EECS 3311 Fall 2017

Software Design

Monday September 11

Lecture 1

class MicrowaveUser → Client

m heat(obj)

↓

Supplier
& Microwave

Specification
vs.
Implementation (code)

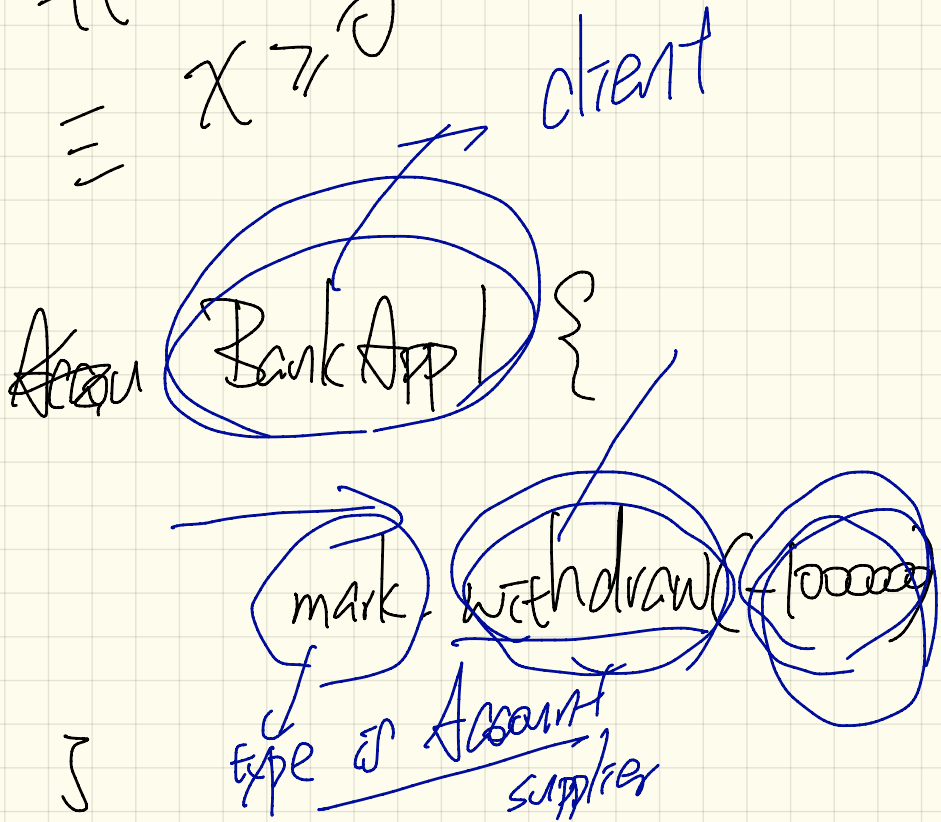
← before-execution
client makes sure
locked & on

← after execution
supplier makes
sure obj is
heated

positive $x > 0$

non-negative $\neg(x < 0)$

$\equiv x \geq 0$



preconditions vs. exceptions

≠

exception condition is a logical negation of the corresponding precondition

divide(int x, int y) $\frac{x}{y}$

DivisionByZeroException. → y == 0

Precondition (when I can do division) → y ≠ 0

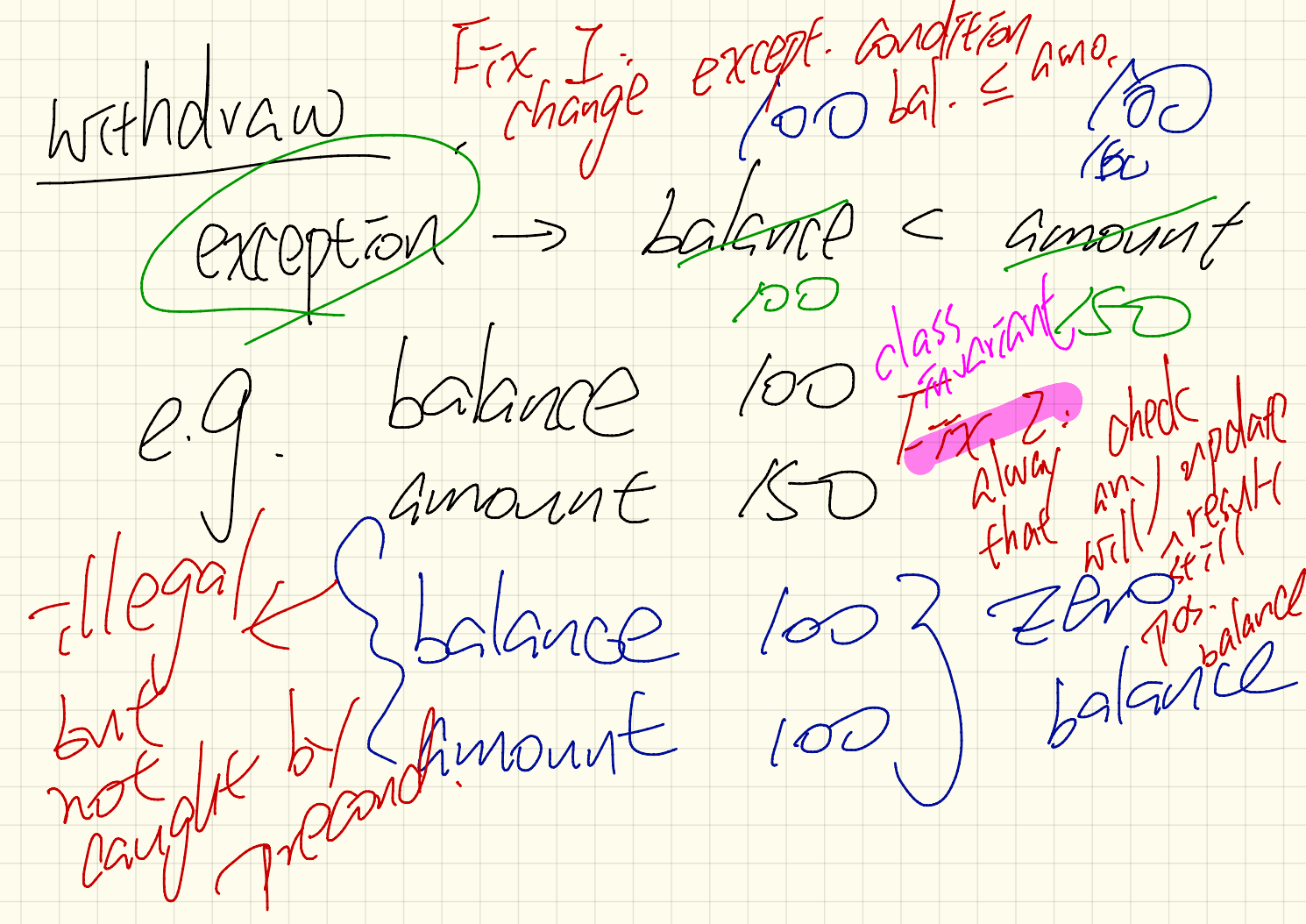
"code smells"

↳ unnecessary repetition.

↳ resolve by using code reusing

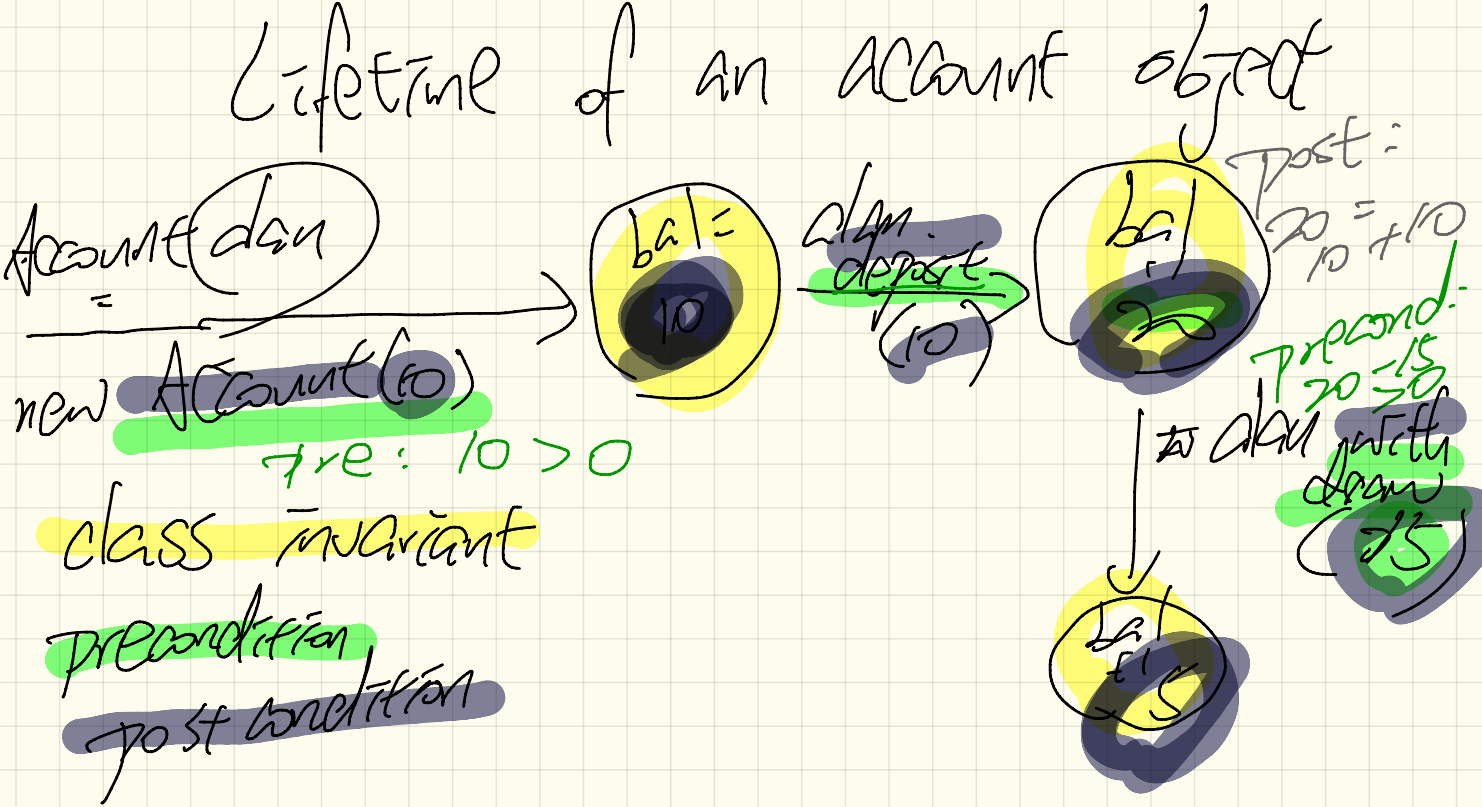
Wednesday Sept. 13

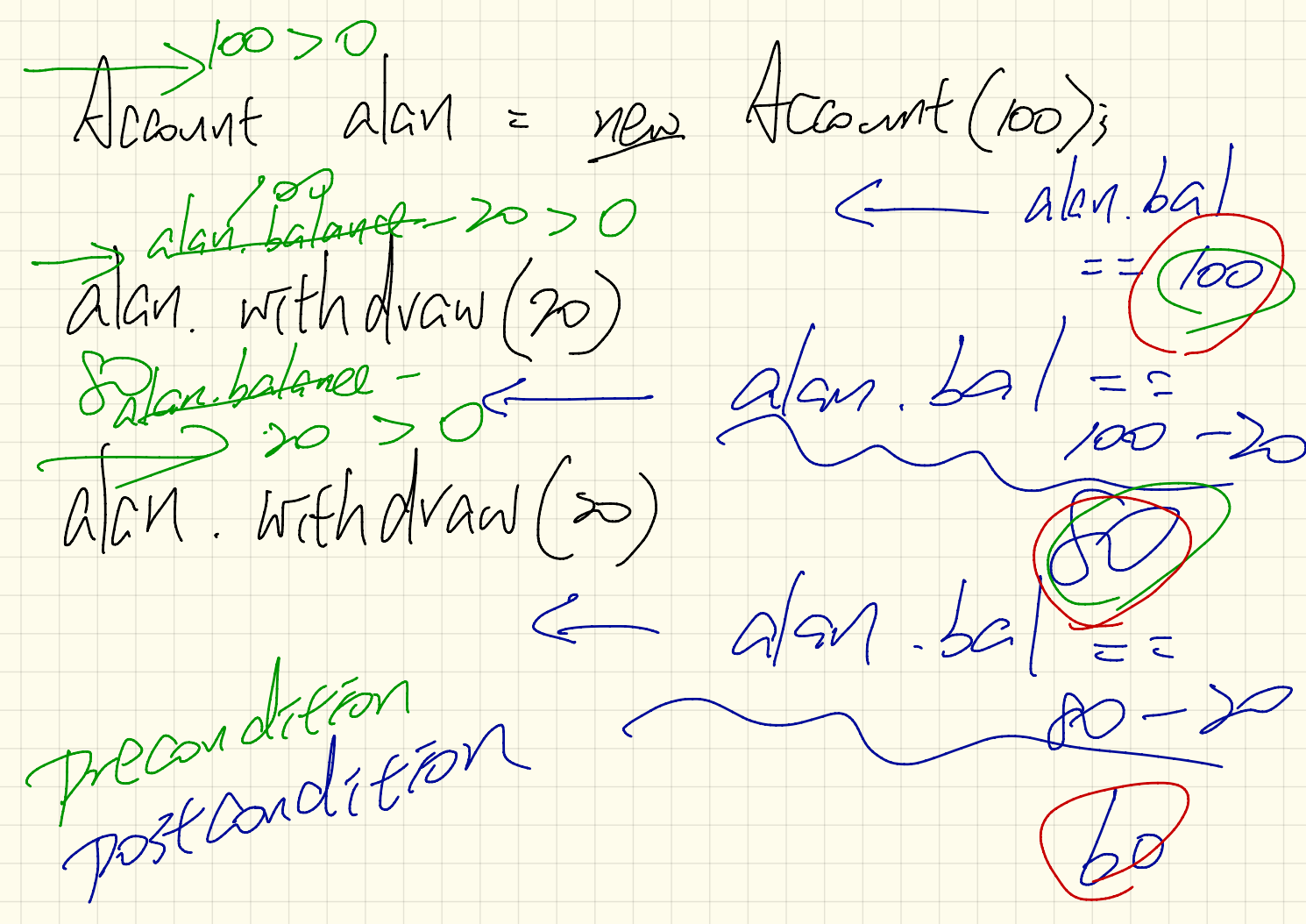
Lecture 2


withdraw
exception → balance < amount
Fix 1: change except. condition
100 bal. ≤ amo.
150
e.g. balance 100
amount 150
class invariant
Fix 2: check always that amount will update result
illegal
but not caught by precondition
balance 100
amount 100
zero
pos. balance
balance

3 kinds of contract

Lifetime of an account object




100 > 0
Account alan = new Account(100);
alan.bal == 100
alan.balance -- 20 > 0
alan.withdraw(20)
alan.bal == 100 - 20
80
alan.balance - 20 > 0
alan.withdraw(20)
alan.bal == 80 - 20
60
precondition
postcondition

precond of m1 is satisfied

$\wedge$ conjunction

obj.m1(...)

← postcond of m1 is satisfied

precond of m2 is satisfied

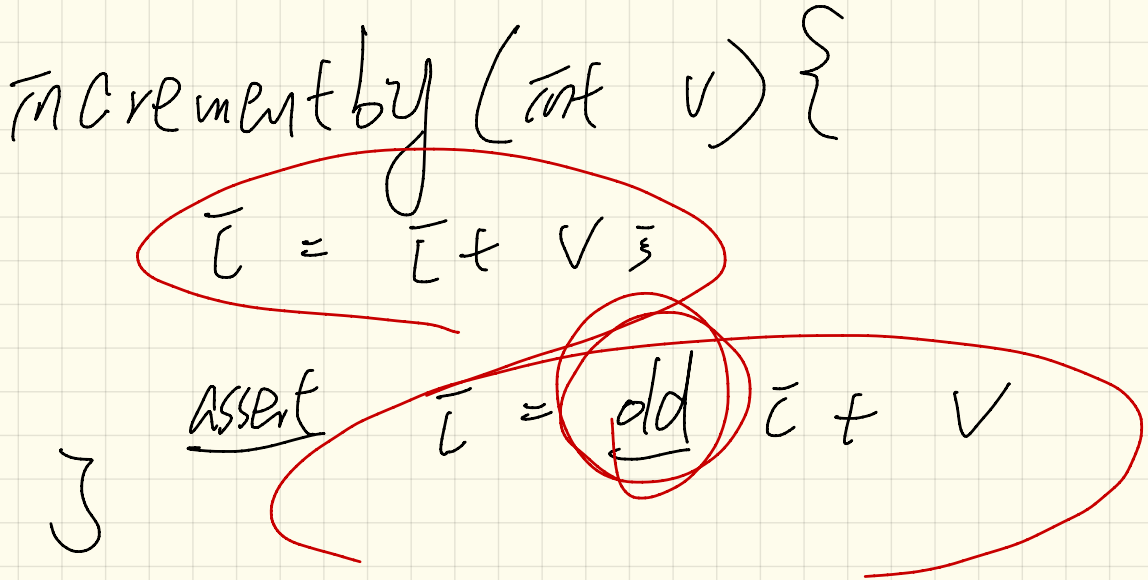
obj.m2(...)

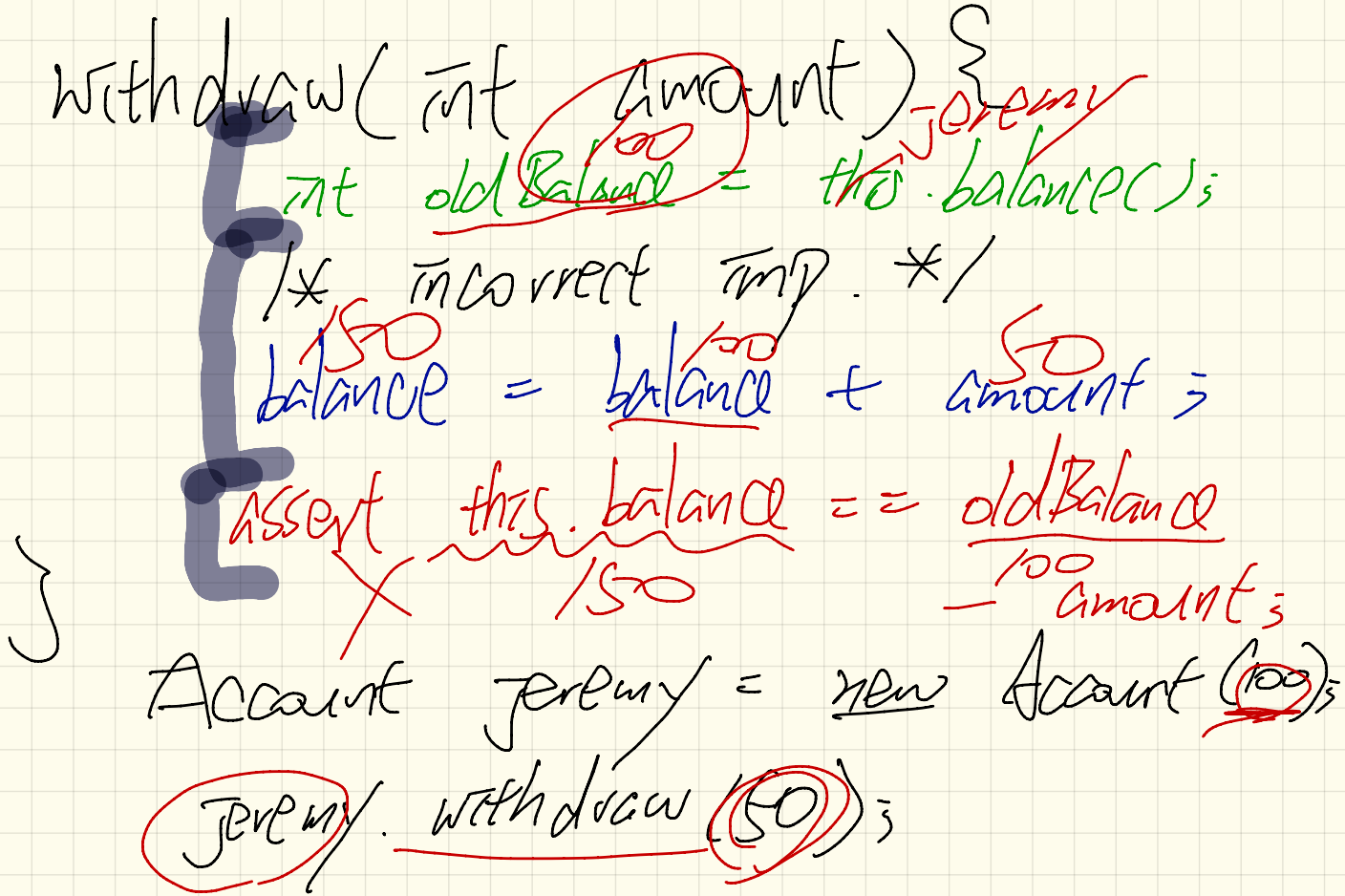
$\wedge$ INV is satisfied

← postcond of m2 is satisfied

$\wedge$ INV is satisfied

Think of INV is the common postcond. of every method


incrementby(int v){
c = c + v;
assert c = old c + v
}


withdraw(int amount) {
int oldBalance = this.balance();
/* incorrect imp. */
balance = balance + amount;
assert this.balance == oldBalance - amount;
}
Account jeremy = new Account(100);
jeremy.withdraw(50);
jeremy
100
150
50
150
100
50

void Sort (int[] x) {

implementation

quick
merge

selection
insertion

bubble sort
heap sort

post condition

$\forall i \mid 0 \le i < x.length - 1 : x[i] \le x[i+1]$

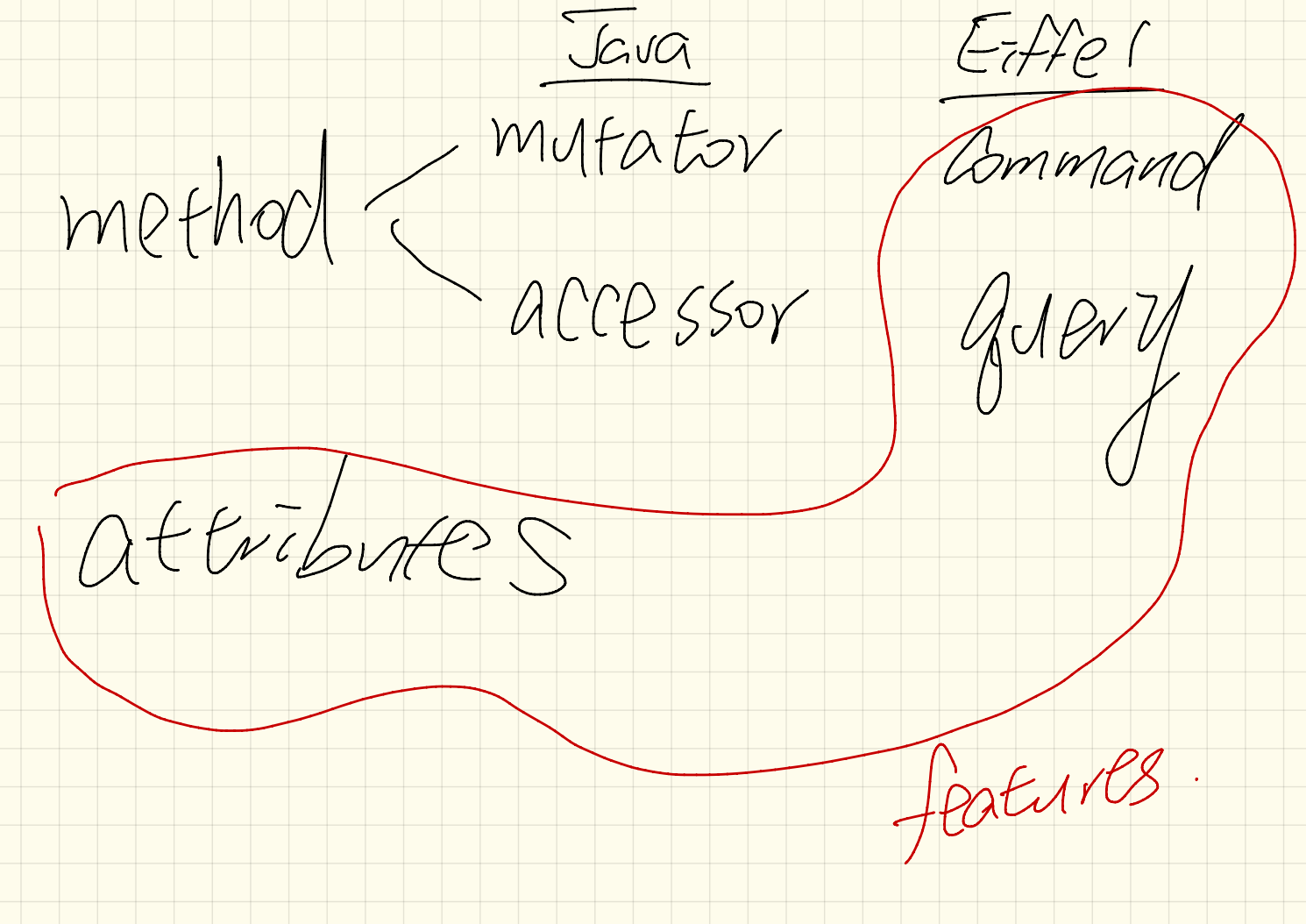
}

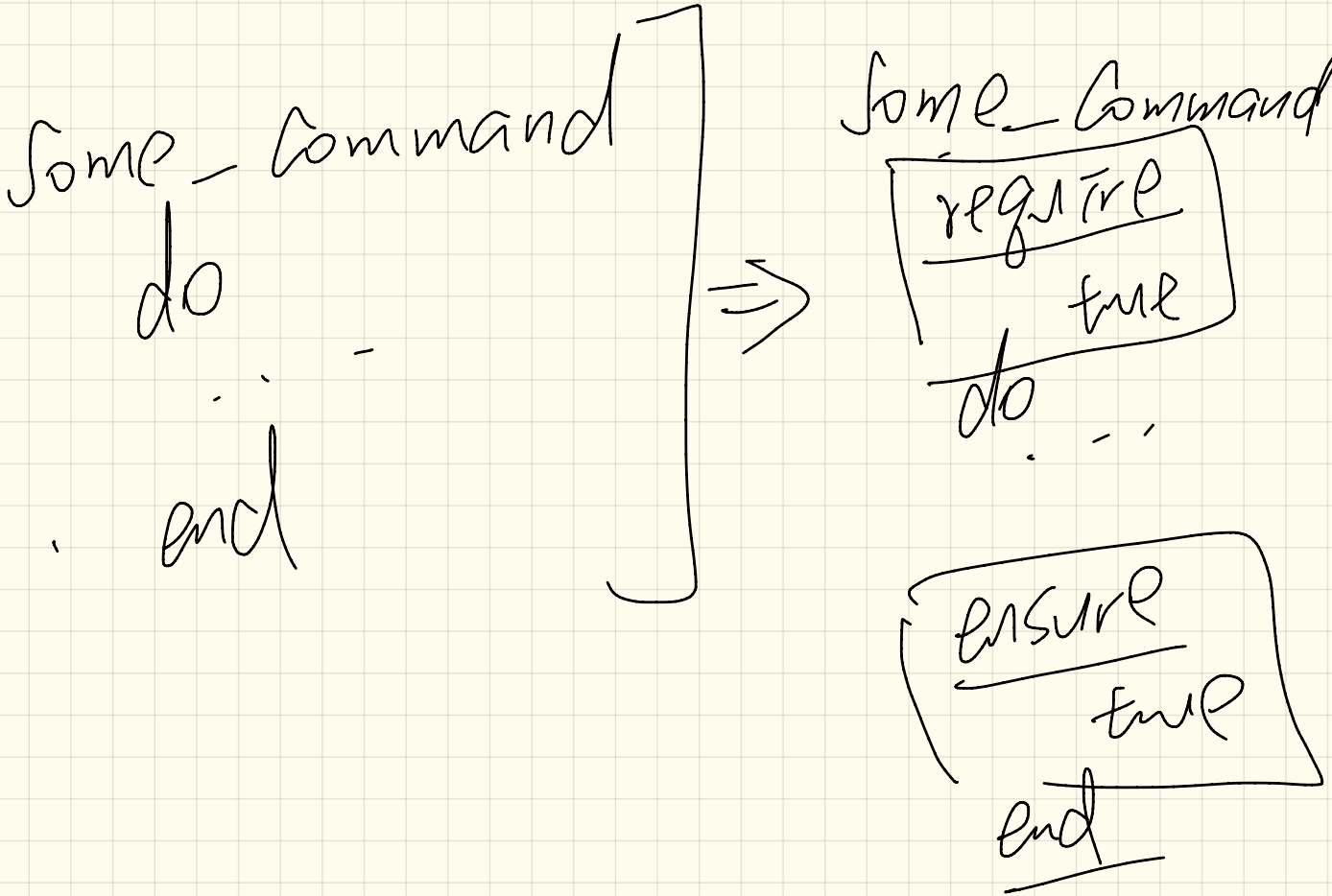
1. Correctness → what to achieve (contracts)

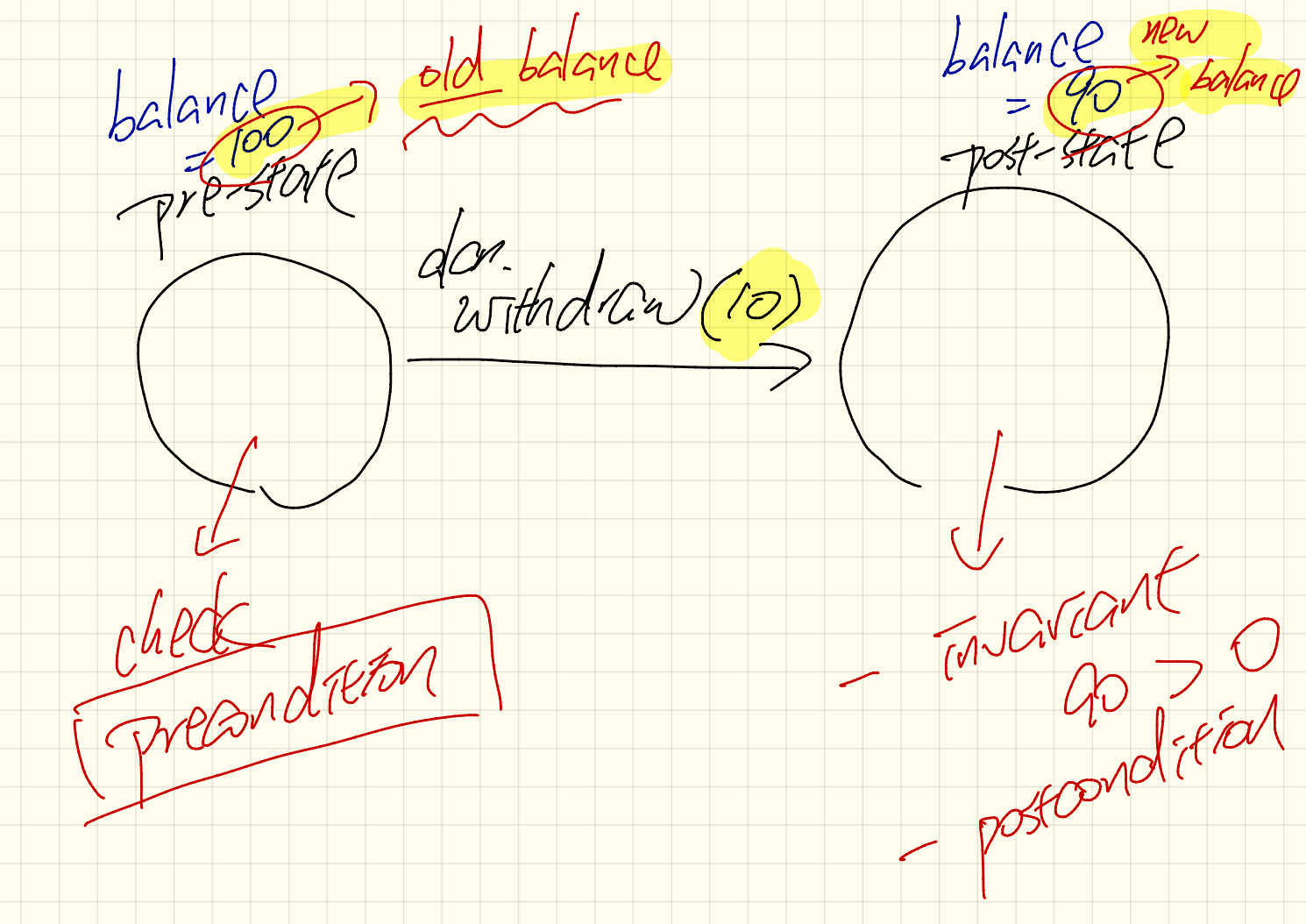
2. Efficiency → how to achieve

Lecture 3

Monday Sept. 18

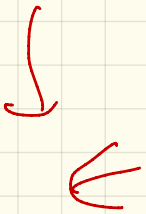

Java
Eiffel
method
mutator
accessor
command
query
attributes
features.


some_command
do
...
end
⇒
some_command
require
true
do
...
ensure
true
end


balance = 100
pre-state
old balance
do_withdraw(10)
balance = 90
post-state
new balance
check precondition
- invariant
90 > 0
- postcondition

int i

i : INTEGER



object creation

Account alan = new Account ("alan", 10);

Eiffel

create {ACCOUNT} alan.
make ("alan", 10)

void m() { ... }

---

m()

db

...

end

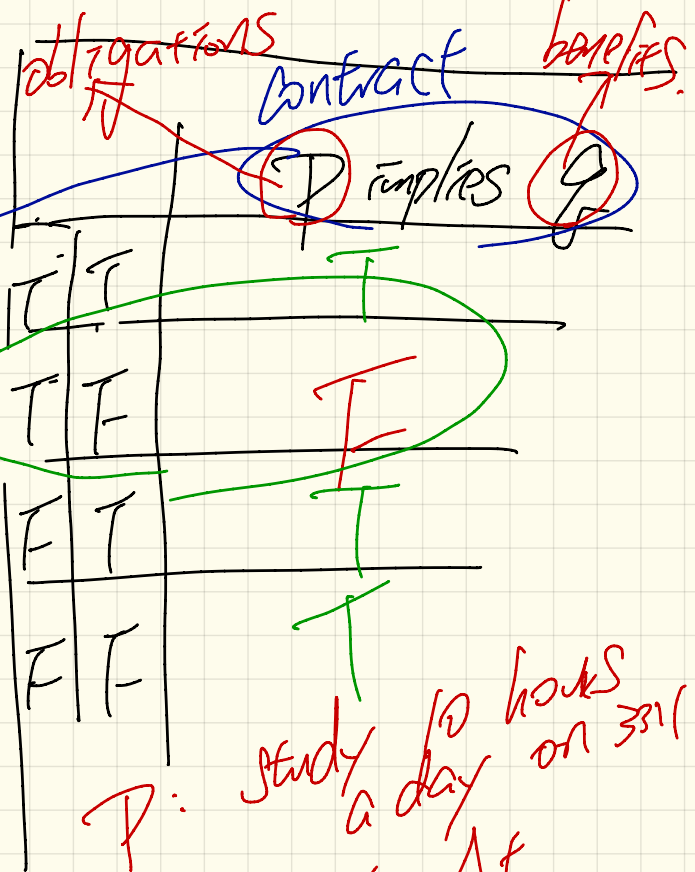
Logical Operators

$\wedge$ and

$\vee$ or

$\Rightarrow$ implies

$\neg$ not



$P \Rightarrow Q$ is True if contract is not violated.

P: study 10 hours a day on 331

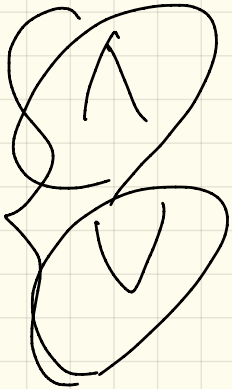
Q: get A+

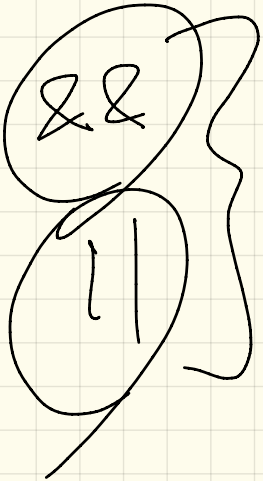
In Java SCE

E_1 && E_2

⇒ no F don't care

¬ !





not quite the same ! short-circuit effect -

SCE.

int a[]

&& ≠ ∧

0 ≤ i && i < a.length a[i]
&&

Eiffel

and ∧

or ∨

without SCE

and then ≡ &&

or else ≡ ||

Math

~~Array is sorted~~

a : ~~ARRAY~~ [INTEGER]

a.lower a.upper

$\forall i : \text{INTEGER} \mid a.lower \leq i \leq a.upper \bullet$

$a[i] \leq a[i+1]$

Eiffel

across

a.lower |..| (a.upper − 1)

a list integers.

as

i

all

some

i.item

cursor

i.item

end

a[i] <= a[i+1]

across

1 | 1..| 10 as i

all some

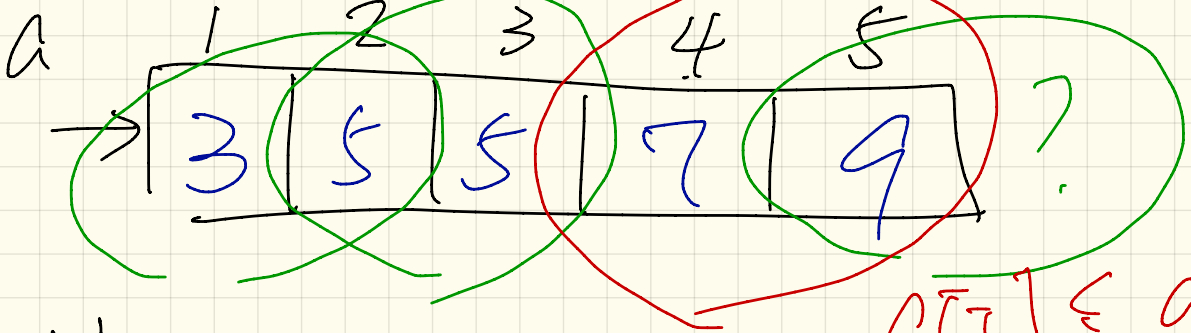
i. item \ 2 = 0

end

boolean expression

false

true



$a[i] \leq a[i+1]$

$\forall i : \text{INTEGER} \mid 1 \leq i \leq 5 \bullet$

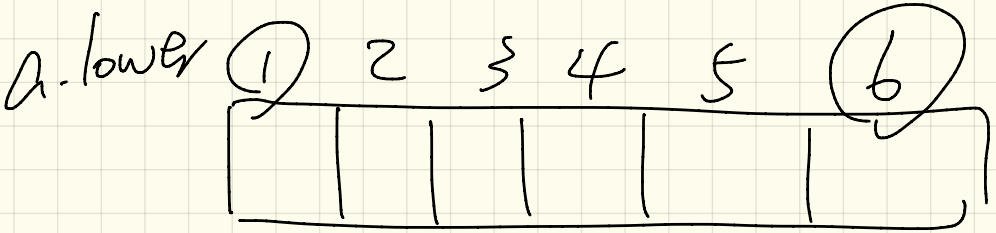
$5 - 1$

$a[i] \leq a[i+1]$

$a[1] \leq a[2]$ $\quad a[5] \leq a[6]$

$a[2] \leq a[3]$

$\vdots$



Size: a.upper − a.lower + 1

create a.make_empty

a.lower = 1

a.upper = 0

0 − 1 + 1 = 0

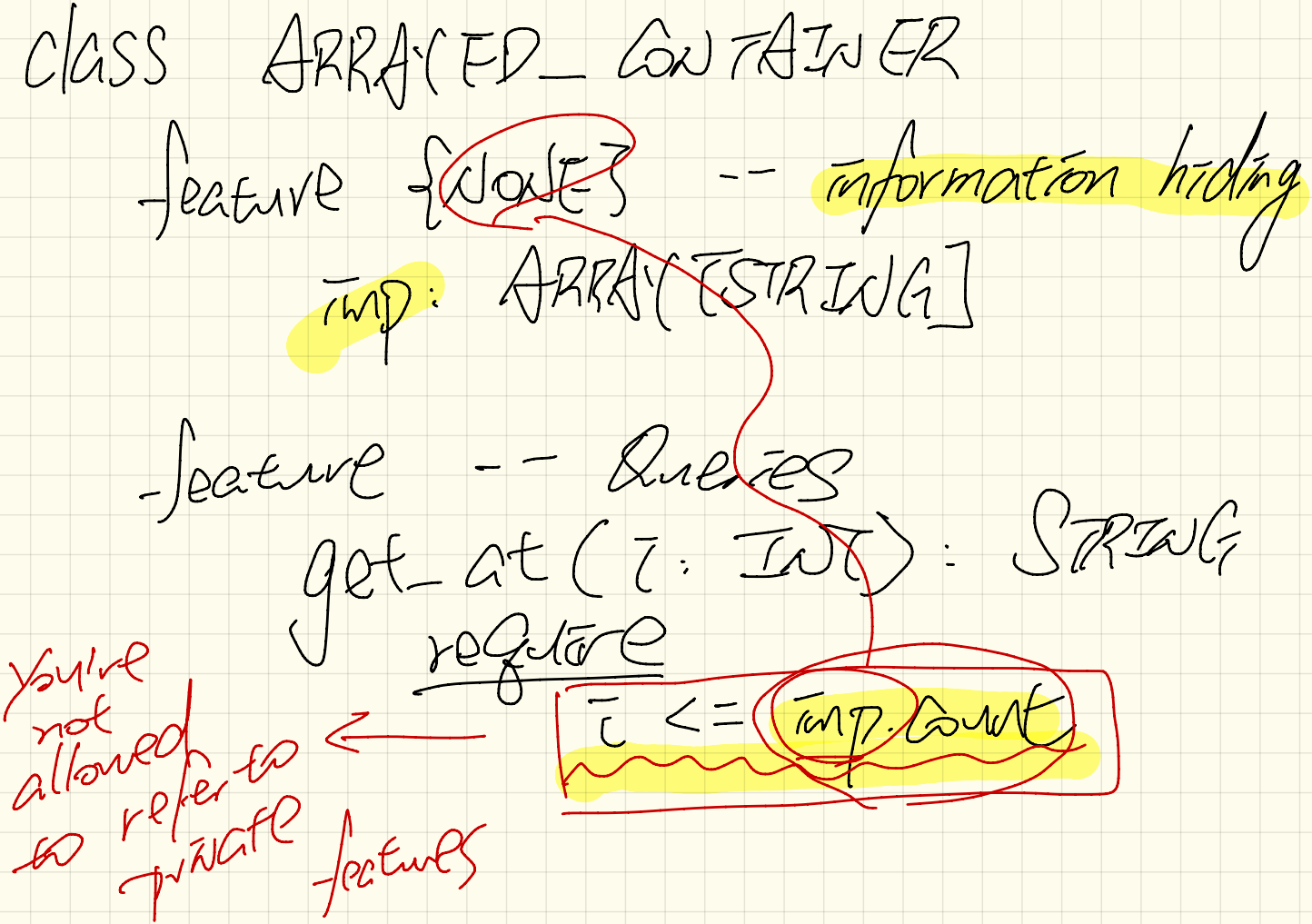
Lecture 4

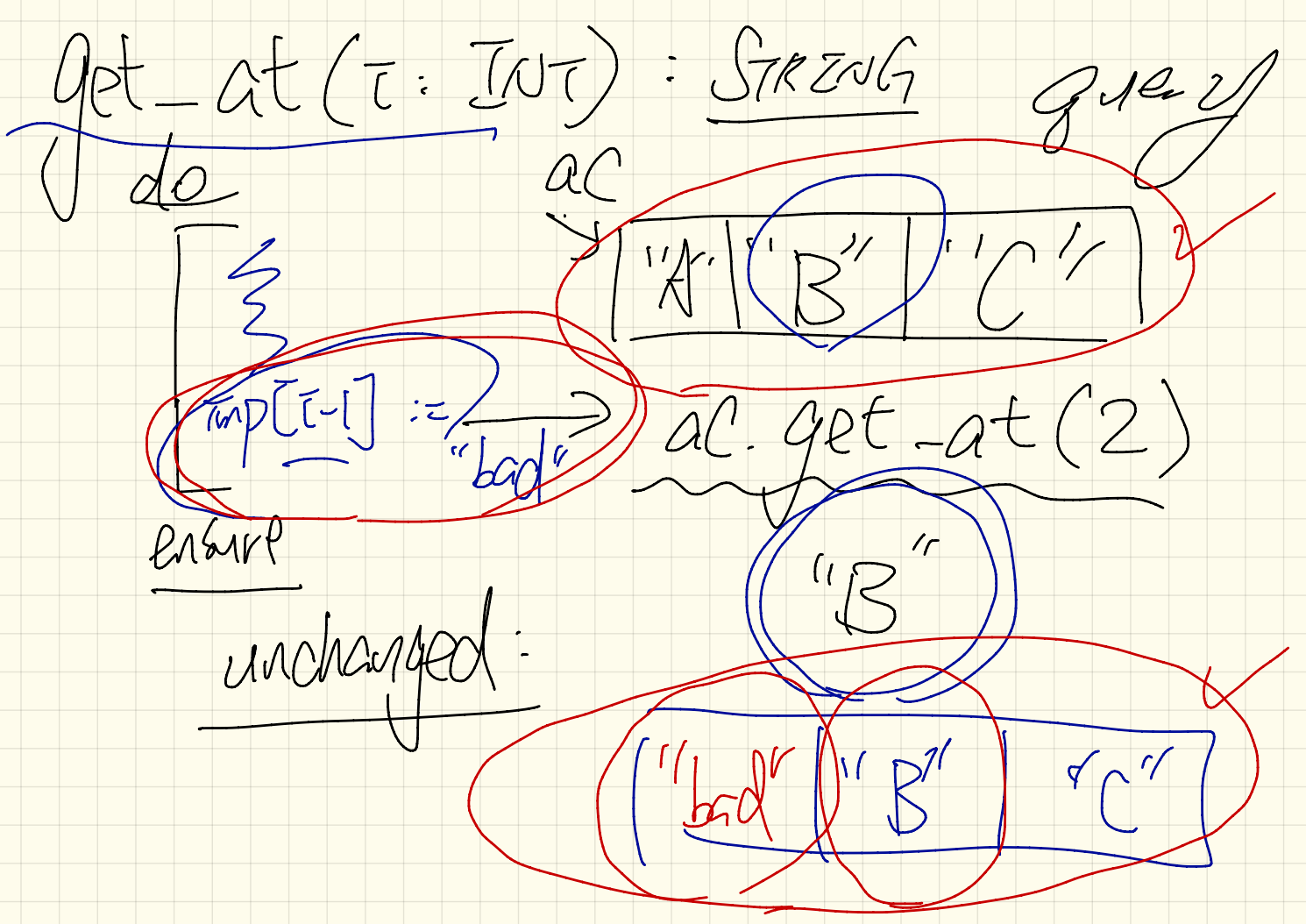
Wednesday Sep. 20

office hours

Mon Tue Thu

13:30 – 15:30


class ARRAYED_CONTAINER
feature {NONE} -- information hiding
imp: ARRAY[STRING]
feature -- Queries
get_at(i: INT): STRING
require
i <= imp.count
you're not allowed to refer to private features

