EECS3311 Software Design (Fall 2020)

Q&A - Lecture Series W1

Monday, September 14

- Lab1 released (2 weeks)
  * Finishing Lab0 asap helps.
  * Instructions (background)
  * Starter tests (precise documentation)
  * Optional tutorials on node and tree routines
- Quiz1 due: 5pm EST, Friday Sep. 18
- Scheduled labs on Wednesday:
  10am - 11:30am
  2:30pm to 4pm
- Lab0 questions?

Missing Case from L5 and L11?

```
1  public class AccountV2 {
2    public AccountV2(String owner, int balance) throws
3      BalanceNegativeException
4    {
5      if( balance < 0 ) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10     WithdrawAmountNegativeException, WithdrawAmountTooLargeExceptior
11     if( amount < 0 ) { /* negated precondition */
12       throw new WithdrawAmountNegativeException(); }
13     else if( balance < amount ) { /* negated precondition */
14       throw new WithdrawAmountTooLargeException(); }
15     else { this.balance = this.balance - amount; }
16   }
```

can we just have an invariant violation ?

balance 100
amount

0

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

Weaker vs. Stronger Precondition

acc.withdraw_1
acc.withdraw_2
100

withdraw_1 (amount)

require

P_1: amount < balance

allows the case when amount = balance

amount ≤ balance

amount < balance

1
2
3 ... 99

amount = balance

Weaker precondition has more satisfying input values than stronger ones.

withdraw_2 (amount)

require

P_2: amount ≤ balance

⇒ Weaker precondition is more tolerant on client.

① P_1 ⇒ P_2

② P_2 ⇒ P_1

Stronger

amount < balance

⇒

amount ≤ balance

weaker

amount / balance 100

No contract violation is good?

What is a Good Design?

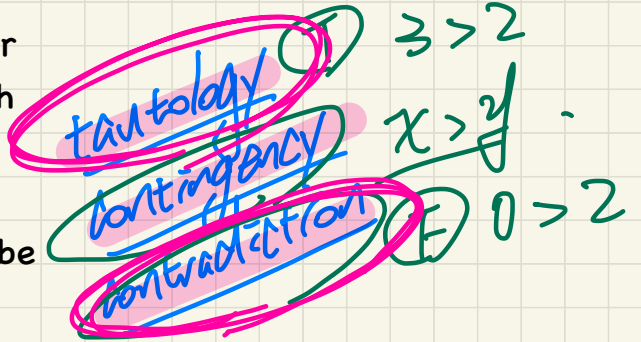
- A "good" design should *explicitly* and *unambiguously* describe the *contract* between **clients** (e.g., users of Java classes) and **suppliers** (e.g., developers of Java classes).
  We call such a contractual relation a *specification*.
- When you conduct *software design*, you should be guided by the "appropriate" contracts between users and developers.
  - Instructions to **clients** should *not be unreasonable*.
    e.g., asking them to assemble internal parts of a microwave
  - Working conditions for **suppliers** should *not be unconditional*.
    e.g., expecting them to produce a microwave which can safely heat an explosive with its door open!
  - You as a designer should strike proper balance between **obligations** and **benefits** of clients and suppliers.
    e.g., What is the obligation of a binary-search user (also benefit of a binary-search implementer)? [The input array is sorted.]
  - Upon contract violation, there should be the fault of **only one side**.
  - This design process is called *Design by Contract (DbC)*.

