The inside-out rule

Caveat about respecting privacy

We show you a general rule for determining the variable declaration associated with a reference to a variable. The issue is independent of whether the variable reference is actually legal. For example, in method `m` in this `Section` object, the reference to variable `title` is associated with field `title` inherited from superclass `Chapter`. But if field `title` is declared with access modifier **`private`**, the reference is illegal, and probably function `getTitle` should be called instead.

The inside-out rule for variables

Each variable name in a program refers to a variable that is declared somewhere within the program. Most programming languages use some form of an inside-out rule to determine the declaration to which a variable-name refers.

For example, the variable might be declared in the body of a loop in which the variable reference appears; if not there, in a surrounding if-statement; if not there, in the body of the method; if not there, it might be a parameter; if not there, it might be a field of the object in which the method is declared; and finally, if not there, it might be a static variable and thus in the file drawer.

Thus, the search is made in the inner scope and expands to increasingly surrounding scopes. That is,

> **Inside-out rule**: To find the declaration corresponding to the use of a variable, look in enclosing scopes from inside-out, until it is found.

In DrJava, we see a class that illustrates this. You see local variables declared in the loop body, the then-part of the conditional statement, and the method body. You see a parameter declaration. You see a field. And you see a static variable.

This inside-out rule, then is the basic mechanism for determining the declaration of a variable that a variable reference references. It is very simple.

The inside-out rule for method calls

The inside-out rule for method calls is similar. There are two differences. First, methods appear only in objects and in the file drawer, so the search is simplified.

For a method in an object, one need look only in that object and then in the surrounding file drawer. For example, method `toString` in object `a0` calls `getTitle`; its declaration is found in object `a0` itself. Method `toString` also calls `m` with an **`int`** argument. That method is not in `a0`, but it is found in the enclosing file drawer.

For a static method, which is in the file drawer, one can look only in the file drawer, because there are no enclosing scopes. For example, the method `m` with a **`boolean`** parameter calls method `m` with an **`int`** argument, and that method `m` is also in the file drawer.

Here is the other difference between searching for a variable and searching for a method. When searching for a method, a method is needed not with the right name but with the right signature —the name together with parameter types. So, in looking for a method declaration for the call m(number), a search is made for a method with a single parameter whose type is as least as wide as **int**.

Looking into a folder or a file drawer

Here are two file drawers, one for class `Chapter`, and the other for class `C`. By the inside-out rule, from the body of method `toString`, all methods and fields of `a0` as well as the static methods and variables in the file drawer can be referenced. From the body of the two methods `m`, the static methods and variables in file drawer `Chapter` can be referenced, but not the components of object `a0`. The inside-out rule doesn't allow looking inward, only outward.

From method `p` in object `a1`, none of the items in `Chapter`'s file drawer can be referenced directly.

However, within method `p`, we can reference field c1, which contains the name a0. Therefore, all components in a0 can be referenced, as well as the static components in Chapter's file drawer, using references like

```
c1.title     c1.getTitle()    c1.m(3)   c1.m(true)
```

Thus, the name of an object gives access not only to the components of the object but also, by the inside-out rule, to the static components in the file drawer.

Just as the name of the object is used as a gateway to that object, the name of a class is used as a gateway to the static components in the class file drawer. For example, in the body of method `p` in class `C`, we can call method a method m of in `Chapter`'s file drawer by preceding the call with the name of the class, `Chapter`:

```
Chapter.m(n)
```

A static component of a class, then, can be referenced from anywhere simply by prepending the name of the class (followed by a period, of course) to the reference.

Note that in method `p`, we can use either `c1.m(n)` or `Chapter.m(n)` to call the first method `m`. The use of the name of the `Chapter` is preferred.