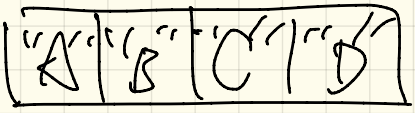

get_at (i : INT) : STRING
query
do
ac
"A"
"B"
"C"
1
2
tmp[i-1] := "bad"
ac.get_at(2)
ensure
"B"
unchanged:
"bad"
"B"
"C"

Kinds of tests

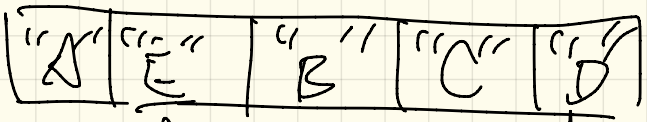
3. Test to fail (postcond)
   insert_at(2, "E")

test query

1. Compare expected value vs. actual value

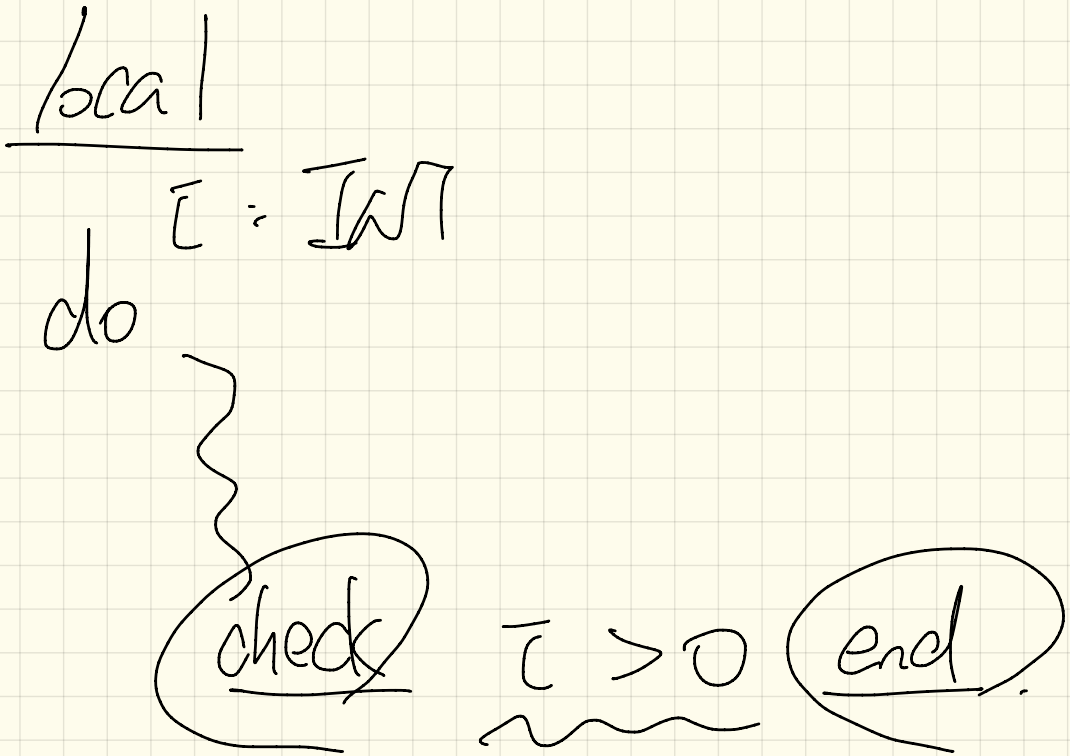
ac → 

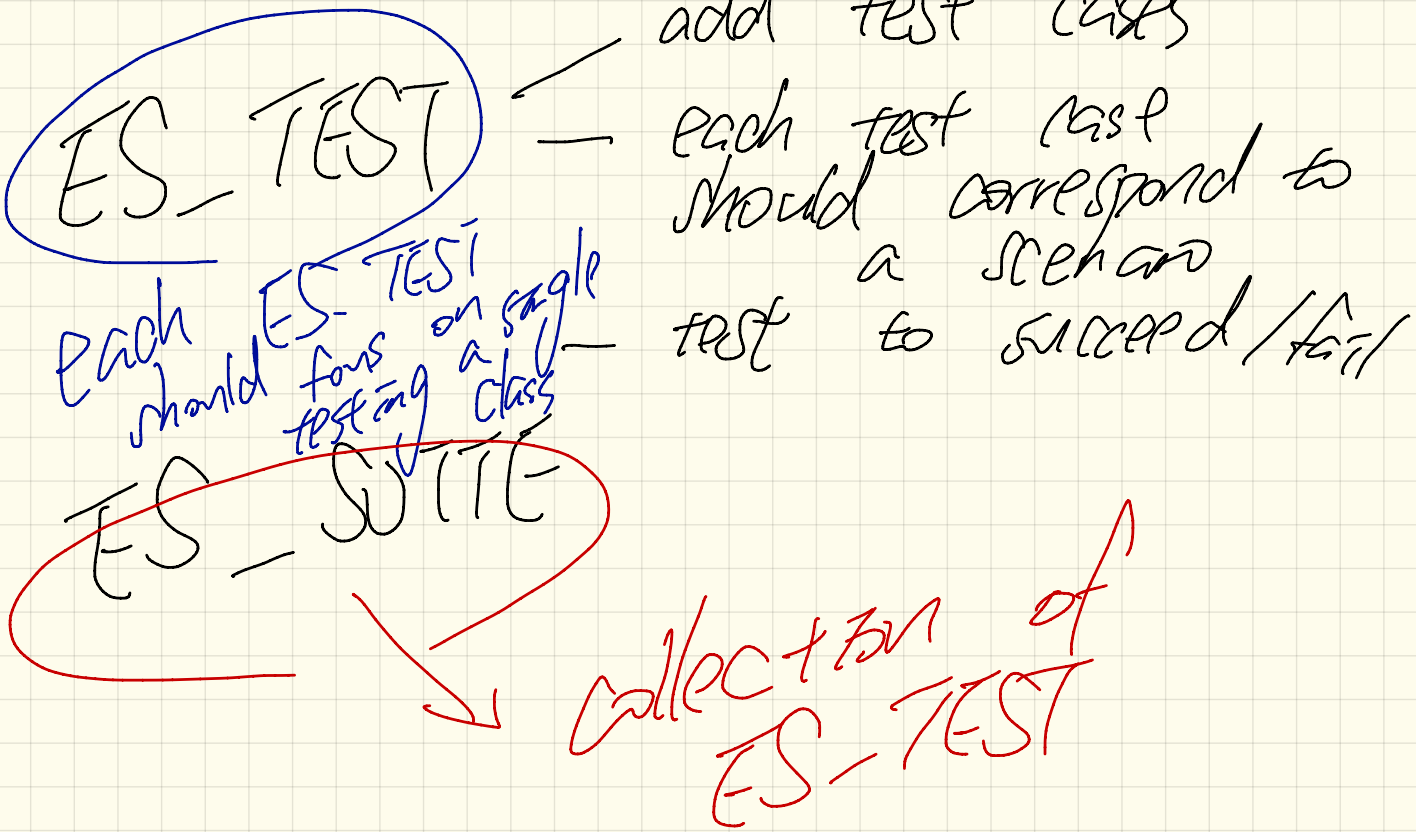
insert_at(2, "E")



test command

2. Test to fail (precondition)
   ac.get_at(5)


local
i : INT
do
check
i > 0
end


ES_TEST
add test cases
each test case should correspond to a scenario
test to succeed/fail
each ES_TEST should focus on single testing a class
ES_SUITE
collection of ES_TEST

Java.

Account acc = new Account();

b. addAccount(acc)

b. addAccount(new Account());

Eiffel

create {ACCOUNT} acc.make

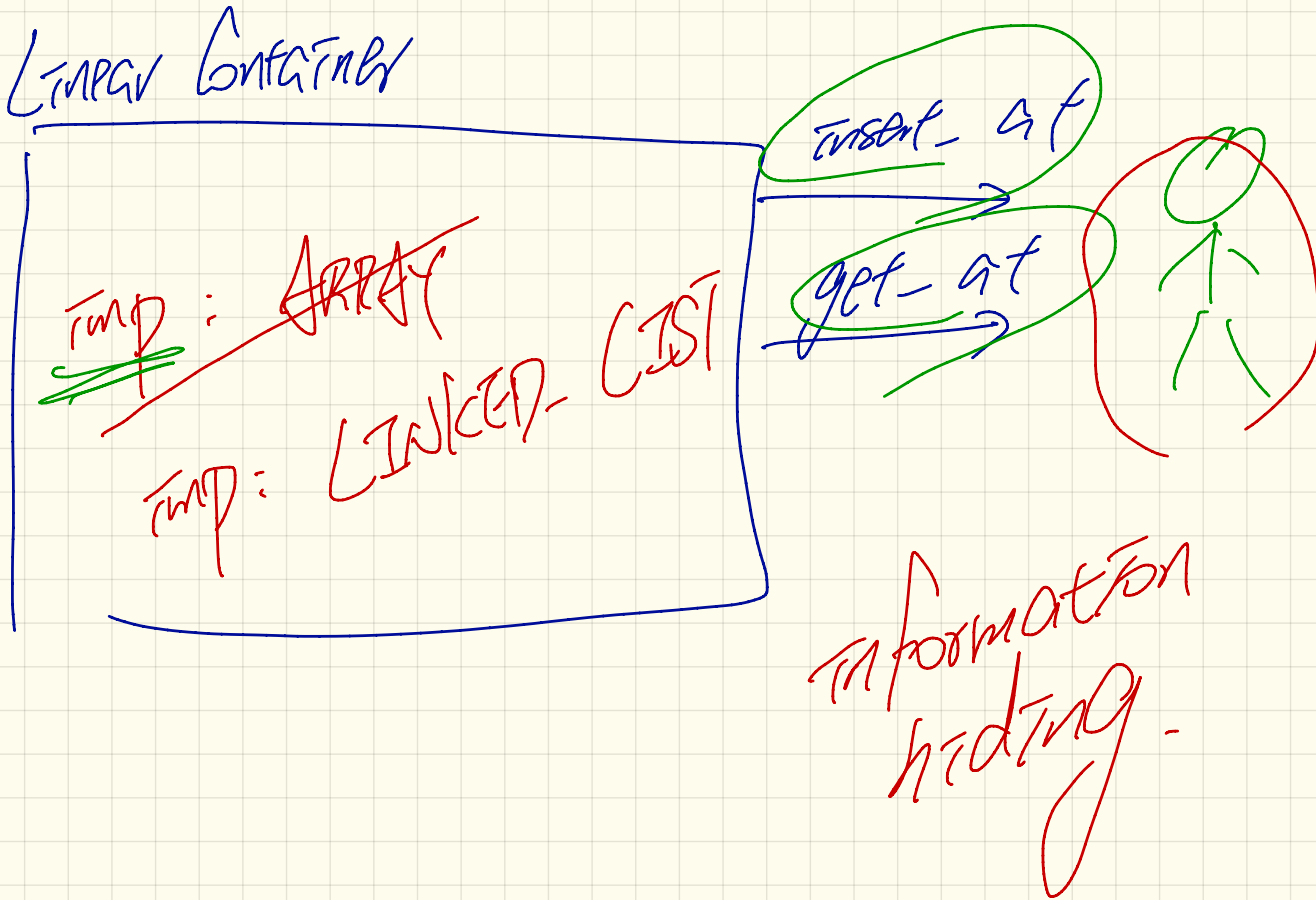
create {ACCOUNT}.make

Feature to test:

```
add2 (i: INT): INT
    -- given i, returns 2 + i
```

Test query

```
test_Add2ForLoopInputs: BOOLEAN
    local
        i: INT
    do
        comment ("...")
        from
            i := 1
        until
            i > 100
        do
            Result := add2 (i) = 2 + i
            check Result end
        end
    end
```


Linear Container
temp : ARRAY
temp : LINKED- CIST
insert_at
get_at
information
hiding -

ADT STACK[G] = generic parameter

ARRAY[STRING]

ARRAY[PERSON]

ARRAY[INTEGER]

operations

ARRAY[G] generic class

Operations

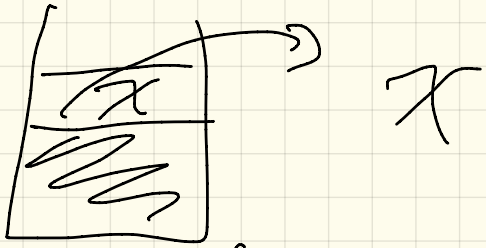
pop
remove : STACK[G] → STACK[G]

put : STACK[INT] → INT → STACK[INT]

empty : STACK[INT]

peek
item : STACK[INT] → INT

Property $S.push(x)$



After pushing an item x into the stack, an immediate pop will give to you x.

$$pop(push(S, x)) = x$$

$push(S, x)$: a new stack with x on top

Lecture 5

Monday Sept. 25

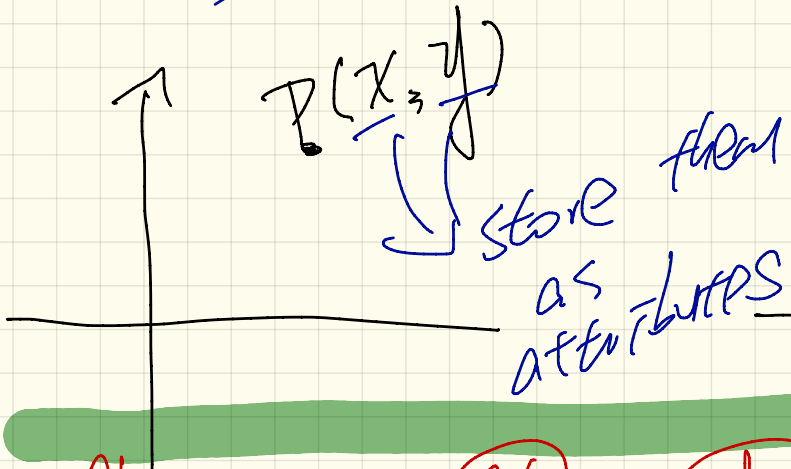

AC
assign_at(c,s)
ensure
P
T-A-C
inherit
BAD-AC-ASSIGN-AT
assign_at
ensure
P

2-D points (Supplier)

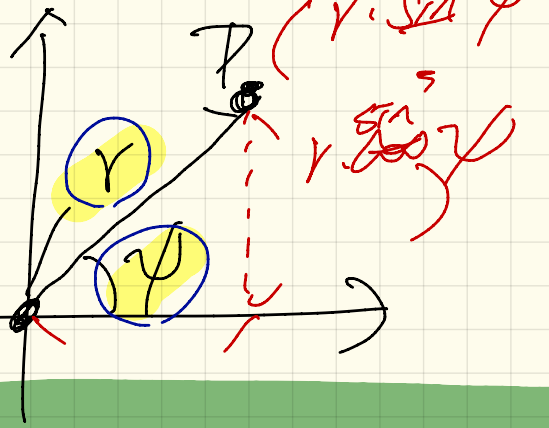
x: r * cos γ

y: r * sin γ

Cartesian



Polar



Client: P.x P.y

```
class Point {
private double r;
private double x;
private double phi;
private double y;

double getX() {
return x;
} }
```

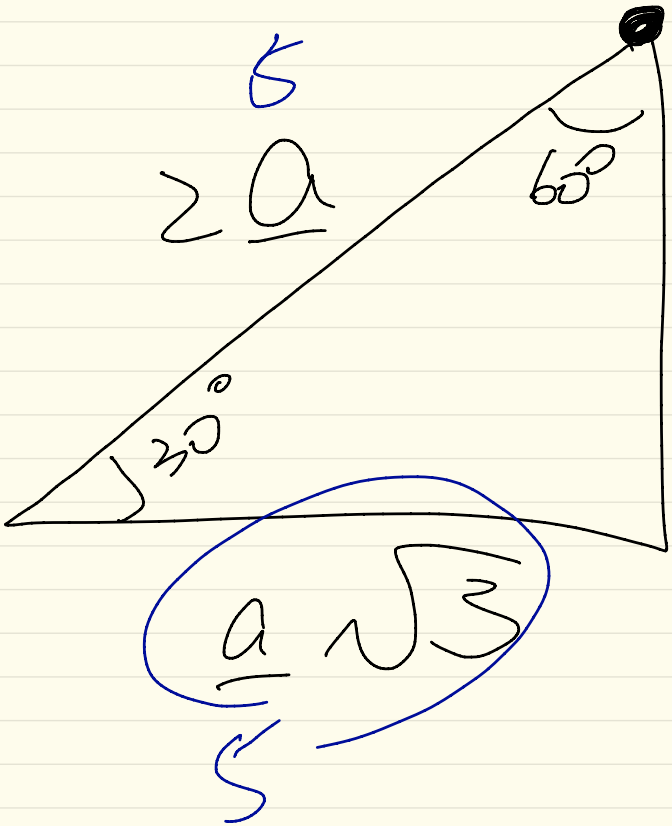
r * sin phi

change from Cart. to Pol.

uniform access

In Eiffel

```
class POINT
x: REAL
y: REAL
end
```



(2a · sin 30° ; 2a · cos 30°)

a ← Polar system

Cartesian system.
these values pass
right away: (5√3 = 5)

BANK

accounts: ARRAY[ACCOUNT]

balance : REAL

less frequent change, is frequent access

attribute

avoid computation e.g. large # of accounts.

query.

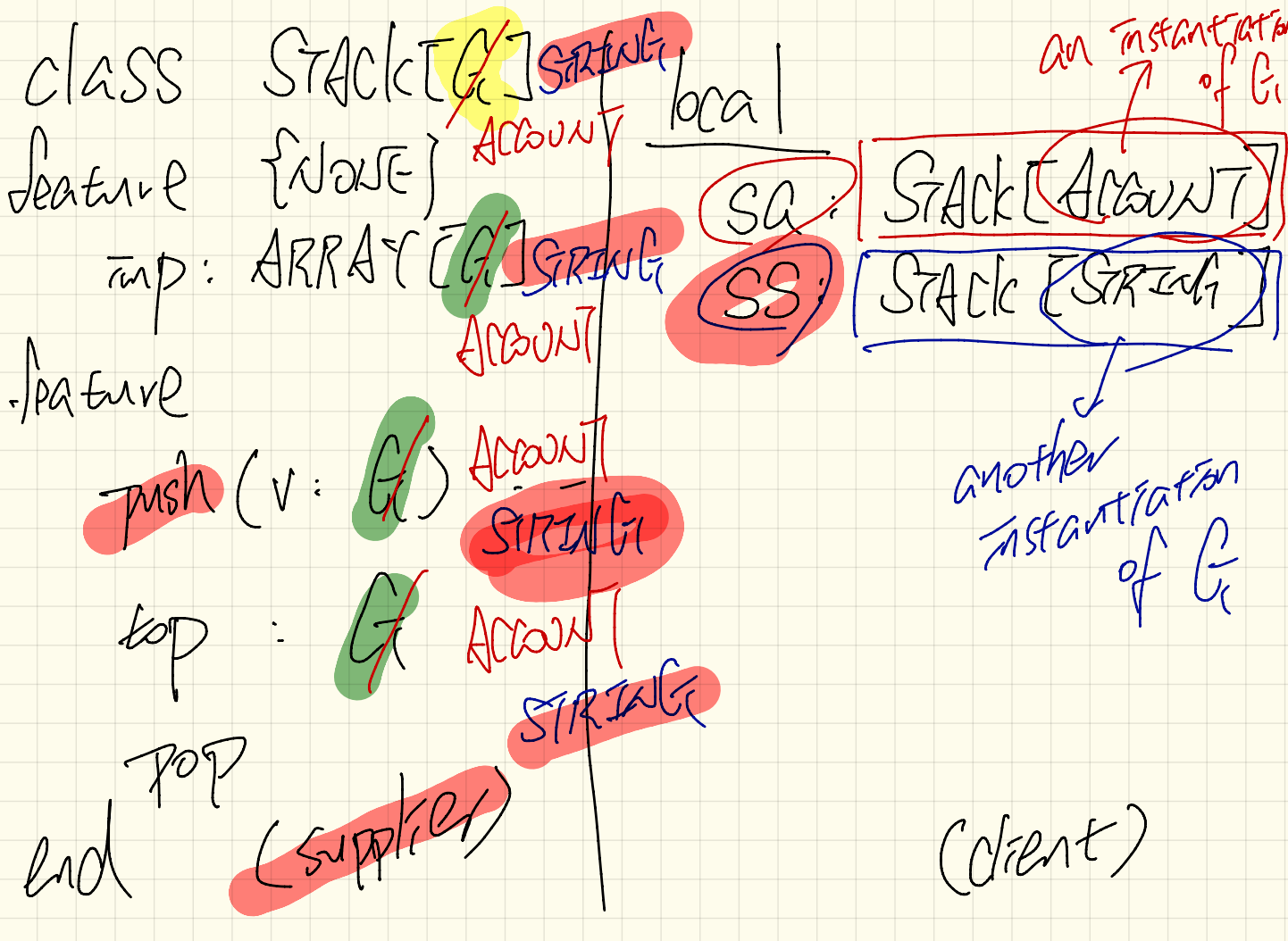
don't have to keep the value in sync for every feature; every access is a fresh comp.

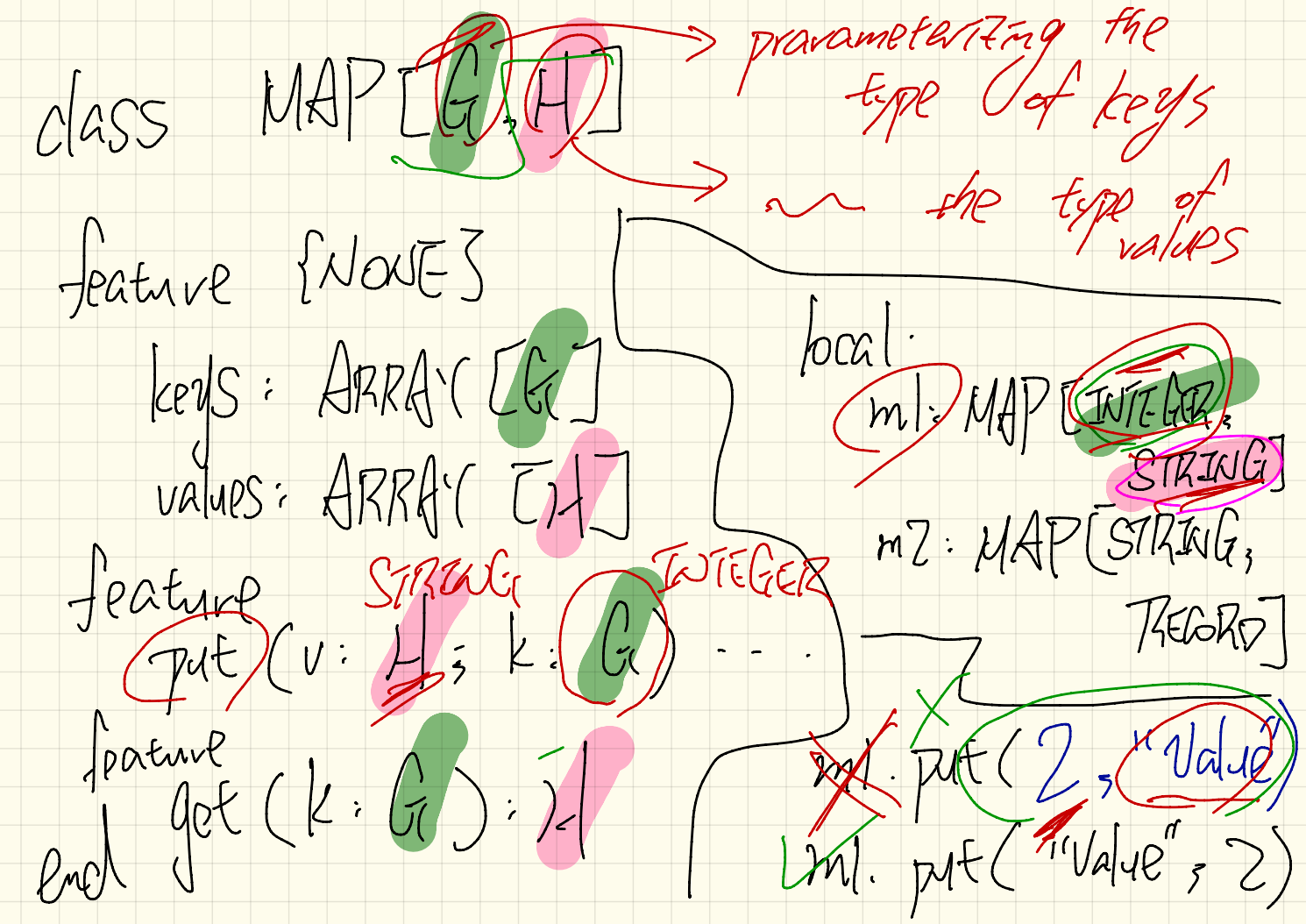
Lecture 6

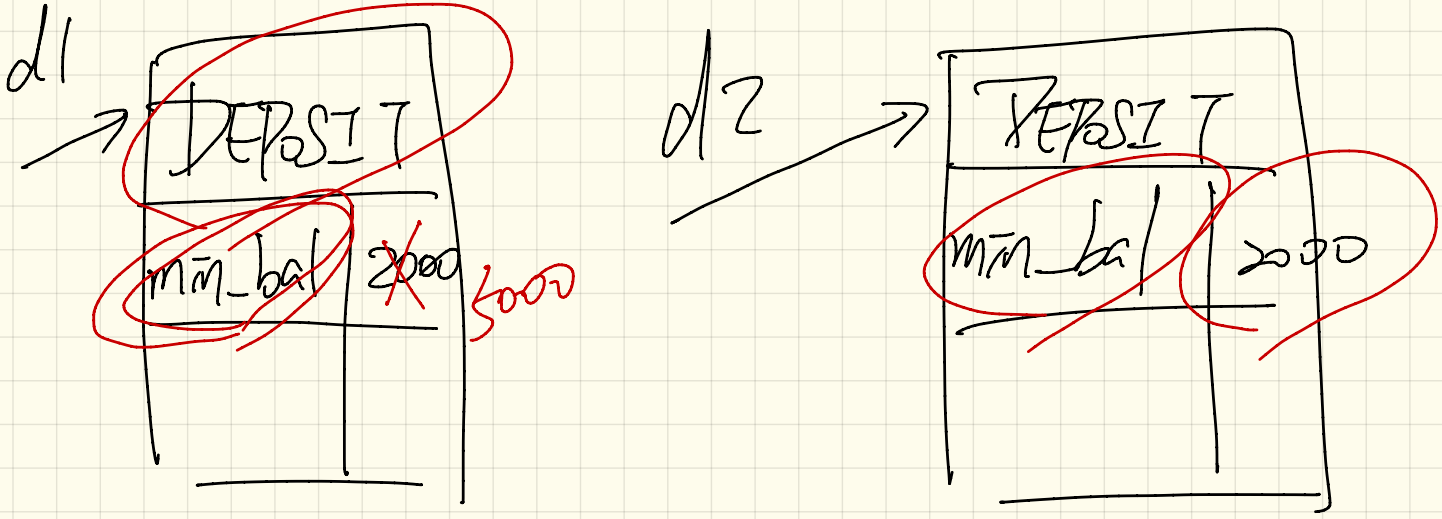
Wednesday Sept. 27

EECS3311_lab1_bon.pdf

.xml


class STACK[G]
feature {NONE}
imp: ARRAY[G]
feature
push (v: G)
top : G
pop
end
STRING
ACCOUNT
STRING
ACCOUNT
ACCOUNT
STRING
ACCOUNT
STRING
local
sa:
ss:
an instantiation of G
STACK[ACCOUNT]
STACK[STRING]
another instantiation of G
(supplier)
(client)


class MAP [G, H]
parameterizing the type of keys
the type of values
feature {NONE}
keys: ARRAY [G]
values: ARRAY [H]
feature
put (v: H; k: G) ...
STRING
INTEGER
feature
get (k: G): H
end
local
m1: MAP [INTEGER, STRING]
m2: MAP [STRING, INTEGER]
m1.put (2, "Value")
m1.put ("Value", 2)



Problem: When min_bal is changed, need to change more than one obj objects.

BON (Business Oriented Notation)

Two kinds of relations between classes:

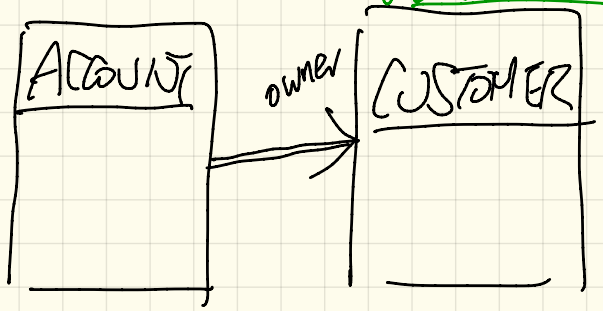
(1) client-supplier
(2) inheritance

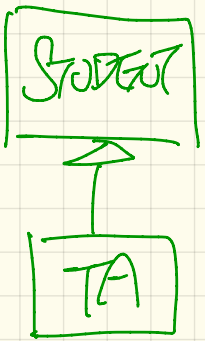
(2)
```
class STUDENT
end
```

```
class TA inherits
end      STUDENT
```

(1)
```
class ACCOUNT
    owner : CUSTOMER

end
```





m1: MAP [INT, STRING] (Student # → Student name)

m2: MAP [STRING, REAL] (Student name → grade)

m1.get(12345)

m2.get(...)

create m1.make_empty

create m2.make_empty

m1.put("jackie", 12345)

m2.put("b.s", "jackie")

Problem: given a student whose id is 12345, return their grade.

m2.get(m1.get(12345))

"jackie"

class ARRAYED_CONTAINER [G]

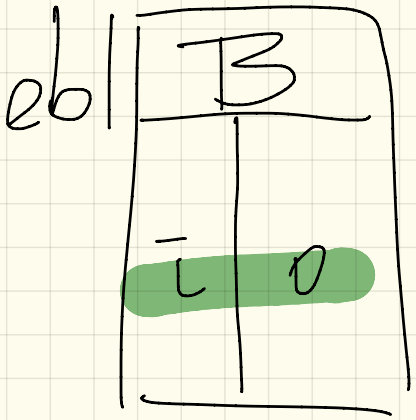
imp: ARRAY[G]

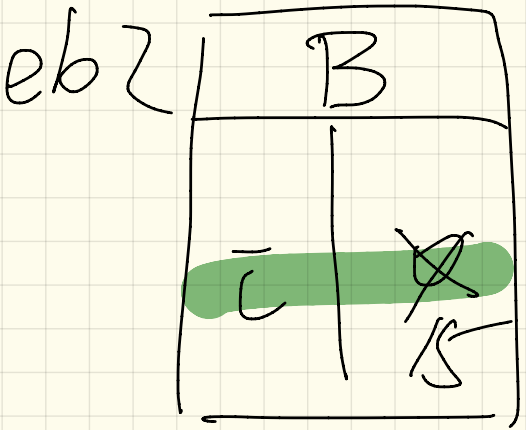
get_at (i: INT): G

insert_at (i: INT; s: G)

Lecture 7

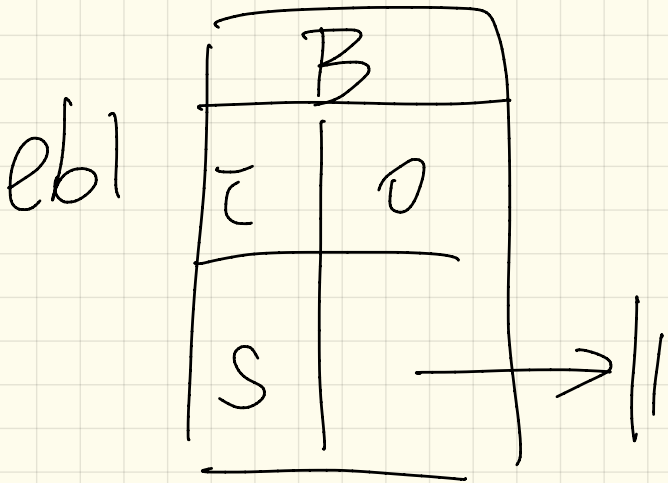
Monday Oct. 2





eb1 = eb2

eb2. change_c(15)

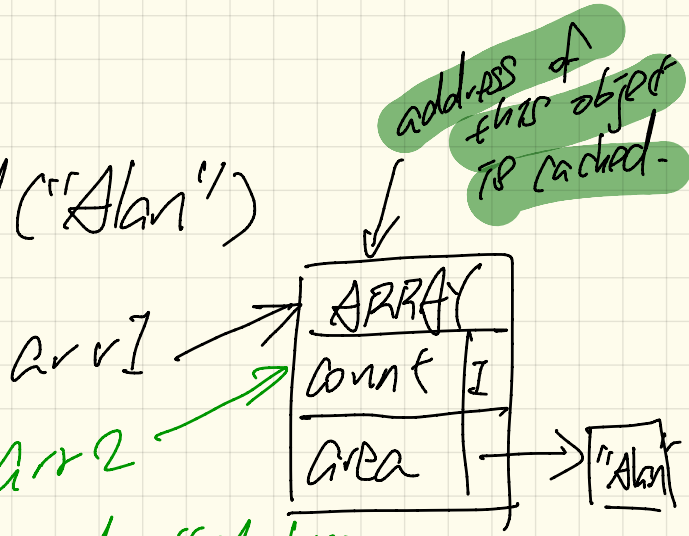


Single-choice principle

When there's a change, only one single place to be update.

Create a.make

arr1 := a.new_once_array("Alan")



arr2 := a.new_once_array("Mark")

class A

create {B}

make

→ only class B can create an instance of A.

feature {B}

make

→ only class B can call make.

Lecture 8

Wednesday Oct. 4

Lab 1 marks

- Log in to red.eecs.yorku.ca
- feedback 331 lab1
- Issues? Thursday 13:30 ~ 15:30

Lab test # 1

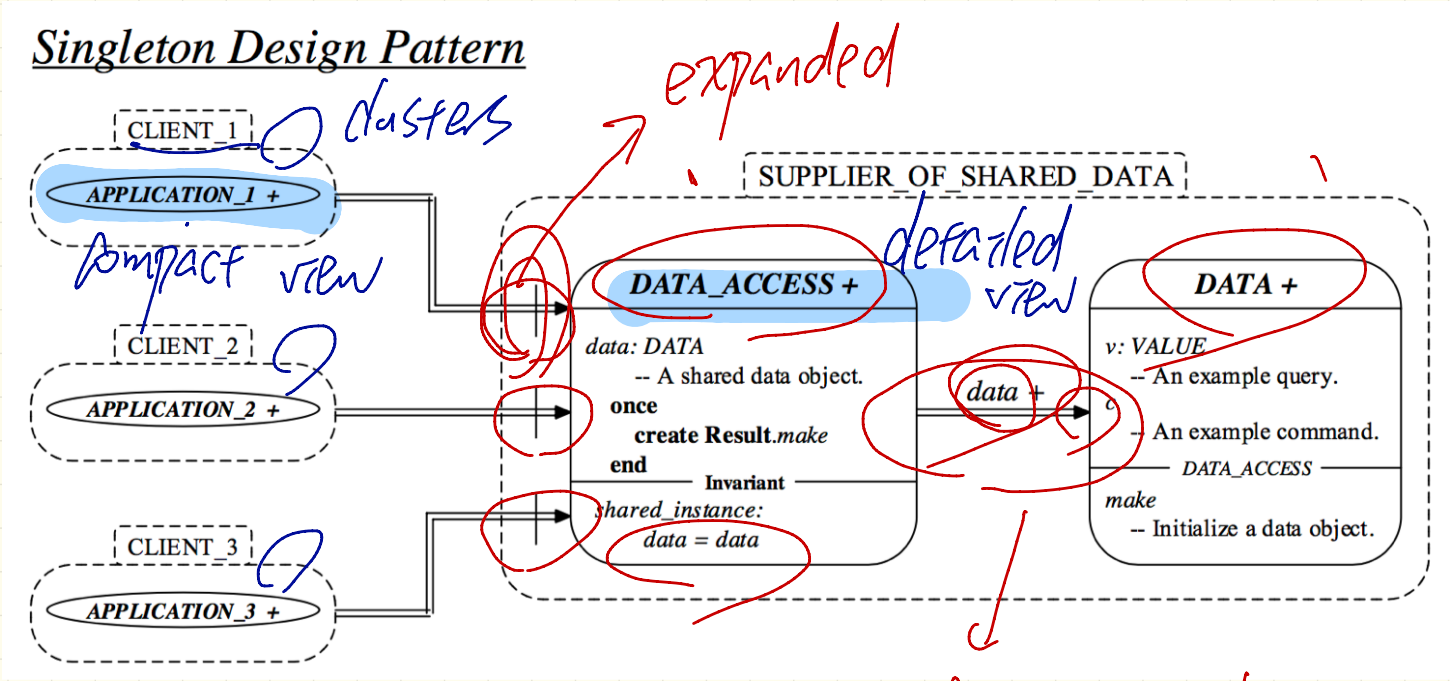
Wednesday Oct. 11

Prepare

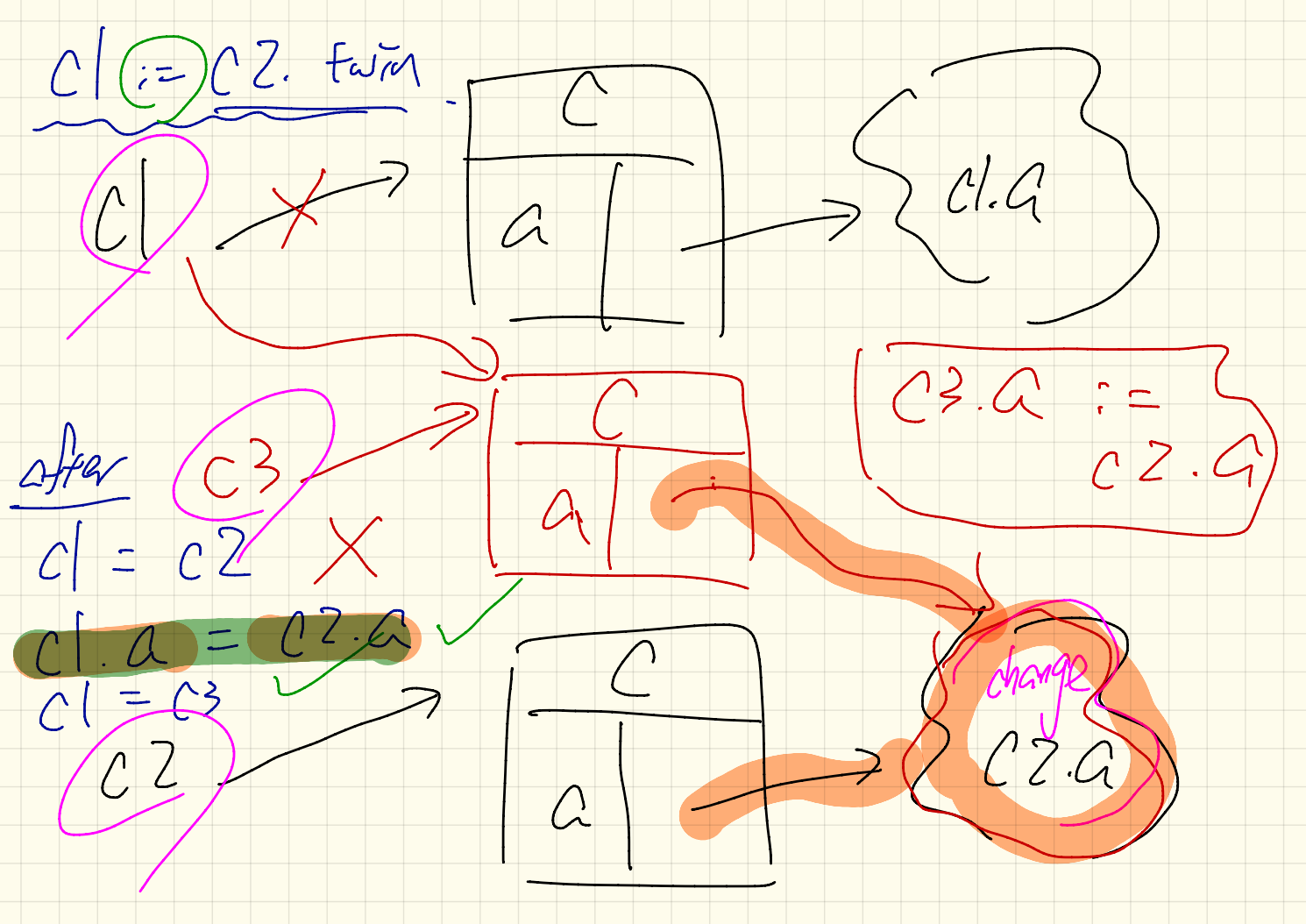
1. slides
2. tutorial videos
3. sample codes
4. Lab 1

