EECS3311 Software Design (Fall 2020)

Q&A - Lecture Series W1

Tuesday, September 15

How is DbC Useful in Guiding System Development?

Client's View:

- A console application.
- Keep entering names randomly until done.
- Keep inquiring if a name exists until quit.

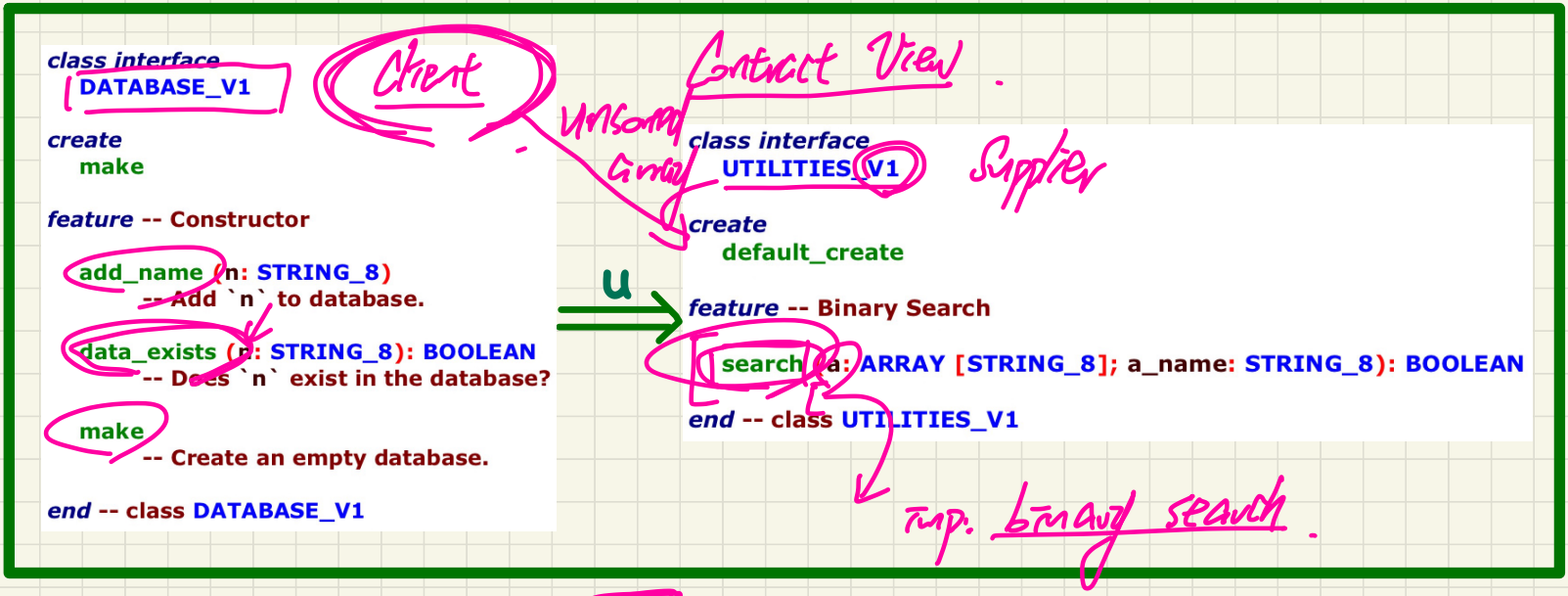
Expected Run

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
a  b  c  d  e
Enter a name, or `quit` to stop inquiring: a
a exists!
Enter a name, or `quit` to stop inquiring: b
b exists!
Enter a name, or `quit` to stop inquiring: c
c exists!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e exists!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```

Supplier's Implementation Strategy

- Store names in an array.
- Upon an inquiry: Binary Search,

Version 1: Wrong Implementation, No Contracts



- Data array in DATABASE is not kept sorted.
- Binary search in UTILITIES does not require a sorted input array.
- When user enters names in an unsorted order, output is wrong.
- But no contract violation!
- A bad design is when something goes wrong, there is no party to blame.

