EEC S3311 Software Design (Fall 2020)

Q&A - Exam

Friday, December 18

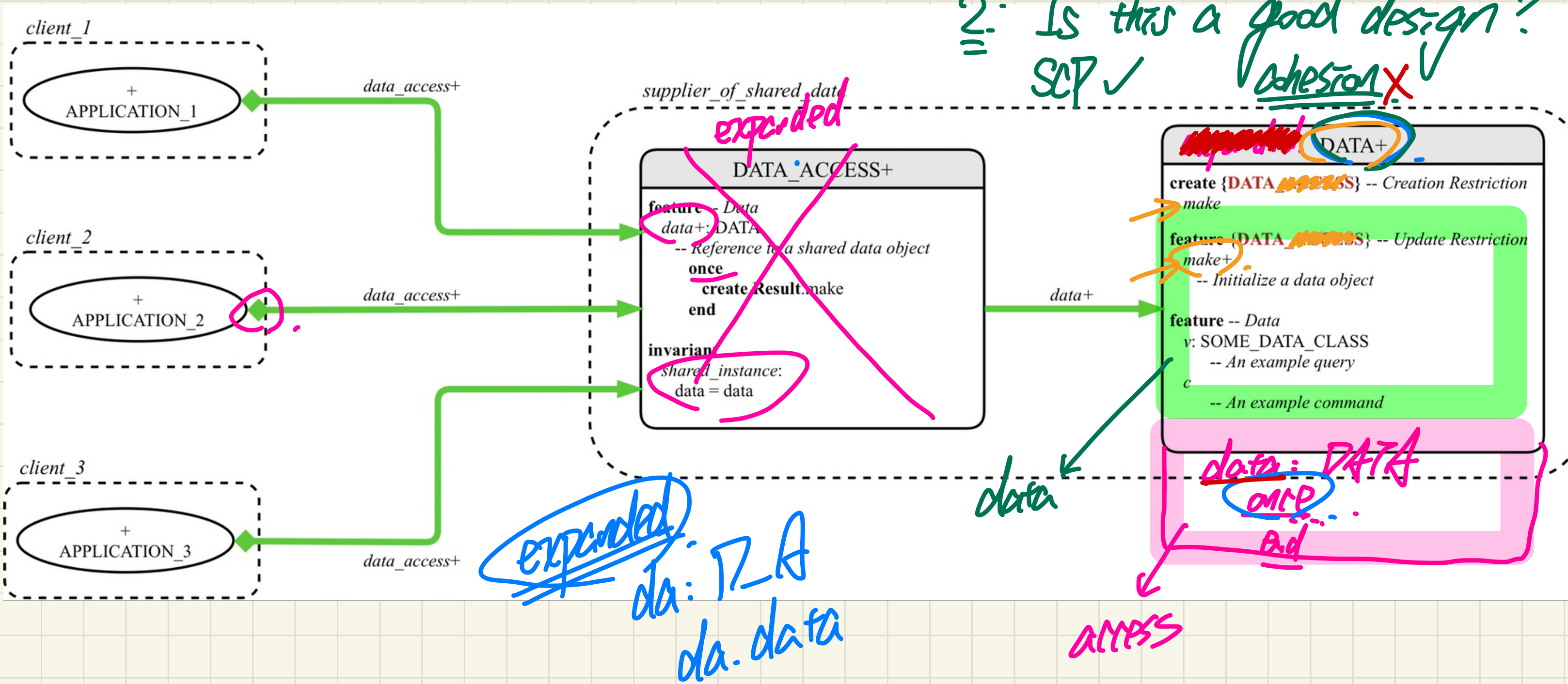
Reem: for the singleton design pattern, would the design pattern still be satisfied if we get rid of the DATA_ACCESS class and add "expanded" and "once" inside of DATA class?

