The subclass definition

The manila folder for subclass Demo

Here is a manilla folder —an object or instance— of class `JFrame`. The tab contains the name of the folder, the name of the class appear in the upper right, and we show only some of the methods and none of the fields in the folder.

The definition of class `Demo` *extends* `JFrame`. We say that `Demo` is a *subclass* of `JFrame` and `JFrame` is the *superclass* of `Demo`. Remember these terms.

We draw a folder of class `Demo` with two partitions. The top partition is labeled `JFrame`; it contains all the components —fields and methods— that a `JFrame` has. We say that `Demo` *inherits* all these components. The bottom partition is for components that are defined in `Demo` itself. There are none yet.

The `Demo` partition appears below the `JFrame` partition because `Demo` is a subclass of `JFrame`.

Defining a function

Let's define a function in class `Demo` that yields the area of the window in pixels —its height times its width. This is your first Java function definition. We start off with its specification, as a comment. It says that a call of this function equals the area of the window.

Now we put the header of the function. Keyword **`public`** means that it can be called from any part of the program; the function will return an integer, an **`int`**; and we give the function the name `area`. Parentheses are needed after the method name.

Then comes the *body* of the method, which is enclosed in braces. Note that we put the opening brace on the same line as the header of the method.

In the body of the method, we put a *return* statement, giving keyword **return** followed by the expression whose value is to be returned followed by a semicolon. And that completes your first function definition:

```
/** = area of the window of this JFrame */
   public int area() {
      return getWidth() * getHeight();
   }
```

Let's compile the class again. Since the definition is now in the class, we write the new function in the lower partition of each instance of `Demo`. To impress upon you that the whole method appears in the object, and to make a point in a moment, we have written the complete method body —we'll erase it later.

The method body calls function `getWidth`. How does it know where this method is? Since `getWidth` is not defined in the method body itself, we simply look outward, and upward, and find it in the `JFrame` partition, so that is the `getWidth` function that is called. This idea of looking from the inside out to enclosing constructs will be explained in more detail later on.

Watching a call on the new function

Now let's create a new instance of class `Demo`, store its name in variable `j`, show the window, drag to make it bigger, and call function `area` using `j.area()` —look, it worked! You have seen your first call of a function that you wrote.

Defining a procedure

A function returns a value; a procedure does not. Instead. The procedure does something. Let's write a procedure that sets the title of the window to the area of the window, so that the bottom partition contains two methods.We start by writing a specification of the procedure:

```
/** Set title to "Area is " + <area of window> */
```

Now, we write the header of the procedure. We start with **public**, so the procedure is available everywhere. Since this is a procedure, in place of the "return type" that is used in a function, we write **void**. We choose `setTitleToArea` for the name of the procedure. And we need the parentheses and then the braces for the procedure body.

The body itself will be a single statement that calls procedure `setTitle`, with an argument that gives the desired string —notice how we call function area.

```
/** Set title to "Area is " + <area of window> */
public void setTitleToArea() {
    setTitle("Area is " + area());
 }
```

We compile the program, store a new instance of `Demo` in `j`, and show it. Let's drag to make it bigger and then call procedure `setTitleToArea`. And let's do it again to see that the area actually changes.

Summary

You have seen a function definition and a procedure definition. The function definition has the *return type* —the type of value the function yields— while the procedure has the word **`void`** in its place.

The function body must have as its last executed statement a return-statement, which gives the expression whose value is to be returned. The procedure body need not have a return-statement. Its body consists of a sequence of statements to be executed.

The name of a function is generally a noun (or noun phrase) that tells you what is computed (like `area`, or `sqrt`). It may have the word `get` in front of it. The name of a procedure is usually a command to do something. Both functions and procedures generally start with small letters, and if they consist of several words, the first letters of all but the first word are capitalized.

We will return to a study of functions and procedures later on.

You have also seen what an instance of a subclass looks like —it has a partition for the superclass and, underneath it, a partition for the subclass. The bodies of methods defined in the subclass can reference inherited methods and fields directly.