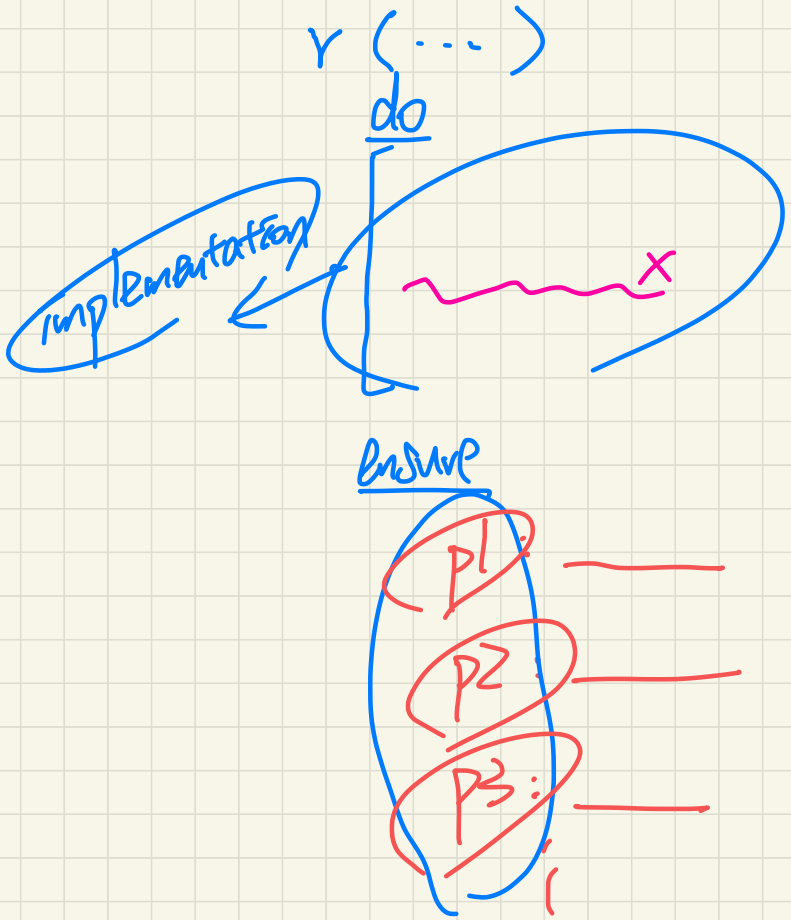
10 of 72

Given a faulty implementation, having no contract violation means your contract is incomplete.

```
abs_val(i: INTEGER): INTEGER
  do
    -- wrong imp.
    if i > 0 then
      Result := i
    end
  ensure
    i > 0 implies Result = i
  end
```

there should be exactly one violation of the contract (by either supplier or client) and that it's better than having none or both violate the contract... I understand why it's better than both violate, but I don't understand why is it better than none violating the contract (in other words, shouldn't the best case be no violations of the contract by either supplier nor client)





① When imp. is correct,

P1, P2, P3,

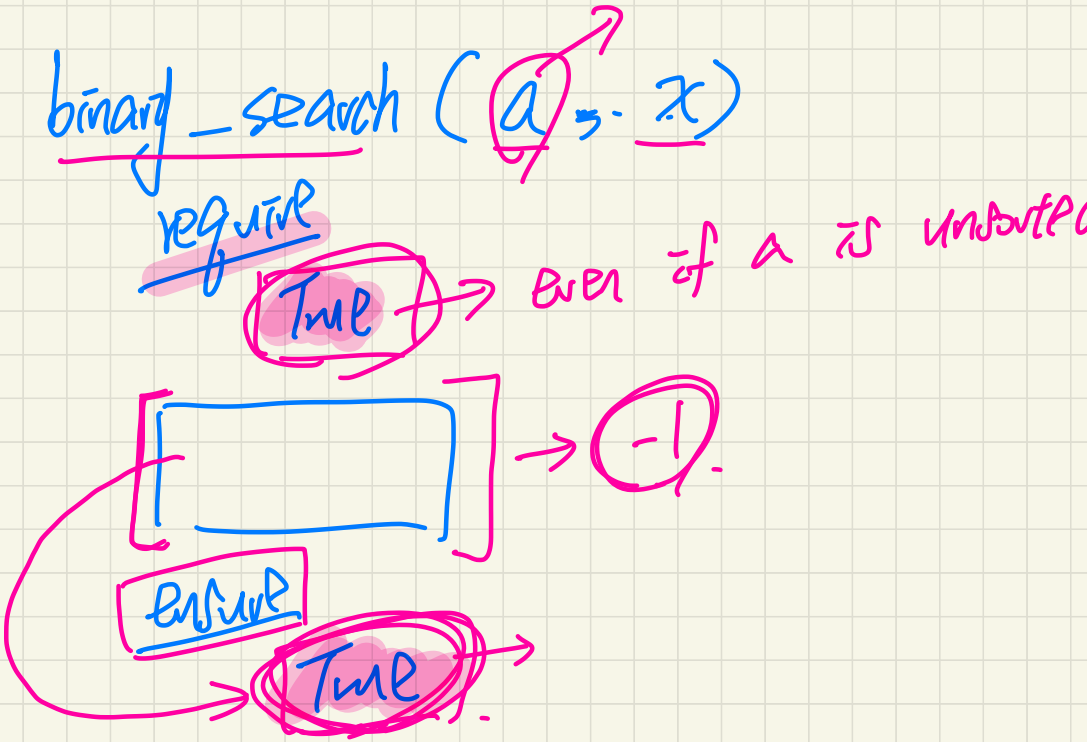
should all be satisfied.

