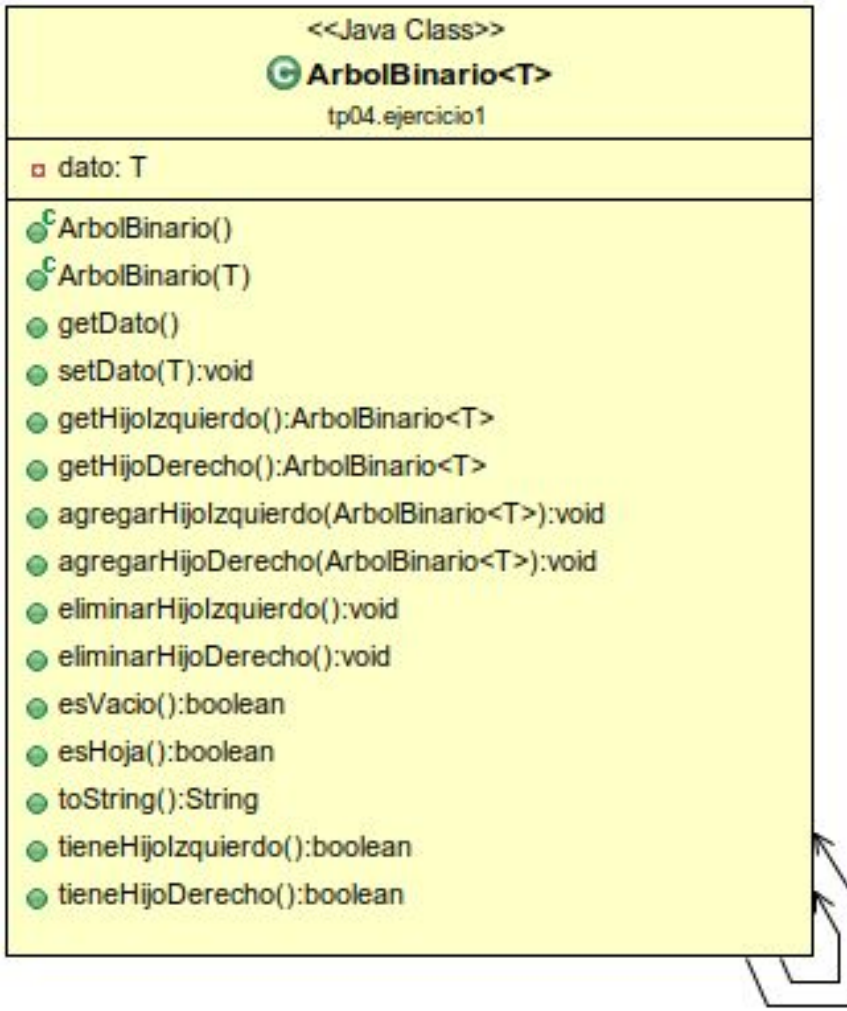
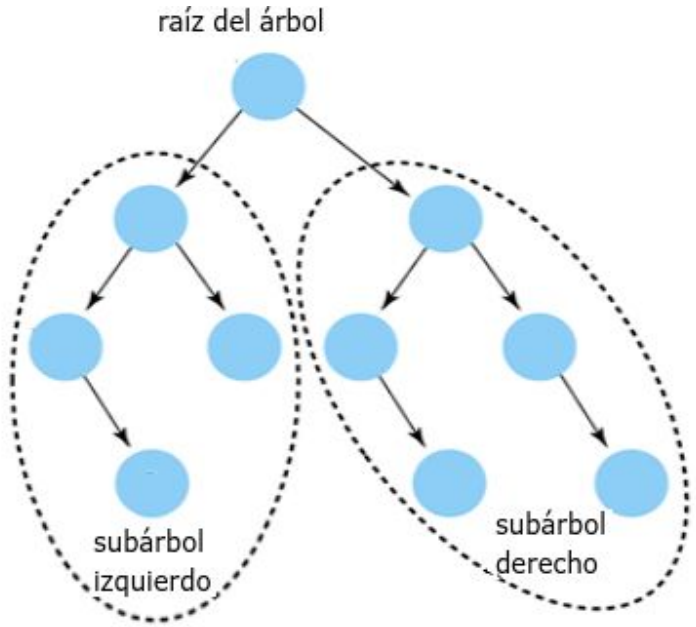
Arboles Binarios

