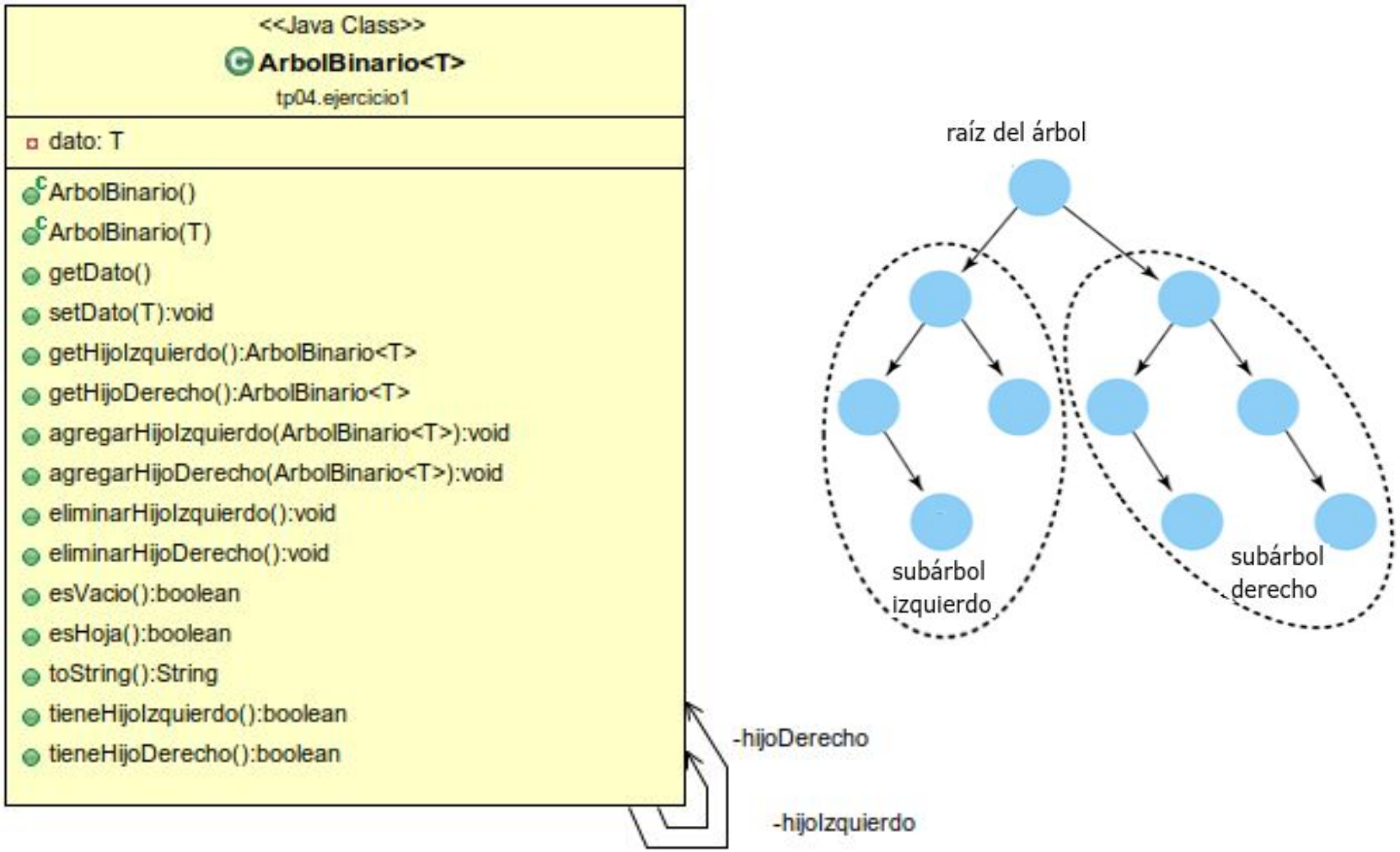
Arboles binarios en Java y árboles de expresión

