EECS 3311

Software Design

Fall 2018

Thursday Sep. 6

Lecture 1

Client vs. Supplier in OOP

```
class Microwave {
  private boolean on;
  private boolean locked;
  void power() {on = true;}
  void lock() {locked = true;}
  void heat(Object stuff) {
    /* Assume: on && locked */
    /* stuff not explosive. */
  } }
```

supplier

```
class MicrowaveUser {
  public static void main(...) {
    Microwave m = new Microwave();
    Object obj = ???;
    m.power(); m.lock();
    m.heat(obj);
  } }
```

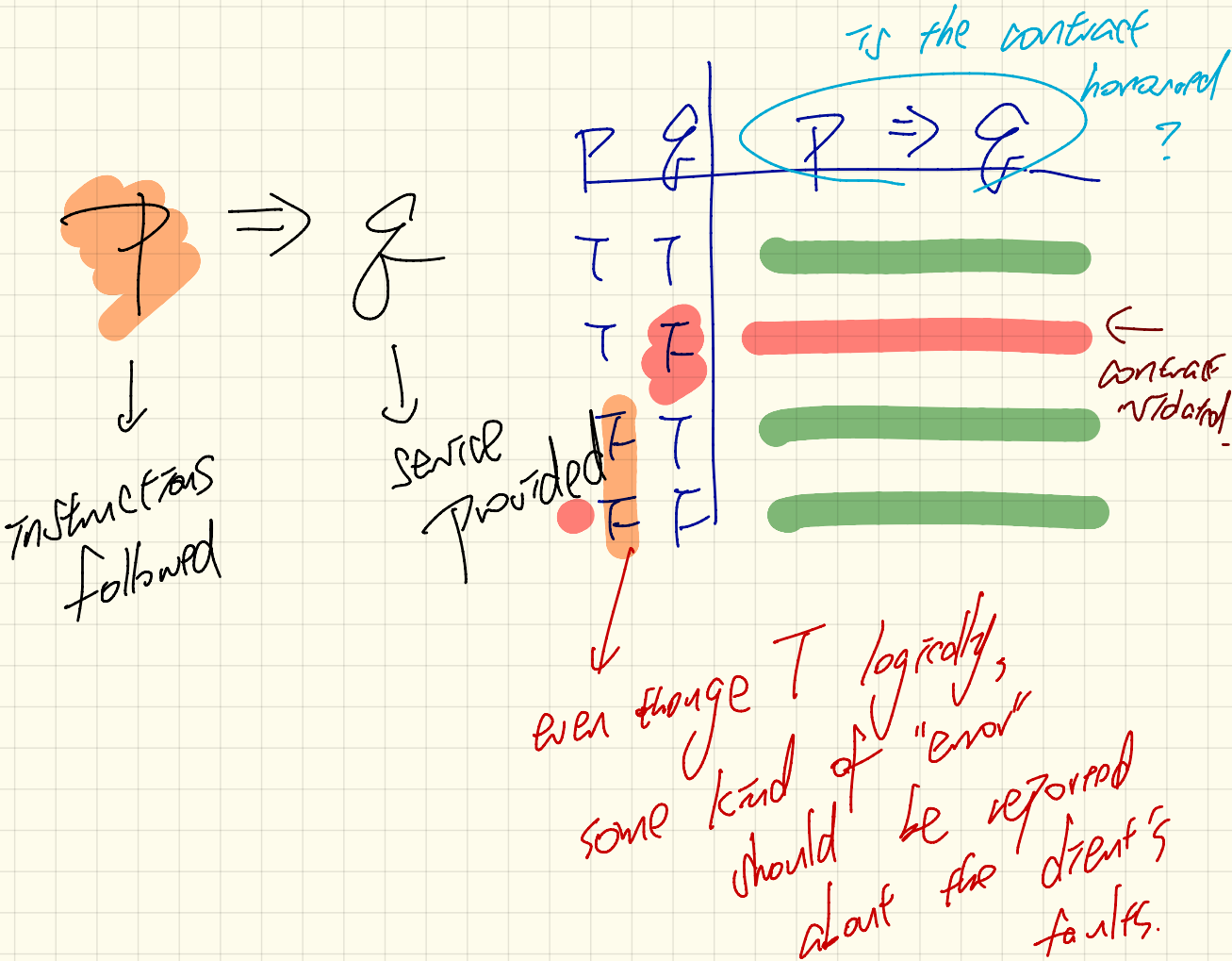
client

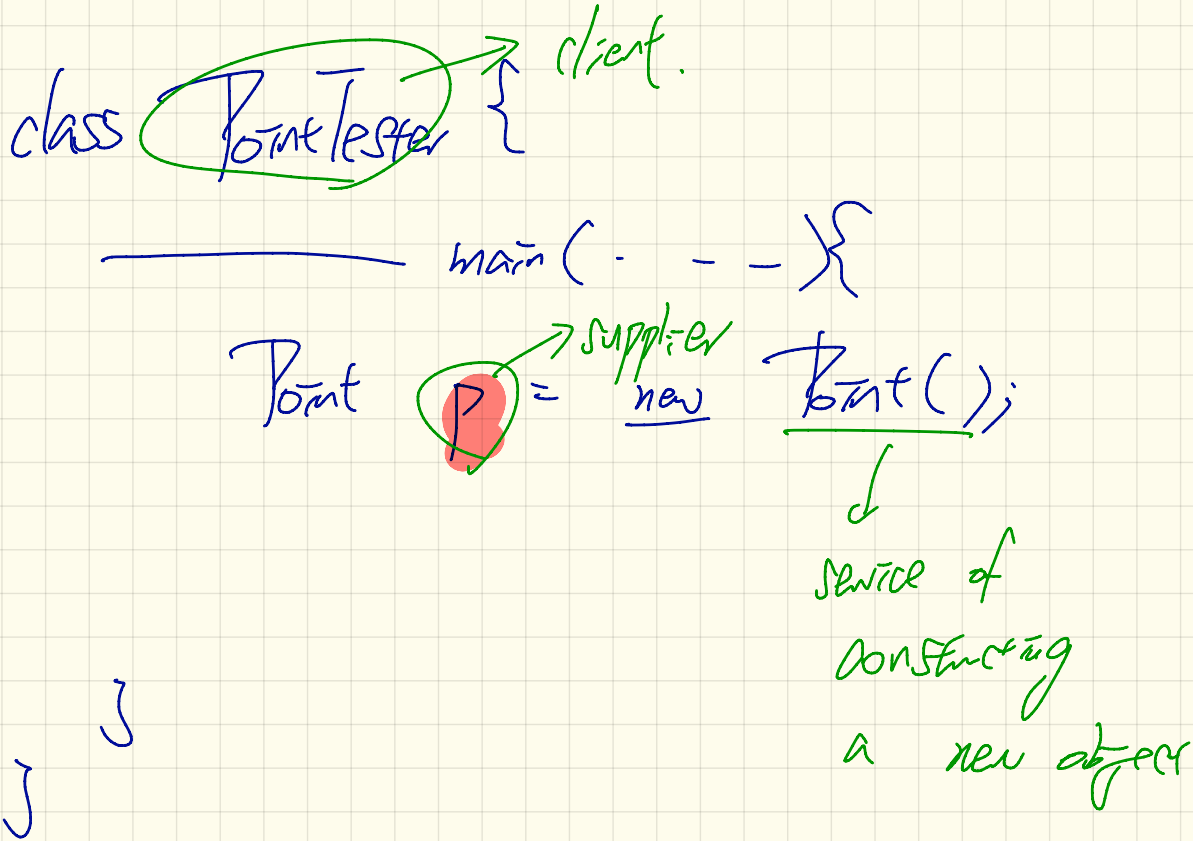
pre-state (client's obligations true) → m.heat(obj); → post-state (supplier's oblig. true) → client's benefits true.

supplier's benefits are true.

Context object

call in the context of MU class


P ⇒ Q
instances followed
service provided
P Q
P ⇒ Q
Is the contract honored ?
T T
T F
F T
F F
← contract violated
even though T logically, some kind of "error" should be reported about the client's faults.


class PointTester {
client.
main(. . .){
Point p = new Point();
supplier
service of constructing a new object
}
}

Tuesday Sep. 11

Lecture 2

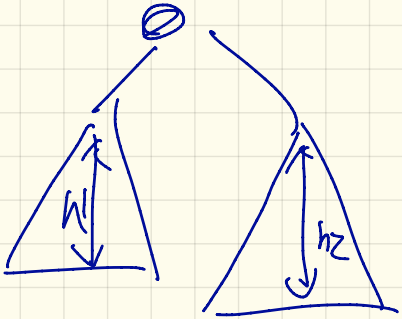
- Lab 1 tonight or tmw morning

- No office hours today

Requirements of Bank

REQ1: Each account is associated with the *name* of its owner (e.g., "Jim") and an integer *balance* that is always positive.

REQ2: We may *withdraw* an integer amount from an account.



$|h1 - h2| \leq 2$

global property for all Account objects.

Version 1: No Contracts

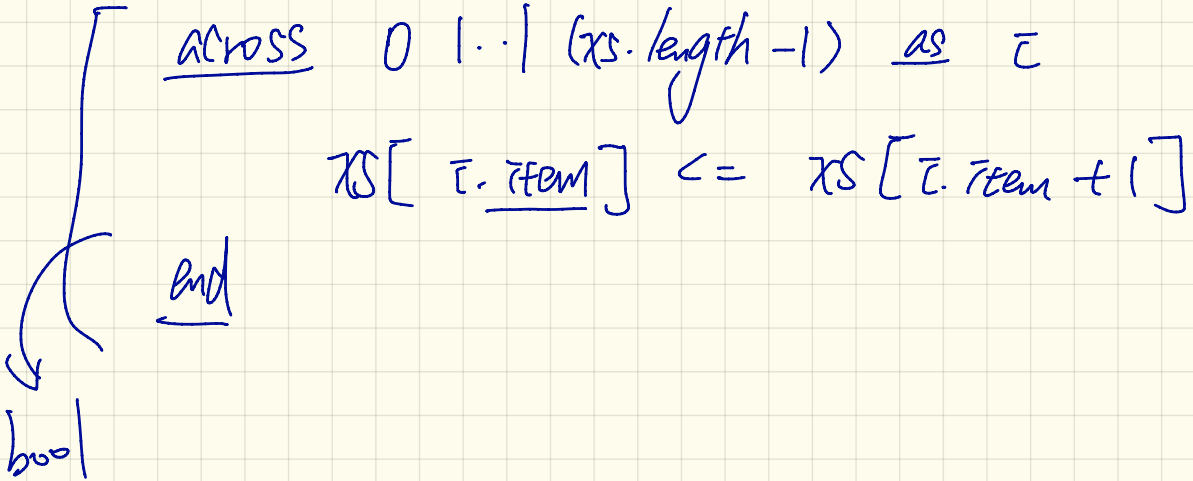
```
public class AccountV1 {
	private String owner;
	private int balance;
	public String getOwner() { return owner; }
	public int getBalance() { return balance; }
	public AccountV1(String owner, int balance) {
		this.owner = owner; this.balance = balance;
	}
	public void withdraw(int amount) {
		this.balance = this.balance - amount;
	}
	public String toString() {
		return owner + "'s current balance is: " + balance;
	}
}
```

Q: int[] xs sorted?

$\forall i : 0 \le i < xs.length \cdot xs[i] \le xs[i+1]$

$i+1 > i$

$xs[i] \le xs[i+1]$


across 0 |..| (xs.length - 1) as i
xs[i.item] <= xs[i.item + 1]
end
bool

Version 2: "Approximated" Preconditions

```
public class AccountV2 {
  public AccountV2(String owner, int balance) throws
     BalanceNegativeException
  {
   if( balance < 0 ) { /* negated precondition */
    throw new BalanceNegativeException(); }
   else { this.owner = owner; this.balance = balance; }
  }
  public void withdraw(int amount) throws
     WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
    if( amount < 0 ) { /* negated precondition */
     throw new WithdrawAmountNegativeException(); }
    else if ( balance < amount ) { /* negated precondition */
     throw new WithdrawAmountTooLargeException(); }
    else { this.balance = this.balance - amount; }
  }
```

B

assert b > 0

assert b > 0

Invariant

b > 0

the global invariant (bal > 0) is not encoded.

try { B } catch (Exception e) { fc }

fce instanceof BVE

else fc

class BalancedBST {

void insert(int c) {

insert

restore bal. if necessary

// invariant: tree is balanced

}

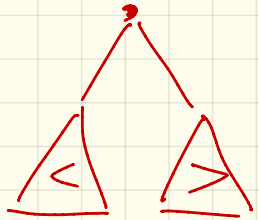
transfer(jim, tom) {

jim.withdraw(a)

tom.deposit(a)

}

possible inv. violation



Version 3: Class Invariant

```
public class AccountV3 {
  public AccountV3(String owner, int balance) throws
     BalanceNegativeException
  {
    if(balance < 0) { /* negated precondition */
      throw new BalanceNegativeException(); }
    else { this.owner = owner; this.balance = balance; }
    assert this.getBalance() > 0 : "Invariant:  positive balance";
  }
  public void withdraw(int amount) throws
     WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
    if(amount < 0) { /* negated precondition */
      throw new WithdrawAmountNegativeException(); }
    else if (balance < amount) { /* negated precondition */
      throw new WithdrawAmountTooLargeException(); }
    else { this.balance = this.balance - amount; }
    assert this.getBalance() > 0 : "Invariant:  positive balance";
  }
```

so balance : 100

this.b = 49

this.b = a - 1

100 50

bool binSearch (x, xs)

$x \in xs \Rightarrow Result == true$

$x \notin xs \Rightarrow Result == false$

$Result \Longleftrightarrow x \in xs$

Version 4: Uncaught Faultly Implementation

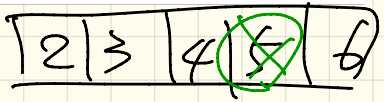
```
public class AccountV4 {
  public void withdraw(int amount) throws
    WithdrawAmountNegativeException, WithdrawAmountTooLargeException
  { if(amount < 0) { /* negated precondition */
      throw new WithdrawAmountNegativeException(); }
    else if (balance < amount) { /* negated precondition */
      throw new WithdrawAmountTooLargeException(); }
    else { /* WRONT IMPLEMENTATION */
      this.balance = this.balance + amount; }
    assert this.getBalance() > 0 :
      owner + "Invariant: positive balance"; }
```

Version 5: Postconditions

update(3, "dog")

```
public class AccountV5 {
  public void withdraw(int amount) throws
      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
    int oldBalance = this.balance;
    if(amount < 0) { /* negated precondition */
      throw new WithdrawAmountNegativeException(); }
    else if (balance < amount) { /* negated precondition */
      throw new WithdrawAmountTooLargeException(); }
    else { this.balance = this.balance - amount; }
    assert this.getBalance() > 0 :"Invariant: positive balance";
    assert this.getBalance() == oldBalance - amount :
      "Postcondition:  balance deducted"; }
```

update(3, 7)



7

0. class Array {

int[] a;

void update(int i, int v)

pre cond: $0 \leq i < a.length$

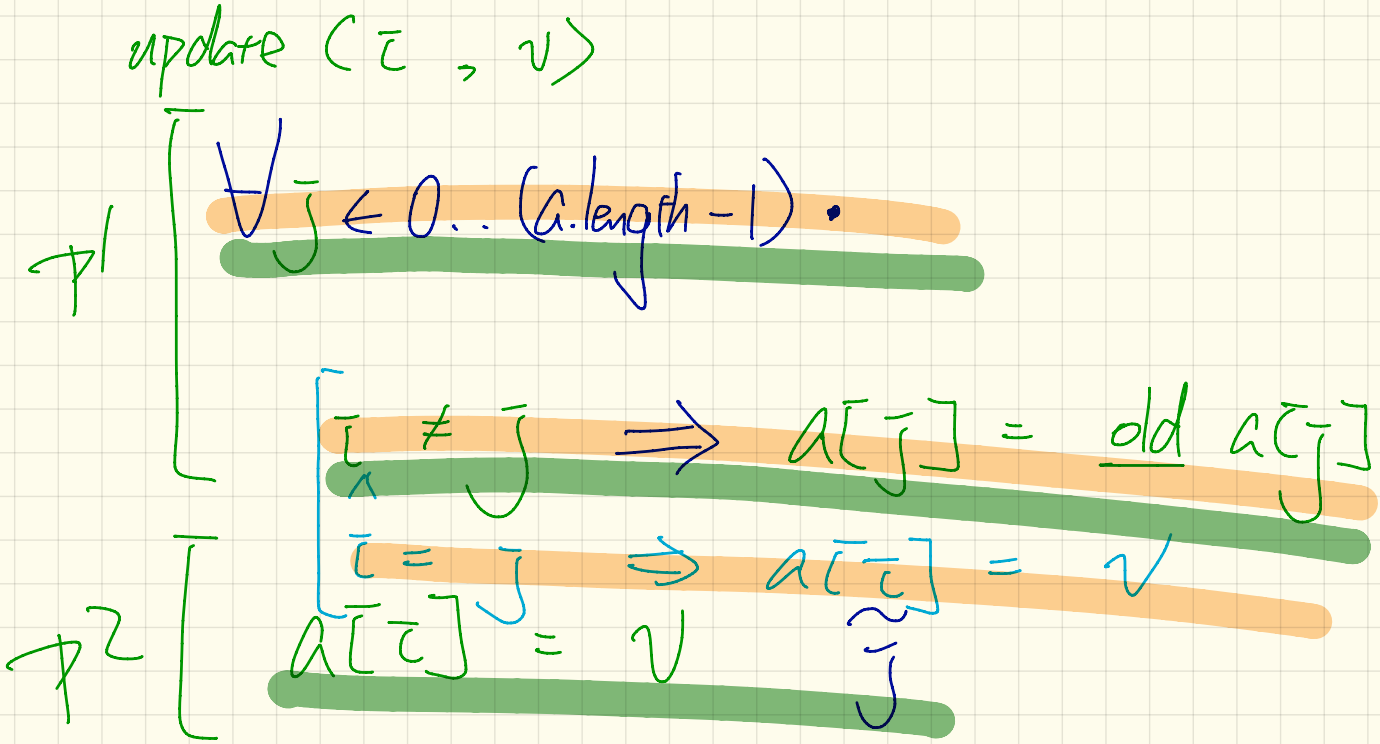
post cond: $a[i] = v$

FI.

a[i] = v;

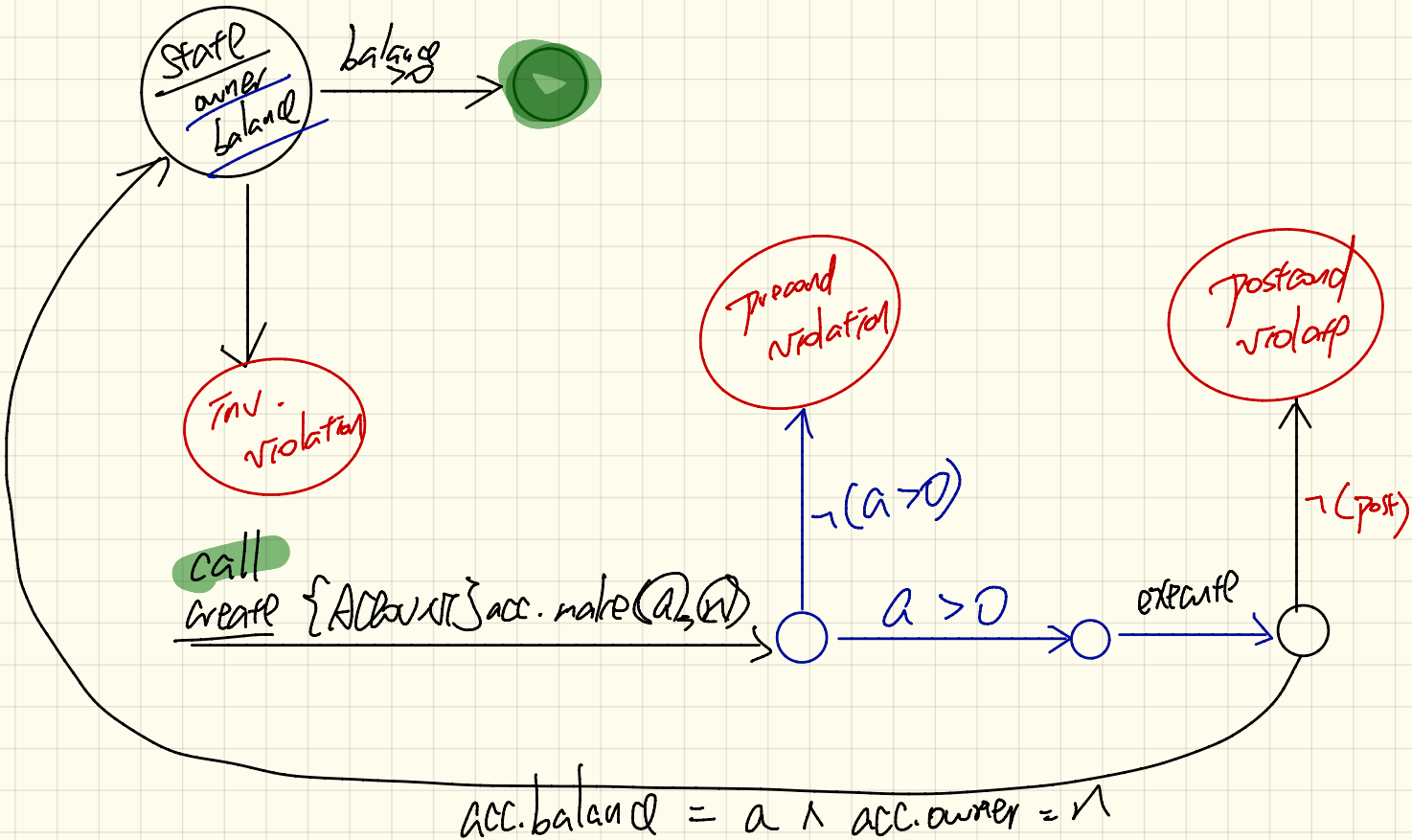
a[i-1] = v;

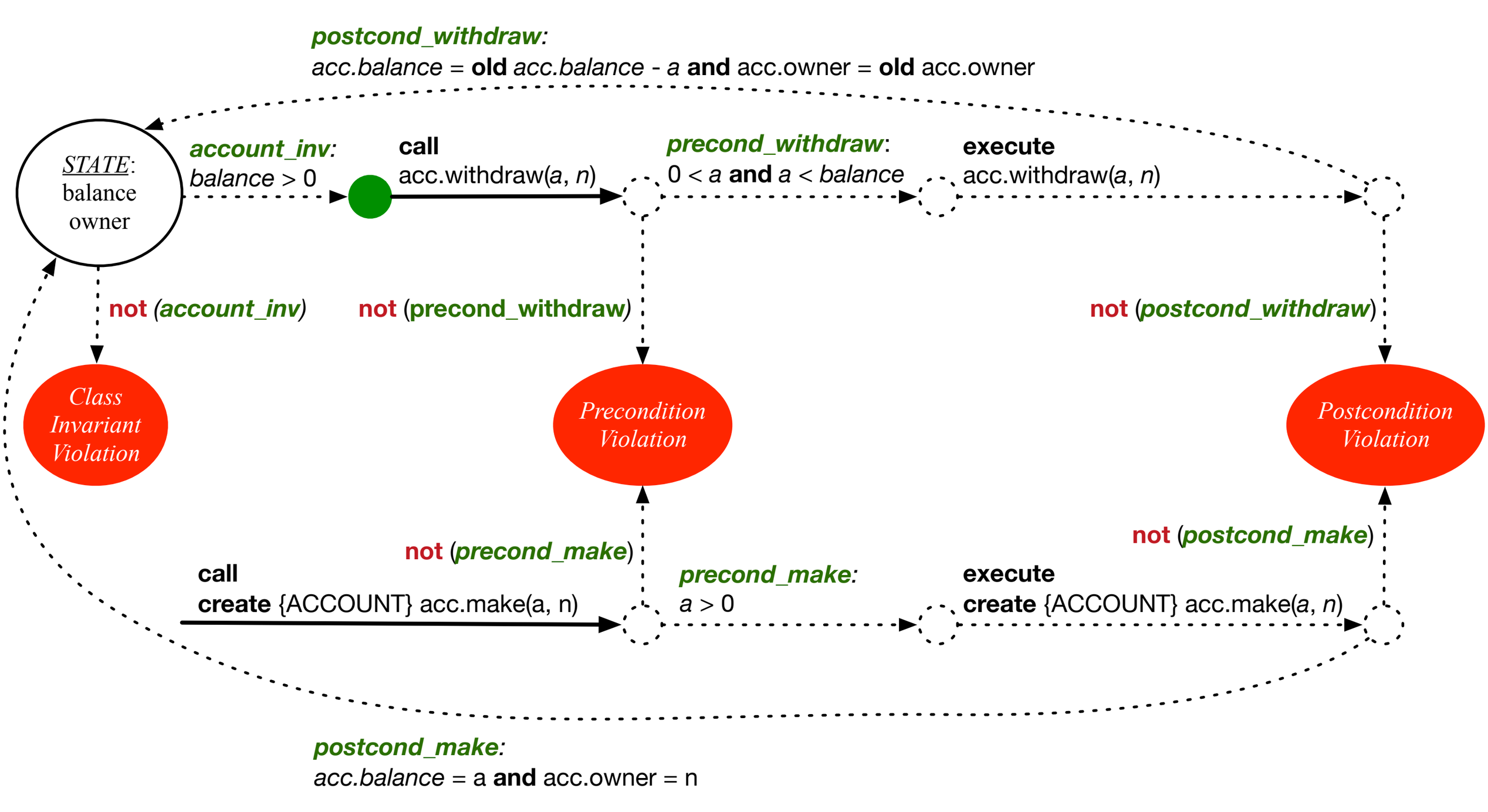
}


update (i , v)
p1
∀ j ∈ 0.. (a.length - 1) •
i ≠ j ⇒ a[j] = old a[j]
i = j ⇒ a[i] = v
p2
a[i] = v
i
j

Account Class: State, Contracts, Creation, Updates

```
class ACCOUNT
create
      make
feature -- Attributes
      owner : STRING
      balance : INTEGER
feature -- Constructors
      make(nn: STRING; nb: INTEGER)
            require -- precondition
                  positive_balance: nb >= 0
            do
                  owner := nn
                  balance := nb
            end
feature -- Commands
      withdraw(amount: INTEGER)
            require -- precondition
                  non_negative_amount: amount >= 0
                  affordable_amount: amount < balance
            do
                  balance := balance - amount
            ensure -- postcondition
                  balance_deducted: balance = old balance - amount
            end
invariant -- class invariant
      positive_balance: balance > 0
end
```

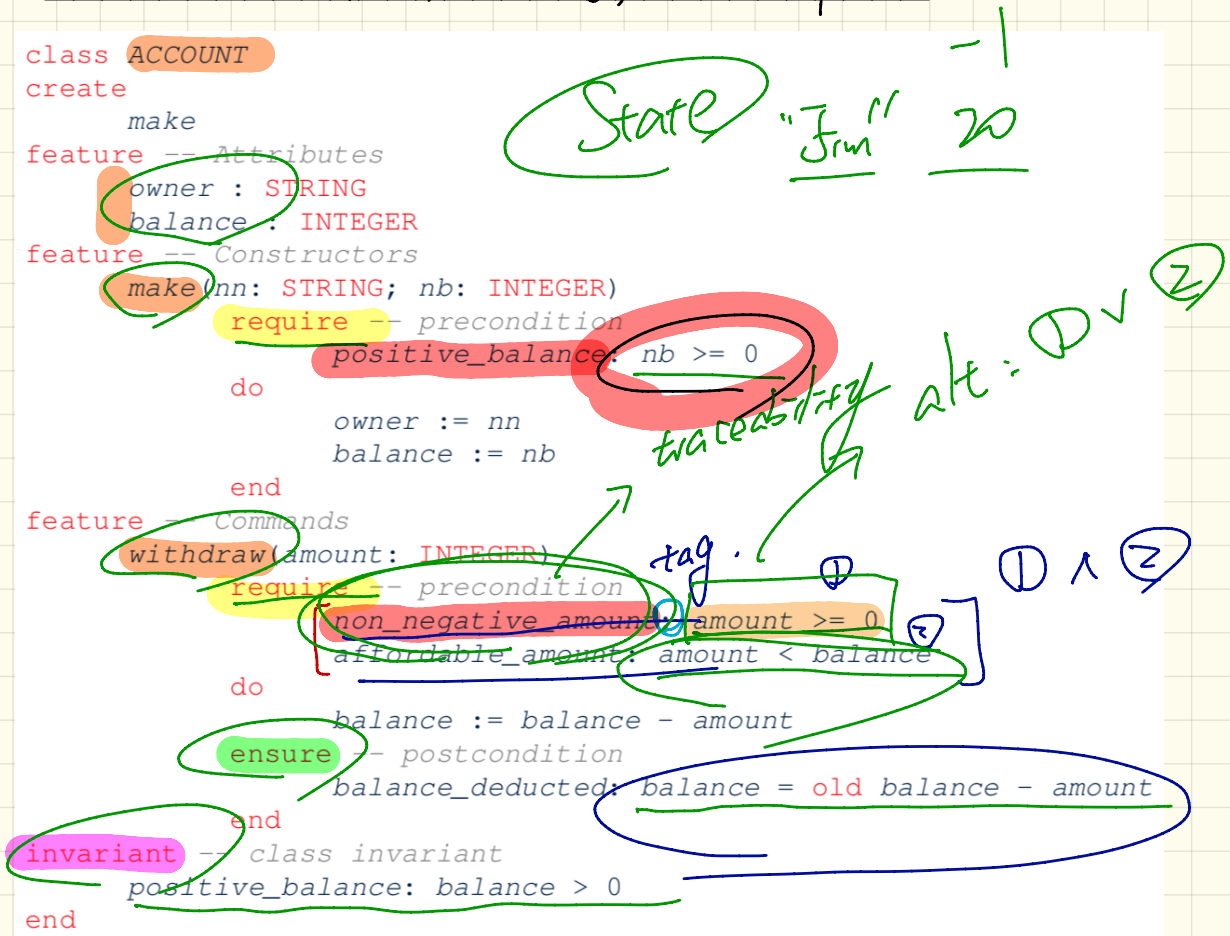

State
owner
balance
balance >0
Inv. violation
call
create {Account}acc.make(a,N)
Precond violation
¬(a>0)
a > 0
execute
Postcond violation
¬(post)
acc.balance = a ∧ acc.owner = N

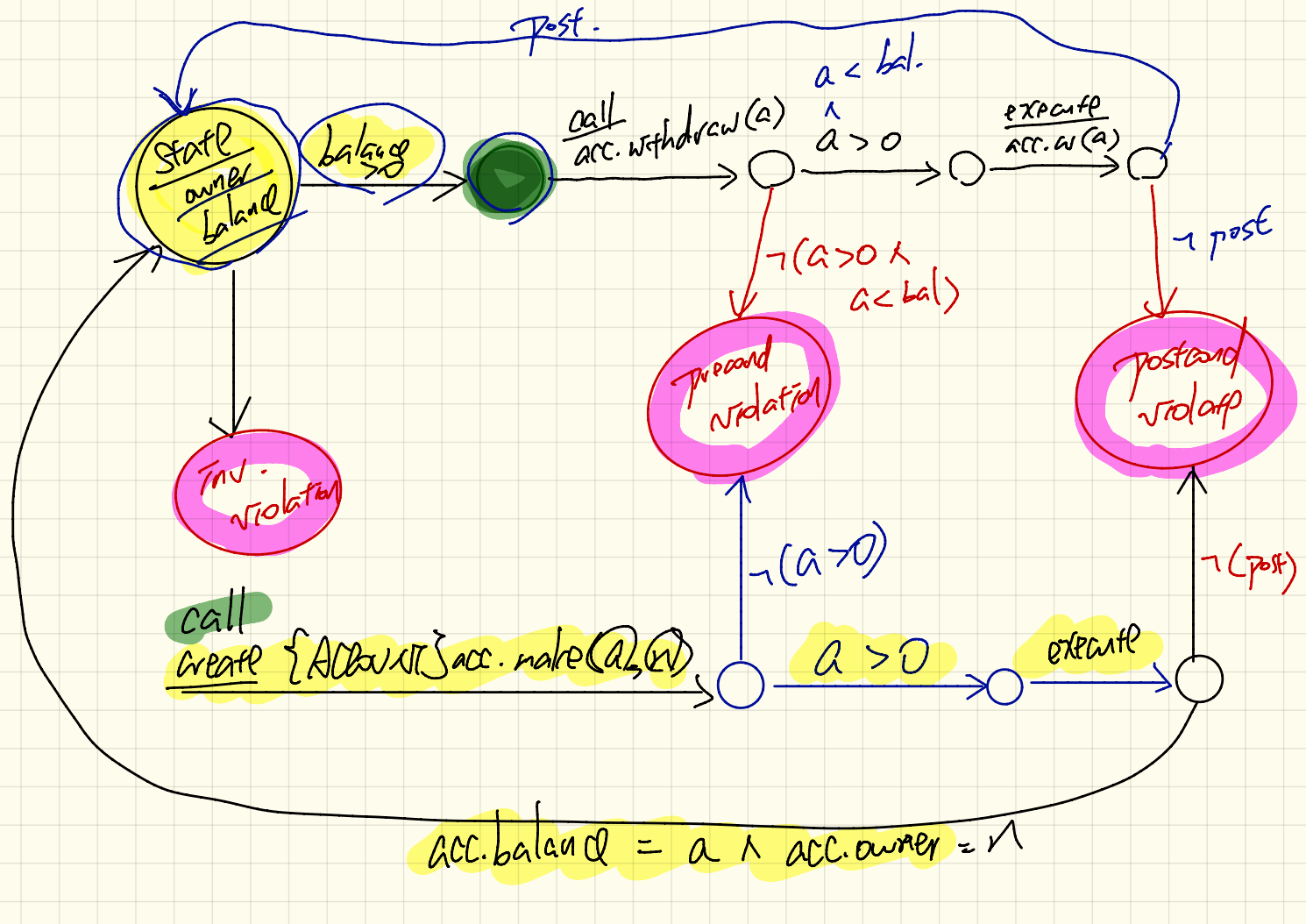

postcond_withdraw:
acc.balance = old acc.balance - a and acc.owner = old acc.owner
STATE:
balance
owner
account_inv:
balance > 0
call
acc.withdraw(a, n)
precond_withdraw:
0 < a and a < balance
execute
acc.withdraw(a, n)
not (account_inv)
not (precond_withdraw)
not (postcond_withdraw)
Class Invariant Violation
Precondition Violation
Postcondition Violation
not (postcond_make)
not (precond_make)
call
create {ACCOUNT} acc.make(a, n)
precond_make:
a > 0
execute
create {ACCOUNT} acc.make(a, n)
postcond_make:
acc.balance = a and acc.owner = n

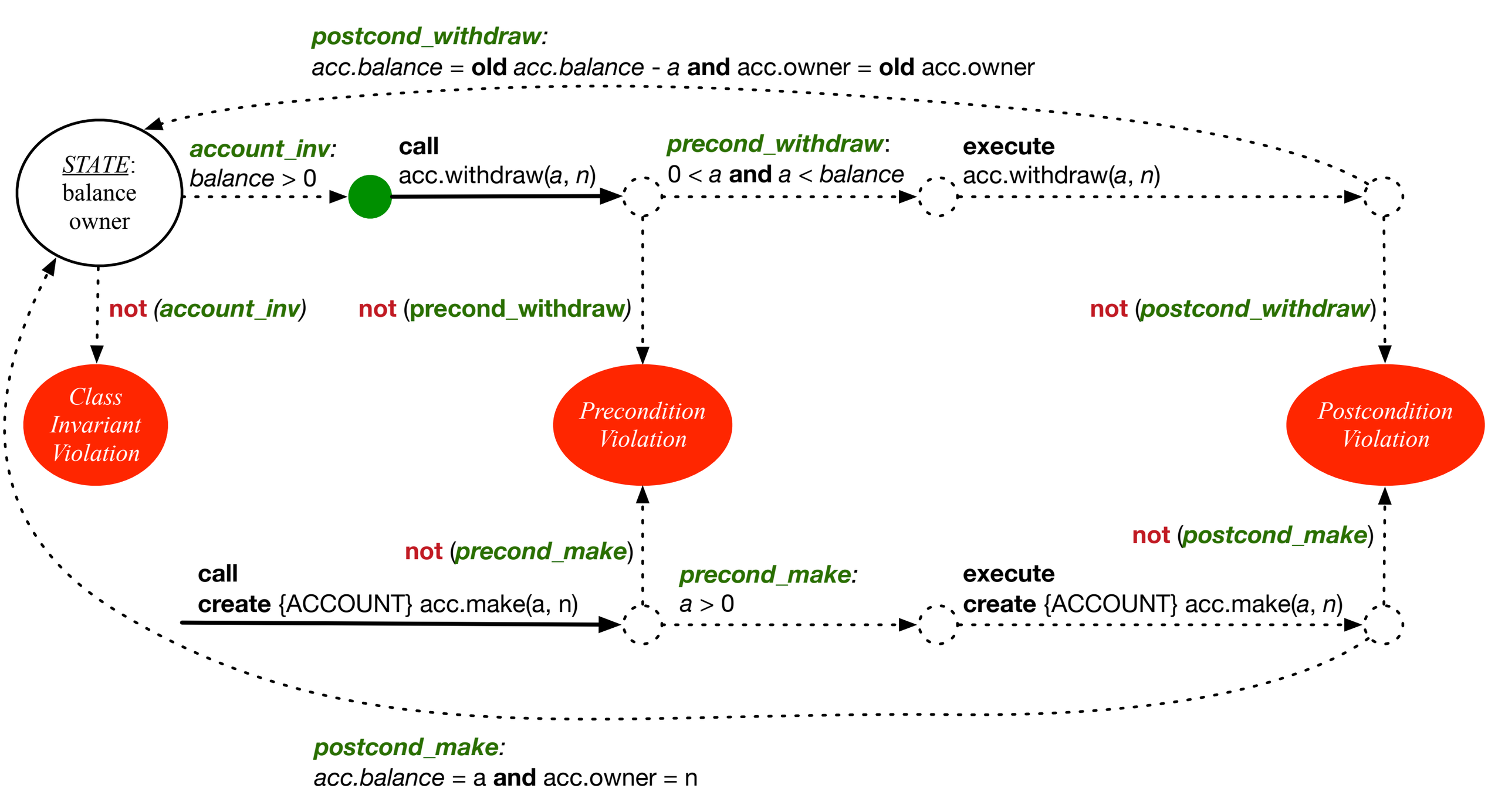
Thursday Sep. 13

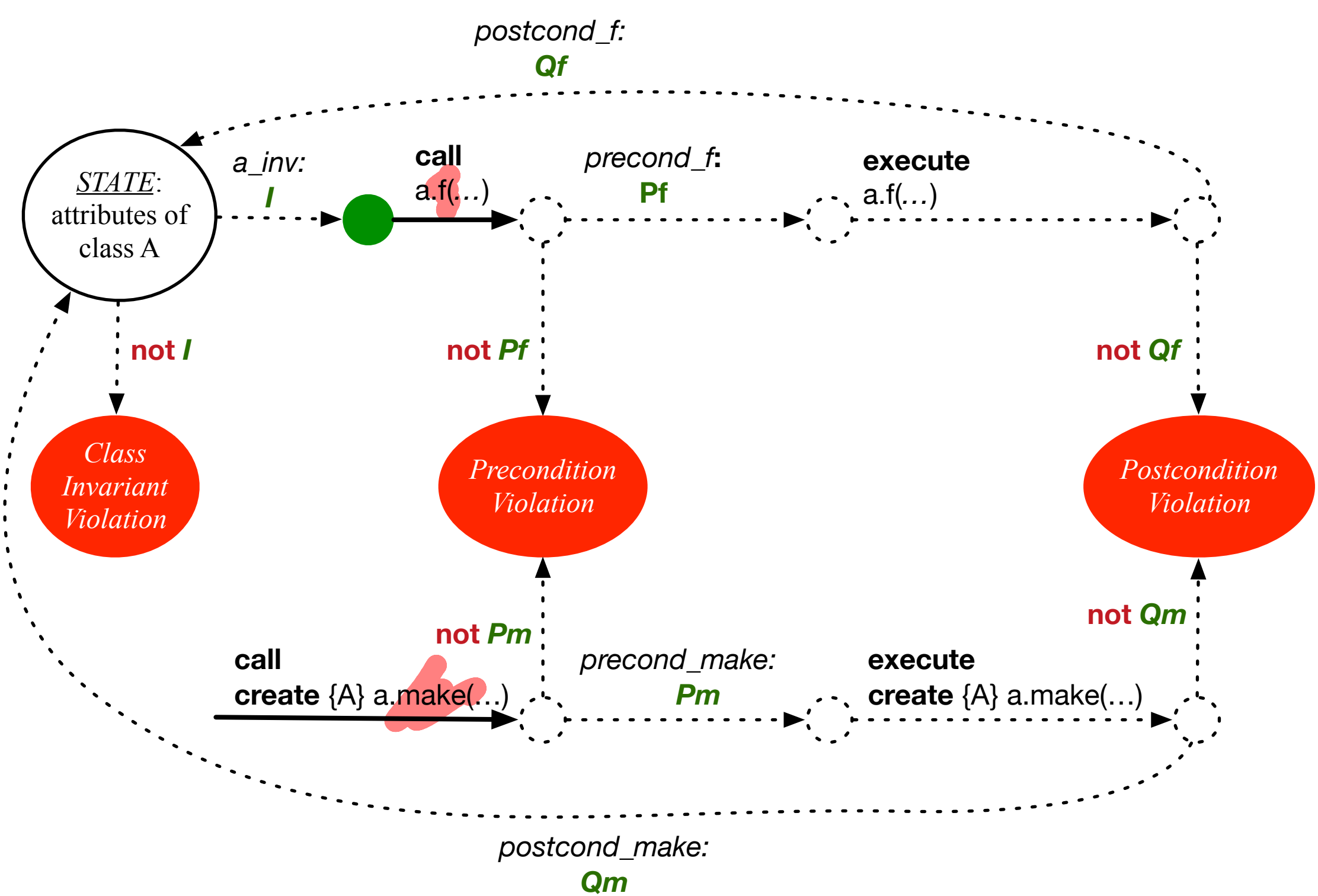
Lecture 3

Account Class: State, Contracts, Creation, Updates




post.
a < bal.
state
owner
balance
balance >0
call
acc.withdraw(a)
a > 0
execute
acc.w(a)
¬(a>0 ∧ a<bal)
¬ post
Inv. violation
Precond violation
Postcond violation
call
create {Account}acc.make(a,v)
¬(a>0)
a > 0
execute
¬(post)
acc.balance = a ∧ acc.owner = v


postcond_withdraw:
acc.balance = old acc.balance - a and acc.owner = old acc.owner
STATE:
balance
owner
account_inv:
balance > 0
call
acc.withdraw(a, n)
precond_withdraw:
0 < a and a < balance
execute
acc.withdraw(a, n)
not (account_inv)
not (precond_withdraw)
not (postcond_withdraw)
Class Invariant Violation
Precondition Violation
Postcondition Violation
not (postcond_make)
not (precond_make)
call
create {ACCOUNT} acc.make(a, n)
precond_make:
a > 0
execute
create {ACCOUNT} acc.make(a, n)
postcond_make:
acc.balance = a and acc.owner = n


postcond_f:
Qf
STATE:
attributes of
class A
a_inv:
I
call
a.f(...)
precond_f:
Pf
execute
a.f(...)
not I
not Pf
not Qf
Class
Invariant
Violation
Precondition
Violation
Postcondition
Violation
not Pm
not Qm
call
create {A} a.make(...)
precond_make:
Pm
execute
create {A} a.make(...)
postcond_make:
Qm

Java

$a = b$

$a == b$

Eiffel

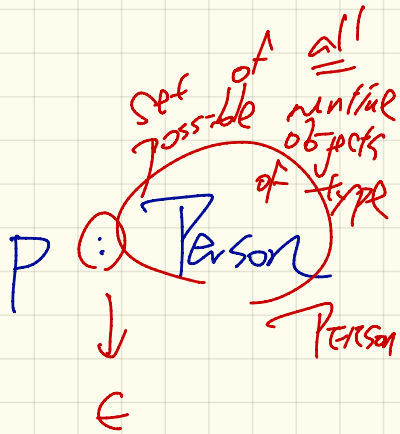
contracts

$a := b$

$a = b$

feature ::
  Name
  Preand
  Local

Person p



Java

a
i
and then

① a[i] >= 0 && 0 <= i && i < a.length

② 0 <= i && i < a.length && a[i] >= 0

∧ Java ×

Eiffel and or implies = ⇔

p	q	p ∧ q
T	T	
T	F	
F	T	
F	F	

command (a: ARRAY[INTEGER]; i: INTEGER)

require

rebase.

v1

not_too_small: i >= 0

not_too_big: i < a.count

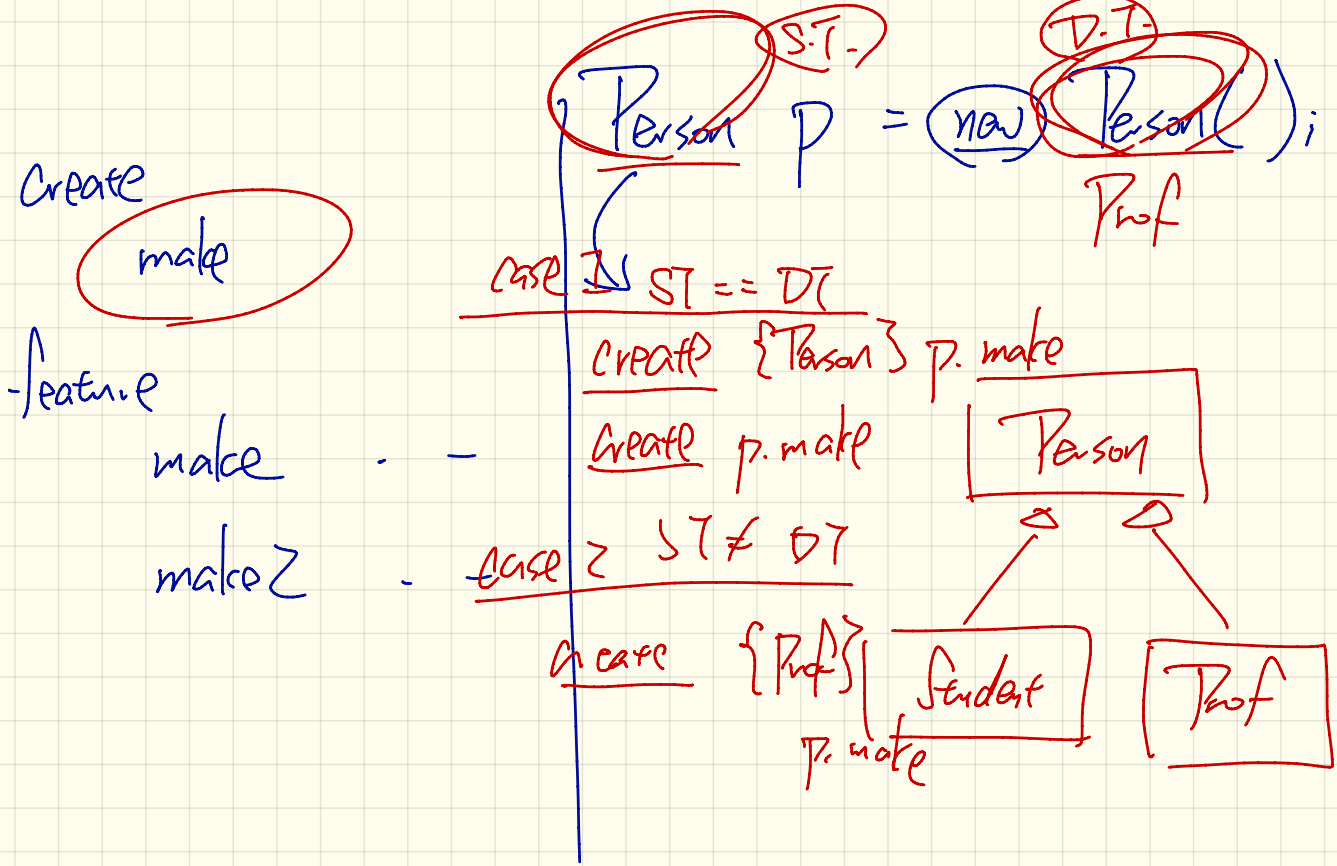
v2 ✗

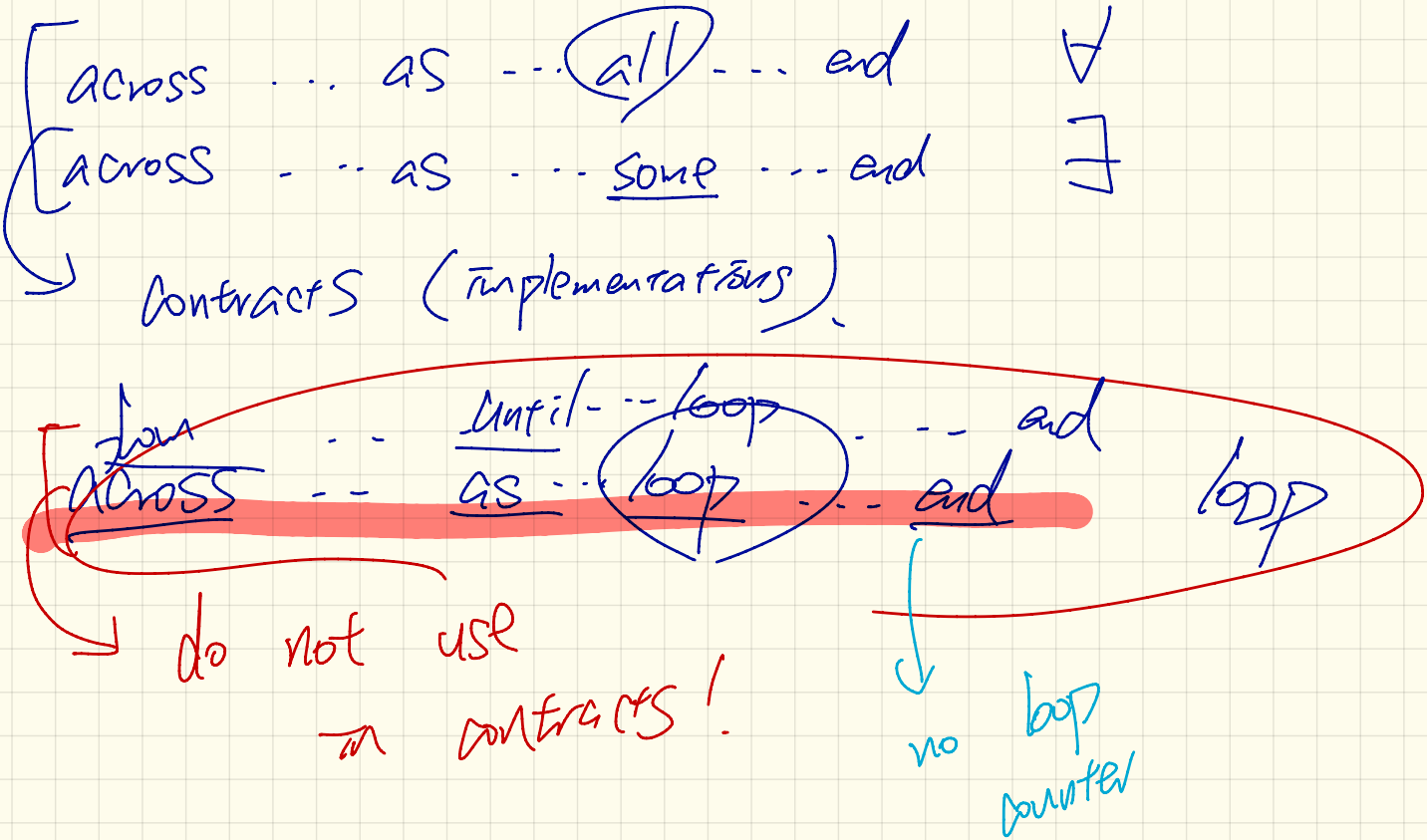
positive: a[i] > 0

pos_element: i >= 0 and then i < a.c and then a[i] > 0

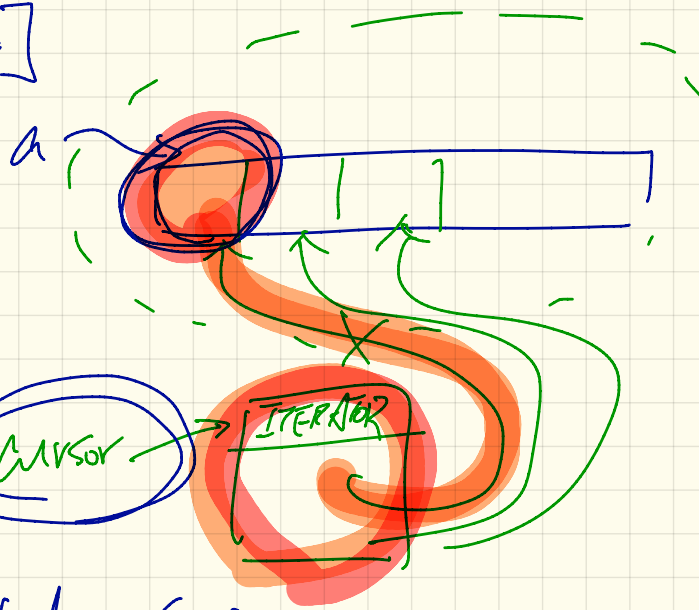
i >= 0 and i < a.count

and a[i] > 0


S.T.
D.T.
Person p = new Person();
Prof
create
make
-feature
make
make2
case 1 ST == DT
create {Person} p.make
create p.make
case 2 ST ≠ DT
create {Prof}
p.make
Person
Student
Prof


across ... as ... all ... end
across ... as ... some ... end
contracts (implementations)
from ... until ... loop ... end
across ... as ... loop ... end
loop
do not use
in contracts!
no loop counter

a: ARRAY [ACCOUNT]



across a as Cursor loop

Cursor.item.withdraw(10)

end

Cursor.withdraw(10)

```
from
    i := a.lower
until
    i > a.upper
loop
    a[i].withdraw(10)
    i := i + 1
end
```

$\forall i \; \forall j$ $\quad \forall i, j \in low..upp \cdot$

$i \neq j \Rightarrow a[i] \neq a[j]$

across

1 1..1 10 as c1

a.lower a.upper

all

across

all 1 1..1 10 as c2

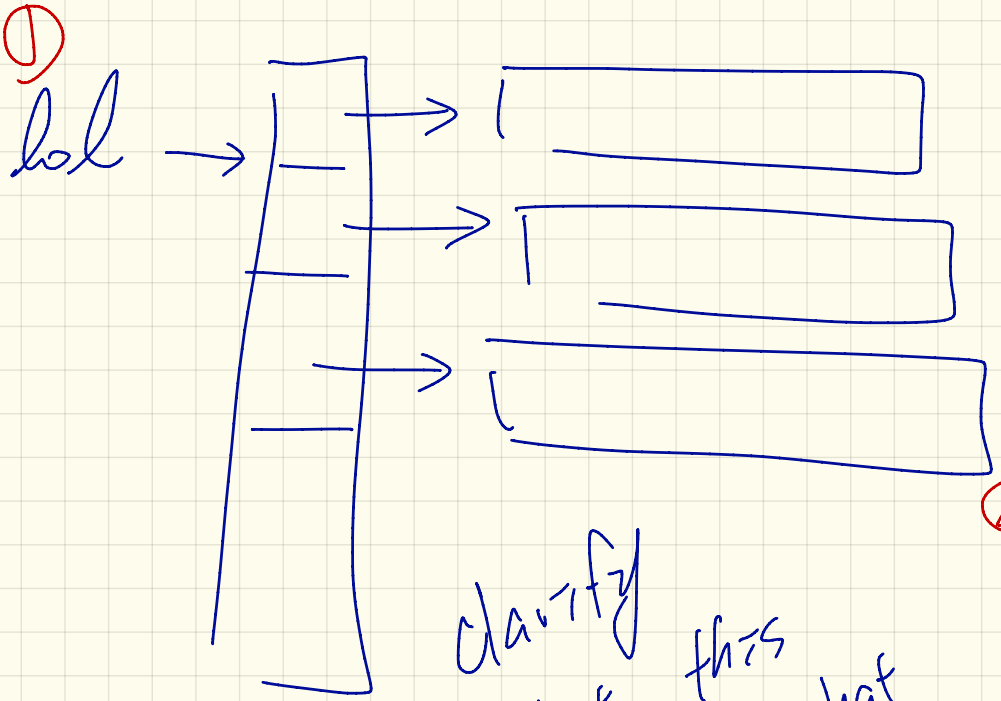
a.lower a.upper

[c1.item /= c2.item implies

a[c1.item] /= a[c2.item]

end

end



clarify that this is what you meant

④ no create, would default-create still work?

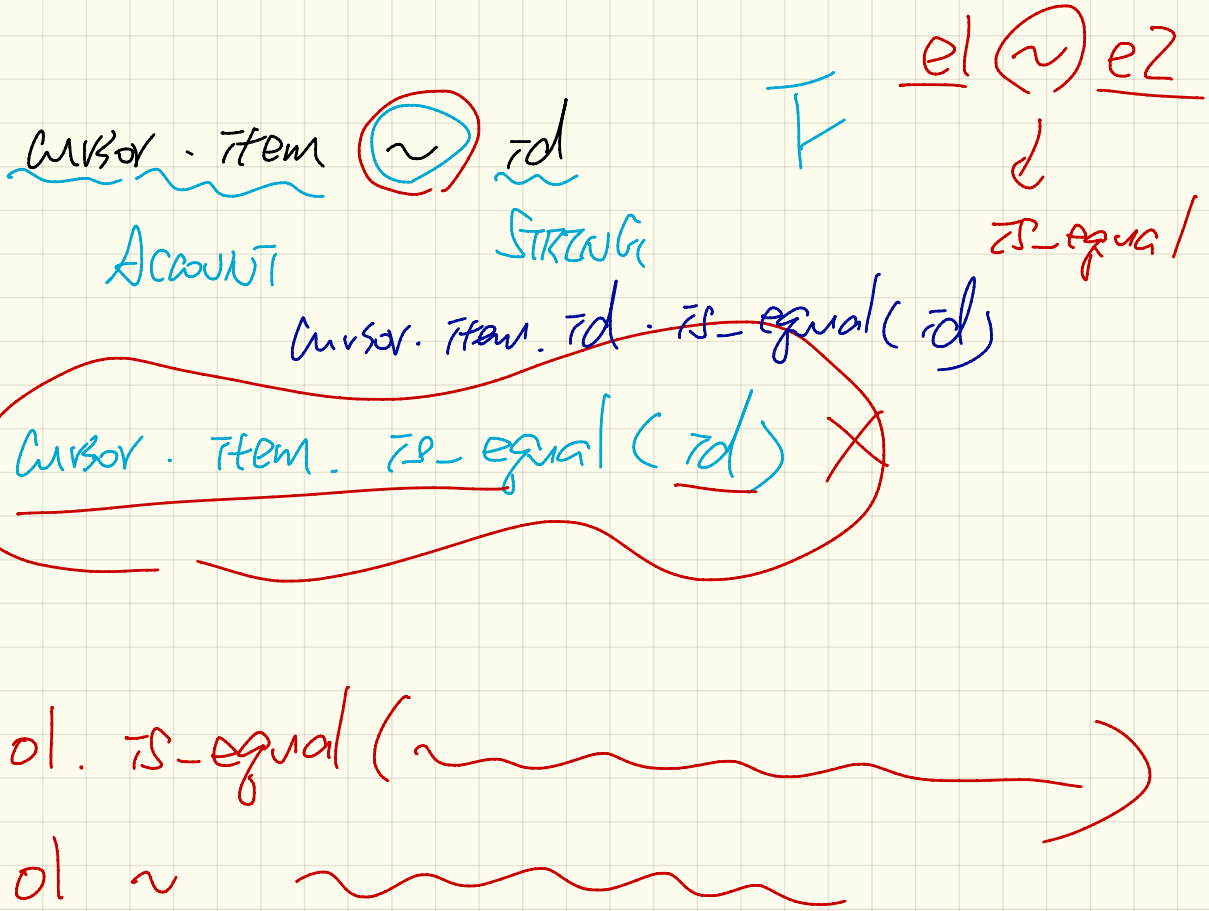
② creat f.make() ✓
create f.make2() ✗

f.make() ✓
f.make2() ✓

③ no overloading
↳ T-T f/e!

