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

Touch of Class application

- Now available from course page

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 \textit{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

\begin{figure}
\centering
\includegraphics[width=\textwidth]{object_structure}
\caption{Object Structure Diagram}
\end{figure}
Two kinds of type

- **Reference** types: value of an entity is a reference.
  
  Example:
  
  $a$: **POINT**

  ![Diagram showing a reference to POINT]

- **Expanded** types: value of an entity is an object.
  
  Example:
  
  $\alpha$: **expanded POINT**

  ![Diagram showing an expanded POINT]

Expanded classes

- A class may also be declared as
  
  `expanded class C`
  
  ... The rest as usual ...

- Then you can declare:
  
  $a$: **C**

  with the same effect as
  
  $\alpha$: **expanded 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` ...

$n$: **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

---

### 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
feature - Element change
set(xval, yval: REAL) is
   require -- Set coordinates to ("xval", "yval")
   do
      x := xval
      y := yval
   ensure
      x := xval
      y := yval
end end
```

Setting fields (in routines of the class)

```
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
adj_positions is
   do
      upper_left := (x - size/2, y - size/2)
   end
```

What you may do
Feature calls

In class POSITION, we assume the feature `set(x:real, y:real)`. From another class, e.g., `METRO STATION`,

```
adjust_position
  do
    upper_left POSITION
  end
```

From class POSITION itself:

```
movex, dy:real =>
  -- Move by dx horizontally, dy vertically
  require
    ...
  do
    (x + dx, y + dy)
  ensure
    ...
```

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(u, v)` 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

```
create obj1.r1
create obj2.r2
create obj3.r3
```

Root procedure
**The system execution process**

![Diagram of system execution process]

**Current object**

- 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`:

```plaintext
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

```plaintext
target := source
```

*source* is an expression:

- Call to a query:
  - position
  - upper_left(position)
- Arithmetic or boolean expression:
  - \( a + (b \times c) \)
  - \( (a \times b) \) and \( (c = d) \)

*target* may be:

- An attribute
- Result of a function
- A "local variable" of a routine (not yet seen)

---

### Assignment

- Replaces a value by another

```
p
```

```
\[
x \quad 0
\]
```

```
y \quad 0
```

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

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

```plaintext
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
      y_positive: yval > 0
    do
      x := xval
      y := yval
    ensure
      x_positive: x > 0
      y_positive: y > 0
end 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

```plaintext
class TWO_VALUES feature
  item: INTEGER
  right: TWO_VALUES
  set(n: INTEGER; r: TWO_VALUES) is
    item := n
    right := r
end
```

---

Assignment

```plaintext
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.**

1. **item right**
2. **item right**
3. **item right**

**extend(v: STRING) is**
- Add `v` to end.
- Do not move cursor.
- Do not move cursor.
- 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`
- `count := count + 1`

**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**
  - **do**
    - `right := other`
  - **ensure**
    - `chained: right = other`
- **end**
Local variables (in routines)

- A form of entity (they are also called "local entities")
- Just declare them on entry to a routine:

```rust
routine r(...) is
  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:

```
LTR count
LTR last_element
LTR first_element

Halden-BAHNHOF - Central - Hauptbahnhof - Paradeplatz
```

LINKABLE
End of lecture 7