Arboles Binarios

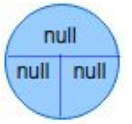
Estructura



Arboles Binarios

```
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

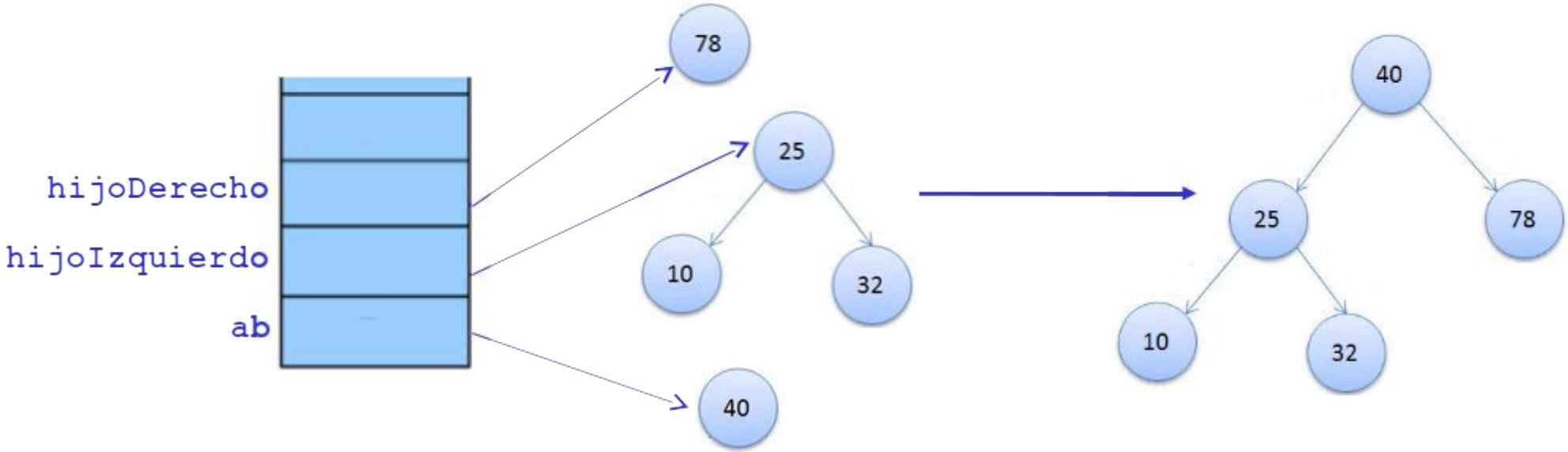


Arbol vacío

Arboles Binarios

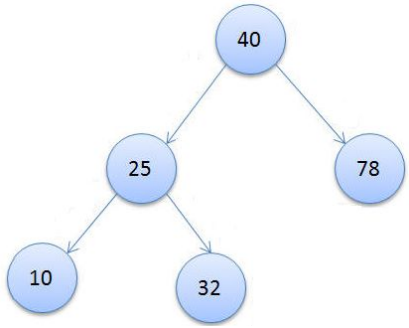
Creación

```
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

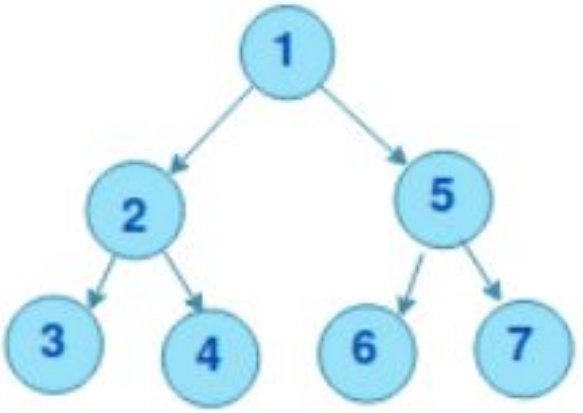
Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc.