② When imp. is incorrect,

at least one of

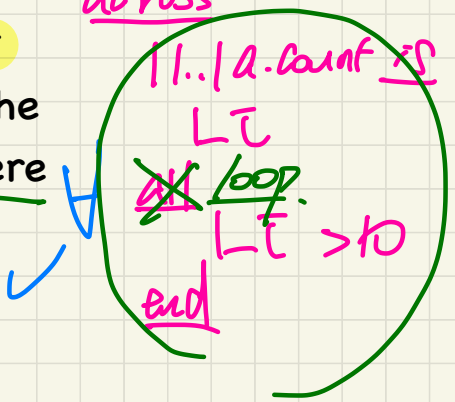
P1, P2, P3

should be violated.


binary_search (a, x)
require
True
even if a is unsorted
-1
ensure
True

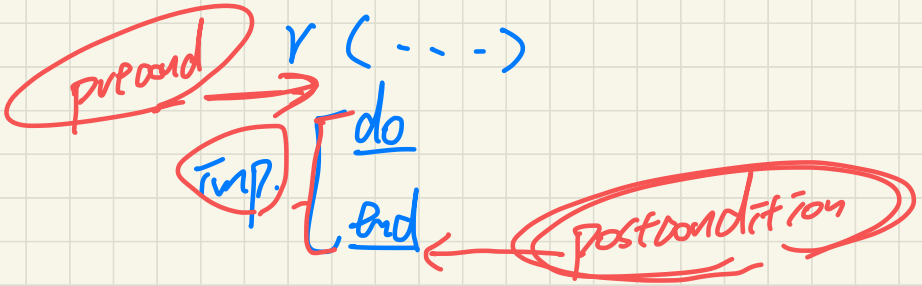
What is a contract?

What exactly does the "contract" part in design by contract mean? Is it referring to the boolean expressions stated in the precondition, postcondition, and class invariant? So when there is a contract violation, the violation refers to a condition where one of those boolean expressions is false?



Think of contract as adding an extra layer of declaration properties (using predicates), specifying obligations/benefits between routines, against which your implementation/code is checked against.

instruction (implemen.)



