EECS3311

Software Design

Winter 2020

Instructor: Jackie Wang

LECTURE 01

MONDAY JANUARY 06

Course Learning Outcomes (CLOs)

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

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

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

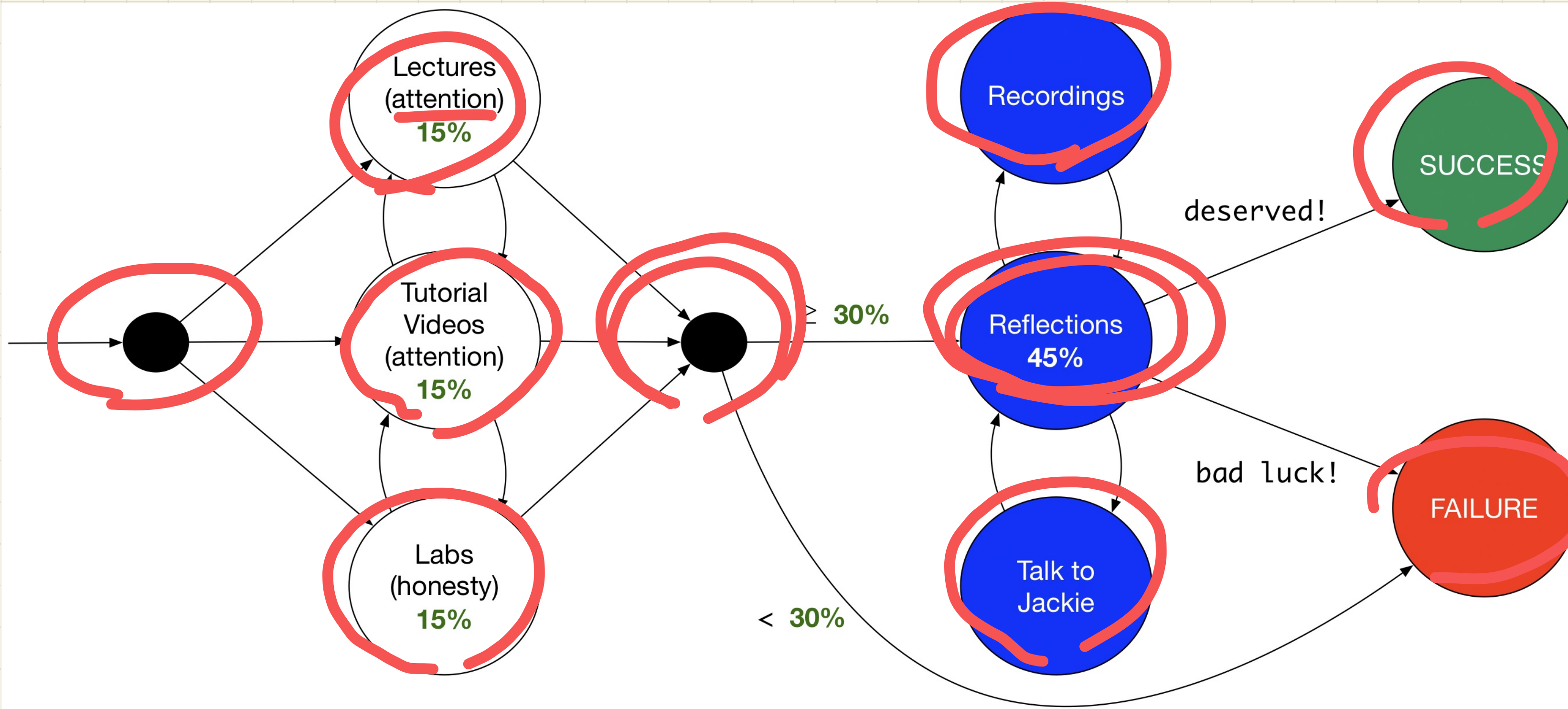
CLO4 Develop insight into the process of moving from an ambiguous problem statement to a well-designed solution.

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

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

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

Surviving through this Course



Software Development Process

EECS 4312

REQUIREMENT

- Natural Language
  (incomplete, ambiguous, contradicting)
- Requirement Elicitation

working payment system

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

web interface and deployable on mobile devices

↓

DESIGN

- Blueprints
- Not necessarily executable & testable

UML

↓

2020
2021

IMPLEMENTATION

- API Given
- Efficient (data structures & algorithms)
- Unit Tests

↓

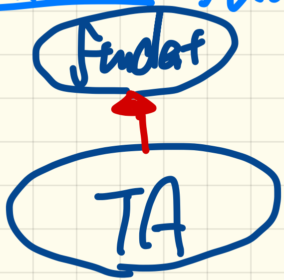
RELEASE

- Customer's $\boxed{\text{Acceptance}}$
- Return?

Relationships between modules / classes

1. Inheritance

```
class Student {
  - -
}
class TA extends Student {
  . -
}
```



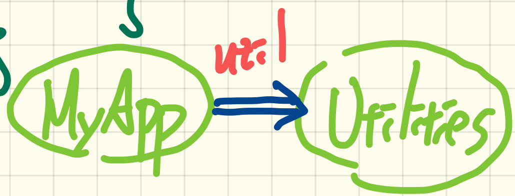
2. Client-Supplier relation.

client.

```
class MyApp {
  Utilities util = new - -
  util.sort(...);
}
```

supplier

```
class Utilities {
  - - sort(int[] --) {
  }
}
```



Client vs. Supplier in OOP

client — the class where the supplier obj is declared and called

```
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);
  } }
```

declare

use

type of C.O. ⇓ type of supplier

context object

supplier.

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. */
  } }
```

supplier
type

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

① ②

satisfied

before-state
pre-state

method.

obligation of client
benefits of supplier.

on? ①
locked? ②
non-explosive? ⊗
↳ ??? is never checked before the call

if any ob. of client not met, method do not exec.

m. heat(obj)

post-state
after-state

→ obligat. of supplier.
bef. of client.

when it happens ← * [fault
exp. is the client's

A Simple Design Problem: Bank Accounts

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

REQ2: We may *withdraw* an integer amount from an account.

Bank Accounts in Java: Version 1

```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;
        }
}
```

-10

-10

Bank Accounts in Java: Version 1 Critique (1)

should be positive.

```
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);
```

obligation of client is not met

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
```

not good ∵ amount of withdraw is neg.

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
```

Precondition vs. Exception

↳ service cond. ↳ error cond.

double divide(double x, double y)

$y == 0$

① $y == 0$

① $\equiv \neg$②, ② $y \neq 0$

Precondition

$\neg(y \neq 0)$

```
if (y == 0) {
    throw IAE(...);
}
```

Lecture 2
Wednesday January 8

- Lab0

- Textbook: OOSC2 on course wiki

- Slides for self-study:
  * Eiffel: Overviews of Syntax
  * Eiffel: Common Error
  * BON Design Diagrams

A Simple Design Problem: Bank Accounts

≥ 0 X

≥ 0

> 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 1

```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 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;
        }
}
```

-10

-10

Supplier

Client

```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)

```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;
    }
                                  -1M
    public void withdraw(int amount) {
        this.balance = this.balance - amount;
    }
    public String toString() {
        return owner + "'s current balance is: " + balance;
    }
}
```

Supplier

Client

```java
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 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;
        }
}
```

150 100 150

Supplier

Client

```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
```

X

divide (10, 0)

Precondition

↳ service

Exception

↳ error

double divide (x , y)

require ← precond

$\boxed{y \mathrel{!}= 0}$ B

0

F

precondition violation

↳ client's fault

caller

double divide(x , y)

!B

```
if (y==0){
    throw new IAE.
}
else{
    -    -
}
```

```
class MicroWave {
  locked
  on
  void heat(object ...) {
    :
  }
}
```

```
m.lock()
m.power()
if(obj is explosive){
  ...
} else {
  m.heat(obj);
}
```

deref

```
class MicrowaveUser {
  main(...){
    Microwave m = ...
    Object obj = ??;
    m.heat(obj);
  }
}
```

m.on F m.locked T

supplier

calling this method should cause precondition violation

∴ ✓ on is F

✓ locked is F

✓ obj is not guaranteed to be non-explosive.

Bank Accounts in Java: Version 2

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

corresponding precondition

$!(amount < 0)$

$\equiv$

$amount \geq 0$

service condition.

exception conditions.

Bank Accounts in Java: Version 2 Critique (1)

Compared with Version 1

```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; }
  }
```

-10

-10 <0

Supplier

Client

```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

100

-1M

```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; }
  }
```

Supplier

Client

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

Console Output:

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

Bank Accounts in Java: Version 2 Critique (3)

Compared with Version 1

```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; }
  }
```

Supplier

Client

```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; }
  }
```

b: 100

solution 1: <=

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
```

class invariant → not change

↳ property that holds true for all objects of a particular class

e.g. Account.

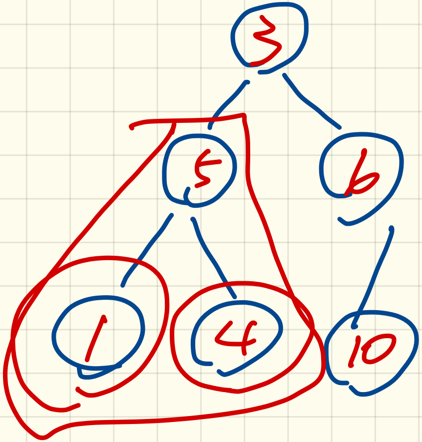
↳ invariant: balance ≥ 0.

invariant violation.

$\xrightarrow{\text{create}} 10 \xrightarrow{\text{withdraw } 5} 5 \xrightarrow{\text{deposit } 100} 105 \xrightarrow{\text{withdraw } 200} -95$ ✗

class invariant for BST

search




```
1. bool isBST( node){
     if( l ){
     l< node ; isBST(l)
     }
     if( r ){
       r> node
     }
   }
```

1. When the data structure is large, checking contract (e.g. class inv) is inefficient.

2. Development

$v1 \to v2 \to \dots \to v100 \to v_{final}$

all contract checking turned off.

finalize

trading time for correctness

contracts are necessary to guide the devel.

Single Choice Principle

```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; }
  }
```

assert bala. $\cancel{>}$ 0. ≥ 0

$\boxed{\text{assert balance } \cancel{>} \text{ 0.}}$ ≥ 0

$\boxed{\text{assert bala. } \cancel{>} \text{ 0}}$ $\geq$

class invariant.

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";
  }
```

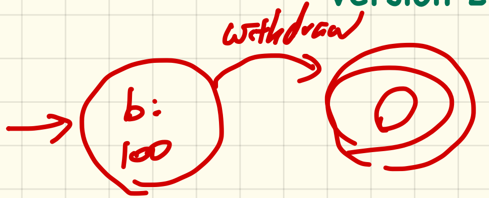
C.I.

Bank Accounts in Java: Version 3 Critique (1)

Compared with Version 2

b: 100

```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";
  }
```



Supplier

Client

```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"; }
```

obl: of clients

where the service is provided

When the amount is neither negative nor too large,
is there any obligation on the supplier of withdraw?

Bank Accounts in Java: Version 4

with an evil supplier

```
public class AccountV4 {
  public void withdraw(int amount) throws
    WithdrawAmountNegativeException, WithdrawAmountTooLargeException
  { if(amount < 0) { /* negated precondition */
      throw new WithdrawAmountNegativeException(); }
    else if (balance < amount) { /* negated precondition */
      throw new WithdrawAmountTooLargeException(); }
    else { /* WRONT IMPLEMENTATION */
      this.balance = this.balance + amount; }
    assert this.getBalance() > 0 :
      owner + "Invariant: positive balance"; }
```

wrong imp. of setter

Bank Accounts in Java: Version 4 Critique

```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

100

150

T

wrong - 150

balance = x

acc.withdraw(a)

$\boxed{\text{balance} = y}$

y = x - a.

Supplier

missing obligation for supplier.

Client

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

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

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"; }
```

postcondition

new value

future balance = old balance - a.

Lecture 3

Monday January 13

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"; }
```

"cache" the pre-state value of balance

current value of bal.

pre-stated value cached

postcondition

obj. of client

pre-state → o.m(...)

post-state → postcondition

Bank Accounts in Java: Version 5 Critique

Compared with Version 4

```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"; }
```

Supplier

Client

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

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

Design by Contract in Java

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

error conditions
(¬ precondition)

single
choice
principle

post-cond.

class inv.

not public.

Supplier

Contracts public

Precondition
Postcondition
class invariants

Client

```java
public static void main(String[] args) {
  System.out.println("Create an account for Jim with balance 100:"
  try {
   AccountV2 jim = new AccountV5("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.");
  }
```

all the client wants to do

overhead of client.

Design by Contract in Eiffel

Contract View

```
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
```

```
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
```

(tracing tag.)

Bool expr.

single place.

Implementation View

Features in Eiffel

Attributes

balance : INTEGER

Queries (≈ accessors)

get_balance : INTEGER
require ...
local ...
do ...
ensure ...
end

Commands (≈ mutators)

set_balance (nb: INTEGER)

require ...
local ...
do ...
ensure ...
end

imp

spec.

Commands vs. Constructors

Constructors → Commands listed under create clause

class A

feature -- Commands

make_1 (..)
do
end -
make_2 (..)
do
end --
end

class B

⋮

local
a: A
do
create a.make_1 (..) X
create a.make_2 (..) X

create
default_create

class A

create
make_2

feature -- Commands {NONE} export

make_1 (...)
do
--
end

make_2 (...)
do
--
end

end

class B

...

local
a: A

do

new

① create a.make_1 (...) ✗

∵ make_1 not listed in create

② create a.make_2 (...) ✓

create a new obj.

used as a constructor

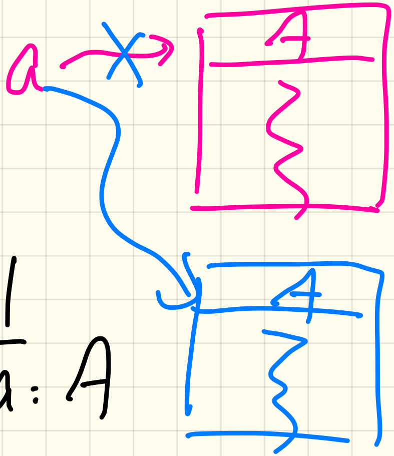
③ a.make_1 (...)

④ a.make_2 (...)

no new objects are created ✓

used as a command.

```
class A

  create
        make_1, make_2

features
        make_1(..)
            :

        make_2(..)
            :
```

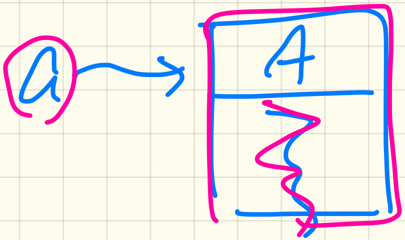



```
   :
local
     a: A
do
  create a.make_1(..)
  create a.make_2(..)
```

```
class A

  create
    make_1, make_2

features
    make_1(..)
      :

    make_2(..)
      :
```

create {A} a.make_2(..)

```
class B
  :
  local
    a : A
  do
    create a.make_2(..)
    a.make_1
```



Java.

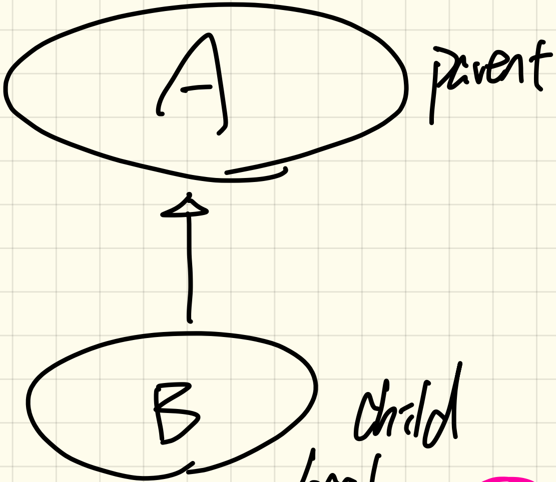
A obj = new ?(...);

may or may not be A.

Eiffel

local
  obj: A
do
  create {?} obj.~~(..).

when the "?" same as the static type of obj, you can omit it {?}.



<u>local</u>
a1 : A
a2 : A

<u>create</u> {A} a1.make (...)

<u>create</u> {B} a2.make (...)

dynamic type.

create a1.make (..)

local

a: A

do

a.make_2(...)

end

Eiffel 101.

not Compile

Void Safety.

cmd (- . . -)
do

- .

end

$\approx$

cmd (. .)
require
True
do
- -
ensure
$\boxed{\text{True}}$
end

not appropriate
-; no constraint
on suppl.

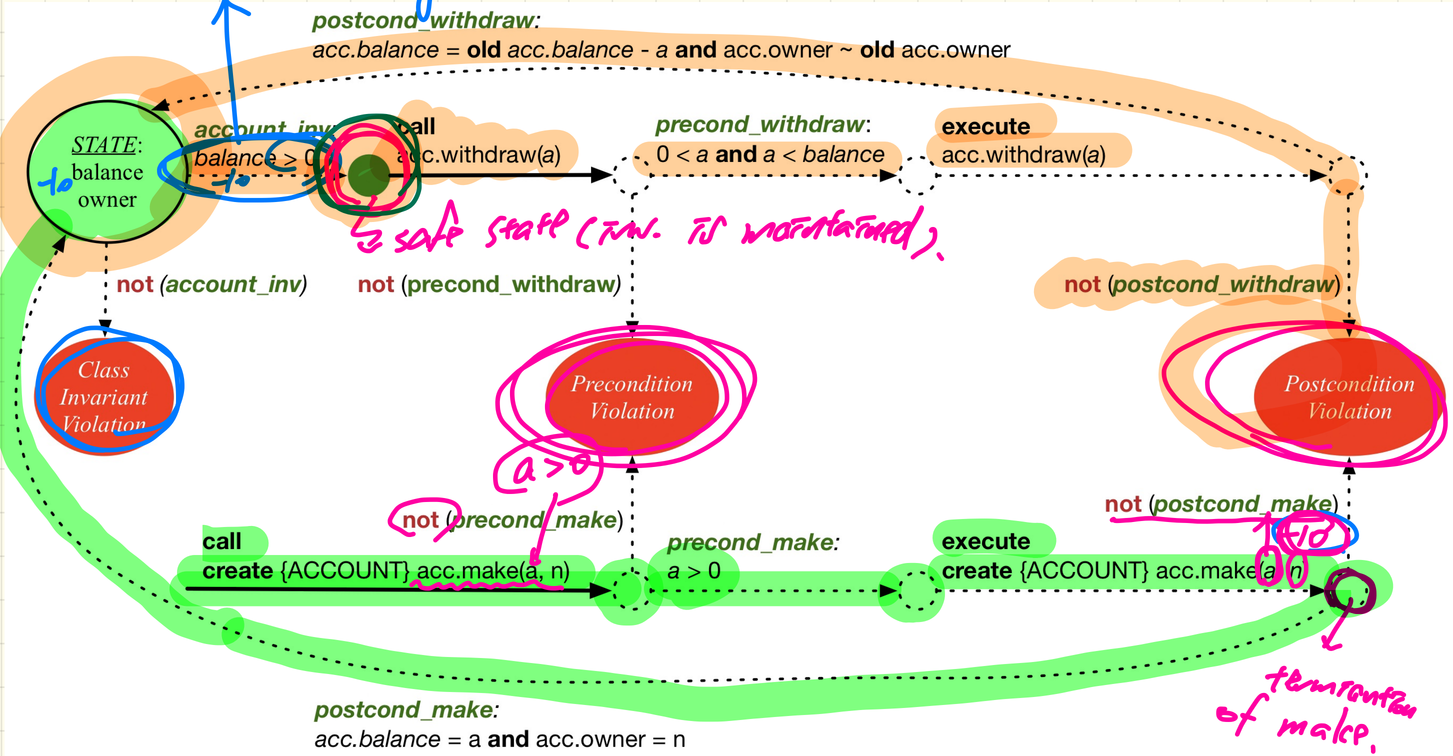
```
class ACCOUNT
create
        make
feature -- Attributes
        owner : STRING
        balance : INTEGER
feature -- Constructors
        make(nn: STRING; nb: INTEGER)
            require -- precondition
                positive_balance: nb > 0
            end
feature -- Commands
        withdraw(amount: 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
```

-10 100 cur. bal. 100 0

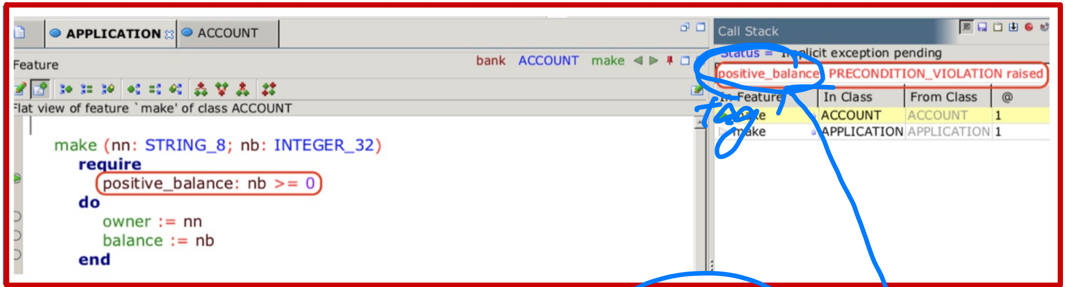
Runtime Monitoring of Contracts

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

checking inv. to make sure integrity of account is not compromised.



Precondition Violation: positive_balance



Supplier

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

-10

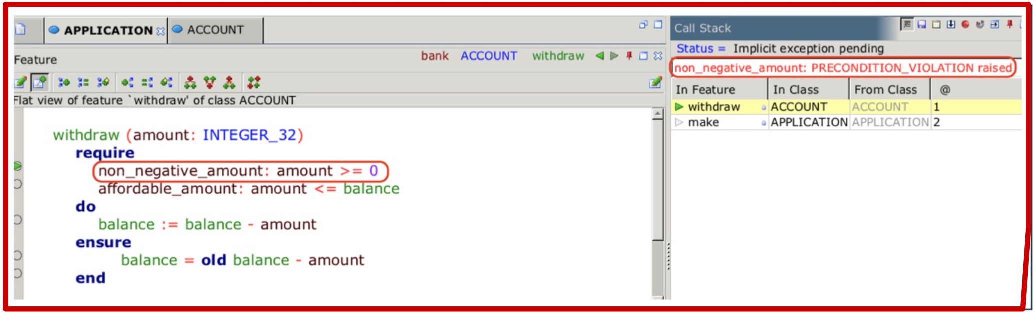
F

Client

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

Precondition Violation:

non_negative_amount



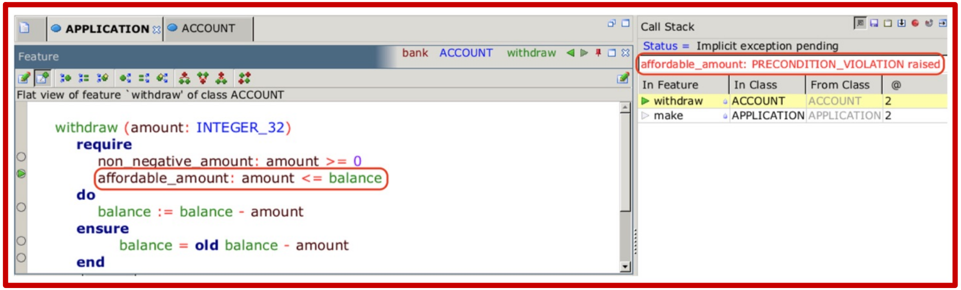
Supplier

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

Client

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

Precondition Violation: affordable_amount



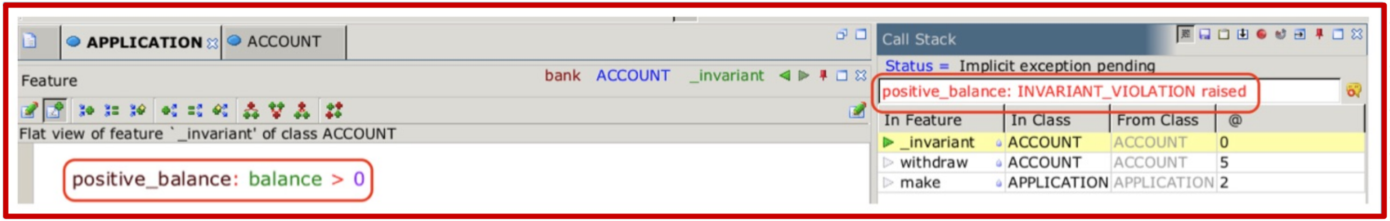
Supplier

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

Client

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

Class Invariant Violation: positive_balance



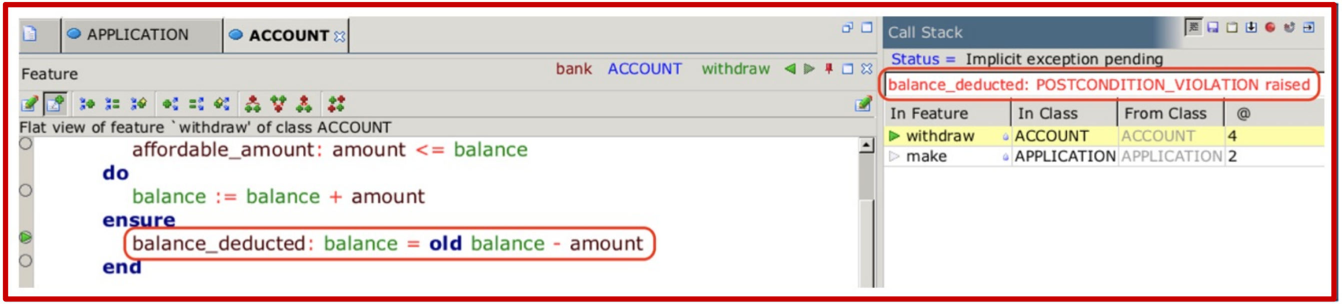
Supplier

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

Client

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

Postcondition Violation: balance_deducted



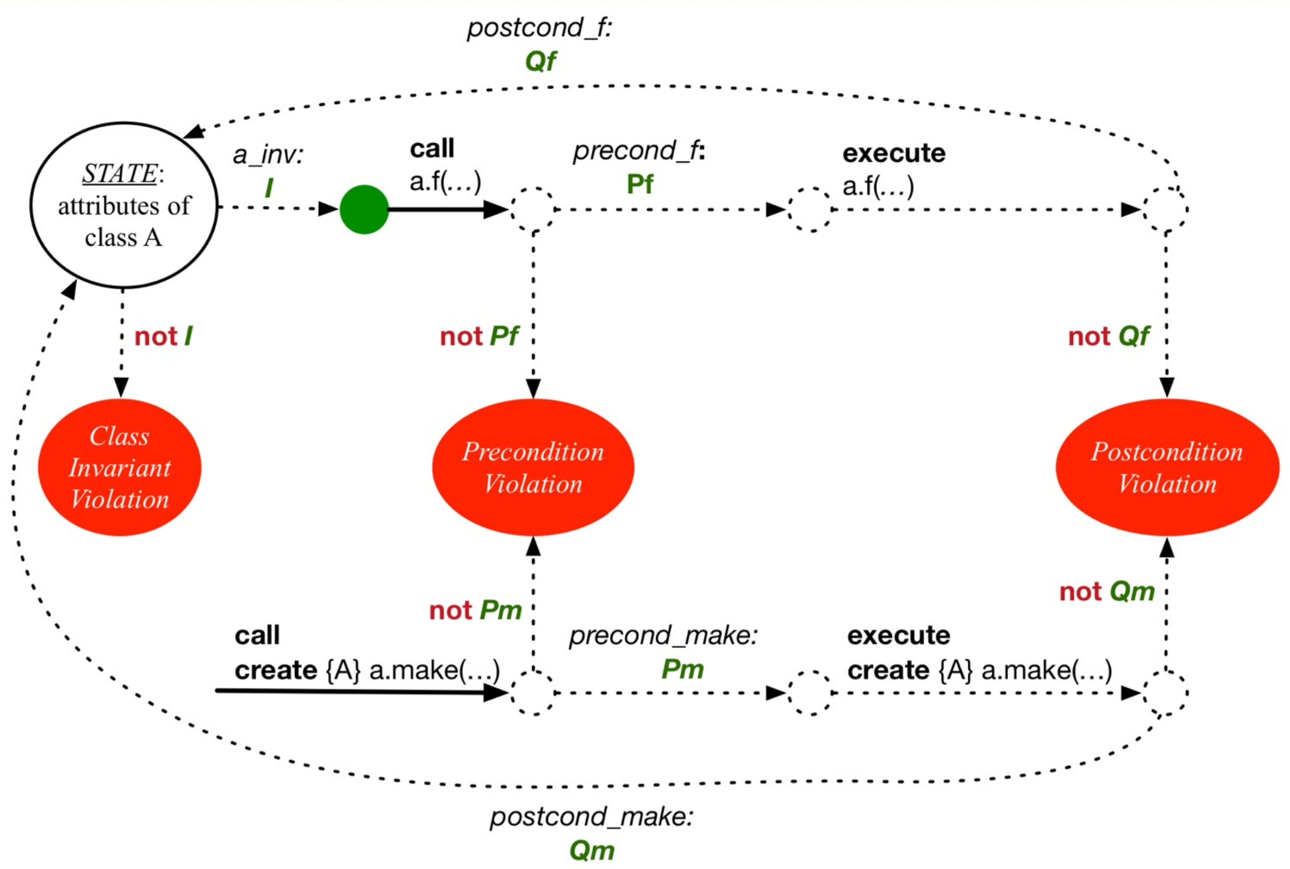
Supplier

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

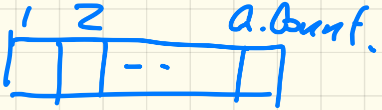
Client

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

Runtime Monitoring of Contracts



Precondition & Postcondition Exercise



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

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

require

??

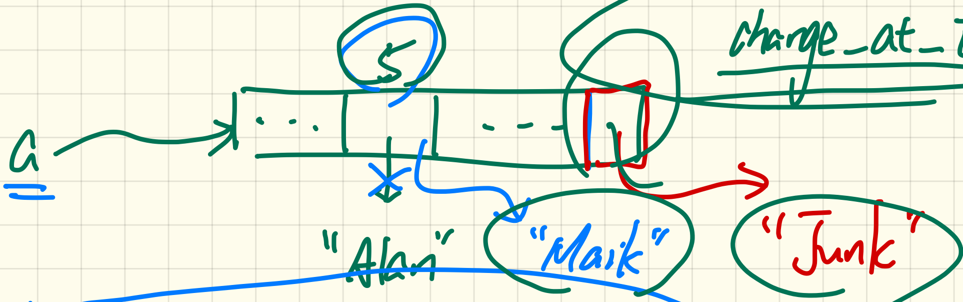
valid_index: 1 <= i and i <= a.count.

ensure

??

a[i] ~ ns

change_at_i : a[i] ~ ns ✓



1010 - ∀ ∃

change_at(a, 5, "Mark")

All elements except the one at index i have not changed their values.

Lecture 4
Wednesday January 15

Shorter Office Hours today: 3pm to 4pm

Lab0: Tutorial Videos (basic syntax, debugger)

Lab1: See due date in course wiki

plagiarism check

This Friday: in-lab demo at 10:30

Precondition & Postcondition Exercise

$\forall$ $\exists$

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

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

require

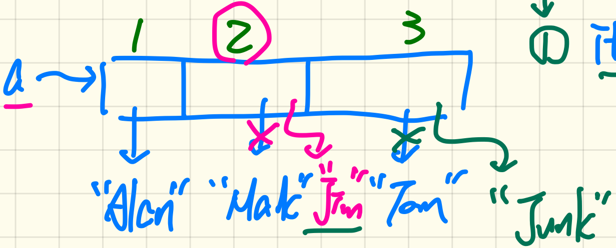
??

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

ensure

??

① item_at_i_changed : $a[i] \sim ns$



② others_unchanged :

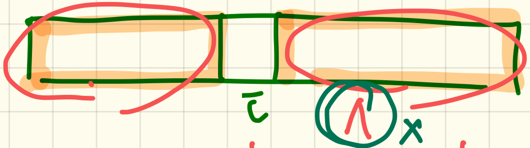
valid index

not the position to change

$\forall j \mid 1 \leq j \leq a.count \wedge j \neq i \bullet$

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

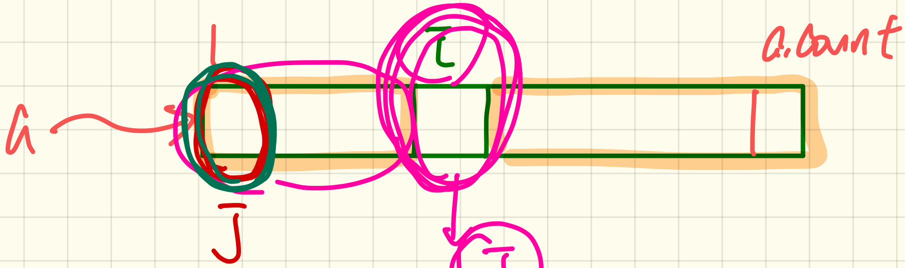
change_at (a, 2, "Jim")



✓ $\forall \bar{j} \mid 1 \leq \bar{j} \leq \bar{i}-1 \vee \bar{i}+1 \leq \bar{j} \leq a.count \bullet$

$a[\bar{j}] \sim old\ a[\bar{j}]$

$$\left(\forall x \mid R(x) \bullet P(x)\right) \equiv \boxed{\left(\forall x \bullet R(x) \Rightarrow P(x)\right)}$$



$\forall j \mid 1 \leq j \leq a.count \bullet$

X ① $j \neq i \land a[j] \sim old\ a[j]$ (T, F, X)

✓ ② $j \neq i \Rightarrow a[j] \sim old\ a[j]$

if $j = i$ (True) **then**

✓ $T \Rightarrow \sqcup$ (T)

else

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

$F \Rightarrow \sqcup$ T

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

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

require

??

ensure

??

$\forall$

= $\Leftrightarrow$

and $\wedge$ ==

or $\vee$

implies $\Rightarrow$

$\exists$ ① item_at_i_changed : $a[i] \sim ns$

② others_unchanged : $\forall j \mid 1 \leq j \leq a.count \bullet j \neq i \Rightarrow a[j] \sim old\ a[j]$

across | 1 .. | a.count is j

all some -

j /= i implies a[j] ~ old a[j]

equals -

end /~

Exercise: translate to Eiffel.

ensure

①

②

③

⋮

① $\wedge$ ② $\wedge$ ③

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

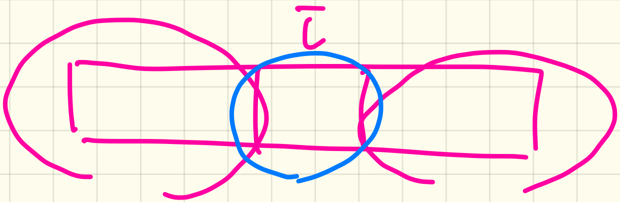
$$i = j \Rightarrow a[j] \sim ns$$

$$\wedge$$

$$i \neq j \Rightarrow a[j] \sim \underline{old}\ a[j]$$

$a[i] \sim ns$ **and** .

across . . . **end**



$$\forall x \underset{\text{such that}}{\mid} \underbrace{\qquad\qquad\qquad}_{\text{Range}} \underset{\text{it's the case}}{\bullet} \underbrace{\qquad\qquad\qquad}_{\text{property}}$$

$$\forall x \mid 1 \le x \le 5 \bullet \left[x^2 > 25 \right]^{F} ; \; \underbrace{\{ \ge 25}_{F}$$

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

(0) array_size_unchanged: a.count = old a.count

(1) a[i] ~ ns

(2) across 1 |..| a.count is j all i /= j implies a[j] ~ old a[j]

end

old a: Alan | Mark | Tom

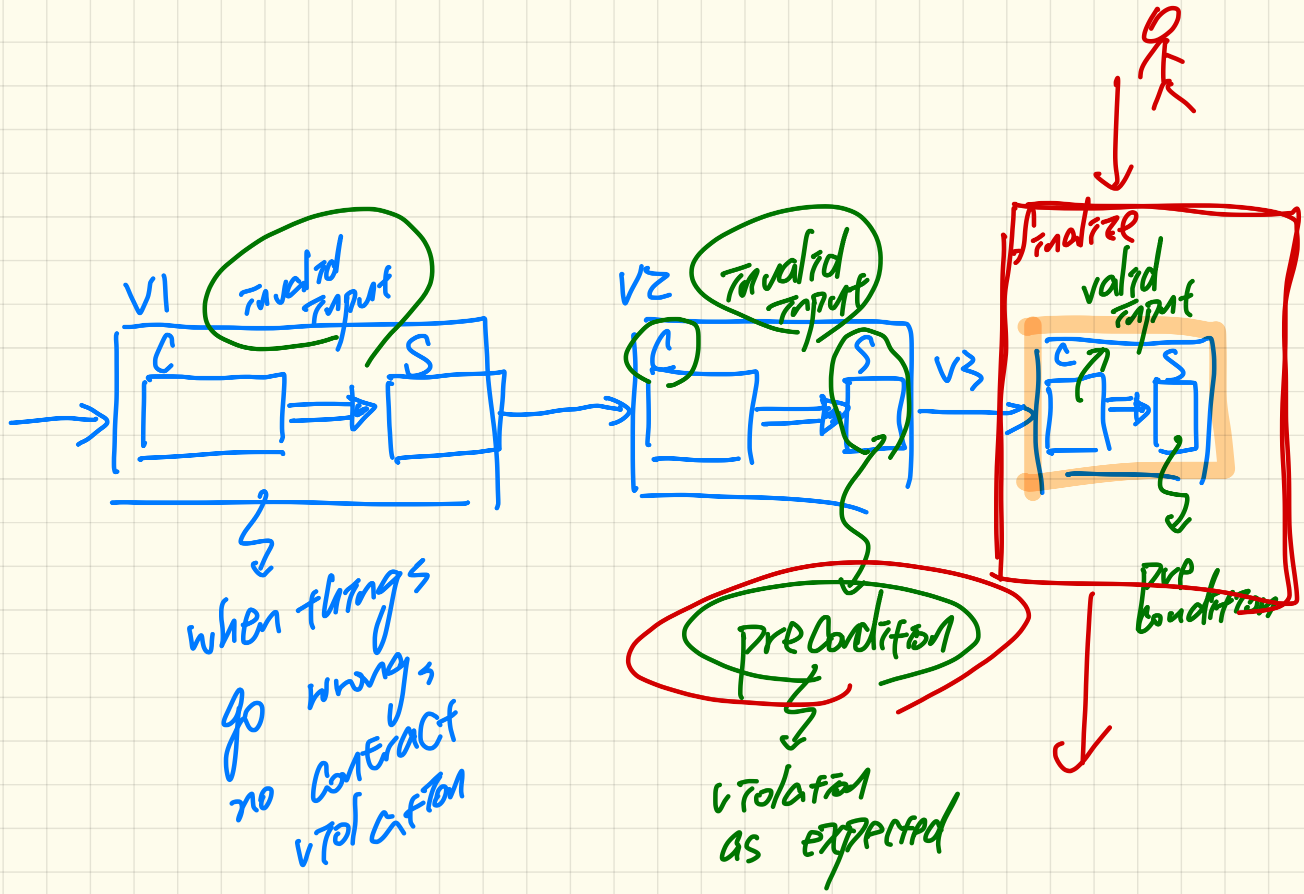
change_at

new a

old a[4]

~~(2) a.count = old a.count~~

array invalid index violation


V1
Invalid input
C
S
when things go wrongs no contract violation
V2
Invalid input
C
S
V3
precondition
violation as expected
finalize
valid input
C
S
pre condition

How is DbC Useful in Guiding System Development?

Client's View:

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

Expected Run

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

Supplier's Implementation Strategy

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

Version 1: Wrong Implementation, No Contracts

```
class interface
   DATABASE_V1

create
   make

feature -- Constructor

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

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

   make
         -- Create an empty database.

end -- class DATABASE_V1
```

u

```
class interface
   UTILITIES_V1

create
   default_create

feature -- Binary Search

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

end -- class UTILITIES_V1
```

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

Version 1: User Interaction Session

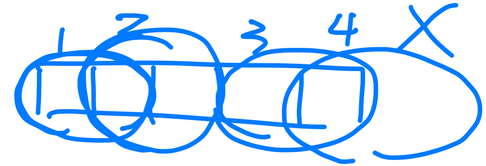
```
Enter a name, or `done` to start inquiring: e
Enter a name, or `done` to start inquiring: c
Enter a name, or `done` to start inquiring: d
Enter a name, or `done` to start inquiring: a
Enter a name, or `done` to start inquiring: b
Enter a name, or `done` to start inquiring: done
e  c  d  a  b
Enter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

Version 2: Wrong Implementation, Proper Precondition

```
class interface
   DATABASE_V2

create
   make

feature -- Constructor

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

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

   make
         -- Create an empty database.

end -- class DATABASE_V2
```

u ⟹

```
class interface
   UTILITIES_V2

create
   default_create

feature -- Binary Search

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

end -- class UTILITIES_V2
```



$\forall i \mid a.lower \le i \le a.upper - 1 \bullet a[i] < a[i+1]$

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

Version 2: User Interaction Session

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

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

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

Version 3: Fixed Implementation, Proper Precondition

input string

```
class interface
   DATABASE_V3

create
   make

feature -- Constructor

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

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

   make
         -- Create an empty database.

end -- class DATABASE_V3
```

$\xRightarrow{u}$

```
class interface
   UTILITIES_V3

create
   default_create

feature -- Binary Search

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

end -- class UTILITIES_V3
```

$\exists$

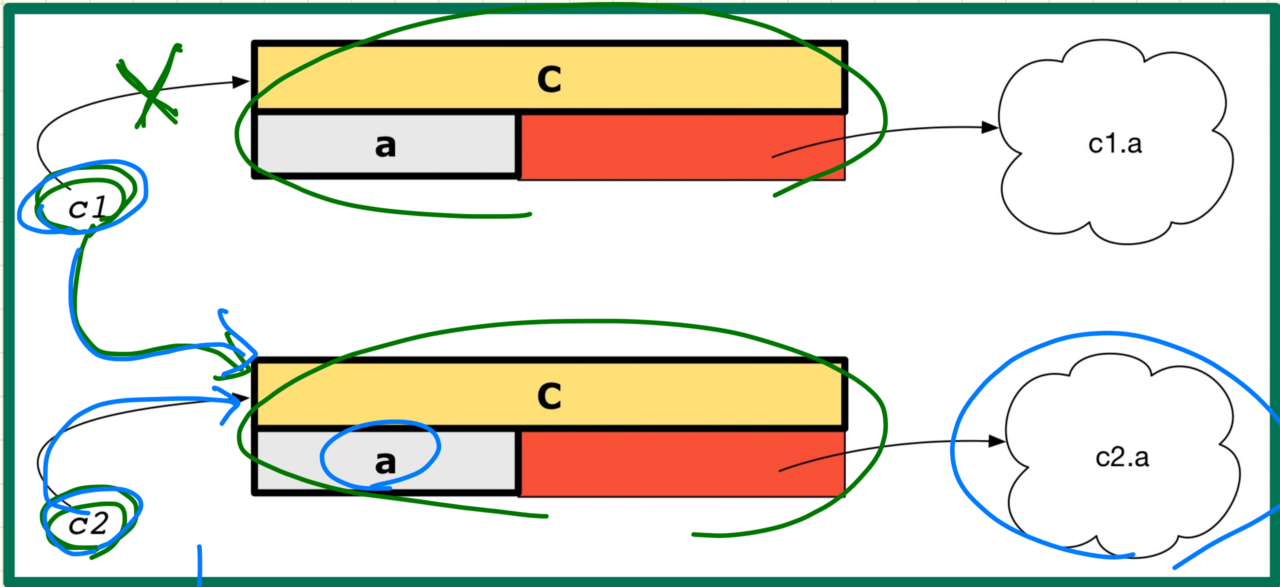
$\neg\exists$

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

Version 3: User Interaction Session

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

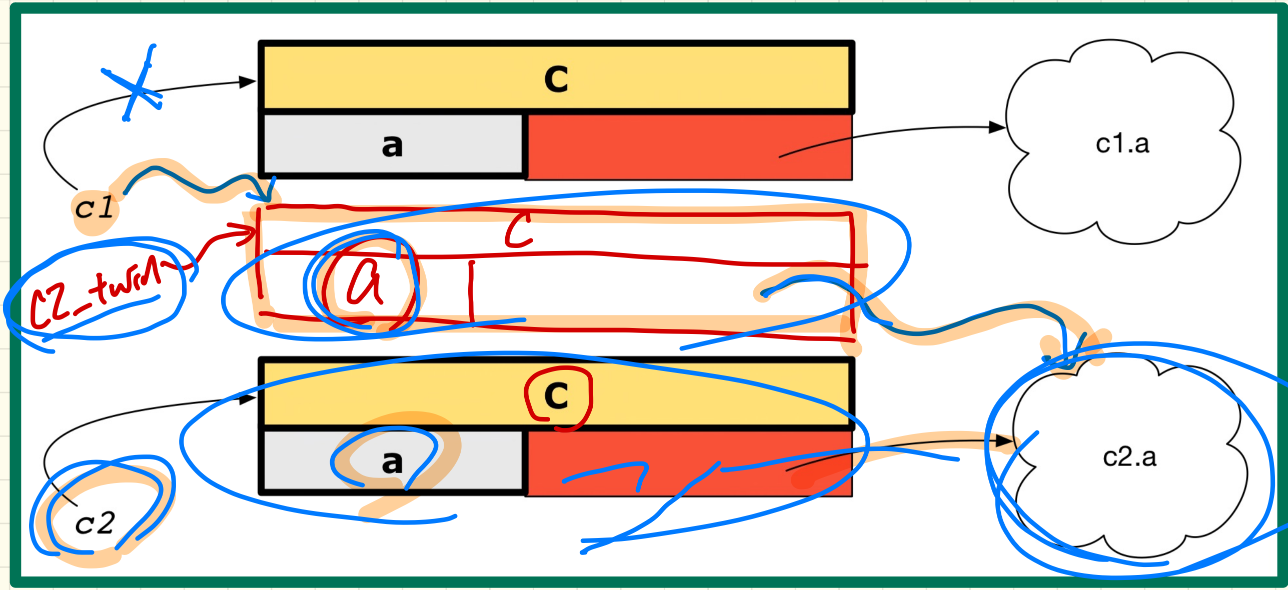
Reference Copy: c1 := c2 cheapest.



c1 = c2 T

c1.a = c2.a T.

Shallow Copy: c1 := c2.twin

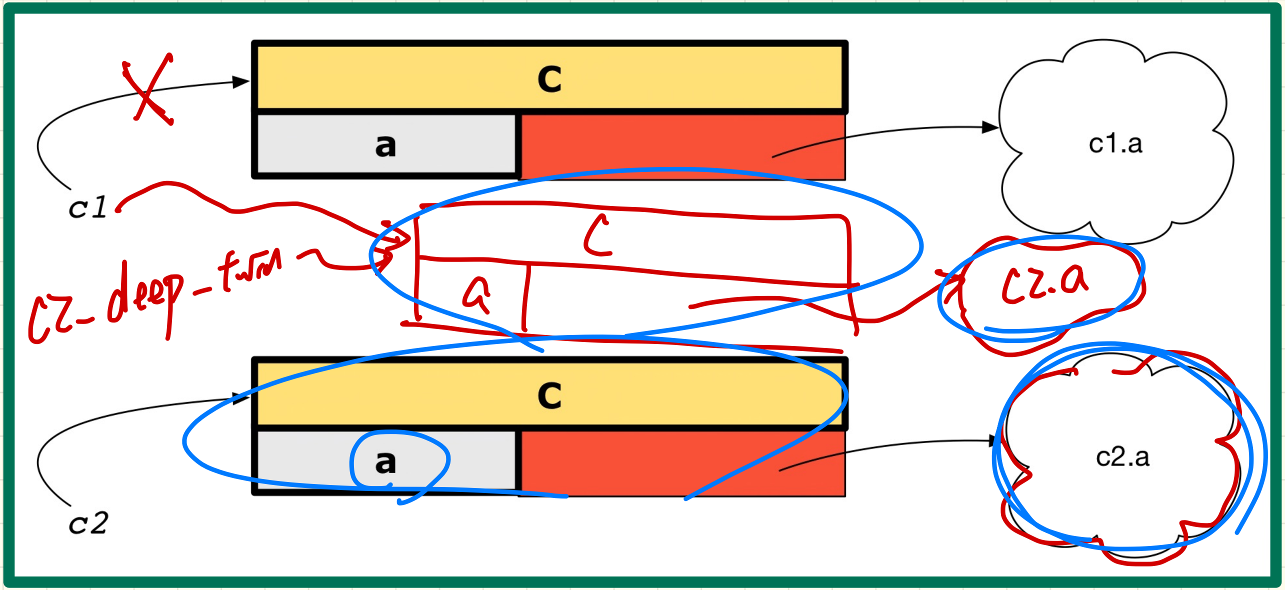


c2_twin.a := c2.a

c2 = c2.twin F

c2.a = c2.twin.a T

Deep Copy: c1 := c2.deep_twin



c2 = c2_deep_twin (F)

c2.a = c2_deep_twin.a (F)

LECTURE 5

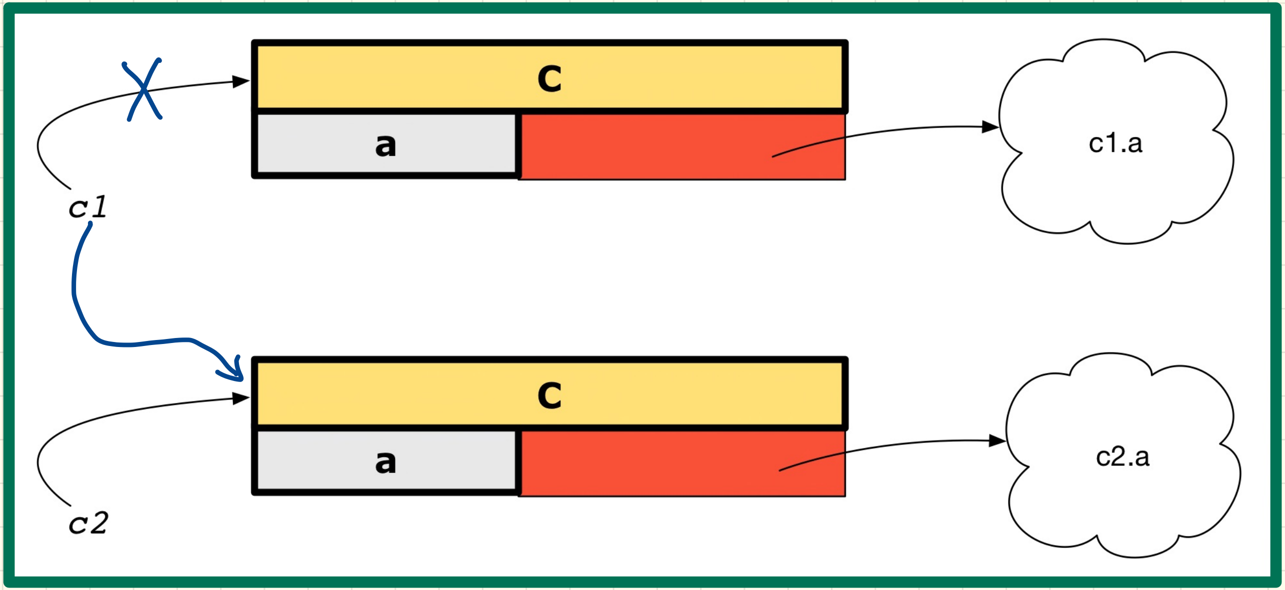
MONDAY JANUARY 20

In-Lab Demo last Friday:

- across
- comparator and sorter

Breakpoints and Debugger: See tutorial video

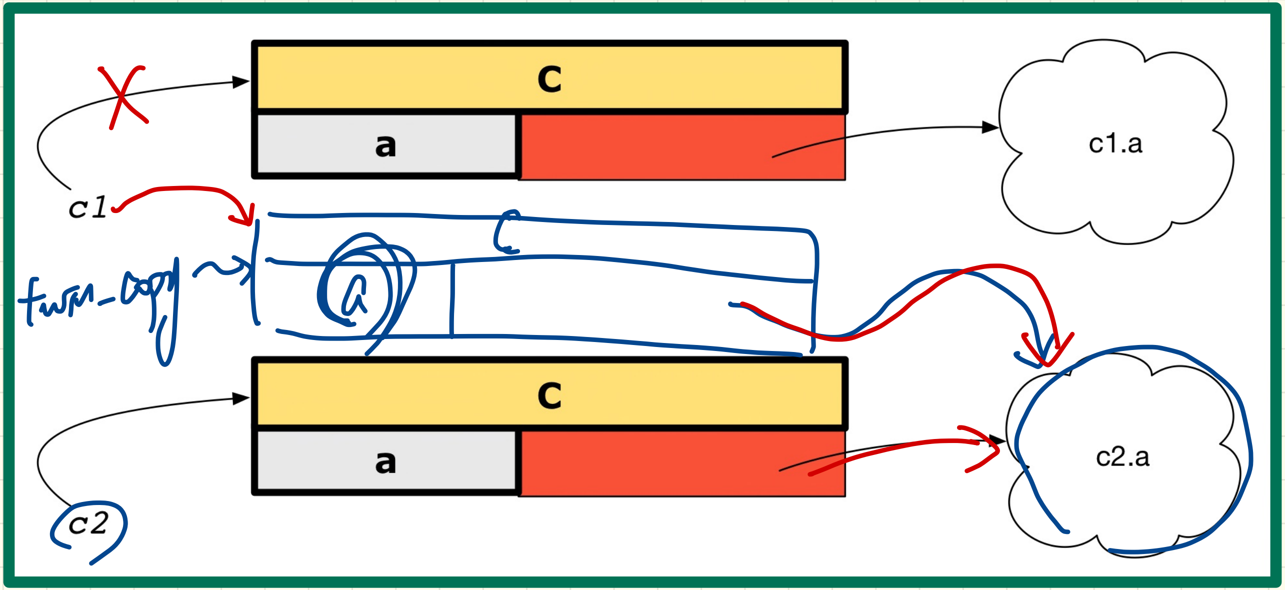
Reference Copy: c1 := c2



Shallow Copy: c1 := c2.twin

c2.a = c2.twin.a T

c2 = c2.twin F

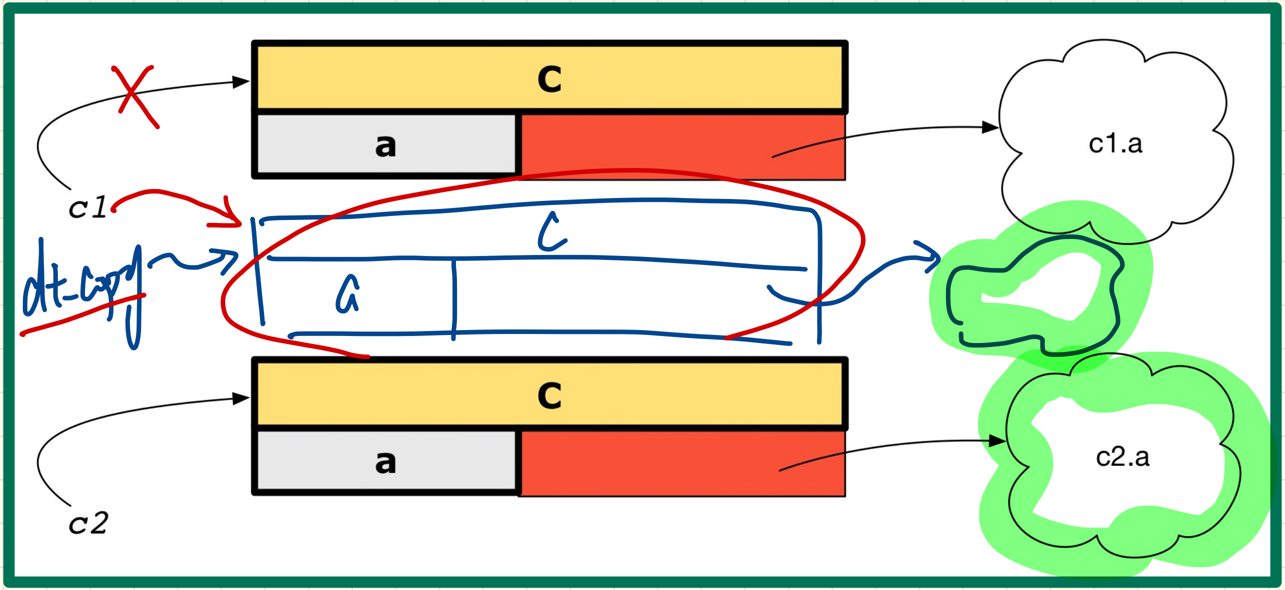


twin_copy.a := c2.a

Deep Copy: c1 := c2.deep_twin

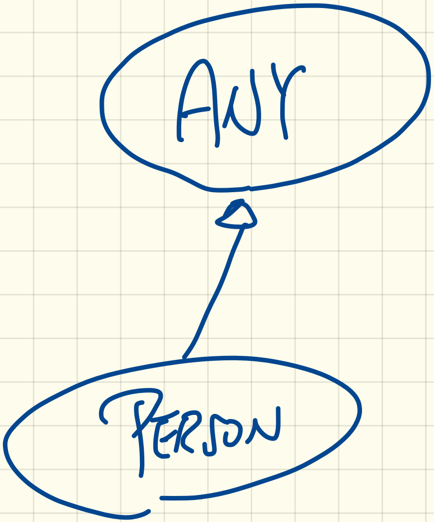
c2 = c2.dt F

c2.a = c2.dt.a F



dt_copy.a := c2.a.deep_twin

c1 := $\boxed{\text{c2.deep\_twin}}$



twin

deep_twin

Reference vs. Shallow vs. Deep Copies

- Initial situation:

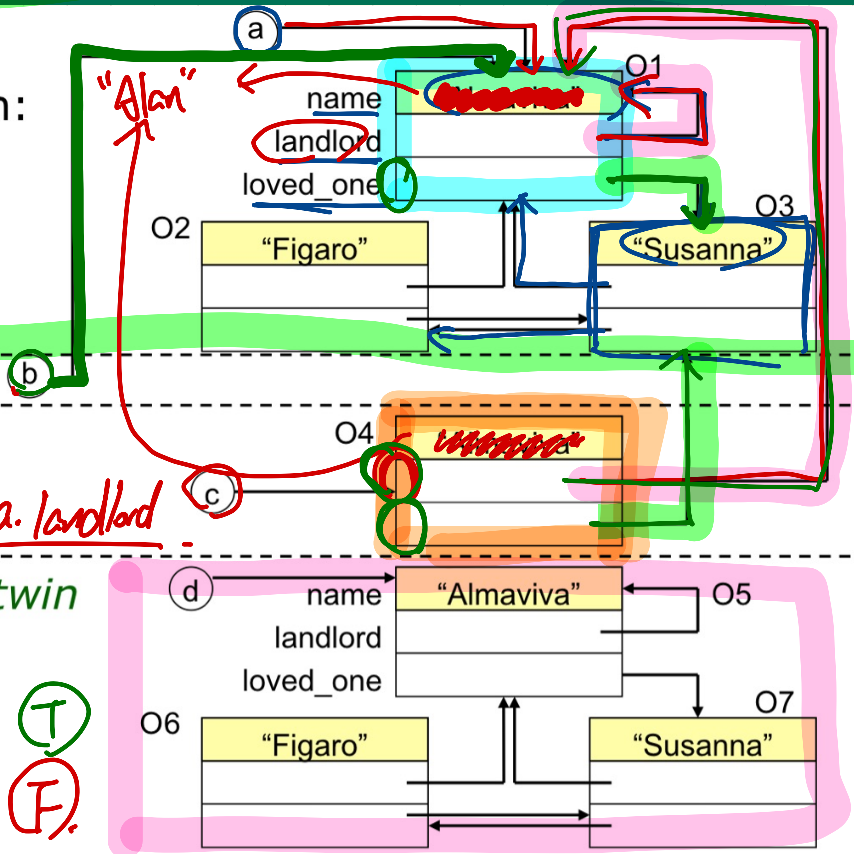
- Result of:

b := a

c := a.twin

c.landlord := a.landlord

d := a.deep_twin



c.landlord = a T

c.landlord = c F

f

do

⋮

ensure

old a

and old a.twin

old a.deep_twin

triggers different kinds of copies in pre-state

Collection Objects: Reference Copy & Make Changes

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

T

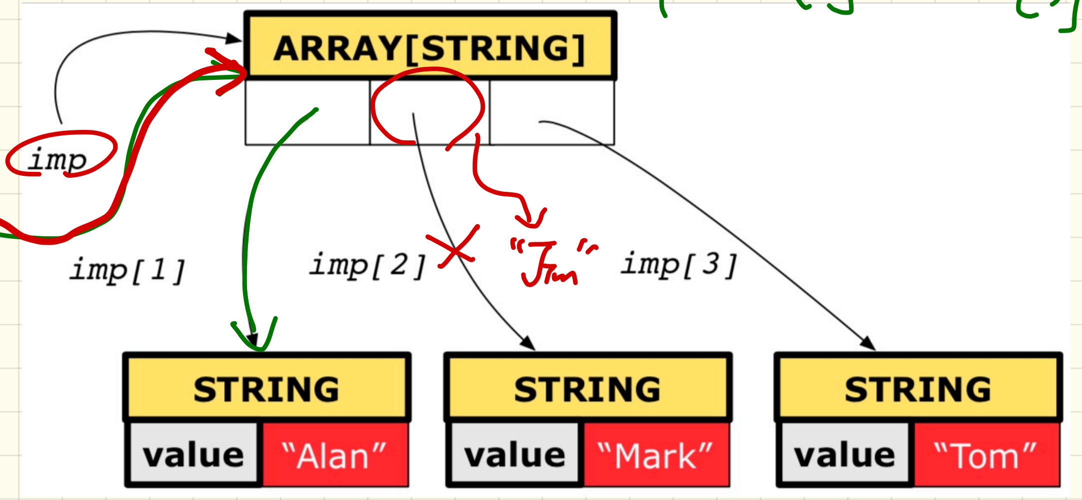
T

T

imp[1] ~ old_imp[1]

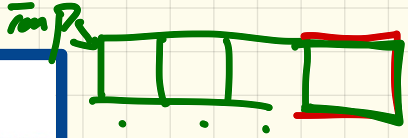
imp[2] ~ old_imp[2]

[3] [3]



old_imp

imp[2] ~
old_imp[2]




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

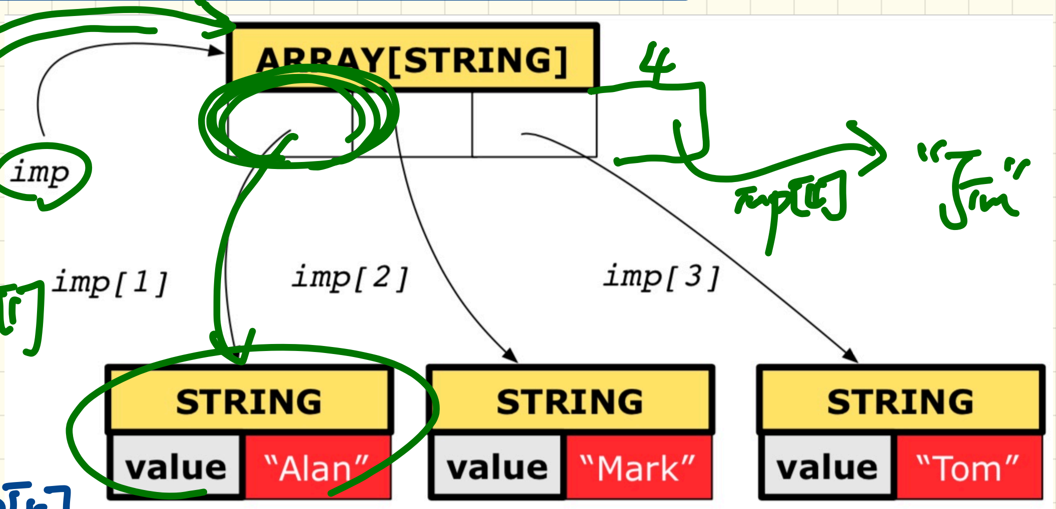
imp.force("Jim", imp.count + 1)

3

4

T

old_imp



imp[1] ~ old_imp[1]

⋮

imp[4] ~ old_imp[4]

Collection Objects: Shallow Copy & Make 1st-Level Changes

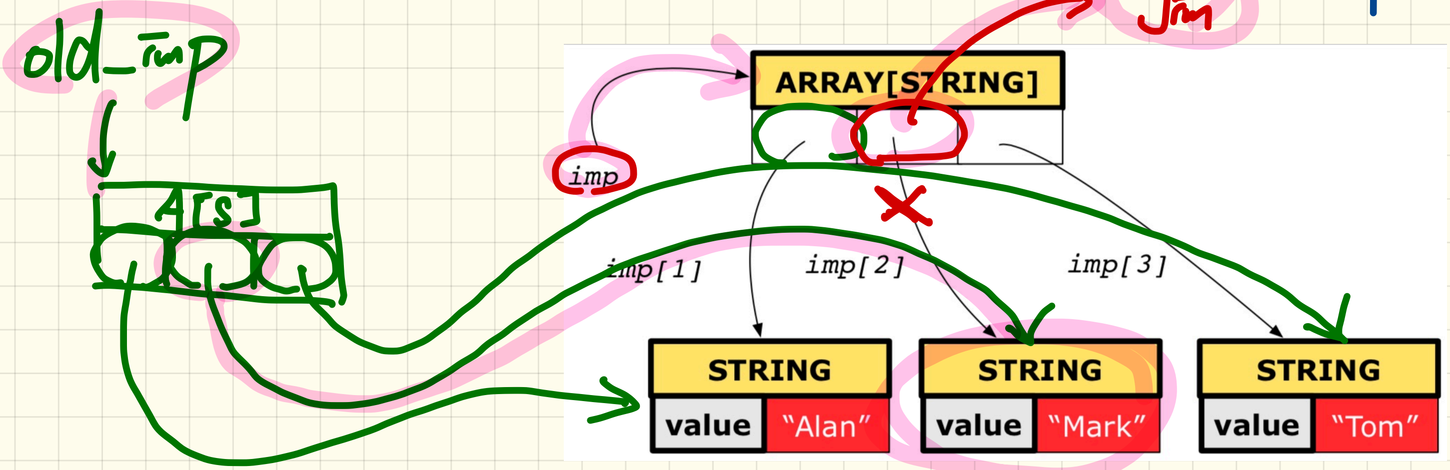
```
old_imp := imp.twin
Result := old_imp = imp   -- Result = [redacted]
imp[2] := "Jim"
Result :=
  across 1 |..| imp.count is j
  all imp [j] ~ old_imp [j]
  end -- Result = [redacted]
```

T

imp[1] ~ old_imp[1]

imp[2] ~ old_imp[2]

"Jim"



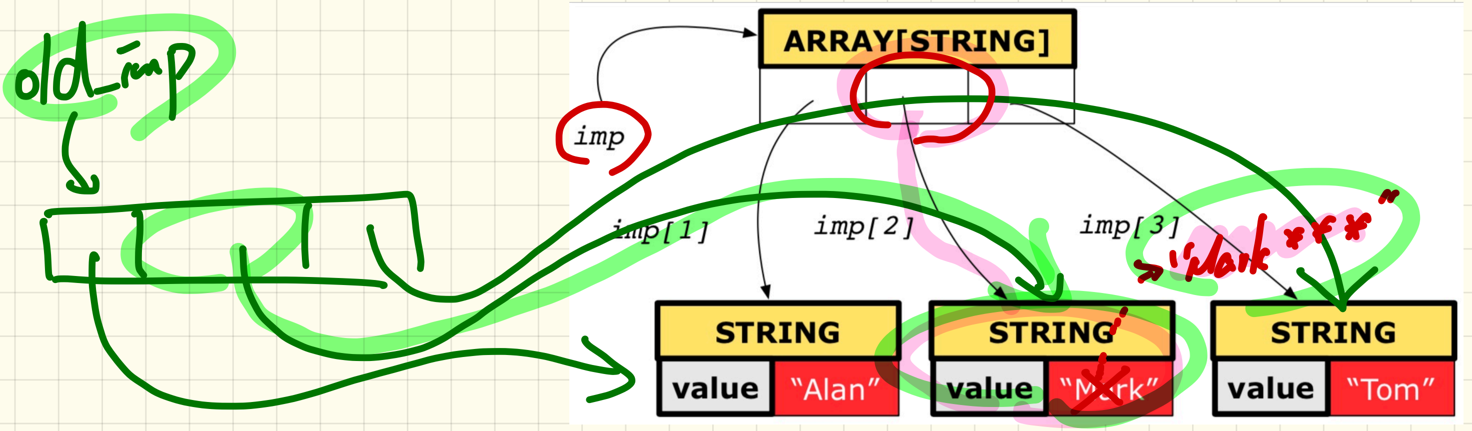
Collection Objects: Shallow Copy & Make 2nd-Level Changes

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

imp[1] ~ old_imp[1]

T

imp[2] ~ old_imp[2]



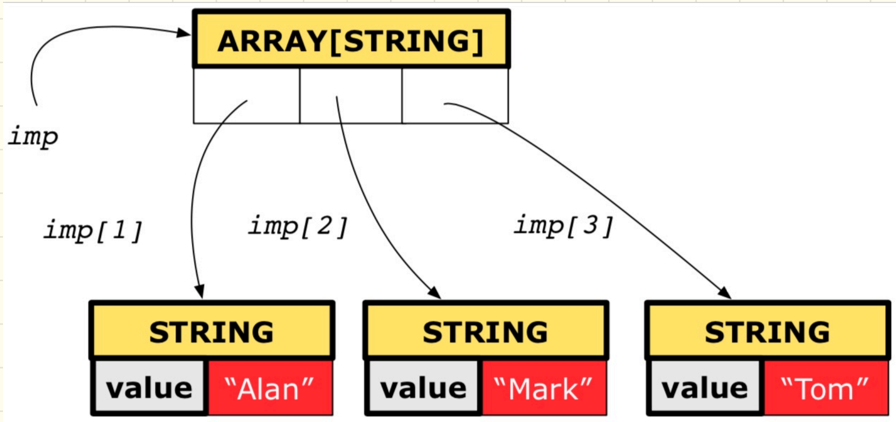
```
old_imp := imp.twin
Result := old_imp = imp  -- Result = false
imp.force("Jim", imp.count + 1)
Result :=
 across 1 |..| imp.count is j
 all imp [j] ~ old_imp [j]
 end -- Result = true
```

T?

F?

??

$\boxed{\text{EXERCISE}}$



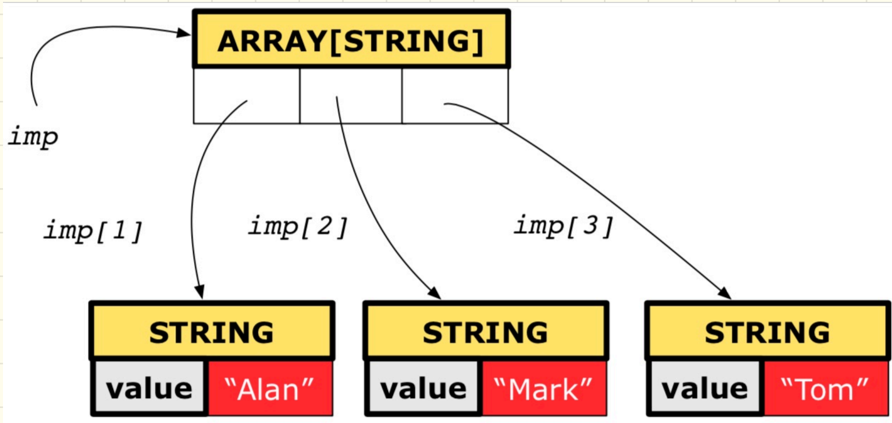
```
old_imp := imp.twin
Result := old_imp = imp  -- Result = false
imp[2].append ("***")
Result :=
  across 1 |..| imp.count is j
  all imp [j] ~ old_imp [j]
  end -- Result = true
```

imp.force("Jim", 2)

$\equiv$

imp[2] := "Jim"

Exercise



@[2]

Invalid index is
a runtime error,
not compile time.

class ___

a

b

f(- - -)

do

g ✓

end

c

d

local

do

g(...)

do

end

class ___

a : INT

f(a : INT)

$a.count = a.upper - a.lower + 1$

Create a.make_empty

a.count 0

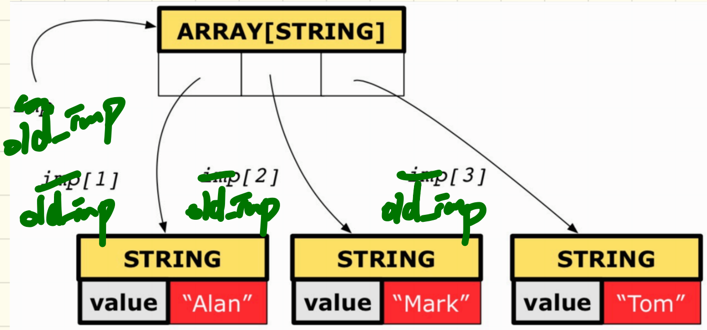
lower 1

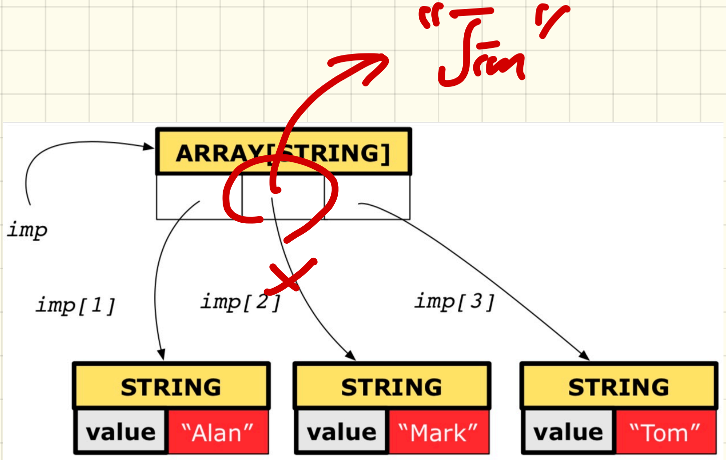
upper 0

$0 - 1 + 1 = 0$

Collection Objects: Deep Copy & Make 1st-Level Changes

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

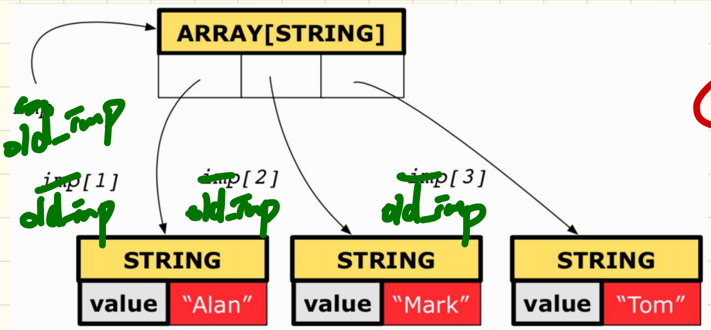


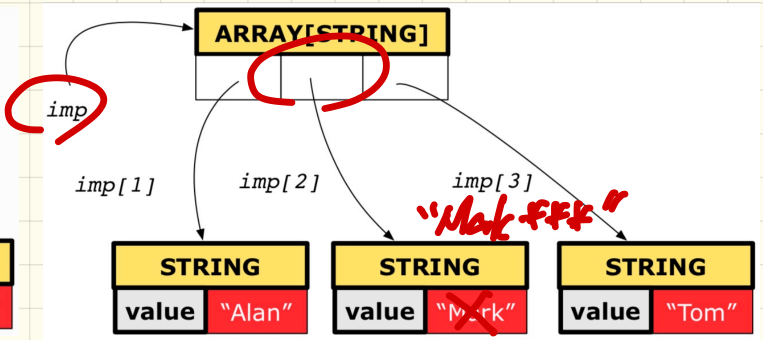


Collection Objects: Deep Copy & Make 2nd-Level Changes

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

F .





Contract View

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

old balance

old

pre-state

post-state

Runtime Contract Checks

call f

check precondition of f

cache old expressions

execute implementation of f

check postcondition of f

pre-state

post state

balance = old balance + 100

f

require

...

do

...

ensure

... **old** *expr* ...

end

before executing the imp.

old_expr := expr

a

a.twin

a.deep_twin

a

a.twin

a.deep_twin

Lecture 6

Wednesday January 22

Lab1: Due at 3pm this Friday

TA Office Hours: 12pm - 2pm LAS 2056

My office hours: 3pm to 5pm, Wednesday

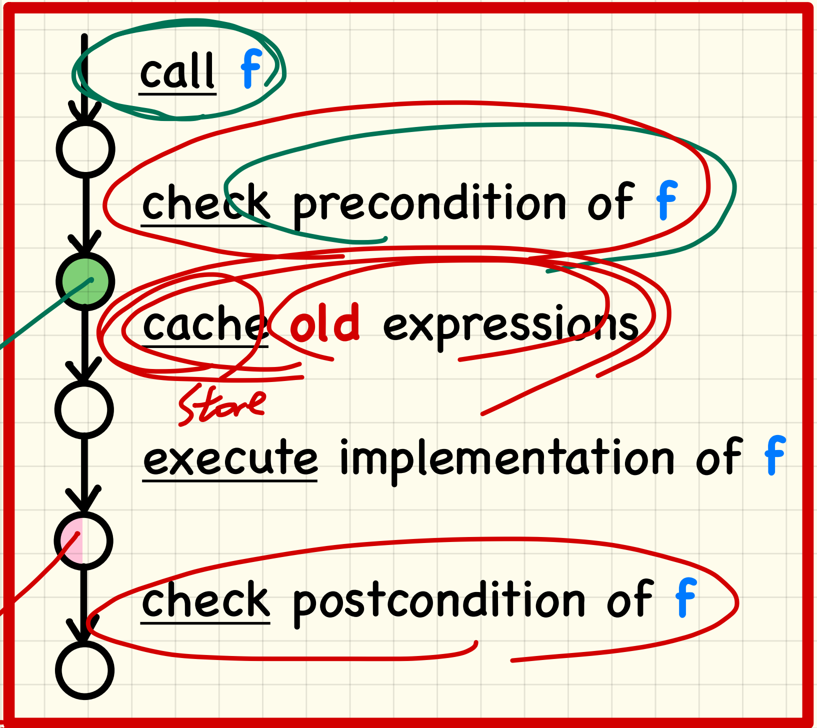
Extra office hours: 3:30pm to 5:30pm, Thursday

Contract View

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

body

Runtime Contract Checks



pre-state

post-state

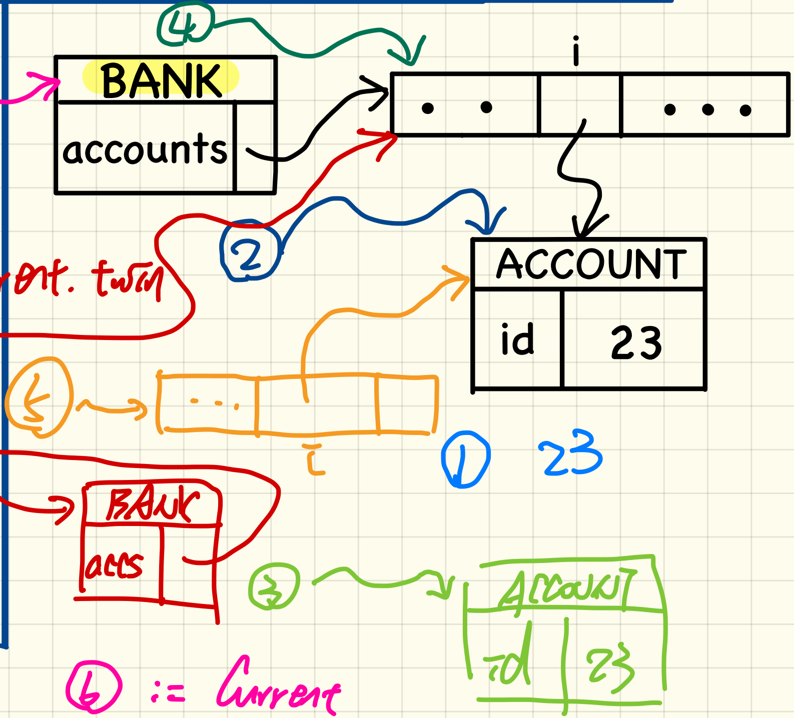
Caching Values for **old** Expressions in Postconditions

ensure (in context of BANK)

1. **old** accounts[i].id
2. (**old** accounts[i]).id
3. (**old** accounts[i].**twin**).id
4. (**old** accounts)[i].id
5. (**old** accounts.**twin**)[i].id
6. (**old** Current).accounts[i].id — this.
7. (**old** Current.**twin**).accounts[i].id

⑦ := Current.twin

How to cache at runtime?



② := accounts[i]

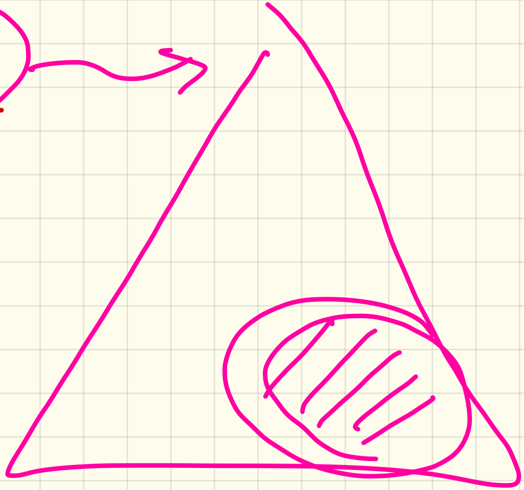
③ := accounts[i].twin — ACCOUNT

⑥ := Current

④ := accounts

⑤ := accounts.twin

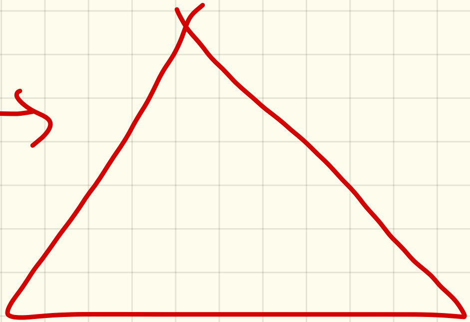
current



bst.insert(. .)

o_dt

h.



class BANK

accounts: ARRAY[ACCOUNT]

end

class ACCOUNT

id: INTEGER

end

accounts

accounts[i]

accounts[i].id

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

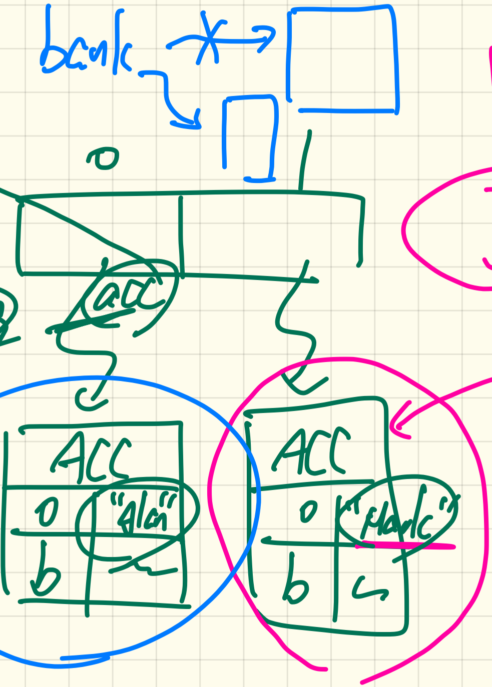
accounts.has(n)

A[C]   S

