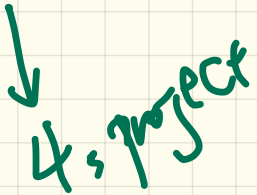
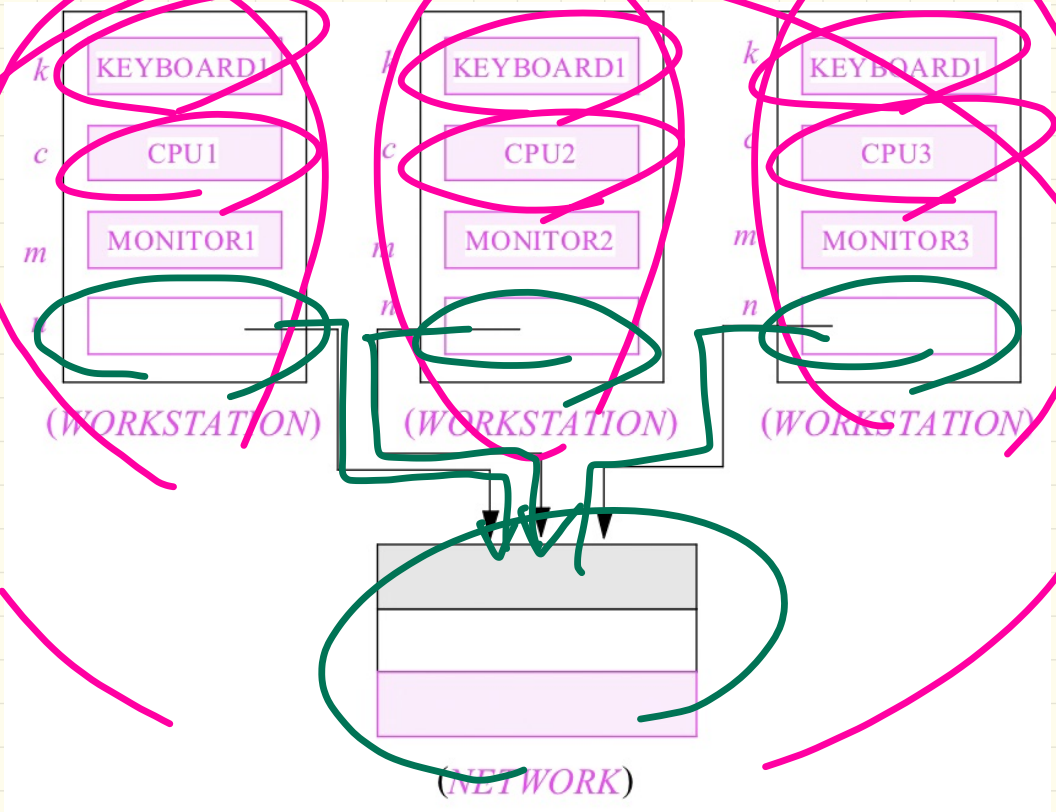
Lecture 12
Wednesday February 12

- Labtest 1 Review Session:
  Thursday 10am to 11am R S201
- Labtest 1 Seat Map
- Lab 3: ETF Turorial Videos



