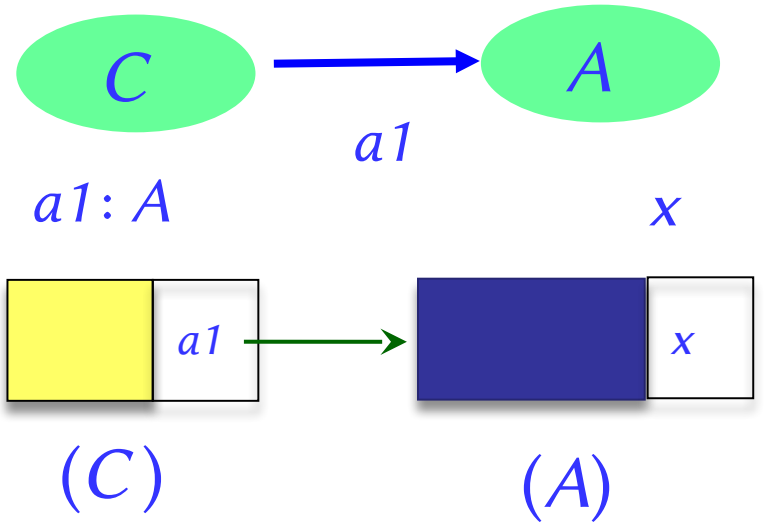
Fondamenti della Programmazione: Metodi Evoluti

Prof. Enrico Nardelli

Lezione 6: Visibilità

Abstraction and client privileges

If class *A* has an attribute *x*,
what may a client class *C*
do with *a1.x* for *a1* of type *A* ?



Read access

- *a1.x* is an expression!
- An assignment ~~*a1.x := v*~~ would be syntactically illegal!

(It would assign to an expression, like ~~*a + b := v*~~)

Applying abstraction principles

To provide clients with writing privileges:
define a **setter procedure**, such as

```
set_temperature (u : REAL)
                -- Set temperature value to u.
        do
                temperature := u
        ensure
                temperature_set: temperature = u
        end
```

Clients will use calls such as

```
a1.set_temperature (21.5)
```

Taking full advantage of a setter command

set_temperature (*u* : *REAL*)

-- Set temperature value to *u*.

require

not_under_minimum: *u* >= -273

not_above_maximum: *u* <= 2000

do

temperature := *u*

update_database

Allows adding preconditions ensuring clients will see them

Allows modification without affecting clients

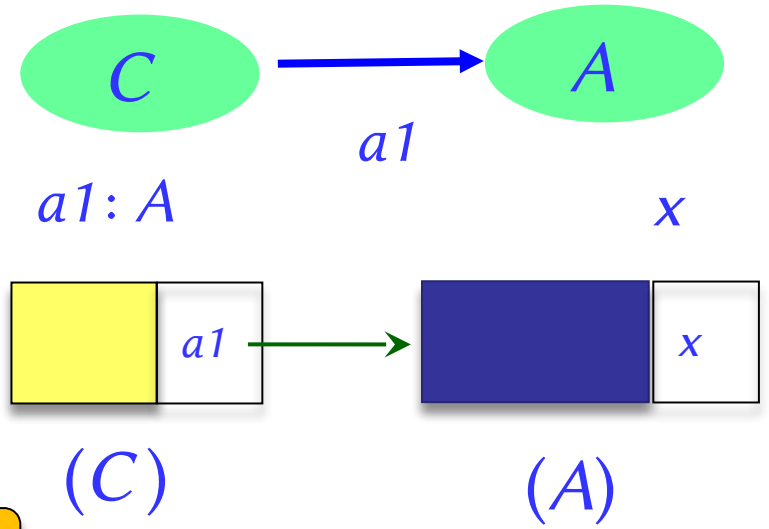
ensure

temperature_set: *temperature* = *u*

end

Abstraction and client privileges

If class *A* has an attribute *x*,
what may a client class *C*
do with *a1.x* for *a1* of type *A* ?



