EECS3311

SOFTWARE DESIGN

SECTION A FALL 2019

INSTRUCTOR:
JACKIE WANG

LECTURE 1

THURSDAY SEPTEMBER 5

COURSE LEARNING OUTCOMES

EECS1090

CLO1 Describe software specifications via Design by Contract, including the use of preconditions, postconditions, class invariants, as well as loop variants and invariants.

CLO2 Implement specifications with designs that are correct, efficient, and maintainable.

CLO3 Develop systematic approaches to organizing, writing, testing, and debugging software.

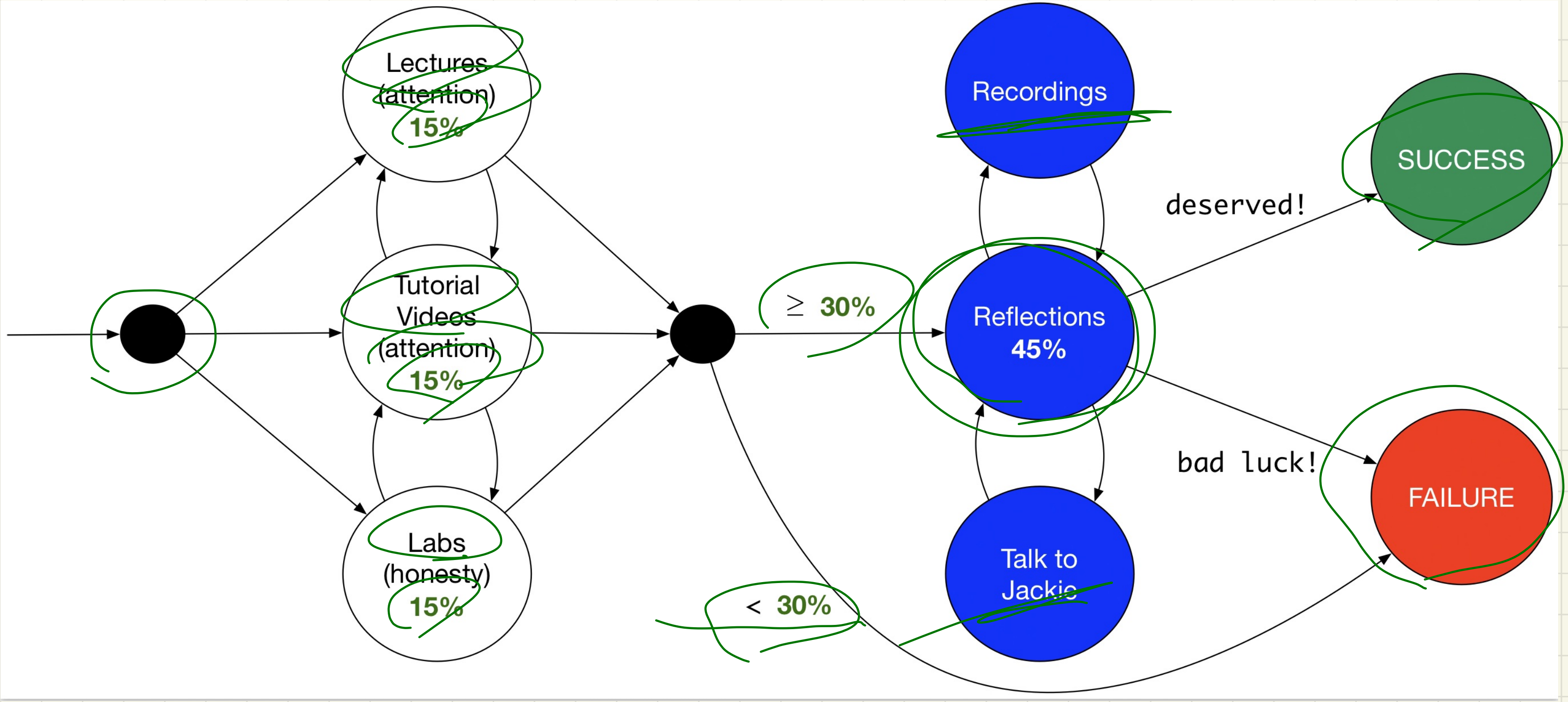
CLO4 Develop insight into the process of moving from an ambiguous problem statement to a well-designed solution.

CLO5 Design software using appropriate abstractions, modularity, information hiding, and design patterns.

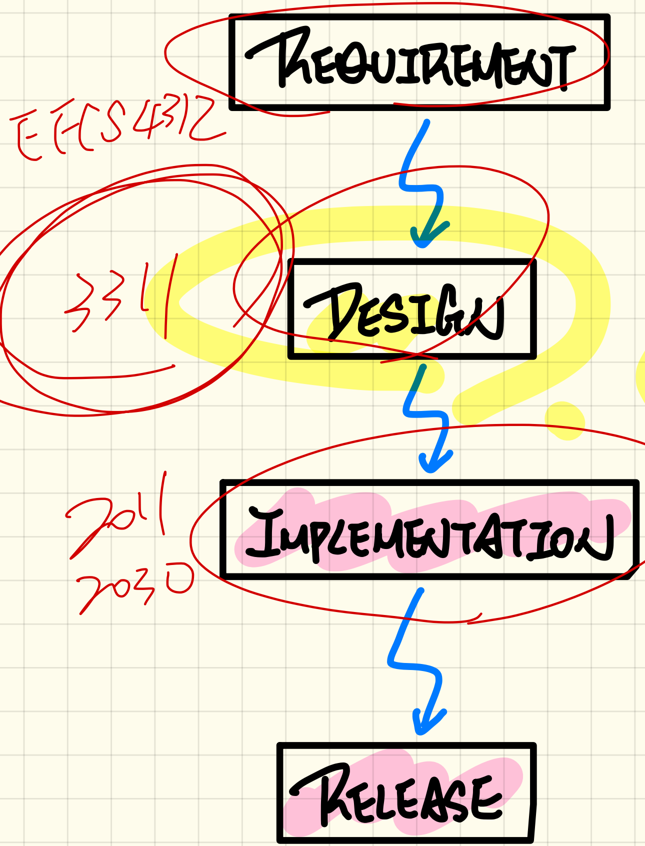
CLO6 Develop facility in the use of an IDE for editing, organizing, writing, debugging, documenting designs, and the ability to deploy the software in an executable form.

CLO7 Write precise and concise software documentation that also describes the design decisions and why they were made.

SURVIVING THROUGH THIS COURSE



SOFTWARE DEVELOPMENT CYCLE



- Natural Language
  (incomplete, ambiguous, contradicting)

working payment system

easy to use
and
4-phase authentication
(face, touch, code, pass)

web interface
and
deployable on mobile devices

- elicitation

- blueprints
- not necessarily executable & testable

- API given
- efficiency (data structures & algorithms)
- unit tests

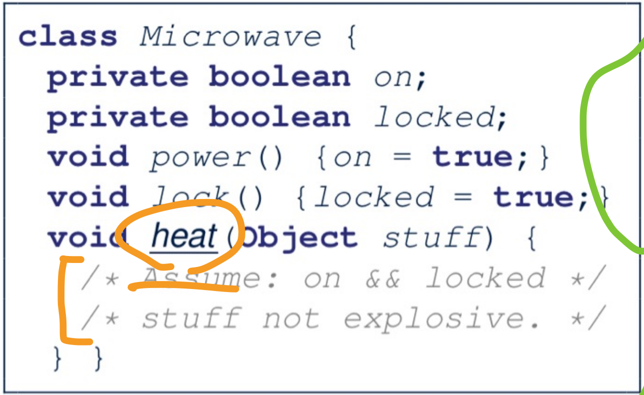
- Customer's acceptance?
- return?

- Unit testing
  - JUnit in Java
  - ESpec in Eiffel
  - API of methods
  - imp. details of methods.

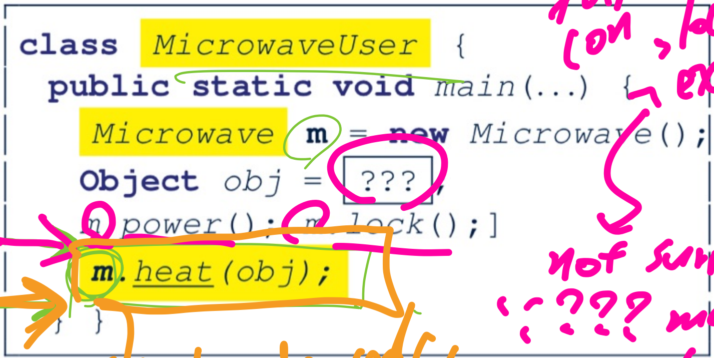
- Acceptance testing
  - aggreed interface with the customers.
  - imp. details hidden
  - design. details

Client vs. Supplier in OOP

pre-state → client's ob. has to be fulfilled (on, locked, expl)



```
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. */
} }
```



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

not sure "???" may be explosive

obj.heated()

Supplier m (Microwave)

Client this (MicrowaveUser)

post-state → supplier's ob. fulfilled

Pre-Condition:
a is sorted.
O(n)

```
binarySearch(int[] a) {
  [
  ]
}
```

O(log n)

inefficient
✓ only meant to be checked during development.

Turn all contracts off when finalizing.

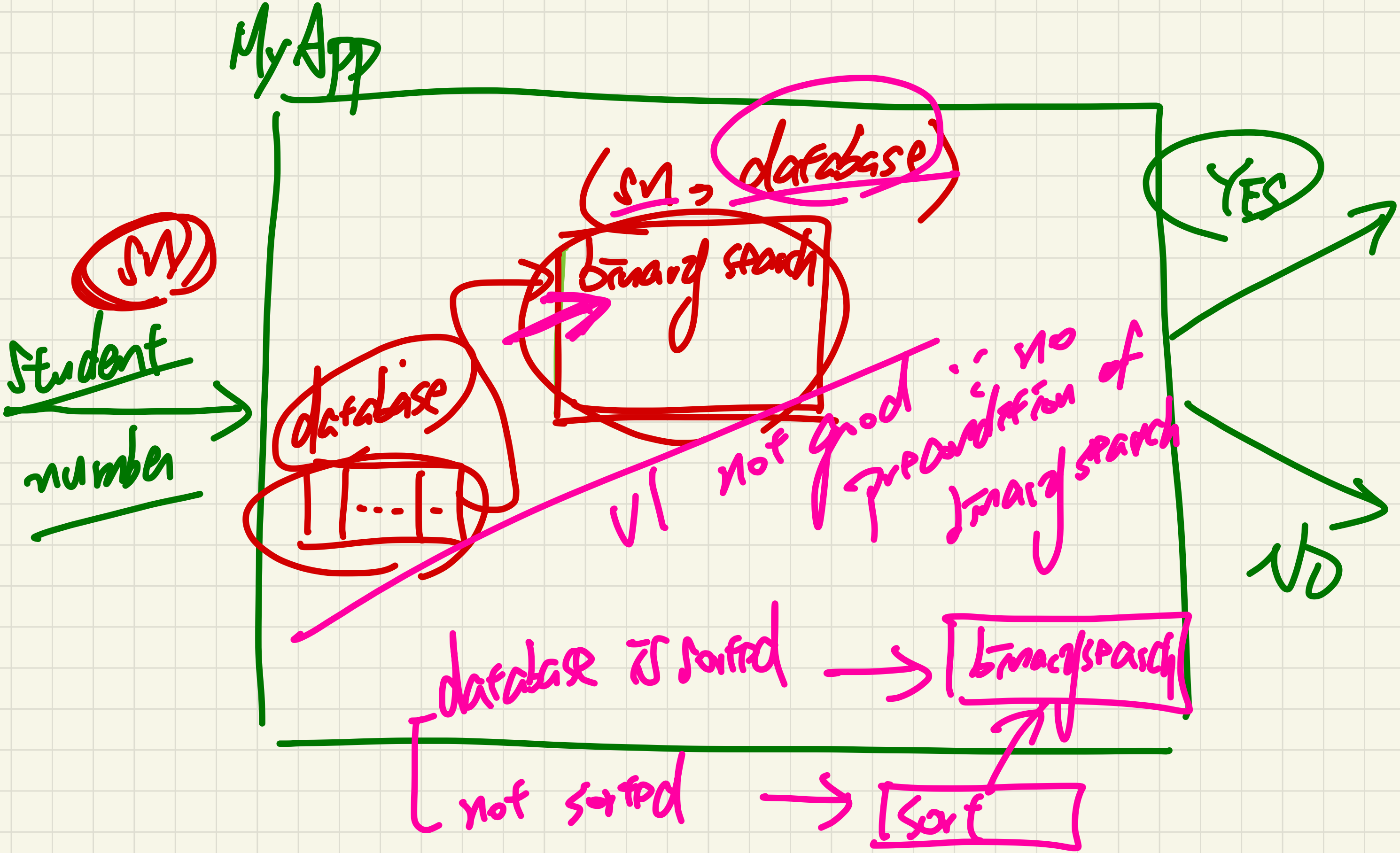
application

obj1.m1(...);

obj2.m2(...);

⋮

objn.mn(...);


MyApp
SN
student number
database
binary search
SN → database
YES
NO
not good ∵ no precondition of binary search
database is sorted → binary search
not sorted → sort

LECTURE 2

TUESDAY SEPTEMBER 10

- Waiting list ?
- Lab 1
- office hours : 4 ~ 6

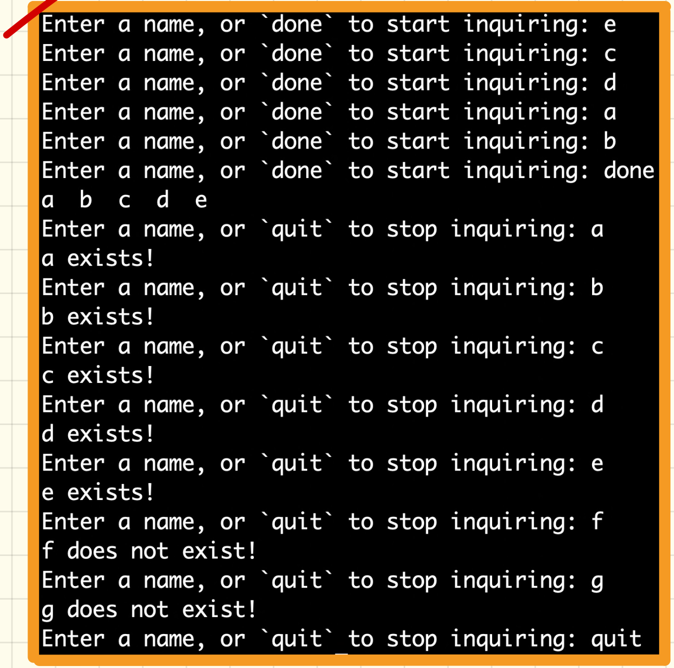
  MON TUE WED
- tomorrow's lab session : syntax demo

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

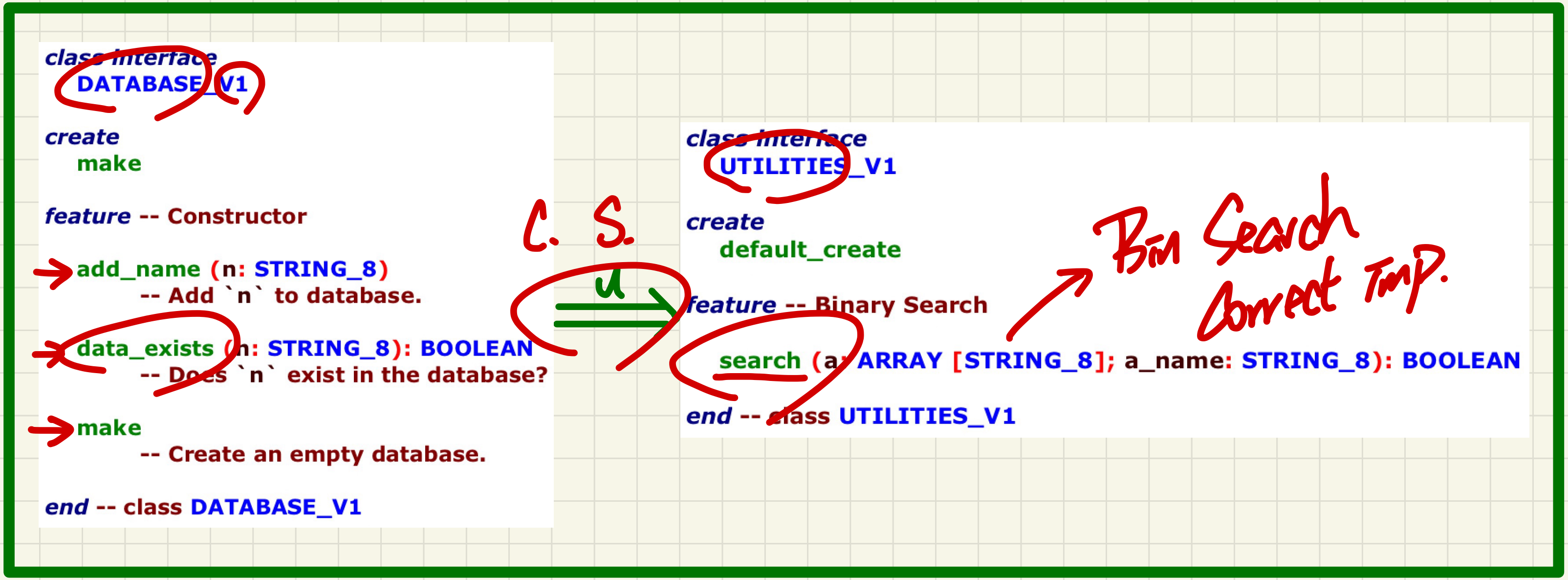


```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
a  b  c  d  e
Enter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

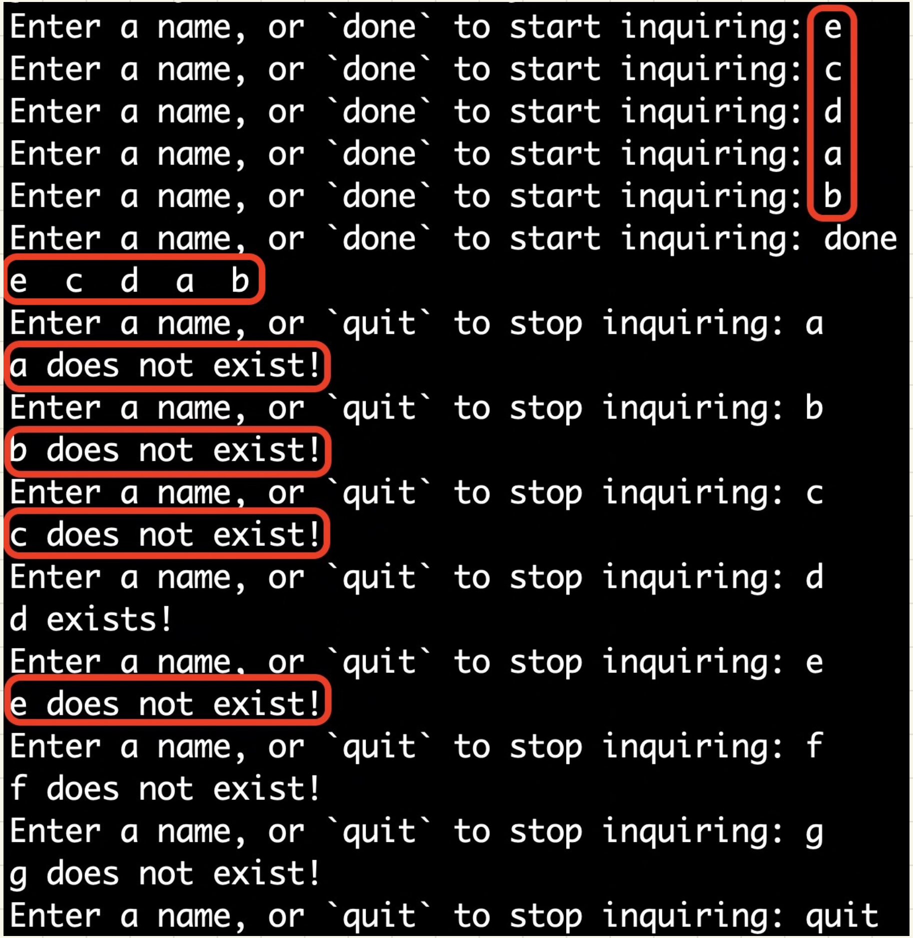


```
class interface
  DATABASE_V1

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_V1
```

```
class interface
  UTILITIES_V1

create
  default_create

feature -- Binary Search

  search (a: ARRAY [STRING_8]; a_name: STRING_8): BOOLEAN

end -- class UTILITIES_V1
```

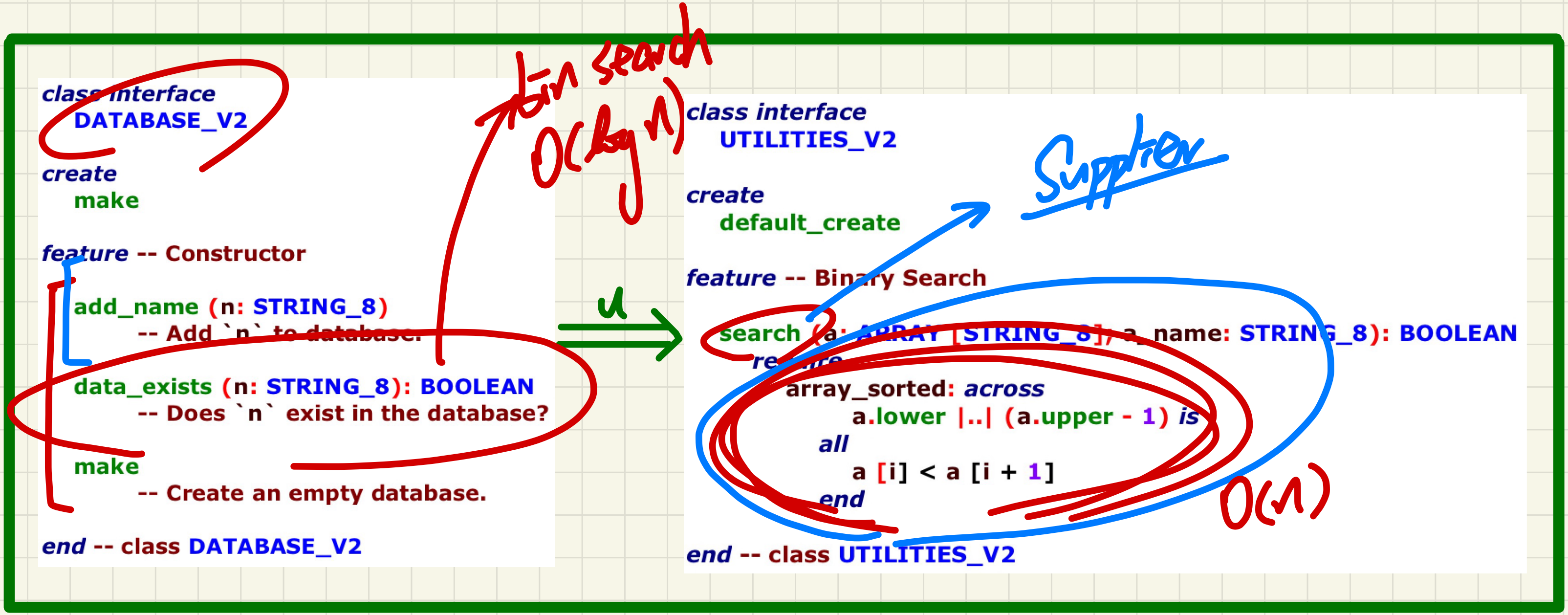
- Data array in DATABASE is not kept sorted.
- Binary search in UTILITIES does not require a sorted input array.
- When user enters names in an unsorted order, output is wrong.
- But no contract violation!
- A bad design is when something goes wrong, there is no party to blame.

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
```

Version 2: Wrong Implementation, Proper Precondition

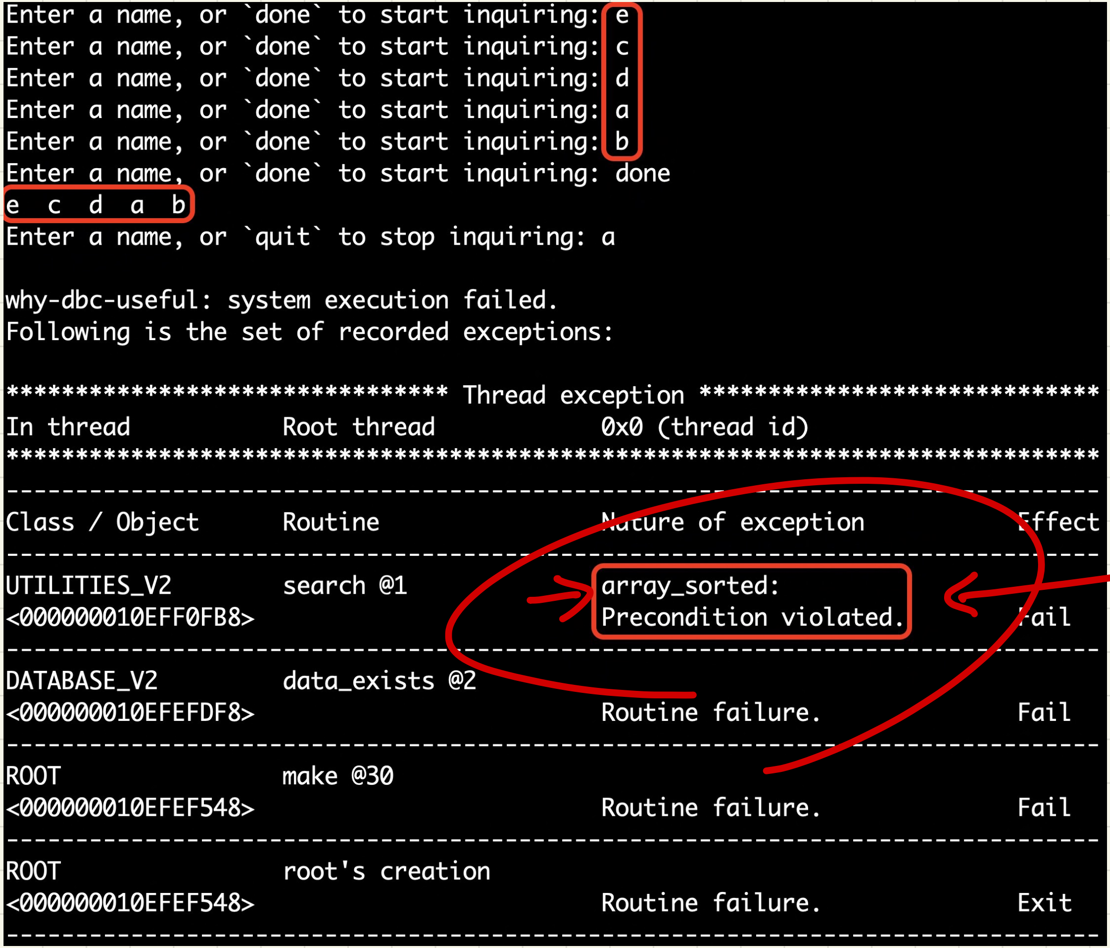


```
class interface
    DATABASE_V2

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
                all
                    a [i] < a [i + 1]
                end

end -- class UTILITIES_V2
```

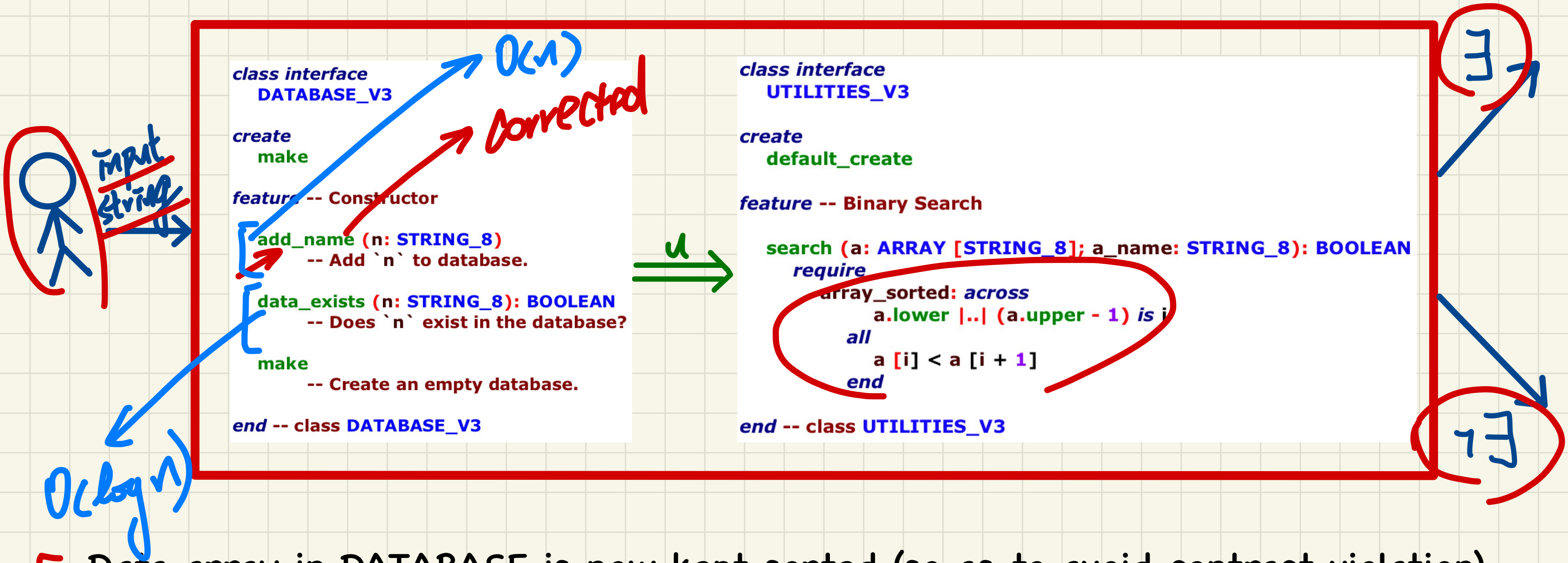
- Data array in DATABASE is not kept sorted.
- Binary search in UTILITIES now requires a sorted input array.
- When an unsorted array is passed for search, a contract violation occurs!
- A good design is when something goes wrong, there is one party to blame.

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              array_sorted:
<000000010EFF0FB8>                         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
-------------------------------------------------------------------------------
```

Version 3: Fixed Implementation, Proper Precondition

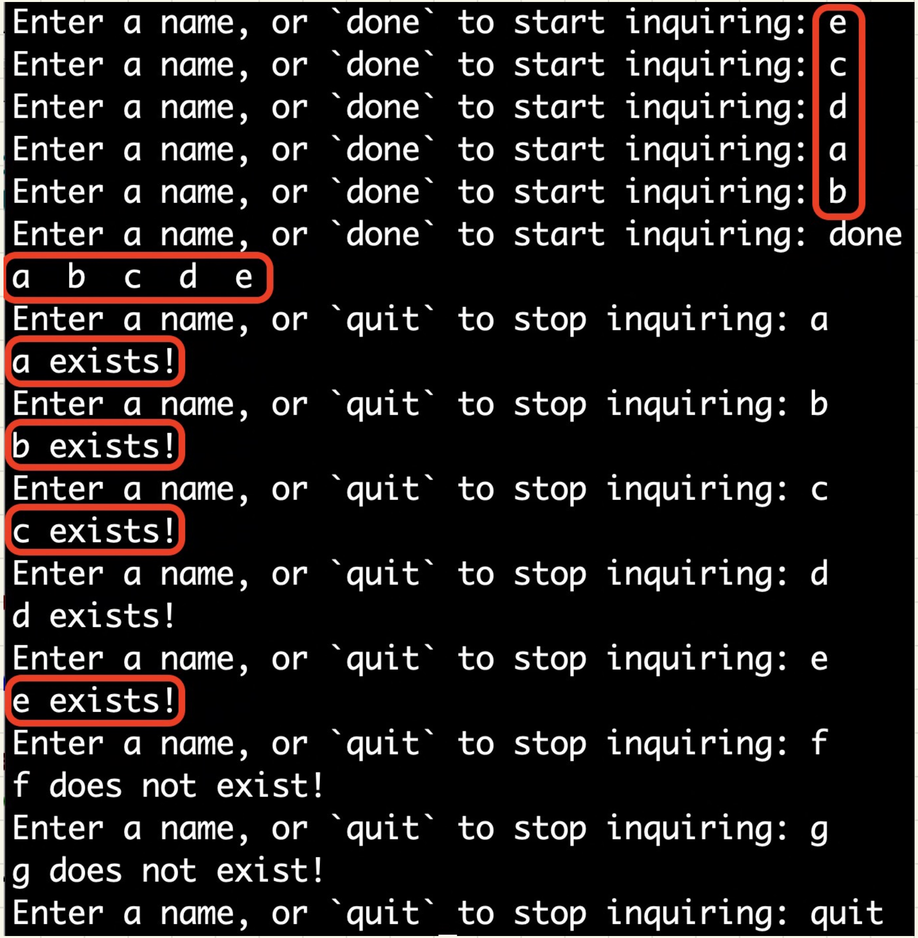


```
class interface
   DATABASE_V3

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_V3
```

```
class interface
   UTILITIES_V3

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_V3
```

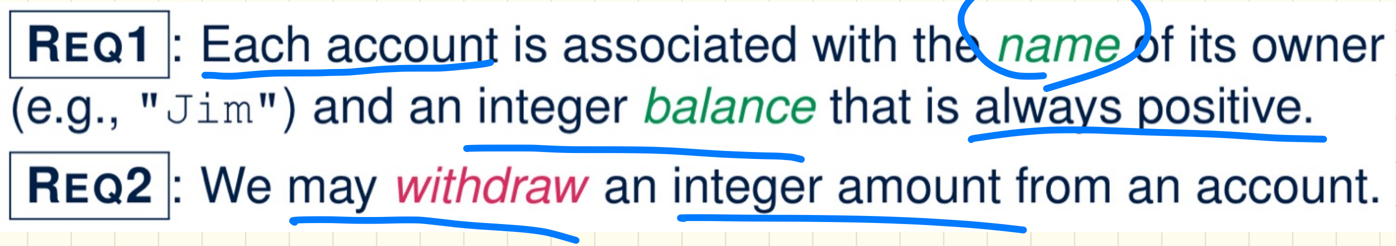
- Data array in DATABASE is now kept sorted (so as to avoid contract violation).
- Binary search in UTILITIES still requires a sorted input array.
- A sorted array is always passed for search, a contract violation never occurs!
- Now finalize/deliver the working system with contracts checking turned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

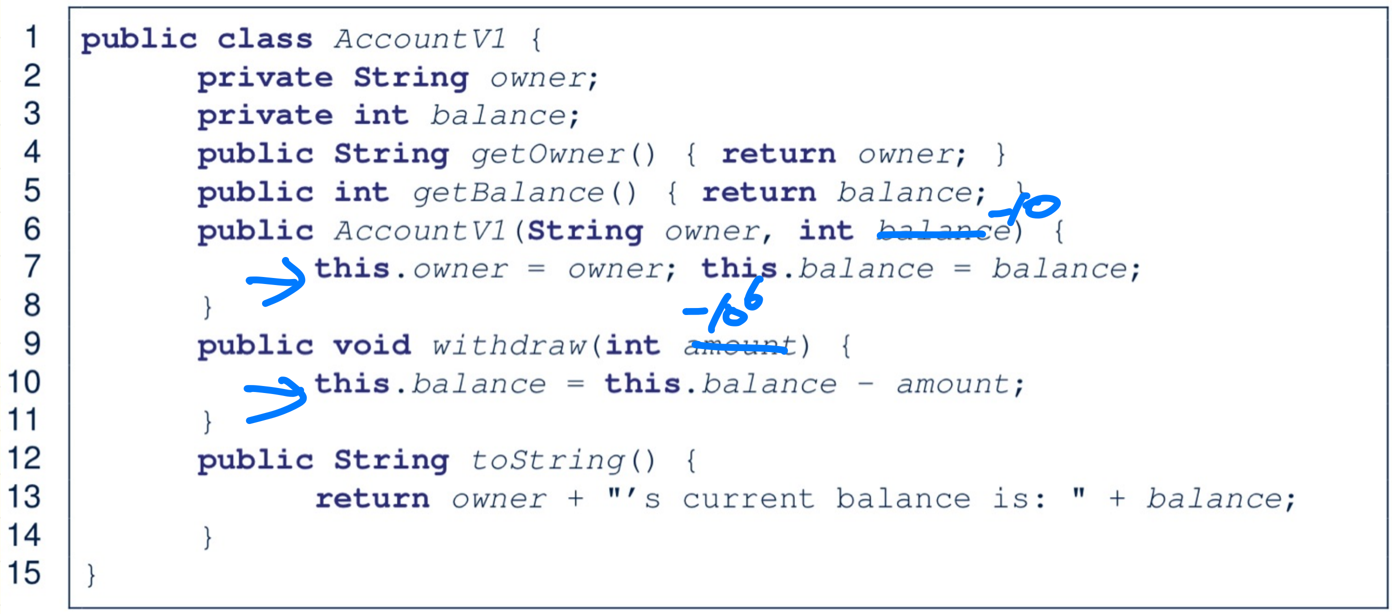
A Simple Design Problems : Bank Accounts



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.

Bank Accounts in Java : Version I

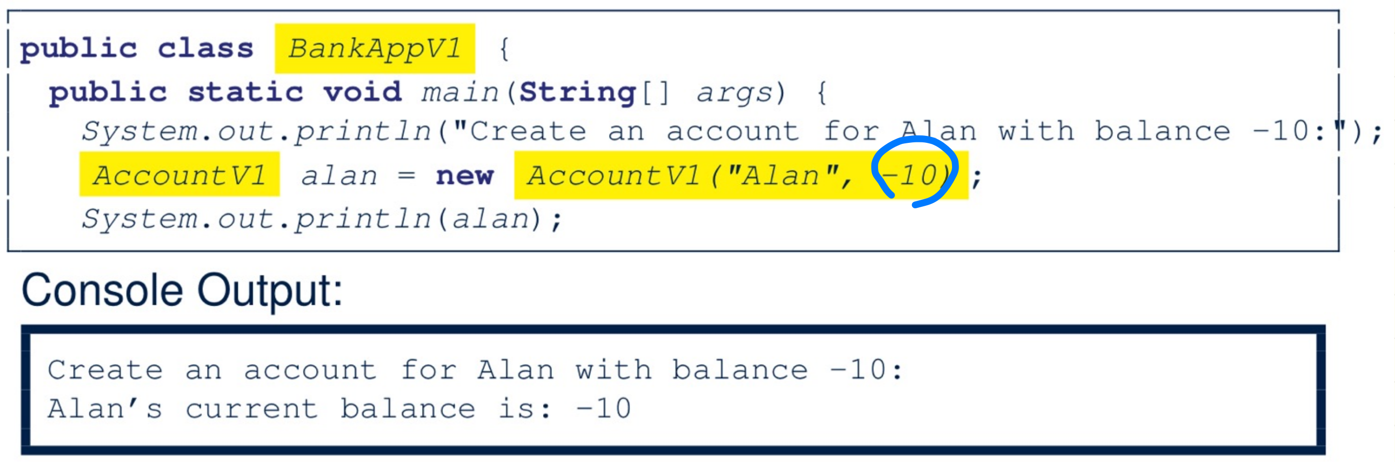


```
public class AccountV1 {
        private String owner;
        private int balance;
        public String getOwner() { return owner; }
        public int getBalance() { return balance; }
        public AccountV1(String owner, int balance) {
            this.owner = owner; this.balance = balance;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

~~balance~~ -10

~~amount~~ -10^6

Bank Accounts in Java: Version I Critique (1)

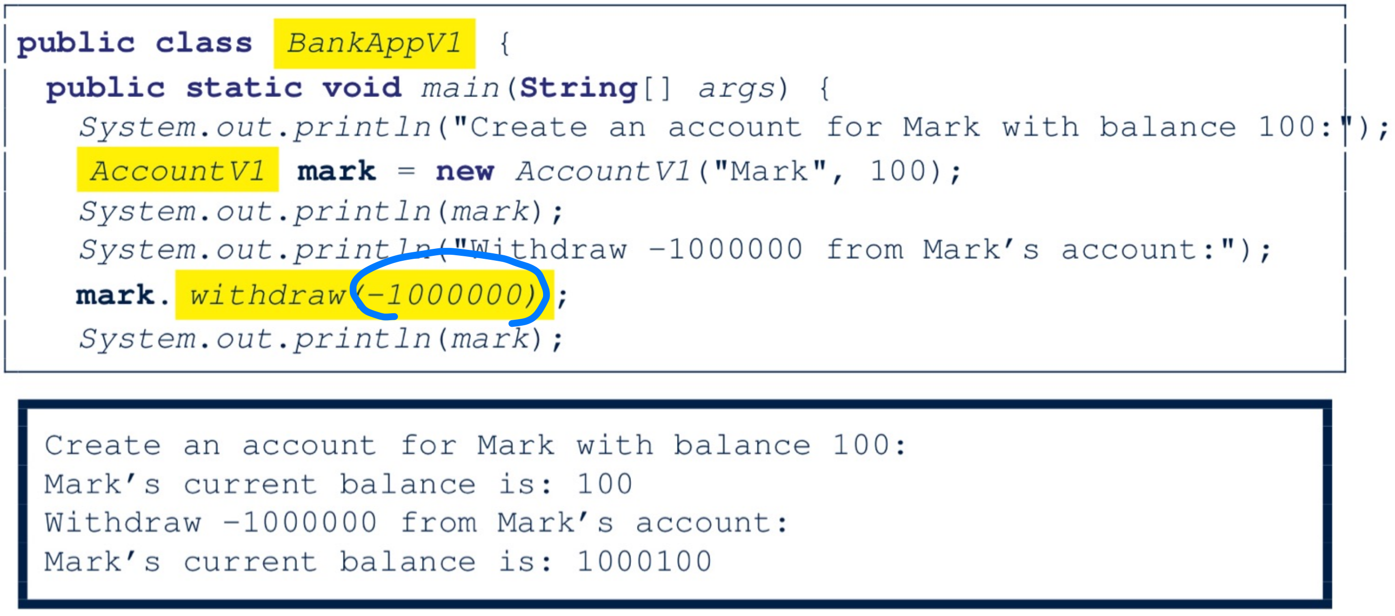


```
public class BankAppV1 {
  public static void main(String[] args) {
    System.out.println("Create an account for Alan with balance -10:");
    AccountV1 alan = new AccountV1("Alan", -10);
    System.out.println(alan);
```

Console Output:

```
Create an account for Alan with balance -10:
Alan's current balance is: -10
```

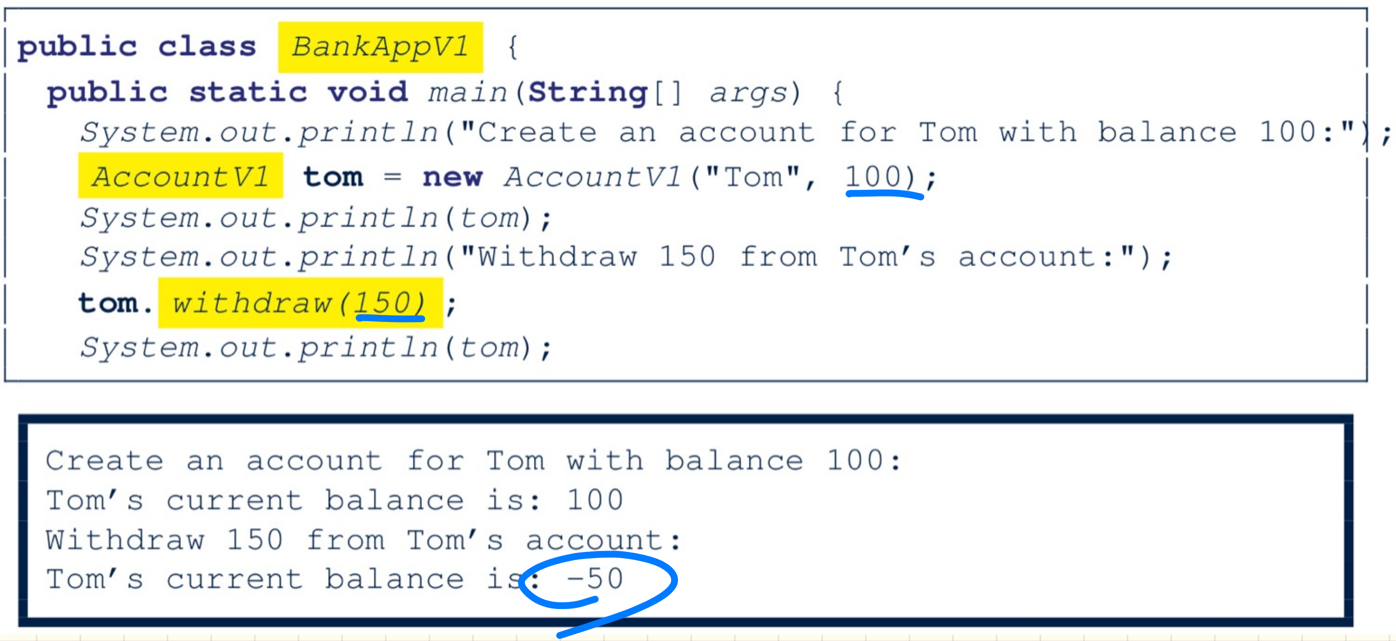
Bank Accounts in Java: Version I Critique (2)



```
public class BankAppV1 {
  public static void main(String[] args) {
    System.out.println("Create an account for Mark with balance 100:");
    AccountV1 mark = new AccountV1("Mark", 100);
    System.out.println(mark);
    System.out.println("Withdraw -1000000 from Mark’s account:");
    mark. withdraw(-1000000);
    System.out.println(mark);
```

```
Create an account for Mark with balance 100:
Mark’s current balance is: 100
Withdraw -1000000 from Mark’s account:
Mark’s current balance is: 1000100
```

Bank Accounts in Java : Version I Critique (3)



```
public class BankAppV1 {
  public static void main(String[] args) {
    System.out.println("Create an account for Tom with balance 100:");
    AccountV1 tom = new AccountV1("Tom", 100);
    System.out.println(tom);
    System.out.println("Withdraw 150 from Tom’s account:");
    tom. withdraw(150) ;
    System.out.println(tom);
```

```
Create an account for Tom with balance 100:
Tom’s current balance is: 100
Withdraw 150 from Tom’s account:
Tom’s current balance is: -50
```

divide(x, y)

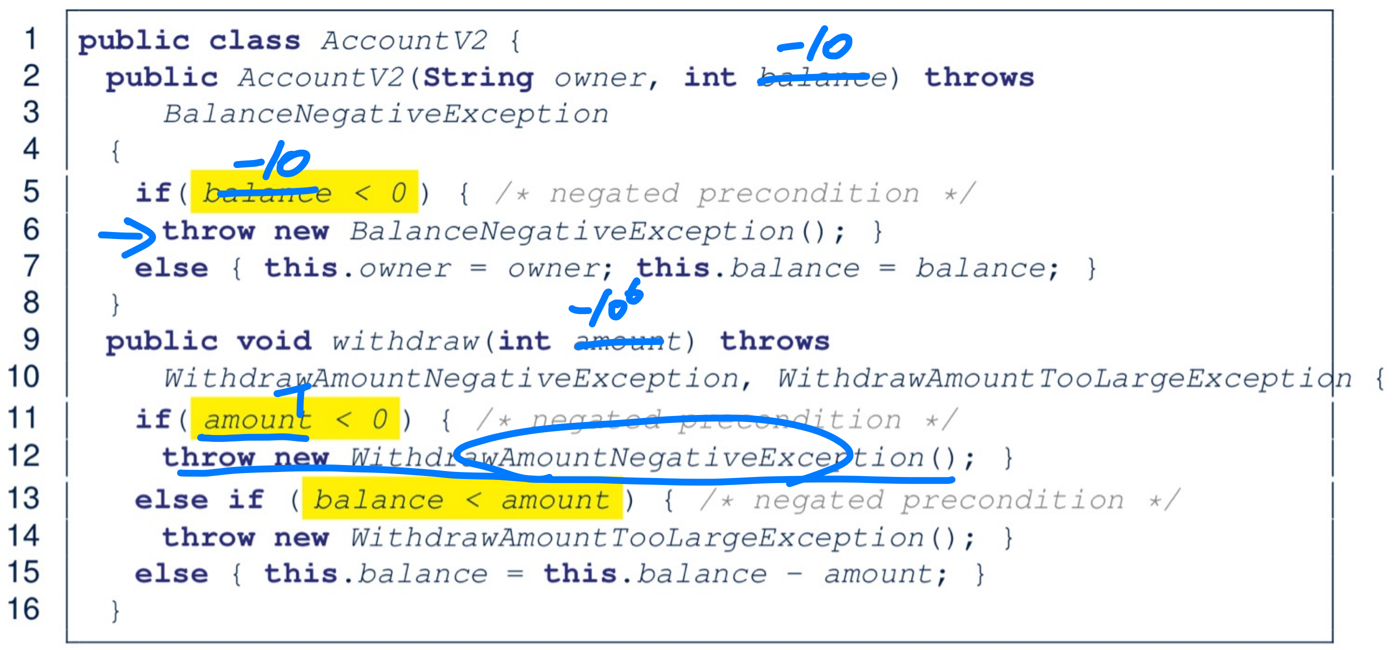
Precondition: Service

$y != 0$

Exceptions: Error

$y == 0$

Bank Accounts in Java: Version 2



```
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; }
  }
```

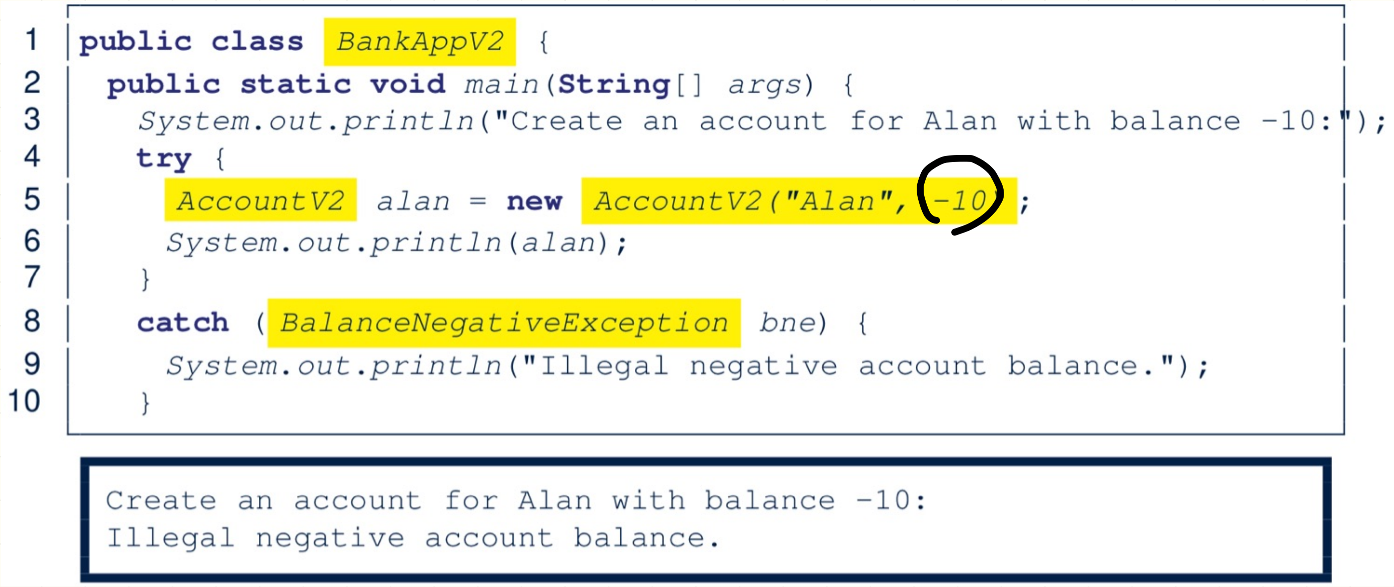
-10

-10

-10^6

Bank Accounts in Java : Version 2 Critique (1)

(Compared with Version I)

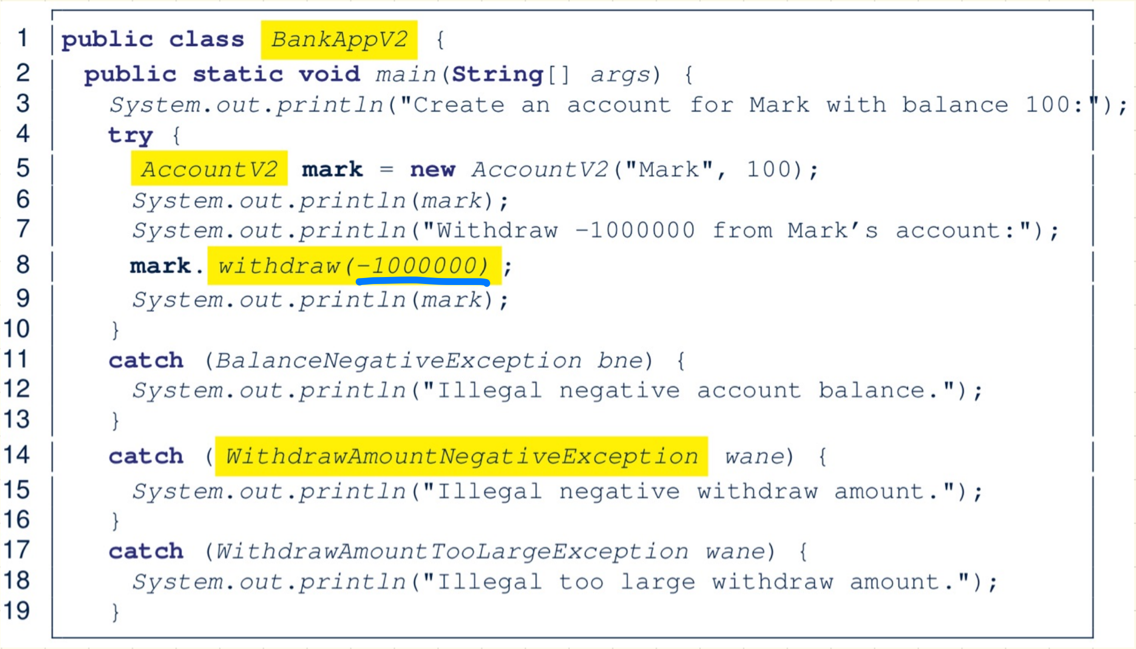


```
public class BankAppV2 {
  public static void main(String[] args) {
    System.out.println("Create an account for Alan with balance -10:");
    try {
      AccountV2 alan = new AccountV2("Alan", -10);
      System.out.println(alan);
    }
    catch (BalanceNegativeException bne) {
      System.out.println("Illegal negative account balance.");
    }
```

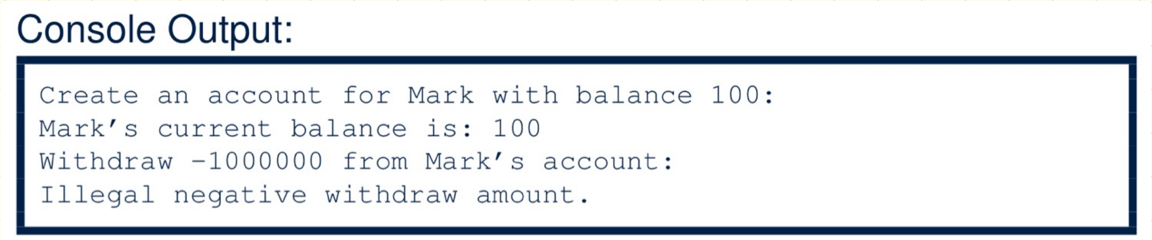
```
Create an account for Alan with balance -10:
Illegal negative account balance.
```

Bank Accounts in Java: Version 2 Critique (2)

(Compared with Version 1)



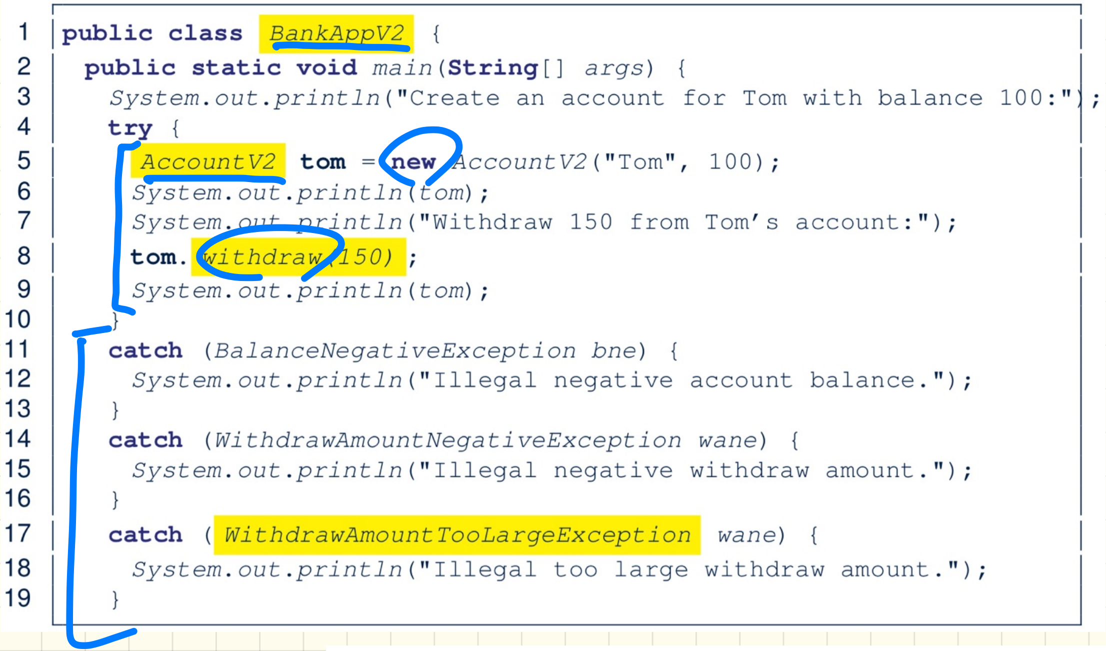
```java
public class BankAppV2 {
  public static void main(String[] args) {
    System.out.println("Create an account for Mark with balance 100:");
    try {
      AccountV2 mark = new AccountV2("Mark", 100);
      System.out.println(mark);
      System.out.println("Withdraw -1000000 from Mark's account:");
      mark.withdraw(-1000000);
      System.out.println(mark);
    }
    catch (BalanceNegativeException bne) {
      System.out.println("Illegal negative account balance.");
    }
    catch (WithdrawAmountNegativeException wane) {
      System.out.println("Illegal negative withdraw amount.");
    }
    catch (WithdrawAmountTooLargeException wane) {
      System.out.println("Illegal too large withdraw amount.");
    }
```



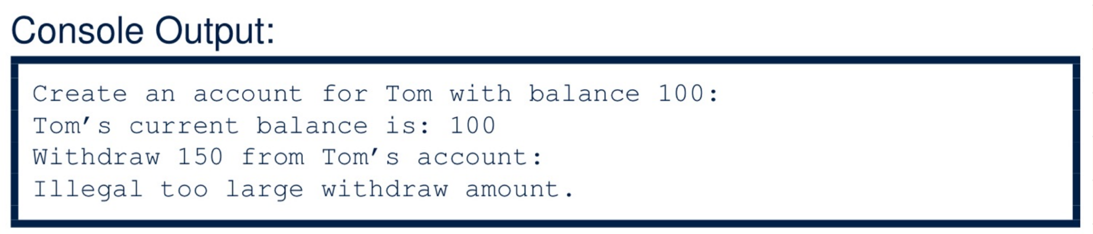
Console Output:

```
Create an account for Mark with balance 100:
Mark's current balance is: 100
Withdraw -1000000 from Mark's account:
Illegal negative withdraw amount.
```

Bank Accounts in Java: Version 2 Critique (3) (Compared with Version 1)



```
public class BankAppV2 {
  public static void main(String[] args) {
    System.out.println("Create an account for Tom with balance 100:");
    try {
      AccountV2 tom = new AccountV2("Tom", 100);
      System.out.println(tom);
      System.out.println("Withdraw 150 from Tom’s account:");
      tom.withdraw(150) ;
      System.out.println(tom);
    }
    catch (BalanceNegativeException bne) {
      System.out.println("Illegal negative account balance.");
    }
    catch (WithdrawAmountNegativeException wane) {
      System.out.println("Illegal negative withdraw amount.");
    }
    catch ( WithdrawAmountTooLargeException wane) {
      System.out.println("Illegal too large withdraw amount.");
    }
```

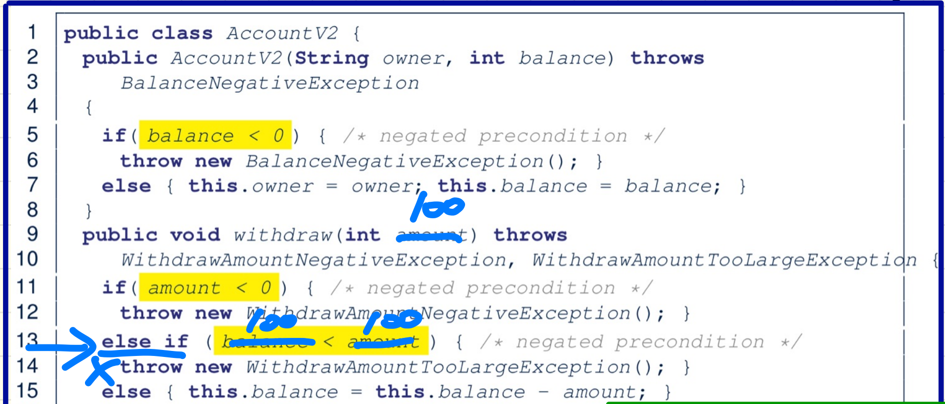


Console Output:

```
Create an account for Tom with balance 100:
Tom’s current balance is: 100
Withdraw 150 from Tom’s account:
Illegal too large withdraw amount.
```

Bank Accounts in Java : Version 2 Critique (4)

Supplier

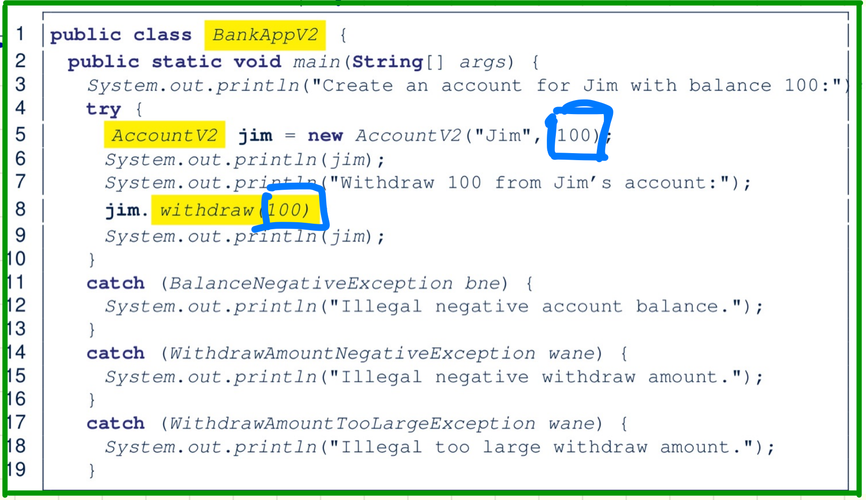


```
public class AccountV2 {
  public AccountV2(String owner, int balance) throws
      BalanceNegativeException
   {
    if( balance < 0 ) { /* negated precondition */
     throw new BalanceNegativeException(); }
    else { this.owner = owner; this.balance = balance; }
   }
  public void withdraw(int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

100

100 100

Client



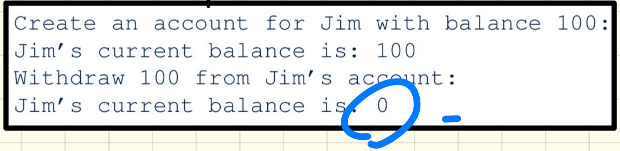
```
public class BankAppV2 {
  public static void main(String[] args) {
   System.out.println("Create an account for Jim with balance 100:");
   try {
    AccountV2 jim = new AccountV2("Jim", 100);
    System.out.println(jim);
    System.out.println("Withdraw 100 from Jim's account:");
    jim. withdraw(100)
    System.out.println(jim);
   }
   catch (BalanceNegativeException bne) {
    System.out.println("Illegal negative account balance.");
   }
   catch (WithdrawAmountNegativeException wane) {
    System.out.println("Illegal negative withdraw amount.");
   }
   catch (WithdrawAmountTooLargeException wane) {
    System.out.println("Illegal too large withdraw amount.");
   }
```

REQ:

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

?

Console Output :



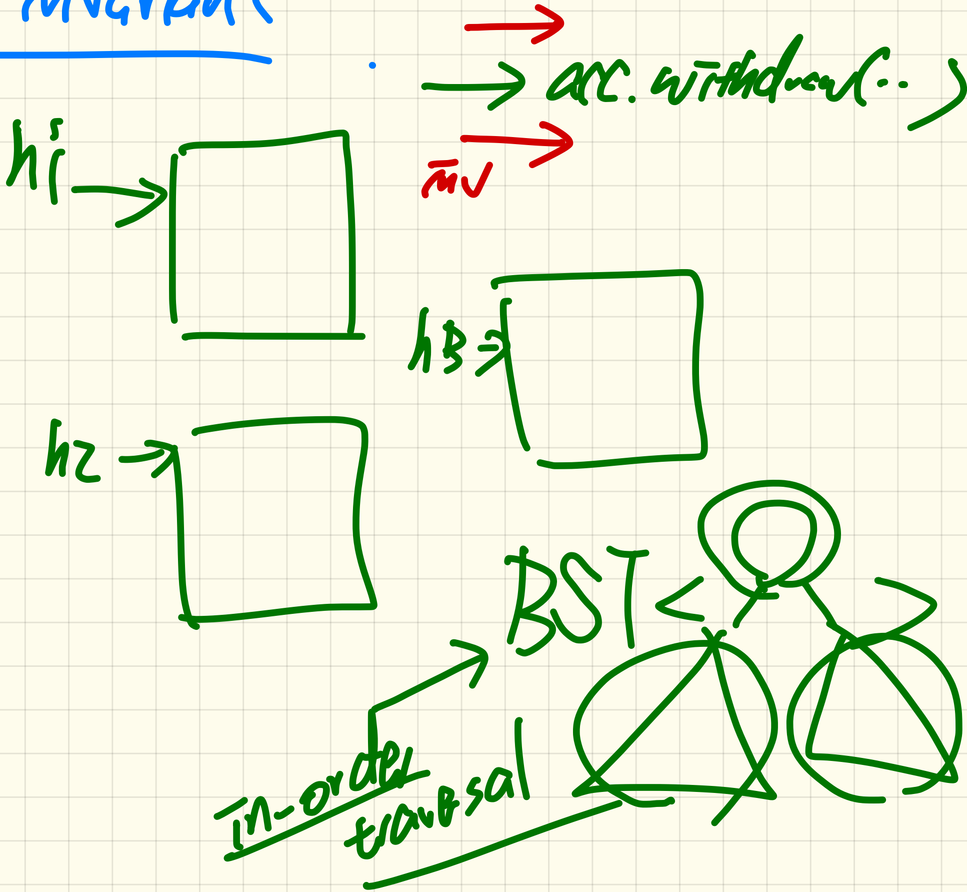
```
Create an account for Jim with balance 100:
Jim's current balance is: 100
Withdraw 100 from Jim's account:
Jim's current balance is: 0
```

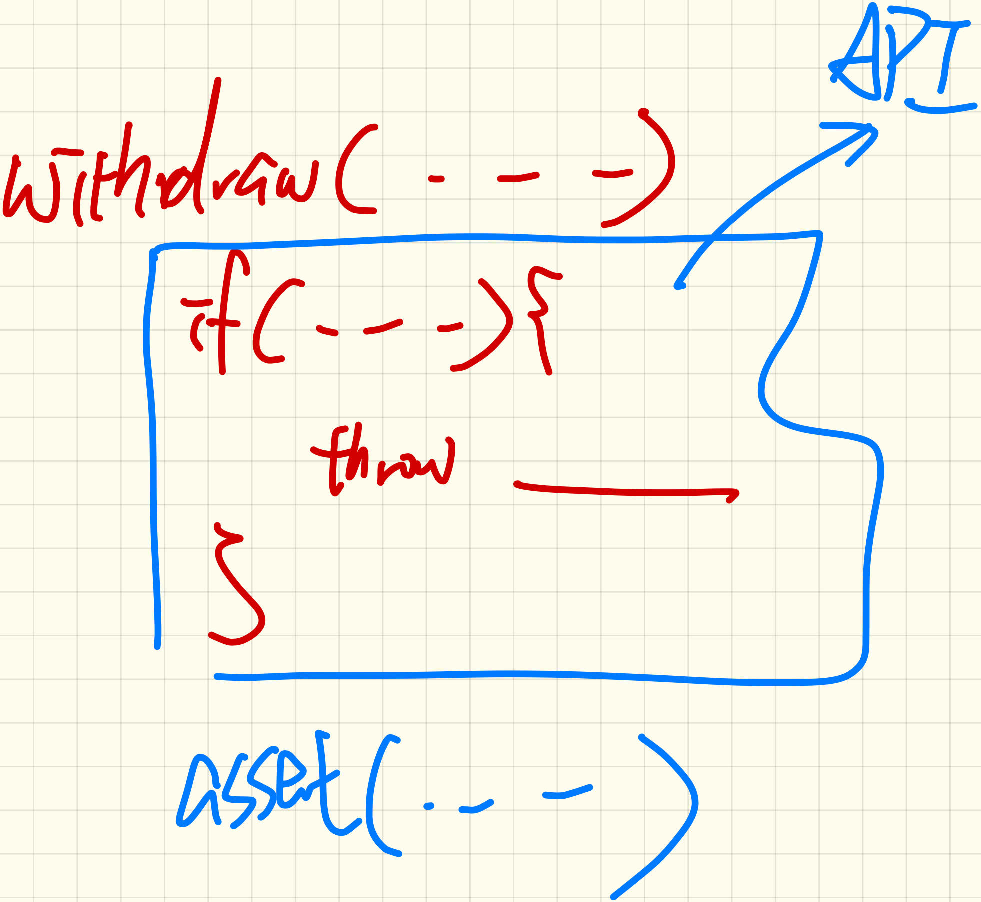
-

class invariant

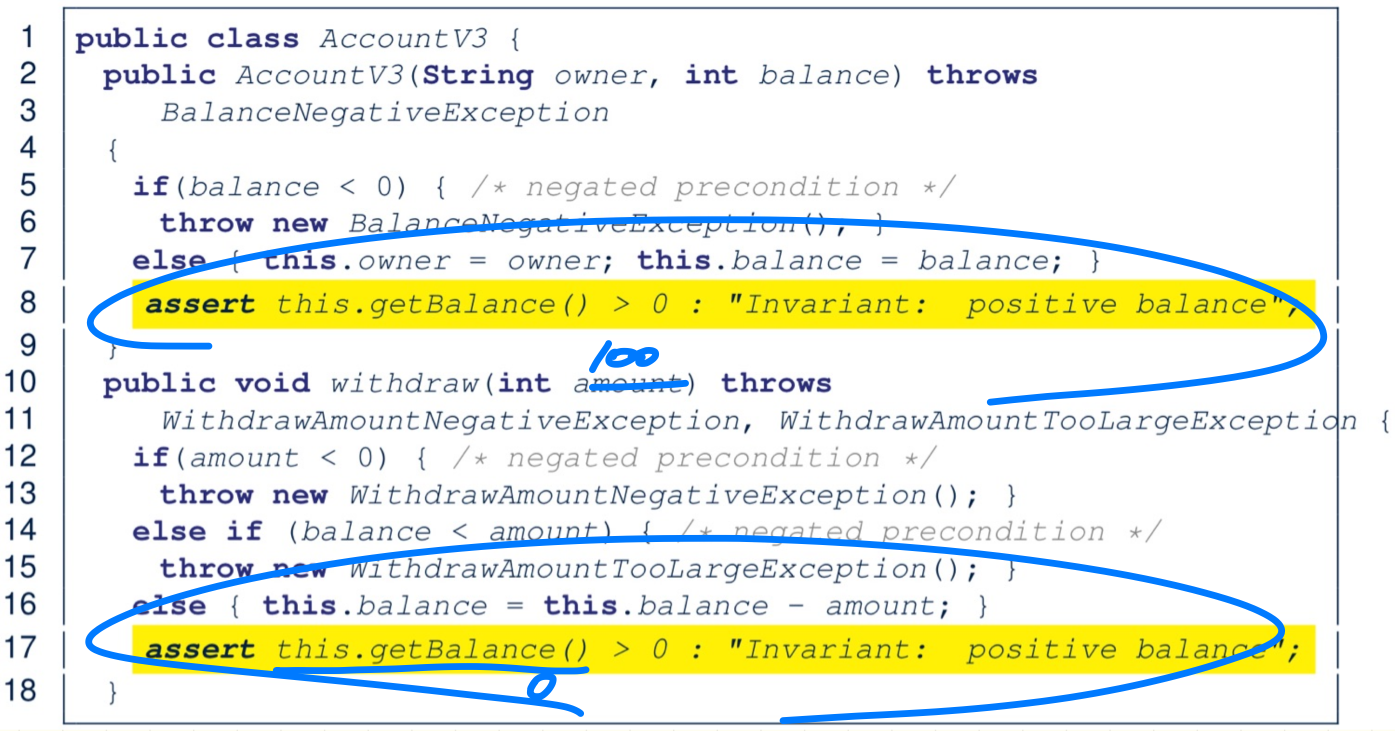
compile time

```
class HEAP

  invariant
    P
end
```




API
withdraw(- - -)
if(- - -){
throw
}
assert(- - -)

Bank Accounts in Java: Version 3



```
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";
  }
```

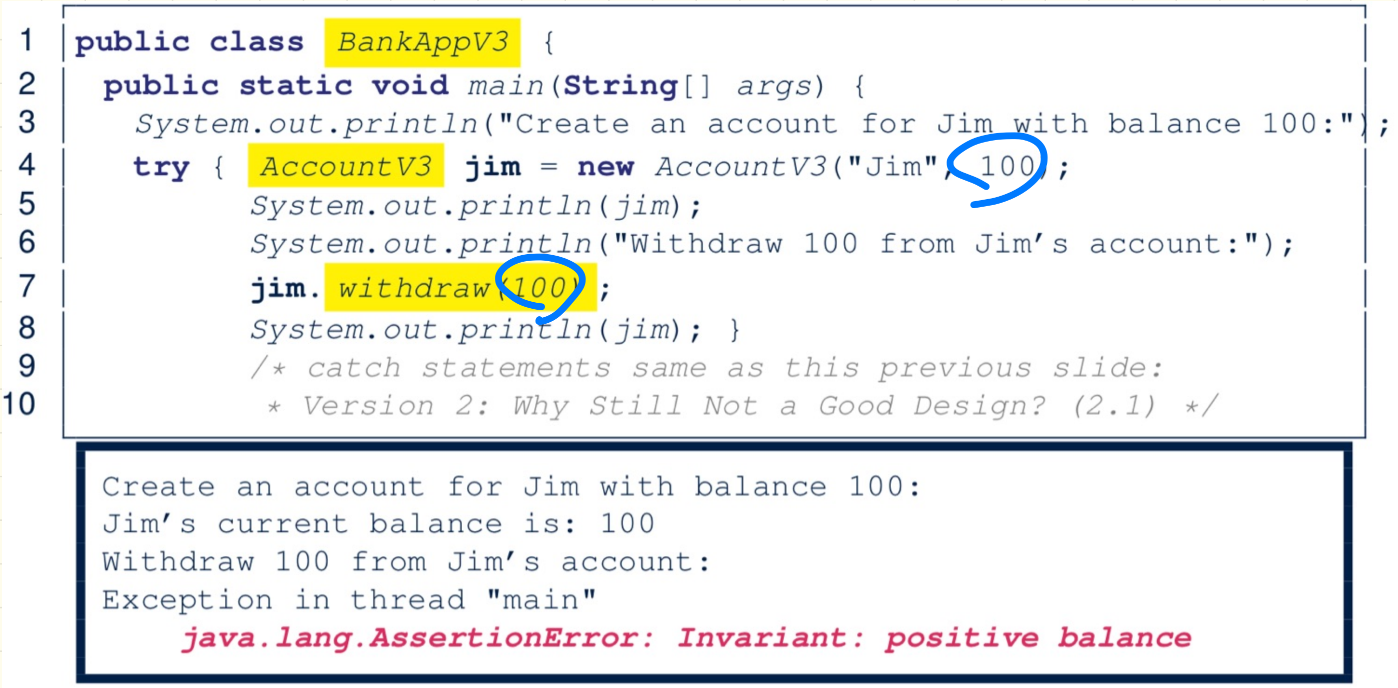
100

0

violates single choice principle.

Bank Accounts in Java: Version 3 Critique (1)

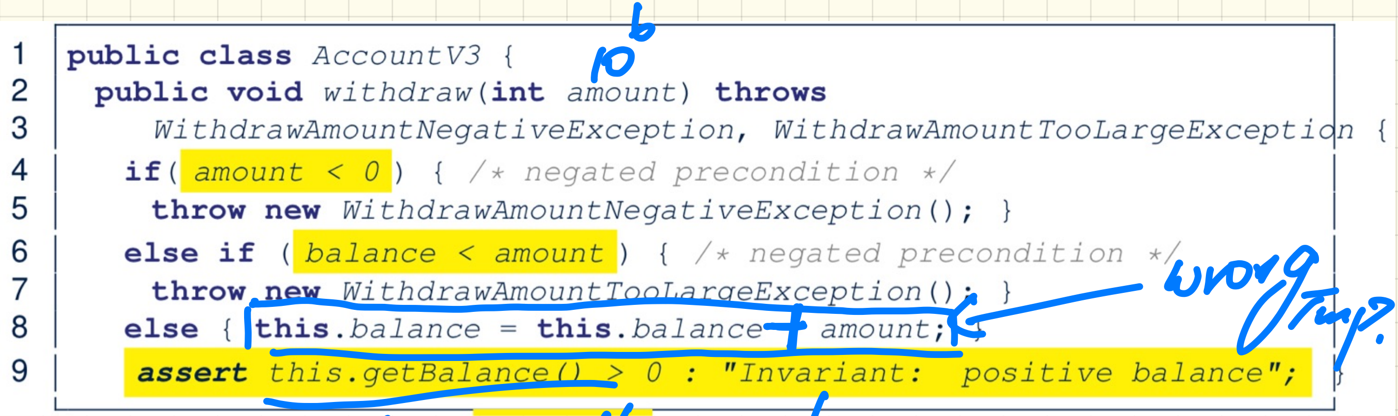
(Compared with Version 2)



```java
public class BankAppV3 {
  public static void main(String[] args) {
    System.out.println("Create an account for Jim with balance 100:");
    try { AccountV3 jim = new AccountV3("Jim", 100);
          System.out.println(jim);
          System.out.println("Withdraw 100 from Jim's account:");
          jim.withdraw(100);
          System.out.println(jim); }
          /* catch statements same as this previous slide:
           * Version 2: Why Still Not a Good Design? (2.1) */
```

```
Create an account for Jim with balance 100:
Jim's current balance is: 100
Withdraw 100 from Jim's account:
Exception in thread "main"
    java.lang.AssertionError: Invariant: positive balance
```

Bank Accounts in Java: Version 3 Critique (2)



```
public class AccountV3 {
  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"; }
```

10^6

+

wrong Imp?

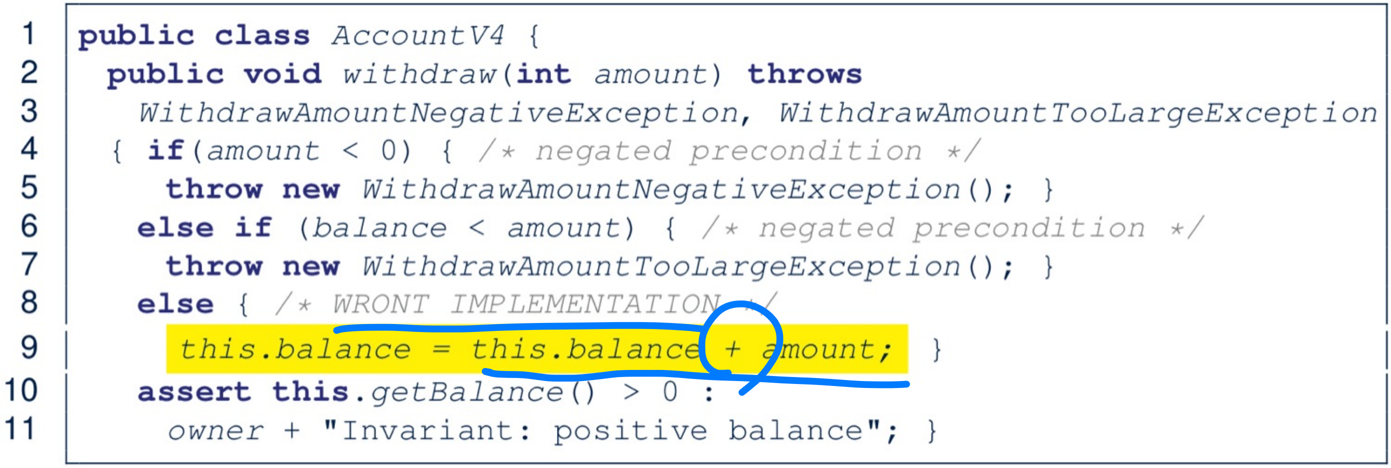
→ nothing will signal an error.

When amount is neither negative nor too large,

Is there any obligation on the supplier of withdraw?

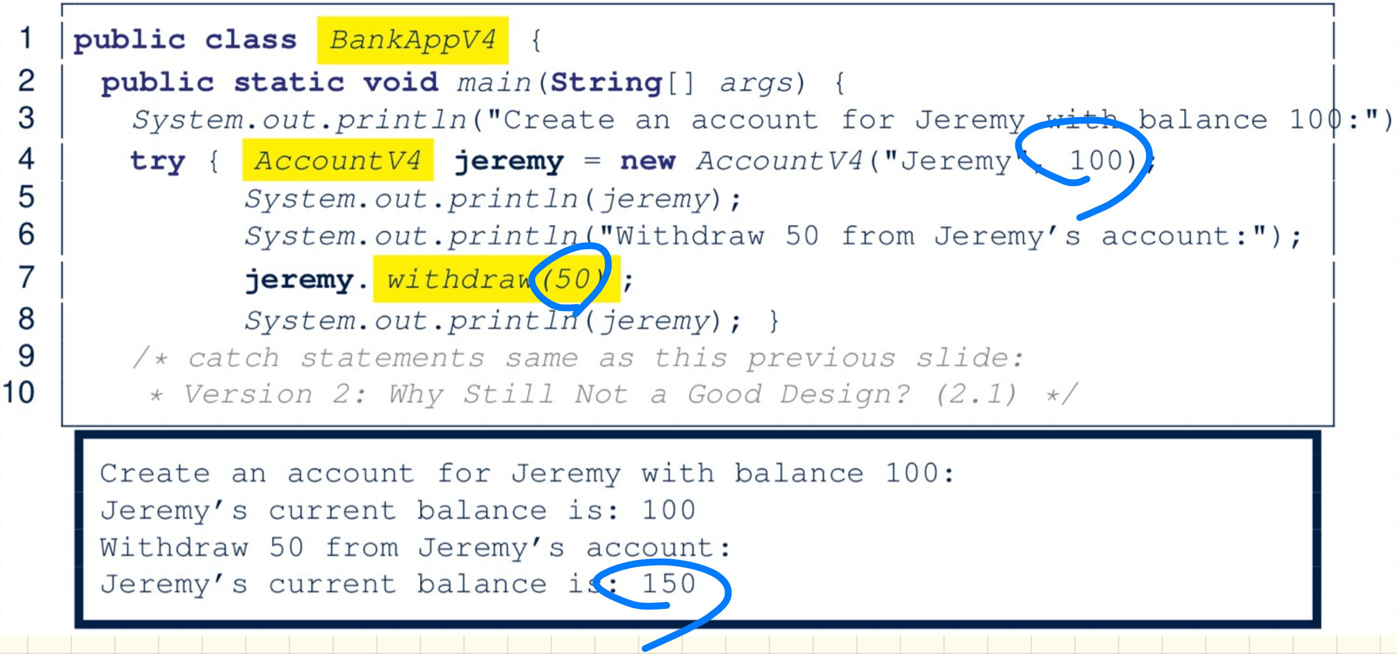
Bank Accounts in Java: Version 4

(with an evil supplier)



```
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"; }
```

Bank Accounts in Java : Version 4 Critique



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

```
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
```

Precondition & Postcondition Exercise

$\forall$ $\exists$

change_at (a: **ARRAY**[**STRING**]; i: **INTEGER**; ns; **STRING**)
- - **Change index \`i\` in array \`a\` to string \`ns\`**
require
??
ensure
??

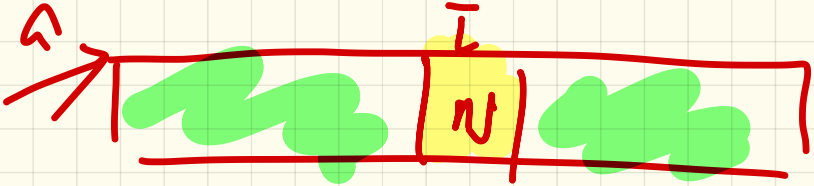
$1 <= i$ and $i <= a.count.$

→ ✓ $a[i] \sim ns$

evil supplier:

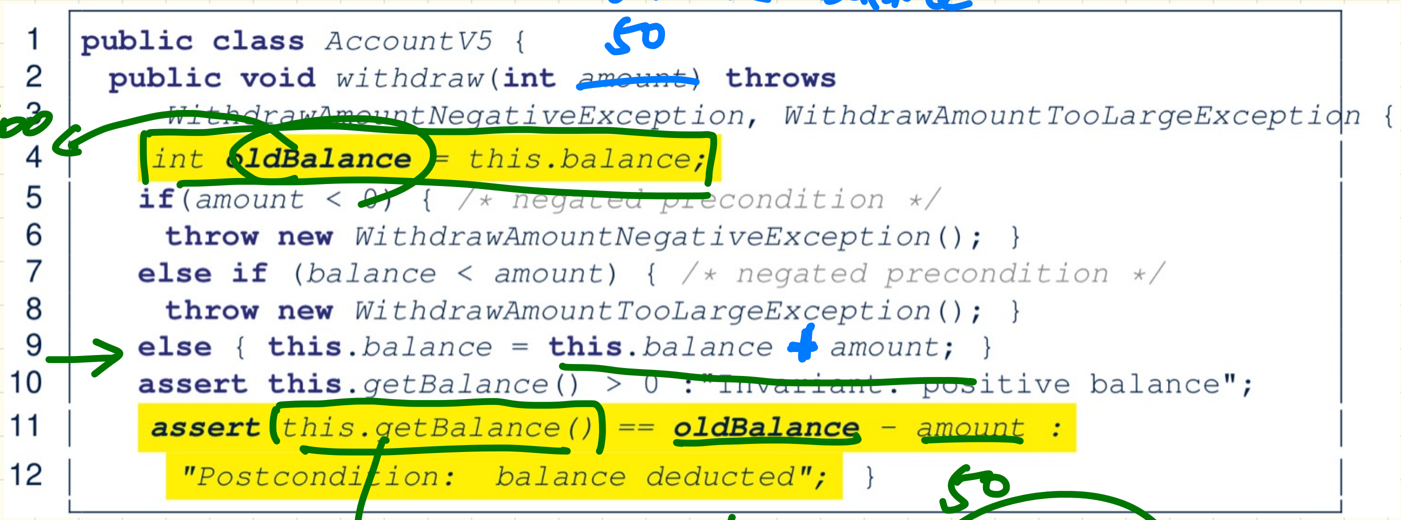
$a[i] = ns;$

$a[i-1] = null;$



Bank Accounts in Java : Version 5

current balance : 100



```
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"; }
```

50

100

150

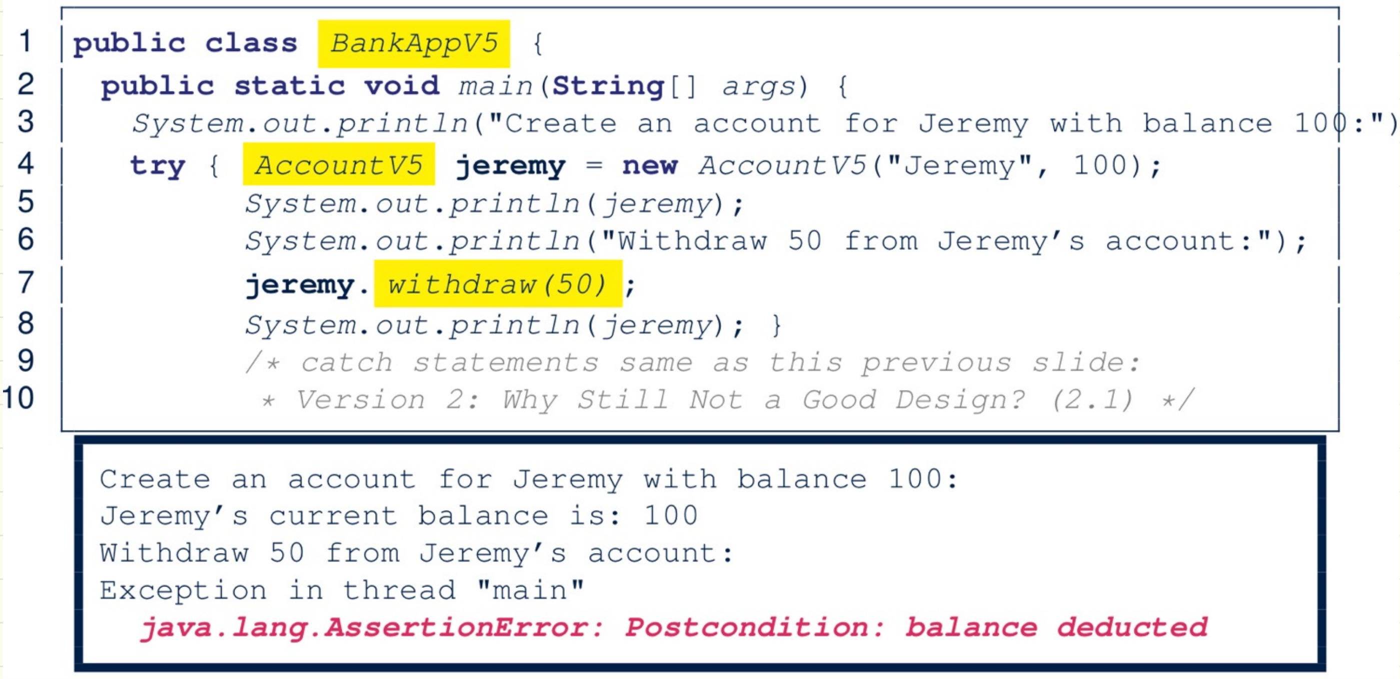
100

50

F

Bank Accounts in Java: Version 5 Critique

(Compared with Version 4)



```java
public class BankAppV5 {
  public static void main(String[] args) {
    System.out.println("Create an account for Jeremy with balance 100:")
    try { AccountV5 jeremy = new AccountV5("Jeremy", 100);
          System.out.println(jeremy);
          System.out.println("Withdraw 50 from Jeremy's account:");
          jeremy.withdraw(50);
          System.out.println(jeremy); }
          /* catch statements same as this previous slide:
           * Version 2: Why Still Not a Good Design? (2.1) */
```

```
Create an account for Jeremy with balance 100:
Jeremy's current balance is: 100
Withdraw 50 from Jeremy's account:
Exception in thread "main"
  java.lang.AssertionError: Postcondition: balance deducted
```

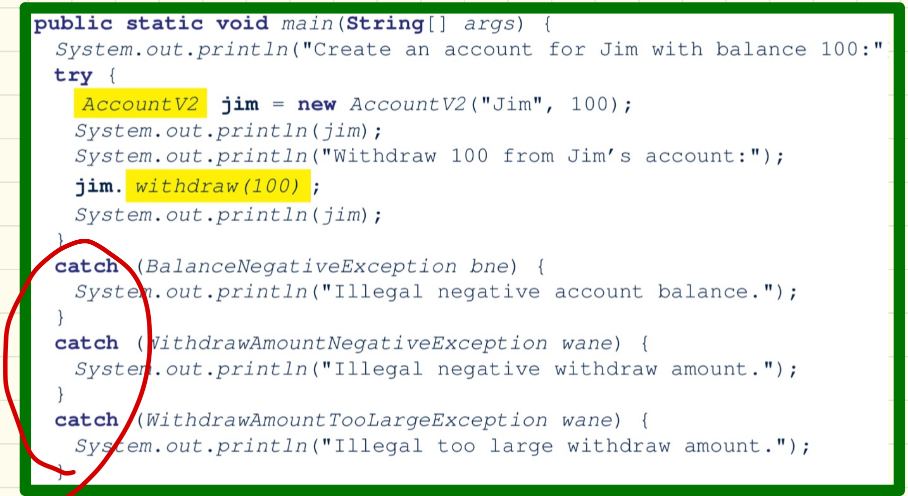
LECTURE 3

THURSDAY SEPTEMBER 12

- In-Lab Demo yesterday
(source code, recording)
- TA office hours
11am ~ 2pm FRIDAYS
LAS 2056
- slides on BoW diagram soon.

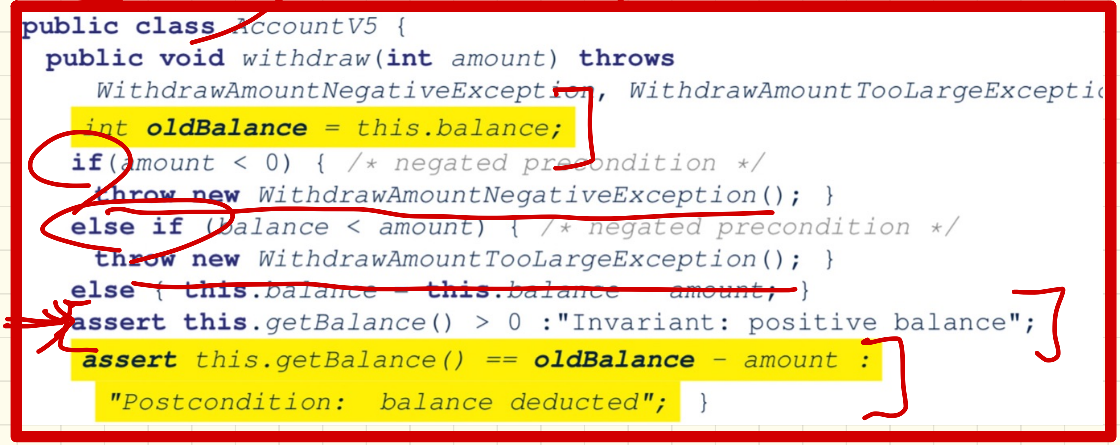
Design by Contract in Java

CLIENT



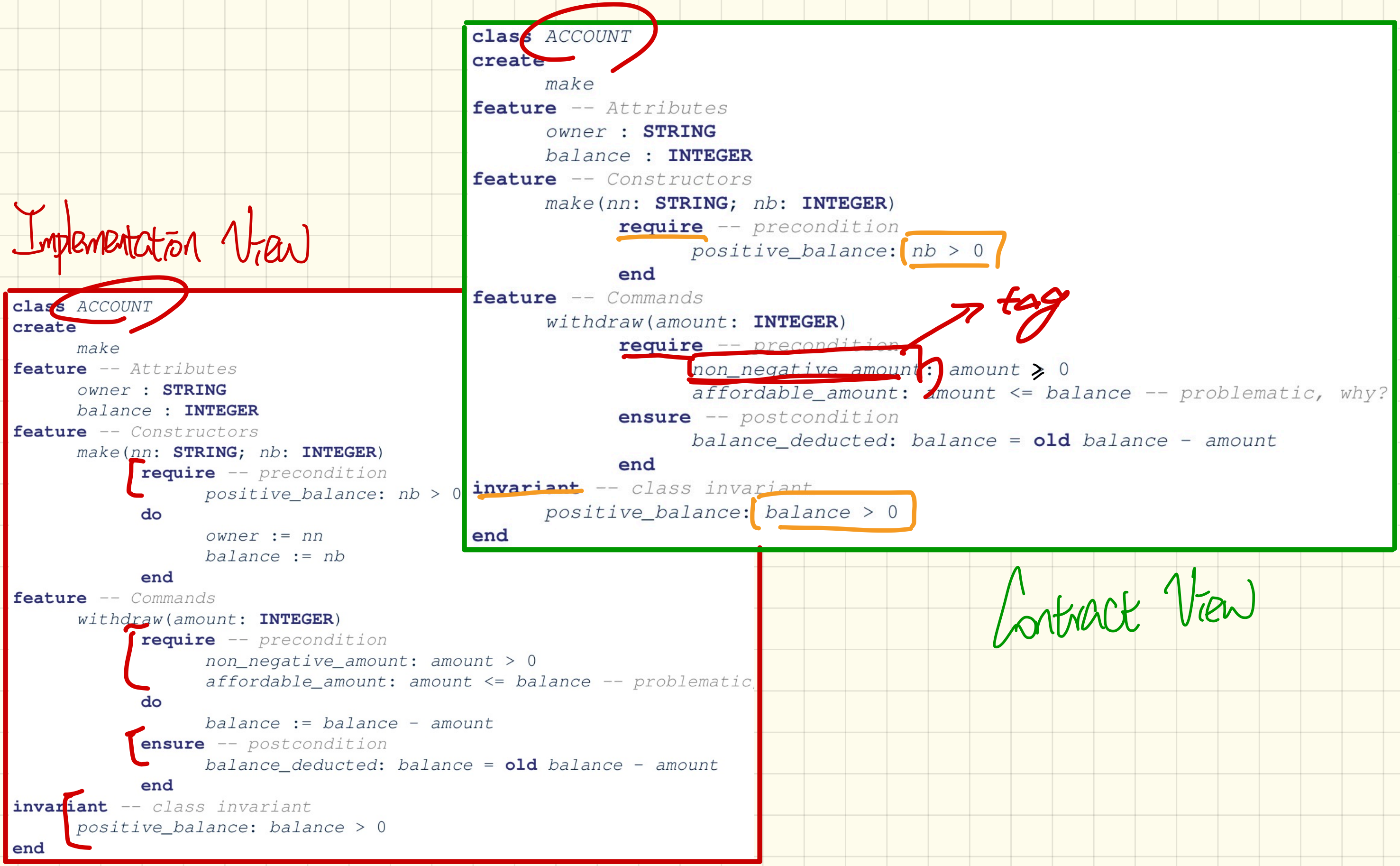
```java
public static void main(String[] args) {
  System.out.println("Create an account for Jim with balance 100:"
  try {
    AccountV2 jim = new AccountV2("Jim", 100);
    System.out.println(jim);
    System.out.println("Withdraw 100 from Jim's account:");
    jim. withdraw(100) ;
    System.out.println(jim);
  }
  catch (BalanceNegativeException bne) {
    System.out.println("Illegal negative account balance.");
  }
  catch (WithdrawAmountNegativeException wane) {
    System.out.println("Illegal negative withdraw amount.");
  }
  catch (WithdrawAmountTooLargeException wane) {
    System.out.println("Illegal too large withdraw amount.");
  }
```

SUPPLIER



```java
public class AccountV5 {
  public void withdraw(int amount) throws
    WithdrawAmountNegativeException, WithdrawAmountTooLargeExcepti
   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"; }
```

Design by Contract in Eiffel



Implementation View

```
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 -- problematic,
        do
            balance := balance - amount
        ensure -- postcondition
            balance_deducted: balance = old balance - amount
        end
invariant -- class invariant
    positive_balance: balance > 0
end
```

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

tag

Contract View

Runtime Monitoring of Contracts