Tuesday Sep. 18

Lecture 4


cursor.item ~ id
ACCOUNT
STRING
F
e1 ~ e2
is_equal
cursor.item.id.is_equal(id)
cursor.item.is_equal(id)
o1.is_equal(
o1 ~

Programming with Expanded Types

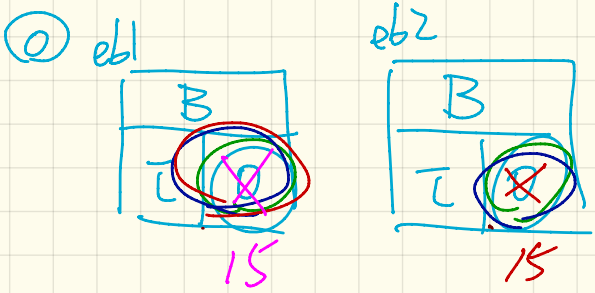
aliasing

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

S: STRING

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

eb1.change_i(15)
Result := eb1 /= eb2



class B
⋮
end

test
local
eb1, eb2: B
do
eb1 = eb2 T
create eb1.make(1)
create eb2.make(1)
eb1 = eb2 F

Point p1 = new Point(3, 4);

Point p2 = p1;

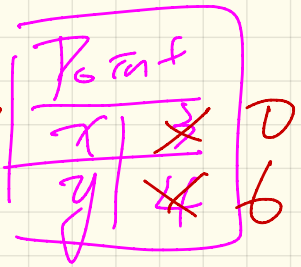
println(p1.x + " " + p1.y); 3, 4

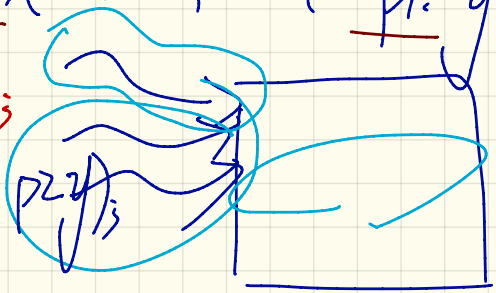
p2.moveUp(2);

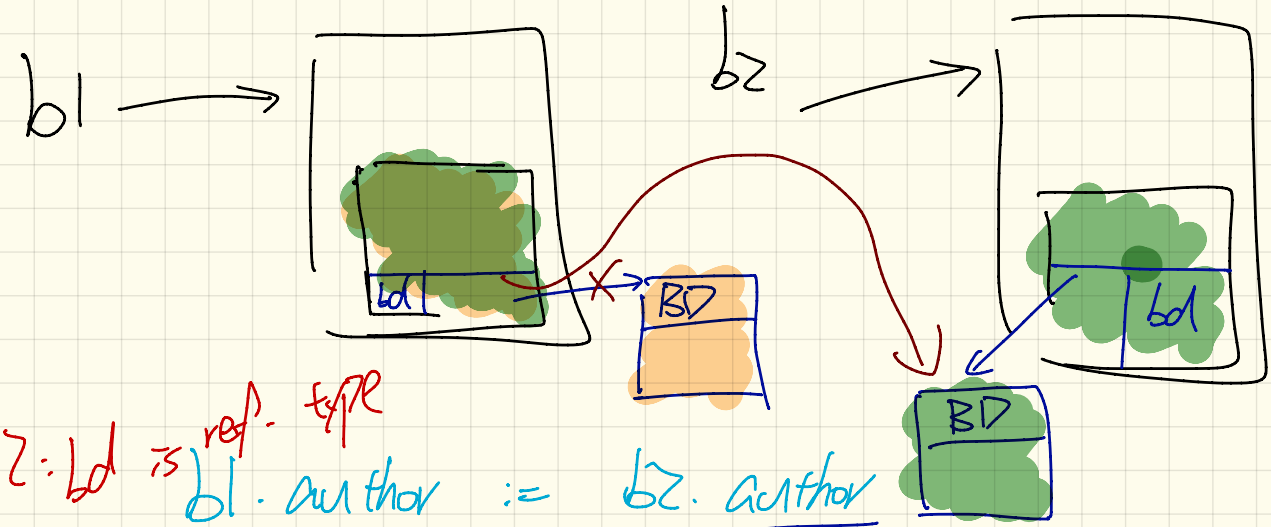
println(p1.x + " " + p1.y); 3, 6

p1.moveLeft(3);

println(p2.x + " " + p2.y); 0, 6



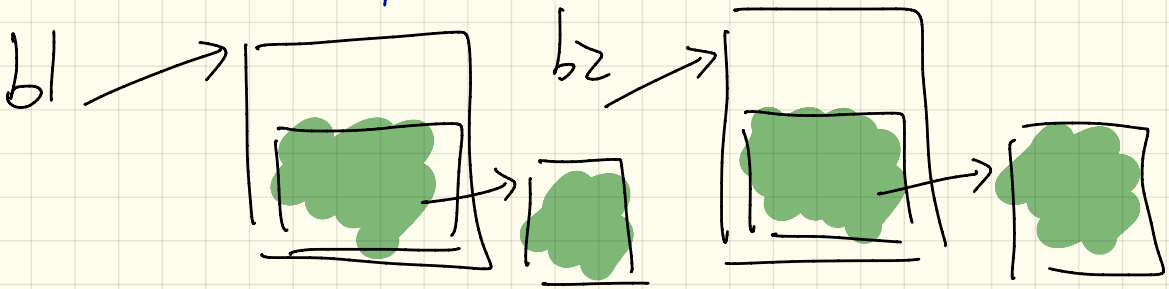


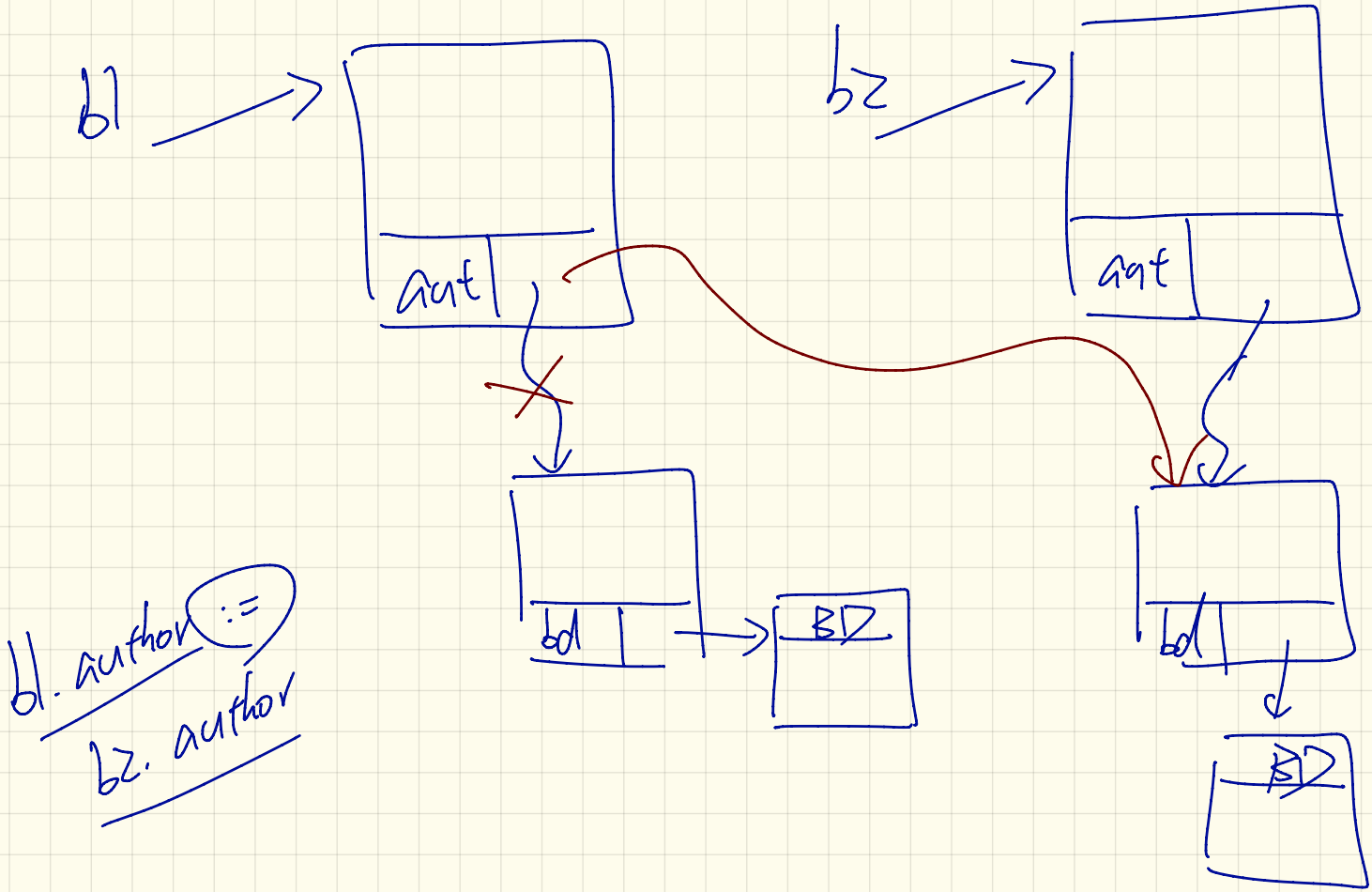


② Case 2: bd is ref. type

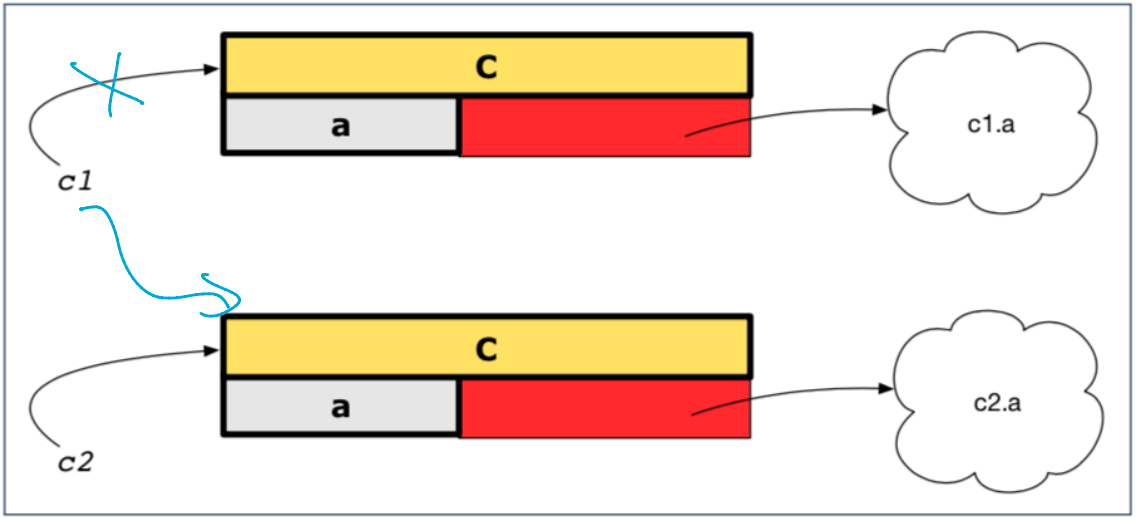
b1.author := b2.author

① Case 1 : bd is expanded type




b1
aut
b2
aut
bd
BD
bd
BD
b1. author := b2. author

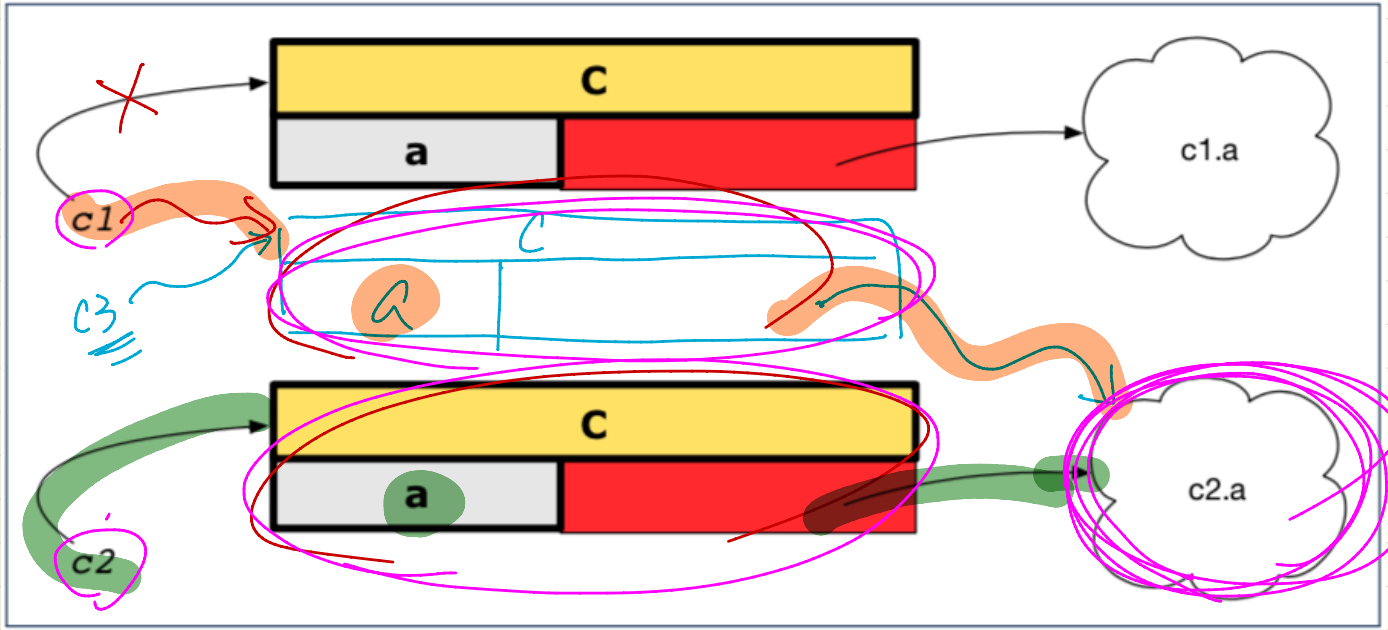
Reference Copy : $C_1 := C_2$



$C_1 = C_2$

$C_1.a = C_2.a$

Shallow Copy : C1 := C2.twin

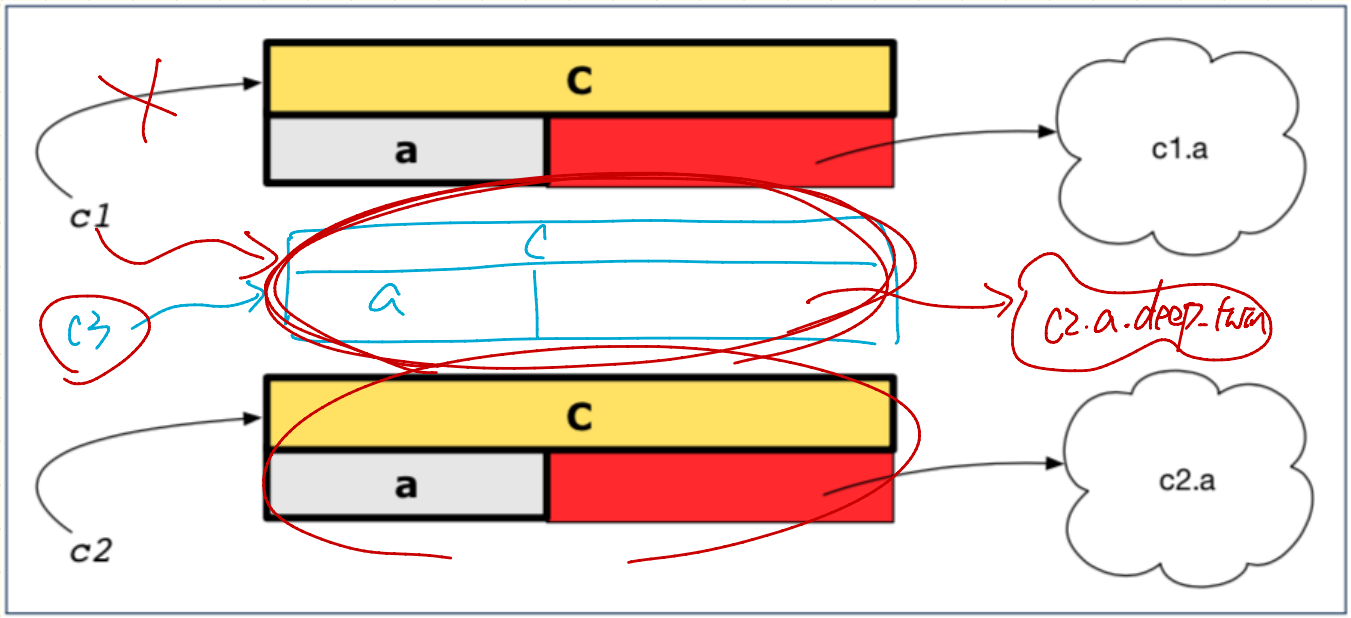


c3.a := c2.a

c1.a = c2.a T

c1 = c2 F

Deep Copy : c1 := c2.deep_twin



c3.a := c2.a.deep_twin

c1 = c2 F c1.a = c2.a F

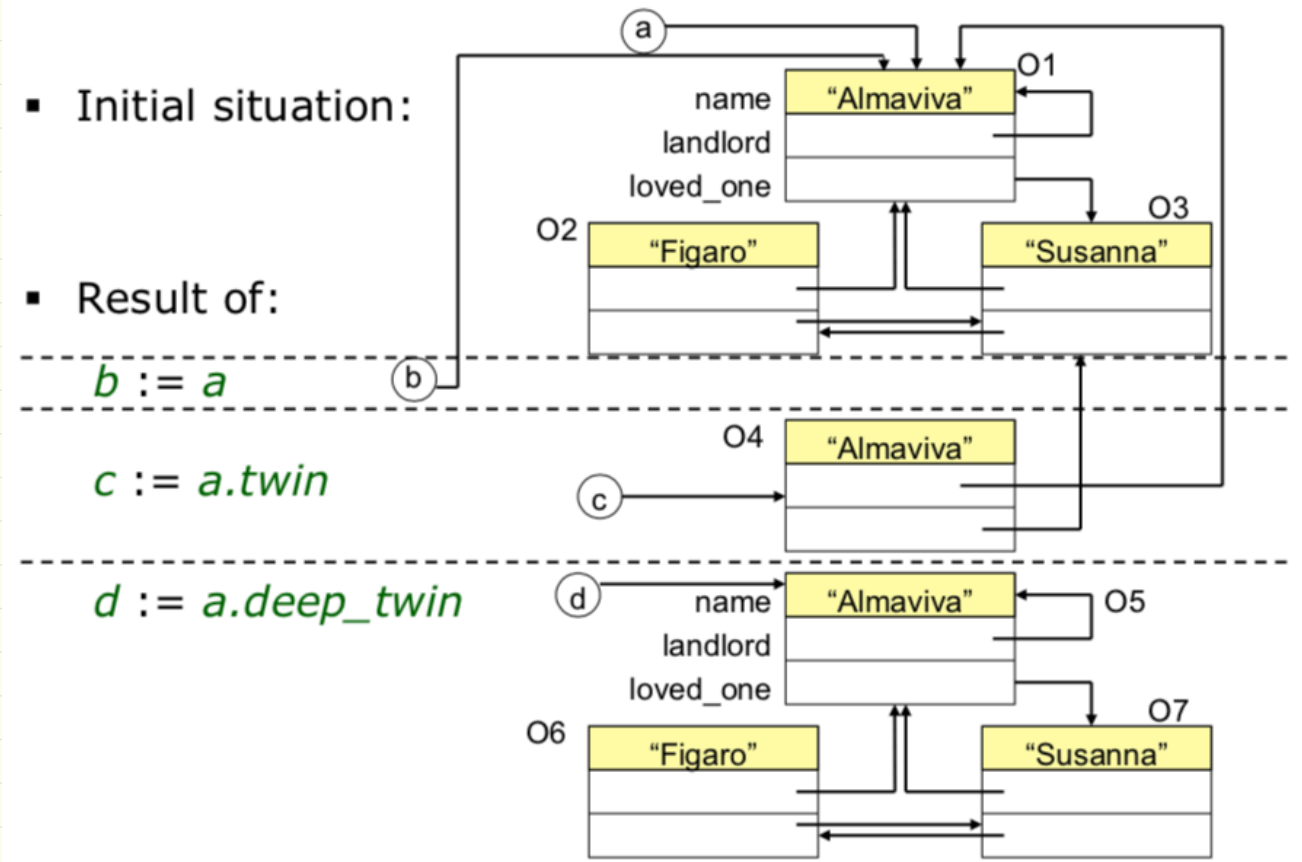
Ref. vs. Shallow vs. Deep Copies

- Initial situation:
- Result of:

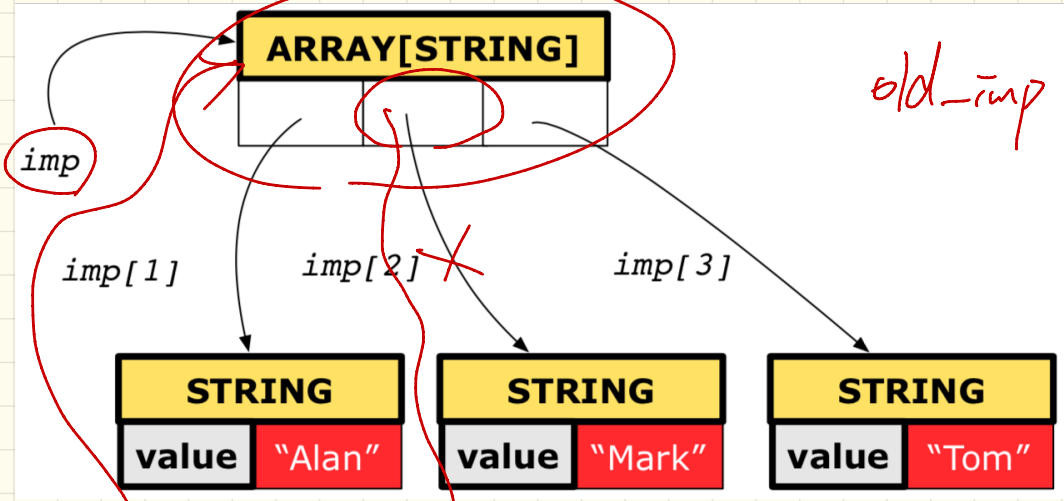
b := a

c := a.twin

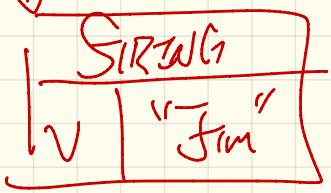
d := a.deep_twin



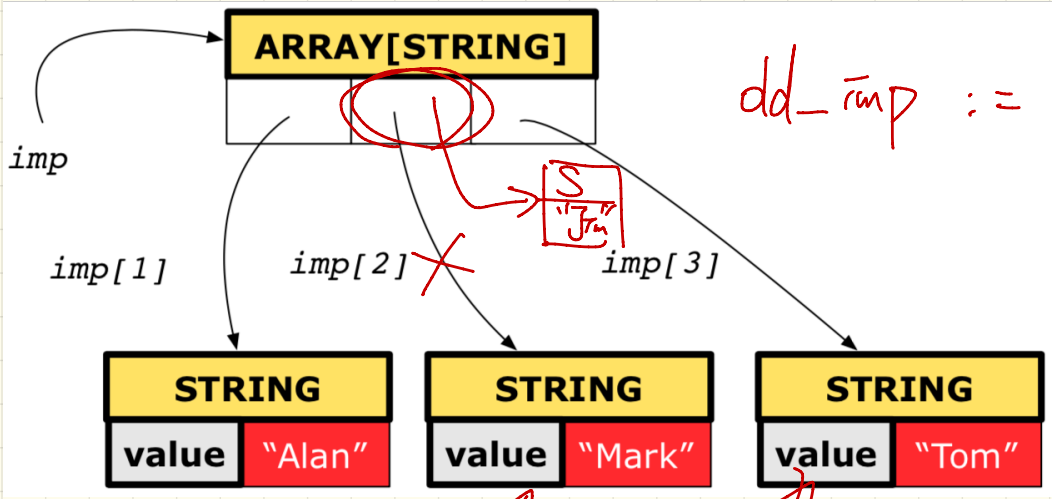
Copying Collection Objects: Reference Copy & Make Changes



old_imp := imp



Copying Collection Objects: Shallow Copy & Make 1st-level changes



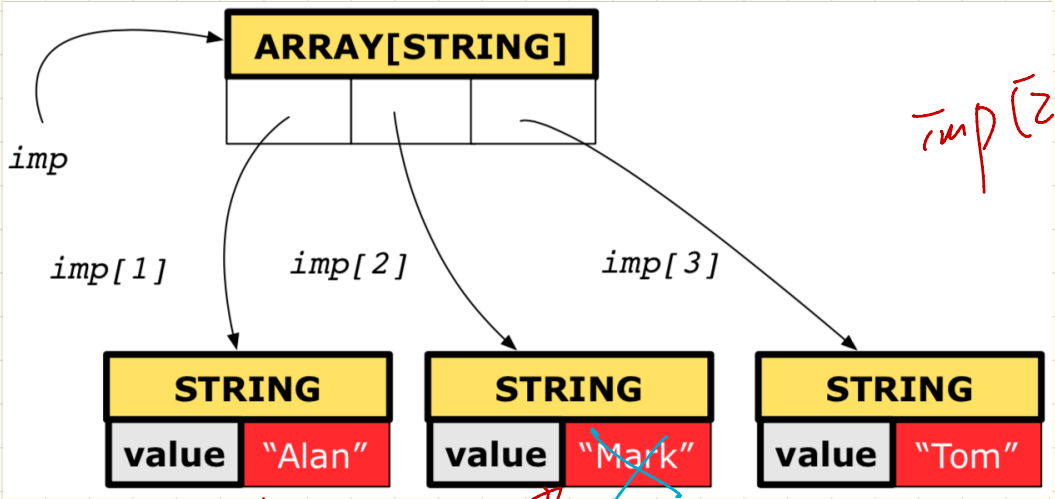
dd_imp := imp.twin

old_imp → [A[S]]

imp[1] = dd_imp[1] T

imp = dd_imp T.

Copying Collection Objects: Shallow Copy & Make 2nd-level changes



imp[2].append("***")

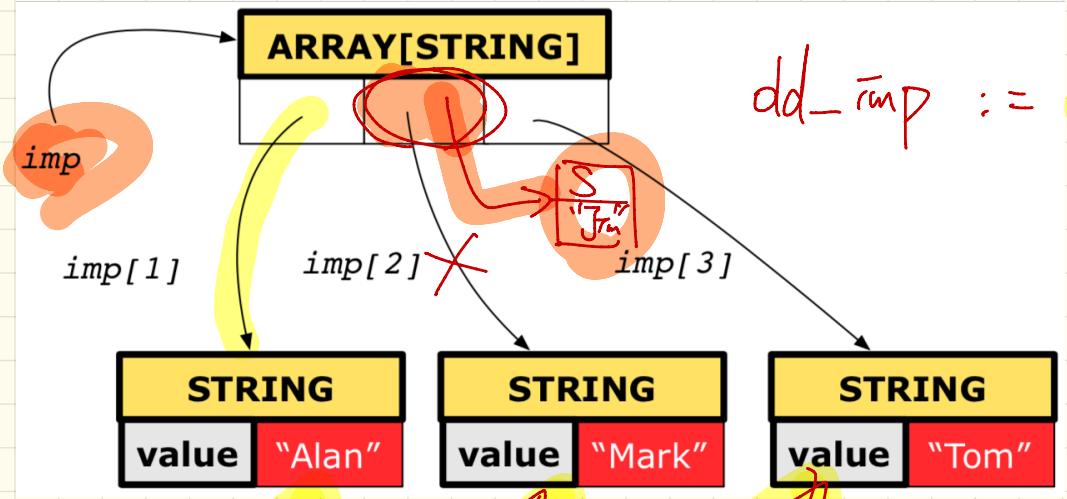
"Mark***"

old_imp → A[S]

Thursday Sep. 20

Lecture 5

Copying Collection Objects: Shallow Copy & Make 1st-level changes



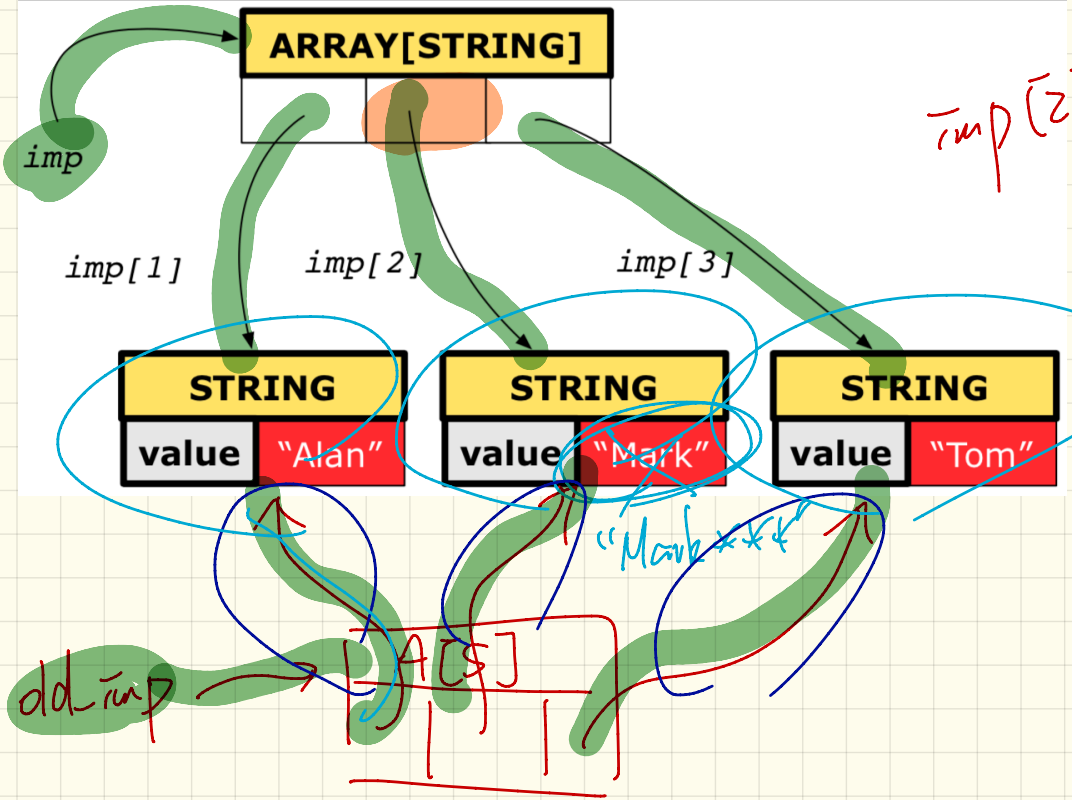
dd_imp := imp.twin

old_imp → A[S]

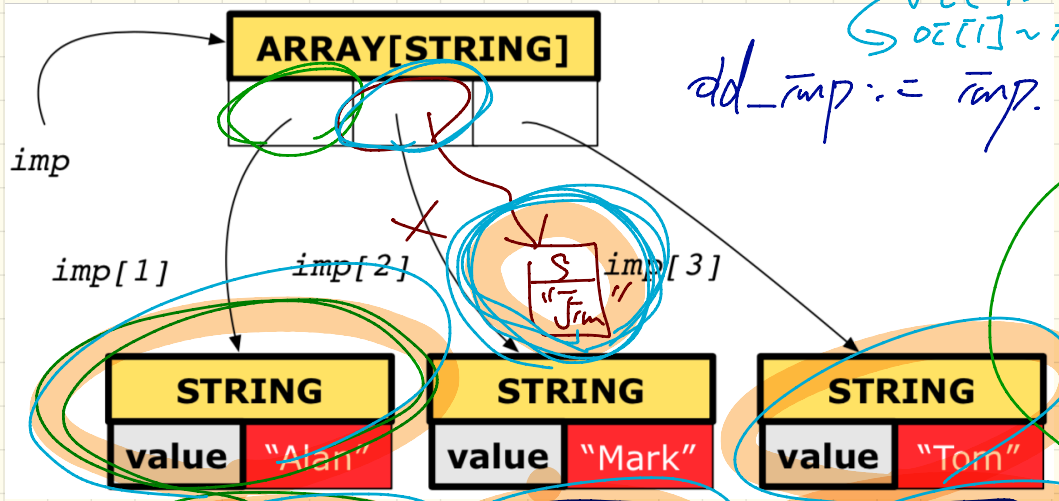
imp[1] = dd_imp[1] T

imp = dd_imp T.

Copying Collection Objects: Shallow Copy & Make 2nd-level changes



Copying Collection Objects: Deep Copy & Make 1st-level changes



$\forall i \in 1..3 \cdot old\_imp[i] \sim imp[i]$

$\hookrightarrow or[1] \sim imp[1]$ and $or[2] \sim imp[2]$ and $or[3] \sim imp[3]$

old_imp := imp.deep_twin

T
F
T

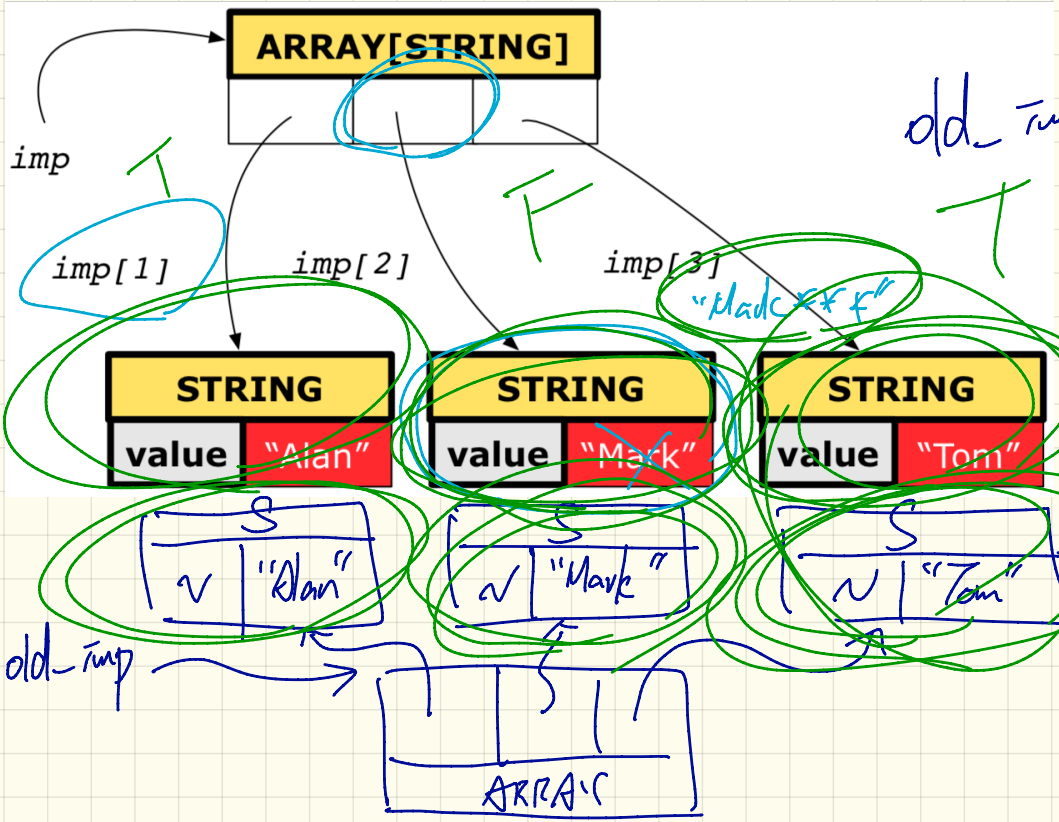
F

imp = old_imp F

imp[1] = old_imp[1]

imp[1] ~ old_imp[1] T

Copying Collection Objects: Deep Copy & Make 2nd-level changes



Caching Values for old Expressions in Postconditions

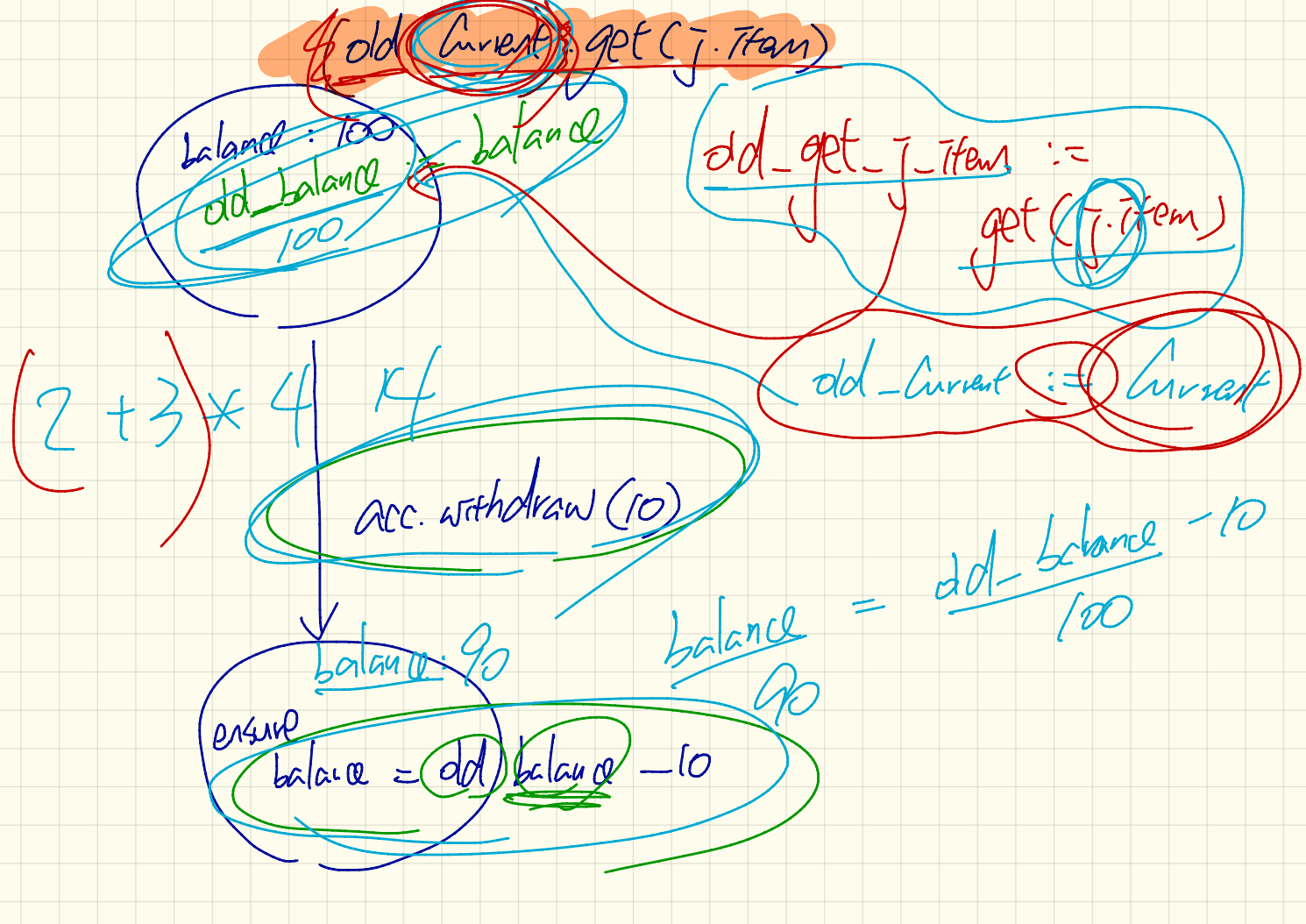
do x old ensure

ensure (current class Bank)	How to cache?
old balance = balance - a	old_balance := balance
old accounts[i].id	
(old accounts[i]).id	
(old accounts)[i].id	
(old current).accounts[i].id	

old accounts[i].id.item(2)

old_current := current
old_c_t := current.twin
old_c_d_t := current.dt

① old current
② old current.twin
③ old current.deep_twin


old (current).get (j.item)
balance : 100
old_balance
100
balance
old_get_j_item :=
get (j.item)
old_current := current
(2+3)*4
K
acc.withdraw (10)
balance : 90
balance
90
balance = old_balance - 10
100
ensure
balance = old balance - 10

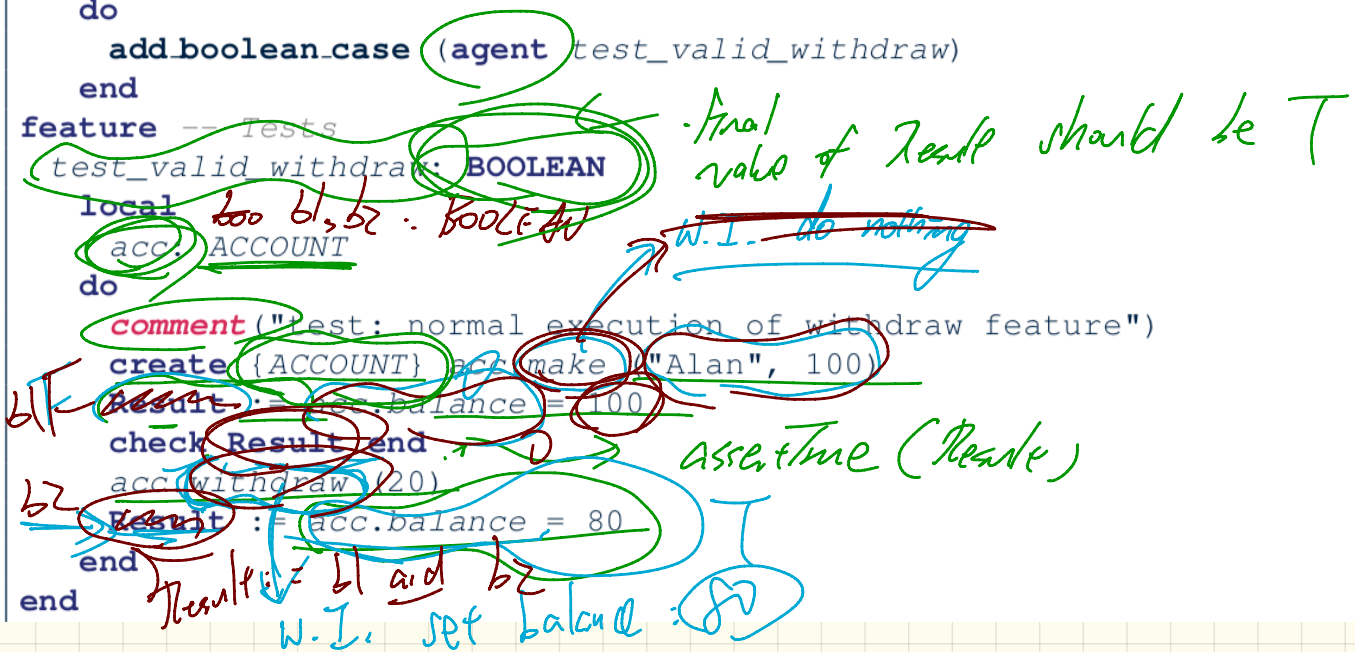
Use of old in across expression in Postcondition

```
class LINEAR_CONTAINER
create make
feature -- Attributes
  a: ARRAY[STRING]
feature -- Queries
  count: INTEGER do Result := a.count end
  get (i: INTEGER): STRING do Result := a[i] end
feature -- Commands
  make do create a.make_empty end
  update (i: INTEGER; v: STRING)
  do ...
  ensure -- Others Unchanged
     across
       1 |..| count as j
     all
       j.item /= i implies old get(j.item) ~ get(j.item)
     end
  end
end
```

Hint: What value will be cached at runtime before executing the imp. of update?

Test for Success

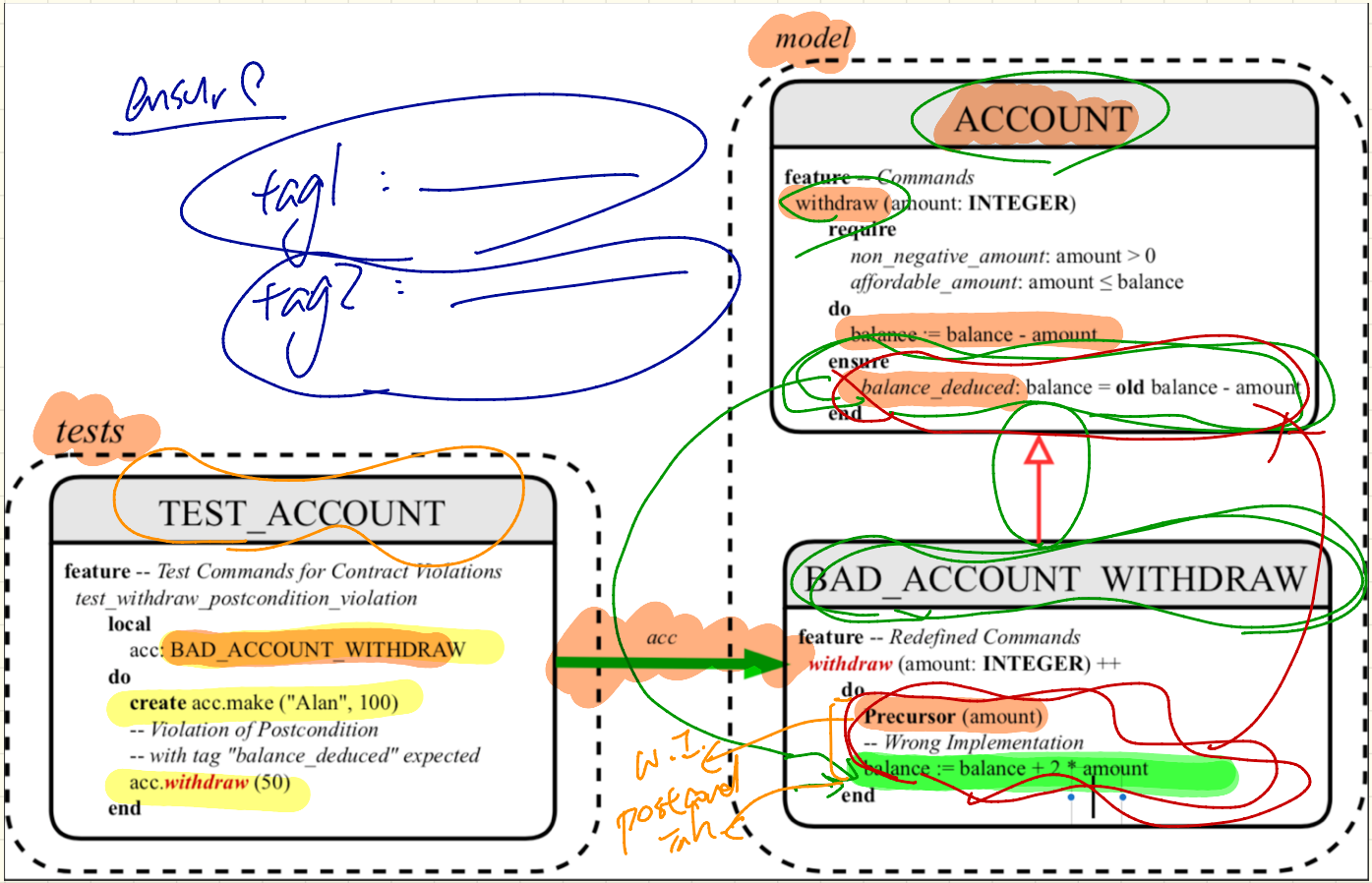
```
class TEST_ACCOUNT
inherit ES_TEST
create make
feature -- Add tests in constructor
  make
    do
      add_boolean_case (agent test_valid_withdraw)
    end
feature -- Tests
  test_valid_withdraw: BOOLEAN
    local
      acc: ACCOUNT
    do
      comment("test: normal execution of withdraw feature")
      create {ACCOUNT} acc.make ("Alan", 100)
      Result := acc.balance = 100
      check Result end
      acc.withdraw (20)
      Result := acc.balance = 80
    end
end
```



Test for Precondition Violation

```
class TEST_ACCOUNT
inherit ES_TEST
create make
feature -- Add tests in constructor
  make
    do
      add_violation_case_with_tag ("non_negative_amount",
        agent test_withdraw_precondition_violation)
    end
feature -- Tests
  test_withdraw_precondition_violation
    local
      acc: ACCOUNT
    do
      comment("test: expected precondition violation of withdraw")
      create {ACCOUNT} acc.make ("Mark", 100)
      -- Precondition Violation
      -- with tag "non_negative_amount" is expected.
      acc.withdraw (-1000000)
    end
end
```

Test for Postcondition Violation : Architecture



Test for Postcondition Violation: Code

```
class ACCOUNT
create
	make
feature -- Attributes
	owner : STRING
	balance : INTEGER
feature -- Constructors
	make(nn: STRING; nb: INTEGER)
		require -- precondition
			positive_balance: nb > 0
		do
			owner := nn
			balance := nb
		end
feature -- Commands
	withdraw(amount: INTEGER)
		require -- precondition
			non_negative_amount: amount > 0
			affordable_amount: amount <= balance -- problema
		do
			balance := balance - amount
		ensure -- postcondition
			balance_deducted: balance = old balance - amount
		end
invariant -- class invariant
	positive_balance: balance > 0
end
```

```
class TEST_ACCOUNT
inherit ES_TEST
create make
feature -- Constructor for adding tests
  make
    do
      add_violation_case_with_tag ("balance_deducted",
        agent test_withdraw_postcondition_violation)
    end
feature -- Test commands (test to fail)
  test_withdraw_postcondition_violation
    local
      acc: BAD_ACCOUNT_WITHDRAW
    do
      comment ("test: expected postcondition violation of withdraw")
      create acc.make ("Alan", 100)
      -- Postcondition Violation with tag "balance_deduced" to occur.
      acc.withdraw (50)
    end
end
```

```
class
  BAD_ACCOUNT_WITHDRAW
inherit
  ACCOUNT
    redefine withdraw end
create
  make
feature -- redefined commands
  withdraw(amount: INTEGER)
    do
      Precursor(amount)
      -- Wrong implementation
      balance := balance + 2 * amount
    end
end
```

Tuesday Sep. 25

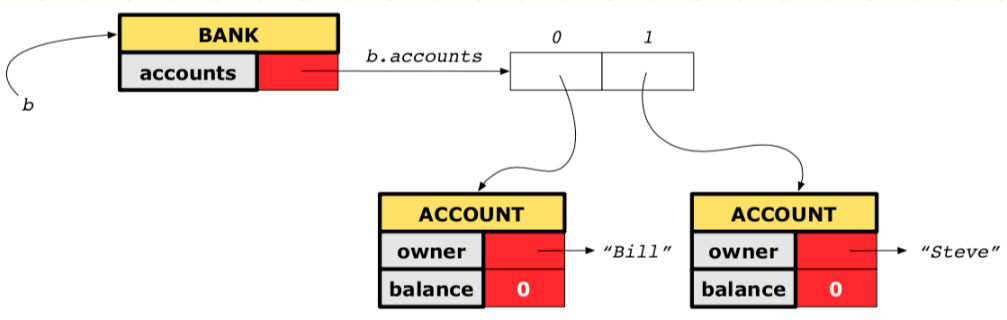
Lecture 6

MATRIX

TS_equal (other: ~~MATRIX~~ like Current) : BOOLEAN

Version 1 : Incomplete Contracts, Correct Implementation

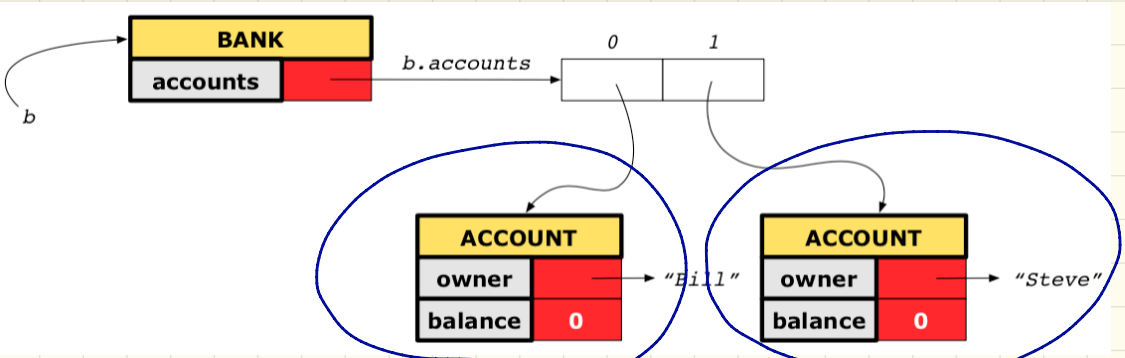
b.deposit ("Steve", 100)




```
class BANK
  deposit_on_v1 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      from i := accounts.lower
      until i > accounts.upper
      loop
        if accounts[i].owner ~ n then accounts[i].deposit(a) end
        i := i + 1
      end
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

Version 2 : Incomplete Contracts, Wrong Implementation

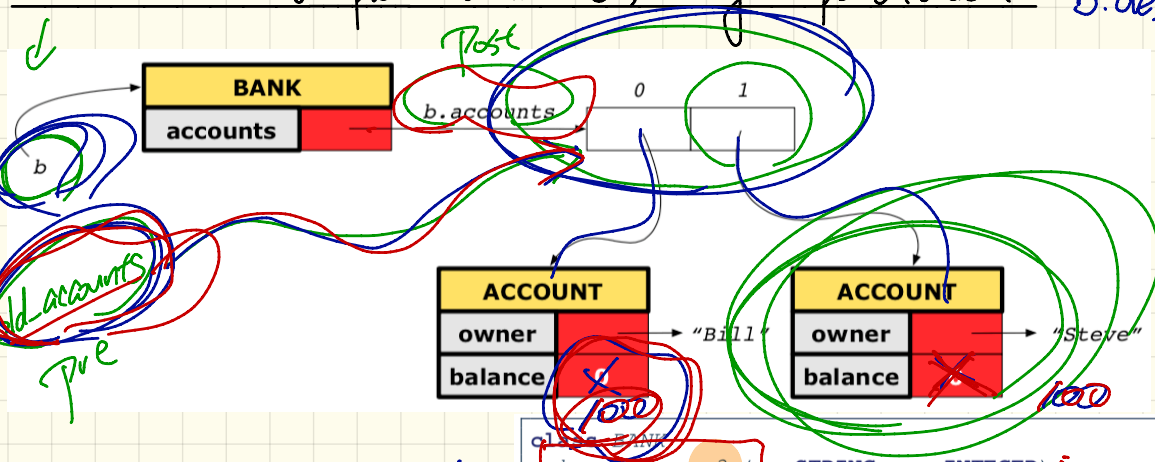
b.deposit ("Steve", 100)




```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1

      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

Version 3: Complete Contracts, Wrong Implementation (Reference Copy)

b.deposit("Steve", 100)



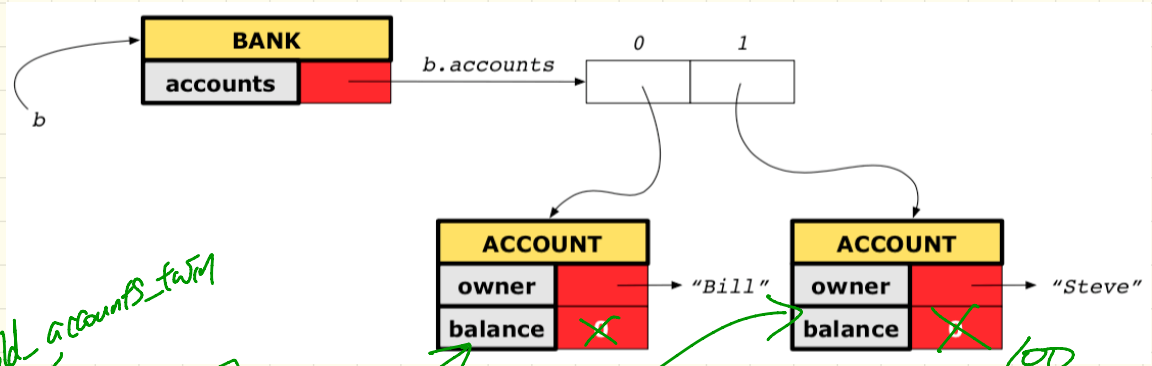
across 1..1 old (accounts.count - 1) as i all old accounts[i.item] ~ account_of(n) end

≡ True

```
class BANK
  deposit_on_v3 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of(n).balance = old account_of(n).balance + a
      others_unchanged:
        across old accounts as cursor
        all cursor.item.owner /~ n implies
          cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

Version 4: Complete Contracts, Wrong Implementation (shallow copy)

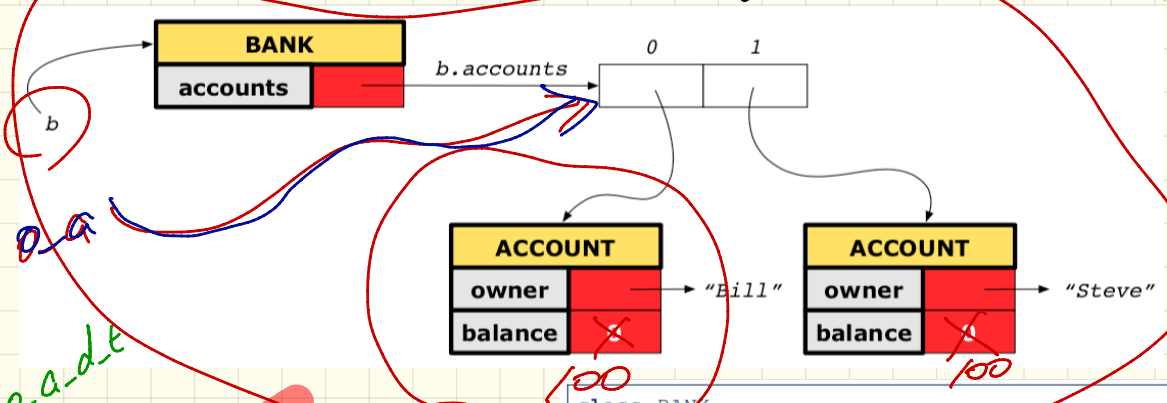
b.deposit ("Steve", 100)




```
class BANK
  deposit_on_v4 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
      others_unchanged :
        across old accounts.twin as cursor
        all cursor.item.owner /~ n implies
            cursor.item ~ account_of (cursor.item.owner)
      end
    end
end
```

Version 5: Complete Contracts, Wrong Implementation (Deep Copy)

b.deposit ("Steve", 100)



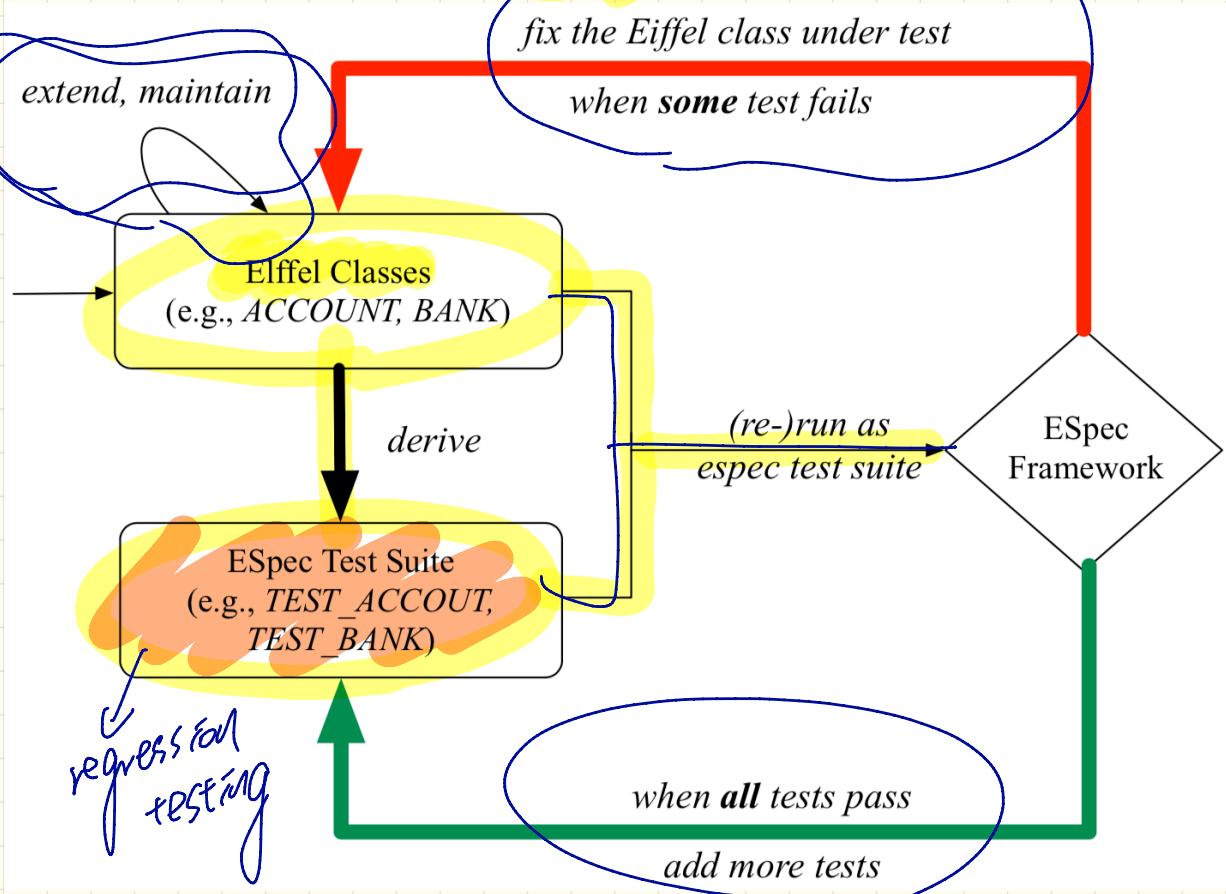
a_a

a_a_d_t

old accounts.deep_twin

```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
  do
    -- same loop as in version 1
    -- wrong implementation: also deposit in the first account
    accounts[accounts.lower].deposit(a)
  ensure
    num_of_accounts_unchanged: accounts.count = old accounts.count
    balance_of_n_increased:
      account_of (n).balance = old account_of (n).balance + a
    others_unchanged :
      across old accounts.deep_twin as cursor
      all cursor.item.owner /~ n implies
          cursor.item ~ account_of (cursor.item.owner)
    end
  end
end
```

Test-Driven Development (TDD)



$$\forall x \cdot P(x) \equiv \neg(\exists x : \neg P(x))$$

$$\exists x \cdot P(x) \equiv \neg(\forall x : \neg P(x))$$

student

x

$P(x)$

get A+

student

does not get A+

Stack of Strings vs. Stack of Accounts

