# Parsers Recursivos Descendentes

Um analisador sintático converte texto para uma árvore.

Precisamos de uma estrutura de dados para a árvore.

Temos duas opções:

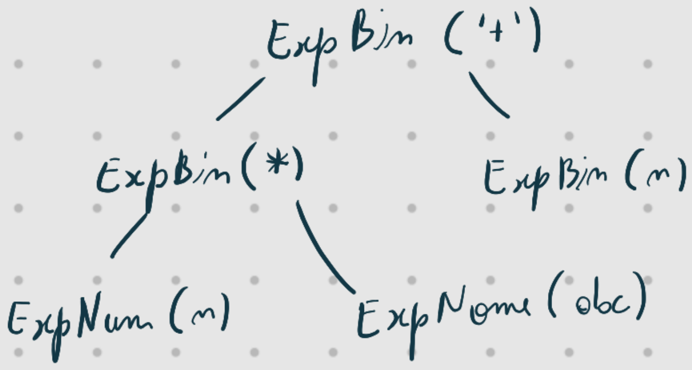
→ Árvore concreta

→ Árvore abstrata

Uma árvore concreta representa exatamente a gramática:

* * *

Uma árvore abstrata só representa as informações importantes.



Em 'Java':

```
enum OP = { OP_PLUS, OP_MINUS }

class Exp { }

class ExpBin {
    OP op;
    Exp e1;
    Exp e2;
}
```

```
class ExpNum {
    int num;
}
```

Em 'python':

```
class Exp
class ExpBin
class ExpNum
```

Em 'Lua':

```
function ExpBin(op, e1, e2):
   return {
      tag = 'ExpBin',
      op = op,
      e1 = e1
      e2 = e2
   }
end
```

## Exercícios

árvore de:

- 10 => ExpNum(10)
- 5+x => ExpBin('+', ExpNum(5), ExpNome('x'))
- (2-3)*4 => ExpBin('*',
  ExpBin('-',
  ExpNum(2),
  ExpNum(3)),
  ExpNum(4))

Vamos construir a árvore x a gramática é:

$E \to n$
$E \to$ '(' $E$ $Op$ $E$ ')'
$Op \to + / - / * / \div$

```
char peak();        {  Exp parseExp() {
void avança();      {
```