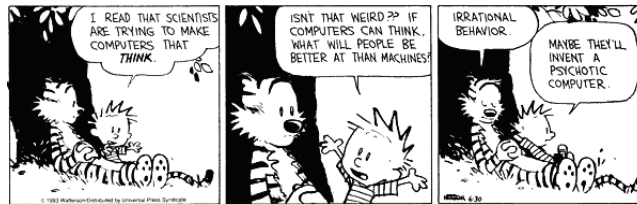


# Assignment 5: References and assignments

ETH Zurich

Hand-out: 17 October 2008

Due: 23 October 2008



Calvin and Hobbes© Bill Watterson

## Goals

- Create more objects in Traffic.
- Make sure you don't have any misconceptions about assignments.
- Write a stand alone application with several classes.

## 1 City building

We have prepared a traffic project that contains a class *CITY\_BUILDING*. In this class, you find four features: *explore*, *add\_station*, *add\_line*, and *random\_color*. The application is programmed to call *add\_station* when you double click with the left mouse button into the city canvas (the white area where the map is usually displayed), and feature *add\_line* when you double click with the right mouse button. At the moment, double clicking will result in a message that is displayed in the Console area of the application, but no station or line is created. In this assignment, you will complete these features to do what their comments promise.

### To do

1. Download [http://se.inf.ethz.ch/teaching/2008-H/eprog-0001/exercises/assignment\\_5.zip](http://se.inf.ethz.ch/teaching/2008-H/eprog-0001/exercises/assignment_5.zip) and extract it in `traffic/example`. You should now have a new directory `traffic/example/assignment_5` with `assignment_5.ecf` directly in it. It is important that the location corresponds to the description here!
2. Open and compile this new project. Open class *CITY\_BUILDING* and solve the tasks below.

3. In feature *explore*, create a new object of type *TRAFFIC\_CITY*. To let the application know that it has to display this city on the screen, you need to use feature *set\_city* of *TRAFFIC\_CITY\_CANVAS* (reachable via the feature *main\_window* in *CITY\_BUILDING*). Add a station called "Central station" at coordinate (0, 0) to the city. We have modified the application to automatically call *explore* at startup, but you can still call it by clicking on the "Run example" button.
4. Implement feature *add\_line* to add a new line to the city. Use the creation feature *make\_with\_terminal* of *TRAFFIC\_LINE*. The line should be of tram type and have the station that you created in step 3 as south and north end.
5. Implement feature *add\_station* to add a new station to the city at the position given through the arguments of *add\_station*. The name of the new station should be "Station *n*" so that the first station created in *add\_station* has name "Station 1", the second "Station 2", etc. Extend the line you created last with the new station. Hint: To make sure that there is a line available you will have to add a call to *add\_line* in *explore*.
6. Since all the created lines have the same default color, it is difficult to distinguish them. Implement the feature *random\_color* and use it to assign a new color to each created line. To achieve this, you can use a class *RANDOM* that generates random numbers for you. The following code illustrates its usage (for more information see [http://www.eiffelroom.com/article/random\\_numbers](http://www.eiffelroom.com/article/random_numbers)):

```

local
  t: TIME
  random: RANDOM
  i: INTEGER
  s: INTEGER
do
  create t.make_now -- Create a time object for the seed
  s := t.hour -- Milliseconds since midnight
  s := s * 60 + t.minute
  s := s * 60 + t.second
  s := s * 1000 + t.milli_second
  create random.set_seed (s) -- Create the random number generator
  -- with 's' as seed
  random.start -- RANDOM is an infinite list: with
  -- 'start' you set it to a first valid position
  i := random.item \ \ 100 -- Access the first random number.
  -- \ \: modulo operator. Make it a number
  -- between 0..99
  random.forth -- Advance the generator to get a new number
end
    
```

7. In your solution, some entities should be local variables and others need to be attributes. Explain when you have used a local variable and when an attribute.

**To hand in**

Submit the class text of *CITY\_BUILDING* to your assistant.