```
class
  ACCOUNT

inherit
  ANY
    redefine is_equal end

create
  make

feature -- Attributes
  owner: STRING
  balance: INTEGER

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

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

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

bank. ______

bank



∀ 2 ~ "3"

∃

return

→ cursor to account

acc.item

accounts

account_of (n : STRING)

as

require

across accounts is acc

some

[acc.name] n

and acc.owner ~

bank.account_of("Mark")

bank.account_of("Tom")

bank.account_of("Alan")

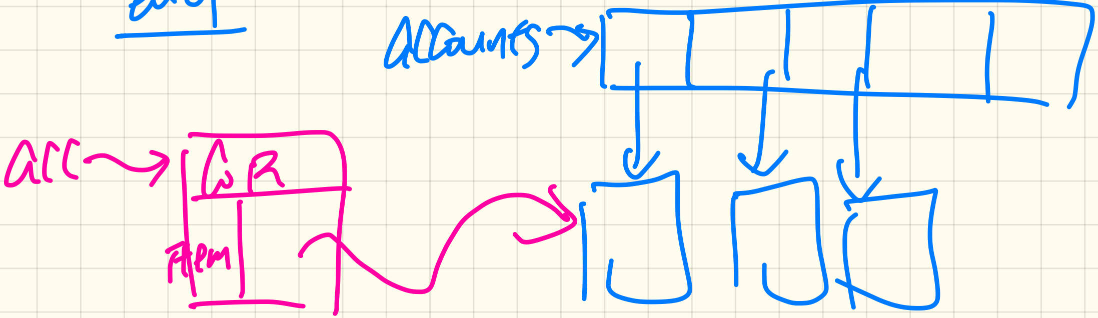
across accounts as acc

some

~~acc.owner~~ ~ n

acc.item.owner

end



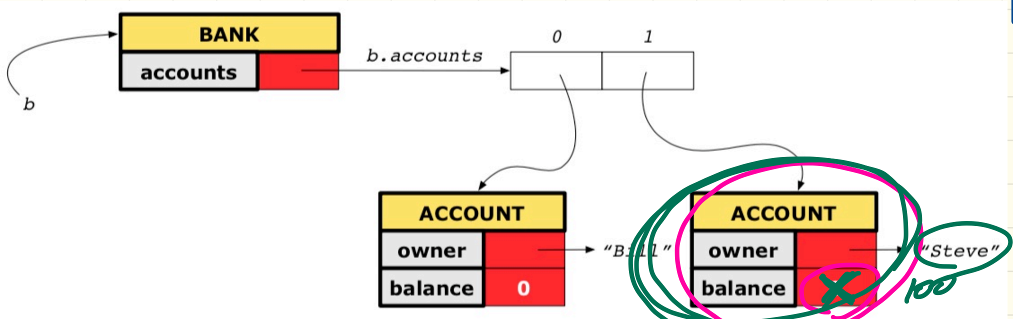
```
class Foo {

  m(..) {
    x this = ...

    } x current :=

}
```

Unit Test for All 5 Versions

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

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

Version 1: Incomplete Contracts, Correct Implementation

b.deposit("Steve", 100)



Steve 100

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

0

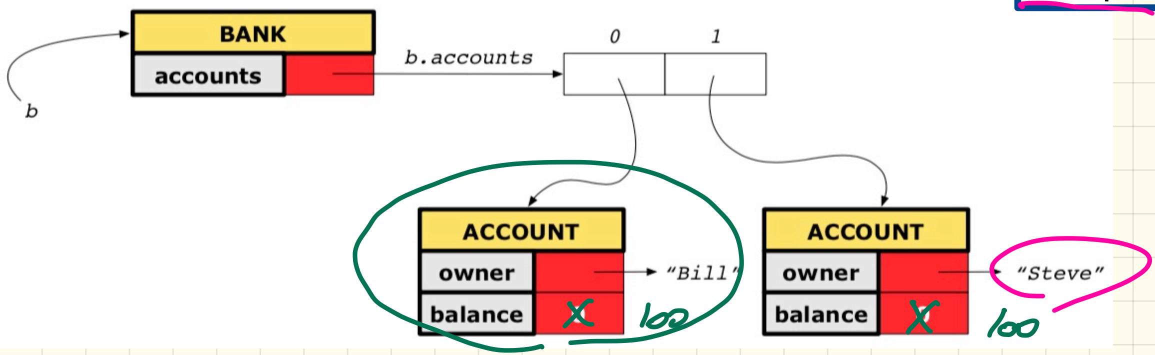
old_b := Current.account_of(n).balance

100 = 0 + 100

T

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)




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

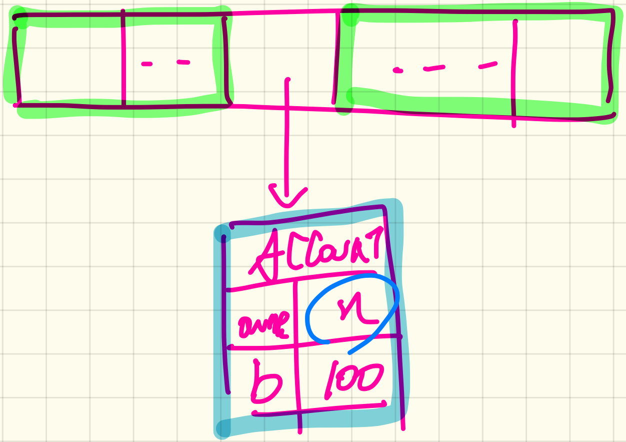
Steve

100

0

only concerns about owner n

accounts



b.deposit_on(n, 50)

How to specify "all accounts except the one with owner n have remained the same"

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

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
```

i

only exist in post-state

(old get)(j.item) ~~X~~

(old Current.deep_twin).get(j.item)

compiles

(old (Current)).get(j.item)

useless

c_v := get(j.item) → X j does not exist in pre-state

Hint: What value will be cached at runtime

before executing the implementation of update?

Lecture 7
Monday January 27

f (. . .)

require

. . .

do

correct, or incorrect

implementation

ensure

complete

postcondition

end

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

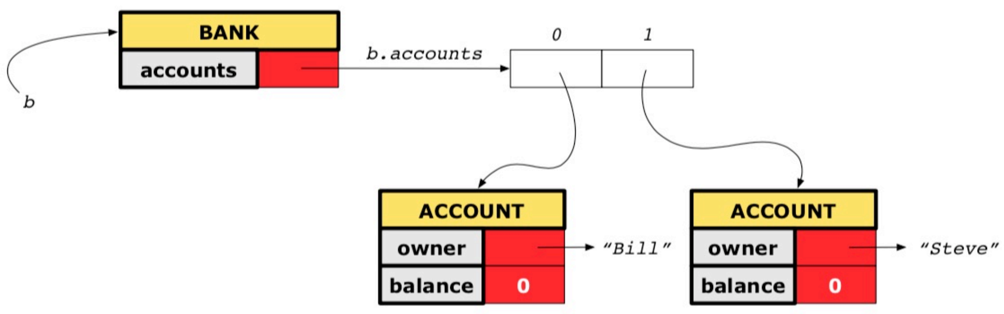
```
class
  ACCOUNT

inherit
  ANY
    redefine is_equal end

create
  make

feature -- Attributes
  owner: STRING
  balance: INTEGER

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

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

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

Unit Test for All 5 Versions

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

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

Version 1: Incomplete Contracts, Correct Implementation

b.deposit("Steve", 100)



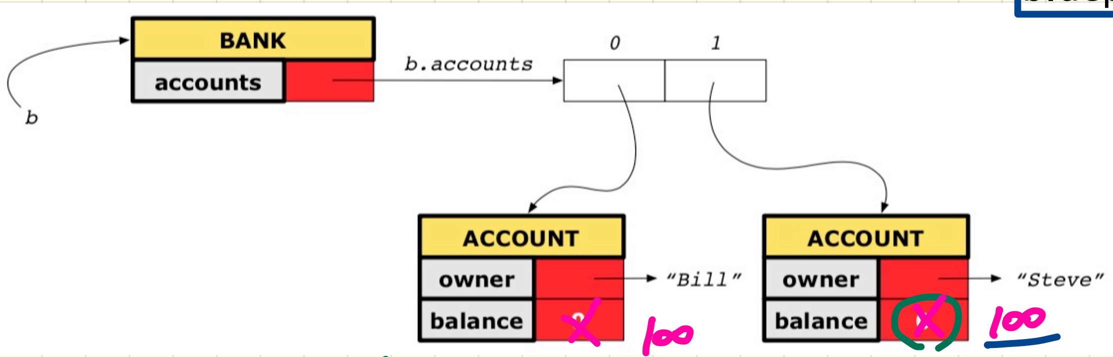
start

Correct

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

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)




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

Steve

100

old_value :=

0

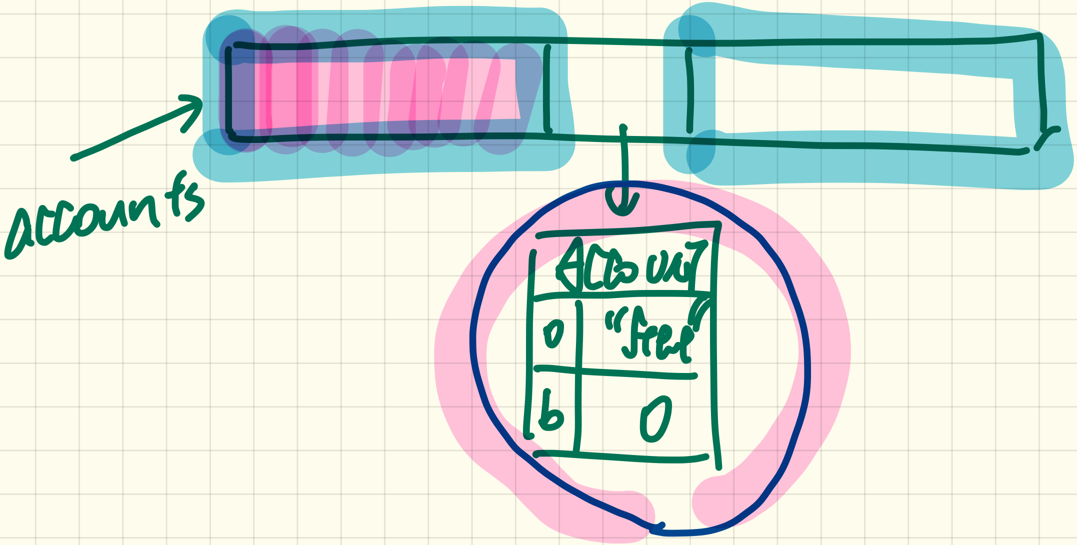
100 = 0 + 100

T

cache

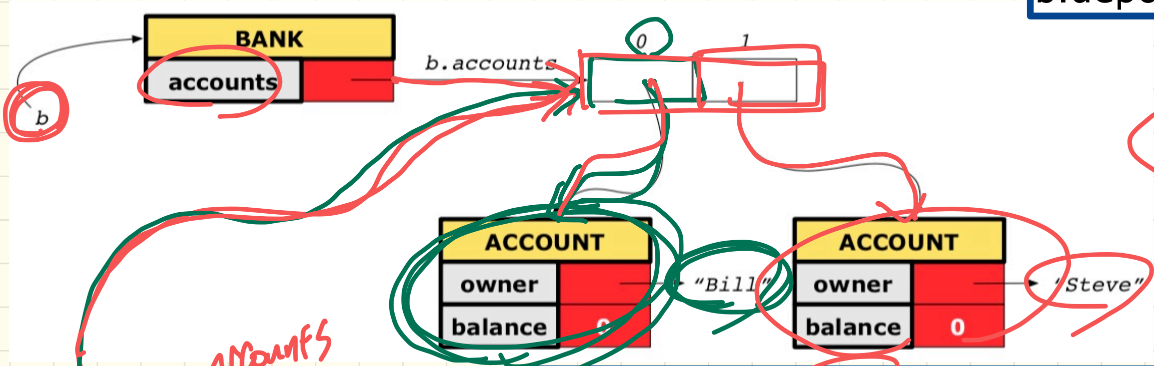
int

value


accounts
Account
0
"Fred"
b
0

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

b.deposit("Steve", 100)



F ⇒ _ ≡ T

old_accs := accounts

1st iter.

Bill /~ Steve ⇒ [ ] T

2nd iter

Steve /~ Steve ⇒ F T

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

steve

T

"old" version of account

post-state

"new" version of acc.

Bill

acc's owner is not the one to be changed

Use of across in Postcondition

Version 1

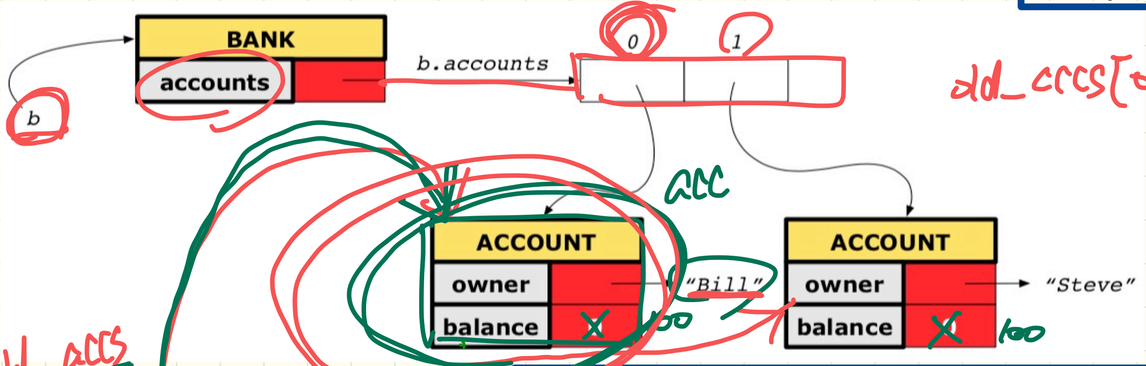
```
across old accounts is acc
all
   acc.owner /~ n
   implies
   acc ~ Current.account_of (acc.owner)
end
```

Version 2

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

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

b.deposit("Steve", 100)



old_accs[0] := b.accs[0]

old_accs

0 1

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

T

1st

Bill ∧ Stee ⇒ T

T

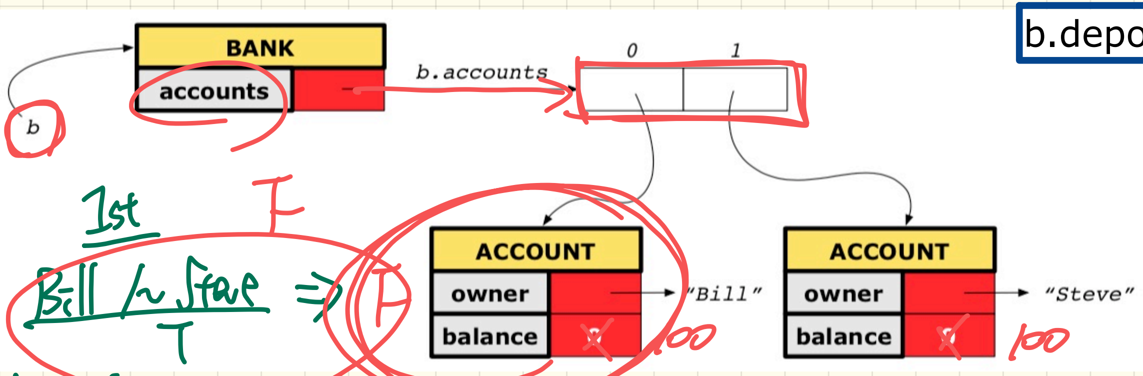
2nd

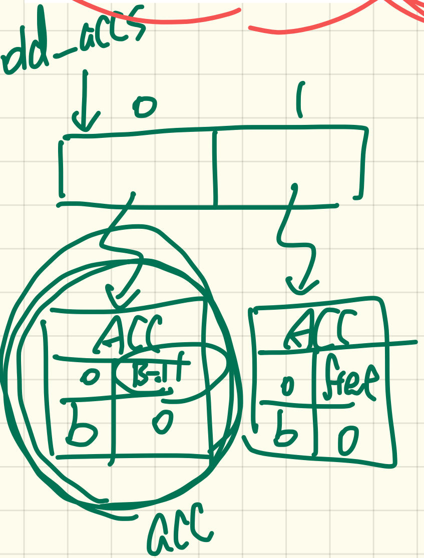
F ⇒ T

T

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

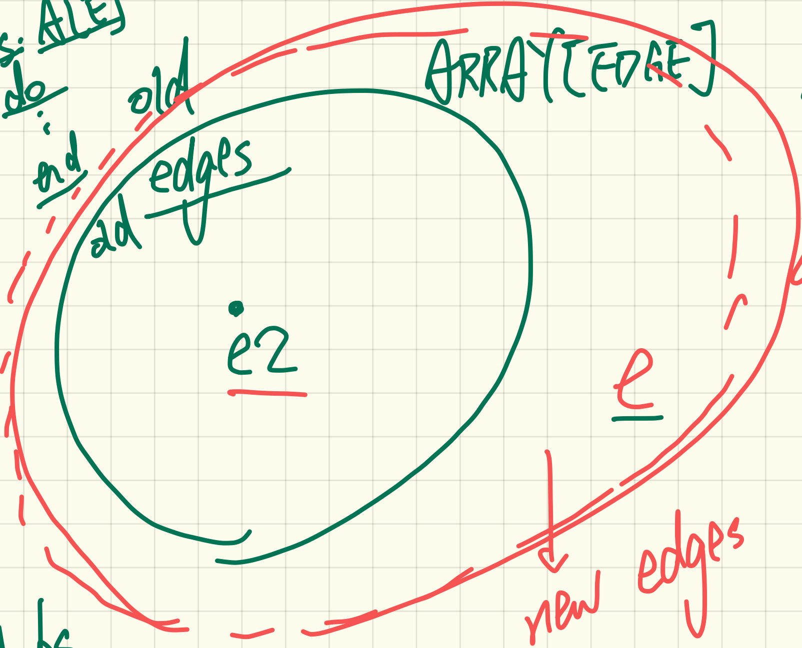
b.deposit("Steve", 100)






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

edges: A[E]
do
:
end



add_edge(e)

add_edge(e)
⋮
ensure
count_incremented:
edges.count = old edges.count + 1

~~edges do end~~

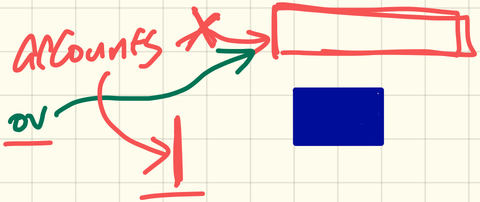
old edges ⊂ new edges

changed: ______

unchanged:

across ~~old~~ edges is e2
old edges.d-t
all
end → current.has_edge(e2)

Complete Postcondition: Exercise



[redacted]

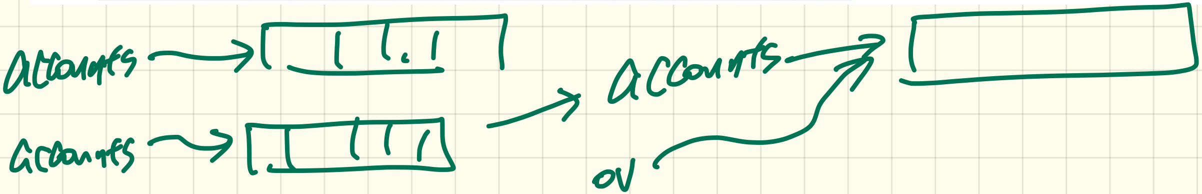
Consider the query *account_of (n: STRING)* of *BANK*.

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

- ✓ `accounts = old accounts` T
- `accounts = old accounts.twin` [redacted]
- `accounts = old accounts.deep_twin` [redacted]
- `accounts ~ old accounts` [redacted]
- `accounts ~ old accounts.twin` [redacted]
- `accounts ~ old accounts.deep_twin` [redacted]

do
[create accounts.make_empty]
old

[×]



Information Hiding

Supplier

client

top
public
push
pop $_{imp.}$
count

~~ARRAY[]~~
LL

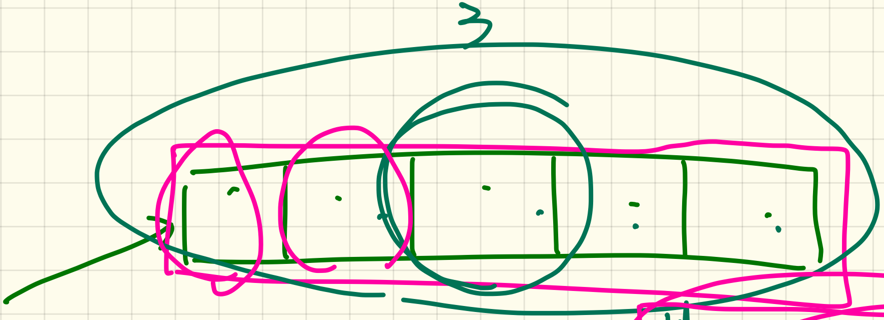
private

① client gets access to imp (not good).

→ change on interface

part of the specification of top, accessed by clients.

top: G
do
…
ensure
Result ~ ~~imp[count]~~ imp[i]



imp

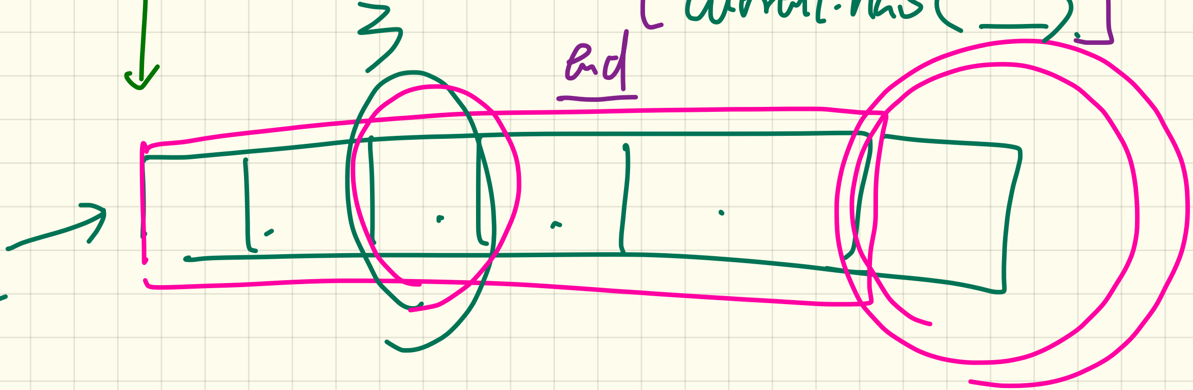
old

push(_)

across old imp.deep_twin as a

all [current.has(_)]

end



imp

tmp → | alan | mark | tom |

push(jim)

across old tmp.deep_twin is a

all

[Current.has( )]

end

| mark | alan | tom | jim |

not good enough.

```
unchanged: across 1 |..| count - 1 as i all
           imp[i.item] ~ (old imp.deep_twin)[i.item] end
```

v1 Stack.

v2 old [imp.deep_twin[i.item]]

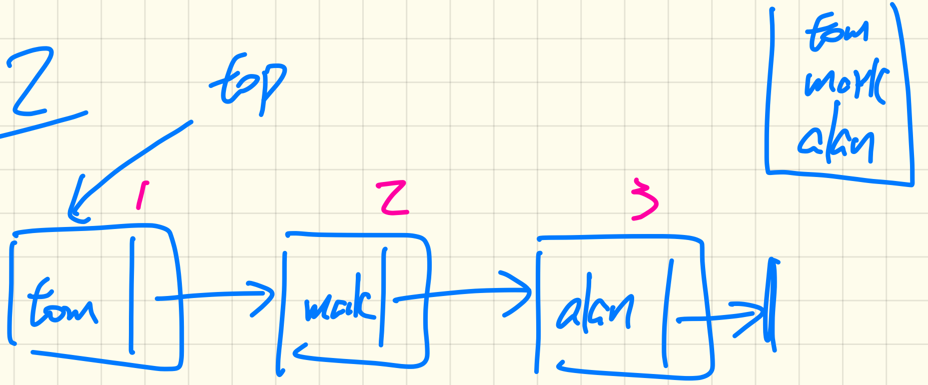
to be cached at pre-state

count
do
Result := imp.count
end

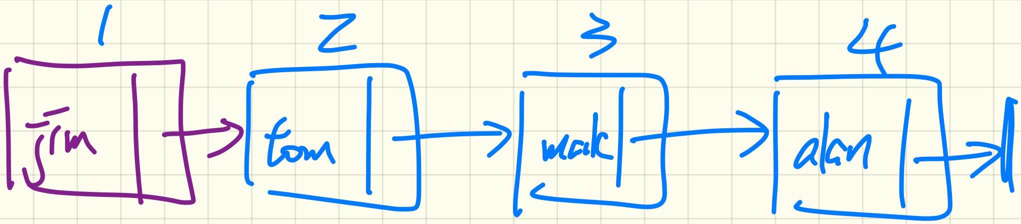
invariant.

imp.count = count

~~Strategy 2~~



push(jim)



Lecture 8
Wednesday January 29

class STACK[G]

imp: ~~ARRAY~~ LL [G] -- ~~end~~ front of ~~a~~ LL is the top

top : G

ensure

→ Result ~ imp[~~count~~ 1]

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
```

```
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 Stack

imp : ~~X~~

model :

do - - - - ensure end

imp_

push

ensure

model

pop

ensure

model

no mention

of imp_

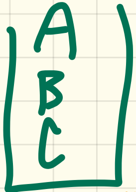
Strategy 2

LL front of LL

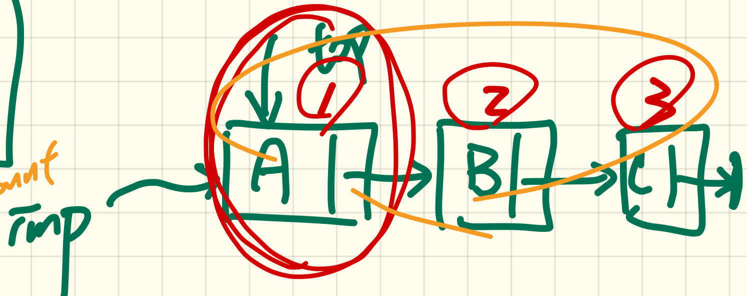
top

o_v := tmp.count

3



across | |..| tmp.count is i



across | |..| old tmp.count is i

all (old tmp.at [i] ~ tmp [i+1]

end

push(D)

push (g)

do

...

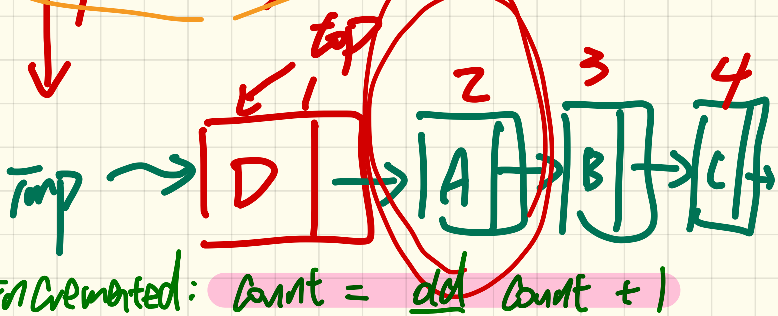
ensure

size_incremented: count = old count + 1

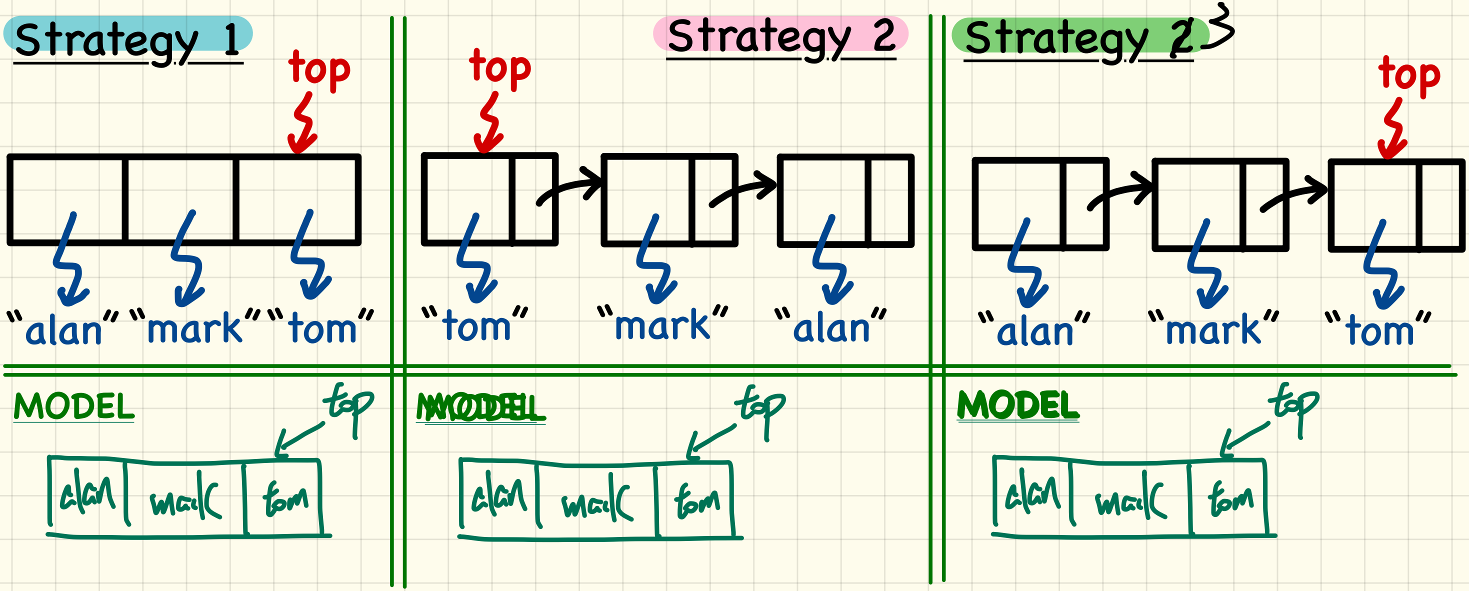
changed: tmp[i] ~ g

$\boxed{\text{unchanged:}}$ ① ~~across | 1..| tmp.count~~

② across | |..| old tmp.count



Implementing a LIFO Stack



Using MATHMODELS Library

Implementing an Abstraction Function

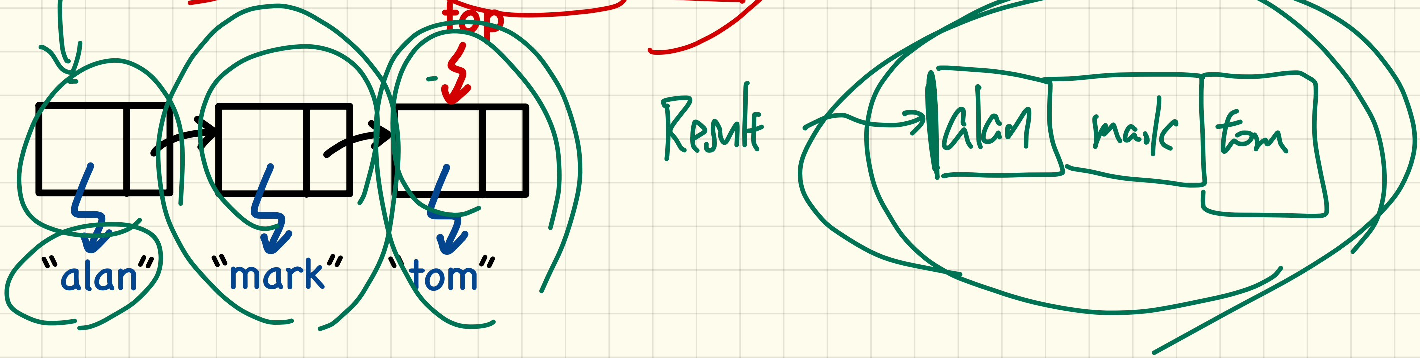
Strategy 3

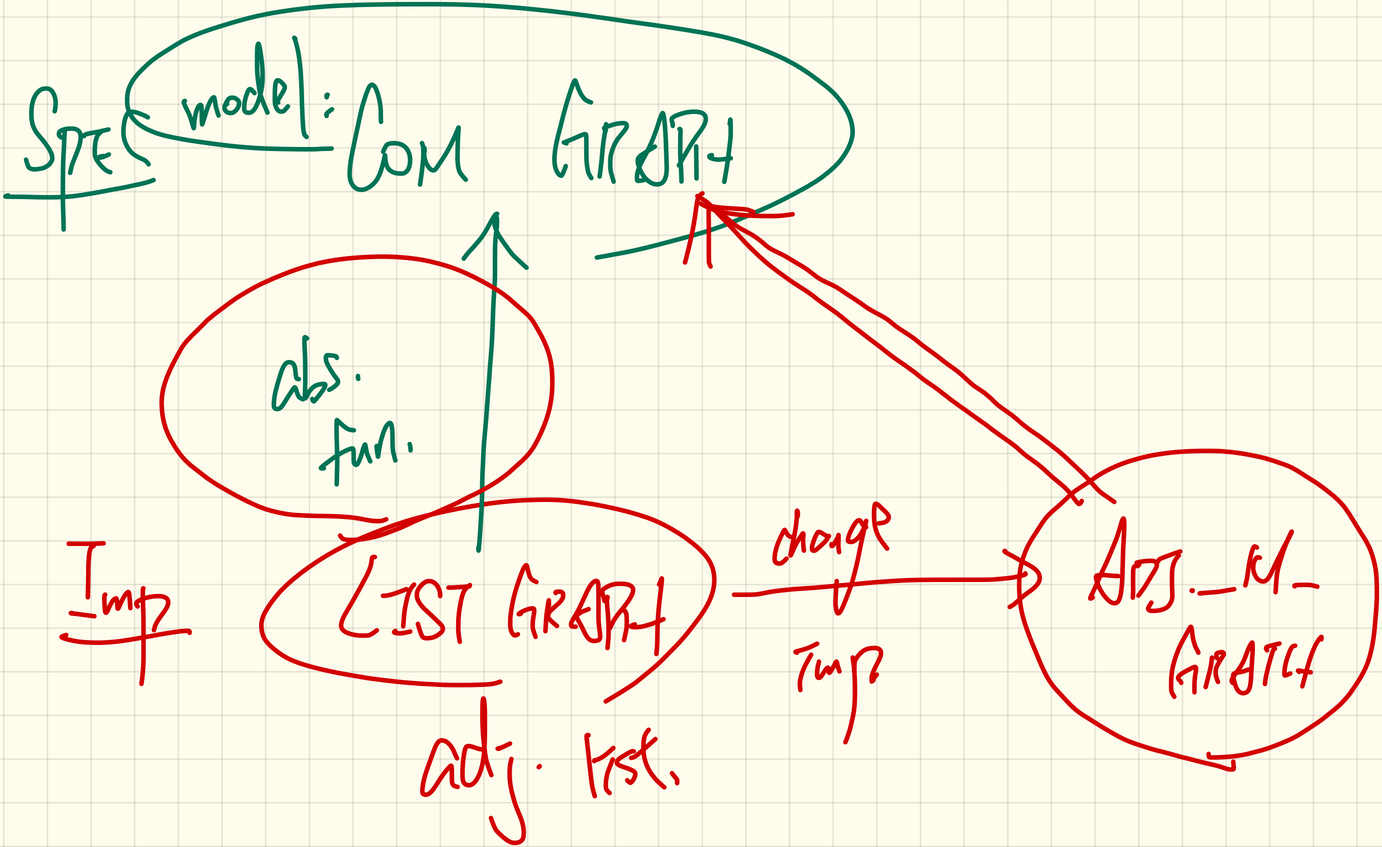
```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.append(cursor.item) end
    end
```

end of CC .

Exercise 1: Write postcondition of model.

Exercise 2: What if Strategy 2 was adopted? Change what?




Spec
model: COM GRAPH
abs. fun.
Imp
LIST GRAPH
adj. list.
change
Imp
ADJ._M_ GRAPH

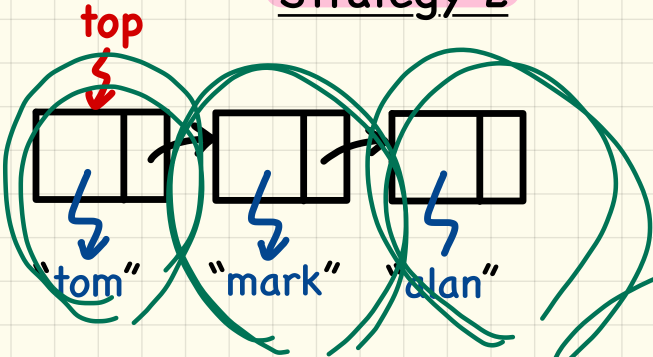
```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
 imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
 model: SEQ[G]
   do create Result.make_empty
      across imp as cursor loop Result.append(cursor.item) end
   end
```

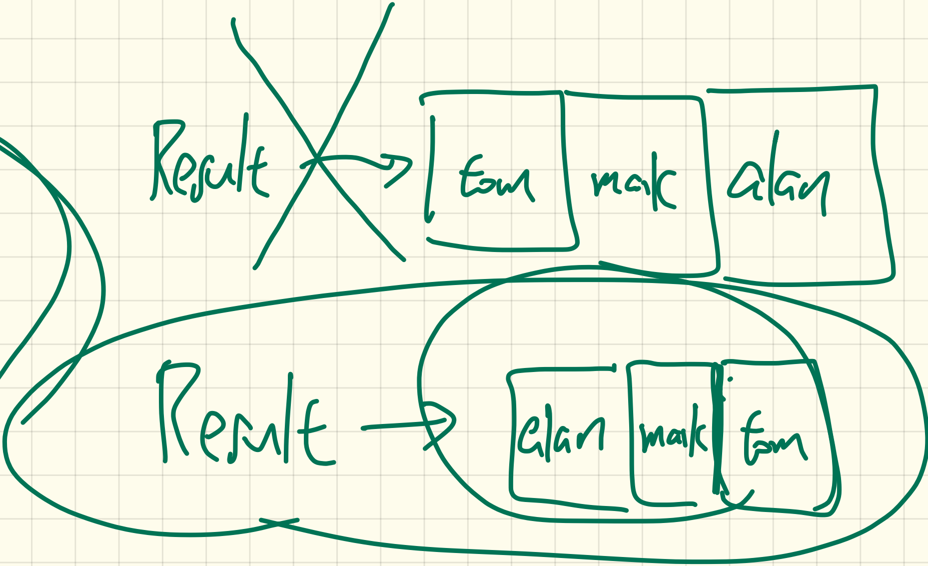
front is the top.

Strategy 2

~~append~~ prepend

Strategy 2





Using MATHMODELS Library

Writing Contracts using the Abstraction Function

```
class LIFO_STACK[G -> attached ANY] create make
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
feature -- Commands
  push (g: G)
    ensure model ~ (old model.deep_twin).appended(g) end
```

a separate call to model in the pre-state.

Question: Can clients tell which strategy is being adopted?

No ∵ no mention of imp.

Exercise: What if strategy was changed? Change what?

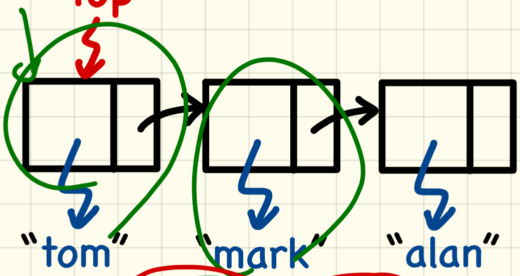
one call to model in the post-state

Checking MATHMODELS Contracts at Runtime

Strategy 2

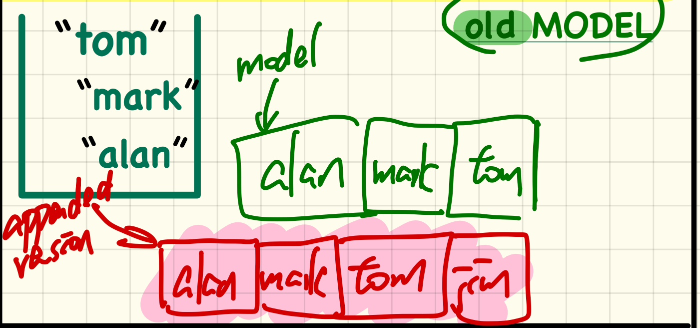
Pre-State

old IMPLEMENTATION



immutable query

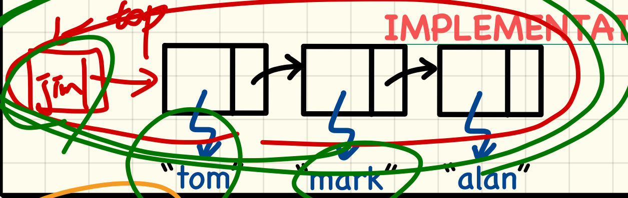
old MODEL



s.push("Jim")

Post-State

IMPLEMENTATION



old model ok

MODEL


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

push (g: G)

ensure **model** ~ (**old** **model**.*deep_twin*).~~*appended*~~ append (g) **end**

class SEQ[G]

append (g: G) → implement abstraction function

appended (g: G) : SEQ[G]

write ~~abstraction~~ contract.

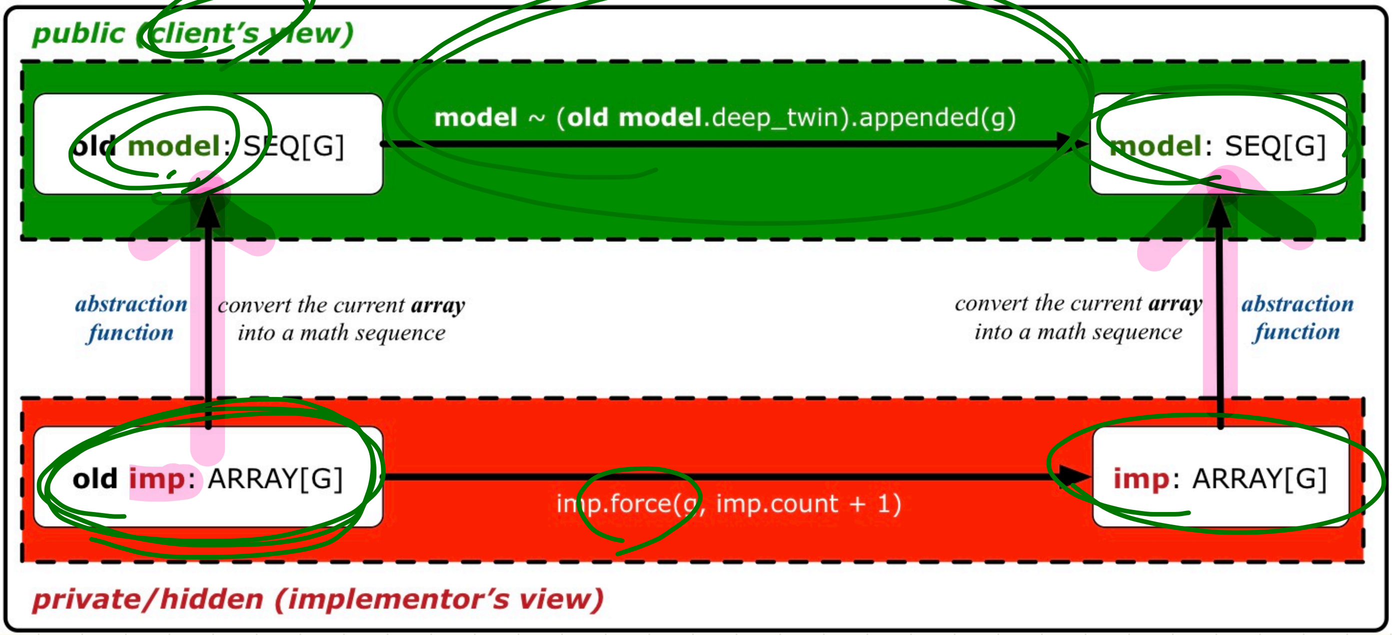
String s = . . —

s.substring(______ , ______)

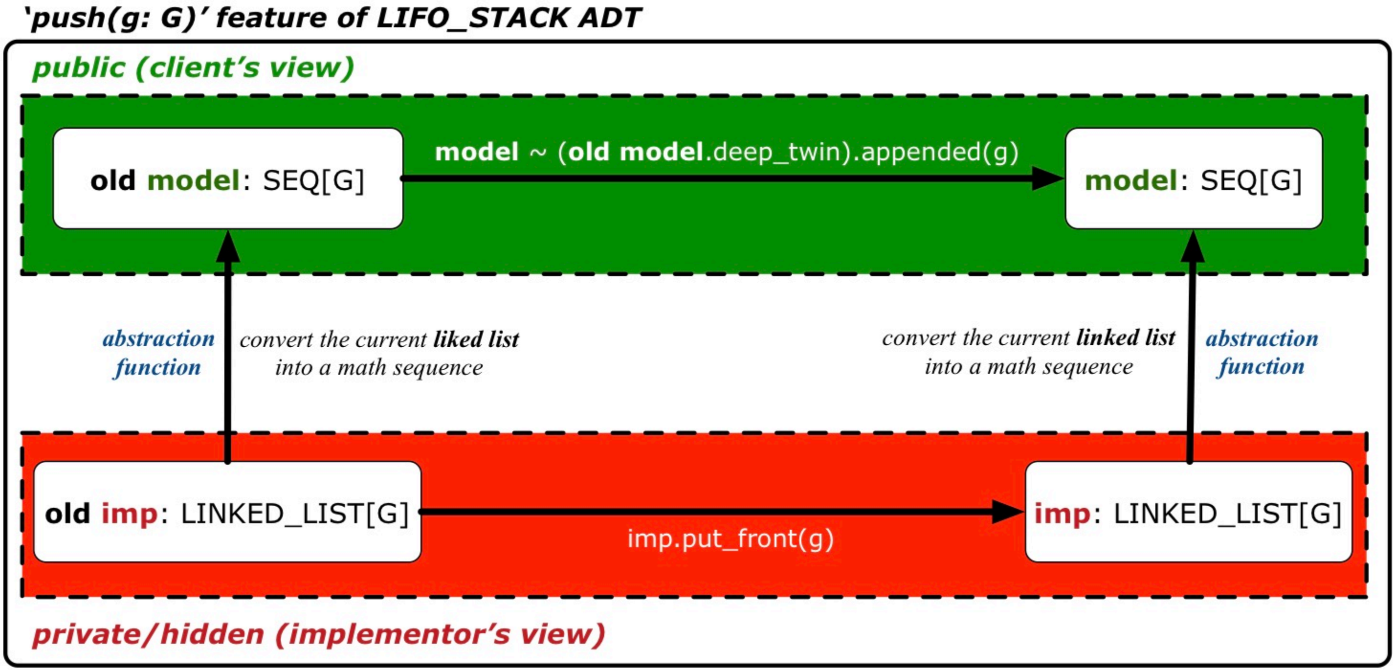
"Jim".substring(. .) → "im"

Strategy 1: Mathematical Abstraction

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



Strategy 2: Mathematical Abstraction



Use of MATHMODELS:

Single-Choice Principle

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 1
  imp: ARRAY[G] end
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
```

Testing REL in MATHMODELS

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

- $\boxed{\text{r.}\textit{domain}}$: set of first-elements from r
  - r.**domain** $= \{\ d \mid (d,r) \in r\ \}$
  - e.g., r.**domain** $= \{a,b,c,d,e,f\}$
- $\boxed{\text{r.}\textit{range}}$: set of second-elements from r
  - r.**range** $= \{\ r \mid (d,r) \in r\ \}$
  - e.g., r.**range** $= \{1,2,3,4,5,6\}$
- $\boxed{\text{r.}\textit{inverse}}$: a relation like r except elements are in reverse order
  - r.**inverse** $= \{\ (r,d) \mid (d,r) \in r\ \}$
  - e.g., r.**inverse** $= \{(1,a),(2,b),(3,c),(4,a),(5,b),(6,c),(1,d),(2,e),(3,f)\}$
- $\boxed{\text{r.}\textit{domain\_restricted}\text{(ds)}}$: sub-relation of r with domain ds.
  - r.**domain_restricted**(ds) $= \{\ (d,r) \mid (d,r) \in r \land d \in ds\ \}$
  - e.g., r.**domain_restricted**({a, b}) $= \{(\mathbf{a},1),(\mathbf{b},2),(\mathbf{a},4),(\mathbf{b},5)\}$
- $\boxed{\text{r.}\textit{domain\_subtracted}\text{(ds)}}$: sub-relation of r with domain <u>not</u> ds.
  - r.**domain_subtracted**(ds) $= \{\ (d,r) \mid (d,r) \in r \land d \notin ds\ \}$
  - e.g., r.**domain_subtracted**({a, b}) $= \{(\mathbf{c},6),(\mathbf{d},1),(\mathbf{e},2),(\mathbf{f},3)\}$
- $\boxed{\text{r.}\textit{range\_restricted}\text{(rs)}}$: sub-relation of r with range rs.
  - r.**range_restricted**(rs) $= \{\ (d,r) \mid (d,r) \in r \land r \in rs\ \}$
  - e.g., r.**range_restricted**({1, 2}) $= \{(a,\mathbf{1}),(b,\mathbf{2}),(d,\mathbf{1}),(e,\mathbf{2})\}$
- $\boxed{\text{r.}\textit{range\_subtracted}\text{(ds)}}$: sub-relation of r with range <u>not</u> ds.
  - r.**range_subtracted**(rs) $= \{\ (d,r) \mid (d,r) \in r \land r \notin rs\ \}$
  - e.g., r.**range_subtracted**({1, 2}) $= \{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

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

```
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
```

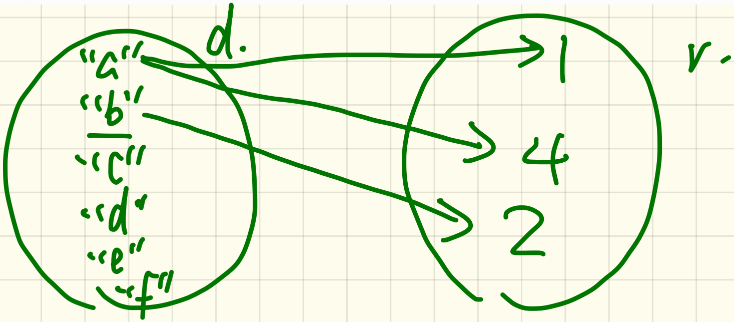
```
test_rel: BOOLEAN
  local
    r, t: REL[STRING, INTEGER]
    ds: SET[STRING]
  do
    create r.make_from_tuple_array (
      <<["a", 1], ["b", 2], ["c", 3],
        ["a", 4], ["b", 5], ["c", 6],
        ["d", 1], ["e", 2], ["f", 3]>>)
    create ds.make_from_array (<<"a">>)
    -- r is not changed 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 →

"a"

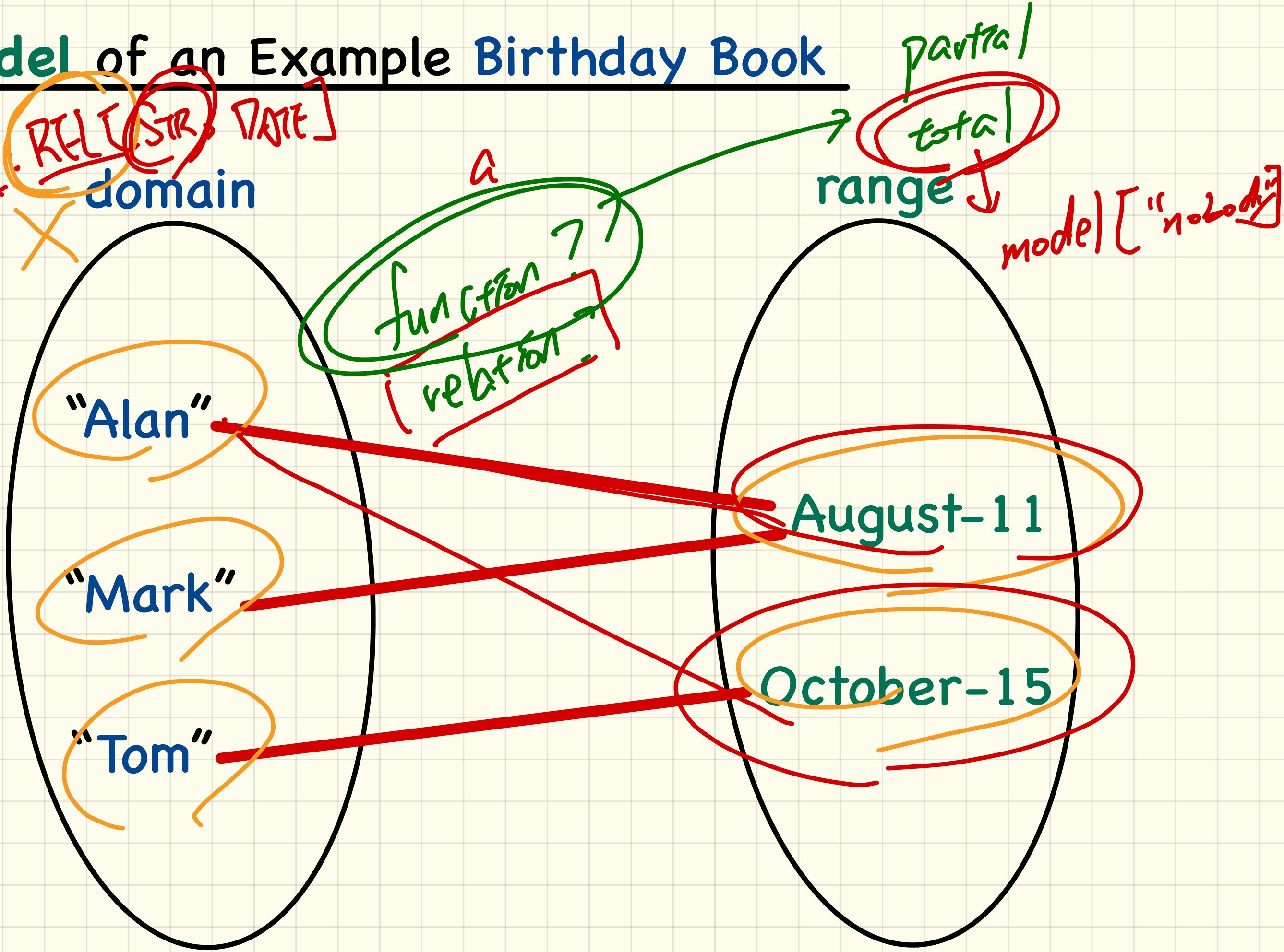
domain_restricted

t → << ~~a 1~~, b 2, c 3,
~~a 4~~, b 5, c 6,
d 1, e 2, f 3 >>



Model of an Example Birthday Book

model : REL[STR, DATE]



LECTURE 9

MONDAY FEBRUARY 3

- Labtest 1:
  * Birthday Book
  * MATHMODELS
  * Iterator Patterns: Two Tutorial Series

REL FUN PAIR

t

key	value
alan	1
~~mark~~	~~2~~
tom	3
jim	4
mark	7

1. domain subtraction on "mark"

2. union the map with entry ("mark", 7)

① t.put("jim", 4)

not an existing key

② t.put("mark", 7)

does not require the key to exist

an existing key

Testing REL in MATHMODELS

REL : override and return new relation

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

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

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

(a,3), (c,4)

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

- $\boxed{\text{r.}\textit{domain}}$: set of first-elements from r
  - r.**domain** $= \{ d \mid (d,r) \in r \}$
  - e.g., r.**domain** $= \{a,b,c,d,e,f\}$
- $\boxed{\text{r.}\textit{range}}$: set of second-elements from r
  - r.**range** $= \{ r \mid (d,r) \in r \}$
  - e.g., r.**range** $= \{1,2,3,4,5,6\}$
- $\boxed{\text{r.}\textit{inverse}}$: a relation like r except elements are in reverse order
  - r.**inverse** $= \{ (r,d) \mid (d,r) \in r \}$
  - e.g., r.**inverse** $= \{(1,a),(2,b),(3,c),(4,a),(5,b),(6,c),(1,d),(2,e),(3,f)\}$
- $\boxed{\text{r.}\textit{domain\_restricted}\text{(ds)}}$: 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}) $= \{(\mathbf{a},1),(\mathbf{b},2),(\mathbf{a},4),(\mathbf{b},5)\}$
- $\boxed{\text{r.}\textit{domain\_subtracted}\text{(ds)}}$: sub-relation of r with domain <u>not</u> ds.
  - r.**domain_subtracted**(ds) $= \{ (d,r) \mid (d,r) \in r \wedge d \notin ds \}$
  - e.g., r.**domain_subtracted**({a, b}) $= \{(\mathbf{c},6),(\mathbf{d},1),(\mathbf{e},2),(\mathbf{f},3)\}$
- $\boxed{\text{r.}\textit{range\_restricted}\text{(rs)}}$: 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}) $= \{(a,\mathbf{1}),(b,\mathbf{2}),(d,\mathbf{1}),(e,\mathbf{2})\}$
- $\boxed{\text{r.}\textit{range\_subtracted}\text{(ds)}}$: sub-relation of r with range <u>not</u> ds.
  - r.**range_subtracted**(rs) $= \{ (d,r) \mid (d,r) \in r \wedge r \notin rs \}$
  - e.g., r.**range_subtracted**({1, 2}) $= \{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

```
test_rel: BOOLEAN
  local
    r, t: REL[STRING, INTEGER]
    ds: SET[STRING]
  do                 ["a", 3]
    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
```

override (s : SET<PAIR ... >)

overridden (s : SET < PAIR ... >) : REL[G, H]

r. override (s) → command
does not return
↳ not to be used
in contract

r. overridden (s) . domain

r.overridden(<<["a", 100], ["a", 200]>>)

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

$(a, 100)$ $(a, 200)$

relation

vs.

function

$$
\begin{tikzcd}
 & r1 \\
d_1 \arrow[ur] \arrow[dr] & \\
 & r2
\end{tikzcd}
$$

$$d_1 \longrightarrow r1$$

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

across | 1..| a.count is i all

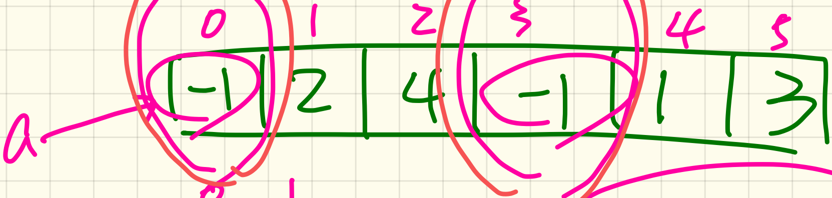
Boolean ← across |1..| a.count is j all

[i+1

]

end

end



0 1

0 2

0 3

⋮

distinct locations implies distinct values

$0 \neq 3$

$a[0] = a[3]$

Writing Postcondition: Exercise

all_positive_values (a: **ARRAY**[**INTEGER**]): **ARRAY**[**INTEGER**]

require

no_duplicates: ??

ensure

across Result is x

all

x > 0

end

all_p_v(<<-1, -7, 2, 1, 10>>)

↳ << 2, 1, 10 >>

Incomplete.

a → | -2 | -7 | 2 | 1 | 10 |

Result → | 101 | 202 | 303 |

wrong Imp. but postcond. evaluates to T.

all_positive_values (a: **ARRAY**[**INTEGER**]): **ARRAY**[**INTEGER**]

require

no_duplicates: ??