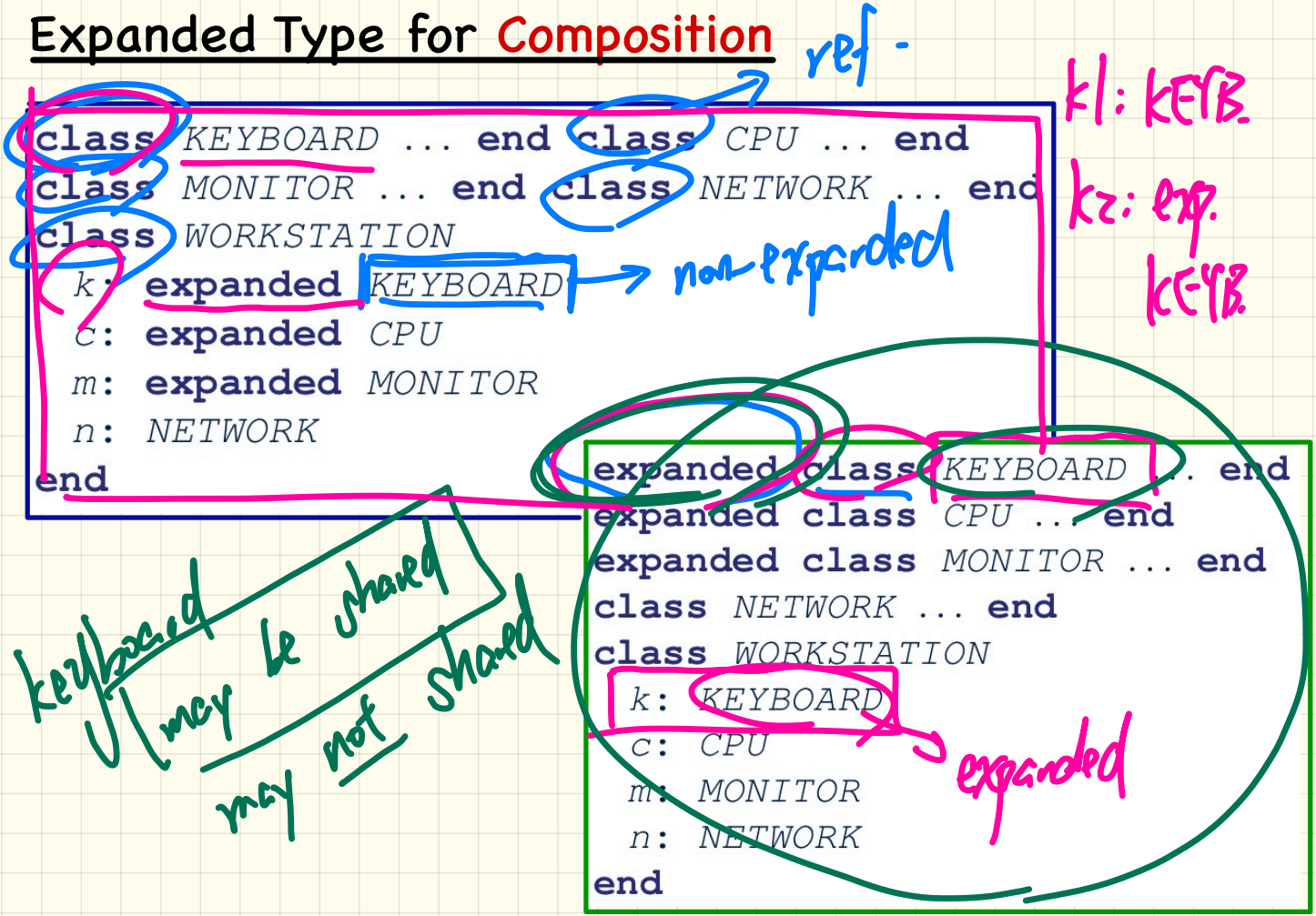
Modelling: Aggregation vs. Composition

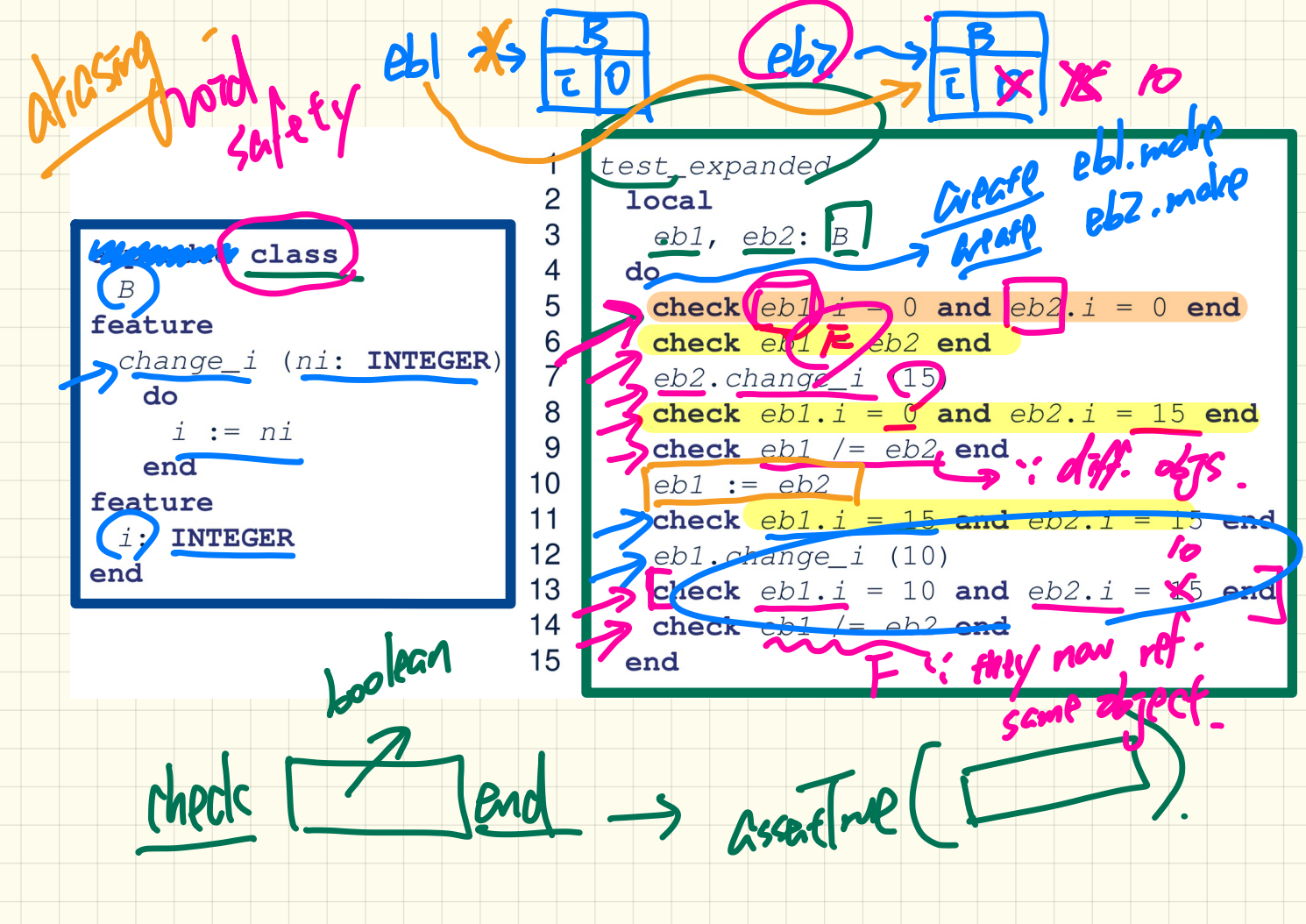


Expanded Type for Composition

```
class KEYBOARD ... end class CPU ... end
class MONITOR ... end class NETWORK ... end
class WORKSTATION
  k: expanded KEYBOARD
  c: expanded CPU
  m: expanded MONITOR
  n: NETWORK
end
```

```
expanded class KEYBOARD ... end
expanded class CPU ... end
expanded class MONITOR ... end
class NETWORK ... end
class WORKSTATION
  k: KEYBOARD
  c: CPU
  m: MONITOR
  n: NETWORK
end
```




aliasing
shared
safety
eb1
eb2
class
B
feature
change_i (ni: INTEGER)
do
i := ni
end
feature
i: INTEGER
end
test_expanded
local
eb1, eb2: B
do
check eb1.i = 0 and eb2.i = 0 end
check eb1 /= eb2 end
eb2.change_i (15)
check eb1.i = 0 and eb2.i = 15 end
check eb1 /= eb2 end
eb1 := eb2
check eb1.i = 15 and eb2.i = 15 end
eb1.change_i (10)
check eb1.i = 10 and eb2.i = 15 end
check eb1 /= eb2 end
end
create
eb1.make
eb2.make
F: diff. objs.
F: they now ref. same object.
boolean
check
end
assert

