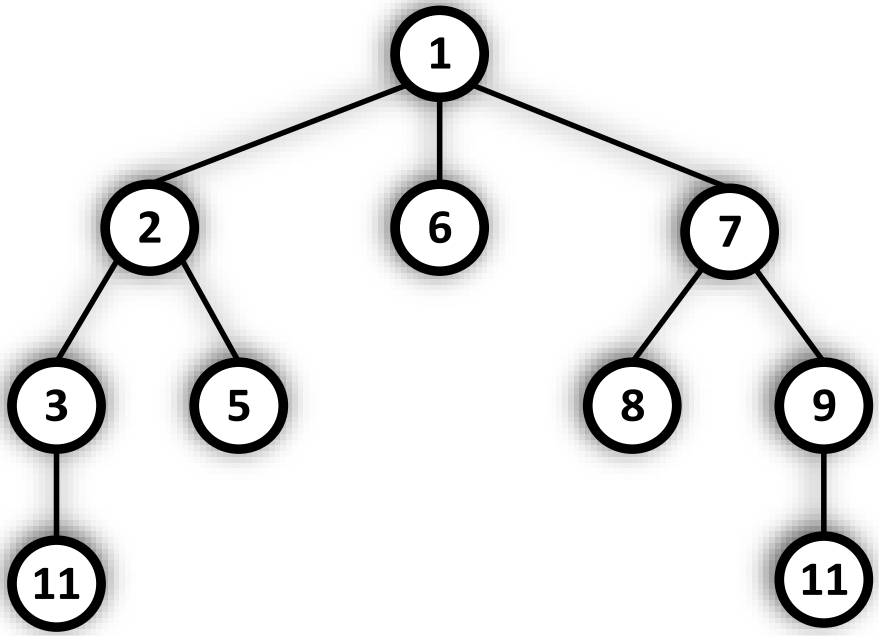
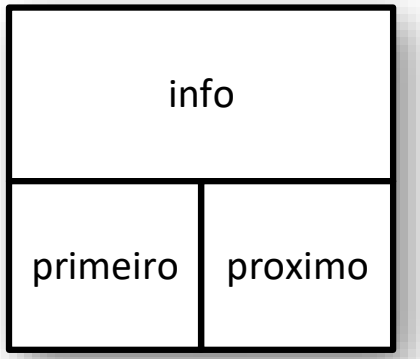
Árvores com número variável de filhos

Árvore com número variável de filhos

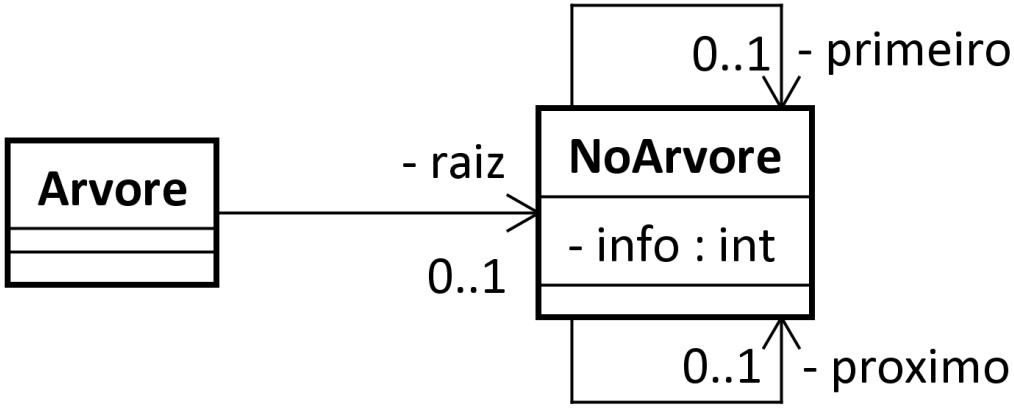
- Cada nó pode ter mais do que duas sub-árvores associadas
- As sub-árvores de um nó são dispostas em ordem: primeira sub-árvore (*sa1*), segunda sub-árvore (*sa2*), etc...



