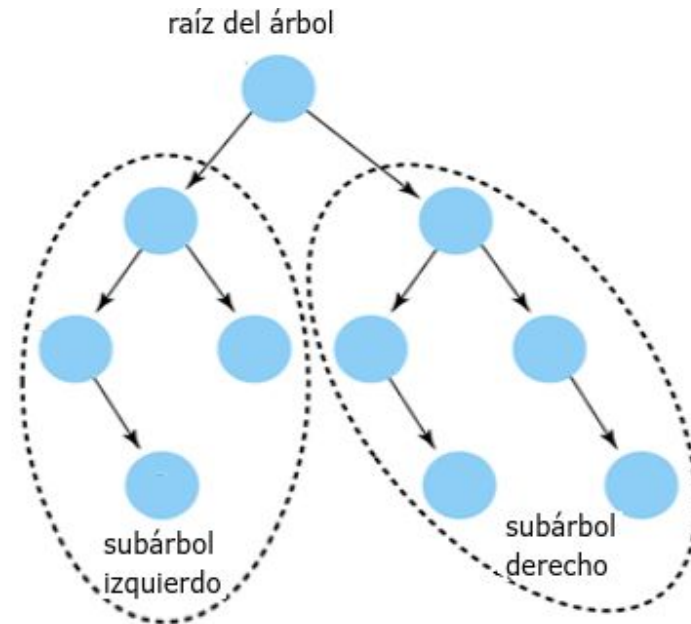
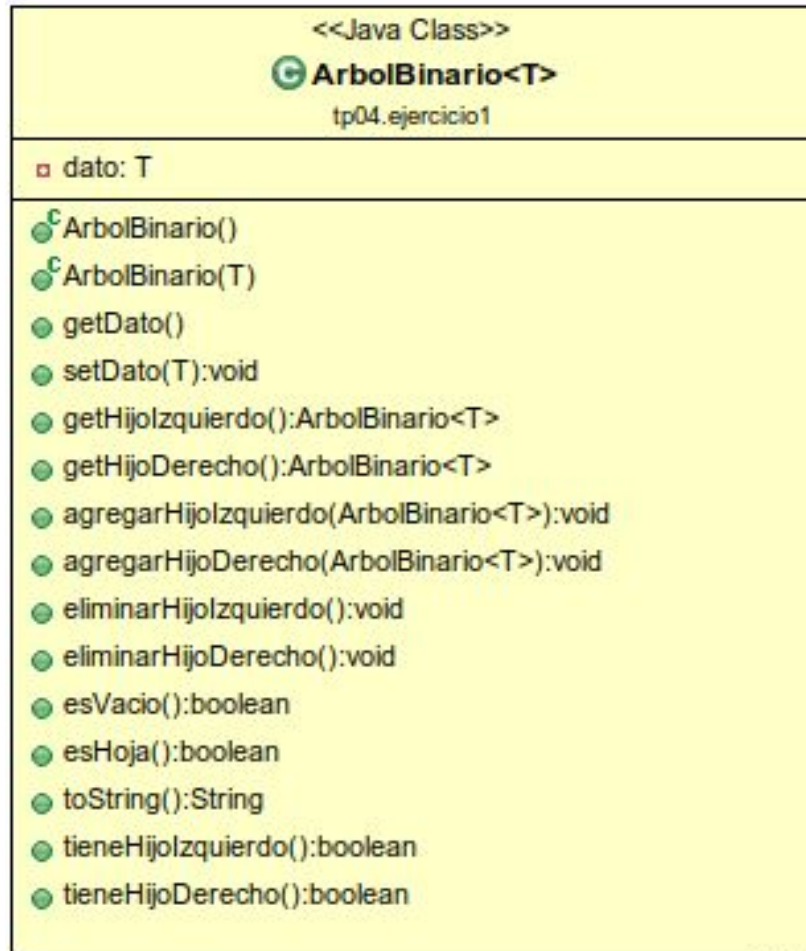


Arboles Binarios

Estructura



-hijoDerecho

-hijoIzquierdo

Arboles Binarios

Código Fuente

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

    public ArbolBinario() { Constructores
        super();
    }

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

    public T getData() {
        return dato;
    }

    public void setData(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;
}

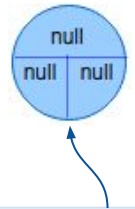
public void eliminarHijoDerecho() {
    this.hijoDerecho = null;
}

public boolean esVacio() {
    return (this.esHoja() && this.getData()==null);
}

public boolean esHoja() {
    return (!this.tieneHijoIzquierdo() &&
        !this.tieneHijoDerecho());
}