Read access **if attribute is exported**

- *a1.x* is an expression!
- An assignment ~~*a1.x := v*~~ would be syntactically illegal!

(It would assign to an expression, like ~~*a + b := v*~~)

Exporting (making public) an attribute

In Eiffel, exporting an attribute means exporting it read-only. By default, all attributes are exported.

From the outside, it is not shown as an attribute, just as a **query**: it could be a function

In C++, Java & C#, if you make public an attribute* *x*, it is available for both read and write:

- *v* := *a1.x*
- *a1.x* := *v*

* (field, member variable)

As a result, it is almost always a bad idea in these languages to export an attribute: better to have it private and provide getter functions

Getter functions

In C++, Java & C#, the standard technique, if attribute *x* is private, is to export an associated **getter function**:

```
get_x: T
        do
                Result := x
        end
```

Eiffel needs no getter functions: just export the attribute

This is safe since the attribute is exported:

- Only for reading
- Without the information that it is an attribute: it could be a function (Uniform Access principle)

Having it both ways (Eiffel syntax)

It is possible to define a query as

temperature: *REAL* **assign** *set_temperature*

Then the syntax

x.temperature := 21.5

Not an assignment, but a procedure call

is accepted as an abbreviation for

x.set_temperature (21.5)

Retains **contracts** and any other supplementary operations

C# has a notion of "property" which pursues the same goal

Information hiding: how to export features

```
class
    A

feature
        f ...
        g ...

feature {NONE}
        h, i ...

feature {B, C}
        j, k, l ...

feature {A, B, C}
        m, n ...
end
```

Status of calls in a client with *a1* : A

- *a1.f*, *a1.g* : valid in any client
- *a1.h* : invalid everywhere

  (including in *A*'s own text! but see later...)
- *a1.j* : valid only in *B*, *C* and their descendants

  (not valid in *A* !)
- *a1.m* : 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).

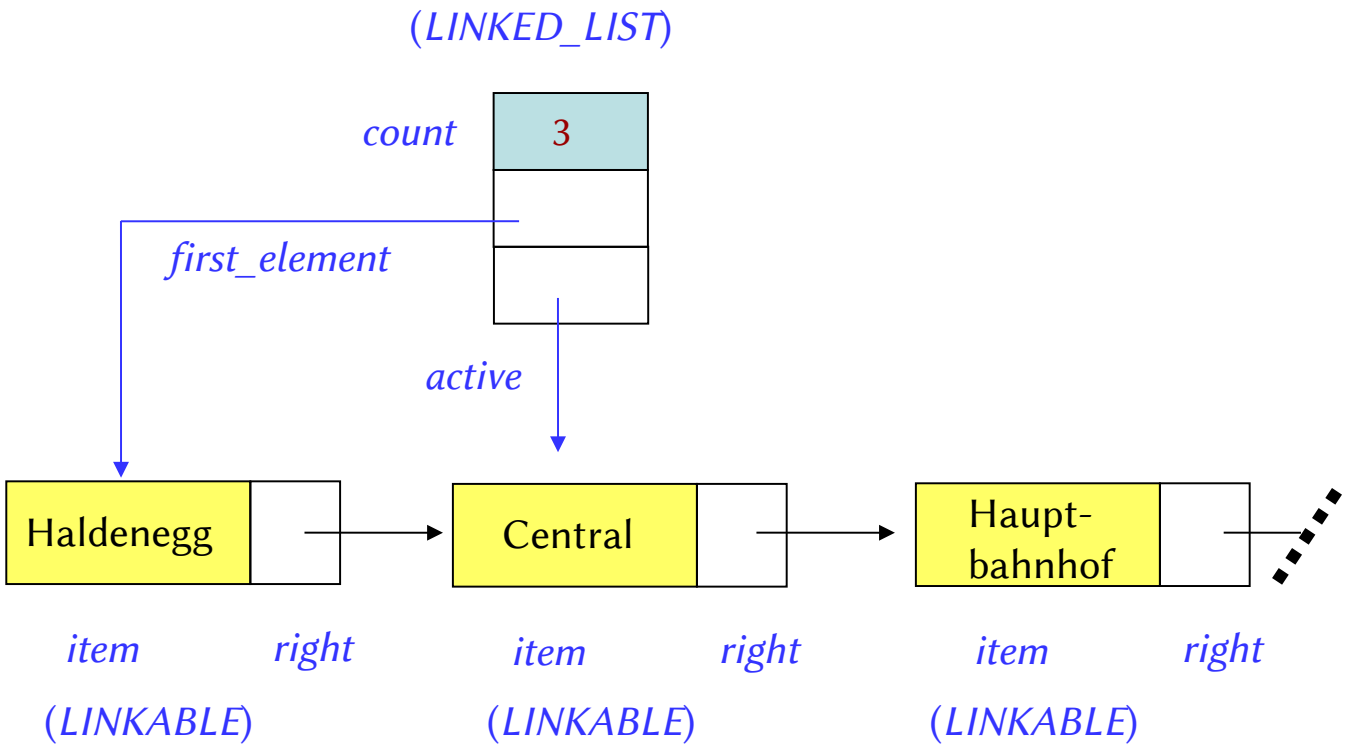
Unqualified calls (only possible within class and its descendants) are **not** subject to information hiding:

