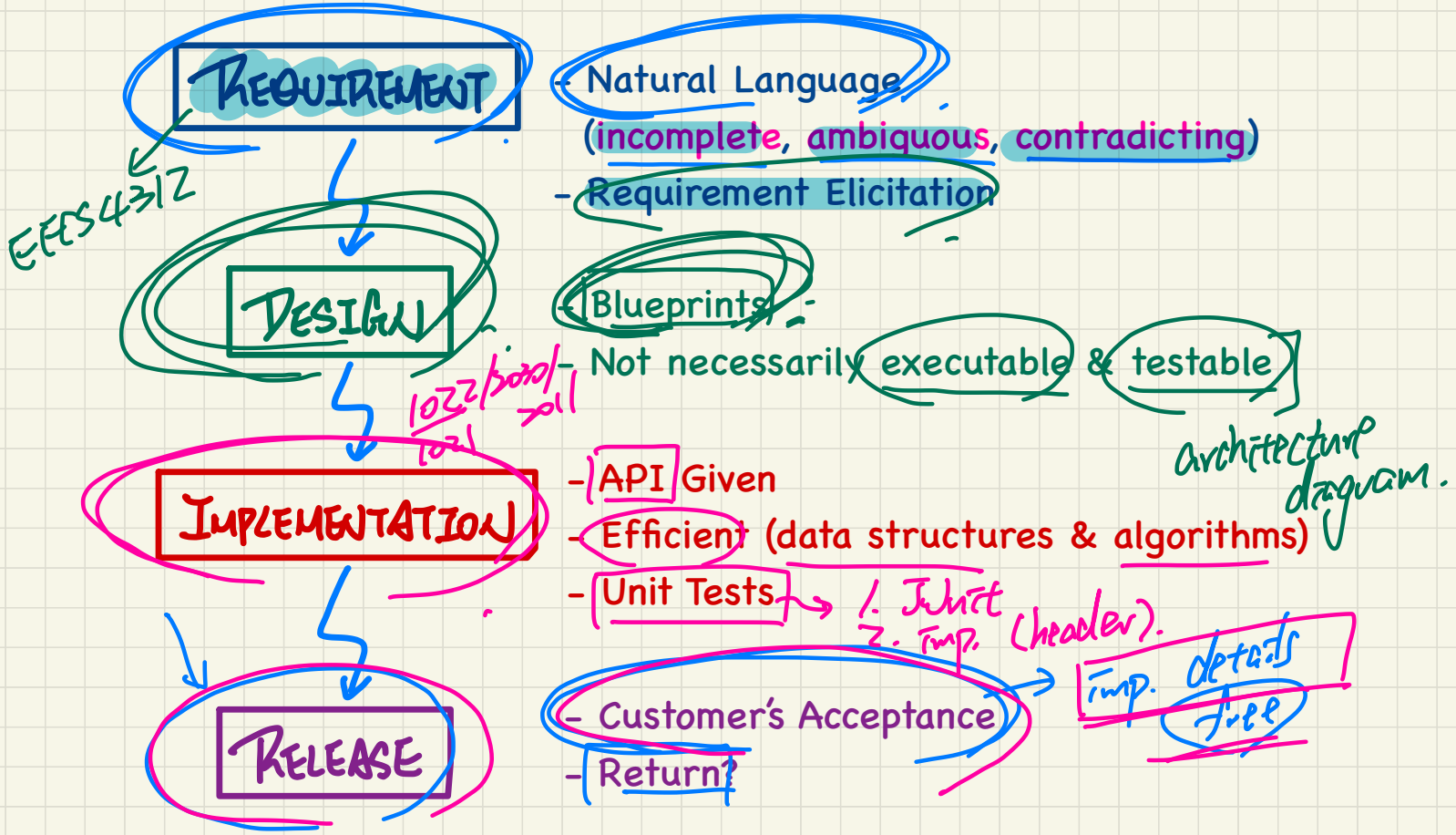
Lecture 1

Part 1

Design by Contract (DbC): Motivation & Terminology

Software Development Process



Informal Requirements

Incompleteness, Ambiguities, Contradictions

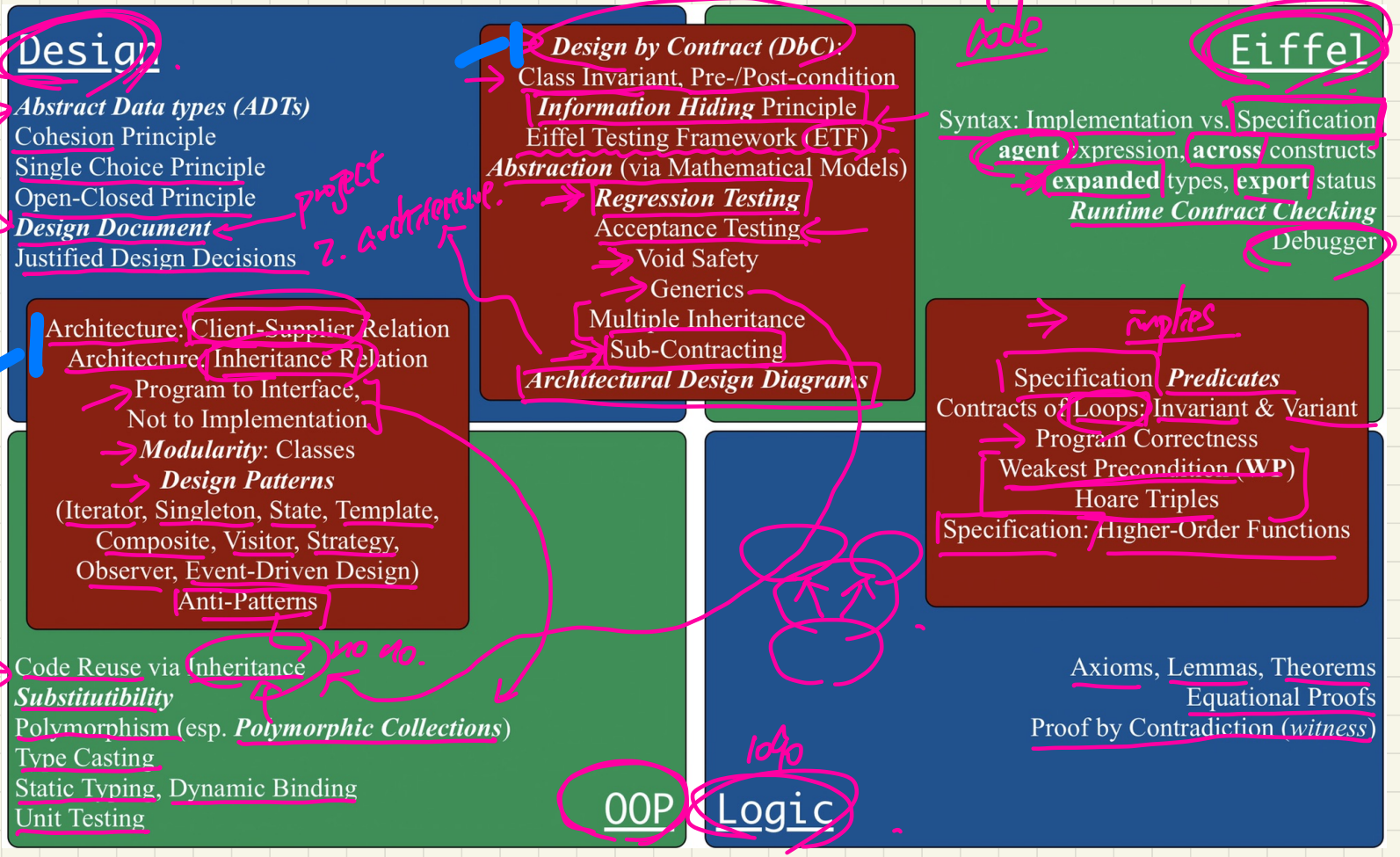
mobile web desktop.

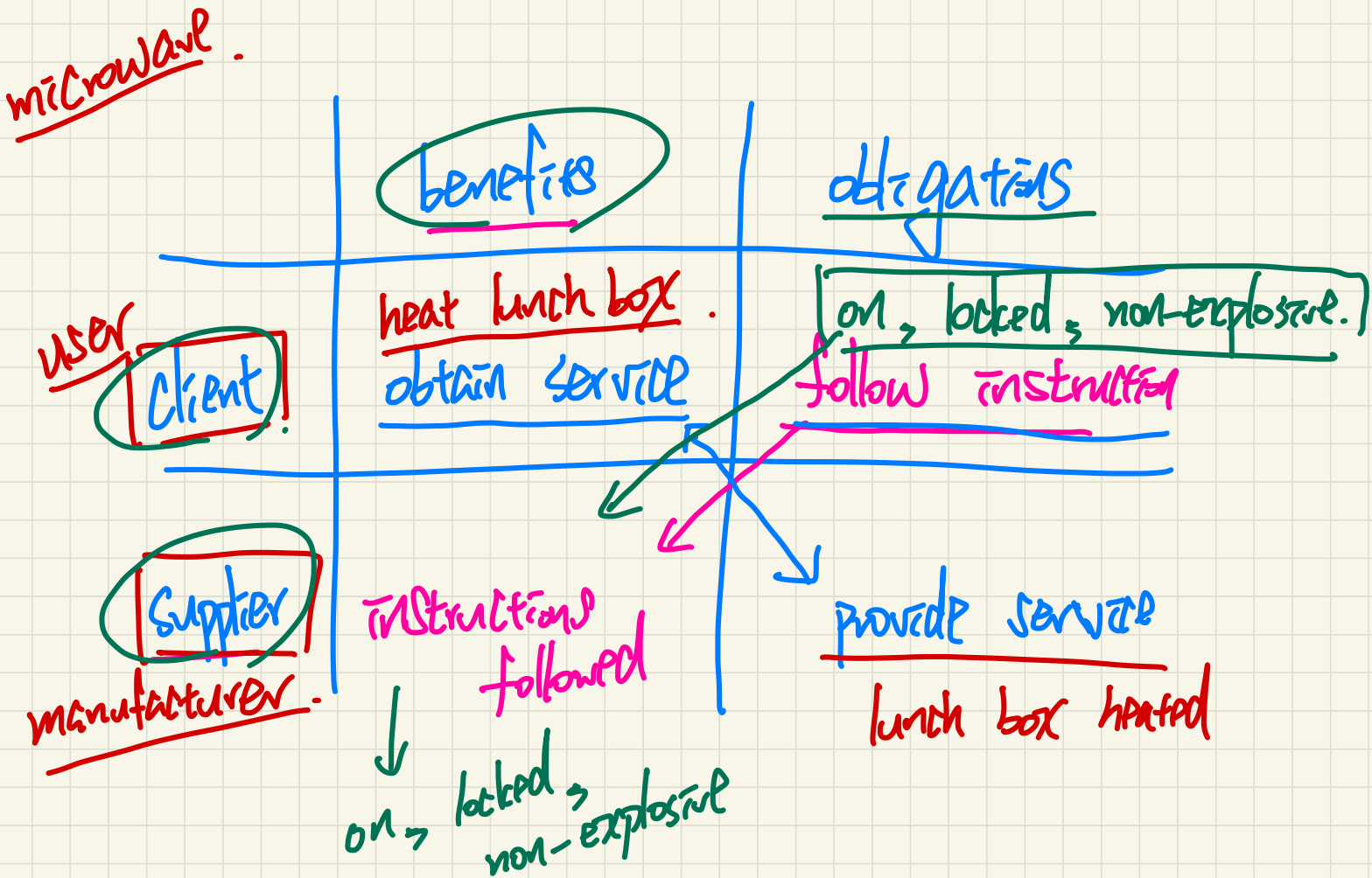
I'd like a working payment system.
Ideally, user can use it to pay their electricity bills and so on.
It should be easy to use and secure with a 4-phased authentication (face, touch, verification code, password).

P ¬P

$P \wedge \neg P \equiv F$.

Roadmap of this Course




microwave.
benefits
obligations
on, locked, non-explosive.
heat lunch box.
obtain service
follow instructions
user
client
supplier
manufacturer.
instructions followed
provide service
lunch box heated
on, locked, non-explosive

Client vs. Supplier in OOP

```
class Microwave {
  private boolean on;
  private boolean locked;
  void power() {on = true;}
  void lock() {locked = true;}
  void heat(Object stuff) {
    /* Assume: on && locked */
    /* stuff not explosive. */
  } }
```

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

client class

client method

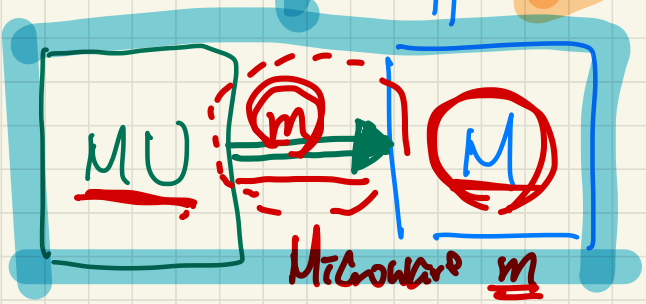
argument/input

context object

Microwave

supplier method

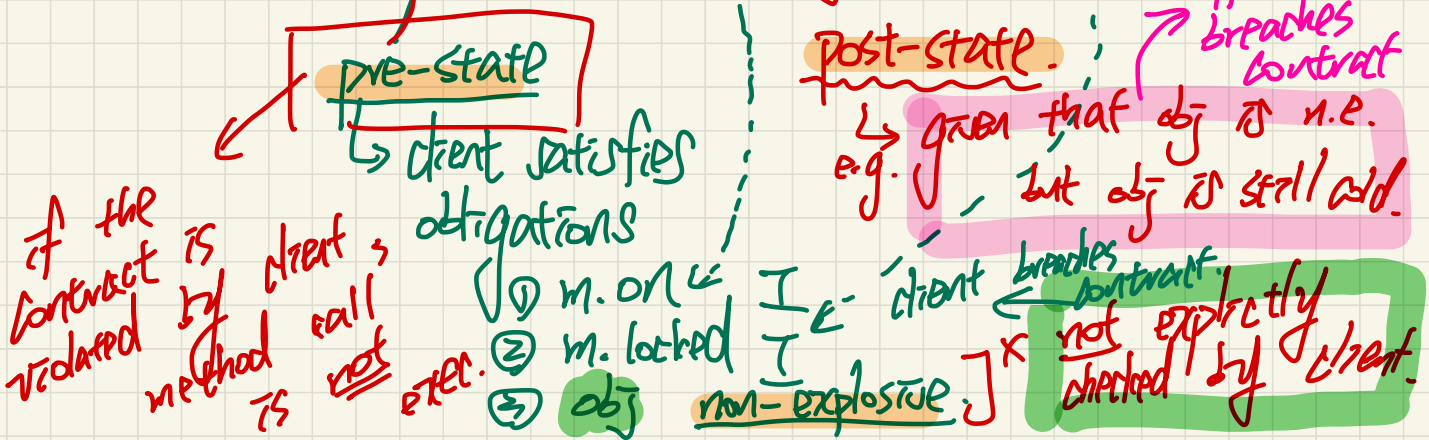
supplier class

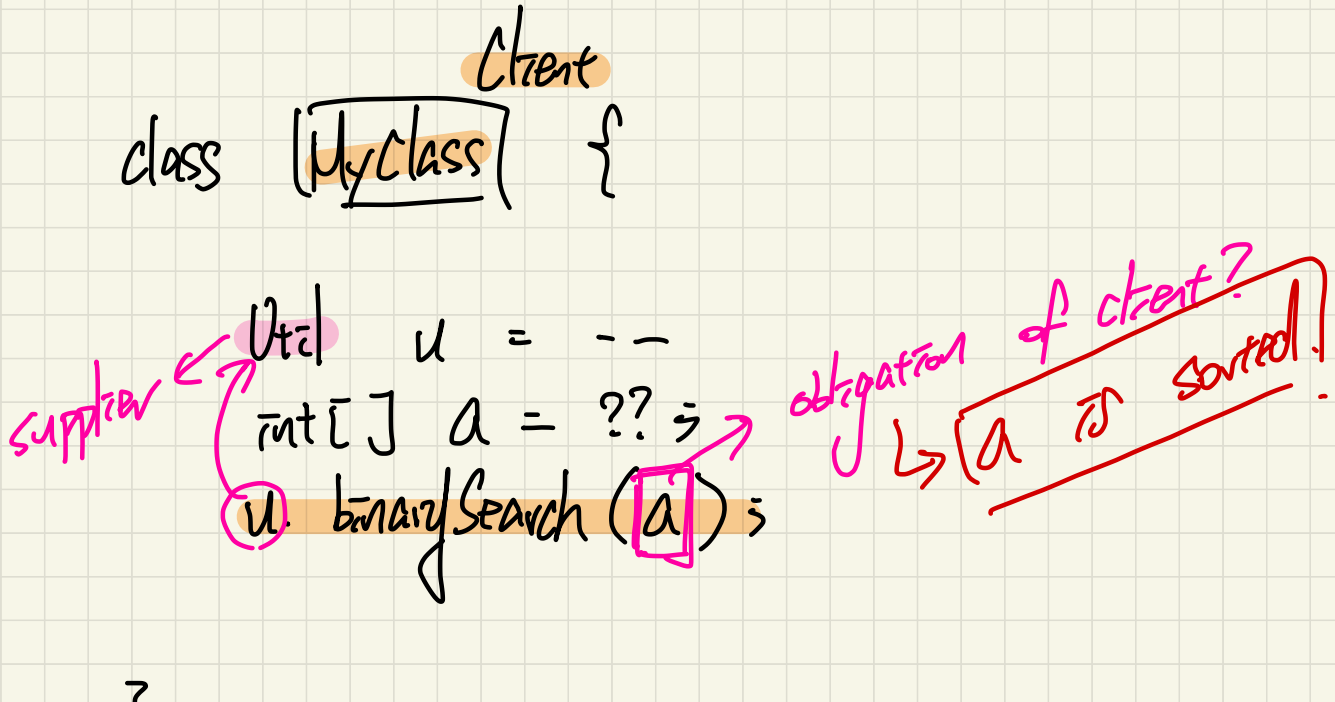


Contract Honoured?

```
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);
  } }
```




Client
class MyClass {
Util u = --
int[] a = ??;
u.binarySearch(a);
}
supplier
obligation of client?
↳ a is sorted!

Lecture 1

Part 2

Supporting DbC in Java:
Preconditions,
Class Invariant,
Postconditions

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. > 0

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

Bank Accounts in Java: Version 1

Supplier

```
1  public class AccountV1 {
2        private String owner;
3        private int balance;
4        public String getOwner() { return owner; }
5        public int getBalance() { return balance; }
6        public AccountV1(String owner, int balance) {
7              this.owner = owner; this.balance = balance;
8        }
9        public void withdraw(int amount) {
10             this.balance = this.balance - amount;
11       }
12       public String toString() {
13             return owner + "'s current balance is: " + balance;
14       }
15 }
```

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;
    }
}
```

supplier to blame
: interface doesn't specify input constraint

client to blame
: -10 is illegal.

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);
  }
}
```

post-state invalid

Console Output:

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

Bank Accounts in Java: Version 1 Critique (2)

```
1  public class AccountV1 {
2        private String owner;
3        private int balance;
4        public String getOwner() { return owner; }
5        public int getBalance() { return balance; }
6        public AccountV1(String owner, int balance) {
7              this.owner = owner; this.balance = balance;
8        }
9        public void withdraw(int amount) {
10             this.balance = this.balance - amount;
11       }
12       public String toString() {
13             return owner + "'s current balance is: " + balance;
14       }
15 }
```

supplier blame.

Supplier

Client

```
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);
```

invalid.

supplier.

```
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
```

client blame

Bank Accounts in Java: Version 1 Critique (3)

```
1  public class AccountV1 {
2        private String owner;
3        private int balance;
4        public String getOwner() { return owner; }
5        public int getBalance() { return balance; }
6        public AccountV1(String owner, int balance) {
7              this.owner = owner; this.balance = balance;
8        }
9        public void withdraw(int amount) {
10             this.balance = this.balance - amount;
11       }
12       public String toString() {
13             return owner + "'s current balance is: " + balance;
14
15 }
```

Supplier

Client

```
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);
```

supplier

```
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
```

Preconditions vs. Exceptions

$y != 0 \equiv \neg(y == 0)$

pre. exception

Service Conditions

```
/**
 * @pre y != 0
 */
divide(int x, int y) {
    ...
}
```

Error Conditions

```
/**
 * @throws DBZE if y == 0
 */
divide(int x, int y) throws DBZE {
    if (y == 0) {
        throw new DBZE()
    }
    ...
}
```

→ ¬P

y == 0

Bank Accounts in Java: Version 2

```
1  public class AccountV2 {
2    public AccountV2(String owner, int balance) throws
3       BalanceNegativeException
4    {
5      if( balance < 0 ) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
11   if( amount < 0 ) { /* negated precondition */
12     throw new WithdrawAmountNegativeException(); }
13   else if( balance < amount ) { /* negated precondition */
14     throw new WithdrawAmountTooLargeException(); }
15   else { this.balance = this.balance - amount; }
16   }
```

fix: balance <= amount

AccountV2 acc = 100

acc.withdraw(100)

100 100 100 100

0

exception condition

And missing case?

missing case

what if balance == amount

precondition:

¬(balance < amount)

≡ balance >= amount

Bank Accounts in Java: Version 2 Critique (1)

Compared with Version 1

```
1  public class AccountV2 {
2    public AccountV2(String owner, int balance) throws
3       BalanceNegativeException
4    {
5     if( balance < 0 ) { /* negated precondition */
6       throw new BalanceNegativeException(); }
7     else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10      WithdrawAmountNegativeException, WithdrawAmountTooLargeException
11    if( amount < 0 ) { /* negated precondition */
12      throw new WithdrawAmountNegativeException(); }
13    else if ( balance < amount ) { /* negated precondition */
14      throw new WithdrawAmountTooLargeException(); }
15    else { this.balance = this.balance - amount; }
16   }
```

-10

-10

bypassed.

Supplier

Client

```
1  public class BankAppV2 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Alan with balance -10:");
4      try {
5        AccountV2 alan = new AccountV2("Alan", -10);
6        System.out.println(alan);
7      }
8      catch (BalanceNegativeException bne) {
9        System.out.println("Illegal negative account balance.");
10     }
```

imp. of AccountV2 not appr.

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

Bank Accounts in Java: Version 2 Critique (2)

Compared with Version 1

```
1  public class AccountV2 {
2   public AccountV2(String owner, int balance) throws
3      BalanceNegativeException
4    {
5     if( balance < 0 ) { /* negated precondition */
6       throw new BalanceNegativeException(); }
7     else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10     WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
11   if( amount < 0 ) { /* negated precondition */
12     throw new WithdrawAmountNegativeException(); }
13    else if ( balance < amount ) { /* negated precondition */
14      throw new WithdrawAmountTooLargeException(); }
15    else { this.balance = this.balance - amount; }
16   }
```

Supplier

Client

```
1  public class BankAppV2 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Mark with balance 100:");
4      try {
5        AccountV2 mark = new AccountV2("Mark", 100);
6        System.out.println(mark);
7        System.out.println("Withdraw -1000000 from Mark's account:");
8        mark.withdraw(-1000000);
9        System.out.println(mark);
10     }
11     catch (BalanceNegativeException bne) {
12       System.out.println("Illegal negative account balance.");
13     }
14     catch (WithdrawAmountNegativeException wane) {
15       System.out.println("Illegal negative withdraw amount.");
16     }
17     catch (WithdrawAmountTooLargeException wane) {
18       System.out.println("Illegal too large withdraw amount.");
19     }
```

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

by passed

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

```
1  public class AccountV2 {
2    public AccountV2(String owner, int balance) throws
3       BalanceNegativeException
4    {
5      if( balance < 0 ) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8    }
9    public void withdraw(int amount) throws
10      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
11     if( amount < 0 ) { /* negated precondition */
12       throw new WithdrawAmountNegativeException(); }
13     else if ( balance < amount ) { /* negated precondition */
14       throw new WithdrawAmountTooLargeException(); }
15     else { this.balance = this.balance - amount; }
16   }
```

Client

```
1  public class BankAppV2 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jim with balance 100:");
4      try {
5        AccountV2 jim = new AccountV2("Jim", 100);
6        System.out.println(jim);
7        System.out.println("Withdraw 100 from Jim's account:");
8        jim.withdraw(100);
9        System.out.println(jim);
10     }
11     catch (BalanceNegativeException bne) {
12       System.out.println("Illegal negative account balance.");
13     }
14     catch (WithdrawAmountNegativeException wane) {
15       System.out.println("Illegal negative withdraw amount.");
16     }
17     catch (WithdrawAmountTooLargeException wane) {
18       System.out.println("Illegal too large withdraw amount.");
19     }
```

Requirement

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

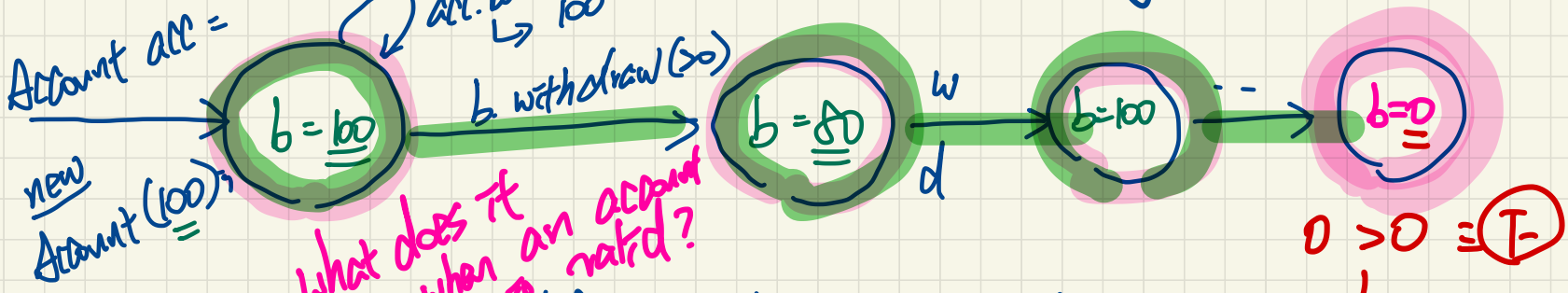
valid

circle

state : values of attributes

transitions: mutator method call which intend to modify state

arrows



Invariant → what does it mean when an account is valid?

acc.balance >0

lifetime of account object

Bank Accounts in Java: Version 3

```
1  public class AccountV3 {
2    public AccountV3(String owner, int balance) throws
3       BalanceNegativeException
4    {
5      if(balance < 0) { /* negated precondition */
6        throw new BalanceNegativeException(); }
7      else { this.owner = owner; this.balance = balance; }
8    assert this.getBalance() > X : "Invariant:  positive balance";
9    }
10   public void withdraw(int amount) throws
11      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
12     if(amount < 0) { /* negated precondition */
13       throw new WithdrawAmountNegativeException(); }
14     else if (balance < amount) { /* negated precondition */
15       throw new WithdrawAmountTooLargeException(); }
16     else { this.balance = this.balance - amount; }
17   assert this.getBalance() > X : "Invariant:  positive balance";
18   }
```

R1

100

R2

100

Change: $b > 100$ → multiple places affected (single choice principle).

Bank Accounts in Java: Version 3 Critique (1)

Compared with Version 2

```
1  public class AccountV3 {
2    public AccountV3(String owner, int balance) throws
3       BalanceNegativeException
4    {
5     if(balance < 0) { /* negated precondition */
6       throw new BalanceNegativeException(); }
7     else { this.owner = owner; this.balance = balance; }
8     assert this.getBalance() > 0 : "Invariant:  positive balance";
9    }
10   public void withdraw(int amount) throws
11      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
12    if(amount < 0) { /* negated precondition */
13      throw new WithdrawAmountNegativeException(); }
14    else if (balance < amount) { /* negated precondition */
15      throw new WithdrawAmountTooLargeException(); }
16    else { this.balance = this.balance - amount; }
17    assert this.getBalance() > 0 : "Invariant:  positive balance";
18   }
```

100 b=100 0 0 > 0 F.

Supplier

Client

```
1  public class BankAppV3 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jim with balance 100:");
4      try { AccountV3 jim = new AccountV3("Jim", 100);
5            System.out.println(jim);
6            System.out.println("Withdraw 100 from Jim's account:");
7            jim.withdraw(100); b=0
8            System.out.println(jim); }
9            /* catch statements same as this previous slide:
10            * 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)

```
1  public class AccountV3 {
2    public void withdraw(int amount) throws
3       WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
4      if(amount < 0) { /* negated precondition */
5        throw new WithdrawAmountNegativeException(); }
6      else if (balance < amount) { /* negated precondition */
7        throw new WithdrawAmountTooLargeException(); }
8      else { this.balance = this.balance - amount; }
9      assert this.getBalance() > 0 : "Invariant: positive balance"; }
```

≈ preconditions (obligations for clients)

≈ invariant (obligation for supplier)

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

```
1  public class AccountV4 {
2    public void withdraw(int amount) throws
3      WithdrawAmountNegativeException, WithdrawAmountTooLargeException
4    { if(amount < 0) { /* negated precondition */
5        throw new WithdrawAmountNegativeException(); }
6      else if (balance < amount) { /* negated precondition */
7        throw new WithdrawAmountTooLargeException(); }
8      else { /* WRONT IMPLEMENTATION */
9        this.balance = this.balance + amount; }
10   assert this.getBalance() > 0 :
11     owner + "Invariant: positive balance"; }
```

faulty

Invariant : is this contract "good" enough to signal an error at runtime?

Bank Accounts in Java: Version 4 Critique

```
1  public class AccountV4 {
2    public void withdraw(int amount) throws
3      WithdrawAmountNegativeException, WithdrawAmountTooLargeException
4    {if(amount < 0) { /* negated precondition */
5      throw new WithdrawAmountNegativeException(); }
6     else if (balance < amount) { /* negated precondition */
7      throw new WithdrawAmountTooLargeException(); }
8      else { /* CORRECT IMPLEMENTATION */
9        this.balance = this.balance + amount; }
10   assert this.getBalance() >= 0 :
11     owner + "Invariant: positive balance"; }
```

Supplier

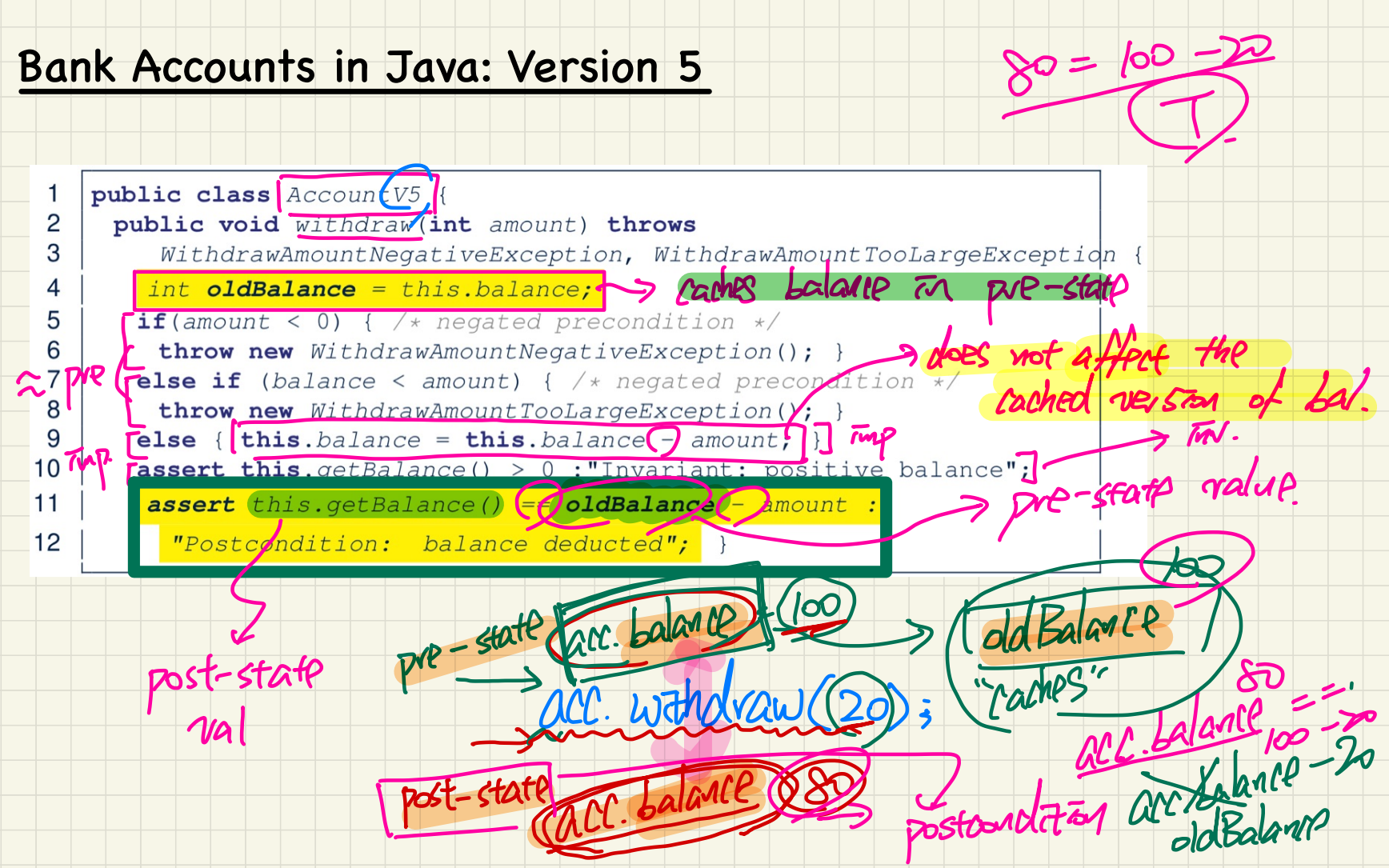
Client

```
1  public class BankAppV4 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jeremy with balance 100:")
4      try { AccountV4 jeremy = new AccountV4("Jeremy", 100);
5            System.out.println(jeremy);
6            System.out.println("Withdraw 50 from Jeremy's account:");
7            jeremy.withdraw(50);
8            System.out.println(jeremy); }
9      /* catch statements same as this previous slide:
10      * 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

```
1  public class AccountV5 {
2    public void withdraw(int amount) throws
3        WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
4      int oldBalance = this.balance;
5    if(amount < 0) { /* negated precondition */
6      throw new WithdrawAmountNegativeException(); }
7    else if (balance < amount) { /* negated precondition */
8      throw new WithdrawAmountTooLargeException(); }
9    else { this.balance = this.balance - amount; }
10   assert this.getBalance() > 0 :"Invariant: positive balance";
11     assert this.getBalance() == oldBalance - amount :
12       "Postcondition:  balance deducted"; }
```



Bank Accounts in Java: Version 5 Critique

Compared with Version 4

Supplier

```
1  public class AccountV5 {
2    public void withdraw(int amount) throws
3        WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
4      int oldBalance = this.balance;
5      if(amount < 0) { /* negated precondition */
6        throw new WithdrawAmountNegativeException(); }
7      else if (balance < amount) { /* negated precondition */
8        throw new WithdrawAmountTooLargeException(); }
9      else { this.balance = this.balance + amount; }
10     assert this.getBalance() > 0 : "Invariant: positive balance";
11     assert (this.getBalance() == oldBalance - amount) :
12       "Postcondition: balance deducted"; }
```

Client

```
1  public class BankAppV5 {
2    public static void main(String[] args) {
3      System.out.println("Create an account for Jeremy with balance 100:")
4      try { AccountV5 jeremy = new AccountV5("Jeremy", 100);
5            System.out.println(jeremy);
6            System.out.println("Withdraw 50 from Jeremy's account:");
7            jeremy.withdraw(50);
8            System.out.println(jeremy); }
9            /* catch statements same as this previous slide:
10            * 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

```
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"; }
```

C.I. Private

Supplier

Client

```
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.");
  }
```

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
```

imp.

imp.

