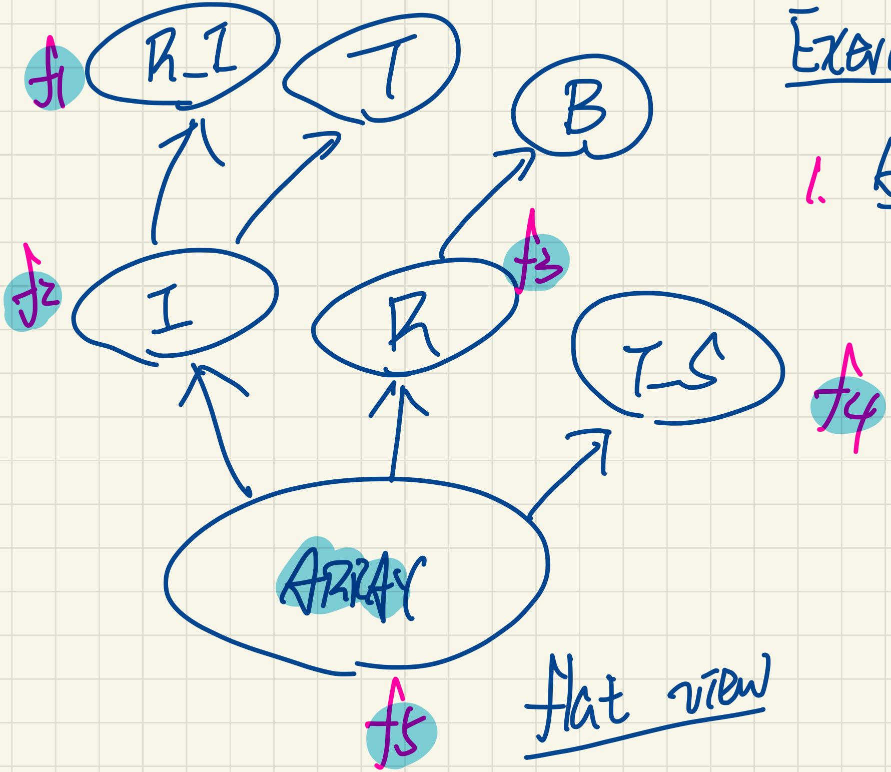
EECS 3311
Software Design

Introductory Tutorial
Blackboard Notes
Jackie Wang



Exercise.

1. Ancestors & Descendants of LIST

2. LIST v;

v = new [ ] ;

dynamic type

Declarations of Variables and Return Values

∈

i : INTEGER → the set of 32-bit int values.

P.T.

i ∈ INTEGER

P : PERSON → the set of addresses of PERSON objects.

∈

stores an address of some PERSON object.

get_absolute_value(x : INTEGER) : INTEGER

∈

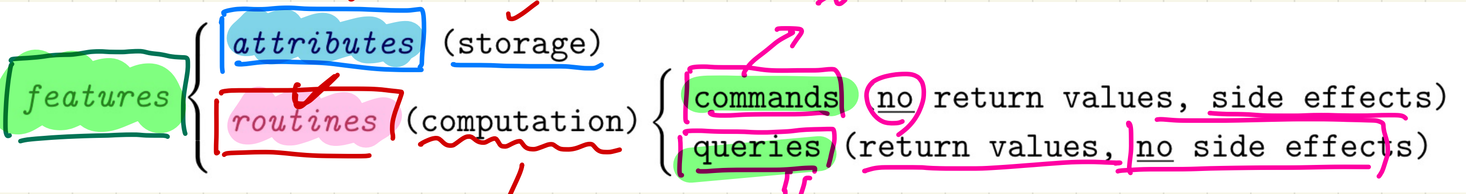
get_spouse : PERSON

Java

P getSpouse()

get_spouse : PERSON

Taxonomy of Eiffel Features



features

- *attributes* (storage)
- *routines* (computation)
  - commands (no return values, side effects)
  - queries (return values, no side effects)

≈ mutator / setter

implementation

accessor/getter

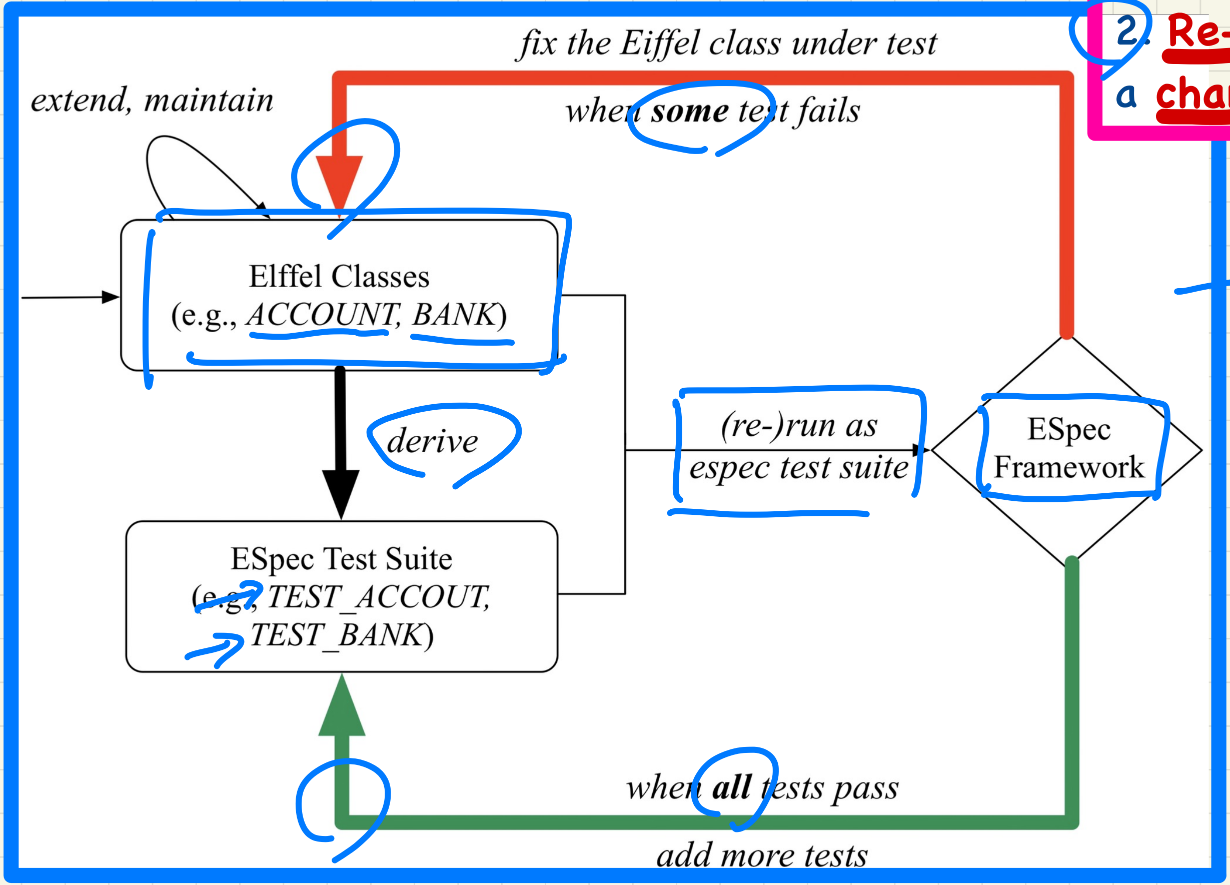
Logic Operations

assignment: $\boxed{:=}$

Java $\rightarrow$ $\boxed{=}$

Math.	Eiffel	Java
$\wedge$	and	&&
$\vee$	or	||
$\Rightarrow$	b1 implies b2	!b1 || b2
$\Leftrightarrow$	=	==
$\neg$	not	!

