STATIC TYPING
AND OTHER MYSTERIES OF LIFE

or:

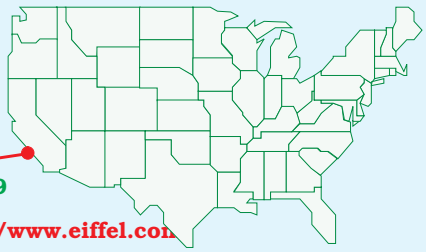
BEWARE OF POLYMORPHIC CATCALLS!

Bertrand Meyer

ISE Inc.

Author's address:

Interactive Software Engineering Inc.
270 Storke Road, Suite 7
Santa Barbara, CA 93117 USA
Telephone 805-685-1006, Fax 805-685-6869
E-mail <info@eiffel.com>
Anonymous FTP: eiffel.com — WWW: http://www.eiffel.com



TYPING: A SIMPLE PROBLEM!

The basic operation of object-oriented computation:

$$x.f\ (arg)$$

THE TYPING PROBLEM

When, and how, do we know that:

- **There is a feature applicable to *OBJ* and corresponding to *f*?**
- ***arg* is an acceptable argument for that feature?**

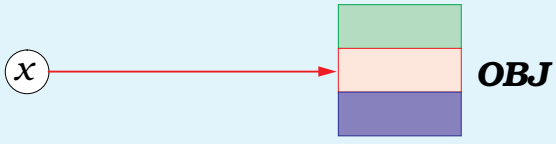
TERMINOLOGY

***f*: Routine**

(One of the two forms of feature; the other is *attributes*).

***x*: Entity.**

***OBJ*: the object attached to *x* at some time during execution.**



THE AMERICA CONJECTURE

(Pierre America at TOOLS EUROPE 1990)

One can have at most two of the three properties of:

- **Static typing.**
- **Substitutivity.**
- **Covariance.**

The present work is intended to disprove the America conjecture.

TYPING vs. BINDING

Static typing:

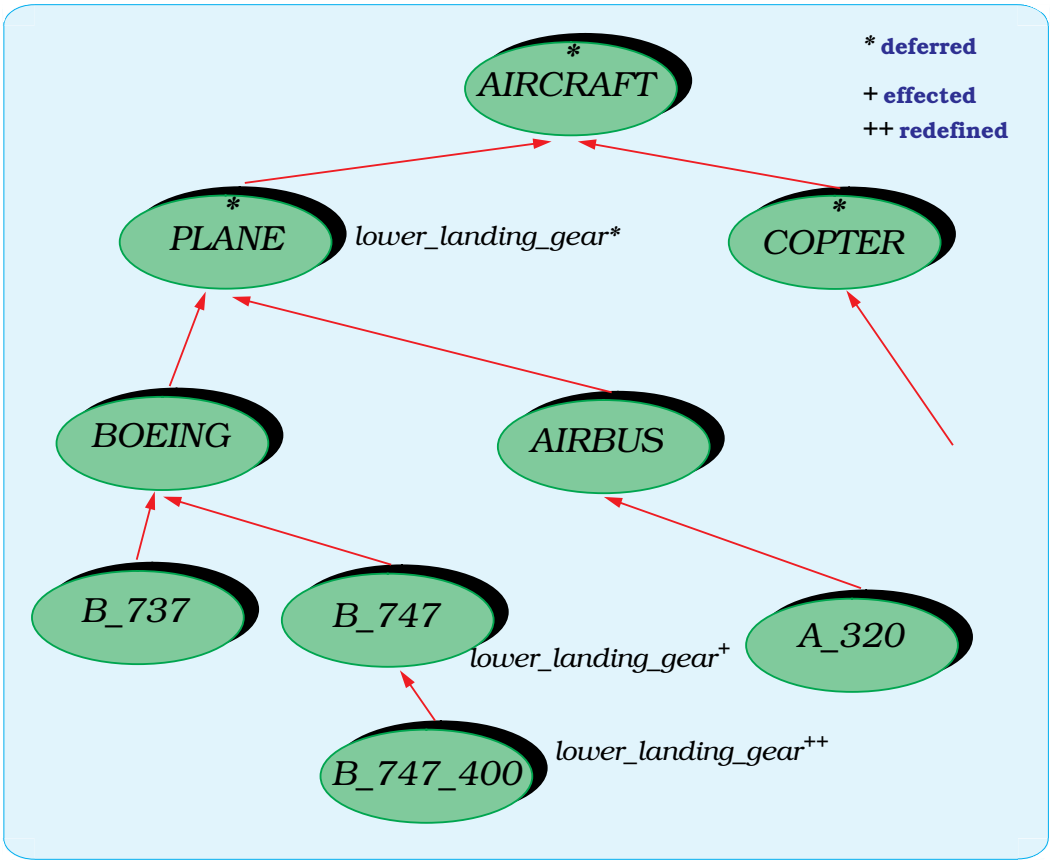
At least one feature.

