EECS3311 Software Design (Fall 2020)

Q&A - Lecture Series W5

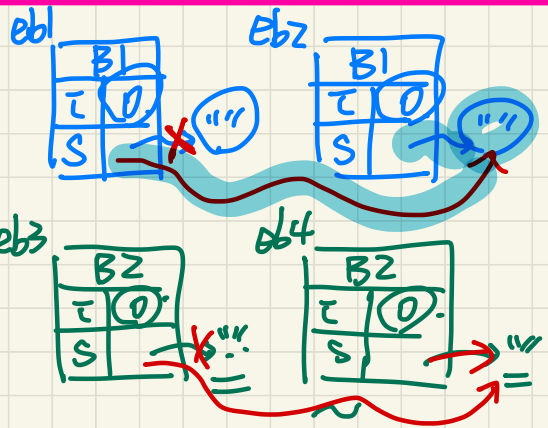
Monday, October 19

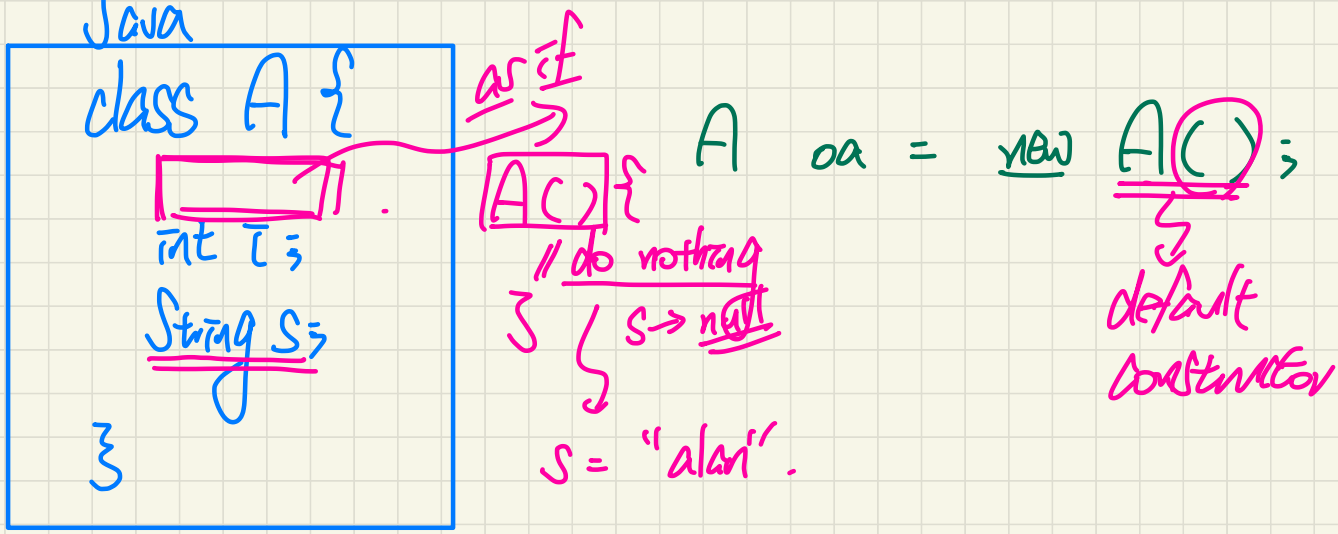
Expanded Class vs Deep Copying (1)

```
expanded class
    B1
inherit
    ANY
        redefine
            default_create
        end
create
    default_create
feature -- command
    default_create
        do
            create s.make_empty
        end
feature
    i: INTEGER
    s: STRING
feature
    change_i (ni: INTEGER)
        do
            i := ni
        end
    change_s (ns: STRING)
        do
            s := ns
        end
end
```

```
expanded class
    B2
inherit
    ANY
        redefine
            is_equal,
            default_create
        end
create
    default_create
feature -- command
    default_create
        do
            create s.make_empty
        end
feature
    i: INTEGER
    s: STRING
feature
    change_i (ni: INTEGER)
        do
            i := ni
        end
    change_s (ns: STRING)
        do
            s := ns
        end
feature
    is_equal (other: like Current): BOOLEAN
        do
            Result := i = other.i and s ~ other.s
        end
end
```

```
test_expanded: BOOLEAN
    local
        eb1, eb2: B1
        eb3, eb4: B2
    do
        Result := eb1.i = 0 and eb2.i = 0 and eb1.s /= eb2.s and eb1.s ~ eb2.s
        check Result end

        Result := eb1 /= eb2 -- eb1.i = eb2.i and eb1.s = eb2.s
        check Result end

        Result := eb3.i = 0 and eb4.i = 0 and eb1.s /= eb2.s and eb1.s ~ eb2.s
        check Result end

        Result := eb3 = eb4 -- eb1.i = eb2.i and eb1.s ~ eb2.s
        check Result end

        eb1.change_s (eb2.s)
        Result := eb1 = eb2 -- eb1.i = eb2.i and eb1.s = eb2.s
        check Result end

        eb2.change_s (eb4.s)
        Result := eb3 = eb4
    end
```