Implementation View

Lecture 1

Part 3

DbC in Eiffel: Runtime Contract Checking

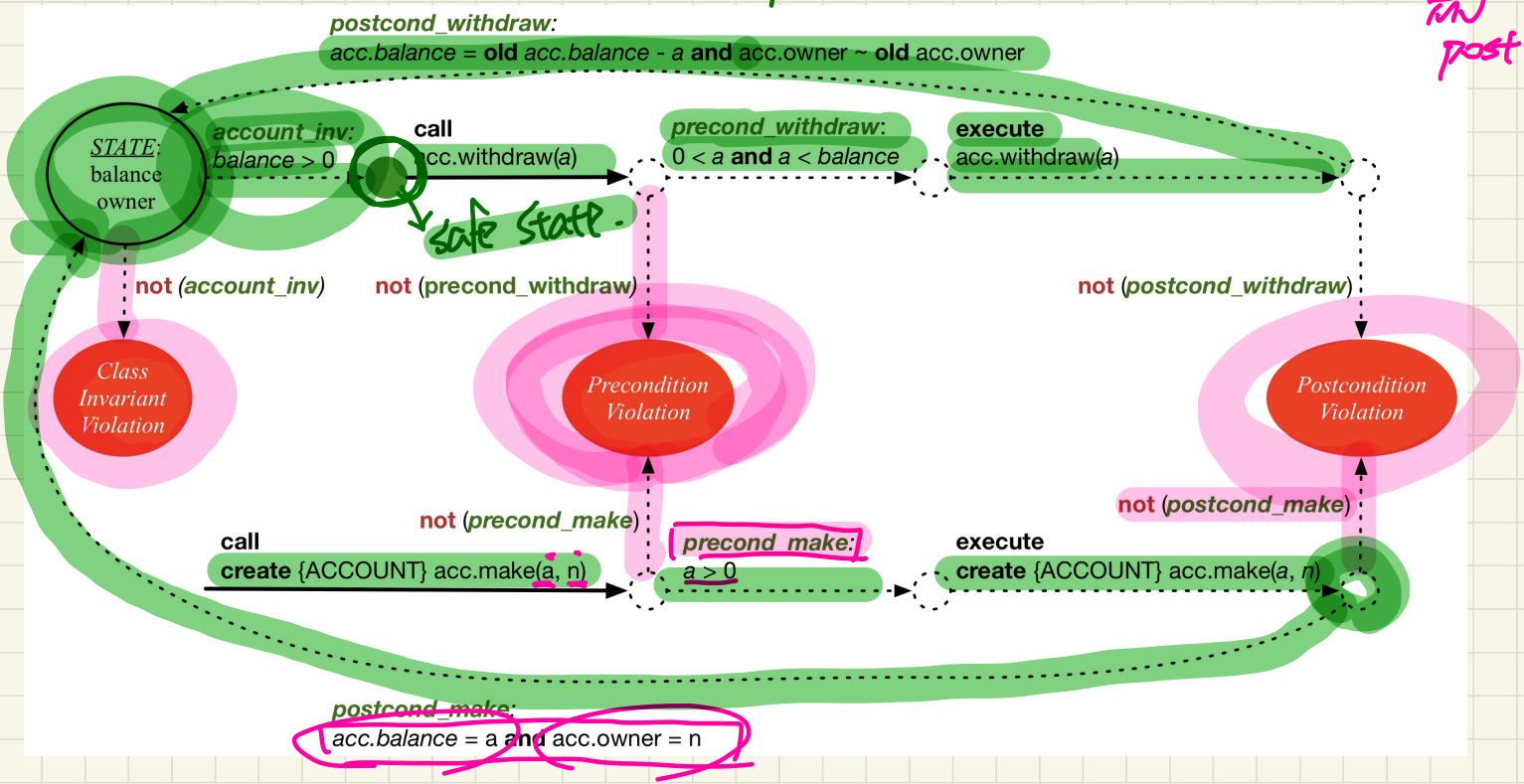
Design by Contract in Eiffel

Implementation View

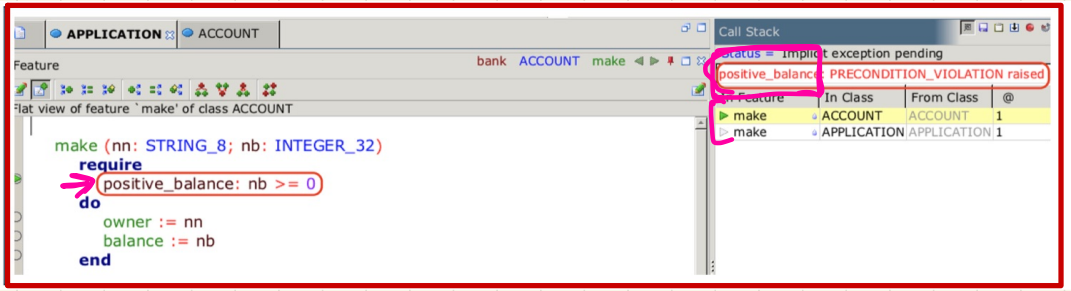
```
class ACCOUNT
create
      make
feature -- Attributes
      owner : STRING
      balance : INTEGER
feature -- Constructors
      make(nn: STRING; nb: INTEGER)
            require -- precondition
                  positive_balance: nb > 0
            do
                  owner := nn
                  balance := nb
            ensure
                  init: balance = nb and owner = nn
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
```

Runtime Monitoring of Contracts

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



Precondition Violation: positive_balance



Supplier

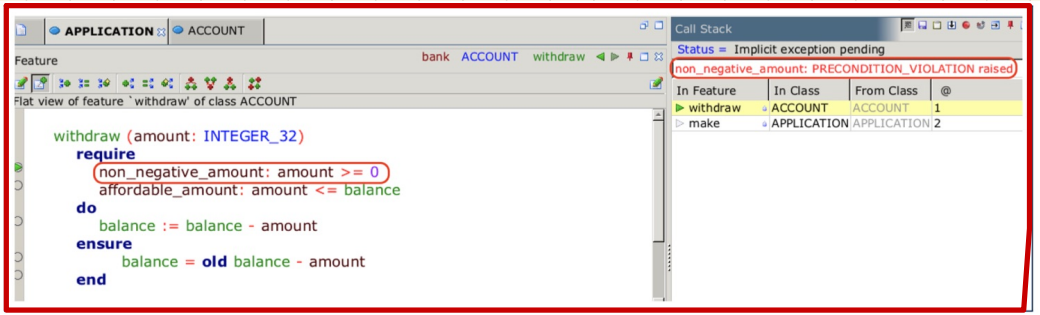
```
class ACCOUNT
create
      make
feature -- Attributes
      owner : STRING
      balance : INTEGER
feature -- Constructors
      make(nn: STRING; nb: INTEGER)
            require -- precondition
                  positive_balance: nb > 0
            end
feature -- Commands
      withdraw(amount: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

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

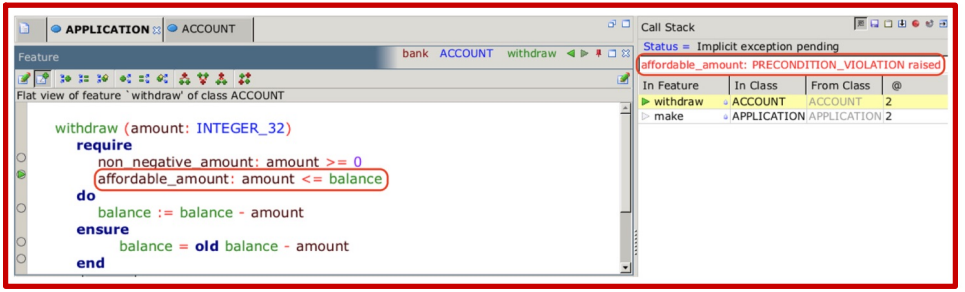
```
class ACCOUNT
create
	make
feature -- Attributes
	owner : STRING
	balance : INTEGER
feature -- Constructors
	make(nn: STRING; nb: INTEGER)
		require -- precondition
			positive_balance: nb > 0
		end
feature -- Commands
	withdraw(amount: INTEGER)
		require -- precondition
			non_negative_amount: amount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



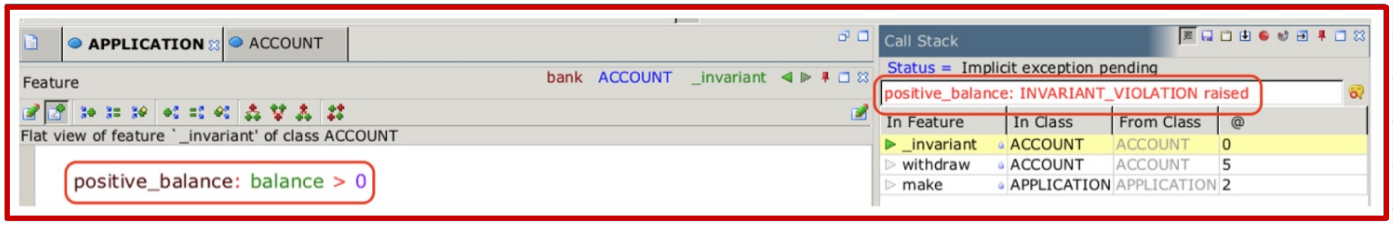
Supplier

```
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



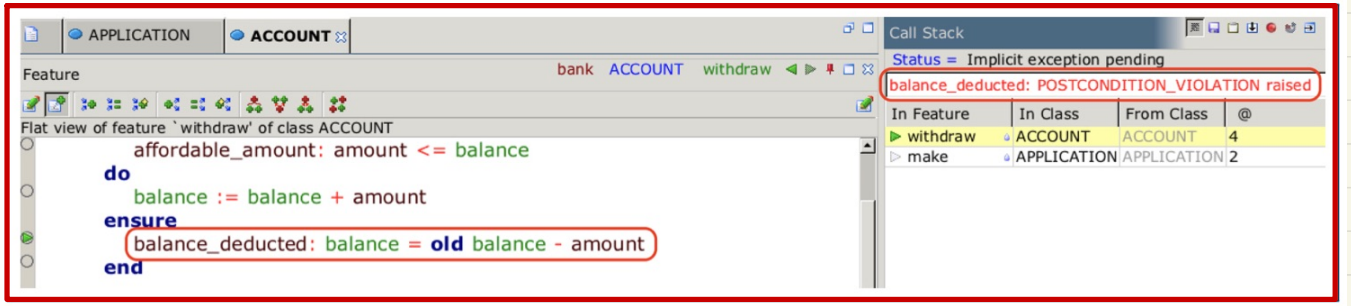
Supplier

```
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



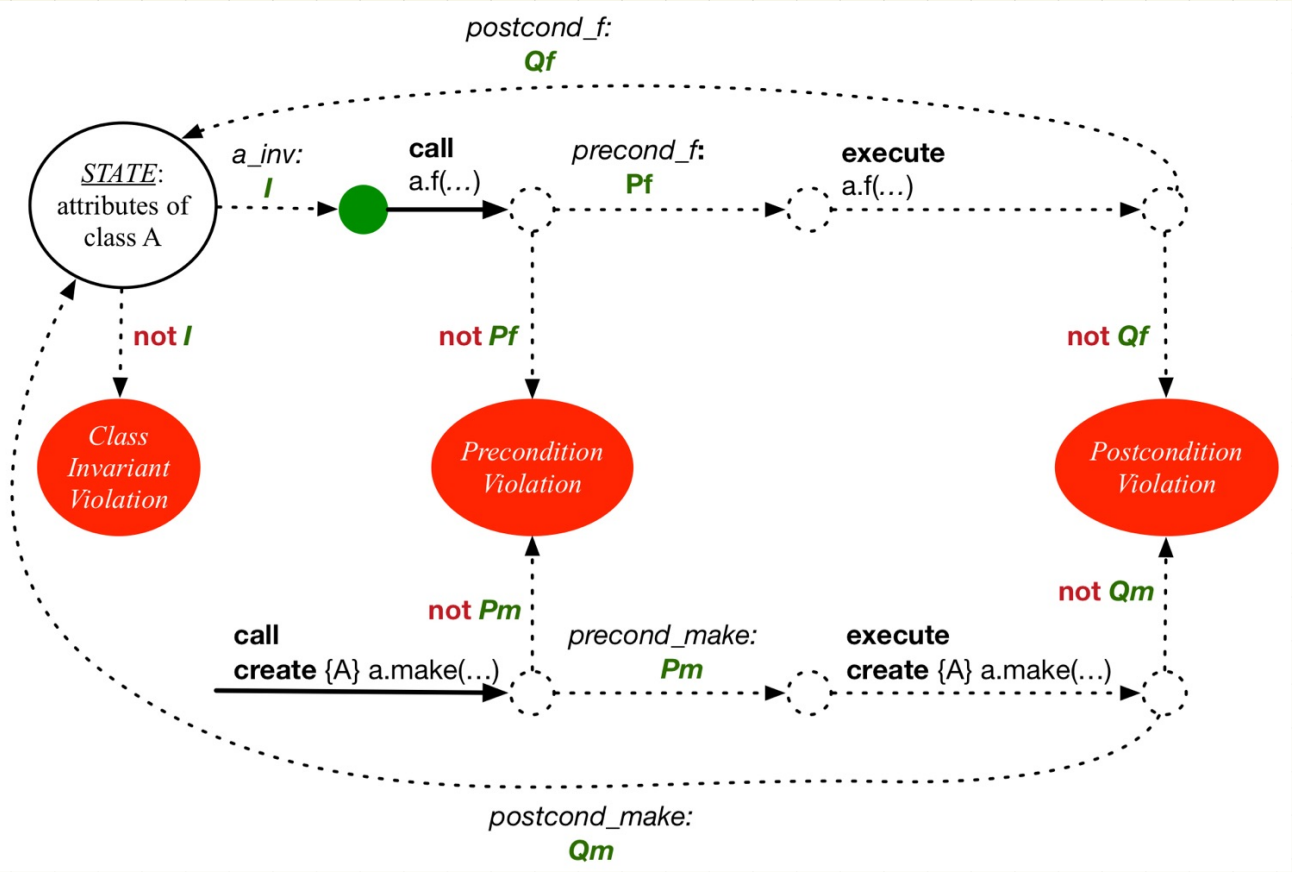
Supplier

```
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



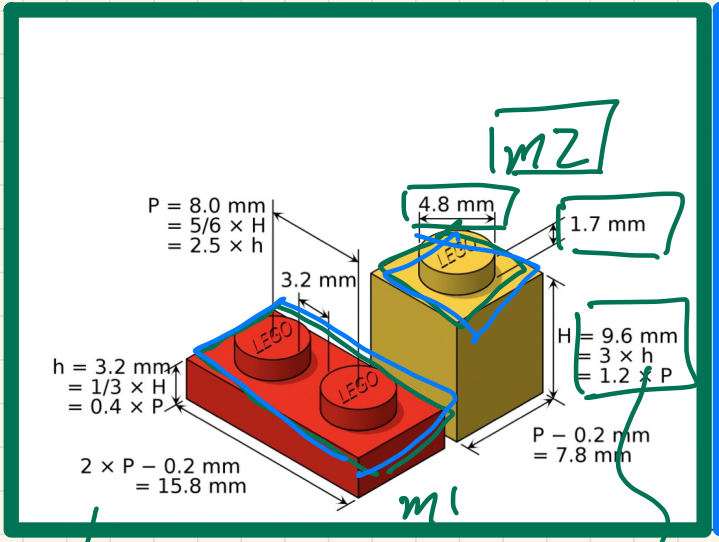
Lecture 2

Part 1

Modularity & Modular Design
Abstract Data Types (ADTs)

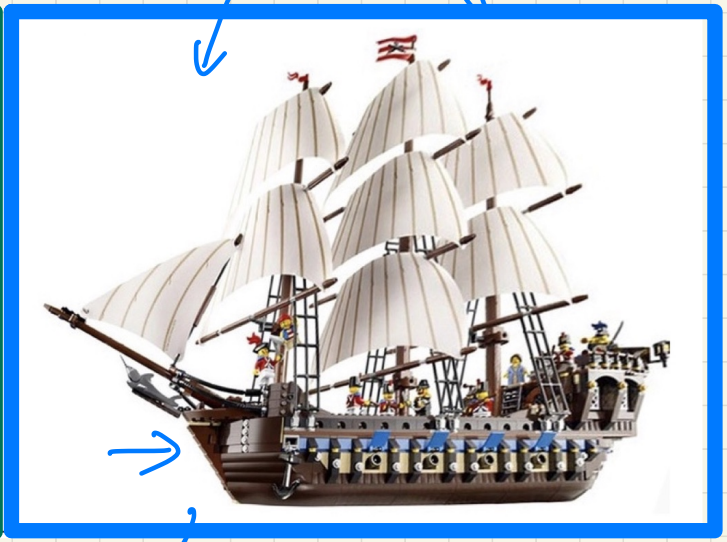
Modularity: Childhood Activities

assembly of modules.



m2

m1

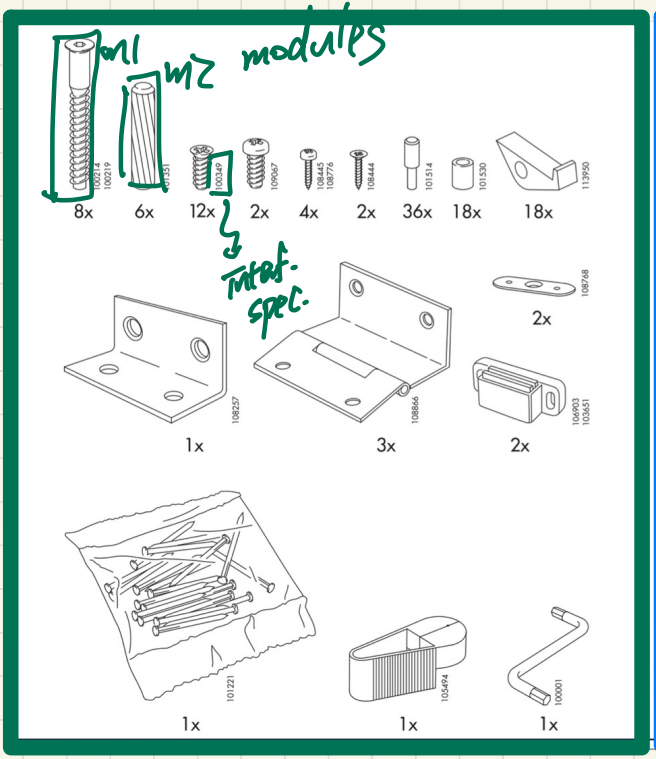


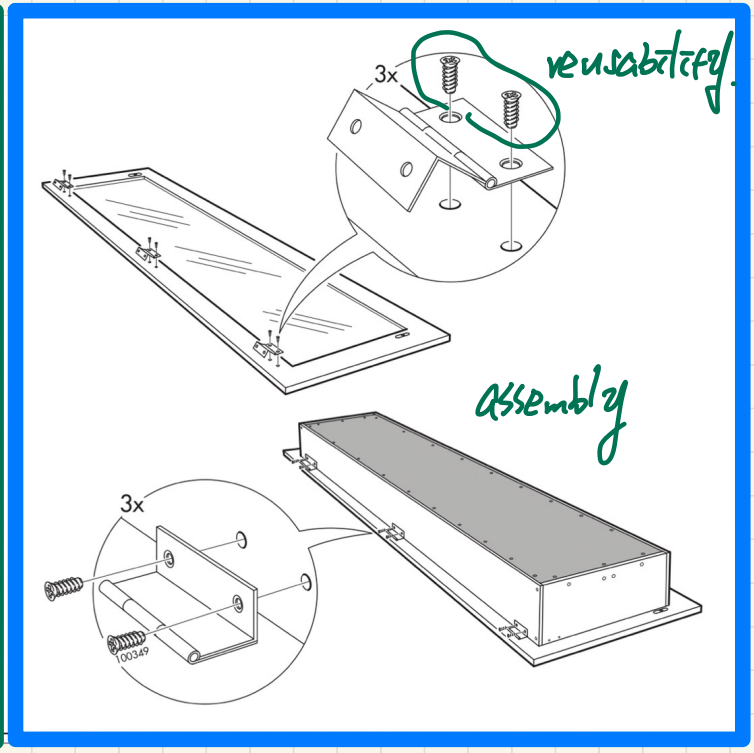
modules

interface specification

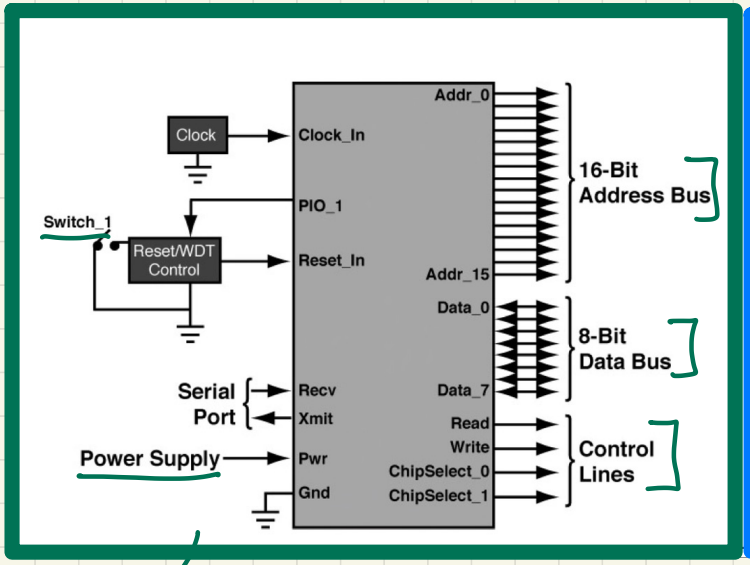
reusable

Modularity: Daily Constructions

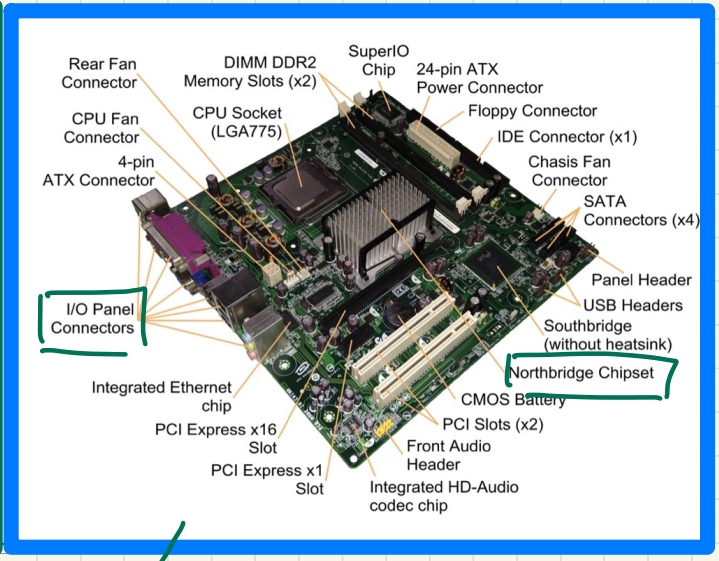




Modularity: Computer Architectures

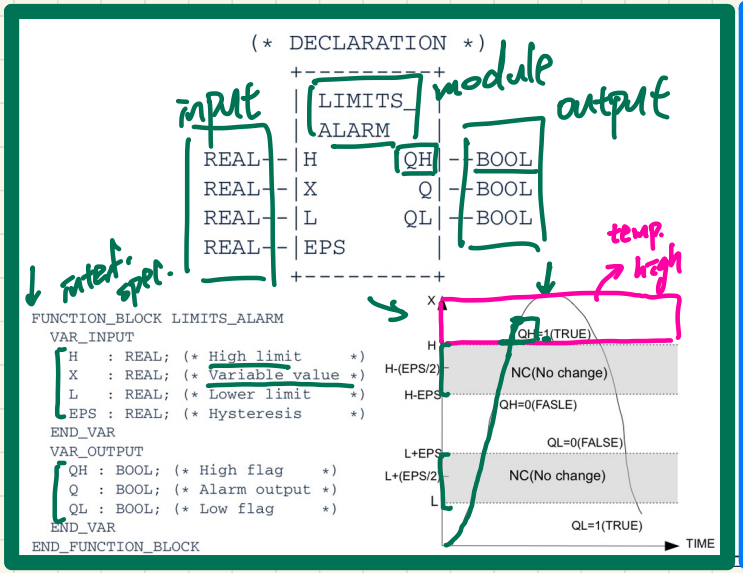


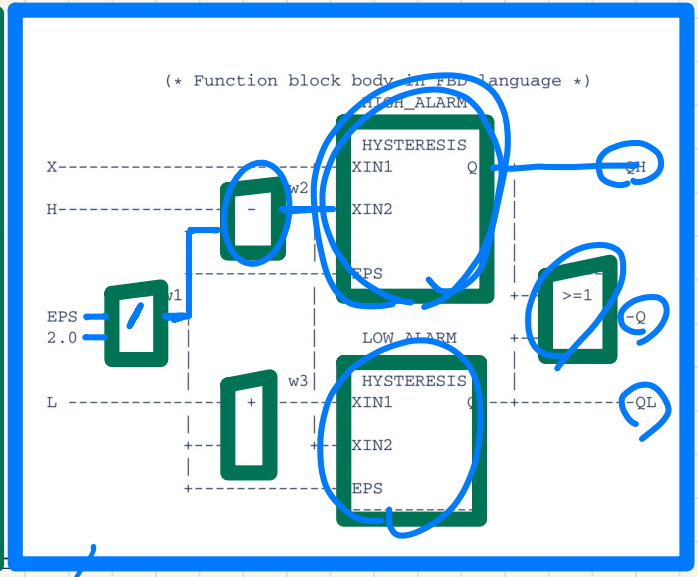
spec.



assembly.

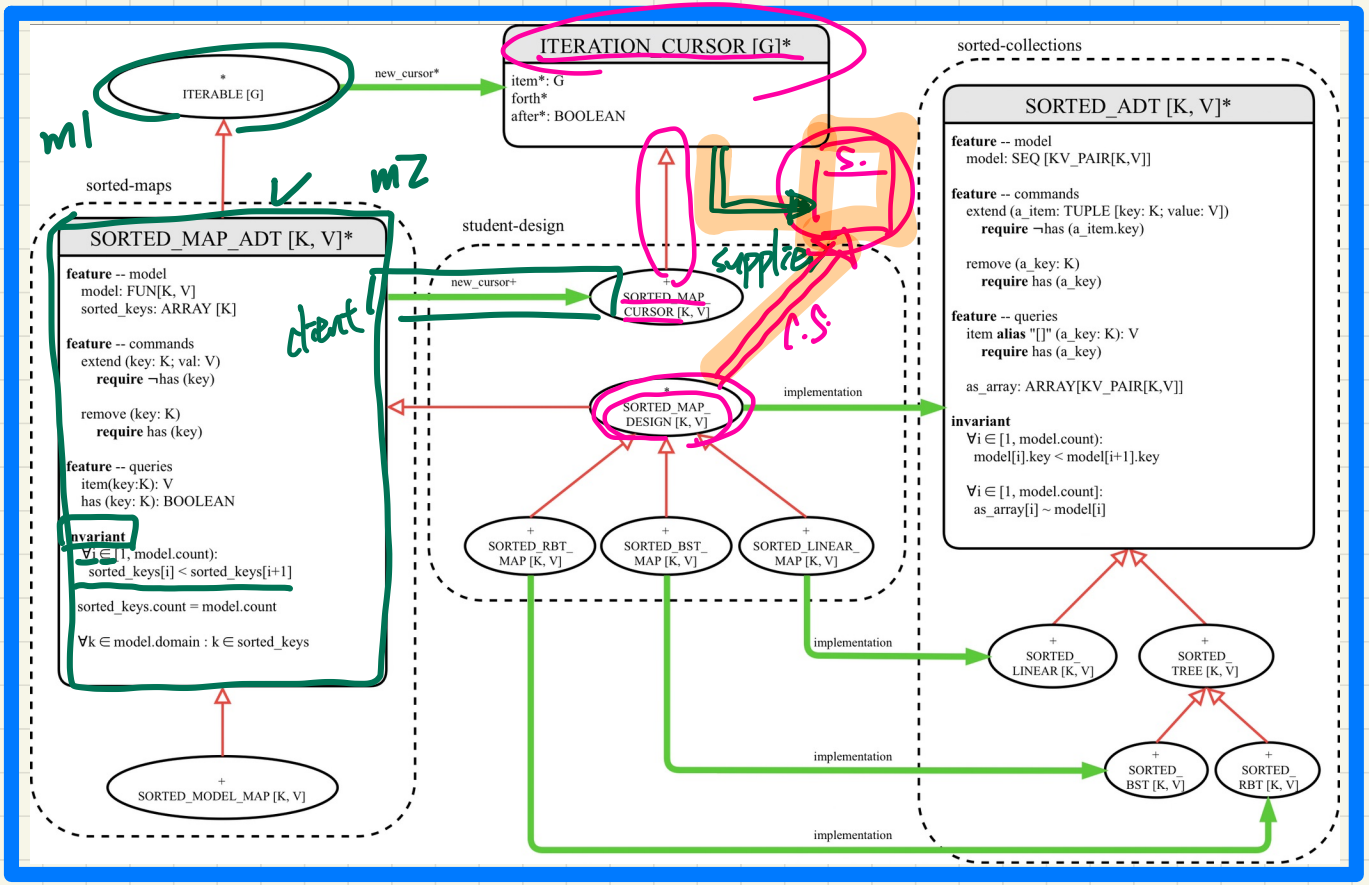
Modularity: System Developments





assembly.

Modularity: Software Design



Java Classes: Abstract Data Types? No.

E	set(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).

set

```
E set(int index,
      E element)
```

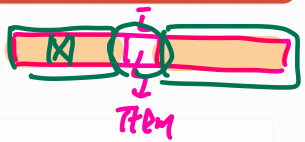
set(i, item)

Replaces the element at the specified position in this list with the specified element (optional operation).

Parameters:

index - index of the element to replace

element - element to be stored at the specified position



Returns:

the element previously at the specified position

Throws:

UnsupportedOperationException - if the set operation is not supported by this list

ClassCastException - if the class of the specified element prevents it from being added to this list

NullPointerException - if the specified element is null and this list does not permit null elements

IllegalArgumentException - if some property of the specified element prevents it from being added to this list

IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())

≠ precond.

gen. param.

Interface List<E>

Type Parameters:

E - the type of elements in this list

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Implementing Classes:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

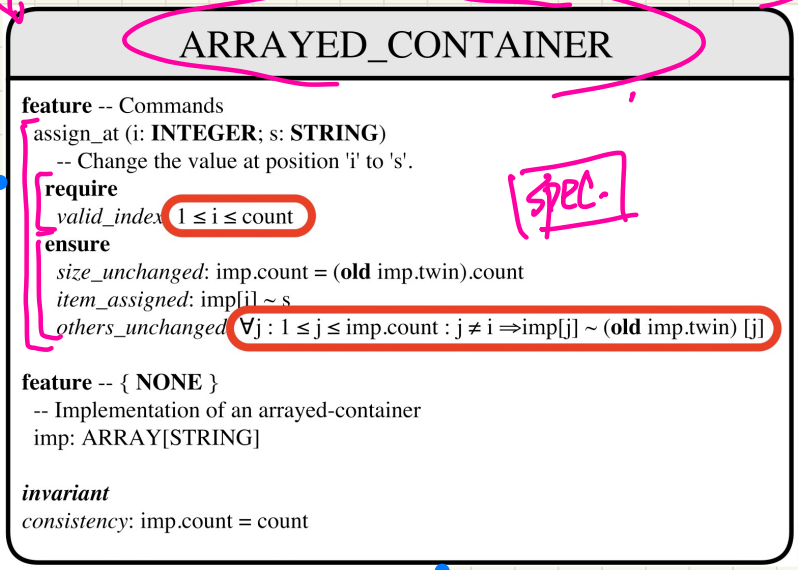
---

```
public interface List<E>
extends Collection<E>
```

An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Eiffel Classes: Abstract Data Types?

Design Diagram



spec.

Contract View

```
class interface ARRAYED_CONTAINER
feature -- Commands
  assign_at (i: INTEGER; s: STRING)
      -- Change the value at position 'i'
    require
      valid_index: 1 <= i and i <= count
    ensure
      size_unchanged:
        imp.count = (old imp.twin).count
      item_assigned:
        imp [i] ~ s
      others_unchanged:
        across
          1 |..| imp.count as j
        all
          j.item /= i implies imp [j.item] ~ (old imp.twin) [j.item]
        end
  count: INTEGER
invariant
  consistency: imp.count = count
end -- class ARRAYED_CONTAINER
```

∀

Lecture 2

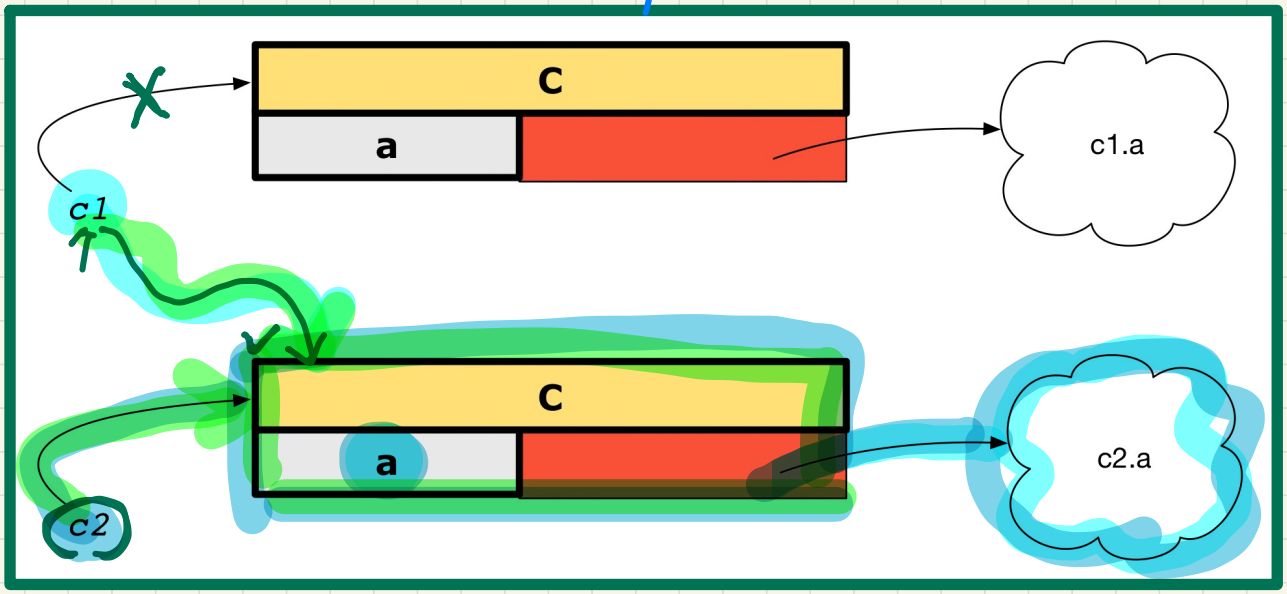
Part 2

Copying Objects:
Reference vs. Shallow vs. Deep

Reference Copy: c1 := c2

ref. copy

↳ cheapest.

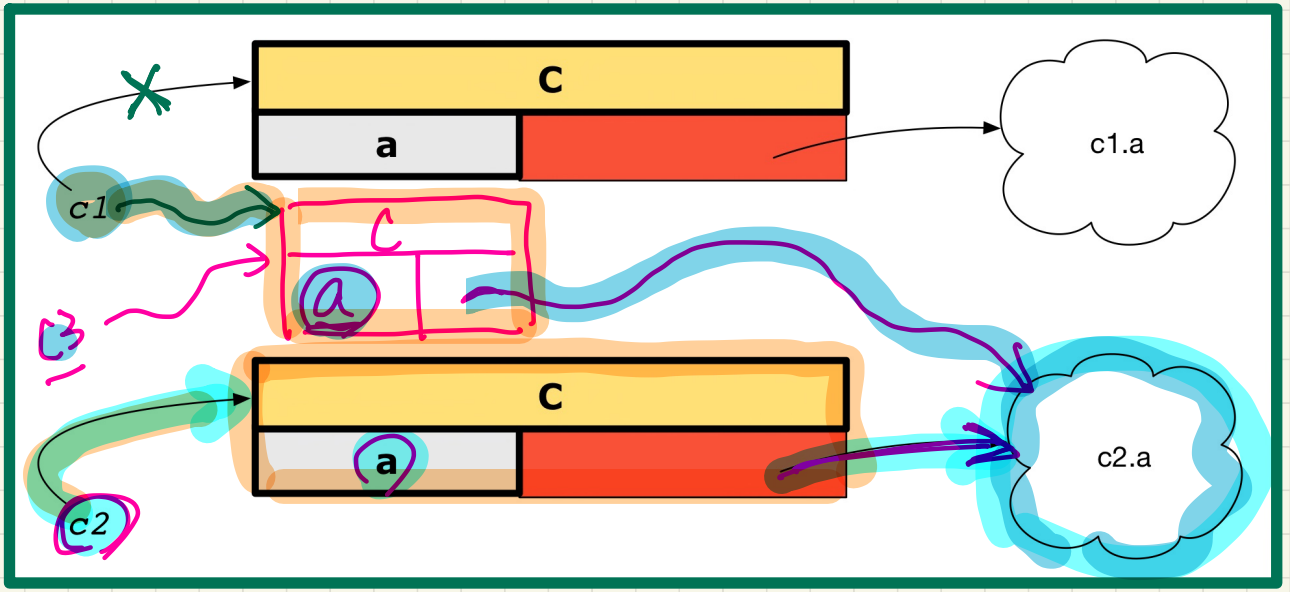


c1 = c2 ↳ equality ⊤.

c1.a = c2.a ⊤.

Shallow Copy: c1 := c2.twin

shallow copy

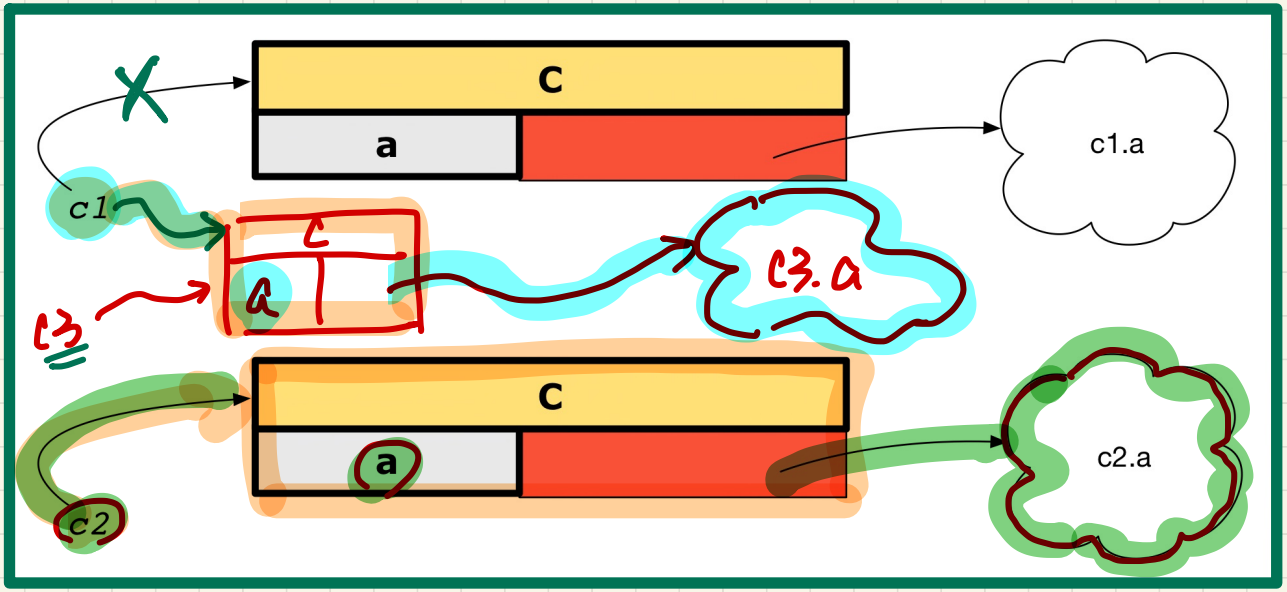


c3.a := c2.a

- c1 = c2 F

c1.a = c2.a T

Deep Copy: c1 := c2.deep_twin deep copy.



c3.a := c2.a.deep_twin

c1 = c2 F.

c1.a = c2.a F

Reference vs. Shallow vs. Deep Copies

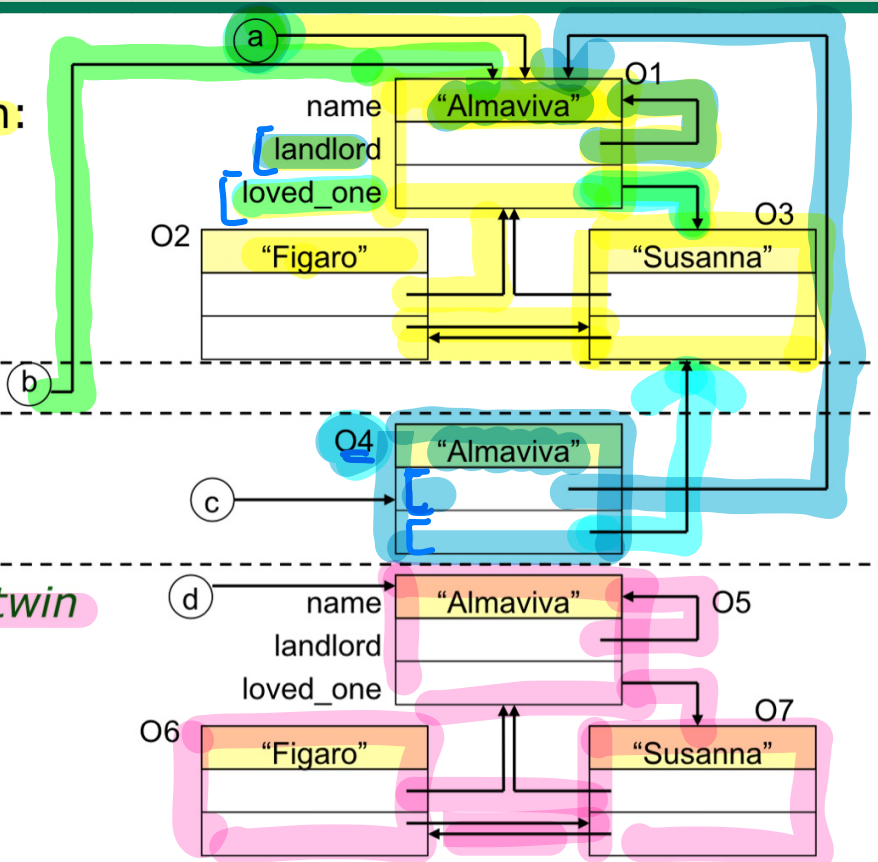
- Initial situation:

- Result of:

b := a

c := a.twin

d := a.deep_twin

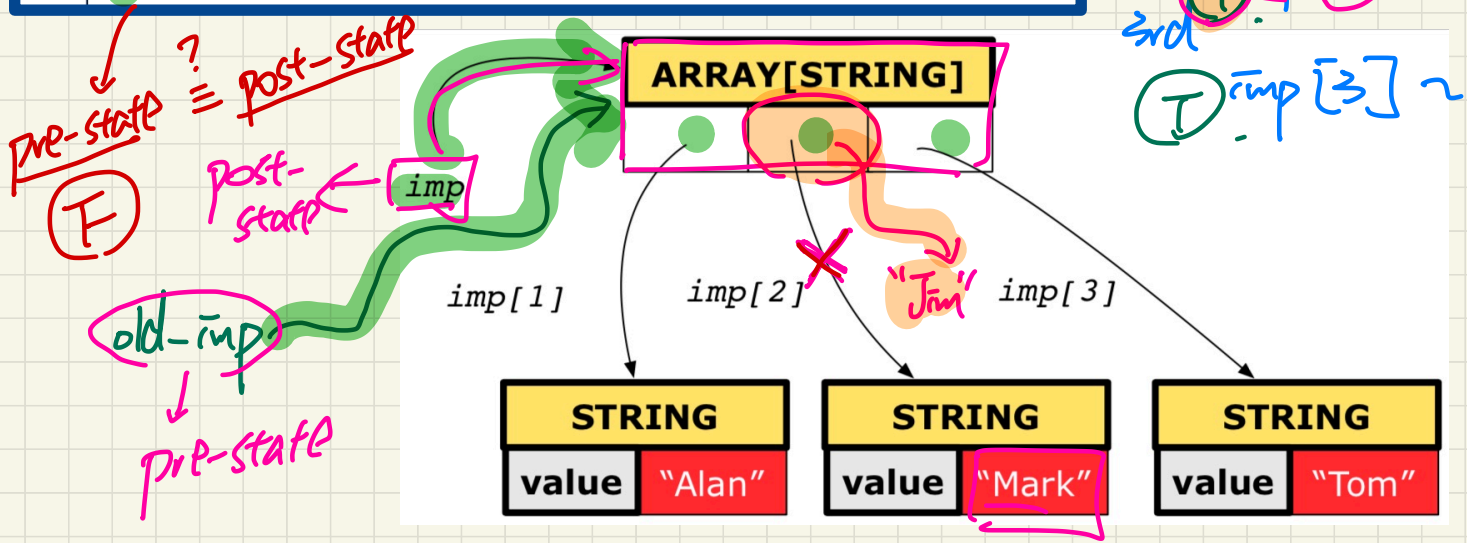


o4.ld := a.ld

o4.lo := a.lo

Collection Objects: Reference Copy & Make Changes

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



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

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

shallow

F → pre-state ≠ post-state

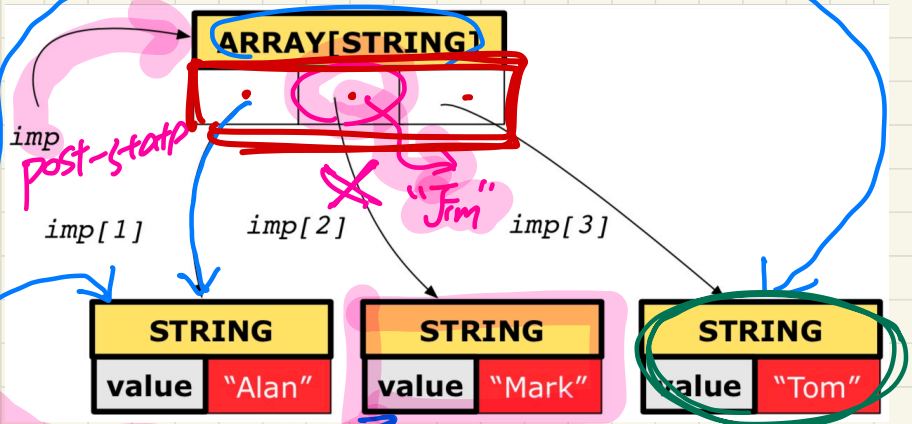
1st T appropriate to see there's a change
imp[1] ~ old_imp[1]

2nd F imp[2] ~ old_imp[2]
post-state pre-state

3rd T imp[3] ~ old_imp[3]

old_imp[1] := imp[1]
2 2
3 3

old_imp
pre-state.



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

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

T

pre-state "=" post-state ?

T

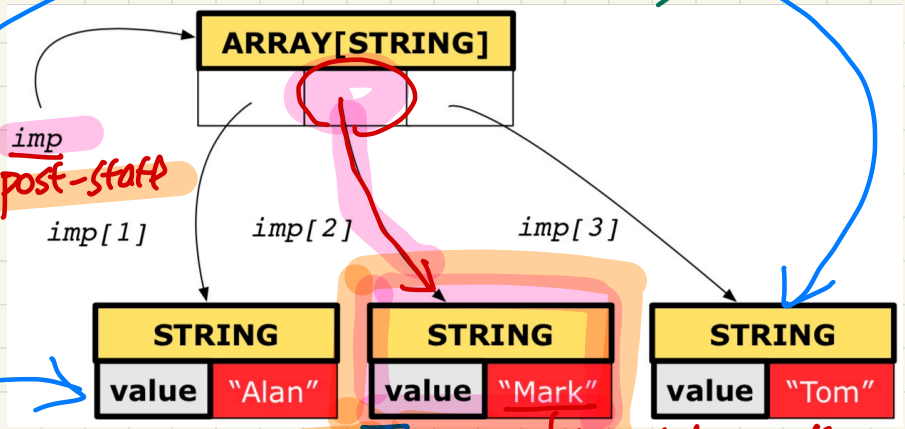
imp[1] ~ old_imp[1]

T

imp[2] ~ old_imp[2]

T

imp[3] ~ old_imp[3]



(pre-state)

old_imp

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

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

pre-state "1" post-state → F

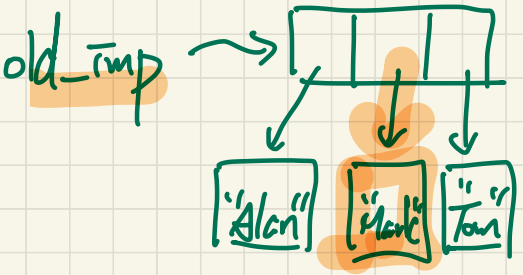
appropriate

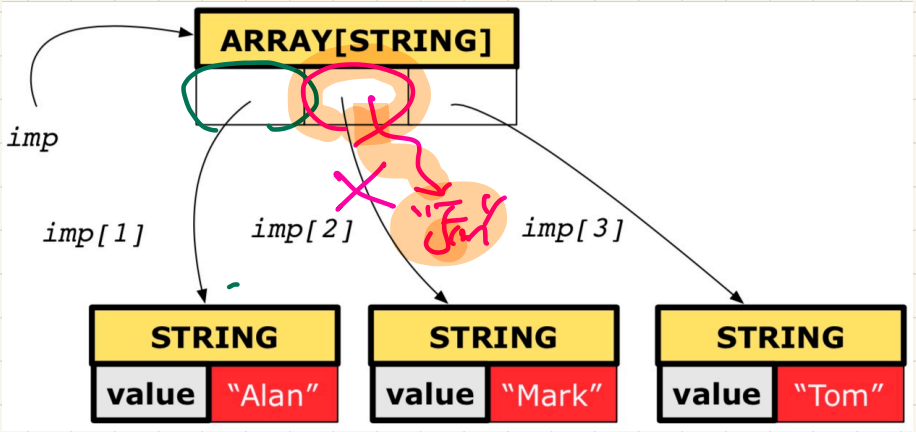
1st T
imp[1] ~ old_imp[1]

2nd F
imp[2] ~ old_imp[2]

3rd
imp[3] ~ old_imp[3]
T

old_imp := imp.deep_twin





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

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

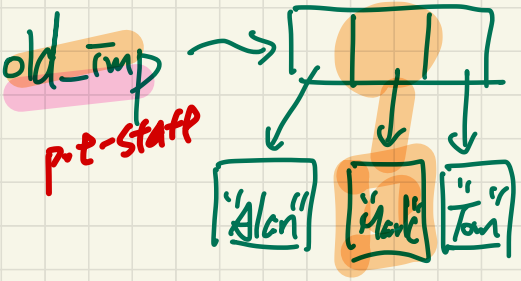
approp. tag

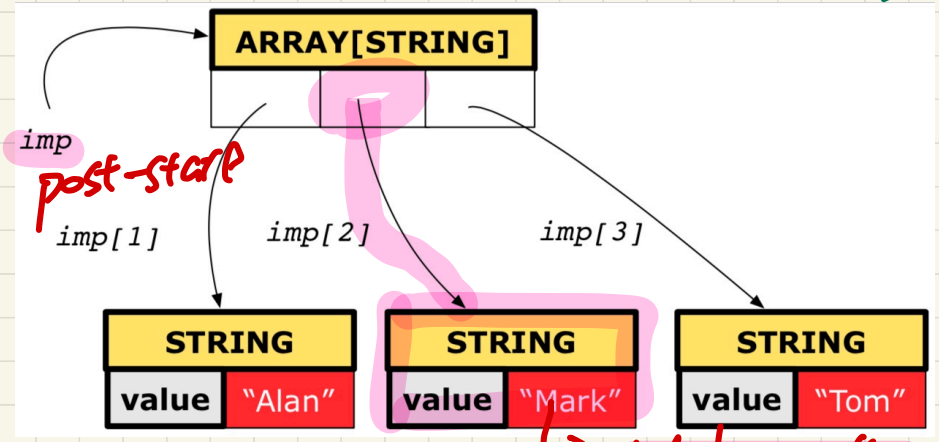
1st T: imp[1] ~ old_i[1]

2nd F: imp[2] ~ old_i[2]

3rd T: imp[3] ~ old_i[3]

post-stamp F → post-stamp "1"

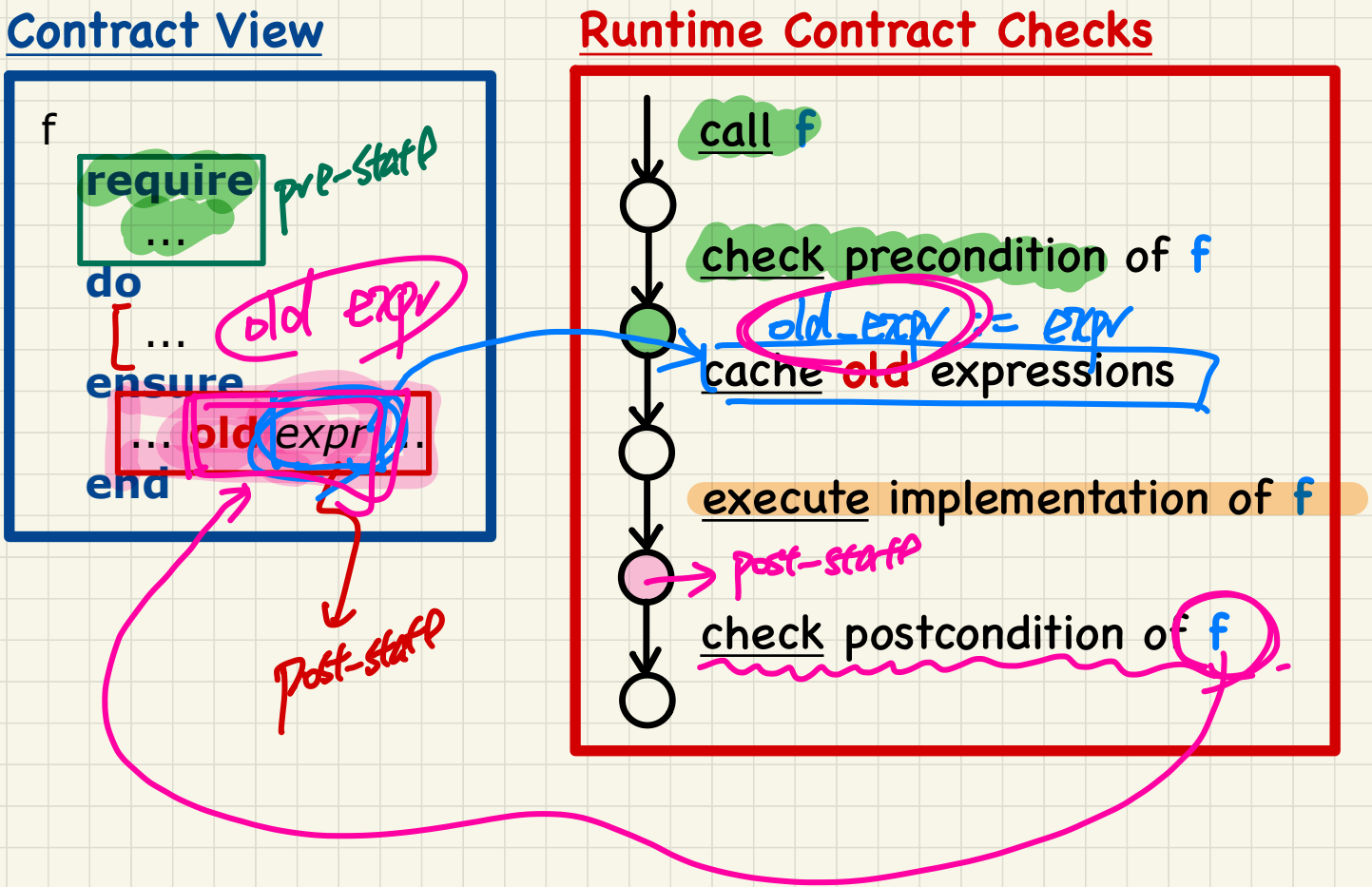




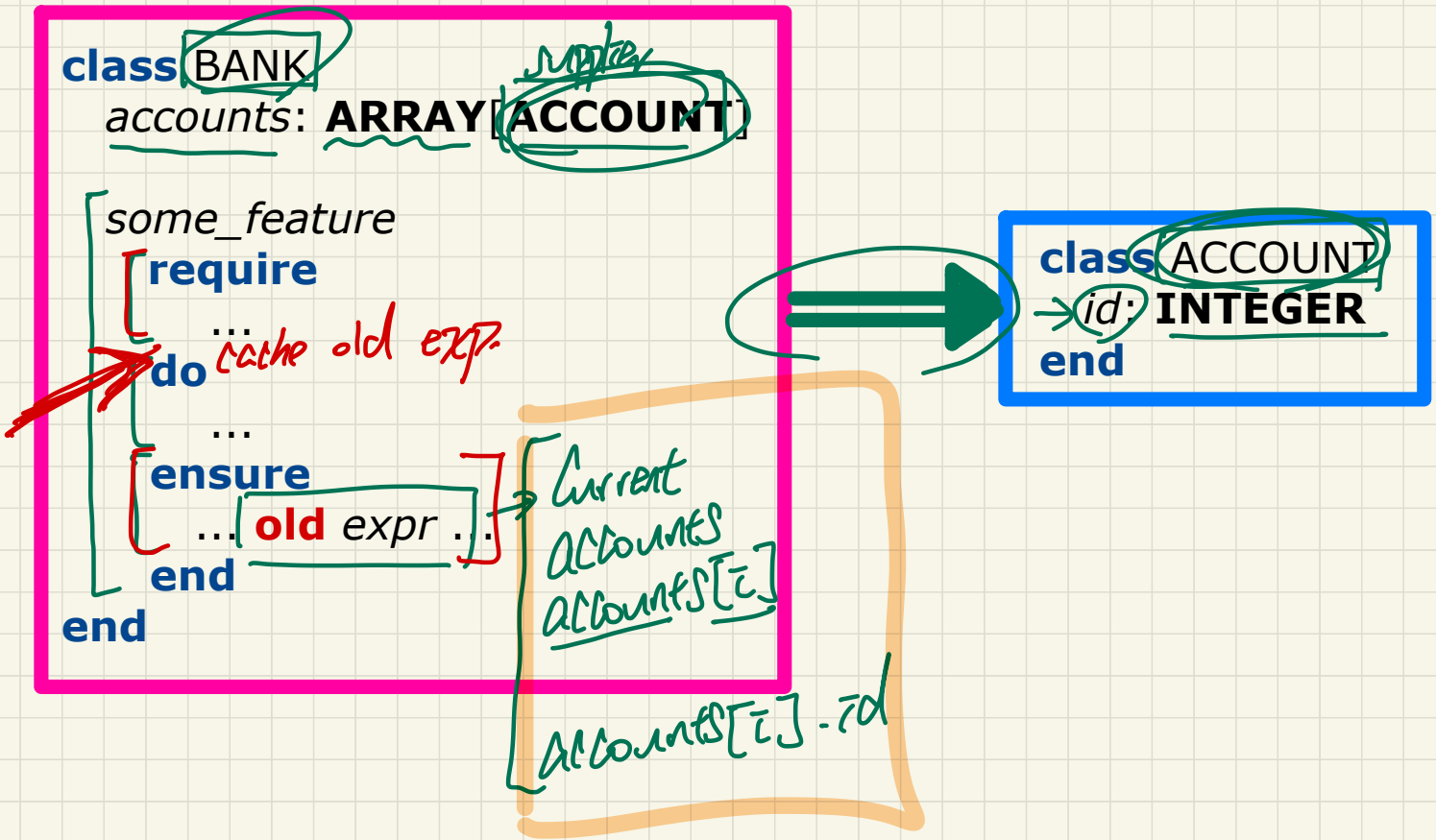
Lecture 2

Part 3

Writing Complete Postconditions


Contract View
f
require
...
pre-state
do
...
old expr
ensure
... old expr ...
end
post-state
Runtime Contract Checks
call f
check precondition of f
old_expr := expr
cache old expressions
execute implementation of f
post-state
check postcondition of f

Caching Values for old Expressions in Postconditions



Caching Values for old Expressions in Postconditions

Lecture 1 → DbC Java Postcond.

ensure (in context of **BANK**)

① (old accounts[i].id) → Inst. A[A]

② (old accounts[i]).id

③ (old accounts[i].twin).id → Account

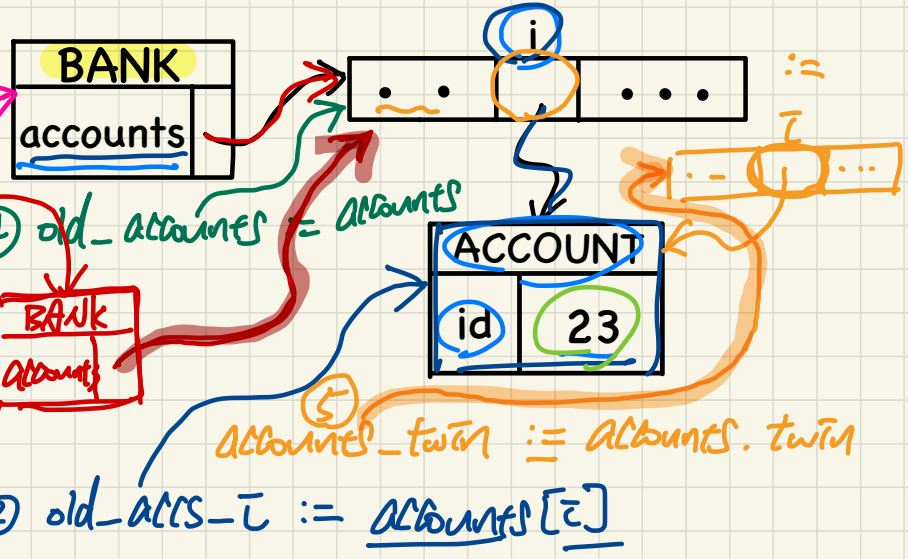
④ (old accounts)[i].id → A[Acc]

⑤ (old accounts.twin)[i].id → A[Acc]

⑥ (old Current).accounts[i].id → BANK

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

How to cache at runtime?



Revisit: Bank Accounts in Java V5