## 2 Assignments

In this assignment you can test your understanding of assignment instructions. Given is the class of *PERSON*:

```
class PERSON
  create make
  feature -- Initialization
    make (s: STRING)
      -- Initialize with 's' as 'name'.
    require
      s_exists: s /= Void and then not s.is_empty
    do
      name := s
    ensure
      name_set: name = s
    end

  feature -- Access
    name: STRING
    loved_one: PERSON

  feature -- Basic operations
    set_loved_one (p: PERSON)
      -- Set 'loved_one' to 'p'.
    do
      loved_one := p
    ensure
      loved_one_set: loved_one = p
    end

  invariant
    has_name: name /= Void and then not name.is_empty
end
```

Below is the code of a feature with a number of declarations and creation instructions:

```
tryout
  -- Tryout assignments
local
  i, j: INTEGER
  a, b, c: PERSON
do
  create a.make ("Anna")
  create b.make ("Ben")
  create c.make ("Chloe")
  a.set_loved_one (b)
  b.set_loved_one (c)
  -- Here the code snippets from below are added
end
```

## To do

On this page, you find a number of subtasks. Each contains a code snippet and statements. Assume that the code snippet is inserted at the location indicated in feature *tryout* above. If the code snippet would produce a compiler error, choose option (a). If it doesn't produce a compiler error, decide for each statement whether it is correct or incorrect after the code snippet has been fully executed. To make the answers easier to read, we use the short form **Anna** for "the object that has the *STRING* "Anna" as *name* attribute", and accordingly **Ben** and **Chloe** for subtasks 6 – 9.

- |    |                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                               |
|----|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. | <pre> j := 3 i := j 2 := i </pre>                                               | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) <i>i</i> has value 2, <i>j</i> has value 3.</li> <li>(c) <i>i</i> and <i>j</i> have both value 2.</li> <li>(d) <i>i</i> and <i>j</i> have both value 3.</li> </ul>                                                                                                                                                     |
| 2. | <pre> i := 7 j := 2 i := i + 3 </pre>                                           | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) <i>i</i> has value 7 and <i>j</i> has value 2.</li> <li>(c) <i>i</i> has value 5 and <i>j</i> has value 2.</li> <li>(d) <i>i</i> has value 10 and <i>j</i> has value 2.</li> </ul>                                                                                                                                     |
| 3. | <pre> i := -7 j := 5 i := j j := i </pre>                                       | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) <i>i</i> has value -7 and <i>j</i> has value 5.</li> <li>(c) <i>i</i> and <i>j</i> have both value -7.</li> <li>(d) <i>i</i> and <i>j</i> have both value 5.</li> </ul>                                                                                                                                                |
| 4. | <pre> j := 8 i := 19 j := i </pre>                                              | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) <i>i</i> and <i>j</i> have both value 19.</li> <li>(c) <i>j</i> has value 19 and <i>i</i> holds no value any more.</li> <li>(d) <i>i</i> and <i>j</i> have both value 8.</li> <li>(e) <i>i</i> has value 8 and <i>j</i> has value 19.</li> </ul>                                                                       |
| 5. | <pre> i := 5 j := i + 7 i := 8 </pre>                                           | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) <i>i</i> and <i>j</i> have both value 8.</li> <li>(c) <i>i</i> has value 8 and <i>j</i> has value 12.</li> <li>(d) <i>i</i> has value 8 and <i>j</i> has value 15.</li> </ul>                                                                                                                                          |
| 6. | <pre> b := a a := b </pre>                                                      | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) <i>a</i> and <i>b</i> are both attached to <b>Ben</b>.</li> <li>(c) <i>a</i> is a void reference and <i>b</i> is attached to <b>Anna</b>.</li> <li>(d) <i>b</i> is attached to <b>Anna</b> and <i>a</i> to <b>Ben</b>.</li> <li>(e) <i>a</i> and <i>b</i> are both attached to <b>Anna</b>.</li> </ul>                 |
| 7. | <pre> b := a.loved_one b.set_loved_one (a.loved_one) a.set_loved_one (c) </pre> | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) The attribute <i>loved_one</i> of <b>Ben</b> references <b>Ben</b>.</li> <li>(c) <i>b</i> is attached to <b>Chloe</b>.</li> <li>(d) <i>a</i> is attached to <b>Anna</b> and <i>b</i> to <b>Ben</b>.</li> <li>(e) <i>b</i> is attached to <b>Anna</b> and <i>a</i> to <b>Chloe</b>.</li> </ul>                          |
| 8. | <pre> b := c b.loved_one := a.loved_one </pre>                                  | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) <i>b</i> is attached to <b>Chloe</b> and its attribute <i>loved_one</i> references <b>Ben</b>.</li> <li>(c) The attribute <i>loved_one</i> of <b>Chloe</b> references <b>Ben</b>.</li> <li>(d) <i>b</i> is attached to <b>Ben</b> and <i>c</i> to <b>Chloe</b>.</li> </ul>                                             |
| 9. | <pre> b := b.loved_one.loved_one a.set_loved_one (c) </pre>                     | <ul style="list-style-type: none"> <li>(a) The compiler reports an error.</li> <li>(b) <i>b</i> is attached to <b>Chloe</b>.</li> <li>(c) <i>b</i> is <b>Void</b> and the attribute <i>loved_one</i> of <i>a</i> is attached to <b>Chloe</b>.</li> <li>(d) <i>a</i> is attached to <b>Anna</b> and <i>b</i> to <b>Ben</b>.</li> <li>(e) The object with name <b>Ben</b> is not reachable any more.</li> </ul> |