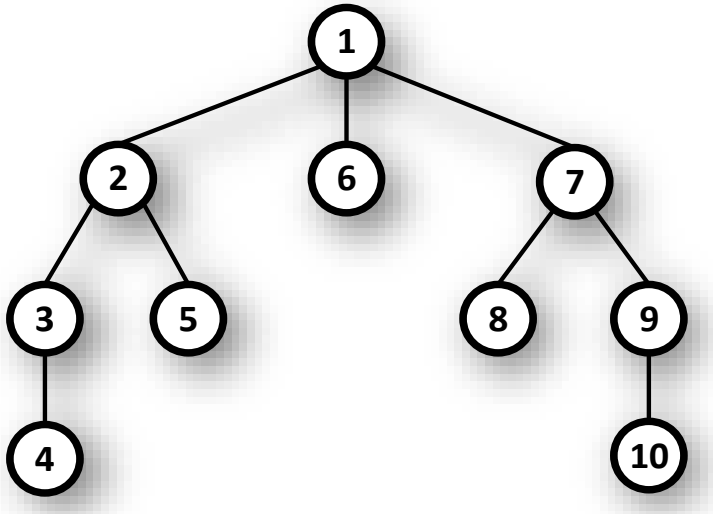
Implementação

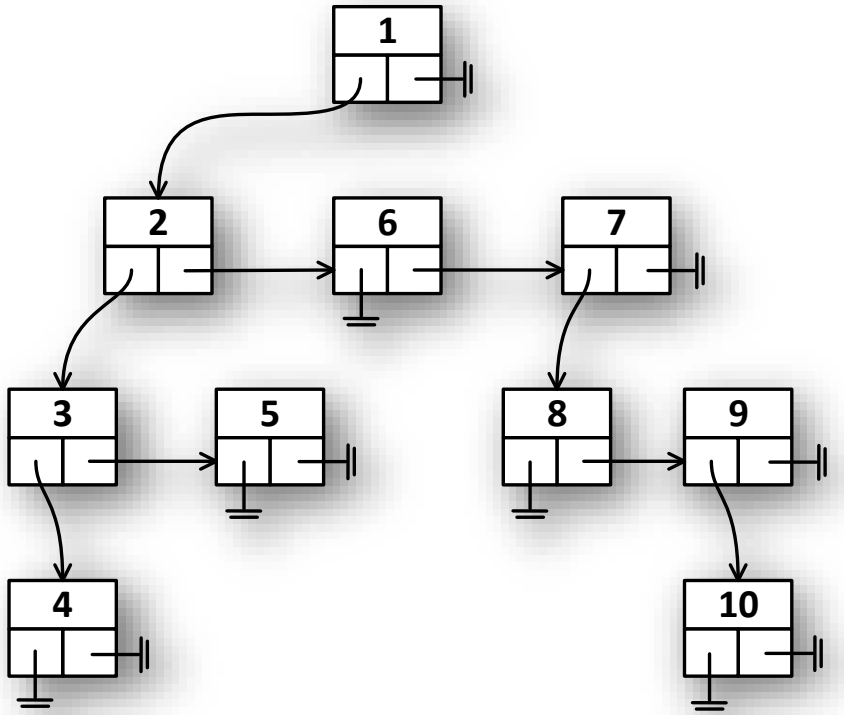


- Um nó aponta apenas para seu primeiro filho (`primeiro`)
- Cada um dos filhos aponta para o próximo "irmão" (`proximo`)