Importance of a postcondition

In Tutorial Part 5: Why does the first incidence of the incorrect query with the assertion result in an assertion violation error but the second incidence results in a postcondition violation error? Both incidents violate the postcondition so shouldn't both of them result in a postcondition violation error if the postcondition is checked before the assertion is checked?

```
is_month_with_31_days (m: INTEGER): BOOLEAN
    -- Is `m` a month with 31 days?
  require -- precondition (valid input constraints)
    valid_month: 1 <= m and m <= 12
  local
    months: ARRAY[INTEGER]
  do
    -- alternative 1
    Result := (m = 1 or m = 3 or m = 5 or m = 7 or m = 8 or m = 10 or m = 12)
    -- alternative 2
    months := <<1, 3, 5, 7, 8, 10, 12>>
    Result := False
  ensure -- postcondition (relationship between inputs and outputs)
    class -- this query belongs to the class (static)
```

```
is_month_with_31_days (m: INTEGER): BOOLEAN
    -- Is `m` a month with 31 days?
  require -- precondition (valid input constraints)
    valid_month: 1 <= m and m <= 12
  local
    months: ARRAY[INTEGER]
  do
    -- alternative 1
    Result := (m = 1 or m = 3 or m = 5 or m = 7 or m = 8 or m = 10 or m = 12)
    -- alternative 2
    months := <<1, 3, 5, 7, 8, 10, 12>>
    Result := False
  ensure -- postcondition (relationship between inputs and outputs)
    class -- this query belongs to the class (static)
    correct_result:
      (m = 1 or m = 3 or m = 5 or m = 7 or m = 8 or m = 10 or m = 12) = Result
  end
```

Faulty Supplier, Weak Contract

v1: no postcondition violation

v2: postcond. violation

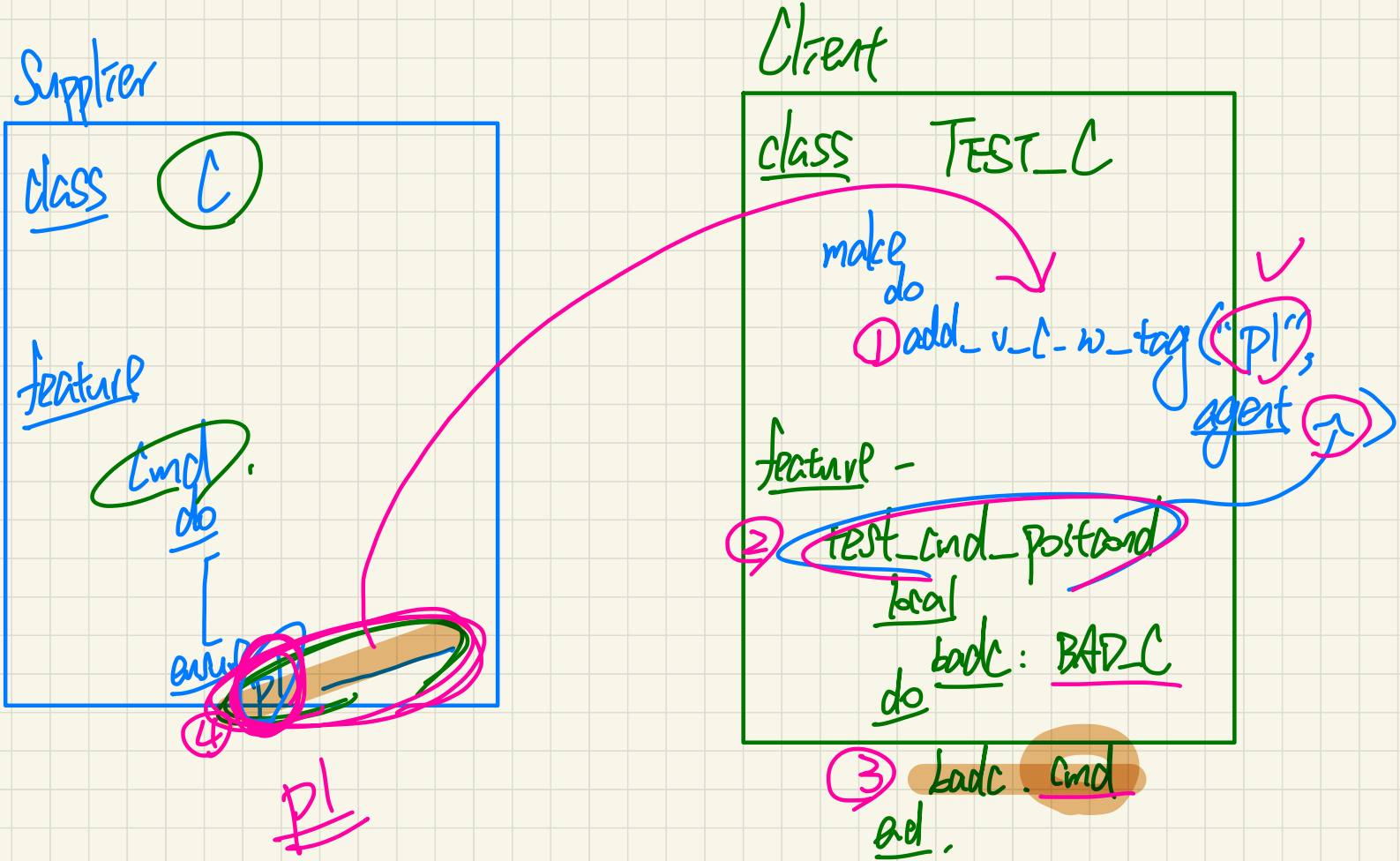
client

```
t_static_query: BOOLEAN
  do
    comment ("t_static_query: test is_month_with_31_days")
    -- for a boolean test query to pass,
    -- 1. no contract violations and last re-assigned value of Result
    --    must be True
    Result := {BIRTHDAY}.is_month_with_31_days (12)
    -- for each intermediate re-assignment to Result,
    -- we must make sure it's not re-assgined to false.
    check Result end
    Result := not {BIRTHDAY}.is_month_with_31_days (4)
  end
```