Test-Driven Development (TDD)

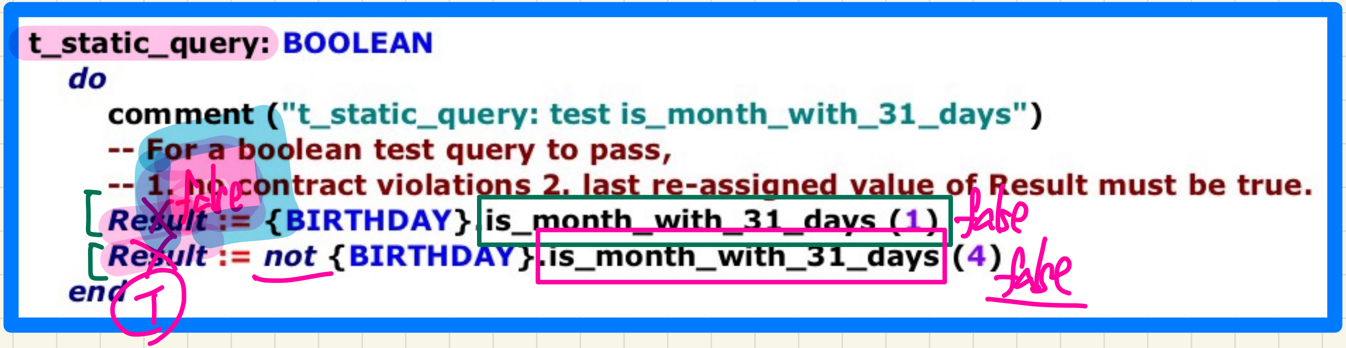


1. Test as soon as a feature becomes executable.

2. Re-run all tests when a change is made.

regression

Checking Multiple Cases in a Boolean Case

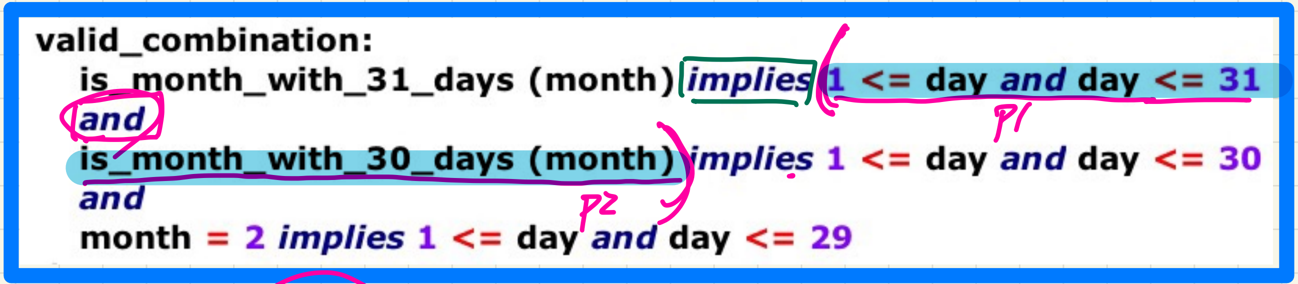


```
t_static_query: BOOLEAN
  do
    comment ("t_static_query: test is_month_with_31_days")
    -- For a boolean test query to pass,
    -- 1. no contract violations 2. last re-assigned value of Result must be true.
    Result := {BIRTHDAY}.is_month_with_31_days (1)
    Result := not {BIRTHDAY}.is_month_with_31_days (4)
  end
```

Hypothetically:

is_month_with_31_days always returs **false**

Precedence of Logical Operators



```
valid_combination:
    is_month_with_31_days (month) implies 1 <= day and day <= 31
    and
    is_month_with_30_days (month) implies 1 <= day and day <= 30
    and
    month = 2 implies 1 <= day and day <= 29
```

tightest

and

or

implies

=

loosest

$p1 \text{ or } (p2 \text{ and } p3)$

```
invariant
    inv : p1 and p2 and p3
```

```
invariant
    inv_1 : p1
    inv_2 : p2
    inv_3 : p3
```

Eiffel Classes: Syntax Overview

```
class SOME_CLASS
create
  -- Explicitly list here commands used as constructors
feature -- Attributes
  -- Declare attribute here
feature -- Commands
  -- Declare commands (mutators) here
feature -- Queries
  -- Declare queries (accessors) here
invariant
  -- List of tagged boolean expressions for class invariants
end
```

Eiffel Routines: Syntax Overview

Command

```
some_command (x: SOME_TYPE_1; y: SOME_TYPE_2)
    -- Description of the command.
  require
    -- List of tagged boolean expressions for preconditions
  local
    -- List of local variable declarations
  do
    -- List of instructions as implementation
  ensure
    -- List of tagged boolean expressions for postconditions
  end
```

=

:=

Query

```
some_query (x: SOME_TYPE_1; y: SOME_TYPE_2): SOME_RT
    -- Description of the query
  require
    -- List of tagged boolean expressions for preconditions
  local
    -- List of local variable declarations
  do
    -- List of instructions as implementation
    Result := ...
  ensure
    -- List of tagged boolean expressions for postconditions
  end
```

=

:=

Manipulate Result

Result : SOME_RT

return Result

Object Creation

Java

static type

dynamic type

```
Birthday bd = new Birthday(10, 15);
```

D.T

Eiffel

```
bd : BIRTHDAY
```

S.T

```
(1) bd := create {BIRTHDAY}.make (10, 15)
```

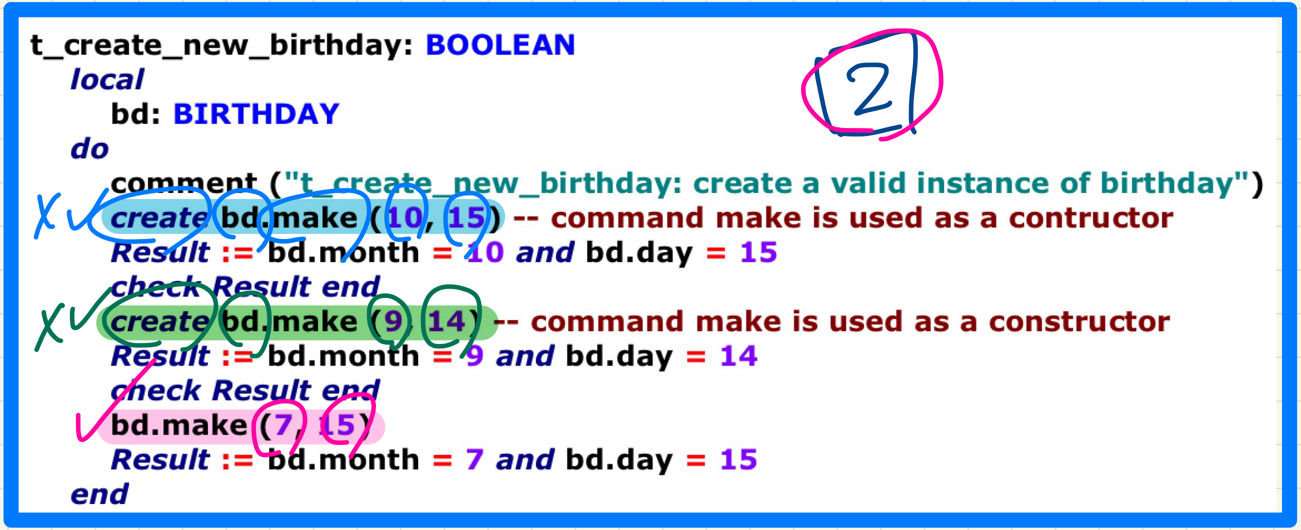
object expression
(anynomous object)

```
(2) create {BIRTHDAY} bd.make (10, 15)
```

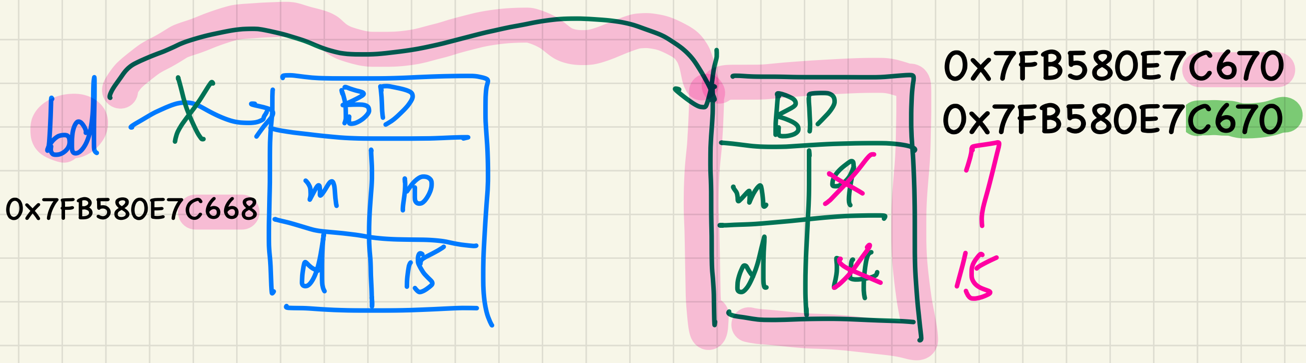
D.T

```
(3) create bd.make (10, 15)
```

