EECS 3311

Software Design

Winter 2019

Monday January 7

Lecture 1

EECS4312

error

① → Requirement (informal → computable)

② → Design (– architecture – specification ?)

③ → Implementation (efficiency).

④ → Deployment (Windows Mac Mobile).

Client vs. Supplier in OOP

```
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);
  } }
```

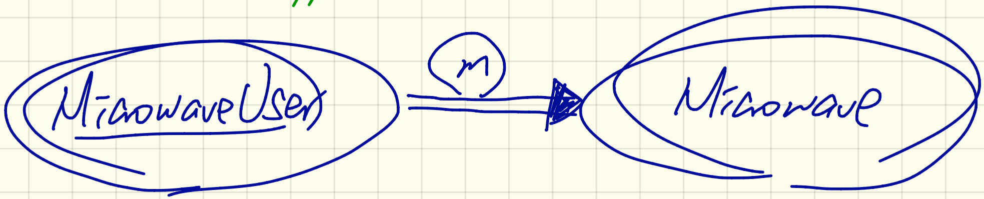
Mic.

service

context object

supplier

client-supplier relation.



→ client

```
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);
  } }
```

↳ check on obj.

Q1: Has the client followed the instructions?

We don't know ∵ obj ???

as part of the API of this method, not clear about what will be achieved.

Q2: Has the supplier fulfilled their obligations?

A Simple Design Problems: Bank Accounts

> 0

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

```java
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;
        }
}
```

Bank Accounts in Java : Version 1 Critique (1)

```java
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 1 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 1 Critique (3)

```java
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
```

Wednesday January 9

Lecture 2

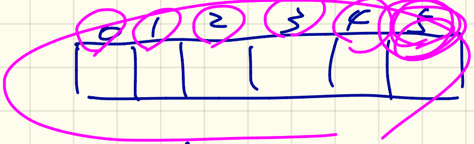
Precondition (service) condition

int divide (int x, int y) throws ______ {
  if (y == 0) {
    throw ______
  }
}

error condition

divide (x, y: INTEGER): INT
require
  y /= 0

service condition



length 6

binSearch (x , xs)

precondition: xs is sorted in non-descending order.

Math

$$\forall i, j \mid 0 \le i, j \le xs.length - 1 \bullet$$

implies

$$i < j \Rightarrow xs[i] \le xs[j]$$

$$\forall i \mid 0 \le i \le xs.length - \cancel{1}\,2 \bullet$$

dummy variable

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

0 1 2 3 4

i

cursor variable

```
across 0 |..| xs.count - 2 as i
all
  xs[i.item] <= xs[i.item
                     +1]
end
```

Bank Accounts in Java : Version 2

error condition

```java
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; }
  }
```

Bank Accounts in Java: Version 2 Critique (1)

(Compared with Version 1)

```java
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 I)

```java
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

```java
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; }
  }
```

new Account(0)

Fix 1: balance <= amount

Client

```java
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
```

Bank Accounts in Java : Version 3

```java
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";
  }
```

0

False

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)

```java
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"; }
```

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)

```java
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

acc.bal precond. 100 balance > 100

acc.withdraw(...)

acc.bal postcondition

```
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) */
```

old bal.

new bal.

```
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
```

balance = old balance − amount

Monday January 14

Lecture 3

Bank Accounts in Java: Version 4

(with an evil supplier)

```java
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"; }
```

50

asset

this.balance

oldBalance

int oldBalance = this.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) */
```

bal == 100

bal == 150

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

Bank Accounts in Java : Version 5

```java
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

cache

+

this.balance = ~~50~~ 150

~~50~~ 150

100

50

F

~~50~~ 150 == 100 - 50

T

int divide (int x, int y)

ensure

Result

$$\boxed{\text{Result} * y = x}$$

boolean binSearch (int x, int[] xs)

ensure

such that it is the case

$$\boxed{\text{Result} = (\exists i \mid 0 \le i < xs.\text{length} \bullet xs[i] = x)}$$

Result = (**across** 0 |..| (xs.length − 1) **as** i
some xs [i.item] = x **end**)

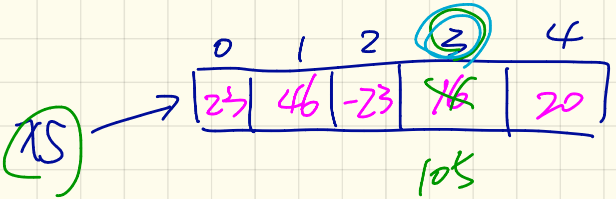
void change (int[] xs , int i , int x)

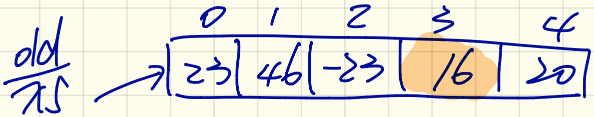
<u>require</u>

$0 <= i$ <u>and</u> $i < xs.length$

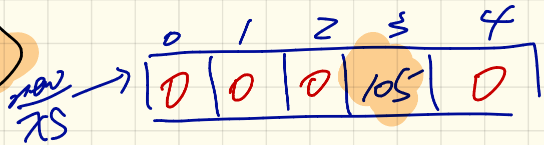
<u>ensure</u>

changed : xs[i] = x





change(xs , 3 , 105)



void change (int[] xs , int i , int x)

require

$0 <= i$ and $i < xs.length$

ensure

changed : $xs[i] = x$

$j == 0$

others unchanged :

change(xs , 3 , 105) ✗

$\forall j \mid 0 \le j < i \lor i+1 \le j < xs.length \cdot$

$\boxed{xs[j]} = \boxed{old}\ xs[j]$

old xs →

0	1	2	3	4
23	46	-23	16	23

$23 == 0$

new xs →

0	1	2	3	4
0	0	0	105	0

$\boxed{\forall j \mid 0 \le j < xs.length \cdot j \ne i \Rightarrow xs[j] = old\ xs[j]}$

$$\forall j \mid \boxed{0 \le j < xs.length} \bullet \boxed{j \ne i \Rightarrow xs[j] = \underline{old}\ xs[j]}$$

across 0 |..| (xs.length − 1) **as** j → Integer Cursor

all

~~j~~ /= i **implies** xs[~~j~~] = **old** xs[~~j~~]

j.item j.item j.item

end

boolean allPositive (int[] xs)

$-1 - 0 + 1 = 0$

$[1, 10]$

$10 - 1 + 1$

$[x, y]$

$y - x + 1$

ensure

Result = (across $\boxed{0}$ |..| $\boxed{xs.length - 1}$ as i

all xs[~~i~~] > 0

i.item

end)

allPositive (<< 1, 2, 3, -4 >>) F

→ allPositive (<< >>)

$\underline{\text{allPos}}(\langle\langle\ \rangle\rangle)$

$\text{somePos}(\langle\langle -2, 3, -4, -8 \rangle\rangle)$

$\text{somePos}(\langle\langle\ \rangle\rangle)$ F T

$$(\forall x \mid x \in \emptyset \cdot P(x)) \equiv \text{True}.$$

↳ "there is no such element $x \in \emptyset$ that can falsify $P(x)$"

witness

$$(\exists x \mid x \in \emptyset \cdot P(x)) \equiv \text{False}$$

↳ "there is no witness in $\emptyset$ that can make $P(x)$ true.

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) */
```

w. C

```
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
```

Design by Contract in Eiffel

```
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
```

Contract View

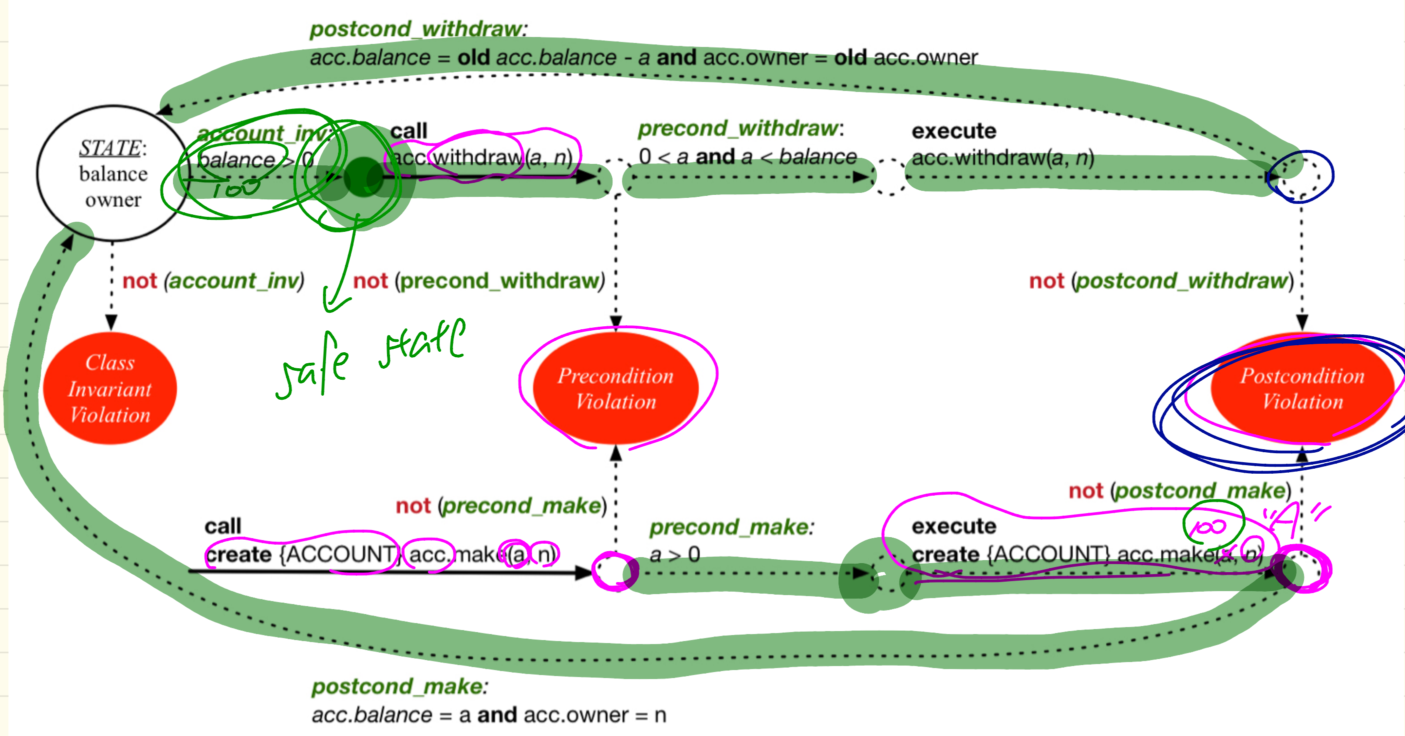
Implementation View

```
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
```

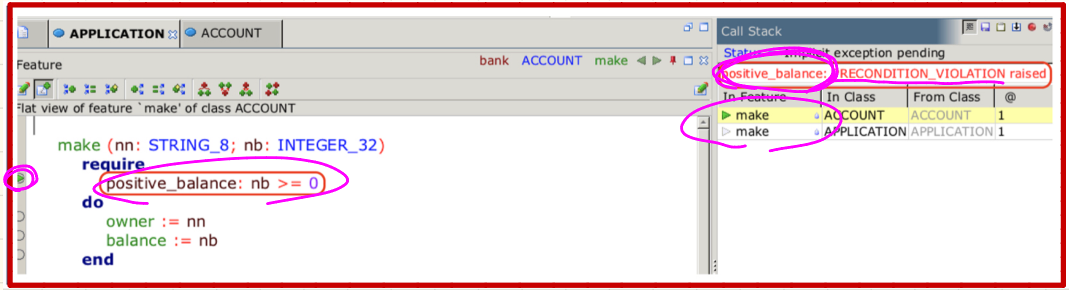
Specification

Implementation

Runtime Monitoring of Contracts



Precondition Violation (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
```

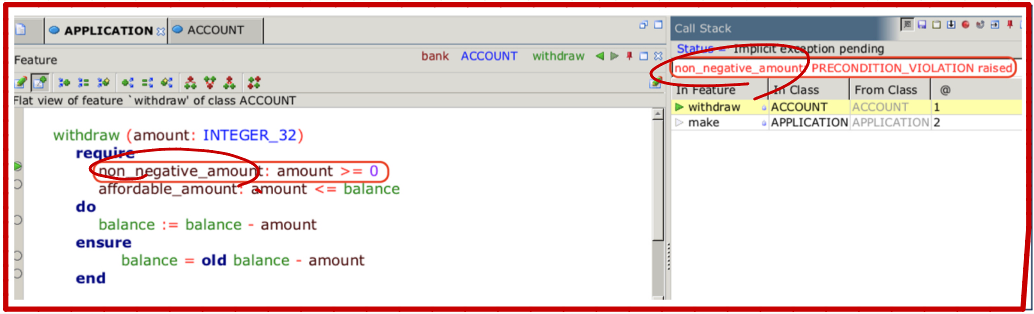
F

-10

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 (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
```

-1000 T

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 (3)

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

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
```

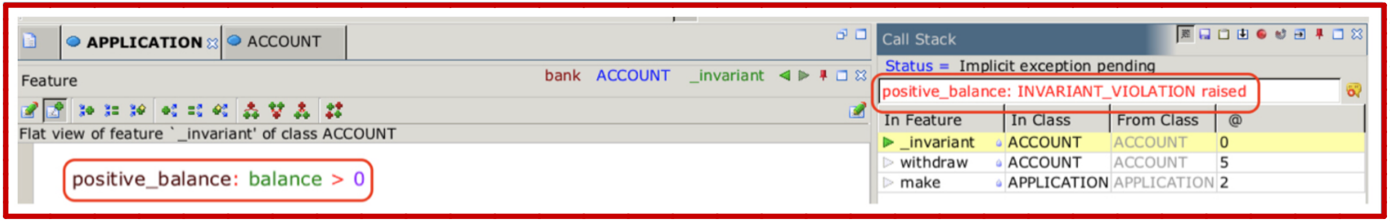
150 100

F

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



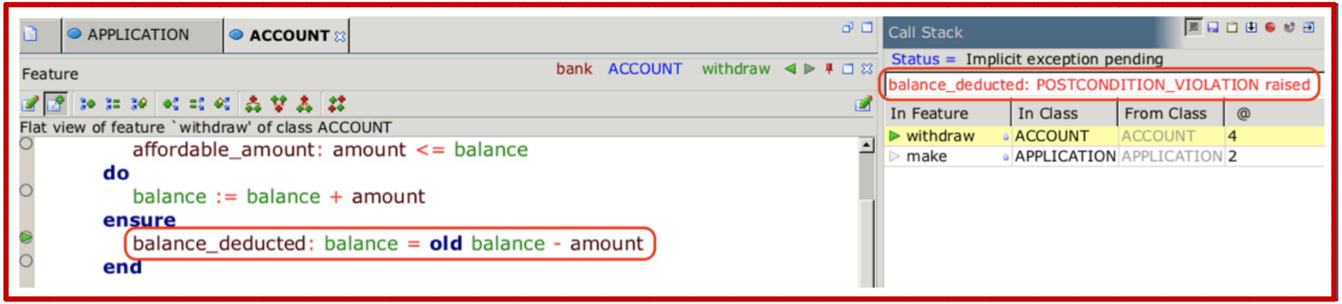
Supplier

```
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



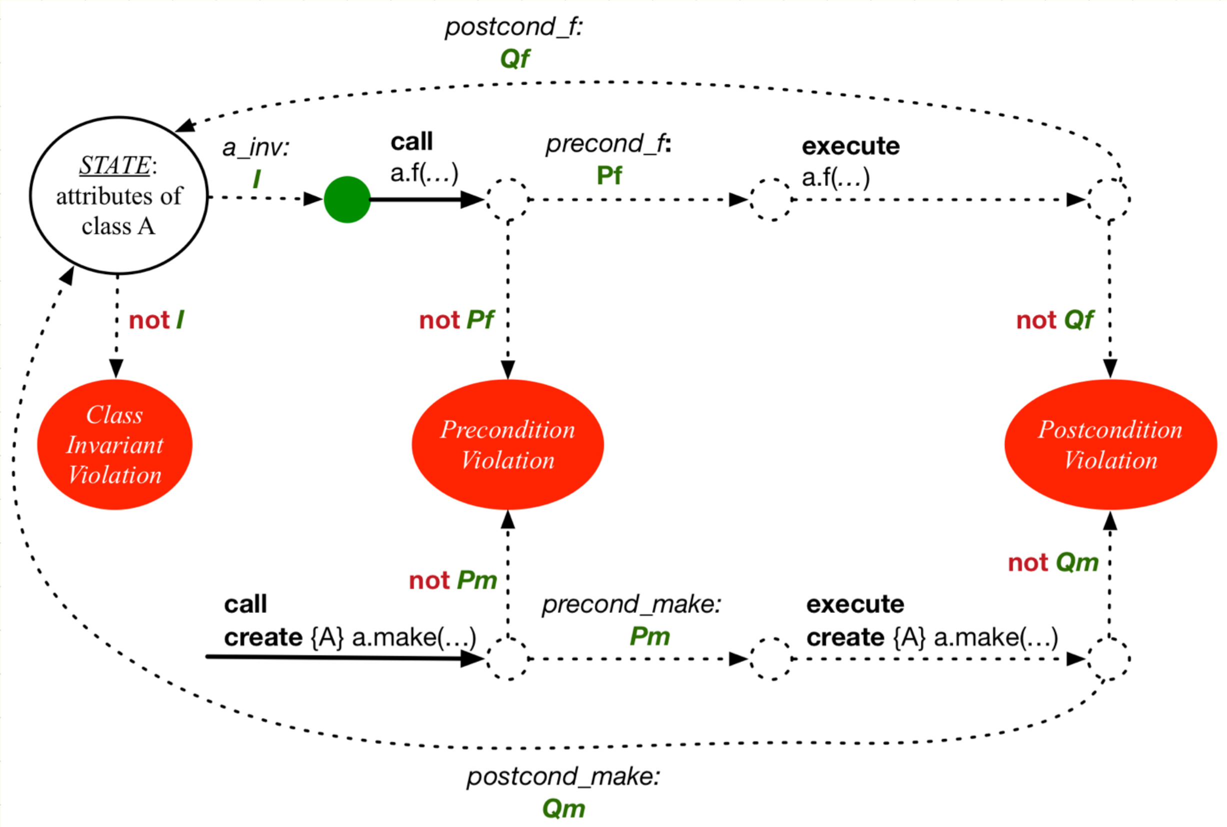
Supplier

```
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



Math = Eiffel

:=

f : INTEGER

local
. . –

require
– –

do
– – .

ensure – – –
end

require

Imp:
local

do

ensure

end

int i;

int i = 5;

local
  i : INTEGER
do
  i := 5

i : INTEGER

ε

Logic

$\neg p$

$p \wedge q$

$p \vee q$

Java

p && q

p || q

f(int i, int[] xs): int

require:

0 <= i && i < xs.length && xs[i] > 0

0 <= i && xs[i] > 0 && i < xs.length

p	q	p ∧ q
T	T	T
T	F	F
F	T	F
F	F	F

Wednesday January 16

Lecture 4

- Lab 2 posted
- Lab 1

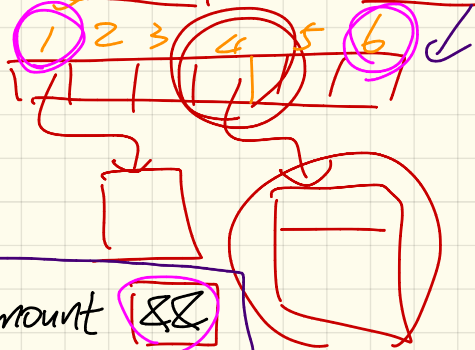
Office Hours

W/F 3pm ~ 5pm

Short-Circuit Operators vs. Logical Operator

7

withdraw (accounts : ARRAY[ACCOUNT] ; i : INT ; amt: INTEGER)



Fix: c1 && c3 && c2

require

F

accounts.lower <= i &&

c1

short-circuit

accounts[i] . balance > amount &&

i <= accounts.upper

c2

c3

6

accounts[7] .balance > amt

1 <= 7 T

c1 and then c3 and then c2

$\vee$

Logic

or

$$p \wedge q \wedge r$$

$$\equiv p \wedge r \wedge q$$

$$\|$$

Prog. (SCE)

$$p \mathbin{\&\&} q \mathbin{\&\&} r$$

$$\stackrel{?}{\equiv} p \mathbin{\&\&} r \mathbin{\&\&} q$$

$\wedge$

p and q

even if p is false, still evaluate q (no SCE).

p and then q

p or else q

a : ARRAY [STRING]

rebase

Create a.make_empty

a.lower 1

a.upper 0

a.force("A", a.upper + 1)

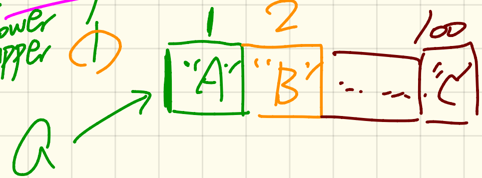
a.lower 1

a.upper 1

a.force("B", a.upper + 1)

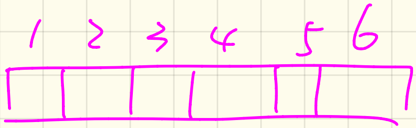
a.lower 1

a.upper 2



[a.lower, a.upper]

a.upper − a.lower + 1

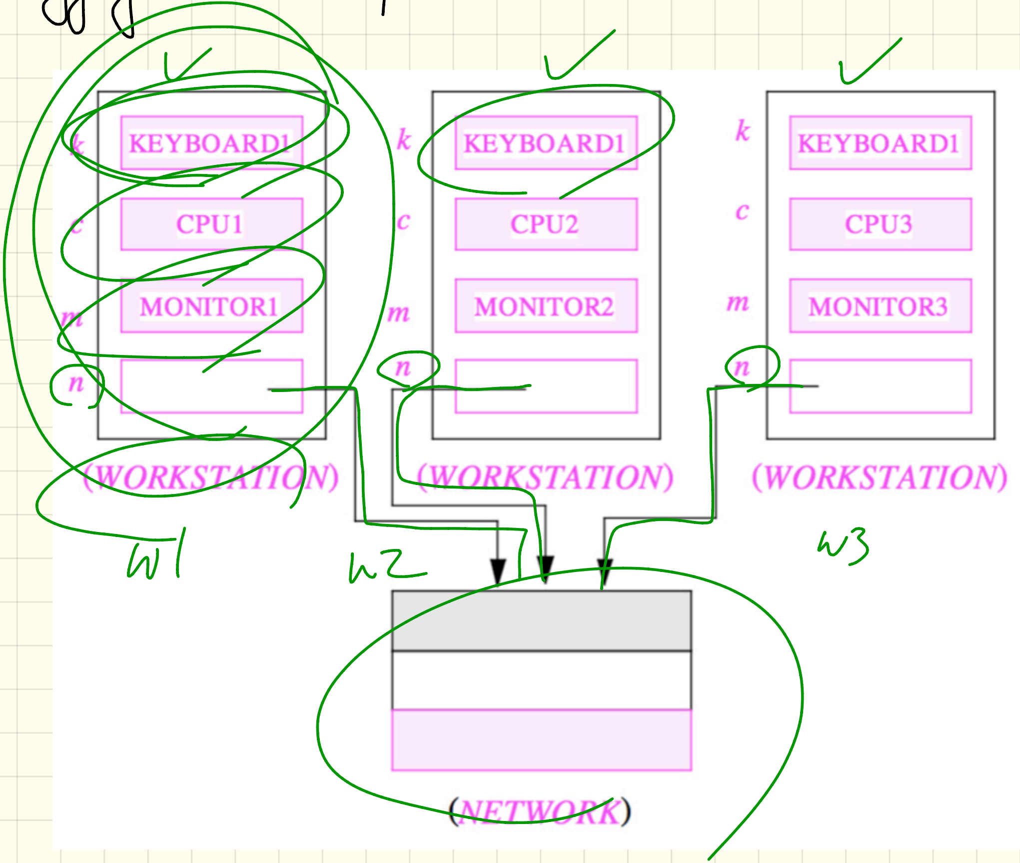


[1, 6]

6 − 1 + 1

a.force("C", 100)

Modelling: Aggregation vs. Composition



Expanded Type for Composition

```
class KEYBOARD ... end class CPU ... end
class MONITOR ... end class NETWORK ... end
class WORKSTATION
  k: expanded KEYBOARD
  c: expanded CPU
  m: expanded MONITOR
  n: NETWORK
end
```

k cannot be shared.

```
expanded class KEYBOARD ... end
expanded class CPU ... end
expanded class MONITOR ... end
class NETWORK ... end
class WORKSTATION
  k: KEYBOARD
  c: CPU
  m: MONITOR
  n: NETWORK
end
```

Use of Expanded Type

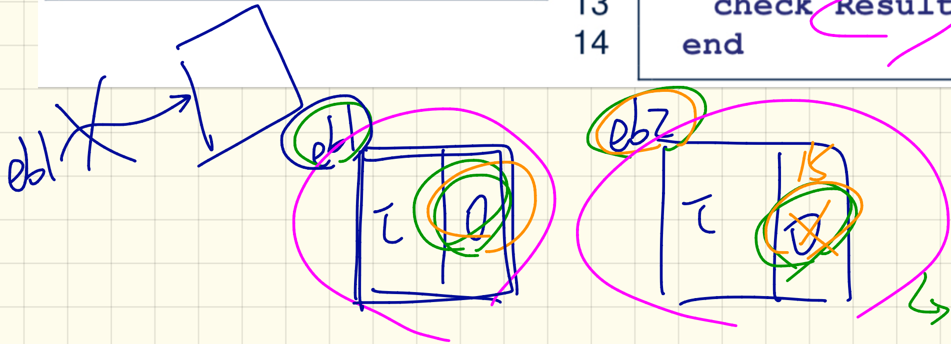
eb1 == eb2

```
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
```

comp. contents

eb1.i vs. eb2.i



obj1 = obj2

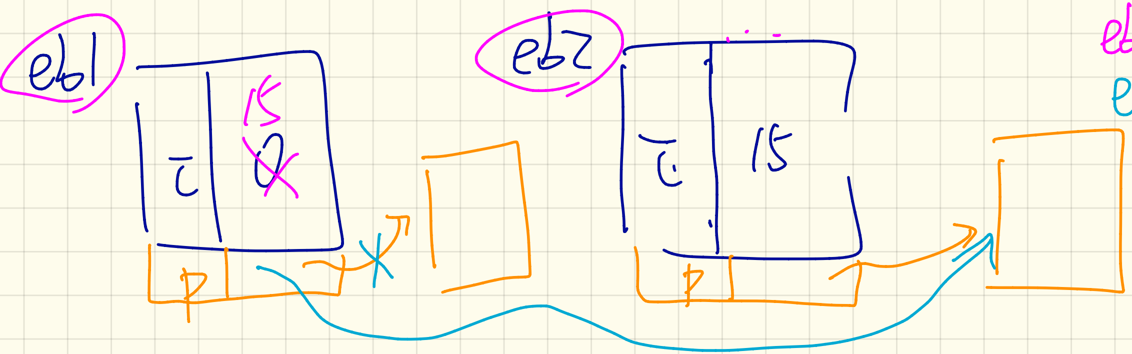
↳ obj1, obj2 Ref. T. → compare addresses.

↳ obj1, obj2 Exp. T → compare contents

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

```
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
```

eb1 := eb2

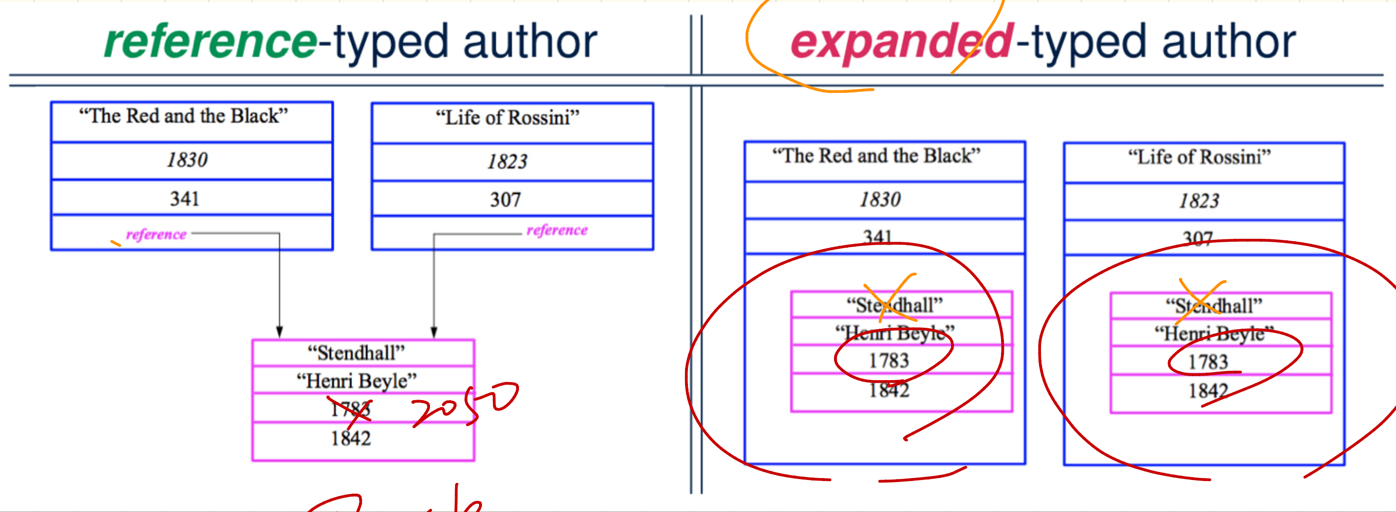


eb1.i := eb2.i

eb1.p := eb2.p

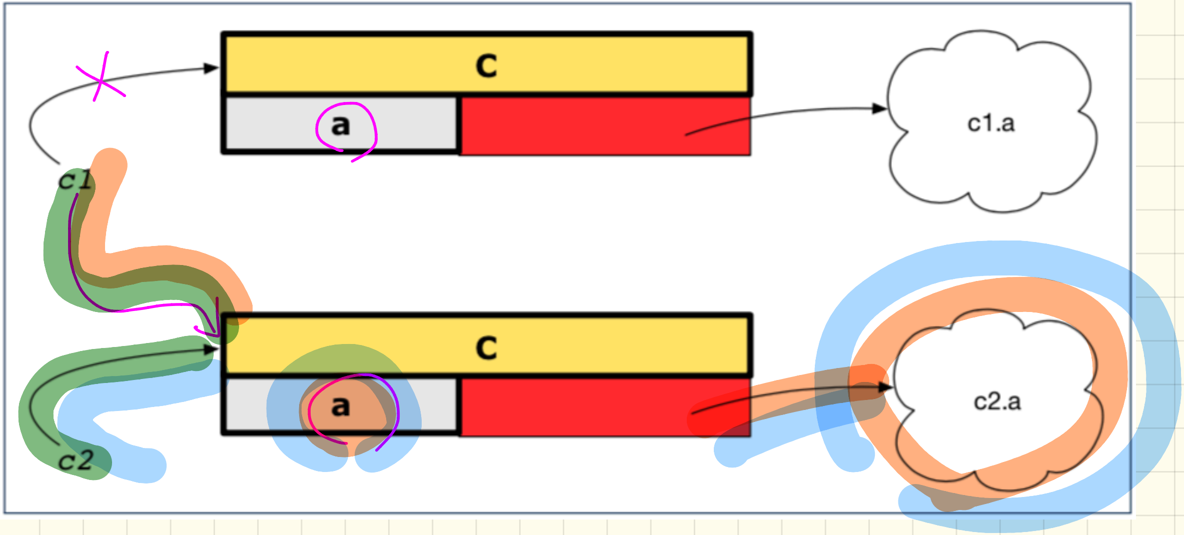
```
class B
    .   -
end

    local
        v1: B
        v2: expanded B
    do
        v1 = v2
```

Reference or Expanded Type



Single Choice Principle

Reference Copy : $c_1 := c_2$

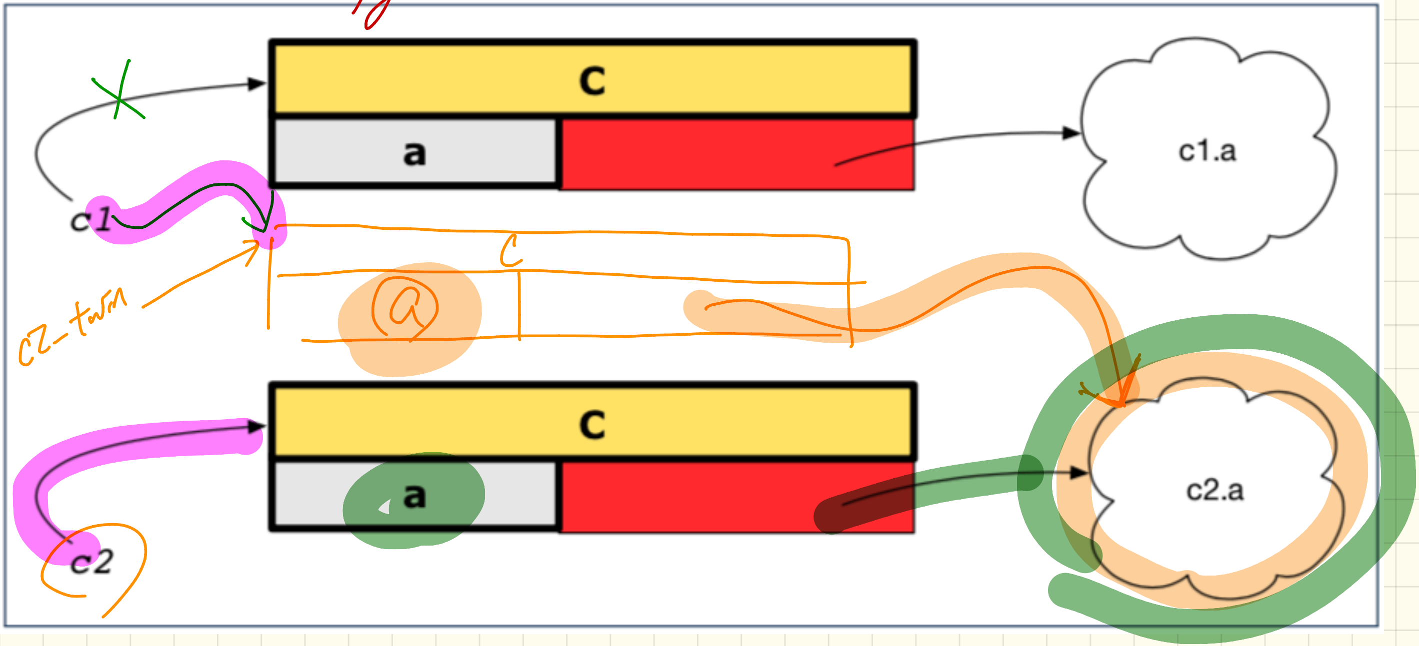


① $c1 = c2$ T

→ ② $c1.a = c2.a$ T

Shallow Copy : $c_1 := c2.twin$

1st-level copy

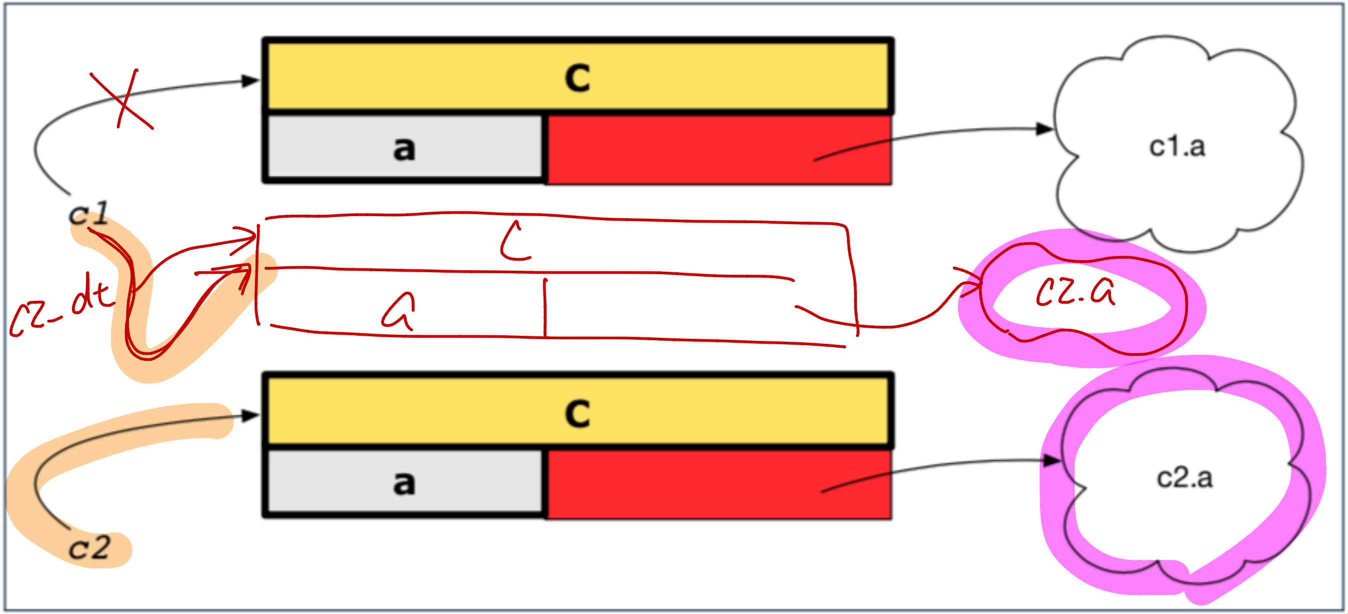


c2_twin.a := c2.a

① c1 = c2 F

② c1.a = c2.a T

Deep Copy : $c1 := c2.deep\_twin$



$c2\_dt.a := c2.a.deep\_twin$

① $c1 = c2$ F

② $c1.a = c2.a$

Ref. vs. Shallow vs. Deep Copies

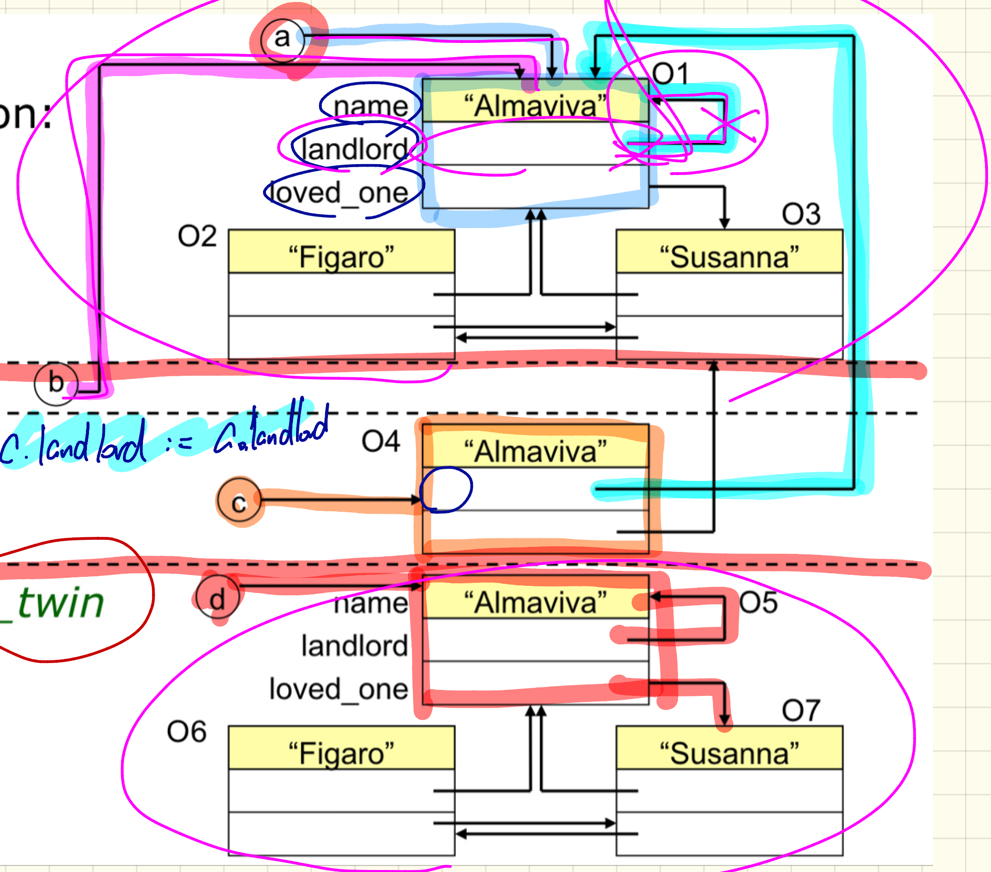
- Initial situation:
- Result of:

b := a

c := a.twin

c.landlord := a.landlord

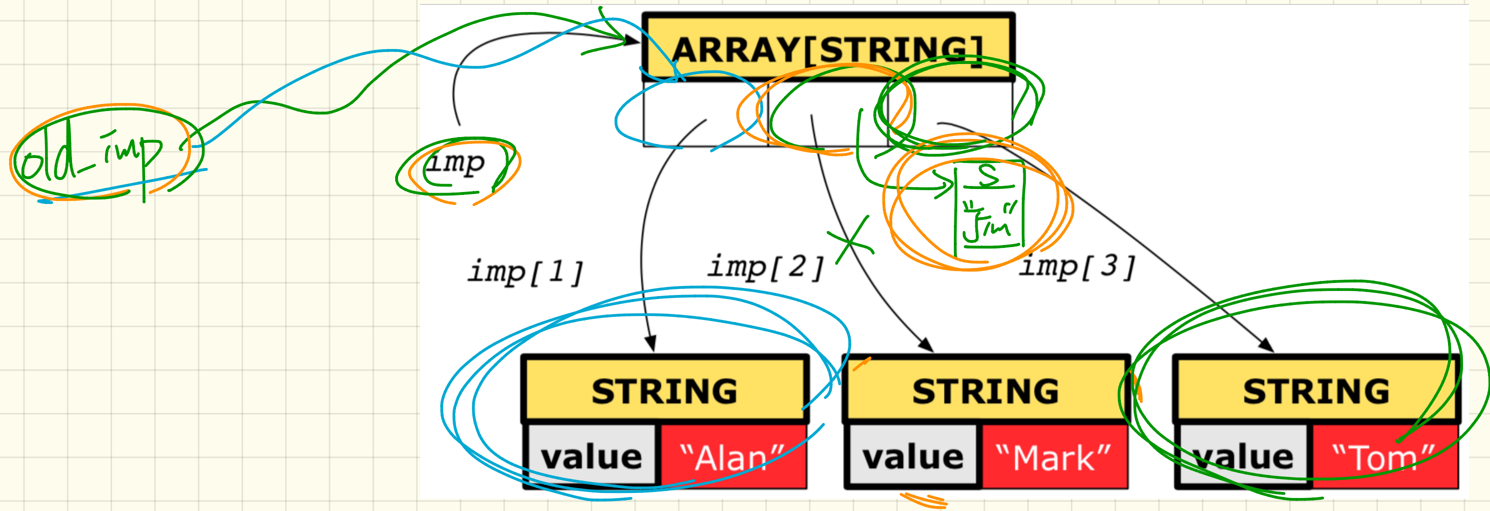
d := a.deep_twin



Copying Collection Objects: Reference Copy & Make Changes

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

1 2 3



Monday January 21

Lecture 5

Copying Collection Objects : Reference Copy & Make Changes

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

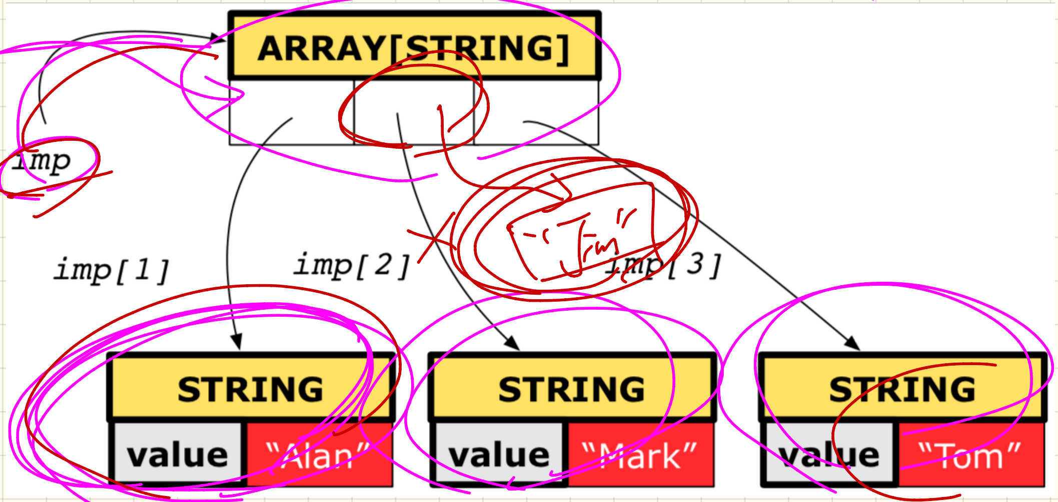
Ref copy

1 2 3

imp[1] ~ 1

old_imp [j.item]

∀



old_imp

old_imp[2] ~ =

imp[2]

T

T

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

```
1 | old_imp := imp.twin
2 | Result := old_imp = imp  -- Result = false
3 | imp[2] := "Jim"
4 | Result :=
5 |   across 1 |..| imp.count as j
6 |   all imp [j.item] ~ old_imp [j.item]
7 |   end -- Result = false
```

← old_imp[2] ?