```
1  public class AccountV5 {
2    public void withdraw(int amount) throws
3        WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
4      int oldBalance = this.balance;
5      if(amount < 0) { /* negated precondition */
6        throw new WithdrawAmountNegativeException(); }
7      else if (balance < amount) { /* negated precondition */
8        throw new WithdrawAmountTooLargeException(); }
9      else { this.balance = this.balance - amount; }
10     assert this.getBalance() > 0 :"Invariant: positive balance";
11     assert this.getBalance() == oldBalance - amount :
12       "Postcondition: balance deducted"; }
```

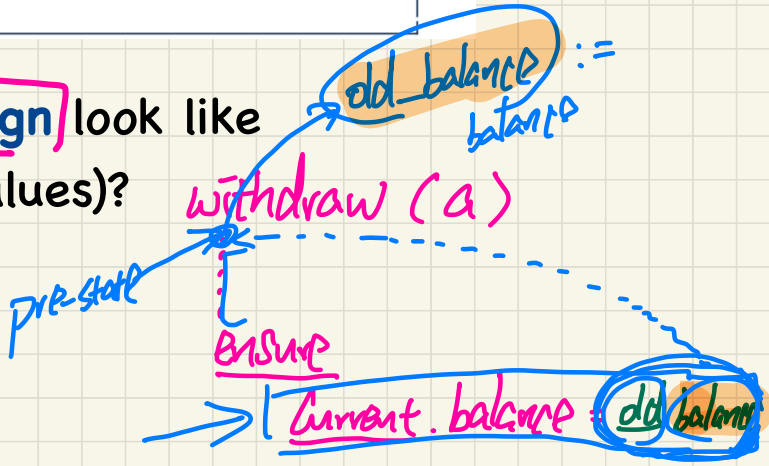
pre-state

manual

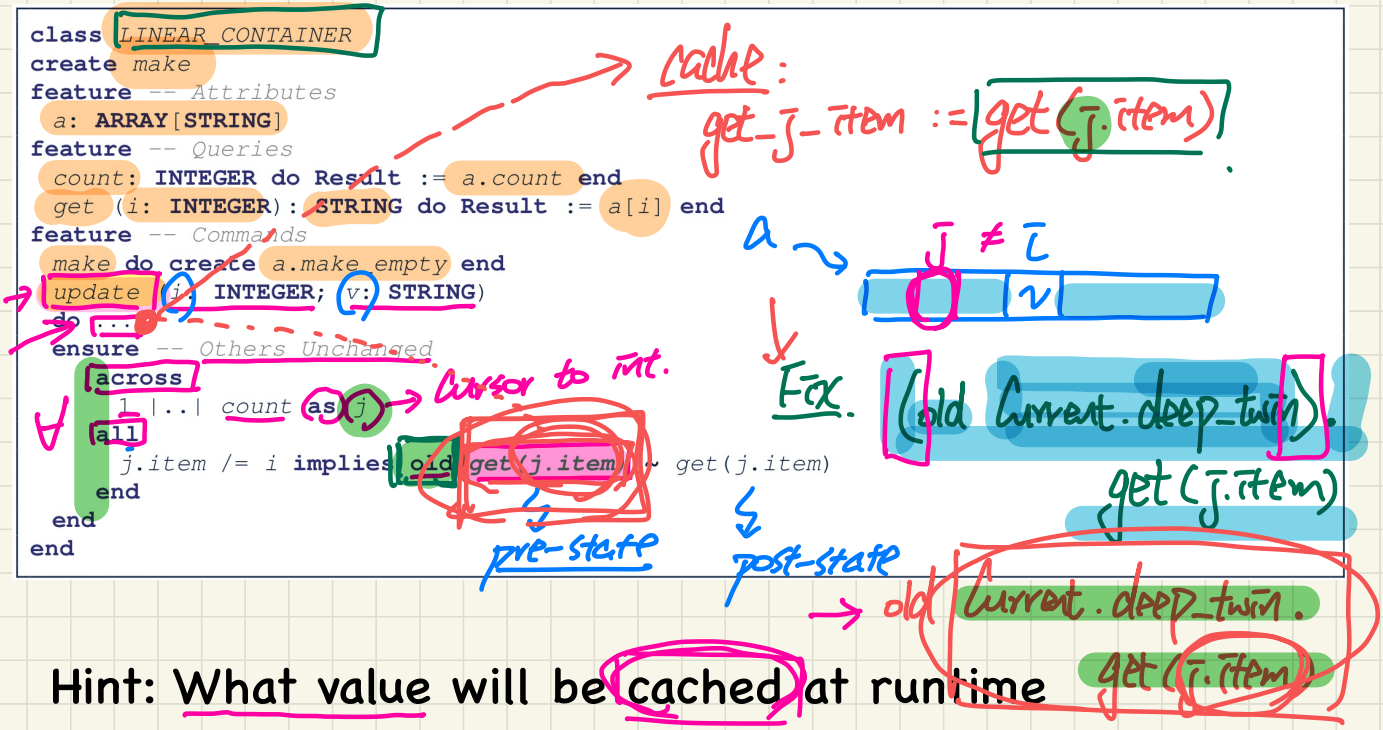
post-state

pre-state

How does the corresponding Eiffel design look like (with automatic caching of pre-state values)?



Use of old in across Expression in Postcondition



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

```
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
```

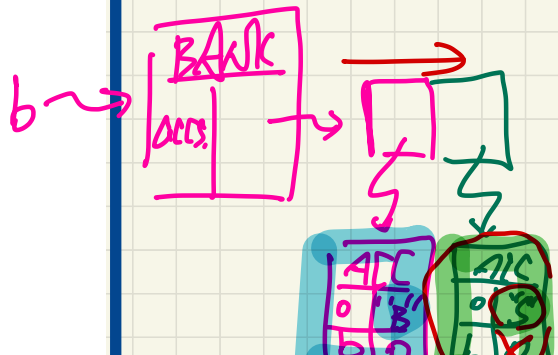
```
  deposit(a: INTEGER)
    do
      balance := balance + a
    ensure
      balance = old balance + a
    end

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

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

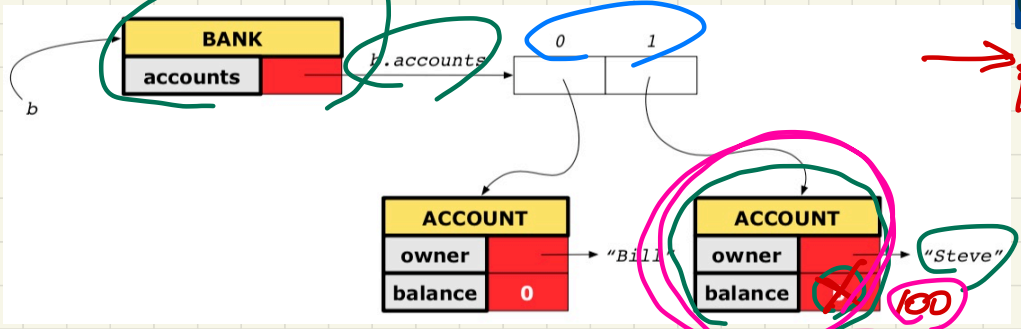
$$\neg(\exists x \mid R(x) \cdot \neg P(x))$$

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



pre

b.deposit("Steve", 100)

post

```
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
```

Steve

2 = 2 (T)

T

100 = 0 + 100

100 = 100 (T)

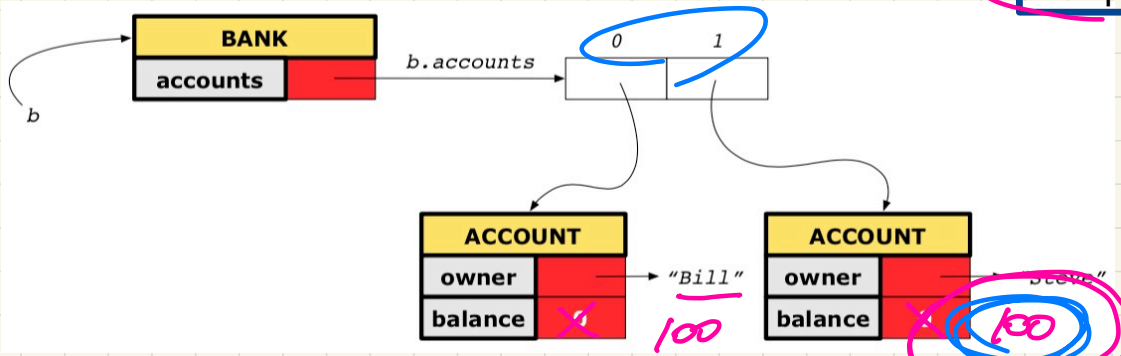
2. cache: accs_count := accounts.count

imp (correct)

0. i_a_o_n_b := ...

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)



Imp. wrong
∧ no post-cond. violation
⇒ non-satisfactory.

v1 2
v2 0
cache

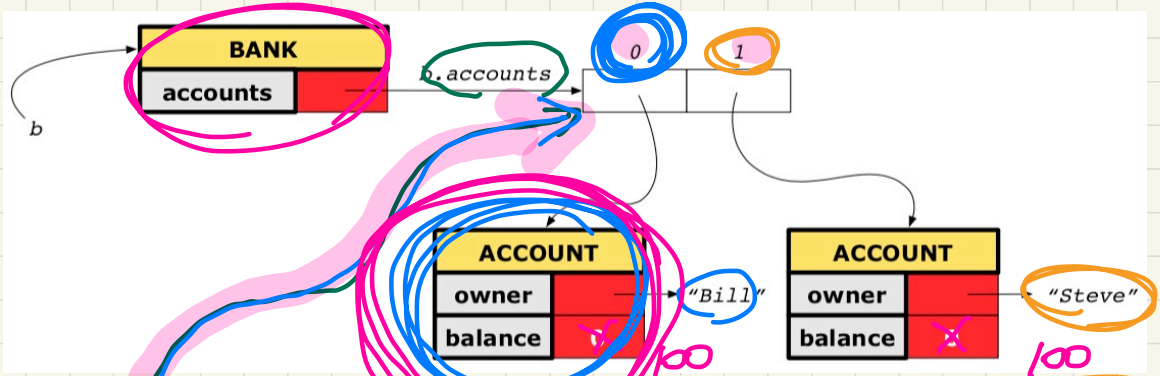
```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
  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

correct
incorrect

2 2 v1 T

v2 100 = 0 + 100 T

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



b.deposit('Steve', 100)

cache.
old_accs := accounts

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

T ⇒ T ≡ T

1st Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

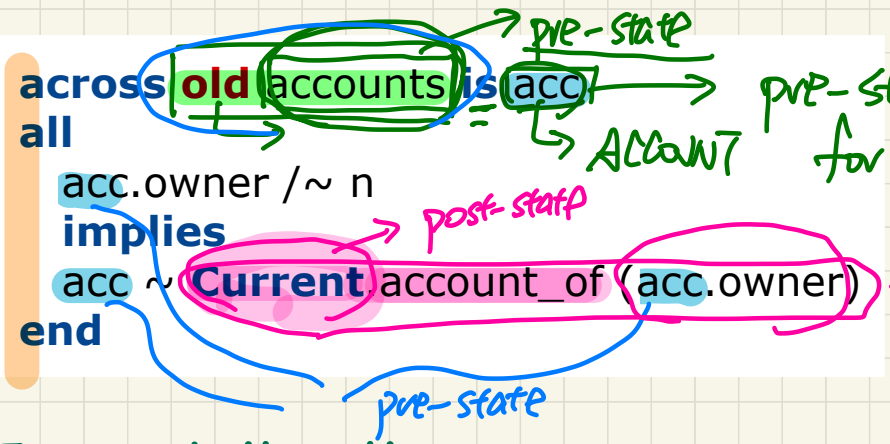
Bill Steve

2nd Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

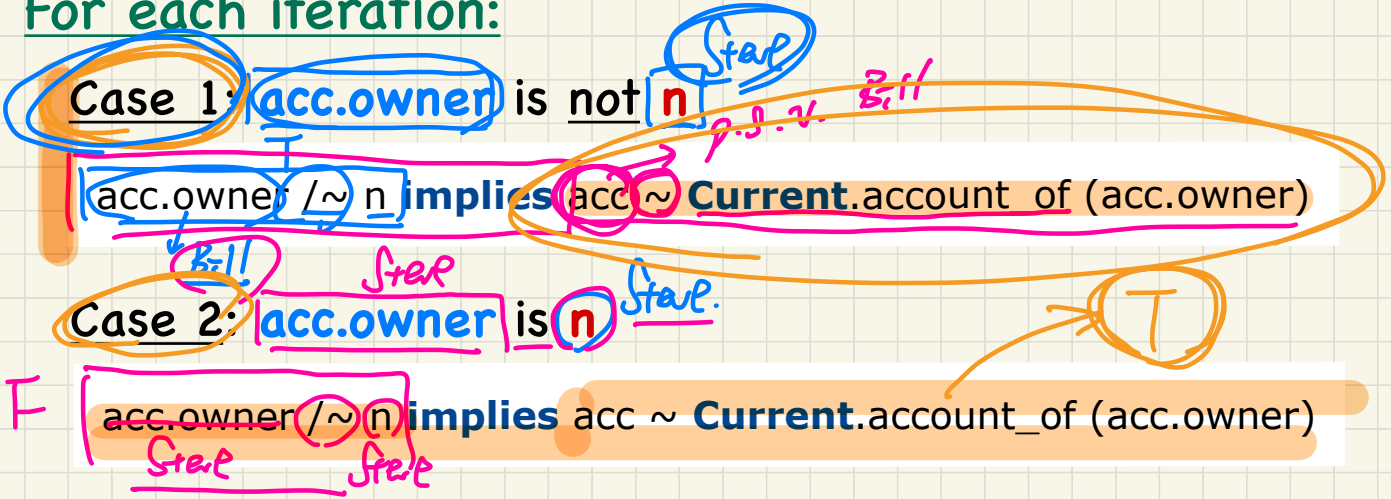
Steve Steve

Use of across in Postcondition

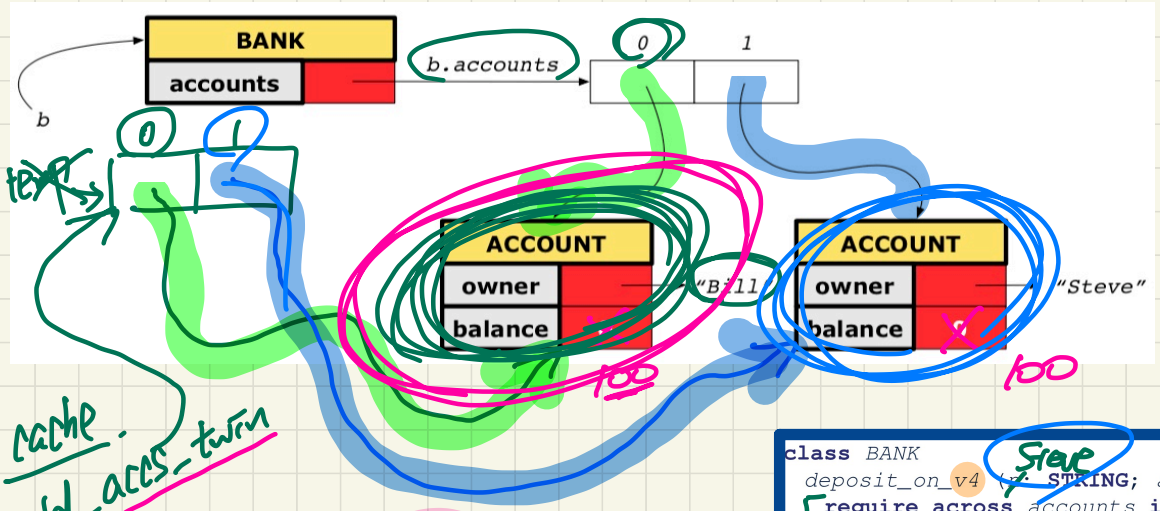


T ⇒ P ≡ P.
F ⇒ P ≡ T

For each iteration:



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



b.deposit("Steve", 100)

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

temp[1] := b.accs[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
```

pre-state

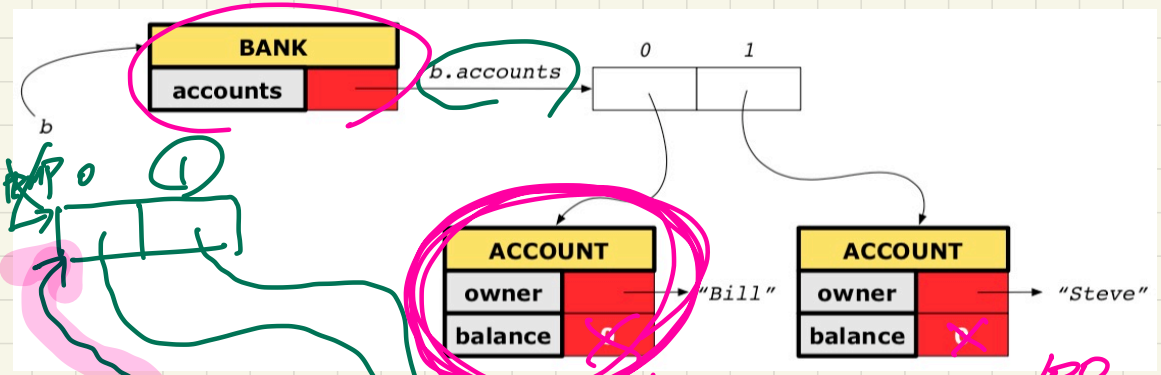
1st Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

2nd Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

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

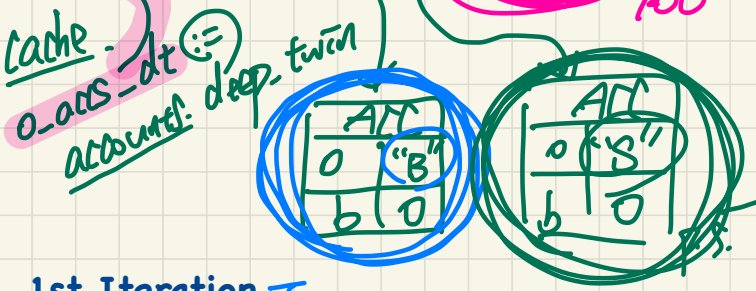


b.deposit("Steve", 100)

temp[0] := b.accs[0].dt

temp[1] := b.accs[1].dt

appropriate
wrong
Postcond.




```
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
```

1st Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

2nd Iteration

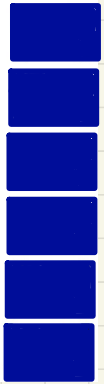
acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

Complete Postcondition: Exercise

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

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

- `accounts = old accounts`
- `accounts = old accounts.twin`
- `accounts = old accounts.deep_twin`
- `accounts ~ old accounts`
- `accounts ~ old accounts.twin`
- `accounts ~ old accounts.deep_twin`



Lecture 3

Part 1

Writing & Using a Generic Class

Stack of Strings vs. Stack of Accounts

```
class STRING_STACK
feature {NONE} -- Implementation
  imp: ARRAY[ STRING ] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
     -- Number of items on stack.
  top: STRING do Result := imp [i] end
     -- Return top of stack.
feature -- Commands
  push (v: STRING ) do imp[i] := v; i := i + 1 end
     -- Add 'v' to top of stack.
  pop do i := i - 1 end
     -- Remove top of stack.
end
```

parameter for the type of stack items

[G, H]

usages of generic parameter

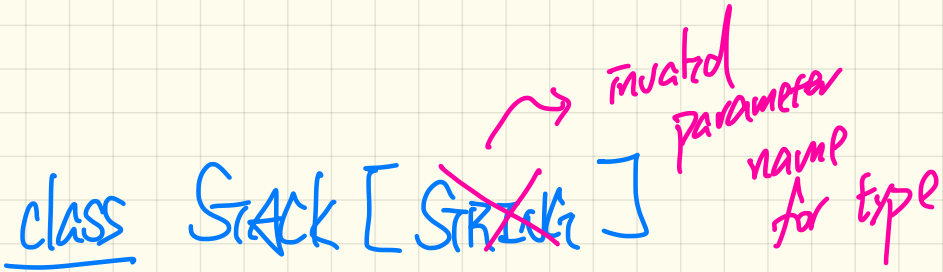
supplier of a generic class

should not clash with existing classes

```
add (i, j : INTEGER)
do
  R := i + j
end
```

param, type, use of param

```
class ACCOUNT_STACK
feature {NONE} -- Implementation
  imp: ARRAY[ ACCOUNT ] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
     -- Number of items on stack.
  top: ACCOUNT do Result := imp [i] end
     -- Return top of stack.
feature -- Commands
  push (v: ACCOUNT ) do imp[i] := v; i := i + 1 end
     -- Add 'v' to top of stack.
  pop do i := i - 1 end
     -- Remove top of stack.
end
```


class STACK [STRING]
invalid parameter name for type

A Generic Stack

Supplier

```
class STACK [G]
feature {NONE} -- Implementation
  imp: ARRAY[G] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
      -- Number of items on stack.
  top: G do Result := imp [i] end
      -- Return top of stack.
feature -- Commands
  push (v: G) do imp[i] := v; i := i + 1 end
      -- Add v to top of stack.
  pop do i := i - 1 end
      -- Remove top of stack.
end
```

Client

```
1  test_stacks: BOOLEAN
2    local
3      ss: STACK[STRING] ; sa: STACK[ACCOUNT]
4      s: STRING ; a: ACCOUNT
5    do
6      ss.push("A")
7      ss.push(create {ACCOUNT}.make ("Mark", 200))
8      s := ss.top
9      a := ss.top
10     sa.push(create {ACCOUNT}.make ("Alan", 100))
11     sa.push("B")
12     a := sa.top
13     s := sa.top
14   end
```

Lecture 3

Part 2

Abstractions via Mathematical Models

Implementing a LIFO Stack

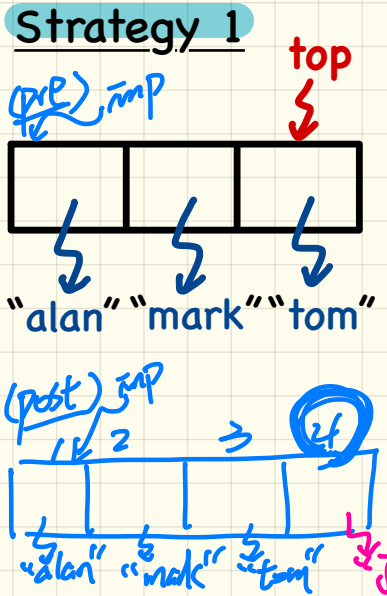
"tom"
"mark"
"alan"

S.push("Jim")

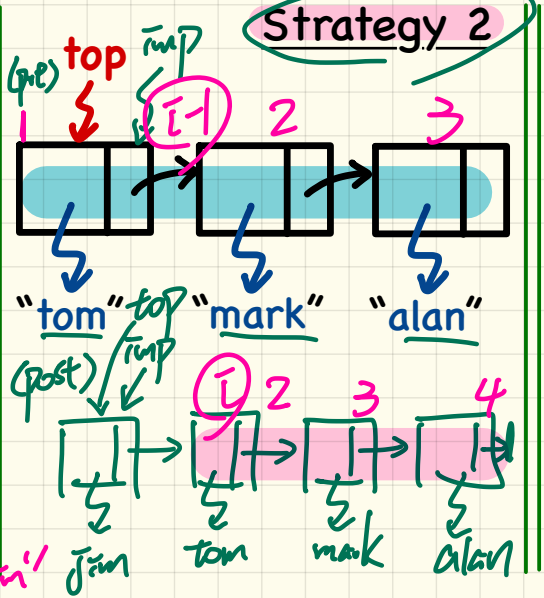
① $\forall i \mid 1 \le i \le (tmp.count - 1) \bullet$ old tmp.deep_twin[i] ~ tmp[i]

② $\forall i \mid 2 \le i \le tmp.count \bullet$ old tmp.deep_twin[i-1] ~ tmp[i]

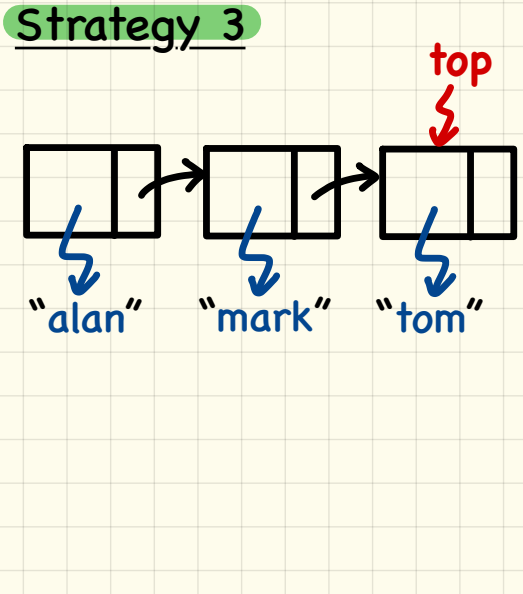
Strategy 1



Strategy 2



Strategy 3



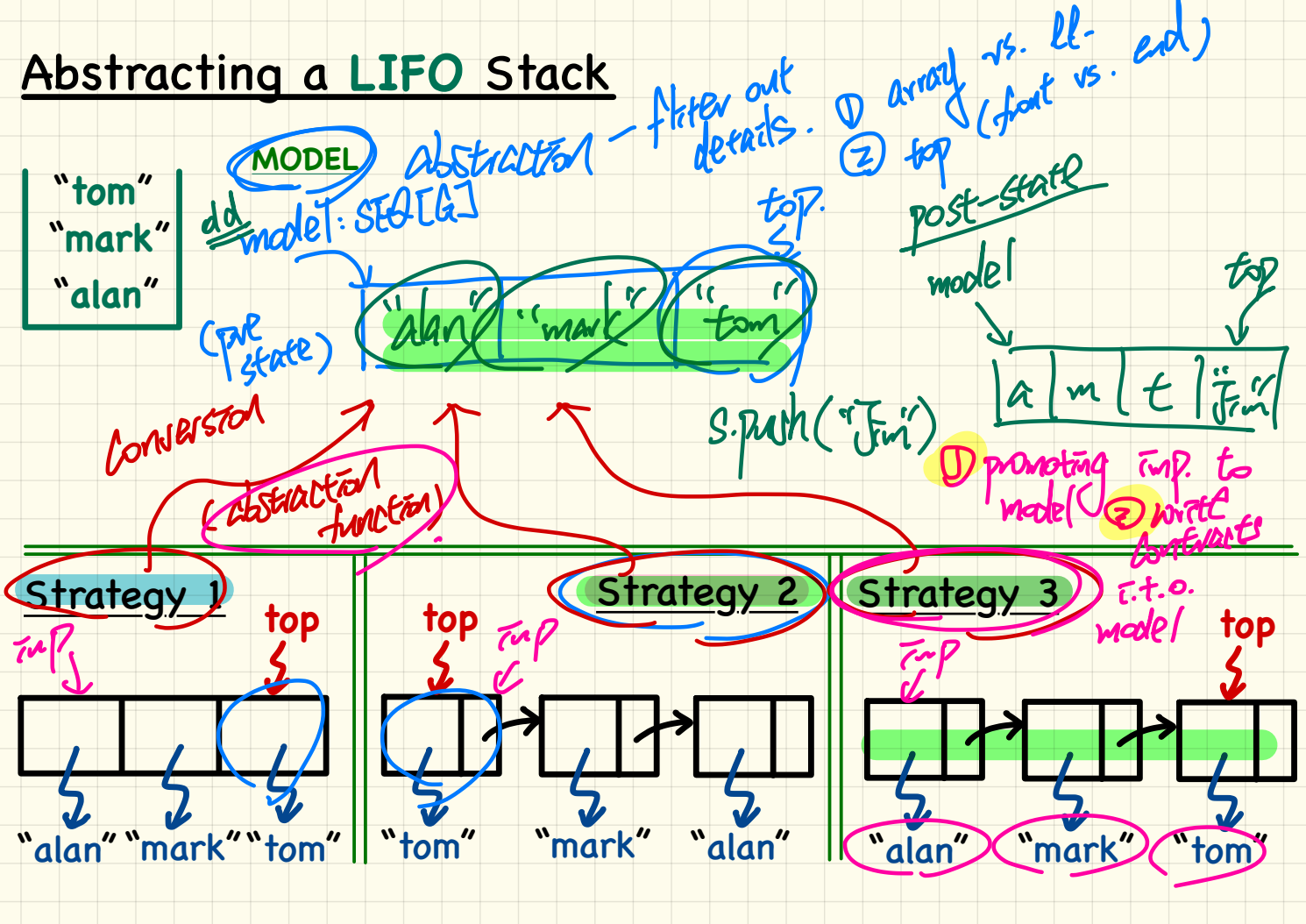
Developing a LIFO Stack

```
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
```

Abstracting a LIFO Stack



Using MATHMODELS Library

Implementing an Abstraction Function

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

Strategy 3

strategy 2.

abs. func.