expanded data : DATA once . . . end

cohesion

1. Will we still have the singleton access to DATA ? ✓

2. Is this a good design ? SCP ✓ cohesion ✗



Parthiv - Can you explain this question please?

```
class
  BANK_DATA
create {BANK_DATA_ACCESS}
  make
feature {BANK_DATA_ACCESS}
  make
    do
      interest_rate := 1.04
    end
feature -- Data Attributes
  interest_rate: REAL
  set_interest_rate (r: REAL)
    do
      interest_rate := r
    end
end
```

```
expanded class
  BANK_DATA_ACCESS
feature
  d: BANK_DATA
    do
      create Result.make
    end
invariant
  d ~ d
end
```

```
class
  TEST_SINGLETON
-- constructor omitted
feature
  t: BOOLEAN
    local
      data1, data2: BANK_DATA
      bda: BANK_DATA_ACCESS
    do
      comment("t: test singleton")
      data1 := bda.d
      data2 := bda.d
      Result := data1 = data2
    end
end
```

failing test.

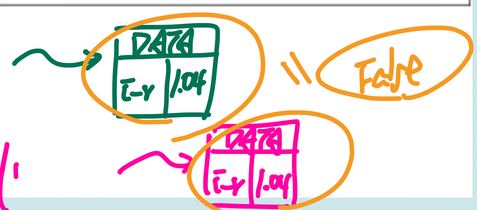
assume: = is equal. redefined.

once.

invariant not appropriate

checked each call after 'd'

F



Not appropriate ∵ the 'd' routine returns distinct objects upon each call

Invariant wrong but will we get inv. violation? No.

What happens when we try to run the test `t` (via the Workbench System in EStudio)? Choose the right description.

```
class
  BANK_DATA
create {BANK_DATA_ACCESS}
  make
feature {BANK_DATA_ACCESS}
  make
    do
      interest_rate := 1.04
    end
feature -- Data Attributes
  interest_rate: REAL
  set_interest_rate (r: REAL)
    do
      interest_rate := r
    end
end
```

```
expanded class
  BANK_DATA_ACCESS
feature
  d: BANK_DATA
    once
      create Result.make
    end
invariant
  d ~ d
end
```

```
class
  TEST_SINGLETON
-- constructor omitted
feature
  t: BOOLEAN
    local
      data1, data2: BANK_DATA
      bda: BANK_DATA_ACCESS
    do
      comment ("t: test singleton")
      data1 := bda.d
      data2 := bda.d
      Result := data1 = data2
    end
end
```

What happens when we try to run the test `t` (via the Workbench System in EStudio)? Choose the right description.

expanded type
x. y.
x = y → content
x ~ y → content

1. Test passes. ① by default, all att. are comp. ② is_equal may be redefined to compare certain att.
2. Singleton pattern spec is not correct
↳ if the imp. was not done correctly (e.g. do), then inv violation would not occur. (but still exp. to occur)

imp. of sing. patt.
too weak (= should've been used).
spec.
T

$x = y$

① x, y <u>ref</u> type = ⇒ address comp

② x, y <u>expanded</u> type = ⇒ content comp (all att.)

$x \sim y$

depends on the redefined version of `is_equal`, if any.

Can you explain Proof tip (2) in W12 lecture note (mentioned below)?

a[i] → $1 \le i \le a.count$.

When calculating $wp(S, R)$, if either program S or postcondition R involves array indexing, then R should be augmented accordingly.

① If you're given the Hoare Triple to prove, always augment the condition that array indices are valid.

1.1 Establishing the Loop Invariant

Proof Obligation:

$$\{\ a.count > 0\ \}$$

```
i := a.lower; Result := a[i]
```

$$\{\ \forall j \mid a.lower \le j \le i-1 \bullet \boxed{a.lower \le j \land j \le a.upper} \land \textbf{Result} \ge a[j]\ \}$$

② You might be asked to state the Hoare Triple yourself.

