Assignment 3: Of objects and features

ETH Zurich

Hand-out: 3 October 2008
Due: 9 October 2008

Calvin and Hobbes © Bill Watterson

Goals

- Understand the difference between a class and an object.
- Know a second (professional) way to distinguish between queries and commands.
- Learn to read feature call instructions.
- Write more feature call instructions.

1 Classes vs. objects

To do

1.1 Describe in your own words the difference between a class and an object (1-2 sentences).
1.2 Find an analogy that captures the relationship between objects and classes in real life.

To hand in

Write down your answers (1.1 and 1.2) and hand them in.

2 Categorizing features (no guessing any more)

There are two main categories of features: queries and commands. In last week’s assignment you categorized features by the sound of their name. This is not a very professional way to do this. In this week’s lecture you learnt how features can be categorized professionally by looking
at their declaration in the class interface. The general patterns for queries and commands in a class interface are:

<table>
<thead>
<tr>
<th>no arguments</th>
<th>with arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>command</strong></td>
<td><strong>feature signature</strong></td>
</tr>
<tr>
<td>command_name</td>
<td>argument declaration</td>
</tr>
<tr>
<td>Examples:</td>
<td></td>
</tr>
<tr>
<td>spotlight</td>
<td></td>
</tr>
<tr>
<td><strong>query</strong></td>
<td><strong>feature signature</strong></td>
</tr>
<tr>
<td>query_name : TYPE</td>
<td>return type</td>
</tr>
<tr>
<td>Examples:</td>
<td></td>
</tr>
<tr>
<td>center: TRAFFIC_POINT</td>
<td></td>
</tr>
<tr>
<td>corner_1: TRAFFIC_POINT</td>
<td></td>
</tr>
</tbody>
</table>

TYPE, TYPE₁, TYPEₙ are class names. In the case of an argument declaration, it tells you the type of expected arguments. In the case of a query it denotes the type of the object you get as an answer when calling the query. Note that the only way to distinguish between a query and a command is to look whether a feature returns an object (i.e. look for the return type in its declaration).

The examples given above are from Listing 1 that shows a shortened interface of class TRAFFICBUILDING. The argument declaration of set_size uses a short form for the declaration of its arguments. Instead of stating for each argument that it is of type REAL₆₄, it separates the identifiers by comma (instead of semicolon) and gives the type at the end. The short form can be used whenever there are two or more arguments of the same type appearing one after the other in the declaration. So the declaration set_size (a_width, a_height, a_depth: REAL₆₄) is equivalent to set_size (a_width: REAL₆₄; a_height: REAL₆₄; a_depth: REAL₆₄) and contains_point (a_x: REAL₆₄; a_y: REAL₆₄): BOOLEAN could also be written as contains_point (a_x, a_y: REAL₆₄): BOOLEAN.

Listing 1: Class TRAFFICBUILDING

```java
deferred class interface TRAFFICBUILDING

2

feature

4

center: TRAFFIC_POINT
    -- Center of the building

6
corner_1: TRAFFIC_POINT
    -- Lower left corner of the building

8

ensure

result_exists : Result /= Void

10

contains_point (a_x: REAL₆₄; a_y: REAL₆₄): BOOLEAN
    -- Is point ('a_x', 'a_y') inside building?

12

spotlight
    -- Highlight.

14

ensure -- from TRAFFIC_CITY_ITEM
```

---

# ETHZ D-INFK

Prof. Dr. B. Meyer

Introduction to programming – Assignments

Fall 2008
Todo

In Listing 2 you find the class interface of *TRAFFIC_TIME* that is responsible in traffic for simulating time in the city, measured e.g. for letting passengers move at a certain speed. Make two lists of features for this class interface: one for queries, the other for commands. Use the way described above to distinguish between queries and commands.