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

An example of selective export

LINKABLE exports its features to *LINKED_LIST*

- No need to export them to the rest of the world
- Clients of *LINKED_LIST* don't need to know about *LINKABLE* cells, they will use *LINKED_LIST* features.



Exporting selectively

class

LINKABLE

These features are selectively exported to *LINKED_LIST* and its descendants (and no other classes)

feature {*LINKED_LIST*}

put_right (...) **do** ... **end**

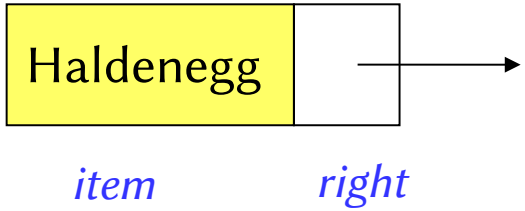
right: *G* **do** ... **end**

...

end

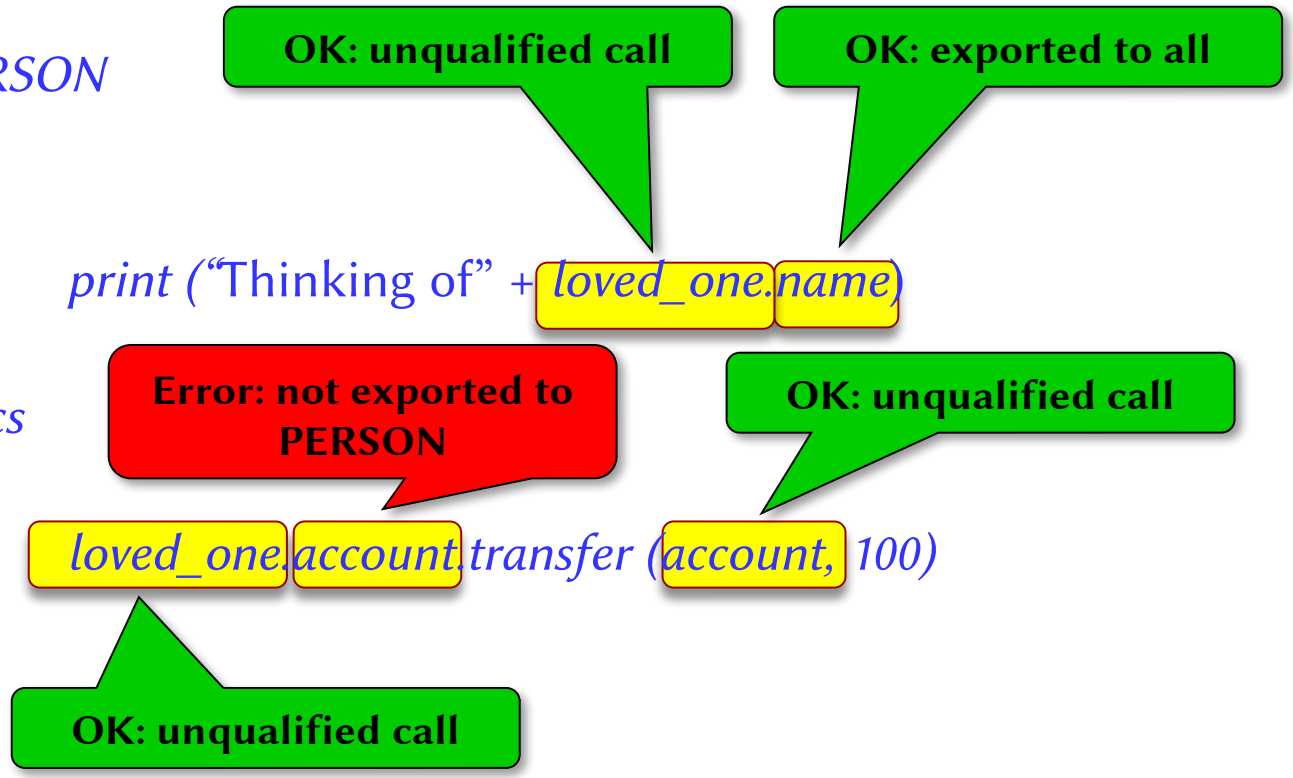
LINKABLE

```
class LINKABLE
feature {LINKED_LIST}
        item: STRING
                        -- Value in this cell

        right: LINKABLE
                        -- Cell, if any, to which this one is chained

        put_right (other: like Current)
                        -- Put other to the right of current cell.
                do
                        right := other
                ensure
                        chained : right = other
                end
end
```