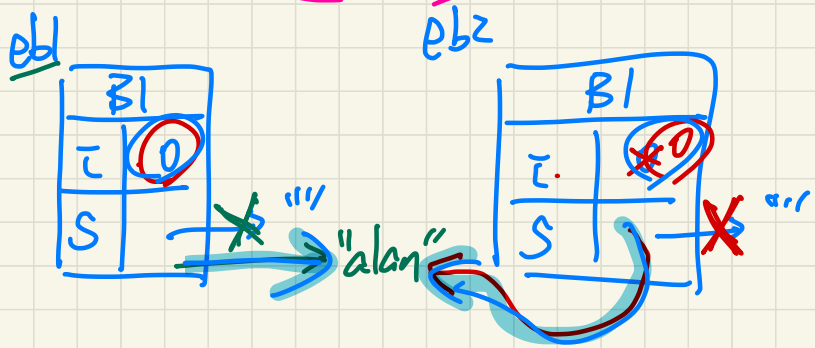



Java
class A {
int i;
String s;
}
as if
A() {
// do nothing
}
s → null
s = "alan".
A oa = new A();
default constructor

Expanded Class vs Deep Copying (2)

```
expanded class
    B1
inherit
    ANY
        redefine
            default_create
        end
create
    default_create
feature -- command
    default_create
        do
            create s.make_empty
        end
feature
    i: INTEGER
    s: STRING
feature
    change_i (ni: INTEGER)
        do
            i := ni
        end
    change_s (ns: STRING)
        do
            s := ns
        end
end
```

```
test_expanded_copying: BOOLEAN
    local
        eb1, eb2: B1
    do
        Result := eb1.i = 0 and eb2.i = 0 and eb1.s /= eb2.s and eb1.s ~ eb2.s
        check Result end

        Result := eb1 /= eb2 -- eb1.i = eb2.i and eb1.s = eb2.s
        check Result end

        eb1.change_s ("alan")
        eb2 := eb1
        -- copying from one expanded object to another can still cause sharing

        Result := eb1.i = 0 and eb2.i = 0 and eb1.s = eb2.s and eb1.s ~ eb2.s
        check Result end

        Result := eb1 = eb2 -- eb1.i = eb2.i and eb1.s = eb2.s
        check Result end
    end
```