public boolean tieneHijoIzquierdo() {
    return this.hijoIzquierdo!=null;
}

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

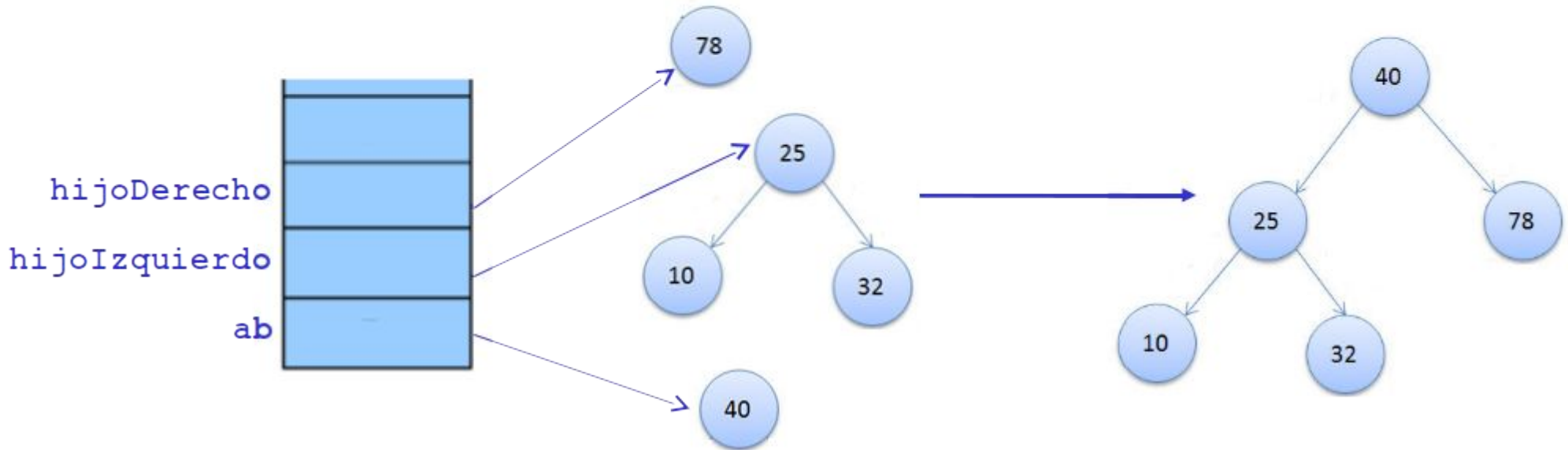


Arbol vacío

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

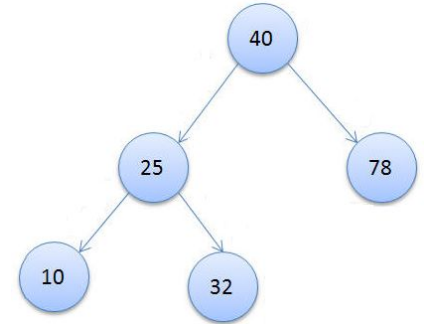
Se procesan primero los hijos, izquierdo y derecho, y luego la raíz

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

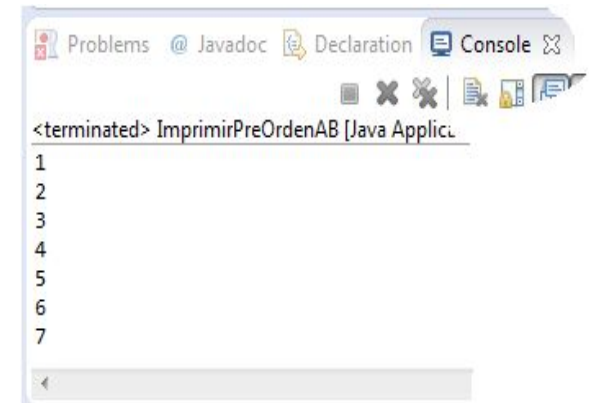
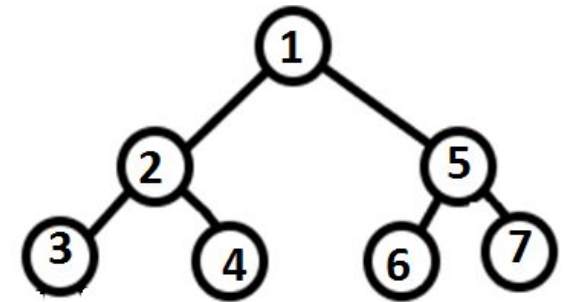