```
acc: ACCOUNT
create acc.make(a, n)
acc.withdraw(a)
```

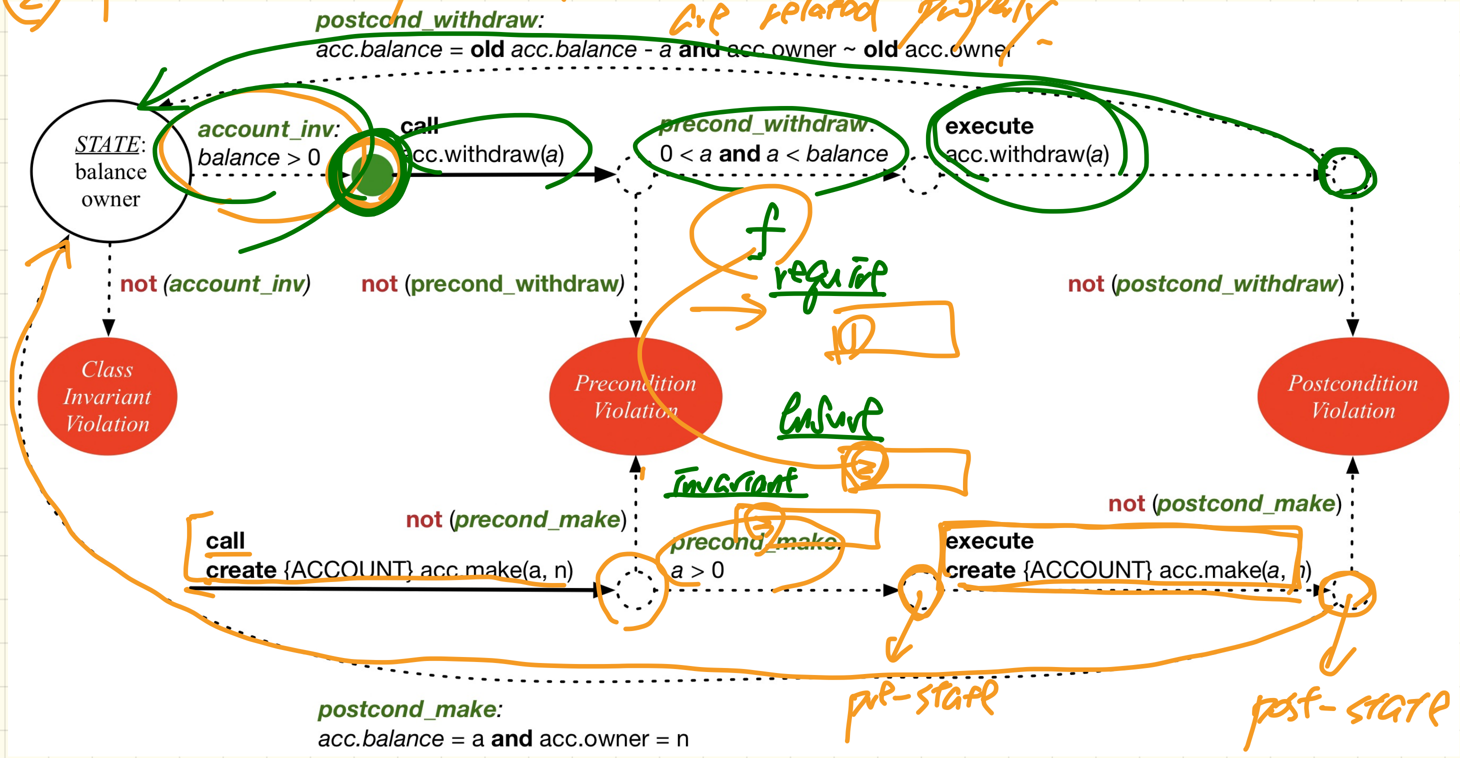
b > 0

pred.

① valid inputs (invariant not related)

③ valid objects

② pre-state and post-state values are related properly.



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*)

precond_withdraw: 0 < *a* **and** *a* < *balance*

execute acc.withdraw(*a*)

not (**account_inv**)

not (**precond_withdraw**)

not (**postcond_withdraw**)

Class Invariant Violation

Precondition Violation

Postcondition Violation

f

require ①

ensure ②

invariant ③

not (**precond_make**)

not (**postcond_make**)

call create {ACCOUNT} acc.make(a, n)

precond_make: *a* > 0

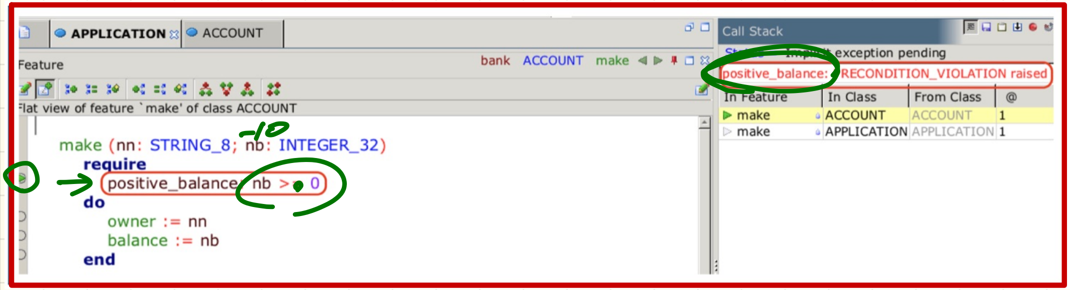
execute create {ACCOUNT} acc.make(a, n)

pre-state

post-state

postcond_make:
acc.balance = a **and** acc.owner = n

Precondition Violation: positive_balance



lat view of feature `make' of class ACCOUNT

```
make (nn: STRING_8; nb: INTEGER_32)
    require
        positive_balance: nb > 0
    do
        owner := nn
        balance := nb
    end
```

-10

St... Impl... exception pending

positive_balance: PRECONDITION_VIOLATION raised

In Feature	In Class	From Class	@
make	ACCOUNT	ACCOUNT	1
make	APPLICATION	APPLICATION	1

Supplier

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

-10

-10

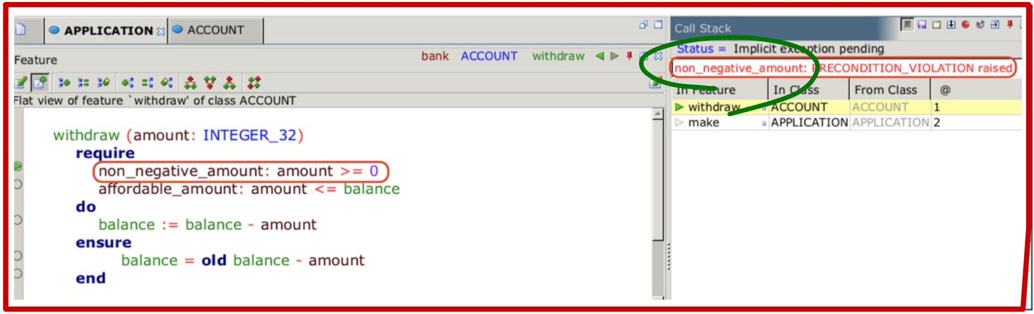
F

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
  local
    alan: ACCOUNT
  do
    -- A precondition violation with tag
    create {ACCOUNT} alan.make ("Alan", -10)
  end
end
```

Precondition Violation:

non_negative_amount



APPLICATION ACCOUNT

Feature

bank ACCOUNT withdraw

Flat view of feature `withdraw' of class ACCOUNT

```
withdraw (amount: INTEGER_32)
    require
        non_negative_amount: amount >= 0
        affordable_amount: amount <= balance
    do
        balance := balance - amount
    ensure
            balance = old balance - amount
    end
```

Call Stack

Status = Implicit exception pending

non_negative_amount: PRECONDITION_VIOLATION raised

In Feature	In Class	From Class	@
withdraw	ACCOUNT	ACCOUNT	1
make	APPLICATION	APPLICATION	2

Supplier

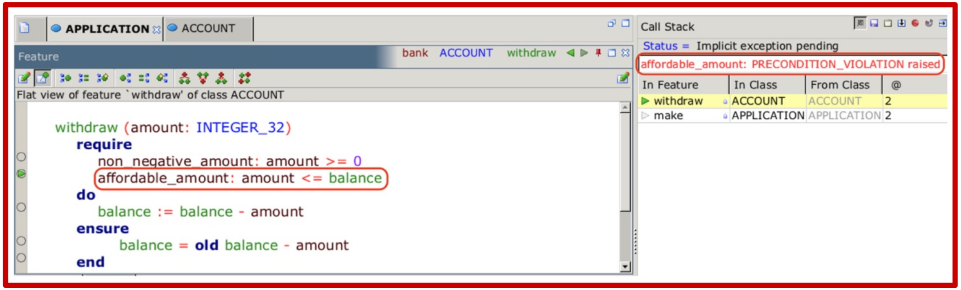
```
class ACCOUNT
create
      make
feature -- Attributes
      owner : STRING
      balance : INTEGER
feature -- Constructors
      make(nn: STRING; nb: INTEGER)
            require -- precondition
                  positive_balance: nb > 0
            end
feature -- Commands
      withdraw(amount: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

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
  local
    mark: ACCOUNT
  do
    create {ACCOUNT} mark.make ("Mark", 100)
    -- A precondition violation with tag "nc
    mark.withdraw(-1000000)
  end
end
```

Precondition Violation:

affordable_amount



Flat view of feature `withdraw' of class ACCOUNT

```
withdraw (amount: INTEGER_32)
    require
        non_negative_amount: amount >= 0
        affordable_amount: amount <= balance
    do
        balance := balance - amount
    ensure
            balance = old balance - amount
    end
```

Status = Implicit exception pending

affordable_amount: PRECONDITION_VIOLATION raised

In Feature	In Class	From Class	@
withdraw	ACCOUNT	ACCOUNT	2
make	APPLICATION	APPLICATION	2

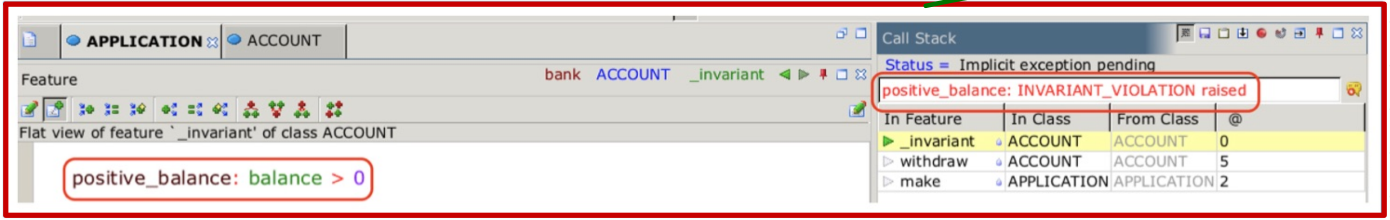
Supplier

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

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
  local
    tom: ACCOUNT
  do
    create {ACCOUNT} tom.make ("Tom", 100)
    -- A precondition violation with tag "
    tom.withdraw(150)
  end
end
```

Class Invariant Violation: positive_balance



Flat view of feature `_invariant' of class ACCOUNT

positive_balance: balance > 0

Status = Implicit exception pending

positive_balance: INVARIANT_VIOLATION raised

In Feature	In Class	From Class	@
_invariant	ACCOUNT	ACCOUNT	0
withdraw	ACCOUNT	ACCOUNT	5
make	APPLICATION	APPLICATION	2

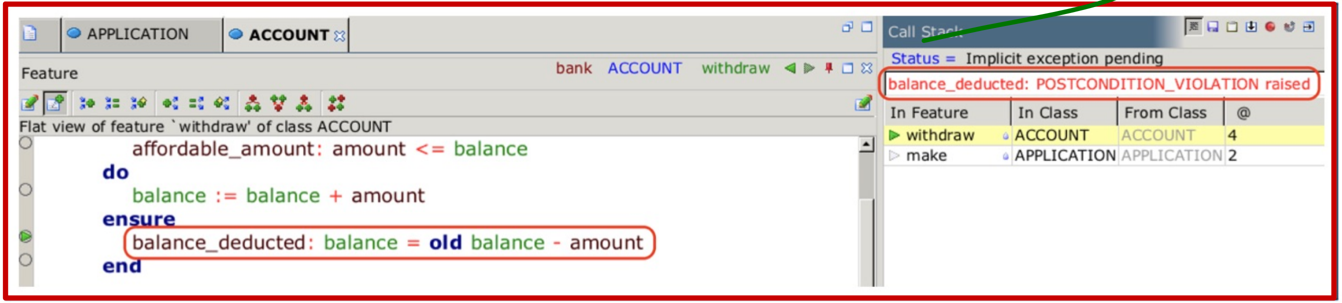
Supplier

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

Client

```
class BANK_APP
inherit
  ARGUMENTS
create
  make
feature -- Initialization
  make
    -- Run application.
  local
    jim: ACCOUNT
  do
    create {ACCOUNT} tom.make ("Jim", 100)
    jim.withdraw(100)
    -- A class invariant violation with tag "positive_balance"
  end
end
```

Postcondition Violation: balance_deducted



Flat view of feature `withdraw' of class ACCOUNT

```
        affordable_amount: amount <= balance
    do
        balance := balance + amount
    ensure
        balance_deducted: balance = old balance - amount
    end
```

Call Stack

Status = Implicit exception pending

balance_deducted: POSTCONDITION_VIOLATION raised

In Feature	In Class	From Class	@
▶ withdraw	ACCOUNT	ACCOUNT	4
▷ make	APPLICATION	APPLICATION	2

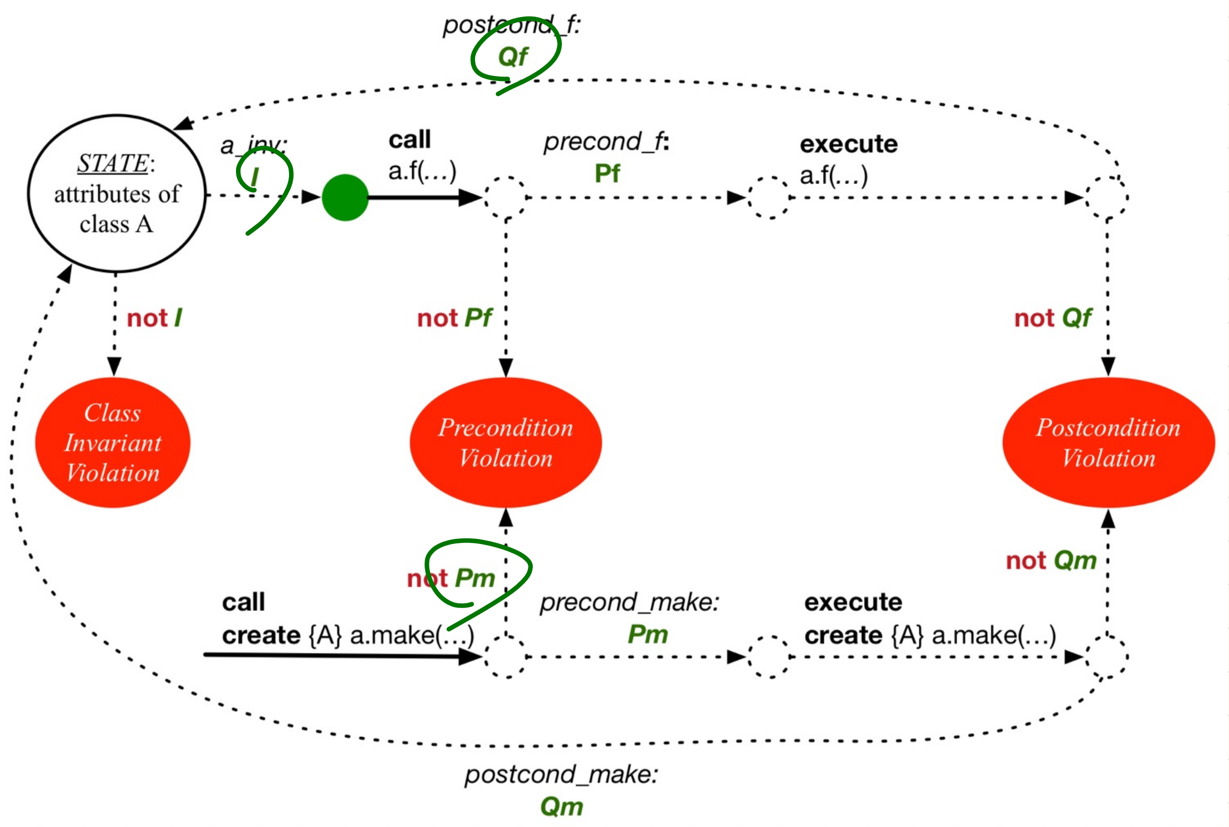
Supplier

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

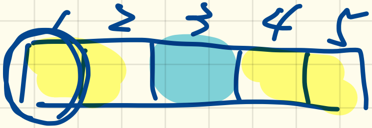
Client

```
class BANK_APP
inherit ARGUMENTS
create make
feature -- Initialization
  make
    -- Run application.
  local
    jeremy: ACCOUNT
  do
    -- Faulty implementation of withdraw in ACCOU
    -- balance := balance + amount
    create {ACCOUNT} jeremy.make ("Jeremy", 100)
    jeremy.withdraw(150)
    -- A postcondition violation with tag "balance_deducted"
  end
end
```

Runtime Monitoring of Contracts



Precondition & Postcondition Exercise



3

→ *change_at* (a: **ARRAY**[**STRING**]; i: **INTEGER**; ns; **STRING**)

- - Change index \`i\` in array \`a\` to string \`ns\`

require

??

ensure

??

do

→ a[i] := ns

a[i-1] := "junk" → wrong imp.

⇒ $a[i] \sim ns$

F ? ? = T and

F ⇒ ? = T or implies

$\forall j \mid 1 \leq j \leq a.count \bullet j \neq i$ implies old $a[j] \sim a[j]$

J	T	
1	1 /= 3	
2	2 /= 3	T
3	3 /= 3	F
4 5		

$\Rightarrow \forall i \mid 1 \le j \le a.count \bullet \cancel{i \mathrel{/}= j}$ ~~implies~~ $\Rightarrow$

$a[j] \sim$ old $a[j]$

across 1 |..| a.count is j

all

i /= j implies

a[j] ~ old a[j]

→ practice: turn this into a single across

$$\left[\ \forall j \mid 1 \le j \le \overset{2}{a.count} \bullet\ \left(\begin{array}{l} \overset{1}{i} = \overset{\cancel{1}\ 2}{j} \Rightarrow a[j] \sim \overset{\text{"tom"}}{\cancel{ns}} \\ \wedge \\ \overset{1}{i} \,/\!=\, \overset{\cancel{1}\ 2}{j} \Rightarrow \overset{2}{a[j]} \sim old\ \overset{2}{a[j]} \end{array}\right)\right]$$

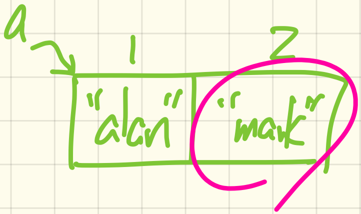
~~k~~	j
	1
	2

✓

F

T

(old)

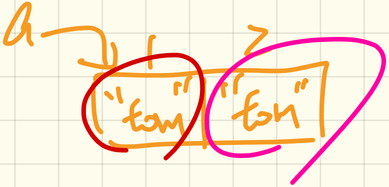


change_at (a , 1 , "tom")

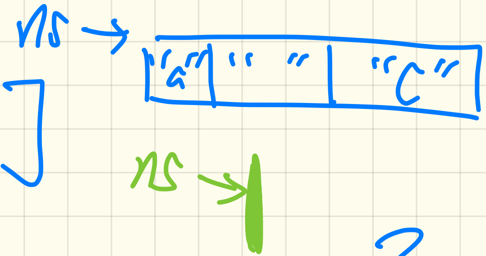
wrong imp:

$a[i] := ns$

$a[i+1] := ns$



ns : ARRAY [STRING]



Are all names in 'ns' non-empty?

V1 :

```
across 1 |..| ns.count is i
all
    not ns[i].is_empty
end
```

V2

```
across ns is n
all
    not n.is_empty
end
```

$$\forall x \mid \underline{\boxed{\text{False}}} \cdot \underline{P(x)} \equiv T$$

$$\exists x \mid \underline{\text{False}} \cdot P(x) \equiv F$$

f

require

. - - -

<u>enfure</u> -

$t1 : p1$

$t2 : p2$

$\vdots$

$tn : Pn$

$t : p1 \wedge p2 \wedge$

$\ldots \wedge Pn$

f

<u>ensure</u>

$t1 : x > y$

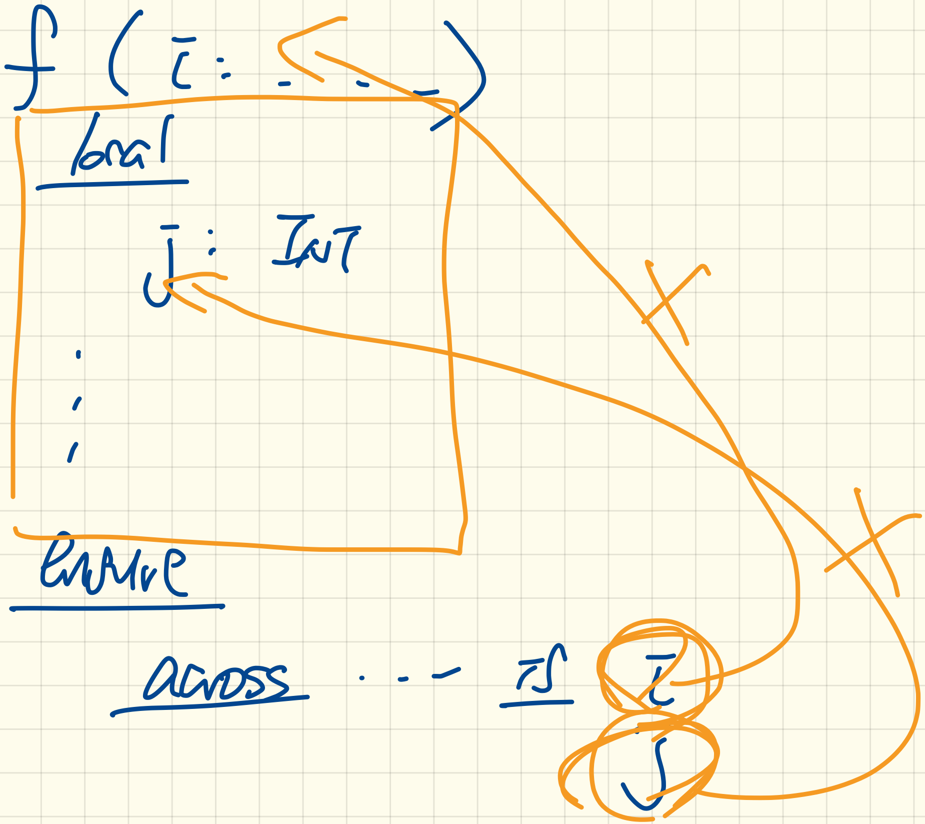
$y > z$

f

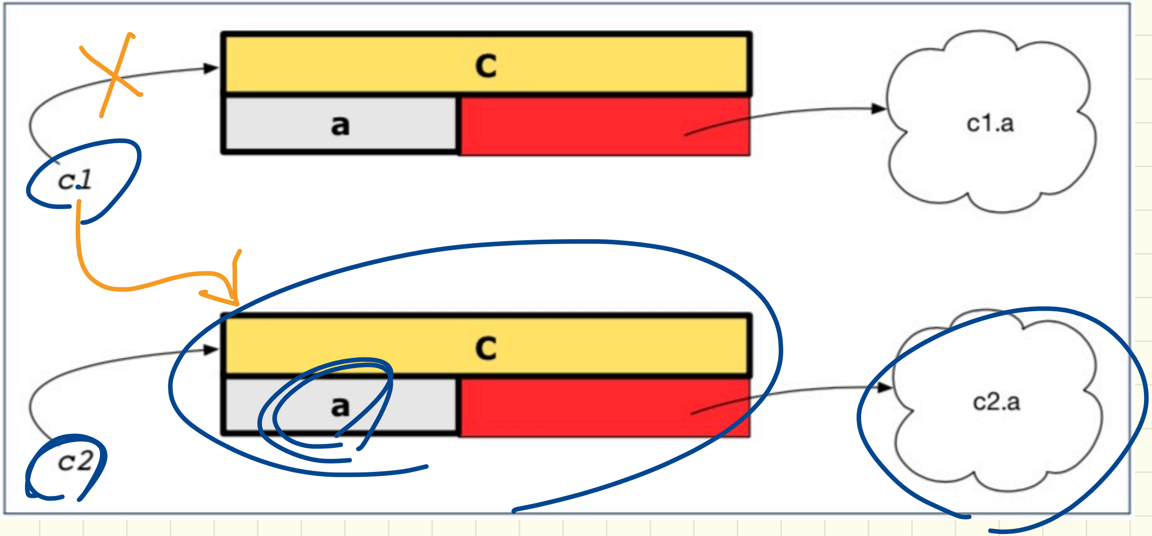
<u>enfure</u>

$t1 : x > y$ <u>and</u>

$y > z$


f (i : ...)
local
j : INT
ensure
across ... is i
j

Reference Copy : $c_1 := c_2$



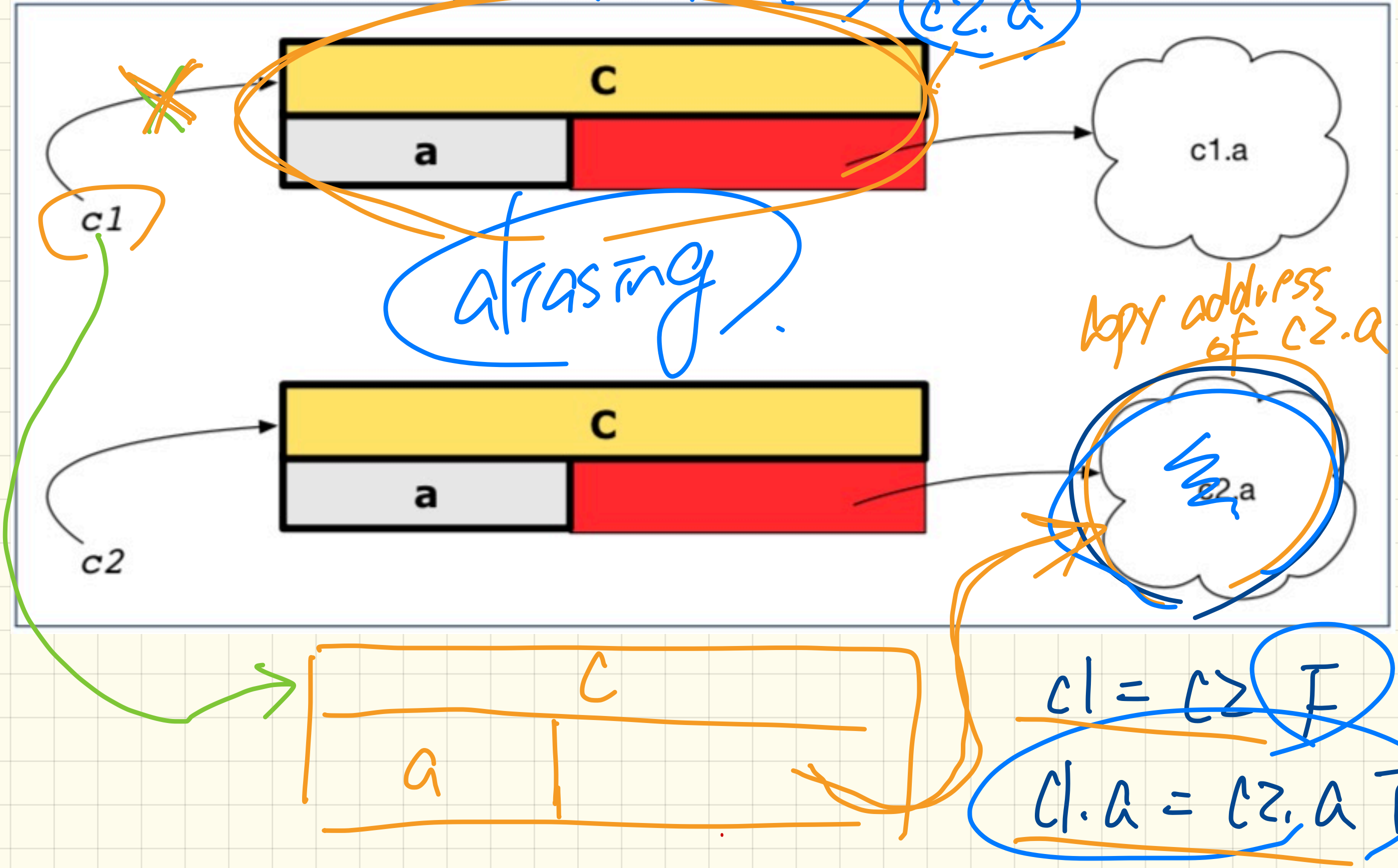
$c1 = c2$ T

$c1.a = c2.a$ T

Shallow Copy : c1 := c2.twin

first-level copy

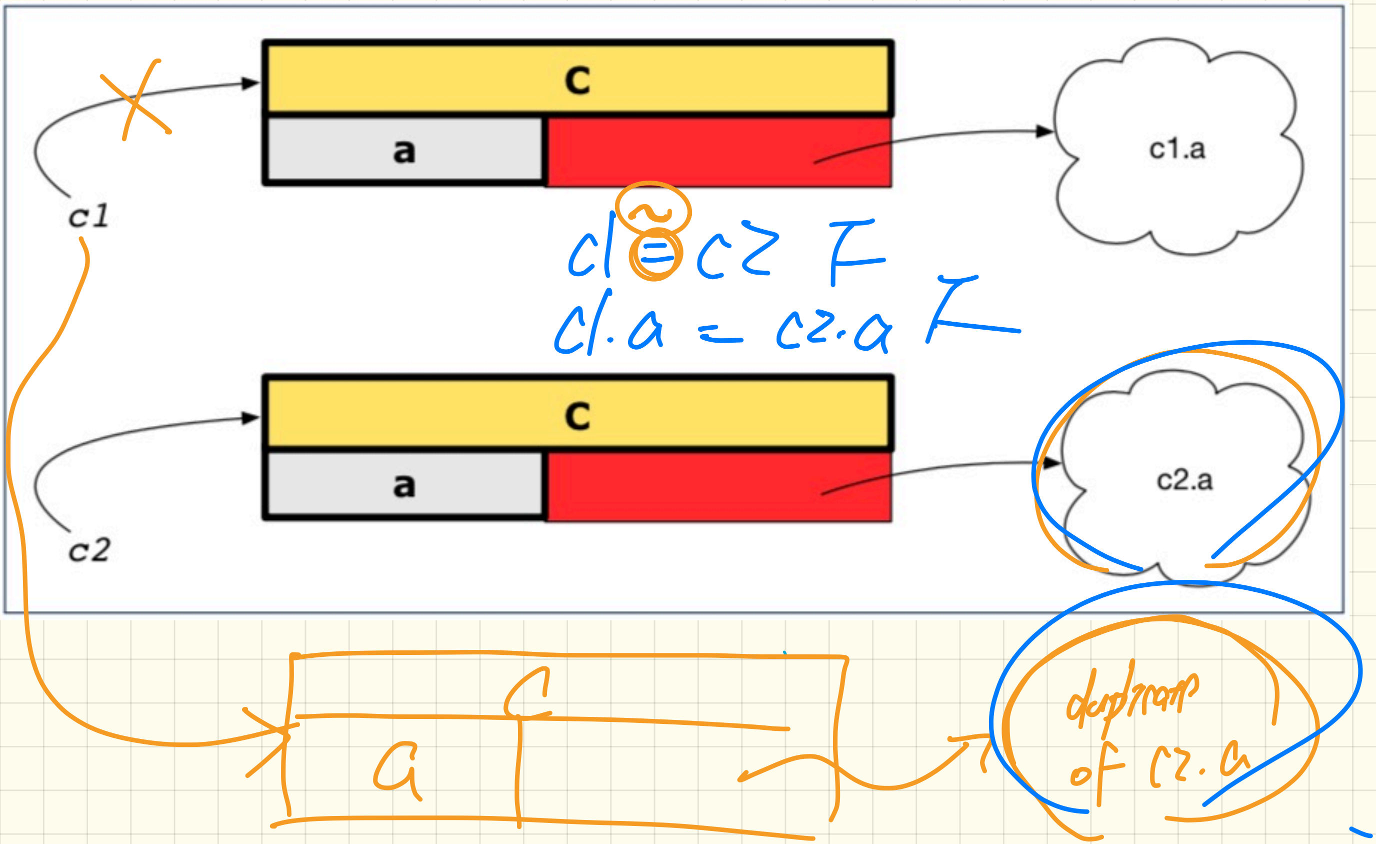
↳ c1.a.set_val(...)



c1 = c2 F

c1.a = c2.a T

Deep Copy : c1 := c2.deep_twin



$c1 \overset{\sim}{=} c2$ F

c1.a = c2.a F

Ref. vs. Shallow vs. Deep Copies

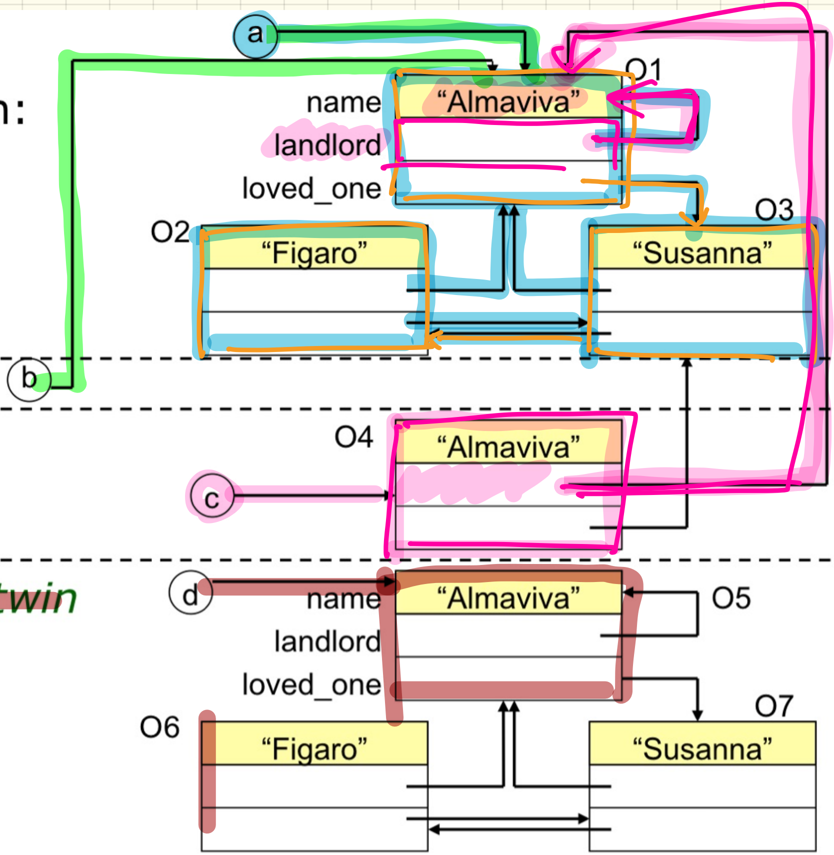
- Initial situation:

- Result of:

b := *a*

c := *a.twin*

d := *a.deep_twin*



LECTURE 4

TUESDAY SEPTEMBER 17

class BANK

like

anchor type

accounts: ~~ARRAY[ACCOUNT]~~

LL

make (new_accounts: like accounts.)

add_accounts (accs : like accounts)

single choice principle

end

Copying Collection Objects : Reference Copy & Make Changes

```
old_imp := imp
Result := old_imp = imp   -- Result = [redacted]
imp[2] := "Jim"
Result :=
  across 1 |..| imp.count is j
  all imp [j] ~ old_imp [j]
  end -- Result = [redacted]
```

2

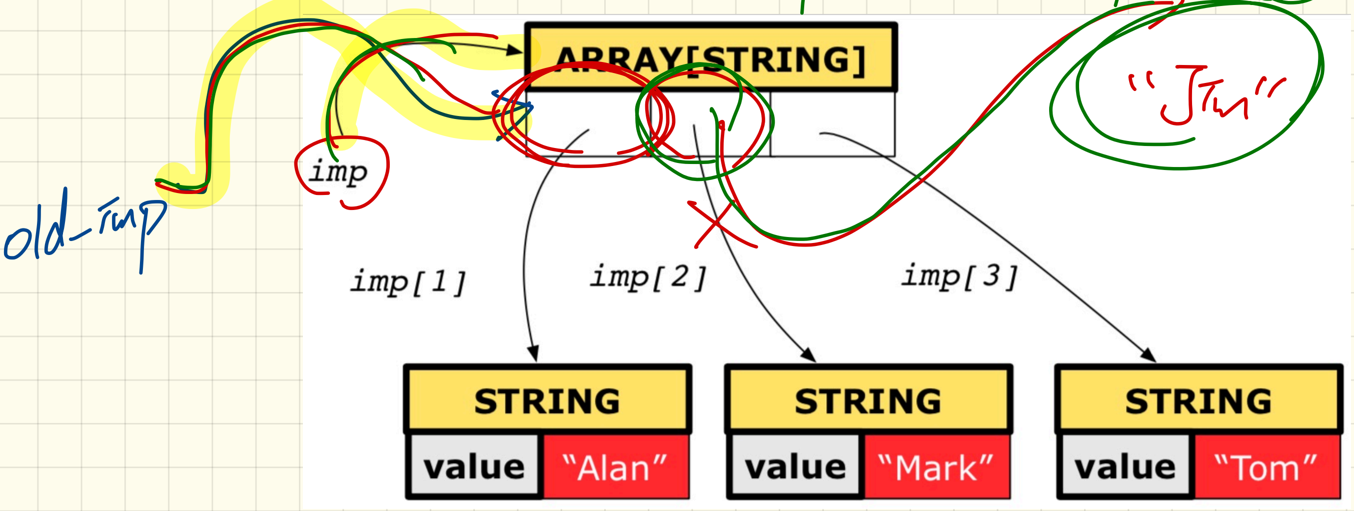
1

imp[1] = old_imp[1] T

imp[2] ~ old_imp[2] T

not good :(

we expect to see two versions being diff.



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

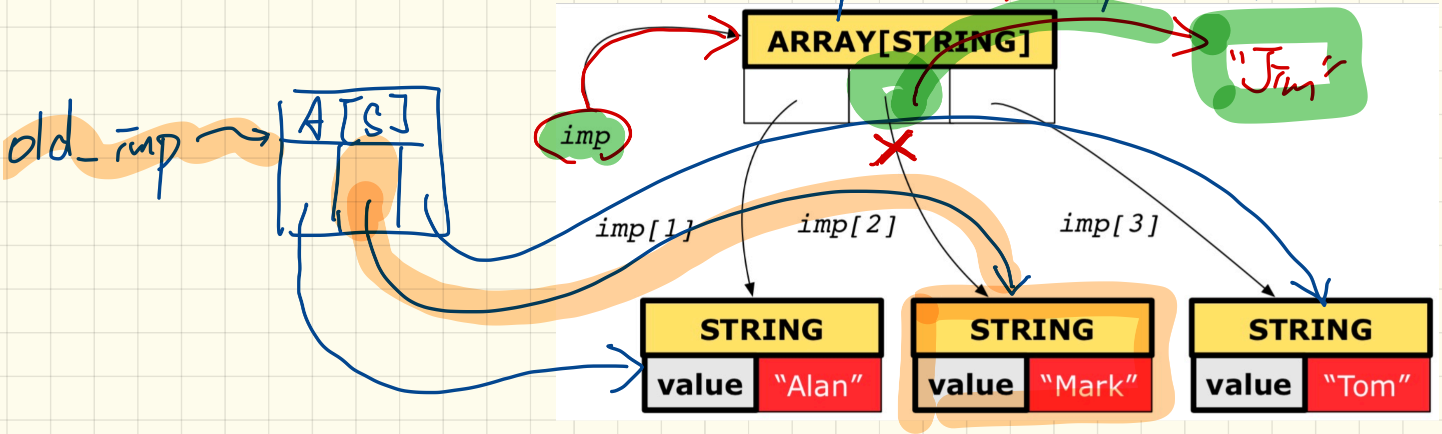
```
old_imp := imp.twin
Result := old_imp = imp   -- Result = false
imp[2] := "Jim"
Result :=
  across 1 |..| imp.count is j
  all imp [j] ~ old_imp [j]
  end -- Result = [redacted]
```

2

imp[1] ~ old_imp[1] (T)

imp[2] ~ old_imp[2] (F)

old_imp[1] = imp[1] (T)



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

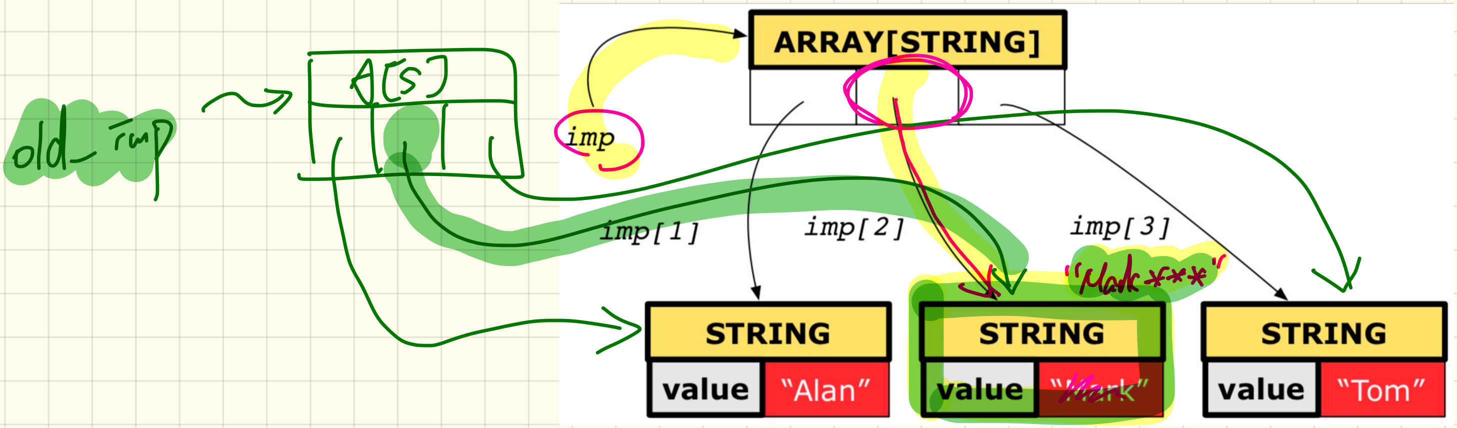
V3

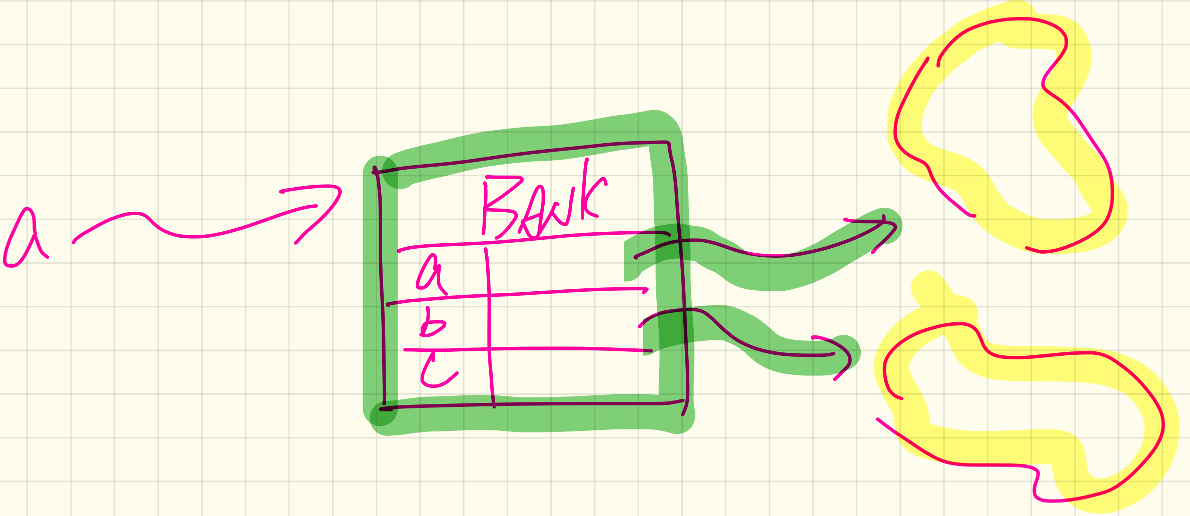
```
old_imp := imp.twin
Result := old_imp = imp  -- Result = false
imp[2].append ("***")
Result :=
  across 1 |..| imp.count is j
  all imp [j] ~ old_imp [j]
  end -- Result = [redacted]
```

2

imp[2] ~ old_imp[2]

T



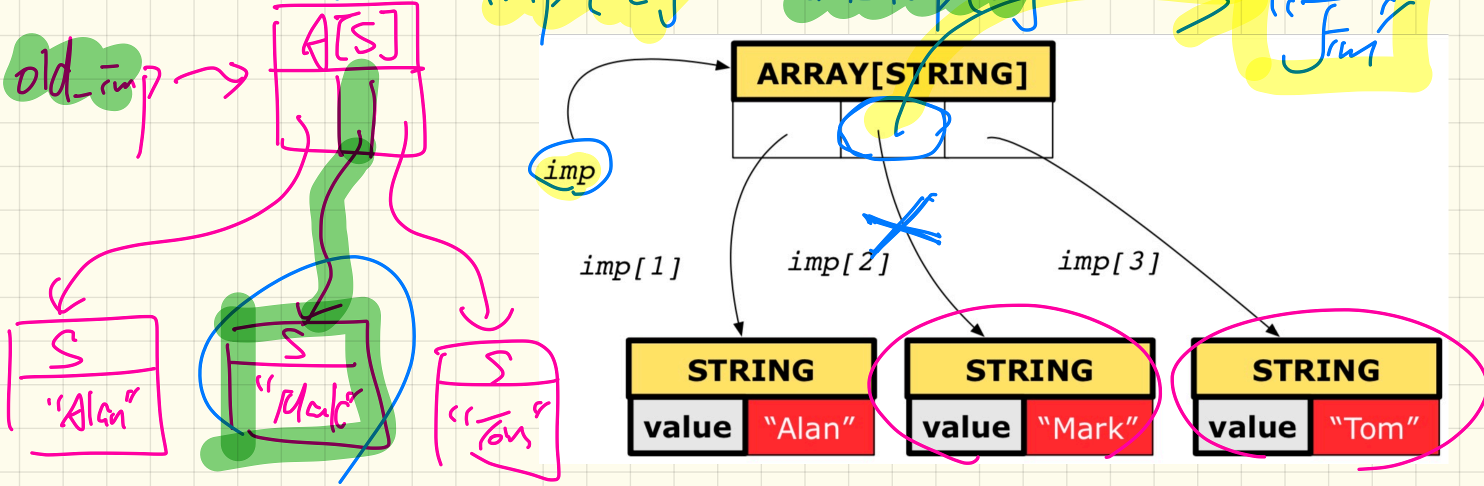

a
BANK
a
b
c

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

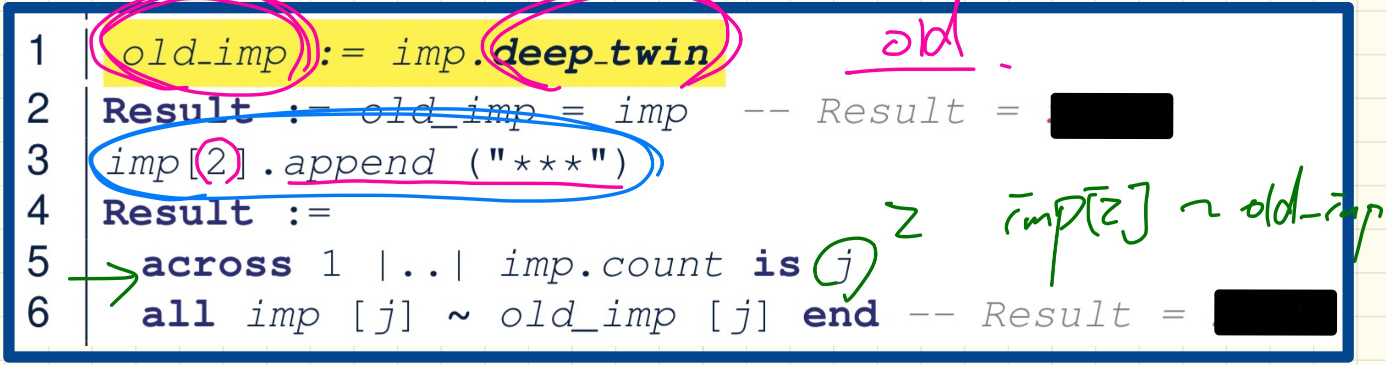
```
old_imp := imp.deep_twin
Result := old_imp = imp   -- Result = false
imp[2] := "Jim"
Result :=
  across 1 |..| imp.count is j
  all imp [j] ~ old_imp [j] end -- Result = [redacted]
```

→ F ?!

imp[2] ~ old_imp[2]



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

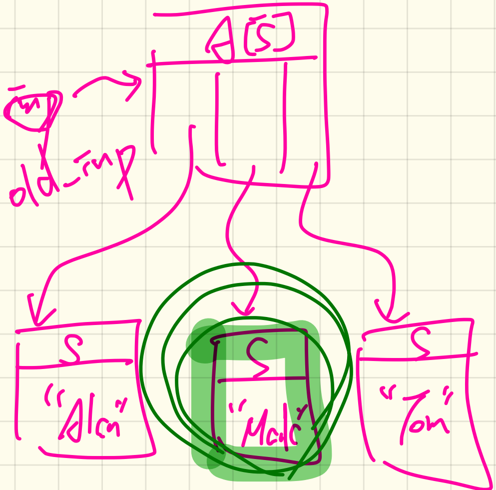


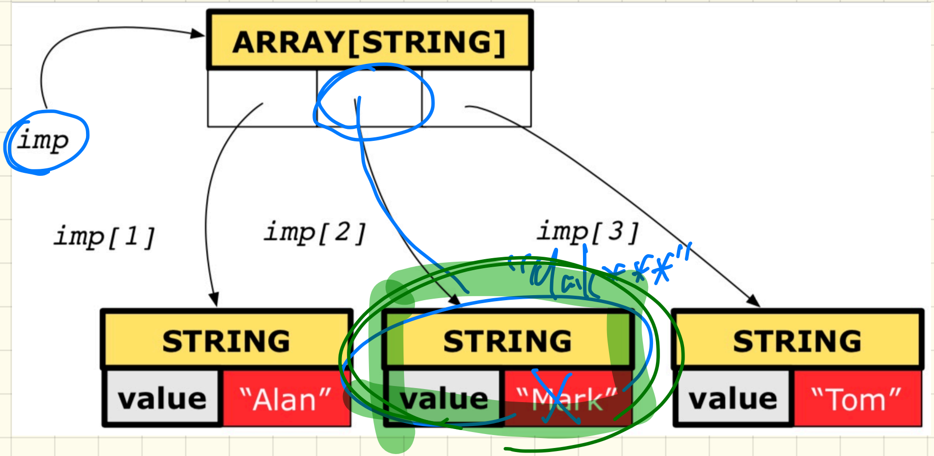
```
old_imp := imp.deep_twin
Result := old_imp = imp   -- Result = [redacted]
imp[2].append ("***")
Result :=
  across 1 |..| imp.count is j
  all imp [j] ~ old_imp [j] end -- Result = [redacted]
```

old

2

imp[2] ~ old_imp[2]





require → expecting boolean expression.

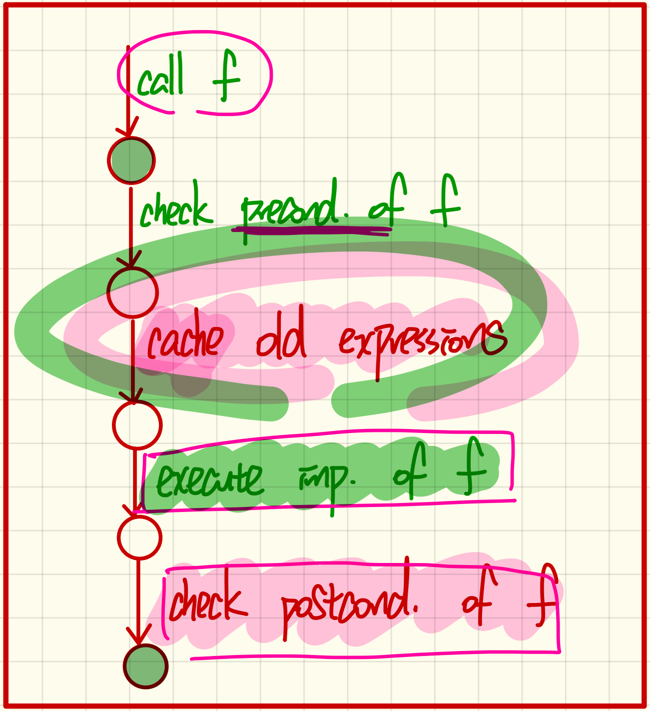
$x := y$

:= → assignment

Contract View

```
f
    require
        ...
    do
        ...
    ensure
        ... old expr ...
    end
```

Runtime Contract Checks



Caching Values for **old** Expressions in Postconditions

ensure (in context of **BANK**)

① **old** accounts[i].id INT

② (**old** accounts[i]).id

③ (**old** accounts[i].**twin**).id

④ (**old** accounts)[i].id

⑤ (**old** accounts.**twin**)[i].id

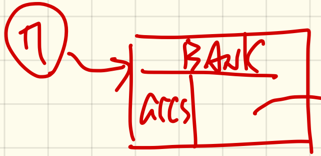
⑥ (**old** **Current**).accounts[i].id

⑦ (**old** **Current**.**twin**).accounts[i].id

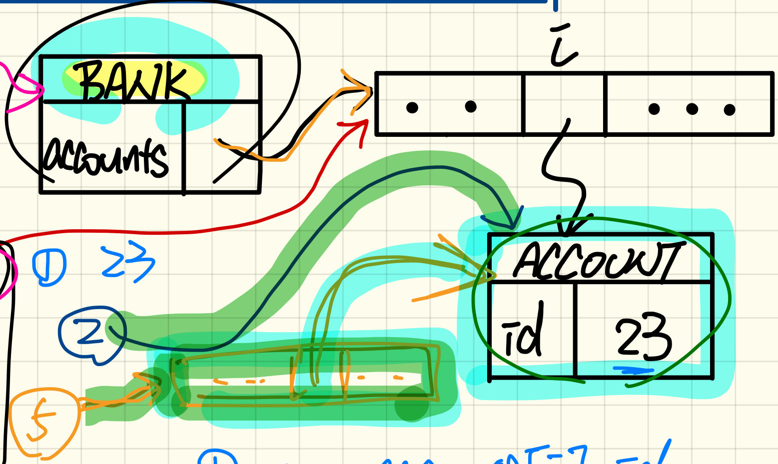
shallow copy of account

class BANK
:
ensure
end

shallow copy of array



How to Cache at Runtime?



① := accounts[i].id

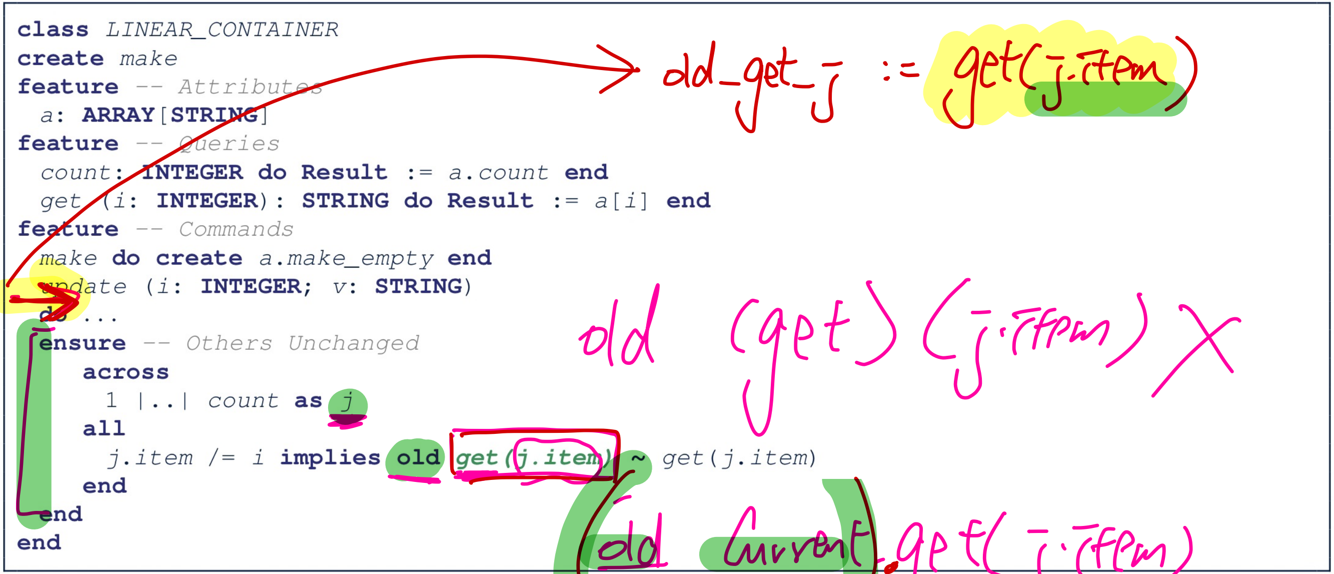
② := accounts [i]

⑤ := accounts . twin

⑦ := Current. twin

⑥ := Current

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
```

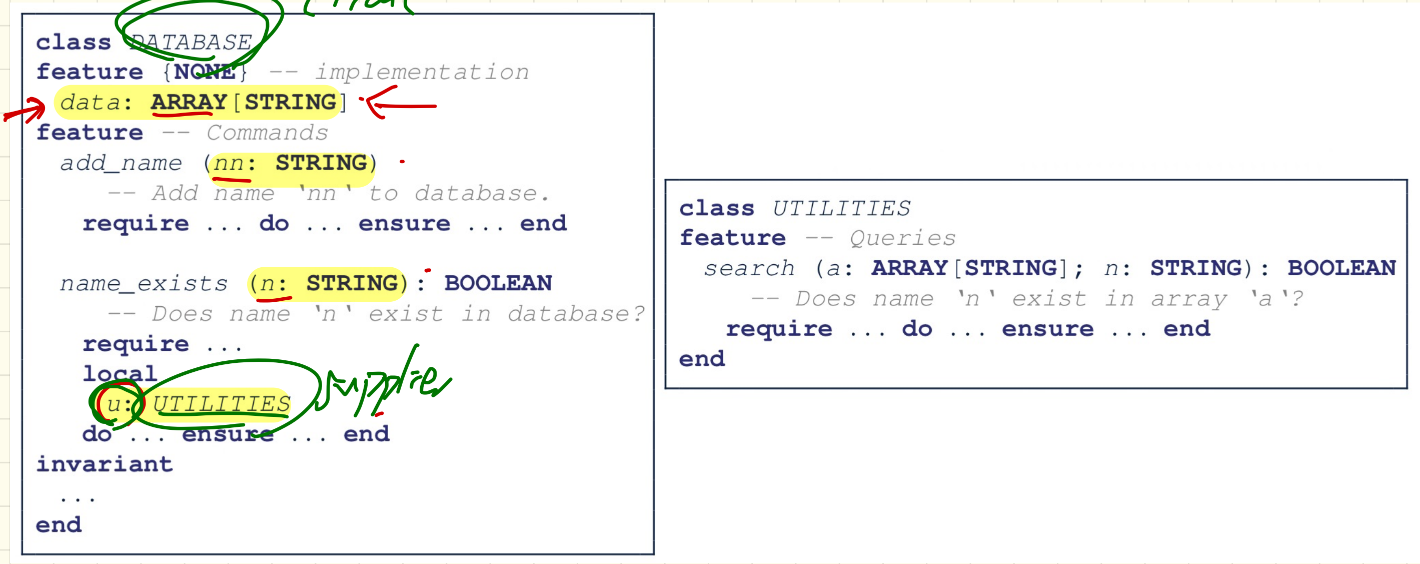
old_get_j := get(j.item)

old (get) (j.item) X

(old Current).get(j.item)

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

Programming Client-Supplier Relation



client

```
class DATABASE
feature {NONE} -- implementation
  data: ARRAY[STRING]
feature -- Commands
  add_name (nn: STRING)
      -- Add name 'nn' to database.
    require ... do ... ensure ... end

  name_exists (n: STRING): BOOLEAN
      -- Does name 'n' exist in database?
    require ...
    local
      u: UTILITIES
    do ... ensure ... end
invariant
  ...
end
```

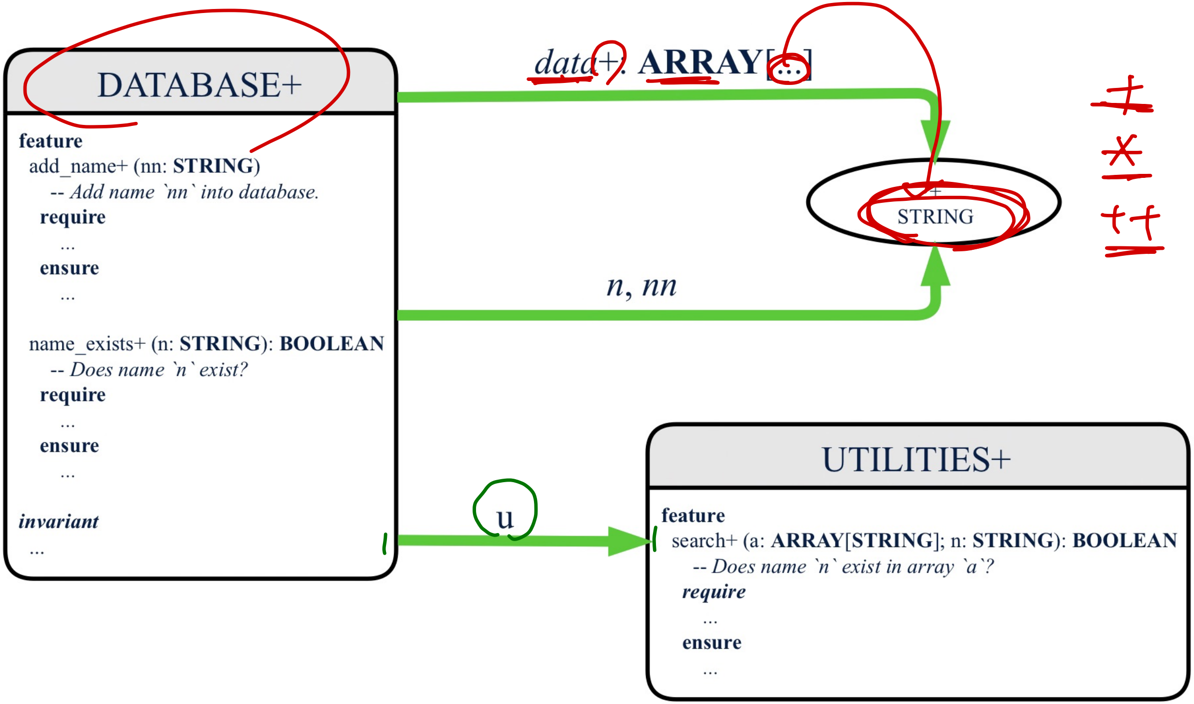
supplier

```
class UTILITIES
feature -- Queries
  search (a: ARRAY[STRING]; n: STRING): BOOLEAN
      -- Does name 'n' exist in array 'a'?
    require ... do ... ensure ... end
end
```

data : ARRAY [STRING]

Presenting CS Relation in Diagram: Approach 1

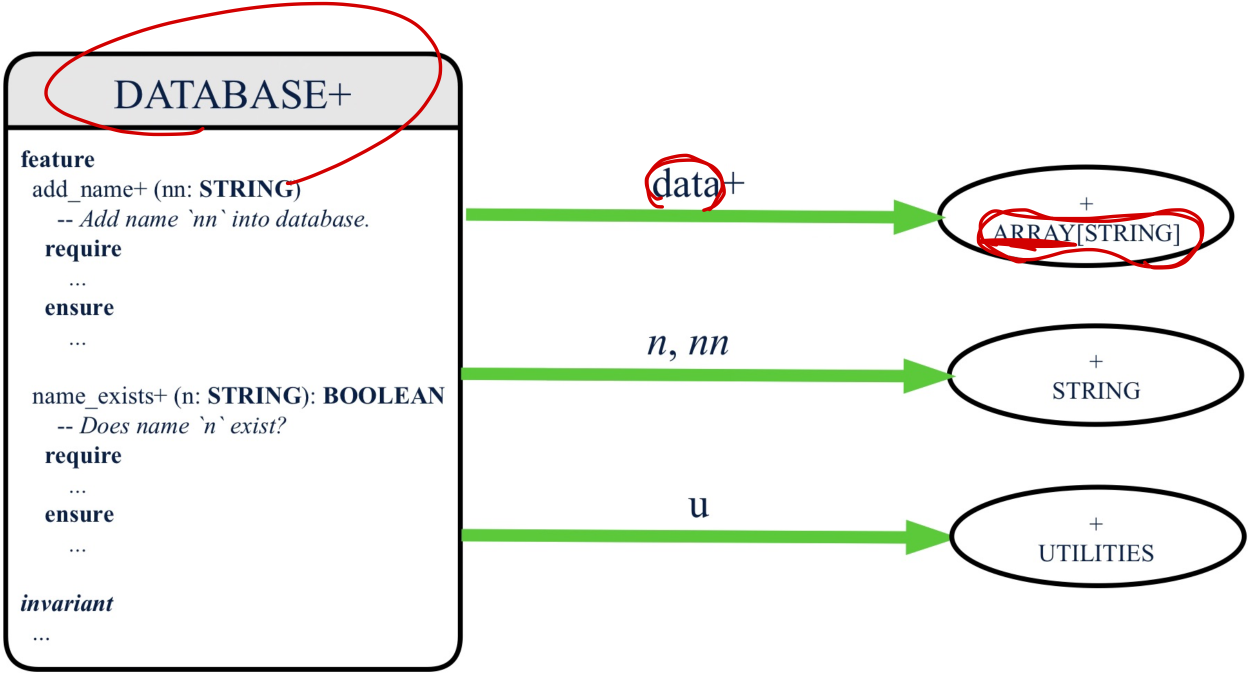
If `STRING` is to be emphasized, label is `data: ARRAY[...]`, where ... denotes the supplier class `STRING` being pointed to.



Presenting CS Relation in Diagram: Approach 2

If `ARRAY` is to be emphasized, label is `data`.

The supplier's name should be complete: `ARRAY[STRING]`



LECTURE 5

THURSDAY SEPTEMBER 19

Caching Values for **old** Expressions in Postconditions

ensure (in context of **BANK**)

① **old** accounts[i].id INT

② (**old** accounts[i]).id

③ (**old** accounts[i].**twin**).id shallow copy of account

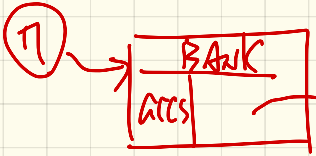
④ (**old** accounts)[i].id

⑤ (**old** accounts.**twin**)[i].id

⑥ (**old** **Current**).accounts[i].id

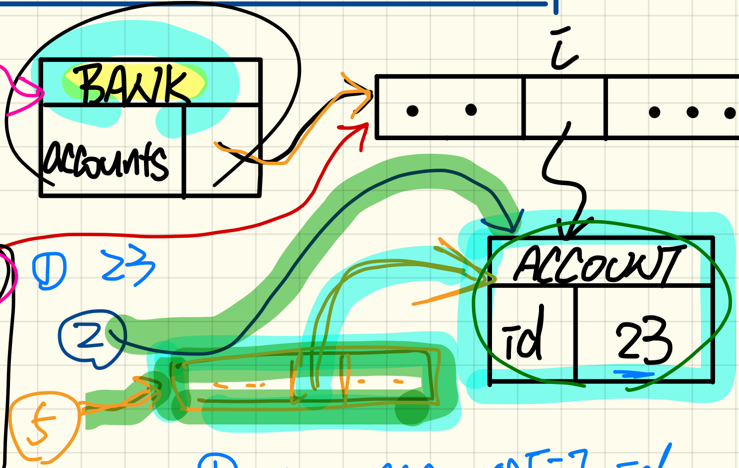
⑦ (**old** **Current**.**twin**).accounts[i].id

class BANK
:
ensure
ad



shallow copy of array

How to Cache at Runtime?



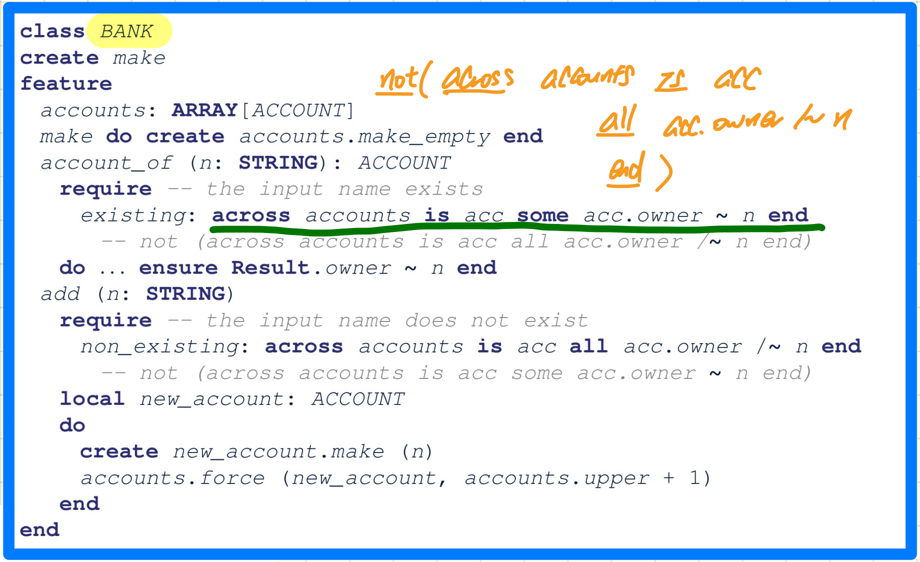
① := accounts[i].id

② := accounts [i]

⑤ := accounts . twin

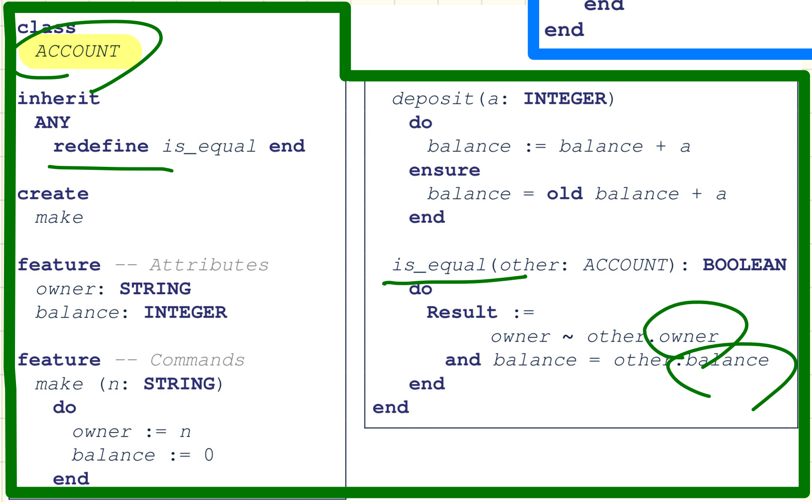
⑦ := Current . twin

⑥ := Current



```
class BANK
create make
feature
  accounts: ARRAY[ACCOUNT]
  make do create accounts.make_empty end
  account_of (n: STRING): ACCOUNT
    require -- the input name exists
      existing: across accounts is acc some acc.owner ~ n end
        -- not (across accounts is acc all acc.owner /~ n end)
    do ... ensure Result.owner ~ n end
  add (n: STRING)
    require -- the input name does not exist
      non_existing: across accounts is acc all acc.owner /~ n end
        -- not (across accounts is acc some acc.owner ~ n end)
    local new_account: ACCOUNT
    do
      create new_account.make (n)
      accounts.force (new_account, accounts.upper + 1)
    end
end
```

not(across accounts is acc
all acc.owner /~ n
end)

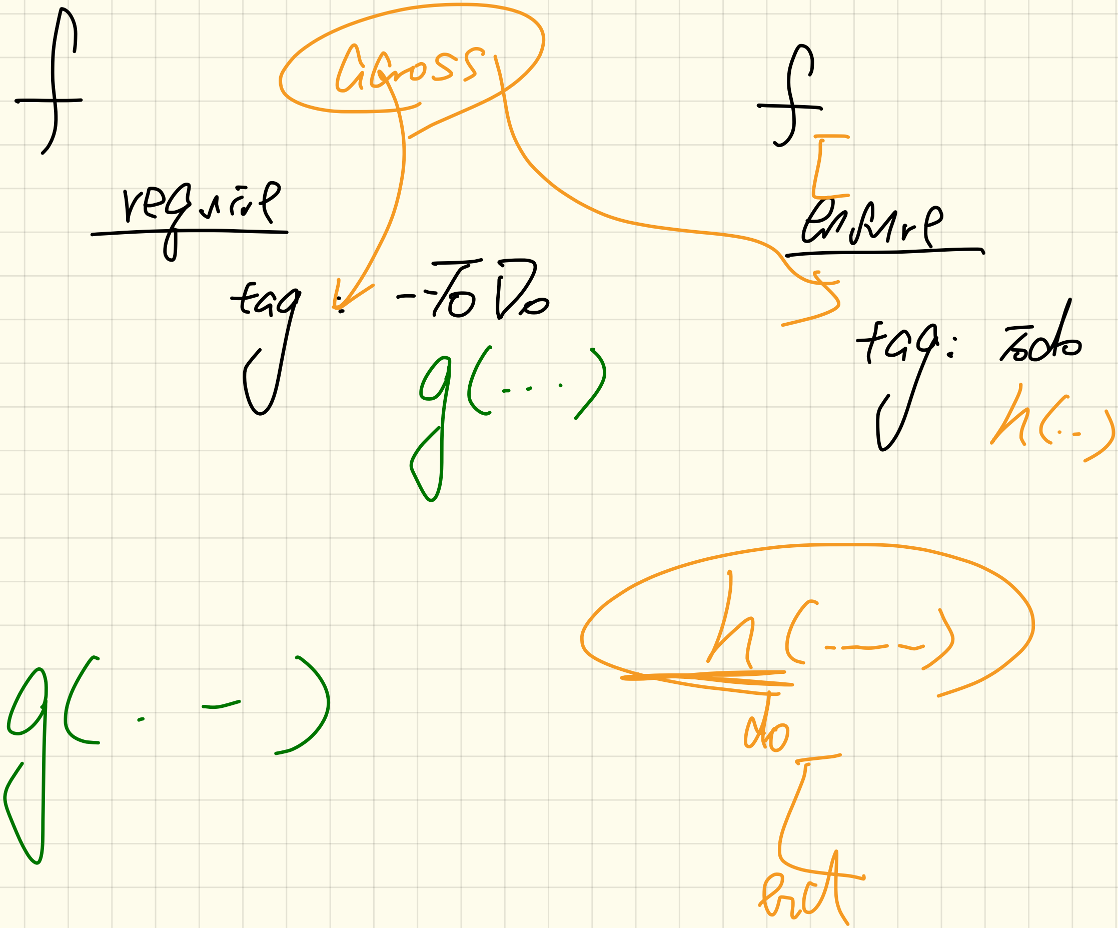


```
class
  ACCOUNT
inherit
  ANY
    redefine is_equal end

create
  make

feature -- Attributes
  owner: STRING
  balance: INTEGER

feature -- Commands
  make (n: STRING)
    do
      owner := n
      balance := 0
    end
```

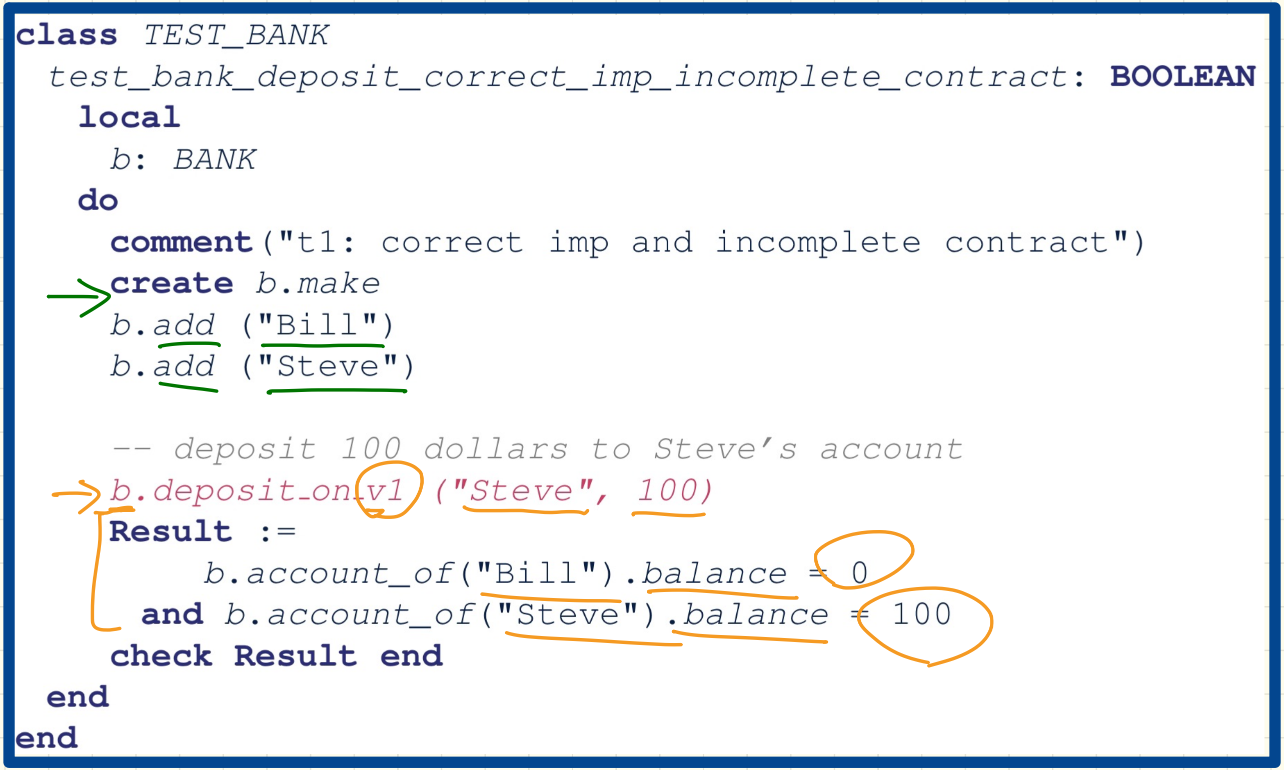
```
  deposit(a: INTEGER)
    do
      balance := balance + a
    ensure
      balance = old balance + a
    end

  is_equal(other: ACCOUNT): BOOLEAN
    do
      Result :=
          owner ~ other.owner
        and balance = other.balance
    end
end
```

$$\forall x : R(x) \bullet P(x)$$

$$\equiv \neg (\exists x : R(x) \neg P(x))$$


f
across
f
require
tag: ToDo
g(...)
ensure
tag: ToDo
h(..)
g(..)
h(...)
do
end

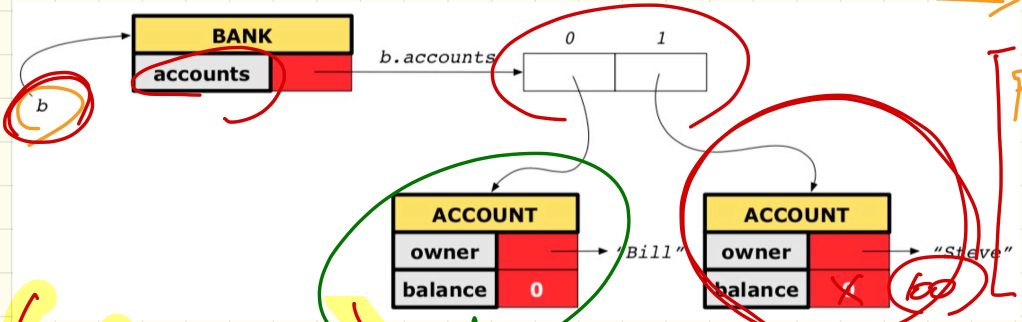
Unit Test for All 5 Versions



```
class TEST_BANK
  test_bank_deposit_correct_imp_incomplete_contract: BOOLEAN
    local
      b: BANK
    do
      comment("t1: correct imp and incomplete contract")
      create b.make
      b.add ("Bill")
      b.add ("Steve")

      -- deposit 100 dollars to Steve's account
      b.deposit_on_v1 ("Steve", 100)
      Result :=
           b.account_of("Bill").balance = 0
        and b.account_of("Steve").balance = 100
      check Result end
  end
end
```

Version 1: Incomplete Contracts, Correct Implementation

b.deposit("Steve", 100)



pre-state: 2

old_count := accounts.count

old_balance := Current.a_o(n).b 0

(old Current.a_o(n)).balance

Steve 100

```
class BANK
  deposit_on_v1 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.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:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
    end
end
```

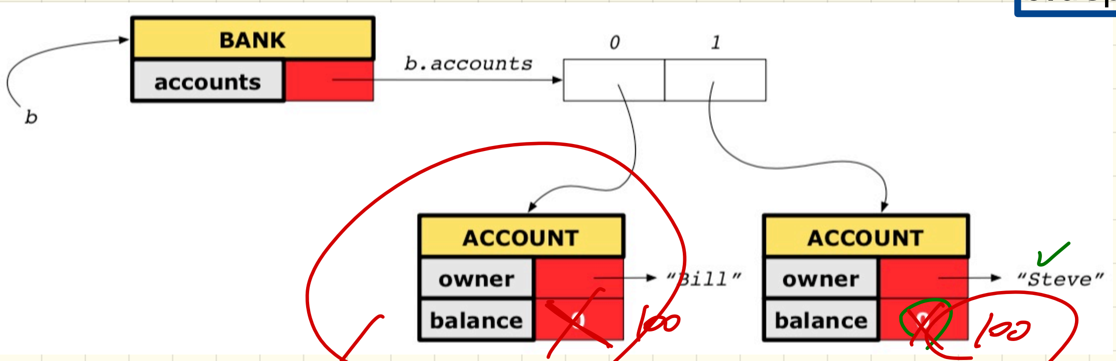
2 2 T

100 0 100 T

Incomplete
-: It talks
nothing about
all other
accounts.

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)



```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
    end
end
```

2

2

T

100

0

0

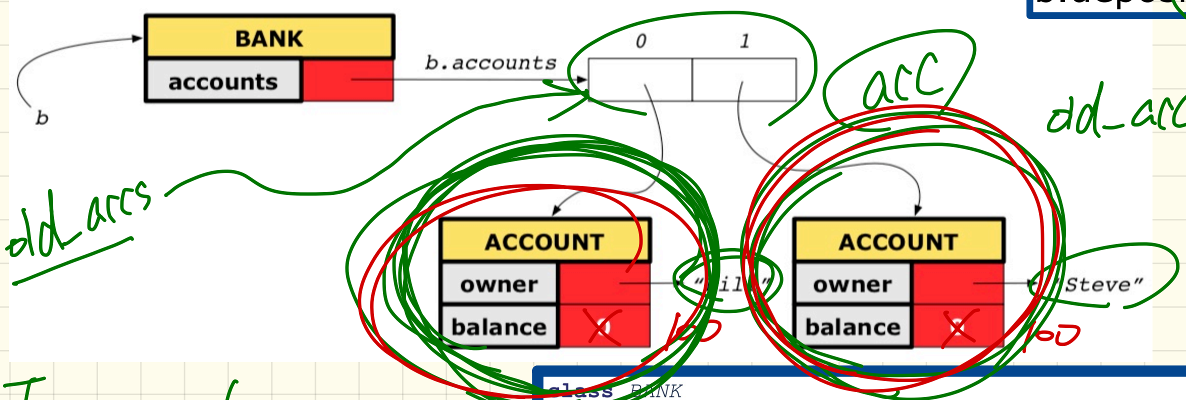
100

T

how this object (rest of accounts) should be not changed addressed in postcond.

Version 3: Complete Contracts (Ref. Copy), Correct Implementation

b.deposit("Steve", 100)



old_accs := accounts

```
class BANK
  deposit_on_v3 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
      others_unchanged :
        across old accounts is acc
        all
          acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
        end
    end
end
```

→ 2 iterations

"Bill"

"Steve"

Iteration 1

"Bill" /~ "Steve" implies

↳ true simply because we considered the object equal to itself.

Iteration 2

T F

"Steve" /~ "Steve" implies

Use of across in Postcondition

Version 1

```
across old accounts is acc
all
   acc.owner /~ n
   implies
   acc ~ Current.account_of (acc.owner)
end
```

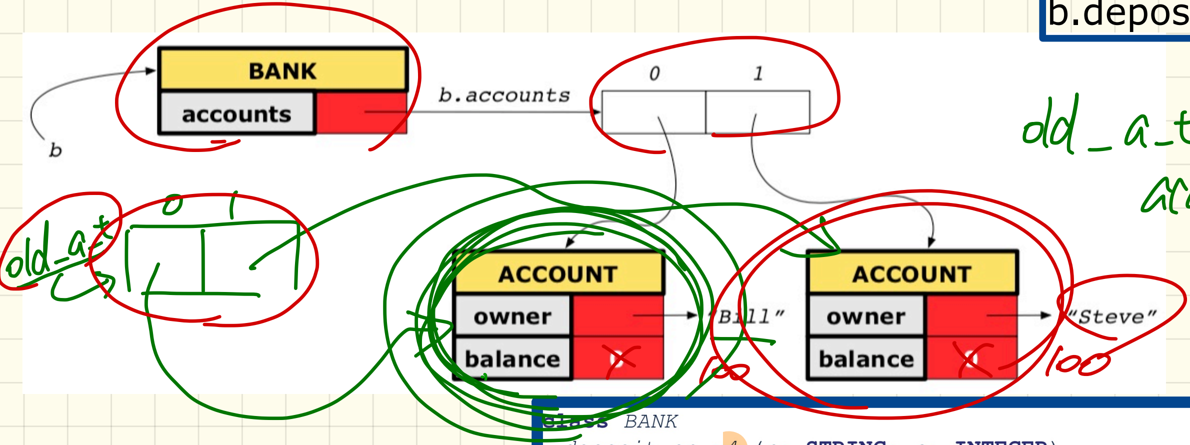
Version 2

```
across (old accounts.lower |..| old accounts.upper) is i
all
   (old accounts)[i].owner /~ n
   implies
   (old accounts)[i] ~ Current.account_of ( (old accounts)[i].owner )
end
```

Version 4: Complete Contracts (Shallow Copy), Correct Implementation

b.deposit("Steve", 100)

old_a_t := accounts.twin



```
class BANK
 deposit_on_v4 (n: STRING; a: INTEGER)
  require across accounts is acc some acc.owner ~ n end
  local i: INTEGER
  do ...
   -- imp. of version 1, followed by a deposit into 1st account
   accounts[accounts.lower].deposit(a)
  ensure
   num_of_accounts_unchanged: accounts.count = old accounts.count
   balance_of_n_increased:
    Current.account_of(n).balance =
     old Current.account_of(n).balance + a
   others_unchanged :
    across old accounts.twin is acc
    all
     acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
    end
  end
end
```

same object

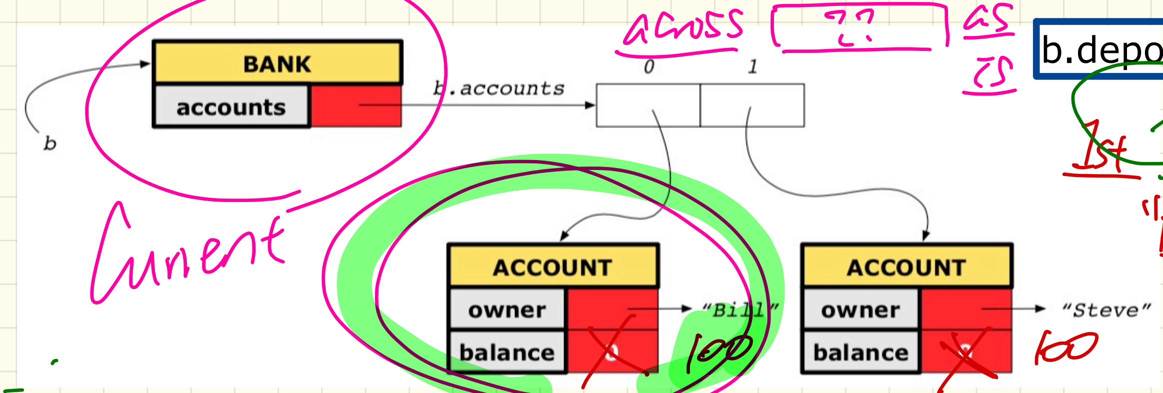
1st iteration

"Bill" /~ "Steve" ⇒ ✓

2nd iteration

"Steve" /~ "Steve" ⇒

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



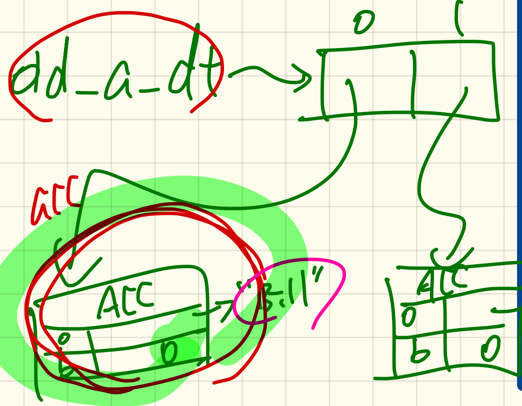
across [??] as
is

b.deposit("Steve", 100)

1st ~

"Bill" /\ "free" =>

old_a_dt := accounts.deep_twin

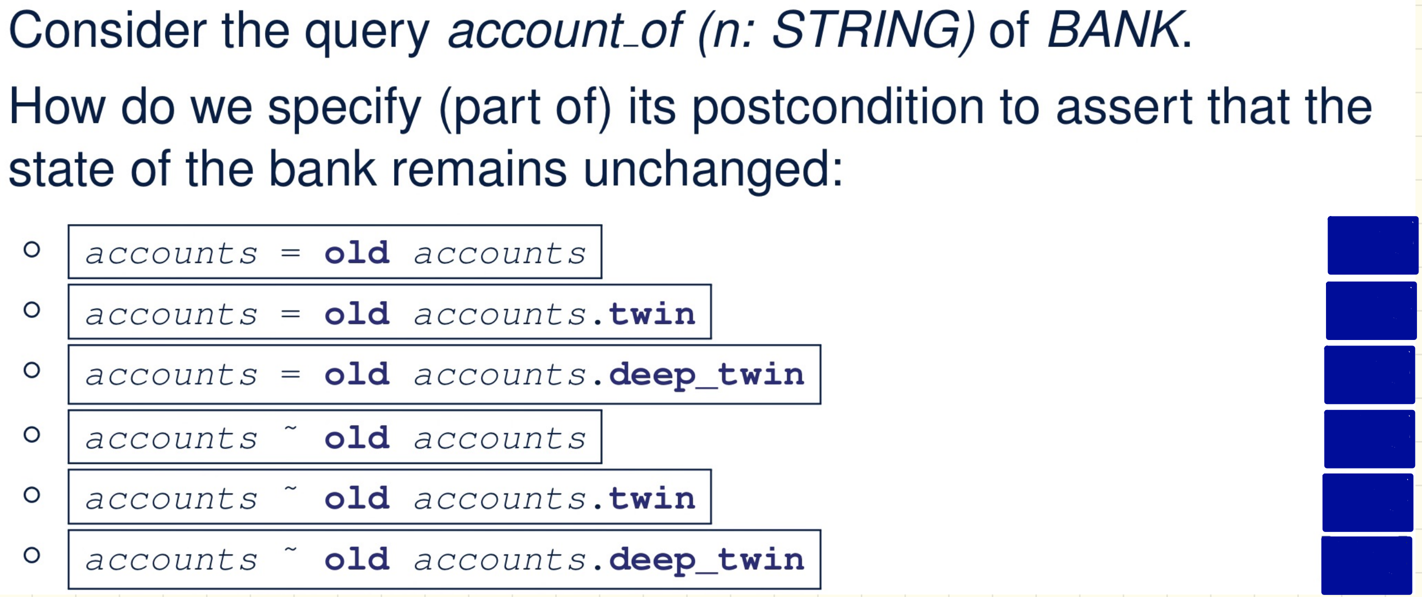


```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
      local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
      others_unchanged :
        across old accounts.deep_twin is acc
        all
          acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
        end
    end
end
```

"Bill"

Complete Postcondition: Exercise

[redacted]



Consider the query *account_of (n: STRING)* of *BANK*.

How do we specify (part of) its postcondition to assert that the state of the bank remains unchanged:

○ `accounts = old accounts` [redacted]

○ `accounts = old accounts.twin` [redacted]

○ `accounts = old accounts.deep_twin` [redacted]

○ `accounts ~ old accounts` [redacted]

○ `accounts ~ old accounts.twin` [redacted]

○ `accounts ~ old accounts.deep_twin` [redacted]

Writing Postcondition: Exercise 1.1

is_positive (i: **INTEGER**): **BOOLEAN**
    ensure
tag: $\overset{i}{\cancel{x}} > 0$

is_p(3)

is_p(−4) ← Result correctly set to false but a postcond_ violation

Writing Postcondition: Exercise 1.2

```
is_positive (x: INTEGER): BOOLEAN
  ensure
    if x > 0 then
      Result = True
  end
```

Syntax

else
  not Result
end

$x > 0$ implies Result

$\neg(x > 0)$ implies not Result

LECTURE 6

TUESDAY SEPTEMBER 24

- LAB TEST 1

GUIDE

PRACTICE QUESTIONS

- Lab 2

Writing Postcondition: Exercise 2

```
a: ARRAY[STRING]
change_at (i: INTEGER; s: STRING)
  ensure
    across a.lower |..| a.upper is j
      all
        j = i implies a[j] ~ s
        and
        j /= i implies a[j] ~ (old a.deep_twin)[j]
    end
```

2

4

4

a ~ old a.deep_twin

a.count = old a.count

(old a.d_t)[4]

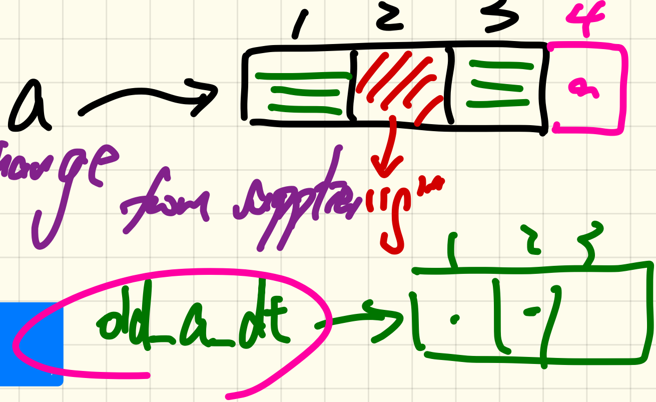
AIOB violation

Complete Postcondition?

not appropriate ∵ it allows no change from applying

What if a.count > **old** a.count?

What if a.count < **old** a.count?



Writing Postcondition: Exercise 3

Pos : positive numbers in 'a'

```
all_positive_values (a: ARRAY[INTEGER]): ARRAY[INTEGER]
    require
        no_duplicates: ??
    ensure
        across Result is x
            all
                x > 0
            end
```

$a \sim$ old a.deep_twin

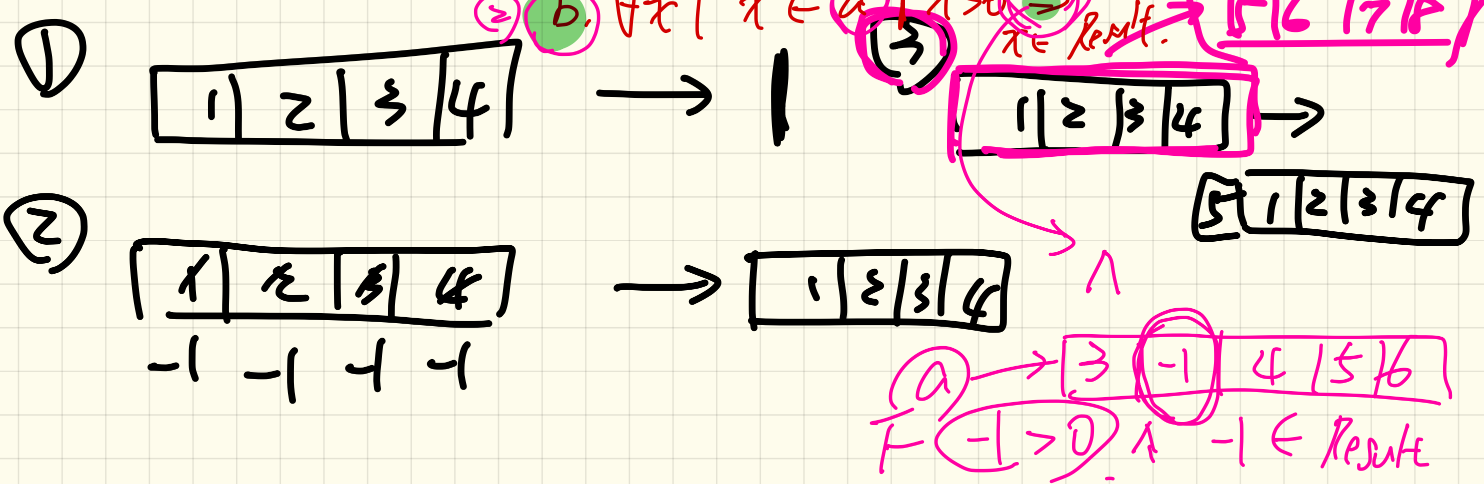
Pos ⊆ Result

Result ⊆ Pos

① $\forall x \mid x \in Result \mid x > 0 \land x \in a$

a. # of pos #s in a = Result.count

② b. $\forall x \mid x \in a \mid x > 0 \Rightarrow x \in Result$.