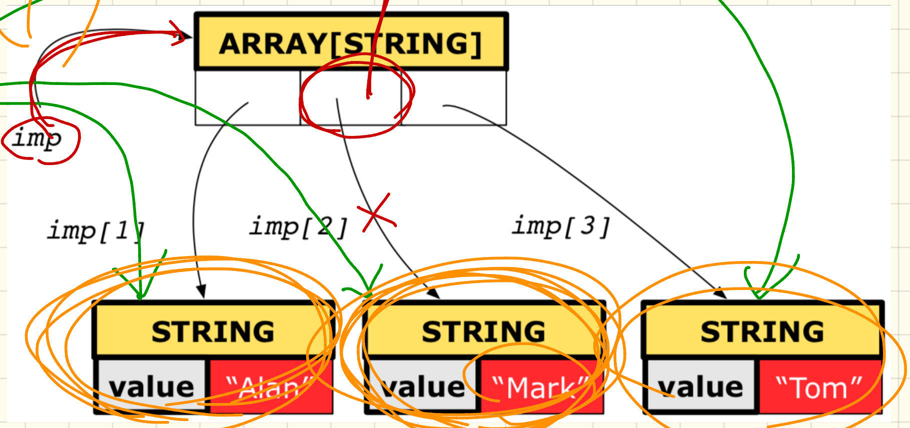
After L3:

old_imp[2] = imp[2]   F

"Jim"

T F T   F

old_imp →



old_imp[1] := imp[1]

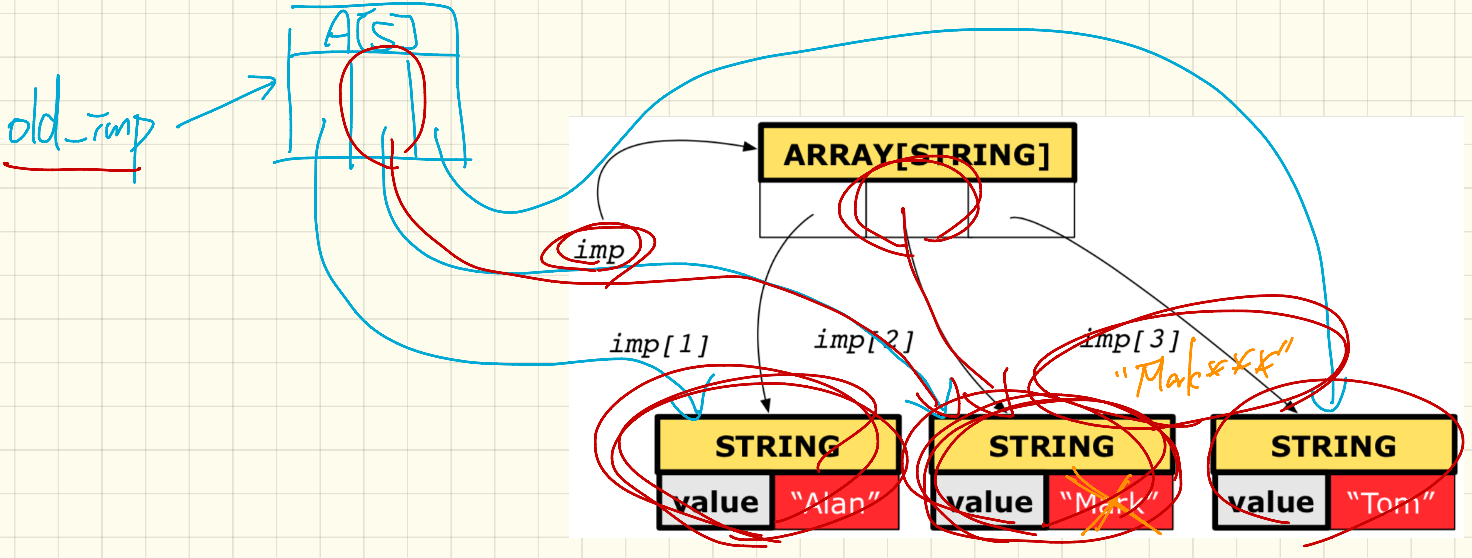
Copying Collection Objects: Shallow Copy & Make 2nd-level changes

```
1  old_imp := imp.twin
2  Result := old_imp = imp   -- Result = false
3  imp[2].append ("***")
4  Result :=
5    across 1 |..| imp.count as j
6    all imp [j.item] ~ old_imp [j.item]
7    end -- Result = true
```

After L3.

imp[2] = old_imp[2]

T



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

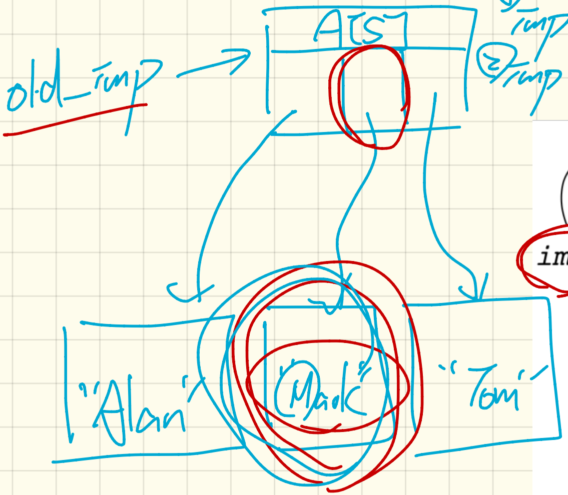
```
old_imp := imp.deep_twin
Result := old_imp = imp   -- Result = false
imp [2] := "Jim"
Result :=
  across 1 |..| imp.count as j
  all imp [j.item] ~ old_imp [j.item] end   -- Result = false
```

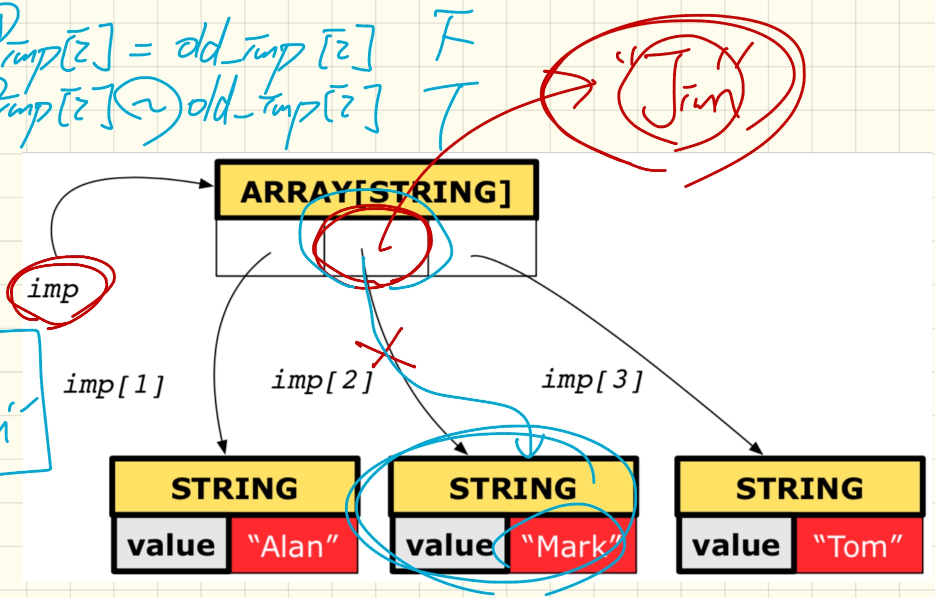
imp[2] = old_imp[2] → F

imp[2] ~ old_imp[2] F

① imp[2] = old_imp[2] F

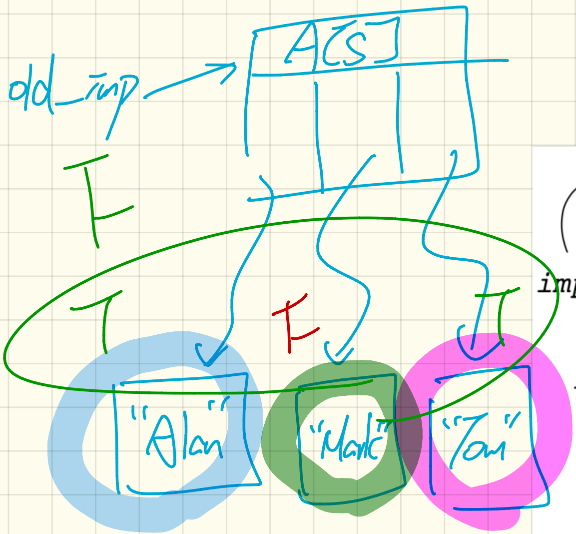
② imp[2] ~ old_imp[2] T

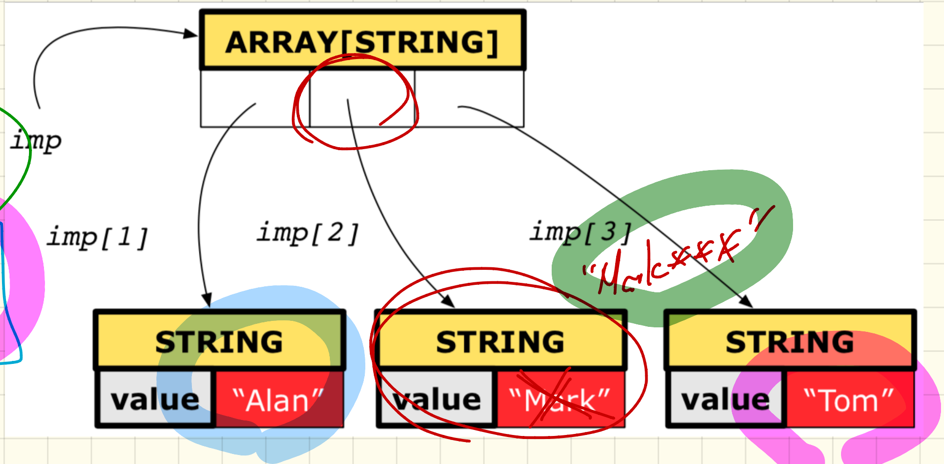




Copying Collection Objects : Deep Copy & Make 2nd-level changes

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





Contract View

f

require

ensure

old balance

... old expr ...

end

balance = old balance - a

Runtime Contract Checks

call f

old_balance := balance

e.g.

check precond. of f

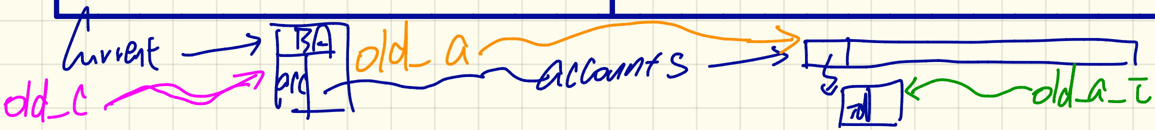
cache old expressions

execute imp. of f

check postcond. of f

Caching Values for old Expressions in Postconditions

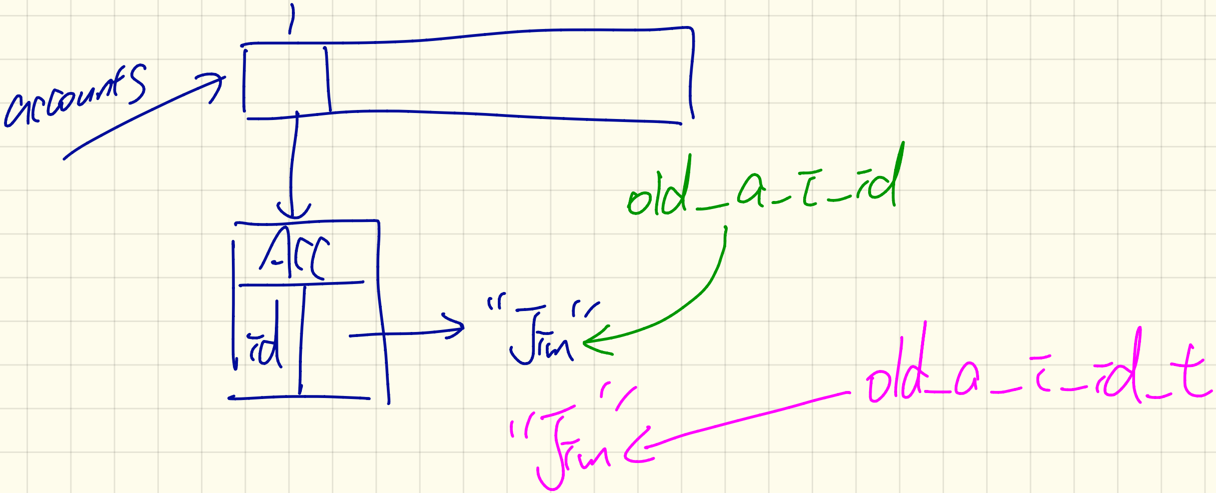
ensure	How to Cache at Runtime?
old balance = balance - a	old_balance := balance
old accounts[i].id	old_accounts_i_id := accounts[i].id
(old accounts[i]).id	old_a_i := accounts[i]
(old accounts)[i].id	old_a := accounts
(old Current)accounts[i].id	old_c := Current



✓ old accounts[i].id   old_a_i_id := accounts[i].id

✓ old accounts[i].id.twin   old_a_i_id_t := accounts[i].id.twin

✓ old accounts[i].id.deep_twin



$$\forall s \mid s \in EECS3311 \bullet s.pass$$

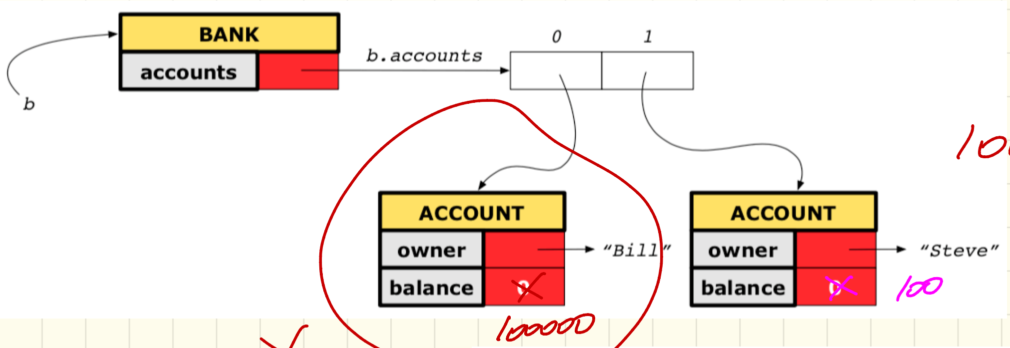
$$\equiv \neg (\exists s \mid s \in EECS3311 \bullet \neg s.pass)$$

```
across accounts as acc
  some
      acc.item.ower ~ n
  end
```

```
not (across accounts as acc
  all
     not (acc.item.ower ~ n)
  end
)
```

Version I: Incomplete Contracts, Correct Implementation

b.deposit("Steve", 100)



100 = 0 + 100

T

not caught.

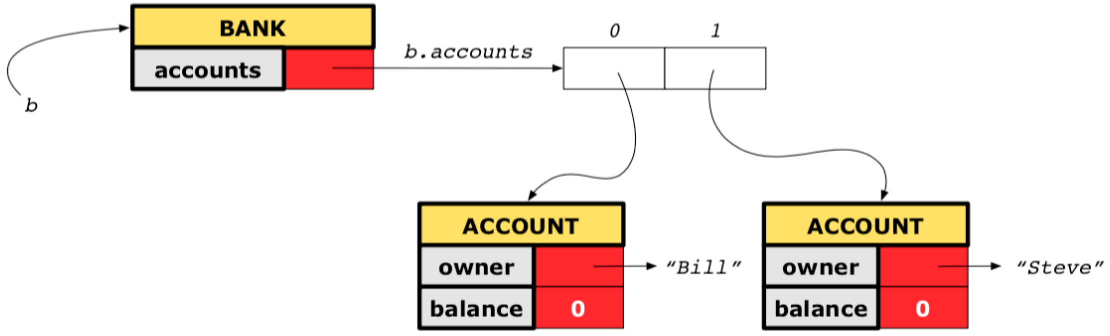
```
class BANK
  deposit_on_v1 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      from i := accounts.lower
      until i > accounts.upper
      loop
        if accounts[i].owner ~ n then accounts[i].deposit(a) end
        i := i + 1
      end
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

to be cached in pre-state.

"Steve"

Version 2 : Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)

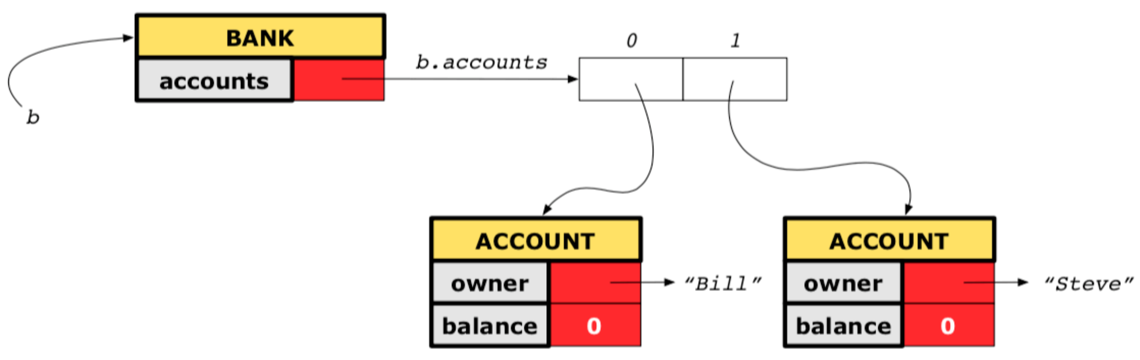



```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1

      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

Version 3: Complete Contracts, Wrong Implementation

(Reference Copy)

b.deposit("Steve", 100)



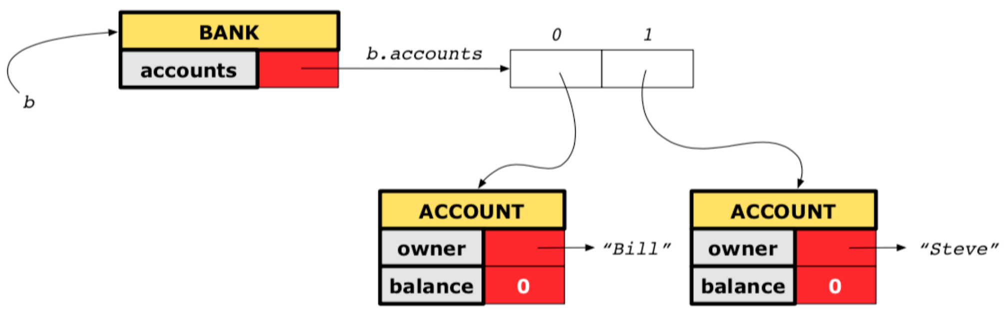

```
class BANK
  deposit_on_v3 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of(n).balance = old account_of(n).balance + a
      others_unchanged :
        across old accounts as cursor
        all cursor.item.owner /~ n implies
              cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

Wednesday January 23

Lecture 6

Version I: Incomplete Contracts, Correct Implementation

b.deposit("Steve", 100)

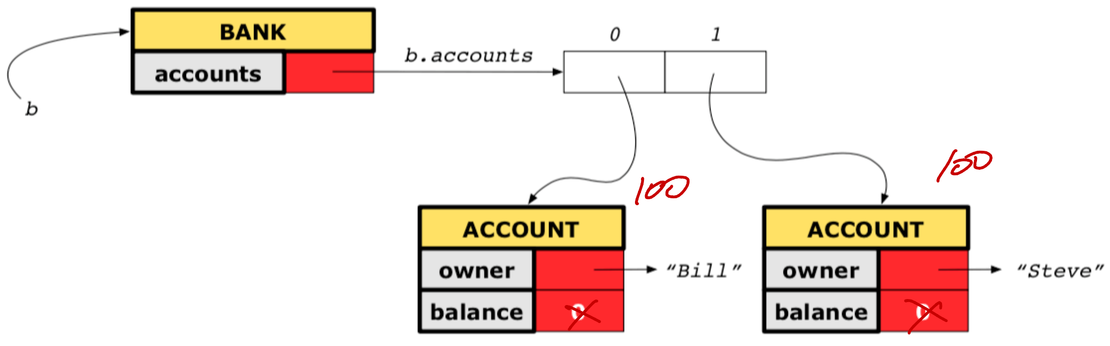


correct

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

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)




```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1

      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

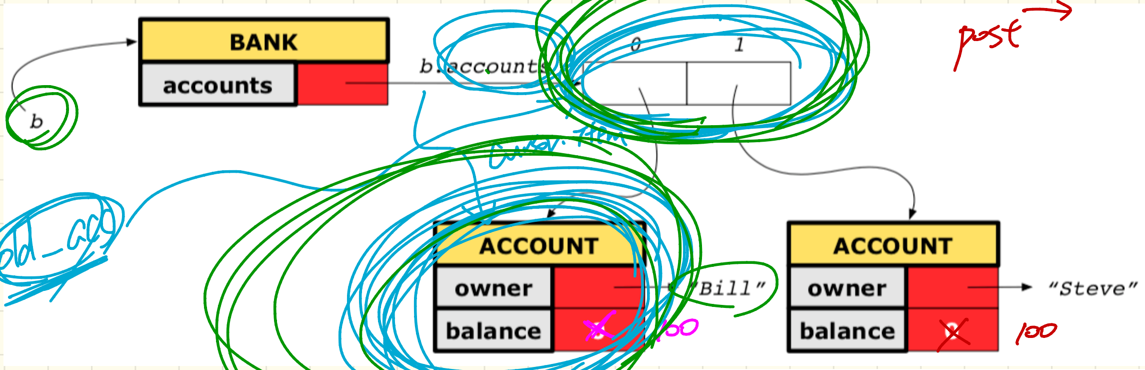
Version 3: Complete Contracts, Wrong Implementation

⟨Reference Copy⟩

pre →

b.deposit("Steve", 100)

post →



old_acc := accounts

```
class BANK
  deposit_on_v3 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of(n).balance = old account_of(n).balance + a
      others_unchanged :
        across old accounts as cursor
        all cursor.item.owner /~ n implies
          cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

old_acc

"Bill"

T

Current.

Use of across in Postcondition

Version 1

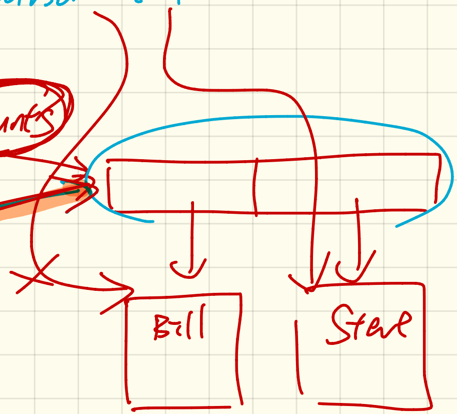
old_a := accounts

```
across old accounts as cursor
all
  cursor.item.owner /~ n
  implies
  cursor.item ~ Current.account_of(n)
end
```

cursor.item.owner

old_a ~

post-state after executing deposit_on fn.



Cursor.item ~ Current.ac_of(n)

Version 2

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

cursor.item.owner

→ empty bank

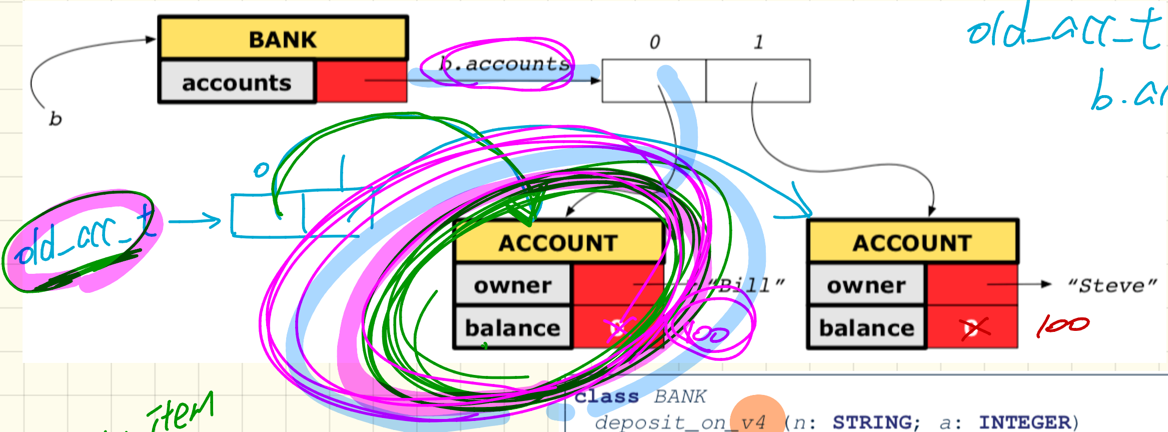
lower	upper
0	-1
1	0

Version 4 : Complete Contracts, Wrong Implementation

⟨ Shallow Copy ⟩

b.deposit("Steve", 100)

old_acc_t[0] := b.accounts[0]



cursor.item

old_acc_t := accounts.twin

```
class BANK
  deposit_on_v4 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
      others_unchanged :
        across old accounts.twin as cursor
        all cursor.item.owner /~ n implies
          cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

"Bill"

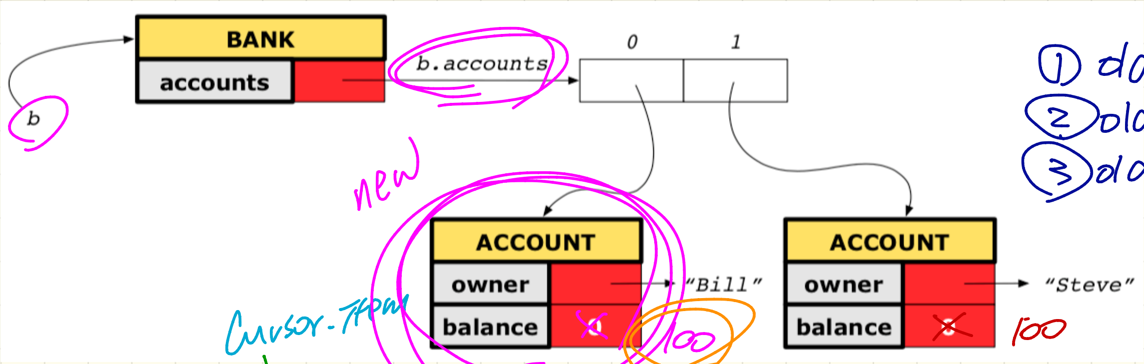
current

<Deep Copy>

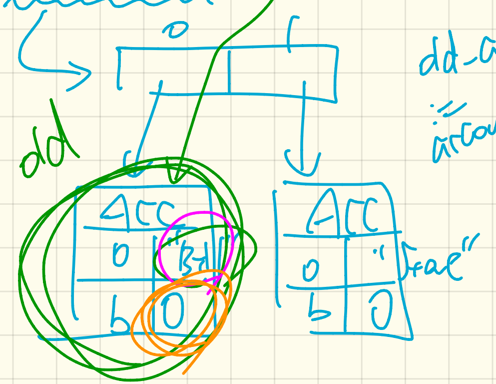
Version 5: Complete Contracts, Wrong Implementation

b.deposit("Steve", 100)

1. old accounts
2. old accounts.twin
3. old accounts.d_t



old_acc_dt



old_acc_dt := accounts.d_t

```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
  do
    -- same loop as in version 1
    -- wrong implementation: also deposit in the first account
    accounts[accounts.lower].deposit(a)
  ensure
    num_of_accounts_unchanged: accounts.count = old accounts.count
    balance_of_n_increased:
      account_of (n).balance = old account_of (n).balance + a
    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
```

"Bill"

F

current.

class Foo

attributes

queries

f ( ) :

ensure

Result .

Complete Postcondition: Exercise

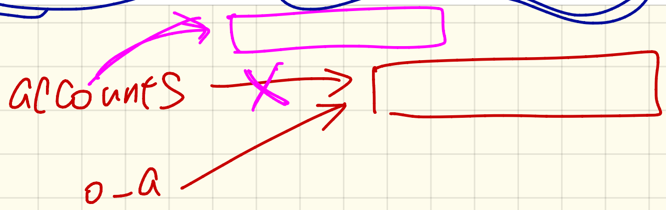
(assuming accounts is not re-assigned) Account

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:

trivially true

- `accounts = old accounts` [×] trivially F.
- `accounts = old accounts.twin` [×] t. f.
- `accounts = old accounts.deep_twin` [×]
- `accounts ~ old accounts` [×] t. t.
- `accounts ~ old accounts.twin` [×] only appropriate if the change is at 1st level e.g. accounts[1] := new account.
- `accounts ~ old accounts.deep_twin` [✓]



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
```

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

Writing Postcondition : Exercise

```
is_positive (x : INTEGER) : BOOLEAN
  ensure
    ~~Result := x > 0~~
    x > 0
```

−2   0

Result := False

−2 > 0

F

Result = (x > 0)

F   F

T

Result := ((x * −1) > 0)

T

post. con. violate

Result implies x > 0

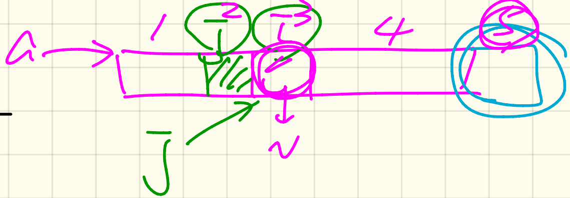
F

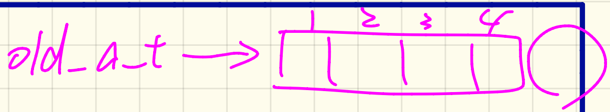
ensure

=

Result ~~:=~~ x > 0

Writing Postcondition : Exercise






```
a: ARRAY [INTEGER]
change_at (i: INTEGER ; v: INTEGER)
                         a.force (v, a.count + 1)
  ensure
  ①   across a.lower |..| a.upper as j
      all
        j.item = i implies a[j.item] = v
        and
        j.item /= i implies a[j.item] = (old a.twin)[j.item]
  ②   end
```

a.count = old a.count

Writing Postcondition: Exercise

a → | 1 | 3 | -2 | 4 |

Result → | 1 | 3 | 4 |

```
all_positive_values(a: ARRAY[INTEGER]): ARRAY[INTEGER]
    ensure
        across Result as x
        all
            x.item > 0 and a.has (x.item)
        end
```

Result → | 10 | 23 | 46 |

Result ↳ | 1 | 3 |

S: Result vs T: all positive numbers

$$S = T \iff S \subseteq T \land T \subseteq S$$

Monday January 28

Lecture 7

Writing Postcondition : Exercise

a → [3 | -1 | 2 | -4 | 5]

Result → [3 | 2 | 5]

postcond! ← Result → [3 | 2 | 5 | 23]

```
all_positive_values (a: ARRAY[INTEGER]): ARRAY[INTEGER]
  require
    a contains no duplicate.
  ensure
    post_con_1:
      across Result as x
      all
        x.item > 0 and a.has( x.item )
      end
    post_con_2:
      across a as x
      all
        x.item>0 implies Result.has(x.item)
      end
```

occurrences

Result → [3 | 2 | 5 | 3]

Result → 1

Result → [3]

Result → [4 | 1 | 6 7 | 8 | 9]

Result → 1

$$\overbrace{\underbrace{\{x \mid x \in a \cdot x > 0\}}_{\text{all pos. elements in } a}}^{S} = \overbrace{\{y \mid y \in Result\}}^{T}$$

$$T \subseteq S$$

$$\wedge \; S \subseteq T.$$

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
```

ss : S_S

as : A_S

ss : STACK[ ]

as : STACK[ ]

STRING

ACCOUNT

```
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
```

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
```

STRING declare [INTEGER] A S S A S A references

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
```

STRING

```
class MY_COLLECTION [G]

    imp : ARRAY [G]

end
```

s.push ("A")
s.push (2)
s.push (create {BST}...)

↓ 100 kinds of elements in stack

---

s : MY_COLLECTION [ANY]

~~s.pop~~. deposit

if s.top instanceof ACCOUNT

elseif s.top instof STRING

Information Hiding Principle

SHOP $\xrightarrow{\text{cart}}$ CART

Supplier:

HASH_TABLE

LINKED_LIST

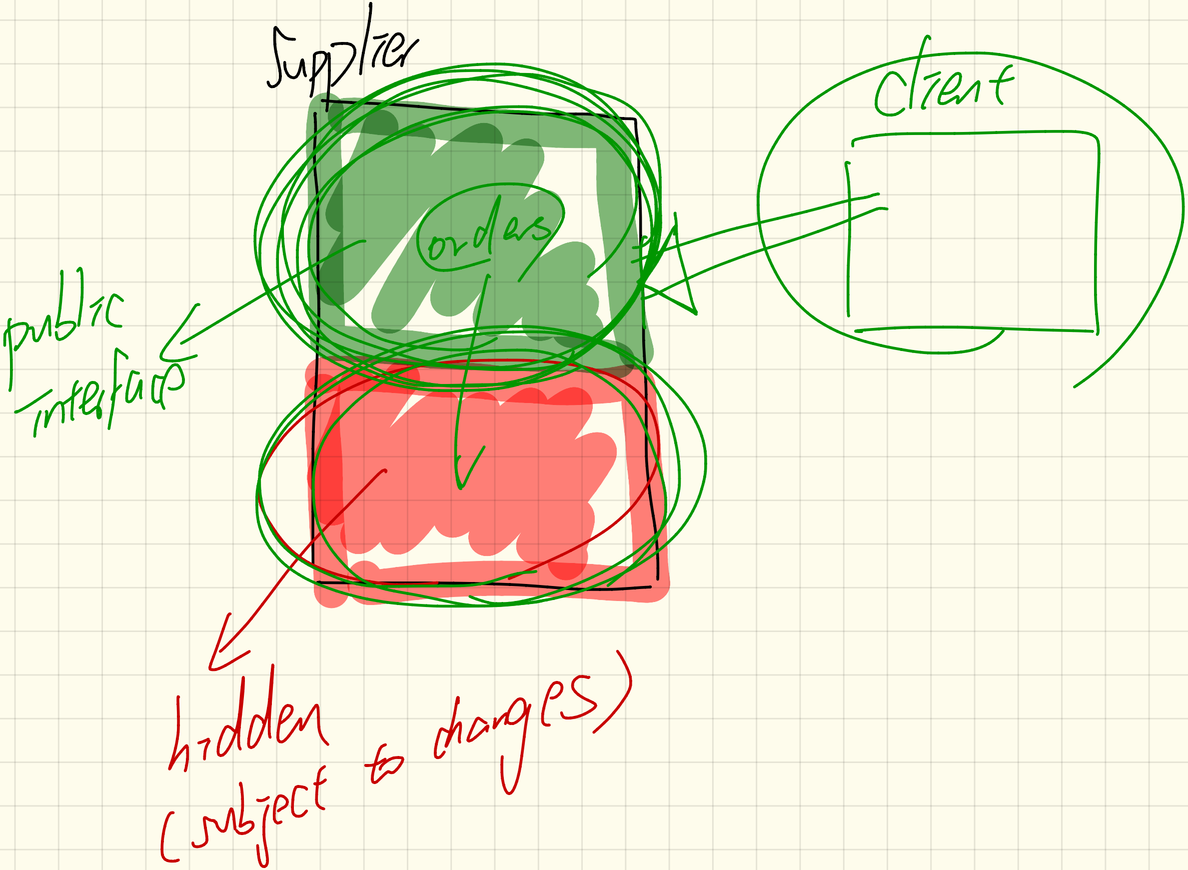
```
class
  CART
feature
  orders: ARRAY[ORDER]
end

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

Client:

cursor

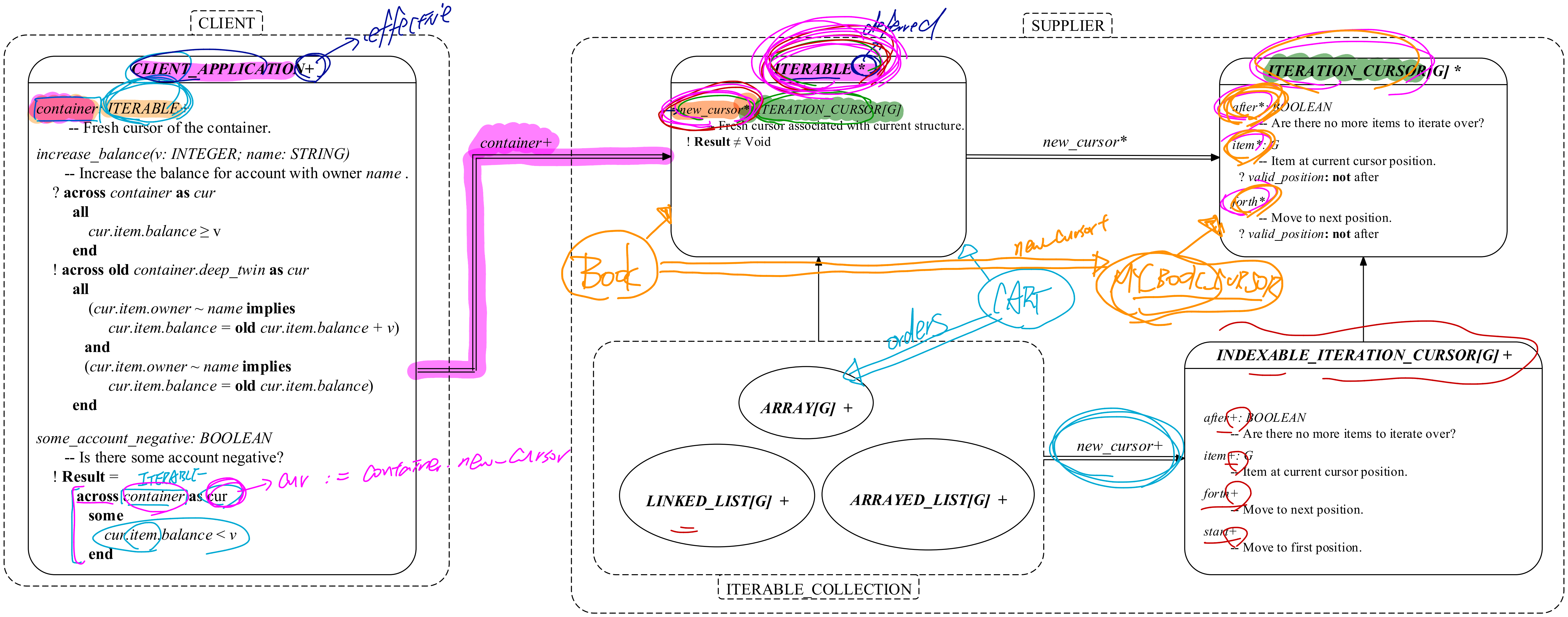
```
class
  SHOP
feature
  cart: CART
  checkout: INTEGER
    do
      from
        i := cart.orders.lower
      until
        i > cart.orders.upper
      do
        Result := Result +
          cart.orders[i].price
          *
          cart.orders[i].quantity
        i := i + 1
      end
    end
end
```

Problems?



Supplier
client
orders
public Interface
hidden (subject to changes)

Iterator Design Pattern



Implementing the ITERATOR Pattern: Easy Case

```
class
    CART
inherit
    ITERABLE[ORDER]          ?
feature {NONE} -- Information Hiding
    orders: ARRAY[ORDER]

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

Implementing the ITERATOR Pattern: Hard Case

```
class
    BOOK[G]                                    TUPLE[STRING, G]
inherit
        ITERABLE[     G     ]
feature {NONE} -- Information Hiding
    names: ARRAY[STRING]
    records: ARRAY[G]

    new_cursor: MY_BOOK_CURSOR[ TUPLE[S, G] ]
        do

        end
end
```

Static vs. Dynamic Types

local

oa : A S.T static

do

create {?} oa.make

dynamic type

A oa = new ?();

```
SORTED_MAP_ADT
deferred class
    SORTED_MAP_ADT [K -> COMPARABLE, V -> ANY]
inherit
    ITERABLE [TUPLE [K, V]]
feature -- model
    model: FUN [K, V]
        deferred
        end
feature {NONE} -- attributes
    instance: like Current
        deferred
        end
feature -- commands
    put (val: V; key: K)
        deferred
        ensure
            inserted: model ~ ((old model.deep_twin) @<+ [key, val])
        end
    sub_map (lower, upper: K): like Current xclusive
        -- may return nothing if no elements between `lower' and `upper'
        require
            lower_less_than_upper: lower < upper
        do
            Result := instance.deep_twin
            across
                Current as cursor
            loop
                if lower <= cursor.item.key and then cursor.item.k
                    Result.extend (cursor.item.key, Current [curso
                end
            end
```

template

Smm.sub_map.

```
SORTED_MODEL_MAP
class
    SORTED_MODEL_MAP [K -> COMPARABLE, V -> ANY]
inherit
    SORTED_MAP_ADT[K,V]
create
    make_empty, make_from_array, make_from_sorted_map
feature -- model
    model: FUN [K, V]
            -- abstraction function
        do
            Result := implementation
        end
feature{NONE} -- attributes
    implementation: FUN[K,V]
            -- inefficient but abstract implementation of sorted map
        attribute
            create Result.make_empty
        end
    instance: like Current
        attribute
            create Result.make_empty
        end
feature -- commands
    put (val: V; key: K) --(key: K; val: V)
            -- puts an element of `key' and `value' into map
            -- behaves like `extend' if `key' does not exist
            -- otherwise behaves like `update'
            -- NOTE: This method follows the convention of `val'/`key'
        do
            implementation.override_by ([key, val])
        end
```

Writing Postcondition: Exercise

```
all_positive_values(a: ARRAY[INTEGER]): ARRAY[INTEGER]
    ensure
        across Result as x
          all
            x.item > 0
          end
```

Wednesday January 30

Lecture 8

```
SORTED_MAP_ADT
deferred class
  SORTED_MAP_ADT [K -> COMPARABLE, V -> ANY]
inherit
    ITERABLE [TUPLE [K, V]]
feature -- model
    model: FUN [K, V]
        deferred
        end
feature {NONE} -- attributes
    instance: like Current
        deferred
        end
feature -- commands
    put (val: V; key: K)
        deferred
        ensure
            inserted: model ~ ((old model.deep_twin) @<+ [key, val])
        end
    sub_map (lower, upper: K): like Current xclusive
        -- may return nothing if no elements between `lower' and `upper'
        require
            lower_less: lower < upper
        do
            Result := instance.deep_twin
            across
                Current as cursor
            loop
                if lower <= cursor.item.key and then cursor.item.k
                    Result.extend (cursor.item.key, Current [curso
                end
            end
```

template

m : SORTED_MAP_ADT

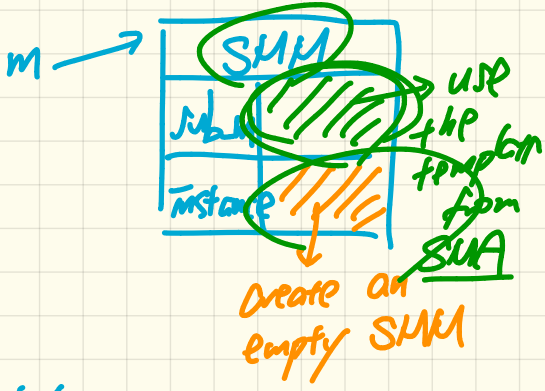
static type

dynamic type

create {SORTED_MODEL_MAP} m.make_empty

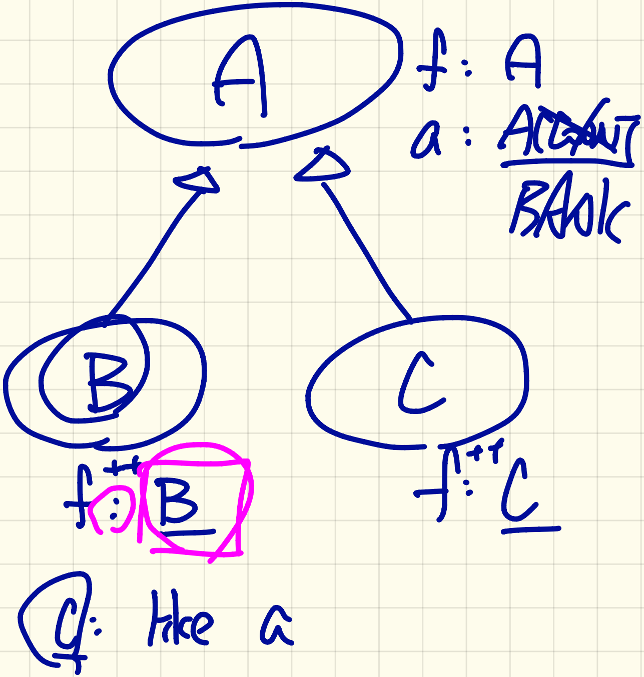
m.sub_map(__ , __)

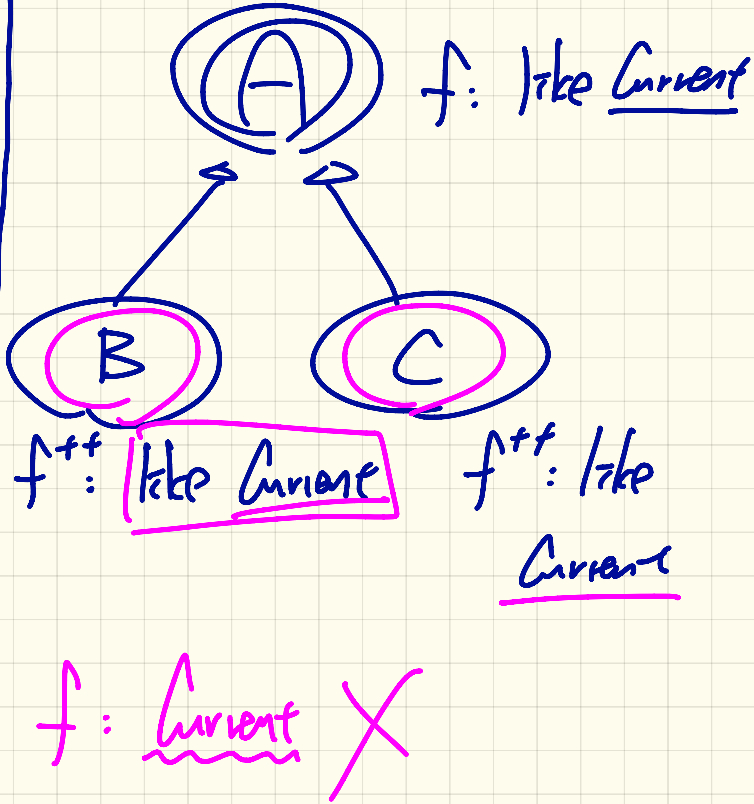
SMM

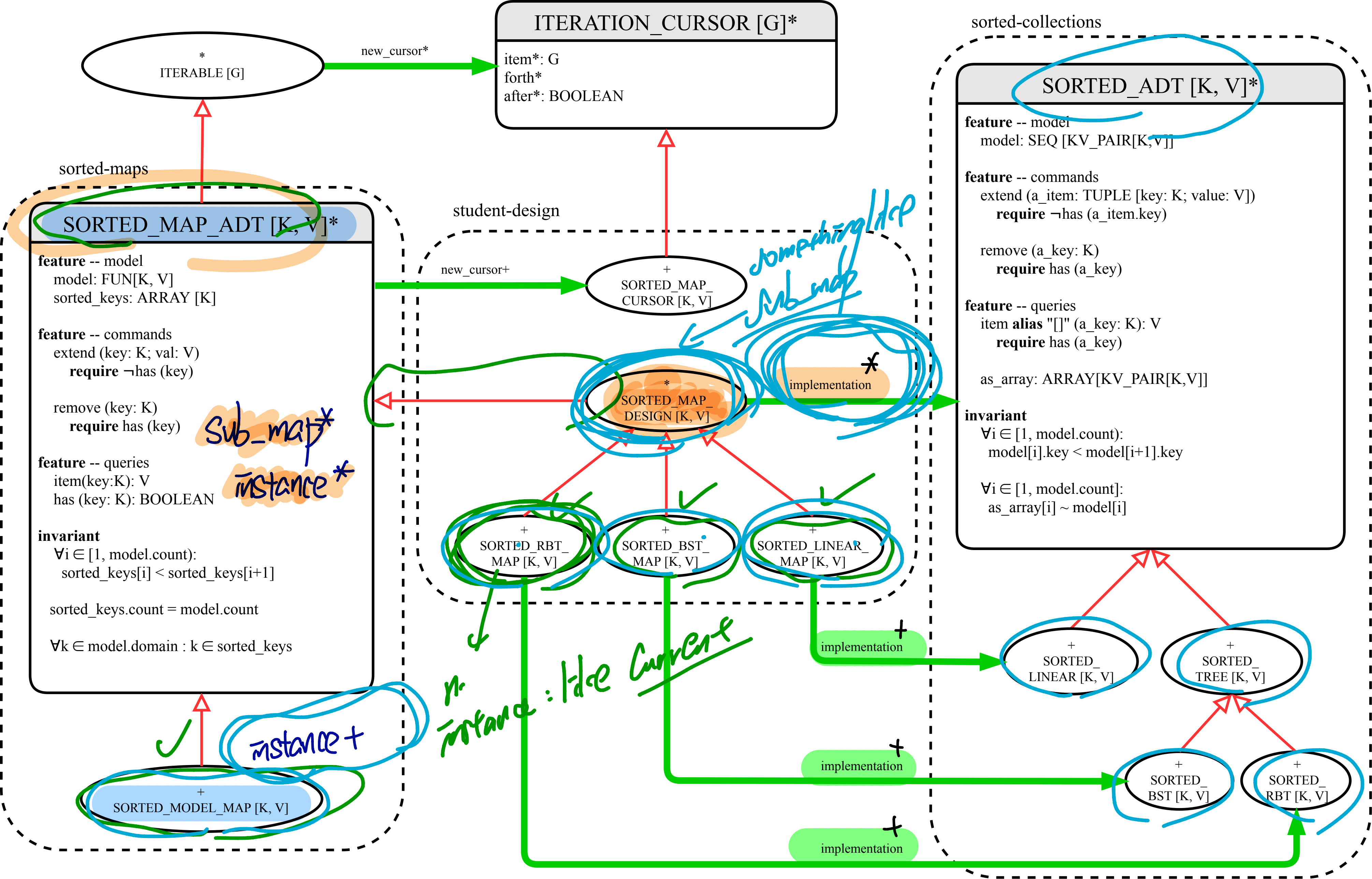



```
SORTED_MODEL_MAP
class
  SORTED_MODEL_MAP [K -> COMPARABLE, V -> ANY]
inherit
  SORTED_MAP_ADT[K,V]
create
  make_empty, make_from_array, make_from_sorted_map
feature -- model
  model: FUN [K, V]
      -- abstraction function
    do
      Result := implementation
    end
feature{NONE} -- attributes
  implementation: FUN[K,V]
      -- inefficient but abstract implementation of sorted map
    attribute
      create Result.make_empty
    end
  instance: like Current
    attribute
      create Result.make_empty
    end
feature -- commands
  put (val: V; key: K) --(key: K; val: V)
      -- puts an element of `key' and `value' into map
      -- behaves like `extend' if `key' does not exist
      -- otherwise behaves like `update'
      -- NOTE: This method follows the convention of `val'/`key'
    do
      implementation.override_by ([key, val])
    end
```

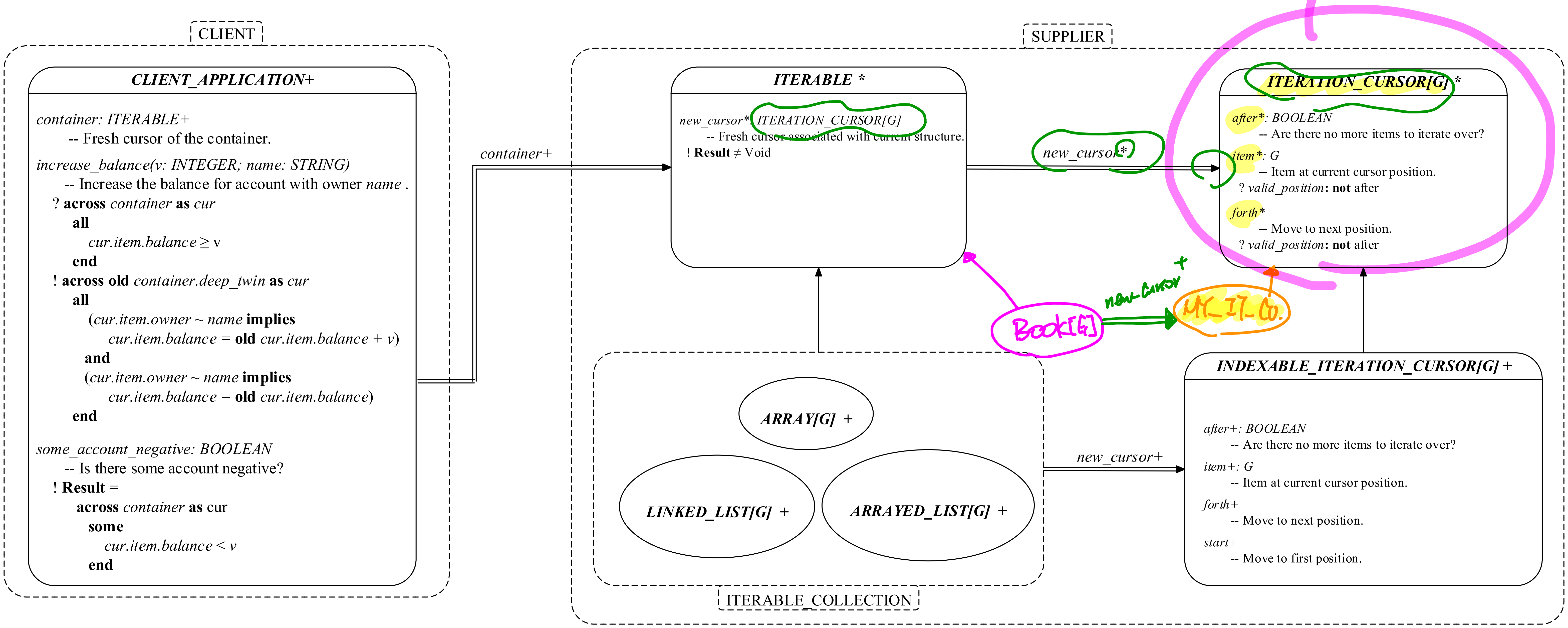
another type


A
B
C
f : A
a : ~~ACCOUNT~~ BANK
f++ : B
f++ : C
a : like a


A
B
C
f : like Current
f++ : like Current
f++ : like Current
f : Current X


ITERATION_CURSOR [G]*
item*: G
forth*
after*: BOOLEAN
*
ITERABLE [G]
new_cursor*
sorted-collections
SORTED_ADT [K, V]*
feature -- model
model: SEQ [KV_PAIR[K, V]]
feature -- commands
extend (a_item: TUPLE [key: K; value: V])
require ¬has (a_item.key)
remove (a_key: K)
require has (a_key)
feature -- queries
item alias "[]" (a_key: K): V
require has (a_key)
as_array: ARRAY[KV_PAIR[K,V]]
invariant
∀i ∈ [1, model.count):
model[i].key < model[i+1].key
∀i ∈ [1, model.count]:
as_array[i] ~ model[i]
sorted-maps
SORTED_MAP_ADT [K, V]*
feature -- model
model: FUN[K, V]
sorted_keys: ARRAY [K]
feature -- commands
extend (key: K; val: V)
require ¬has (key)
remove (key: K)
require has (key)
feature -- queries
item(key:K): V
has (key: K): BOOLEAN
invariant
∀i ∈ [1, model.count):
sorted_keys[i] < sorted_keys[i+1]
sorted_keys.count = model.count
∀k ∈ model.domain : k ∈ sorted_keys
+
SORTED_MODEL_MAP [K, V]
student-design
new_cursor+
+
SORTED_MAP_
CURSOR [K, V]
*
SORTED_MAP_
DESIGN [K, V]
implementation
+
SORTED_RBT_
MAP [K, V]
+
SORTED_BST_
MAP [K, V]
+
SORTED_LINEAR_
MAP [K, V]
implementation
implementation
implementation
+
SORTED_
LINEAR [K, V]
+
SORTED_
TREE [K, V]
+
SORTED_
BST [K, V]
+
SORTED_
RBT [K, V]
Sub_map*
instance*
instance+