Estructura



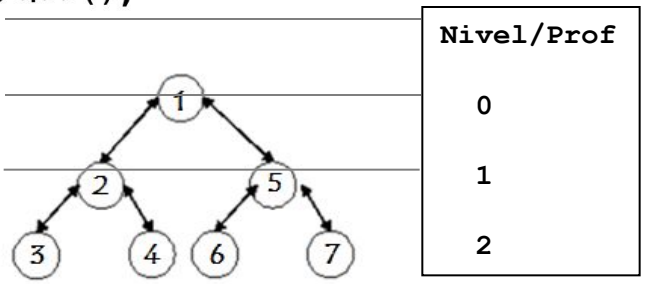


Arboles Binarios

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	

Arboles Binarios

Es árbol lleno? Recursivo

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 llenoRecur() {
    ResultadoLLeno result = new ResultadoLLeno();
    this.llenoRecurPrivate(result, 0);
    return result.getLleno();
}

private void llenoRecurPrivate(ResultadoLLeno result, int nivel) {
    if (this.esVacio()) result.setLLeno(false);
    else {
        if (this.esHoja()){
            if(result.encontreHoja()) {
                if(result.getNivelPrimerHoja()!= nivel)
                    result.setLLeno(false);
            }else {
                result.setEncontrePrimerHoja();
                result.setNivelPrimerHoja(nivel);
            }
        }
        if(this.tieneHijoIzquierdo() && !this.tieneHijoDerecho()) result.setLLeno(false);
        if(!this.tieneHijoIzquierdo() && this.tieneHijoDerecho()) result.setLLeno(false);

        if(result.getLleno()) {
            if (this.tieneHijoIzquierdo())
             this.getHijoIzquierdo().llenoRecurPrivate(result, nivel +1);
            if(result.getLleno() && this.tieneHijoDerecho())
             this.getHijoDerecho().llenoRecurPrivate(result, nivel +1);
        }
    }

}
```

Arboles Binarios

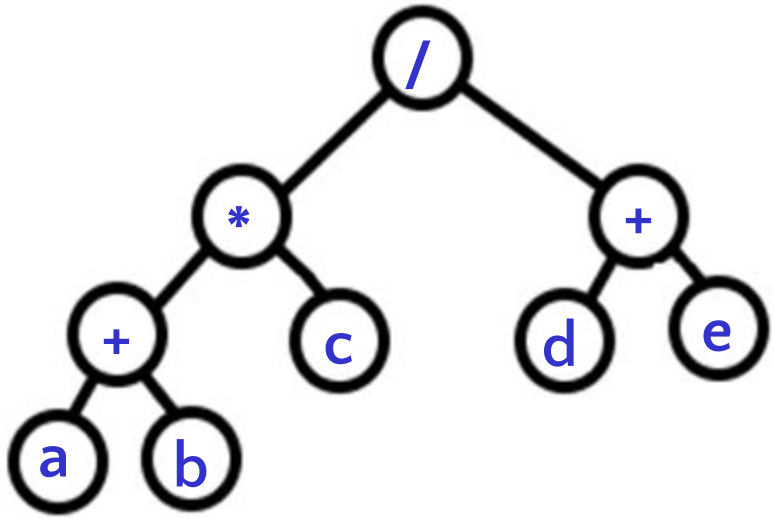
Es árbol lleno? Recursivo

```
public class ResultadoLLeno {
    boolean lleno = true;
    boolean encontre_primer_hoja = false;
    int nivel_primer_hoja;

    public boolean getLleno() {
        return this.lleno;
    }
    public void setLLeno(boolean b) {
        this.lleno = b;

    }
    public boolean encontreHoja() {
        return this.encontre_primer_hoja;
    }
    public void setEncontrePrimerHoja() {
        this.encontre_primer_hoja = true;

    }
    public void setNivelPrimerHoja(int nivel) {
        this.nivel_primer_hoja = nivel;

    }
    public int getNivelPrimerHoja() {
        return this.nivel_primer_hoja;
    }

}
```

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



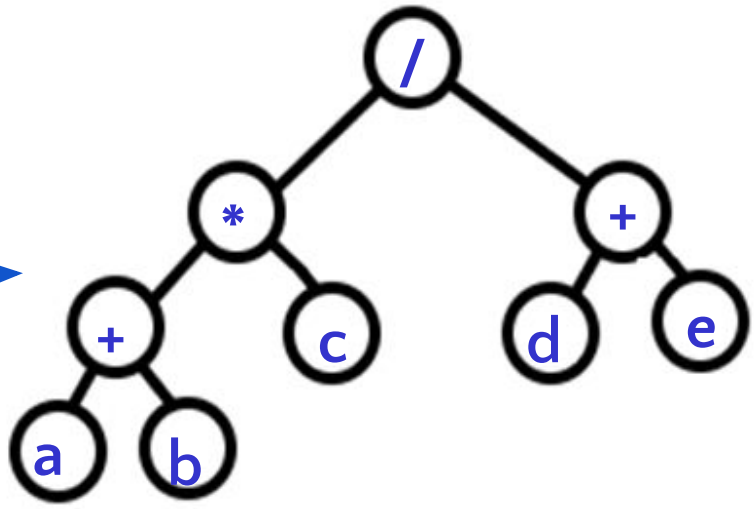
Arboles Binarios

Convertir expresión prefija en árbol de expresión

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

