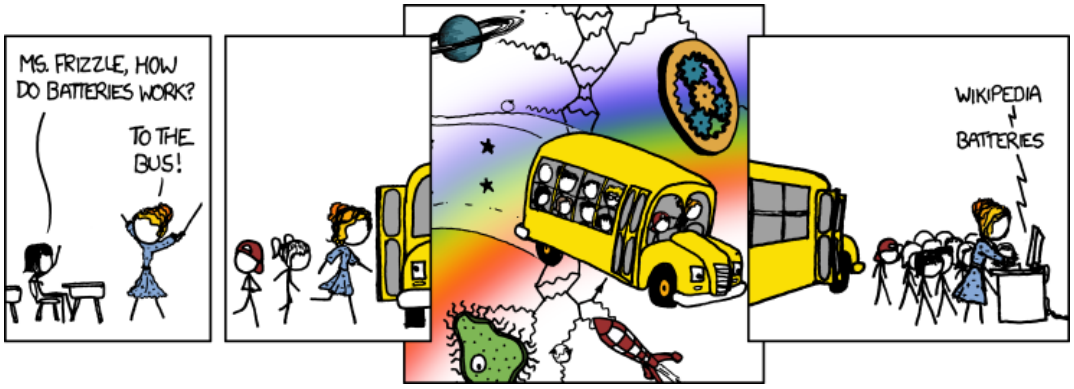
Assignment 3: Of objects and features

ETH Zurich

Hand-out: Friday, 30 September 2011
Due: Tuesday, 11 October 2011



Magic School Bus © Randall Munroe (xkcd.com)

Goals

- Understand the difference between a *class* and an *object*.
- Read and write query call chains.
- Process user input.
- Add new features to a class.

1 Classes vs. objects

To do

1.1 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, *Zurich* is defined in the class *ZURICH_OBJECTS* as *Zurich*: *CITY*, which means that it always returns an object of type *CITY*. This, in turn, means that you can call features defined in class *CITY* on *Zurich*, as in *Zurich.name*, which returns an object of type *STRING*.

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

$$\underbrace{\underbrace{\underbrace{\underbrace{Zurich}_{CITY}.line\ (24)}_{LINE}.west\_terminal}_{STATION}.is\_exchange}_{BOOLEAN}$$

This query call chain answers the question "Is the west terminal of line 24 in Zurich an exchange station?" and returns an object of type *BOOLEAN*.

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

Zurich.station (**"Central"**).*position.distance* (*Zurich.station* (**"Polyterrasse"**).*position*)

This query call chain answers the question "What is the distance between positions of stations Central and Polyterrasse in Zurich?" and returns and object of type *REAL_64* — a 64 bit real number.

To do

1. To answer the questions below you will need to read feature declarations, so download http://se.inf.ethz.ch/courses/2011b_fall/eprog/assignments/03/traffic.zip, unzip it and open `assignment_3.ecf`. All query call chains in this task start with the query *Zurich*, so open *ZURICH_OBJECTS*, where the query is defined.

2. For each query call chain below, determine the type of the object it returns and state informally the question it answers, following the example given above. To understand what individual queries mean, read the header comments in their declaration.

   2.1 *Zurich.line* (5).*kind.name*

   2.2 *Zurich.station* (**"Hardplatz"**).*position.length*

   2.3 *Zurich.line* (2).*distance* (*Zurich.station* (**"Bellevue"**), *Zurich.line* (2).*west_terminal*)

3. For each question below write a query call chain that answers the question. You can test your answers in the feature *explore* of class *PREVIEW* (following the examples that are already there).

   3.1 How bright is the color of line 13?

   3.2 How many meters to the north of the city center is the third station of line 31 located?

   3.3 What is the next station of line 31 after Loewenplatz in the direction of its west terminal?

   3.4 *(Optional)* How many public transportation lines go through station Paradeplatz?

   3.5 *(Optional)* Does line 7 directly connect stations Paradeplatz and Rennweg?

Hint

To navigate between classes and features in EiffelStudio, in addition to the mechanisms described in the previous assignment, 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 2.1–2.3 hand in the type of the object they return and the question they answer; for the questions 3.1–3.5 hand in the queries that answer the question.

3 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.

To do

1. Download http://se.inf.ethz.ch/courses/2011b_fall/eprog/assignments/03/business_card.zip and extract it into a directory of your choice.

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

   BUSINESS_CARD contains three attributes: *name*, *job* and *age*, which are meant to store the corresponding data about the business card owner. The class is also equipped with commands *set_name*, *set_job* and *set_age* that change the values of the corresponding attributes.

2. 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 and stores it in the attribute *name* using the command *set_name*.

   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_character*, *read_real*, etc. To access the data read by the last *read_*x command, use the queries *last_string*, *last_integer*, *last_character*, *last_real*, etc., accordingly. For example, the sequence of instructions:

   ```
   Io.read_line
   set_country (Io.last_string)
   ```

   reads a line and passes it as an argument to a command *set_country*.

3. 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. After this step, a run of your program should look similar to this:

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

4. In the feature clause "Output" declare a new procedure *print_card* [Touch Of Class, page 219]. This procedure should output a textual 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 you can 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.

   Don't forget to format your code properly and add a header comment.

5. Now add a call to *print_card* at the end of the feature *fill_in*. 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 procedure *print_card* so that it also prints the border. For the top and the bottom border 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.