Iterator Design Pattern



Implementing the ITERATOR Pattern: Hard Case

```
class
    Book[G]
inherit
    IR[ TUPLE[S, G] ]
feature {NONE} -- Information Hiding
    names : ARRAY[STRING]
    records : ARRAY[G]

new_cursor : MY_IT_CUR[T[S, G]]
    do
    end
end
```

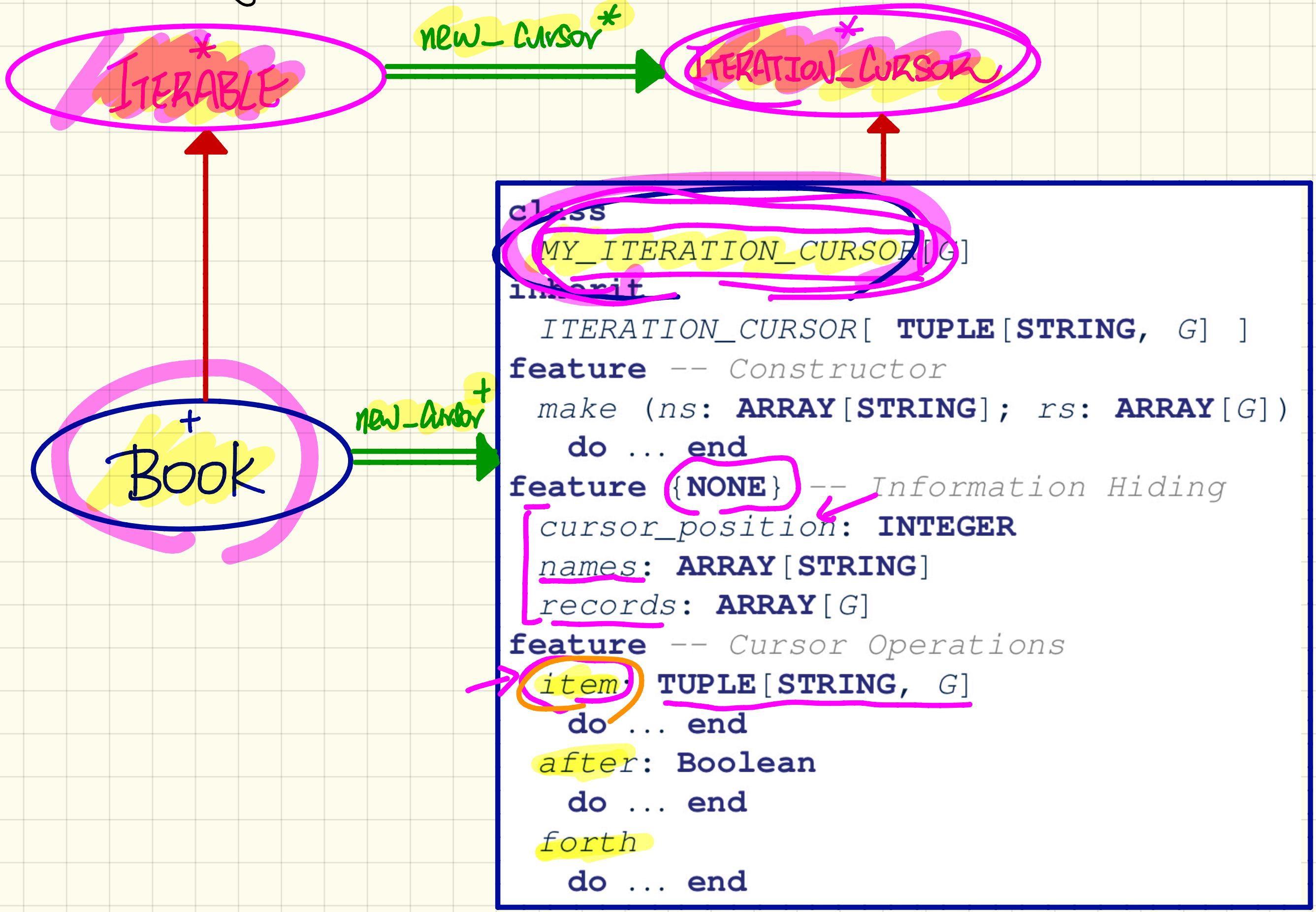
Book_REC[~~STRING~~ G]

G

S

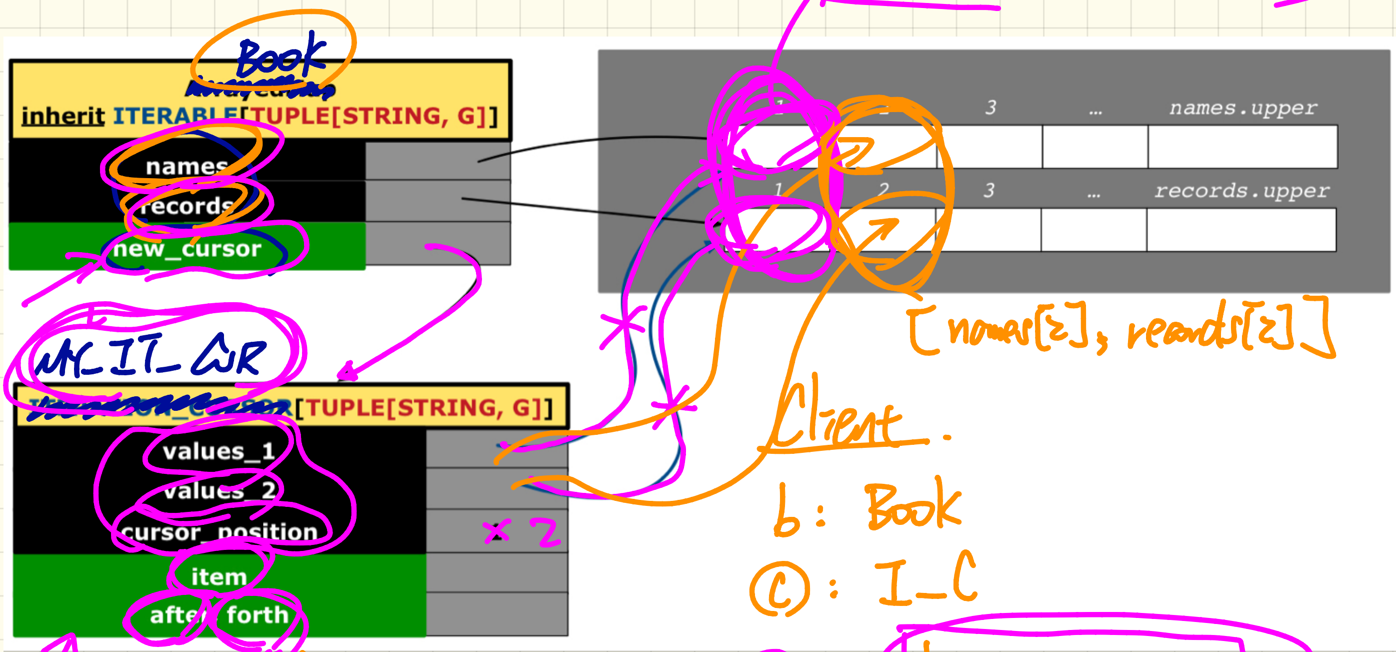
```
class Book_RECORD[~~G~~ S]
    name : STRING
    record : ~~G~~ S
end
```

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



Client.

b : Book

c : I_C

c := b.new_cursor

c

from
  c.start
until
  c.after
do
  c.item
  c.forth
end

Use of Iterable in Contracts

ARRAY, LIN-LIST

?

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

cursor := 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) as cursor
      all
        accounts [cursor.item].id <= accounts [cursor.item + 1].id
      end
    do
      ...
    ensure
      Result.id = acc_id
    end
```

Iterable
Interval

Use of Iterable in Contracts: Exercise

```
class BANK
...
  accounts: LIST [ACCOUNT]
  contains_duplicate: BOOLEAN
      -- Does the account list contain duplicate?
    do
      ...
    ensure
```

$$\forall i, j : INTEGER \mid 1 \leq i \leq accounts.count \wedge 1 \leq j \leq accounts.count \bullet \boxed{accounts[i] \sim accounts[j] \Rightarrow i = j}$$

```
    end
```

Contra-positive

$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$

$i \ /= j \Rightarrow accounts[i] \ /\sim \ accounts[j]$

local

~~i, j: INT~~

~~do~~

~~from~~

~~until~~

~~end loop~~

end

```
across 1 |..| accounts.count as i
    across 1 |..| accounts.count as j
    i.item
    end
end
```

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
```

across accounts
as cursor
loop

max := cursor.item

-- no need ~~to~~
-- say cursor. forth

end

Use of Iterable in Implementation (2)

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

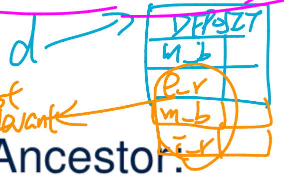
```
class SHOP
  cart: CART
  checkout: INTEGER
      -- Total price calculated based
    require ??
    local
      order: ORDER
    do
      across
        cart as cursor
      loop
        order := cursor. item
        Result := Result + order.price * order.quantity
      end
    ensure ??
  end
```

no cursor.forth

Shared Data via Inheritance

Cohesion

Single Choice Principle



not relevant

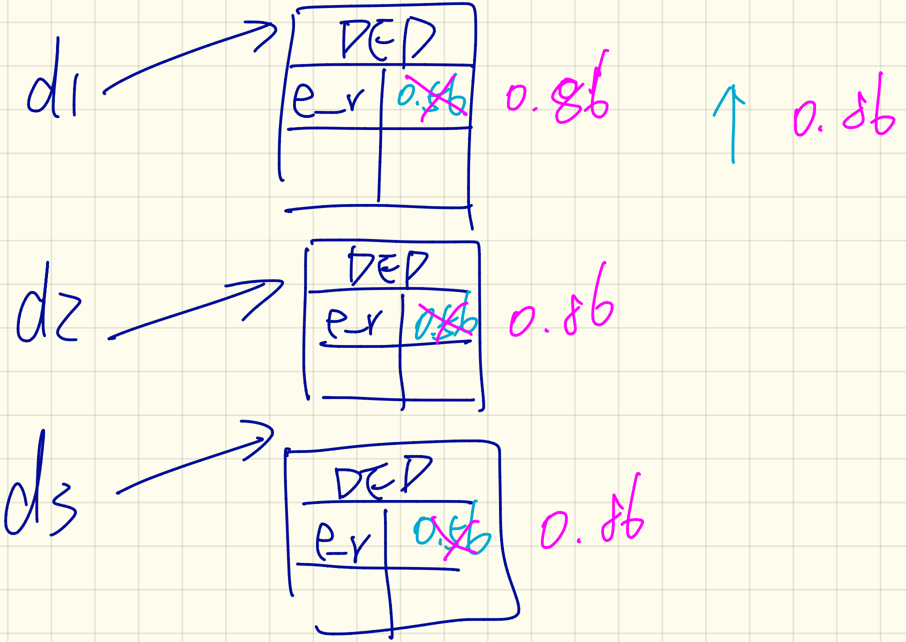
Descendant:

```
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?


d1
d2
d3
DED
e_r
~~0.86~~
0.86
DED
e_r
~~0.86~~
0.86
DED
e_r
~~0.86~~
0.86
↑ 0.86

Once Routine (1)

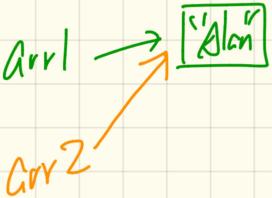
arr1 → "Alan"

arr2 → "Mark"

```
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)




```
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
```

very 1st time

not 1st time

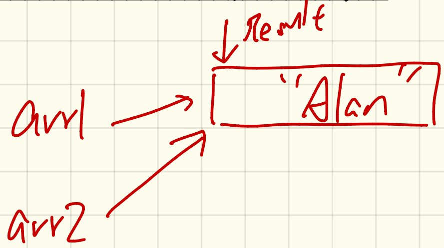
Monday February 4

Lecture 9

- Lab 4 released

  Tutorial Videos on ETF

Once Routine (2)




```
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
```

1st time

```
    arr1 := a.new_once_array ("Alan")
    Result := arr1.count = 1 and arr1[1] ~ "Alan"
    check Result end
```

not 1st time

ignored.

```
    arr2 := a.new_once_array ("Mark")
    Result := arr2.count = 1 and arr2[1] ~ "Alan"
    check Result end

    Result := arr1 = arr2
    check Result end
end
```

Approximating Once Routines in Java (1)

breaking singleton.

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

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

BankData d2 = new BankData();

data == d2

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

false

T F

X

factory method

Problem?

d1

d2

BD

1.23

data

BankDataAccess bda = new ______();

BankData d1 = bda.getData();

BankData d2 = bda.getData();

d1.setIR(1.23);

Approximating Once Routines in Java (2)

Separation of Concerns: Data / Data Access

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;
  }
}
```

static

data

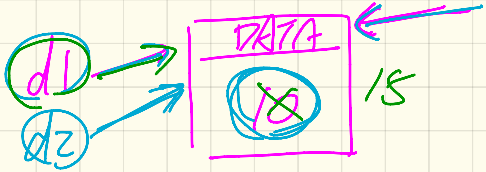
data access

Problem?

BankData d1 = new BankData();

X ∵ it's private.

Singleton 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
invariant data = data
```

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
```

create access.make ✗ ∵ it's expanded

→ 1st call

→ not 1st call

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

Singleton Pattern: Code (2.1)

class BANK_DATA_A_2
data : BD
once create Result.make
end
end

Supplier:

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

```
expanded class
  BANK_DATA_ACCESS
feature
  data: BANK_DATA
    -- The one and only access
  once create Result.make end
invariant data = data
```

Client:

```
class
ACCOUNT
feature
  data: BANK_DATA
  make (...)
      -- Init. access to bank data.
    local
      data_access: BANK_DATA_ACCESS
    do
      data := data_access.data
      ... create data.make
    end
end
```

Writing **create** *data.make* in client's `make` feature does not compile. Why?

```
class BANK_DATA
  create {DATA_ACCESS} make
  ...
end
```

only this client may create an instance of BANK_DATA using this constructor

```
class ACCOUNT
  f
    local
      bd: BANK_DATA
    do
      create bd.make      ✗
    end
end
```

da: DATA_ACCESS

bd := da.data ✓

client

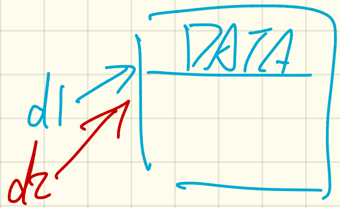
BAN_DATA

create {B_D_A} make

BANK_DATA_ACCESS

BANK_D_A_2 → create data. make ? X

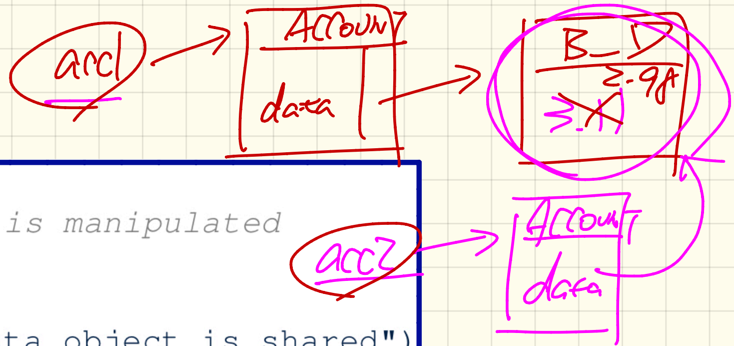
```
class ACCOUNT

  make
    local
      da1, da2 : B_D_A
      data1, data2 : B_D
    do
      data1 := da2.data
      data2 := da1.data
    end
end
```



data1 = data2.

2nd time
that {B_D_A}.data
once routine is called

Singleton Pattern: Code (2.2)



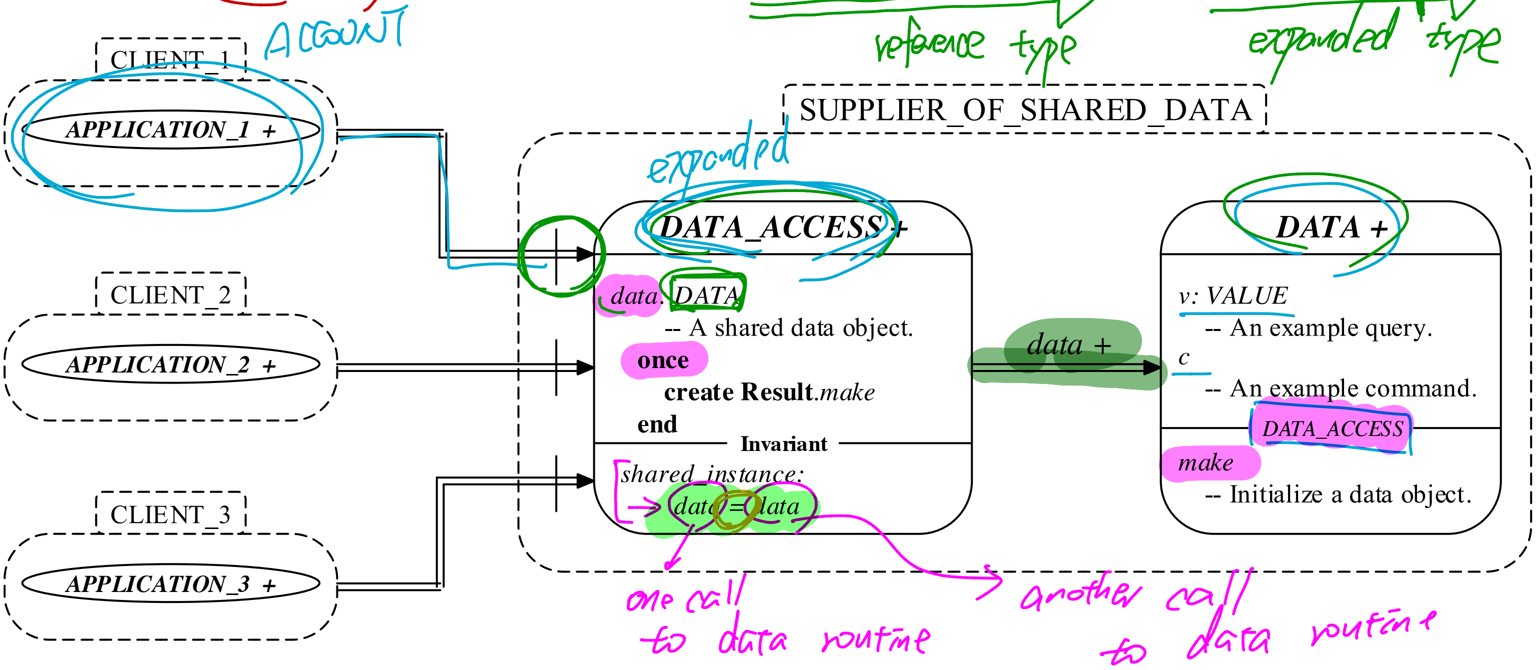

```
test_bank_shared_data: BOOLEAN
    -- Test that a single data object is manipulated
  local acc1, acc2: ACCOUNT
  do
    comment("t1: test that a single data object is shared")
    create acc1.make ("Bill")
    create acc2.make ("Steve")
    Result := acc1.data = acc2.data
    check Result end
    Result := acc1.data ~ acc2.data
    check Result end
    acc1.data.set_interest_rate (3.11)
    Result :=
          acc1.data.interest_rate = acc2.data.interest_rate
      and acc1.data.interest_rate = 3.11
    check Result end
    acc2.data.set_interest_rate (2.98)
    Result :=
          acc1.data.interest_rate = acc2.data.interest_rate
      and acc1.data.interest_rate = 2.98
  end
```

1st time calling {B_D_A}.data

T

Singleton Design Pattern

client supplier



$$= \quad o1 := o2$$

$$= \quad o.f(o2)$$

$$= \quad o1 := f(o1)$$

8 kinds of students.

STUDENT

is_resident: BOOL

set_p_r

set_d_r

not coherent.

$\underline{b1}, \underline{b2}, \underline{b3}$: BOOL

$b1 \land \neg b2 \land b3 \longrightarrow$ 1 kind

$\neg b1 \land b2 \land \neg b3 \longrightarrow$ another kind.

Violation of 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   * inf_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   * inf_rate
    end
end
```

Without Inheritance: Collection of Students

too tolerant

→ students : LL[ANY] ×

```
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
```

polymorphic collection

Inheritance:

Code Reuse

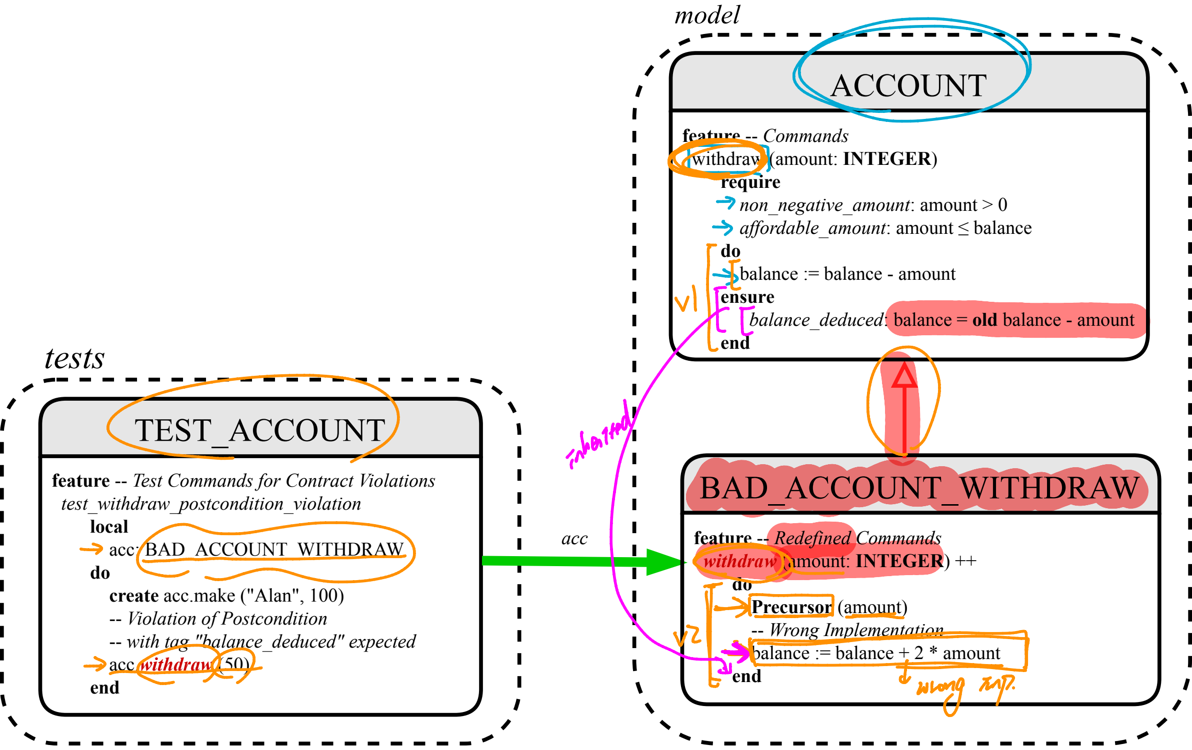
```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: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

refers to the version defined in super class

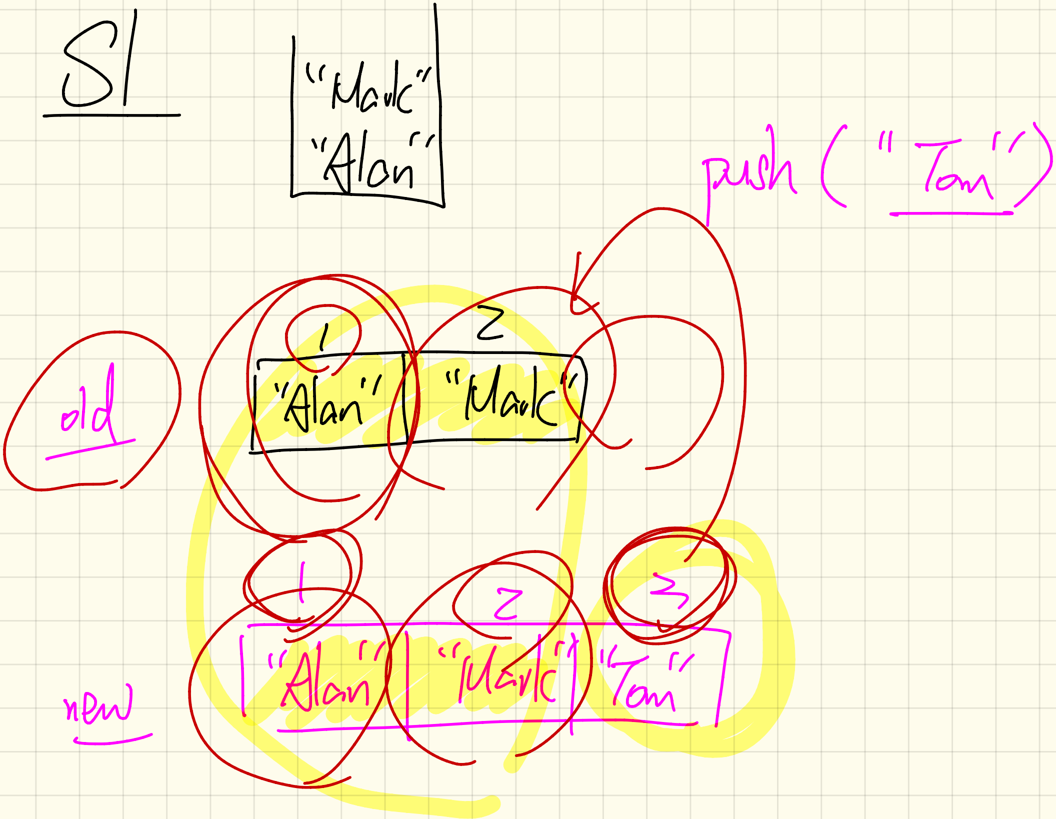
```
class
  NON_RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  discount_rate: REAL
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ; Result := base * discount_rate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

Precursor(— , —)


model
ACCOUNT
feature -- Commands
withdraw (amount: INTEGER)
require
non_negative_amount: amount > 0
affordable_amount: amount ≤ balance
do
balance := balance - amount
ensure
balance_deduced: balance = old balance - amount
end
v1
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_deduced" expected
acc.withdraw(50)
end
acc
inherited
BAD_ACCOUNT_WITHDRAW
feature -- Redefined Commands
withdraw (amount: INTEGER) ++
do
Precursor (amount)
-- Wrong Implementation
balance := balance + 2 * amount
end
v2
wrong impl.

Monday February 11

Lecture 10


S1
"Mark"
"Alan"
push ("Tom")
old
1
2
"Alan"
"Mark"
new
1
2
3
"Alan"
"Mark"
"Tom"

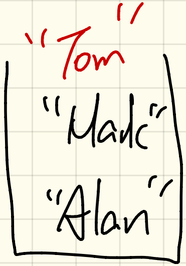
S2

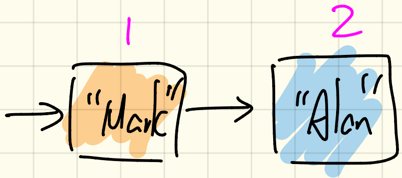
list.first

"="

list[1]

old





Q: empty collection?

push("Tom")

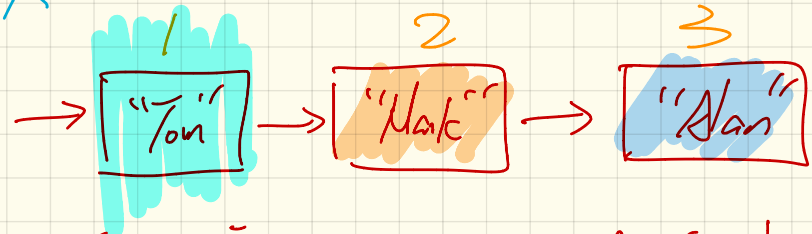
across 2 |..| count as i

all

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

end

list.lower upper new X



across 1 |..| count as i

all

[count - i.item.

end

across count |..| 1 as i X

Developing a LIFO Stack

```
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
```

not only imp. but also contracts we must modify accord. ⇒ violates SCP.

```
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
```

class C

imp : $\overset{\{NONE\}}{?}$?

f1

ensure

imp → imp

f2

ensure

imp → imp

end

class C

{NONE}

imp : ? ?

model : FUN

do [[hash table]] [[AVL tree]]

end

f1

ensure

imp model

f2

ensure

imp model

end

Using MATHMODELS Library

SEQ

abstraction

ARRAY

LL (front)

LL (end)

Implementing 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
```

CS3

Seq Model

end of seq is the top

Writing Contracts using 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
```

call to query

a later call to same query.

dt of the return value

```
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
```

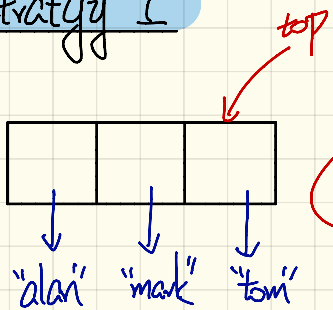
```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 2 (first as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.prepend(cursor.item) end
    ensure
      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
```

Implementing a LIFO Stack

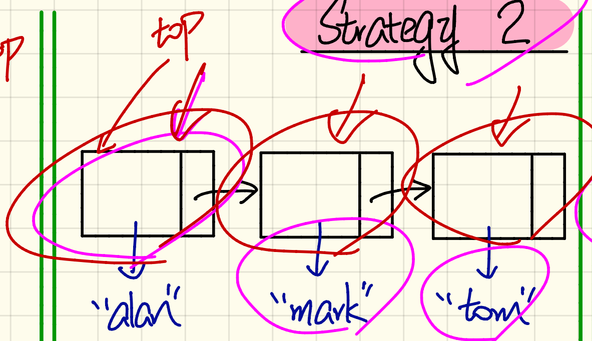

```
class LIFO_STACK[...]
  imp: LL[G] -- s2
  model: SEQ[G]
    do
      create Result.make_empty
      across imp as cursor
        Result.append prepend (imp.item) end
    end
```

Strategy 1

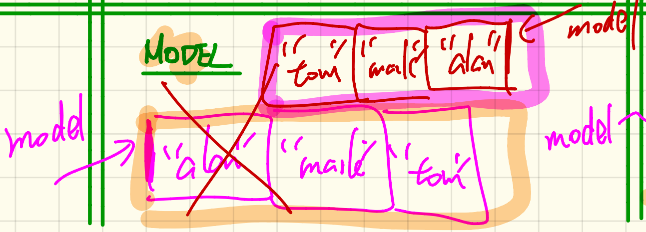


MODEL

Strategy 2

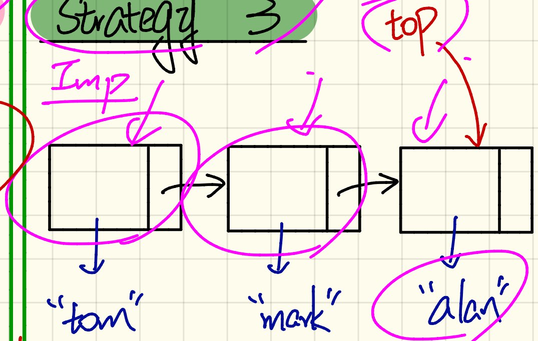


MODEL

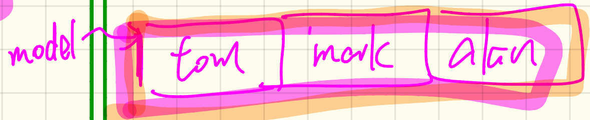


Strategy 3

Imp

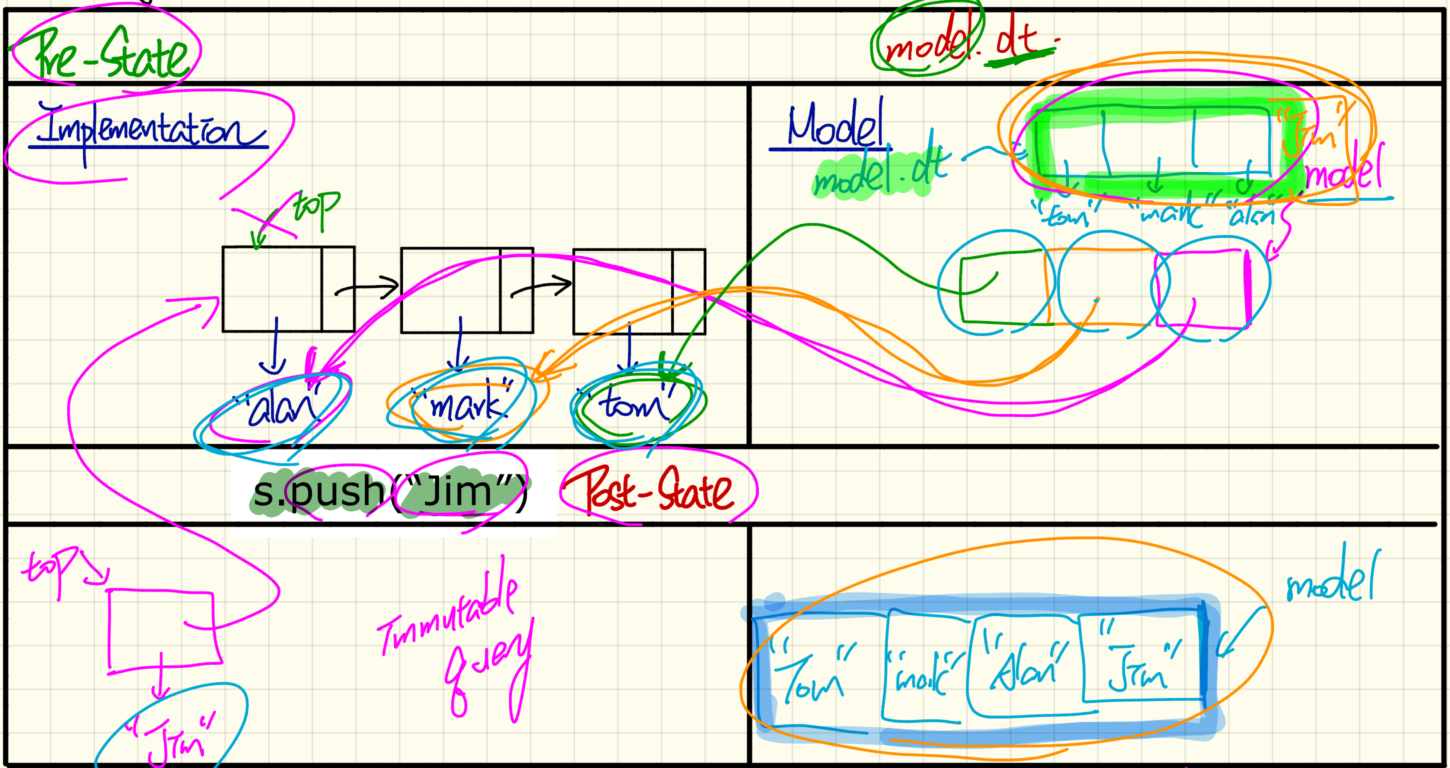


MODEL



Checking MATHMODELS Contracts at Runtime — Strategy 2

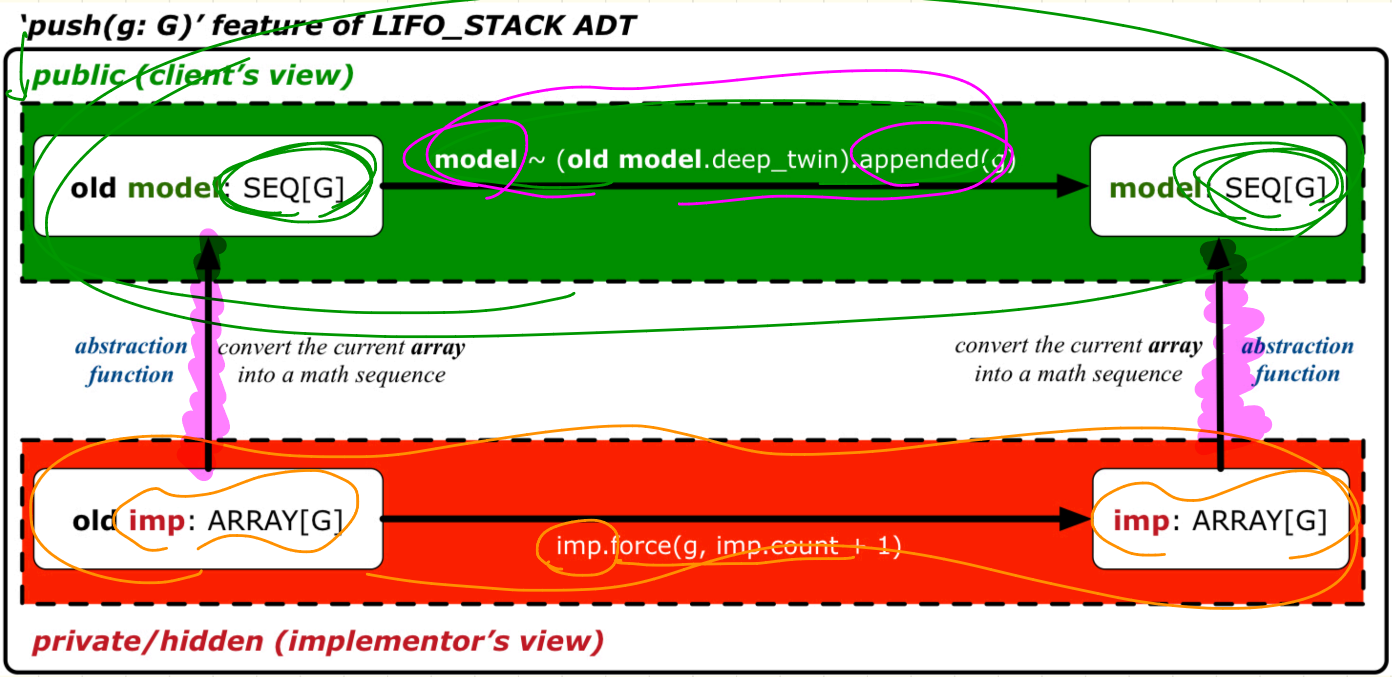
"alan"
"mark"
"tom"




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

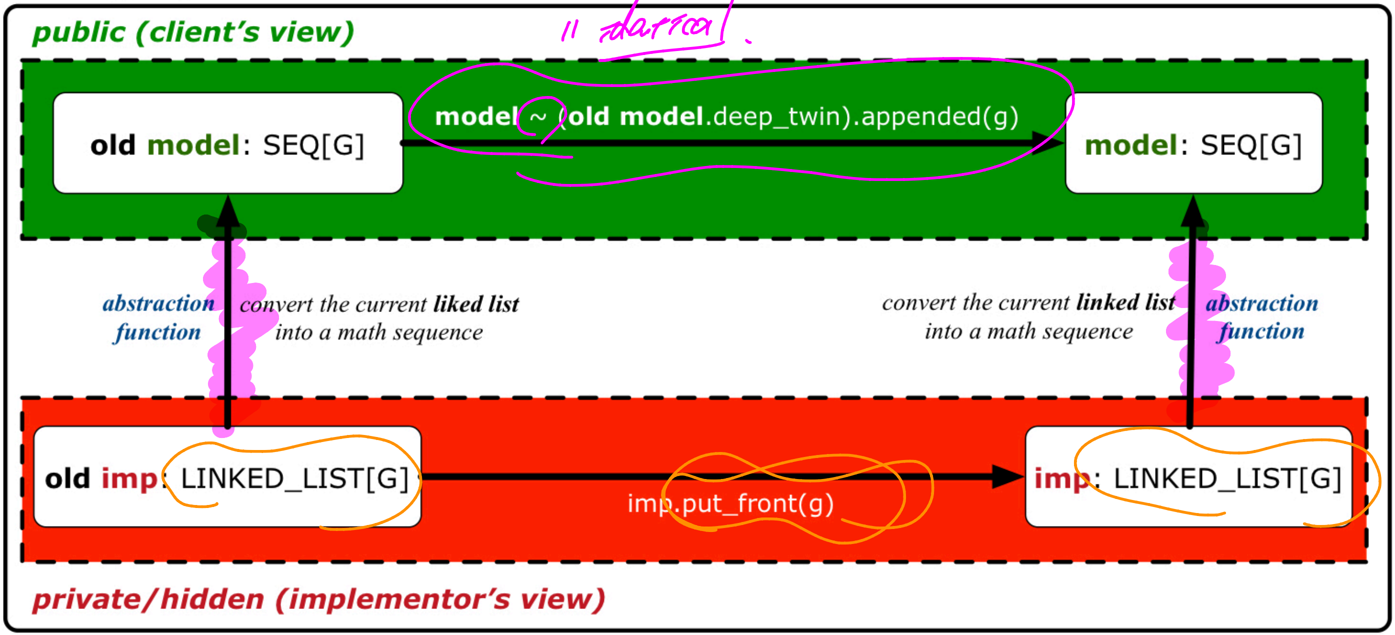
query

Strategy 1: Mathematical Abstraction



Strategy 2: Mathematical Abstraction

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



Wednesday February 13

Lecture 11

Testing REL in MATHMODELS

overridden
overridden_by ([a,3])

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

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

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

(a ~~1~~ 3) (c 3) (b 5) (d 1) (f 3)
(b 2) (a ~~4~~ 3) (c ~~6~~ 4) (e 2)

```
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
```

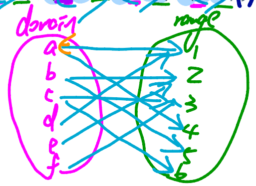
r is not mutated

query: use it in contracts

command: use it in abs. fun. Imp.

r is mutated

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

domain_restricted_by (...)