```
class STRING_STACK
feature {NONE} -- Implementation
  imp: ARRAY[STRING] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
     -- Number of items on stack.
  top: STRING do Result := imp [i] end
     -- Return top of stack.
feature -- Commands
  push (v: STRING) do imp[i] := v; i := i + 1 end
     -- Add 'v' to top of stack.
  pop do i := i - 1 end
     -- Remove top of stack.
end
```

STACK[G]

```
class ACCOUNT_STACK
feature {NONE} -- Implementation
  imp: ARRAY[ACCOUNT] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
     -- Number of items on stack.
  top: ACCOUNT do Result := imp [i] end
     -- Return top of stack.
feature -- Commands
  push (v: ACCOUNT) do imp[i] := v; i := i + 1 end
     -- Add 'v' to top of stack.
  pop do i := i - 1 end
     -- Remove top of stack.
end
```

A Generic Stack

```
class STACK [G]
feature {NONE} -- Implementation
  imp: ARRAY[G] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
      -- Number of items on stack.
  top: G do Result := imp [i] end
      -- Return top of stack.
feature -- Commands
  push (v: G) do imp[i] := v; i := i + 1 end
      -- Add 'v' to top of stack.
  pop do i := i - 1 end
      -- Remove top of stack.
end
```

What happens if the client declares:

cl

s_s: STACK[STRING]

a_s: STACK[ACCOUNT]

Information Hiding Principle

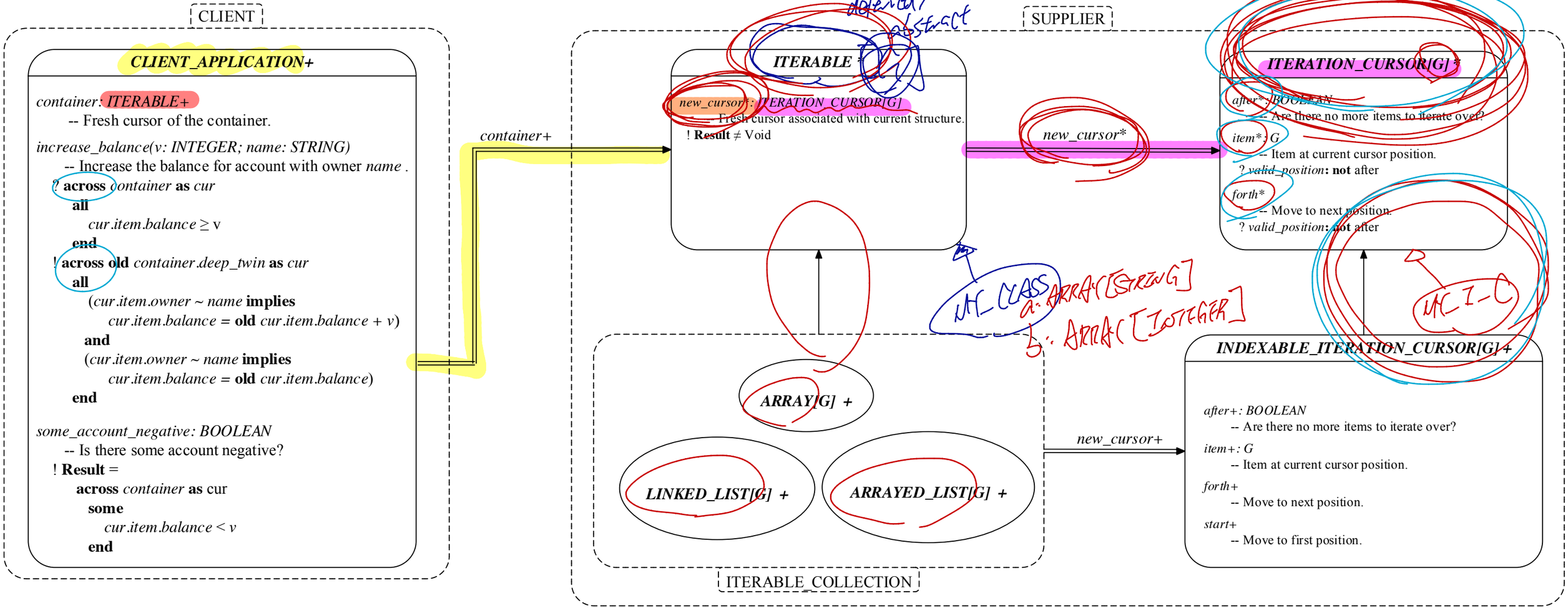
Supplier:

```
class
  CART
feature
  orders: ARRAY[ORDER]
end

class
  ORDER
feature
  price: INTEGER
  quantity: INTEGER
end
```

Client:

```
class
  SHOP
feature
  cart: CART
  checkout: INTEGER
    do
      from
        i := cart.orders.lower
      until
        i > cart.orders.upper
      do
        Result := Result +
          cart.orders[i].price
          *
          cart.orders[i].quantity
        i := i + 1
      end
    end
end
```

Problems with the current design?


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] +
ITERABLE_COLLECTION
new_cursor+
INDEXABLE_ITERATION_CURSOR[G] +
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.

Implementing the ITERATOR Pattern: Easy Case

```
class
    CART
inherit  ITERABLE [ORDER]
feature {NONE} -- Information Hiding
    orders: ARRAY [ORDER]

feature -- Iterable
    new_cursor : ITERATION_CURSOR [ORDER]
        do
            Result := orders.new_cursor
        end
end
```

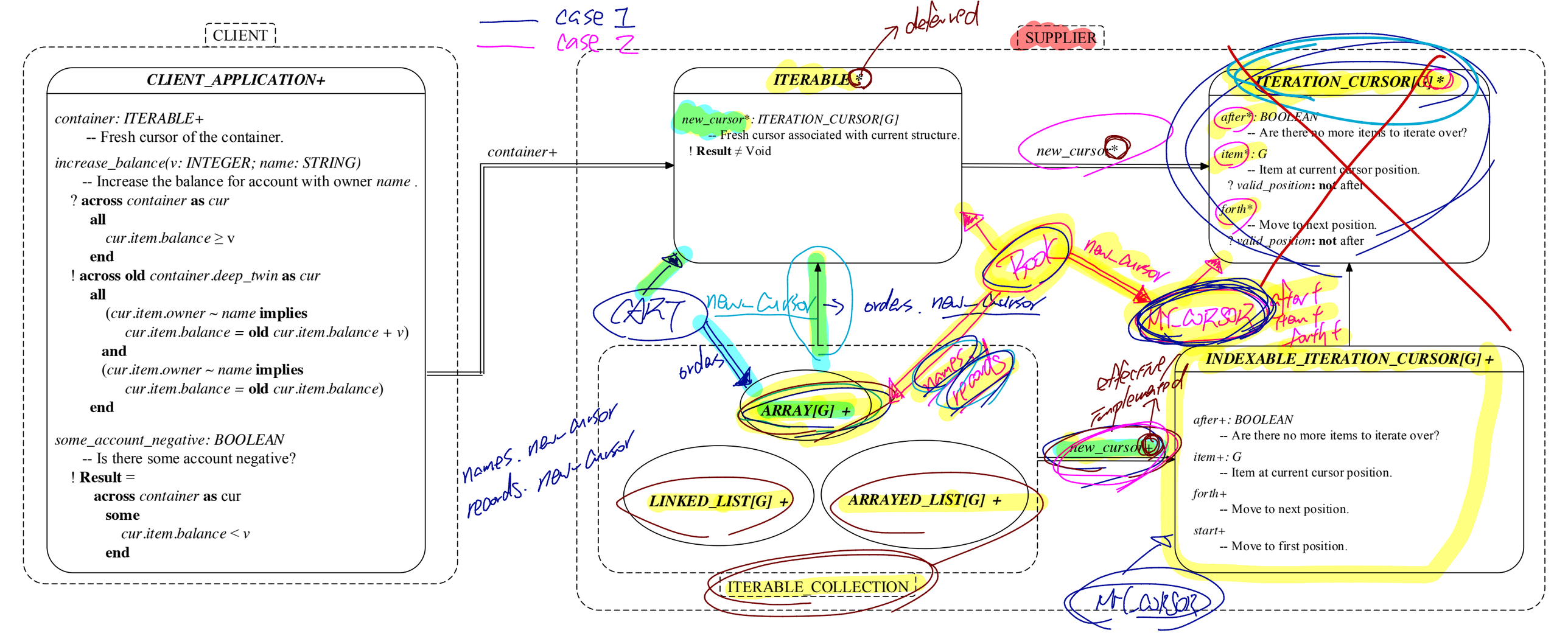
Implementing the ITERATOR Pattern: Hard Case

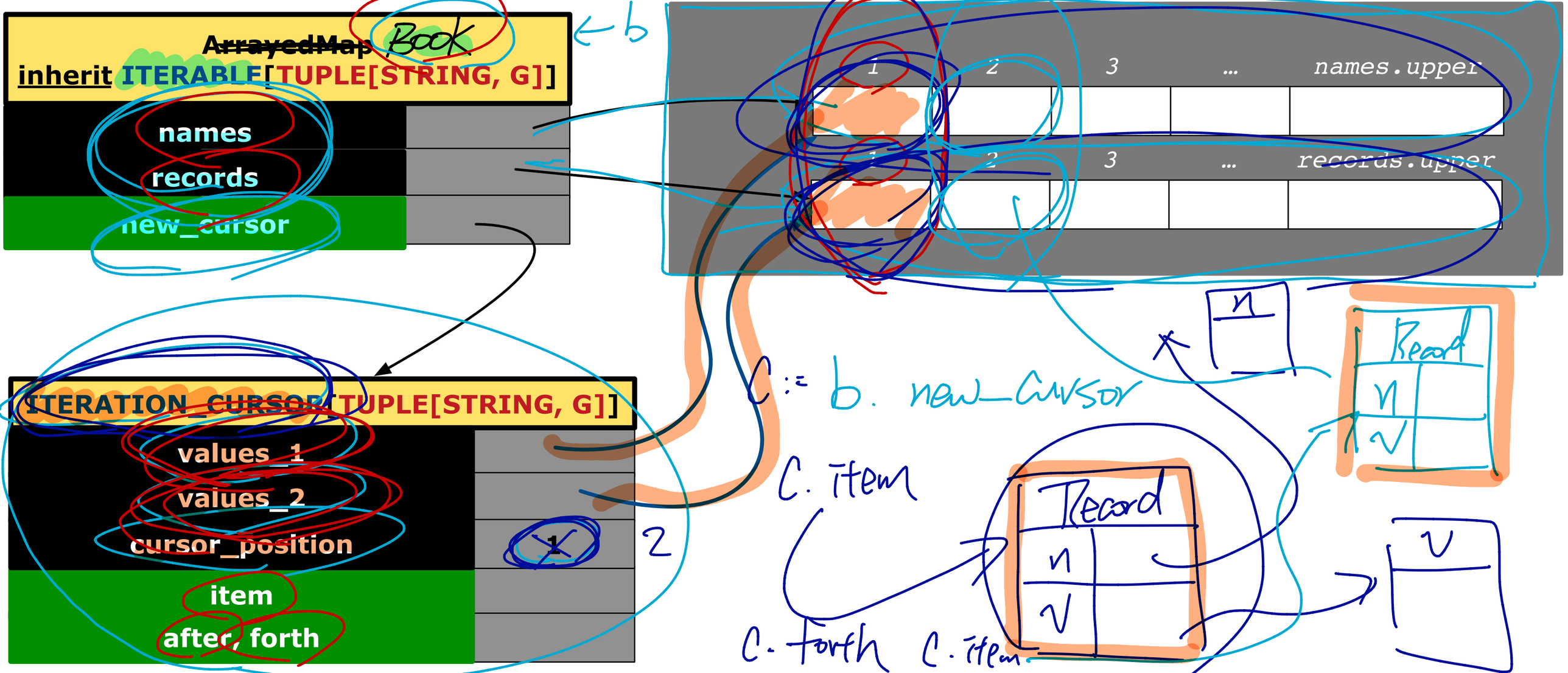
```
class
    BOOK[G]
inherit ITERABLE[ _____ ]
feature {NONE} -- Information Hiding
    names: ARRAY[STRING]
    records: ARRAY[G]

feature
    new_cursor: _____[_____]
        do

        end

end
```

TUPLE[STRING, G]

Thursday Sep. 27

Lecture 7


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
case 1
case 2
deferred
SUPPLIER
ITERABLE*
new_cursor*: ITERATION_CURSOR[G]
-- Fresh cursor associated with current structure.
! Result ≠ Void
container+
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
CART
new_cursor
orders.new_cursor
Rock
new_cursor
MY_CURSOR
after f
item f
forth f
orders
ARRAY[G] +
names
records
names.new_cursor
records.new_cursor
LINKED_LIST[G] +
ARRAYED_LIST[G] +
ITERABLE_COLLECTION
effective
implemented
new_cursor+
INDEXABLE_ITERATION_CURSOR[G] +
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.
MY_CURSOR

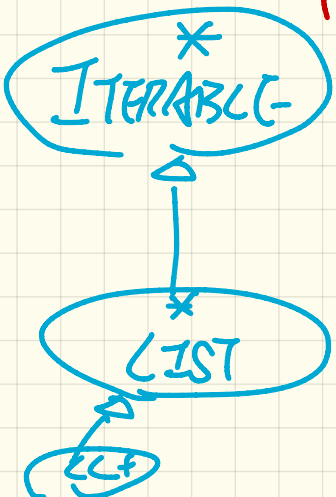

ArrayedMap
Book
inherit ITERABLE[TUPLE[STRING, G]]
names
records
new_cursor
b
1
2
3
...
names.upper
records.upper
ITERATION_CURSOR[TUPLE[STRING, G]]
values_1
values_2
cursor_position
item
after, forth
2
c := b.new_cursor
c.item
c.forth
c.item
n
v
Record

```
class
  CHECKER
feature -- Attributes
  collection: ITERABLE [INTEGER]
feature -- Queries
  is_all_positive: BOOLEAN
      -- Are all items in collection positive?
    do
      ...
    ensure
      across
        collection as cursor
      all
        cursor.item > 0
      end
  end
```

Inv

Programming with Interface vs. Implementation

change LIST to ITERABLE?




```
class BANK
...
  accounts: ITERABLE [ACCOUNT]
  binary_search (acc_id: INTEGER): ACCOUNT
      -- Search on accounts sorted in non-descending order.
    require
      across
        1 |..| (accounts.count - 1) as cursor
      all
        accounts [cursor.item].id <= accounts [cursor.item + 1].id
      end
    do
      ...
    ensure
      Result.id = acc_id
    end
```

Across

Use of ITERABLE vs. ITERATION_CURSOR

```
class BANK
 accounts: ITERABLE [ACCOUNT]
 max_balance: ACCOUNT
    -- Account with the maximum balance value.
  require ??
  local
   cursor: ITERATION_CURSOR[ACCOUNT]; max: ACCOUNT
  do
   from max := accounts [1]; cursor := accounts.new_cursor
   until cursor.after
   do
    if cursor.item.balance > max.balance then
      max := cursor.item
    end
    cursor.forth
   end
  ensure ??
  end
```

across accounts as cursor
loop
  f . . .
  cursor.item
  X cursor.forth
end

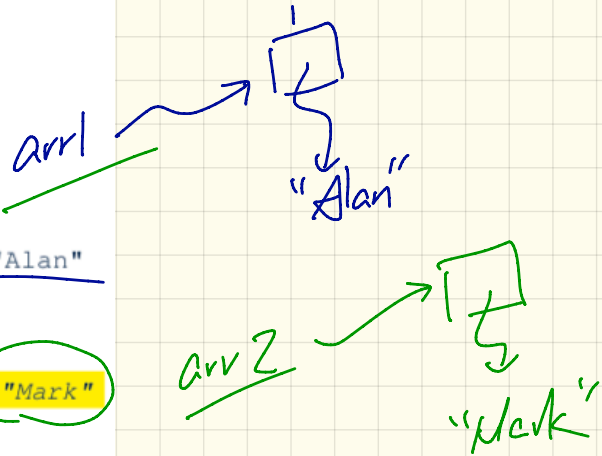
What's the equivalent use of across without any mention of the iteration cursor?

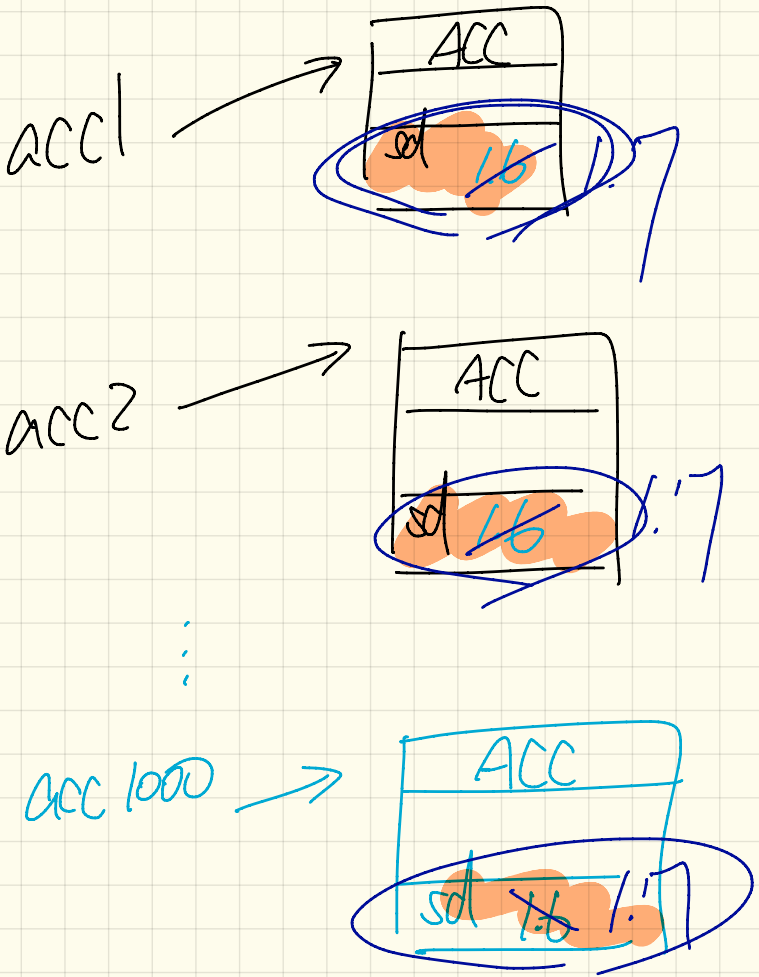
Non-Once Query

"Alan" "Mark"

```
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
```

```
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
```



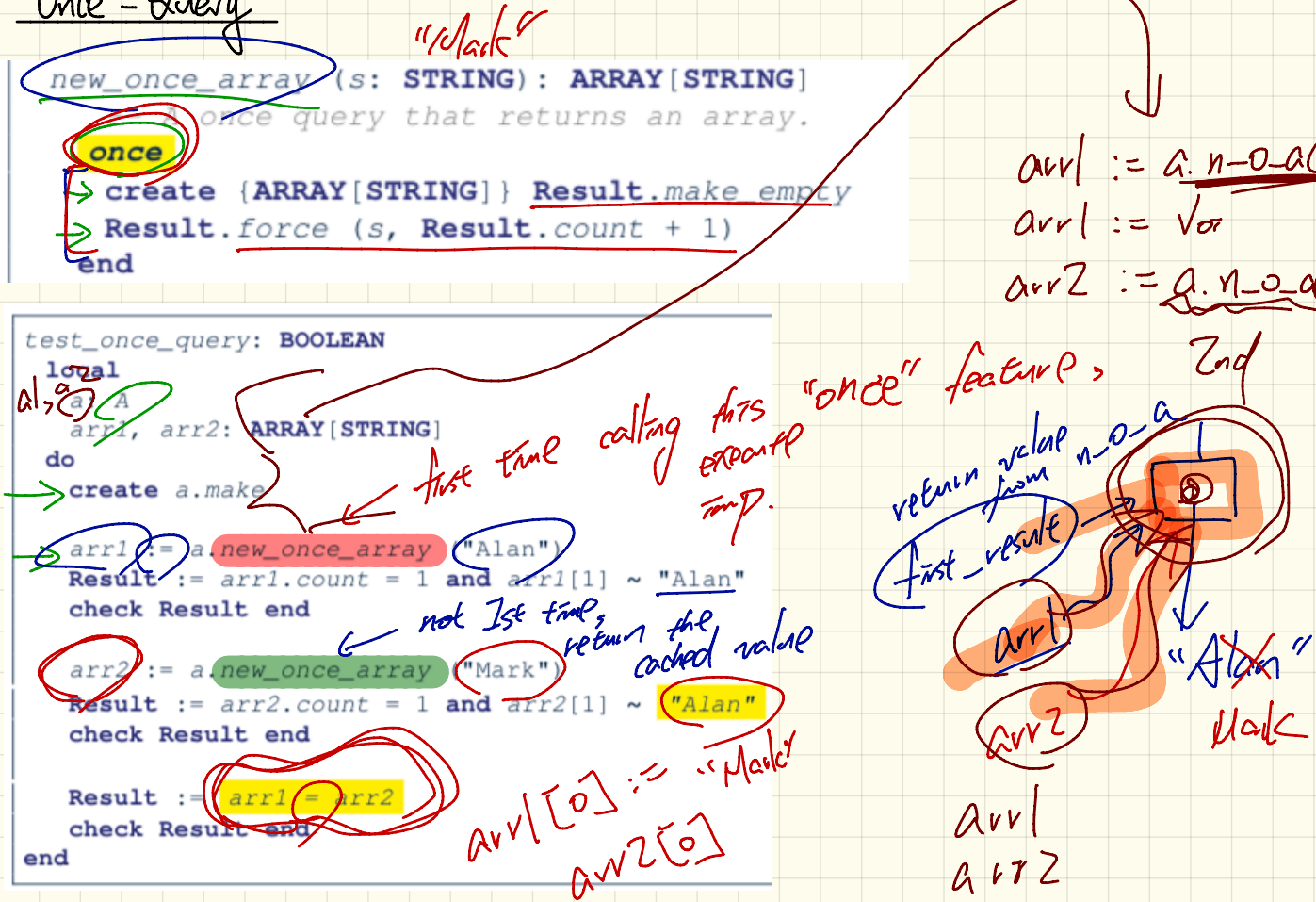


1.6 ↑ 1.7

Once - Query

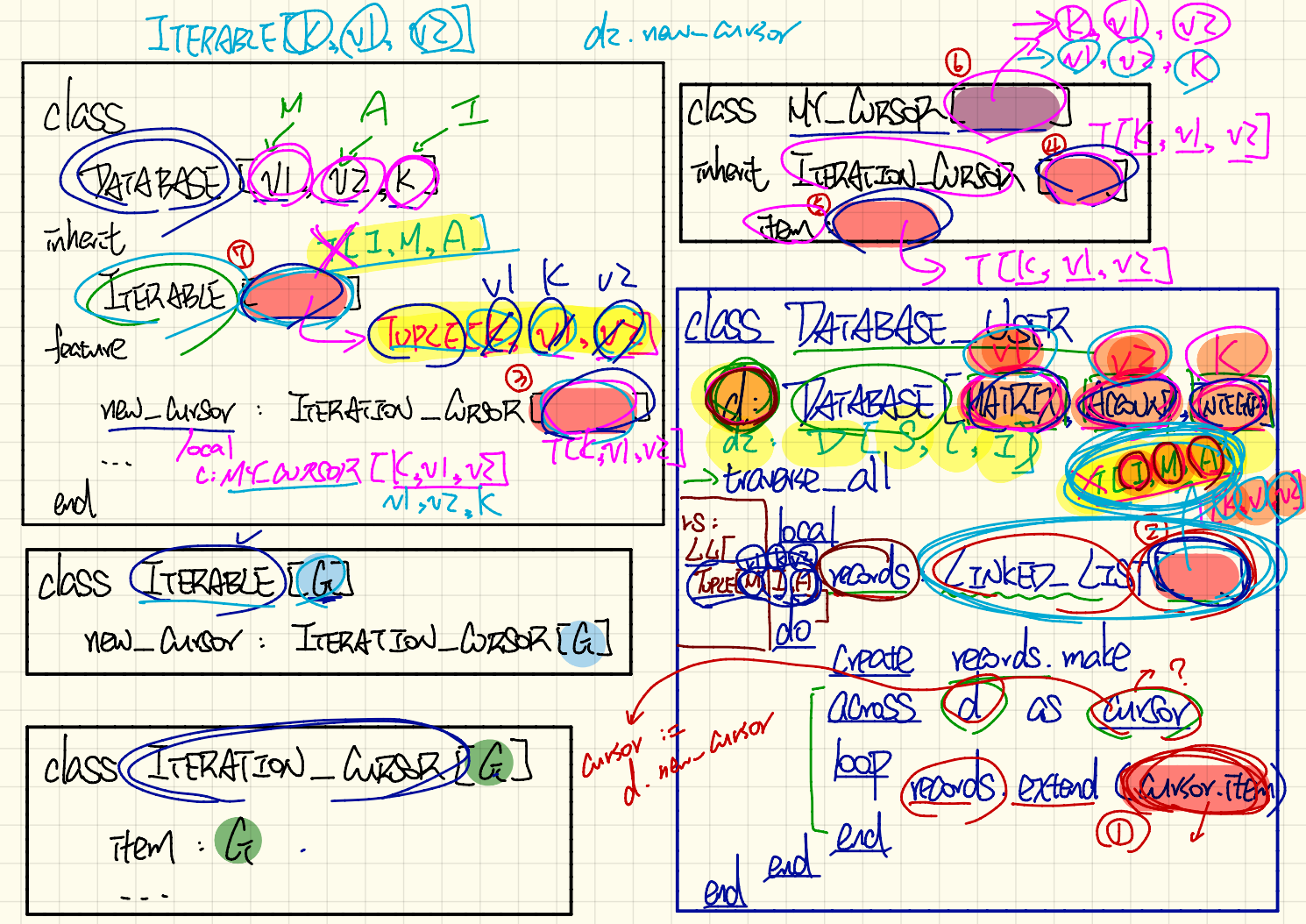
```
new_once_array (s: STRING): ARRAY[STRING]
    -- A once query that returns an array.
  once
    create {ARRAY[STRING]} Result.make_empty
    Result.force (s, Result.count + 1)
  end
```

```
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
```



Tuesday Oct. 2

Lecture 8


ITERABLE [K, V1, V2]
dz.new_cursor
class
DATABASE [V1, V2, K]
M A I
inherit
ITERABLE [I]
X [I, M, A]
feature
new_cursor : ITERATION_CURSOR [TUPLE [K, V1, V2]]
local
c: MY_CURSOR [K, V1, V2]
V1, V2, K
end
class ITERABLE [G]
new_cursor : ITERATION_CURSOR [G]
class ITERATION_CURSOR [G]
item : G
class MY_CURSOR [ ]
inherit ITERATION_CURSOR [ ]
item
T [K, V1, V2]
T [G, V1, V2]
class DATABASE_USER
dz : DATABASE [MATRIX, CHARACTER, INTEGER]
traverse_all
vs : LL [TUPLE [V1, V2, K]]
local
records
LINKED_LIST
do
create records.make
across dz as cursor
loop
records.extend (cursor.item)
end
end
cursor := dz.new_cursor
X [I, M, A]
V1, V2, K

Singleton Pattern: Code (1)

data ~ data

Supplier:

```
class DATA
create {DATA_ACCESS} make
feature {DATA_ACCESS}
  make do v := 10 end
feature -- Data Attributes
  v: INTEGER
  change_v (nv: INTEGER)
    do v := nv end
end
```

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

Client:

```
test: BOOLEAN
  local
    access: DATA_ACCESS
    d1, d2: DATA
  do
    d1 := access.data
    d2 := access.data
    Result := d1 = d2
      and d1.v = 10 and d2.v = 10
    check Result end
    d1.change_v (15)
    Result := d1 = d2
      and d1.v = 15 and d2.v = 15
  end
end
```

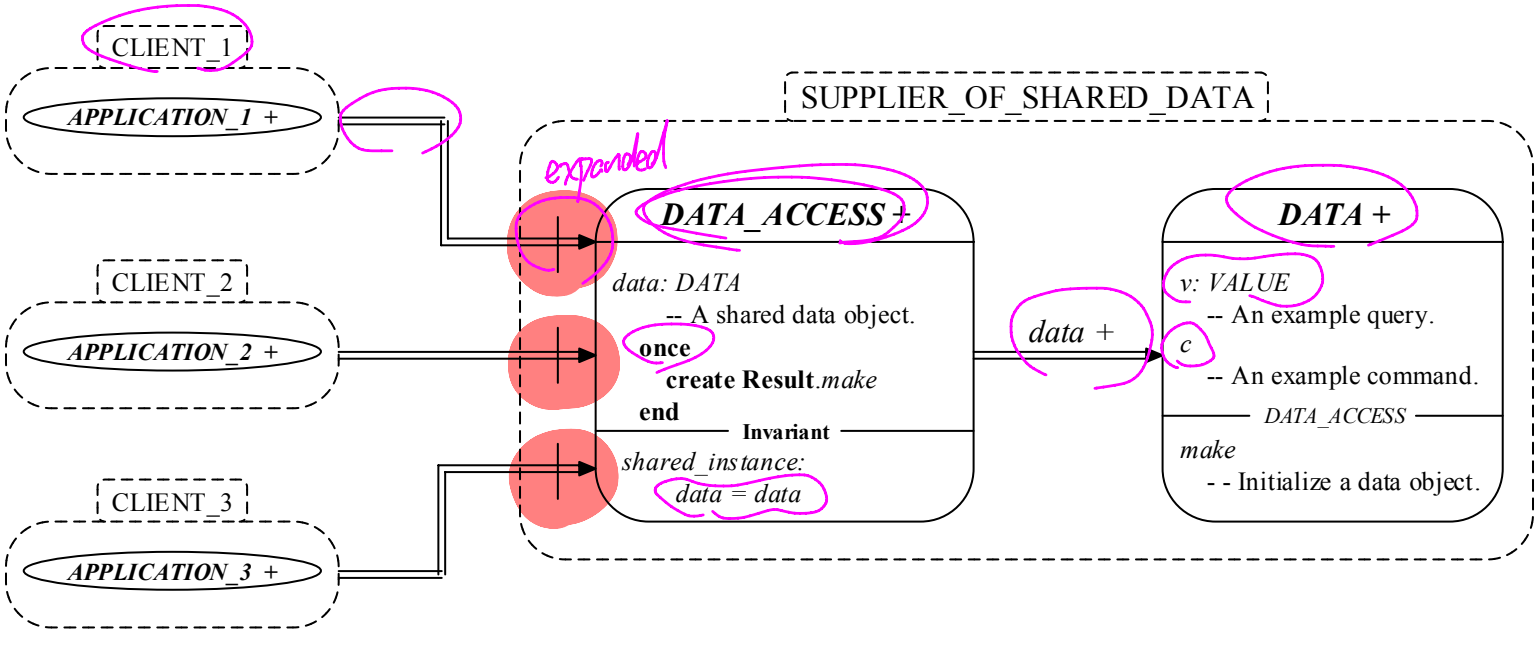
data : DATA
do
create data.make
end

create d1.make ✗ ①

d1.make ✗ ②

data
a call to q. 'data' ✓

a separate call to q. 'data'


CLIENT_1
APPLICATION_1 +
CLIENT_2
APPLICATION_2 +
CLIENT_3
APPLICATION_3 +
SUPPLIER_OF_SHARED_DATA
expanded
DATA_ACCESS +
data: DATA
-- A shared data object.
once
create Result.make
end
Invariant
shared_instance:
data = data
data +
DATA +
v: VALUE
-- An example query.
c
-- An example command.
DATA_ACCESS
make
- - Initialize a data object.

Singleton Pattern: Code (2)

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
```

create data.make ✗

Testing of Singleton Pattern

```
test_bank_shared_data: BOOLEAN
    -- Test that a single data object is manipulated
  local acc1, acc2: ACCOUNT
  do
    comment("t1: test that a single data object is shared")
    create acc1.make ("Bill")
    create acc2.make ("Steve")
    Result := acc1.data = acc2.data
    check Result end
    Result := acc1.data ~ acc2.data
    check Result end
    acc1.data.set_interest_rate (3.11)
    Result :=
          acc1.data.interest_rate = acc2.data.interest_rate
      and acc1.data.interest_rate = 3.11
    check Result end
    acc2.data.set_interest_rate (2.98)
    Result :=
          acc1.data.interest_rate = acc2.data.interest_rate
      and acc1.data.interest_rate = 2.98
  end
```

1st time data in DATA_Access is called

2nd time

Violation of Single Choice Principle

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * discount_rate
    end
end
```

STUDENT

resident: BOOLEAN

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * premium_rate
    end
end
```

change or register

Thursday Oct. 4

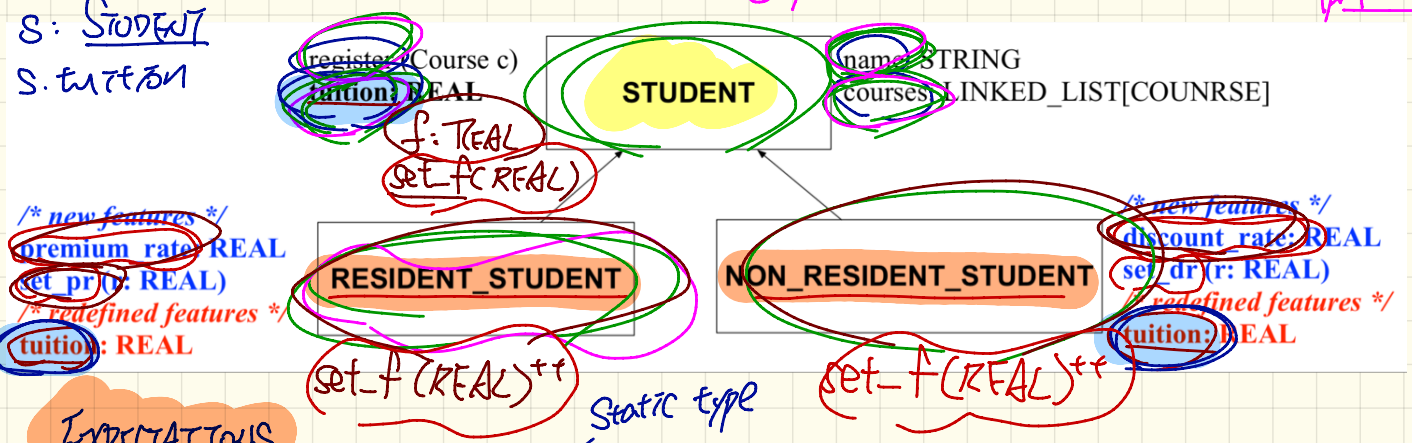
Lecture 9

- Lab 2 (next Sat)
- Lab 3 (next Wed)
- Midterm: Oct. 24 Wednesday

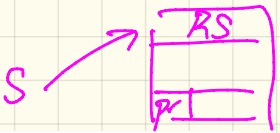
After reading week:

Tue Oct. 16 (will be covered)
Thu Oct. 18 (covered? to be confirmed)

Inheritance Architecture : Students



create {RS} s.make



s: STUDENT

s.tuition

f: REAL

set_f(REAL)

set_f(REAL)++

set_f(REAL)++

EXPECTATIONS

Static type

	s: STUDENT	rs: RESIDENT_STUDENT	nrs: NON_RESIDENT_STUDENT
callable features	s.name s.cs s.reg s.tuition s.pr s.set_pr	rs.name rs.cs rs.reg rs.tuition rs.pr rs.set_pr rs.dr ✗	nrs.name nrs.cs nrs.reg nrs.tuition nrs.dr nrs.set_dr

rs: RS

s: STUDENT

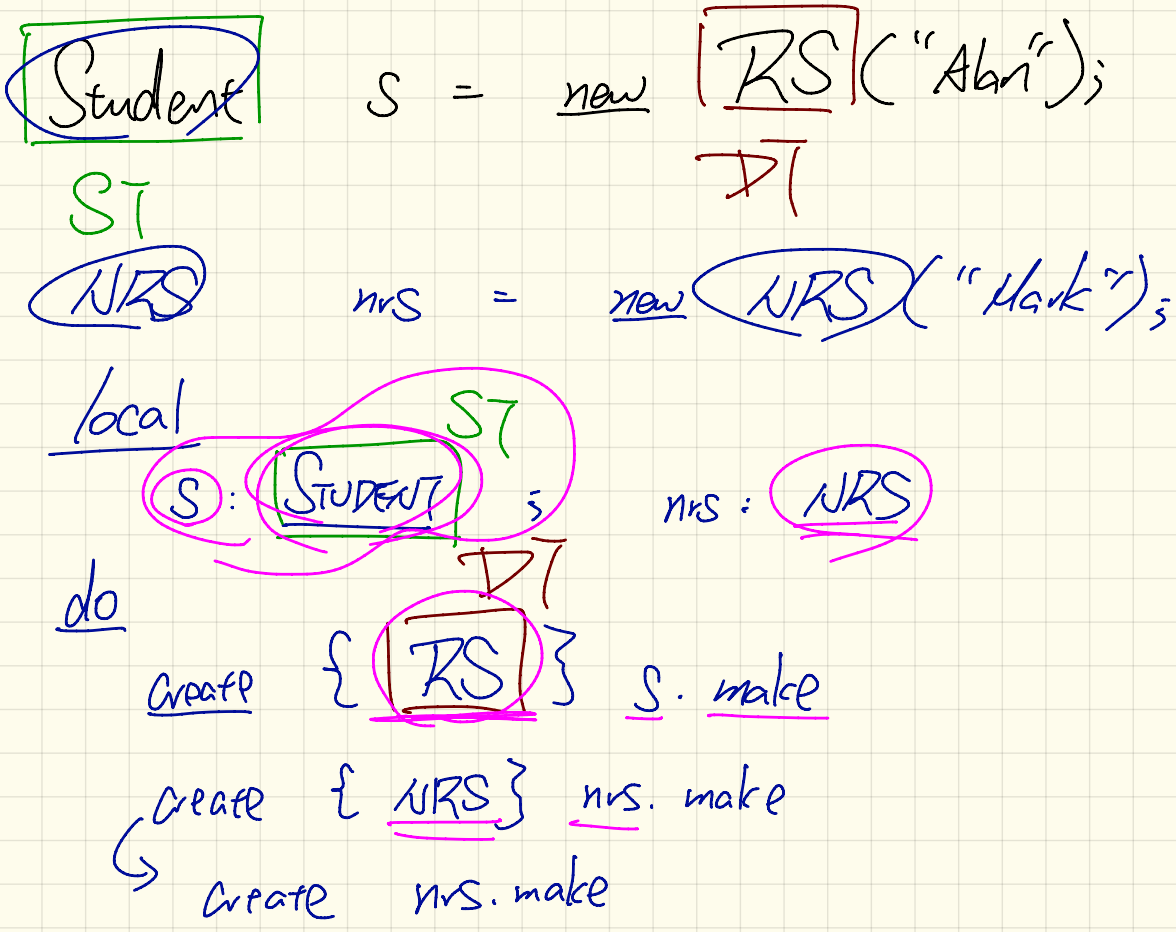
create {RS} rs.make

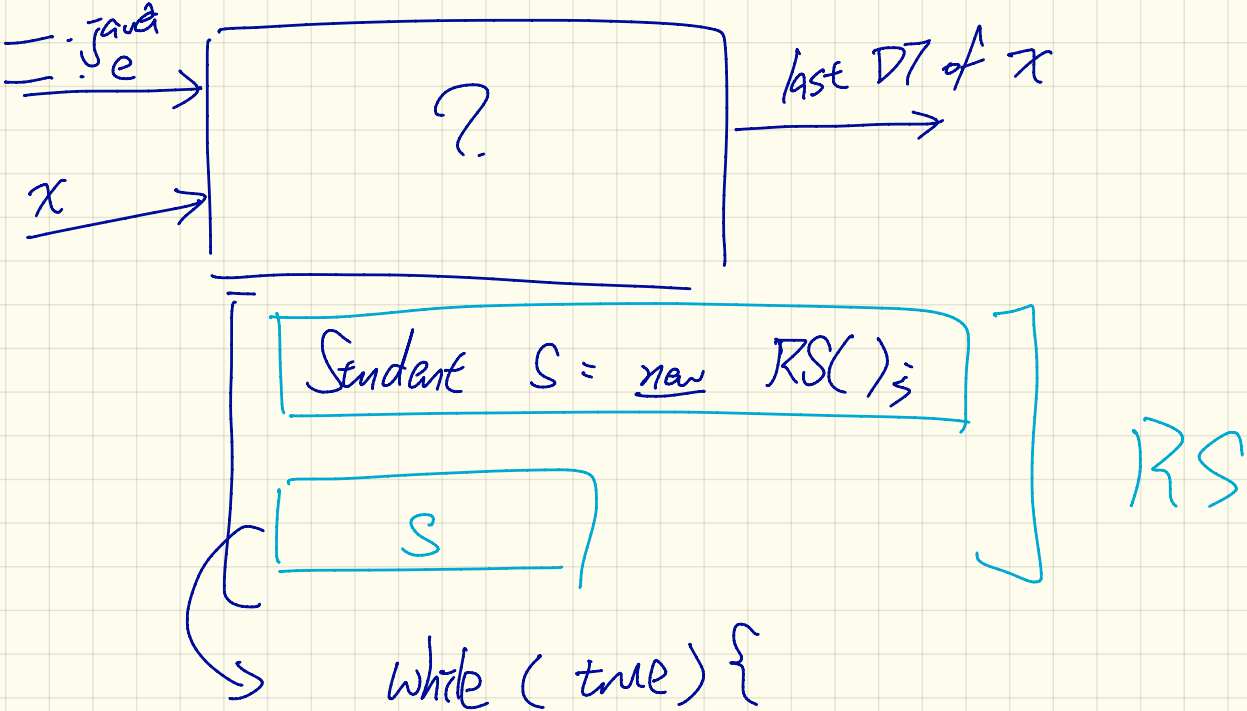
create {RS} s.make

s.f ← inherited version of STUDENT

s.set_f

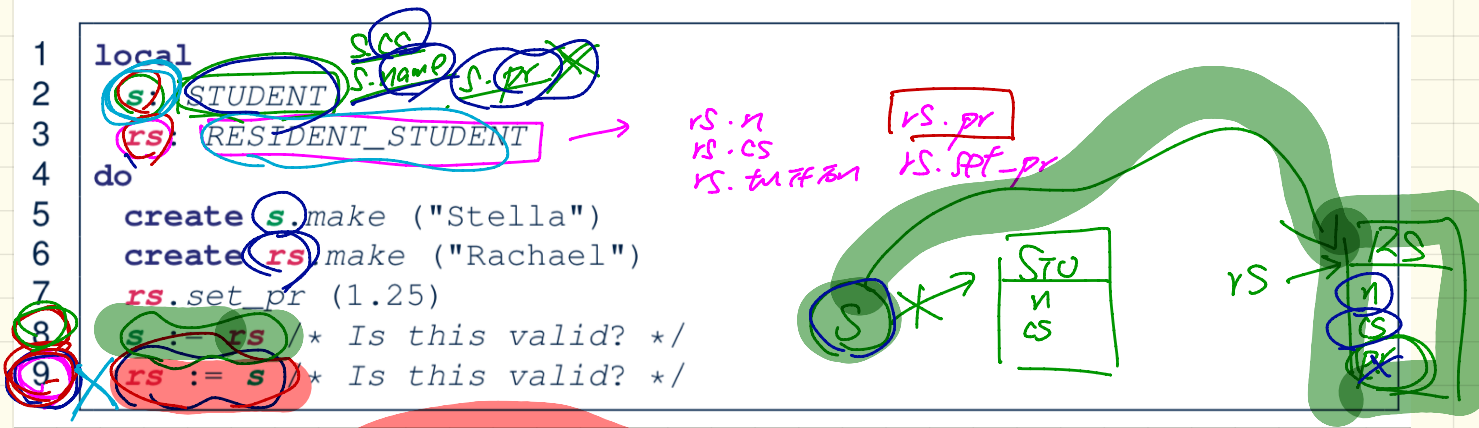
s := rs ← s.set_f

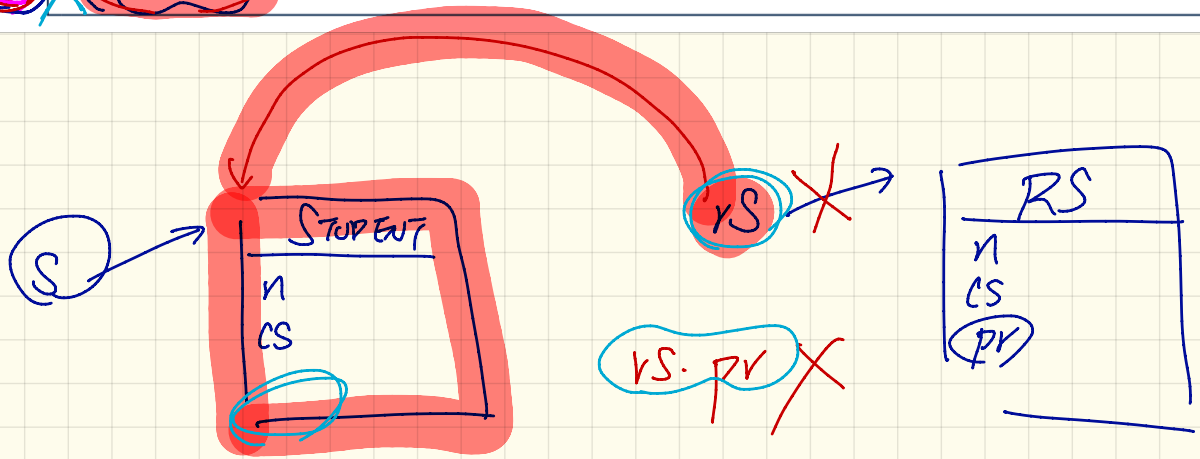

Student
s = new RS("Alan");
ST
DT
NRS
nrs = new NRS("Mark");
local
s: STUDENT;
nrs: NRS
do
create {RS} s.make
create {NRS} nrs.make
create nrs.make


.java
.e
x
?
last DT of x
Student S = new RS();
S
RS
while (true) {
Student S = new RS();
}

Polymorphism: Intuition

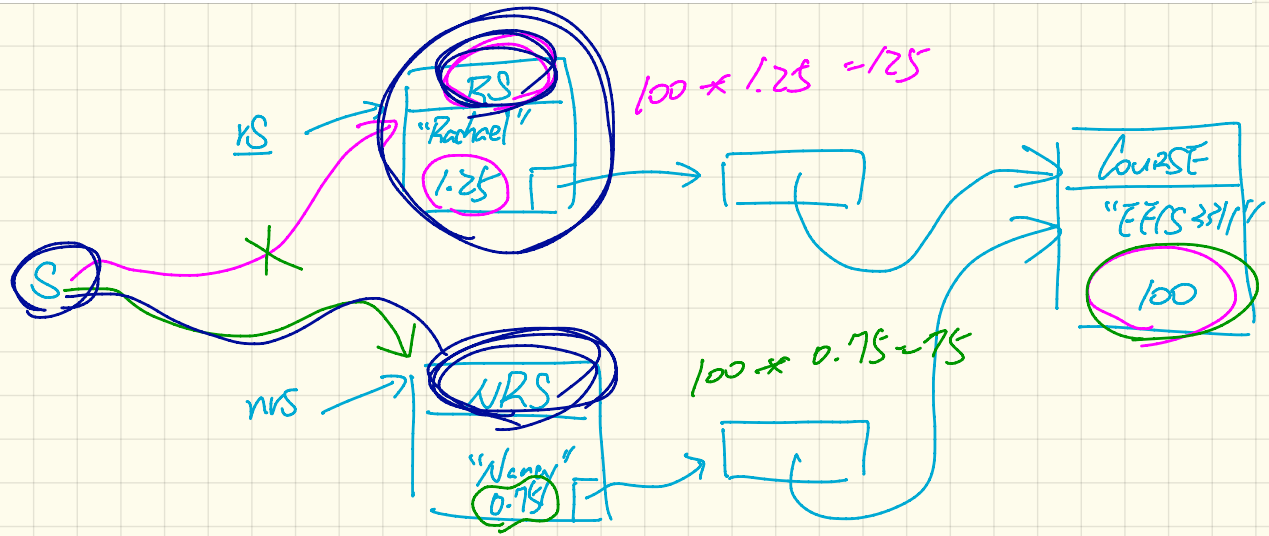
```
1  local
2    s: STUDENT
3    rs: RESIDENT_STUDENT
4  do
5    create s.make ("Stella")
6    create rs.make ("Rachael")
7    rs.set_pr (1.25)
8    s := rs  /* Is this valid? */
9    rs := s  /* Is this valid? */
```





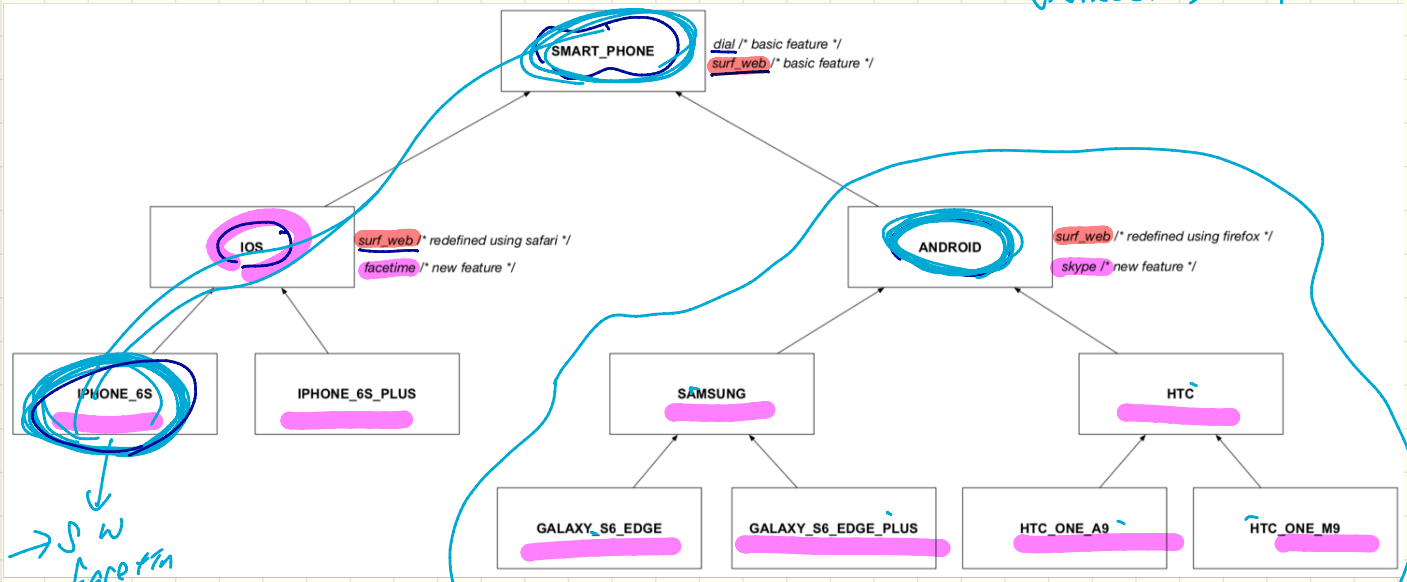
Dynamic Binding: Intuition

```
local c : COURSE ; s : STUDENT
do crate c.make ("EECS3311", 100.0)
 → create {RESIDENT_STUDENT} rs.make("Rachael")
 → create {NON_RESIDENT_STUDENT} nrs.make("Nancy")
   rs.set_pr(1.25); rs.register(c)
 → nrs.set_dr(0.75); nrs.register(c)
   s := rs ; check s.tuition = 125.0 end
 → s := nrs ; check s.tuition = 75.0 end
```



Inheritance Hierarchy : SmartPhones

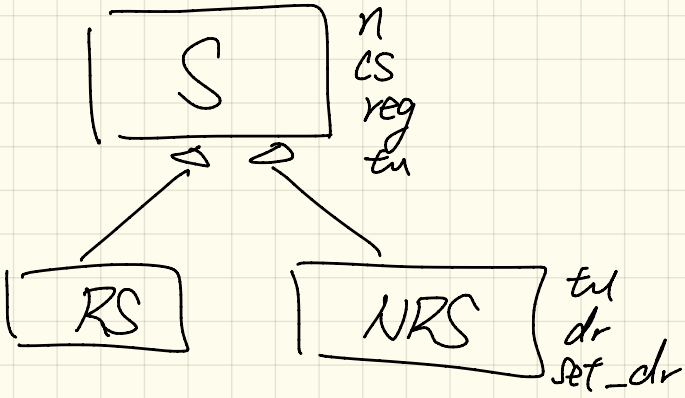
ancestors for ANDROID:
ANDROID > SP

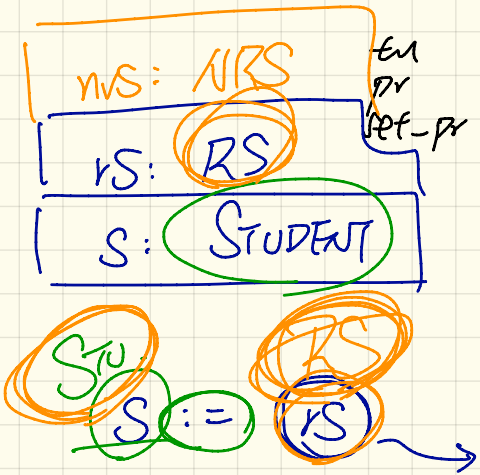


→ s w
→ facetm
→ dial
→ force_touch

descendants for ANDROID

Substitution



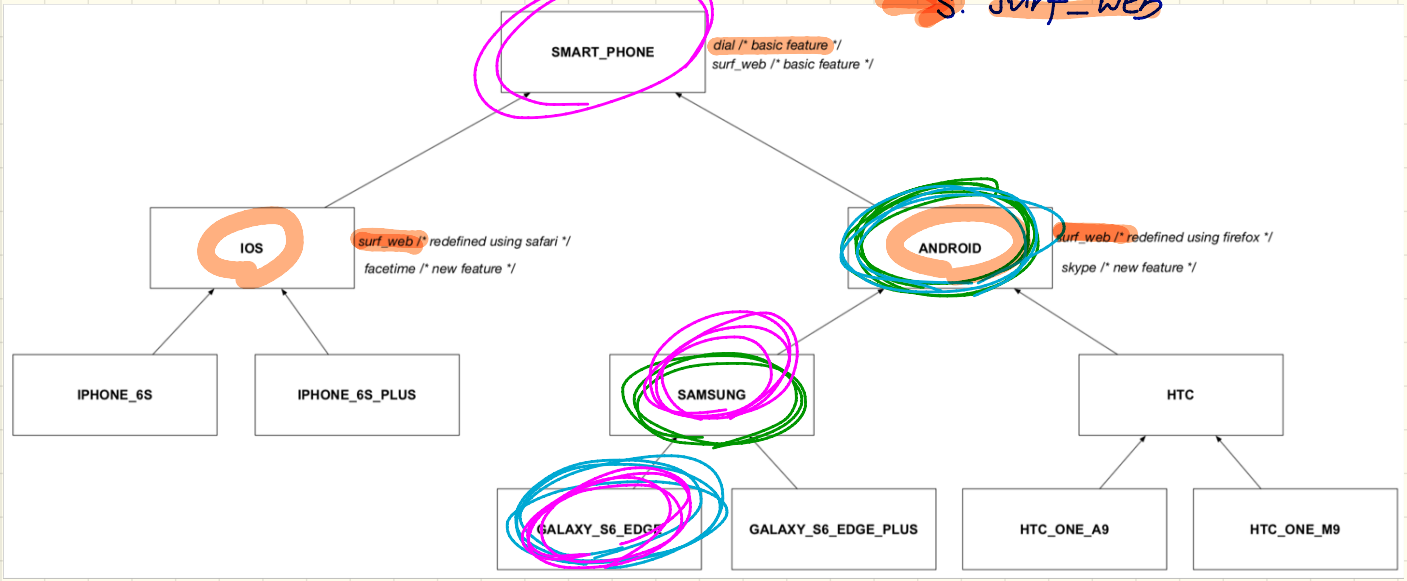


s := NRS

When expecting a STUDENT you can substitute it by any of its descendant classes.

