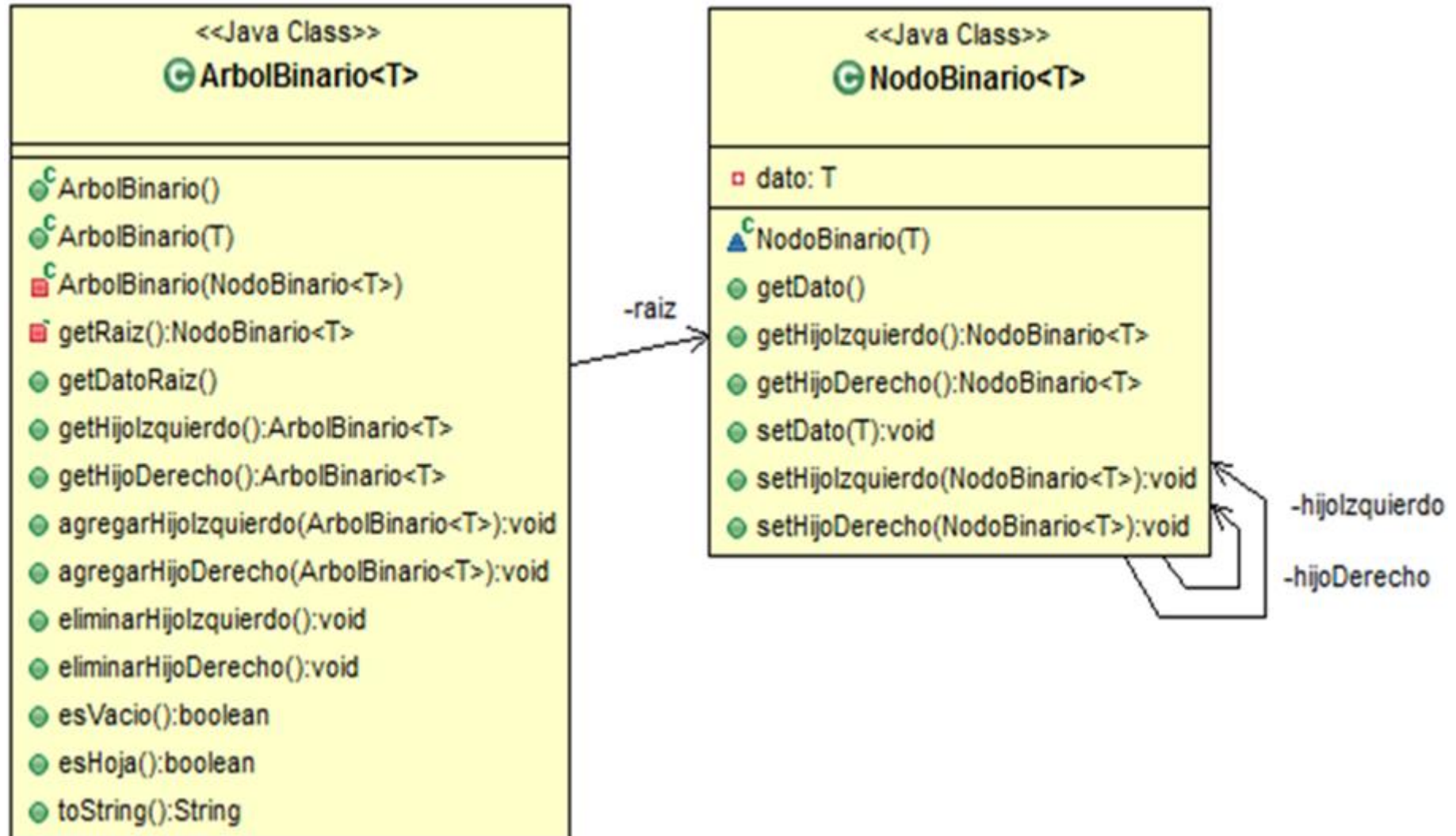


Arboles Binarios

Estructura



Arboles Binarios

Código Fuente

```
package tp02;

public class ArbolBinario<T> {
    private NodoBinario<T> raiz;

    public ArbolBinario() {
        this.raiz = null;
    }
    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 tp02;

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;
    }

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

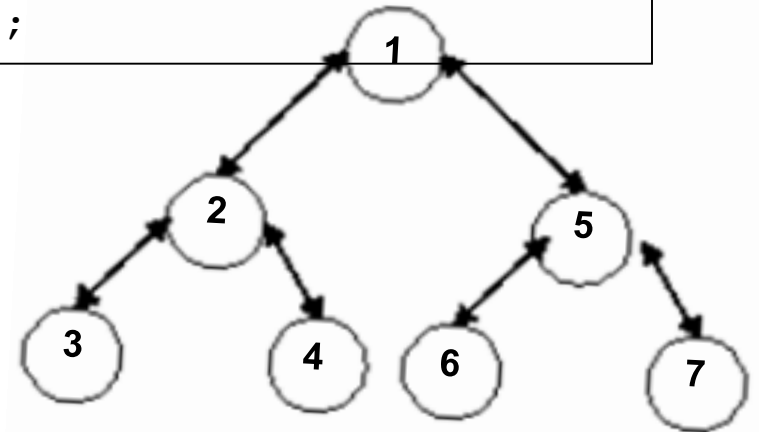
Creación

```
ArbolBinario<Integer> arbolBinarioA = new ArbolBinario<Integer>(1);
```

```
ArbolBinario<Integer> hijoIzquierdo=new ArbolBinario<Integer>(2);  
hijoIzquierdo.agregarHijoIzquierdo(new ArbolBinario<Integer>(3));  
hijoIzquierdo.agregarHijoDerecho(new ArbolBinario<Integer>(4));
```

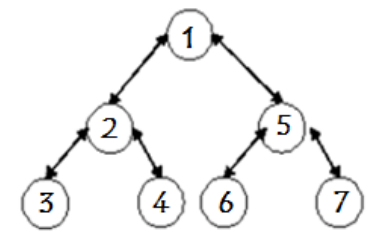
```
ArbolBinario<Integer> hijoDerecho=new ArbolBinario<Integer>(5);  