Arboles Binarios

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() {  
        System.out.println(this.getDato());  
        if (this.tieneHijoIzquierdo()) {  
            this.getHijoIzquierdo().printPreorden();  
        }  
        if (this.tieneHijoDerecho()) {  
            this.getHijoDerecho().printPreorden();  
        }  
    }  
}
```

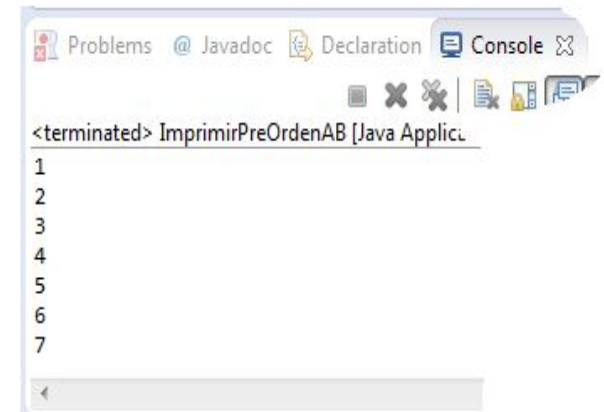
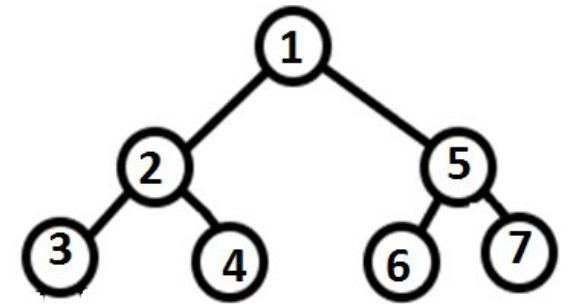


Arboles Binarios

Recorrido PreOrden

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

```
package tp04.ejercicio1;  
  
import tp03.ejercicio4.ListaEnlazadaGenerica;  
import tp03.ejercicio4.ListaGenerica;  
import tp04.ejercicio1.ArbolBinario;  
  
public class ArbolBinarioExamples<T> {  
  
    public void preorder(ArbolBinario<T> arbol) {  
        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?

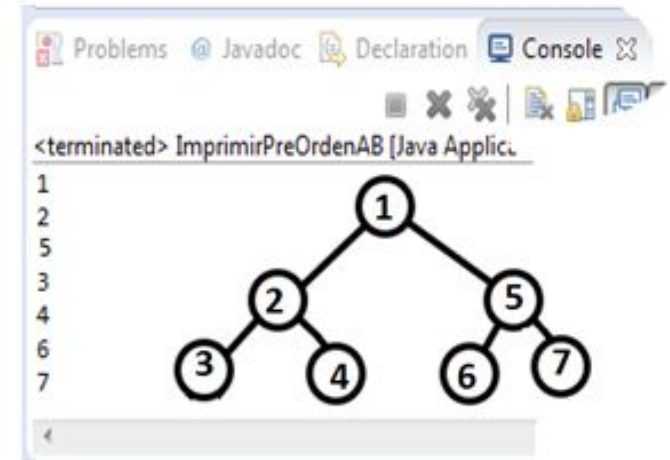
```
package tp04.ejercicio1;

import tp03.ejercicio4.ListaEnlazadaGenerica;
import tp03.ejercicio4.ListaGenerica;
import tp04.ejercicio1.ArbolBinario;

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) {
        result.agregarFinal(arbol.getDato());
        if (arbol.tieneHijoIzquierdo()) {
            this.preorder_private(arbol.getHijoIzquierdo(), result);
        }
        if (arbol.tieneHijoDerecho()) {
            this.preorder_private(arbol.getHijoDerecho(), result);
        }
    }
}
```

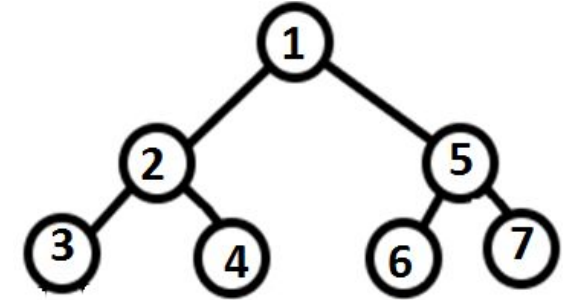


Arboles Binarios

Recorrido por Niveles

Recorrido por niveles implementado en la clase `ArbolBinario`

```
public class ArbolBinario<T> {  
    private T dato;  
    private ArbolBinario<T> hijoIzquierdo;  
    private ArbolBinario<T> hijoDerecho;  
    ...  
    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.getDato());  
                if (arbol.tieneHijoIzquierdo())  
                    cola.encolar(arbol.getHijoIzquierdo());  
                if (arbol.tieneHijoDerecho())  
                    cola.encolar(arbol.getHijoDerecho());  
            } else if (!cola.esVacia()) {  
                System.out.println();  
                cola.encolar(null);  
            }  
        }  
    }  
}
```

