如代码清单 8-6 所示，作为末端节点的文件类只需要输出 space 个空格再加上自己的名称即可，这里与父类的展示方法 tree(int space) 应该保持一致，所以我们在第 14 行直接调用父类的展示方法。其实文件类可以不做任何修改，而是直接继承父类的展示方法，此处是为了让读者更清晰直观地看到这种继承关系，同时方便后续做出其他修改。接下来的文件夹类就比较特殊了，它不仅要先输出自己的名字，还要换行再逐个输出子节点的名字，并且要保证空格逐级递增，请参看代码清单 8-7。

代码清单 8-7 文件夹类 Folder

```
1.  public class Folder extends Node{
2.      //文件夹可以包含子节点（子文件夹或者文件）
3.      private List<Node> childrenNodes = new ArrayList<>();
4.
5.      public Folder(String name) {
6.          super(name);//调用父类“节点”的构造方法命名
7.      }
8.
9.      @Override
10.     protected void add(Node child) {
11.         childrenNodes.add(child);//可以添加子节点
12.     }
13.
14.     @Override
15.     public void tree(int space){
16.         super.tree(space);//调用父类通用的tree方法列出自己的名字
17.         space++;//在循环的子节点前，空格数要加1
18.         for (Node node : childrenNodes) {
19.             node.tree(space);//调用子节点的tree方法
20.         }
21.     }
22. }
```

如代码清单 8-7 所示，同样，文件夹类也重写并覆盖了父类的 tree() 方法，并且在第 16 行调用父类的通用 tree() 方法输出本文件夹的名字。接下来的逻辑就非常有意思了，对于下一级的子节点我们需要依次输出，但前提是要把当前的空格数加 1，如此一来子节点的位置会往右偏移一格，这样才能看起来像树形结构一样错落有致。可以看到，在第 19 行的循环体中我们直接调用了子节点的展示方法并把“加 1”后的空格数传递给它即可完成展示。至于当前文件夹下的子节点到底是“文件夹”还是“文件”，我们完全不必操心，因为子节点们会使用自己的展示逻辑。如果它们还有下一级子节点，则与此处逻辑相同，继续循环，把逐级递增的空格数传递下去，直至抵达叶节点为止——始于“文件夹”而终于“文件”，非常完美的递归逻辑。