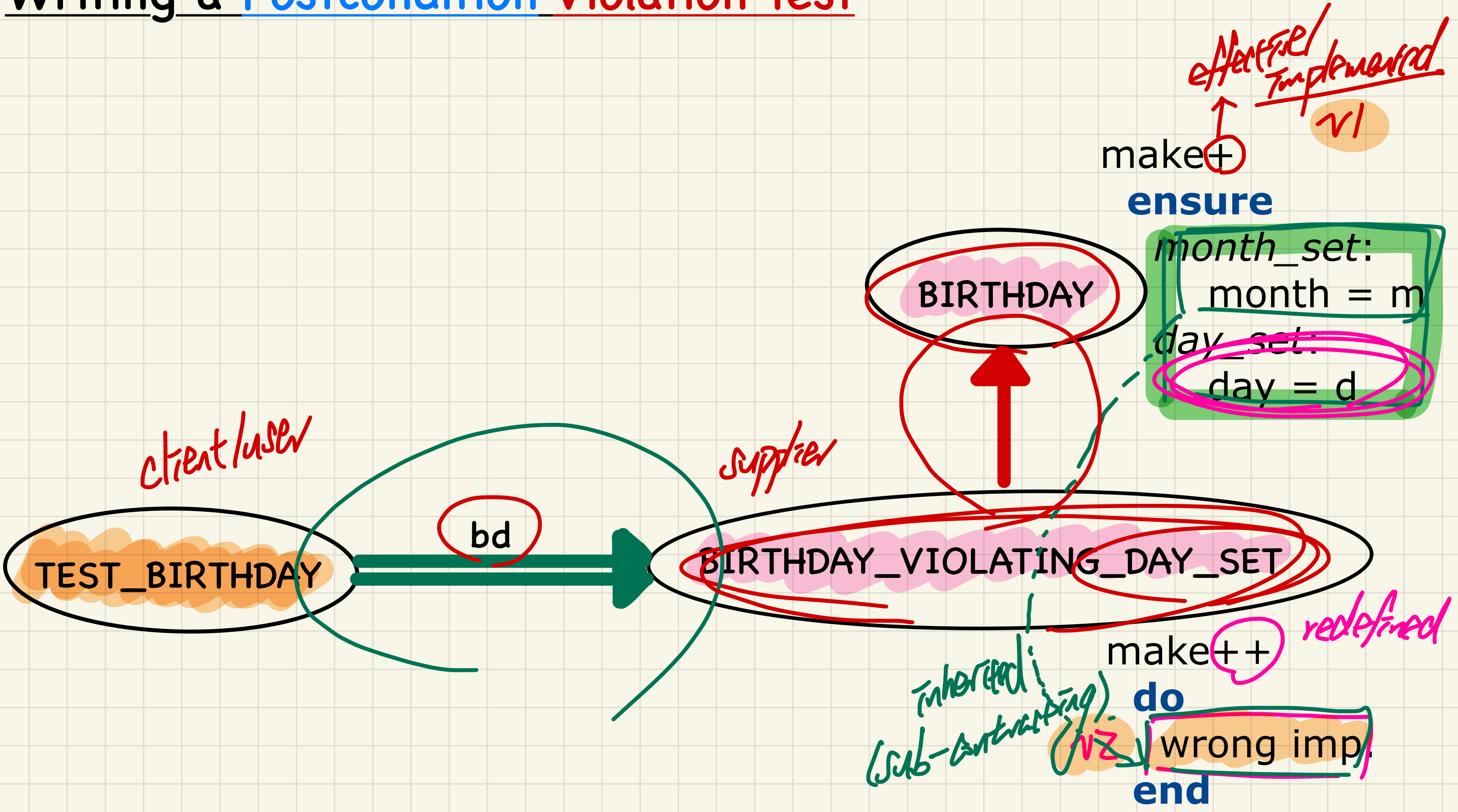
Using make as a Command vs. a Constructor



```
t_create_new_birthday: BOOLEAN
    local
        bd: BIRTHDAY
    do
        comment ("t_create_new_birthday: create a valid instance of birthday")
        create bd.make (10, 15) -- command make is used as a contructor
        Result := bd.month = 10 and bd.day = 15
        check Result end
        create bd.make (9, 14) -- command make is used as a constructor
        Result := bd.month = 9 and bd.day = 14
        check Result end
        bd.make (7, 15)
        Result := bd.month = 7 and bd.day = 15
    end
```



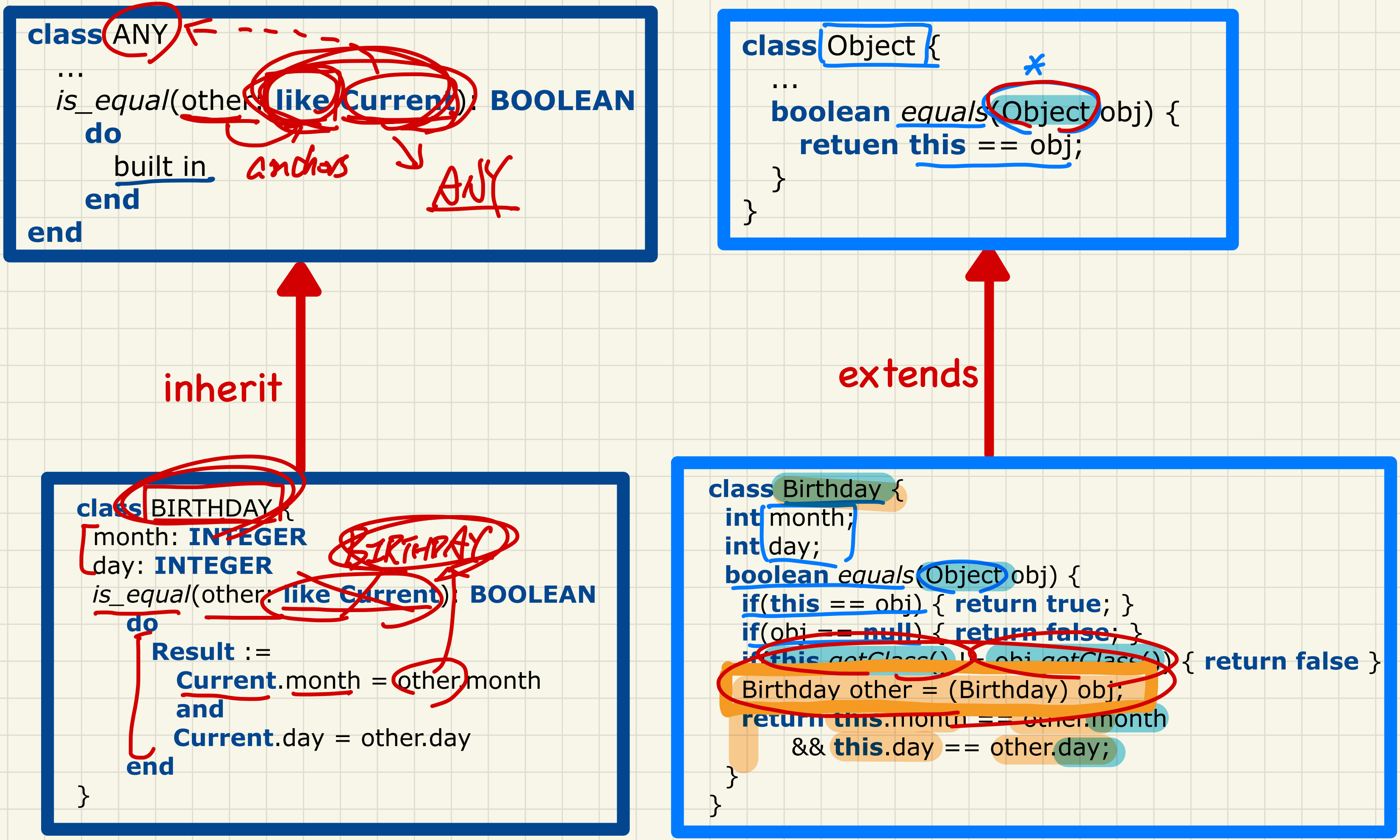
Writing a Postcondition Violation Test



```
make+
  ensure
  month_set:
    month = m
  day_set:
    day = d
```

```
make++
  do
    wrong imp.
  end
```

Object Equality: Eiffel vs. Java



```
class ANY
  ...
  is_equal(other: like Current): BOOLEAN
    do
      built in
    end
end
```

inherit

```
class BIRTHDAY {
 month: INTEGER
 day: INTEGER
 is_equal(other: like Current): BOOLEAN
   do
     Result :=
       Current.month = other.month
       and
       Current.day = other.day
   end
}
```

```
class Object {
  ...
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

extends

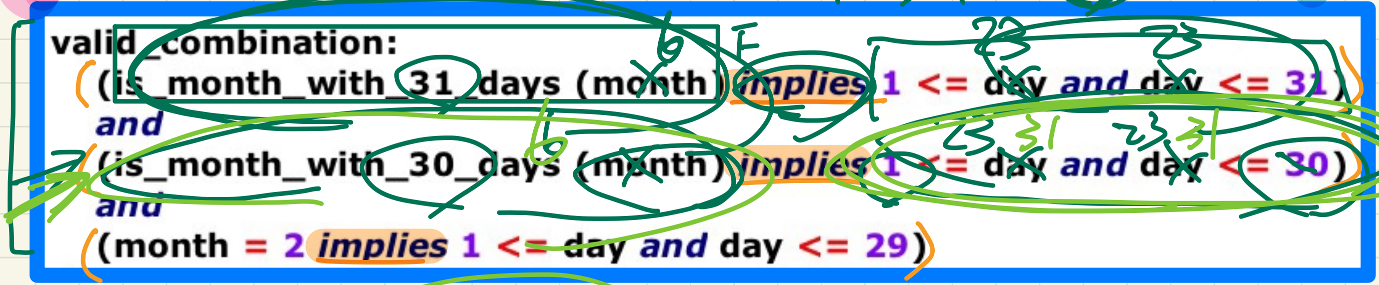
```
class Birthday {
  int month;
  int day;
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Birthday other = (Birthday) obj;
    return this.month == other.month
        && this.day == other.day;
  }
}
```

Logical Pattern: Conjunction vs. Implication

$F \Rightarrow \_ \equiv T$

$F \wedge \_ \equiv F$

$F \Rightarrow T \equiv T$