Dynamic binding:

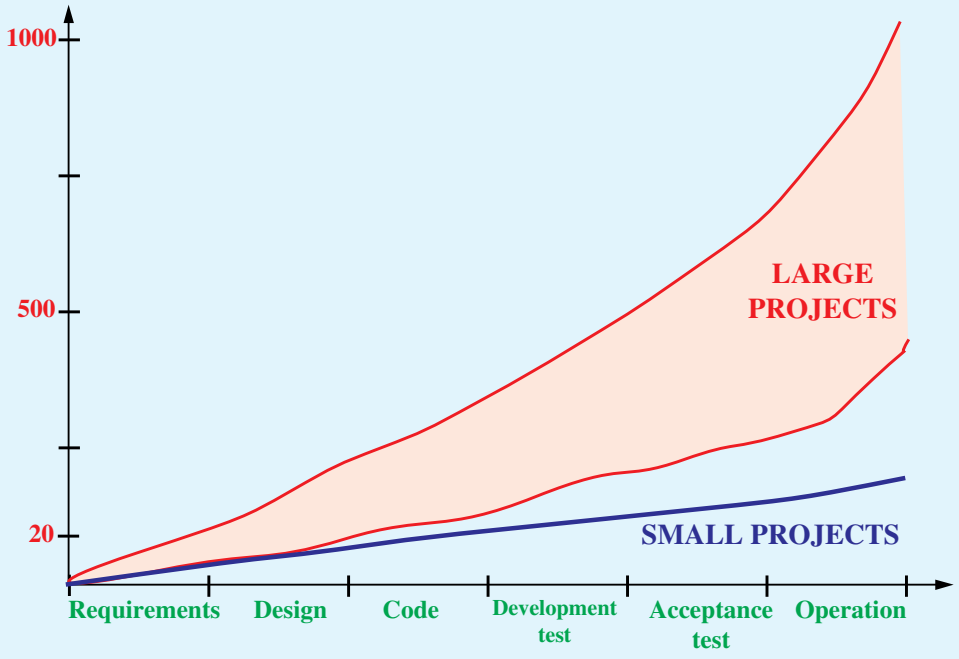
The right feature.

Example:

my_aircraft.*lower_landing_gear*



THE RELATIVE COST OF CORRECTING ERRORS (After Barry W. Boehm, "Software Engineering Economics")



TYPING RULES (ALL LOCALLY CHECKABLE)

Declaration rule.

Call rule.

Attachment rule.

(For the precise formulations see *Eiffel: The Language*, Prentice Hall.)

DECLARATION RULE

Every entity must be declared as being of a certain type.

For example:

x: AIRCRAFT;

n: INTEGER;

ba1: BANK_ACCOUNT

CALL RULE

If a class C contains the call

$x.f(\ldots)$

there must be a feature of name f in the base class of the type of x, and that feature must be available (exported) to C.

ATTACHMENT RULE

In an assignment x := y, or the corresponding argument passing, the base class of the type of y must be a descendant of that of x.

FOR TYPING TO BE ACCEPTABLE

No exception to type rules ("casts").

Multiple inheritance

Unconstrained genericity
-- e.g. *class* *LIST* [*G*] ...

Constrained genericity
-- e.g. *class* *VECTOR* [*G* –> *NUMERIC*] ...

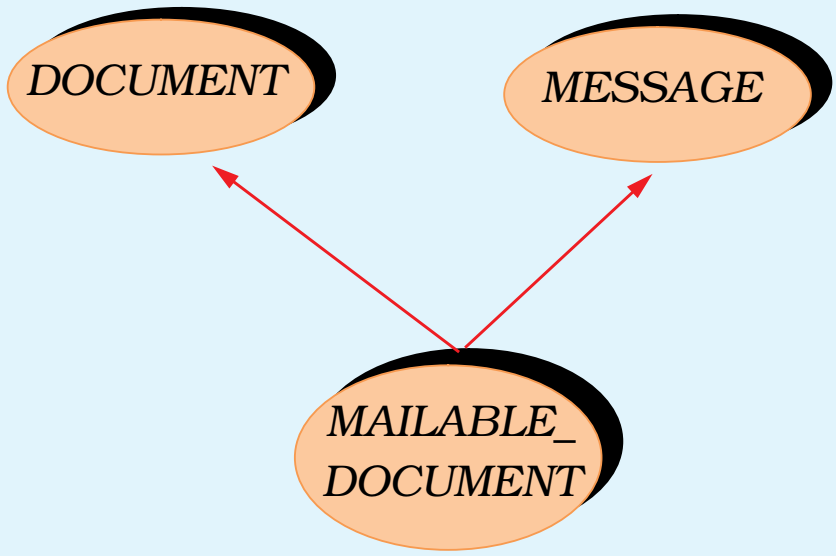
Assignment attempt
-- *x* ?= *y*

Assertions, to clarify the semantics and include constraints other than typing (e.g. numerical ranges).