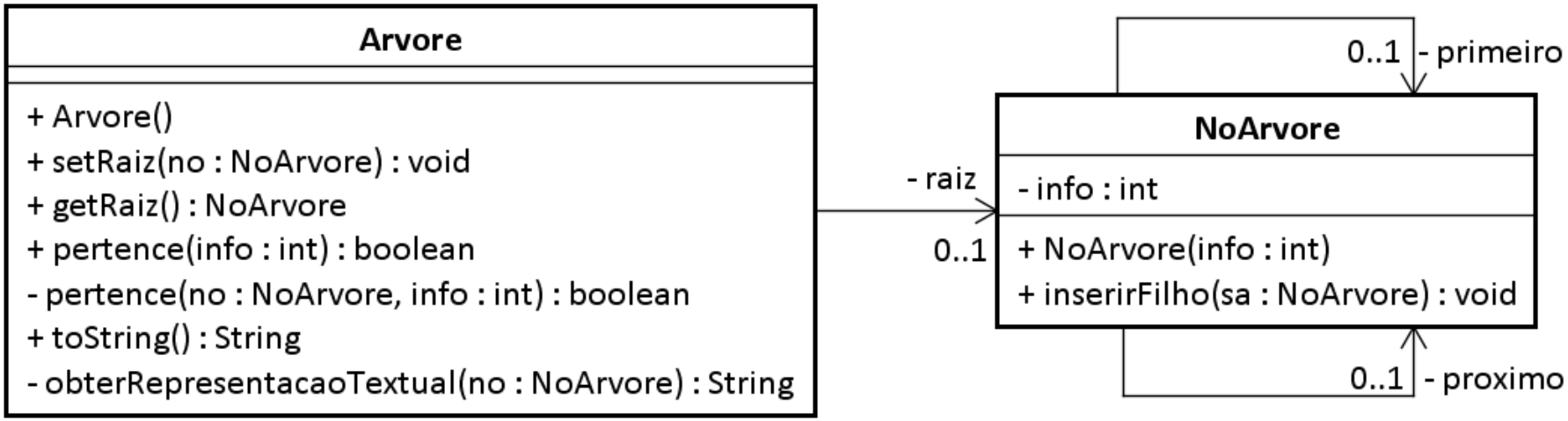
Representação (modelagem)





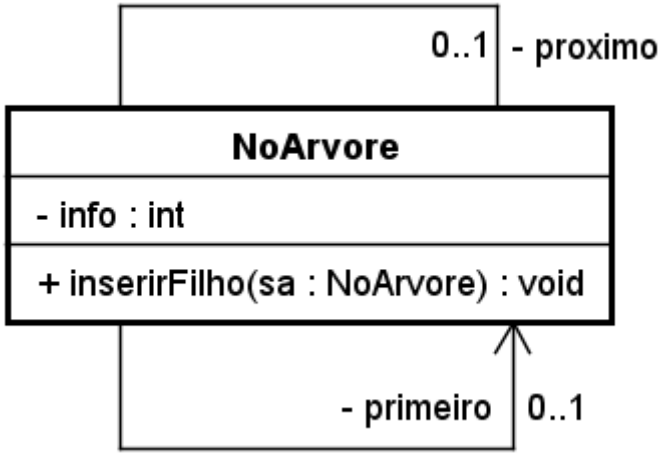
Cada nó tem uma “lista de filhos”

Implementação



Implementação

- Objeto da classe NoArvore: representa um nó da árvore

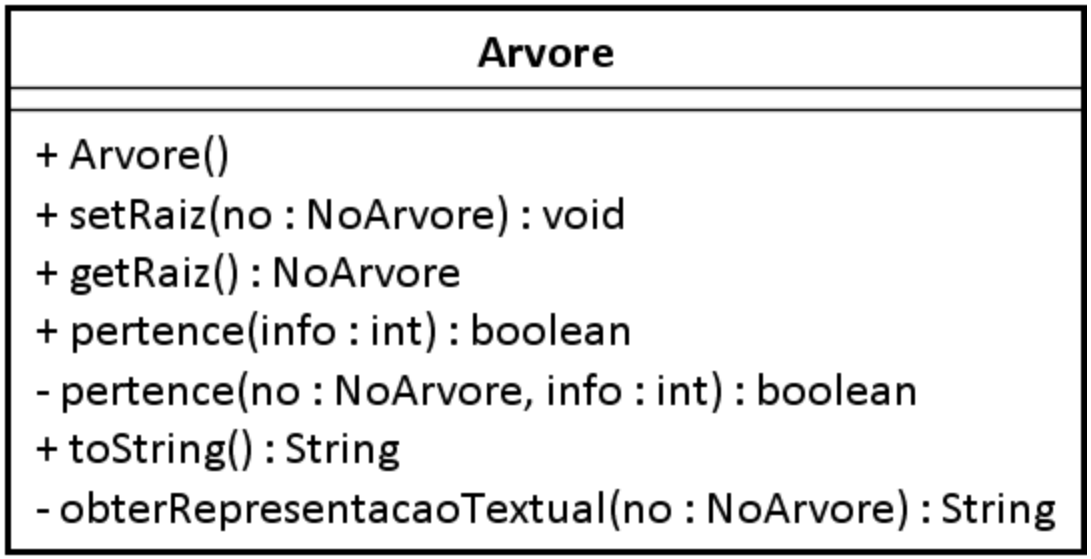


- `primeiro`: ponteiro para a primeira sub-árvore filha (`null` se o nó for uma folha)
- `proximo`: ponteiro para o próximo irmão (isto é, a raiz da próxima sub-árvore irmã). Possui `null` quando for o último irmão