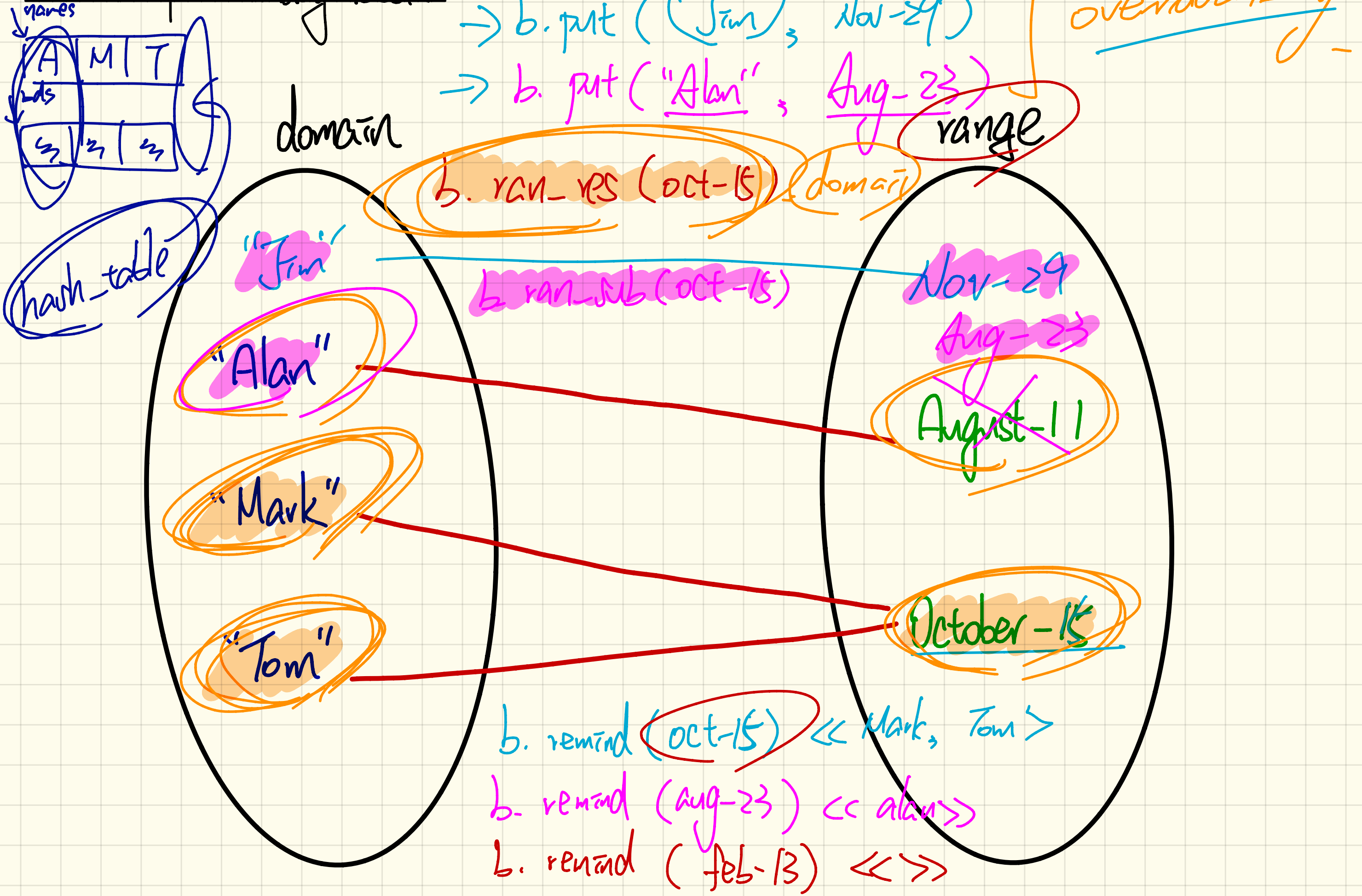
- r.*domain_restricted*(ds): sub-relation of r with domain ds.
  - r.**domain_restricted**(ds) = $\{ (d,r) \mid (d,r) \in r \wedge d \in ds \}$
  - e.g., r.**domain_restricted**($\{a,b\}$) = $\{(\mathbf{a},1),(\mathbf{b},2),(\mathbf{a},4),(\mathbf{b},5)\}$
- r.*domain_subtracted*(ds): sub-relation of r with domain not ds.
  - r.**domain_subtracted**(ds) = $\{ (d,r) \mid (d,r) \in r \wedge d \notin ds \}$
  - 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) = $\{ (d,r) \mid (d,r) \in r \wedge r \in rs \}$
  - 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) = $\{ (d,r) \mid (d,r) \in r \wedge r \notin rs \}$
  - e.g., r.**range_subtracted**($\{1,2\}$) = $\{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

r relation

ds set of pairs

$$\left(\begin{array}{l} r.\text{domain\_restricted}(ds) \\ |\vee| \\ r.\text{domain\_subtracted}(ds) \end{array} \right) \sim r$$

An Example Birthday Book



Birthday Book: Design

m: 2
d: 30

BIRTHDAY_BOOK

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

count: INTEGER → size of book
-- 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: @<+

model ~ ~~old model~~ model.ove-by ([n,d])
(old model.dt)

@<+

old model.dt
not necessary

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
year: INTEGER

invariant
1 ≤ month ≤ 12
1 ≤ day ≤ 31

(m = 1 ∨ m = 3 ...)
⇒ d = 31
(m = 4 ∨ m = 6 ∨ ...)
⇒ d = 30

is_leap_year ∧ m = 2
⇒ d = 29

NAME

item: STRING

invariant
item[1] ∈ A..Z

across Result as n
all
model[n.item] ~ d
end

name: STRING   name → "@#()"

name: NAME   name → NAME / item → ~~"@#()"~~
add(s: STRING; ..)
require

inv. violation

v2 ✓

```
across model.ran_res_by(d).domain as n
    Result.has(n.item)
end
```

Birthday Book: Implementation

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

has

FUN

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

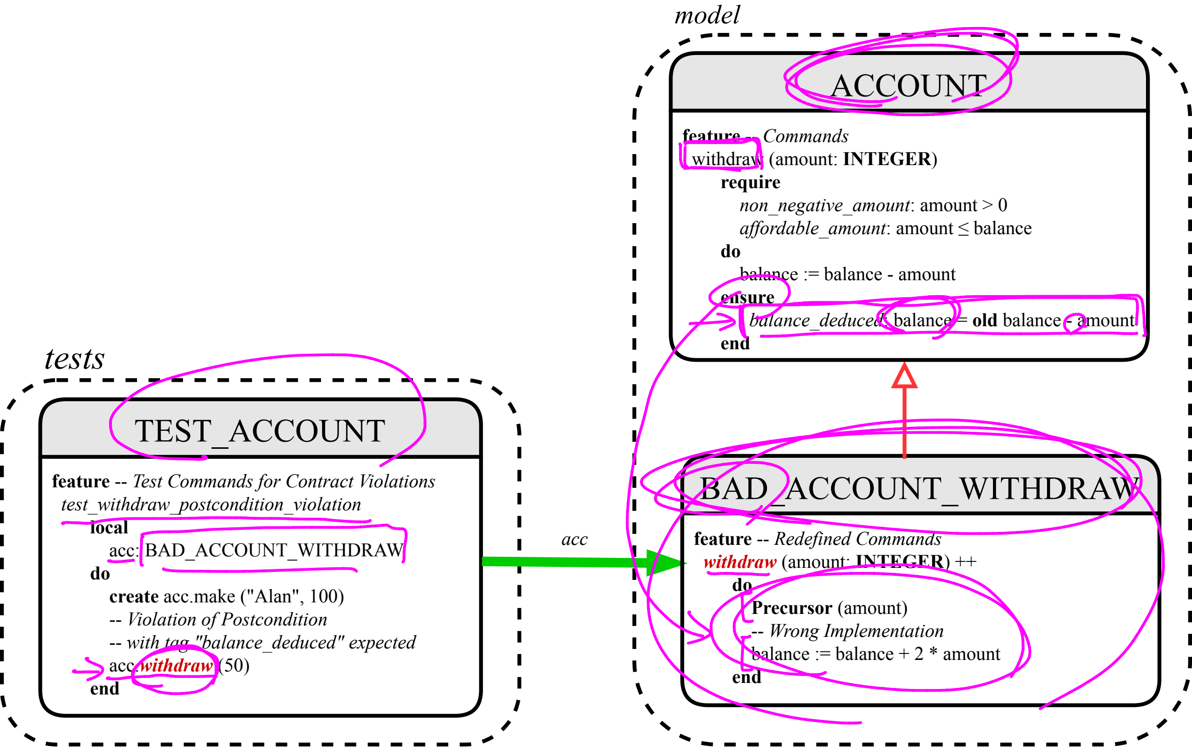
*
HASHABLE

remind: ARRAY[NAME]

NAME

item: STRING

invariant
item[1] ∈ A..Z


model
ACCOUNT
feature -- Commands
withdraw (amount: INTEGER)
require
non_negative_amount: amount > 0
affordable_amount: amount ≤ balance
do
balance := balance - amount
ensure
balance_deduced: 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_deduced" 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

```
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
```

Monday February 25

Lecture 12

Static Type vs. Dynamic Type

static type

Dynamic

- In Java:

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

ST

- In Eiffel:

```
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*:

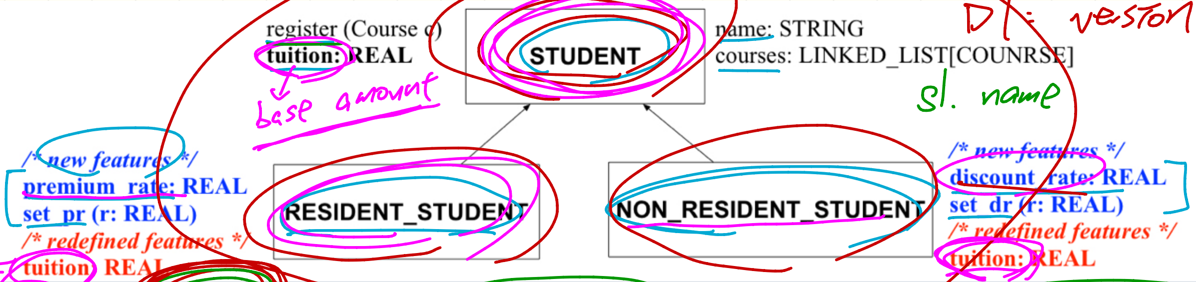
```
local s: STUDENT
do create s.make ("Alan")
```

DT of s is the same as ST of s.

Student classes (with inheritance): Expectations

ST: expectation

DT: version




```
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")
```

STUDENT ST.

s1. RY

STUD. S2

RS

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

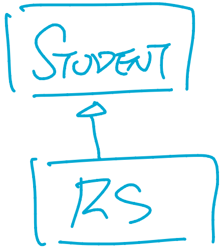
```
if ( o instanceof C ) {
    ______
}
```

```
if attached {C} o then
    ______
end
```

Polymorphism: Intuition

s := rs compiles.
substitute rs for s.

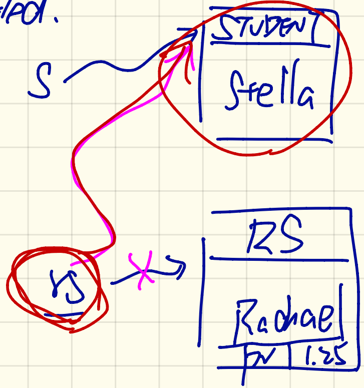
```
local
  s: STUDENT
  rs: RESIDENT_STUDENT
do
  create s.make ("Stella")
  create rs.make ("Rachael")
  rs.set_pr (1.25)
  s := rs /* Is this valid? */
  rs := s /* Is this valid? */
```



4. rs := s should not compile.

1. Assume rs := s compiled.

2. Expectations on rs? S.T.
name, courses, reg, tui.
pr, set_pr



3. ST: RS
rs.pr
Compiles ∵ ST of rs declares pr

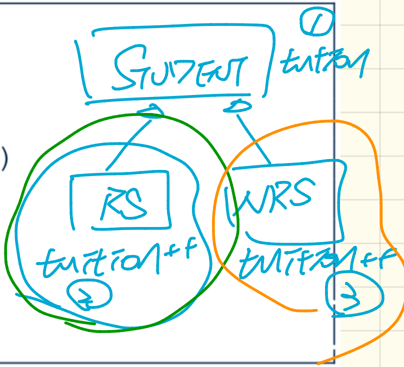
Runtime?
Crash ∵ STUDENT object does not have pr.

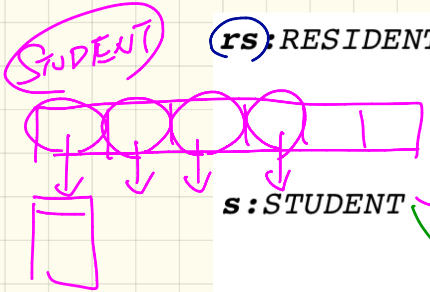
Dynamic Binding : Intuition

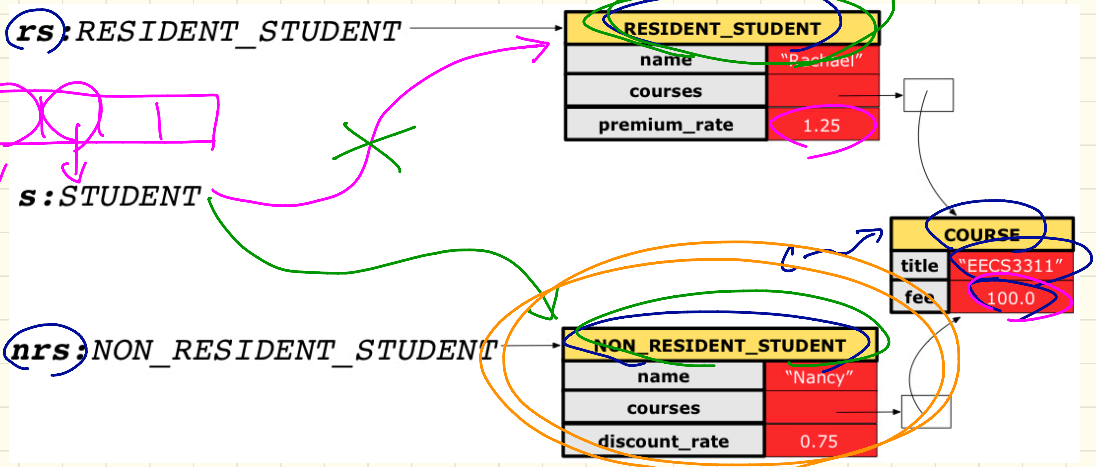
S := ??

ST: STUDENT

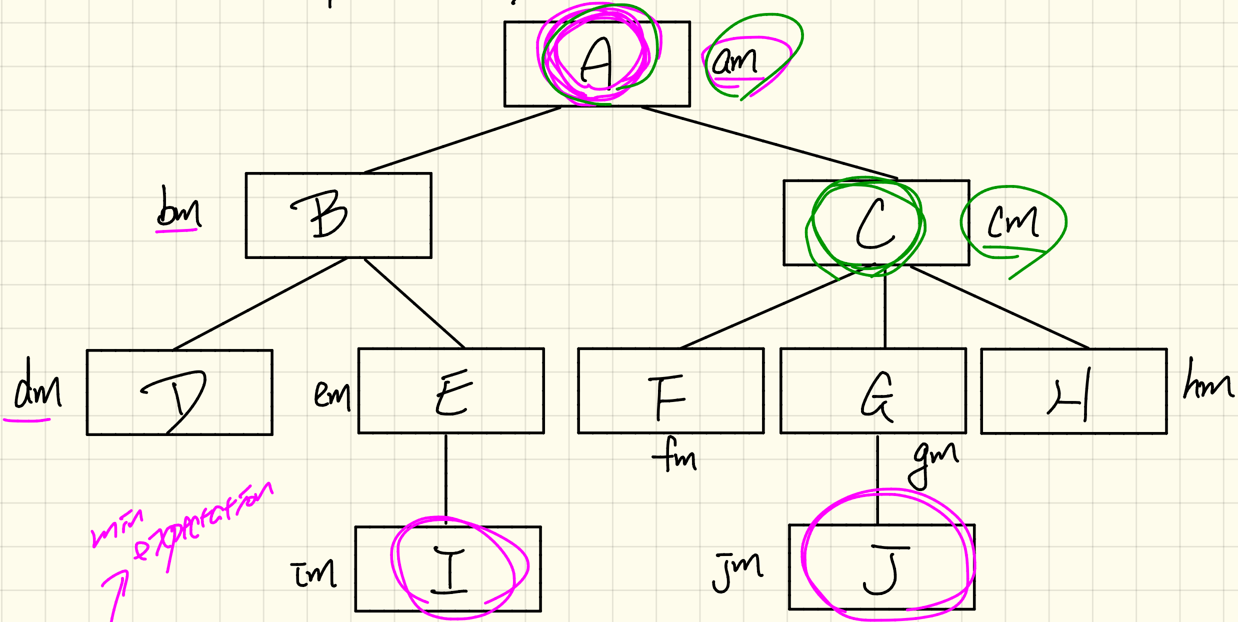
```
local c : COURSE ; s : STUDENT
do crate c.make ("EECS3311", 100.0)
  create {RESIDENT_STUDENT} rs.make("Rachael")
  create {NON_RESIDENT_STUDENT} nrs.make("Nancy")
  rs.set_pr(1.25); rs.register(c)
  nrs.set_dr(0.75); nrs.register(c)
  s := rs;  ; check s.tuition = 125.0 end
  s := nrs;  ; check s.tuition = [redacted] end
```







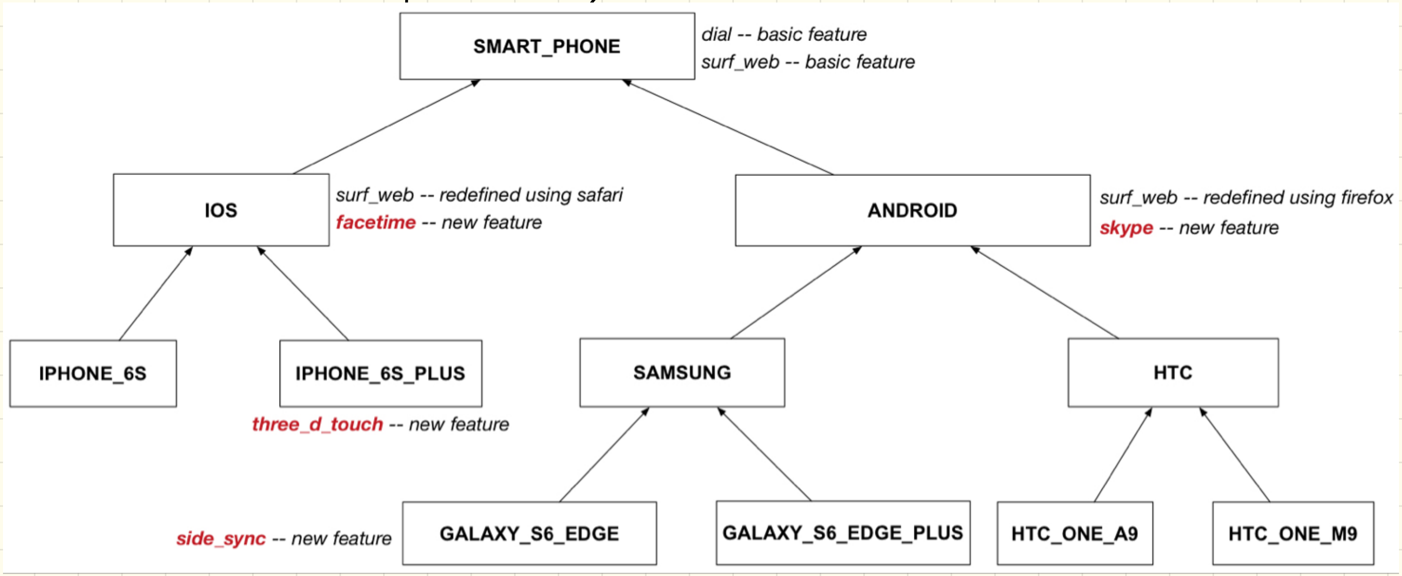
Inheritance Forms a Type Hierarchy (1)



with expectation

	ancestors	expectations	descendants
A	A	am	all classes.
C	A, C	am, cm	C, F, G, H, J
G			

Inheritance Forms a Type Hierarchy (2)



	ancestors	expectations	descendants
SMART_PHONE			
ANDROID			
GS6EP			

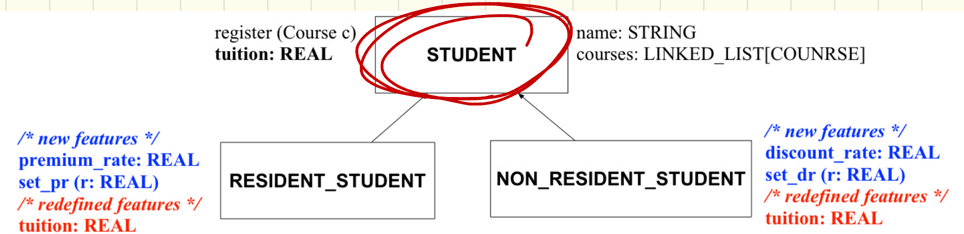
ST: A   ST: D

o1 := o2

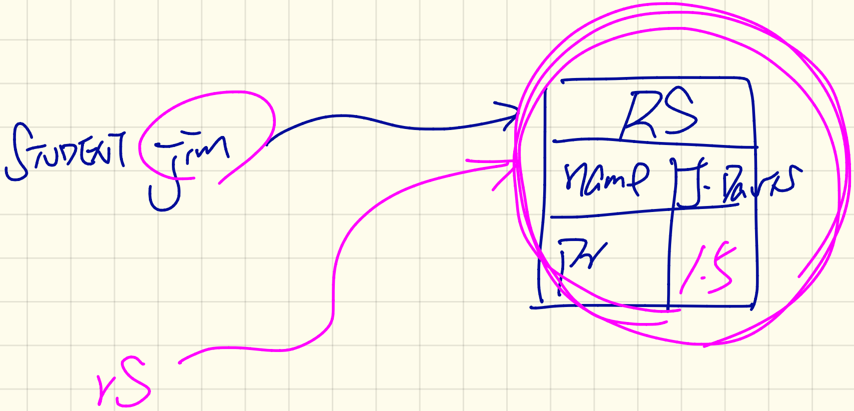
Compile?

ST of o2 is a descendant of ST of o1.

Type Cast : Motivation




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



undecidable

C.e

x

Your Program

last DT of x in class C.

NRS

x: STUDENT

create {RS}.x.make

create {NRS} x.make

from

until

false

loop . . .

end

create {NRS} x.make

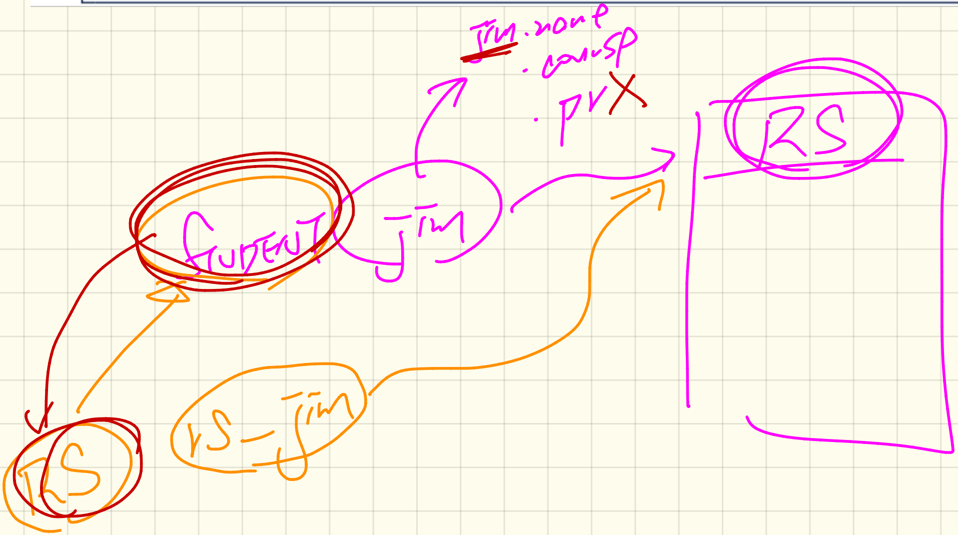
while(true){

}

Type Cast: Syntax

jim instanceof RS

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



cast

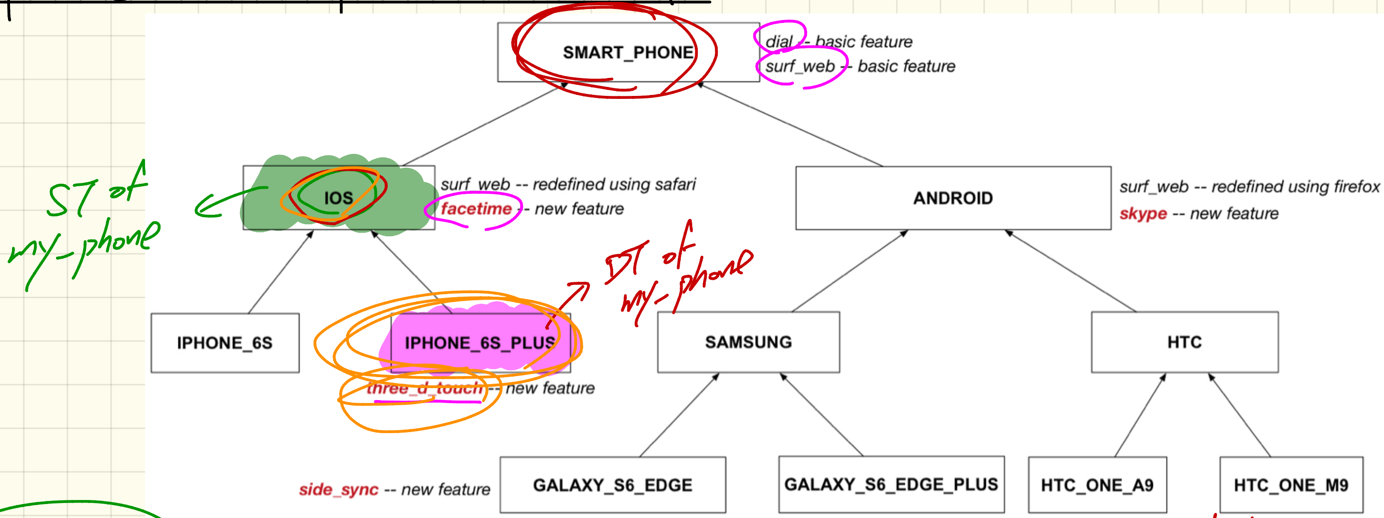
↳ upward cast

↳ restricting less expectations

↳ downward cast

↳ allowing more expectations

Compilable Cast: Upward or Downward




```
my_phone: IOS
create {IPHONE_6S_PLUS} my_phone.make
-- can only call features defined in IOS on myPhone
-- dial, surf_web, facetime ✓   three_d_touch, skype ×
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ✓   facetime, three_d_touch, skype ×
end
check attached {IPHONE_6S_PLUS} my_phone as ip6s_plus then
  -- can now call features defined in IPHONE_6S_PLUS on ip6s_plus
  -- dial, surf_web, facetime, three_d_touch ✓   skype ×
end
```

upward

my_phone . d s_w facetime

sp. dial s_w

ip6s_plus . three_d_touch

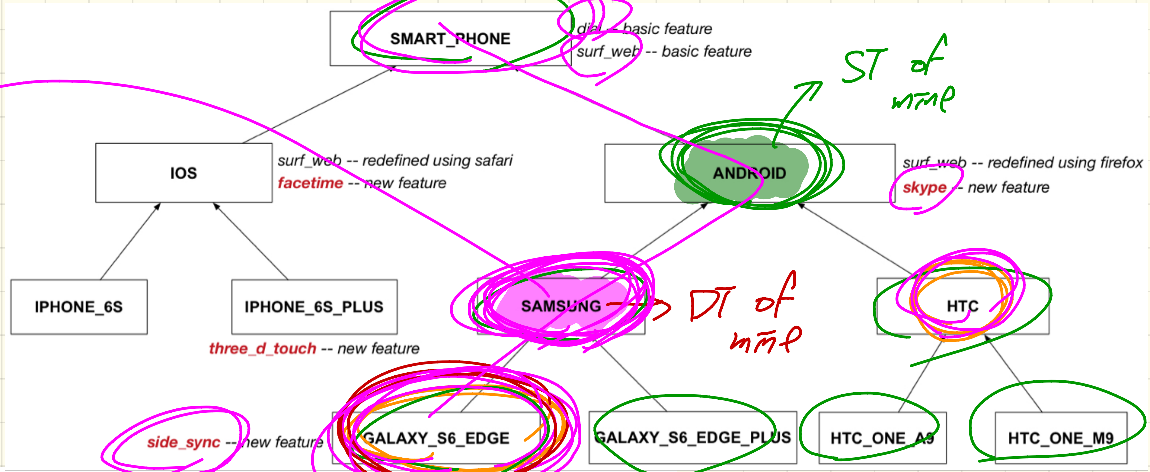
IOS my_phone

SP sp

I_6S_P ip6s_plus

I_P6S_P

Compilable Cast May Fail at Runtime



runtime ✓ error

cast if the type is not an ancestor of the DT.

```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {SAMSUNG} mine.make
    -- ST of mine is ANDROID; DT of mine is SAMSUNG
   ✓check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is SAMSUNG
   ✓check attached {SAMSUNG} mine as samsung then ... end
    -- ST of samsung is SAMSNG; DT of samsung is SAMSUNG
   ✓check attached {HTC} mine as htc then ... end
    -- Compiles ∵ HTC is descendant of mine's ST (ANDROID)
    -- Assertion violation
    -- ∵ HTC is not ancestor of mine's DT (SAMSUNG)
   ✓check attached {GALAXY_S6_EDGE} mine as galaxy then ... end
    -- Compiles ∵ GALAXY_S6_EDGE is descendant of mine's ST (ANDROI
    -- Assertion violation
    -- ∵ GALAXY_S6_EDGE is not ancestor of mine's DT (SAMSUNG)
end
```

False

Assume the cast was ok.

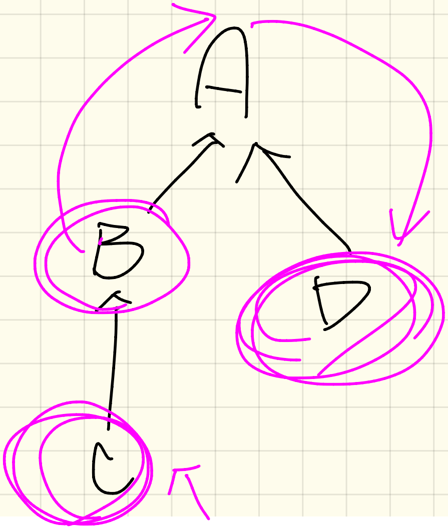
cast error

GALAXY galaxy → SAMSUNG

galaxy.side_sync

✓ compile

× runtime



F

```
local b: B ; d: D
do
  create {C} b.make
  check attached {D} b as temp then d := temp end
end
```

Compile ?

Wednesday February 27

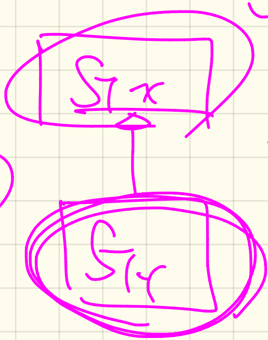
Lecture 13

Type Checking Rules (1)

$\boxed{x := y}$

ST_x ST_y

ST_y is a descendant of ST_x



s1 : STUDENT

s2 : RS

s3 : NRS

s1 := s2

s1 := s3

s2 := s1

s3 := s1

s2 := s3

s3 := s

not declared!

Type Checking Rules (2)

```
check attached {C} y then
    ...
end
```

C is either ancestor of ST of y (upward)

descendant of ST of y (downward)

```
class SMS
    get_s(i: INTEGER): STUDENT
        do
            ...
        end
end
```

sms : SMS

s1 : STUDENT

s2 : RS

s3 : NRS

```
check attached {RS} s1 then ... end
check attached {STUDENT} s2 then ... end
check attached {SMS} s1 then ... end
check attached {RS} s3 then ... end
check attached {RS} sms.get(1) then ... end
```

ST

Type Checking Rules (3)

```
check attached {C} y as temp then
    x := temp
end
```

C

an. de. of Sty

ST temp = C

ST x

C

```
class SMS
    get_s(i: INTEGER): STUDENT
      do
        ...
      end
end
```

sms : SMS

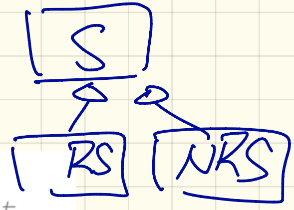
s1 : STUDENT

s2 : RS

s3 : NRS

```
check attached {RS} sms.get_s(1) as temp then
    s1 := temp
    s3 := temp
end
```

Feature Call Arguments : Client



ST[ss[0]] ST_s ST[ss[0]] ST_{rs}

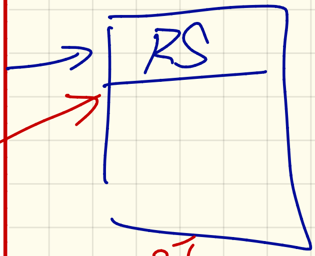
```
class STUDENT_MANAGEMENT_SYSTEM {
  ss : ARRAY[STUDENT] -- ss[i] has static type Student
  add_s (s: STUDENT) do ss[0] := s end
  add_rs (rs: RESIDENT_STUDENT) do ss[0] := rs end
  add_nrs (nrs: NON_RESIDENT_STUDENT) do ss[0] := nrs end
```

ST: STUD.

ss[i]
:
ss[ss.count]

```
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")
```

rs   s



sms.add_s(rs)   S s := rs RS   argument   formal pa.   p. RS

sms.add_rs(s1)   ST: STUDENT   RS rs := ST s

Type Checking Rules (4)

ST: X

$\boxed{x.f(y)}$

ST: Y

① feature f is declared in X

② ST of y is a des. of formal par. type of f.

sms : SMS

s1 : STUDENT

s2 : RS

s3 : NRS

s1.add_rs(s2)

sms.add_rs(s1)

sms.add_rs(s2)

$\boxed{\text{sms.add\_rs(s3)}}$

ST : NRS

s := s3

Is NRS descendant of RS ?

NO

```
class SMS
    add_rs (s: RS)
        do
        end ...
end
```

Type Checking Rules (5)

```
check attached {C} y as temp then
    x.f(temp)
end
```

ST: C

```
class SMS
    get_s(i: INTEGER): STUDENT
        do
            ...
        end
    add_rs(s: RS)
        do
            ...
        end
```

sms: SMS

s1: STUDENT

s2: RS

s3: NRS

```
check attached {RS} sms.get_s(1) as temp then
    sms.add_rs(temp)
end
```

```
check attached {NRS} sms.get_s(1) as temp then
    sms.add_rs(temp)
end
```

Polymorphic Collection


```
class STUDENT_MANAGEMENT_SYSETM
  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
```

iteration # | ST | DT
1 | S | RS
2 | S | NRS

(1,2)

s.item.set_pr ✗

ST: S

∵ ST Student declares it

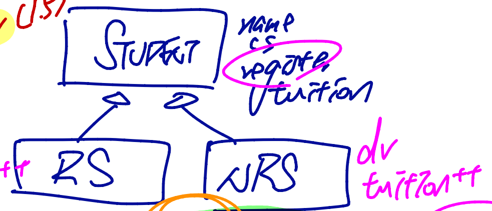
rs then

```
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
```

check {RS} s.item as rs then

rs.set_pr (1.5)

end



Feature Call Return Value

Supplier

```
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
```

ST: STUDENT

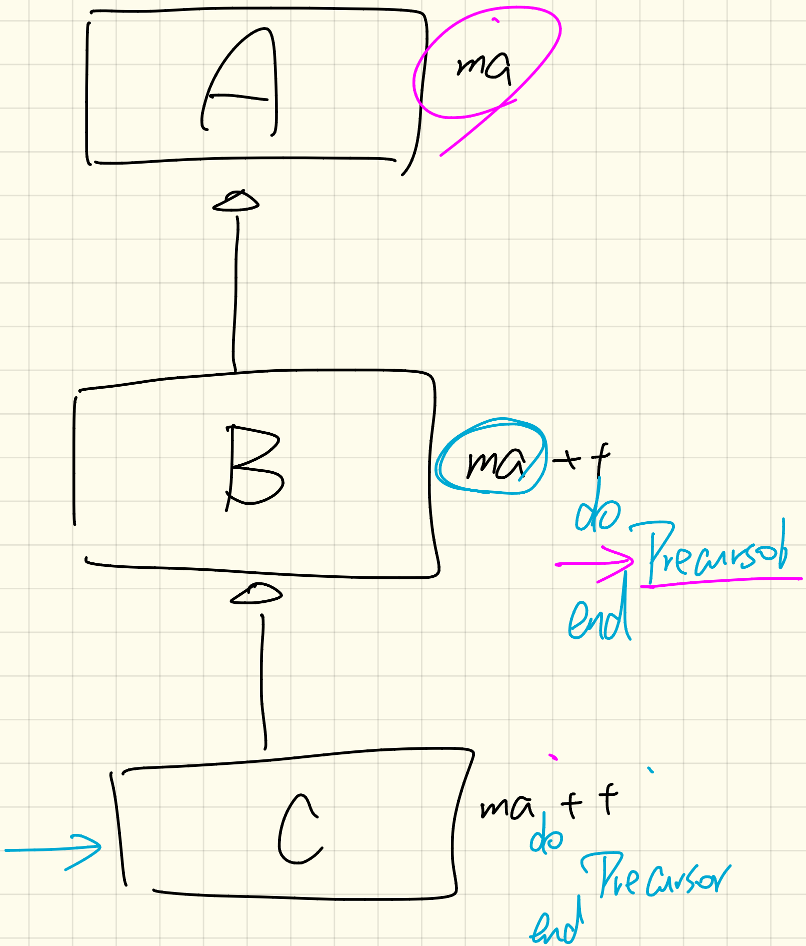
ST: S

Q: Possible DTs of Result?

Client

```
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	DT
S	RS
S	NRS


A
ma
B
ma + f
do
Precursor
end
C
ma + f
do
Precursor
end

Type Checking Rules (6)

$$\boxed{z := x.f(y)}$$

return type of f in des. of x of z

```
class SMS
    get_s(i: INTEGER): STUDENT
        do
            ...
        end
end
```

sms: SMS

s1: STUDENT

s2: RS

s3: NRS

s1 := s2.get_s(1)

s1 := sms.get_s(1)

s2 := $\boxed{\text{sms.get\_s(1)}}$ → ST: STUDENT

s3 := sms.get_s(1)

sms.get_s("2")

D_1 ss : A[STUDENT] S rs nrS

ss[c] . $\frac{\text{reg tuition}}{\text{pr}}$ X

D_2 ss: A[RS] rs

ss[c]. reg
tut
pr
sef-pr

D_3, ss: A[NRS]

General Book

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
```

ANY

DATE

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
```

ST: ANY

Monday March 4

Lecture 14

Lab Test 2

- ETF
- Undo / Redo (OOSC2 ch. 21)

General Book

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
```

ANY

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
```

b.get("SuYeon")

ANY

DATE

if attached {CI} b.get("Jim") then

check attached {CI} b.get("Jim") as ⊖ then

X

work

but

violate

SCP

end

else if

Supplier

Generic Book

b. add ("..", pn)

```
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

ST of context object b → Book[DATE]

Since we only allow DATE objects to be stored, what's returned is guaranteed to be a DATE

Client

```
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
```

STRING

DATE

b2: Book[ADDRESS]

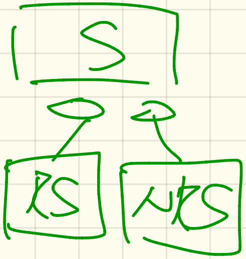
3 + 4

6 + (-2)

7 + 6

add (x, y: INT): INT
do
  x + y
end

```
class Book [~~G~~ STUD. / RS]

    add (r: ~~G~~ STUD. / RS)
        do
            - -
        end

end
```



client:

```
b1: Book[STUDENT]
b2: Book[RS]

s1: STUDENT
s2: RS
s3: NRS

① b1.add(s1)    ④ b2.add(s1)  ×
② b1.add(s2)    ⑤ b2.add(s2)
③ b1.add(s3)    ⑥ b2.add(s3)
```

class Book[G]

Supplier / Client	General Book	Generic Book
	b : Book	b : Book[ ]
	b.add(x)	b.add(__)
Any	b.get(..)	$\boxed{\text{b.get}}$

b : Book[STUDENT]

b. get(______)

b. add(______)

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:

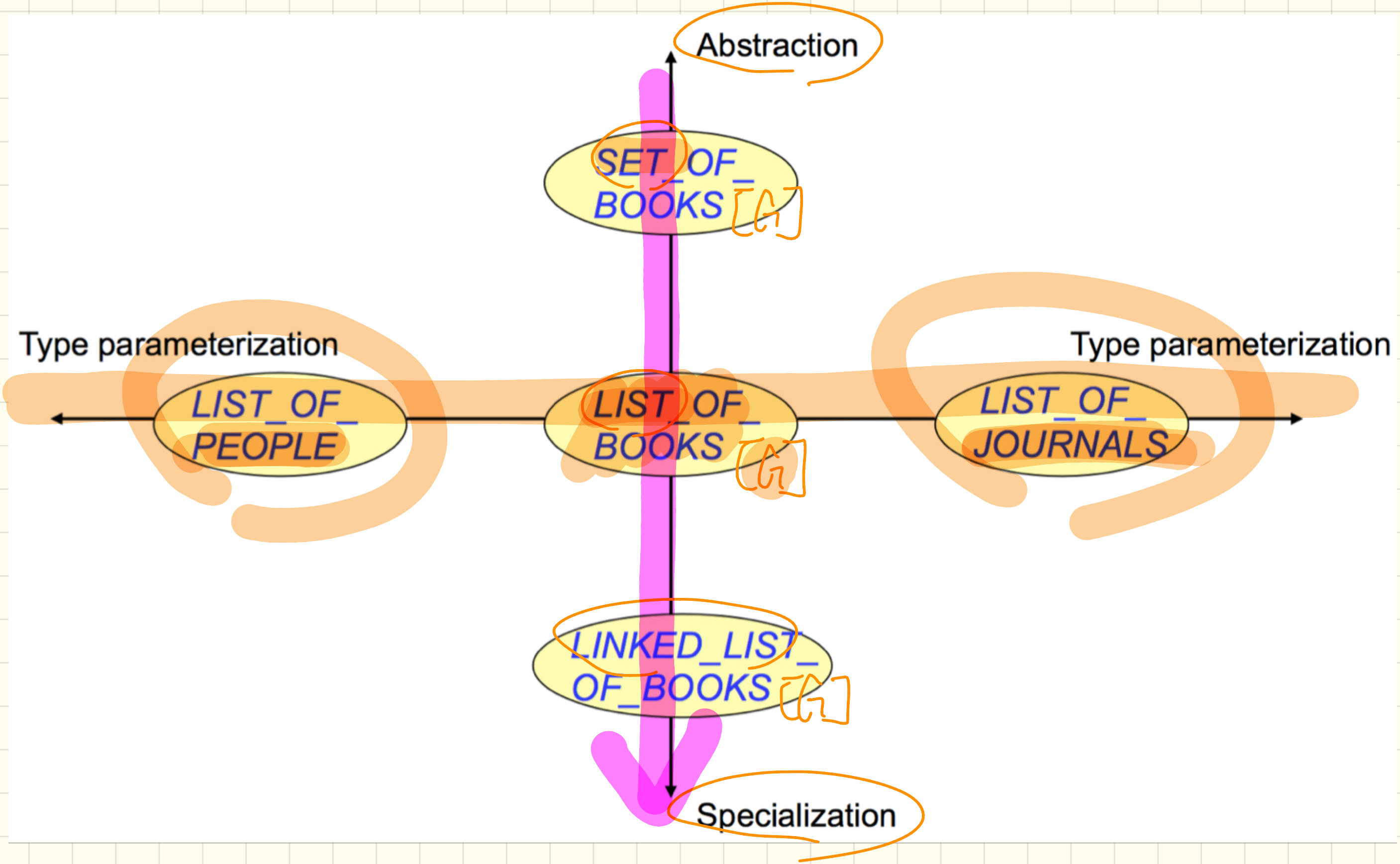
```
class DATABASE_TABLE[K, V]
  imp: DICTIONARY[V, K]
end
```

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

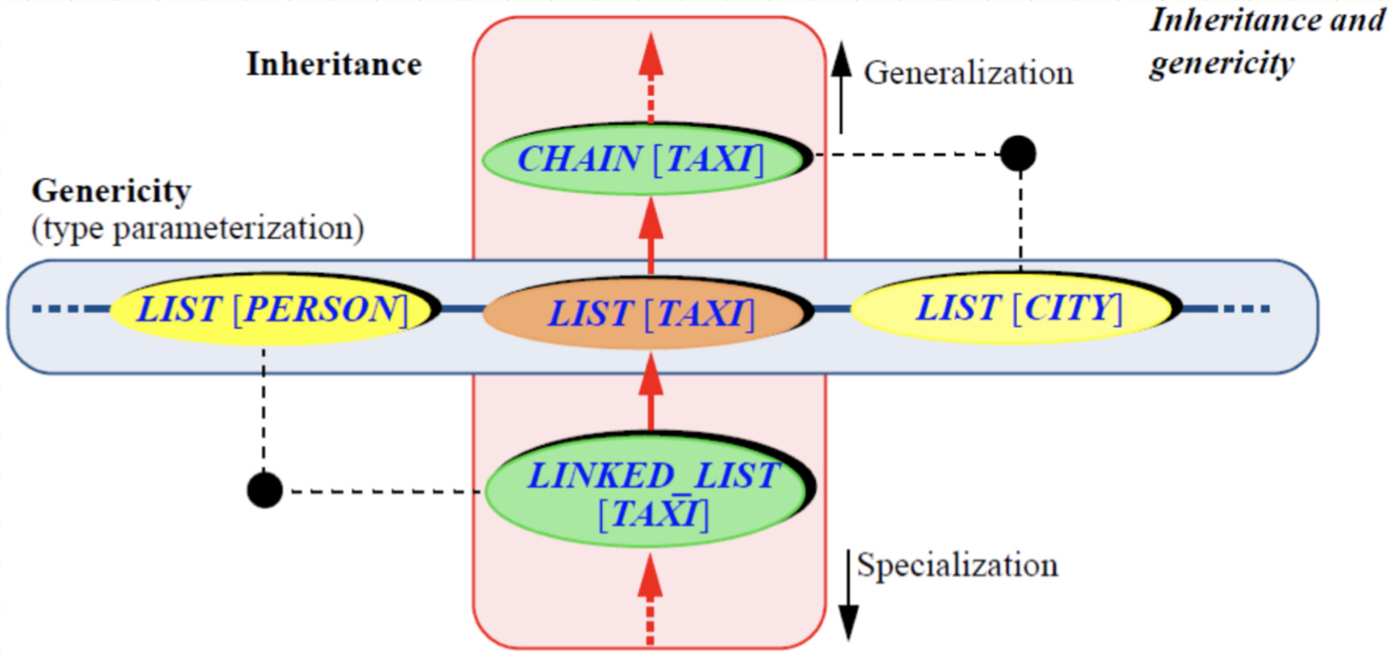
```
class STUDENT_BOOK[V]
  imp: DICTIONARY[V, STRING]
end
```

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

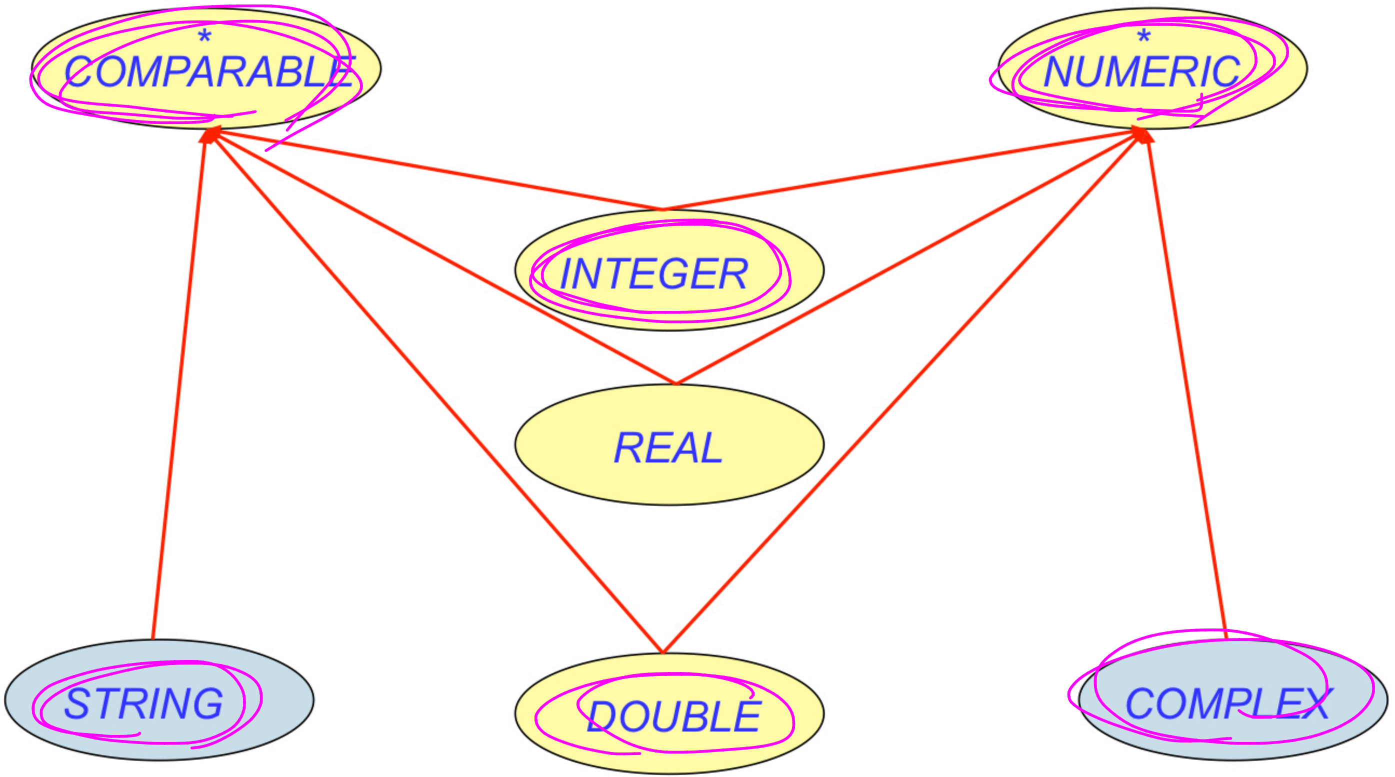
Generics vs. Inheritance (1)



Generics vs. Inheritance (2)



Multiple Inheritance : Example



D1

X

C1

change_width
change_height

C2

parent
descent
width
height

D2

C1

height
width
change_width
descendants

?

C2

parent

D3

parent
descendants

↓

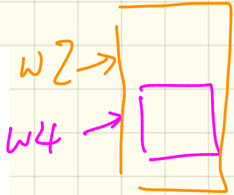
hierarchy

height
width
change_w
change_h

↓

normal attributes

Multiple Inheritance : Exercise

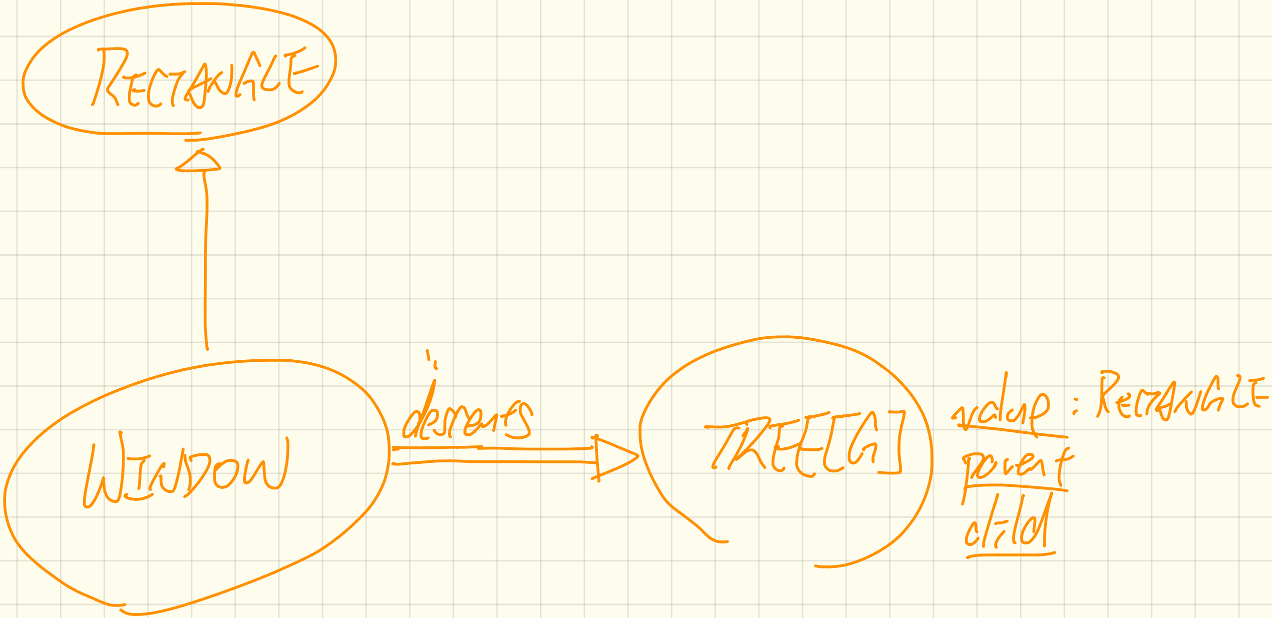



```
class RECTANGLE
  feature -- Queries
    width, height: REAL
    xpos, ypos: REAL
  feature -- Commands
    make (w, h: REAL)
    change_width
    change_height
    move
end
```

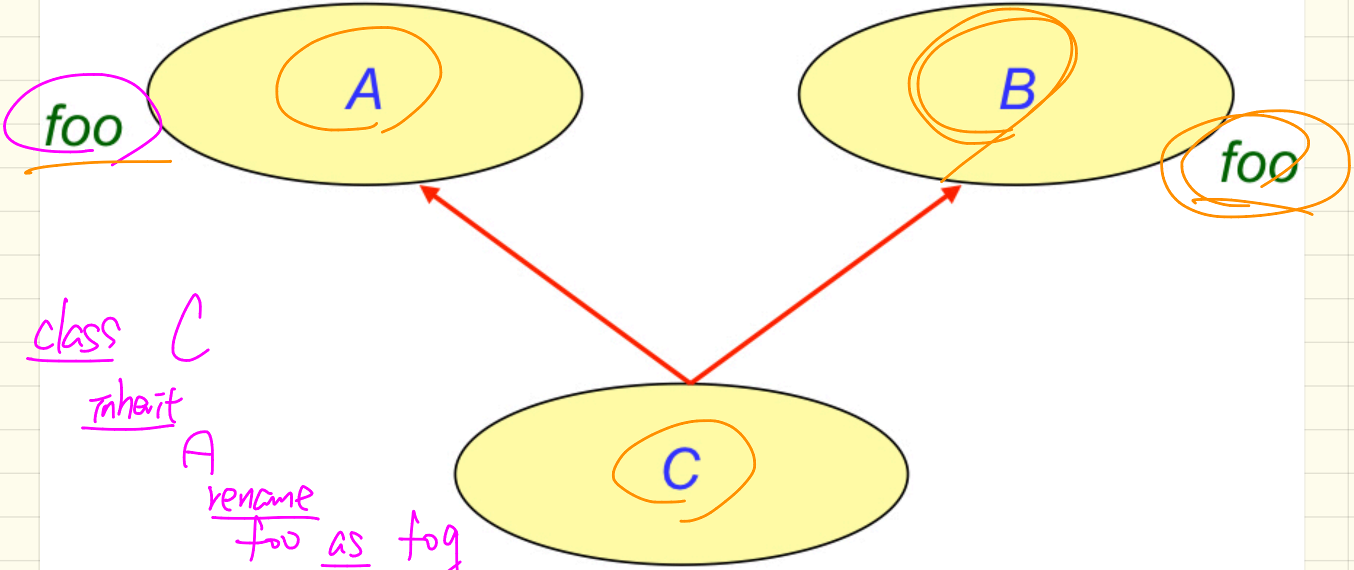
```
class TREE[G]
  feature -- Queries
    parent: TREE[G]
    descendants: LIST[TREE[G]]
  feature -- Commands
    add_child (c: TREE[G])
end
```

