Objects - Instance Variables

When you create an object, that object can have instance variables. These are variables that belong to the object. As long as the object isn't destroyed, its instance variables continue to exist and hold the last value they were set to. In this way, instance variables are much like the workspace variables discussed above. In fact, workspace variables are just instance variables of the workspace object!

Creating an Object and its Instance Variables

Create a new class called cash-register and add three fields to it named amount, sub-total, and daily-total. To do this:

1.
Tap on New Class… from the Utilities menu. The class creator window appears.
2.
Enter the name of the class to be cash-register, and select the top-most window style icon.
3.
Tap Create & Close. The class creator window will close and the new cash-register object will open.
4.
For each field:
a.
Caret in an empty place in the grid. A pop-up list of field types appears. Choose an empty place in the right column.
b.
Tap Number in the pop-up menu. A number field is added to the cash-register object.
c.
Circle on the new field to rename it.
d.
Edit the name to one of amount, sub-total, or daily-total, then tap OK.

5.
If you want, you can add labels beside the fields and edit the labels to name the fields.

You now have created an object, named cash-register. This object has three instance variables (or properties) named amount, sub-total and daily-total. Cash-register also has a form-like view that displays these three properties in numeric write-in fields. If you toggle the mode of the window to user-mode (the aligner lines disappear), you can edit the values of the variables in the cash-register windows. If you do this, toggle the mode back to author-mode (the aligner lines appear again) before proceeding.

Accessing the Instance Variables

When you write scripts for an object, its instance variables can be accessed and set by name. You will write a script for cash-register that clears all three variables to zero:

1.
Add a button to cash-register. Circle the button and rename it ‘Clear’.
2.
Tap on the button to edit its script.
3.
Replace the entire script with:

```
$ clear-daily.

clear-daily := ask user "Clear daily-total too?".

amount := 0.

sub-total := 0.

if clear-daily then [ daily-total := 0 ].
```

4.
Tap Save. Make sure there were no errors. A common error is to get the message “Unknown local or property”. This generally means that you either misspelt the name of a property in the script, or when you named the property. Correct the misspelling in the script or of the property and save again.

When you add a button, you are adding a script property to an object. Tapping the button (in user-mode) causes the script to run. Toggle cash-register into user-mode and try it.

In the rest of the manual, you will be asked to add buttons to objects with various scripts. To save space, the detailed instructions will not be give. Just follow the procedure above.

» The Ask Message

The script above uses a new message: ask sent to user. User is an object that represents the real user to your scripts. The ask message puts up a note asking a question and offers two choices:

Yes and No. If the user taps Yes, the result of the ask message is true, otherwise the result is false. Notice how the script stores the result of the ask message in a local variable and then use that result later.

It is better practice to offer the user the option of canceling an operation as destructive as clear. Fortunately, the ask message has just such an option. Replace the first line in the script above with these two lines:

```
clear-daily := ask user "Clear daily-total too?" with-cancel.

if (clear-daily is ???) then [ return ].
```

The optional argument with-cancel directs the ask message to include a Cancel button in the note. If the user taps Cancel instead of Yes or No, then the result of the ask message is ??? (unknown). The if statement tests for this case and executes a return if so, stopping all further processing of the script.

Accessing Scripts from Elsewhere

Once you have defined the Clear button above, the object cash-register will now understand the message clear. In a workspace you can execute:

```
clear cash-register.
```

This will act exactly as if you had tapped the Clear button. In fact, when you tap the Clear button, this is the expression the system evaluates.

»
Remember that the order of the message and the receiver can be reversed. If you find it reads better, you can write the expression above as:

```
cash-register clear.
```

Accessing Instance Variable from Elsewhere

In a script of an object (like the script of the Clear button), expressions can refer to that object's instance variables simply by name. However, if you want to access the instance variables from a script that doesn't belong to the object (from a workspace, for example), then you must use a different syntax.

To refer to the instance variable of another object, you use dot-notation. You write the name of the object, a period, and the name of the variable. For example, in the workspace you can execute:

```
cash-register.sub-total := 22.50.

cash-register.total := cash-register.total + cash-register.sub-total.
```

!!
There are no spaces around a period used in dot-notation. If you accidentally put a space in, then the system will confuse the period for the period at the end of an expression.

»
Notice that dot-notation is exactly the same as the short-hand notation for messages that have no arguments. In fact, instance variables and messages with no arguments can be each be written in the same three different ways. Across each line, the three expression are equivalent and do the same thing:

Dot-Notation	Message After	Message Before
cash-register.total	cash-register total	total cash-register
cash-register.clear	cash-register clear	clear cash-register

Having three different ways of writing the same thing may be a bit confusing at first, but it allows you to choose the best way of writing an expression.

Sometimes it is convenient to set a local variable to an object. When you do this, the variable refers to the object. This is the same as saying the value of the variable is the object, but it

makes it clear that any use of the variable will really use the object. For example, the above script could have been written as:

```
$ cr.

cr := cash-register.

cr.sub-total := 22.50.

cr.total := cr.total + cr.sub-total.
```