hijoDerecho.agregarHijoIzquierdo(new ArbolBinario<Integer>(6));  
hijoDerecho.agregarHijoDerecho(new ArbolBinario<Integer>(7));
```

```
arbolBinarioA.agregarHijoIzquierdo(hijoIzquierdo);  
arbolBinarioA.agregarHijoDerecho(hijoDerecho);
```



Arboles Binarios

Recorridos



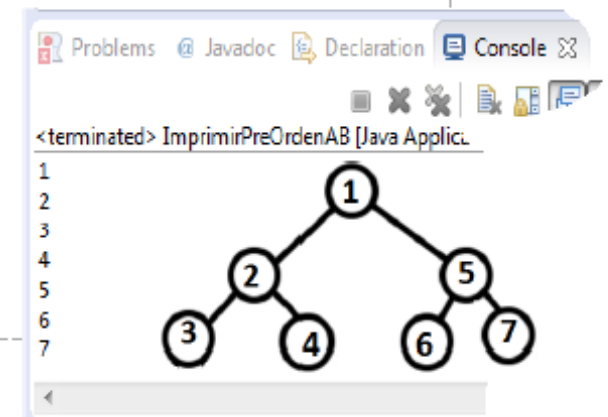
- **Preorden:** Se procesa primero la raíz y luego sus hijos, izquierdo y derecho.
1, 2, 3, 4, 5, 6, 7
- **Inorden:** Se procesa el hijo izquierdo, luego la raíz y último el hijo derecho.
3, 2, 4, 1, 6, 5, 7
- **Postorden:** Se procesan primero los hijos, izquierdo y derecho, y luego la raíz.
3, 4, 2, 6, 7, 5, 1
- **Por niveles:** Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc.
1, 2, 5, 3, 4, 6, 7

Arboles Binarios

Recorrido Preorden

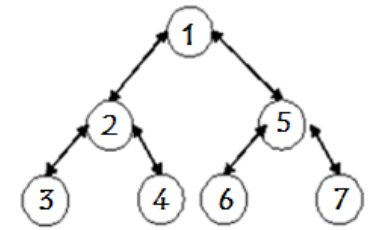
Se procesa primero la raíz y luego sus hijos, izquierdo y derecho.