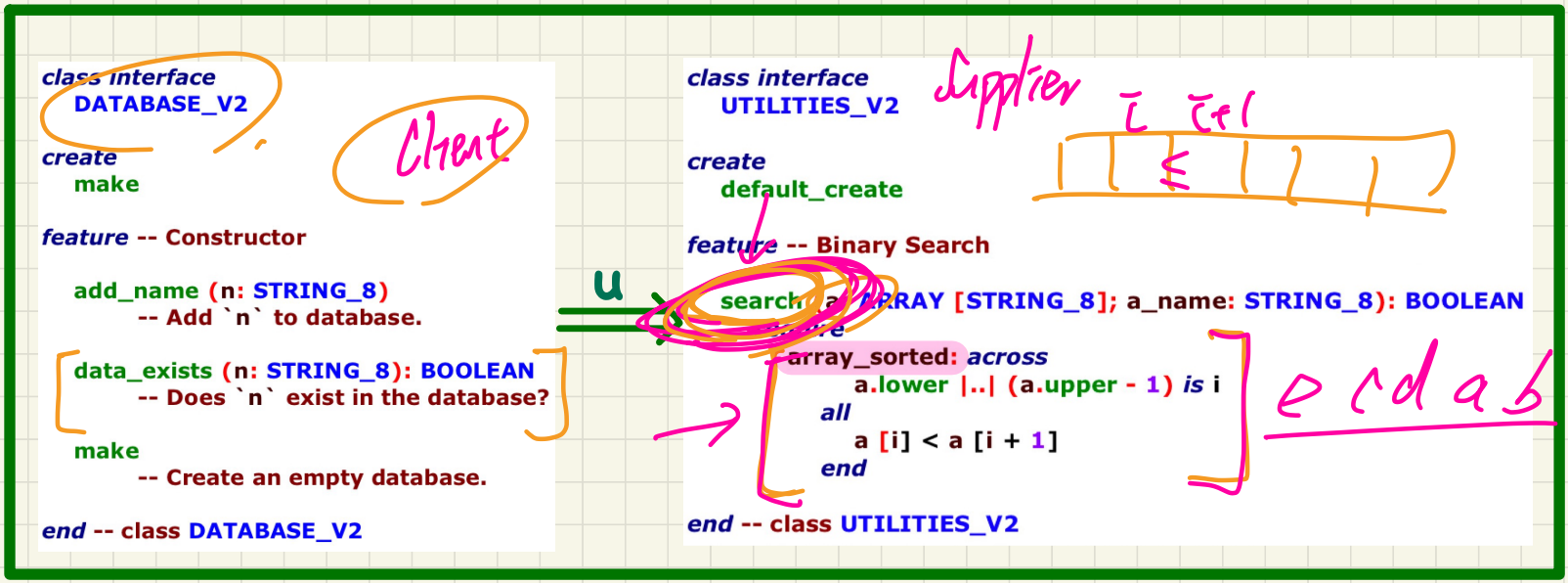
Version 1: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
e  c  d  a  b
Enter a name, or `quit` to stop inquiring: a
a does not exist!
Enter a name, or `quit` to stop inquiring: b
b does not exist!
Enter a name, or `quit` to stop inquiring: c
c does not exist!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e does not exist!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```

unexpected result (lin. search on unsorted array)

↳ but no contract violation to signal an error.

Version 2: Wrong Implementation, Proper Precondition



```
class interface
   DATABASE_V2

create
   make

feature -- Constructor

   add_name (n: STRING_8)
         -- Add `n` to database.

   data_exists (n: STRING_8): BOOLEAN
         -- Does `n` exist in the database?

   make
         -- Create an empty database.

end -- class DATABASE_V2
```

```
class interface
   UTILITIES_V2

create
   default_create

feature -- Binary Search

   search (a: ARRAY [STRING_8]; a_name: STRING_8): BOOLEAN
         require
            array_sorted: across
               a.lower |..| (a.upper - 1) is i
            all
               a [i] < a [i + 1]
            end

end -- class UTILITIES_V2
```

- Data array in DATABASE is not kept sorted.
- Binary search in UTILITIES now requires a sorted input array.
- When an unsorted array is passed for search, a contract violation occurs!
- A good design is when something goes wrong, there is one party to blame.