1.2 Maintaining the Loop Invariant

Proof Obligation:

$$\{\ \neg(i > a.upper) \land (\ \forall j \mid a.lower \le j \le i-1 \bullet \boxed{a.lower \le j \land j \le a.upper} \land \textbf{Result} \ge a[j]\ )\ \}$$

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

$$\{\ \forall j \mid a.lower \le j \le i-1 \bullet \boxed{a.lower \le j \land j \le a.upper} \land \textbf{Result} \ge a[j]\ \}$$

```
r (...)
  require
    x > 4
  do
    y := z + x
  ensure
    y >= x + 4
  end
```

① **Hoare Triple**

$\{x > 4\}\ y := z + x\ \{y \geq x + 4\}$

② **Calculate**

$wp(y := z + x,\ y \geq x + 4)$

$= \{\ ...\ \}$

$= \{\ ...\ \}$

③ $x > 4 \Rightarrow$ ( )

Termination.

1. Program is guaranteed to terminate if there are no loops.

   (no need to prove termination).

2. Program with loops is likely to loop forever.

   ① loop variant

   (2.1) $LV \geq 0$

   (2.2) $LV_0 > LV$.

Denoting old values in proofs

$\underline{x}$ (post-state).

① $\underline{old}$ x ← pre-state.

② x_0

Also about the solution to wp proof exercise question 1.1,
I solved it and came up with "to prove" to be
a.count>0 -> ∀j | a.lower ≤ j ≤ a.lower -1 • a.lower ≤ j ≤ a.upper ∧ a.lower >= a[j].
Is this correct? And how do we do the full proof from here?

1.1 Establishing the Loop Invariant

Proof Obligation:

$\{ a.count > 0 \}$

```
i := a.lower; Result := a[i]
```

$\{ \forall j \mid a.lower \leq j \leq i - 1 \bullet \boxed{a.lower \leq j \wedge j \leq a.upper} \wedge \textbf{Result} \geq a[j] \}$

Regarding the wp proof exercise that you gave us,
can you explain how the proof works (such as split range and antecedent)?
I understand all the other previous steps,
but have trouble coming up with proof techniques.

$a[i] > R \land R \geq a[j] \Rightarrow a[i] > a[j]$

(I.2) Maintenance of LI.

To prove: (not exit.) (LI.) (wp.)

$$\neg(i > a.upper) \land (\forall j \mid a.lower \leq j \leq i-1 \bullet a.lower \leq j \land j \leq a.upper \land \textbf{Result} \geq a[j])$$
$$\implies a[i] > \textbf{Result} \implies \forall j \mid a.lower \leq j \leq i \bullet a.lower \leq j \land j \leq a.upper \land a[i] \geq a[j]$$

Proof:

$\forall j \mid a.lower \leq j \leq i \bullet a.lower \leq j \land j \leq a.upper \land a[i] \geq a[j]$

$\equiv$ {split range: $\forall j \mid a.lower \leq j \leq i \bullet P(j) \equiv (\forall j \mid a.lower \leq j \leq i-1 \bullet P(j)) \land P(i)$}

$(\forall j \mid a.lower \leq j \leq \mathbf{i - 1} \bullet a.lower \leq j \land j \leq a.upper \land a[i] \geq a[j]) \land (a.lower \leq \mathbf{i} \land \mathbf{i} \leq a.upper \land a[i] \geq a[\mathbf{i}])$

$\equiv$ {antecedent: $a[i] > \text{Result}$; and RHS of precond: $\forall j \mid a.lower \leq j \leq i-1 \bullet a.lower \leq j \land j \leq a.upper \land \text{Result} \geq a[j]$}

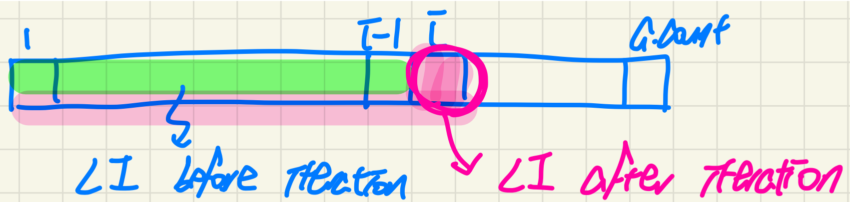
$true \land (a.lower \leq i \land i \leq a.upper \land a[i] \geq a[i])$

