25.8 Declaring Methods

[*Note:* This section may be read after studying Chapter 6, Methods: A Deeper Look, and the preceding portions of Chapter 25.]

You can use JShell to prototype methods. For example, let’s assume we’d like to write code that displays the cubes of the values from 1 through 10. For the purpose of this discussion, we’re going to define two methods:

- Method `displayCubes` will iterate 10 times, calling method `cube` each time.
- Method `cube` will receive one int value and return the cube of that value.

25.8.1 Forward Referencing an Undeclared Method—Declaring Method `displayCubes`

Let’s begin with method `displayCubes`. Start a new JShell session or `/reset` the current one, then enter the following method declaration:

```
void displayCubes() {
      for (int i = 1; i <= 10; i++) {
         System.out.println("Cube of " + i + " is " + cu
      }
   }
```

When you complete the method declaration, JShell displays:

```
   | created method displayCubes(), however, it cannot b
   until method cube(int) is declared

   jshell>
```

Again, we *manually* added the indentation. Note that after you type the method body's opening left brace, JShell displays continuation prompts (...>) before each subsequent line until you complete the method declaration by entering its closing right brace. Also, although JShell says `"created method displayCubes()"`, it indicates that you cannot call this method until `"cube(int) is declared"`. This is *not* fatal in JShell—it recognizes that `displayCubes` depends on an undeclared method (`cube`)—this is known as **forward referencing** an undeclared method. Once you define `cube`, you can call `displayCubes`.

25.8.2 Declaring a Previously Undeclared Method

Next, let’s declare method `cube`, but *purposely make a logic error* by returning the square rather than the cube of its argument:

```
jshell> int cube(int x) {
   ...>    return x * x;
   ...> }
|  created method cube(int)

jshell>
```

At this point, you can use JShell’s **`/methods`** command to see the complete list of methods that are declared in the current JShell session:

```
jshell> /methods
| void displayCubes()
| int cube(int)

jshell>
```

Note that JShell displays each method’s return type to the right of the parameter list.

25.8.3 Testing `cube` and Replacing Its Declaration

Now that method `cube` is declared, let’s test it with the argument 2:

```
jshell> cube(2)
$3 ==> 4

jshell>
```

The method correctly returns the 4 (that is, 2 * 2), based on how the method is implemented. However, our the method’s purpose was to calculate the cube of the argument, so the result should have been 8 (2 * 2 * 2). You can edit `cube`’s snippet to correct the problem. Because `cube` was declared as a multiline snippet, the easiest way to edit the declaration is using **JShell Edit Pad**. You could use `/list` to determine `cube`’s snippet ID then use `/edit` followed by the ID to open the snippet. You also edit the method by specifying its name, as in:

```
jshell> /edit cube
```

In the **JShell Edit Pad** window, change `cube`’s body to:

```
return x * x * x;
```

then press **Exit**. JShell displays:

```
jshell> /edit cube
|   modified method cube(int)
```

```
jshell>
```

25.8.4 Testing Updated Method `cube` and Method `displayCubes`

Now that method `cube` is properly declared, let's test it again with the arguments `2` and `10`:

```
jshell> cube(2)
$5 ==> 8

jshell> cube(10)
$6 ==> 1000

jshell>
```

The method properly returns the cubes of `2` (that is, `8`) and `10` (that is, `1000`), and stores the results in the implicit variables `$5` and `$6`.

Now let's test `displayCubes`. If you type `"di"` and press *Tab*, JShell auto-completes the name, including the parentheses of the method call, because `displayCubes` receives no parameters. The following shows the results of the call:

```
jshell> displayCubes()
Cube of 1 is 1
Cube of 2 is 8
Cube of 3 is 27
Cube of 4 is 64
Cube of 5 is 125
Cube of 6 is 216
Cube of 7 is 343
Cube of 8 is 512
Cube of 9 is 729
Cube of 10 is 1000

jshell>
```