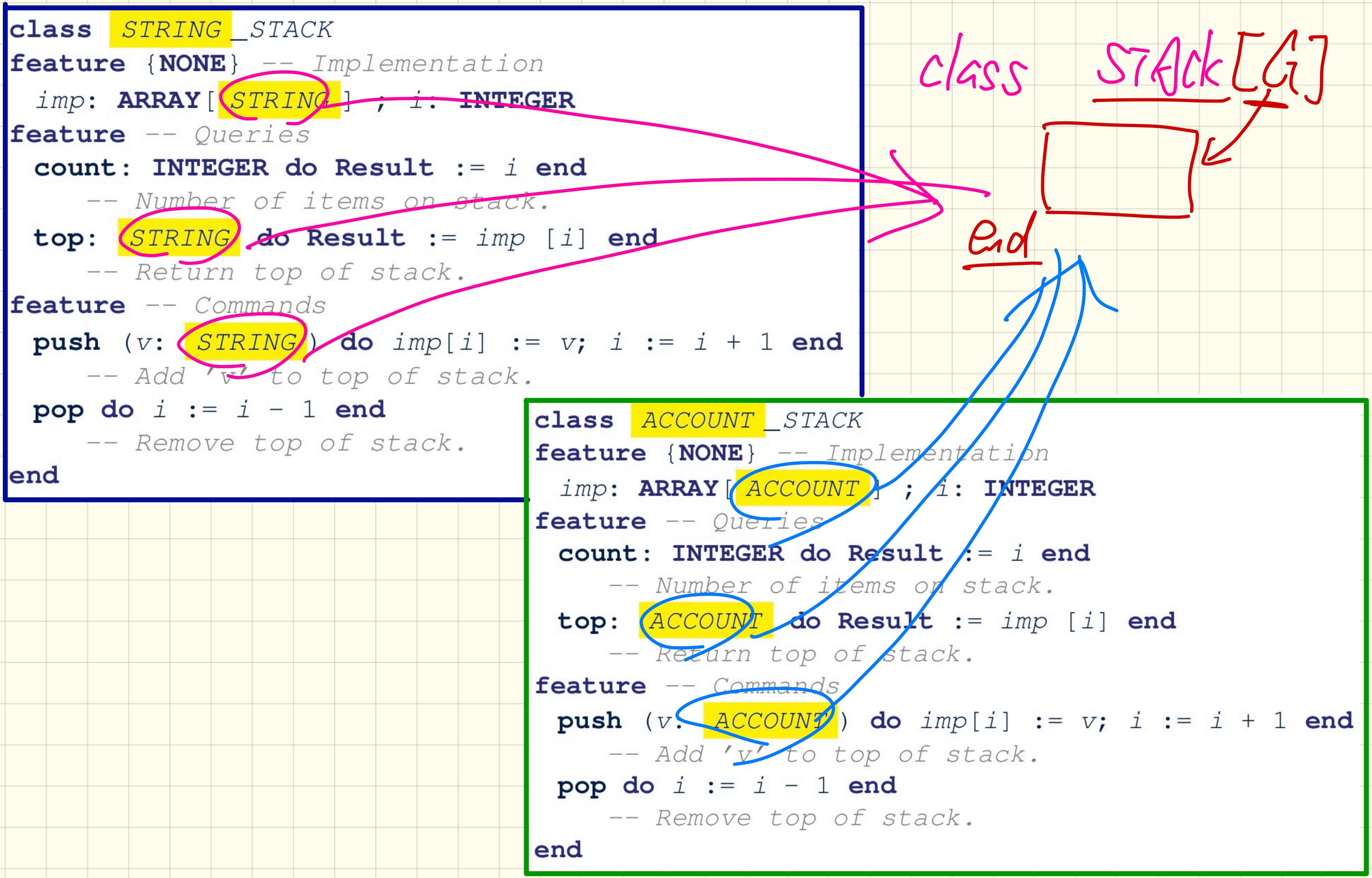
1. how the collection works

LIFO

2. type of collection members

(STRING)

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
```

```
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
```

class STACK[G]

end

A Generic Stack

Supplier

```
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
```

A S

declare

depends on what G is instantiated into.

Client

```
test_stacks: BOOLEAN
  local
    ss: STACK[STRING] ; sa: STACK[ACCOUNT]
    s: STRING ; a: ACCOUNT
  do
    ss.push("A")
    ss.push(create {ACCOUNT}.make ("Mark", 200))
    s := ss.top
    a := ss.top
    sa.push(create {ACCOUNT}.make ("Alan", 100))
    sa.push("B")
    a := sa.top
    s := sa.top
  end
```

Java

class Collection < E extends Comparable >

Eiffel

c : Collection < ___ >

?? any descendant class of Comparable.

class COLLECTION [E -> COMPARABLE]

Principle of Information Hiding

1. As supplier, you should be free to change imp. strategy.

Supplier:

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

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

orders: ~~ITERABLE~~[ORDER]

ITERABLE

LINKED_LIST

client's code won't compile.

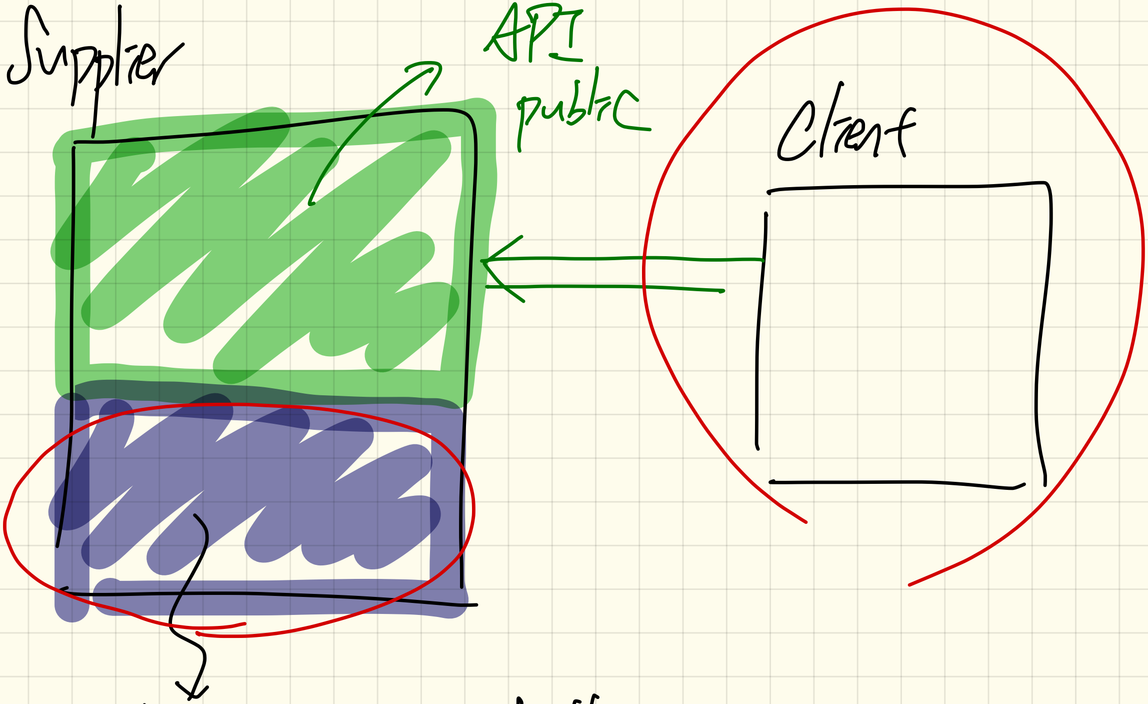
Client:

```
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
```

✓

2. When you change imp. strategy, all existing clients should NOT be affected

Problems?

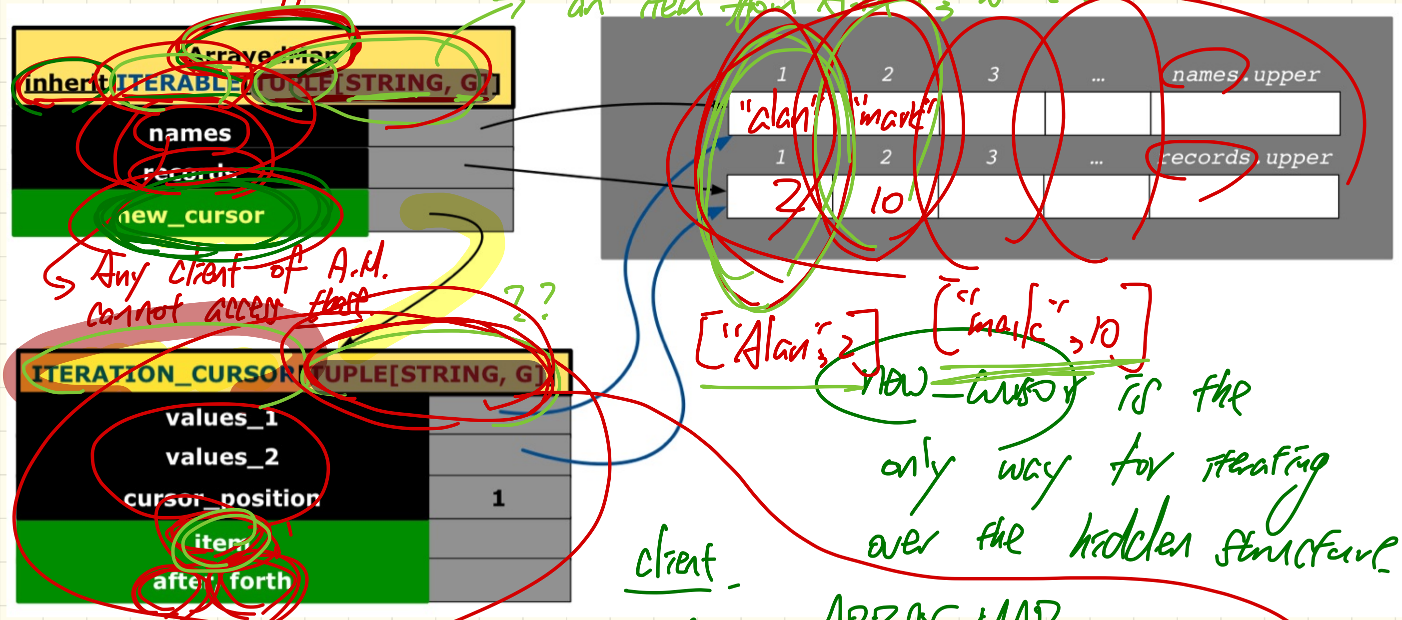


design secrete that's hidden.

Iterator Pattern at Runtime

feature {NONE} item's type?

when retrieving an item from A.M. → what should be the



→ Any client of A.M. cannot access these

["Alan", 2] ["mark", 10]

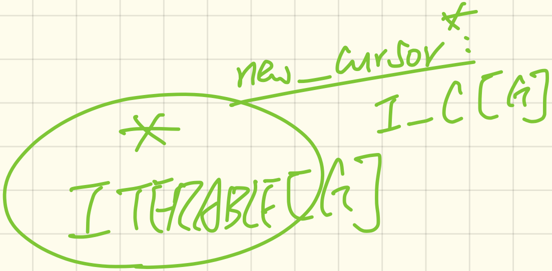
new_cursor is the only way for iterating over the hidden structure

client

```
am : ARRAY_MAP
from
    cursor := am.new_cursor
until
    cursor.after
loop
    print(cursor.item)
    cursor.forth
end
```

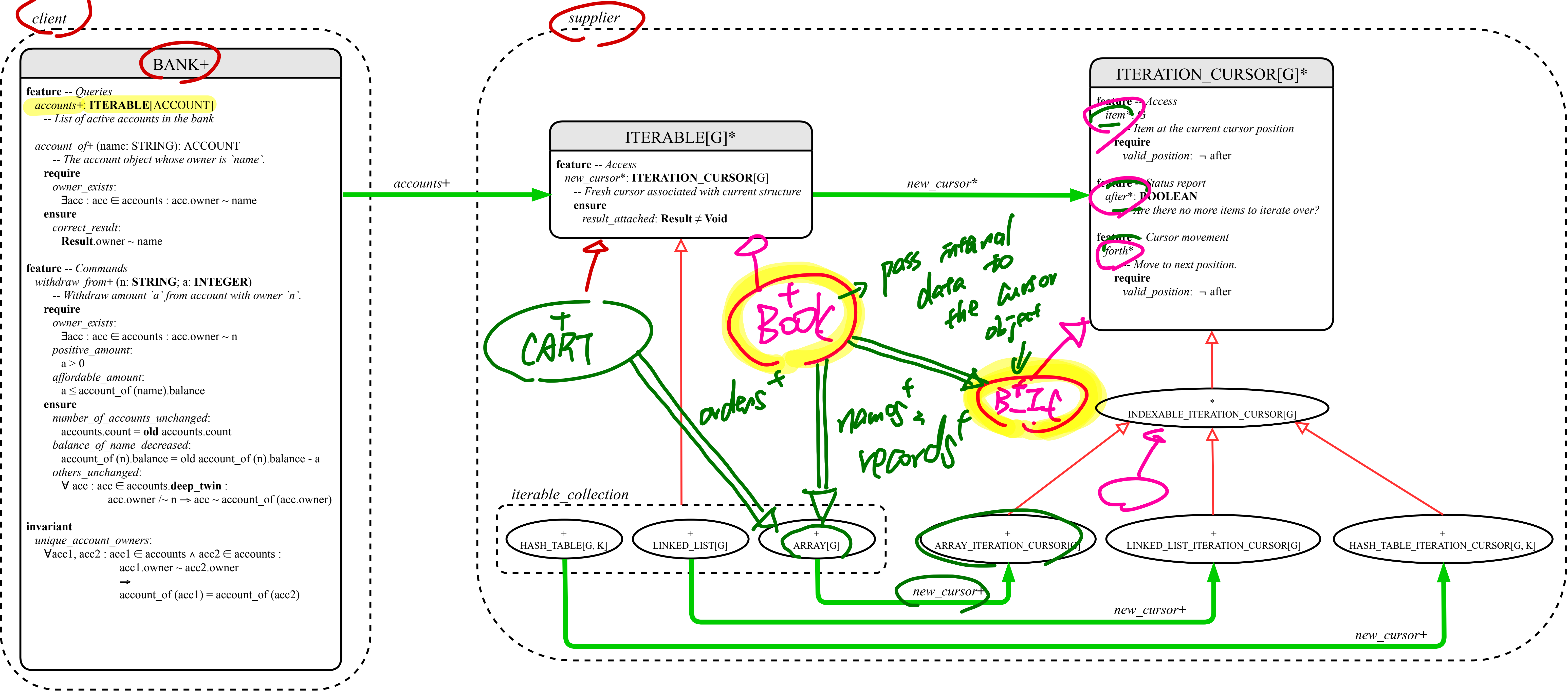
Implementing the Iterator Pattern: Easy Case

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

feature -- cursor

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



LECTURE 7

THURSDAY SEPTEMBER 26



client

BANK+

feature -- *Queries*
accounts+: **ITERABLE**[ACCOUNT]
-- *List of active accounts in the bank*

account_of+ (name: STRING): ACCOUNT
-- *The account object whose owner is `name`.*
require
owner_exists:
∃acc : acc ∈ accounts : acc.owner ~ name
ensure
correct_result:
Result.owner ~ name

feature -- *Commands*
withdraw_from+ (n: **STRING**; a: **INTEGER**)
-- *Withdraw amount `a` from account with owner `n`.*
require
owner_exists:
∃acc : acc ∈ accounts : acc.owner ~ n
positive_amount:
a > 0
affordable_amount:
a ≤ account_of (name).balance
ensure
number_of_accounts_unchanged:
accounts.count = **old** accounts.count
balance_of_name_decreased:
account_of (n).balance = old account_of (n).balance - a
others_unchanged:
∀ acc : acc ∈ accounts.**deep_twin** :
acc.owner /~ n ⇒ acc ~ account_of (acc.owner)

invariant
unique_account_owners:
∀acc1, acc2 : acc1 ∈ accounts ∧ acc2 ∈ accounts :
acc1.owner ~ acc2.owner
⇒
account_of (acc1) = account_of (acc2)

accounts+

supplier

ITERABLE[G]*

feature -- *Access*
*new_cursor**: **ITERATION_CURSOR**[G]
-- *Fresh cursor associated with current structure*
ensure
result_attached: **Result** ≠ **Void**

*new_cursor**

ITERATION_CURSOR[G]*

feature -- *Access*
*item**: G
-- *Item at the current cursor position*
require
valid_position: ¬ after

feature -- *Status report*
*after**: **BOOLEAN**
-- *Are there no more items to iterate over?*

feature -- *Cursor movement*
*forth**
-- *Move to next position.*
require
valid_position: ¬ after

*
INDEXABLE_ITERATION_CURSOR[G]

iterable_collection

+
HASH_TABLE[G, K]

+
LINKED_LIST[G]

+
ARRAY[G]

+
ARRAY_ITERATION_CURSOR[G]

+
LINKED_LIST_ITERATION_CURSOR[G]

+
HASH_TABLE_ITERATION_CURSOR[G, K]

new_cursor+

new_cursor+

new_cursor+

+
CART

orders+

+
Book

pass internal data to the cursor object

names+ & records+

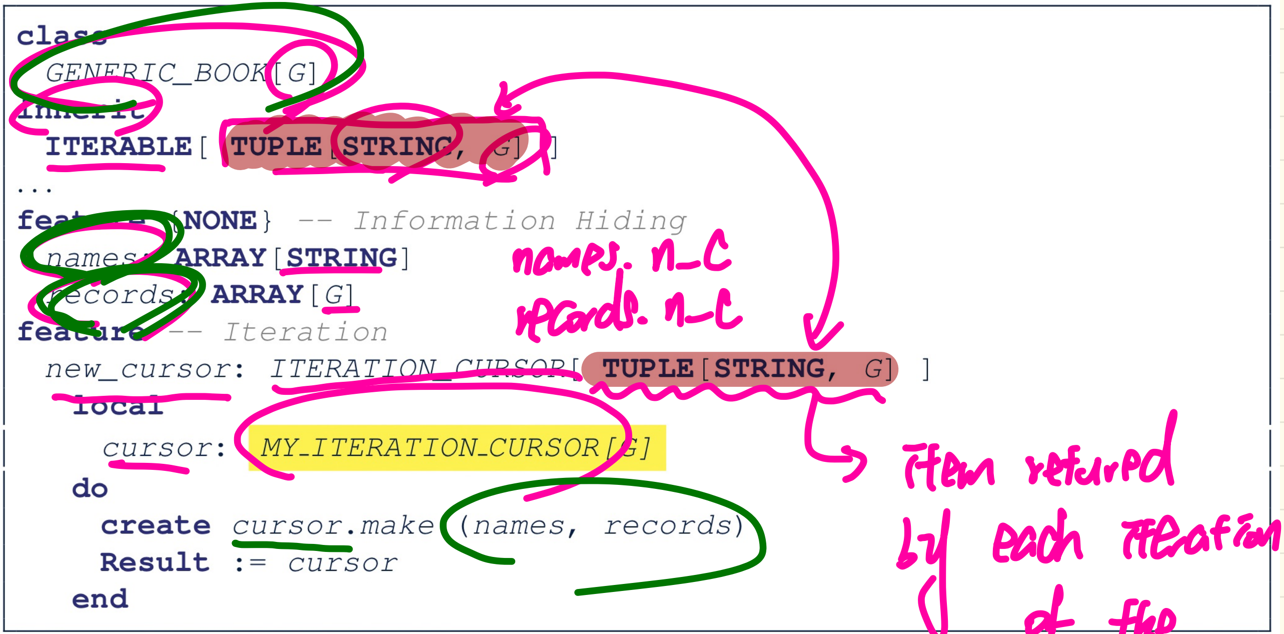
B_I_C
+

Implementing the Iterator Pattern: Easy Case

```
class
  CART
inherit
  ITERABLE[ORDER]

...

feature {NONE} -- Information Hiding
  orders: ARRAY[ORDER]

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

collection$^+$

Implementing the Iterator Pattern: Hard Case (1)



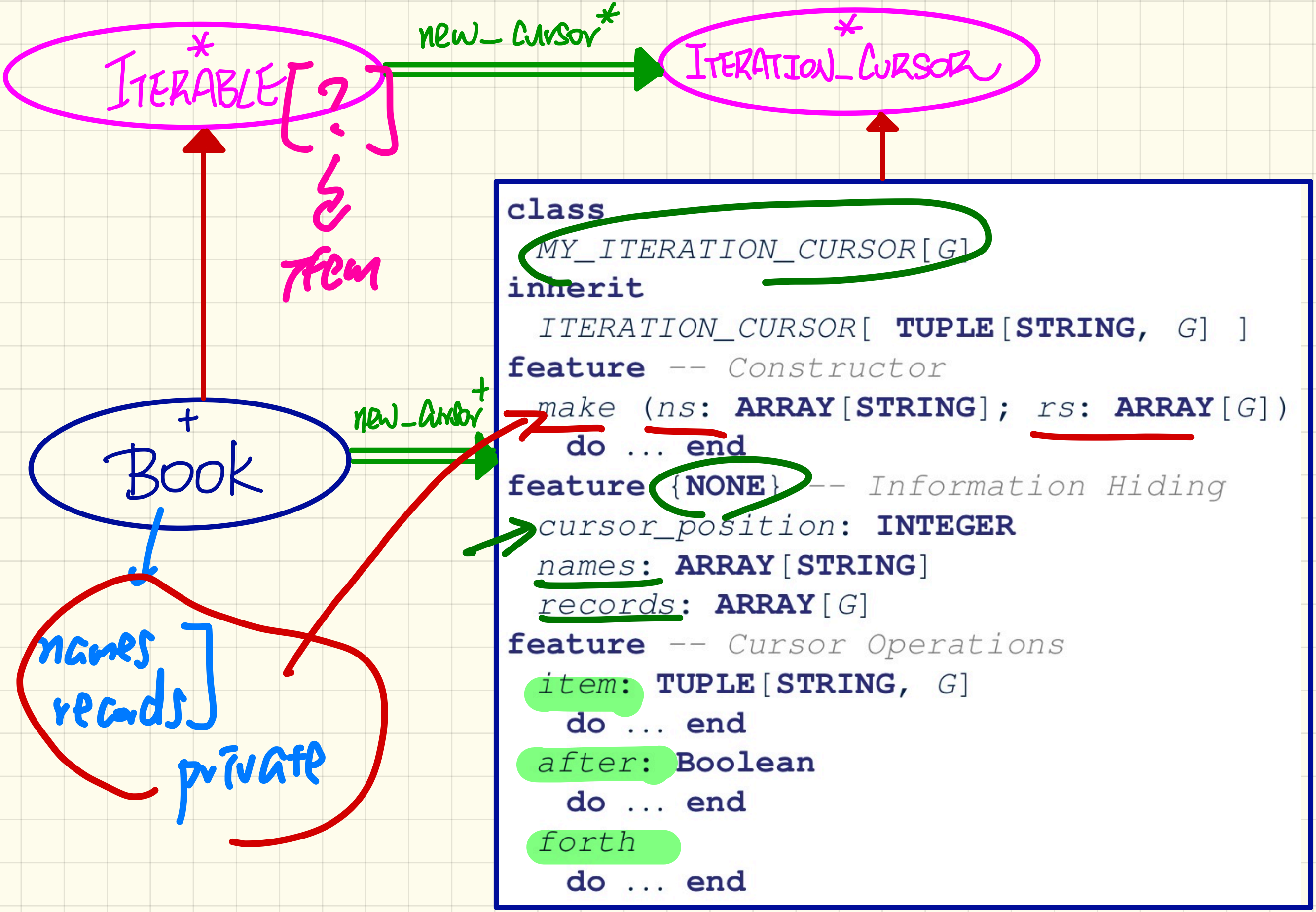
```
class
  GENERIC_BOOK[G]
inherit
  ITERABLE[ TUPLE[STRING, G] ]
...
feature {NONE} -- Information Hiding
  names: ARRAY[STRING]
  records: ARRAY[G]
feature -- Iteration
  new_cursor: ITERATION_CURSOR[ TUPLE[STRING, G] ]
    local
      cursor: MY_ITERATION_CURSOR[G]
    do
      create cursor.make (names, records)
      Result := cursor
    end
```

names. n_c

records. n_c

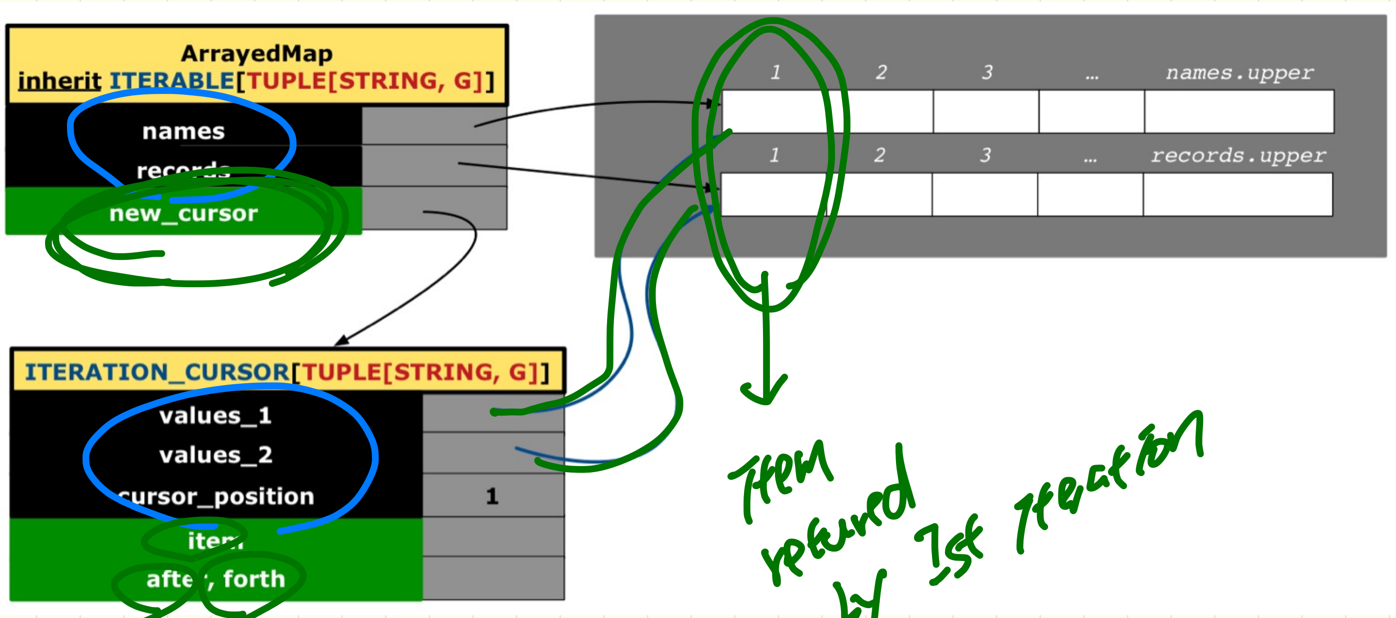
item returned by each iteration of the cursor

Implementing the Iterator Pattern: Hard Case (2)



```
class
  MY_ITERATION_CURSOR[G]
inherit
  ITERATION_CURSOR[ TUPLE[STRING, G] ]
feature -- Constructor
  make (ns: ARRAY[STRING]; rs: ARRAY[G])
    do ... end
feature {NONE} -- Information Hiding
  cursor_position: INTEGER
  names: ARRAY[STRING]
  records: ARRAY[G]
feature -- Cursor Operations
  item: TUPLE[STRING, G]
    do ... end
  after: Boolean
    do ... end
  forth
    do ... end
```

Iterator Pattern at Runtime



item returned by 1st iteration

TUPLE[STRING, G]

+ RECORD

Use of Iterable in Contracts

empty

10 |..| 1 is i

10 ≤ i ≤ 1

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

Iterable

collection.
new-cursor

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

```
across 1 |..| 5 is i
    io.put_int(i)
               6-i
end
```

1	5
2	4
3	3
4	2
5	1

Use of Iterable in Contracts: Exercise

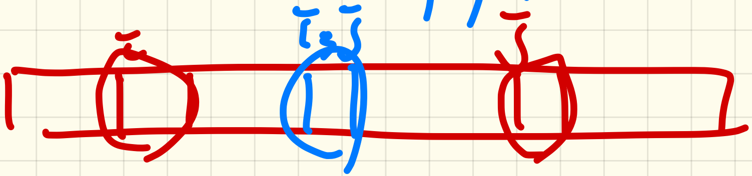
```
class BANK
...
  accounts: LIST [ACCOUNT]
  contains_duplicate: BOOLEAN
      -- Does the account list contain duplicate?
    do
      ...
    ensure
      ∀i, j : INTEGER |
        1 ≤ i ≤ accounts.count ∧ 1 ≤ j ≤ accounts.count •
        accounts[i] ~ accounts[j] ⇒ i = j
    end
```

across

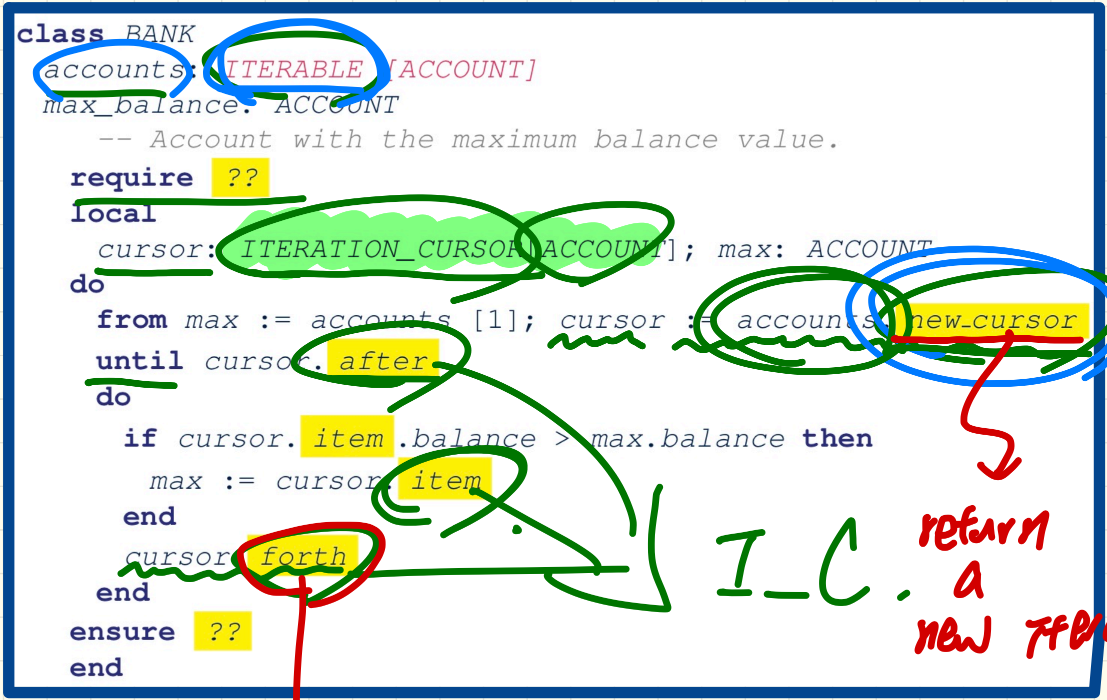
i ; j ✗

range

property



Use of Iterable in Implementation (1)

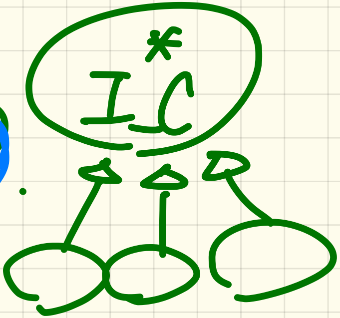


```
class BANK
  accounts: ITERABLE [ACCOUNT]
  max_balance: ACCOUNT
      -- Account with the maximum balance value.
    require ??
    local
      cursor: ITERATION_CURSOR[ACCOUNT]; max: ACCOUNT
    do
      from max := accounts [1]; cursor := accounts. new_cursor
      until cursor. after
      do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

I_C.

return a new iteration cursor for 'accounts' in the starting pos.

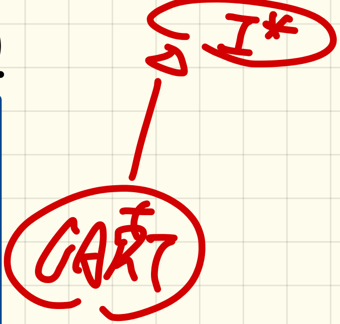
move cursor to next item



Use of Iterable in Implementation (2)

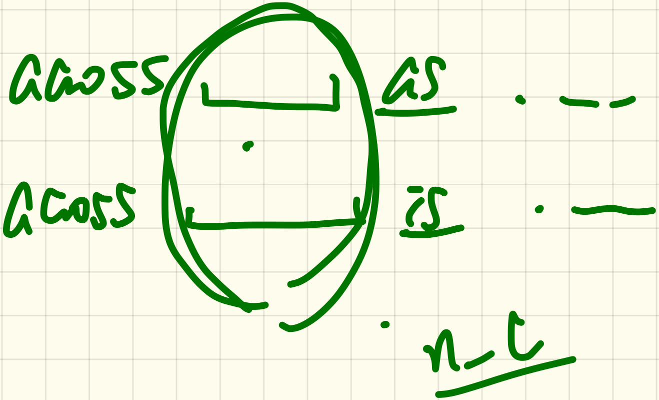
```
class SHOP
  cart: CART
  checkout: INTEGER
     -- Total price calculated based on orders in the cart.
    require ??
    do
      across
        cart is order
      loop
        Result := Result + order.price * order.quantity
      end
    ensure ??
  end
```

cart.new_cursor

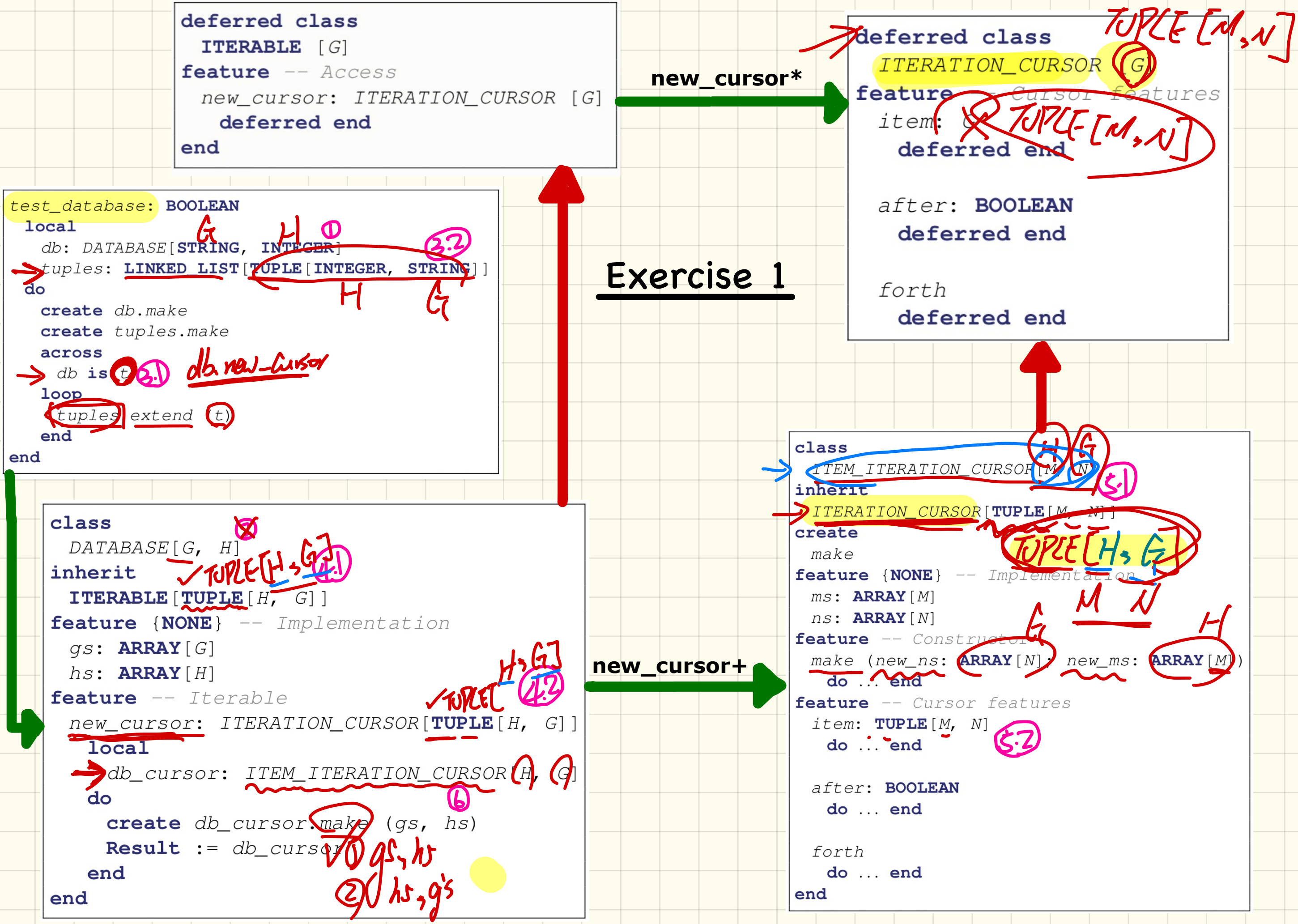


```
class BANK
  accounts: ITERABLE [ACCOUNT]
  max_balance: ACCOUNT
     -- Account with the maximum balance value.
    require ??
    local
      max: ACCOUNT
    do
      max := accounts [1]
      across
        accounts is acc
      loop
        if acc.balance > max.balance then
          max := acc
        end
      end
    ensure ??
  end
```

accounts.n_c



Exercise 1



```
deferred class
  ITERABLE [G]
feature -- Access
  new_cursor: ITERATION_CURSOR [G]
    deferred end
end
```

```
deferred class
  ITERATION_CURSOR [G]
feature -- Cursor features
  item: G
    deferred end

  after: BOOLEAN
    deferred end

  forth
    deferred end
```

TUPLE[M, N]

TUPLE[M, N]

```
test_database: BOOLEAN
  local
    db: DATABASE[STRING, INTEGER]
    tuples: LINKED_LIST[TUPLE[INTEGER, STRING]]
  do
    create db.make
    create tuples.make
    across
      db is t
    loop
      tuples.extend (t)
    end
end
```

G H ①

3.2

H G

3.1 db.new_cursor

```
class
  DATABASE[G, H]
inherit
  ITERABLE[TUPLE[H, G]]
feature {NONE} -- Implementation
  gs: ARRAY[G]
  hs: ARRAY[H]
feature -- Iterable
  new_cursor: ITERATION_CURSOR[TUPLE[H, G]]
    local
      db_cursor: ITEM_ITERATION_CURSOR[H, G]
    do
      create db_cursor.make (gs, hs)
      Result := db_cursor
    end
end
```

✓TUPLE[H, G] 4.1

✓TUPLE[H, G] 4.2

6

① gs, hs

② hs, g's

```
class
  ITEM_ITERATION_CURSOR[M, N]
inherit
  ITERATION_CURSOR[TUPLE[M, N]]
create
  make
feature {NONE} -- Implementation
  ms: ARRAY[M]
  ns: ARRAY[N]
feature -- Constructor
  make (new_ns: ARRAY[N]; new_ms: ARRAY[M])
    do ... end
feature -- Cursor features
  item: TUPLE[M, N]
    do ... end

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```

H G

5.1

TUPLE[H, G]

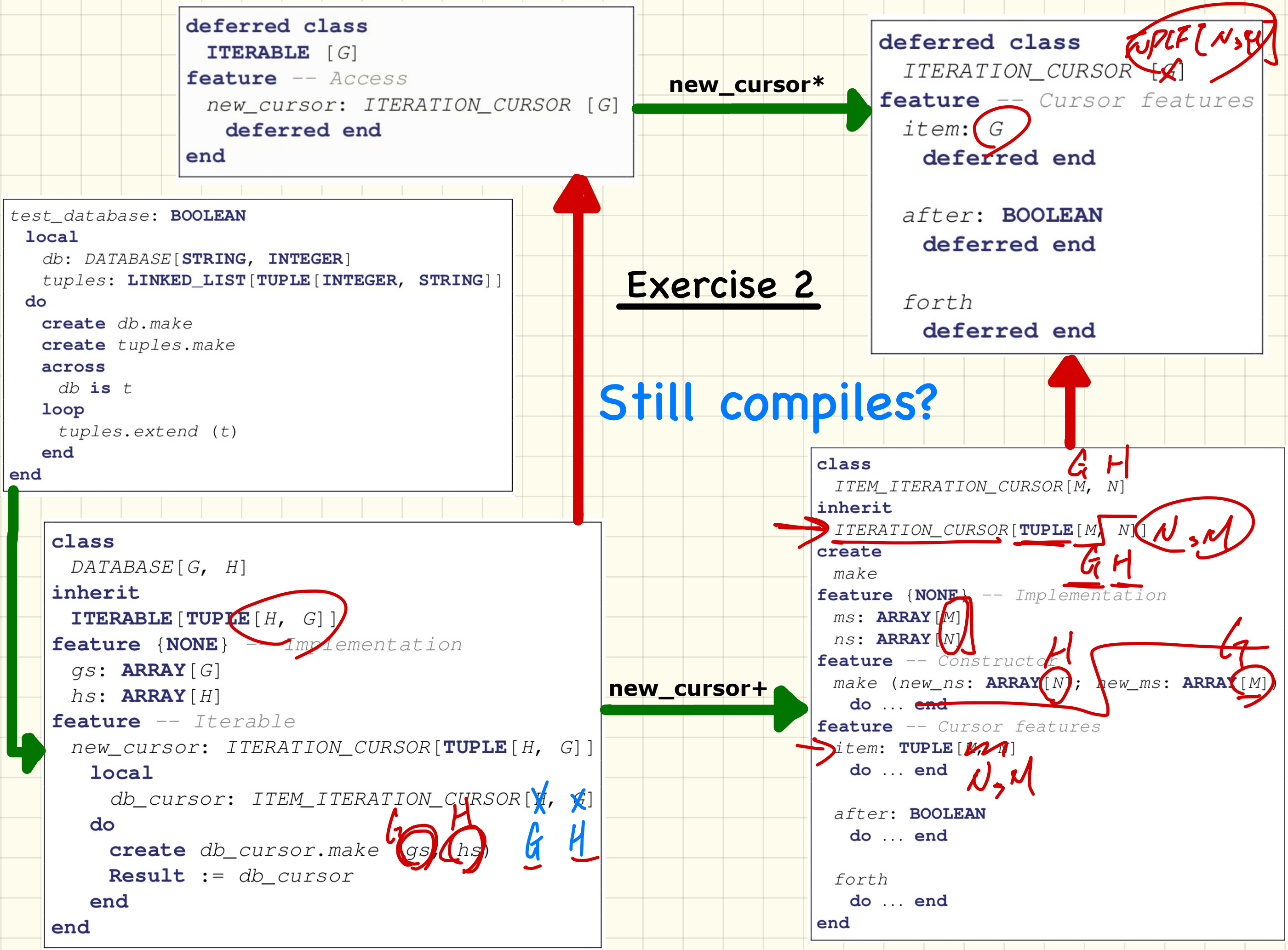
M N

G H

5.2

Exercise 2

Still compiles?



```
deferred class
  ITERABLE [G]
feature -- Access
  new_cursor: ITERATION_CURSOR [G]
    deferred end
end
```

```
deferred class
  ITERATION_CURSOR [~~G~~]
feature -- Cursor features
  item: G
    deferred end

  after: BOOLEAN
    deferred end

  forth
    deferred end
end
```

TUPLE[N, M]

```
test_database: BOOLEAN
  local
    db: DATABASE[STRING, INTEGER]
    tuples: LINKED_LIST[TUPLE[INTEGER, STRING]]
  do
    create db.make
    create tuples.make
    across
      db is t
    loop
      tuples.extend (t)
    end
end
```

```
class
  DATABASE[G, H]
inherit
  ITERABLE[TUPLE[H, G]]
feature {NONE} -- Implementation
  gs: ARRAY[G]
  hs: ARRAY[H]
feature -- Iterable
  new_cursor: ITERATION_CURSOR[TUPLE[H, G]]
    local
      db_cursor: ITEM_ITERATION_CURSOR[~~H~~, ~~G~~]
    do
      create db_cursor.make (gs, hs)
      Result := db_cursor
    end
end
```

G H

G H

```
class
  ITEM_ITERATION_CURSOR[M, N]
inherit
  ITERATION_CURSOR[TUPLE[M, N]]
create
  make
feature {NONE} -- Implementation
  ms: ARRAY[M]
  ns: ARRAY[N]
feature -- Constructor
  make (new_ns: ARRAY[N]; new_ms: ARRAY[M])
    do ... end
feature -- Cursor features
  item: TUPLE[~~M, N~~]
    do ... end

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```

G H

N, M

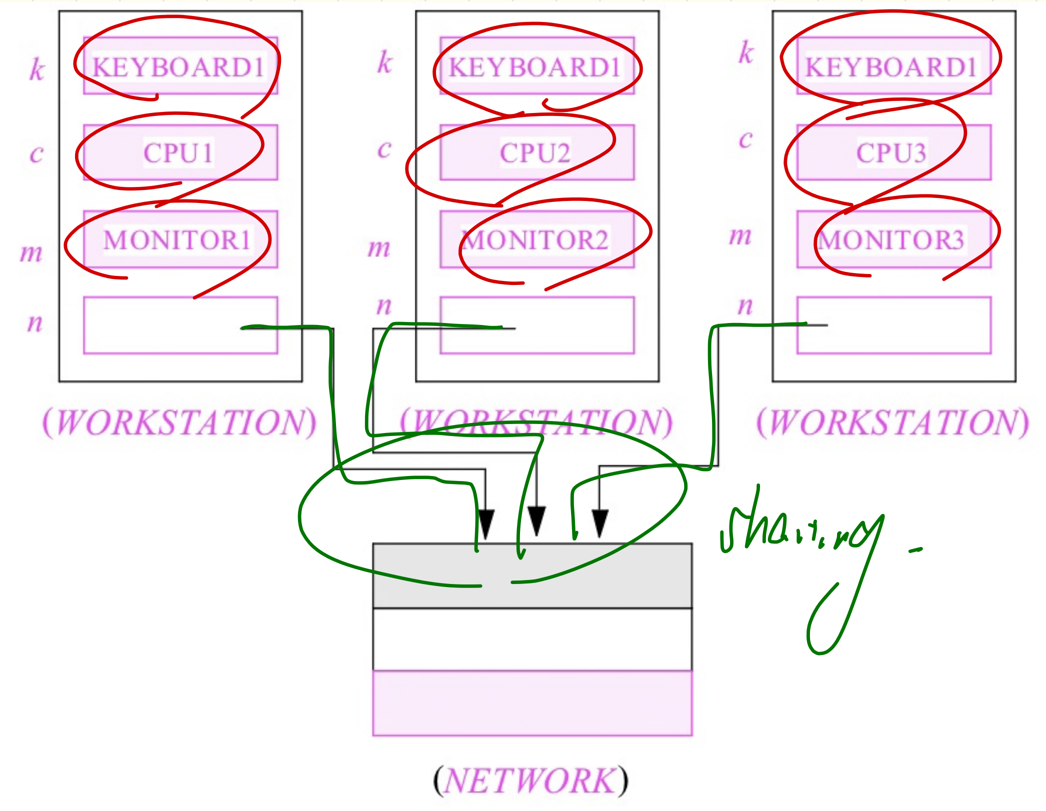
G H

H

G

N, M

Modelling: Aggregation vs. Composition



sharing.

Use of **Expanded** Type

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

```
test_expanded: BOOLEAN
  local
    eb1, eb2: B
  do
    Result := eb1.i = 0 and eb2.i = 0
    check Result end
    Result := eb1 = eb2
    check Result end
    eb2.change_i (15)
    Result := eb1.i = 0 and eb2.i = 15
    check Result end
    Result := eb1 /= eb2
    check Result end
  end
```

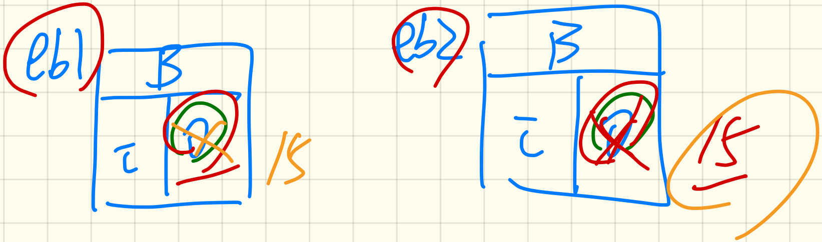
for expanded type, compare contents.

0 15

copy

eb1 := eb2

Q: ARRAY[INT]

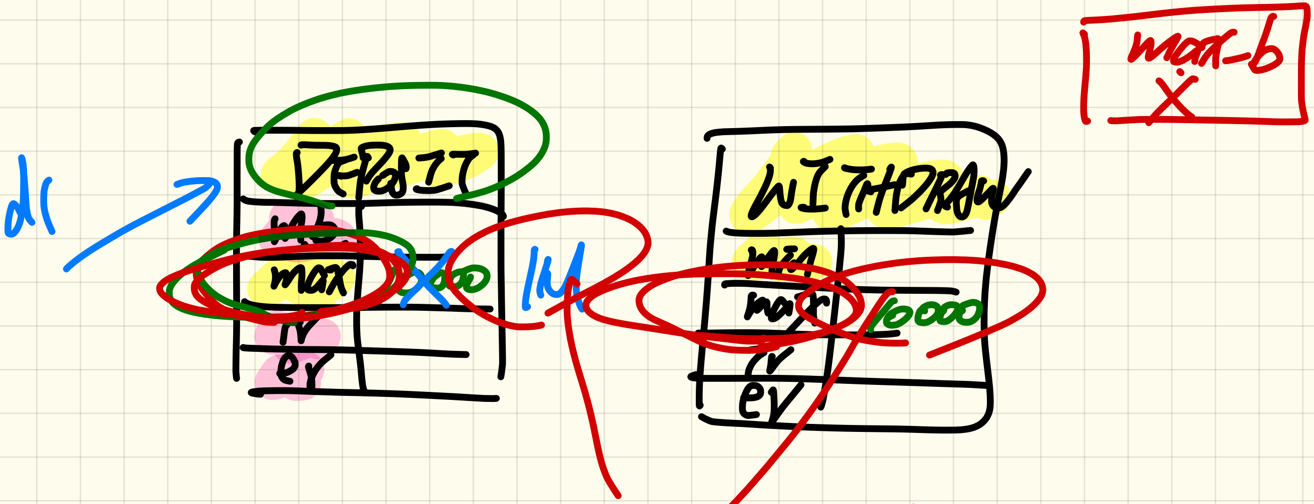


LECTURE 8

TUESDAY OCTOBER 7

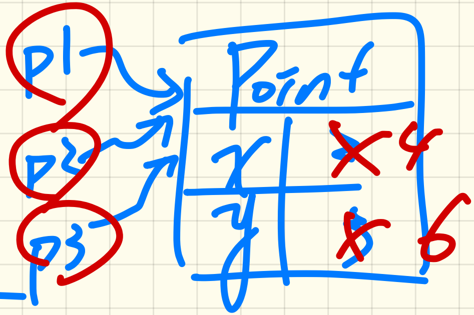
Runtime

d1.set_max_balance(1M)



Inconsistency.

```
Point p1 = new Point(2,3);
Point p2 = p1;
Point p3 = p2;
```

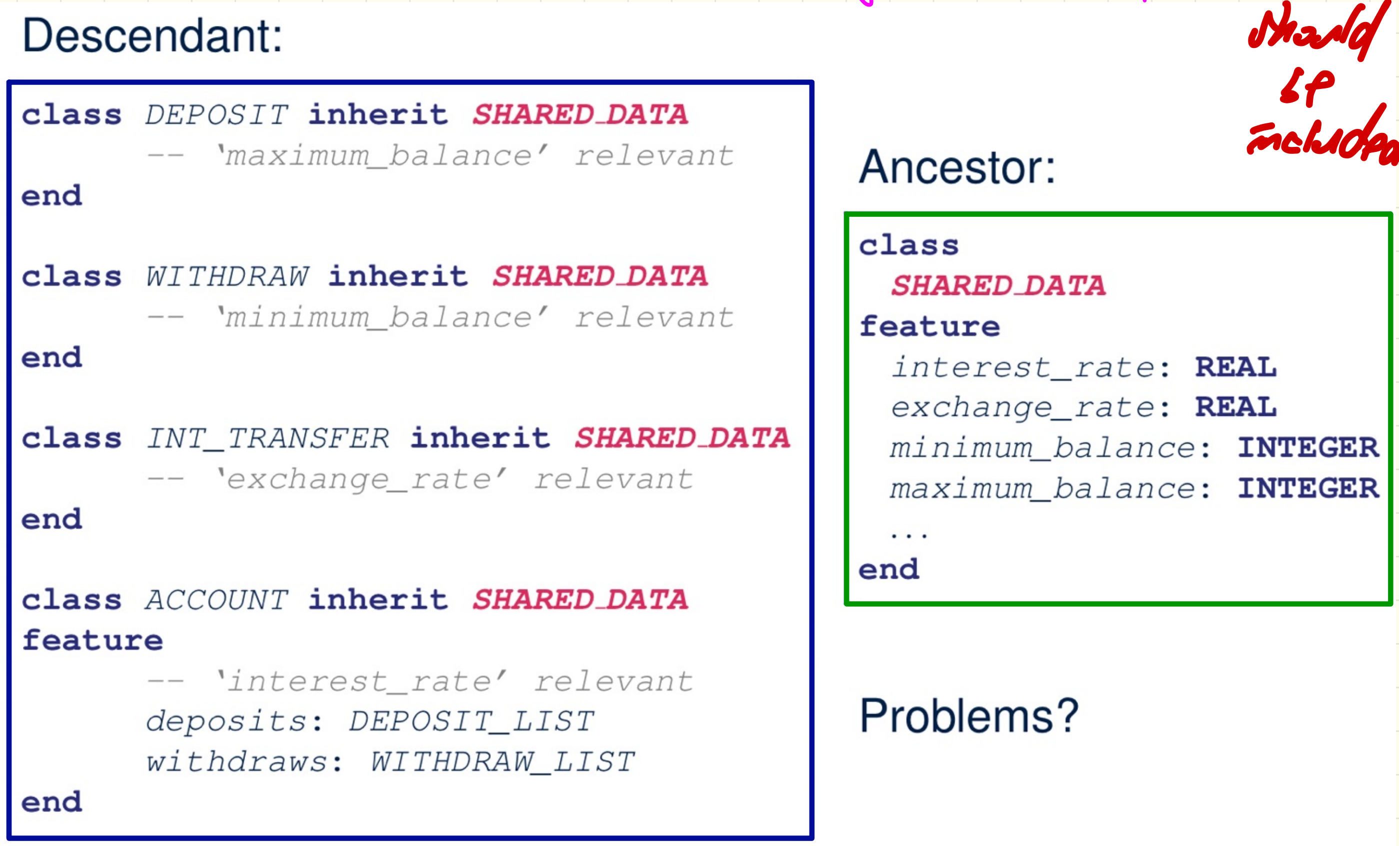


```
→ p2.setPoint(4,6);
    p1.getX()
    p1.getY()
```

Shared Data via Inheritance

Cohesion → e.g. DEPOSIT only max_b should be included

Single Choice Principle



Descendant:

```
class DEPOSIT inherit SHARED_DATA
      -- 'maximum_balance' relevant
end

class WITHDRAW inherit SHARED_DATA
      -- 'minimum_balance' relevant
end

class INT_TRANSFER inherit SHARED_DATA
      -- 'exchange_rate' relevant
end

class ACCOUNT inherit SHARED_DATA
feature
      -- 'interest_rate' relevant
      deposits: DEPOSIT_LIST
      withdraws: WITHDRAW_LIST
end
```

Ancestor:

```
class
  SHARED_DATA
feature
  interest_rate: REAL
  exchange_rate: REAL
  minimum_balance: INTEGER
  maximum_balance: INTEGER
  ...
end
```

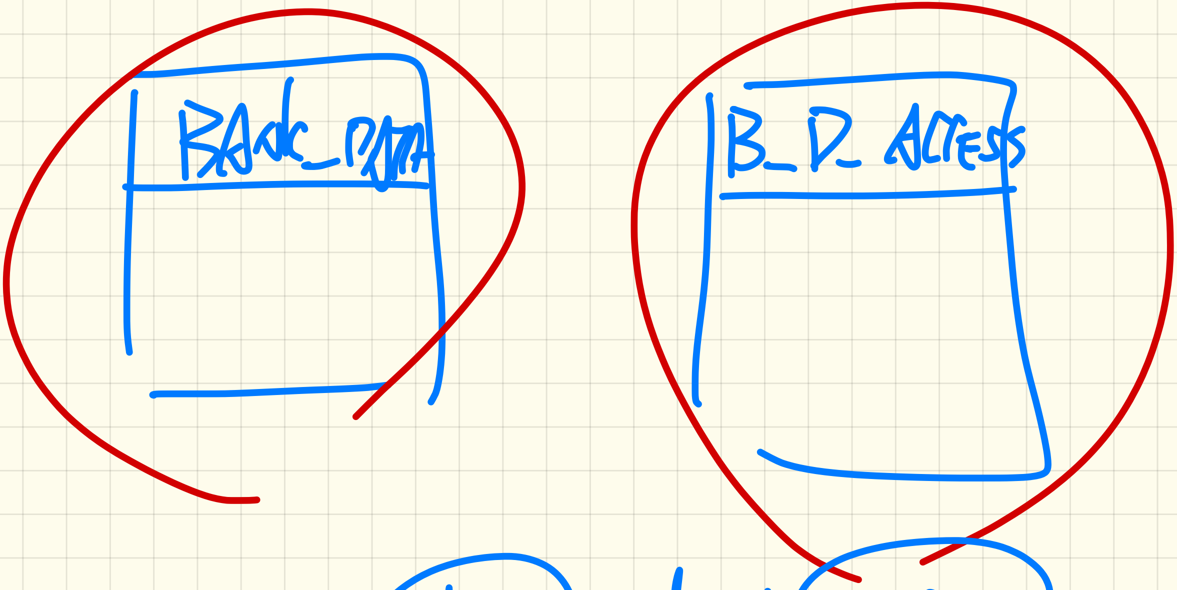
Problems?

SCP

- When a kind of information is duplicated in multiple places.

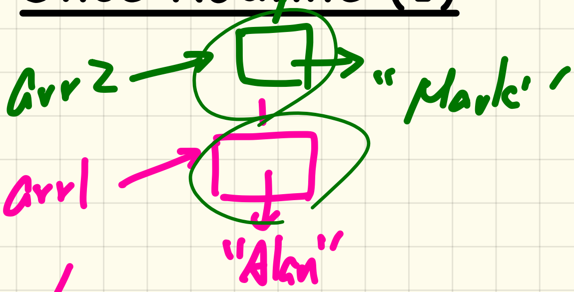
↳ when there's an update on info, you have to update multiple places.

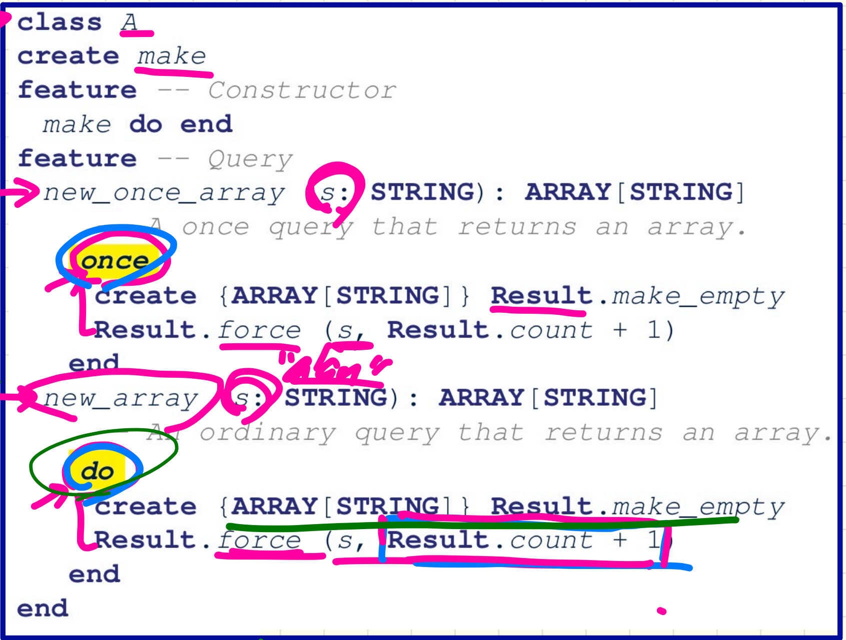
hard to maintain



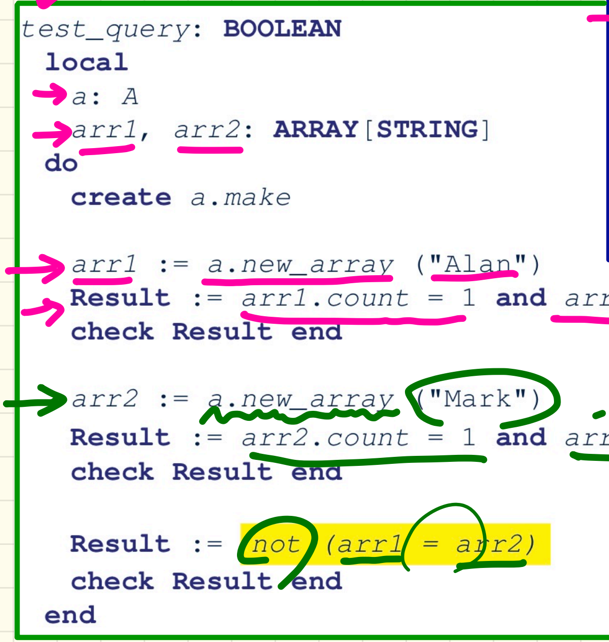
Cohesion: data and their access belong to different classes

Once Routine (1)

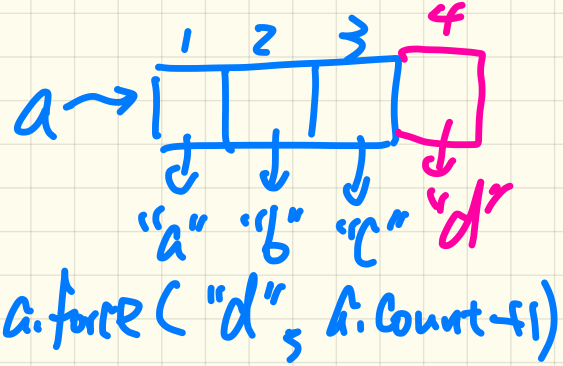




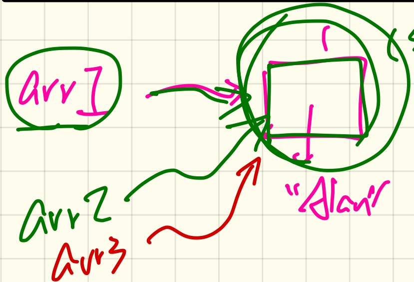
```
class A
create make
feature -- Constructor
  make do end
feature -- Query
  new_once_array (s: STRING): ARRAY[STRING]
      -- A once query that returns an array.
    once
      create {ARRAY[STRING]} Result.make_empty
      Result.force (s, Result.count + 1)
    end
  new_array (s: STRING): ARRAY[STRING]
      -- An ordinary query that returns an array.
    do
      create {ARRAY[STRING]} Result.make_empty
      Result.force (s, Result.count + 1)
    end
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
```



Once Routine (2)



(system will cache the address of this return value)

```
class A
create make
feature -- Constructor
  make do end
feature -- 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
  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
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
```

arr3: ARRAY[STRING]

1st time of calling this once query

ignored: it's past the 1st time

arr3 := a.new_once_array ("Tony")

R1. Shared Data (single instance)

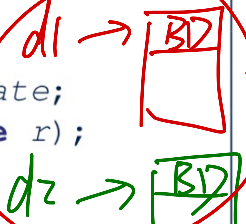
R2. Cohesion: separate between data and shared access to data

Approximating Once Routines in Java (1)

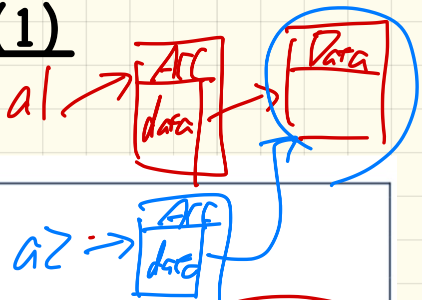
BankData d1 = BD(..);
BankData d2 = BD(..);

```
class BankData {
  BankData() { }
  double interestRate;
  void setIR(double r);
  ...
}
```

R1 ✗



```
class Account {
  BankData data;
  Account() {
    data = BankDataAccess.getData();
  }
}
```



```
class BankDataAccess {
  static boolean initOnce;
  static BankData data;
  static BankData getData() {
    if(!initOnce) {
      data = new BankData();
      initOnce = true;
    }
    return data;
  }
}
```

data to be shared

Problem?

Account a1 = new A();
Account a2 = new A();

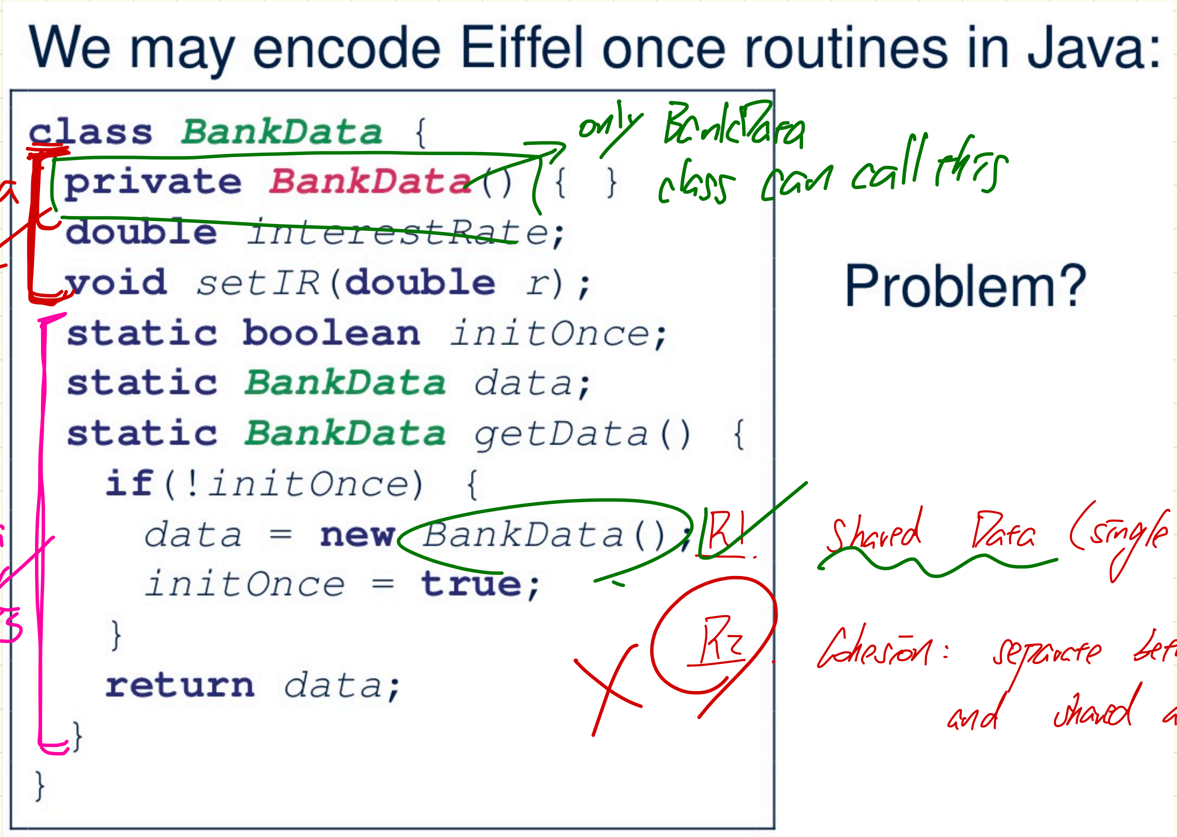
R1 Shared Data (single instance)

R2 . Cohesion: separate between data and shared access to data

for 1st call, new instance

for subsequent calls, return just existing data.

Approximating Once Routines in Java (2)



We may encode Eiffel once routines in Java:

```
class BankData {
  private BankData() { }
  double interestRate;
  void setIR(double r);
  static boolean initOnce;
  static BankData data;
  static BankData getData() {
    if(!initOnce) {
      data = new BankData();
      initOnce = true;
    }
    return data;
  }
}
```

Problem?

only BankData class can call this

data

data access

R1

R2

Shared Data (single instance)

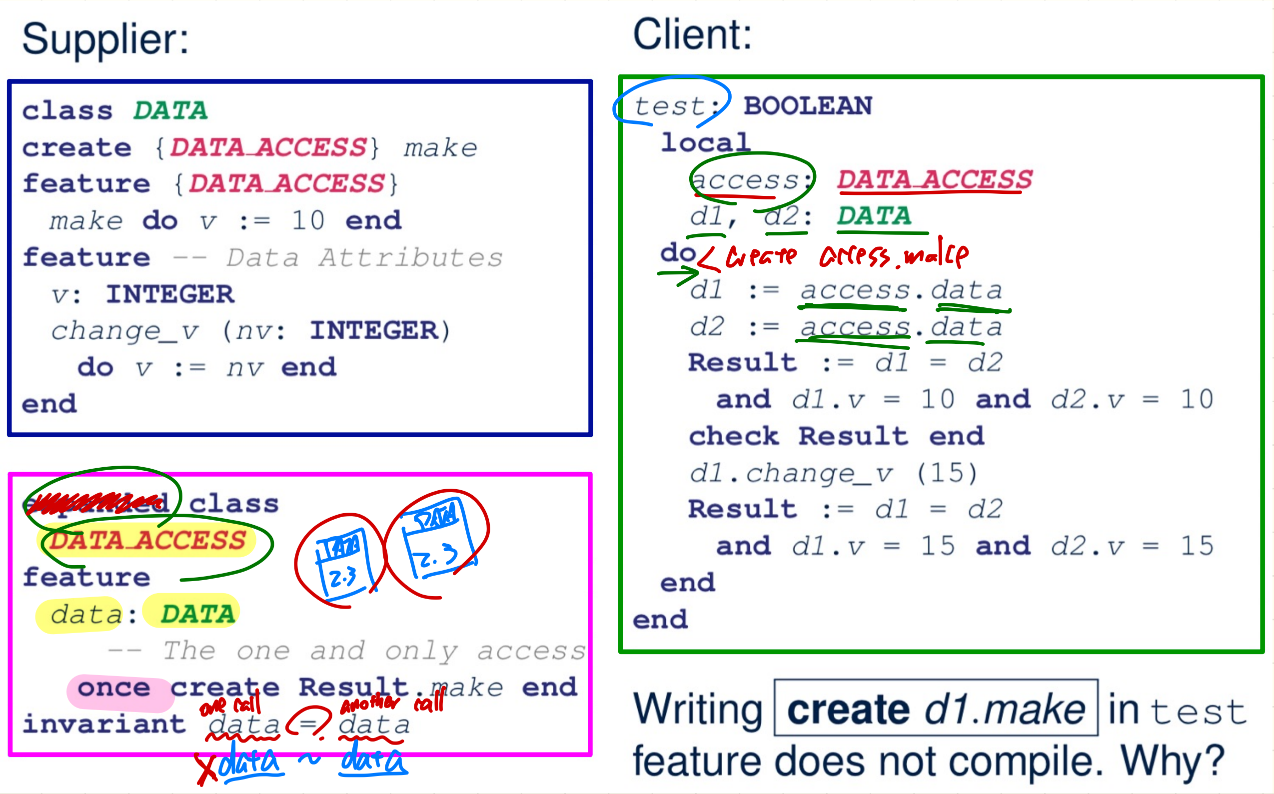
Cohesion: separate between data and shared access to data feature {NONE} private

feature public

class DATA
feature {DATA_ACCESS}
make (. . .) do . . end
feature
data : BANK_DATA
[

class DATA_ACCESS
[

Singleton Design Pattern: Code (1)



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
                 one call    another call
invariant data =? data
               Xdata ~ data
```

Client:

```
test: BOOLEAN
  local
    access: DATA_ACCESS
    d1, d2: DATA
  do < create access.make
    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
```

Writing **create *d1.make*** in `test` feature does not compile. Why?

violates sharing ∵ any client can create a new data.

Supplier:

```
class DATA
create [redacted] make
feature {[redacted]}
  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
```

Client:

```
test: BOOLEAN
  local
    access: DATA_ACCESS
    d1, d2: DATA
  do        create d1.make
    d1 := access.data        create d2.make
    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
```

Writing **create** *d1.make* in `test` feature does not compile. Why?

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
```

~~once~~ do

violates sharing

∴ different calls to 'data' gives for different objects

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
```

violating inv.

access.data = access.data

Writing **create *d1.make*** in `test` feature does not compile. Why?

Supplier:

Still singleto

```
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
```

Client:

```
test: BOOLEAN
  local
    access: DATA_ACCESS
    d1, d2: DATA
  do create access.make
    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
```

Writing **create *d1.make*** in `test` feature does not compile. Why?

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
```

~

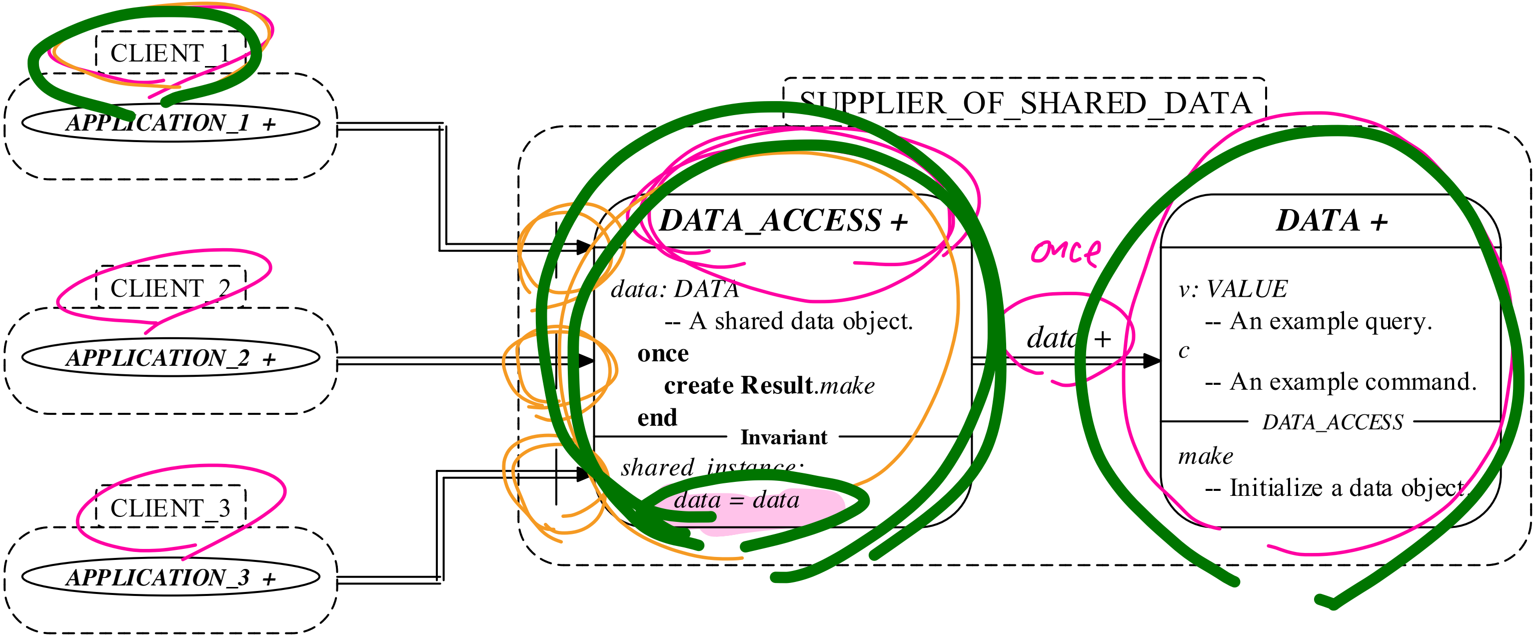
Client:

TEST

```
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
```

Writing **create *d1.make*** in `test` feature does not compile. Why?

Singleton Design Pattern



DATA_ACCESS +

```
data: DATA
    -- A shared data object.
  once
    create Result.make
  end
```

Invariant

shared_instance:
data = data

DATA +

v: VALUE
-- An example query.
c
-- An example command.

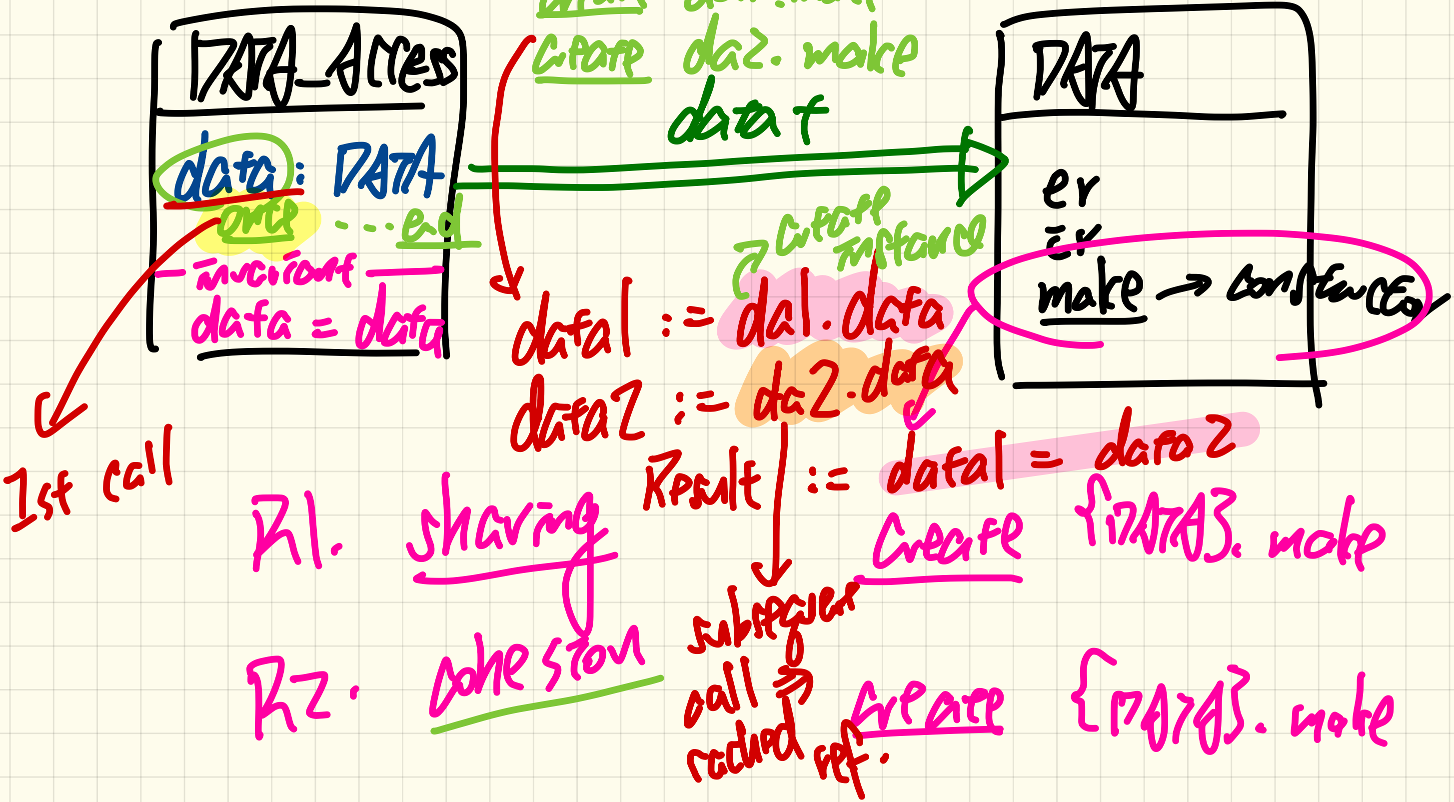
DATA_ACCESS

make
-- Initialize a data object.

LECTURE 9

THURSDAY OCTOBER 3

```
local data1, data2 : DATA
      da1, da2 : DATA_ACCESS
do
  create da1.make
  create da2.make
  data1 := da1.data
  data2 := da2.data
  Result := data1 = data2
```



R1. sharing create {DATA}.make

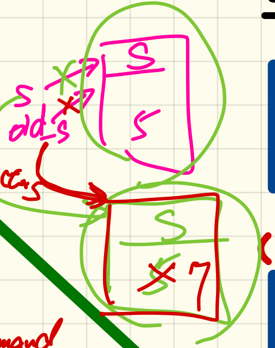
R2. cohesion create {DATA}.make

Export Status Case 1

```
class CLIENT_1
  ...
  test: BOOLEAN
    local
      s, old_s: SUPPLIER
    do
      create s.make (5)
      old_s := s
      create s.make (5)
      print (old_s = s)
      old_s := s
      s.make (7)
      print (old_s = s)
    end
end
```

calls to constructors

call to command



```
class CLIENT_2
  ...
  test: BOOLEAN
    local
      s, old_s: SUPPLIER
    do
      create s.make (5)
      old_s := s
      create s.make (5)
      print (old_s = s)
      old_s := s
      s.make (7)
      print (old_s = s)
    end
end
```

```
class SUPPLIER

create {ANY}
      make
              {ANY}
feature <- - Command
    make (init_i: INTEGER)
      do
         i := init_i
      end

feature
    i: INTEGER
end
```

Export Status Case 2

```
class CLIENT_1
  ...
  test: BOOLEAN
    local
      s, old_s: SUPPLIER
    do
      create s.make (5)
      old_s := s
      create s.make (5)
      print (old_s = s)
      old_s := s
      s.make (7)
      print (old_s = s)
    end
end
```

```
class CLIENT_2
  ...
  test: BOOLEAN
    local
      s, old_s: SUPPLIER
    do
     ✗create s.make (5)
      old_s := s
     ✗create s.make (5)
      print (old_s = s)
      old_s := s
      s.make (7)
      print (old_s = s)
    end
end
```

```
class SUPPLIER

create {CLIENT_1}
      make

feature
   make (init_i: INTEGER)
      do
         i := init_i
      end

feature
   i: INTEGER
end
```

only C_1 can call make as a constructor.

Export Status Case 3

```
class CLIENT_1
  ...
  test: BOOLEAN
    local
      s, old_s: SUPPLIER
    do
      create s.make (5)
      old_s := s
      create s.make (5)
      print (old_s = s)
      old_s := s
      s.make (7)
      print (old_s = s)
    end
end
```

```
class CLIENT_2
  ...
  test: BOOLEAN
    local
      s, old_s: SUPPLIER
    do
     ✓create s.make (5)
      old_s := s
      create s.make (5)
      print (old_s = s)
      old_s := s
     ✗s.make (7)
      print (old_s = s)
    end
end
```

```
class SUPPLIER

create
    make

feature ~~{NONE}~~ {CLIENT_1}
  make (init_i: INTEGER)
    do
      i := init_i
    end

feature
  i: INTEGER
end
```

Export Status Case 4

```
class CLIENT_1
  ...
  test: BOOLEAN
    local
      s, old_s: SUPPLIER
    do
      create s.make (5)
      old_s := s
      create s.make (5)
      print (old_s = s)
      old_s := s
      s.make (7)
      print (old_s = s)
    end
end
```

```
class CLIENT_2
  ...
  test: BOOLEAN
    local
      s, old_s: SUPPLIER
    do
     ✗create s.make (5)
      old_s := s
     ✗create s.make (5)
      print (old_s = s)
      old_s := s
     ✗s.make (7)
      print (old_s = s)
    end
end
```

```
class SUPPLIER

create {CLIENT_1}
    make

feature {CLIENT_1}
  make (init_i: INTEGER)
    do
      i := init_i
    end
feature
  ———>        set_i(.. )
feature
  i: INTEGER
end
```

Writing Postcondition: Exercise 4

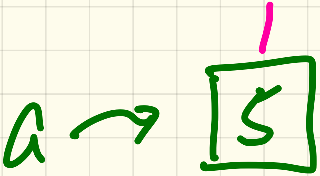
```
is_all_positive (a: ARRAY[INTEGER]): BOOLEAN
    ensure
        across a is i
        all
            a[i] > 0
        end
```

5

element

index

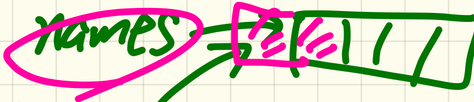
a[5] X



Writing Postcondition: Exercise 5

```
names: ARRAY[STRING]
name_at (i: INTEGER)
  require
    names.valid_index(i)
  ensure
    names.count = (old names).count
    across 1 |..| names.count is j
      all
        names[j] ~ (old names)[j]
      end
```

① ②



o-t

✓ (old names.twin).count

✓ (old names.d-t).count

✓ old names.count

(old names.deS)[i]

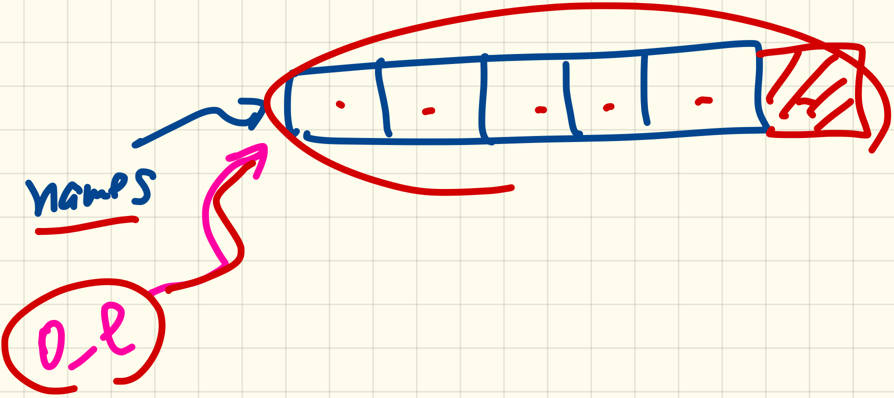
(old names.twin).count or old names.count

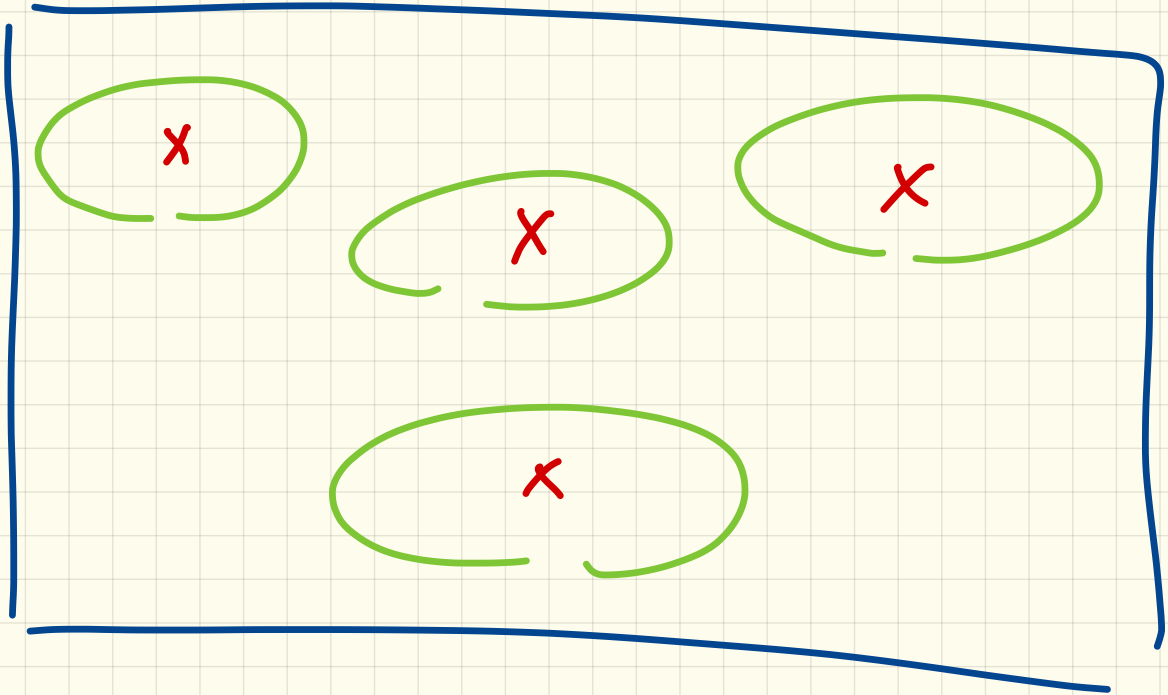
(old names.deep_twin)[i]

→ o_e := names

ensure

names.count = (old names).count




x
x
x
x

Violation of the 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
```

if courses.count < 5 then

```
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
```

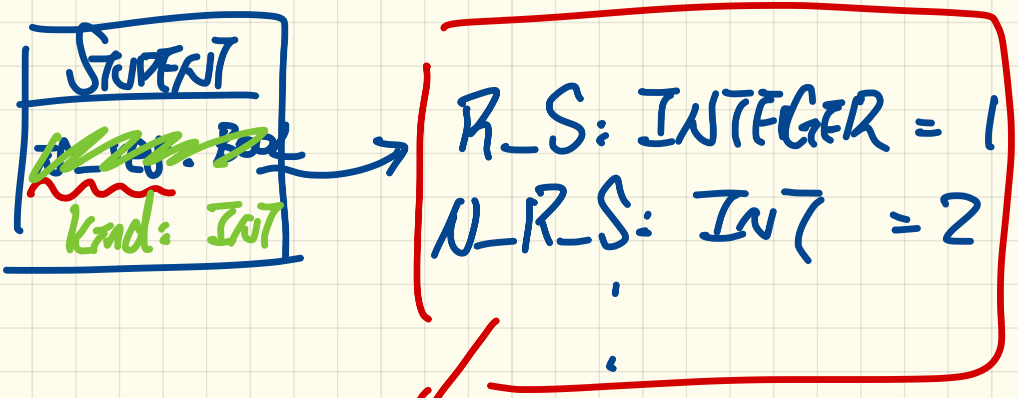
if courses. count < 5 then

N-R-S

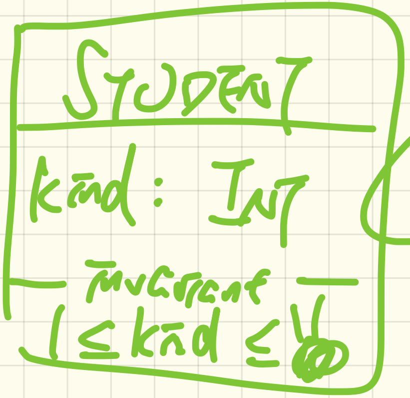
R-S

I-S

- -



encode the kind of student object manually



SCP -

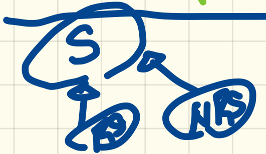
```
charge (s: STUDENT)
do
  if s.kind = 1 then
    ~
  else if s.kind = 2 then
    ~
  :
  end
end
```

else if s.kind = 11
~

```
get_tuition (s: STUDENT) : INT
do
  if s.kind = 1 then
    ~
  else if s.kind = 2 then
    ~
  :
  end
end
```

else if s.kind = 11
~

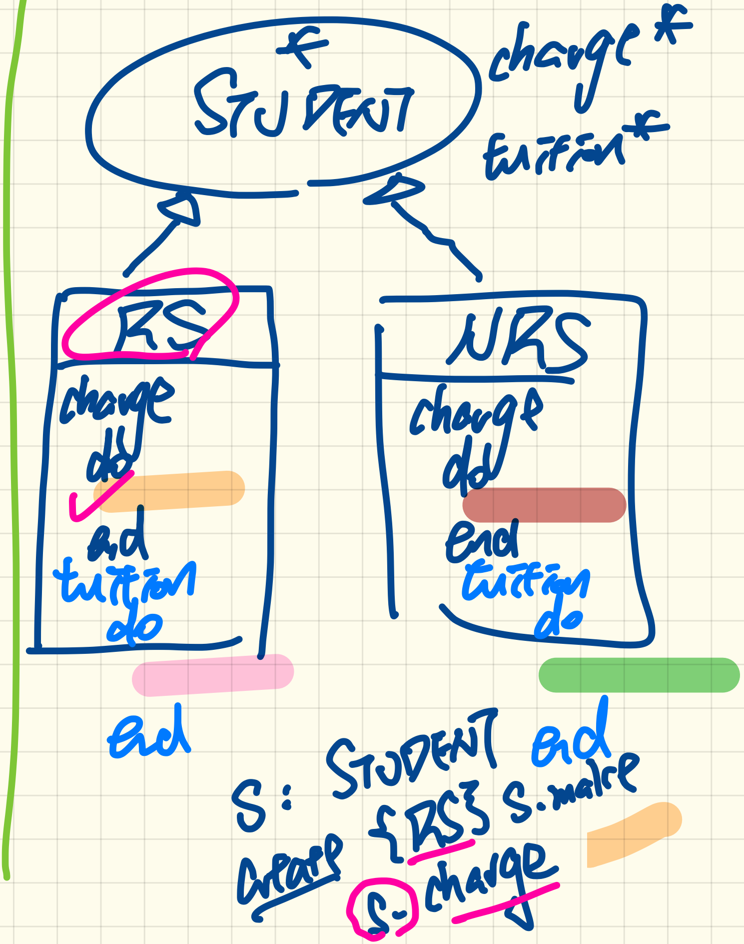
Q. What if adding a 11th kind student.



```
s: STUDENT
;
s.charge()
```

Without inheritance

```
STUDENT
kind: INT
charge (..)
  do
    if kind = 1 then          ← RS
    elseif kind = 2 then      ← NRS
    end
tuition (...)
    if kind = 1 then
    elseif kind = 2 then
    end
```



```
s: STUDENT
create {RS} s.make
s.charge
```

LECTURE 10

TUESDAY OCTOBER 8

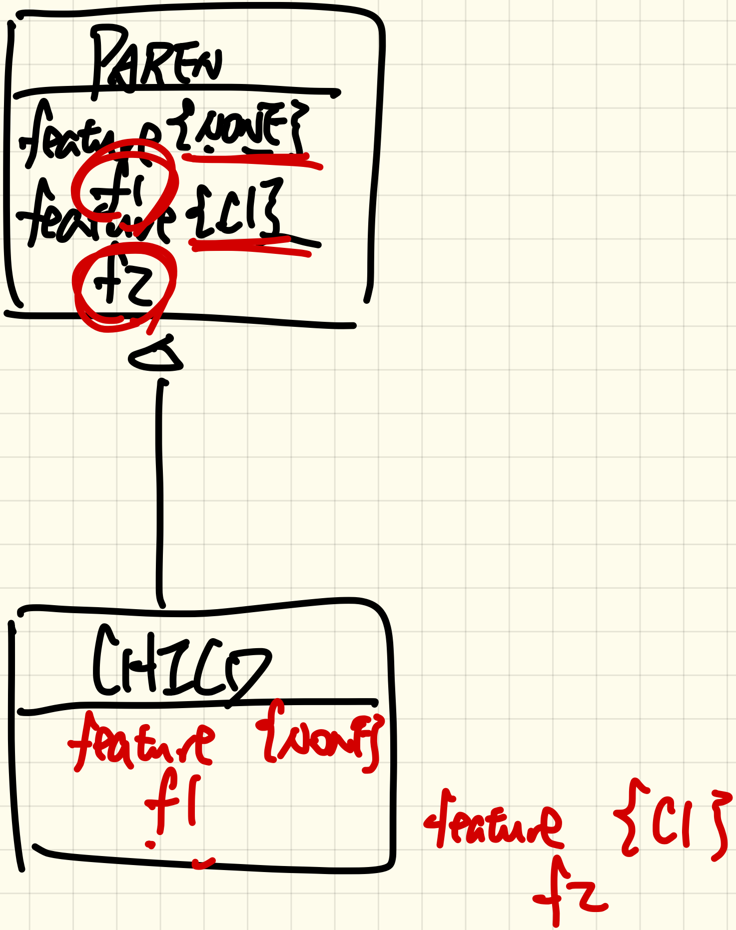
Violation of the Single Choice Principle

```
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
```

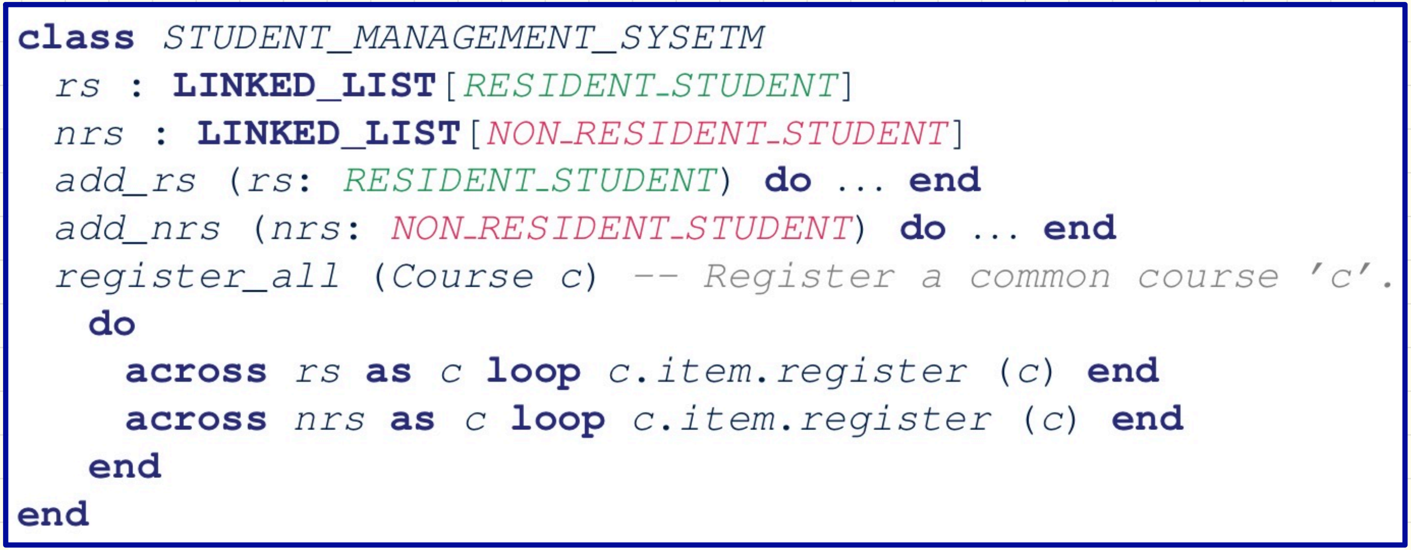
```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate:  REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r:  REAL) do premium_rate := r end
  register (c: COURSE) 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
```

```
cmd (...)
  do
    Precursor (...)
  end
```

```
f (...): INT
  do
    Result :=
      Precursor (...)
  end
```


PAREN
feature {NONE}
f1
feature {C1}
f2
CHILD
feature {NONE}
f1
feature {C1}
f2

Without Inheritance: Collection of Students



```
class STUDENT_MANAGEMENT_SYSETM
  rs : LINKED_LIST[RESIDENT_STUDENT]
  nrs : LINKED_LIST[NON_RESIDENT_STUDENT]
  add_rs (rs: RESIDENT_STUDENT) do ... end
  add_nrs (nrs: NON_RESIDENT_STUDENT) do ... end
  register_all (Course c) -- Register a common course 'c'.
    do
      across rs as c loop c.item.register (c) end
      across nrs as c loop c.item.register (c) end
    end
end
```

Q: What if **more** kinds of students are to be introduced?

Inheritance:

Code Reuse

```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  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
    end
end
```

```
class
  RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  premium_rate: REAL
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ;  Result := base * premium_rate end
end
```

```
class
  NON_RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  discount_rate: REAL
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ;  Result := base * discount_rate end
end
```

Static Type vs. Dynamic Type

- In Java:

```
Student s = new Student("Alan");
Student rs = new ResidentStudent("Mark");
```

- In Eiffel:

ST

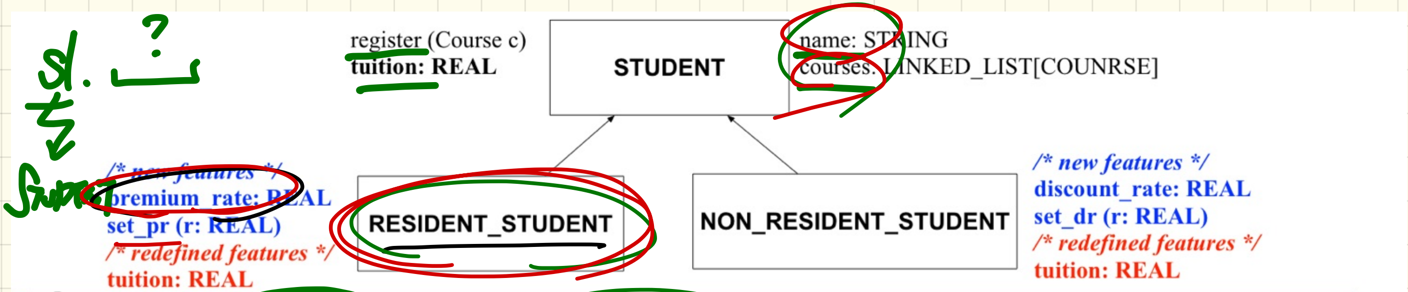
DT

```
local s: STUDENT
      rs: STUDENT
do create ~~{STUDENT}~~ s.make ("Alan")
   create {RESIDENT_STUDENT} rs.make ("Mark")
```

  - In Eiffel, the *dynamic type* can be omitted if it is meant to be the same as the *static type*:

```
local s: STUDENT
do create s.make ("Alan")
```

Student Classes (with Inheritance): Expectations



```
s1, s2, s3: STUDENT ; rs: RESIDENT_STUDENT ; nrs : NON_RESIDENT_STUDENT
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")
```

Create {RS} s.make

	name	courses	reg	tuition	pr	set_pr	dr	set_dr
s1	✓	✓	✓	✓	✗	✗	✗	✗
s2	✓	✓	✓	✓	✗	✗	✗	✗
s3								
rs	✓	✓	✓	✓	✓	✓	✗	✗
nrs								

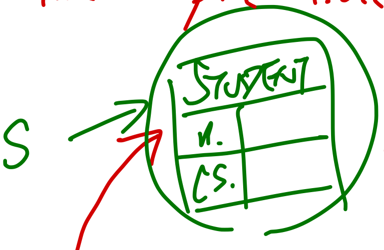
Polymorphism: Intuition

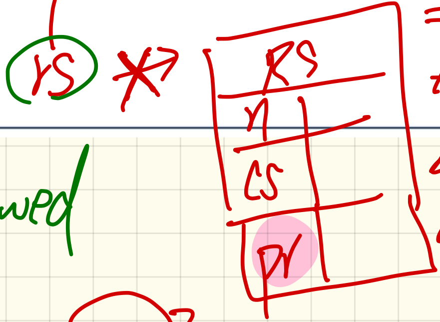
```
local
  s: STUDENT
  rs: RESIDENT_STUDENT
do
  create s.make ("Stella")
  create rs.make ("Rachael")
  rs.set_pr (1.25)
  s := rs /* Is this valid? */
  rs := s /* Is this valid? */
```

cs

ST of rs

if this was allowed at the compile time, we would get a runtime crash





⇒ this should be a compilation error.