```
valid_combination:
  (is_month_with_31_days (month) implies 1 <= day and day <= 31)
  and
  (is_month_with_30_days (month) implies 1 <= day and day <= 30)
  and
  (month = 2 implies 1 <= day and day <= 29)
```

$T \Rightarrow F \equiv F$

$T \Rightarrow T \equiv T$

C1: F and T ≡ F

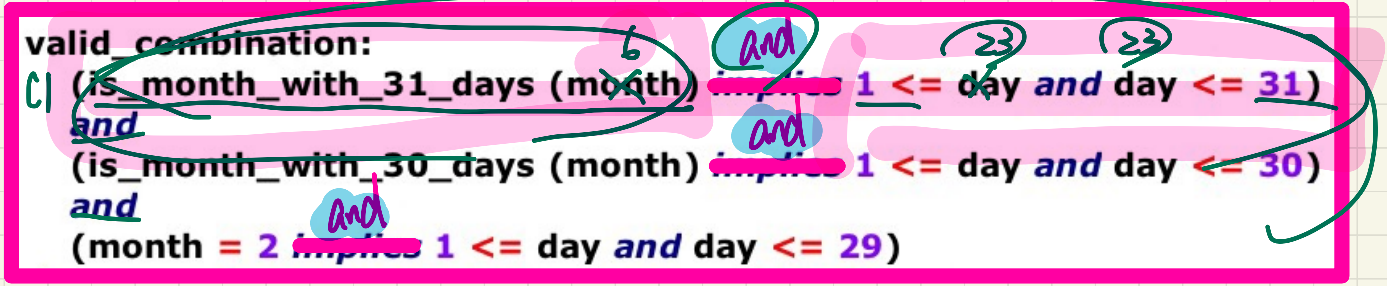
Birthday Instance

June 23

June 31

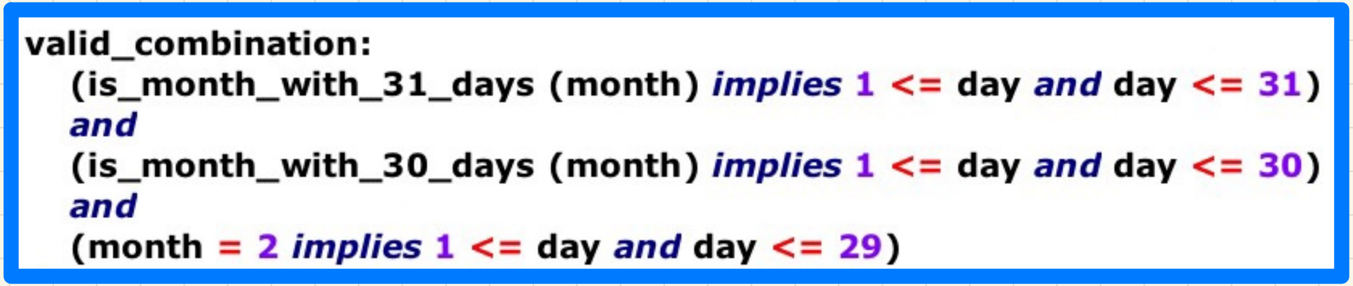
January 12

Can we change implies to and?



```
valid_combination:
  (is_month_with_31_days (month) implies 1 <= day and day <= 31)
  and
  (is_month_with_30_days (month) implies 1 <= day and day <= 30)
  and
  (month = 2 implies 1 <= day and day <= 29)
```

Logical Pattern: Conjunction vs. Implication



```
valid_combination:
    (is_month_with_31_days (month) implies 1 <= day and day <= 31)
    and
    (is_month_with_30_days (month) implies 1 <= day and day <= 30)
    and
    (month = 2 implies 1 <= day and day <= 29)
```

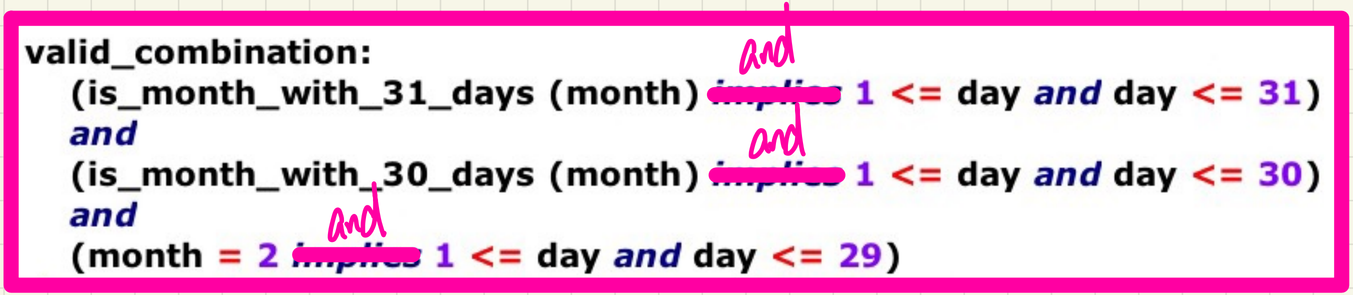
Exercise

Birthday Instance

June 23

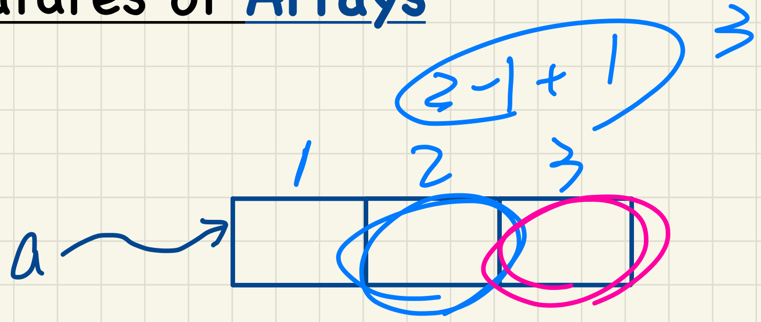
January 12

Can we change **implies** to **and**?