v1: check assertion error.

↳ not satisfactory (no postcond. viol.)

Faulty Supplier, Strong Contract


Supplier
class C
feature
cmd
do
Client
class TEST_C
make
do
add_v_C_w_tag ("P1",
agent
feature -
test_cmd_postcond
local
badc : BAD_C
do
badc . cmd
P1

Bank Accounts in Java: Version 4 Critique

```
1  public class AccountV4 {
2    public void withdraw(int amount) throws
3      WithdrawAmountNegativeException, WithdrawAmountTooLargeException
4    { if(amount < 0) { /* negated precondition */
5        throw new WithdrawAmountNegativeException(); }
6      else if (balance < amount) { /* negated precondition */
7        throw new WithdrawAmountTooLargeException(); }
8      else { /* WRONT IMPLEMENTATION */
9        this.balance = this.balance + amount; }
10     assert this.getBalance() >= 0 :
11       owner + "Invariant: positive balance"; }
```

wrong imp.
↳ there should

Supplier

```
1  public class BankAppV4 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jeremy with balance 100:")
4      try { AccountV4 jeremy = new AccountV4("Jeremy", 100);
5            System.out.println(jeremy);
6            System.out.println("Withdraw 50 from Jeremy's account:");
7            jeremy.withdraw(50);
8            System.out.println(jeremy); }
9      /* catch statements same as this previous slide:
10      * Version 2: Why Still Not a Good Design? (2.1) */
```

100 +50 150

```
Create an account for Jeremy with balance 100:
Jeremy's current balance is: 100
Withdraw 50 from Jeremy's account:
Jeremy's current balance is: 150
```

no postcond. violation will occur.

So when there's a contract violation, only one side should be blamed.
But in this example,
you said that it is not the case. Is it because both supplier and client are to be blamed, or neither?

Le some postcond. violation

Client

④ Once you're sure "everything is working", you can turn off contracts

① Contracts are for guarding your imp.

```
bubble_sort ( a: ARRAY[INTEGER] ): ARRAY[INT]
```

② Contract need not be efficient.

③ Imp. should be efficient.

