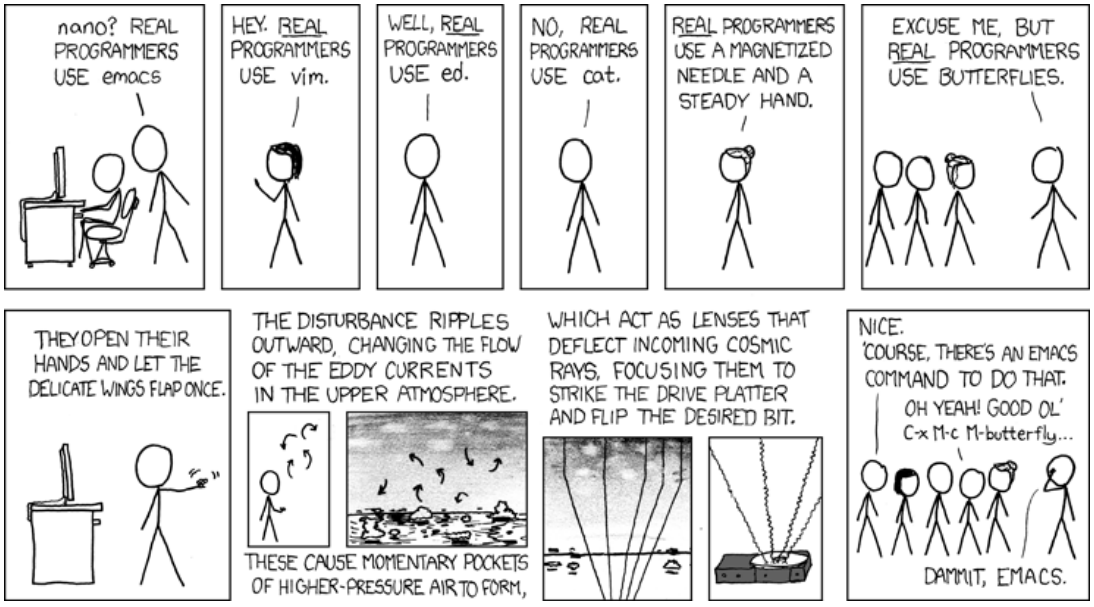
Assignment 2: Give me your feature name and I'll call you

ETH Zurich

Hand-out: Friday, 24 September 2010
Due: Monday, 4 October 2010



Real Programmers © Randall Munroe (xkcd.com)

Goals

- Write more feature calls.
- Write your first standalone program.
- Get used to EiffelStudio (editor, navigation and debugger).
- Learn to distinguish between queries and commands.
- Learn what makes up a valid feature call.

1 Adding more feature calls

Open the 02_objects system again and open the class *PREVIEW* in the editor area.

Todo

1. In feature *explore*, add an additional feature call to revert the action of highlighting Line8 (use feature *unhighlight*). To make the highlight-unhighlight sequence noticeable, put a *wait* instruction between the two calls and execute the system. The feature *wait* pauses the application for a couple of seconds and updates the screen. Notice that *wait* is not invoked as the other features, by using an object name and then a dot, but just as it is (it is an *unqualified call* [Touch Of Class, page 134]).

2. Let us find out where the feature *wait* comes from. As it appears in an unqualified call, it must be defined either in the same class or in an *ancestor* class. An ancestor class for a class C is a class C *inherits* from. You may have noticed the **inherit** *TOURISM* clause after **class** *PREVIEW*. It means that *PREVIEW* can use all the features defined in *TOURISM*.

   In *PREVIEW* there is no *wait*, so let us check *TOURISM*. Right-click on the label *TOURISM* and choose the option "Retarget to class TOURISM". You can also type "tourism" in the drop down box on the top left (labeled "Class").

   Let us now check the features of class *TOURISM*. On the bottom of the right panel select the tab labeled "Features". You should now see a list of all the features defined in class class *TOURISM*. Unfortunately, *wait* is not there yet, but there is still hope: *TOURISM* inherits from *TOUCH_PARIS_OBJECTS*, so you can repeat what you have just done and finally you should find the wanted feature.

   Tip: there are two shorter ways to find *wait*. While in class *PREVIEW*, type "wait" in the drop down box labeled "Feature" above the editor window. Alternatively, right-click on *wait* in the program text and then select "Retarget to Feature wait". This will bring up the desired feature in class *TOUCH_PARIS_OBJECTS*.

3. Imagine you want to give a friend step-by-step directions how to get from one subway station to another in Paris. You decided to do it by sequentially highlighting relevant stations and line segments. For example, to show a way from "Trocadéro" to "Porte Dauphine" (figure 1), you would first highlight the station "Trocadéro", then (after a short time) the segment of line 6 between "Trocadéro" and "Charles de Gaulle Étoile", then the station "Charles de Gaulle Étoile", where he has to change lines, then the segment of line 2 between "Charles de Gaulle Étoile" and "Porte Dauphine", and finally the station "Porte Dauphine".

