Assignment 3: Of objects and features

ETH Zurich

Hand-out: 1 October 2010
Due: 11 October 2010

Calvin and Hobbes© Bill Watterson

Goals

- Understand the difference between a *class* and an *object*.
- Learn to read query call chains.
- Write more feature calls.
- Learn to read user input from the console.
- Add new features to a class.

1 Classes vs. objects

To do

1.1 Try to describe the difference between a class and an object (1-2 sentences).

1.2 Find an *analogy* (not an example!) that illustrates the relationship between objects and classes in real life.

To hand in

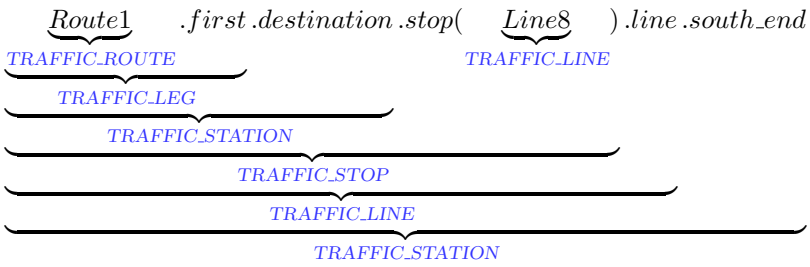
Hand in your answers.

2 Query call chains

As you already know, queries (unlike commands) always return an object. The type of this object can be determined from the signature of the query [Touch Of Class, page 220].

For example, *Route1* is defined in the class *TOUCH_PARIS_OBJECTS* as *Route1*: *TRAFFIC_ROUTE*, which means that it always returns an object of type *TRAFFIC_ROUTE*. This, in turn, means that you can call features defined in class *TRAFFIC_ROUTE* on *Route1*, as in *Route1.first*, which returns an object of type *TRAFFIC_LEG*.

You can form even longer chains of query calls in the same way, as in:



It is also possible to use query call chains as arguments, e.g.

Station_balard.location.distance (*Station_concorde.location.zero*)

To do

For each query call chain below, determine the type of the object it returns (following the example given above). You will need to read class declarations, so start EiffelStudio and open the project located under `traffic/example/02_objects`. The first query in each chain is defined in the class *TOUCH_PARIS_OBJECTS*.

1. *Route2.first.line*
2. *Route1.first.next.origin.location*
3. *Line2.i_th* (*Line2.count*).*stop* (*Route3.first.line*).*station*

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 class (feature) name. The cursor will change shape to an oval (or a thick cross in case you picked a feature). You can then 'drop' it in another tools pane within EiffelStudio by right-clicking again. When this is not possible, a thin red cross appears on the cursor.

To hand in

For the queries 1-3 hand in the type of object they return.

3 Writing more feature calls

Todo

1. Download se.inf.ethz.ch/teaching/2010-H/eprog-0001/assignments/03/assignment_3.zip and extract it into `traffic/example`. You should now have a new directory `traffic/example/assignment_3` with "assignment_3.ecf" directly in it.

2. Open and compile this new project.

3. Open the class text of *PLANNER* which you will change in this task. Change the original metro system of Paris (Figure 1(a)) in the following way (Figure 1(b)): lines 1, 3, 8 and 7a all only consist of one connection going from the original starting terminal (*terminal_1*) to Concorde (*Station_Concorde*). Line 2 is a cyclic line containing its original starting terminal and the starting terminals of lines 3, 7a, 1, and 8.

   Browse the class *TRAFFIC_LINE* to find the features you need. One of them is *remove_all_segments*, which removes all stations from a line except the starting terminal.

To hand in

Hand in the class *PLANNER*.

4 In and out

In this task you will read some input data from the user and then output the processed data back to the user. You will also have to add new features to a class.

Todo

1. Download se.inf.ethz.ch/teaching/2010-H/eprog-0001/assignments/03/business_card.zip and extract it into some directory.

