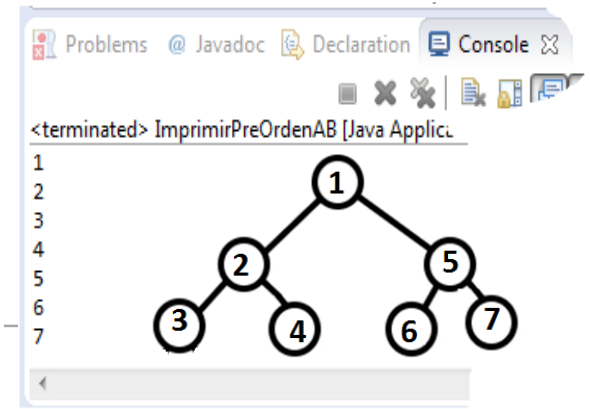


Arboles Binarios

Recorrido PreOrden

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

```
public class ArbolBinario<T> {  
    private NodoBinario<T> raiz;  
    . . .  
    public void printPreorden() {  
        System.out.println(this.getDatoRaiz());  
        if (!this.getHijoIzquierdo().esVacio())  
            this.getHijoIzquierdo().printPreorden();  
        }  
        if (!this.getHijoDerecho().esVacio()) {  
            this.getHijoDerecho().printPreorden();  
        }  
    }  
    public boolean esVacio() {  
        return (this.getDatoRaiz() == null);  
    }  
}
```

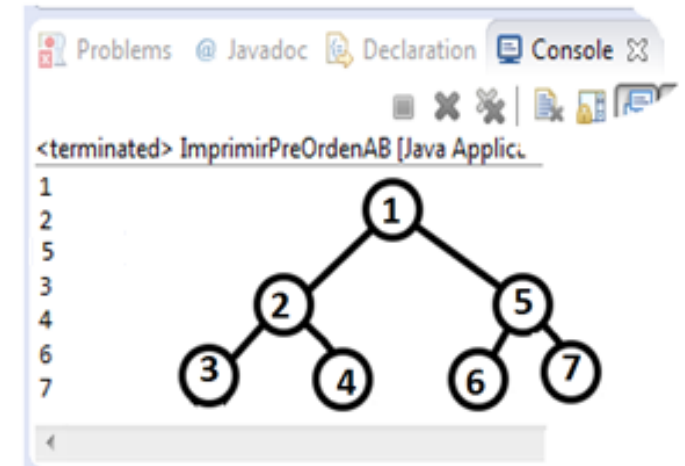


Arboles Binarios

Recorrido por Niveles

Recorrido implementado en la clase ArbolBinario

```
public class ArbolBinario<T> {  
  
    private NodoBinario<T> raiz;  
    . . .  
    public void recorridoPorNiveles() {  
        ArbolBinario<T> arbol = null;  
        ColaGenerica<ArbolBinario<T>> cola = new ColaGenerica<ArbolBinario<T>>();  
        cola.encolar(this);  
        cola.encolar(null);  
        while (!cola.esVacia()) {  
            arbol = cola.desencolar();  
            if (arbol != null) {  
                System.out.println(arbol.getDatoRaiz());  
                if (!arbol.getHijoIzquierdo().esVacio()) {  
                    cola.encolar(arbol.getHijoIzquierdo());  
                }  
                if (!arbol.getHijoDerecho().esVacio()) {  
                    cola.encolar(arbol.getHijoDerecho());  
                }  
            } else {  
                if (!cola.esVacia()) {  
                    System.out.println();  
                    cola.encolar(null);  
                }  
            }  
        }  
    }  
}
```

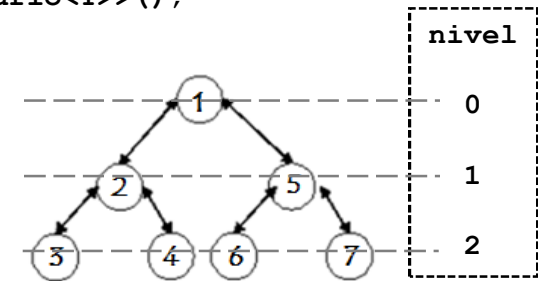


Arboles Binarios

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). Este método determine si un árbol binario es lleno.

```
public boolean lleno() {
    ArbolBinario<T> arbol = null;
    ColaGenerica<ArbolBinario<T>> cola = new ColaGenerica<ArbolBinario<T>>();
    boolean lleno = true;
    cola.encolar(this);
    int cant_nodos=0;
    cola.encolar(null);
    int nivel= 0;
    while (!cola.esVacia() && lleno) {
        arbol = cola.desencolar();
        if (arbol != null) {
            System.out.print(arbol.getDatoRaiz());
            if (!arbol.getHijoIzquierdo().esvacio()) {
                cola.encolar(arbol.getHijoIzquierdo());
                cant_nodos++;
            }
            if (!arbol.getHijoDerecho().esvacio()) {
                cola.encolar(arbol.getHijoDerecho());
                cant_nodos++;
            }
        } else if (!cola.esVacia()) {
            if (cant_nodos == Math.pow(2, ++nivel)){
                cola.encolar(null);
                cant_nodos=0;
                System.out.println();
            }
            else lleno=false;
        }
        return lleno;
    }
}
```