Assume rs := s was allowed

What can expect to call on rs?

rs.pr

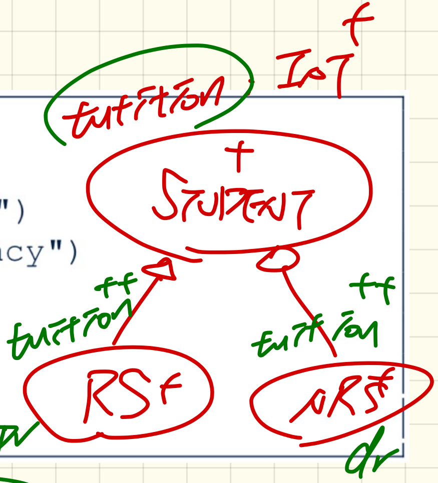
rs.set_pr

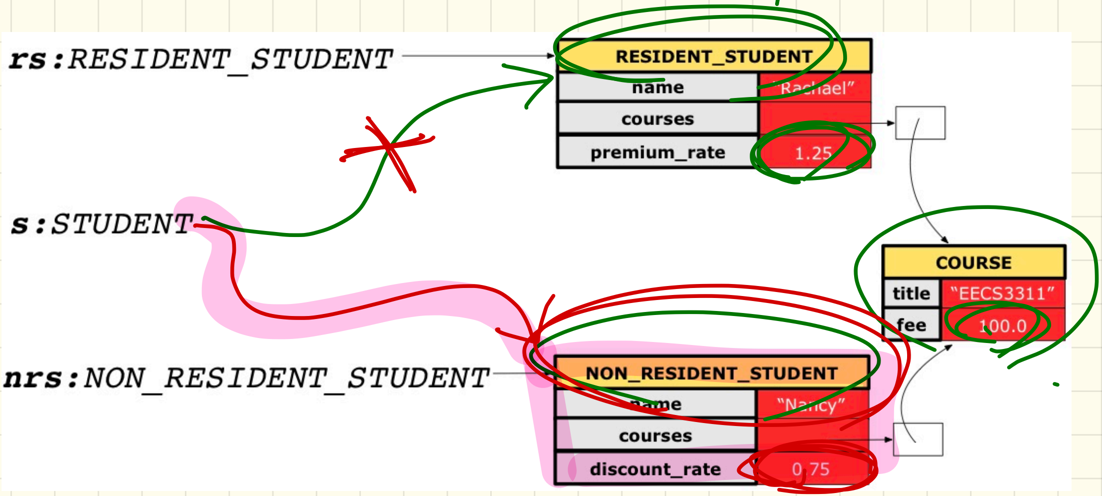
Crash ∵ no pr on a STUDENT object

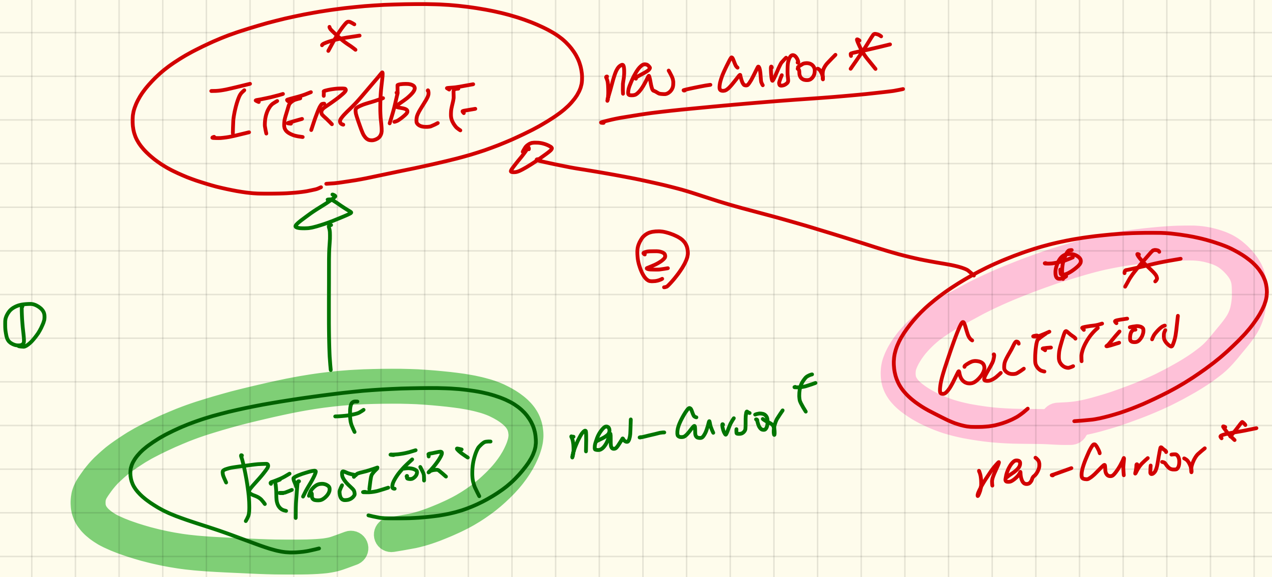
∴ rs := s not allowed.

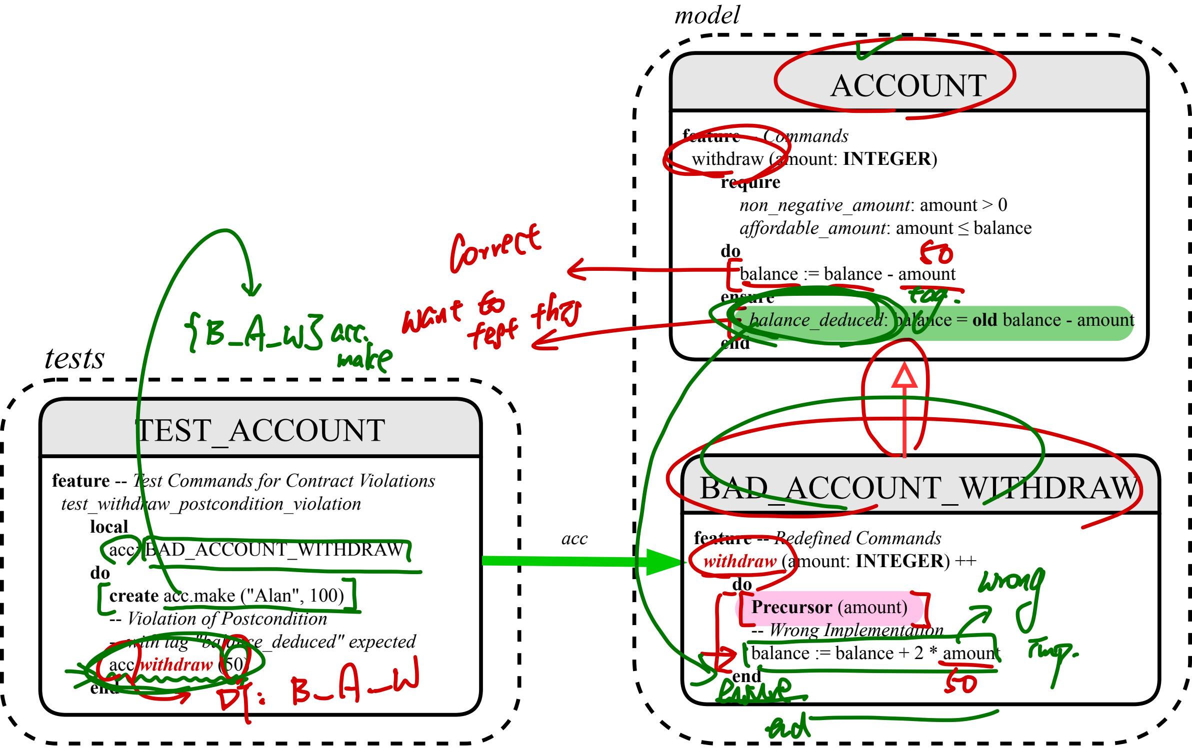
Dynamic Binding: Intuition

```
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
```






*
ITERABLE
new_cursor *
①
②
+
REPOSITORY
new_cursor +
+ *
COLLECTION
new_cursor *



model

ACCOUNT

```
feature -- Commands
  withdraw (amount: INTEGER)
      require
        non_negative_amount: amount > 0
        affordable_amount: amount ≤ balance
      do
        balance := balance - amount
      ensure
        balance_deducted: balance = old balance - amount
      end
```

tests

TEST_ACCOUNT

```
feature -- Test Commands for Contract Violations
  test_withdraw_postcondition_violation
      local
        acc: BAD_ACCOUNT_WITHDRAW
      do
        create acc.make ("Alan", 100)
        -- Violation of Postcondition
        -- with tag "balance_deducted" expected
        acc.withdraw (50)
      end
```

acc

BAD_ACCOUNT_WITHDRAW

```
feature -- Redefined Commands
  withdraw (amount: INTEGER) ++
      do
        Precursor (amount)
        -- Wrong Implementation
        balance := balance + 2 * amount
      end
```

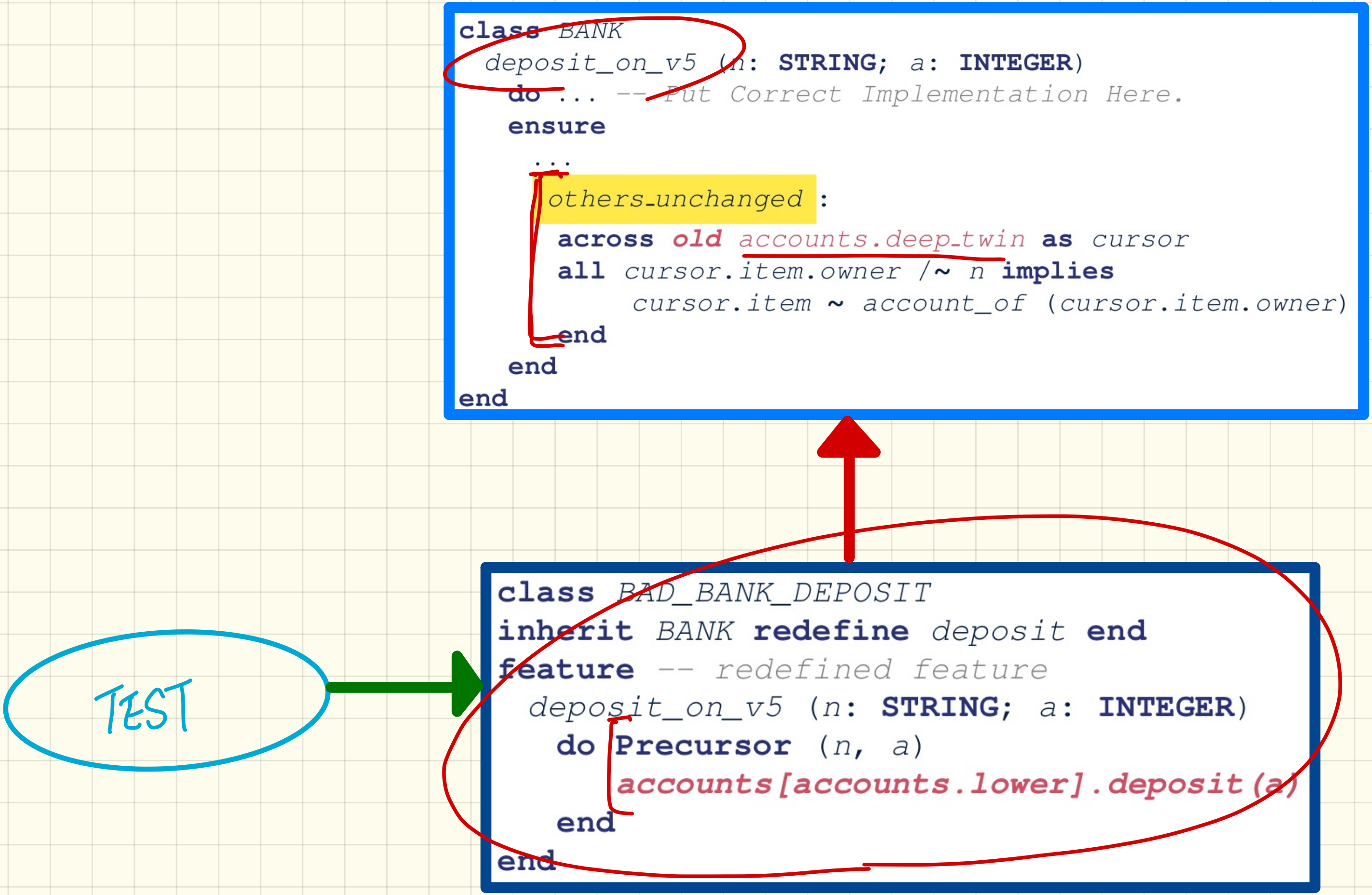
Adding Postcondition Tests

LAB 2

```
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
```

1. encourage to test postconditions
2. do not submit the new classes that you created

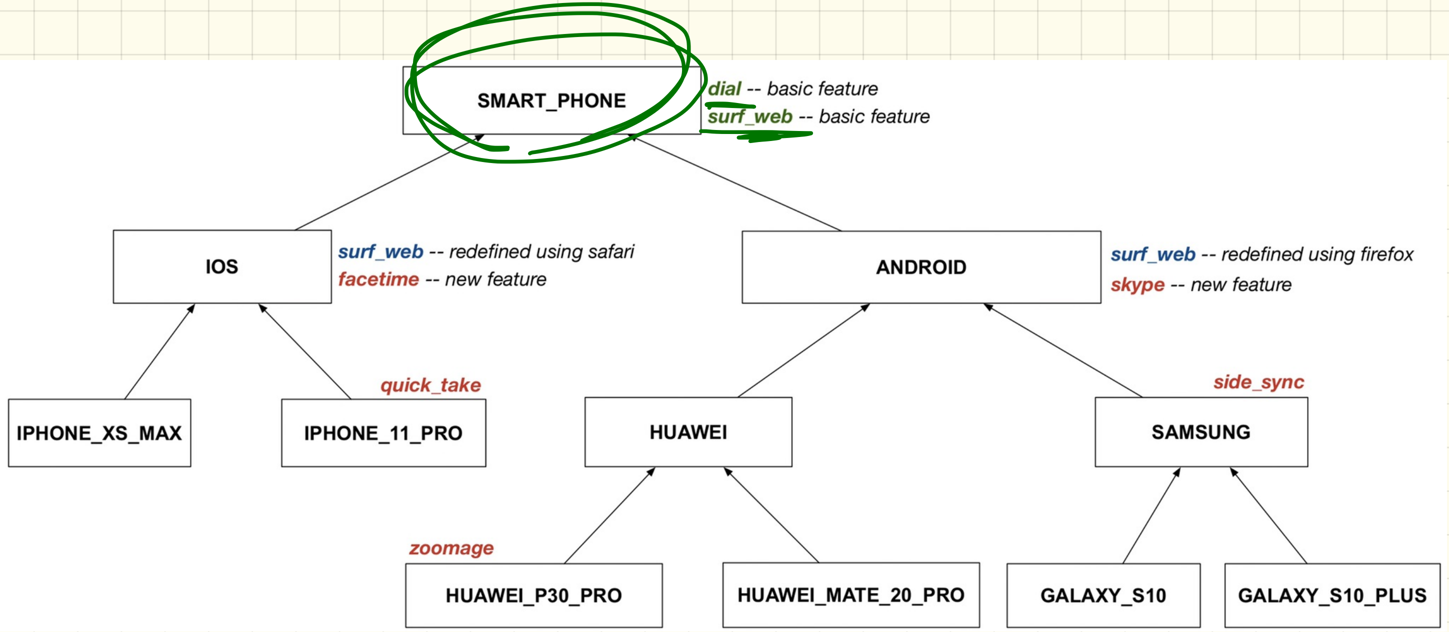
Testing of Postcondition: Exercise



```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    do ... -- Put Correct Implementation Here.
    ensure
      ...
      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
```

```
class BAD_BANK_DEPOSIT
inherit BANK redefine deposit end
feature -- redefined feature
  deposit_on_v5 (n: STRING; a: INTEGER)
    do Precursor (n, a)
       accounts[accounts.lower].deposit(a)
    end
end
```

Multi-Level **Inheritance** **Hierarchy** of Smartphones



jim : STUDENT

rs : RS

rs := jim

No. ? ST of jim (STUDENT) is not a descendant class of the ST of rs (RS)

S : STUDENT

S. tuition

↳ STUDEN

RS

NRS

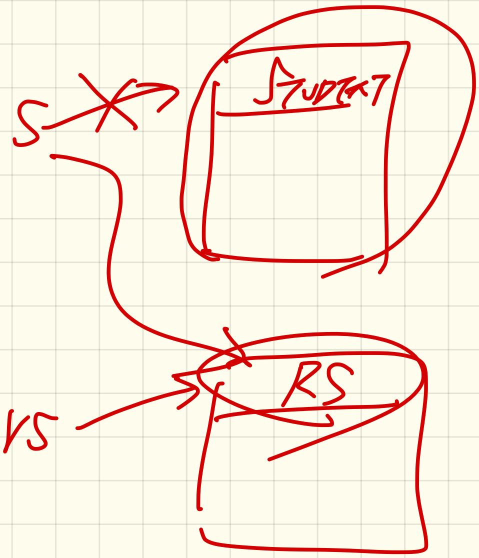
s : STUDENT

rs : RS

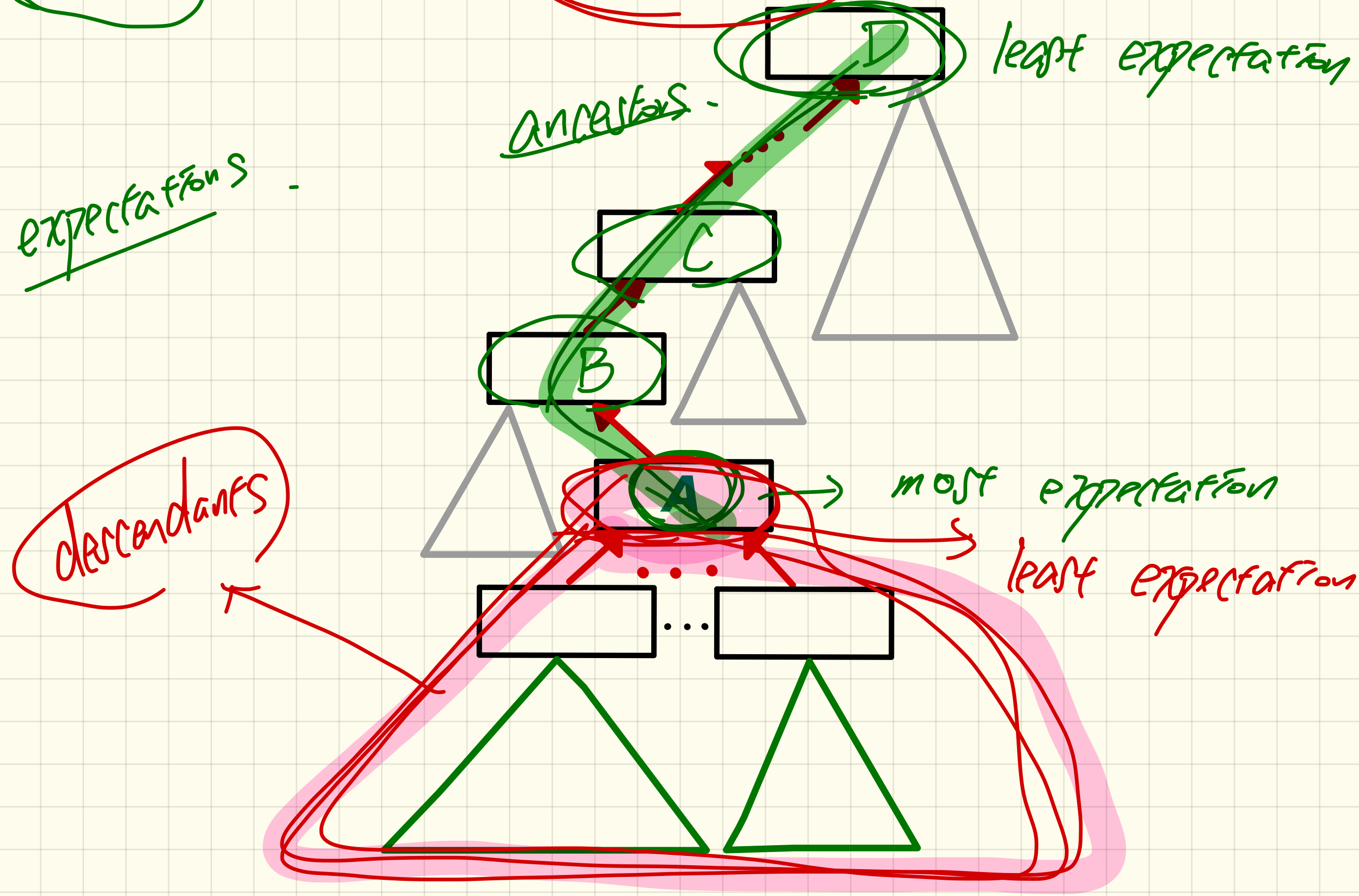
create s.make

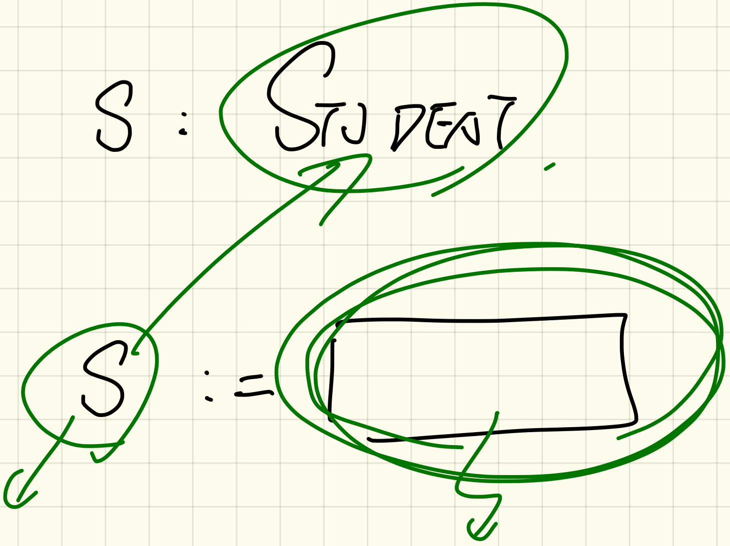
create rs.make

s := rs



Ancestors, Expectations, Descendants, and Code Reuse

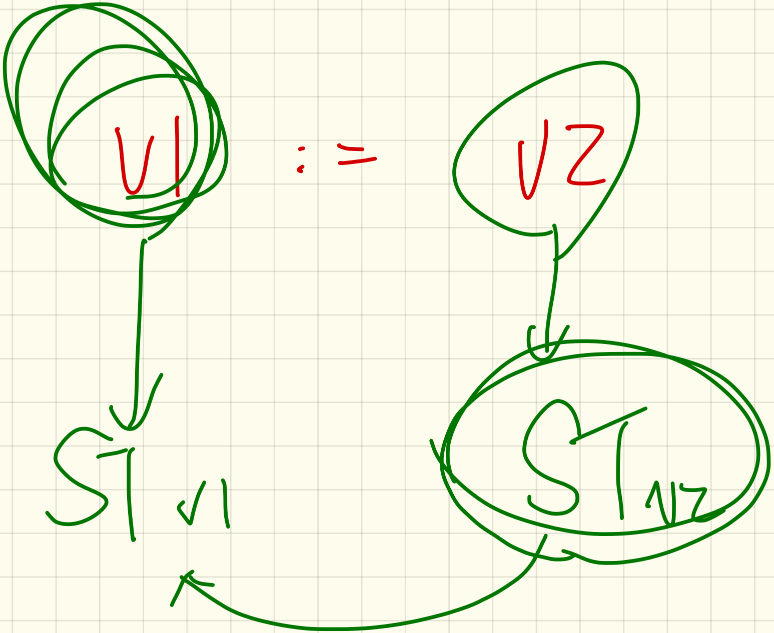




s. n

s. cs

any expression whose static type is a descendant of the ST of S.



a descendant class of ST_{v1}?

S : STUDENT

S. tuition

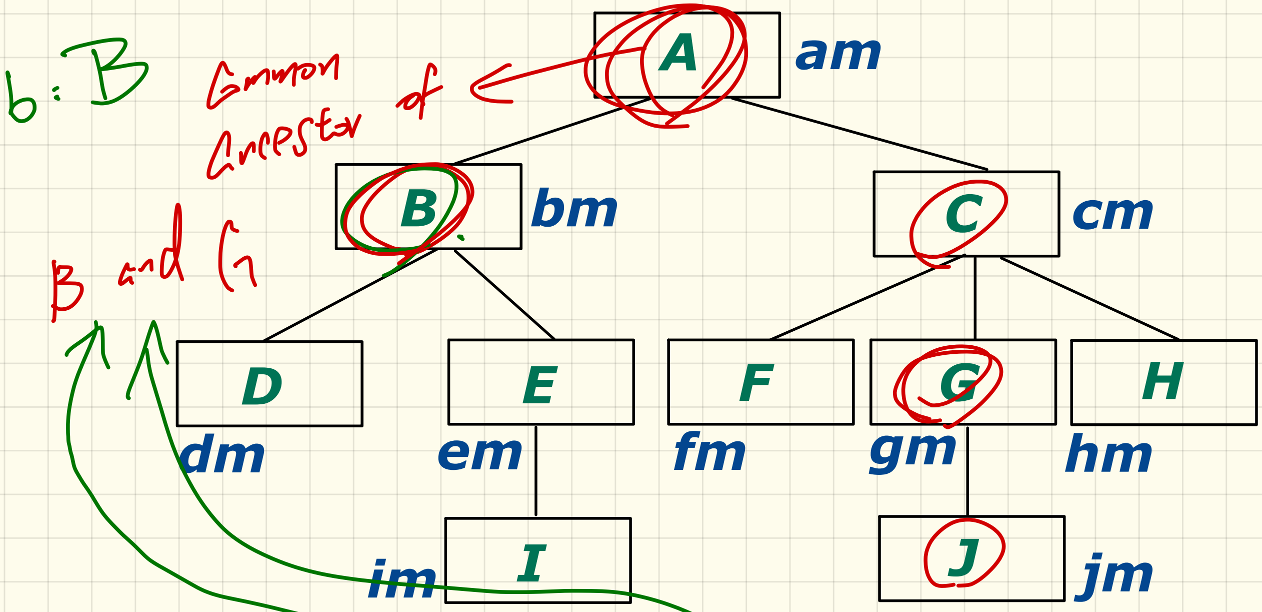
look up the
ST of S :

↳ does tuition exist in that ST?

LECTURE 11

THURSDAY OCTOBER 10

Inheritance Forms a Type Hierarchy (1)

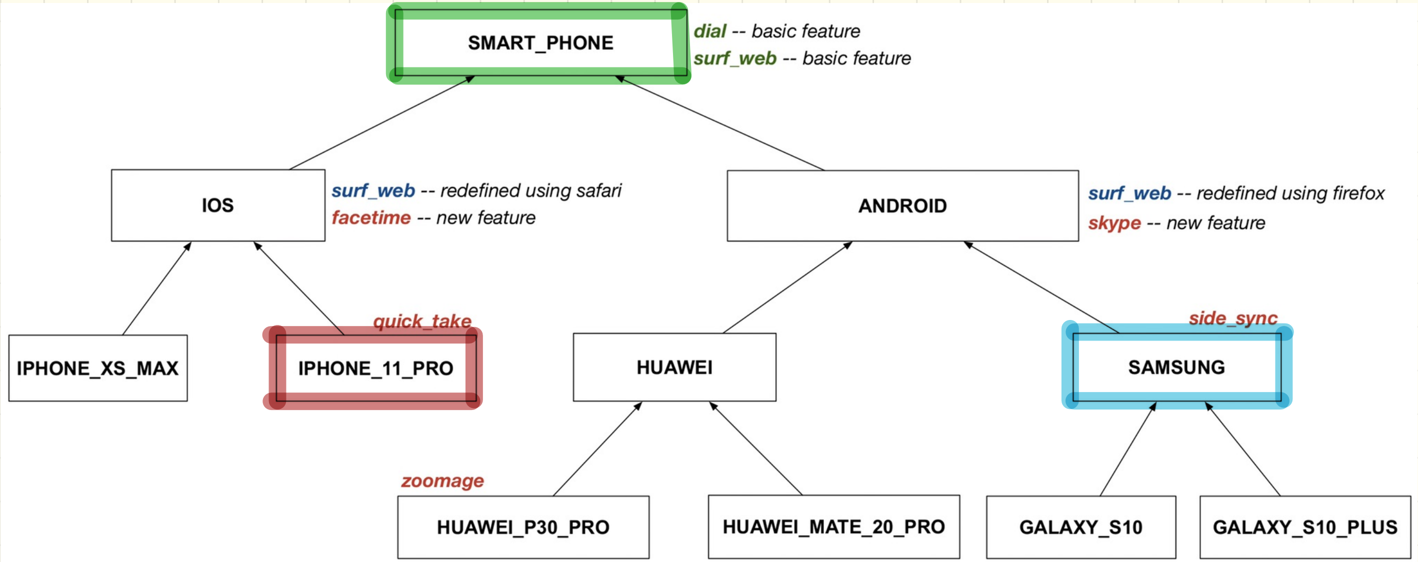


	ancestors	expectations	descendants
B	B, A	am bm	B D E I
G	G C A	gm cm am	
J	J G C A	jm gm cm am	

∵ G is an ancestor of J

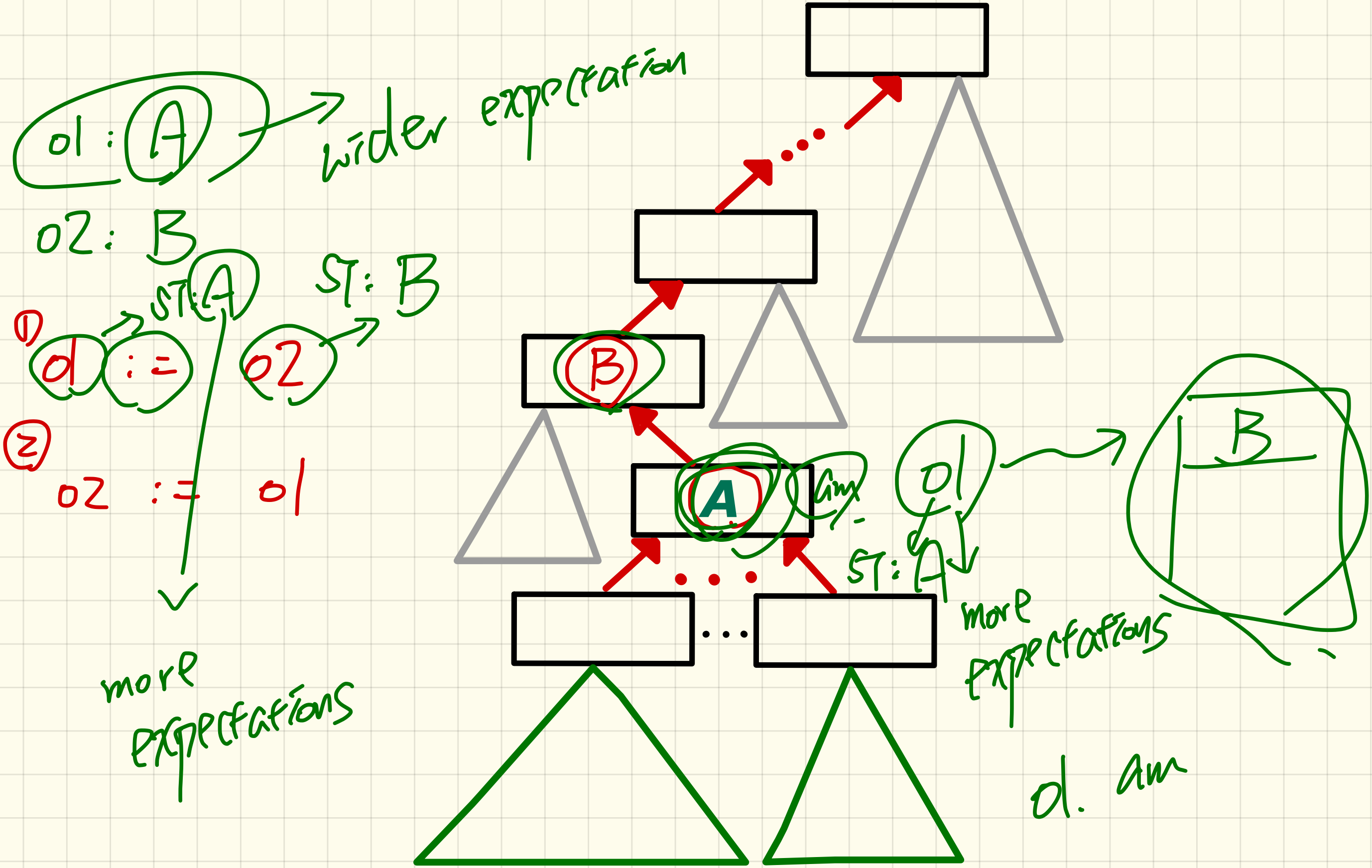
$E(G) \subseteq E(J)$

Inheritance Forms a Type Hierarchy (2)



	ancestors	expectations	descendants
🟩			
🟦			
🟥			

Ancestors, Expectations, Descendants, and Code Reuse



ol : B

Create {B} ol. make

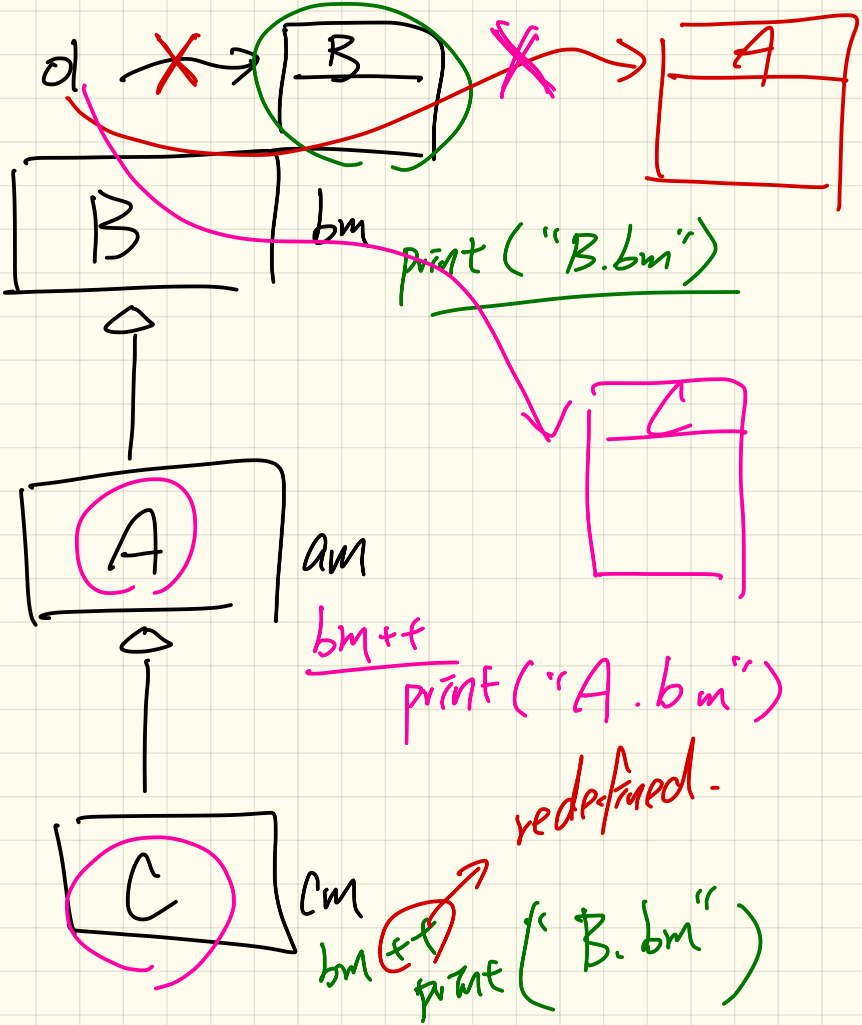
ol. bm → B.bm

Create {A} ol. make

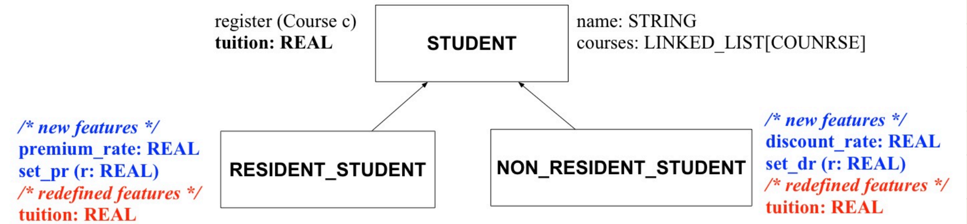
ol. bm → A.bm

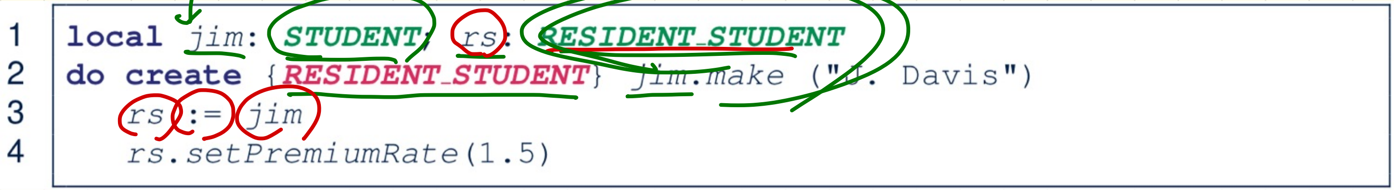
Create {C} ol. make

ol. bm → A.bm

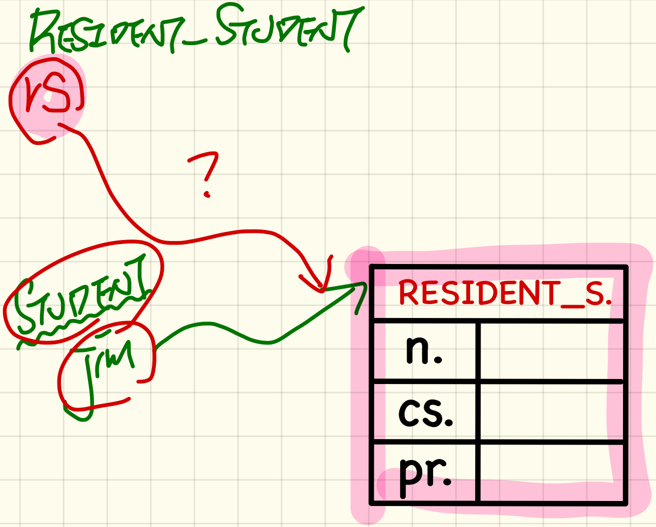


Type Cast: Motivation





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



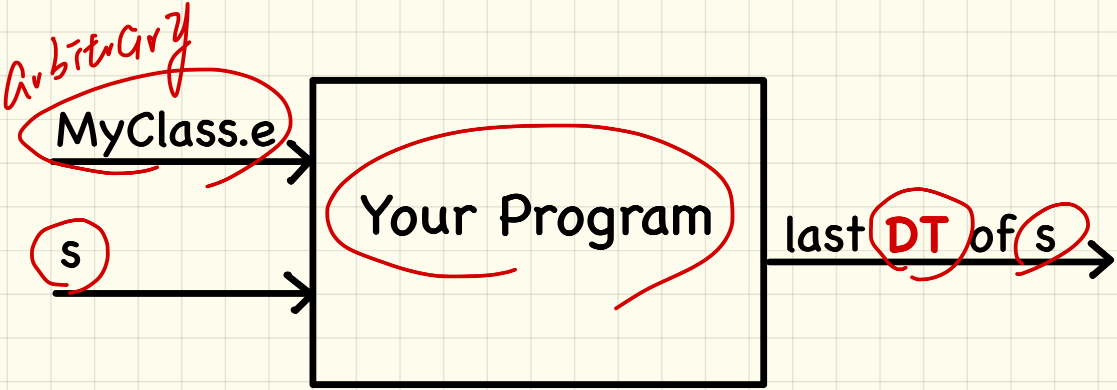
rs := jim

ST: RS

ST: STUDENT

X not descendant class.

Inferring the **DT** of a Variable is **Undecidable**



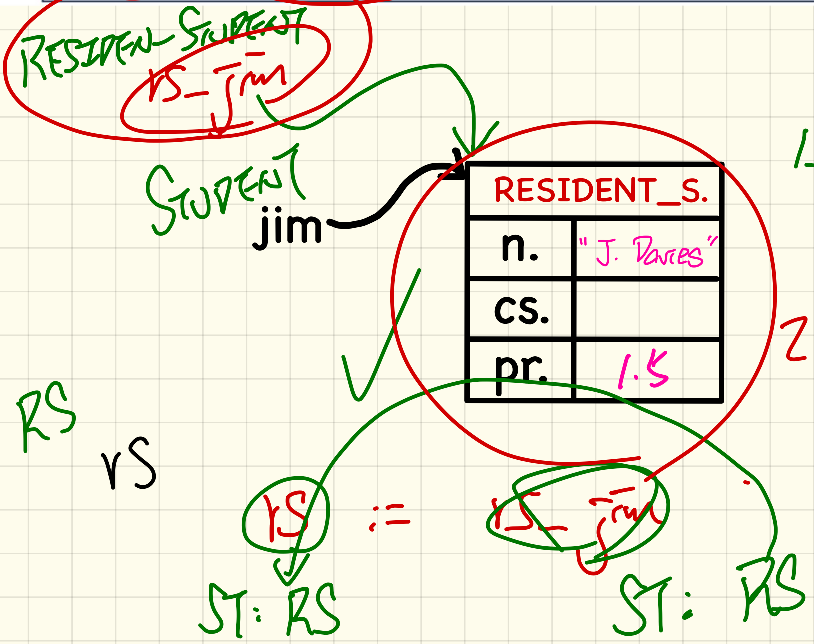
```
class MyClass
  make
    local
      s: STUDENT
    do
      create {RESIDENT_STUDENT} s.make
    end
end
```

```
while ( ... ) {
  create {R_S} s.make
}
create {N_R_S} s.make
```

Type Cast: Syntax

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

rs := rs_jim



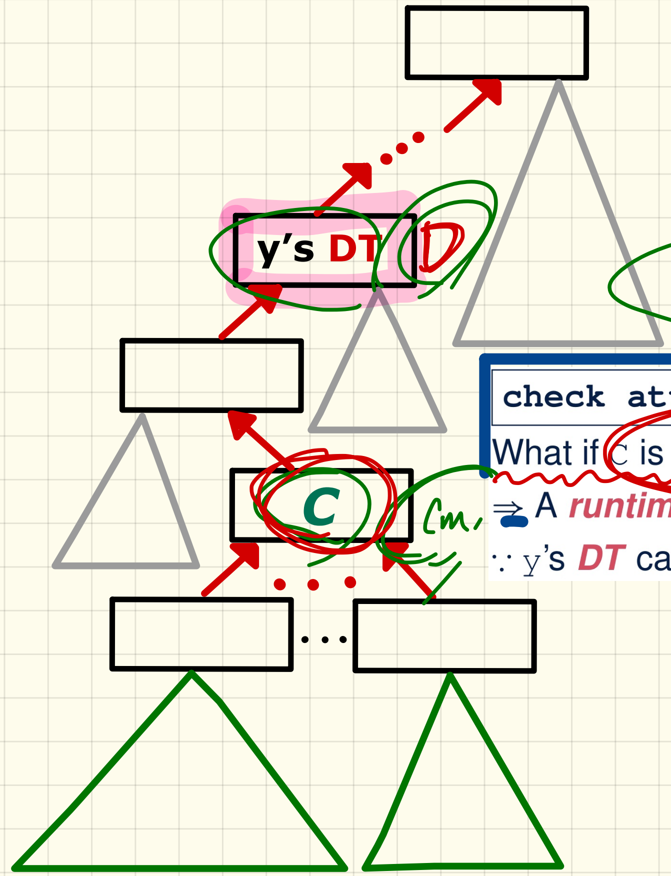
1. no new object created

2. ST of jim was not modified

Boolean Exp.

```
check attached {RS} jim as rs_jim
  and
  attached {NRS} alan as nrs_alan
then
  rs_jim      ST : RS
  nrs_alan    ST : NRS
end
```

Ancestors, Expectations, Descendants, and Code Reuse



Create {D} y.make

Assume we allowed this cast

⇒ ST of c_y : C

c_y.Cm as c_y.

`check attached {C} y then ... end`	*always compiles*

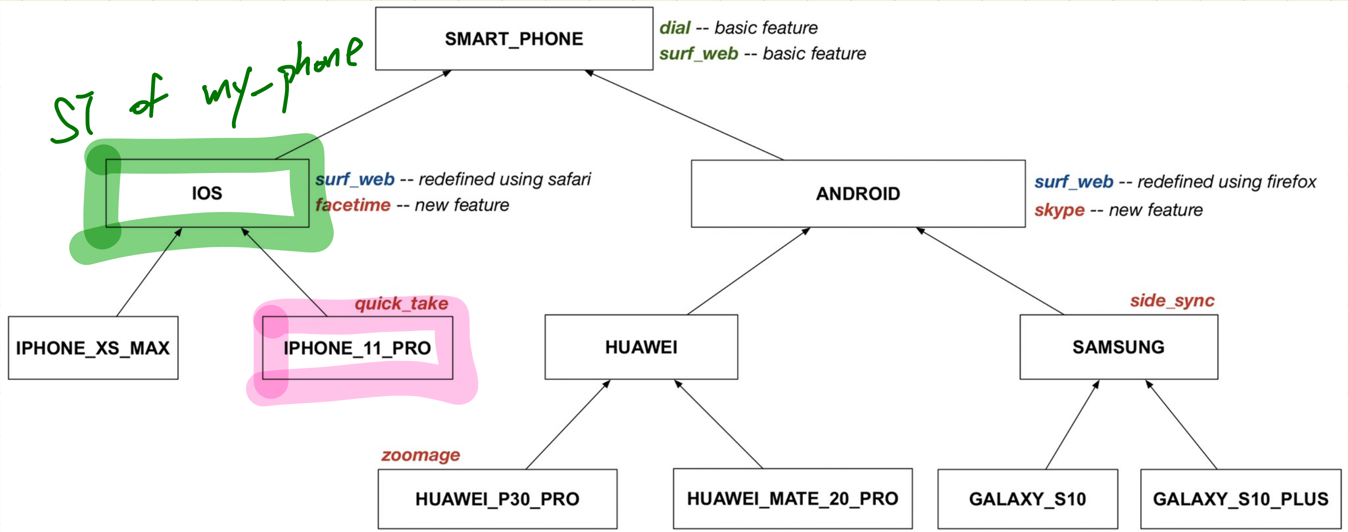
What if C is not an **ancestor** of y's ***DT***?

⇒ A ***runtime*** assertion violation occurs!

∵ y's ***DT*** cannot fulfill the expectation of C.

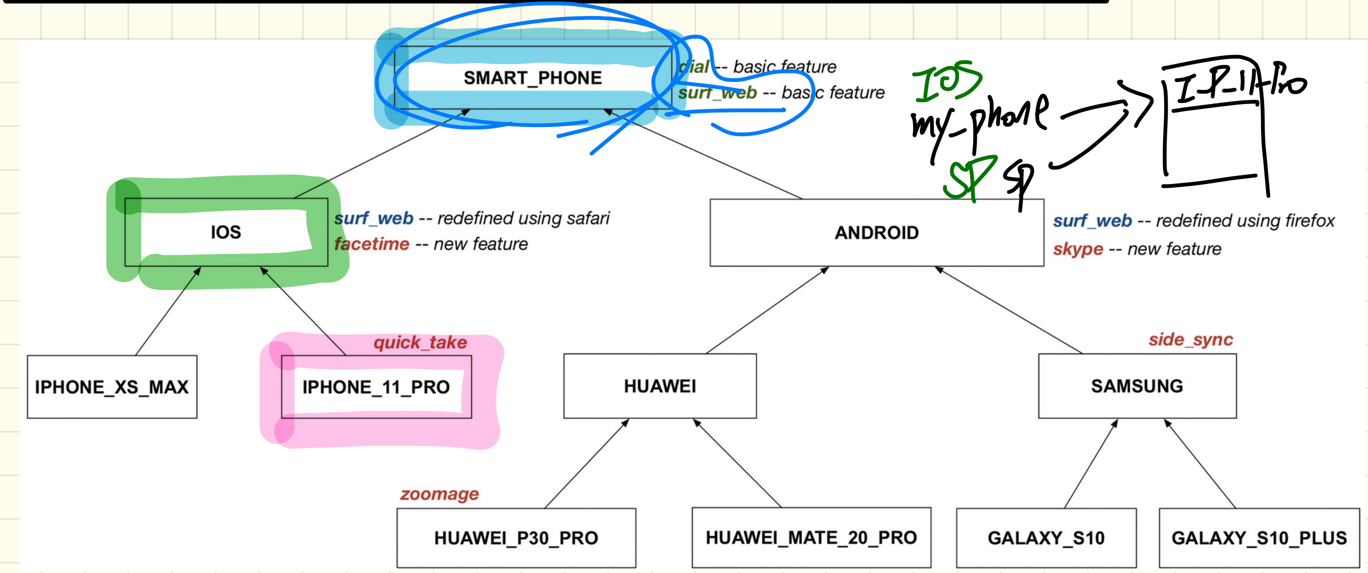
for a cast to be successful, "C" must be an ancestor of y's DT.

Violation-Free Cast: Upwards or Downwards (1)



```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime ● quick_take, skype, side_sync, zoomage ●
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ● facetime, quick_take, skype, side_sync, zoomage ●
end
check attached {IPHONE_11_PRO} my_phone as ip11_pro then
  -- can now call features defined in IPHONE_11_PRO on ip11_pro
  -- dial, surf_web, facetime, quick_take ● skype, side_sync, zoomage ●
end
```

Violation-Free Cast: Upwards or Downwards (2)

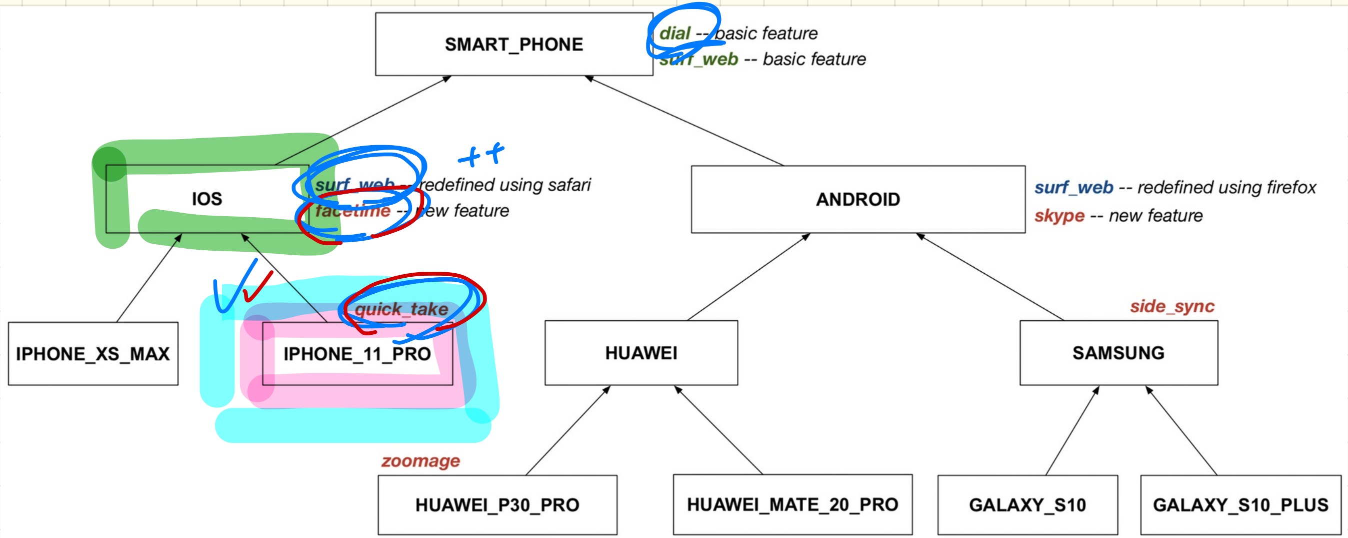


```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime ● quick_take, skype, side_sync, zoomage ●
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ● facetime, quick_take, skype, side_sync, zoomage ●
end
check attached {IPHONE_11_PRO} my_phone as ip11_pro then
  -- can now call features defined in IPHONE_11_PRO on ip11_pro
  -- dial, surf_web, facetime, quick_take ● skype, side_sync, zoomage ●
end
```

ST: IOS

ST: SMART_PHONE

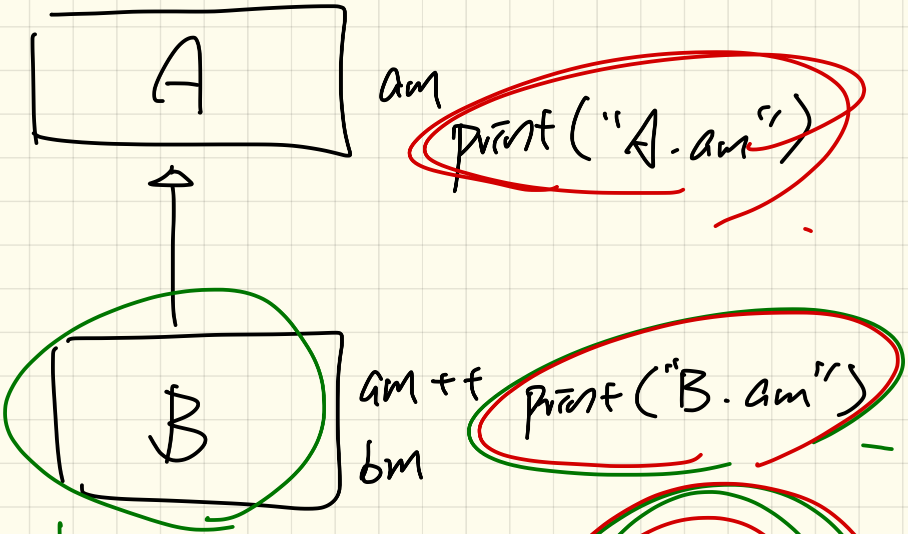
Violation-Free Cast: Upwards or Downwards (3)



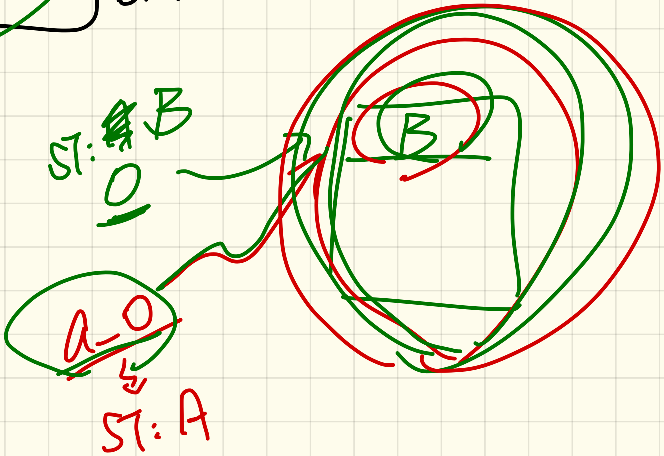
```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime ● quick_take, skype, side_sync, zoomage ●
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ● facetime, quick_take, skype, side_sync, zoomage ●
end
check attached {IPHONE_11_PRO} my_phone as ip11_pro then
  -- can now call features defined in IPHONE_11_PRO on ip11_pro
  -- dial, surf_web, facetime, quick_take ● skype, side_sync, zoomage ●
end
```

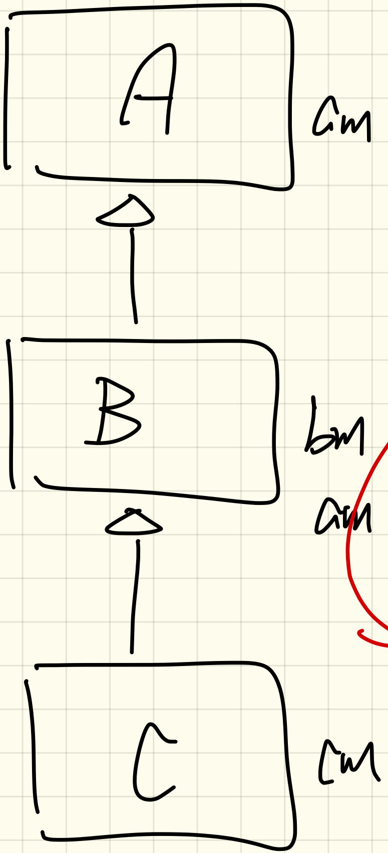
DT of m_p to IP_11_Pro

ip11_pro can be expected features of ancestors of



```
o : B

create {B} o.make

o.am → "B.am"

check {A} o as a_O then
  ST: A
  a_O.am → "B.am"
end
```





b : ~~??~~ A

check attached {C} b as c_o

end

whether this cast succeeds at runtime depends on if C is an ancestor of b's DT.

Does a line compile?

ST

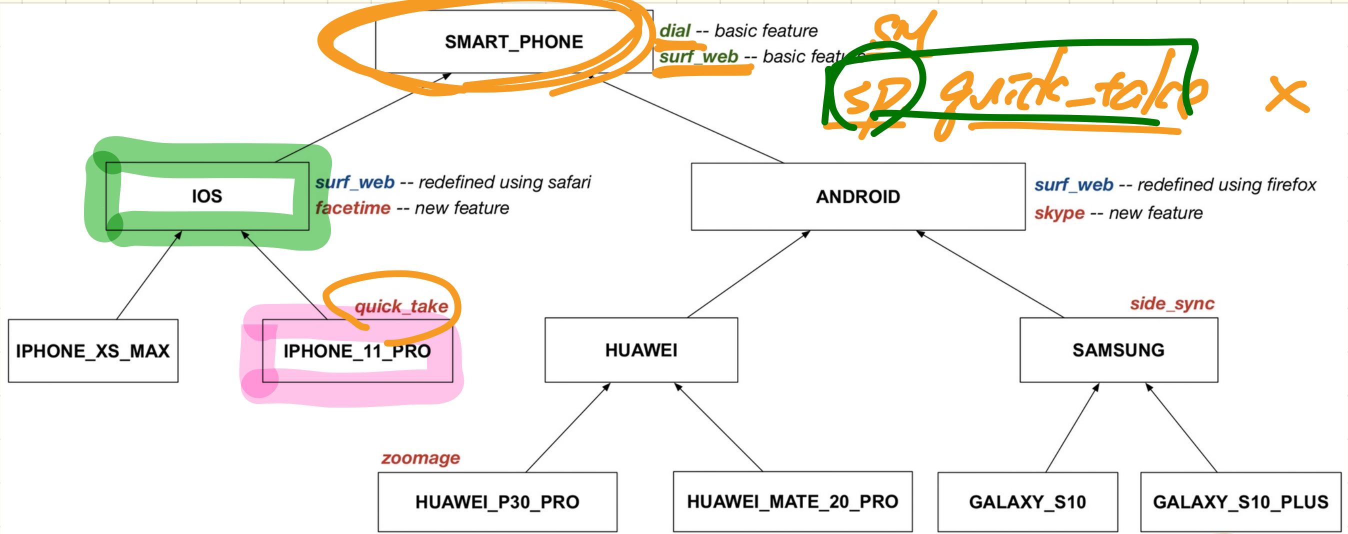
Version of feature called?

DT

LECTURE 12

TUESDAY OCTOBER 22

Violation-Free Cast: Upwards or Downwards (2)



sy

sp. quick_take ×

```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime, ● quick_take, skype, side_sync, zoomage ●
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ● facetime, quick_take, skype, side_sync, zoomage ●
end
check attached {IPHONE_11_PRO} my_phone as ip11_pro then
  -- can now call features defined in IPHONE_11_PRO on ip11_pro
  -- dial, surf_web, facetime, quick_take ● skype, side_sync, zoomage ●
end
```

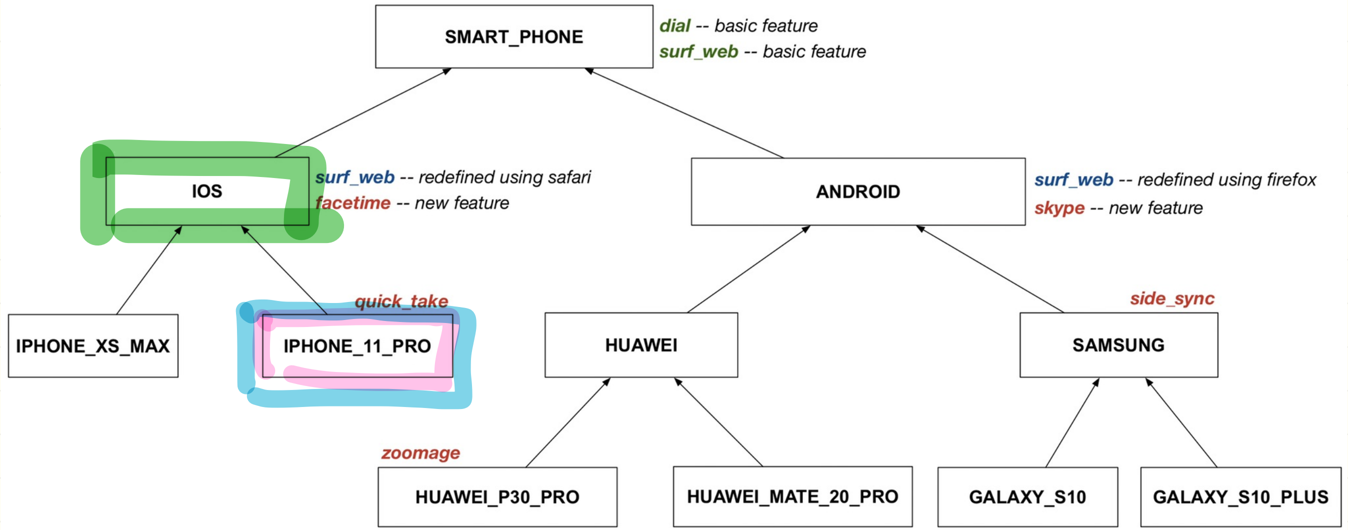
IOS my_phone

sp

sp

IP_11_Pro

Violation-Free Cast: Upwards or Downwards (3)



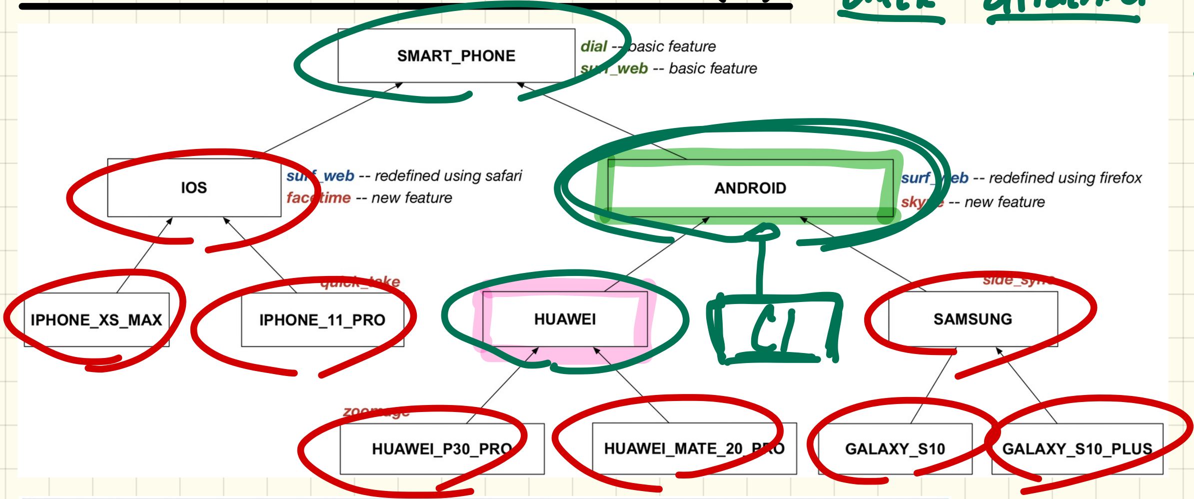
```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime ● quick_take, skype, side_sync, zoomage ●
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ● facetime, quick_take, skype, side_sync, zoomage ●
end
check attached {IPHONE_11_PRO} my_phone as ip11_pro then
  -- can now call features defined in IPHONE_11_PRO on ip11_pro
  -- dial, surf_web, facetime, quick_take ● skype, side_sync, zoomage ●
end
```

Cast Violation at Runtime (1)

check attached {C1}

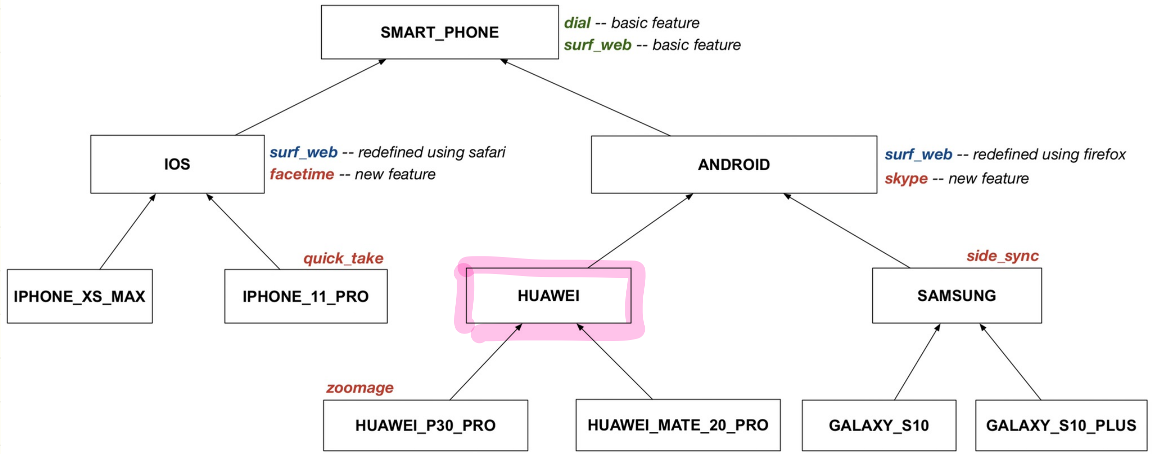
mine X

↳ assertion violation.



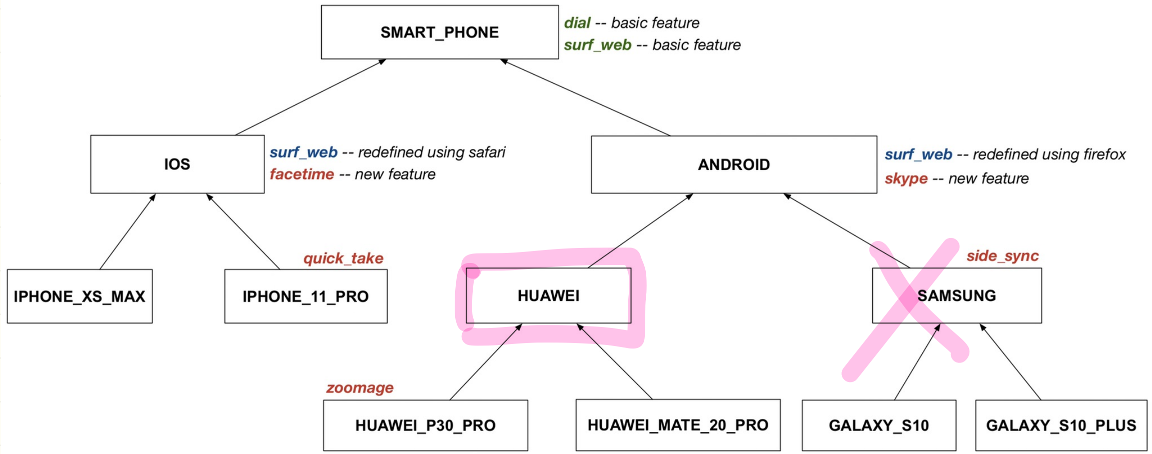
```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {HUAWEI} mine.make
    -- ST of mine is ANDROID; DT of mine is HUAWEI
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is HUAWEI
    check attached {HUAWEI} mine as huawei then ... end
    -- ST of huawei is HUAWEI; DT of huawei is HUAWEI
    check attached {SAMSUNG} mine as samsung then ... end
    -- Assertion violation
    -- ∵ SAMSUNG is not ancestor of mine’s DT (HUAWEI)
    check attached {HUAWEI_P30_PRO} mine as p30_pro then ... end
    -- Assertion violation
    -- ∵ HUAWEI_P30_PRO is not ancestor of mine’s DT (HUAWEI)
end
```

Cast Violation at Runtime (2)



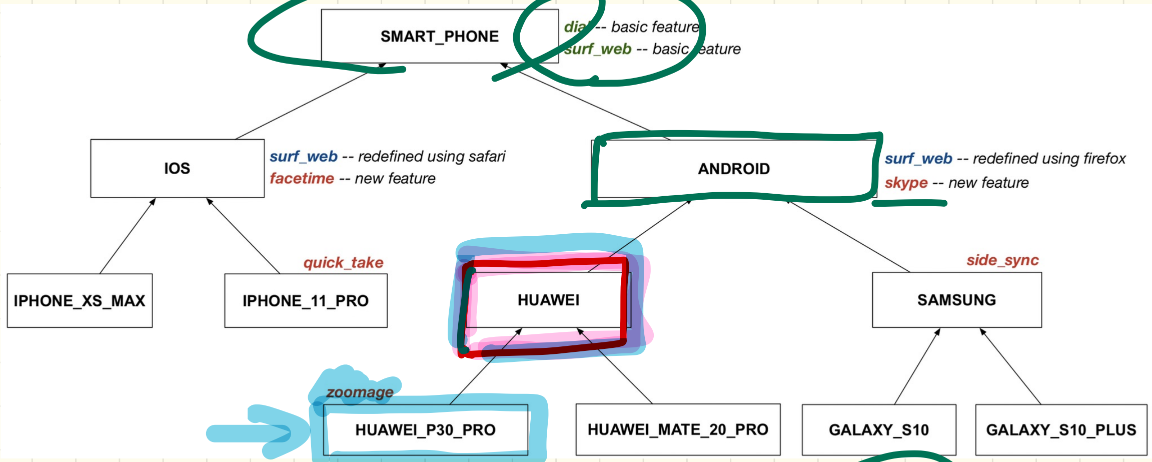
```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {HUAWEI} mine.make
    -- ST of mine is ANDROID; DT of mine is HUAWEI
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is HUAWEI
    check attached {HUAWEI} mine as huawei then ... end
    -- ST of huawei is HUAWEI; DT of huawei is HUAWEI
    check attached {SAMSUNG} mine as samsung then ... end
    -- Assertion violation
    -- ∵ SAMSUNG is not ancestor of mine's DT (HUAWEI)
    check attached {HUAWEI_P30_PRO} mine as p30_pro then ... end
    -- Assertion violation
    -- ∵ HUAWEI_P30_PRO is not ancestor of mine's DT (HUAWEI)
end
```

Cast Violation at Runtime (3)



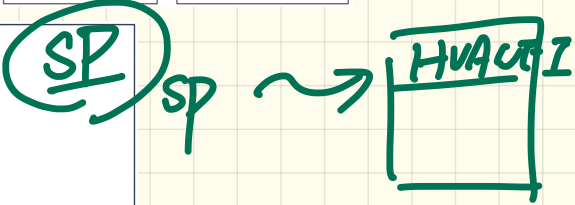
```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {HUAWEI} mine.make
    -- ST of mine is ANDROID; DT of mine is HUAWEI
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is HUAWEI
    check attached {HUAWEI} mine as huawei then ... end
    -- ST of huawei is HUAWEI; DT of huawei is HUAWEI
    check attached {SAMSUNG} mine as samsung then ... end
    -- Assertion violation
    -- ∵ SAMSUNG is not ancestor of mine’s DT (HUAWEI)
    check attached {HUAWEI_P30_PRO} mine as p30_pro then ... end
    -- Assertion violation
    -- ∵ HUAWEI_P30_PRO is not ancestor of mine’s DT (HUAWEI)
end
```

Cast Violation at Runtime (4)



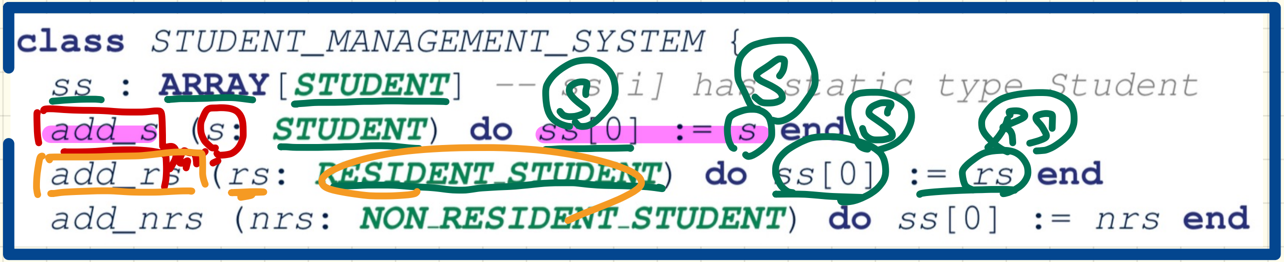
SMART_PHONE
dial -- basic feature
surf_web -- basic feature
IOS
surf_web -- redefined using safari
facetime -- new feature
ANDROID
surf_web -- redefined using firefox
skype -- new feature
IPHONE_XS_MAX
quick_take
IPHONE_11_PRO
HUAWEI
side_sync
SAMSUNG
zoomage
HUAWEI_P30_PRO
HUAWEI_MATE_20_PRO
GALAXY_S10
GALAXY_S10_PLUS

```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {HUAWEI} mine.make
    -- ST of mine is ANDROID; DT of mine is HUAWEI
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is HUAWEI
    check attached {HUAWEI} mine as huawei then ... end
    -- ST of huawei is HUAWEI; DT of huawei is HUAWEI
    check attached {SAMSUNG} mine as samsung then ... end
    -- Assertion violation
    -- ∵ SAMSUNG is not ancestor of mine's DT (HUAWEI)
    check attached {HUAWEI_P30_PRO} mine as p30_pro then ... end
    -- Assertion violation
    -- ∵ HUAWEI_P30_PRO is not ancestor of mine's DT (HUAWEI)
end
```



sp. skype ?

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
```

Say:

sms: STUDENT_MANAGEMENT_SYSTEM

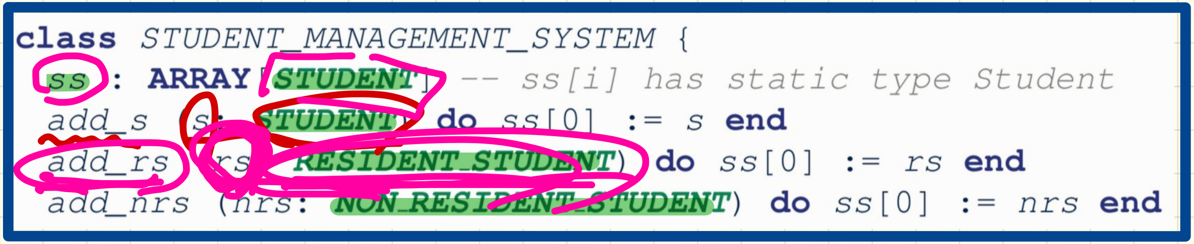
When should the following calls compile?

sms.add_s (o) → arg.   S := o

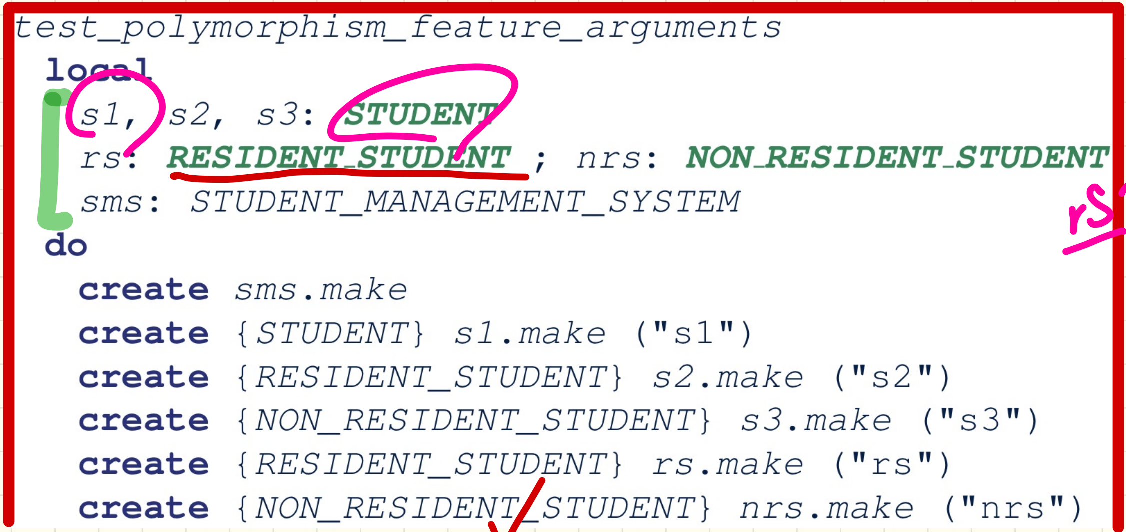
sms.add_rs (o) → rs := o   RS   → STo

sms.add_nrs (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
```

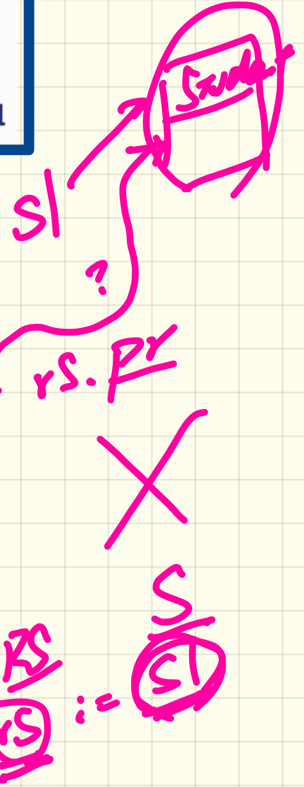


```
test_polymorphism_feature_arguments
  local
    s1, s2, s3: STUDENT
    rs: RESIDENT_STUDENT ; nrs: NON_RESIDENT_STUDENT
    sms: STUDENT_MANAGEMENT_SYSTEM
  do
    create sms.make
    create {STUDENT} s1.make ("s1")
    create {RESIDENT_STUDENT} s2.make ("s2")
    create {NON_RESIDENT_STUDENT} s3.make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

sms.add_s (rs) ✓ s := rs

S RS

sms.add_rs (s1)



s : ~~STUDENT~~ NRS → create {NRS} s.make(..)

s ✓

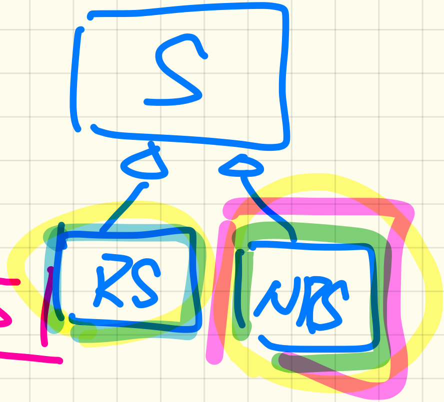
check attached {RS} as rs1 then

sms.add_rs(rs1)

end

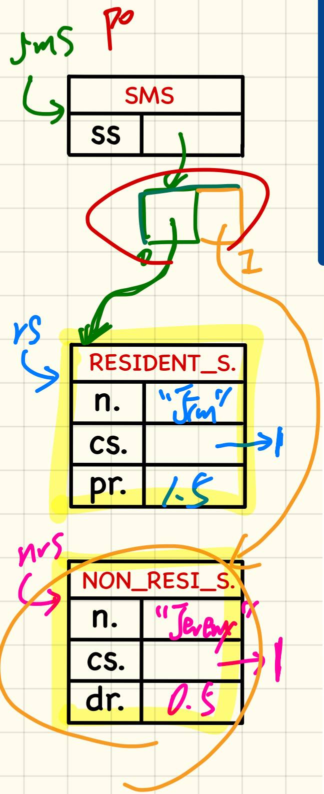
✓ rs := rs1

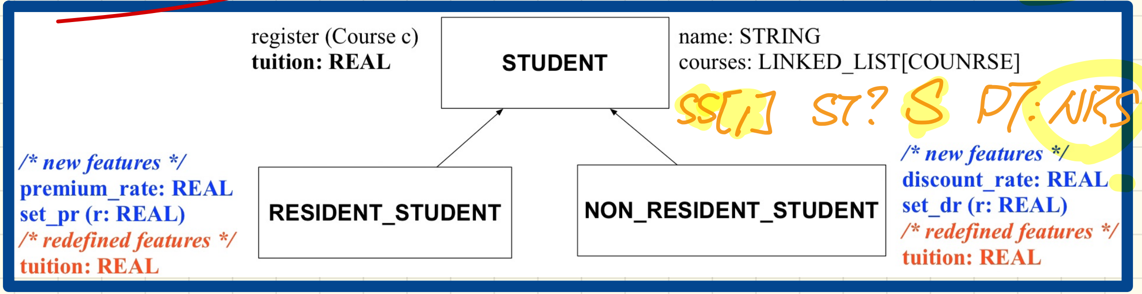
RS RS



Polymorphic Collection

SS[0] ST? S DT? RS





register (Course c)
tuition: REAL

STUDENT

name: STRING
courses: LINKED_LIST[COUNRSE]

SS[1] ST? S DT: NRS

/* new features */
premium_rate: REAL
set_pr (r: REAL)
/* redefined features */
tuition: REAL

RESIDENT_STUDENT

NON_RESIDENT_STUDENT

/* new features */
discount_rate: REAL
set_dr (r: REAL)
/* redefined features */
tuition: REAL

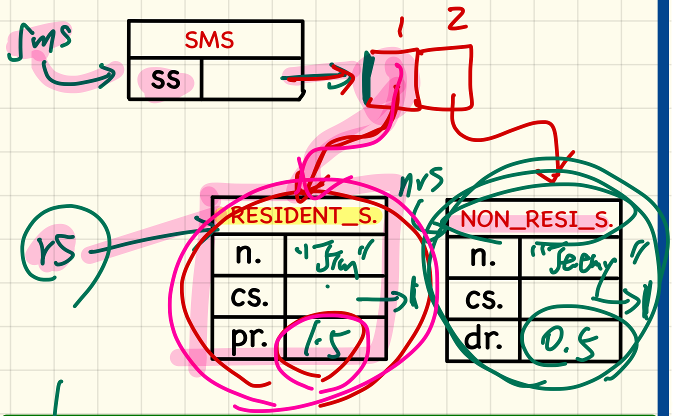
SS[0] := rS

```
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
```

```
class STUDENT_MANAGEMENT_SYSTEM
  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
```

Feature Call Return Values

sms.ss[i] vs. rs RS



STUDENT

```
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
```

2 / 2

Possible **DT** of **Result**?

```
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
```

ST

{RS}

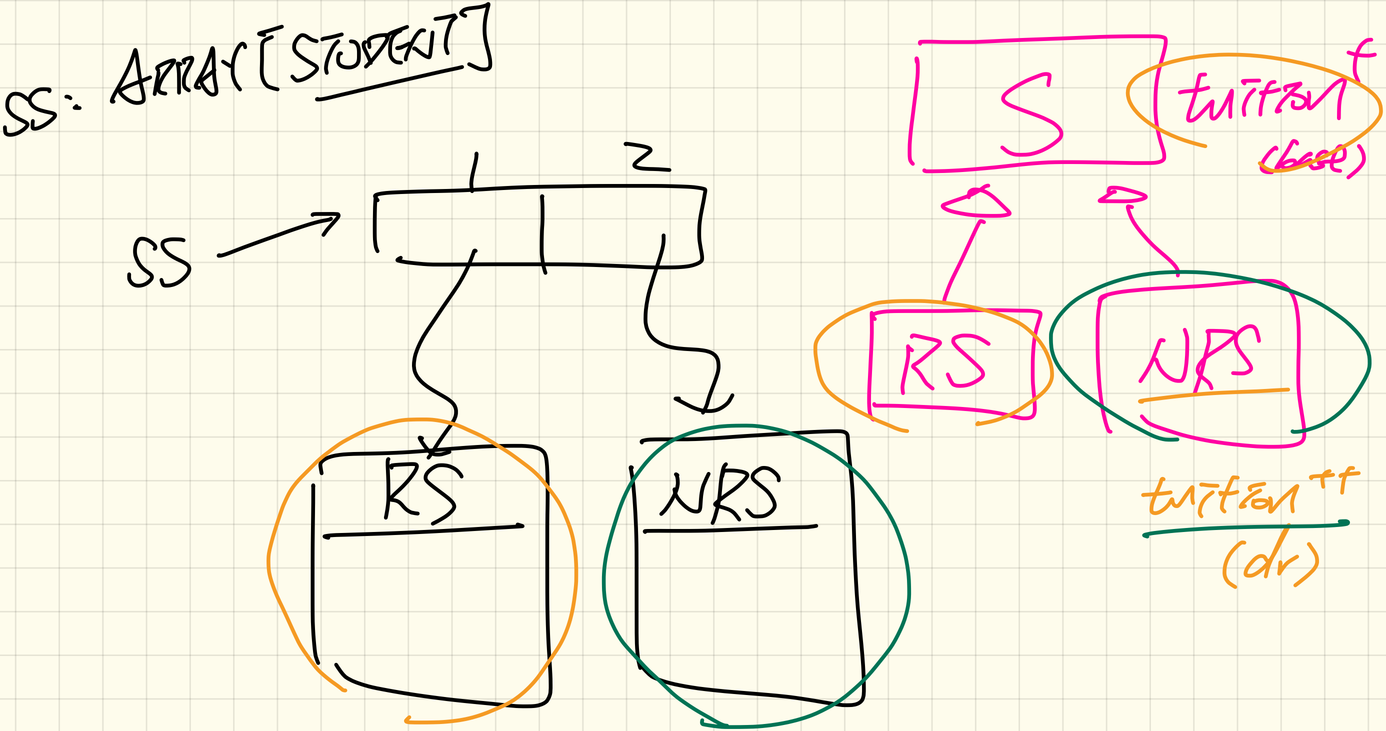
pr

dr

SS : u[RS]

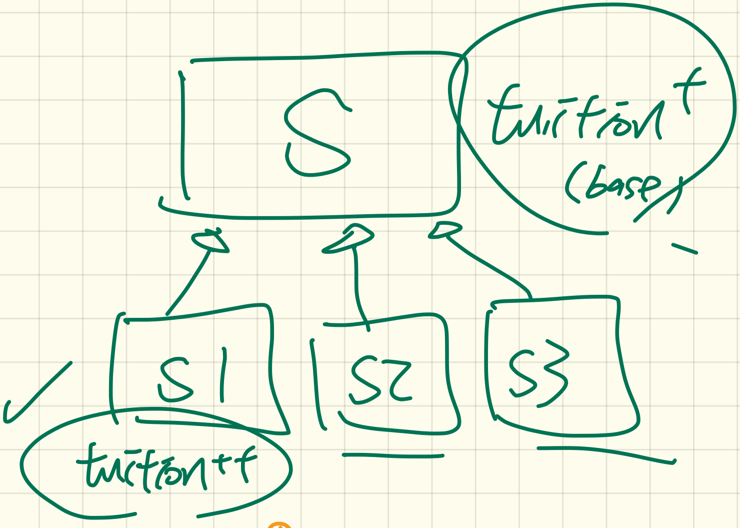
[ss[i].dr ✗

ST: RS

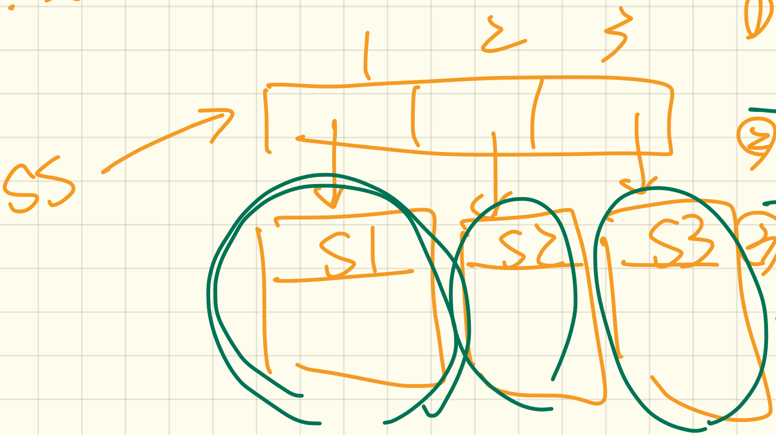


① SS[1].tuition

② SS[2].tuition



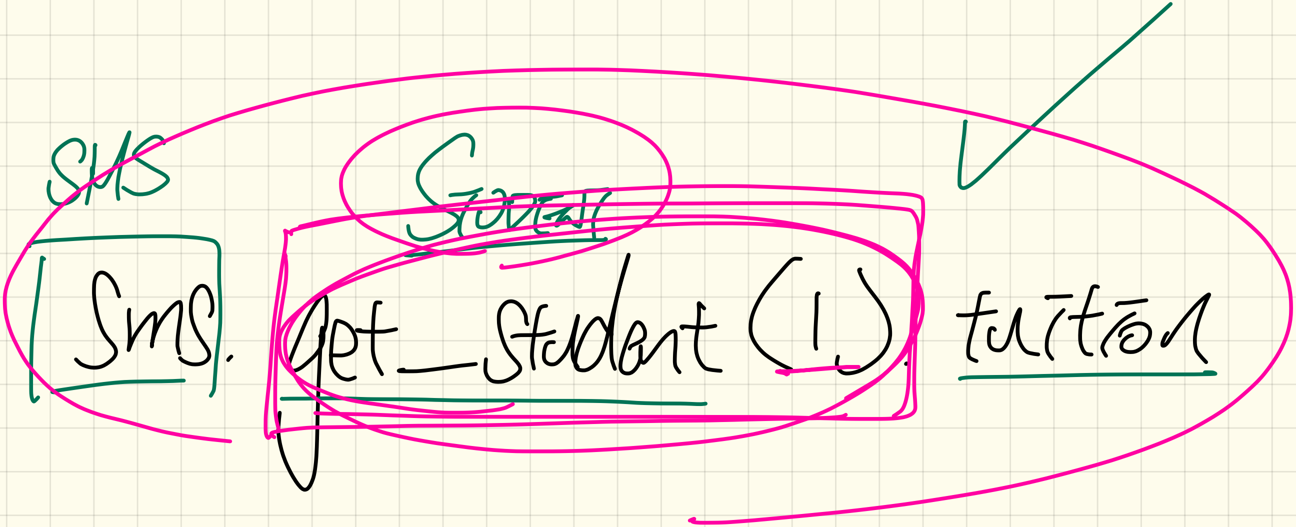
SS : A[STUDENT]



① SS[1].tuition

② SS[2].tuition

③ SS[3].tuition



`sms.get_student(1).tuition`

1. Compile?

2. version of tuition run?

LECTURE 13

THURSDAY OCTOBER 24

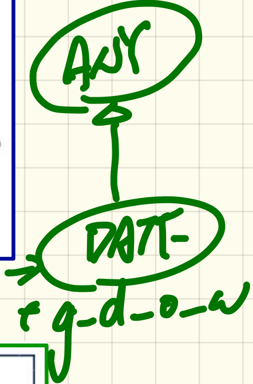
General Book

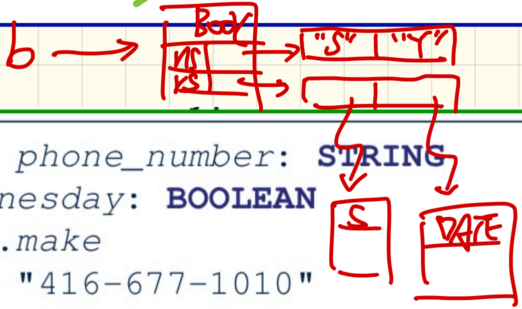
① compilation? ✓ ② runtime ✓ violation.

Supplier

```
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
```

```
check attached {DATE} b.get("SuYeon") as d then
  is_w := d.g_d_o_w = 4
end
```



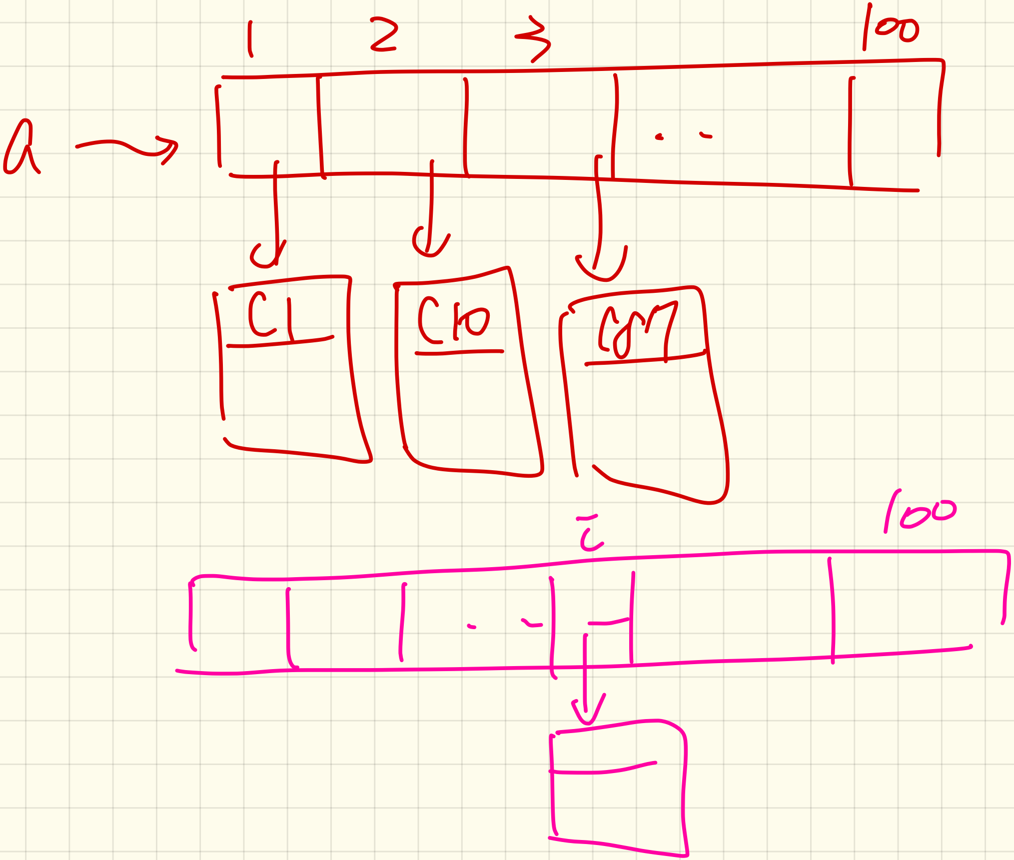


Client

```
birthday: DATE; phone_number: STRING
b: BOOK; is_wednesday: BOOLEAN
create {BOOK} b.make
phone_number := "416-677-1010"
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
```

ANY ✓

X

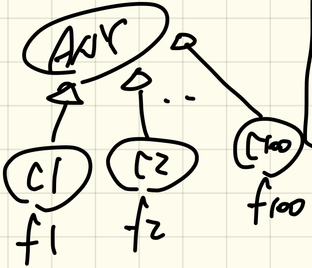

a
1
2
3
100
. .
C1
C10
C97
i
100
. .

General Book violates Single Choice Principle

Storage

```
rec1: C1
... -- declarations of rec2 to rec99
rec100: C100
create {C1} rec1.make(...) ; b.add(..., rec1)
... -- additions of rec2 to rec99
create {C100} rec100.make(...) ; b.add(..., rec100)
```

Retrievals



repetition!

```
-- assumption: 'f1' specific to C1, 'f2' specific to C2, etc.
if attached {C1} b.get("Jim") as c1 then
  c1.~~get("Jim")~~ f1
... -- cases for C2 to C99
~~elseif attached {C100} b.get("Jim") as c100 then~~
  ~~c100.f100~~
  elseif attached {C101} - - then } -
end
```

repetition?

```
-- assumption: 'f1' specific to C1, 'f2' specific to C2, etc.
if attached {C1} b.get("Jim") as c1 then
  c1.~~get("Jim")~~ f1
... -- cases for C2 to C99
~~elseif attached {C100} b.get("Jim") as c100 then~~
  ~~c100.f100~~
end
  elseif attached {C101} - - then - —
```

✓ What if a new type **C101** is introduced?

✓ What if type **C100** becomes obselete?

parameter

```
deposit (amount : REAL)
  do
    balance := balance + amount
  end
```

acc.deposit (23.4)

deposit (46.8)

param denotes a value

Generic Book

Supplier

generic parameter denoting some type, not value

```
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
```

DATE

DATE

DATE

DATE

use of parameter G.

local

acc: Account

b: Book[~~acc~~]

Client

type that instantiates G of Book.