## To hand in

Hand in your solution to the questions above.

## 3 Currency conversion

### To do

In this task you will write a stand-alone application which works with money and currencies. The specifications are listed below:

- The application should consist of three classes: *CURRENCY*, *MONEY* and *APPLICATION*.
- Class *APPLICATION* is the root class of your system.
- Class *CURRENCY* stores a smallest unit (e.g. 0.05 for Swiss francs or 0.01 for Euro), the name of the currency, and offers a feature *rounded* that, given a number as argument, returns a rounded down number in the right format for the currency. For example, assume that the currency is CHF with the *smallest\_unit* set to 0.05 and *rounded* is called with 17.68 as argument, then it should return 17.65 – the next lower value in steps of *smallest\_units*. So the class *CURRENCY* should include the following features:

```
* make (s: STRING; r: REAL)  
  -- Initialize with unit 's' and smallest unit to 'r'.  
  -- This is a creation procedure.
```

```
* unit: STRING  
  -- Currency name
```

```
* smallest_unit: REAL  
  -- Smallest unit for this currency (e.g. 0.05 for CHF)
```

```
* rounded (i: REAL): REAL  
  -- i cut off at the appropriate smallest unit.
```

- Class *MONEY* has a currency and is always in the format defined by the currency. Additionally, it offers a feature that lets you convert the money to another currency given the new currency and an exchange rate. So, class *MONEY* should include the following features:

```
* make (c: CURRENCY; a: REAL)  
  -- Initialize with currency and amount.  
  -- This is a creation procedure.
```

```
* currency: CURRENCY  
  -- Currency of the money
```

```
* amount: REAL  
  -- Value of the money
```

```
* convert_to (c: CURRENCY; rate: REAL)  
  -- Convert money to currency 'c' with 'rate'.
```

- Class *APPLICATION* uses your class *MONEY* and *CURRENCY*, creates objects of these types and uses their features.
- Class *APPLICATION* reads input from the command prompt and outputs information on the currencies and money. You could for example write an application that asks the user to input exchange rates for the purchase and the sale of a currency such that he can find out how much money he loses by first buying and then reselling foreign money. Information on current exchange rates are available on the internet.

## Hints

Task 3 of assignment 4 contains explanations for creating a new project and for input and output reading. Also have a look at the section "Things you need to know". The class *REAL* (for floating point numbers) offers a feature called *floor* that returns the rounded down value as an *INTEGER*.

## To hand in

Submit the three class files.