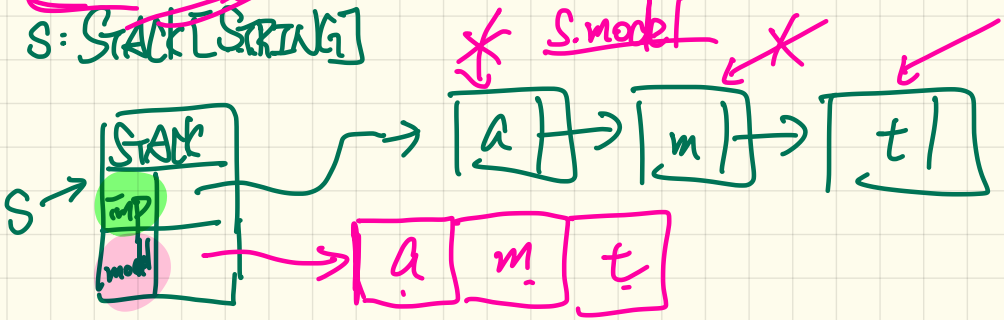
SEQ

command from SEQ.

ensure Result related to imp

Exercise 1: Write postcondition of **model**.

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



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
```

post-state

pre-state

Question: Can clients tell which strategy is being adopted?

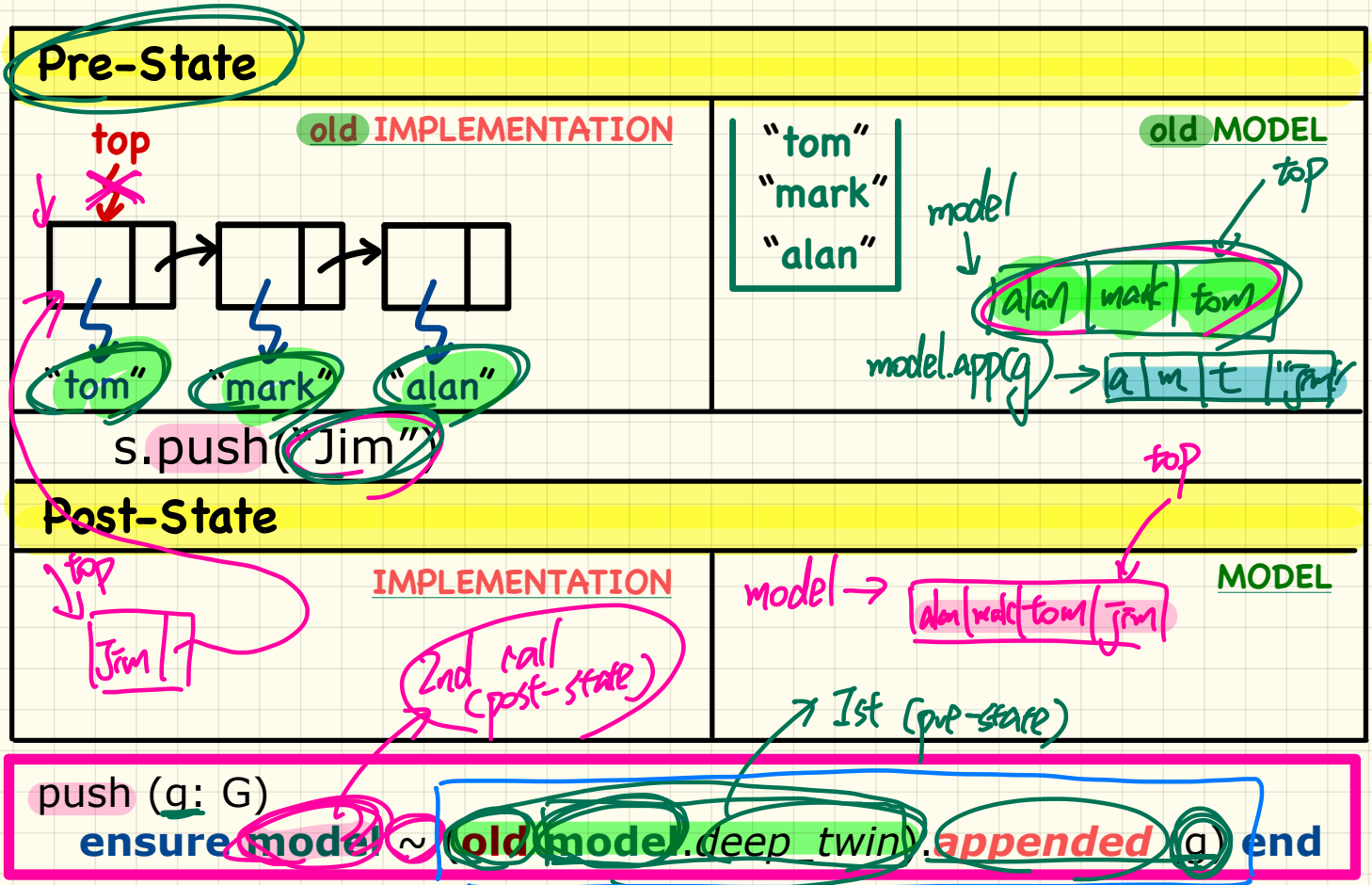
No. ⇒ Information Hiding!

Exercise: What if strategy was changed? Change what?

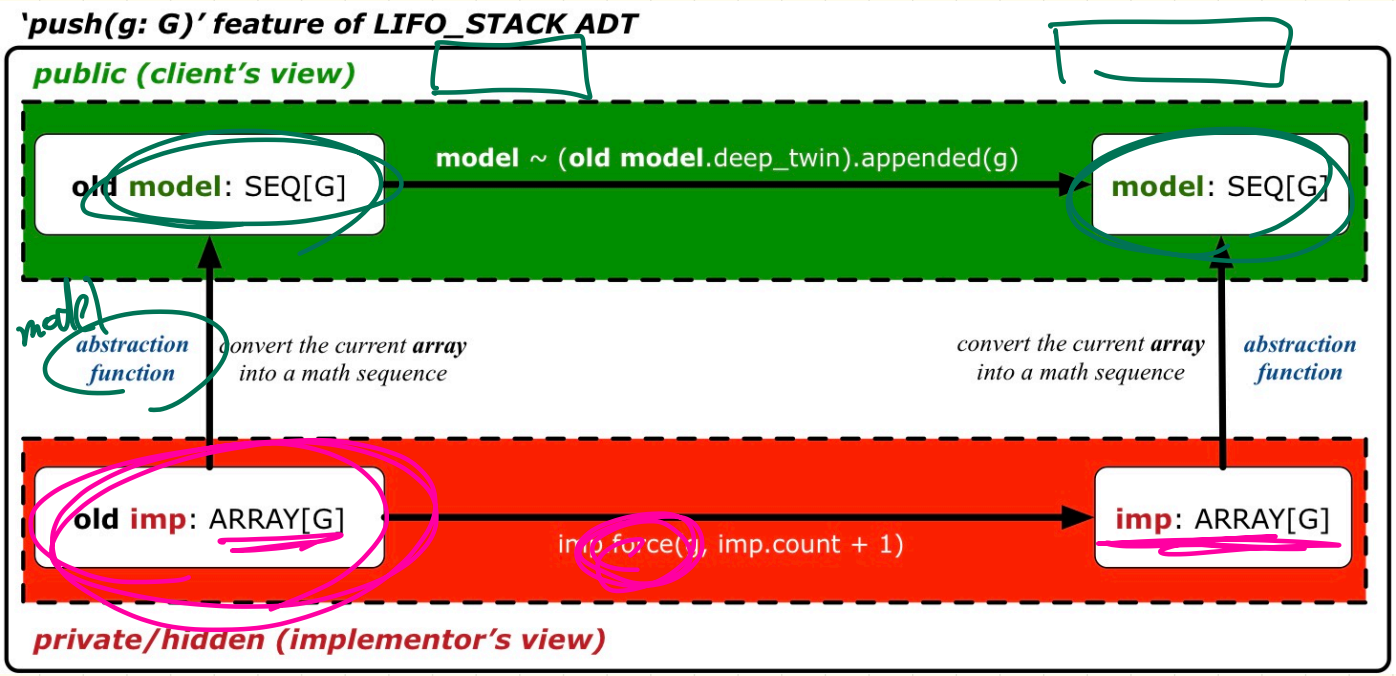
↳ change the abstraction function (i.e. inv. and contract of model)

Checking MATHMODELS Contracts at Runtime

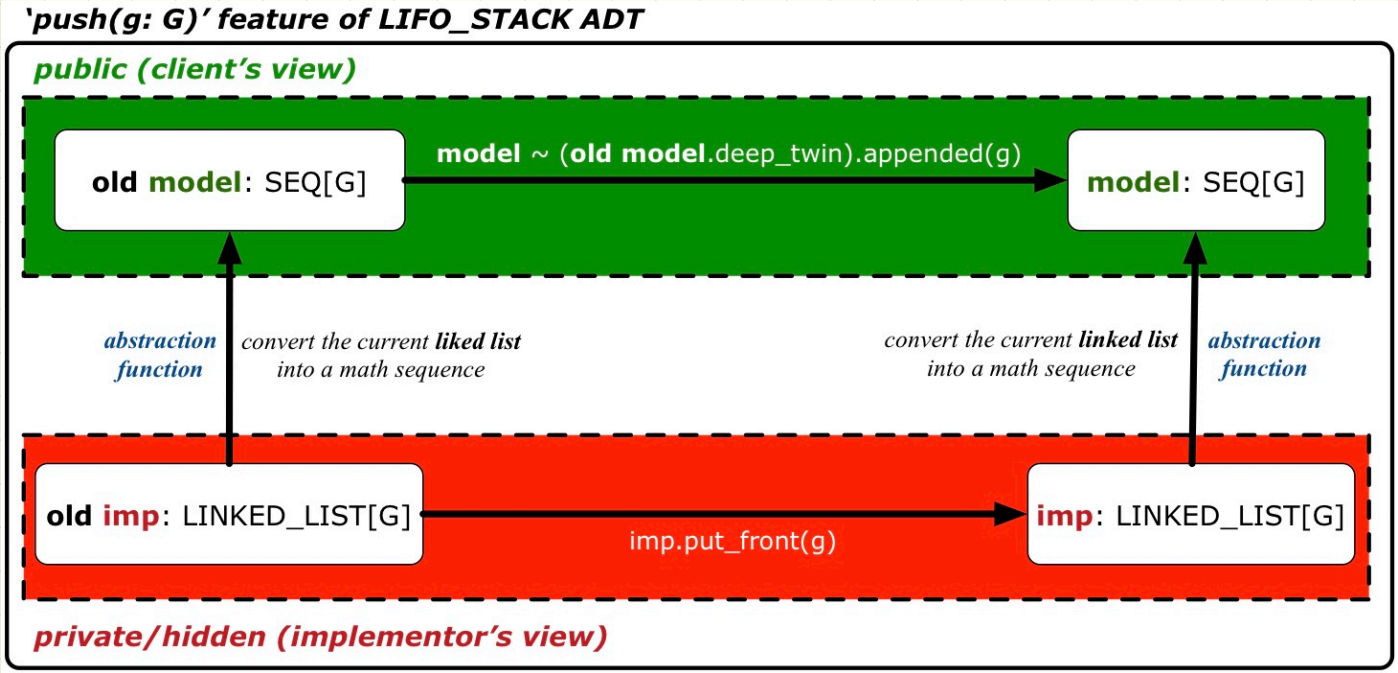
Strategy 2



Strategy 1: Mathematical Abstraction



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]
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
```

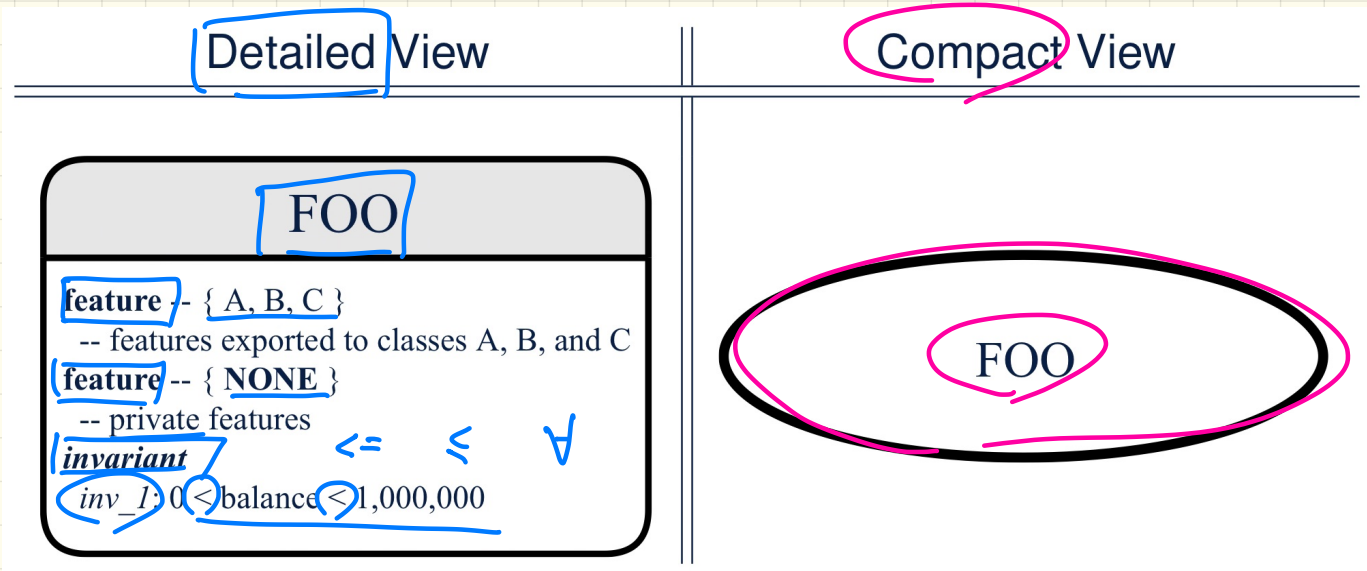
```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 3 (last as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.append(cursor.item) end
    ensure
      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
```

Lecture 4

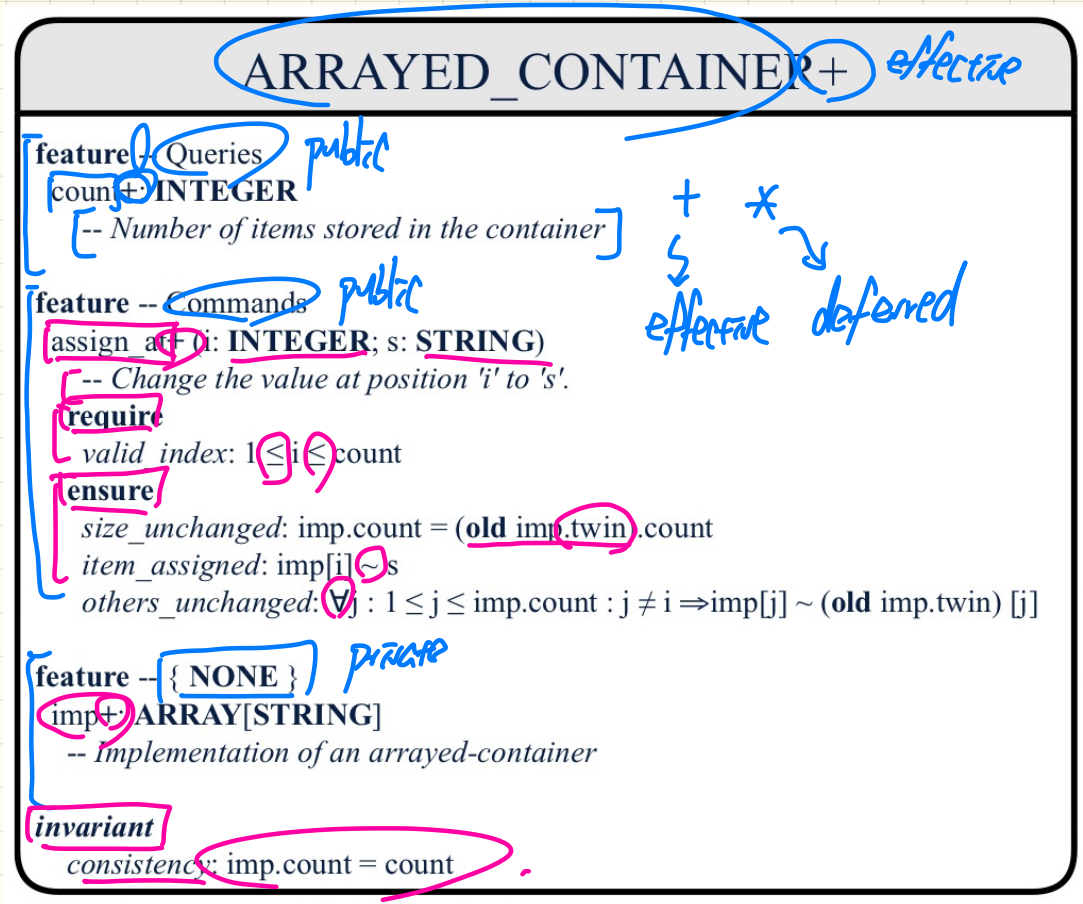
Part 1

Architectural Design Diagrams

Classes: Detailed vs. Compact



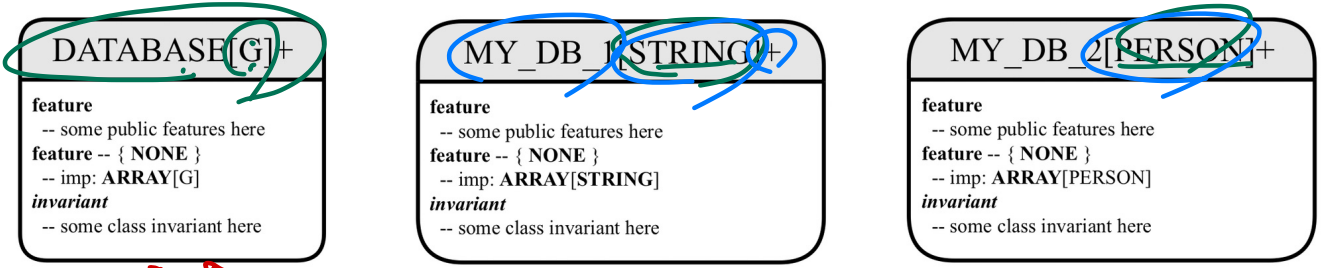
Contracts: Mathematical vs Programming



Generic Classes

- Type parameter(s) of a class may or may not be ***instantiated***:

- If necessary, present a generic class in the detailed form:

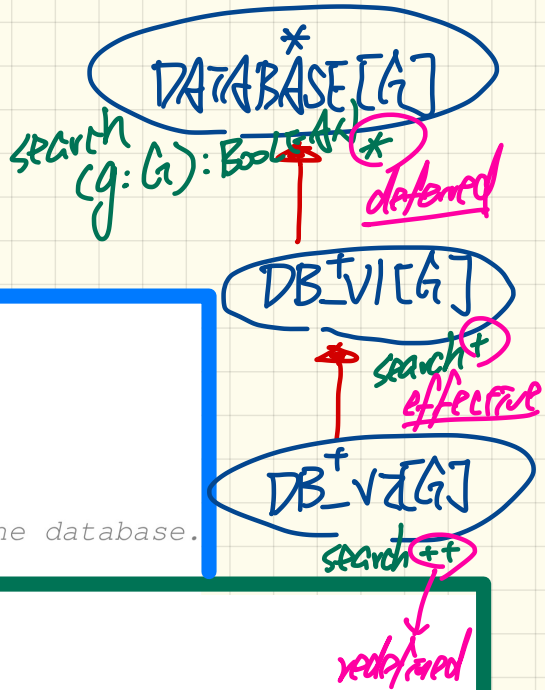


Programming Classes: Deferred vs. Effective

```
deferred class
  DATABASE [G]
feature -- Queries
  search (g: G): BOOLEAN
      -- Does item 'g' exist in database?
  deferred end
end
```

```
class
  DATABASE_V1 [G]
inherit
  DATABASE [G]
feature -- Queries
  search (g: G): BOOLEAN
      -- Perform a linear search on the database.
  do end
end
```

```
class
  DATABASE_V2 [G]
inherit
  DATABASE_V1 [G]
    redefine search end
feature -- Queries
  search (g: G): BOOLEAN
      -- Perform a binary search on the database.
  do end
end
```



Presenting Deferred/Effective Classes in Compact Form

Presenting Deferred/Effective Classes in Detailed Form

DATABASE[G]*

feature {NONE} -- Implementation
 data: **ARRAY**[G]

feature -- Commands
 add_item*(g: G)
  -- Add new item `g` into database.
  require
  non_existing_item: ¬ exists (g)
  ensure
  size_incremented: count = **old** count + 1
  item_added: exists (g)

feature -- Queries
 count+: **INTEGER**
  -- Number of items stored in database
  ensure
  correct_result: **Result** = data.count

 exists*(g: G): **BOOLEAN**
  -- Does item `g` exist in database?
  ensure
  correct_result: **Result** = (∃i: 1 ≤ i ≤ count : data[i] ~ g)

DATABASE_V1[G]+

feature {NONE} -- Implementation
 data: **ARRAY**[G]

feature -- Commands
 add_item+(g: G)
  -- Append new item `g` into end of `data`.

feature -- Queries
 count+: **INTEGER**
  -- Number of items stored in database

 exists+(g: G): **BOOLEAN**
  -- Perform a linear search on `data` array.

DATABASE_V2[G]+

feature {NONE} -- Implementation
 data: **ARRAY**[G]

feature -- Commands
 add_item++(g: G)
  -- Insert new item `g` into the right slot of `data`.

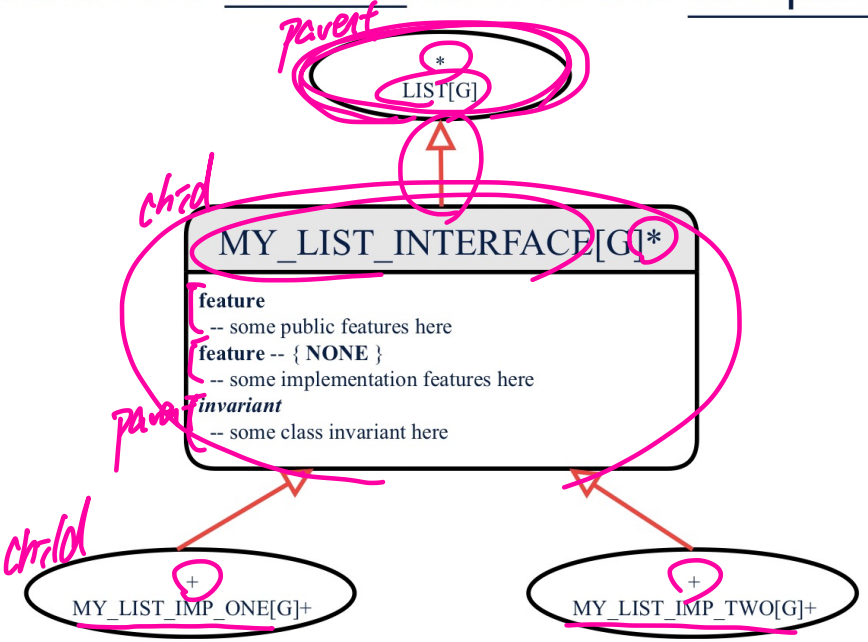
feature -- Queries
 count+: **INTEGER**
  -- Number of items stored in database

 exists++(g: G): **BOOLEAN**
  -- Perform a binary search on `data` array.

 invariant
 sorted_data: ∀i : 1 ≤ i < count : data[i] < data[i + 1]

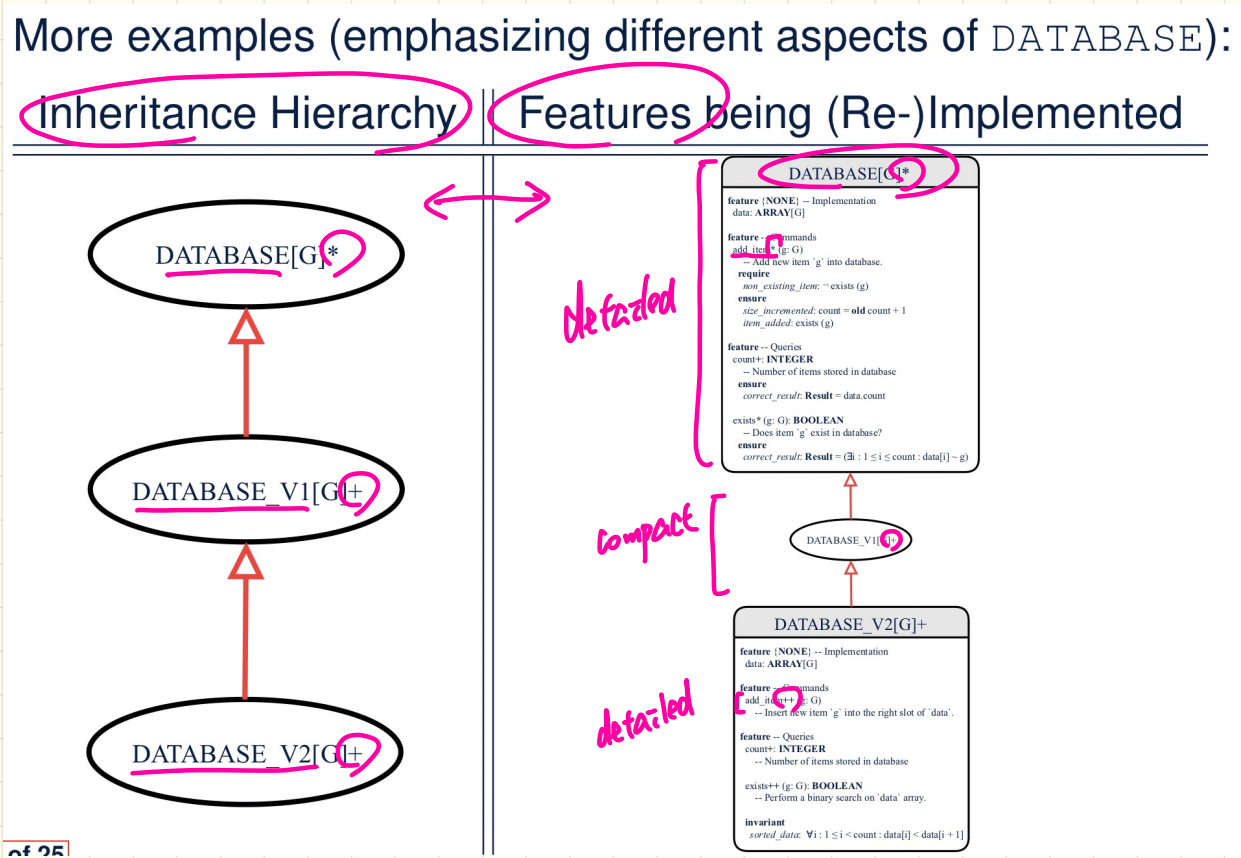
Inheritace (1)

- You may choose to present each class in an inheritance hierarchy in either the detailed form or the compact form:



Inheritace (2)

More examples (emphasizing different aspects of DATABASE):



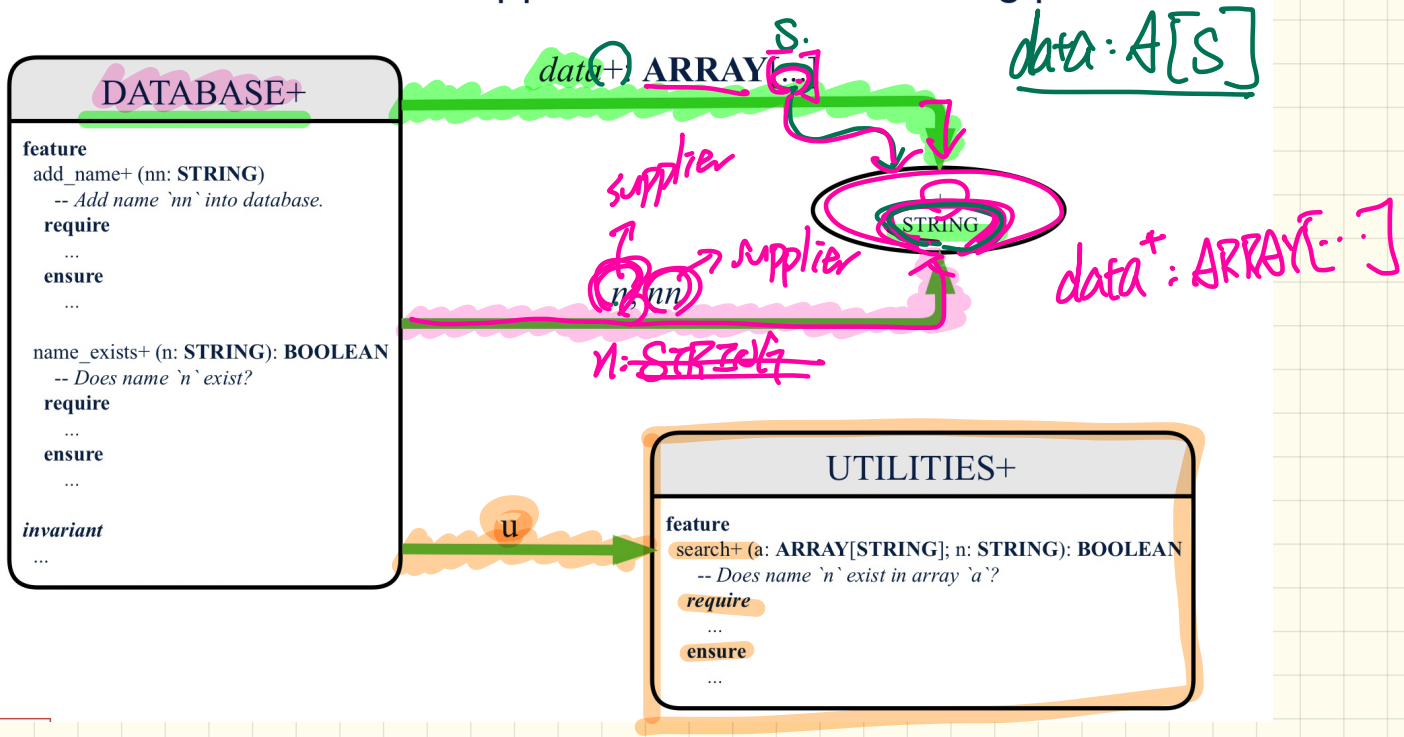
Programming Client-Supplier Relation

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

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

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

Presenting CS Relation in Diagram: Approach 1

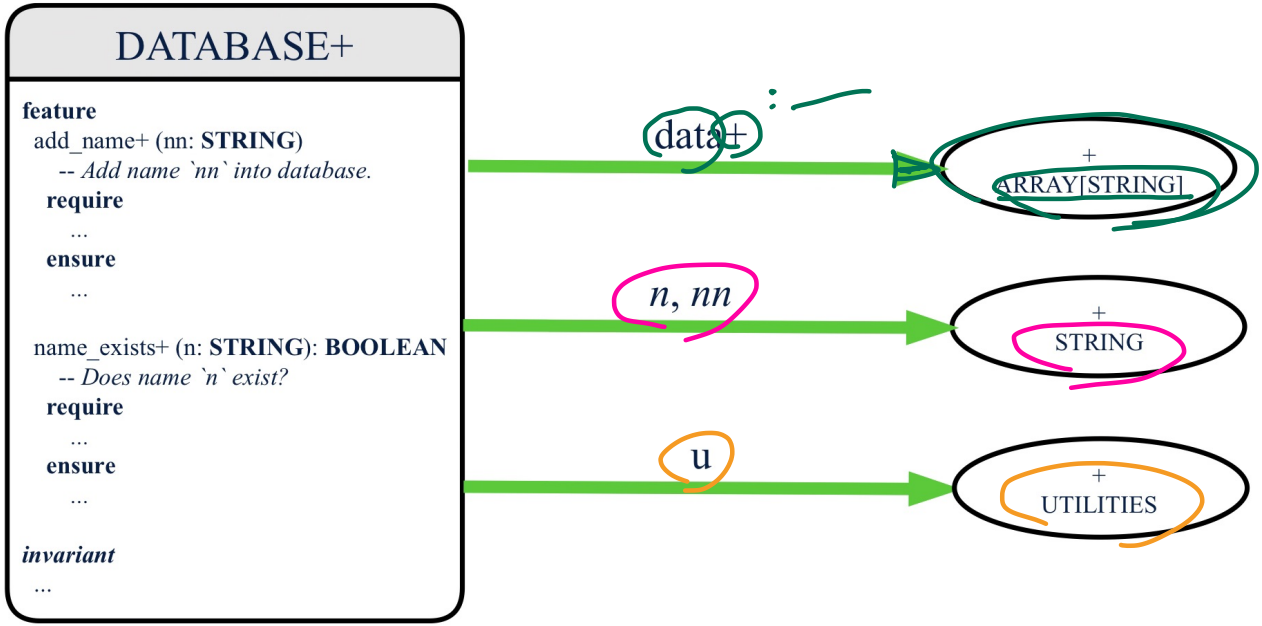
If `STRING` is to be emphasized, label is `data: ARRAY[...]`, where ... denotes the supplier class `STRING` being pointed to.



Presenting CS Relation in Diagram: Approach 2

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

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



Programming Client-Supplier Relation

Design One:

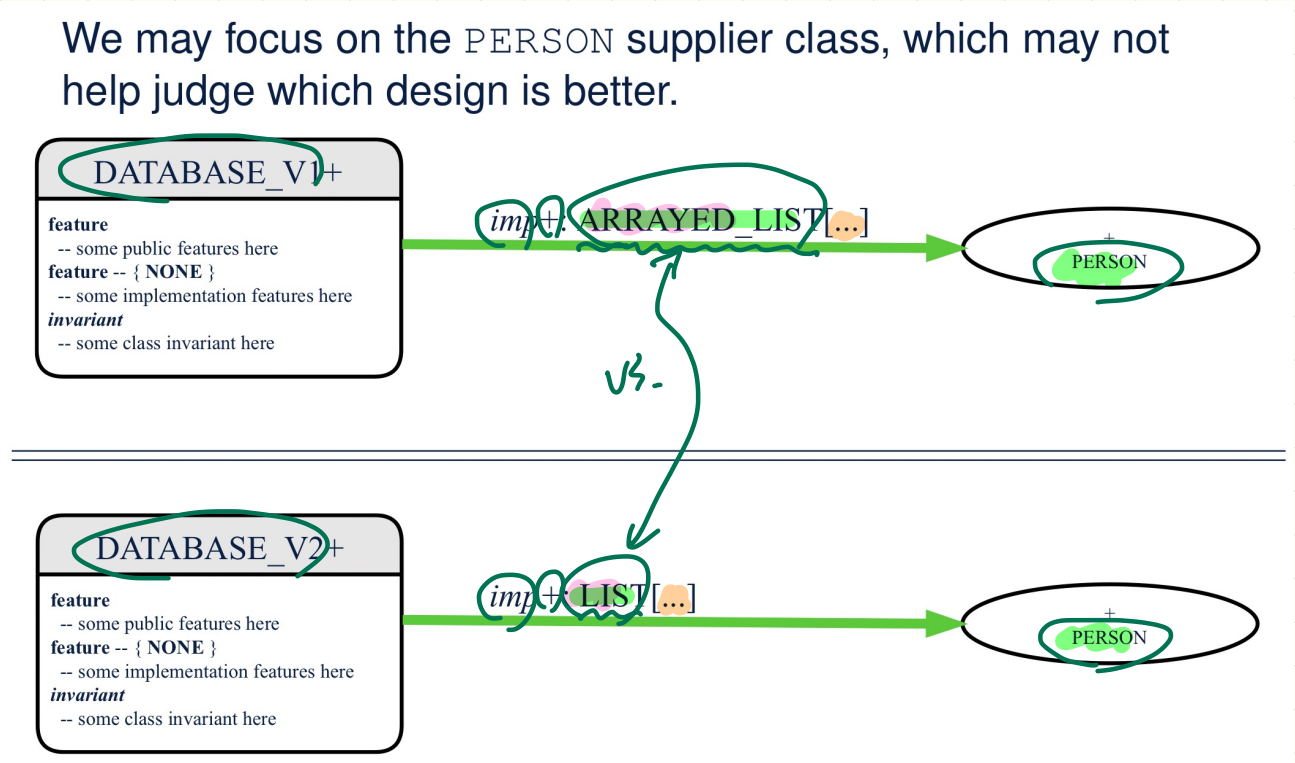
```
class DATABASE_V1
feature {NONE} -- implementation
  imp: ARRAYED_LIST[PERSON]
... -- more features and contracts
end
```

Design Two:

```
class DATABASE_V2
feature {NONE} -- implementation
  imp: LIST[PERSON]
... -- more features and contracts
end
```

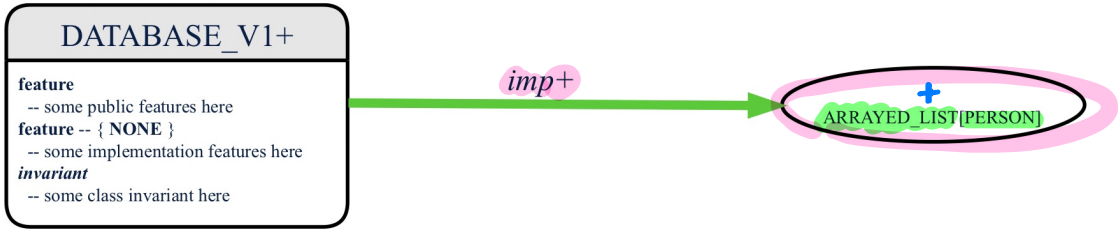
Presenting CS Relation in Diagram: Approach 1

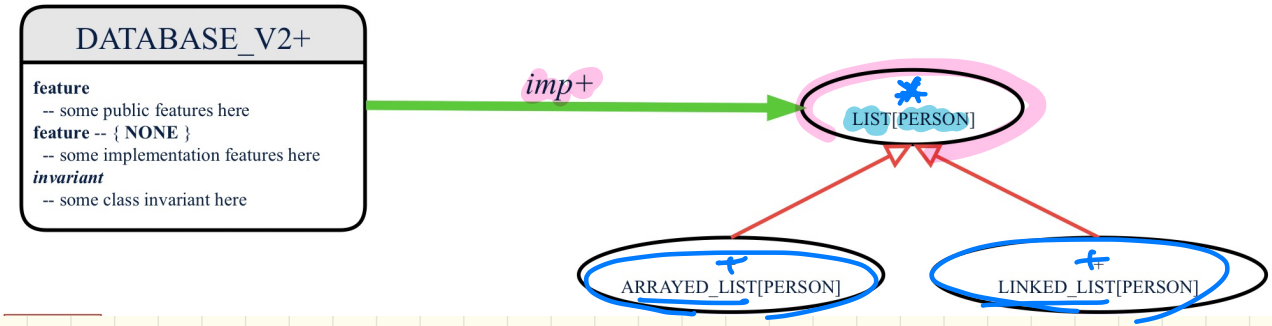
We may focus on the `PERSON` supplier class, which may not help judge which design is better.



Presenting CS Relation in Diagram: Approach 2

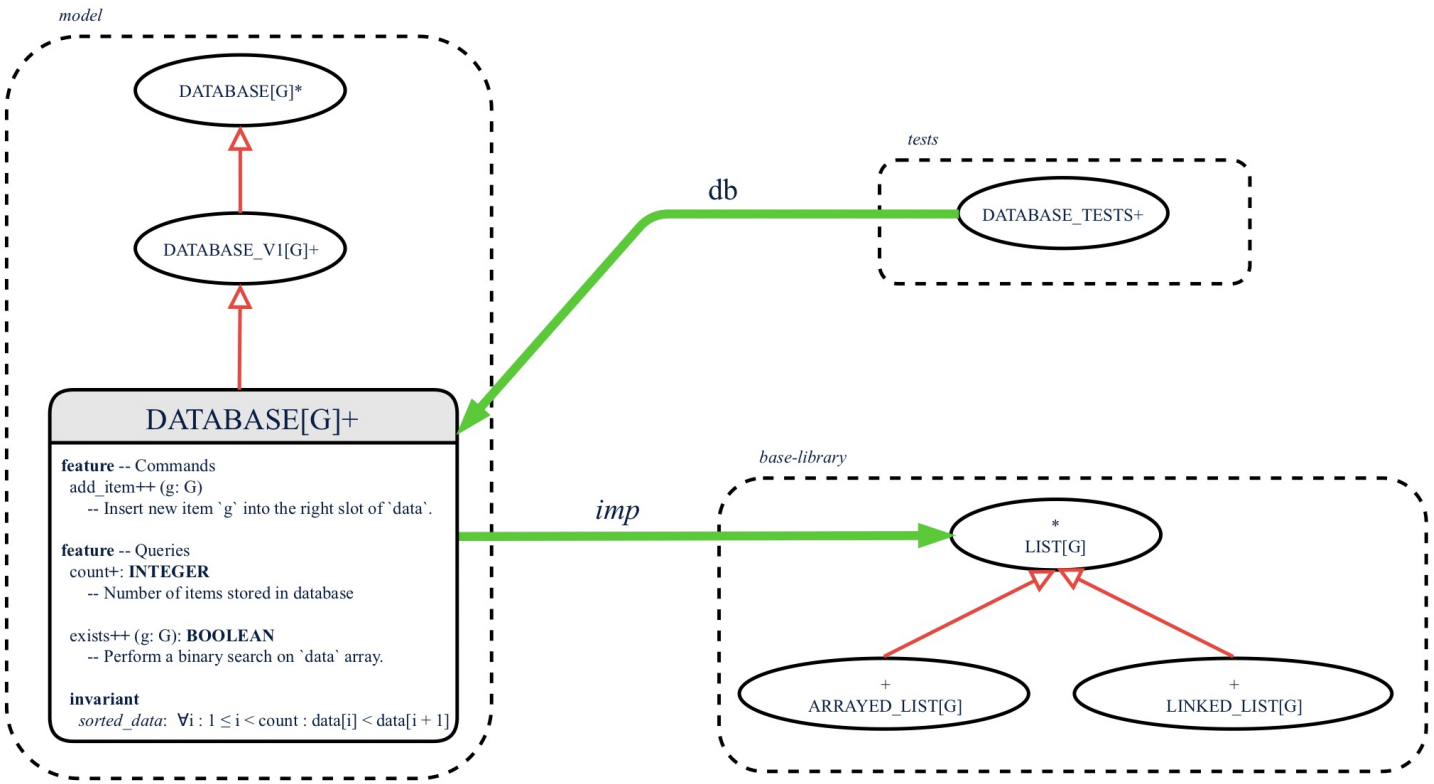
Alternatively, we may focus on the `LIST` supplier class, which in this case helps us judge which design is better.