Inheritance Hierarchy : SmartPhones

S : SMART_PHONE
:
S. surf_web

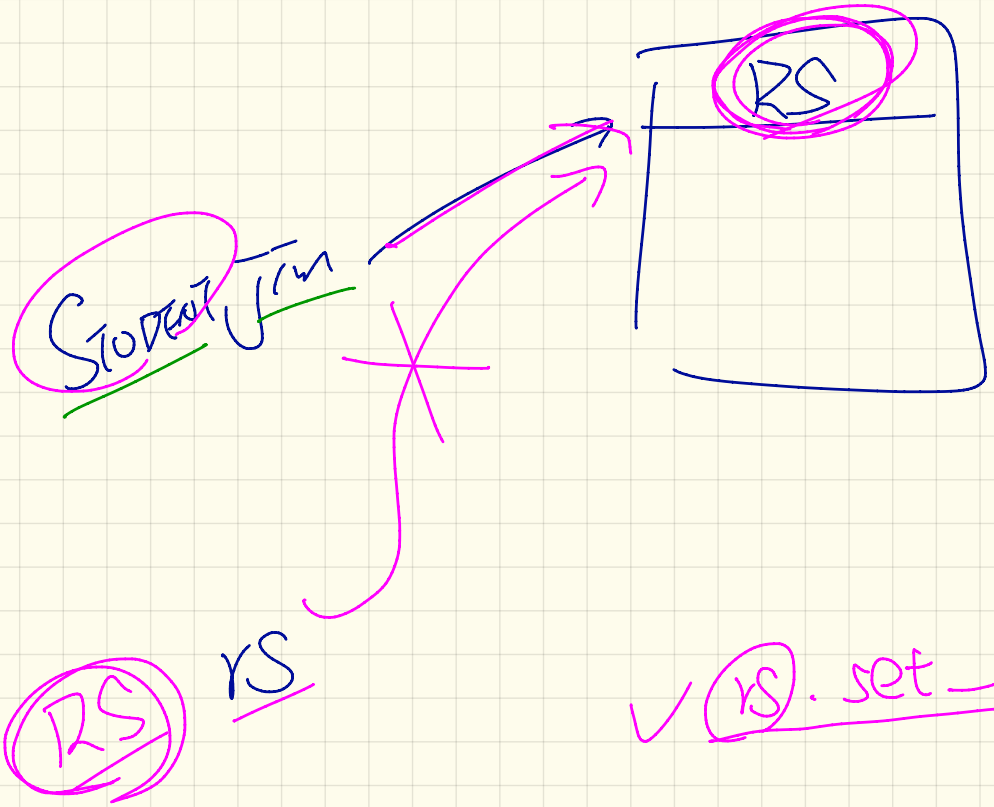


- sp : SAMSUNG
- a : ANDROID
- g : GALAXY_S6_EDGE

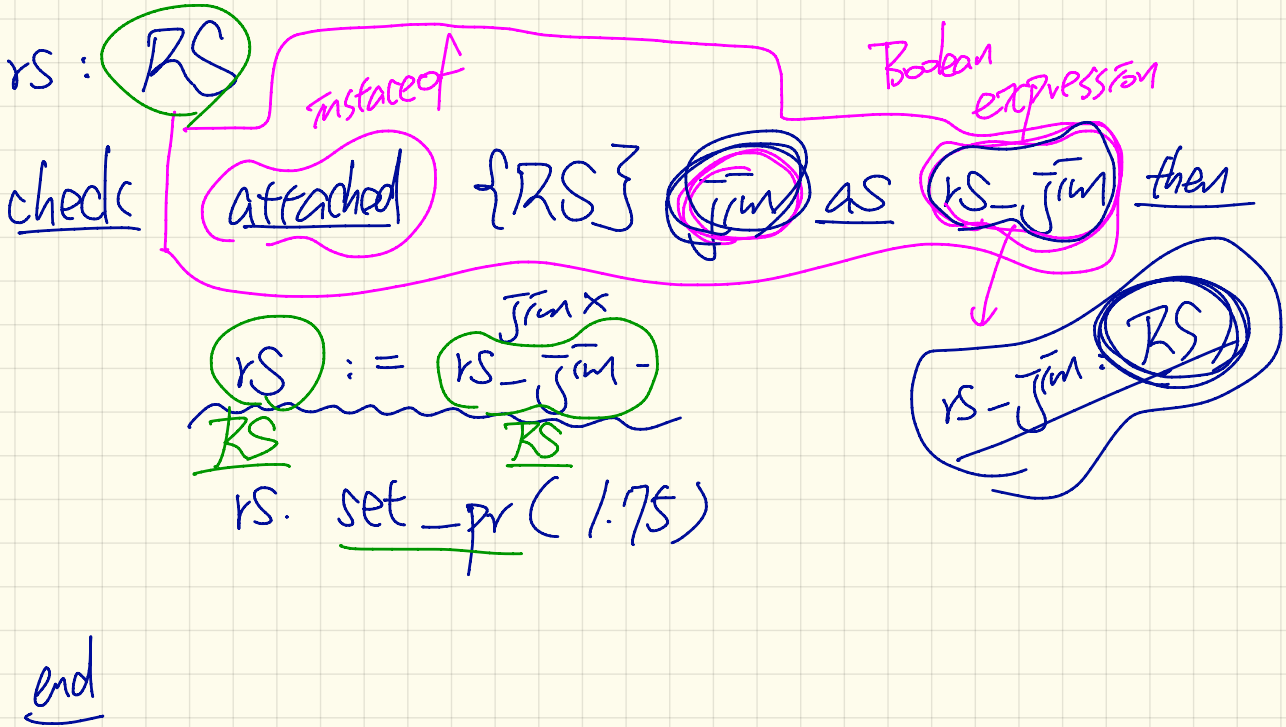
① sp := a ✗

② a := g ✓

③ sp := g ✓



✓ (RS).set_pr(..)


rs : RS
instaceof
Boolean expression
check attached {RS} Jim as rs_Jim then
Jim x
rs := rs_Jim
RS
RS
rs. set_pr (1.75)
rs_Jim : RS
end

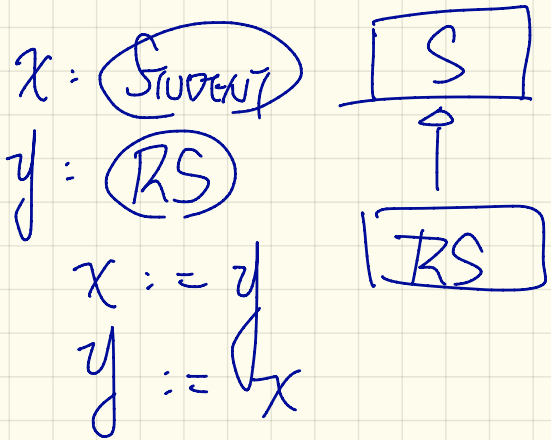
Tuesday Oct. 16

Lecture 10

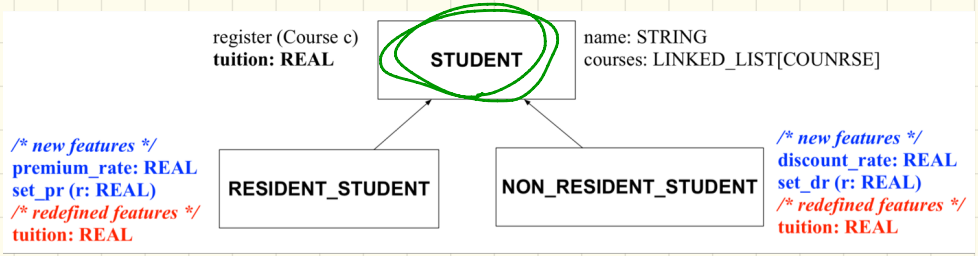
- Lab 1 marks (programming) by Friday
- Midterm
  - Coverage: until Wednesday's lecture
  - Review: Friday 3 pm
- Lab 3
  - Tutorial Videos
  - Tomorrow's Lab Session

Type-Checking Rules

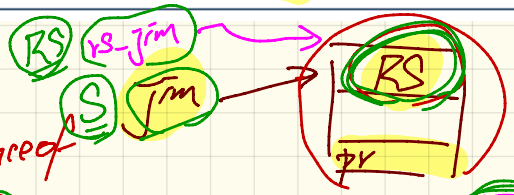
Code	Condition to be Type Correct
`x := y`	`y`'s ***ST*** a **descendant** of `x`'s ***ST***
`x.f(y)`	Feature `f` defined in `x`'s ***ST*** `y`'s ***ST*** a **descendant** of `f`'s parameter's ***ST***
`z := x.f(y)`	Feature `f` defined in `x`'s ***ST*** `y`'s ***ST*** a **descendant** of `f`'s parameter's ***ST*** ***ST*** of `m`'s return value a **descendant** of `z`'s ***ST***
`check attached` `{C} y` **`then`** `...` **`end`**	`C` an **ancestor** or a **descendant** of `y`'s ***ST***
`check attached` `{C} y` **`as`** `temp` **`then`** `x := temp` **`end`**	`C` an **ancestor** or a **descendant** of `y`'s ***ST*** `C` a **descendant** of `x`'s ***ST***
`check attached` `{C} y` **`as`** `temp` **`then`** `x.f(temp)` **`end`**	`C` an **ancestor** or a **descendant** of `y`'s ***ST*** Feature `f` defined in `x`'s ***ST*** `C` a **descendant** of `f`'s parameter's ***ST***



Cast: Motivation

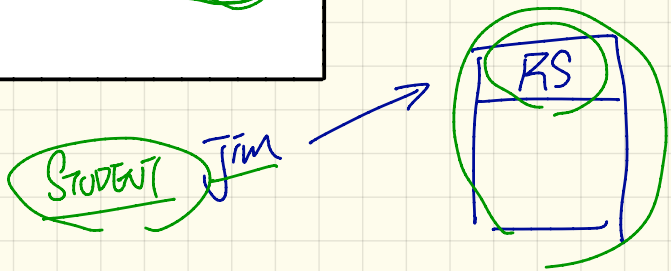



```
local jim: STUDENT  rs: RESIDENT_STUDENT
do create {RESIDENT_STUDENT} jim.make ("J. Davis")
  rs := jim
  rs.setPremiumRate(1.5)
```

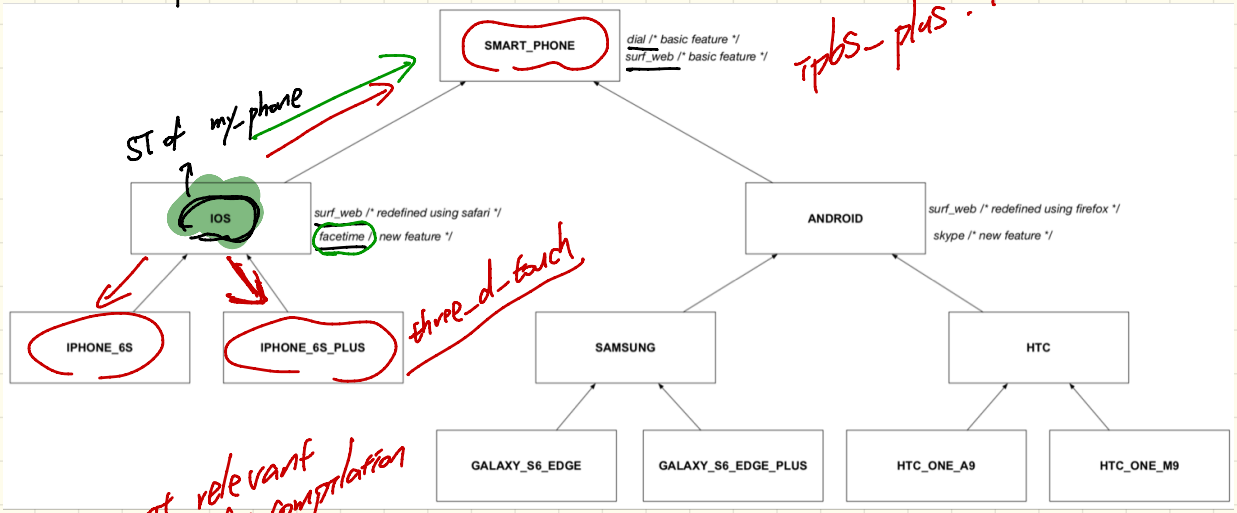


jim.set_pr ✗

```
check attached {RESIDENT_STUDENT} jim as rs_jim then
  rs := rs_jim
  rs.set_pr (1.5)
end
```

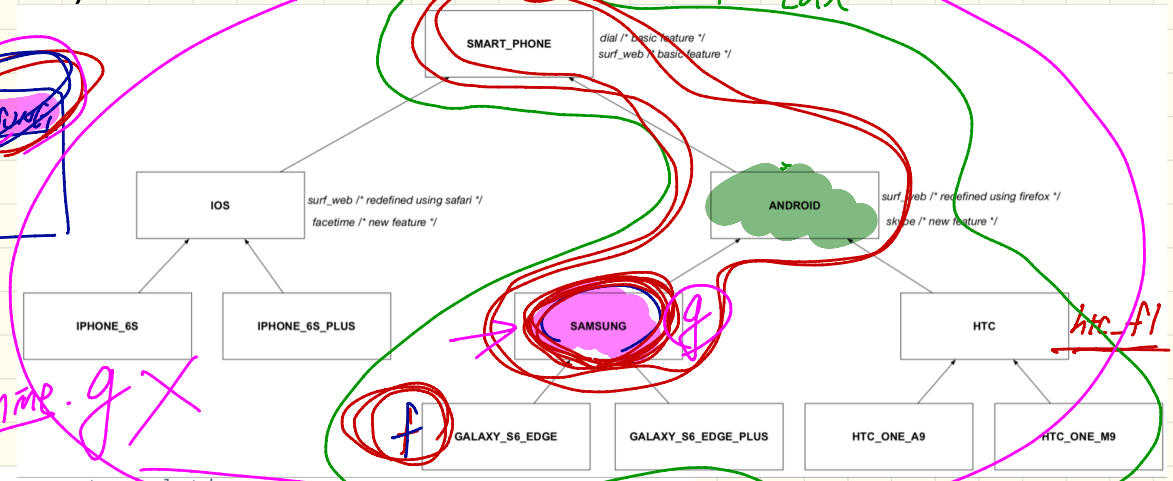


Compilable Cast : Upward or Downward




```
my_phone: IOS
create {IPHONE_6S_PLUS} my_phone.make
-- can only call features defined in IOS on myPhone
-- dial, surf_web, facetime ✓ three_d_touch, skype ×
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ✓  facetime, three_d_touch, skype ×
end
check attached {IPHONE_6S_PLUS} my_phone as ip6s_plus then
  -- can now call features defined in IPHONE_6S_PLUS on ip6s_plus
  -- dial, surf_web, facetime, three_d_touch ✓  skype ×
end
```

Compilable Cast May Fail at Runtime



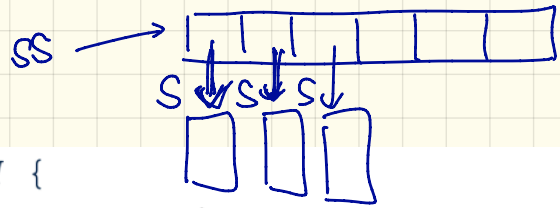

```
test_smart_phone_type_cast_violation
local mine: ANDROID
  do create {SAMSUNG} mine.make
    -- ST of mine is ANDROID; DT of mine is SAMSUNG
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is SAMSUNG
    check attached {SAMSUNG} mine as samsung then ... end
    -- ST of samsung is SAMSUNG; DT of samsung is SAMSUNG
    check attached {HTC} mine as htc then ... end
    -- Compiles ∵ HTC is descendant of mine's ST (ANDROID)
    -- Assertion violation
    -- ∵ HTC is not ancestor of mine's DT (SAMSUNG)
    check attached {GALAXY_S6_EDGE} mine as galaxy then ... end
    -- Compiles ∵ GALAXY_S6_EDGE is descendant of mine's ST (ANDROI
    -- Assertion violation
    -- ∵ GALAXY_S6_EDGE is not ancestor of mine's DT (SAMSUNG)
end
```

Assume this cast could succeed

⇒ galaxy.f would fail

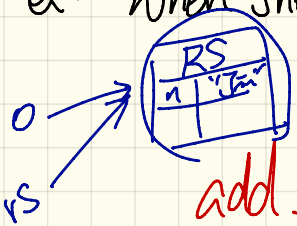
∵ DT is SAMSUNG

Feature Call Arguments : Supplier




```
class STUDENT_MANAGEMENT_SYSTEM {
  ss : ARRAY[STUDENT]  -- ss[i] has static type Student
add_s  (s: STUDENT)  do ss[0] := s end
add_rs (rs: RESIDENT_STUDENT) do ss[0] := rs end
add_nrs (nrs: NON_RESIDENT_STUDENT) do ss[0] := nrs end
```

Q: When should the call sms.add_rs(o) compile?



add_rs (rs: RS) do ... end

rs := o

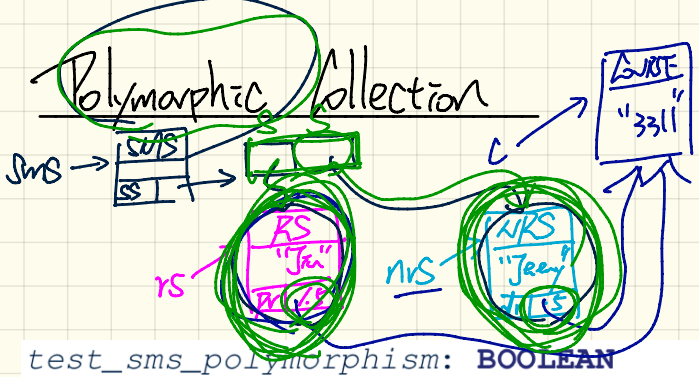
Feature Call Arguments : Client

```
class STUDENT_MANAGEMENT_SYSTEM {
ss : ARRAY[STUDENT] -- ss[i] has static type Student
add_s (s: STUDENT) do ss[0] := s end
add_rs (rs: RESIDENT_STUDENT) do ss[0] := rs end
add_nrs (nrs: NON_RESIDENT_STUDENT) do ss[0] := nrs end
```

```
test_polymorphism_feature_arguments
  local
    s1, s2, s3: STUDENT
    rs: RESIDENT_STUDENT ; nrs: NON_RESIDENT_STUDENT
    sms: STUDENT_MANAGEMENT_SYSTEM
  do
  create sms.make
    create {STUDENT} s1.make ("s1")
    create {RESIDENT_STUDENT} s2.make ("s2")
    create {NON_RESIDENT_STUDENT} s3.make ("s3")
    create {RESIDENT_STUDENT} rs.make ("rs")
    create {NON_RESIDENT_STUDENT} nrs.make ("nrs")
```

sms.add_s(s|)

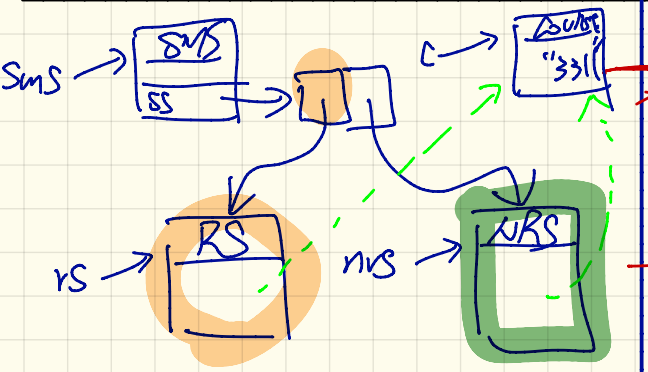
s




```
class STUDENT_MANAGEMENT_SYSETM
  students: LINKED_LIST [STUDENT]
  add_student (s: STUDENT)
    do
      students.extend (s)
    end
  registerAll (c: COURSE)
    do
      across
        students as s
      loop
        s.item.register (c)
      end
    end
end
```

```
test_sms_polymorphism: BOOLEAN
  local
    rs: RESIDENT_STUDENT
    nrs: NON_RESIDENT_STUDENT
    c: COURSE
    sms: STUDENT_MANAGEMENT_SYSTEM
  do
    create rs.make ("Jim")
    rs.set_pr (1.5)
    create nrs.make ("Jeremy")
    nrs.set_dr (0.5)
    create sms.make
    sms.add_s (rs)
    sms.add_s (nrs)
    create c.make ("EECS3311", 500)
    sms.register_all (c)
    Result := sms.ss[1].tuition = 750 and sms.ss[2].tuition = 250
  end
```

Feature Call Return Value



Supplier

```
class STUDENT_MANAGEMENT_SYSTEM {
  ss: LINKED_LIST[STUDENT]
  add_s (s: STUDENT)
    do
      ss.extend (s)
    end
  get_student (i: INTEGER): STUDENT
    require 1 <= i and i <= ss.count
    do
      Result := ss[i]
    end
end
```

Result : STUDENT

Client

```
test_sms_polymorphism: BOOLEAN
local
  rs: RESIDENT_STUDENT ; nrs: NON_RESIDENT_STUDENT
  c: COURSE ; sms: STUDENT_MANAGEMENT_SYSTEM
do
  create rs.make ("Jim") ; rs.set_pr (1.5)
  create nrs.make ("Jeremy") ; nrs.set_dr (0.5)
  create sms.make ; sms.add_s (rs) ; sms.add_s (nrs)
  create c.make ("EECS3311", 500) ; sms.register_all (c)
  Result :=
    get_student(1).tuition = 750
    and get_student(2).tuition = 250
end
```

DT: RS

DT: NRS

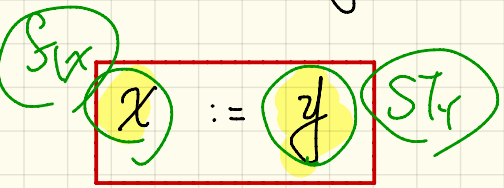
Q: Possible DTs of Result?

Thursday Oct. 18
Lecture 11

- Review Session

Friday 3pm LAS B

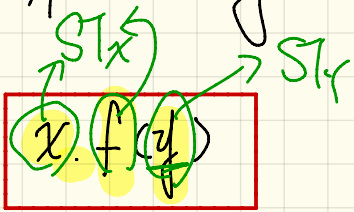
Type Checking Rules (1)



s1 : STUDENT
s2 : RS
s3 : NRS

1 s1 := s2 ✓
2 s1 := s3 ✓
3 s2 := s1 ✗
4 s3 := s1 ✗
5 ✓ s2 := s3 ✗
6 s3 := s1 ✗

Type Checking Rules (2)




```
class SMS
  add_rs(s : RS)
    do
    end ...
end
```

sms : SMS

s1 : STUDENT

s2 : RS

s3 : NRS

STUDENT

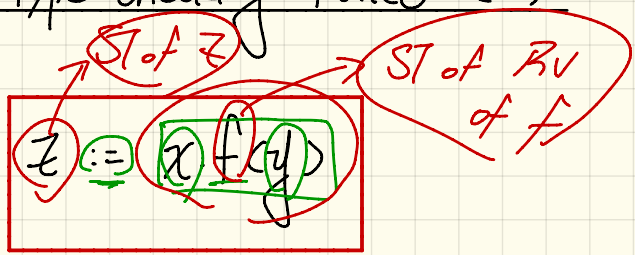
1 ✓ s1.add_rs(s2)

2 ✗ sms.add_rs(s1) → ST : S

3 ✓ sms.add_rs(s2) → ST : RS

4 ✗ sms.add_rs(s3) → NRS

Type Checking Rules (3)




```
class SMS
    get_s(i: INTEGER): STUDENT
    do
        ...
    end
end
```

sms: SMS

s1: STUDENT

s2: RS

s3: NRS

1 s1 := s2.get_s(1) ✗

2 s1 := sms.get_s(1) ✓

3 s2 := sms.get_s(1) ✗

4 s3 := sms.get_s(1) ✗

NRS STUDENT

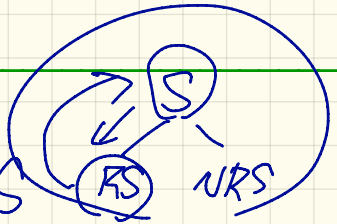
Type Checking Rules (4)

```
check attached {C} y then
    ...
end
```

STy

C is [ancestor of STy
       descendant of STy]

```
class SMS
    get_S (i: INTEGER): STUDENT
    do
        ...
    end
end
```



sms : SMS
s1 : STUDENT
s2 : RS
s3 : URS

1 check attached {RS} s1 then ... end ✓
2 check attached {STUDENT} s2 then ... end ✓
3 check attached {SMS} s1 then ... end ✗
4 check attached {RS} s3 then ... end ✗
5 check attached {RS} sms.get(1) then ... end ✓

STUDENT

Type Checking Rules (5)

```
check attached {C} y as temp then
    x := temp
end
```

ST: C

ST_x

ST: C

```
class SMS
    get_s(i: INTEGER): STUDENT
    do
        ...
    end
end
```

```
check attached {RS} sms.get_s(1) as temp then
    s1 := temp
    s3 := temp
end
```

S

ST: RS

sms: SMS

s1: STUDENT

s2: RS

s3: NRS

s1 := temp ✓ RS

2 s3 := temp ✗

NRS ✗ RS

Type Checking Rules (6)

```
check attached {C} y as temp then
    x.f(temp)
end
```

ST: C

```
class SMS
    get_s(i: INTEGER): STUDENT
    do
        ...
    end
    add_rs(s: RS)
    do
        ...
    end
```

```
RS temp = (RS) sms.get_s(1);
sms.add_rs(temp);
```

sms: SMS

s1: STUDENT

s2: RS

s3: NRS

① compiles

```
check attached {RS} sms.get_s(1) as temp then
    sms.add_rs(temp)
end
```

RT. S ST: RS

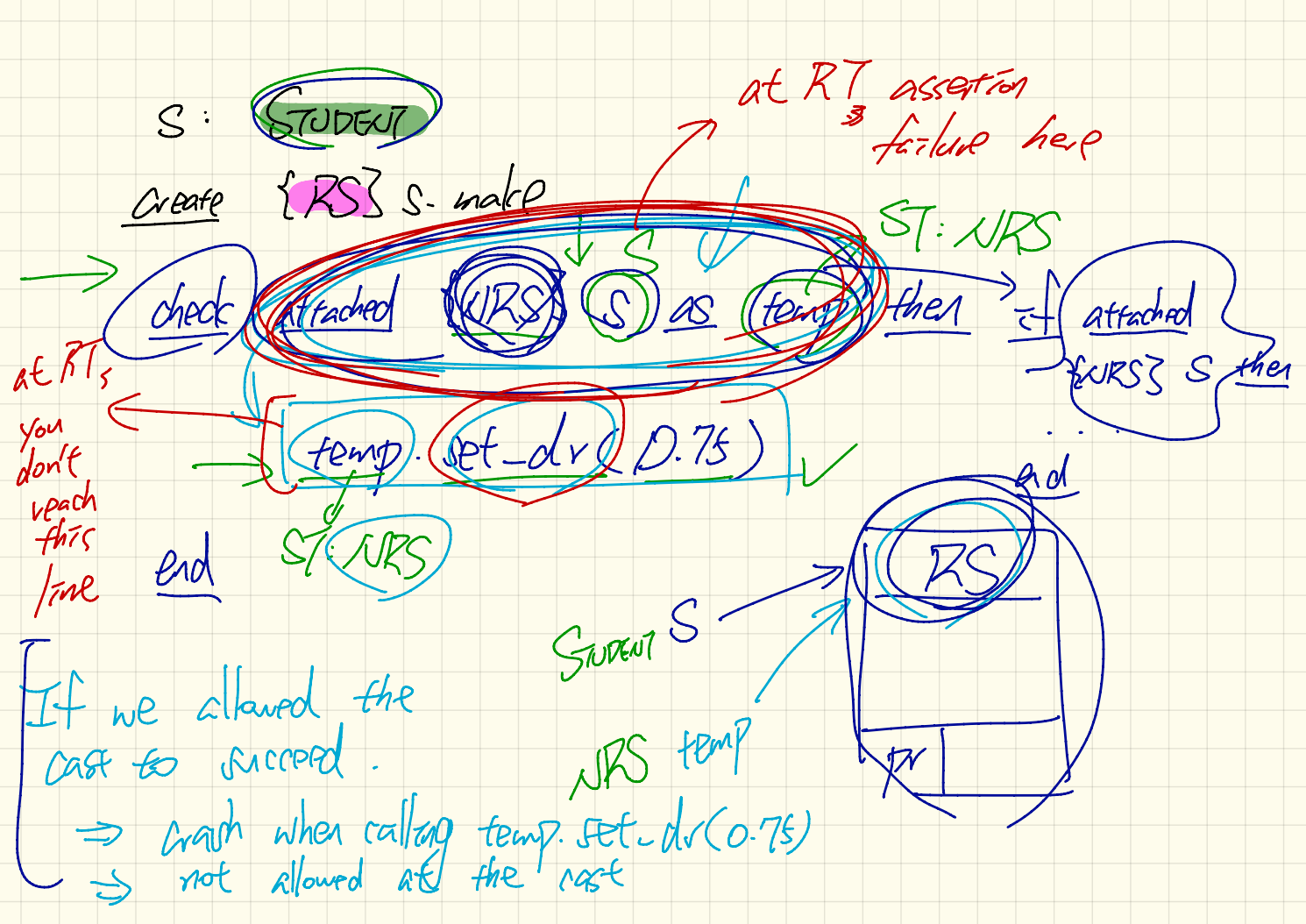
DT: NRS

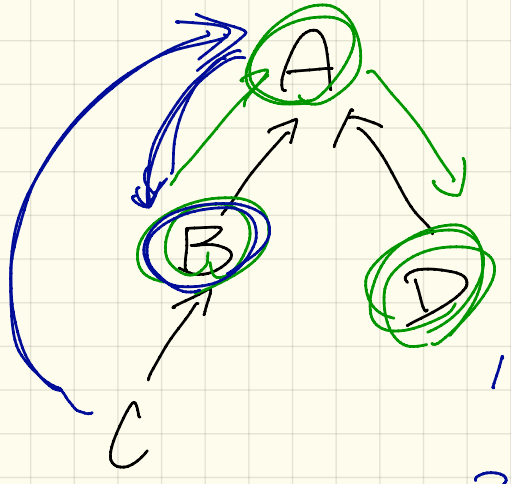
temp.set_pr

② ST: NRS

```
check attached {NRS} sms.get_s(1) as temp then
    sms.add_rs(temp)
end
```

NRS ✗


S : STUDENT
create {RS} s.make
at RT assertion failure here
ST: NRS
check attached {NRS} S as temp then
attached {NRS} S then
temp.set_dr(0.75)
at RT, you don't reach this line
end
ST: NRS
end
RS
dr
STUDENT S
NRS temp
If we allowed the cast to succeed.
⇒ crash when calling temp.set_dr(0.75)
⇒ not allowed at the cast



b: B

d: D

c: C

1 create {C} b.make

2 check attached {D} b as ST: B temp then

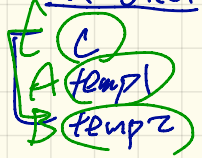
3 d := temp

end

check attached {A} c as temp1 A then B

check attached {B} temp1 as temp2 then

end

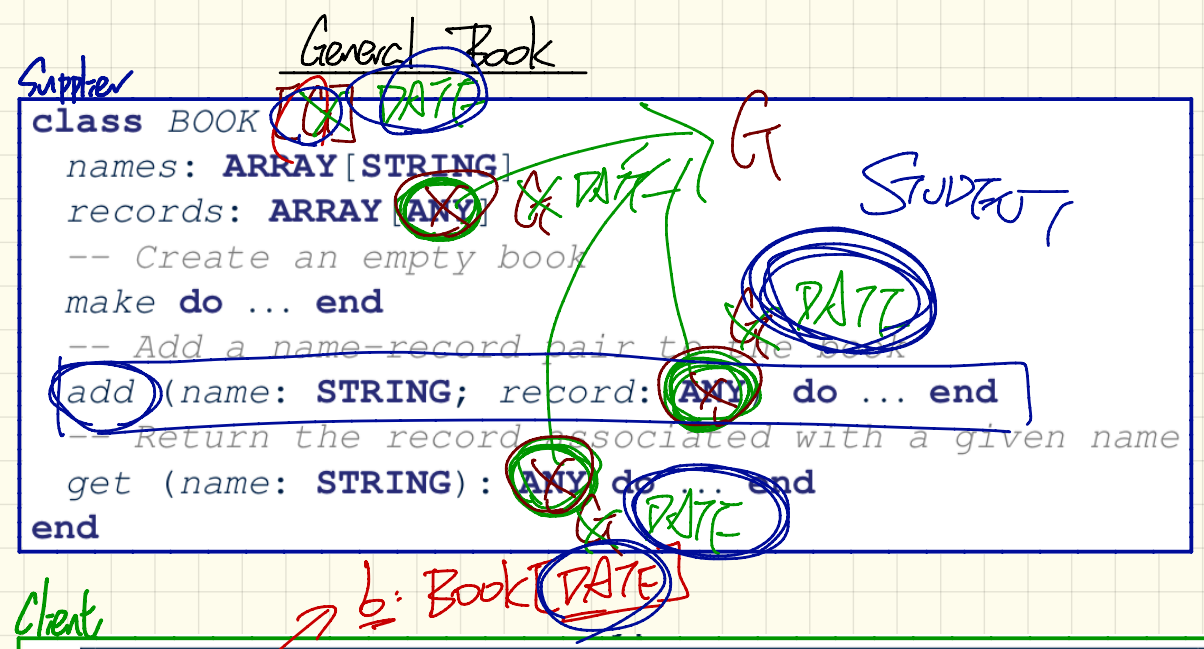


check attached {A} b as temp 1 and then

attached {B} temp1 as temp2 then

:

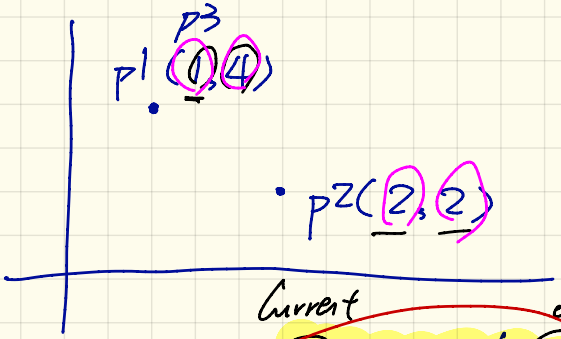
end




```
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_of_week = 4
```

Friday Oct. 19

Midterm Review



class POINT

inherit ANY
redefine is_equal end

feature

feature x, y : INTEGER

feature
is_equal (other : like Current) : BOOLEAN

ensure

p1: x = other.x and y = other.y

end

p2 : Current ~ old Current.deep_twin

Current: p1. is_equal (p3) → POINT

Current: p1. is_equal (p2)

Current		other
x 1	F	x 2
y 4	F	y 2

Is the postcondition appropriate?

obligation of supplier

Fix?

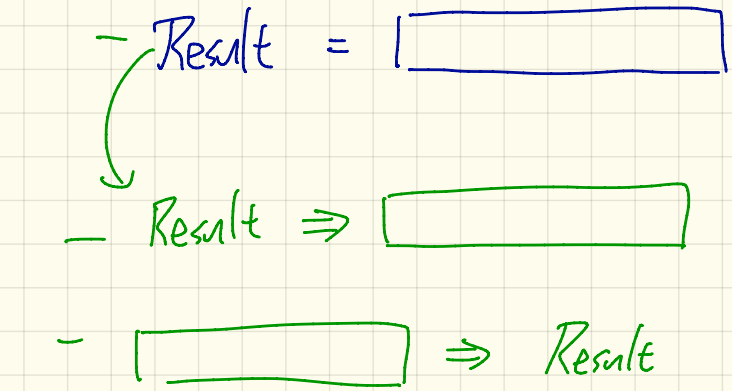
P.C. violation

Result = (x = other.x and y = other.y)

F F T

1 2 4 2

ensure



$$(p \equiv q) = (p \Rightarrow q \wedge q \Rightarrow p)$$

do

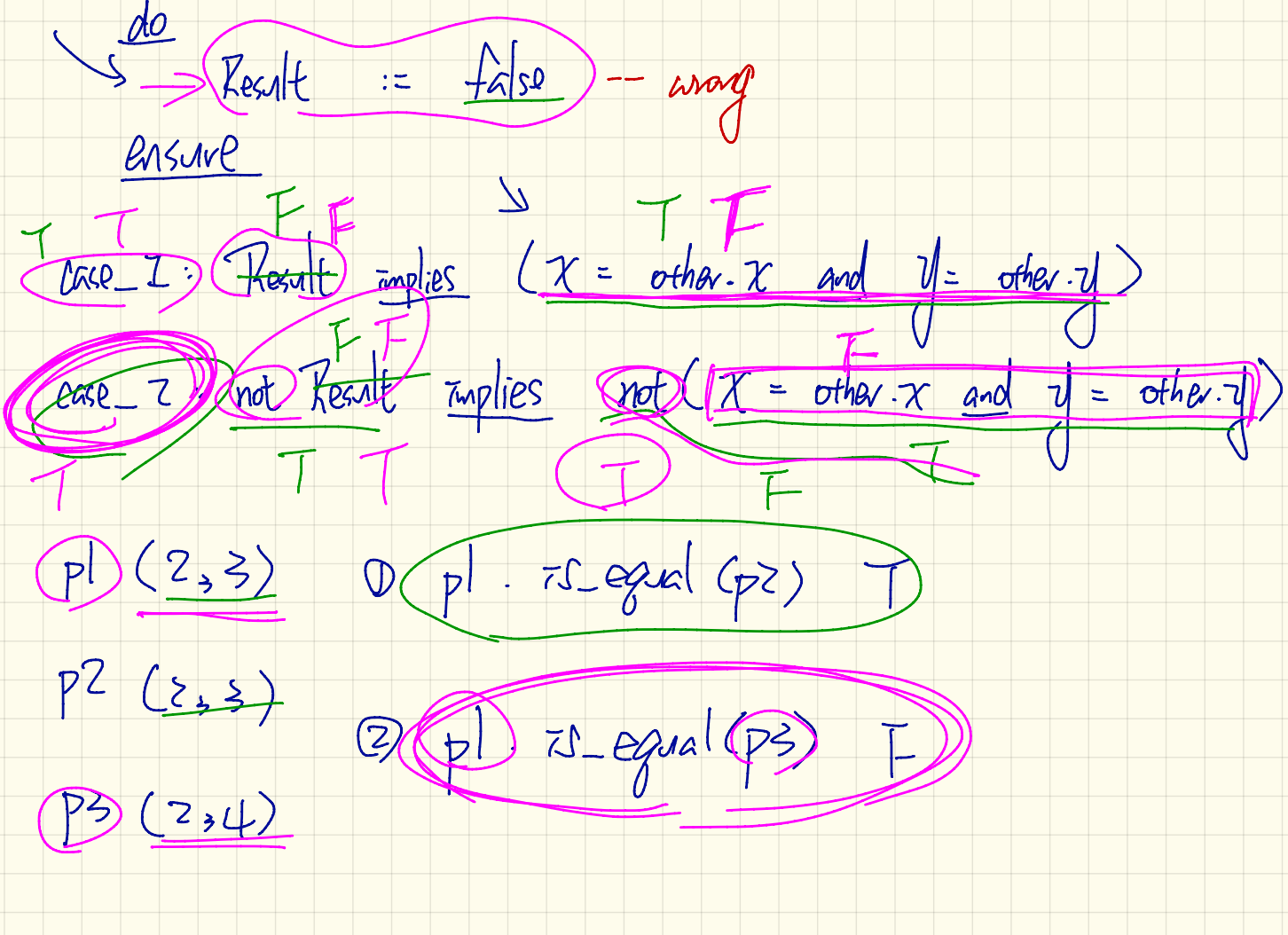
Result := false

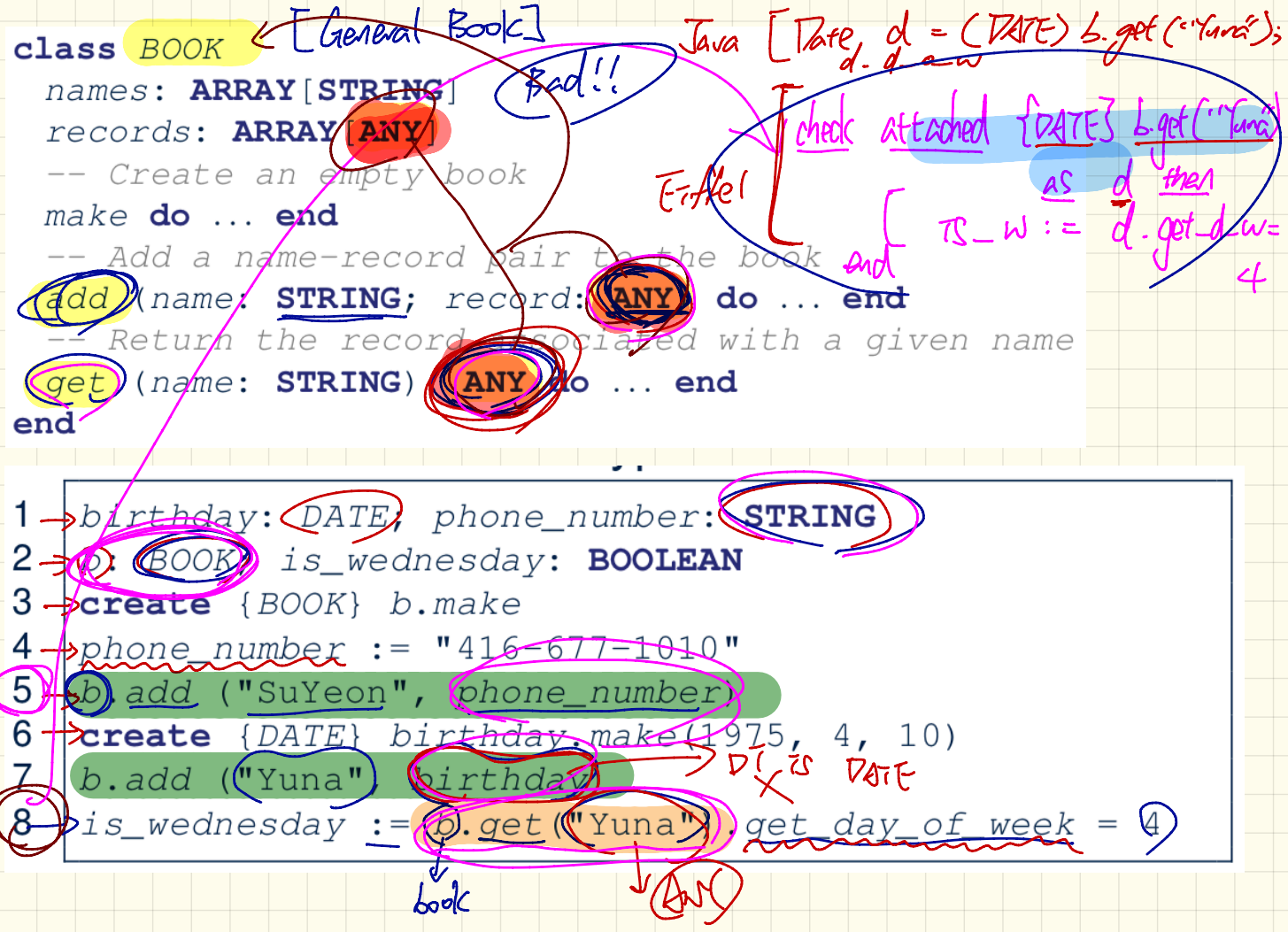
ensure

p: Result implies (x = other.x and y = other.y)

Hint.. F ⇒ T = T

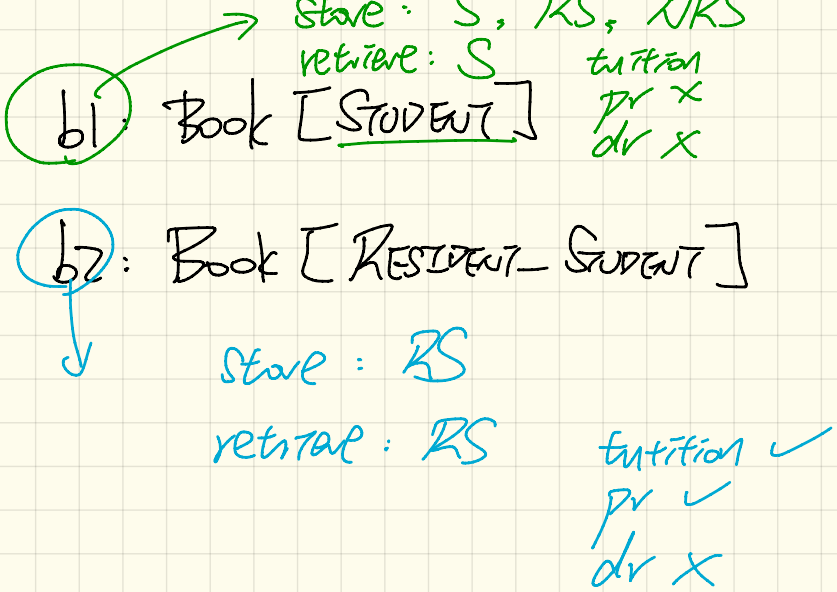
produced by supplier


do
Result := false -- wrong
ensure
case_1: Result implies (x = other.x and y = other.y)
case_2: not Result implies not (x = other.x and y = other.y)
p1 (2, 3)
p2 (2, 3)
p3 (2, 4)
① p1.is_equal (p2) T
② p1.is_equal (p3) F


class BOOK
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
[General Book]
Bad!!
Java [Date d = (DATE) b.get("Yuna");
d.d_o_w
Eiffel
check attached {DATE} b.get("Yuna") as d then
ts_w := d.get_d_o_w = 4
end
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_of_week = 4
D1 is DATE
book
ANY

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

```
birthday: DATE; phone_number: STRING
b: BOOK[DATE]; is_wednesday: BOOLEAN
create BOOK[DATE] b.make
phone_number := "416-67-1010"
b.add ("SuYeon", phone_number)
create {DATE} birthday.make (1975, 4, 10)
b.add ("Yuna", birthday)
is_wednesday := b.get("Yuna").get_day_of_week == 4
```

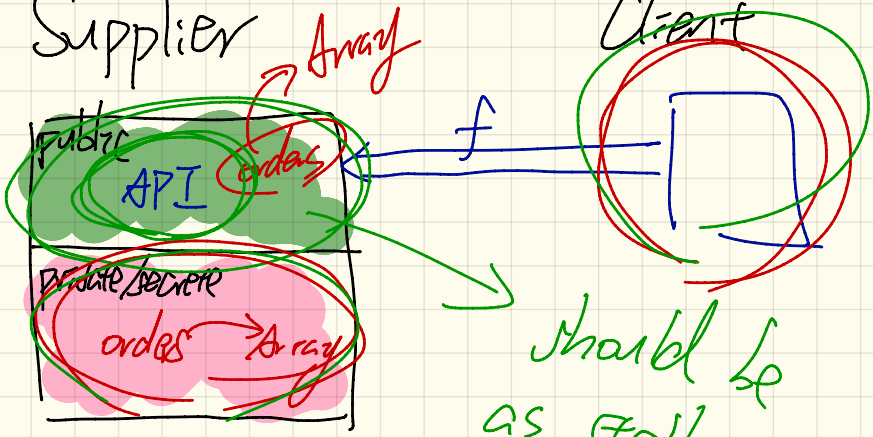

store : S, RS, NRS
retrieve : S
tuition
pr ×
dr ×
b1 : Book [STUDENT]
b2 : Book [RESIDENT_STUDENT]
store : RS
retrieve : RS
tuition ✓
pr ✓
dr ×

Information Hiding

e.g. choice of data. str.

Design decisions that are subject to constant changes should be hidden from the clients.

Supplier



Eiffel

deferred *

effective +

class Point

end

Java

abstract

non-abstract / Concrete

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

STUDENT

STUDENT

b1: BOOK[STUDENT]

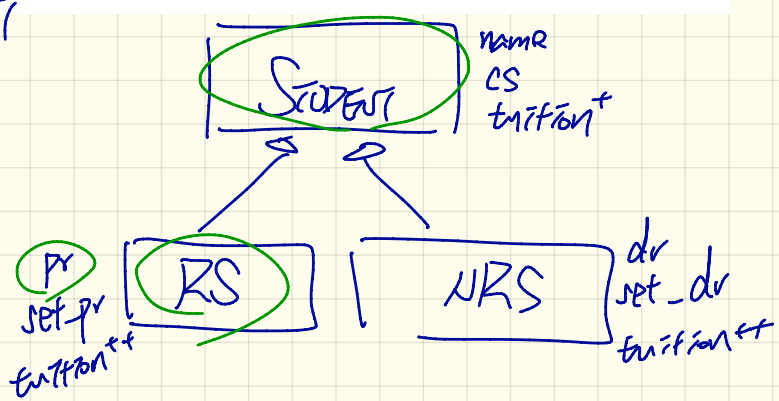
⋮

- b1.get("Stella").tuition

STUDENT

b1.get("Rachael").set_pr ✗

STUDENT



Say a client wants to store objects of types $C_1, C_2, \ldots, C_n$,

KS URS

IP_bS IP_bSP

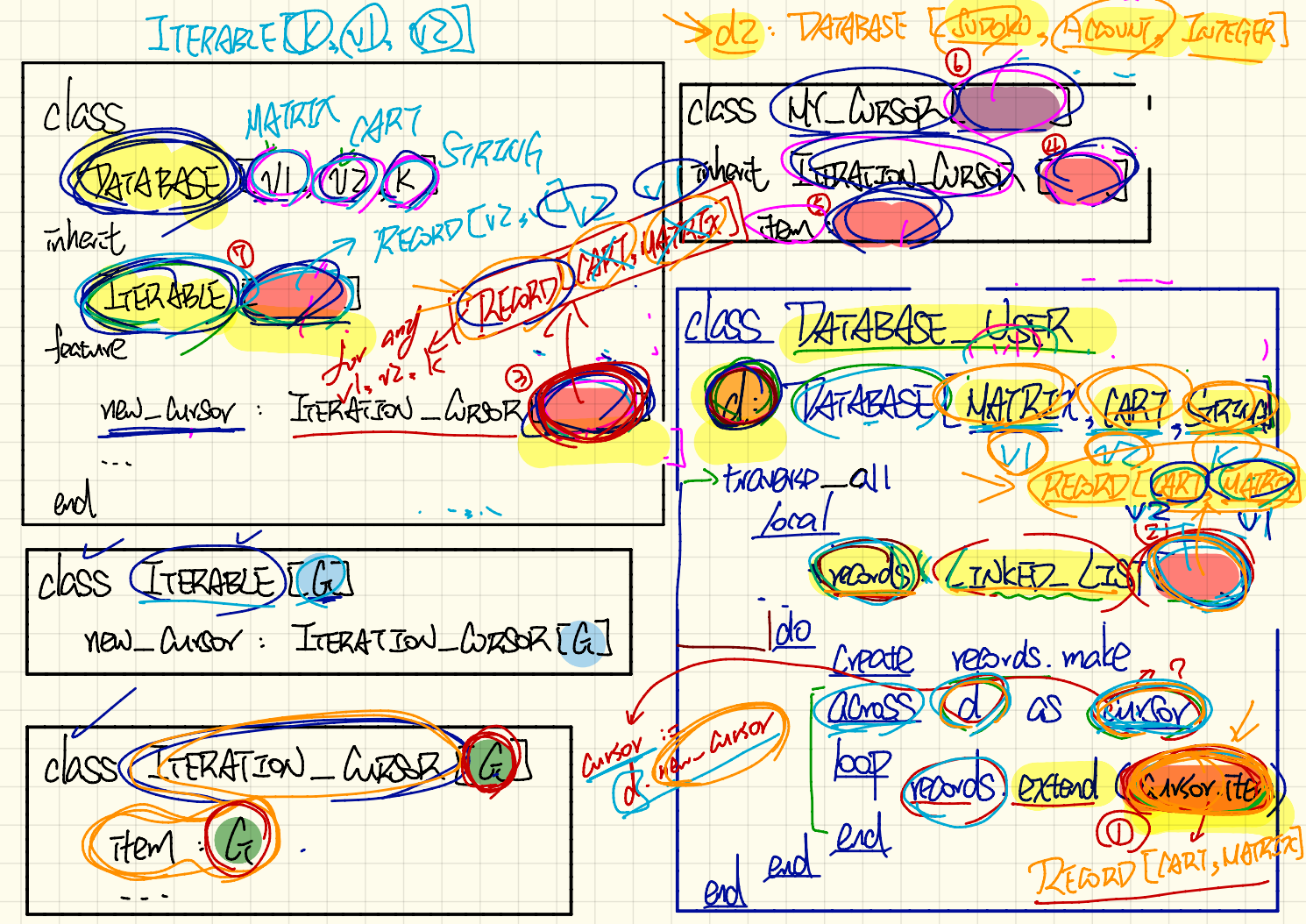
how should they declare using the generic book?

```
b : Book[ ? ]
```

STUDENT

IoS

choose the "lowest" common ancestor of $C_1, C_2 \ldots, C_n$


ITERABLE [K, V1, V2]
class
DATABASE
MATRIX
CART
STRING
V1
V2
K
inherit
ITERABLE
RECORD [V2, K]
RECORD [CART, MATRIX]
for any V1, V2, K
feature
new_cursor : ITERATION_CURSOR
...
end
d2 : DATABASE [SUDOKU, ACCOUNT, INTEGER]
class MY_CURSOR
inherit ITERATION_CURSOR
item :
class DATABASE_USER
d : DATABASE [MATRIX, CART, STRING]
RECORD [CART, MATRIX]
traverse_all
local
records
LINKED_LIST
do
create records.make
across d as cursor
loop
records.extend (cursor.item)
RECORD [CART, MATRIX]
end
end
end
class ITERABLE [G]
new_cursor : ITERATION_CURSOR [G]
class ITERATION_CURSOR [G]
item : G
...
cursor := d.new_cursor

Invariant

keys_unique:

such that

the case

$\forall i : 1 \le i \le keys.count \bullet$

$\forall j : 1 \le j \le keys.count \bullet$

$i \neq j \Rightarrow keys[i] \neq keys[j]$

$keys[i] \sim keys[j] \Rightarrow i = j$

$\forall i, j : 1 \le i \le keys.count \wedge 1 \le j \le keys.count \bullet$

$i \neq j \Rightarrow keys[i] \neq keys[j]$

keys

across |1..|keys.count as i

all

across |1..|keys.count as j

all

i.item /= j.item implies keys[i.item] /~ keys[j.item]

end

end

across keys as k1

all

across keys as k2

all

get_index_of_key(k1)

k1.cursor_index /= k2.cursor_index

implies

k1 /~ k2

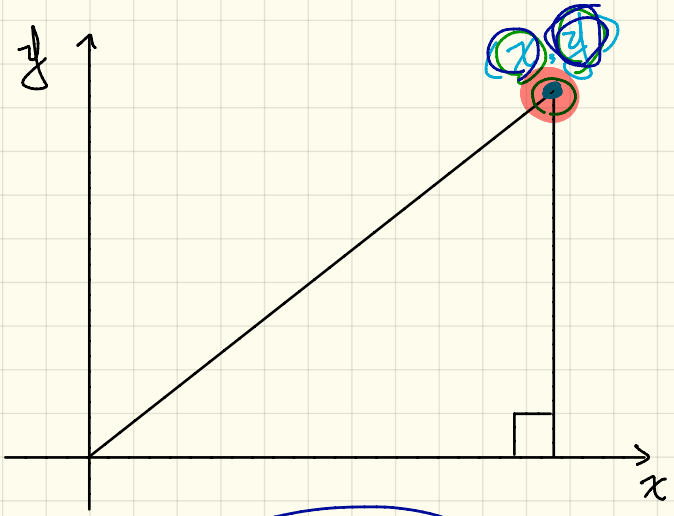
end

end

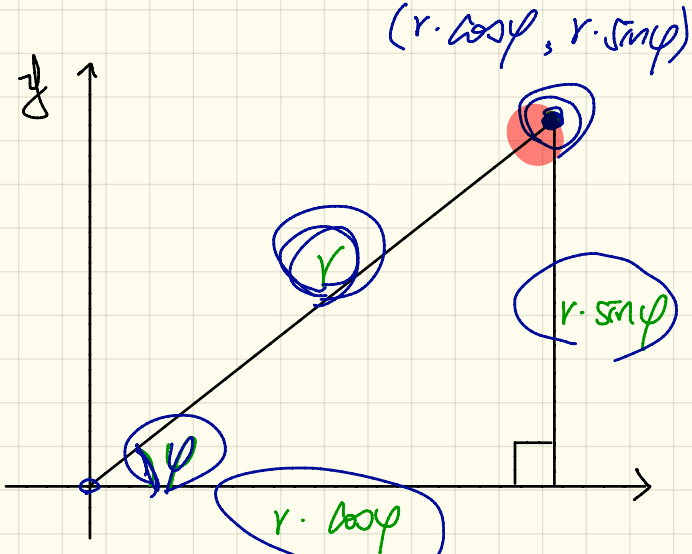
Tuesday Oct. 22

Lecture 12

Uniform Access to a 2D Point



Cartesian



Polar

Two Possible Ways to Implementing POINT

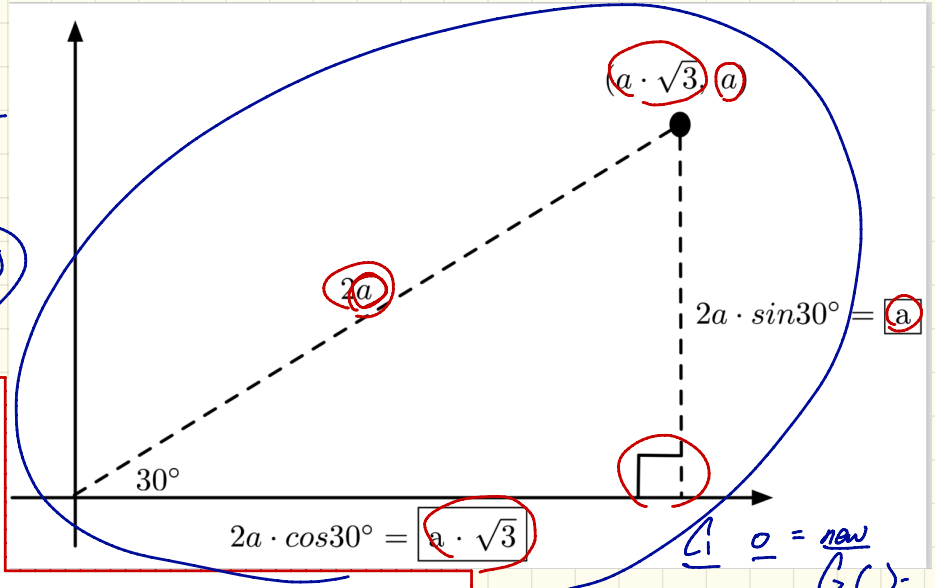
```
class POINT -- Version 1
feature -- Attributes
  x : REAL
  y : REAL
feature -- Constructors
  make_cartisian(nx: REAL; ny: REAL)
    do
      x := nx
      y := ny
    end
end
```

```
class POINT -- Version 2
feature {NONE} -- Attributes
  r : REAL
  p : REAL
feature -- Constructors
  make_polar(nr: REAL; np: REAL)
    do
      r := nr
      p := np
    end
feature -- Queries
  x : REAL do Result := r × cos(p) end
  x : REAL do Result := r × sin(p) end
end
```

Testing Uniform Access

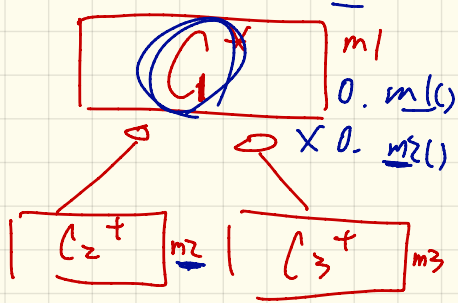
Point p1 = new Point()?