$\equiv$ {LHS of precond: $\neg(i > a.upper)$ and $a[i] \geq a[i] \equiv true$}

$true$

$\neg(i > a.upper)$

$\equiv i \leq a.upper.$



Eric -

Can you guide us through calculating the wp and proving and disproving this proof ?
I got stuck calculating the wp. I was only able to apply the rules {wp rule : sequential comp.}, {wp rule : variable assignment}, {wp rule : conditional}, and then i got stuck. I wasn't sure if you could perform {wp rule variable assignment} when the variables didn't exist. Also the proving/disproving part.

2.1 Loop Variant Stays Positive

Proof Obligation:

$$\{ \neg(i > a.upper) \land (\forall j \mid a.lower \le j \le i-1 \bullet \boxed{a.lower \le j \land j \le a.upper} \land \textbf{Result} \ge a[j]) \}$$

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

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

Proof tip 1.

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

$\neg(i > a.upper) \Rightarrow (a[i] > R \Rightarrow a.upper - i \ge 0)$

Assume: $\neg(i > a.upper)$, $a[i] > R$

$i \le a.upper$

$a.upper - i \ge 0$

$wp(\text{if } a[i] > R \text{ then } R := a[i] \text{ else } R := R \text{ end}; \; i := i+1; \; a.upper - i + 1 \ge 0)$

$= wp(\text{if } a[i] > R \text{ then } R := a[i] \text{ else } R := R, \; wp(i := i+1, \; a.upper - (i+1) + 1 \ge 0))$

$a.upper - i \ge 0$

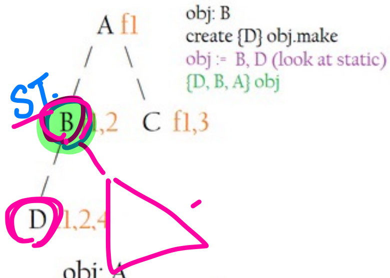
$a[i] > R \Rightarrow wp(R := a[i], \; a.upper - i \ge 0)$

$= a[i] > R \Rightarrow a.upper - i \ge 0$

Reem: the following two pictures are what I understood variable assignment and type casting is,
but there were some inconsistency when I redid the quizzes…
for some reason this explanation seemed off but not sure why…

Variable assignment: (purple)
_ can assign to children only.

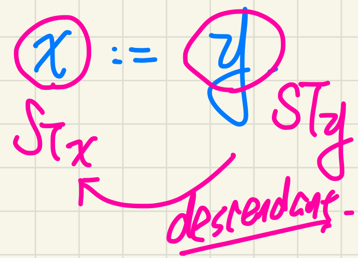
Casting: (green)
_ can cast to parents only.
_ can only cast to something that has all the features or less.



obj: B
create obj.make
obj := D, B
{A,B} obj
D

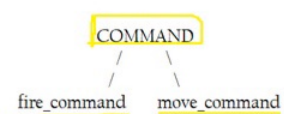
obj: A
create obj.make
obj := B,C,D, A
{A} obj .... nothing else

① Variable Assignments.



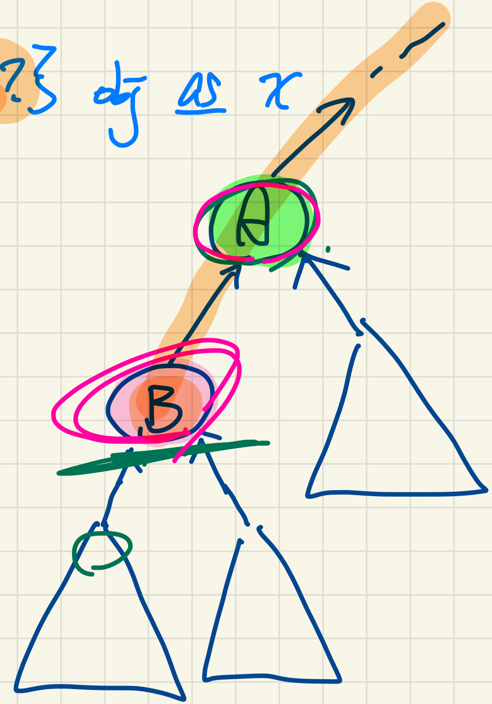
The following is the example that brought the inconsistency…. So in order to satisfy the given answer in one of the questions from quiz 7, the way history array in lab3 must be as follows. But then that means you can downcast (to children) which doesn't match what I said in the definition in previous picture for casting…