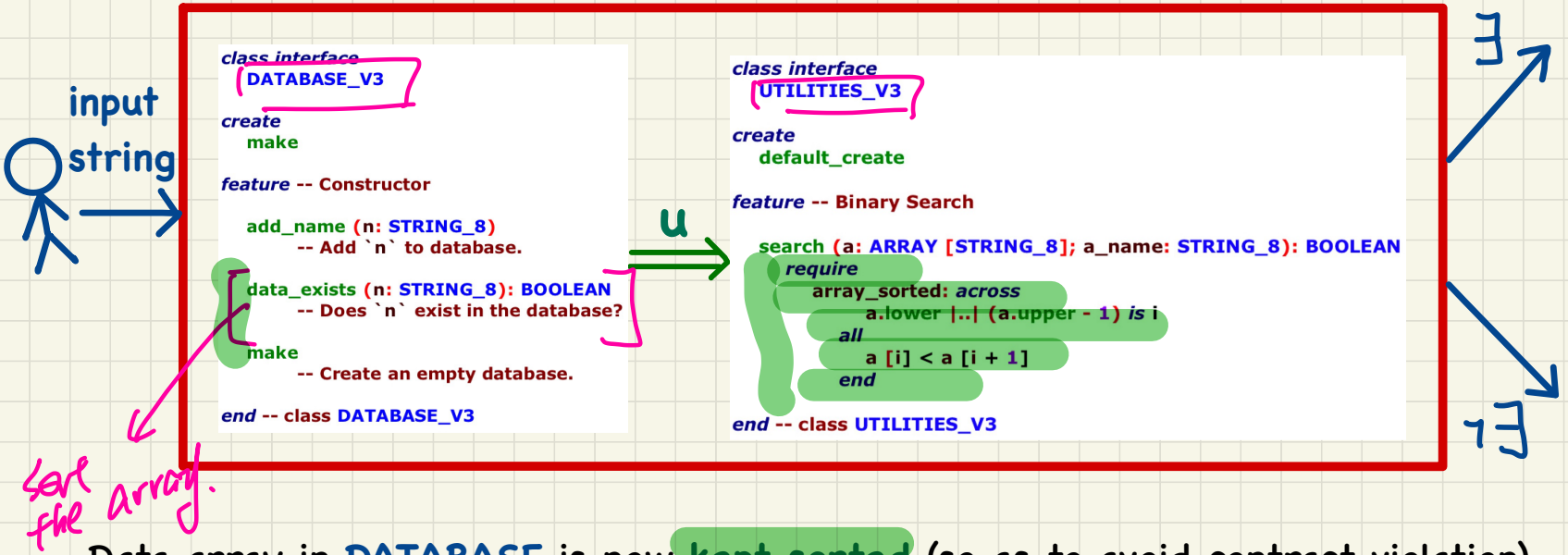
Version 2: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
e  c  d  a  b
Enter a name, or `quit` to stop inquiring: a

why-dbc-useful: system execution failed.
Following is the set of recorded exceptions:

******************************** Thread exception ********************************
In thread           Root thread           0x0 (thread id)
**********************************************************************************
--------------------------------------------------------------------------------
Class / Object      Routine               Nature of exception          Effect
--------------------------------------------------------------------------------
UTILITIES_V2        search @1
<000000010EFF0FB8>                        array_sorted:
                                          Precondition violated.       Fail
--------------------------------------------------------------------------------
DATABASE_V2         data_exists @2
<000000010EFEFDF8>                        Routine failure.             Fail
--------------------------------------------------------------------------------
ROOT                make @30
<000000010EFEF548>                        Routine failure.             Fail
--------------------------------------------------------------------------------
ROOT                root's creation
<000000010EFEF548>                        Routine failure.             Exit
--------------------------------------------------------------------------------
```

1. unstacked away passed to Eiffel side

2. a precond. violation has happened, forcing the supplier to change.

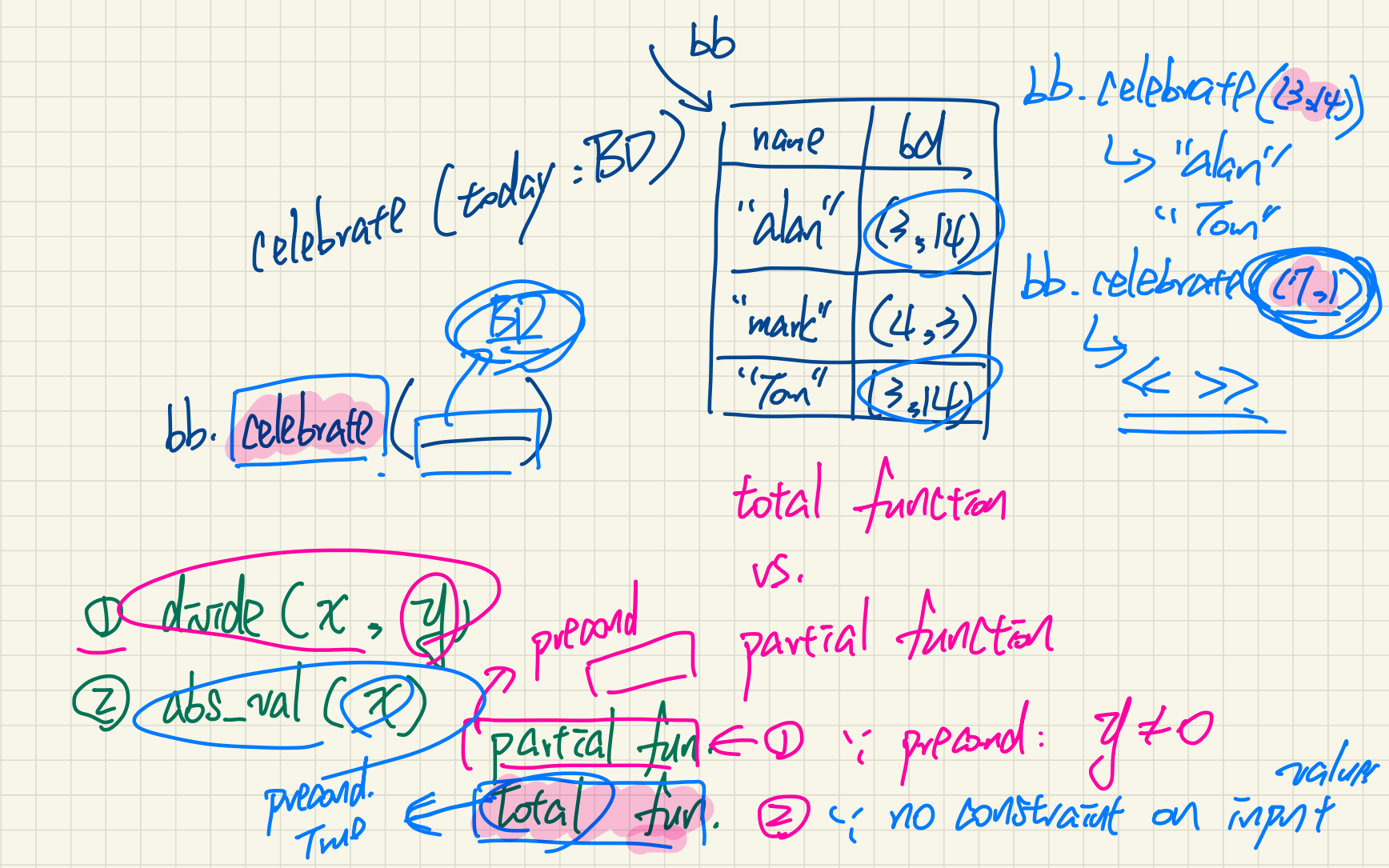
Version 3: Fixed Implementation, Proper Precondition



- Data array in DATABASE is now kept sorted (so as to avoid contract violation).
- Binary search in UTILITIES still requires a sorted input array.
- A sorted array is always passed for search, a contract violation never occurs!
- Now finalize/deliver the working system with contracts checking turned off.

Version 3: User Interaction Session