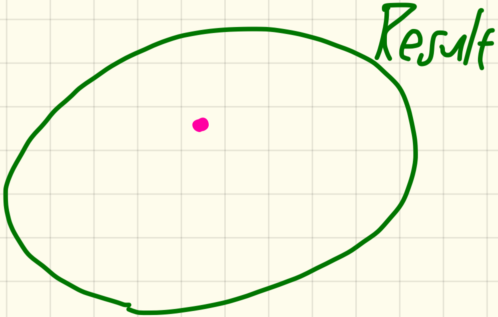
ensure

~~**across Result is** x~~

~~**all**~~

~~x > 0~~

~~**end**~~



post_1: all_pos_in_a_also_in_result :

across a is n all

if n > 0 then

Result.has(n)

else

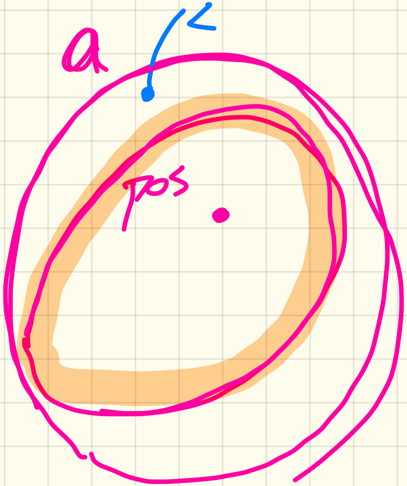
~~not Result.has(n)~~ True

end

n > 0

implies

R.has(n)

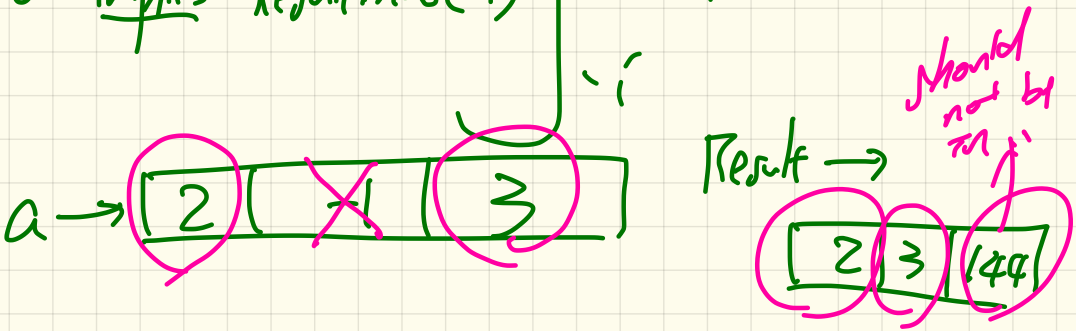


```
all_positive_values (a: ARRAY[INTEGER]): ARRAY[INTEGER]
    require
        no_duplicates: ??
    ensure
        across Result is x
            all
                x > 0
            end
```

assume 'a' is not modified

```
across a is n all
    n > 0 implies Result.has(n)
end
```

incomplete



all pos. #s in a

$$\boxed{S} = T \longrightarrow \text{Result.}$$

$$\equiv \boxed{S \subseteq T} \wedge \boxed{T \subseteq S}$$

each pos. # in a is also in Result

each # in Result is also in a. pos.

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

Witness

a → [-1 | 2]

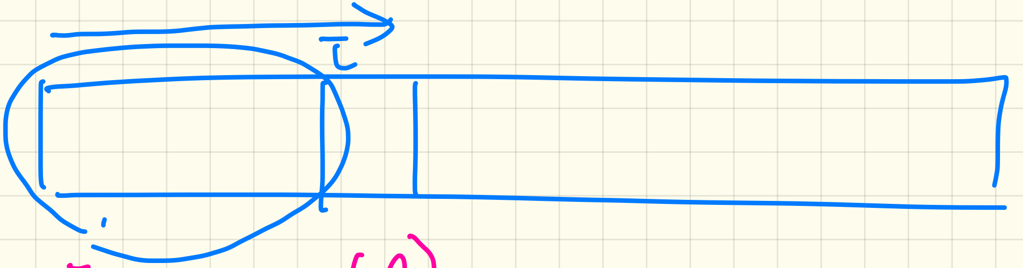
Result → [2 | 2]

all_pos_in_a_in_result : across a is x all
  x > 0 implies Result.has(x)
end

all_x_in_result_in_a : across Result is x all
  x > 0 and a.has(x)
end.

not complete

resolution : $\boxed{\text{no\_duplicates\_in\_result}}$: ??



{ARRAY}

occurrance(g)

require
↳ attributes
↳ queries
↳ ✗ local variables
↳ ✗ old

ensure
↳ attributes
↳ queries
↳ ✗ local variable
↳ ✓ old -

tmp

f
require
local
do
⋮
ensure
end

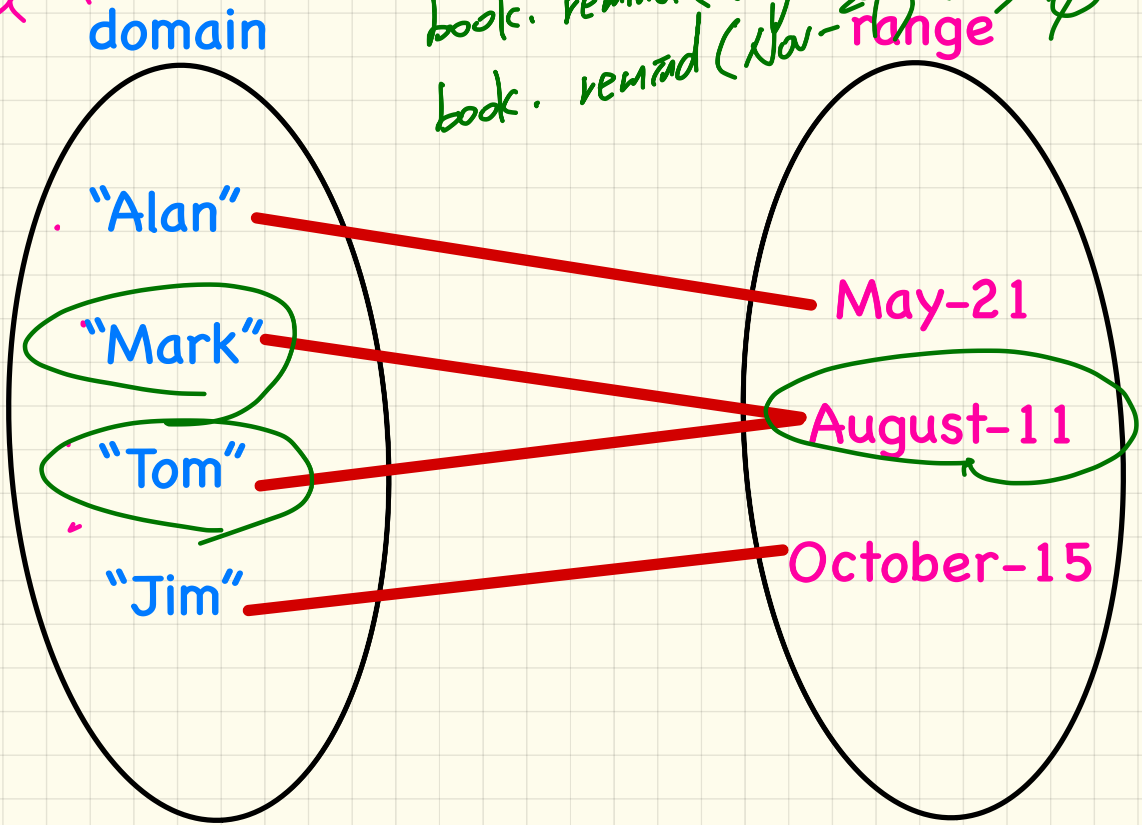
spec.

Model of an Example Birthday Book

Count 4

book.remind(August-11) → Mark, Tom

book.remind(Nov-29) → ∅



Birthday Book: Design

client

D

BIRTHDAY_BOOK

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

supplier

count: INTEGER
  -- number of entries

put(n: NAME; d: BIRTHDAY)
  ensure
    model_operation: [redacted])
    -- infix symbol for override operator: @<+

remind(d: BIRTHDAY): ARRAY[NAME]
  ensure
    nothing_changed: [redacted])
    same_counts: [redacted]t
    same_contents: [redacted]t
    -- infix symbol for range restriction: model @> (d)

invariant:
  consistent_book_and_model_counts: count = model.count

FUN[NAME,

model: FUN[NAME, ..]

supplier

BIRTHDAY

day: INTEGER
month: INTEGER

invariant
  1 ≤ month ≤ 12
  1 ≤ day ≤ 31

"@#_" vs. STRING

remind: ARRAY[..]

remind: ..

NAME

NAME

item: STRING

invariant
  item[1] ∈ A..Z

remind: ARRAY[..]

Lecture 10
Wednesday February 5

- **Labtest 1**:
  * Birthday Book
  * MATHMODELS
  * **Iterator Patterns**: Two Tutorial Series

Birthday Book: Design

BIRTHDAY_BOOK

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

count: INTEGER ✓ +
  -- number of entries

no precond.

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

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

invariant:
  consistent_book_and_model_counts: count = model.count

BB model

model: FUN[NAME, ..]

BIRTHDAY

day: INTEGER
month: INTEGER

invariant
  1 ≤ month ≤ 12
  1 ≤ day ≤ 31

remind: ARRAY[..]

NAME

item: STRING

invariant
  item[1] ∈ A..Z

S⊆T

TCS

|T| = |S|

put(NAME, ..)

put(.., BD)

BD

NAME

BON design diagram

is an "abstraction" of

your system : details

\# only relevant are shown

Concrete

(code)

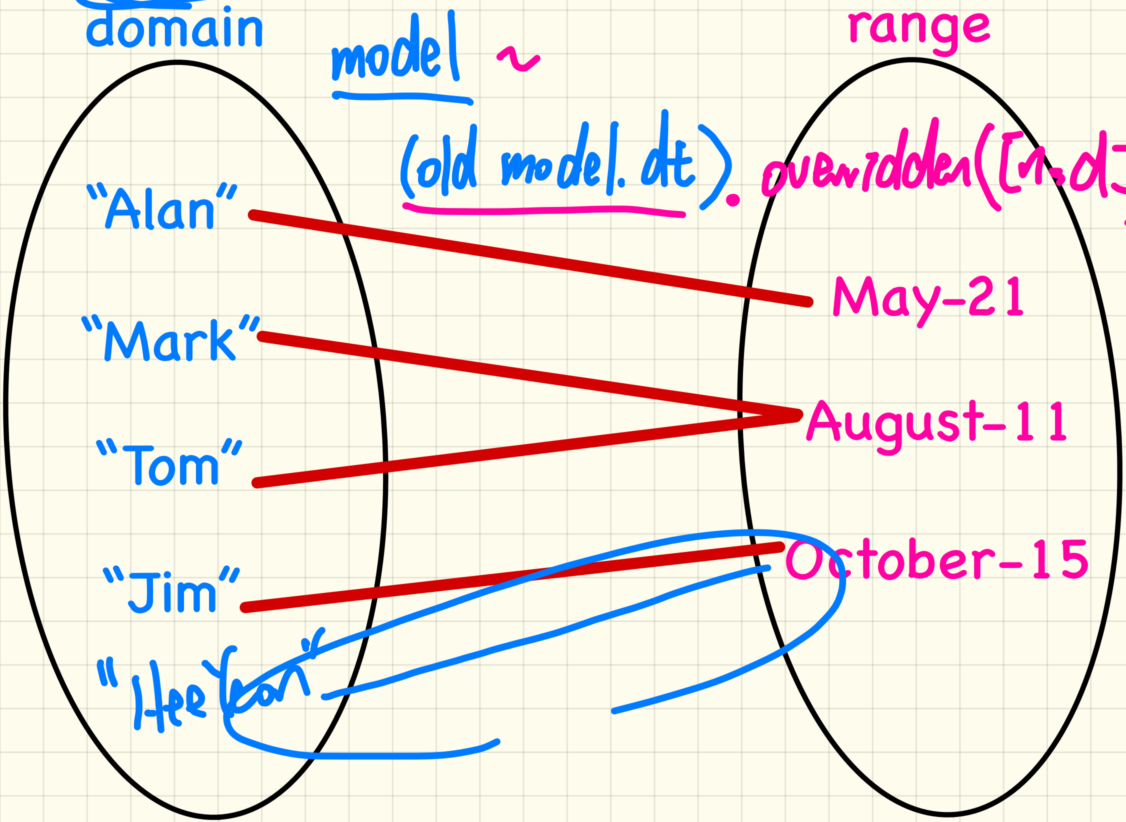
Birthday Book: Model Operation (1.1)

book.put("Heeyeon", October-15)

override

model ~

(old model.dt).overridden([n:d])



model ~ (old model l.d t) . overridden([n,d])

model ~ (old model l.d t) @<+ [n,d]

$\oplus$

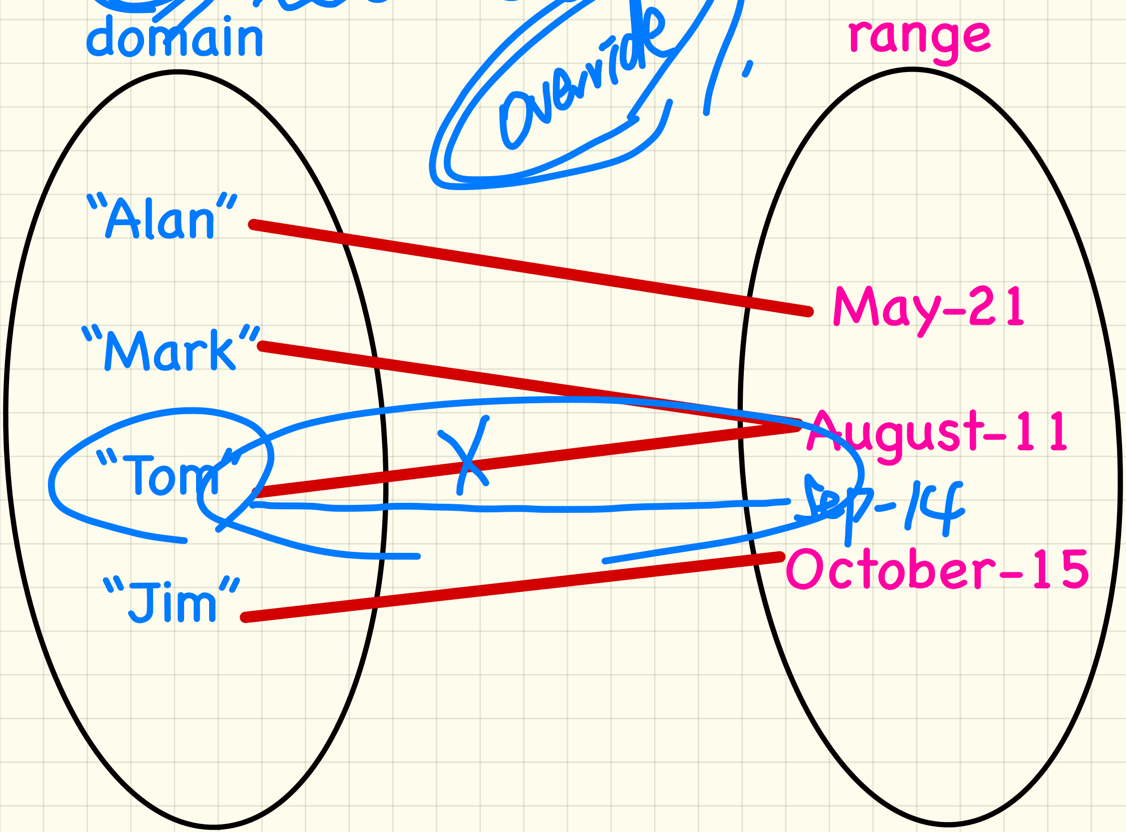
FUN

↳ extend cmd.

extended gv.

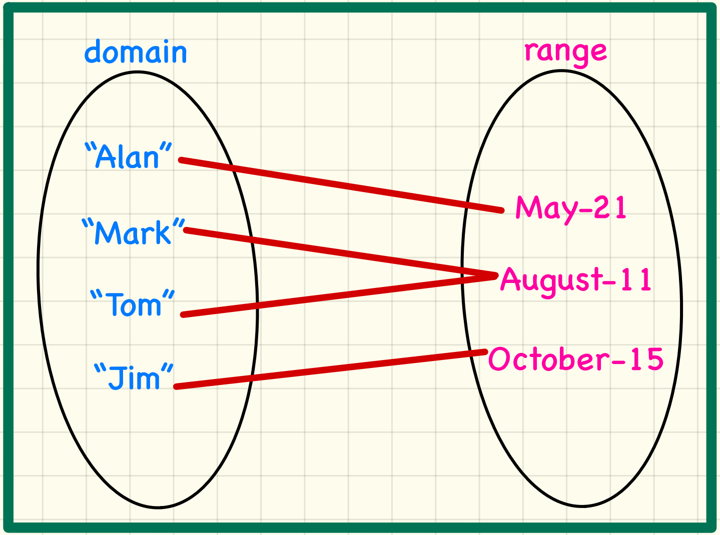
Birthday Book: Model Operation (1.2)

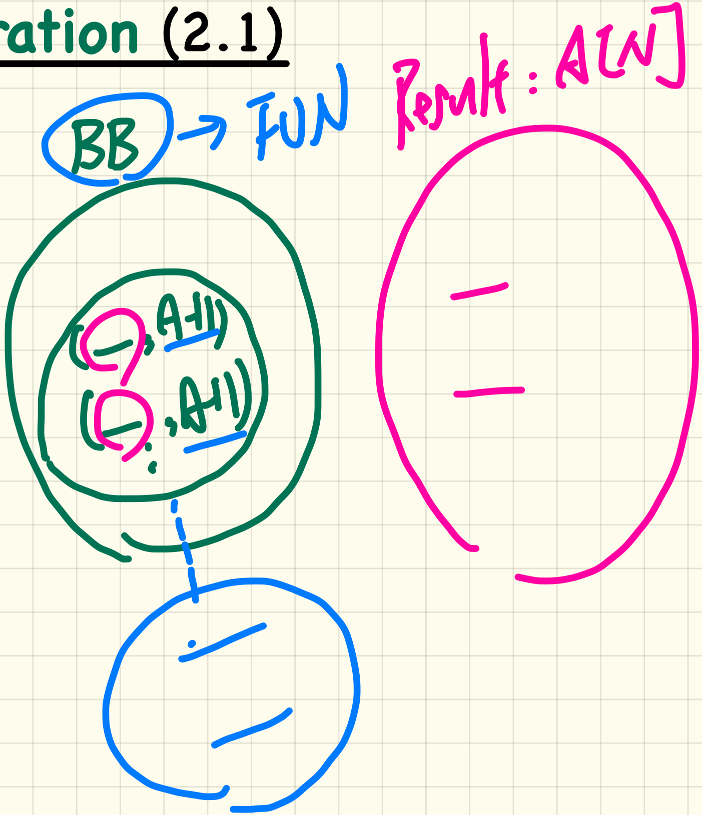
book.put("Tom", September-14)



Birthday Book: Model Operation (2.1)

book.remind(August-11)

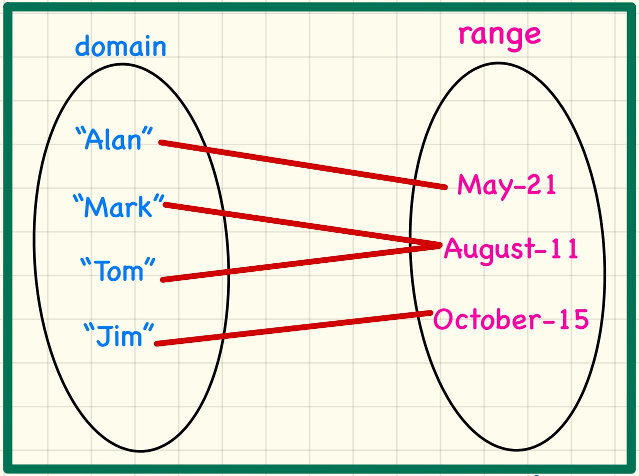




book.remind(August-11)

FUN[NAME, BD]

d. r.



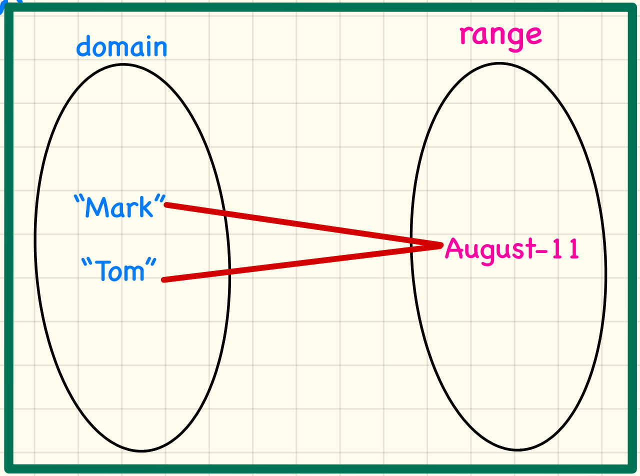
model.range_restricted (Aug-11)

~~domain restric.~~

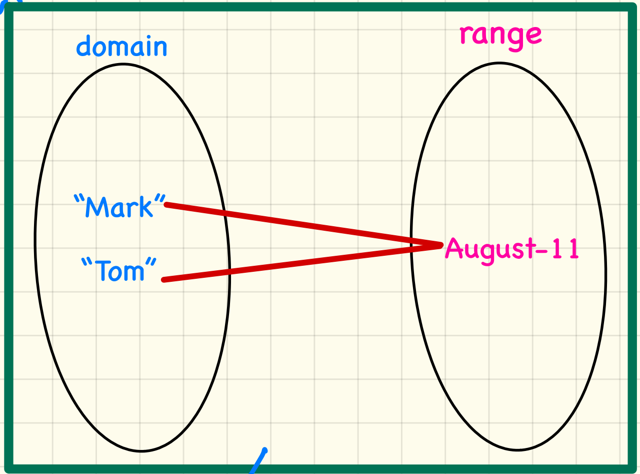
~~domain sub.~~

range restric.

range sub.

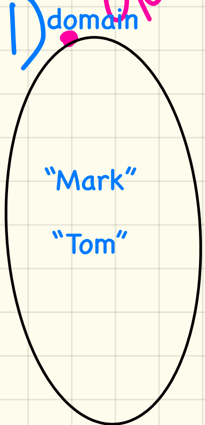


model.range_restricted(Aug-11)



model.range_restricted(Aug-11)

domain



remind (d) : A[N]

model[d_r (d).domain]S

||

Result] T

$$\boxed{\begin{array}{c} S \subseteq T \\ T \subseteq S \end{array}}$$

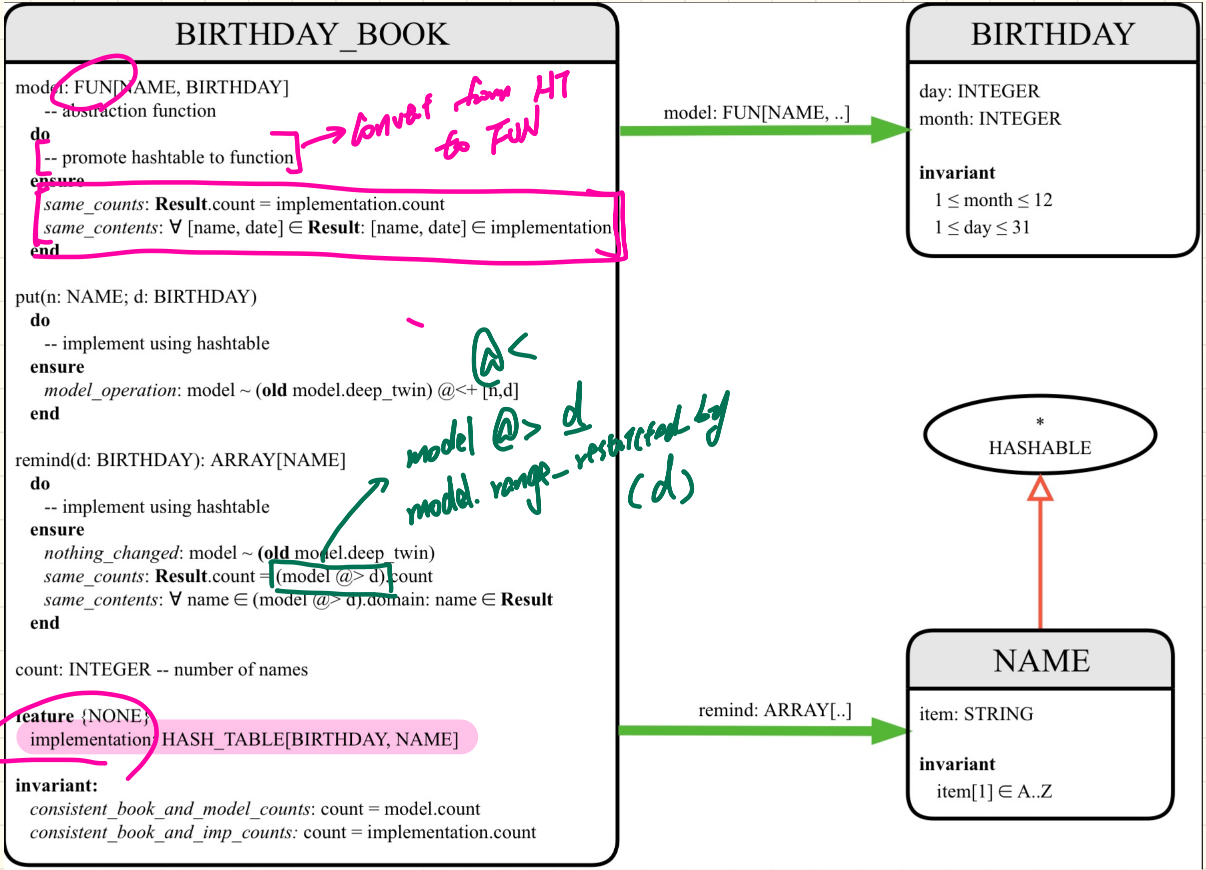
set of tuples

range_restricted (______)

range_restricted_by (______)

single tuple

Birthday Book: Implementation



Stack of Strings vs. Stack of Accounts

unknow id.

```
class STRING_STACK        Stack[G]
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
```

single def. of Stack

```
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]  STRING  ACCOUNT
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
```

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
```

✗ not compile

✓

HashTable < String , Account > table =

new HashTable<>();

Principle of **Information Hiding**

across cart ≠ order
Result := Result + order.p * order.q
end

array-specific

Supplier:

Hide {NONE}

legal change

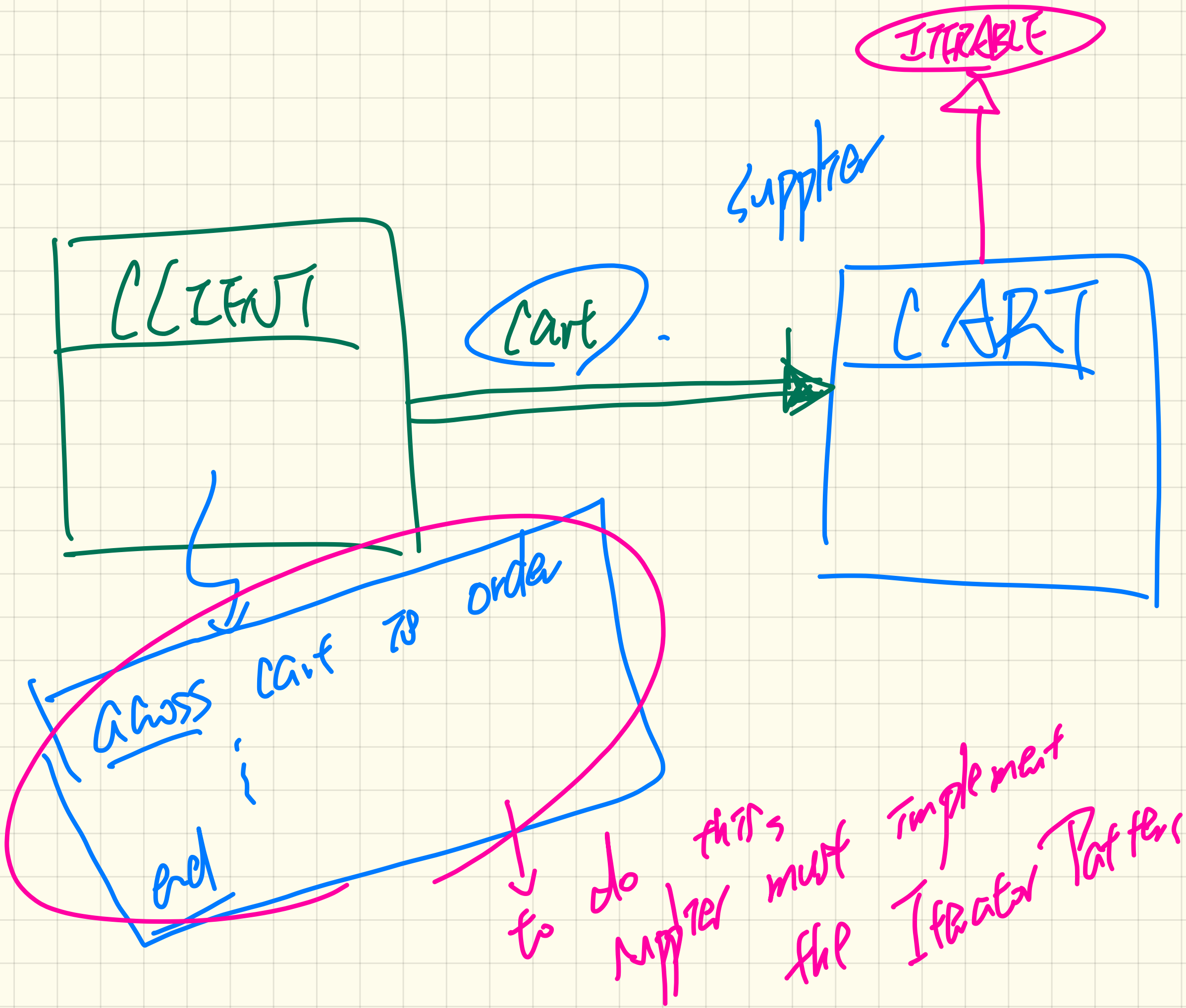
```
class
  CART
feature
  orders: ARRAY[ORDER]
end

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

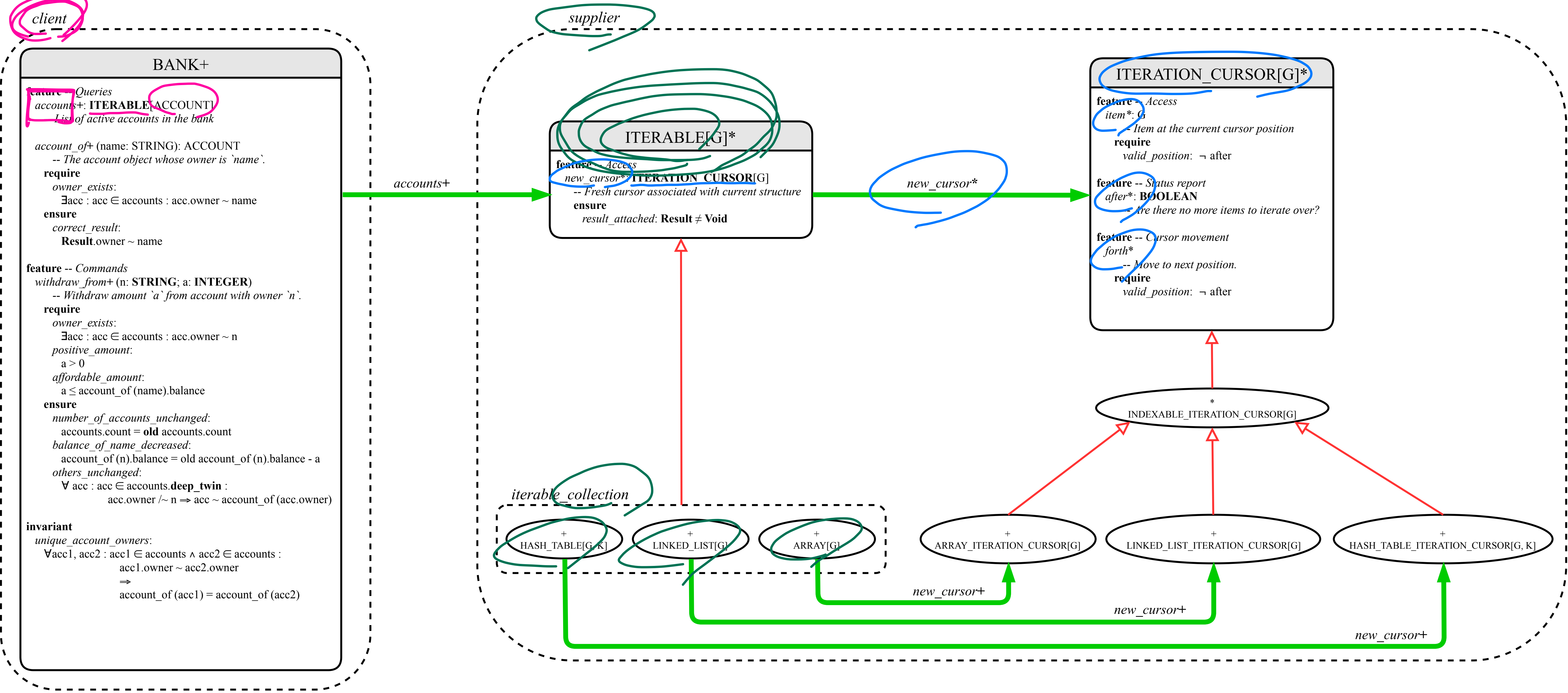
Client:

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

Problems?

DS. of supplier was not hidden from the client


ITERABLE
supplier
CLIENT
cart
CART
access cart to order
end
to do this supplier must implement the Iterator Pattern

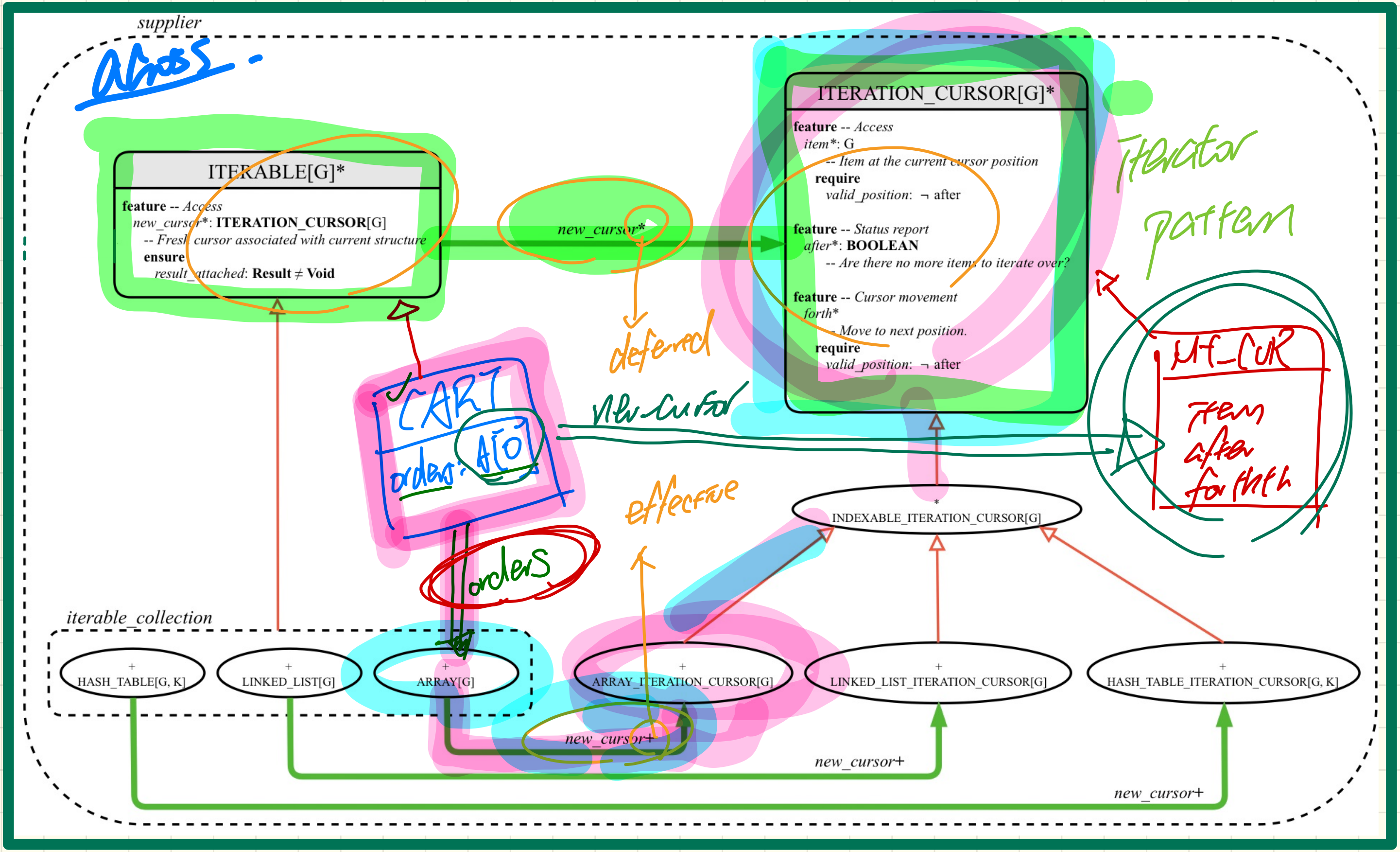


client
supplier
BANK+
feature -- Queries
accounts+: ITERABLE[ACCOUNT]
-- List of active accounts in the bank
account_of+ (name: STRING): ACCOUNT
-- The account object whose owner is `name`.
require
owner_exists:
∃acc : acc ∈ accounts : acc.owner ~ name
ensure
correct_result:
Result.owner ~ name
feature -- Commands
withdraw_from+ (n: STRING; a: INTEGER)
-- Withdraw amount `a` from account with owner `n`.
require
owner_exists:
∃acc : acc ∈ accounts : acc.owner ~ n
positive_amount:
a > 0
affordable_amount:
a ≤ account_of (name).balance
ensure
number_of_accounts_unchanged:
accounts.count = old accounts.count
balance_of_name_decreased:
account_of (n).balance = old account_of (n).balance - a
others_unchanged:
∀ acc : acc ∈ accounts.deep_twin :
acc.owner /~ n ⇒ acc ~ account_of (acc.owner)
invariant
unique_account_owners:
∀acc1, acc2 : acc1 ∈ accounts ∧ acc2 ∈ accounts :
acc1.owner ~ acc2.owner
⇒
account_of (acc1) = account_of (acc2)
accounts+
ITERABLE[G]*
feature -- Access
new_cursor*: ITERATION_CURSOR[G]
-- Fresh cursor associated with current structure
ensure
result_attached: Result ≠ Void
new_cursor*
ITERATION_CURSOR[G]*
feature -- Access
item*: G
-- Item at the current cursor position
require
valid_position: ¬ after
feature -- Status report
after*: BOOLEAN
-- Are there no more items to iterate over?
feature -- Cursor movement
forth*
-- Move to next position.
require
valid_position: ¬ after
*
INDEXABLE_ITERATION_CURSOR[G]
iterable_collection
+
HASH_TABLE[G, K]
+
LINKED_LIST[G]
+
ARRAY[G]
+
ARRAY_ITERATION_CURSOR[G]
+
LINKED_LIST_ITERATION_CURSOR[G]
+
HASH_TABLE_ITERATION_CURSOR[G, K]
new_cursor+
new_cursor+
new_cursor+

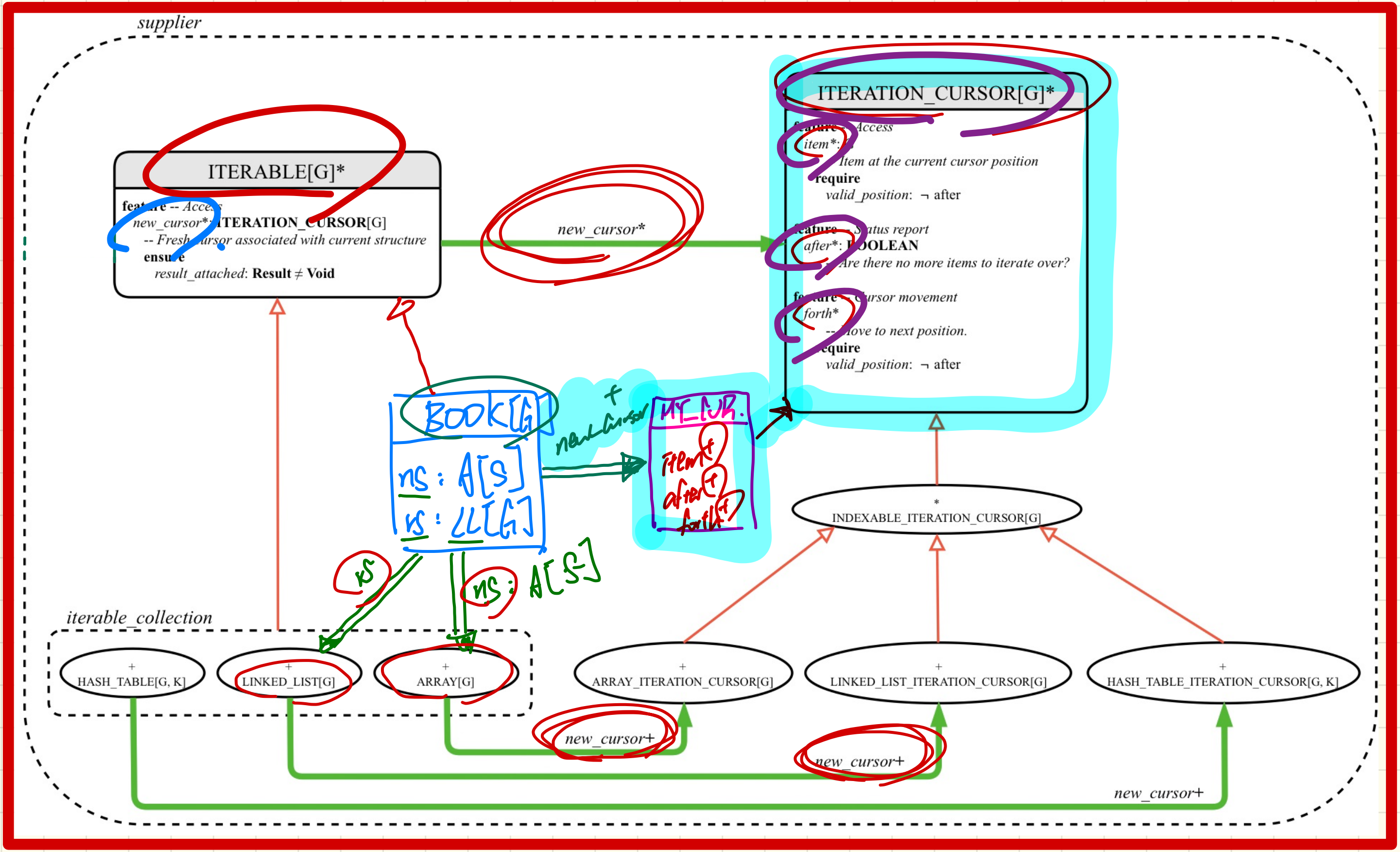
Lecture 11

Monday February 10

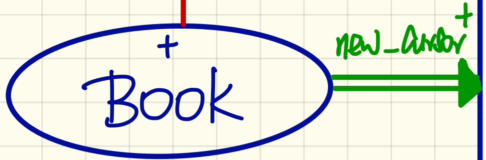
Implementing the Iterator Pattern: Easy Case



Implementing the Iterator Pattern: Hard Case

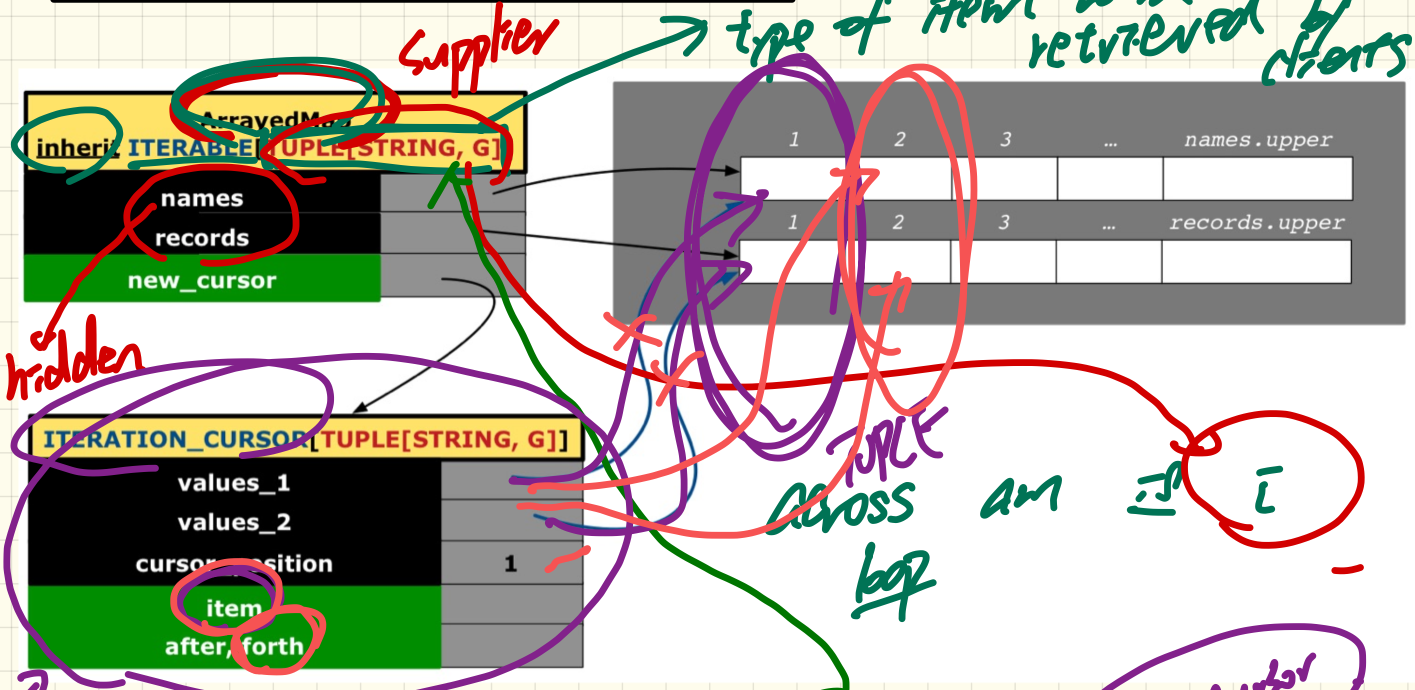


Implementing the Iterator Pattern: Hard Case




```
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



am: A_M[..]

am. names ✗

across am is i
loop
:
end

across
am
loop
:
end

am. new_cursor
ITERATION_CURSOR.

Use of Iterable in Contracts

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

INTERFACE TYPE

↳ Dynamic type of collection can be any descendant class of ITER.

collection: ARRAY[I].

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

deferred IT. LL, descendant classes of LIST.

Client.

collection: ITERABLE[STRING]

collection.make_empty

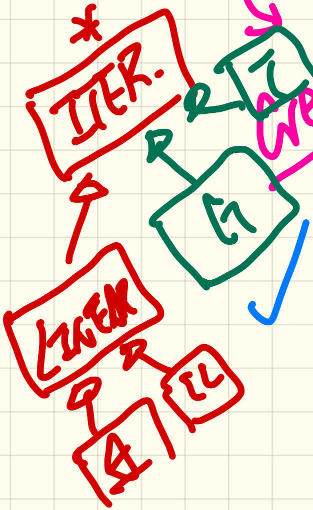
create

ARRAY[S]

LL[S]

static type.

collection is STRING



across collection is i loop end

across 1 |..| collection.count is i loop collection[i] end

collection.item(i)

[ ]

~~[]~~

Array → Linear

list [ ].

Use of Iterable in Contracts: Exercise

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

$\forall i, j : INTEGER \mid$
$1 \le i \le accounts.count \land 1 \le j \le accounts.count \bullet$
$accounts[i] \sim accounts[j] \Rightarrow i = j$

```
end
```

cannot be ITERABLE
∵ we want to refer to positions.

across | 1..| accounts.count is ~~I~~, J.

each across can only be bound to a single dum. var.

Use of Iterable in Implementation (1)

Col. new_cursor.

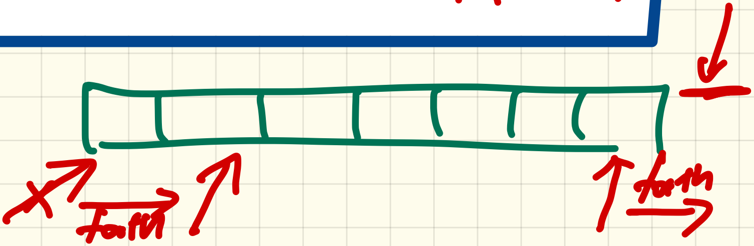
```
class BANK
  accounts: ITERABLE [ACCOUNT]
  max_balance: ACCOUNT
      -- Account with the maximum balance value.
    require ??
    local
      cursor: ITERATION_CURSOR[ACCOUNT]; max: ACCOUNT
    do
      from max := accounts [1]; cursor := accounts. new_cursor
      until cursor. after
      do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

set the cur. to beginning pos.

↳ cursor. start ✗

↳ not necessary.

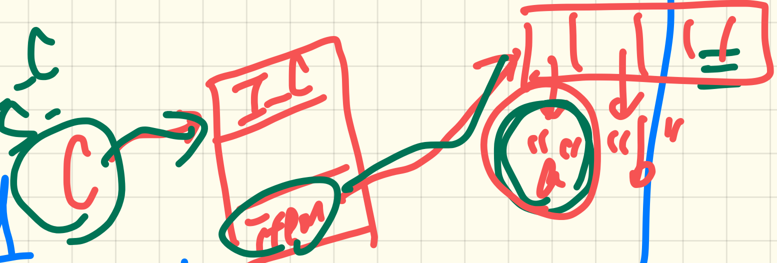
without this line. ⇒ inf. loop



collection : ITERABLE[S]

```
local
    c : I_C[S]
from
    c := col.new_cursor
until
    c.after
loop
    c.item
    c.forth
end
```

```
class collection is item
    loop
        c :
        c.item :
    end
```




```
class collection as c → I_C[S]
    loop
        c.item
        c.forth
    end
```

Use of Iterable in Implementation (2)

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

→ I.

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

Exercise 1

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

new_cursor*

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

  after: BOOLEAN
    deferred end

  forth
    deferred end
end
```

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

db+

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

new_cursor+

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

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```

Exercise 1

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

1.1 2.2 2.1 T[I,S]

+db

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

1.2 STR INT 2.3 T[INT, STR] T[G, H] T[H, G] 2.6 3.1 4.2 gs, hs G, H

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

2.4 T[H, G] 2.5 T[H, G]

new_cursor*

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

  after: BOOLEAN
    deferred end

  forth
    deferred end
end
```

T[H, G] 3.5 3.4 TUPLE[H, G]

new_cursor+

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

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```

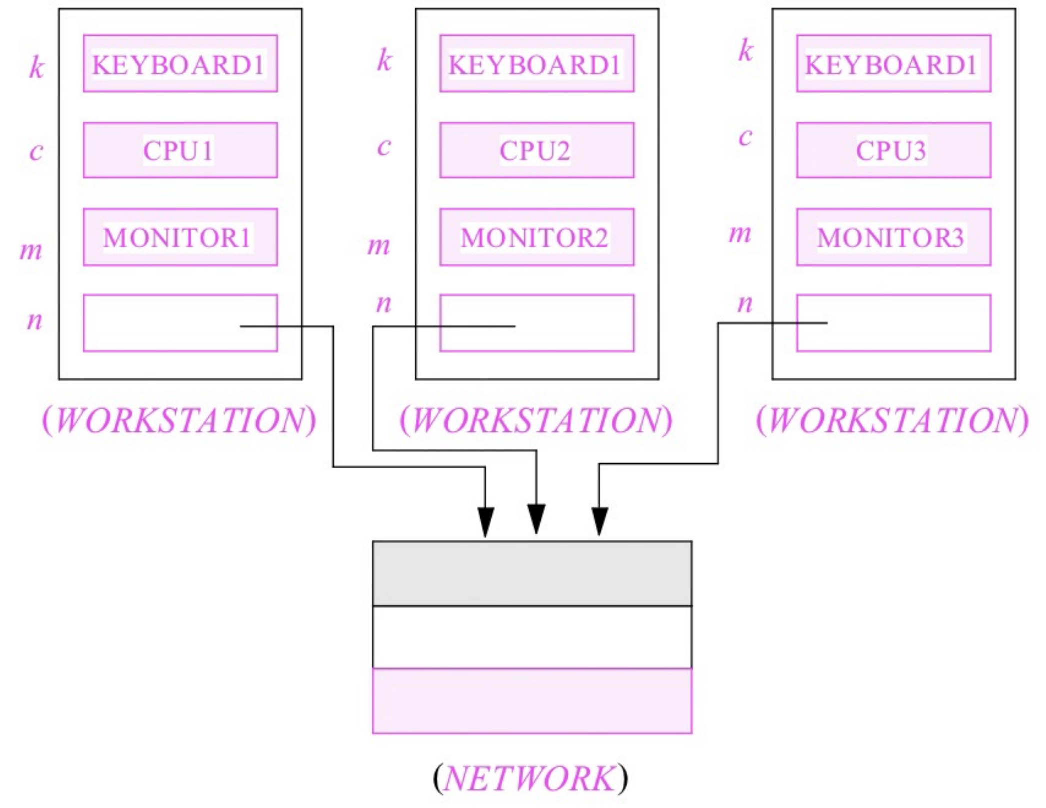
3.2 H G 3.6 G H 4.1 3.3 TUPLE[~~H~~, ~~G~~] M N add(x, y: Int)

=

x y

y + x

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
```

```
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
```

change:
monitor may be shared

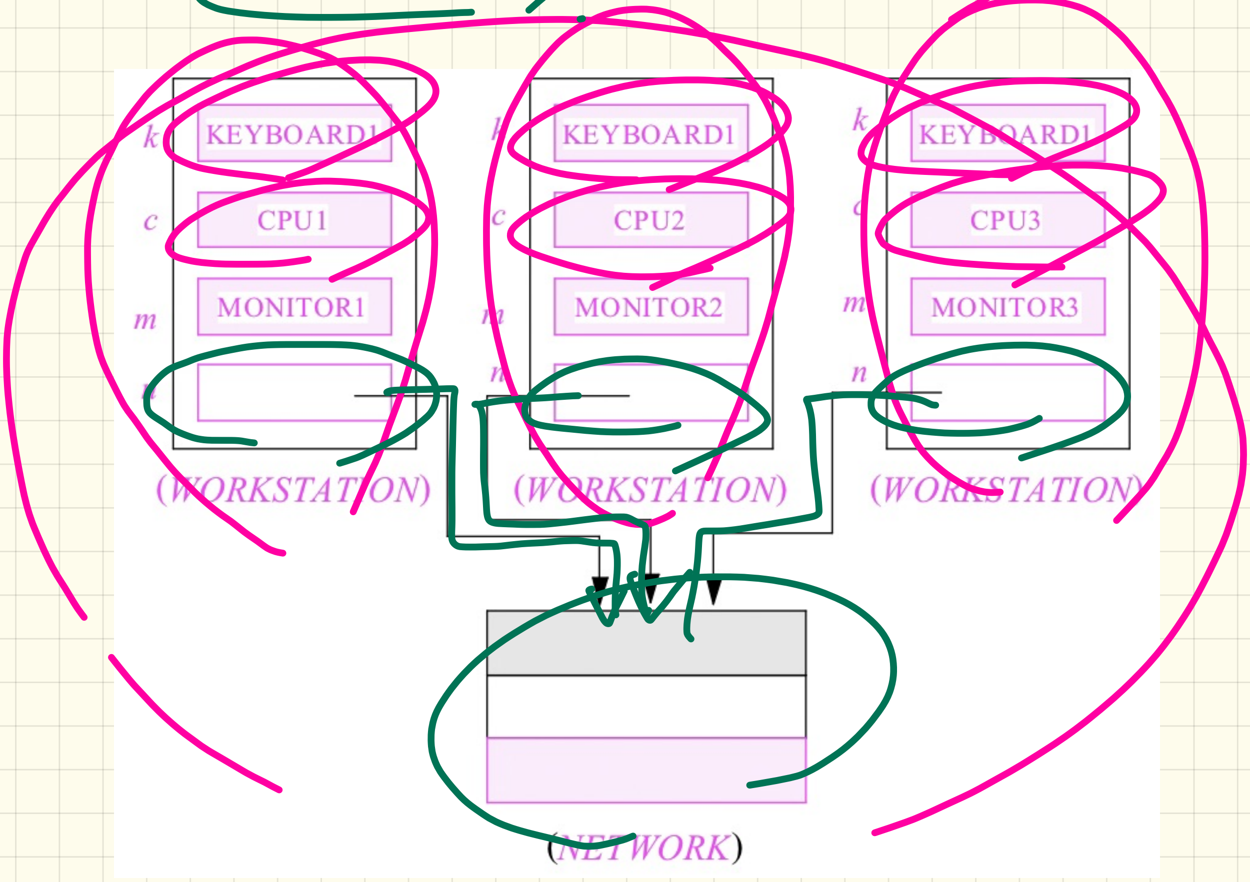
LECTURE 12

WEDNESDAY FEBRUARY 12

- Labtest 1 Review Session:
  Thursday 10am to 11am R S201
- Labtest 1 Seat Map
- Lab 3: ETF Turorial Videos

4, project

Modelling: Aggregation vs. Composition



Expanded Type for Composition

ref

```
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
```

non-expanded

k1: KEYB.

k2: exp. KEYB.