```
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
```

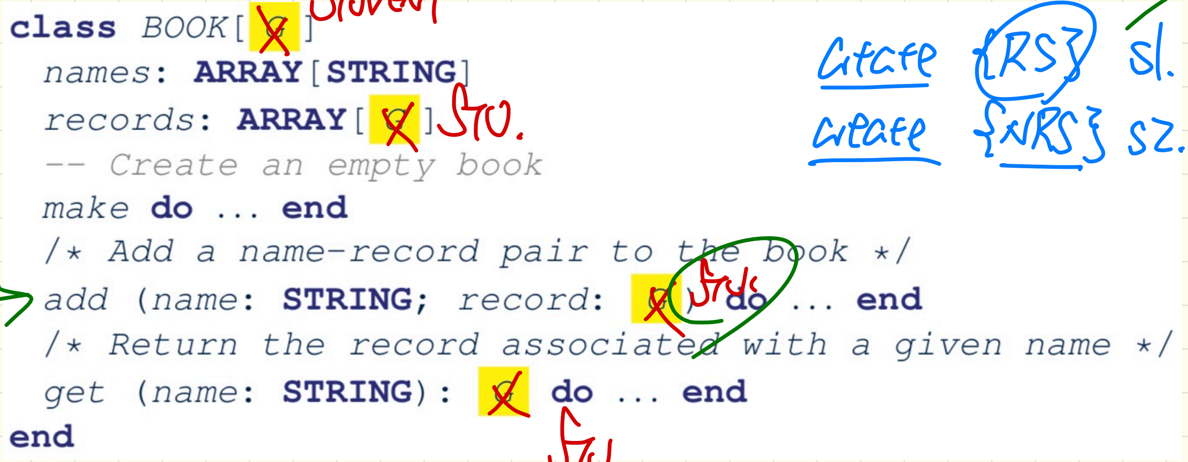
① ② STRING ③ DATE ④ ⑤

b2: Book[STUDENT]

s1, s2: STUDENT

create {RS} s1.make(..)

create {NRS} s2.make(..)



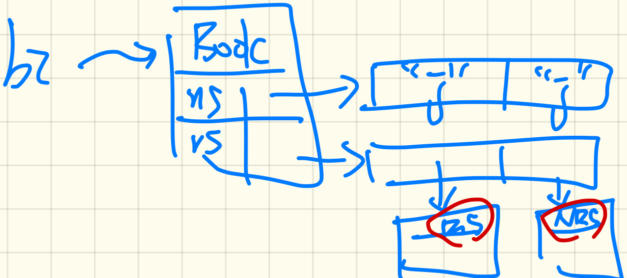
```
class BOOK[ G ]
  names: ARRAY[STRING]
  records: ARRAY[ G ]
  -- Create an empty book
  make do ... end
  /* Add a name-record pair to the book */
  add (name: STRING; record: G ) do ... end
  /* Return the record associated with a given name */
  get (name: STRING): G do ... end
end
```

STUDENT

STU.

STU.

STU.

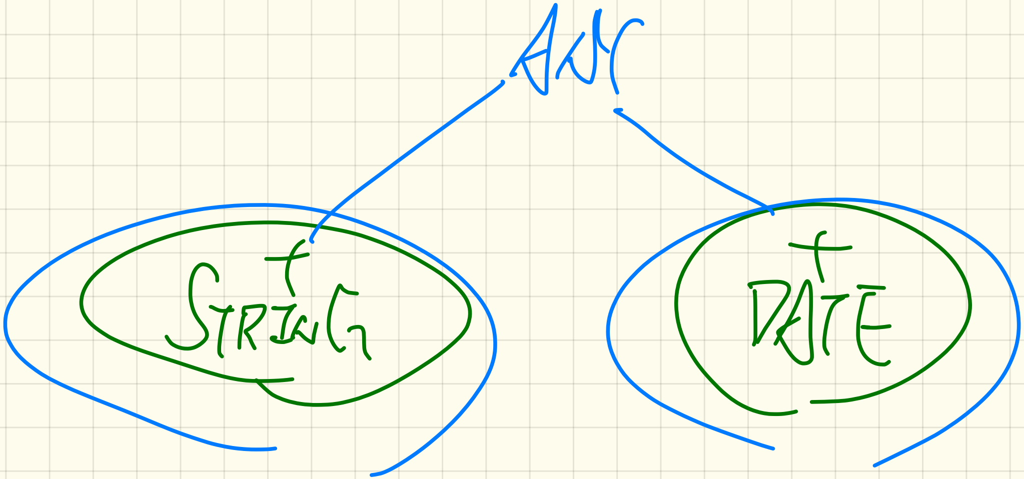


ST S

b2.add("jim", s1)

b2.add("jeremy", s2)

b3 : Book[ANY]



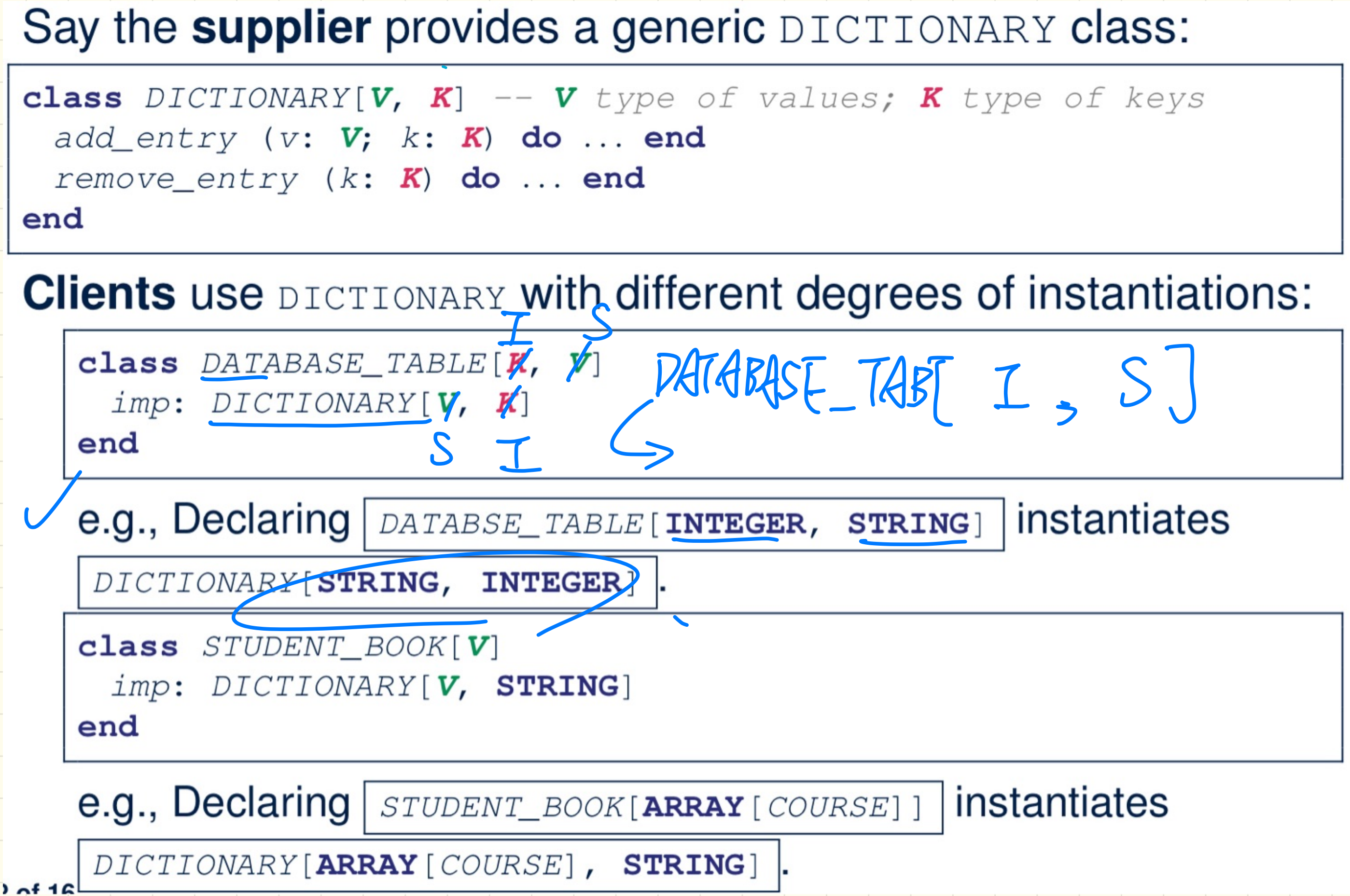
class Book2 [like acc] X

class Book [G -> STUDENT]

local

acc : Account
b : Book [the acc]

Instantiating Generic Parameters



Say the **supplier** provides a generic `DICTIONARY` class:

```
class DICTIONARY[V, K] -- V type of values; K type of keys
  add_entry (v: V; k: K) do ... end
  remove_entry (k: K) do ... end
end
```

Clients use `DICTIONARY` with different degrees of instantiations:

```
class DATABASE_TABLE[K, V]
  imp: DICTIONARY[V, K]
end
```

I S

S I

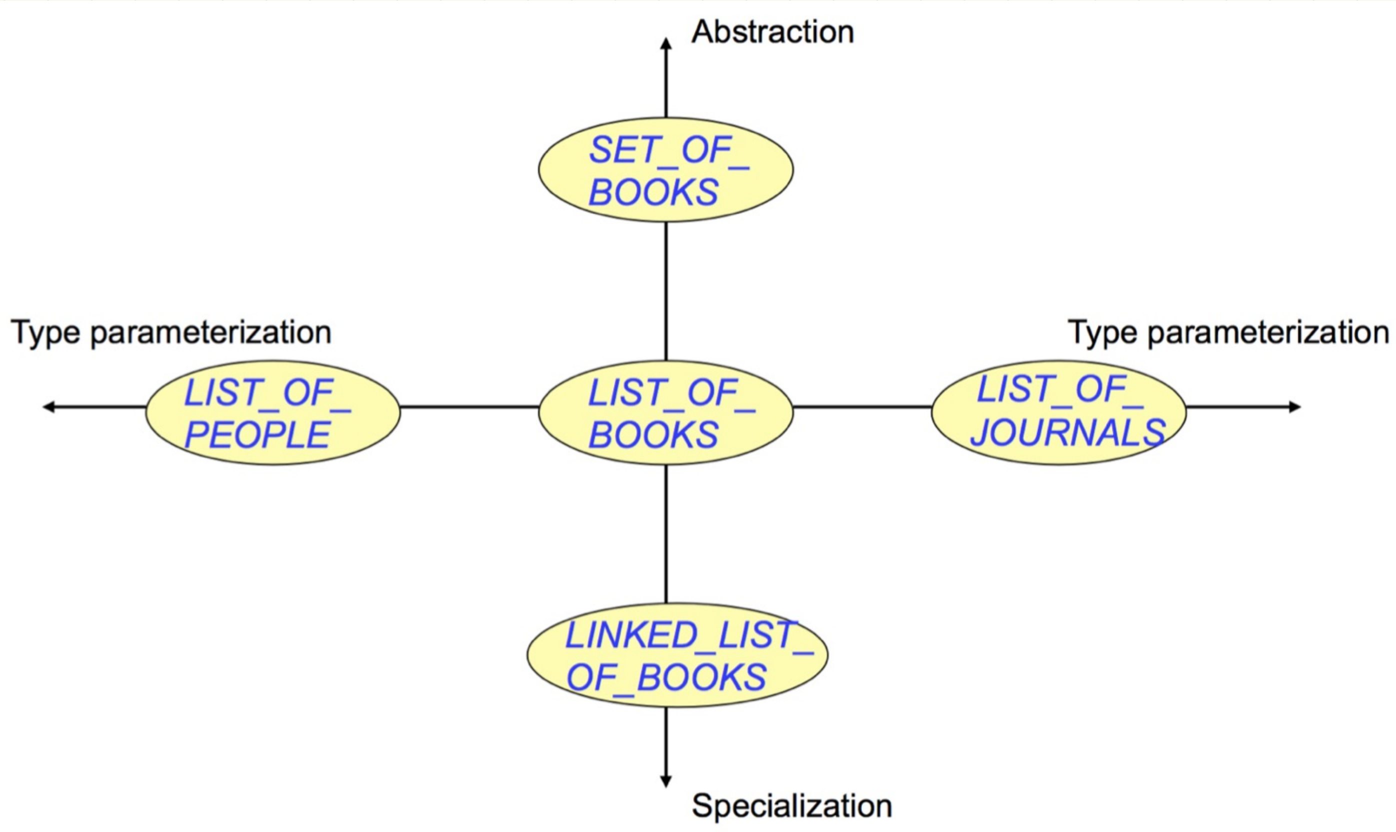
DATABASE_TABL[I , S]

✓ e.g., Declaring `DATABSE_TABLE[INTEGER, STRING]` instantiates `DICTIONARY[STRING, INTEGER]`.

```
class STUDENT_BOOK[V]
  imp: DICTIONARY[V, STRING]
end
```

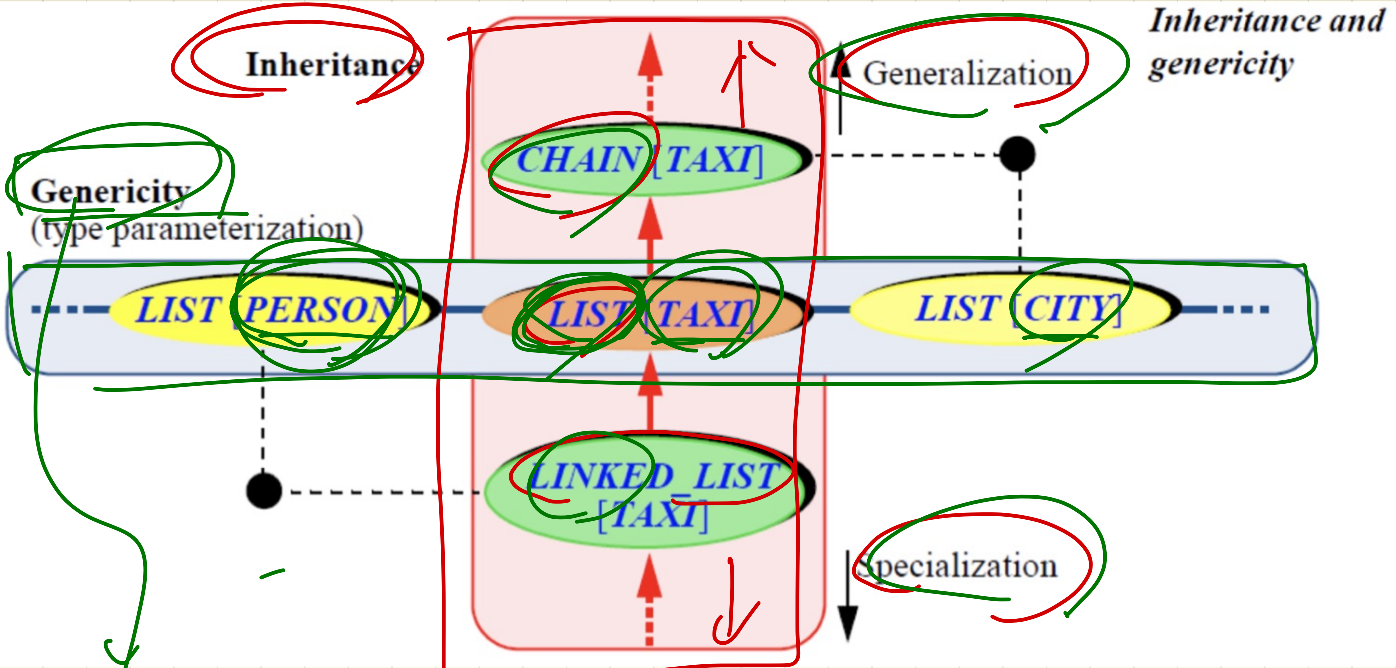
e.g., Declaring `STUDENT_BOOK[ARRAY[COURSE]]` instantiates `DICTIONARY[ARRAY[COURSE], STRING]`.

Generics vs. Inheritance (1)

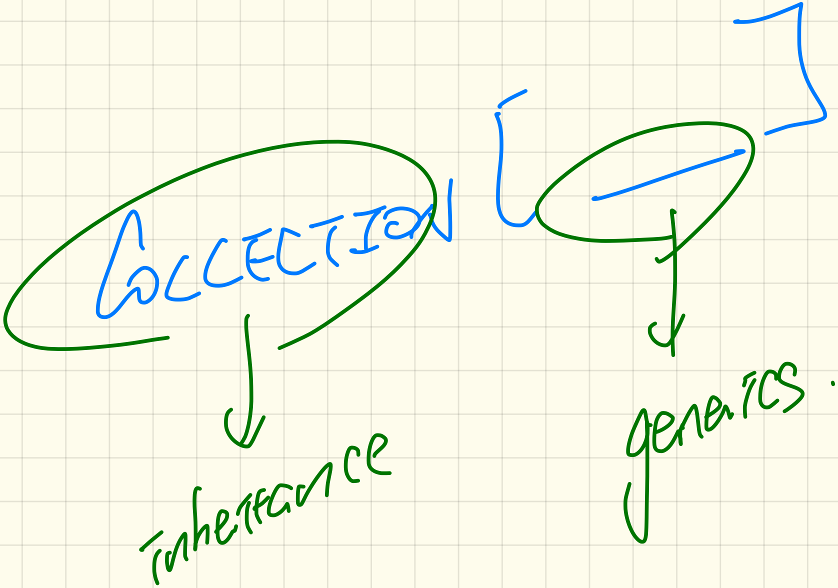


Generics vs. Inheritance (2)

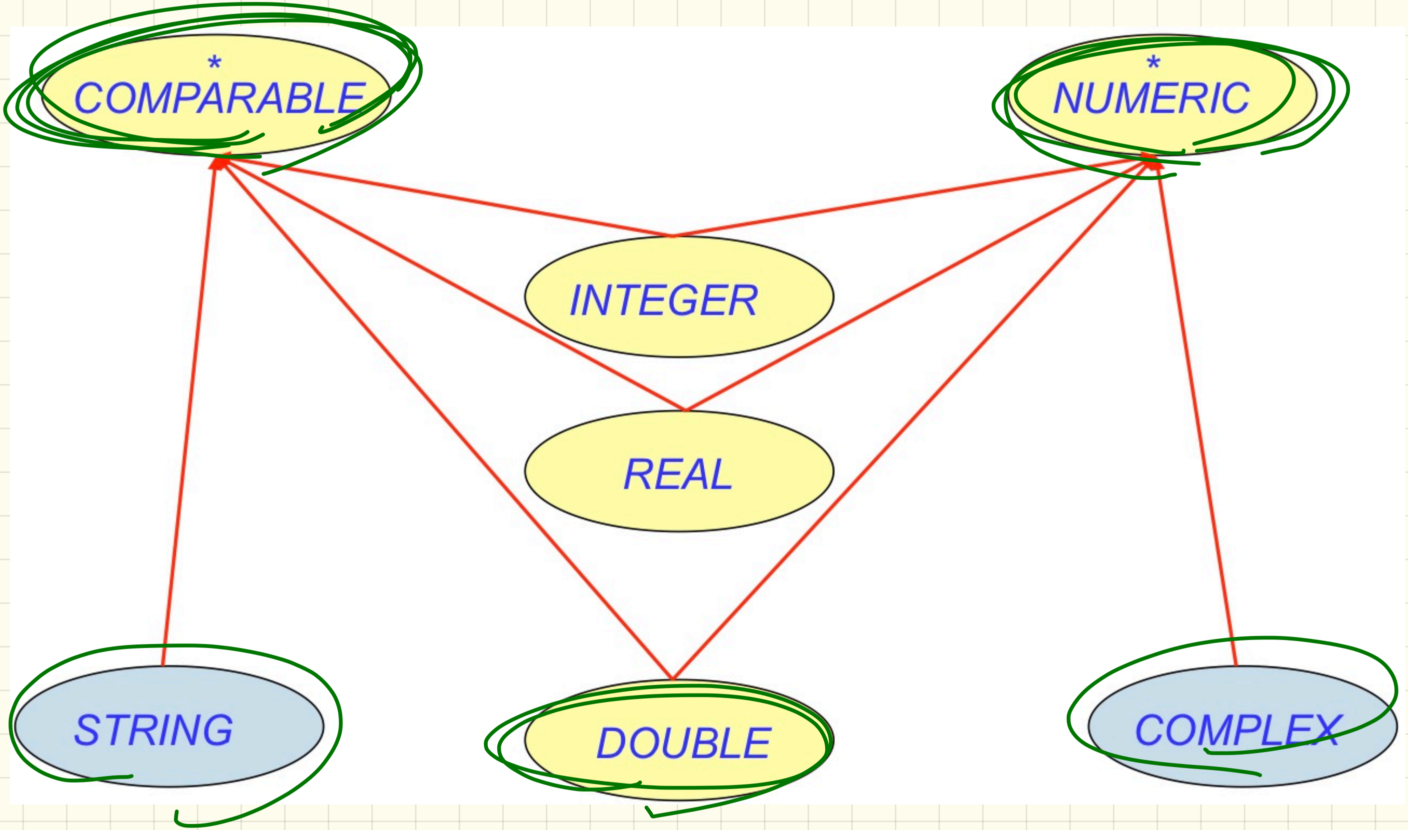
class LIST[G]

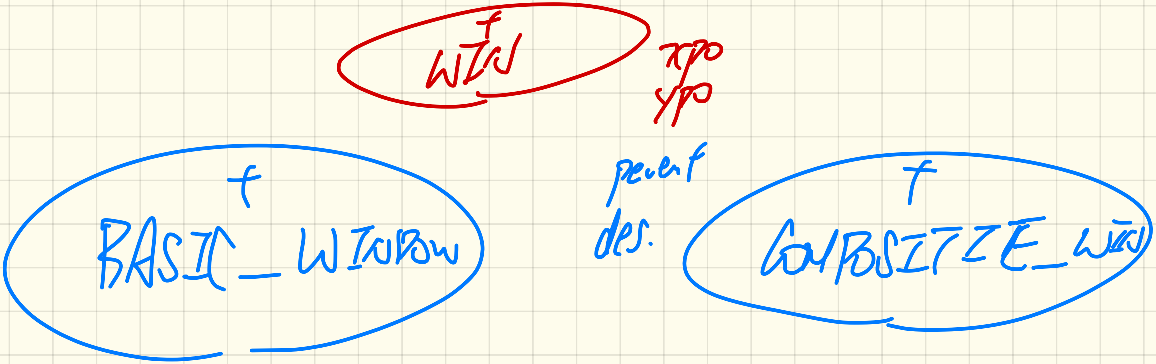


parameterize the type of members in a collection


COLLECTION
inheritance
generics.

Multiple Inheritance: Example




f
WIN
xpo
ypo
parent
des.
f
BASIC_WINDOW
f
COMPOSITE_WIN

Multiple Inheritance: Exercise

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

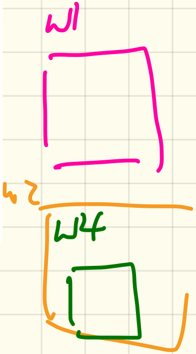
TREE[G → WINDOW]

```
class TREE[G]
  feature -- Queries
    descendants: ITERABLE[G]
  feature -- Commands
    add (c: G)
        -- Add a child 'c'.
end
```

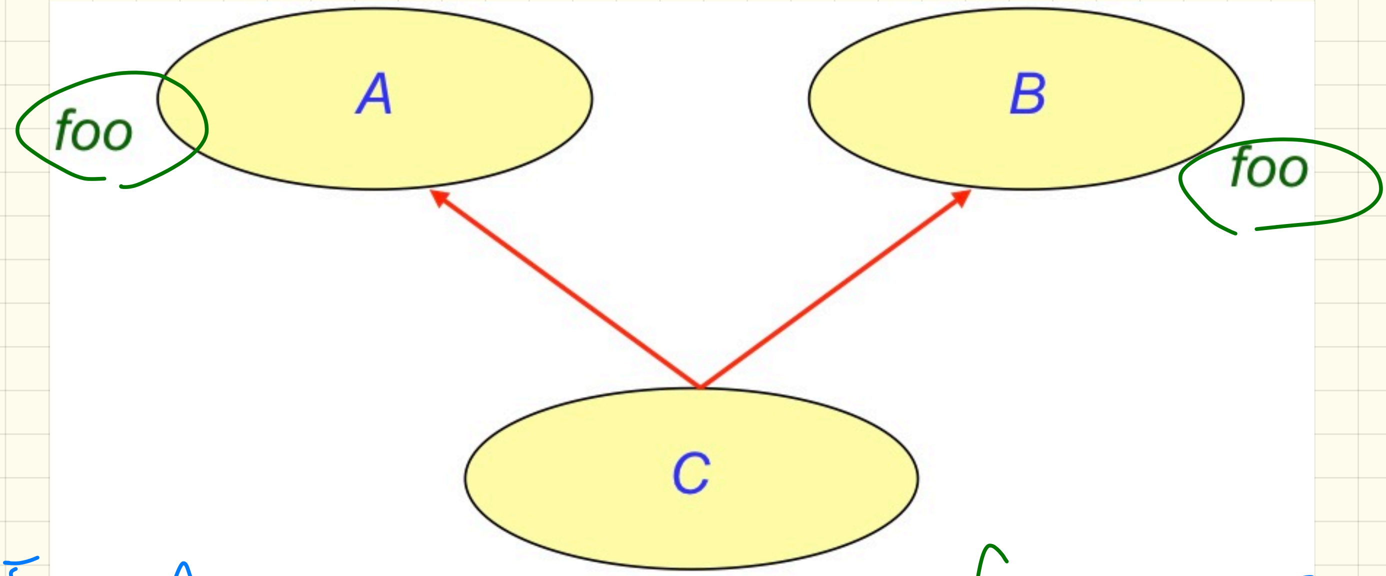
orthogonal

```
class WINDOW
  inherit
    RECTANGLE
    TREE[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



```
class C
  inherit A rename foo as fog
          B
```

C : C

C.foo → B

C.fog → foo from A

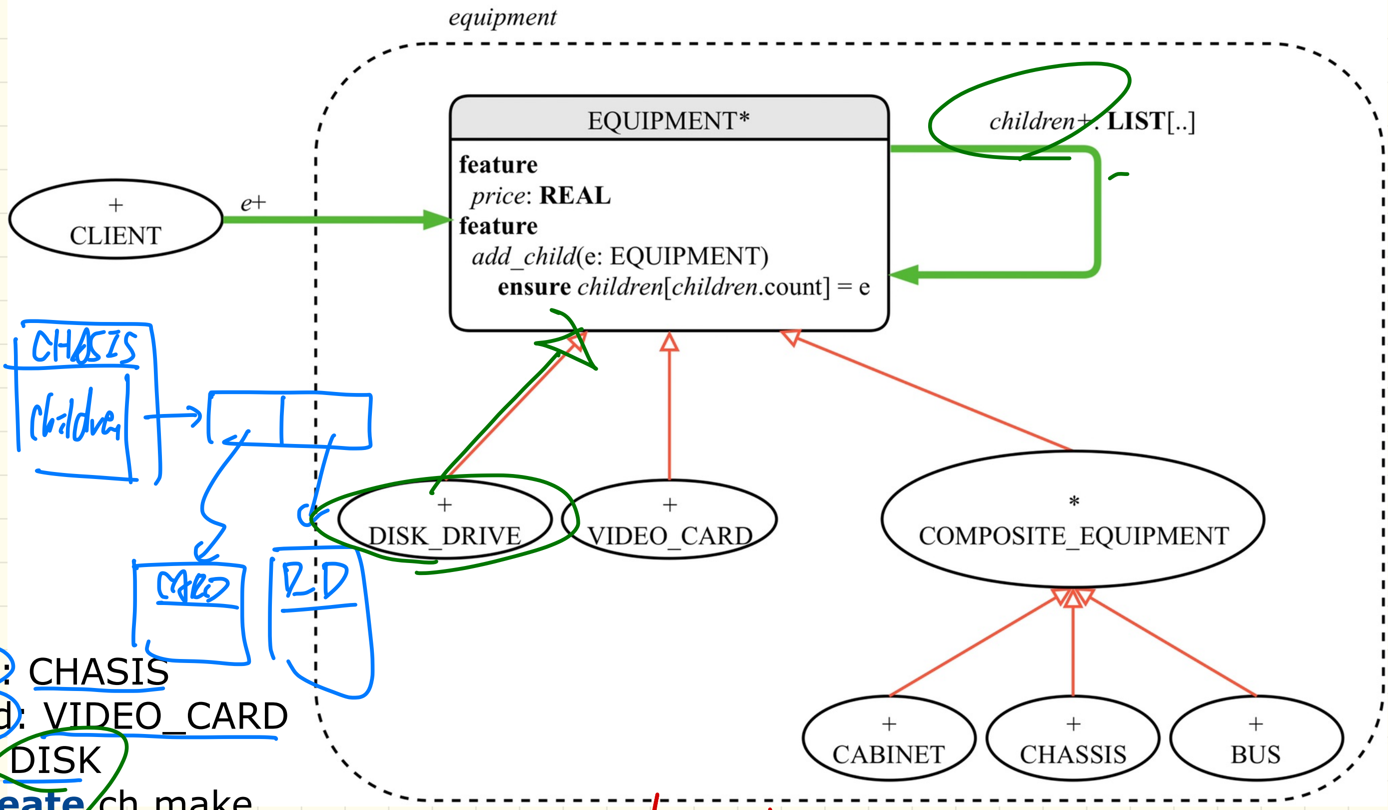
Overloading.

Java

```
deposit(String id, int amount)
deposit(int amount)
```

Eiffel

```
deposit_1(____, ____)
deposit_2(____)
```



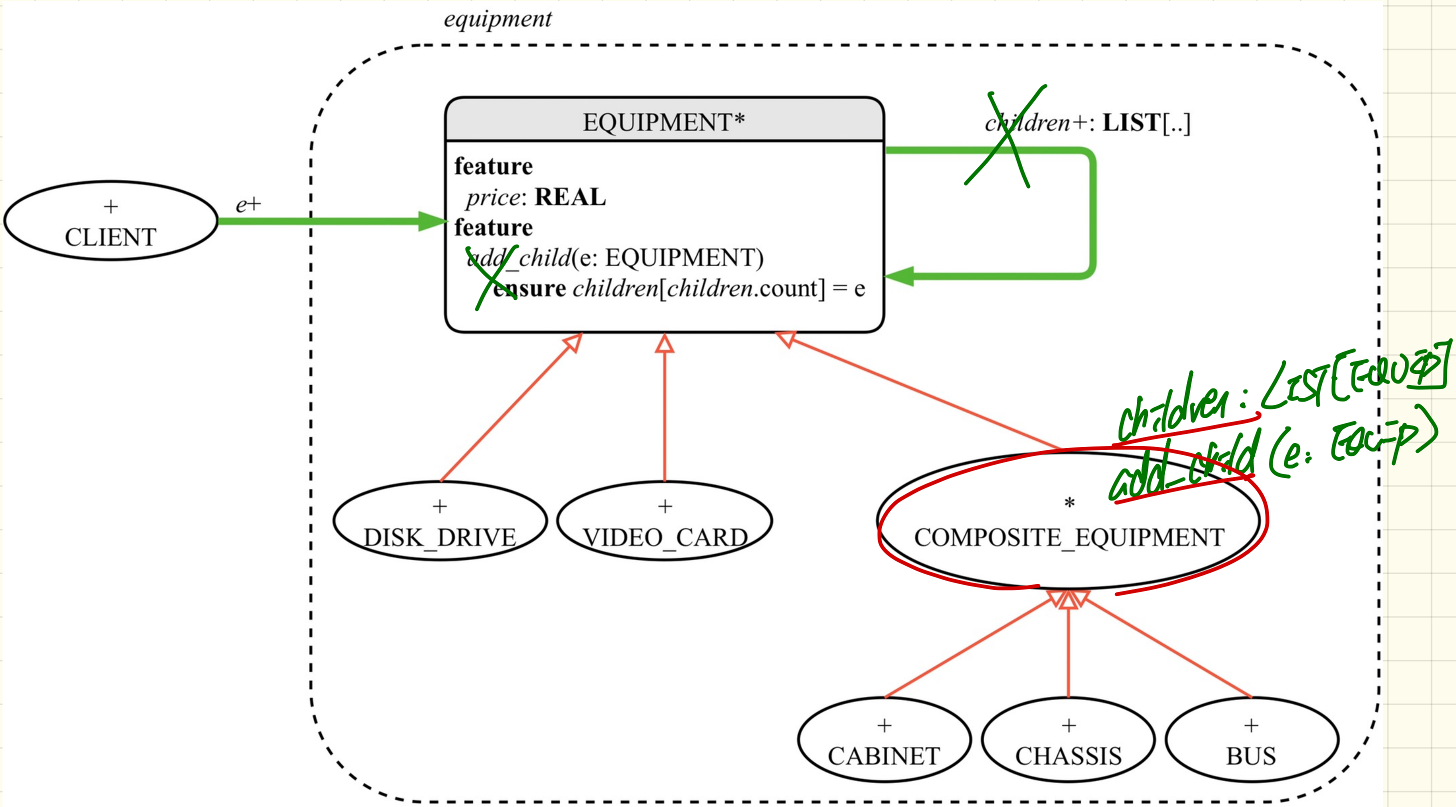
```
ch: CHASIS
crd: VIDEO_CARD
d: DISK
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)
```

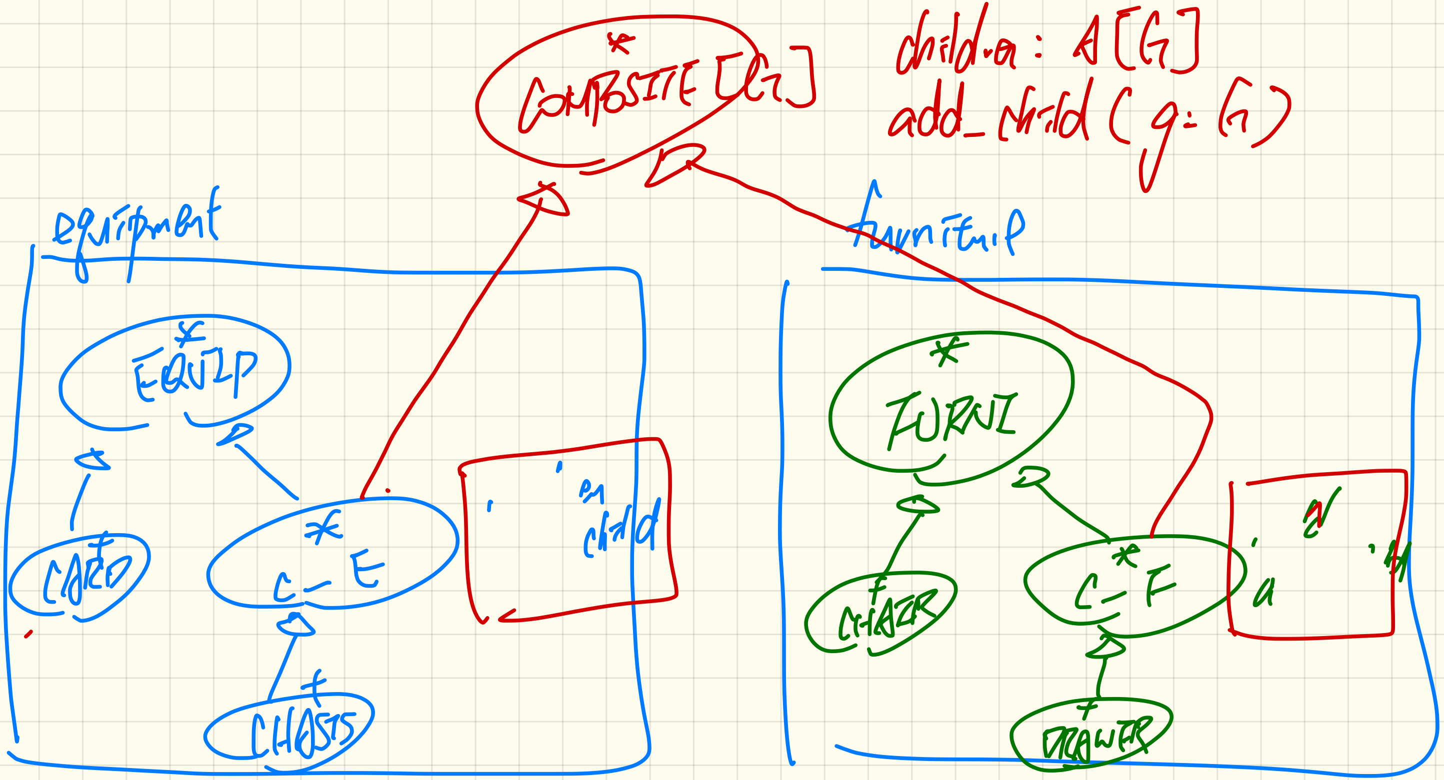
d.add_child(crd)

ST: DISK

defined in EQUIP.

First Design Attempt


equipment
+ CLIENT
e+
EQUIPMENT*
feature
price: REAL
feature
~~add~~_child(e: EQUIPMENT)
~~ensure~~ children[children.count] = e
~~children~~+: LIST[..]
+ DISK_DRIVE
+ VIDEO_CARD
* COMPOSITE_EQUIPMENT
children : LIST[EQUIP]
add_child (e: EQUIP)
+ CABINET
+ CHASSIS
+ BUS

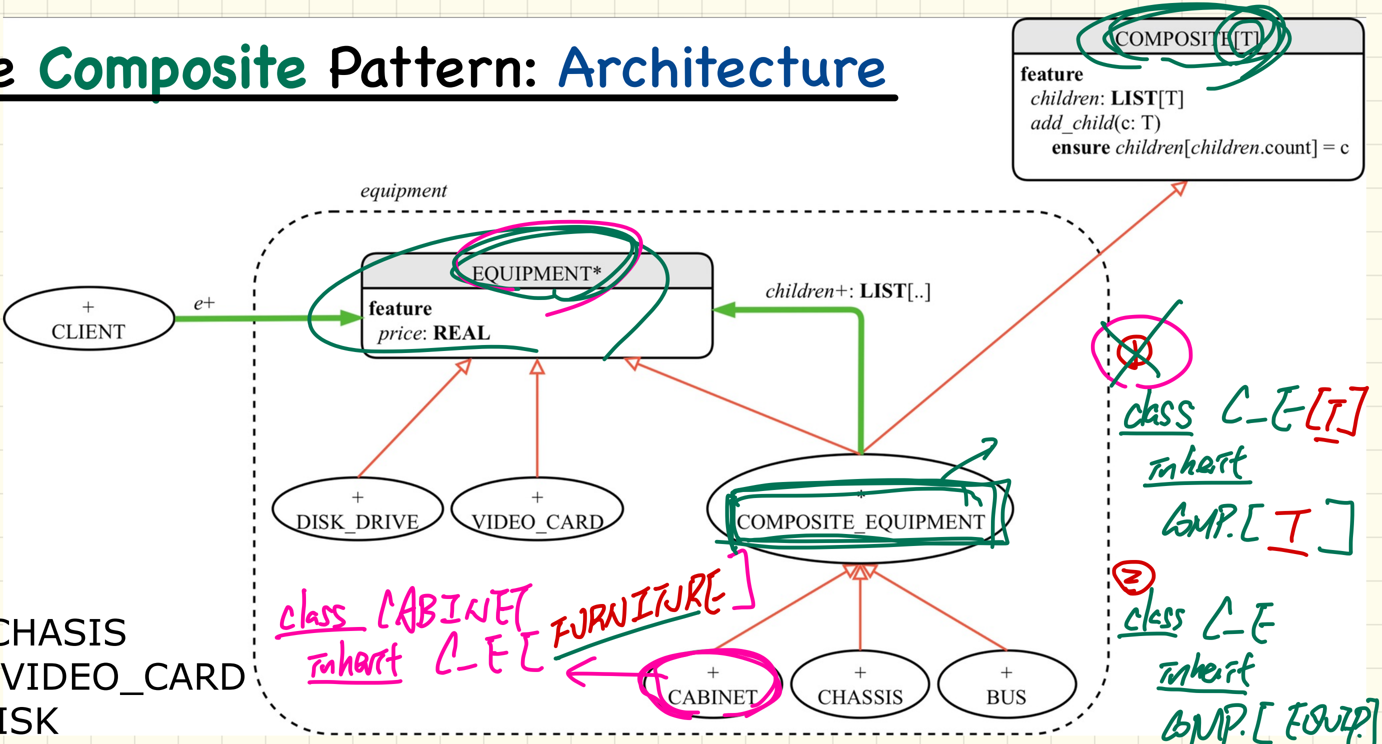


children: A[G]
add_child(g: G)

LECTURE 14

TUESDAY OCTOBER 29

The Composite Pattern: Architecture



class CABINET
inherit C_E[FURNITURE]

①
class C_E[T]
inherit
COMP.[T]

②
class C_E
inherit
COMP.[EQUIP.]

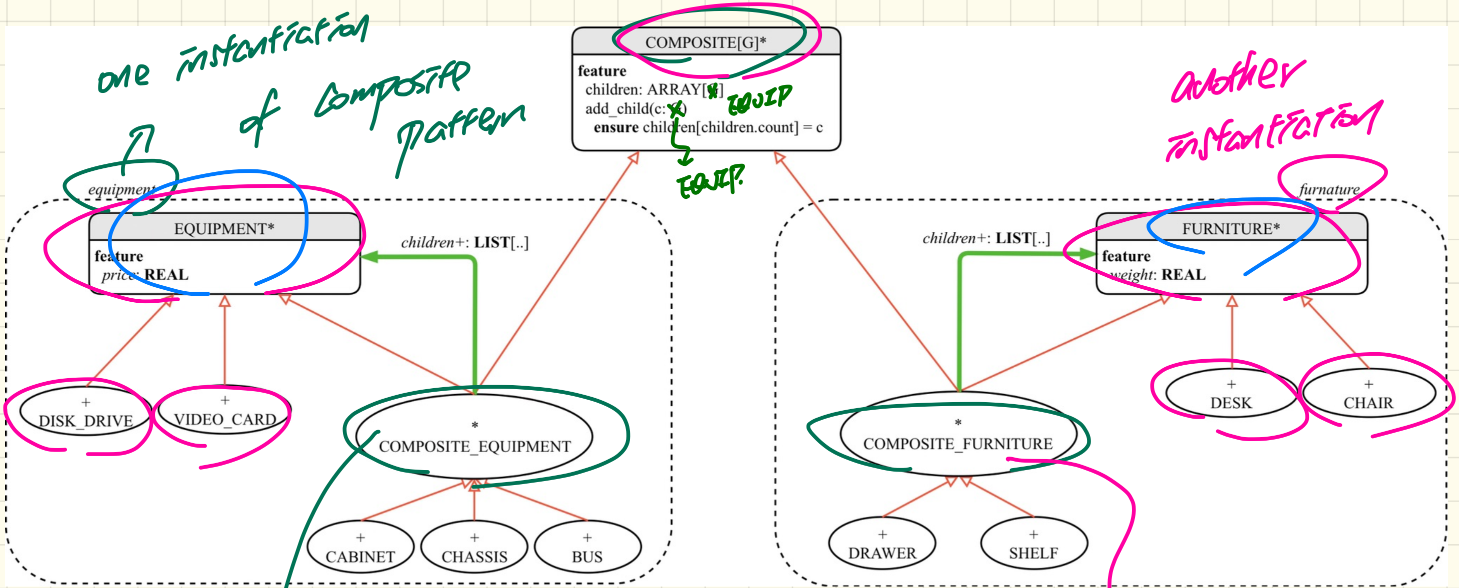
```
ch: CHASIS
crd: VIDEO_CARD
d: DISK
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)
crd.add_child(d)
```

Why is COMPOSITE a separate class?

single choice principle

The Composite Pattern: Architecture

COMPOSITE class is **reusable** by instances of the **composite** pattern.



class C_E
inherit COMP[EQUIP]

class C_F
inherit COMP[FUR]

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
```

EQUIP

EQUIP

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

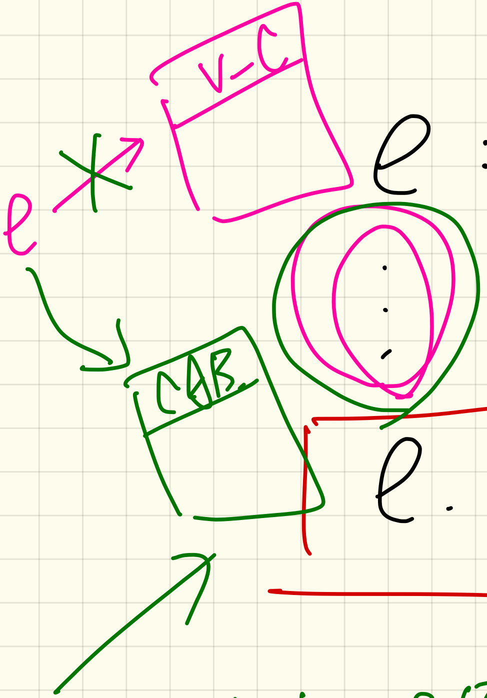
```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
    -- Sum the net prices of all sub-equipments
    do
      across
        children as cursor
      loop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

extra : REAL

price

ST?

Result := R + extra



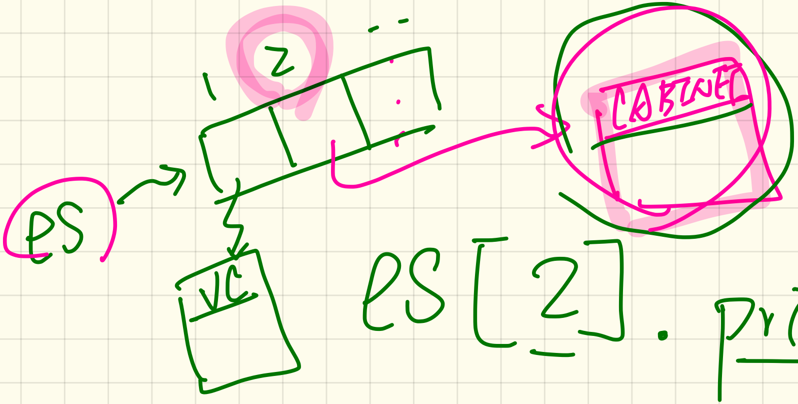
e : EQUIPMENT

e.price

which version of price will be called?

It depends on what ... does to change e's dynamic type.

es : ARRAY[EQUIPMENT]



es[2].price

Q1. compile? ✓

Q2 which version?
depends.

σ : EQUIP.

es[2].add_child(vc) X

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
```

700 -

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

①

```
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
```

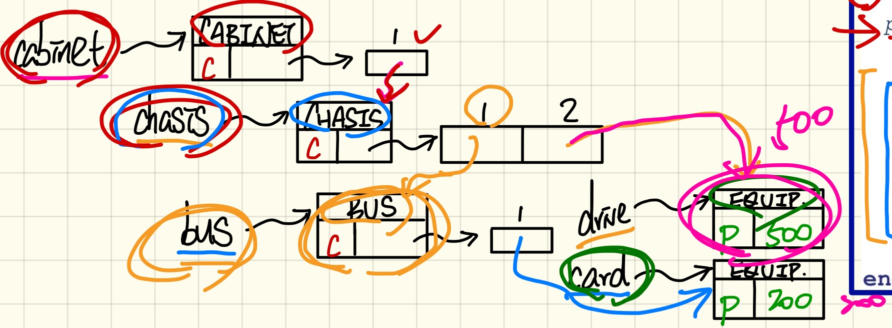
②

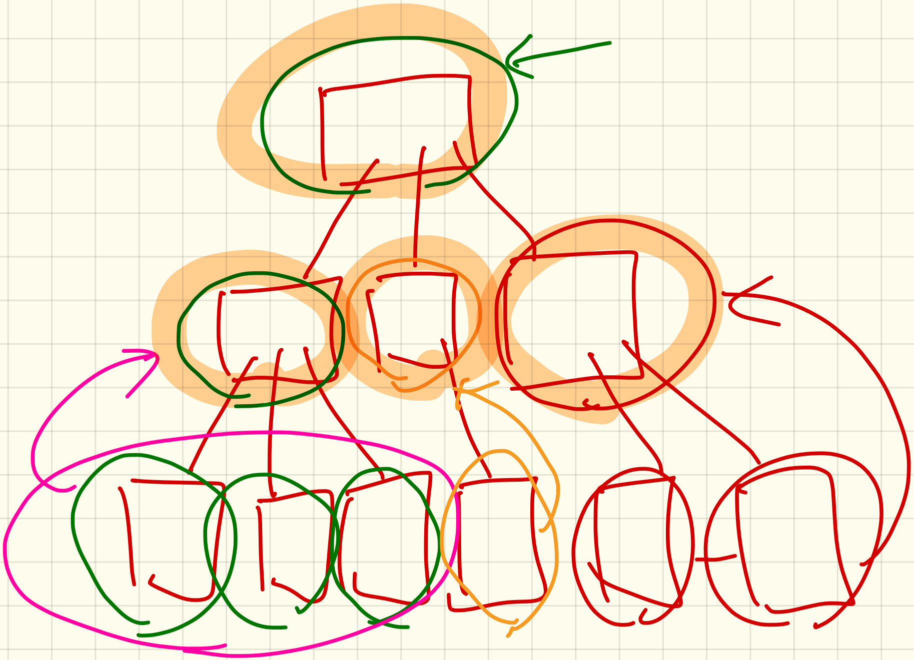
500
drive.price

200 card.price.

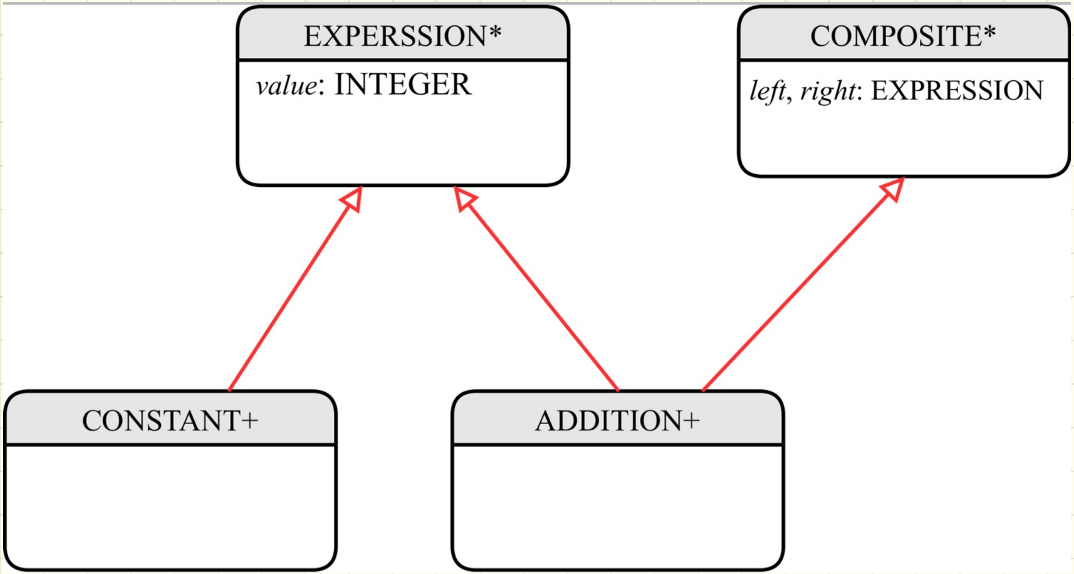
500 bus.price

chasis



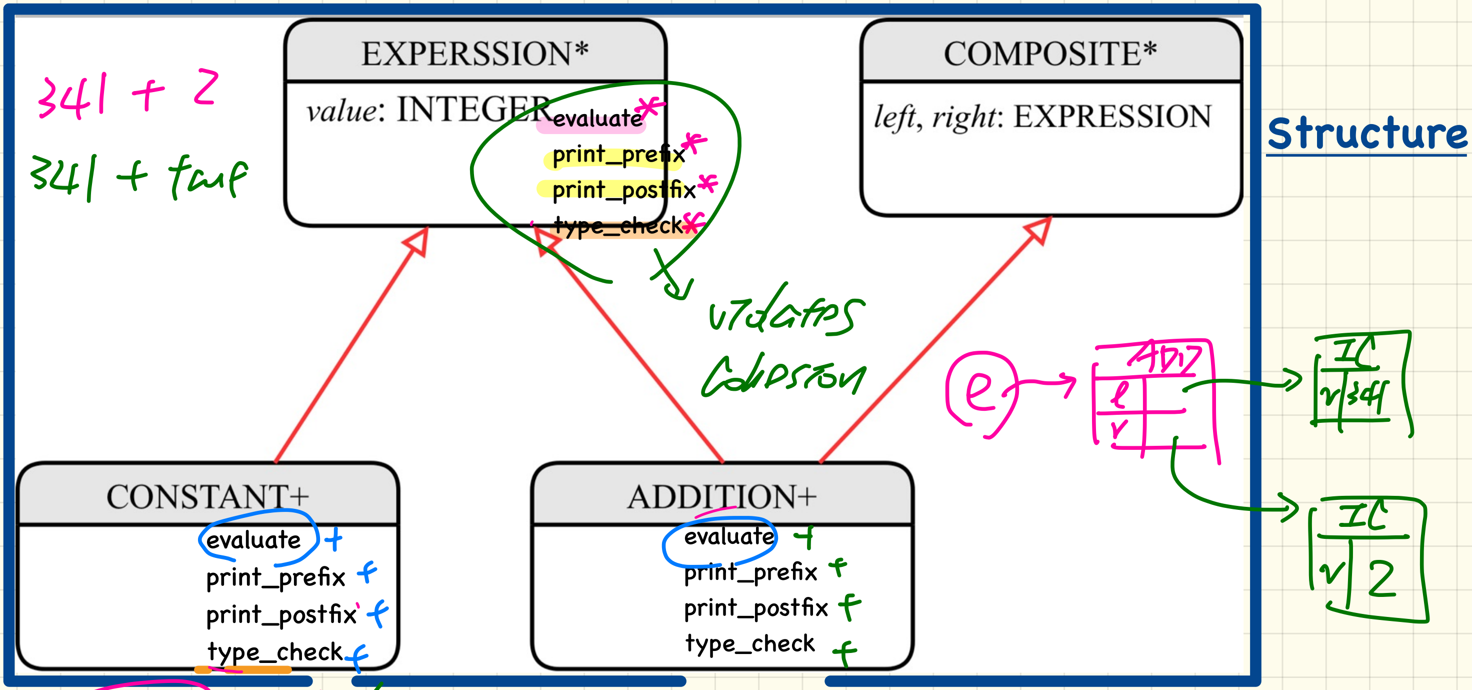


Design of Language Structure: Composite Pattern



Q: How do you construct a composite object representing "341 + 2"?

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



341 + 2

341 + true

EXPERSSION*

value: INTEGER

evaluate*

print_prefix*

print_postfix*

type_check*

violates cohesion

COMPOSITE*

left, *right*: EXPRESSION

Structure

CONSTANT+

evaluate +

print_prefix +

print_postfix +

type_check +

ADDITION+

evaluate +

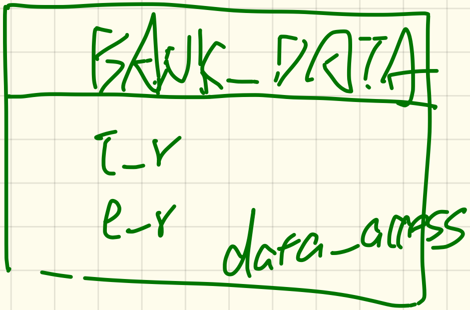
print_prefix +

print_postfix +

type_check +

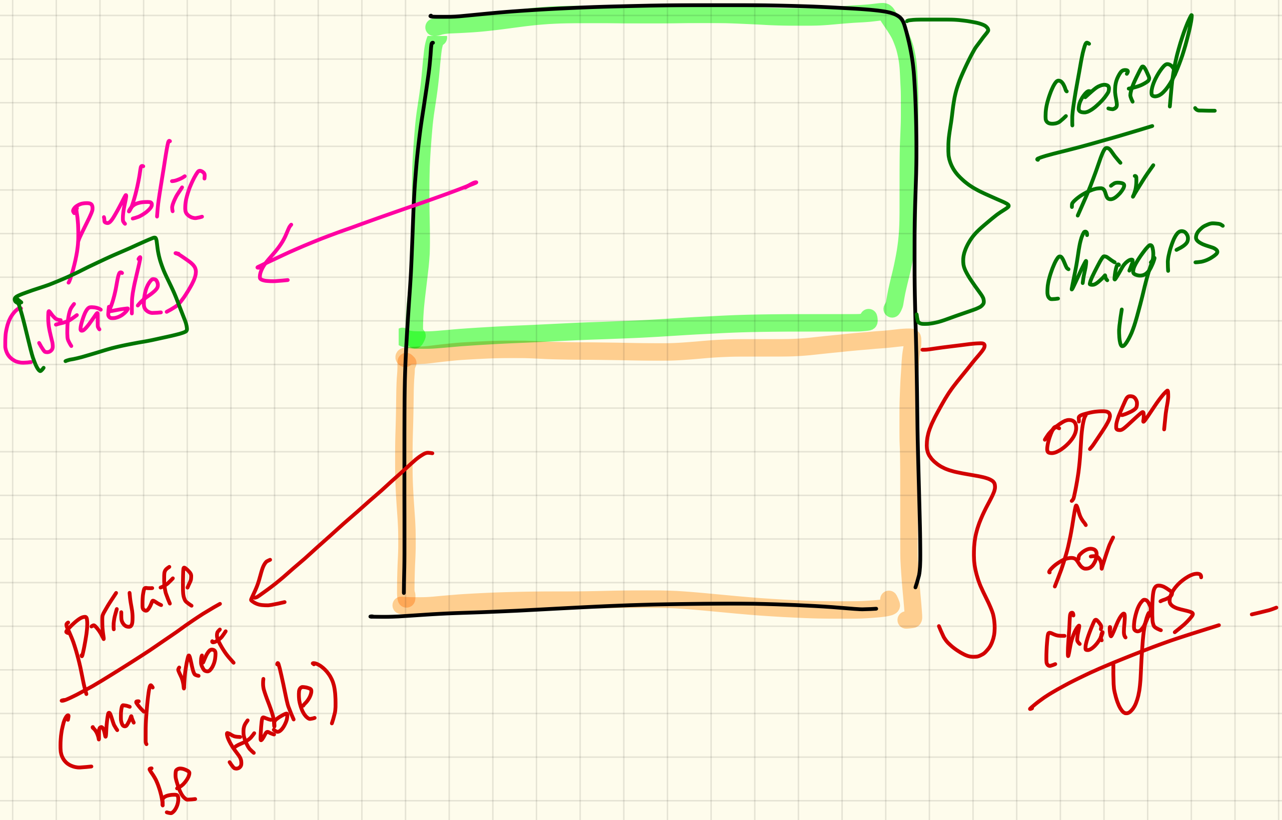
Operations

- evaluate → 343 ✓
- print_prefix → "+ 341 2" ✓
- print_postfix → "341 2 +" ✓
- type_check → Ⓣ ✓

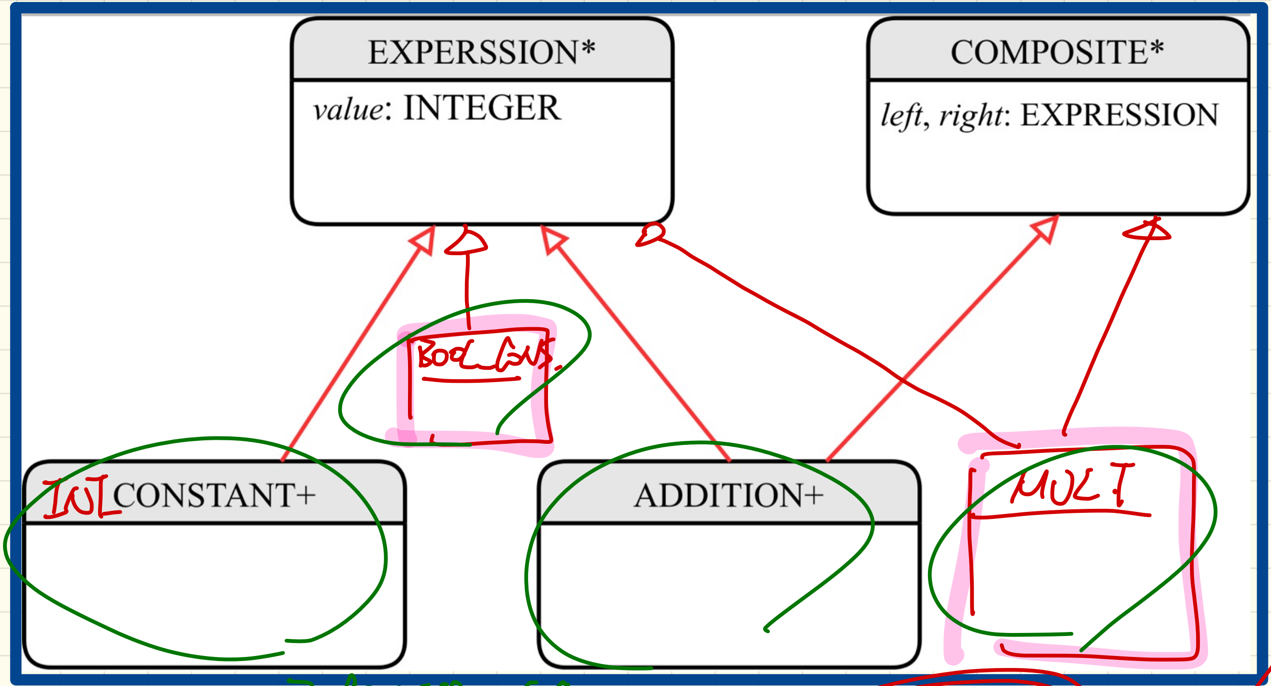


COHESION

↳ for each class only place features related to a sing purpose.


public
(stable)
closed for changes
private
(may not be stable)
open for changes

Design of a Language Application: Open-Closed Principle



Structure

- evaluate
- print_prefix
- print_postfix
- type_check

generate_java

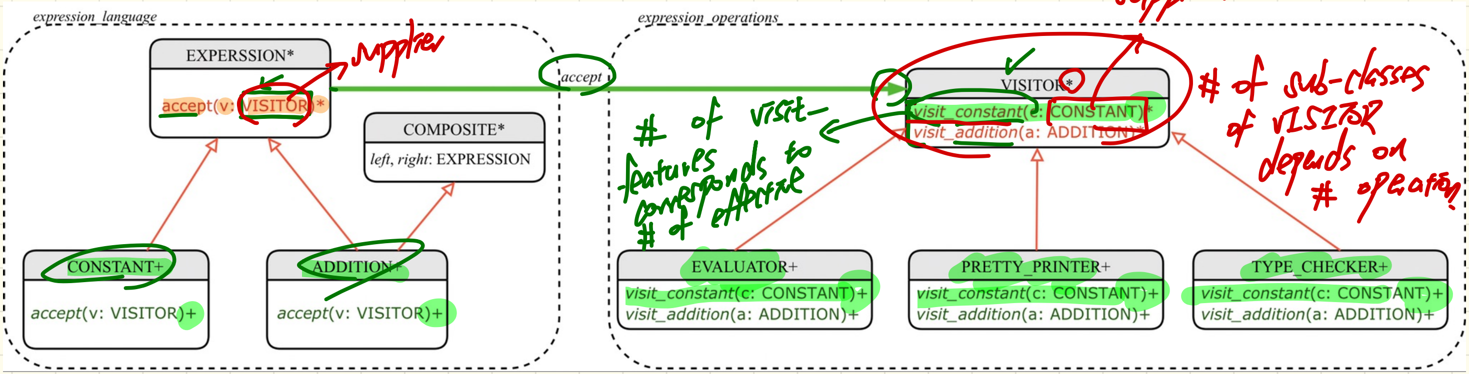
Operations

good candidate for VISITOR

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

stable

Visitor Design Pattern: Architecture



supplier

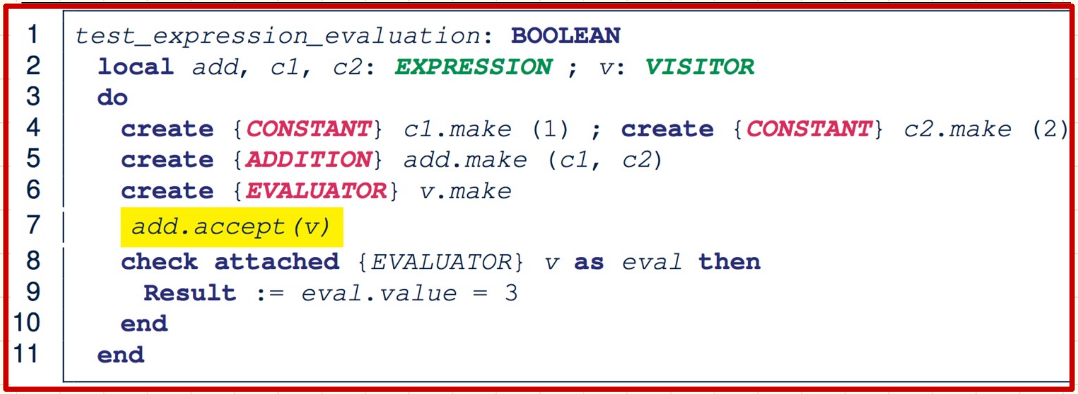
supplier

# of visit- features corresponds to # of effective

# of sub-classes of VISITOR depends on # operations

classes in composite

How to Use Visitors

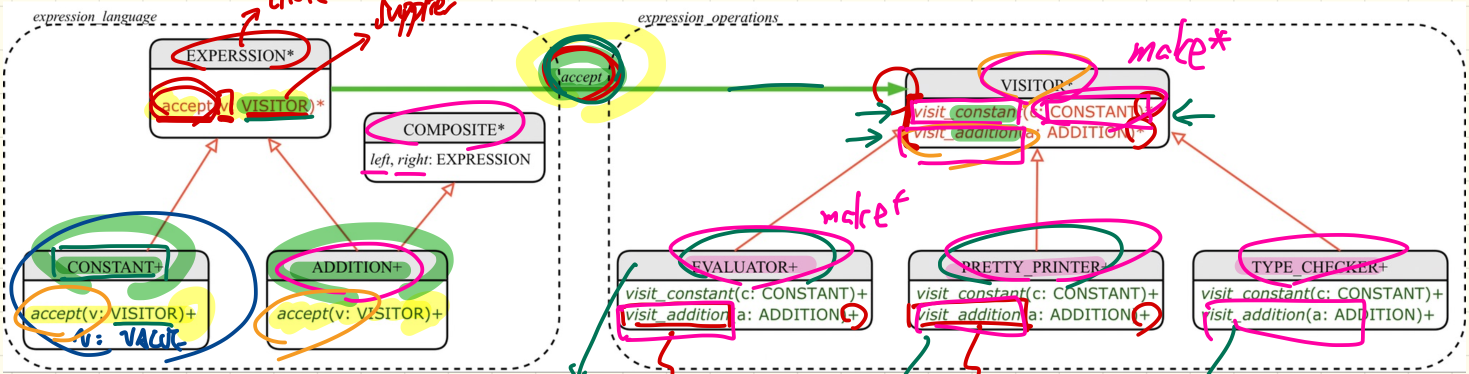


```
test_expression_evaluation: BOOLEAN
  local add, c1, c2: EXPRESSION ; v: VISITOR
  do
    create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
    create {ADDITION} add.make (c1, c2)
    create {EVALUATOR} v.make
    add.accept (v)
    check attached {EVALUATOR} v as eval then
      Result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

LECTURE 15

THURSDAY OCTOBER 31

Visitor Design Pattern: Architecture



new feature: value: INT

for evaluation

print_result: STRING

printing.

type_correct: BOOL.

How to Use Visitors

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

v: EVALUATOR

add. accept (v)

root of expression tree

operation to perform

tree of Exp.

ST: VISITOR

DT: EVALUATOR v.value ✓

v.value ✗

```
class ADDITION
  inherit EXPRESSION
    accept (v: VISITOR)
      do
        v.visit_constant (Current)
      end
```

expecting: CONSTANT

ADDITION

X

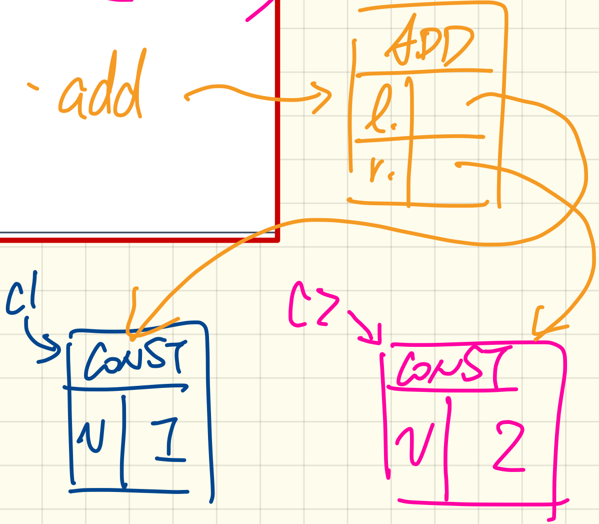
add.accept (v)

→ v.visit_add (add)

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
```

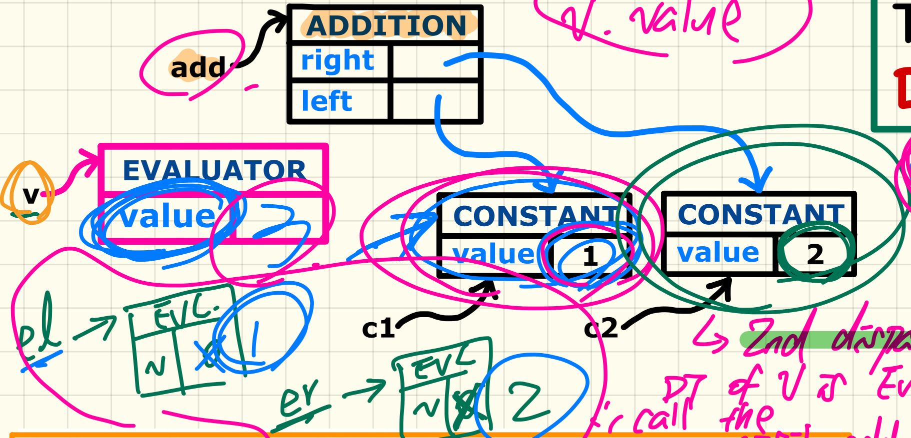
Visualizing Line 4 to Line ~~6~~ 5



Executing **Composite** and **Visitor** Patterns at **Runtime**

v. value

Tracing add.accept(v)
Double Dispatch



add. accept (v)

↳ 1st dispatch: DT of add is ADDI. ∴ call accept from ADDI.

↳ 2nd dispatch: DT of v is EVA. ∴ call the visit_add. in EVL.

```
deferred class VISITOR
  visit_constant(c: CONSTANT) deferred end
  visit_addition(a: ADDITION) deferred end
end
```

```
class CONSTANT inherit EXPRESSION
...
  accept(v: VISITOR)
    do
      v.visit_constant(Current)
    end
end
```

```
class EVALUATOR inherit VISITOR
  value : INTEGER
  visit_constant(c: CONSTANT) do value := c.value end
  visit_addition(a: ADDITION)
    local eval_left, eval_right: EVALUATOR
    do a.left.accept(eval_left)
       a.right.accept(eval_right)
       value := eval_left.value + eval_right.value
    end
end
```

add

DT?

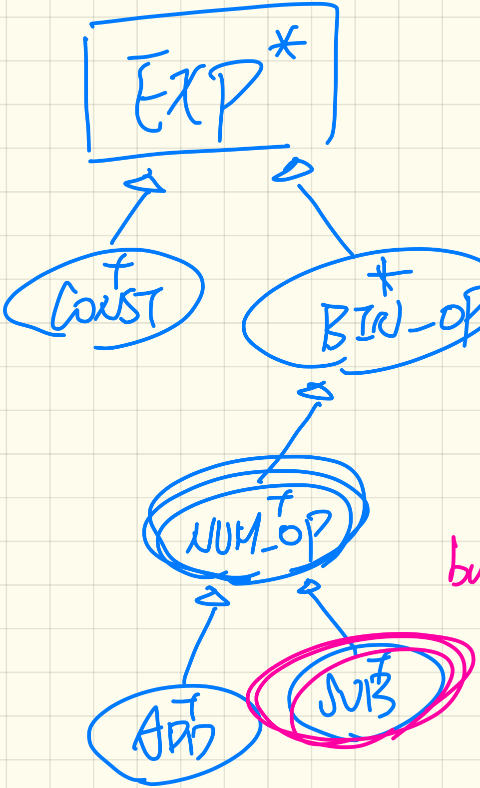
DT?

2

Exercise: Explain the D.D. here.

```
class ADDITION
inherit EXPRESSION COMPOSITE
...
  accept(v: VISITOR)
    do
      v.visit_addition(Current)
    end
end
```

add



eval : EVALUATOR (2-3)-(4-4)

e : EXP

create {SUB} e.make

⋮

build the tree

create eval.make

e. accept (eval)

↳ 1st : DT of e : SUB

2nd : DT of eval is EVAL

which tells which version of visit_sub feature to call.

v: VISITOR

{ }

create v.make

↳ create {VISITOR} v.make X

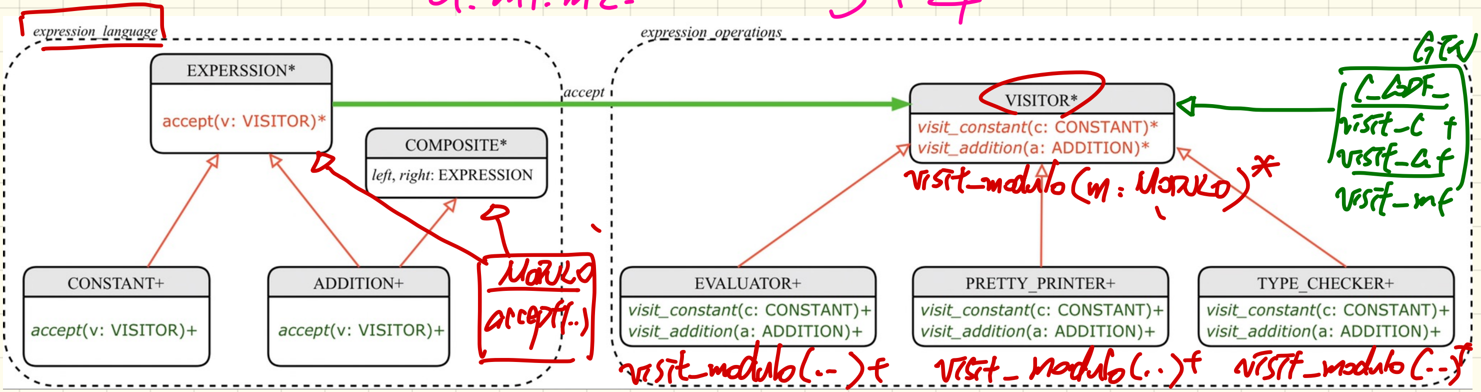
x

-

Visitor Pattern: Open-Closed and Single-Choice Principles

d. m1. m2.

3 + 4



MODULO

What if a **new language construct** is added?

What if a **new language operation** is added?

If the **visitor pattern** is adopted, what should be **open**?

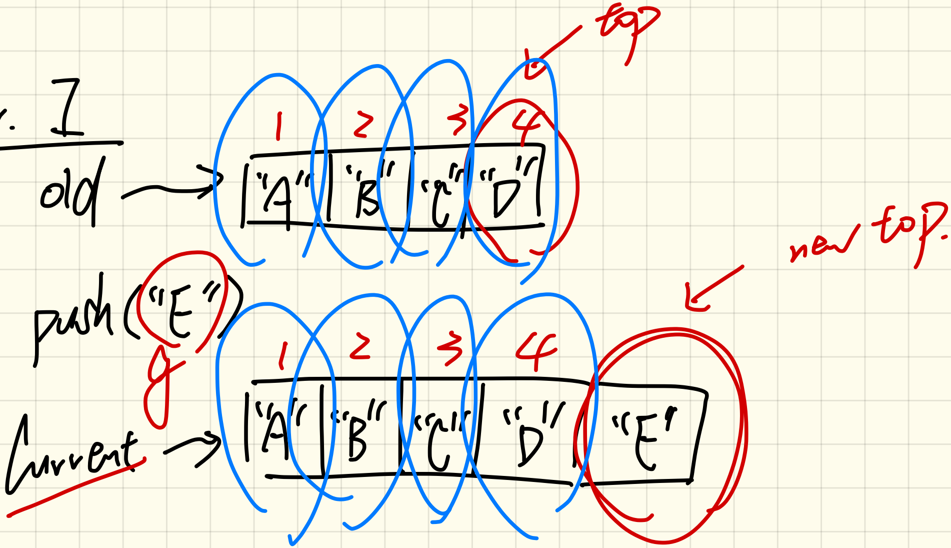
If the **visitor pattern** is adopted, what should be **closed**?

LECTURE 16

TUESDAY NOVEMBER 5

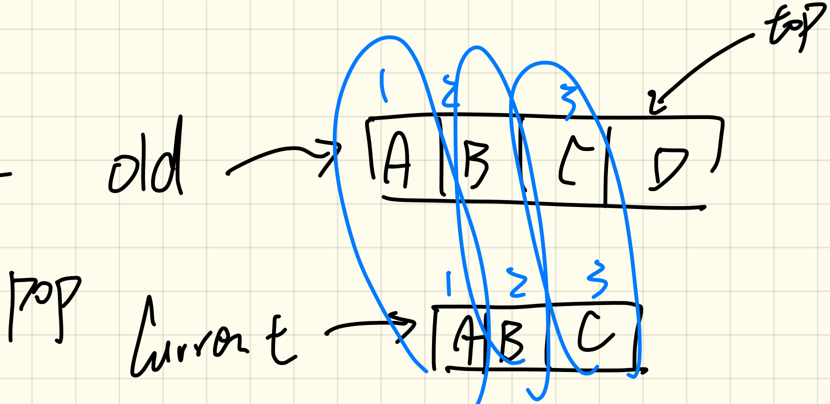
Push

Str. I



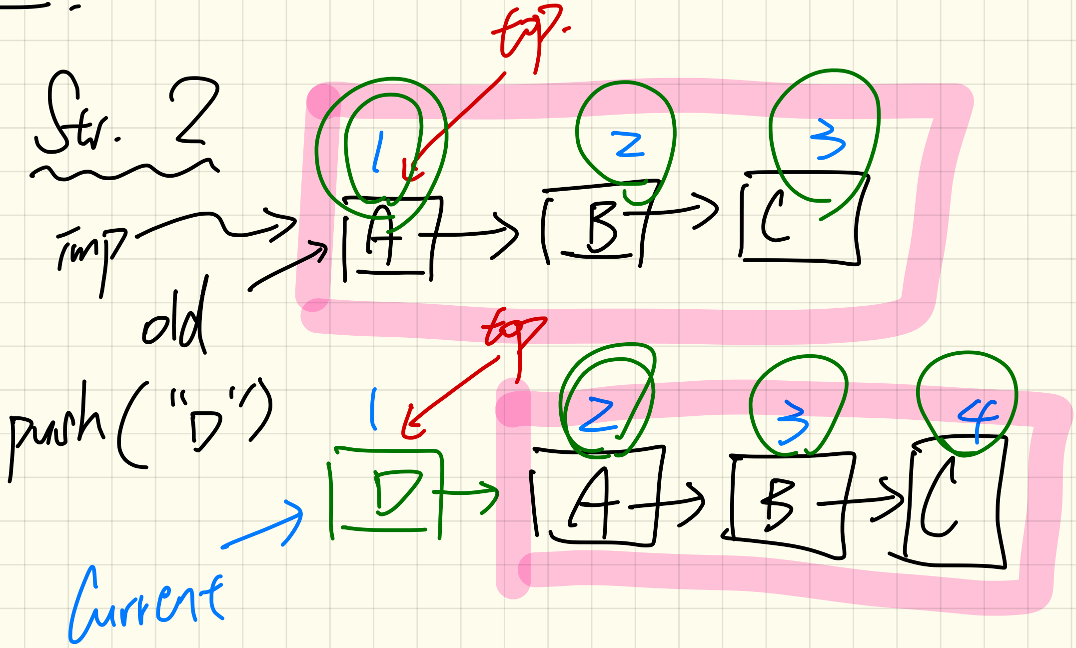
Pop

Str I



push.

Str. 2



across 2 |..| count is i

all

i

end

tmp[i] ~ (old tmp.d_t)[i-1]

Developing a LIFO Stack

SCP violated:

1. duplicates (contracts)

2. change on implementation may involve multiple changes on contracts

```
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
```

imp[1]

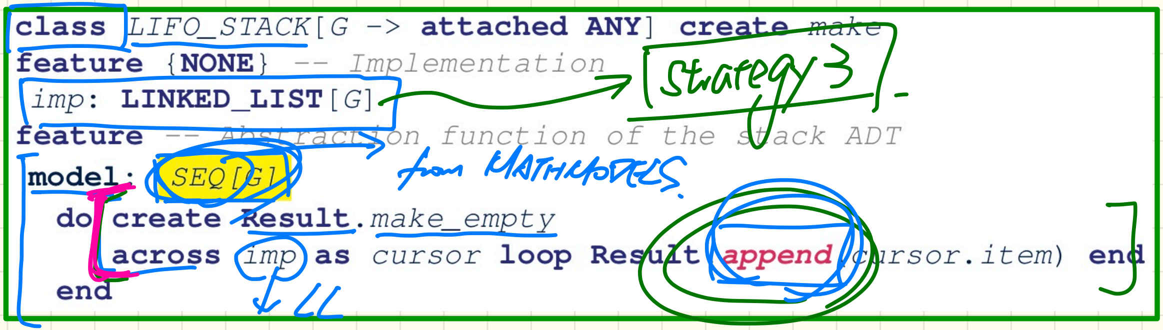
1. use a linked list where the 1st element is the top.

```
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 - 1] 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
```

```
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 an Abstraction Function



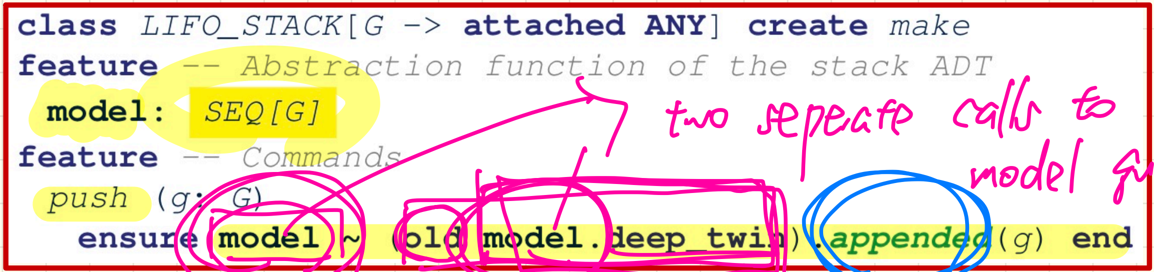
```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
       across imp as cursor loop Result.append(cursor.item) end
    end
```

strategy 3

from MATHMODELS

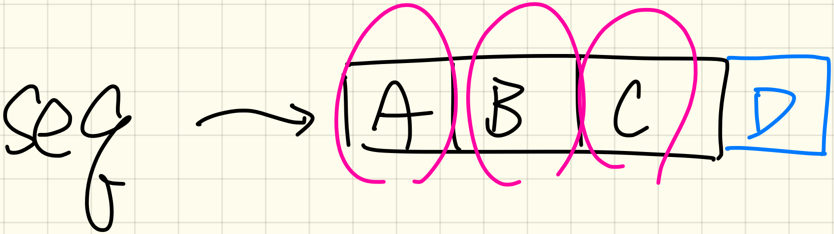
↓ LL

Writing Contracts using the 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
```

two sepeate calls to model quey

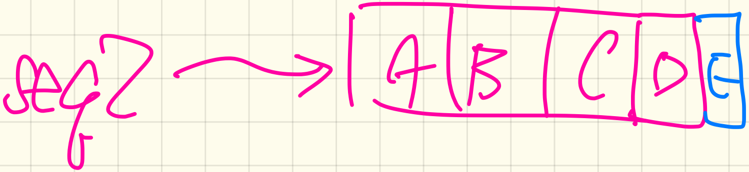


seq.append(D)

command.

seq2 := seq.appended(E)

immutable seq.



MATHMODELS

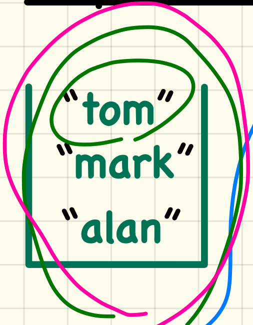
1. Commands

   ↳ use to implement the "mode" (e.g. append)

2. Immutable queries

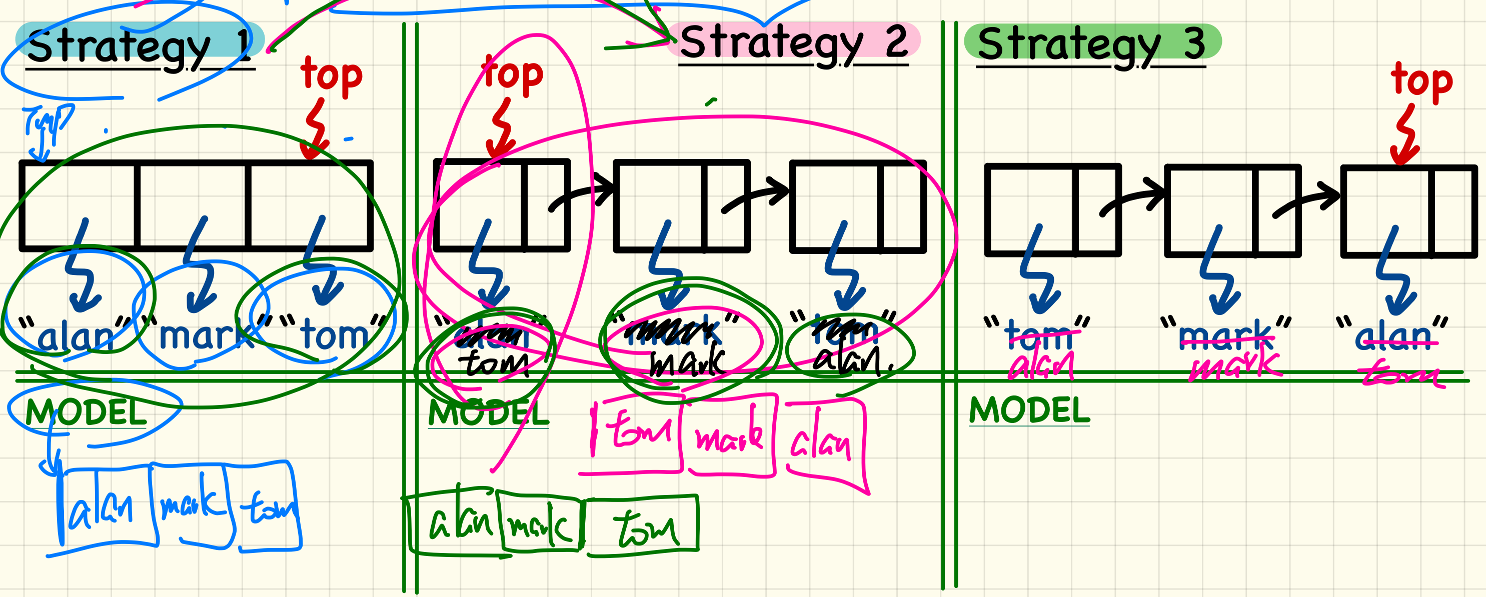
   ↳ use to write contracts (e.g. push)

Implementing a **LIFO** Stack



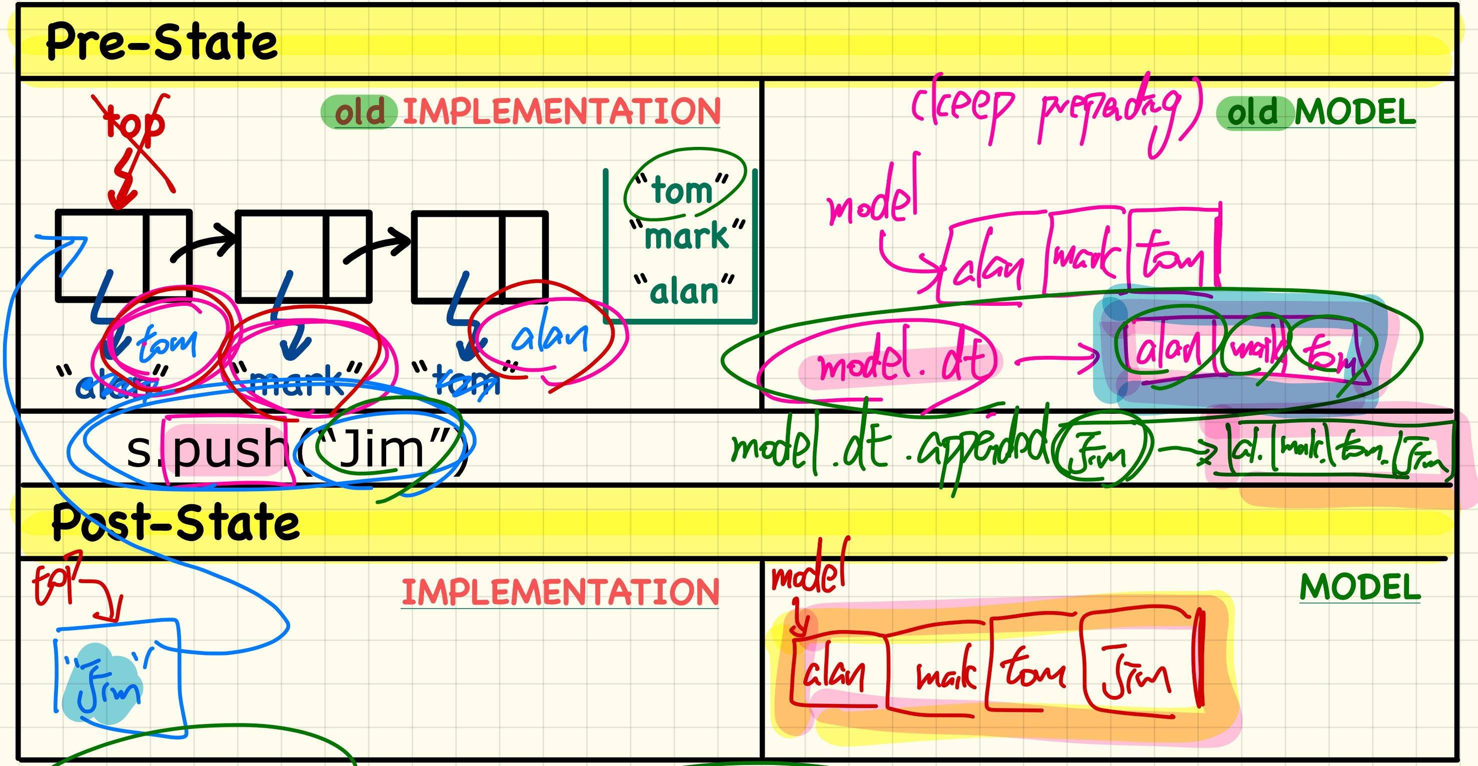
```
model : SEQ[G]
do
    across imp is g loop
        Result.append(g)
    end
end
```

```
model : SEQ[G]
do
    across imp is g loop
        Result.~~append~~ prepend (g)
    end
end
```



Checking **MATHMODELS** Contracts at **Runtime**

Strategy 2



```
push (g: G)
   ensure model ~ (old model.deep_twin).appended (g) end
```

class STACK[G]

feature {NONE}

tmp: ~~ARRAY~~[G]

LL [G]

↳ front of list is top

feature

public

model: SEQ[G]

do

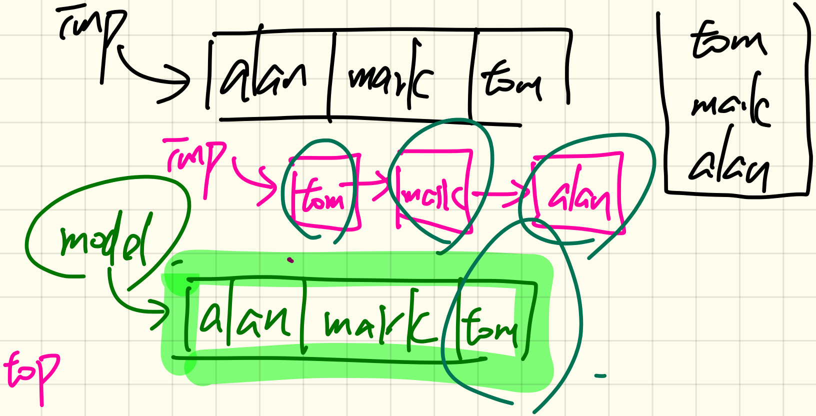
[keep ~~appending~~ prepending to the end of Result

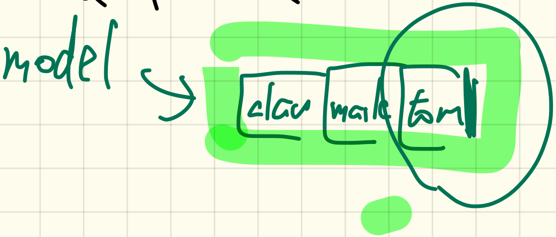
end

top: G

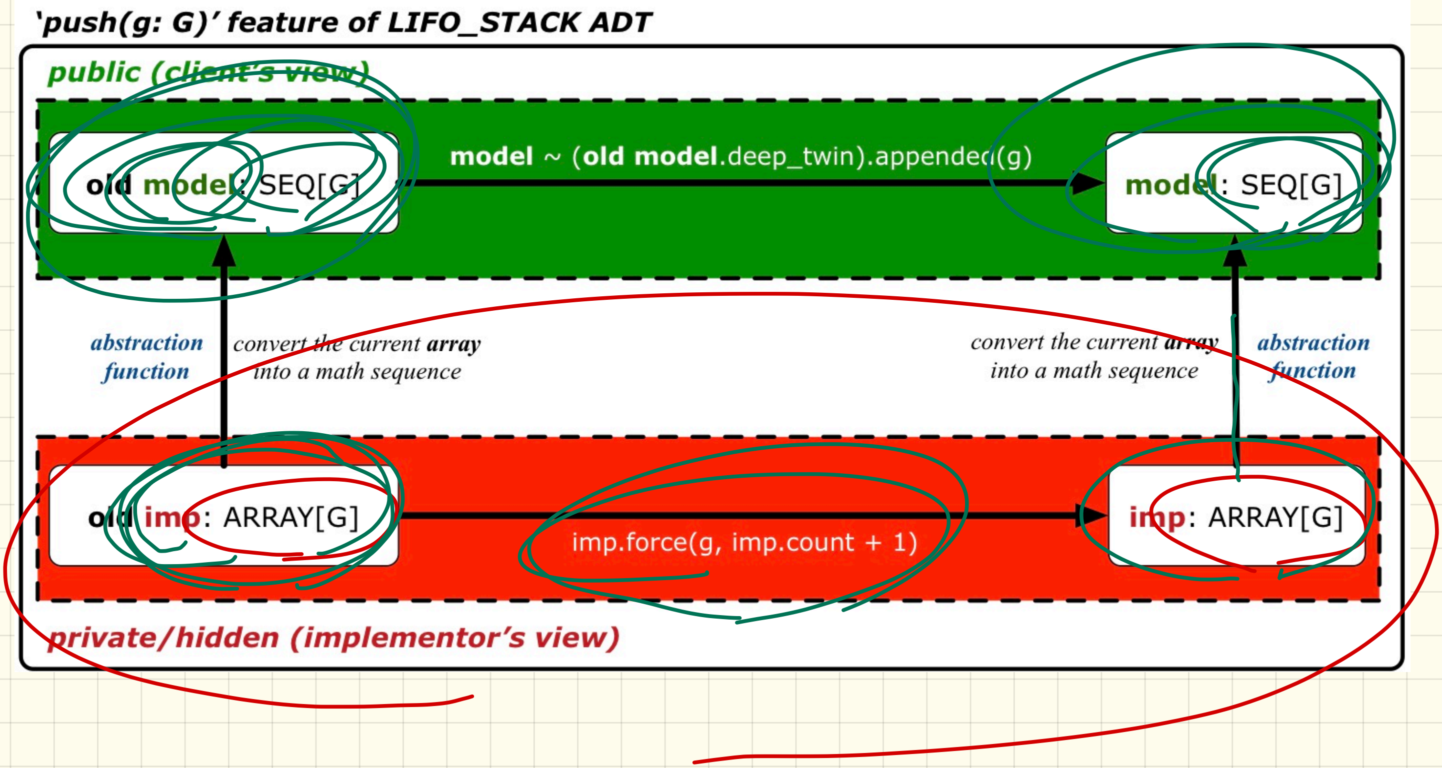
ensure

Result ~ model.last





Strategy 1: Mathematical Abstraction



'push(g: G)' feature of LIFO_STACK ADT

public (client's view)

old model: SEQ[G]

model ~ (**old model**.deep_twin).appended(g)

model: SEQ[G]

abstraction function

convert the current ***array*** *into a math sequence*

convert the current ***array*** *into a math sequence*

abstraction function

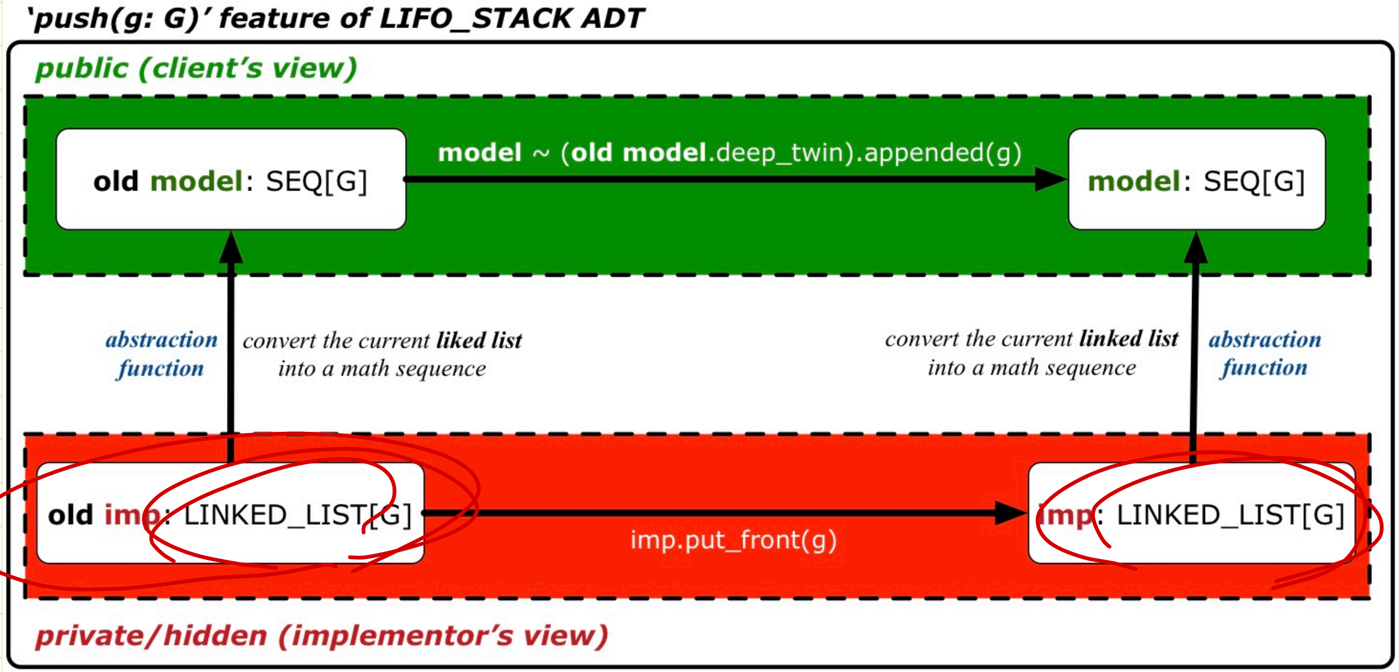
old imp: ARRAY[G]

imp.force(g, imp.count + 1)

imp: ARRAY[G]

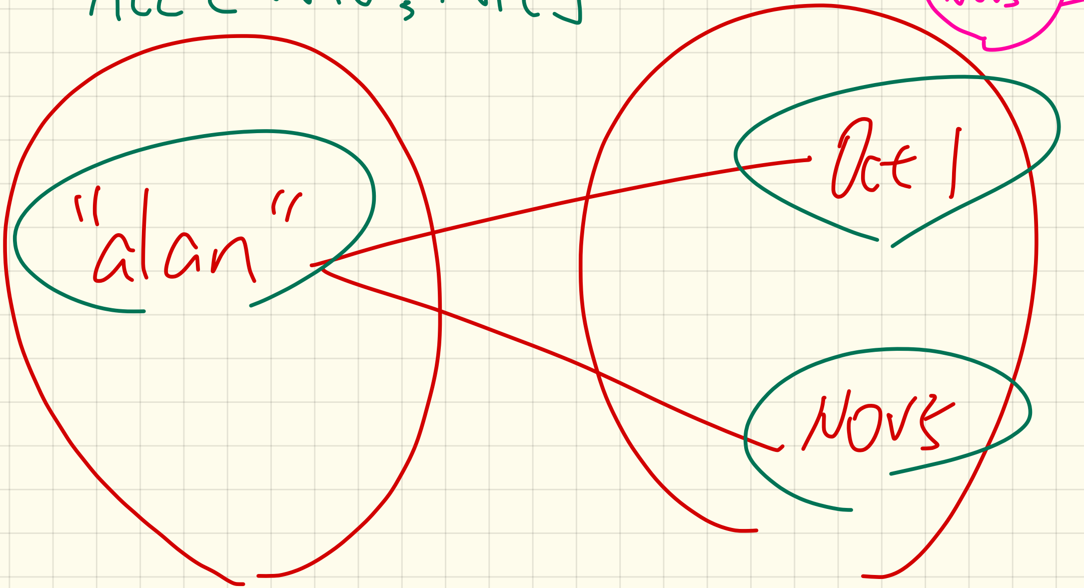
private/hidden (implementor's view)

Strategy 2: Mathematical Abstraction

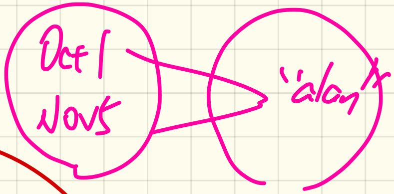


REL ?

REL [NAME, DATE]



FUN ? REL[DATE, NAME]



```
feature {NONE}
    model: SEQ[G]

feature
    get_model : SEQ [G]
```

LECTURE 17

THURSDAY NOVEMBER 7

Use of **MATHMODELS**:

Single-Choice Principle

```
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
```

```
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
```

```
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.append(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
```

Safe Use of model by Evil Clients

```
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
```

Result := Result.d_t

Client:

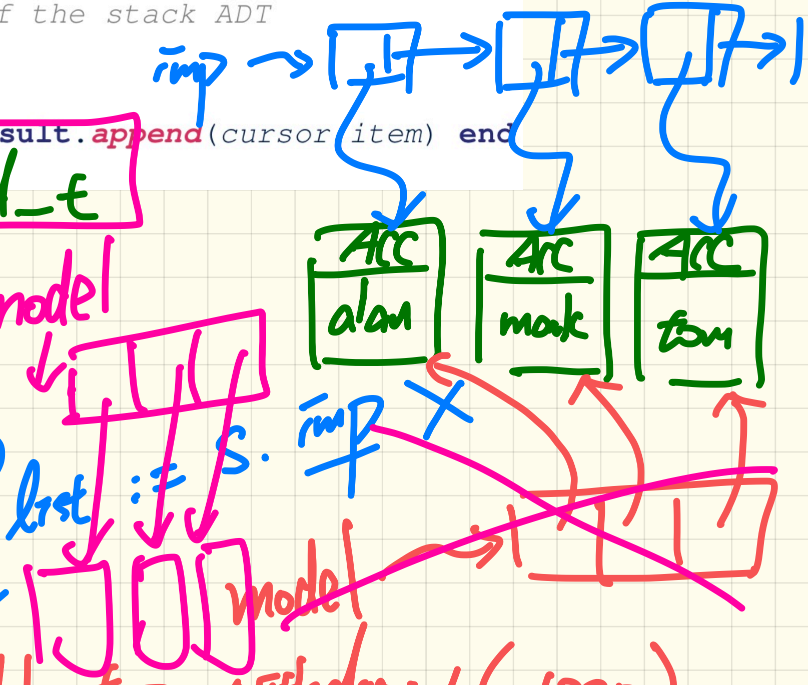
s: STACK[ACCOUNT]

s.push(alan)

s.push(mark)

s.push(tom)

① seq := s.**model**



s. model . top . withdraw (1000)

Testing **REL** in **MATHMODELS**

$$r.\textbf{overridden}(\underbrace{\{(a,3),(c,4)\}}_{t})$$

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

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

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) = $\{ (d,r) \mid (d,r) \in r \land d \in ds \}$
  - 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) = $\{ (d,r) \mid (d,r) \in r \land d \notin ds \}$
  - 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) = $\{ (d,r) \mid (d,r) \in r \land r \in rs \}$
  - 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) = $\{ (d,r) \mid (d,r) \in r \land r \notin rs \}$
  - e.g., r.**range_subtracted**({1, 2}) = $\{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

