Types and referencing components of an object

A class name like Patient is also considered to be a type. Its values are the names of manila folders, or objects, of that class. The type has no operations, because one shouldn't be able to operate on these names, much like you can't change the address of your house or apartment.

Since Patient is a type, just as one can declare variables of type **int**, one can declare variables of type Patient.

Calling methods

Suppose variable p contains the name a0 of an object. In this situation, in Java, one can call method getName of the object using (1) the name of the variable, (2) a period, (3) the function name, and (4) parentheses:

```
p.getName()
```

For example, we could use this call in an assignment statement to store the name "Jack Smith" in a variable s:

```
S=  p.getName();
```

In the same way, we can call procedure deposit using a statement that consists of (1) the name of the variable, (2) a period, (3) the name of the procedure, (4) parentheses with an argument between them —the amount to be deposited—, and (5) a semicolon:

```
p.deposit(250.00);
```

A statement like this one is a command to do something, and its execution in this situation will change the amount that is owed to 0.

In summary, to write a call on a method of an object whose name is in a variable p,

(1) use the name of the name of the variable followed by a period, and then

(2) a call that is typical in most programming languages, consisting of the method name followed by arguments (if any) in parentheses.

(3) If the method is a procedure, so that the call is a statement, a semicolon is also needed.

We will go over method calls in more detail later.

Fields can be referenced in much the same way.

```
p.name
```

On your screen, you see two examples of this, one assigning p.name to a String variable s and the other subtracting $250.00 from field p.owes.

However, the standard practice in OO programming is to make most fields "private" —we'll see how this is done later— so they cannot be accessed from outside the class. This is standard software engineering practice. The reasons for it will be discussed at the appropriate place, later on.

Value null

On your screen is a declaration of a variable of type Patient. Initially, variable p does not contain the name of an object because none has been assigned to it. Instead, p contains the value **null**, a special value that means the absence of an object name.

In this situation, if the program attempts to evaluate the expression

```
p.getName()
```

a "NullPointerException" will occur and execution of the program will abort. You can't reference a component (a method or a field) of an object that doesn't exist.

Creating an object —the new-expression

Finally, we show you how to create a new object, using a new-expression. Evaluation of the expression

```
new Patient()
```

creates an object and yields as its value the name of the new object. If Java is evaluating the new-expression, it creates an arbitrary name for the object. When *you* evaluate the new-expression and draw a new object, choose any name you want. Here, we have chosen the name a0.

- It is important for you to understand that the new expression produces a value —the name of the newly created object. We can say that the new-expression is evaluated in a two-step process, which will be expanded on later:

    Step 1. Create a new object of class Patient (giving it a name in its tab)

    Step 2. Yield as value of the expression the name on the tab.

- Now, we need to store the value of this expression in variable p. Execution of this statement therefore changes the value of p to a0.

- **Summary**

  You have seen several new items in this little lecture:

  1. How to call a method or reference a field of an object whose name is in a variable p.

- 2. The meaning of value **null**.

- 3. The new-expression, whose evaluation creates (draws) a new manila folder or object and has as its value the name of the new object.

In this little lecture and the previous one, you saw powerpoint slides, making the lectures just a bit abstract. In the next lecture, you will see this made concrete as we use DrJava to illustrate what we have discussed thus far.