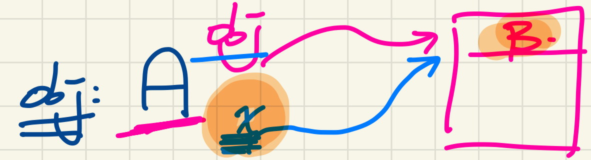
history[COMMAND] = {fire, move, projectile}



A_correct = {{fire}static_command, {move}static_command, {projectile}static_command}

```
check attached {??} obj as x
  -- x
end
```






```
create {B} obj.make.
```

what can we cast obj into without any runtime violation?

must be satisfiable by obj's dynamic type: ancestor of B.

Daniel - Could you explain

why the query `client.get_f(2).fa` is invalid from this question on quiz 8 please?

client.get_f(2).fa

```
class
  B
inherit
  G
  redefine fa end
feature
  fa: STRING
    do
      Result := "B.fa"
    end
  fb: STRING
    do
      Result := "B.fb"
    end
end
```

```
class
  D
inherit
  A
  redefine fa end
feature
  fa: STRING
    do
      Result := "D.fa"
    end
  fd: STRING
    do
      Result := "D.fd"
    end
end
```

```
class
  G
inherit
  A
  redefine fa end
feature
  fa: STRING
    do
      Result := "G.fa"
    end
  fg: STRING
    do
      Result := "G.fg"
    end
end
```

```
class
  F
feature
  ff: STRING
    do
      Result := "F.ff"
    end
end
```

```
class
  C
inherit
  F
feature
  fc: STRING
    do
      Result := "C.fc"
    end
end
```

```
class
  E
inherit
  G
feature
  fe: STRING
    do
      Result := "E.fe"
    end
end
```

```
class
  A
inherit
  F
feature
  fa: STRING
    do
      Result := "A.fa"
    end
end
```