```
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
```

expanded

keyboard { may be shared / may not shared aliasing

avoid

safety

eb1 → B: i = 0

eb2 → B: i = ~~0~~ ~~15~~ 10

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

```
test_expanded
  local
    eb1, eb2: B
  do
    check eb1.i = 0 and eb2.i = 0 end
    check eb1 = eb2 end
    eb2.change_i (15)
    check eb1.i = 0 and eb2.i = 15 end
    check eb1 /= eb2 end
    eb1 := eb2
    check eb1.i = 15 and eb2.i = 15 end
    eb1.change_i (10)
    check eb1.i = 10 and eb2.i = 15 end
    check eb1 /= eb2 end
  end
```

create

create

eb1.make

eb2.make

F

∵ diff. objs.

10

F ∵ they now ref. same object.

boolean

check [ ] end → assertTrue([ ]).

class B

obj1 : B

obj2 : expanded B

obj1 := obj2

~~obj1 = obj2~~

~~not compiling~~ ..

Use of Expanded Type

eb1: B, i = ~~0~~ ~~15~~ 10

eb2: B, i = ~~0~~ 15

no notion of address

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

inherit redefine

is_equal
end

```
test_expanded
  local
    eb1, eb2: B
  do
    check eb1.i = 0 and eb2.i = 0 end
    check eb1 = eb2 end
    eb2.change_i (15)
    check eb1.i = 0 and eb2.i = 15 end
    check eb1 /= eb2 end
    eb1 := eb2
    check eb1.i = 15 and eb2.i = 15 end
    eb1.change_i (10)
    check eb1.i = 10 and eb2.i = 15 end
    check eb1 /= eb2 end
  end
```

T ∵ eb1.i = 0
eb2.i = 0.

eb1.i /= eb2.i

T

list: LL[S].

is_equal (- - -)
do
end

R := i = other.i
and
list ~ other.list

```
expanded class B
  i : INTEGER
```

eb1 : B

↳ initialize **all** attributes to *default* values

```
expanded class B
  inherit ANY
    redefine default_create end
  default_create do i := 5 end
```

eb2 : B

create eb2.

d_c

expanded class A

i : INT

s : STRING

end

ea1 : A

ea2 : ~~expanded~~ A

⋮

$\boxed{ea1 := ea2}$

expanded class C

⋮

is_equal (..)

do

⋮

end

ec1 : C

ec2 : C

⋮

ec1 = ec2

ec1 ~ ec2

obj1 , obj2 : A .

obj1 == obj2

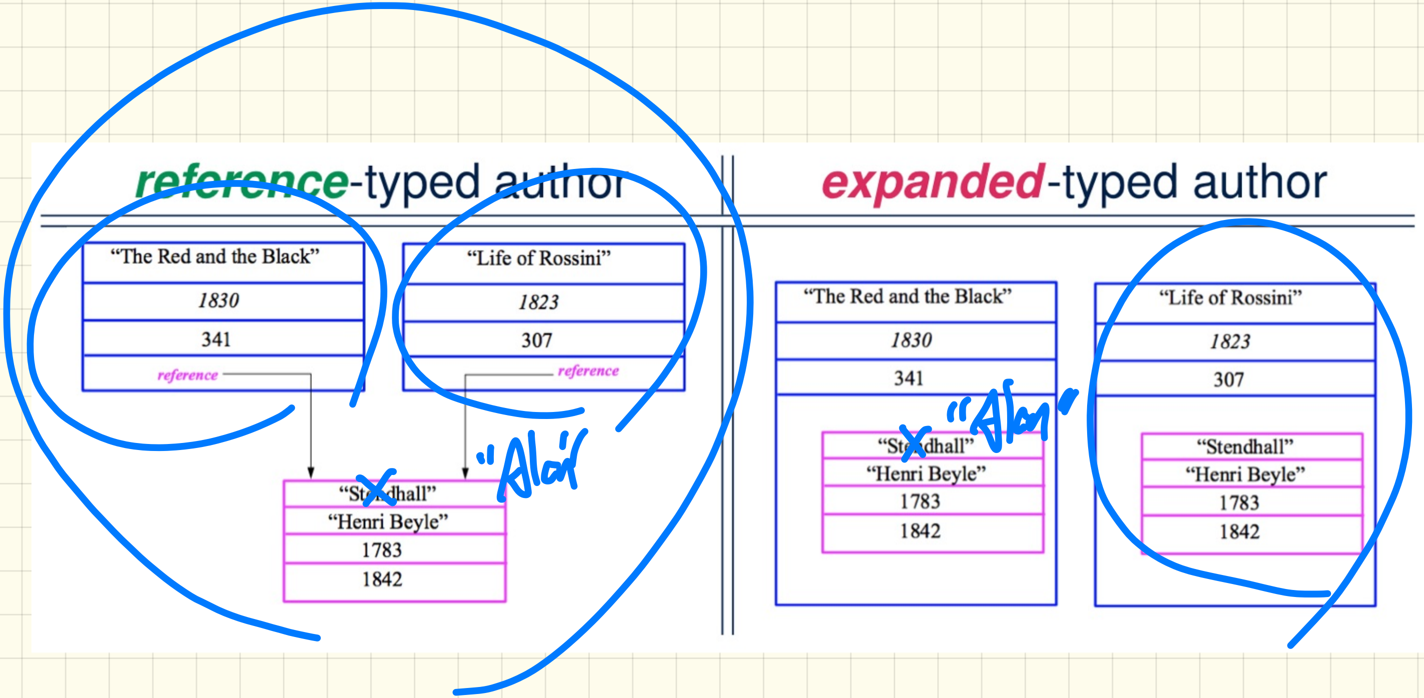
① A is not expanded,

compare address, is equal

② A is expanded, obj1 ~ obj2

compare contents

Reference Type or Expanded Type



Shared Data via **Inheritance**

Cohesion

Single Choice Principle

features in a single class should serve the same purpose. (UNIX: do one thing & do it well).

Descendant:

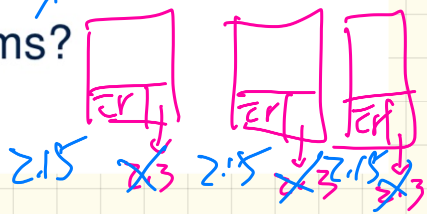
```
class DEPOSIT inherit SHARED_DATA
        -- 'maximum_balance' relevant
end

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
```

2.15

Problems?



Shared Data via Inheritance

Cohesion

Single Choice Principle

deferred

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

$\boxed{ir} \rightarrow 0.11 \rightarrow 0.09$

d1 →

	DEPOSIT
ir	~~0.11~~ 0.09
er	2.34
min	1000
max	1000000

d2 →

	DEPOSIT
ir	~~0.11~~
er	2.34
min	1000
max	1000000

w1 →

	WITHDRAW
ir	~~0.11~~
er	2.34
min	1000
max	1000000

w2 →

	WITHDRAW
ir	~~0.11~~
er	2.34
min	1000
max	1000000

t1 →

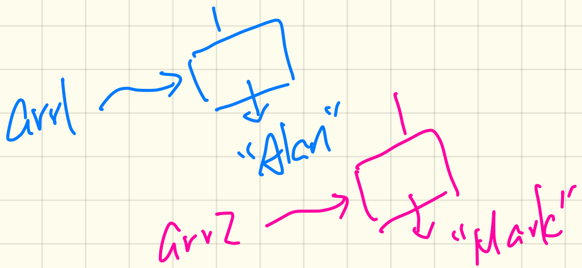
	TRANSFER
ir	~~0.11~~
er	2.34
min	1000
max	1000000

t2 →

	TRANSFER
ir	~~0.11~~
er	2.34
min	1000
max	1000000

```
d1, d2: DEPOSIT
w1, w2: WITHDRAW
t1, t2: INTERNATIONAL_TRANSFER
...
d1.set_max_balance
w2.set_min_balance
t2.set_exchange_rate
```

Once Routine (1)



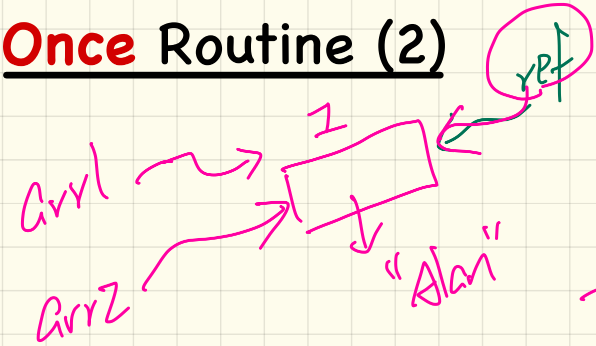

```
test_query: BOOLEAN
  local
    a: A
    arr1, arr2: ARRAY[STRING]
  do
    create a.make

    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
```

F

T

```
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
```

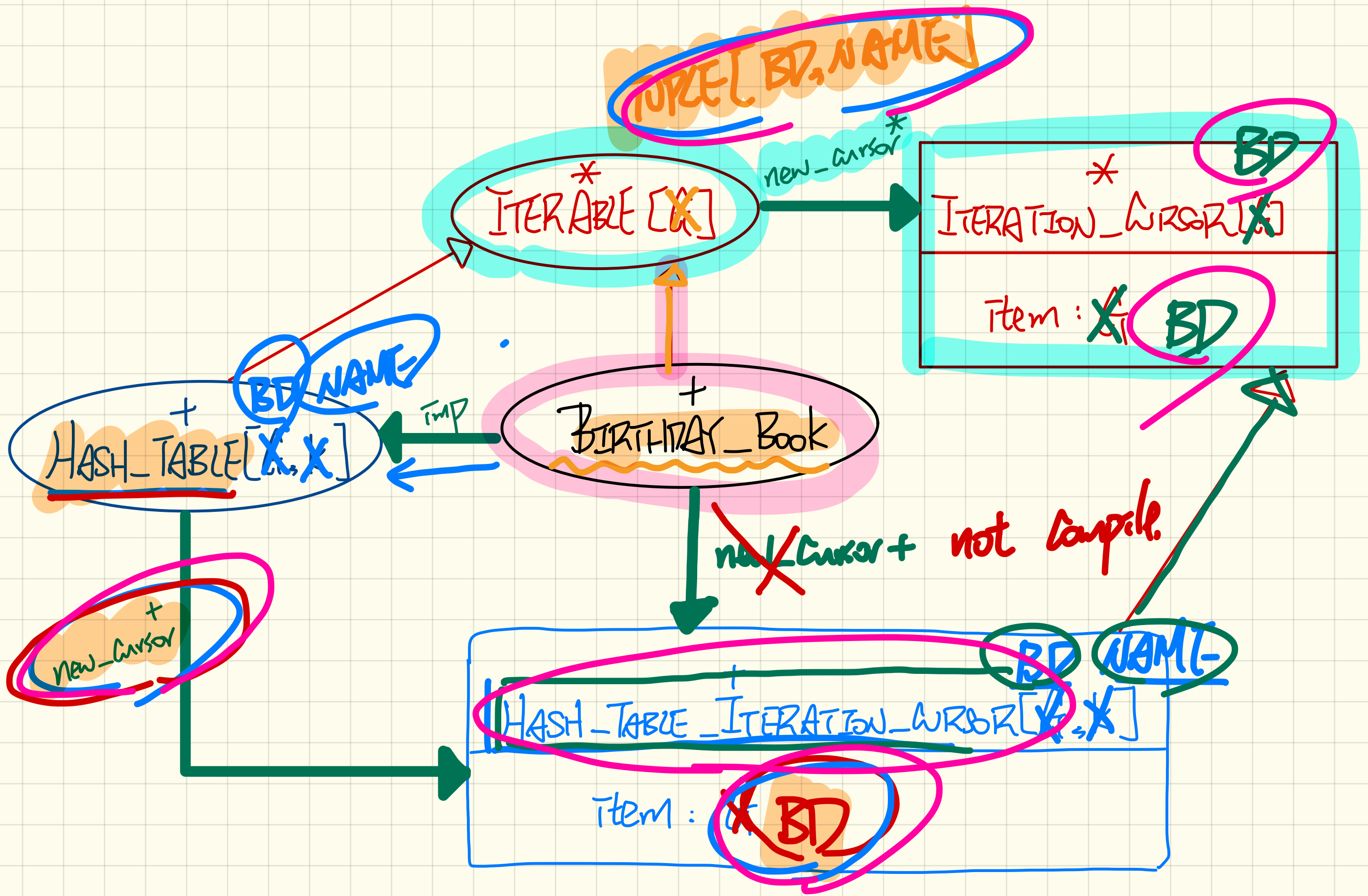
Once Routine (2)



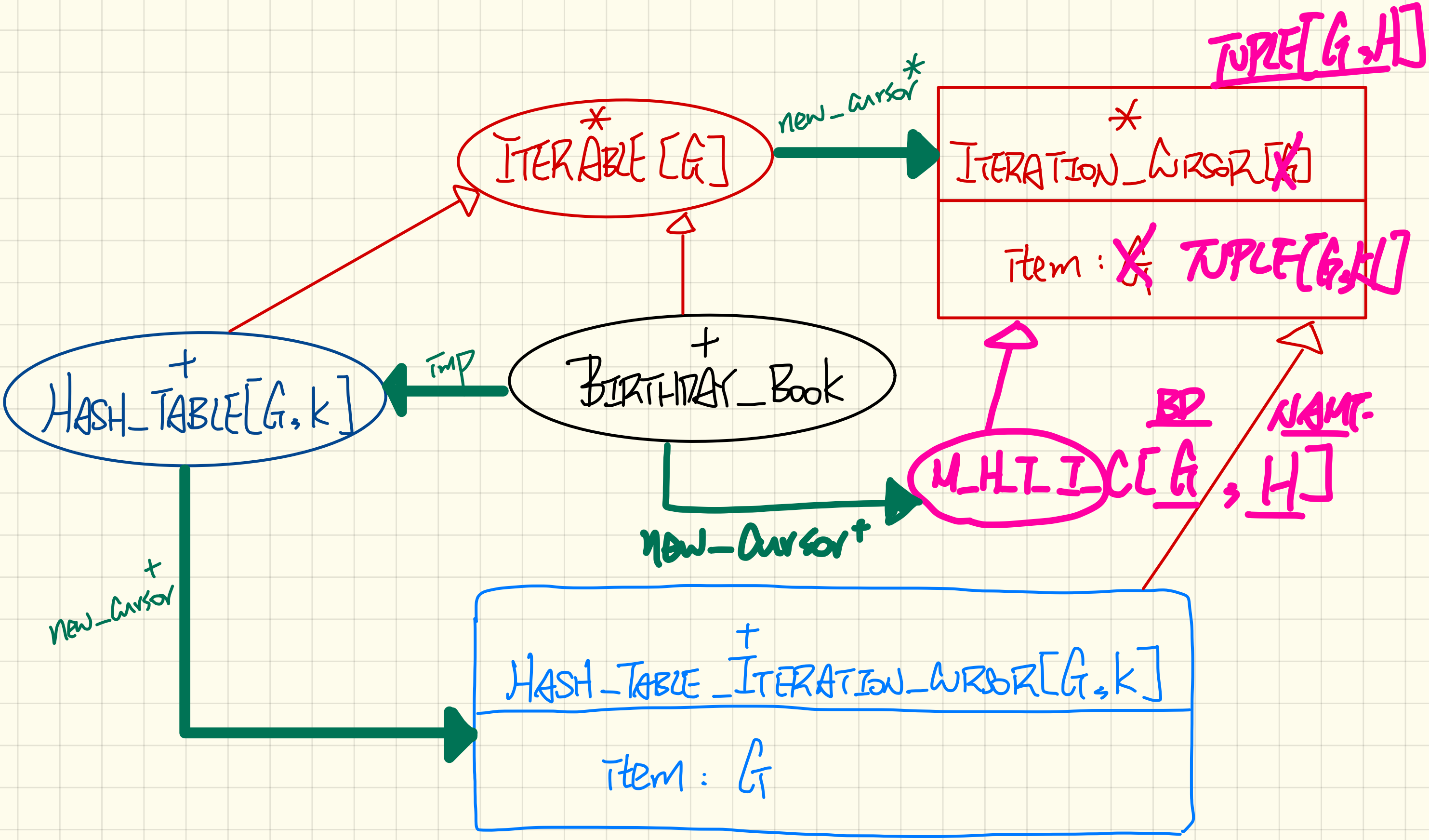

```
class A
create make
feature -- Constructor
  make do end
feature -- Query
  new_once_array (s: STRING): ARRAY[STRING]
      -- A once query that returns an array.
    once
      create {ARRAY[STRING]} Result.make_empty
      Result.force (s, Result.count + 1)
    end
  new_array (s: STRING): ARRAY[STRING]
      -- 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 call
    arr1 := a.new_once_array ("Alan")
    Result := arr1.count = 1 and arr1[1] ~ "Alan"
    check Result end
                 subsequent call
    arr2 := a.new_once_array ("Mark")   ignored
    Result := arr2.count = 1 and arr2[1] ~ "Alan"
    check Result end

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

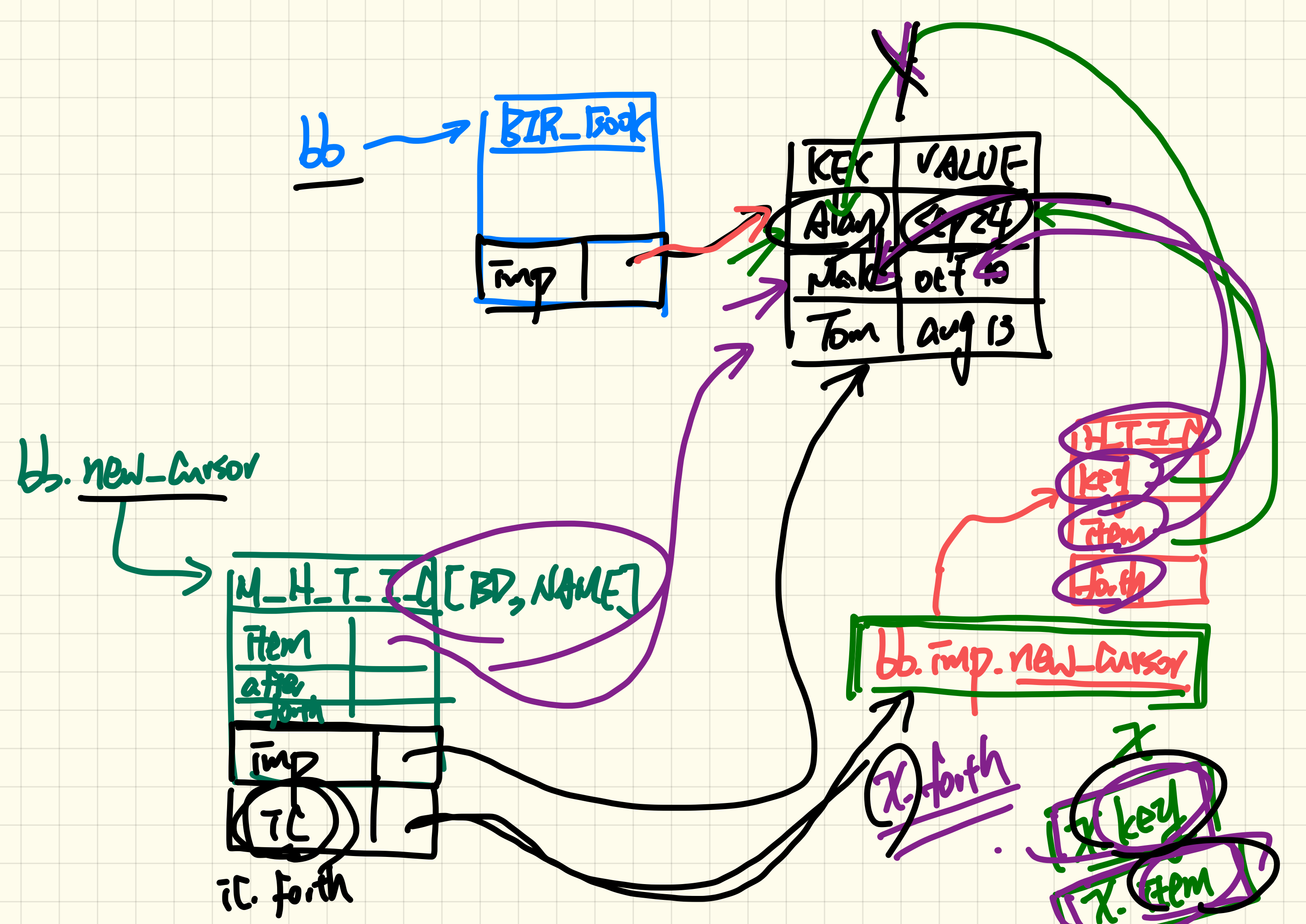
LABTEST I REVEW
TUTORIAL
THURSDAY FEBRUARY 13

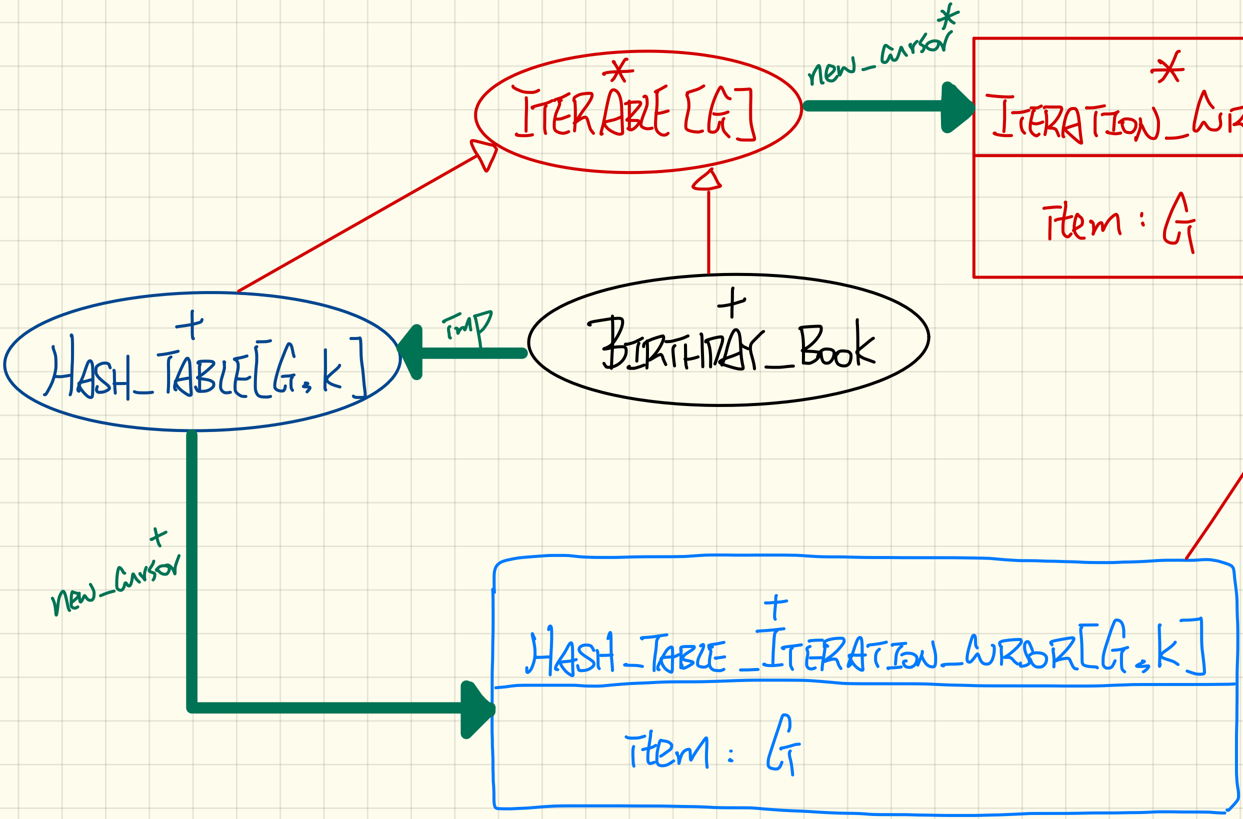


TUPLE[BD, NAME]
ITERABLE[X]
new_cursor
ITERATION_CURSOR[X]
BD
item : X
BD
BD
NAME
HASH_TABLE[X, X]
imp
BIRTHDAY_BOOK
new_cursor
new_cursor
not compile
BD
NAME
HASH_TABLE_ITERATION_CURSOR[X, X]
item : X
BD

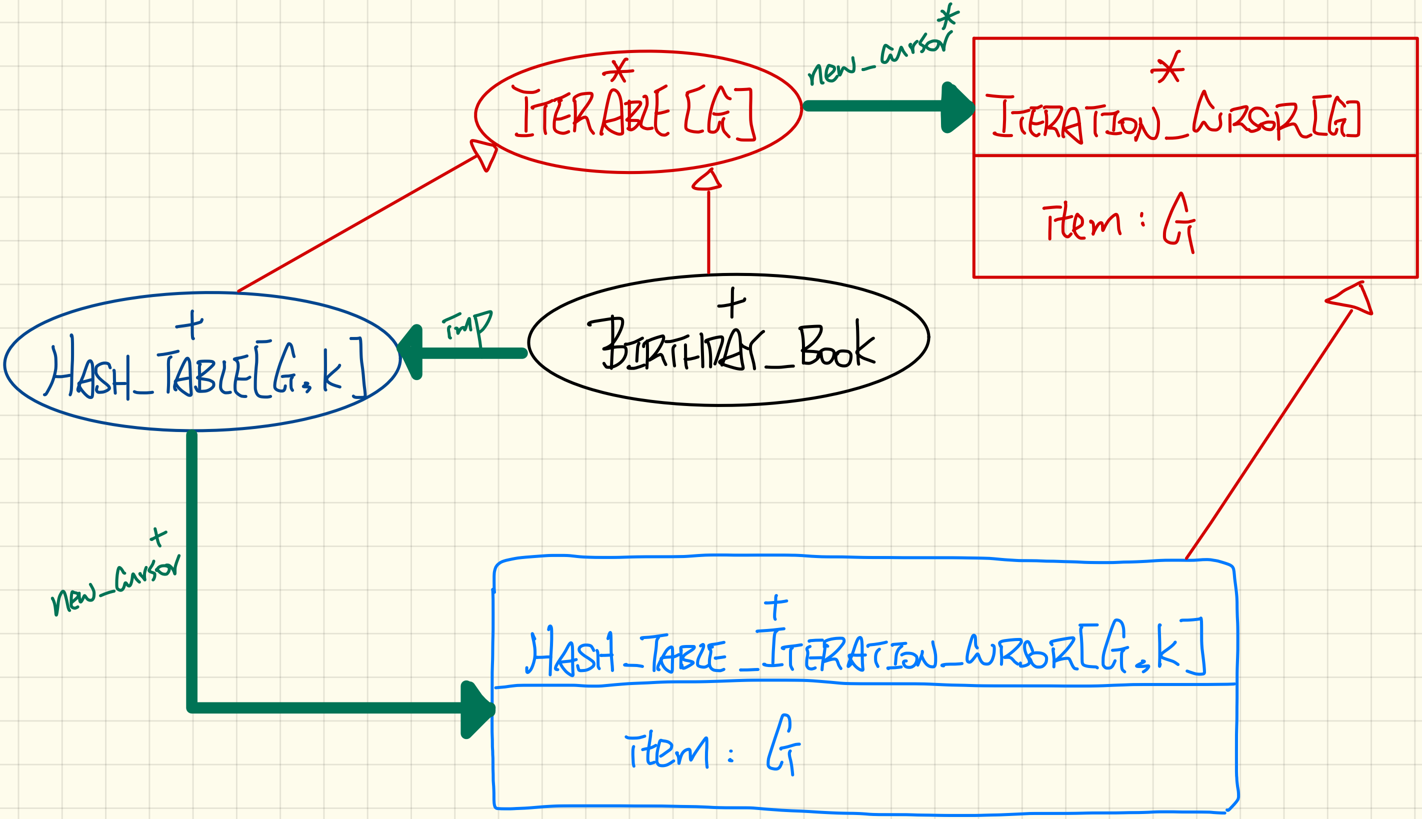


TUPLE[G, H]
ITERABLE[G] *
new_cursor *
ITERATION_CURSOR[G] *
item : G TUPLE[G, H]
HASH_TABLE[G, K] +
imp
BIRTHDAY_BOOK +
BD
NAME
M_H_T_I_C[G, H]
new_cursor +
new_cursor +
HASH_TABLE_ITERATION_CURSOR[G, K] +
item : G


bb
BIR_Book
imp
KEY
VALUE
Alan
Sep 24
Mark
Oct 10
Tom
Aug 13
bb.new_cursor
M_H_T_I_C [BD, NAME]
item
after
forth
imp
IC
ic.forth
H_T_I_C
key
item
forth
bb.imp.new_cursor
x
x.forth
x.key
x.item



ITERABLE[G]
*
new_cursor*
ITERATION_CURSOR[G]
*
item : G
HASH_TABLE[G, K]
+
imp
BIRTHDAY_BOOK
+
new_cursor+
HASH_TABLE_ITERATION_CURSOR[G, K]
+
item : G



ITERABLE[G] *
new_cursor *
ITERATION_CURSOR[G] *
item : G
HASH_TABLE[G, K] +
imp
BIRTHDAY_BOOK +
new_cursor +
HASH_TABLE_ITERATION_CURSOR[G, K] +
item : G

LECTURE 13

MONDAY, FEBRUARY 24

- Office hours today moved to
  Tuesday 12:30 - 14:30
- Moodle announcement on Labtest1
- ETF Tutorial

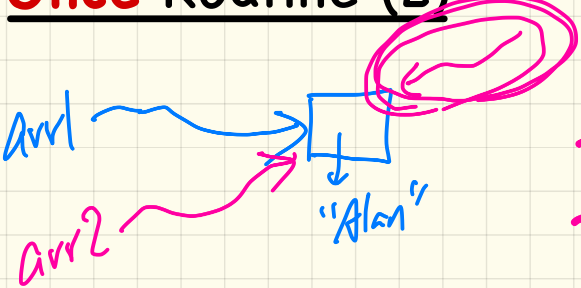
Video 7 on automating acceptance testing

Once Routine (1)

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

```
test_query: BOOLEAN
 local
   a: A
   arr1, arr2: ARRAY[STRING]
 do
   create a.make

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

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

   Result := not (arr1 = arr2)
   check Result end
 end
```

Once Routine (2)



"Mark"

~~"Alan"~~

```
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
```

1st call

2nd call

```
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
```

- Cohesion
- single choice principle
- single data instance

Approximating Once Routines in Java (1)

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

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

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

data2 = BDA.getData();

BD data2 = new BD();

initOnce = T

1st

F

have not yet called

Problem?

cohesion ✓

single data instance ?

Approximating Once Routines in Java (2)

We may encode Eiffel once routines in Java:

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

data

data access

Problem?

Tester

BD d = new
BD(); ×

Singleton Design Pattern: Code (1)

Supplier:

cohesion

```
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
```

only this class can call 'make' as a constructor

only this class can call -make as a command

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

only place where make can be called as a con.

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
```

1st call

2nd call

X not compile

create {DATA} d3.make(23)

Writing $\boxed{\textbf{create } \textit{d1.make}}$ in `test` feature does not compile. Why?

Supplier:

```
class DATA
create {DATA_ACCESS} make
feature {DATA_ACCESS}
  make do v := 10 end
feature -- Data Attributes
  v: INTEGER
  change_v (nv: INTEGER)
    do v := nv end
end
```

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

Client:

```
test: BOOLEAN
  local
    access: DATA_ACCESS
    d1, d2: DATA
  do
    d1 := access.data
    d2 := access.data
    Result := d1 = d2
      and d1.v = 10 and d2.v = 10
    check Result end
    d1.change_v (15)
    Result := d1 = d2
      and d1.v = 15 and d2.v = 15
  end
end
```

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

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
```

is_equal do -- end .

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

one call

another call

?② 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
```

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

```
class A
  s: STRING
    do
      create Result.make ("a")
    end

invariant
  s = s
```

x

"a"

"a"

```
Client
  a: A
  create a
```

① any class invariant violation? No.

Supplier:

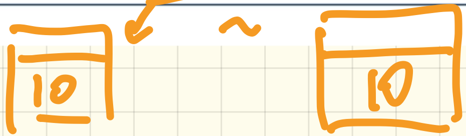
```
class DATA
create {DATA_ACCESS} make
feature {DATA_ACCESS}
  make do v := 10 end
feature -- Data Attributes
  v: INTEGER
  change_v (nv: INTEGER)
    do v := nv end
end
```

is-equal --

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

do

~



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
```

② satisfies single instance of data

No

separate data instances created.

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

Export Status Case 1

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

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

```
class SUPPLIER

create ________ {ANY}
    make

feature ________ {ANY}
  make (init_i: INTEGER)
end   do
        i := init_i
      end

feature
  i: INTEGER
end
```

Export Status Case 2

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

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

```
class SUPPLIER
create {CLIENT_1}
  make

feature {CLIENT_2}
  make (init_i: INTEGER)
    do
      i := init_i
    end

feature
  i: INTEGER
end
```

Singleton Design Pattern: Code (2.1)

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
      ...
    end
end
```

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

Singleton Design 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 call to data

2nd call to data

Error Handling using Singleton

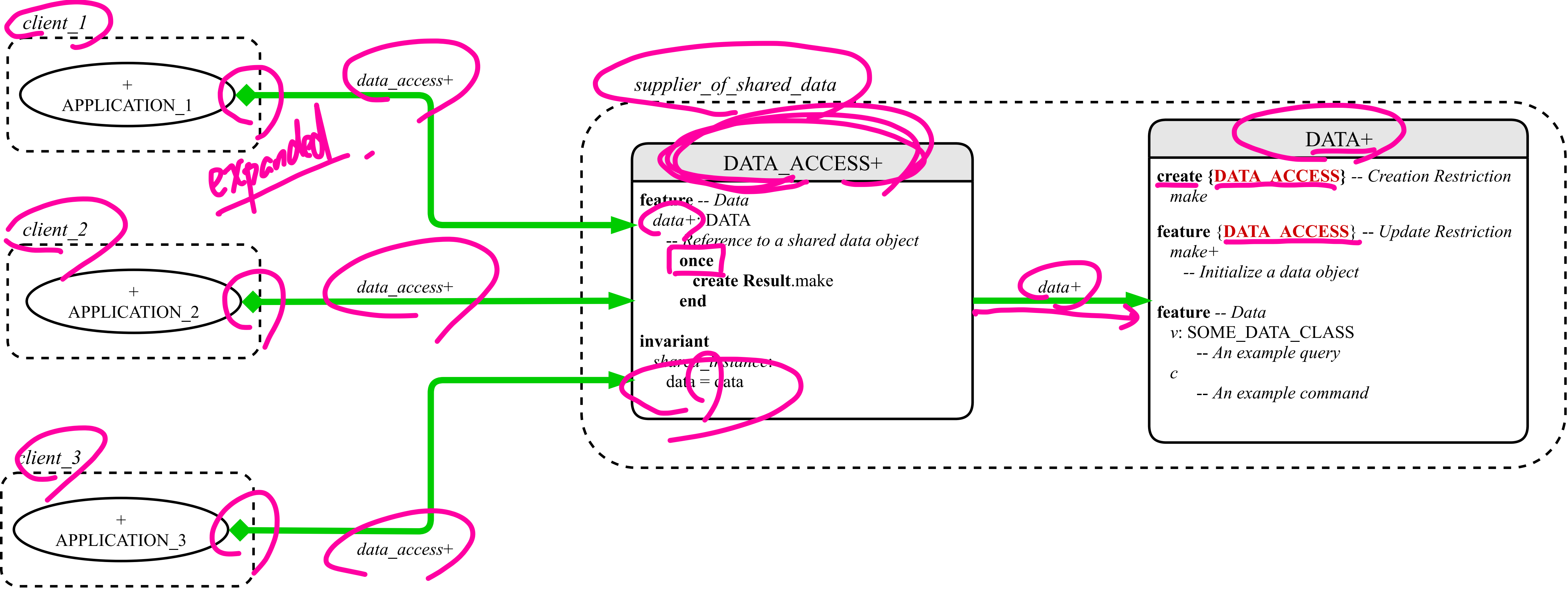
- only a single, shared error object at runtime.

ETF_DEPOSIT
deposit(..)
local
ea : E_A
do
ea.error
end

ERROR_ACCESS
feature
error
once
- -

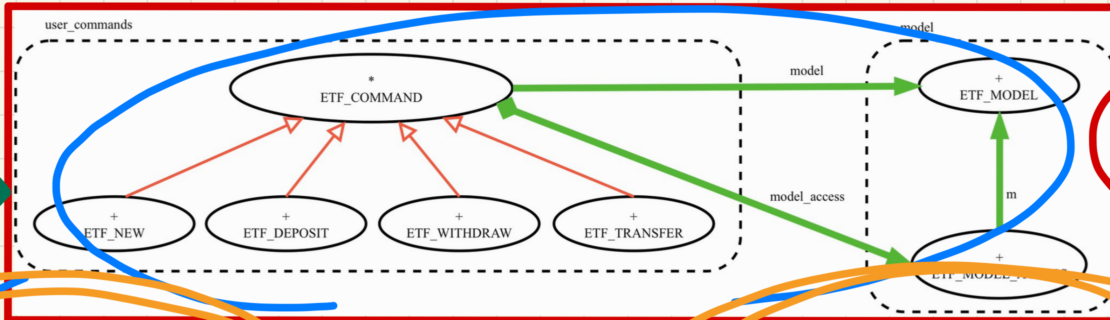
error

ERROR
create {E_A}
feature {E_A}
make


client_1
+
APPLICATION_1
data_access+
expanded
client_2
+
APPLICATION_2
data_access+
client_3
+
APPLICATION_3
data_access+
supplier_of_shared_data
DATA_ACCESS+
feature -- Data
data+: DATA
-- Reference to a shared data object
once
create Result.make
end
invariant
shared_instance:
data = data
data+
DATA+
create {DATA_ACCESS} -- Creation Restriction
make
feature {DATA_ACCESS} -- Update Restriction
make+
-- Initialize a data object
feature -- Data
v: SOME_DATA_CLASS
-- An example query
c
-- An example command

ETF: Input-Output-Based Acceptance Testing

input



output

new("alan")
new("mark")
deposit("alan", 200)
deposit("mark", 100)
transfer("alan", "mark", 50)

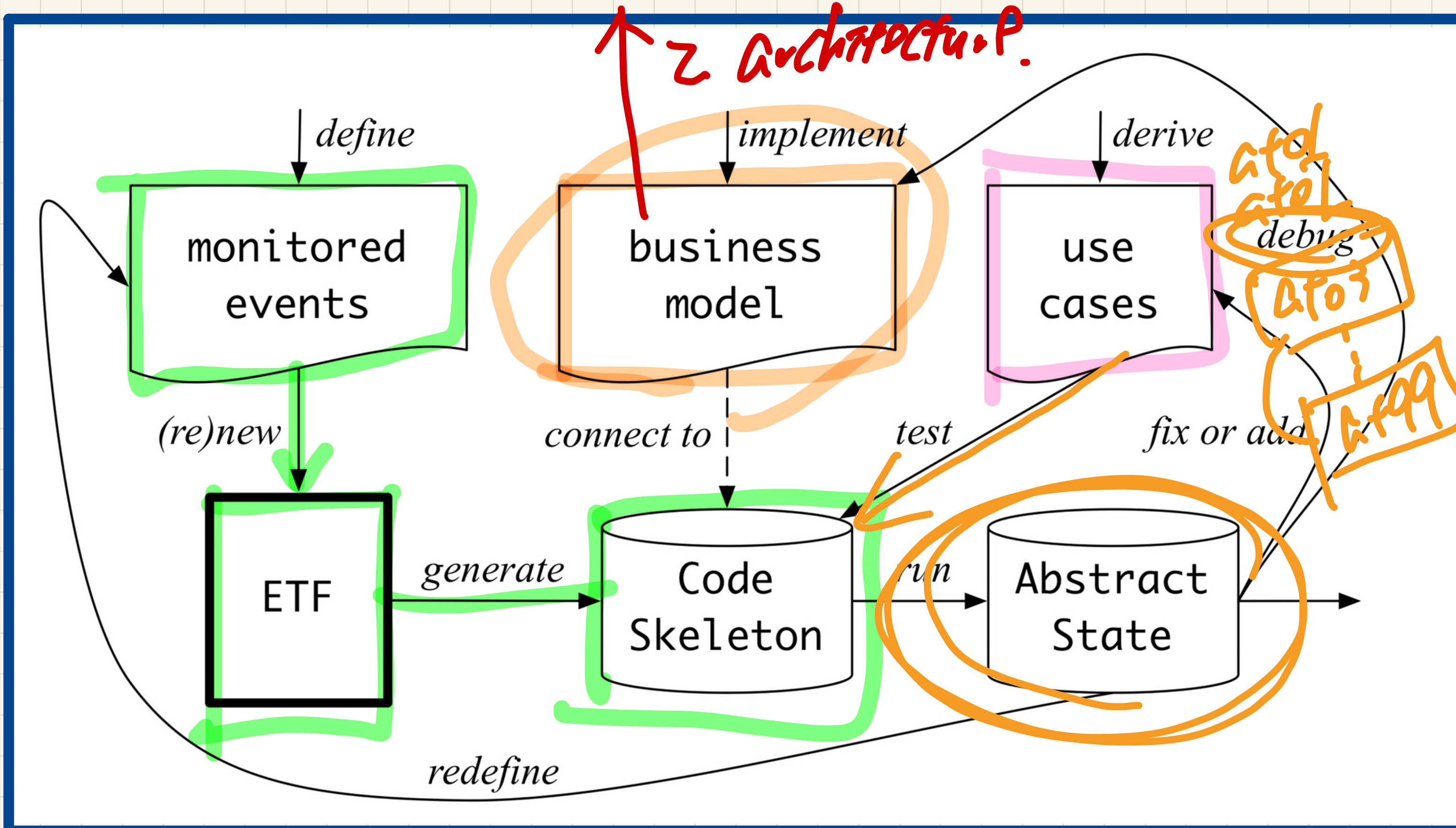
{}
-> new("alan")
{alan: 0}
-> new("mark")
{alan: 0, mark: 0}
-> deposit("alan", 200)
{alan: 200, mark: 0}
-> deposit("mark", 100)
{alan: 200, mark: 100}
-> transfer("alan", "mark", 50)
{alan: 150, mark: 150}

abstract state

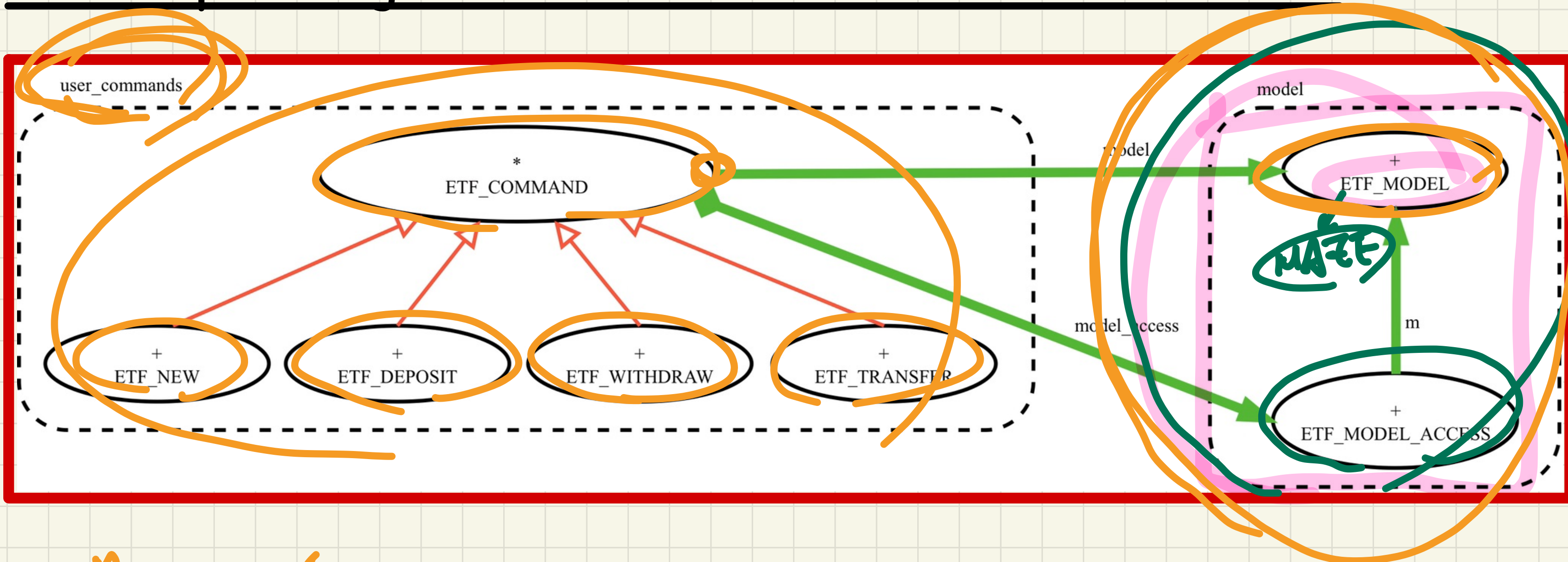
acceptance test
(no syntax of a specific prog. lang.)

ETF: Workflow

1. correctness
2. architecture

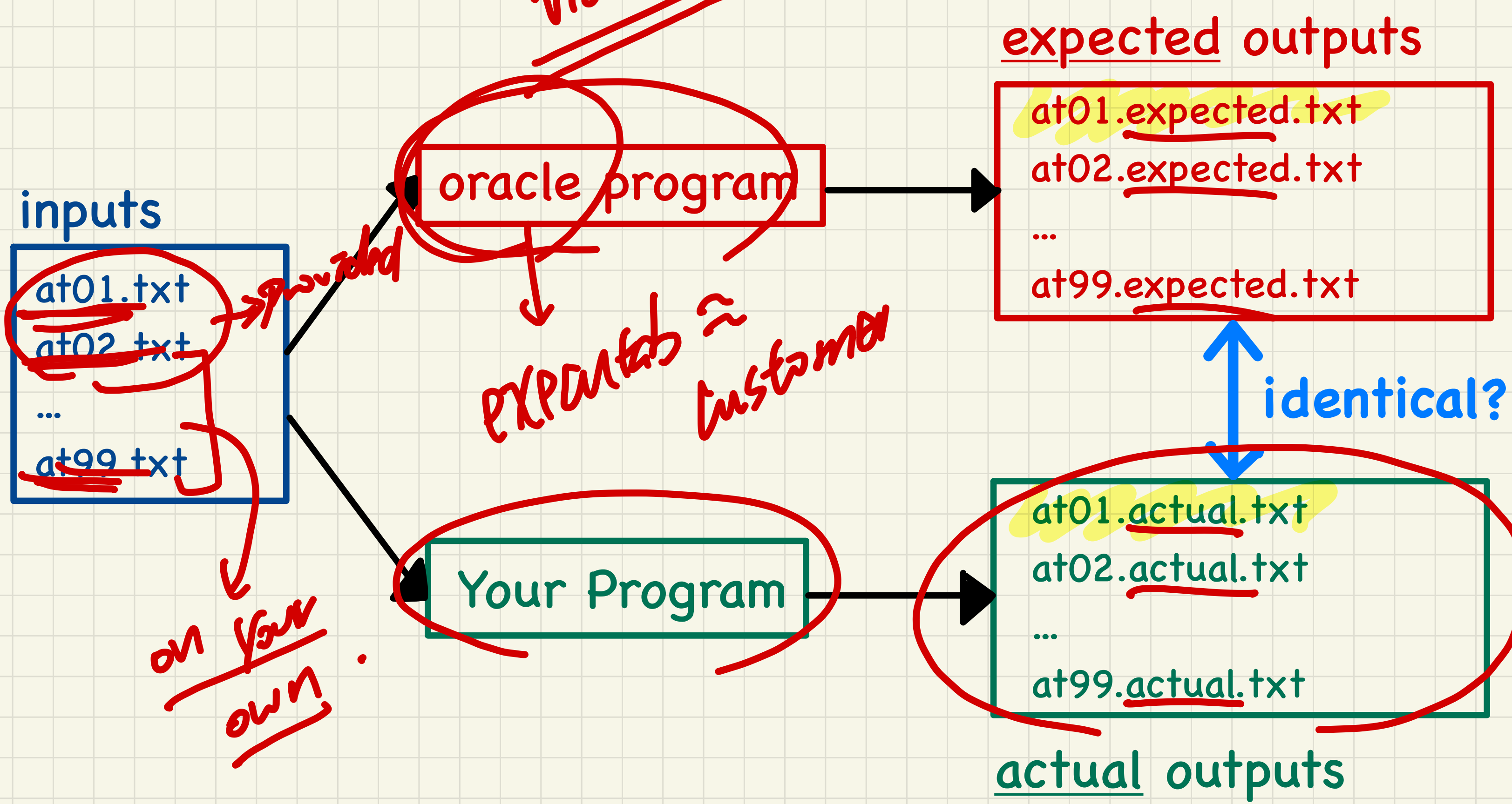


ETF: Separating User Interface and Business Model

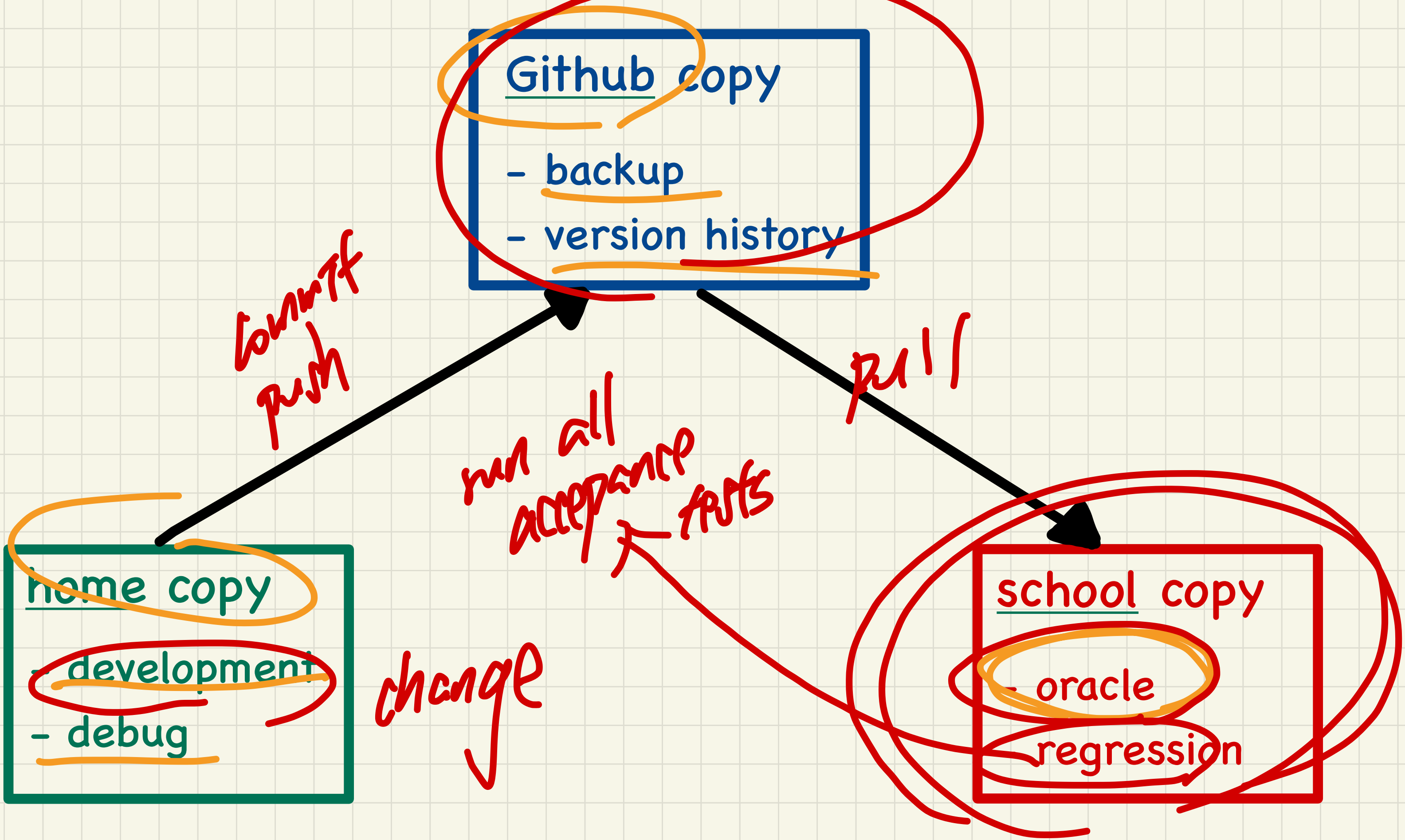


m
ETF_MOVE
ETF_ABORT
⋮

Regression Testing



Automating Regression Testing



Lecture 14

Wednesday February 26

features

easy:

INT = 1

nomp constants

inherit

ETF_I_C

ETF_NEW_GAME

new_game(level:

do if level = easy

- Office hours today moved to Thursday 12:30 - 14:30

- Lab 3: Use of Enumeration Types

ui.txt

type level = {easy, m, h}

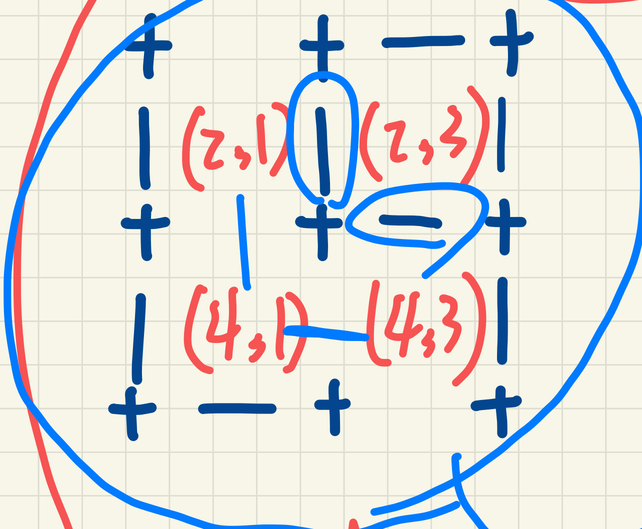
type dir = {E, W, S, N}

ETF

Maze

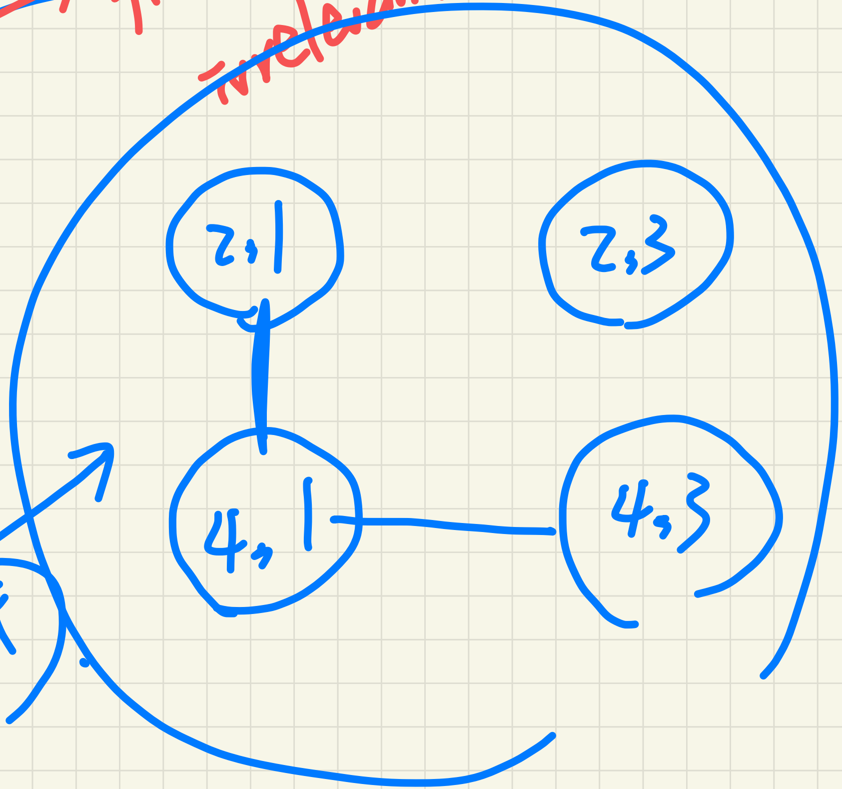
abstraction

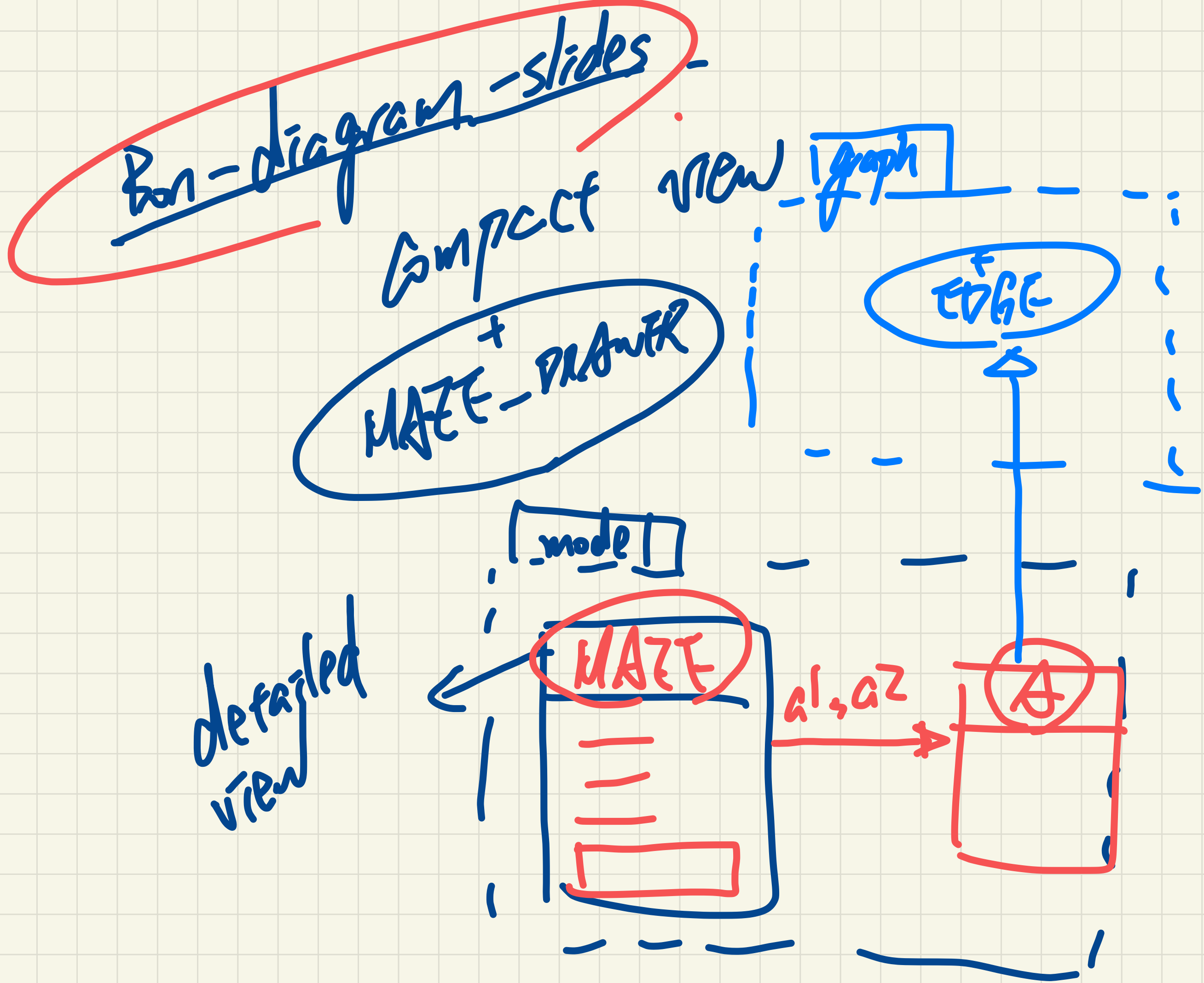
→ filter out irrelevant deatials



2D Array

abstract




Bon-diagram-slides
compact view
graph
EDGE
MAZE_DRAWER
model
MAZE
a1, a2
A
detailed view

Inheritance: Motivating Problem

Nouns -> classes, attributes, accessors

Verbs -> mutators

Problem: A *student management system* stores data about students. There are two kinds of university students: *resident* students and *non-resident* students. Both kinds of students have a *name* and a list of *registered courses*. Both kinds of students are restricted to *register* for no more than 10 courses. When *calculating the tuition* for a student, a base amount is first determined from the list of courses they are currently registered (each course has an associated fee). For a non-resident student, there is a *discount rate* applied to the base amount to waive the fee for on-campus accommodation. For a resident student, there is a *premium rate* applied to the base amount to account for the fee for on-campus accommodation and meals.

Without Inheritance

DESIGN 1

DS
register

NRS
register

RS
register

1. cohesion
2. single-choice P.

DESIGN 2

STUDENT

is-resident : BOOLEAN

T

F

kind : INTEGER

1

2

3

32

2. collection of students

1st Design Attempt

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
      across courses as c loop base := base + c.item.fee end
      Result := base * premium_rate
    end
end
```

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
      across courses as c loop base := base + c.item.fee end
      Result := base * discount_rate
    end
end
```

1st Design Attempt

Good design? ✓

Judge by Cohesion

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * premium_rate
    end
end
```

specific to RS

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * discount_rate
    end
end
```

specific to NRS

1st Design Attempt

Good design?

any duplicates?

Judge by Single Choice Principle
- A new kind is introduced?
- Change on registration policy?

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * premium_rate
    end
end
```

if cs.count < 7 then
cs.extend(c)
else ...
end

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * discount_rate
    end
end
```

unrelated.

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
      across courses as c loop base := base + c.item.fee end
      Result := base * premium_rate
    end
end
```

register

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
      across courses as c loop base := base + c.item.fee end
      Result := base * discount_rate
    end
end
```

register -

1st Design Attempt

Good design?

How do you build a **STUDENT_MANGEMENT_SYSTEM** class accordingly?

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * premium_rate
    end
end
```

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base * discount_rate
    end
end
```

RS NRS

class SMS Object

students : LL[ANY] RS NRS

static type

create {RS} rs . make (...)

create {NRS} nrs . make (...)

sms . add (rs)

sms . add (nrs)

ST. ~~student~~ ANY

sms.add(mazel)

① sms . students [1] . register

compile sms . x

② sms . students [1] . set_pr (7.3)

ad

Without Inheritance (Design 1) Collection of Students

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

duplicated?

Clinet's Code

```
c: COURSE
rs: RESIDENT_STUDENT
nrs: NON_RESIDENT_STUDENT
sms: SMS
create c.make("3311")
create sms.make
sms.add_rs(rs)
sms.add_nrs(nrs)
sms.register_all(c)
```

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

2nd Design Attempt

```
class
  STUDENT
create
  make
feature -- attribures
  courses: LINKED_LIST[COURSE]
  kind: INTEGER
  premiumRate: REAL
  discountRate: REAL
feature -- command
  make (kind: INTEGER)
    do
      kind := a_kind
    end
  ...
end
```

```
get_tuition: REAL
  local
    tuition: REAL
  do
    across courses is c loop
      tuition := tuition + c.fee
    end
    if kind = 1 then
      Result := tuition * premiumRate
    elseif kind = 2 then
      Result := tuition * discountRate
    end
  end
```

```
register (c: COURSE)
  local
    max: INTEGER
  do
    if kind = 1 then MAX := 6
    elseif kind = 2 then MAX := 4
    end
    if courses.count = MAX then -- Error
    else courses.extend (c)
    end
  end
```

R

NR

create {STUDENT} rs. make (1)

ns. make (2)

2nd Design Attempt

```
class
  STUDENT
create
  make
feature -- attribures
  courses: LINKED_LIST[COURSE]
  kind: INTEGER
  premiumRate: REAL
  discountRate: REAL
feature -- command
  make (kind: INTEGER)
    do
      kind := a_kind
    end
  ...
end
```

not belonging to the scope kind of student

```
get_tuition: REAL
  local
    tuition: REAL
  do
    across courses is c loop
      tuition := tuition + c.fee
    end
    if kind = 1 then
      Result := tuition * premiumRate
    elseif kind = 2 then
      Result := tuition * discountRate
    end
  end
```

```
register (c: COURSE)
  local
    max: INTEGER
  do
    if kind = 1 then MAX := 6
    elseif kind = 2 then MAX := 4
    end
    if courses.count = MAX then -- Error
    else courses.extend (c)
    end
  end
```

Good design?

Judge by **Cohesion** X

2nd Design Attempt

```
class
  STUDENT
create
  make
feature -- attribures
  courses: LINKED_LIST[COURSE]
  kind: INTEGER
  premiumRate: REAL
  discountRate: REAL
feature -- command
  make (kind: INTEGER)
    do
      kind := a_kind
    end
  ...
end
```

how to simulate OO using a non-OO language.

KS

URS

DS

```
get_tuition: REAL
  local
    tuition: REAL
  do
    across courses is c loop
      tuition := tuition + c.fee
    end
    if kind = 1 then
      Result := tuition * premiumRate
    elseif kind = 2 then
      Result := tuition * discountRate
    end
    elseif kind = 3 - - -
  end
```

```
register (c: COURSE)
  local
    max: INTEGER
  do
    if kind = 1 then MAX := 6
    elseif kind = 2 then MAX := 4
    end
    elseif kind = 3 - - -
    if courses.count = MAX then -- Error
    else courses.extend (c)
    end
  end
```

Good design?

Judge by Single Choice Principle

- A new kind is introduced?
- An existing kind is obeselete?

2nd Design Attempt

```
class
   STUDENT
create
   make
feature -- attribures
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
   premiumRate: REAL
   discountRate: REAL
feature -- command
   make (kind: INTEGER)
     do
       kind := a_kind
     end
   ...
end
```

max: ARRAY[INT]

tuition: ARRAY[REAL]

```
get_tuition: REAL
   local
     tuition: REAL
   do
     across courses is c loop
       tuition := tuition + c.fee
     end
     if kind = 1 then
       Result := tuition * premiumRate
     elseif kind = 2 then
       Result := tuition * discountRate
     end
   end
```

```
register (c: COURSE)
   local
     max: INTEGER
   do
     if kind = 1 then MAX := 6
     elseif kind = 2 then MAX := 4
     end
     if courses.count = MAX then -- Error
     else courses.extend (c)
     end
   end
```

INT

Good design?

How do you build a **STUDENT_MANGEMENT_SYSTEM** class accordingly?

Without Inheritance (Design 2) Collection of Students

```
class
  STUDENT_MANAGEMENT_SYSTEM
feature -- attributes
  students: LINKED_LIST[STUDENT]
feature -- command
  add_student(s: STUDENT)
     do
       students.extend(s)
     end
  register_all (c: COURSE)
    do
      across students is s
        loop
          s.register(c)
        end
    end
end
```

violates SCP.

Clinet's Code

```
c: COURSE
rs: STUDENT
nrs: STUDENT
sms: SMS
create c.make("3311")
create sms.make
Create rs.make(1)
Create nrs.make(2)

sms.add_student(rs)
sms.add_student(nrs)
sms.register_all(c)
```

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

LECTURE 15

MONDAY MARCH 2

- Office Hours Change (for the rest of the term)
  Monday's office hours moved to:
  12:30 to 14:30 on Tuesdays

- Lab 4 due 3pm Friday March 13

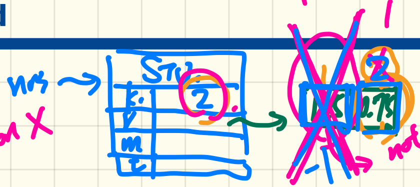
- Automated regression testing

Design Attempt 2.5

```
class
  STUDENT
create
  make
feature -- kind-specific attributes
  rate: LINKED_LIST[REAL]
  tuition: LINKED_LIST[REAL]
  max: LINKED_LIST[INTEGER]
feature -- attributes
  courses: LINKED_LIST[COURSE]
  kind: INTEGER
feature -- command
  make (kind: INTEGER)
    do
      kind := a_kind
      rate := << 1.25, 0.75 >>
      max := << 6, 4 >>
      tuition := <<0.0, 0.0 >>
    end
  ...
end
```

size of lists corresponds to # of kinds

RS obselete



cohesion X

not applicable for NRS

```
get_tuition: REAL
  do
    across courses is c loop
      tuition[kind] :=
        tuition[kind] + c.fee
    end
    tuition[kind] :=
      tuition[kind] * rate[kind]
  end
```

```
register (c: COURSE)
  do
    if courses.count = MAX then -- Error
    else courses.extend (c)
    end
  end
```

Good design?

Judge by Single Choice Principle

- A new kind is introduced?
- An existing kind is obeselete?

Design 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

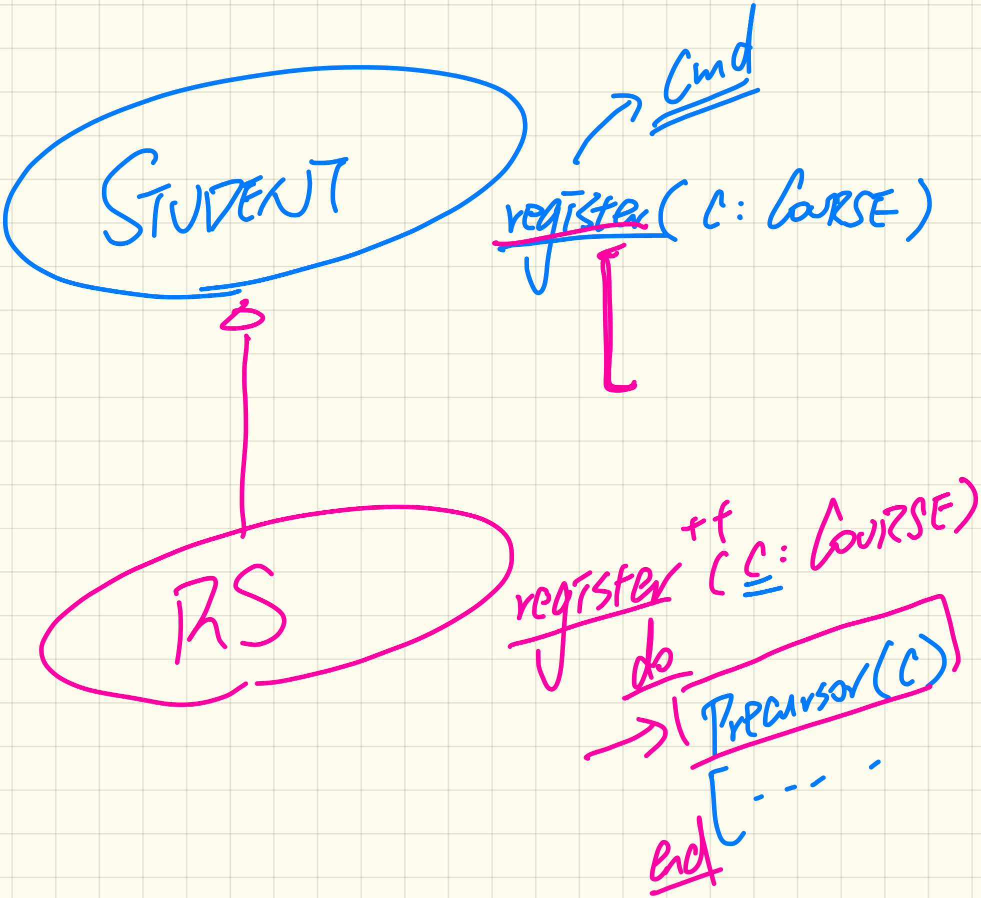
```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
      across courses as c loop base := base + c.item.fee end
      Result := base
    end
end
```

Precursor

```
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
```

version of tuition from the parent 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
```



STUDENT
Cmd
register (c: COURSE)
RS
register +f (c: COURSE)
do
Precursor (c)
...
end

A

f

B

nf

f ++f

redefined.