Implementação

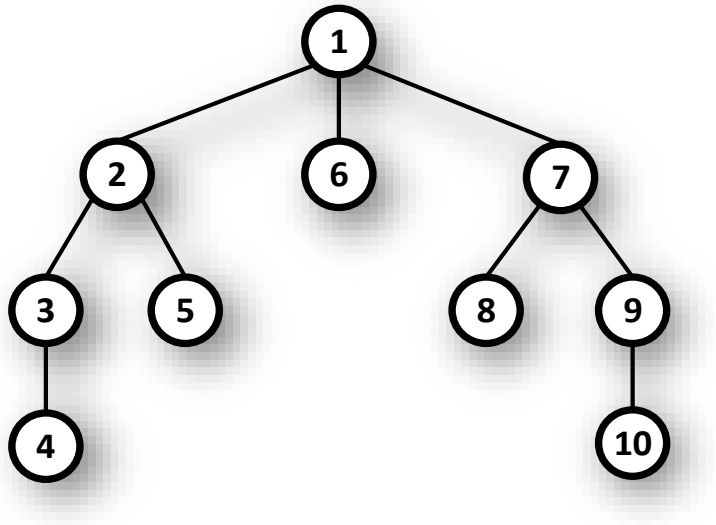
```
public class NoArvore {

        private int info;
        private NoArvore primeiro; // primeiro filho
        private NoArvore proximo; // próximo irmão

        public NoArvore(int info) {
                this.info = info;
        }

        // métodos setter/getter...
}
```

Classe da Árvore



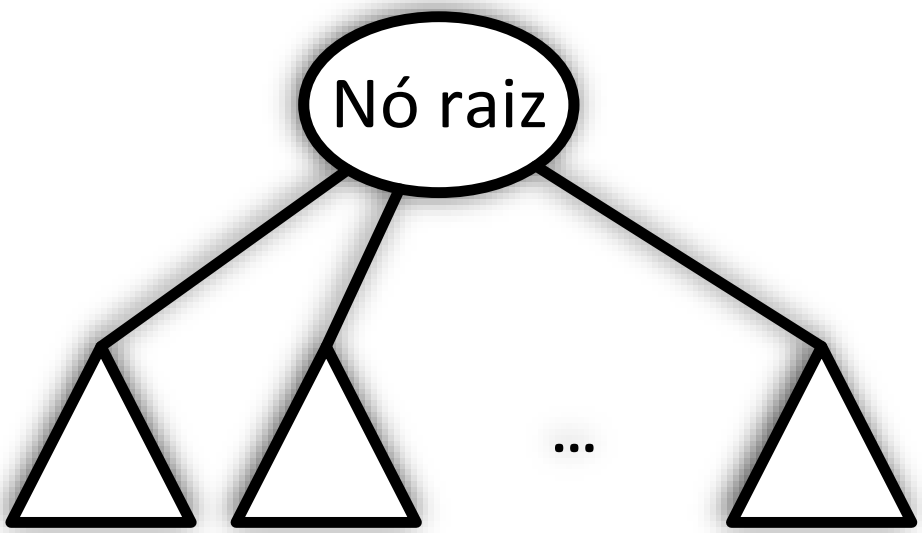
Exemplo

```
 1      NoArvore<Integer> no4 = new NoArvore<>(4);
 2      NoArvore<Integer> no3 = new NoArvore<>(3);
 3      no3.inserirFilho(no4);
 4
 5      NoArvore<Integer> no5 = new NoArvore<>(5);
 6
 7      NoArvore<Integer> no2 = new NoArvore<>(2);
 8      no2.inserirFilho(no5);
 9      no2.inserirFilho(no3);
10
11      NoArvore<Integer> no10 = new NoArvore<>(10);
12      NoArvore<Integer> no9  = new NoArvore<>(9);
13      no9.inserirFilho(no10);
14
15      NoArvore<Integer> no8 = new NoArvore<>(8);
16
17      NoArvore<Integer> no7 = new NoArvore<>(7);
18      no7.inserirFilho(no9);
19      no7.inserirFilho(no8);
20
21      NoArvore<Integer> no6 = new NoArvore<>(6);
22
23      NoArvore<Integer> no1 = new NoArvore<>(1);
24      no1.inserirFilho(no7);
25      no1.inserirFilho(no6);
26      no1.inserirFilho(no2);
27
28      Arvore<Integer> arvore = new Arvore<>();
29      arvore.setRaiz(no1);
30      System.out.println(arvore);
```