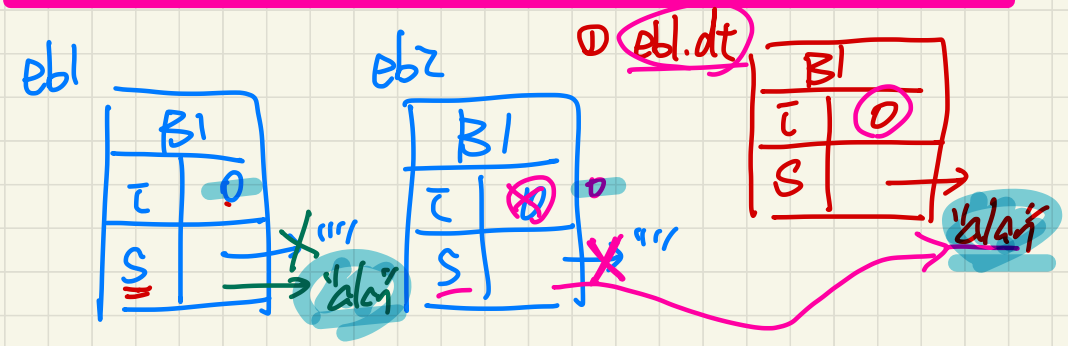


Expanded Class vs Deep Copying (3)

```
expanded class
    B1
inherit
    ANY
        redefine
            default_create
        end
create
    default_create
feature -- command
    default_create
        do
            create s.make_empty
        end
feature
    i: INTEGER
    s: STRING
feature
    change_i (ni: INTEGER)
        do
            i := ni
        end
    change_s (ns: STRING)
        do
            s := ns
        end
end
```

```
test_expanded_deep_twin: BOOLEAN
    local
        eb1, eb2: B1
    do
        Result := eb1.i = 0 and eb2.i = 0 and eb1.s /= eb2.s and eb1.s ~ eb2.s
        check Result end

        Result := eb1 /= eb2 -- eb1.i = eb2.i and eb1.s = eb2.s
        check Result end

        eb1.change_s ("alan")
        eb2 := eb1.deep_twin
        -- copying from one expanded object to another can still cause sharing

        Result := eb1.i = 0 and eb2.i = 0 and eb1.s /= eb2.s and eb1.s ~ eb2.s
        check Result end

        Result := eb1 /= eb2 -- eb1.i = eb2.i and eb1.s = eb2.s
        check Result end
    end
```