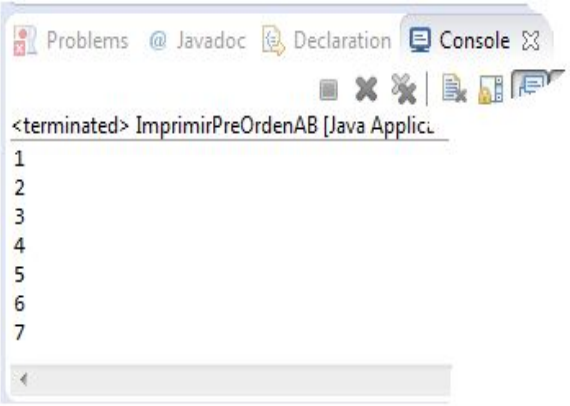
```
40, 25, 78, 10, 32
```

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

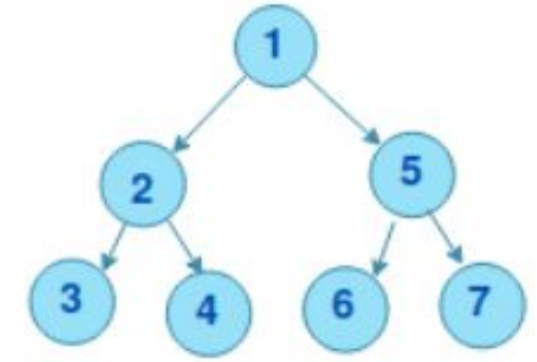


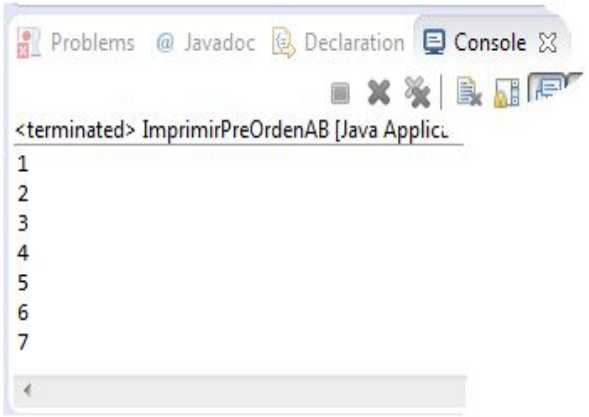


Arboles Binarios
Recorrido PreOrden

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

```
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());
        }

    }
```

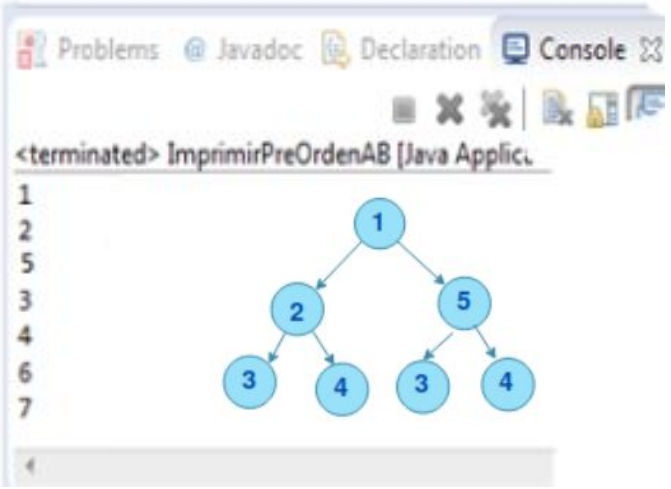




Arboles Binarios
Recorrido PreOrden

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

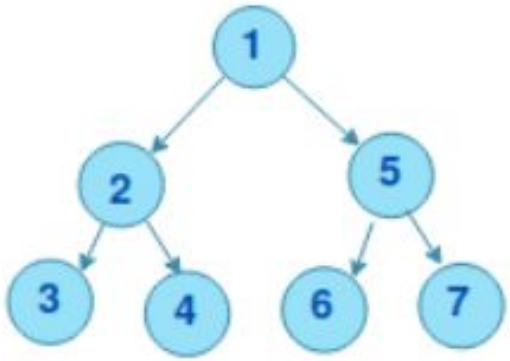
```java
package tp04.ejercicio1;

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`**



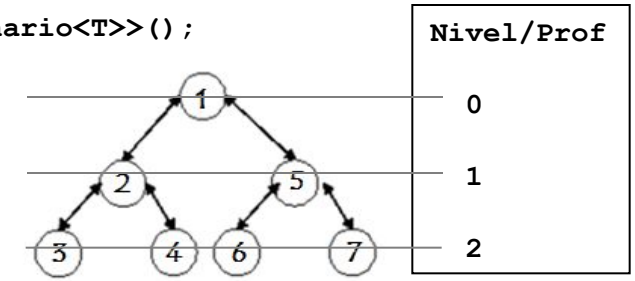