class B

obj1 : B

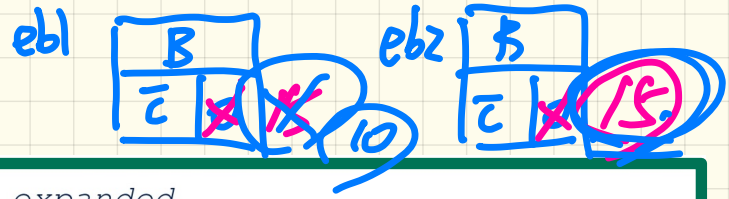
obj2 : expanded B

obj1 := obj2

obj1 = obj2

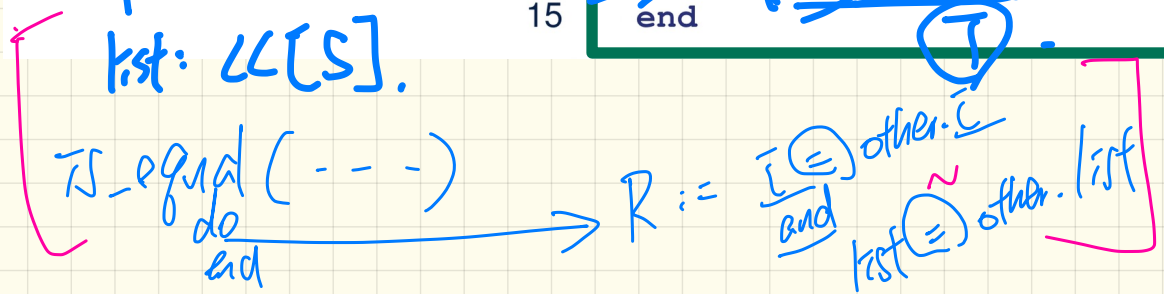
not compiling ...

Use of Expanded Type




```
expanded class
  B
feature
  change_i (ni: INTEGER)
    do
      i := ni
    end
feature
  i: INTEGER
end
```

```
1  test_expanded
2    local
3      eb1, eb2: B
4    do
5      check eb1.i = 0 and eb2.i = 0 end
6      check eb1 = eb2 end
7      eb2.change_i (15)
8      check eb1.i = 0 and eb2.i = 15 end
9      check eb1 /= eb2 end
10     eb1 := eb2
11     check eb1.i = 15 and eb2.i = 15 end
12     eb1.change_i (10)
13     check eb1.i = 10 and eb2.i = 15 end
14     check eb1 /= eb2 end
15   end
```

