

Solution 3: Of objects and features

ETH Zurich

1 Classes vs. objects

There is no unique solution. Sample answers:

- 1.1 A class can be viewed as a software *module* (a piece of source code that contains descriptions of related operations and data), and a *type* (a set of objects that support the same operations).

An object can be viewed as a *collection of data* (whose structure is defined in the object's generating class) and a *member of a type* (an entity in a running program, to which the operations defined in the generating class are applicable).

- 1.2 A class can be viewed as the blueprint of a machine, while an object is the actual machine built according to the blueprint.

2 Query call chains

- 2.1 What is the name of the kind of public transportation that runs on line 5? Type: *STRING*.
- 2.2 What is the distance between station Hardplatz and the city center? Type: *REAL_64*.
- 2.3 What is the distance along line 2 between station Bellevue and the west terminal of the line? Type: *REAL_64*.

3.1 `Zurich.line (13).color.brightness`

3.2 `Zurich.line (31).i_th (3).position.y`

3.3 `Zurich.line (31).next_station(Zurich.station ("Loewenplatz"),
Zurich.line (31).west_terminal)`

3.4 `Zurich.station ("Paradeplatz").lines.count`

3.5 `Zurich.connecting_lines (Zurich.station ("Paradeplatz"),
Zurich.station ("Rennweg")).has (Zurich.line (7))`

3 In and out

Listing 1: Class *BUSINESS_CARD*

```
class
  BUSINESS_CARD

create
  fill_in

feature {NONE} -- Initialization

  fill_in
    -- Fill in the card and print it.
    do
      Io.put_string ("Your name: ")
      Io.read_line
      set_name (Io.last_string)

      Io.put_string ("Your job: ")
      Io.read_line
      set_job (Io.last_string)

      Io.put_string ("Your age: ")
      Io.read_integer
      set_age (Io.last_integer)

      print_card
    end

feature -- Access

  name: STRING
    -- Owner's name.

  job: STRING
    -- Owner's job.

  age: INTEGER
    -- Owner's age.

feature -- Setting

  set_name (a_name: STRING)
    -- Set 'name' to 'a_name'.
    require
      name_exists: a_name /= Void
    do
      name := a_name.twin
    end

  set_job (a_job: STRING)
```

```
    -- Set 'job' to 'a_job'.
  require
    job_exists: a_job /= Void
  do
    job := a_job.twin
  end

  set_age (a_age: INTEGER)
    -- Set 'age' to 'a_age'.
  require
    age_non_negative: a_age >= 0
  do
    age := a_age
  end

feature -- Output
  age_info: STRING
    -- Text representation of age on the card.
  do
    Result := age.out + " years old"
  end

  print_card
    -- Output business card.
  do
    -- Io.put_string (name + "%N" + job + "%N" + age_info + "%N")
    Io.put_string (line (Width + 2) + "%N"
      + "|" + name + spaces (Width - name.count) + "|%N"
      + "|" + job + spaces (Width - job.count) + "|%N"
      + "|" + age_info + spaces (Width - age_info.count) + "|%N"
      + line (Width + 2) + "%N")
  end

  Width: INTEGER = 50
    -- Width of the card (in characters), excluding borders.

  line (n: INTEGER): STRING
    -- Horizontal line on length 'n'.
  do
    Result := "_"
    Result.multiply (n)
  end

  spaces (n: INTEGER): STRING
    -- String consisting of 'n' whitespaces.
  do
    Result := " "
    Result.multiply (n)
  end
end
```

- The main benefit of using the constant attribute is that the width of the card is stored in a single place. Thus, when you want to change it you just have to edit a single constant attribute definition as opposed to searching the whole program text for usages of number 50 and trying to remember, which ones of them actually refer to the width and which ones mean something else and just happen to be equal to 50.

Another benefit is that using a meaningful name for the attribute improves code readability.

- The answer depends on the actual implementation; we give the answer for the master solution.

If the name is too long you will get a precondition violation when calling feature *multiply* of class *STRING* from within the function *spaces*. It happens because we are trying to fill up the line with a negative (or zero) number of spaces.

A solution would be to go through all the lines in the card, calculate the maximum of their lengths and change the width to be larger than this maximum. Otherwise, if there is a good reason for the width to be equal to 50, the *APPLICATION* class should check the user input to be sufficiently short, and if it isn't, either truncate it or ask the user to input a shorter string.

4 Interfaces, commands, and queries

The order in which the questions and the answers appear here in the solution may vary because they are randomly shuffled at each attempt.

Interfaces and Information Hiding Quiz:

- Suppose to have a class X that refers in its code to class Y, and class Y that refers in its code to class X. Which of the following are true (multiple answers are possible)? Class X is a supplier of class Y; Class X is both a client and a supplier of class Y, and vice-versa; Class X is a client of class Y; Class Y is a client of class X; Class Y is a supplier of class X
- Suppose to have class X that is a client of class Y. Which of the following may be true (multiple answers are possible)? Class X has a function with a return type of type Y; Class X has a routine with an argument of type Y; Class X has an attribute of type Y; Class X has a routine with the same name as one in class Y; Class X has a routine with a local variable of type Y.
- Who are the clients and the suppliers of a user interface and of a program interface? In a user interface the clients are people, the supplier is software. In a program interface both the client and the supplier are software.
- Given a class that generates a password, specify which of the following features you would made accessible to all clients (by making them public) which not (by making them secret), and which are the ones for which the decision would depend on the requirements. Procedure `generate_password` (no return type): public. Attribute `password_generating_algorithm_name`: secret. Attribute `generated_password`: public. Attribute `internal_data_structure`: secret. Attribute `password_strength`: it depends.

Commands, Queries, and Uniform Access Principle Quiz:

- Given a class that generates a password, specify which of the following features are commands and which ones are queries. `generate_password` (command); Attribute `password_generating_algorithm_name`: *STRING* (query); Attribute `generated_password`: *STRING* (query); Routine `print_password` (command); Function `compute_strength`: *STRING* (query).

- Imagine that the following feature: `query_database (my_query: QUERY): INTEGER` is used to query a database, returns the result within argument `my_query`, and returns an error code. Which of the following is true (more answers possible)? `query_database` is a query and a command, which means you should also mark the separate answers “query” and “command”.
- Taking the client (specification) view, a feature that yields no result is ... a command; Taking the client (specification) view, a query that yields a result through computation is ... a function; Taking the client (specification) view, a query that yields a result through memory is ... an attribute; Taking the internal (implementation) view, a feature that performs some computation is ...a routine; Taking the internal (implementation) view, a feature that just stores some memory (no computation) is ... an attribute; Taking the internal (implementation) view, a routine that yields no result is ... a procedure; Taking the internal (implementation) view, a routine that yields a result is ... a function.
- Suppose to have an Eiffel class `PERSON` that has a query `age`. Which of the following is true? The fact that clients of class `PERSON` do not need to change the calls to feature `age` if the feature itself changes from being a function to being an attribute or vice versa is an example of the Uniform Access Principle. In Eiffel clients of class `PERSON` do not need to change the calls to feature `age` if the feature itself changes from being a function to being an attribute or vice versa.