   Open "business_card.ecf" in EiffelStudio. This project consists of a single class *BUSINESS_CARD*; open it in the editor.

2. There is an attribute [Touch Of Class, page 238] *age* in *BUSINESS_CARD*, which is meant to store the age of the card's owner. Add two new attributes of type *STRING* to store his name and job title. Place your new attributes into the feature clause called "Access". Don't forget to format your code properly and add comments.

3. Modify the feature *fill_in* so that it prompts the user for his name (i.e. prints "Your name:" or "Please enter your name:"), then reads the user input from the console and stores it in the attribute you just declared for this purpose.

   Like in the previous assignment, use the predefined object *Io* to perform input and output. To read input data use the commands *read_line*, *read_integer*, *read_real*, etc. To access the data read by the last *read_*xxx command, use the queries *last_string*, *last_integer*, *last_real*, etc., accordingly. For example, the sequence of instructions:

   ```
   Io.read_line
   country := Io.last_string.twin
   ```

   reads a line from the console and stores the result in a string attribute called *country*. The call to *twin* is needed because *STRING* is a reference type (you will learn more about that later in the course); it is not needed when reading an integer.

4. Modify the feature *fill_in* so that after asking the user for his name, it also asks for his job title and his age. Store this information in the corresponding attributes.

5. In the feature clause "Output" declare a new function *text*: *STRING* [Touch Of Class, page 219]. This function should return a string representation of the business card, containing all the data that it stores (name, job and age), each on a separate line.

To assemble a string out of several parts use the string concatenation operator +. For example, an expression `"Hello" + ", " + " world!"` results in a string `"Hello, world!"`. You can put `"%N"` into a string at the position where you want a new line to be printed.

Use the function *age_info* already defined in the class *BUSINESS_CARD* to output information about the age. You can also use *age_info* as an example of a function that returns a string, constructed using concatenation.

Now add the instruction *Io.put_string* (*text*) at the end of the feature *fill_in* to print the string representation of the business card. After this step a run of your program should look similar to this:

```
Your name: John Smith
Your job: Programmer
Your age: 20
John Smith
Programmer
20 years old
```

6. To make the output of your program look more like a business card let's put a border around it. After this step a run of your program should look like to this:

```
Your name: John Smith
Your job: Programmer
Your age: 20
--------------------------------------------------
|John Smith                                      |
|Programmer                                      |
|20 years old                                    |
--------------------------------------------------
```

Modify the function *text* so that it also prints the border. For the top and the bottom part use the function *line* (*n*: *INTEGER*): *STRING* already defined in *BUSINESS_CARD*, which returns a horizontal line of length *n*. Use the constant attribute *Width* [Touch Of Class, page 250] every time you refer to the width of the card.