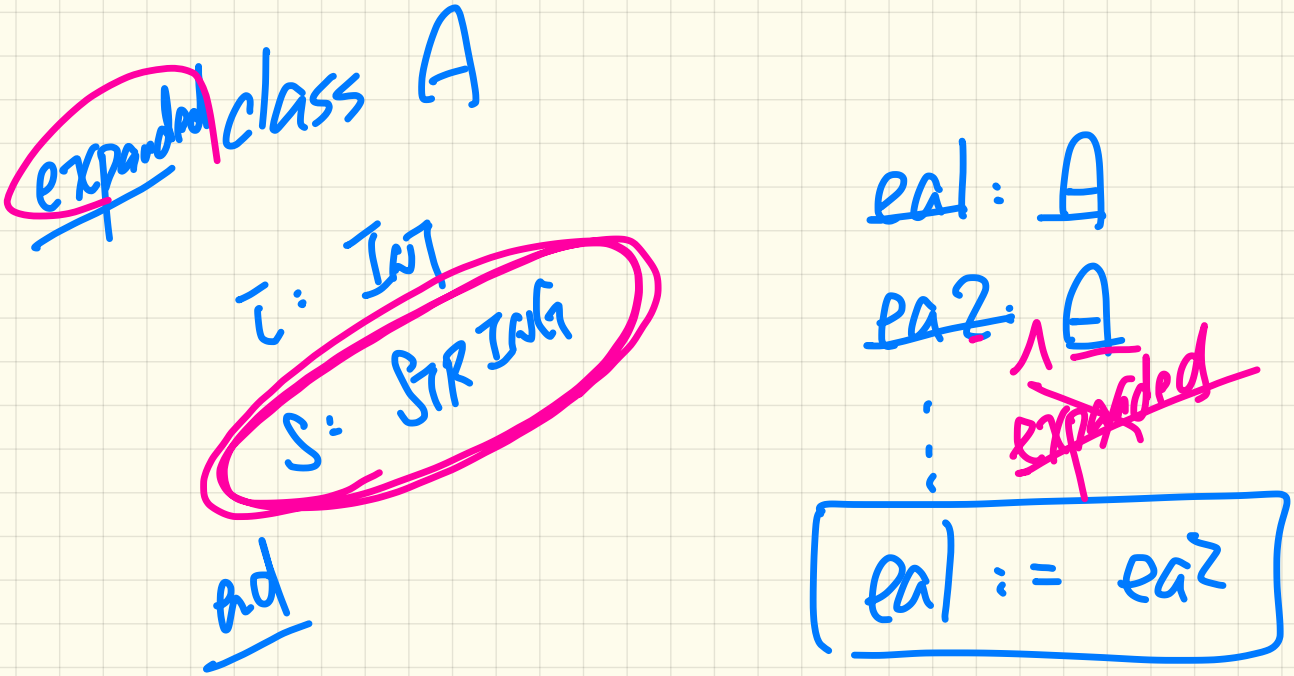


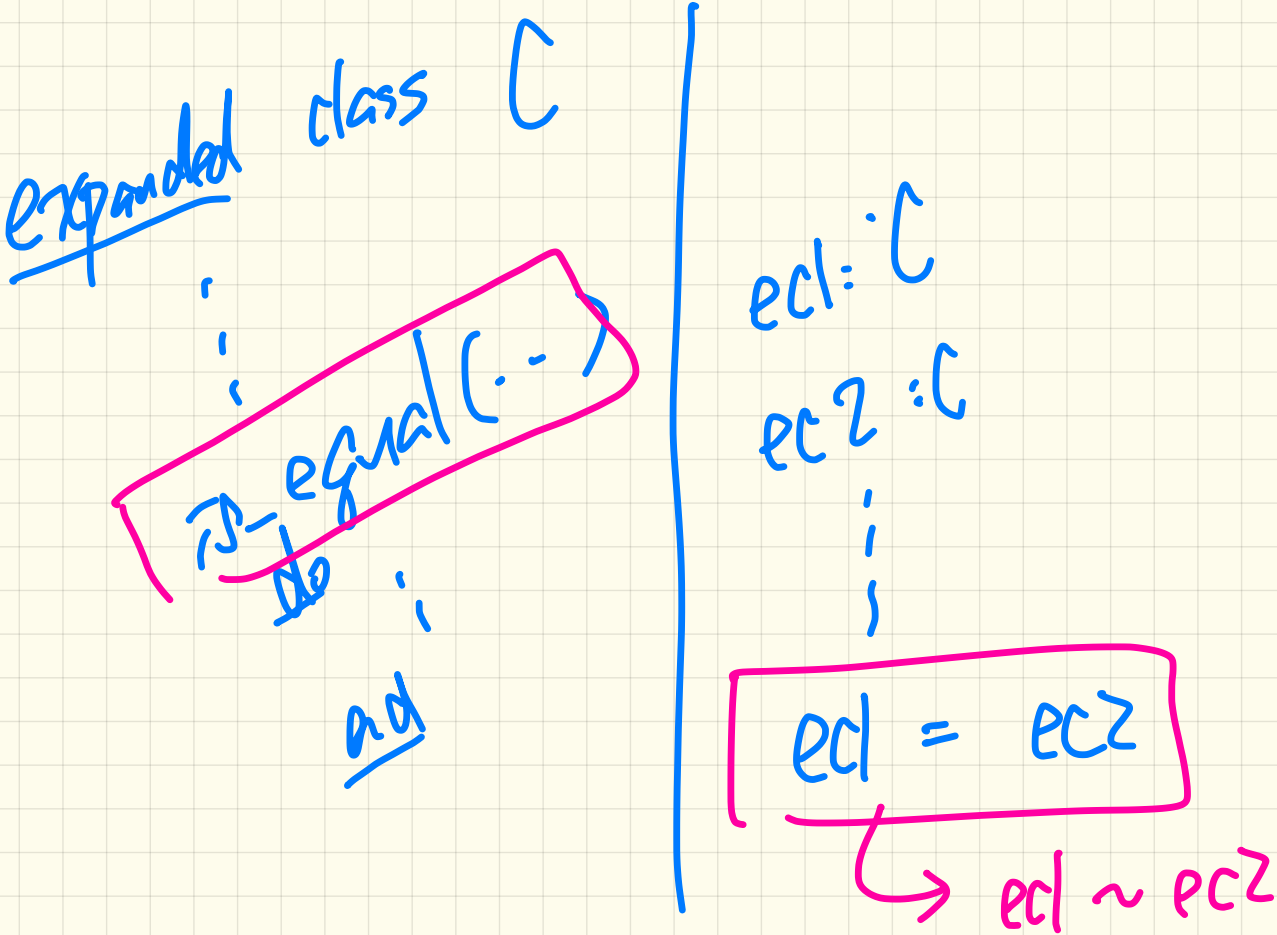
```
expanded class B
do
    i: INTEGER
end
```