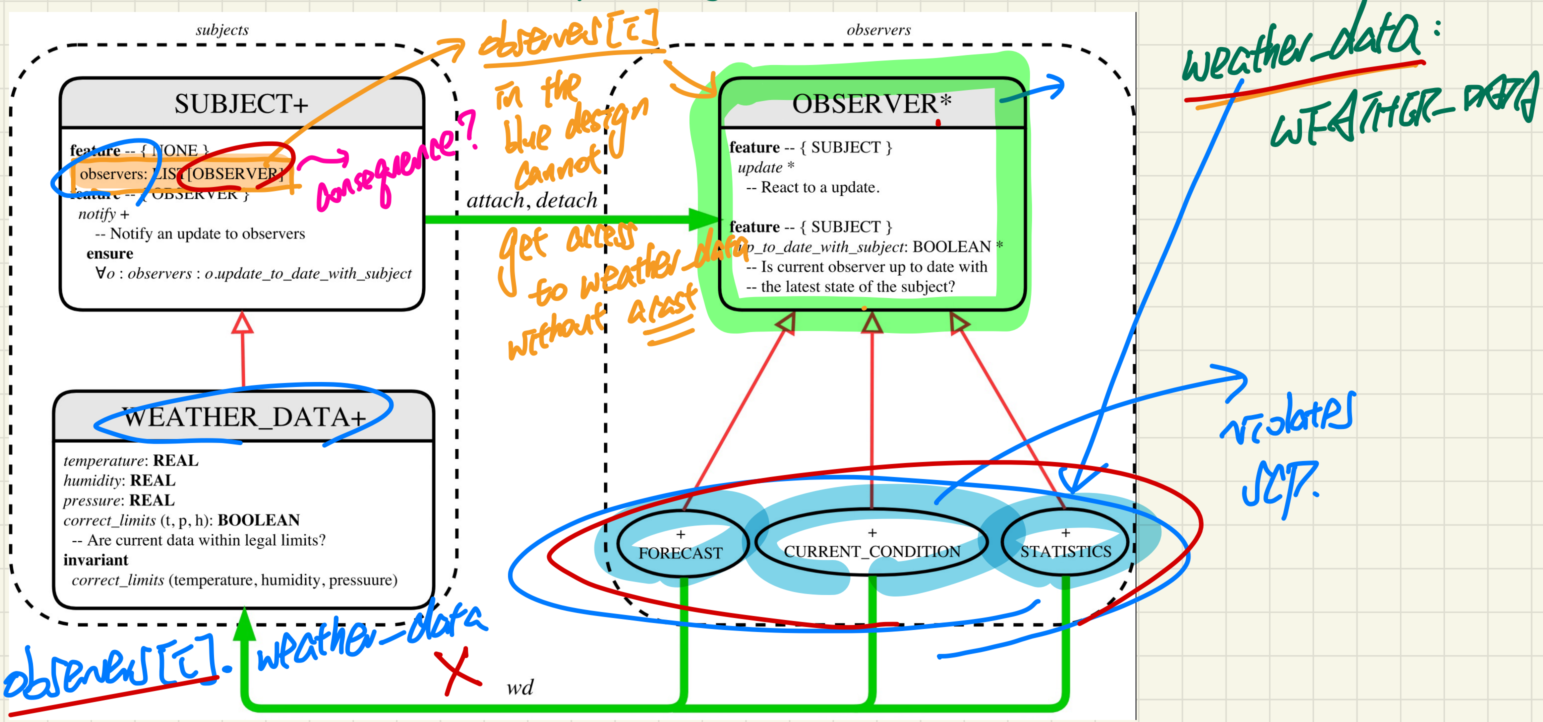
```
class
  CLIENT_2
create
  make
feature -- Collection
  a: LIST[A]
  make
    do
      create {LINKED_LIST[A]} a.make
    end
feature -- Routines
  add_a (obj: A)
    do
      a.extend (obj)
    end
  get_a (i: INTEGER): A
    require
      1 <= i and i <= a.count
    do
      Result := a[i]
    end
  get_f (i: INTEGER): F
    require
      1 <= i and i <= a.count
    do
      Result := a[i]
    end
end
```

client.get_f(2).fa — Invalid

fa not declared in ancestors of F.