```
class WINDOW
  inherit
    RECTANGLE
    TREE[WINDOW]
feature
  add (w: 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
```


RECTANGLE
WINDOW
elements
TREE[G]
value : RECTANGLE
parent
child

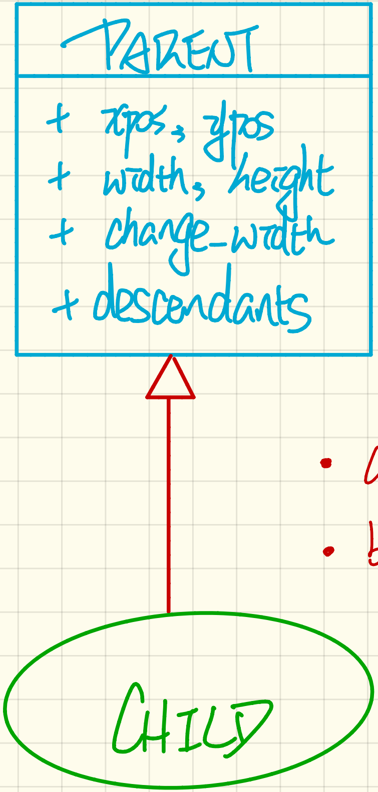
Multiple Inheritance : Name Clashes

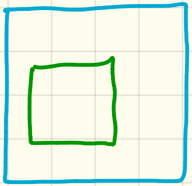



```
class C
  inherit
    A
      rename
        foo as fog
      end
    B
      rename
        foo as zoo
      end
```

Wednesday March 6

Lecture 15





- coherence
- base vs. composite

+ parent

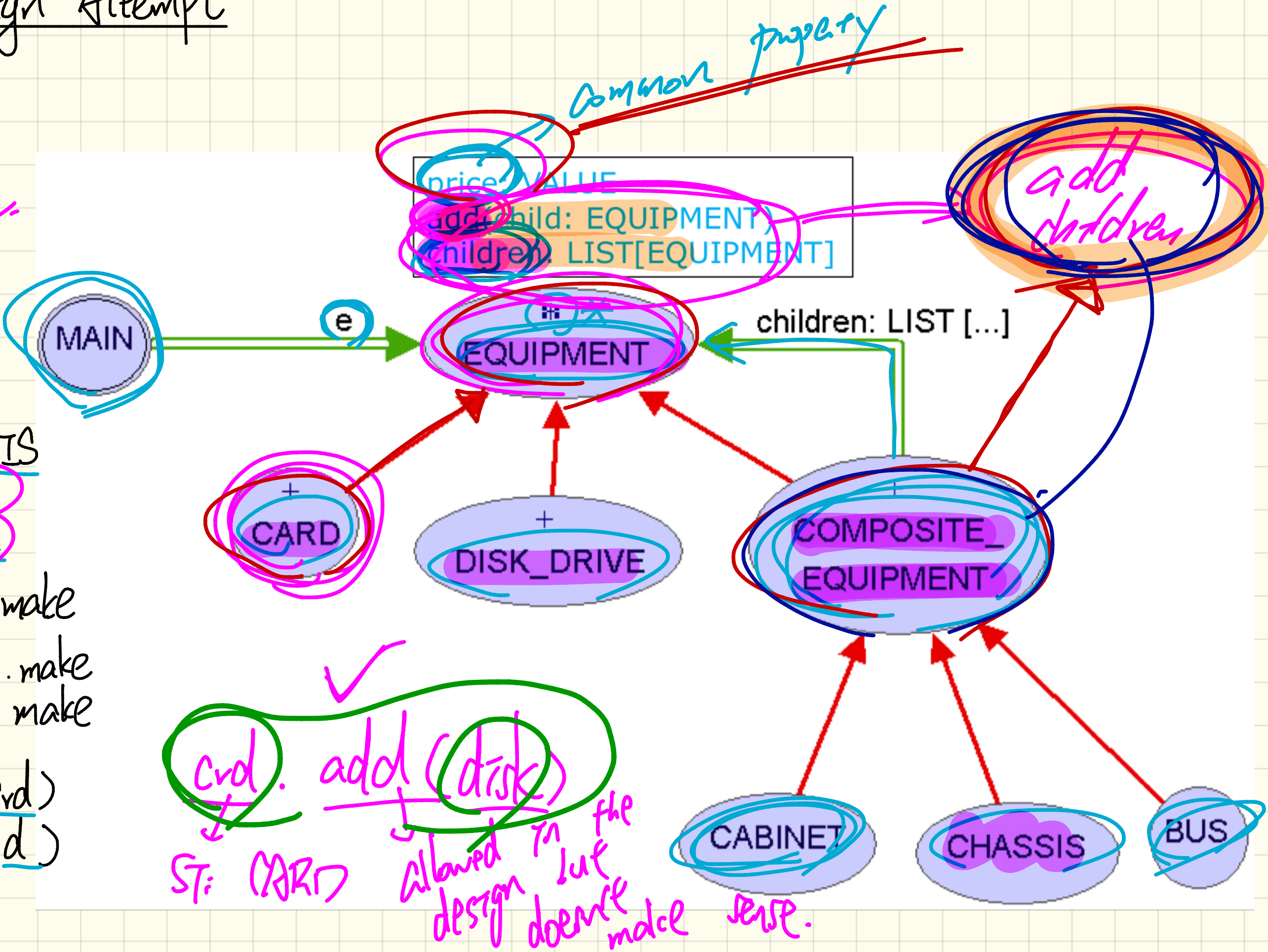
First Design Attempt

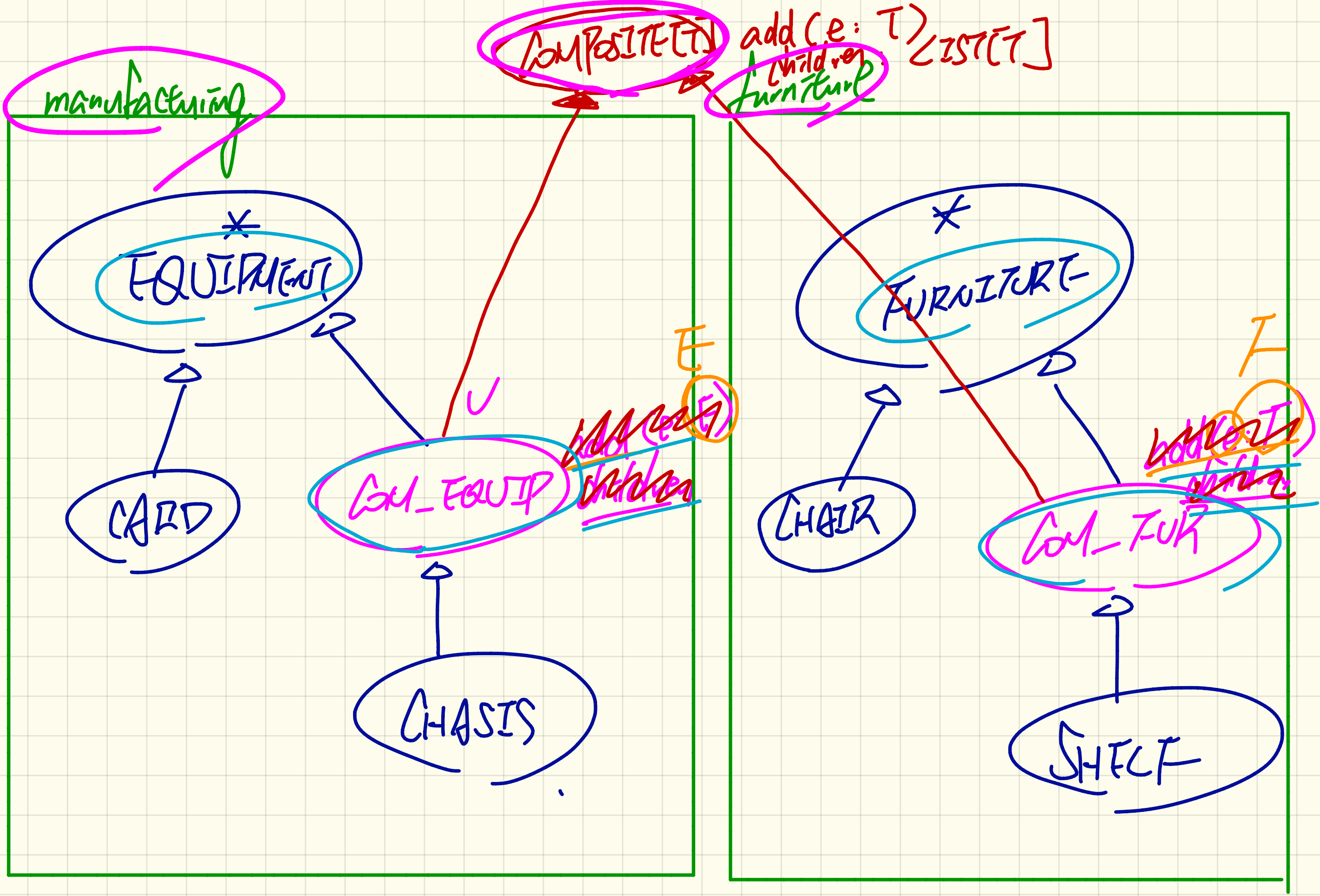
cohesion

ch: CHASIS
crd: CARD
d: DISK

create ch.make
create crd.make
create d.make

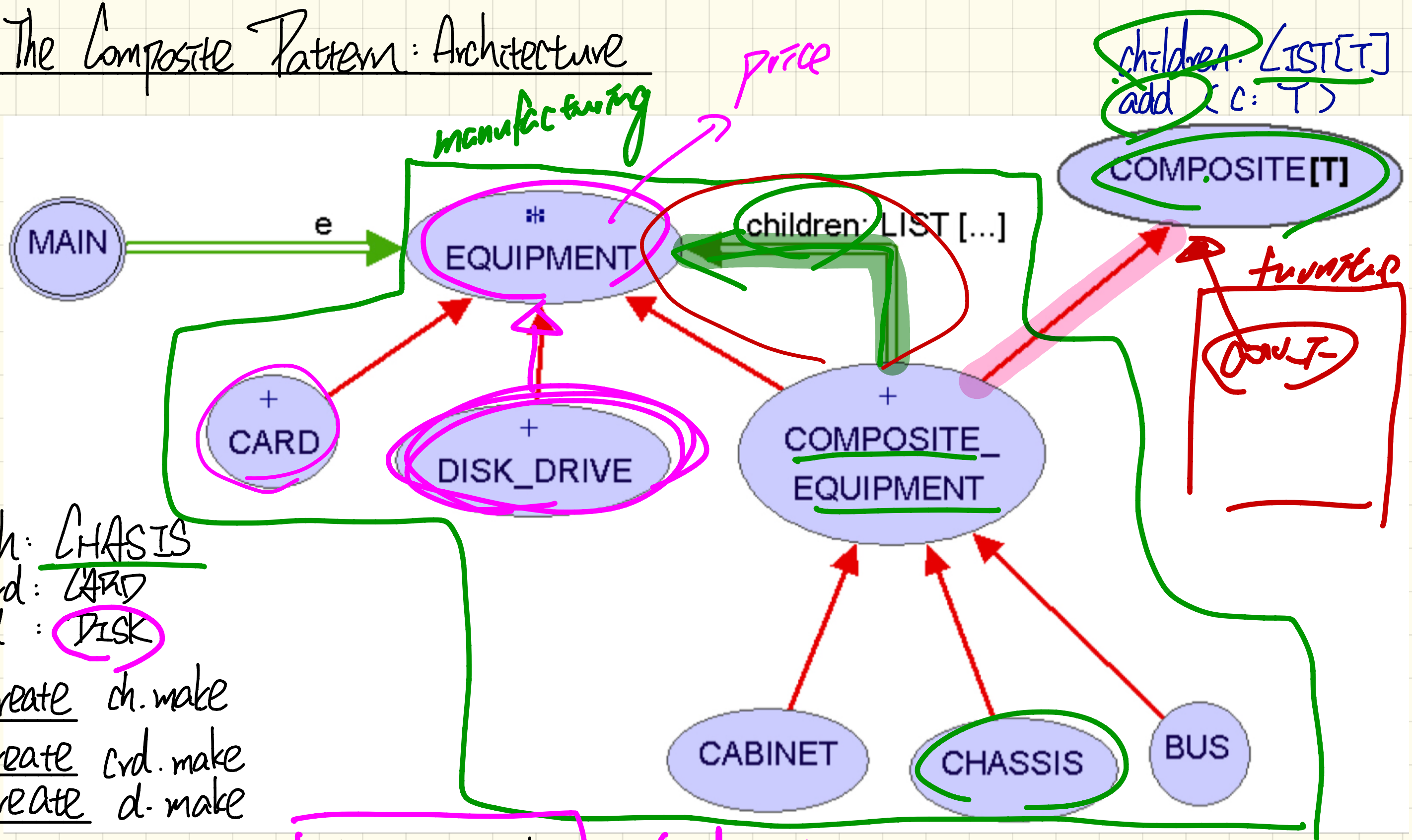
ch.add(crd)
ch.add(d)





COMPOSITE[T]
add(e: T)
children: LIST[T]
furniture
manufacturing
EQUIPMENT
CARD
COM_EQUIP
CHASIS
E
FURNITURE
CHAIR
COM_FUR
SHELF
F

The Composite Pattern: Architecture

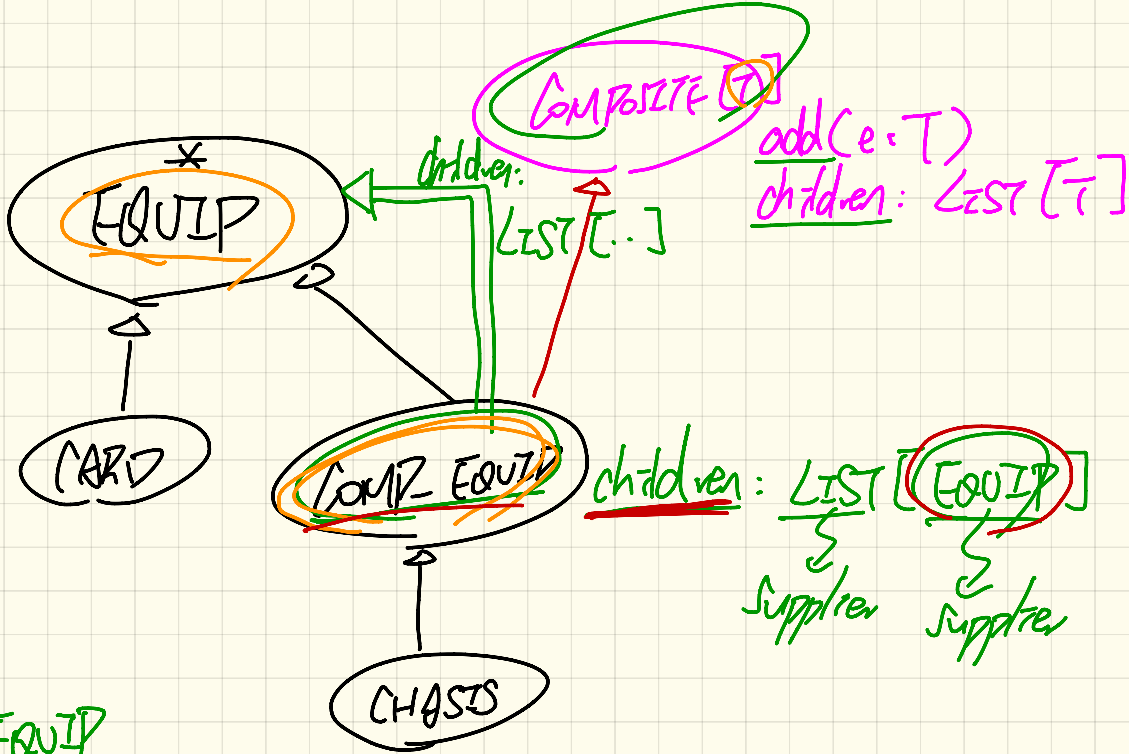


ch : CHASIS
crd : CARD
d : DISK

create ch.make
create crd.make
create d.make

ch.add (crd)
ch.add (d)

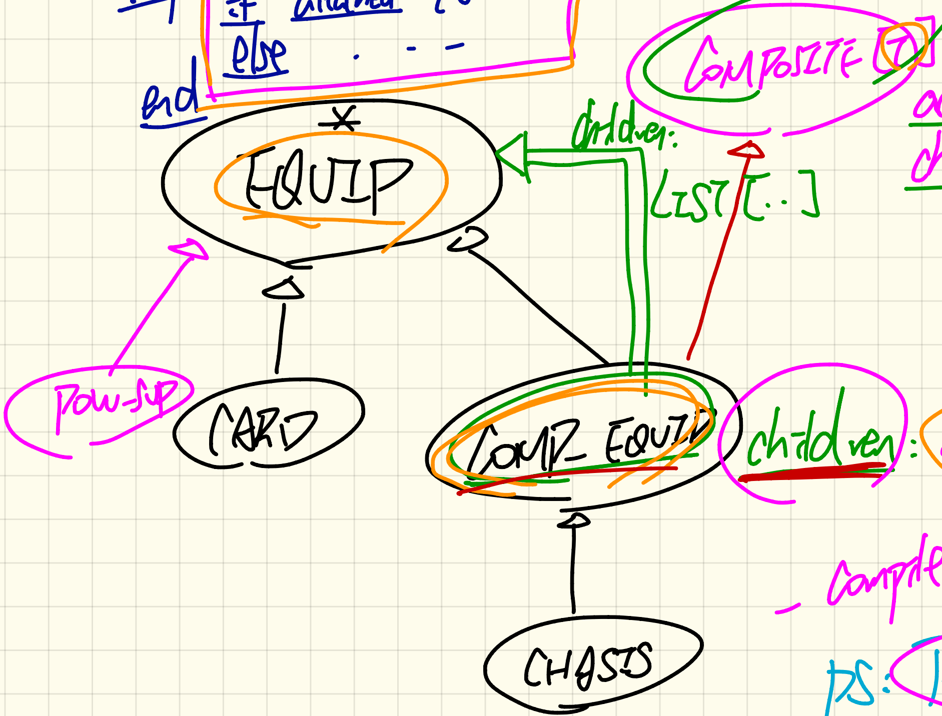
$\boxed{\text{d.add (crd)}}$ X doesn't even compile.




```
class COMP_EQUIP
  inherit
    COMPOSITE[EQUIP]
```

across children as c
loop
  if attached {EQUIP} c.item then -- end
  else ...
end

→ EQUIP
c.item.price



add(e: T)
children: LIST[T]

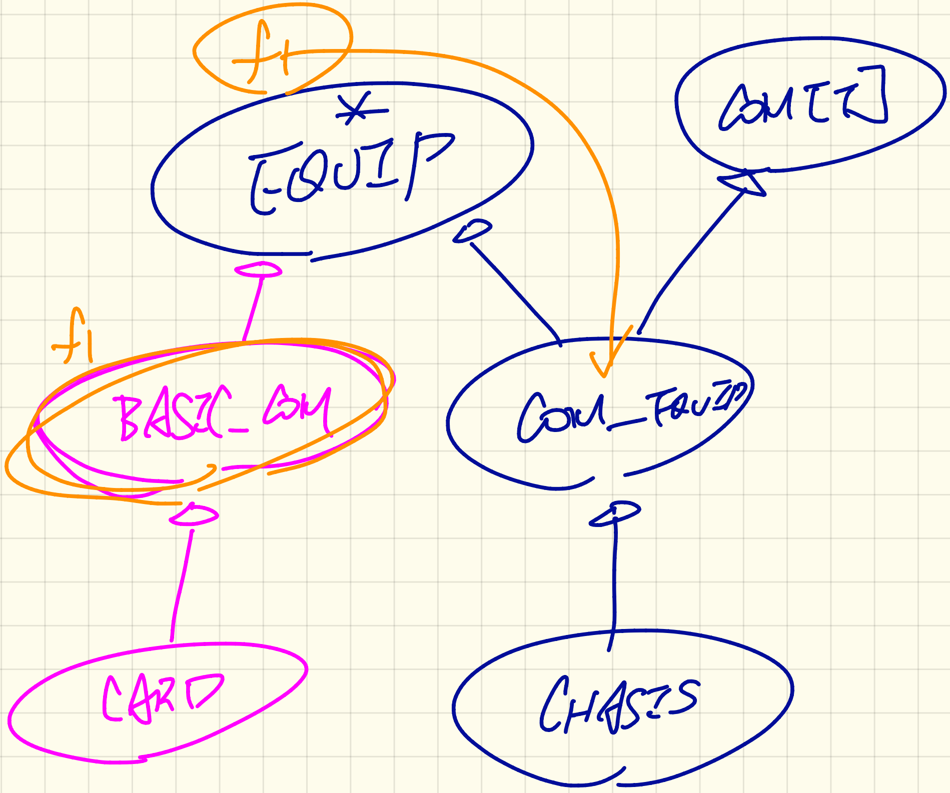
EQUIP
children: LIST[COMP_EQUIP]

- compile ✓

ps: POW_SUP
c : CABINET
c.add(ps)

ST: POW_SUP

not element



f1
EQUIP
COM[T]
f1
BASIC_COM
COM_EQUIP
CARD
CHASIS

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
```

```
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
```

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
```

cabinet.price

card.price

DT: CABINET

cabinet.price

↳ chassis.price

↳ bus.price

↳ card.price

↳ drive.price

200

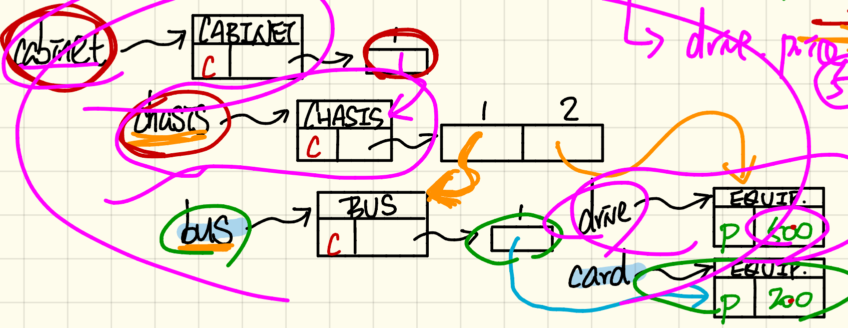
500

```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is
    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-equip
    do
      across
        children as cursor
      loop
        Result := Result + cursor.item.price
      end
    end
end
```

1. bus.price

chassis



$\frac{341}{2}$ -2

+

2 + 3 3

341 + 2

(341 + 2) + (461 + 3)

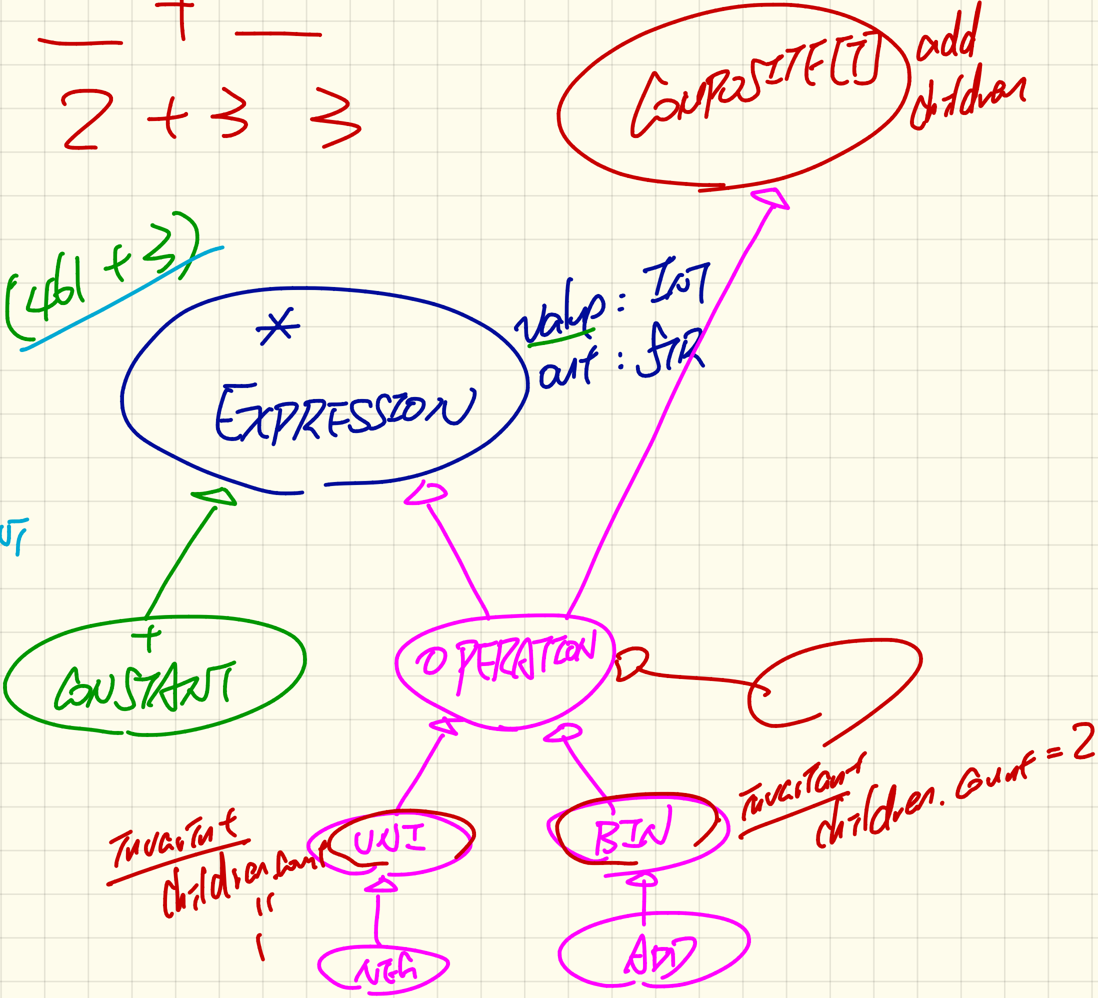
c1, c2, c3: CONSTANT

add: ADDITION

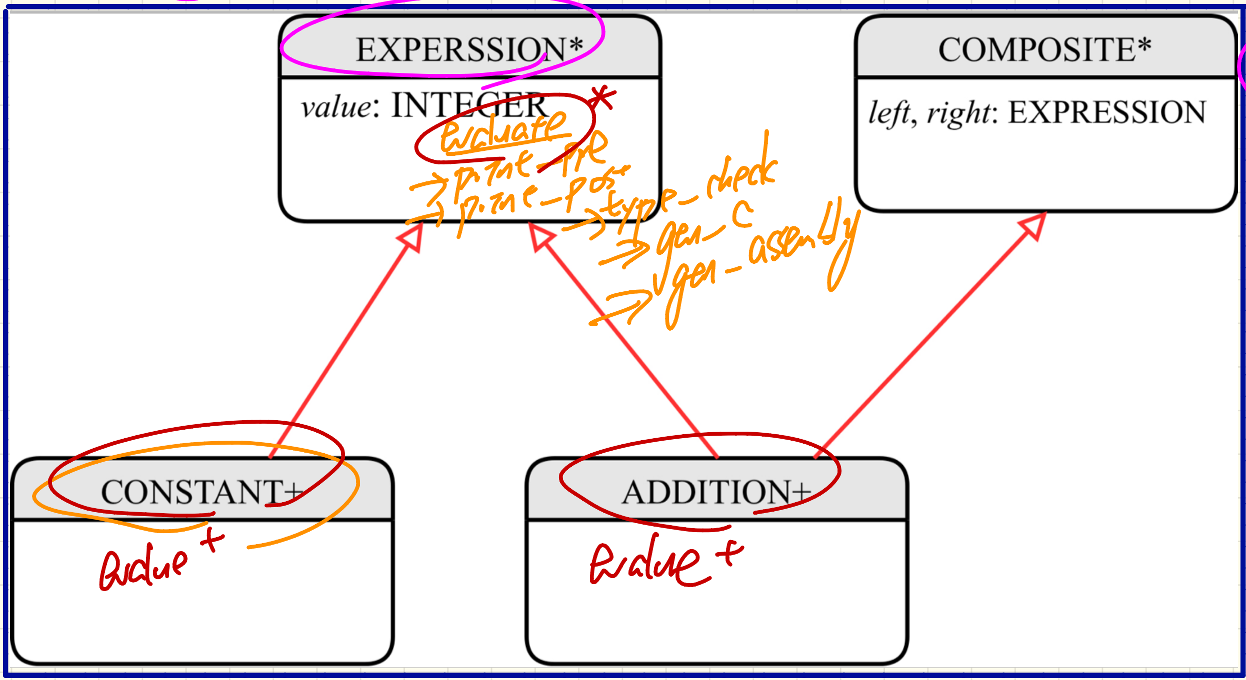
add.(c1)

add.(c2)

add.(c3)



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



Structure

Composite

Operations: evaluate
print_prefix
print_postfix
type_check

Operations

3 + 4

7

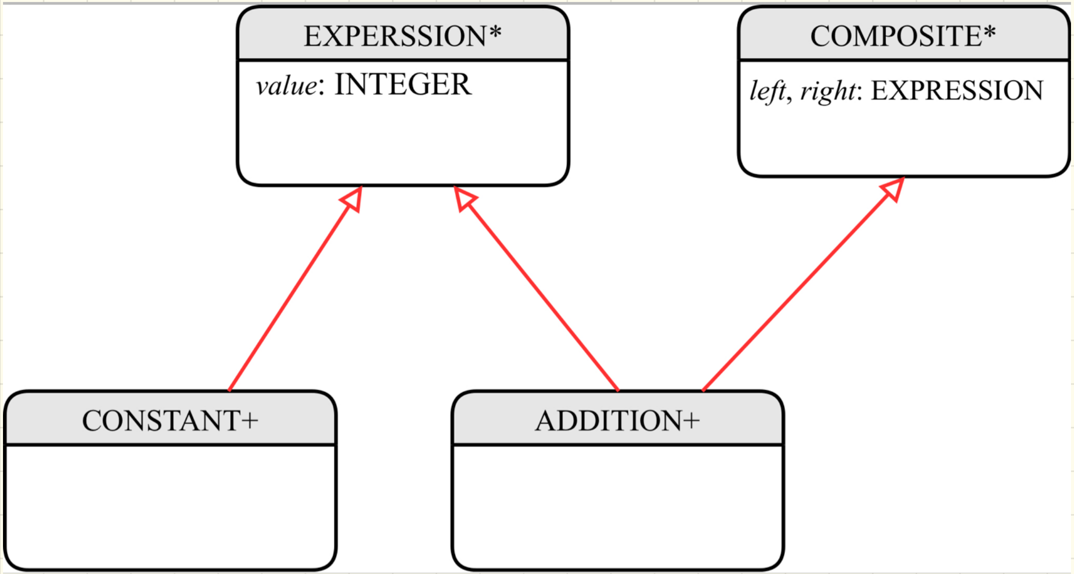
3 4 +

+ 3 4

Monday March 11

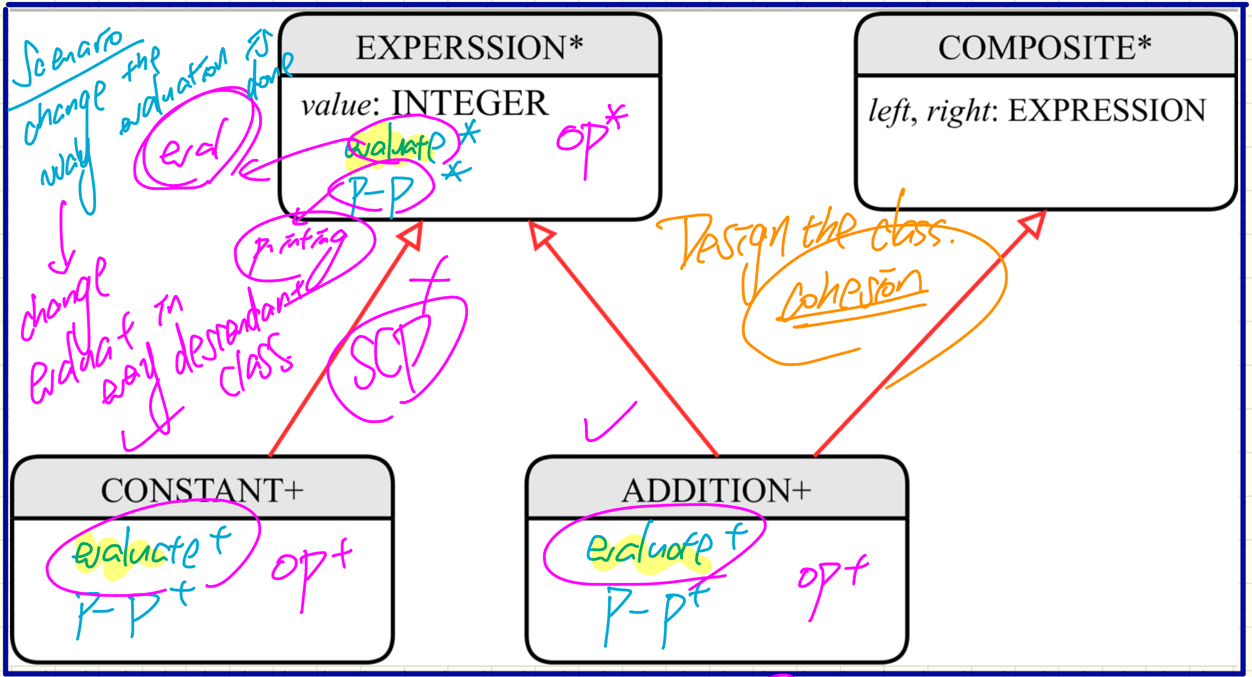
Lecture 16

Design of Language Structure: Composite Pattern



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

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

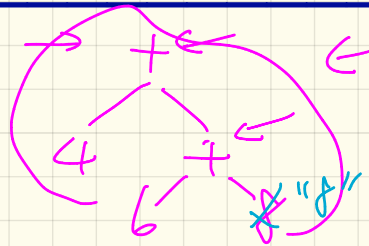


Structure

op1
op2
:
opn

Operations:	✓ evaluate
	✓ print_prefix
	✓ print_postfix
	✓ type_check

Operations

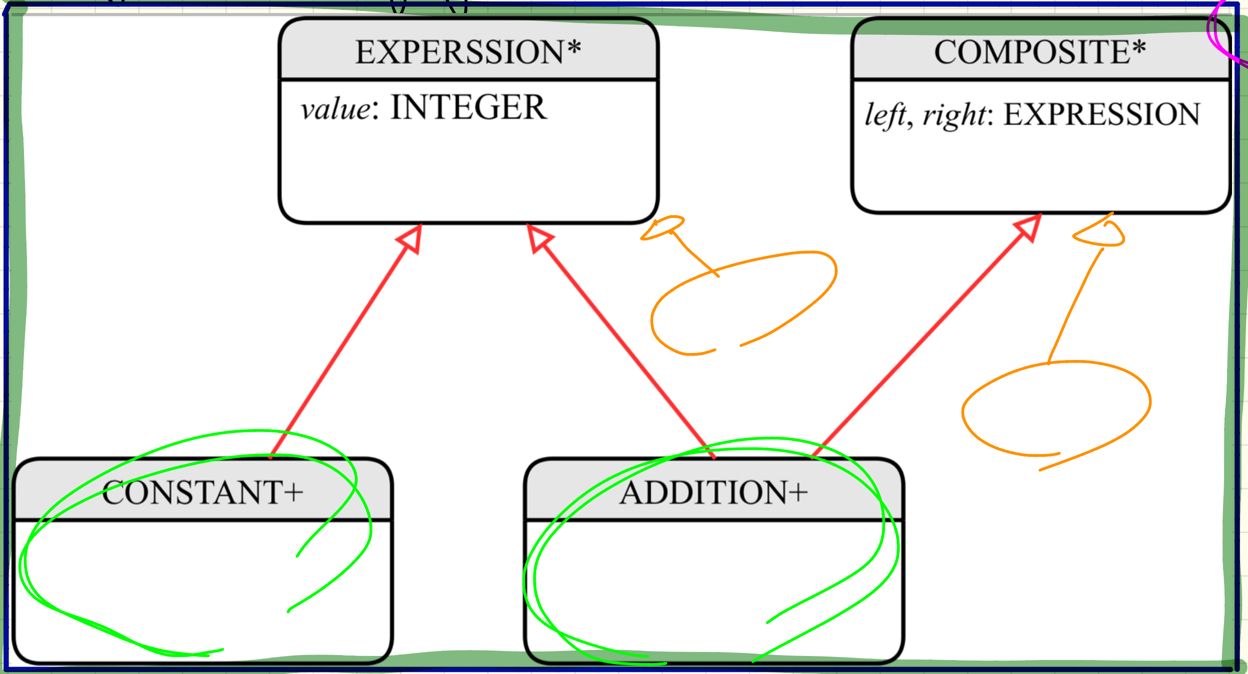


+ 4 + 6 8

4 + 6 8 +


open for extension
closed for modification

Design of a Language Application: Open-Closed Principle



Structure

both open
↳ confusing where to extend

both closed
↳ not flexible

Operations: evaluate
print_prefix
print_postfix
type_check

Operations
generate_assembly

	Structure	Operations
Alt. 1	open	closed
Alt. 2	closed	open

visitor

Visitor

open-closed principle

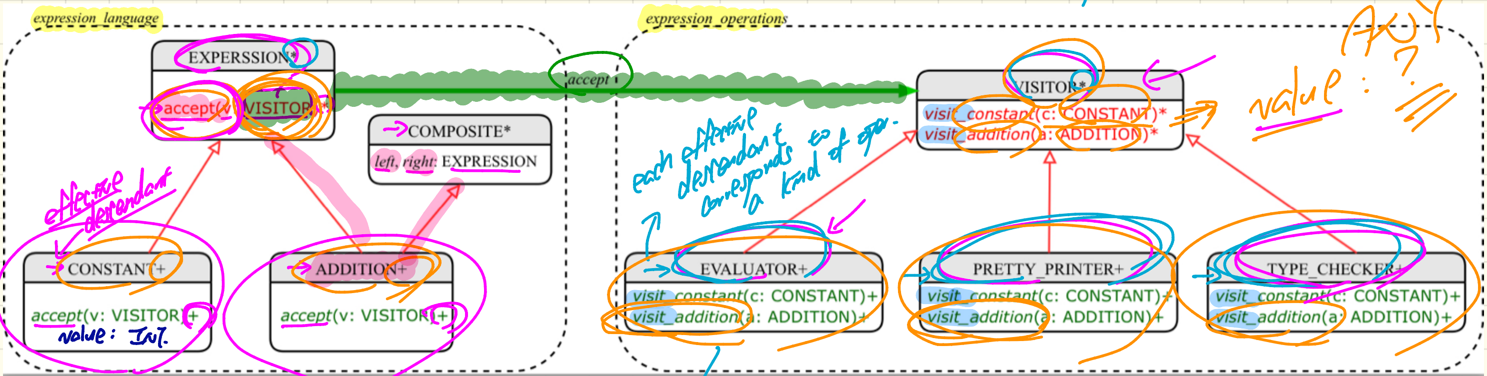
open part : operations

closed part : structure

Visitor Design Pattern: Architecture

visit_expression X

value : ANY ?



list of visit_ features correspond to the list of effective descendants of Expression.

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 is descendant of VIS

build the composite tree

v. value

Client of Visitor

1. e : $\boxed{\text{EXPRESSION}}$ → deferred

build the composite tree

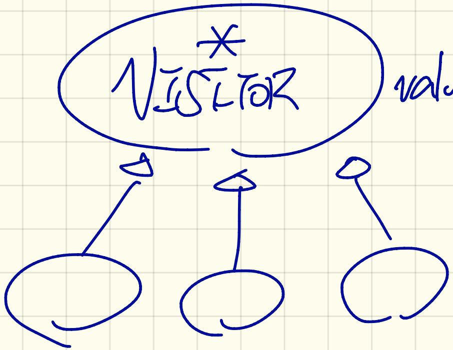
2. v : $\boxed{\text{VISITOR}}$ → deferred

attach v to a particular VISITOR type

3. e . accept (v)

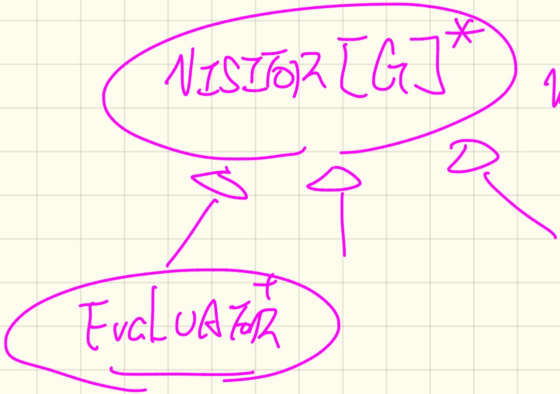
4. retrieve the result of visit from v.

Poor Design



value: ANY

Casts will be necessary ✗



value: G

class EVALUATOR
inherit
VISITOR [INT]

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
```

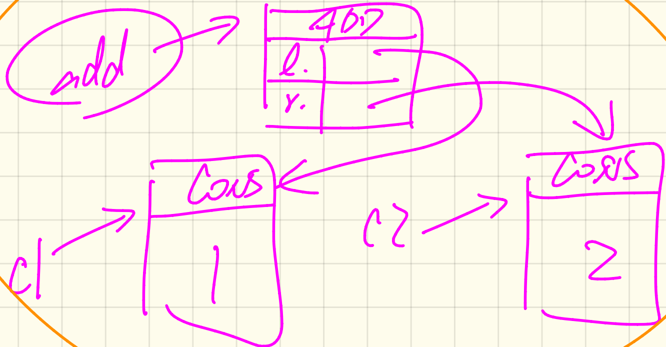
v1

v2

create {TYPE_CHECKER} v.make

add.accept (v)

Visualizing Line 4 to Line 7



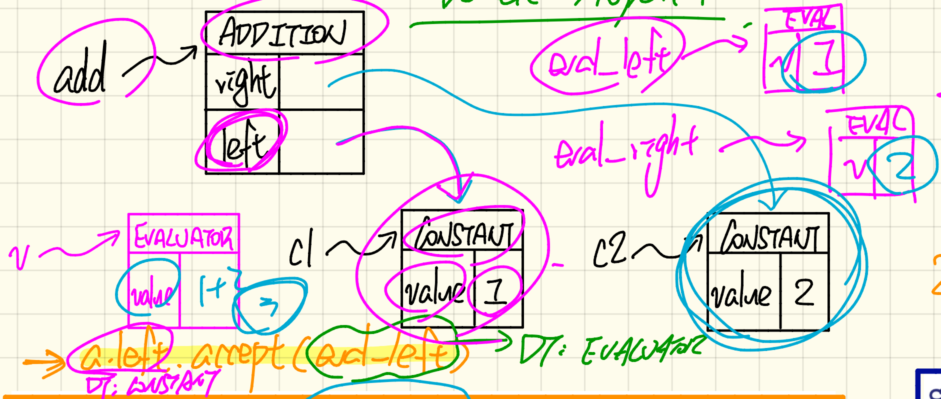
1st dispatch : scale add

2nd dispatch : EVALUATOR vs. TYPE_CHECKER.

Executing Composite and Visitor Patterns at Runtime (double dispatch)

Double Dispatch

Tracing add.accept(v)



1st Dispatch
↳ ∵ DT of add is ADD.
∴ version of accept in ADD. is called

2nd Dispatch
↳ ∵ DT of v is EVAL.
∴ version of visit_add

```
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
```

a.left

1st dispatch

1 2

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

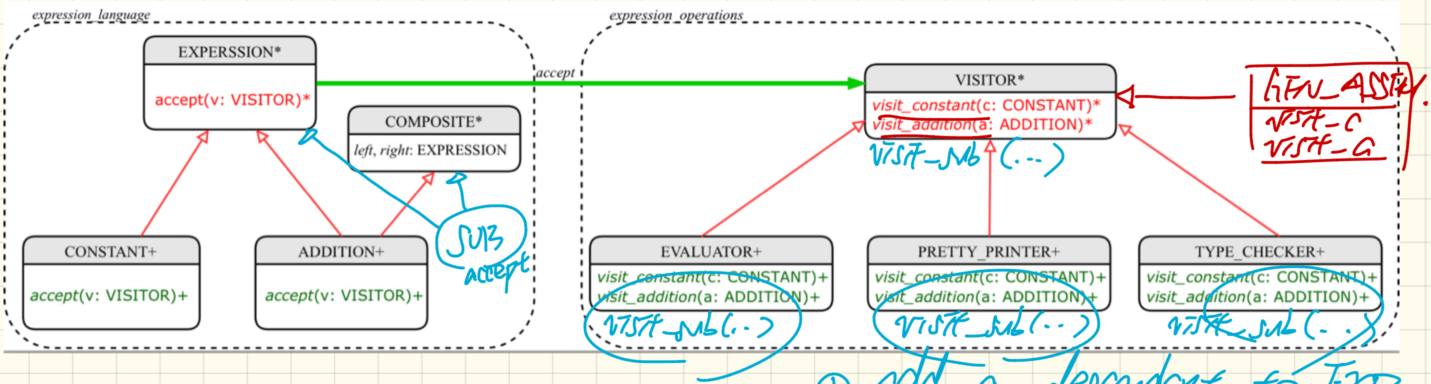
in EVALUATOR is called

a.left

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

DT: EVALUATOR

Visitor Pattern: Open-Closed and Single Choice Principles



Adding a new language construct?
↳ not good ∵ this is supposed to be closed for visitor.

① add a descendant to EXP.
② change every descendant of VISITOR
↳ violate SCP

Adding a new language operation?