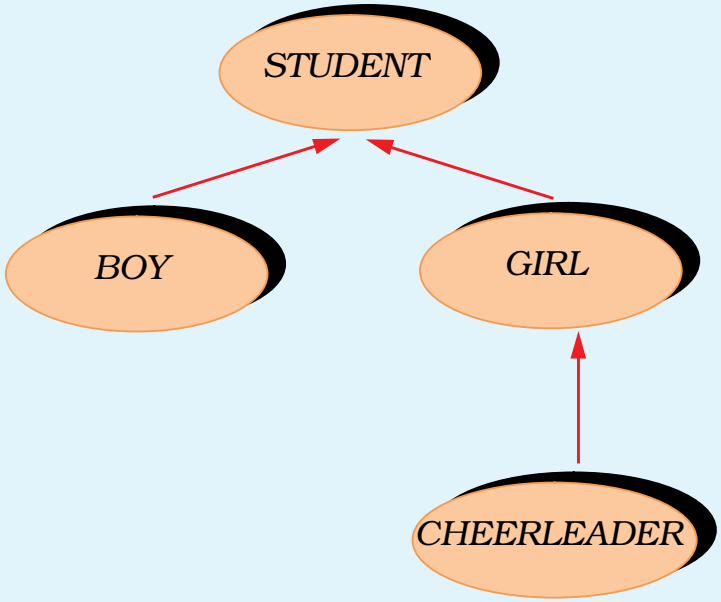
Covariance for routine arguments.

Anchored declarations (*like* *x*) to avoid endless redeclarations.

MULTIPLE INHERITANCE



A CLASS HIERARCHY



CHOOSING A ROOMMATE

```
class STUDENT feature

    roommate: STUDENT;
            -- This student's roommate.


    share (other: STUDENT) is
            -- Choose other as roommate.
        require
            other /= Void
        do
            roommate := other
        end

    ...

end -- class STUDENT
```

A TYPICAL CALL

```
s1, s2: STUDENT;

...

s1.share (s2)
```


SPECIALIZING

```
class GIRL inherit

    STUDENT
        redefine roommate end

feature

    roommate: GIRL;
            -- This student's roommate.

    ...

end -- class STUDENT
```

THE NEED FOR COVARIANCE

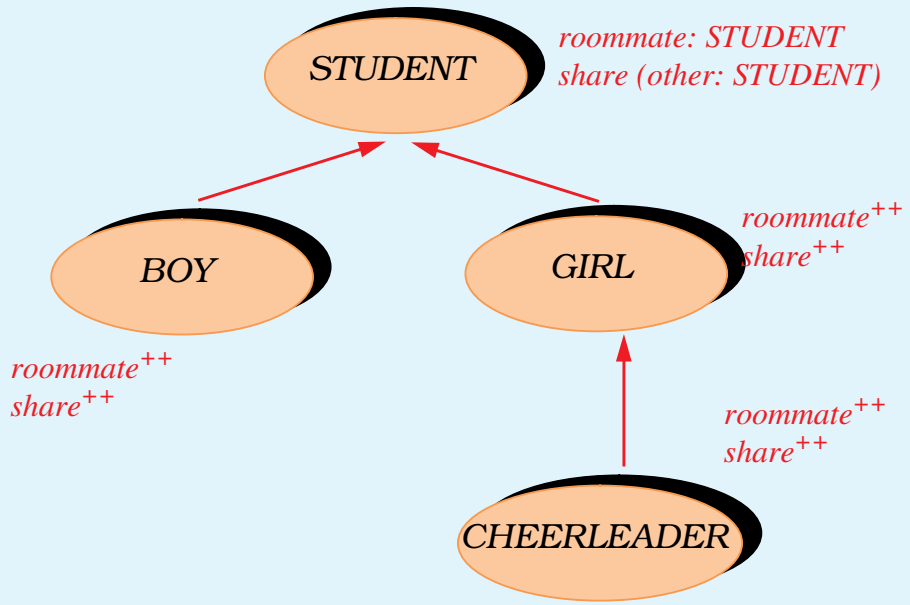
```
class GIRL inherit
    STUDENT
        redefine roommate, share end

feature
    roommate: GIRL;
            -- This student's roommate.

    share (other: GIRL) is
            -- Choose other as roommate.
        require
            other /= Void
        do
            roommate := other
        end

    ...
end -- class STUDENT
```


