Assignment 4: Object creation

ETH Zurich

Hand-out: 13 November 2006
Due: 21 November 2006

1 Summary

Today you are going to create a stand-alone program.

How to create objects

To create an object represented by an entity

\[ x : T \]
• use the basic creation instruction `create x`, if no creation clause appears in class T, i.e. T does not list any creation procedures. (In this case, default_create is used implicitly.)

• use `create x.make (...)`, if `make` is among the procedures appearing in the creation clause of T.

2 My first application

Goal
• Write your first application from scratch.
• Introduce and use a new class.
• Learn about the basic input and output operations in Eiffel.

Description
Write an application which converts temperatures between Celsius, Fahrenheit, and Kelvin units. The application should consist of two classes: `TEMPERATURE` and `TEMPERATURE.APPLICATION`. The latter is the root class.

Things you need to know
• To print something in the console window, use `io.put_string`, `io.put_integer`, `io.put_boolean` and so on, depending on the type of the argument.

• To read user input, use `io.read ...`. This places the read value in a buffer. Use `io.last ...` to retrieve the value from the buffer.

• The formulas for conversion:

\[
\text{Fahrenheit} = \left(\frac{9}{5}\right) \times \text{Celsius} + 32
\]

\[
\text{Kelvin} = \text{Celsius} + 273.15
\]

Input/Output example
To read an `INTEGER` from the console, and display it on the screen, you might do something like that:

```eiffel
f is
-- Read integer and display it.
local
  i: INTEGER
do
  io.read_integer
  i := io.last_integer
  io.put_integer (i)
end
```
To do

• Option 1: Solving the exercise with Tooter.
  – Download and install the tool. You will find the link on the course website. The instructions on how to install and use Tooter are provided there.
  – Launch Tooter and choose the Temperature exercise.
  – With Tooter’s guidance, work on the tasks it presents until the whole exercise is solved. Tooter will help you implement the interface given in figure 3.

• Option 2: Writing a solution from scratch in EiffelStudio.
  – Launch EiffelStudio. Create a new project of type “Basic application (no graphics library included)”, using the settings shown in figure 1.

  ![Choose Your Project Name and Directory](image)

  Figure 1: New project

  – Enable all the assertions: Project Settings → Assertions

• Add a new class TEMPERATURE to your project, using one of the buttons highlighted in figure 2.

• Implement the class TEMPERATURE using the interface given in figure 3.

• Do not forget to add contracts.
• Feature `make` in class `TEMPERATURE_APPLICATION` should use the `TEMPERATURE` class to do the following:

1. Ask the user to enter a temperature in Celsius.
2. Create a temperature object with the input value.
3. Convert the temperature to Fahrenheit and display it.
4. Convert the temperature to Kelvin and display it.
5. Repeat points 1–4 for temperatures in Kelvin and Fahrenheit.

Example

The execution of your application should yield the result shown in figure 4.

![Figure 2: New class](image-url)
class
    TEMPERATURE
create
    make_with_celsius,
    make_with_fahrenheit,
    make_with_kelvin

feature -- -- Initialization
    make_with_celsius (a_value: DOUBLE)
    -- -- Create Celsius temperature.
    make_with_fahrenheit (a_value: DOUBLE)
    -- -- Create Fahrenheit temperature.
    make_with_kelvin (a_value: DOUBLE)
    -- -- Create Kelvin temperature.

feature -- -- Access
    value: DOUBLE
    -- -- Temperature value
        unit: STRING
        -- -- Temperature unit

feature -- -- Status report
    is_celsius : BOOLEAN
    -- -- Is temperature in Celsius?
    is_fahrenheit : BOOLEAN
    -- -- Is temperature in Fahrenheit?
    is_kelvin : BOOLEAN
    -- -- Is temperature in Kelvin?

feature -- -- Conversion
    celsius_to_fahrenheit : TEMPERATURE
    -- -- Temperature converted from Celsius to Fahrenheit
    celsius_to_kelvin : TEMPERATURE
    -- -- Temperature converted from Celsius to Kelvin
    fahrenheit_to_celsius : TEMPERATURE
    -- -- Temperature converted from Fahrenheit to Celsius
    fahrenheit_to_kelvin : TEMPERATURE
    -- -- Temperature converted from Fahrenheit to Kelvin
    kelvin_to_celsius : TEMPERATURE
    -- -- Temperature converted from Kelvin to Celsius
    kelvin_to_fahrenheit : TEMPERATURE
    -- -- Temperature converted from Kelvin to Fahrenheit

feature -- -- Output
    display
    -- -- Display (on the console) value followed by unit and new line.

end

Figure 3: Interface to implement
Remarks

- Temperature objects have obvious invariants; class \textit{TEMPERATURE} should capture them.

- Write your code following Eiffel style rules:

\footnotetext[1]{\url{http://se.inf.ethz.ch/teaching/ws2006/0001/exercises/style.pdf}}

To hand in

Submit class files \texttt{temperature.e} and \texttt{temperature_application.e}. (If you used Tootor, you’ll find both class files in the subdirectory \texttt{TutorProjects\Temperature} of your Tootor directory.) Don’t forget to upload your EiffelStudio logs (and your Tootor logs if you used the tool).
3 It’s Logic!

Goal

- Understand the difference between strict and non-strict boolean operators.

To do

1. Describe the difference between non-strict and strict boolean operators.
2. Explain when you would prefer non-strict operators over strict operators and give an example for:
   - and
   - and then
   - or
   - or else

To hand in

Hand in your solution to the two questions above.