✓

① add a descendant to VISITOR

Wednesday March 13

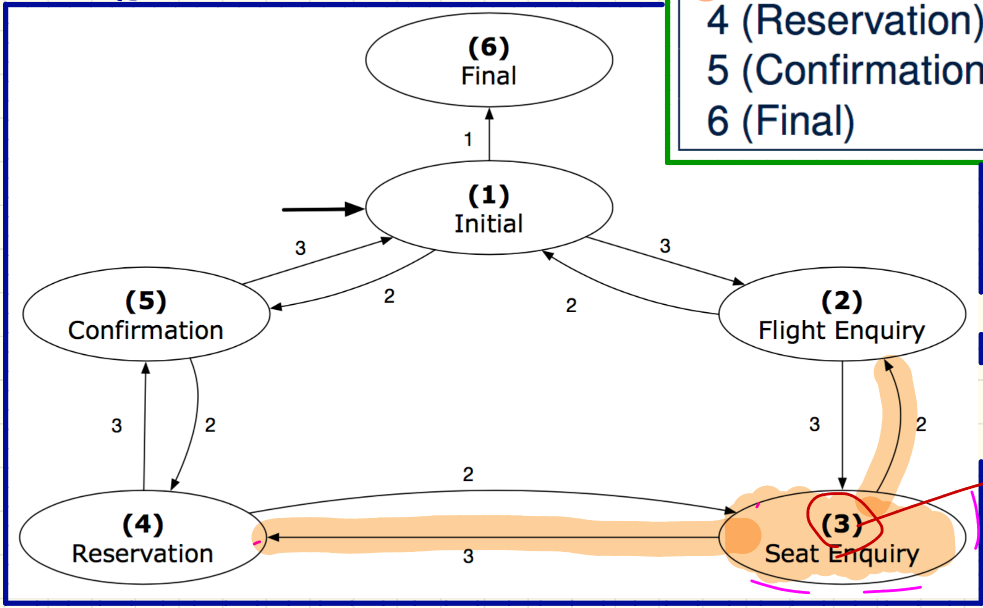
Lecture 17

State Transition Diagram (FSM)

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)	–	–	–

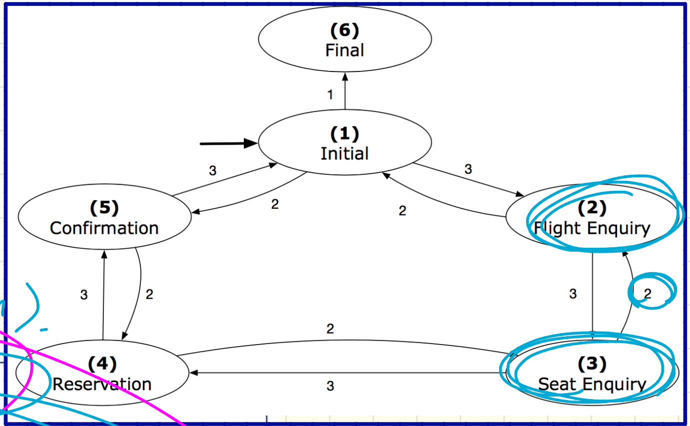
Finite State Machine



1 wrong choice

Design of a Reservation System: First Attempt

- Debugging (spa.. code).
- SCP (duplicates between labels).
- Reusability. (1. states 2. template for interaction).




```
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
```

Replay

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
```

e.g. transition (3, 2)
transition (3, 3)

states[3][2]   states[3, 2]

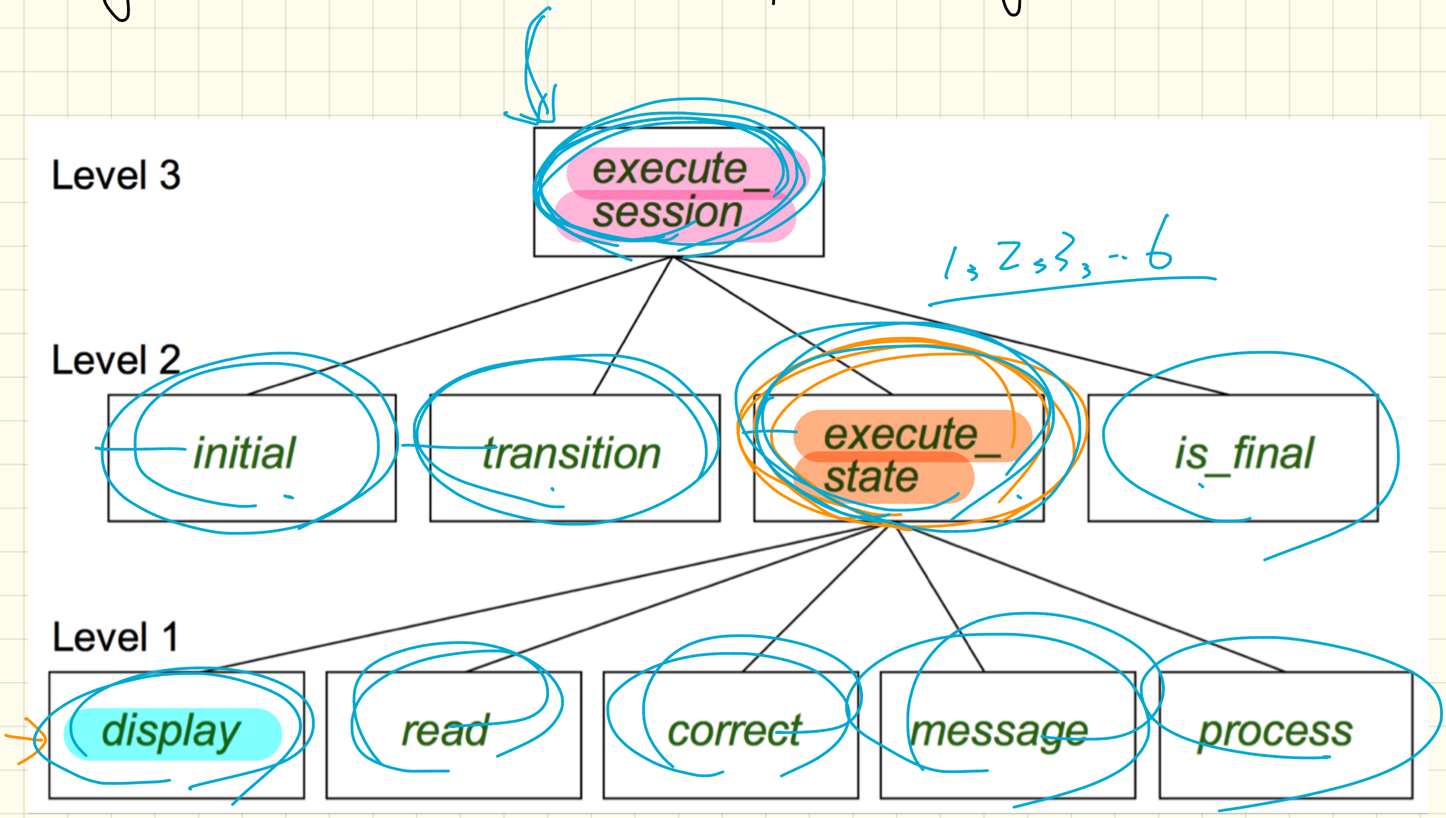
Transition Table

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)	–	–	–

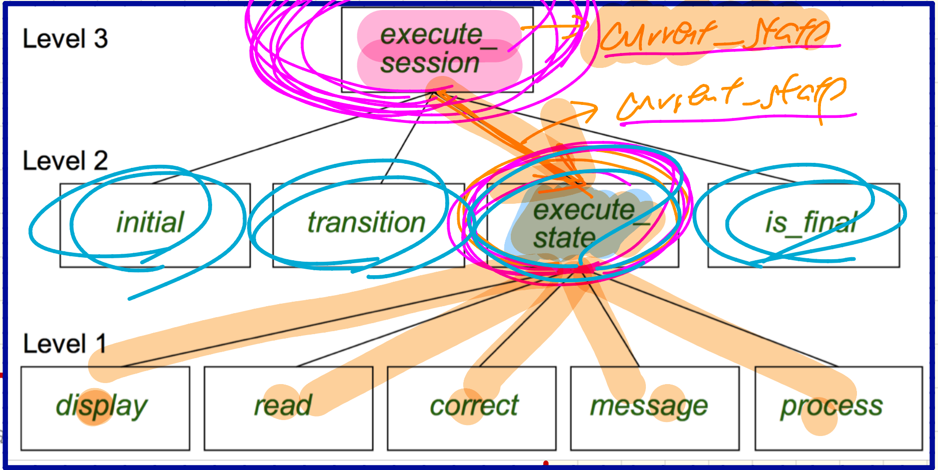
2D-Array Implementation

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: a Top-Down Design



Design of a Reservation System: Second Attempt (2)




```
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
```

assign initial state

as soon as we reach final state, stop interacting

Design of a Reservation System: Second Attempt (2)

```
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
```

: ARRAY[

case current_state of

1 :

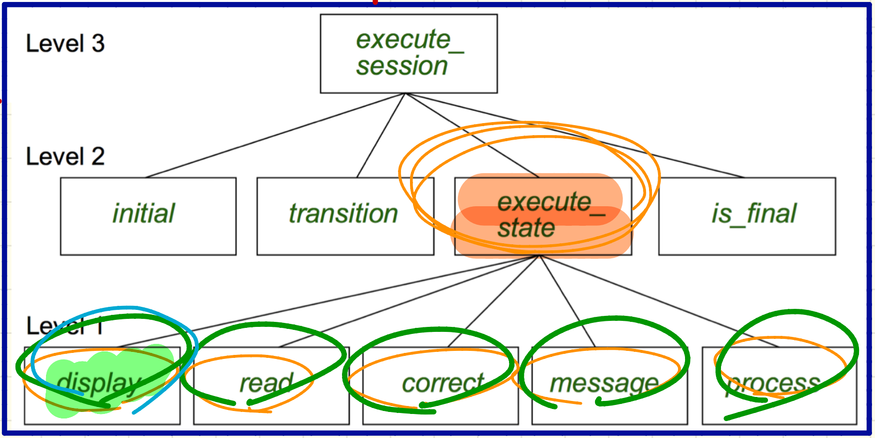
2 :

3 :

4 :

5 :

6 :



delete state 2

add state 7

display(s: INT)

if s = 1 then

...

~~elseif s = 2 then~~

~~...~~

elseif s = 3 then

...

elseif s = 7 then

...

read_answer(s: INT)

if s = 1 then

~~elseif s = 2 then~~

elseif s = 3 then

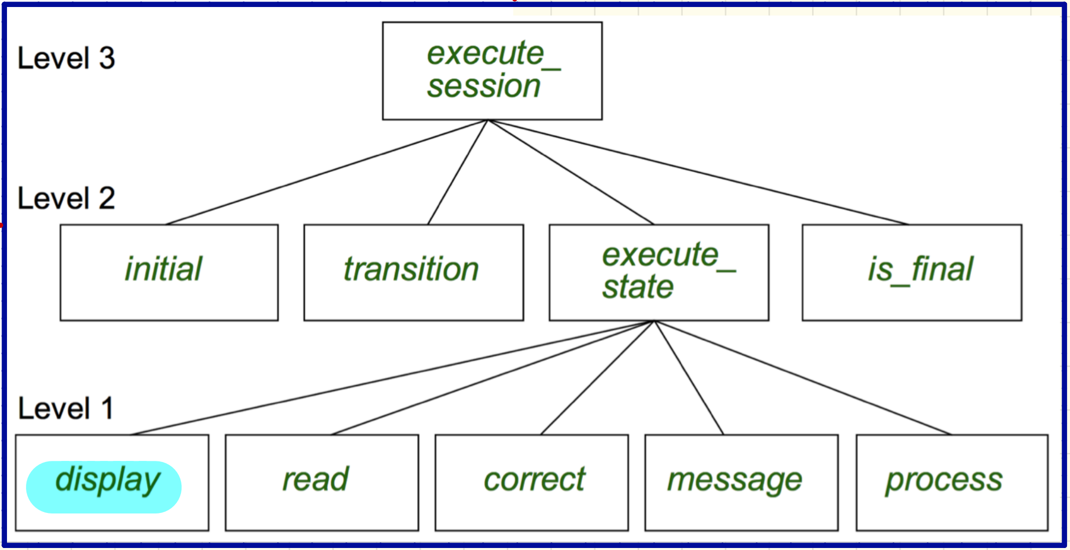
...

elseif s = 7 then

...

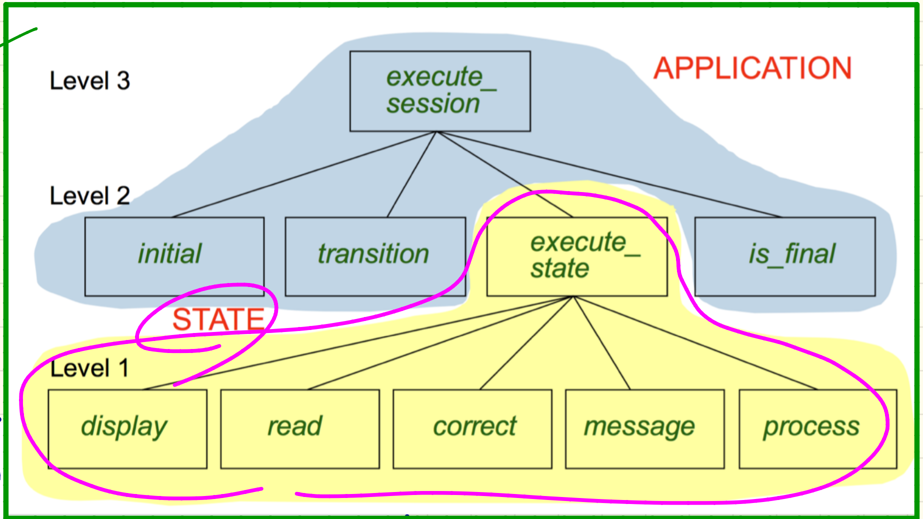
Design of a Reservation System: Second Attempt (3)

```
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
      -- Display
    end
  end
```



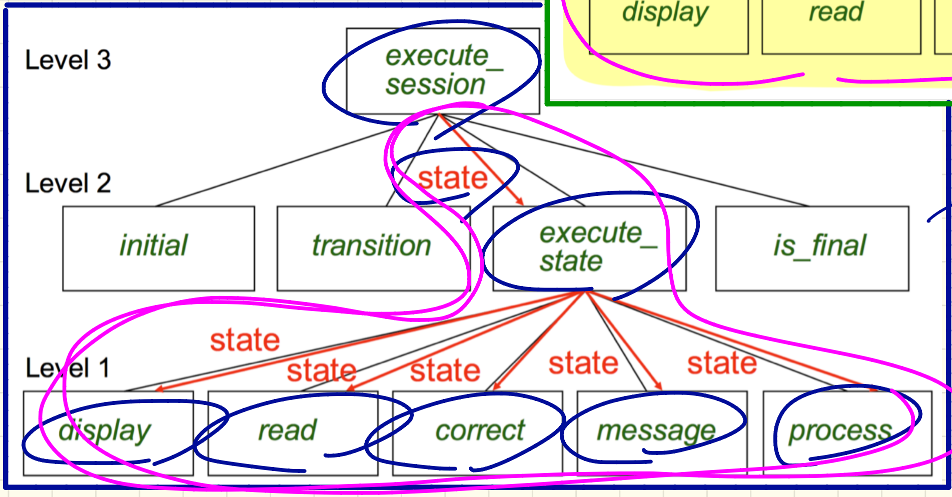
Moving from Hierarchical Design to OO Design

OO



current_state : STATE

current_state . execute_session



HIERARCHICAL

current_state : INTEGER

execute_session (current_state)

↳ read (current_state)

non-OO.

current_state := 2
→ execute_state (current_state)
current_state := 4
→ execute_state (current_state)

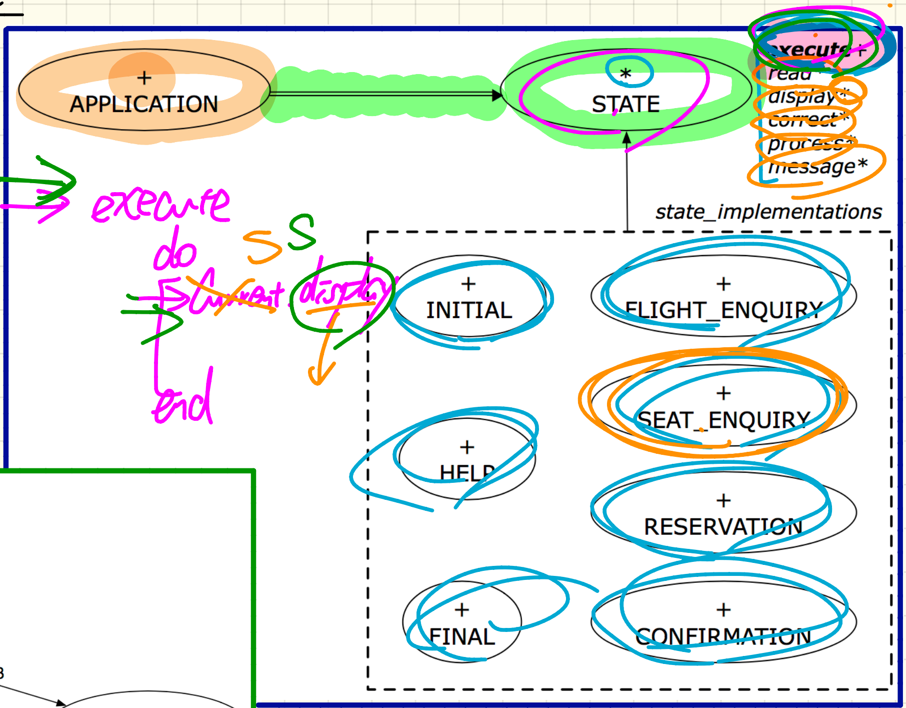
change input into context object

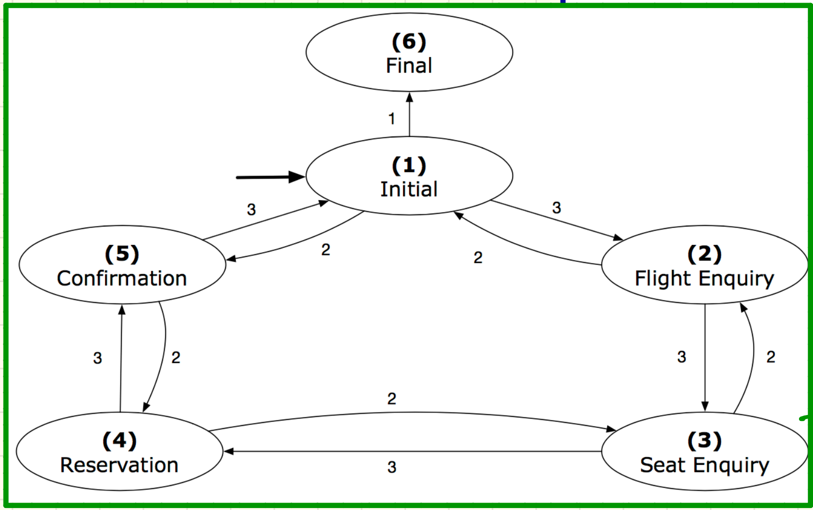
OO.

current_state : STATE

create {FLIGHT_INQ} current_state.make
→ current_state.execute
create {RESERVATION} current_state.make
→ current_state.execute

STATE PATTERN : Architecture






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

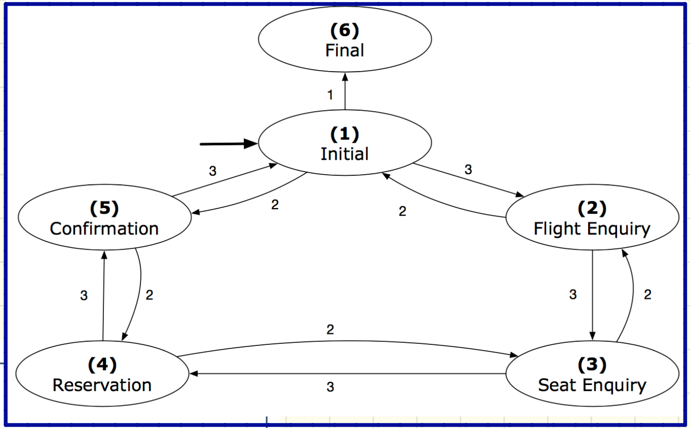
s.execute → call the S_E version of display

s.execute → call the CON. version of display

Monday March 18

Lecture 18

Design of a Reservation System: First Attempt

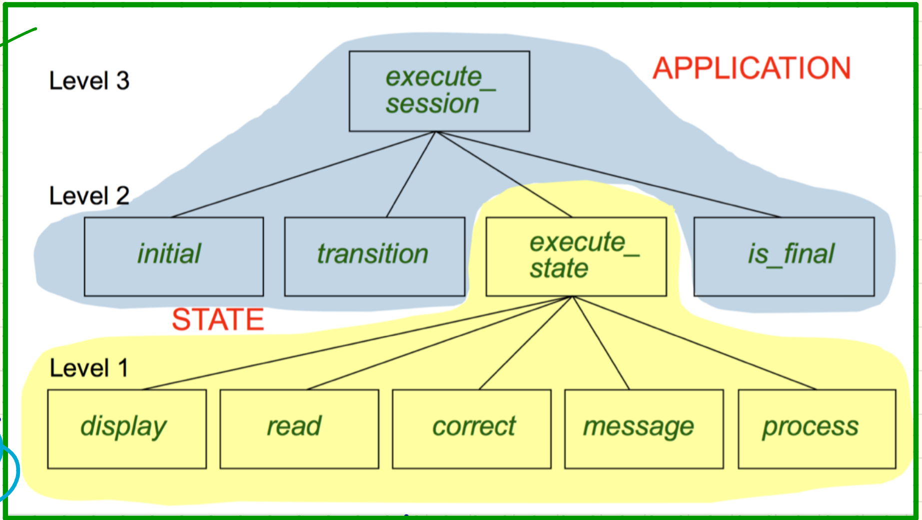



```
1_Initial_panel:
  -- Actions for Label 1.
2_Flight_Enquiry_panel:
  -- Actions for Label 2.
3_Seat_Enquiry_panel:
  -- Actions for Label 3.
4_Reservation_panel:
  -- Actions for Label 4.
5_Confirmation_panel:
  -- Actions for Label 5.
6_Final_panel:
  -- Actions for Label 6.
```

```
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
```

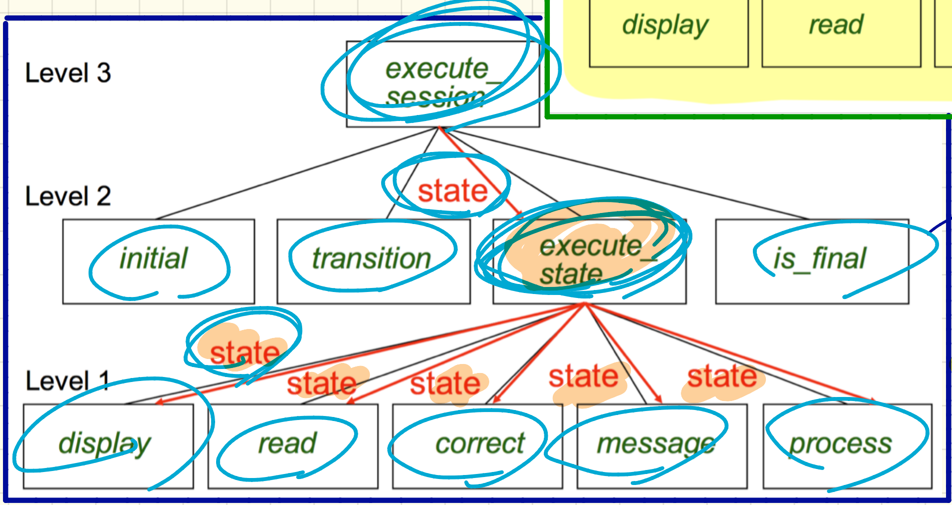
Moving from Hierarchical Design to OO Design

OO



current_state : STATE

current_state.execute_session



HIERARCHICAL

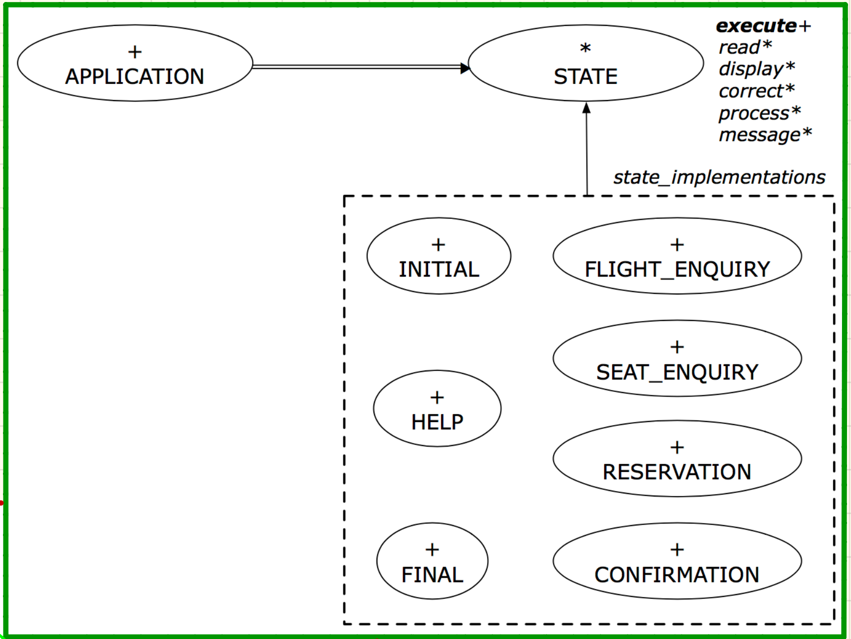
current_state : INTEGER

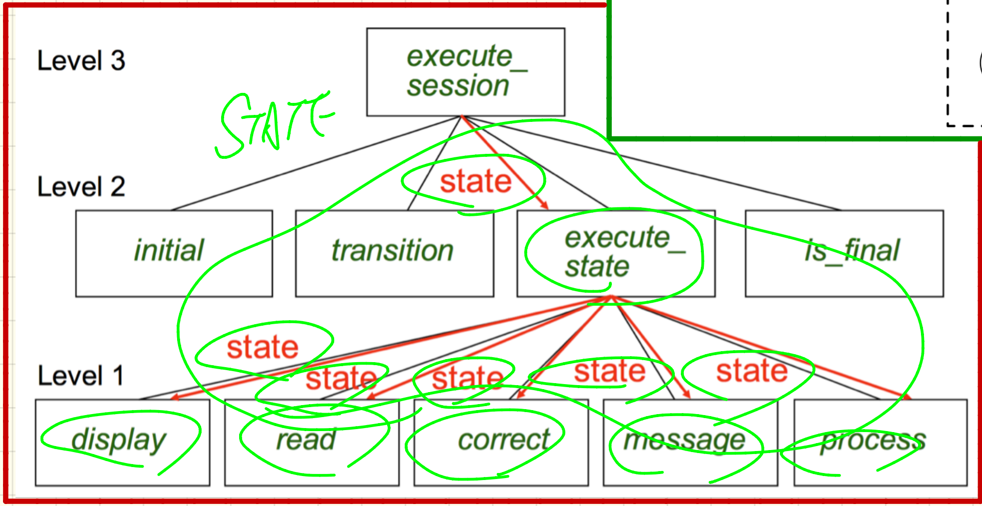
execute_session (current_state)

Interactive System: Non-OO vs. OO

current_state : STATE

current_state.execute_session





current_state : INTEGER

execute_session (current_state)

→ s1. execute

→ s2. execute

Non-OO

→ execute_session (cs: INT)

do

display (cs)

read_answer (cs)

end

OO (State Pattern)

class STATE

execute

do

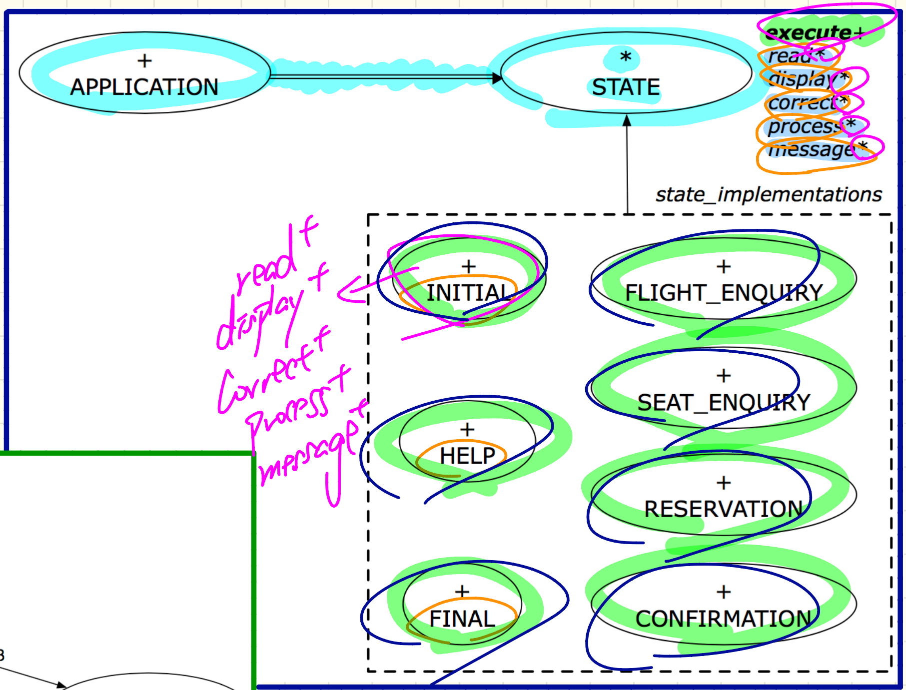
s1 s2 ~~current~~.display

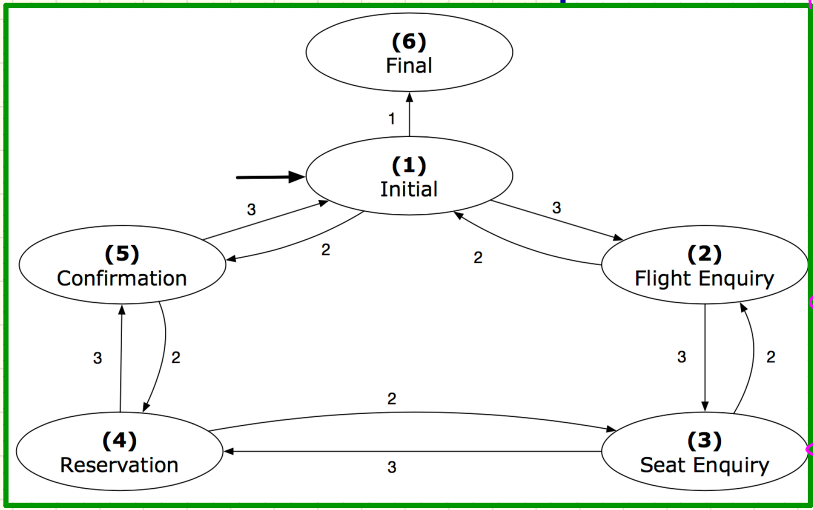
s1 s2 ~~current~~.read_answer

end

end

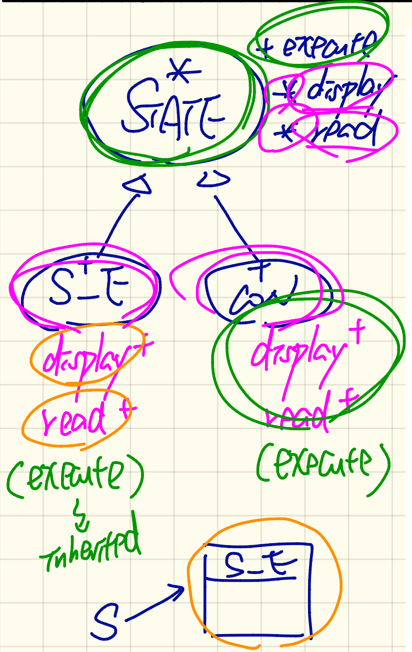
STATE PATTERN : Architecture






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

STATE PATTERN: STATE Module




```
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
```

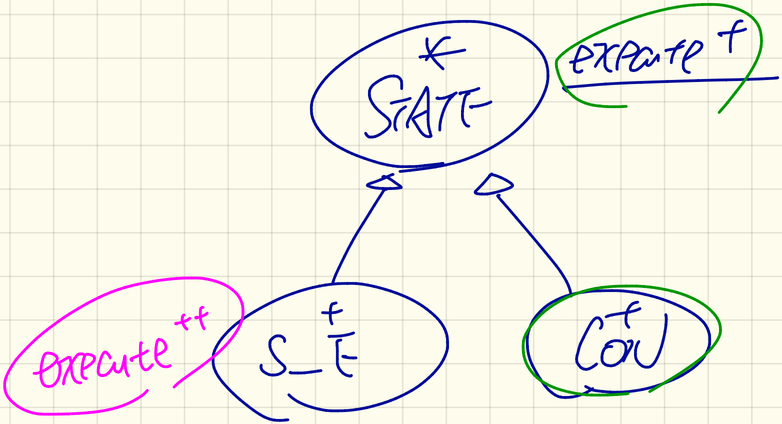
```
execute
  local
    good: BOOLEAN
  do
    from
    until
      good
    loop
      display
        -- set answer and choice
      read
      good := correct
      if not good then
        message
      end
    end
    process
  end
end
```

pattern of calling helper features

TEMPLATE

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

→ version in STATE



s : STATE

create {S_E} s.make

s.execute ← execute^{++}

create {CON} s.make

s.execute ← execute^+

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
```

STATE PATTERN: TEST

```
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 1.
    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
```

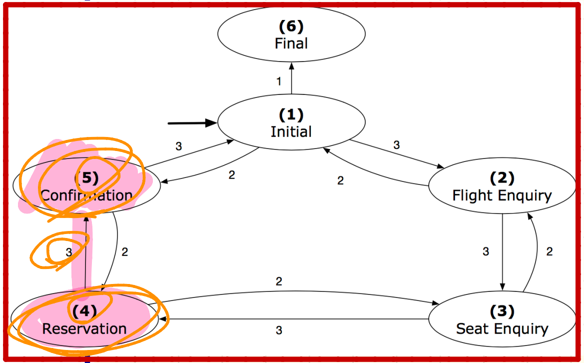
#states

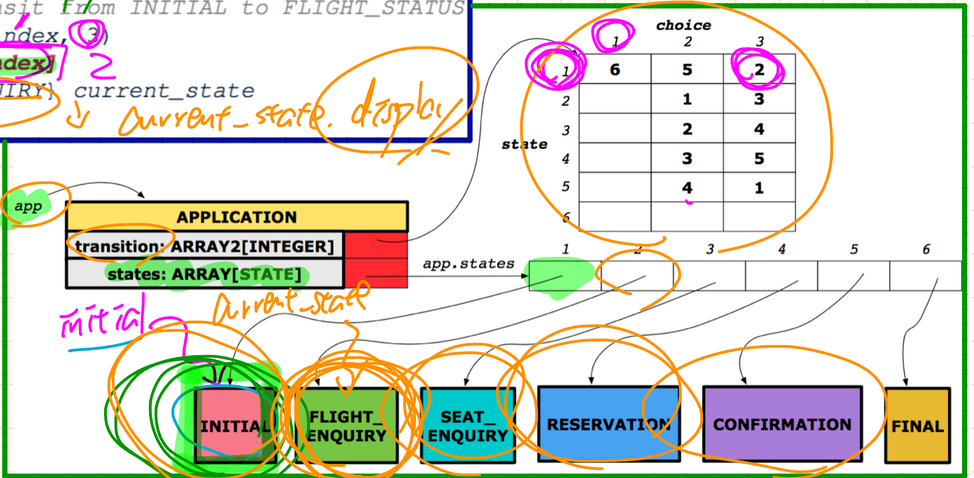
trans.

current_state.display

2

current_state.display

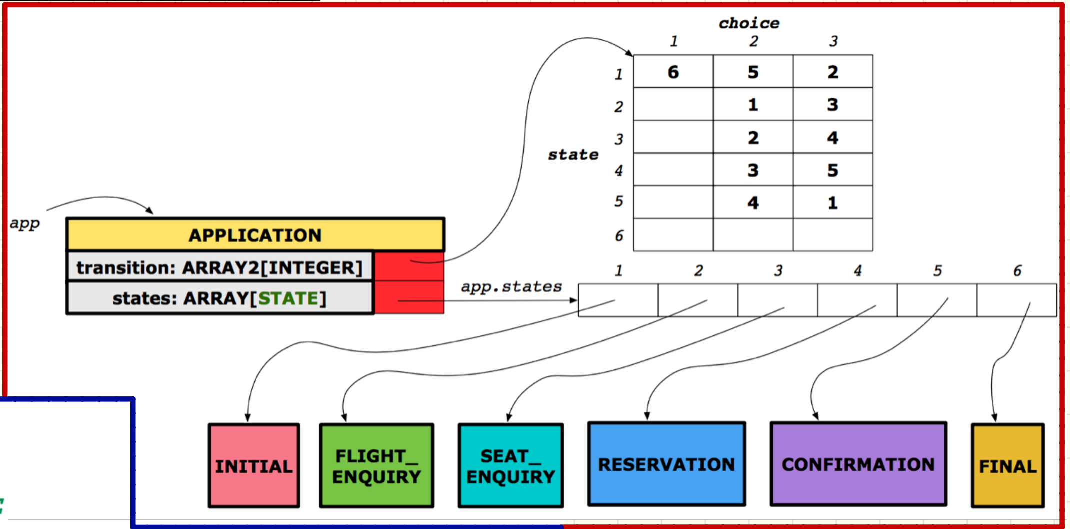




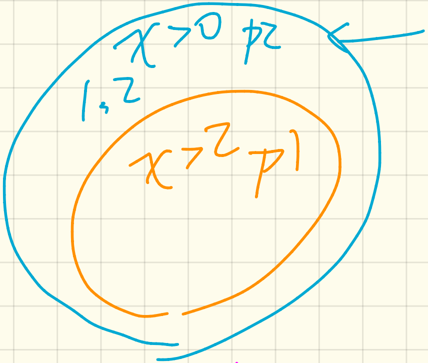
app.put_trans(6, 1, 1)
src act

app.put_trans(5, 4, 3)

State Pattern: Interactive Session




```
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
```



sqrt(x: Int)

P_1: $x > 2$ → 3, ...

P_2: $x > 0$ → 1, 2, 3, ...

- P_2 require less than P_1

= P_1 vs. P_2 which one is correct?

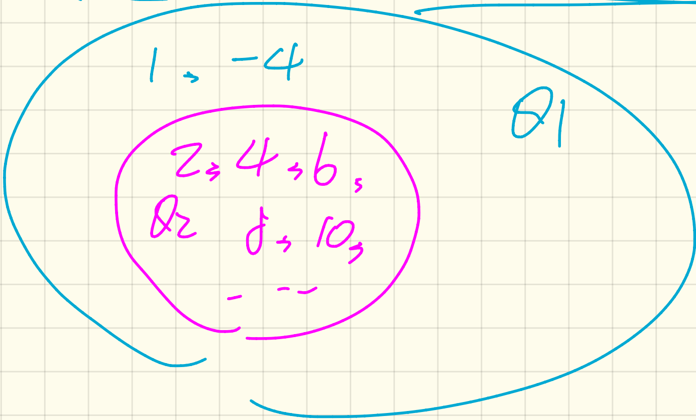
It's up to your design decision.

Q_2 : $Result = (i > 0) \wedge (i \bmod 2 = 0)$

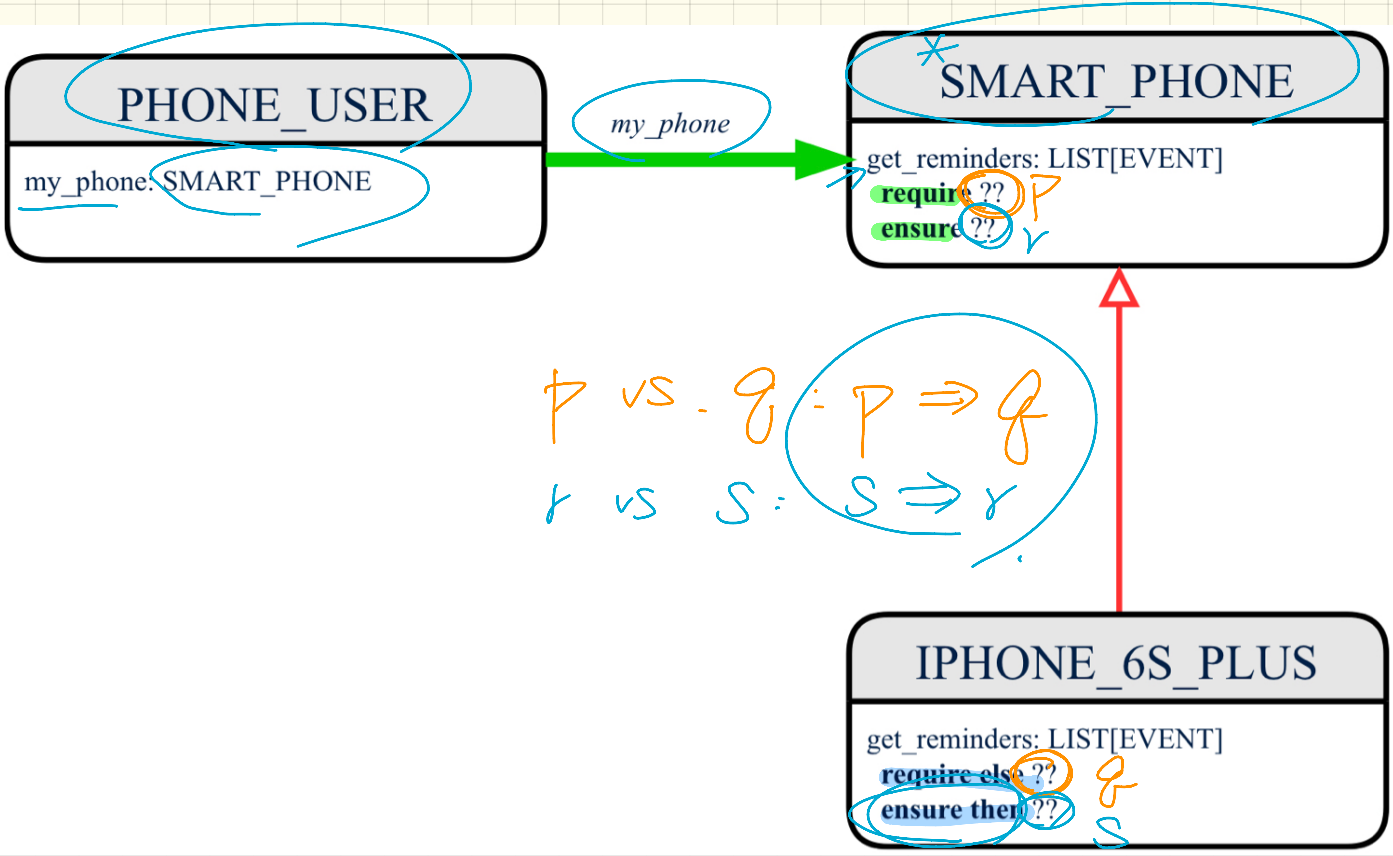
Q_1 : $Result = (i > 0) \vee (i \bmod 2 = 0)$

(1) $Q_2 \Rightarrow Q_1$

(2) $Q_1 \Rightarrow Q_2$



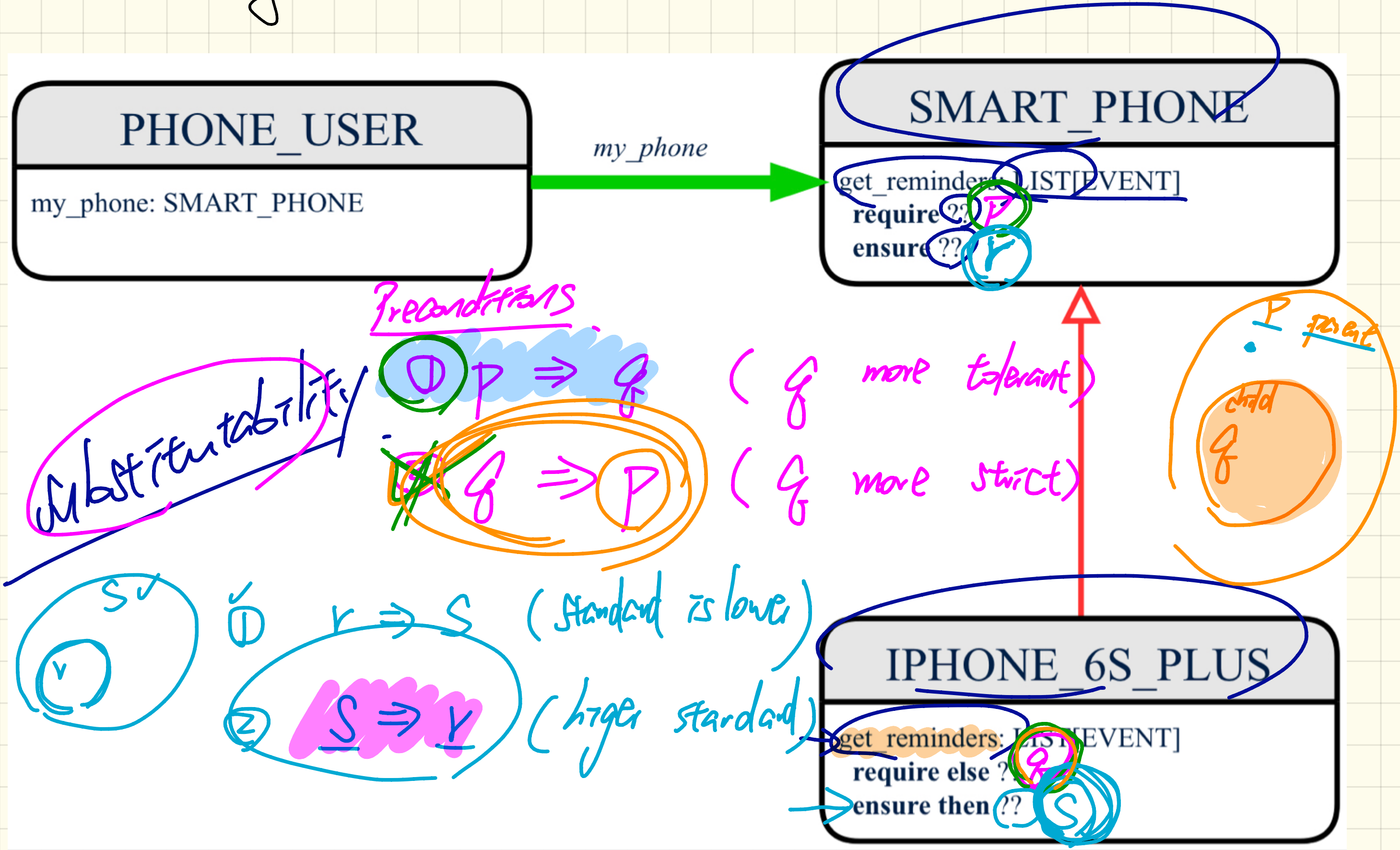
Subcontracting : Architectural View



Wednesday March 20

Lecture 19

Subcontracting: Architectural View



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
```

✓ 12%

0.1
0.11
0.12
0.13
0.14

0.15
0.16
0.17

$\gamma \Rightarrow \alpha$

$bl \geq 0.15 \Rightarrow bl \geq 0.1$ ✓

```
class IPHONE_6S_PLUS
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
```

✗

Fix: γ, 0.1

γ 0.09

not appropriate

① ● ⇒ ●

② ● ⇒ ●

Counter example: all events reminded happen tomorrow

get_reminders : LIST
  ensure
    $\forall e : Result \bullet \underbrace{e \text{ happens today}}_{p}$

appropriate:

$(p \Rightarrow q) \Rightarrow p$?

F T T F

get_reminders : LIST
  ensure
    $\forall e : Result \bullet (\underbrace{e \text{ happens today}}_{p} \Rightarrow \underbrace{e \text{ happens tomorrow}}_{q})$

$$P \Rightarrow P \vee q$$

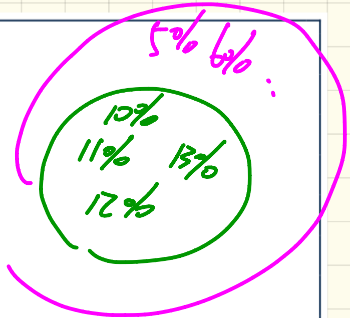
$$P \wedge q \Rightarrow P$$

$$P \wedge q \Rightarrow P \vee q$$

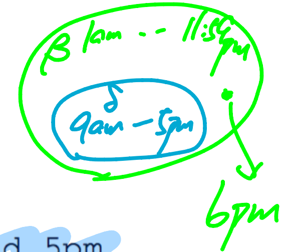
Subcontracting : Example (2)

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

$\alpha \Rightarrow \gamma$




```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.05 -- 5%
    ensure then
      δ: ∀e : Result | e happens today between 9am and 5pm
end
```



1.

PARENT: f require P (bl ≥ 10%) ensure r

CHILD: f require 4 (bl ≥ 15%) ensure S

as long as the inherited precond is satisfied it's ok, even if the child precond is stronger.

→ ① P ⇒ q → more

② S ⇒ r tolerant.

$\overline{S}$ higer standard

c.bl ≥ 10% √

c.bl ≥ 15%

bl ≥ 10% ⇒ bl ≥ 15% ✗

↳ not appropriate.

say c.bl is 16%

c.f ✓

say c.bl is 12%

c.f ← no precond violation

p: PARENT

c: CHILD

→ say p.bl is 9%

p.f precond. violation

→ say p.bl is 12%

p.f ✓

1.

PARENT

f

ensure

$P: x \geq 0$

CHILD

f

ensure

then.

$q: x > -2$

To be appropriate:

$$q \Rightarrow P$$

but $x > -2 \Rightarrow x \geq 0$ is not the case.

$\therefore$ not appropriate.

c: CHILD

$c.f$

-- say upon termination, x is -1

check: $x > -2 \wedge x \geq 0$.

Foo

g

① g require false

② g require true

BAR

g require else $x > 0$

f : Foo

b : BAR

$f.g$ × precond violation

-- $x \geq 5$

$b.g$

Monday March 25

Lecture 20

Contract Re-Declaration: Missing Pre-condition in Ancestor

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

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

x > 0

as if:

class FOO
f
require
True
do ..
end

~~require~~ false

Runtime:

x>0

true ∨ new_pre

||

true.

b: BAR

check b.x = -1 and

→ b.f

Contract Re-Declaration: Missing Post-condition in Ancestor

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

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

$y > 0$

as if:

f
  do ...
  ensure
    True
  end

$\text{True} \wedge \boxed{\text{new\_post}} \equiv \text{new\_post}$

F

b: BAR

b.f -- b.y = -1

postcond violation

Contract Re-Declaration: Missing Pre-condition in Descendant

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

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

as if: f
require else ~~?~~ False
do ...
end

At runtime:

ori_pre ∨ False

ori-pre.

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
```

~~orig-post ∧ ~~False~~ True~~

~~False~~.

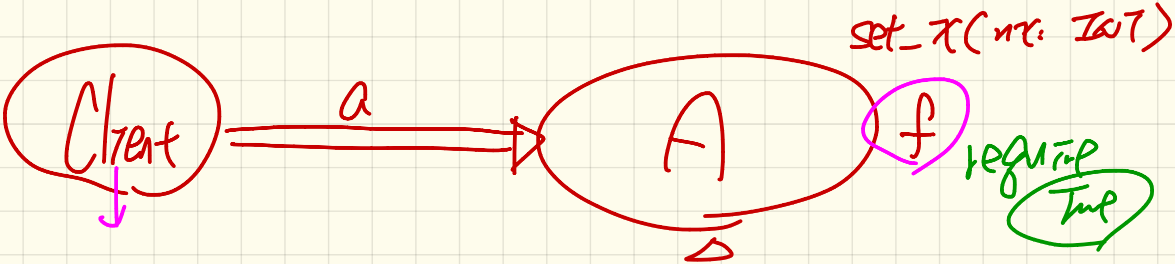
orig-post.