   Modify the feature *explore* so that it shows your friend a way from "Invalides" to "Palais Royal Musée du Louvre" (comment out the code that was in *explore* before, but leave *Paris*. *display* as the first instruction, otherwise the program will fail).

   Browse the class *TOUCH_PARIS_OBJECTS* to find the features you need. In particular pay attention to:

   - features that return specific stations and lines;
   - feature *line_section* that takes a line, a start station and an end station as arguments [Touch Of Class, page 30] and returns a line segment between these two stations[1];
   - features *wait* and *short_wait*.

   Tip: To check the list of all available features, press "Ctrl + Space" while in the editor window. To check the list of available features, whose names start with a certain prefix, type this prefix and then press "Ctrl + Space" (figure 2).

[1] In order for highlighting to work properly the first station you pass to *line_section* should be closer to the south end of the line and the second one — to the north end.

Figure 1: Paris subway (fragment)

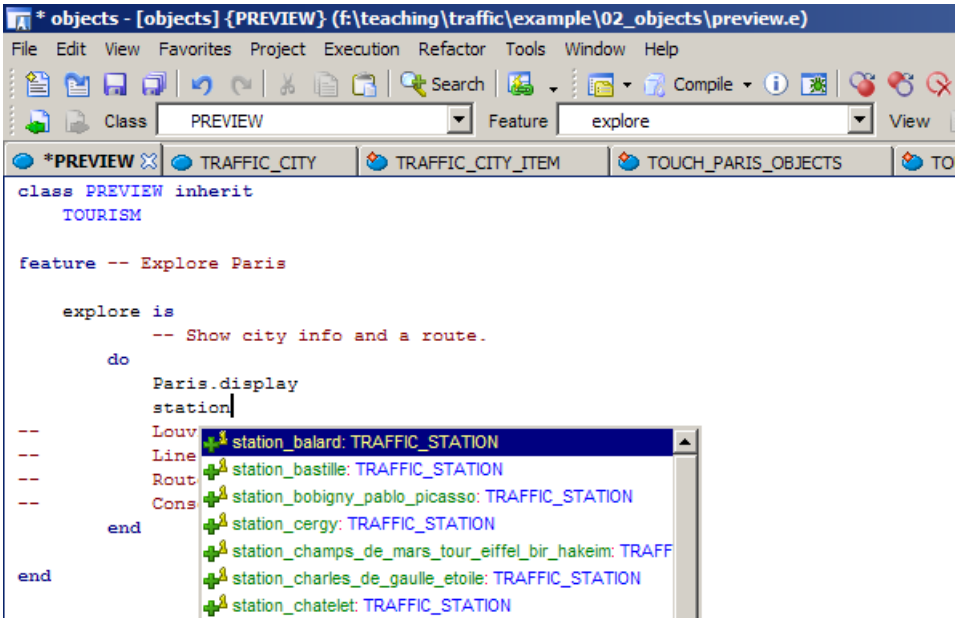


Figure 2: EiffelStudio auto-completion feature

To hand in

Hand in the code of feature *explore*.

2 Introducing yourself

In this task you will write your first standalone program (not based on traffic). The program will introduce yourself to your assistant.

Todo

1. Download the project file "introduction.ecf" from http://se.inf.ethz.ch/teaching/2010-H/eprog-0001/assignments/02/introduction.ecf and the source code file "application.e" from http://se.inf.ethz.ch/teaching/2010-H/eprog-0001/assignments/02/application.e. Put both files into the same directory.

   Open the "introduction.ecf" file you just downloaded in EiffelStudio. In the "Groups" tool on the right you can see that the whole project consists of a single class *APPLICATION*. Open this class in the editor. You will see that it has a single feature, *execute*, whose body is empty so far.

2. Modify the feature *execute* so that it prints the following text (replace the information about John Smith with your personal data):

   ```
   Name:  John Smith
   Age:  20
   Mother tongue:  English
   Has a cat:  True
   ```

   You can also add any other information you like.

   To do the printing you will use the predefined object called *Io* (input-output). The features you can call on *Io* are defined in the class *STD_FILES*. Browse this class to find the features you need. In particular pay attention to:

   - feature *put_string* that takes a text string (e.g. `"Hello, world!"`) as an argument and prints it;
   - feature *put_integer* that takes an integer number (e.g. `5`) as an argument and prints it;
   - feature *put_boolean* that takes a boolean value (`True` or `False`) as an argument and prints it;
   - feature *new_line* that moves to the next line.

3. Until now you have compiled and executed a program without having the possibility to check what happened after each single instruction was executed. Now let us see how to use EiffelStudio in *debug mode* [Touch Of Class, page 170]. Being in debug mode means being able to observe the application execution instruction by instruction, therefore increasing the chances to discover errors ("bugs").