DC. 1

obj : A

static type

obj.f ✓

obj.nf ✗

DC. 2

obj : B

obj.f ✓

obj.nf ✓

DC 1.

a : ARRAY[A]

a[i].f ✓

a[i].nf ✗

DC 2

a : ARRAY[B]

a[i].f ✓

a[i].nf ✓

Desigh 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base
    end
end
```

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

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

Desigh 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base
    end
end
```

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

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

With Inheritance (Design 3) Collection of Students

```
class
   STUDENT_MANAGEMENT_SYSTEM
feature -- attributes
   students: LINKED_LIST[STUDENT]
feature -- command
   add_student(s: STUDENT)
      do
         students.extend(s)
      end
   register_all (c: COURSE)
     do
       across students is s
          loop
             s.register(c)
          end
     end
end
```

```
c: COURSE
rs: STUDENT
nrs: STUDENT
sms: SMS
create c.make("3311")
create sms.make

sms.add_student(rs)
sms.add_student(nrs)
sms.register_all(c)
```

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

Static Type vs. Dynamic Type

static

dynamic

- In Java:

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

static

dynamic.

- 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")
```

ST

=

create {STUDENT} s.make(...)

DT

Inheritance :

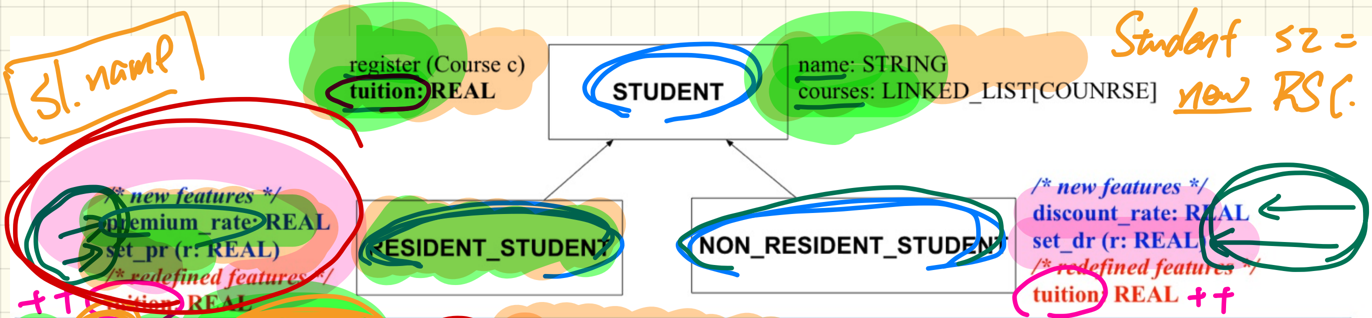
1. Does the code compile? → consider static type

2. If it compiles, how does it behave (e.g. version of feature, cast exception?) → both static & dynamic types

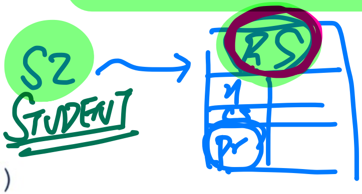
Student Classes (with Inheritance): Expectations

s1.name

Student s2 = new RS(...);

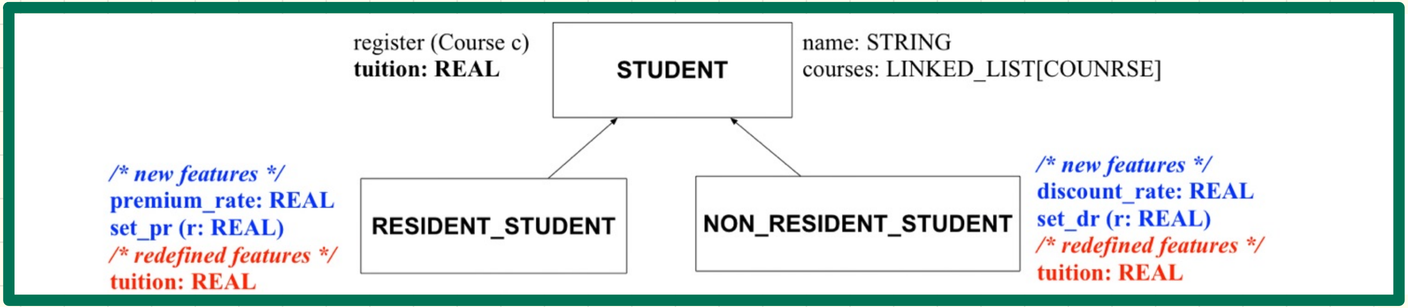



```
s1, s2, s3: STUDENT ; rs: RESIDENT_STUDENT ; nrs : NON_RESIDENT_STUDENT
create {STUDENT} s1.make ("S1")
create {RESIDENT_STUDENT} s2.make ("S2")
create {NON_RESIDENT_STUDENT} s3.make ("S3")
create {RESIDENT_STUDENT} rs.make ("RS")
create {NON_RESIDENT_STUDENT} nrs.make ("NRS")
```

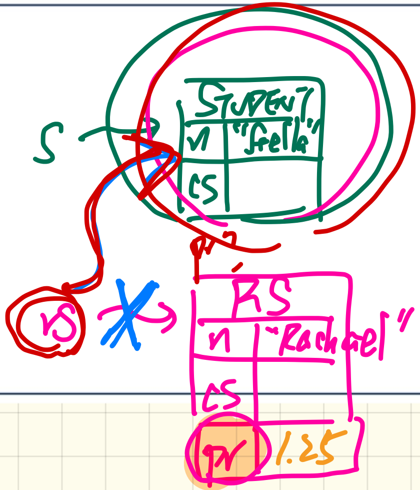


	name	courses	reg	tuition	pr	set_pr	dr	set_dr
s1								
s2								
s3								
rs								
nrs								

Polymorphism: Intuition




```
1  local
2    s: STUDENT
3    rs: RESIDENT_STUDENT
4  do
5    create s.make ("Stella")
6    create rs.make ("Rachael")
7    rs.set_pr (1.25)
8    s := rs /* Is this valid? */
9    rs := s /* Is this valid? */
```



Proof by Contradiction

1. Assume L9 compiled.
2. ⇒ rs points to the STUDENT object
3. Expectations on rs? name, pr, cs
4. $\boxed{rs.pr}$ → crash.

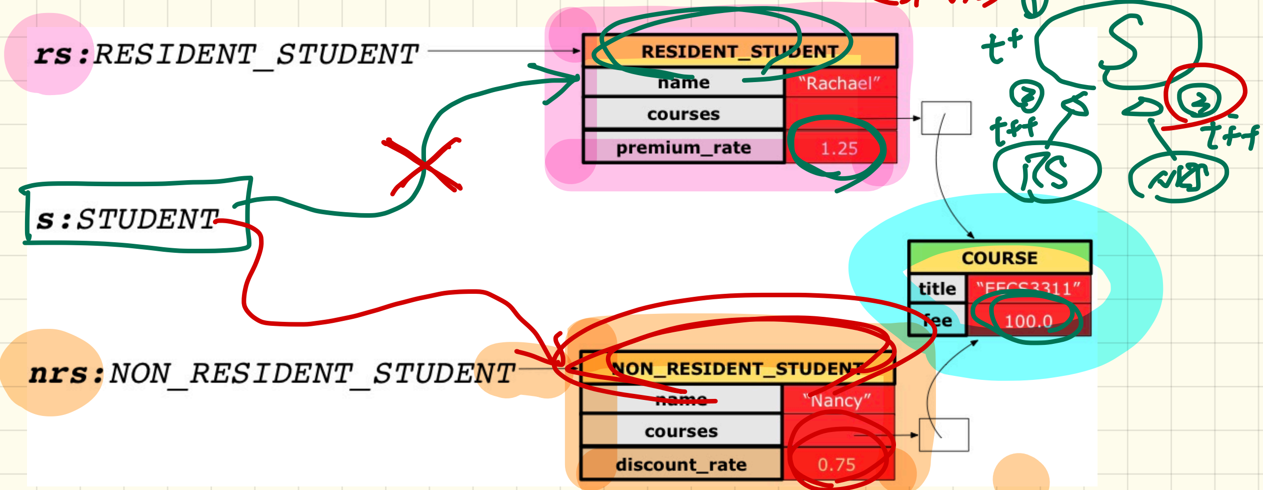
Dynamic Binding: Intuition

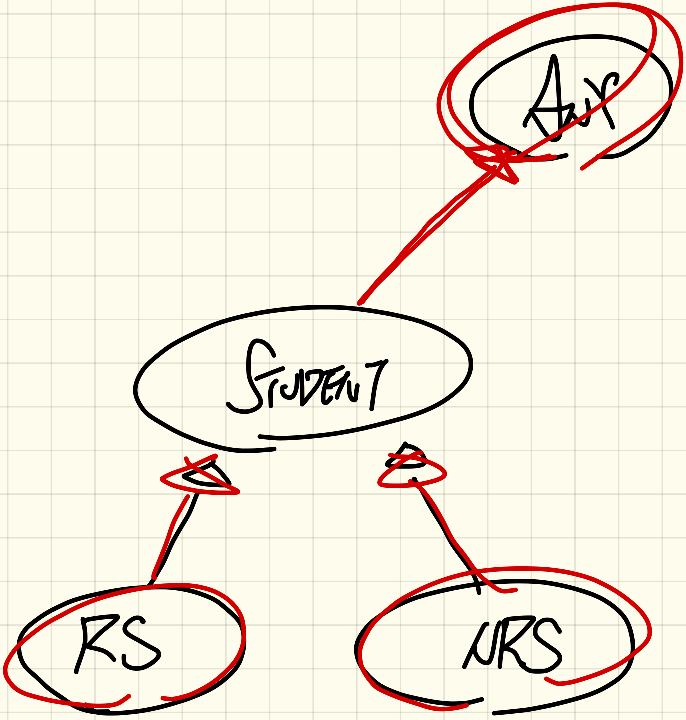
```
local c : COURSE ; s : STUDENT
do crate c.make ("EECS3311", 100.0)
   create {RESIDENT_STUDENT} rs.make("Rachael")
   create {NON_RESIDENT_STUDENT} nrs.make("Nancy")
   rs.set_pr(1.25); rs.register(c)
   nrs.set_dr(0.75); nrs.register(c)
   s := rs; ; check s .tuition = 125.0 end
   s := nrs; ; check s .tuition = 75.0 end
```

rs: RS ; nrs : NRS

S: ST = STUDENT
DT = RS

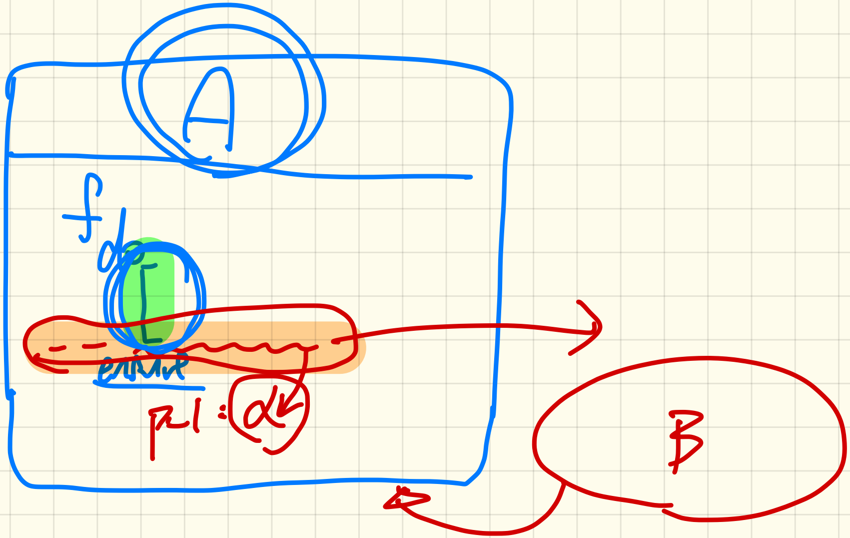
S: ST = STUDENT
DT = NRS

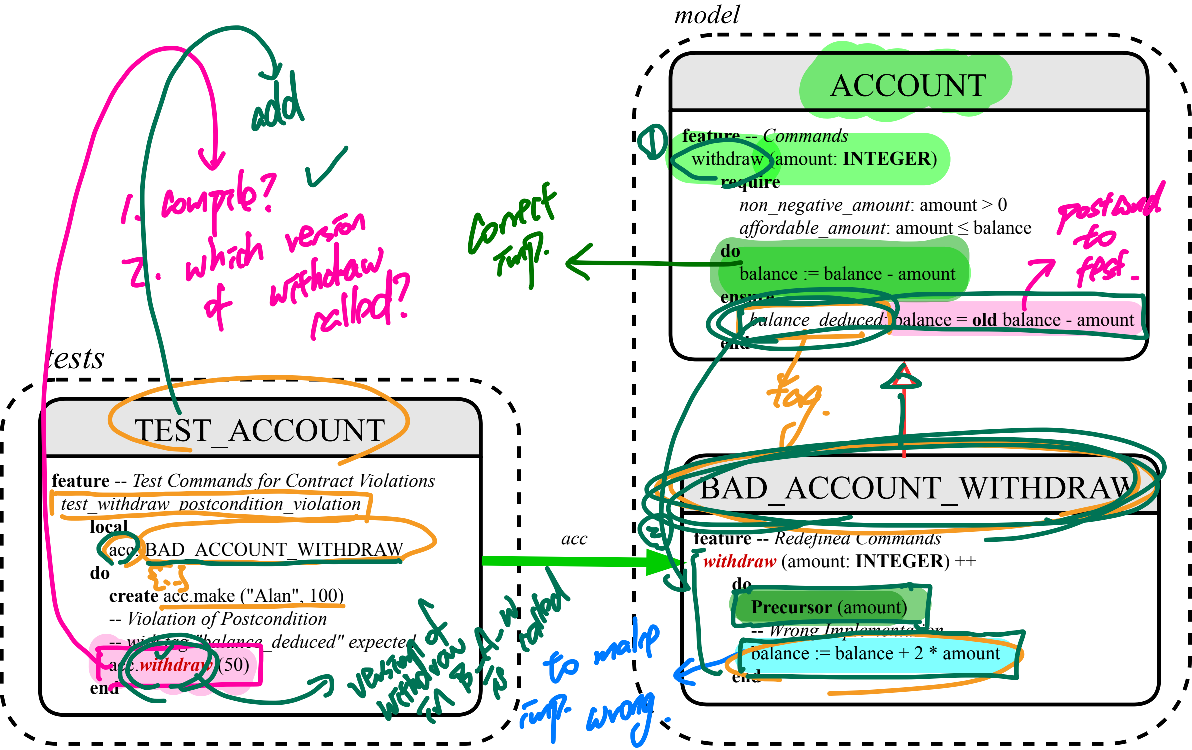




Aux
STUDENT
RS
NRS

Testing of Postcondition.





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
BAD_ACCOUNT_WITHDRAW
feature -- Redefined Commands
withdraw (amount: INTEGER) ++
do
Precursor (amount)
-- Wrong Implementation
balance := balance + 2 * 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
add
1. compile? ✓
2. which version of withdraw called?
Correct Imp.
postcond. to fail.
tag.
version of withdraw in B_A_W is called
to make imp. wrong.

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
```

Testing of Postcondition: Exercise

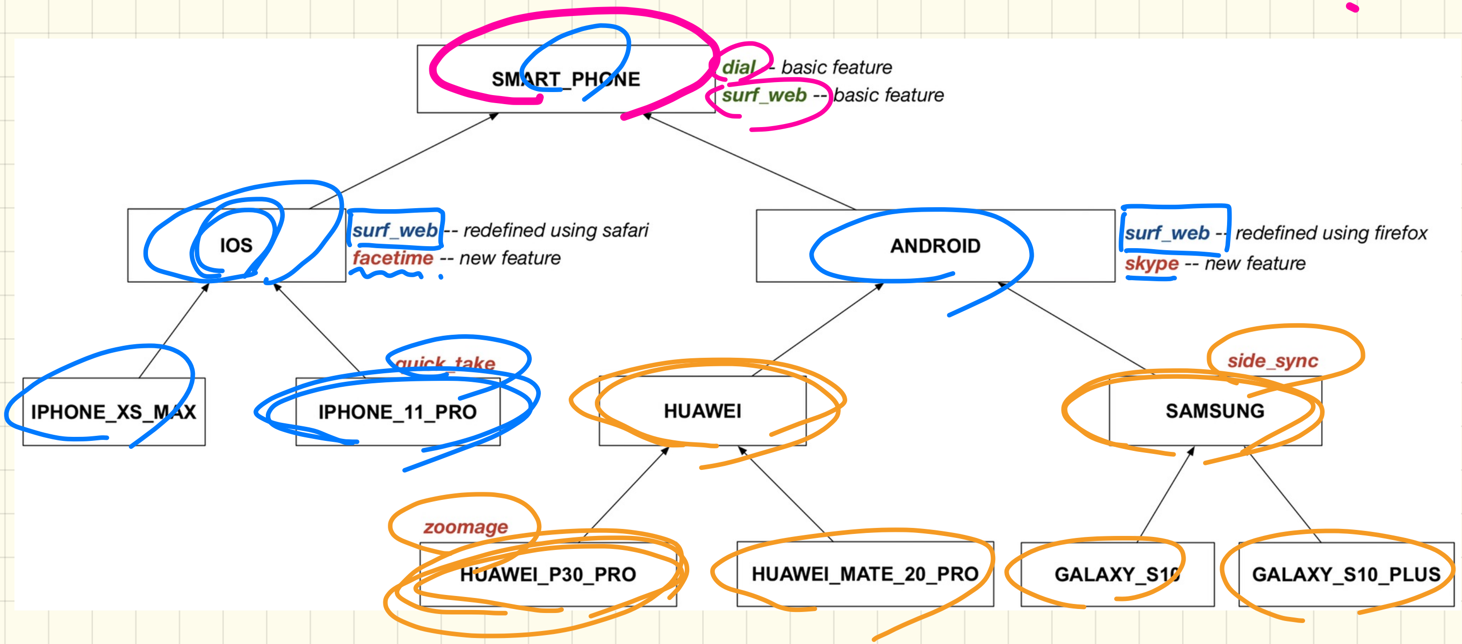
```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    do ... -- Put Correct Implementation Here.
    ensure
      ...
      others_unchanged :
        across old accounts.deep_twin as cursor
        all cursor.item.owner /~ n implies
              cursor.item ~ account_of (cursor.item.owner)
  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

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

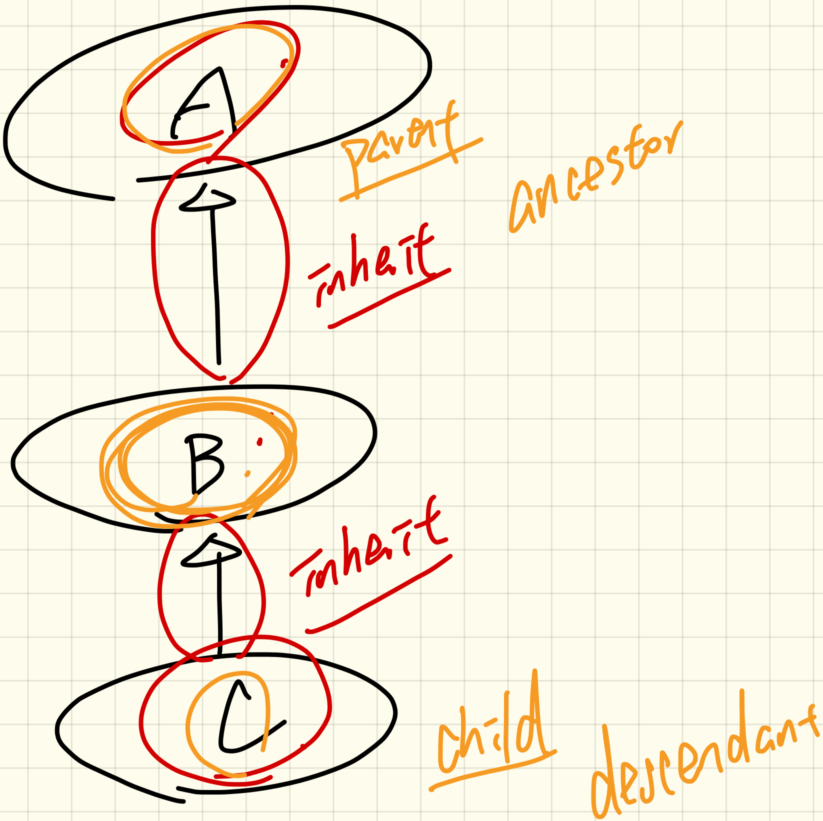
TEST

Lecture 16
Wednesday March 4

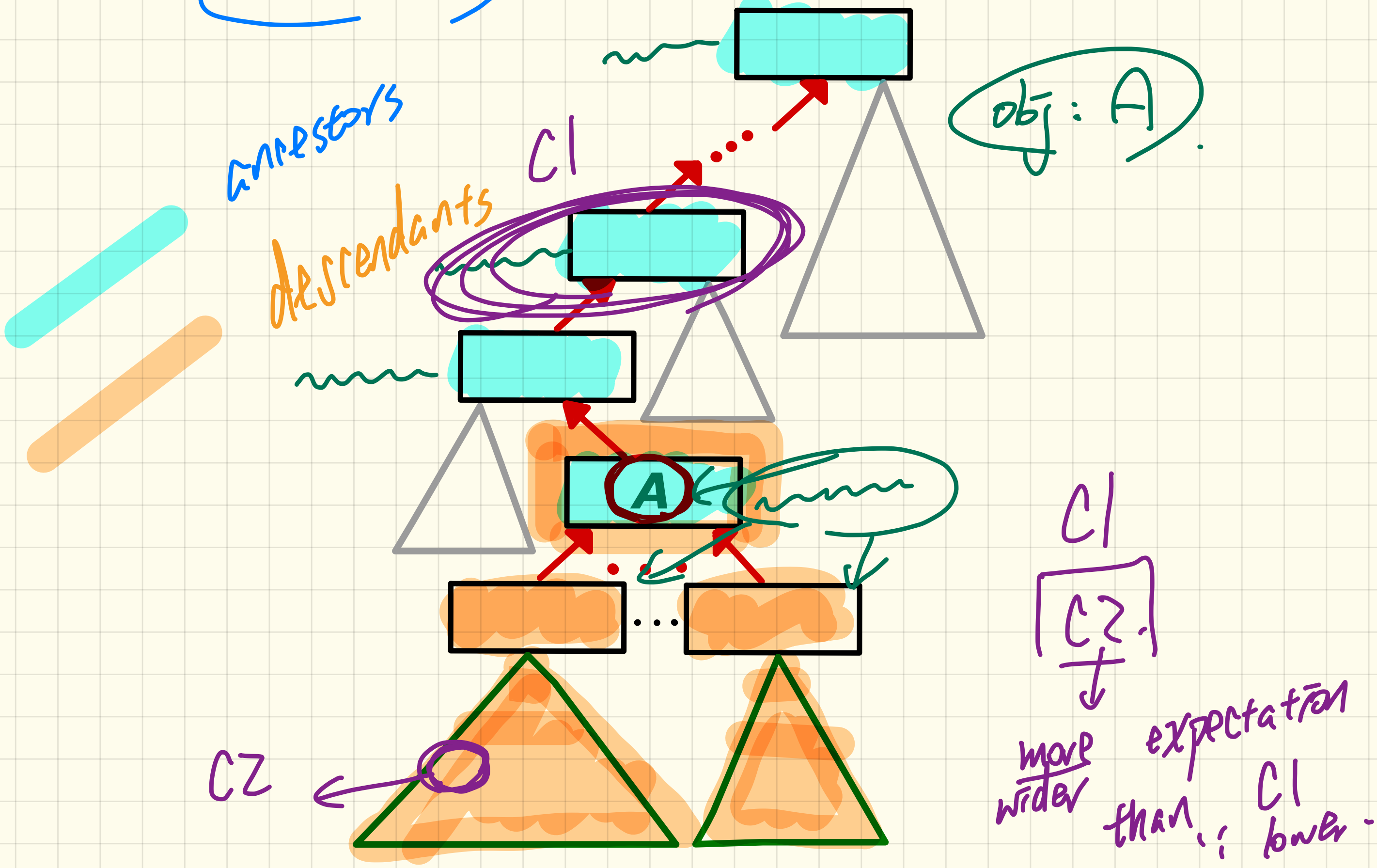
Multi-Level Inheritance Hierarchy of Smartphones

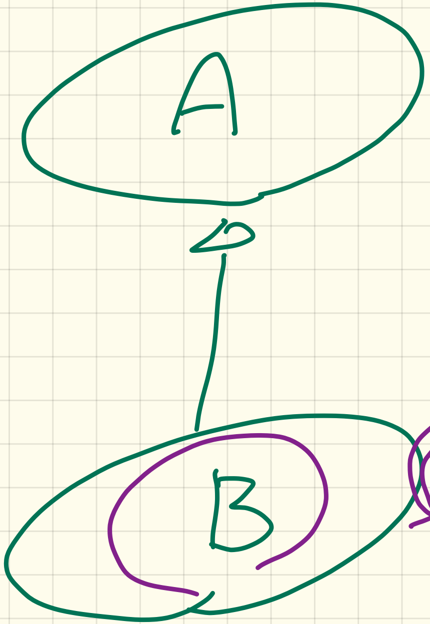


$$\frac{C < B \qquad B < A}{C < A}$$



Ancestors, Expectations, Descendants, and Code Reuse





f

require

pre_1

ensure

post_1

f — what will be checked

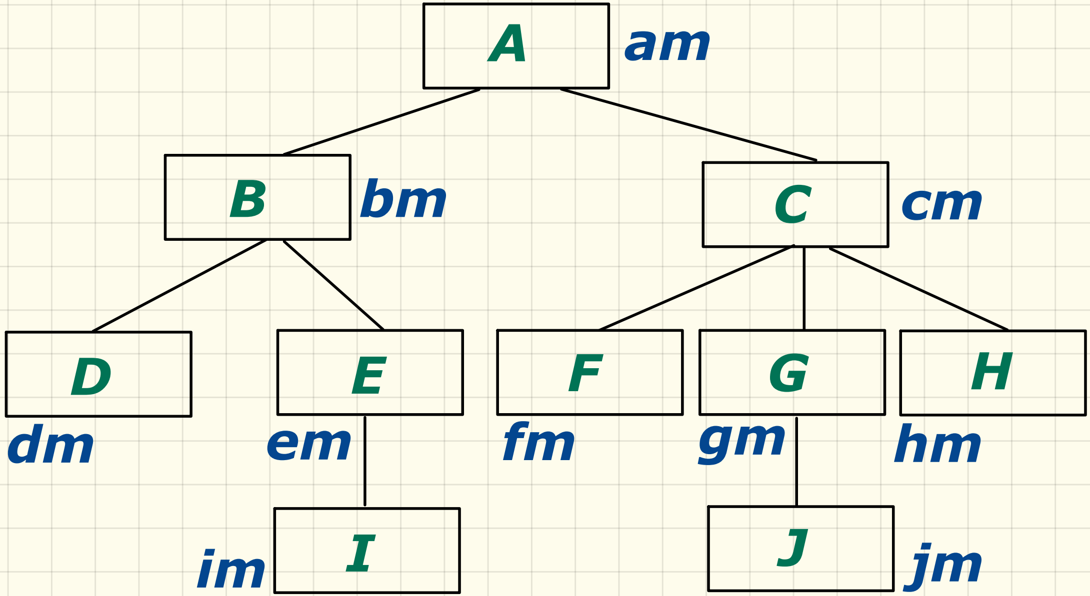
require

pre_2 $\vee$ pre_1

ensure

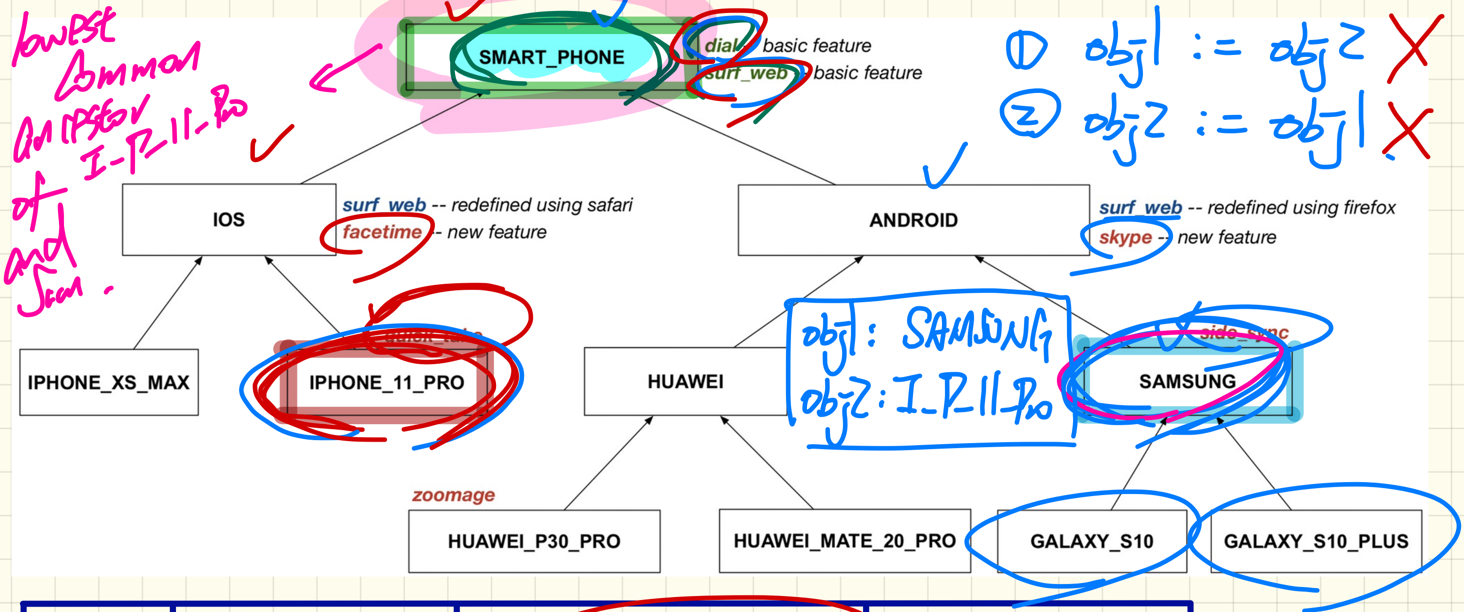
post_2 $\wedge$ post_1

Inheritance Forms a Type Hierarchy (1)



	ancestors	expectations	descendants
B			
G			
J			

Inheritance Forms a Type Hierarchy (2)



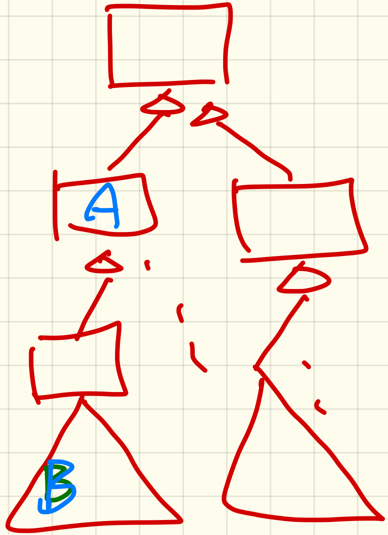
	ancestors	expectations	descendants
	S_P	dial, surf_web	all classes in hierarchy.
	S, A, S_P	dial, surf_web, skype, side_sync	3 classes
	I_11_P, IOS, S_P	dial, surf_web, face, q_t	

obj1: A

obj2: B

ST? A ST? B

obj1 := obj2



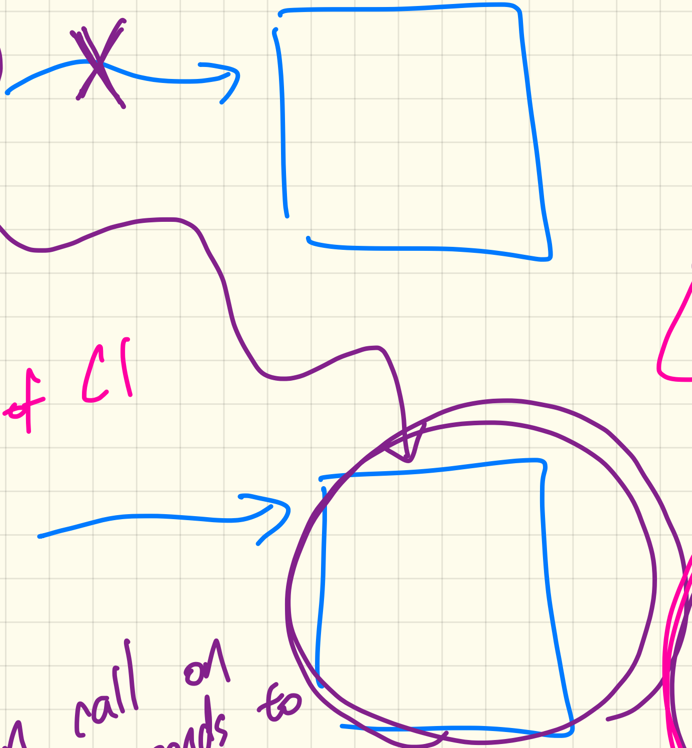
Q: compile?

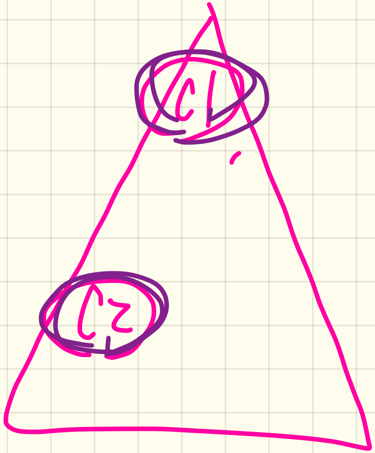
expectation of B should be <u>at least</u> as wide/many as the expec. of A.

obj1: C1

obj2: C2

obj1

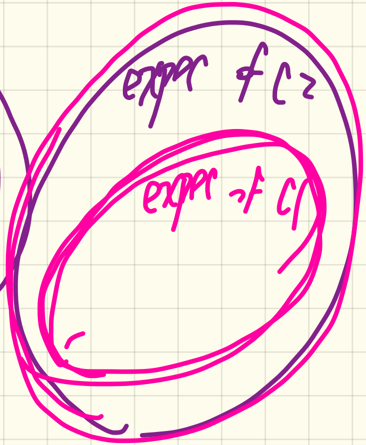




obj1 := obj2

↳ C2 should be a descendant of C1

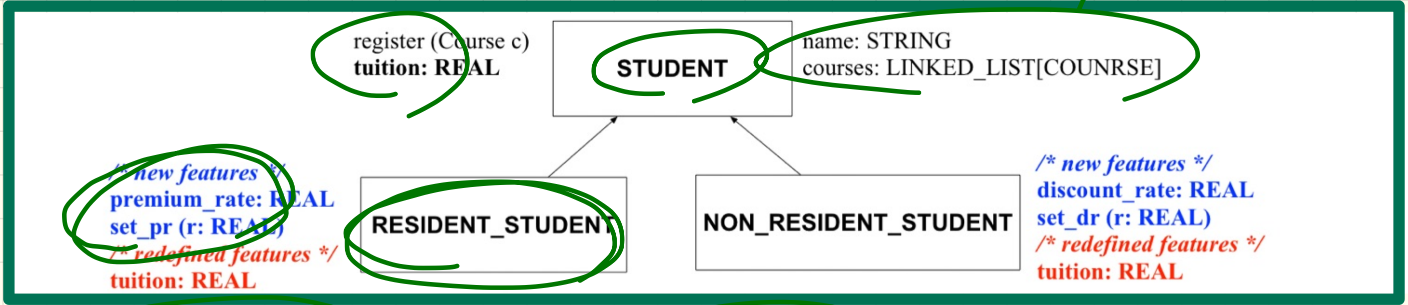
obj2



after the :=, what we can call on obj1 corresponds to ST of obj1 (C1).

Reference Variables: Static Type

expectations



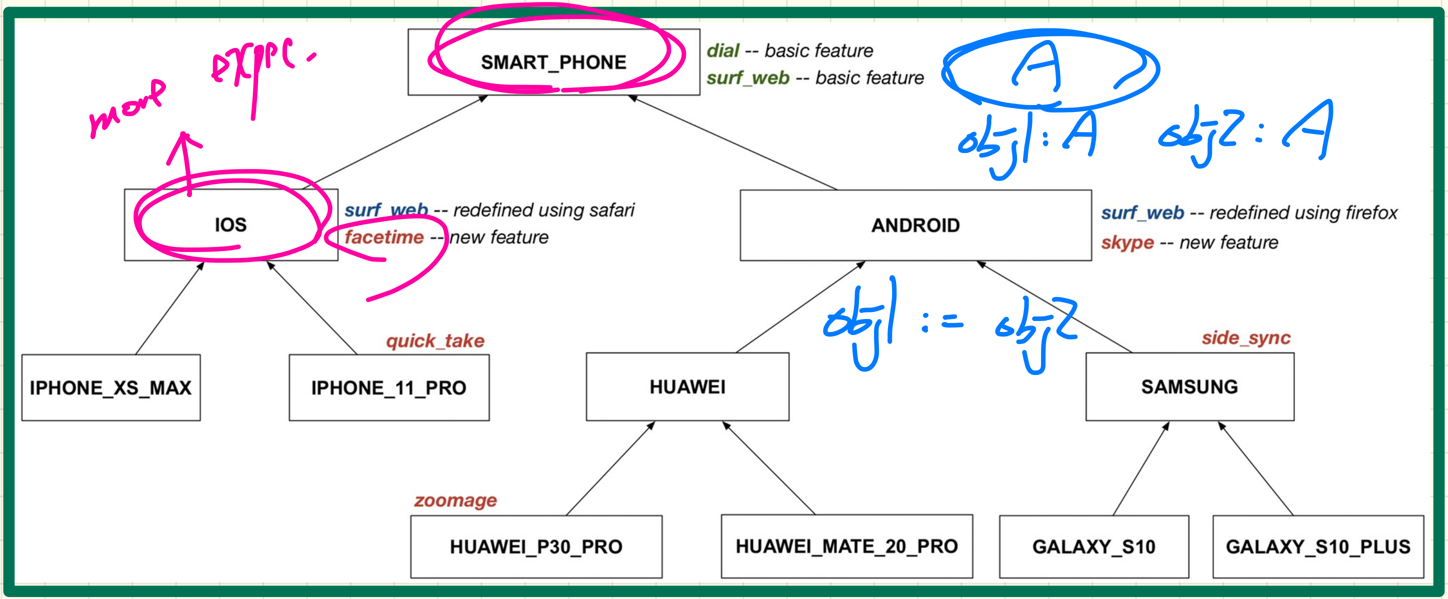
Design 1:	Design 2:
jim: **STUDENT**	jim: **RESIDENT_STUDENT**

more expec

jim . r
t
n
c_pr ✗

jim . r
t
n
c_pr ✓
set_pr ✓

Reference Variables: Static Type



Design 1:

mp: **SMART_PHONE**

mp.facetime X

Design 2:

mp: **IOS**

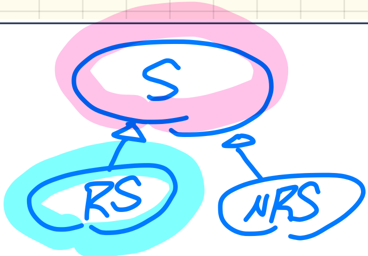
mp.facetime ✓

Change of Dynamic Type

```
test_polymorphism_students
  local
    jim: STUDENT
    rs: RESIDENT_STUDENT
    nrs: NON_RESIDENT_STUDENT
  do
    create {STUDENT} jim.make ("J. Davis")
    create {RESIDENT_STUDENT} rs.make ("J. Davis")
    create {NON_RESIDENT_STUDENT} nrs.make ("J. Davis")
    jim := rs   ✓
    rs := jim   ×
    jim := nrs   ✓
    rs := jim   ×
  end
```

ST: STUDENT

ST: RS



STUDENT	
n.	
cs.	

RESIDENT_S.	
n.	
cs.	
pr.	

NON_RESI_S.	
n.	
cs.	
dr.	

Testing of Dynamic Binding

RESIDENT_S.	
n.	
cs.	
pr.	

NON_RESI_S.	
n.	
cs.	
dr.	

STUDENT	
n.	
cs.	

```
test_dynamic_binding_students: BOOLEAN
  local
    jim: STUDENT
    rs: RESIDENT_STUDENT
    nrs: NON_RESIDENT_STUDENT
    c: COURSE
  do
    create c.make ("EECS3311", 500.0)
    create {STUDENT} jim.make ("J. Davis")
    create {RESIDENT_STUDENT} rs.make ("J. Davis")
    rs.register (c)
    rs.set_pr (1.5)
    jim := rs
    Result := jim.tuition = 750.0
    check Result end
    create {NON_RESIDENT_STUDENT} nrs.make ("J. Davis")
    nrs.register (c)
    nrs.set_dr (0.5)
    jim := nrs
    Result := jim.tuition = 250.0
end
```

ST.

S

RS

NRS

RS is a dec. class of the ST of rs.

```
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
```

COURSE	
t.	
fee	

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

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

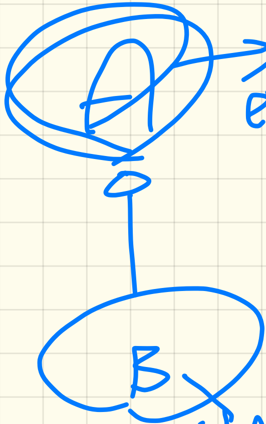
obj1 : A ↳ ST

⋮

→ Create {B} obj1.make

↓ ↳ DT.

obj1 → B -



expectation of obj1

can support at least as many features

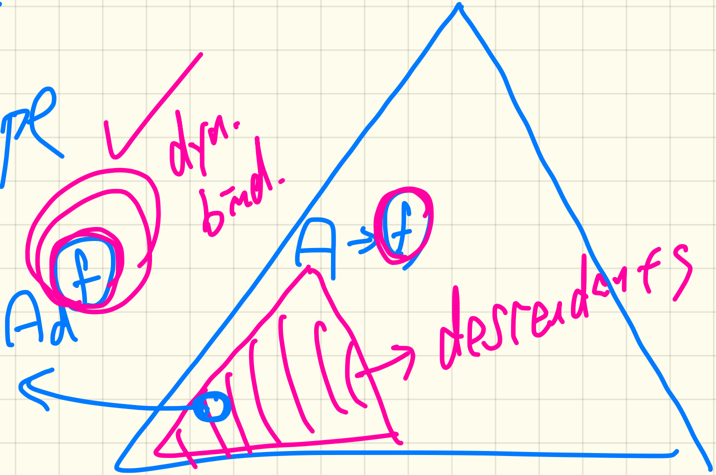
depends on A

obj1. ___?___ (ST)

Polymorphism

Poly → multi

morphism → shape



obj : A

? → any descendant class of A.

Create {Ad} obj. make(. ~)

obj. f

Ad → must support at least as many prope[rties] as A

Create {B}

Create {A}

obj1. make ✓

obj2. make

A

f+

[obj1 : A

obj1 : B

obj2 := obj1

✓ Current

expec. on A:

{f}

B

f++

✓ Current

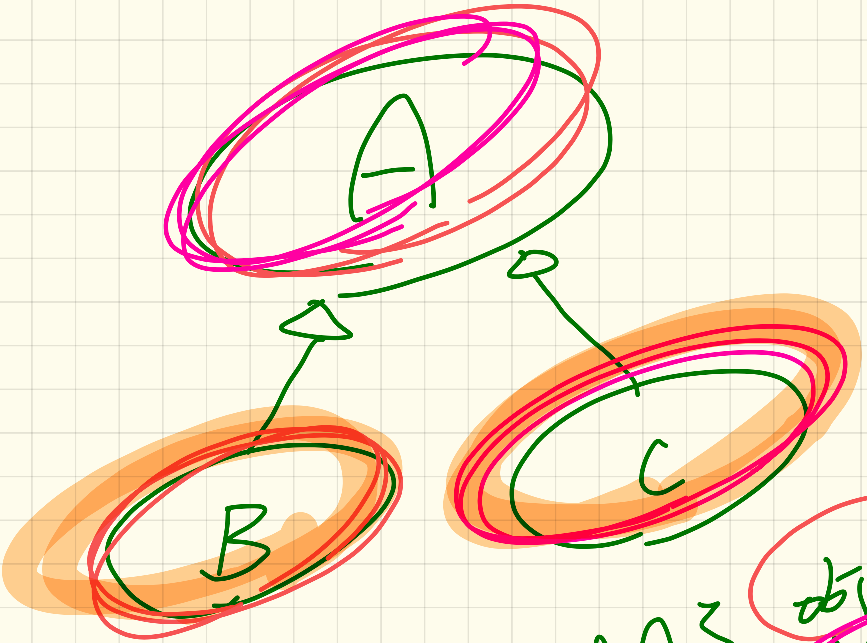
expec. on B:

{f}

g

g.

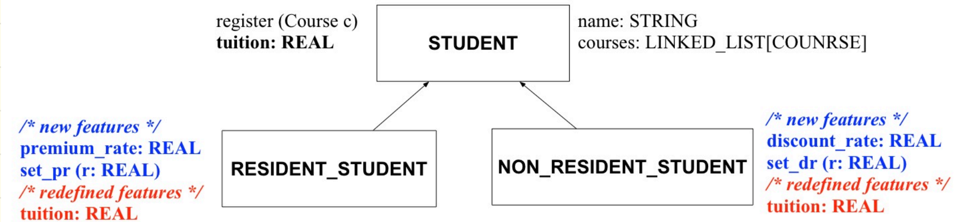
complication



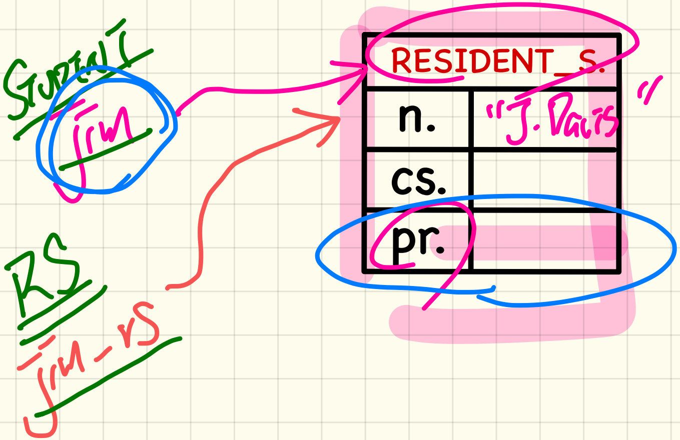
obj1 : A
obj2 : B
obj3 : C

Create {A} obj2. make (--)
Create {A} obj3. make (--)
Create {B} obj3. make (---)

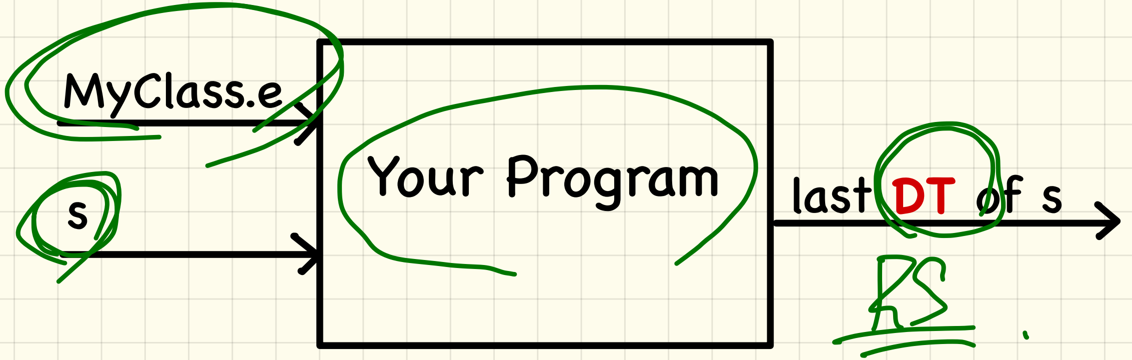
Type Cast: Motivation




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



Inferring the DT of a Variable is Undecidable




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

while ( )

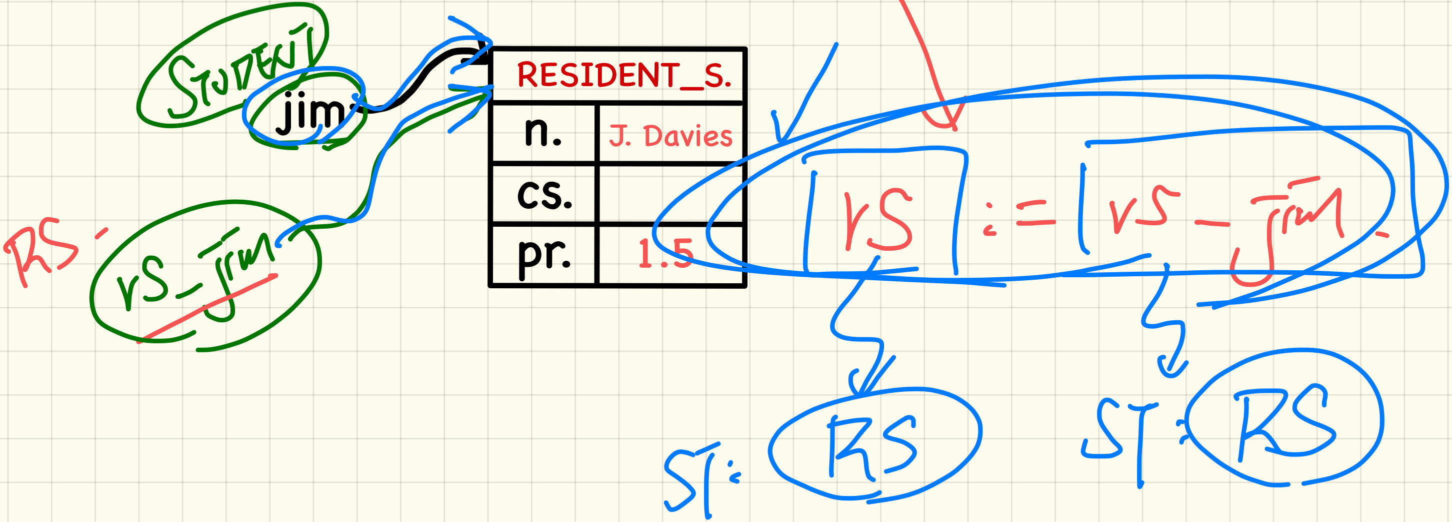
s := new RS(..);

Type Cast: Syntax

T or F.

alias of jim with ST

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



F

```
if attached {RS} as rs_jrm then
    X
    ⋮
        rs_jrm
end
```

F → check assertion

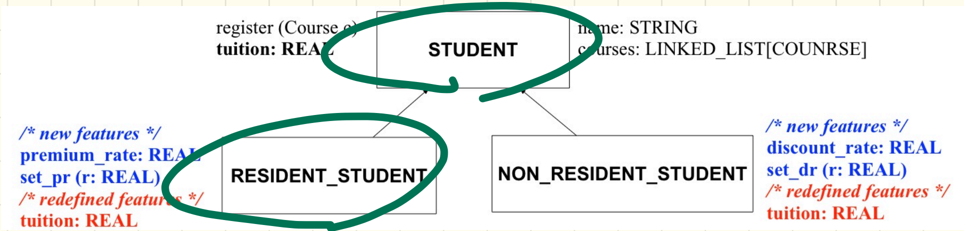
```
check attached {RS} as rs_jrm then
    ⋮
        rs_jrm
end
```

LECTURE 17
MONDAY MARCH 9

Labtest 2 (course wiki/forum):

- undo/redo design pattern
- Reading: OOSC Ch. 21
- Exercise from Github

Type Cast: Motivation



RS S

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

X

STUDENT jim

RS rs_jim

RESIDENT_S.	
n.	
cs.	
pr.	

alias

ST

check attached {RS} jim as rs_jim then

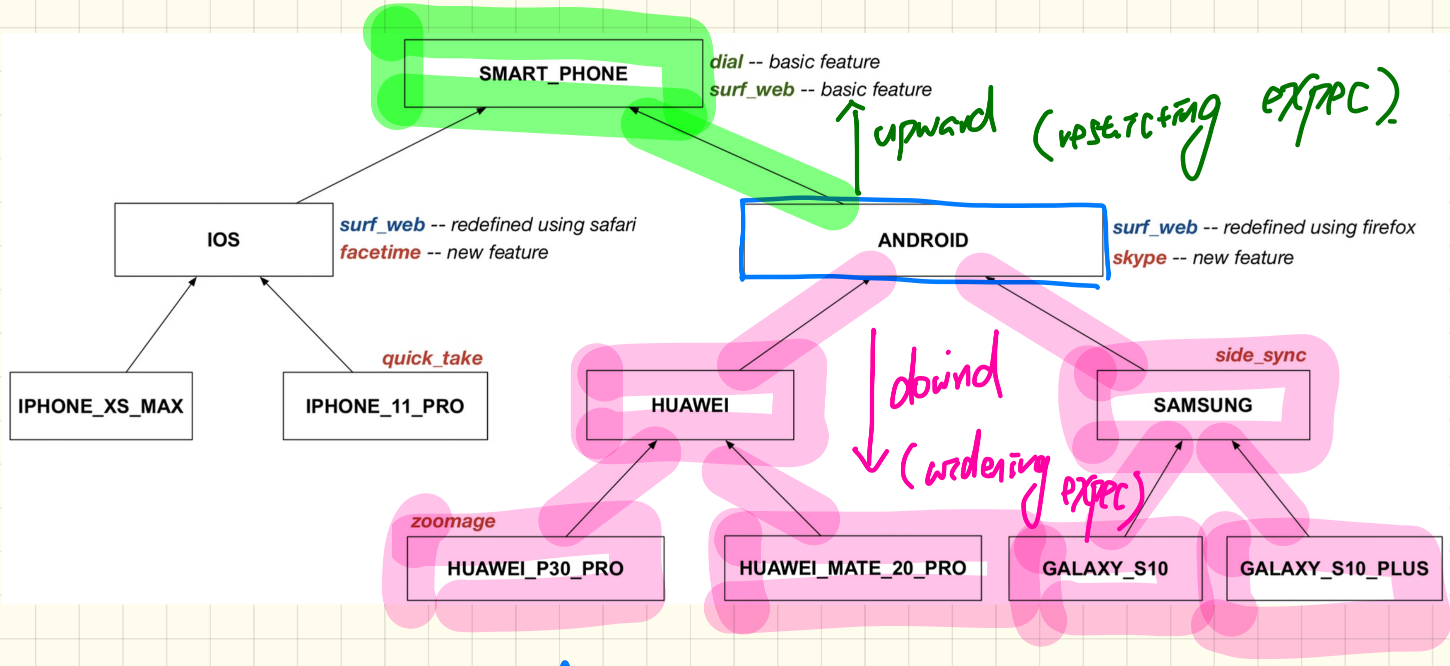
rs := rs_jim

ST: RS ST: RS

end

rs_jim X

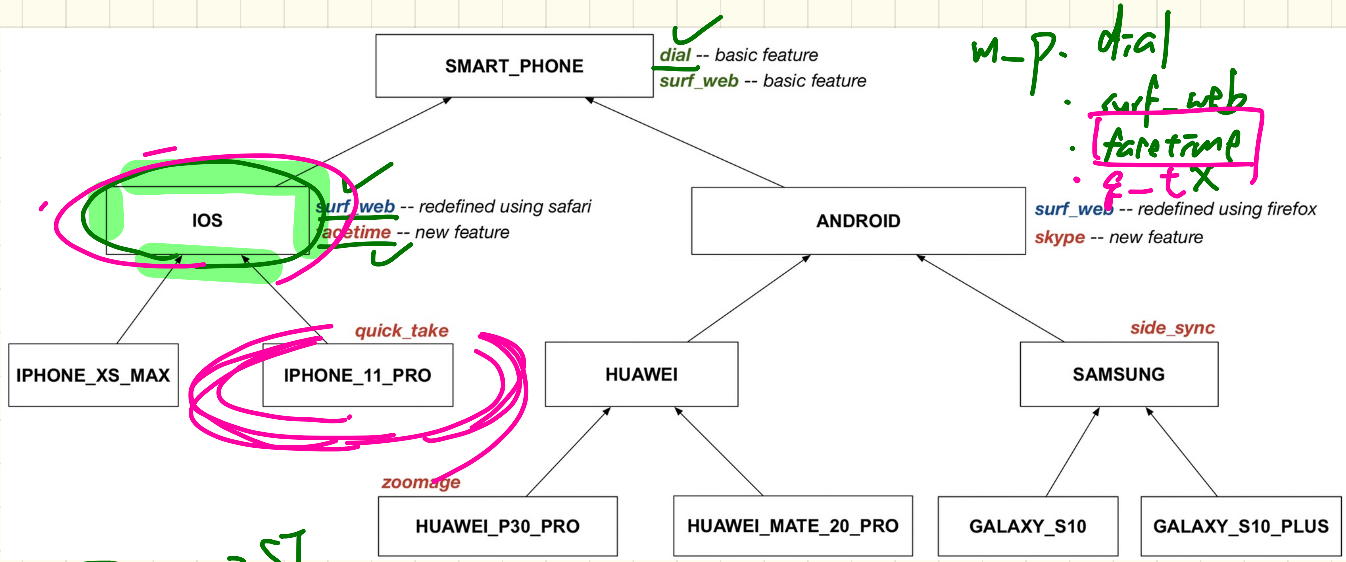
Multi-Level Inheritance Hierarchy of Smartphones



P : ANDROID

↳ ST

Violation-Free Cast: Upwards or Downwards (1)

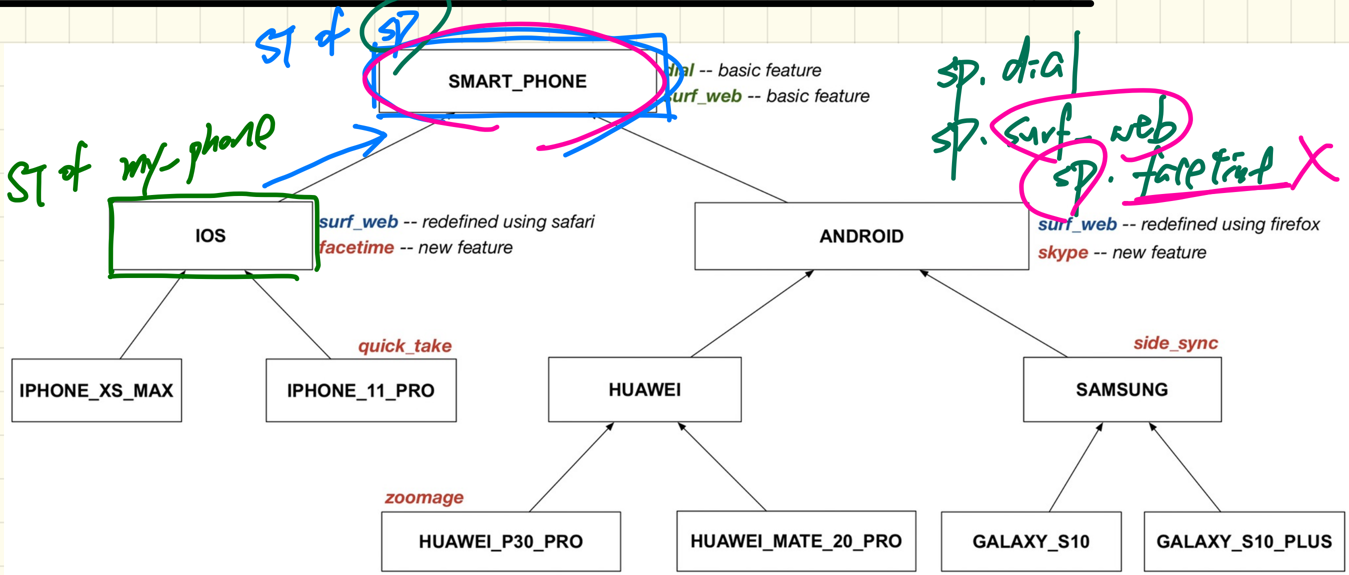


ST

DT

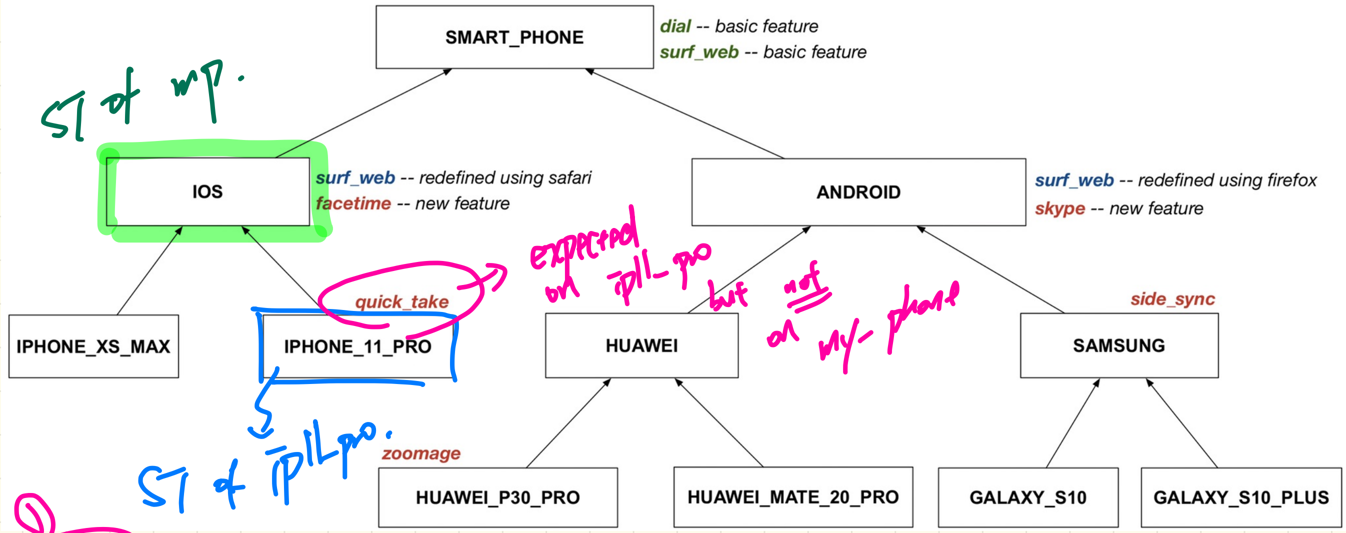
```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime [redacted] quick_take, skype, side_sync, zoomage [redacted]
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on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

Violation-Free Cast: Upwards or Downwards (2)




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

Violation-Free Cast: Upwards or Downwards (3)

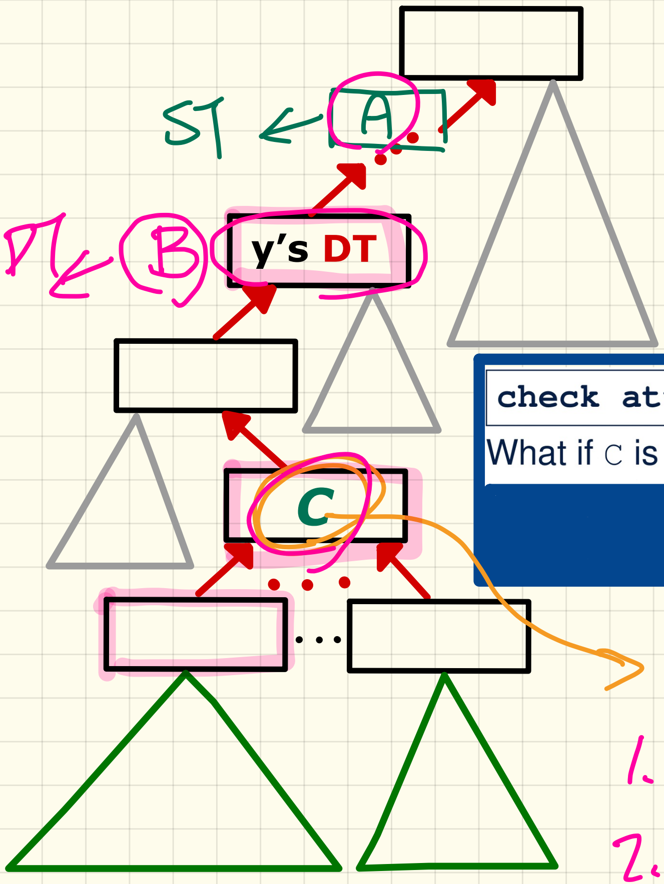


1

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

2

Ancestors, Expectations, Descendants, and Code Reuse



obj : A

create {B} obj.make

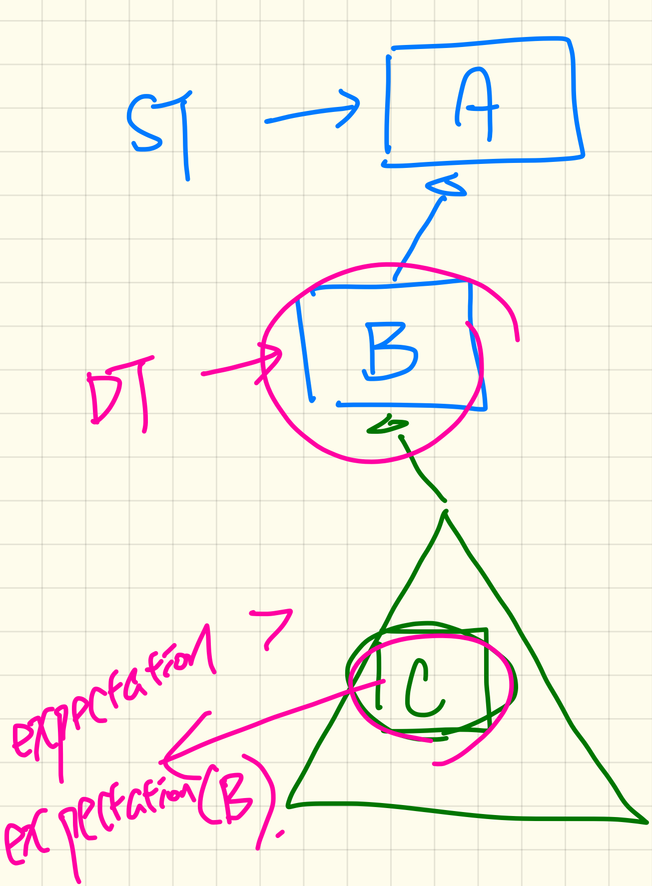
`check attached {C}` ~~`y`~~ obj `then ... end`	*always compiles*

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

[redacted]

the type to cast obj into

1. Casting obj downto C compiles
2. Runtime?



obj : A

create {B} obj.make

check attached {C} obj as c_obj

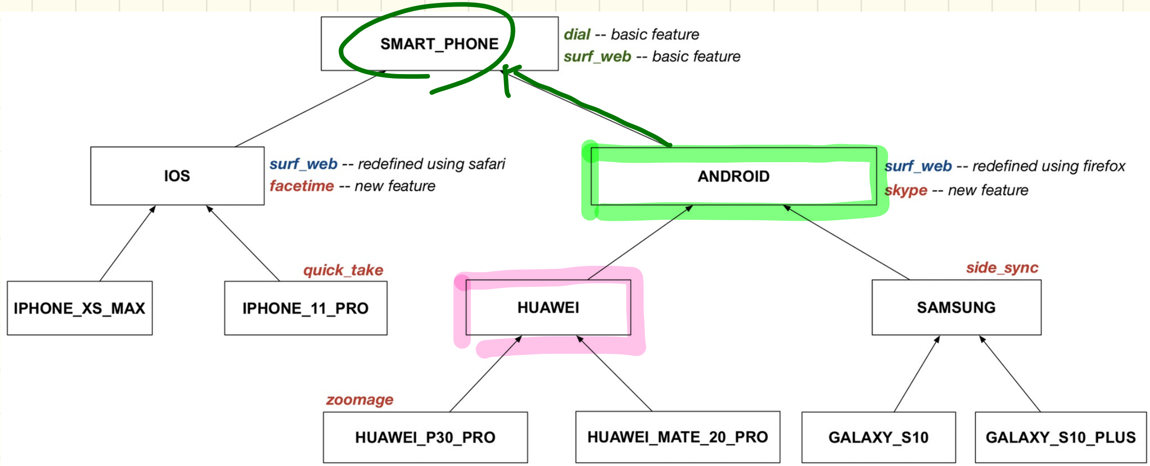
c_obj

end

↳ cast violation at runtime

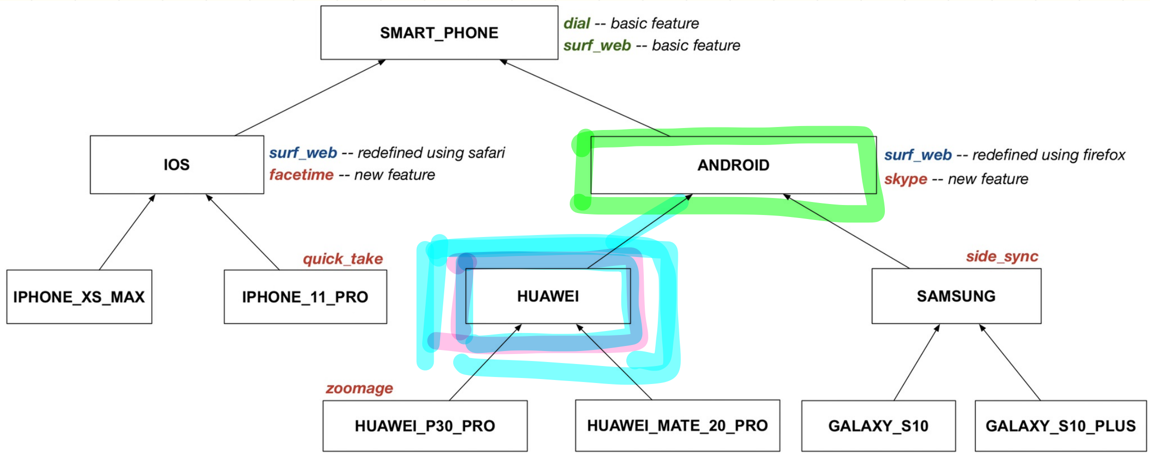
∵ c_obj would be expected to be called features from C

Cast Violation at Runtime (1)



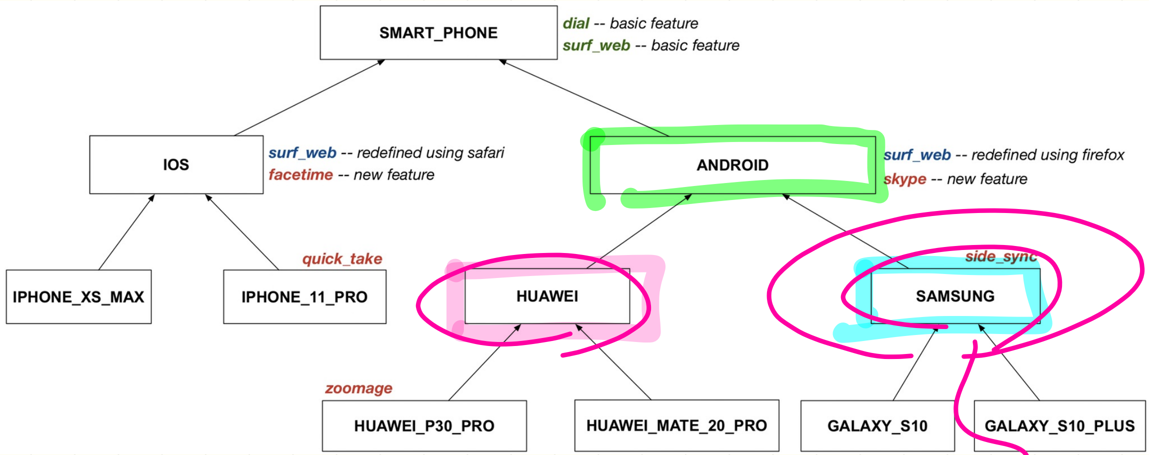

```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {HUAWEI} mine.make
    -- ST of mine is ANDROID; DT of mine is HUAWEI
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is HUAWEI
    check attached {HUAWEI} mine as huawei then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

Cast Violation at Runtime (2)




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

Cast Violation at Runtime (3)

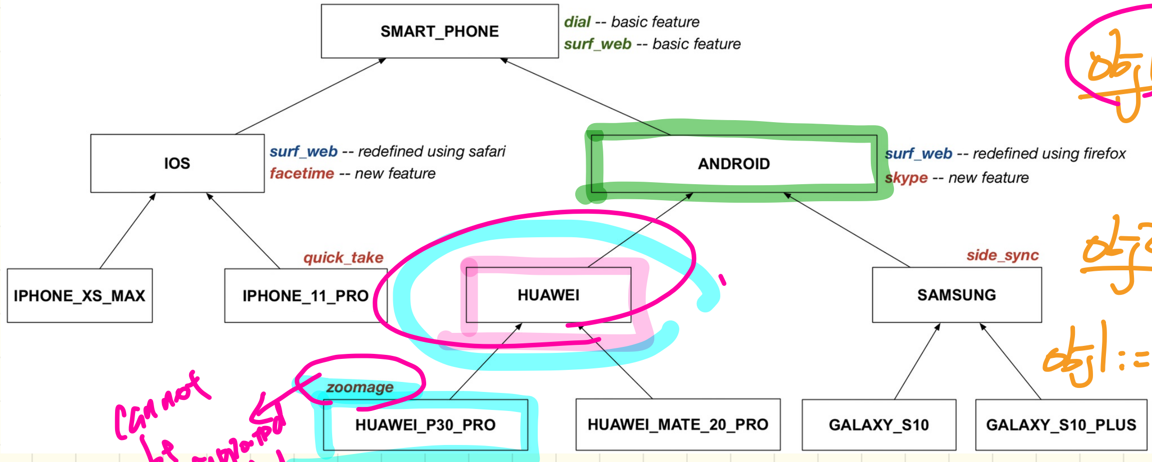


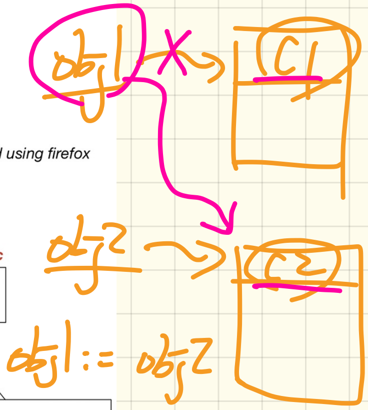

```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {HUAWEI} mine.make
    -- ST of mine is ANDROID; DT of mine is HUAWEI
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is HUAWEI
    check attached {HUAWEI} mine as huawei then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

Runtime violation

∵ DT cannot support expec on SAMSUNG.