```
public ArbolBinario<Character> convertirPrefija(StringBuffer exp) {
  char 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

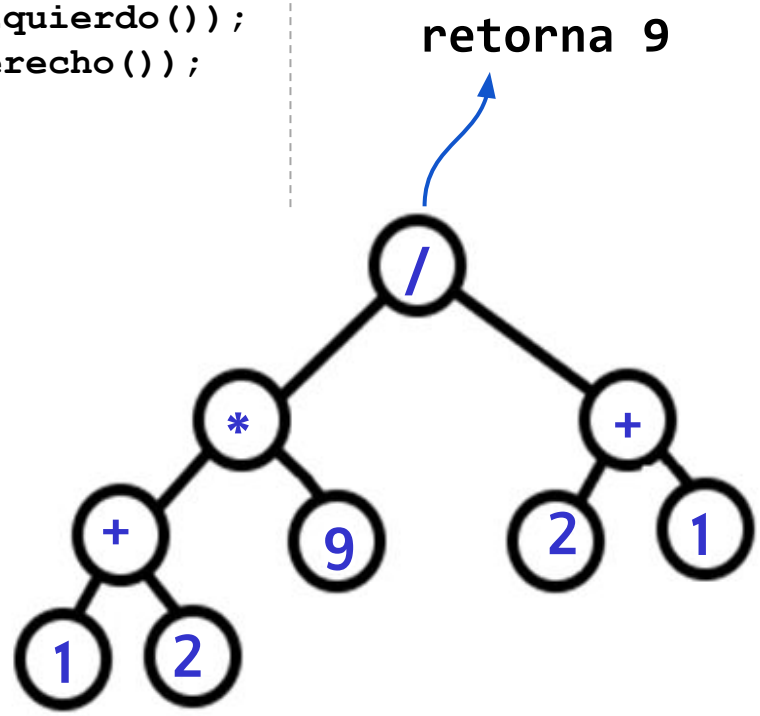


Arboles Binarios

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();
  // es operador
  if ((c == '+') || (c == '-') || (c == '/') || c == '*') {
          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());
}
```



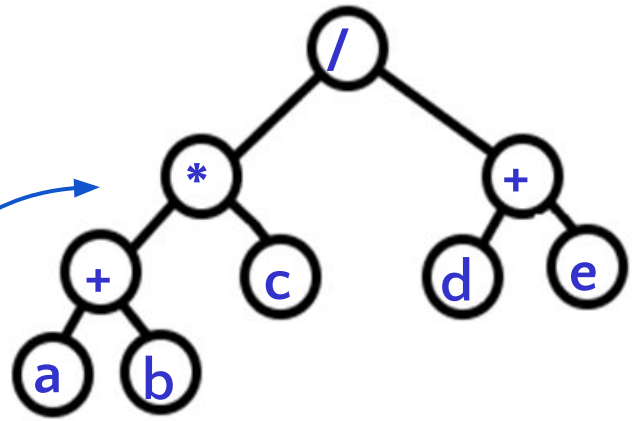
Arboles Binarios

Convertir expresión posfija en árbol de Expresión v1

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>>();
 if (exp.length <=0) return new ArbolBinario();
 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 posfija en árbol de Expresión v2

Este método convierte una expresión ***postfija*** en un **`ArbolBinario`** de manera recursiva. Puede estar implementado en cualquier clase.

```
public ArbolBinario<Character> construirDesdePosfija(StringBuffer exp){
    ArbolBinario<Character> arbol = new ArbolBinario<Character>();
    if (exp.length() > 0) {
        char c = exp.charAt(exp.length()-1);
        arbol.setDato(c);
        if(c== '+' || c =='*' || c == '/' || c == '-') {
            arbol.agregarHijoDerecho(this.construirDesdePrefija(exp.deleteCharAt(exp.length()-1)));
            arbol.agregarHijoIzquierdo(this.construirDesdePrefija(exp.deleteCharAt(exp.length()-1)));
        }
    }
    return arbol;
}
```

ab+c*de+/