	Eiffel	Java
Attribute	o.f	o.a
Query	o.f	o.am()




```
test_points: BOOLEAN
  local
    A, X, Y: REAL
    p1, p2: POINT
  do
    comment("test: two systems of points")
    A := 5; X := A × √3; Y := A
    create {POINT} p1.make_cartisian (X, Y)
    create {POINT} p2.make_polar (2 × A, 1/6 π)
    Result := p1.x = p2.x and p1.y = p2.y
  end
```

C1 o = new C2();



Can Overloading support Uniform Access

YES

```
void ml (int i) {...} ←
void ml (String s) {...}
```

```
o. ml(2)←
o. ml ("alan")
```

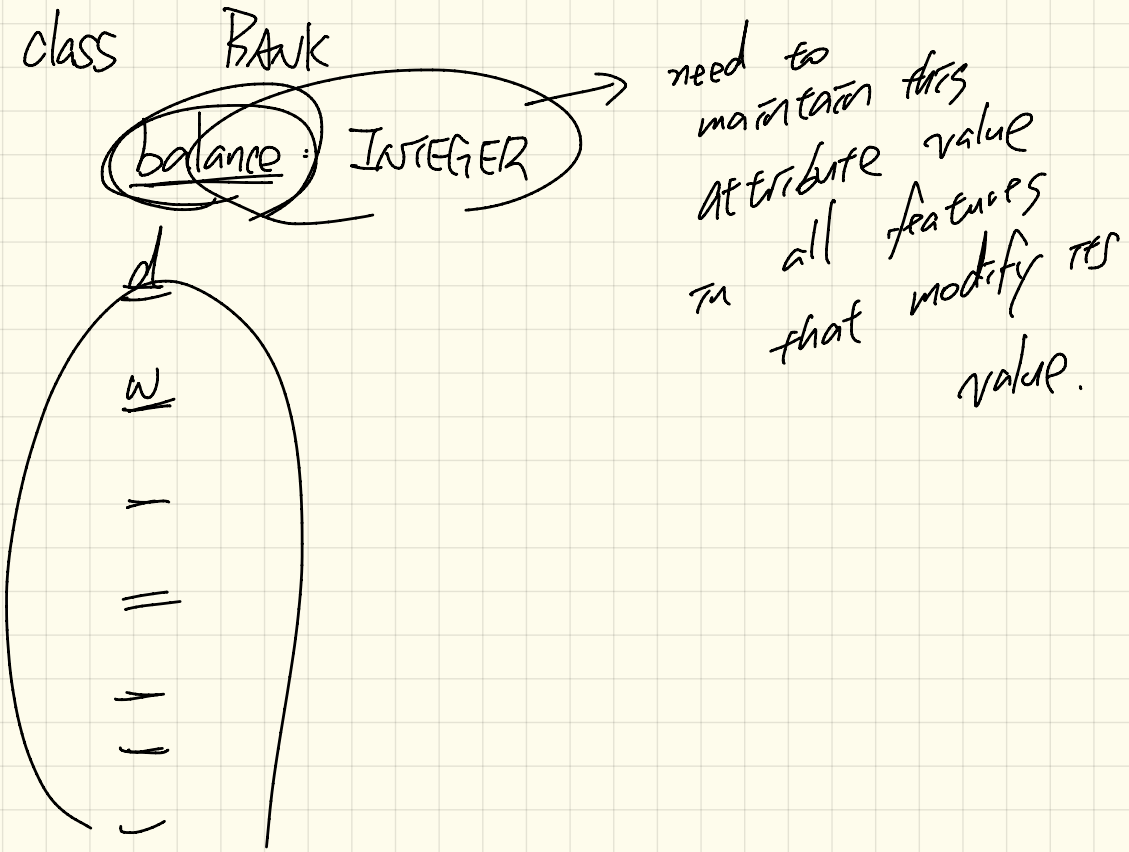
o. ml (23)

~~void ml (int i)..~~
~~void ml (int j)..~~

No

```
void m2 (int i)
void m2 (String s, int i)
```

o. m2 (2)
o. m2 ("a", 2)

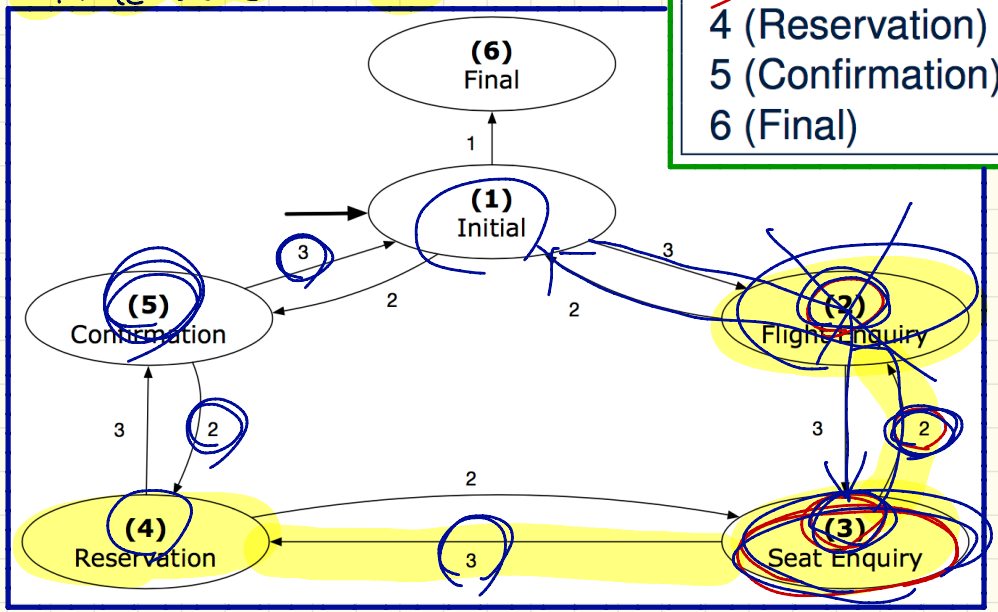

class BANK
balance : INTEGER
need to maintain this attribute value in all features that modify its value.
d
w

State Transition Diagram (FSM)

Transition Table

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)	–	–	–

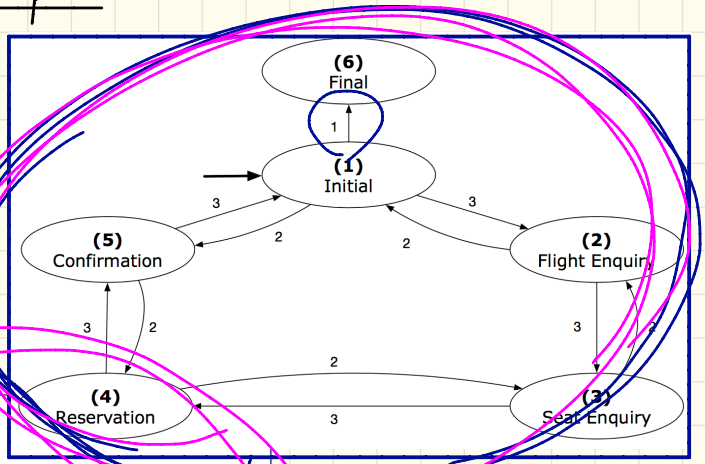
Finite State Machine



Design of a Reservation System: First Attempt

- code duplicates between labels

→ solution is not reusable for another problem




```
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
```

Design of a Reservation System: Second Attempt (1)

```
transition (src: INTEGER; choice: INTEGER): INTEGER
  -- Return state by taking transition 'choice' from 'src' state.
require valid_source_state: 1 ≤ src ≤ 6
        valid_choice: 1 ≤ choice ≤ 3
ensure valid_target_state: 1 ≤ Result ≤ 6
```

e.g. transition (3, 2) → 4
transition (3, 3) → 5

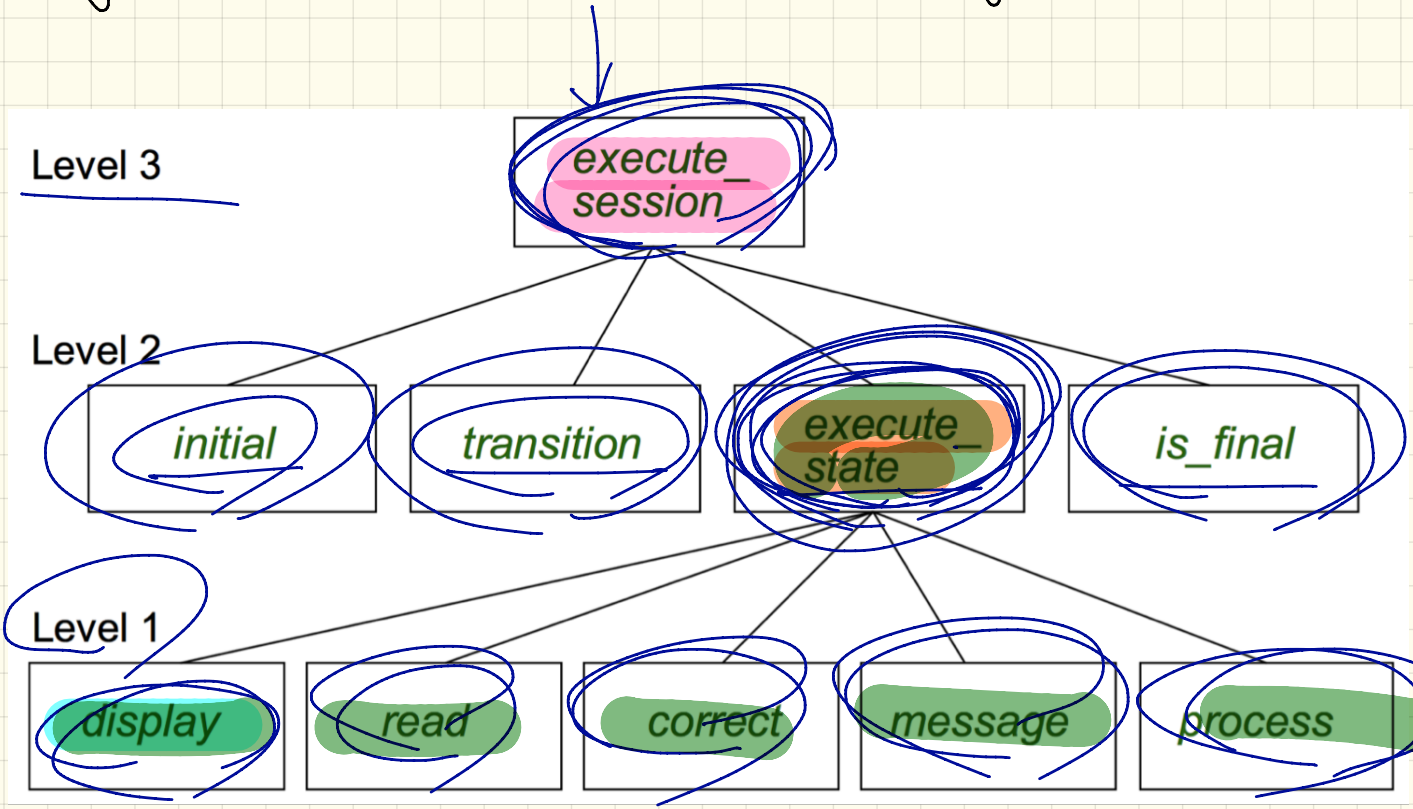
Transition Table

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)	–	–	–

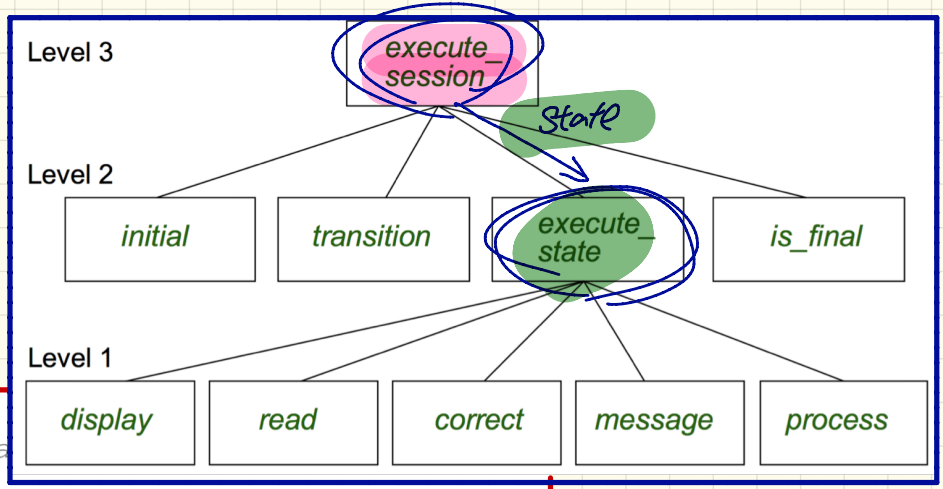
2D-Array Implementation

state \ choice	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

Design of a Reservation System: a Top-Down Design



Design of a Reservation System: Second Attempt (2)



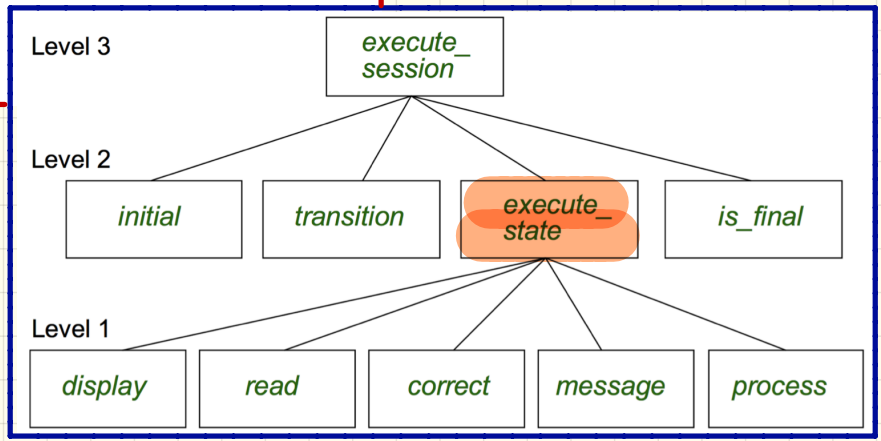

```
execute_session
    -- Execute a full intera
  local
    current_state, choice: INTEGER
  do
    from
      current_state := initial
    until
      is_final (current_state)
    do
      choice := execute_state (current_state)
      current_state := transition (current_state, choice)
    end
  end
```

Design of a Reservation System: Second Attempt (2)

```
execute_state (current_state: INTEGER): INTEGER
    -- Handle interaction at the current state.
    -- Return user's exit choice.
  local
    answer: ANSWER; valid_answer: BOOLEAN; choice: INTEGER
  do
    from
    until
      valid_answer
    do
      display (current_state)
      answer := read_answer (current_state)
      choice := read_choice (current_state)
      valid_answer := correct (current_state, answer)
      if not valid_answer then message (current_state, answer)
      end
    process (current_state, answer)
    Result := choice
  end
```

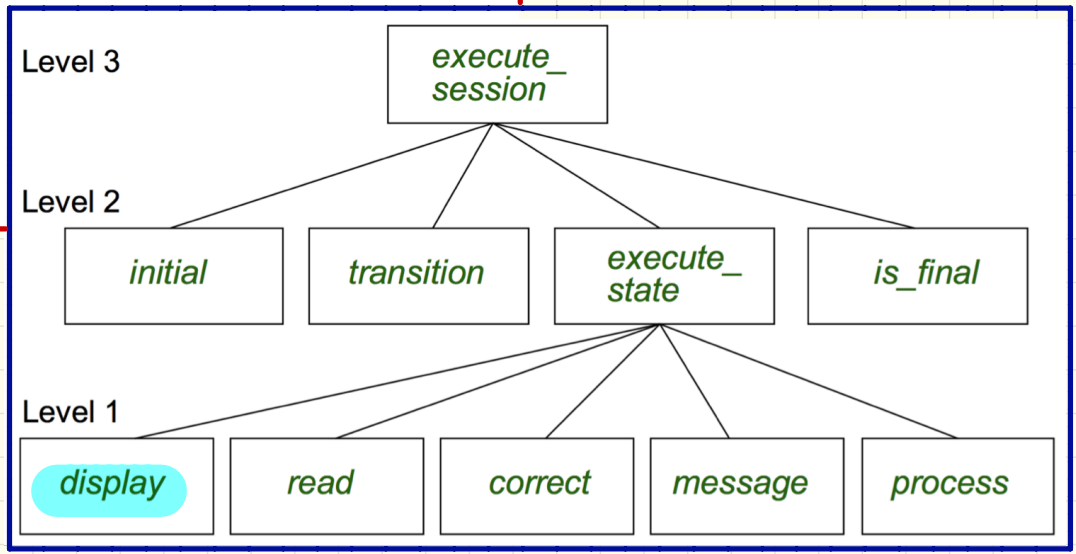
display (current_state)

CS. display



Design of a Reservation System: Second Attempt (3)

```
display(current_state: INTEGER)
  require
    valid_state: 1 ≤ current_state ≤ 6
  do
    if current_state = 1 then
      -- Display Initial Panel
    elseif current_state = 2 then
      -- Display Flight Enquiry Panel
    ...
    else
      -- Display
    end
  end
```



Thursday Oct. 25

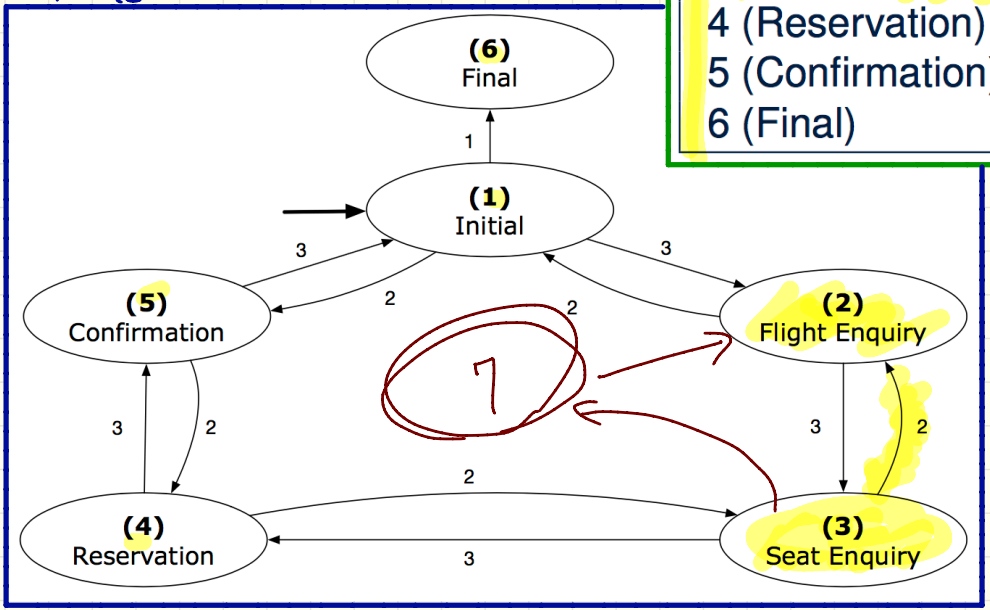
Lecture 13

State Transition Diagram (FSM)

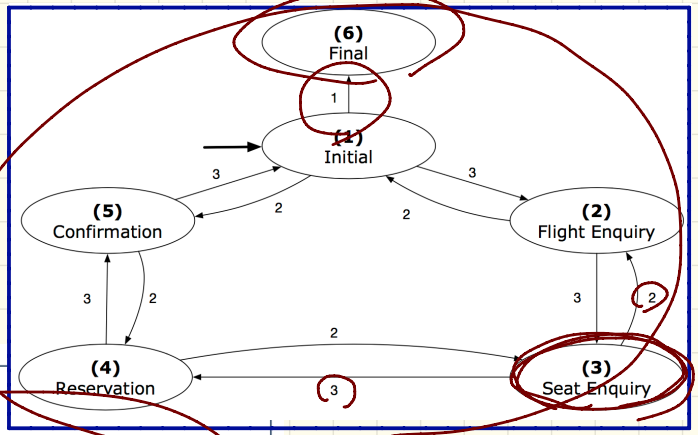
Transition Table

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)	–	–	–

Finite State Machine



Design of a Reservation System: First Attempt




```
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.
```

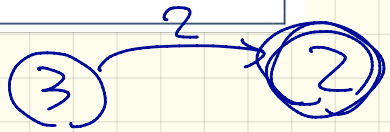
```
3_Seat_Enquiry_panel:
from
  Display Seat Enquiry Panel
until
  not (wrong answer or wrong choice)
do
  Read user's answer for current panel
  Read user's choice C for next step
  if wrong answer or wrong choice then
    Output error messages
  end
end
Process user's answer
case C in
  2: goto 2_Flight_Enquiry_panel
  3: goto 4_Reservation_panel
end
```

Design of a Reservation System: Second Attempt (1)

```
transition (src: INTEGER; choice: INTEGER): INTEGER
  -- Return state by taking transition 'choice' from 'src' state.
require valid_source_state: 1 ≤ src ≤ 6
        valid_choice: 1 ≤ choice ≤ 3
ensure valid_target_state: 1 ≤ Result ≤ 6
```

e.g. transition (3, 2) → 2

transition (3, 3)



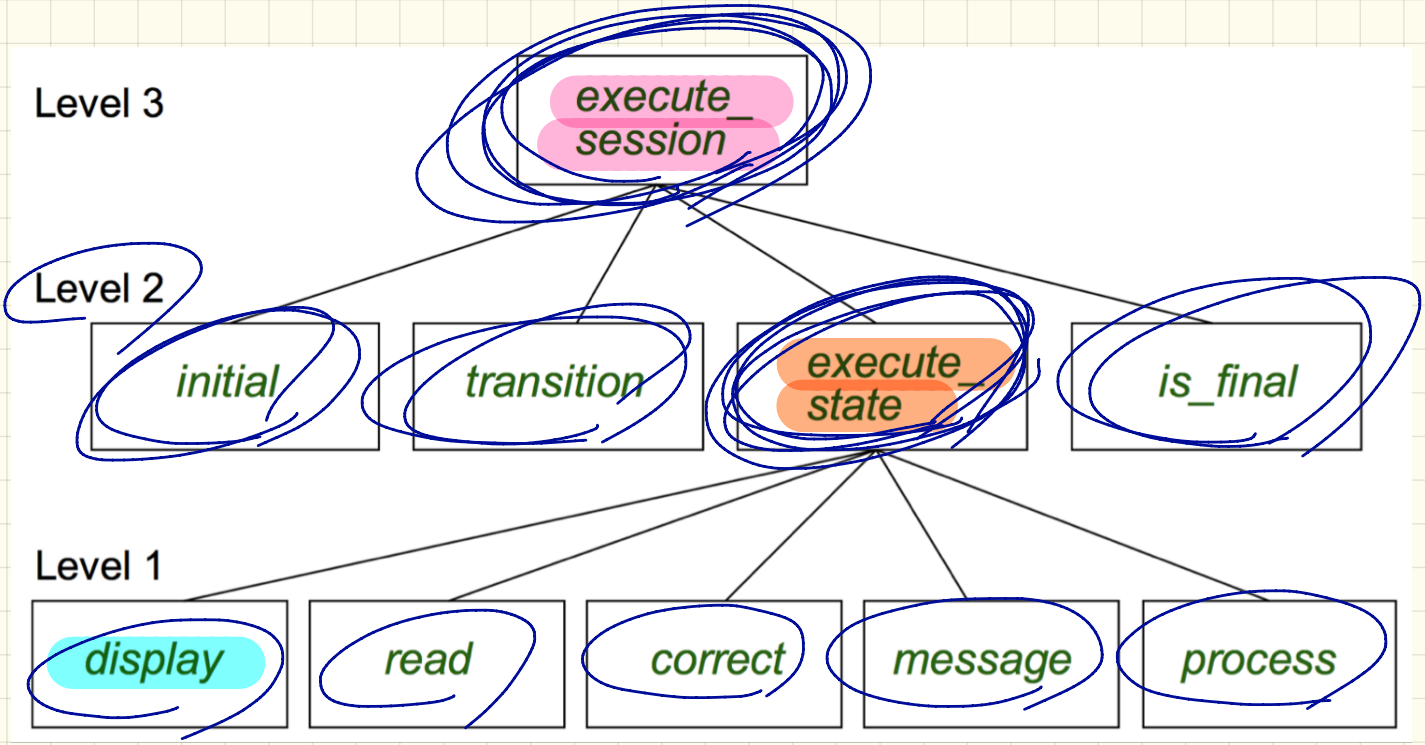
Transition Table

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)	–	–	–

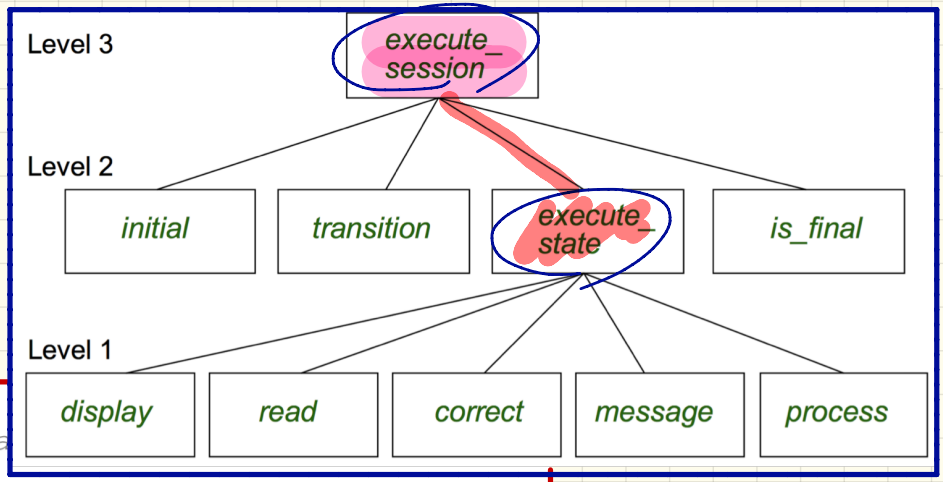
2D-Array Implementation

state \ choice	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

Design of a Reservation System: a Top-Down Design



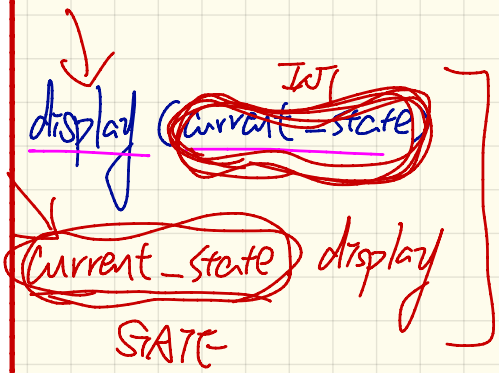
Design of a Reservation System: Second Attempt (2)

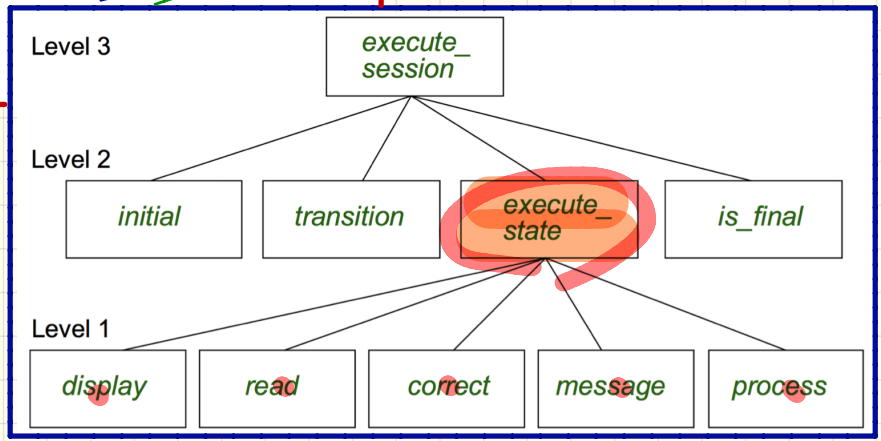



```
execute_session
    -- Execute a full intera
  local
    current_state, choice: INTEGER
  do
    from
      current_state := initial
    until
      is_final (current_state)
    do
      choice := execute_state (current_state)
      current_state := transition (current_state, choice)
    end
  end
```

Design of a Reservation System: Second Attempt (2)

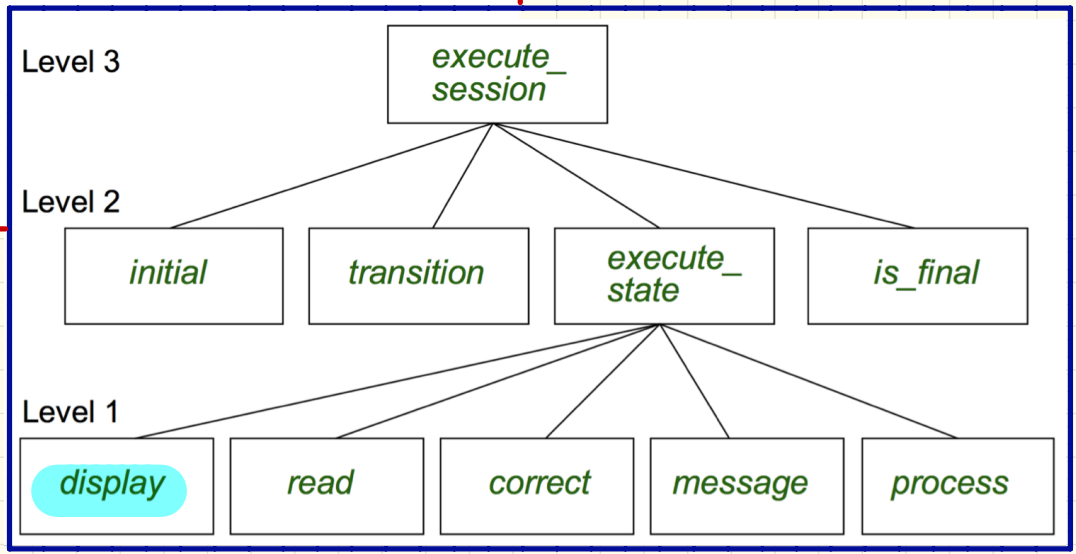
```
execute_state ( current_state : INTEGER): INTEGER
    -- Handle interaction at the current state.
    -- Return user's exit choice.
  local
    answer: ANSWER; valid_answer: BOOLEAN; choice: INTEGER
  do
    from
    until
      valid_answer
    do
      display( current_state )
      answer := read_answer( current_state )
      choice := read_choice( current_state )
      valid_answer := correct( current_state, answer)
      if not valid_answer then message( current_state , answer)
    end
    process( current_state , answer)
    Result := choice
  end
```





Design of a Reservation System: Second Attempt (3)

```
display(current_state: INTEGER)
  require
    valid_state: 1 ≤ current_state ≤ 6
  do
    if current_state = 1 then
      -- Display Initial Panel
    elseif current_state = 2 then
      -- Display Flight Enquiry Panel
    ...
    else
      -- Display
    end
  end
```



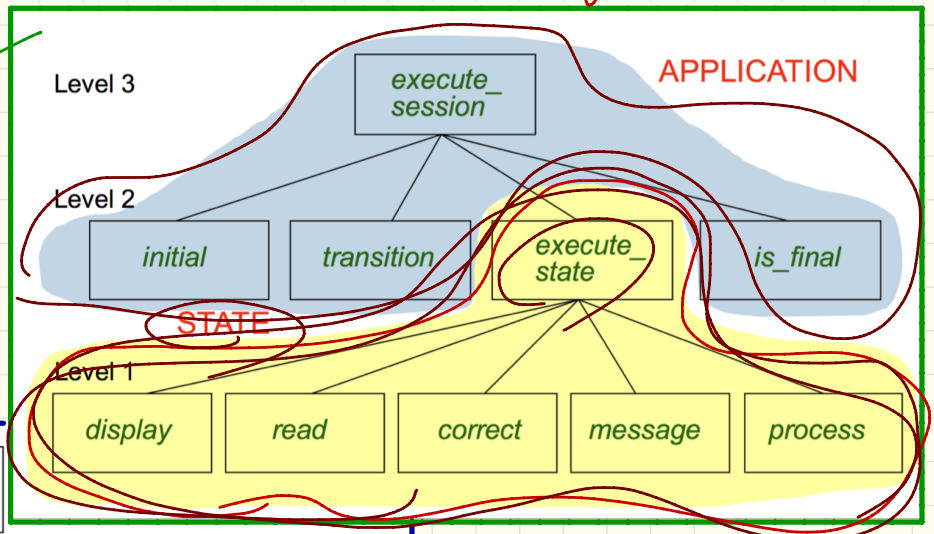
Moving from Hierarchical Design to OO Design

good

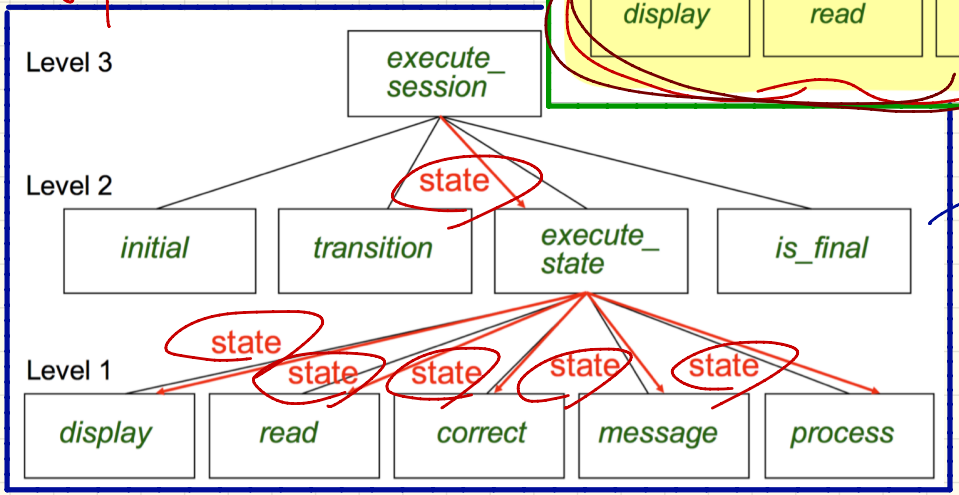
OO

current_state : STATE

current_state . execute_session



poor

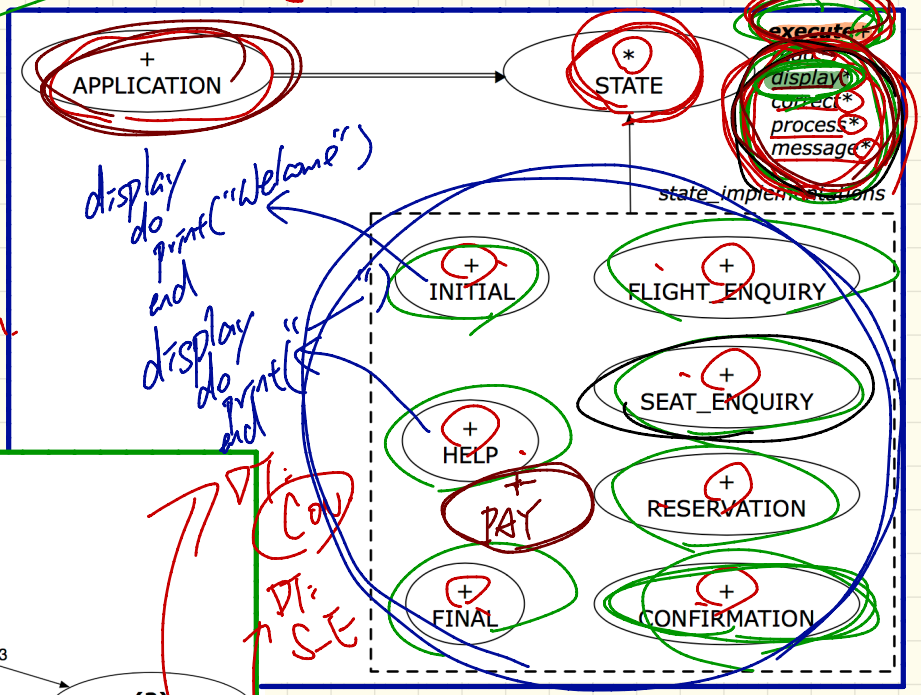


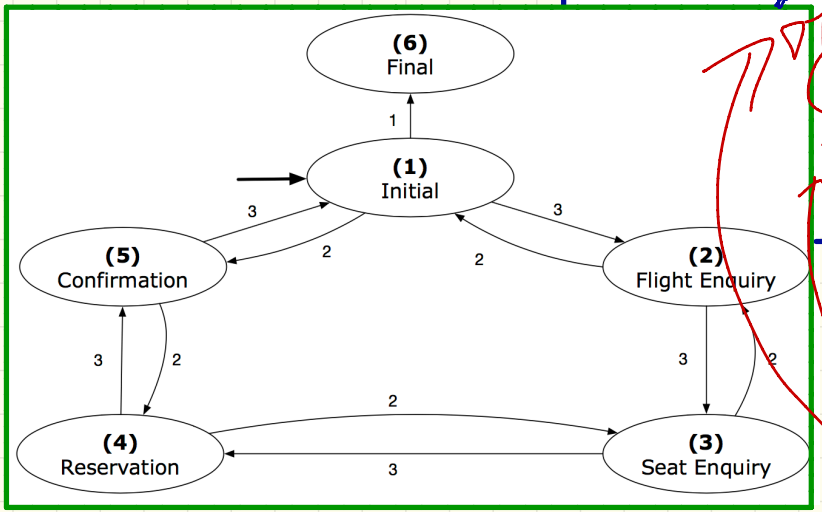
HIERARCHICAL

current_state : INTEGER

execute_session (current_state)

STATE PATTERN Architecture



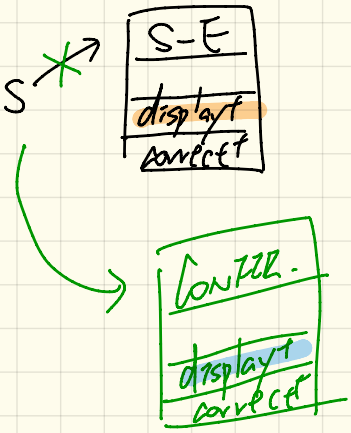



```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

STATE PATTERN : STATE Module

```
deferred class STATE
  read
    -- Read user's inputs
    -- Set 'answer' and 'choice'
    deferred end
  answer: ANSWER
    -- Answer for current state
  choice: INTEGER
    -- Choice for next step
  display
    -- Display current state
    deferred end
  correct: BOOLEAN
    deferred end
  process
    require correct
    deferred end
  message
    require not correct
    deferred end
```

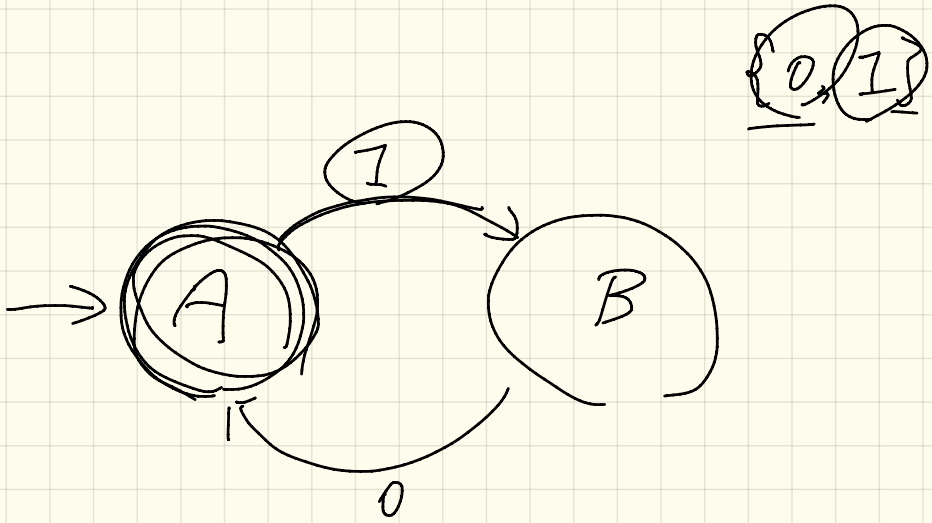
```
execute
  local
    good: BOOLEAN
  do
    from
    until
      good
    loop
      display
      -- Set answer and choice
      read
      good := correct
      if not good then
        message
      end
    end
    process
  end
end
```




```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

due to Dynamic Binding, the DT of current object determines which version of display will be called.

TEMPLATE


{0, 1}
1
A
B
1
0

STATE PATTERN: Application Module

```
class APPLICATION create make
feature {NONE} -- Implementation of Transition Graph
  transition: ARRAY2[INTEGER]
    -- State transitions: transition[state, choice]
  states: ARRAY[STATE]
    -- State for each index, constrained by size of 'transition'
feature
  initial: INTEGER
  number_of_states: INTEGER
  number_of_choices: INTEGER
  make(n, m: INTEGER)
    do number_of_states := n
       number_of_choices := m
       create transition.make_filled(0, n, m)
       create states.make_empty
    end
feature
  put_state(s: STATE; index: INTEGER)
    require 1 ≤ index ≤ number_of_states
    do states.force(s, index) end
  choose_initial(index: INTEGER)
    require 1 ≤ index ≤ number_of_states
    do initial := index end
  put_transition(tar, src, choice: INTEGER)
    require
      1 ≤ src ≤ number_of_states
      1 ≤ tar ≤ number_of_states
      1 ≤ choice ≤ number_of_choices
    do
      transition.put(tar, src, choice)
    end
```

```
invariant
  transition.height = number_of_states
  transition.width = number_of_choices
```

STATE PATTERN : TEST

```
test_application: BOOLEAN
  local
    app: APPLICATION ; current_state: STATE ; index: INTEGER
  do
    create app.make (6, 3)
    app.put_state (create {INITIAL}.make, 1)
    -- Similarly for other 5 states.
    app.choose_initial (1)
    -- Transit to FINAL given current state INITIAL and choice
    app.put_transition (6, 1, 1)
    -- Similarly for other 10 transitions.
    index := app.initial
    current_state := app.states [index]
    Result := attached {INITIAL} current_state
    check Result end
    -- Say user's choice is 3, transit from INITIAL to FLIGHT_STATUS
    index := app.transition.item (index, 3)
    current_state := app.states [index]
    Result := attached {FLIGHT_ENQUIRY} current_state
end
```

index into states array for re-assigning current_state

states

transitions

far

svc

choice

ST:State

D.T. of current_state

INITIAL

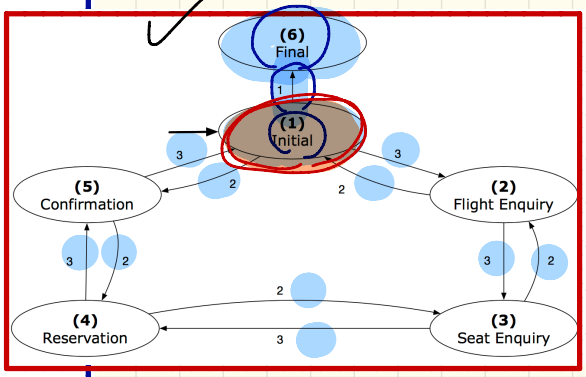
C.S.1

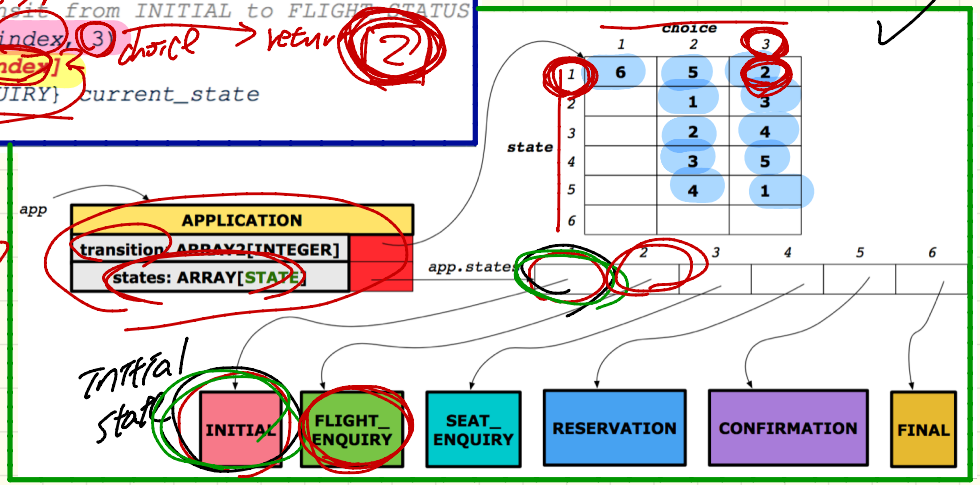
choice

return 2

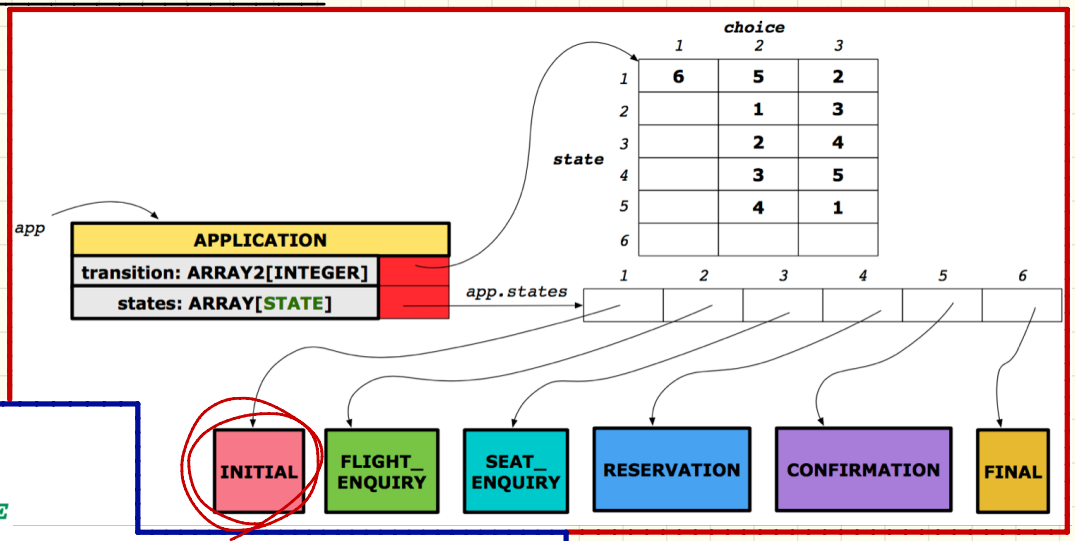
D.T. of current_state?

F_T





STATE PATTERN : Interactive Session




```
feature
  execute_session
    local
      current_state: STATE
      index: INTEGER
    do
      from
        index := initial
      until
        is_final (index)
      loop
        current_state := states[index]  -- polymorphism
        current_state.execute  -- dynamic binding
        index := transition.item (index, current_state.choice)
      end
    end
end
```

C.S.

Tuesday Oct. 30

Lecture 14

Interactive System: Non-OO vs. OO

current_state : STATE

current_state . execute_session

C.O.

create {INITIALIS} cs.make

cs . display

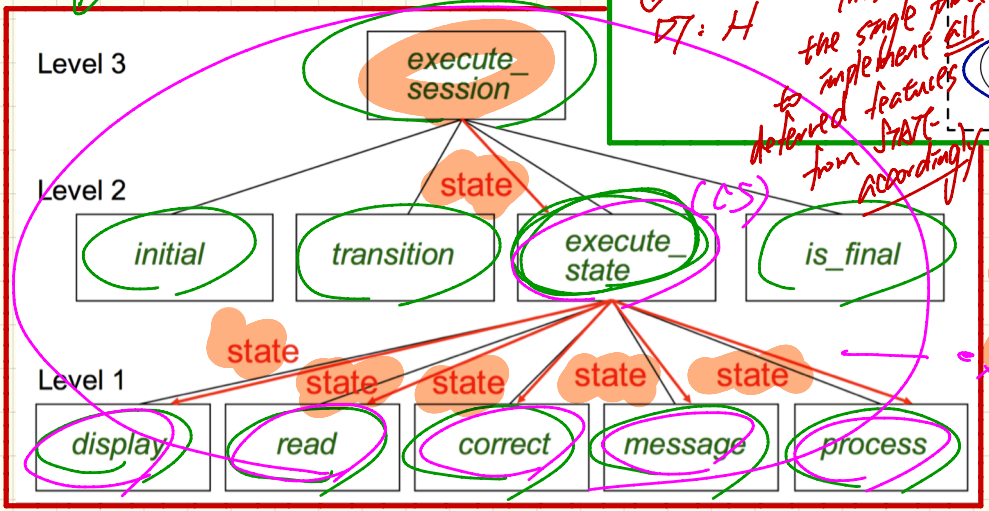
DT: I

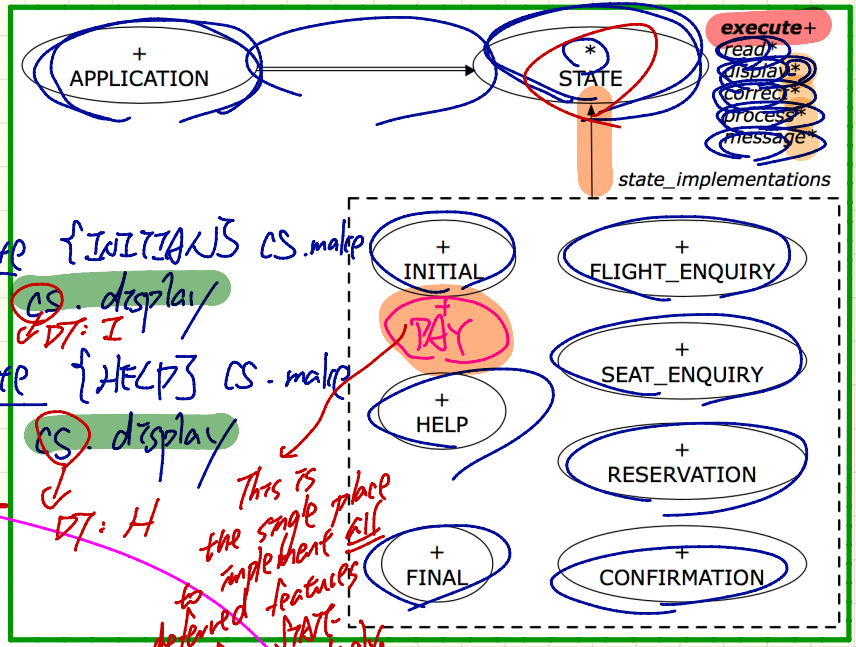
create {HELP} cs . make

cs . display

DT: H

This is the single place to implement all deferred features from STATE accordingly.





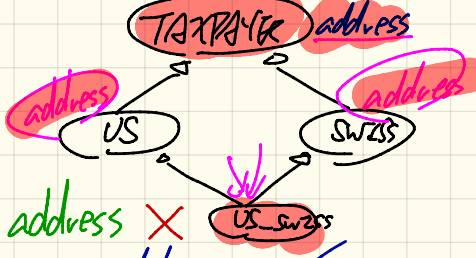
current_state : INTEGER

execute_session (current_state)

Multiple Inheritance : Example (1)

tpl : US_TAXPAYER

tpl : SWISS_US_TAXPAYER



tpl

tpl. address ✗

tpl. us_address ✓

tpl. pay_taxes ✗

tpl. pay_us_taxes ✓

tpl. swiss_address

tpl. swiss_tax_id

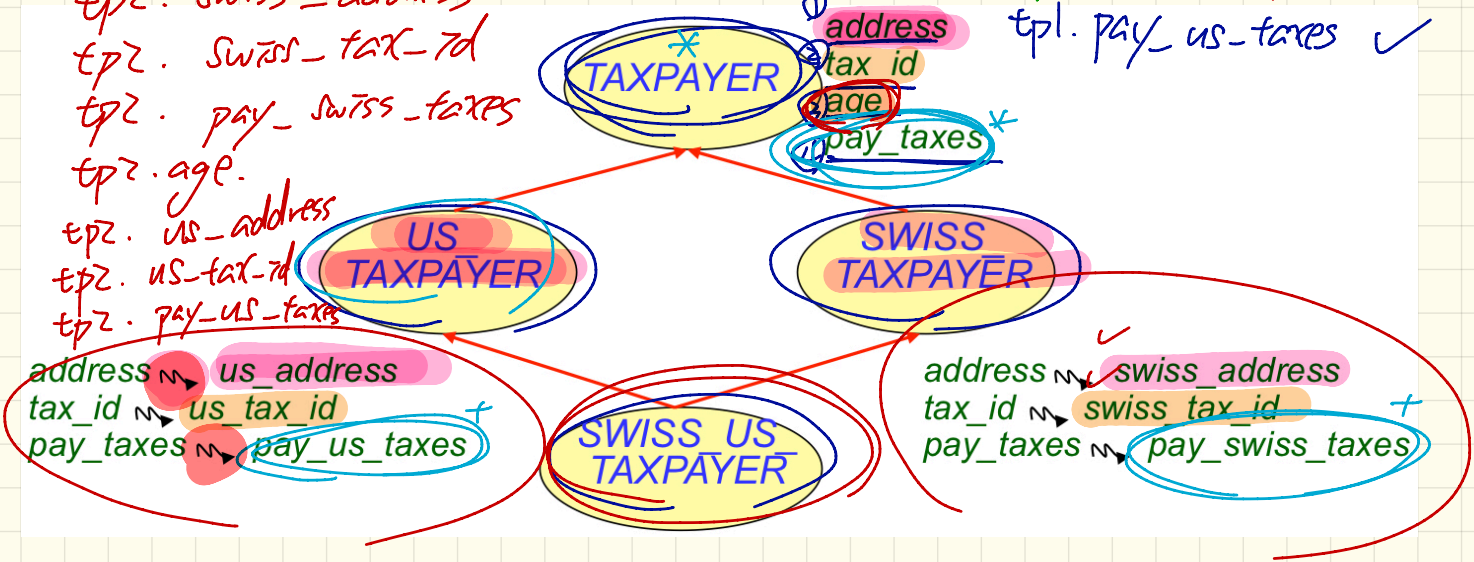
tpl. pay_swiss_taxes

tpl. age.

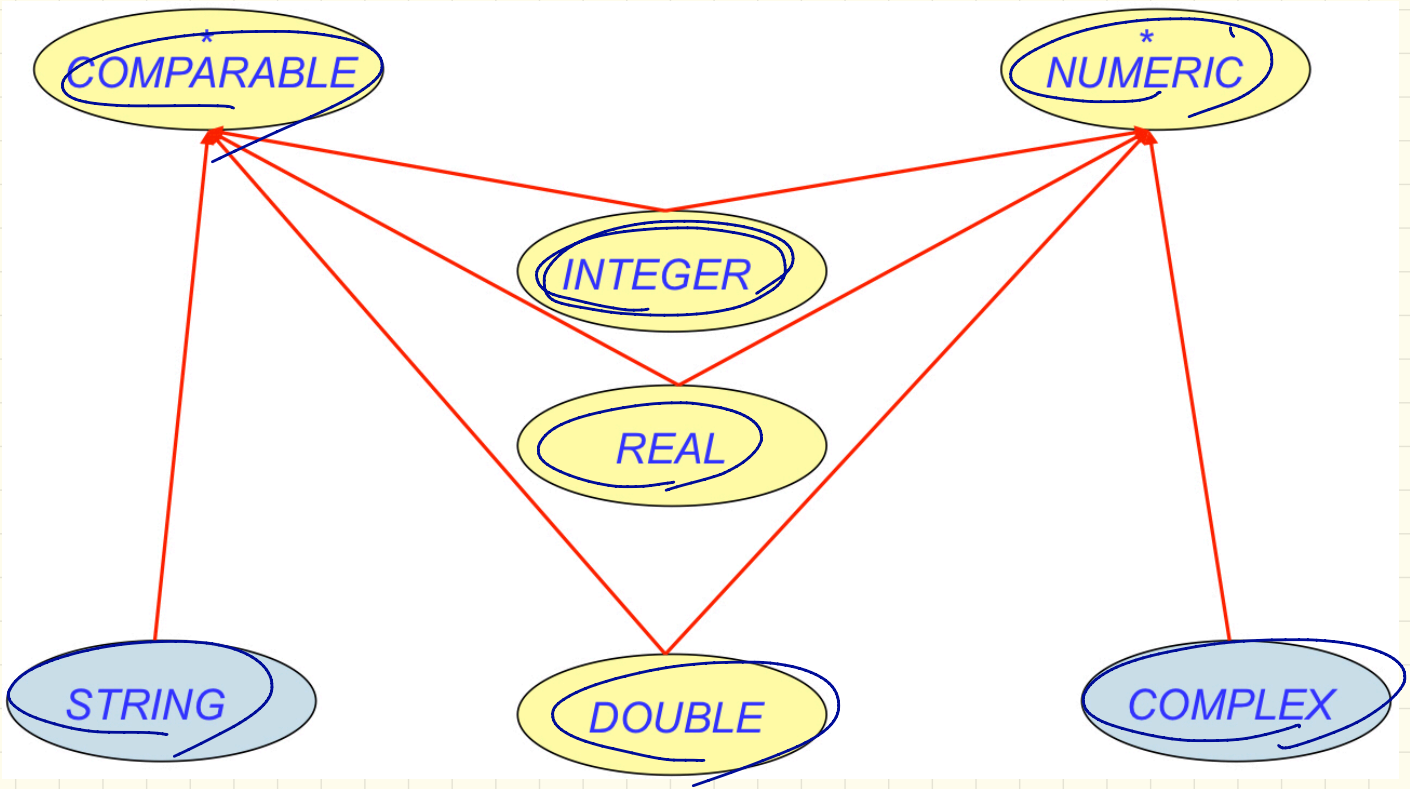
tpl. us_address

tpl. us_tax_id

tpl. pay_us_taxes



Multiple Inheritance : Example (2)

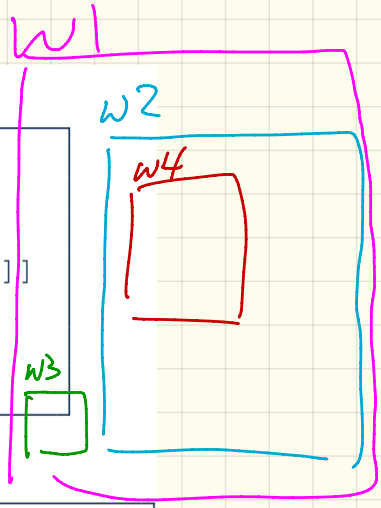


Multiple Inheritance : Exercise

w2.add(w4)
ST: Window

```
class RECTANGLE
  feature -- Queries
    width, height: REAL
    xpos, ypos: REAL
  feature -- Commands
    make (w, h: REAL)
    change_width
    change_height
    move
end
```

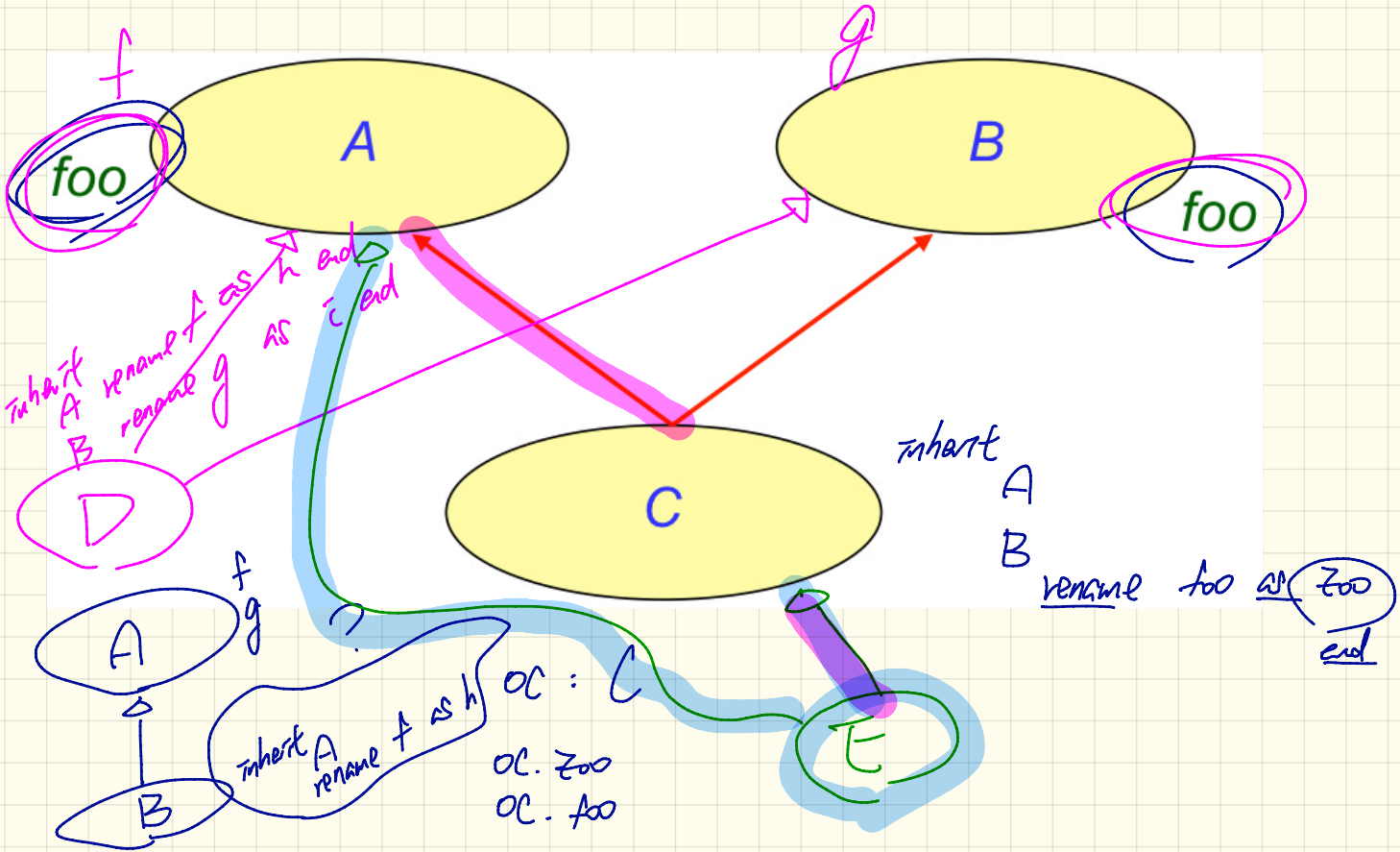
```
class TREE[G]
  feature -- Queries
    parent: TREE[G]
    descendants: LIST[TREE[G]]
  feature -- Commands
    add_child (c: TREE[G])
end
```



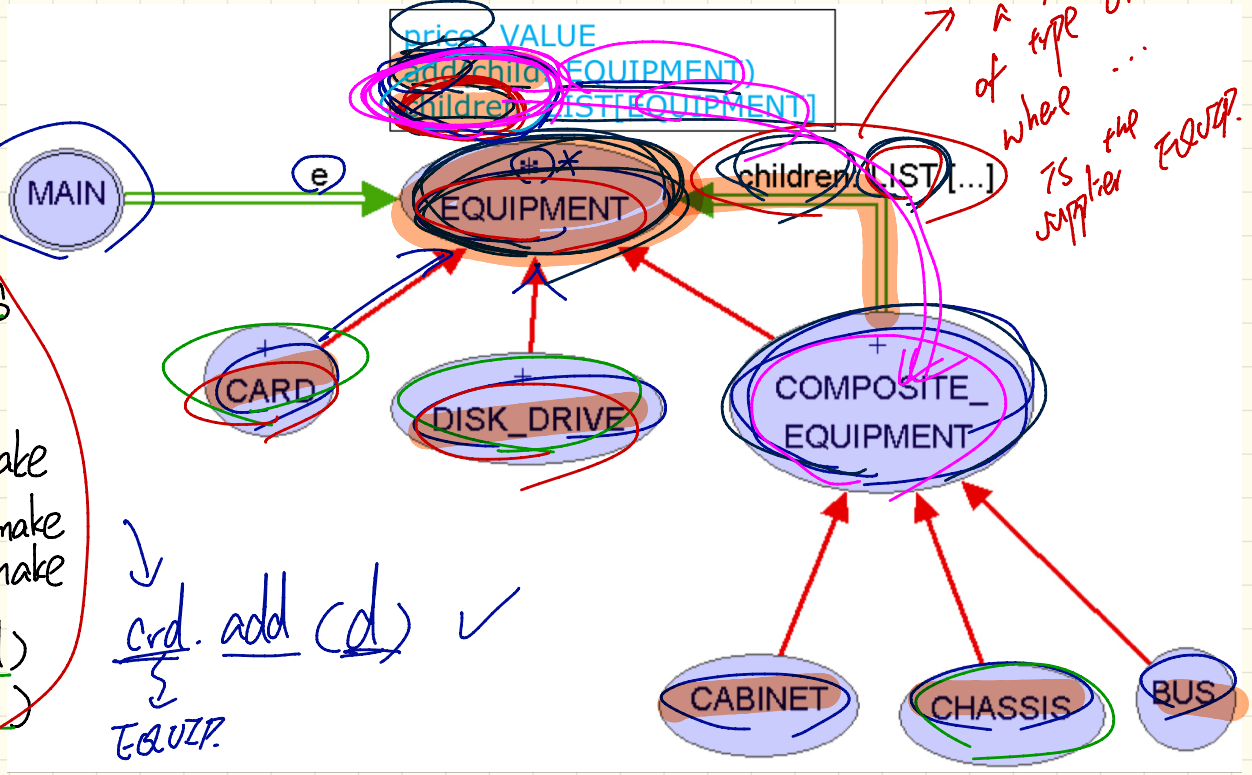

```
class WINDOW
  inherit
    RECTANGLE
    TREE[WINDOW]
feature
  add (w: WINDOW)
end
```

```
test_window: BOOLEAN
  local w1, w2, w3, w4: WINDOW
  do
    create w1.make(8, 6) ; create w2.make(4, 3)
    create w3.make(1, 1) ; create w4.make(1, 1)
    w2.add(w4) ; w1.add(w2) ; w1.add(w3)
    Result := w1.descendants.count = 2
  end
```

Multiple Inheritance: Name Clashes



First Design Attempt



C_E being the client declares feature of type LIST[...] where ... is the supplier EQUIP.

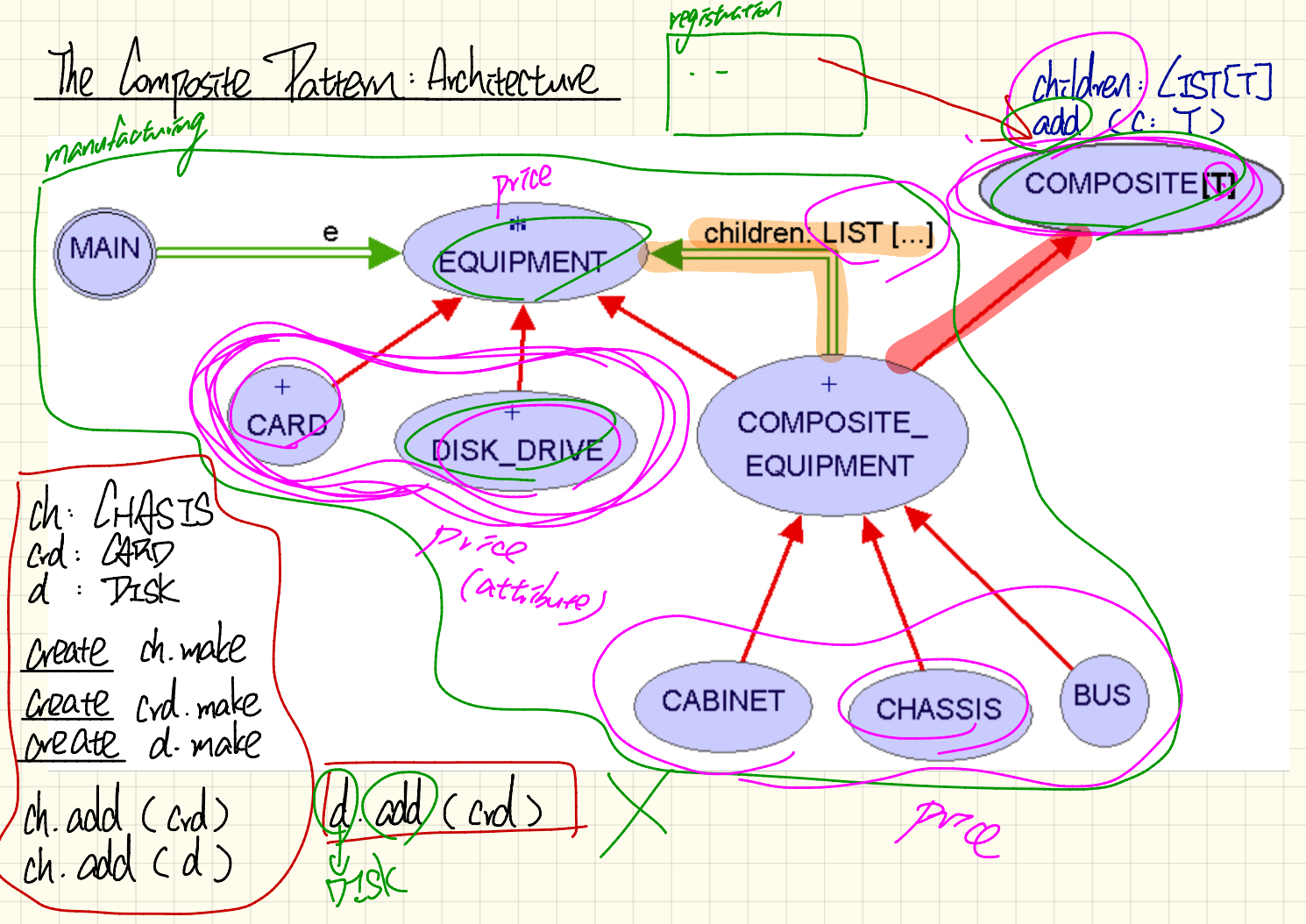
ch: CHASIS
crd: CARD
d : DISK

create ch.make
create crd.make
create d.make

ch.add (crd)
ch.add (d)

crd.add (d) ✓
↓
EQUIP.