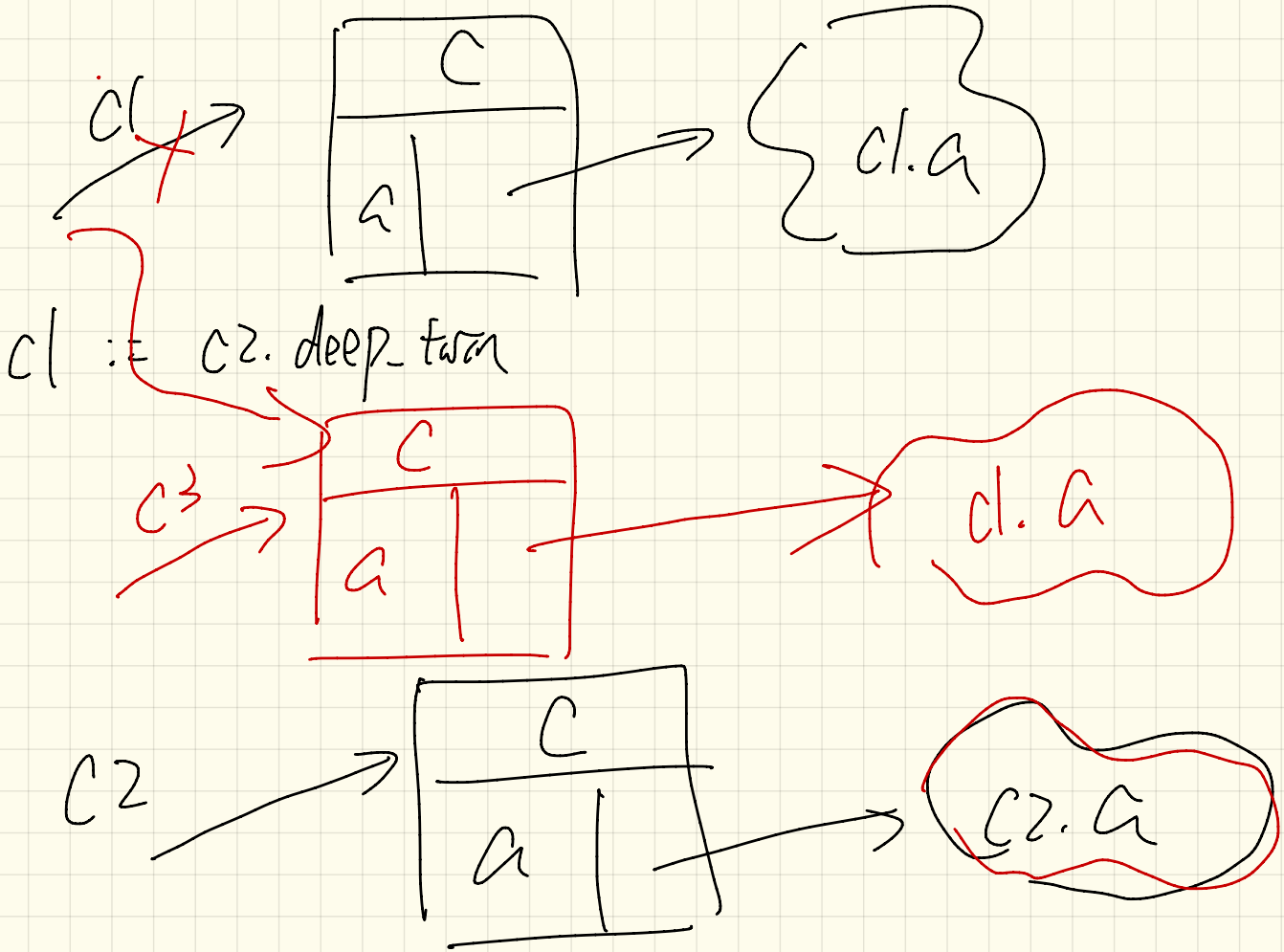
Format

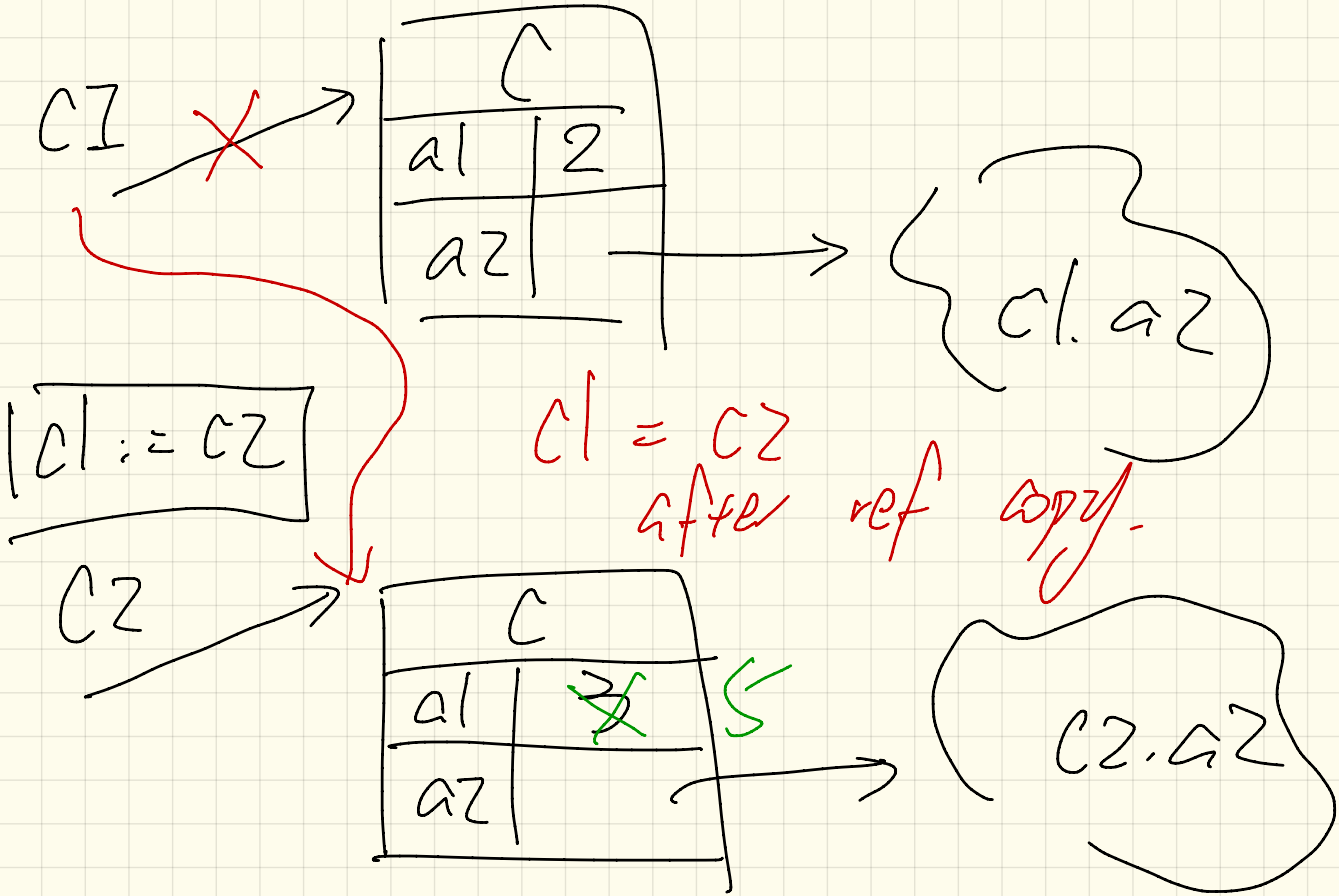
1. EStudio
2. Written

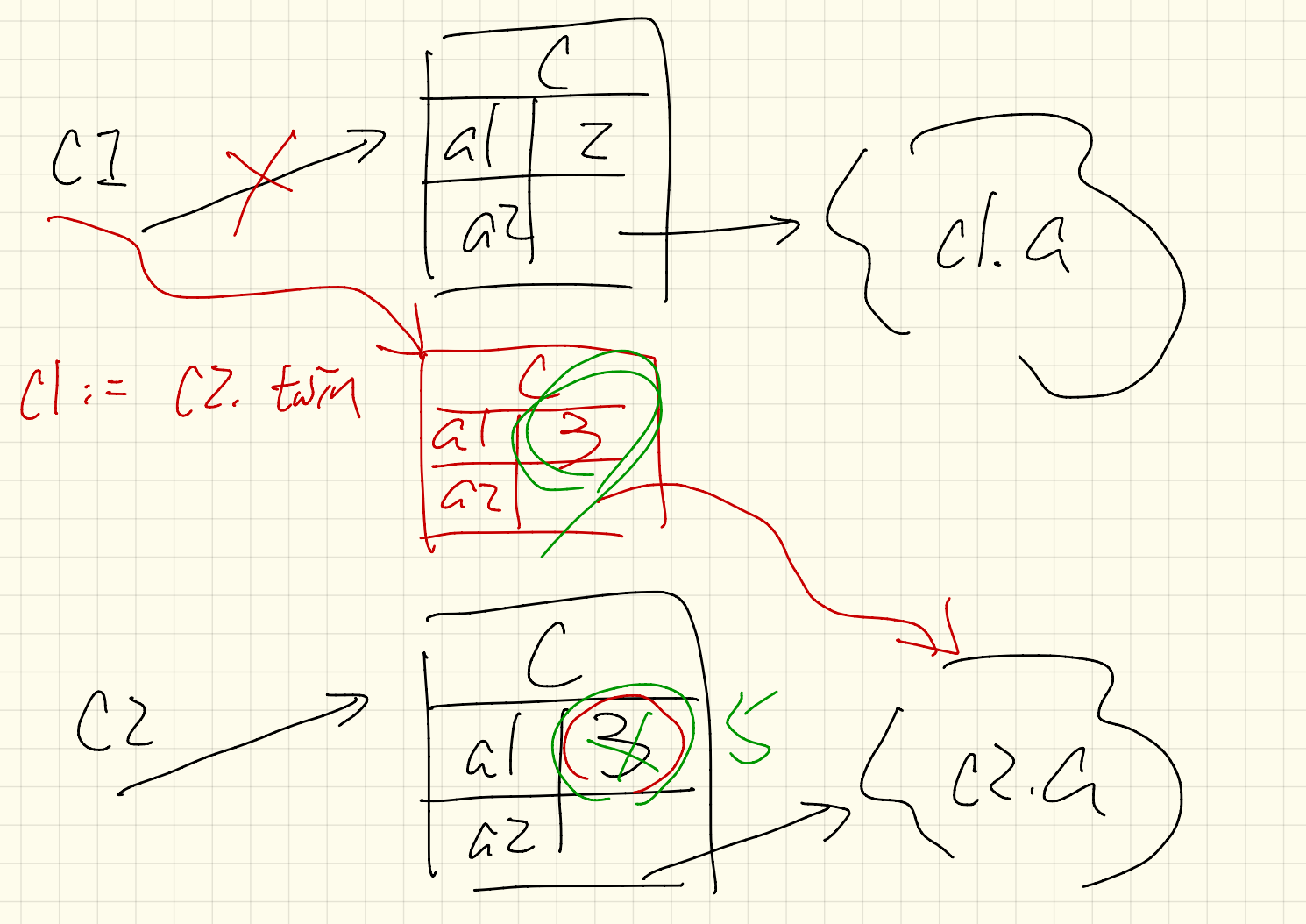


BON diagram SS US client-supplier draw.io

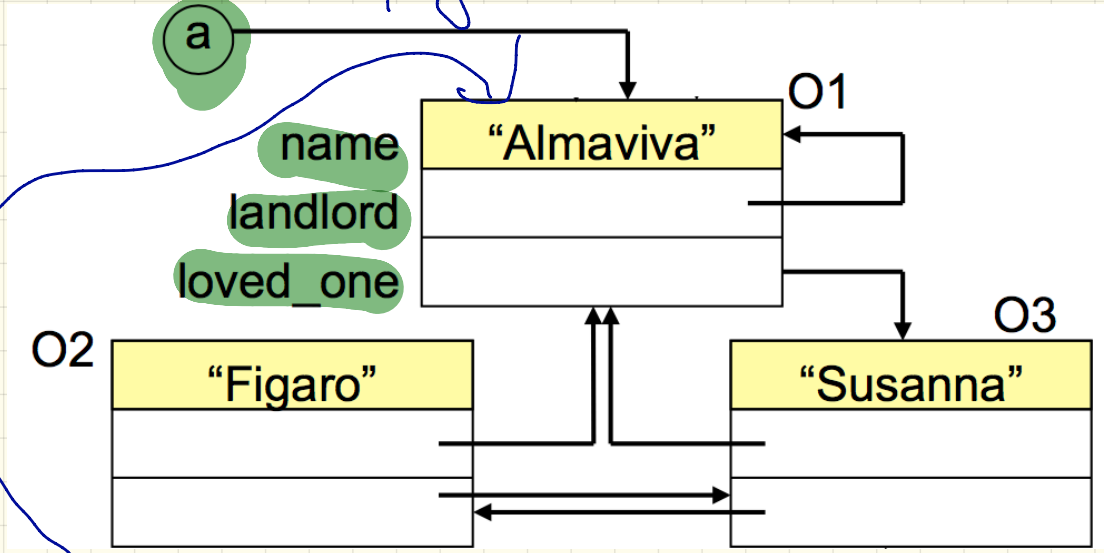

c1 := c2. twin
c1
C
a
c1.a
after
c3
c1 = c2
c3.a :=
c2.a
c1.a = c2.a
c1 = c3
c2
C
a
C
a
change
c2.a


c1
C
a
c1.a
c1 := c2.deep_twin
c3
C
a
c1.a
c2
C
a
c2.a

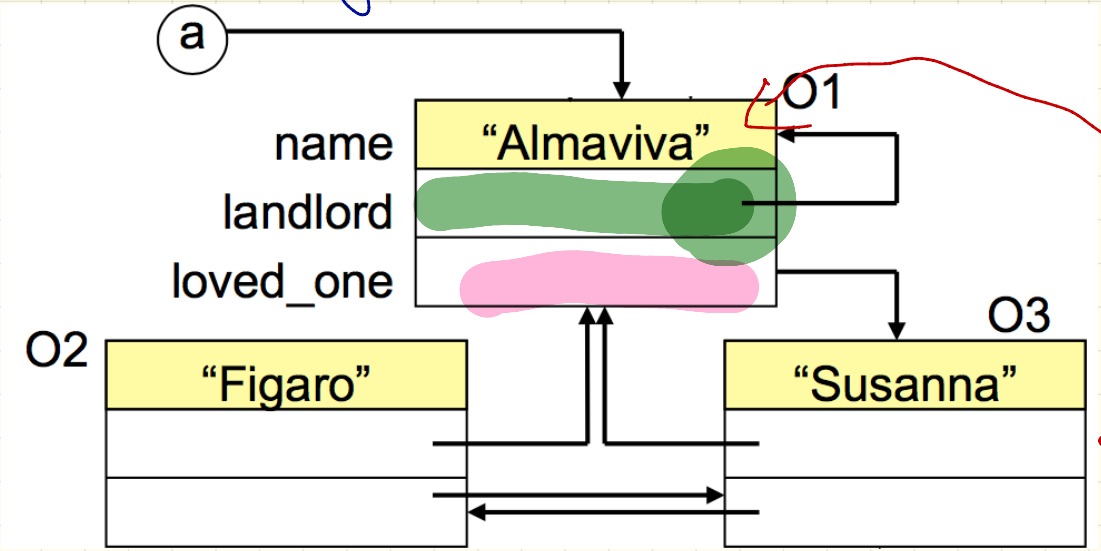

C1
C
a1
2
a2
C1.a2
c1 := c2
C1 = C2
after ref copy.
C2
C
a1
2
5
a2
C2.a2


C1
C
a1
z
a2
c1.a
c1 := c2.twin
C
a1
3
a2
C2
C
a1
3
5
a2
c2.a

Reference Copy: b := a



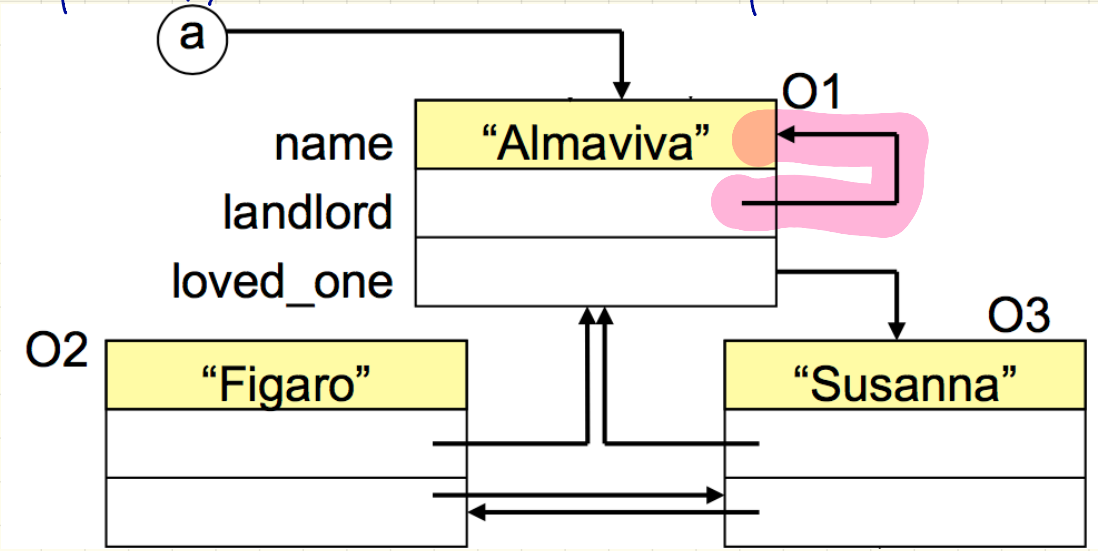
Shallow Copy: c := a.twin

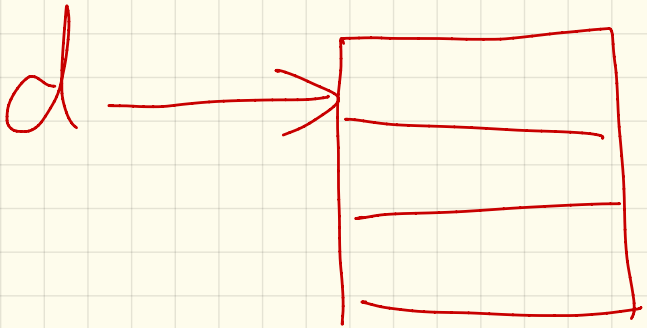


c

"Almaviva"

Deep Copy: d := a.deep_twin





s1, s2 : STRING

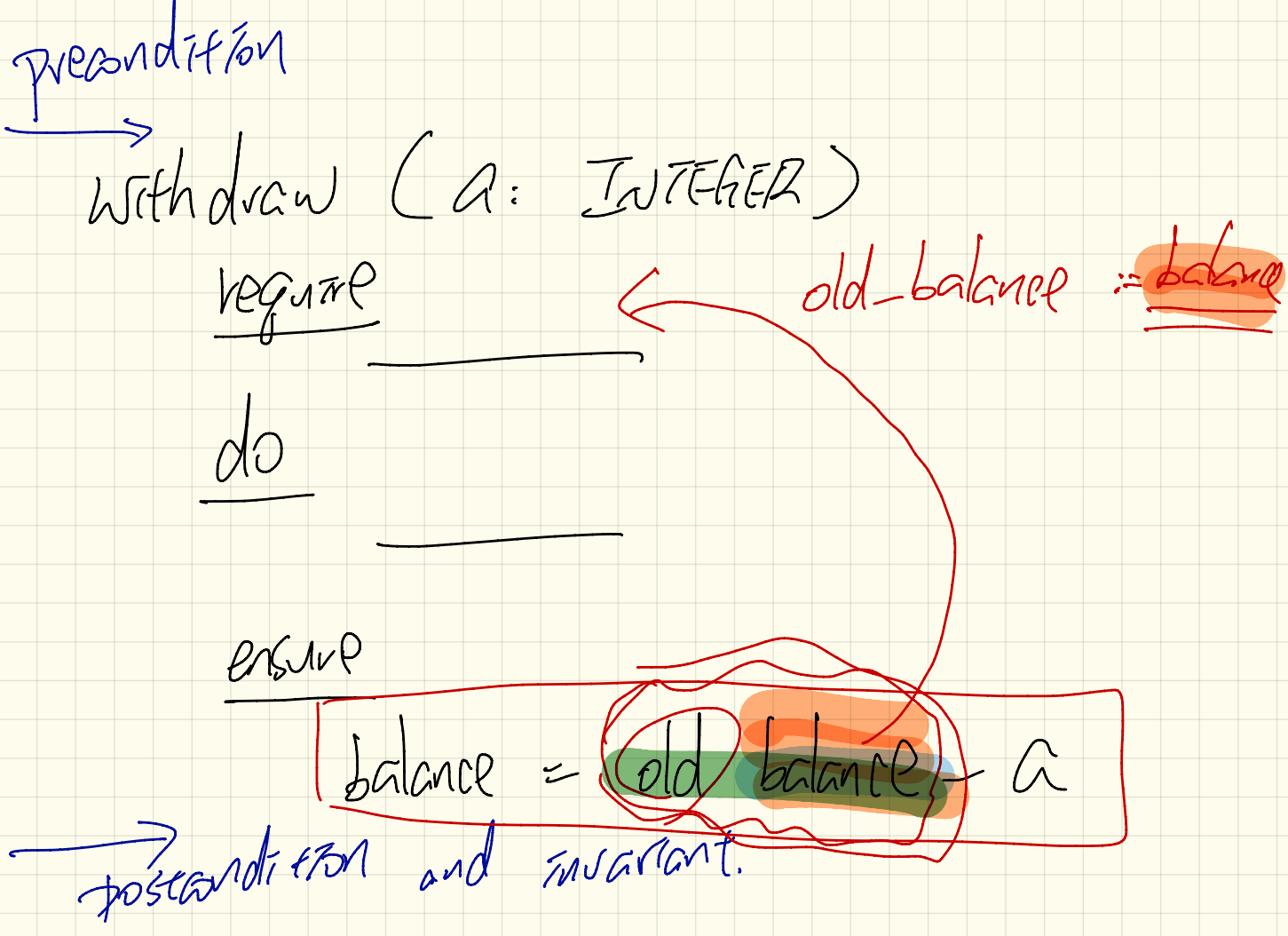
s1 := "York"

s2 := "York"

s1 = s2

create s1.make_from_string ("York")

create s2.make_from_string ("York")


precondition
withdraw (a: INTEGER)
require
do
ensure
old_balance := balance
balance = old balance - a
postcondition and invariant.

insert_at (——————>

require

do

old_imp := imp

old_imp_twin := imp.twin

old_imp_deep_twin := imp.deep_twin

ensure

——————

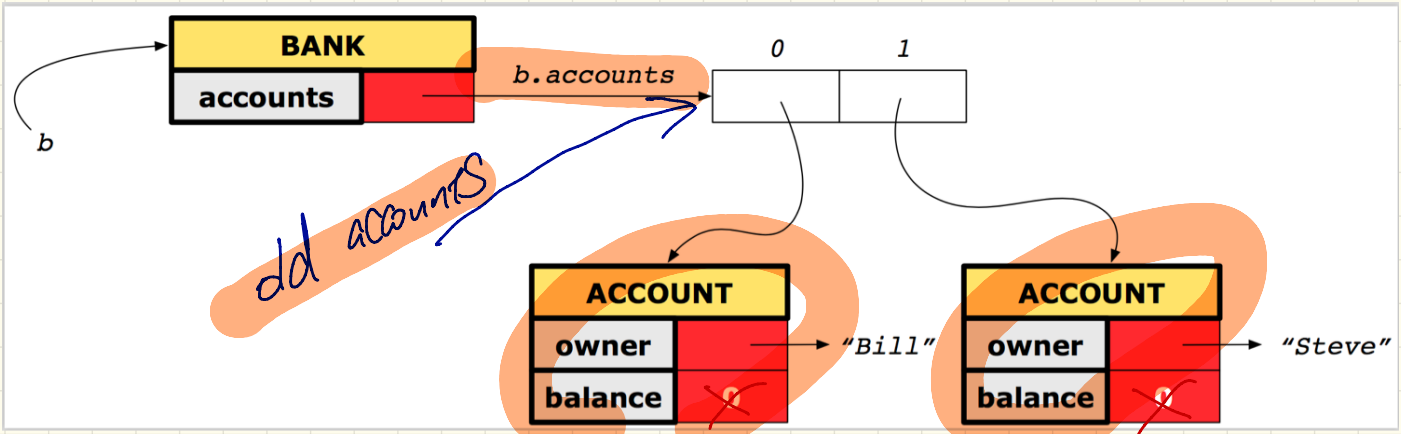
——————

end ——————

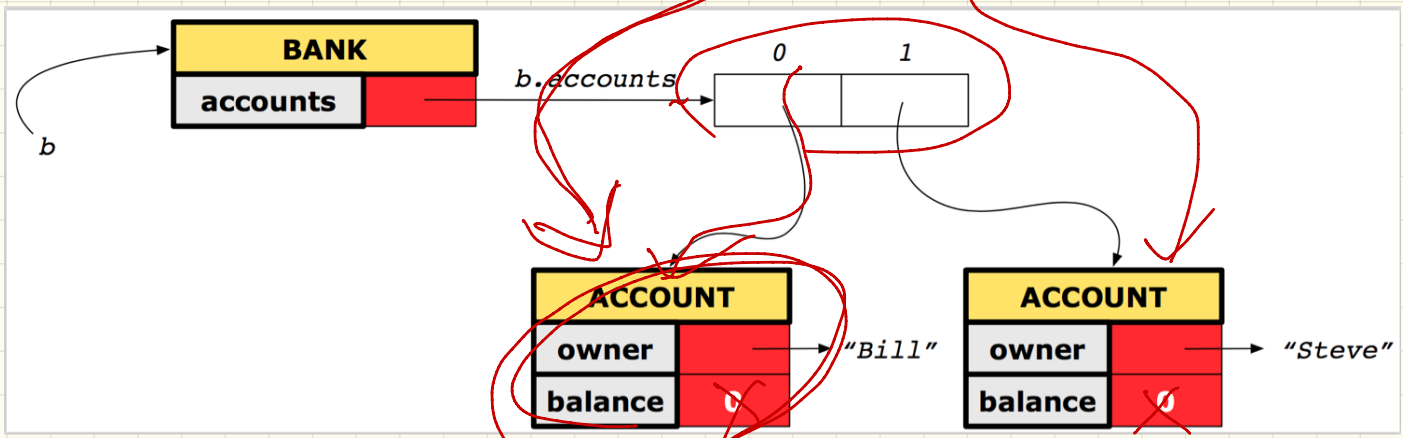
old temp

old imp.twin

old imp.deep_twin



ensure across old accounts



accounts.txfr

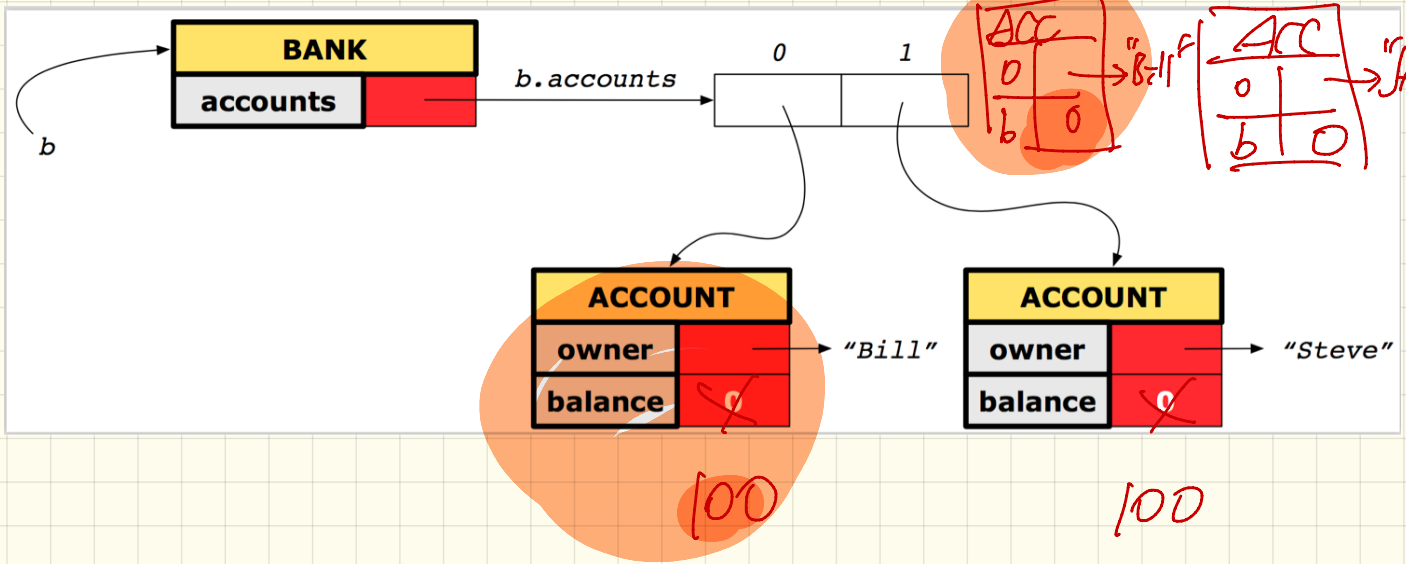
100

100

ensure

across accounts.txfr

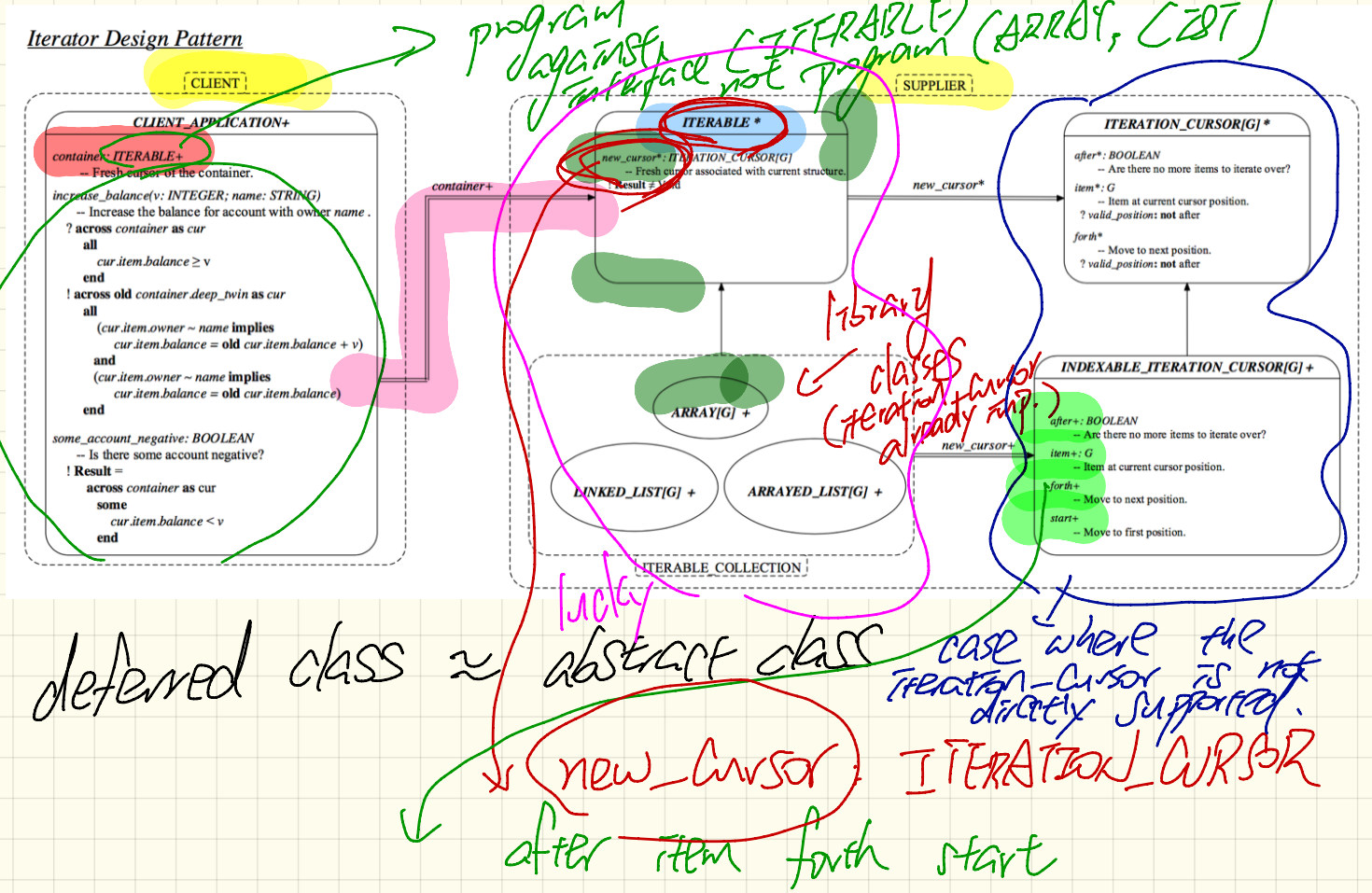
accounts.deep_twin
dd



ensure across accounts.deep_twin

Lecture 9

Wednesday Oct. 11

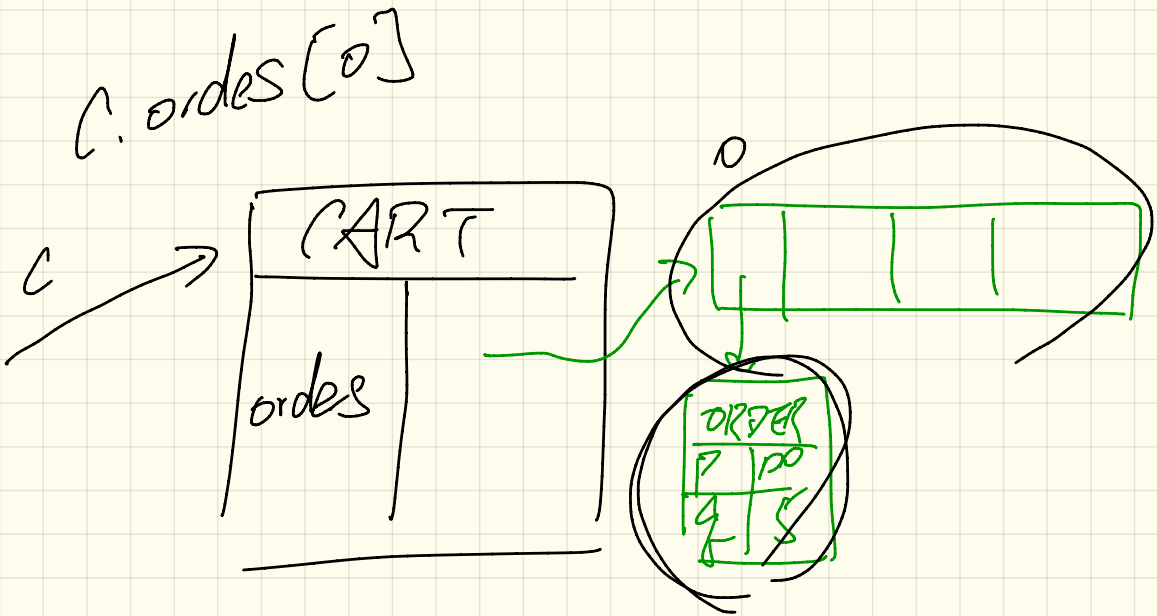

Iterator Design Pattern
CLIENT
CLIENT_APPLICATION+
container: ITERABLE+
-- Fresh cursor of the container.
increase_balance(v: INTEGER; name: STRING)
-- Increase the balance for account with owner name .
? across container as cur
all
cur.item.balance ≥ v
end
! across old container.deep_twin as cur
all
(cur.item.owner ~ name implies
cur.item.balance = old cur.item.balance + v)
and
(cur.item.owner ~ name implies
cur.item.balance = old cur.item.balance)
end
some_account_negative: BOOLEAN
-- Is there some account negative?
! Result =
across container as cur
some
cur.item.balance < v
end
container+
SUPPLIER
ITERABLE *
new_cursor*: ITERATION_CURSOR[G]
-- Fresh cursor associated with current structure.
Result ≠ Void
new_cursor*
ITERATION_CURSOR[G] *
after*: BOOLEAN
-- Are there no more items to iterate over?
item*: G
-- Item at current cursor position.
? valid_position: not after
forth*
-- Move to next position.
? valid_position: not after
ARRAY[G] +
LINKED_LIST[G] +
ARRAYED_LIST[G] +
INDEXABLE_ITERATION_CURSOR[G] +
new_cursor+
after+: BOOLEAN
-- Are there no more items to iterate over?
item+: G
-- Item at current cursor position.
forth+
-- Move to next position.
start+
-- Move to first position.
ITERABLE_COLLECTION
program against interface not program (ARRAY, CHAT)
library classes (iteration cursor already imp.)
lucky
deferred class ≈ abstract class
case where the iteration-cursor is not directly supported.
new_cursor: ITERATION_CURSOR
after item forth start

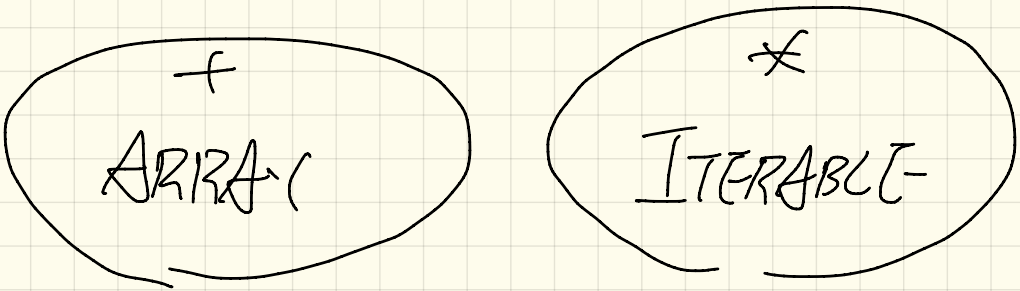
CLASS CART inherit ITERABLE [ORDER]

orders: ARRAY [ORDER]

c.orders[0]

end

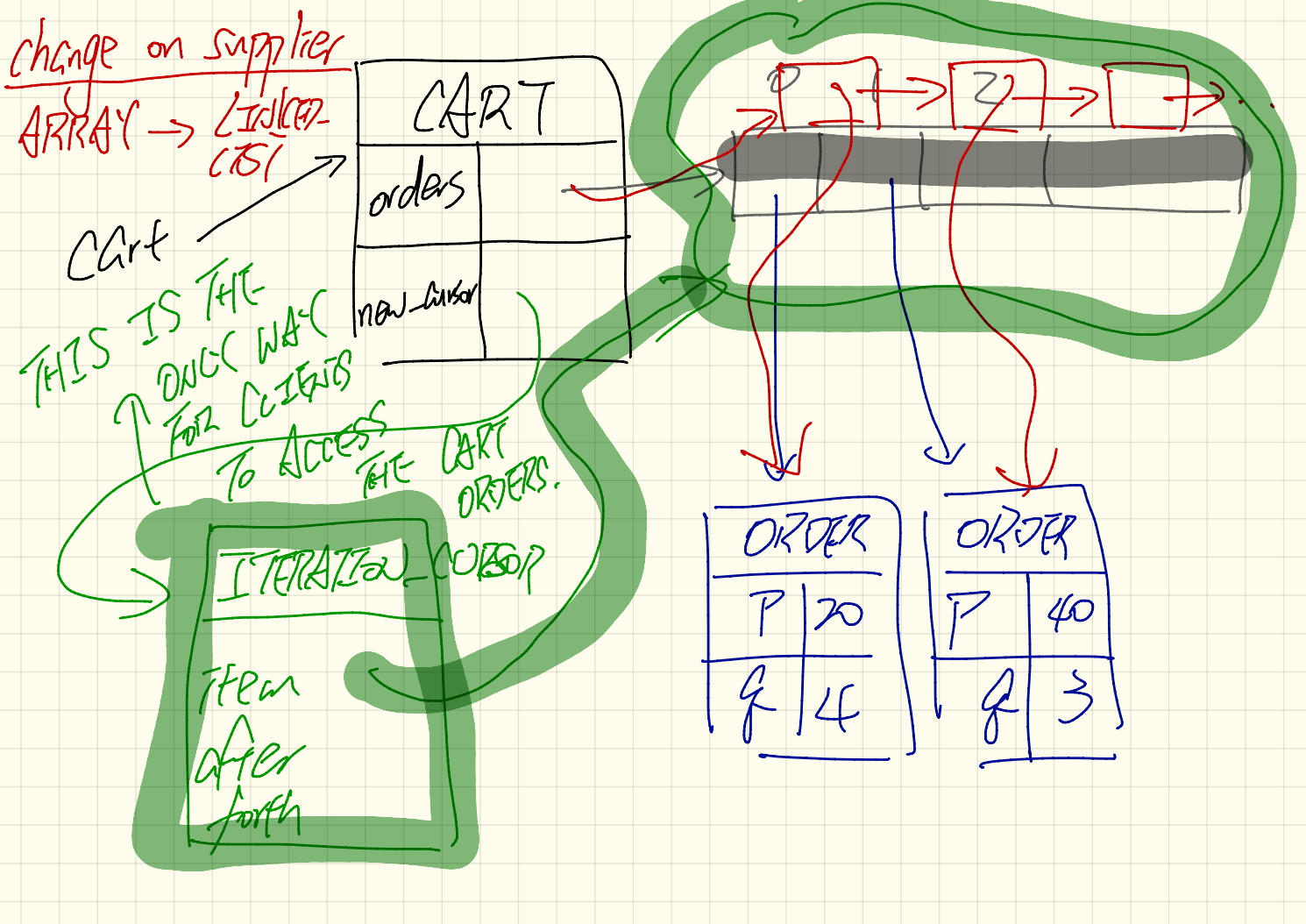


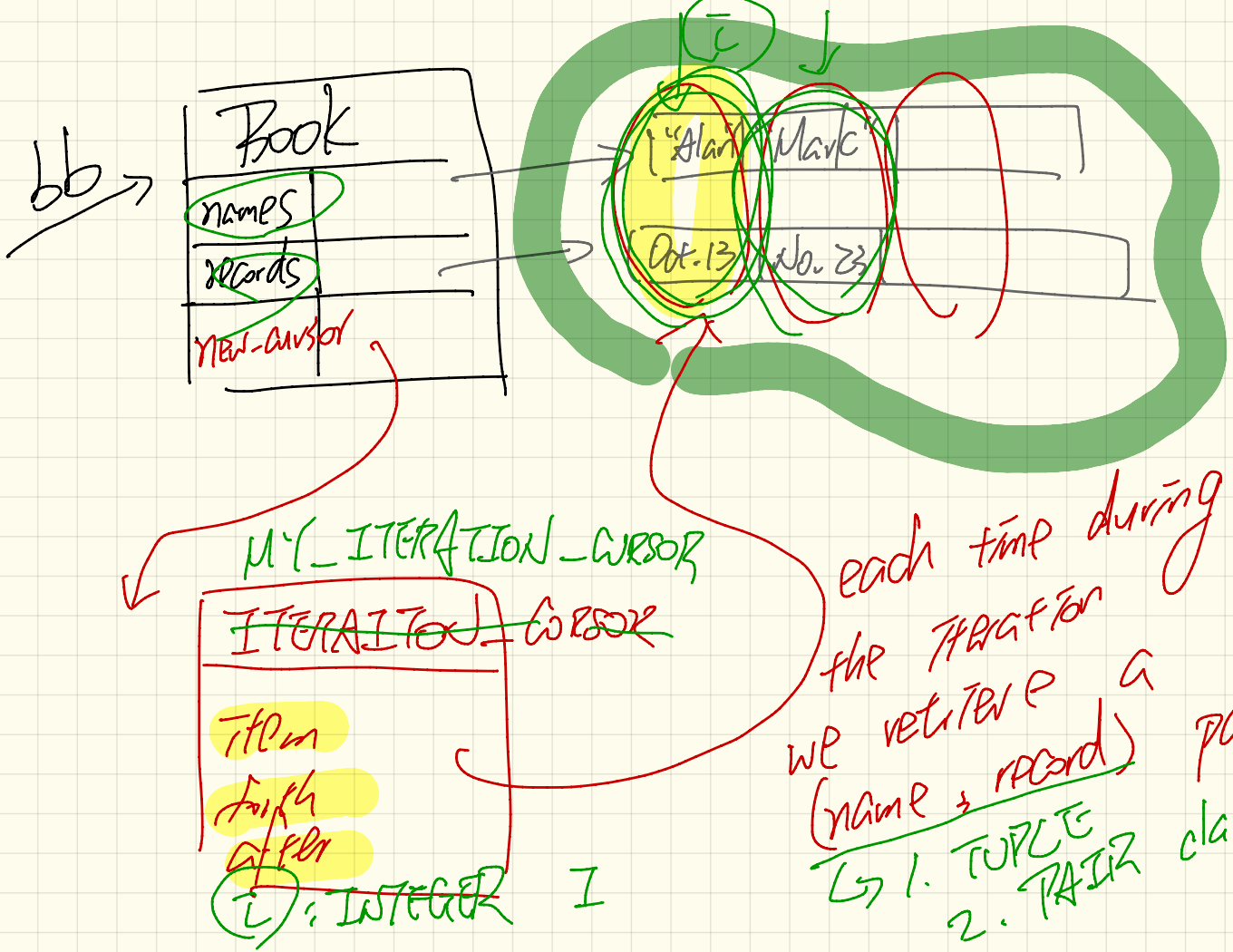


+ : effective — concrete/non-abstract.

* : deferred — abstract

implementation secrets (should be hidden)
- data structure (a, l)
- algorithm.


change on supplier
ARRAY → LINKED-LIST
CART
orders
new_cursor
cart
THIS IS THE ONLY WAY FOR CLIENTS TO ACCESS THE CART ORDERS.
ITERATOR_CURSOR
item after forth
0
1
2
ORDER
P 20
Q 4
ORDER
P 40
Q 3

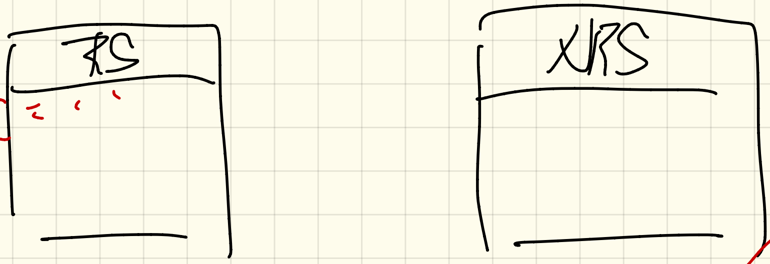

bb
Book
names
records
new_cursor
"Alan"
"Mark"
Oct. 13
No. 23
MY_ITERATION_CURSOR
ITERATION_CURSOR
from
forth
after
i : INTEGER
I
each time during the iteration we retrieve a (name, record) pair
1. TUPLE
2. PAIR class

Lecture 10

Monday Oct. 16

Without the Inheritance

RS s1 = ...
NRS s2 = ...
Account acc = ...

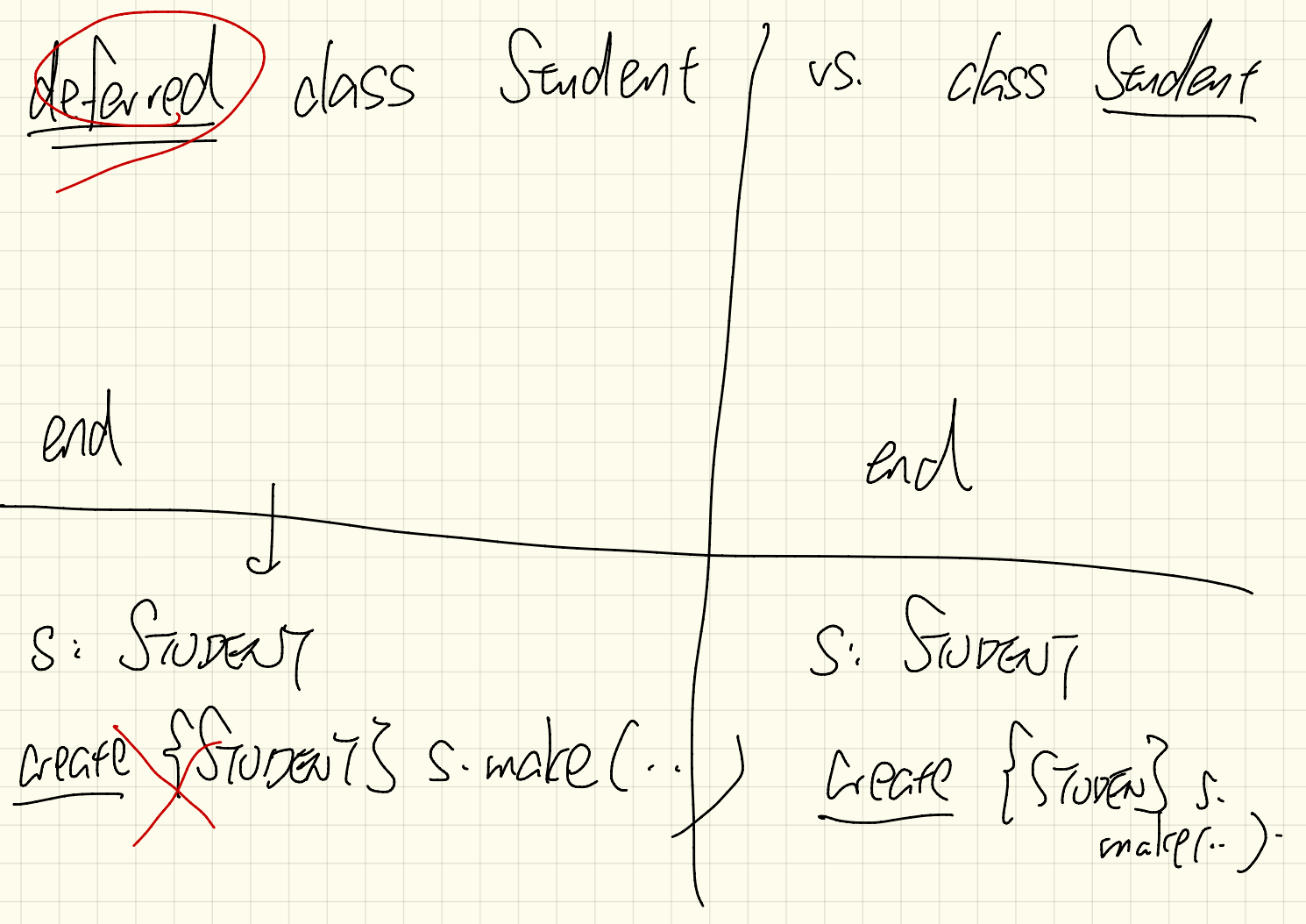


students.extend(s1 / s2 / acc)
ANY

Object[]
ARRAY[ANY]

```
class S_M_S
  students : LINKED_LIST[ANY]

end
```


deferred class Student
vs.
class Student
end
end
s: STUDENT
s: STUDENT
create {STUDENT} s.make(..)
create {STUDEN} s.
make(..)

