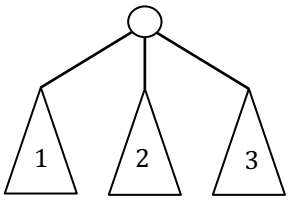
Pattern: Accumulating information about a tree

Assume that a node of a tree contains a value of type **int** and a set of children. The set is empty if the node is a leaf. We make the fields public only to simplify the code as we write the method. Generally, the fields would not be public.

```
/** An instance is node of a tree. */
public class Node {
   public int val;  // the value in the node
   public Set<Node> children; // children
}
```

We develop a method that can serve as a model for recursive methods that accumulate information about a tree. Our method will count the leaves in a tree, but there are many other possibilities. For example, we could count the nodes, the nodes with a value less than 10, the nodes with only one child, or the leaves at a certain depth.



To the right, we show a tree with three children, numbered 1, 2, 3. There could be fewer children or more.

The specification and header of the method appears to the right. Using it, we write the method body.

```
/** Return the number of leaves of tree t.
  * Precondition: t is not null. */
public static int leaves(Node t)
```

The precondition says that *t* is not null, so the first step is to take care of the base case: root *t* is a leaf:

```
if (t.children.size() == 0) return 1;
```

We now handle the recursive case. Here is a recursive definition, written in English.

> For a tree that contains more than one node:
> The number of leaves of the tree is
> the sum of the numbers of leaves of the children of the root.

This means that the children have to be processed. Actually, all we have to do is translate the recursive definition into Java! We need to calculate a sum, so we write:

```
int sum= 0;
for (Node ch : t.children) {
   // Add to sum the number of leaves of tree ch;
}
return sum;
```

Looking at the specification of method *leaves*, we implement the statement-comment using recursion:

```
sum= sum + leaves(ch);
```

The resulting method is in the textbox to the right.

```
/** Return the number of leaves of tree t.
  * Precondition: t is not null. */
public static int leaves(Node t) {
   if (t.children.size() = 0) return 1;
   int sum= 0;
   for (Node ch : t.children) {
      sum= sum + leaves(ch);
   }
   return sum;
}
```

The model, the design pattern

Use this method as a model for any function that counts the number of nodes with a certain property. There will be differences, of course. For example, the precondition might not be present, in which case the empty tree has to be handled.

Discussion

Some people start by automatically declaring local variable *sum* at the beginning. This is illogical; *don't do it*. The first step handles the base case, and for that, variable *sum* is not necessary.

```
/** Return the number of leaves of tree t.
  * Precondition: t is not null. */
public static int leaves(Node t) {
   int sum= 0; // DOES NOT BELONG HERE
   if (t.children.size() > 0) return 1;
   …
}
```