Cast Violation at Runtime (4)





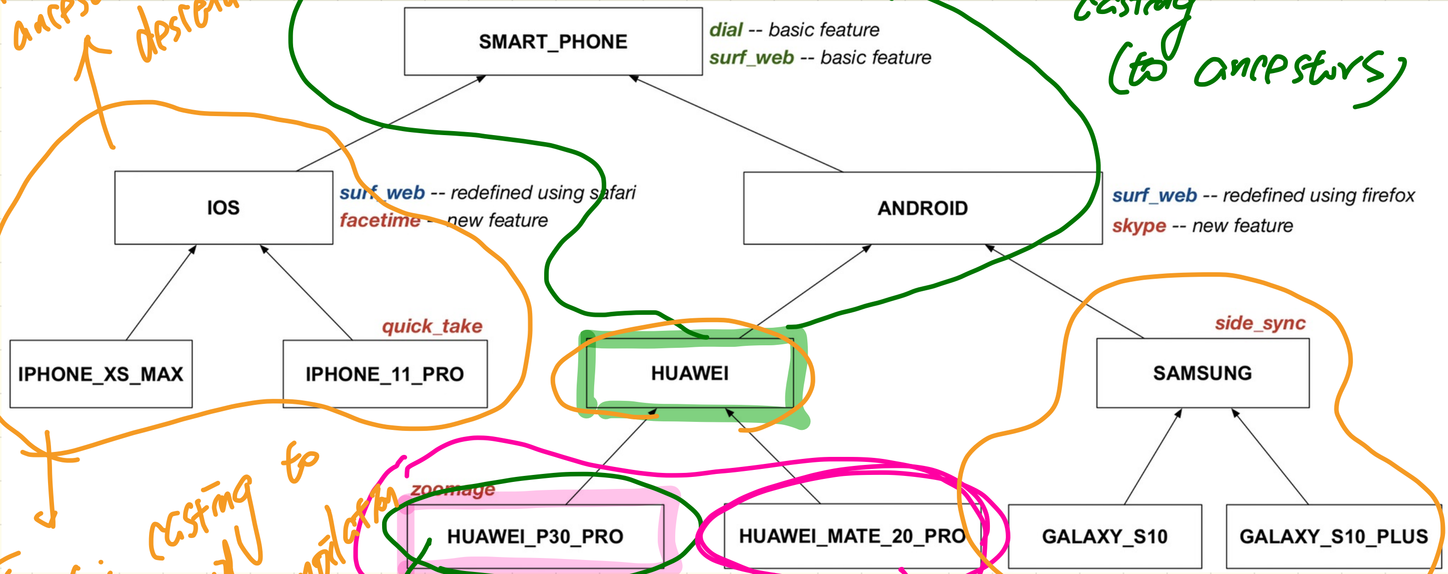

```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {HUAWEI} mine.make
    -- ST of mine is ANDROID; DT of mine is HUAWEI
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is HUAWEI
    check attached {HUAWEI} mine as huawei then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

Rule for avoiding RT cast violation

Do Not cast lower than DT

neither ancestors nor descendants.

upward casting (to ancestors)



Java: casting to this category results in a compilation error

Eiffel: compile.

can cast to any of DT's ancestors without violation

downward casting (to descendants)

e.g. cast violation

Feature Call Arguments: Supplier

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

STUDENT

ST: ?

ST RS

parameter

ss[1], ss[2], ... ST: STUDENT.

Say:

sms: STUDENT_MANAGEMENT_SYSTEM

When should the following calls compile?

sms.add_s (o)

sms.add_rs (o)

sms.add_nrs (o)

argument

pass by value

parament := argument

s := o

supplier

add_s (s : STUDENT)

s := rs

client

rs : RS

sms.add_s (rs)

Feature Call Arguments: Client

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

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

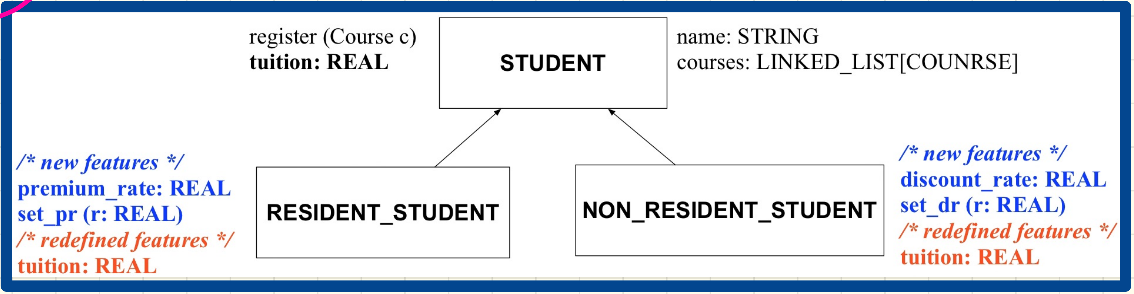
sms. add_s(s1)

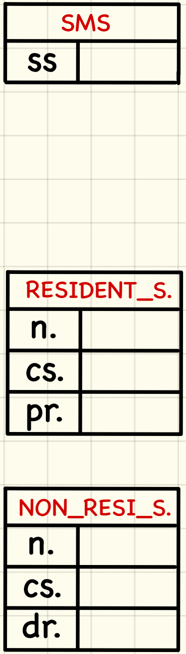
s := s1

sms.add_s (rs)

sms.add_rs (s1)

Polymorphic Collection

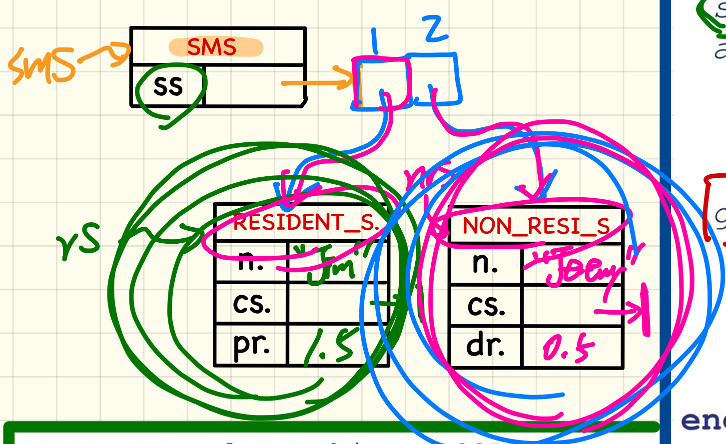





```
class STUDENT_MANAGEMENT_SYSETM
  students: LINKED_LIST[STUDENT]
  add_student(s: STUDENT)
    do
      students.extend (s)
    end
  registerAll (c: COURSE)
    do
      across
        students as s
      loop
        s.item.register (c)
      end
    end
end
```

```
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
```

Feature Call **Return Values**




```
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 of R

ST: STUDENT

```
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
```

ss[i]. pr

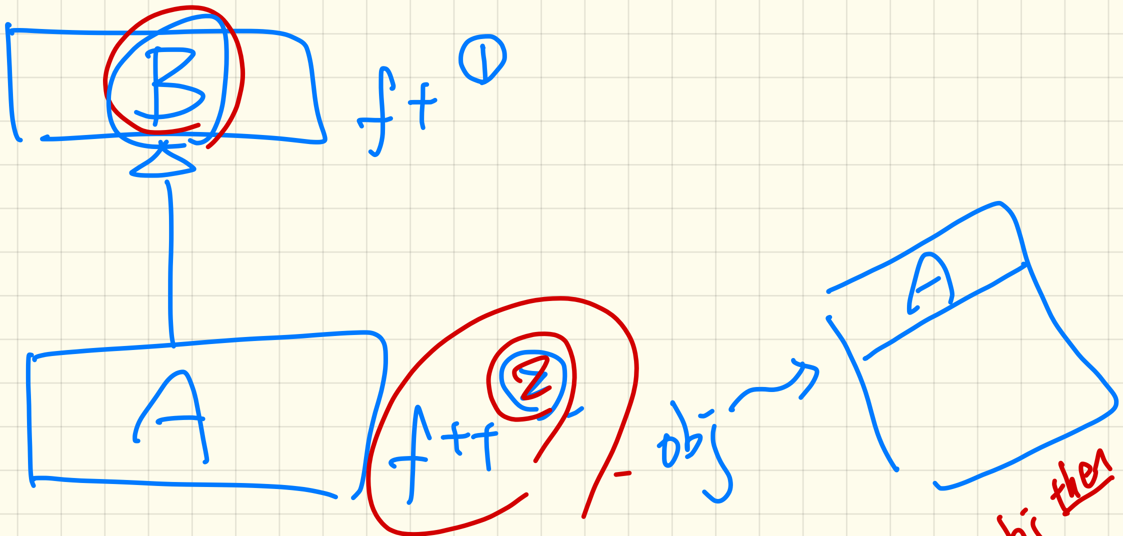
ST: STUDENT

get_student(2). dr X

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

descendant classes of ST

	ST	DT
sms[1]		RS
sms[2]	STUDENT	NRS



DB: version to be called depends on DT.

obj: A

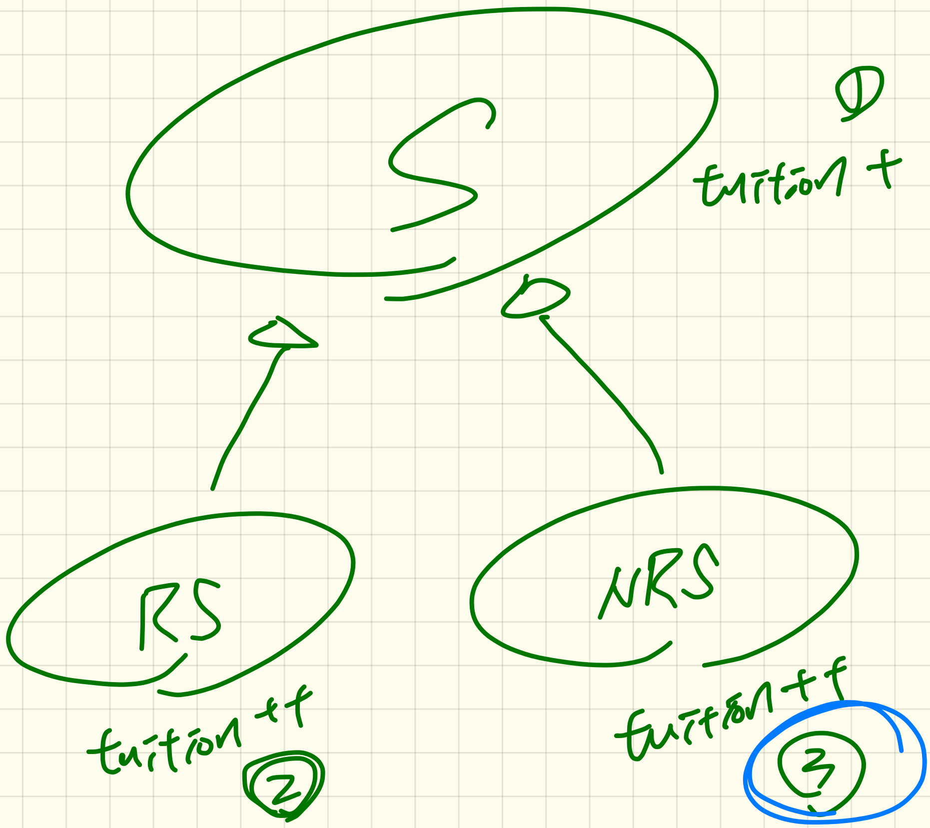
create {A} obj. make

check attached {B} obj as b_obj then

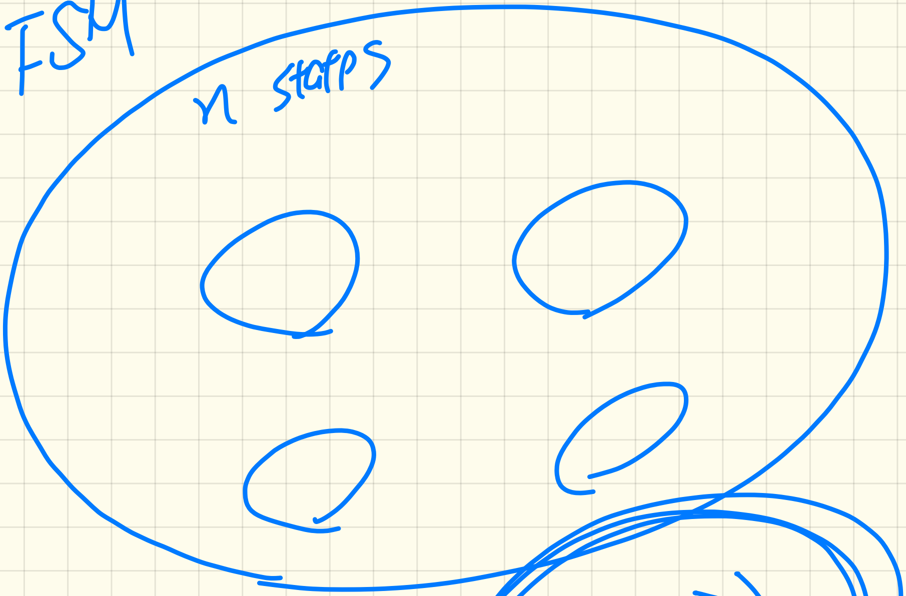
b_obj. f compiles

end

②


S
① tuition ↑
RS
tuition ↑↑
②
NRS
tuition ↑↑
③

FSM



transitions $O(n^2)$

Lecture 18
Wednesday March 11

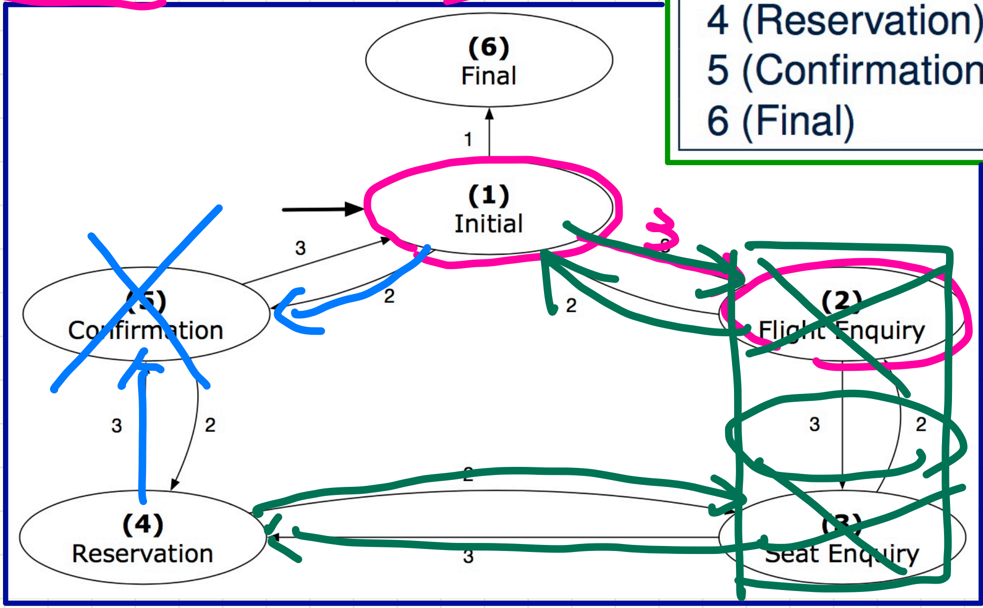
- Lab4 extended until 11am on Monday
- TA Hours: 9:30 to 11:30 on Thursday
- Office hours today shifted: 1pm to 3pm on Friday
    * Lab4
    * Labtest2
    * Exam

Finite State Machine (FSM)

ARRAY2.

model

State Transition Diagram



State Transition Table

Src State \ Choice	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

ARRAY[ARRAY2...]

Design of a Reservation System: First Attempt

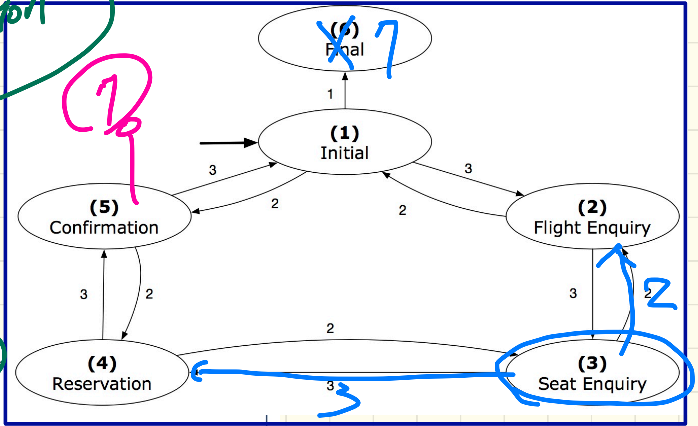
pattern of interaction

$\neg(w.a. \vee w.c.) \rightarrow$ exit.

$\equiv \boxed{\neg w.a} \wedge \boxed{\neg w.c} \rightarrow$ exit

correct a. c.c.

exit condition




```
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
```

loop

while (C) {

stay

}

Single Choice Principle.

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
```

transition(3, 3) ⇝ 4

transition : ARRAY2[INT]

Examples:

transition(3, 2)

transition(3, 3)

transition(3, 3) ↳ 4

State Transition Table

Src State \ Choice	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

2D Array Implementation

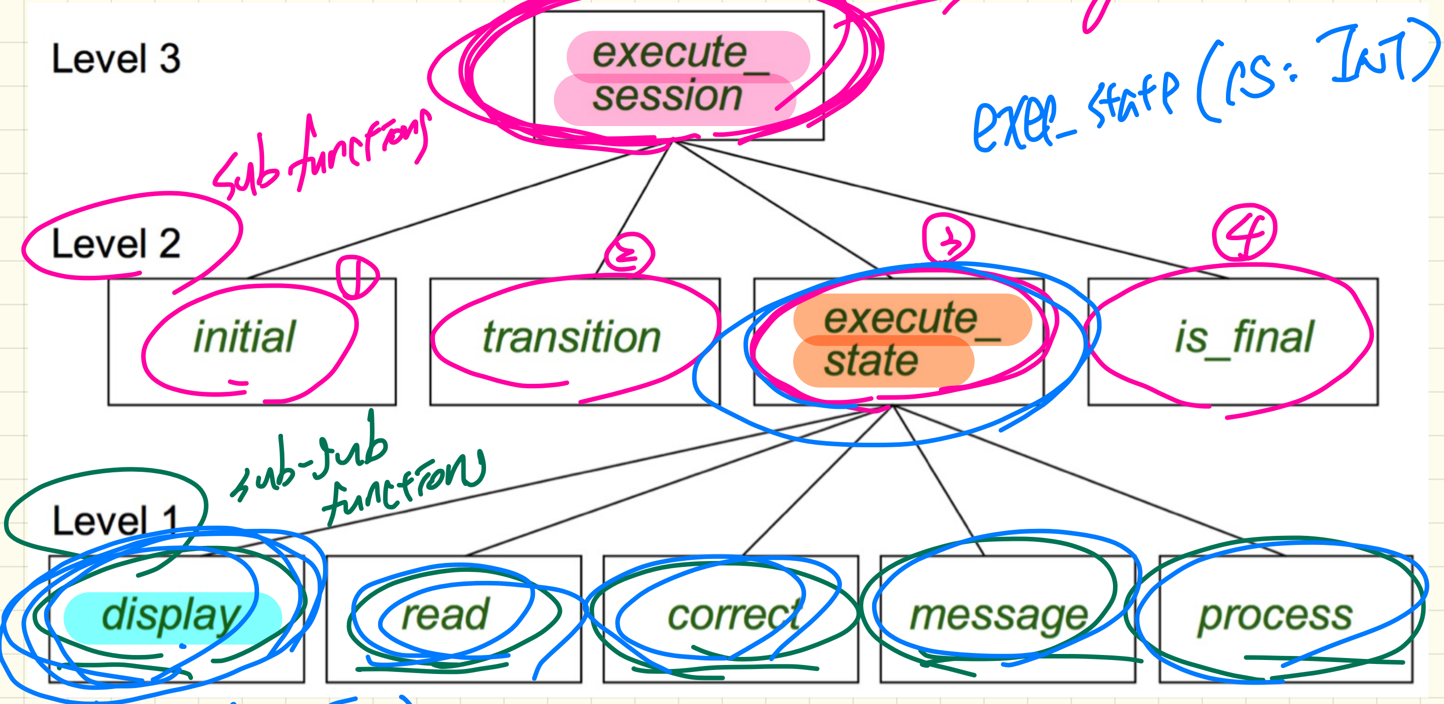
state \ choice	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

Design of a Reservation System: Second Attempt (2)

A Top-Down & Hierarchical Design

entry point of execution

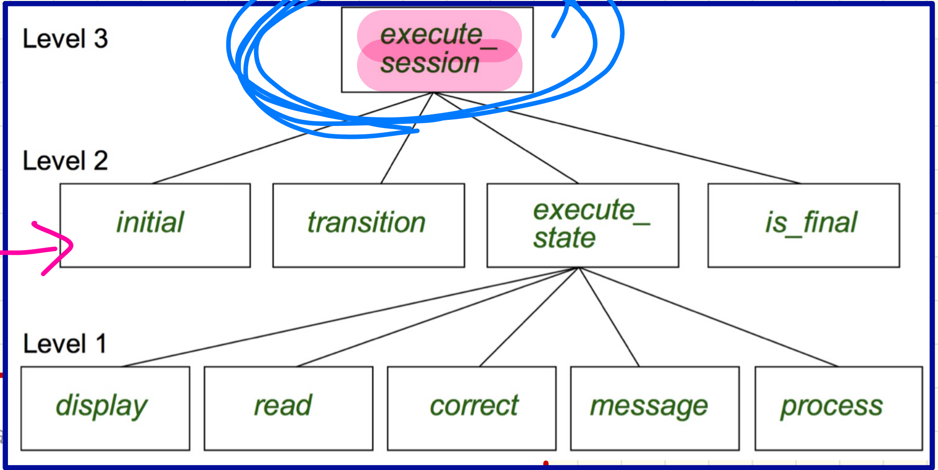
exec_state(cs: INT)



display(cs: INT)

Design of a Reservation System: Second Attempt (3)

act (e: ENT)




```
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
```

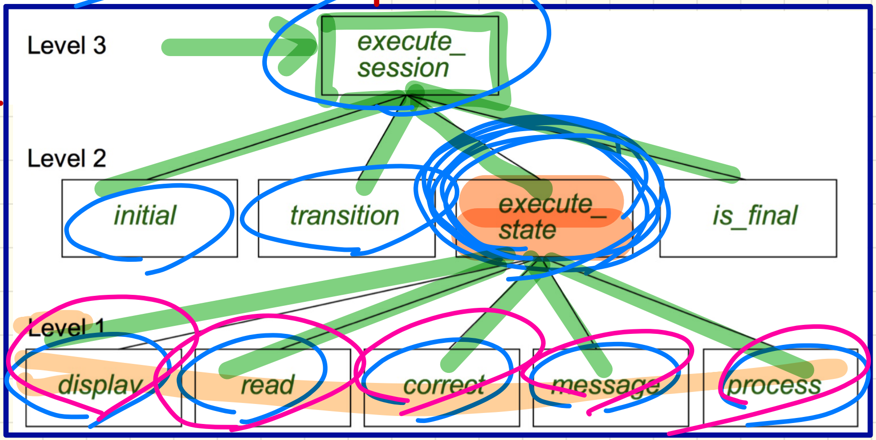
state-specific actions

Design of a Reservation System: Second Attempt (4)

```
execute_state ( current_state : INTEGER) : INTEGER
  -- Handle interaction at the current state.
  -- Return user's exit choice.
local
  answer: ANSWER; valid_answer: BOOLEAN; choice: INTEGER
do
  from
  until
    valid_answer
  do
    display( current_state )
    answer := read_answer( current_state )
    choice := read_choice( current_state )
    valid_answer := correct( current_state , answer)
    if not valid_answer then message( current_state , answer)
  end
  process( current_state , answer)
  Result := choice
end
```

pattern of interaction in each state (template)

e.s
t
in.
e.s.



add state 7.

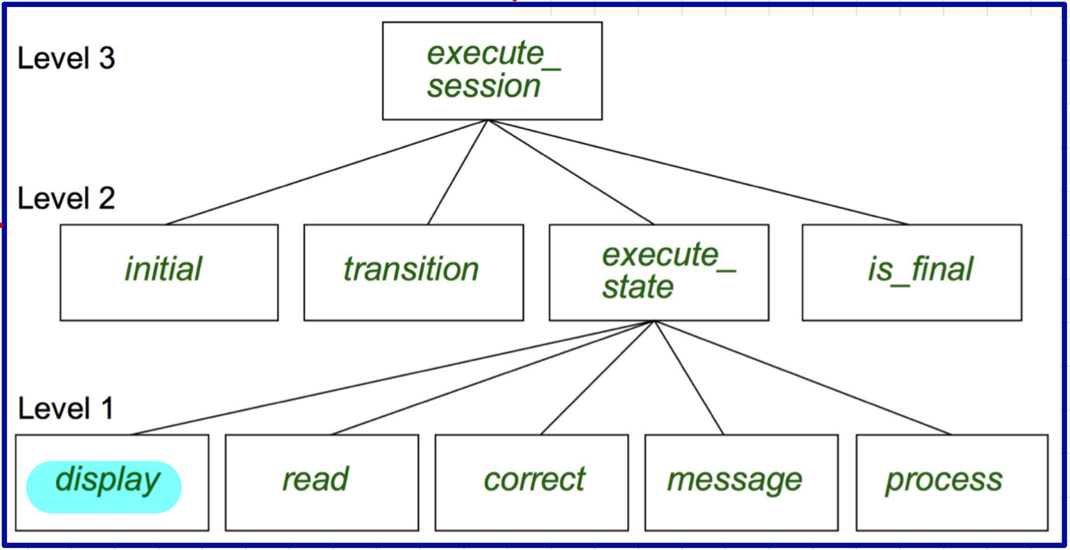
display(cs: INT)
do
if cs = 1 then
[
~~elseif cs = 2 then~~
~~[~~
elseif cs = 6 then
[
elseif cs = 7 then
[
end
end

delete state 2.

read(cs: INT)
do
if cs = 1 then
[
~~elseif cs = 2 then~~
~~[~~
elseif cs = 6 then
[
elseif cs = 7 then
[
end
end

Design of a Reservation System: Second Attempt (5)

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



2nd Design Attempt

```
class
   STUDENT
create
   make
feature -- attribures
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
   premiumRate: REAL
   discountRate: REAL
feature -- command
   make (kind: INTEGER)
     do
       kind := a_kind
     end
   ...
end
```

```
get_tuition: REAL
   local
      tuition: REAL
   do
      across courses is c loop
         tuition := tuition + c.fee
      end
      if kind = 1 then
         Result := tuition * premiumRate
      elseif kind = 2 then
         Result := tuition * discountRate
      end
   end
```

```
register (c: COURSE)
   local
      max: INTEGER
   do
      if kind = 1 then MAX := 6
      elseif kind = 2 then MAX := 4
      end
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

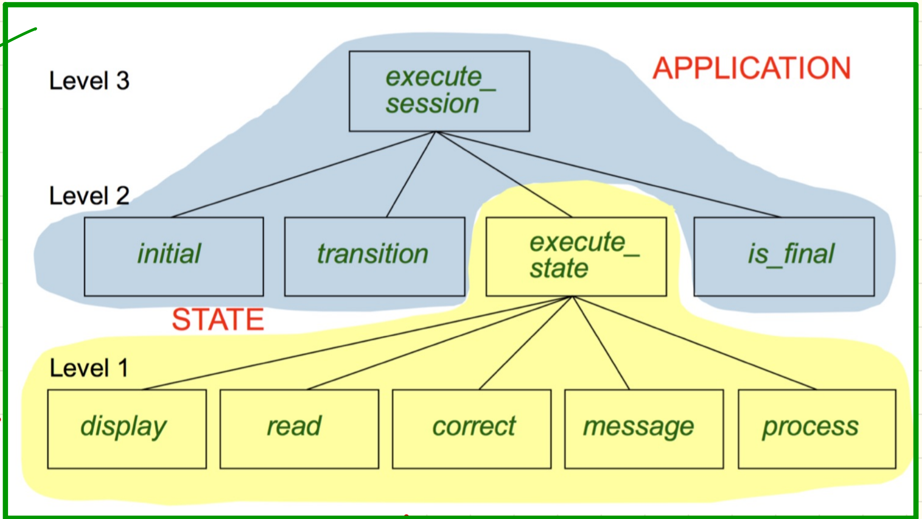
Moving from Top-Down Design to OO Design

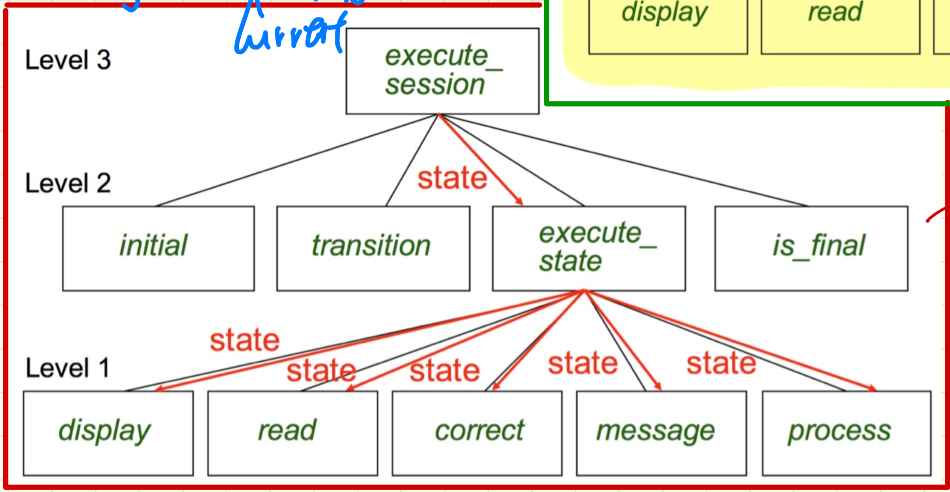
Object-Oriented

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

context object

context object of this current



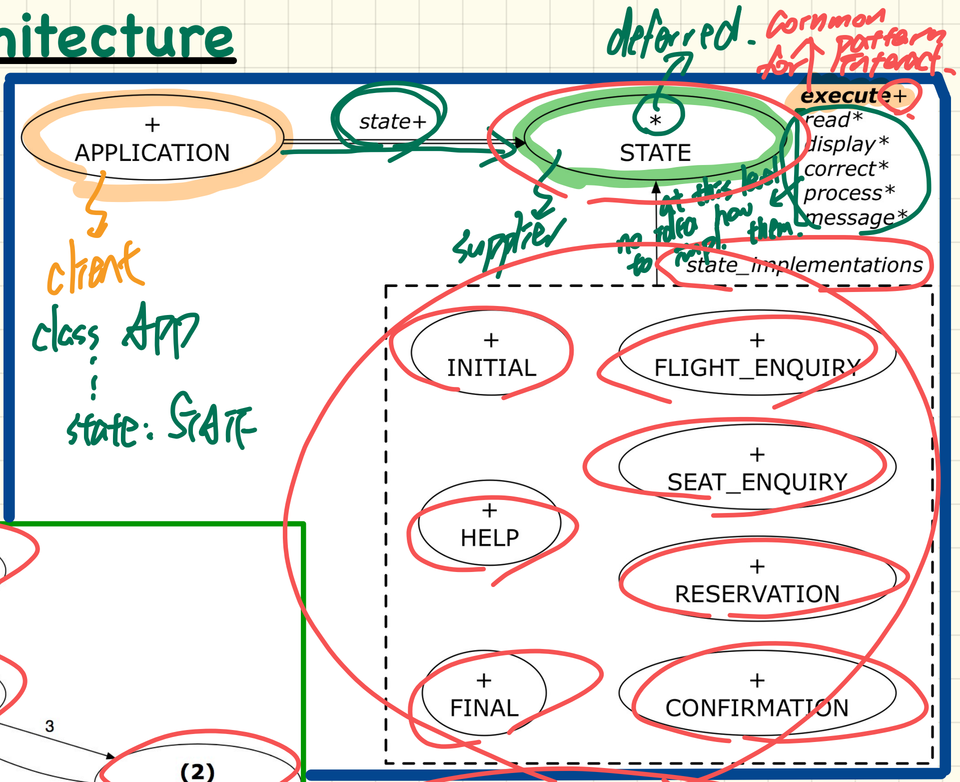


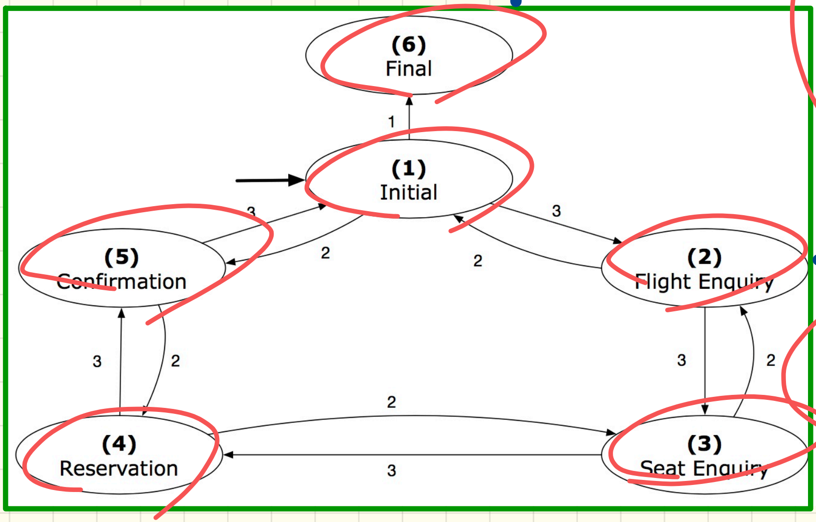
Top-Down

current_state: **INTEGER**
execute_session(current_stste)

input argument

State Pattern: Architecture






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

State Pattern: State Module

STATE*
display*
read*
execute+

S_E+
display+
read+

CON+
display+
read+

```
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
```

implemented

version: S_E

version: CON

version: S_E

version: CON

unimp.

TEMPLATE

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

DT: S_E

version: STATE

DT: CON.

DT: CON.

STATE
s

S_E

CON.

version: STATE

```
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
```

template

```
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
```

implemented.

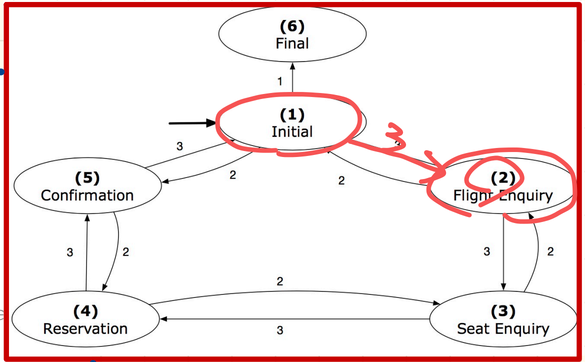
deferred.

display+

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
```

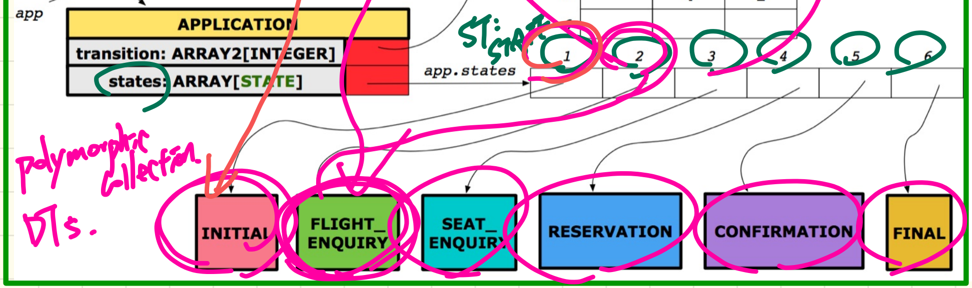
Init



transition (1, 3) → 2

states : ARRAY[STATE]

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



polymorphic collection

DTs.

CS : STATE

DT: RES.

DT: F_E

CS := states[2]

CS.display (1) → F_E

CS := states[4]

CS.display (2) → RES

LECTURE 19
MONDAY MARCH 16

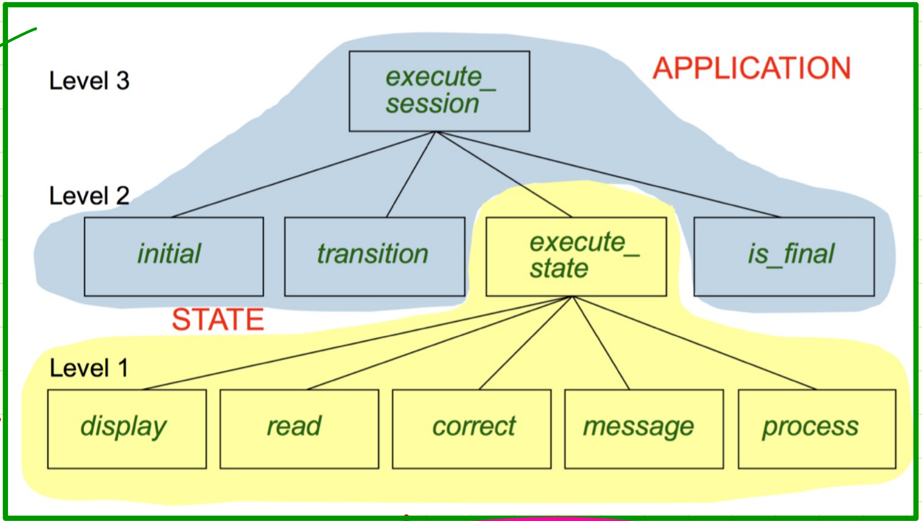
Moving from Top-Down Design to OO Design

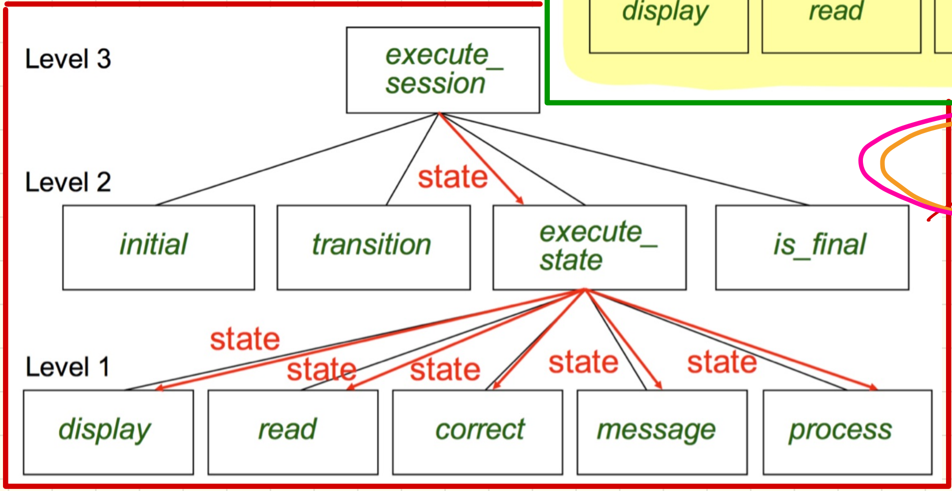
Object-Oriented

current_state: **STATE** → tp.

current_state.*execute_session*

DT.





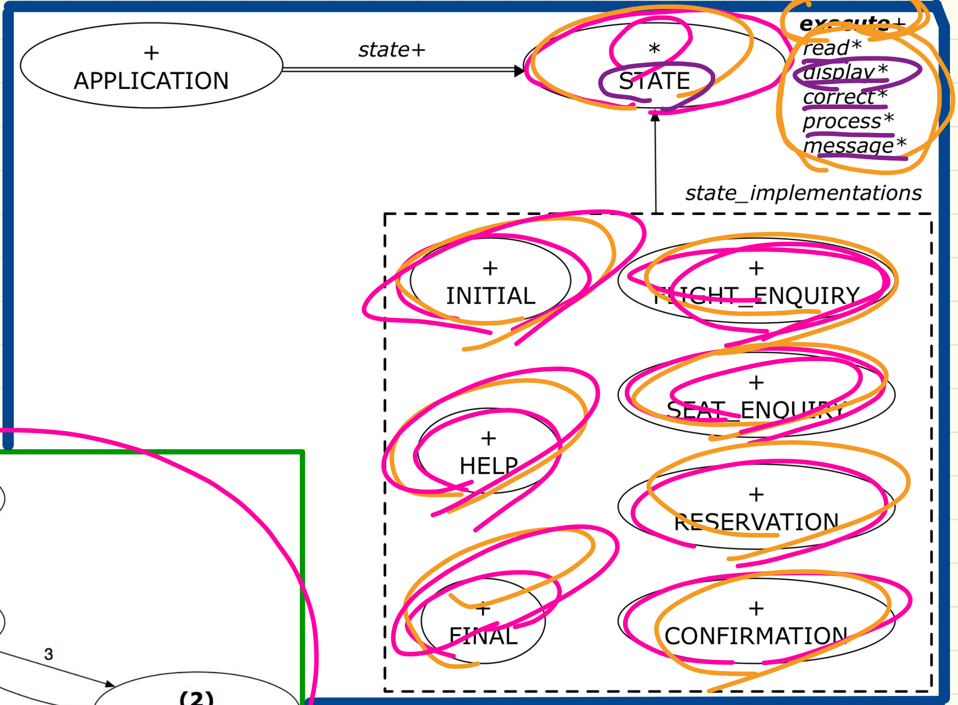
Top-Down

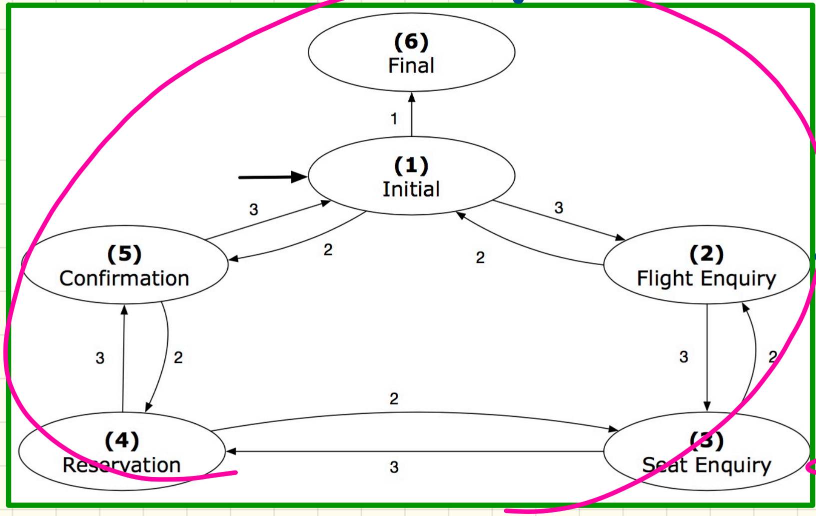
current_state: **INTEGER**

execute_session(current_stste)

State Pattern: Architecture

template






```
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
```

template

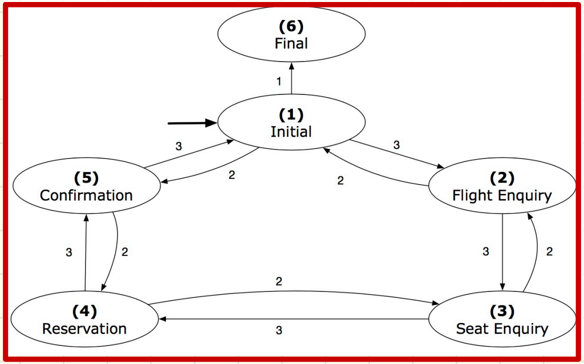
```
execute
  local
    good: BOOLEAN
  do
    from
    until
      good
    loop
      display
      -- set answer and choice
      read
      good := correct
      if not good then
        message
      end
    end
    process
  end
end
```

TEMPLATE

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



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
```

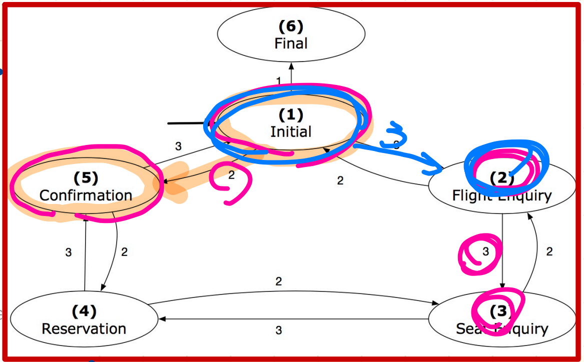
DT

cs.display() V. INITIAL

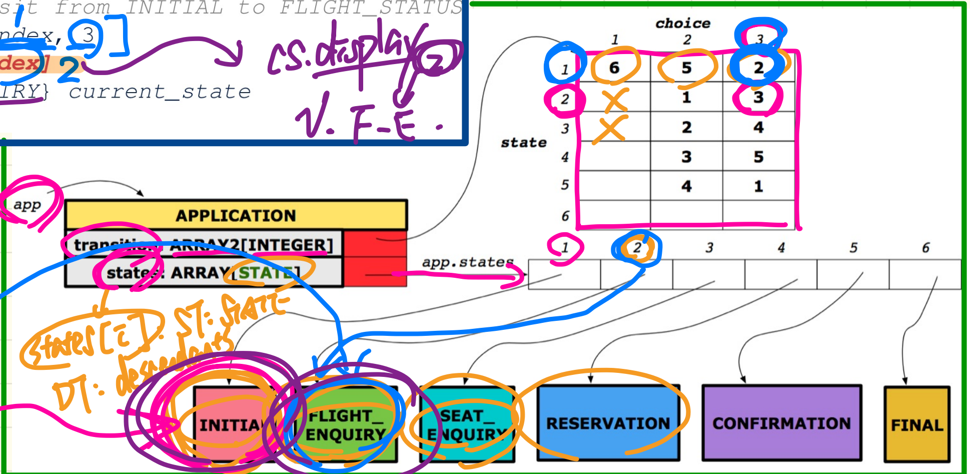
DT

instanceof

cs.display() V. F-E.



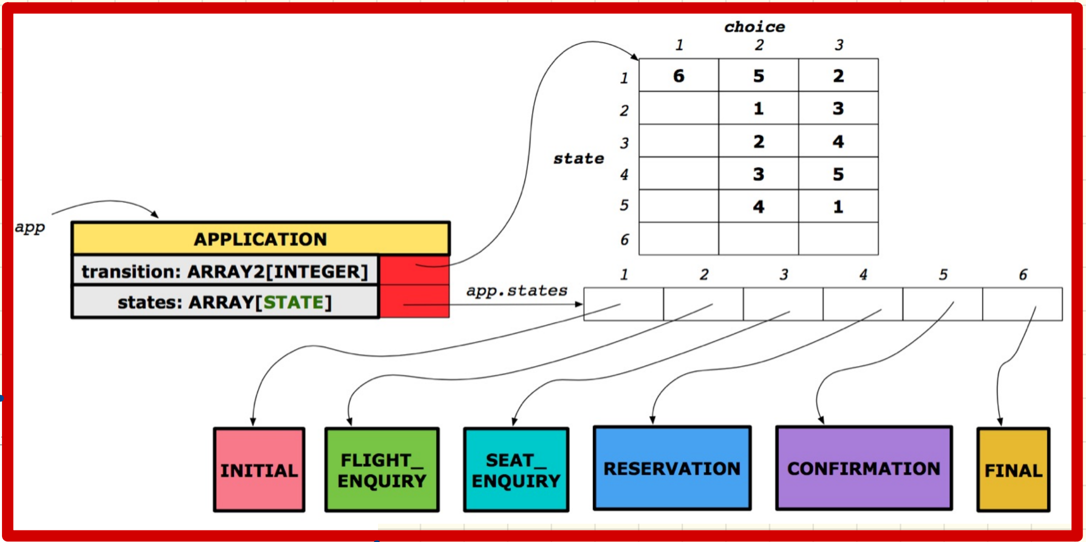
put_transition(5, 1, 2)



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

STATE current_state

State Pattern: Interactive Session




```
class APPLICATION
feature {NONE} -- Implementat
  transition: ARRAY2[INTEGER]
  states: ARRAY[STATE]
feature
  execute_session
    local
      current_state: STATE
      index: INTEGER
    do
      from
        index := initial
      until
        is_final (index)
      loop
        current_state := states[index] -- polymorphism
        current_state.execute -- dynamic binding
        index := transition.item (index, current_state.choice)
      end
    end
end
```

ST

ST

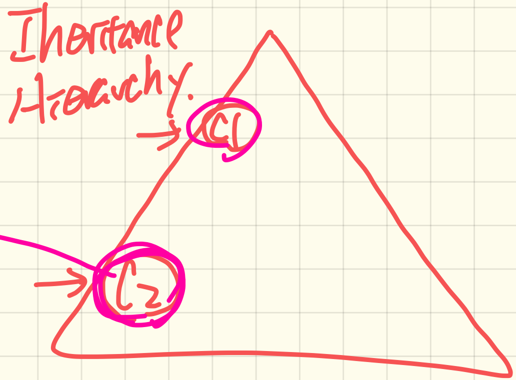
ST: STATE

compile?

ST: STATE

D.B.: depending on the DT of the curr. C.S.'s version of execute will be called

Polymorphism.



expectation:
as much as
what c1 can
support

v1 := v2

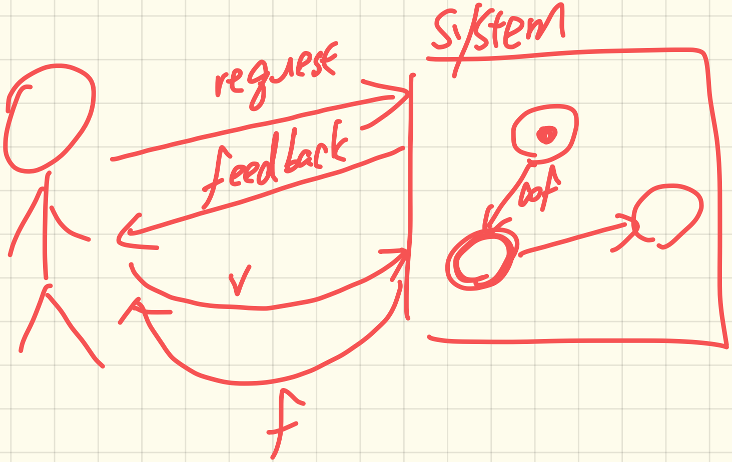
compile?

ST: ?
C1

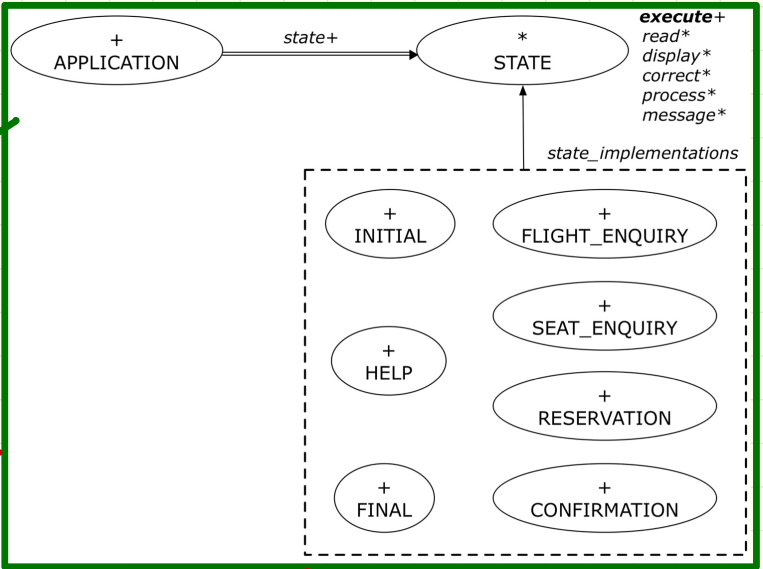
ST: ?
C2

a descendant of

ETF $-\bar{\iota}$

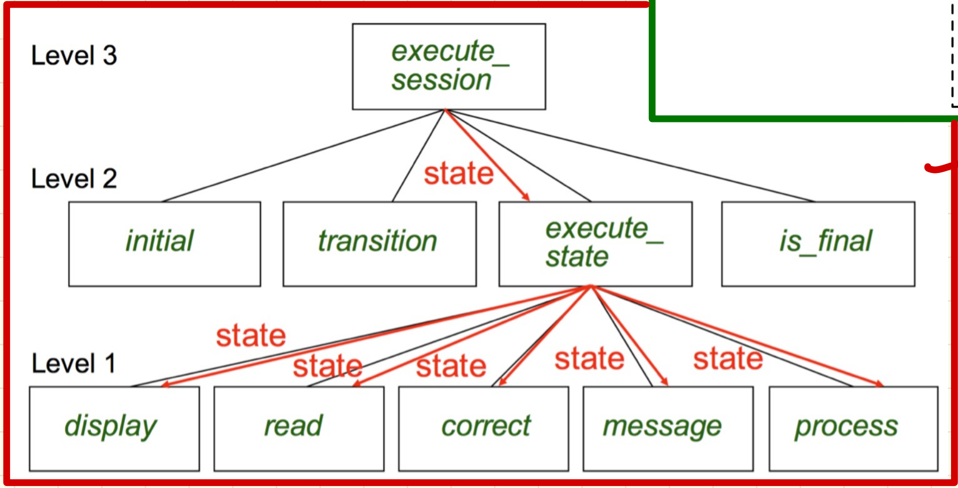


Interactive System: Top-Down Design vs. OO Design



Object-Oriented

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



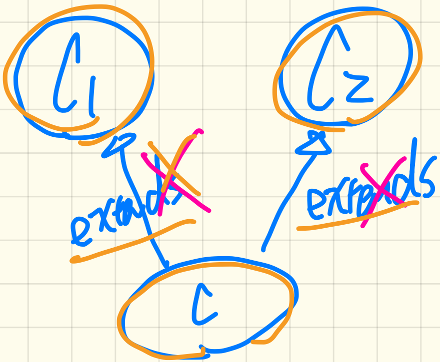
Top-Down

current_state: **INTEGER**
execute_session(current_stste)

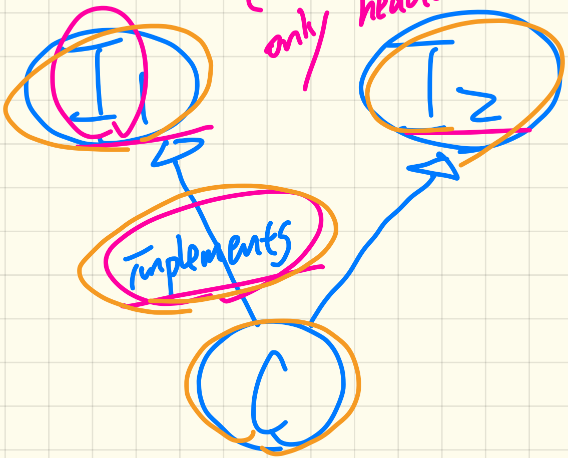
Java

M.I. limited to interface.

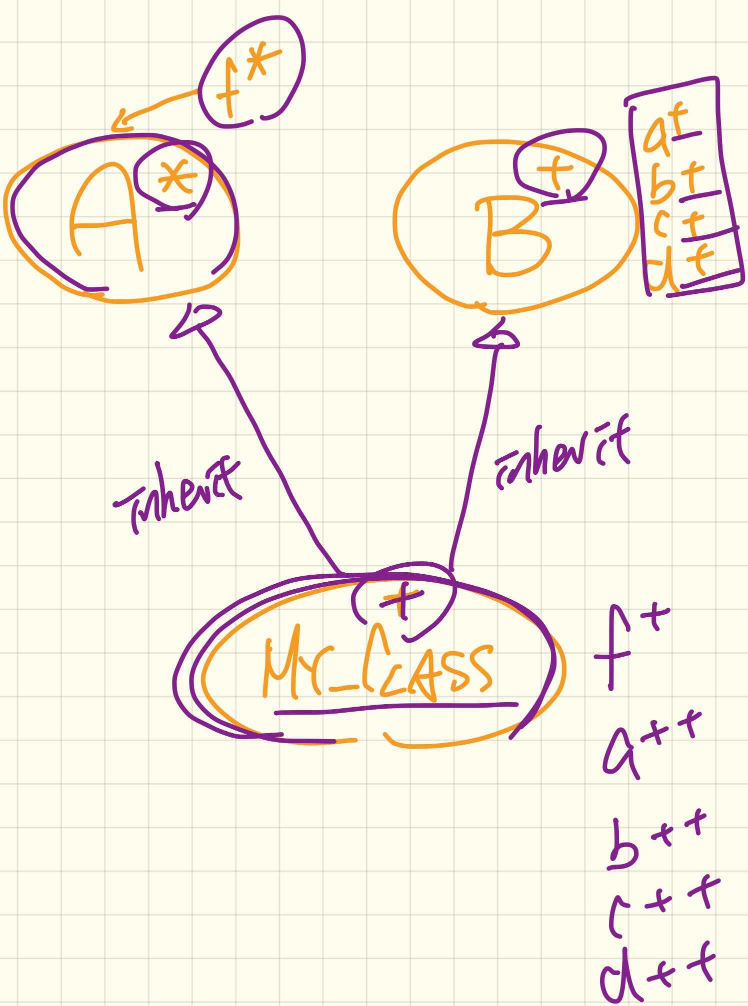
class C extends C1, ~~C2~~



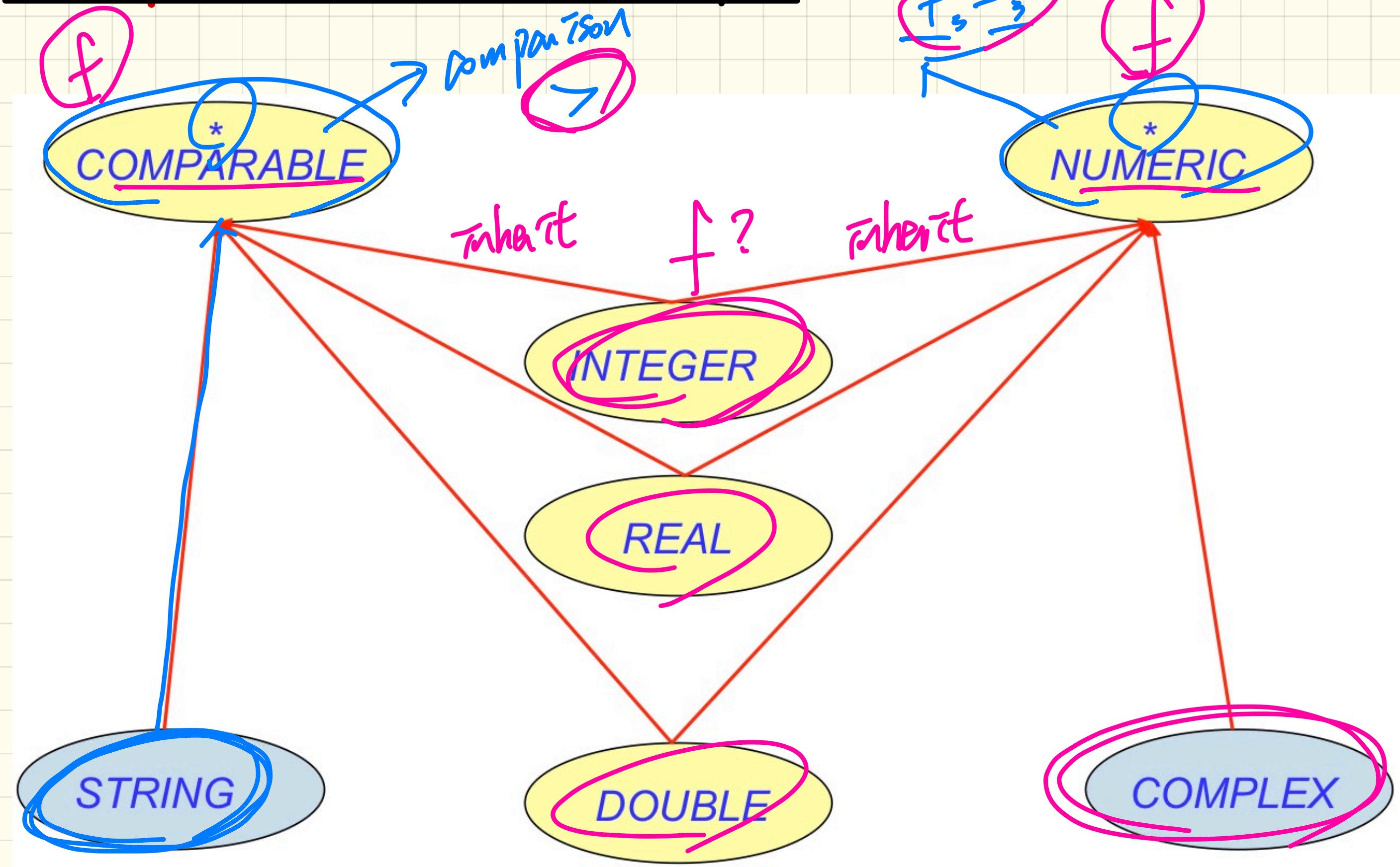
Interfaces (no implementation only headers are inherited)



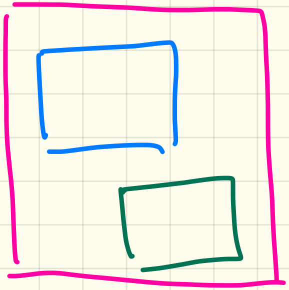
Eiffel

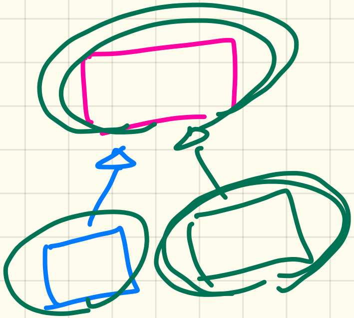


Multiple Inheritance: Example



Design I





superman class

Multiple Inheritance

single choice principle

class Window

feature

xpos, ypos

width, height

feature

move, add_child

parents, children

end

Cohesion violated ✓

∴ parents and children are included in the same class.

dimension of win.

tree of wid.