① eb2.i := eb1.i
eb2.s := eb1.i.s

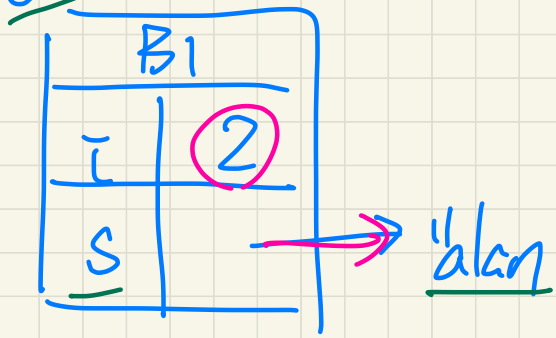


ebl.s.deep_twin = ebl.deep_twin.s

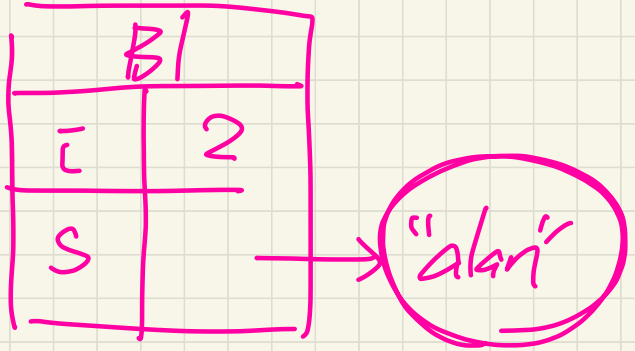
Ⓕ

Ⓘ̃

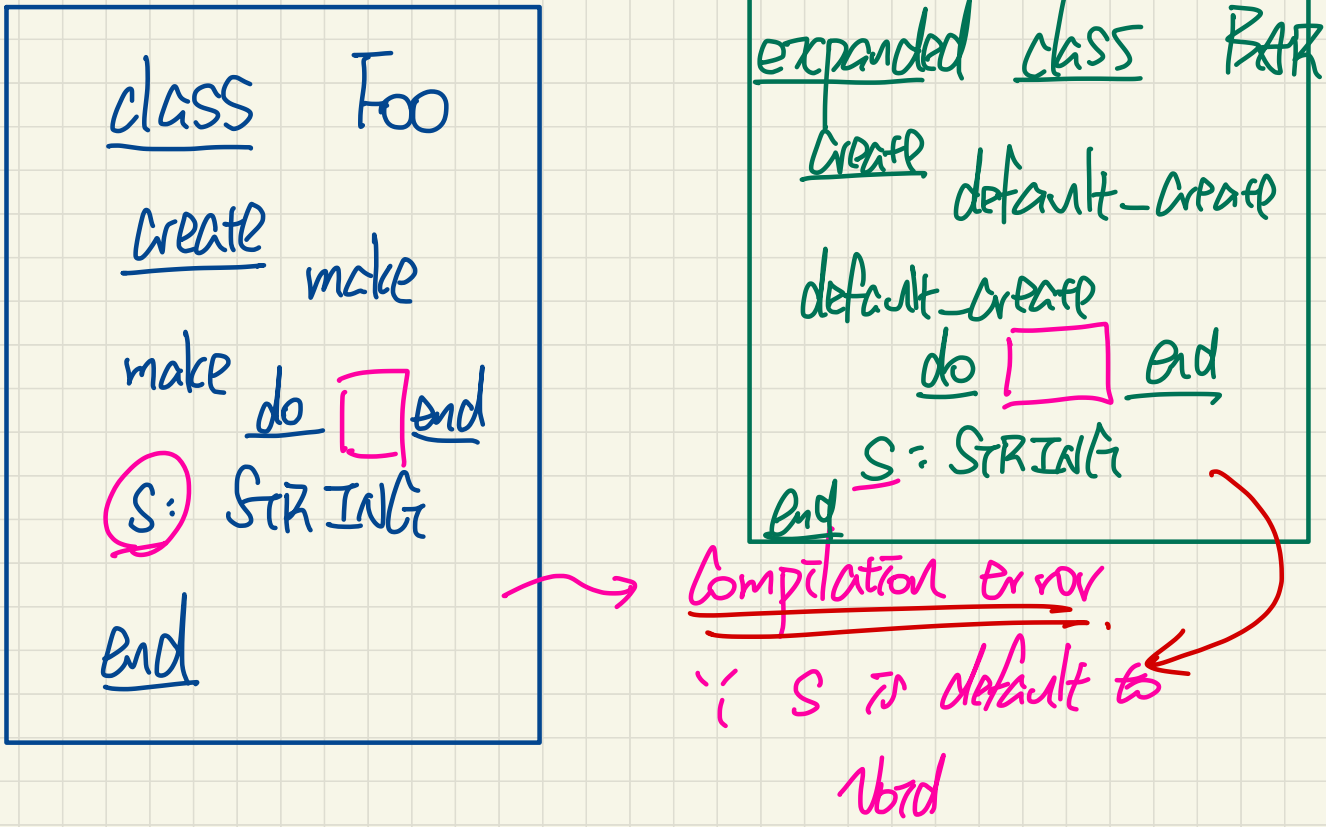
ebl



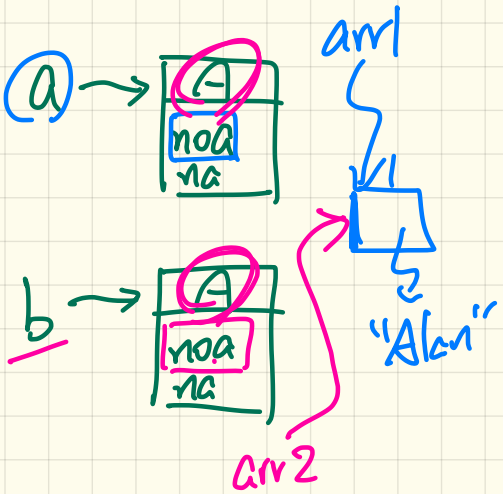
ebl.dt



"alan"


class Foo
create make
make do end
s: STRING
end
expanded class BAR
create default_create
default_create do end
s: STRING
end
Compilation error
`s' is default to
void

Once Routine (2)




```
local
        a,b:A
        arra1,arr2:ARRAY[STRING]
do
        create a.make
        create b.make
        arr1:=a.new_once_array("Alan")
        arr2:=b.new_once_array("Mark")
```

```
class A
create make
feature -- Constructor
  make do end
feature -- Query
  new_once_array (s: STRING): ARRAY[STRING]
      -- A once query that returns an array.
    once
      create {ARRAY[STRING]} Result.make_empty
      Result.force (s, Result.count + 1)
    end
  new_array (s: STRING): ARRAY[STRING]
      -- An ordinary query that returns an array.
    do
      create {ARRAY[STRING]} Result.make_empty
      Result.force (s, Result.count + 1)
    end
end
```

local

bda: BANK_DATA_ACCESS

data1, data2: BANK_DATA.