Clusters

Use *clusters* to group classes into logical units.



Lecture 4

Part 2

Postcondition: Asserting Set Equality

Writing Postcondition: Exercise

Problem

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

require

no_duplicates: ??

ensure

across Result **is** x

all

x > 0

end

all_p_vs (<< 2, 3, -1, 4, -2 >>)

↳ << 2, 3, 4 >>

Correct

incomplete

Witness/Counter-example

all_p_vs (<< 2, 3, -1, 4, -2 >>)

wrong imp.

↳ << 1, 5, 7 >>

↳ since each num. is > 0 ⇒ violation no postcond.

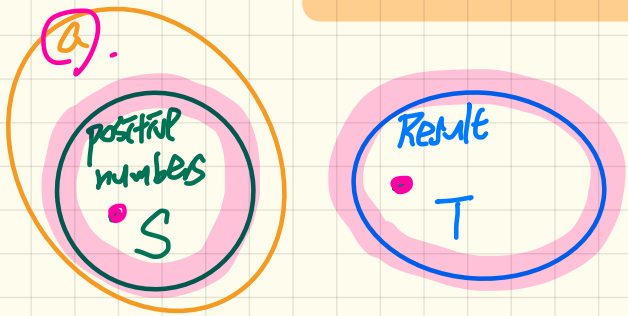
But the output is incorrect

design is flawed

Writing Postcondition: Exercise

Solution

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



$S = T \equiv (\forall x \mid x \in S \Rightarrow x \in T) \wedge (x \in T \Rightarrow x \in S)$

$S \subseteq T$ $\quad T \subseteq S$

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

```
all_num_in_result_pos_and_
    in_a :
    across Result is x
    all x > 0 and a.has(x) end
```

Lecture 4

Part 3

Abstraction of a Birthday Book

Testing REL in MATHMODELS

r.count → 9

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

- ✓ r.domain → {a, b, c, d, e, f}
- r.range → {1, 2, 3, 4, 5, 6}
- ✓ r.image(a) → {1, 4} → function
- ✓ r[g] → ∅

r.image(⊖).count = 1 → singleton set

domain → at most 1 value in range

domain value

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

t

(a,3)
(c,4)

r.overridden({(a,3), (c,4)})

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

t r.domain_subtracted(t.domain) {a,c}

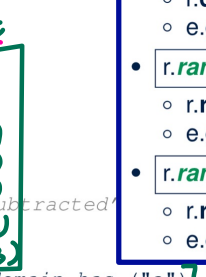
= {(a,3), (c,4), (b,2), (b,5), (d,1), (e,2), (f,3)}

Testing REL in MATHMODELS

Command : imp. of model

query : contracts

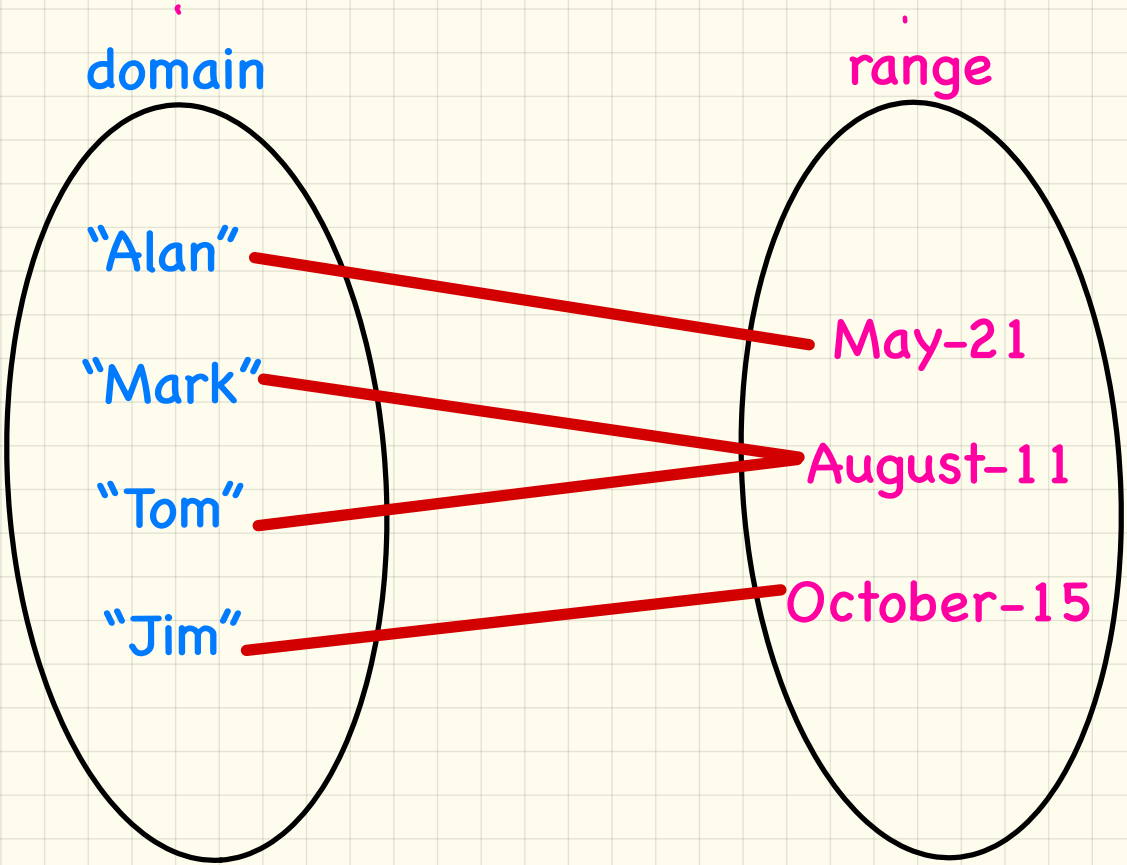
```
test_rel: BOOLEAN
 local
  r, t: REL[STRING, INTEGER]
  ds: SET[STRING]
 do
  create r.make_from_tuple_array (
   <<["a", 1], ["b", 2], ["c", 3],
    ["a", 4], ["b", 5], ["c", 6],
    ["d", 1], ["e", 2], ["f", 3]>>)
  create ds.make_from_array (<<"a">>)
  -- r is not changed by the query 'domain_subtracted'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



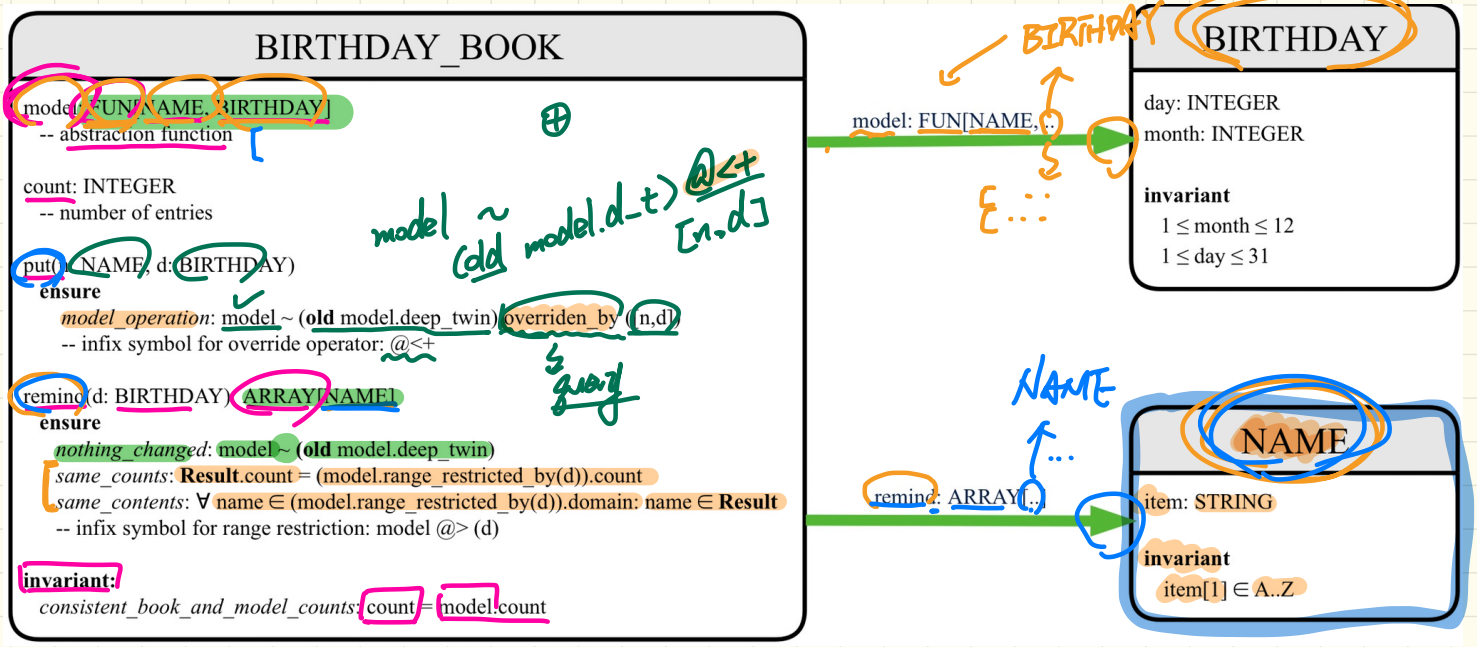
Say $r = \{(a,1),(b,2),(c,3),(a,4),(b,5),(c,6),(d,1),(e,2),(f,3)\}$

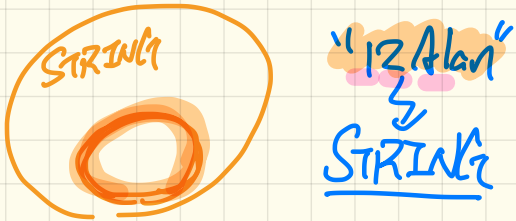
- r.*domain*: set of first-elements from *r*
  - r.**domain** = $\{ d \mid (d,r) \in r \}$
  - e.g., r.**domain** = $\{a,b,c,d,e,f\}$
- r.*range*: set of second-elements from *r*
  - r.**range** = $\{ r \mid (d,r) \in r \}$
  - e.g., r.**range** = $\{1,2,3,4,5,6\}$
- r.*inverse*: a relation like *r*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

Model of an Example Birthday Book



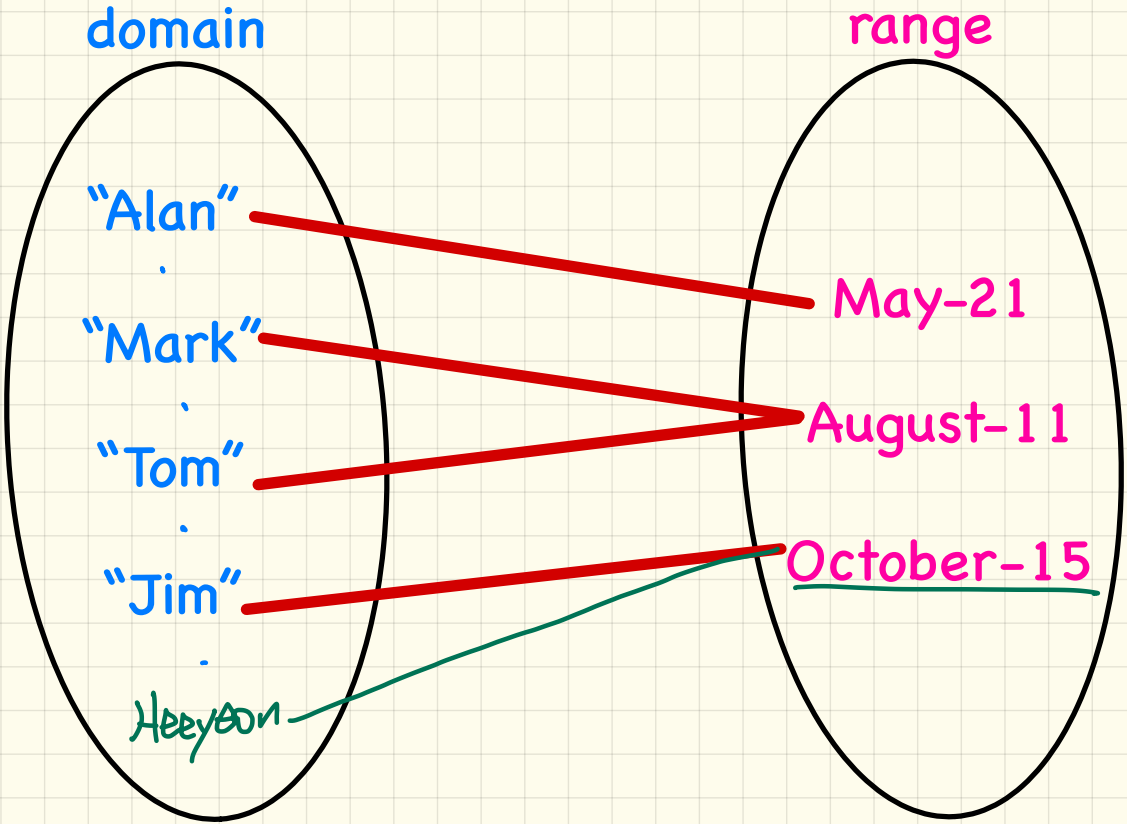
Birthday Book: Design





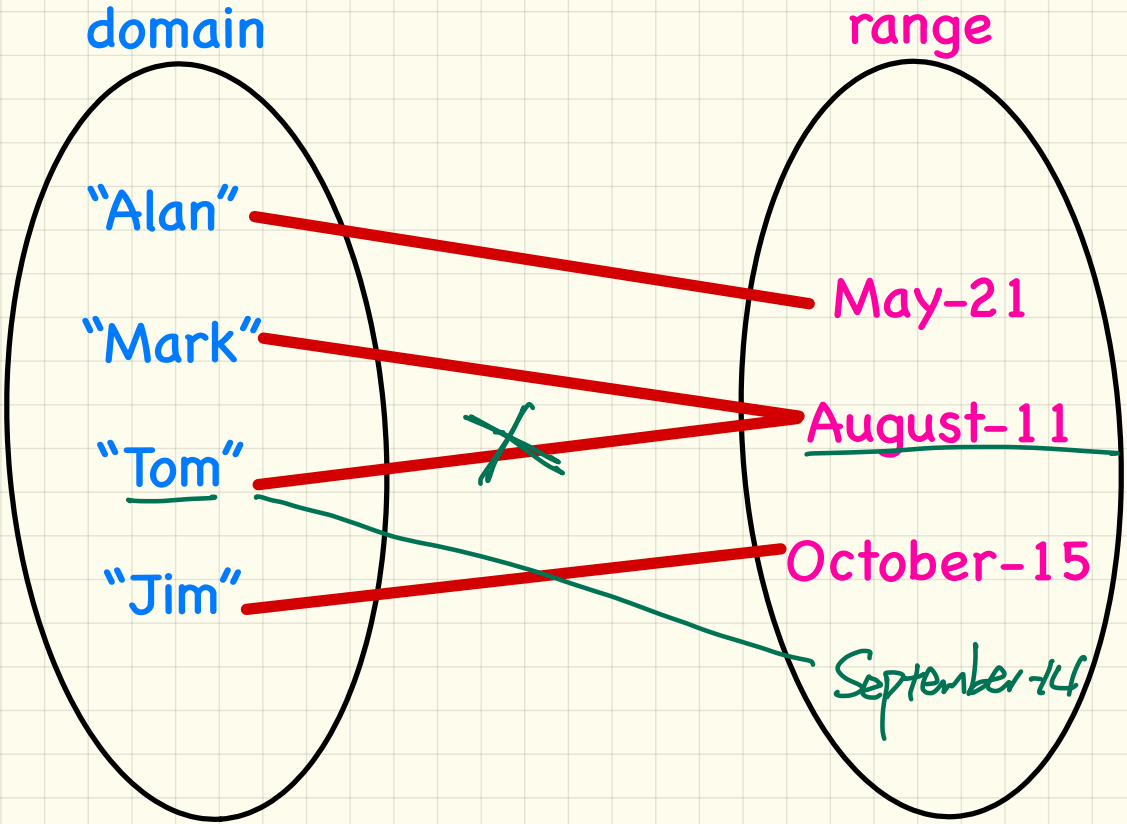
Birthday Book: Model Operation (1.1)

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



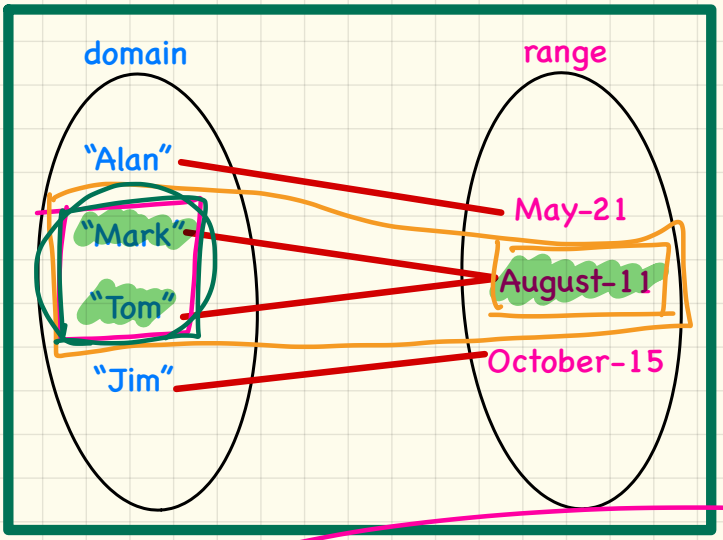
Birthday Book: Model Operation (1.2)

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



Birthday Book: Model Operation (2.1)

book.remind(August-11)



$\#S_1 = \#S_2$ ① 1

$S_1 \subseteq S_2$ ① 2

$S_1 = S_2$

↳ $S_1 \subseteq S_2$, $S_2 \subseteq S_1$

set

(model.range_restricted_by(August-11)).domain

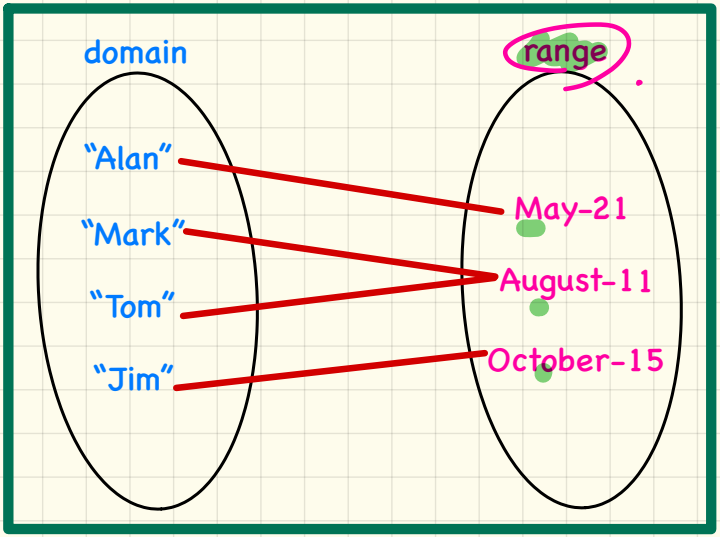
relation

~ Result Array S_2

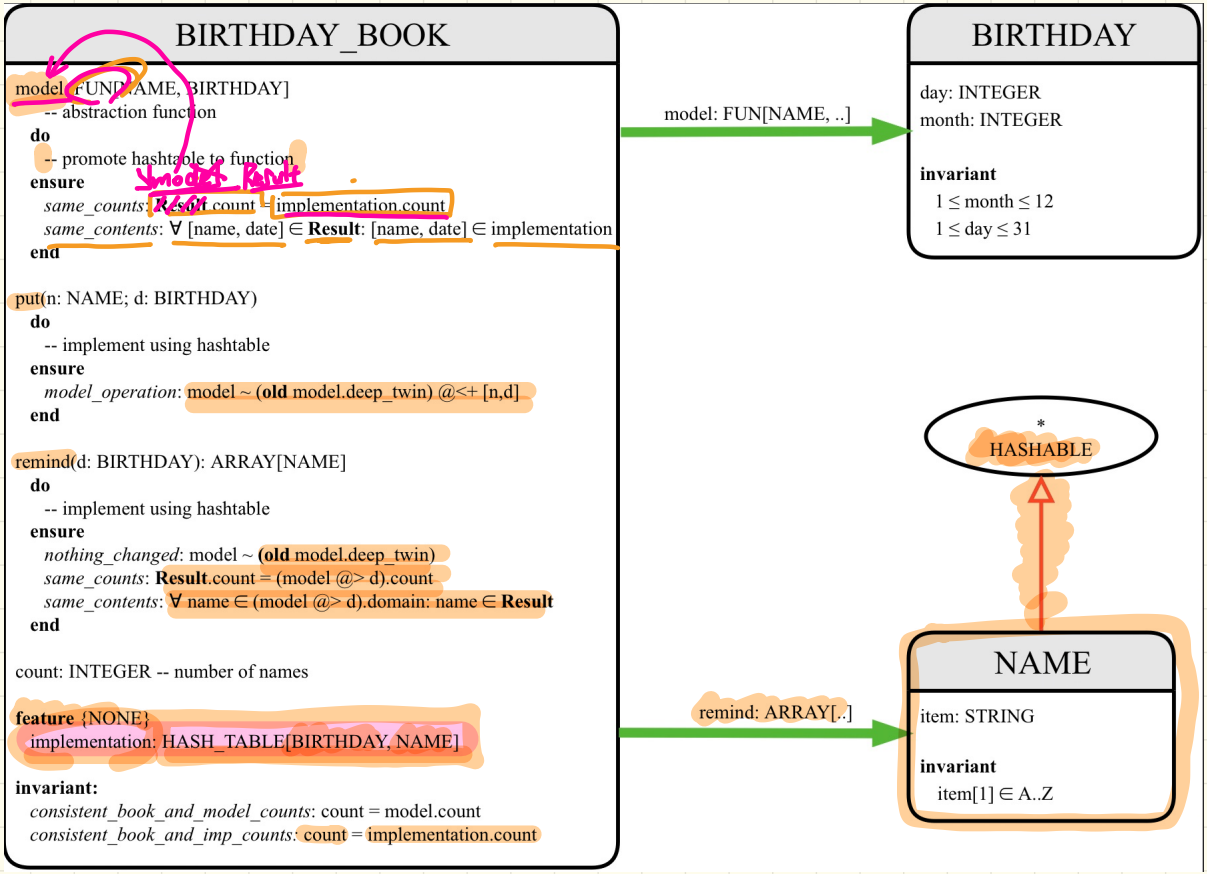
Birthday Book: Model Operation (2.2)

book.remind(November-29)

↳ empty array.



Birthday Book: Implementation



Lecture 4

Part 4

Design Pattern: Iterator

Principle of Information Hiding

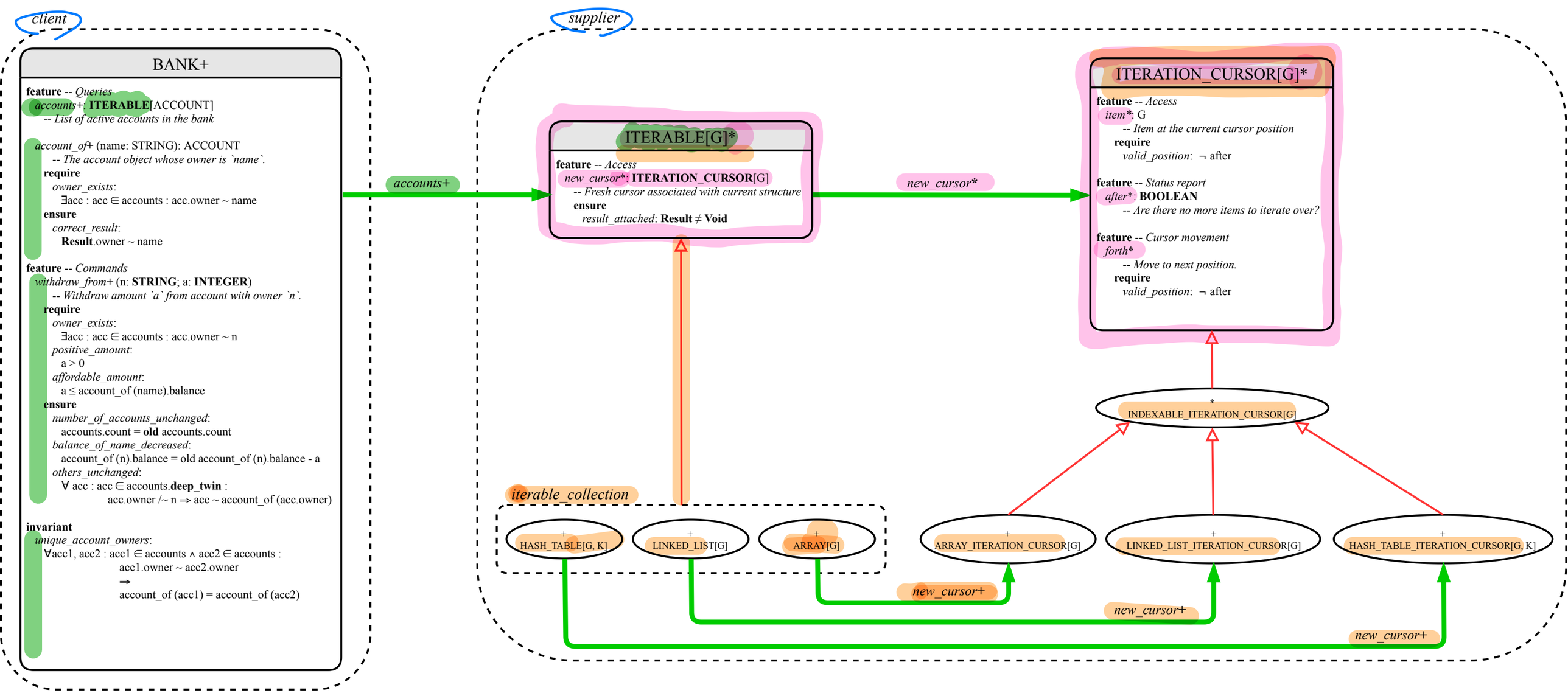
Supplier:

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

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

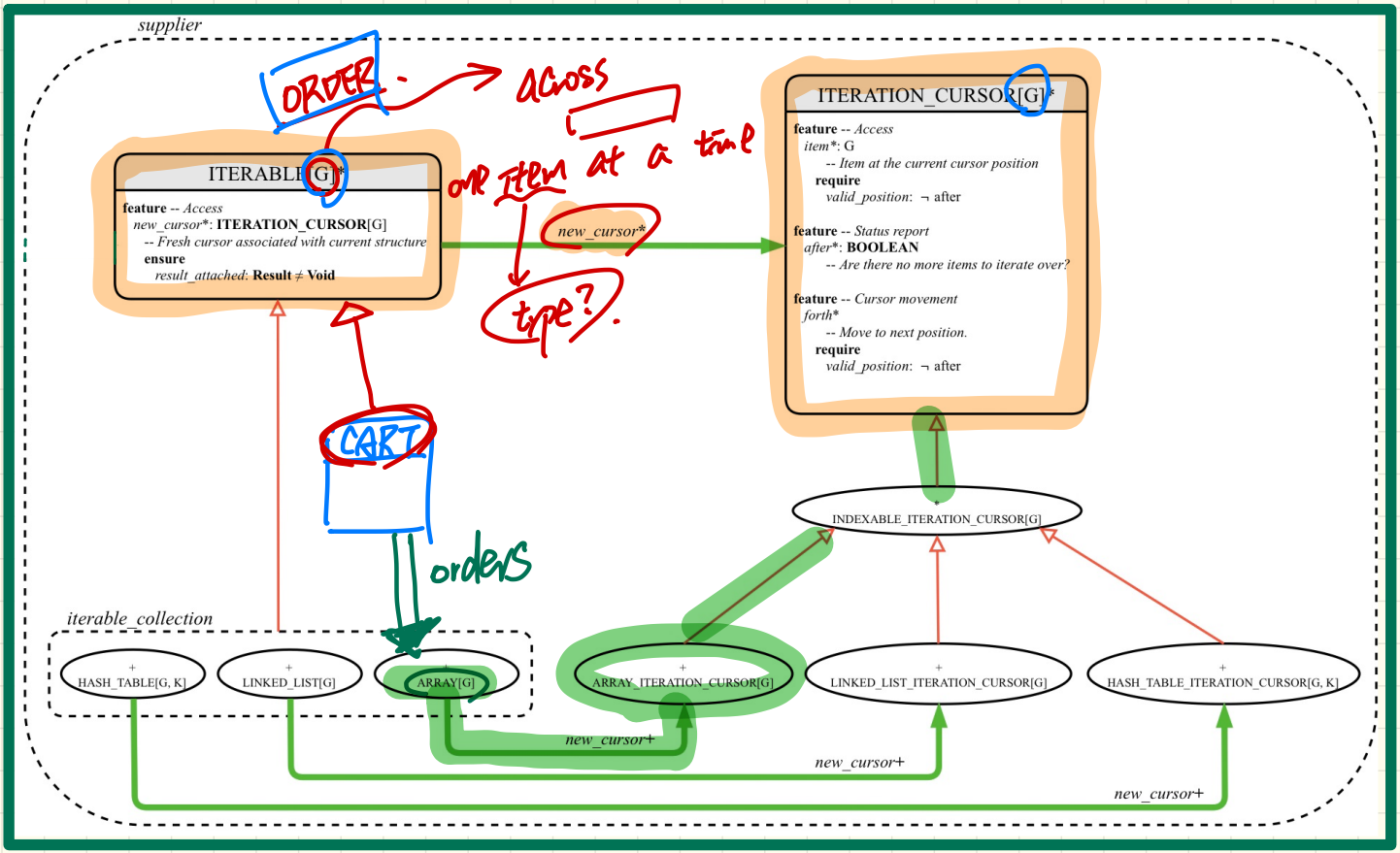
Problems?

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
```


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+

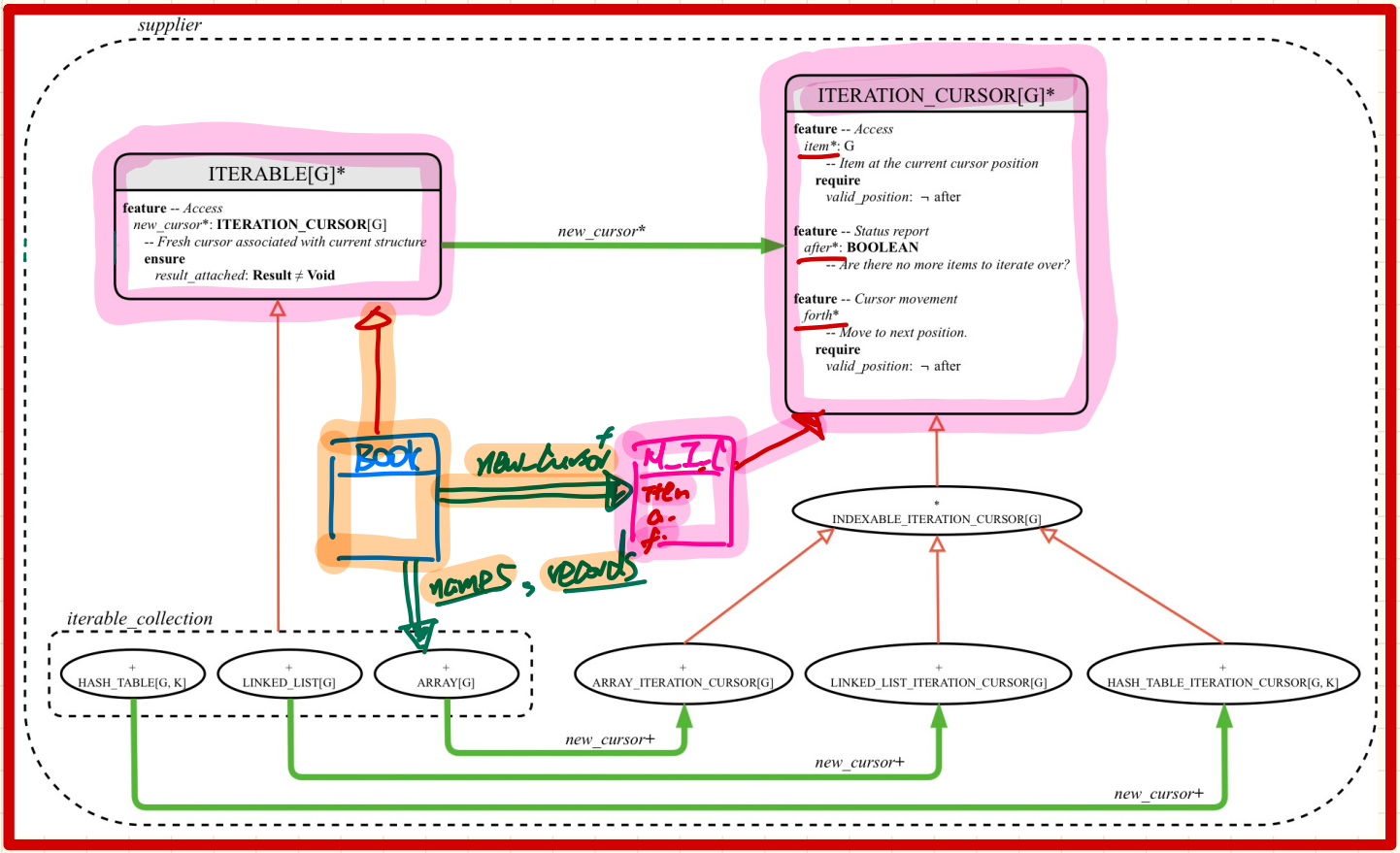
Implementing the Iterator Pattern: Easy Case



Implementing the Iterator Pattern: Easy Case

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

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

Implementing the Iterator Pattern: Hard Case



Implementing the Iterator Pattern: Hard Case

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

feature --
    new_cursor: I_C [TUPLE[S, G]]
        do
            create {M_I_C[S, G]}.
                make(...)
        end
end
```

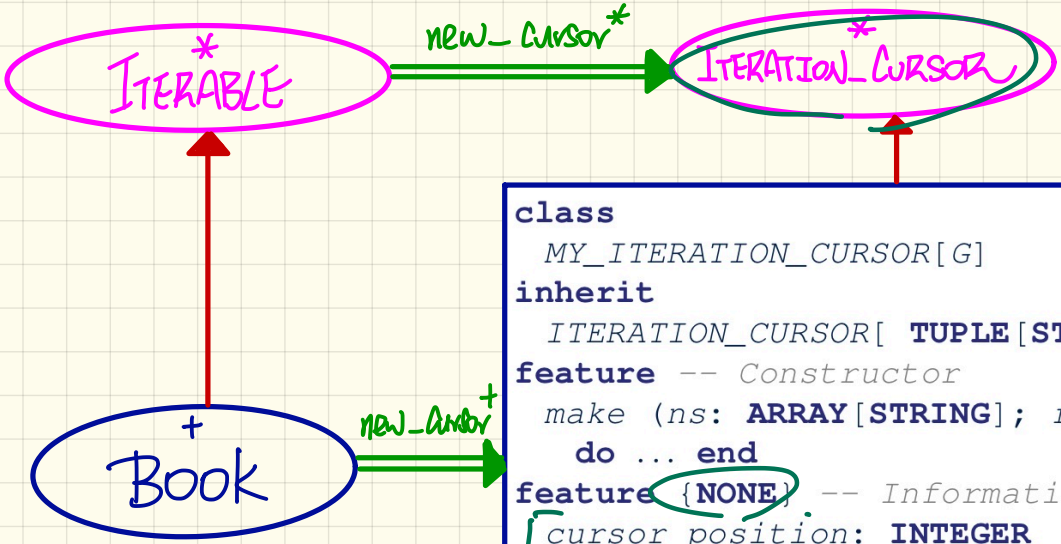
data

across bb

["alan", oct-10]

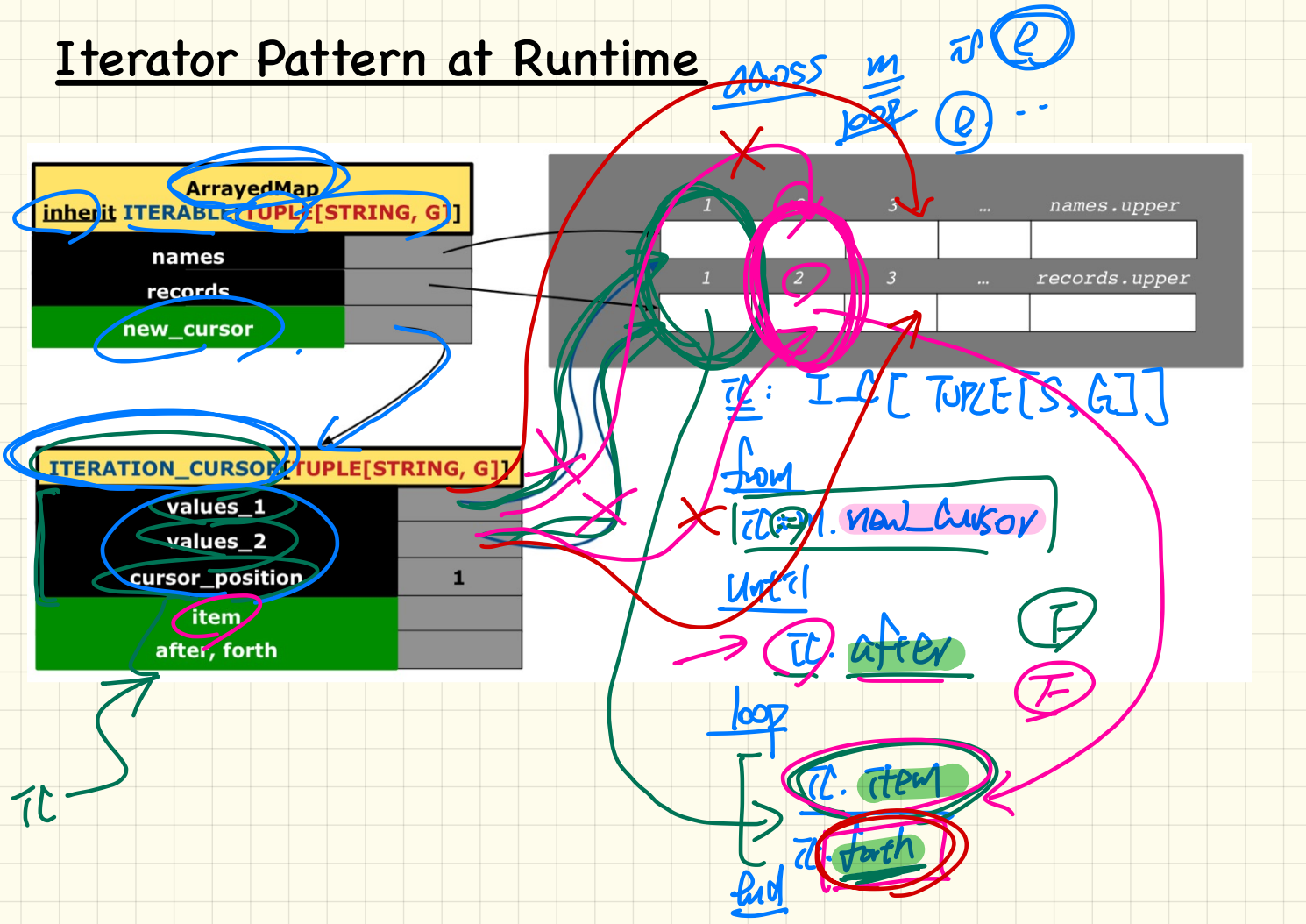
TUPLE[_, _]

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



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
```

```
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
```

Use of Iterable in Contracts: Exercise

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

across

Use of Iterable in Implementation (1)