```
valid_combination:
    (is_month_with_31_days (month) ~~implies~~ and 1 <= day and day <= 31)
    and
    (is_month_with_30_days (month) ~~implies~~ and 1 <= day and day <= 30)
    and
    (month = 2 ~~implies~~ and 1 <= day and day <= 29)
```

Features of Arrays

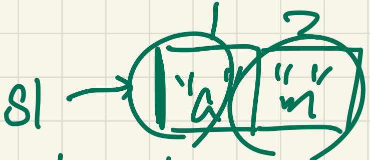


a.item(2)

a[3]

s1.force("a", s1.count+1)
0

s1.force("m", s1.count+1)
1



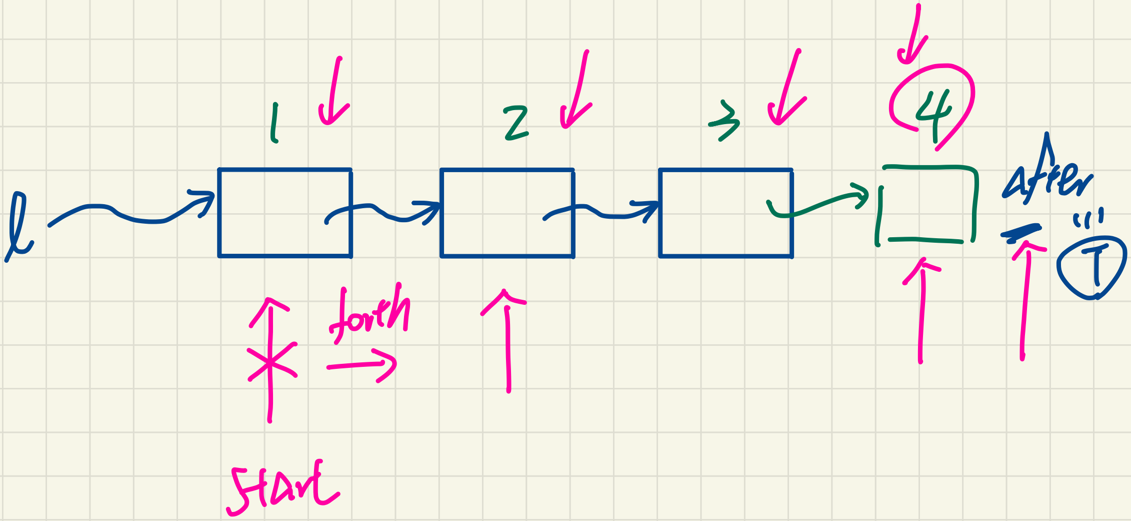
lower	1	1	1
upper	0	1	2

(2-1)+1 $\textcircled{2}$

- make_empty
- force
- lower
- upper
- count 3
- valid_index
- is_empty
- item
- indexing

count = upper - lower + 1

Features of Linked Lists



make

extend

count

valid_index

is_empty

item

indexing [_]

start

forth

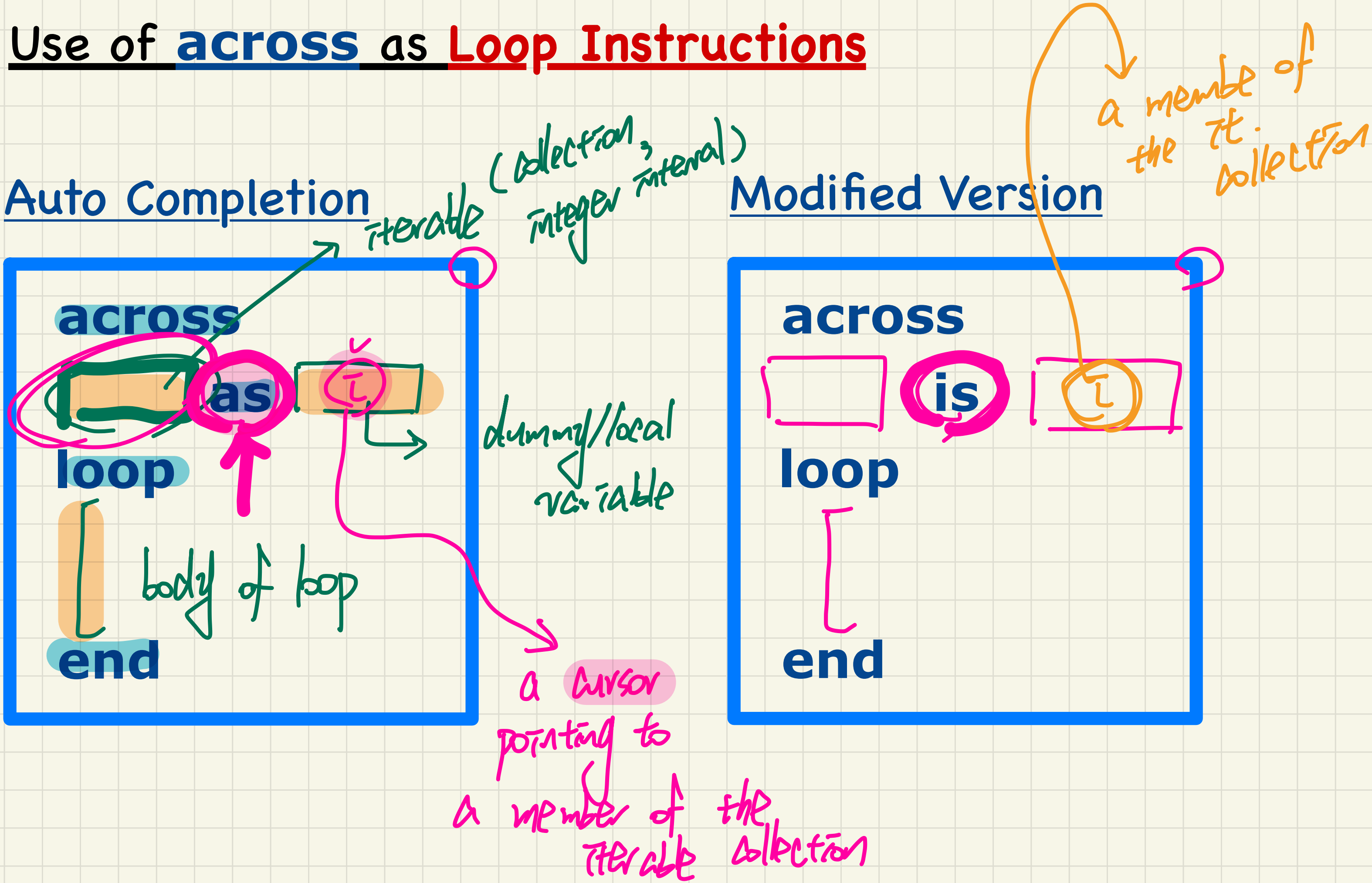
Commands

after

query

false

Use of across as Loop Instructions



Auto Completion

```
across
      as
loop

end
```

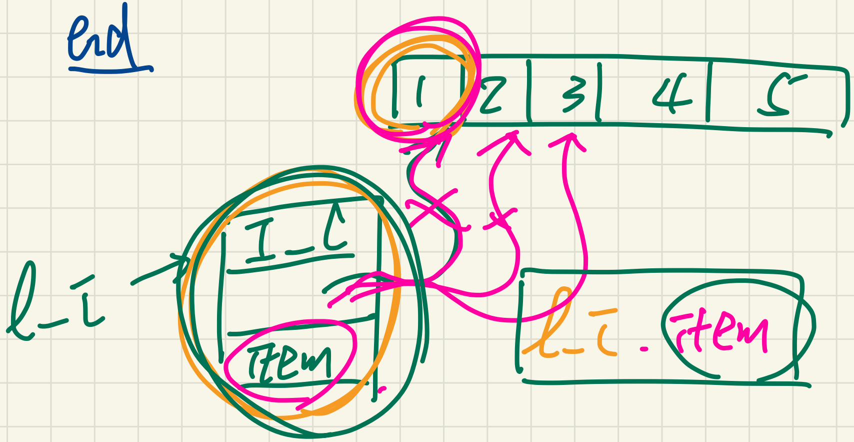
Modified Version

```
across
      is
loop

end
```

iterable (collection, integer interval)
i
dummy/local variable
body of loop
a cursor pointing to a member of the iterable collection
i
a member of the it. collection

```
across 1 |..| 5 as l_i
  loop
    :
  end
```



Use of across as Boolean Expressions

Universal Quantification ∀

```
across
      is x
all  ∀
      Boolean Expression
end
```

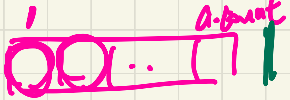
P

$\forall x \cdot P(x)$

Existential Quantification ∃