The Composite Pattern : Architecture



The Composite Pattern: Implementation

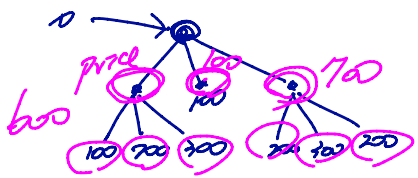
```
deferred class
  EQUIPMENT
feature
  name: STRING
  price: REAL -- uniform access principle
end
```

```
deferred class
  COMPOSITE[T]
feature
  children: LINKED_LIST[T]

  add_child (c: T)
    do
      children.extend (c) -- Polymorphism
    end
end
```

```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is an attribute
    end
end
```

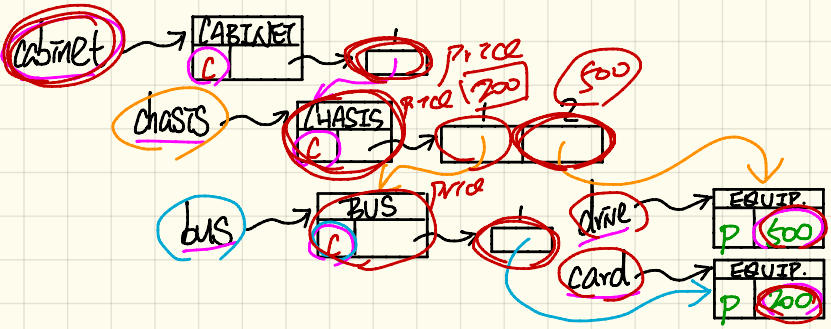
Storage

```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
      -- Sum the net prices of all sub-equipments
    do
      across
        children as cursor
      loop
        Result := Result + cursor.item.price -- dynamic binding
      end
    end
end
```



Testing the Composite Pattern

```
test_composite_equipment: BOOLEAN
  local
    card, drive: EQUIPMENT
    cabinet: CABINET -- holds a CHASSIS
    chassis: CHASSIS -- contains a BUS and a DISK_DRIVE
    bus: BUS -- holds a CARD
  do
    create {CARD} card.make("16Mbs Token Ring", 200)
    create {DISK_DRIVE} drive.make("500 GB harddrive", 500)
    create bus.make("MCA Bus")
    create chassis.make("PC Chassis")
    create cabinet.make("PC Cabinet")

    bus.add(card)
    chassis.add(bus)
    chassis.add(drive)
    cabinet.add(chassis)
    Result := cabinet.price = 700
  end
```



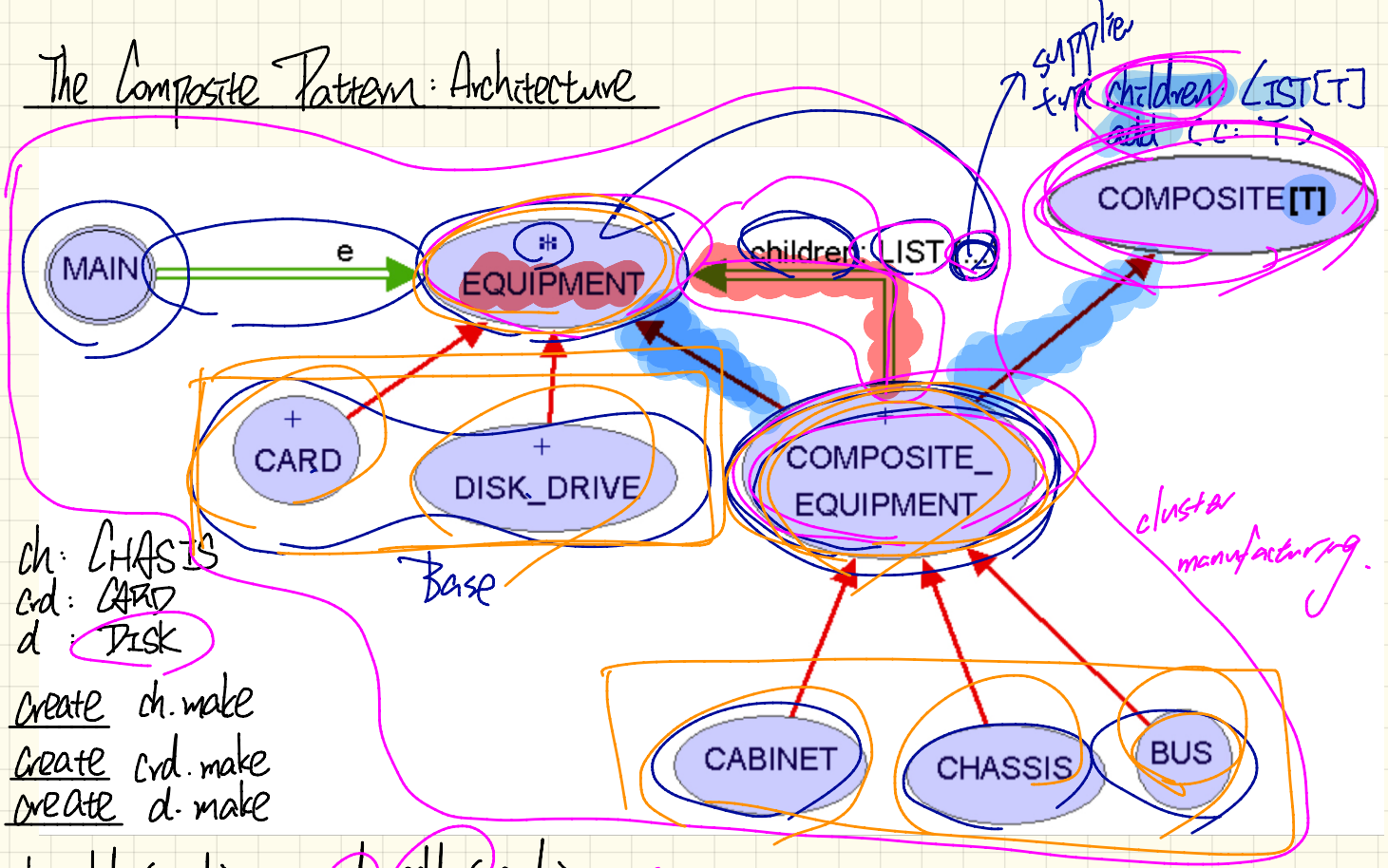

```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is
    end
end
```

```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
      -- Sum the net prices of all sub-equip
    do
      across
        children as cursor
      loop
        Result := Result + cursor.item.price
      end
    end
end
```

Thursday Nov. 7

Lecture 15

The Composite Pattern: Architecture



ch: CHASIS
crd: CARD
d : DISK

create ch.make
create crd.make
create d.make

ch.add (crd)
ch.add (d)

d.add (crd) ×

The Composite Pattern: Implementation

```
deferred class
  EQUIPMENT
feature
  name: STRING
  price: REAL -- uniform access principle
end
```

```
deferred class
  COMPOSITE[T]
feature
  children: LINKED_LIST[T]

  add_child (c: T)
    do
      children.extend (c) -- Polymorphism
    end
end
```

```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is an attribute
    end
end
```

attribute

```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
      -- Sum the net prices of all sub-equipments
    do
      across
        children as cursor
      loop
        Result := Result + cursor.item.price -- dynamic binding
      end
    end
end
```

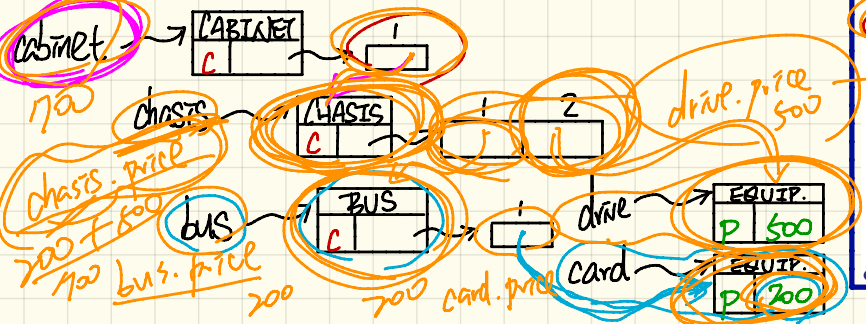
children : LL [EQUIP]

UAP: if the current child is basic ⇒ att.
otherwise ⇒ comput.

Testing the Composite Pattern

```
test_composite_equipment: BOOLEAN
  local
    card, drive: EQUIPMENT
    cabinet: CABINET -- holds a CHASSIS
    chassis: CHASSIS -- contains a BUS and a DISK_DRIVE
    bus: BUS -- holds a CARD
  do
    create {CARD} card.make("16Mbs Token Ring", 200)
    create {DISK_DRIVE} drive.make("500 GB harddrive", 500)
    create bus.make("MCA Bus")
    create chassis.make("PC Chassis")
    create cabinet.make("PC Cabinet")

    bus.add(card)
    chassis.add(bus)
    chassis.add(drive)
    cabinet.add(chassis)
    Result := (cabinet.price = 700)
  end
```

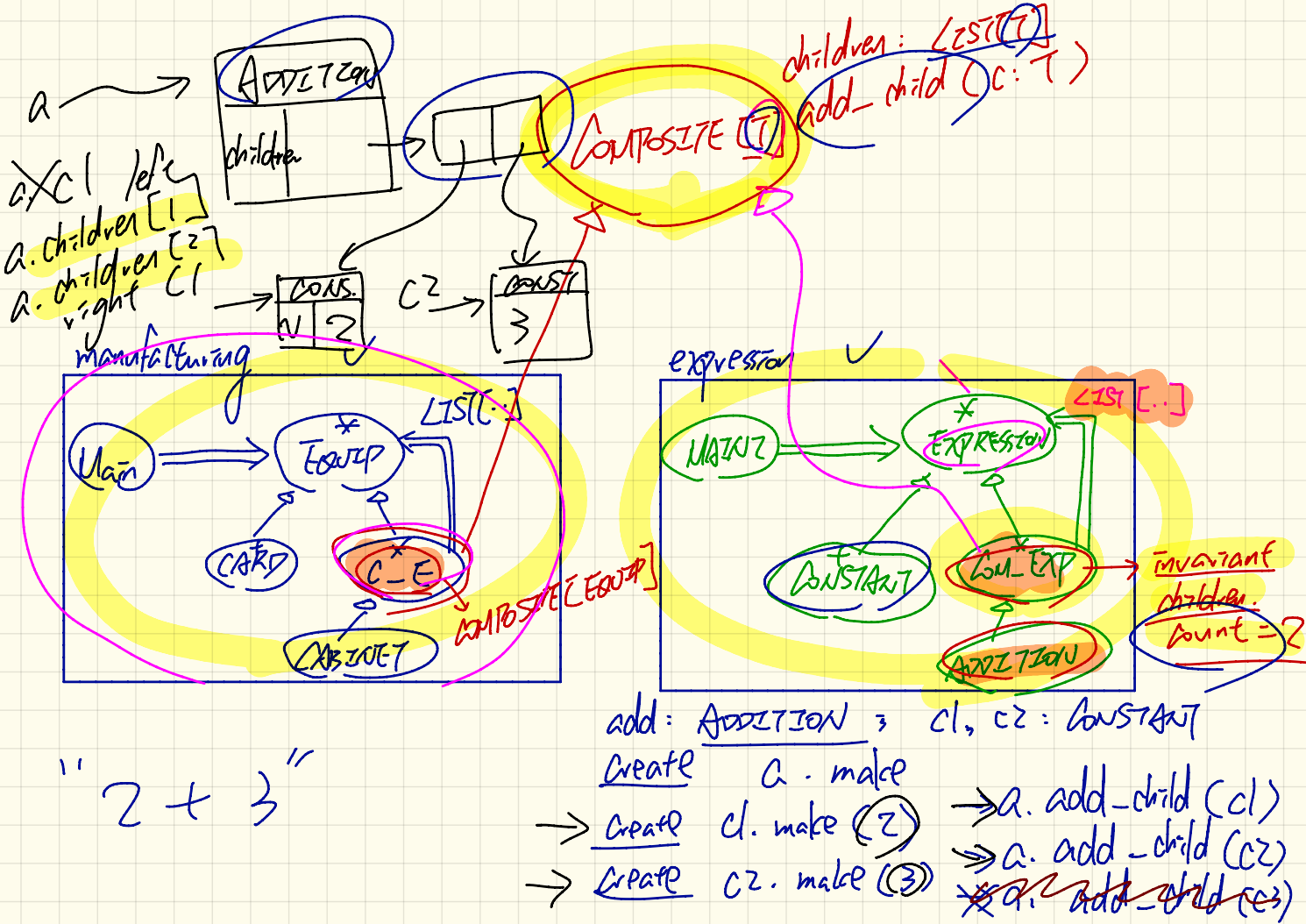
DT: CABINET



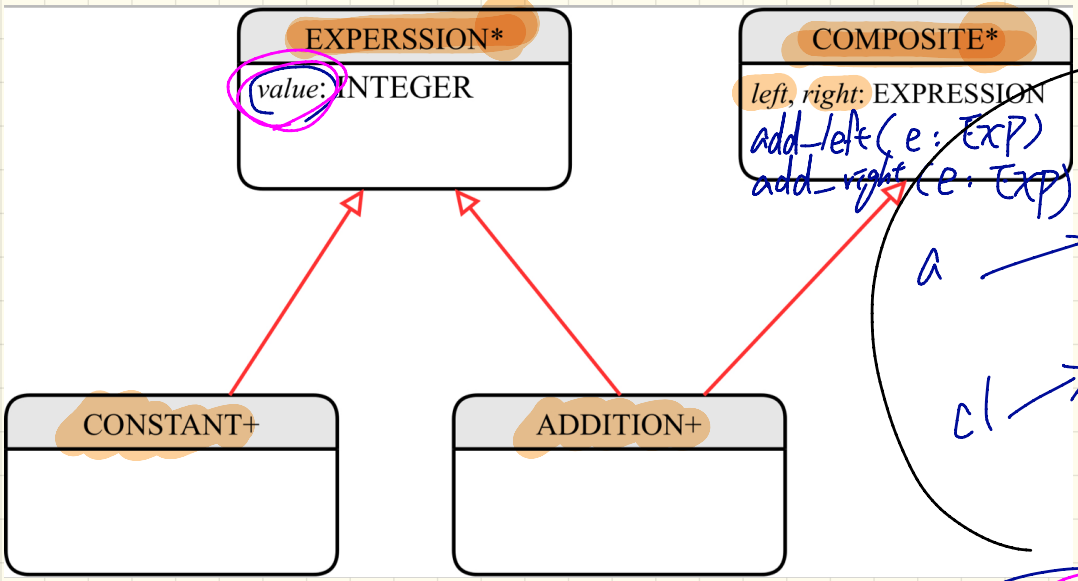

```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is
    end
end
```

```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
    -- Sum the net prices of all sub-equip
    do
      across
        children as cursor
      loop
        Result := Result + cursor.item.price
      end
    end
end
```

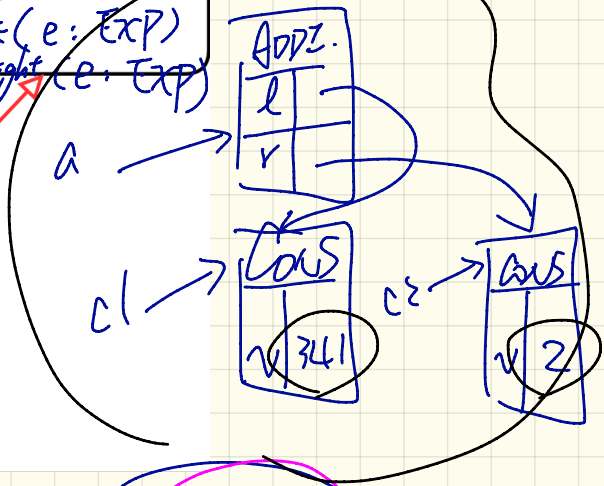
$$\frac{341}{(2+3)} + \frac{\quad}{((4+7)+9)}$$


a
ADDITION
children
CONSTANT
children : LIST[T]
add_child (c : T)
COMPOSITE [T]
a.children[1]
a.children[2]
right c1
c2
manufacturing
Main
EQUIP
LIST[..]
CARD
C_E
ABSTRACT
COMPOSITE[EQUIP]
expression
MAIN
EXPRESSION
CONSTANT
COM_EXP
ADDITION
invariant
children.
count = 2
add : ADDITION ; c1, c2 : CONSTANT
create a.make
create c1.make (2)
create c2.make (3)
a.add_child (c1)
a.add_child (c2)
a.add_child (c3)
"2 + 3"

Design of Language Structure : Composite Pattern



a.value
a.left,
a.right



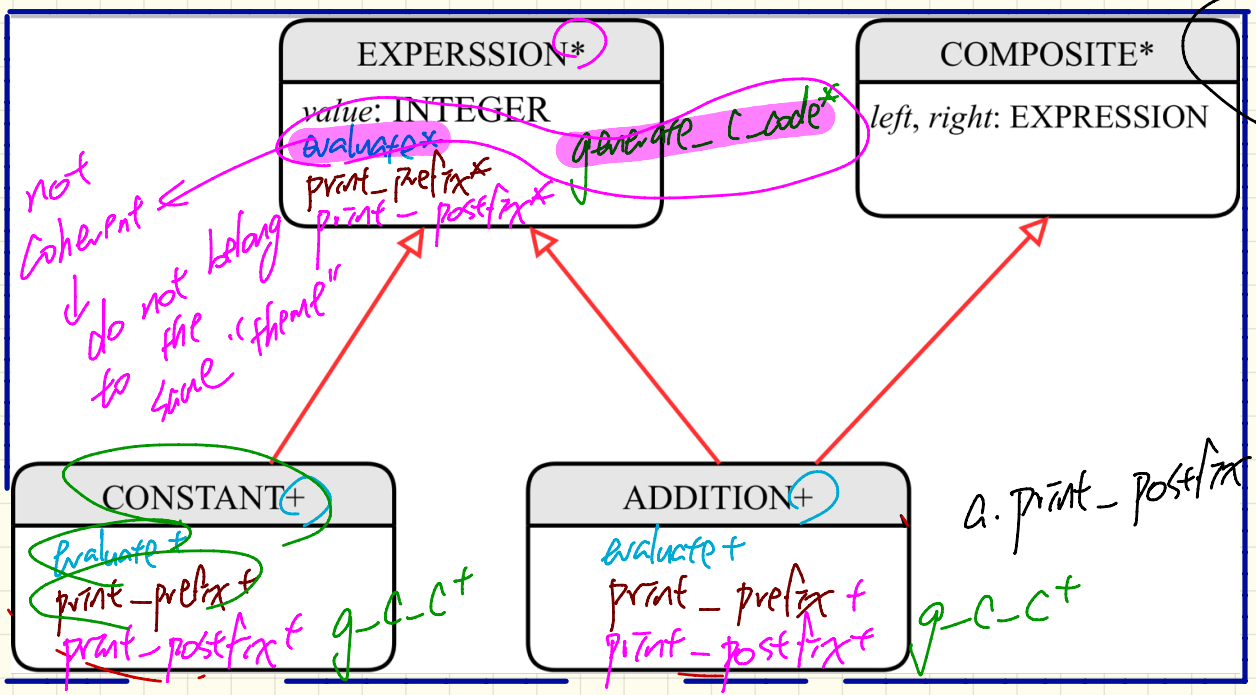
Q: How do you construct an object representing "341 + (2)"?

c1, c2: CONSTANT
a: ADDITION

Create c1.make (341)
Create c2.make (2)

Create a.make
a.add_left (c1)
a.add_right (c2)

Design of Language Operations : How to Extend the Composite Pattern?



not Coherent ↓ do not belong to the "theme"

Structure

a. print_postfix 341 2 +

Operations: → evaluate
print_prefix
print_postfix
type_check

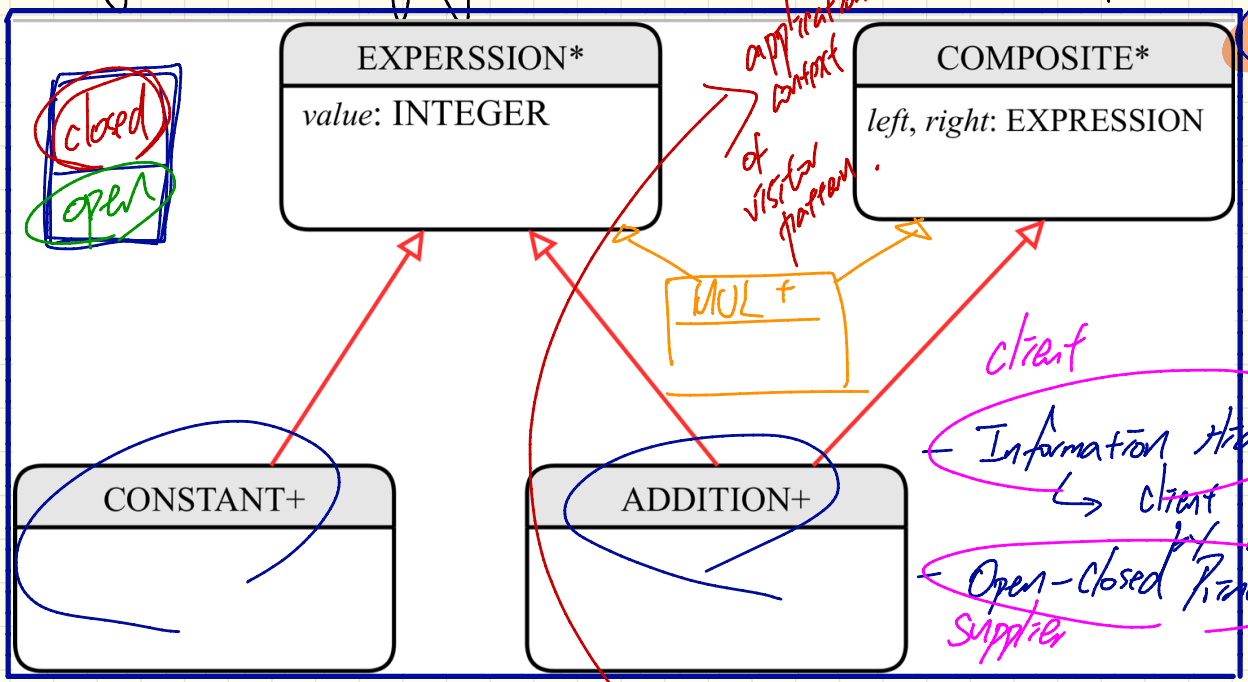
Operations

a 341 + 2

a. evaluate 343

a. print_prefix + 341 2

Design of a Language Application : Open-Closed Principle



Structure of the expression language

client
- Information Hiding
  - client not affected consciously chosen design decision
- Open-closed Principle
supplier

Operations: evaluate, print_prefix, print_postfix, type_check

Operations

generate_java_code

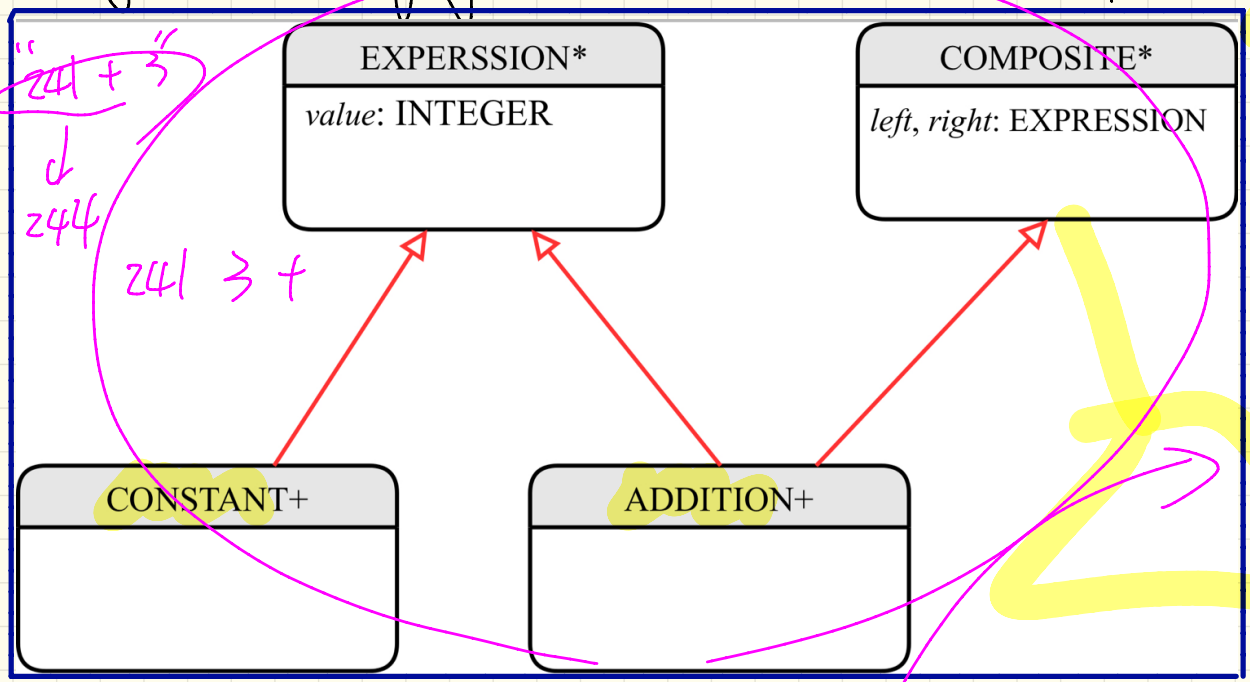
	Structure	Operations
Alt. 1	open	closed
Alt. 2	closed	open

Tuesday Nov. 6

Lecture 16

- Exam: Sunday Dec. 9 7pm
- Midterm results available this Thursday
- Lab 3 (programming) marks available early Friday
- Project released by next Wed.
  (3 weeks)

Design of a Language Application: Open-Closed Principle



Structure

"24 + 3"

24 3 +

app. context of Visitor

Operations: evaluate, print_prefix, print_postfix, type_check

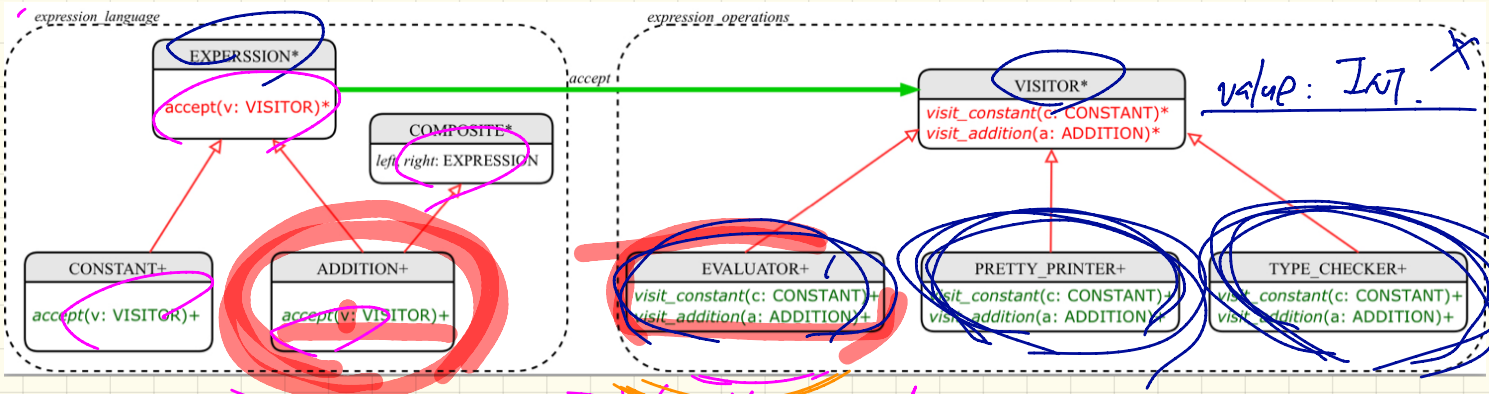
Operations

gene_

	Structure	Operations
Alt. 1	open	closed
Alt. 2	closed	open

Visitor Design Pattern: Architecture

add accept 234 + "a"



effective descendants of EXPRESSION

create a visit_x feature in VISITOR

How to Use Visitors

add.accept(v) ⇒

c1.accept(v) → 1 3

c2.accept(v) → 2

```
1  test_expression_evaluation: BOOLEAN
2    local add, c1, c2: EXPRESSION ; v: VISITOR
3    do
4      create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
5      create {ADDITION} add.make (c1, c2)
6      create {EVALUATOR} v.make
7      add.accept (v)
8      check attached {EVALUATOR} v as eval then
9        Result := eval.value = 3
10     end
11   end
```

what you have to write

e: 234 + 2

236

eval.visit_add(e)

PP.visit_add(e)

"234 2 +"

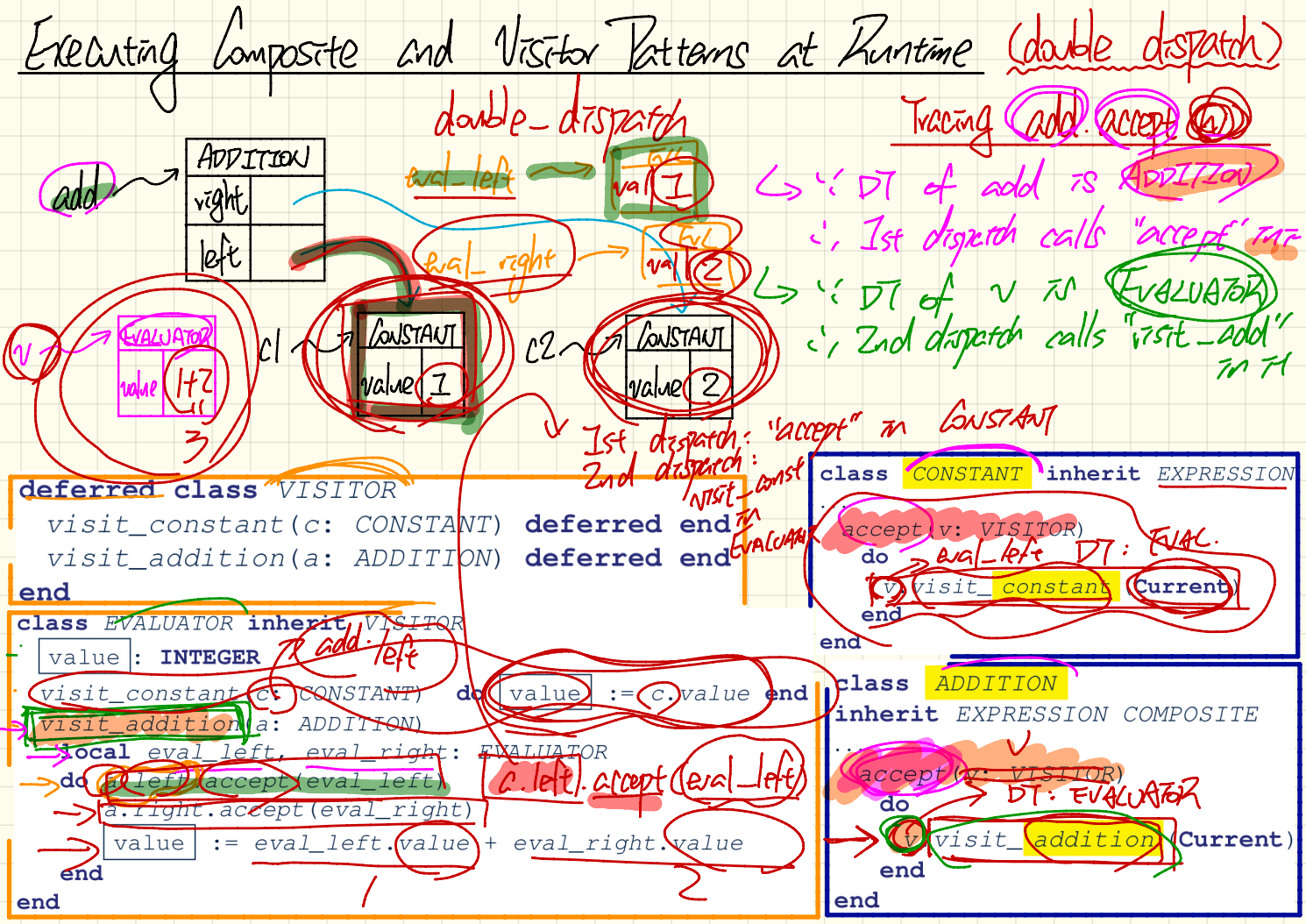
Visitor Design Pattern : Implementation

```
1  test_expression_evaluation: BOOLEAN
2   local add, c1, c2: EXPRESSION ; v: VISITOR
3   do
4    create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
5    create {ADDITION} add.make (c1, c2)
6    create {EVALUATOR} v.make
7    add.accept (v)
8    check attached {EVALUATOR} v as eval then
9     Result := eval.value = 3
10   end
11  end
```

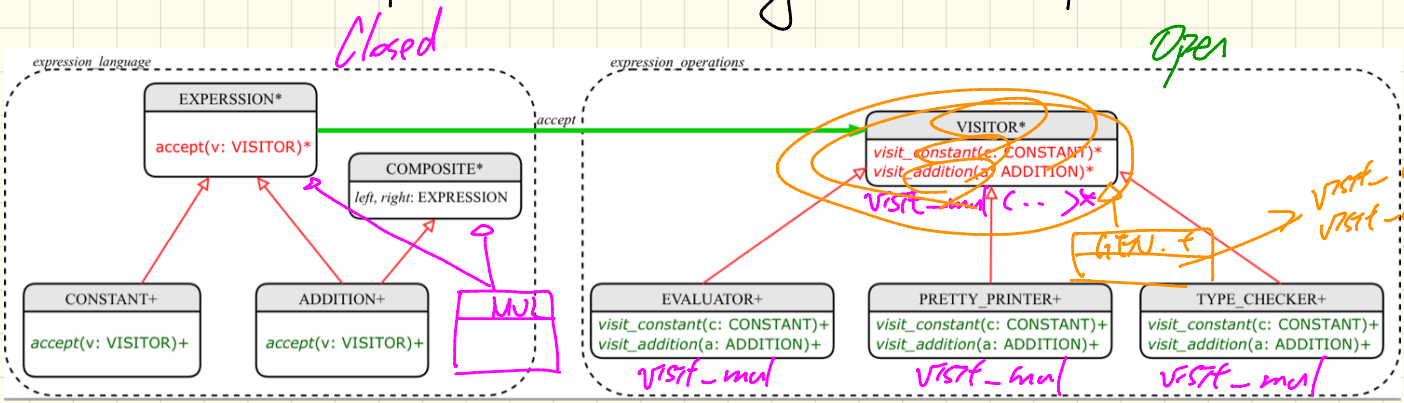
Visualizing Line 4 to Line 7

1 + 2

Executing Composite and Visitor Patterns at Runtime (double dispatch)



Visitor Pattern: Open-Closed and Single Choice Principles



Adding a new language construct? → violates SCP ⇒ this part should be closed

Adding a new language operation? → satisfied SCP ⇒ this part can be open

GEN_ASSEMBLY

Void Safe in Java? (1)

```
1  class Point {
2    double x;
3    double y;
4    Point(double x, double y) {
5      this.x = x;
6      this.y = y;
7  }
```

```
1  class PointCollector {
2    ArrayList<Point> points;
3    PointCollector() { }
4    void addPoint(Point p) {
5  points.add(p); }
6    Point getPointAt(int i) {
7      return points.get(i); } }
```

The above Java code ***compiles***. But anything wrong?

```
1  @Test
2  public void test1() {
3    PointCollector pc = new PointCollector();
4    pc.addPoint(new Point(3, 4));
5    Point p = pc.getPointAt(0);
6    assertTrue(p.x == 3 && p.y == 4); }
```

pc.points null

Void Safe in Java? (2)

```
1  class Point {
2    double x;
3    double y;
4    Point(double x, double y) {
5      this.x = x;
6      this.y = y;
7  }
```

```
1  class PointCollector {
2    ArrayList<Point> points;
3    PointCollector() {
4      points = new ArrayList<>(); }
5    void addPoint(Point p) {
6      points.add(p); }
7    Point getPointAt(int i) {
8      return points.get(i); } }
```

```
1  @Test
2  public void test2() {
3    PointCollector pc = new PointCollector();
4    Point p = null;
5    pc.addPoint(p);
6    p = pc.getPointAt(0);
7    assertTrue(p.x == 3 && p.y == 4); }
```

null

null

Thursday Nov. 8

Lecture 17

Void Safe in Java? (3)

```
1  class Point {
2    double x;
3    double y;
4    Point(double x, double y) {
5      this.x = x;
6      this.y = y;
7  }
```

```
1  class PointCollector {
2    ArrayList<Point> points;
3    PointCollector() {
4      points = new ArrayList<>(); }
5    void addPoint(Point p) {
6      points.add(p); }
7    Point getPointAt(int i) {
8      return points.get(i); } }
```

```
1  public void test3() {
2    PointCollector pc = new PointCollector();
3    Scanner input = new Scanner(System.in);
4    System.out.println("Ener an integer:");
5    int i = input.nextInt();
6    if (i < 0) { pc = null; }
7    pc.addPoint(new Point(3, 4));
8    assertTrue(pc.getPointAt(0).x == 3 && pc.getPointAt(0).y == 4);
9  }
```

Non-detachable vs. Detachable

nullable

attached

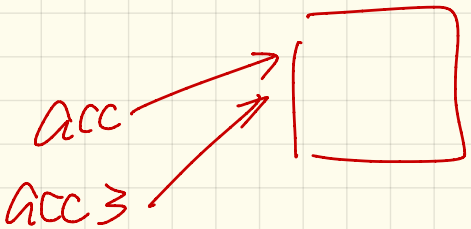
acc → ACC

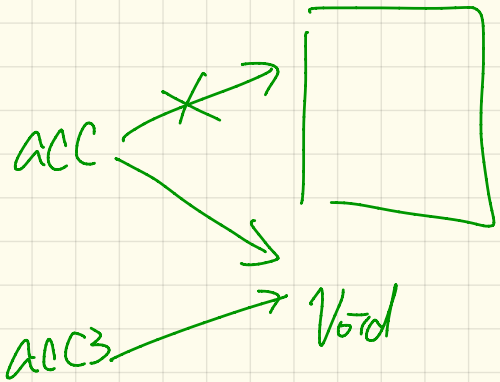
null = void

acc	ACCOUNT	
acc2	ACCOUNT	
acc3	detachable	ACCOUNT

!!! Account acc3; Java

1. acc := acc2 ✓
2. acc := acc3 ✗
3. acc3 := acc





Void Safe in Eiffel ! (1)

```
1  class
2    POINT
3  create
4    make
5  feature
6    x: REAL
7    y: REAL
8  feature
9    make (nx: REAL; ny: REAL)
10     do x := nx
11        y := ny
12     end
13 end
```

```
1  class
2    POINT_COLLECTOR_1
3  create
4    make
5  feature
6    points: LINKED_LIST[POINT]
7  feature
8    make do end
9  add_point (p: POINT)
10   do points.extend (p) end
11 get_point_at (i: INTEGER): POINT
12   do Result := points [i] end
13 end
```

Void Safe in Eiffel ! (2)

```
1  class
2    POINT
3  create
4    make
5  feature
6    x: REAL
7    y: REAL
8  feature
9    make (nx: REAL; ny: REAL)
10     do x := nx
11        y := ny
12     end
13 end
```

```
1  class
2    POINT_COLLECTOR_2
3  create
4    make
5  feature
6    points: LINKED_LIST[POINT]
7  feature
8    make do create points.make end
9  add_point (p: POINT)
10   do points.extend (p) end
11 get_point_at (i: INTEGER): POINT
12   do Result := points [i] end
13 end
```

```
1  test_2: BOOLEAN
2    local
3      pc: POINT_COLLECTOR_2 ; p: POINT
4    do
5      create pc.make
6      p := Void
7      pc.add_point (p)
8      p := pc.get_point_at (0)
9      Result := p.x = 3 and p.y = 4
10   end
```

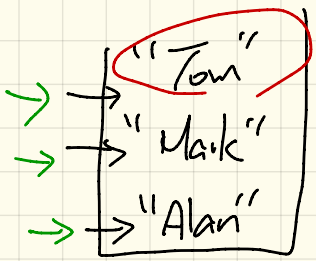
the 1st usage of pc that might be void

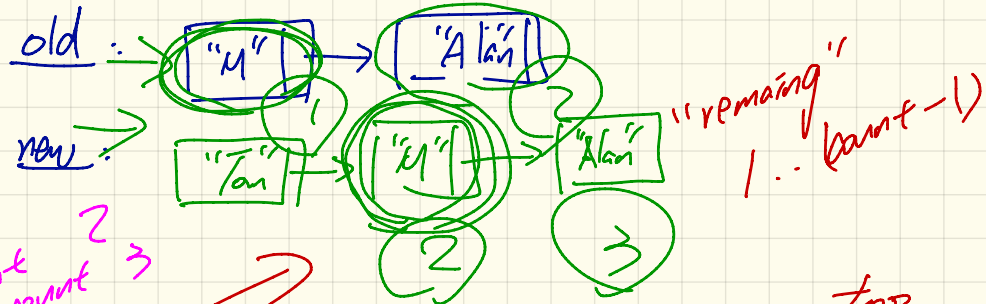
Void Safe in Eiffel ! (3)

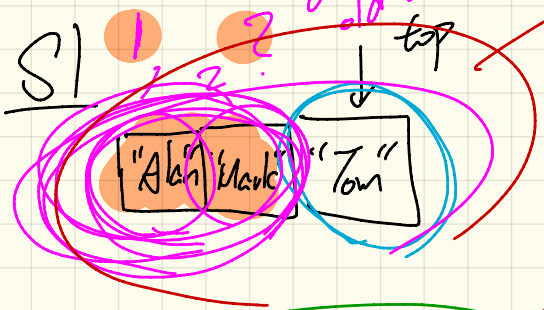
```
1  class
2    POINT
3  create
4    make
5  feature
6    x: REAL
7    y: REAL
8  feature
9    make (nx: REAL; ny: REAL)
10     do x := nx
11        y := ny
12     end
13 end
```

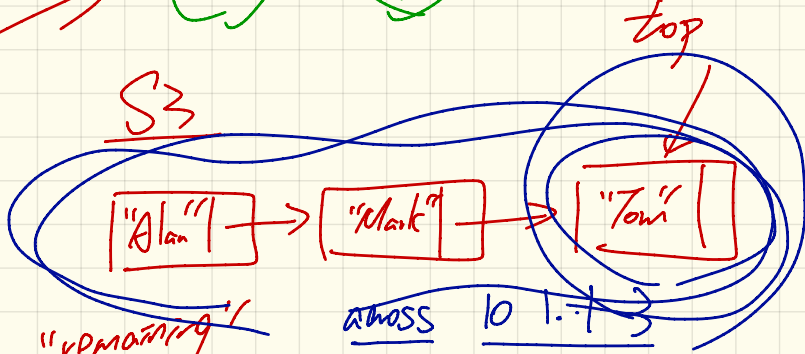
```
1  class
2    POINT_COLLECTOR_2
3  create
4    make
5  feature
6    points: LINKED_LIST[POINT]
7  feature
8    make do create points.make end
9  add_point (p: POINT)
10   do points.extend (p) end
11 get_point_at (i: INTEGER): POINT
12   do Result := points [i] end
13 end
```

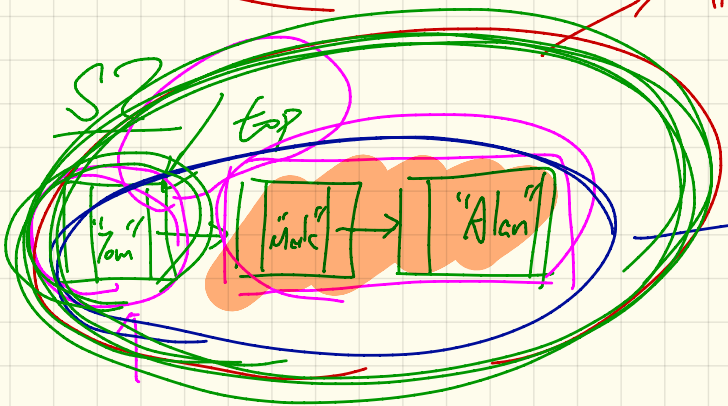
```
1  test_3: BOOLEAN
2    local pc: POINT_COLLECTOR_2 ; p: POINT ; i: INTEGER
3    do create pc.make
4       io.print ("Enter an integer:%N")
5       io.read_integer
6       if io.last_integer < 0 then pc := Void end
7       pc.add_point (create {POINT}.make (3, 4))
8       p := pc.get_point_at (0)
9       Result := p.x = 3 and p.y = 4
10   end
```











push (g)

remaining_unchanged :

across (2) 1..1 count as c

all

tmp[c.item] ~ (old tmp.d_t).[c.item]

end

Developing a LIFO STACK

- information hiding
- single-choice principle

when changing strategy, flags to change multiple stacks in both imp. and some client

change of strategy

1. imp is private but mentioned in postcondition
2. when changing the type of imp, client view affected.

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 1: array
  imp: ARRAY[G]
feature -- Initialization
  make do create imp.make_empty ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.force(g, imp.count + 1)
    ensure
      changed: imp[count] ~ g
      unchanged: across 1 |..| count - 1 as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
  pop
    do imp.remove_tail(1)
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
```

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 2: linked-list first item as top
  imp: LINKED_LIST[G]
feature -- Initialization
  make do create imp.make ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.put_front(g)
    ensure
      changed: imp.first ~ g
      unchanged: across 2 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
  pop
    do imp.start ; imp.remove
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item + 1] end
    end
```

solution: have a common aspect to specify the contract with.

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 3: linked-list last item as top
  imp: LINKED_LIST[G]
feature -- Initialization
  make do create imp.make ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.extend(g)
    ensure
      changed: imp.last ~ g
      unchanged: across 1 |..| count - 1 as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
  pop
    do imp.finish ; imp.remove
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
```

Using MATHMODELS Library

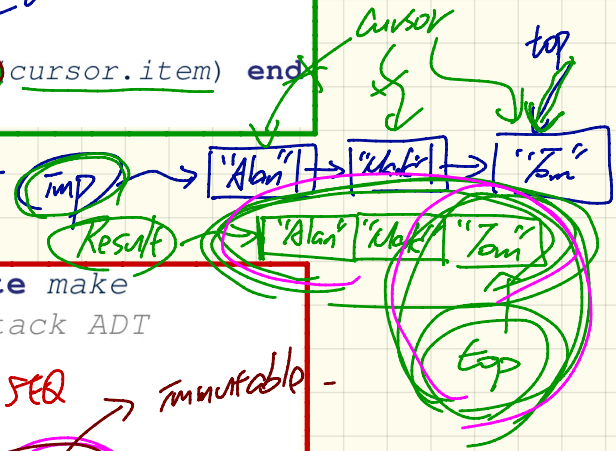
Implementing Abstraction Function

model query: convert from imp to SEQ.

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.append(cursor.item) end
    end
```

prepend command

SEQ



Writing Contracts using Abstraction Function

```
class LIFO_STACK[G -> attached ANY] create make
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
feature -- Commands
  push (g: G)
    ensure model ~ (old model.deep_twin).appended(g) end
```

conversion of the old imp to SEQ

immutable -

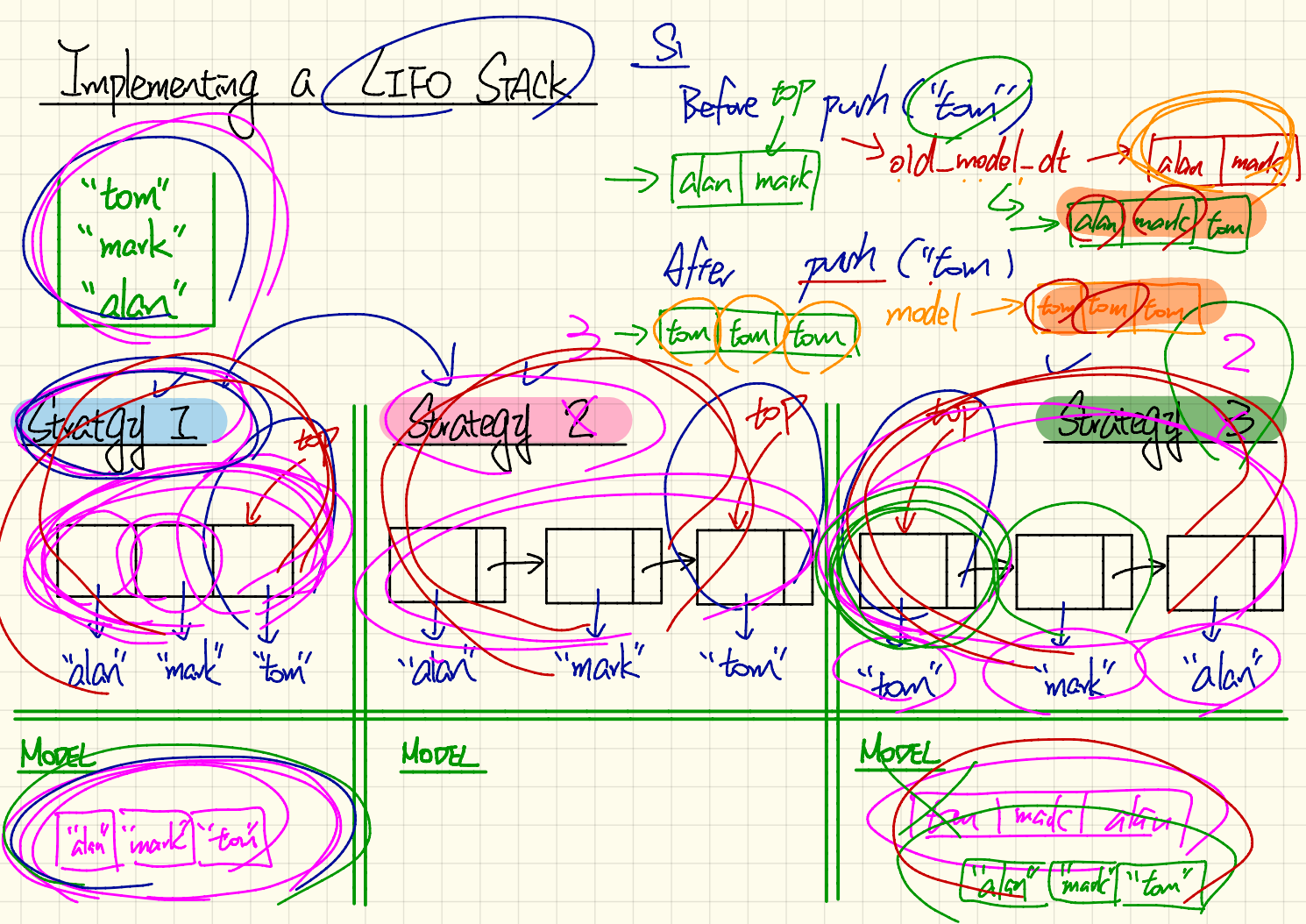
conversion of the new imp to SEQ

model ~ (old model).append(g) X not compile

Tuesday Nov. 13

Lecture 18

Implementing a LIFO STACK



Using MATHMODELS Library

Implementing Abstraction Function

(old model.deep_twin).appended(g)

call it to the query

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
 imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
 model: SEQ[G]
  do create Result.make_empty
    across imp as cursor loop Result.append(cursor.item) end
  end
```

prepend

command

SEQ

converting from imp into SEQ

Writing Contracts using Abstraction Function

```
class LIFO_STACK[G -> attached ANY] create make
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
feature -- Commands
 push (g: G)
   ensure model ~ old model.deep_twin.appended(g) end
```

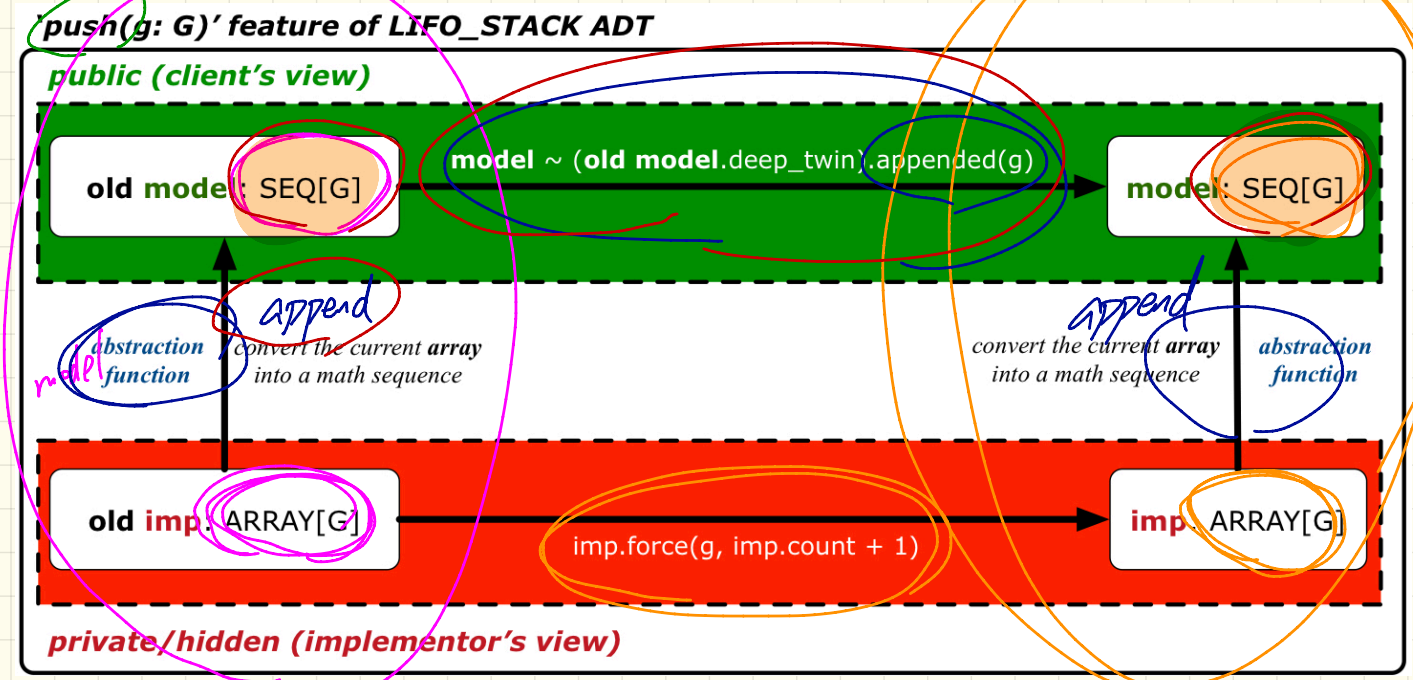
query

after push is executed

before

immutable query

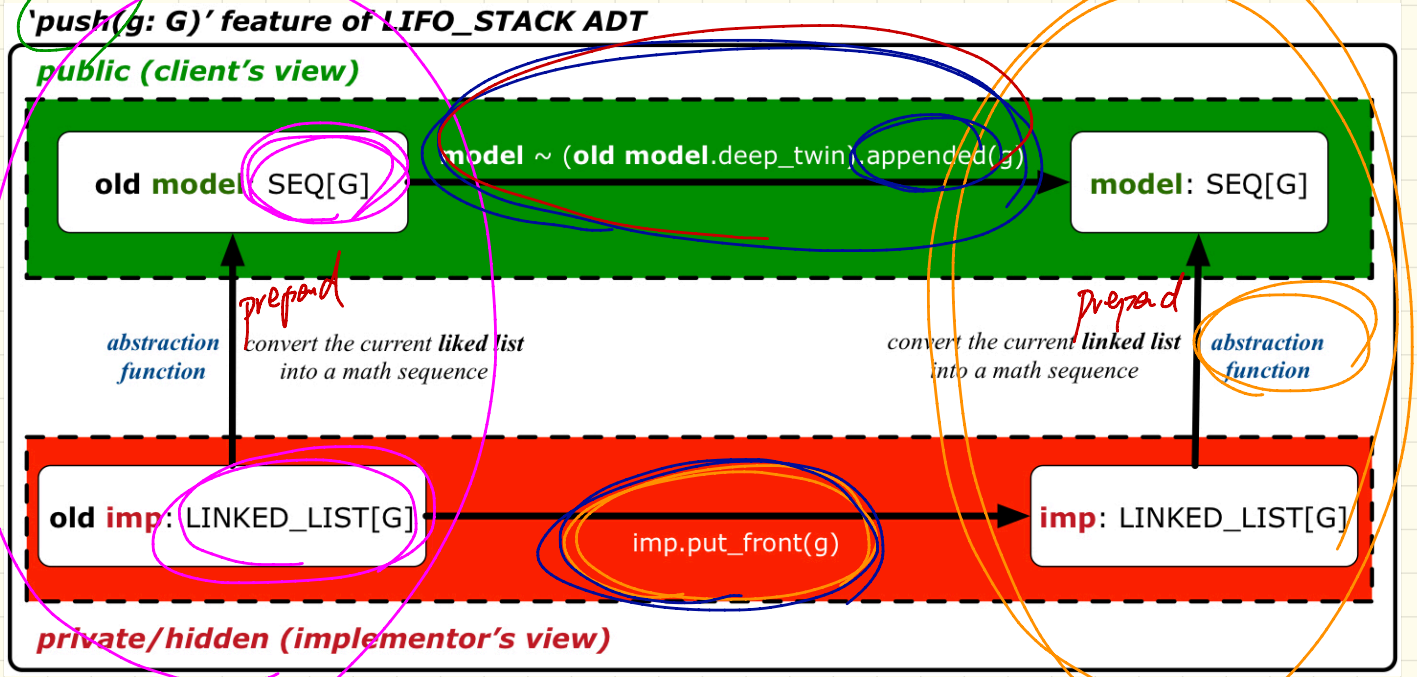
Strategy 1 : Mathematical Abstraction



before

after

Strategy 2: Mathematical Abstraction



```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 1
  imp: ARRAY[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_from_array (imp)
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[i.item]
    end
feature -- Commands
  make do create imp.make_empty ensure model.count = 0 end
  push (g: G) do imp.force(g, imp.count + 1)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.remove_tail(1)
    ensure popped: model ~ (old model.deep_twin).front end
end
```

C1

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 2 (first as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.prepend(cursor.item) end
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[count - i.item + 1]
    end
feature -- Commands
  make do create imp.make ensure model.count = 0 end
  push (g: G) do imp.put_front(g)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.start ; imp.remove
    ensure popped: model ~ (old model.deep_twin).front end
end
```

C2

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 3 (last as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.extend(cursor.item) end
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[i.item]
    end
feature -- Commands
  make do create imp.make ensure model.count = 0 end
  push (g: G) do imp.extend(g)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.finish ; imp.remove
    ensure popped: model ~ (old model.deep_twin).front end
end
```

C3

Testing REL in MATHMODELS

$$r.\text{overridden}(\underbrace{\{(a,3),(c,4)\}}_{t}) = \underbrace{\{(a,3),(c,4)\}}_{t} \cup \underbrace{\{(b,2),(b,5),(d,1),(e,2),(f,3)\}}_{r.\text{domain\_subtracted}(\underbrace{t.\text{domain}}_{\{a,c\}})} = \{(a,3),(c,4),(b,2),(b,5),(d,1),(e,2),(f,3)\}$$