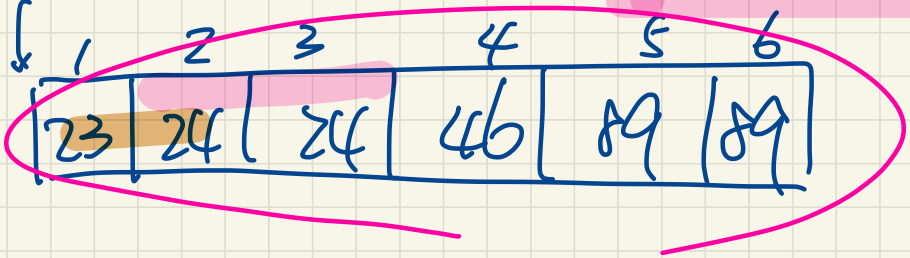
ensure → finalize

a.count = 1^M

Array_is_sorted:

O(n) → not efficient

a

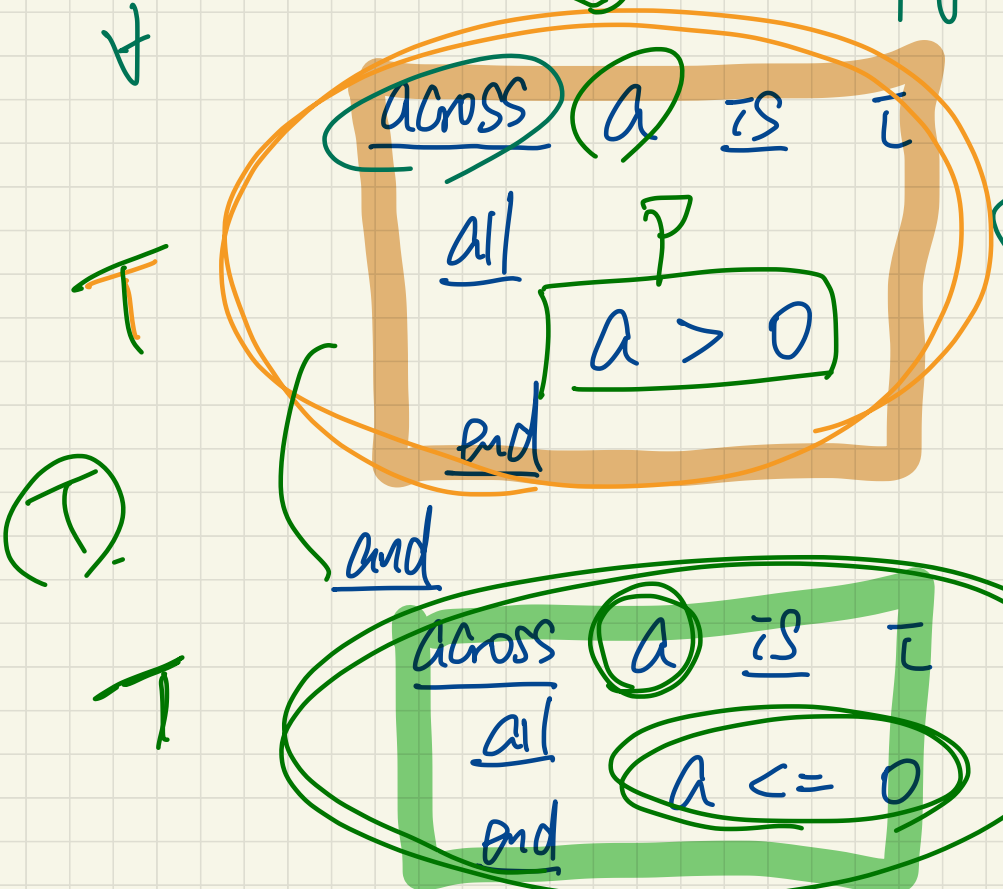


$a[1] \le a[2]$

$a[2] \le a[3]$

a: ARRAY[INTEGER]

create a.make_empty.



is_all_pos(a)

↳ is_all_pos(<<>>)