```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
a  b  c  d  e
Enter a name, or `quit` to stop inquiring: a
a exists!
Enter a name, or `quit` to stop inquiring: b
b exists!
Enter a name, or `quit` to stop inquiring: c
c exists!
Enter a name, or `quit` to stop inquiring: d
d exists!
Enter a name, or `quit` to stop inquiring: e
e exists!
Enter a name, or `quit` to stop inquiring: f
f does not exist!
Enter a name, or `quit` to stop inquiring: g
g does not exist!
Enter a name, or `quit` to stop inquiring: quit
```


bb
name
bd
"alan"
(3,14)
"mark"
(4,3)
"Tom"
(3,14)
celebrate (today : BD)
bb. celebrate (BD)
bb. celebrate (13,4)
↳ "alan"
"Tom"
bb. celebrate (7,1)
↳ << >>
total function
vs.
partial function
① divide (x, y)
② abs_val (x)
precond
partial fun. ← ① ;; precond: y ≠ 0
precond. True
total fun. ② ;; no constraint on input value

Design Decision: Class Invariant vs. Precondition

Example 1

V1: No Precondition

```
class ACCOUNT
create
    make
feature
    balance: INTEGER

    make(init: like balance)
        ensure
            balance = init
invariant
    pos_balance: balance > 0
end
```

V2: Precondition w.r.t. Class Invariant

```
class ACCOUNT
create make
feature
    balance: INTEGER

    make(init: like balance)
        require
            pos_balance: init > 0
        ensure
            balance = init
invariant
    pos_balance: balance > 0
end
```

```
acc: ACCOUNT
create acc.make (-100)
```

V1 Class Inv. Violation

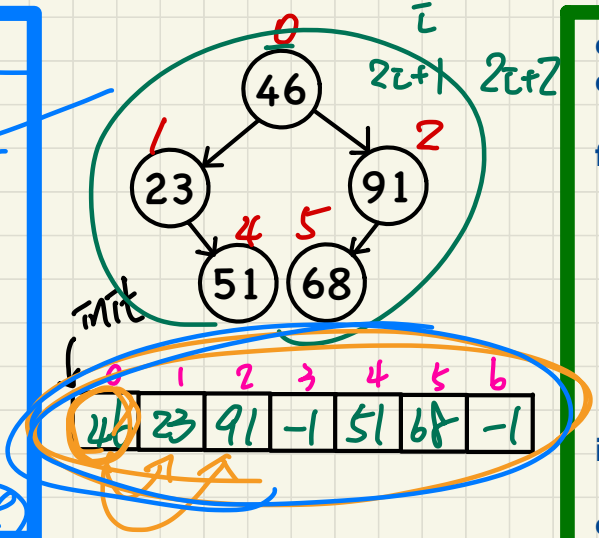
V2 Precond. violation

Design Decision: Class Invariant vs. Precondition

Example 2

V1: No Precondition

```
class BINARY_SEARCH_TREE
create
    make
feature
    rep: ARRAY[INTEGER]

    make(init: like rep)
        ensure
            rep ~ init
invariant
    valid_rep: is_bst(rep)
end
```



V2: Precondition w.r.t. Class Invariant