Exporting feature: compilation error?

```
class PERSON
feature
        name: STRING
feature {BANK}
        account: BANK_ACCOUNT
feature {NONE}
        loved_one: PERSON
        think
                do
                        print ("Thinking of" + loved_one.name)
                end
        lend_100_francs
                do
                        loved_one.account.transfer (account, 100)
                end
end
```



The export status does matter! (1)

```
class PROFESSOR


create
        make
feature
        make (a_exam_draft: STRING)
                do
                        exam_draft := a_exam_draft
                end
feature
        exam_draft: STRING
end
```

The export status does matter! (2)

```
class ASSISTANT

create
        make
feature
        make (a_prof: PROFESSOR)
                do
                        prof := a_prof
                end
feature
         prof: PROFESSOR
feature
        review_draft
                do
                        -- review  prof.exam_draft
                end
end
```

The export status does matter! (3)

```
class STUDENT

create
        make
feature
        make (an_assistant: ASSISTANT)
                do
                        assistant := an_assistant
                end
feature
        assistant: ASSISTANT
feature
        stolen_exam: STRING
                do
                        Result := assistant.prof.exam_draft
                end
end
```

The export status does matter! (4)

you: STUDENT
your_prof: PROFESSOR
your_assistant: ASSISTANT
stolen_exam: STRING

create *your_prof.make (*“top secret exam!”*)*
create *your_assistant.make (your_prof)*
create *you.make (your_assistant)*

stolen_exam := you.stolen_exam

Rev. 2.5.1 (2022-23) di Enrico Nardelli (basato su touch.ethz.ch)
Rev. 2.7.1 (2024-25) di Enrico Nardelli (basato su touch.ethz.ch)

The export status does matter! (5)

```
class PROFESSOR
create
        make
feature
        make (a_exam_draft: STRING)
                do
                        exam_draft := a_exam_draft
                end
feature   {PROFESSOR, ASSISTANT}
        exam_draft: STRING
end
```

The export status does matter! (6)

```
class STUDENT
create
        make
feature
        make (an_assistant: ASSISTANT)
                do
                        assistant := an_assistant
                end
feature
        assistant: ASSISTANT
feature
        stolen_exam: STRING
                do
                        Result := assistant.prof.exam_draft
                end
end
```

Invalid call!