```
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
```

D

R

→ imm. query

→ command

MATHMODELS

SEQ

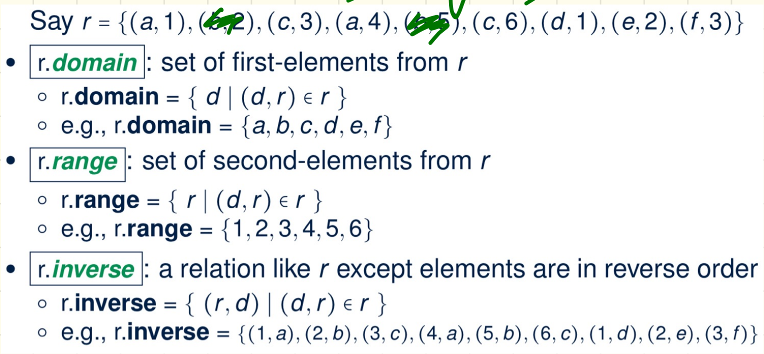
SET

REL

↑

FUN

(b,6) (g,4)



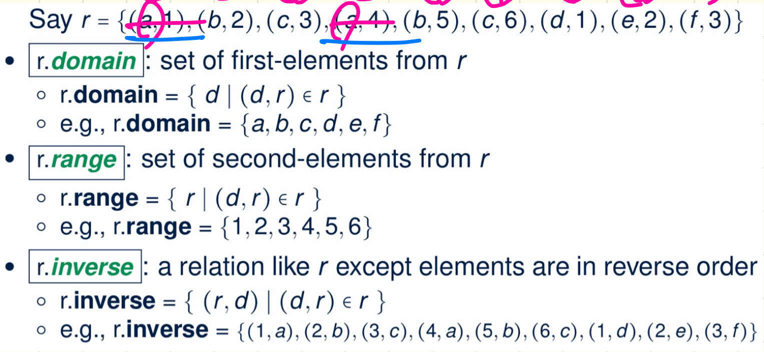
Say $r = \{(a,1), \cancel{(b,2)}, (c,3), (a,4), \cancel{(b,5)}, (c,6), (d,1), (e,2), (f,3)\}$

- **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. override({(a,3), (c,4)})

r. override({(g,4), (b,6)})

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨



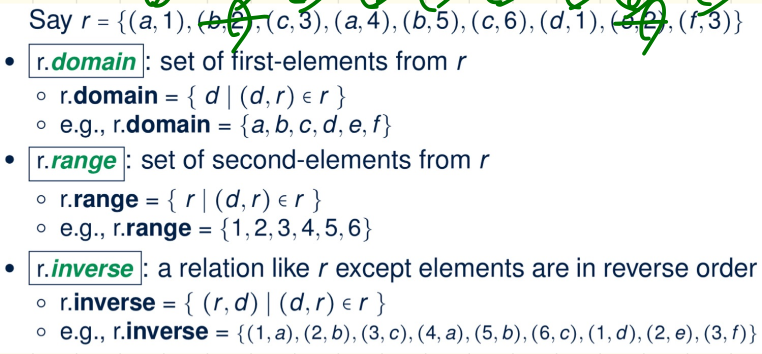
Say $r = \{\cancel{(a,1)}, (b,2), (c,3), \cancel{(a,4)}, (b,5), (c,6), (d,1), (e,2), (f,3)\}$

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

$\underline{\text{r. domain\_subtract}}(\{\underline{a}\})$

$\text{r. domain\_restrict}(\{a\}) = \{(a,1), (a,4)\}$

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨



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

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

$\text{r. range\_subtract}(\{\underline{2}\})$

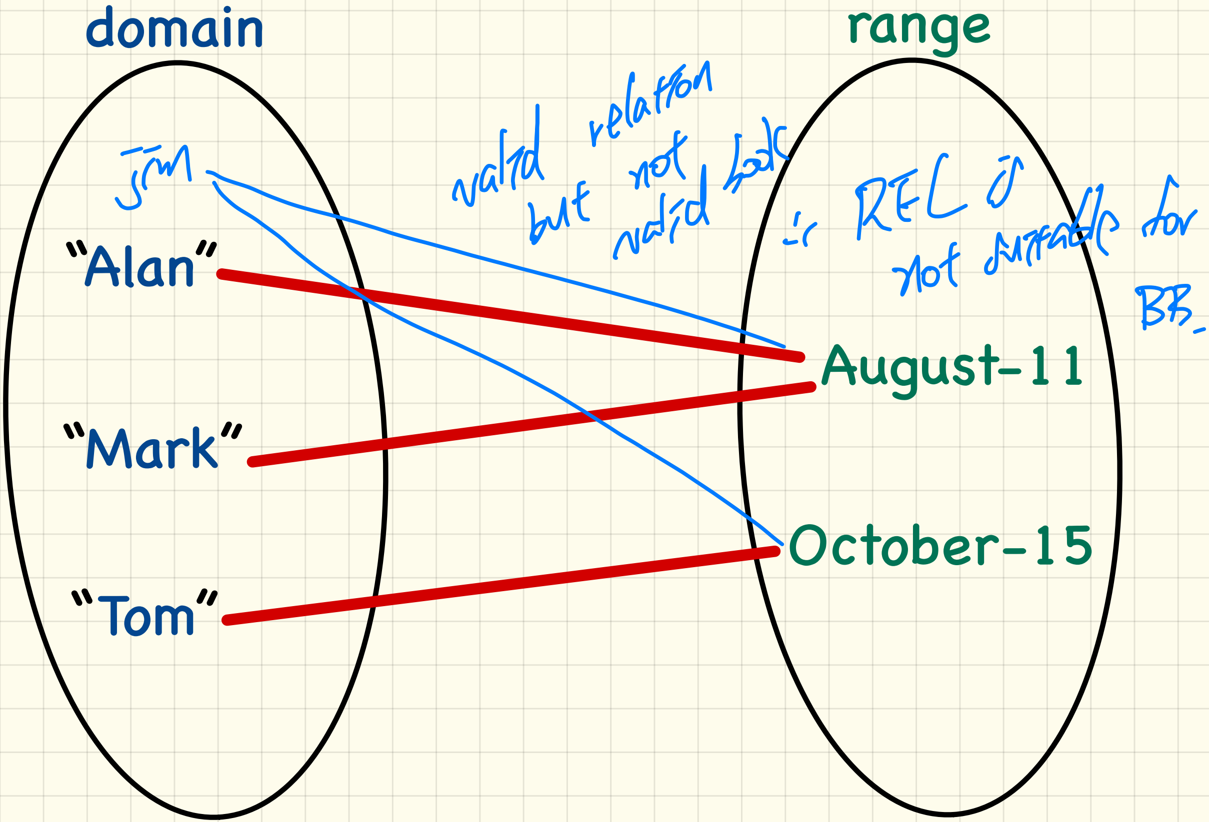
r. override(..)

Command

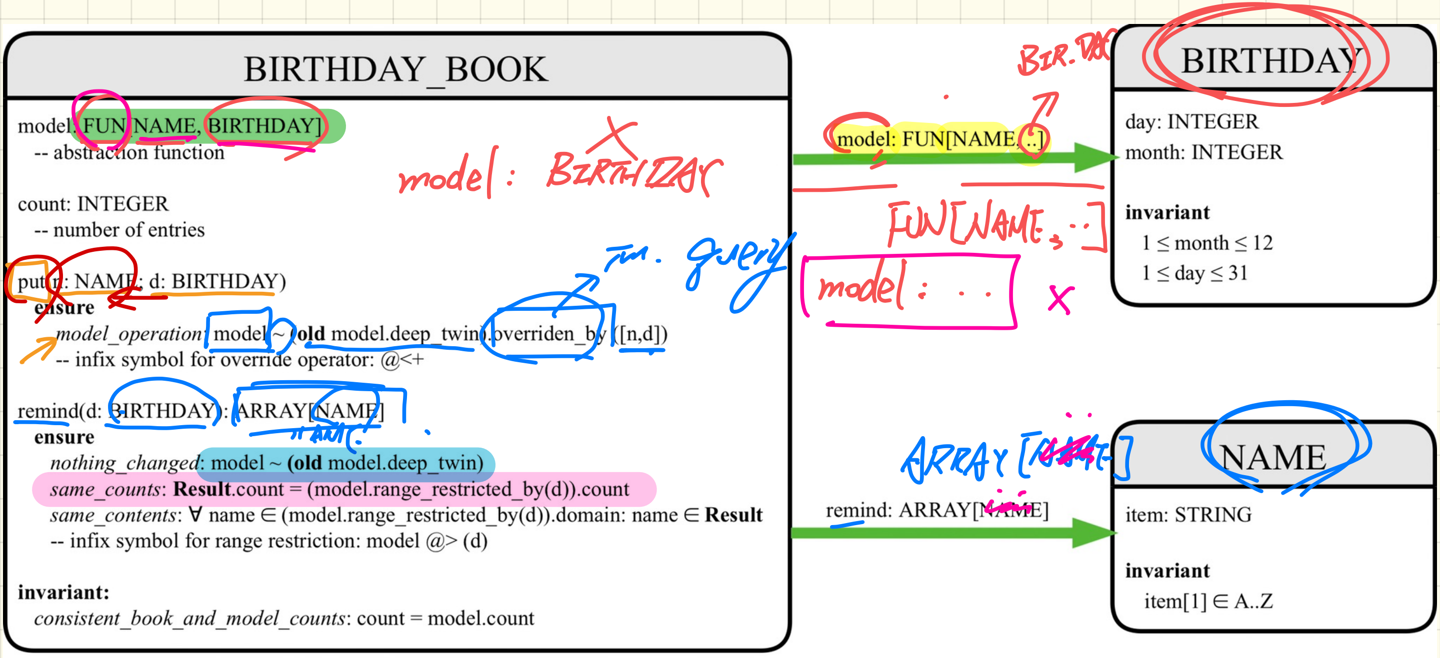
r. overridden_by(..)

↳ immutable query

Model of an Example Birthday Book



Birthday Book: Design



BIRTHDAY_BOOK

model: FUN[NAME, BIRTHDAY]
  -- abstraction function

count: INTEGER
  -- number of entries

put(n: NAME; d: BIRTHDAY)
  ensure
    model_operation: model ~ (**old** model.deep_twin).overriden_by ([n,d])
    -- infix symbol for override operator: @<+

remind(d: BIRTHDAY): ARRAY[NAME]
  ensure
    nothing_changed: model ~ (**old** model.deep_twin)
    same_counts: **Result**.count = (model.range_restricted_by(d)).count
    same_contents: ∀ name ∈ (model.range_restricted_by(d)).domain: name ∈ **Result**
    -- infix symbol for range restriction: model @> (d)

invariant:
  consistent_book_and_model_counts: count = model.count

model: FUN[NAME, ..]

remind: ARRAY[NAME]

BIRTHDAY

day: INTEGER
month: INTEGER

invariant
  1 ≤ month ≤ 12
  1 ≤ day ≤ 31

NAME

item: STRING

invariant
  item[1] ∈ A..Z

model: BIRTHDAY

fn. query

BIR.DAY

FUN[NAME, ..]

model : . .

ARRAY[NAME]

NAME!

Imp.

ns → | alan | mark | tom | "jim" |

birthdays → | jan5 | aug6 | jan5 | dec4 |

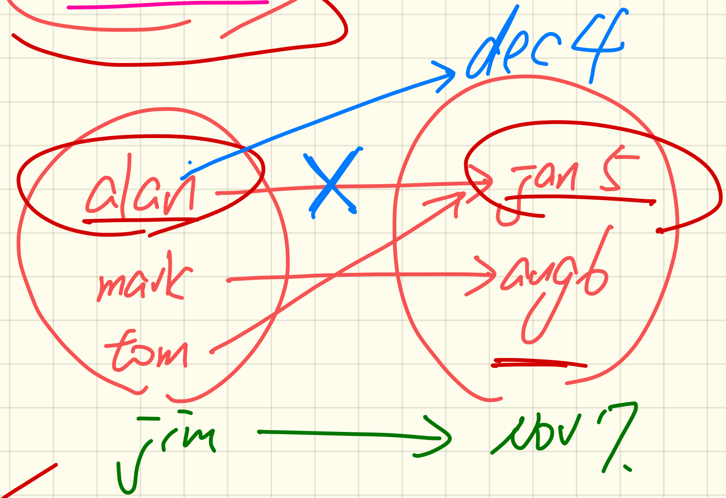
put(alan, dec 4)

put(jim, nov 7)

↳ design decision

: if the name exists, overwrite the birthday.

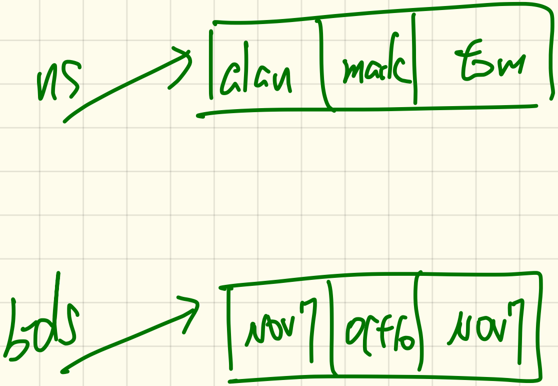
Model



✓ unioned

✓ overridden

Imp

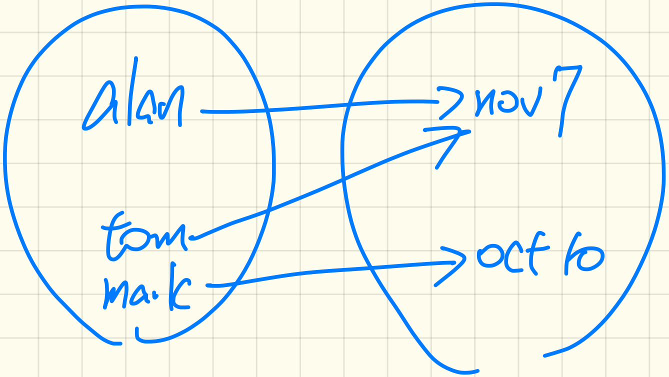


a : ARRAY [STRING]

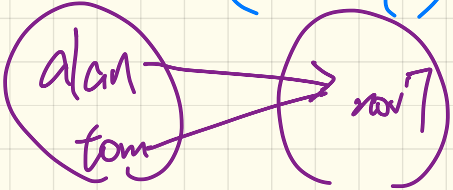
book.remind(nov 7)

↳ << alan, tom >>

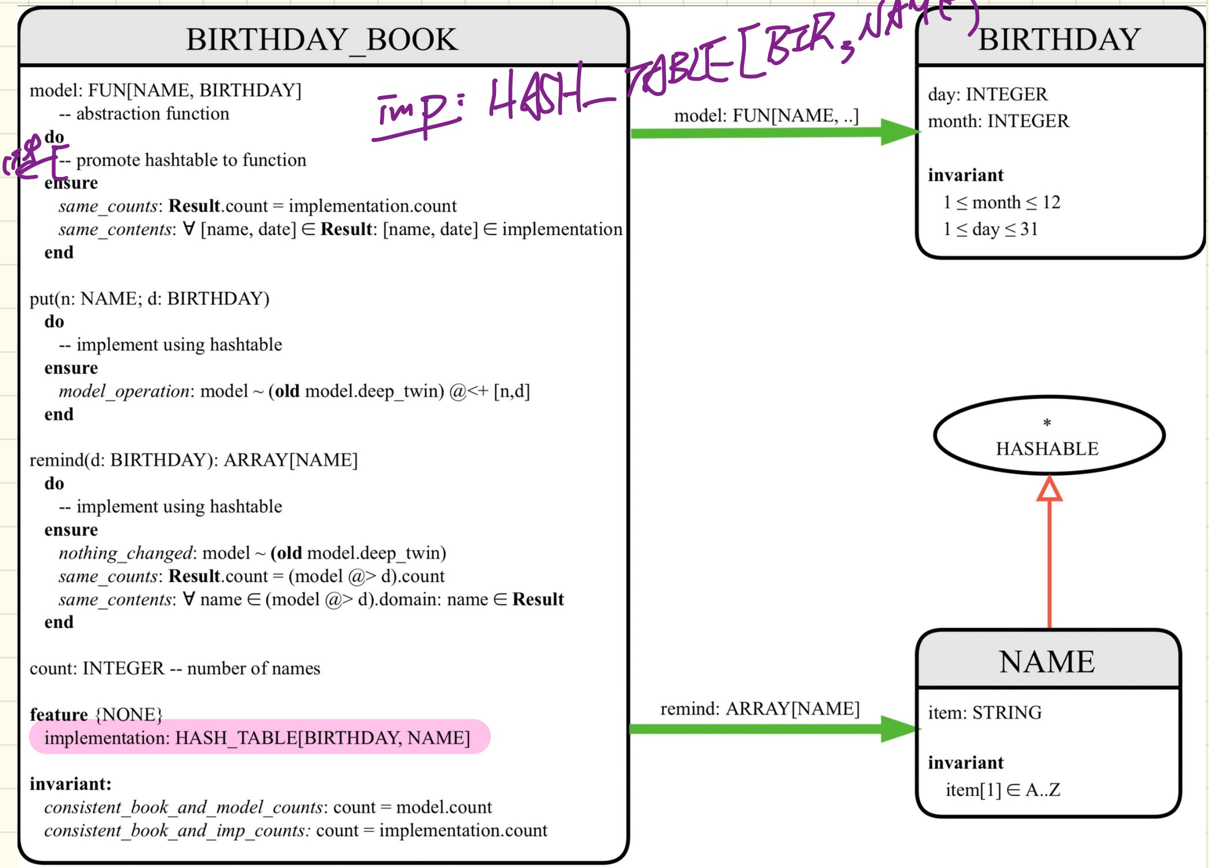
Model



model.range_restricted_by (nov 7)



Birthday Book: Implementation



imp: HASH_TABLE[BIR, NAME]

exercise

BIRTHDAY_BOOK

```
model: FUN[NAME, BIRTHDAY]
    -- abstraction function
  do
    -- promote hashtable to function
  ensure
    same_counts: Result.count = implementation.count
    same_contents: ∀ [name, date] ∈ Result: [name, date] ∈ implementation
  end

put(n: NAME; d: BIRTHDAY)
  do
    -- implement using hashtable
  ensure
    model_operation: model ~ (old model.deep_twin) @<+ [n,d]
  end

remind(d: BIRTHDAY): ARRAY[NAME]
  do
    -- implement using hashtable
  ensure
    nothing_changed: model ~ (old model.deep_twin)
    same_counts: Result.count = (model @> d).count
    same_contents: ∀ name ∈ (model @> d).domain: name ∈ Result
  end

count: INTEGER -- number of names

feature {NONE}
  implementation: HASH_TABLE[BIRTHDAY, NAME]

invariant:
  consistent_book_and_model_counts: count = model.count
  consistent_book_and_imp_counts: count = implementation.count
```

model: FUN[NAME, ..]

BIRTHDAY

```
day: INTEGER
month: INTEGER

invariant
  1 ≤ month ≤ 12
  1 ≤ day ≤ 31
```

*
HASHABLE

remind: ARRAY[NAME]

NAME

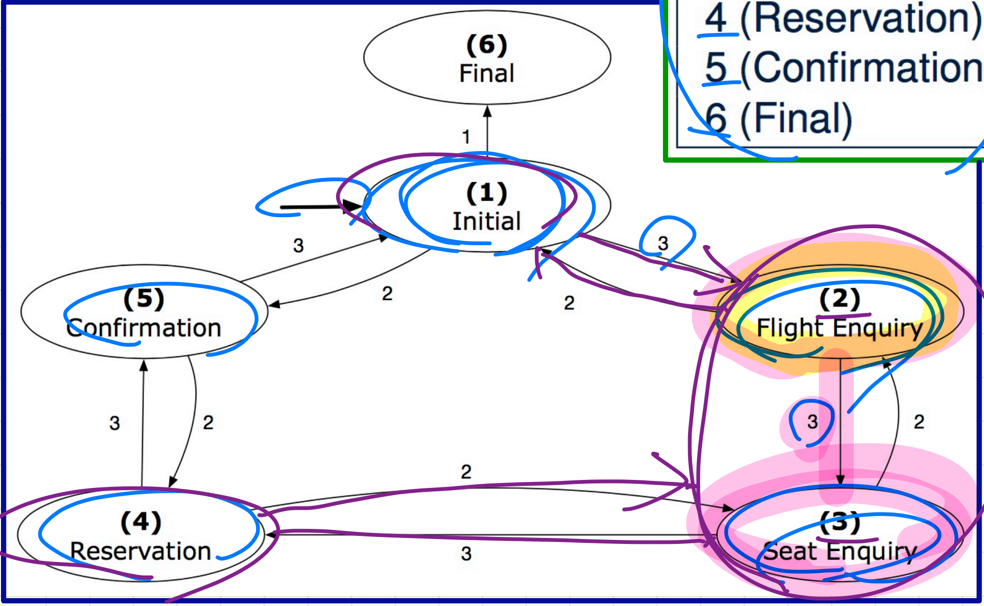
```
item: STRING

invariant
  item[1] ∈ A..Z
```

Finite State Machine (FSM)

State 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)	–	–	–

State Transition Diagram

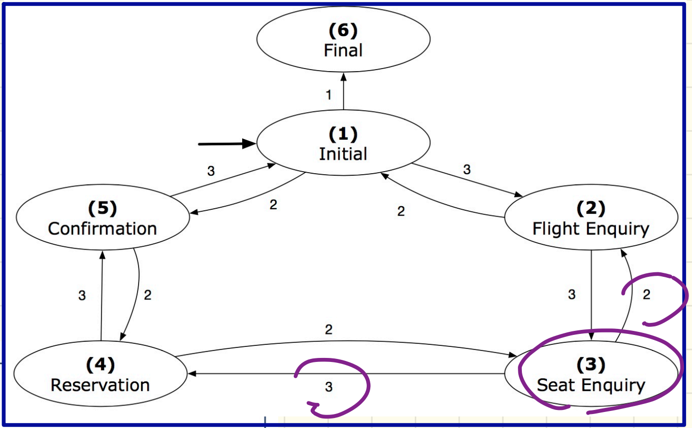


Design of a Reservation System: First Attempt

from
⋮
until
B → as soon as B becomes true, the exit from loop.
loop
⋮
end

while (B) {
j not 3}

as long as B is true, stay.



```
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
```

→ not wrong ans
and
not wrong choice.

while (wrong ans
or
wrong choice)
{ .. }

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
```

Examples: src choice

transition(3, 2) → 2

transition(3, 3) → 4

✓

State Transition Table

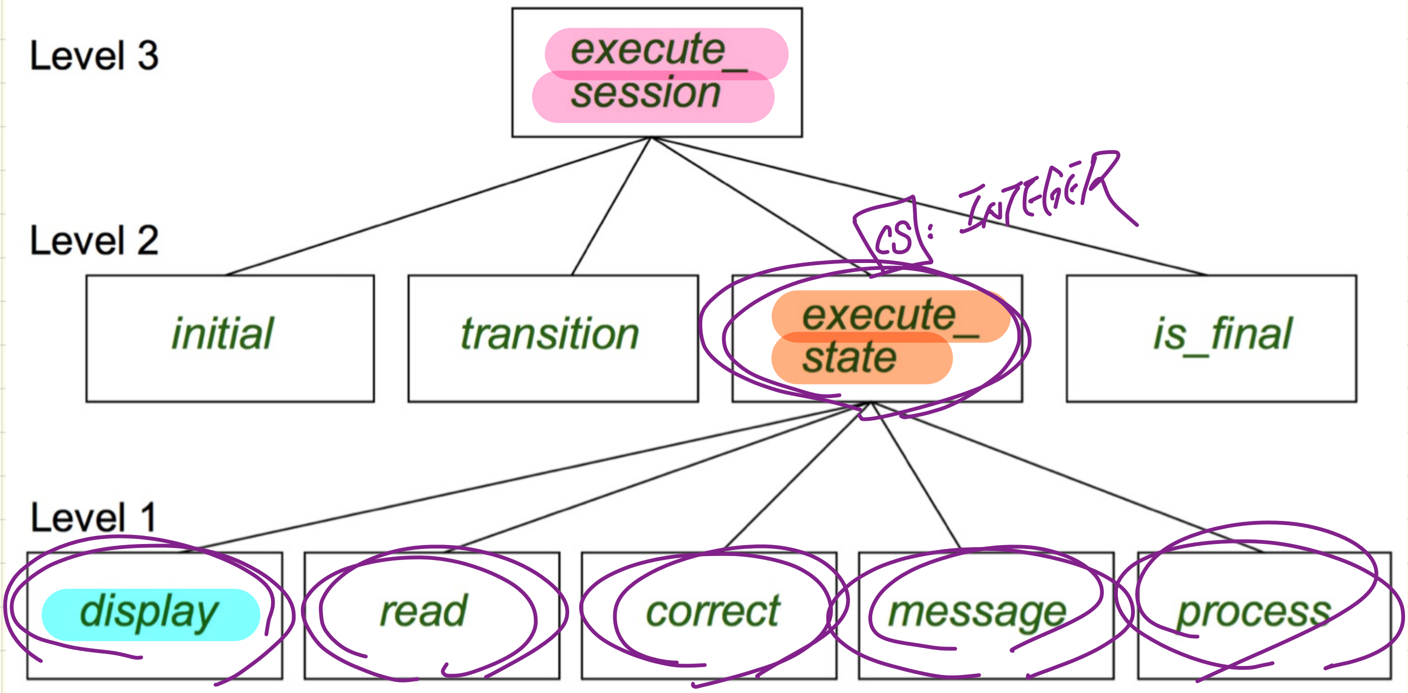
SRC STATE \ CHOICE	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

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: Second Attempt (2)

A Top-Down & Hierarchical Design



LECTURE 18

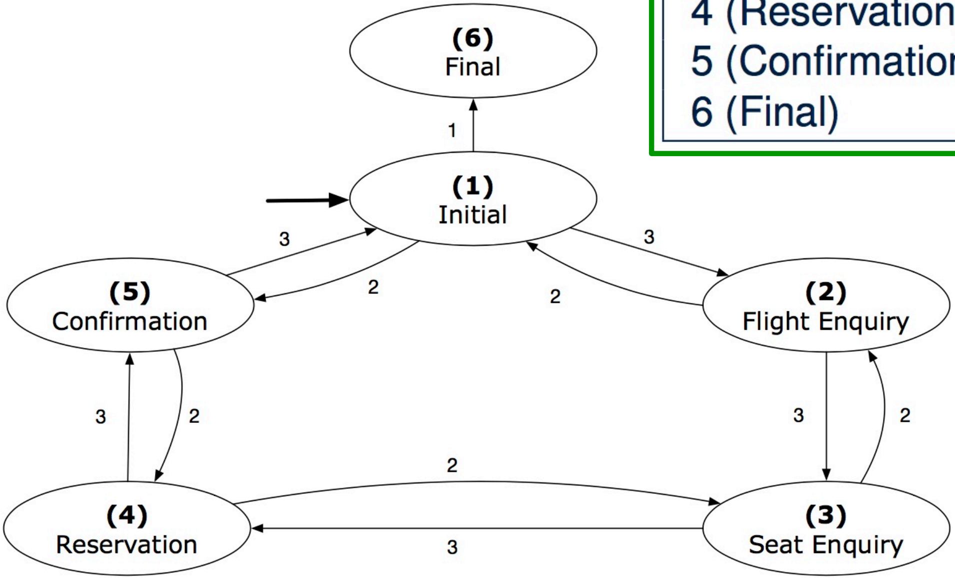
TUESDAY NOVEMBER 12

Finite State Machine (FSM)

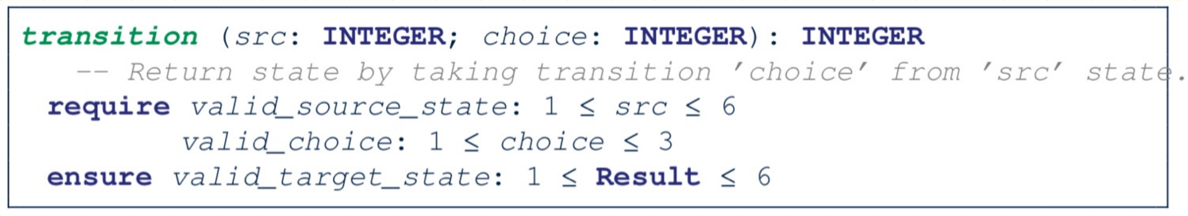
State Transition Table

CHOICE SRC STATE	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)	–	–	–

State Transition Diagram



Design of a Reservation System: Second Attempt (1)



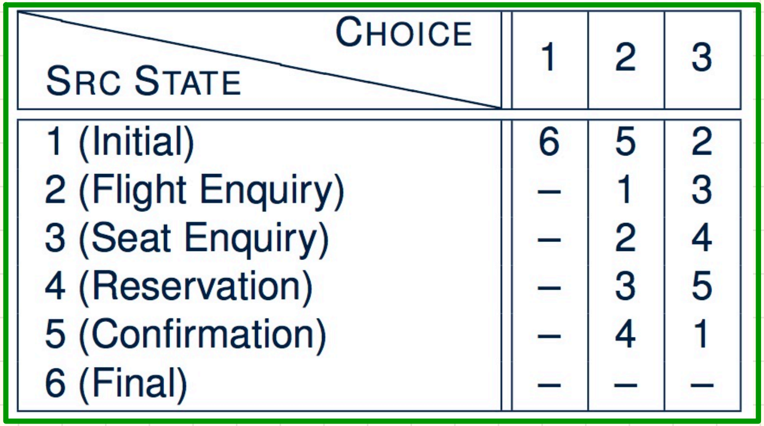
```
transition (src: INTEGER; choice: INTEGER): INTEGER
    -- Return state by taking transition 'choice' from 'src' state.
  require valid_source_state: 1 ≤ src ≤ 6
          valid_choice: 1 ≤ choice ≤ 3
  ensure valid_target_state: 1 ≤ Result ≤ 6
```

Examples:

transition(3, 2)

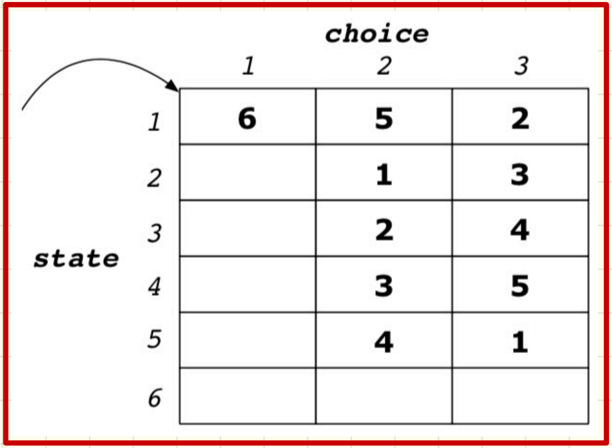
transition(3, 3)

State Transition Table



CHOICE / SRC STATE	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)	–	–	–

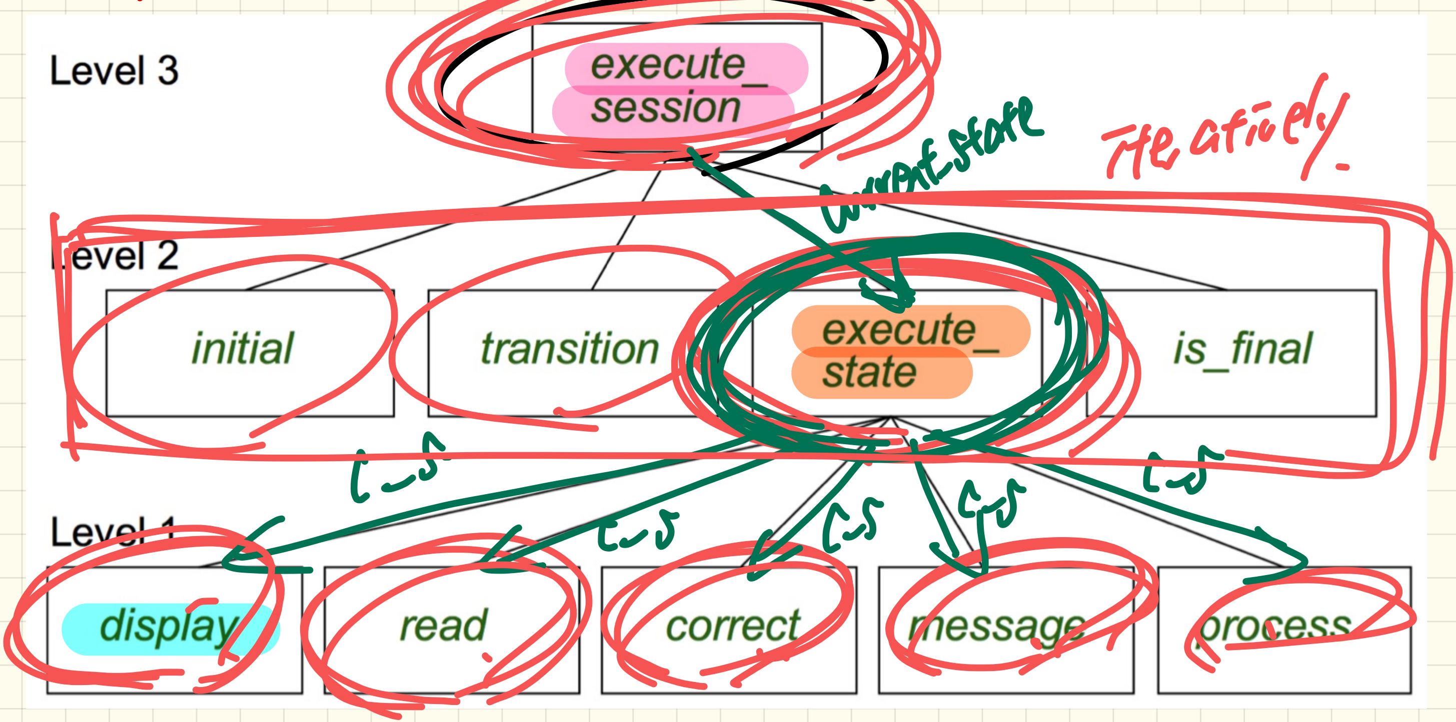
2D Array Implementation



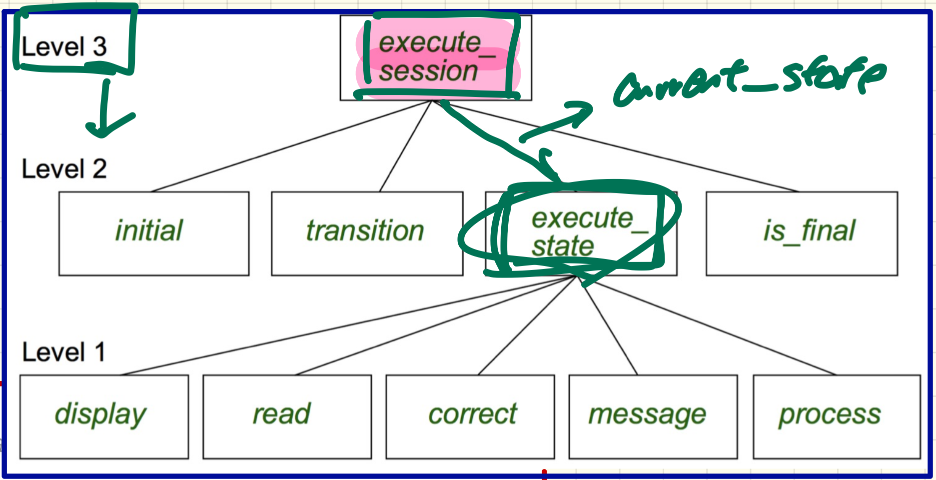
	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

Design of a Reservation System: Second Attempt (2)

A Top-Down & Hierarchical Design



Design of a Reservation System: Second Attempt (3)



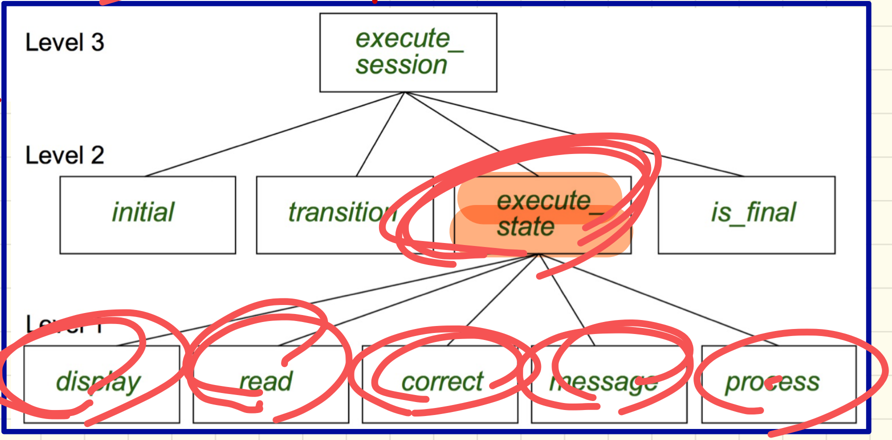
routine

```
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
```

helper routine of execute_session

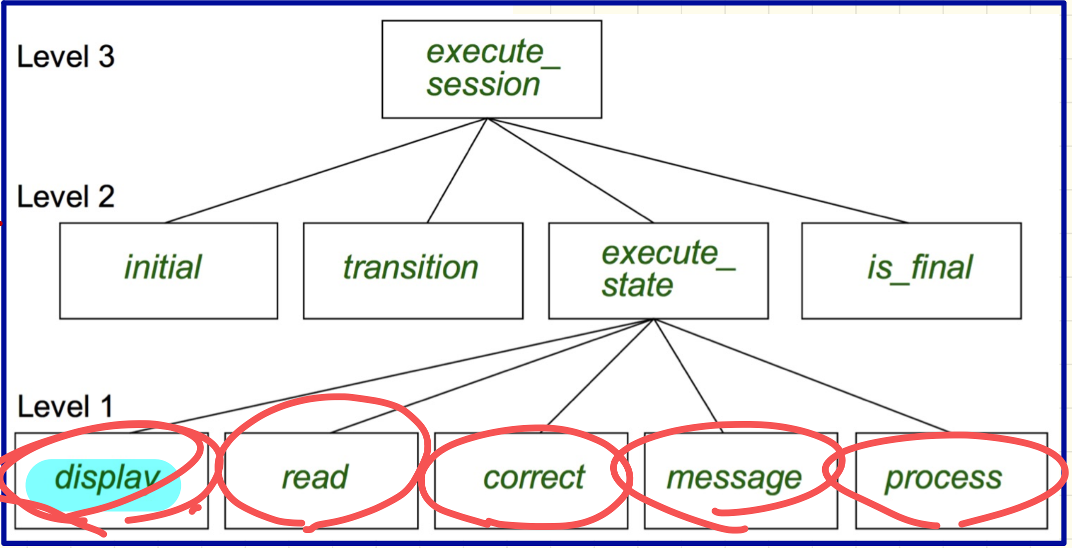
Design of a Reservation System: Second Attempt (4)

```
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 (5)

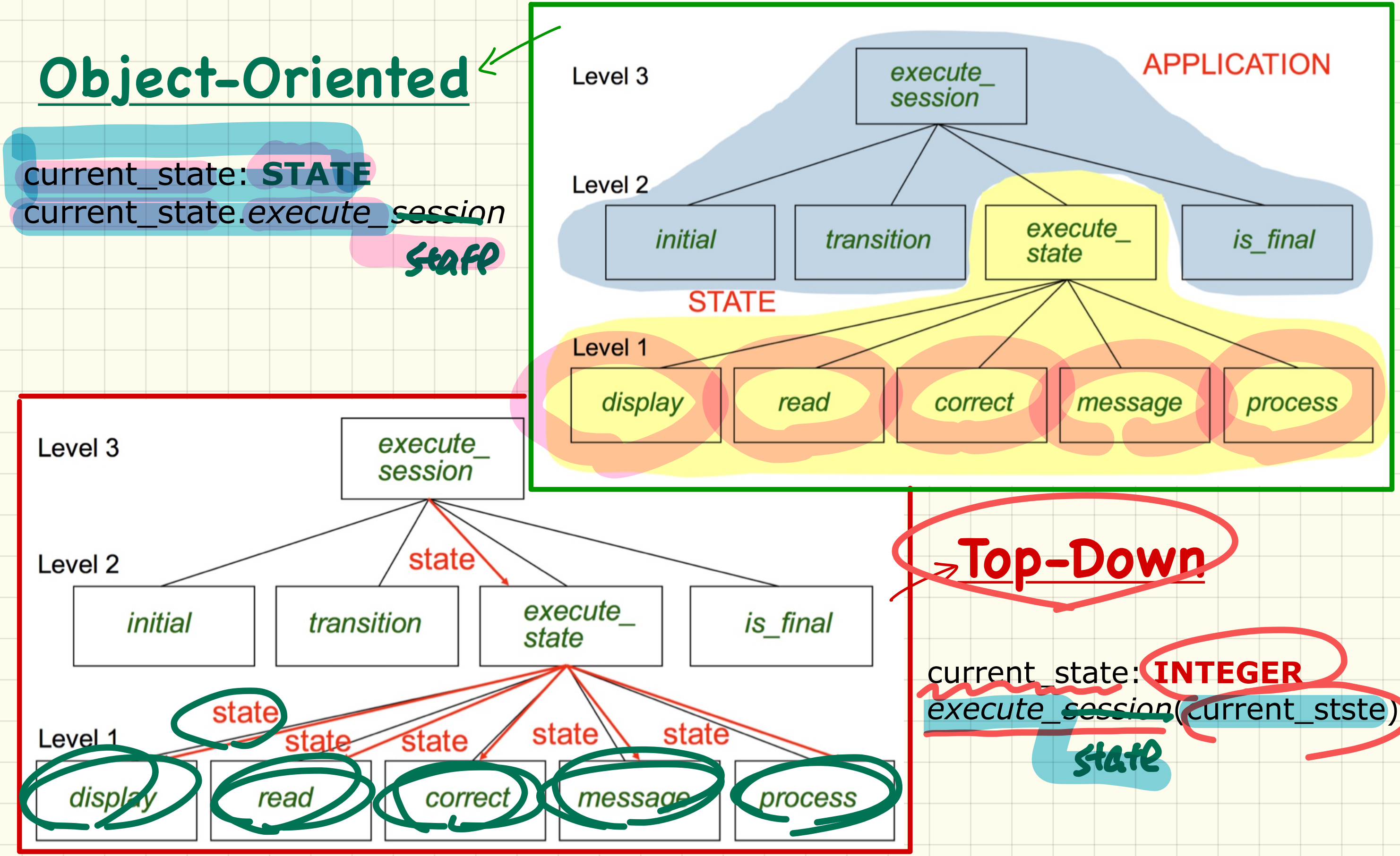
```
display(current_state: INTEGER)
  require
    valid_state: 1 ≤ current_state ≤ 6
  do
    if current_state = 1 then
      -- Display Initial Panel
    elseif current_state = 2 then
      -- Display Flight Enquiry Panel
    else if c_s = 7
    else
      -- Display
    end
  end
```



Moving from Top-Down Design to OO Design

Object-Oriented

current_state: **STATE**
current_state.*execute_~~session~~*
state



Top-Down

current_state: **INTEGER**
execute_~~session~~(current_stste)
state

Non-OO solution

current_state : INT

1 .. 6

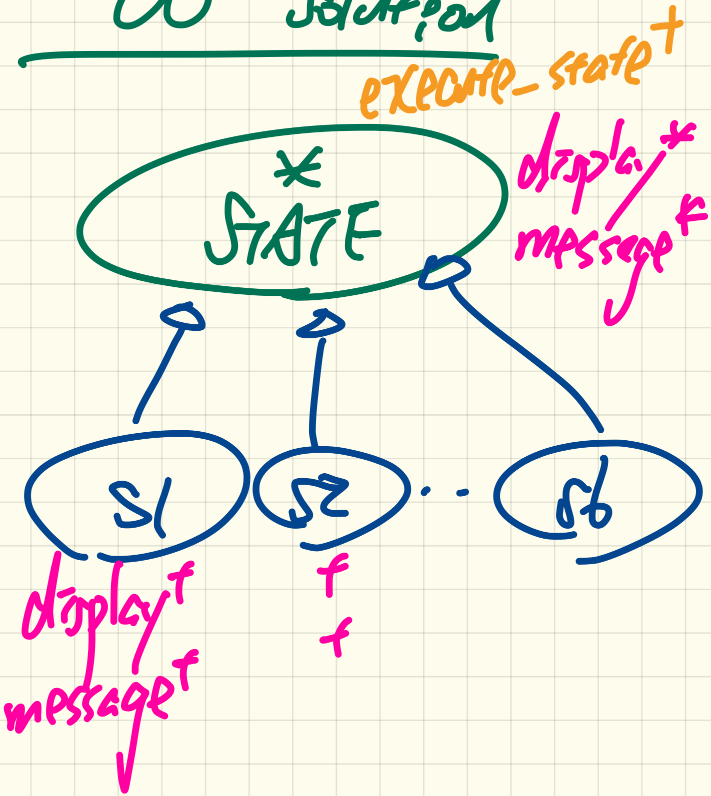
s1 ~ s6

execute_state (cs : INT)

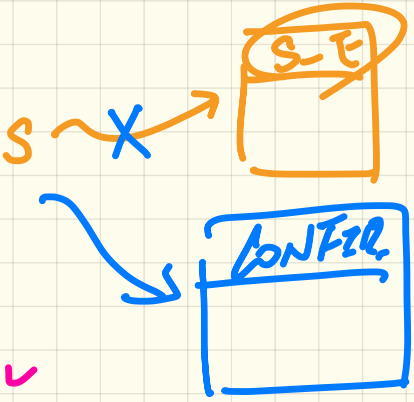
~~display~~ (cs : INT)

~~message~~ (cs : INT)

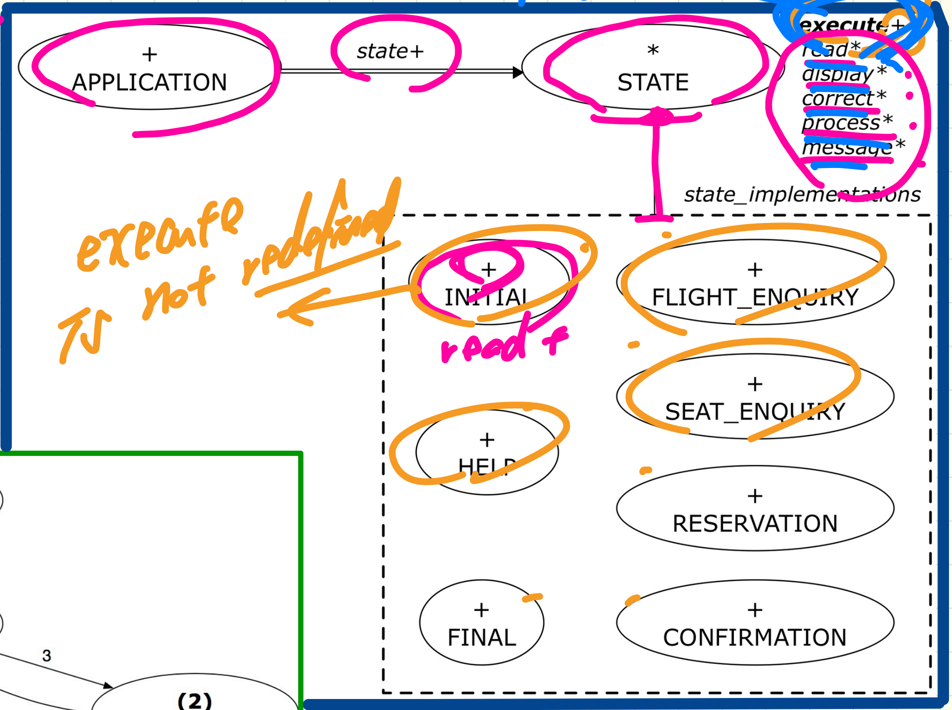
OO solution



State Pattern: Architecture

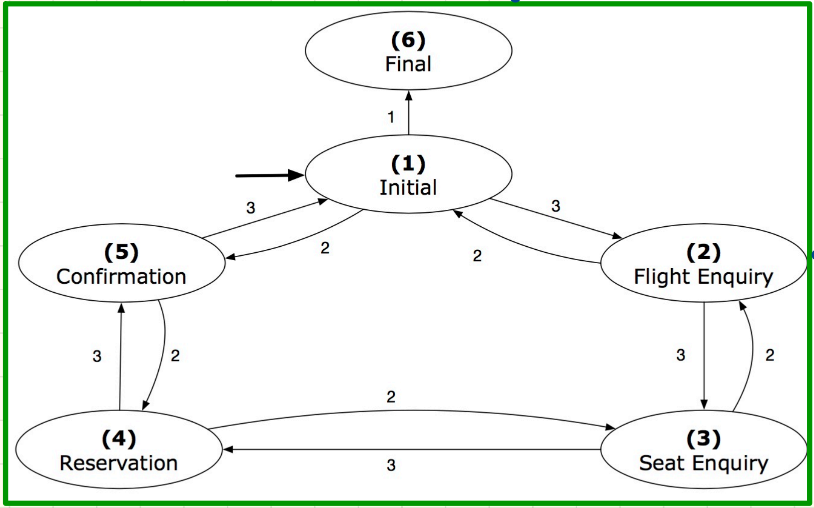


TEMPLATE ← execute_s



execute is not redefined

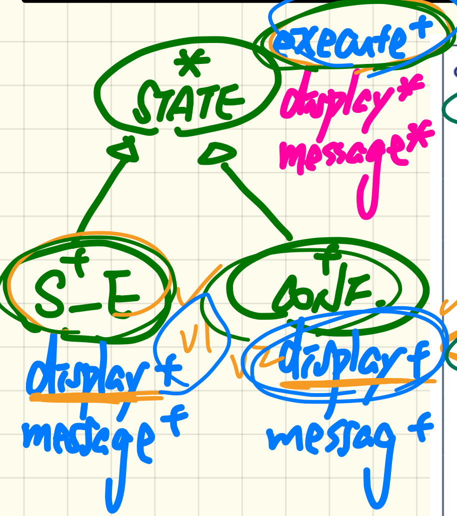
read+



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

→ version from STATE.

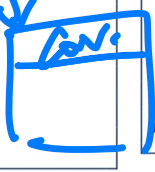
State Pattern: State Module

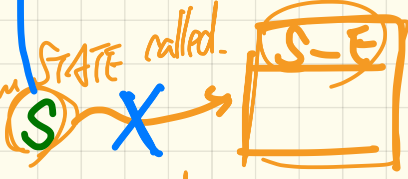


from STATE is called.
execute display from CON.

```
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
```





TEMPLATE

from STATE called.

execute display from S_E.

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

→ DT: S_E. execute

→ DT: CON. execute

s : STATE

<u>Create</u> {STATE} s.make X

↓

deferred.

n_S : INTEGER

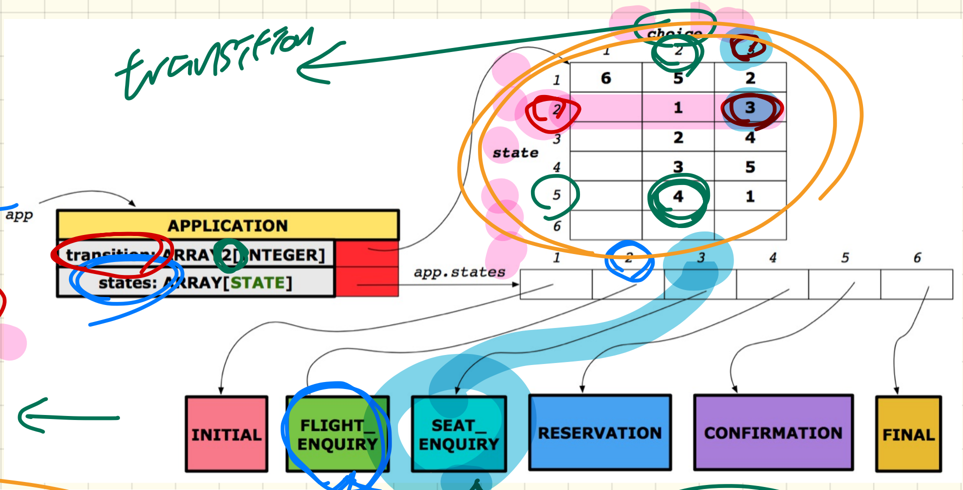
current_state : STATE

c_S := states[2]

~~n_S~~ := transition(2, 3)

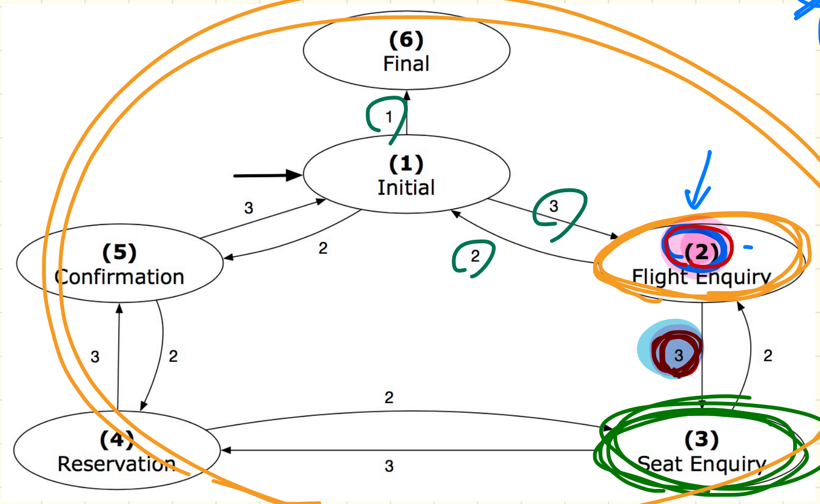
3

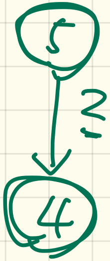
c_S := states[n_S]



	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

c_S





put_transition(4, 5, 2)

State Pattern: Application Module

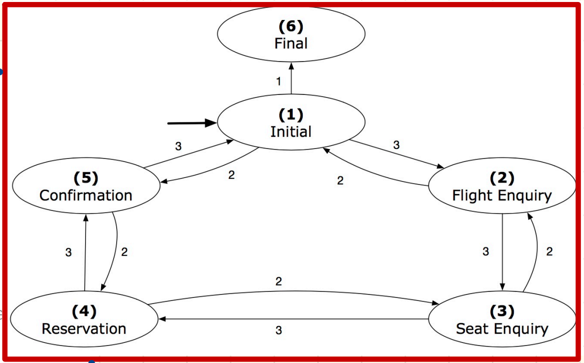
```
class APPLICATION create make
feature {NONE} -- Implementation of Transition Graph
  transition: ARRAY2[INTEGER]
    -- State transitions: transition[state, choice]
  states: ARRAY[STATE]
    -- State for each index, constrained by size of 'transition'
feature
  initial: INTEGER
  number_of_states: 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
end
```

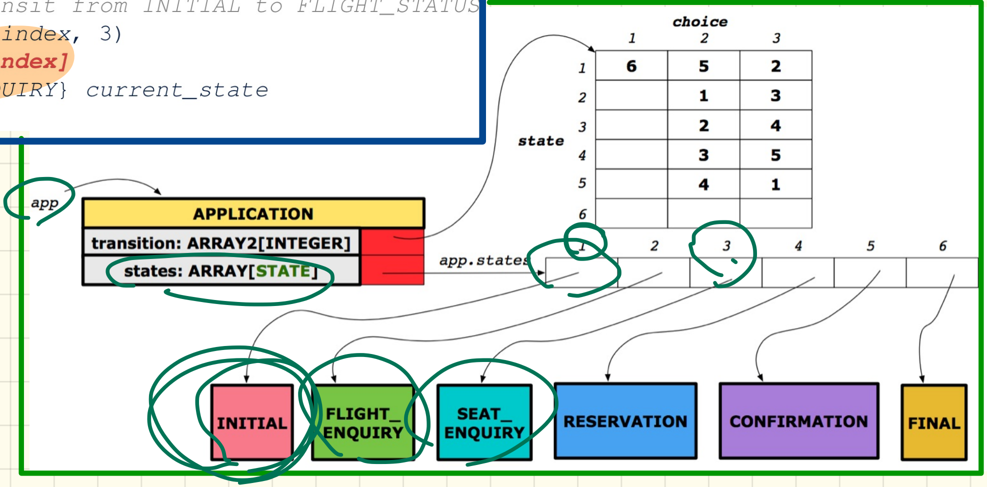
State Pattern: Test

of states

→ # of transitions

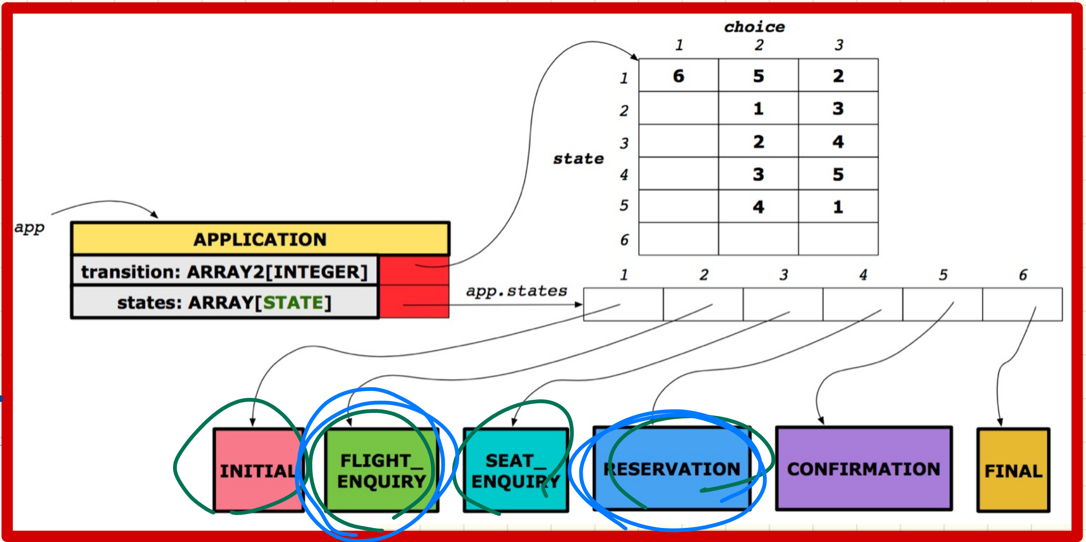
```
test_application: BOOLEAN
  local
    app: APPLICATION ; current_state: STATE ; index: INTEGER
  do
    create app.make (6, 3)
    app.put_state (create {INITIAL}.make, 1)
    -- Similarly for other 5 states.
    app.choose_initial (1)
    -- Transit to FINAL given current state INITIAL and choice
    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
```





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

State Pattern: Interactive Session



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

```
class APPLICATION
feature {NONE} -- Implementat
  transition: ARRAY2[INTEGER]
  states: ARRAY[STATE]
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
```

ST: STATE

indexing into the polymorphic states array.

STATE.

Current_state :=

ST: STATE states[index]

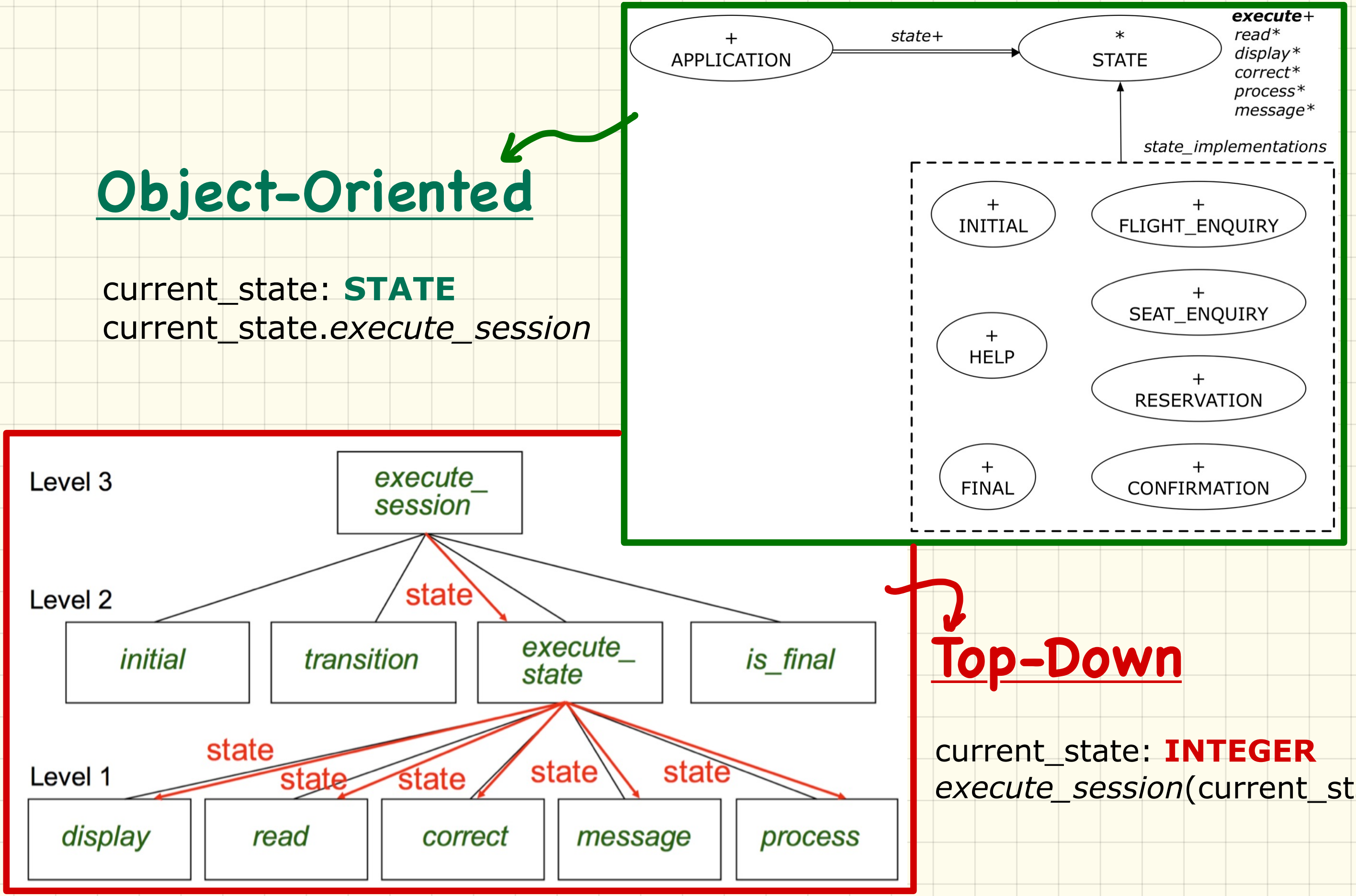
ST: STATE

TEMPLATE

display

message

Interactive System: Top-Down Design vs. OO Design



Object-Oriented

current_state: **STATE**
current_state.*execute_session*

Top-Down

current_state: **INTEGER**
execute_session(current_stste)

LECTURE 19

THURSDAY NOVEMBER 24

$\underline{\{x \mid x > 0\}} \subseteq \underline{\{x \mid x \geq 0\}}$

$1, 2, 3, \ldots$ $\quad$ $\boxed{0}, 1, 2, \ldots$

① $x > 0$

||

$x > 0 \Rightarrow x \geq 0$

② $x \geq 0$

Does ① require **more** or **less** than ②?

$\because$ $x = 0$ is not allowed by ① but is allowed by ②.

✓ ① requires more than ②

② requires less than ①

$$P_1 \Rightarrow P_2$$

precondition precondition

P_2 requires less than P_1

∵ P_2 allows more input values.

$\text{sort}(\text{input} : \text{ARRAY}[\text{INTEGER}])$

ensure

$② \Rightarrow ①$

↳ ② ensures more than ①.

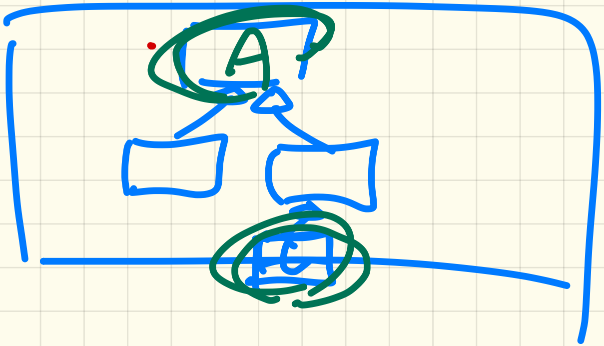
① $\forall i \mid 1 \leq i \leq \text{input.count} - 1 \bullet$

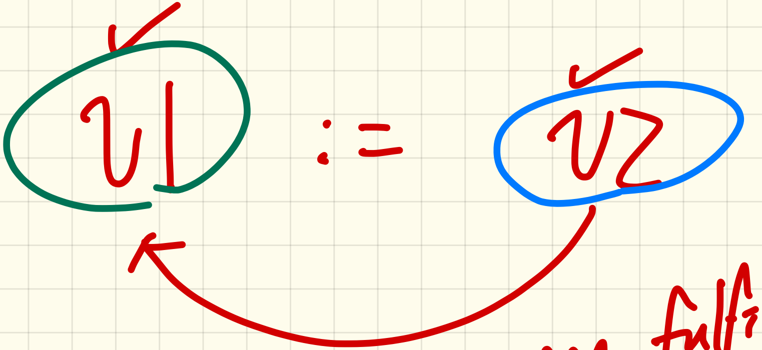
weaker $\boxed{\text{input}[i] \leq \text{input}[i+1]}$

② $\forall i \mid 1 \leq i \leq \text{input.count} - 1 \bullet$

stronger $\boxed{\text{input}[i] < \text{input}[i+1]}$

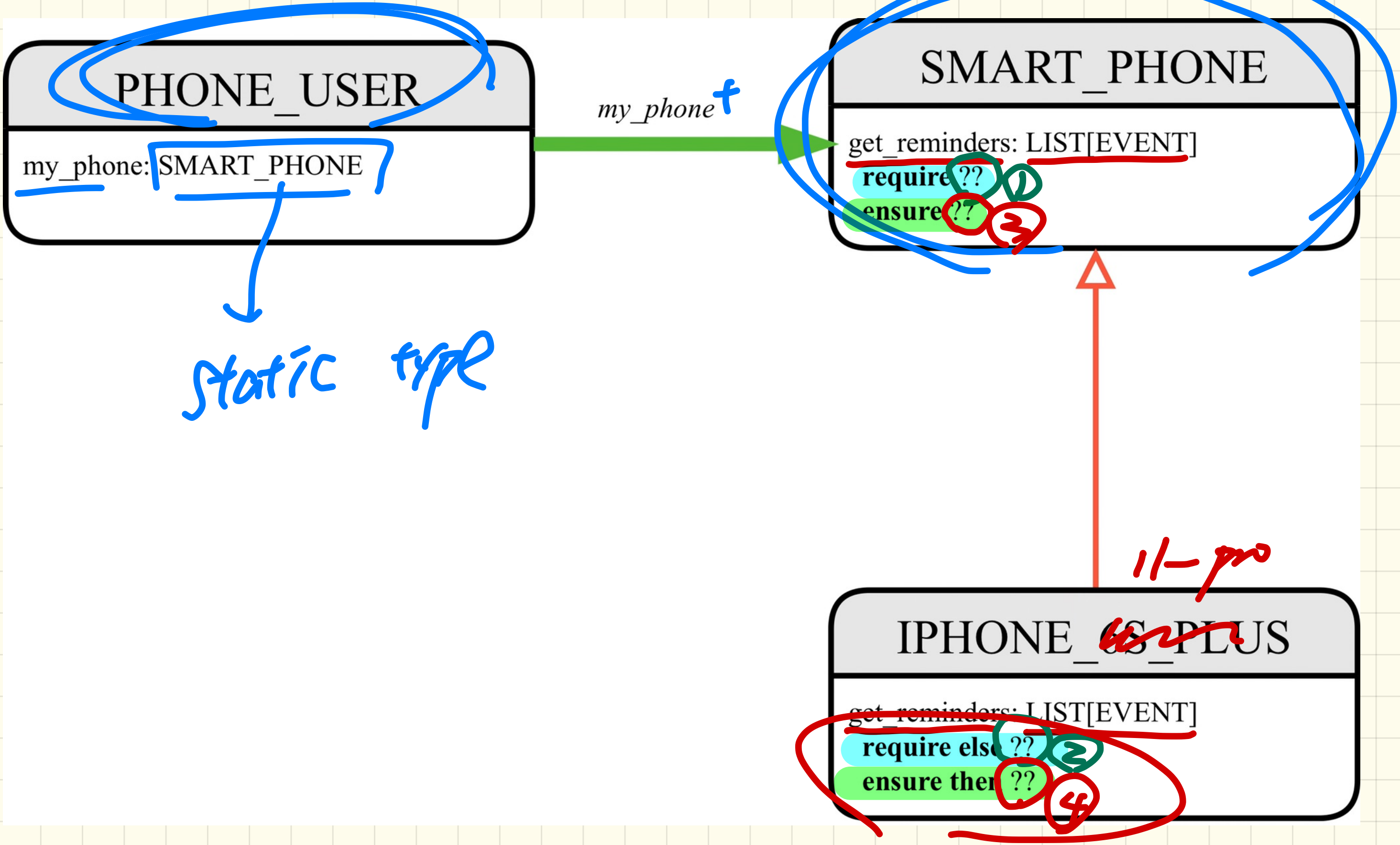
less output values will be able to satisfy.





ST of v2 can fulfill all expectations on the ST of v1.

Subcontracting: Architectural View



PHONE_USER

my_phone: SMART_PHONE

static type

my_phone

SMART_PHONE

get_reminders: LIST[EVENT]

require ?? ①

ensure ?? ③

IPHONE_~~6S~~_PLUS

11-pro

get_reminders: LIST[EVENT]

require else ?? ②

ensure then ?? ④

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
```

T

0.13

13

✓ α requires less than γ

$\gamma \Rightarrow \alpha$

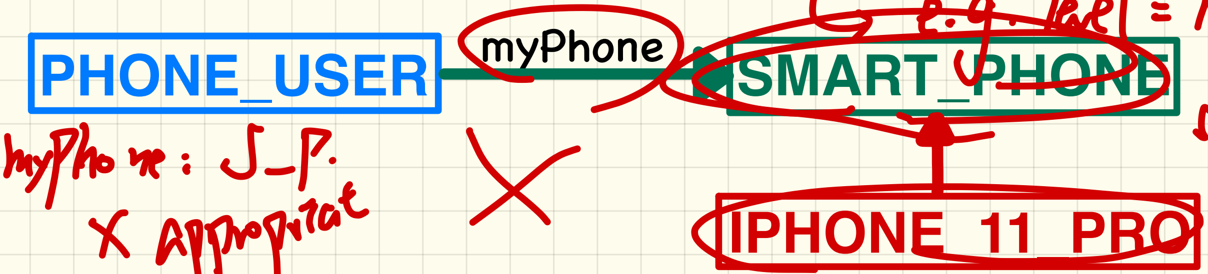
↓ ✓

```
class IPHONE_11_PRO
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
```

F

0.13

γ requires more than α



e.g. level = 13%

↓ satisfies α but fail γ

myPhone : S_P.

✗ Appropriate

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

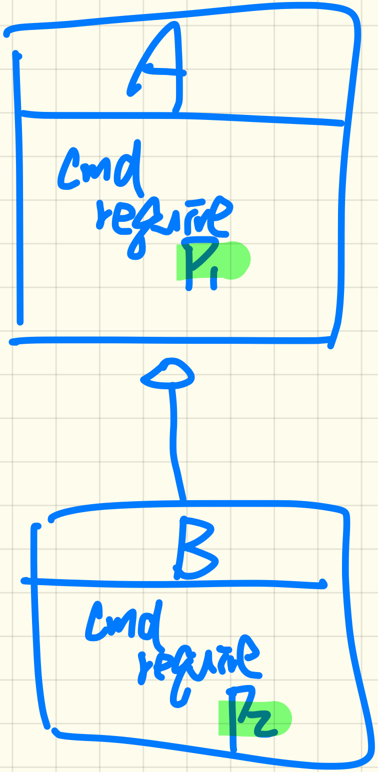
```
class IPHONE_11_PRO
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ ~~0.15~~ 0.05 -- ~~15%~~ 5%
    ensure then
      δ: ∀e: Result | e happens today or tomorrow
end
```

$\alpha \Rightarrow \underline{\gamma}$

level $\geq$ 10% $\Rightarrow$ level $\geq$ 5%

$\{10, 11, 12, \ldots\} \subseteq \{5, 6, 7,$

Exam



Are the preconditions P_1 and P_2 design appropriately?

① To be appropriate:

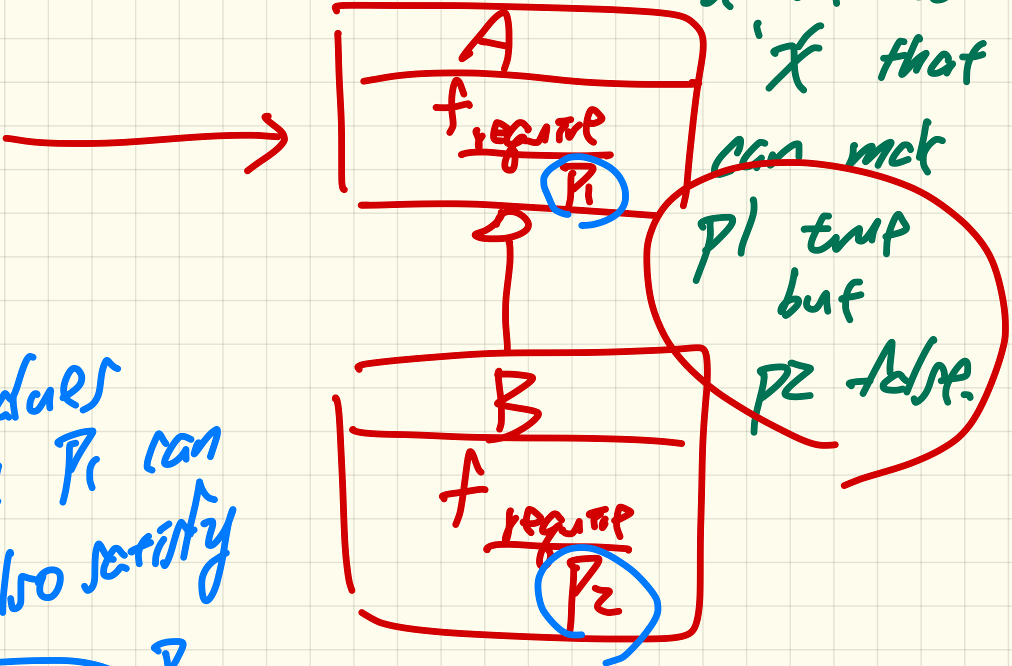
$P_1 \Rightarrow P_2$ (P_2 less strict)

② prove it (e.g. counterexample)

P	Q	P ⇒ Q
F	F	T
F	T	T
T	F	F
T	T	T

T

¬(P1 ⇒ P2) means there is a witness 'x' that can make P1 true but P2 false.



any input values that satisfy P1 can also satisfy P2

$\boxed{P_1 \Rightarrow P_2}$ P2

α : level $\geq 10\%$

$\qquad\quad 10$

γ : level $\geq 15\%$

allows less values ?

$$\{x \mid \alpha(x)\} = \{\underline{10\%}, \underline{11\%}, \underline{12}, \underline{13}, \underline{14}, \dots\}$$

$$\{y \mid \underline{\gamma}(y)\} = \{15\%, 16\%, \dots\}$$

Subcontracting: Example (2)

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

① If β and δ are appropriate, then:

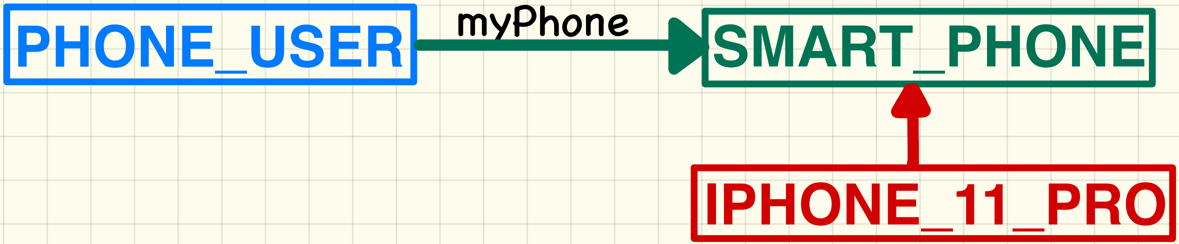
$\delta \Rightarrow \beta$

```
class IPHONE_11_PRO
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 the case.

counter example:

list of events containing only those tom.

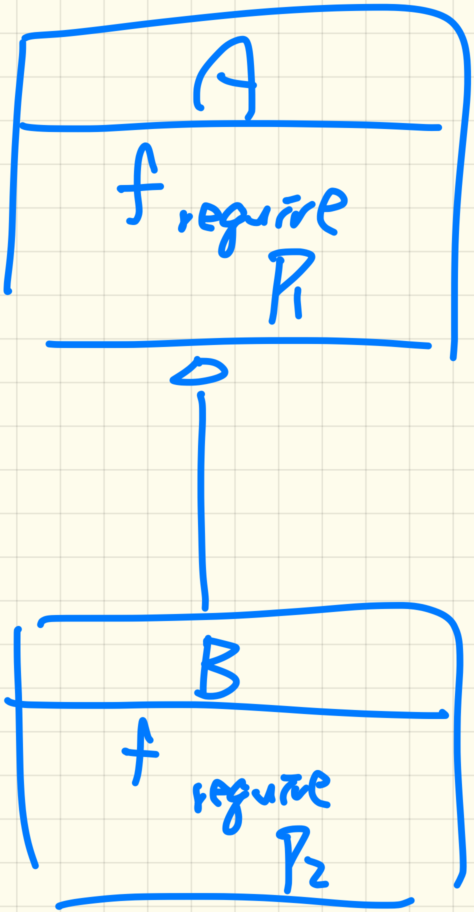


PHONE_USER

myPhone

SMART_PHONE

IPHONE_11_PRO



Valid:
$P_1 \Rightarrow P_2$

Is it ever possible that our design requires $P_2 \Rightarrow P_1$?

↳ poor design ∵ it breaks substitutability.

$$S_1 \subseteq S_2$$

$\hookrightarrow$ to disprove it,

find x s.t.

$x \in S_1 \wedge x \notin S_2$.

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

P1

not appropriate :(

13%

```
class IPHONE_11_PRO
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
```

P2 → should be weaker when T ⇒ not; T has no effect

s : SMART_PHONE

Create {IP_11_PRO} s.make

s.get_reminders.

Run time checks

level ≥ 10% ✓

level ≥ 15%

T

level $\geq$ 10% || or else level $\geq$ 15%

T

↯

short-circuit effect

X don't bother

13%

and then

p1	p2	p1 && p2	p2 || p1
F	F		
F	T		
T	F		
T	T		

p1 and then ~~p2~~

p1 or else ~~p2~~

T

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

```
class IPHONE_11_PRO
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
```

won't compile

won't compile.

$x>1$? $2y$: $3z$

↳ if $x>1$ then

$2y$

else $3z$. end

LECTURE 20

FRIDAY NOVEMBER 15

Preconditions

subclass

P_1 requires less than P_2

$P_2 \Rightarrow P_1$ e.g. $x=9$

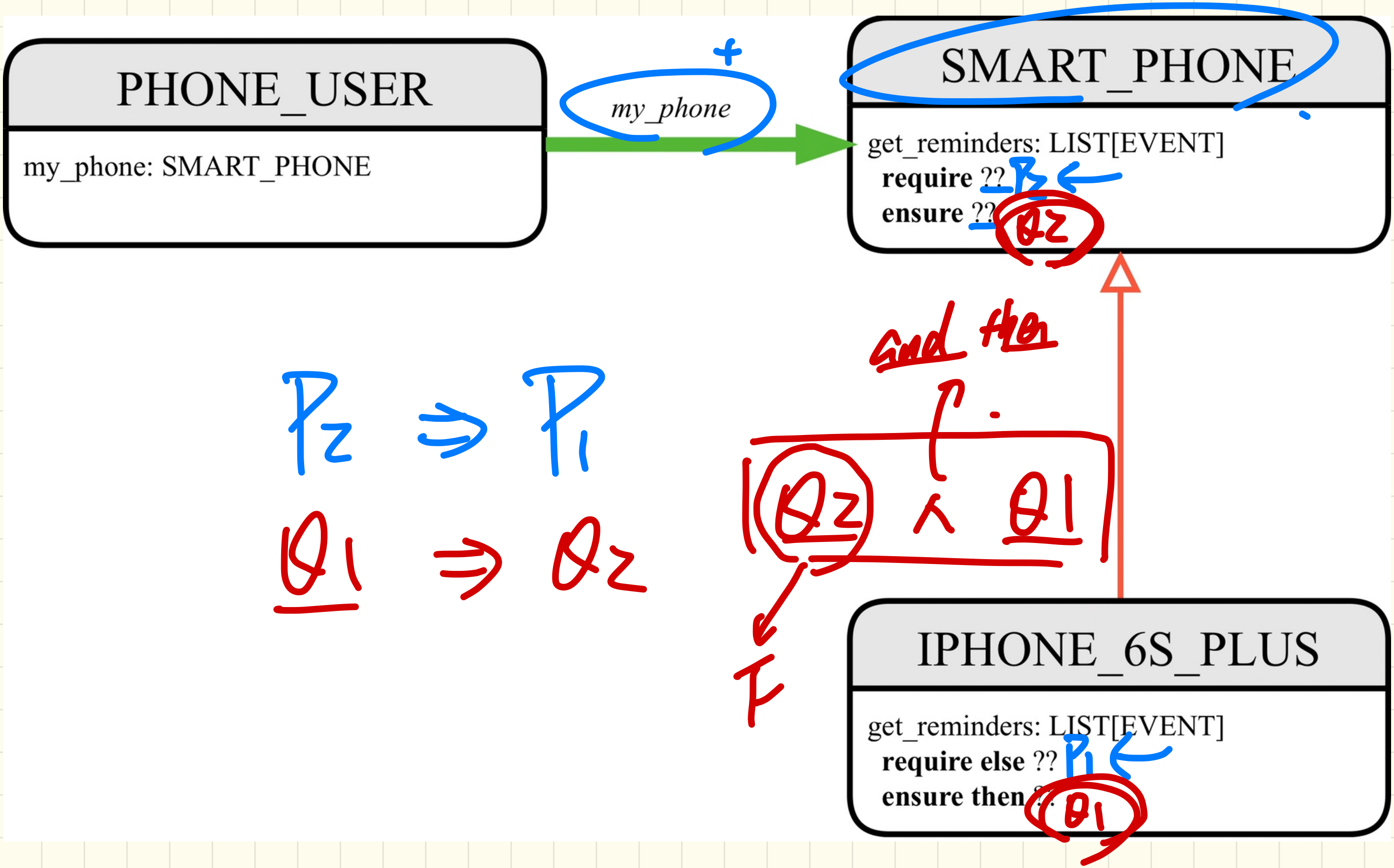
$x>0$ $x \geq 0$

Postconditions

Q_1 ensures more than Q_2

subclass. $Q_1 \Rightarrow Q_2$

Subcontracting: Architectural View



PHONE_USER

my_phone: SMART_PHONE

SMART_PHONE

get_reminders: LIST[EVENT]
require ?? P_2
ensure ?? Q_2

IPHONE_6S_PLUS

get_reminders: LIST[EVENT]
require else ?? P_1
ensure then ?? Q_1

$P_2 \Rightarrow P_1$

$Q_1 \Rightarrow Q_2$

and then

$Q_2 \wedge Q_1$

F

Subcontracting: Example (1)

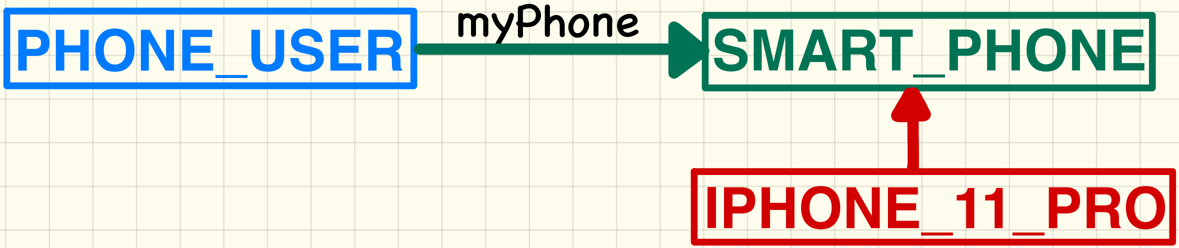
```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

level ≥ 10% (13%)

level ≥ 10% ⇒ level ≥ 15%

```
class IPHONE_11_PRO
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
```

level ≥ 15%



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

weaker

T

13%.

substitutability.

```
class IPHONE_11_PRO
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
```

stronger

not appropriate

F

Runtime.

level ≥ 10% or else level ≥ 15%

T

T

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

level ≥ 10% ⇒ level ≥ 5%

```
class IPHONE_11_PRO
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ ~~0.15~~ 5% -- 15%
    ensure then
      δ: ∀e:Result | e happens today or tomorrow
end
```

p: SMART_PHONE

create {IP_11_PRO} p.make

6% → p.get_reminders.

6%

level ≥ 10%

or else

6%

level ≥ 5%

F

T

f

require

p1

p2

$p1 \wedge p2$

ensure

q1

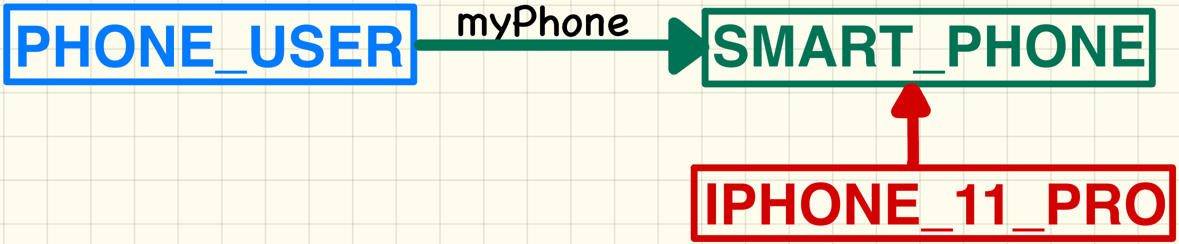
q2

$q1 \wedge q2$

Subcontracting: Example (2)

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

```
class IPHONE_11_PRO
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
```



Contract Re-Declaration:

Missing Pre-Condition in Ancestor

true or else x>0

```
class FOO
  f  require true
    do ...
    end
end
```

```
class BAR
inherit FOO redefine f end
  f require else new_pre
                 x > 0
    do ...
    end
end
```

Runtime

if in the parent class there's no precondition ⇒ no precond. in all descendants.

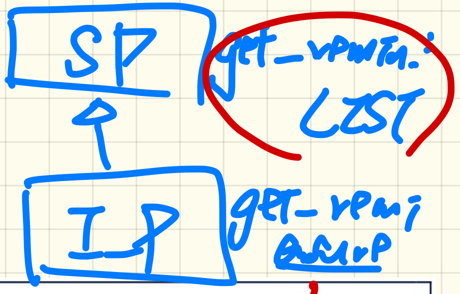
① true or else new_pre

② false or else new_pre

weaken f at the FOO level.

Contract Re-Declaration:

Missing Post-Condition in Ancestor



if no postcondition in parent class ⇒ expected that some postcondition will be added in descendants.

```
class FOO
  f
     do ...
     end
end
```

same

```
class BAR
inherit FOO redefine f end
  f
     do ...
     ensure then new_post
     end
end
```

today

F

never succeed at runtime ← [① false and then new_post]

② true and then new_post

Contract Re-Declaration:

Missing Pre-Condition in Descendant

out

```
class FOO
  f require
      ~~original_pre~~ level >= 10%
    do ...
    end
end
```

```
class BAR
inherit FOO redefine f end
  f [ ].
    do ...
    end
end
```

① original_pre or else true

T

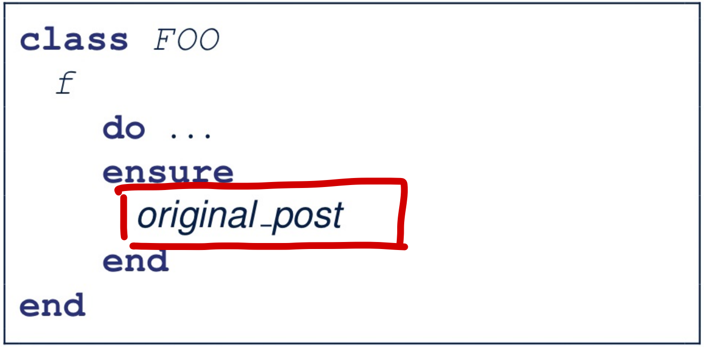
$\geq 10\%$

$\equiv$ true

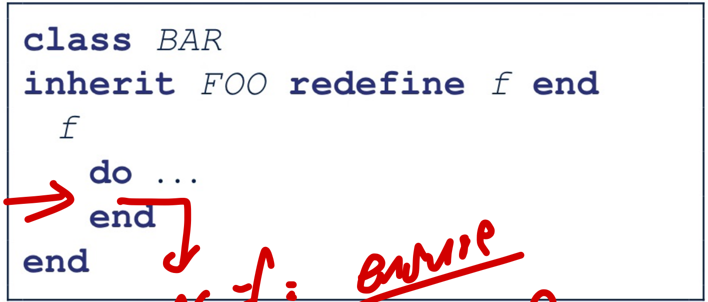
↳ not appropriate ∵ no constraint on f.

Contract Re-Declaration:

Missing Post-Condition in Descendant



```
class FOO
  f
    do ...
    ensure
      original_post
    end
end
```



```
class BAR
inherit FOO redefine f end
  f
    do ...
    end
end
```

as if: ~~ensure~~ true

original_post and then false F.

↳ not appropriate

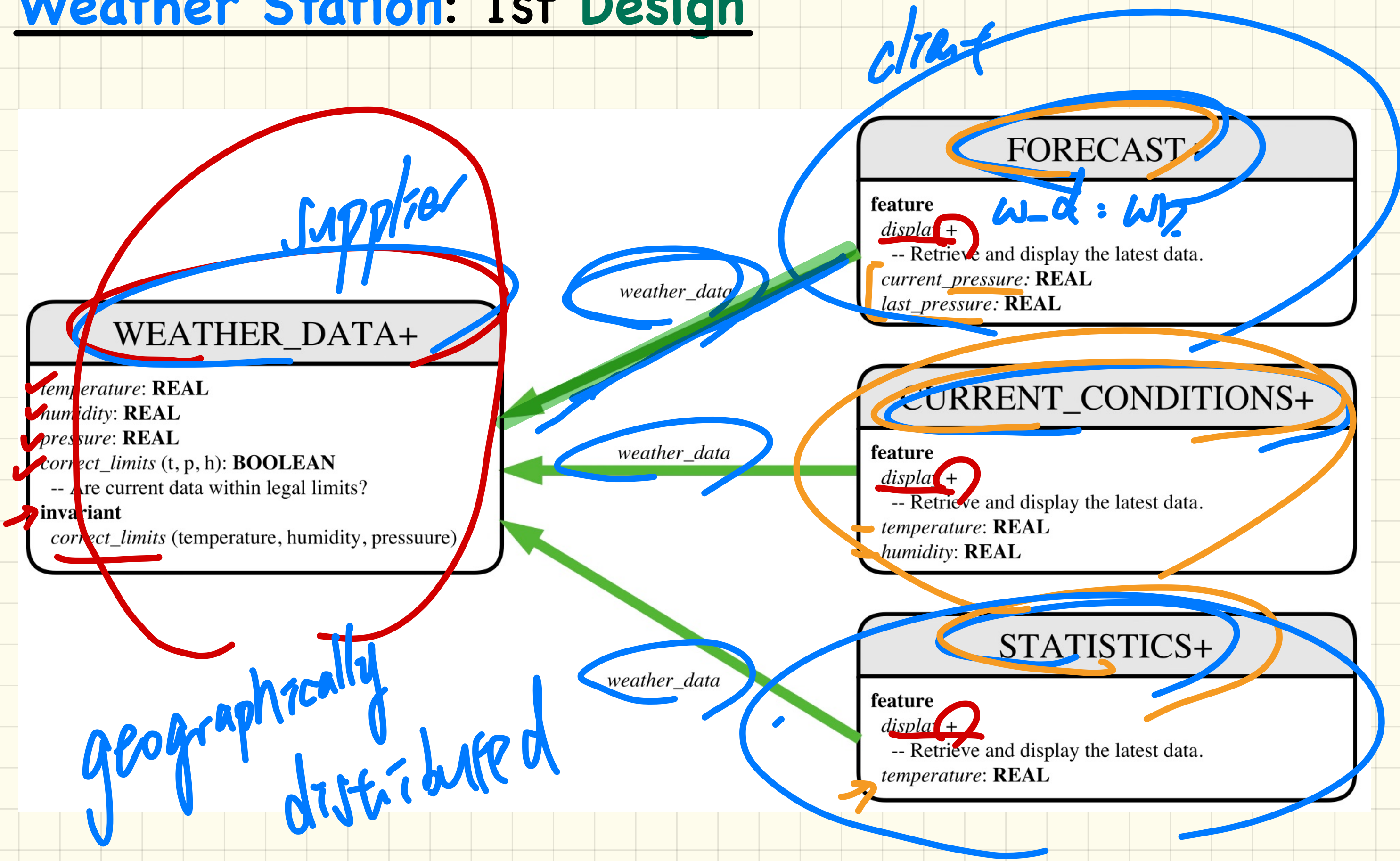
↳ ∵ no supplier can satisfy false.

- if there's $\underline{\underline{B}}$

  s.t. B should be the postcond. of every Routine (Cmd or query)

  ↳ B should be a class invariant.