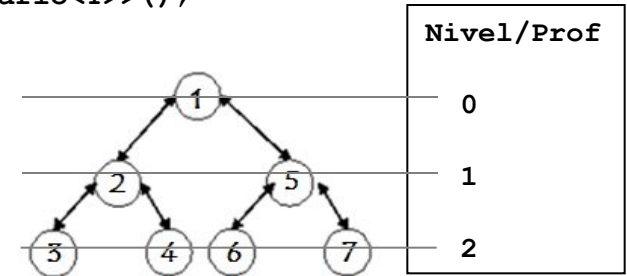


Arboles Binarios

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). Implementar un método para determinar 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();

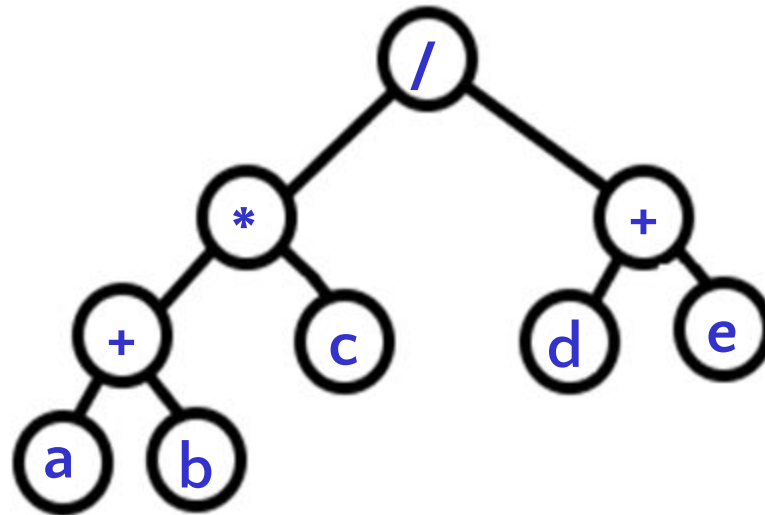
arbol = null
cant_nodos = 2

Arboles Binarios

Árbol 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



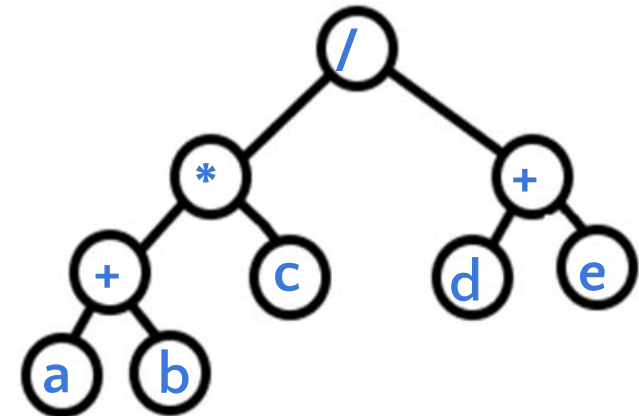
Arboles Binarios

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+ /



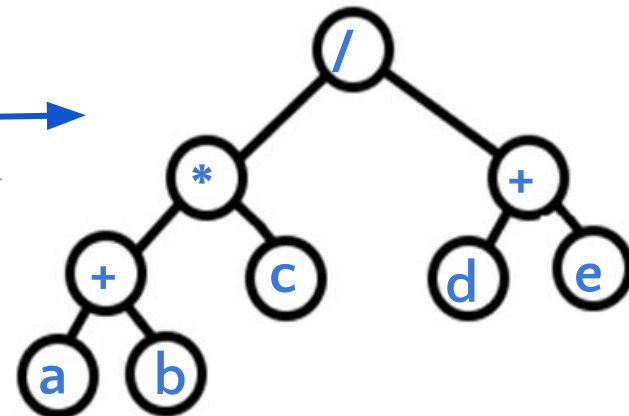
Arboles Binarios

Convertir expresión prefija en árbol 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 →



Arboles Binarios

Evaluación de un árbol de expresión

Este método evalúa y retorna un número de acuerdo a la expresión aritmética representada por el **ArbolBinario** que es enviado como parámetro.

```
public Integer evaluar(ArbolBinario<Character> arbol) {  
    Character c = arbol.getDato();  
    if ((c == '+') || (c == '-') || (c == '/') || c == '*') {  
        // es operador  
        int operador_1 = evaluar(arbol.getHijoIzquierdo());  
        int operador_2 = evaluar(arbol.getHijoDerecho());  
        switch (c) {  
            case '+':  
                return operador_1 + operador_2;  
            case '-':  
                return operador_1 - operador_2;  
            case '*':  
                return operador_1 * operador_2;  
            case '/':  
                return operador_1 / operador_2;  
        }  
    }  
    // es operando  
    return Integer.parseInt(c.toString());  
}
```

