Einführung in die Programmierung

Introduction to Programming

Prof. Dr. Bertrand Meyer

Exercise Session 4
Today

- Review of concepts from last week
- Aliasing
- Contracts
Review of Concepts from Last Week

- Two kinds of types
  - Basic type and reference type (Java)
  - Expanded type and reference type (Eiffel)
- Stack vs. heap
- Are these entities on the stack or the heap?
  - Local variables (of different types)
  - Arguments
  - Attributes
- What is “information hiding”/“encapsulation” about?
- Why are creation procedures necessary?
Aliasing

```ruby
class BUSINESS_CARD
  feature
    fill_in
      do
        Io.read_line
        set_name(Io.last_string)
        Io.read_line
        set_job(Io.last_string)
        Io.put_string(name + "%N")
        Io.put_string(job + "%N")
      end
end

set_name(a_name: STRING)
  require a_name /= Void
  do
    name := a_name
  end

set_job(a_job: STRING)
  require a_job /= Void
  do
    job := a_job
  end
```

```ruby
class STD_FILES
  ... read_line
  do
    ... ensure
      last_string /= Void
  end
end

set_name(a_name: STRING)
  require a_name /= Void
  do
    name := a_name.twin
  end

set_job(a_job: STRING)
  require a_job /= Void
  do
    job := a_job.twin
  end
```

Tom Singer
Singer
Singer

Tom Singer
Tom Singer
Aliasing and ownership
twin, copy, deep_twin, and deep_copy

- **twin**: A shallow copy where references are duplicated. The references remain the same, but the values are copied.
- **copy**: A shallow copy where references are duplicated. The references remain the same, but the values are copied.
- **deep_twin**: A deep copy with references to the same values. Modifying one does not affect the other.
- **deep_copy**: A deep copy where both references and values are duplicated, making it completely independent.
Contracts and finalization

- Contracts specify properties to be satisfied during execution.
- The program should make sure all the contracts are satisfied.
- Removing the contracts in finalized code shouldn’t make any difference.

```ruby
class BUSINESS_CARD
  create fill_in
  feature {NONE} -- Initialization
    fill_in
    do
      ...
    Io.read_integer
    set_age (Io.last_integer)
    ...
  end

  feature -- Access
  age: INTEGER

  feature -- Setting
  set_age (a_age: INTEGER)
    require
      age_non_negative: a_age >= 0
    do
      ...
  end

  if Io.last_integer >= 0 then
    set_age (Io.last_integer)
  else
    -- abort
  end
```
Contracts: hands-on exercise
Specifying Software through Contracts

In the Western calendar, a year is divided into 12 months, numbered from 1 to 12, and months into days, numbered starting from 1. The 1st, 3rd, 5th, 7th, 8th, 10th, and 12th months have 31 days each, while the 4th, 6th, 9th, and 11th months have 30 days each. The 2nd month usually has 28 days, but it has 29 days in leap years. A year is a leap year if and only if either it can be divided by 4 but not by 100, or it can be divided by 400. For example, the years 2000 and 2004 are leap years, but the years 2011 and 2100 are not.

The following deferred class DATE defines a simple interface for dates in the Western calendar. Please fill in the missing contracts (preconditions, postconditions, and class invariants) of the class; the contracts must reflect all the details given in the informal specification above. They also must ensure that the following client procedure always executes without contract violations:

```plaintext
client (d: DATE)
    require
        d /= Void
    local
        i: INTEGER
    do
        d.set (2011, 8, 22)
        check not d.is_leap (2011) end
        check d.is_leap (2012) end
        from i := 1 until i > 366 loop
            d.proceed
            i := i + 1
        end
        check d.year = 2012 and d.month = 8 and d.day = 22 end
    end

Please note that the number of dotted lines is not indicative of the number of missing assertions (contract elements).
```

defered class
    DATE

    feature -- Access

        year: NATURAL
            -- Year of the date.
            deferred
        end

        month: NATURAL
            -- Month of the date.
            deferred
        end

        day: NATURAL
            -- Day of the date.
            deferred
        end

    feature -- Status set
set (y, m, d: NATURAL)
   -- Set 'year', 'month', and 'day' using 'y', 'm', and 'd', respectively.
require

feature -- Auxiliary queries
is_long_month (m: NATURAL): BOOLEAN
   -- Does month 'm' have 31 days?
defered
ensure

end

is_short_month (m: NATURAL): BOOLEAN
   -- Does month 'm' have 30 days?
defered
ensure

end
end

days_in_month (y, m: NATURAL): NATURAL
  -- Number of days in month 'm' of year 'y'.
require

ensure

deferred
ensure

is_leap (y: NATURAL): BOOLEAN
  -- Is 'y' a leap year?
deferred
ensure

end

feature -- Basic operation

proceed
  -- Proceed to the next date.
  -- For example, if 'Current' represents the date Dec. 31, 2011 before the call,
  -- then it represents Jan. 1, 2012 afterwards.
deferred
ensure
end

— Other features omitted for brevity.

invariant

end