```
public class ArbolBinario<T> {
    private T dato;
    private ArbolBinario<T> hijoIzquierdo;
    private ArbolBinario<T> hijoDerecho;
     . . .
    public void recorridoPorNiveles() {
        ArbolBinario<T> arbol = null;
        ColaGenerica<ArbolBinario<T>> cola = new ColaGenerica<ArbolBinario<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);
            }
        }
    }
}
```

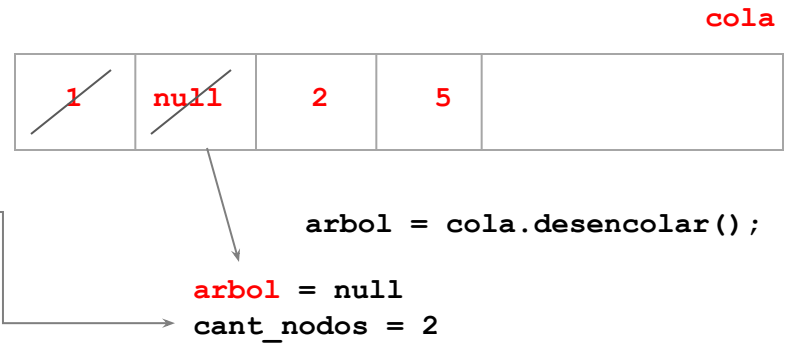
Arboles Binarios

Es árbol lleno?

Dado un árbol binario de altura h, diremos que es un **árbol 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;
}
```



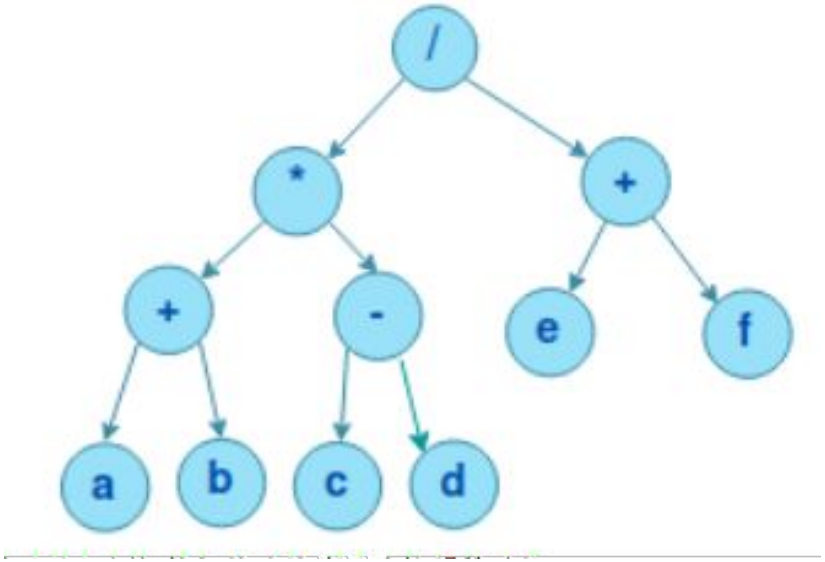


Arboles Binarios

Arboles de Expresión

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

- Los nodos internos representan operadores
- Los nodos externos (hojas) representan operandos

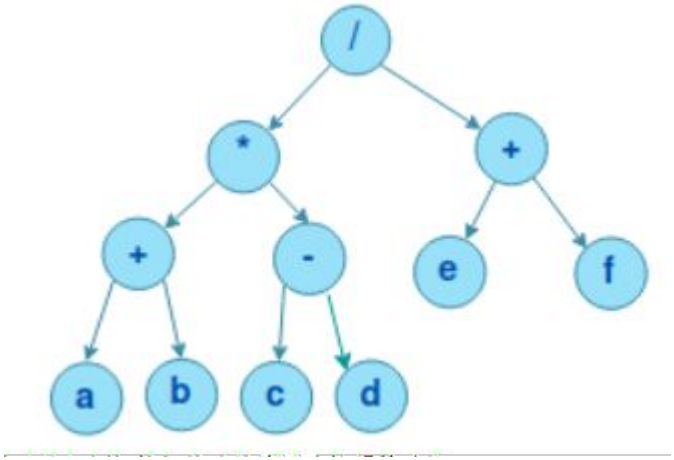


No necesitan el uso de paréntesis

Arboles de Expresión
Casos de uso

Algunas aplicaciones de los árboles de expresión son:

- En compiladores se usa para analizar, optimizar y traducir programas.
- Evaluar expresiones algebraicas o lógicas complejas de manera eficiente
- Los árboles pueden almacenar expresiones algebraicas y a partir de ellos se puede generar notaciones sufijas, prefijas e infijas.