```
across
      is
some  ∃
end
```

a →



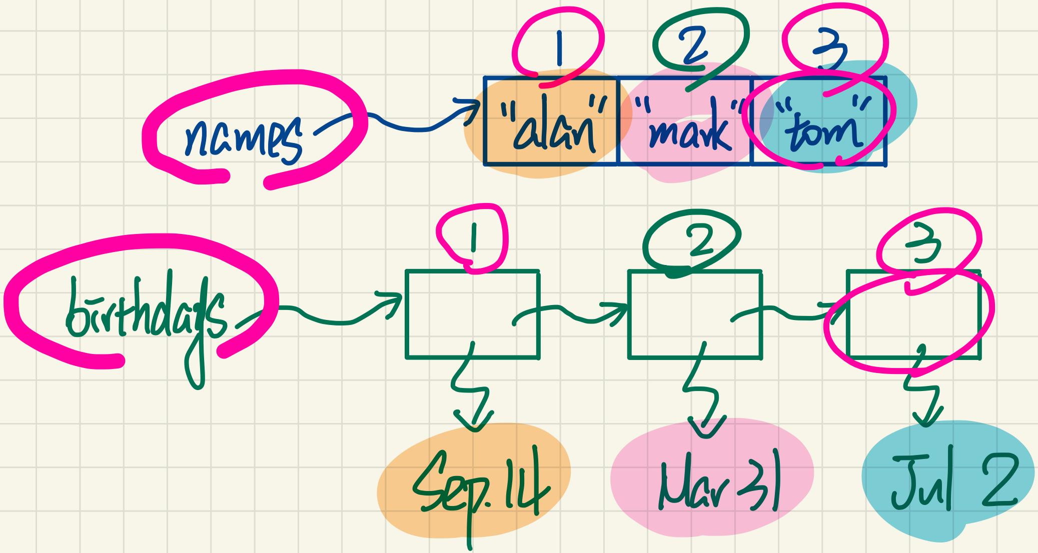
$\forall i \mid 1 \le i \le \boxed{a.count} \cdot$

a.count - 1

a.count + 1

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

Implementing a Birthday Book



Void Safety

void null

Declaration	Check for Null Pointers
Eiffel nick_name: detachable STRING name: STRING name can never be void.	Eiffel attached nick_name
Java String nickName; optional String name; required NullPointerExcep.	Java nickName != null T or F

Initialization Required?

void

detachable attached

Program from the Interface, Not from the Implementation

Declaration of Variable

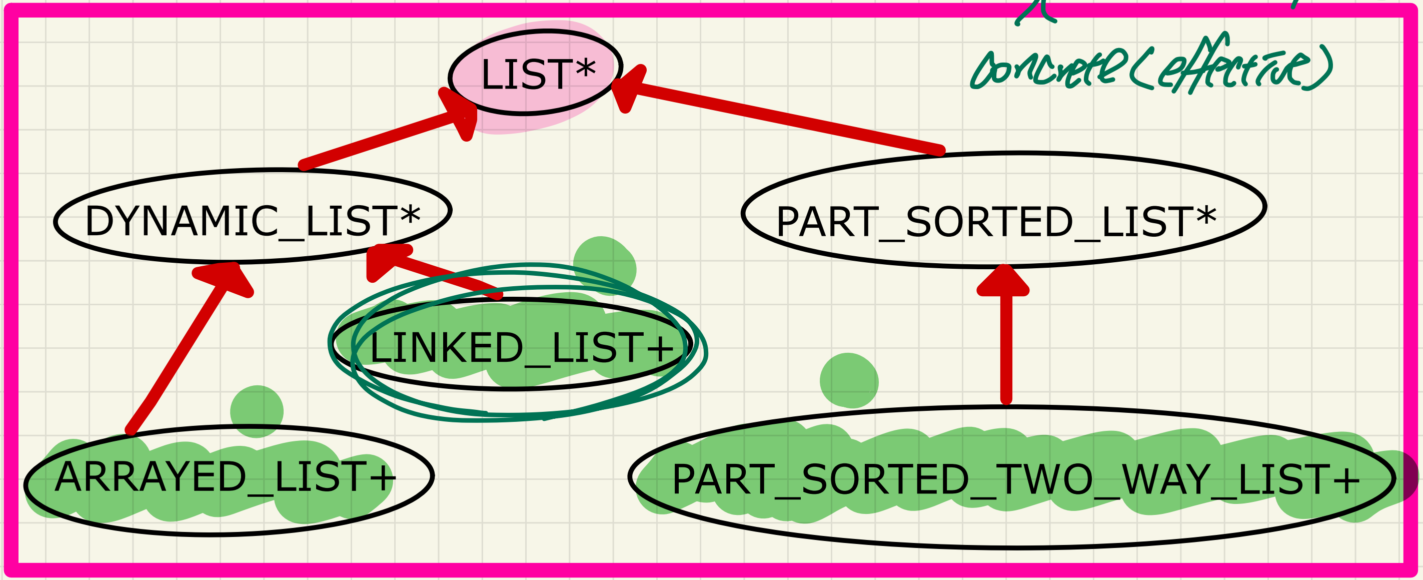
```
birthdays: LIST[BIRTHDAY]
```

static type

Creation of Object

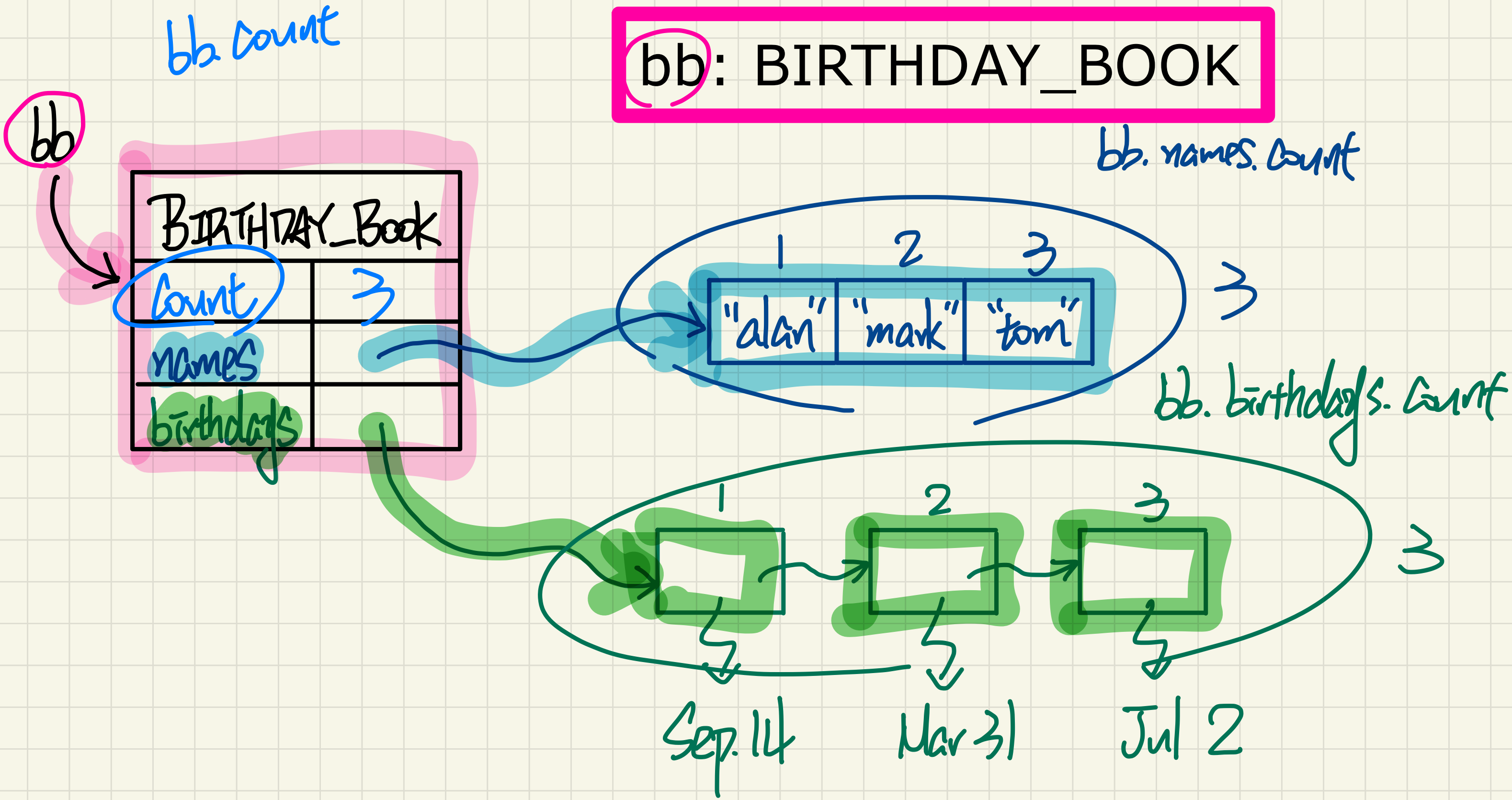
```
create {??} birthdays.make(...)
```

?? descendants of LIST



concrete (effective)

Birthday Book: Invariant



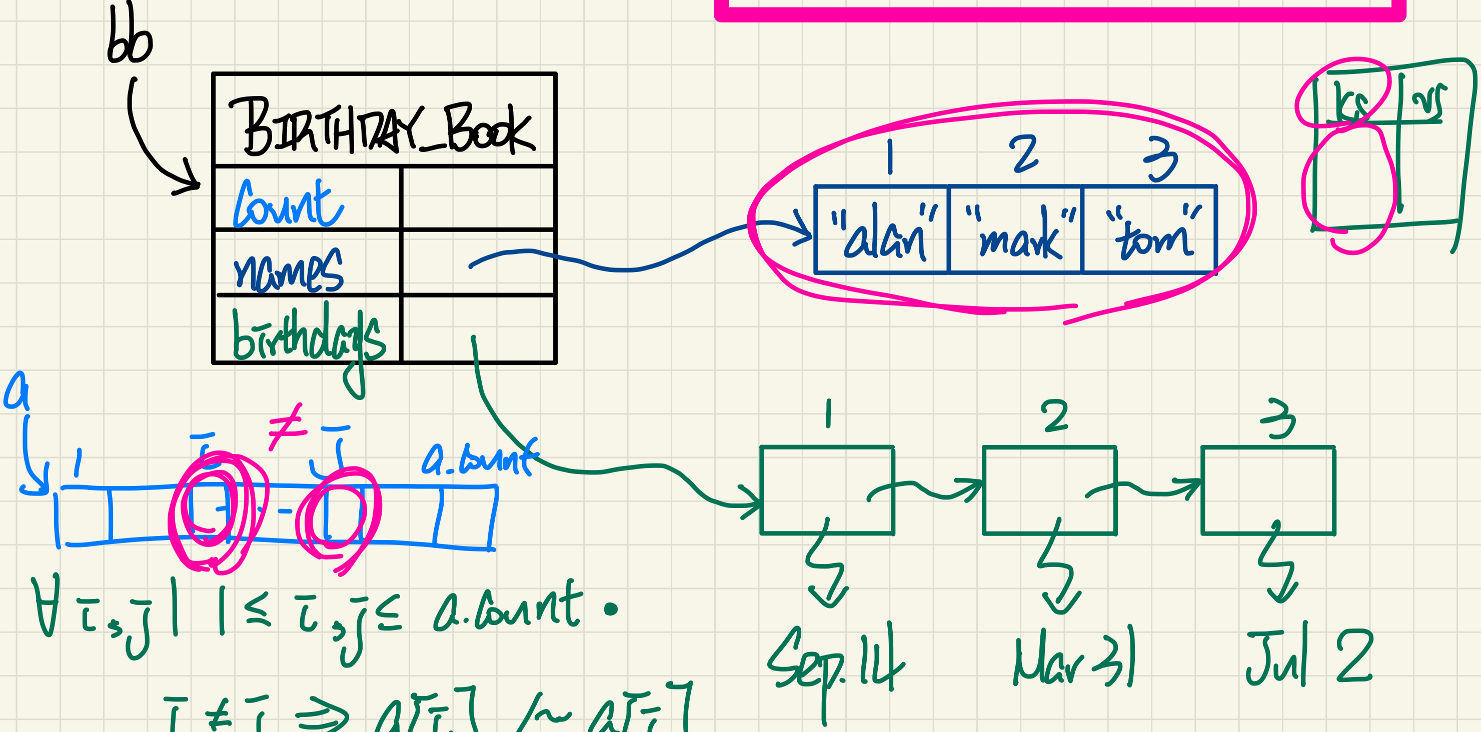
bb: BIRTHDAY_BOOK

Birthday Book: Invariant

bb("alan") → Sept.14

bb("jim") → ⊥

bb: BIRTHDAY_BOOK

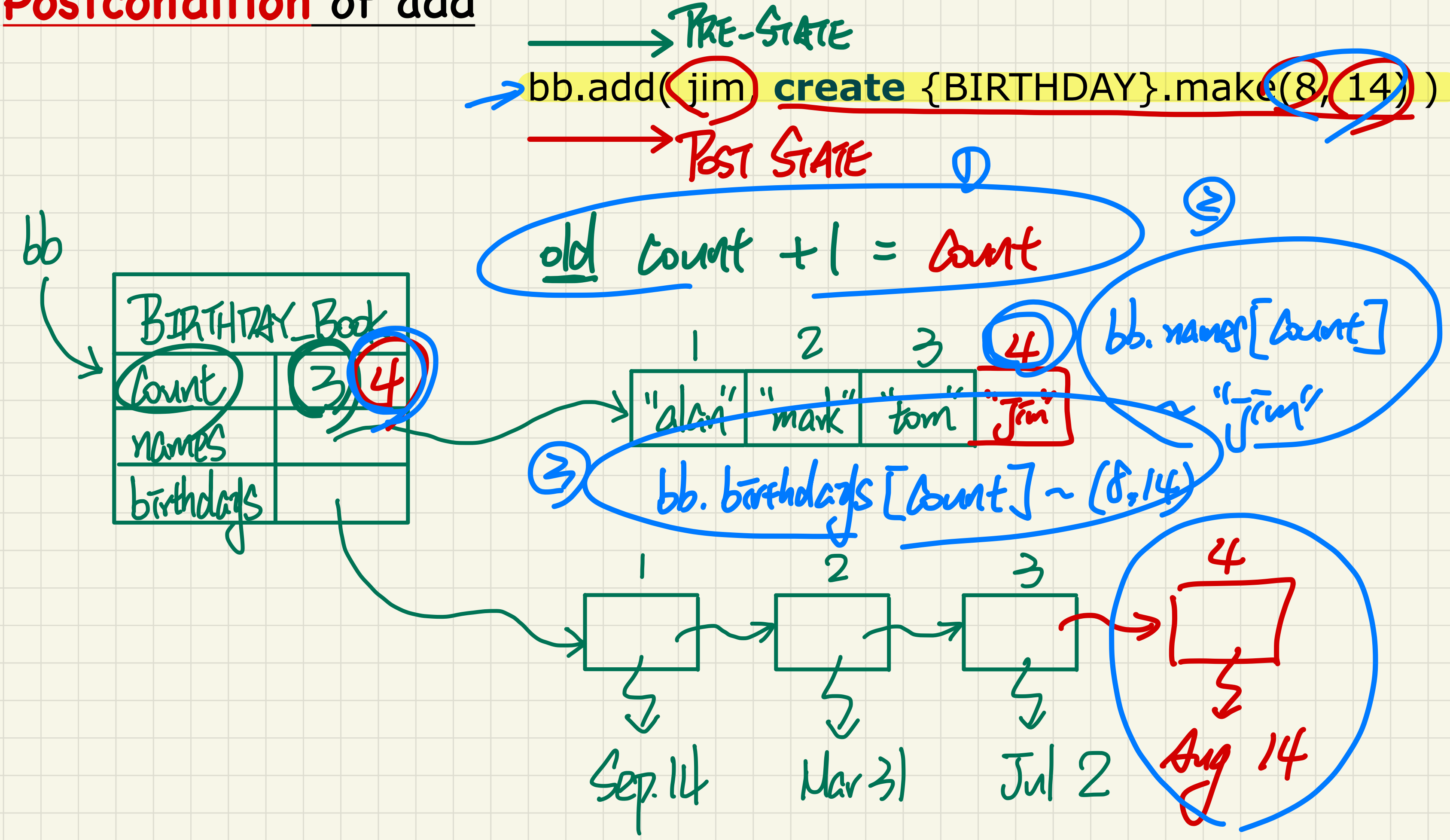


$$\forall i, j \mid 1 \le i, j \le a.count \bullet$$

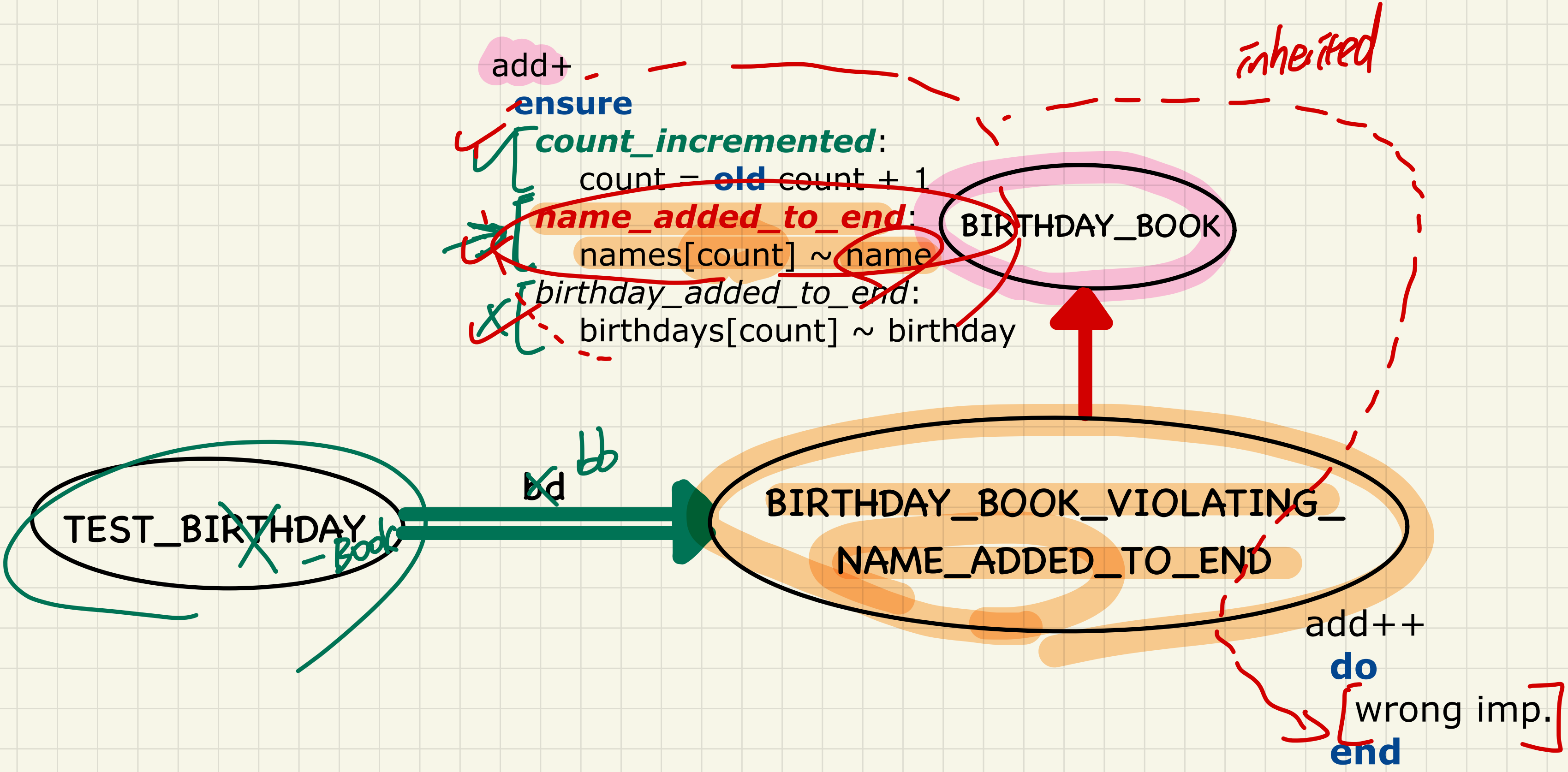
$$i \neq j \Rightarrow a[i] \nsim a[j]$$

Postcondition of add