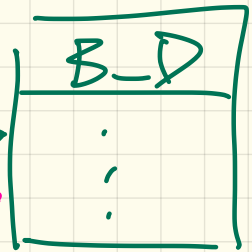
do

data1 := bda.data

↳ once routine.

data2 := bda2.data

the same once routine (2nd call)



The "s.make(7)" in CLIENT_1 doesn't compile because it's not able to use command `make`.

But how would it be able to use constructor `make` in the first place?

Doesn't only CLIENT_2 have access to the implementation of `make`?

Would the descendants of CLIENT_2 be able to make use of SUPPLIER's make command through inheritance?

Would the descendants of CLIENT_1 be able to make use of SUPPLIER's make constructor through inheritance?

class (A)
S.make

In class CLIENT_2, old_s cannot be instantiated by using the constructor `make`.

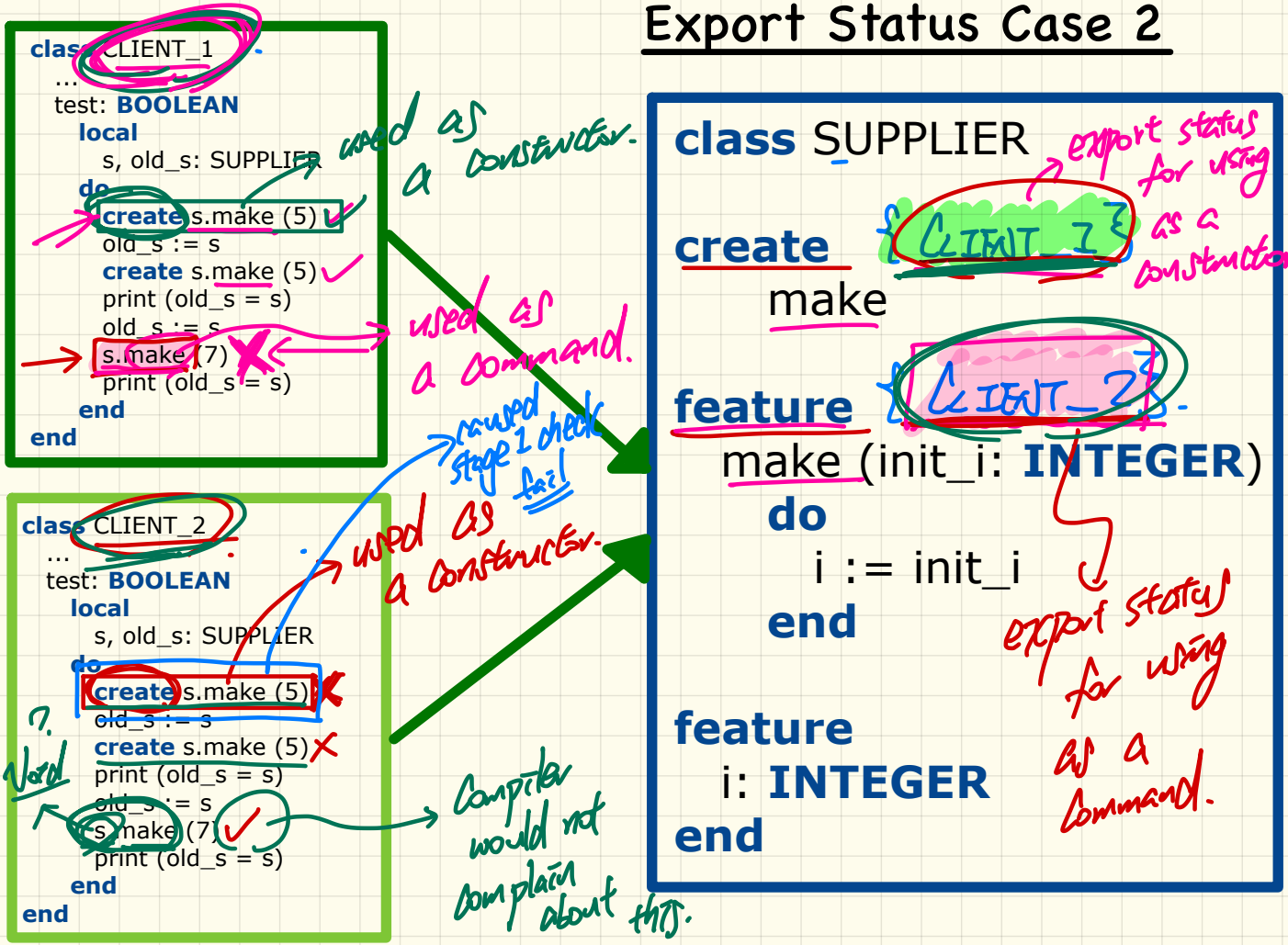
If it did not get instantiated, wouldn't there be no pointer pointing to the object?