Listing 2: Class *TRAFFIC_TIME*

defered class interface TRAFFIC_TIME

feature — All features

pause

require

is_time_running

ensure

not is_time_running

actual_time: TIME

— Simulated time

reset

— Reset the time to (0:0:0).

ensure

is_time_running = False

actual_time.hour = 0

actual_time.minute = 0

actual_time.second = 0

duration (a_start_time, a_end_time: TIME): TIME_DURATION

— Duration from ‘a_start_time’ until ‘a_time2’.

— Takes into account midnight.

require
both_exist: a_start_time /= Void and a_end_time /= Void
ensure
result_exists: Result /= Void
result_positive: Result.is_positive

speedup: INTEGER_32
  -- Speedup to let the time run faster than the real time
set_speedup (a_speedup: INTEGER_32)
  -- Set speedup to 'a_speedup'.
require
  a_speedup_valid: a_speedup >= 1
ensure
  speedup_set: speedup = a_speedup

start
  -- Start to count the time at (0:0:0).
require
  not is_time_running
ensure
  is_time_running

is_time_running: BOOLEAN
  -- Is the time running?

resume
  -- Resume the paused time.
require
  not is_time_running
ensure
  is_time_running

set (a_hour, a_minute, a_second: INTEGER_32)
  -- Sets the time to (`a_hour`:`a_minute`:`a_second`).
require
  valid_time: a_hour >= 0 and a_minute >= 0 and a_second >= 0

invariant
  actual_time.hour >= 0
  actual_time.minute >= 0
end

3  Feature reading

In Task 2 you saw that feature declarations of queries always include the declaration of a return type. The return type is the type of the object that is returned as an answer when calling the query. This knowledge in combination with the fundamental mechanism of program execution - applying a “feature” to an “object” - allows to build complex targets and arguments to feature call instructions. To make it even clearer:

- Queries return a value (an object), e.g. Station_balard.location yields an object of type
TRAFFIC_POINT, the position of Balard. Since the result is an object, it is possible to apply features to it, e.g. Station\_{balard}.location.up\_by(5.0). What features can be applied is defined in the class TRAFFIC_POINT. As a side note: Station\_{balard} is also a query returning an object of type TRAFFIC\_STATION and is declared in class TOUCH\_PARIS\_OBJECTS (why class PREVIEW offers this feature is part of the magic and will be resolved later).

- Similarly, it is possible to use results of queries as arguments, e.g. Console.show(Line8.south\_end)
- The result of an arithmetic expression (say $x \times 3 + 72$) is also an object on which you can call features, e.g. $(x \times 3 + 72).out$

Expressions built using the “.” notation are evaluated from left to right, e.g. $x.y.z.f$ is evaluated as $((x.y).z).f$. This knowledge helps us dissecting feature call instructions. Note that in feature call instructions below the prefix TRAFFIC is omitted.

Example 1

\[
\begin{array}{c|c|c}
\text{target} & \text{command} & \text{arguments} \\
\hline
\text{Line2.color} & \text{set.red} & \text{Line8.color.red} \\
\end{array}
\]

Explanation:

- Line2 is a query defined in class TOUCH\_PARIS\_OBJECTS and returns an object of type TRAFFIC\_LINE.
- In class TRAFFIC\_LINE there is a query color defined that returns an object of type TRAFFIC\_COLOR.
- In class TRAFFIC\_COLOR there is a command set\_red defined. It takes an argument of type INTEGER.
- Line8 is a query defined in class TOUCH\_PARIS\_OBJECTS and returns an object of type TRAFFIC\_LINE.
- In class TRAFFIC\_LINE there is a query color defined that returns an object of type TRAFFIC\_COLOR.
- In class TRAFFIC\_COLOR there is a query red defined that returns an object of type INTEGER.
- The argument is thus an INTEGER that conforms to the type requested by set\_red.