S: STUDENT
RS: RS
NRS: NRS

S → STUDENT | name | "S" | courses

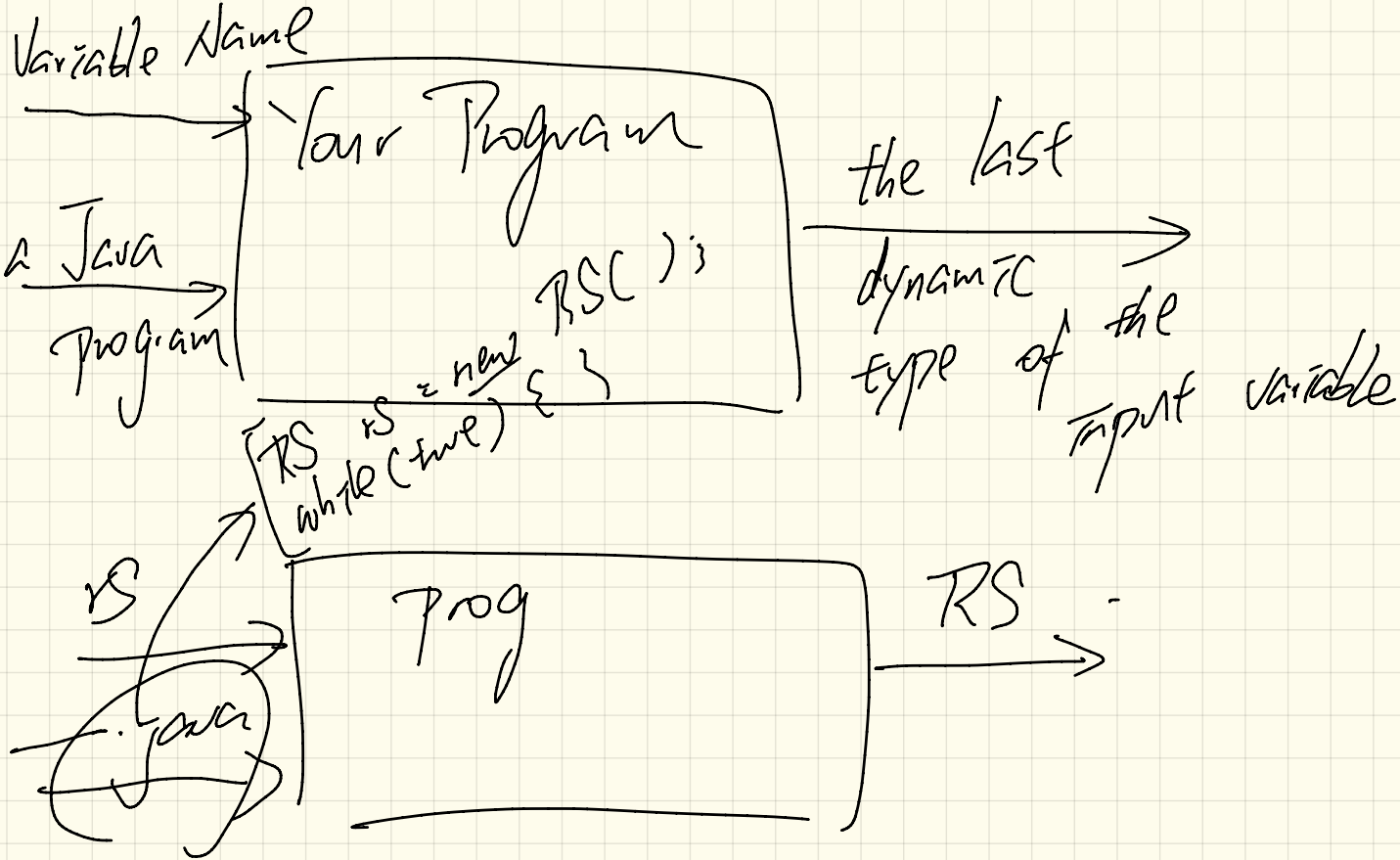
create {STUDENT} S. make("S")

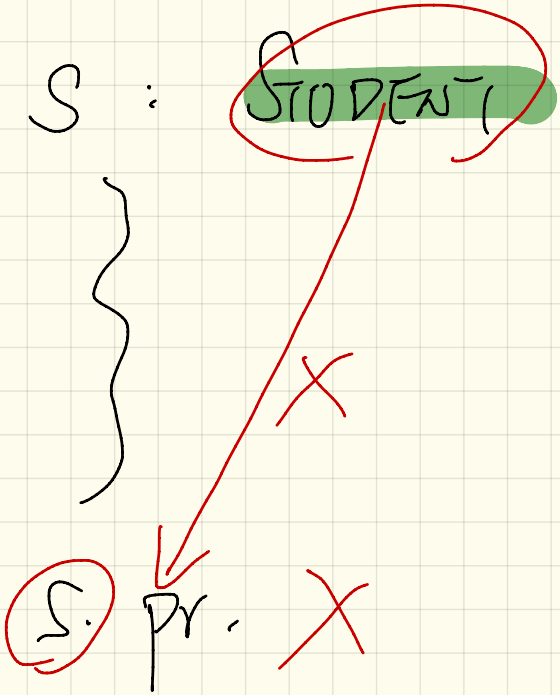
create {RS} RS. make("RS") RS → RS | name | courses | sv

inherited from STUDENT

create {NRS} NRS. make("NRS")

NRS → NRS | name | CS | dv


Variable Name
Your Program
a Java program
RS();
RS rs = new RS() { }
while (true)
the last dynamic type of the input variable
RS
Prog
RS
.java


S :
STUDENT
X
S. Pr. X

s: STUDENT

rs: RS

[creations

rs.set_pr (1.25)

s := rs

rs := s

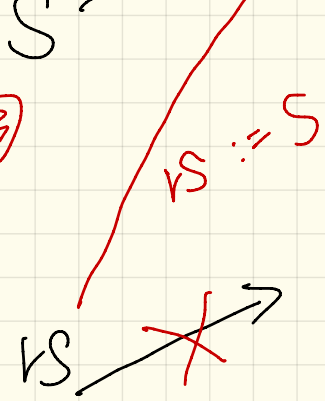
① assume this is ok.

② What features can call on rs?

rs.name

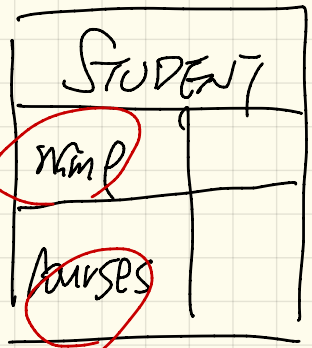
rs.pr

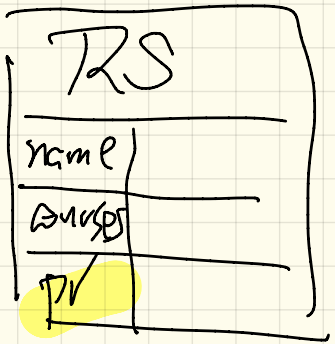
③



④ rs pointing to STUDENT with no pr

:= rs.pr crashes!!



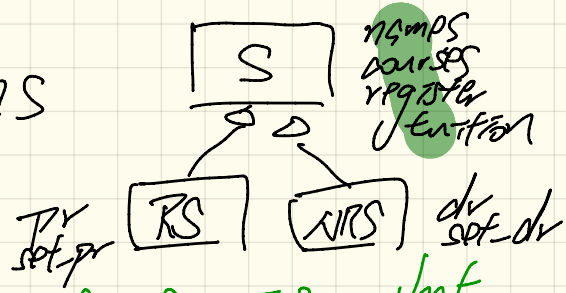


Lecture 11

Wednesday Oct. 18

s : STUDENT | declarations

rs : RS



A variable's declared type constrains what features you can call upon it.

What can we call on S?

rs.pr ✓ rs.dr ✗

S.pr ✗ S.dr ✗

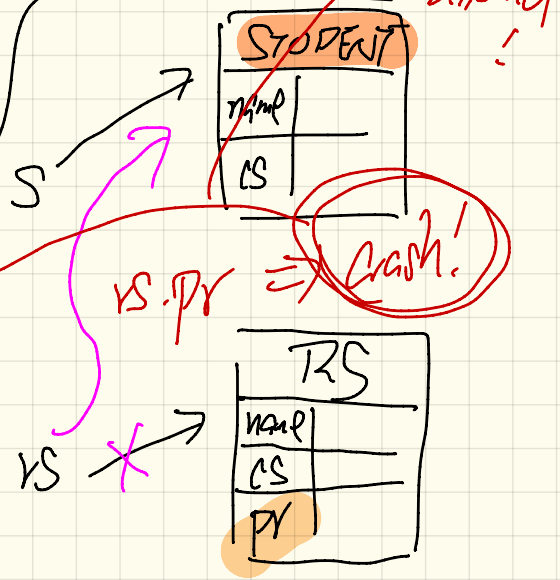
Create {STUDENT} s.make("s")

Create {RS} rs.make("rs")

rs := s ✗ not compile

① if rs := s was allowed...

② We know that rs.pr should be possible



S: STUDENT
rs: RS
nrs: NRS
c: COURSE

declarations

---

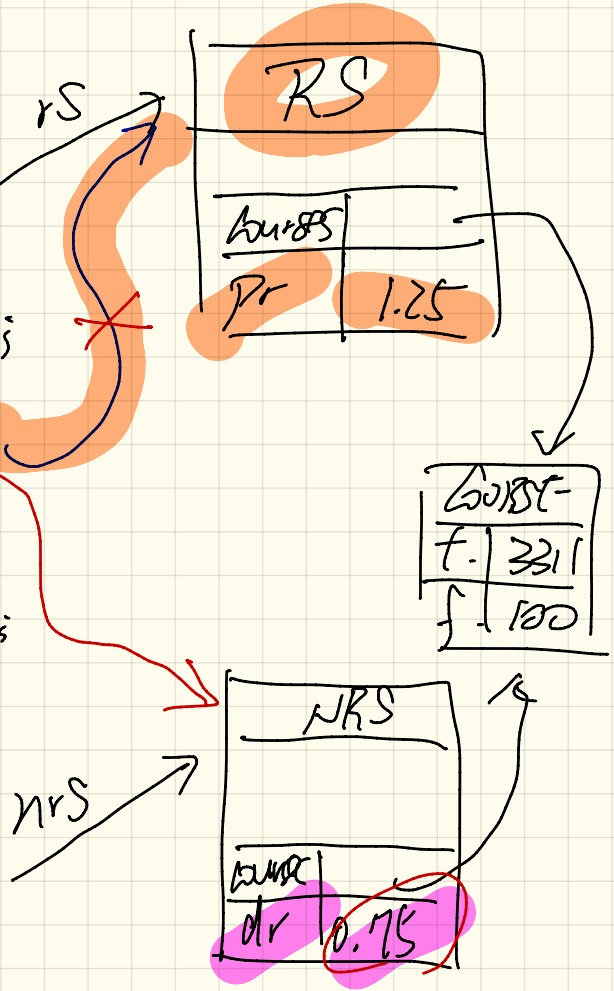
create c. make("3311", 100);
create {RS} rs. make(...);
rs. set_pr(1.25)
rs. register(c)

---

create {NRS} nrs. make(..);
nrs. set_dr(0.75)
nrs. register(c)

S := rs
S. tuition ?

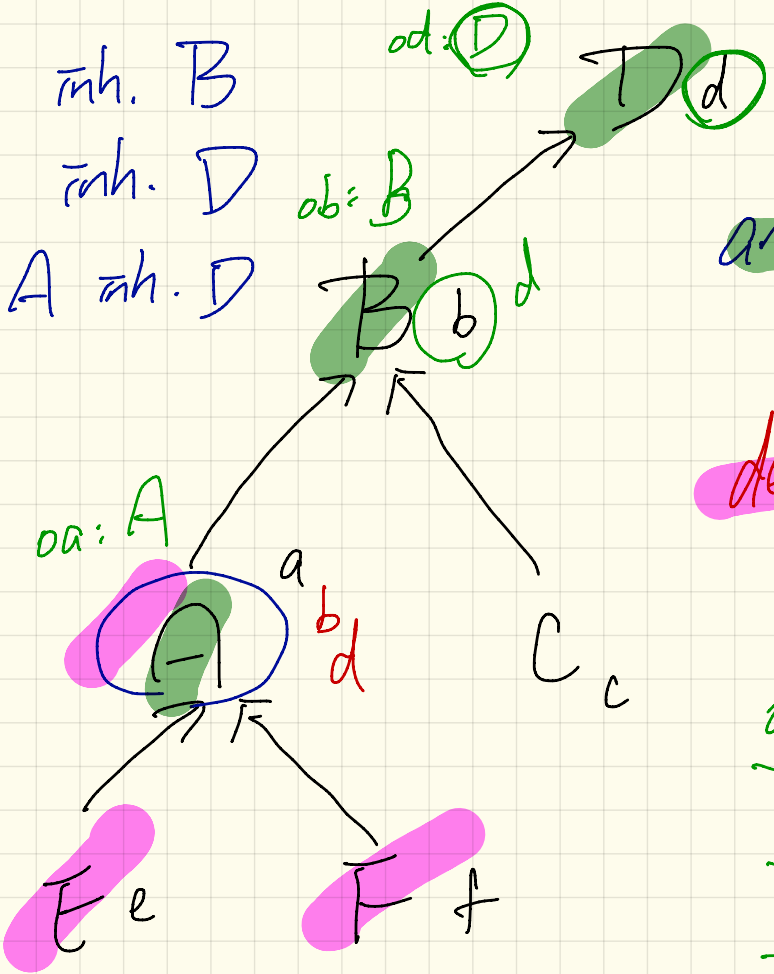
S := nrs
S. tuition ?



A inh. B

B inh. D

$\Rightarrow$ A inh. D



ancestors of A:

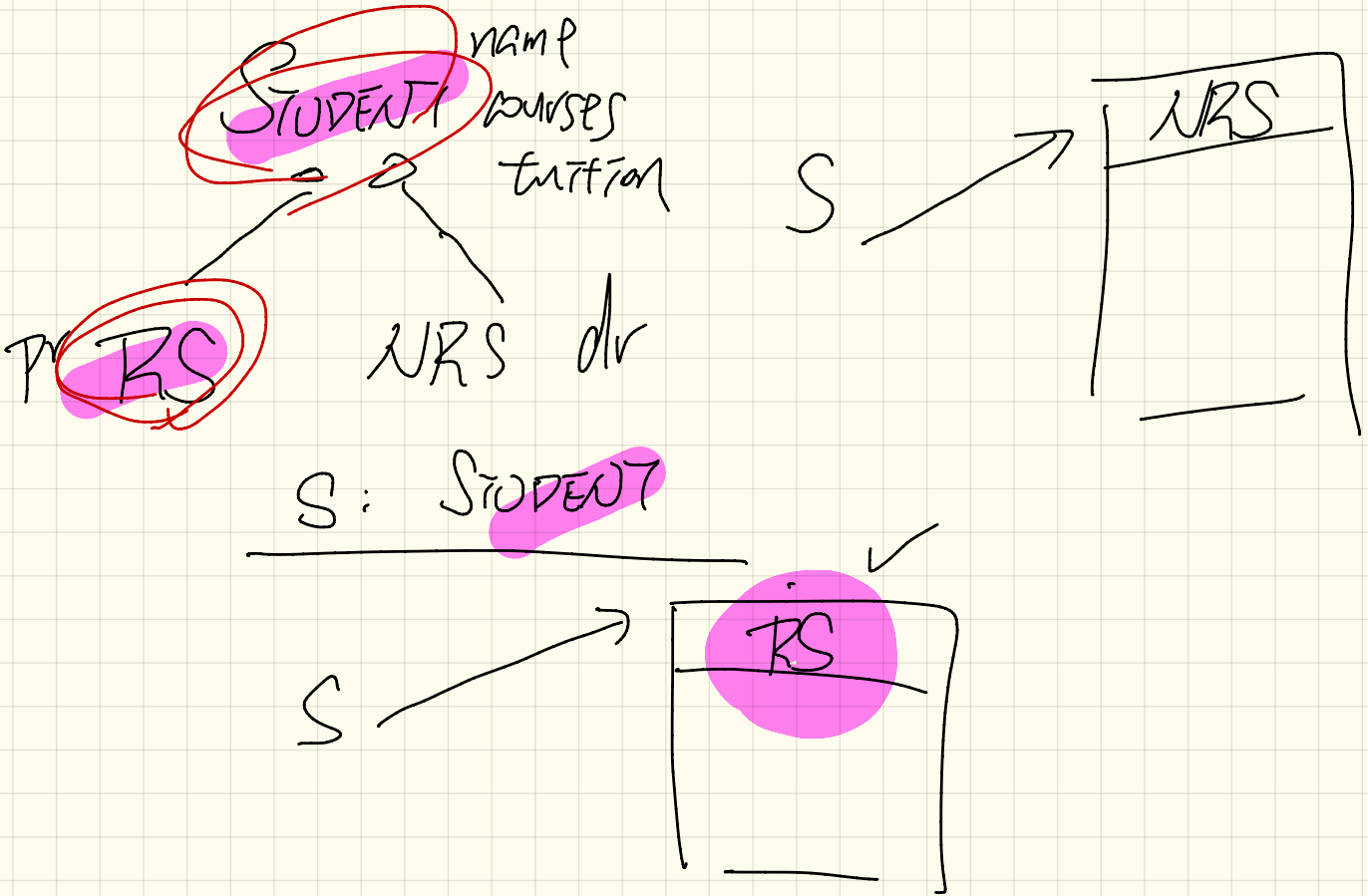
A, B, D

descendents

A ⊂ F

object variable	expectation
od	od.d
ob	ob.b ob.d
oa	oa.a oa.b

and

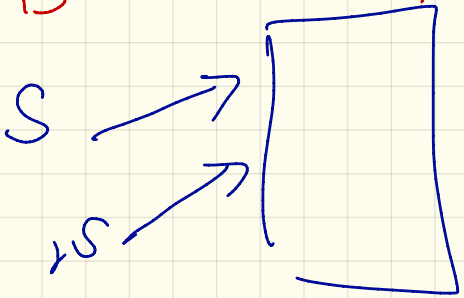

STUDENT
name
courses
tuition
PRS
NRS
dir
S
NRS
S : STUDENT
S
RS

S := rS

assignment.

rS := S ✗

ST: RS ST: STUDENT



We substitute rS for S.

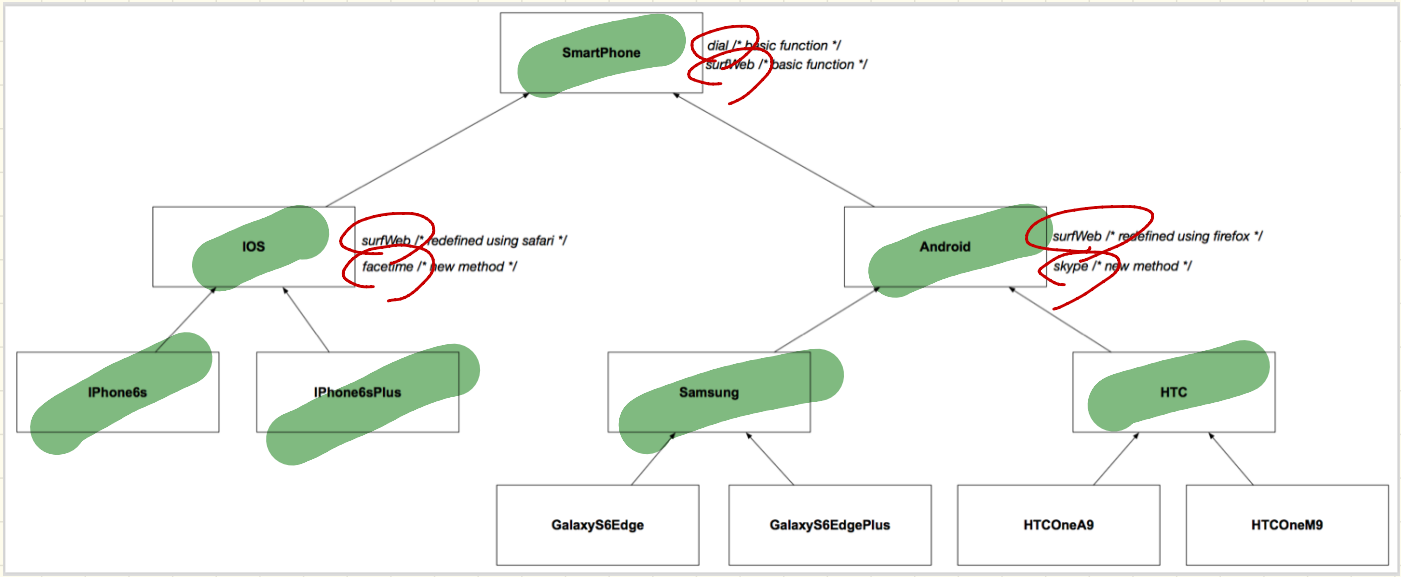
S: STUDENT
rS: RS
nrS: NRS

S := rS

ST: STUDENT ST: RS ✓

Should this assignment compile?

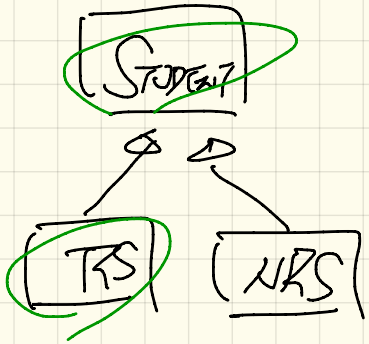
Is the ST of rS a descendent class of ST of S?


SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
Samsung
HTC
GalaxyS6Edge
GalaxyS6EdgePlus
HTCOneA9
HTCOneM9

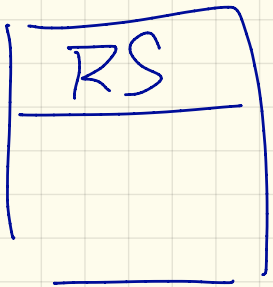
S: STUDENT

create {RS} S.make(...)

do they compile?



STUDENT S



Is the new DT (RS) a descendent class of the ST of S?

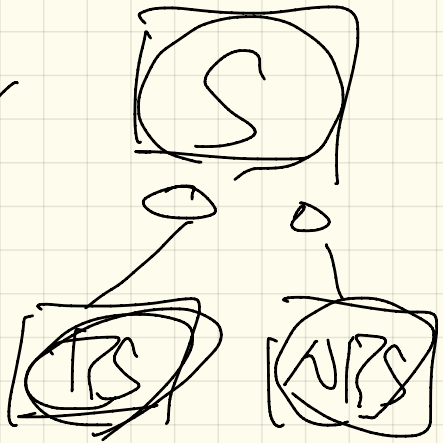
STUDENT

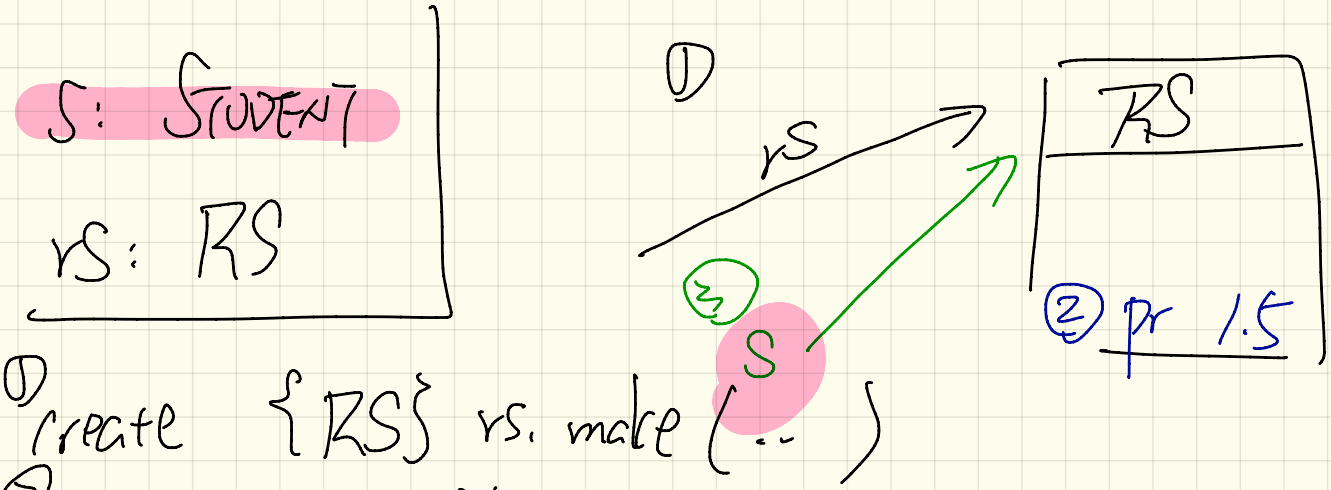
rs : RS

① create {STUDENT} rs. make (...) X

② create {URS} rs. make (...) X

③ create {RS} rs. make (..) ✓





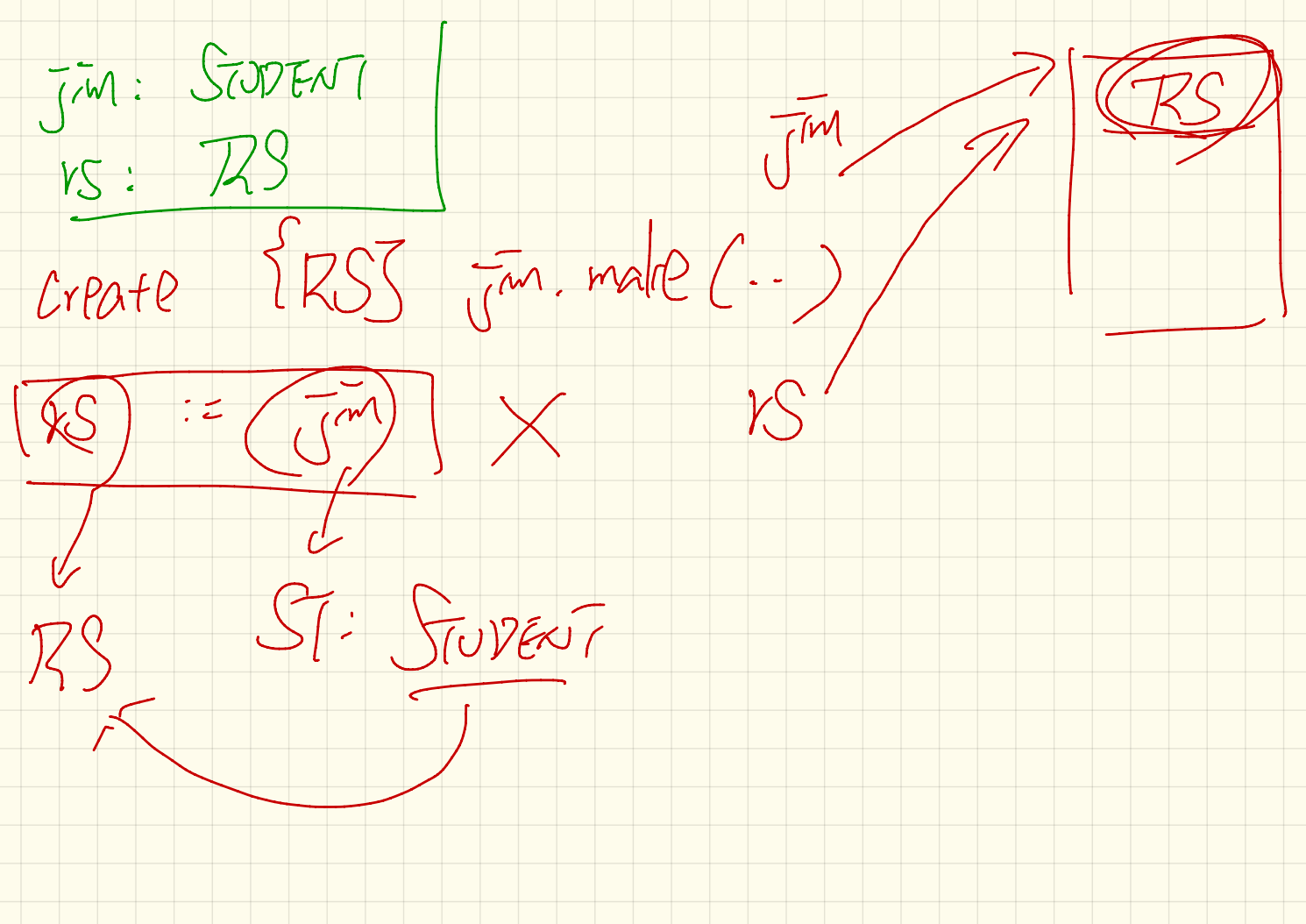
① create {RS} rs.make (...)

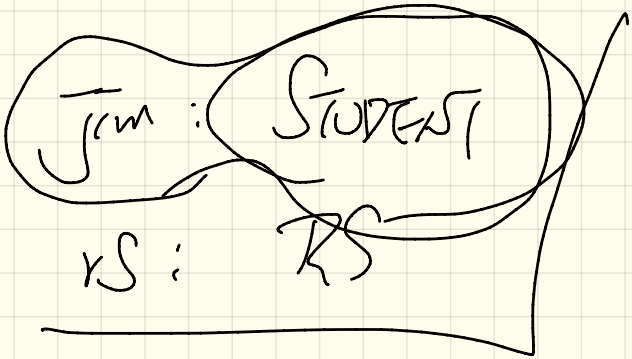
② rs.set_pr(1.5)

③ s := rs

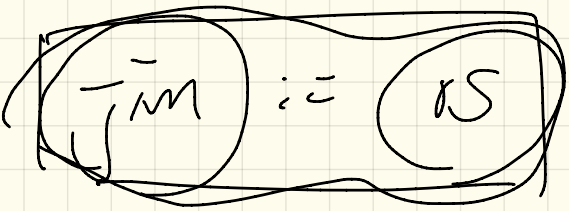
④ s.set_pr(1.7)

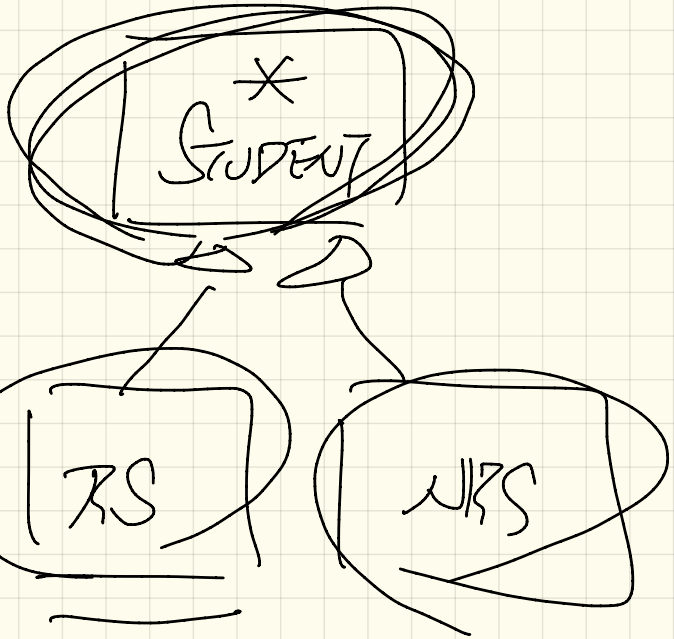
X does not compile
∵ ST of s (STUDENT) does not have set_pr feature.


Jim : STUDENT
RS : RS
create {RS} Jim.make(..)
RS := Jim
RS
ST: STUDENT
Jim
RS
RS



rs := S X





Jim.set_pr X

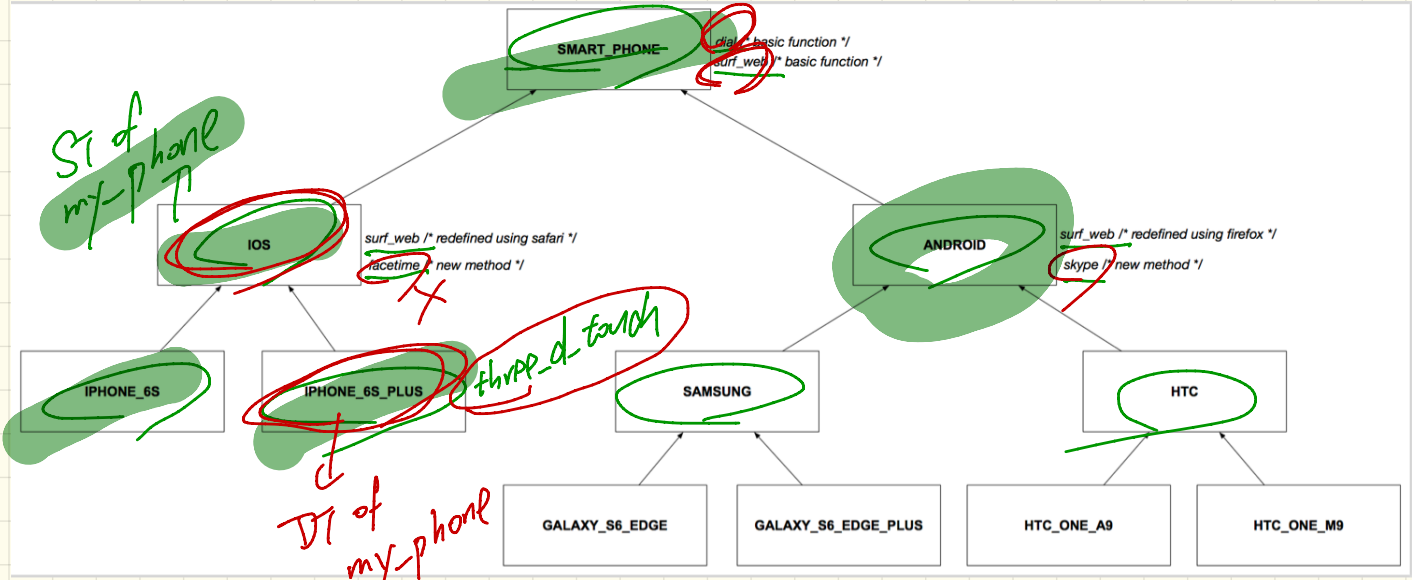
Jim.set_pb

s: STUDENT*

create {STUDENT} s.make

Lecture 12

Monday Oct. 23

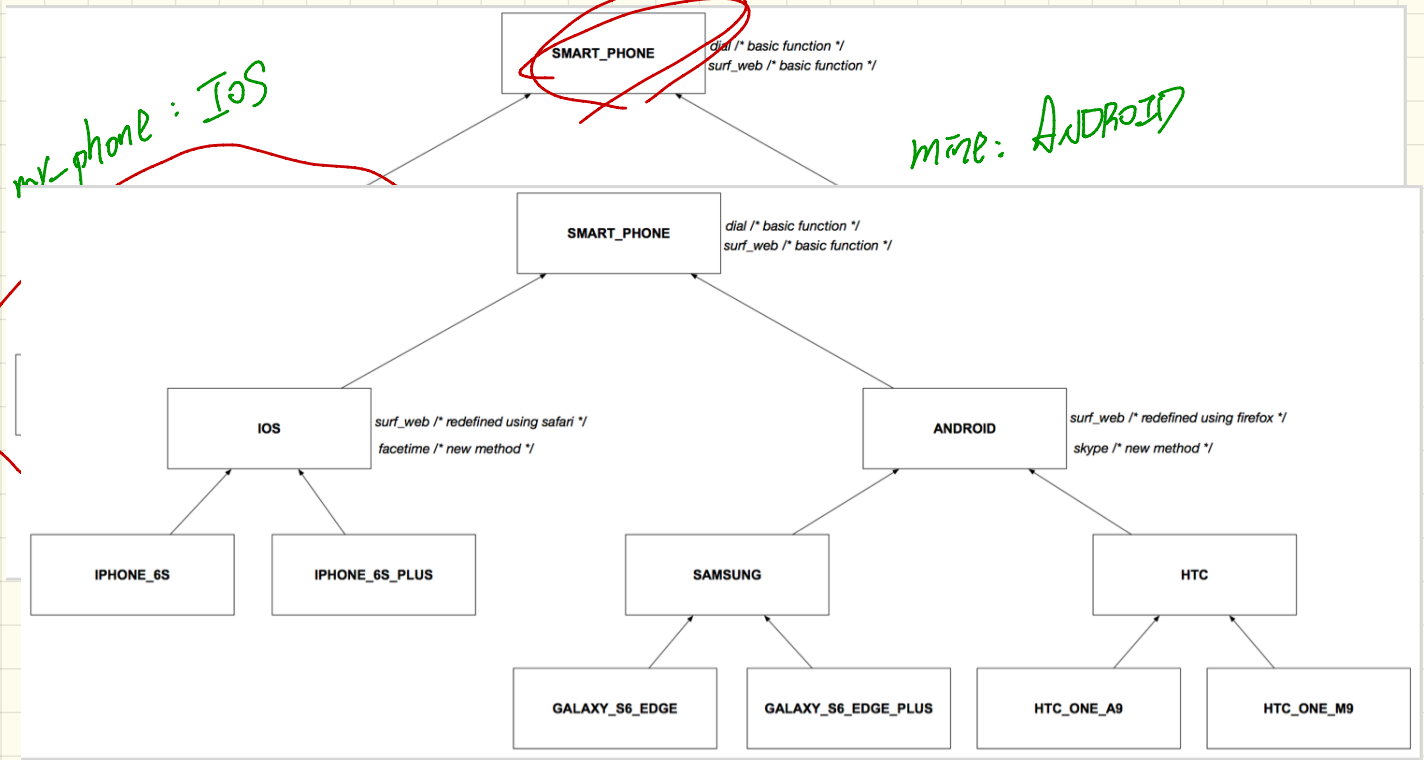


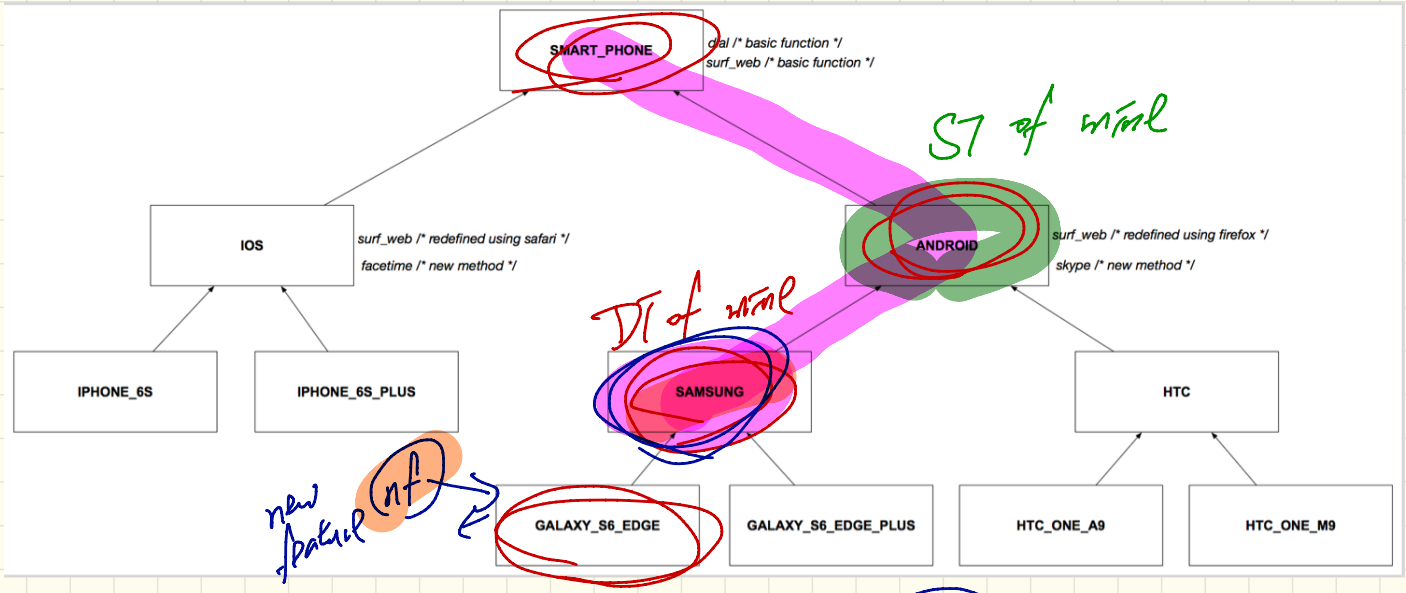
my_phone : IOS

Is the following cast ampible?

check attached {ANDROID} my_phone as android then

end


SMART_PHONE
dial /* basic function */
surf_web /* basic function */
IOS
surf_web /* redefined using safari */
facetime /* new method */
ANDROID
surf_web /* redefined using firefox */
skype /* new method */
IPHONE_6S
IPHONE_6S_PLUS
SAMSUNG
HTC
GALAXY_S6_EDGE
GALAXY_S6_EDGE_PLUS
HTC_ONE_A9
HTC_ONE_M9



check attached {G_S6_E} mine as P then

. . .