`arbol = cola.desencolar();`

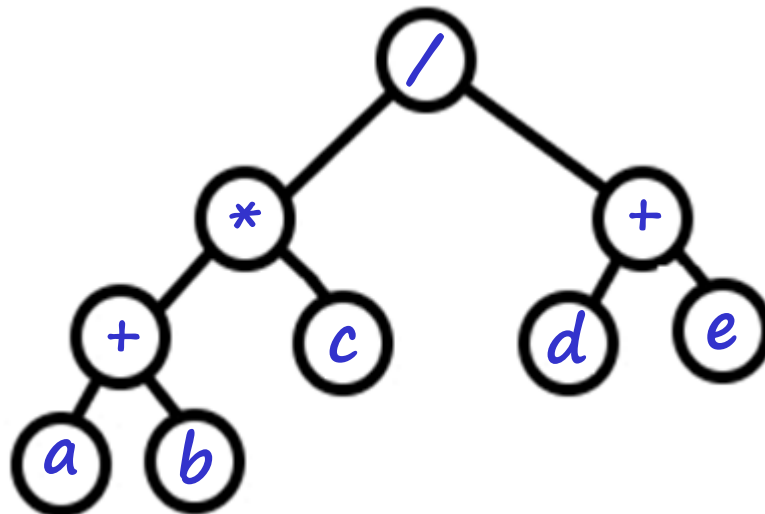
`arbol = null`
`cant_nodos = 2`
`nivel= 0 -> 1`

Arboles Binarios

Arbol de Expresión

Un árbol de expresión es un árbol binario asociado a una expresión aritmética donde:

- Nodos internos representan operadores
- Nodos externos (hojas) representan operandos



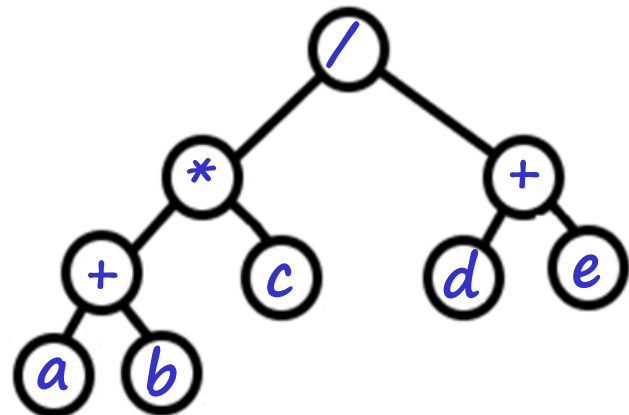
Arboles Binarios

Convertir expresión posfija en Árbol de Expresión

Este método convierte una expresión *posfija* en un `ArbolBinario`. Puede estar implementado en cualquier clase.

```
public ArbolBinario<Character> convertirPostfija(String exp) {  
    Character c = null;  
    ArbolBinario<Character> result;  
    PilaGenerica<ArbolBinario<Character>> p = new PilaGenerica<ArbolBinario<Character>>();  
  
    for (int i = 0; i < exp.length(); i++) {  
        c = exp.charAt(i);  
        result = new ArbolBinario<Character>(c);  
        if ((c == '+') || (c == '-') || (c == '/') || (c == '*')) {  
            // Es operador  
            result.agregarHijoDerecho(p.desapilar());  
            result.agregarHijoIzquierdo(p.desapilar());  
        }  
        p.apilar(result);  
    }  
    return (p.desapilar());  
}
```

$ab+c*de+ /$



Arboles Binarios

Convertir expresión prefija en Arbol de Expresión

Este método convierte una expresión **prefija** en un `ArbolBinario`. Puede estar implementado en cualquier clase.

```
public ArbolBinario<Character> convertirPrefija(StringBuffer exp) {  
    Character c = exp.charAt(0);  
    ArbolBinario<Character> result = new ArbolBinario<Character>(c);  
    if ((c == '+') || (c == '-') || (c == '/') || c == '*') {  
        // es operador  
        result.agregarHijoIzquierdo(this.convertirPrefija(exp.delete(0,1)));  
        result.agregarHijoDerecho(this.convertirPrefija(exp.delete(0,1)));  
    }  
    // es operando  
    return result;  
}
```

/+abc+de*