```
class BINARY_SEARCH_TREE
create
    make
feature
    rep: ARRAY[INTEGER]

    make(init: like rep)
        require
            valid_rep: is_bst(rep)
        ensure
            rep ~ init
invariant
    valid_rep: is_bst(rep)
end
```

```
bst: BINARY_SEARCH_TREE
create bst.make (init)
```

V1. Class Inv. Violation

V2. Precond Violation.

Conversion between ∀ and ∃

$\forall x \mid R(x) \cdot P(x) \equiv \neg(\exists x \mid R(x) \cdot \neg P(x))$

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

such that | it's the case ·

$\forall x \mid x \in S \cdot P(x) \Rightarrow \forall x \cdot x \in S \Rightarrow P(x)$

e.g. $(\forall x \mid 1 \le x \le 10 \cdot x^2 \le 100) \equiv \neg(\exists x \mid 1 \le x \le 10 \cdot x^2 > 100)$

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

$\exists x \mid R(x) \cdot P(x) \equiv \exists x \cdot R(x) \wedge P(x)$

Proof:

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

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

$\equiv \neg(\exists x \cdot \neg(R(x) \Rightarrow P(x)))$

$\equiv \neg(\exists x \cdot R(x) \wedge \neg P(x))$

$\equiv \neg(\exists x \mid R(x) \cdot \neg P(x))$