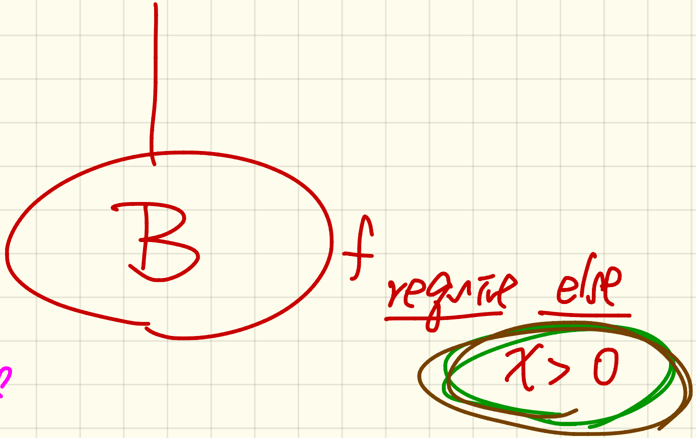
as f:

f do ...

ensure then ?? ~~False~~ True.

end





a: <u>A</u>

<u>create</u> {A} a

a.set_x(-2)

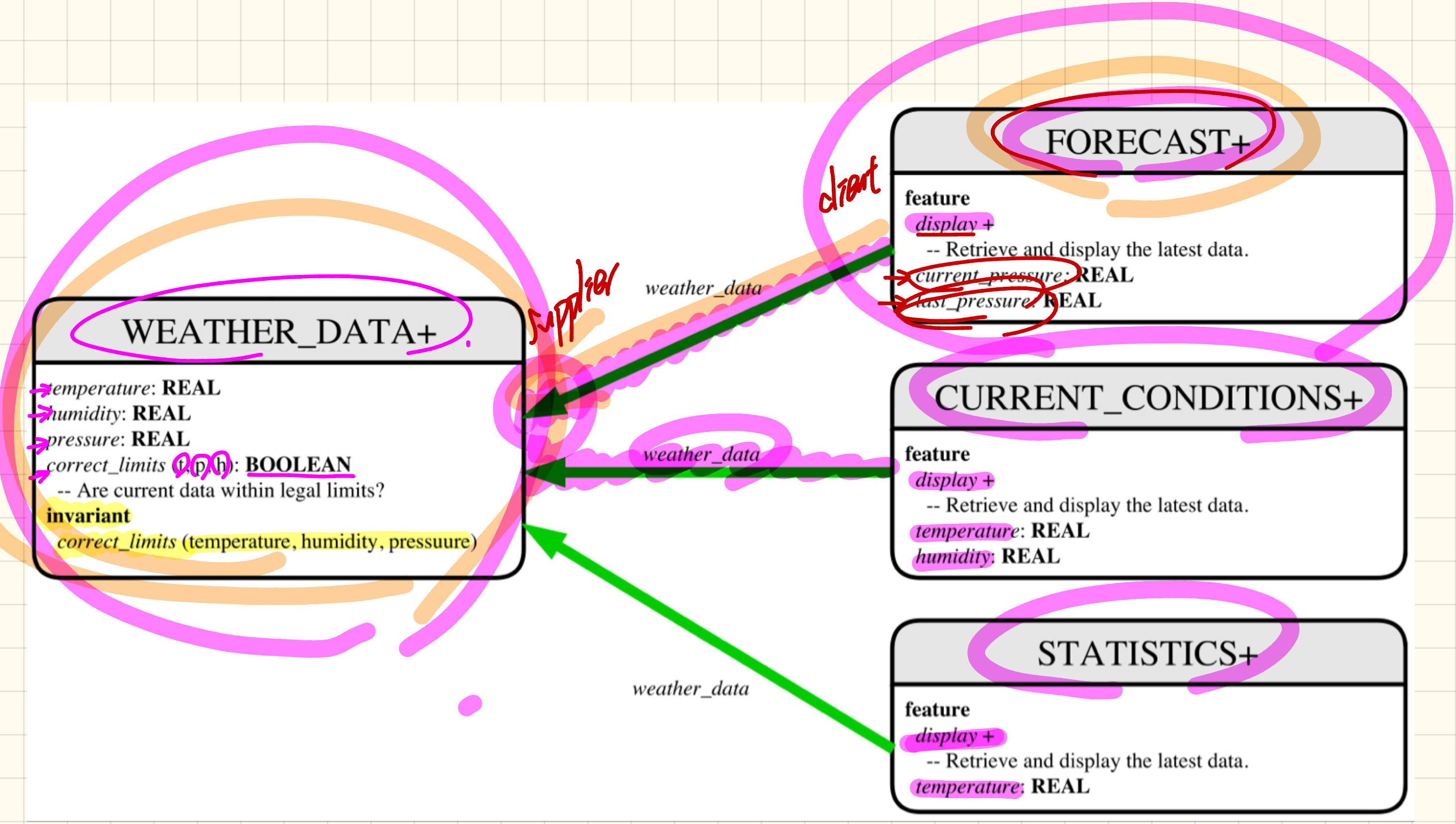
→ a.f ← no precondition violate

create {B} a

a.set_x(-2)

→ a.f ← True

Weather Station: 1st Design



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
```

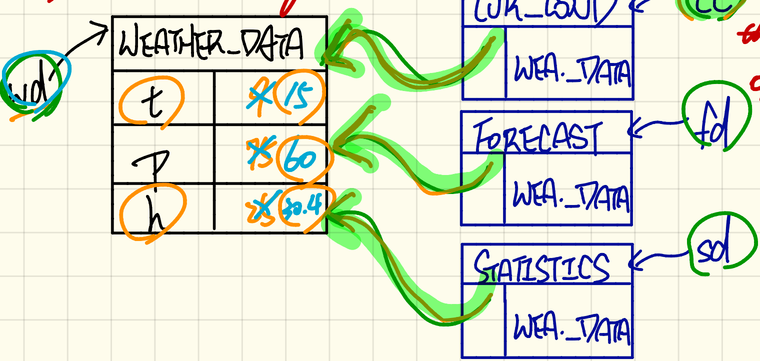
```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature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

no change on measur.
⇒ the 2nd updates are redundant
according to the frequency of display, rather than

the current design update



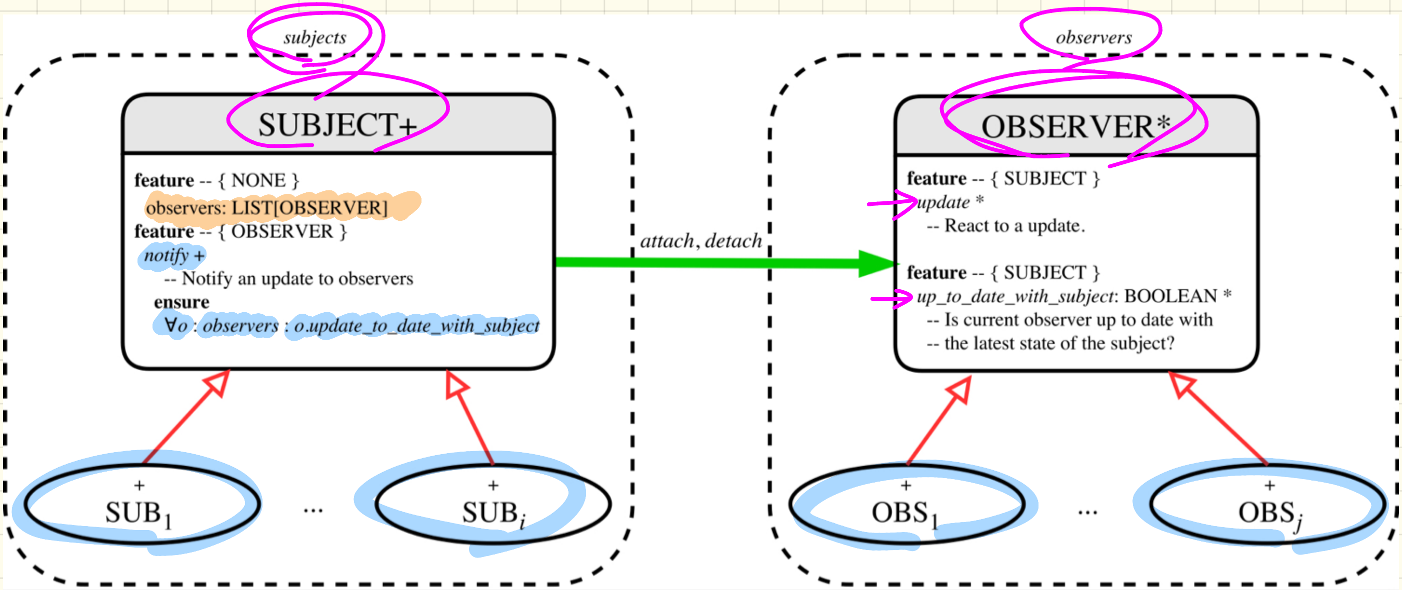
according to the frequency of data change.

```
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
```

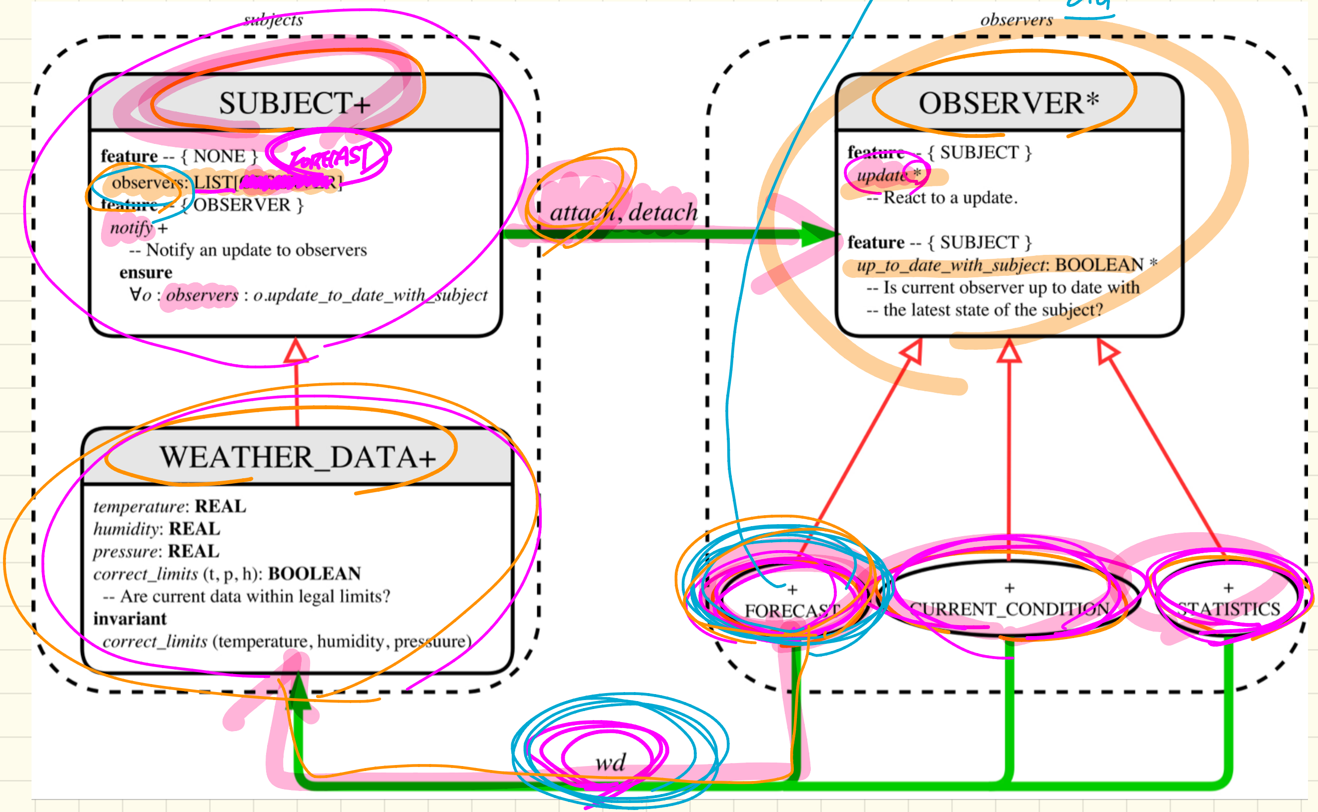
```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature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

The Observer Pattern



Weather Station: Applying the Observer Pattern

make (wd: WEATHER_DATA)
do
  weather_data := wd
  wd.attach (Current)
end



class SUBJECT

observers: LIST [OBSERVER]

notify

do

across observers as ob

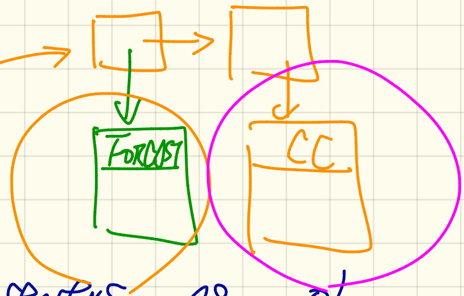
loop

ob.item.update

end

end

observes



ST: OBSERVER

dynamic binding.

loop

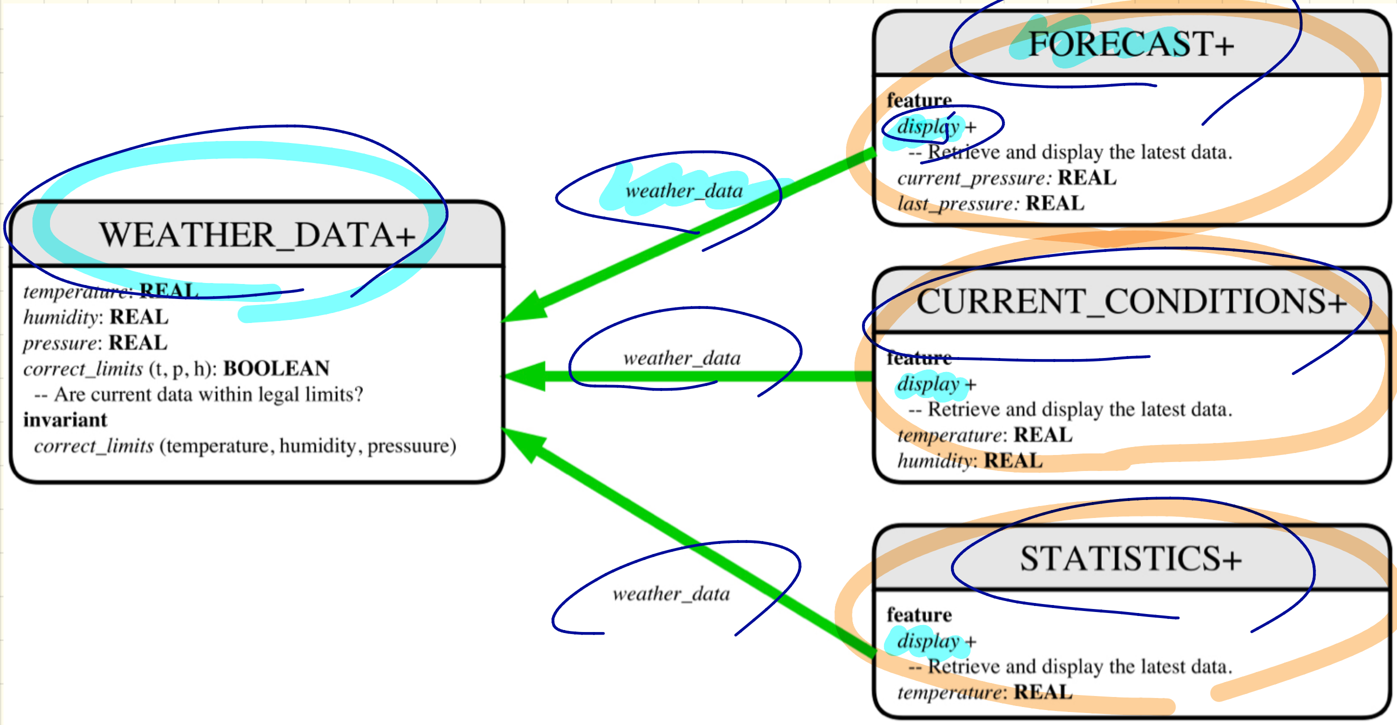
all

some

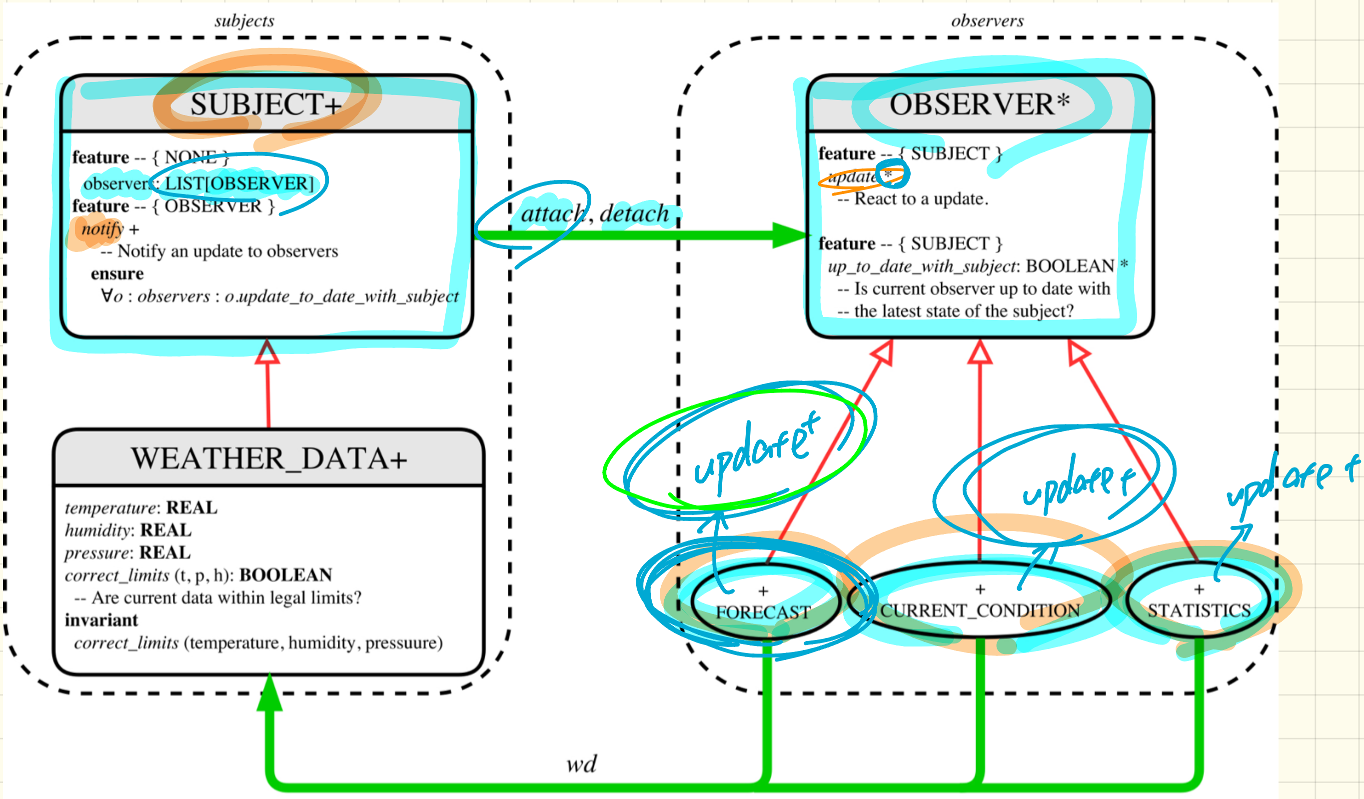
Wednesday March 27

Lecture 21

Weather Station : 1st Design



Weather Station: Applying the Observer Pattern



Implementing Weather Station: Subject

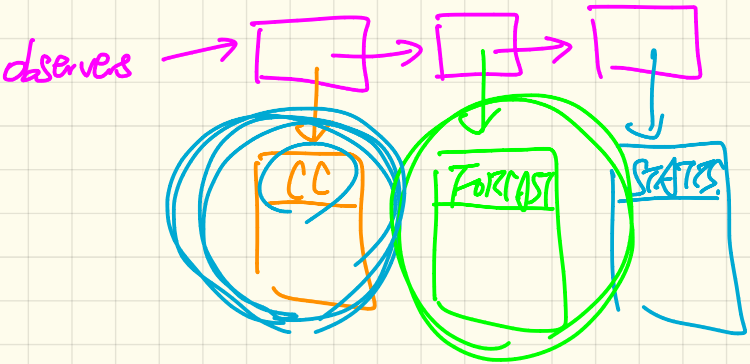
```
class SUBJECT create make
feature -- Attributes
  observers : LIST[OBSERVER]
feature -- Commands
  make
   do create {LINKED_LIST[OBSERVER]} observers.make
   ensure no_observers:  observers.count = 0 end
feature -- Invoked by an OBSERVER
  attach (o: OBSERVER) -- Add 'o' to the observers
    require 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
```

OBSERVER

↳ declared at OBSERVER

```
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
```

notify.



Implementing 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
```

wd.notify.

wd.attach(cc)

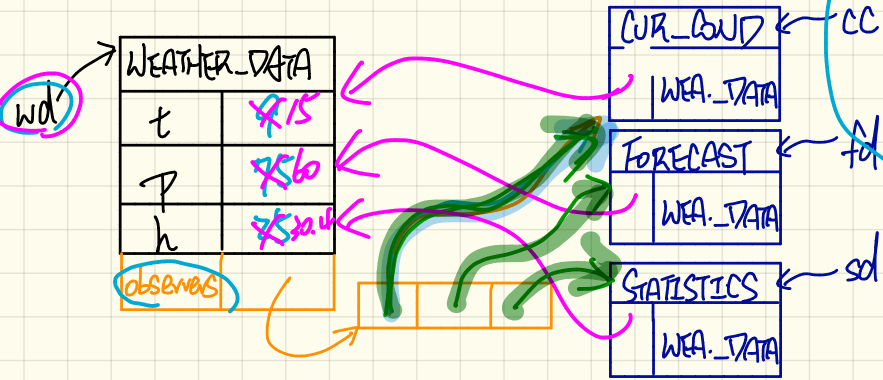
```
class FORECAST
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
      weather_data.attach (Current)
    ensure weather_data = a_weather_data
      weather_data.observers.has (Current)
    end
```

display update

```
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
```

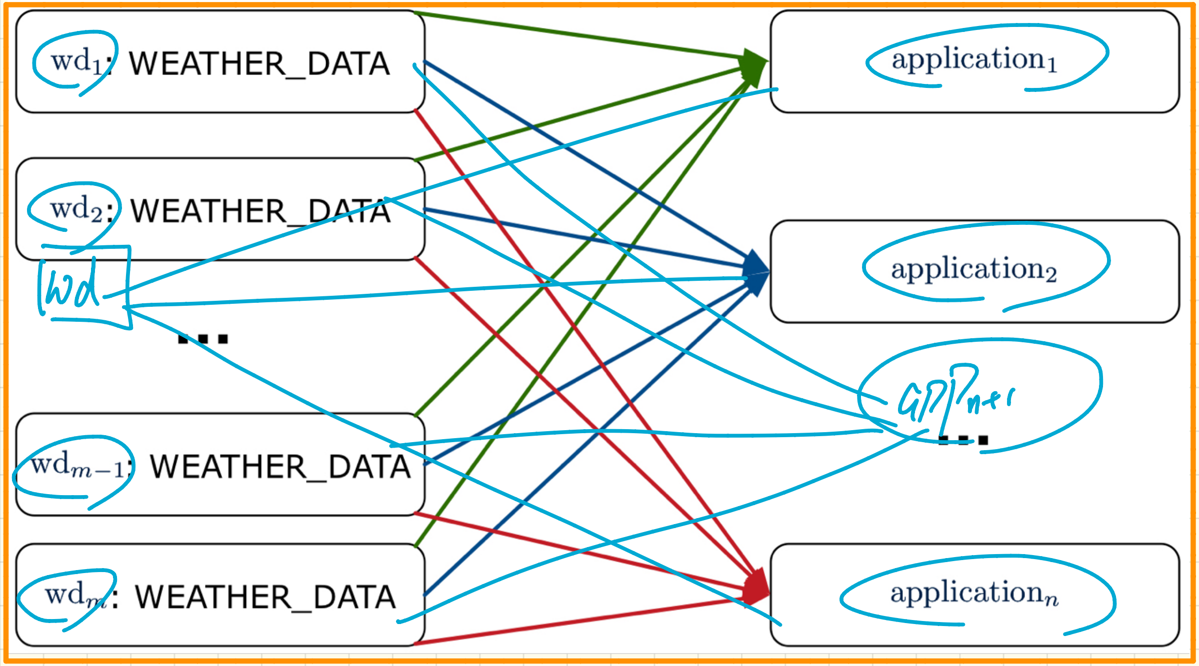
wd wd wd

```
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
```



a_weather_data.attach(Current)

Observer Pattern: Multiple Subjects and Observers



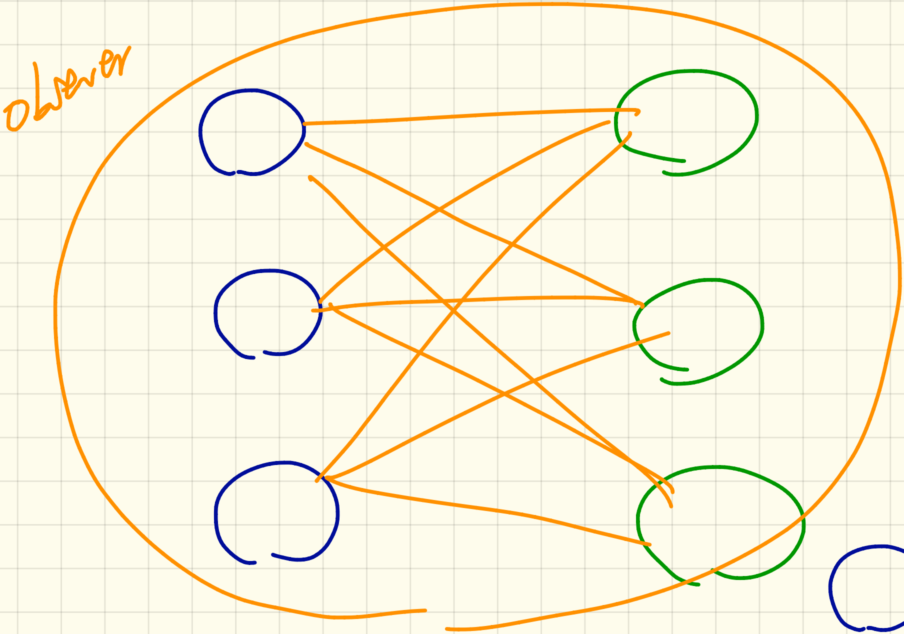
m

n

Complexity? $O(m*n)$

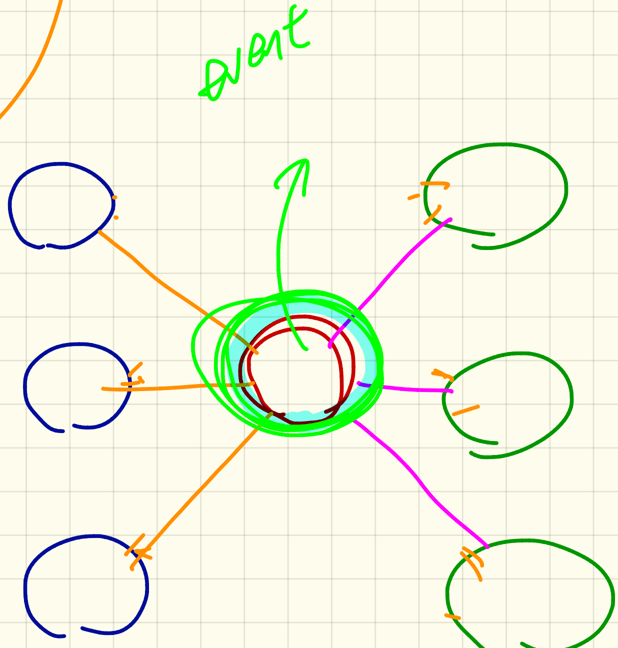
Adding a new subject? $O(n)$

Adding a new observer? $O(m)$

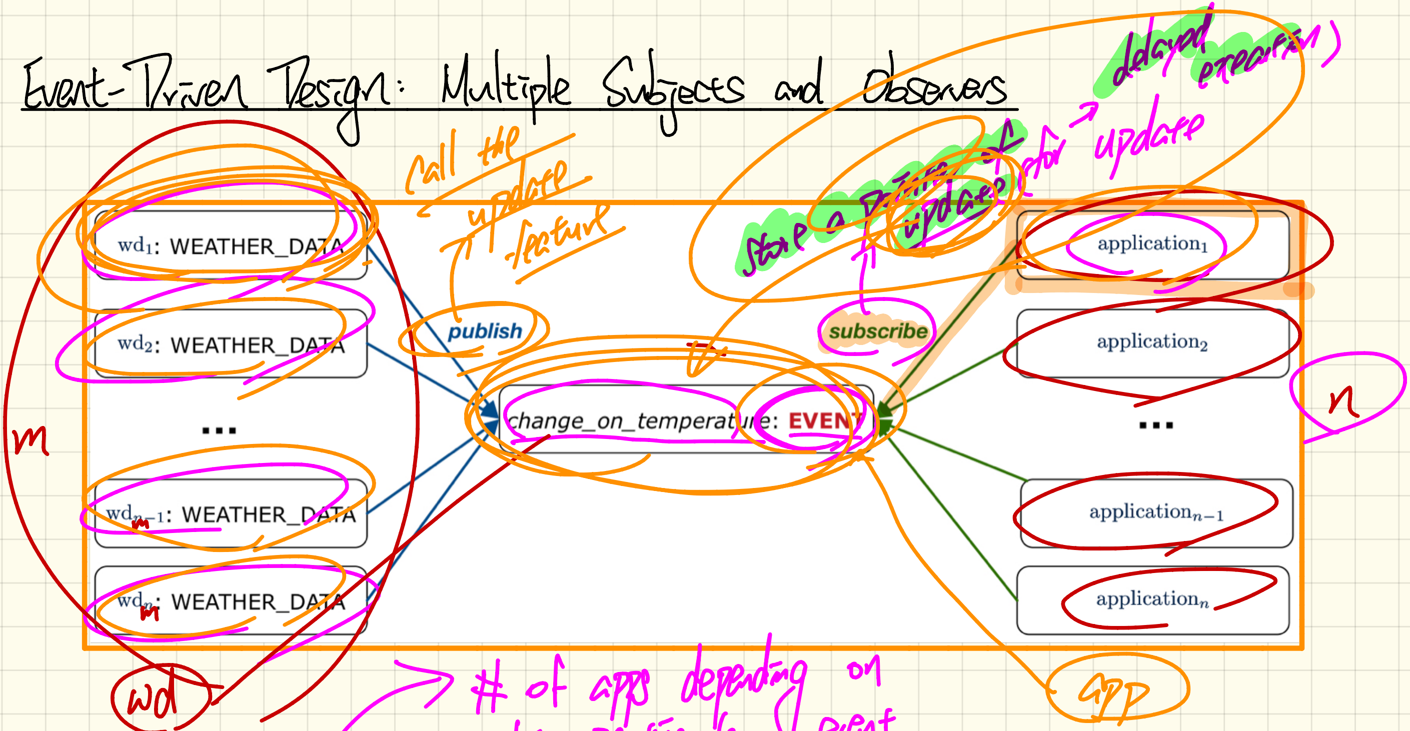


$$\frac{9}{\ } \quad \text{vs.} \quad \frac{6}{\ }$$

$$O(m \cdot n) \qquad O(m+n)$$



Event-Driven Design: Multiple Subjects and Observers



\# of apps depending on this particular event

$O(n)$ $O(n+m)$. $O(1)$

$O(1)$

Complexity?

$O(m+n)$

Adding a new subject?

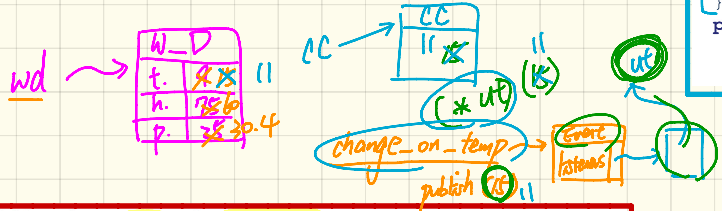
Adding a new observer?

Adding a new event type?

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 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) { }
    }
  }
}
```

```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);
  }
}
```

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
```

Command not of type PROCEDURE.

```
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
```

when you call the update on observers, it takes one input.

```
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
```

Monday April 1

Lecture 22

Wedn. Exam.

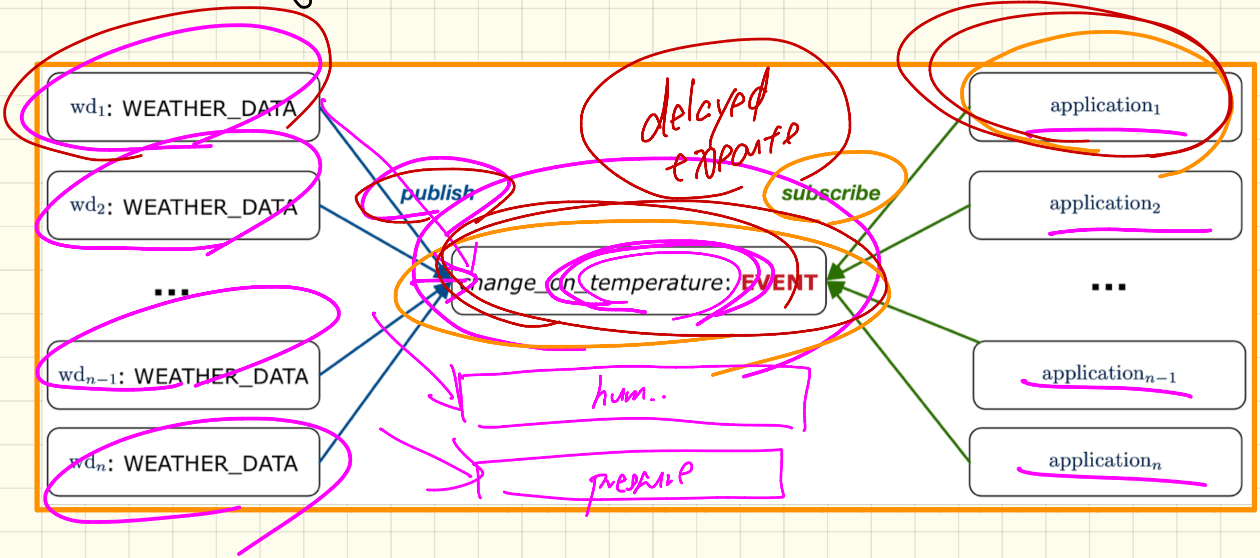
2pm

Fri. April 5 makeup class.

review

Fri April 12.

Event-Driven Design: Multiple Subjects and Observers



Complexity ?

Adding a new subject? Adding a new observer?

Adding a new event type ?

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);
  }
}
```

```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 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
```

WB pub sub obs E

```
class CURRENT_CONDITIONS
create make
feature -- Initialization
  make(wd: WEATHER_DATA)
    do
      wd.change_on_temperature.subscribe (agent update_temperature)
      wd.change_on_temperature.subscribe (agent update_humidity)
    end
feature
  temperature: REAL
  humidity: REAL
  update_temperature (t: REAL) do temperature := t end
  update_humidity (h: REAL) do humidity := h end
  display do ... end
end
```

humidity

τ: INT.

```
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
```

Arg.

TUPLE[REAL, INT]

[t, τ]

expect the stored update fun.

Target for executing the stored update.

```
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
```

update feature input

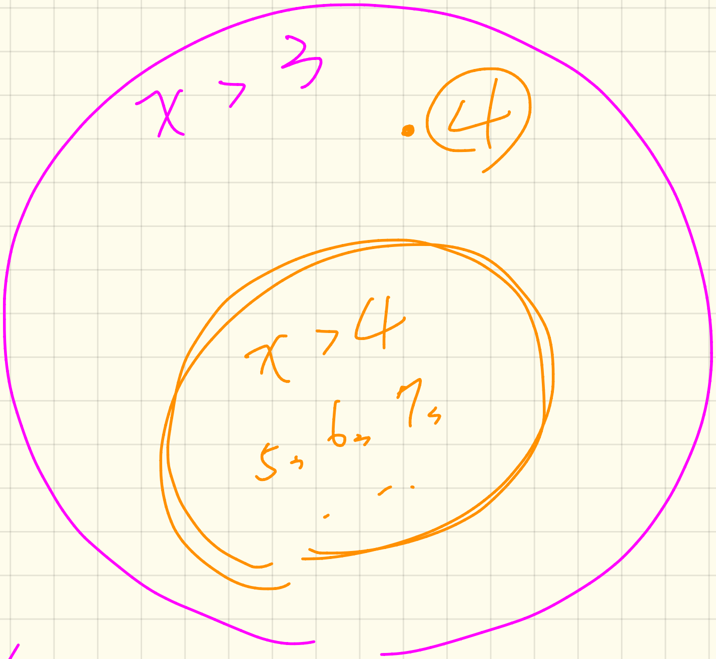
$x > 3$

$x > 4$

stronger

$x > 4 \Rightarrow x > 3$

weaker



Program Correctness : Example (1)

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

i = 14

13

too weak

f

require

??

do

imp

ensure

Q

end

Program Correctness : Example (2)

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

6

15

wp: $i > 4$

stronger: $i > 6$, $i > 7$ stronger ✓

weaker: $i > 4$ ✓ boundary

$i > 3$ ✗ Too weak

appropriate: satisfying precond and executing guarantees postcond.

weakest precondition (wp) to establish the postcond:

1. a precondition stronger than wp
2. a precondition weaker than wp problematic

Given.

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

Task: Prove that inc_by_9 is correct.

formulate → ?? predicate → tautology

Hoare Triple

Boolean predicate

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

precondition

Impleme.

postcondi.

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

tautologies

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

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

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

disprove:
counterexample:
$i = 4$

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

(a) starting with Q and executing S will terminate.

total correctness

(b) Assume (a), does the resulting state satisfy R.

partial correctness

actual precondition

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

$$\equiv \quad Q \Rightarrow wp(S, R)$$

weakest precondition

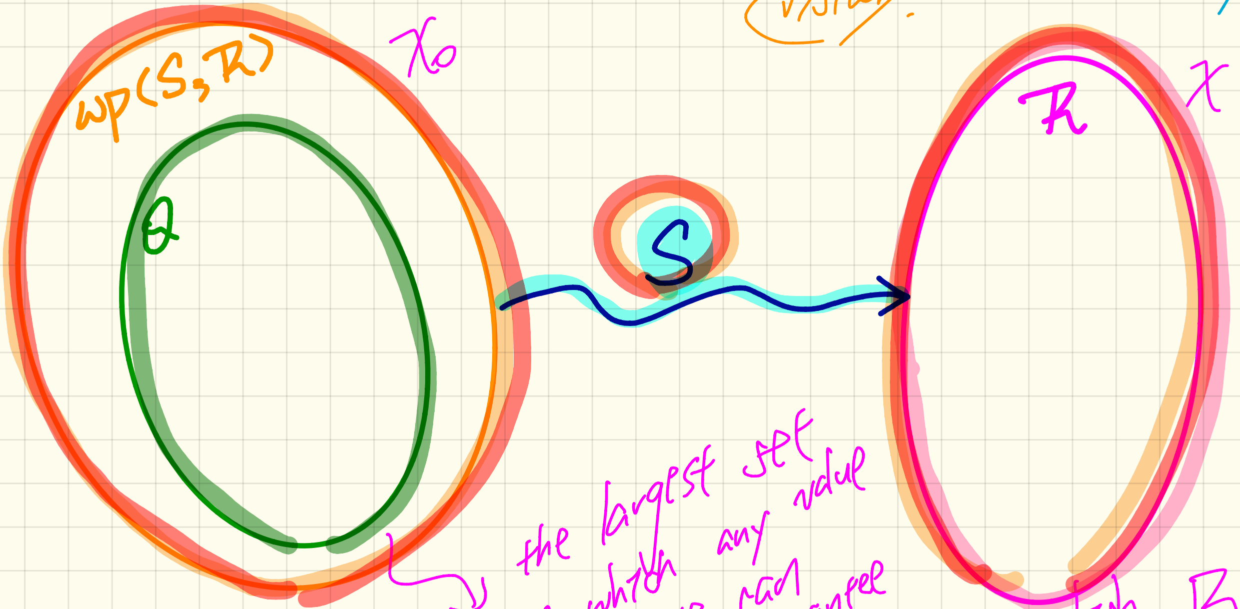
Hoare Triple as a Predicate

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

wp (S, R)

visitor

program structure (composite)



the largest set in which any value there can guarantee S to establish R.

$$wp(\underbrace{x}_{x} := \underbrace{x+9}_{e} , \underbrace{x > 13}_{R})$$

$$= \underbrace{x}_{x+9} > 13\,[x := \boxed{x+9}]$$

$$= x+9 > 13 \qquad x > 4.$$

Wednesday April 3

Lecture 23

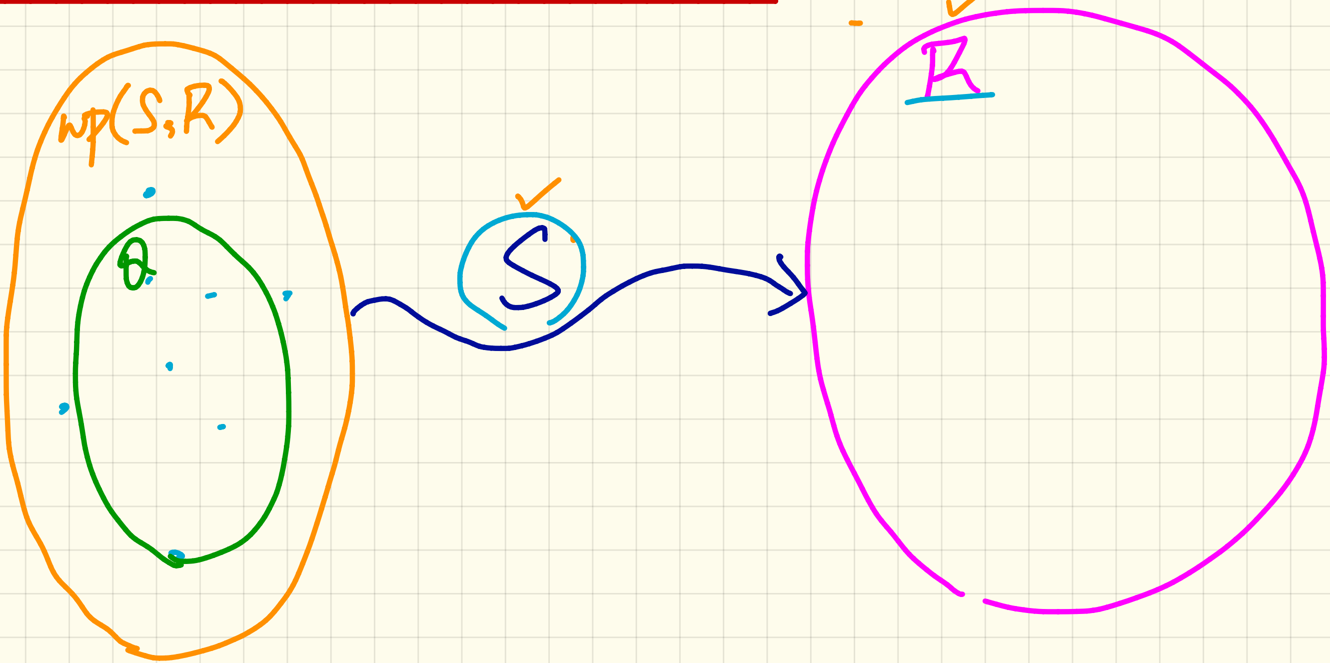
Makeup Lecture

Friday April 5 2pm ~ 4pm

LAS B

Hoare Triple as a Predicate

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



Program Correctness : Example (1)

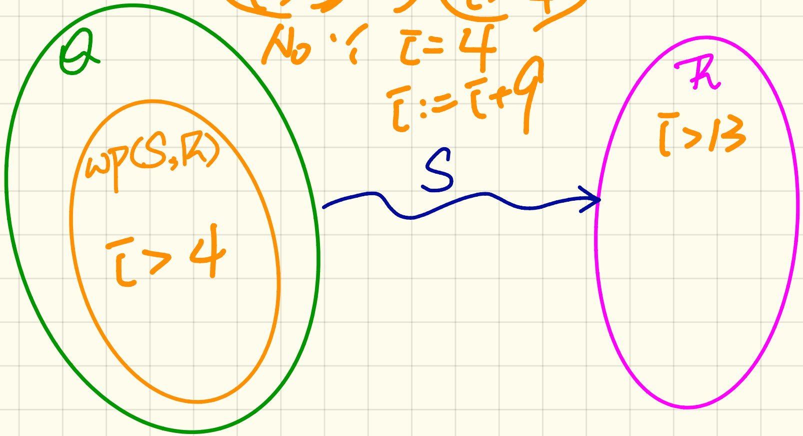
```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 3
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

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

$i > 3 \Rightarrow i > 4$

No ∵ $i = 4$

$i := i + 9$



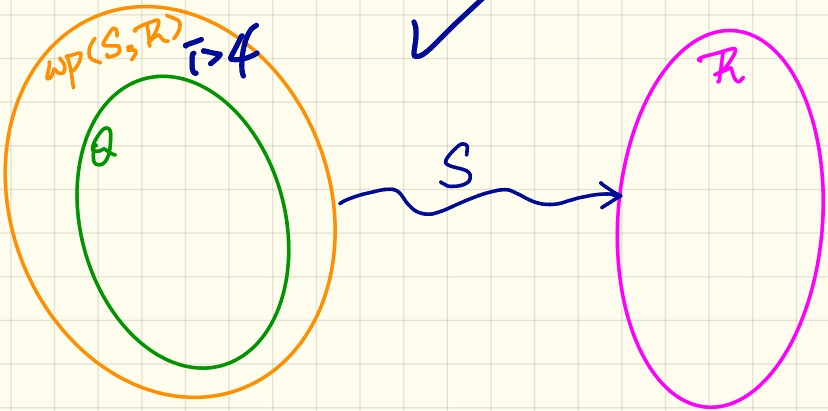
$\{i > 3\}\ i := i + 9\ \{i > 13\}$

Program Correctness : Example (2)

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

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

$i > 5 \Rightarrow i > 4$ ✓



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

$$wp(\underbrace{S}_{}\ ;\ \underbrace{R}_{})$$

① $:=$

② if then else

③ ____ ; ____

④ from .. until .. loop .. end

post-condition

predicate

1. pre-state
2. post-state

Precondition

pre-state

$x_0 > 4$

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

post-state

pre-state

x

$x > 13$

$\underline{x_0} + \underline{9}$

$x_0 > 4$.

$$\mathrm{wp}(x := \overset{\overset{-1}{2}}{x} + \overset{\overset{0}{3}}{1},\ x > x_0)$$

$= \{$ wp rule for assignment $\}$

$$\underline{x} > x_0\ [x := \underline{x_0 + 1}]$$

$= \{$ substitution $\}$

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

any x being incremented will become larger

$$wp(x := x+1,\ x < x_0)$$

$$= \{ \ \cdot \ \cdot \ - \ \}$$

$$\underline{x} < x_0 [x := \underline{x_0 + 1}]$$

$$x_0 + 1 < x_0$$

$$1 < 0 \quad \text{false}.$$

When wp is true,

any precondition Q would be correct

$\because Q \Rightarrow true \equiv true$

When wp is false

but useless

only precondition false is correct

$\because false \Rightarrow false \equiv true$

$$\{x \geq 22\} \; \boxed{x := x+1 \; \{x = 23\}}$$

$$wp(x := x+1,\ x = 23)$$

$$= \boxed{x = 22}$$

$$x \geq 22 \Rightarrow x = 22 \quad \times$$

$$x = 23$$

$P \Rightarrow Q$

$wp(\underline{\text{if}}\ B\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{end}},\ R)$

$B \Rightarrow wp(S_1, R)$

$\wedge\ \vee$ ※

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

Rule of wp: Conditionals

wp(if B then S_1 else S_2 end , R)

$B \Rightarrow wp(S_1, R)$
$\vee$
$\neg B \Rightarrow wp(S_2, R)$

vs.

$B \Rightarrow wp(S_1, R)$
$\wedge$
$\neg B \Rightarrow wp(S_2, R)$

??

$x = -1$
$y = -1$

$x + 1 \geq 0$

Consider:

B $\quad S_1 \quad S_2 \quad$ R

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

T

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

$\vee$ $\quad x \geq -1$ $\quad$ X

$y \leq 0 \Rightarrow wp(x := x - 1, x \geq 0)$ $\quad x \geq 1$

Counter example $y = -1$, $x = -1$

$x = -1$
$\downarrow x := x - 1$
$x = -2$

Correctness of Program: Conditionals

Is this program correct?

$\{x > 0 \wedge y > 0\}$ B

```
if x > y then
  bigger := x ; smaller := y
else
  bigger := y ; smaller := x
end
```

S1

S2

$\{bigger \geq smaller\}$

$x > 0 \wedge y > 0 \Rightarrow wp$

wp

$wp(\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ end}, \; bigger \geq smaller)$

$= \{$ wp rule for alternation $\}$

$$x > y \Rightarrow wp(S_1, \; bigger \geq smaller)$$
$$\wedge$$
$$x \leq y \Rightarrow wp(S_2, \; bigger \geq smaller)$$

$$\cdot \; S_1 \; \cdot \; S_2 \; \cdot \; R$$

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

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

Friday April 5

Lecture 24

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

$$Q \Rightarrow\ = \boxed{wp(S_1, \boxed{wp(S_2, R)})} \quad ①$$

— enough for S_2 to establish R

$$Q \quad S_1 \quad \_\_\_ \quad S_2 \quad R$$

$$\boxed{wp(S_2, R)}$$

Intermediate postcondition

② upon termination S_1

— can be established

Correctness of Program : Sequential Composition

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

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

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

① calculate $wp(\underbrace{tmp := x}_{S_1} \; ; \; \underbrace{x := y \; ; \; y := tmp}_{S_2}, \; x > y)$

= { wp rule for ; seq. comp. }

$wp(tmp := x, \; wp(\underbrace{x := y}_{S_1} \; ; \; \underbrace{y := tmp}_{S_2}, \; x > y))$

= { wp rule for ; }

✓

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

= { wp for := }

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

= { wp for := }

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

= { wp rule for := }

$y > x$

② True ⇒ $y > x$?

ex counter
$y = 1$
$x = 2$

$$\{y > x\} \;\_\_\_\_\_\_\_\_\; \{x > y\}$$

Swap without introducing an intermediate variable.