↳ T

∀

T

T

T

$P \wedge \neg P \equiv F$

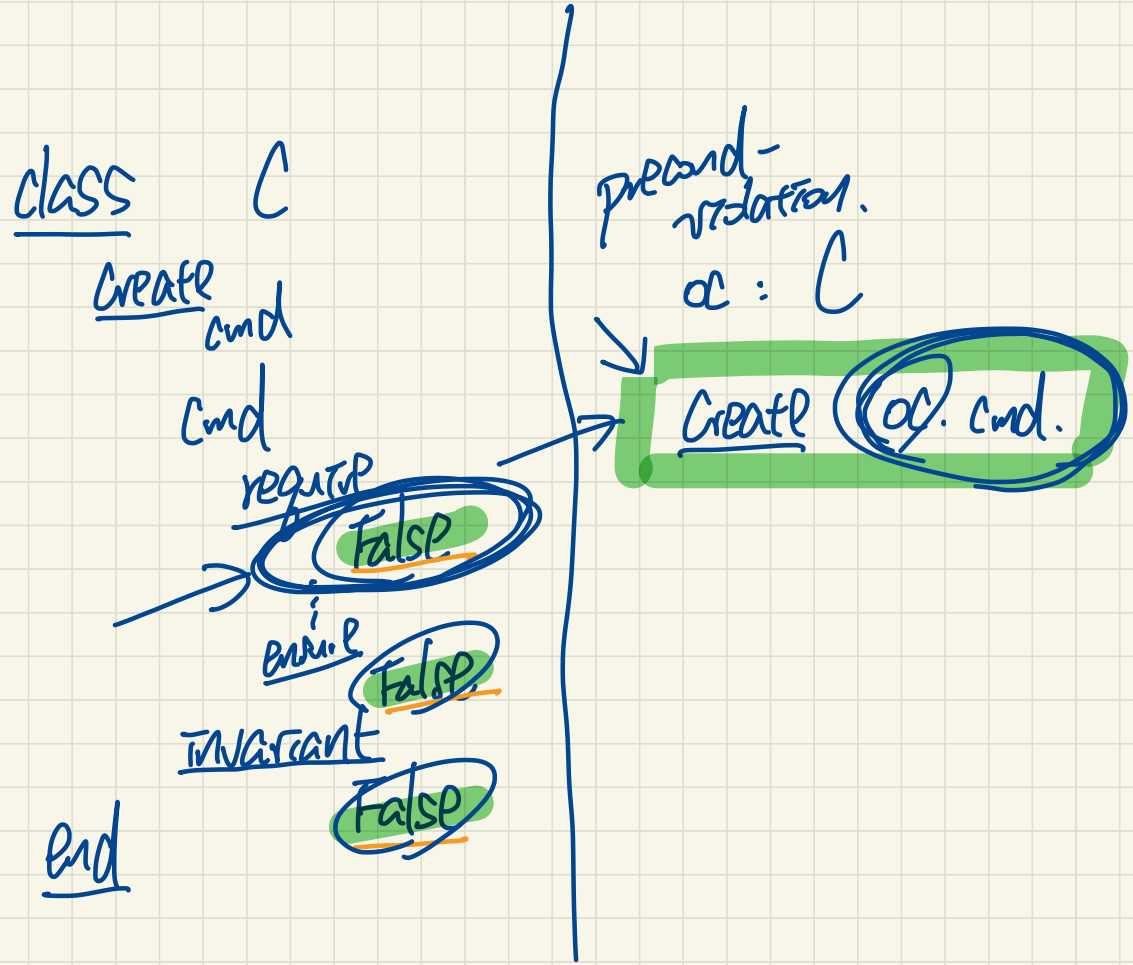
① T

② F

witness

$\exists x \in \emptyset \; P(x) = T$

↳ "you can't find any violation of P(x)"


class C
create
cmd
cmd
require
False
ensure
False
invariant
False
end
precond-
violation.
oc : C
Create (oc. cmd.