Concurrent Object-Oriented Programming

Prof. Dr. Bertrand Meyer

Exercise Session 1: Eiffel Introduction
Overview

- Eiffel
- EiffelStudio
Eiffel: Classes

class NUMERIC_COMPARABLE_VALUE_HOLDER

inherit A

inherit B

end
Eiffel: Features

inherit
...

feature {APPLICATION, NUMERIC_COMPARABLE_VALUE_HOLDER}
value: INTEGER assign set_value

set_value (a_value: like value)
  do
    value := a_value
  end

sum_of_values (a_value: like value): like value
  local
    l_value: like value
  do
    l_value := value + a_value
    Result := l_value
  end

export status: The default is ANY.

attribute with assigner

procedure routine

anchored type

function routine

Routines (procedures, functions) can have local variables.

current object
Eiffel: Creation Procedures

inherit
  ...

create {APPLICATION, NUMERIC_COMPARABLE_VALUE_HOLDER}
  set_value

feature
  ...

export status: The default is ANY.

list of procedure to be used as creation procedures
sum_of_value_holders (a_value_holder: like Current): like Current

local

   l_value_holder: like Current

   do
      create l_value_holder.set_value (value + a_value_holder.value)
      Result := l_value_holder
   end
Eiffel: Further Instructions

multiplication_of_values (a_value: like value): like value

local
    l_counter: like value

do
    from
        Result := value.zero
        l_counter := value.zero
    until
        l_counter.is_equal (a_value)
    loop
        if a_value >= value.zero then
            Result := Result + value
            l_counter := l_counter + value.one
        else
            Result := Result - value
            l_counter := l_counter - value.one
        end
    end
end
feature

... set_value (a_value: value)

  require

    value_is_not_zero: not a_value.is_equal (value.zero)

  do

    ...

  ensure

    value_is_set: value = a_value

end

invariant

  value_is_not_zero: not value.is_equal (value.zero)
class NUMERIC_COMPARABLE_VALUE_HOLDER
[T -> {NUMERIC rename is_equal as numeric_is_equal end, COMPARABLE}]

... feature
value: T assign set_value

... end

l_integer_value_holder: NUMERIC_COMPARABLE_VALUE_HOLDER[INTEGER]
create l_integer_value_holder.set_value (1)
Eiffel: Console I/O

- `io.read_integer` to read an integer from the console.
- `io.put_integer (io.last_integer)` to write an integer to the console.
- `io.new_line` to start a new line on the console.

Read an integer from the console and make it accessible through `io.last_integer`.
Write an integer to the console.
Start a new line on the console.
Eiffel: Deferred Classes and Features

defered class NUMERIC_COMPARABLE_VALUE_PRINTER
[T -> {NUMERIC rename is_equal as numeric_is_equal end, COMPARABLE}]

inherit
NUMERIC_COMPARABLE_VALUE_HOLDER[T]

feature
print_value
  deferred
  end
end

print_value_multiple_times (a_number: NATURAL)
local
  l_counter: like a_number
  do
    from
      l_counter := a_number.zero
    until
      l_counter = a_number
    loop
      print_value
      io.put_new_line
      l_counter := l_counter + a_number.one
    end
  end
end

A deferred class can have effective features which rely on deferred features.
Eiffel: Frozen and Expanded Classes

frozen expanded class INTEGER

inherit

    NUMERIC

    ... 

    COMPARABLE

    ... 

feature

    ... 

der
EiffelStudio

- EiffelStudio is an IDE for Eiffel.
- components
  - editor
  - context tool
  - clusters pane
  - features pane
  - compiler
  - project settings
  - ...

EiffelStudio: Writing Code

- syntax highlighting
- class name completion (SHIFT+CTRL+space)
- smart indenting
- block indent or exdent (TAB and SHIFT+TAB)
- block commenting or uncommenting (CTRL+K and SHIFT+CTRL+K)
- infinite level of undo/redo (reset after a save)
- quick search features (first CTRL+F to enter words then F3 and SHIFT+F3)
EiffelStudio: Compiling

- uses incremental compilation
  - freezing: Generates C code from the whole system and then compiles it to machine code. This code is used during development. Initially the system is frozen.
  - melting: Generates bytecode for the changed parts of the system. This is much faster than freezing. This code is used during development.
  - finalizing: Creates an executable production version. Finalization performs extensive time and space optimizations.
EiffelStudio: Debugging

• The system must be melted/frozen (finalized systems cannot be debugged).

• Set / delete breakpoints. An efficient way of adding breakpoints consists in dropping a feature in the context tool.

• Run the program or step over / into the first statement. Pause or wait for a triggered breakpoint.

• Analyze the program.
  • Use the object tool and the call stack pane.
  • Run the program or step over / into the next statement.

• Stop the running program.
Resources

• http://www.eiffel.com/
• http://docs.eiffel.com/