Stefan -

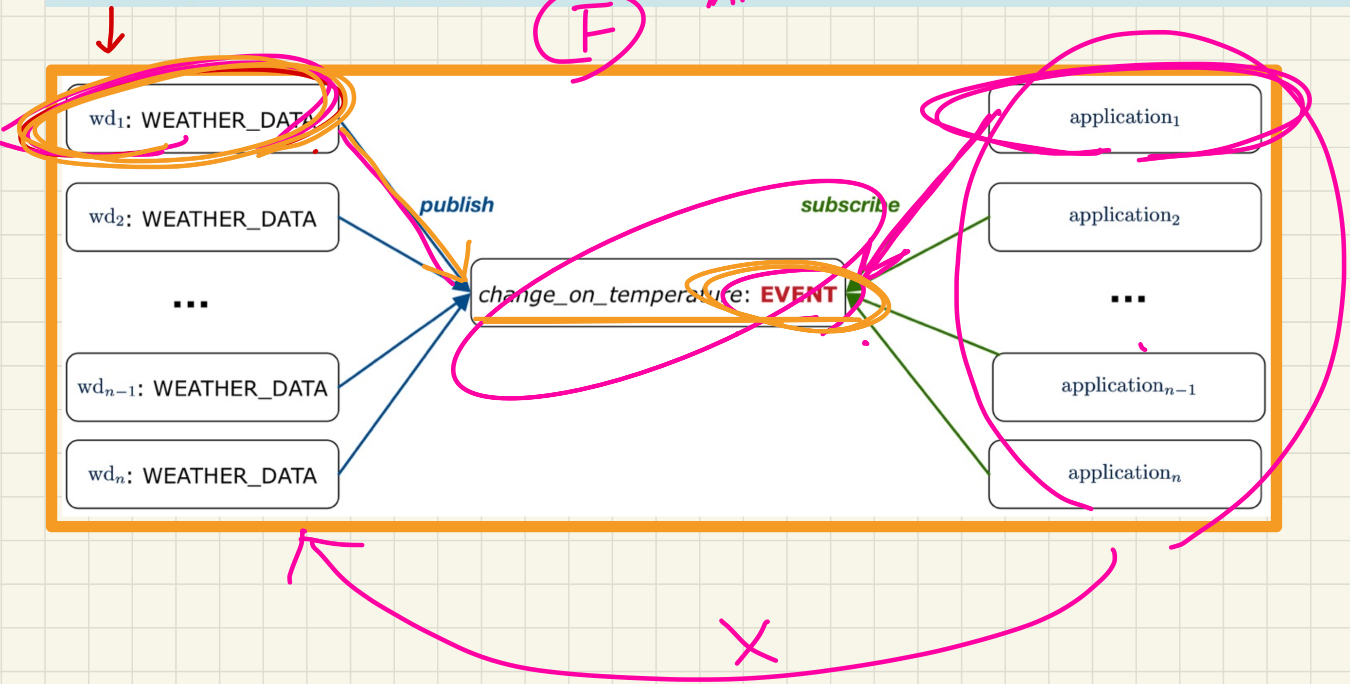
In the Observer Pattern Lecture, the second design attempt, is the variable weather_data: WEATHER_DATA declared at the level of the deferred class OBSERVER, or at the level of the corresponding descendants of OBSERVER?



Mohammad: Can you explain the following questions from quiz 9? Thank you.

For the design problem of a distributed client/server system, consider the **3rd design attempt** (i.e., the event-driven design) discussed in the lecture. When there is an update occurring in a subject/server, the subject/server notifies all subscribed clients/observers by invoking their update commands (stored previously for delayed execution).

F



Mahnoor - Lecture 9, slide #31/37 -
What is the relationship of subscribe and publish and
how it is ensured that subscribe will be invoked
(in class CURRENT_CONDITIONS) when the publish command is called?
In addition, what is the significance use of agent in the constructor in the
descendent classes of observer (e.g. CURRENT_CONDITIONS)

```
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
```

observer.

```
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
```

humidity.

subject

```
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
```

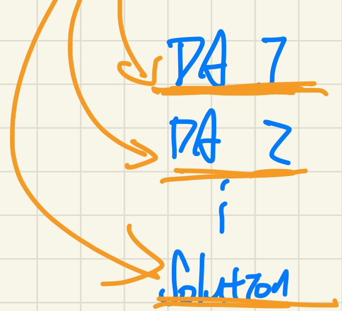
event.

```
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.h
    do actions.extend (an_action)
    ensure action_subscribed: action.has(an_action) end
  publish (args: G)
    do from actions.start until actions.after
       loop actions.item.call (args) ; actions.forth end
    end
end
```

Sabreena: Should we study the Design Attempts in some lectures, the ones before we learn a new design pattern?
For example: Design Attempt 1, Design Attempt 2, before learning The Design pattern. (quiz 9, design attempts 1 and 2)

YES

Parthiv - Proving or justifying that a design violates a certain design principle is not that difficult but can you please guide us(a general idea) on how to justify or prove that a particular design attempt satisfies a particular design principle.
(namely Information Hiding, SCP, Programming from Interface and not from the Implementation and cohesion)



Pavel: Professor, could you, please, go over questions in quiz 12?