How can its init_i be modified via feature `make`?

Stage 1 ok

A. Qualified calls (w.r.t. export status) are checked before void safety.

Export Status Case 2



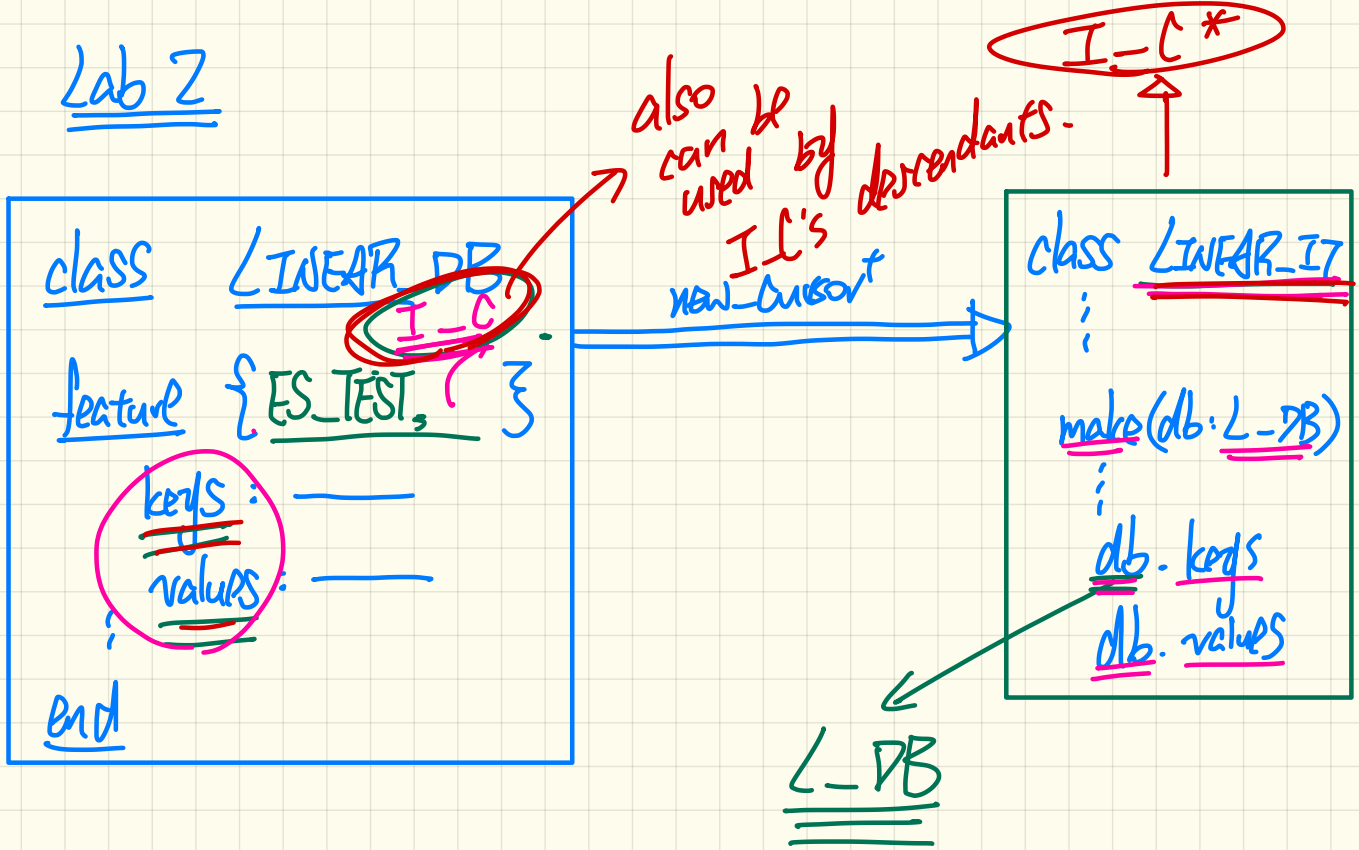
① Using a command with a context object

e.g. S.make(2)

independant

② Using a command to create an object

e.g. create S.make(2)