W

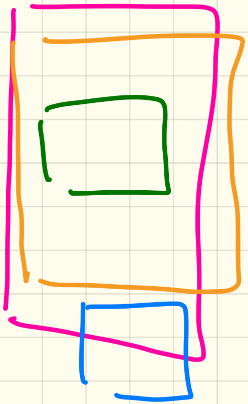
Multiple Inheritance: Exercise

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

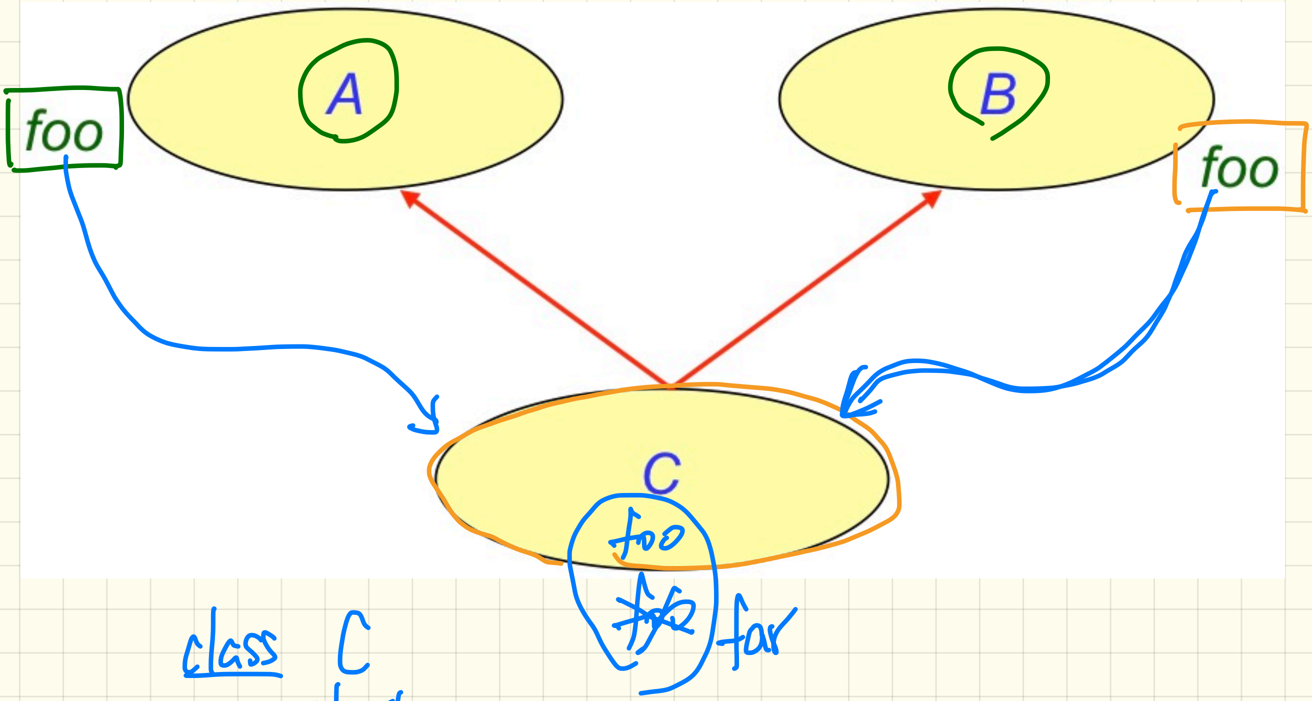
```
class TREE[G]
  feature -- Queries
    descendants: ITERABLE[G]
  feature -- Commands
    add (c: G)
        -- Add a child 'c'.
end
```

```
class WINDOW
  inherit
    RECTANGLE
    TREE[WINDOW]
end
```

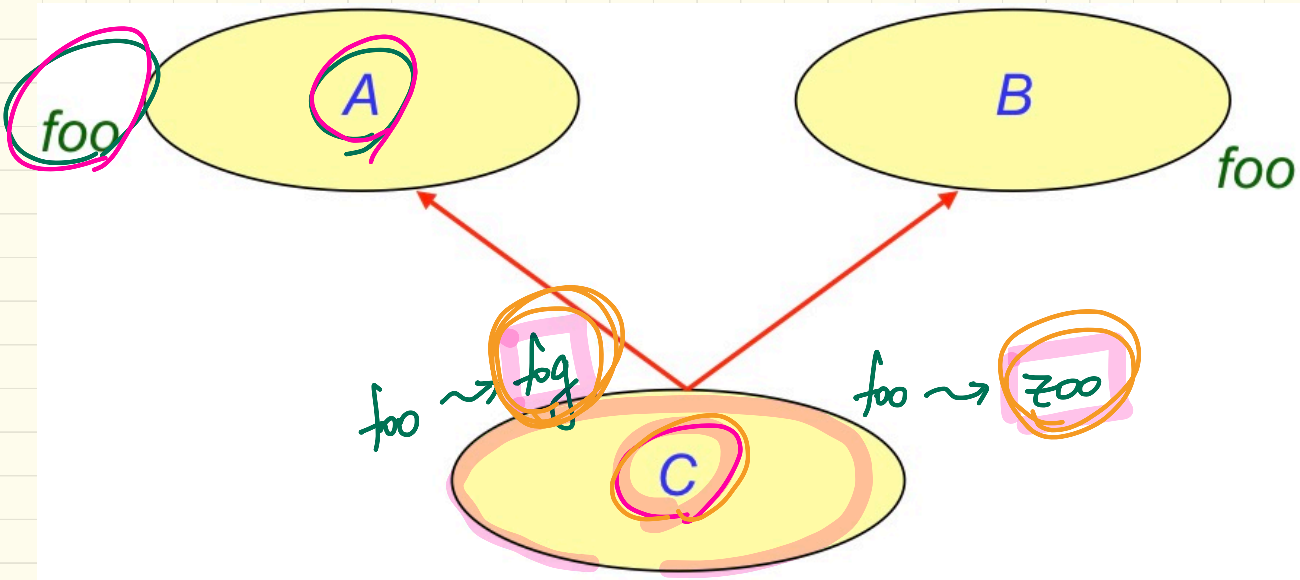
```
test_window: BOOLEAN
  local w1, w2, w3, w4: WINDOW
  do
    create w1.make(8, 6) ; create w2.make(4, 3)
    create w3.make(1, 1) ; create w4.make(1, 1)
    w2.add(w4) ; w1.add(w2) ; w1.add(w3)
    Result := w1.descendants.count = 2
  end
```



Multiple Inheritance: Name Clashes



class C
inherit
A
B



obj1 : A

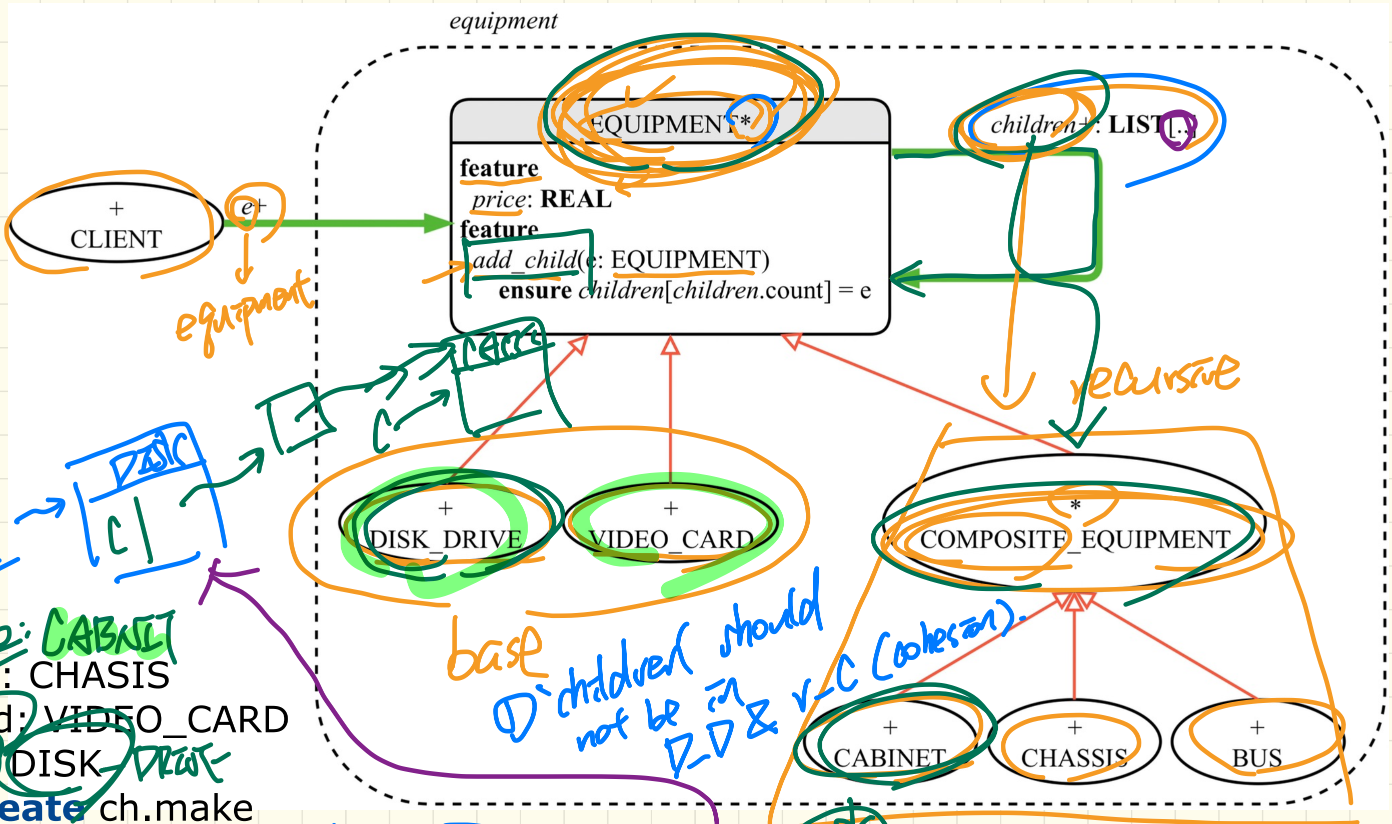
① obj1.foo ✓
↳ ST: A

② obj1.fog ✗
↳ ST: A

obj2 : C

③ obj2.foo ✗
↳ ST: C

④ obj2.fog ✓



cab: CABINET
ch: CHASIS
crd: VIDEO_CARD
d: DISK_DRIVE
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)

First Design Attempt

d.add_child(cab).

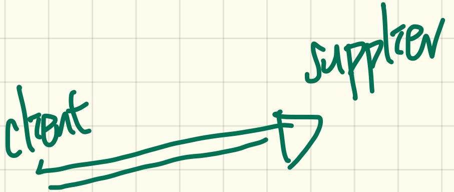
```
deferred class EQUIPMENT
    feature
        children: LIST[EQUIPMENT]
end
```

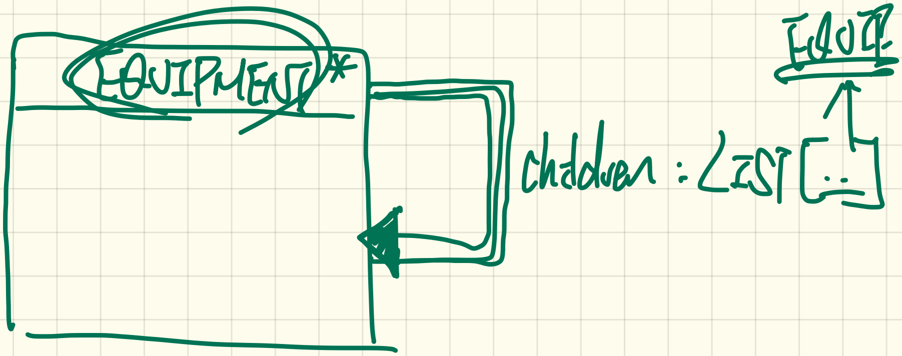
client

supplier name

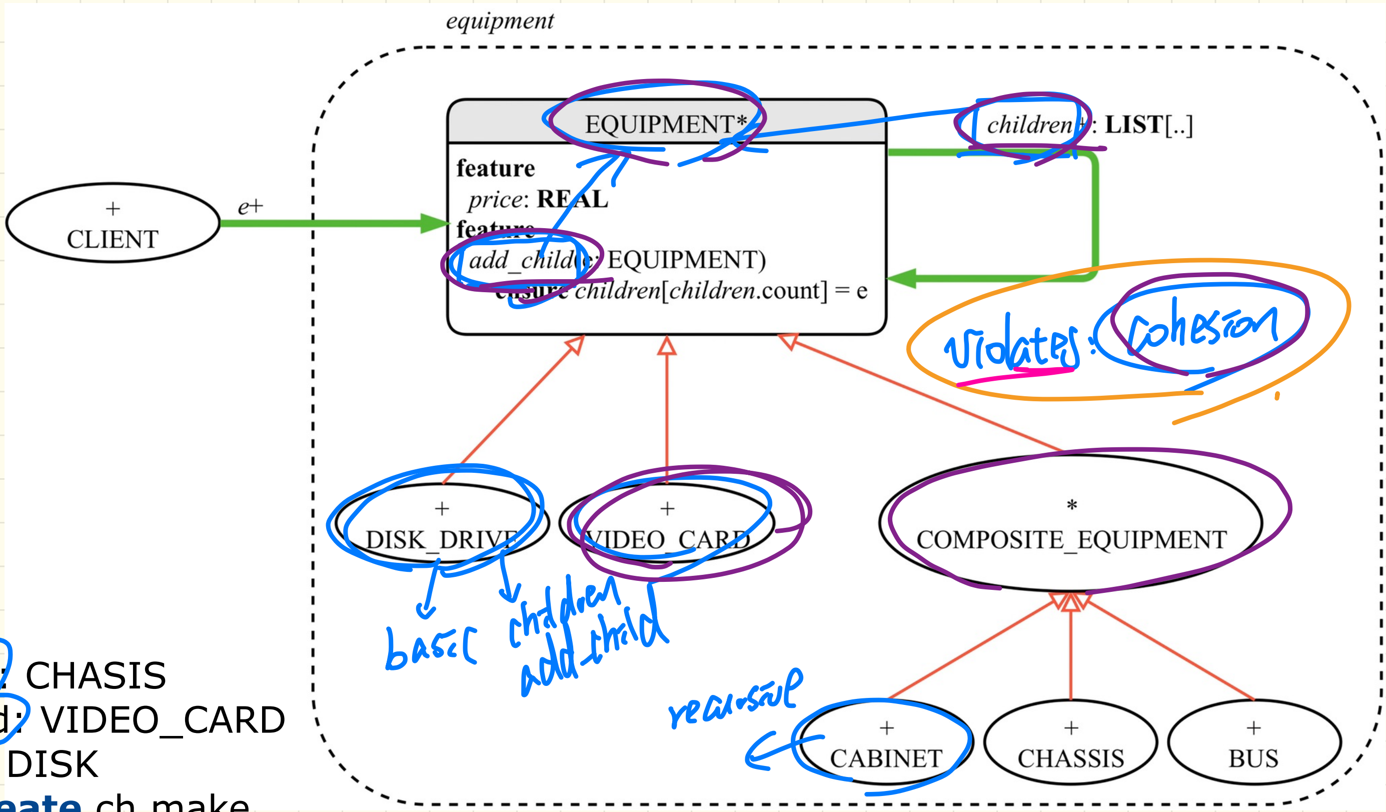
supplier type

```
class Node {
    Node next;
}
```





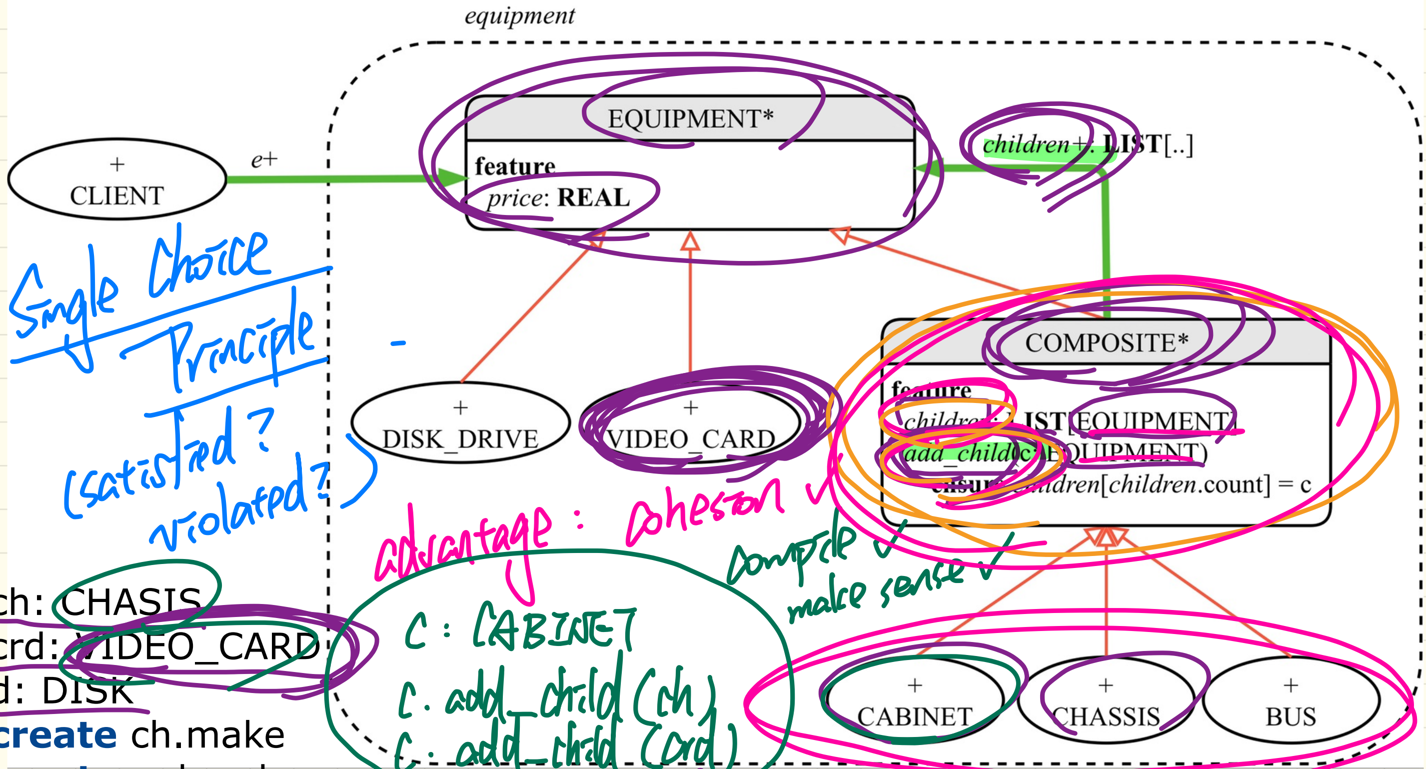
Lecture 20
Wednesday March 18



First Design Attempt

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

crd.add_child(d) ✓



Single Choice Principle (satisfied? violated?)

advantage: cohesion ✓
compile ✓
make sense ✓

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

✗ not compiling ∵ add_child is a feature applicable to descendants of COMPOSITE

c: CABINET
c.add_child(ch)
c.add_child(crd)

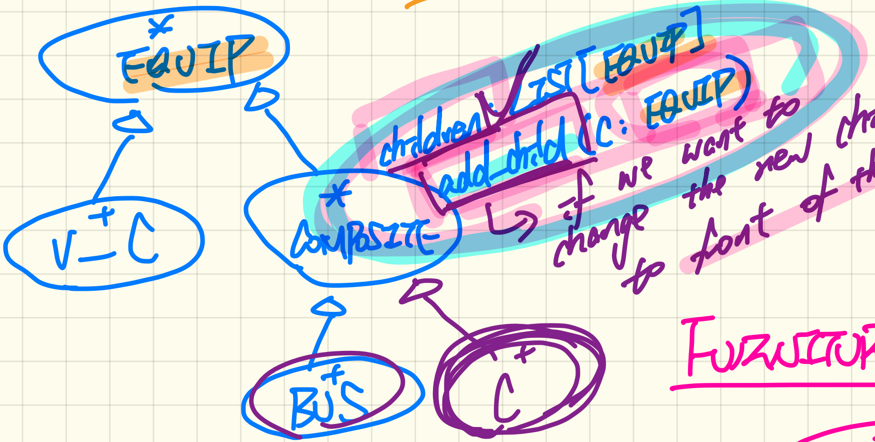
c.o. ✓ ✓

Second Design Attempt

EQUIPMENT

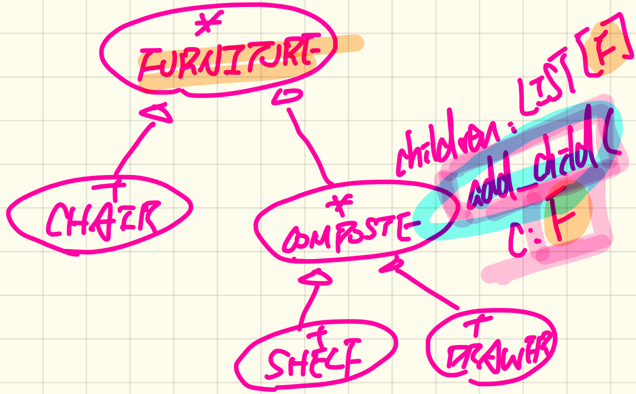
Single Choice Principle

violation ⇒ changes take multiple places to undo.



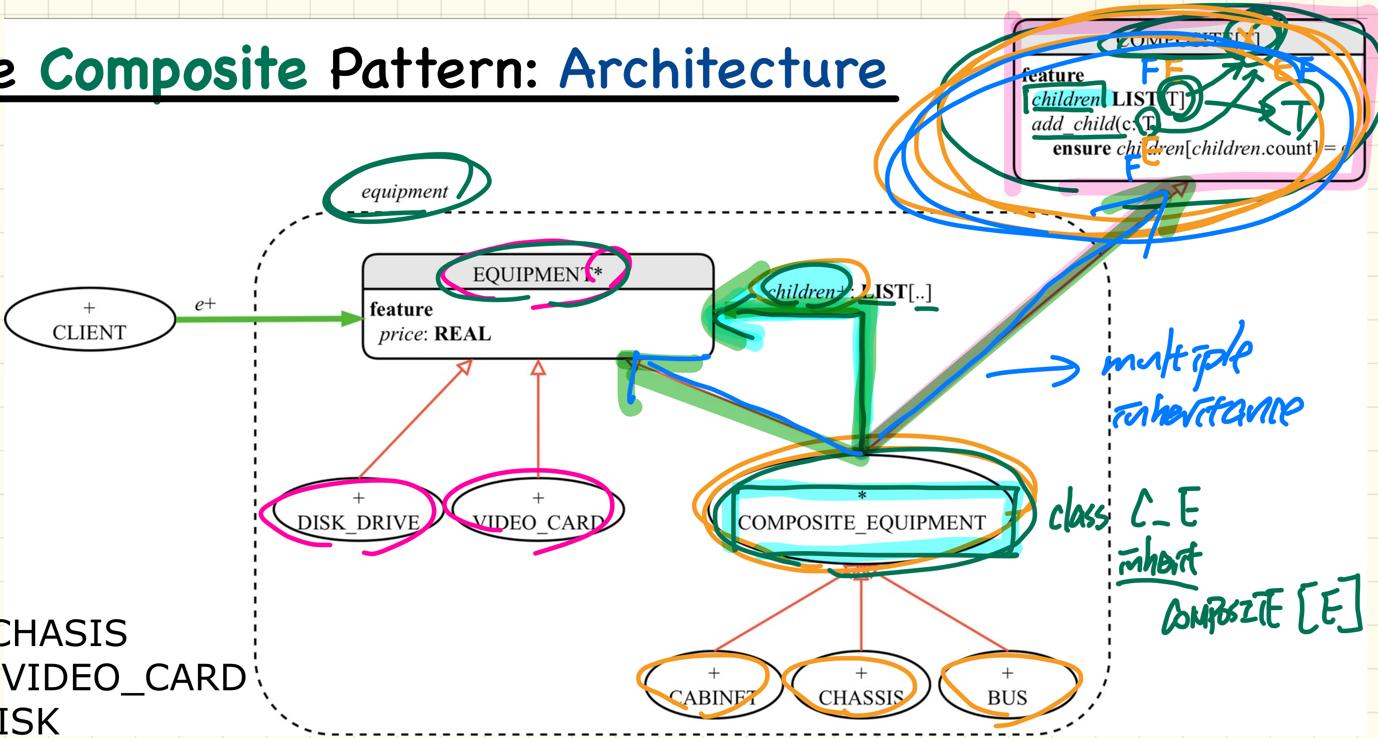
↳ if we want to change the new child to front of the list.

FURNITURE



change: change add_child s.t. insert it to the end of the list.

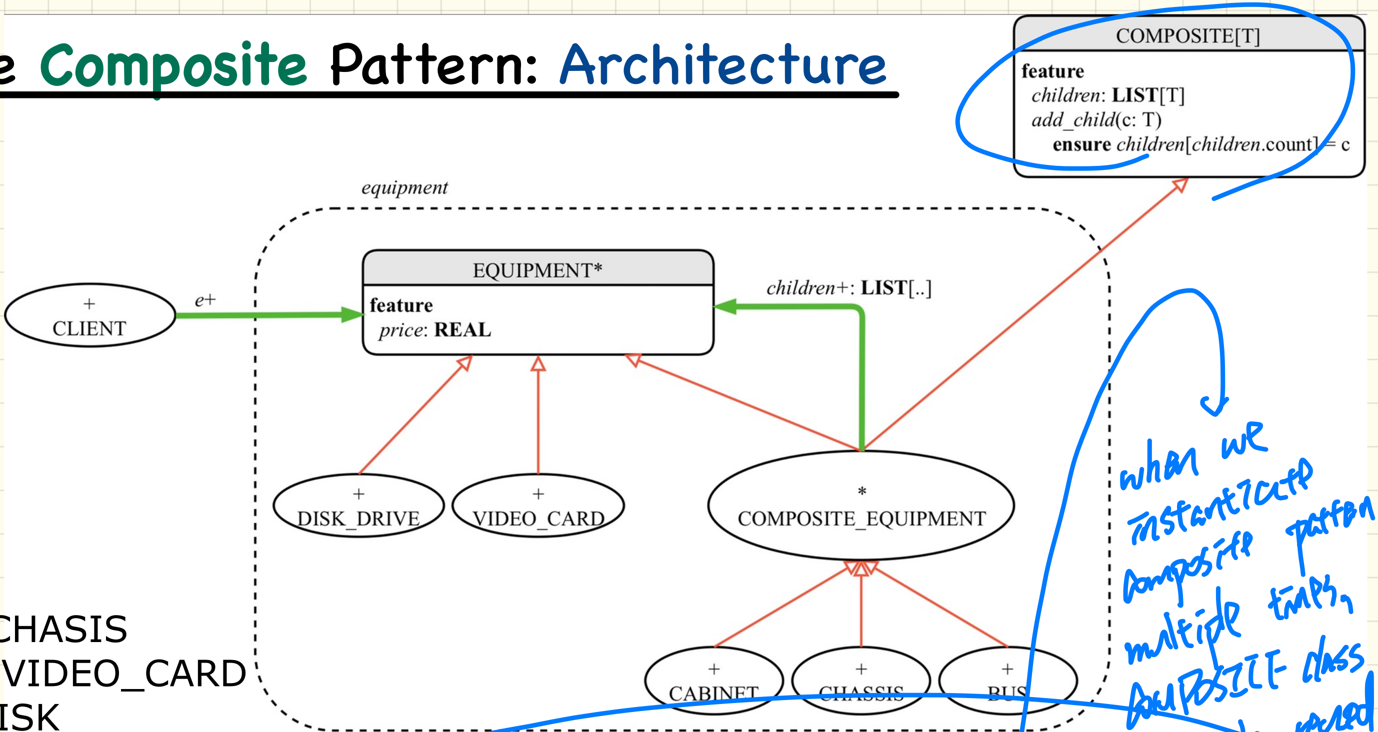
The Composite Pattern: Architecture



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

exercise: should this compile?

The Composite Pattern: Architecture



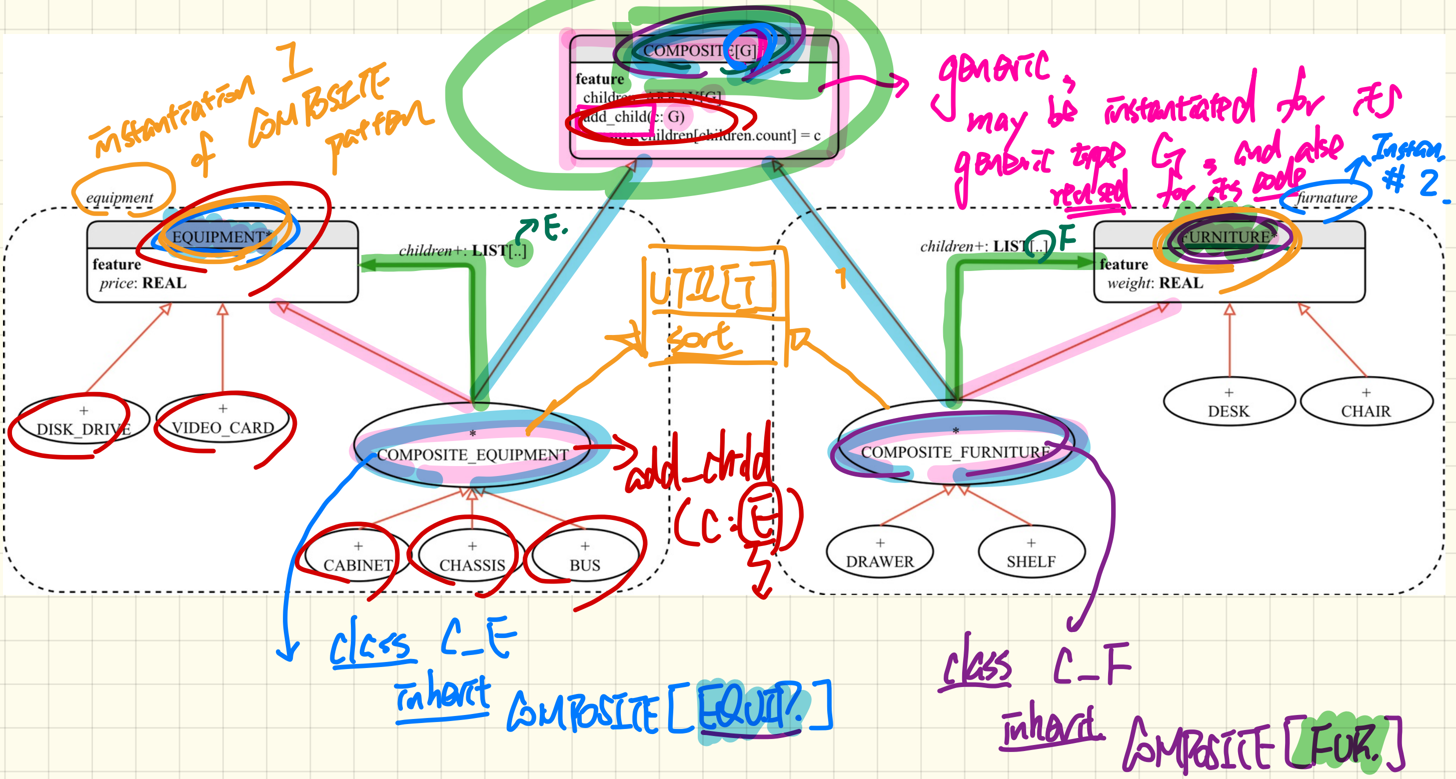
when we instantiate composite pattern multiple times, COMPOSITE class can be reused

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

Why is COMPOSITE a separate class?

The Composite Pattern: Architecture

COMPOSITE class is reusable by instances of the composite pattern.



template design pattern

A^*

f^+ → effective/implemented.

$g1^*$

$g2^*$

f
do
$g1$
$g2$
end

pattern for composing the exe. of g1 & g2

obj: A

create {B} obj.make

obj.f

create {C} obj.make

obj.f

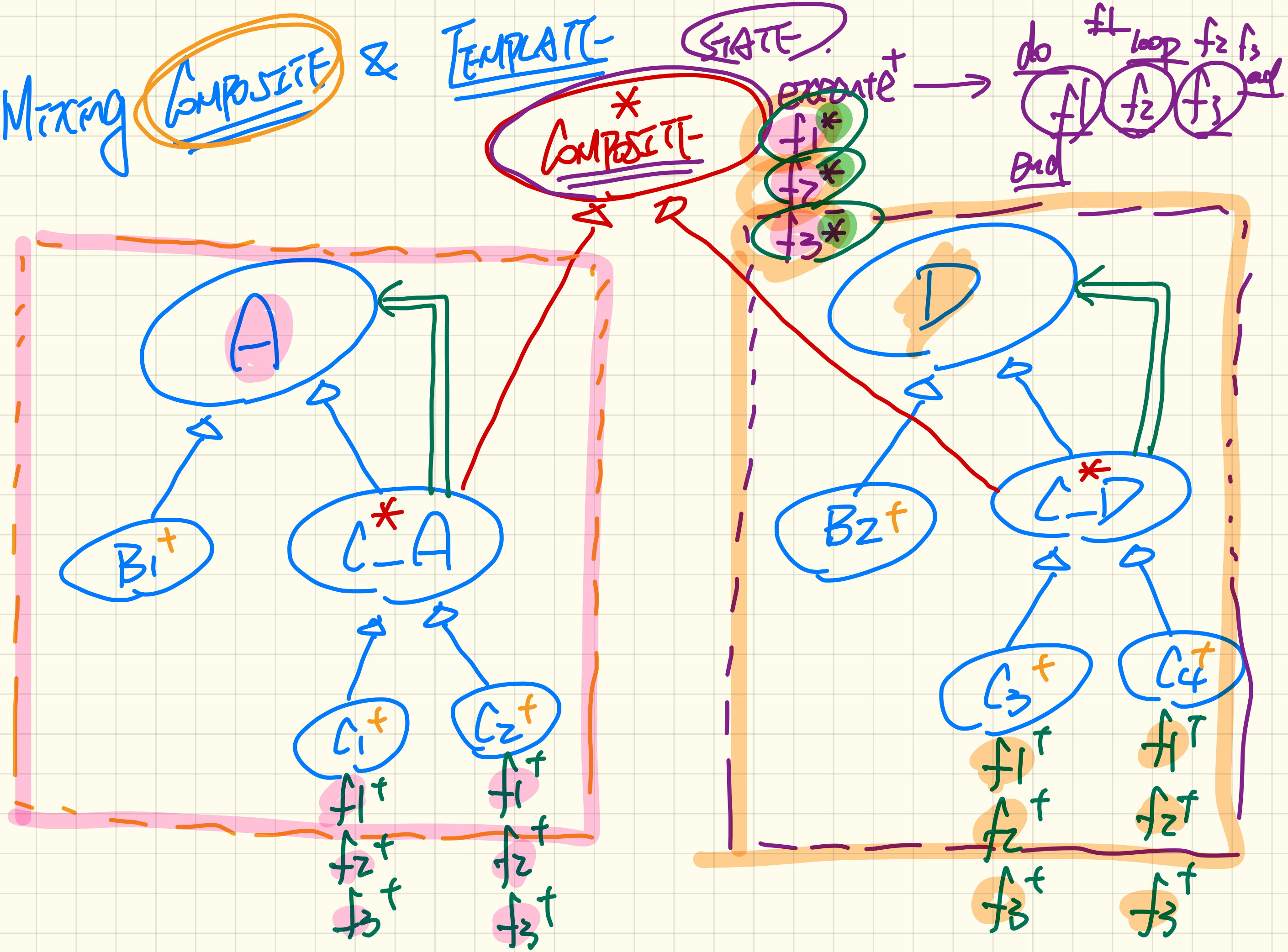
B^+

version B
$g1^+$
$g2^+$

C^+

$g1^+$
$g2^+$
version C

Mixing COMPOSITE & TEMPLATE STATE



The Composite Pattern: Implementation

```
deferred class
  EQUIPMENT
feature
  name: STRING
  price: REAL -- uniform access principle
end
```

```
deferred class
  COMPOSITE[T]
feature
  children: LINKED_LIST[T]

  add_child (c: T)
    do
      children.extend (c) -- Polymorphism
    end
end
```

EQUIP.

EQUIP

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

v1

200

basic components

```
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
```

v2

ST? EQUIP.

Composite Component

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

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

v1

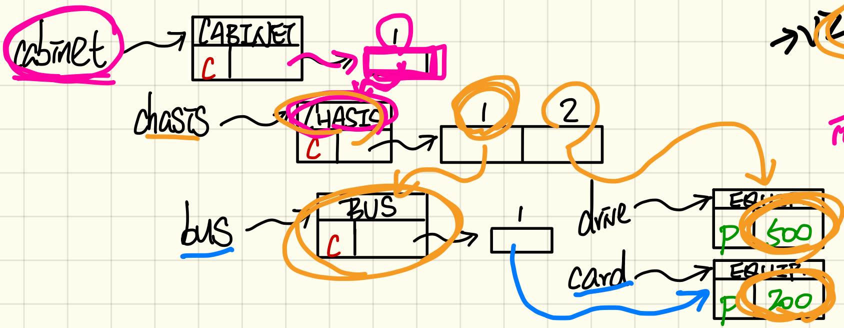
```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
    -- Sum the net prices of all sub-equip
    do
      across
        children as cursor
      loop
   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

v2

DT? BUS

DT? CHASIS

{CHASSIS}.price



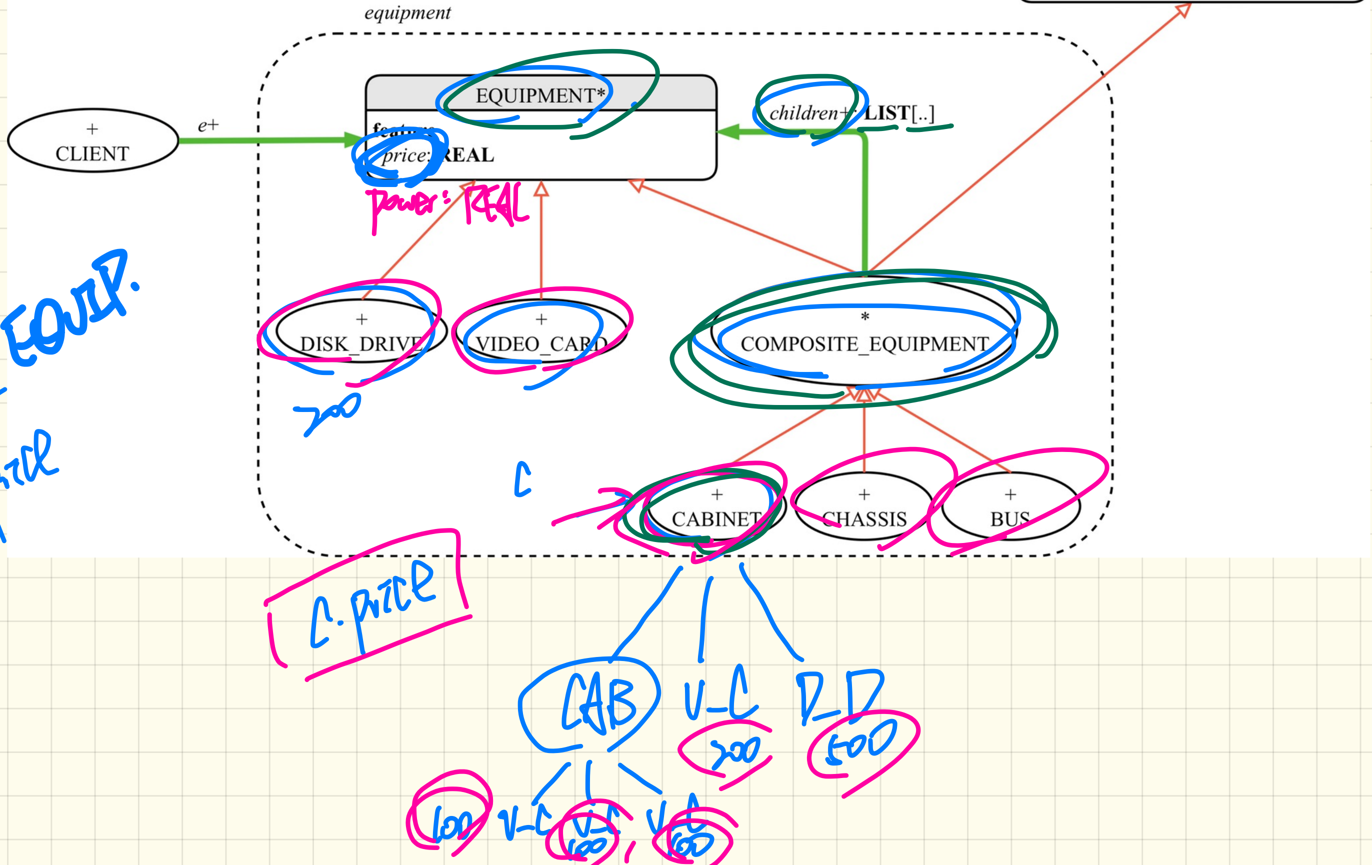
Lecture 21

Monday March 23

The Composite Pattern: Architecture

e: EQUIP.

e.price

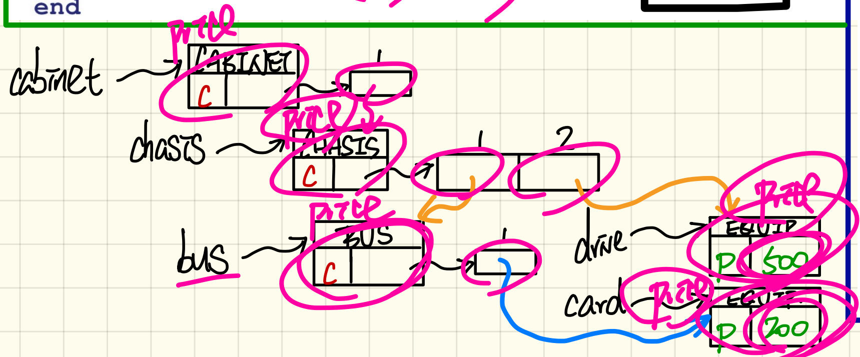


Testing the Composite Pattern

```
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
```

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

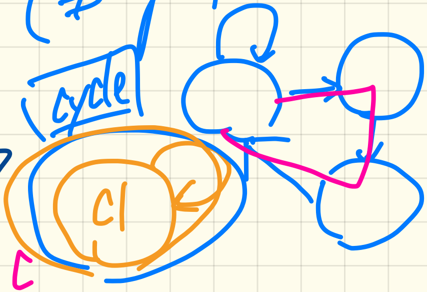
```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
      -- Sum the net prices of all sub-equip
    do
      across
        children as cursor
      loop
 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



Composite Design Pattern

Runtime: Tree (≠ graph)

cycle



what if self-loops?

① cycle good? bad?

② if bad then how to prevent

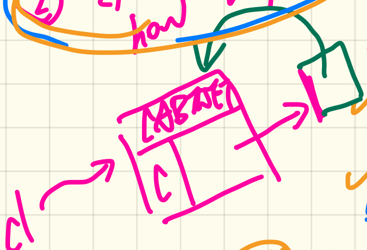
across all children ≠ c

current ≠ c

ref

and

c1, c2 : CABINET

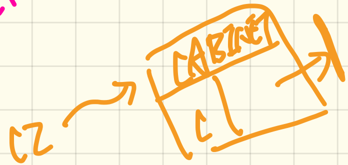


create c1.make ✓

create c2.make ✓

c1.add (c2) ✓

c1.add (c1) → ok



COMPOSITE_EQUIP
children: LIST[EQ.]
invariant
no_self_loop: ??
~~① children.has(current)~~
~~② across children~~

allowed by the current design of the C.D. architecture but it doesn't make sense.

object_comparison

① not children.has (Current)

LIST[EQUIP]

② across children is C → EQUIP.

all

Current ~ C

end

def. class EQUIP

is_equal

if was not redefined

Current = C

⇒ ref

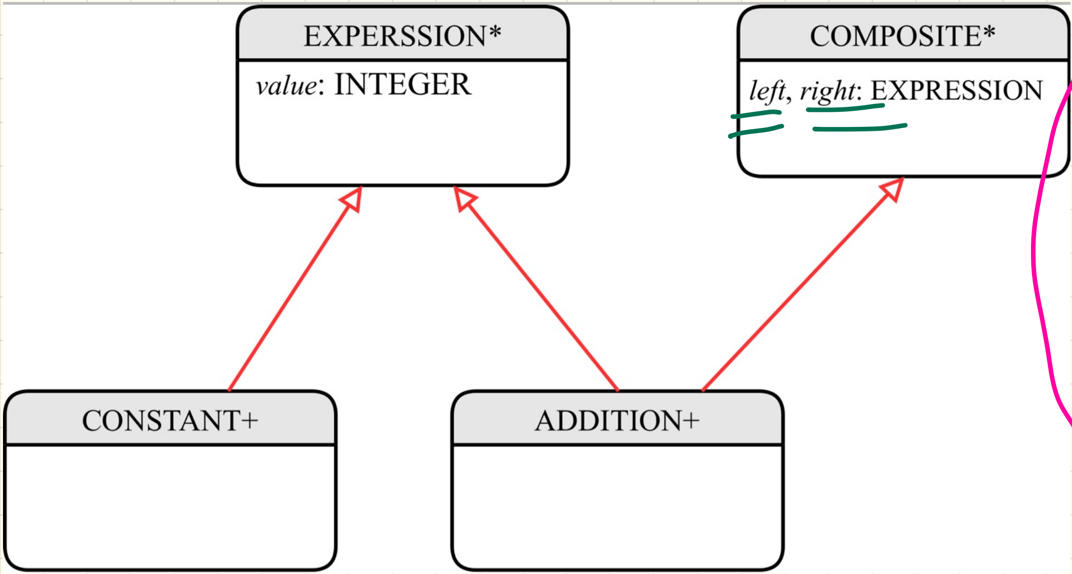
otherwise

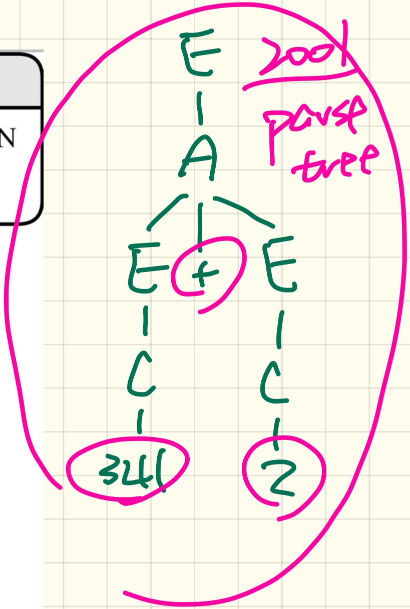
⇒ Current.is_equal(C)

→ comparing contents

not ref.

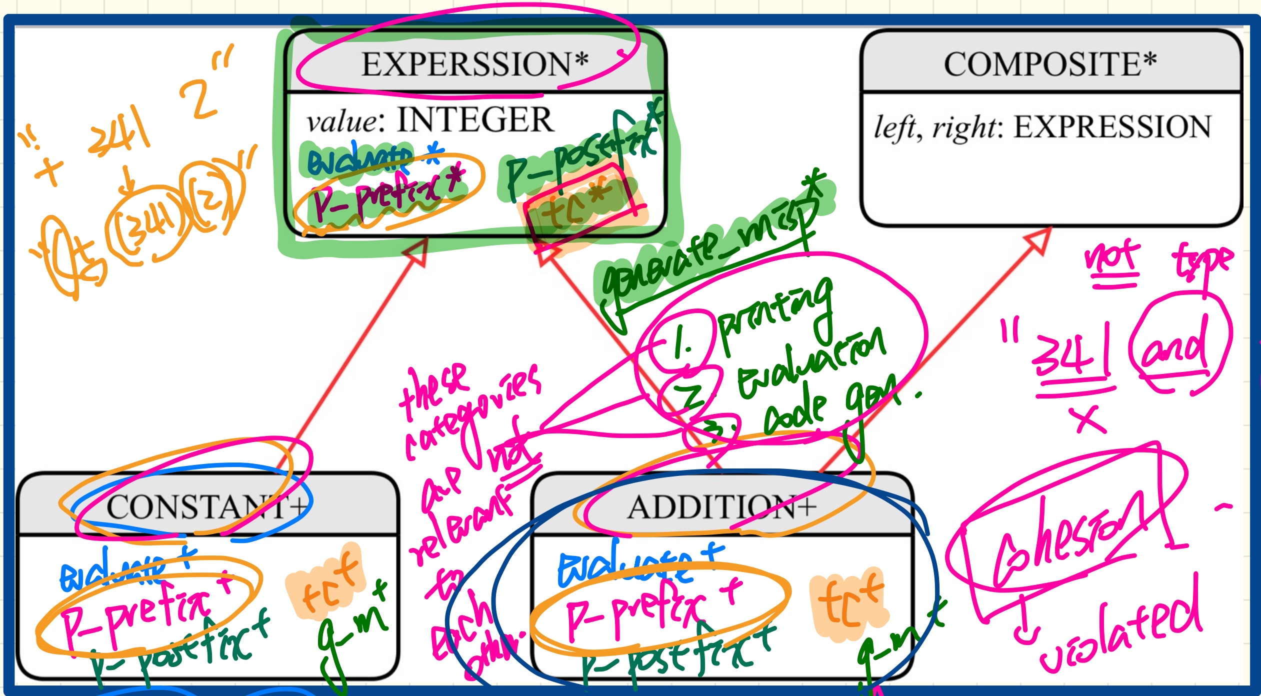
Design of Language Structure: Composite Pattern





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

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



Structure

evaluate 343
print_prefix "+ 341 2"
print_postfix "341 2 +"
type_check true

Operations

341 + 2

1. single choice p.

As the # of ops ↑ do we violate any d. principle?

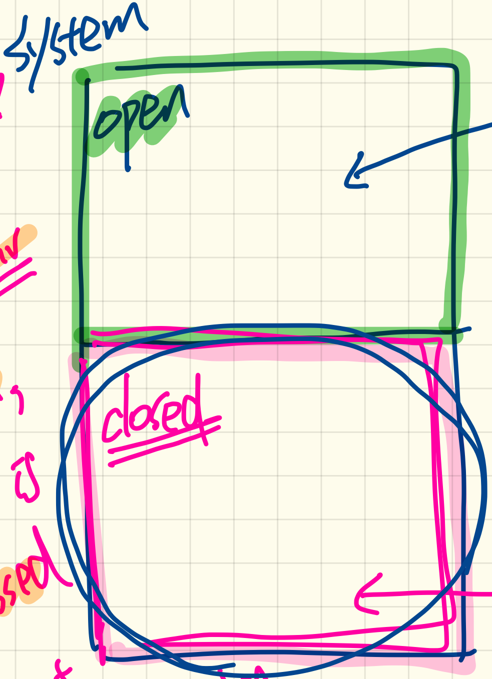
Open/closed Principle

do we satisfy OCP?

↳ ① Yes if there's a clear distinction about which part is open, and which part is closed

② Overtime, the open part should be touched when there's an extend.

↳ the closed part should be touched ① supplier ② very rarely. (if not never)



extensions

↳ ① a new operation
e.g. (code gen).

e.g. ② a new bin. op.
(multiplication)

extensions X

Open closed principle

visitor pattern is only applicable if one of the alternatives is satisfied.

alt 1

language structure (+)

language operation (code_gen)

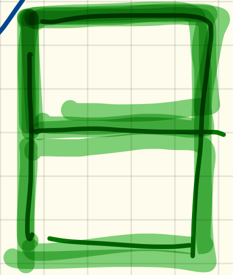
alt 2

language structure (+)

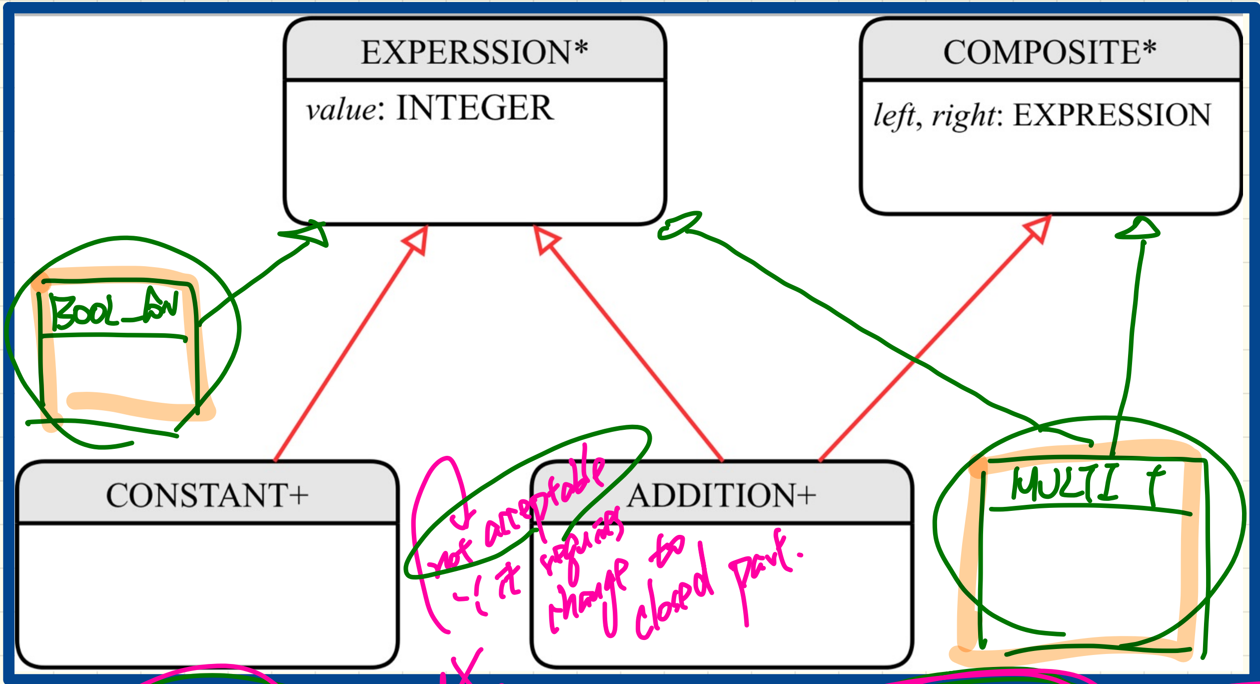
language operation (code_gen)

for visitor pattern

if this was judged to be the case, do not use visitor.



Design of a Language Application: Open-Closed Principle



Structure

- evaluate
- print_prefix
- print_postfix
- type_check

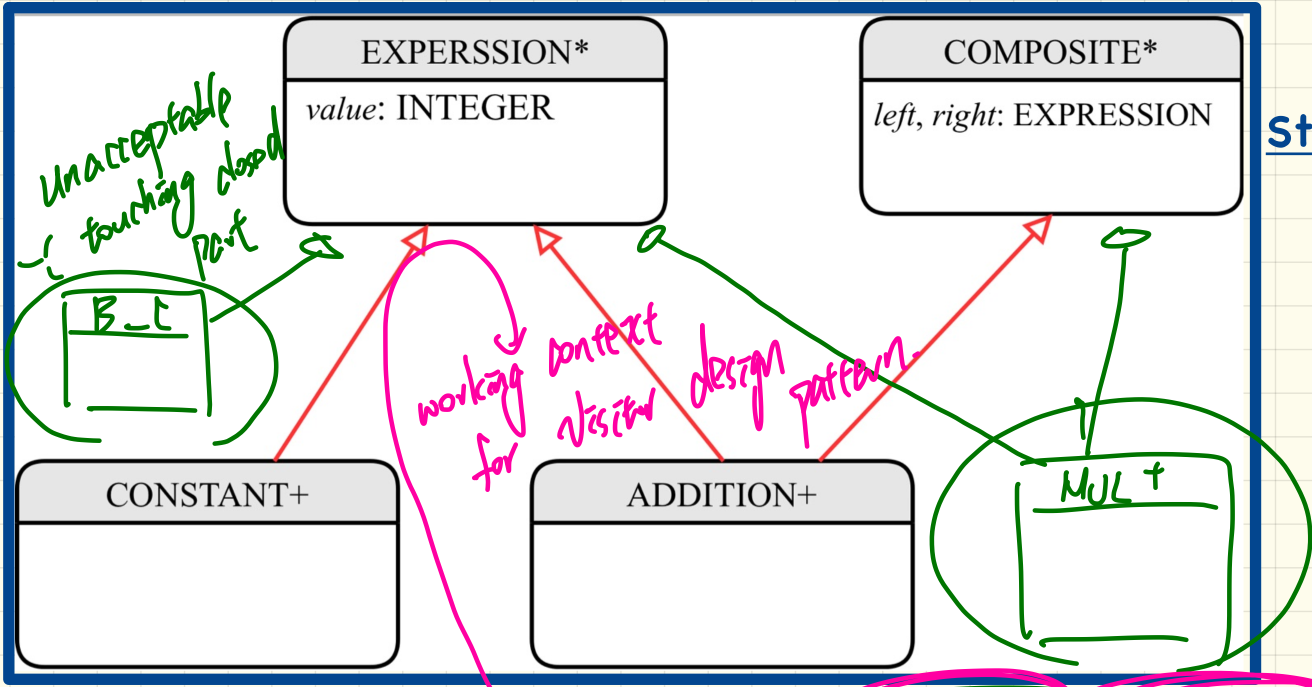
Operations

code-gen

stable
no extension

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Design of a Language Application: Open-Closed Principle



Structure

evaluate
print_prefix
print_postfix
type_check → code-gen

Operations

acceptable touching open part

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

apply_visitor_pattern(my_app)

require

alt 2: structure closed
operations open

do

;

ensure

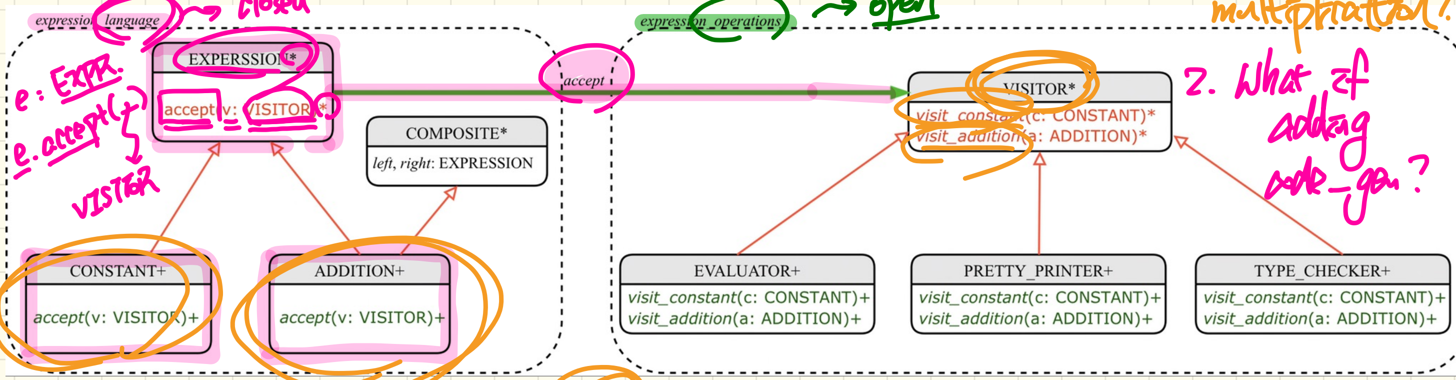
SCP.

cohesion.

Visitor Design Pattern: Architecture

alt 2.

1. What if adding multiplication?



2. What if adding code_gen?

How to Use Visitors

Q1. How many descendant classes of VISITOR?
↳ how many operations for ~~comp.~~ tree.