Implementação

- A implementação utiliza uma visão recursiva, onde:
  - Uma árvore é um nó raiz contendo zero ou mais sub-árvores;



Implementação

- Considerações para os métodos recursivos:
  - Uma árvore não pode ser vazia
    - Os métodos recursivos não consideram o caso de árvores vazias, mas os métodos públicos sim.
  - Uma folha é identificada como um nó com zero sub-árvores
    - Uma folha não é um nó com sub-árvores vazias como nas árvores binárias

Criação de um nó

- Cria um nó sem filhos ou “irmãos”

```
Algoritmo: NoArvore(int info)

this.info ← info;
primeiro ← null;
proximo ← null;
```

Incluir uma sub-árvore para um nó

- Dado um nó já criado, definir uma sub-árvore para este nó

```
Algoritmo: inserirFilho (NoArvore sa)

sa.proximo ← primeiro;
primeiro  ← sa;
```

Pesquisar na árvore

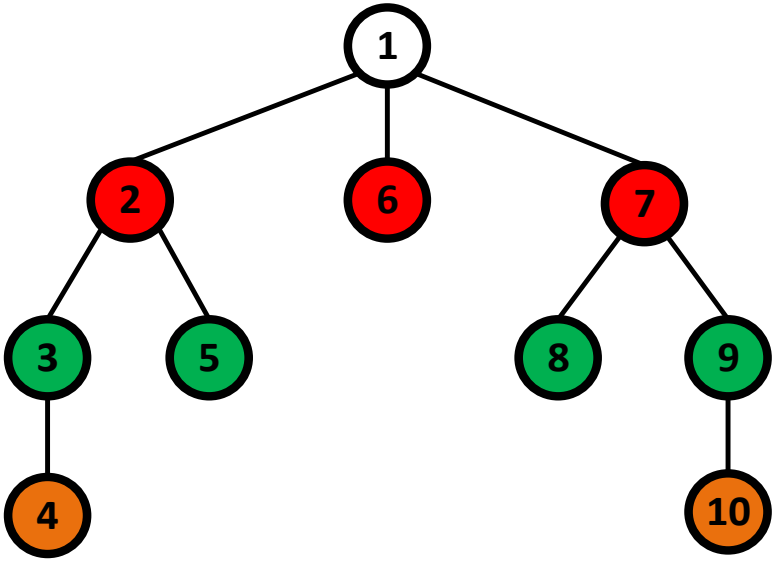
```
Algoritmo: pertence(int info)

se (raiz = null) então
   retornar falso;
senão
   retornar pertence(raiz, info);
fim-se;
```

```
Algoritmo: pertence(NoArvore no, int info)

se (no.info = info) então
   retornar verdadeiro;
senão
   NoArvore p;
   p ← no.primeiro;
   enquanto (p ≠ null) faça
      se (pertence(p, info) então
         retornar verdadeiro;
      fim-se;
      p ← p.proximo;
   fim-enquanto;

   retornar falso;
fim-se;
```

Notação textual de árvore com número variável de filhos

- Notação textual: **<raiz sa1 sa2 ... san>**
- Exemplo:

α = <1 <2 <3 <4> > <5> > <6> <7 <8> <9 <10> > > >



Exibir o conteúdo da árvore

- O algoritmo abaixo percorre a árvore da raiz até as folhas, navegando nas sub-árvores filhas da esquerda para a direita

```
Algoritmo: toString()

se (raiz = null) então
   retornar " ";
senão
   retornar obterRepresentacaoTextual (raiz);
fim-se;
```

```
Algoritmo: obterRepresentacaoTextual(NoArvore no)
s ← "<";
s ← s + no.info;

p ← no.primeiro;
enquanto (p ≠ null) faça
  s ←  s + obterRepresentacaoTextual(p);
  p ←  p.proximo;
fim-enquanto;
s ← s + ">";
retornar s;
```