1

Various Points about Conditions

In this chapter we present some extra points about conditional such as how to use conditional to debug your code. The material presented here is more advanced and can be skip in a first reading.

1 Debugging using Conditional

Often one would like to check whether a conditional code has been executed. One way to do that is to insert expression such as `Transcript show: 'I pass there' ; cr.`. Another approach is to insert `self halt` that when executed opens a debugger. A third way is insert some messages that produce sounds.

Script 1.1 (*Only clink)*

```
| caro |
caro := Turtle new.
caro distanceDetector.
caro color = Color red
   ifTrue: [Smalltalk beep].
caro jump: 250.
caro distanceDetector.
caro color = Color green
   ifTrue: [Smalltalk beep].
```

Another interesting use of conditional is to specify conditional breakpoints. For example, when we want to stop when a loop arrives at a given point we just have to include an expression similar to `myCondition ifTrue: [self halt]`.

An interesting way to have a conditional stop of a program execution is to check when certain keys are pressed such as control or shift. The expression `InputSensor default shiftPressed ifTrue: [self halt]` only opens a debugger when the shift key is pressed. The class `InputSensor` defines other messages such as `commandKeyPressed`, `controlKeyPressed`, or `anyButtonPressed` that can be used to define conditional break points.

2 Tricky Aspects of Parenthesis

It may happen that you get some trouble with the syntax when using the methods `ifTrue:` and its companion methods `ifFalse:`, `ifTrue:ifFalse:`, and `ifFalse:ifTrue:`. Indeed you have to take care that the the boolean expression may be parenthesed.

In the script **??** there is no problem because the expression used in the boolean expression, `Time now > (Time new hours: 8)` are not keywords messages, so they are evaluated prior to the `ifTrue:` that is executed if the result of the expression is true.

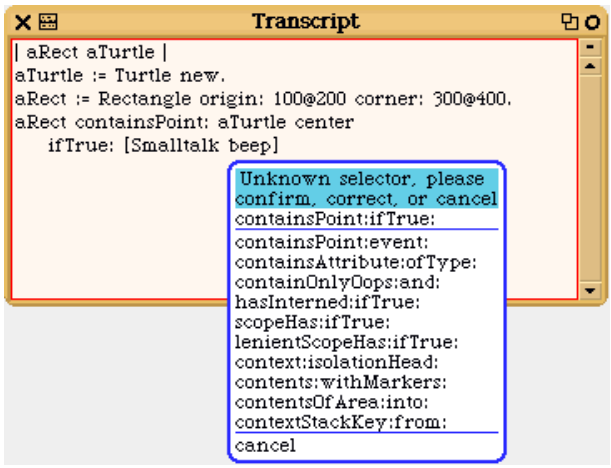

Figure 1.1: The environment is trying to recover an error due to missing parenthesis. The message is not `containsPoint:ifTrue:` and the boolean expression containing the message `containsPoint:` should be enclosed by parentheses.

In the script 1.2 the boolean expression uses a keyword message, `containsPoint:`, which has the *same weight* than the `ifTrue:` message. So the boolean expression will not be evaluated first as the system believes that the message is in fact named `containsPoint:ifTrue:`. To disambigate this situation parenthesis are required as shown in the script 1.2.

Script 1.2 (*Conditional Requiring Parenthesis)*

```
| aRect aTurtle |
aTurtle := Turtle new.
aRect := Rectangle origin: 100@200 corner: 300@400.
(aRect containsPoint: aTurtle center)
    ifTrue: [Smalltalk beep]
```

Important!

Surround boolean expressions with parenthesis when they contain keyword based messages to avoid message ambiguities

3 Factoring Commonalities

As we already mentioned it, duplicated logic is not a sign for potential improvements. The method `distanceDetector` exhibits some duplicated logic in each branch of the condition `self color:` . Here the duplication is not severe but we use this method as a pretext to illustrate our point.

Method 1.1

```
distanceDetector

   | dist aColor|
   dist := self distanceFrom: World bounds center.
   dist > 300
     ifTrue: [ self color: Color green]
     ifFalse: [dist < 200
                ifTrue: [self color: Color red]
                ifFalse: [self color: Color yellow]]
```

Let us start by factoring commonalities out of the conditions. As we mentioned earlier, the code in the conditions does not have to be the same. The script 1.2 shows an equivalent version in which the expression `self color:` has been factored out of the condition. In this new version the conditional code returns the new color. The final line changes the receiver color using the selected color. This way is it easier to apply a consistent manipulation to the color such as darkening it.

Method 1.2

```
distanceDetector

   | dist aColor|
   dist := self distanceFrom: World bounds center.
   aColor := dist > 300
         ifTrue: [Color green]
         ifFalse: [dist < 200
            ifTrue: [Color red]
            ifFalse: [Color yellow]].
   self color: aColor
```

4 About Method Returns

When a return statement, `^ anExpression`, is encountered, the execution of the method is terminated and the result of the expression following the return is returned to the caller of the method. No other piece of code is executed in the method containing the return statement. This behavior can be used to write condition is a different way. The method 1.3 shows the method **??** using explicit return. Here as soon as a boolean expression is true, the return statement is executed, *i.e.*,its expression is executed and the rest of the method is skipped.

Method 1.3

```
coloredTurn: anAngle
    "change the color of the turtle so that it is blue aiming
    at the north and red to the south"

   self turn: anAngle.
   self direction = 90
      ifTrue: [^ self color: Color blue].
   self direction = -90
      ifTrue: [^ self color: Color red].
   self color: Color green]
```

Note that per default in SQUEAK, a method returns its receiver, even if the code does not show it as

shown by the method 1.4 which is then equivalent to the method 1.5.

Method 1.4

```
doSomething

   Transcript show: 'doSomething' ; cr.
```

Method 1.5

```
doSomething

   Transcript show: 'doSomething' ; cr.
   ^ self
```

5 About Code Formatting

In Smalltalk, the code can be layouted in all kind of ways and the indentation (its shape regardings the left margin) has no semantics. However, using a clear indentation really helps the reader to understand the code. We suggest to follow the convention we chose to format `ifTrue:ifFalse:` expressions.

Again, the idea is that the block of expressions delimited by the characters `[` and `]` should form a visual and textual rectangle. First the boolean expression is always on one line and the keywords are always indented on the next line. Then when the block fits into a single line we just put it after the `ifTrue:` or `ifFalse:` keyword to gain vertical space and we close the block with the `]` at the end of the line as follow:

```
self direction = 90
      ifTrue: [self color: Color blue]
```

When the block cannot stand on one single line, we apply the same strategy that for `timesRepeat:`. We start the block with `[` on the next line and align all the expressions inside the block to one tab and finish by `]` that indicates that the block ends as follows:

```
dist < 200
      ifTrue: [self color: Color red.
              Transcript show: 'red' ; cr]
      ifFalse: [self color: Color green
               Transcript show: 'green' ; cr]
```