```
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 void recorridoPorNiveles() {

    ArbolBinario<T> arbol = null;
    ColaGenerica<ArbolBinario<T>> cola = new ColaGenerica<ArbolBinario<T>>();
    cola.encolar(this);
    cola.encolar(null);

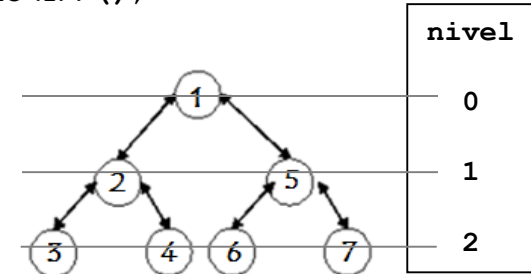
    while (!cola.esVacia()) {
        arbol = cola.desencolar();
        if (arbol != null) {
            System.out.print(arbol.getDatoRaiz());
            if ((arbol.getHijoIzquierdo().getDatoRaiz() != null))
                cola.encolar(arbol.getHijoIzquierdo());
            if ((arbol.getHijoDerecho().getDatoRaiz() != null))
                cola.encolar(arbol.getHijoDerecho());
        } else
            if (!cola.esVacia()) {
                System.out.println();
                cola.encolar(null);
            }
    }
}
```

Arboles Binarios

Recorrido por Niveles

Implementar un método que 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().getDatoRaiz() != null)) {
                cola.encolar(arbol.getHijoIzquierdo());
                cant_nodos++;
            }
            if ((arbol.getHijoDerecho().getDatoRaiz() != null)) {
                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= 1

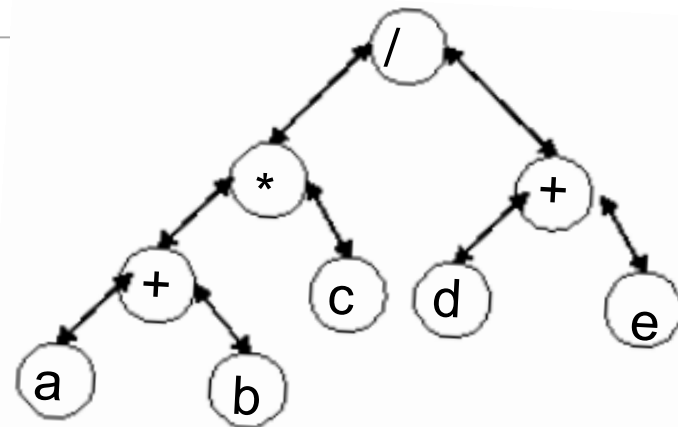
Arboles Binarios

Convertir expresión prefija en Arbol Binario

Convierte una expresión **prefija** en un ArbolBinario

```
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



Arboles Binarios

Convertir expresión posfija en Árbol Binario

Convierte una expresión **posfija** en un Árbol Binario.

```
public ArbolBinario<Character> convertirPostfija(String exp) {  
  
    ArbolBinario<Character> result;  
    PilaGenerica<ArbolBinario<Character>> p = new PilaGenerica<ArbolBinario<Character>>();  
  
    for (int i = 0; i < exp.length(); i++) {  
        Character 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+/

