A GLIMPSE OF EIFFEL (2): MASTERING ABSTRACTION

Bertrand Meyer

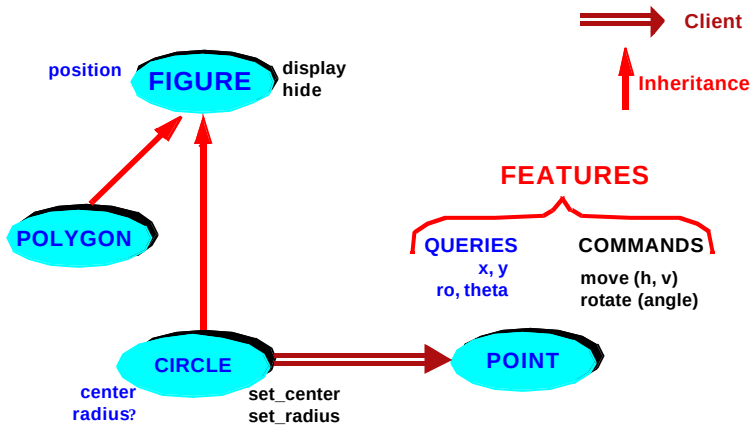
OBJECT-ORIENTED DESIGN

Object technology is about ...?

OBJECT-ORIENTED DESIGN

Object technology is about abstraction.

O-O STRUCTURE



THE KEY CONCEPT: CLASS

A class is the description (in the software text) of a set of potential runtime objects, accessible to the rest of the software exclusively through a set of specified operations, or "features".

ABSTRACTION

From the outside, a POINT object is accessible ONLY through its features:

What is your abscissa?	(x)
What is your ordinate?	(y)
What is your distance to the center?	(ro)
What is your angle to the horizontal?	(theta)
Move yourself by a certain displacement!	(move)
Rotate around the origin by a certain angle!	(rotate)

THE POINT CLASS

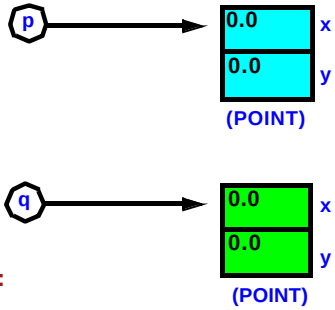
```
class POINT feature

    x, y: REAL

    move (a, b: REAL) is
        -- Move by a horizontally, b vertically.
      do
        x := x + a;   y := y + b
      end

    scale (factor: REAL) is
        -- Change the distance to the origin by factor.
      do
        x := factor * x;   y := factor * y
      end
```


CLASS POINT (CONTINUED)

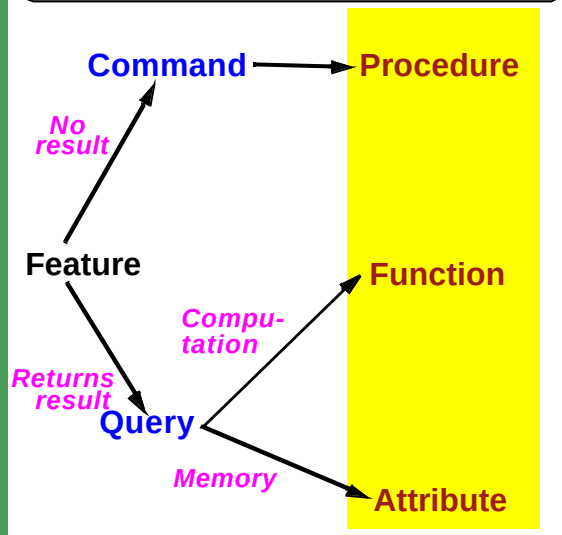
```
distance (p: POINT): REAL is
            -- Distance to p
      do
            Result := sqrt ((x – p.x) ^ 2 + (y – p.y) ^ 2)
      end
ro: REAL is
            -- Distance to origin (0, 0)
      do
            Result := sqrt (x ^ 2 + y ^ 2)
      end
theta: REAL is
            -- Angle to horizontal axis
    do
      ...
    end
end -- class POINT
```

USE OF THE CLASS (IN A CLIENT)

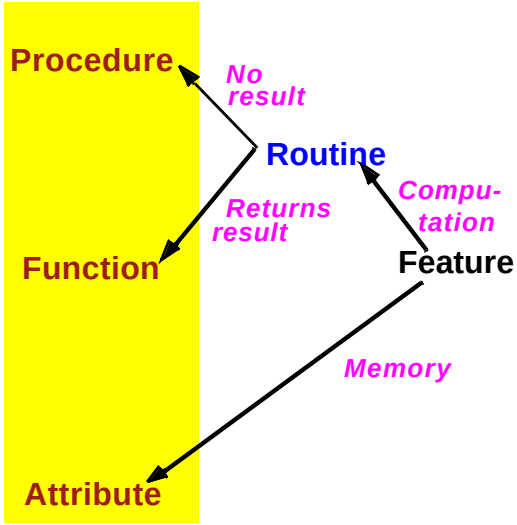
```
class GRAPHICS feature
  p, q: POINT
  ...
  some_routine is
  local
     u, v: REAL
  do
        -- Creation instructions:
     create p ;  create q
     p.move (4.0, –2.0)
        -- Compare with Pascal, C, Ada:
        -- move (p, 4.0, –2.0)
     p.scale (0.5)
     u := p.distance (q)
     v := p.x
     p := q
     p.scale (–3.0)
  end
end -- class GRAPHICS
```