Lab 2
class LINEAR_DB
I_C
feature {ES_TEST, }
keys : ——
values : ——
end
also can be used by I_C's descendants.
I_C*
class LINEAR_I7
make(db: L_DB)
db.keys
db.values
L_DB

When we are creating a once routine in a class,
after the first call is made to this routine by any instance of this class,
that result is cached to ~~all instances of this class~~?

all callers of {DATA_ACCESS}.data

```
class DATA
        create {DATA_ACCESS} make
        feature {DATA_ACCESS} make...
end

expanded class DATA_ACCESS
        feature
          data: DATA
            once Result.make end
end
```

① 1st call
② Subsequent calls

Say I have two different clients who need access to DATA

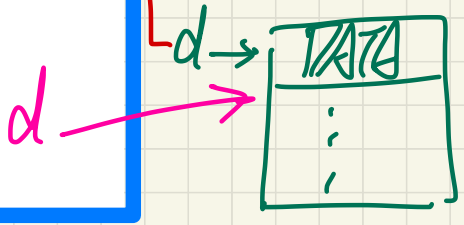
```
class CLIENT_1
        access: DATA_ACCESS
        d: DATA
        d := access.data
end

class CLIENT_2
        access: DATA_ACCESS
        d: DATA
        d := access.data
end
```

1st call

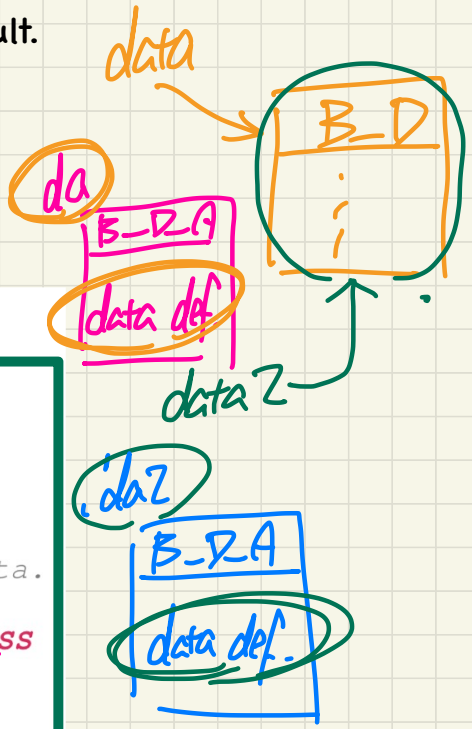
Since DATA_ACCESS is an expanded class,
both `access` objects in these clients would be different instances of DATA_ACCESS.

Assuming CLIENT_1 executes first then CLIENT_2,
does CLIENT_2 get a reference to the DATA object made by CLIENT_1?

What would have been the case if DATA_ACCESS was not expanded (and assuming `access` was properly initialized in each client as a separate object)?



We can avoid initializing the object as expanded classes do it by default. However, I was wondering in a class where we can have multiple constructors (some classes have make_empty and make_from_tuple) , how will expanded classes work in that case or are they allowed to have multiple constructors?



Supplier:

```
class BANK_DATA
create {BANK_DATA_ACCESS} make
feature {BANK_DATA_ACCESS}
  make do ... end
feature -- Data Attributes
  interest_rate: REAL
  set_interest_rate (r: REAL)
  ...
end
```

```
expanded class
  BANK_DATA_ACCESS
feature
  data: BANK_DATA
    -- The one and only access
  once create Result.make end
invariant data = data
```

Client:

```
class
  ACCOUNT
feature
  data: BANK_DATA
  make (...)
      -- Init. access to bank data.
    local
      data_access: BANK_DATA_ACCESS
    do
      data := data_access.data
      ...
    end
end
```

Writing **create** *data.make* in client's `make` feature does not compile. Why?