COVARIANCE



Avoiding redefinition avalanche:

ANCHORED DECLARATIONS

```
class STUDENT feature

    roommate: STUDENT;
            -- This student's roommate.

    share (other: like roommate) is
            -- Choose other as roommate.
        require
            other /= Void
        do
            roommate := other
        end

    ...
end -- class STUDENT
```


POLYMORPHISM AND THE ATTACHMENT RULE

In an assignment *x* := *y*, or the corresponding argument passing, the base class of the type of *y* must be a descendant of that of *x*.

POLYMORPHISM AND COVARIANCE:

s1: *STUDENT*; *b1*: *BOY*; *g1*: *GIRL*;

...

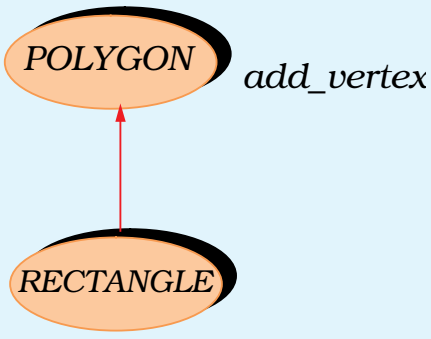
-- Create a “boy” and a “girl” objects:

!! *b1* ; !! *g1*;

s1 := *b1*;

s1.*share* (*g1*)

DESCENDANT HIDING




```
class RECTANGLE inherit
        POLYGON
            export
                {NONE} add_vertex
            end
feature
        ...
invariant
        vertex_count = 4
end
```

POLYMORPHISM AND DESCENDANT HIDING

```
p: POLYGON; r: RECTANGLE;
...
!! r
p := r ;

p.add_vertex (...)
```


SOLUTIONS TO THE COVARIANT TYPING PROBLEM

Solution described in *Eiffel: The Language*: system validity.

Compute set of all potential types for each entity. This will for example disallow combined presence of

```
s1 := b1
...
s1.share (g1)
```

because one of the possible types for *s1* is *BOY*.

Disadvantage: requires (at least in a straightforward implementation) access to the entire system. Errors not easy to explain to users.

STATIC TYPING IS PESSIMISTIC

(Pascal conventions):

n: INTEGER;

n := 0.0;

n := 1.0;

n := 3.67

n := 3.67 – 3.67

THE NEW SOLUTION

For the full rules:

http://www.eiffel.com

(Follow link to "Recent typing work".) Starting 20 October 1995.

Full discussion with details:

- ***Object-Oriented Software Construction*, second edition.**
- **Forthcoming Eiffel workshops at TOOLS PACIFIC (Melbourne, late November) and TOOLS EUROPE (Paris, 26-29 February).**

POLYMORPHIC ENTITY

An entity x is polymorphic if it satisfies one of the following properties:

- **It appears in an assignment x := y where y is of a different type or (recursively) polymorphic.**
- **It is a formal routine argument.**
- **It is an external function.**

CATCALL

A routine is a CAT (Changing Availability or Type) if some redefinition changes its export status or the type of one of its arguments.

A call is a catcall if some redefinition of the routine would make it invalid because of a change of export status or argument type.

POLYMORPHIC CATCALLS

```
s1: STUDENT; b1: BOY; g1: GIRL; ...

!! b1 ; !! g1

s1 := b1;

s1.share (g1)
```

```
p: POLYGON; r: RECTANGLE; ...

!! r
p := r
p.add_vertex (...)
```


THE NEW TYPE RULE

Polymorphic catcalls are invalid.

Note: can be checked incrementally; will be flagged either when adding a call or when adding a redefinition.

Compatible with the idea of precompiled libraries distributed without source; interface documentation (Eiffel's flat-short form) will use a convention to mark polymorphic entities.

A NEW FUNCTION IN CLASS GENERAL (from the Eiffel Library Kernel Standard)

```
fitted (other: GENERAL): like other is
        -- Current if other is attached to an object of
        -- exactly the same type; void otherwise.
    do
        if other /= Void and then same_type (other) then
            Result ?= Current
        end
    end
```


MATCHING SAFELY

```
match (s1, s2: STUDENT) is
        -- Assign s1 to same room as s2 if permissible.
    local
        gender_ascertained_s2: like s1
    do
        gender_ascertained_s2 := s2.fitted (s1);
        if gender_ascertained_s2 /= Void then
            s1.share (gender_ascertained_s2)
        else
            "Report matching is impossible for s1 and s2"
        end
    end
```