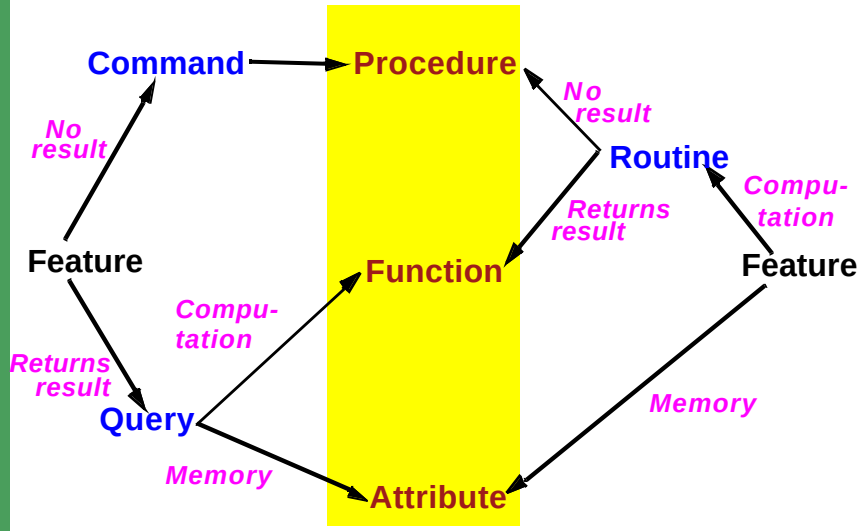
FEATURE CATEGORIES BY ROLE



FEATURE CATEGORIES BY IMPLEMENTATION



FEATURE CATEGORIES



NO DIRECT FIELD MANIPULATION!

~~your_point.x := 23~~

THE PRIVILEGES OF A CLIENT ON A QUERY

APPLYING ABSTRACTION PRINCIPLES

Beyond read access: full or restricted write, through exported procedures.

Full write privileges: set_attribute procedure, e.g.

```
set_x (new_abscissa: REAL) is
        -- Set horizontal coordinate to new_abscissa.
    do
        c := new_abscissa
    ensure
        has_been_set: x = new_abscissa
    end
```

Clients will use e.g. my_point.set_x (21.5).

INFORMATION HIDING

```
class A feature

    f ...

    g ...

feature {NONE}

    h ...

feature {B, C}

    j ...

feature {A, B, C}

    k

end -- class A
```

In clients, with the declaration a1: A, we have:

a1.f, a1.g: valid in any client

a1.h: invalid anywhere (including in A's own text).

a1.j: valid only in B, C and their descendants (not valid in A!)

a1.k: valid in B, C and their descendants, as well as in A and its descendants

INFORMATION HIDING

Information hiding only applies to use by clients, using dot notation or infix notation, as with a1.f ("Qualified calls").

Unqualified calls (within the class itself) are not subject to information hiding:

```
class
    A
feature {NONE}
    h is do ... end
feature
    f is
        do
            ...
            h
        end
end -- class A
```


FORMS OF ASSIGNMENT AND COPY

Reference assignment (a and b of reference types):

b := a

Object duplication (shallow):

c := clone (a)

Object duplication (deep):

d := deep_clone (a)

Also: shallow field-by-field copy (no new object is created):

e.copy (a)

SHALLOW AND DEEP CLONING

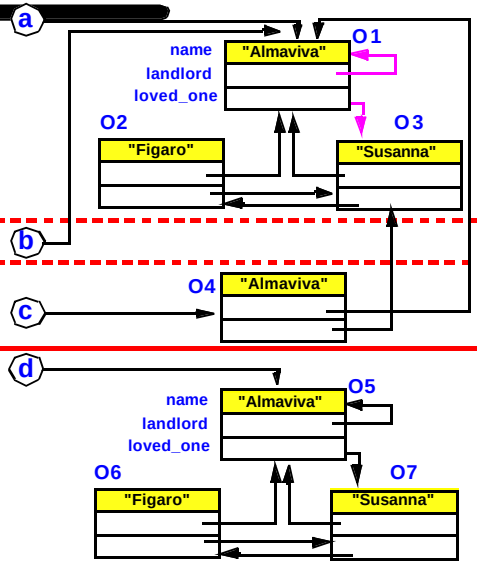
Initial situation:

Result of:

b := a

c := clone (a)

d := deep_clone (a)



A RELATED MECHANISM: PERSISTENCE

a.basic_store (file)

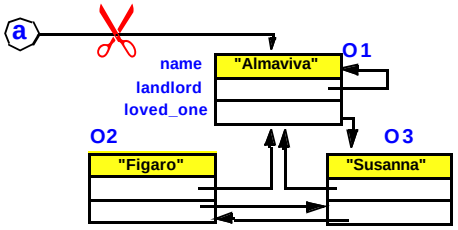
....

b ?= retrieved (file)

- Storage is automatic.
- Persistent objects identified individually by keys.