```
bb.add(jim, create {BIRTHDAY}.make(8, 14))
```



Writing a Postcondition Violation Test



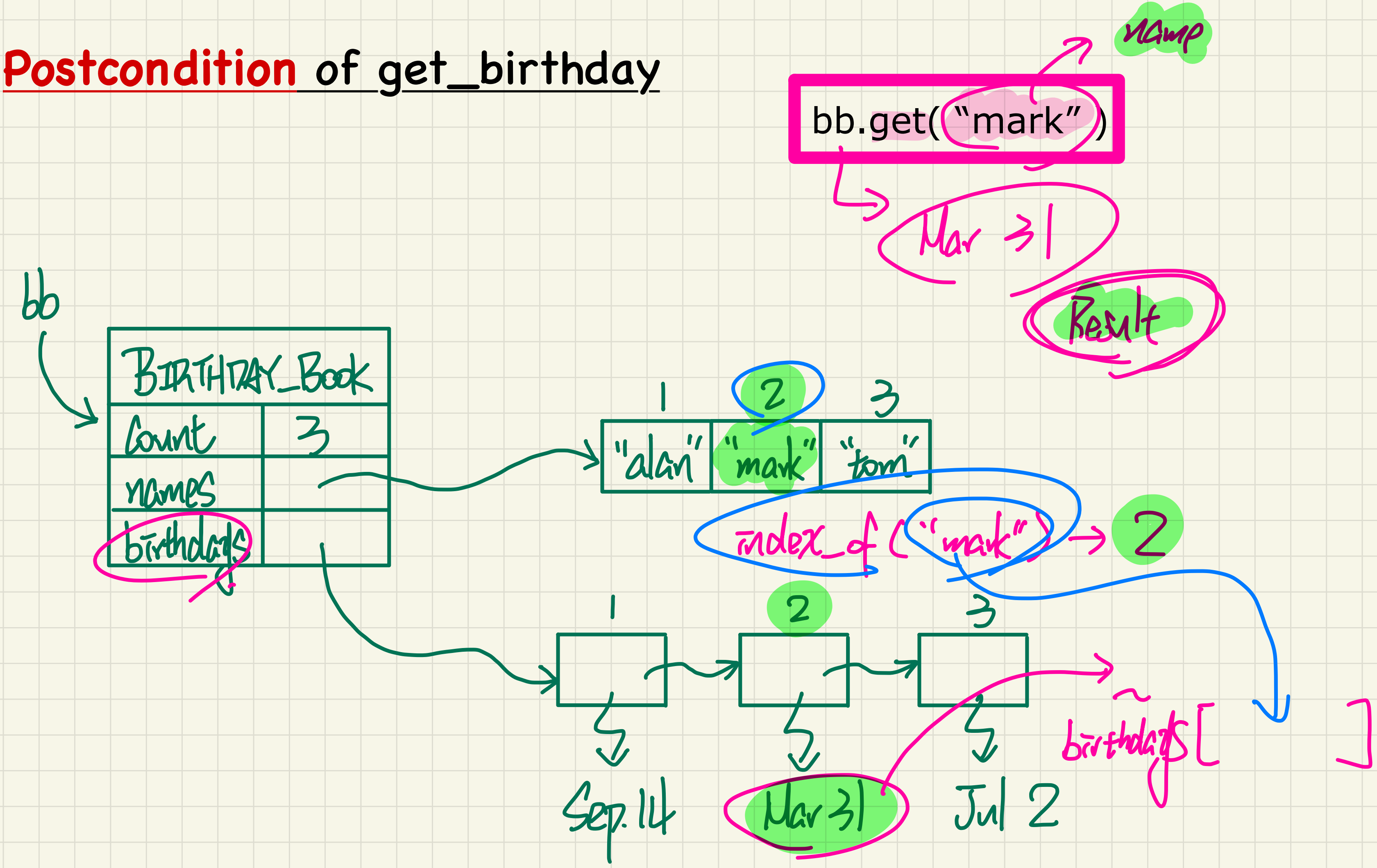
```
add+
  ensure
    count_incremented:
      count = old count + 1
    name_added_to_end:
      names[count] ~ name
    birthday_added_to_end:
      birthdays[count] ~ birthday
```