Recorridos

Inorden: (((a + b) * (c – d)) / (e + f)) → expresión infija

Preorden: /*+ab-cd+ef → expresión prefija

Postorden: ab+cd-*ef+/ → expresión posfija

Arboles de expresión

Construcción de un árbol de expresión a partir de una expresión posfija

Estretegia:

crear una Pila vacía

mientras (existe un carácter) hacer

tomo un carácter de la expresión

si es un ***operando***

- ***creo*** *un nodo y lo apilo*

si es un ***operador*** *(lo tomo como la raíz de los dos últimos nodos creados)*

- ***creo*** *un nodo R con ese operador*

desapilo *y lo agrego como hijo derecho de R*

desapilo *y lo agrego como hijo izquierdo de R*

apilo *R.*

desapilar ***árbol de expresión posfija final***

Este proceso es posible ya que la expresión posfija está organizada en una forma en la que los operandos aparecen antes de los operadores. Esto nos permite construir el árbol de expresión utilizando una pila, donde se apilan operandos hasta que se encuentre un nodo operador que tome los dos últimos nodos de la pila como hijos.

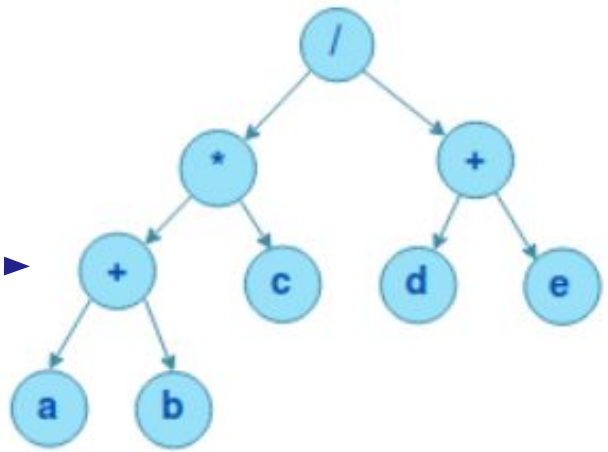
Arboles de expresión

Construcción de un árbol de expresión a partir de una expresión posfija

Este método convierte una expresión ***postfija*** 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 de expresión

Construcción de un árbol de expresión a partir de una expresión prefija

***Estretegia**:*

convertir(expr_prefija)

tomo primer carácter de la expresión

***creo** un nodo R con ese operador*

*si el caracter es un **operador***

- *agrego como hijo derecho de R(**convertir(expr_prefija sin 1 carácter())***
- *agrego como hijo izquierdo de R(**convertir(expr_prefija sin 1 carácter))***

//es un operador

devuelvo el nodo R

Este proceso es posible ya que la expresión posfija está organizada en una forma en la que los operadores siempre aparecen antes de los operandos. Cuando se llega a las hojas, la recursión retorna y permite ir armando el arbol desde abajo hacia arriba.

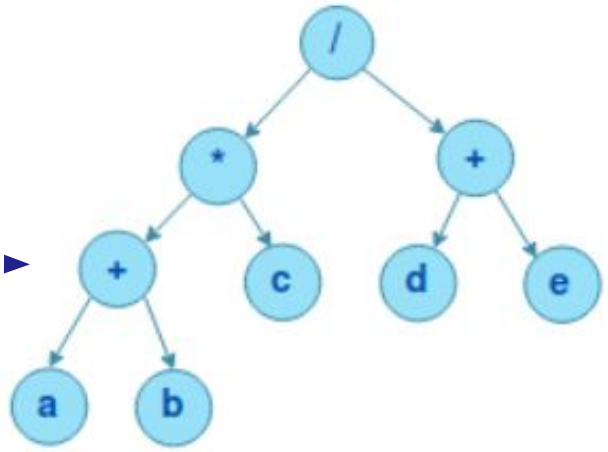
Arboles de expresión

Construcción de un árbol de expresión a partir de una expresión prefija

Este método convierte una expresión ***prefija*** en un árbol de expresión. 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 de expresión

Construcción de un árbol de expresión a partir de una expresión infija

La estrategia para crear un árbol de expresión a partir de una expresión ***infija*** es un poco más compleja. Primero se debe convertir a una expresión ***posfija***.