Example 2
Explanation:

- **Route1** is a query defined in class *TOUCH_PARIS_OBJECTS* and returns an object of type *TRAFFIC_ROUTE*.
- In class *TRAFFIC_ROUTE* there is a query **first** defined that returns an object of type *TRAFFIC_LEG*.
- In class *TRAFFIC_LEG* there is a query **destination** defined that returns an object of type *TRAFFIC_STATION*.
- In class *TRAFFIC_STATION* there is a query **stop** defined that returns an object of type *TRAFFIC_STOP* and takes an object of type *TRAFFIC_LINE* as argument.
- **Line8** is a *TRAFFIC_LINE* and thus can be used as such an argument.
- In class *TRAFFIC_STOP* there is a query **line** that returns a *TRAFFIC_LINE*.
- In class *TRAFFIC_LINE* there is a query **south_end** that returns a *TRAFFIC_STATION*.
- And in class *TRAFFIC_STATION* the command **highlight** is defined and thus can be called on the target.

A remark on methodology

Generally, long chains of feature calls are considered bad practice, violating a well known principle called "Don’t talk to strangers" or "Law of Demeter". For a discussion see [http://en.wikipedia.org/wiki/Law_of_Demeter](http://en.wikipedia.org/wiki/Law_of_Demeter). We include this task all the same to show you how to read feature calls properly.

To do

For each of the instructions below, determine the type of the target following the scheme from the examples. You will need to read class declarations, so start EiffelStudio and open the project located under ‘traffic/example/02_objects’.

Note that for certain classes there exist aliases. As an example, *DOUBLE* might appear named as *REAL_64* and *STRING* as *STRING_8* depending on the view you are using to look at the classes in EiffelStudio.

1. **Route2.first.line.extend**(Line7.a.i_th (1)) where **Route2** is of type *TRAFFIC_ROUTE* and **Line7.a** of type *TRAFFIC_LINE*.
2. **Route1.first.next.origin.location.left_by** (20.0) where **Route1** is of type *TRAFFIC_ROUTE*.
3. **Line2.i_th** (Line2.count).stop (Route3.first.line).station.highlight where **Route3** is of type *TRAFFIC_ROUTE* and **Line2** of type *TRAFFIC_LINE*.

Hint

To navigate between classes and features in EiffelStudio, you can use the ‘pick-and-drop’ technique. Just ‘pick’ a class or a feature by holding down the [SHIFT] key and right-clicking on the feature/class name and ‘drop’ it in another pane within EiffelStudio, and see what happens.

To hand in

Your answers to questions 1-5.
4 Writing more feature calls

To do

1. Download http://se.inf.ethz.ch/teaching/2008-H/eprog-0001/exercises/assignment_3.zip and extract it in traffic/example. You should now have a new directory traffic/example/assignment_3 with assignment_3.ecf directly in it. It is important that the location corresponds to the description here!

2. Open and compile this new project.

3. Open the class text of PLANNER which you will change in this task. Assume that you are planning to change the original metro system of Paris (see Figure 1(a)) in the following way: Line1, Line3, Line8, and Line7 all only consist of one connection going from the original south end (south_end) to Concorde (Station/Concorde). Line2 is a cyclic line containing its original south end and the south end stations of Line1, Line3, Line8, and Line7 connected as shown in Figure 1(b).

Hint

To complete the task you need features from the class TRAFFIC_LINE such as remove_all_segments and extend.

In the text editor, when you type the name of an entity followed by a dot, EiffelStudio will automatically display a list of all the features that can be called at the current position (see Figure below). To get the list of almost all features applicable to the Current object, press [CTRL] + [SPACE]. But if you really want to see all the features applicable to the Current object you have to change an option: from the menu Tools/Preferences.../Editor/Eiffel set the ‘Show ANY features’ option to True, and when pressing [CTRL] + [SPACE] you should be able to see, in addition to the others, the most general features, those that can be applied to all objects. Pressing [SHIFT] at the same time will do the same for class names.

To hand in

Submit class PLANNER to your assistant.
Figure 1: Changing the metro system