eb1 : B
↳ initialize all attributes to default values

```
expanded class B
inherit ANY
    redefine default_create end
default_create do i := 5 end
```

eb2 : B
create eb2.default_create


expanded class A
i : INT
s : STRING
end
a1 : A
a2 : A
expanded
a1 := a2


expanded class C
is_equal (...)
do
end
ec1 : C
ec2 : C
ec1 = ec2
ec1 ~ ec2

obj1, obj2 : A.

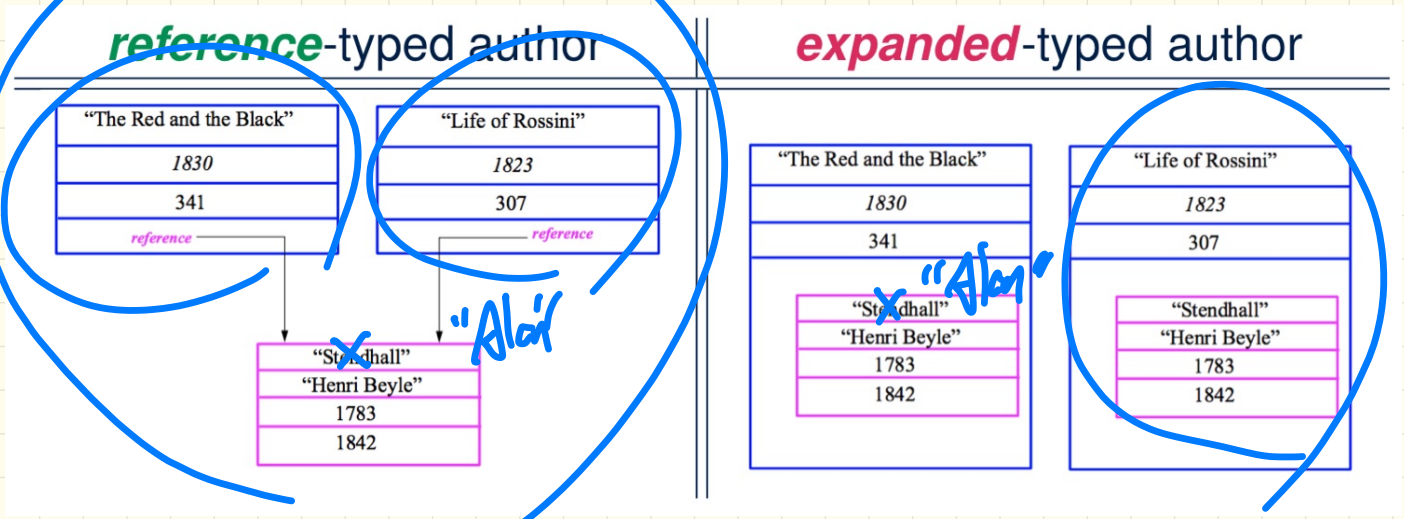
obj1 = obj2

① A is not expanded,
compare address is equal

② A is expanded,
compare contents

obj1 ~ obj2

Reference Type or Expanded Type



Shared Data via Inheritance

Cohesion

Single Choice Principle

features in a single class should serve the same purpose. (UNIX: do one thing & do it well).