These features come from the library class STORABLE..

WHAT TO DO WITH UNREACHABLE OBJECTS



Reference assignments may make some objects useless.

a := b
a := *Void*

Two possible approaches:

- Manual reclamation.
- Automatic garbage collection as in Eiffel.

ARGUMENTS FOR AUTOMATIC GC

- Manual reclamation is dangerous. Hampers software reliability.
- In practice bugs arising from manual reclamation are among the most difficult to detect and correct. Manifestation of bug may be far from source.
- Manual reclamation is tedious: need to write “recursive dispose” procedures.
- Modern garbage collectors have acceptable overhead (a few percent) and can be made compatible with real-time requirement.
- GC is tunable: disabling, activation, parameterization....

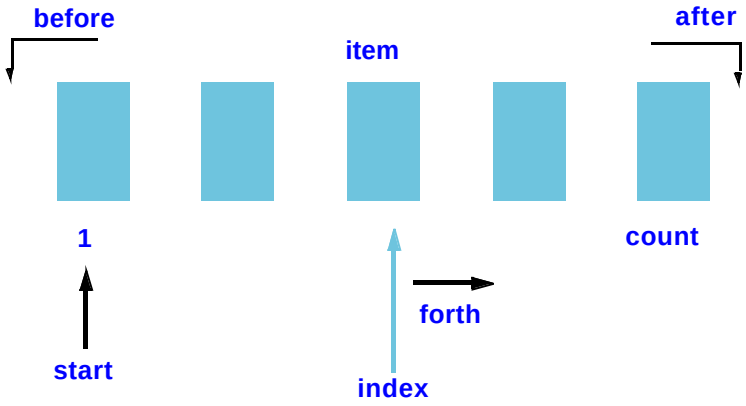
LISTS (FIRST ATTEMPT)

Queries: item (i), count, is_empty...

Commands: insert (x, i), remove (i)

n := search (x)

LISTS (THE PROPER API)



SEARCHING PATTERN

```
from
    your_list.start
until
    your_list.after or else item = x
loop
    your_list.forth
end

found := not your_list.after
```


PSEUDO-RANDOM NUMBER GENERATOR

Non-O-O:

```
x := random_start (some_seed)
...
x := random_next (x)
...
```


WHAT IS THE ABSTRACTION?

Random number?

AN ABSTRACTION IS CHARACTERIZED BY FEATURES

Abstraction: random number sequence

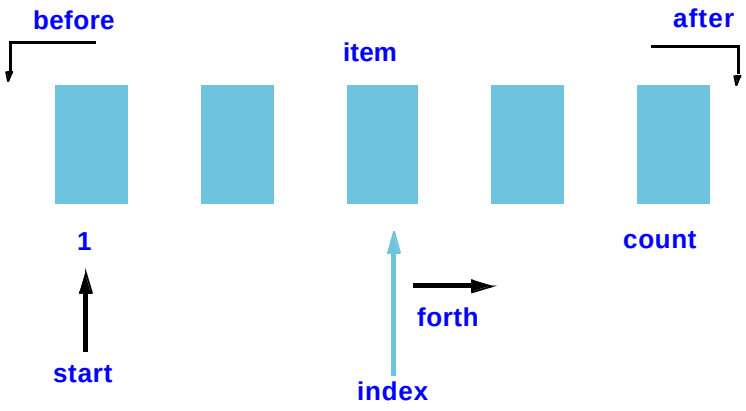
Commands:

- Set seed
- Go to next

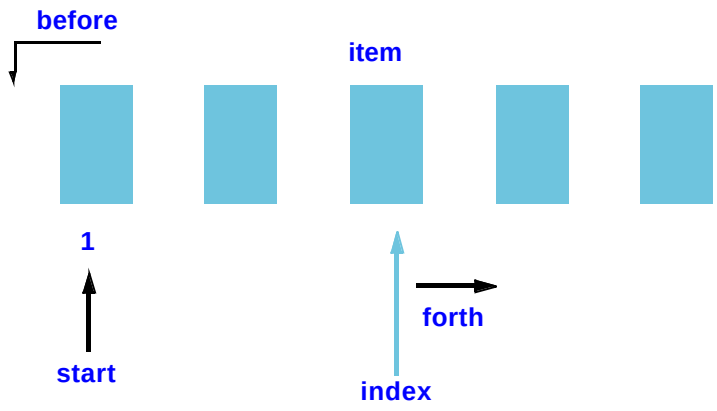
Queries:

- Current random number

LISTS (THE PROPER API)



INFINITE LISTS



EXAMPLES

Fibonacci sequence

Prime numbers

Pseudo-random numbers

THE MOTTO

Search for the right abstractions

MORE TO COME...