Assignment 4: Object creation

ETH Zurich

Hand-out: 15 November 2005
Due: 22 November 2005

1 Summary

Today you are going to write your first stand-alone program. Please create the solution to this assignment alone...
How to create objects

To create an object, if you declare:

\[ z : T \]

- If class T has no creation clause (i.e. uses `default_create`), use the basic form: `create z`
- If class T has a creation clause listing one or more procedures, use: `create z.make (...)`, where `make` is one of the creation procedures, and `(...)` stands for arguments, if any.

2 My first application

Goal

- Write your first application from scratch.
- Create and use a new class.
- Learn about basic input/output in Eiffel.

Description

In this exercise you will write your first application from scratch. You will probably find it not that fancy and it cannot do a lot, but Hey!, it might be your first complete program... You have to write an application that can convert temperatures between Celsius, Fahrenheit and Kelvin. The application should consist of two classes, one called `TEMPERATURE` and the other called `TEMPERATURE_APPLICATION` (the root class).

Things you need to know

- To print something to the console window, use `io.put ...`
- Tip: Use `<CTRL> + <SPACE>` to see the different possibilities.
- To get user input, use `io.read ...` to read input into a buffer.
- ... followed by `io.last ...` to access the last read element.
- and most important, the magic formulas:
  \[
  Fahrenheit = (9/5) \times Celsius + 32 \\
  Kelvin = Celsius + 273.15
  \]

Input/Output example

If you wanted to read an `INTEGER` from the console, and then to print it again on the screen, you might do something like Listing 1.
f is
    -- An input/output example..
local  i: INTEGER
do
    io.read_integer
    i := io.last_integer
    io.put_integer(i)
end

Figure 1: Input/output example

To do
- Open EiffelStudio. For Linux users, start EiffelStudio using estoudio, not estudio& (ie. no '&')
- In the opening dialog, create a new project of type Basic application (no graphics library included).
- ... (if this dialog does not appear, select File -> New project).
- Enter the values from Figure 2
- Important: make sure that you enable all the default assertions: Project Settings -> General
• For windows users: enable the Console Application option in Project Settings → Advanced

• Create a new class using one of the buttons highlighted in Figure 3

• Call the new class TEMPERATURE, see Figure 4

• Implement the class TEMPERATURE to supply the class interface in Listing 5 (i.e. provide bodies for the routines and creation procedures to all class elements where they apply).

• Do not forget to add contracts to the interface of your classes. There are a lot of obvious preconditions, postconditions and invariants that have to hold for the class TEMPERATURE.

Figure 3: New class
Finally the feature make of the root class `TEMPERATURE_APPLICATION` should use the `TEMPERATURE` class as a client to do the following:

- Ask user to enter a temperature in Celsius.
- Create temperature object with input value.
- Convert the temperature to Fahrenheit and display it.
- Convert the temperature to Kelvin and display it.
- Do the same thing for a temperature in Kelvin and a Temperature in Fahrenheit.

**Example**

Executing your application should look like Figure 6.
class interface
TEMPERATURE
create
make_with_celsius,
make_with_fahrenheit,
make_with_kelvin
feature -- Initialization
  make_with_celsius (a_value: DOUBLE)
    -- Create with 'a_value' of unit Celsius.
  make_with_fahrenheit (a_value: DOUBLE)
    -- Create with 'a_value' of unit Fahrenheit.
  make_with_kelvin (a_value: DOUBLE)
    -- Create with 'a_value' of unit Kelvin.
feature -- Access
  value: DOUBLE
    -- Temperature value
  unit: STRING
    -- Unit of the temperature value
feature -- Status report
  is_celsius: BOOLEAN
    -- Is temperature value in Celsius?
  is_fahrenheit: BOOLEAN
    -- Is temperature value in Fahrenheit?
  is_kelvin: BOOLEAN
    -- Is temperature value in Kelvin?
feature -- Conversion
  celsius_to_fahrenheit : TEMPERATURE
    -- Converted temperature in Fahrenheit from Celsius
  celsius_to_kelvin : TEMPERATURE
    -- Converted temperature in Kelvin from Celsius
  fahrenheit_to_celsius : TEMPERATURE
    -- Converted temperature in Celsius from Fahrenheit
  fahrenheit_to_kelvin : TEMPERATURE
    -- Converted temperature in Kelvin from Fahrenheit
  kelvin_to_celsius : TEMPERATURE
    -- Converted temperature in Celsius from Kelvin
  kelvin_to_fahrenheit : TEMPERATURE
    -- Converted temperature in Fahrenheit from Kelvin
feature -- Output
  display
    -- Display the temperature value followed by the unit
    -- into the console, terminated by a new line.
end -- class TEMPERATURE

Figure 5: Class interface to implement
Remarks

- Temperatures have a lot of properties that almost shout for preconditions and even invariants ...

- Try to build your classes according to the EIFFEL style rules.

- A short overview of these rules:


To hand in

Hand in your two classes `temperature.e` and `temperature_application.e`. Don’t forget to upload your learning logs.
3 It's Logic!

Goal
- Understand non-strict vs. 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.