```
test_rel: BOOLEAN
  local
    r, t: REL [STRING, INTEGER]
    ds: SET [STRING]
  do
    create r.make_from_tuple_array (
      <<["a", 1], ["b", 2], ["c", 3],
        ["a", 4], ["b", 5], ["c", 6],
        ["d", 1], ["e", 2], ["f", 3]>>)
    create ds.make_from_array (<<"a">>)
    -- r is not changed by the query 'domain_subtracted'
    t := r.domain_subtracted (ds)
    Result :=
      t /~ r and not t.domain.has ("a") and r.domain.has ("a")
    check Result end
    -- r is changed by the command 'domain_subtract'
    r.domain_subtract (ds)
    Result :=
      t ~ r and not t.domain.has ("a") and not r.domain.has ("a")
  end
```

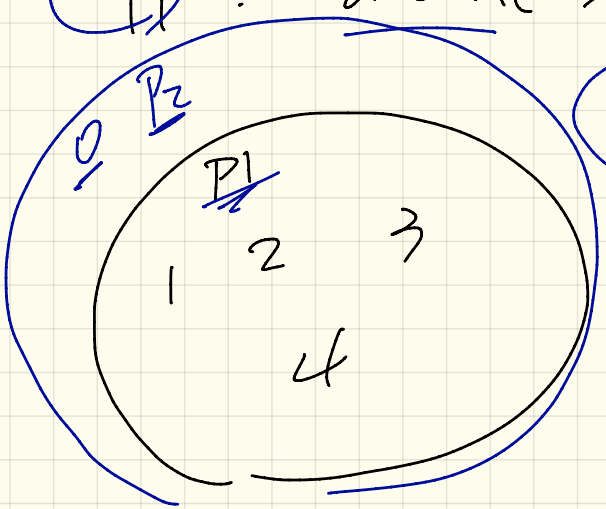
Say $r = \{(a,1),(b,2),(c,3),(a,4),(b,5),(c,6),(d,1),(e,2),(f,3)\}$

- r.***domain***: set of first-elements from *r*
  - r.**domain** = $\{ d \mid (d,r) \in r \}$
  - e.g., r.**domain** = $\{a,b,c,d,e,f\}$
- r.***range***: set of second-elements from *r*
  - r.**range** = $\{ r \mid (d,r) \in r \}$
  - e.g., r.**range** = $\{1,2,3,4,5,6\}$
- r.***inverse***: a relation like *r* except elements are in reverse order
  - r.**inverse** = $\{ (r,d) \mid (d,r) \in r \}$
  - e.g., r.**inverse** = $\{(1,a),(2,b),(3,c),(4,a),(5,b),(6,c),(1,d),(2,e),(3,f)\}$

- r.***domain_restricted***(ds): sub-relation of *r* with domain *ds*.
  - r.**domain_restricted**(ds) = $\{ (d,r) \mid (d,r) \in r \wedge d \in ds \}$
  - e.g., r.**domain_restricted**({a, b}) = $\{(\mathbf{a},1),(\mathbf{b},2),(\mathbf{a},4),(\mathbf{b},5)\}$
- r.***domain_subtracted***(ds): sub-relation of *r* with domain not *ds*.
  - r.**domain_subtracted**(ds) = $\{ (d,r) \mid (d,r) \in r \wedge d \notin ds \}$
  - e.g., r.**domain_subtracted**({a, b}) = $\{(\mathbf{c},6),(\mathbf{d},1),(\mathbf{e},2),(\mathbf{f},3)\}$
- r.***range_restricted***(rs): sub-relation of *r* with range *rs*.
  - r.**range_restricted**(rs) = $\{ (d,r) \mid (d,r) \in r \wedge r \in rs \}$
  - e.g., r.**range_restricted**({1, 2}) = $\{(a,\mathbf{1}),(b,\mathbf{2}),(d,\mathbf{1}),(e,\mathbf{2})\}$
- r.***range_subtracted***(ds): sub-relation of *r* with range not *ds*.
  - r.**range_subtracted**(rs) = $\{ (d,r) \mid (d,r) \in r \wedge r \notin rs \}$
  - e.g., r.**range_subtracted**({1, 2}) = $\{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

P_2 : amount ≥ 0

P_1 : amount > 0



① $p1 \Rightarrow p2$

② $p2 \Rightarrow p1$

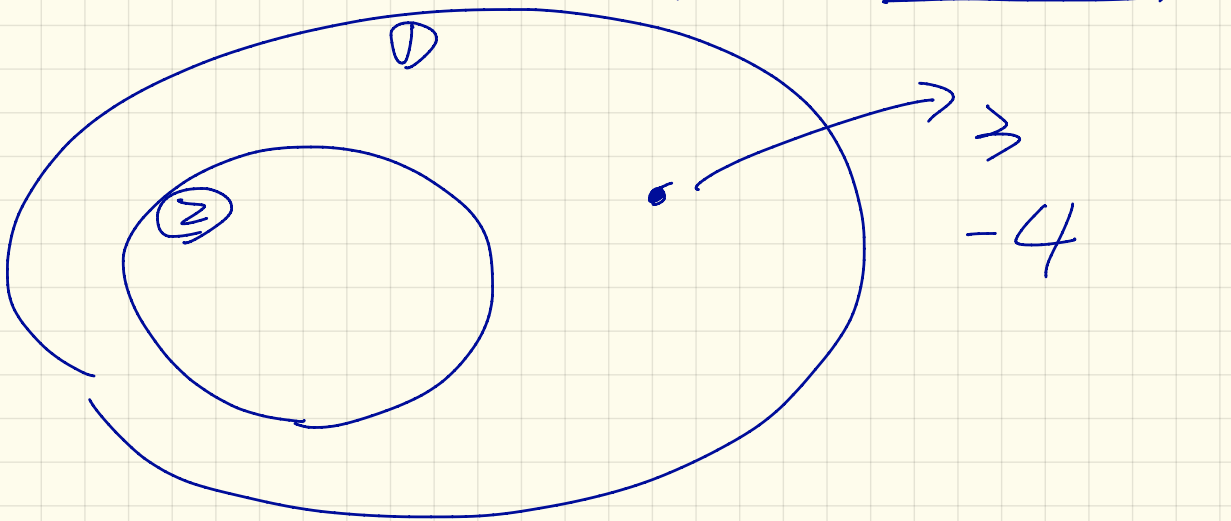
amount = 0 $\quad T \Rightarrow F \quad F$

q (i : INTEGER) : BOOLEAN

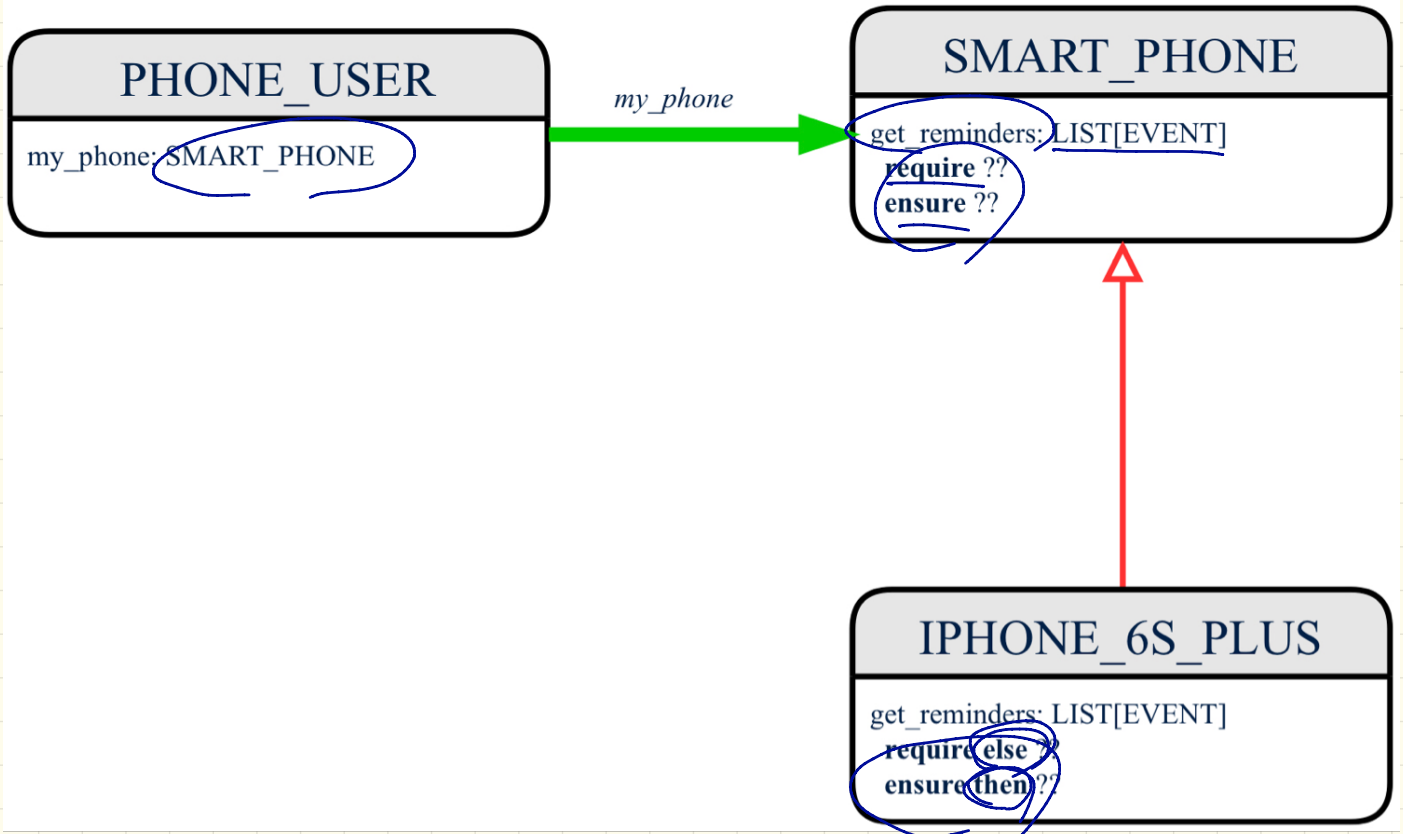
$p \wedge q \Rightarrow p \vee q$

① Result = (i > 0) ∨ (i % 2 = 0)

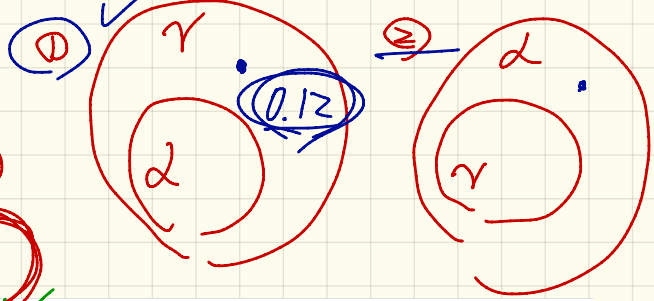
② Result = (i > 0) ∧ (i % 2 = 0)



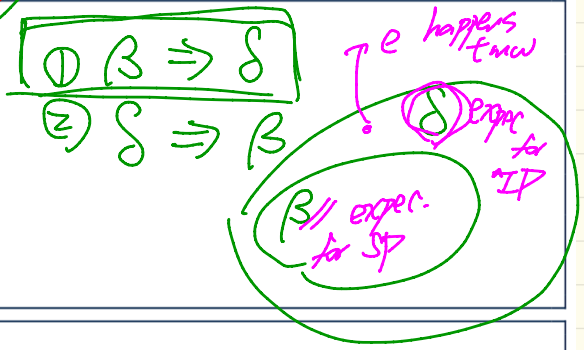
Subcontracting : Architectural View



Subcontracting : Example (1)

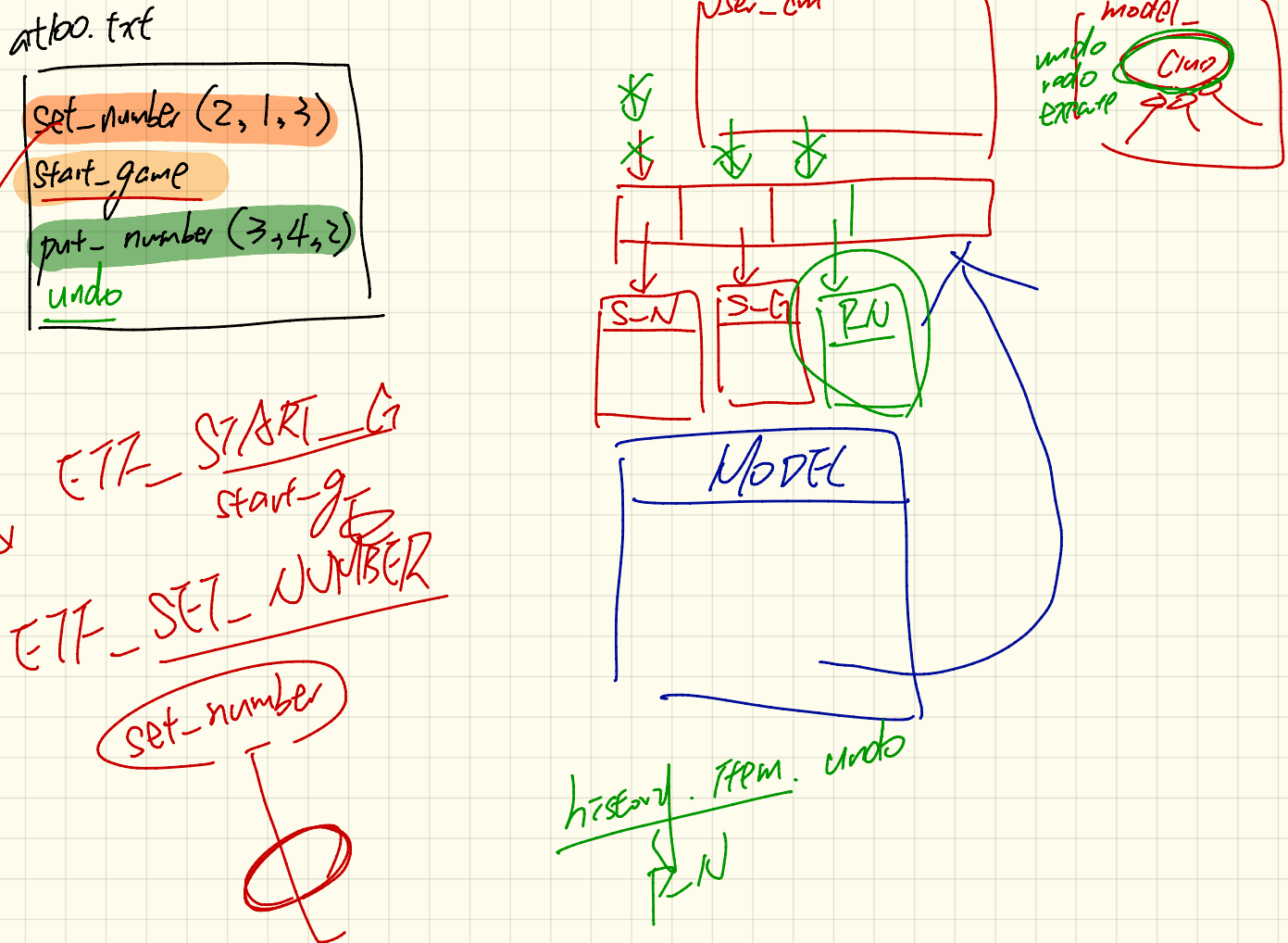


```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

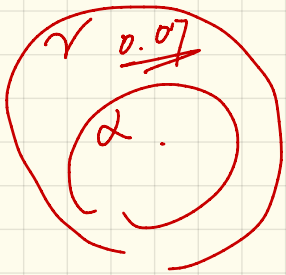


```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e : Result | e happens today or tomorrow
end
```

not appropriate :(it requires more than α


atloo. txt
set_number (2, 1, 3)
start_game
put_ number (3,4,2)
undo
ETF_ START_G
start-g
ETF_ SET_ NUMBER
set-number
User_cm
S_N
S_G
P_N
MODEL
model_
undo
redo
execute
Ctrl
history. ifpm. undo
P_N

Subcontracting : Example (2)



① $\alpha \Rightarrow \gamma$

② $\gamma \Rightarrow \alpha$

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

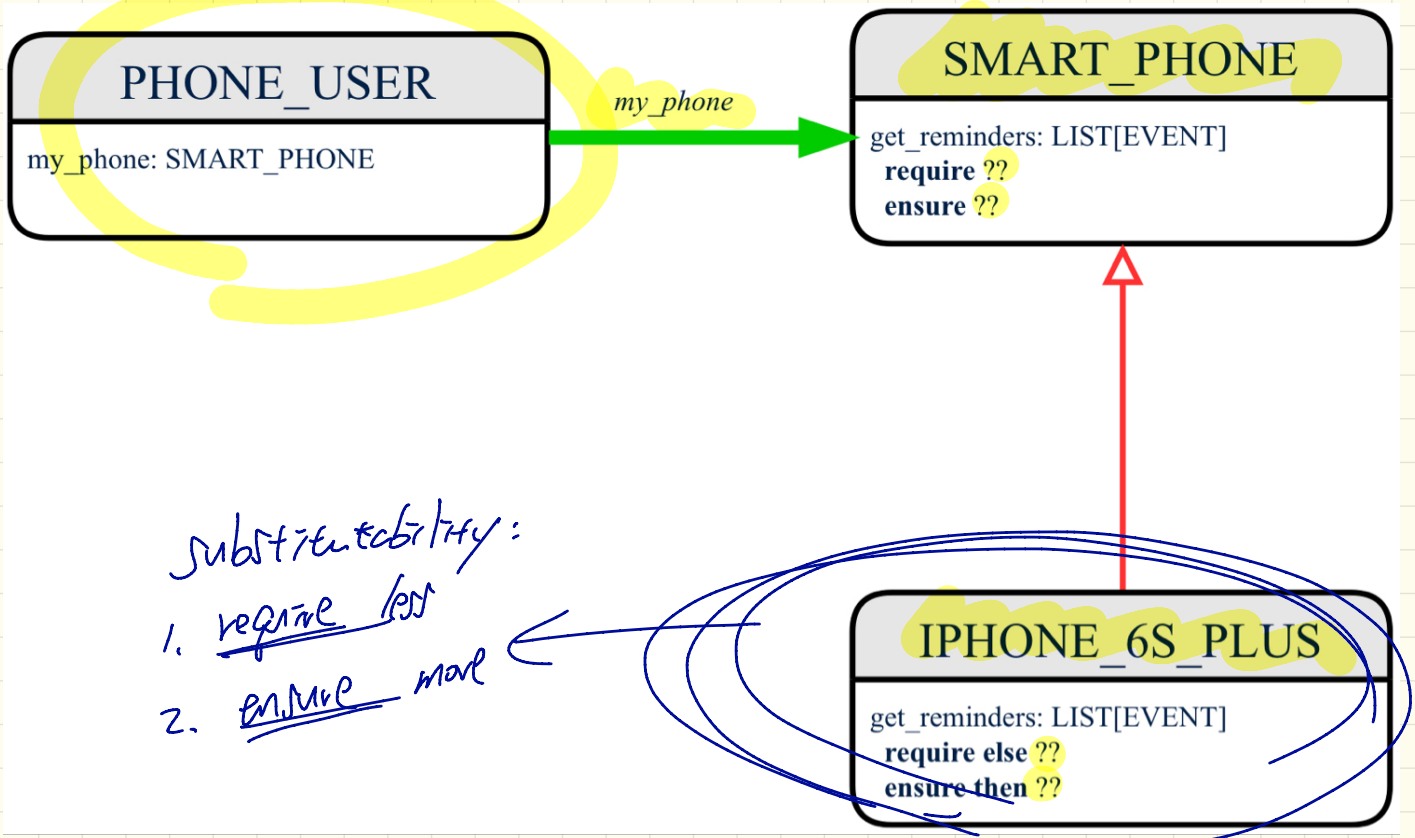
```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.05 -- 5%
    ensure then
      δ: ∀e : Result | e happens today between 9am and 5pm
end
```

Thursday Nov. 15

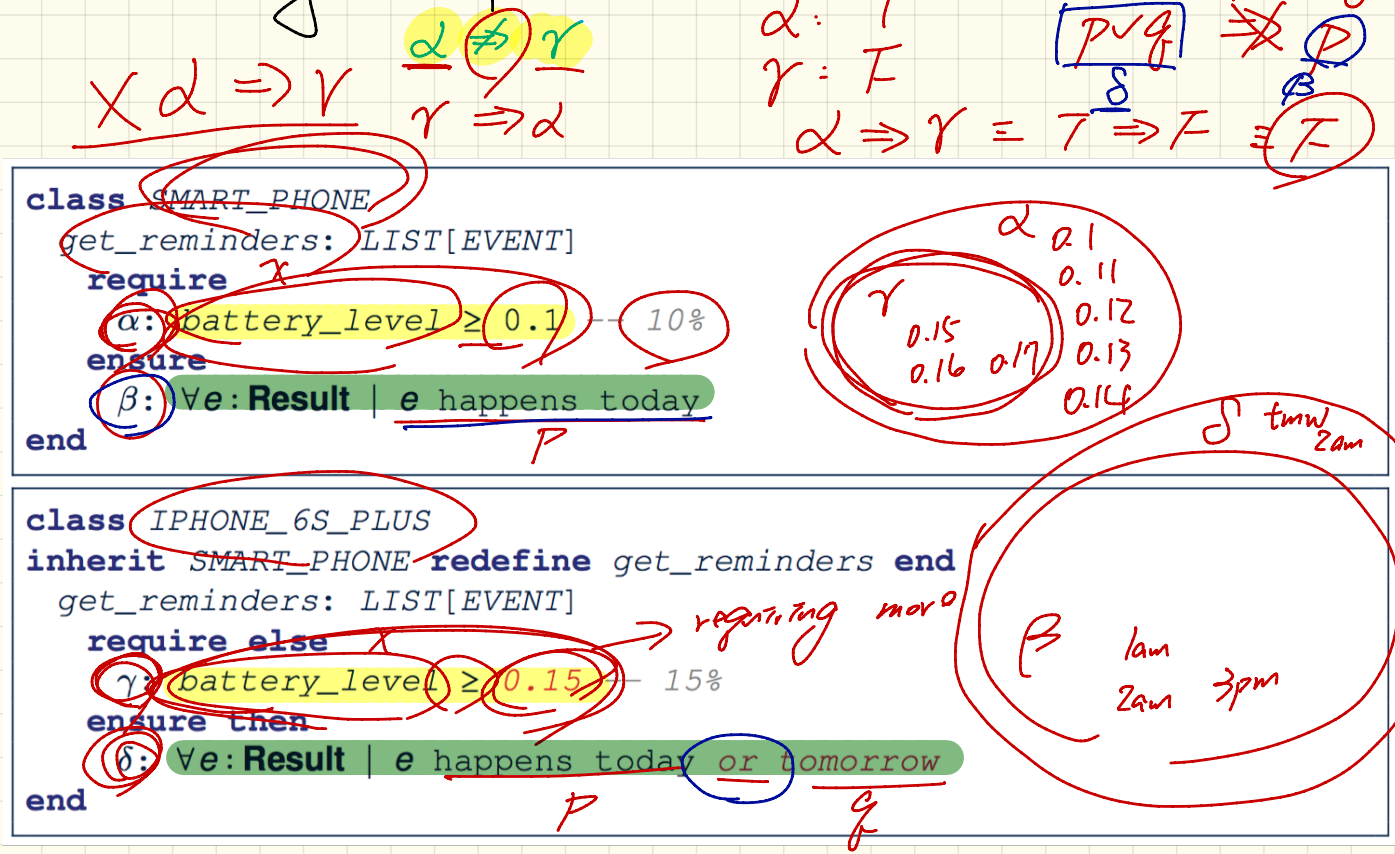
Lecture 19

- Project
- Lab Test Nov. 28
- Lab 5 Next Wed.

Subcontracting : Architectural View



Subcontracting : Example (1)



```
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
```

Subcontracting : Example (2)

appropriate design of preand: $\alpha \overset{?}{\Rightarrow} \gamma$

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

β: 1am 2am 3am 9am – 5pm δ

$\delta \Rightarrow \beta$

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

requires less

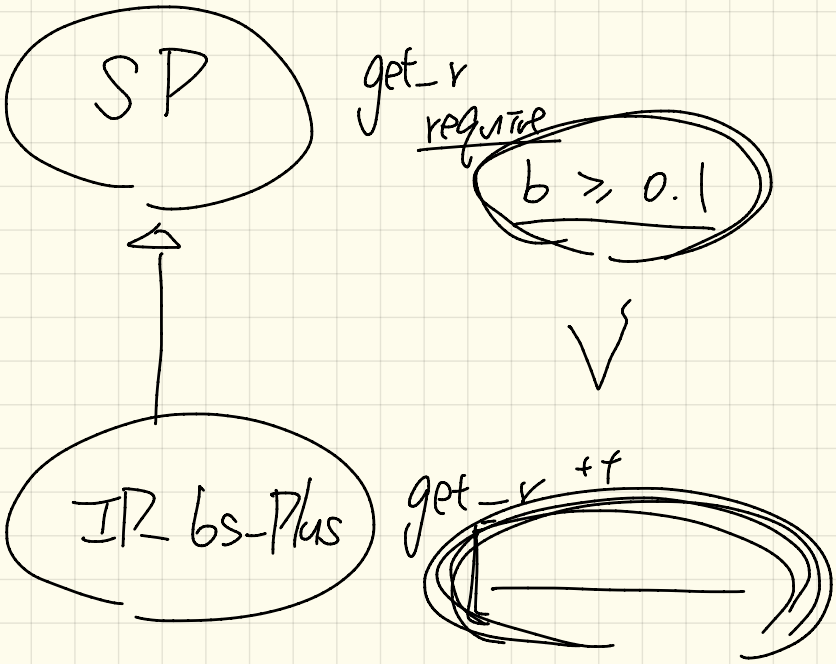
At runtime.

$\alpha \vee \gamma$

$\beta \wedge \delta$

require else false

require else true



① Design Appropriate? Yes ∵ no precondition specified for descendant

② At runtime: $b \geq 0.1 \vee false \equiv b \geq 0.1$

parent_require ∨ child_require ✓

T ≡ T

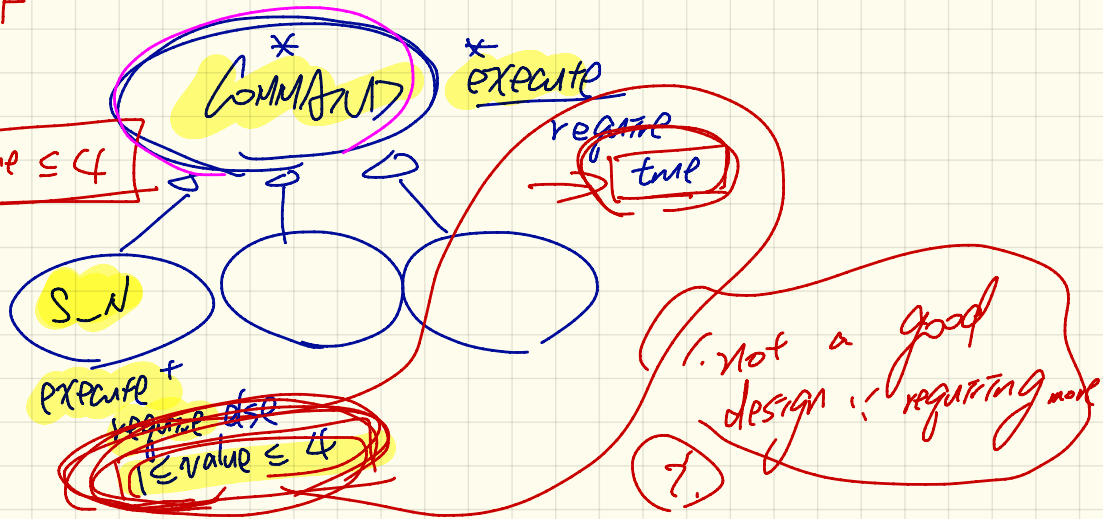
parent_ensure ∧ child_ensure ≡ T

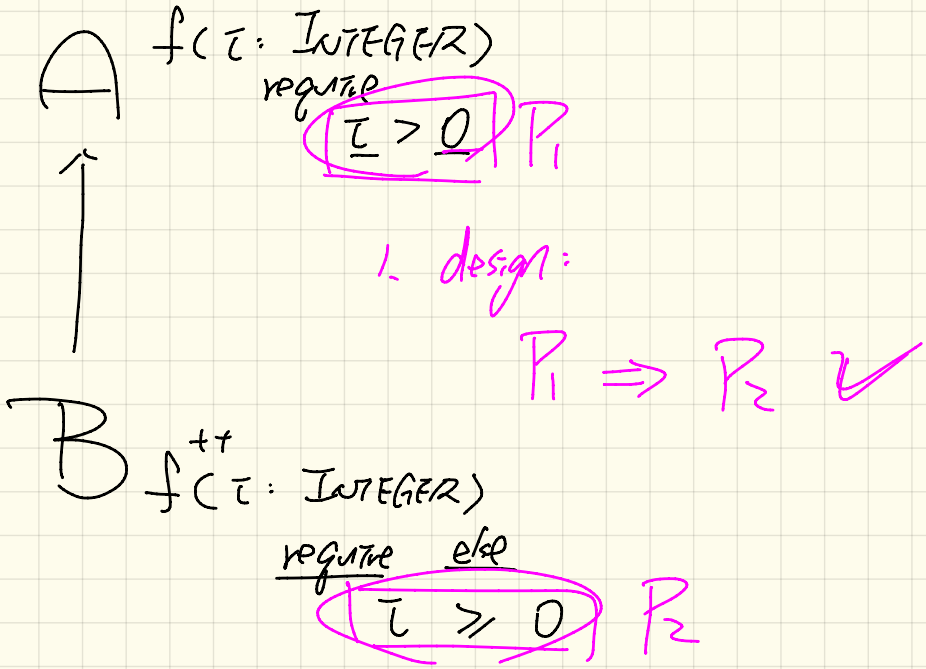
F

7. At runtime:

true ∨ 1 ≤ value ≤ 4

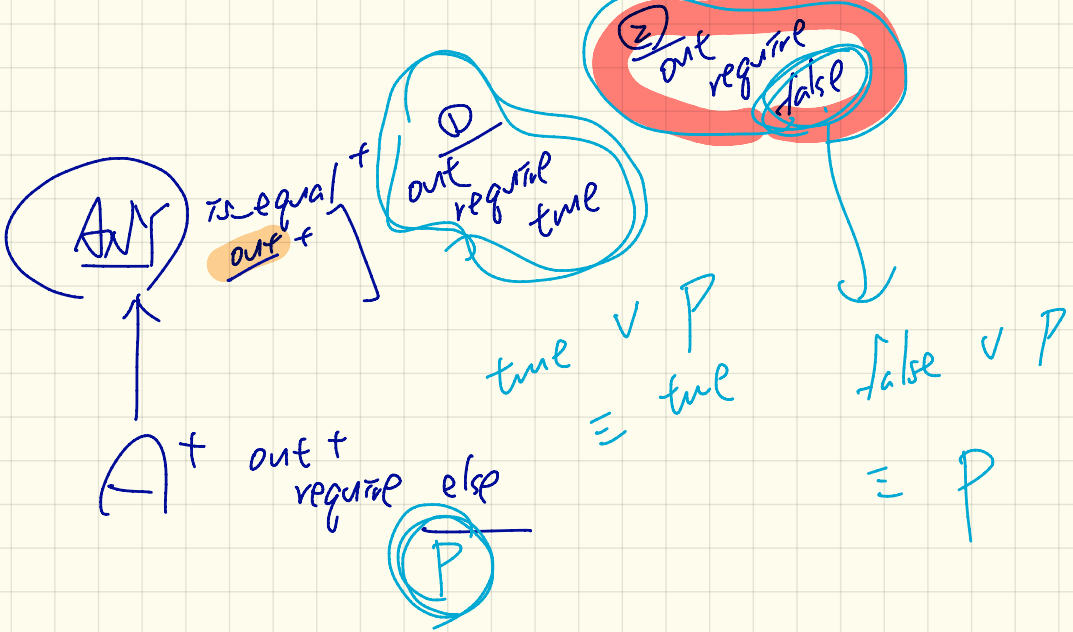
≡ true.



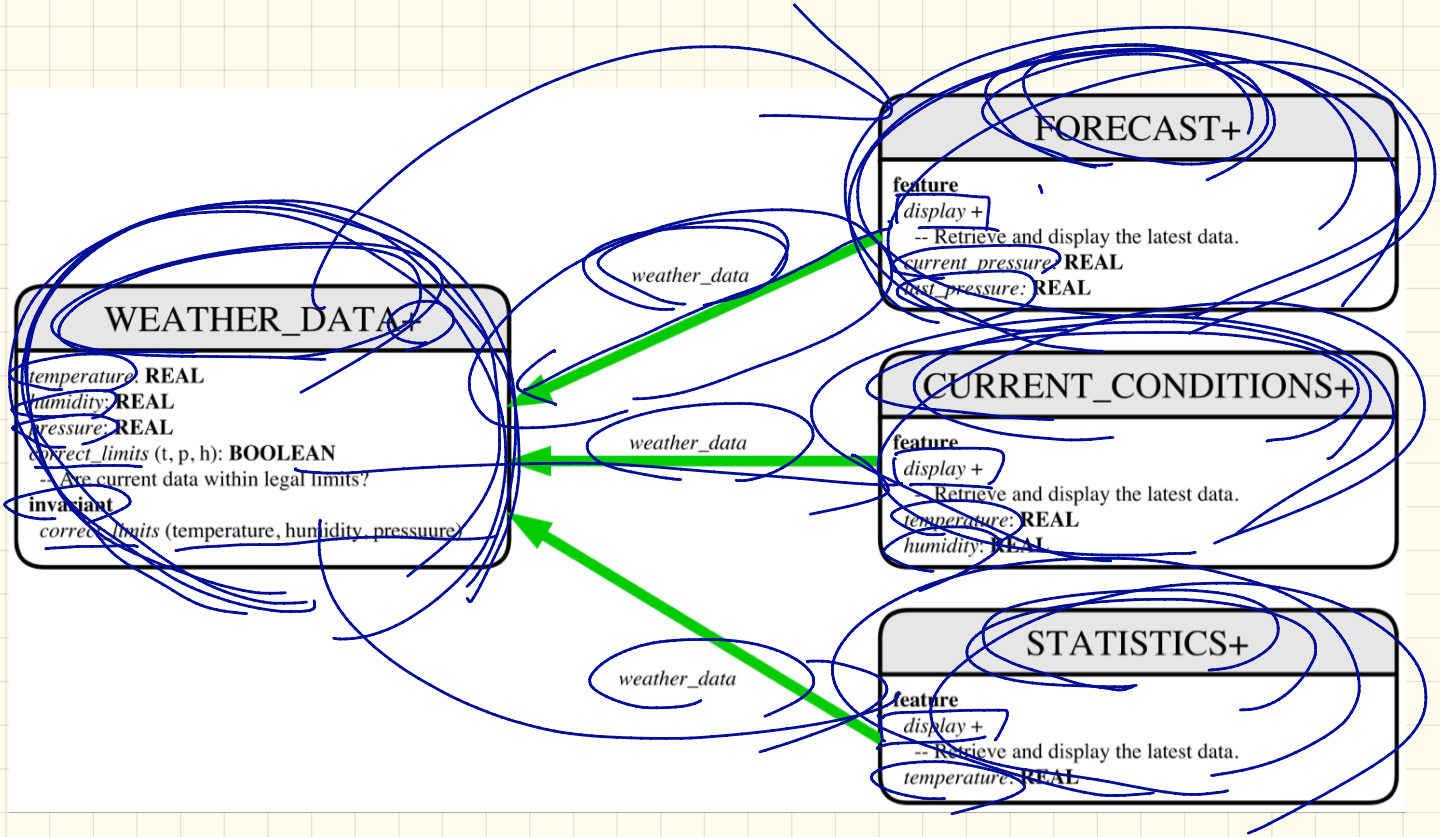


2. At runtime:

$i > 0 \quad \vee \quad i \geq 0 \equiv i \geq 0$


Any
is_equal(t)
out t
①
out
require
true
②
out
require
false
A
t
out t
require else
P
true ∨ P
≡ true
false ∨ P
≡ P

Weather Station : 1st Design



Weather Station: 1st Implementation

```
class WEATHER_DATA create make
feature -- Data
  temperature: REAL
  humidity: REAL
  pressure: REAL
feature -- Queries
  correct_limits(t,p,h: REAL): BOOLEAN
    ensure
      Result implies -36 <=t and t <= 60
      Result implies 50 <= p and p <= 110
      Result implies 0.8 <= h and h <= 100
feature -- Commands
  make (t, p, h: REAL)
    require
      correct_limits(temperature, pressure, humidity)
    ensure
      temperature = t and pressure = p and humidity = h
invariant
  correct_limits(temperature, pressure, humidity)
end
```

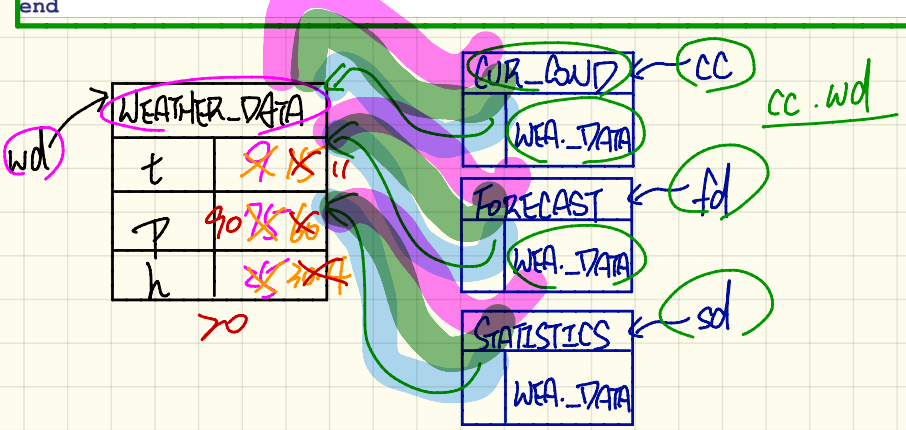
```
class FORECAST create make
feature -- Attributes
  current_pressure: REAL
  last_pressure: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do last_pressure := current_pressure
       current_pressure := weather_data.pressure
    end
  display
    do update
```

```
class CURRENT_CONDITIONS create make
feature -- Attributes
  temperature: REAL
  humidity: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = wd
  update
    do temperature := weather_data.temperature
       humidity := weather_data.humidity
    end
  display
    do update
```

```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature
       -- Update min, max if necessary.
    end
  display
    do update
```

Weather Station: Testing 1st Design

```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd) ; create fd.make (wd) ; create sd.make(wd)

       wd.set_measurements (15, 60, 30.4)
       cc.display ; fd.display ; sd.display
       cc.display ; fd.display ; sd.display

       wd.set_measurements (11, 90, 20)
       cc.display ; fd.display ; sd.display
  end
end
```



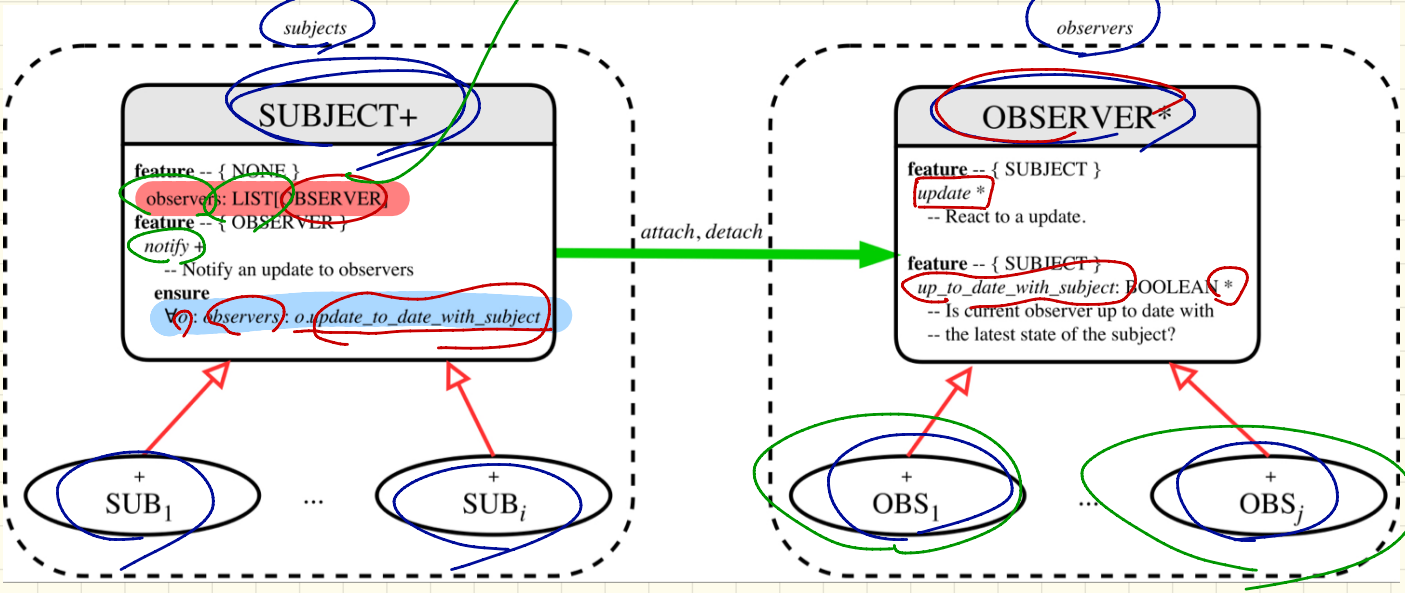

```
class FORECAST create make
feature -- Attributes
  current_pressure: REAL
  last_pressure: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do last_pressure := current_pressure
       current_pressure := weather_data.pressure
    end
  display
    do update
```

```
class CURRENT_CONDITIONS create make
feature -- Attributes
  temperature: REAL
  humidity: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = wd
  update
    do temperature := weather_data.temperature
       humidity := weather_data.humidity
    end
  display
    do update
```

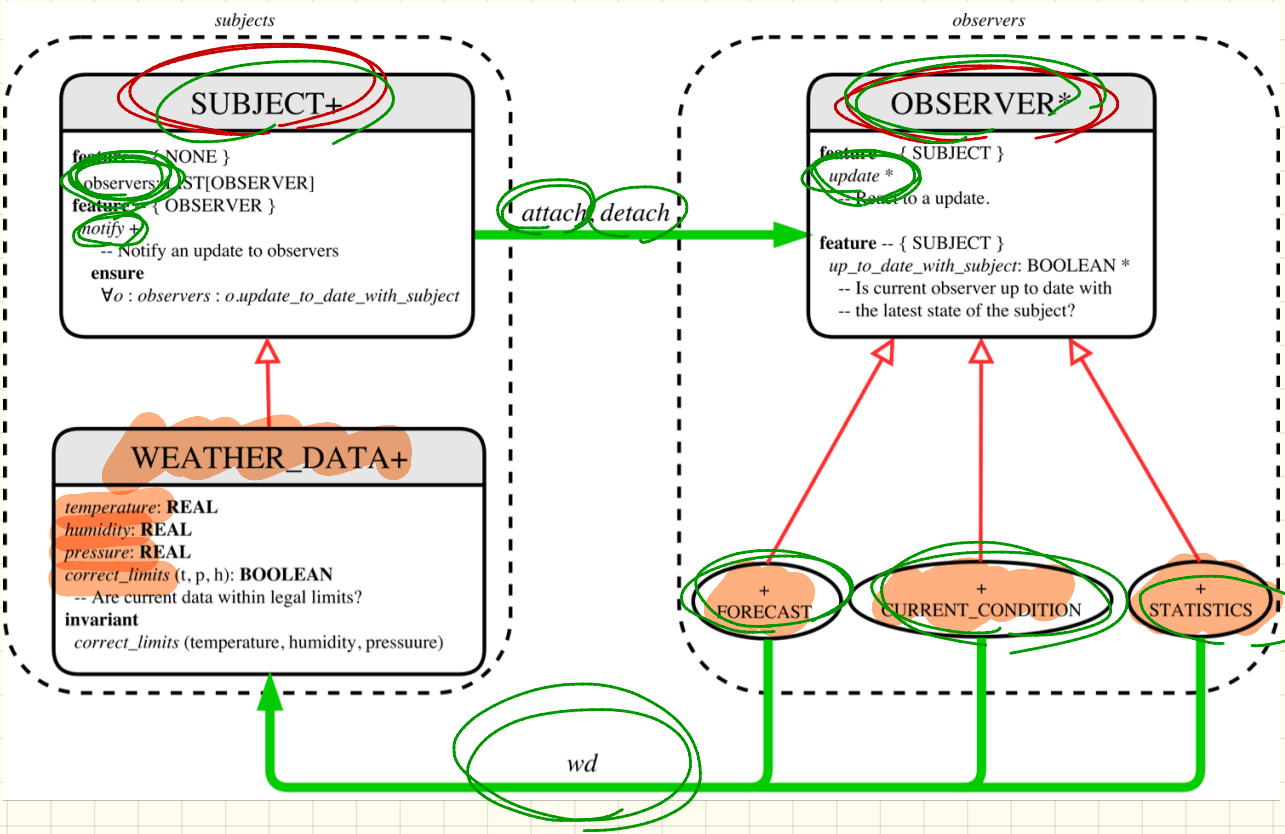
```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature
       -- Update min, max if necessary.
    end
  display
    do update
```

The Observer Pattern

→ Static Type : OBSERVER



Weather Station: Applying the Observer Pattern



Implementing Weather Station : Subject

```
class SUBJECT create make
feature -- Attributes
  observers: LIST[OBSERVER]
feature -- Commands
  make
   do create {LINKED_LIST[OBSERVER]} observers.make
   ensure no_observers:  observers.count = 0 end
feature -- Invoked by an OBSERVER
  attach (o: OBSERVER) -- Add 'o' to the observers
    require not_yet_attached: not observers.has (o)
    ensure is_attached: observers.has (o) end
  detach (o: OBSERVER) -- Add 'o' to the observers
    require currently_attached: observers.has (o)
    ensure is_attached: not observers.has (o) end
feature -- invoked by a SUBJECT
  notify -- Notify each attached observer about the update.
    do across observers as cursor loop cursor.item.update end
    ensure all_views_updated:
      across observers as o all o.item.up_to_date_with_subject end
    end
end
```

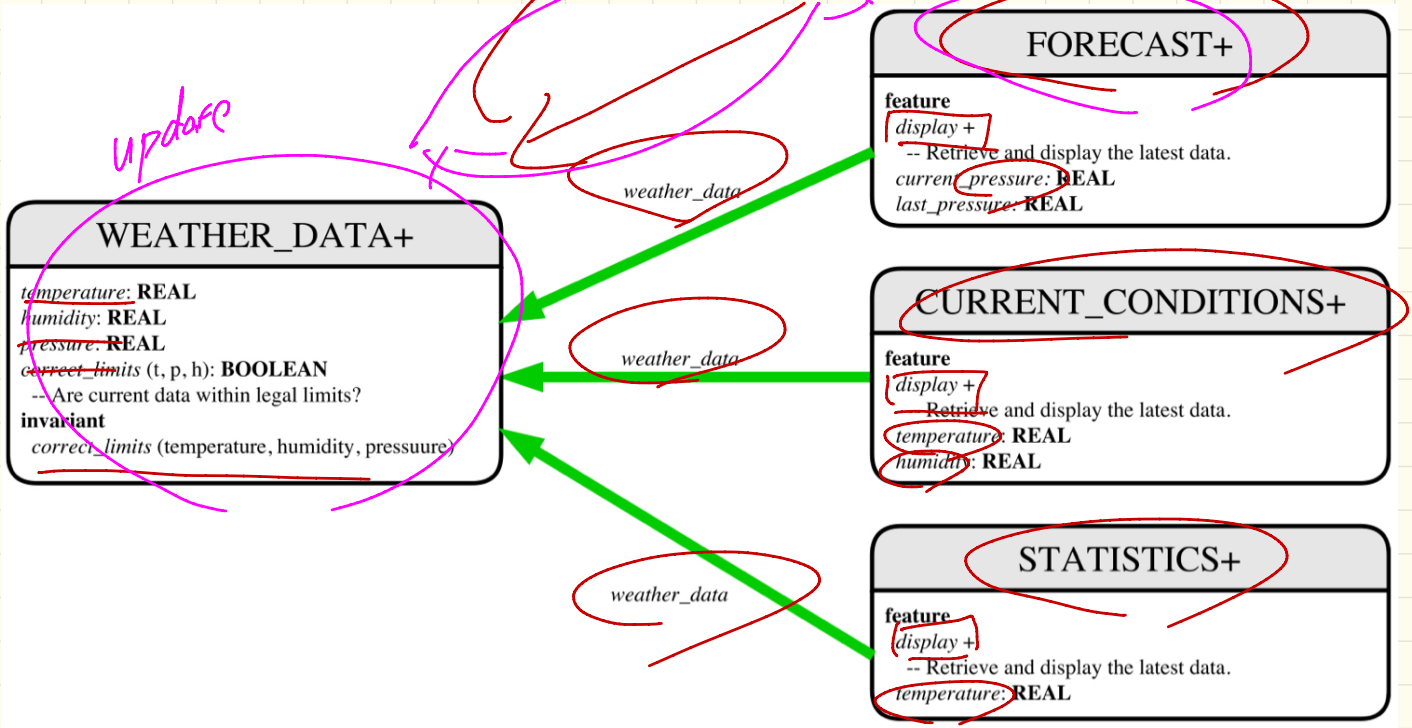
```
class WEATHER_DATA
inherit SUBJECT  rename make as make_subject end
create make
feature -- data available to observers
  temperature: REAL
  humidity: REAL
  pressure: REAL
  correct_limits(t,p,h: REAL): BOOLEAN
feature -- Initialization
  make (t, p, h: REAL)
    do
      make_subject -- initialize empty observers
      set_measurements (t, p, h)
    end
feature -- Called by weather station
  set_measurements(t, p, h: REAL)
    require correct_limits(t,p,h)
invariant
  correct_limits(temperature, pressure, humidity)
end
```

Tuesday Nov. 20

Lecture 20

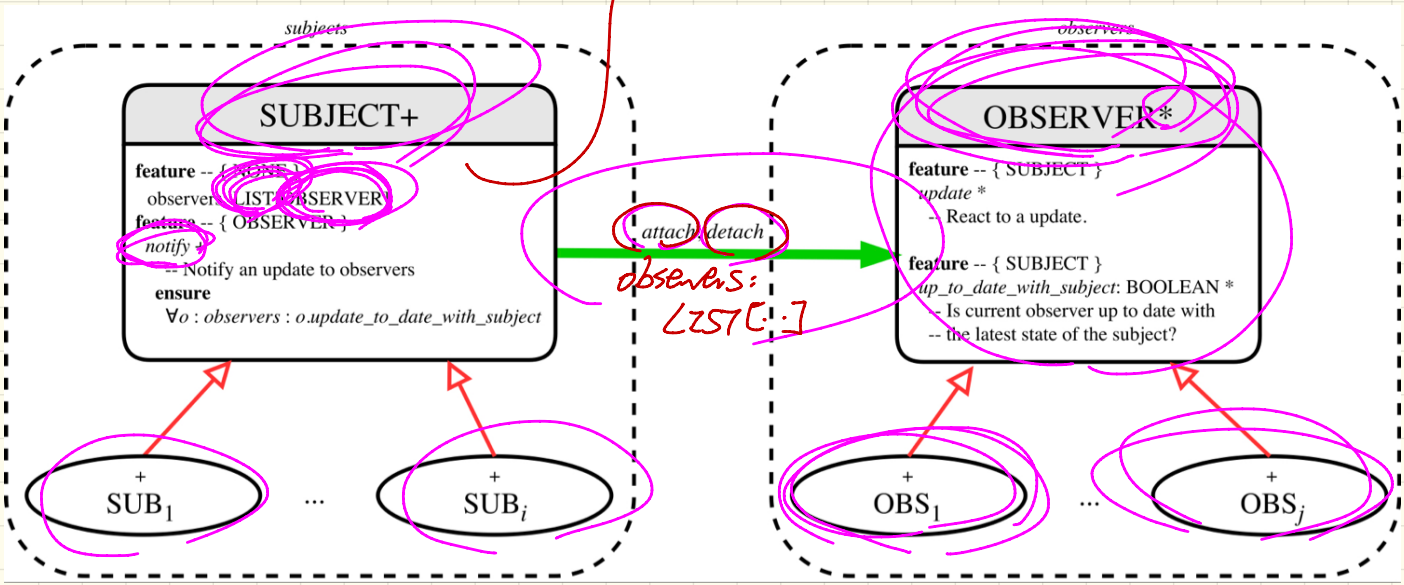
- Lab 5
- Project
- Lab test

Weather Station : 1st Design

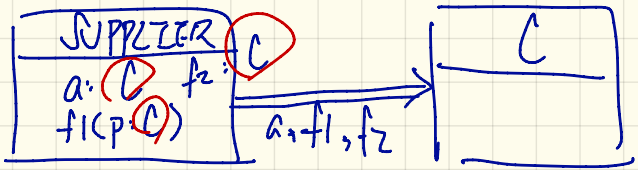


The Observer Pattern

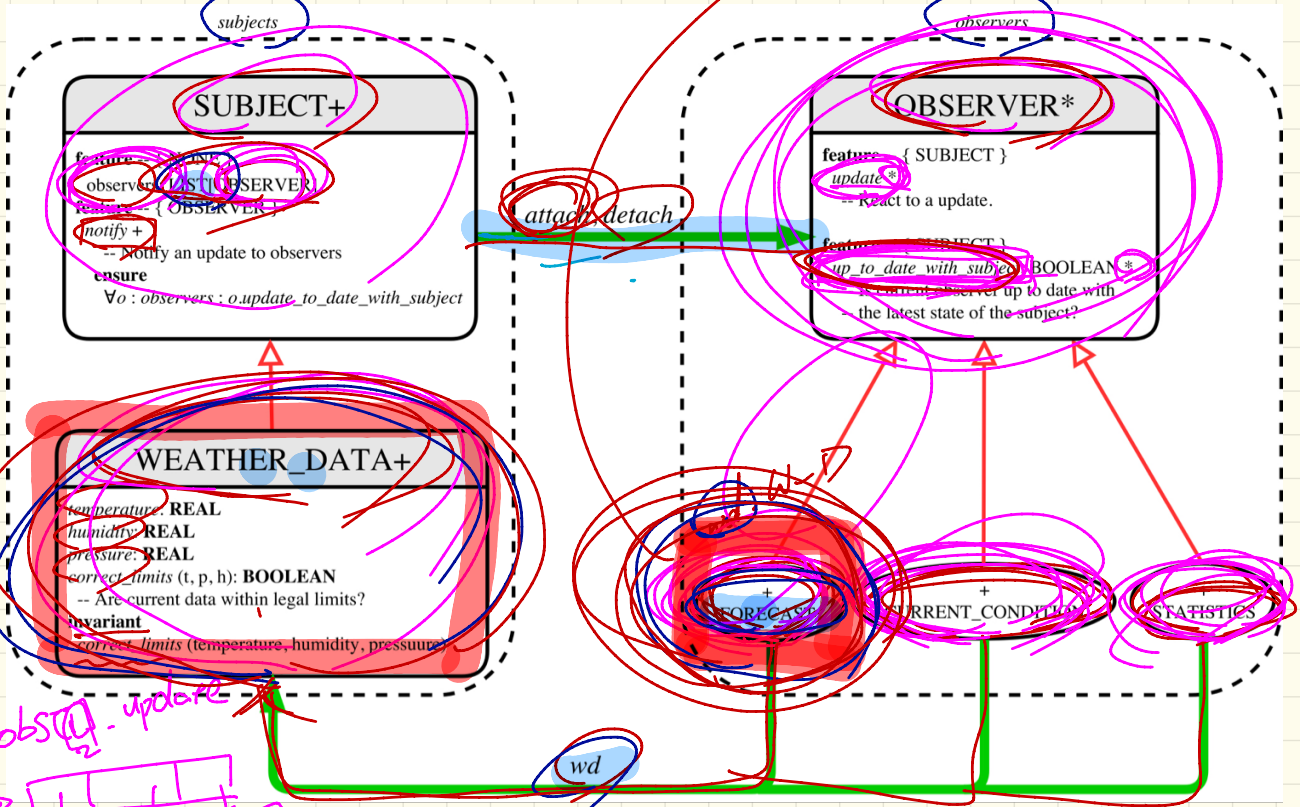
attach(o: OBSERVER)



Client Supplier



Weather Station: Applying the Observer Pattern



Implementing Weather Station: Observers

```
deferred class
  OBSERVER
feature -- To be effected by a descendant
  up_to_date_with_subject: BOOLEAN
      -- Is this observer up to date with its subject?
    deferred
    end

  update
      -- Update the observer's view of 's'
    deferred
    ensure
      up_to_date_with_subject: up_to_date_with_subject
    end
end
```

```
class FORECAST
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
      do weather_data := a_weather_data
         weather_data.attach (Current)
      ensure weather_data = a_weather_data
             weather_data.observers.has (Current)
      end
feature -- Queries
  up_to_date_with_subject: BOOLEAN
    ensure then
      Result = current_pressure = weather_data.pressure
  update
    do -- Same as 1st design; Called only on demand
    end
```

```
class CURRENT_CONDITIONS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
      do weather_data := a_weather_data
         weather_data.attach (Current)
      ensure weather_data = a_weather_data
             weather_data.observers.has (Current)
      end
feature -- Queries
  up_to_date_with_subject: BOOLEAN
    ensure then Result = temperature = weather_data.temperature and
                         humidity = weather_data.humidity
  update
    do -- Same as 1st design; Called only on demand
    end
```

```
class STATISTICS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
      do weather_data := a_weather_data
         weather_data.attach (Current)
      ensure weather_data = a_weather_data
             weather_data.observers.has (Current)
      end
feature -- Queries
  up_to_date_with_subject: BOOLEAN
    ensure then
      Result = current_temperature = weather_data.temperature
  update
    do -- Same as 1st design; Called only on demand
    end
```

Weather Station: Testing the Observer Pattern

```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (0, 75, 25)
       create cc.make (wd) ; create fd.make (wd) ; create sd.make (wd)

       wd.set_measurements (15, 60, 30.4)
       wd.notify
       cc.display ; fd.display ; sd.display
       cc.display ; fd.display ; sd.display

       wd.set_measurements (11, 90, 20)
       wd.notify
       cc.display ; fd.display ; sd.display
  end
end
```

wd.attach(cc)

① wd.os[1].update (cc)

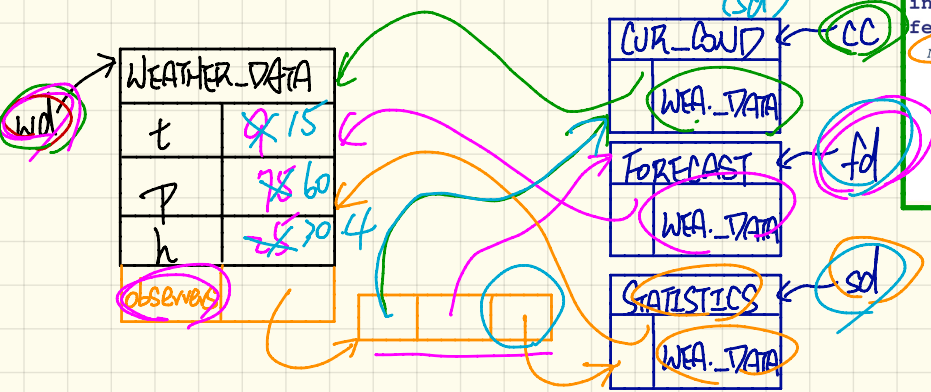
② wd.os[2].update (fd)

③ wd.os[3].update (sd)

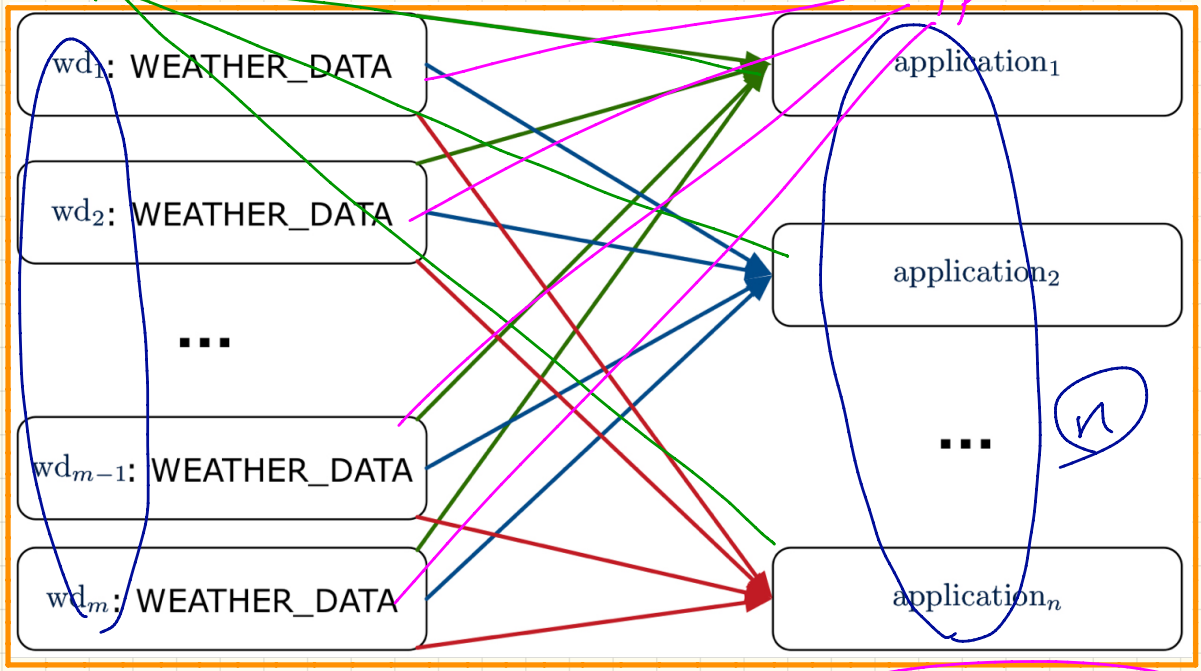
```
class FORECAST
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```

