Announcement: lab sessions

- Every Friday (25 November - 16 December 2005)
- IFW D31, 12:00 a.m. - 2:00 p.m.
- During the second hour there will be two assistants answering your questions

Reminder: field study

- You are encouraged to take part in Marie-Helene Ng Cheong Vee’s field study about learning programming.
- Just submit your Eiffel log files regularly.
- For any questions contact your assistant.

Introduction to Programming
Bertrand Meyer

Lecture 7: References & assignment

Last revised 18 November 2005

Classes and objects

- At run time: objects (software machines)
- In the program text: classes

Each class describes a set of possible run-time objects.

Object structure

An object is made of fields
Each field is a value, which is either:

- A basic value: integer, character, “real” number...
  (known as an expanded value)
- A reference to another object

METRO_STOP

SEGMENT

METRO_STATION

POSITION

x

y

23.5

-34.9
**Two kinds of type**

- **Reference** types: value of an entity is a reference.
  Example:
  \[
  \odot \text{POINT}
  \]

- **Expanded** types: value of an entity is an object.
  Example:
  \[
  \odot \text{expanded POINT}
  \]

**Expanded classes**

- A class may also be declared as an expanded class \( C \)
  ... The rest as usual ...

- Then you can declare:
  \[
  d : \text{expanded } C
  \]
  with the same effect as
  \[
  d : C
  \]
  in the earlier syntax (still permitted, with same meaning).

**Basic types as expanded classes**

- expanded class `INTEGER` ...
- expanded class `BOOLEAN` ...
- expanded class `CHARACTER` ...
- expanded class `REAL` ...
- expanded class `DOUBLE` ...

\[
\text{n: INTEGER}
\]

**Initialization**

- Automatic initialization rules:
  - 0 for numbers (integers, reals)
  - "Null" character for characters
  - False for booleans
  - Void for references

- These rules apply to:
  - Fields (from class attributes), on object creation
  - Local variables, on start of routine execution (includes Result)

**References may cause cycles**

- The `name` field is a reference field

**Strings are objects**

- The `name` field is a reference field
**Fields reflect attributes of the class**

```
class POSITION
  feature - Access
    x: REAL  -- Horizontal position
    y: REAL  -- Vertical position
  end
```

Attributes are features of the class

**Setting fields (in routines of the class)**

```
class POSITION
  feature - Access
    x: REAL  -- Horizontal position
    y: REAL  -- Vertical position
  feature - Element change
    set(xval, yval: REAL) is
      require
        x_positive: xval >= 0
      do
        x := xval
        y := yval
      ensure
        x := xval
        y := yval
  end
end
```

**What you may do**

```
class METRO_STATION
  feature
    x, y: REAL  -- Coordinates of metro station
    size: REAL  -- Size of bounding square
    upper_left: POSITION  -- Upper-left position of bounding square
  end

  size
  "Station"

  bounding square

  adjust_positions is
    do
      upper_left.set(x := size/2, y := size/2)
    end
end
```

**Feature calls**

In class POSITION, we assume the feature

```
set(xval, yval: REAL) is
  require
    x_positive: xval >= 0
  do
    x := xval
    y := yval
  ensure
    x := xval
    y := yval
  end
```

From another class, e.g. METRO_STATION

```
adjust_positions is
  do
    upper_left.set(x := size/2, y := size/2)
  end
end
```

 qualify call

From class POSITION itself:

```
moving (dx, dy: REAL) is
  require
    move horizontally, dy vertically
  do
    (Please complete) ...
  ensure
    set(x := x + dx, y := dy)
  end
```

unnatural call

**The current object**

At every moment during execution, operations are being performed on a current object.

Initially: the root object. Then:
- An unqualified call such as `set(x, y)` applies to the current object.
- A qualified call such as `x.set(u, v)` causes the object attached to `x` to become the current object. After the call the previous current object becomes current again.

To denote the current object: use `Current`

**Executing a system**

```
Root object
  root

  Root procedure
    create obj1, obj2, obj3
```

create obj1, obj2
The system execution process

- At any time during execution, there is a current object, on which the current feature is being executed.
  - Initially it is the root object.
  - During a "qualified" call $x.f(a)$, the new current object is the one attached to $x$.
  - At the end of such a call, the previous current object resumes its role.

Feature calls

- In `METRO_STATION`:
  `adjust_positions` is
  ```
  do
  upper_left : set(x - size/2, y + size/2)
  end
  upper_left : POSITION
  ```

The client relation

- Because class `METRO_STATION` has a feature `upper_left : POSITION` (and calls of the form `upper_left : set(...)`),
  `METRO_STATION` is a client of class `POSITION`.

Client and inheritance, graphically

Entities

- An entity is a name in the program that denotes possible run-time values.
  - Some entities are constant.
  - Others are variable:
    - Attributes
    - Local variables
Changing variable values: **assignment**

```
| target := source |
```

**source** is an expression:
- Call to a query:
  - position
  - upper_left, position
- Arithmetic or boolean expression:
  - \( a = (b + c) \)
  - \( a = \beta_0 \) and \( c = \alpha \)

**target** may be:
- An attribute
- Result in a function
- A “local variable” of a routine (not yet seen)

---

Assignment

- Replaces a value by another

```
| \( x \rightarrow 0 \) |
| \( y \rightarrow 0 \) |
```

\( p \).set(2, 1)```

---

Assigning fields (in routines of the class)

```
class MOVE_PHONE
feature: Access
  x: REAL
  y: REAL
    position
    horizontal position
    vertical position
feature: Element change
  set(xval, yval, REAL) is
    require xval, yval then...
    ensure x_set := xval
      y_set := yval
end
```

Do not confuse assignment with equality

```
x := y
```

```
if x == y then...
```

if \( x = \text{Current} \) then...

---

Effect of an assignment

- Reference types: reference assignment
- Expanded types: value copy

```
class MOVE_PHONE
feature
  set(x: INTEGER, y: REAL, r: REAL)
    require x
    x_set := x
end
```

---

Assignment

```
class METRO_STATION
feature
  location: POSITION
  name: STRING
  length: REAL
  set_all(p: POSITION, l: REAL, n: STRING) is
    location := p
    length := l
    name := n
end
```
A linked list of strings: inserting at the end

```
extend(v: STRING) is
  -- Add v to end.
  -- Do not move cursor.
  local
    p: LINKABLE[STRING]
  do
    create p, make(v)
    if is_empty then
      first_element := p
      active := p
    else
      last_element.put_right(p)
      if after then active := p end
    end
  count := count + 1
end
```

### Linkable cells

```
class LINKABLE feature
  item: STRING
    -- Value in this cell
  right: LINKABLE
    -- Cell, if any, to which this one is chained
    put_right(other: like Current) is
      -- Put other to the right of current cell.
      do
        right := other
        ensure
          chained: right = other
      end
end
```

### Local variables (in routines)

- A form of entity (they are also called "local entities")
- Just declare them on entry to a routine:
  ```
x(...) is
  -- Header comment
  require
  local
  x: REAL
  m: METRO STATION
  do
    -- Can use x and m here ...
    ensure
  end
  --
```
- Local variables include `Result` for a function

### Reading assignment

Chapter 6 (control structures), up to 6.6.

### Exercise (uses loops)

Reverse a list!

```
reverse(v: LINKED_LIST) is
  local
    count
    first_element
    last_element
  do
    if is_empty then
      last_element := first_element
    else
      last_element.put_right(first_element)
      first_element := last_element
    end
  end
end
```
End of lecture 7