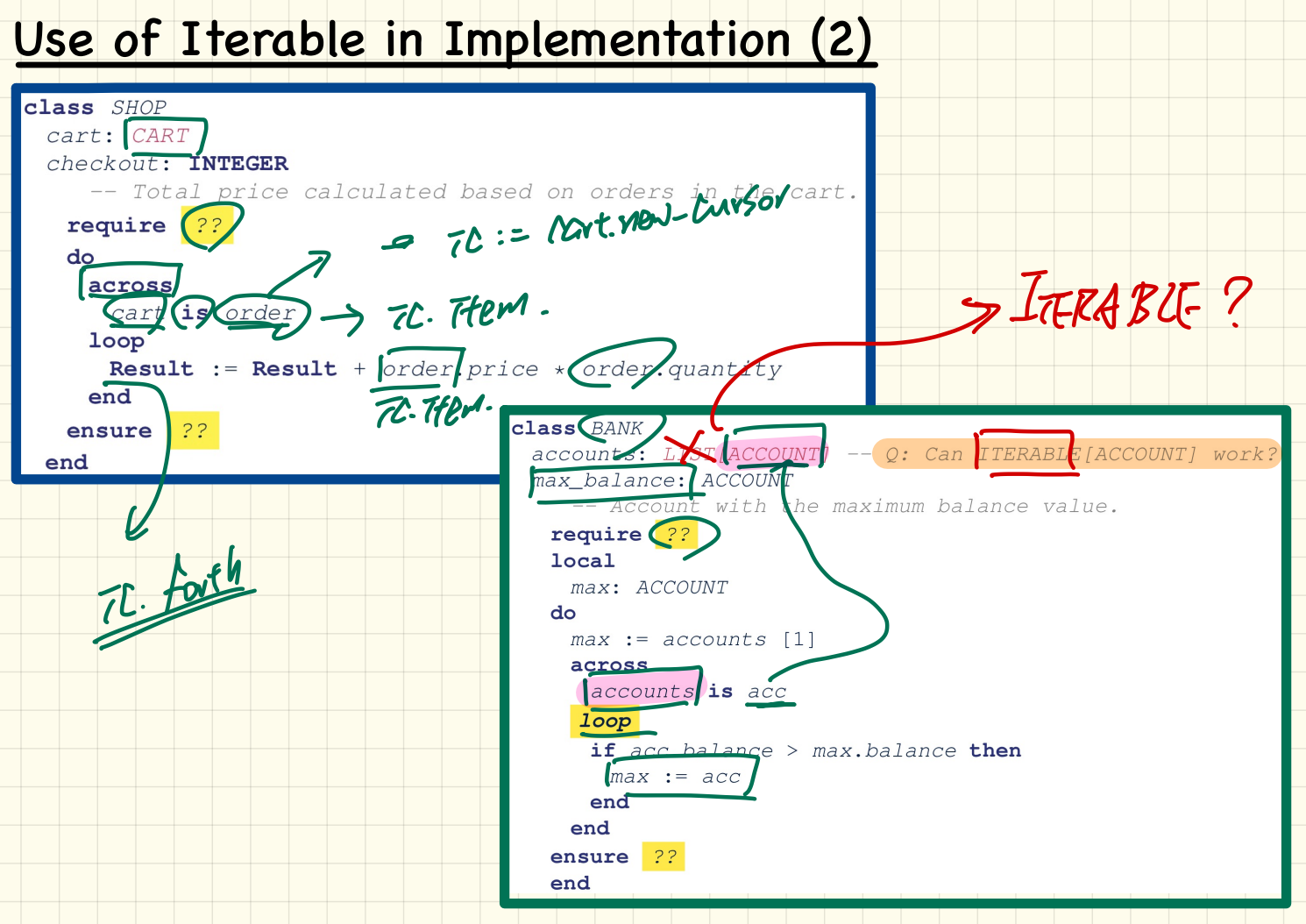
```
class BANK
  accounts: ITERABLE [ACCOUNT]
  max_balance: ACCOUNT
      -- Account with the maximum balance value.
    require ??
    local
      cursor: ITERATION_CURSOR[ACCOUNT]; max: ACCOUNT
    do
      from cursor := accounts.new_cursor; max := cursor.item
      until cursor.after
      loop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

Use from-until loop if you wish to work with an I_C.

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
```

```
class BANK
  accounts: LIST[ACCOUNT]  -- Q: Can ITERABLE[ACCOUNT] work?
  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
```



Lecture 4

Part 5

Exercise: Generics in Iterator

Exercise

```
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
```

```
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+

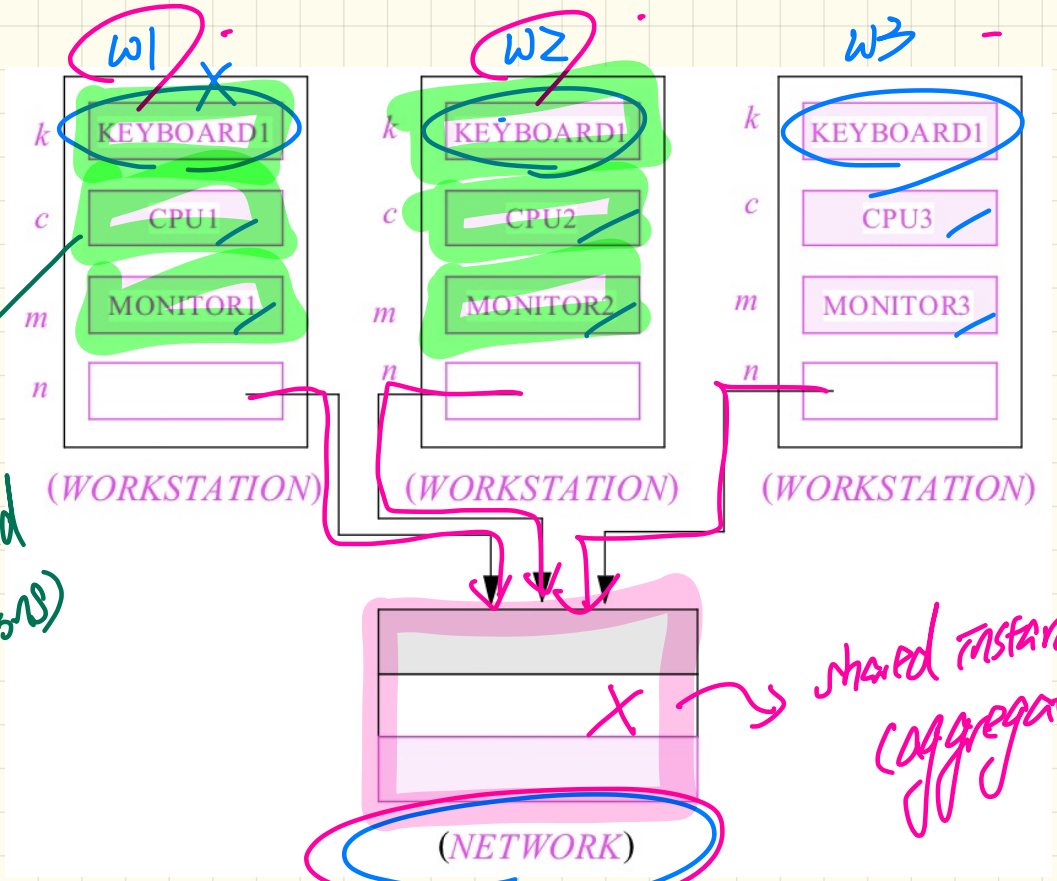
```
class
  ITEM_ITERATION_CURSOR[M, N]
inherit
  ITERATION_CURSOR[                ]
create
  make
feature {NONE} -- Implementation
  ms: ARRAY[M]
  ns: ARRAY[N]
feature -- Constructor
  make (new_ns: ARRAY[N]; new_ms: ARRAY[M])
    do ... end
feature -- Cursor features
  item:
    do ... end

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```

Lecture 5

Part 1

Expanded Types for Compositions

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
```

Use this if the keyboard class may be either an integral or sharing part.

Use this if keyboard should always be an integral part.

```
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
```

not storing address of a keyboard object

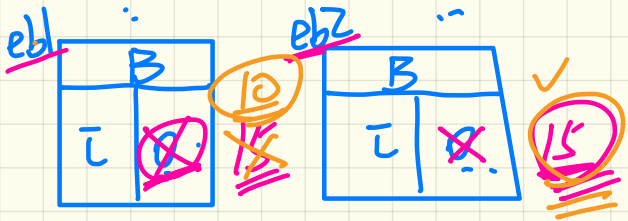
Use of **Expanded** Type

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

```
1  test_expanded
2    local
3      eb1, eb2: B
4    do
5      check eb1.i = 0 and eb2.i = 0 end
6      check eb1 = eb2 end
7      eb2.change_i (15)
8      check eb1.i = 0 and eb2.i = 15 end
9      check eb1 /= eb2 end
10     eb1 := eb2
11     check eb1.i = 15 and eb2.i = 15 end
12     eb1.change_i (10)
13     check eb1.i = 10 and eb2.i = 15 end
14     check eb1 /= eb2 end
15   end
```

not comparing addresses
eb1 ~ eb2

Void object call?



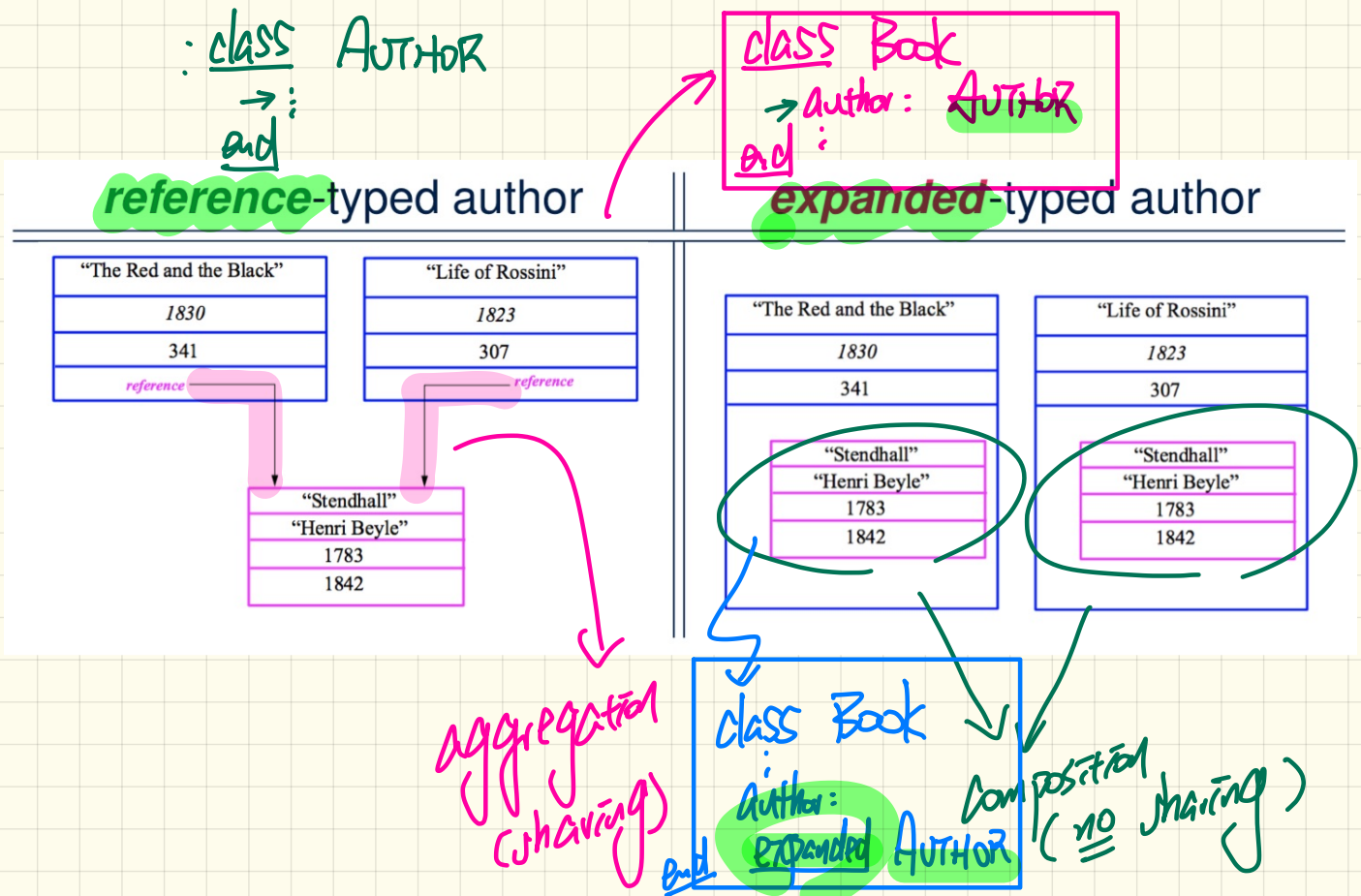
does not copy address to result in aliasing
simply copy over contents

EXERCISE

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

```
1   test_expanded
2    local
3     eb1, eb2: B
4    do
5     check eb1.i = 0 and eb2.i = 0 end
6     check eb1 = eb2 end
7     eb2.change_i (15)
8     check eb1.i = 0 and eb2.i = 15 end
9     check eb1 /= eb2 end
10    eb1 := eb2
11    check eb1.i = 15 and eb2.i = 15 end
12    eb1.change_i (10)
13    check eb1.i = 10 and eb2.i = 15 end
14    check eb1 /= eb2 end
15   end
```

Reference Type or Expanded Type



Lecture 5

Part 2

Sharing via Inheritance

Shared Data via Inheritance

Cohesion → CD (S) and Single Choice Principle

A class has a unifying theme of purpose (all features are relevant w.r.t. that purpose)

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
```

inherited to DEPOSIT but irrelevant ⇒ violation of cohesion.

Ancestor:

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

Problems?

Shared Data via Inheritance

Cohesion

Single Choice Principle

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

d1 ~> | | DEPOSIT |
|---|---|
| ir | 0.11 |
| er | 2.34 |
| min | 1000 ~~500~~ |
| max | 1000000 |

2M

d2 ~> | | DEPOSIT |
|---|---|
| ir | 0.11 |
| er | 2.34 |
| min | 1000 ~~500~~ |
| max | 1000000 |

2M

w1 ~> | | WITHDRAW |
|---|---|
| ir | 0.11 |
| er | 2.34 |
| min | 1000 ~~500~~ |
| max | 1000000 |

2M

w2 ~> | | WITHDRAW |
|---|---|
| ir | 0.11 |
| er | 2.34 |
| min | 1000 ~~500~~ |
| max | 1000000 |

2M

t1 ~> | | TRANSFER |
|---|---|
| ir | 0.11 |
| er | 2.34 |
| min | 1000 ~~500~~ |
| max | 1000000 |

2M

t2 ~> | | TRANSFER |
|---|---|
| ir | 0.11 |
| er | 2.34 |
| min | 1000 ~~500~~ |
| max | 1000000 |

2M

d1, d2: **DEPOSIT**
w1, w2: **WITHDRAW**
t1, t2: **INTERNATIONAL_TRANSFER**

...

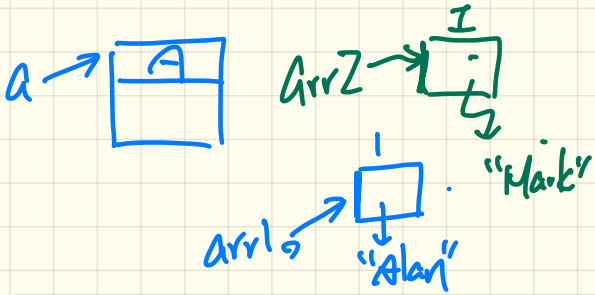
d1.set_max_balance (2M)
w2.set_min_balance (500)
t2.set_exchange_rate

Lecture 5

Part 3

Once Routines

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
```

"Alan" "Mark"

```
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
```

1st call

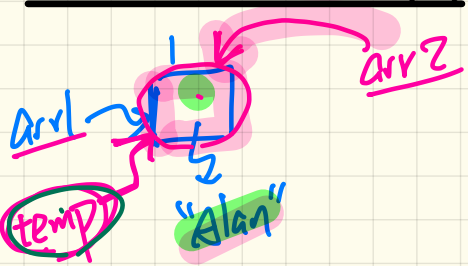
2nd call

F

T

address

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
```

"Alan" "Mark"

```
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
```

1st call.

subsequent call.

✓

Once Routine

1. The very first call
2. All subsequent calls.

Approximating Once Routines in Java (1)

public

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

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

Problem. sharing is not guaranteed.

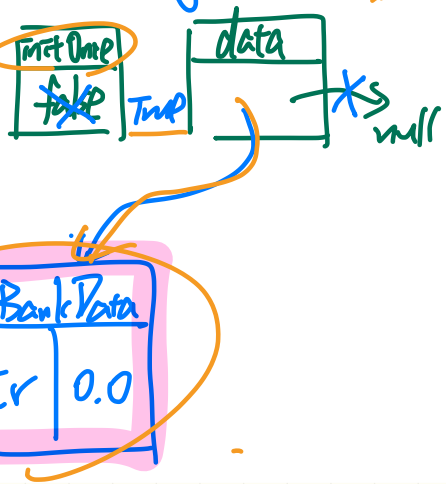
```
class Account {
  BankData data;
  Account() {
    data = BankDataAccess.getData();
  }
}
```

1st call

2nd call

data = BankDataAccess.getData();

Problem?



Approximating Once Routines in Java (2)

We may encode Eiffel once routines in Java:

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

Problem?

Violating cohesion!

Lecture 5

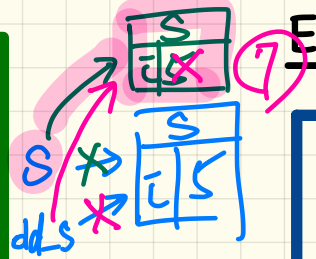
Part 4

Export Status

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
    make

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

feature
    i: INTEGER
end
```

Export Status Case 2

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

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

```
class SUPPLIER

create {CLIENT_1}
    make

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

feature
  i: INTEGER
end
```

Export Status Case 3

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

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

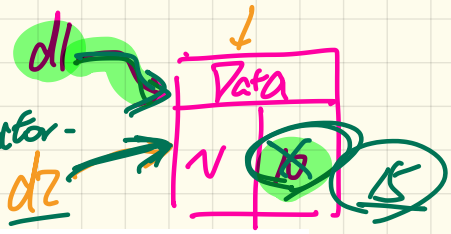
```
class SUPPLIER

create {CLIENT_3}
    make

feature {CLIENT_3}
  make (init_i: INTEGER)
    do
      i := init_i
    end

feature
  i: INTEGER
end
```

Lecture 5

Part 5

Singleton Pattern

Singleton Design Pattern: Code (1)



only DATA_ACCESS can use 'make' as a constructor.

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
```

all separate calls to 'data' retrieve the same shared data object.

one call to 'data'

a separate call to 'data'

Client:

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

create {DATA} d1. ✗ make.

1st call

2nd call

T

T

T

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

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
```

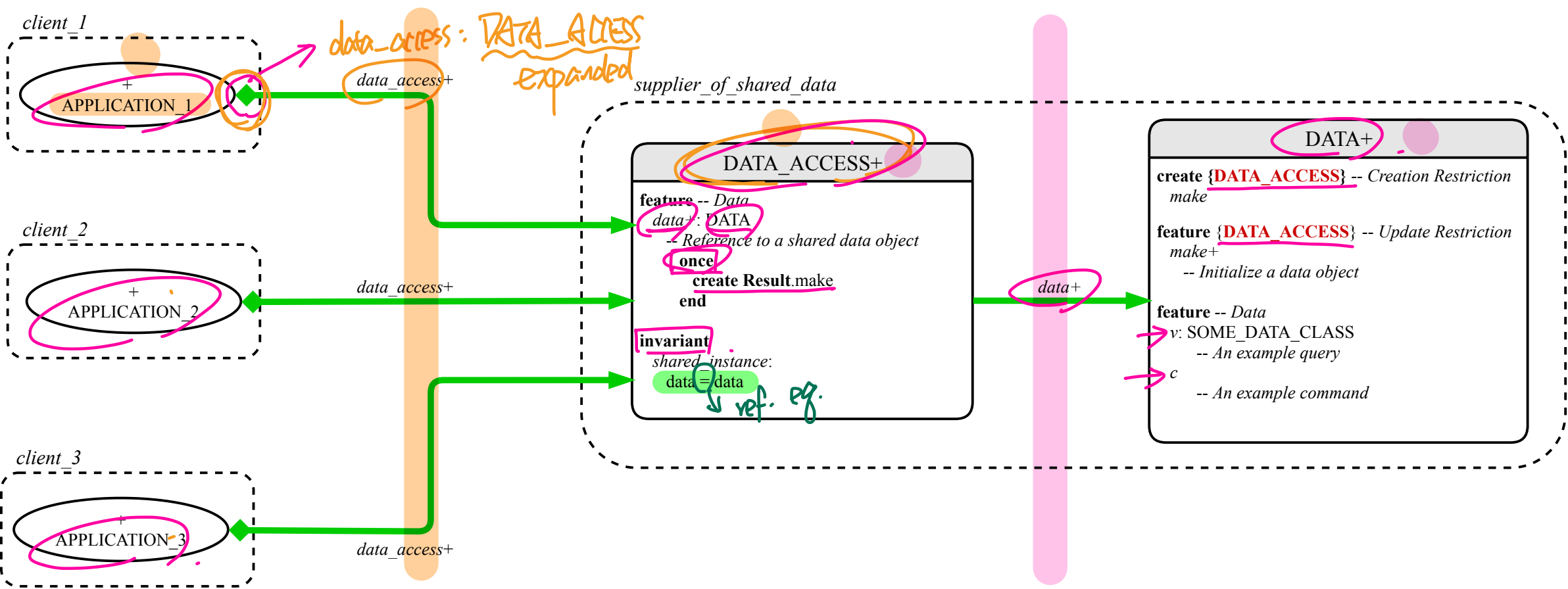
Writing **create** *data.make* in client's `make` feature does not compile. Why?

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
```

triggers the 1st call to {D_A3}.data

2nd call to {D_A3}.data


client_1
APPLICATION_1
data_access+
data_access: DATA_ACCESS
expanded
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
ref. eg.
client_2
APPLICATION_2
data_access+
client_3
APPLICATION_3
data_access+
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

Lecture 6

Part 1

Abstract UI

ETF: Abstract UI vs. Concrete UI

Concrete UI Interaction

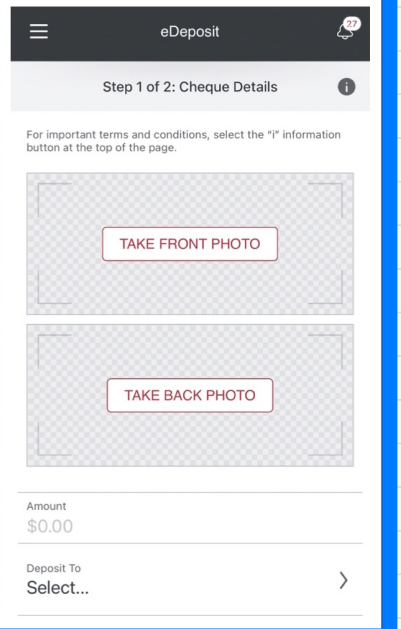
- Insert a bank card
- Validate password
- Select Deposit Transaction
- Select cheque account
- Enter amount
- Select confirm

abstracted

abstract UI

```
system bank

new(id: STRING)
  -- create a new bank account for "id"
deposit(id: STRING; amount: INTEGER)
  -- deposit "amount" into the account of "id"
withdraw(id: STRING; amount: INTEGER)
  -- withdraw "amount" from the account of "id"
transfer(id1: STRING; id2: STRING; amount: INTEGER)
  -- transfer "amount" from "id1" to "id2"
```

Concrete UI

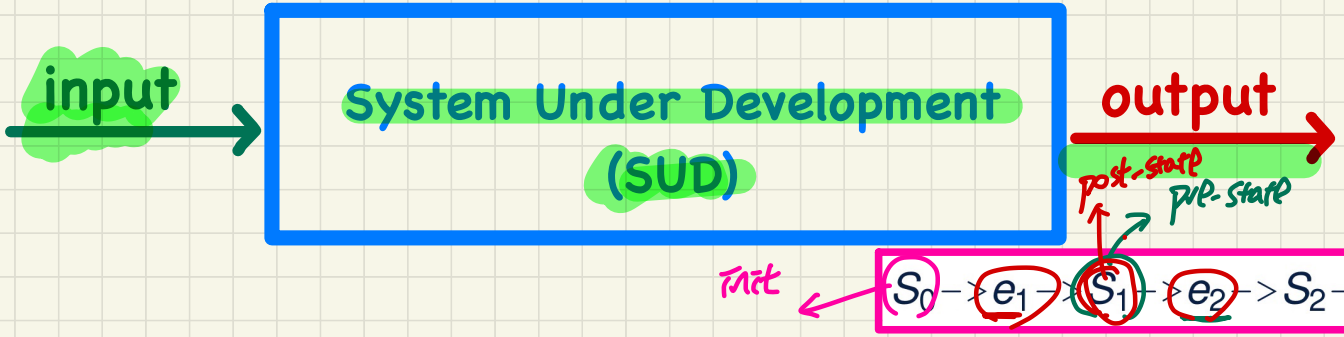


Lecture 6

Part 2

Abstract States, Acceptance Tests

Acceptance Testing




```
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}
```

Acceptance Test vs. Unit Test

```
{}
->new("alan")
 {alan: 0}
->deposit("alan", 200)
 {alan: 200}
```

abstract states

abstract events (abstract UI)

```
test: BOOLEAN
  local acc: ACCOUNT
  do create acc.make("alan")
     acc.add(200)
     Result := acc.balance = 200
  end
```

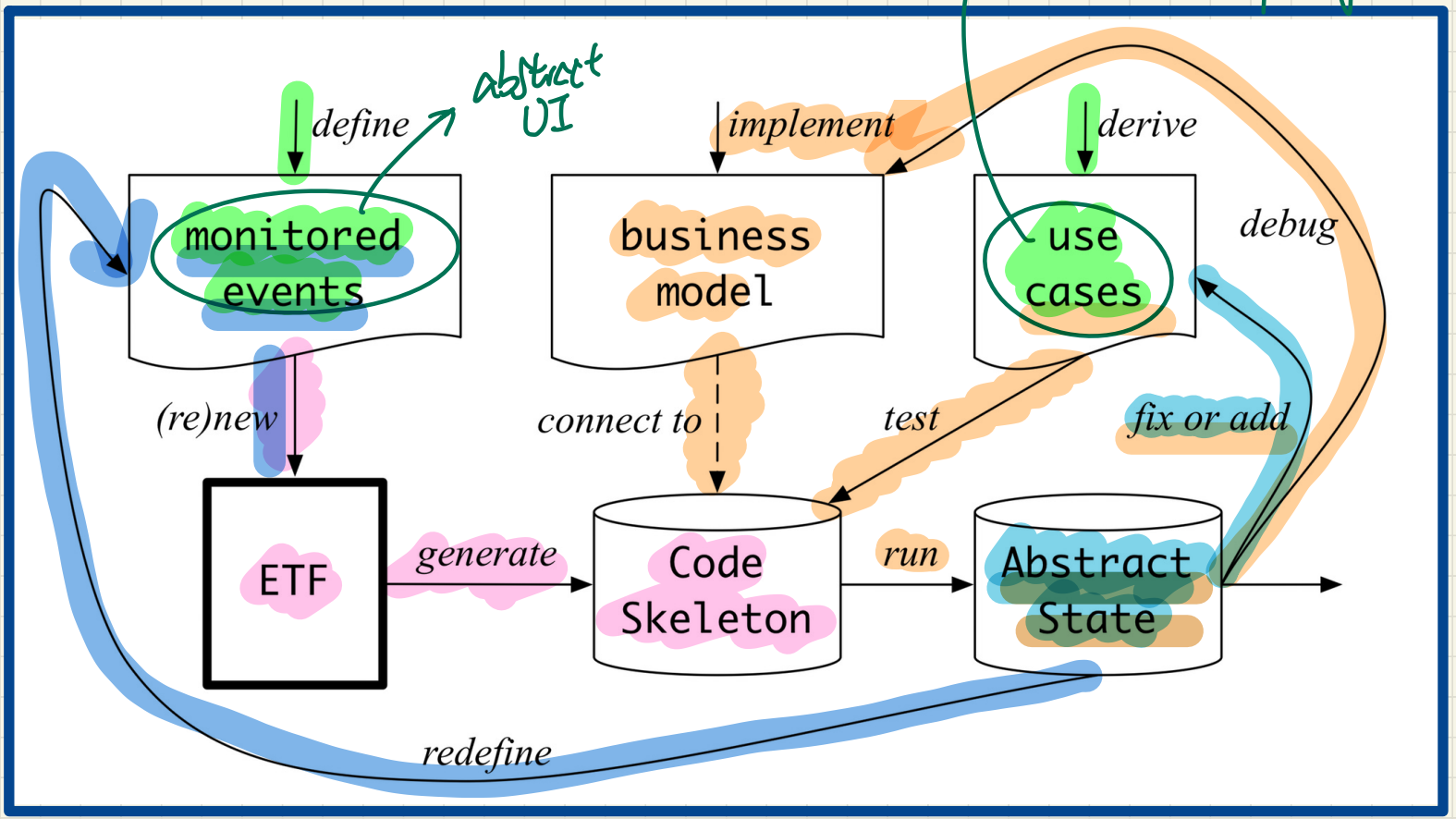
implementation details

Lecture 6

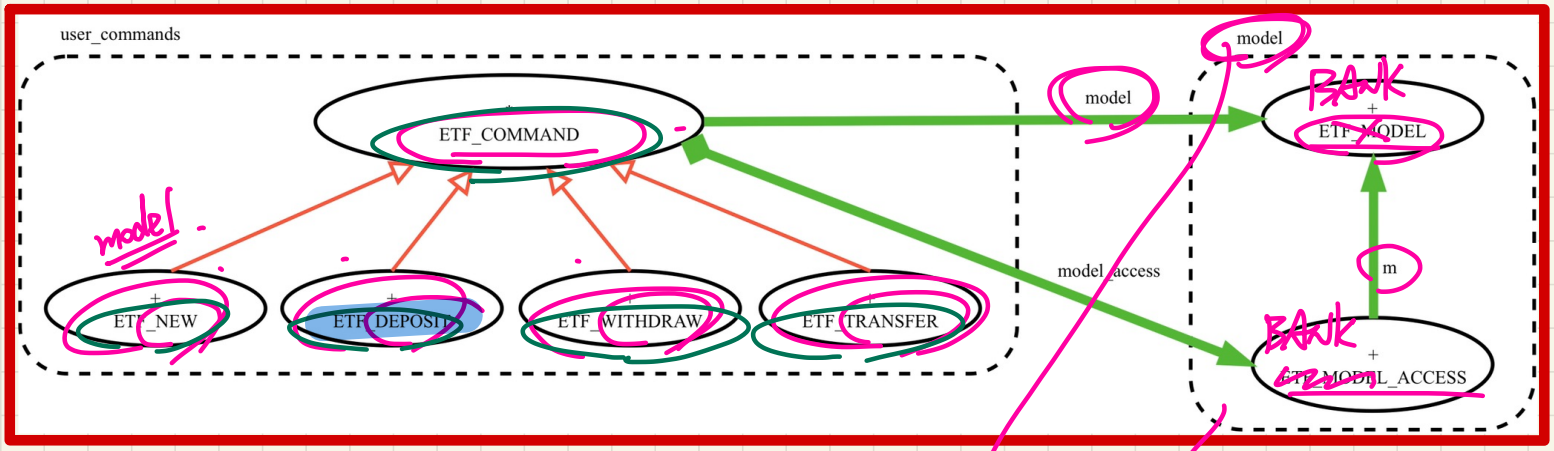
Part 3

ETF: Abstract UI, Model, Regression Tests

ETF: Workflow



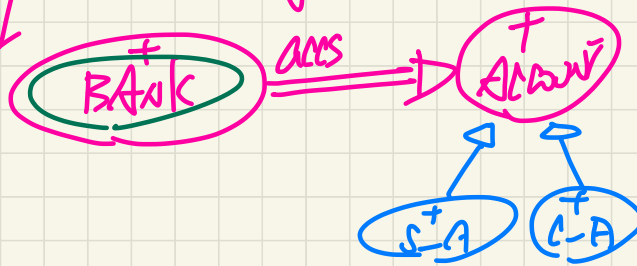
ETF: Separating User Interface and Business Model



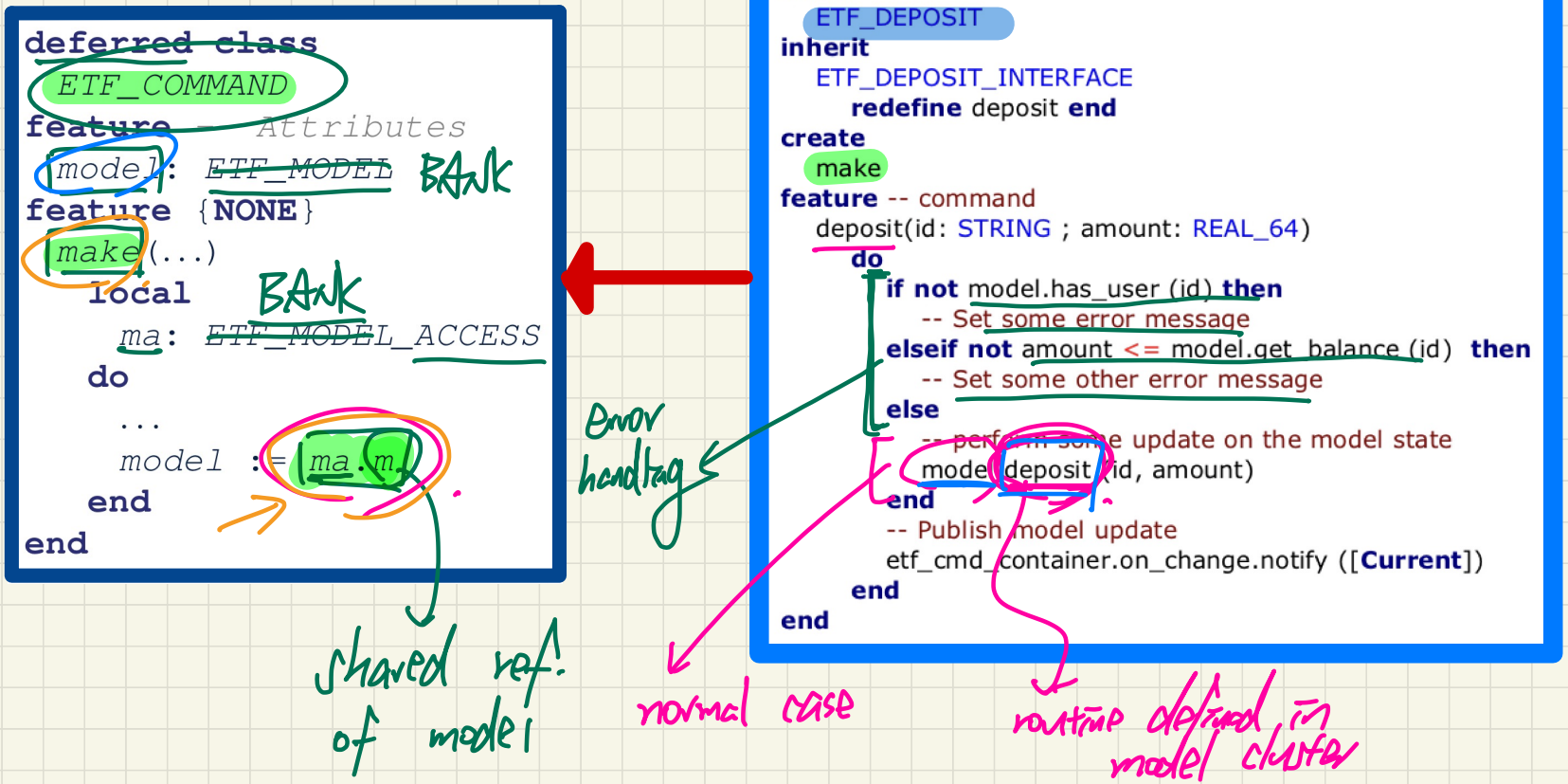
abstract UI

```
system bank

new(id: STRING)
  -- create a new bank account for "id"
deposit(id: STRING; amount: INTEGER)
  -- deposit "amount" into the account of "id"
withdraw(id: STRING; amount: INTEGER)
  -- withdraw "amount" from the account of "id"
transfer(id1: STRING; id2: STRING; amount: INTEGER)
  -- transfer "amount" from "id1" to "id2"
```

modularity

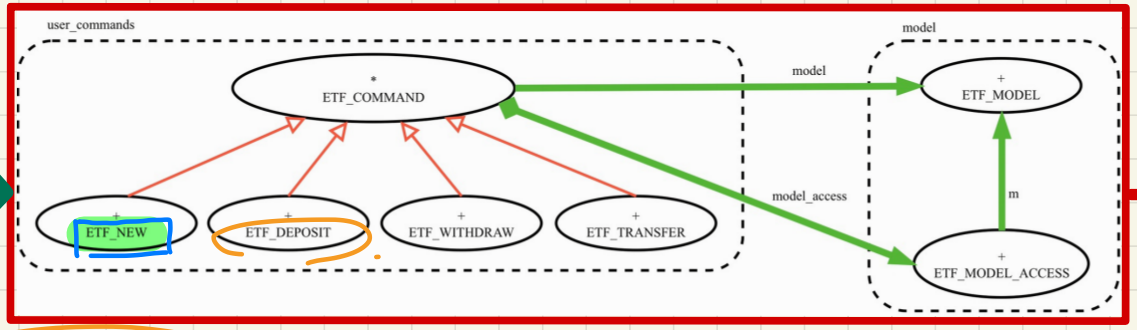


ETF: Singleton Pattern



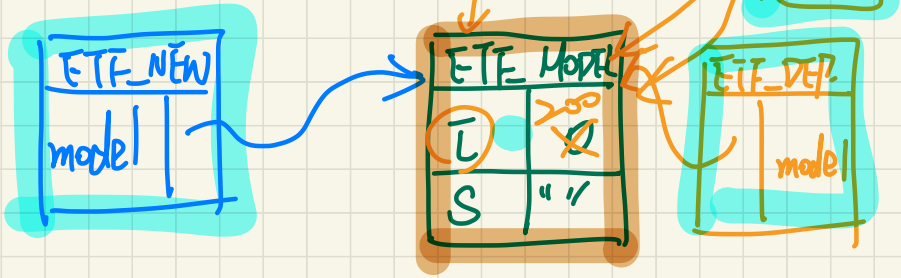
ETF: Input-Output-Based Acceptance Testing

input



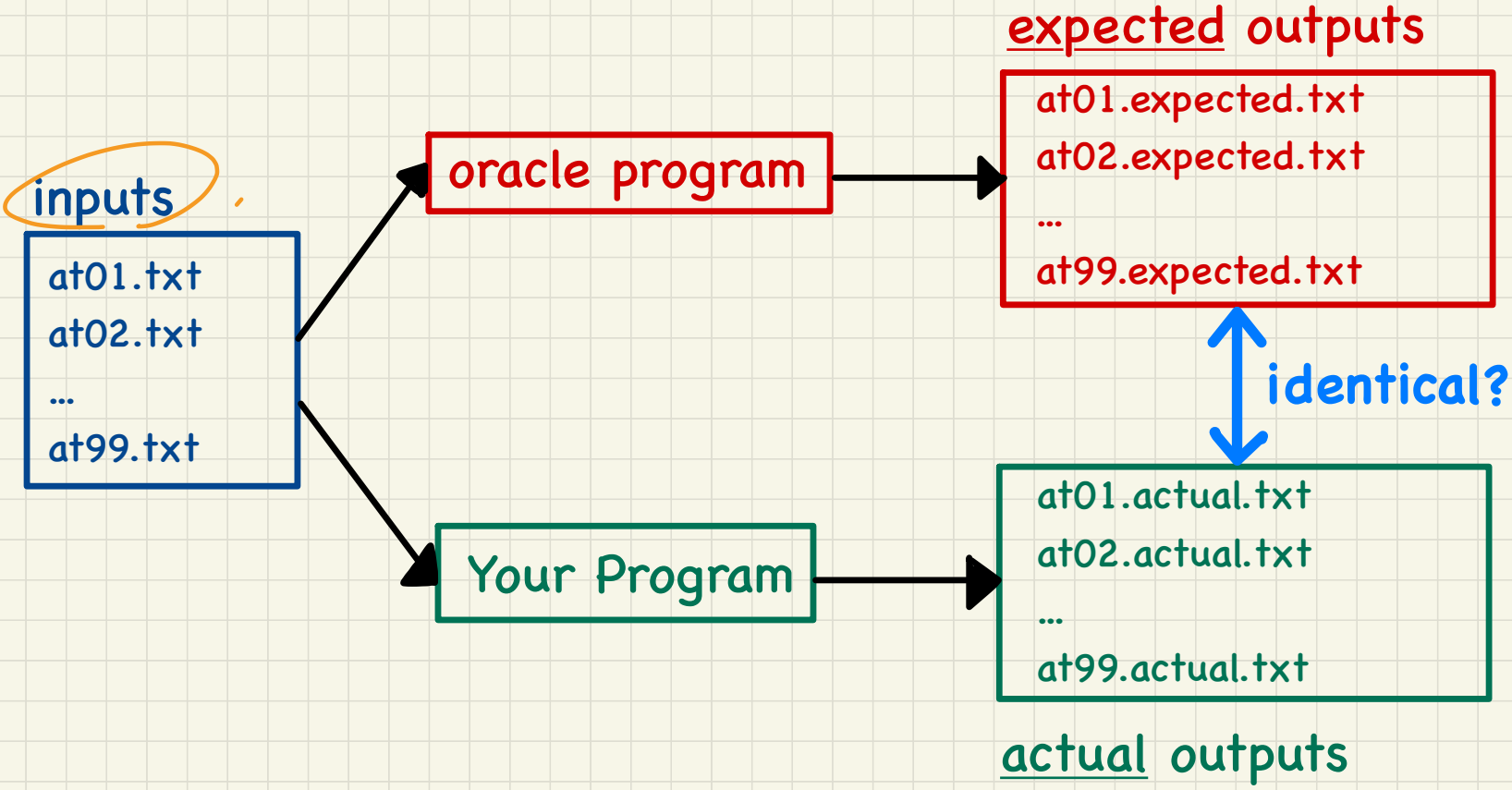
output

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

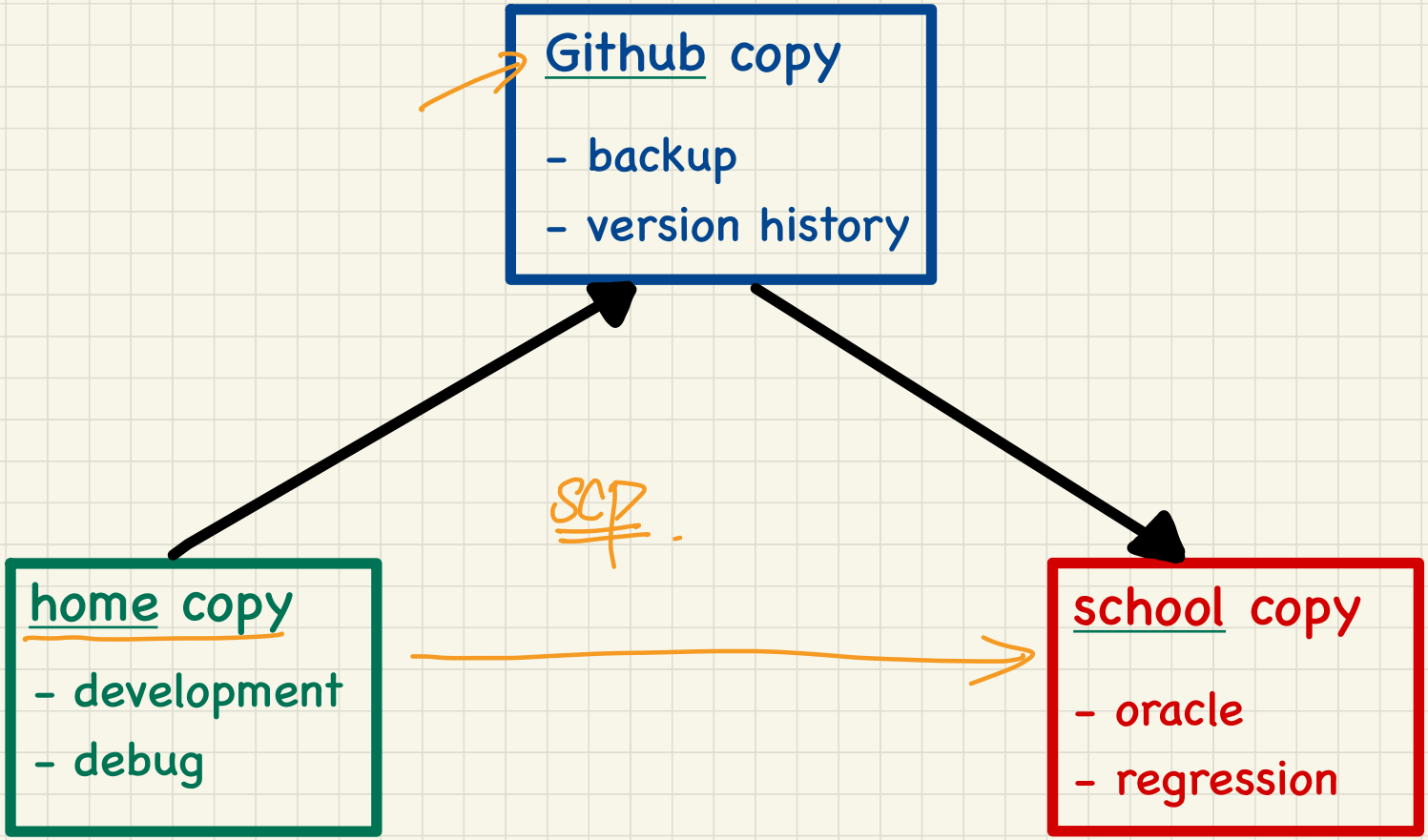



```
   {}
-> new("alan")
   {alan: 0}
-> new("mark")
   {alan: 0, mark: 0}
-> deposit("alan", 200)
   {alan: 200, mark: 0}
-> deposit("mark", 100)
    {alan: 200, mark: 100}
-> transfer("alan", "mark", 50)
    {alan: 150, mark: 150}
```

Regression Testing



Automating Regression Testing



Lecture 7

Part 1

A Motivating Problem

Inheritance: Motivating Problem

Nouns -> classes, attributes, accessors (queries)

Verbs -> mutators (command)

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.