Weather Station: 1st Design



supplier

WEATHER_DATA+

temperature: **REAL**
humidity: **REAL**
pressure: **REAL**
correct_limits (t, p, h): **BOOLEAN**
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)

geographically distributed

weather_data

weather_data

weather_data

client

FORECAST+

w_d : WD

feature
display+
-- Retrieve and display the latest data.
current_pressure: **REAL**
last_pressure: **REAL**

CURRENT_CONDITIONS+

feature
display+
-- Retrieve and display the latest data.
temperature: **REAL**
humidity: **REAL**

STATISTICS+

feature
display+
-- Retrieve and display the latest data.
temperature: **REAL**

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
```

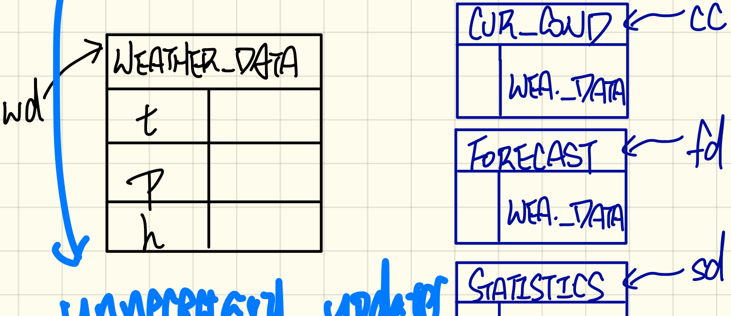
```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature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

after this, app must update



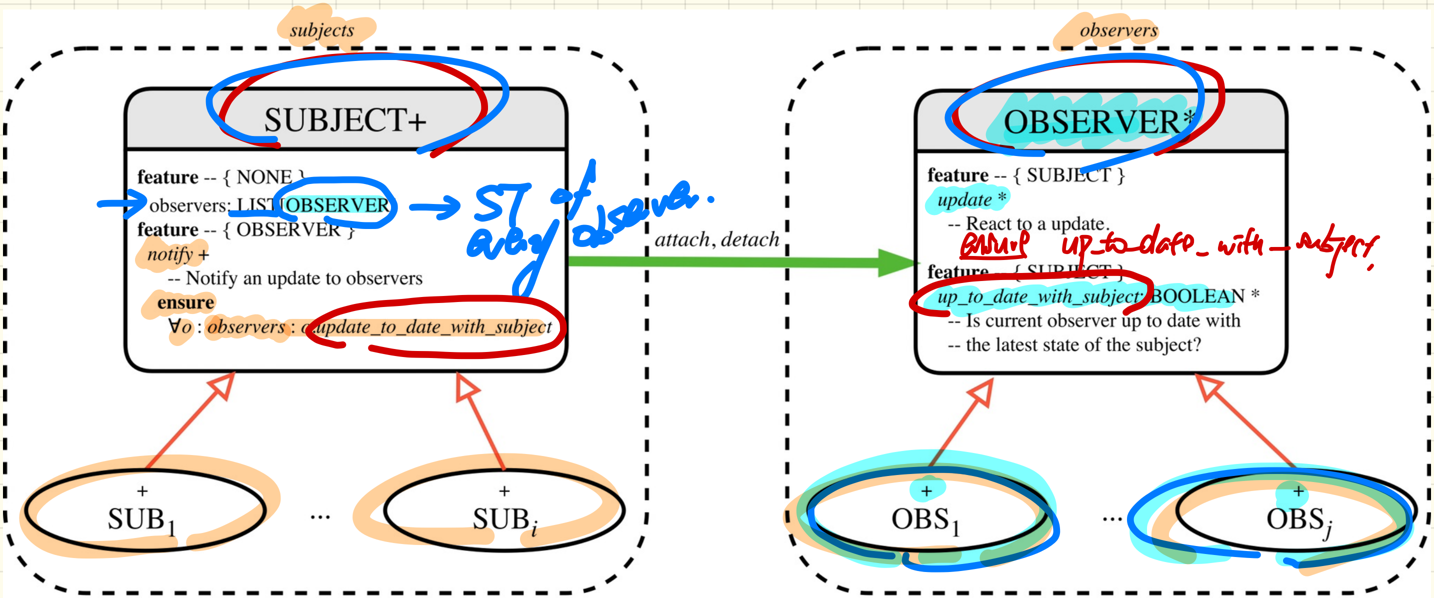
unnecessary updates
∵ no changes on measurement.

```
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



subjects

SUBJECT+

feature -- { NONE }

observers: LIST[OBSERVER] → ST of every observer.

feature -- { OBSERVER }

notify +

-- Notify an update to observers

ensure

$\forall o : observers : o.up\_to\_date\_with\_subject$

attach, detach

observers

OBSERVER*

feature -- { SUBJECT }

update *

-- React to a update.

ensure up_to_date_with_subject

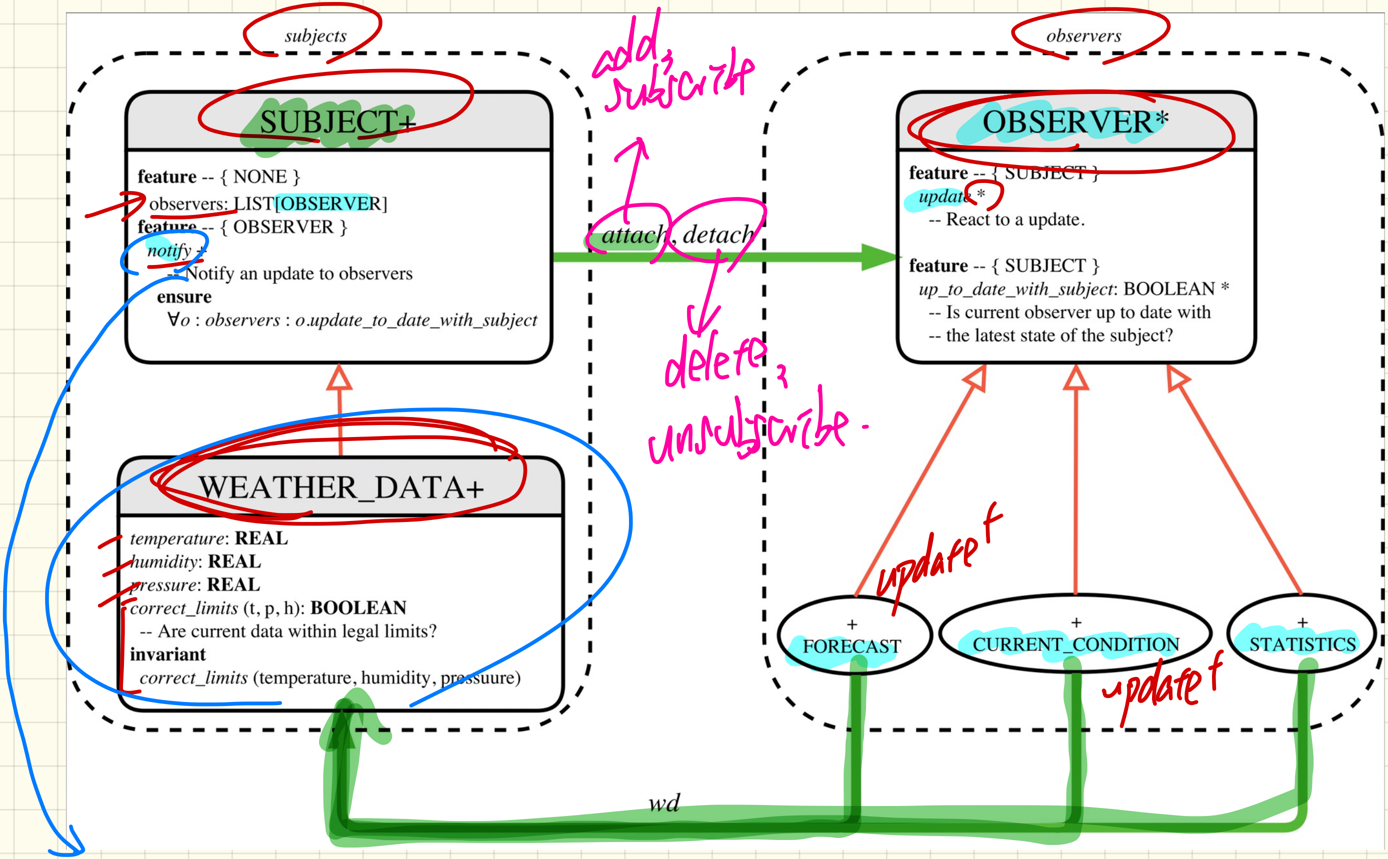
feature -- { SUBJECT }

up_to_date_with_subject: BOOLEAN *

-- Is current observer up to date with

-- the latest state of the subject?

The Observer Pattern: Application to Weather Station



subjects

SUBJECT+

feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
∀*o* : *observers* : *o.update_to_date_with_subject*

WEATHER_DATA+

temperature: **REAL**
humidity: **REAL**
pressure: **REAL**
correct_limits (t, p, h): **BOOLEAN**
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)

attach, *detach*

add, subscribe

delete, unsubscribe.

observers

OBSERVER*

feature -- { SUBJECT }
update *
-- React to a update.

feature -- { SUBJECT }
up_to_date_with_subject: BOOLEAN *
-- Is current observer up to date with
-- the latest state of the subject?

\+ FORECAST

\+ CURRENT_CONDITION

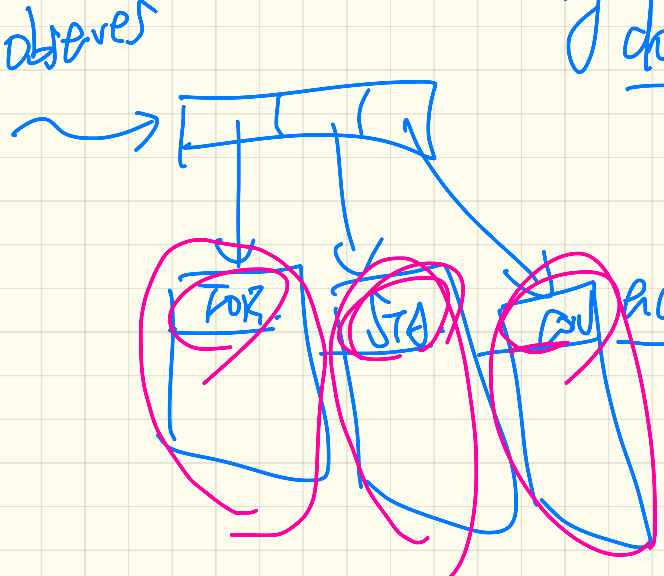
\+ STATISTICS

update f

update f

wd

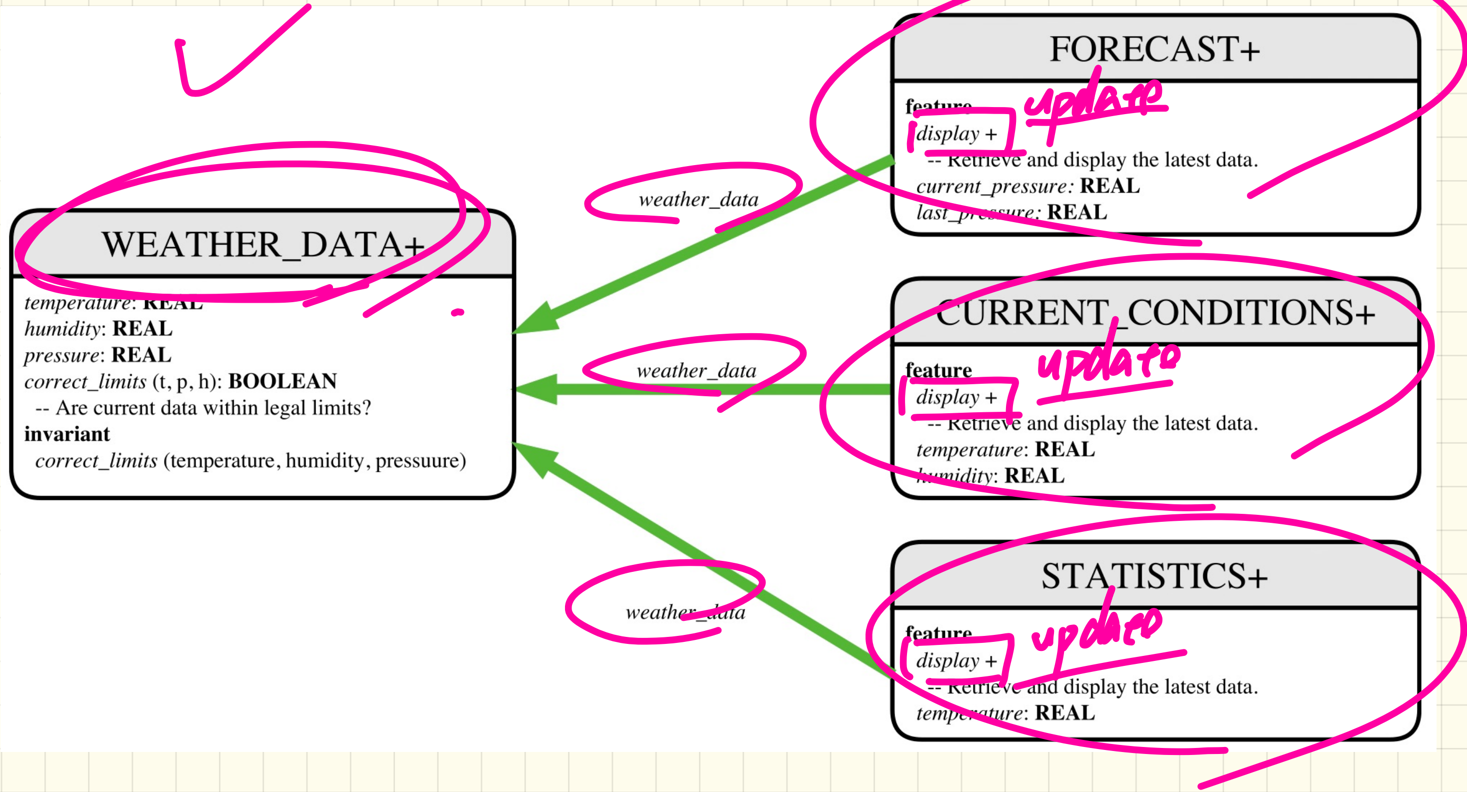
```
class SUBJECT
    observers : LIST[OBSERVER]

    notify
        do
            across observers is observer
            loop
                observer.update
            end
        end
```



LECTURE 21

FRIDAY NOVEMBER 22

Weather Station: 1st Design



✓

WEATHER_DATA+

temperature: **REAL**
humidity: **REAL**
pressure: **REAL**
correct_limits (t, p, h): **BOOLEAN**
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)

weather_data

weather_data

weather_data

FORECAST+

feature
display + update
-- Retrieve and display the latest data.
current_pressure: **REAL**
last_pressure: **REAL**

CURRENT_CONDITIONS+

feature
display + update
-- Retrieve and display the latest data.
temperature: **REAL**
humidity: **REAL**

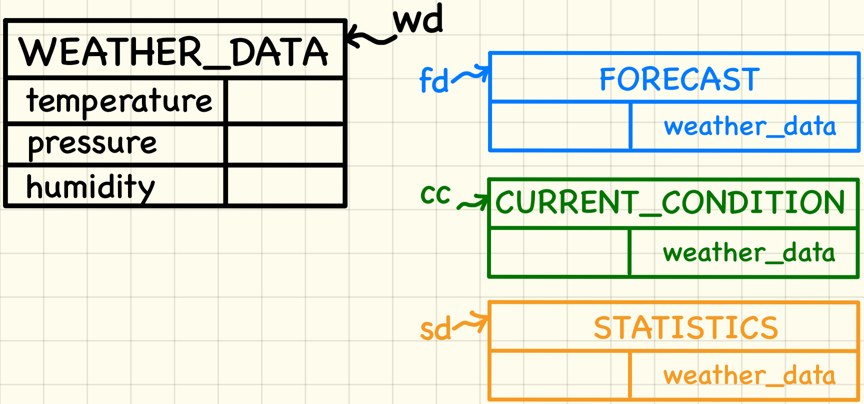
STATISTICS+

feature
display + update
-- Retrieve and display the latest data.
temperature: **REAL**

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
```

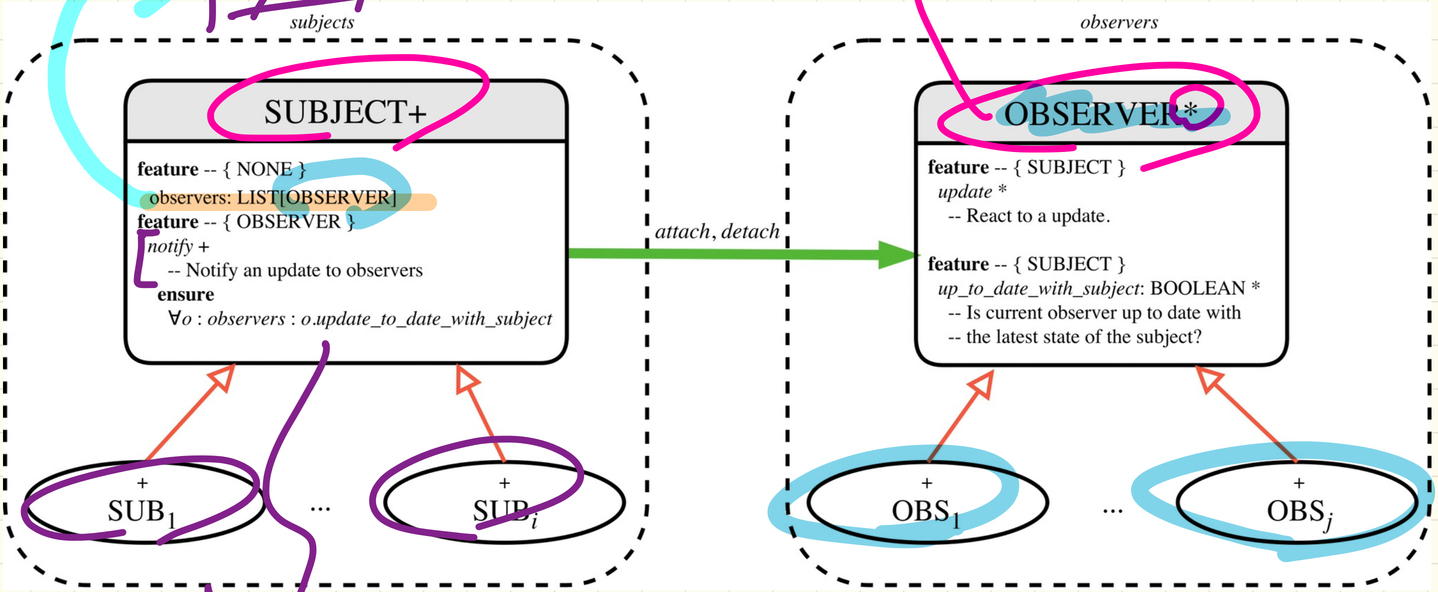
```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature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

The Observer Pattern

polymorphic collection

o : OBSERVER

~~create {OB} o. make~~



subjects

SUBJECT+

feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
$\forall o : observers : o.update\_to\_date\_with\_subject$

attach, detach

observers

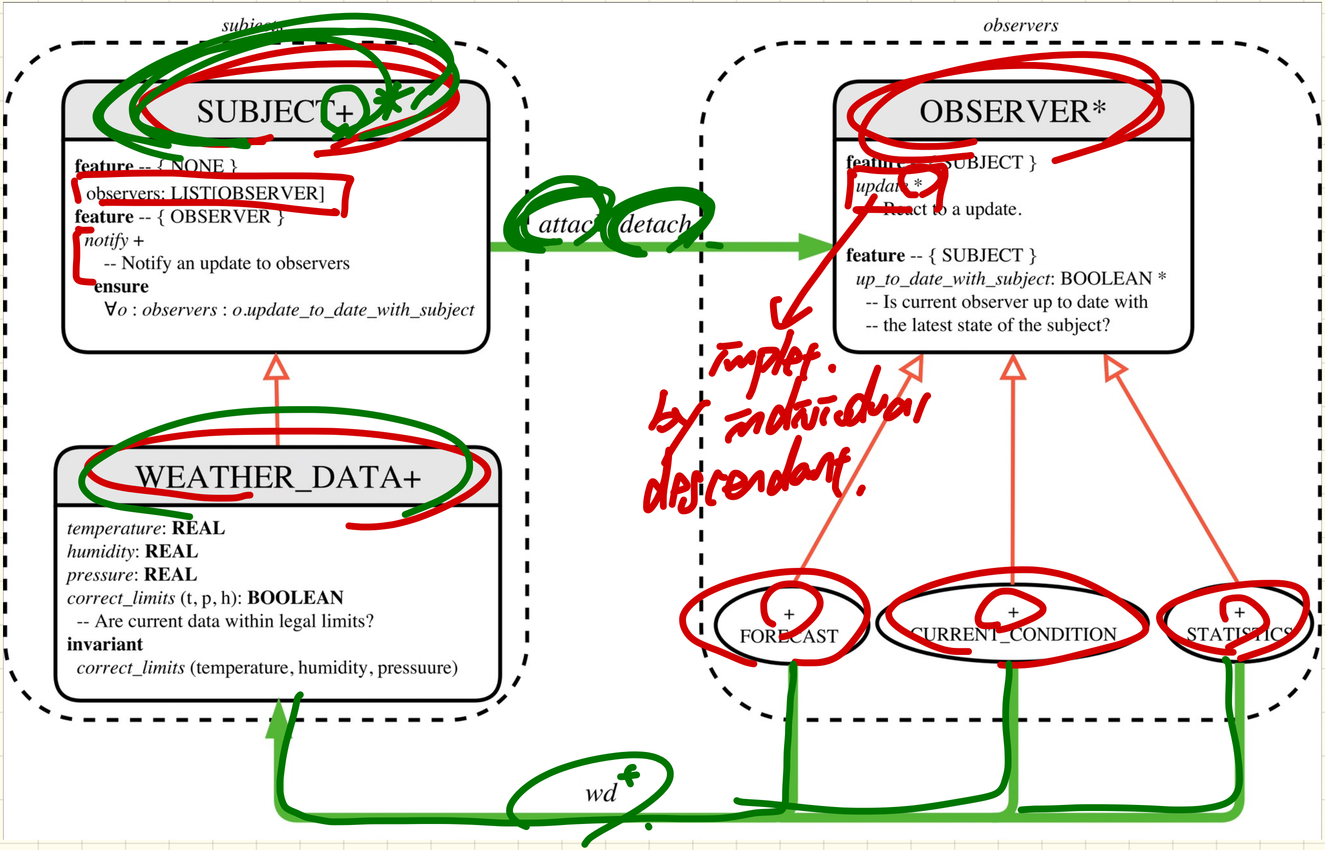
OBSERVER*

feature -- { SUBJECT }
update *
-- React to a update.

feature -- { SUBJECT }
up_to_date_with_subject: BOOLEAN *
-- Is current observer up to date with
-- the latest state of the subject?

timings of update is up to the subject.

Observer Pattern: Application to Weather Station



subject

SUBJECT+

feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
∀o : *observers* : *o.update_to_date_with_subject*

WEATHER_DATA+

temperature: **REAL**
humidity: **REAL**
pressure: **REAL**
correct_limits (t, p, h): **BOOLEAN**
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)

attach *detach*

observers

OBSERVER*

feature -- { SUBJECT }
update *
-- React to a update.

feature -- { SUBJECT }
up_to_date_with_subject: BOOLEAN *
-- Is current observer up to date with
-- the latest state of the subject?

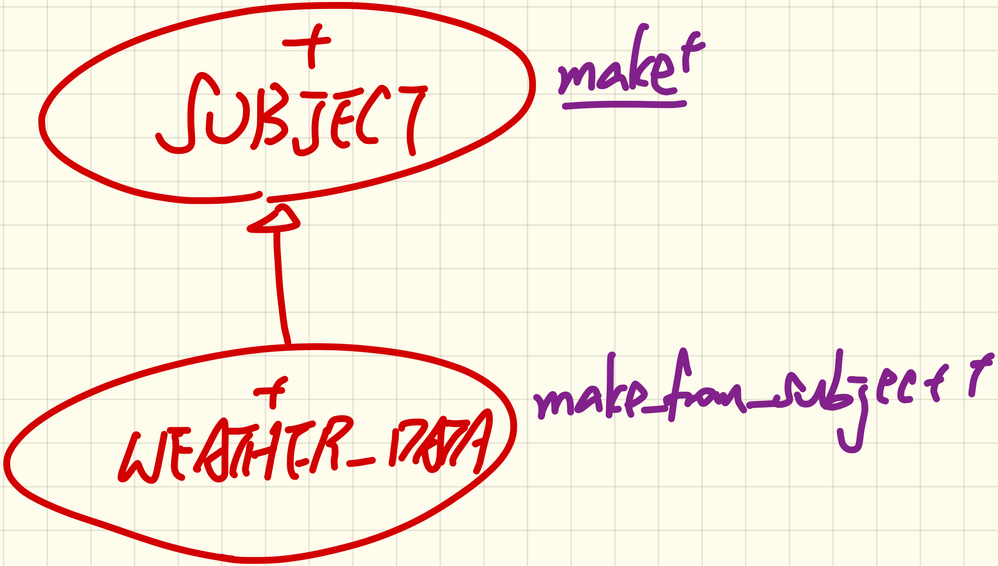
Implem. by individual descendant.

\+ FORECAST

\+ CURRENT_CONDITION

\+ STATISTICS

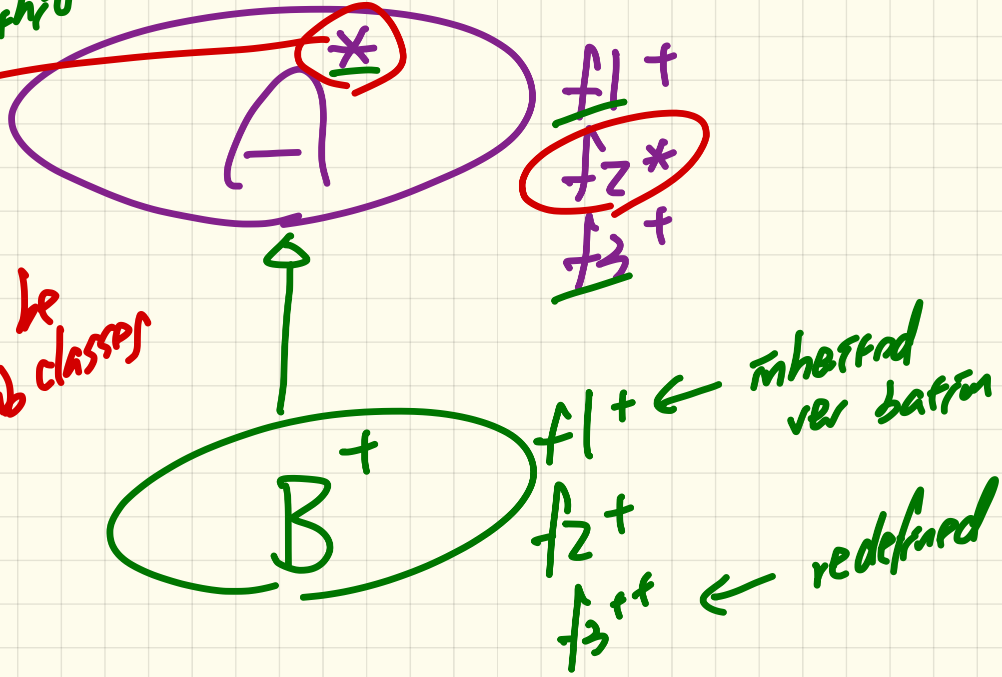
wd

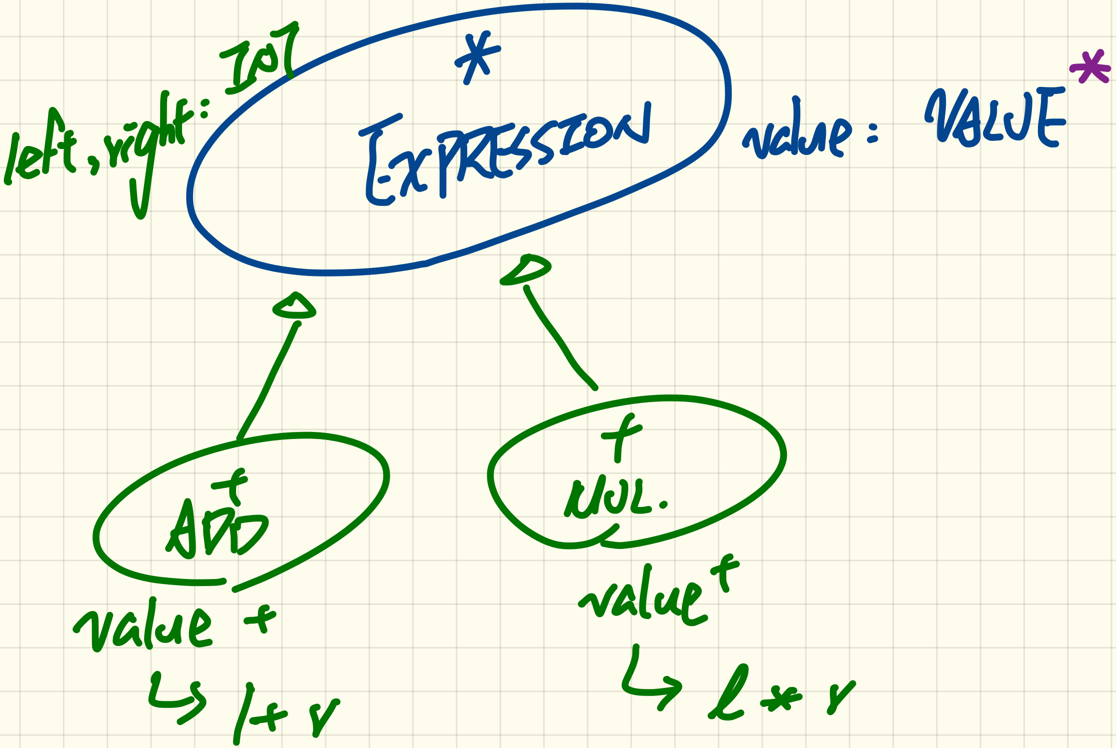

+
SUBJECT
make+
+
WEATHER_DATA
make_from_subject +

When to make a class deferred?

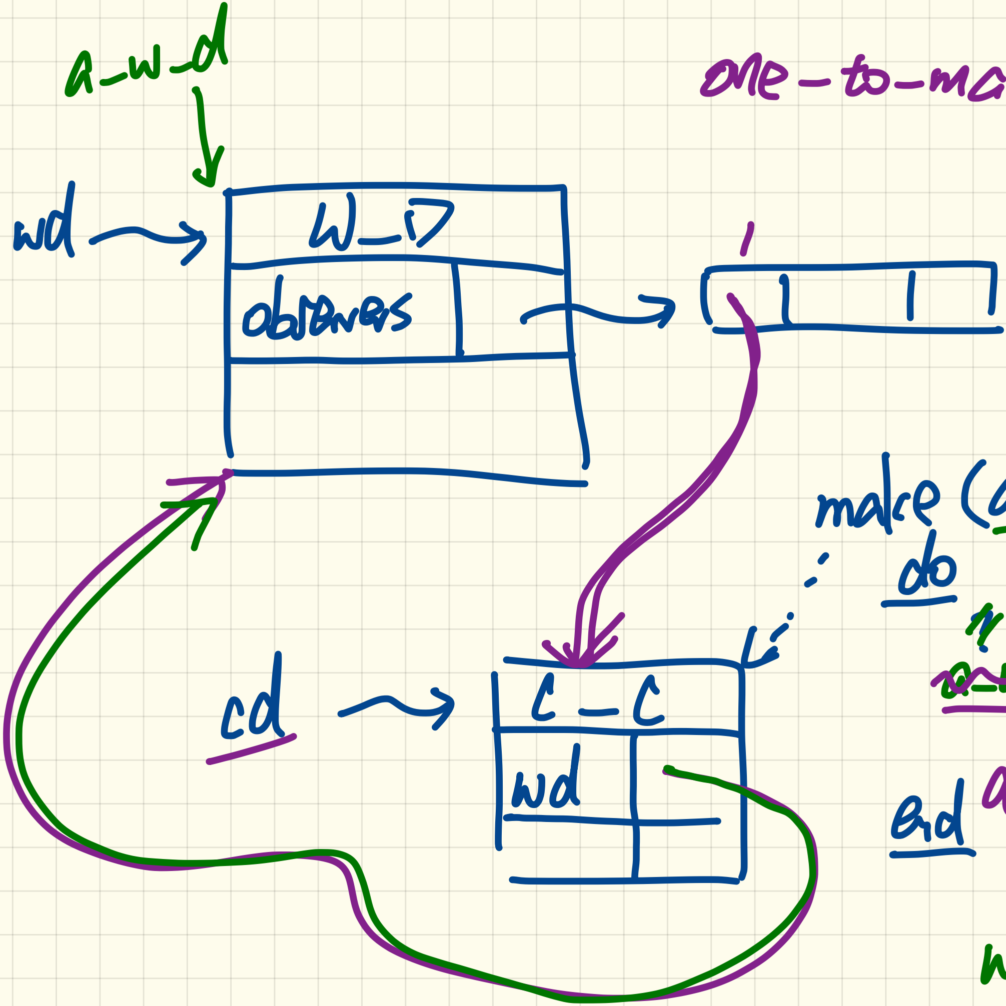
1. some features just cannot be implemented at this level

2. expecting: all deferred features to be imp. in sub classes




*
EXPRESSION
left, right:
value: VALUE*
+
ADD
value +
↳ l + r
+
MUL.
value+
↳ l * r

one-to-many



make (a_w_d: WD)

do

--

~~a_w_d.obs.extend(Current)~~

a_w_d.attach(Current)

end

wd := a_w_d

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
```

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 (9, 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
```

cc.weather_data := wd
wd.attach (cc)

across observers
∀ ob
ob.update

```
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
```

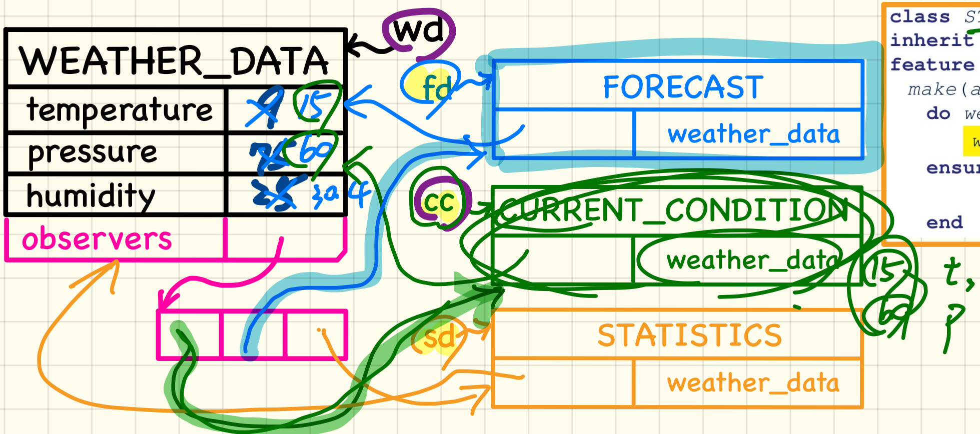
p

```
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
```

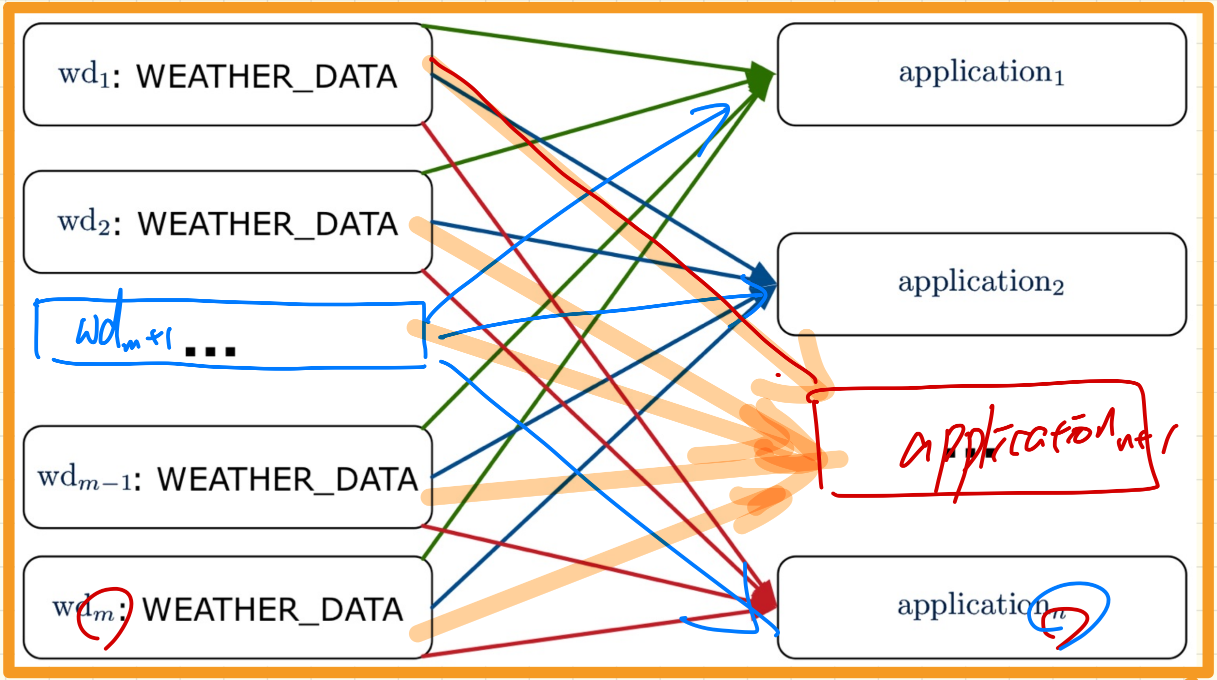
t, P

```
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
```

t



Multiple Subjects vs. Multiple Observers: Observer Pattern

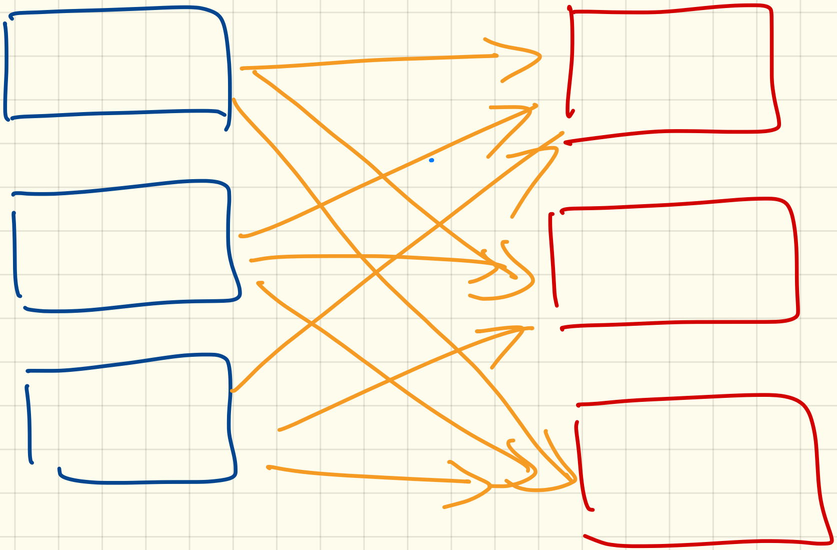


Q1. Overall **Complexity**? $O(m \cdot n)$. → $O(m)$ or $O(n)$

Q2. **Complexity** of adding a new **subject**? $O(n)$ → $O(1)$

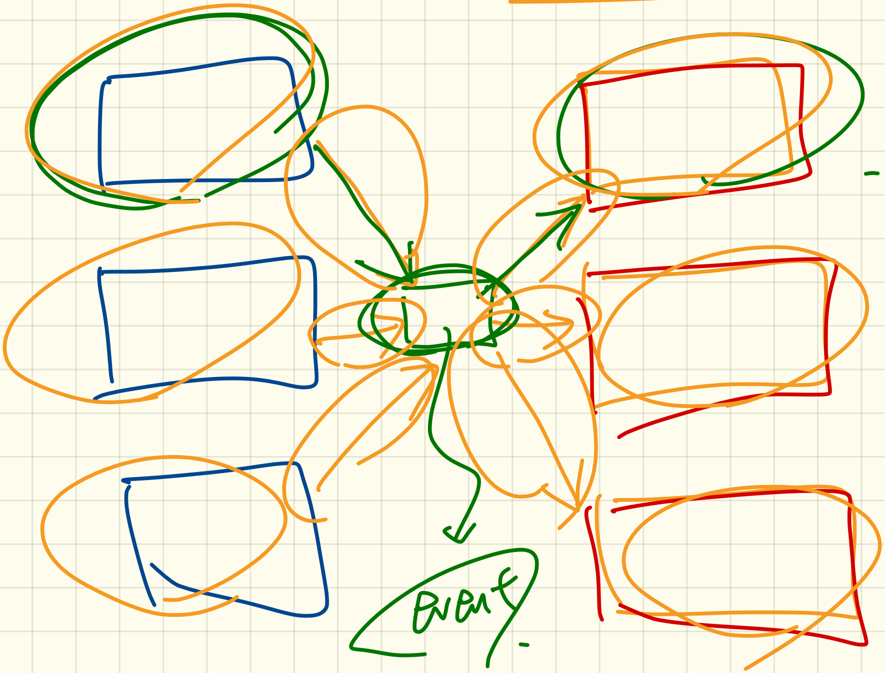
Q3. **Complexity** of adding a new **observer**? $O(m)$ → $O(1)$

objects

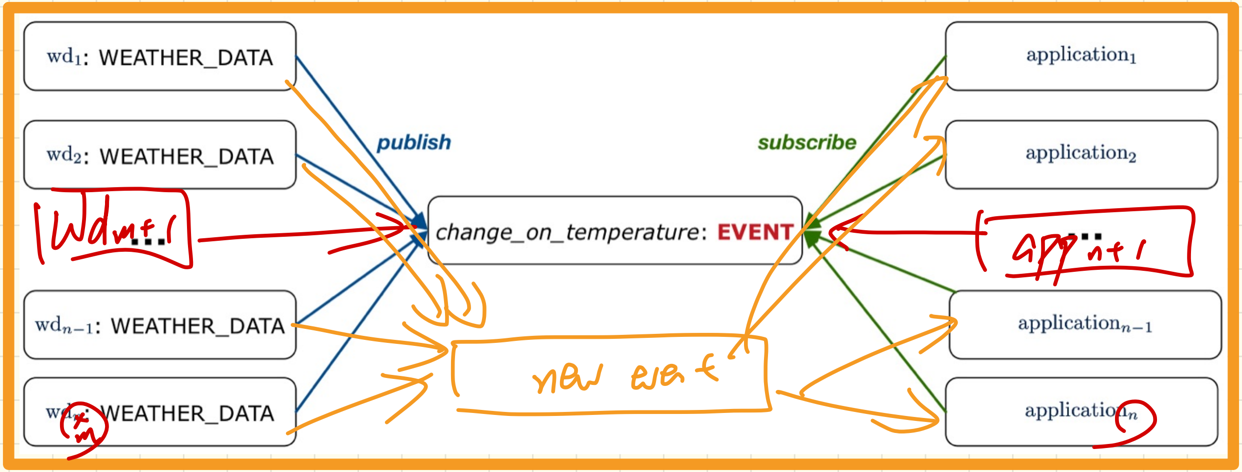


event-driven

$$O(m^3 + n^3)$$



Multiple **Subjects** vs. Multiple **Observers**: Event-Driven Design



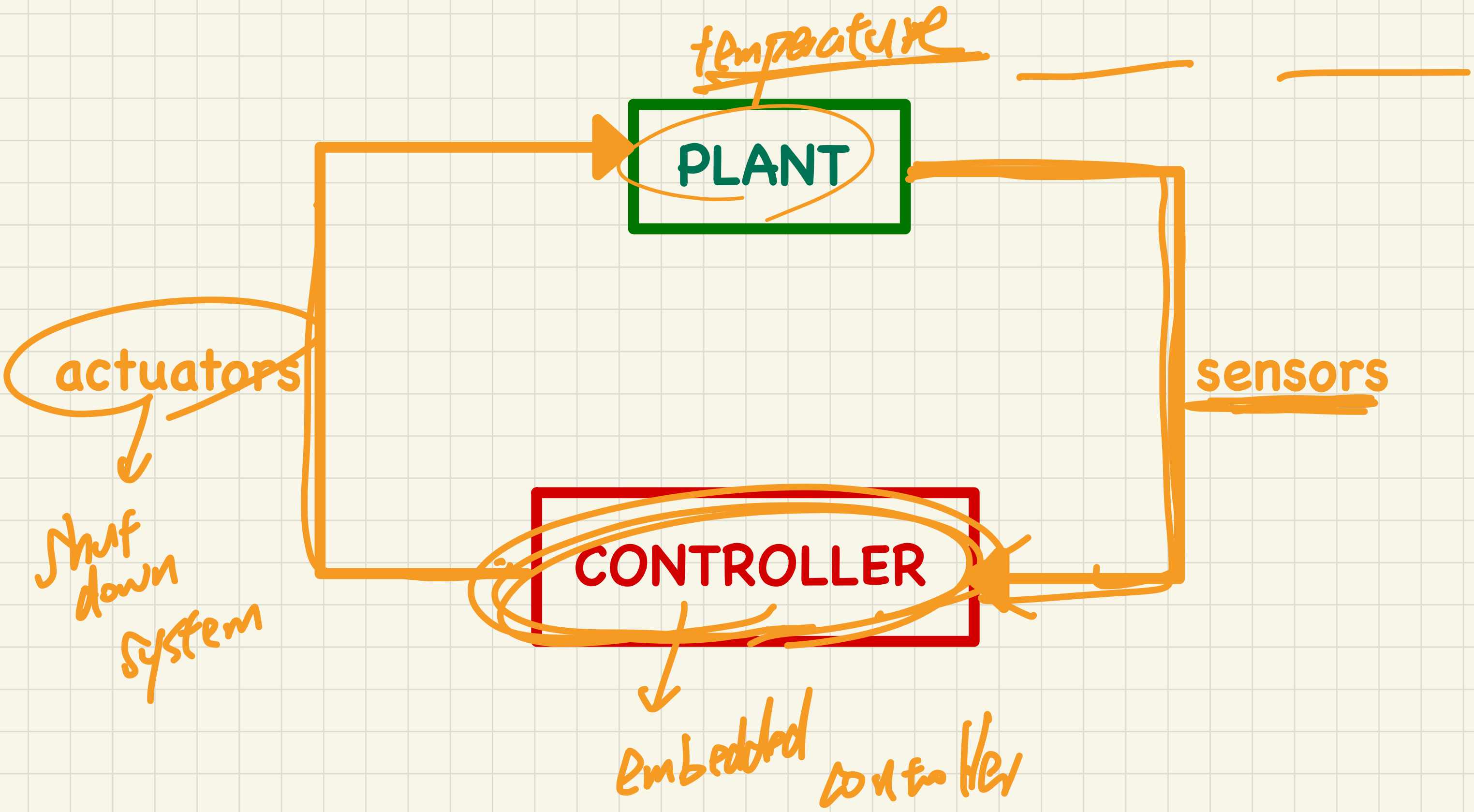
Q1. Overall **Complexity**? O(m + n)

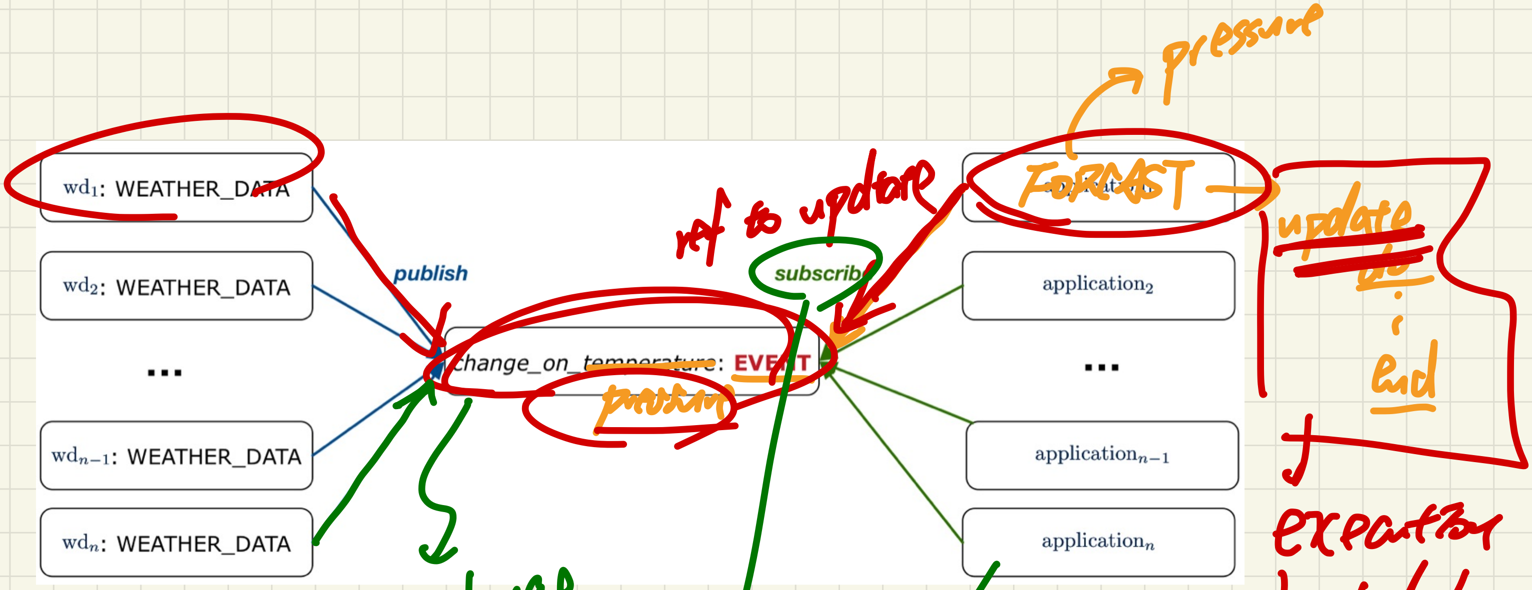
Q2. **Complexity** of adding a new **subject**?

Q3. **Complexity** of adding a new **observer**? O(1)

Q4. **Complexity** of adding a new **event type**?

Cyber-Physical Systems: Plant, Sensors, Controller, Actuators





when a change is published, call the update that's stored.

store ref. to update command for delayed execution

execution should be delayed until some wd publishes changes.

```
void   print () {
    print ( "Hello" ) ;
}
```

→ execution = * print()

{

→ execution

Event-Driven Design in Java

```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();
  } }
```

```java
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); } }
```

```java
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);
  }
}
```

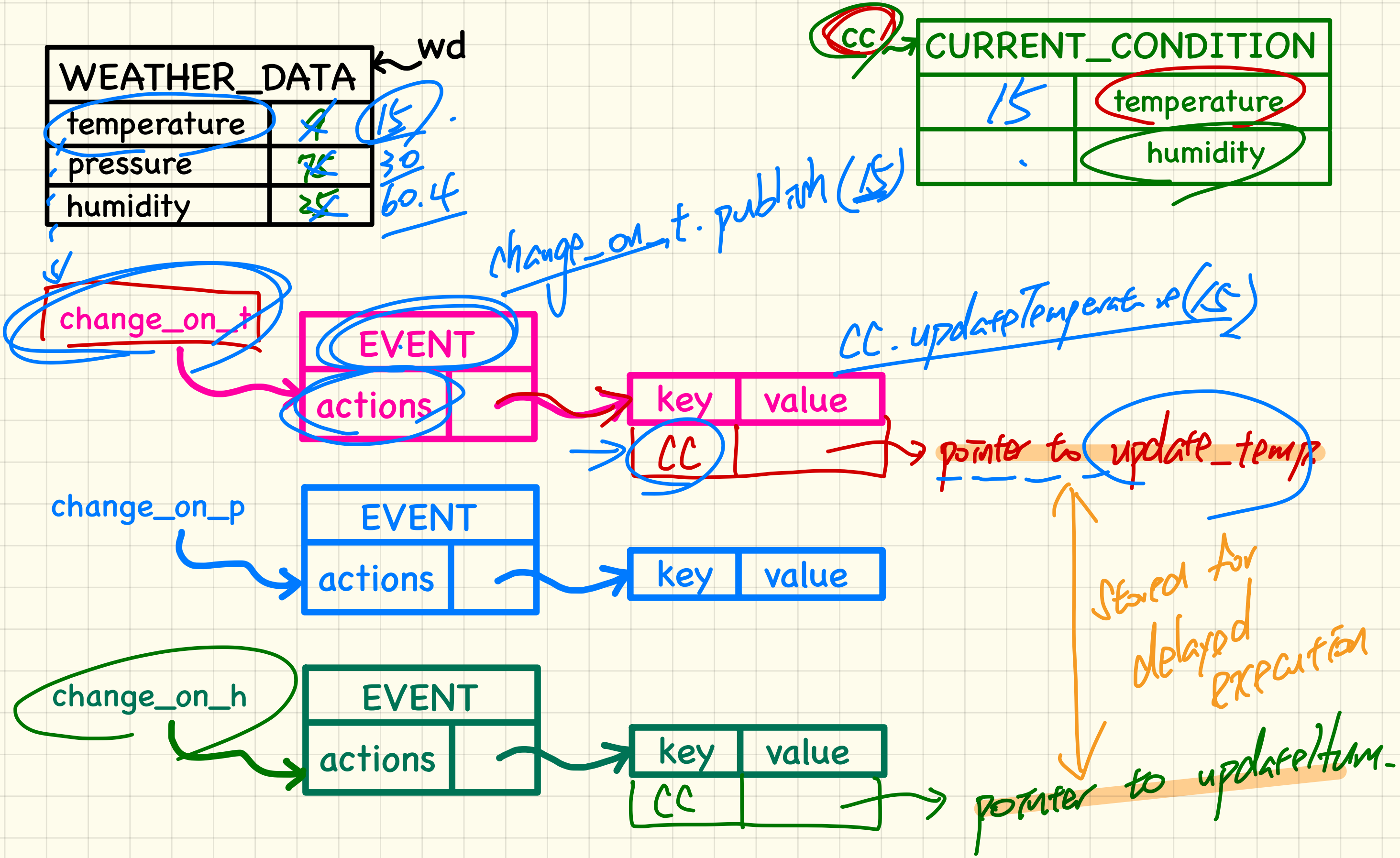
15 60 30.4

15

Context

```java
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) { }
    }
  }
}
```

Event-Driven Design in Java: Runtime



LECTURE 22

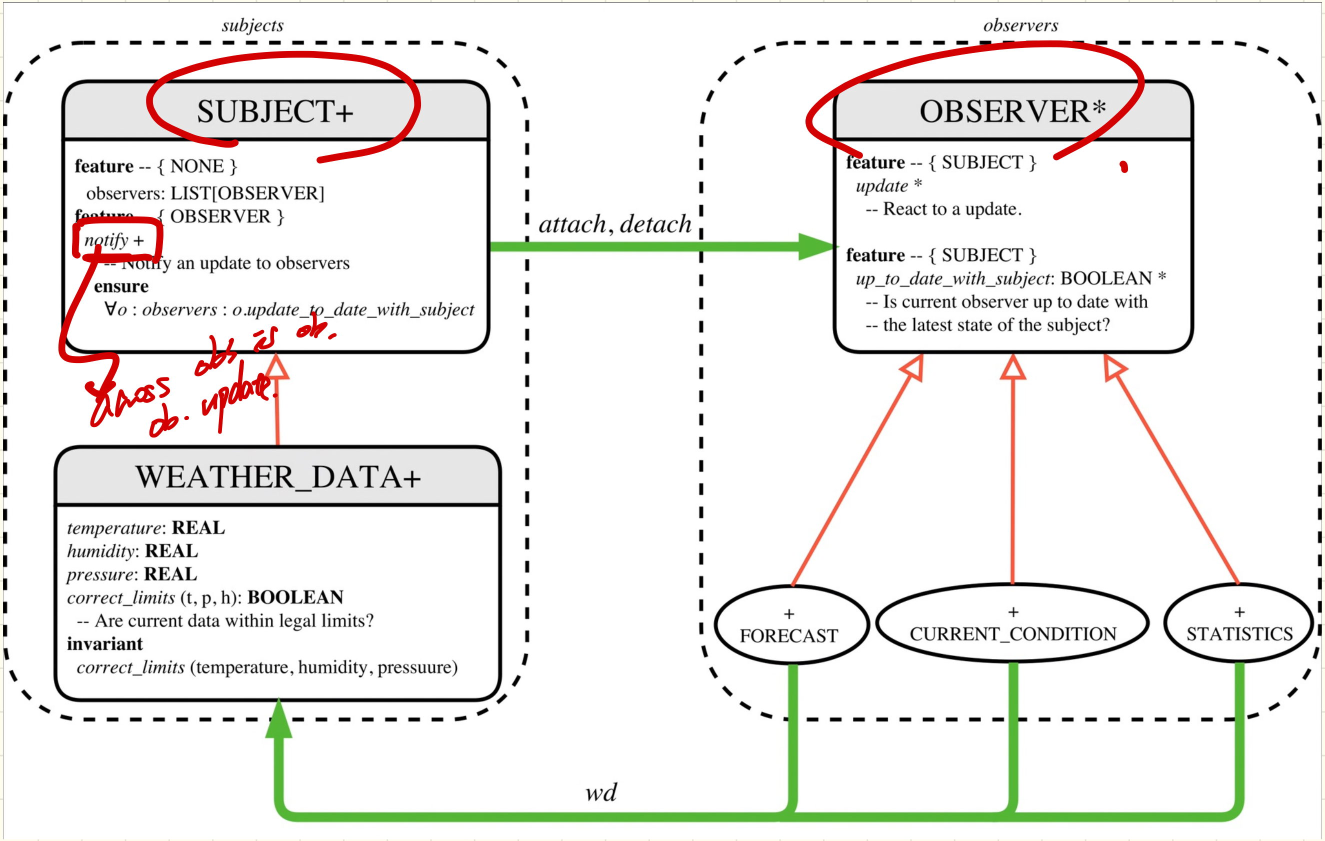
TUESDAY NOVEMBER 26

- REVIEW SESSIONS FOR EXAM
  SURVEY ON MOODLE

- MAKE-UP LECTURES:

  NOV. 15 } RECORDINGS
  NOV. 22 }

Observer Pattern: Application to Weather Station



subjects

SUBJECT+

feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
∀*o* : *observers* : *o.update_to_date_with_subject*

across obs is ob.
ob. update.

WEATHER_DATA+

temperature: **REAL**
humidity: **REAL**
pressure: **REAL**
correct_limits (t, p, h): **BOOLEAN**
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)

attach, *detach*

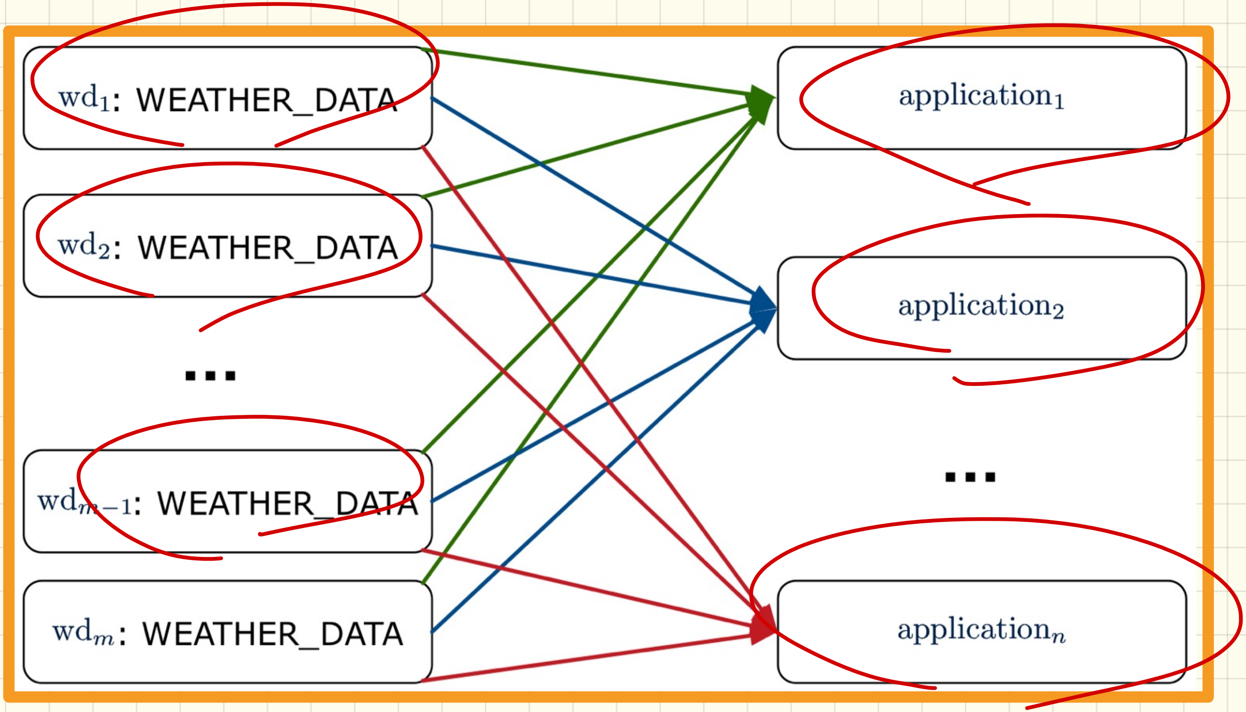
observers

OBSERVER*

feature -- { SUBJECT }
update *
-- React to a update.

feature -- { SUBJECT }
up_to_date_with_subject: BOOLEAN *
-- Is current observer up to date with
-- the latest state of the subject?

Multiple Subjects vs. Multiple Observers: Observer Pattern

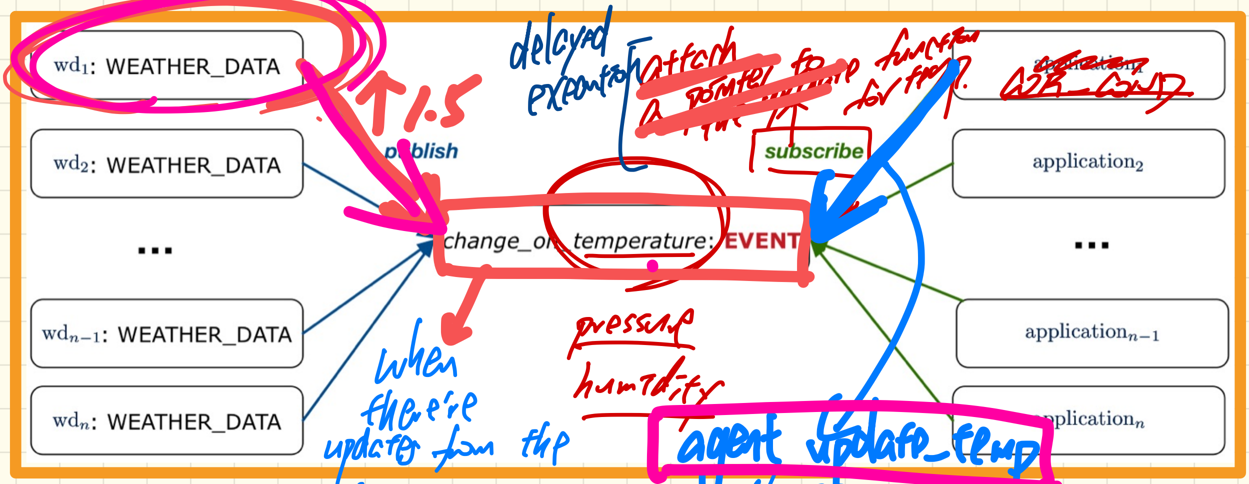


Q1. Overall **Complexity**?

Q2. **Complexity** of adding a new **subject**?

Q3. **Complexity** of adding a new **observer**?

Multiple **Subjects** vs. Multiple **Observers**: Event-Driven Design



when there're updates from the subject, the event will execute the pointer to functions stored previously.

Q1. Overall **Complexity**?

Q2. **Complexity** of adding a new **subject**?

Q3. **Complexity** of adding a new **observer**?

Q4. **Complexity** of adding a new **event type**?

① update_tempeatre

↳ 1. does not return anything

2. execute ing of u_t. right away.

② $\boxed{\text{agent update\_temperature}}$

PROCEDURE (for delayed execution).

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
```

EVENT

humidity

```
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
```

15 60 30.4

(1) [15]

```
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
```

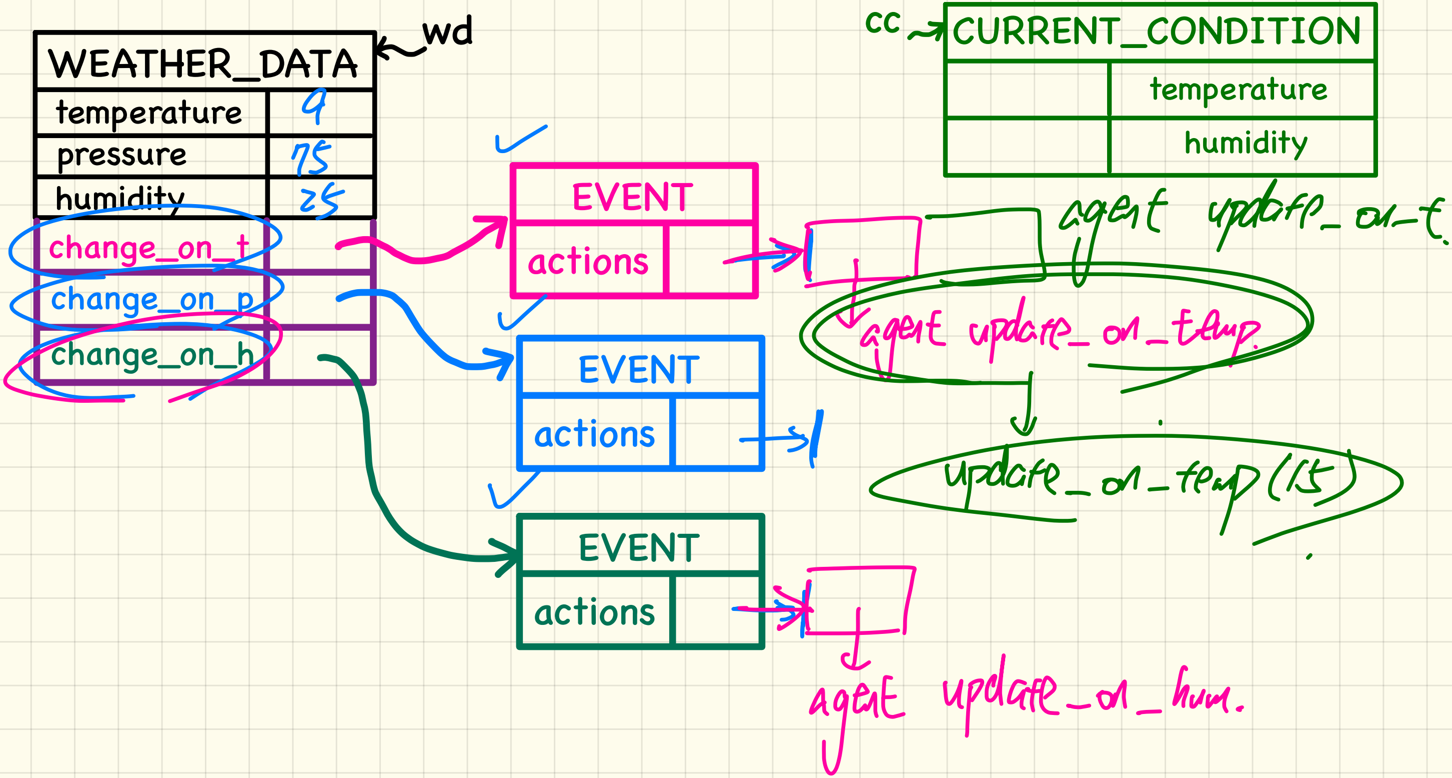
e1: EVENT[[REAL]]

e2: EVENT[[S, I]]

[15]

A PROCEDURE [15]

Event-Driven Design in Eiffel: Runtime



$\boxed{x > 3}$

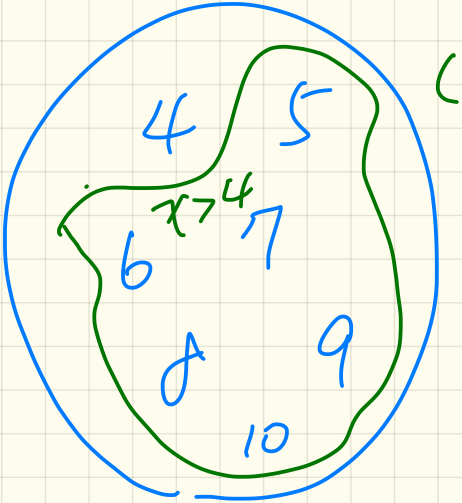
↳ allows more value (i.e. 4)

$\underline{x > 4}$

↳ stronger

↓

does not allow 4.

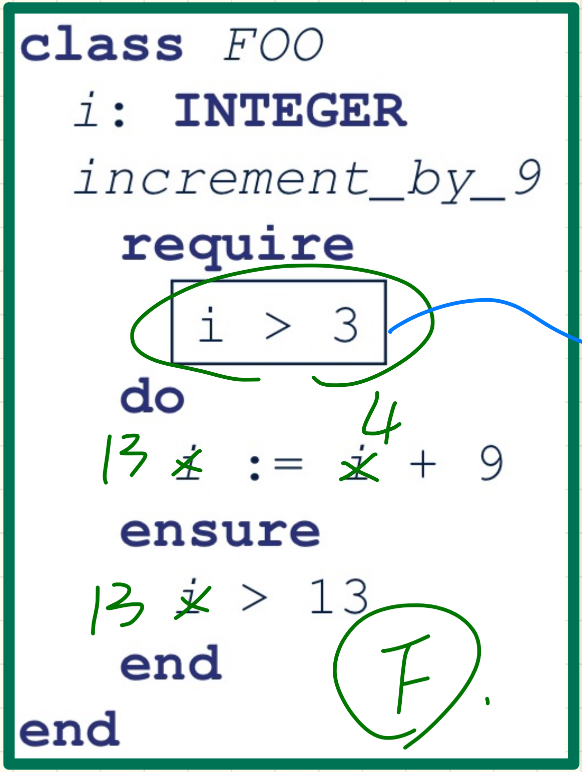


$\boxed{x > 4} \Rightarrow \boxed{x > 3}$

antecedent stronger

consequence weaker.

Program Correctness: Example (1)



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

4

13 ~~i~~ := ~~i~~ + 9

13 ~~i~~ > 13

F

i = 4

too weak

↳ it allows i = 4, which will cause postcondition violation.

Program Correctness: Example (2)

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

6
7
8

$i > 5 \Rightarrow i > 4$

Can any input value allowed by the precondition cause a postcondition?

No.

alternatively: $i > 4$

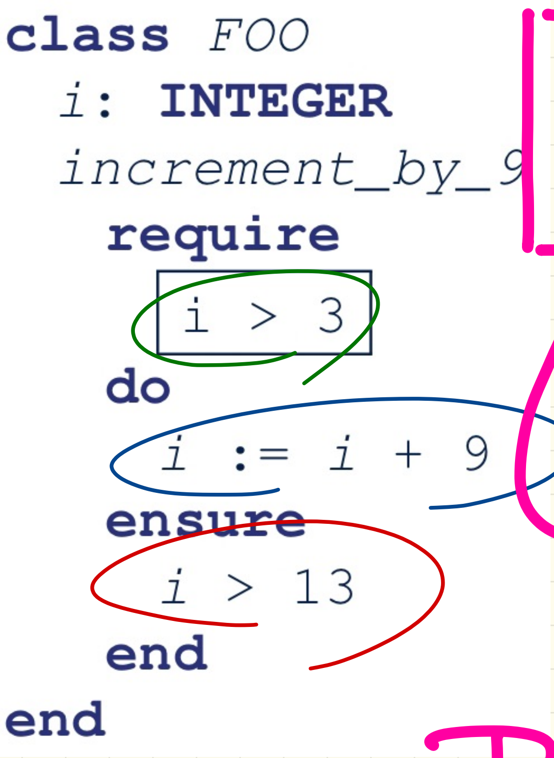
Hoare Triple

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

precondition

program/implementation

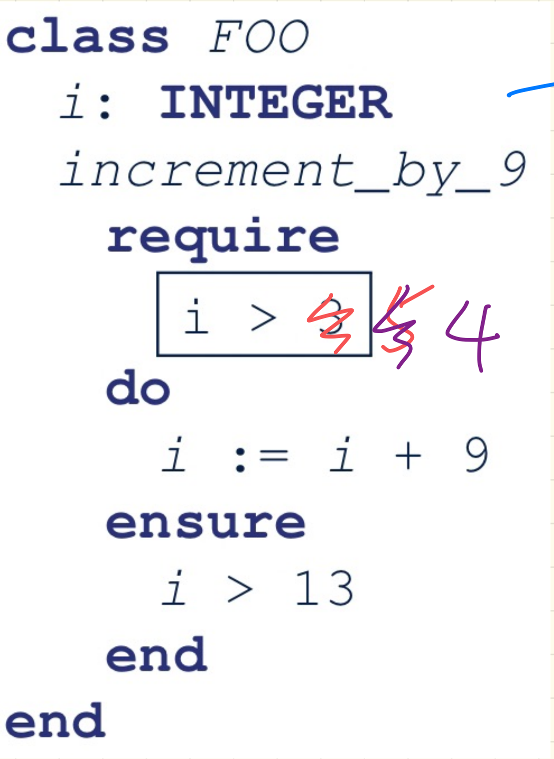
postcondition



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

Boolean expression

Starting in a state that satisfies the precondition, if we execute imp., then it will
① terminate
② establish postcond.



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

~~3~~ ~~5~~ 4

$\{i > 4\}$ $i := i + 9$ $\{i > 13\}$

True

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

→ False ∵ $i = 4$ will be allowed by precondition.

$\{i > 5\}$ $i := i + 9$ $\{i > 13\}$

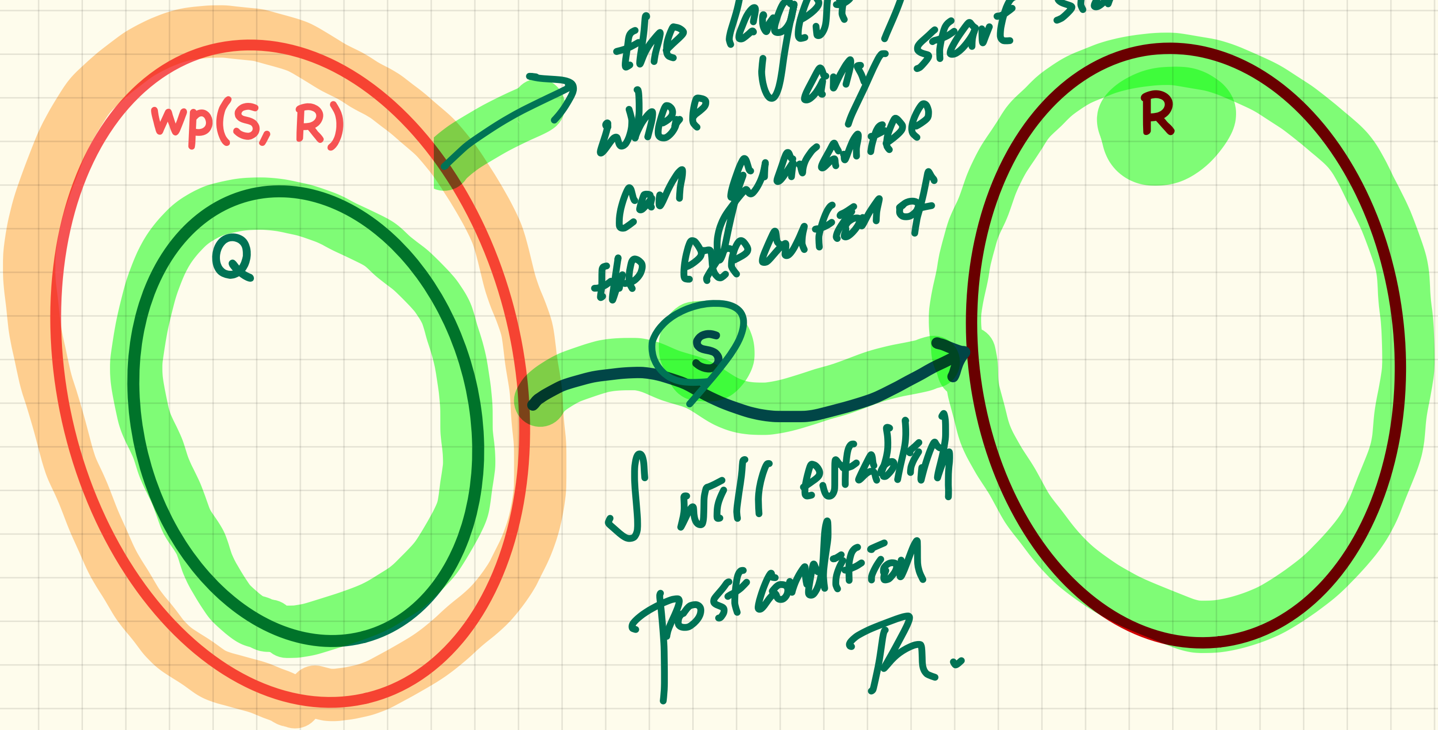
→ True ∵ all values allowed by $i > 5$ will establish $i > 13$ after executing $i := i + 9$.

LECTURE 23

THURSDAY NOVEMBER 28

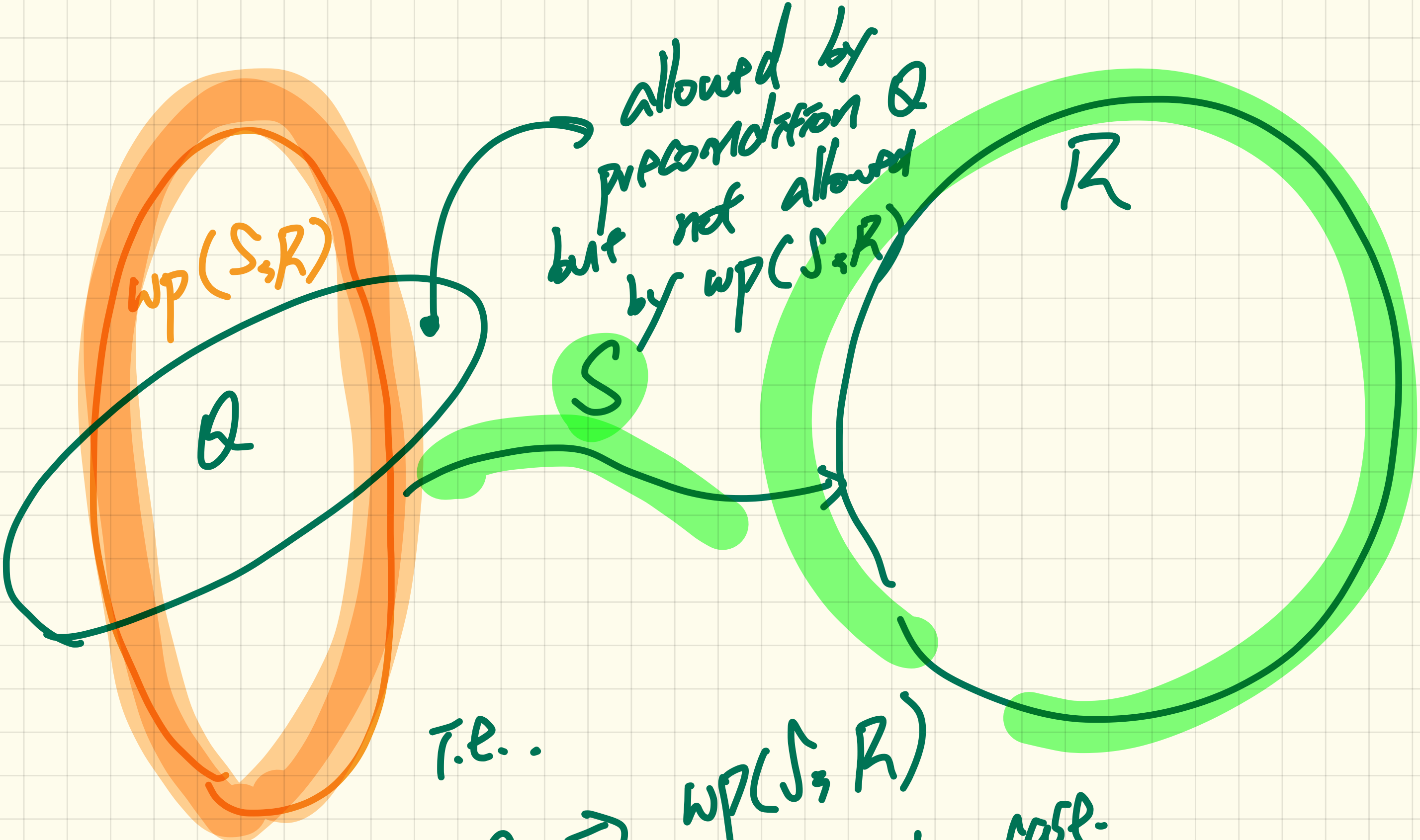
Hoare Triple as a Predicate

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



the largest possible set where any start state from here can guarantee the execution of

S will establish postcondition R.



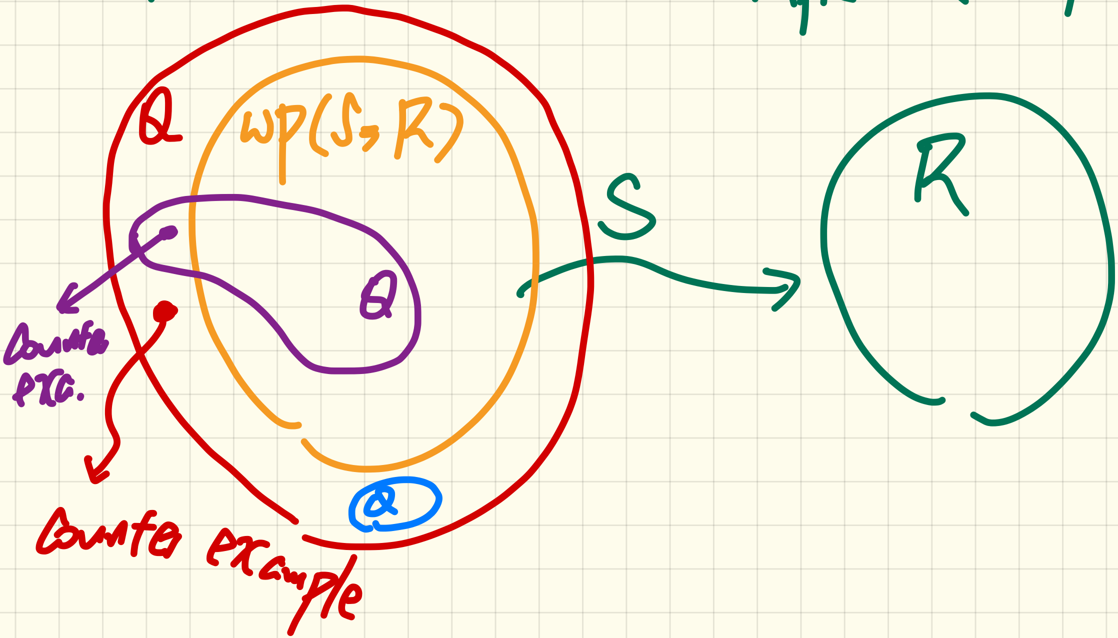
allowed by precondition Q but not allowed by wp(S;R)

i.e..

$Q \Rightarrow wp(S;R)$

is not the case.

How can a Hoare Triple be false?



Program Correctness: Revisiting Example (1)

the resp.

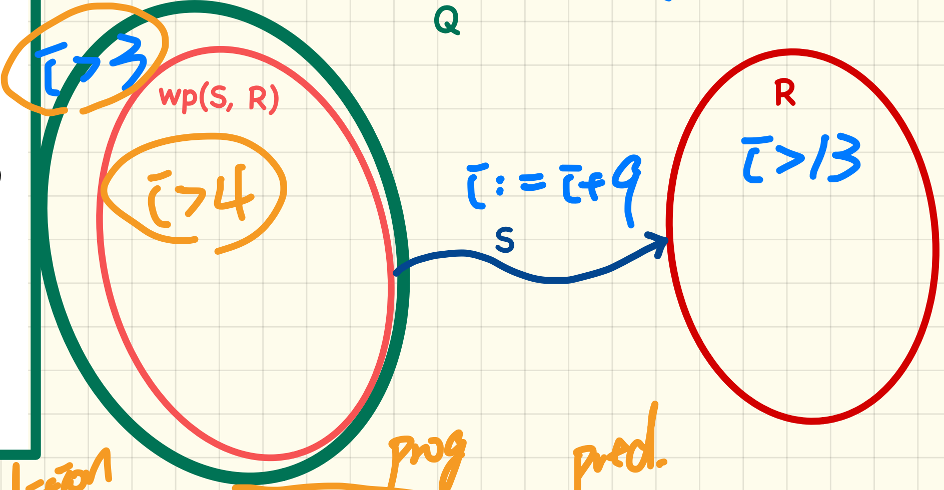
$i > 3 \Rightarrow i > 4$? not

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

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

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

Q



weakest precondition

prog

pred.

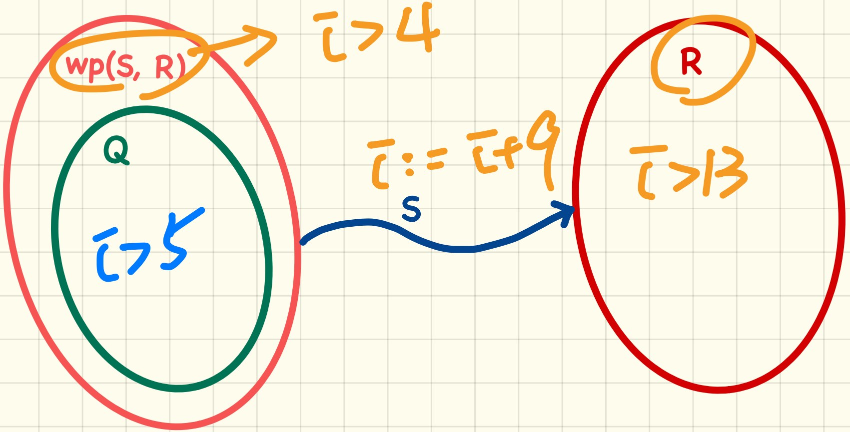
$$wp(\boxed{i := i + 9}, \boxed{i > 13}) = i > 4$$

Program Correctness: Revisiting Example (2)

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

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

$\{i > 5\}\ i := i + 9\ \{i > 13\}$



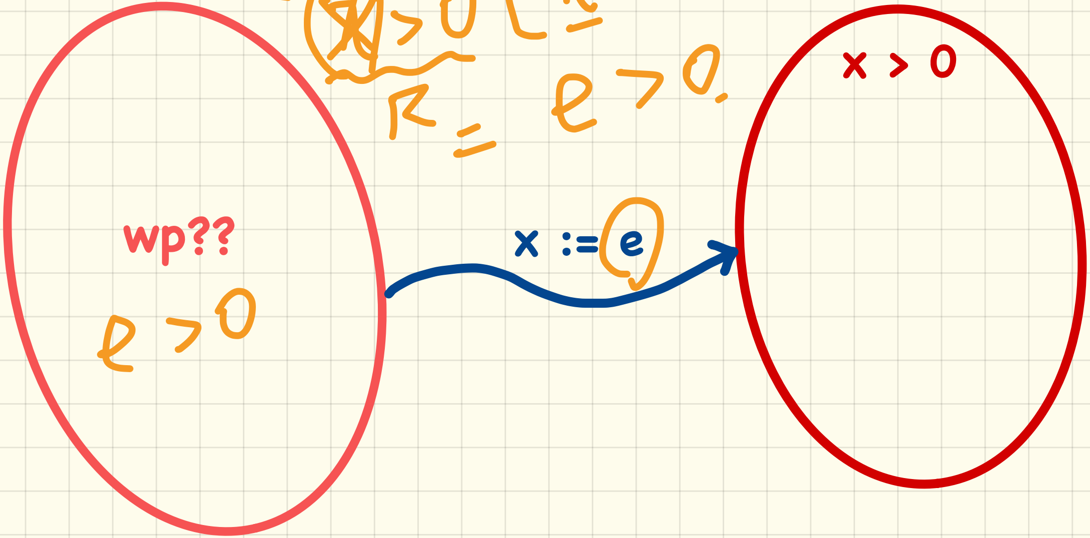
$wp(\ i := i + 9\ ,\ i > 13\ ) = i > 4$

Rules of Weakest Precondition: Assignment

$$wp(x := e, \boxed{R}) = R[x := e]$$

$$e\ \underbrace{\cancel{x} > 0}_{R}\ [\underline{x} := e]$$

$$= e > 0$$



Correctness of Programs: Assignment (1)

What is the weakest precondition for a program `x := x + 1` to establish the postcondition $x > x_0$?

$$\{??\}\ \texttt{x := x + 1}\ \{x > x_0\}$$

$$\boxed{wp(x := \overset{e}{x+1},\ x > x_0)}$$

$$= \{\text{wp rule for assign.}\}$$

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

$$= \cancel{x_0}+1 > \cancel{x_0}$$

$$\boxed{\text{True}}$$

→ this program works for any precondition.

Correctness of Programs: Assignment (2)

What is the weakest precondition for a program `x := x + 1` to establish the postcondition $x > x_0$?

$x \geq 22$

$\{??\}$ `x := x + 1` $\{x = 23\}$

Is this program correct?

1. Calculate $wp(x := x+1, x = 23)$

$= \{$ wp rule for assign. $\}$

$x = 23\,[x := x+1] = x+1 = 23$

$x = 22$

$\shortparallel$

given precond.

2. Prove $x \geq 22 \Rightarrow$ ~~$wp(x := x+1, x = 23)$~~ $x = 22$

wp

not the case e.g. $x = 23$

Rules of Weakest Precondition: Conditionals

wp(if B then S1 else S2 end, R)

$B \Rightarrow wp(S_1, R)$

$\vee$

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

vs.

$B \Rightarrow wp(S_1, R)$

$\wedge$

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

??

(x, y)

$(-4, 1)$

$x - 1 \geq 0$ $x + 1 \geq 0$

Consider:

wp(if y > 0 then x := x + 1 else x := x - 1 end, x >= 0)

True

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

$x \geq -1$

$\vee$ F

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

$x \geq 1$

T

T

$x := -4$

$y := 1$

$P \wedge T \equiv P \quad P \vee T \equiv T$

wp(**if** y > 0 **then** x := x + 1 **else** x := x - 1 **end**, x >= 0)

-4

$x \geq -1$

T

Try:

$(x, y) = (-4, 1)$

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

$x \geq -1 \quad -4$

$\wedge$

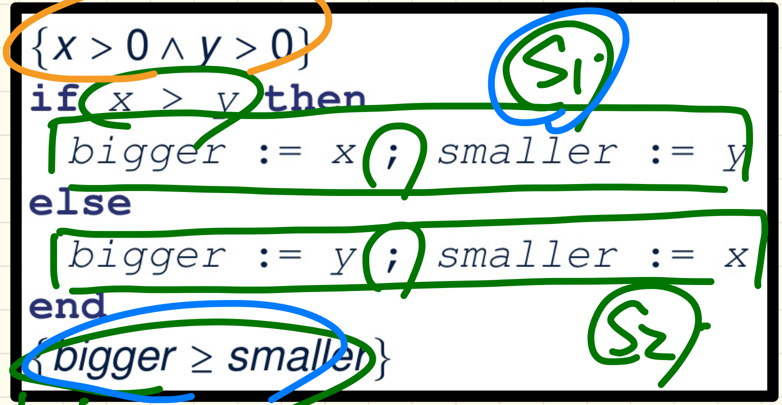
$\neg(y > 0) \Rightarrow wp(x := x - 1, x \geq 0)$

F

T

Correctness of Programs: 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}
```

S_1

S_2

2. Prove or disprove:

$x > 0 \land y > 0$
$\Rightarrow wp.$

1. Calculate

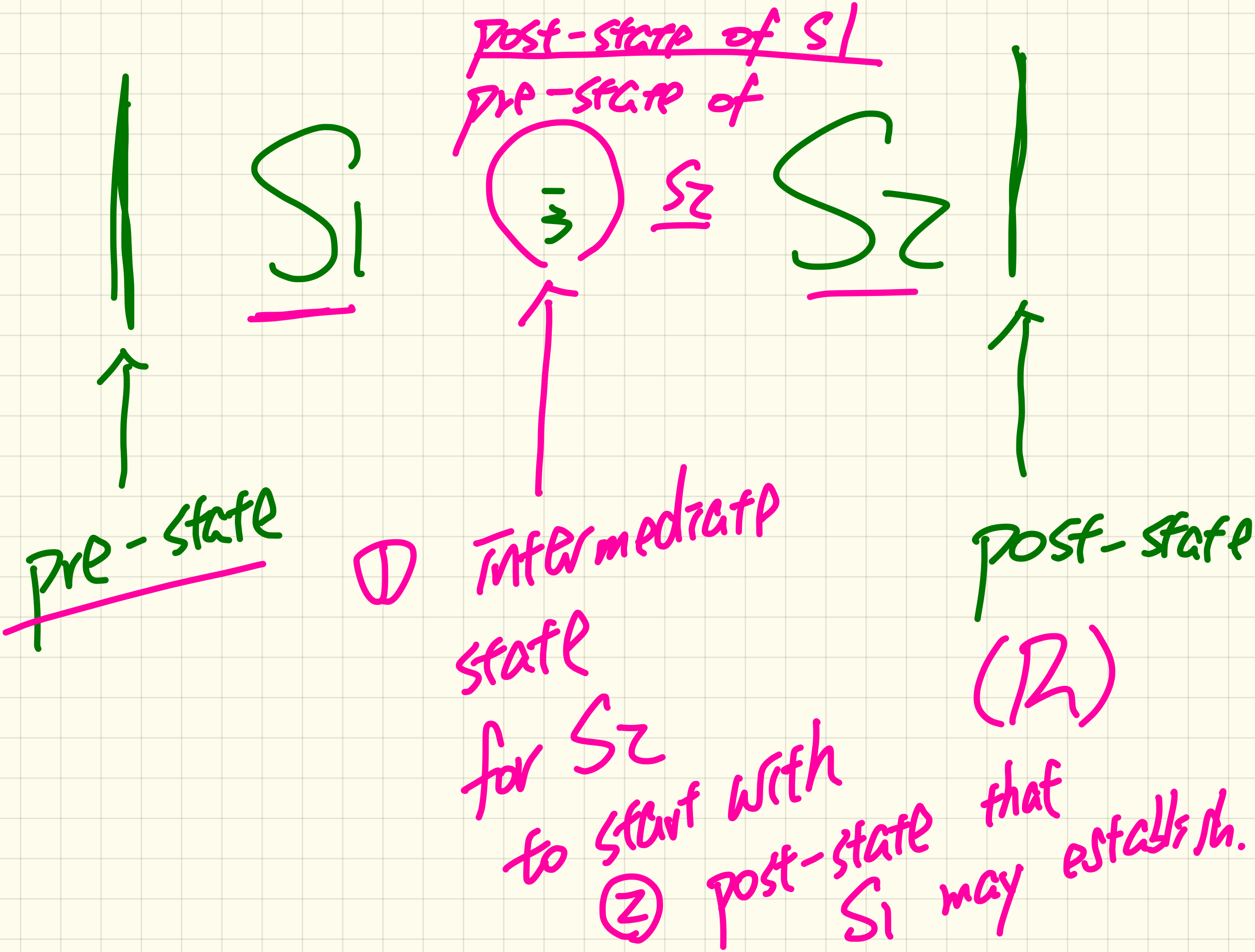
$wp(\text{if } x > y \text{ then } S_1 \text{ else } S_2 \text{ end} , b \geq s)$

$=$ { wp rule for conditionals }

$x > y \Rightarrow wp(S_1 , b \geq s)$

$\land$

$\neg(x > y) \Rightarrow wp(S_2 , b \geq s)$


post-state of S1
pre-state of
S1
S2
S2
pre-state
① intermediate state for S2 to start with
② post-state S1 may establish.
post-state
(R)

$$S_1 \; ; \; \underline{S_2} \;\Big|\; \underline{R}$$

should be "good" enough for S_2 to establish R

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

$$= wp(S_1 \; ; \; \underline{wp(S_2 \; ; \; R)})$$

$$wp(S_1 ; \underline{S_2} ; S_3 , R)$$

$$= wp(S_1 , wp(S_2 , wp(S_3 , R)))$$

Correctness of Programs: Sequential Composition

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

① Calculate $wp(\underline{tmp := x};\ \underline{x := y};\ \underline{y := tmp},\ x > y)$

$= \{ wp \text{ rule of } ; \}$

$wp(tmp := x,\ wp(\underline{x := y;\ y := tmp},\ x > y))$

$= \{ wp \text{ rule of } ; \}$

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

numbers

Swap x and y

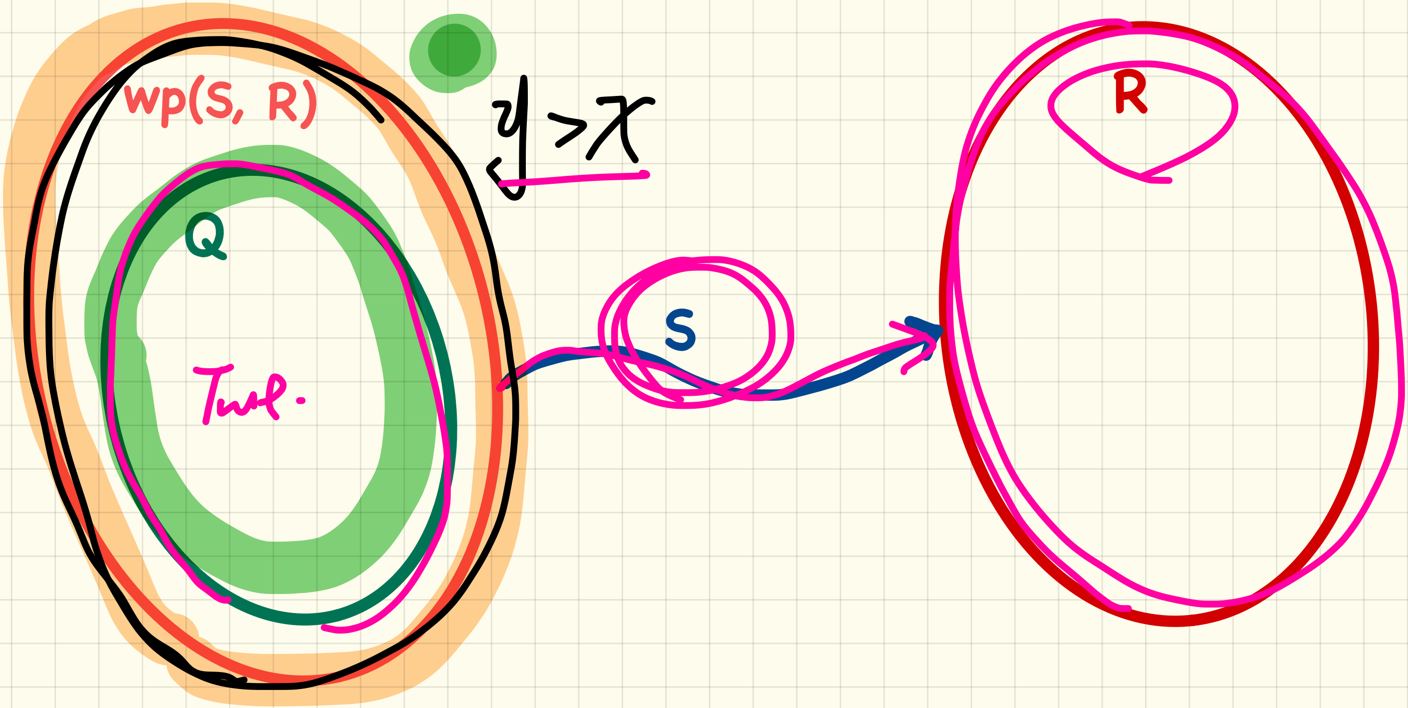
without using an intermediate variable.

LECTURE 24

TUESDAY DECEMBER 3

Hoare Triple as a Predicate

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



Rules of Weakest Precondition: Summary

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

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

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

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

$S_1\ \boxed{;}\ S_2 \mid R$

end for S_1
start for S_2

Correctness of Programs: Sequential Composition

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

① Calculate $wp(tmp := x;\ x := y;\ y := tmp,\ x > y)$

$= \{\text{wp rule of } ;\}$

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

$= \{\text{wp rule of } ;\}$

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

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

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

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

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

$y > x$

$x > y\,[y := tmp]$

$x > tmp\,[x := y]$

$y > tmp\,[tmp := x]$

②

$Q \Rightarrow wp$

$True \Rightarrow y > x$

C.E.

$y = x$

Proof Rules using
Weakest Precondition

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

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

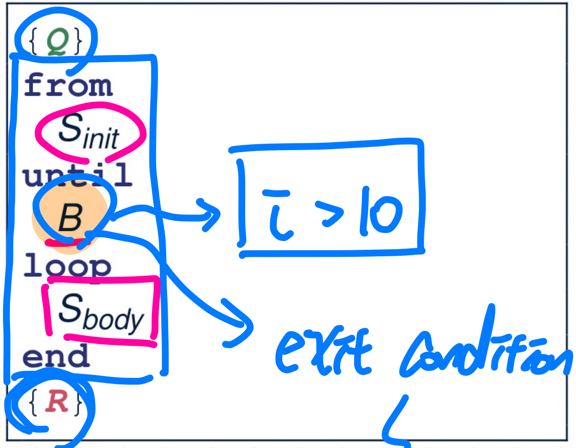
$$\{Q\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{end}\ \{R\}$$

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

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

loop.

Loops: Eiffel vs. Java

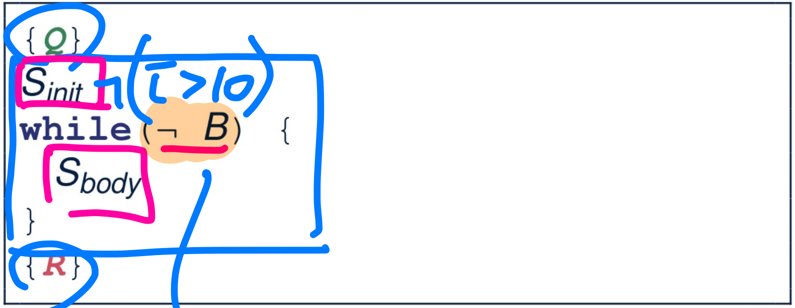


```
{Q}
from
  S_init
until
  B
loop
  S_body
end
{R}
```

$i > 10$

exit condition

as soon as $i > 10$ is true, exit.



