

## DO IT WITH STYLE – A Guide to the Eiffel Style

The following text is a summary of chapter 26 of Bertrand Meyer's book *Object-Oriented Software Construction*, second edition. The whole chapter is available on <http://archive.eiffel.com/doc/manuals/language/style/style.pdf>.

If you have any corrections or comments, please send an e-mail to Marcel Kessler ([kesslema@student.ethz.ch](mailto:kesslema@student.ethz.ch)).

### CHOOSING THE RIGHT NAMES

- For feature and class names, use full words, not abbreviations, call *number*, not *num*.
- Do not hesitate to use several words connected by underscores, as in *ANNUAL\_RATE*.
- For features, there is seldom a need for more than two or possibly three underscore-connected words.
- Do not include in a feature name the name of the underlying data abstraction (which should serve as the class name).
  - The feature giving the part number in class *PART* should be called just *number*, not *part\_number*.
- Sometimes, every instance of a certain class contains a field representing an instance of another class. Although you should try to find a more specific name, you may, if this fails, just declare the feature as *rate: RATE*.
- Local entities and arguments of a routine only have a local scope, so they do not need to be as evocative.

*move* (*i: INTEGER*) **is**

-- Move cursor *i* positions, or *after* if *i* is too large.

**local**

*c: CURSOR; counter: INTEGER; p: like FIRST\_ELEMENT*

...

*remove* **is**

-- Remove current item; move cursor to right neighbor.

**local**

*succ, pred, removed: like first\_element*

...

- If *succ* and *pred* had been features they would have been called *successor* and *predecessor*.

### Letter case

- Class names appear in all upper case: *POINT*, *LINKED\_LIST*...

- Names of attributes, routines etc. appear in all lower case: *balance*, *deposit*, *succ*, *i*.
- Constant attributes have their first letter in upper case and the rest in lower lower case: *Pi: INTEGER is 3.1415926524*; *Welcome\_message: STRING is "Welcome!"*
- A few *reserved words* are written with an initial upper case since they are similar to constants, they include *Current*, *Result*, *Precursor*, *True* and *False*.

## Grammatical categories

- For class names, you should always use a noun, possibly qualified as in *LONG\_TERM\_SAVINGS\_ACCOUNT*.
- Routine names should faithfully reflect the Command-Query separation principle:
  - Procedures (commands) should be verbs in the infinitive or imperative: *make*, *move*, *deposit*, *set\_color*.
  - Attributes and functions (queries) should never be imperative or infinitive verbs; never call a query *get\_value*, but just *value*.
- Non-boolean query names should be nouns, such as *number*.
- A frequent convention for boolean queries is the *is\_* form, as in *is\_empty*.

## HEADER COMMENTS AND INDEXING CLAUSES

Instead of the long comment in

*tangent\_from (p: POINT): LINE is*

*-- Return the tangent line to the circle going through the point p,  
-- if the point is outside of the current circle.*

**require**

*outside\_circle: not has (p)*

...

just write

*-- Tangent from p.*

because of the following reasons:

- The comment for a query, as here, should not start with “Return the...” or “Compute the...”. Simply name what the query returns, typically using a qualified noun.
- We can get rid of the auxiliary words, especially *the*, where they are not required for understandability.
- Another mistake is to have used the words *line* to refer to the result and *point* to refer to the argument: this information is immediately obvious from the declared types, *LINE* and *POINT*.
- Header comments for commands (procedures) should end with a period.

For boolean-valued queries, the comment should always be in the form of a question, terminated by a question mark:

```
has (v: G): BOOLEAN is
    -- Does `v` appear in list?
```

...

- Software entities — attributes, arguments — appearing in comments in the source text should always appear between an opening quote (“backquote”) and a closing quote.

Because an exported attribute should be externally indistinguishable from argumentless functions — remember the Uniform Access principle — it should also have a comment:

```
count: INTEGER
    -- Number of students in course
```

## TEXT LAYOUT AND PRESENTATION

The textual layout of the notation follows a **comb-like structure**; the idea is that a syntactically meaningful part of a class, such as an instruction or an expression, should either:

- Fit on a line together with a preceding and succeeding operators.
- Be indented just by itself on one or more lines.

```
if c then a else b end
```

or

```
if
    c
then
    a
else
    b
end
```

or

```
if c then
    a
else b end
```

## Spaces

You will use a space:

- Before an opening parenthesis, but not after:  $f(x)$ .
- After a closing parenthesis *unless* the next character is a period or semicolon; but not before. Hence:  $proc1(x); x := f1(x) + f2(y)$ .
- After a comma but not before:  $g(x, y, z)$ .

Spaces should appear before and after arithmetic operators, as in  $a + b$ .

## A layout example

### indexing

*description: "Example for formatting"*

### class

*EXAMPLE*

### inherit

*MY\_PARENT*

**redefine** *f1, f2* **end**

*MY\_OTHER\_PARENT*

**rename**

*g1* **as** *old\_g1*, *g2* **as** *old\_g2*

**redefine**

*g1*

**select**

*g2*

**end**

### create

*make*

### feature -- Initialization

*make is*

*-- Do something.*

### require

*some\_condition: correct (x)*

### local

*my\_entity: MY\_TYPE*

### do

**if** *a* **then**

*b; c*

**else**

*other\_routine*

*new\_value := old\_value / (max2 - max1)*

**end**

**end**

### feature -- Access

*my\_attribute: SOME\_TYPE*

*-- Explanation of its role (aligned with comment for make)*

*...*

### invariant

*upper\_bound: x <= y*

### end