   Right-click on the feature name *execute* in the program text and choose "Pick feature execute". Now right-click in the context tool (the area below the editor). The code of *execute* should now appear in the context tool, with gray circles on the left (see an example on figure 3). These circles identify instructions that will be executed. Click on the first gray circle; it should become red. You have just set a breakpoint, at which the program will pause execution.

   Now click on the green full arrow (or press "F5"): the program will start, but almost immediately it will pause its execution at your breakpoint. Now you can observe the program behavior step by step by clicking on the button to the right of the one with a red square (or pressing "F10"). To resume the normal execution click on the green full arrow again (or pressing "F5").

To hand in

Hand in the code of feature *execute*.

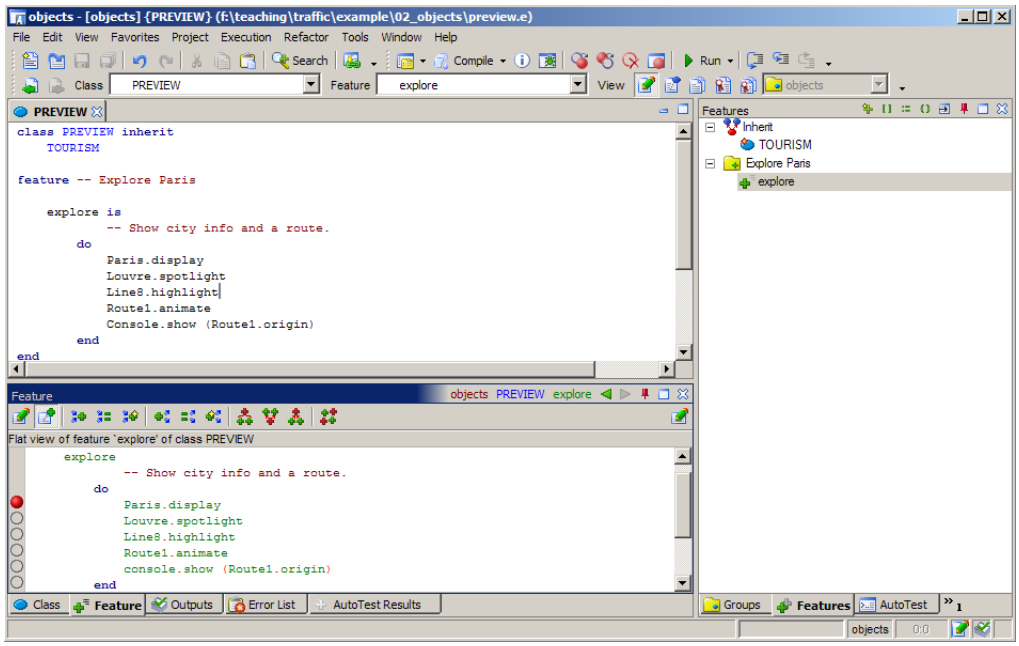


Figure 3: Setting a breakpoint

3 Command or Query?

Todo

The features listed below can be found in class *TRAFFIC_STATION*, representing stations in the city. We want to find out which features are *commands* and which features are *queries* [Touch Of Class, page 29]. Let us have a look at the feature definition. If it appears in the form:

feature_name: *CLASS_NAME* or *feature_name* (...) : *CLASS_NAME*,

then it is a query. If it appears in the form:

feature_name or *feature_name* (...) ,

then it is a command.

Now for each of the following features in *TRAFFIC_STATION*, figure out whether it is a command or a query:

1. Feature *is_exchange*, like in *Station_balard*.*is_exchange*.
2. Feature *set_location*, like in *Station_balard*.*set_location* (*a_point*).
3. Feature *outgoing_line_connections*, like in *Station_balard*.*outgoing_line_connections*.
4. Feature *name*, like in *Station_balard*.*name*.
5. Feature *highlight*, like in *Station_balard*.*highlight*.
6. Feature *has_stop*, like in *Station_balard*.*has_stop* (*Line7*).

To hand in

Hand in your answers.

4 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. A feature call instruction is valid only if:

- The target and the arguments are expressions, which contain only query calls (possibly with arguments).
- There is exactly one command call 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 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*))
χ *Station_Balard* . *is_highlighted* [no command in instruction]
χ *Paris* . *station_at_location* (*Station_Balard* . *unhighlight*) [command in argument]
χ *Line7_a* . *set_color* (*Line3* . *set_color* (*Line8* . *color*)) [command in argument]
χ *Line1* [no command in instruction]
χ *Console* . *show* (*Line3*). *Station_Balard* [query after command]

Todo

Assume that *highlight*, *show*, *set_color*, *set_radius* and *set_location* are commands and all other feature names denote queries. Which of the following instructions are valid? Explain your decision. Note that you do not need your computer to answer these questions!

1. *Console.show.Station_Balard*
2. *Station_Balard. set_location* (*Station_Issy . location*)
3. *Line2. set_color* (*Line8. highlight*)
4. *Line2. city . set_radius* (*Line3. city . radius*)
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

Hand in your answers.