```
{Q}
S_init
while (¬ B) {
  S_body
}
{R}
```

$\neg(i > 10)$

as long as $\neg(i > 10)$ is the case, stay.

Contracts of Loops

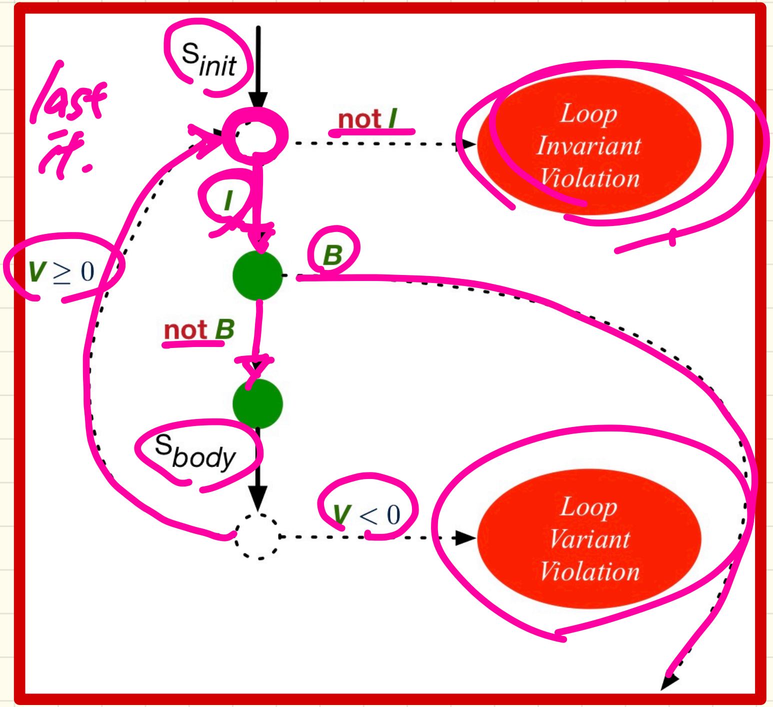
I checked: before 1st it.
before 2nd it.
⋮
after last it.

V checked: after 1st
after 2nd
⋮
after last it.

Syntax

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

Runtime Checks



Contracts of Loops: Example

Syntax

of times LI is checked: 6

# of time LV is checked: 5

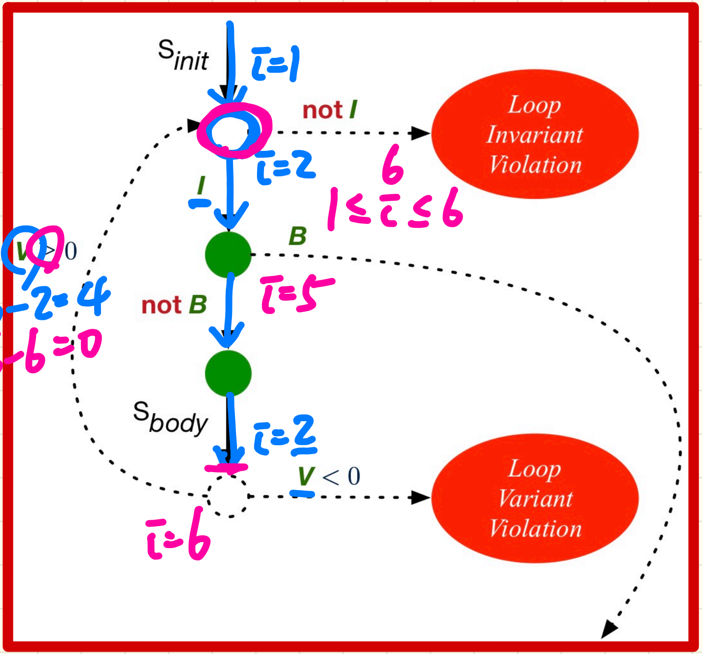
```
test
  local
    i: INTEGER
  do
    from
      i := 1
    invariant
      1 <= i and i <= 6
    until
      i > 5
    loop
      io.put_string ("iteration " + i.out)
      i := i + 1
    variant
      6 - i
    end
end
```

6>5 exit.

1
2
3
4
5

LI when exiting the loop should imply postcondition.

Runtime Checks



6-2=4
6-6=0

Contracts of Loops: Violations

Syntax

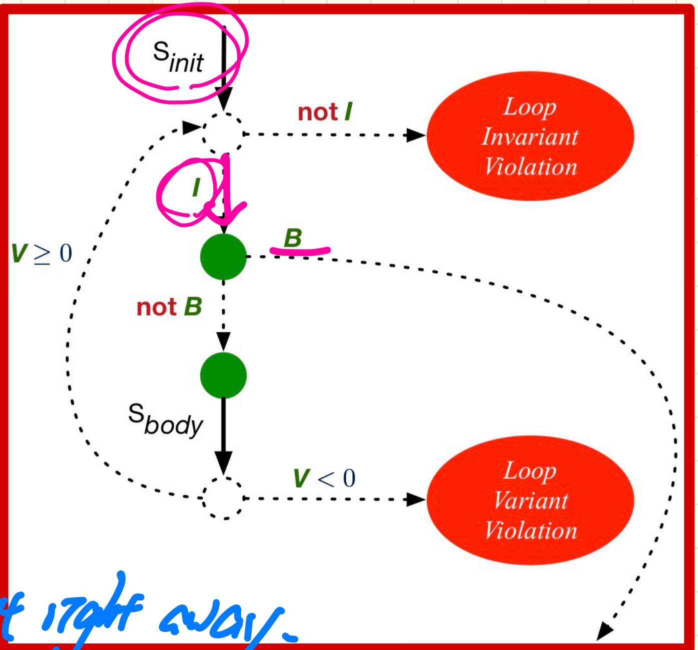
```
test
  local
    i: INTEGER
  do
    from
      i := 1
    invariant
      1 <= i and i <= ~~6~~ 5
    until
      ~~i > 5~~ i > 0
    loop
      io.put_string ("iteration " + i.out
      i := i + 1
    variant
      ~~6~~ 5 - i
    end
end
```

6

→ i = 6

(-1)

Runtime Checks



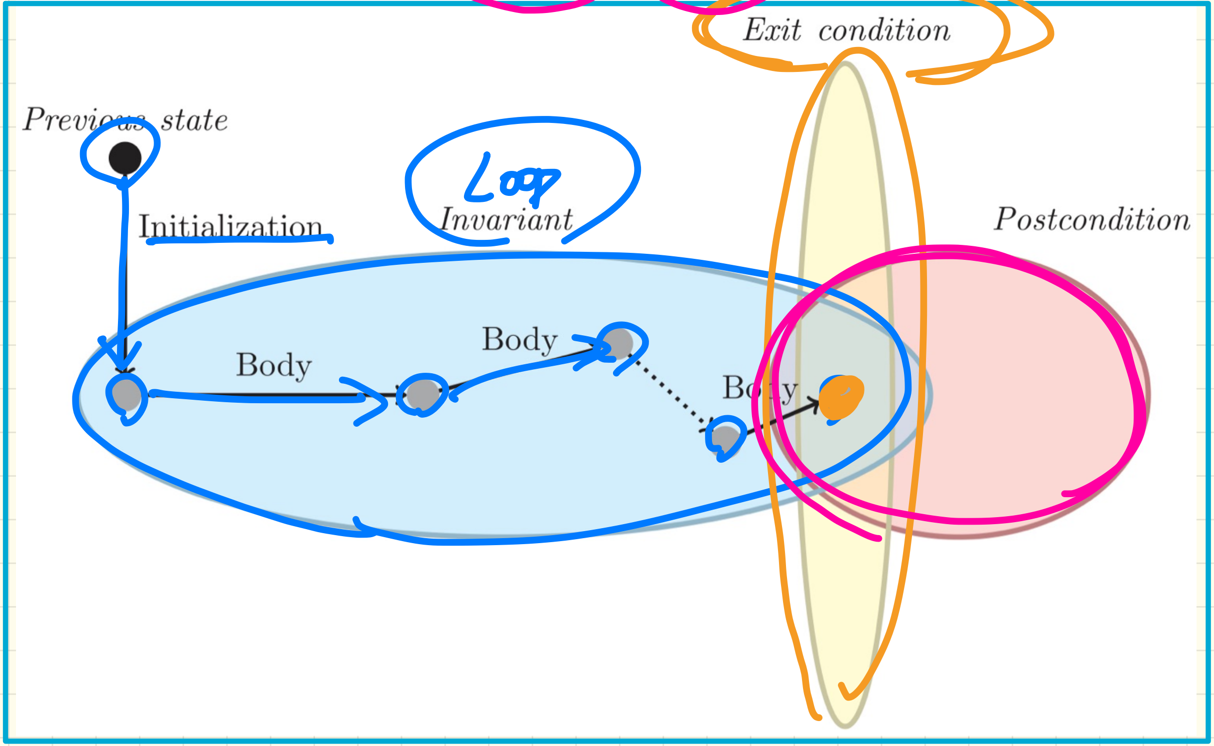
exit condition: $\boxed{i > 0}$ → exit right away.

invariant: 1 <= i <= 5 → LI Violation.

variant: 5 - i

Contracts of Loops: Visualization

- LI ∧ Exit ⇒ Postcondition



Contracts of Loops: Loop Invariant

```
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
```

i-1

LI

?

i i-1

$\forall j \mid \overset{1}{a.lower} \leq j \leq \overset{0}{\cancel{i}} .$ F

Result ≥ a[j]

LI :

i-1

across a.lower |..| ~~i~~ j

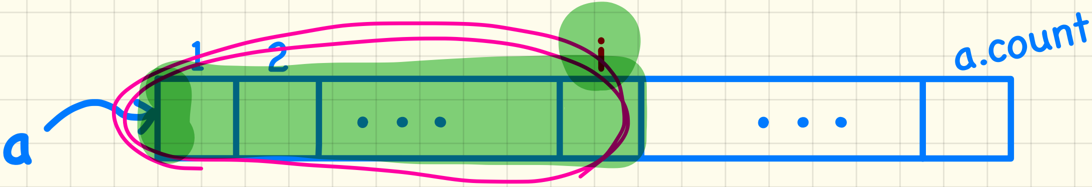
i

all

Result ≥ a[j]

end

Invariant: **Result** stores the max of the array scanned **so far**.



$$\forall x \mid F \bullet P(x) \equiv \boxed{T}.$$

∵ no violation witness can be found.

Finding Max: Version 1

a →

1	2	3	4
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 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
```

wrong.

i	R
1	20
2	20

R ≥ a[2]

3

20 40

R ≥ a[3]

F

AFTER ITERATION	i	Result	*LI*	EXIT (*i* > *a.upper*)?	*LV*
Initialization	●	●	●	●	●
1st	●	●	●	●	●
2nd	●	●	●	●	●

Finding Max: Version 2

1	2	3	4
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
```

-1

last Iteration :

i = a.upper

a.upper + 1

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	●	●	●	●	●

Correct Loops: Proof Obligations

```
{$Q$}   from
          $S_{init}$
        invariant
          $I$
        until
          $B$
        loop
          $S_{body}$
        variant
          $V$
        end        {$R$}
```

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

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

    $\{Q\}\ S_{init}\ \{LI\}$

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

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

    $LI \land$

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

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

    $I \land B \Rightarrow R$

    $B \land LI \Rightarrow R$

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

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

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

    [redacted]

$LI \land \neg B$

$V \geq 0$

$\{\ \}\ S_{body}\ \{\ \}$

$\{LI \land \neg B\}\ S_{body}\ \{V < V_0\}$

Correct Loops: Proof Obligations

```
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
```

ทำให้เป็นจริง

(I

B

Initialization:

Before Termination:

Upon Termination:

Non-Negative Variant:

Decreasing Variant:

Prove

$1 \leq j < 1$ $\forall x \mid F \bullet X(x)$ $\equiv$ T

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

Establishment of Loop Invariant:

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

① Calculate $wp(i := a.lower ; Result := a[i], \forall j \mid a.lower \leq j < i \bullet Result \geq a[j])$

$= \{ \text{wp rule on ;} \}$

$wp(i := a.lower, wp(Result := a[i], \forall j \mid a.lower \leq j < i \bullet Result \geq a[j]))$

$= \{ \text{wp rule of ;} \}$

$wp(i := a.lower, \forall j \mid a.lower \leq j < \overset{a.lower}{\cancel{i}} \bullet a[i] \geq a[j])$

$= \{ \text{wp on ;} \}$

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

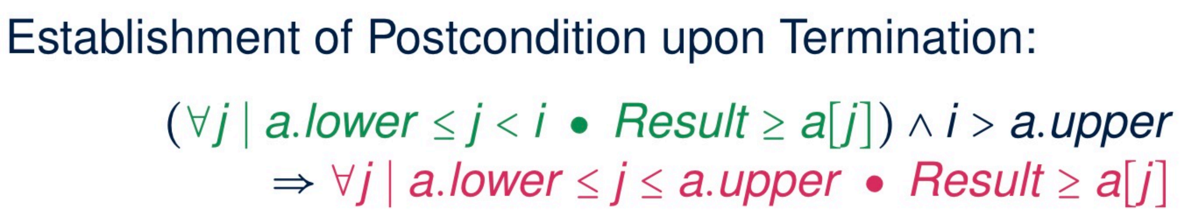
$\equiv True$

$$\forall x \mid \underline{\text{False}} \bullet \underline{P(x)}$$

$$|||$$

$$\boxed{\text{True}}$$

Prove



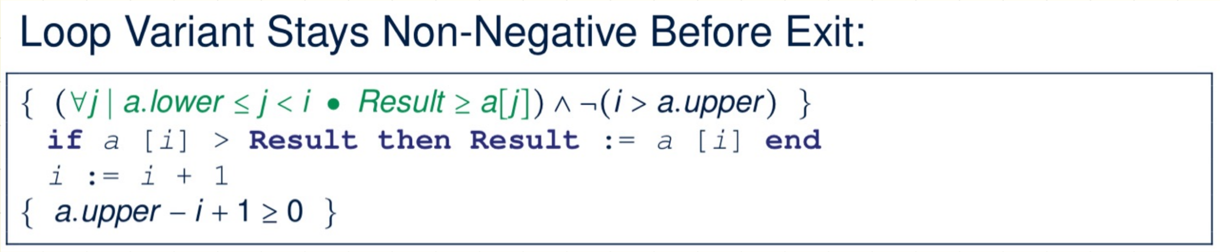
Establishment of Postcondition upon Termination:

$$(\forall j \mid a.lower \leq j < i \bullet Result \geq a[j]) \land i > a.upper$$
$$\Rightarrow \forall j \mid a.lower \leq j \leq a.upper \bullet Result \geq a[j]$$

Hint: Rewrite $j < i$ and $i > a.upper$ using >=

Hint: Identify i, j, a.upper on the number line.

Prove



Loop Variant Stays Non-Negative Before Exit:

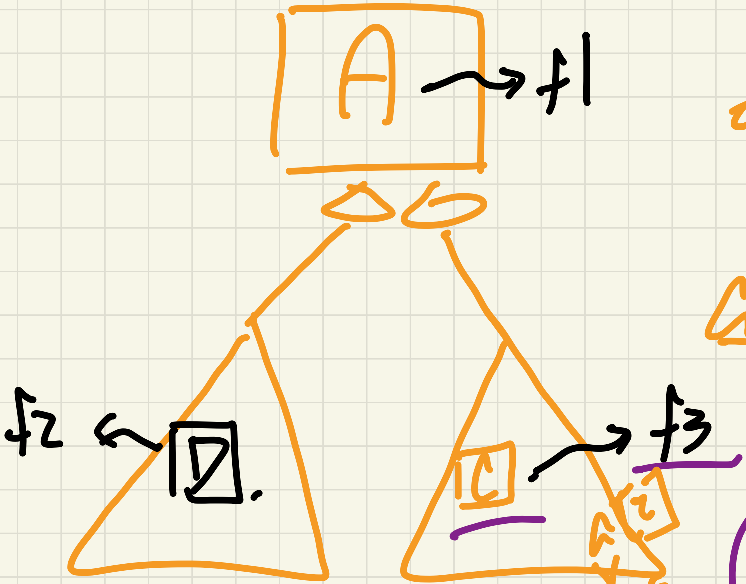
$\{\ (\forall j \mid a.lower \le j < i \bullet Result \ge a[j]) \land \neg(i > a.upper)\ \}$

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

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

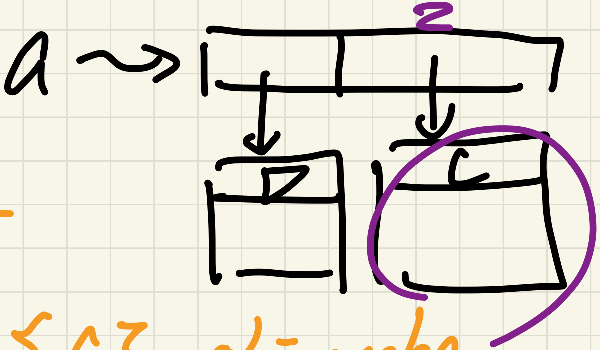
EXAM REVIEW I

MONDAY DECEMBER 9



obj : A

Create {C} obj.make



a: ARRAY[A]

a[i] := ?

across a is obj loop

obj.?f1

end

a[i].f3

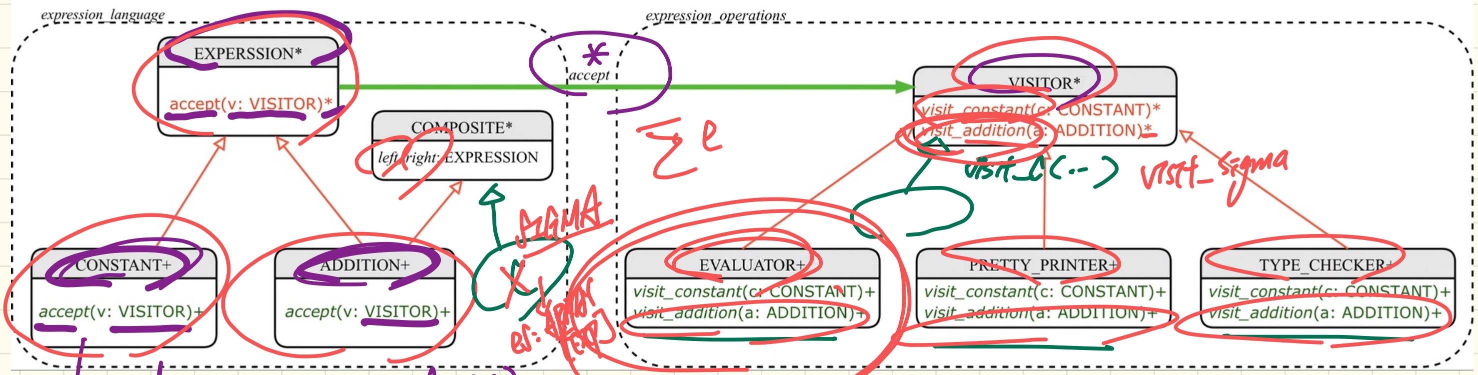
st : A

check attached {C} a[i] as c_obj then

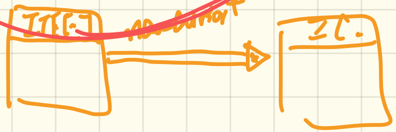
c_obj.f3

end

Visitor Design Pattern: Architecture



do
  v.visit_constant(Current)
end



How to Use Visitors

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

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
```

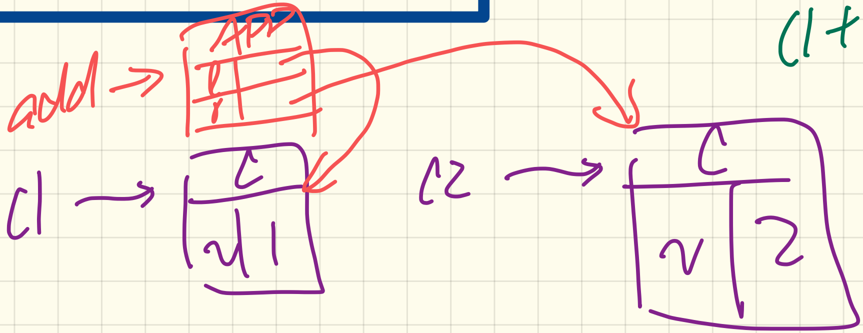
composite

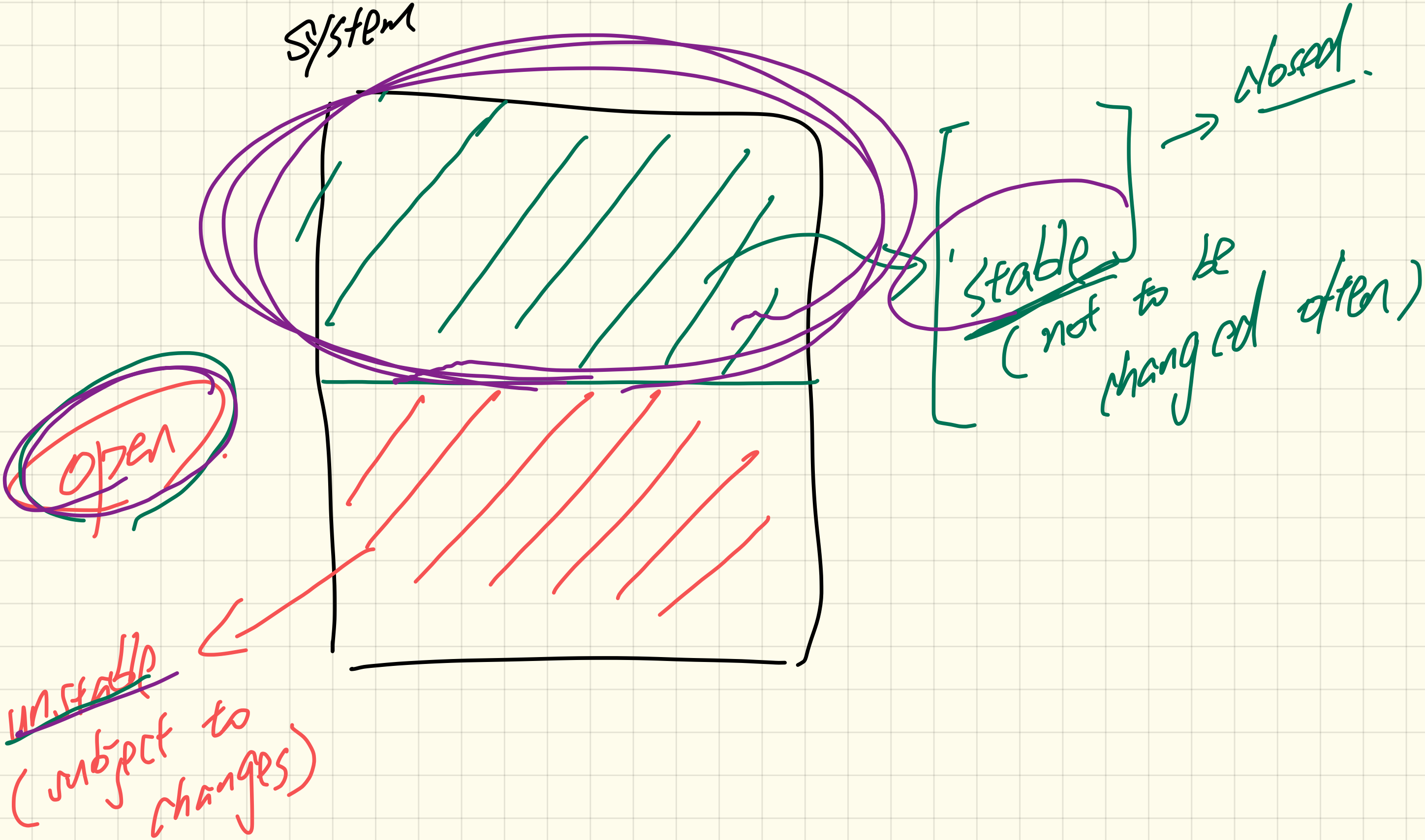
visitor

Write a fragment of code which builds:

$(1+2)+(3+4)$

Visualizing Line 4 to Line 6



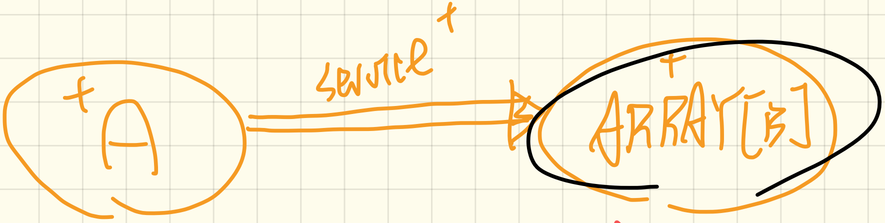

system
closed.
stable
(not to be changed often)
open
unstable
(subject to changes)

```
class A
  service: ARRAY[B]
  ⋮
end

class B
  ⋮
end
```

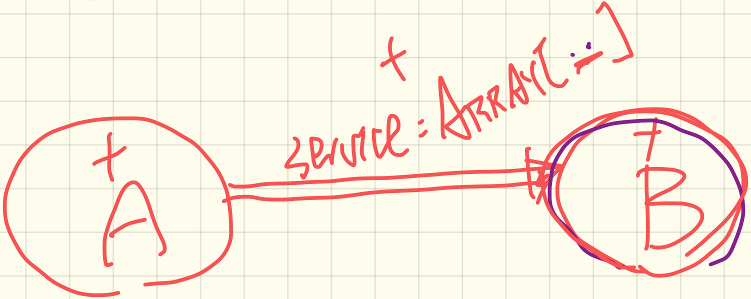
supplier

supplier

①

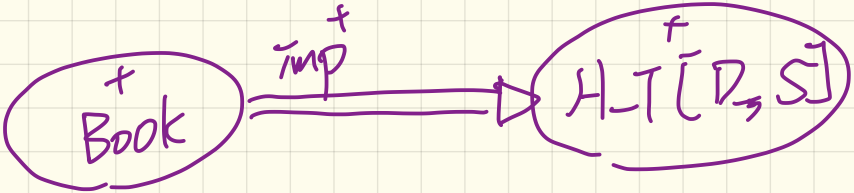


②

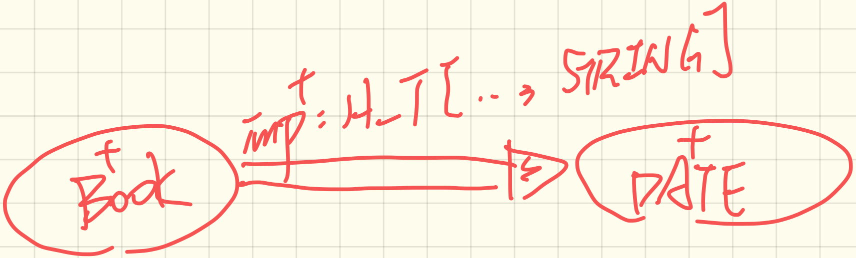


```
class Book
  imp : HASH_TABLE[DATE, STRING]
end
```

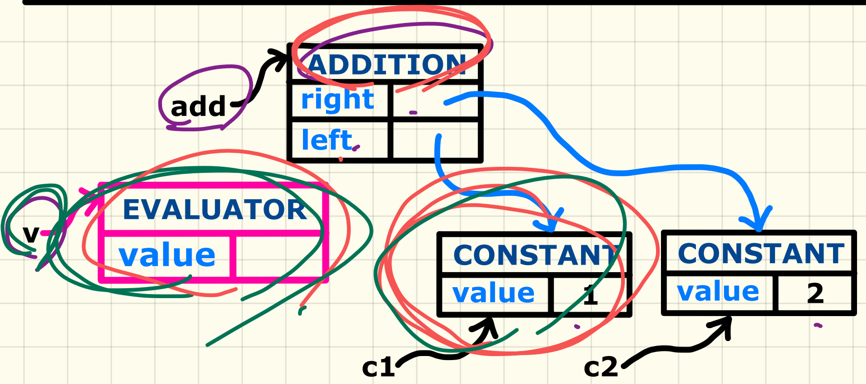
①



②



Executing **Composite** and **Visitor** Patterns at **Runtime**



Tracing add.accept(v)

Double Dispatch

add.accept(v)

↳ DT of add: ADDITION

⇒ call accept in ADDITION

↳ DT of v: EVALUATOR

⇒ call visit_addition in

```
deferred class VISITOR
  visit_constant(c: CONSTANT) deferred end
  visit_addition(a: ADDITION) deferred end
end
```

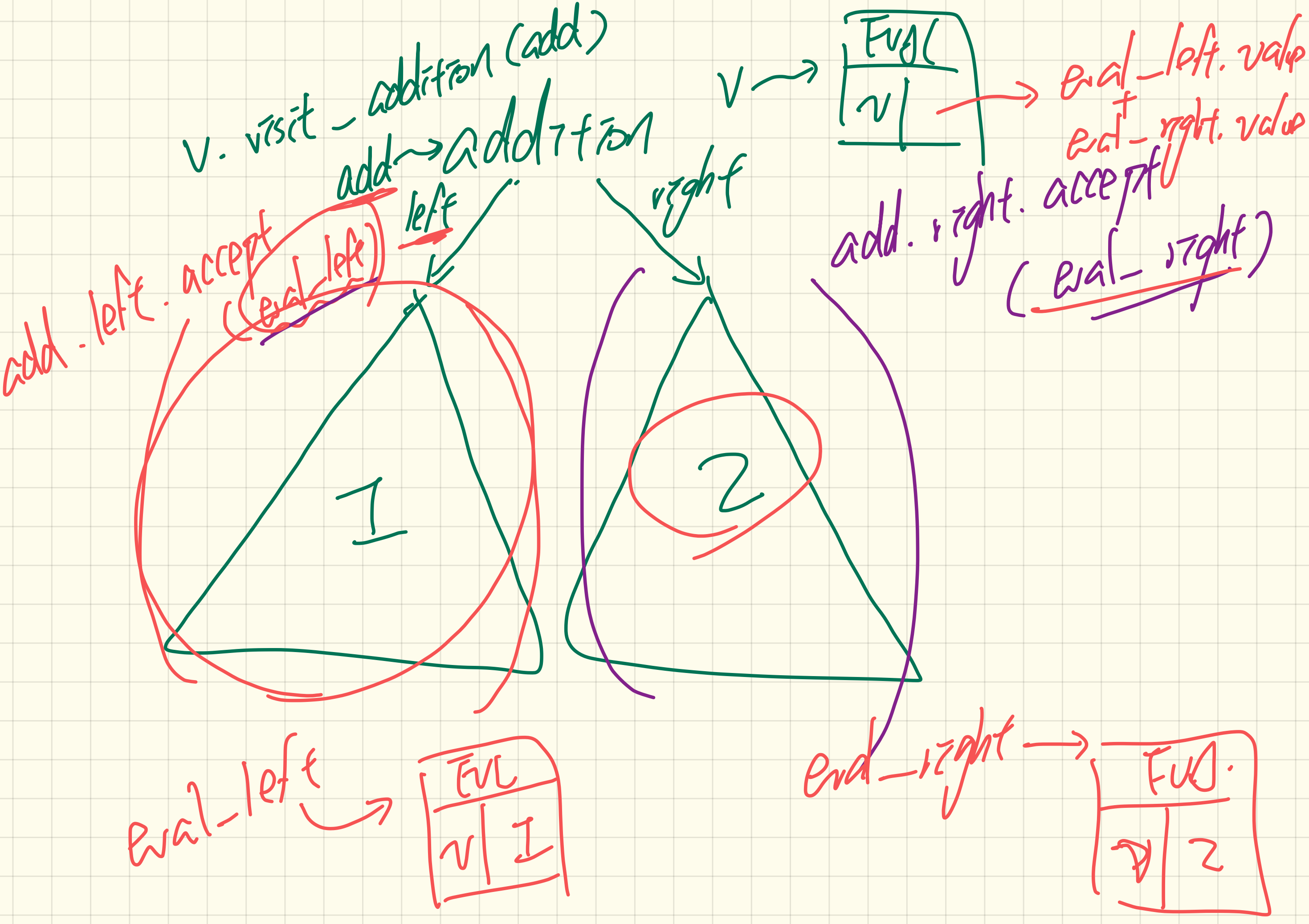
```
class EVALUATOR inherit VISITOR
  value : INTEGER
  visit_constant(c: CONSTANT)  do  value := c.value end
  visit_addition(a: ADDITION)
    local eval_left, eval_right: EVALUATOR
    do a.left.accept(eval_left)
       a.right.accept(eval_right)
       value := eval_left.value + eval_right.value
    end
end
```

double dispatch

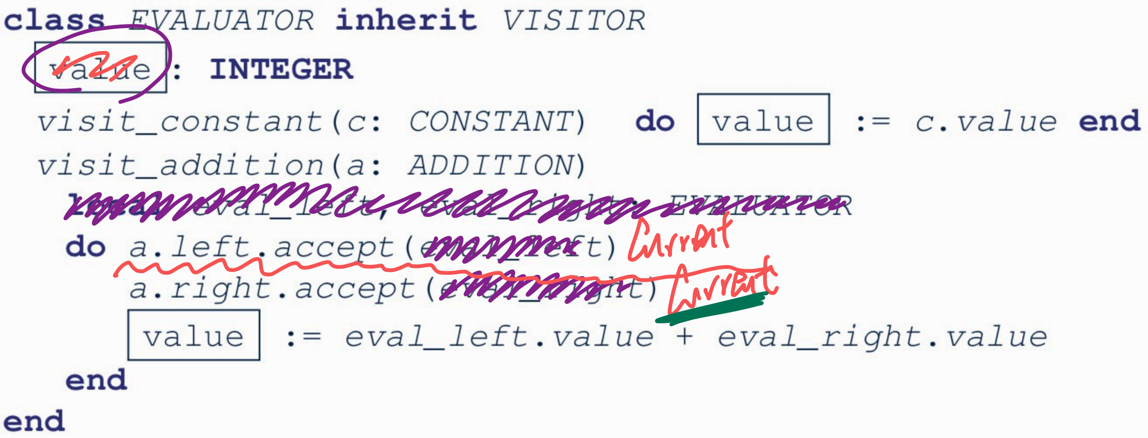
double dispatch

```
class CONSTANT inherit EXPRESSION
...
  accept(v: VISITOR)
    do
      v.visit_constant(Current)
    end
end
```

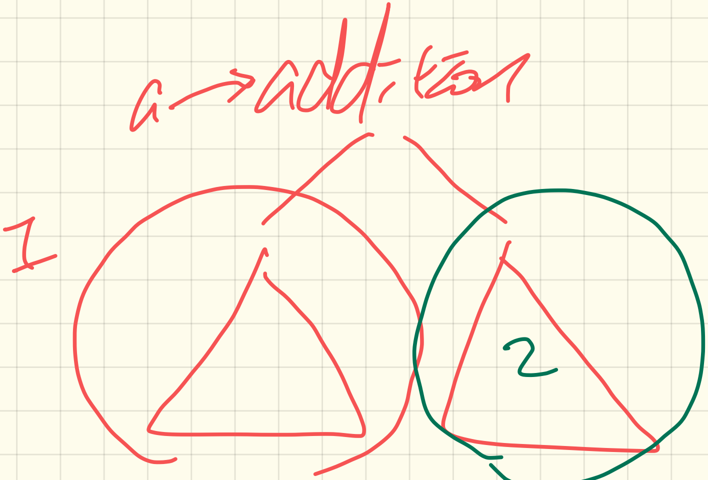
```
class ADDITION
inherit EXPRESSION COMPOSITE
...
  accept(v: VISITOR)
    do
      v.visit_addition(Current)
    end
end
```

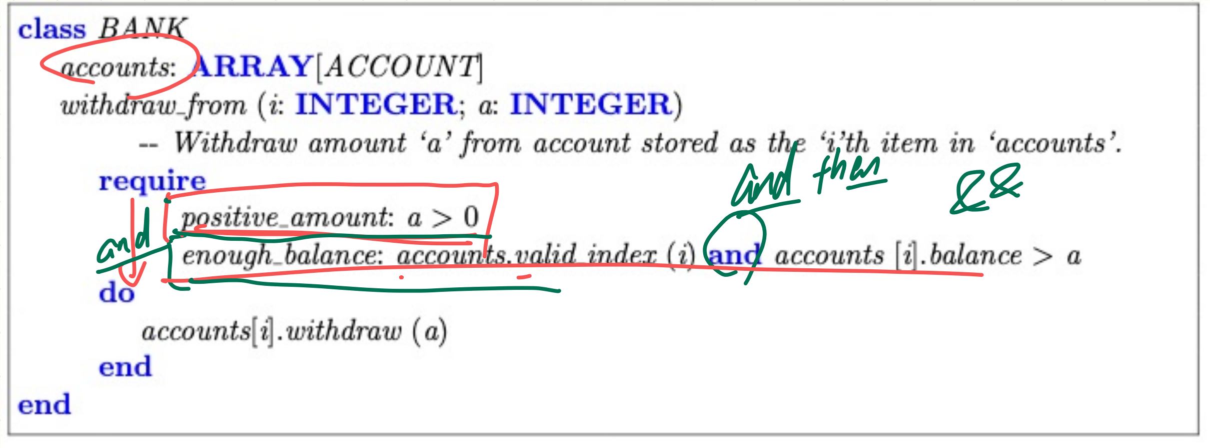

v. visit_addition(add)
add → addition
left
right
v → Eval(
v
eval_left.value
+
eval_right.value
add.left.accept(eval_left)
add.right.accept(eval_right)
1
2
eval_left → Evl
v | 1
eval_right → Evl.
v | 2

2 X



```
class EVALUATOR inherit VISITOR
  ~~value~~: INTEGER
  visit_constant(c: CONSTANT)  do value  := c.value end
  visit_addition(a: ADDITION)
   ~~local eval_left, eval_right: EVALUATOR~~
   do a.left.accept(~~eval_left~~) current
      a.right.accept(~~eval_right~~) current
      value  := eval_left.value + eval_right.value
   end
end
```





```
class BANK
  accounts: ARRAY[ACCOUNT]
  withdraw_from (i: INTEGER; a: INTEGER)
      -- Withdraw amount 'a' from account stored as the 'i'th item in 'accounts'.
    require
      positive_amount: a > 0
      enough_balance: accounts.valid_index (i) and accounts [i].balance > a
    do
      accounts[i].withdraw (a)
    end
end
```

and then

&&

and

-1

-1

balance_cond: accounts[i].balance > a

→ valid_index: accounts.valid_index(i)

require

p1 i >= 0 and i <= count

↓

p2 a[i] > 0

⋮

if there's dependency among preconditions, put the least-depending first.

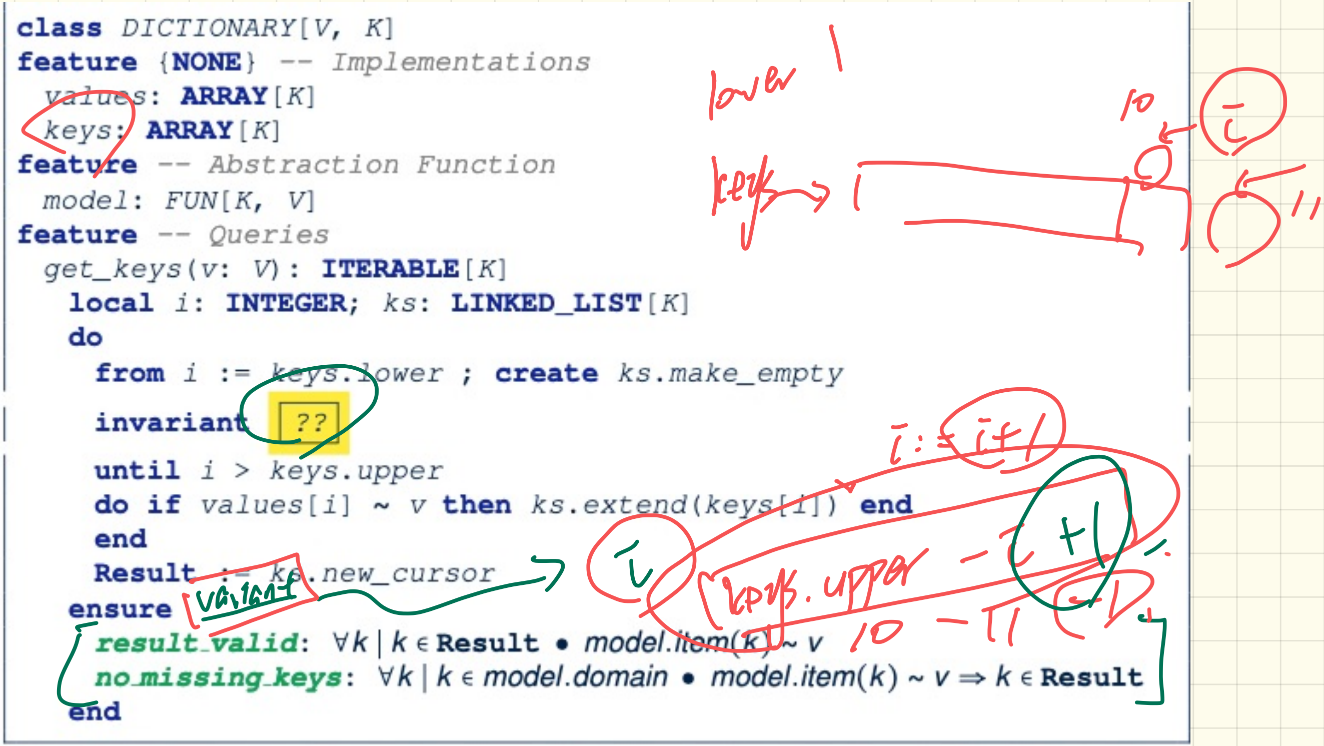
p1 and then p2

ensure

q1

↓

q2



```
class DICTIONARY[V, K]
feature {NONE} -- Implementations
  values: ARRAY[K]
  keys: ARRAY[K]
feature -- Abstraction Function
  model: FUN[K, V]
feature -- Queries
  get_keys(v: V): ITERABLE[K]
    local i: INTEGER; ks: LINKED_LIST[K]
    do
      from i := keys.lower ; create ks.make_empty
      invariant ??
      until i > keys.upper
      do if values[i] ~ v then ks.extend(keys[i]) end
      end
      Result := ks.new_cursor
    ensure
      result_valid: ∀k | k ∈ Result • model.item(k) ~ v
      no_missing_keys: ∀k | k ∈ model.domain • model.item(k) ~ v ⇒ k ∈ Result
    end
```

lower

keys → i

10

i

11

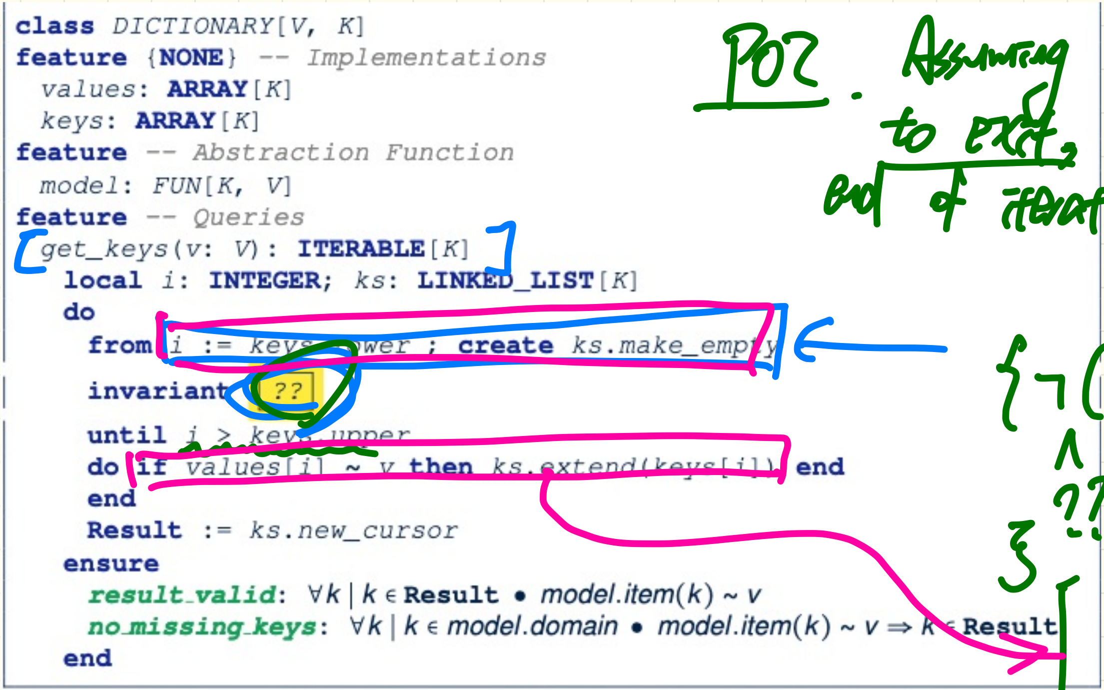
i := i+1

variant

i

keys.upper - i + 1

10 - 11 (-1)



```
class DICTIONARY[V, K]
feature {NONE} -- Implementations
  values: ARRAY[K]
  keys: ARRAY[K]
feature -- Abstraction Function
  model: FUN[K, V]
feature -- Queries
  get_keys(v: V): ITERABLE[K]
    local i: INTEGER; ks: LINKED_LIST[K]
    do
      from i := keys.lower ; create ks.make_empty
      invariant ??
      until i > keys.upper
      do if values[i] ~ v then ks.extend(keys[i]) end
      end
      Result := ks.new_cursor
    ensure
      result_valid: ∀k | k ∈ Result • model.item(k) ~ v
      no_missing_keys: ∀k | k ∈ model.domain • model.item(k) ~ v ⇒ k ∈ Result
    end
```

PO2. Assuming not ready to exit, after the end of iteration, LI is maintained.

$\{ \neg(i > keys.upper) \wedge ?? \}$

$\{ LI\ ?? \}$

PO1: Init establishes the LI.

$\{ True \}$ i := keys.lower ; create ks.m_e $\{ ?? \}$

Correct Loops: Proof Obligations

```
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
```

maintaining

I

B

Initialization:

Before Termination:

Upon Termination:

Non-Negative Variant:

Decreasing Variant:

Prove

$1 \leq j < 1 \quad \forall x \mid F \bullet P(x) \equiv T$

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

Establishment of Loop Invariant:

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

① Calculate $wp(i := a.lower ; Result := a[i], \forall j \mid a.lower \leq j < i \bullet Result \geq a[j])$

$= \{ \text{wp rule on } ; \}$

$wp(i := a.lower, wp(Result := a[i], \forall j \mid a.lower \leq j < i \bullet Result \geq a[j]))$

$= \{ \text{wp rule of } ; \}$

$wp(i := a.lower, \forall j \mid a.lower \leq j < \overset{a.lower}{\cancel{i}} \bullet a[i] \geq a[j])$

$= \{ \text{wp on } ; \}$

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

$\equiv T$

$\equiv True$

from

:

invariant I ← loop invariant

until

B ← exit condition

→ constraint on loop counter.

loop

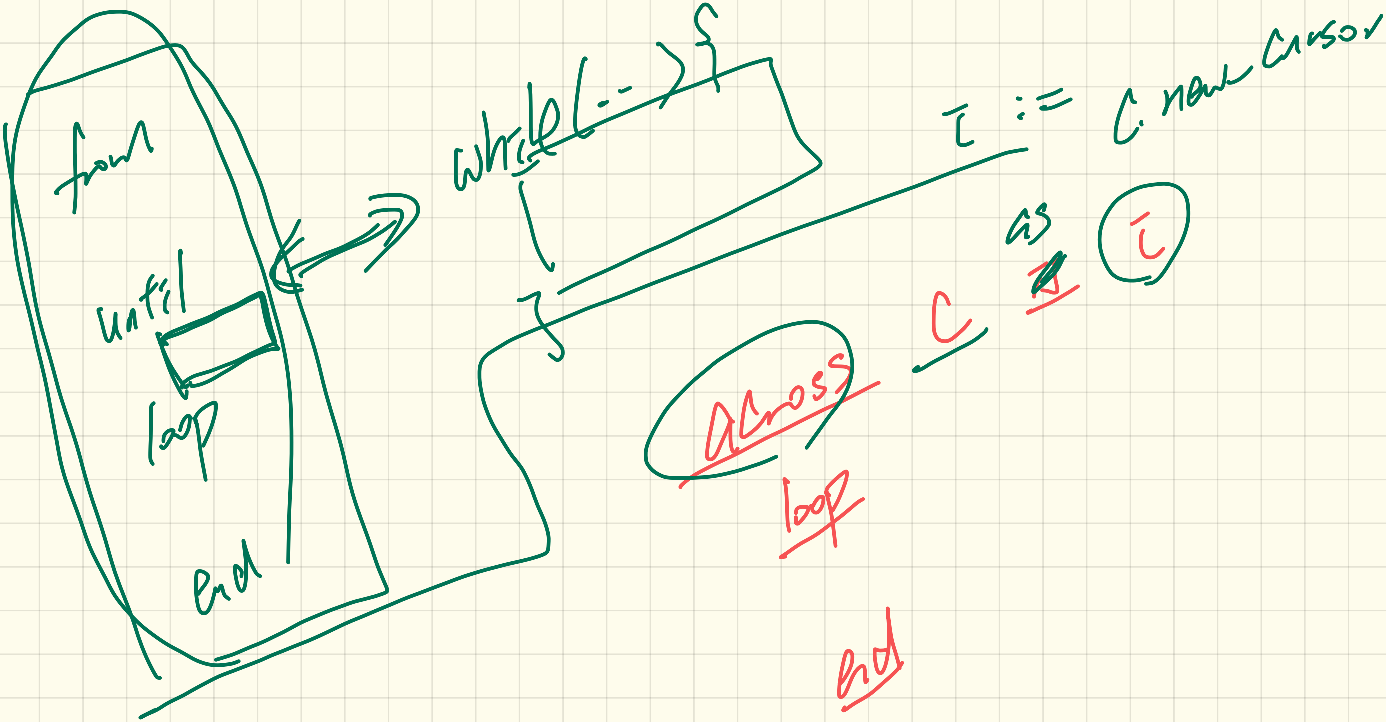
:

end

ensure R ← postcondition.

PO: Upon termination, given that I is maintained, postcondition is established.

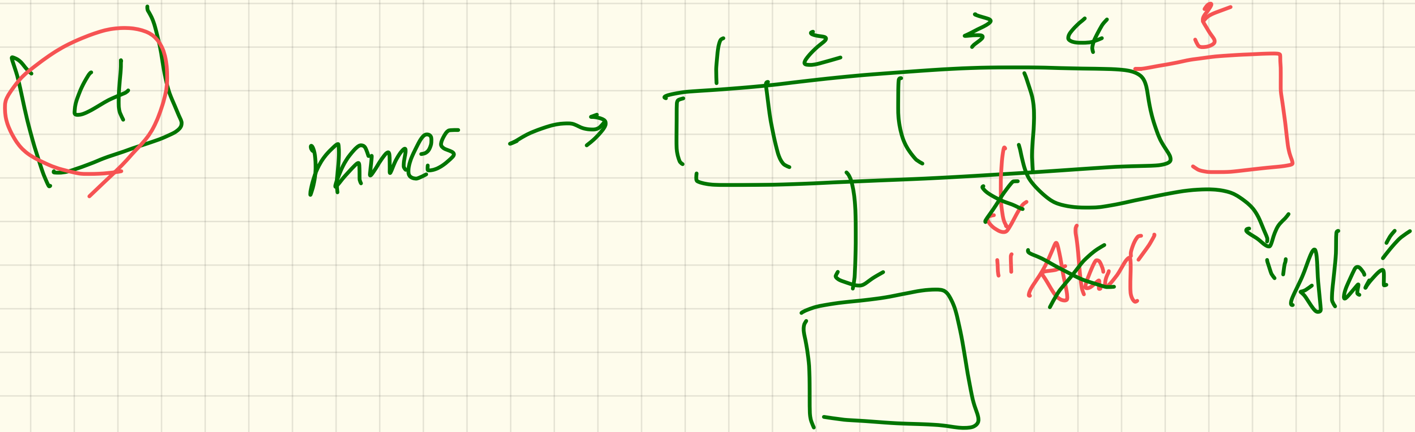
$$B \wedge I \Rightarrow R$$


from
until
loop
end
while (- -) {
}
i := c.new_cursor
as
i
across
c
loop
end

names.count = dd names.count ~~+~~ 1

integer

names.count = (dd names.dt.count + 1



EXAM REVIEW II

WEDNESDAY DECEMBER 11

Math Models

REL . SET FUN

$r1, r2 : REL[I, S]$

Command

union(other: REL)

API

$\cup$

Impl.

$r1.union(r2)$

infix |\/|

modify r1

queries

unioned(other: REL) : REL

Contract

$r3 := r1.unioned(r2)$

not modified

WP rules

$wp(x := e,\ R) = ?$

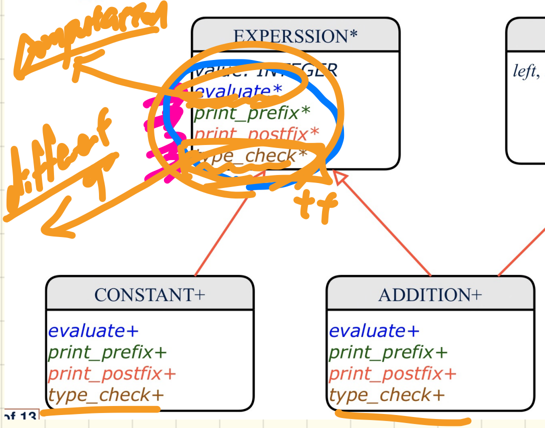
$wp(\underline{\text{if}}\ \cdots\ \underline{\text{then}}\ \cdots\ \underline{\text{else}}\ ,\ R) = ?$

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

Extend the *composite pattern* to support *operations* such as `evaluate`, pretty printing (`print_prefix`, `print_postfix`), and `type_check`.

cohesion

SCP



Shared Data via Inheritance

Descendant:

```
class DEPOSIT inherit SHARED_DATA
      -- 'maximum_balance' relevant
end

class WITHDRAW inherit SHARED_DATA
      -- 'minimum_balance' relevant
end

class INT_TRANSFER inherit SHARED_DATA
      -- 'exchange_rate' relevant
end

class ACCOUNT inherit SHARED_DATA
feature
      -- 'interest_rate' relevant
      deposits: DEPOSIT_LIST
      withdraws: WITHDRAW_LIST
end
```

violating cohesion

inherited.

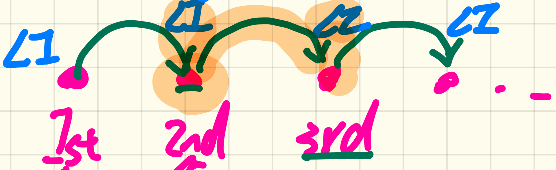
Ancestor:

```
class
 SHARED_DATA
feature
 interest_rate: REAL
 exchange_rate: REAL
 minimum_balance: INTEGER
 maximum_balance: INTEGER
 ...
end
```

Problems?

cohesion.

Correct Loops: Proof Obligations



$\{\neg B$

begin. begin

$\{Q\}$ Sinit $\{LI\}$

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

- 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.

    B

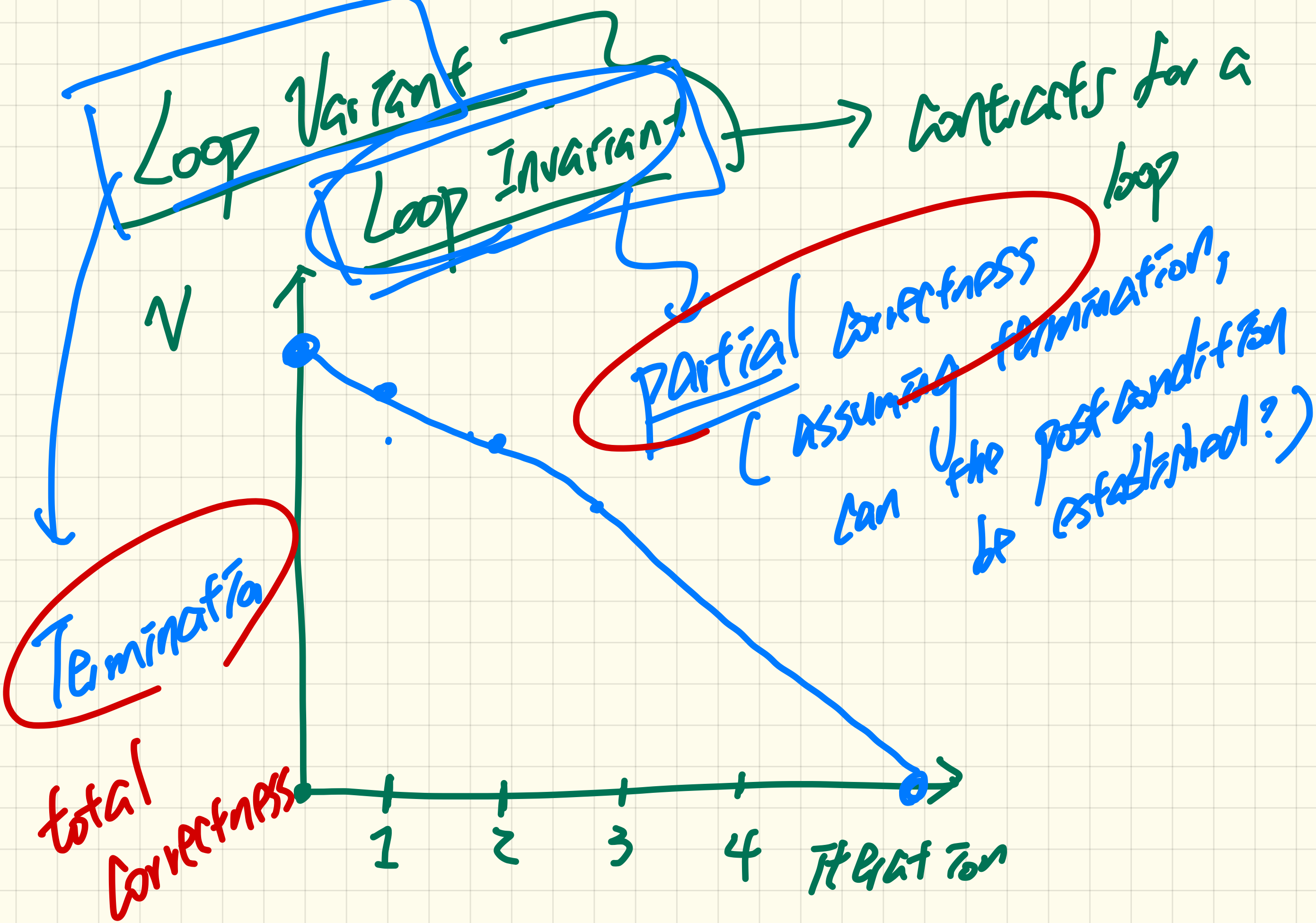
    $\{I \land \neg B\}\ S_{body}\ \{I\}$
  - If ready to exit and ***LI*** I maintained, postcondition ***R*** is established.

    LI

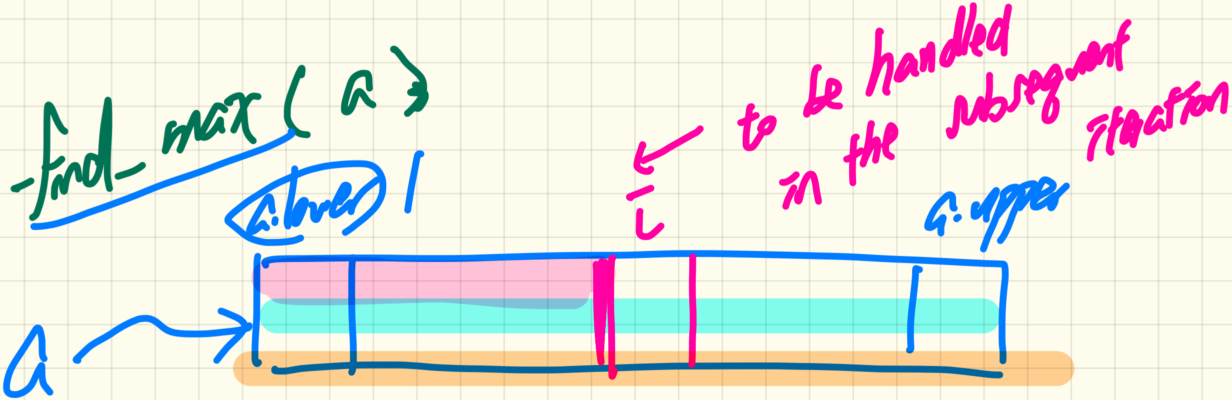
    $I \land B \Rightarrow R$
- A loop ***terminates*** if:
  - Given ***LI*** I, and not yet to exit, S_{body} maintains ***LV*** V as non-negative.

    $\{I \land \neg B\}\ S_{body}\ \{V \geq 0\}$
  - Given ***LI*** I, and not yet to exit, S_{body} decrements ***LV*** V.

    [redacted]


Loop Variant
Loop Invariant
contracts for a loop
V
Partial correctness
(assuming termination, can the postcondition be established?)
Termination
total correctness
1
2
3
4
iteration

find_max (a)



Result

postcondition :

$$\forall i \mid 1 \leq i \leq a.count \bullet Result \geq a[i]$$

$$\forall j \mid 1 \leq j \leq i - 1 \bullet Result \geq a[j]$$

range constraint

it's the case

↗ $\forall x \mid 1 \le x \le 5 \bullet x^2 \geq 25 \,]\; F$

↗ $\underline{\exists x} \mid \underline{1 \le x \le 5} \bullet \underline{x^2 \geq 25} \,]\; T$

such that

and

$\forall x \bullet 1 \le x \le 5 \Rightarrow x^2 \geq 25$

$\exists x \bullet 1 \le x \le 5 \wedge x^2 \geq 25$

Correct Loops: Proof Obligations

```
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
```

S

Init

LI

Initialization:

$\{T\}$ ______ $\{\forall j \mid a.lower \ldots \cdot \ \}$

$\{T\}$ S $\{LI\}$ ×

Before Termination:

Upon Termination:

a.upper − i + 1 ≥ 0

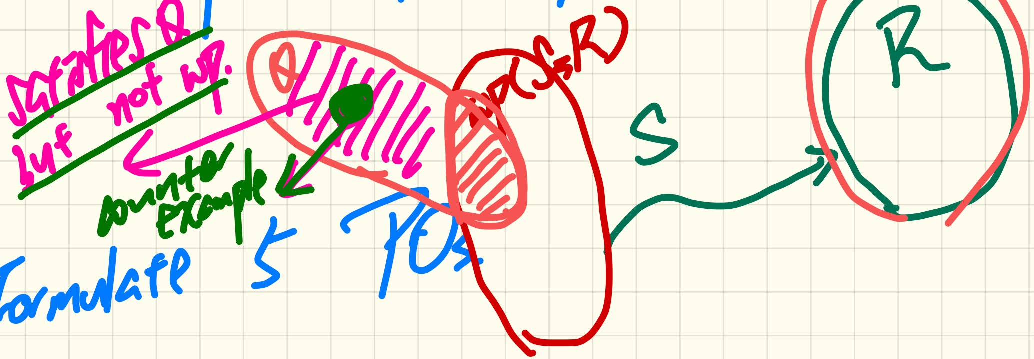
if ___ $\{$ ______ $\}$

Non-Negative Variant:

Decreasing Variant:

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

Given a loop (Eiffel syntax)



Formulate S POs

4 of them

Hoare Triples

{Q} S {R}

$Q \Rightarrow wp(S, R).$

```
max_of (x: INTEGER ; y: INTEGER)
  do
    Result := x
    if y > x then
      Result := y
    end
  end
  ensure Result >= x ∧ Result >= y
```

$wp(\text{if} \ldots \text{then} \ldots \text{else} \ldots, R)$

$= [\quad] \Rightarrow wp \ldots$

$\wedge$

$[\quad] \Rightarrow w\ldots$

$wp(R := x, P \wedge q) = P \wedge q[R := x]$

0. Formulate:

✓ {True}

$\{ q \}$

2. $True \Rightarrow (P \wedge q)$

$\equiv P \wedge q$

1. Calculate $wp(\text{Result} := x \,;\, \text{if } y > x \text{ then Result} := y \text{ else Result} := \text{Result}, \; R \geq x \wedge R \geq y)$

$\neg(y > x) \Rightarrow wp(R := R, \ldots)$

$\equiv R \geq x \wedge R \geq y$

{ rule of ; }

$= wp(\text{Result} := x, \; wp(\_\_\_\_ \,;\, \_\_\_\_))$

```
if B then
  Result := e1
end
```

$$B \Rightarrow wp(Result := e1, R)$$

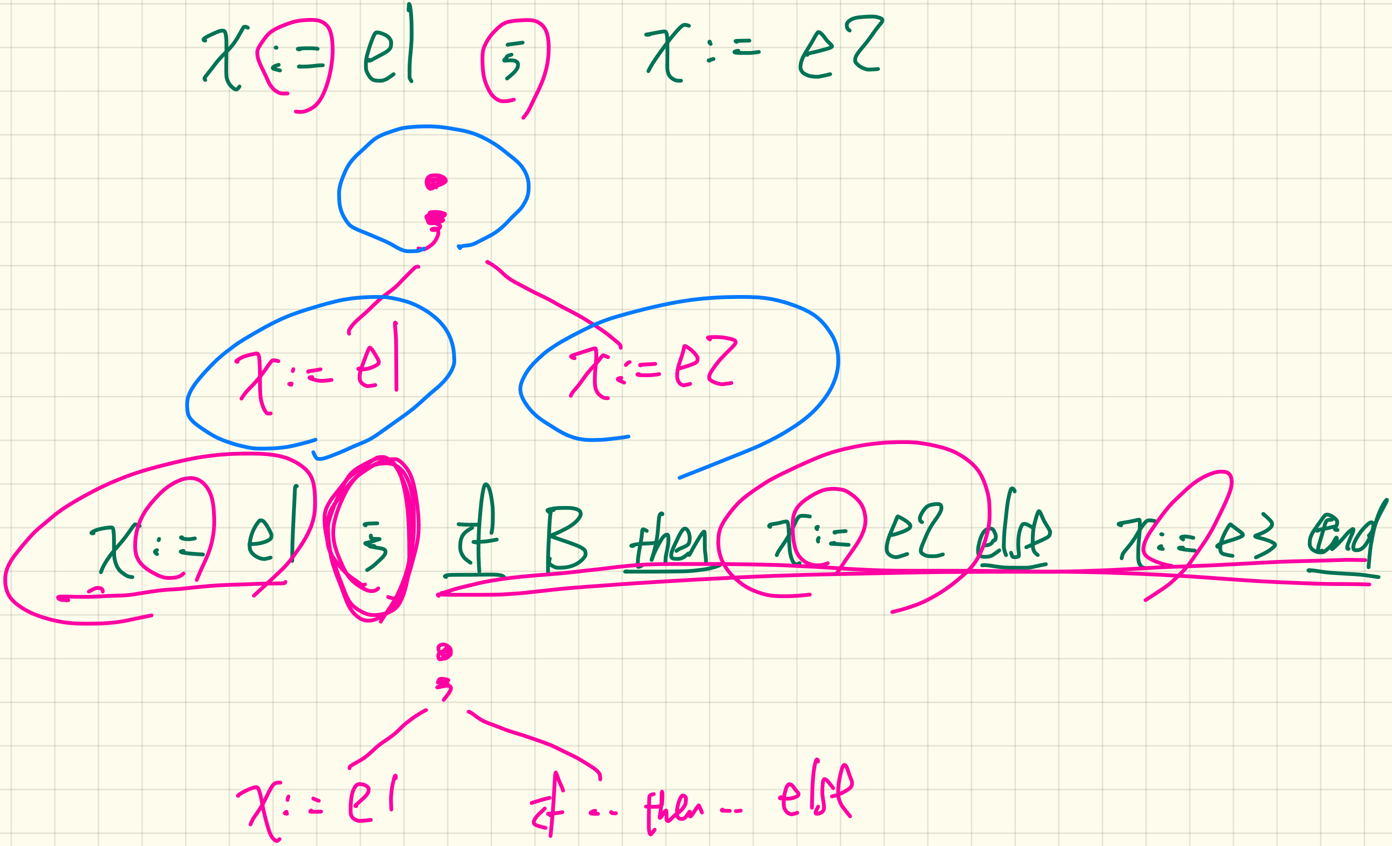
$$\wedge$$

$$\neg B \Rightarrow wp(R := R, R)$$

$$\overset{T}{\boxed{p}} \Rightarrow (q \wedge r)$$

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

$$\cancel{q \wedge r} \qquad (x > 0 \Rightarrow q) \wedge (x > 0 \Rightarrow r)$$

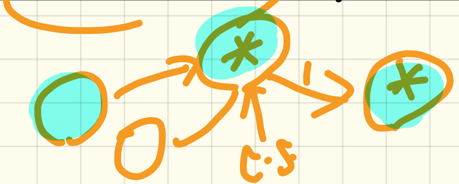

x := e1 ; x := e2
;
x := e1
x := e2
x := e1 ; if B then x := e2 else x := e3 end
;
x := e1
if .. then .. else

require Q
do
  ; S
ensure
  - R_1
  - R_2
end

$$\boxed{\{Q\}\ S\ \{\underline{R_1} \wedge \underline{R_2}\}}$$

$$\{Q\}\ S\ \{R_1\}$$
$$\wedge$$
$$\{Q\}\ S\ \{R_2\}$$

Interactive System: Top-Down Design vs. OO Design

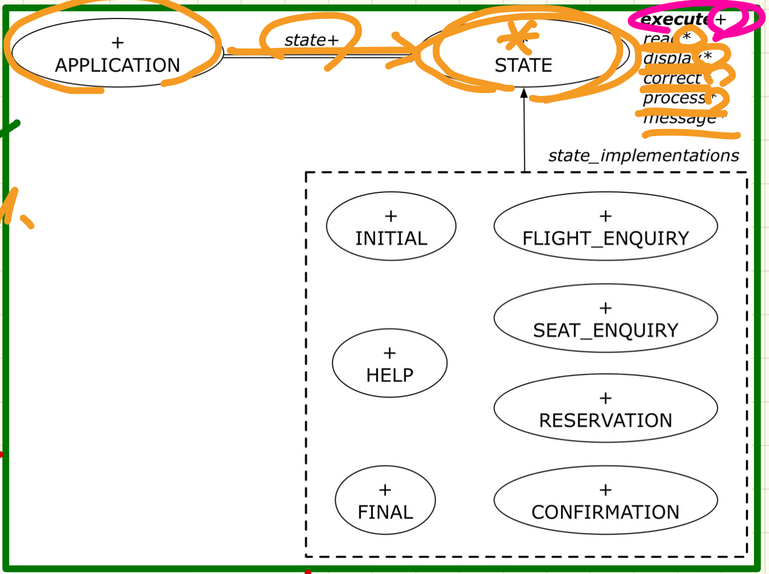


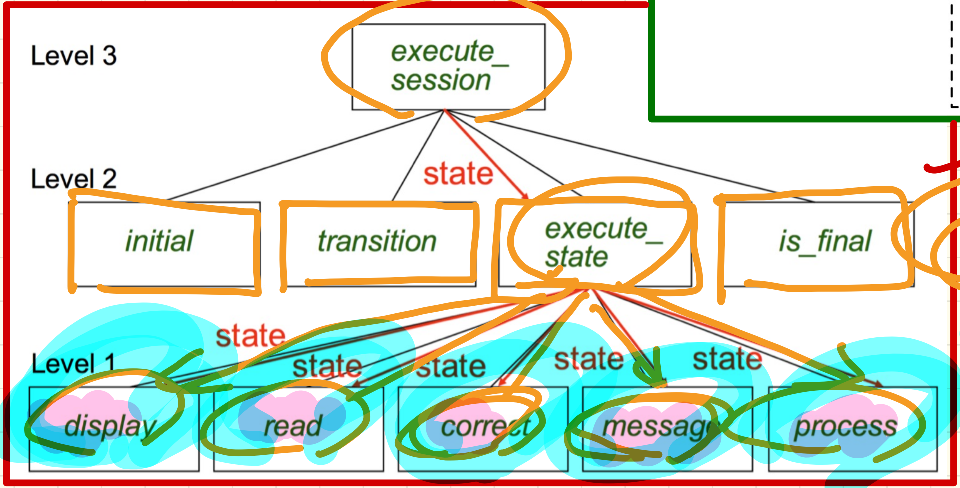
Object-Oriented

defined.

current_state: **STATE**

current_state.*execute_session*



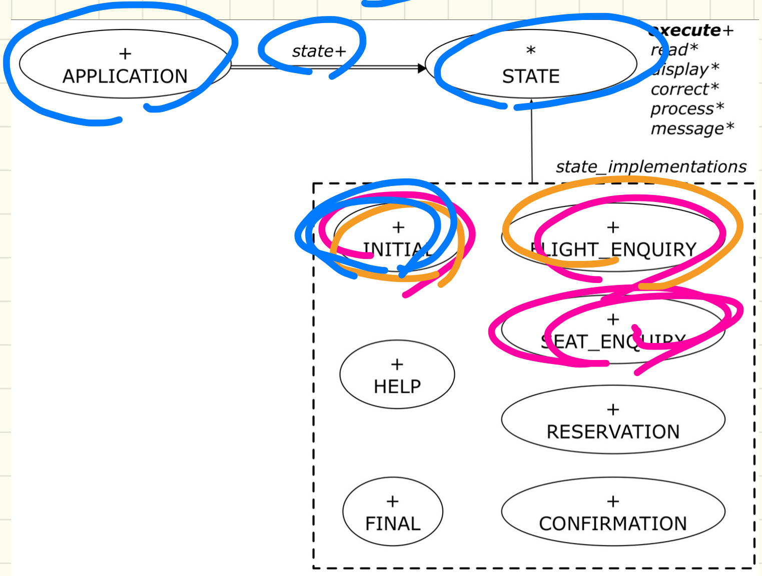


Top-Down

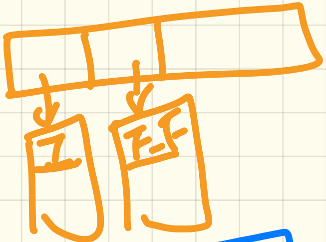
current_state: **INTEGER**

execute_session(current_stste)

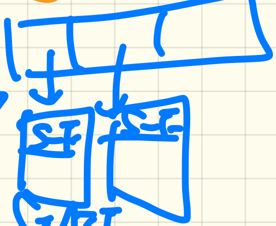
Code to the interface, not to the imp.



states →



states2 →



class APP

state : STATE

state2 : SEAT_ENQ

states : A[STATE]

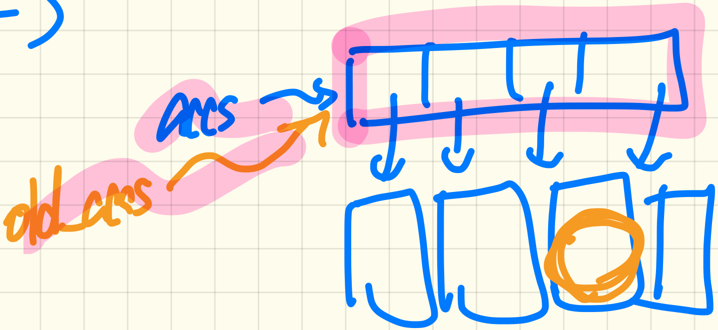
states2 : A[S_E]

: ACCOUNT

- Consider the query *account_of (n: STRING)* of *BANK*.
- How do we specify (part of) its postcondition to assert that the state of the bank remains unchanged:
  - ○ `accounts = old accounts` [×]
  - ○ `accounts = old accounts.twin` [×]
  - ○ `accounts = old accounts.deep_twin` [×]
  - ○ `accounts ~ old accounts` [×]
  - ○ `accounts ~ old accounts.twin` [×]
  - ○ `accounts ~ old accounts.deep_twin` [✓]

old_acc := accounts

account_of (. - - -)



First Design Attempt

```
class Student {
  Course[] courses;
  int noc;
  int kind;
  double premiumRate;
  double discountRate;
  Student (int kind){
    this.kind = kind;
  }
  ...
}
```

not related to each other

Cohesion

```
double getTuition(){
  double tuition = 0;
  for(int i = 0; i < noc; i++){
    tuition += courses[i].fee;
  }
  if (this.kind == 1) {
    return tuition * premiumRate;
  }
  else if (this.kind == 2) {
    return tuition * discountRate;
  }
}
```

```
double register(Course c){
  int MAX;
  if (this.kind == 1) { MAX = 6; }
  else if (this.kind == 2) { MAX = 4; }
  if (noc == MAX) { /* Error */ }
  else {
    courses[noc] = c;
    noc++;
  }
}
```

EXAM REVIEW III

THURSDAY DECEMBER 13

Solution to Proving (1.2)

We first calculate the wp for the loop body to maintain the LI:

$$\begin{array}{ll}
& wp(\texttt{if a[i] > Result then Result := a[i] end; i := i + 1}, \boxed{\forall j \mid a.lower \le j \le i - 1 \bullet a.lower \le j \land j \le a.upper \land \textbf{Result} \ge a[j]})\\
= & \{wp \texttt{ rule for seq. comp. }\}\\
& wp(\texttt{if a[i] > Result then Result := a[i] end}, \boxed{wp(\texttt{i := i + 1}, \boxed{\forall j \mid a.lower \le j \le \overset{(i+1)-1}{\cancel{i-1}} \bullet a.lower \le j \land j \le a.upper \land \textbf{Result} \ge a[j]})})\\
= & \{wp \texttt{ rule for assignment}\}\\
& wp(\texttt{if a[i] > Result then Result := a[i] end}, \boxed{\forall j \mid a.lower \le j \le i \bullet a.lower \le j \land j \le a.upper \land \textbf{Result} \ge a[j]})\\
= & \{wp \texttt{ rule for conditional}\}\\
& a[i] > \textbf{Result} \implies wp(\texttt{Result := a[i]}, \boxed{\forall j \mid a.lower \le j \le i \bullet a.lower \le j \land j \le a.upper \land \overset{\checkmark\, a[i]}{\cancel{\textbf{Result}}} \ge a[j]})\\
& \land\\
& a[i] \le \textbf{Result} \implies wp(\texttt{Result := Result}, \boxed{\forall j \mid a.lower \le j \le i \bullet a.lower \le j \land j \le a.upper \land \overset{\checkmark}{\textbf{Result}} \ge a[j]})\\
= & \{wp \texttt{ rule for assignment, twice}\}\\
& a[i] > \textbf{Result} \implies \forall j \mid a.lower \le j \le i \bullet a.lower \le j \land j \le a.upper \land \textbf{a[i]} \ge a[j] \qquad \bullet P\\
& \land \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land\\
& a[i] \le \textbf{Result} \implies \forall j \mid a.lower \le j \le i \bullet a.lower \le j \land j \le a.upper \land \textbf{Result} \ge a[j] \qquad \bullet Q
\end{array}$$

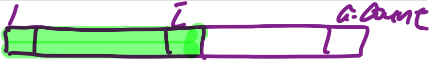
We then prove that the precondition (i.e., ¬(exit condition) and LI) is no weaker than the above calculated wp:

- To prove:

$$\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad \neg B \qquad\qquad\qquad\qquad\qquad\qquad LI\\
\neg(i > a.upper) \land (\ \forall j \mid a.lower \le j \le i - 1 \bullet a.lower \le j \land j \le a.upper \land \textbf{Result} \ge a[j]\ )\\
\implies a[i] > \textbf{Result} \implies \boxed{\forall j \mid a.lower \le j \le i \bullet a.lower \le j \land j \le a.upper \land a[i] \ge a[j]}
\end{array}$$

$$\neg B \land LI \Rightarrow (P \land Q) \quad = \quad (\neg B \land LI \Rightarrow P) \land (\neg B \land LI \Rightarrow Q)$$

Proof:



$$\begin{array}{ll}
\checkmark & \boxed{\forall j \mid a.lower \le j \le i \bullet a.lower \le j \land j \le a.upper \land a[i] \ge a[j]}\\
\equiv & \{\texttt{split range: } \forall j \mid a.lower \le j \le i \bullet P(j) \equiv (\forall j \mid a.lower \le j \le i - 1 \bullet P(j)) \land P(i)\}\\
\rightarrow & (\forall j \mid a.lower \le j \le \mathbf{i - 1} \bullet a.lower \le j \land j \le a.upper \land a[i] \ge a[j]) \land (a.lower \le \mathbf{i} \land \mathbf{i} \le a.upper \land a[i] \ge a[\mathbf{i}])\\
\equiv & \{\texttt{antecedent: } a[i] > \texttt{Result}\texttt{; and RHS of precond: } \forall j \mid a.lower \le j \le i - 1 \bullet a.lower \le j \land j \le a.upper \land \texttt{Result} \ge a[j]\}\\
& true \land (a.lower \le i \land i \le a.upper \land a[i] \ge a[i])\\
\equiv & \{\texttt{LHS of precond: } \neg(i > a.upper) \texttt{ and } a[i] \ge a[i] \equiv true\}\\
& true
\end{array}$$

Given.

wp(x := e , . .)

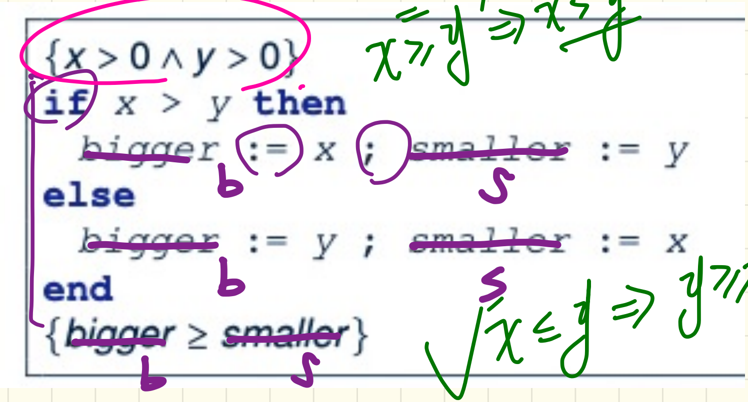
wp(if . . , . .)

wp(S_1 ; S_2 , . .)

5 POs. X

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

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



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

b s b s b s

$x \geq y \overset{X}{\Rightarrow} \cancel{x > y}$

$\surd\ x \leq y \Rightarrow y \geq x$

$wp(b := x ; s := y , b \geq s)$

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

$wp(b := x , wp(s := y , b \geq \underset{y}{s}))$

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

$wp(b := x , b \geq y)$

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

$x \geq y$ T

0. $\{x > 0 \wedge y > 0\}$ if $x > y$ then $b := x ; s := y$ else $b := y ; s := x$ $\{b \geq s\}$

② $x > 0 \wedge y > 0 \Rightarrow T$

1. $wp(\text{if } x > y \text{ then } b := x ; s := y \text{ else } b := y ; s := x , b \geq s)$

$= \{wp \text{ rule for if} \cdots\}$

T $\quad (x > y \Rightarrow \overbrace{wp(b := x ; s := y , b \geq s)}^{x \geq y})$

$\wedge$

T $\quad \underbrace{\neg(x > y)}_{x \leq y} \Rightarrow wp(b := y ; s := x , \underset{y}{b} \geq \underset{x}{s})$ $\quad y \geq x$

$x > y \Rightarrow x \geq y$

$\equiv x > y \Rightarrow x > y \vee x = y$

$$\forall x \mid \underline{1} \le x \le 5 \bullet x^2 \ge 3$$

$$\equiv (1^2 \ge 3 \wedge 2^2 \ge 3 \wedge 3^2 \ge 3 \wedge 4^2 \ge 3) \wedge 5^2 \ge 3$$

$$\equiv \underbrace{\underbrace{(\forall x \mid 1 \le x \le 4 \bullet x^2 \ge 3)}_{F} \wedge \underbrace{5^2 \ge 3}_{T}}_{F}$$

$$(\forall x \mid i \leq x \leq j \cdot P(x))$$

$$\equiv (\forall x \mid i \leq x \leq j-1 \cdot P(x)) \land P(j)$$

$$(\exists x \mid i \leq x \leq j \cdot P(x))$$

$$\equiv (\exists x \mid i \leq x \leq j-1 \cdot P(x)) \underline{\lor P(j)}$$

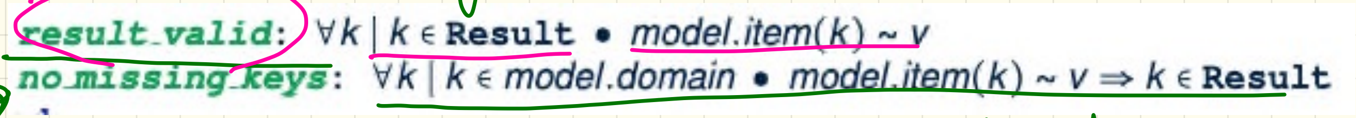
Given that the loop is not ready to exit, and that the LI has been maintained by previous iterations, the current iteration maintains the LI.

$$\{\neg B \land LI\}\ S_{body}\ \{LI\}.$$

```
from
  Sinit
invariant
  LI
until
  B
loop
  Sbody
variant
  V
end
```

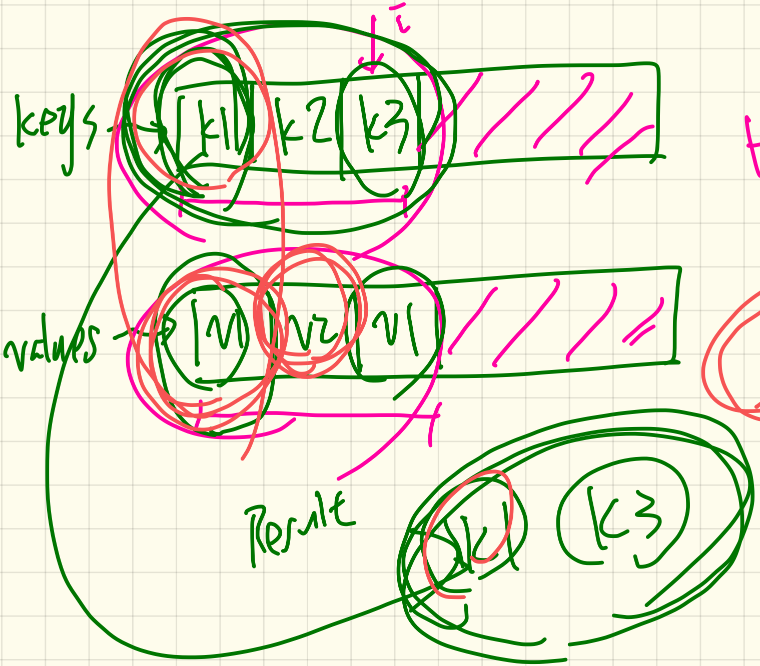
Postcondition .

get_keys(v).



result_valid: $\forall k \mid k \in \textbf{Result} \bullet model.item(k) \sim v$

no_missing_keys: $\forall k \mid k \in model.domain \bullet model.item(k) \sim v \Rightarrow k \in \textbf{Result}$



keys[j] values[j]

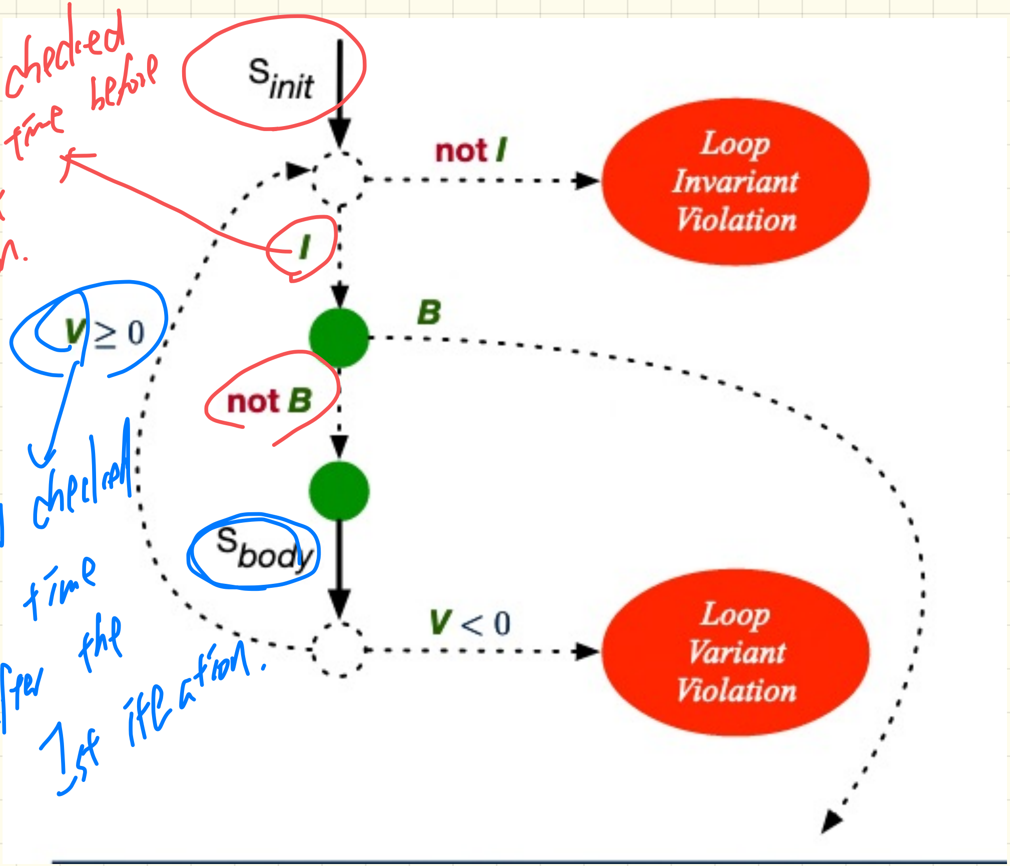
$\forall j \mid 1 \leq j < i \bullet$ or $i-1$

values[j] ~ v

and_~~implies~~

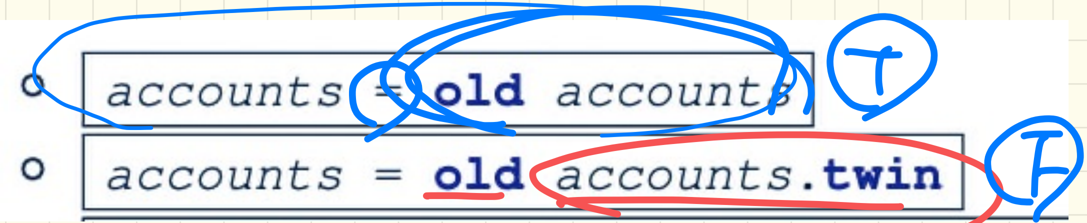
Result.has(keys[j])

ITERABLE across .



LI checked first time before the 1st iteration.

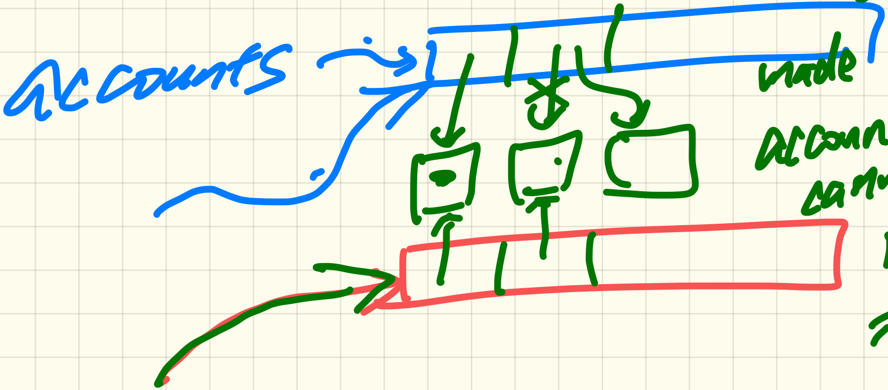
LV checked 1st time after the 1st iteration.

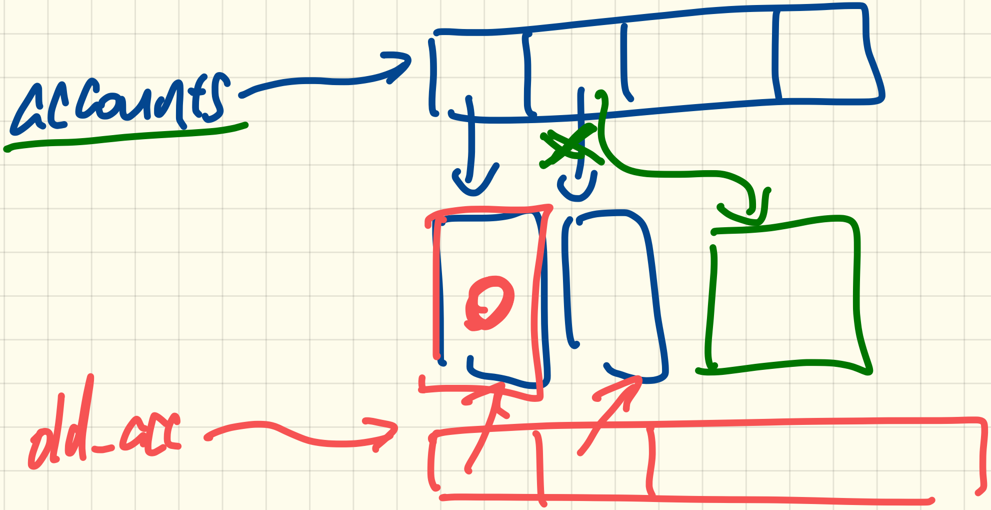


○ accounts = **old** accounts T

○ accounts = **old** accounts.**twin** F

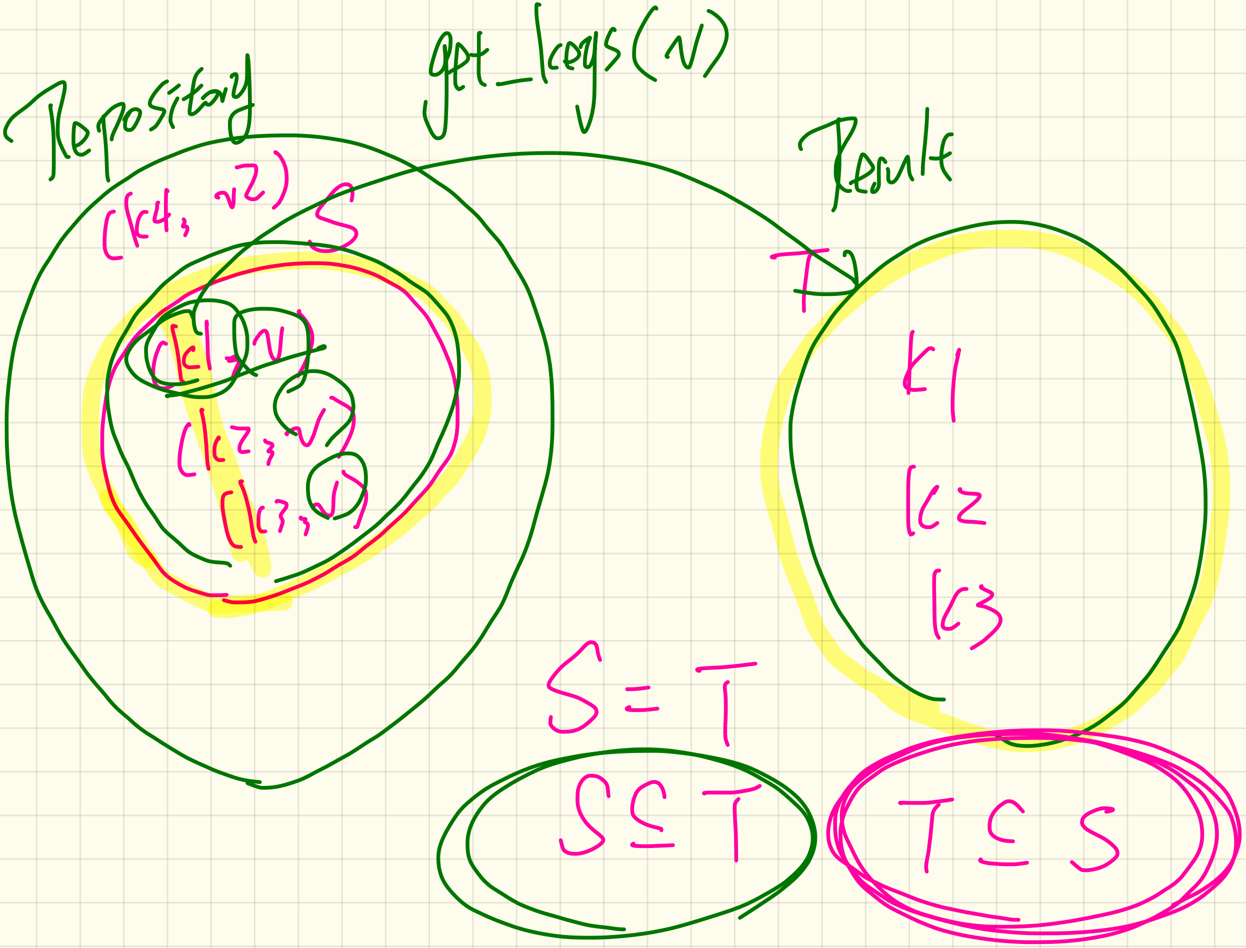
accounts ~ old accounts.twin. not appr ∵ changes made on account cannot be identified.

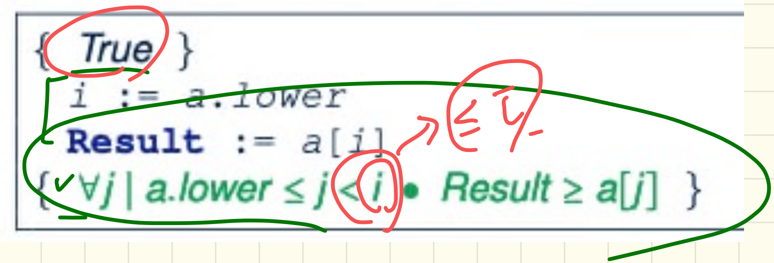




① accounts ~ old accounts.twin

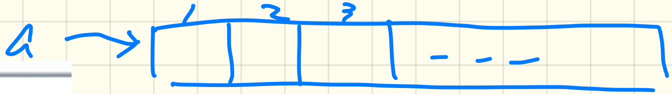
② accounts ~ old accounts.deep-twin


Repository
get_keys(v)
Result
(k4, v2)
S
(k1, v)
(k2, v)
(k3, v)
T
k1
k2
k3
S = T
S ⊆ T
T ⊆ S



{ True }
```
i := a.lower
Result := a[i]
```
$\{ \forall j \mid a.lower \le j < i \bullet Result \ge a[j] \}$

$\le \bar{i}$



$\forall x \mid F \bullet P(x) \equiv T$

1. $wp(\bar{i} := a.lower ; Result := a[\bar{i}], \forall j \mid a.lower \le j < \bar{i} \bullet Result \ge a[\bar{j}])$

a.lower

a[i]

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

$wp(\underline{\bar{i}} := \underline{a.lower} ; wp(\underline{R} := \underline{a[\bar{i}]}, \forall j \mid a.lower \le j < \cancel{i} \bullet \cancel{R} \ge a[\bar{j}]))$

$\underline{a.lower}$

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

F

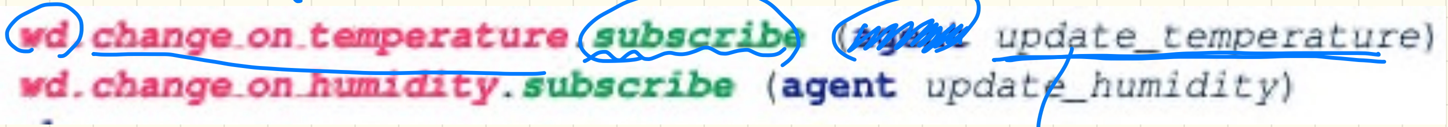
$[\forall j \mid (a.lower \le \bar{j} < a.lower) \bullet a[\bar{x}] \ge a[\bar{j}]]$

a.lower

$\equiv T$

$\cancel{a.lower} \le \bar{j} \wedge \bar{j} < \cancel{a.lower}$

EVENT



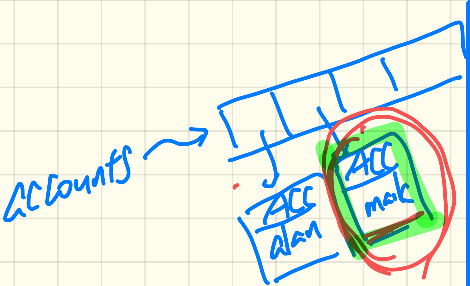
```
wd.change_on_temperature.subscribe (~~agent~~ update_temperature)
wd.change_on_humidity.subscribe (agent update_humidity)
```

Type? void

& f

*

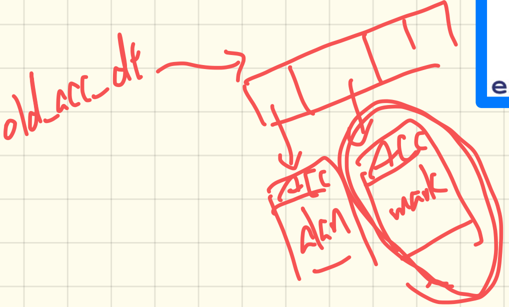
Testing of Postcondition: Exercise

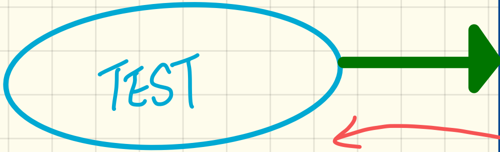


```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    do ... -- Put Correct Implementation Here.
    ensure
      ...
      others_unchanged :
        across old accounts.deep_twin as cursor
        all cursor.item.owner /~ n implies
            cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

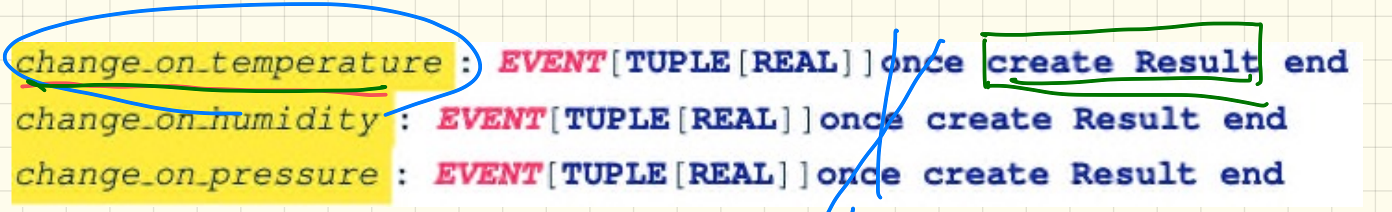
alan

balance = old balance



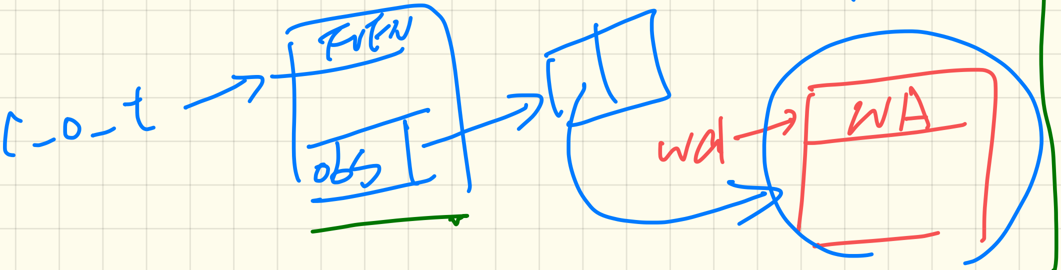


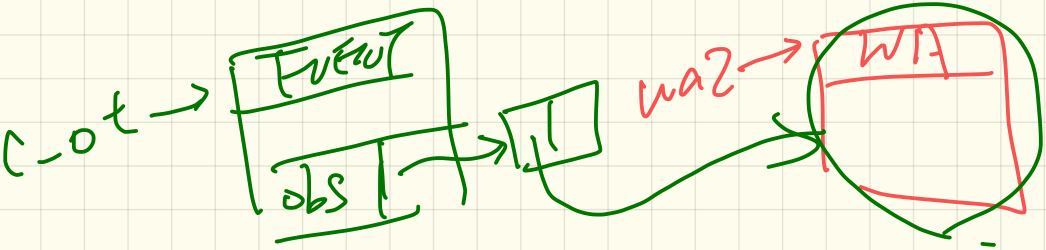
```
class BAD_BANK_DEPOSIT
inherit BANK redefine deposit end
feature -- redefined feature
  deposit_on_v5 (n: STRING; a: INTEGER)
    do Precursor (n, a)
      accounts[~~accounts.lower~~ 2].deposit(a)
    end
end
```



```
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
```

do



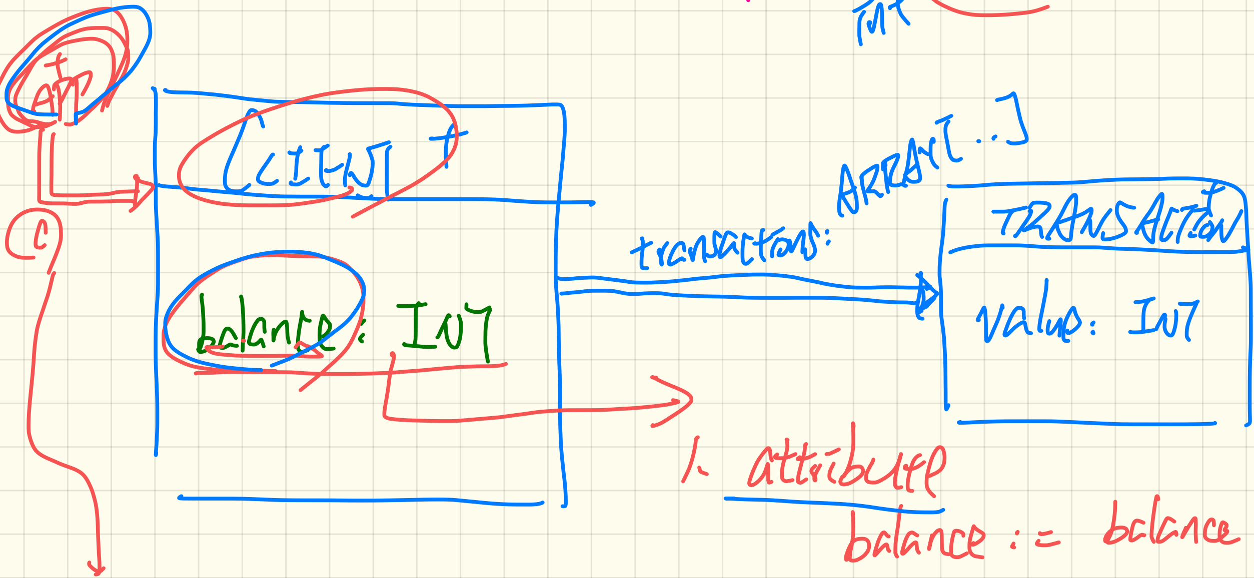


```
class Foo
create
  default_create
feature
  i : INT.

obj : Foo
create obj.d_t
create obj.
```

Uniform Access Principle

int balance.

int balance()



1. attribute

balance := balance + - -

c: CLIENT

~~c.balance~~ := 200

2. query

do across ts is

end

END

ALL THE BEST !