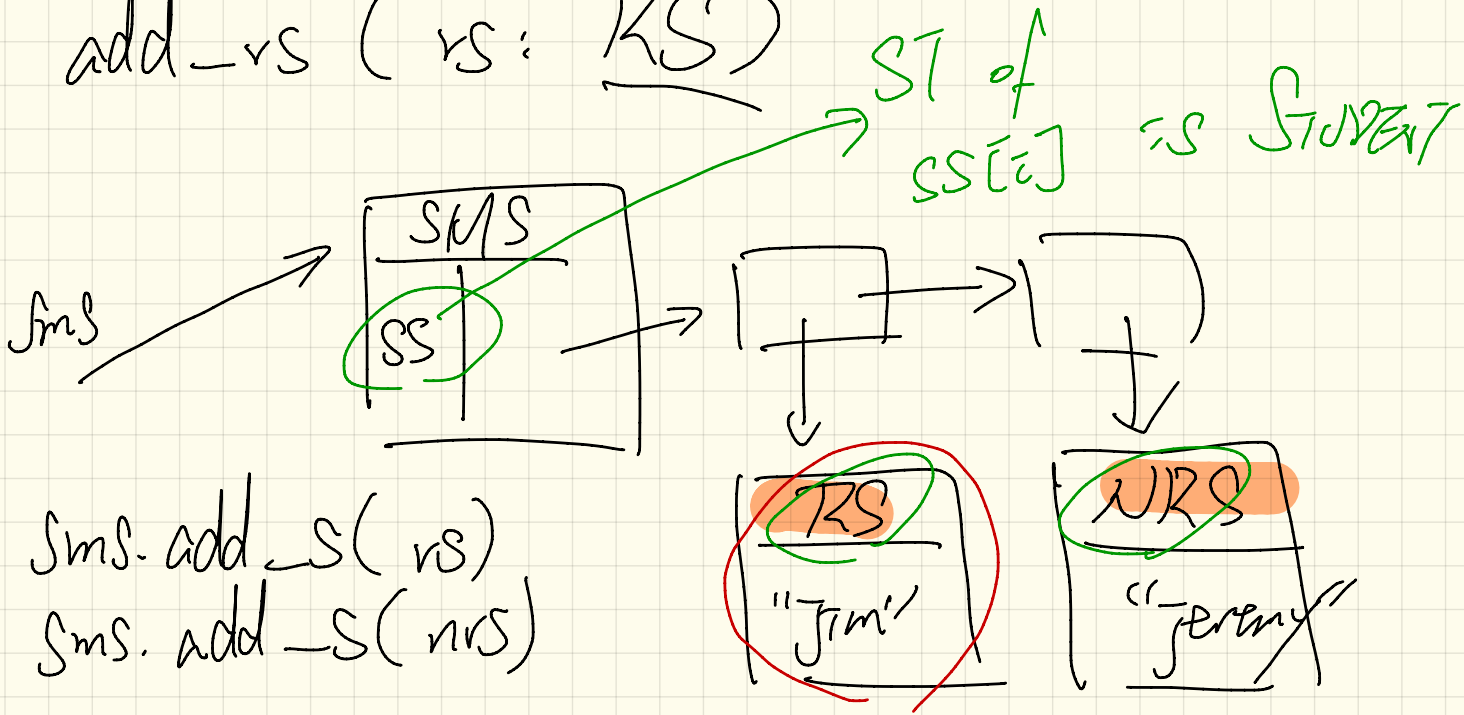
end.

if this_case was allowed:
1. DT of P remains SAMSUNG
2. Temp. ST of P: G_S6_E
⇒ P.nf

contradiction if DT does not support nf

add_s(s : STUDENT)

add_rs (rs : RS)



sms.add_s(rs)

sms.add_s(nrs)

SMS

get_student (i : INT) : STUDENT

end

ST of return value of get_S is

(sms.get_student(1).tuition ①

_____ . set_pr ②

_____) . set_dr ③

Lecture 13

Wednesday Oct. 25

```
class BOOK [G]
  names: ARRAY[STRING]
  records: ARRAY[ANY]
  -- Create an empty book
  make do ... end
  -- Add a name-record pair to the book
  add (name: STRING; record: ANY) do ... end
  -- Return the record associated with a given name
  get (name: STRING): ANY do ... end
end
```

DATE [G] → parameter

generic class

[G] DATE

[G] DATE

[G] DATE

avg(x, y) = 2(x + y)/2 3

parameters

Supplier.

b: BOOK[DATE] [3] create {BOOK[DATE]} b.make

```
1  birthday: DATE; phone_number: STRING
2  b: BOOK; is_wednesday: BOOLEAN
3  create {BOOK} b.make
4  phone_number := "416-677-1010"
5  b.add ("SuYeon", phone_number)
6  create {DATE} birthday.make(1975, 4, 10)
7  b.add ("Yuna", birthday)
8  is_wednesday := b.get("Yuna").get_day() = 4
```

class of DATE

STRING not a descendant

client

things returned from Book are DATE's only.

```
class BOOK [G] ANY
  names: ARRAY[STRING]
  records: ARRAY[ANY] G ANY
  -- Create an empty book
  make do ... end
  -- Add a name-record pair to the book
  add (name: STRING; record: ANY) do ... end
  -- Return the record associated with a given name
  get (name: STRING): ANY do ... end
end
```

b: Book[ANY]

b.get(..).get_day

ANY

Fix 1: make MY_Book generic

```
class MY_Book[E, F]
    imp: DICTIONARY[V, K]
    :
    :
end
```

1. V and K are not known classes
2. V and K are not parameters of the current class

Fix 2: make MY_Book not generic

```
class MY_Book
    imp: DICTIONARY[RECORD, STRING]
```

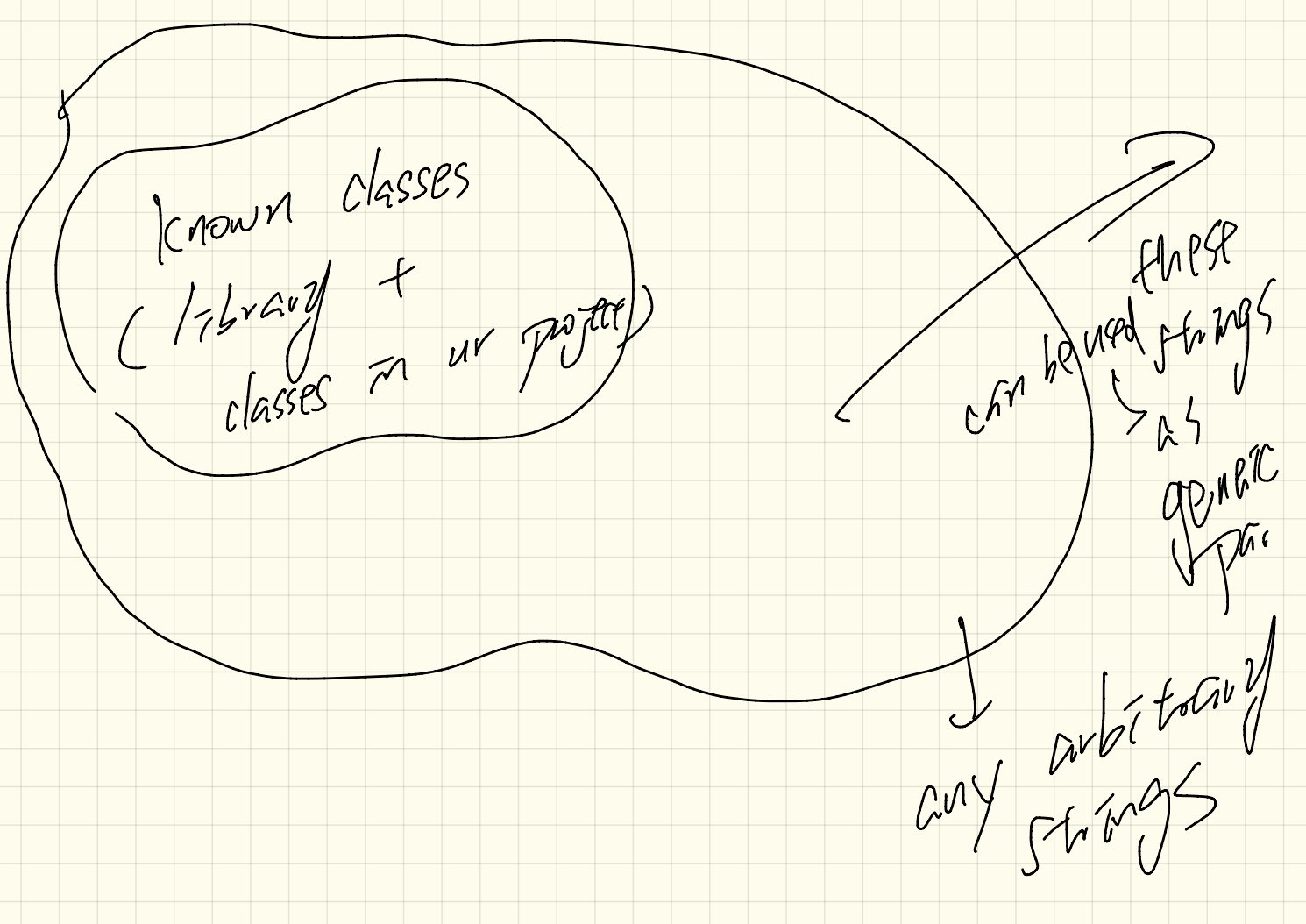
class DICTIONARY[E, F]

imp1: DIC[E, F]

imp2: DIC[E, STRING]

imp3: DIC[STRING, F]

imp4: DIC[INT, STRING]


known classes
(library +
classes in ur project)
can be used
these strings
as
generic
types
any arbitrary
strings

class Book [I_LOVE_EIFFEL, EIFFEL_IS_GREAT]

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

T F
α ✓ γ2

10%

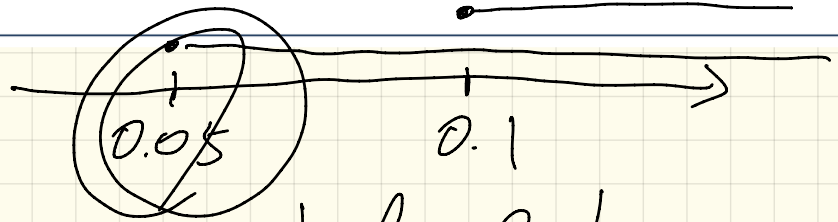
```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.05 -- 5%
    ensure then
      δ: ∀e : Result | e happens today between 9am and 5pm
end
```

bad design: γ2 : b_l ≥ 0.15

if this is good!

① α ⇒ γ

② γ ⇒ α ✗ e.g. b_l = 0.06



50% 50%

1. Reduce % on programming

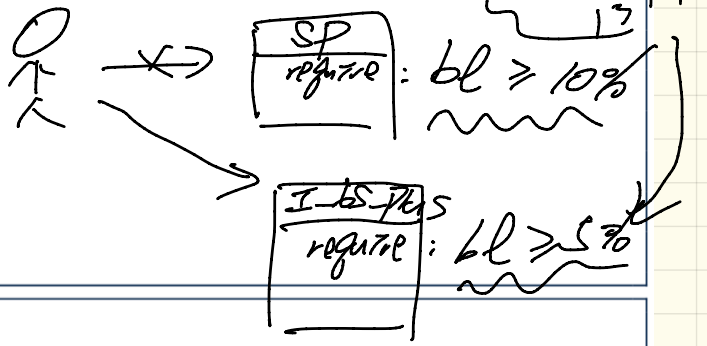
2. Reduce % on written

Lecture 14

Monday Oct. 30

e happens today ∧ e happens between 9am and 5pm ⇒ e happens today

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```




```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.05 -- 5%
    ensure then
      δ: ∀e : Result | e happens today between 9am and 5pm
end
```