Lecture 7

Part 2

1st Design Attempt without Inheritance

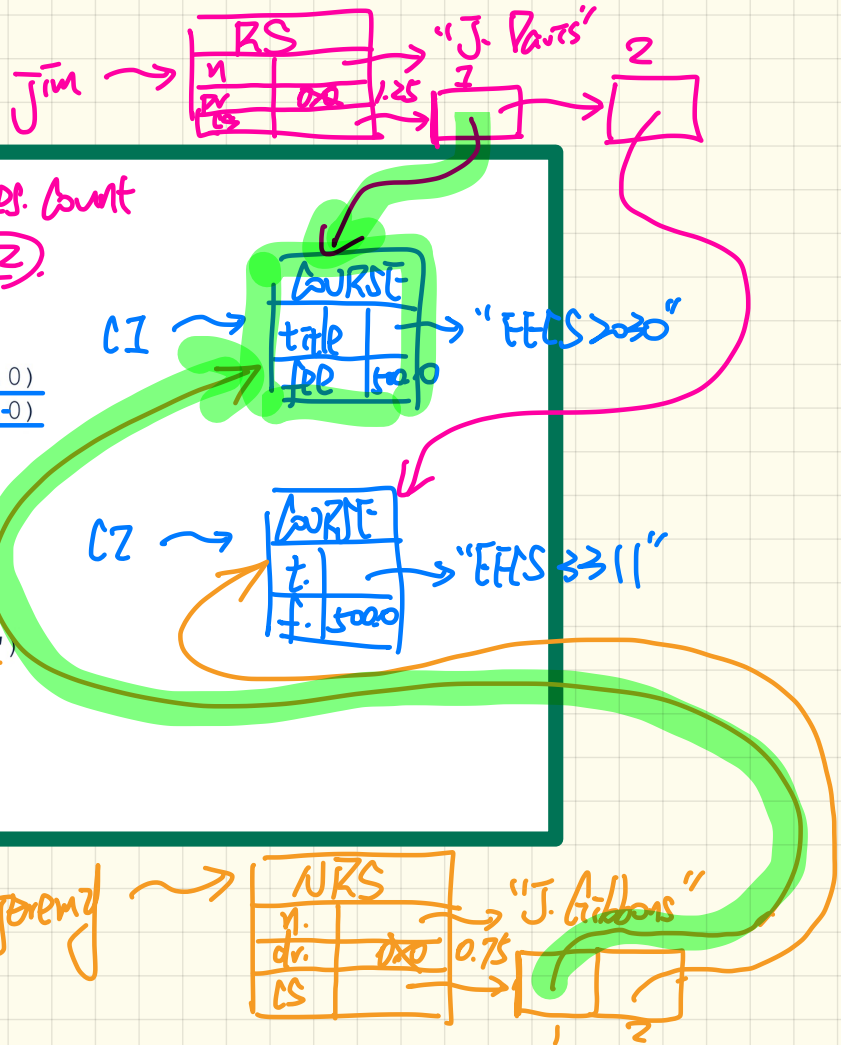
1st Design Attempt

```
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 Test

```
test_students: BOOLEAN
  local
    c1, c2: COURSE
    jim: RESIDENT_STUDENT
    jeremy: NON_RESIDENT_STUDENT
  do
    create c1.make ("EECS2030", 500.0)
    create c2.make ("EECS3311", 500.0)
    create jim.make ("J. Davis")
    jim.set_pr (1.25)
    jim.register (c1)
    jim.register (c2)
    Result := jim.tuition = 1250
    check Result end
    create jeremy.make ("J. Gibbons")
    jeremy.set_dr (0.75)
    jeremy.register (c1)
    jeremy.register (c2)
    Result := jeremy.tuition = 750
  end
```



1st Design Attempt

Good design?

Judge by Cohesion

↳ satisfied.

```
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 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
```

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

class I_S

n

c

i_s.r

set

r.

1st Design Attempt

Good design?

How do you build a

STUDENT_MANGEMENT_SYSTEM

class accordingly?

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

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

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
```

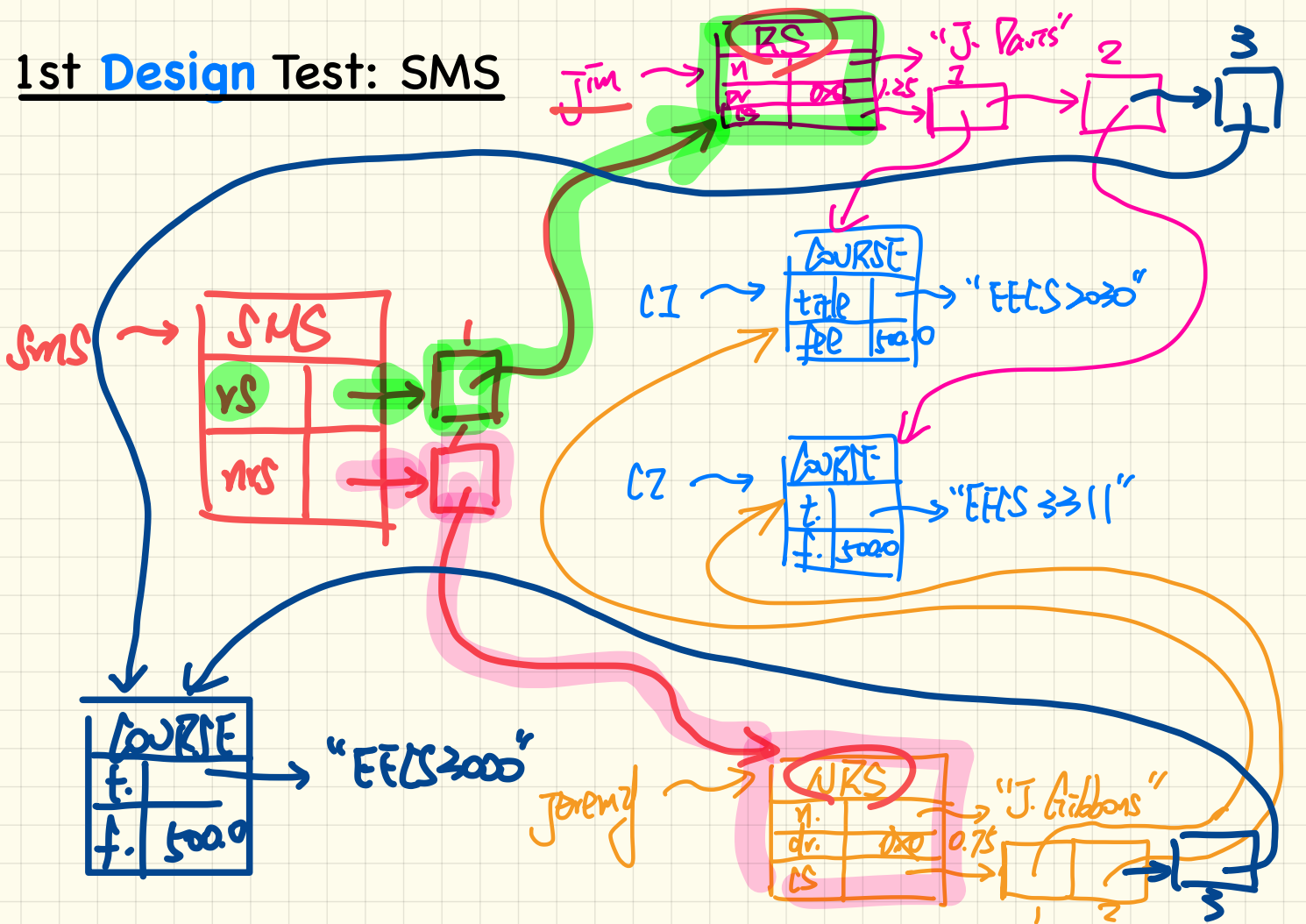
Clinet's Code

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?

1st Design Test: SMS



Lecture 7

Part 3

2nd Design Attempt without Inheritance

2nd Design Attempt

```
class
  STUDENT
create
  make
feature -- attributes
  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
```

2nd Design Attempt

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

in case kind=1 ↳ RS

in case kind=2 ↳ NRS

would not apply

would not make sense.

Good design?

Judge by **Cohesion**

```
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
```

2nd Design Attempt

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

invariant 1 <= kind <= 3

```
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
```

elseif kind = 3 then

```
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
```

elseif kind = 3 then

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

Good design?

How do you build a STUDENT_MANGEMENT_SYSTEM class accordingly?

```
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
```

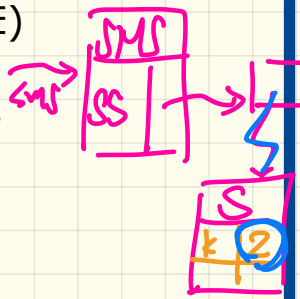
```
register (c: COURSE)
   local
      max: INTEGER
   do
      if kind = 1 then MAX := 6
      elseif kind = 2 then MAX := 4
      end
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

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
```

kind 1 2

STUDENT.



Clinet's Code

```
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)
```

TS : I-S

Create rs.make(1)

Create nrs.make(2)

Create TS.make(3)

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

Lecture 7

Part 4

Using Inheritance for Code Reuse

Desigh 3:

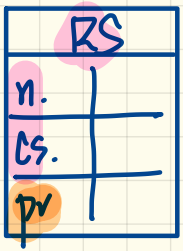
Inheritance

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

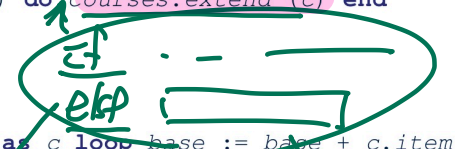
Code Reuse

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
```

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)
```

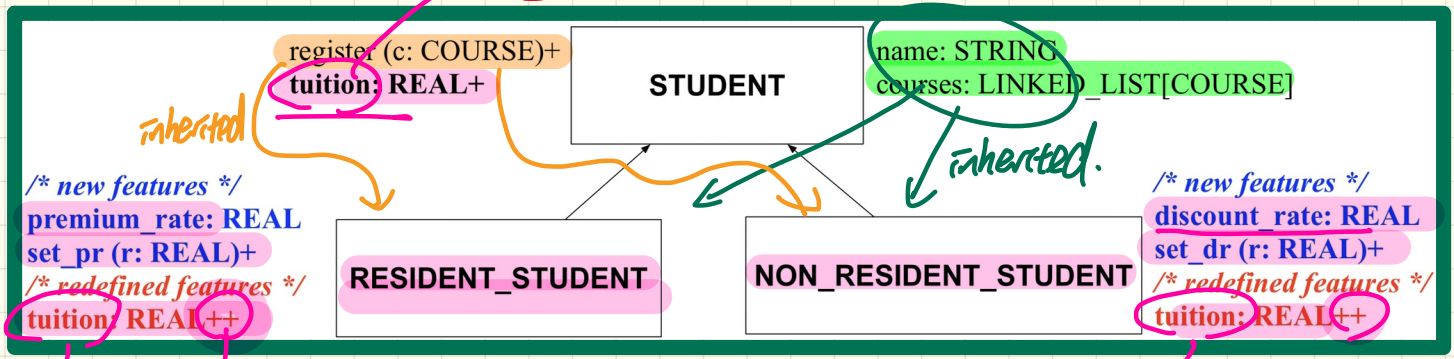
Q: What if more kinds of students are to be introduced?

Lecture 7

Part 5

Static vs. Dynamic Types

Revisit: Inheritance for Code Reuse



v1 : base rate

v2 : base rate * p_r

redefined

v3 : base rate * d_r

Static Type vs. Dynamic Type

S. S = new RS(..);
;
S = new NRS(..);

- In Java:

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

- In Eiffel:

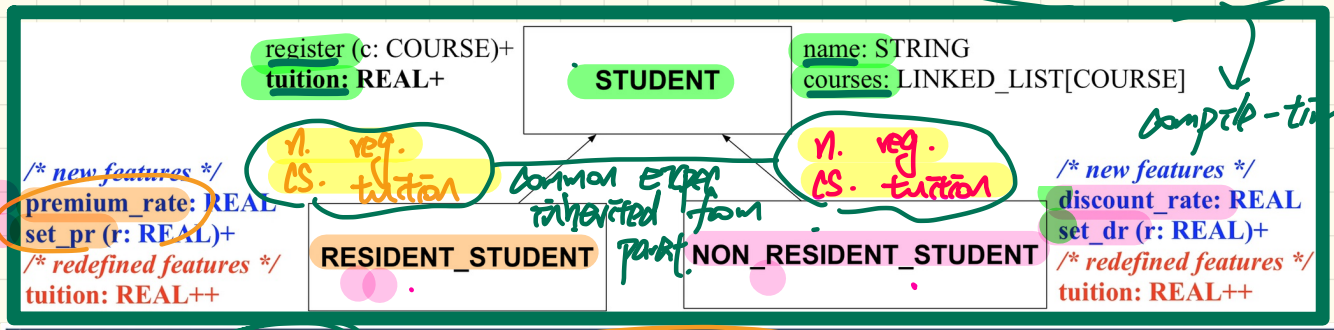
```
local s: STUDENT
      rs: STUDENT
do create {STUDENT} s.make ("Alan")
   create {RESIDENT_STUDENT} rs.make ("Mark")
```

  - 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")
```

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

Student Classes (with Inheritance): Expectations




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

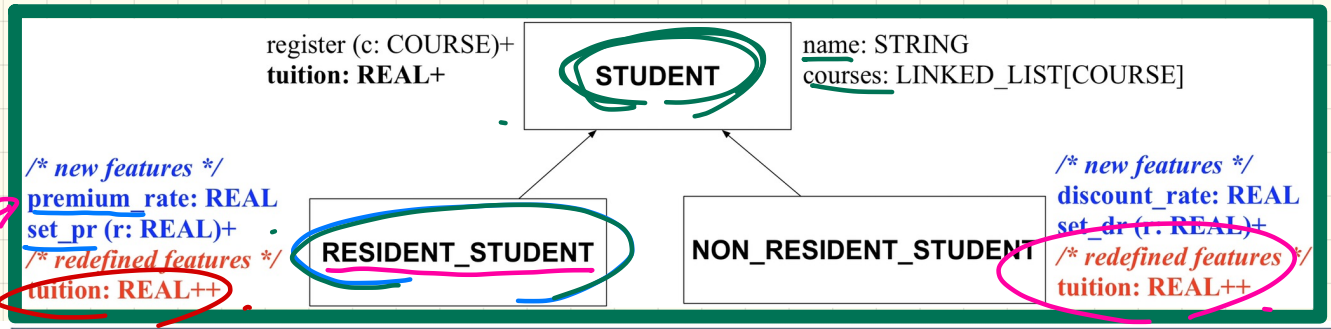
	name	courses	reg	tuition	pr	set_pr	dr	set_dr
s1	✓	✓	✓	✓	×	×	×	×
s2					×			
s3								
rs								
nrs								

Lecture 7

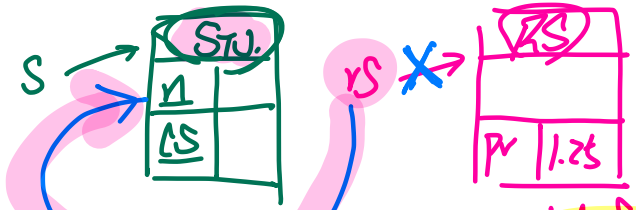
Part 6

Intuition:
Polymorphism vs. Dynamic Binding

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  X rs := s /* Is this valid? */
```



rs.pr → Crash ' not supported

not compiled. opposite should be true

Assume: rs := s Compiled.

⇒ Runtime: execute the re-assignment

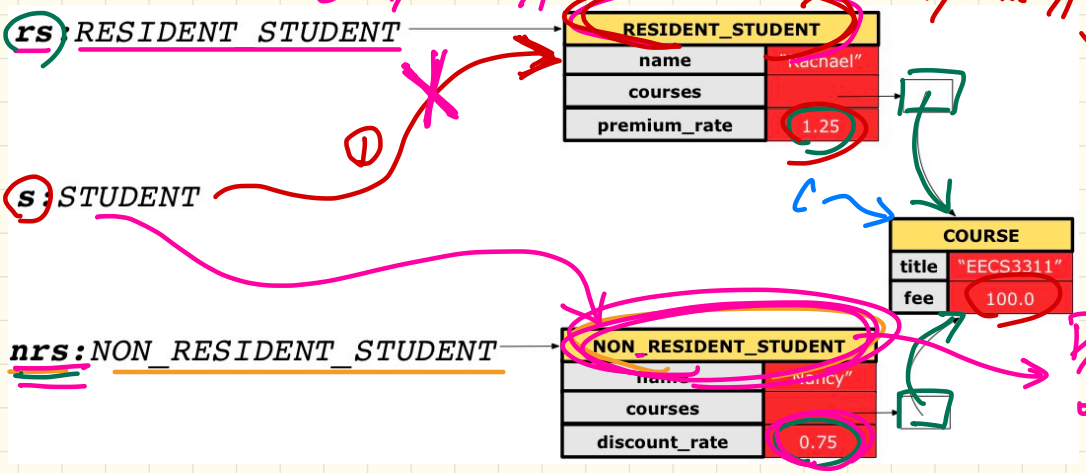
Q. Expectations on rs?

rs' ST: n. LS. reg. tuition

pr set_pr.

Dynamic Binding: Intuition

```
1  local c : COURSE ; s : STUDENT
2        .rs : RESIDENT_STUDENT ; nrs : NON_RESIDENT_STUDENT
3  do create c.make ("EECS3311", 100.0)
4     create {RESIDENT_STUDENT} rs.make("Rachael")
5     create {NON_RESIDENT_STUDENT} nrs.make("Nancy")
6     rs.set_pr(1.25); rs.register(c)
7     nrs.set_dr(0.75); nrs.register(c)
8  ① s := rs  ; check s.tuition = 125.0 end
9  ② s := nrs ; check s.tuition = 75.0 end
```

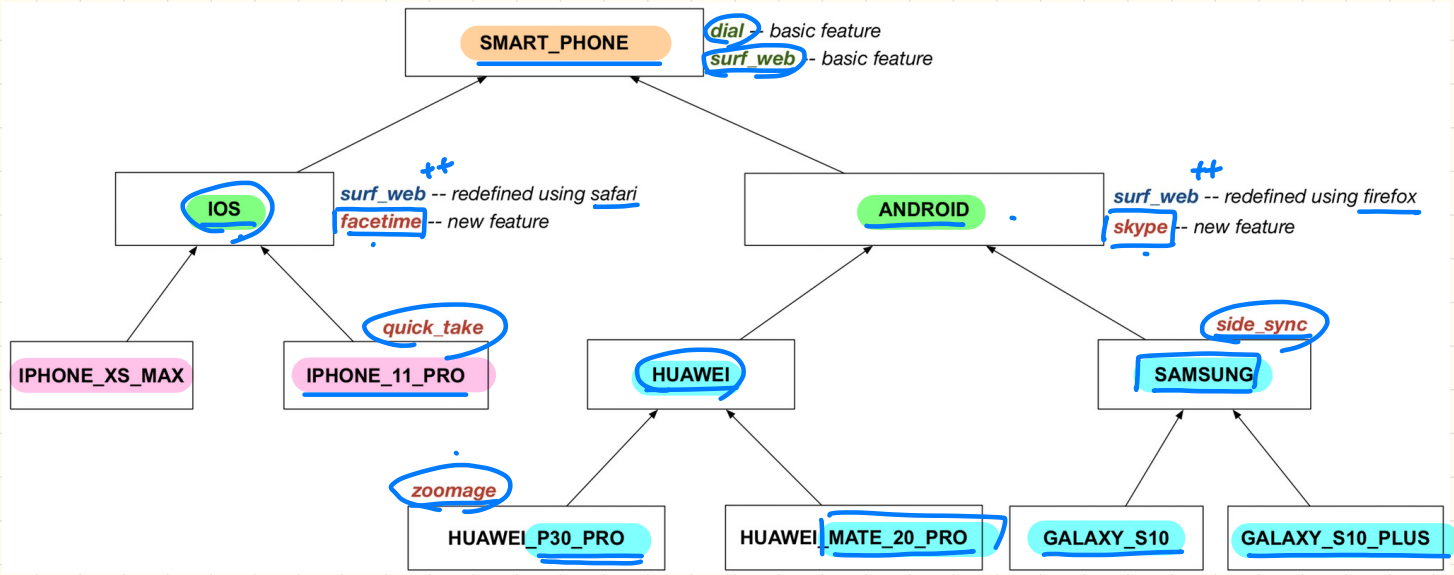


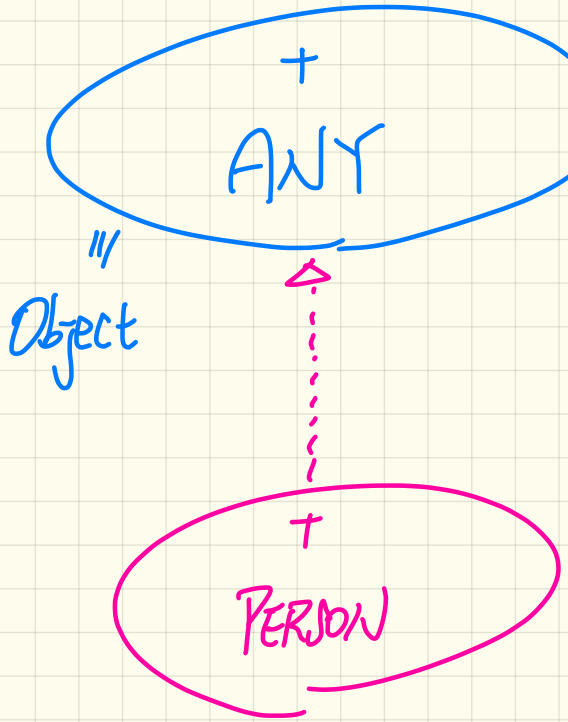
Lecture 7

Part 7

Multi-Level Inheritance Hierarchy

Multi-Level Inheritance Hierarchy of Smartphones





is_equal ← equals()

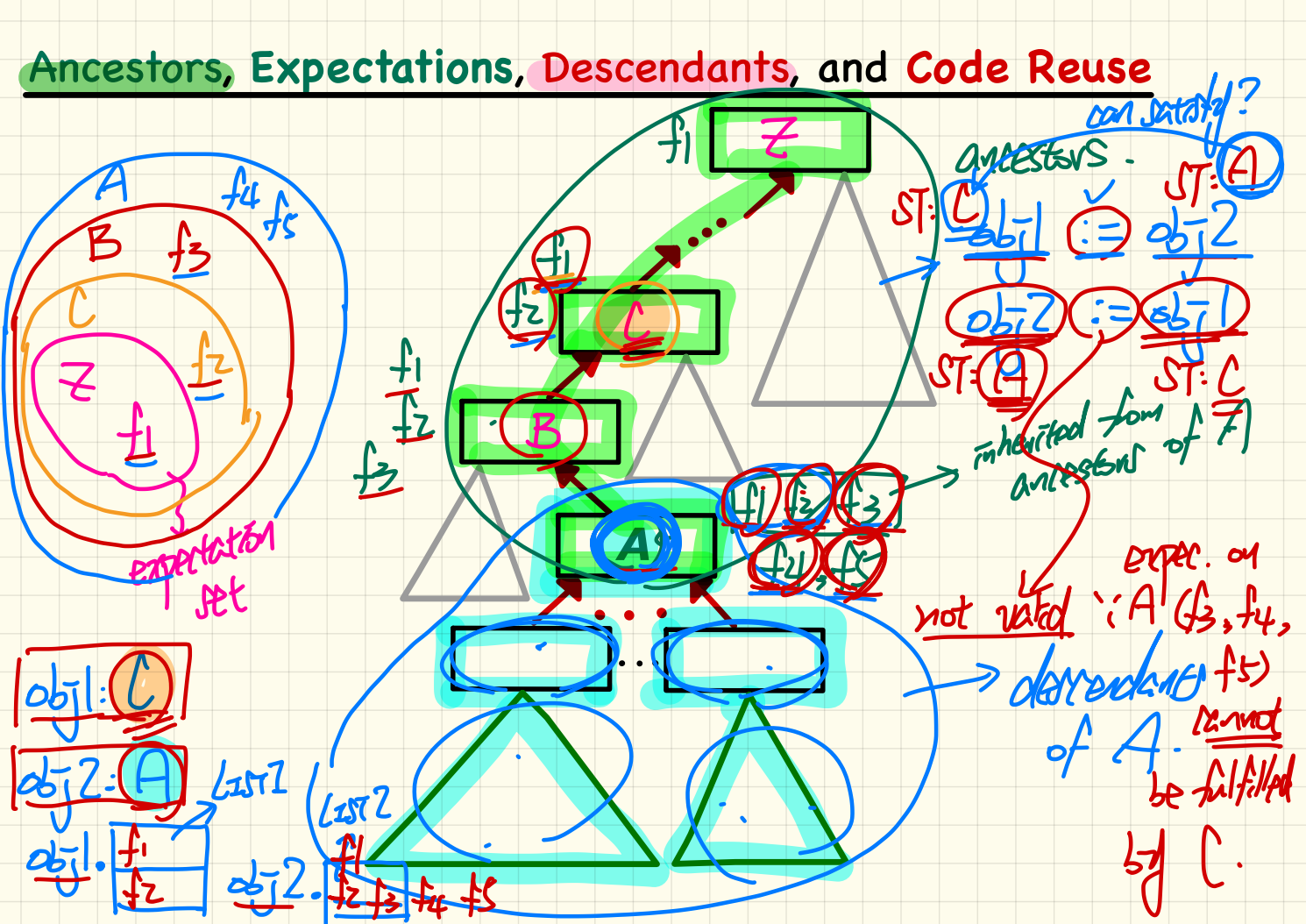
out ← to_string()

inherit ANY

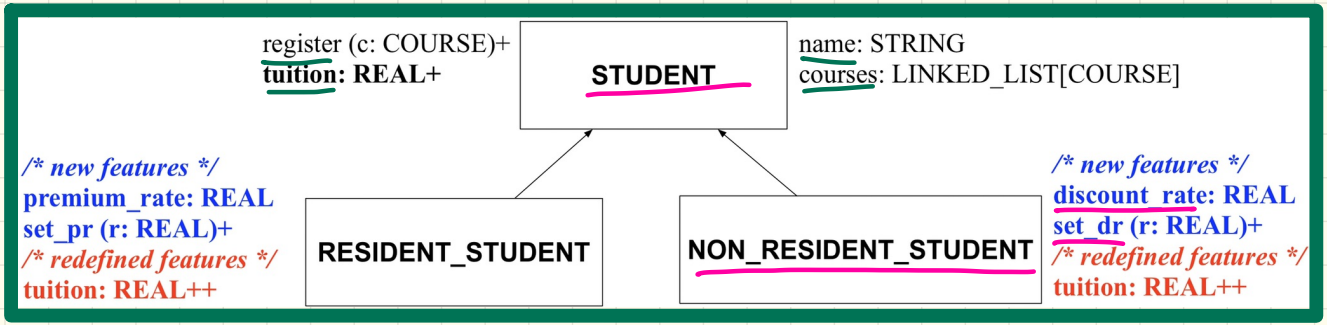
redefine is_equal end

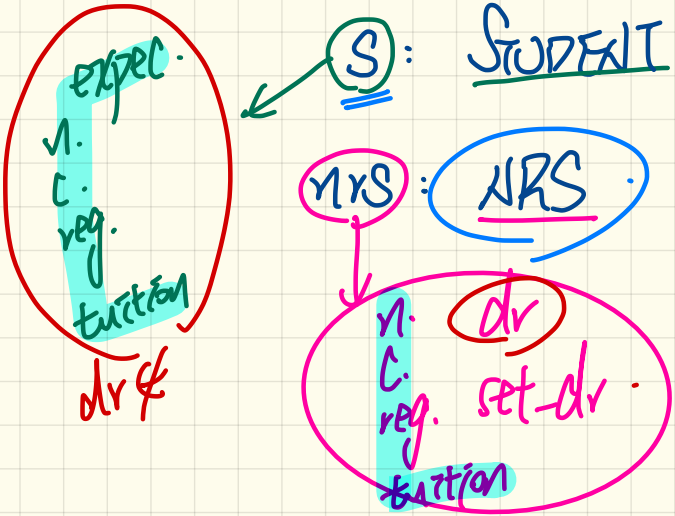
is_equal(...): Bool

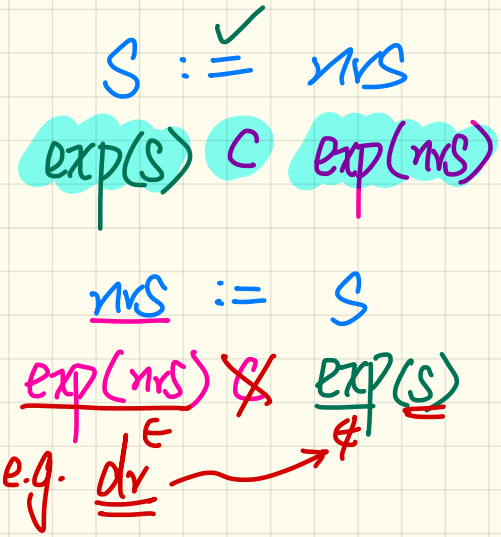
Ancestors, Expectations, Descendants, and Code Reuse



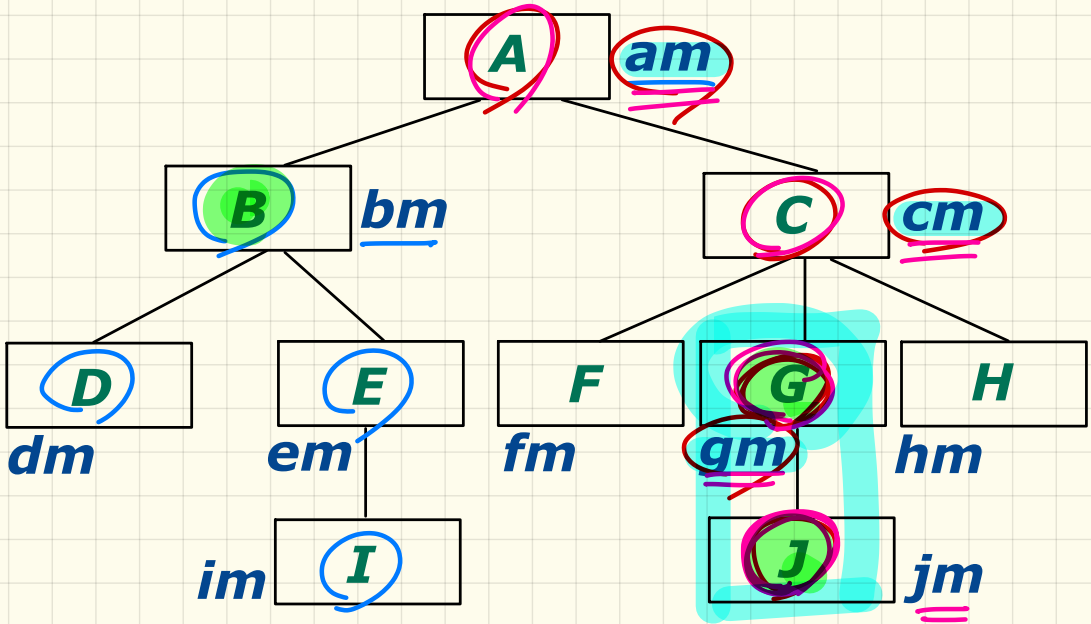
Substitutions by a Descendant Type





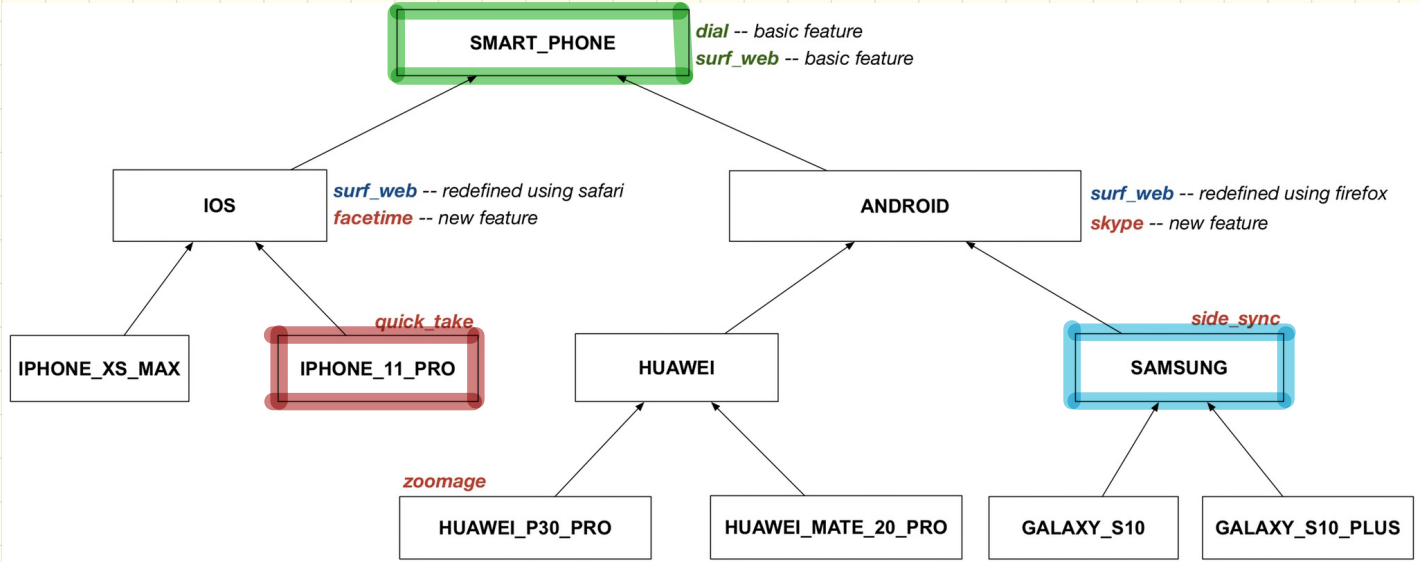


Inheritance Forms a Type Hierarchy (1)



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

Inheritance Forms a Type Hierarchy (2)



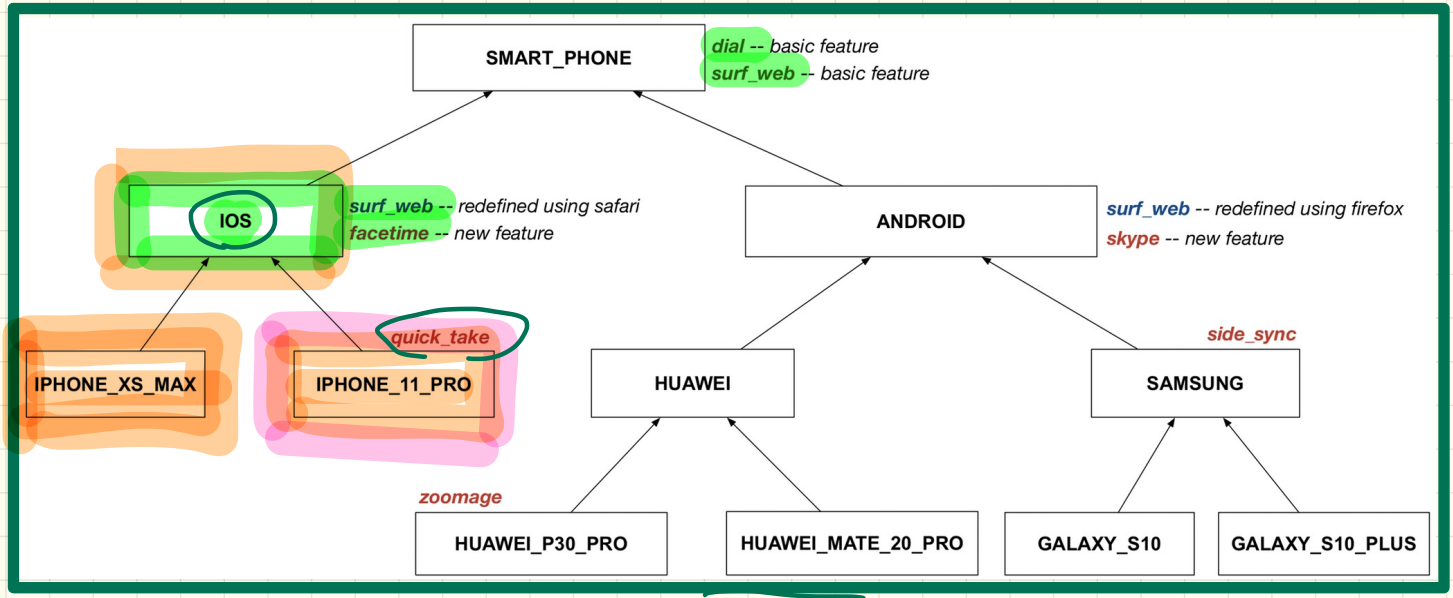
	ancestors	expectations	descendants

Lecture 7

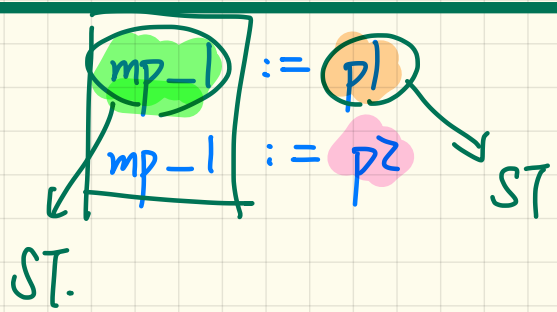
Part 8

Rules of Substitutions via Assignments

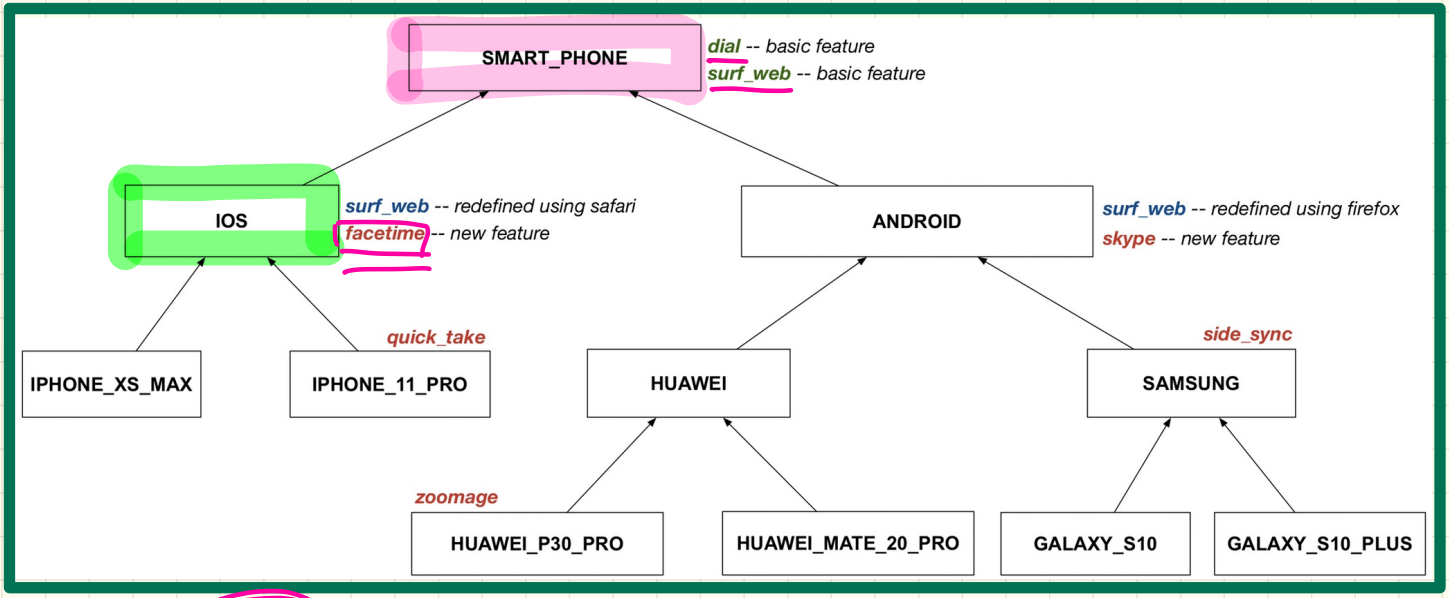
Rules of Substitutions (1)



mp_1: IOS
p1: I_PHONE_XS_MAX
p2: I_PHONE_11_PRO



Rules of Substitutions (2)



