# Aula Prática 13

**Resumo:**
- Dicionários;
- Árvores binárias de procura na implementação de dicionários;
- Árvores Binárias.

**Exercício 13.1**

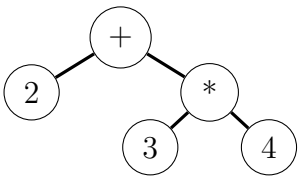
Implemente o tipo de dados abstrato de um dicionário na classe `BinarySearchTree`, o qual se deve basear numa árvore binária de procura. Experimente a classe com os exercícios da aula anterior ou usando o programa `TestBST` fornecido.

**Exercício 13.2**

Podemos escrever expressões aritméticas de três formas distintas, consoante a posição do operador:[1]

- *Notação infixa*: `2 + 3` ou `2 + 3 * 4` ou `3 * (2 + 1) + (2 - 1)`
- *Notação prefixa*: `+ 2 3` ou `+ 2 * 3 4` ou `+ * 3 + 2 1 - 2 1`
- *Notação sufixa*: `2 3 +` ou `2 3 4 * +` ou `3 2 1 + * 2 1 - +`

Independentemente da notação utilizada, uma expressão aritmética pode ser representada por uma árvore binária em que os nós representam as (sub)expressões existentes. Os números são expressões constantes e, portanto, serão as folhas da árvore. Por exemplo, a expressão (prefixa) `+ 2 * 3 4` é expressa pela árvore binária:




[1] No exercício 10.4 já utilizámos a notação sufixa como forma de simplificar o cálculo de expressões.

A geração desta árvore binária a partir de uma expressão em notação *prefixa* é bastante simples (quando feita recursivamente):

```
createPrefix() {
   if (in.hasNextDouble()) { // next word is a number
      // leaf tree with the number
   }
   else { // next word is the operator
      // tree with the form: operator leftExpression rightExpression
      // leftExpression and rightExpression can also be created with createPrefix
   }
}
```

a. Implemente uma classe – `ExpressionTree` – que cria uma árvore binária a partir de uma expressão em notação *prefixa* para os 4 operadores aritméticos elementares (`+`, `-`, `*` e `/`). Considere que cada número e operador é uma palavra a ser lida no *standard input*.

b. Implemente um novo serviço na classe – `printInfix` – que escreve a expressão (já lida) na notação *infixa*. Execute o programa `ParsePrefixExpression.jar` para ver o formato desejado.

c. Implemente a classe de uma forma robusta por forma a detectar expressões inválidas (note que a responsabilidade por uma expressão inválida é exterior ao programa pelo que deve fazer uso de uma abordagem defensiva).

d. Implemente um novo serviço na classe – `eval` – que calcula o valor da expressão.

**Nota**: A criação da árvore binária com base numa expressão em notação *sufixa* é um pouco mais complexa. Se tiver essa curiosidade pode tentar fazer essa implementação.