bl ≥ 10% ⇒ bl ≥ 5%

require from SP

require from IP6S+

$p \wedge q \Rightarrow p$

$p \wedge q \Rightarrow q$

$p \vee \text{false} \equiv p$

$p \wedge \text{true} \equiv p$

e happens today ∨ e happens tmw ⇒ e happens today

e happens tmw

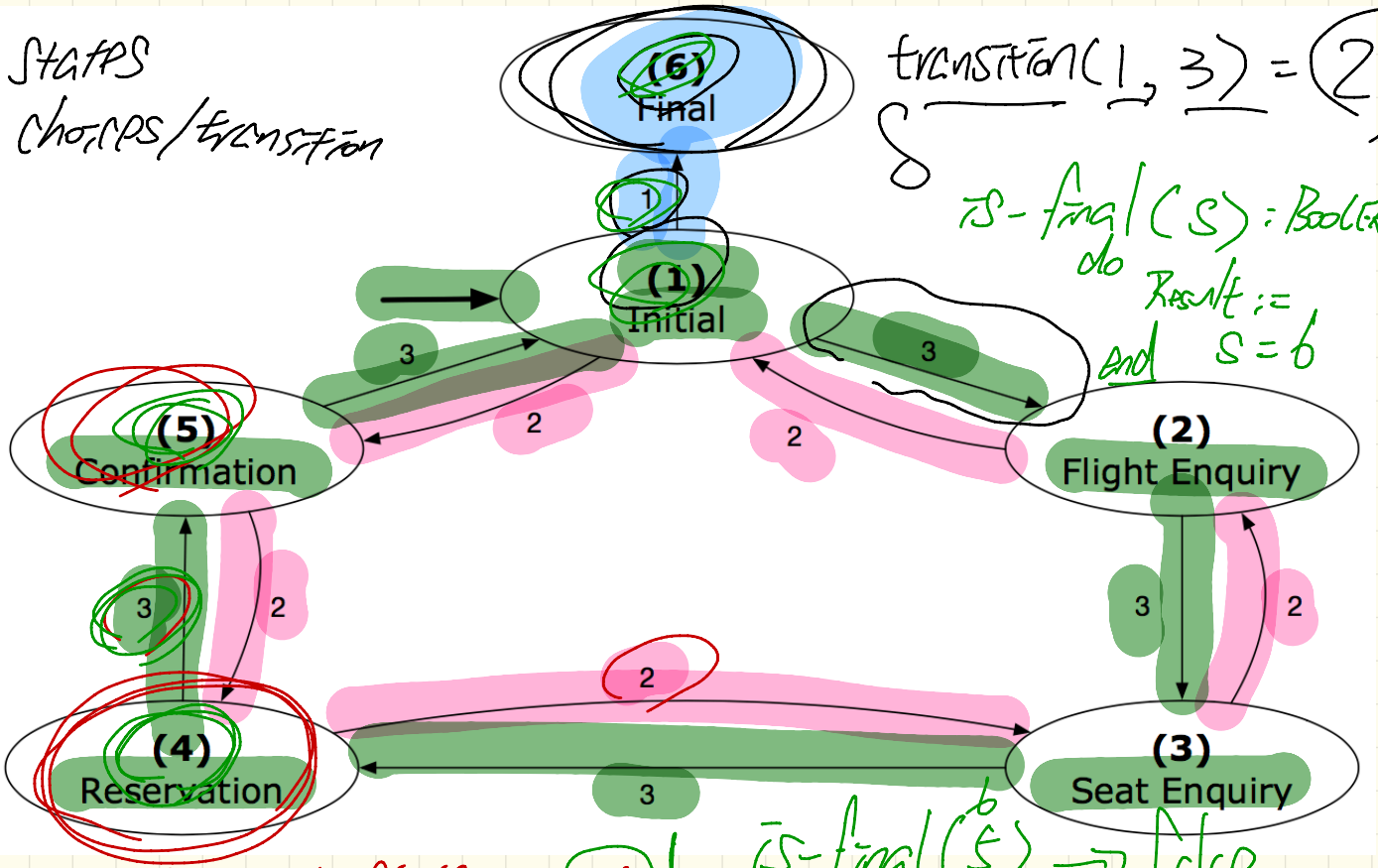
```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

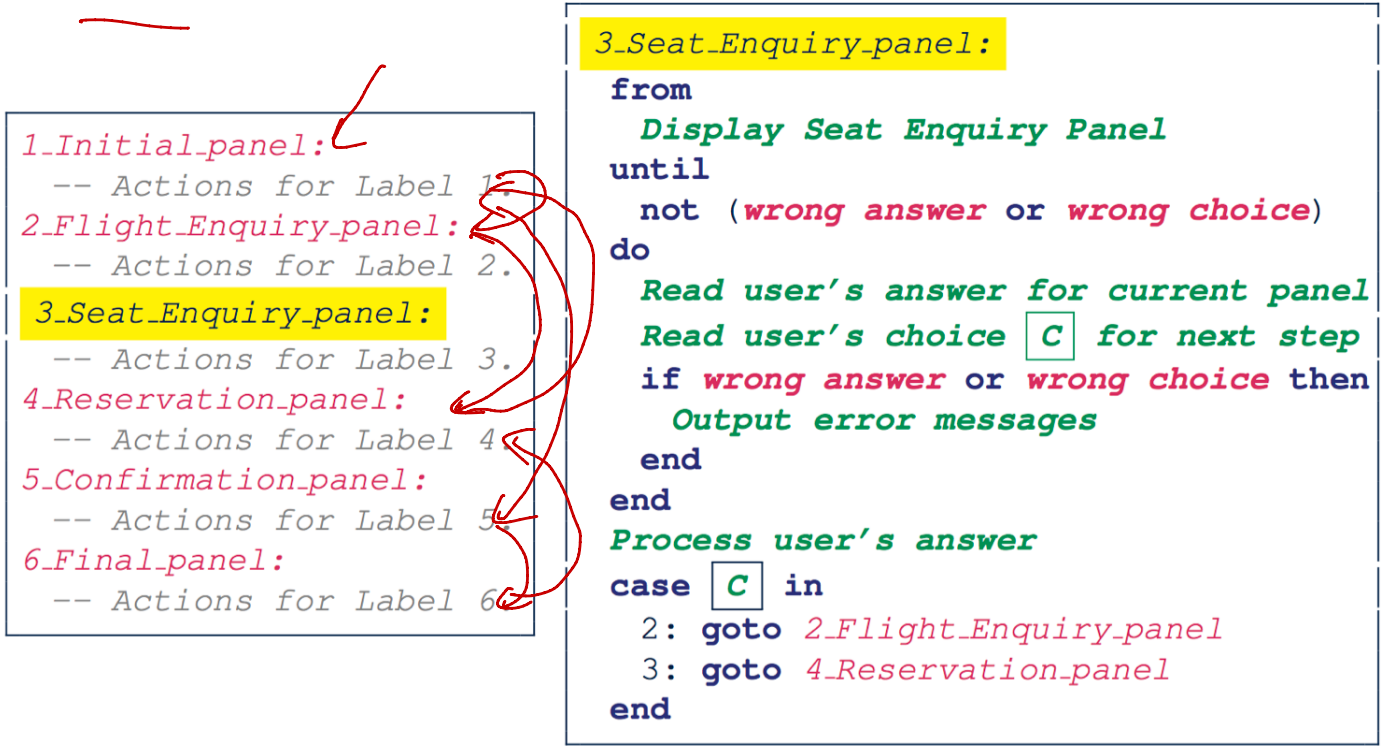
```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e : Result | e happens today or tomorrow
end
```

Is the precond of I_6S_Plus ok?

bl ≥ 10% ⇒ bl ≥ 15%

e.g. (11%)

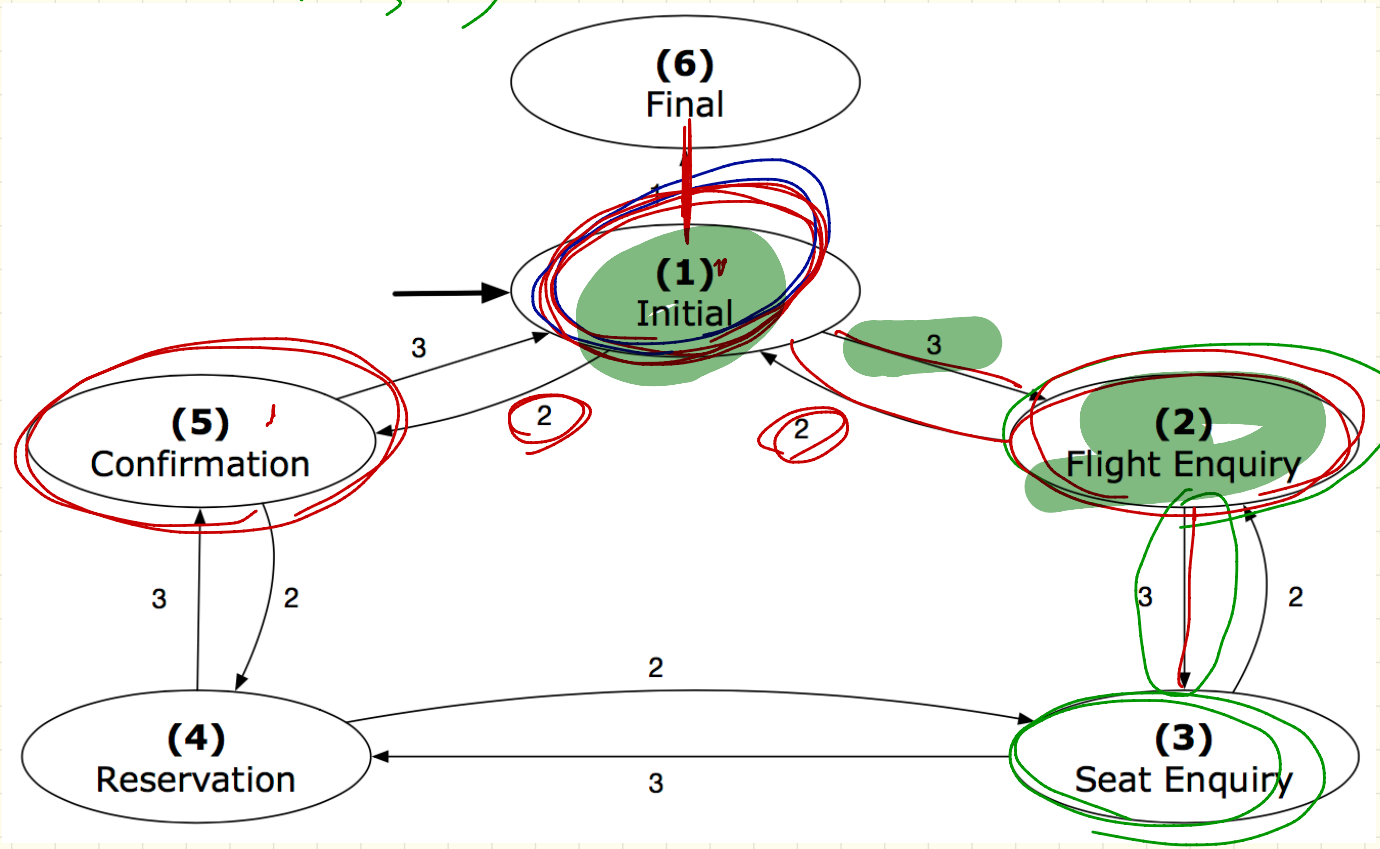

States
Choices/transition
transition(1, 3) = 2
is-final(S): Boolean
do
Result :=
end
S = 6
(6)
Final
1
(1)
Initial
3
2
3
2
(5)
Confirmation
(2)
Flight Enquiry
3
2
3
2
2
(4)
Reservation
(3)
Seat Enquiry
3
current_state : (4) 1
choice : (3) 1
is-final(6) → false
transition(4, 3) returns (5) 6


1_Initial_panel:
-- Actions for Label 1.
2_Flight_Enquiry_panel:
-- Actions for Label 2.
3_Seat_Enquiry_panel:
-- Actions for Label 3.
4_Reservation_panel:
-- Actions for Label 4.
5_Confirmation_panel:
-- Actions for Label 5.
6_Final_panel:
-- Actions for Label 6.
3_Seat_Enquiry_panel:
from
Display Seat Enquiry Panel
until
not (wrong answer or wrong choice)
do
Read user's answer for current panel
Read user's choice C for next step
if wrong answer or wrong choice then
Output error messages
end
end
Process user's answer
case C in
2: goto 2_Flight_Enquiry_panel
3: goto 4_Reservation_panel
end

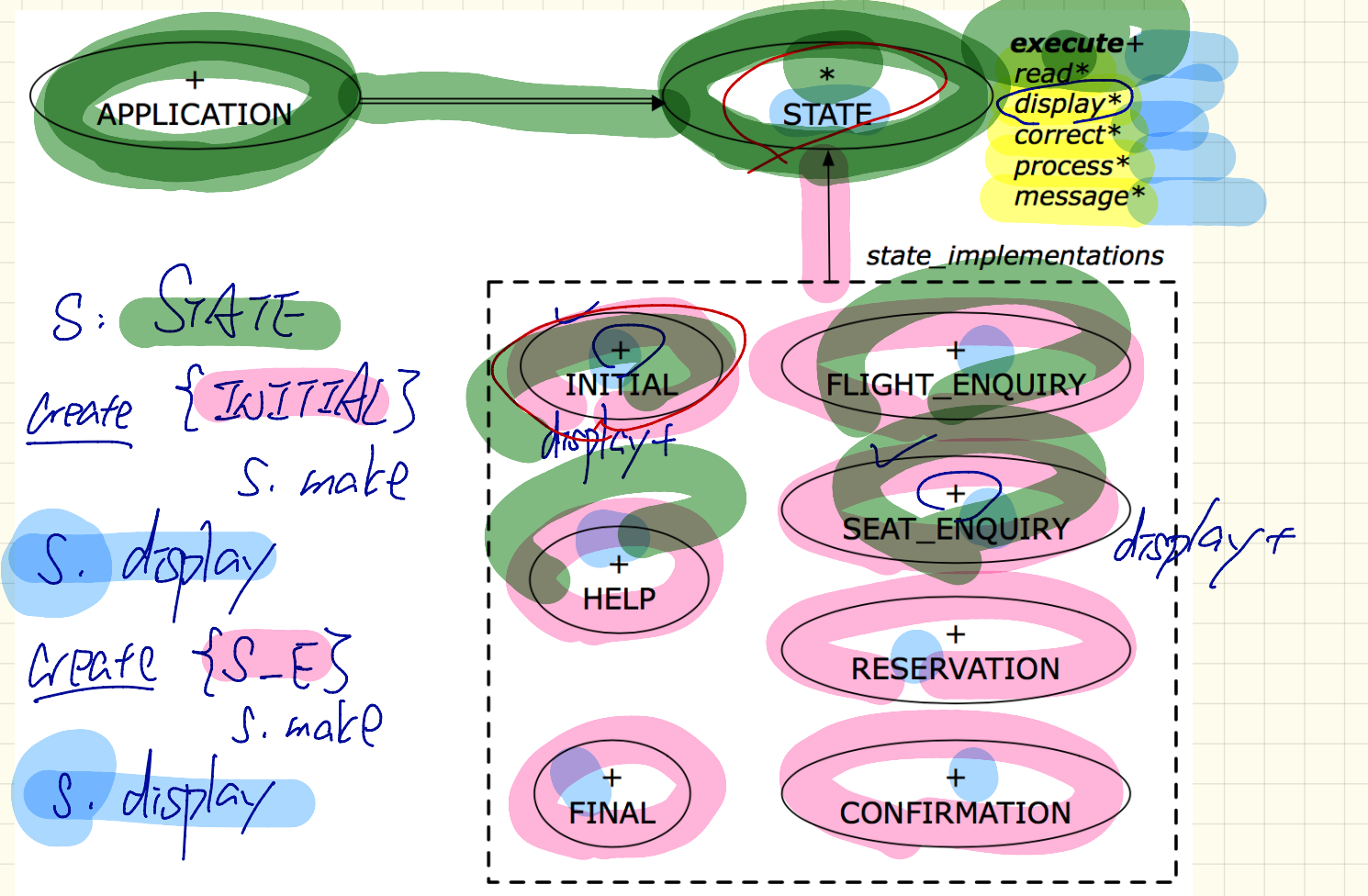
Lecture 15

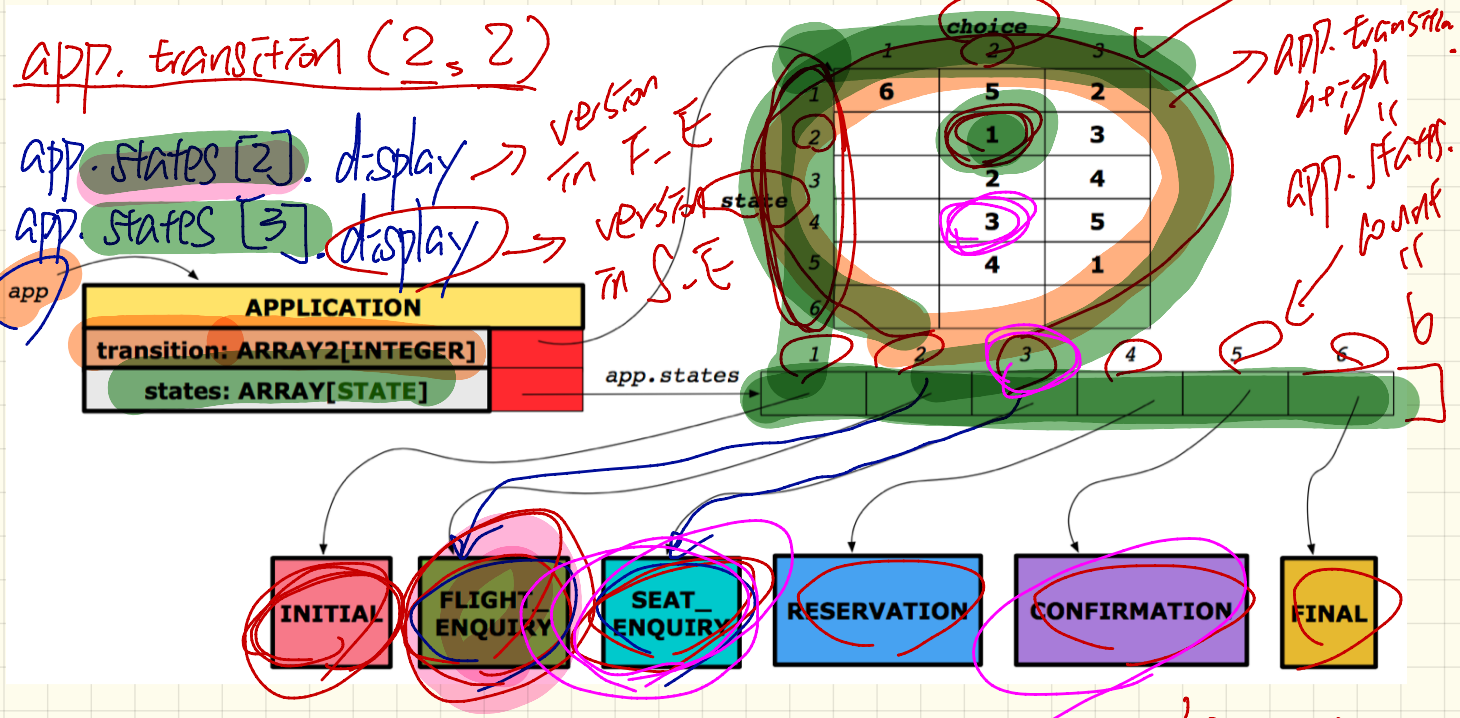
Wednesday Nov. 1

transition (2, 3) = 3



SRC STATE \ CHOICE	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–


+
APPLICATION
*
STATE
execute+
read*
display*
correct*
process*
message*
state_implementations
+
INITIAL
+
FLIGHT_ENQUIRY
+
SEAT_ENQUIRY
+
HELP
+
RESERVATION
+
FINAL
+
CONFIRMATION
S: STATE
create {INITIAL}
S.make
S.display
create {S_E}
S.make
S.display
display+
display+



version in F.E

version in S.E

app.transition. height is

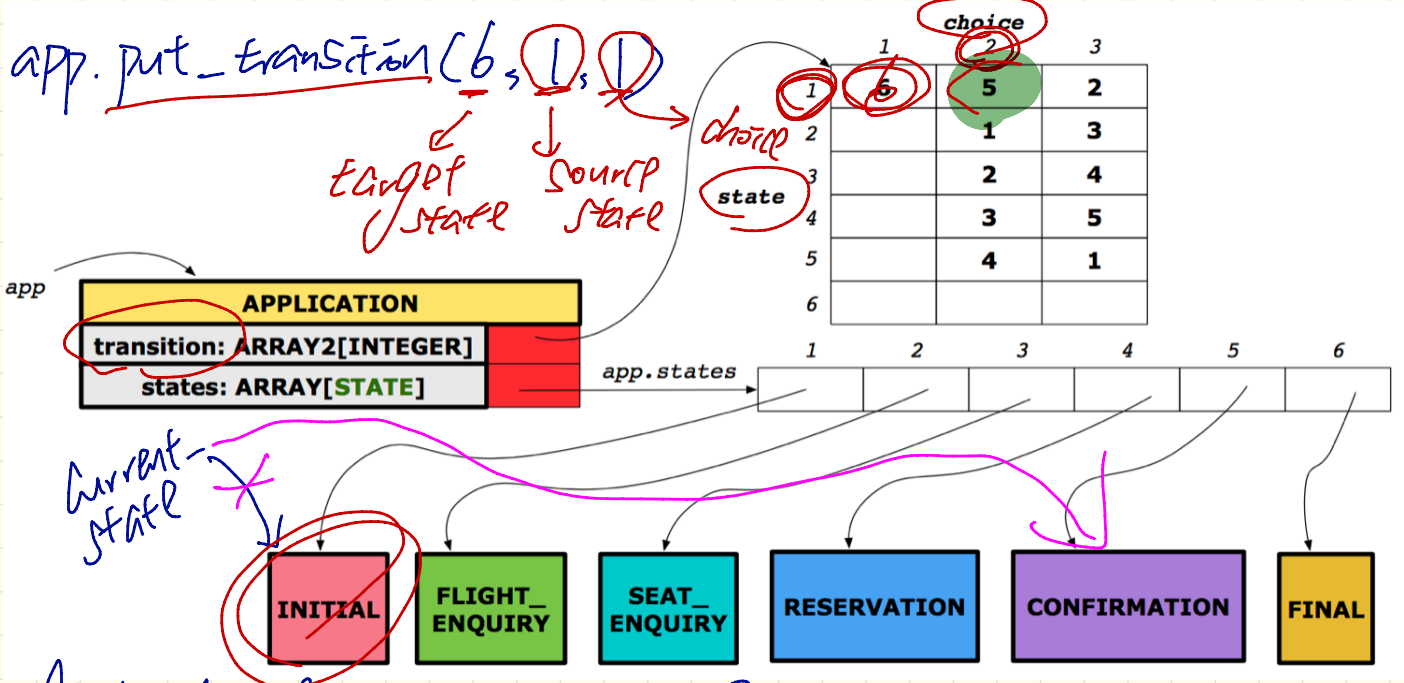
app.states. count is 6

states : ARRAY[STATE]

states can be stored into array.

↳ each item has ST STATE

∵ polymorphism objects whose types are descendants of



APPLICATION current_state : STATE

initial := 1

current_state := states [1]

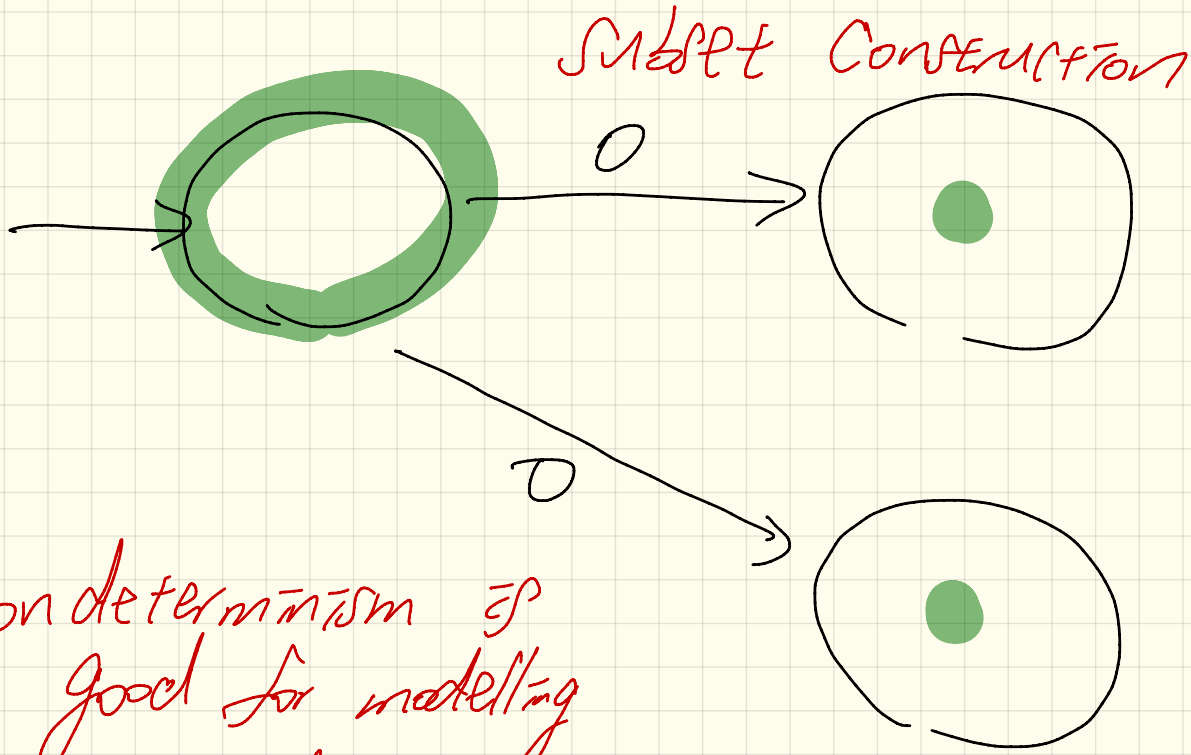
current_state.execute → version in INITIAL

choice : 2

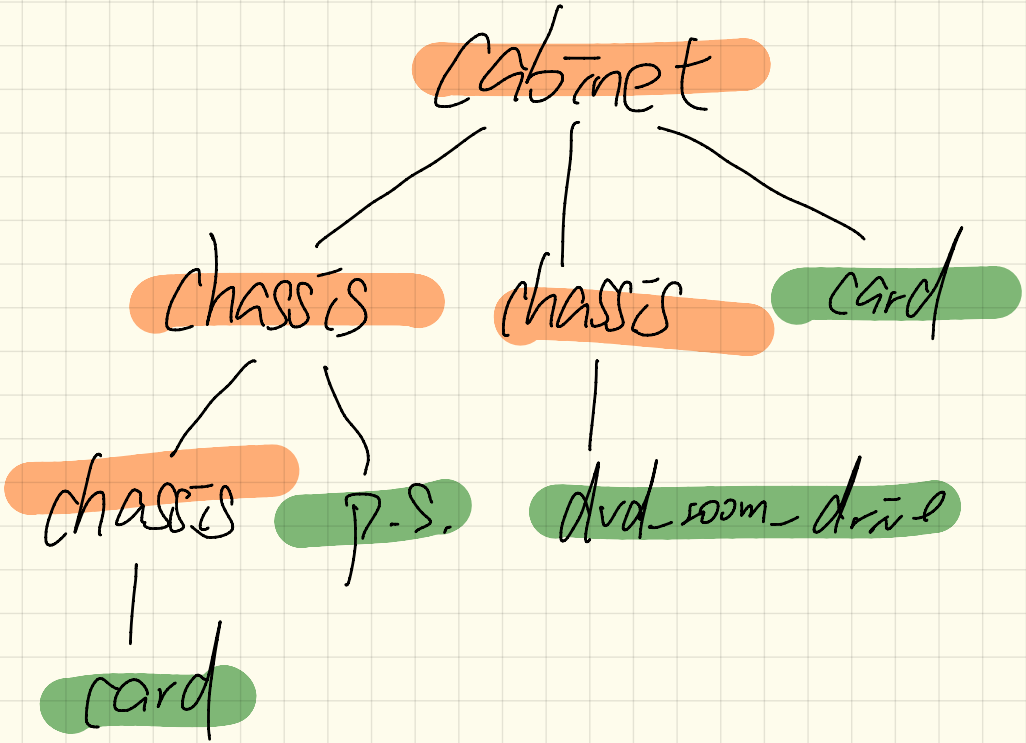
transition.item (1, 2)

current_state := → 5

states [5]

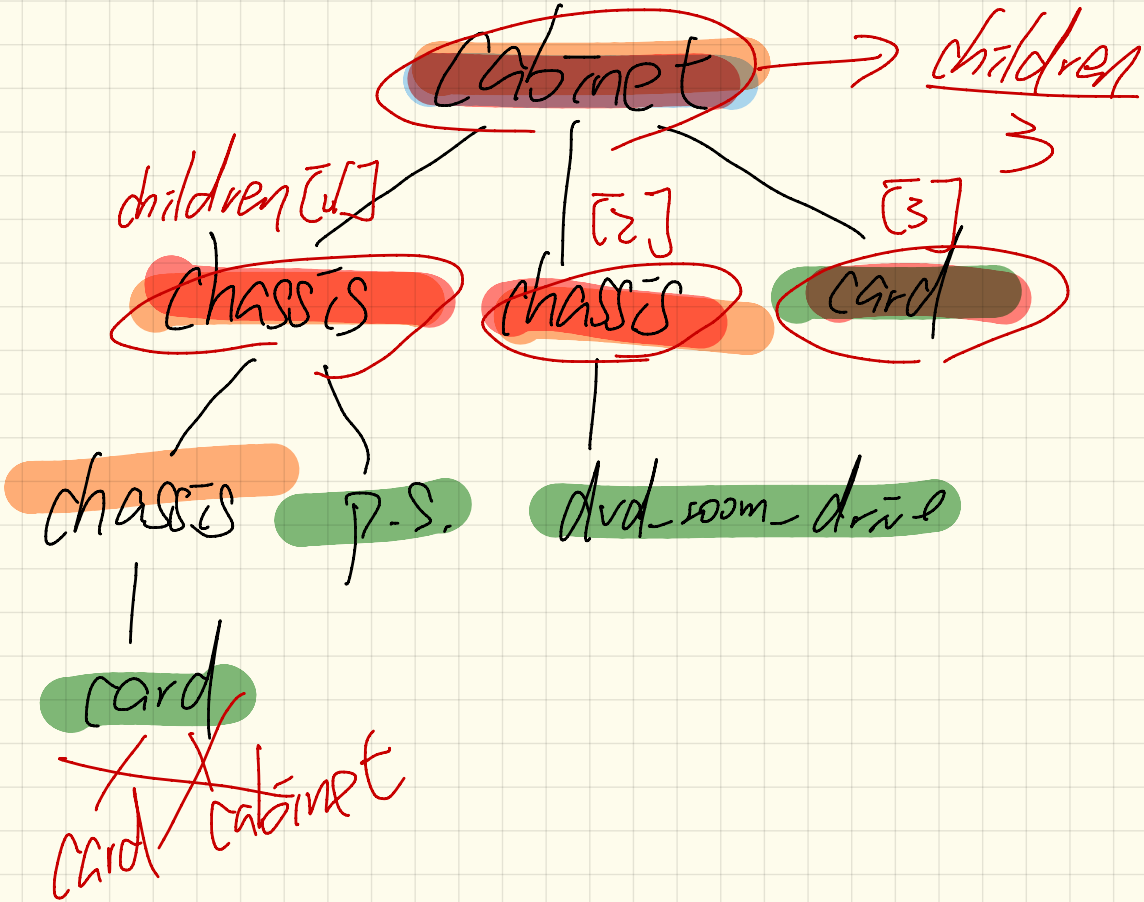


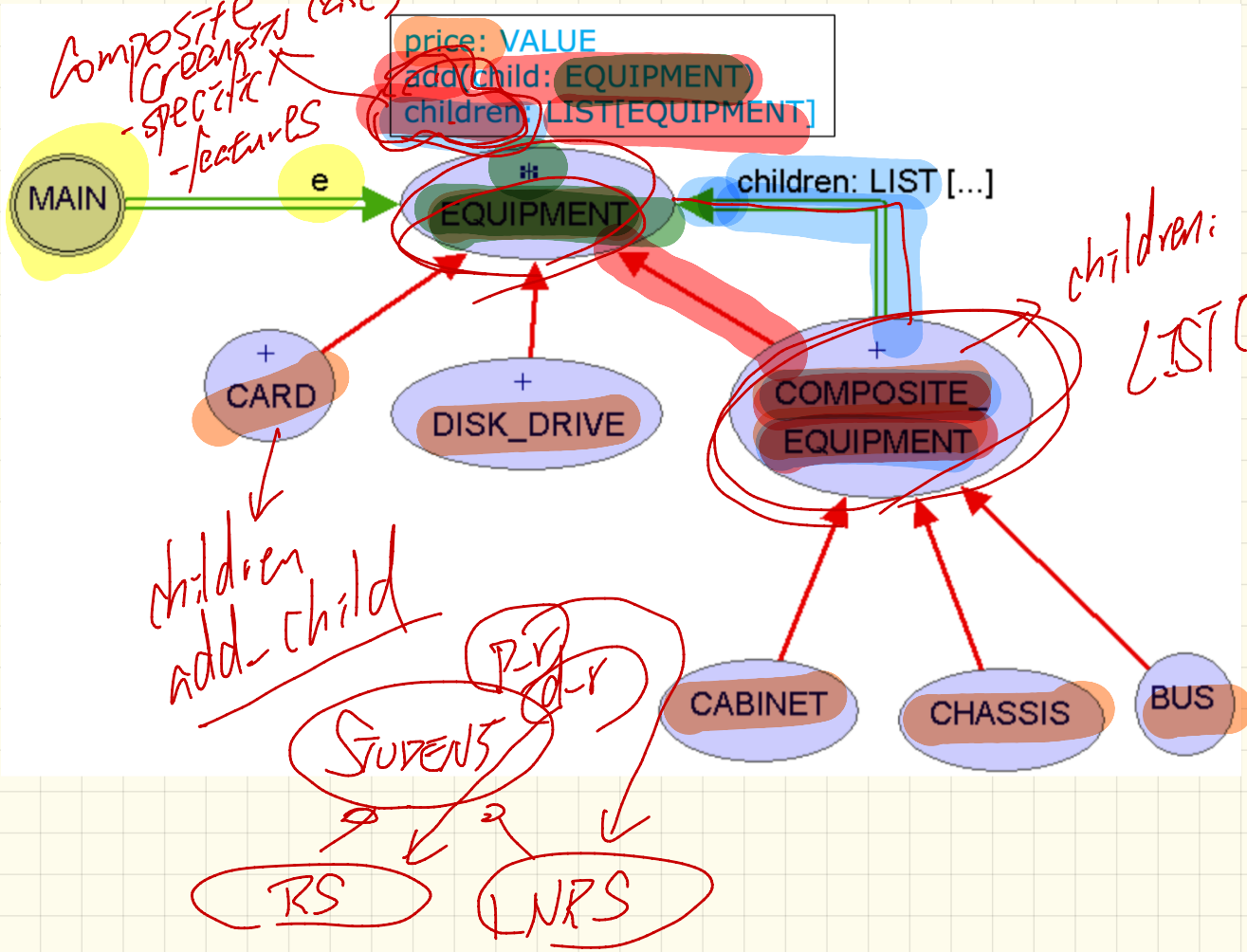
Nondeterminism is good for modelling but not for programming (i.e. not predictable).

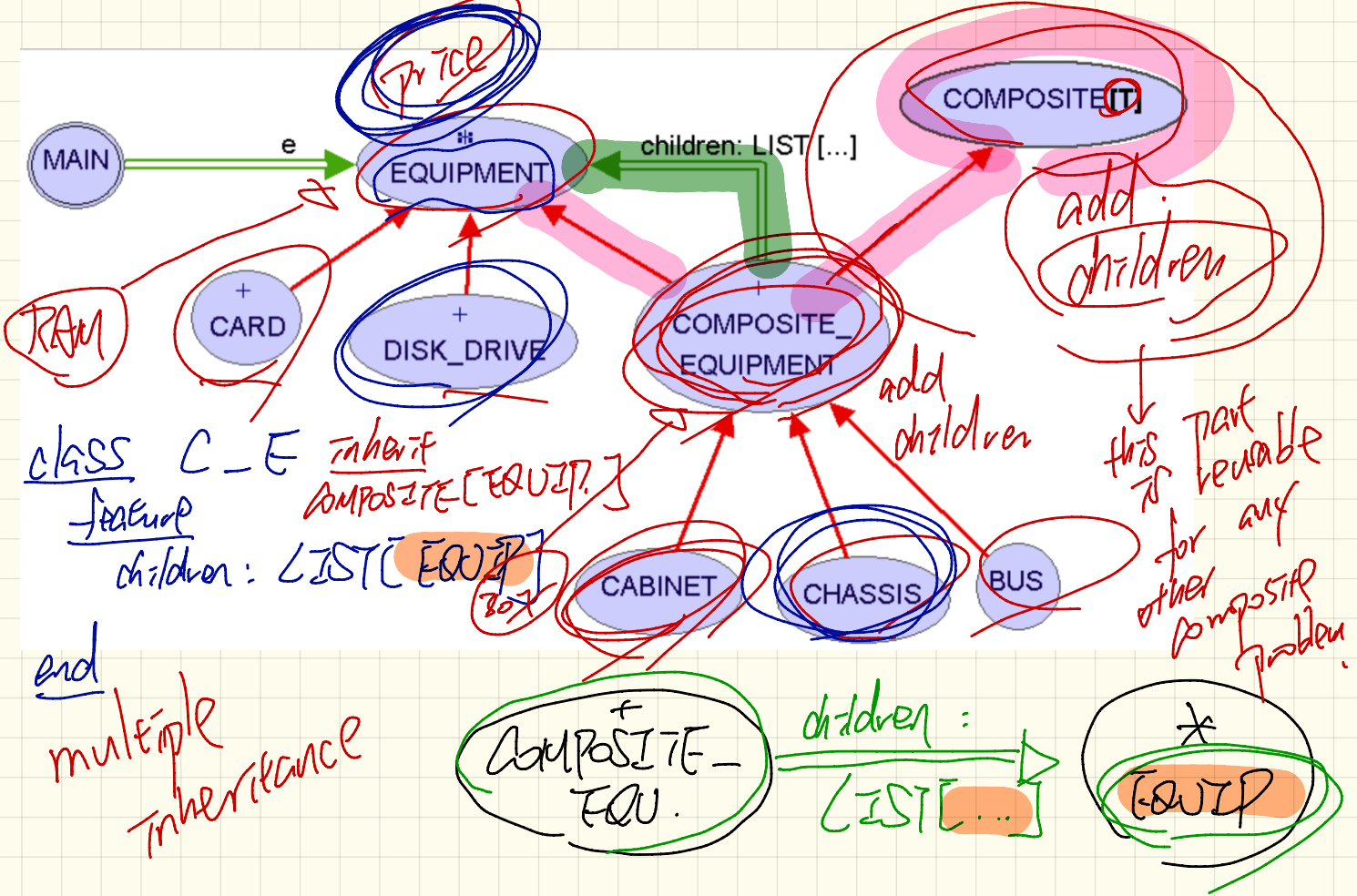

cabinet
chassis
chassis
card
chassis
p.s.
dvd_room_drive
card

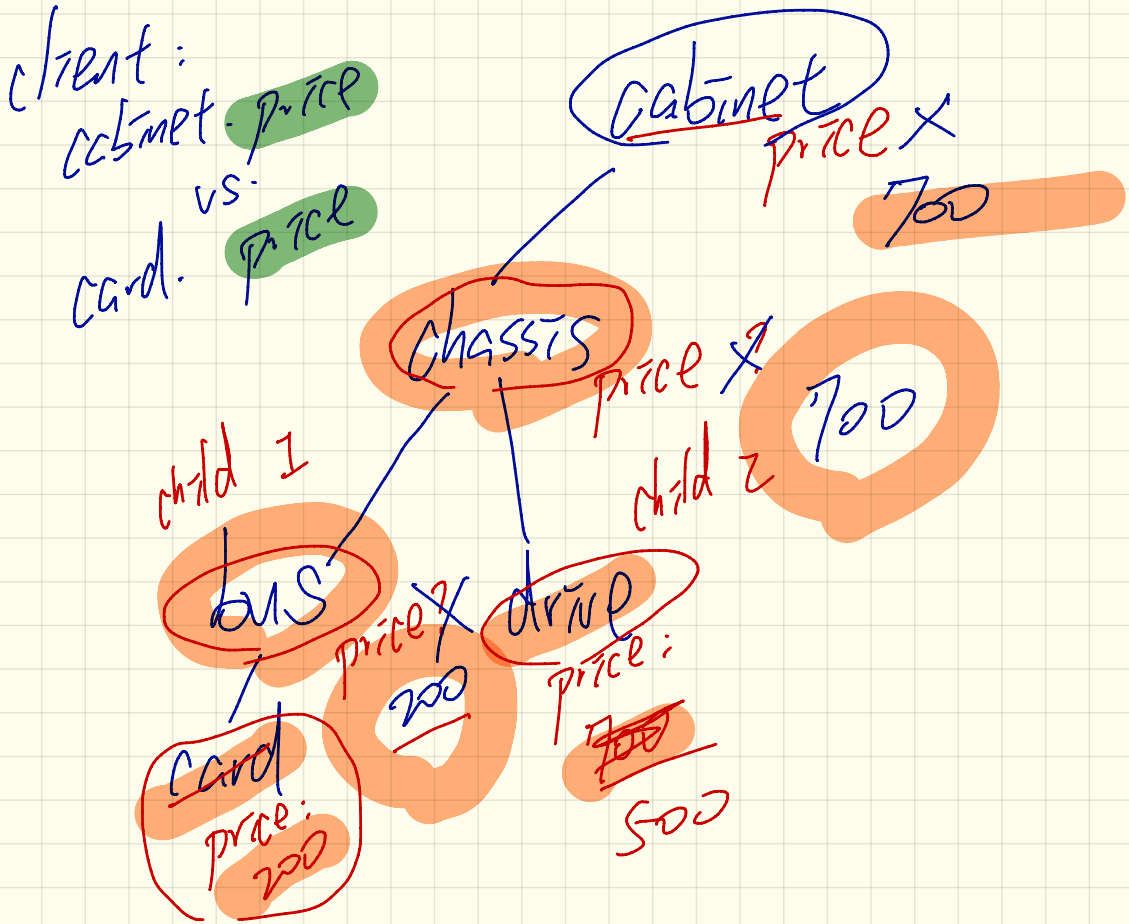
Lecture 16

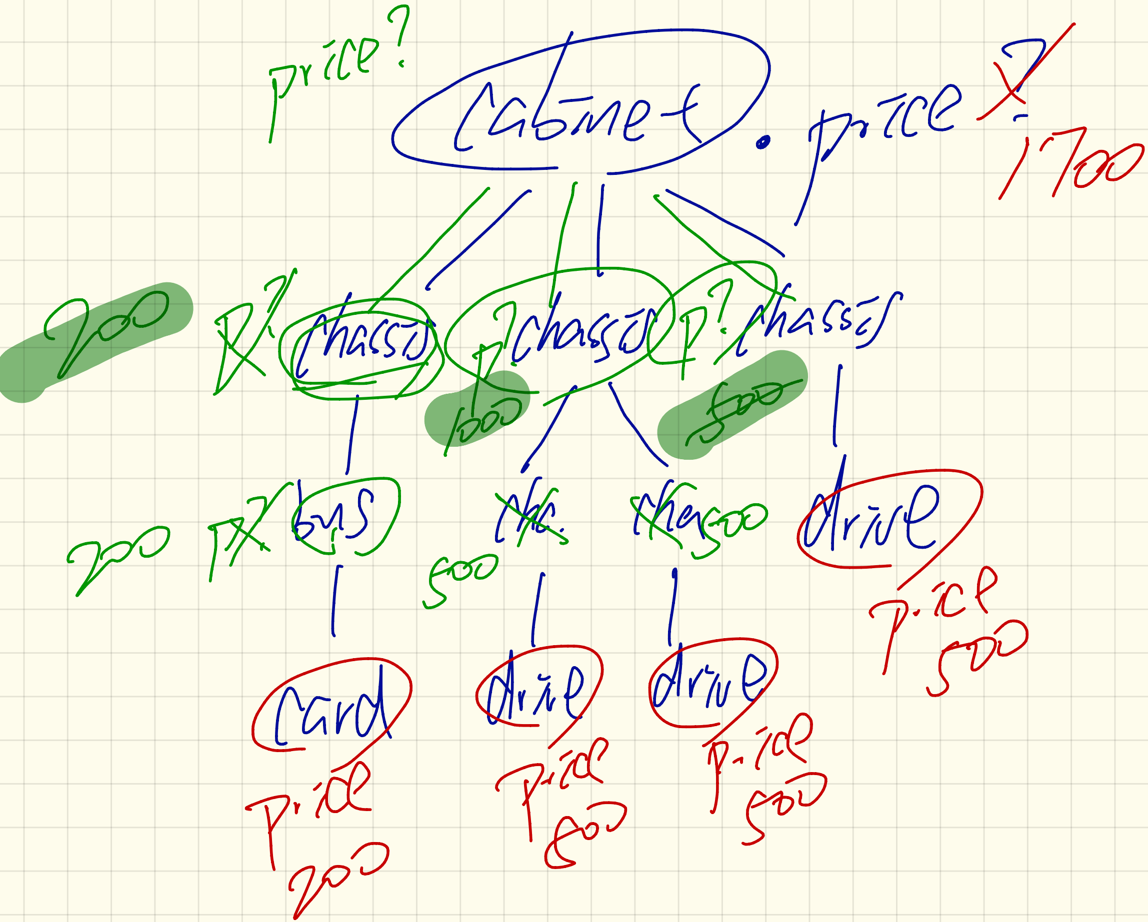
Monday Nov. 6

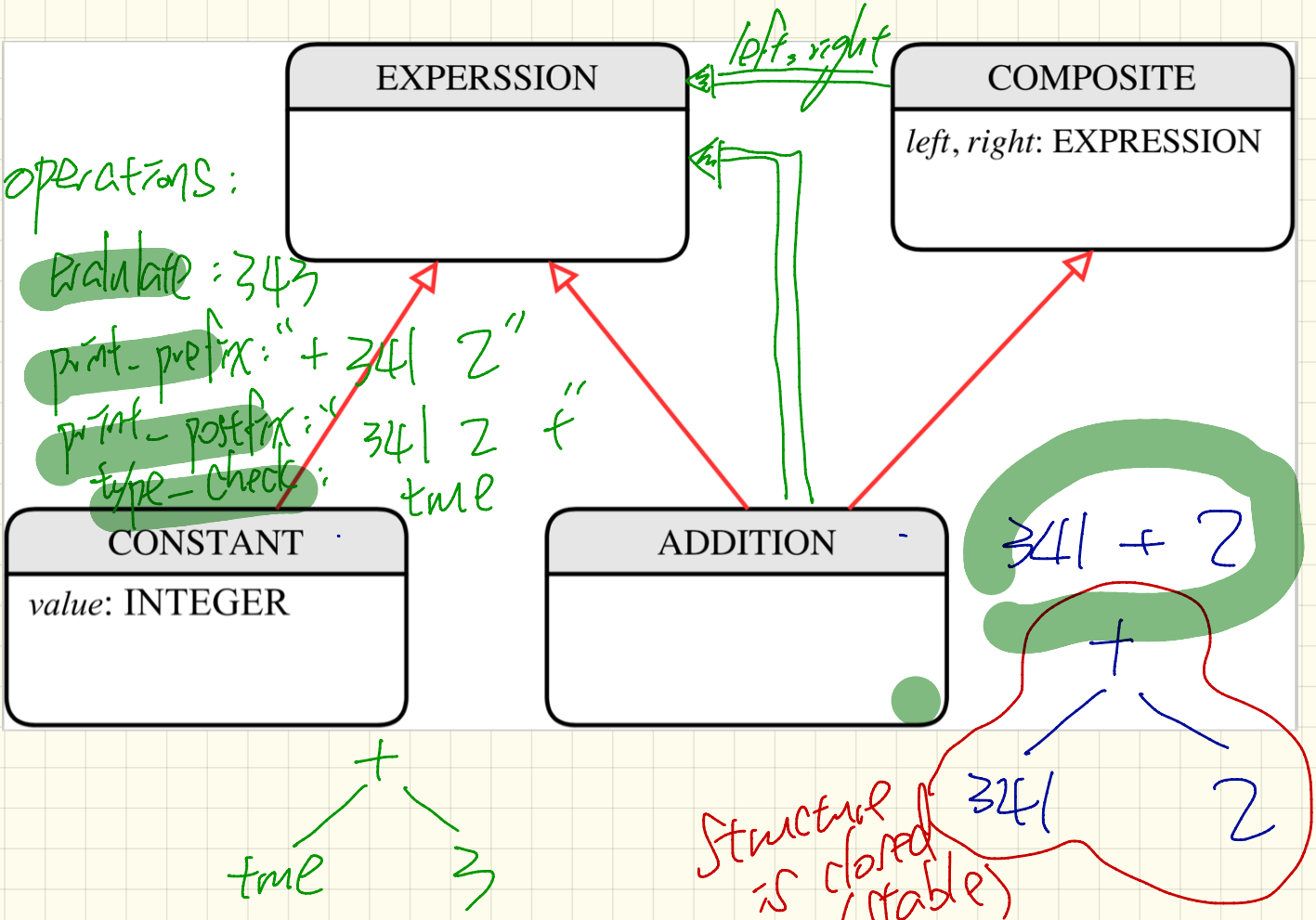

Cabinet
children
3
children[0]
[1]
[2]
[3]
chassis
chassis
card
chassis
P.S.
dvd_room_drive
card
card
cabinet

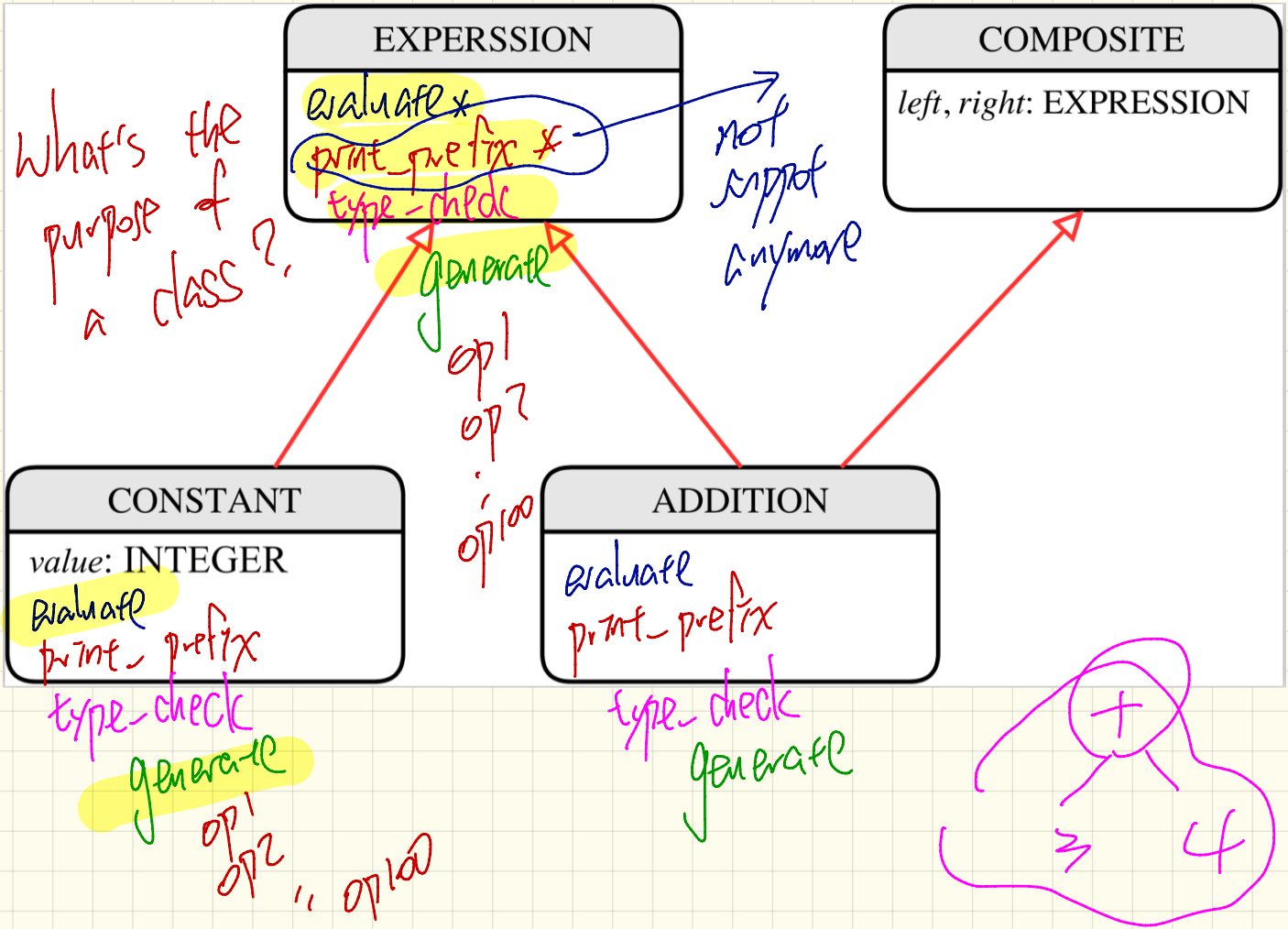

price: VALUE
add(child: EQUIPMENT)
children: LIST[EQUIPMENT]
MAIN
e
EQUIPMENT
children: LIST [...]
CARD
DISK_DRIVE
COMPOSITE_
EQUIPMENT
CABINET
CHASSIS
BUS
RS
LNRS


price
MAIN
e
EQUIPMENT
children: LIST [...]
COMPOSITE [T]
add children
RAM
+ CARD
+ DISK_DRIVE
+ COMPOSITE_ EQUIPMENT
add children
this is reusable for any other composite problem
class C_E
inherit COMPOSITE[EQUIP]
feature
children: LIST[EQUIP]
Box
CABINET
CHASSIS
BUS
end
multiple inheritance
+ COMPOSITE_ EQU.
children : LIST[...]
*
EQUIP


client:
cabinet.price
vs.
card.price
cabinet
price x
700
chassis
price x
700
child 1
child 2
bus
price x
200
drive
price:
500
card
price:
200

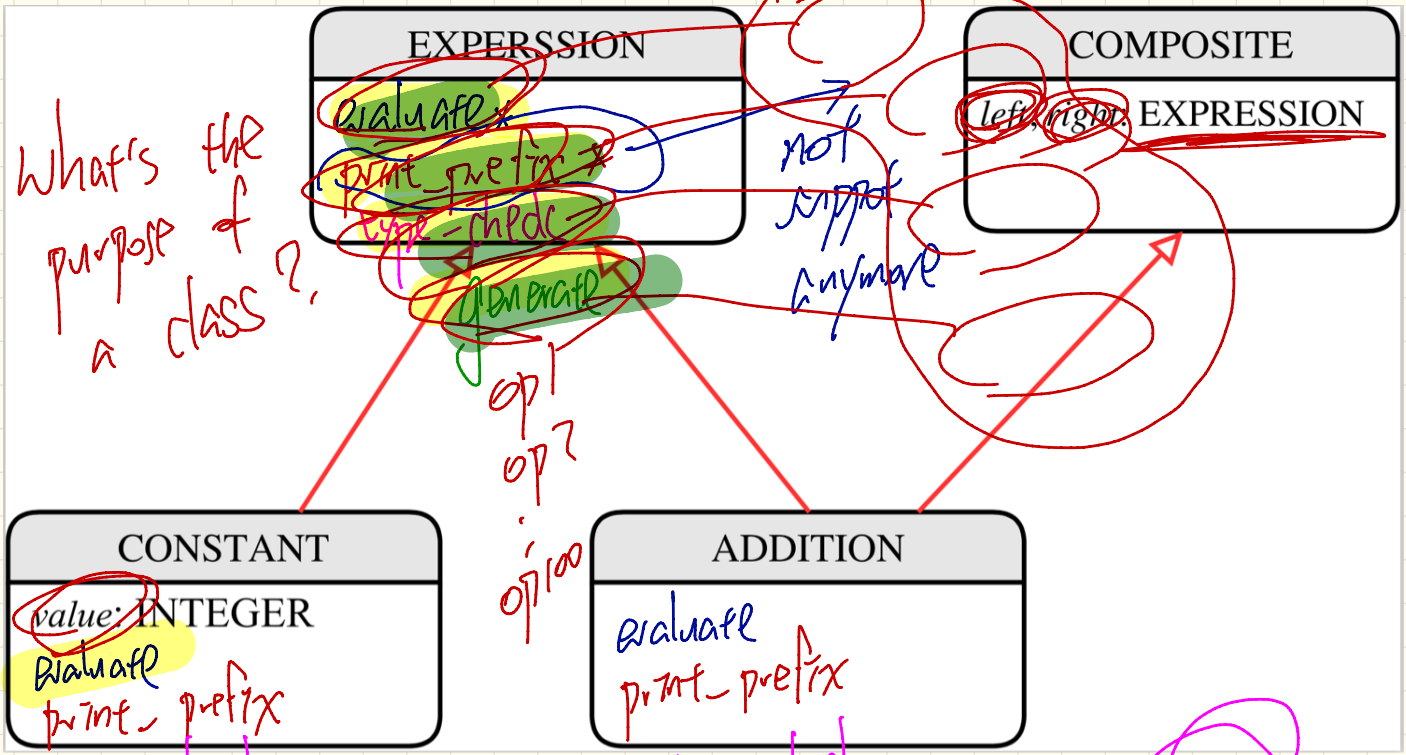

price?
cabinet . price ?
1700
200
p?
chassis
1000
p?
chassis
p?
chassis
500
200
p?
bus
500
chassis
500
drive
price
500
card
price
200
drive
price
500
drive
price
500

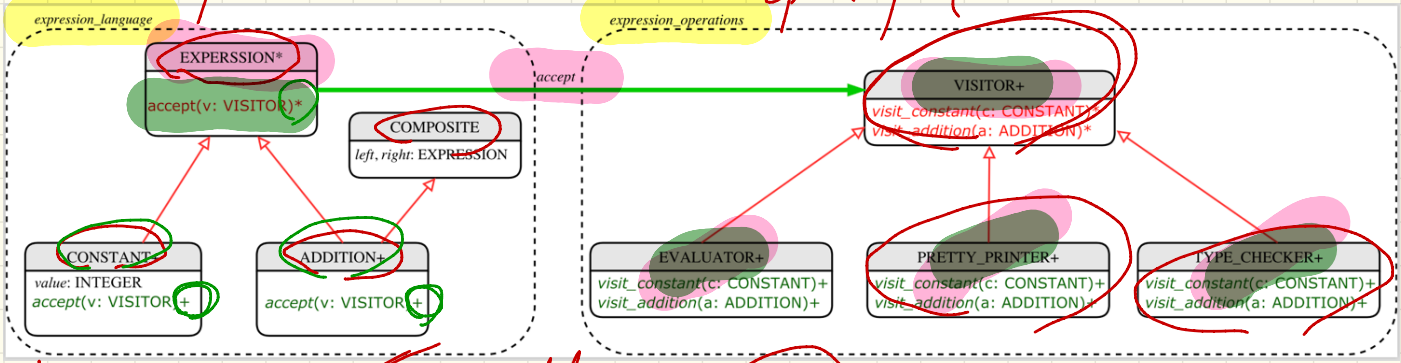

EXPERSSION
left, right
COMPOSITE
left, right: EXPRESSION
operations:
evaluate : 343
print_prefix: "+ 341 2"
print_postfix: "341 2 +"
type_check: true
CONSTANT
value: INTEGER
ADDITION
341 + 2
+
341
2
+
true
3
structure is closed (stable)


EXPERSSION
evaluate *
print_prefix *
type-check
generate
COMPOSITE
left, right: EXPRESSION
CONSTANT
value: INTEGER
evaluate
print_prefix
type-check
generate
ADDITION
evaluate
print_prefix
type-check
generate
What's the purpose of a class?
not support anymore
op1
op2
op100
op1
op2
op100
+
3
4

Lecture 17

Wednesday Nov. 8


EXPERSSION
evaluate
print_prefix
type_check
generate
COMPOSITE
left, right: EXPRESSION
CONSTANT
value: INTEGER
evaluate
print_prefix
type_check
generate
ADDITION
evaluate
print_prefix
type_check
generate
What's the purpose of a class?
not support anymore
op1
op2
:
op100
op1
op2
:
op100
+
3
4



add.accept(v)

add

+

3

4

ADDITION.

accept (v: VISITOR) *

CONSTANT

accept (v: VISITOR)

do

v.visit_constant(Current)

end

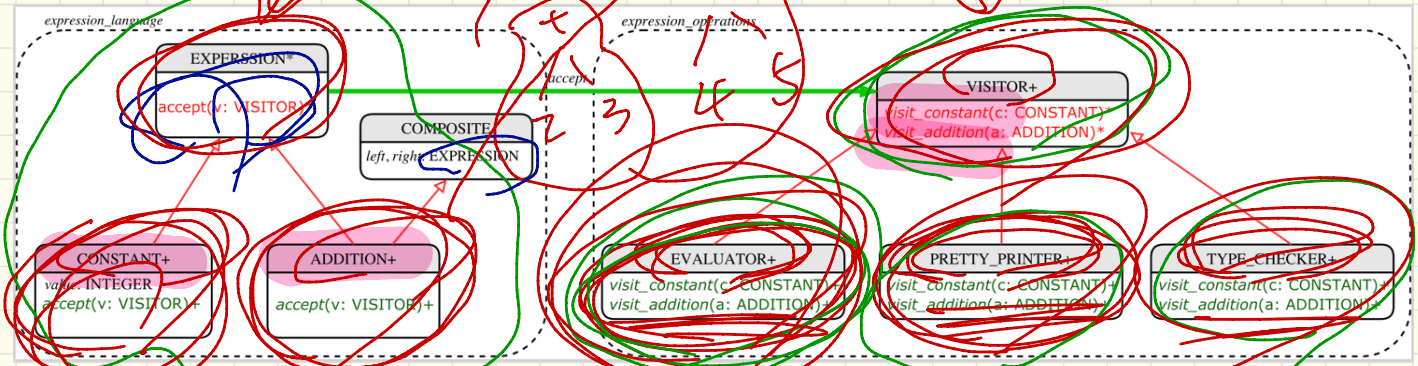
ADDITION

accept (v: VISITOR)

do

v.visit_addition(Current)

end



VISITOR

[visit_constant (c: CONSTANT)
visit_addition (a: ADDITION)] correspond to the list of descendats of EXPRESSION

PRETTY_PRINTER → for pretty printing

visit_constant (c: CONSTANT)
do to.print (c.value) end

(effective) → for pretty printing.

visit_addition (a: ADDITION)
do a.left.accept (current)
to.print ("+")
a.right.accept (Cur.)
end

class PRETTY_PRINTER

visit_addition (a: ADDITION)

:

class EVALUATOR

value: INTEGER

visit_addition (a: ADDITION)

local

left_eval: EVALUATOR

right_eval: EVALUATOR

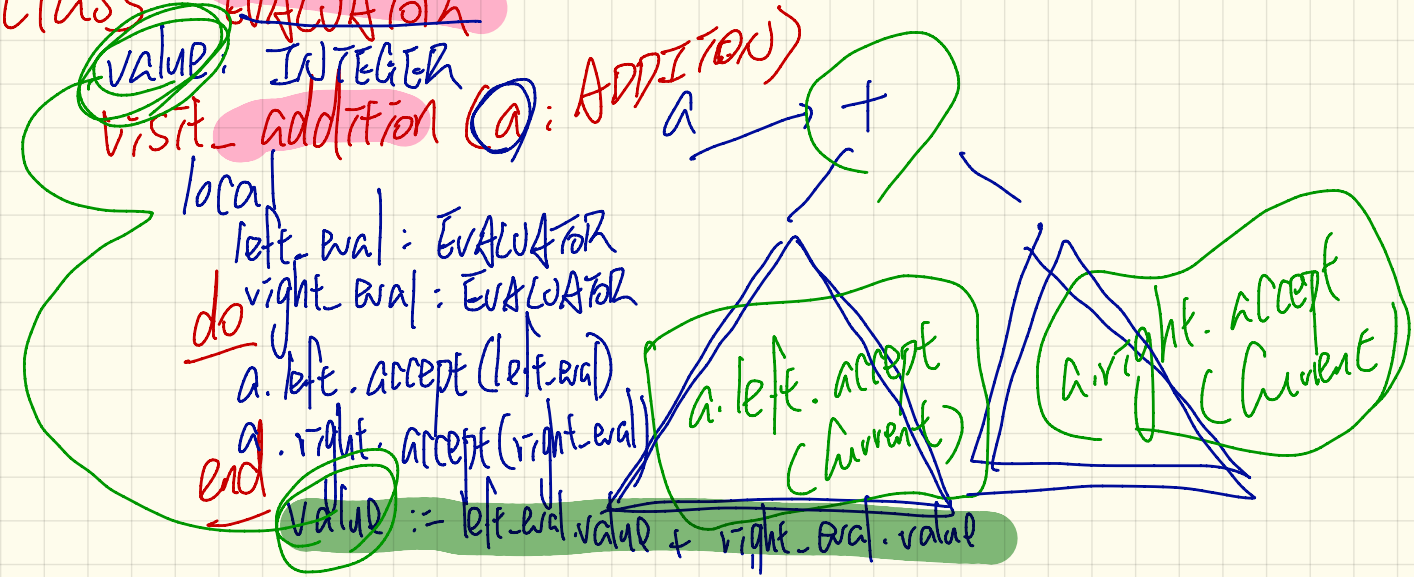
do

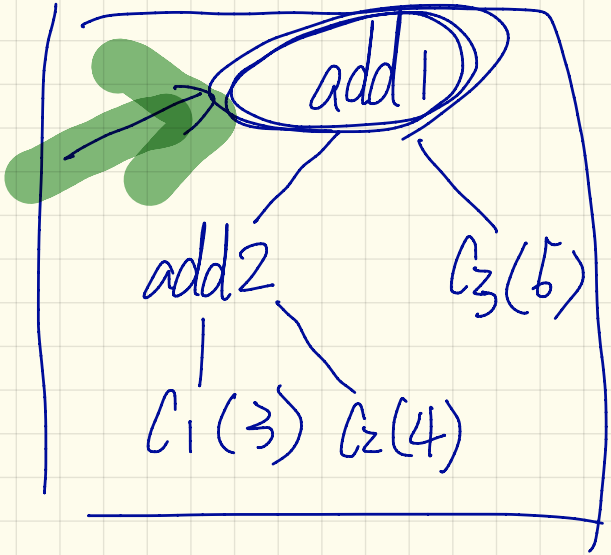
a.left.accept (left_eval)

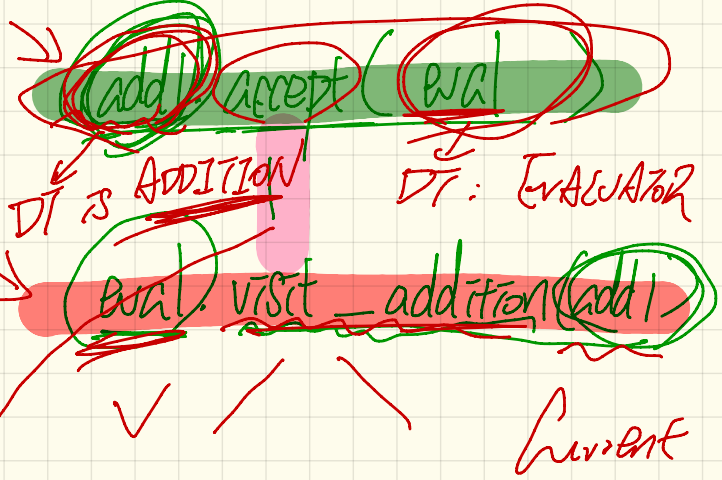
a.right.accept (right_eval)

end

value := left_eval.value + right_eval.value







eval: EVALUATOR

create {EVALUATOR} eval. make

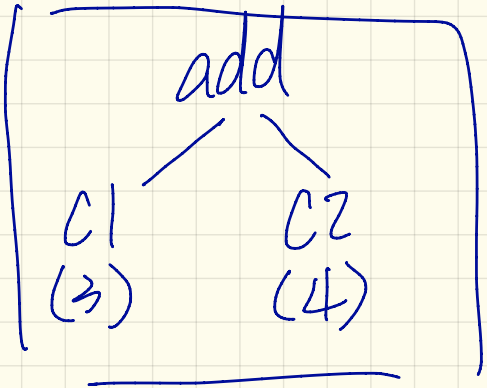
1st patch:
DT of add1 is ADDITION
⇒ version of accept in ADDITION is called.

Double dispatch

add.accept(v)

DT: ADDITION

1st patch: call the version of accept in ADDITION.



add: EXPRESSION

create {ADDITION} add.make

v.visit_addition(add)

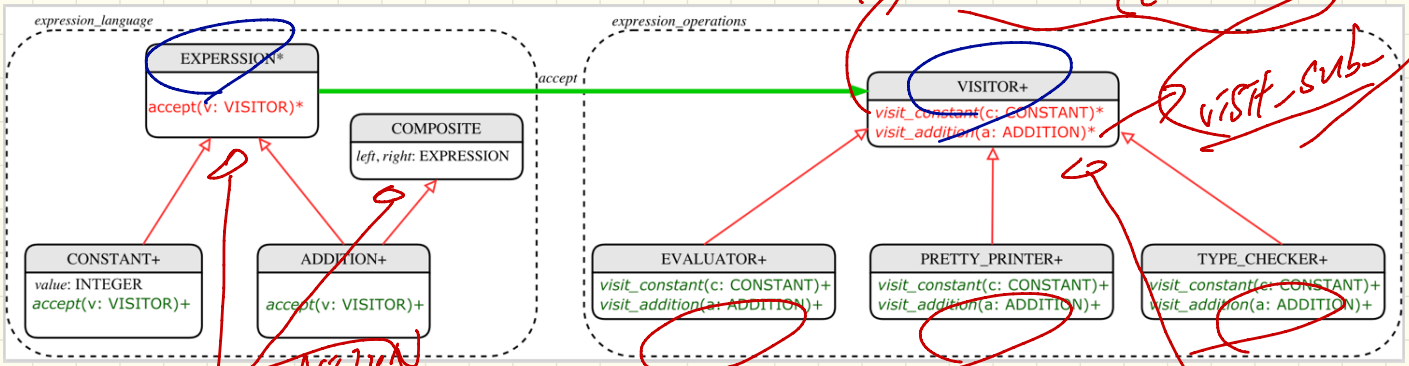
DT: EVALUATOR

2nd patch: call version of visit_addition in EVALUATOR

v: VISITOR

create {EVALUATOR} v.make

add.left.accept(left_eval) add.right.accept(right_eval)



visit_expression (e:

visit_sub-

SUBTRACTION

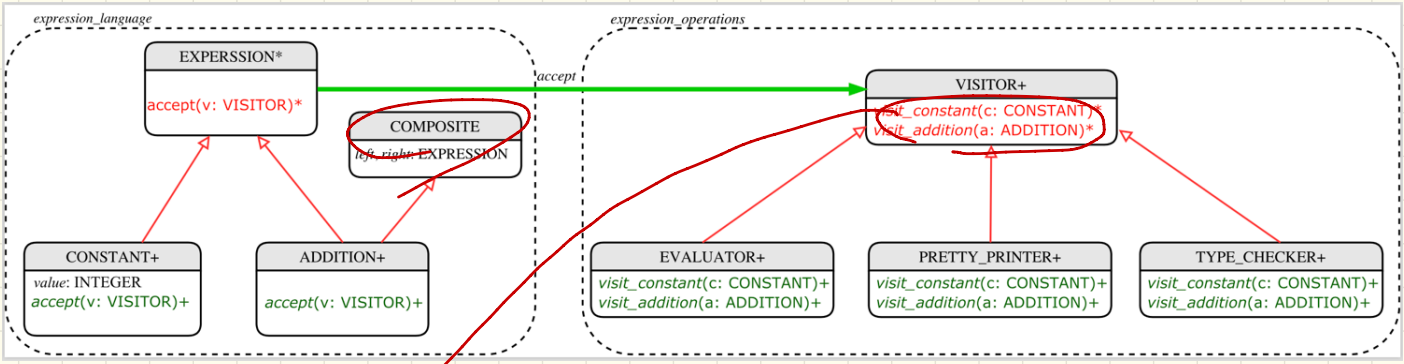
Extensions ?

visit_expression

SIMPLIFIER

- add a new EXPRESSION (e.g. SUBTRACTION)

- add a new VISITOR (e.g. SIMPLIFIER)



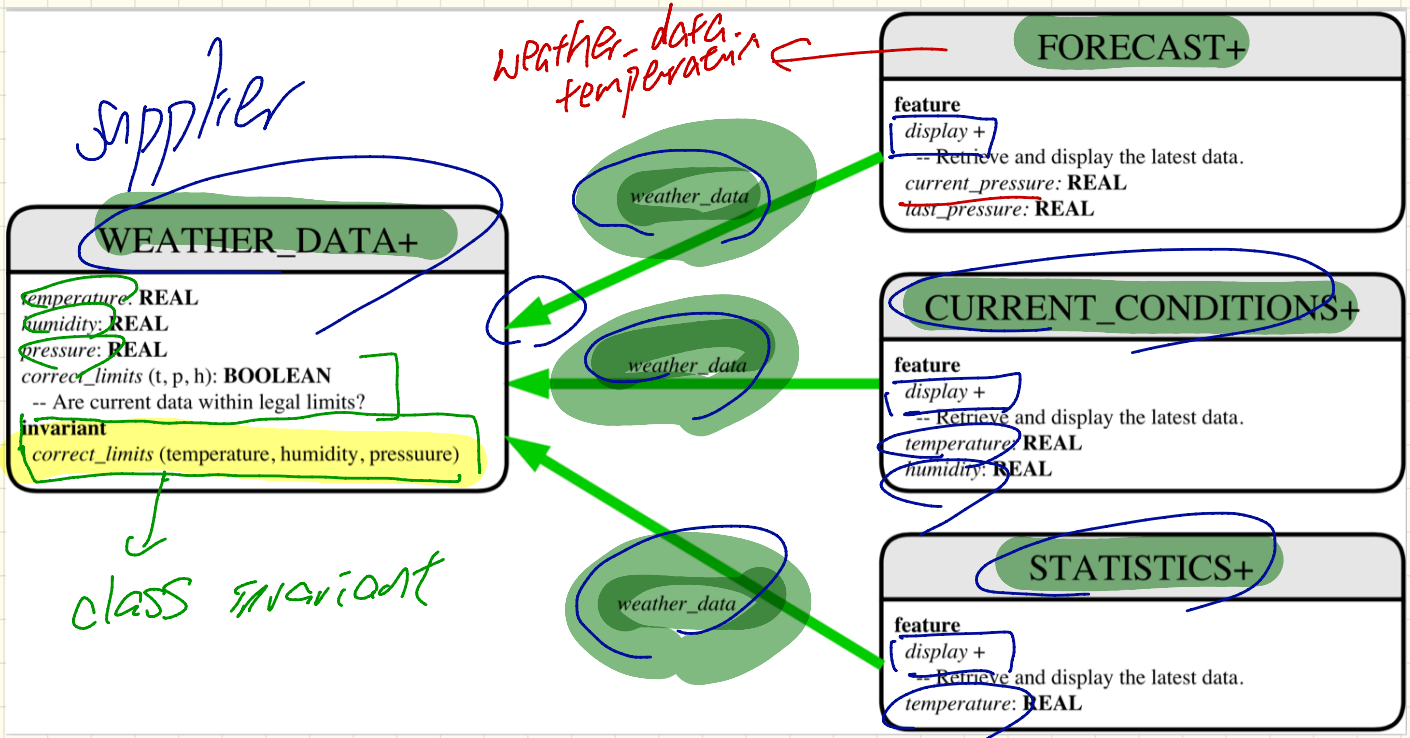
visit_expression(e: EXPRESSION) X

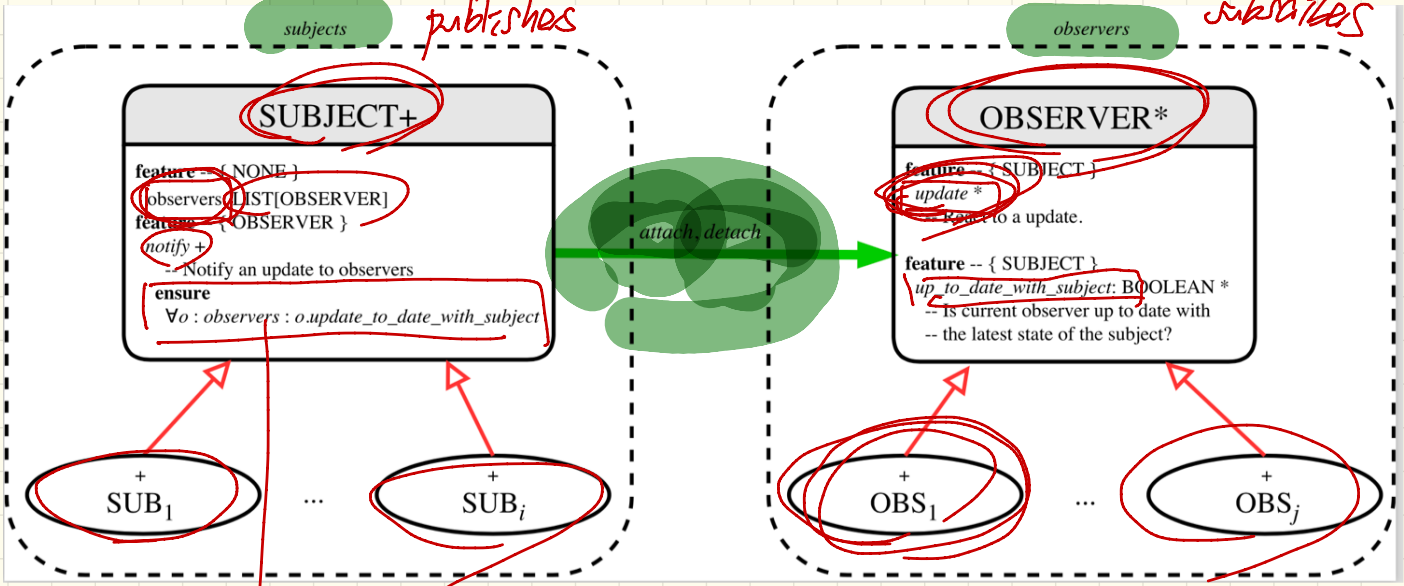
EVALUATOR

visit_expression(e: EXPRESSION)
do
[e.left]
end

Lecture 18

Monday Nov. 13


supplier
weather_data
temperature
FORECAST+
feature
display +
-- Retrieve and display the latest data.
current_pressure: REAL
last_pressure: REAL
weather_data
WEATHER_DATA+
temperature: REAL
humidity: REAL
pressure: REAL
correct_limits (t, p, h): BOOLEAN
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)
weather_data
CURRENT_CONDITIONS+
feature
display +
-- Retrieve and display the latest data.
temperature: REAL
humidity: REAL
class invariant
weather_data
STATISTICS+
feature
display +
-- Retrieve and display the latest data.
temperature: REAL
clients

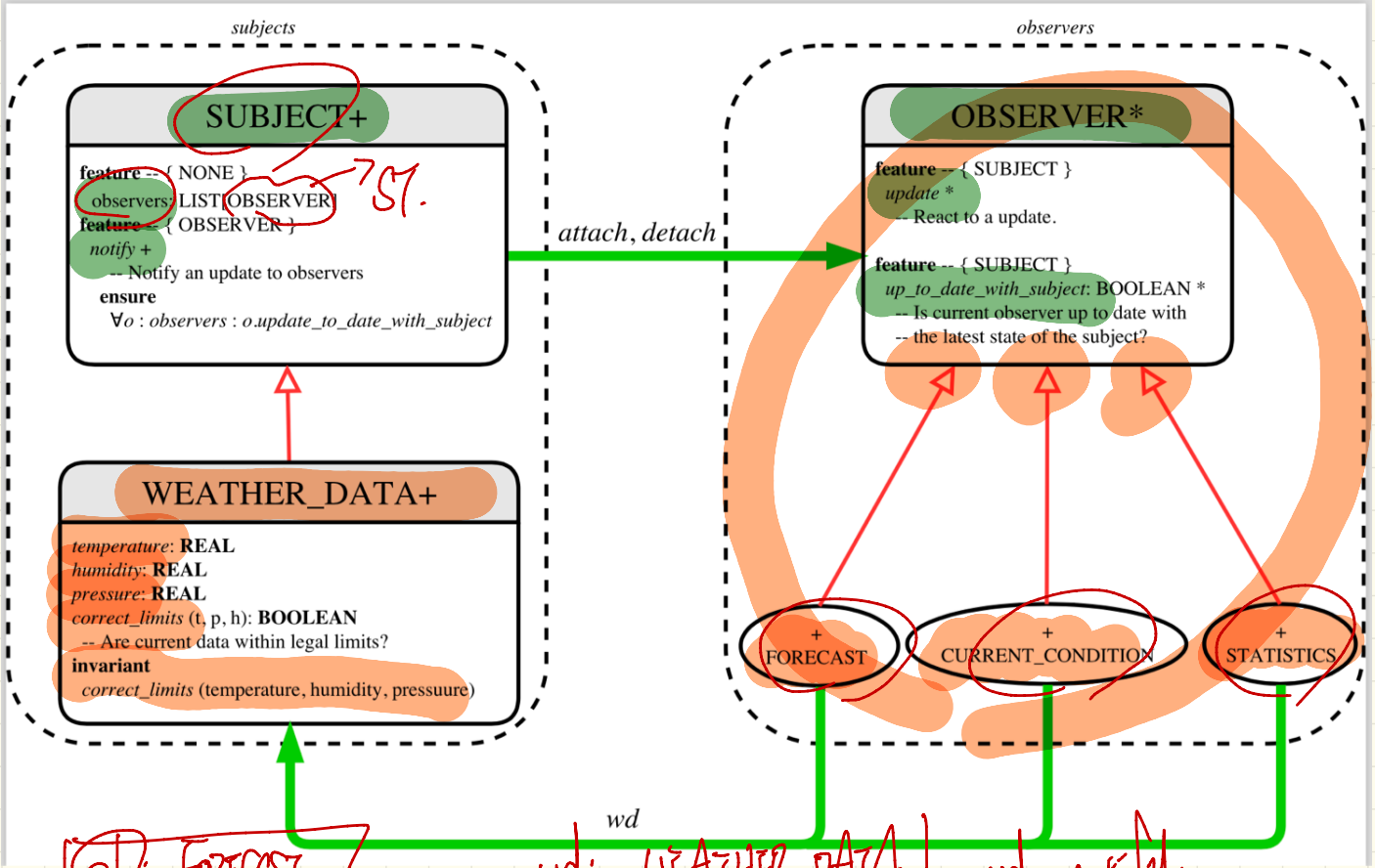


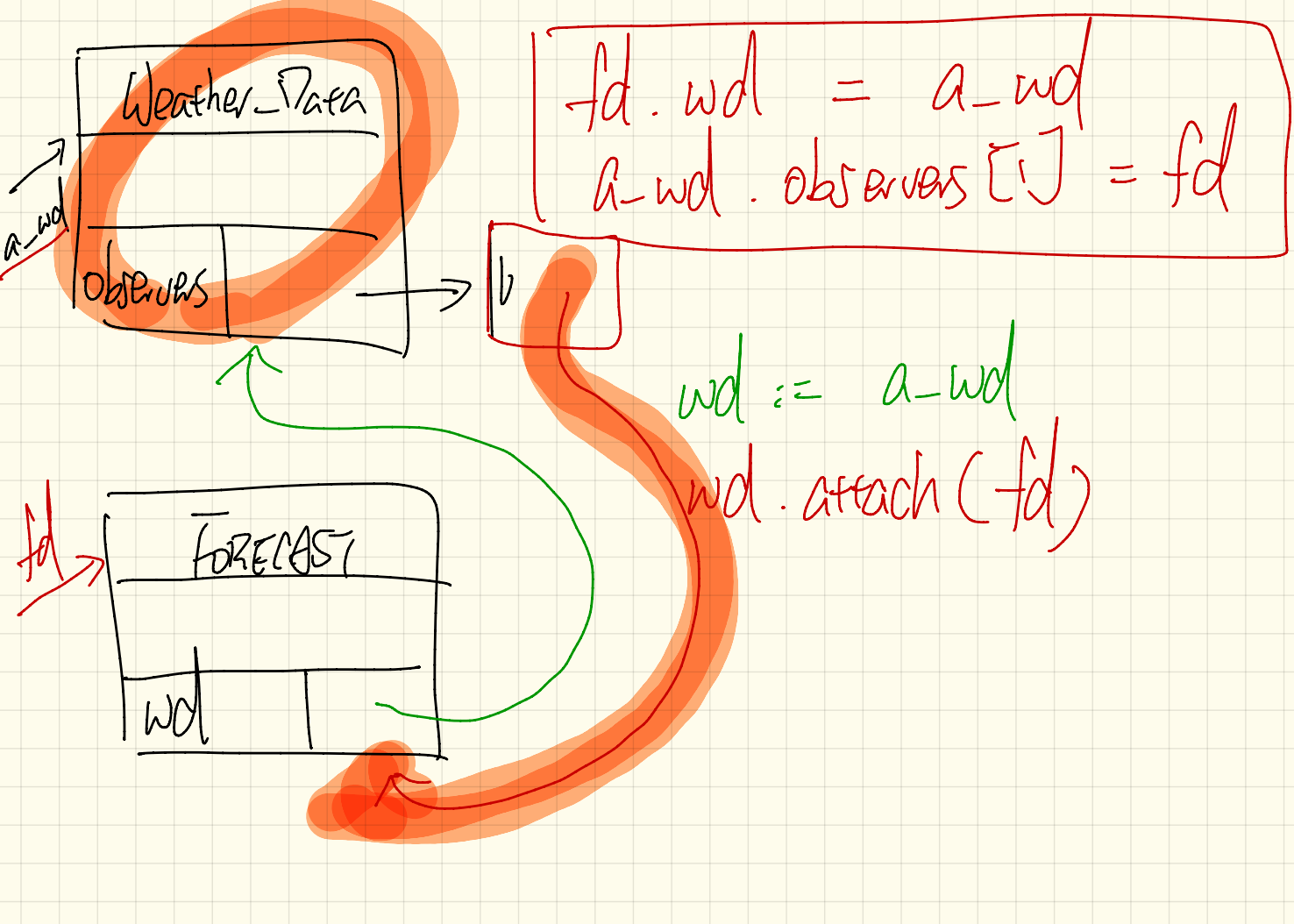
notify

ensure

∀o : observers : o.up_to_date_with_subject

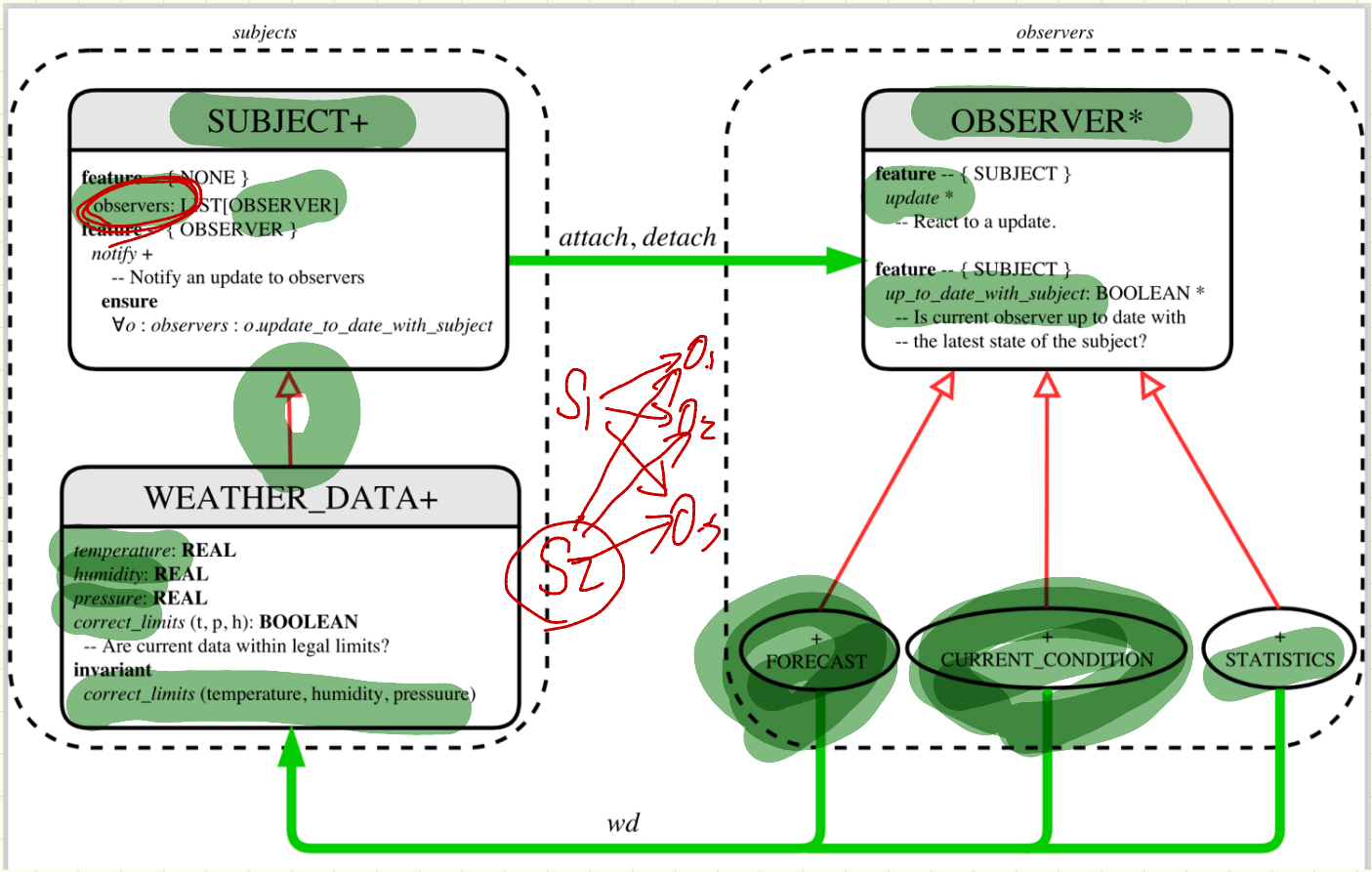
?

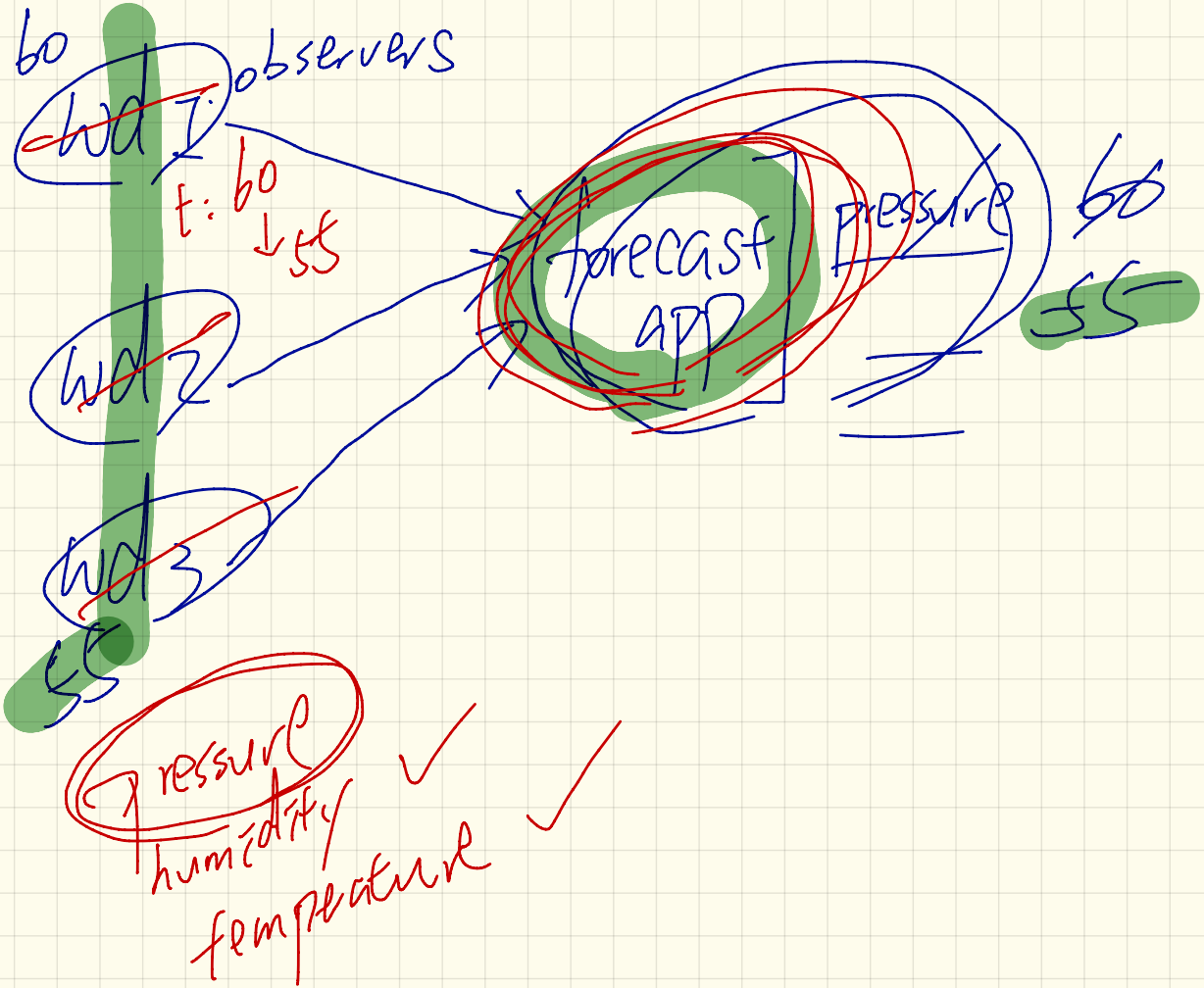

subjects
observers
SUBJECT+
feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
∀o : observers : o.update_to_date_with_subject
attach, detach
OBSERVER*
feature -- { SUBJECT }
update *
-- React to a update.
feature -- { SUBJECT }
up_to_date_with_subject: BOOLEAN *
-- Is current observer up to date with
-- the latest state of the subject?
WEATHER_DATA+
temperature: REAL
humidity: REAL
pressure: REAL
correct_limits (t, p, h): BOOLEAN
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)
+
FORECAST
+
CURRENT_CONDITION
+
STATISTICS
wd

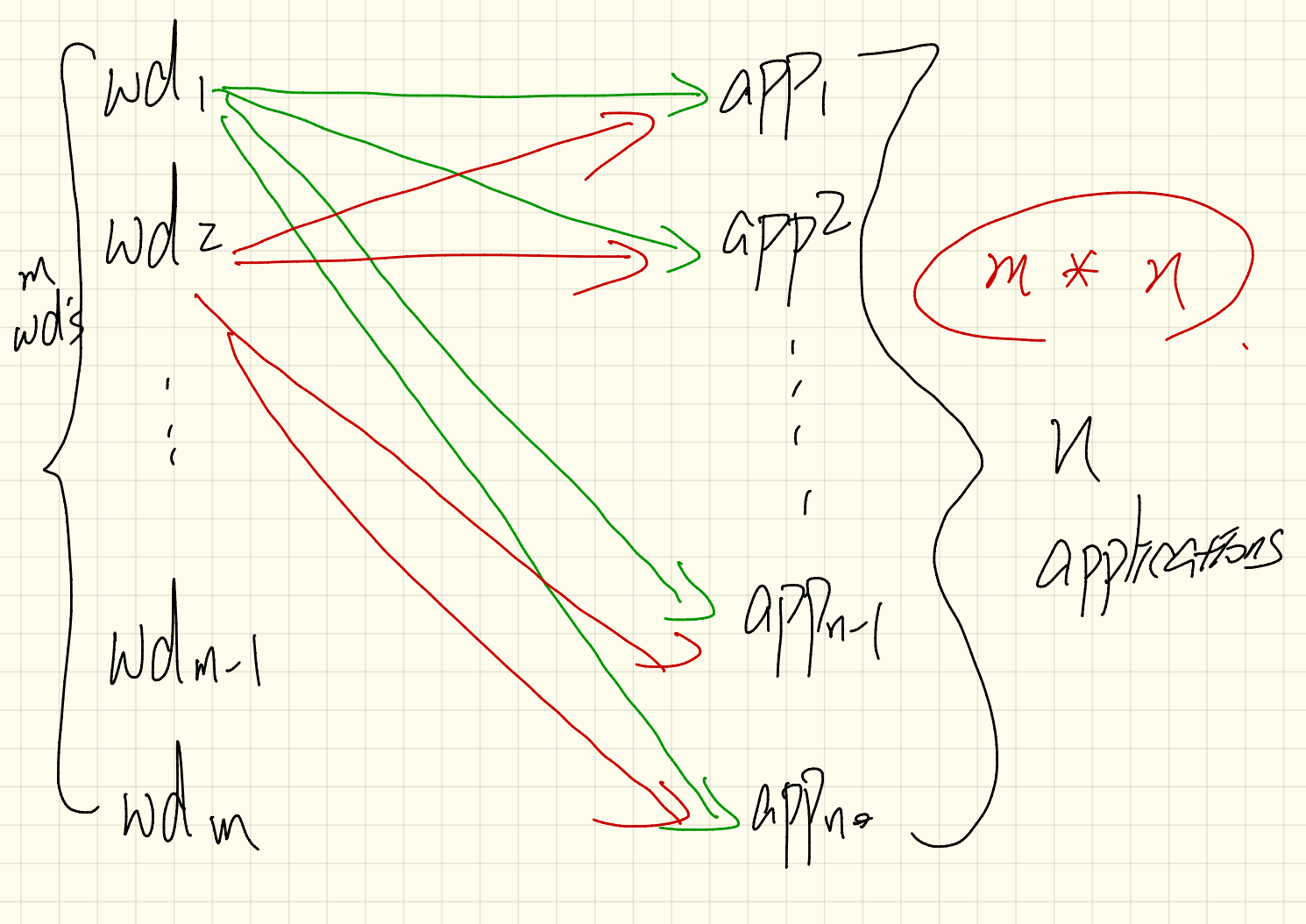

Weather_Data
a_wd
observers
fd
FORECAST
wd
fd.wd = a_wd
a_wd.observers[·] = fd
wd := a_wd
wd.attach(fd)

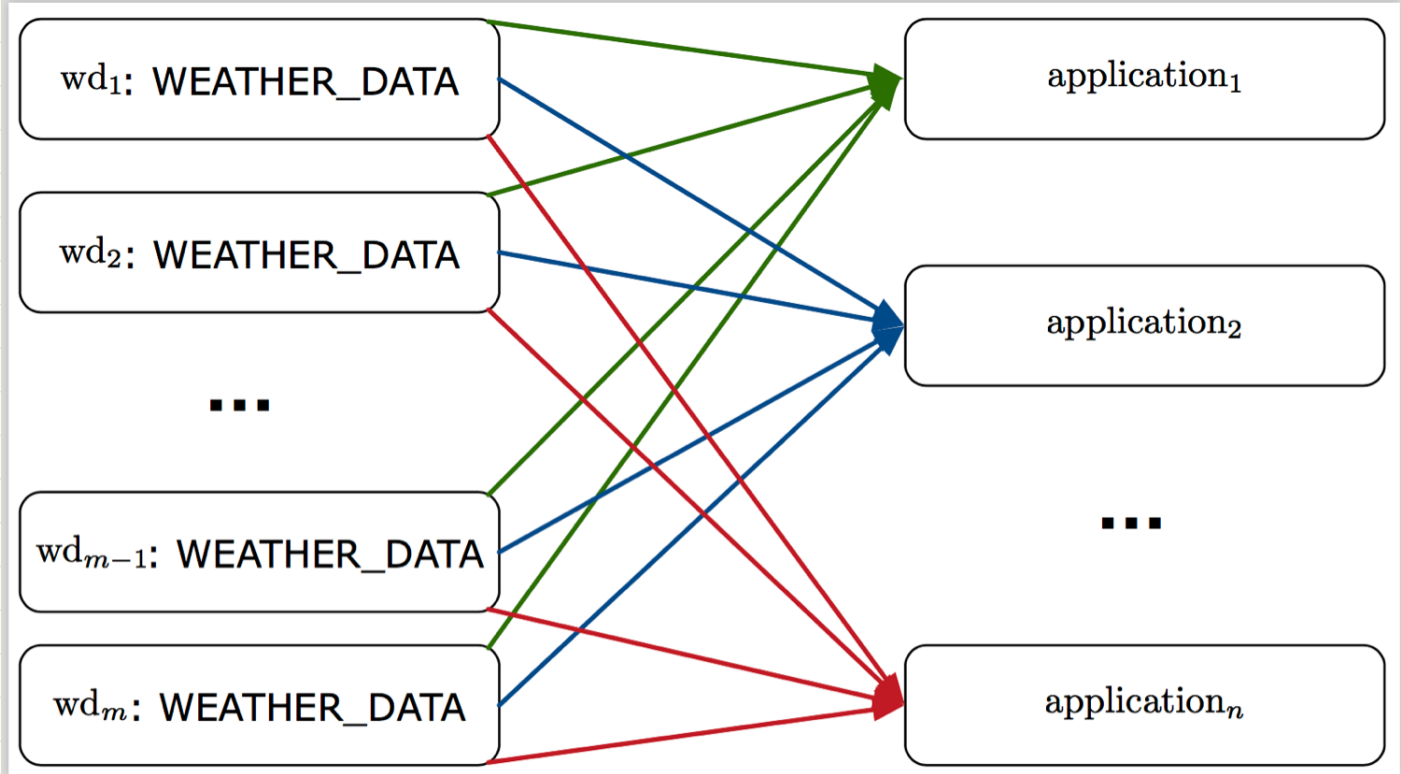
Lecture 19

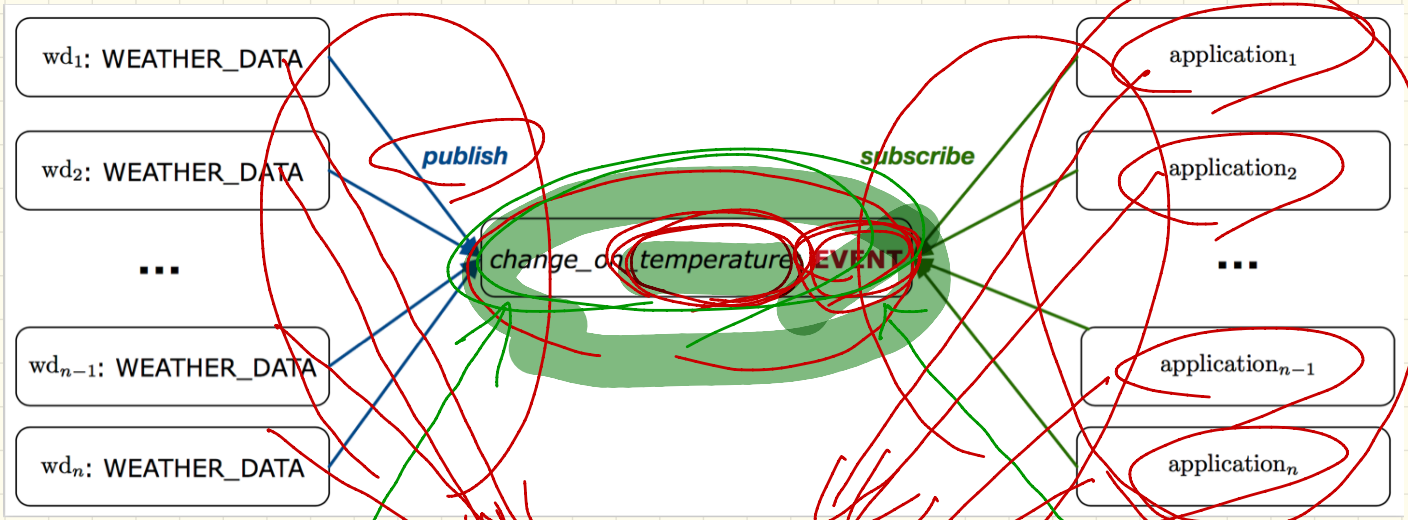
Wednesday Nov. 15

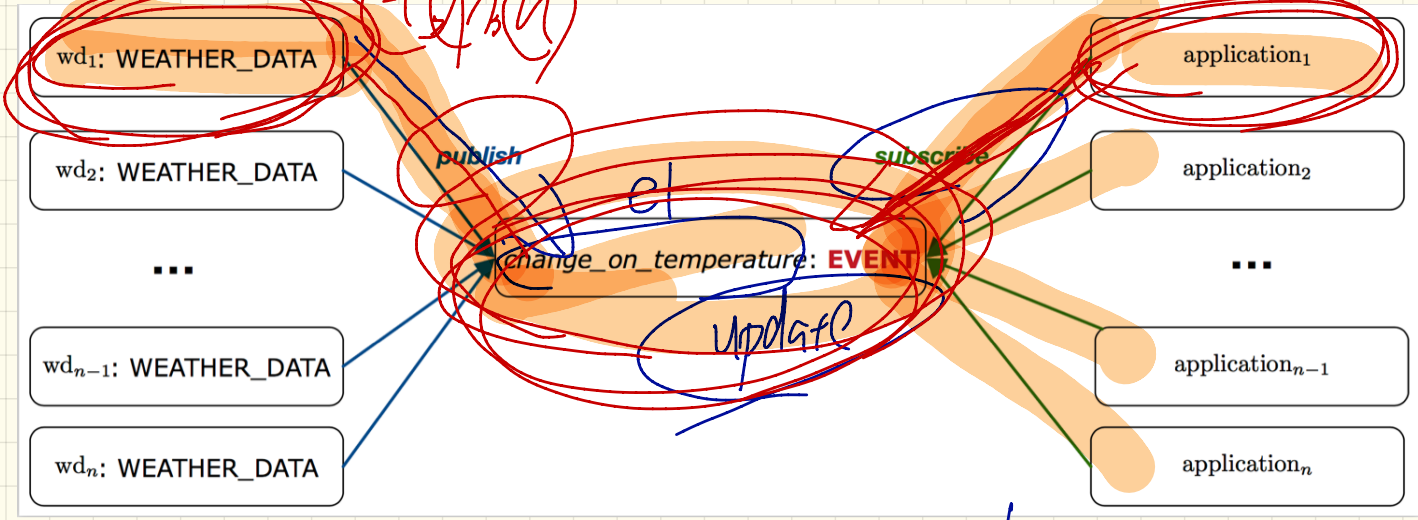

subjects
observers
SUBJECT+
feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
∀o : observers : o.update_to_date_with_subject
attach, detach
OBSERVER*
feature -- { SUBJECT }
update *
-- React to a update.
feature -- { SUBJECT }
up_to_date_with_subject: BOOLEAN *
-- Is current observer up to date with
-- the latest state of the subject?
WEATHER_DATA+
temperature: REAL
humidity: REAL
pressure: REAL
correct_limits (t, p, h): BOOLEAN
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)
S1
S2
O1
O2
O3
+
FORECAST
+
CURRENT_CONDITION
+
STATISTICS
wd


bo
observers
wd 1
wd 2
wd 3
t: bo
ss
forecast
app
pressure
bo
ss
pressure
humidity
temperature


wd1
wd2
wdm-1
wdm
m wd's
app1
app2
appn-1
appn
n applications
m * n

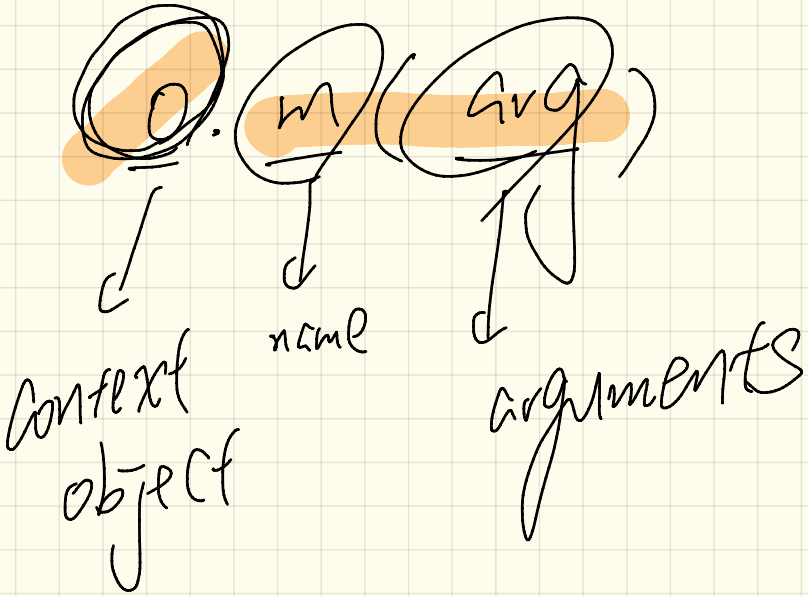

wd$_1$: WEATHER_DATA
wd$_2$: WEATHER_DATA
...
wd$_{m-1}$: WEATHER_DATA
wd$_m$: WEATHER_DATA
application$_1$
application$_2$
...
application$_n$


wd1: WEATHER_DATA
wd2: WEATHER_DATA
...
wdn−1: WEATHER_DATA
wdn: WEATHER_DATA
publish
change_on(temperature):EVENT
subscribe
application1
application2
...
applicationn−1
applicationn

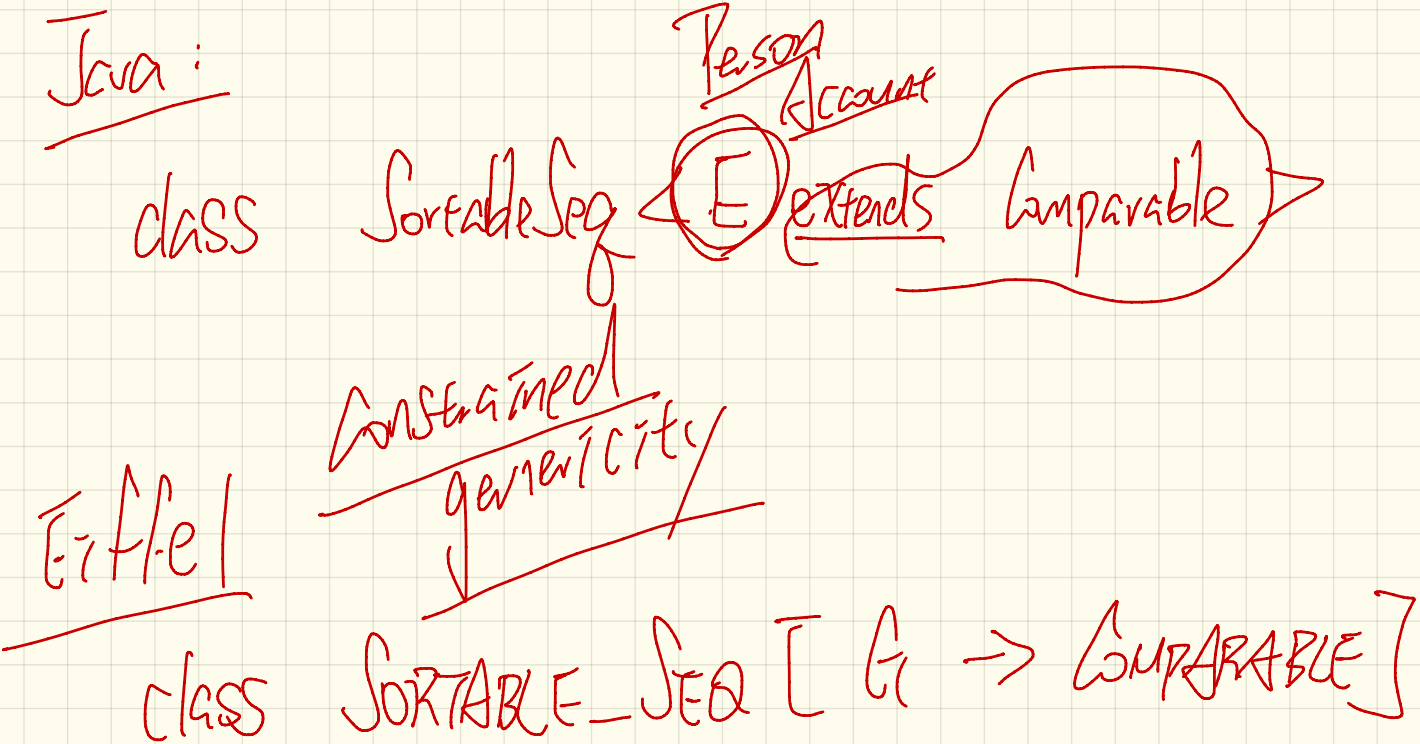


app1 subscribe to e1 (attach pointer to some update function)

When there's an update from wd1, it will publish to e1, calling the saved update op.



listener.action(args);


Java :
class SortableSeq < E extends Comparable
Person
Account
Constrained genericity
Eiffel
class SORTABLE_SEQ [G -> COMPARABLE]

t_1: TUPLE

t_2: TUPLE [REAL]

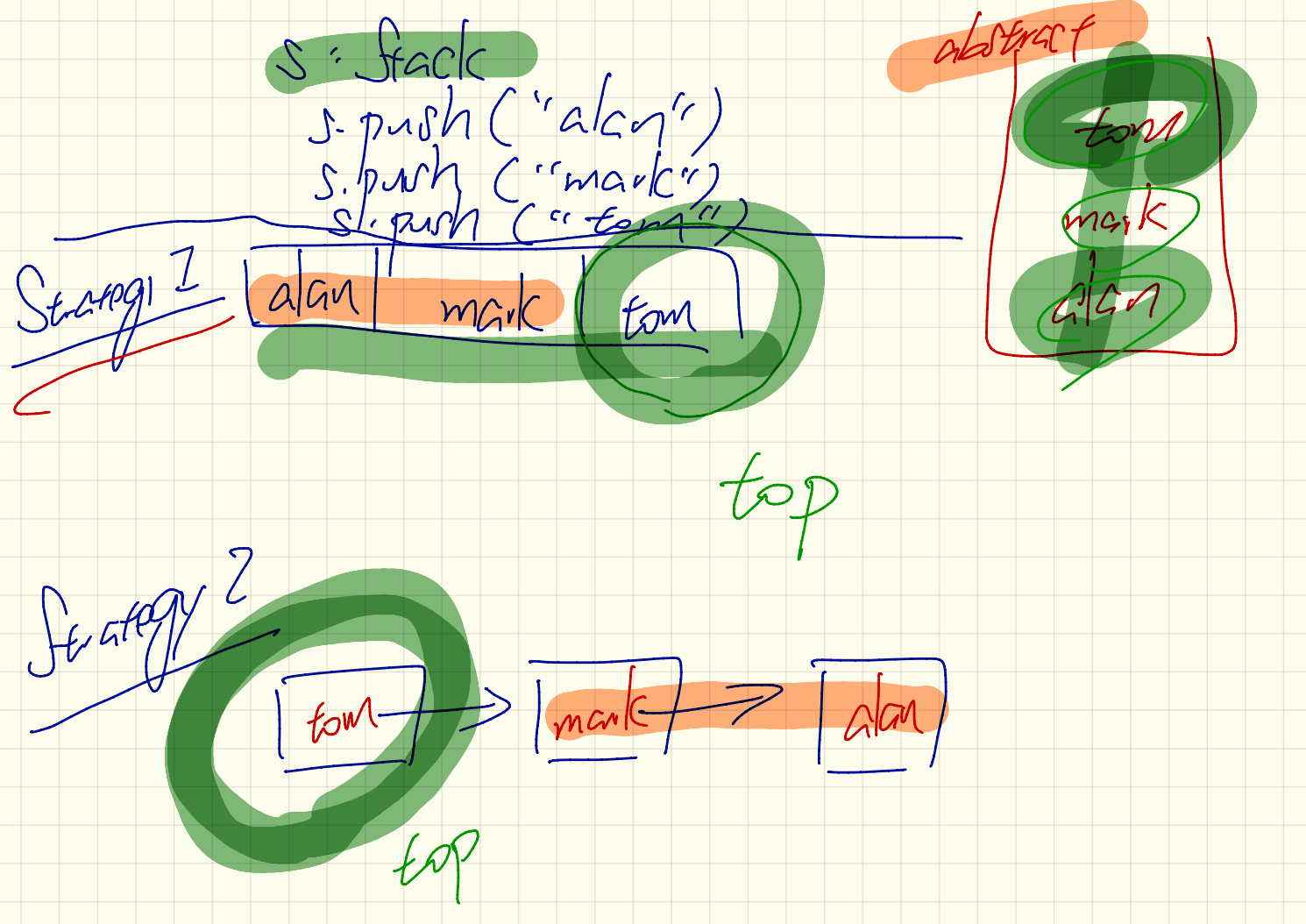
t_3: TUPLE [REAL; REAL]

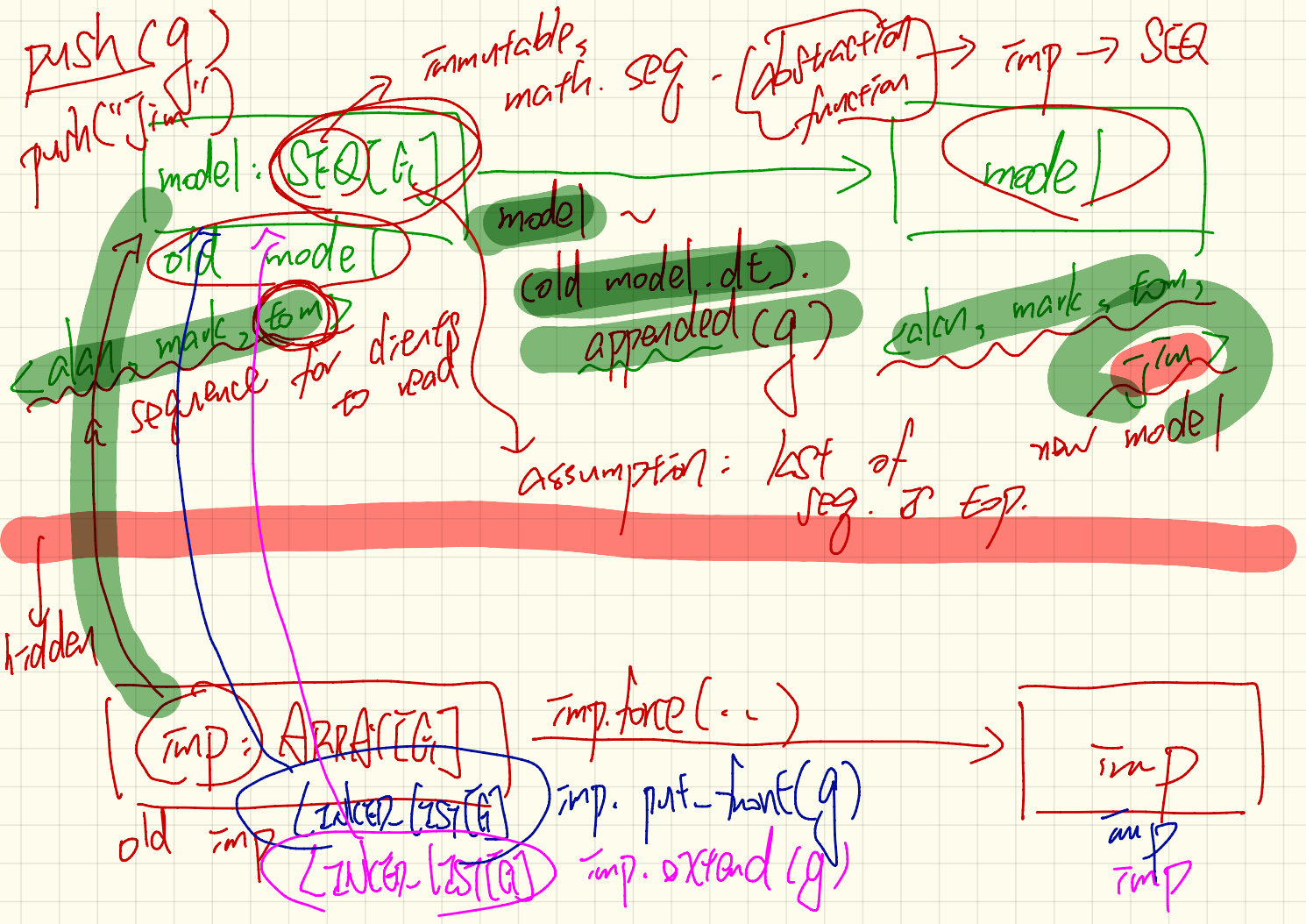
```
class Point {
  int x;
  void change_x(int nx) {
    x := nx;
  }
}
```

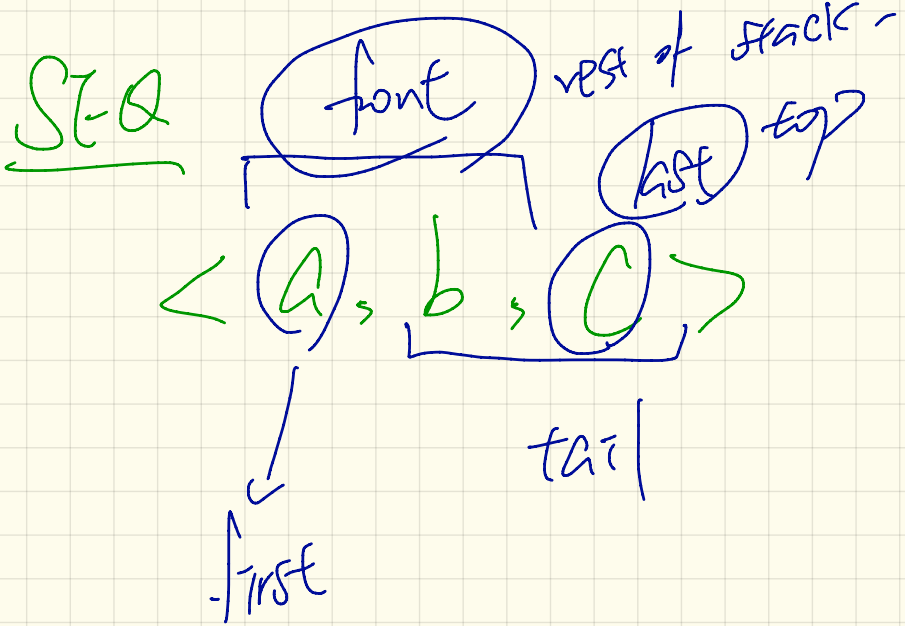
this.x

Lecture 20

Monday Nov. 20


S : Stack
S.push("alan")
S.push("mark")
S.push("tom")
Strategy 1
alan
mark
tom
top
abstract
tom
mark
alan
Strategy 2
tom
mark
alan
top


push(g)
push("Jim")
immutable, math. seq
abstraction function
imp → SEQ
model : SEQ[G]
old model
model
model ~ (old model.dt).appended(g)
assumption: last of seq. & esp.
new model
clean, mark, from, sequence for clients to read
hidden
imp : ARRAY[G]
imp.force(...)
imp.put_front(g)
imp.extend(g)
old imp
imp
imp


Stack
front
rest of stack
top
last
a
b
c
tail
first

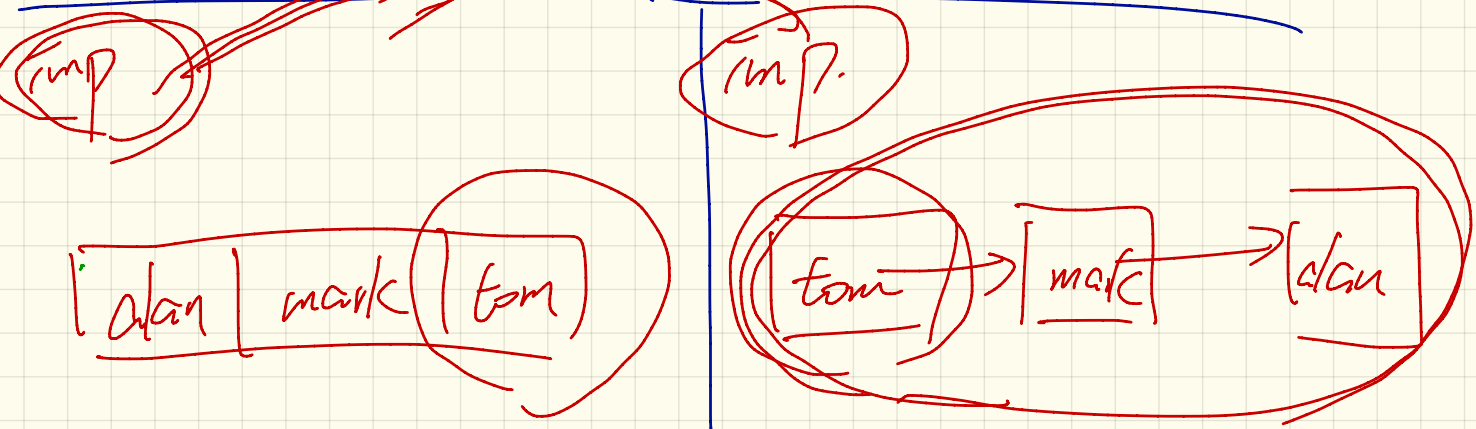
Strategy 1

tom
mark
alan

Strategy 2

create an abstraction of STACK ADT as a sequence, where last item is top.

model: <alan, mak, tom>



prepend $\approx$ addFirst

append $\approx$ addLast

$$\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$$

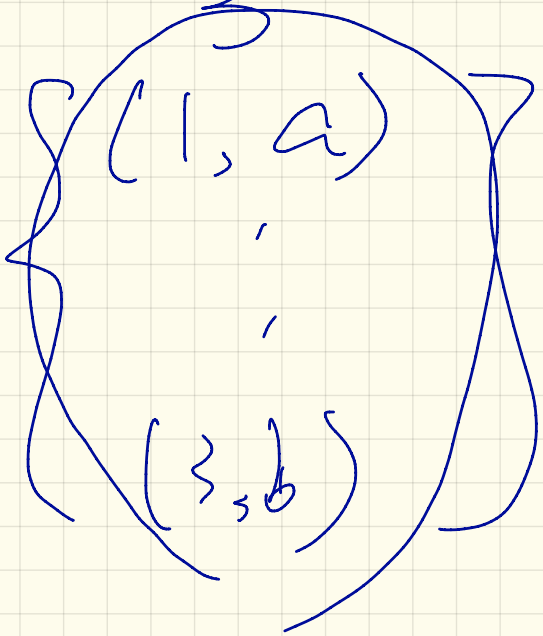
$$2^{|\{a,b\}|}$$

$$\{a,b\} \times \{x,y\} \times \{\varepsilon,\sigma\}$$

$$= \{(a,x),(a,y),(b,x),(b,y)\}$$

$\underbrace{\{1, 2, 3\}}_{S} \times \underbrace{\{a, b\}}_{T}$

$3 \qquad 2$



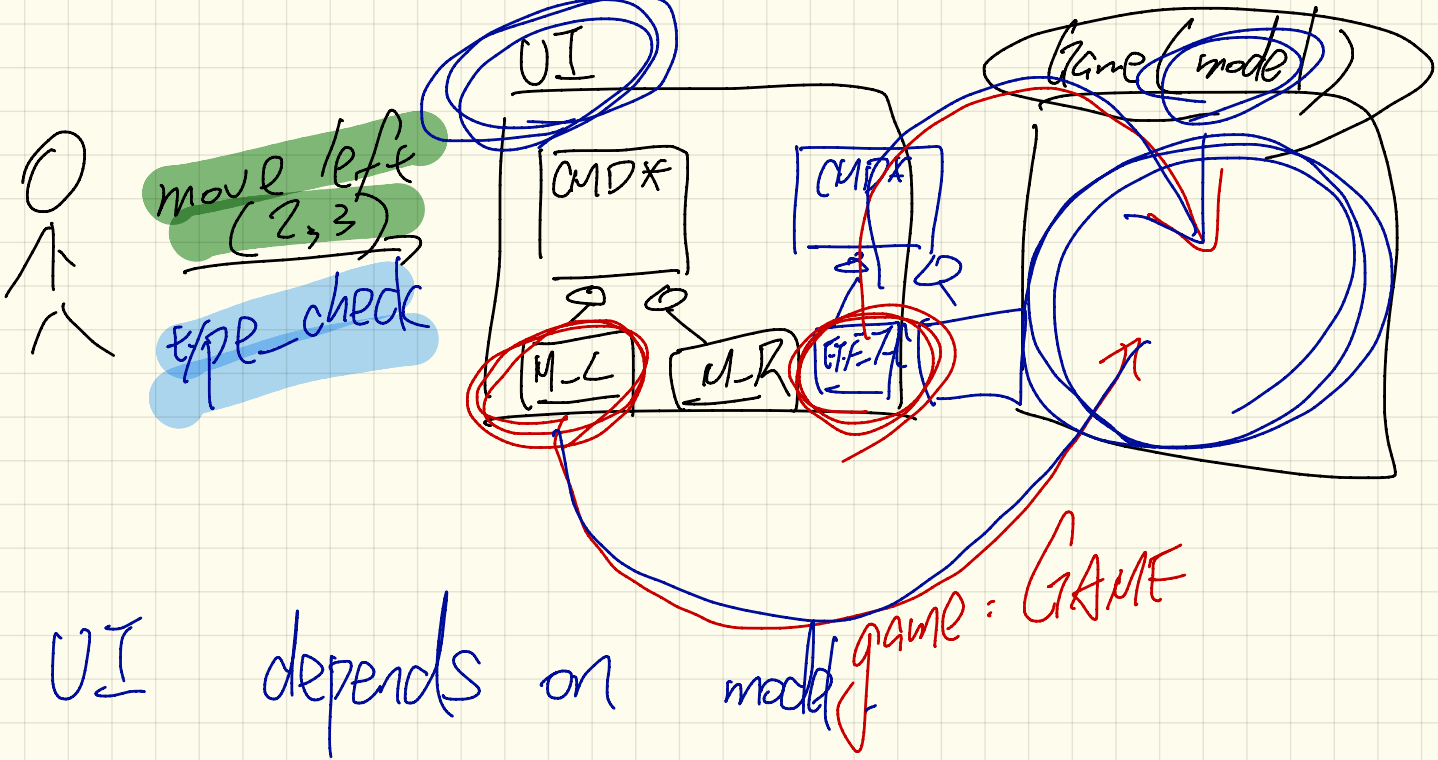
6 pairs.

$S \times T$ → largest relation for S and T

$\emptyset$ → Smallest relation.

Lecture 21

Monday Nov. 27



UI depends on model

Model doesn't depen on UI.

FIT / ??

- test cases not written in a programming language.

- acceptance testing

- imp./design is not relevant.

clients

act.txt
aft.txt

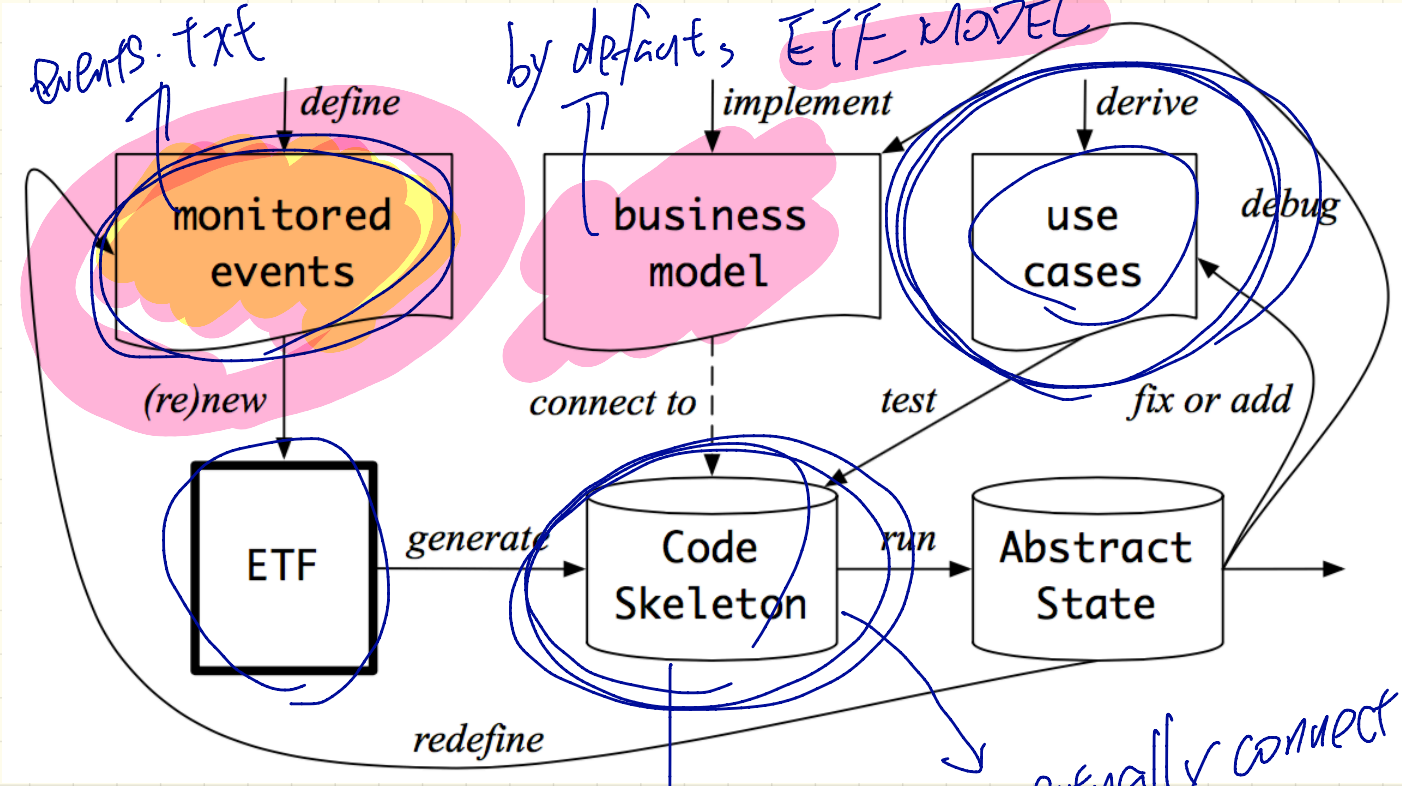
```
new ("Jane")
deposit ("Jane", 200)
```

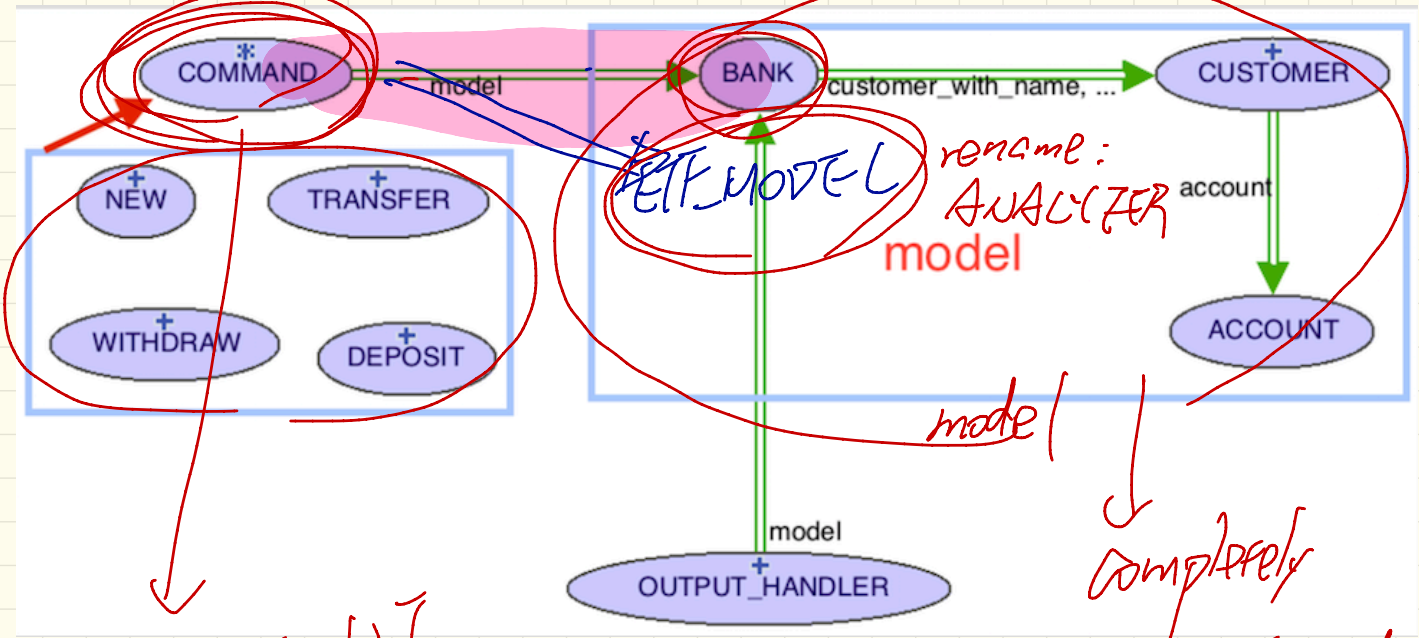
FSpec / JUnit

- test cases written in a prog. language.

- Unit testing (classes/features/methods)

- imp. is relevant.

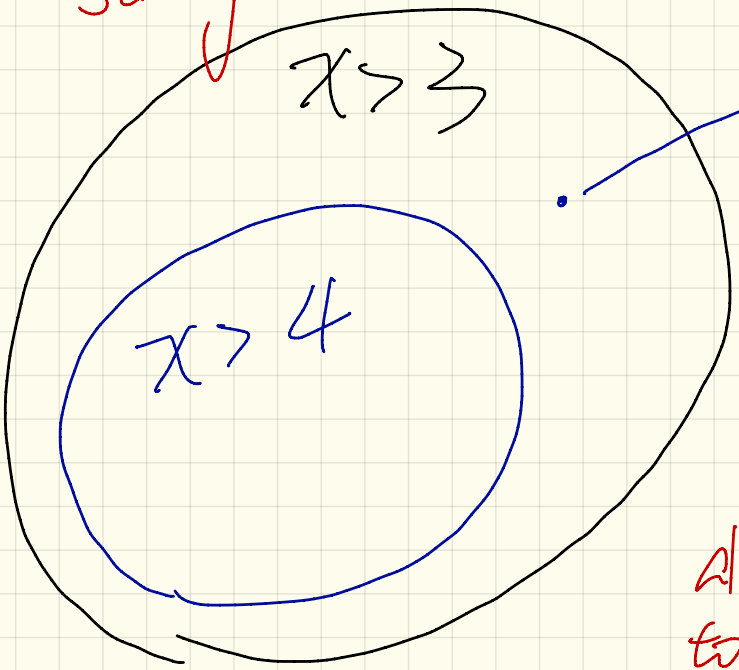

events.txt
define
by default, ETF_MODEL
implement
derive
monitored events
business model
use cases
debug
(re)new
connect to
test
fix or add
ETF
generate
Code Skeleton
run
Abstract State
redefine
starter project. eventually connect to ur own desing/imp.


COMMAND
model
BANK
customer_with_name, ...
CUSTOMER
NEW
TRANSFER
WITHDRAW
DEPOSIT
SELF_MODEL
rename:
ANALYZER
model
account
ACCOUNT
model
model
OUTPUT_HANDLER
abstract UI
(stable)
completely
up to d.

$x > 4 \Rightarrow x > 3$

stronger

weaker



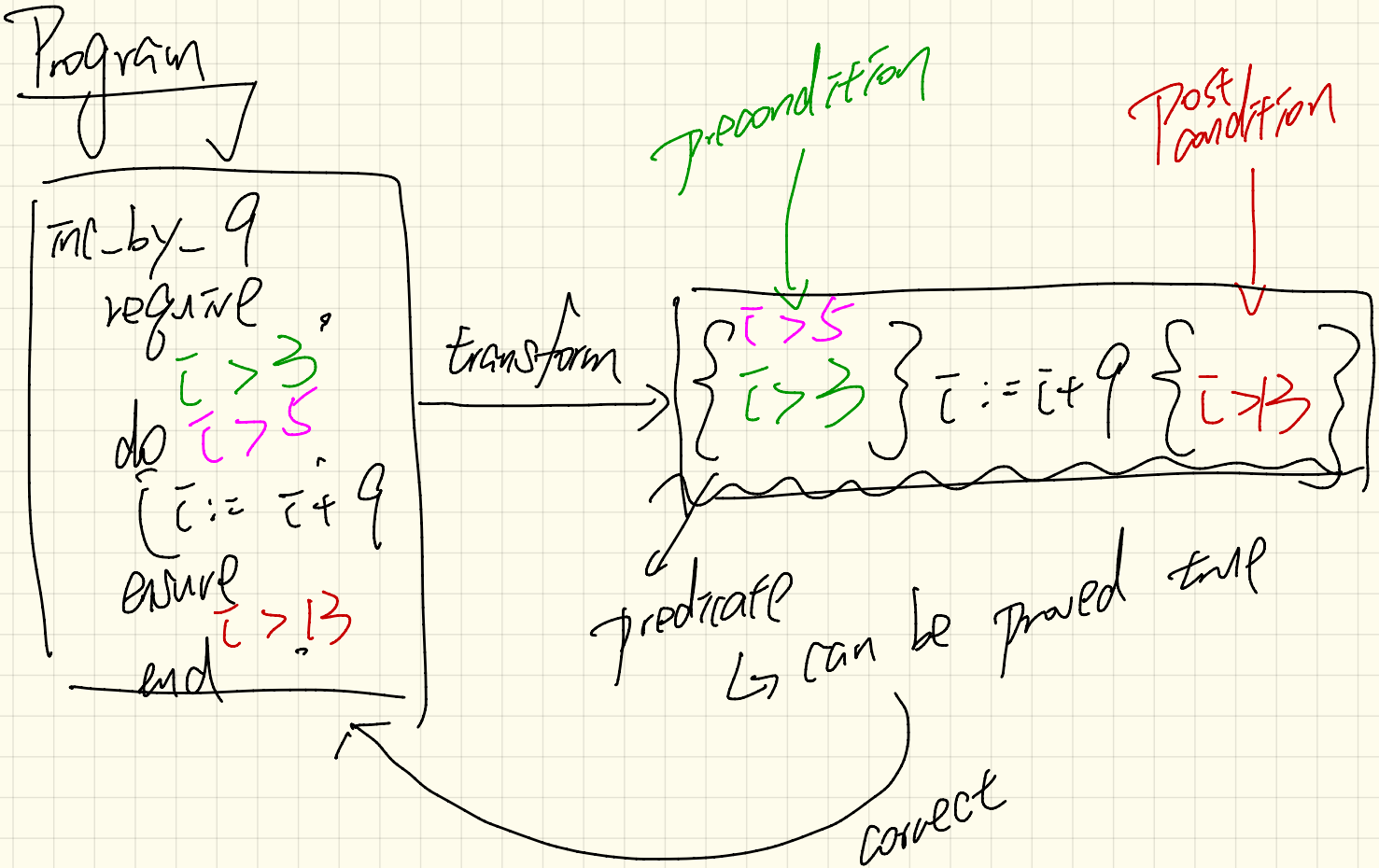
e.g. $\sqrt{4}$

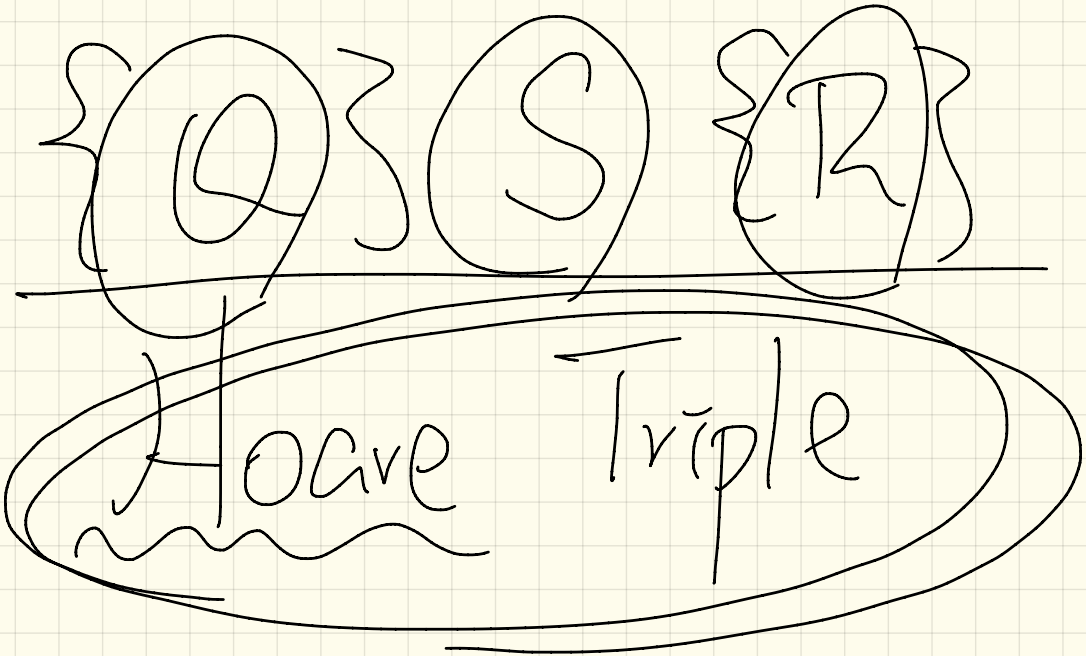
allows more values to be included

$x > 3$ is weaker than $x > 4$

$x > 4$ is stronger than $x > 3$

allows fewer values to be included in the set.


Program
inc_by_9
require
i > 3
i > 5
do
i := i + 9
ensure
i > 13
end
transform
precondition
Post condition
i > 5
i > 3
i := i + 9
i > 13
predicate
↳ can be proved true
correct


{Q} S {R}
Hoare Triple

How to prove $\{Q\}\ S\ \{R\}$?

1. Calculate the wp for S to establish R.

$wp(S, R)$

4 loops

1. $x := e$
2. if B then S_1 else S_2
3. $S_1 ; S_2$

2. Prove that Q is no weaker than $wp(S, R)$:

$$Q \Rightarrow wp(S, R)$$

Define wp

wp(program statement, assertion)

wp($x := e$, R) = ?

wp(if B then S_1 else S_2 end, R) = ?

wp($S_1 ; S_2$, R) = ?

$$wp(x := x_0 + 1;\ x > x_0)$$

$$= x > x_0\,[x := x_0 + 1]$$

$$= x_0 + 1 > x_0$$

$$= 1 > 0 \equiv \text{True}$$

Lecture 22

Wednesday Nov. 29

Program:

```
inc_by_9
  require  i > 3
  do       i > 5
    i := i + 9
  ensure
    i > 13
  end
```

Is this correct?

4. $i > 5 \Rightarrow i > 4$ ☐ (No e.g. $i = 4$; $i > 3$)

existing ✓ (Proved) wp ☐

1. Translate prog. into Hoare Triple

$\{ i > 5 \}$ $i := i + 9$ $\{ i > 13 \}$ ($i > 3$)

2. Prove existing precond. $i > 5$ is no weaker than $i > 3$

$wp(i := i + 9, i > 13)$

3. Calculate $wp(i := i + 9, i > 13)$

$wp(i := i + 9, i > 13)$ (post-state)

$= \{$ wp rule for assign. $\}$

$i > 13 \; [i := i_0 + 9]$

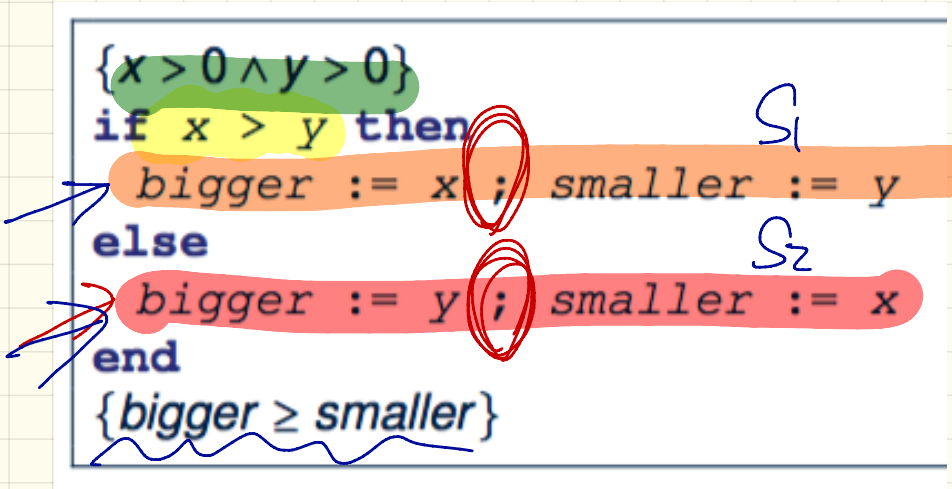
post-state value

$= \{$ subs. $\}$

$i_0 + 9 > 13$

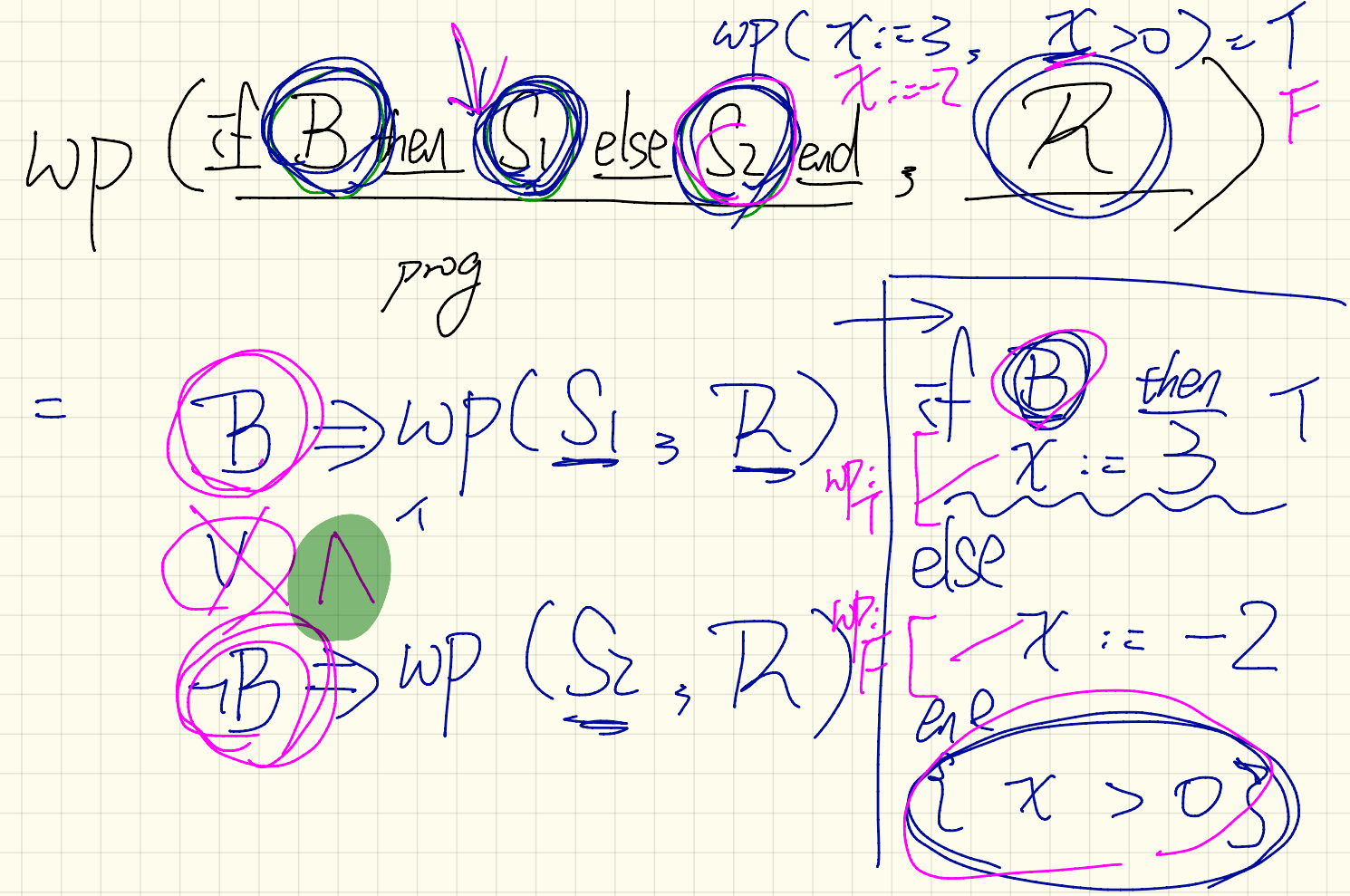
$= \{$ simp. $\}$

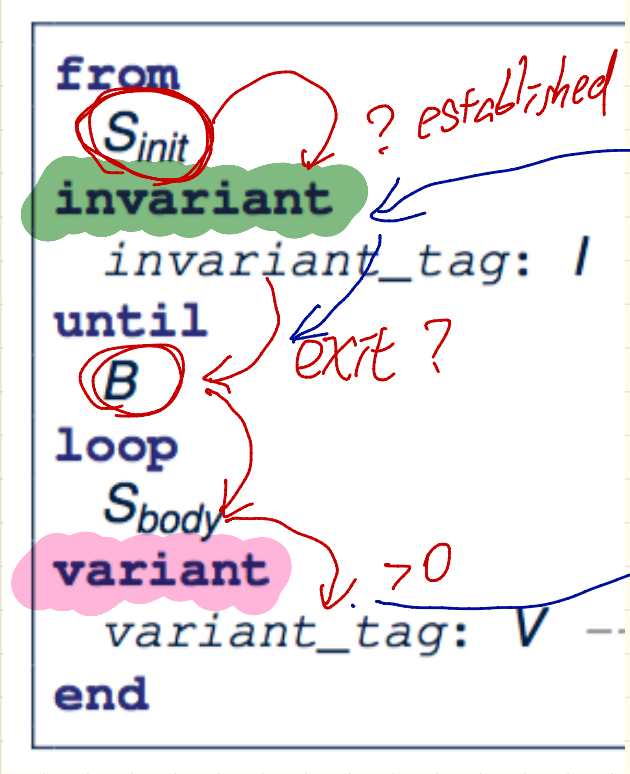
$i_0 > 4$ ← wp for $i := i + 9$ to establish $i > 13$

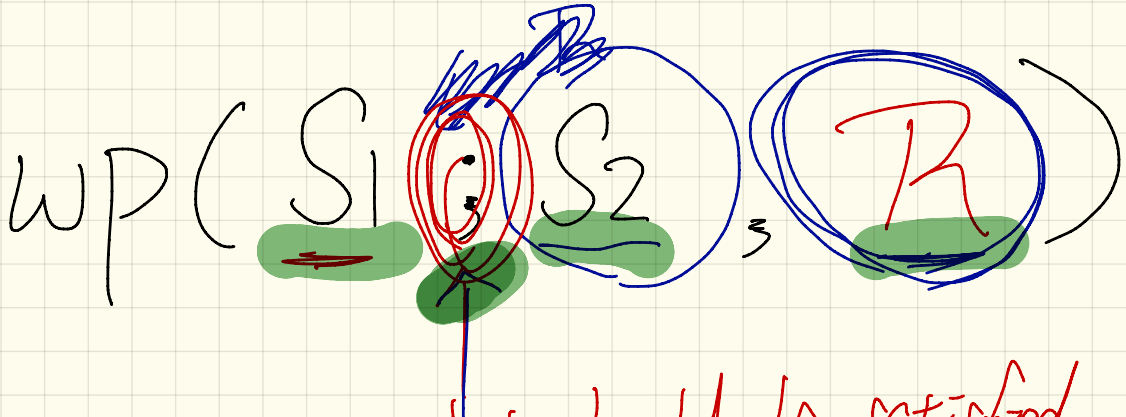


$\{x>0 \wedge y>0 \wedge x>y\}\ S_1\ \{bigger \geq smaller\}$

$\{x>0 \wedge y>0 \wedge \neg(x>y)\}\ S_2\ \{bigger \geq smaller\}$


wp(x := 3, x > 0) = T
x := -2
F
wp (if B then S1 else S2 end ; R)
prog
=
B ⇒ wp(S1 ; R)
∧
¬B ⇒ wp (S2 ; R)
wp;
wp;
if B then T
x := 3
else
x := -2
end
{ x > 0 }


from
S_{init}
invariant
invariant_tag: I
until
B
loop
S_{body}
variant
variant_tag: V
end
? established
? maintained
exit ?
>0


wp(S1 ; S2 , R)
what should be satisfied
(starting precondition for S2)
established by S1
wp(S1 ; wp(S2 , R))
starting pointing of S2
ending point of S1

$$wp(S_1 ; S_2, R)$$

$$wp(S_1, wp(S_2, R))$$

starting condition for S_2 to establish R.

$\{True\}\ tmp := x;\ x := y;\ y := tmp\ \{x > y\}$

1. $wp(tmp := x;\ x := y;\ y := tmp,\ x > y)$

$= \{$ wp for seq. comp. $\}$

$wp(tmp := x,\ wp(x := y;\ y := tmp,\ x > y))$

$= \{$ wp for seq. comp. $\}$

$wp(tmp := x,\ wp(x := y,\ wp(y := tmp,\ x > y)))$

$= \{$ wp for assign. $\}$

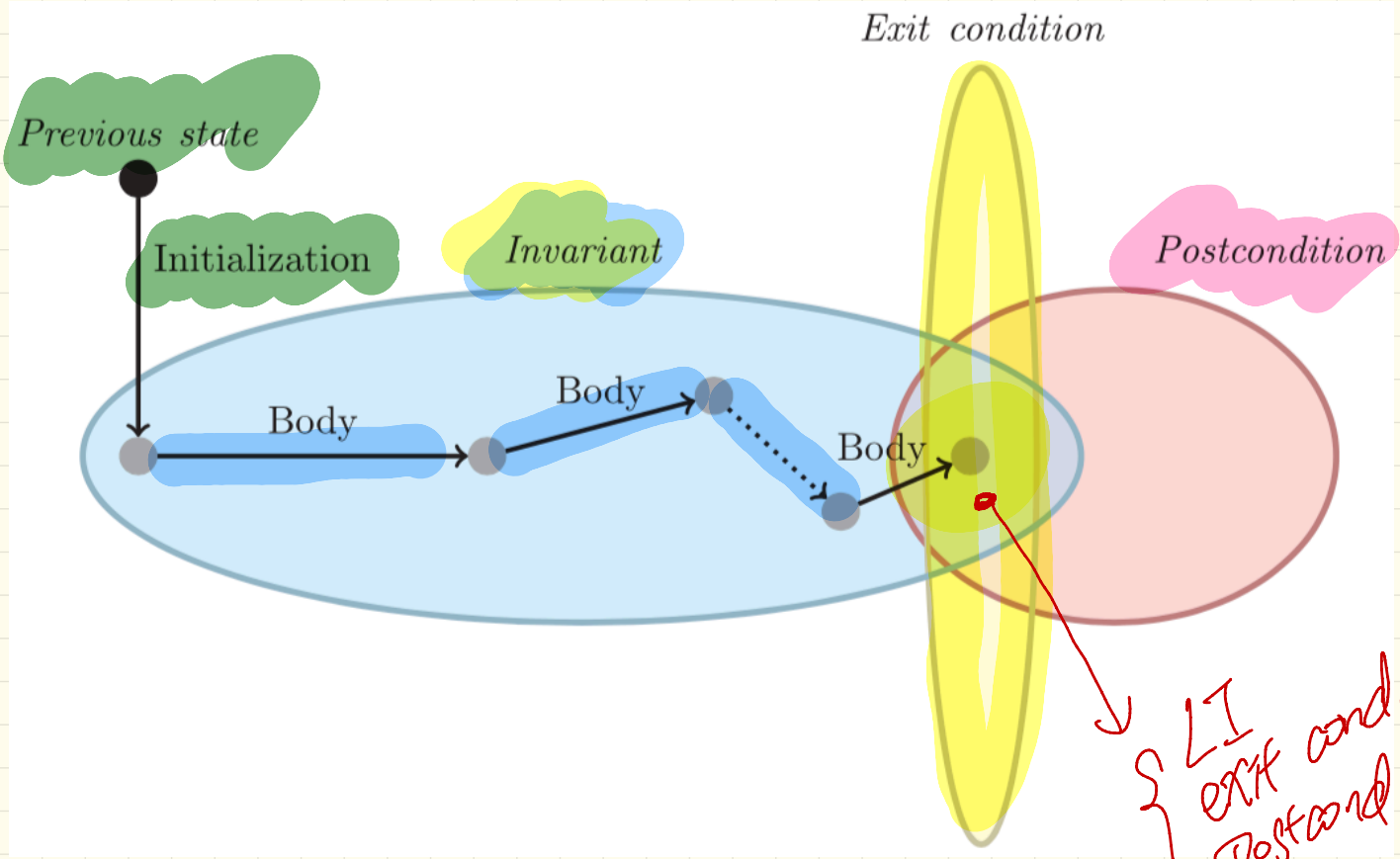
$wp(tmp := x,\ wp(x := y,\ x > y\ [y := tmp]))$

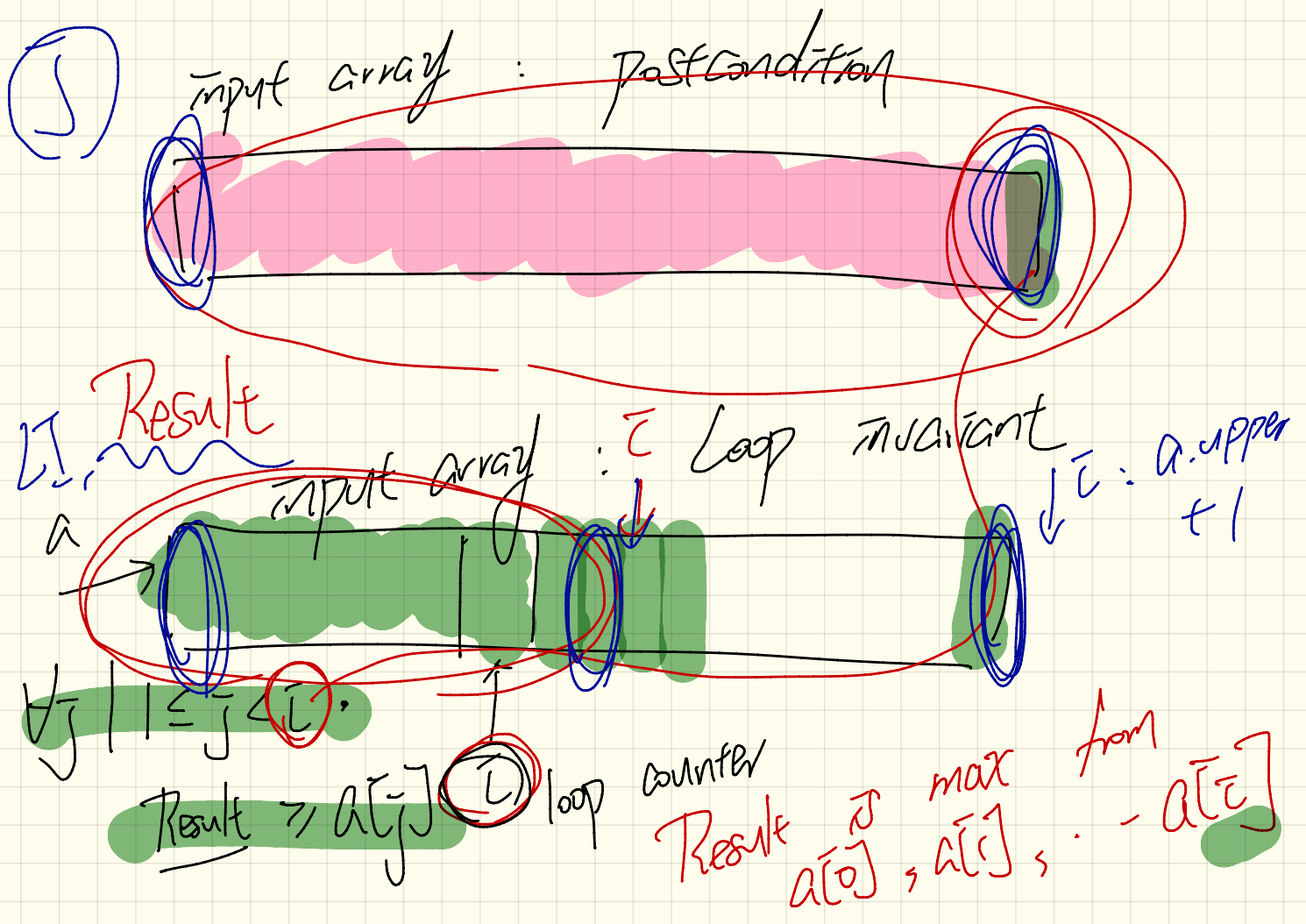
$= \{$ wp for assign $\}$

$wp(tmp := x,\ y > tmp) = y > x$

$x \geq tmp$

$True \Rightarrow y \geq x$


Exit condition
Previous state
Initialization
Invariant
Postcondition
Body
Body
Body
LI
exit cond.
postcond.


input array : postcondition
Result
input array : i Loop invariant
i : a.upper + 1
a
∀j | 1 ≤ j < i :
Result ≥ a[j]
i
loop counter
Result
max
from
a[0] , a[1] , . . . a[i]

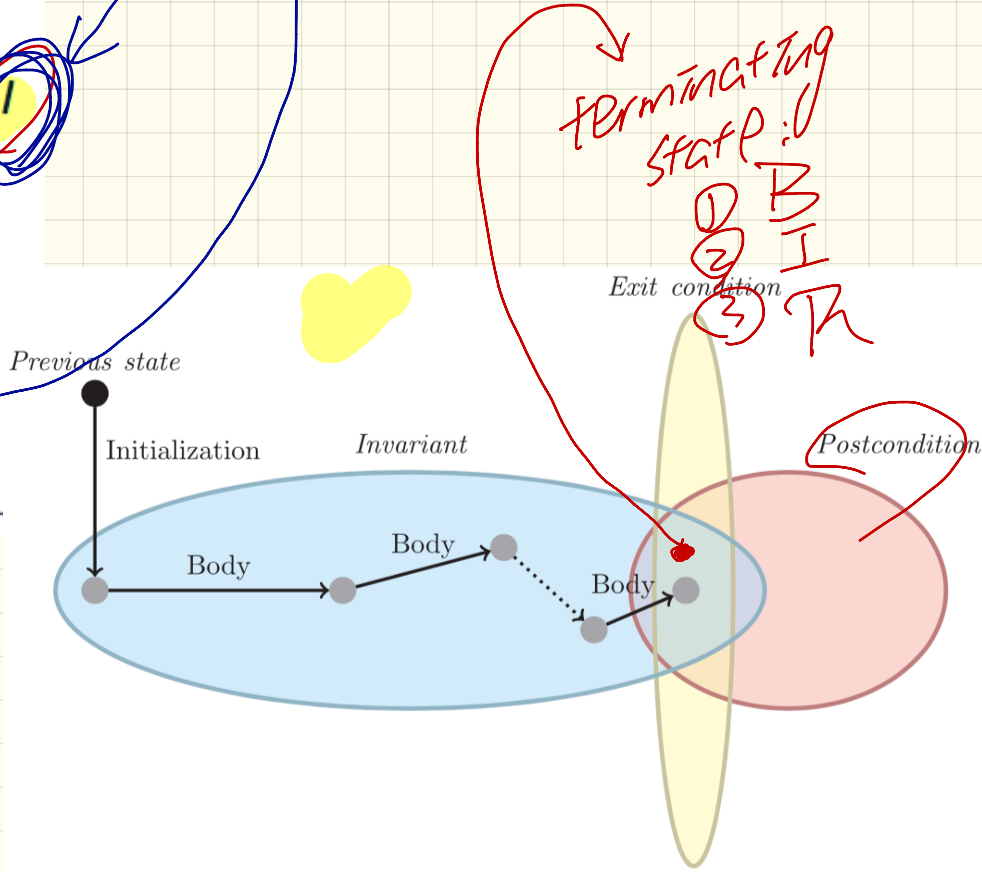
Lecture 23

Monday Dec. 4

