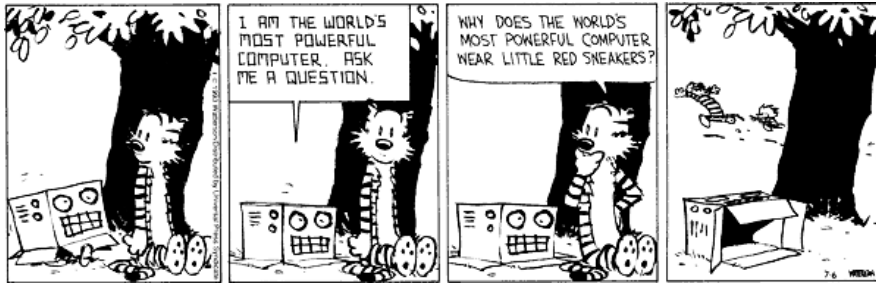


Assignment 2: Give me your feature name and I'll call you

ETH Zurich

Hand-out: 26 September 2008
Due: 2 October 2008



Calvin and Hobbes© Bill Watterson

Goals

- Write more feature calls.
- Get used to EiffelStudio.
- Know a first (very unprofessional) way to distinguish between queries and commands (to be improved in the next assignment).
- Learn what makes up a valid feature call.

1 Adding more feature calls

Open the 02_objects system again and navigate to the class [PREVIEW](#).

Todo

1. Add additional feature calls that revert the actions of highlighting Line8 (use feature *unhighlight*) and spotlighting the Louvre (feature *unspotlight*). Execute the system again.
2. Have you noticed that the calls to *unhighlight* and *unspotlight* seem to be executed simultaneously? This has two reasons:
 - First of all the computer is very fast, so that in general the human eye may hardly notice when the two simple instructions are executed one after the other (but they are indeed executed one after the other!). It usually does not wait for a couple of seconds to continue with the next instruction.

- Second, GUI (Graphical User Interface) programs normally do not update the screen after every instruction.

In fact, the previous instructions *Louvre.spotlight*, *Line8.highlight*, and *Route1.animate* are tweaked to internally call a feature *wait* that is available from *TOURISM*. This is part of the "magic". The feature *wait* halts the application for a couple of seconds and updates the screen. You can use this feature in *explore* to time the execution of instructions. Add calls to *wait* before unhighlighting Line8 and unspotlighting the Louvre.

3. Add additional feature calls to the end of the above code:
 - i Display information about Line2 on the Console and *wait*.
 - ii Highlight Line2 and *wait*.
 - iii Highlight the south end of Line2 (using features *south_end* and *highlight*) and *wait*.
 - iv Highlight the north end of Line2 (using features *north_end* and *highlight*) and *wait*.

To hand in

Hand in the code of feature *explore*.

2 Category guessing

Todo

The features listed below can be found in class *TRAFFIC_STATION* that represents stations in the city to which roads may lead or that may have public transportation lines passing through. For each of the features, figure out whether it is a command or a query. Until you have learnt a better way to do this (next week), you can solve this by asking yourself:

- "Does the feature name sound as if the target object is modified?" If yes, it is probably a command.
- "Does the feature name sound as if you would get information on the object?" If yes, it is probably a query.

Features:

1. A feature *is_exchange*, used under the form *Station_balard.is_exchange*.
2. A feature *set_location*, used under the form *Station_balard.set_location (a_point)*.
3. A feature *outgoing_line_connections*, used under the form *Station_balard.outgoing_line_connections*.
4. A feature *name*, used under the form *Station_balard.name*.
5. A feature *highlight*, used under the form *Station_balard.highlight*.
6. A feature *has_stop*, used under the form *Station_balard.has_stop (Line7)*.

To hand in

Submit your solution to your assistant.

3 Valid feature call instructions

A feature call instruction is composed of an optional *target* (the object to which an operation is applied), exactly one *command* (the operation to apply), and possibly some arguments. The target and the arguments are expressions made up by queries only. No commands are allowed within the arguments or the target. Queries may have arguments themselves (see fifth example below). There is exactly one command in a feature call instruction. It appears after the optional target and may be followed only by its arguments. Below you find examples of feature call instructions where **queries** are marked in yellow and **commands** are marked darker in red. The first six instructions are valid (i.e. they compile) and the others are invalid instructions. For the invalid instructions an explanation is given in square brackets. Make sure you understand these examples.

- ✓ `Station_Balard.highlight`
- ✓ `Line1.south_end.location.left_by (Line1.south_end.width)`
- ✓ `Line7_a.set_color (Line3.color)`
- ✓ `wait`
- ✓ `Paris.station_at_location (Station_Balard.location).unhighlight`
- ✓ `Console.show (Line3.south_end.has_stop (Line7_a))`
- ✗ `Station_Balard.is_highlighted` [no command in instruction]
- ✗ `Paris.station_at_location (Station_Balard.unhighlight)` [command in argument]
- ✗ `Line7_a.set_color (Line3.set_color (Line8.color))` [command in argument]
- ✗ `Line1` [no command in instruction]
- ✗ `Console.show (Line3).Station_Balard` [query after command]

Todo

Assume that *highlight*, *show*, *set_red*, *set_color*, and *set_location* are commands and all other feature names denote queries. Which of the following instructions are valid? Explain your decision. You do not need your computer to answer these questions and there is no guessing involved in this task!

1. `Console.show.Station_Balard`
2. `Station_Balard.set_location (Station_Issy.location)`
3. `Line2.set_color (Line8.highlight)`
4. `Line2.color.set_red (Line8.color.red)`
5. `Console.show (Paris.station_at_location (Station_Balard.location))`
6. `Console.show (Paris.station_at_location (Station_Balard.location).name)`
7. `Line8.north_end.set_location (Route1.city.station_at_location (Station_Balard.location).set_location)`

To hand in

Submit your answers to your assistant.