Descendant:

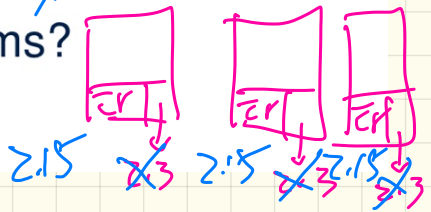
```
class DEPOSIT inherit SHARED_DATA
      -- 'maximum_balance' relevant
end

class WITHDRAW inherit SHARED_DATA
      -- 'minimum_balance' relevant
end

class INT_TRANSFER inherit SHARED_DATA
      -- 'exchange_rate' relevant
end

class ACCOUNT inherit SHARED_DATA
feature
      -- 'interest_rate' relevant
      deposits: DEPOSIT_LIST
      withdraws: WITHDRAW_LIST
end
```

Ancestor:

```
class
  SHARED_DATA
feature
  interest_rate: REAL
  exchange_rate: REAL
  minimum_balance: INTEGER
  maximum_balance: INTEGER
  ...
end
```

2.15

Problems?



Shared Data via Inheritance

Cohesion

Single Choice Principle

```
class
    SHARED_DATA
feature
    interest_rate: REAL
    exchange_rate: REAL
    minimum_balance: INTEGER
    maximum_balance: INTEGER
    ...
end
```

ir → 0.11 ~ 0.09

d1 →

	DEPOSIT
ir	0.11 0.09
er	2.34
min	1000
max	1000000

d2

	DEPOSIT
ir	0.11
er	2.34
min	1000
max	1000000

w1 →

	WITHDRAW
ir	0.11
er	2.34
min	1000
max	1000000

w2

	WITHDRAW
ir	0.11
er	2.34
min	1000
max	1000000

d1, d2: DEPOSIT
w1, w2: WITHDRAW
t1, t2: INTERNATIONAL_TRANSFER
...
d1.set_max_balance
w2.set_min_balance
t2.set_exchange_rate

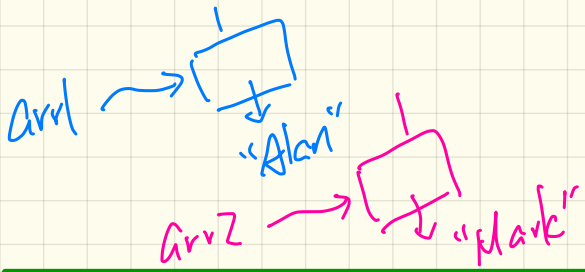
t1 →

	TRANSFER
ir	0.11
er	2.34
min	1000
max	1000000

t2

	TRANSFER
ir	0.11
er	2.34
min	1000
max	1000000

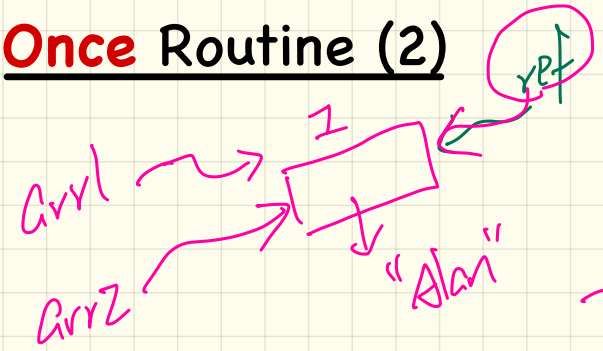
Once Routine (1)




```
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
```

```
test_query: BOOLEAN
  local
    a: A
    arr1, arr2: ARRAY[STRING]
  do
    create a.make

    arr1 := a.new_array ("Alan")
    Result := arr1.count = 1 and arr1[1] ~ "Alan"
    check Result end

    arr2 := a.new_array ("Mark")
    Result := arr2.count = 1 and arr2[1] ~ "Mark"
    check Result end

    Result := not (arr1 = arr2)
    check Result end
  end
```

Once Routine (2)




```
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
```

```
test_once_query: BOOLEAN
  local
    a: A
    arr1, arr2: ARRAY[STRING]
  do
    create a.make
    arr1 := a.new_once_array ("Alan")
    Result := arr1.count = 1 and arr1[1] ~ "Alan"
    check Result end
    arr2 := a.new_once_array ("Mark")
    Result := arr2.count = 1 and arr2[1] ~ "Alan"
    check Result end

    Result := arr1 = arr2
    check Result end
end
```