```
add++
  do
    wrong imp.
  end
```

$$\forall\, \underline{x} \mid \underbrace{R(x)}_{\text{range}} \cdot \underbrace{\underline{P(x)}}_{\text{Property}}$$

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

Postcondition of get_birthday



bb.get(“mark”)

attached Return Value: get_birthday

Supplier

```
get_birthday(n: STRING): BIRTHDAY
    require
        existing_name: names.has(n)
    do
        Result : BIRTHDAY


        return Result
    end
```

attached

Client

```
some_routine(...)
    local
        bb: BIRTHDAY_BOOK
    do
        ...
        bb.get_birthday(“yuna”).month
    end
```

attached BIRTHDAY

detachable Return Value: get_detachable_birthday

Supplier

```
get_detachable_birthday(n: STRING): detachable BIRTHDAY
  ~~do~~
    Result : detachable BIRTHDAY
    [   no initialization

    return Result
  end
```

" voidable "

Client

```
some_routine(...)
  local
    bb: BIRTHDAY_BOOK
  do
    ...
    bb.get_detachable_birthday("yuna").month
  end
```

call on void target

NPE

might be void

Boolean expr

T

F

check as then

attached

dummy var.

end

Declaration of celebrate

```
class BIRTHDAY_BOOK
...
feature
  names: ARRAY[STRING]
  birthdays: LIST[BIRTHDAY]
...
feature
  celebrate (today: BIRTHDAY): like names
    do
      ...
    end
```

TWO_WAY_SORTED_LIST

A[S]

anchor type

~~A[S]~~ T_W_S_L

f (p : like names) ~~A[S]~~ T_W_S_L

g : like names ~~A[S]~~ T_W_S_L

Result : A[S]

return result.

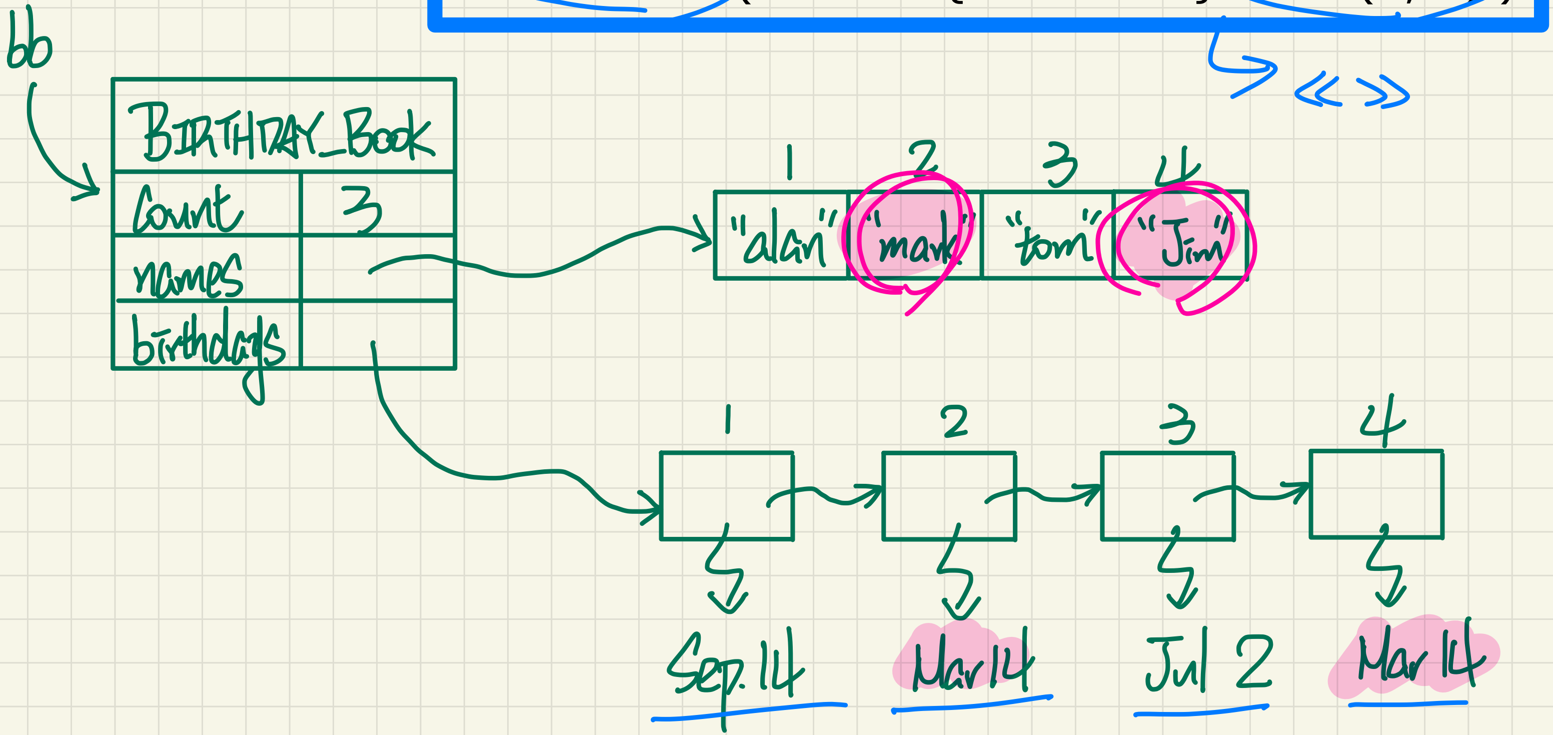
Postcondition of celebrate

<< "mark", "Jim" >>

```
bb.celebrate( create {BIRTHDAY}.make(3, 14) )
```

```
bb.celebrate( create {BIRTHDAY}.make(8, 7) )
```

<< >>



bb
BIRTHDAY_Book
Count
3
names
birthdays
1
2
3
4
"alan"
"mark"
"tom"
"Jim"
1
2
3
4
Sep. 14
Mar 14
Jul 2
Mar 14