```
class CURRENT_CONDITIONS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```

```
class STATISTICS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```



Observer Pattern: Multiple Subjects and Observers



Complexity ?

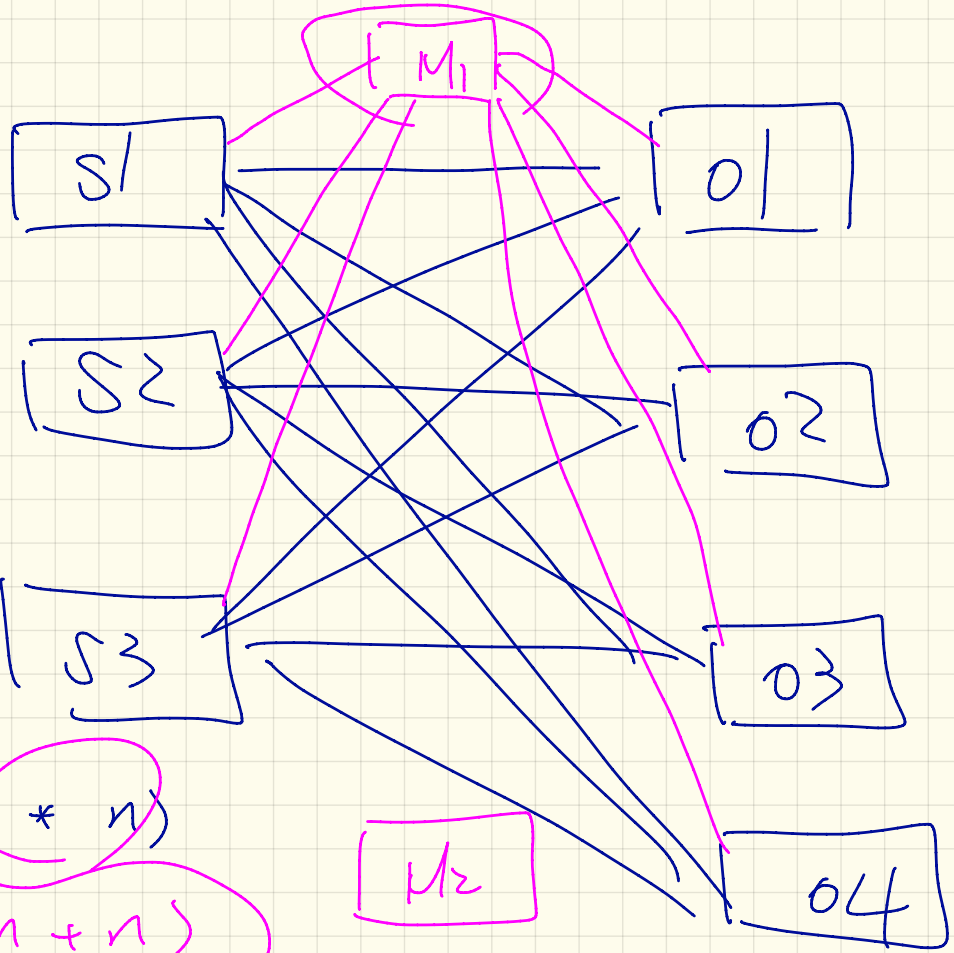
$O(m \cdot n)$

Adding a new subject?

$O(n)$

Adding a new observer?

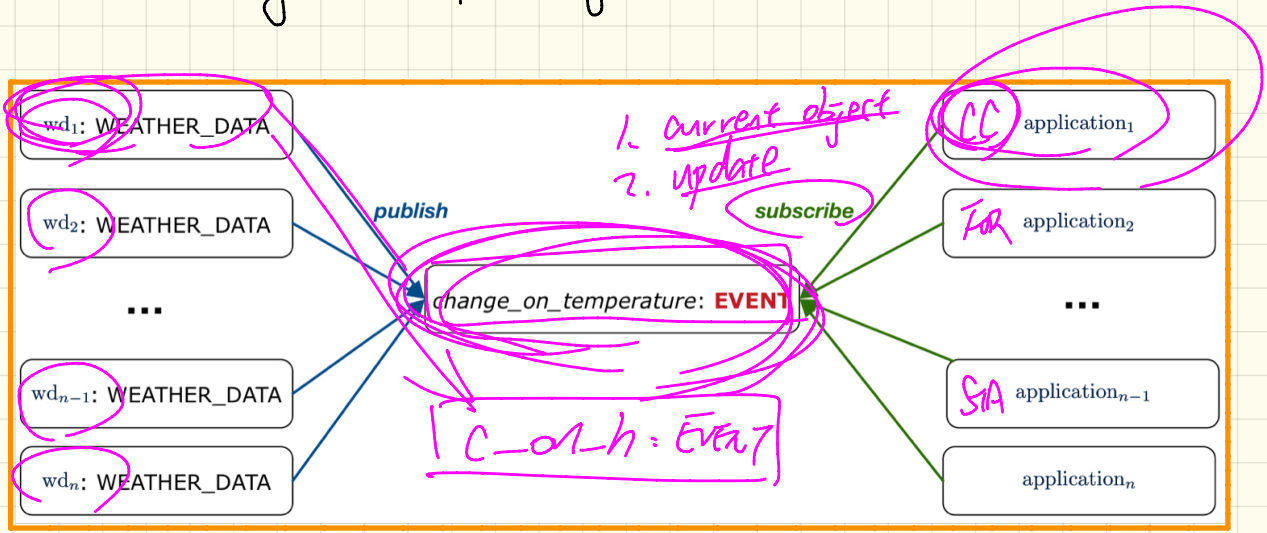
$O(m)$



$O(m * n)$

$O(m + n)$

Event-Driven Design: Multiple Subjects and Observers



Complexity ?

Adding a new subject?

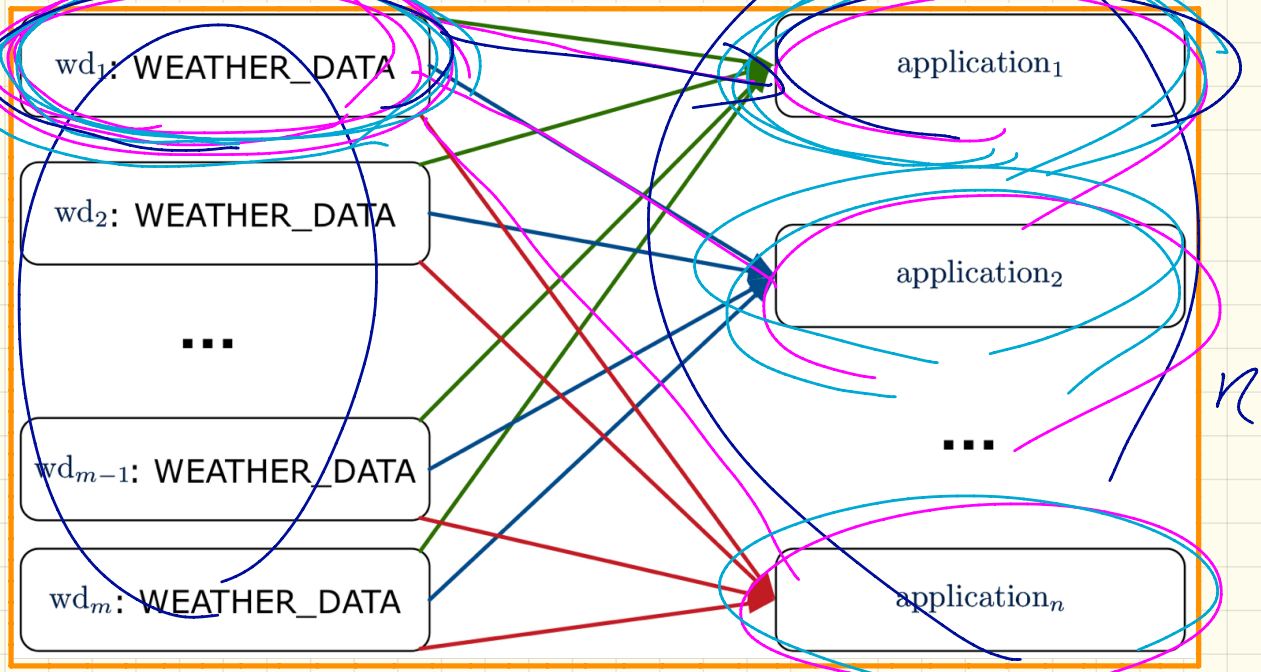
Adding a new observer?

Adding a new event type?

Thursday Nov. 22

Lecture 21

Observer Pattern: Multiple Subjects and Observers



m

n

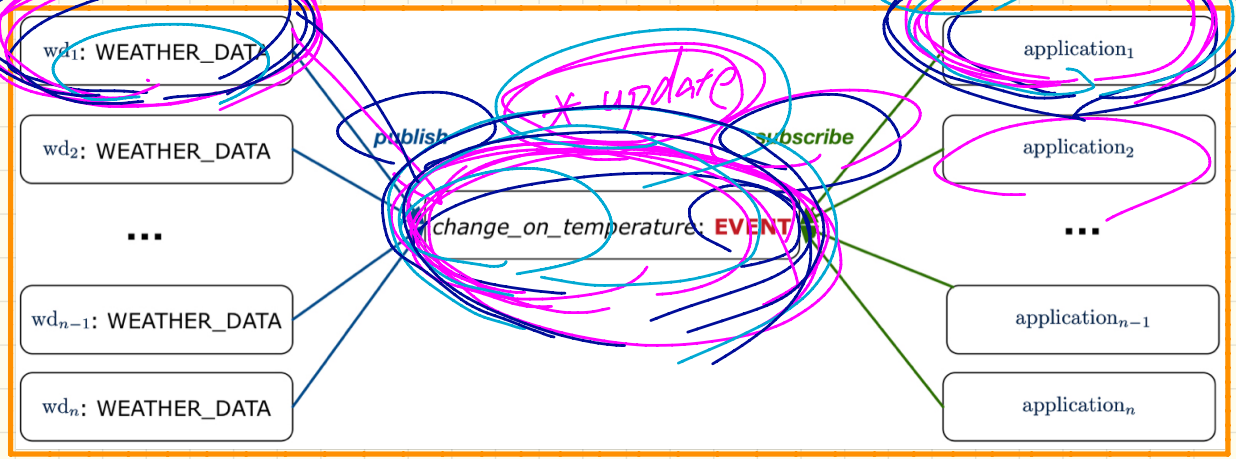
Complexity ?

m x n

Adding a new subject?

Adding a new observer?

Event-Driven Design: Multiple Subjects and Observers



Complexity ?

Adding a new subject?

Adding a new observer?

Adding a new event type?

Event-Driven Design in Java

```
public class WeatherStation {
  public static void main(String[] args) {
    WeatherData wd = new WeatherData(9, 75, 25);
    CurrentConditions cc = new CurrentConditions();
    System.out.println("=======");
    wd.setMeasurements(15, 60, 30.4);
    cc.display();
    System.out.println("=======");
    wd.setMeasurements(11, 90, 20);
    cc.display();
  } }
```

```
public class CurrentConditions {
  private double temperature; private double humidity;
  public void updateTemperature(double t) { temperature = t; }
  public void updateHumidity(double h) { humidity = h; }
  public CurrentConditions() {
    MethodHandles.Lookup lookup = MethodHandles.lookup();
    try {
      MethodHandle ut = lookup.findVirtual(
        this.getClass(), "updateTemperature",
        MethodType.methodType(void.class, double.class));
      WeatherData.changeOnTemperature.subscribe(this, ut);
      MethodHandle uh = lookup.findVirtual(
        this.getClass(), "updateHumidity",
        MethodType.methodType(void.class, double.class));
      WeatherData.changeOnHumidity.subscribe(this, uh);
    } catch (Exception e) { e.printStackTrace(); }
  }
  public void display() {
    System.out.println("Temperature: " + temperature);
    System.out.println("Humidity: " + humidity); } }
```

```
public class WeatherData {
  private double temperature;
  private double pressure;
  private double humidity;
  public WeatherData(double t, double p, double h) {
    setMeasurements(t, h, p);
  }
  public static Event changeOnTemperature = new Event();
  public static Event changeOnHumidity = new Event();
  public static Event changeOnPressure = new Event();
  public void setMeasurements(double t, double h, double p) {
    temperature = t;
    humidity = h;
    pressure = p;
    changeOnTemperature.publish(temperature);
    changeOnHumidity.publish(humidity);
    changeOnPressure.publish(pressure);
  }
}
```

```
public class Event {
  Hashtable<Object, MethodHandle> listenersActions;
  Event() { listenersActions = new Hashtable<>(); }
  void subscribe(Object listener, MethodHandle action) {
    listenersActions.put(listener, action);
  }
  void publish(Object arg) {
    for (Object listener : listenersActions.keySet()) {
      MethodHandle action = listenersActions.get(listener);
      try {
        action.invokeWithArguments(listener, arg);
      } catch (Throwable e) { }
    }
  }
}
```

delayed execution

execute the stored method now.

Event-Driven Design in Eiffel

```
class WEATHER_STATION create make
feature
  cc: CURRENT_CONDITIONS
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd)
       wd.set_measurements (15, 60, 30.4)
       cc.display
       wd.set_measurements (11, 90, 20)
       cc.display
    end
end
```

```
class CURRENT_CONDITIONS
create make
feature -- Initialization
  make(wd: WEATHER_DATA)
    do
      wd.change_on_temperature.subscribe (agent update_temperature)
      wd.change_on_temperature.subscribe (agent update_humidity)
    end
feature
  temperature: REAL
  humidity: REAL
  update_temperature (t: REAL) do temperature := t end
  update_humidity (h: REAL) do humidity := h end
  display do ... end
end
```

```
class WEATHER_DATA
create make
feature -- Measurements
  temperature: REAL ; humidity: REAL ; pressure: REAL
  correct_limits(t,p,h: REAL): BOOLEAN do ... end
  make (t, p, h: REAL) do ... end
feature -- Event for data changes
  change_on_temperature : EVENT[TUPLE[REAL]]once create Result end
  change_on_humidity : EVENT[TUPLE[REAL]]once create Result end
  change_on_pressure : EVENT[TUPLE[REAL]]once create Result end
feature -- Command
  set_measurements (t, p, h: REAL)
    require correct_limits(t,p,h)
    do temperature := t ; pressure := p ; humidity := h
       change_on_temperature.publish ([t])
       change_on_humidity.publish ([p])
       change_on_pressure.publish ([h])
    end
invariant correct_limits(temperature, pressure, humidity) end
```

```
class EVENT [ARGUMENTS -> TUPLE]
create make
feature -- Initialization
  actions: LINKED_LIST[PROCEDURE[ARGUMENTS]]
  make do create actions.make end
feature
  subscribe (an_action: PROCEDURE[ARGUMENTS])
    require action_not_already_subscribed: not actions.ha
    do actions.extend (an_action)
    ensure action_subscribed: action.has(an_action) end
  publish (args: ARGUMENTS)
    do from actions.start until actions.after
       loop actions.item.call (args) ; actions.forth end
    end
end
```

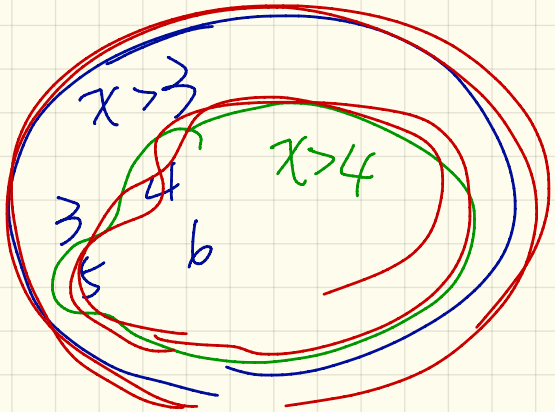
$x > 3$

$x > 4$

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

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

$x > 4 \Rightarrow x > 3$



$p_1 \Rightarrow p_2$

p_1: stronger

p_2: weaker

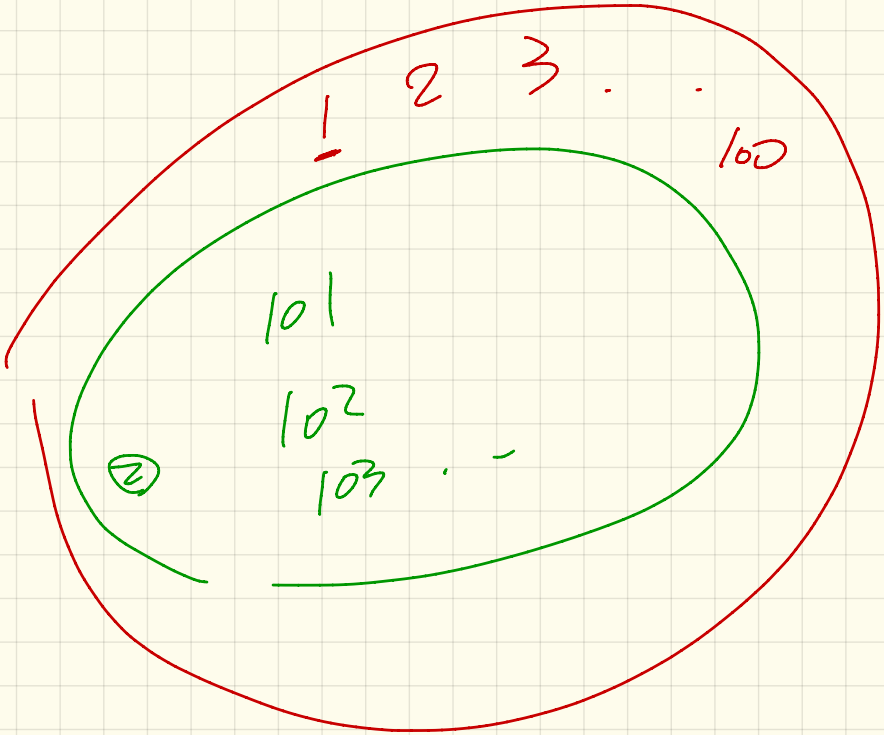
Invariant

weaker → ① balance > 0

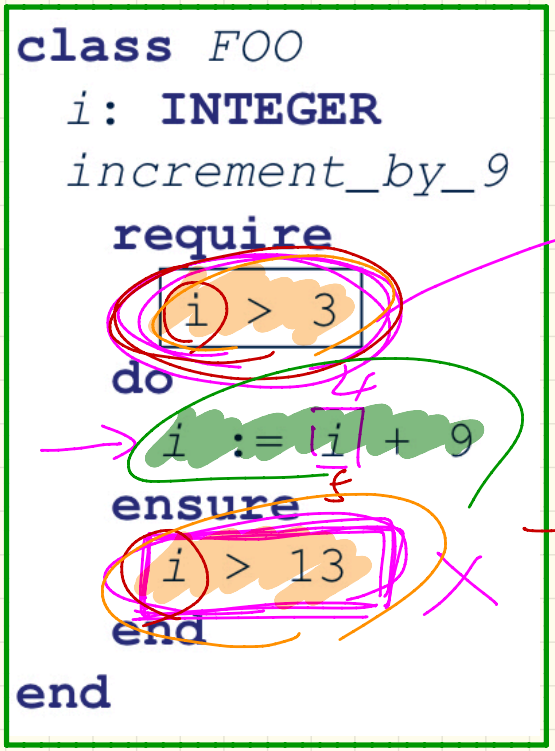
② balance > 100

stronger

② ⇒ ①



Program Correctness : Example (1)



$i = 5$

too weak
↳ $i = 4$

Hoare Triple

predicate

$\{ i > 3 \}$

$i := i + 9$

$\{ i > 13 \}$

Program Correctness : Example (2)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 5
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

may be too strong

e.g. $i = 5$

↳ precond. violation

but $i + 9 = 14 > 13$

$\Rightarrow$ True

i.e.

$\{ i > 5 \}$

$i := i + 9$

$\{ i > 13 \}$

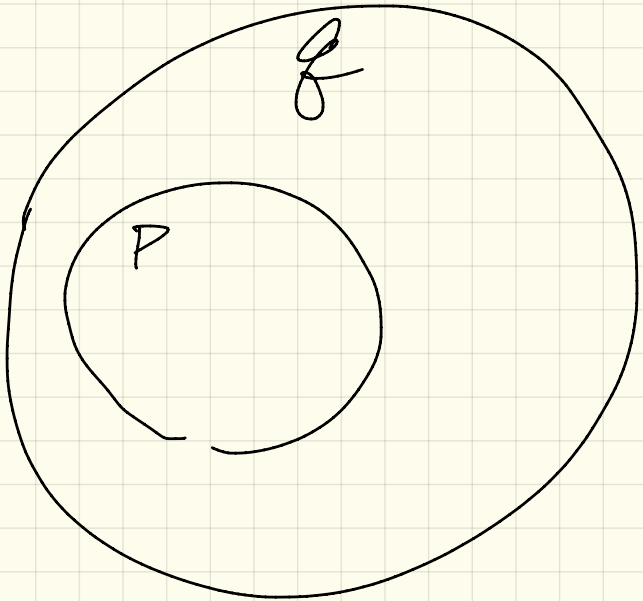
Tuesday Nov. 27

Lecture 22

$P \Rightarrow q$

P ↓ Stronger

q ↓ weaker



Program Correctness : Example (1)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 3
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

Q

S

R

$\{Q\}\ S\ \{R\}$

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

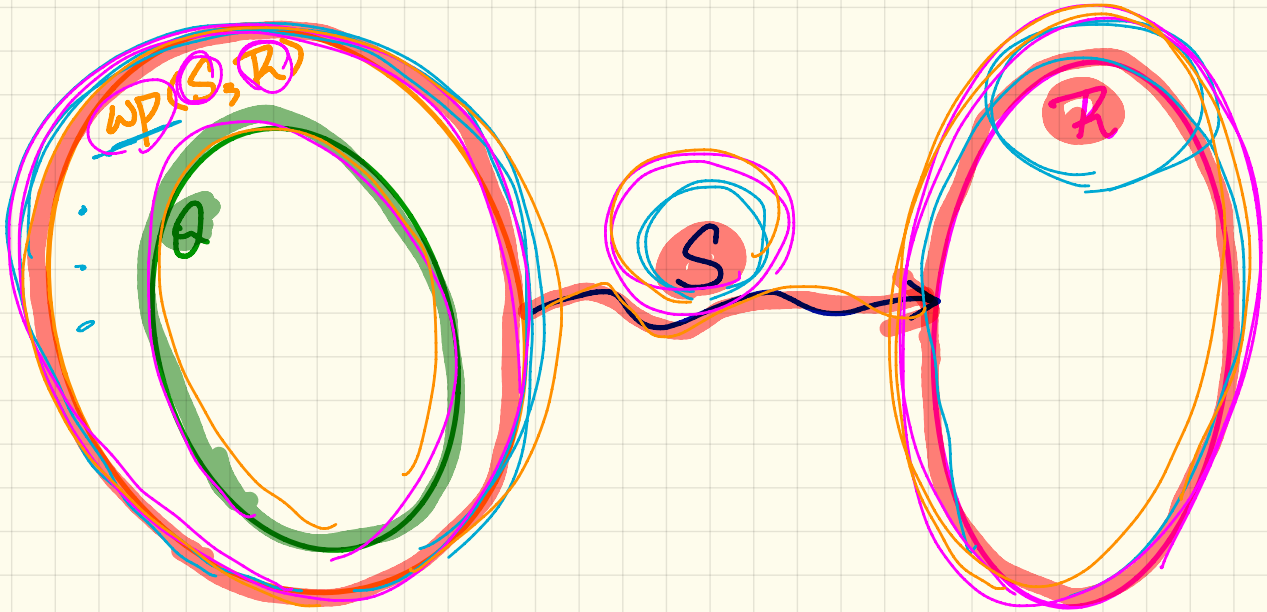
predicate (true or false)

Program Correctness : Example (2)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 5
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

Hoare Triple as a Predicate

$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$



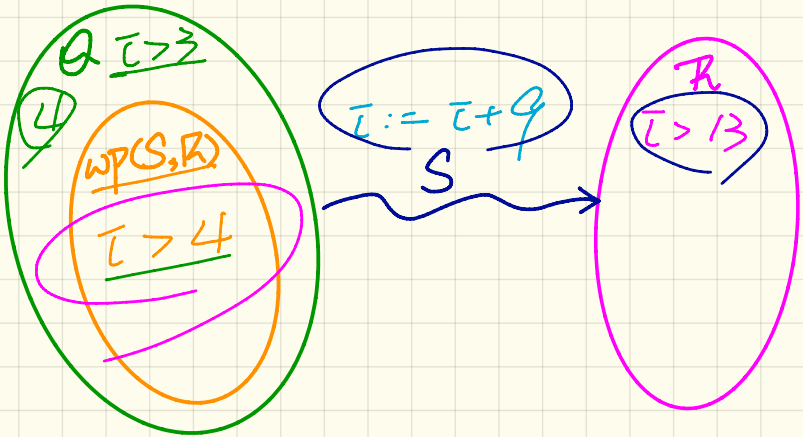
Program Correctness : Example (1)

wp(x := x + 1 , $\sum_{x=1}^{n} x + x > 10$)

bounded — free — x+1

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 3
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$



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

= { rule of wp for := }

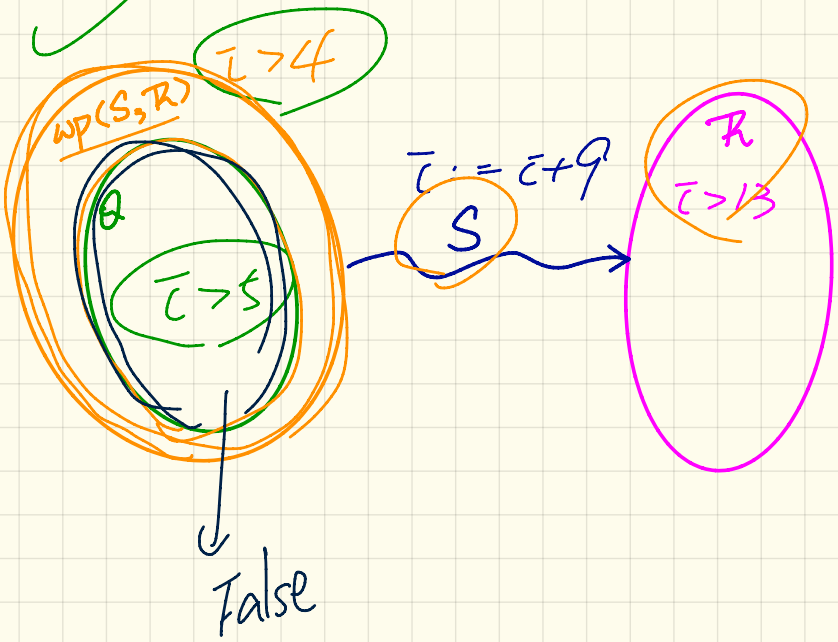
i+9 > 13 [i := i+9] = i+9 > 13

= i > 4

Program Correctness : Example (2)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 5
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$



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

$= \{ wp \text{ rule for } := \}$

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

post pre post

$= x_0 + 1 > x_0$

$= 1 > 0$

$= \text{True}$

1. To be correct, choose any Q stronger than True.
   e.g. $x > 0$, $x = 0$, $x < 0$, False
2. To be appropriate design, up to the agreement between UC and S.

$wp(x := x+1,\ x = 22)$

$= [x = 22]$

$x \neq 22$

22 23

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

→ [B S_1] $B \Rightarrow wp(S_1, R)$

$\neg B$ S_2 $\neg B \Rightarrow wp(S_2, R)$

Rule of wp: Conditionals

$wp(\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ end}, R)$

$(B \Rightarrow wp(S_1, R)) \vee (\neg B \Rightarrow wp(S_2, R))$

vs.

$(B \Rightarrow wp(S_1, R)) \wedge (\neg B \Rightarrow wp(S_2, R))$

??

choose $x = -1$

$x \geq -1$

$\vee$

Consider: $x \geq 1$

$\equiv x \geq -1$

$x \geq -1$

$\wedge$

$x \geq 1$

$\equiv x \geq 1$

$wp(\text{if } y > 0 \text{ then } x := x + 1 \text{ else } x := x - 1 \text{ end}, x \geq 0)$

$y > 0 \Rightarrow wp(x := x + 1, x \geq 0)$ $\quad x \geq -1$

$y \leq 0 \Rightarrow wp(x := x - 1, x \geq 0)$ $\quad x \geq 1$

{Q}

if B_1 then

S_1

elseif B_2 then

S_2

else

S_3

end

{R}

$B_1 \Rightarrow wp(S_1, R)$

$\wedge$

$\neg B_1 \wedge B_2 \Rightarrow wp(S_2, R)$

$\wedge$

$\neg(B_1 \vee B_2) \Rightarrow wp(S_3, R)$

$\{Q\}$ if B then S_1 else S_2 end $\{R\}$

$\equiv$

$Q \Rightarrow \left(\begin{array}{l} \neg B \Rightarrow wp(S_1, R) \\ \wedge \\ \neg B \Rightarrow wp(S_2, R) \end{array} \right)$

shunting

$p \Rightarrow (q \Rightarrow r)$

$\equiv$

$(p \wedge q) \Rightarrow r$

Correctness of Program: Conditionals

Is this program correct?

```
{x > 0 ∧ y > 0}
if x > y then
  bigger := x ; smaller := y
else
  bigger := y ; smaller := x
end
{bigger ≥ smaller}
```

1. Calculate the wp

wp (if $x > y$ then b := x ; s := y else b := y ; s := x end , $b \geq s$)

= $x > y \Rightarrow$ wp([ ] , $b \geq s$)

$\neg(x > y) \Rightarrow$ wp([ ] , $b \geq s$)

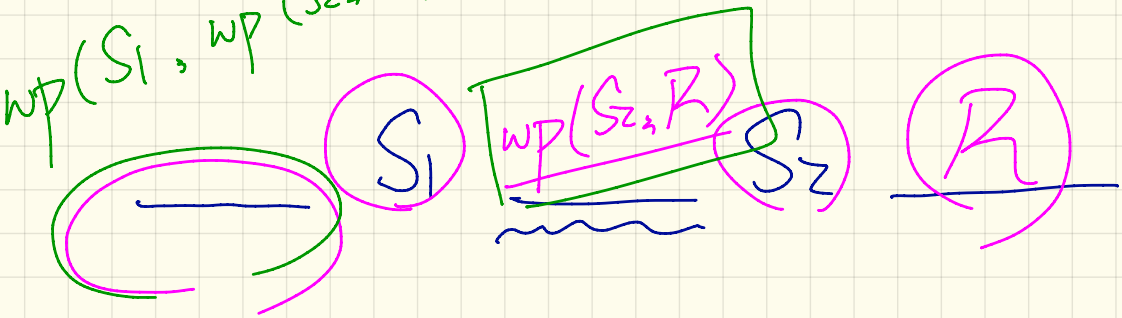
2. Prove or disprove

$x > 0 \wedge y > 0 \Rightarrow$ wp

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

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

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



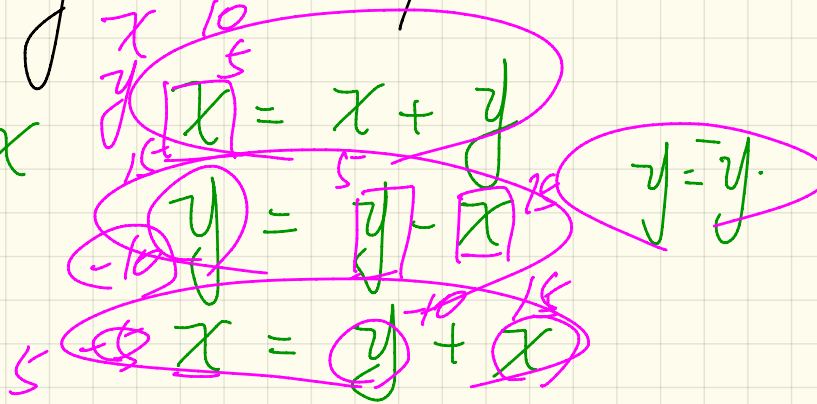
Correctness of Program : Sequental Composition

Is { ***True*** } `tmp := x; x := y; y := tmp` { $x > y$ } correct?

Q: Swap x and y without using a temp variable.

$x = x + y - x$

$y =$



Thursday Nov. 29

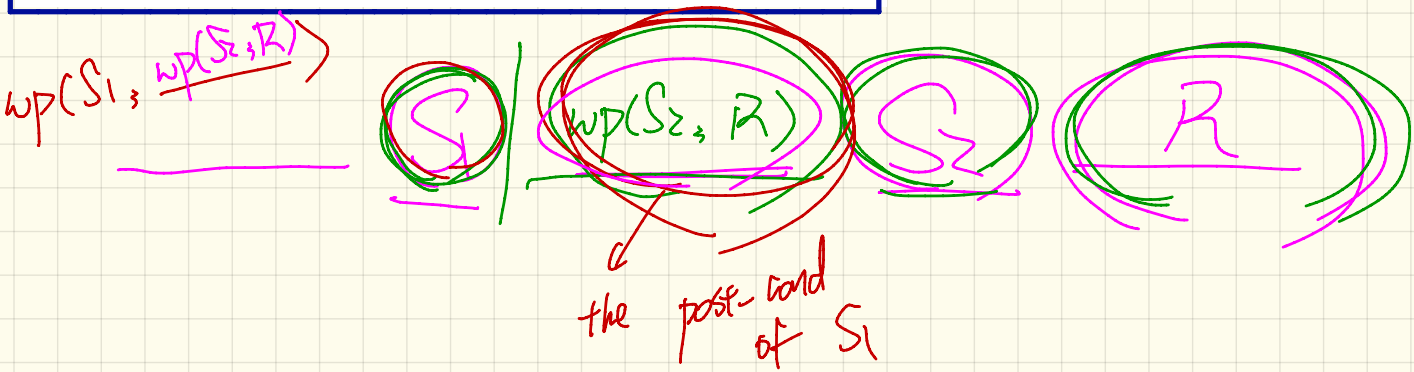
Lecture 23

wp Rules

$$wp(x := e, R) = R[x := e]$$

$$wp(\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ end}, R) = \left(\begin{array}{l} B \Rightarrow wp(S_1, R) \\ \wedge \\ \neg B \Rightarrow wp(S_2, R) \end{array} \right)$$

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



Proof Rules

$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$

$$\{Q\}\ \texttt{x := e}\ \{R\} \iff Q \Rightarrow \underbrace{R[x := e]}_{wp(\texttt{x := e}, R)}$$

$$\{Q\}\ \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{end}\ \{R\}$$
$$\iff \begin{pmatrix} \{\ Q \wedge B\ \}\ S_1\ \{\ R\ \} \\ \wedge \\ \{\ Q \wedge \neg B\ \}\ S_2\ \{\ R\ \} \end{pmatrix} \iff \begin{pmatrix} (Q \wedge B) \Rightarrow wp(S_1, R) \\ \wedge \\ (Q \wedge \neg B) \Rightarrow wp(S_2, R) \end{pmatrix}$$

$$\{Q\}\ S_1\ ;\ S_2\ \{R\} \iff Q \Rightarrow \underbrace{wp(S_1, wp(S_2, R))}_{wp(S_1\ ;\ S_2, R)}$$

Correctness of Program : Sequental Composition

Step 2: Q: True ⇒ y > x
↳ No e.g. y := 1, x := 2

Is {True} tmp := x; x := y; y := tmp {x > y} correct?

Goal: True ⇒ wp(tmp := x; x := y; y := tmp; x > y)

Step 1:

wp(tmp := x; x := y; y := tmp; x > y)

= { wp rule for ; }

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

= { wp rule for ; }

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

= { wp for := }

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

= { wp for := }

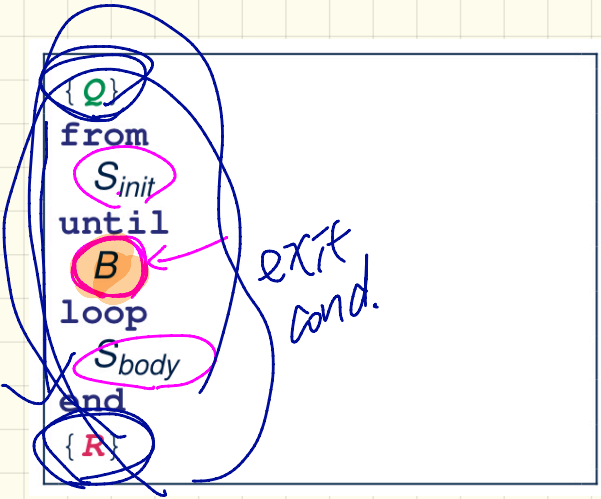
wp(tmp := x, y > tmp) = { wp for := } = y > x

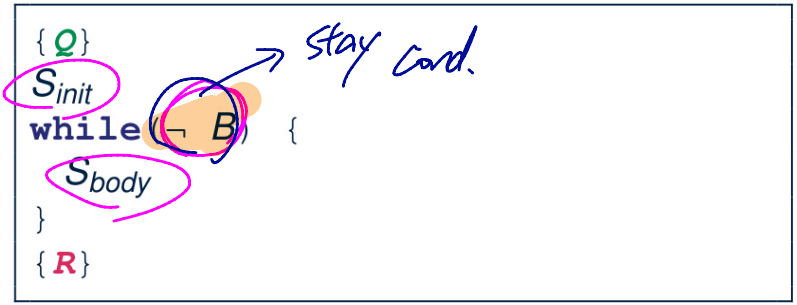
$$x > 4 \Rightarrow x > 3$$

it is obvious that

$$x > 4 \Rightarrow x > 3$$

Loops: Eiffel vs. Java



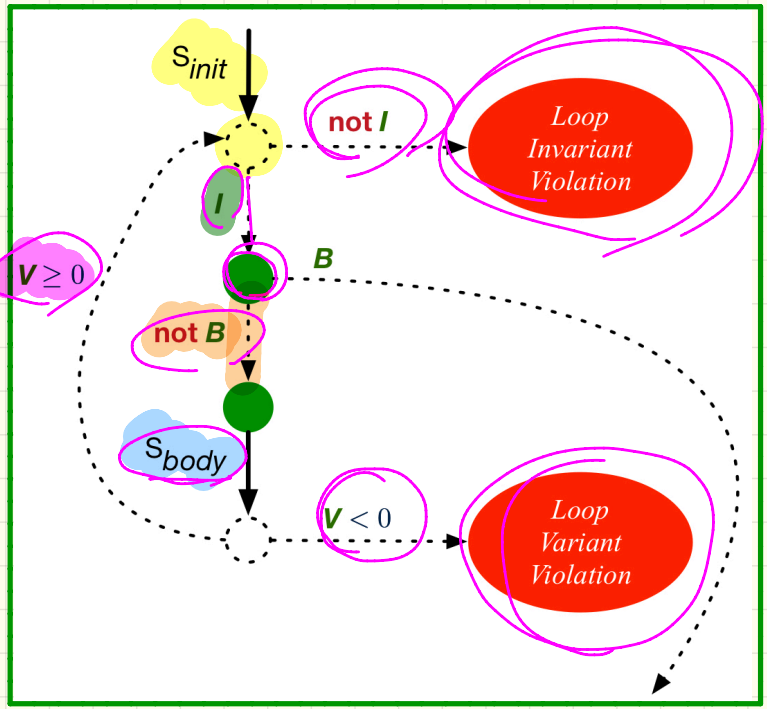


Contracts of Loops

Syntax

```
from
  Sinit
invariant
  invariant_tag: I
until
  B
loop
  Sbody
variant
  variant_tag: V
end
```

Runtime Checks

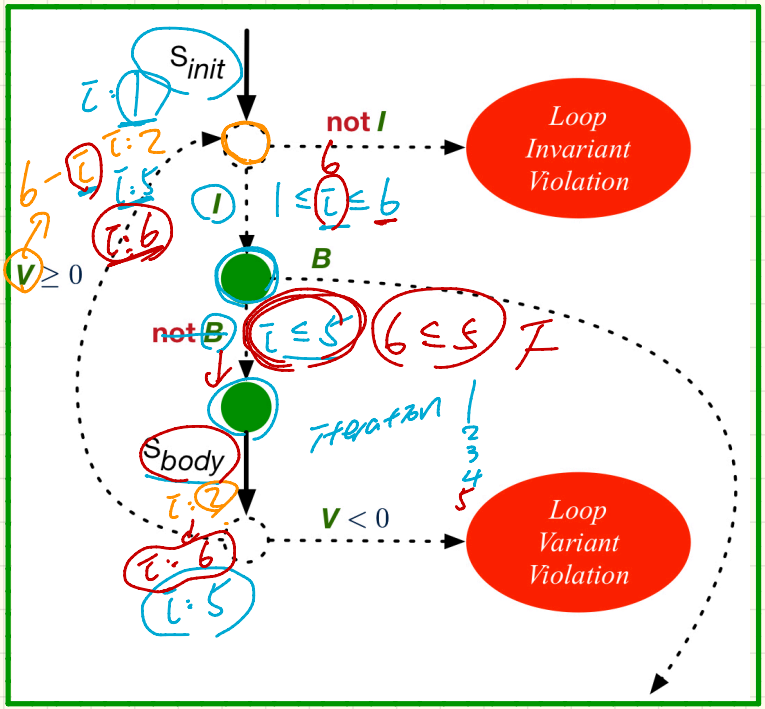


Contracts of Loops: Example

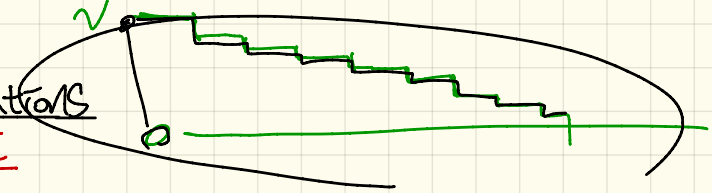
Example

```
test
  local
    i: INTEGER
  do
    from
      i := 1
    invariant
      1 <= i and i <= 6
    until
      i > 5
    loop
      io.put_string ("iteration " + i.out
      i := i + 1
    variant
      6 - i
    end
end
```

Runtime Checks



Contracts of Loops: Violations



iteration	i	6 - i
	I	
1st	3	3
2nd	5	1
3rd	7	-1

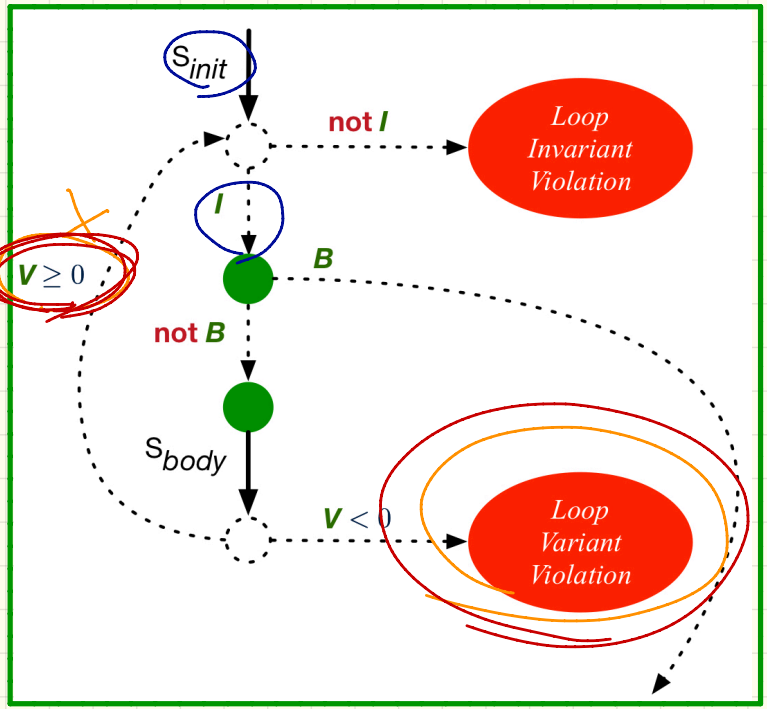
Example

```
test
  local
    i: INTEGER
  do
    from
      i := 1
    invariant
      1 <= i and i <= 6
    until
      i > 5
    loop
      io.put_string ("iteration " + i.out
      i := i + 1
    variant
      6 - i
    end
end
```

Correctness

Termination

Runtime Checks

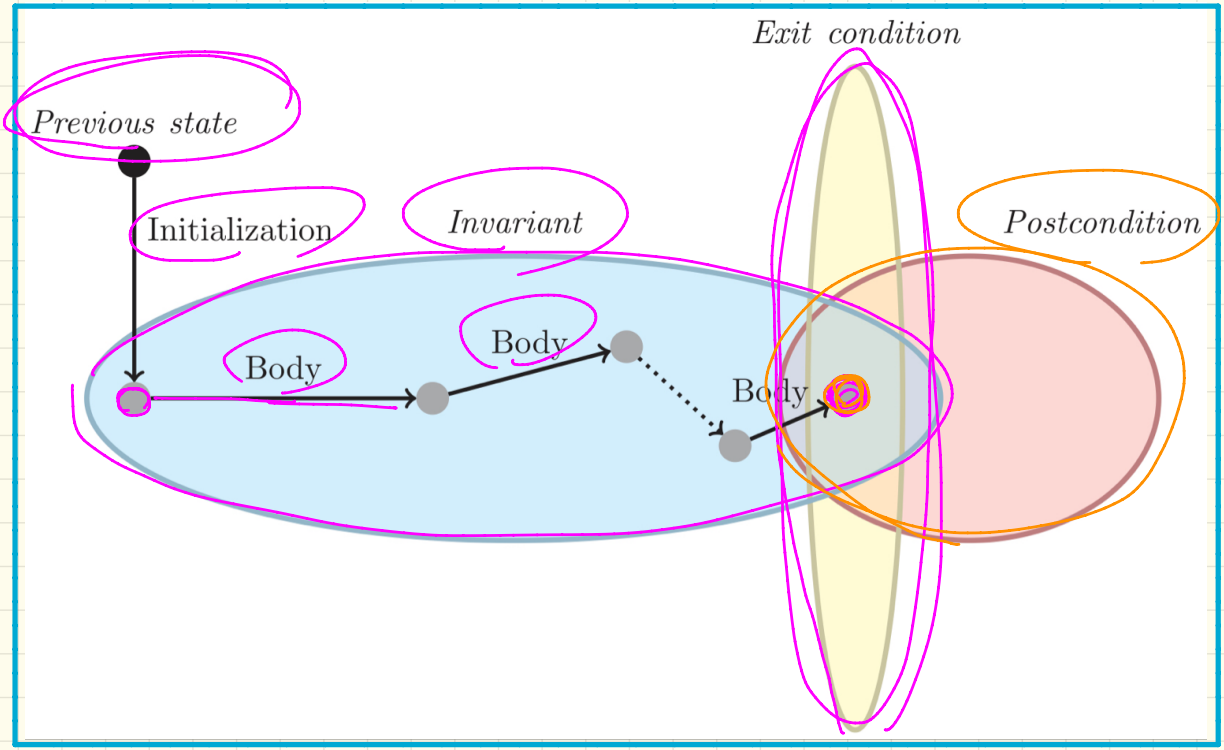


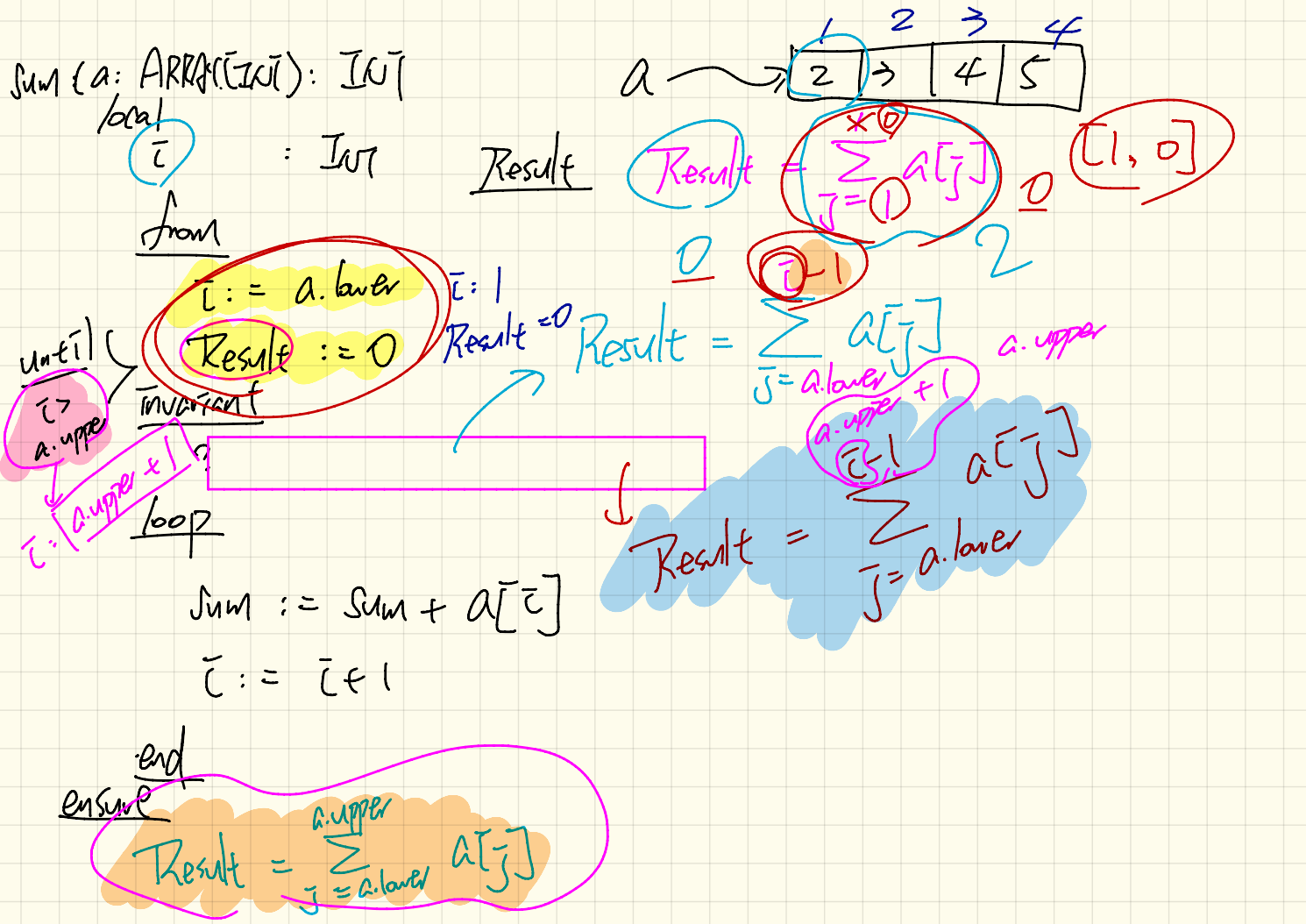
Invariant Violation: $1 \leq i \leq 5$

Variant Violation: $5 - i$

Skipping Loop Body: $i > 0$

Contracts of Loops: Visualization




sum (a: ARRAY[INT]): INT
local
i : INT
from
i := a.lower
Result := 0
invariant
until
i > a.upper
loop
i := a.upper + 1 ?
sum := sum + a[i]
i := i + 1
end
ensure
Result = Σ a[j] (j = a.lower .. a.upper)
Result
Result = Σ a[j] (j = a.lower .. i-1)
i : 1
Result = 0
a
1 2 3 4
2 4 5
Result + Σ a[j] (j = 1 .. 0)
0
[1, 0]
0
i-1
2
a.upper
a.upper + 1
i : 1
Result = Σ a[j] (j = a.lower .. i-1)

Tuesday Dec. 4

Lecture 24

Review Sessions for Exam

LAS C

2pm ~ 4pm Thursday Dec. 6

11am ~ 1pm Friday Dec. 7

Confirm your attendance on Moodle !

Marks for:

Labs and Lab Test

Available around Exam day

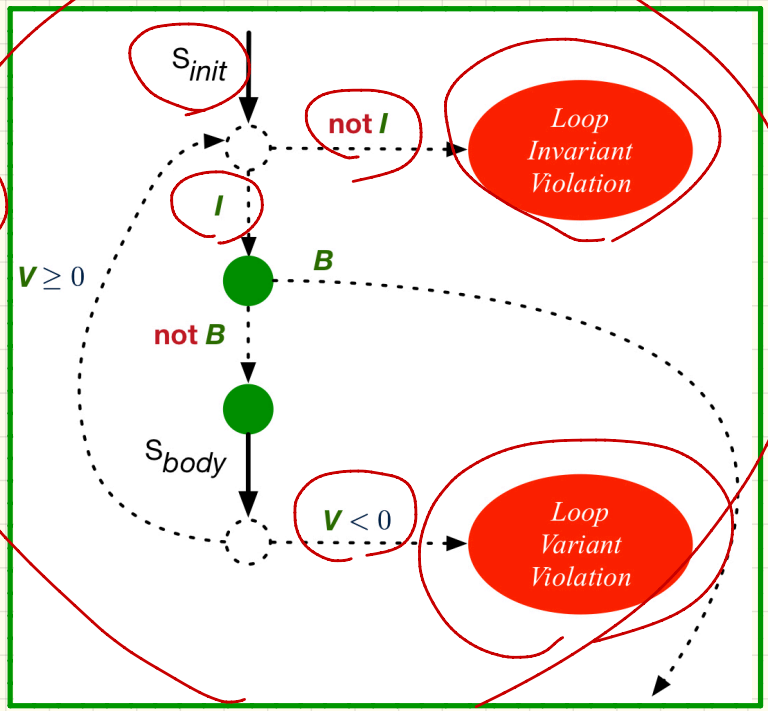
You will be able to speak to me about these shortly after the exam.

Contracts of Loops

Syntax

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

Runtime Checks

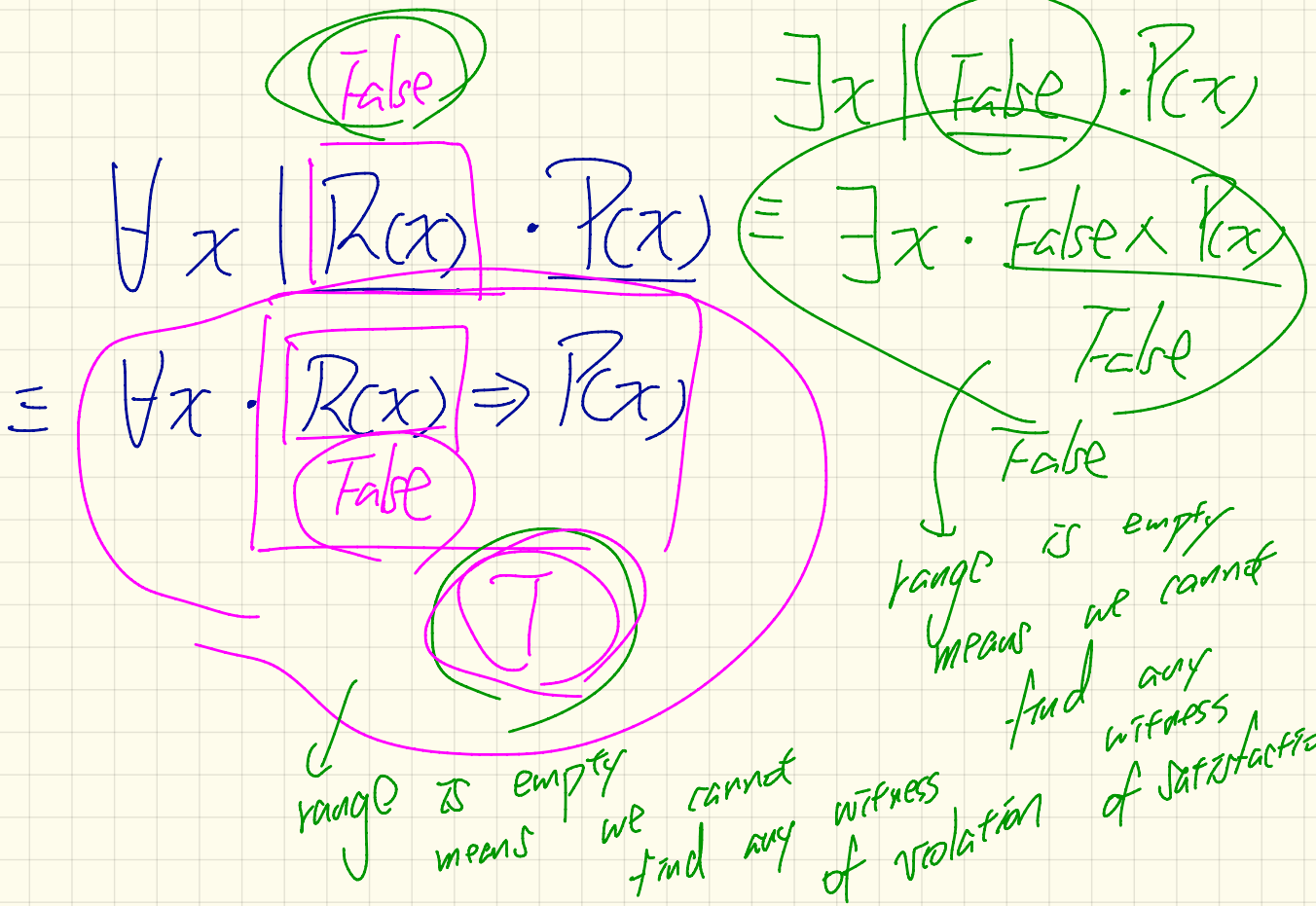


Finding Max : v1

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: -- ∀j | a.lower ≤ j ≤ i • Result ≥ a[j]
      across a.lower |..| i as j all Result >= a [j.item] end
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper - i + 1
    end
  ensure
    correct_result: -- ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
      across a.lower |..| a.upper as j all Result >= a [j.item]
  end
end
```

After Iteration	i	Result	LI	Exit (i > a.upper)?	LV
Initialization	1	20	✓	×	●
1st	2	20	✓	×	●
2nd	3	20	×	●	●

Exercise: change i := i+1 of LI of Sbody.


False
$\forall x \mid R(x) \cdot P(x)$
$\equiv \forall x \cdot R(x) \Rightarrow P(x)$
False
T
range is empty
means we cannot
find any witness
of violation
$\exists x \mid \text{False} \cdot P(x)$
$\equiv \exists x \cdot \text{False} \wedge P(x)$
False
False
range is empty
means we cannot
find any
witness
of satisfaction

Finding Max : v2

20	10	40	30

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: -- ∀j | a.lower ≤ j < i • Result ≥ a[j]
        across a.lower |..| (i - 1) as j all Result >= a [j.item] end
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper - i
    end
  ensure
    correct_result: -- ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
      across a.lower |..| a.upper as j all Result >= a [j.item]
  end
end
```

i ↑ LV ↓

After Iteration	i	Result	*LI*	Exit (*i* > *a.upper*)?	*LV*
Initialization	1	20	✓	×	–
1st	2	20	✓	×	2
2nd	3	20	✓	×	1
3rd	4	40	✓	×	0
4th	5	●	●	●	-1

Proof Obligations for Correct Loops

$I \wedge B \Rightarrow R$

$\{I \wedge \neg B\}\ S_{body}\ \{V \geq 0\}$

$\{I\}\ \neg B\ S_{body}\ \{I\}$

```
{Q}   from
        S_init
      invariant
        I
      until
        B
      loop
        S_body
      variant
        V
      end        {R}
```

$\{I \wedge \neg B\}\ S_{body}\ \{\ \}$

$V < \text{old } V$

- A loop is ***partially*** *correct* if:
  - Given precondition ***Q***, the initialization step S_{init} establishes ***LI*** *I*.

    $$\{Q\}\ S_{init}\ \{I\}$$

  - At the end of S_{body}, if not yet to exit, ***LI*** *I* is maintained.

    $$\{I \wedge \neg B\}\ S_{body}\ \{I\}$$

  - If ready to exit and ***LI*** *I* maintained, postcondition ***R*** is established.

    $$I \wedge B \Rightarrow R$$

- A loop *terminates* if:
  - Given ***LI*** *I*, and not yet to exit, S_{body} maintains ***LV*** *V* as non-negative.

    $$\{I \wedge \neg B\}\ S_{body}\ \{V \geq 0\}$$

  - Given ***LI*** *I*, and not yet to exit, S_{body} decrements ***LV*** *V*.

    $$\{I \wedge \neg B\}\ S_{body}\ \{V < V_0\}$$

Proof Obligations for Correct Loops: Example

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: ∀j | a.lower ≤ j < i • Result ≥ a[j]
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper − i + 1
    end
  ensure
    correct_result: ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
  end
end
```

I

Initialization:

{True} i := a.lower ; Result := a[i] {I}

Before Termination:

Upon Termination:

Non-Negative Variant:

Decreasing Variant:

Prove

Establishment of Loop Invariant:

```
{ True }
  i := a.lower
  Result := a[i]
{ ∀j | a.lower ≤ j < i • Result ≥ a[j] }
```

$wp(\ i := a.lower\ ;\ Result := a[i]\ ,\ \forall j \mid a.l \le j < i \bullet R \ge a[j])$

$= \{$ wp rule for ; $\}$

$wp(\ i := a.lower\ ,\ wp(\ Result := a[i]\ ,\ \forall j \mid a.l \le j < i \bullet R \ge a[j]))$

$= \{$ wp rule for := $\}$ $\quad [1 := 1]$

$wp(\ i := a.lower\ ,\ \forall j \mid a.l \le j < i \bullet a[i] \ge a[j]\ )$

$= \{$ wp rule for := $\}$

$\forall j \mid a.l \le j < a.l \bullet a[a.l] \ge a[j] \equiv T$

Prove

Establishment of Postcondition upon Termination:

$$(\forall j \mid a.lower \le j < i \bullet Result \ge a[j]) \wedge i \ge a.upper \Rightarrow \forall j \mid a.lower \le j \le a.upper \bullet Result \ge a[j]$$

$P \Rightarrow Q$

$\Rightarrow (\forall x \mid Q \cdot R)$

$\Rightarrow (\forall x \mid P \cdot R)$

$(\forall x \mid 1 \le x \le 10 \cdot x^2 \le 100)$

$\Rightarrow (\forall x \mid 1 \le x \le 9 \cdot x^2 \le 100)$

$(\forall x \mid 1 \le x \le 9 \cdot x^2 \le 81)$

$\Rightarrow (\forall x \mid 1 \le x \le 10 \cdot x^2 \le 81)$

Prove

Loop Variant Stays Non-Negative Before Exit:

{ $(\forall j \mid a.lower < j < i \bullet Result \geq a[j]) \wedge \neg(i > a.upper)$ }

```
if a [i] > Result then Result := a [i] end
i := i + 1
```

{ $a.upper - i + 1 \geq 0$ }

End of Notes

All the Best !