"A loop invariant occurs at the end of 1st iteration"

```
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
```

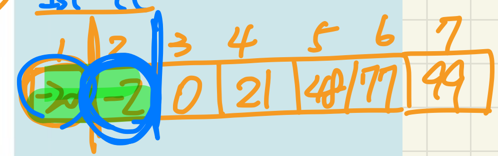
You can assume that any input passed to the routine is non-empty.

At the runtime, what will happen if we invoke the above **find_max** routine with the input array <<-20, -2, 0, 21, 48, 77, 99>>?

loop counter i

① before 1st it. (T)

② At the end of 1st it. i=2



Result: -20

Hi professor, in quiz 2, could you explain why (1)(2) does not compile, and why (3) is true?

```
class A
feature -- attributes
  i: INTEGER
  b: B
end

class B
feature -- attributes
  s: STRING
end
```

Now assume the following variable declaration:

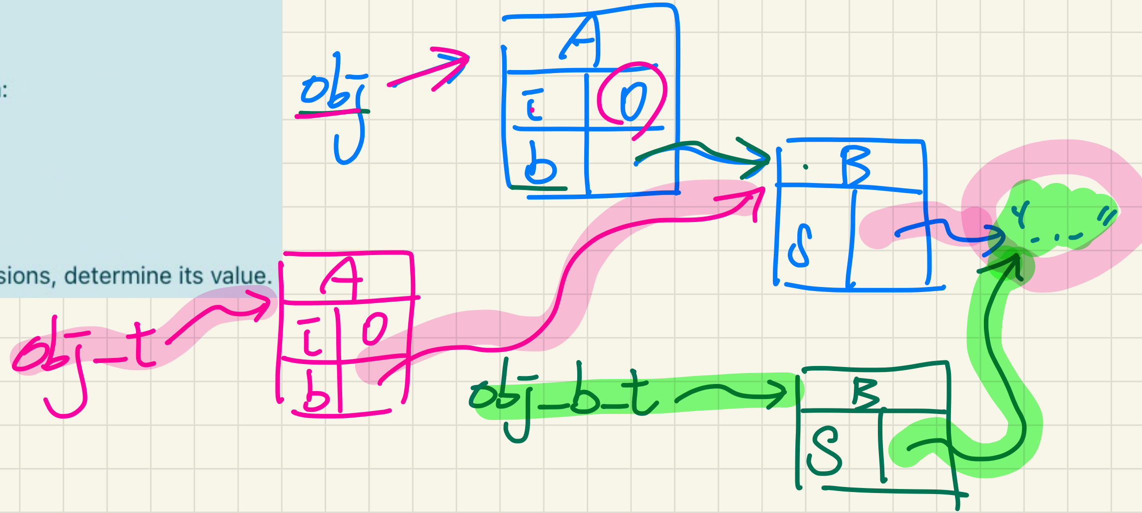
```
obj: A
```

And the following initialization:

```
create obj.make
```

Now, for each of the following Boolean expressions, determine its value.

obj.b.**twin**.s = obj.**twin**.b.s	true
obj.b.i = obj.b.**twin**.i	It does not compile
obj.b.i = obj.b.**deep_twin**.i	It does not compile



Varuhn - I am still confused how the cardinality of question 4 on quiz 4 is 16. Could you please help me find out how this answer is derived?

Given two sets S and T, say we write:

- $S \bigvee T$ for their union
- $S \bigwedge T$ for their intersection
- $S \setminus T$ for their difference

What is the **cardinality** of the power set of ({a, b, c, d, e} \ {a, f}) \ {f, g}? Enter an integer value (with no spaces).

Answer: 16

2^4

$\{b, c, d, e\}$

$\{a, b, c, d, e\} \setminus \{a, f\}$

$= \{b, c, d, e\}$

$\{b, c, d, e\} \setminus \{f, g\} = \{b, c, d, e\}$

0000 $\emptyset$

0001 $\{e\}$

0010 $\{d\}$

⋮

1111 $\{b, c, d, e\}$