Expresión infija

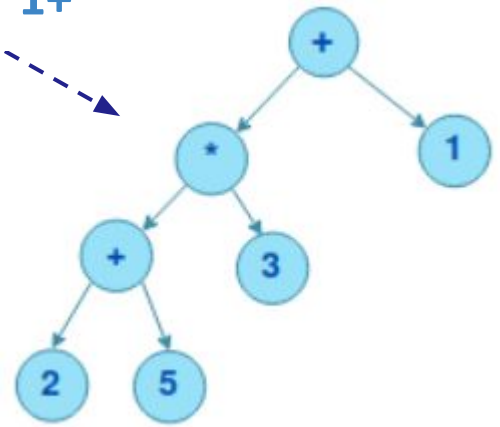
(i) *Se usa una pila y se tiene en cuenta la precedencia de los operadores*

(2+5)*3+1

Expresión postfija

(ii) *Se usa la estrategia 1*

25+3*1+



Prioridades
() [] {}
^
*, /
+, -

Arboles de expresión

Construcción de un árbol de expresión a partir de una expresión prefija

Este método convierte una expresión ***infija*** en una expresión ***posfija***. Luego se aplica el algoritmo iterativo visto anteriormente.

crear una Pila vacía
mientras (existe un carácter) hacer
tomo un carácter de la expresión
si es un ***operando*** □ *coloca en la salida*
si es un ***operador*** □ *se analiza su prioridad respecto del topo de la pila:*
si es un "(" , ")"□
"(" se apila
")" se desapila todo hasta el "(", incluído éste
sino
operador con **>** *prioridad que el tope –> se apila*
operador con **<=** *prioridad que el tope –> se desapila, se manda a la salida*
y se vuelve a comparar el operador con el tope de la pila
//se terminó de procesar la expresión infija
Se desapilan todos los elementos llevándolos a la salida, hasta que la pila quede vacía.

(2+5)*3+(10/5)

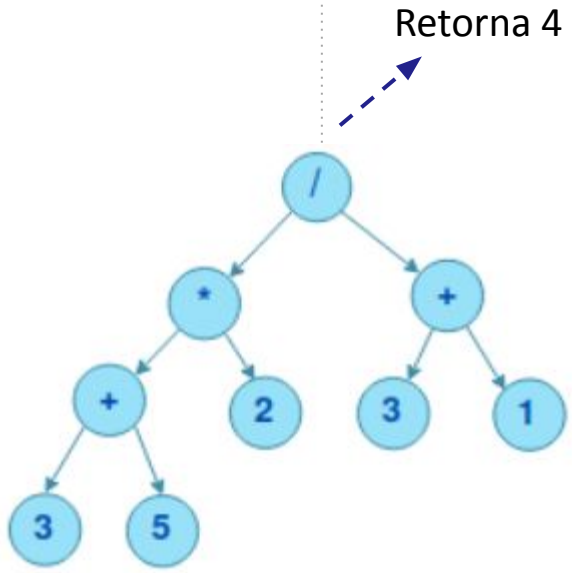
2 5+3*10 5 /+

Arboles de expresión

Evaluación

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());
}
```



Arboles de expresión

Ejercitación

Ejercicio 1

Dada la siguiente expresión postfija: $I\ J\ K\ +\ +\ A\ B\ *\ C\ -\ *$ *, dibuje su correspondiente árbol binario de expresión*

Convierta la expresión: $((a + b) + c * (d + e) + f) * (g + h)$ *en expresión prefija*

Ejercicio 2

Dada la siguiente expresión prefija: $*\ +\ I\ +\ J\ K\ -\ C\ *\ A\ B$ *, dibuje su correspondiente árbol binario de expresión*

Convierta la expresión: $((a + b) + c * (d + e) + f) * (g + h)$ *en expresión postfija*