def. of $\Rightarrow$: $\neg(R(x) \Rightarrow P(x)) \equiv \neg(\neg R(x) \vee P(x)) \equiv R(x) \wedge \neg P(x)$

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

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

$$\forall x \cdot P(x)$$

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

$$\exists x \mid R(x) \cdot P(x)$$

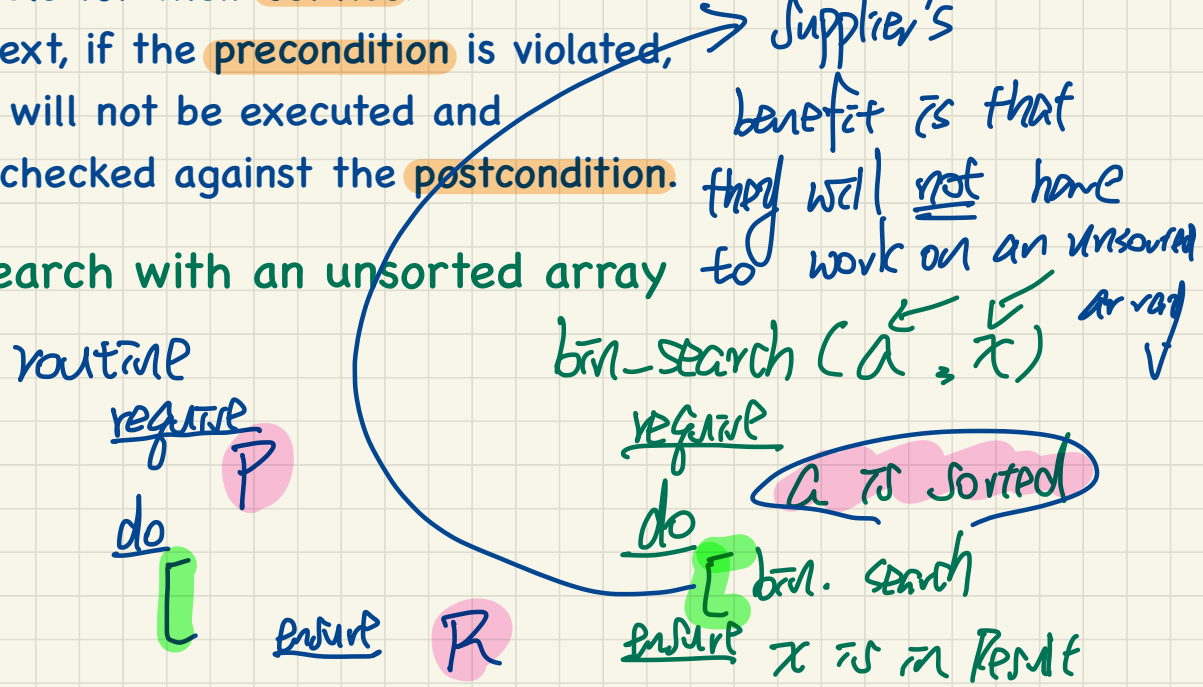
$$\equiv \exists x \cdot R(x) \wedge P(x)$$

$\forall x \in \{x \mid 1 <= x <= 10\} \mid \cdots$

How the assumption of instructions being followed is the benefit for the supplier?

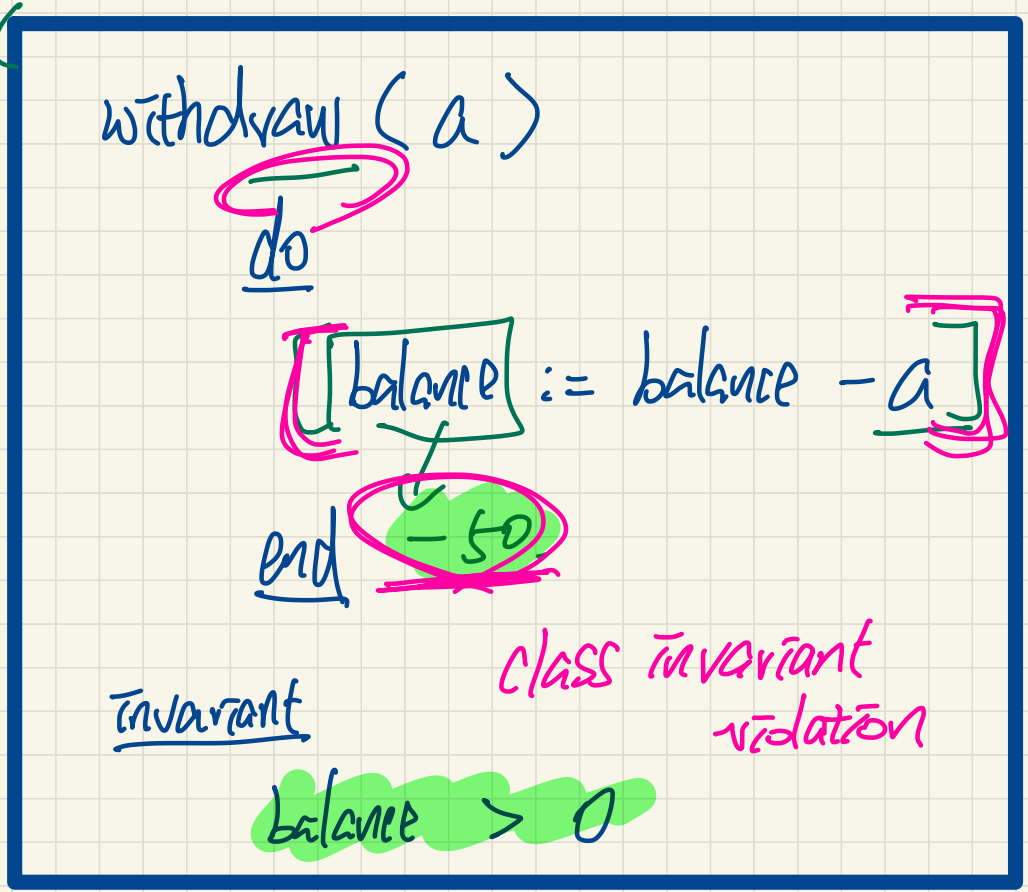
- If the client's obligation is not fulfilled, the contract is breached, so the supplier will not be held responsible for their service.
- In a programming context, if the precondition is violated, then the implementation will not be executed and the supplier will not be checked against the postcondition.

e.g., calling binary search with an unsorted array



create acc - make (100).

acc. withdraw (150)



deferred class vs. effective class

partially implemented

all routines are implemented

(Java) abstract class normal class