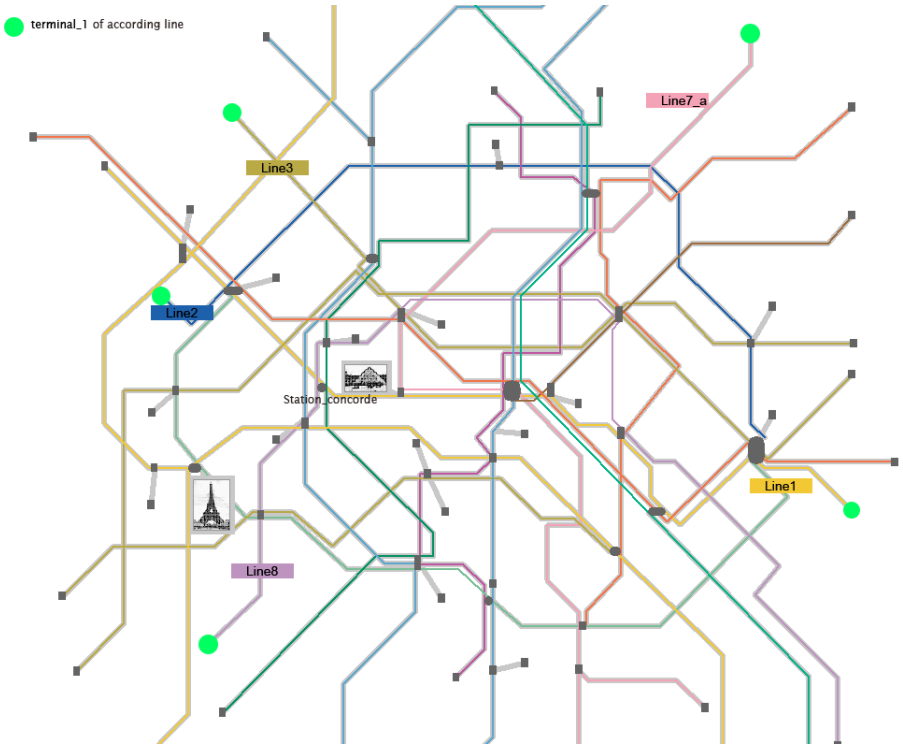
To output the right border you need print a correct number of spaces between it and the text. For this define a new function *spaces* (*n*: *INTEGER*): *STRING*, which would return a string consisting of *n* spaces. Use the function *line* as an example. To get the number of characters in a string use the query *count*.

7. What is the benefit of using the constant attribute *Width* compared to just writing 50 every time?

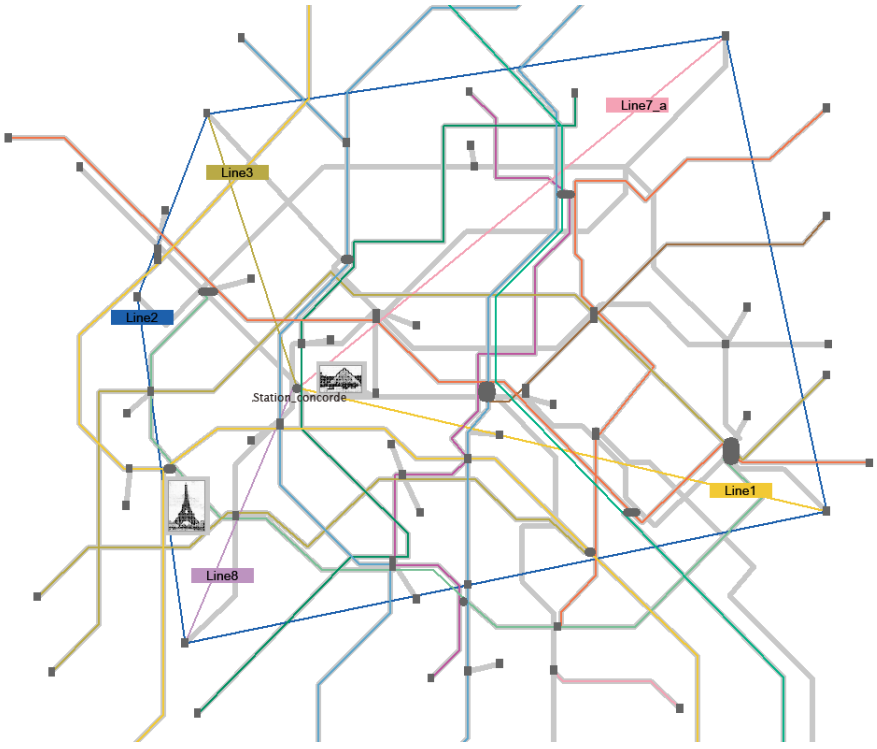
   What will happen with your program if the name entered by the user is longer than *Width*? How could you solve this problem (describe the general idea)?

To hand in

Hand in the code of the class *BUSINESS_CARD* and answers to the questions in point 7.



(a) original metro system



(b) new metro system

Figure 1: Changing the metro system