Loops : Eiffel vs. Java

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

exit condition

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

for (; ¬B ;){
Sbody
}

stay condition

from
until
not (x > 0) → x ≤ 0
loop
end

while (x > 0){
[. — —
}

Contracts of Loops

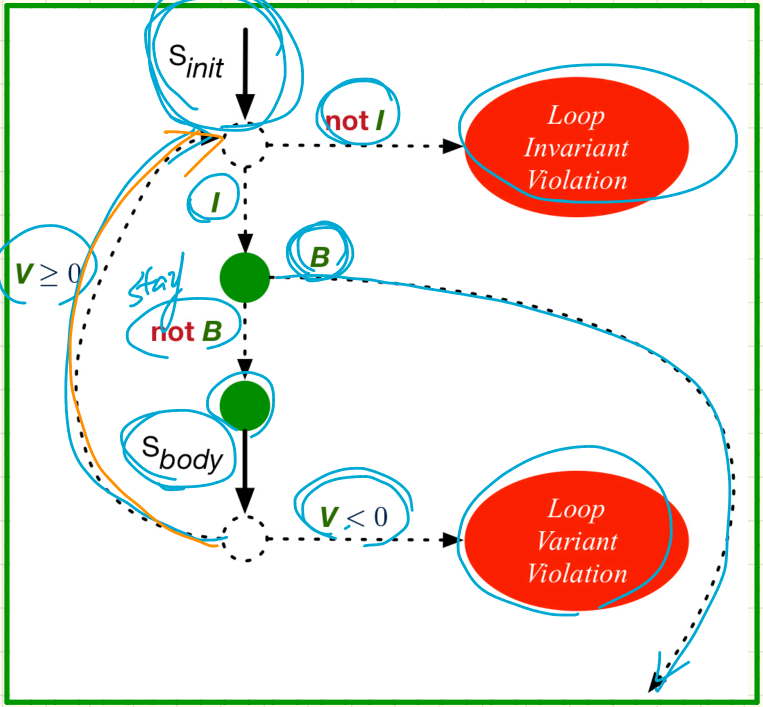
Syntax

```
from
  Sinit
invariant
  invariant_tag: I
until
  B
loop
  Sbody
variant
  variant_tag: V
end
```

established

maintained

Runtime Checks



Contracts of Loops: Example

Example

```
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
```

1 > 5 F

2 > 5 F

6 > 5 F T

6 - 2 = 4

6 - 6 = 0

i

Iteration 1 2

Iteration 2

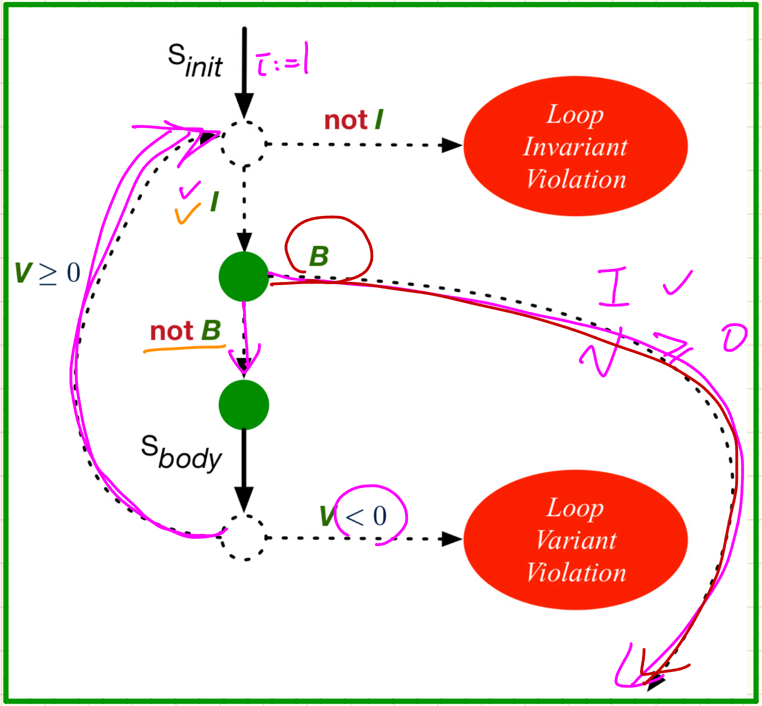
3

4

5

6

Runtime Checks



```
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
```

5

LI violation

∵ after 5th iteration

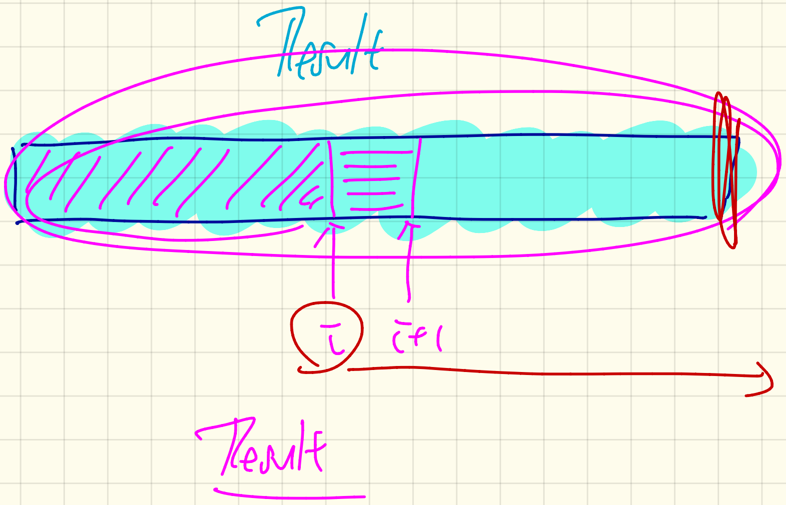
i becomes $\underline{\underline{6}}$

```
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
```

5th iteration

i becomes 6

$5 - 6 = -1$ LV violation.

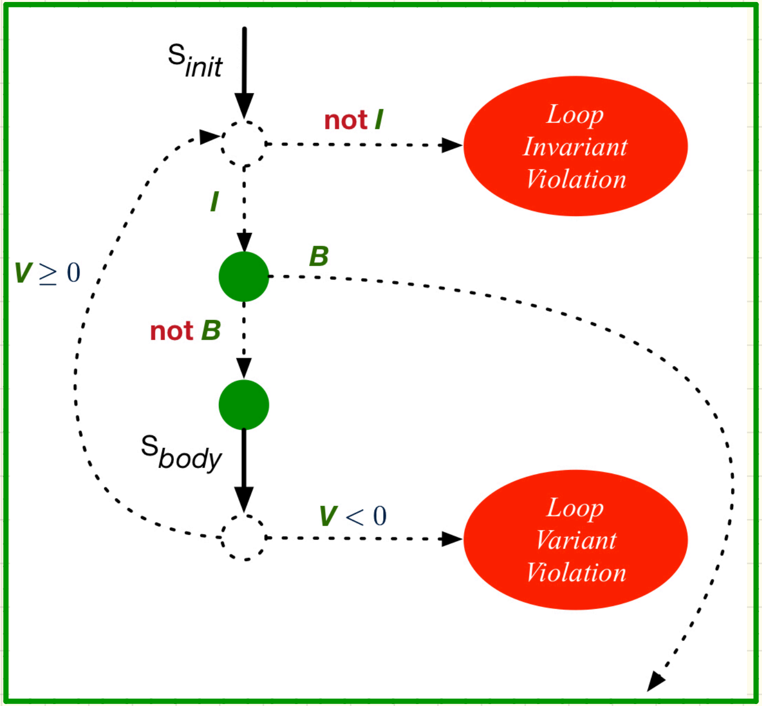

Result
i
i+1
Result

Contracts of Loops: Violations

Example

```
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
```

Runtime Checks

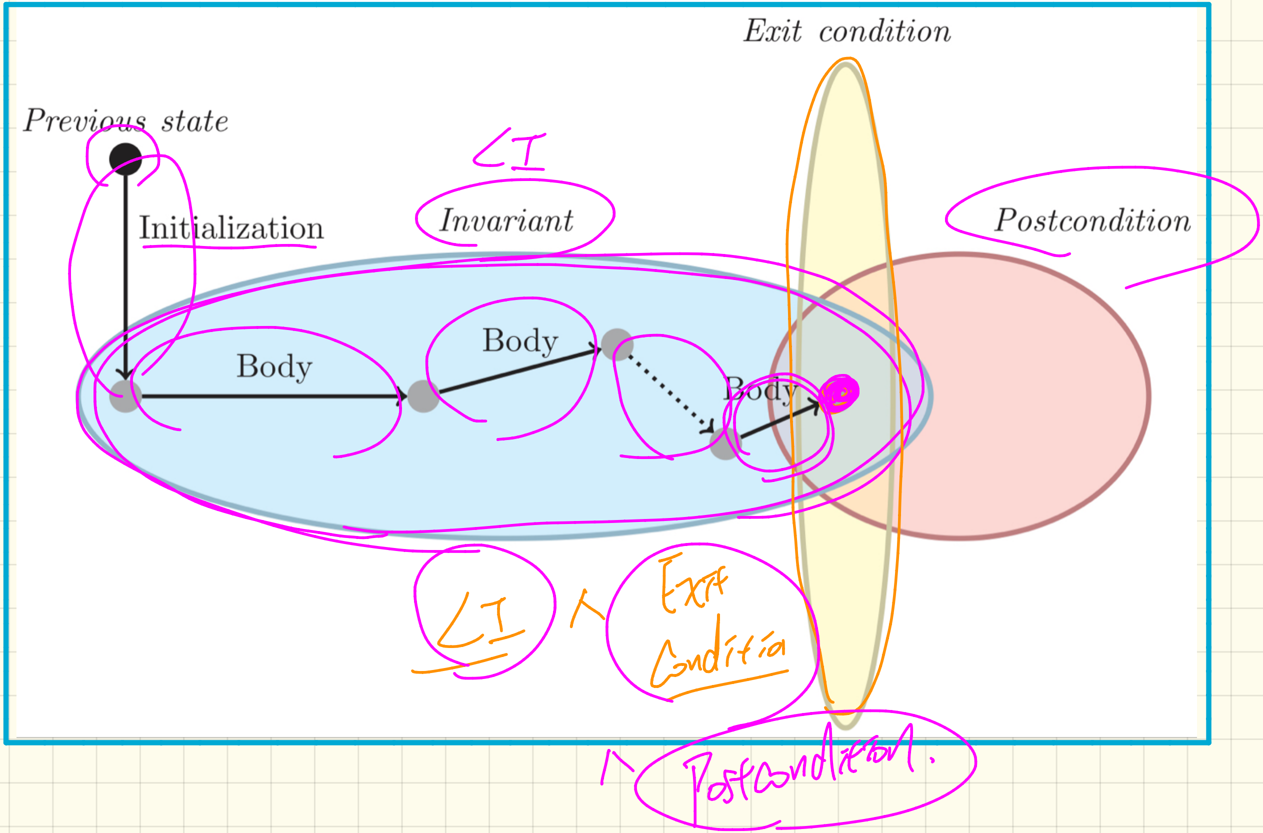


Invariant Violation : $1 \leq i \leq 5$

Variant Violation : $5 - i$

Skipping Loop Body : $i > 0$

Contracts of Loops: Visualization



Finding Max: V1

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
```

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

i only to be considered in the coming iteration

20 20

a.upper + 1

i	Result
1	20

$\forall j \mid 1 \leq j \leq 1 \bullet 20 \geq a[j]$

$20 \geq a[1]$

i	Result
2	20

$\forall j \mid 1 \leq j \leq 2 \bullet 20 \geq a[j]$

$20 \geq a[1]$

$20 \geq a[2]$

i	Result
3	20

$\forall j \mid 1 \leq j \leq 3 \bullet 20 \geq a[j]$

$20 \geq a[3]$ F

LI ✗

After Iteration	i	Result	LI	Exit ($i > a.upper$)?	LV
Initialization	1	20	✓	×	–
1st	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
2nd	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Finding Max: V2

```
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
```

i	Result
1	20

20	10	40	30

F

$\forall j \mid 1 \leq j \leq 0 \bullet$

$20 \geq a[j].$

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	●	●	●	●	●

$$\forall x \mid \underline{F} \cdot P(x) \qquad T$$

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

$$\equiv \forall x \cdot \underset{F}{\underline{R(x)}} \Rightarrow P(x)$$

Proof Obligations for Correct Loops

```
{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*.

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

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

    $B \wedge I$

    $\boxed{I \land B \Rightarrow R}$
- A loop ***terminates*** if:

  $V \geq 0$
  - Given ***LI*** *I*, and not yet to exit, S_{body} maintains ***LV*** *V* as non-negative.

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

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

Wednesday April 10

Review Lecture

expanded class UTIL

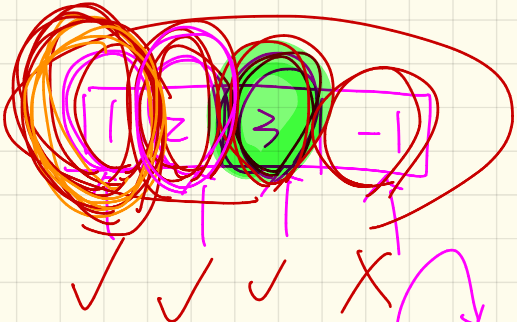
is_positive (i: INTEGER) : BOOLEAN
do
Result := i > 0
end

f

param

return type

counting (a: ARRAY[INT]; f: FUNCTION[[INT], BOOL]) : INT
do
across a as cursor loop
if f (cursor.item) then Result := Result + 1 end
end
end



agent u.is_positive(2)

agent u.is_positive(-2)

① ② ③

Result := u.counting (a, agent u.is_positive(?))

counting do
across a as cursor loop
if f(cursor.item) then ...
end

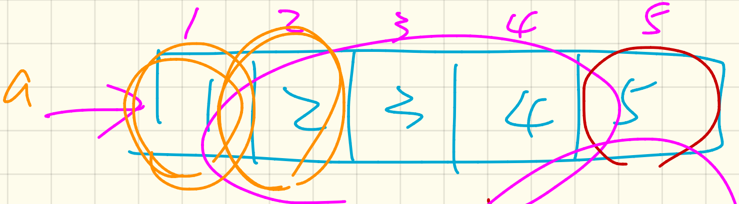
test: BOOL
local
a: ARRAY[INT]
u: UTIL
do
a := << 1, 2, 3, -1 >>
Result := u.counting(a, agent u.is_positive(?))
end

0?? ?? 4

expanded

agent & add

```
class UTIL
  add (i, j: INT): INT
    do
      Result := i + j
    end

  counting (a: A[INT]; f: FUNCTION[INT, INT, INT]): INT
    require
      a.count > 2
    local
      i: INT
    do
      from i := a.lower
      until i = a.upper - 1
      loop
        Result := Result + f(a[i], a[i+1])
        i := i + 1
      end
```



1 + 2

2 + 3

3 + 4

4 + 5

5 7 9

agent u.add (?, ?)

[cursor.item

(cursor + 1).item

test: BOOL

① pattern of loop

② segment of function

③ agent and open arguments

u1 u2

add

counting

counting

```
local
  a: A[INT]
  u1: UTIL
  u2
do
  a := <<1, 2, 3, 4, 5>>
  Result := u.counting(a, agent u.add(?, ?))
end
```

= 24.

FUNCTION[INT, INT, INT]

add(2, 3) → 5

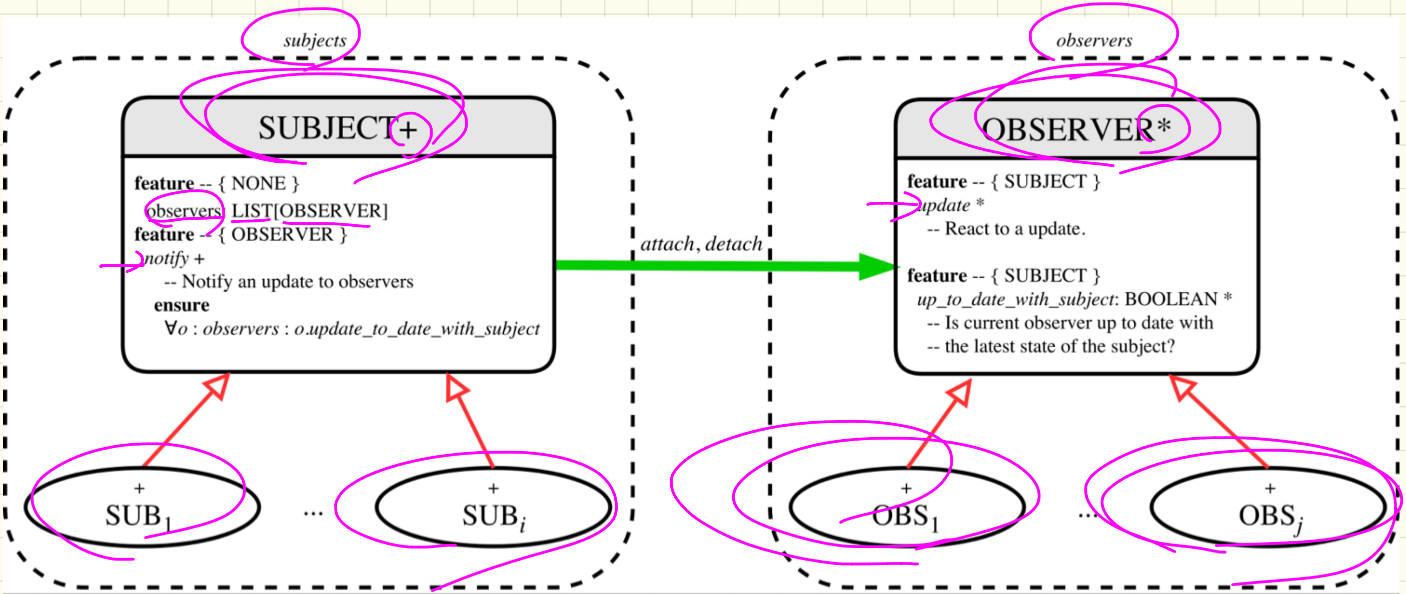
PROCEDURE[INT]

increment_by(3)

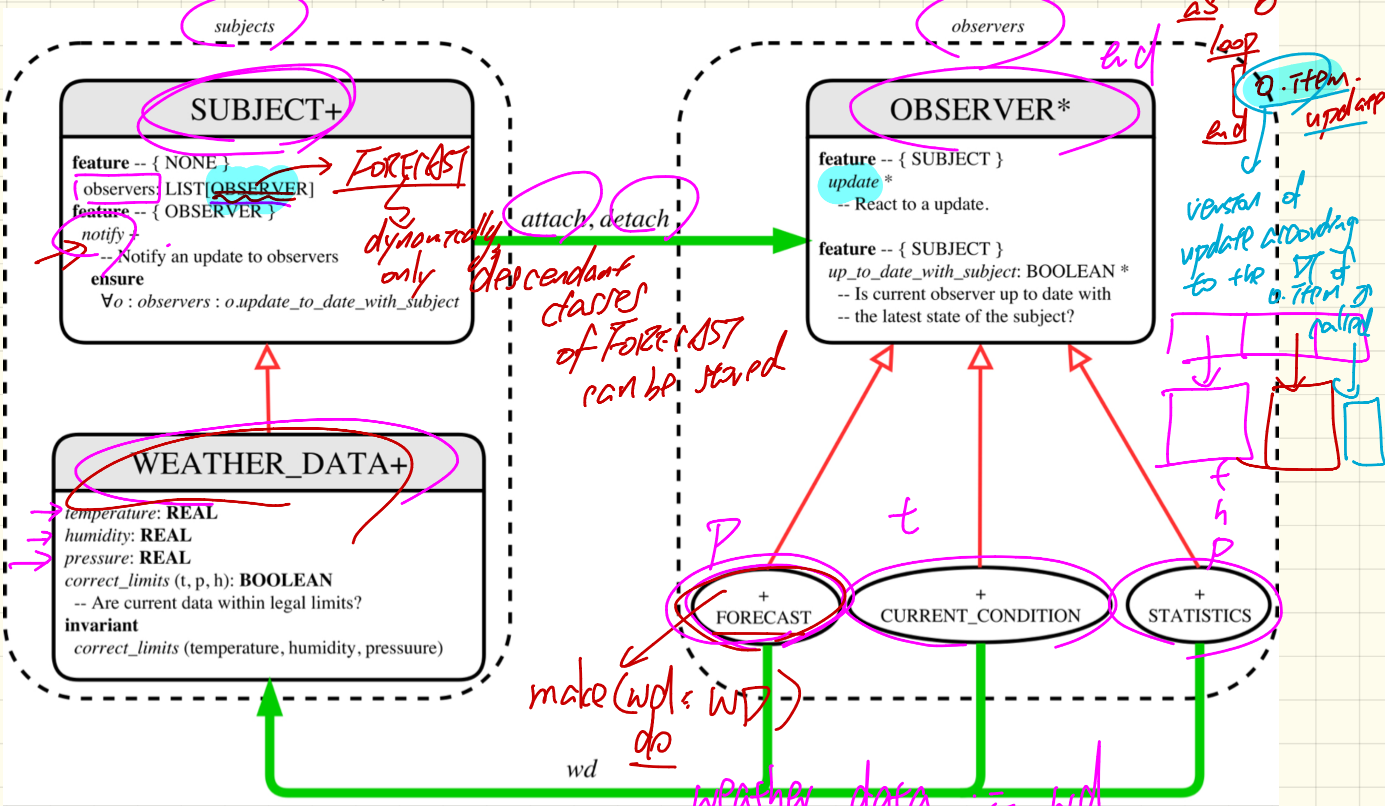
function returning boolean

PREDICATE[INT] → is_positive(3)

The Observer Pattern



Weather Station: Applying the Observer Pattern



notify
do
across observers as o
loop
o.item.update
end
end

weather_data := wd
wd.attach (Current)
end

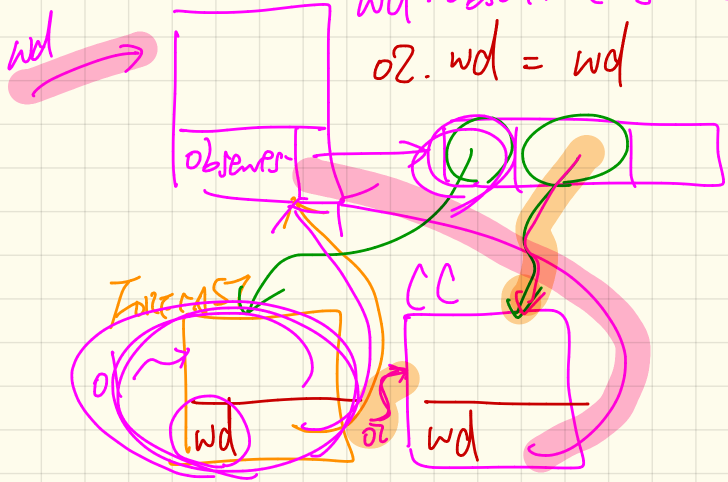
Implementing Weather Station : Subject

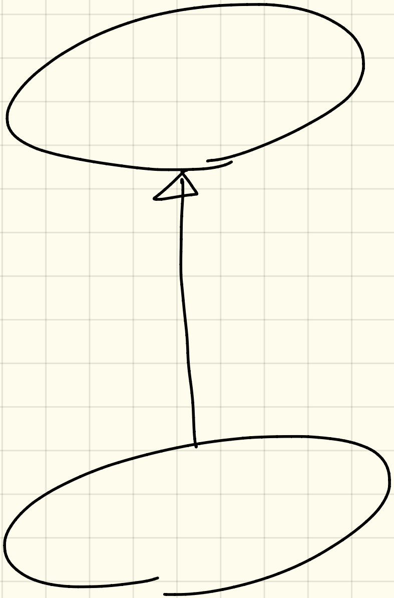
```
class SUBJECT create make
feature -- Attributes
  observers : LIST[OBSERVER]
feature -- Commands
  make
   do create {LINKED_LIST[OBSERVER]} observers.make
   ensure no_observers:  observers.count = 0 end
feature -- Invoked by an OBSERVER
  attach (o: OBSERVER) -- Add 'o' to the observers
    require 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
```

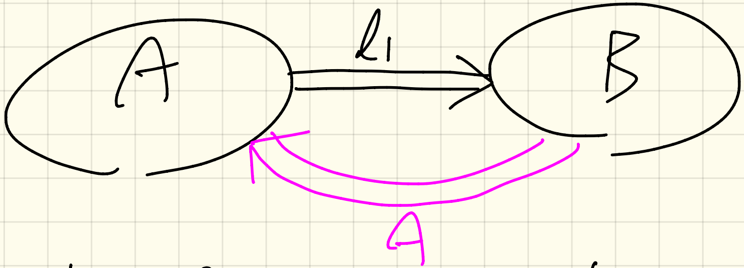
```
class WEATHER_DATA
inherit SUBJECT  rename make as make_subject end
create make
feature -- data available to observers
  temperature: REAL
  humidity: REAL
  pressure: REAL
  correct_limits(t,p,h: REAL): BOOLEAN
feature -- Initialization
  make (t, p, h: REAL)
    do
      make_subject -- initialize 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
```

wd.observers[2] = o2

o2.wd = wd





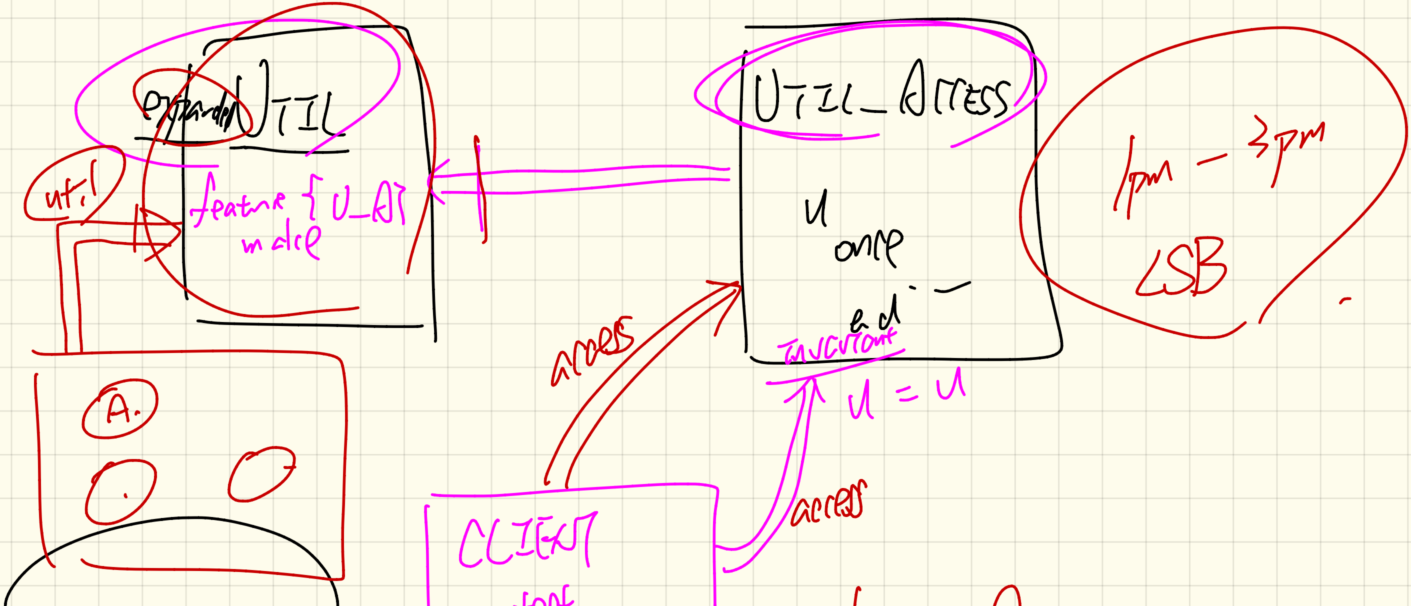


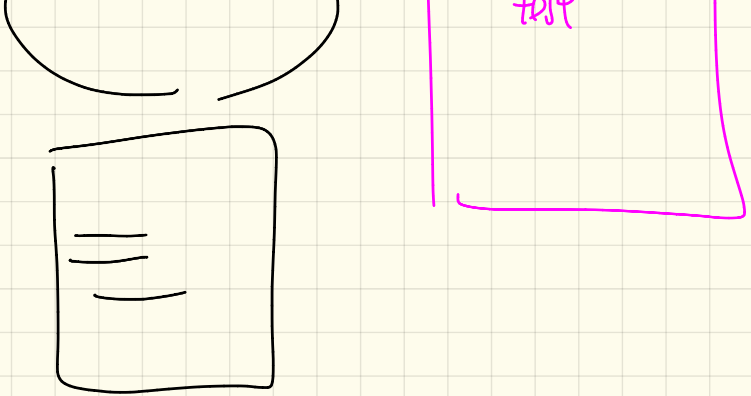
class A

$l1 : B$

class B

$l2 : A$





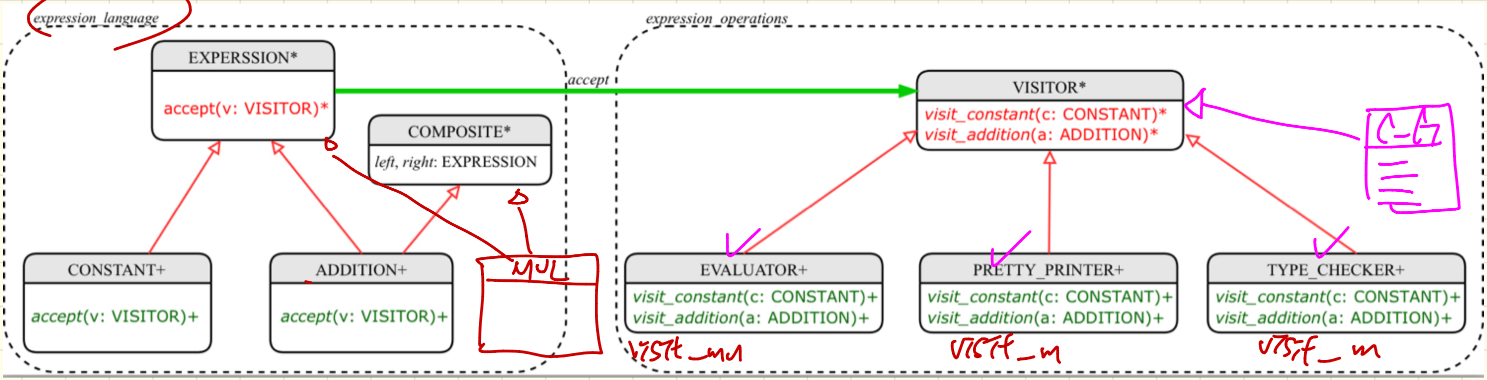
class A

util: UTIL

make
local
access: U_A
do
util := access.u
end

end

Visitor Design Pattern: Architecture



OCP → closed / open

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
```

change 1: add a new structural component MULTIPLICATION

change 2: add a new operation

SCP.

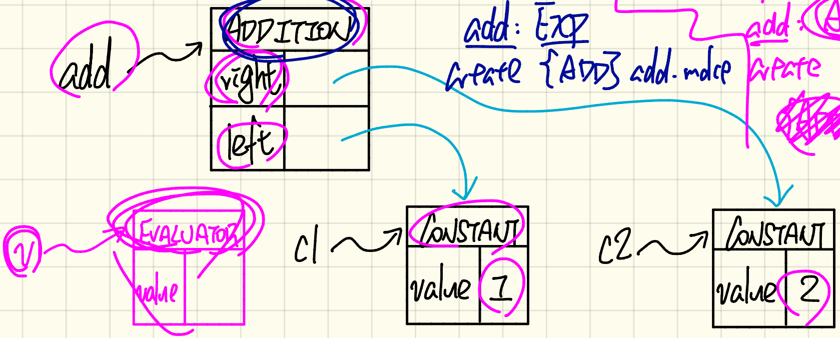
Executing Composite and Visitor Patterns at Runtime (double dispatch)

Tracing add.accept(v)

Opt 1
add: EXP
create {ADD} add.make

Opt 2
add: ADDITION
create add.make

1st Dispatch:
DT of add is ADDITION;
call the accept in



2nd Dispatch
DT of v is EVALUATOR,
call the visit_add

```
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
```

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

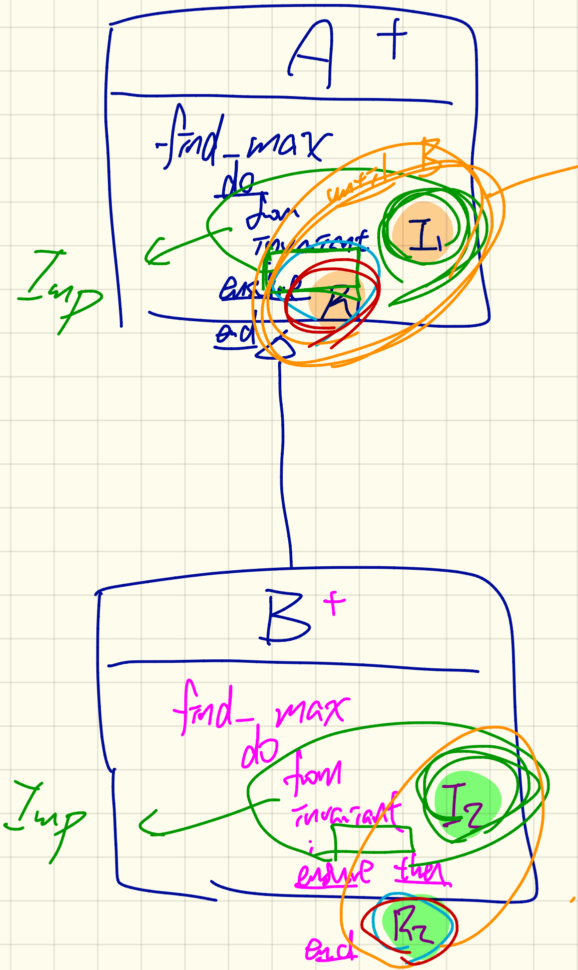
in
Evaluator

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

in

Thursday April 11

Review Lecture



I_1 $\quad$ I_2

$I_1 \wedge B \Rightarrow R_1$

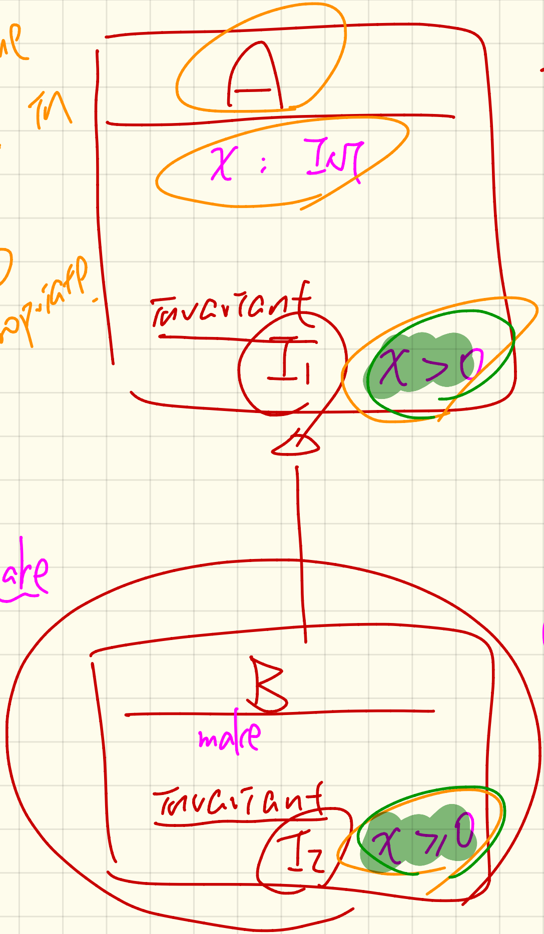
runtime check

$\boxed{R_1 \wedge R_2}$

judge design correctness

$\boxed{R_2 \Rightarrow R_1}$

Justify whether or not the class inv. in A and B are appropriate.



$b : B$

create b.make

runtime $I_1 \wedge I_2$.

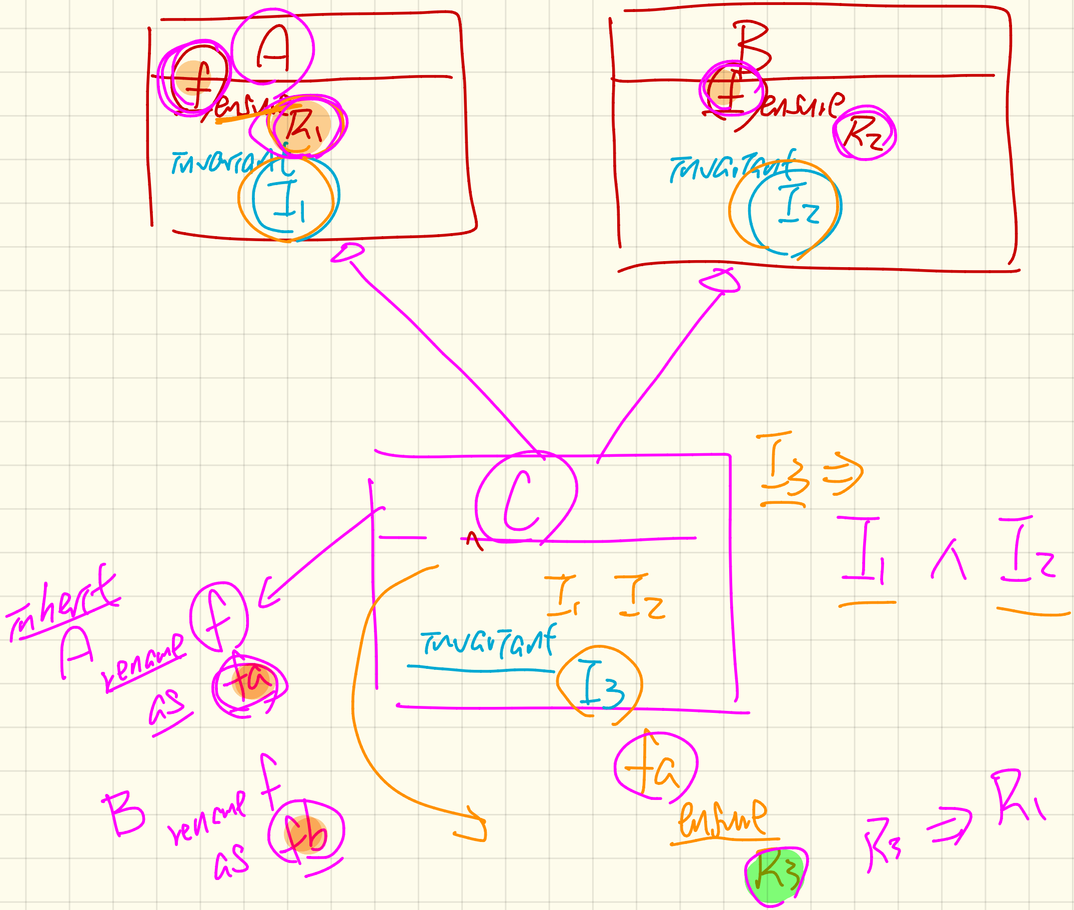
prove $I_2 \Rightarrow I_1$.

$\boxed{\text{At runtime}}$

↳ $x > 0 \wedge x \geq 0 = x > 0$

Prove: $\underset{T}{x \geq 0} \Rightarrow \underset{F}{x > 0}$

Counter example: $x = 0$.



A
f
ensure
R1
invariant
I1
B
f
ensure
R2
invariant
I2
C
I1 I2
invariant
I3
inherit
A
rename
f
as
fa
B
rename
f
as
fb
fa
ensure
R3
I3 ⇒
I1 ∧ I2
R3 ⇒ R1

a : A create {B} a.make (0) x > 0

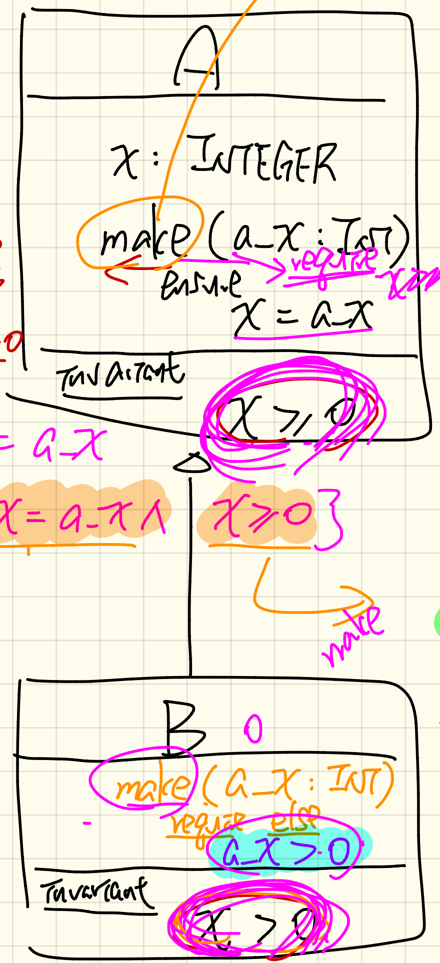
require $a\_x \geq 0$

b : B

create b.make (0) → CI violation

-- $x \geq 0 \wedge \underline{x > 0} \equiv x > 0$

require ?? $a\_x \geq 0$



{True} $x := a\_x$ $\{x = a\_x \wedge x \geq 0\}$

make

$x \geq 0$

Compiler/runtime assertion monitor

check $\boxed{x \geq 0 \wedge x > 0}$

True $\Rightarrow$ wp($x := a\_x$, $x = a\_x \wedge x \geq 0$)

$a\_x = a\_x \wedge a\_x \geq 0$

$\equiv$ $a\_x \geq 0$

True $\Rightarrow a\_x \geq 0$

×

Counter example: $a\_x = -1$

Runtime

$a\_x \geq 0$

$\vee$

$a\_x > 0$

$\equiv$

$a\_x \geq 0$

$$wp(x := 23,\ x > 22)$$

$= \{$ wp rule for assignment

programming assignment

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

substitution of free occ. of x.

```
max_of(x, y: INT): INT
  require
    x ≠ y
  do
    if x > y then
      Result := x
    else
      Result := y
    end
  ensure
    Result ≥ x ∧ Result ≥ y
```

max_of(4, 2)

S1 S2

$Result = x \vee Result = y$

Q: Prove or disprove max_of is correct.

1. Formulate program:

$\{x \neq y\}$ if $x > y$ then S_1 else S_2 $\{R \geq x \wedge R \geq y\}$

2. Calculate wp:

wp(if $x>y$ then $R := x$ (x+1) else $R := y$, $R \geq x \wedge R \geq y$)

= { wp rule for alternation }

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

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

= { wp for assignment twice }

$x > y \Rightarrow x \geq x \wedge x \geq y$ (x+1, x+1)

$$x > y \Rightarrow \underline{x \geq x} \wedge x \geq y$$

$$\wedge$$

$$x \leq y \Rightarrow y \geq x \wedge \underline{y \geq y}$$

$$= \{ x \geq x \equiv T,\ y \geq y \equiv T,\ T \wedge P \equiv P \}$$

$$\underline{x \overset{4}{>} \overset{3}{y} \Rightarrow x \overset{4}{\geq} \overset{3}{y}\ \ T}$$

$$\wedge$$

$$x \leq y \Rightarrow y \geq x\ ]\ T$$

$$= \{ \text{arithmetic},\ T \wedge T \equiv T \}$$

$$T \rightsquigarrow wp(prog, R)$$

$$\underline{x > y} \Rightarrow \boxed{\overset{T}{\underline{x+1 \geq x}}} \wedge \underline{x+1 \geq y}$$

$$T$$

3. Prove:

$$x \neq y \Rightarrow \overset{F}{T} \equiv \overset{F}{T}$$

$\therefore$ Program is correct.

a and then b

&& a and b

$p \wedge q \wedge r$

$\equiv p \wedge r \wedge q$

✓

$x \neq 0$ and then $y/x > 2$

$\wedge$ $\wedge$

$\vee 1$ a.lower $\leq$ i ~~and then~~ i $\leq$ a.upper and ~~then~~ a[i] > 3

$\wedge$ $\wedge$

$\vee 2$ i $\leq$ a.upper and ~~then~~ a[i] > 3 and ~~then~~ a.lower $\leq$ i

0

T

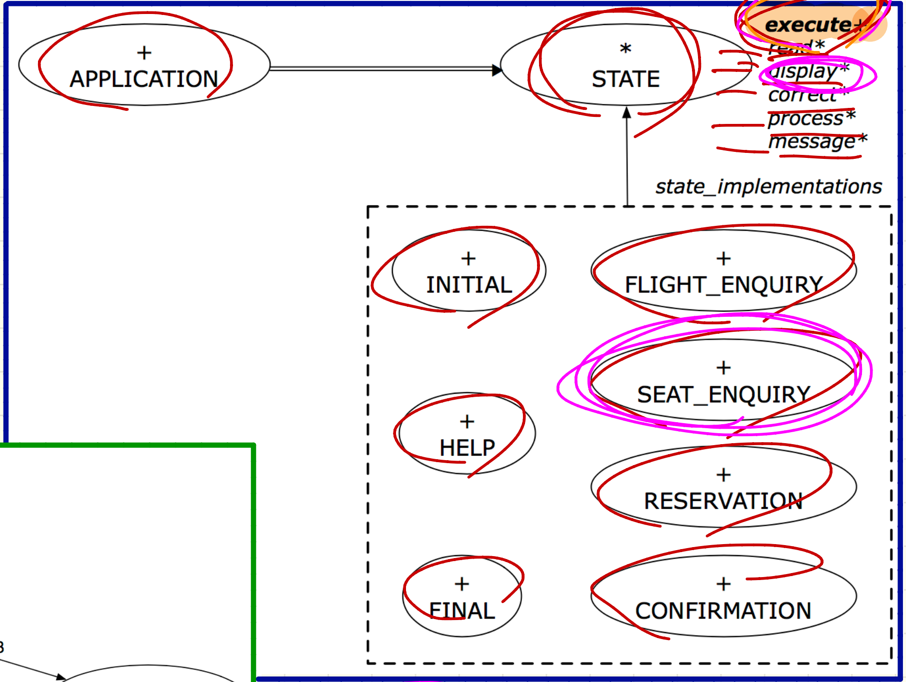
b1 <u>and</u> <u>then</u> b2 → evaluated only if b1 is T

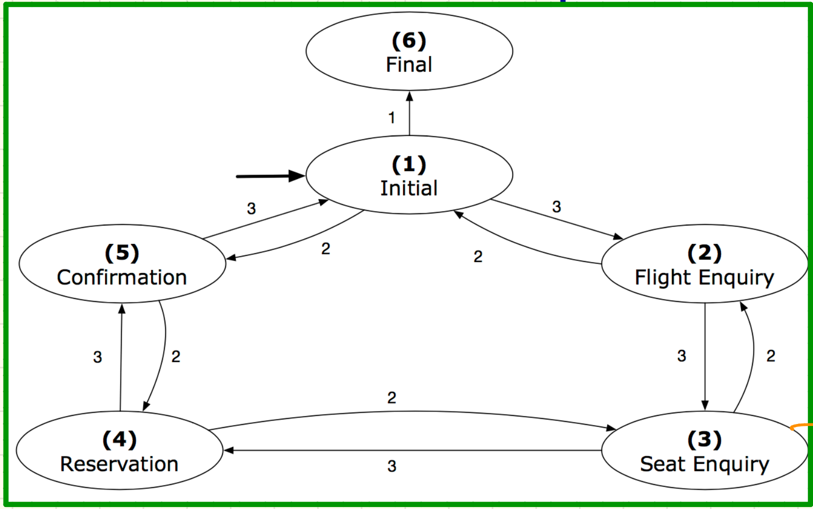
b1 <u>or</u> <u>else</u> b2 → evaluated only if <u>b1</u> is F

$F \vee P \equiv P$

State Pattern: Architecture

execute
do
display
end






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

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
```

class WEATHER_DATA

fd

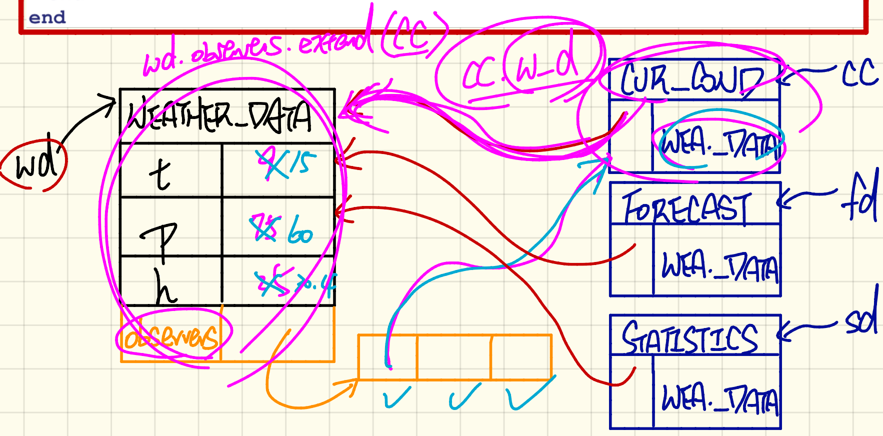
up to date

```
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
```

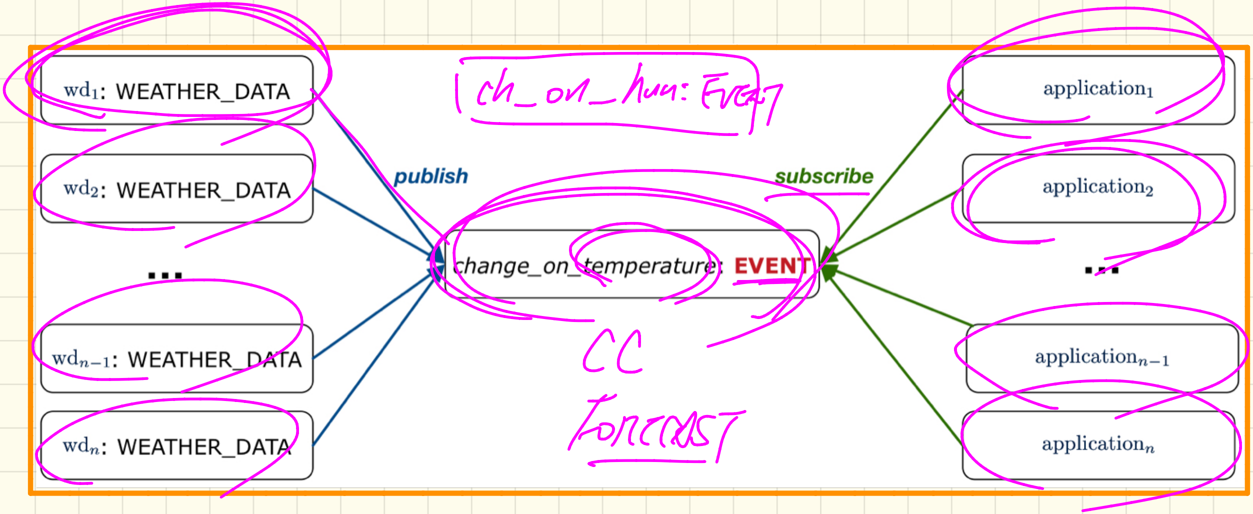
```
class CURRENT_CONDITIONS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
       weather_data.observers.has (Current)
    end
```

wd wd wd

```
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
```



Event-Driven Design: Multiple Subjects and Observers



Complexity?

Adding a new subject?

Adding a new observer?

Adding a new event type?

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
```

humidity

agent update_hum(?)

agent u_f(?, ??, ?, _)

```
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
```

```
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
```

[ ]

[t]

[t1, t2]

PROCEDURE

actions.item(args) ✓

END OF NOTES

ALL THE BEST !