```
from
  S_init
invariant
  invariant_tag: I
until
  B
loop
  S_body
variant
  variant_tag: V
end
```

ensure R

end



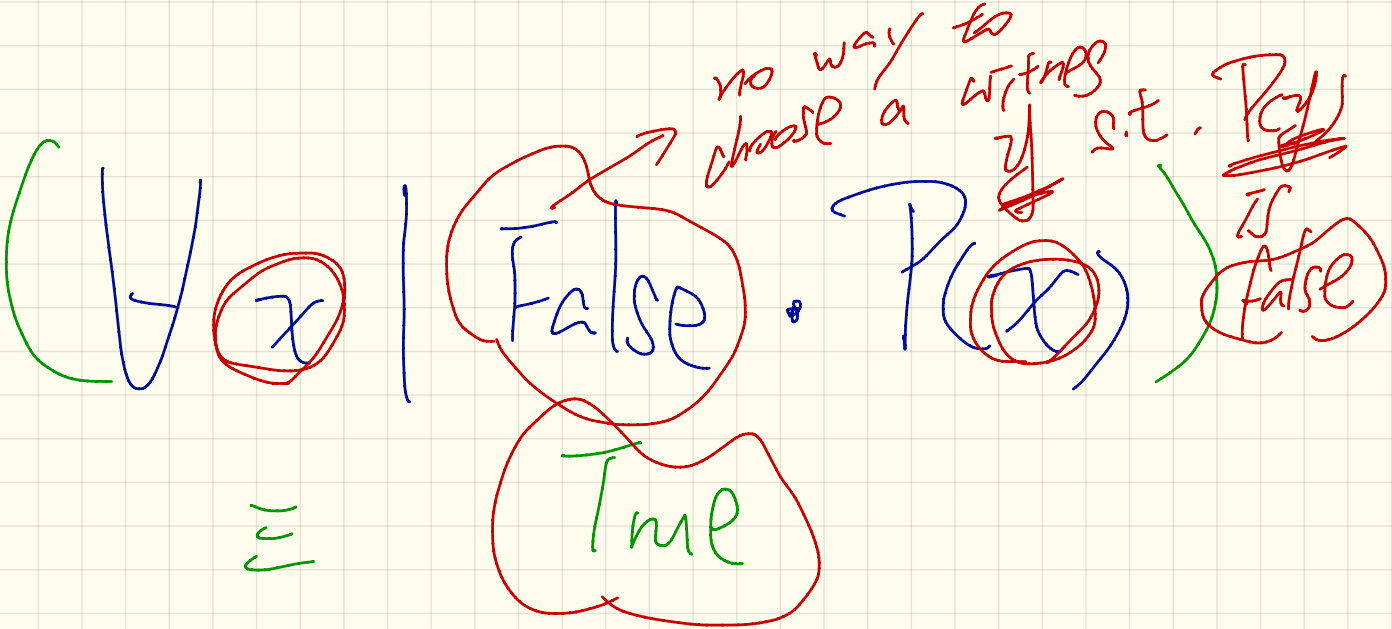
Consider the feature call find_max(⟨⟨20, 10, 40, 30⟩⟩), given:

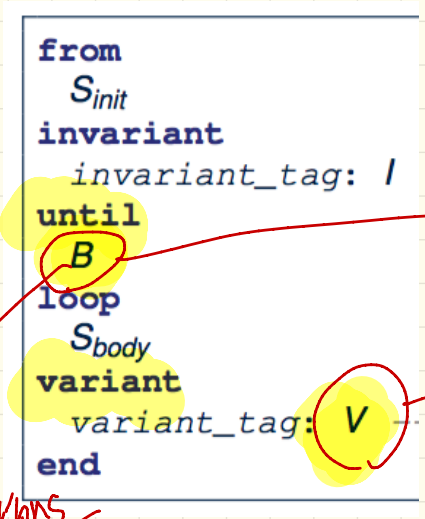
- **Loop Invariant**: $\forall j \mid a.lower \le j < i \bullet Result \ge a[j]$
- **Loop Variant**: $a.upper - i + 1$
- **Postcondition**: $\forall j \mid a.lower \le a.lower \le a.upper \bullet Result \ge a[j]$

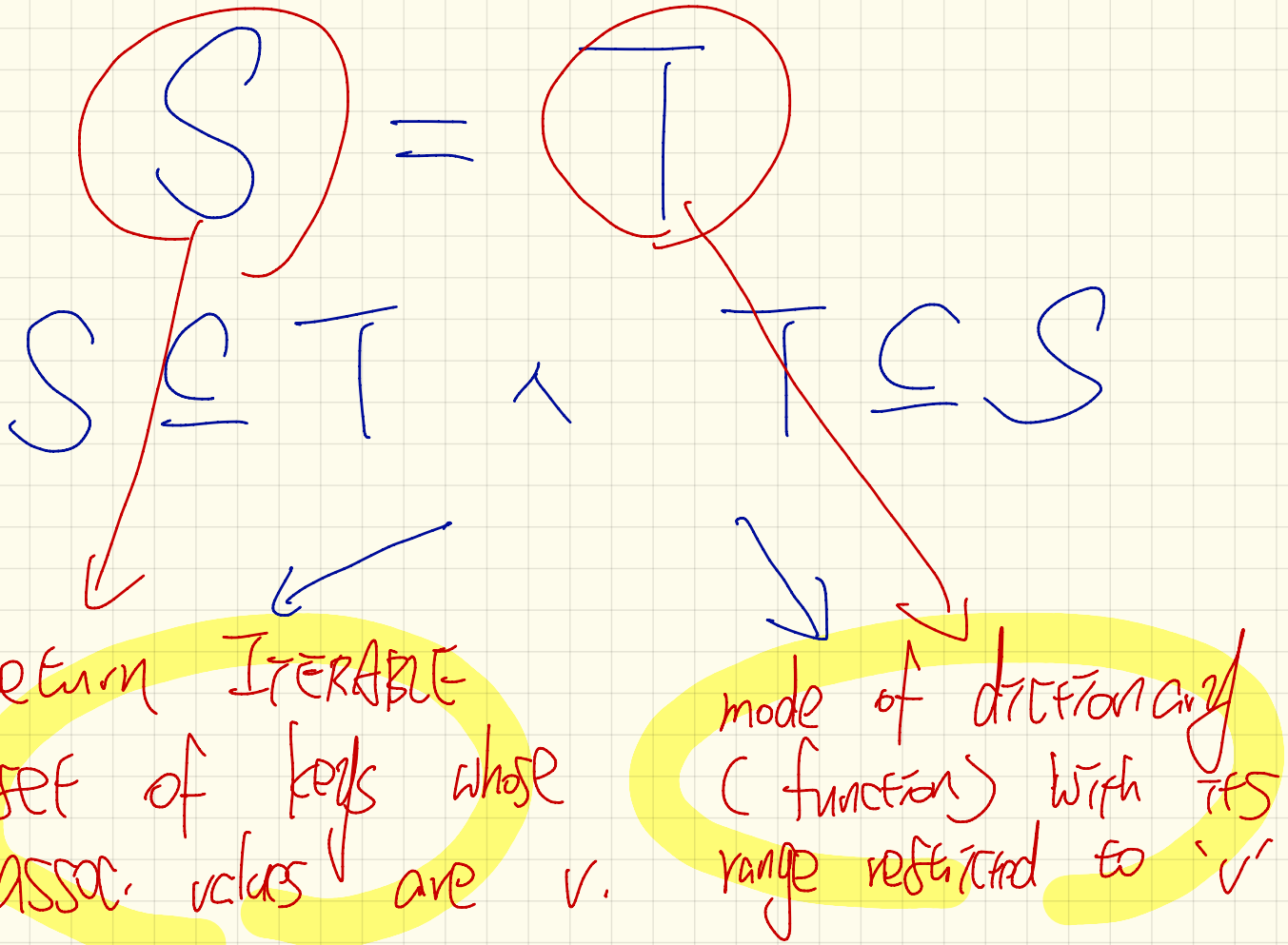
After Iteration	i	Result	LI	Exit (i > a.upper)?	LV
Initialization	1	20	T	F	–
1st	2	20			3
2nd	3				2
3rd	4				1
4th	5				0

Init: i := 1
Result := a[1] (20)

$\forall j \mid 1 \le j < 1 \cdot 20 \ge a[j]$
F ≡ T


no way to choose a x from y s.t. P(x) is False
(∀x | False . P(x))
≡
True


from
S_{init}
invariant
invariant_tag: I
until
B
loop
S_{body}
variant
variant_tag: V
end


S = T
S ⊆ T ∧ T ⊆ S
return ITERABLE set of keys whose assoc. values are v.
mode of dictionary (function) with its range restricted to 'v'

END.

ALL THE BEST ☺