Q2. How many visit_* commands in VISITOR?
↳ how many effective/implemented language element classes

```
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
```

type

D.T.

visitor.

add

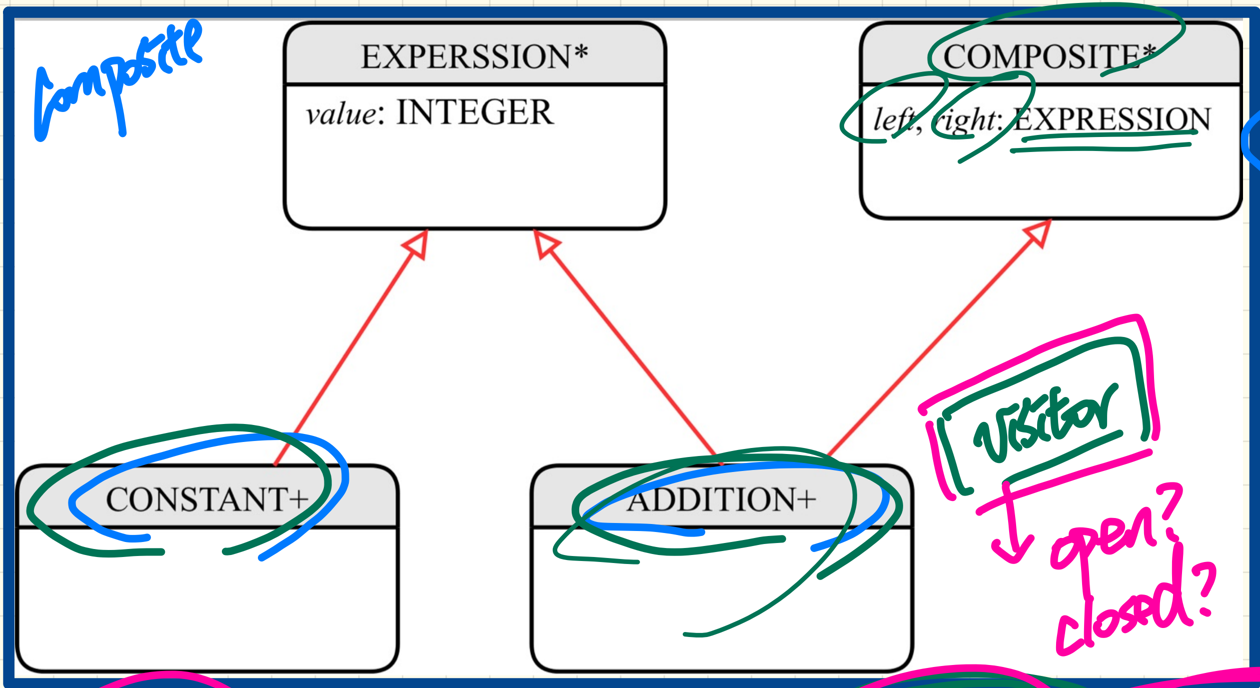
cast

Can I say: $\boxed{\text{v. value?}}$

1 + 2

LECTURE 22
WEDNESDAY MARCH 25

Design of a Language Application: Open-Closed Principle



Structure

- evaluate
- print_prefix
- print_postfix
- type_check

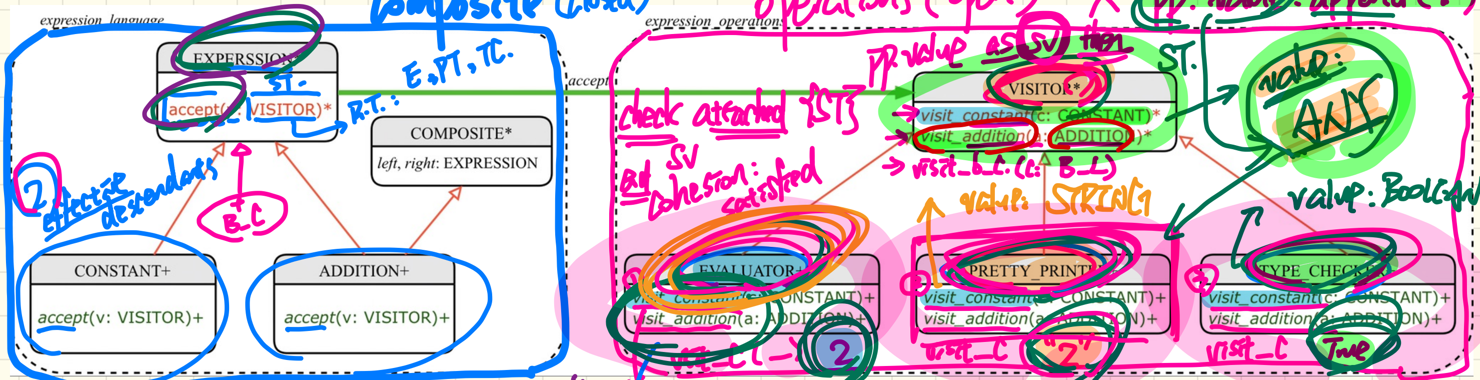
Operations

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Visitor Design Pattern: Architecture

Client of PP: PRETTY_PRT.
X PP.value.append(".")
PP.value as SV then

composite (closed) operations (open)



v. e.accept

value: INT

①, or ②, or ③ which version to call at R.T depends on the D.T. of visitor

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
```

tree

D.T.

INT

alt. ST. VISITOR

X v.value = 3

∵ value is declared

For each descendant of VISITOR, the return value may be of different type.

at the individual visitor descend.

value

Eiffel (no feature overloading)

check attached
of attached

visit_constant (CONSTANT)

visit_addition (ADDITION)

Java (supports method overloading)

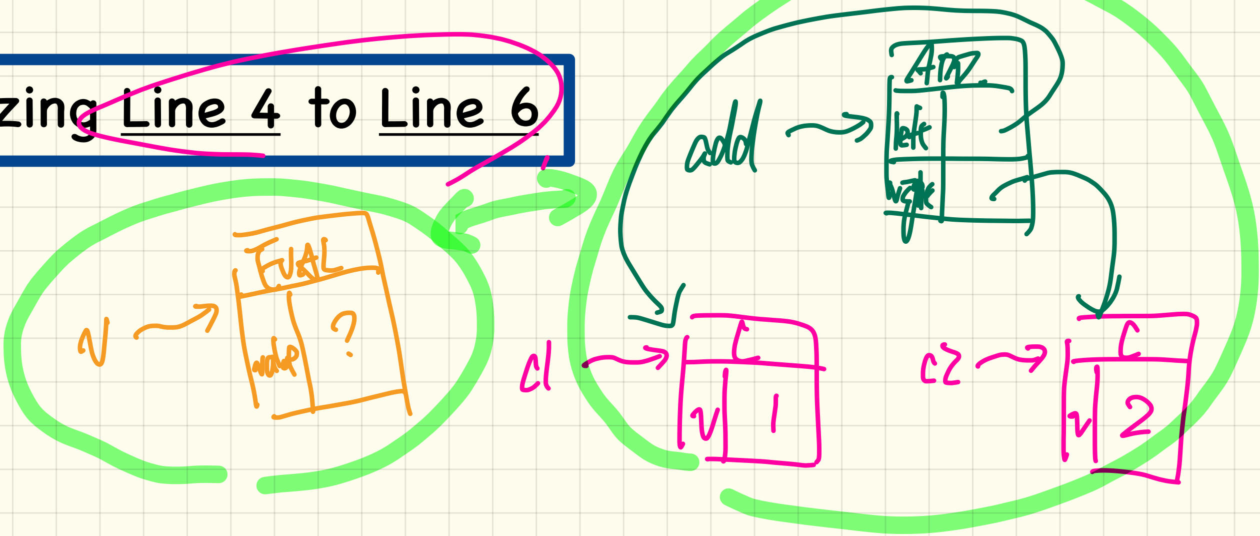
① visitConstant(Constant)
visitAddition(Addition)

② visit (Constant c)
visit (Addition a)

Visitor Design Pattern: Implementation

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

Visualizing Line 4 to Line 6



eval_left.value +

evaluate eval_right.value

Evaluate addition

1 + 2

(1+2) + (3+4)

1 2

How do we evaluate left & right recursively, then combine their results using "+" ?

add

+

eval : EVALUATOR

eval.visit_addition(add)

eval_right : EVALUATOR.

eval.

eval_left :

eval_left.visit_addition(add.left)

~~ADD.~~

~~ADDITION~~

not compile

eval_left.value

add.left

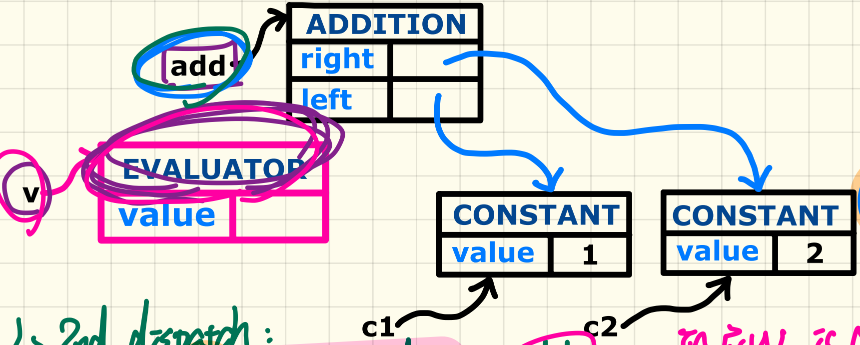
↳ ST: EXP.

add.right

add.left.accept(eval_left)

add.right.accept(eval_r.)

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



Tracing add.accept(v)

Double Dispatch ↳ dyn. bind.

× 2

add.accept(v)

↳ 1st dispatch :

∵ DT of add is ADDITION

∴ accept in ADDITION is called

↳ execute : v.visit_add(add)

↳ 2nd dispatch :

v.visit_addition(add)

∵ DT of v is EVAL. ∴ visit_add

in EVAL is called

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

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

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

add

add

add

Exercise. Explain D.D. for ___.

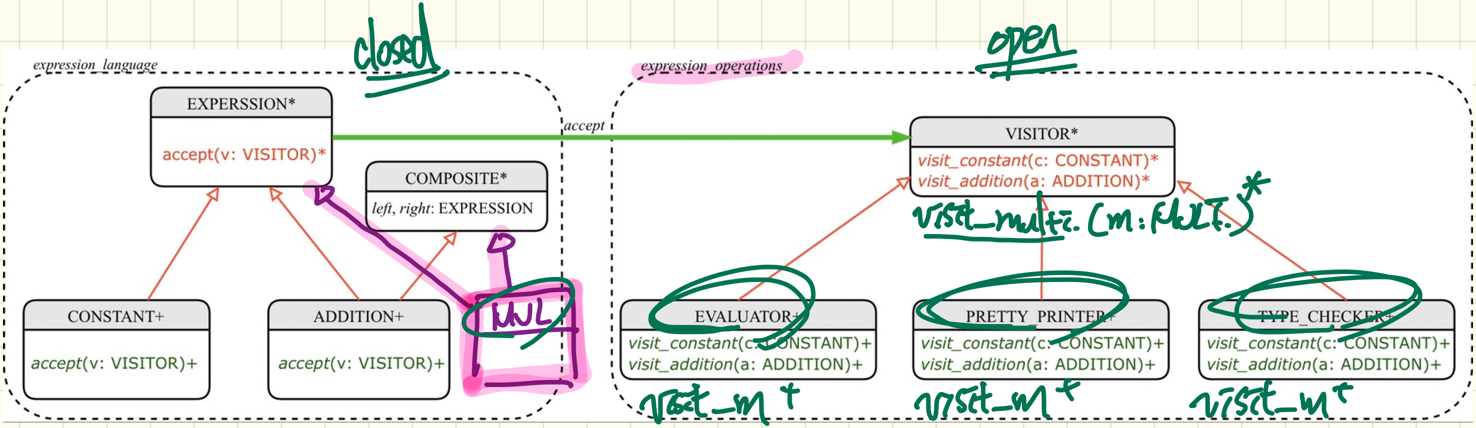
1st dispatch

Explain D.D. for ___.

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

v : DT is EVALUATOR

Visitor Pattern: Open-Closed and Single-Choice Principles



MUL. Violates S.C.P.

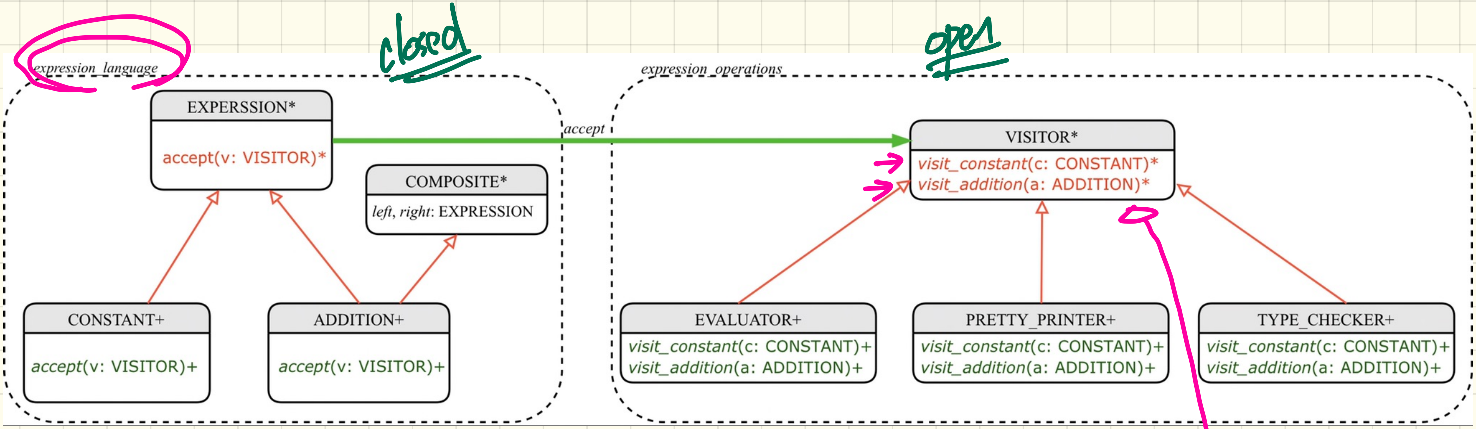
What if a new language construct is added?

↳ added to the closed part X

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

↳ language structure

Visitor Pattern: Open-Closed and Single-Choice Principles



+
CODE_GEN.
visit_c+
visit_a+

code-gen.

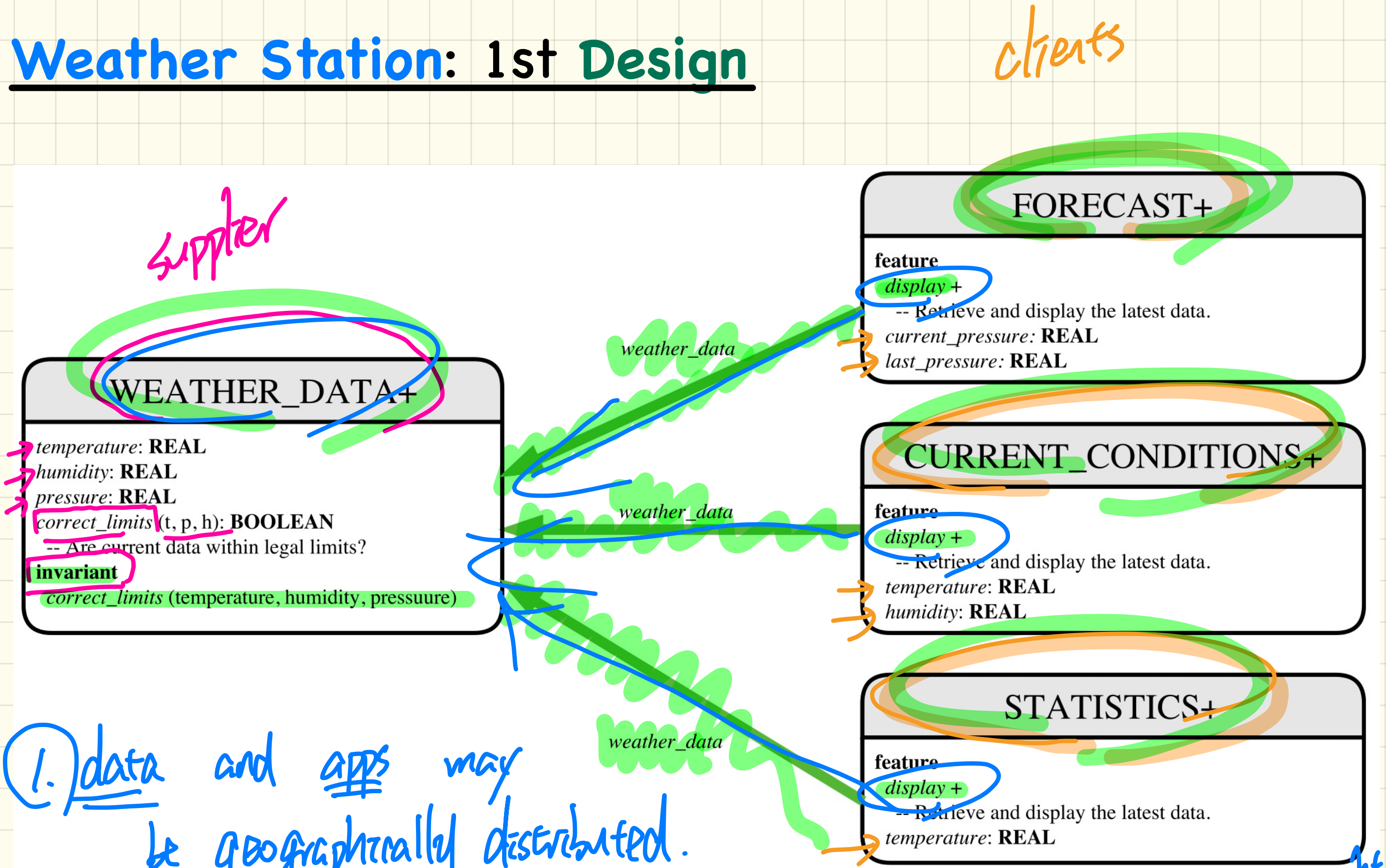
What if a new language operation is added?

↳ satisfies S.C.P

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

↳ operation.

Weather Station: 1st Design



1. data and apps may be geographically distributed.

2. when "display" is invoked, retrieve data (which might be remote)

- Observer
- Event-Driven Design

extra lecture

tentatively.

Friday

1pm - 2:30pm

- software verification

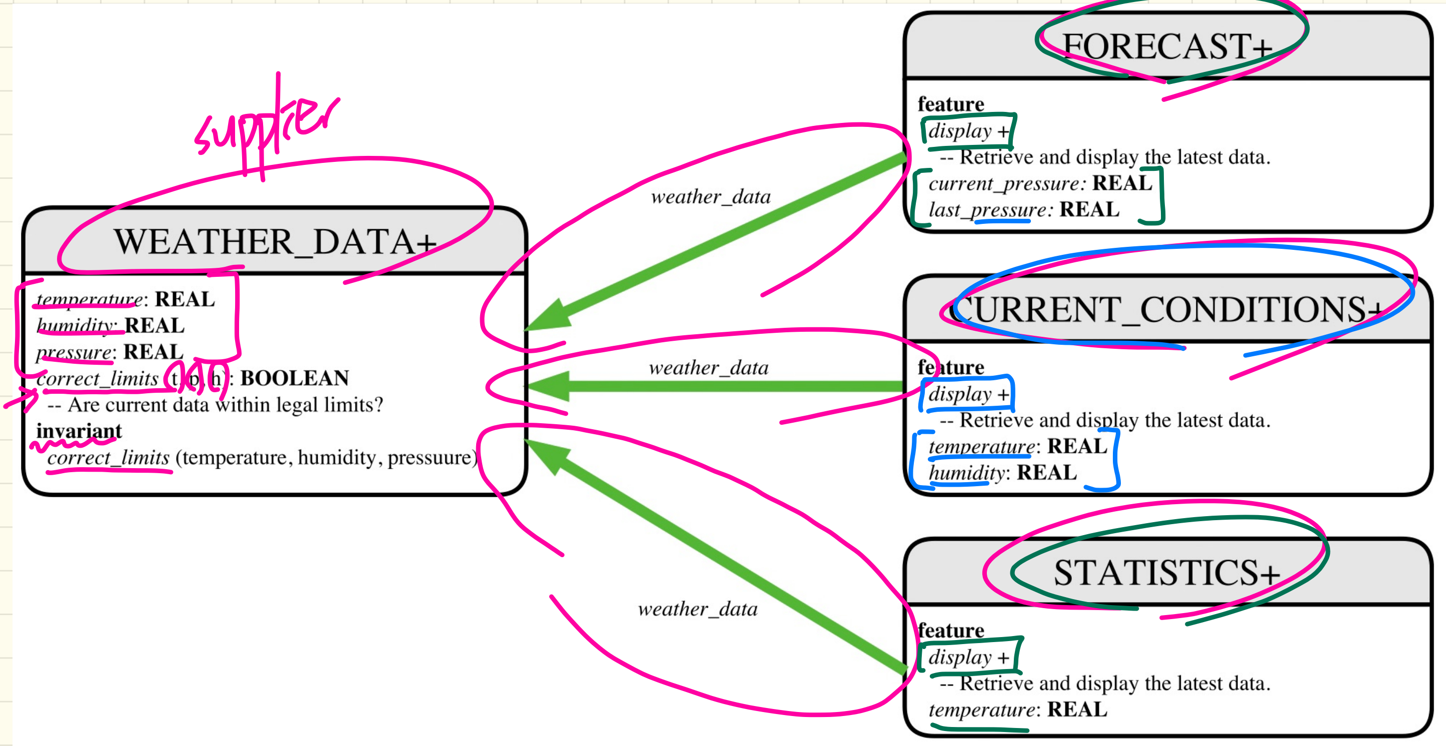
next M/W

LECTURE 23

FRIDAY MARCH 27

Weather Station: 1st Design

one-directional clients



Weather Station: 1st Implementation

Q. Whenever we update, is it really necessary?

geographically distributed

```
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
```

wd

app1

app2

app3

remote procedure call.

```
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
```

retrieve the latest value

```
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
```

retrieve

```
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
```

retrieve

- Display periodically.
- First step to display is to retrieve the latest measure.

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
```

necessary

unnecessary

① ②

there's no change on measure

② ∵ no change on measurement ∴ unnecessary

necessary

```
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
```

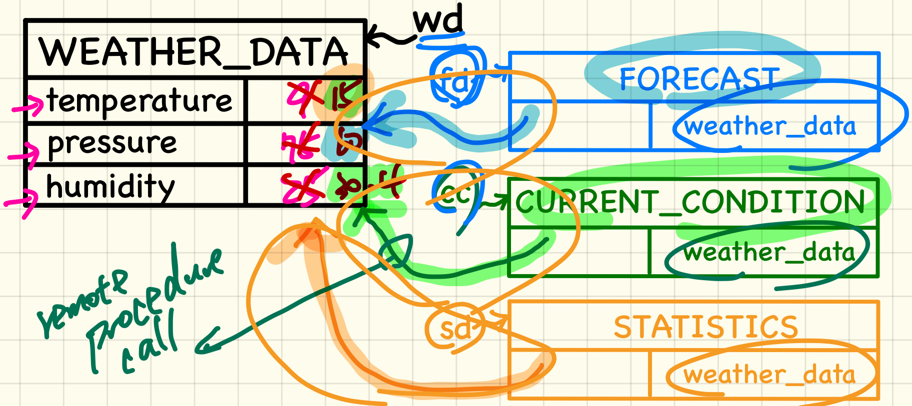
wd

```
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
```

wd

a.w_d := wd

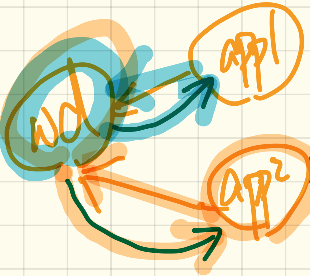
```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature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



1st design

app's update may be unnecessary
(if they retrieve the same value)

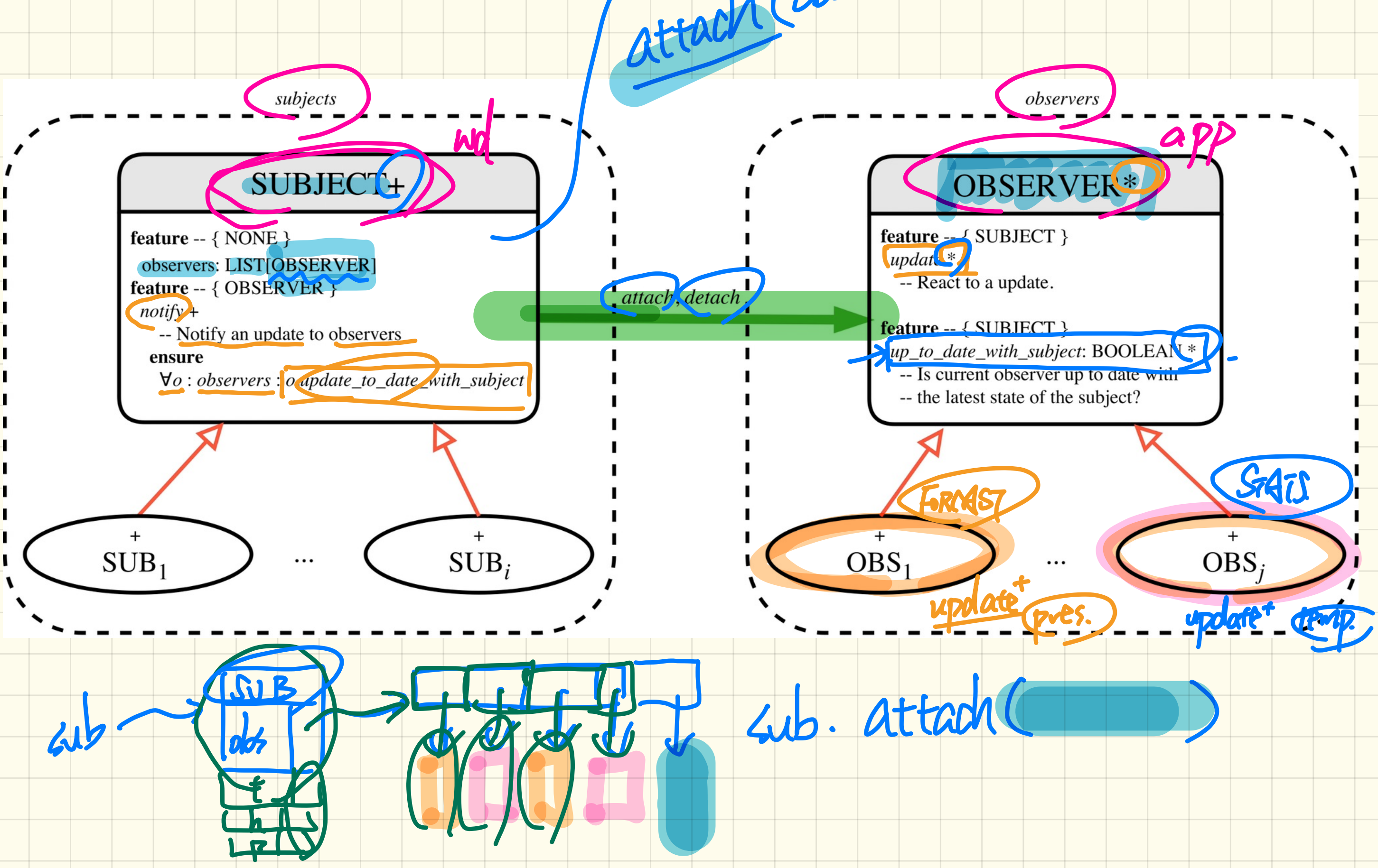
2nd design (observer pattern).



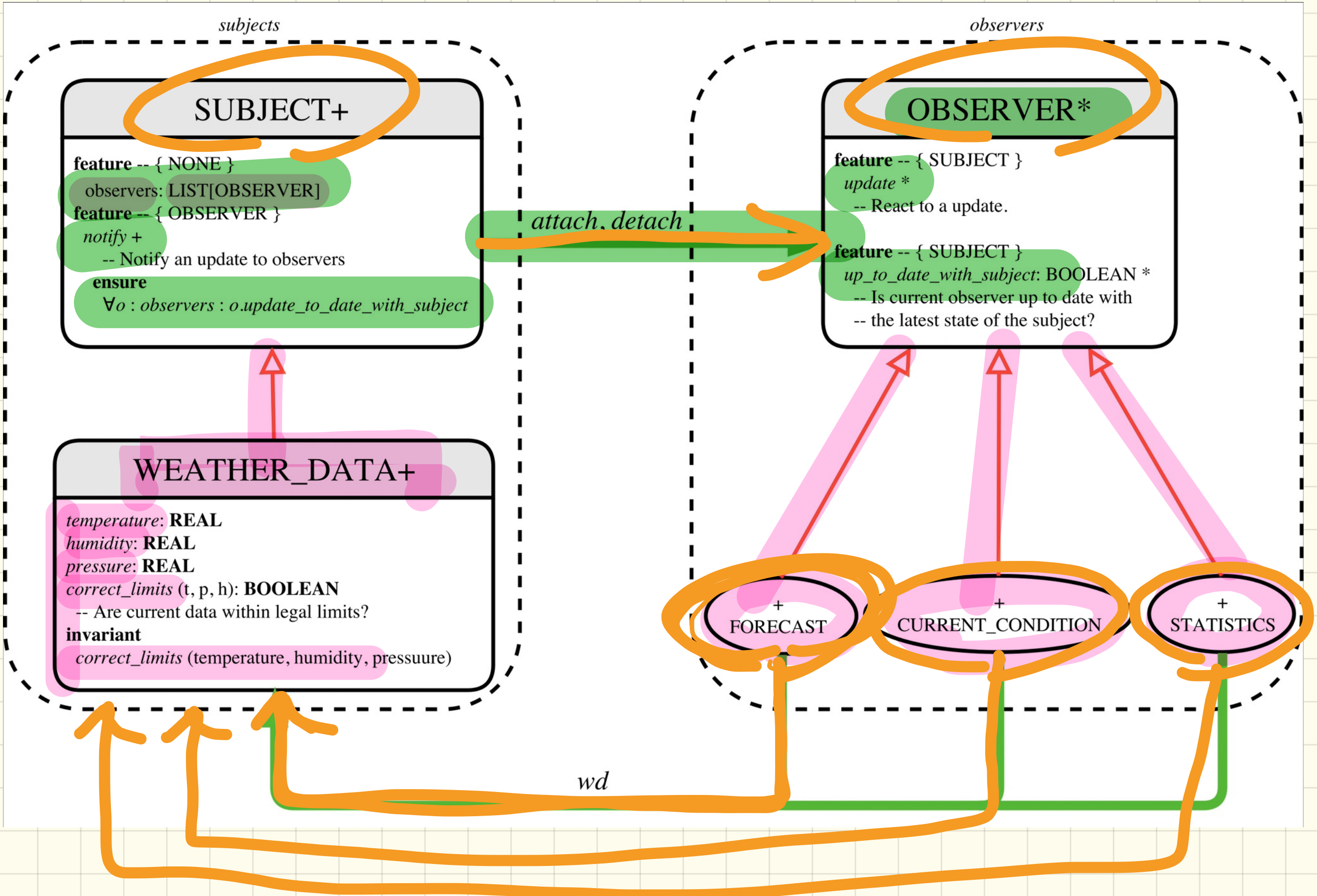
let all apps be **passive** on updating
only when there's a change on the weather data
will weather data invoke updates on apps

The Observer Pattern

attach(obs : OBSERVER)



Observer Pattern: Application to Weather Station

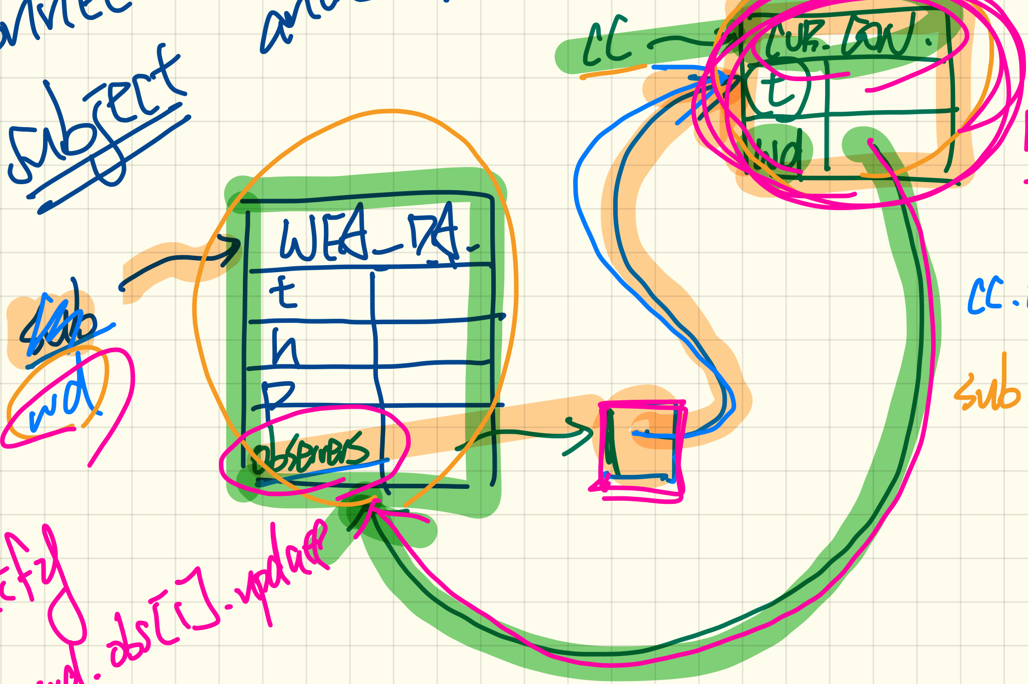


Bi-directional connections between subject and observers

cc.wd := wd

wd.observers.extend(cc)

wd.attach(cc)



cc.wd = sub

sub.observers[i]

=

cc

wd. notify

↳ wd.obs[i].update

Weather Station: Subject

RPC is still necessary when notify is called on each observer. (but it's necessary)

```
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
```

call update on each observer when necessary.

Polymorphic list

ST: OBSERVER.

obs[1].update

obs[2].update.

Dynamic binding:

e.g. obs[i] has d.t. CURRENT_CONDITIONS

call update in

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
```

p.

old pressure

RPC

```
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
```

h. t.

```
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
```

t

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
```

across wd.observers is obs loop obs.update

no update invoked

cc.wea_da := wd

cc.wea_da.attach (cc)

no update invoked

```
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
```

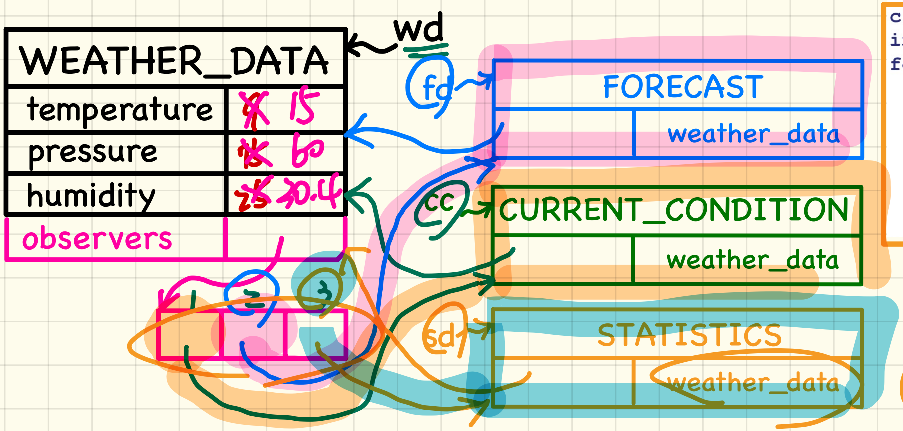
```
class CURRENT_CONDITIONS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
       weather_data.observers.has (Current)
    end
```

wd

wd

cc

```
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
```



① Bi-directional links

② notify (subject on demand)

LECTURE 24

MONDAY MARCH 30

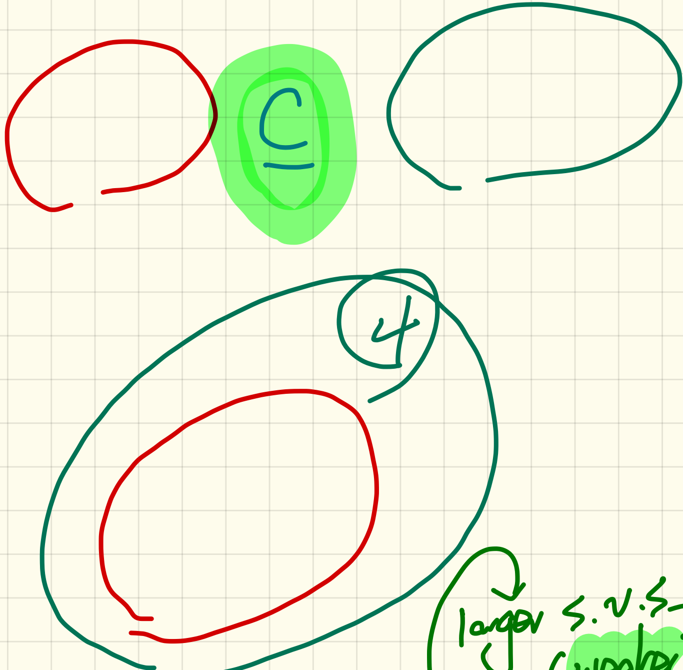
Assertions: Weak vs. Strong

$x > 3$

$4, 5, 6, 7, \ldots, \infty$

$x > 4$

$5, 6, 7, \ldots, \infty$



$x > 4 \Rightarrow x > 3$

larger s.v.s. (weaker)

↳ smaller satisfying value set (stronger) runtime

BANK1	
b	~~0~~

class inv. violation.

BANK_1

balance : INT

invariant

I1: balance > 0

BANK_2

balance : INT

invariant

I2: balance ≥ 0

BANK2	
b	0

valid object state

Which class invariant is stronger? I1.

I_1: 1, 2, 3 --- ∞

I_2: → 0

$I_1 \Rightarrow I_2$

I_1 → stronger.

Assertions: Preconditions

precondition violation

withdraw_v1(amount: **INTEGER**)
require 0 > 0 ≡ F
P1: amount > 0

acc.withdraw_v1(0)

require more

$P_1 \Rightarrow P_2$

acc.withdraw_v2(0)

no pre. violation

withdraw_v2(amount: **INTEGER**)
require
P2: amount ≥ 0

withdraw_v2 is more tolerant on accepting input values

require less

Assertions: Postconditions

f1(i: **INTEGER**): **BOOLEAN**
    ensure
        Q1: **Result** = (i > 0) $\boxed{\vee}$ (i **mod** 2 = 0)

weaker

f2(i: **INTEGER**): **BOOLEAN**
    ensure
        Q2: **Result** = (i > 0) $\boxed{\wedge}$ (i **mod** 2 = 0)

✓

$\boxed{\text{stronger}}$ → more demanding task for supplier

smaller satisfying value set ⇒ more demanding.

3, 5, −2, 7, ... −4, −6

Q2

2, 4, 6, 8, 10

Program Correctness: Example (1)

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

spec

imp.

4

3

13

too weak

c.e. 4

F

postcond. violation.

not correct.

Correctness of program: (relative)

implementation satisfies specification.

Given valid input (precond. satisf.), executing the implementation will (1) terminate. (2) upon termination, the postcondition is satisf.

4 counter example

$4+9 > 13$ F.

Program Correctness: Example (2)

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

F

Guiding Principle

cannot be too weak

whether a precond. is too strong or not, it's up to the designed.

Correct

i: 6, 7, 8, - - -

valid input values

i + 9 is always > 13

Incorrect (?)

i > 5 (precondition) stronger than necessary.

disallow some input value that would not cause postcon. violation!

5 currently not considered a valid input.

If 5 was allowed, 5 + 9 = $\boxed{14 > 13}$ is T

V1

```
f
  require
    Q
  do
    S
  ensure
    R
  end
```

verify whether when Q is satisfied, executing S will establish R.

When you justify that program is incorrect. you may fix: Q, S, R

Hoare Triple → Tony Hoare → Quick sort

Incorrect

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

counter example

cannot be proved as TRUE

precond. postcond.

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

Correct

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

can be proved as a theorem

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

f

require

Q

do

S

ensure

R

end

input

formulate

Hoare Triple

Correctness Predicate

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

S is correct with respect to Q and R

means

predicate which may be $\boxed{\text{proved}}$ or disproved.

Hoare Triple as a Predicate

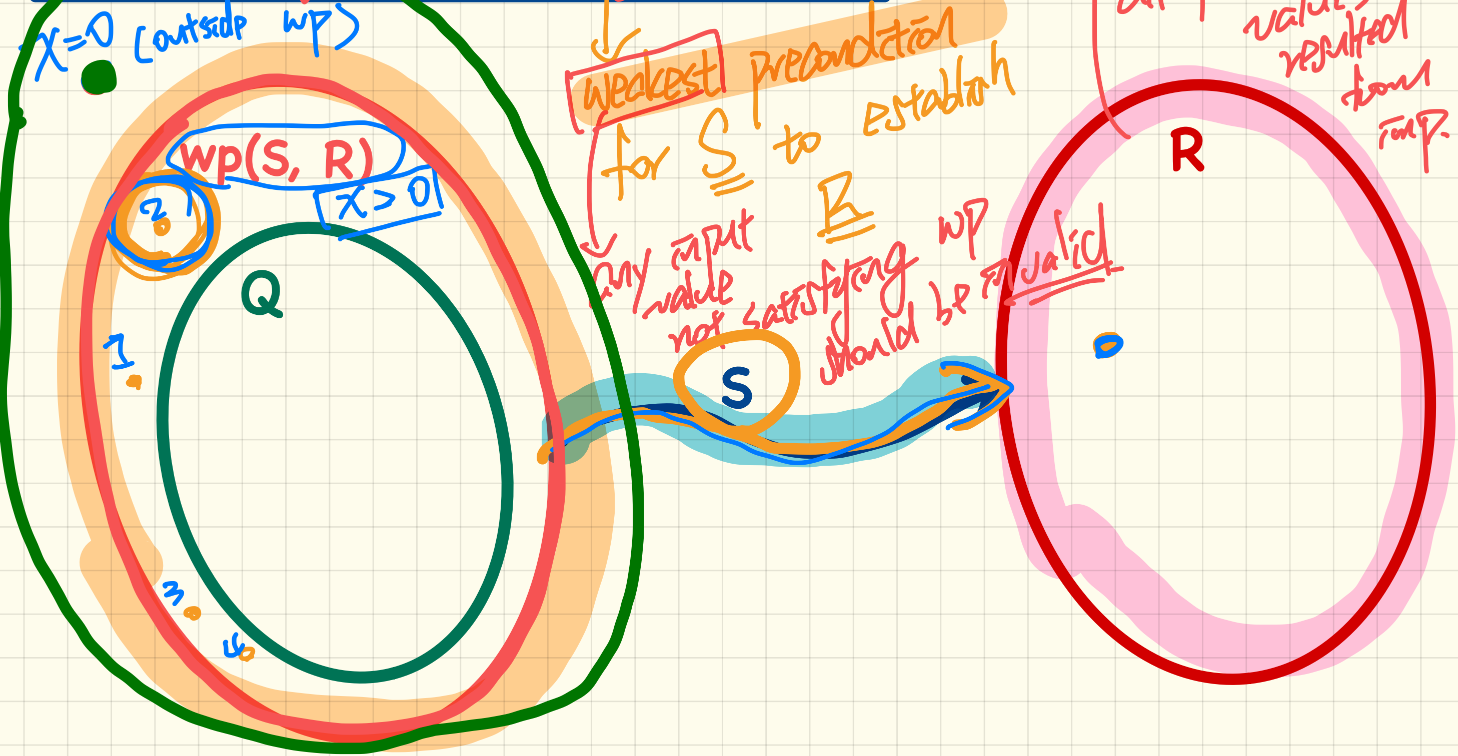
$x = old\ x + 1$

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

weakest precondition for S to establish R

any input value not satisfying wp should be invalid

all post-state values resulted from imp.



Program Correctness: Revisiting Example (1)

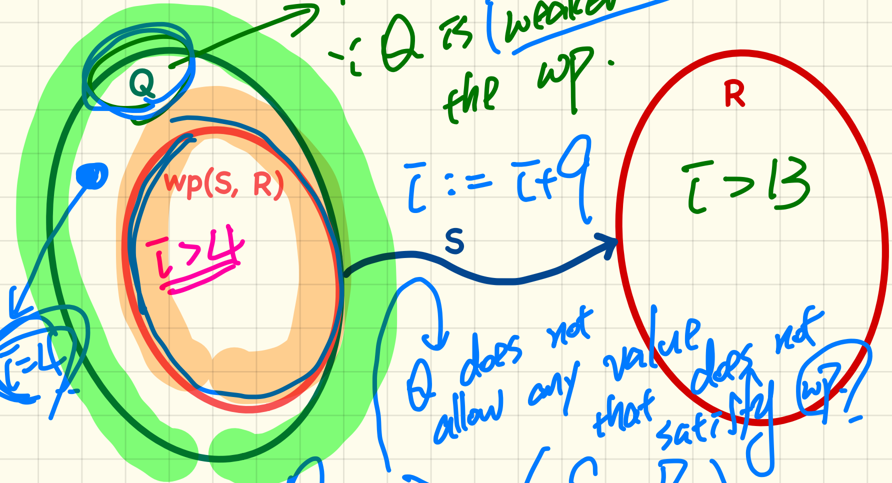
$i > 4$ $wp(i := i + 9,\ i > 13)$

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

incorrect

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

incorrect program
∵ Q is weaker than the wp.



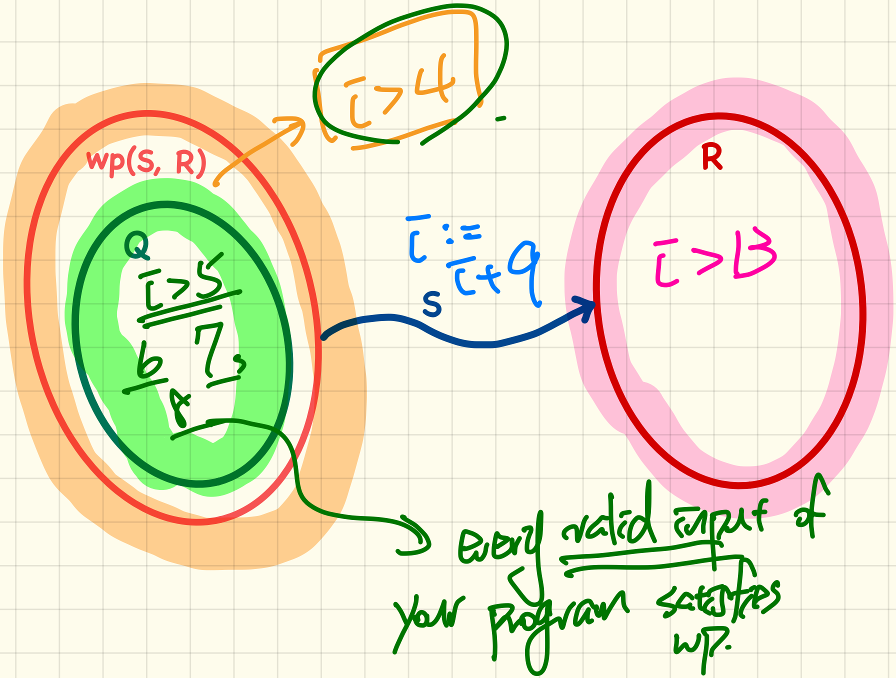
Q does not allow any value that does not satisfy wp?

$Q \Rightarrow wp(S, R)$

Program Correctness: Revisiting Example (2)

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

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

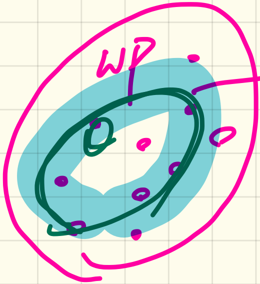


Program

f
require
Q
do
S
ensure
R

↳ program

① formulate into $\{Q\}\ S\ \{R\}$



only values satisfying wp should be valid input.

② Calculate $wp(S, R)$

③ Prove or disprove:

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

Hoare triple proof (wp)

only makes sure your precond. is not too weak

F

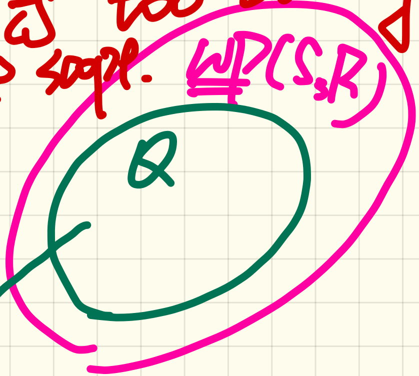
$Q \Rightarrow wp(S, R)$

whether Q is too strong is beyond the scope.

Q is no weaker than wp.

f require false

by def. correct



what is the Q that is no weaker than any wp?

Rules of Weakest Precondition: Assignment

$$wp(x := \underset{x-1}{e}, R) = R[x := e]$$

imp: $x := x-1$

post: $x > 0$

$\rightarrow e > 0$.

precondition (values should be in pre-state)

wp??

$x = 3$

x

$x := e$ ($x-1$)

$x > 0$

(post-state value).

$\rightarrow x = 3 - 1 = 2$.

e

Correctness of Programs: Assignment (1)

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

$$\{x > 0\}\ Q\ \text{True} \quad \{??\}\ \texttt{x := x + 1}\ \{x > x_0\}$$

impl → post.

does it matter?

wp is True.

$$Q \Rightarrow \text{True}$$

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

pre-state value

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

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

$x_0 + 1$

R

$$= \cancel{x_0} + 1 > \cancel{x_0} = 1 > 0$$

wp. = True

For this prog, wp is T, any precond. is stronger than that.

Lecture 25
Wednesday April 1

Hoare Triple as a Predicate

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

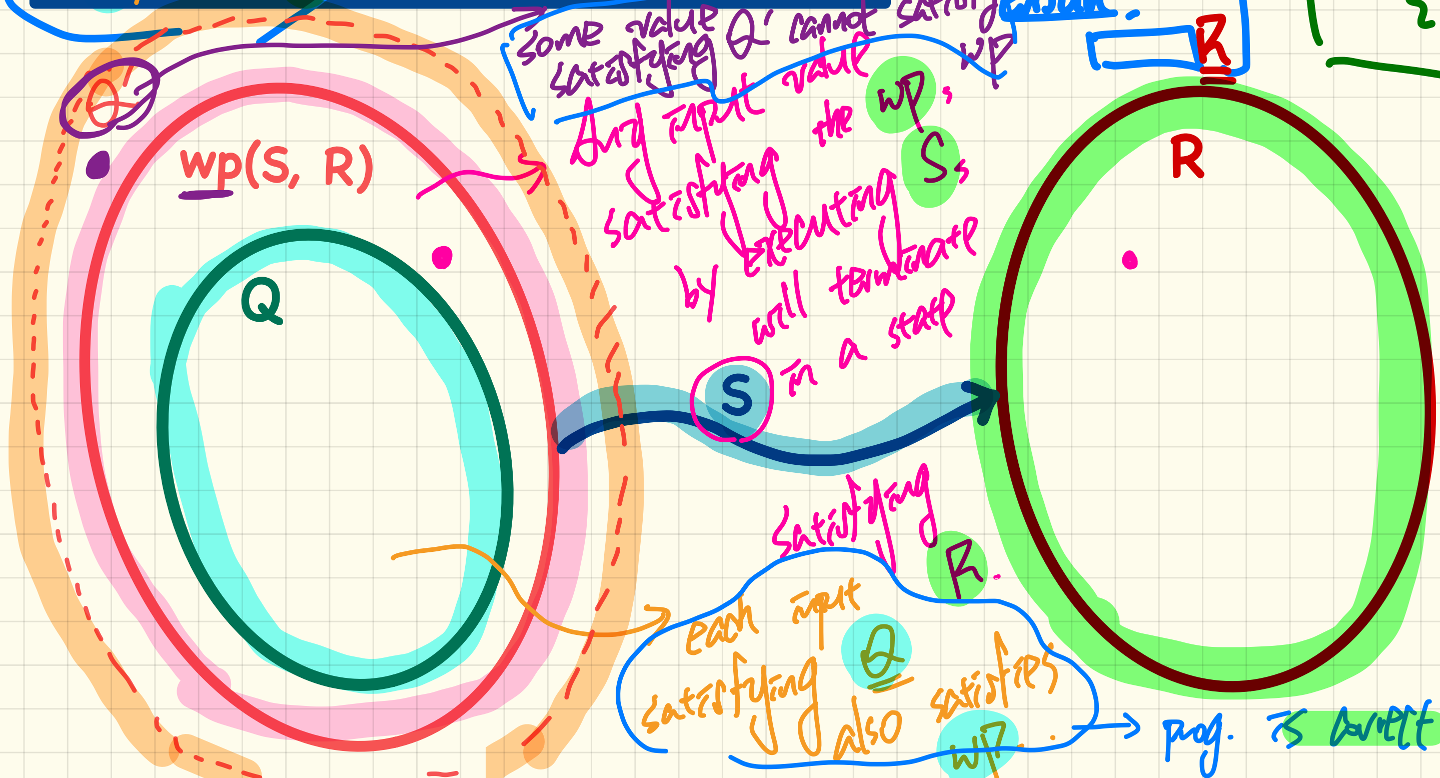
prog is f
require Q.
do S.
ensure R.

BOOLEAN (PREDICA)

some value satisfying Q' cannot satisfy wp

Any input value satisfying the wp, S, by executing S, will terminate in a state satisfying R.

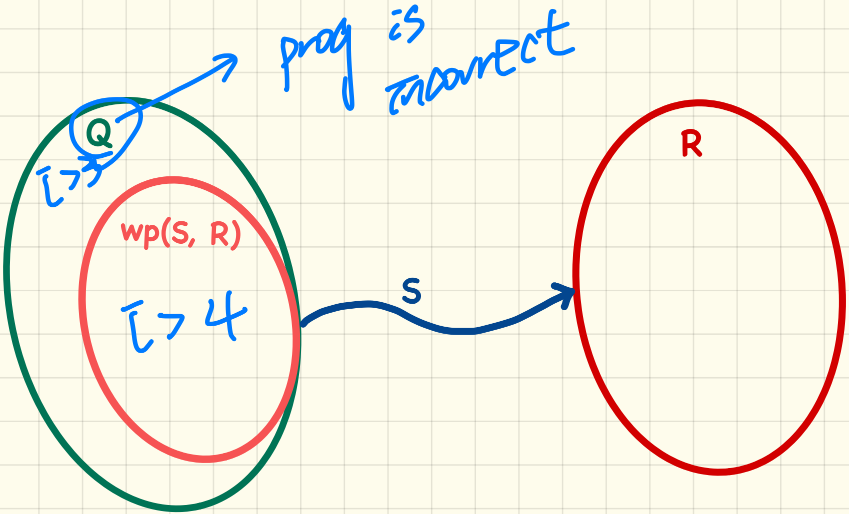
each input satisfying Q also satisfies wp. → prog. is correct.

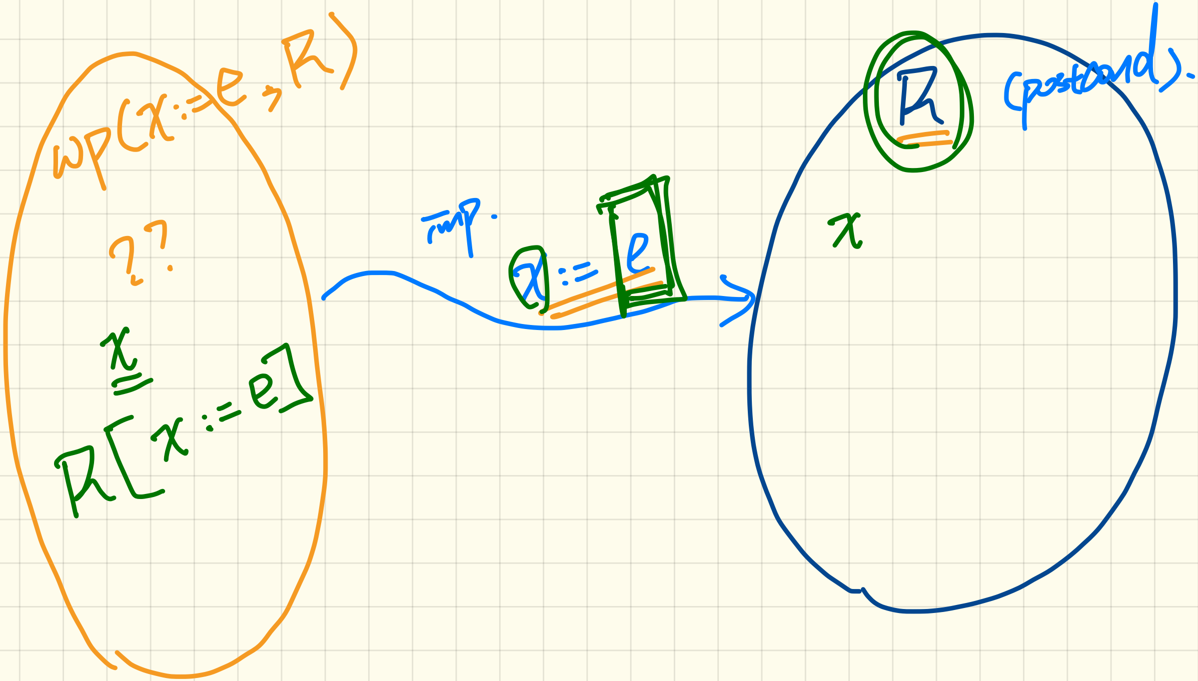


Program Correctness: Revisiting Example (1)

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

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




wp(x := e, R)
??
x
R[x := e]
imp.
x := e
x
R
(postcond).

Correctness of Programs: Assignment (2)

What is the weakest precondition for a program `x := x + 1` to establish the postcondition ~~$x > x_0$~~ $x = 23$?

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

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

$$= \{\text{def. of } wp \text{ for } :=\}$$

$$\boxed{x} = 23\,[x := x+1]$$

$$= x+1 = 23 \equiv \boxed{x = 22}$$

prog not correct

$x = 22$

$x = 21$

Rules of Weakest Precondition: Conditionals

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

if B then

[S_1

else

[S_2

end

{R}

$B \Rightarrow wp(S_1, R)$

① $\wedge$ ② $\vee$??

if : –

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

else : –

y=1,x=-4

Rules of Weakest Precondition: Conditionals

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

wp will just be T

$B \Rightarrow wp(S1, R)$
$\vee$ → incorrect
$\neg B \Rightarrow wp(S2, R)$

vs.

$B \Rightarrow wp(S1, R)$
$\wedge$ → correct
$\neg B \Rightarrow wp(S2, R)$

→ if branch establishes R and else branch establishes ?? R

should this program be correct? postcond. violation.

x≥0

Consider:

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

x+1

y>0 T

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

$\vee$

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

$1 \leq 0$ F $\Rightarrow$ — $\equiv$ T

mistakenly, wp will still evaluate to T

wp should not evaluate to T if an input value can violate postcon.

y=1 x=-4

Rules of Weakest Precondition: Summary

$$wp(\mathtt{x} \;\; \mathtt{:=} \;\; \mathtt{e},\ \boldsymbol{R}) = \boldsymbol{R}[x := e]$$

$$wp(\mathbf{if}\ \boxed{B}\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{end},\ \boldsymbol{R}) = \left(\begin{array}{l} \boxed{B} \Rightarrow wp(S_1,\ \boldsymbol{R}) \\ \wedge \\ \neg \boxed{B} \Rightarrow wp(S_2,\ \boldsymbol{R}) \end{array}\right)$$

$$wp(S_1 \;\; \mathtt{;} \;\; S_2,\ \boldsymbol{R}) = wp(S_1,\ wp(S_2,\ \boldsymbol{R}))$$

Proof Rules using Weakest Precondition

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

2 ways to proving correctness

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

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

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

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

Q

if B

S_1

else

S_2

{R}

① prove

$Q \Rightarrow wp(\underline{\qquad}, R)$

②

$\{Q \land B\}\ S_1\ \{R\}$

$\land$

$\{Q \land \neg B\}\ S_2\ \{R\}$

Correctness of Programs: Conditionals

Is this program correct?

$\{x > 0 \land y > 0\}$
```
if x > y then
  bigger := x ; smaller := y
else
  bigger := y ; smaller := x
end
```
$\{bigger \geq smaller\}$

--- $x \leq y$

$\rightarrow S$

To prove, follow 2 steps.

① calculate $wp(S, b \geq s)$

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

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

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

② Prove:

$\boxed{x > 0 \land y > 0} \Rightarrow \boxed{??}$

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

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

T

$?$

$\underline{\underline{S_1}}$ $\quad ; \quad$ S_2 $\quad \underline{R}$

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

S_1 terminates before S_2 is executed

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

phase 2

Correctness of Programs: Sequential Composition

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

① Step1: Calculate $wp(tmp := x ; x := y ; y := tmp, x > y)$

$= \{$ def. of wp for ; $\}$

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

$= \{$ def. of wp for ; $\}$

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

$= \{$ def. of wp for $:= \}$

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

$= \{$ def of wp for $:= \}$

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

$= \{$ def. of wp for $:= \}$

$y > x$

Prove

② True $\Rightarrow y > x$

$= \{$identity of $\Rightarrow\}$

$\boxed{y > x}$

$\boxed{\text{not}}$ a tautology (theorem)

counterexample: any x, y satisfying $\neg(y > x)$

e.g. $y = 3$, $x = 4$

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}
```

stay condition

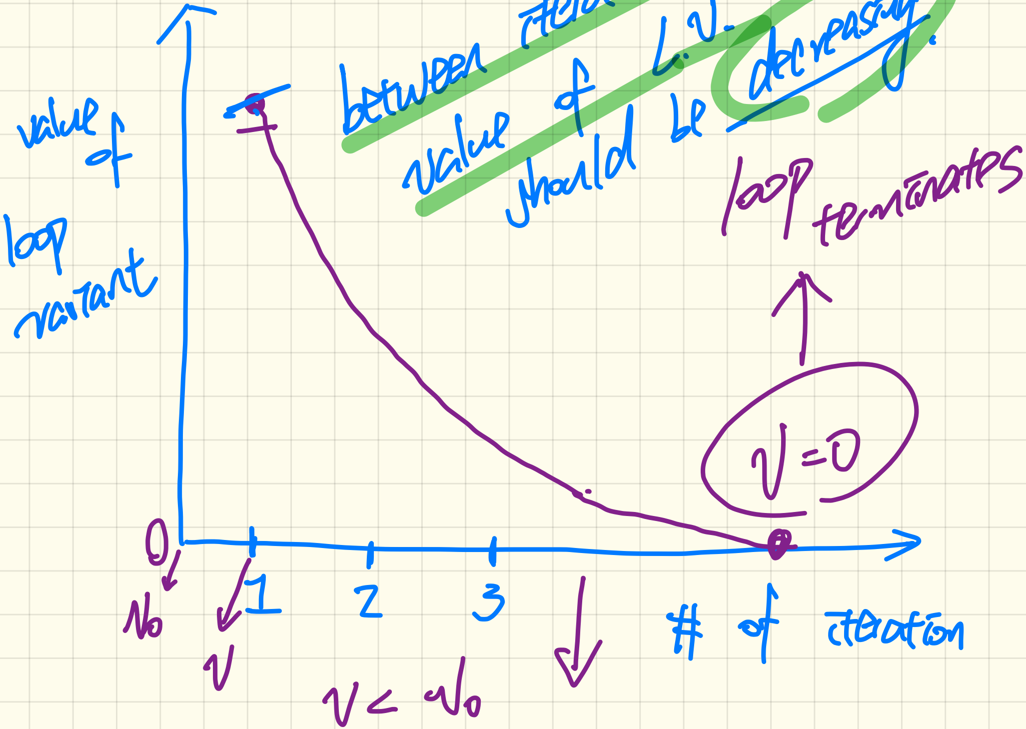
```
from
  i := 1
until
  i = 10
loop print(i)
  i := i + 1
end
```

1~9

```
int i = 1;
while( ¬(i = 10) ){
  print(i)
  i++;
}
```

Loop Variant

between iterations, value of L.V. should be decreasing.



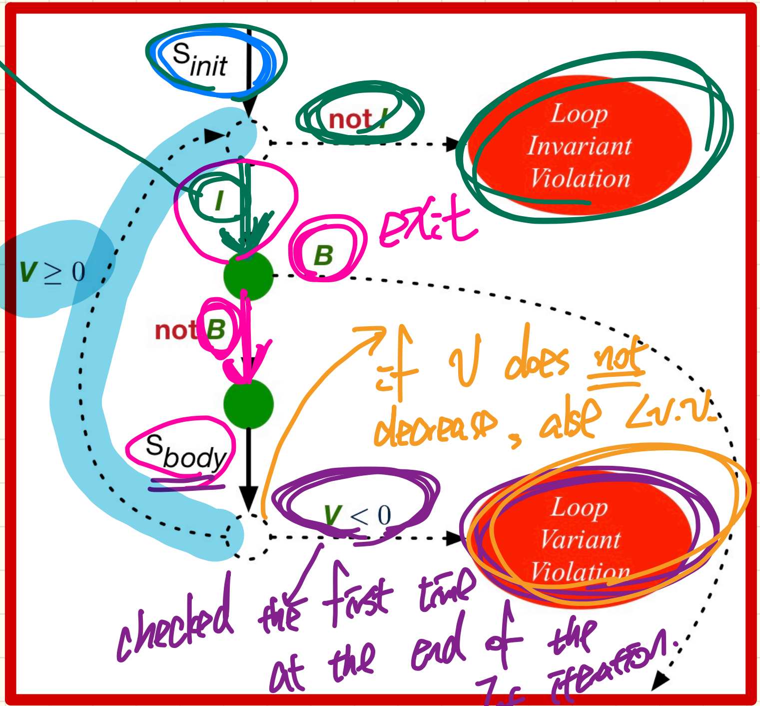
Contracts of Loops

Syntax

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

checked for the first time before the 1st iteration.

Runtime Checks



Contracts of Loops: Example

Syntax

```
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
```

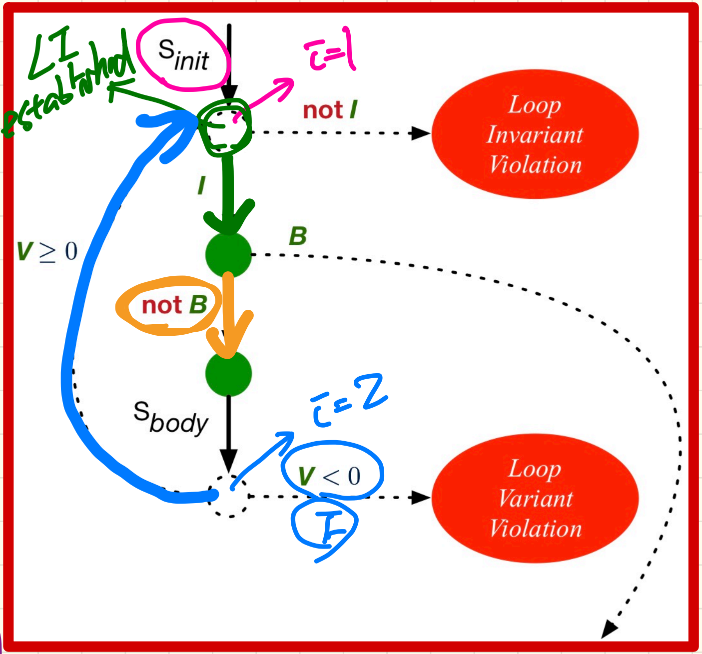
i > 5
as soon as i gets to 6, we exit.

6 - 2 = 4.

the last time LV is checked.
6-6 = 0.

i = 1
2
3
4
5 | 6 exit

Runtime Checks

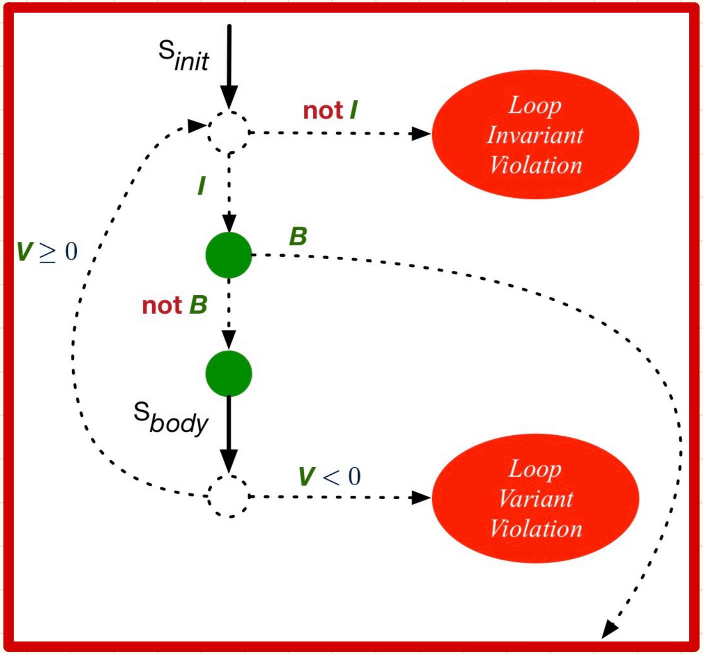


Contracts of Loops: Violations

Syntax

```
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

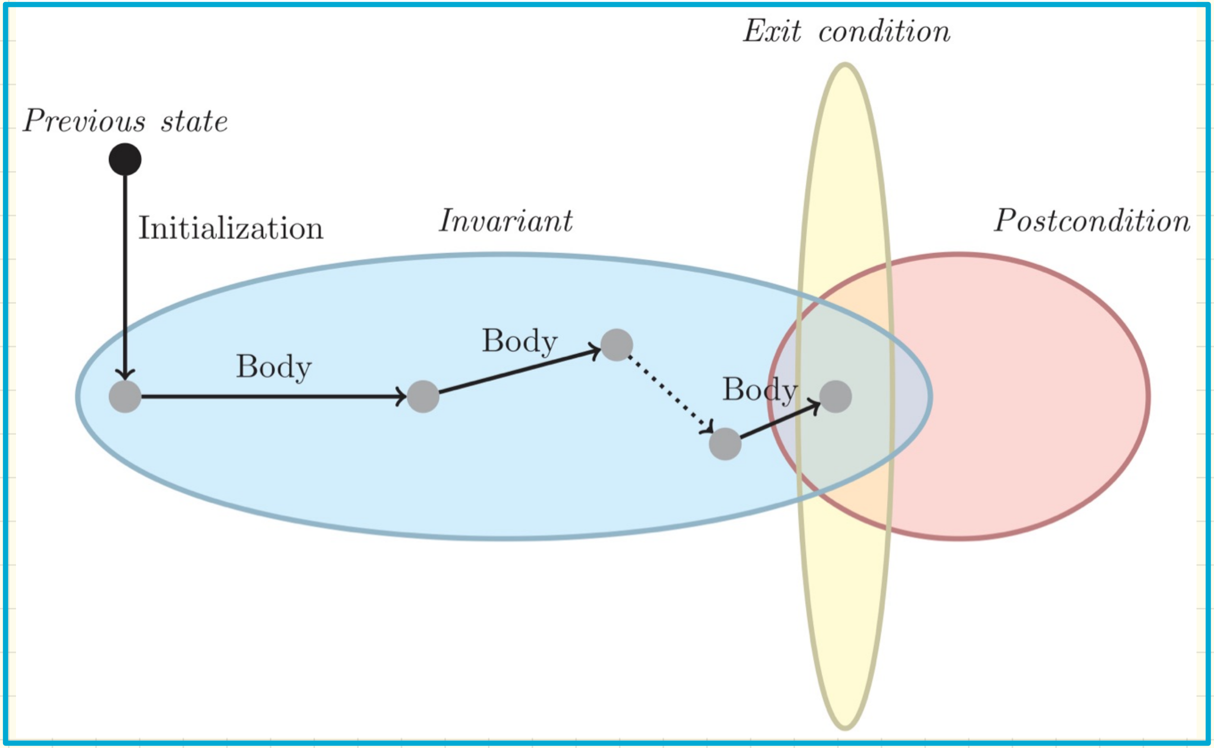


exit condition: i > 0

invariant: 1 <= i <= 5

variant: 5 - i

Contracts of Loops: Visualization



END OF COURSE

BEST OF LUCK !