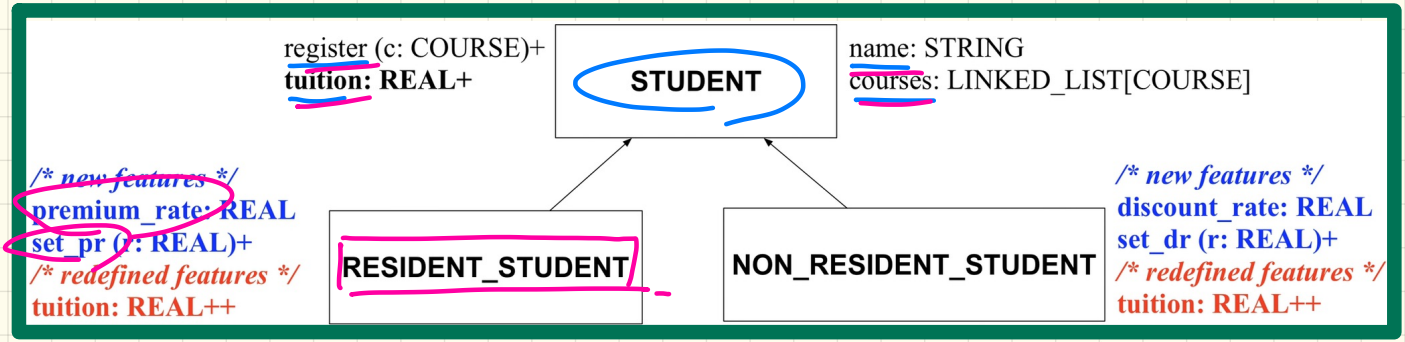
mp_2: IOS
p3: SMART_PHONE

mp_2 := p3

facetime

not safe for substituting mp_2 e.g. facetime

Reference Variables: Static Type



Design 1:

jim: STUDENT

jim.
- n
- cs
- reg
- tuition

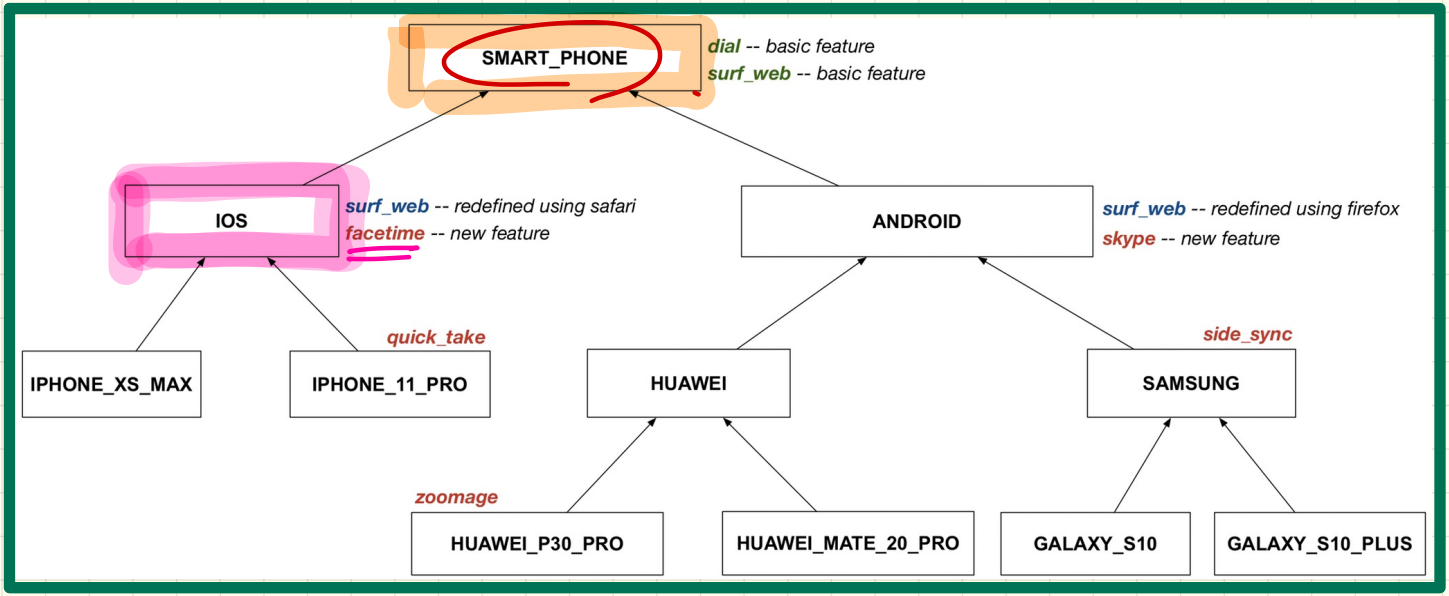
Design 2:

under expectations on jim.

jim: RESIDENT_STUDENT

jim.
- n.
- cs.
- reg.
- tuition
- pr.
- set_pr

Reference Variables: Static Type



Design 1:

mp: **SMART_PHONE**

mp. facetime ✗

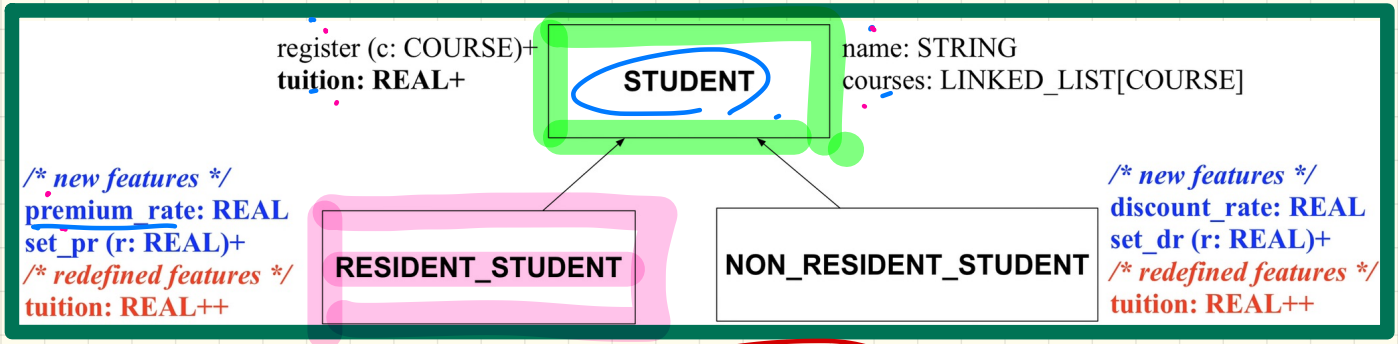
Design 2:

mp: **IOS**

→ wider expectation.

e.g. facetime.

Change of Dynamic Type (1)



Design 1:

exp(STUDENT)

jim: STUDENT

create { ? } jim.make(...)

Jim → ? → STU. RS. NRS.

Design 2:

ST has wider expec, making there fewer choices of classes that can

exp(RS)

jim: RESIDENT_STUDENT

create { } jim.make(...)

RS

fulfill its expectations.

X STUDENT

Change of Dynamic Type (2)

```
1  test_polymorphism_students
2   local
3     jim: STUDENT
4     rs: RESIDENT_STUDENT
5     nrs: NON_RESIDENT_STUDENT
6   do
7    create {STUDENT} jim.make ("J. Davis")
8    create {RESIDENT_STUDENT} rs.make ("J. Davis")
9    create {NON_RESIDENT_STUDENT} nrs.make ("J. Davis")
10     jim := rs   ✓
11     rs := jim   ×
12     jim := nrs  ✓
13     rs := jim   ×
14   end
```

ST: RS

ST: S

changes DT of Jim. from S. to RS.

changes DT of Jim from RS to NRS.

Jim

rs

nrs

STUDENT	
n.	"J. Davis"
cs.	

RESIDENT_S.	
n.	"J. Davis"
cs.	
pr.	

NON_RESI_S.	
n.	"J. Davis"
cs.	
dr.	

Testing of Dynamic Binding

RESIDENT_S.	
n.	
cs.	
pr.	

NON_RESI_S.	
n.	
cs.	
dr.	

STUDENT	
n.	
cs.	

```
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
```

```
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
```

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
```

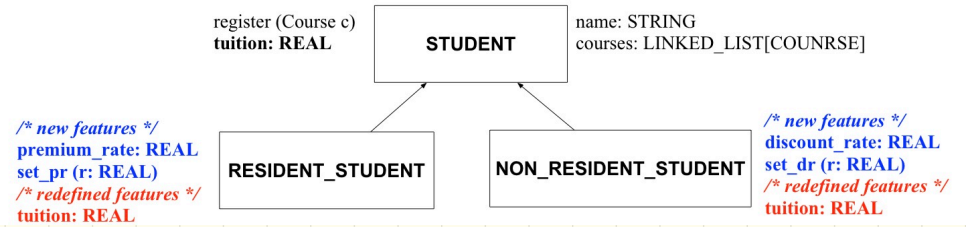
```
class
  NON_RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  discount_rate: REAL
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ; Result := base * discount_rate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

Lecture 7

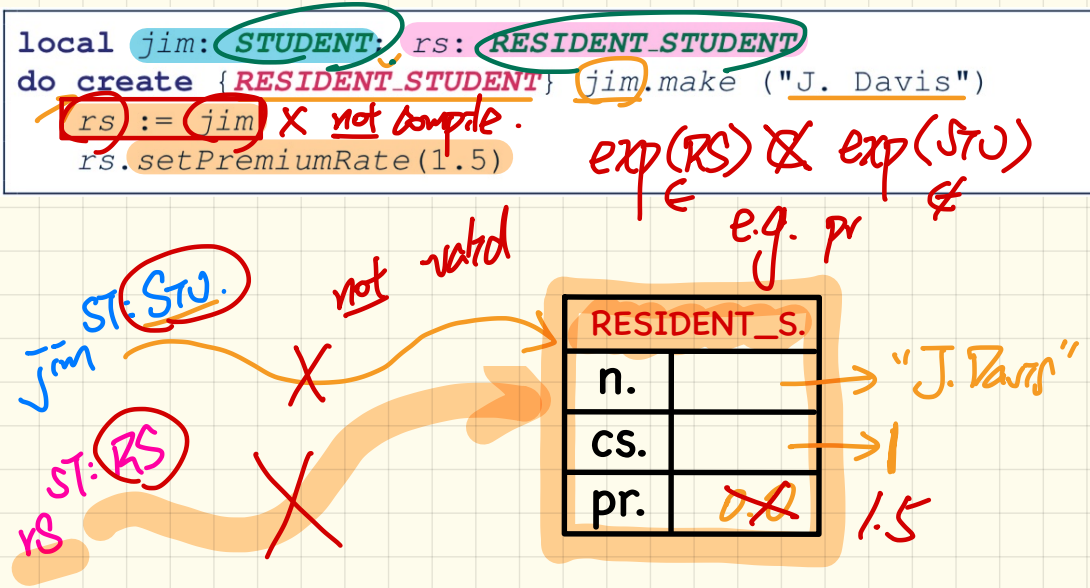
Part 9

Type Casting

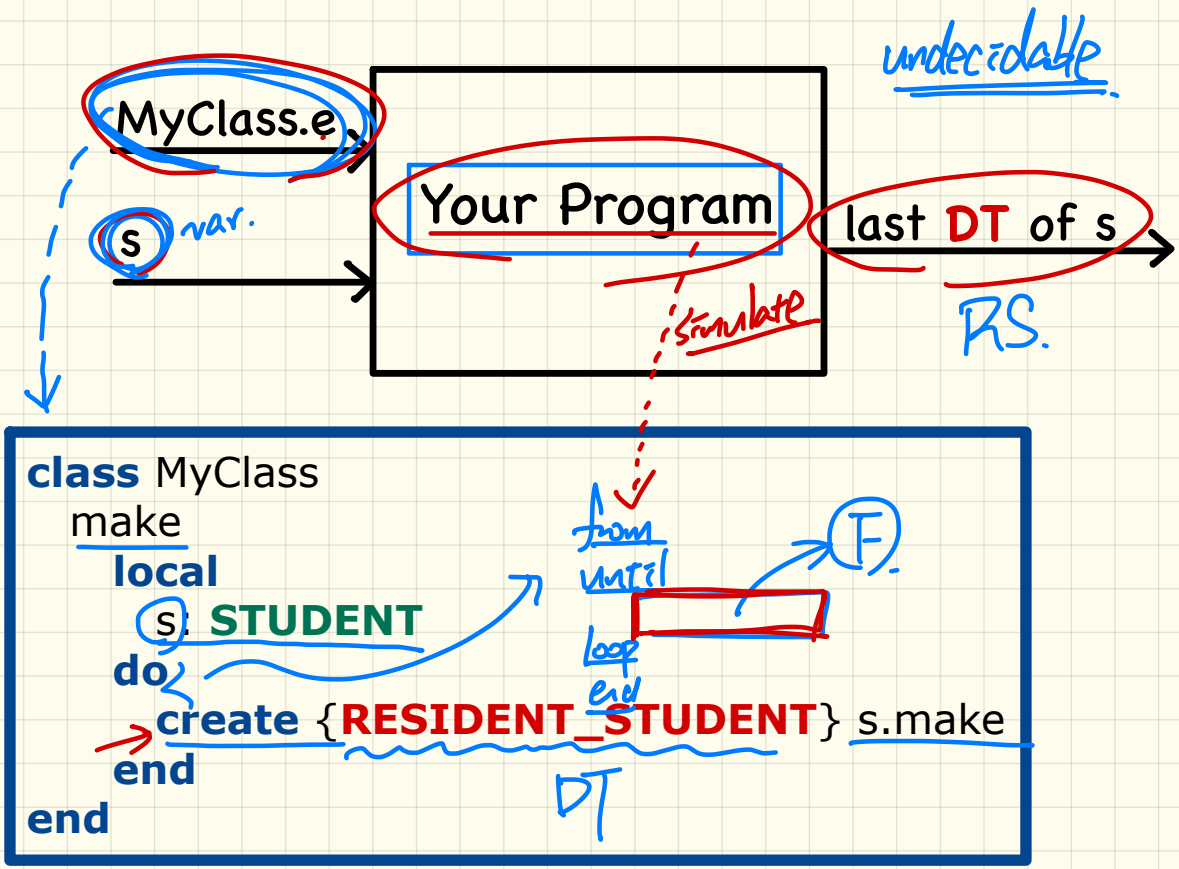
Type Cast: Motivation




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



Inferring the DT of a Variable is Undecidable



Type Cast: Syntax

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

① if ⇒ assertion violation

can the DT of jim fulfill the exp of RS

① eval to true
② alias to what jim is pointing to with ST RS.

ST: RS. rs_jim

RS

jim

STUDENT

rs

RS

RESIDENT_S.	
n.	J. Davies
cs.	
pr.	1.5

②

if attached {RS} jim as r then

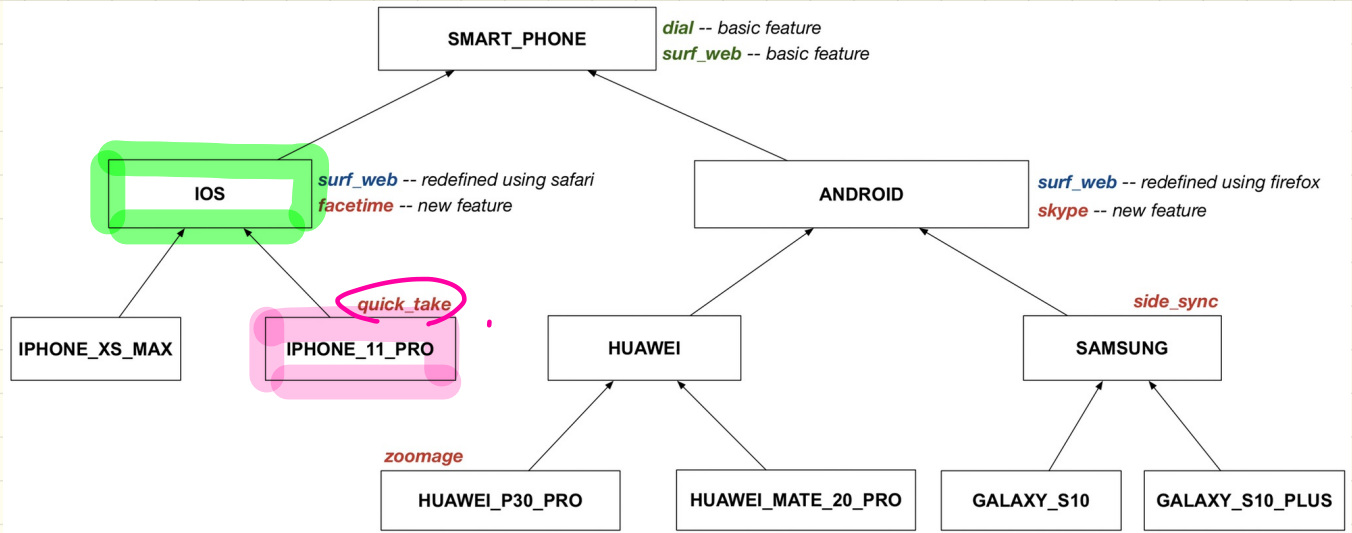
rs := r

rs.set_pr(1.5)

else

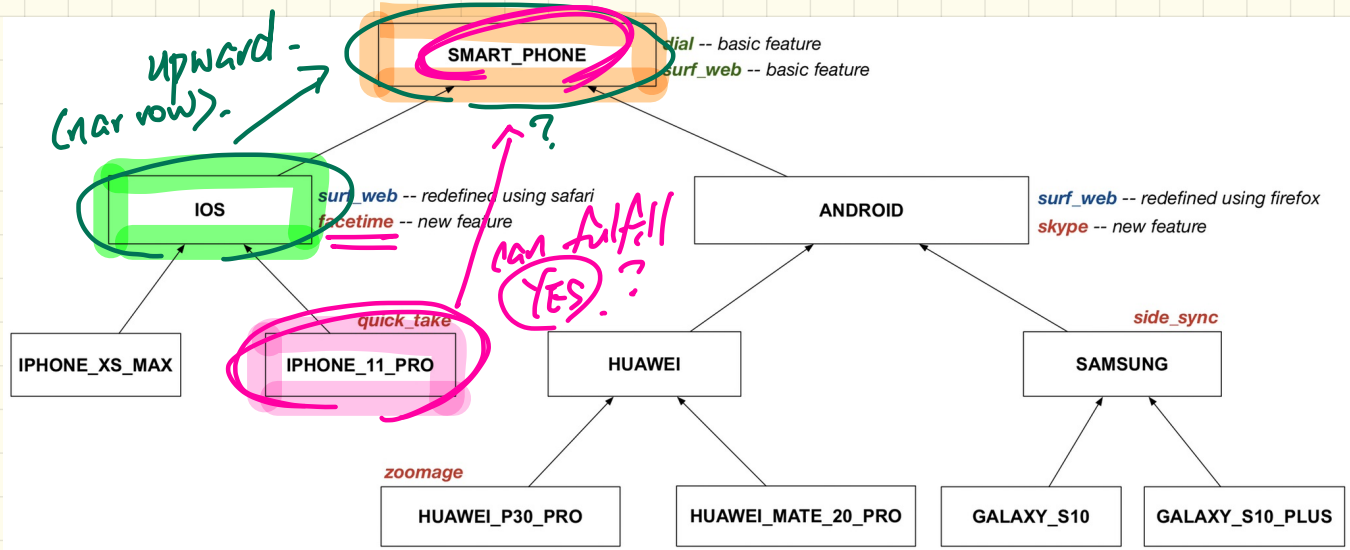
end

Violation-Free Cast: Upwards or Downwards (1)



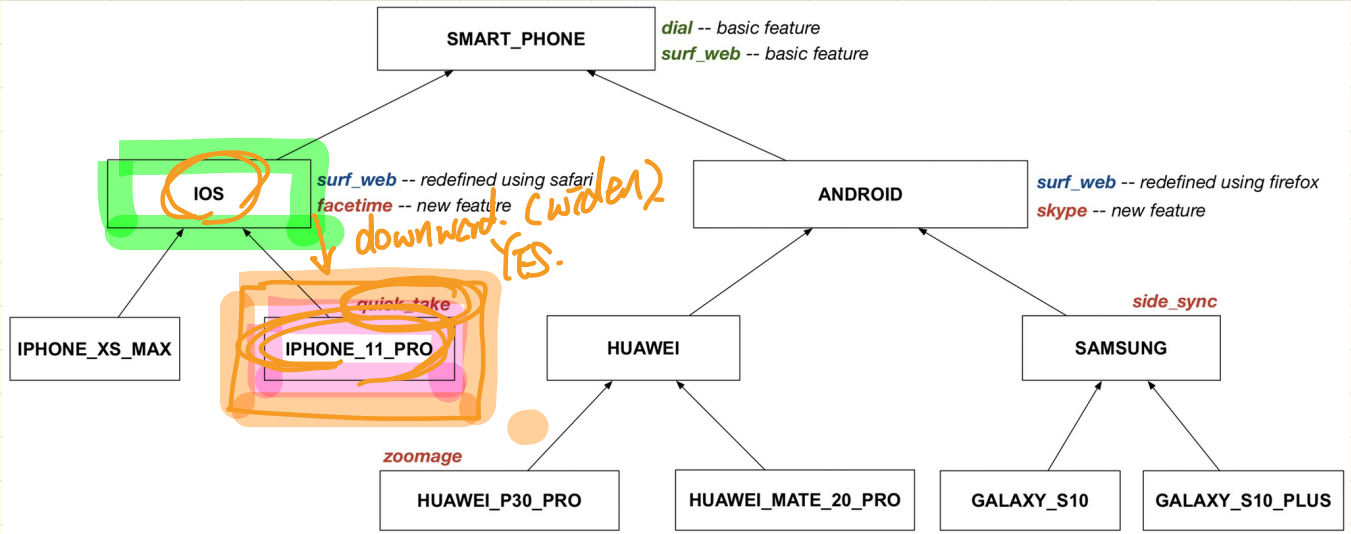

```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime ✓  quick_take, skype, side_sync, zoomage ×
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web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

Violation-Free Cast: Upwards or Downwards (2)




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

Violation-Free Cast: Upwards or Downwards (3)

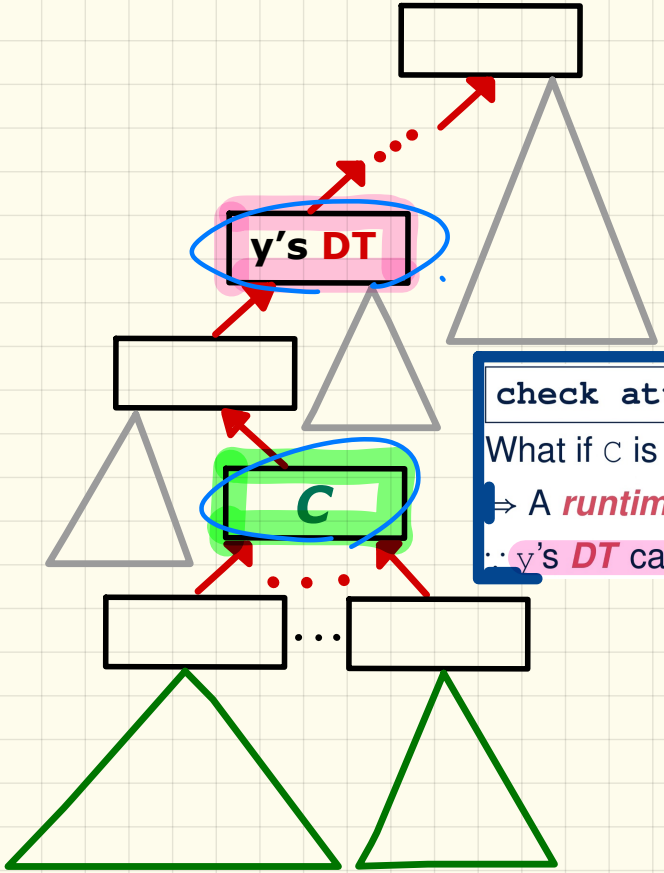



```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime ●  quick_take, skype, side_sync, zoomage ●
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web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

has a wider expectation than my_phone.

ST: I_11_Pro.

Ancestors, Expectations, Descendants, and Code Reuse



can the DT of y fulfill the exper. on C?

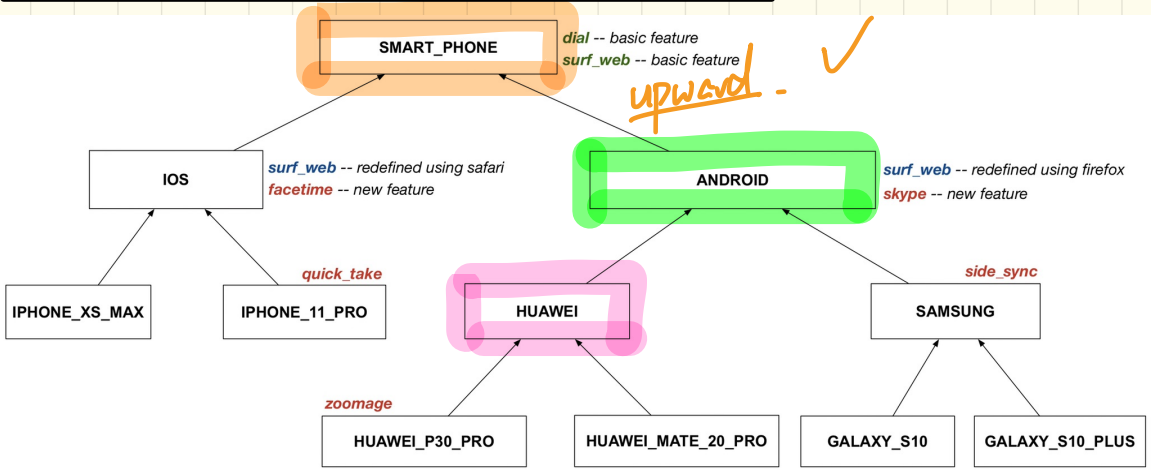
`check attached {C} (y) then ... end` *always compiles*

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

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

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

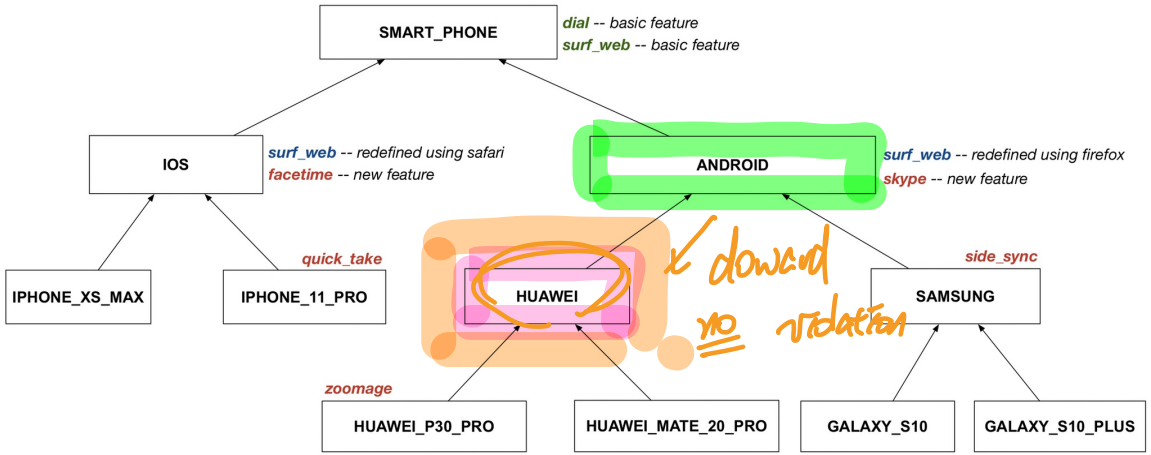
Cast Violation at Runtime (1)




```
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
```

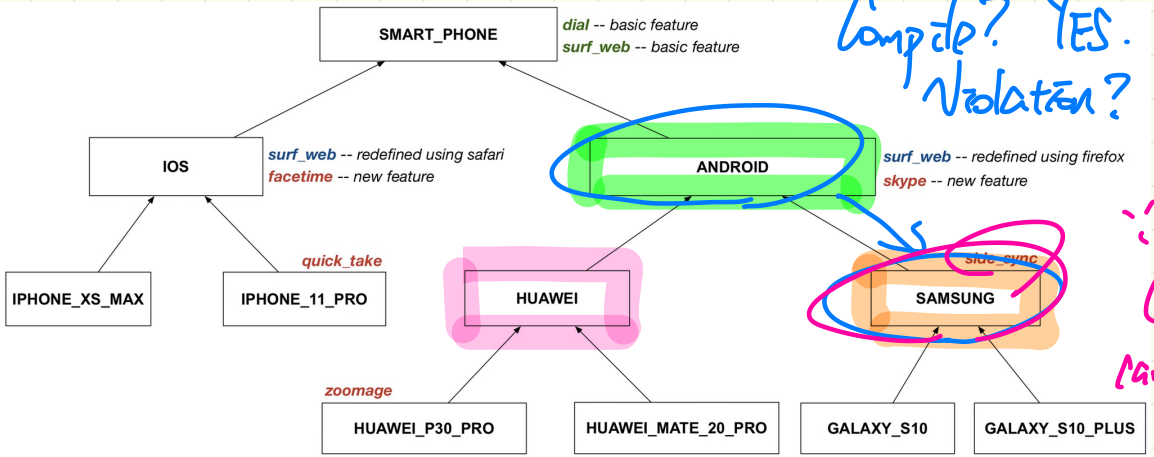
No. violation

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
```

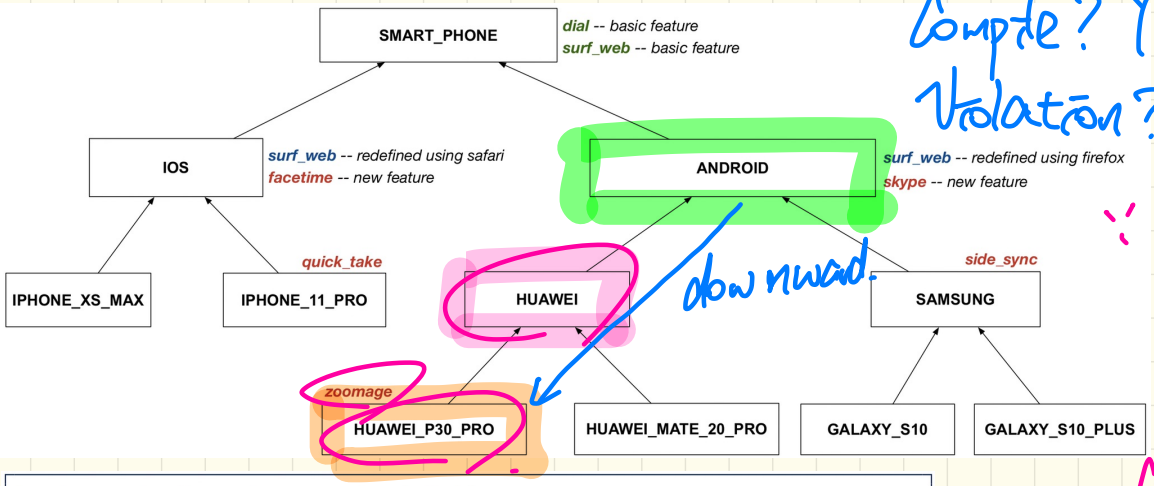
Cast Violation at Runtime (3)



Compile? YES. Downward-Violation? YES.
↓
∵ DT of mine (HUAWEI) cannot fulfil the cast type SAMSUNG's expec (e.g. side_sync)

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

Cast Violation at Runtime (4)



Compile? YES.
Violation? YES
∵ DT HUAWEI can't fulfill the last type HUAWEI_P30_PRO's exp. (zoomage).

```
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
```

Lecture 7

Part 10

Polymorphic Routine Arguments

Feature Call Parameters: Supplier

```
class STUDENT_MANAGEMENT_SYSTEM
  ss : ARRAY[STUDENT]
  add_s (s: STUDENT) do ss[0] := s end
  add_rs (rs: RESIDENT_STUDENT) do ss[0] := rs end
  add_nrs (nrs: NON_RESIDENT_STUDENT) do ss[0] := nrs end
```

Say:

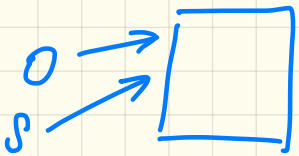
sms: STUDENT_MANAGEMENT_SYSTEM

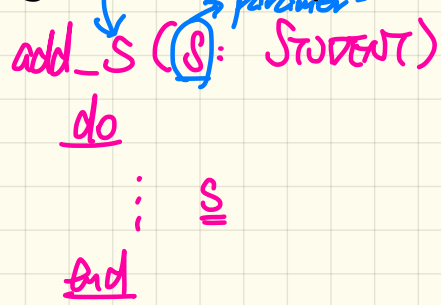
When should the following calls compile?

sms.add_s (o)

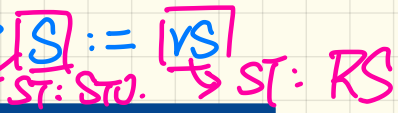
sms.add_rs (o)

sms.add_nrs (o)

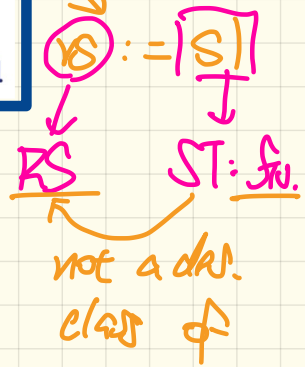




Feature Call Arguments: Client




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




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

not relevant to Static compilation

sms.add_s (rs) ✓

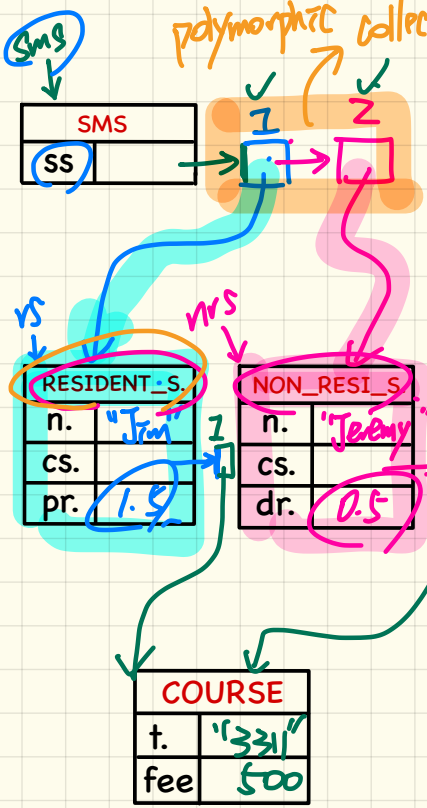
sms.add_rs (s1) ✗

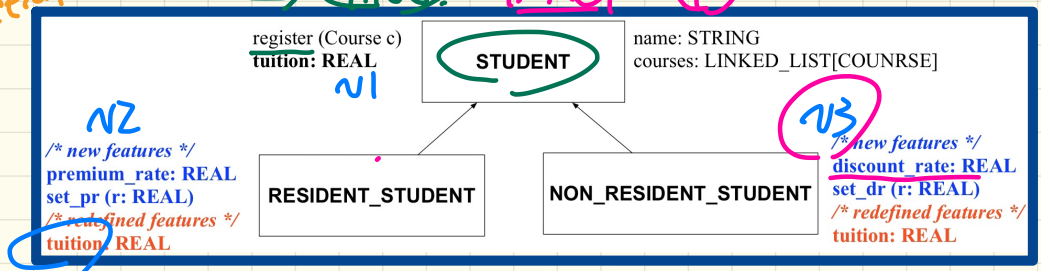
Lecture 7

Part 11

Polymorphic Collections

Polymorphic Collection






```
test_sms_polymorphism: BOOLEAN
  local
    rs: RESIDENT_STUDENT
    nrs: NON_RESIDENT_STUDENT
    c: COURSE
    sms: STUDENT_MANAGEMENT_SYSTEM
  do
    create rs.make ("Jim")
    rs.set_pr (1.5)
    create nrs.make ("Jeremy")
    nrs.set_dr (0.5)
    create sms.make
    sms.add_s (rs)
    sms.add_s (nrs)
    create c.make ("EECS3311", 500)
    sms.register_all (c)
    Result := sms.ss[1].tuition = 750 and sms.ss[2].tuition = 250
  end
```

```
class STUDENT_MANAGEMENT_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
```

v2 of tuition

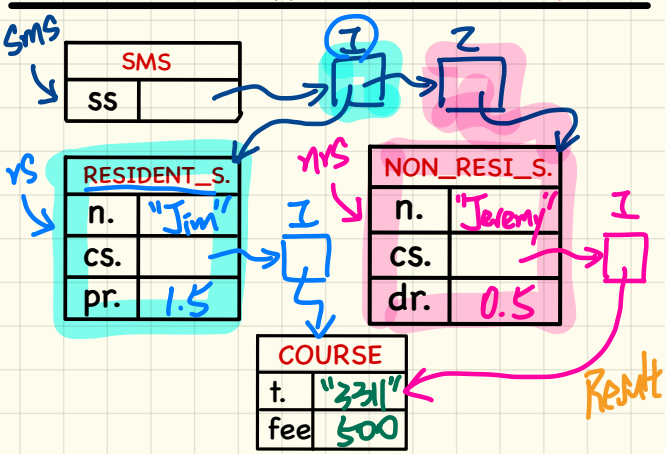
v3 of tuition

Lecture 7

Part 12

Polymorphic Return Values

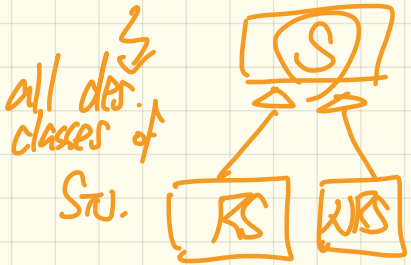
Feature Call Return Values




```
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
```

```
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.get_student(1).tuition = 750
    and get_student(2).tuition = 250
end
```

Possible DT of Result?



Lecture 8

Part 1

General Book: Storage vs. Retrieval

General Book

Supplier

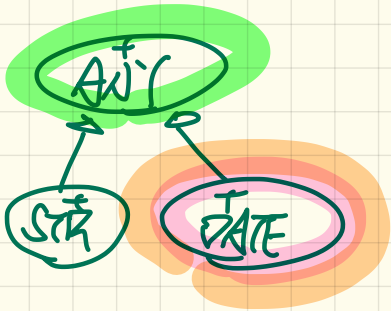
```
class BOOK
  names: ARRAY[STRING]
  records: ARRAY[ANY]
  -- Create an empty book
  make do ... end
  -- Add a name-record pair to the book
  add (name: STRING; record: ANY) do ... end
  -- Return the record associated with a given name
  get (name: STRING): ANY do ... end
end
```

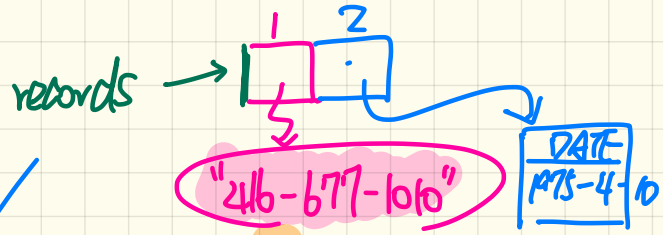
Client

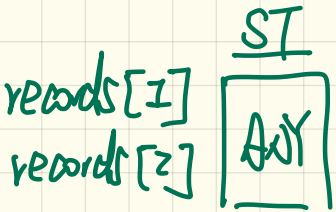
```
1  birthday: DATE; phone_number: STRING
2  b: BOOK; is_wednesday: BOOLEAN
3  create {BOOK} b.make
4  phone_number := "416-677-1010"
5  b.add ("SuYeon", phone_number)
6  create {DATE} birthday.make(1975, 4, 10)
7  b.add ("Yuna", birthday)
8  is_wednesday := b.get("Yuna").get_day_of_week = 4
```

General Book: Retrieval from Polymorphic Arrary

```
1  birthday: DATE; phone_number: STRING
2  b: BOOK; is_wednesday: BOOLEAN
3  create {BOOK} b.make
4  phone_number := "416-677-1010"
5  b.add ("SuYeon", phone_number)
6  create {DATE} birthday.make(1975, 4, 10)
7  b.add ("Yuna", birthday)
```







DT
STRING
DATE

```
check attached {DATE} b.get("Yuna") as yuna_bday then
  is_wednesday := yuna_bday.get_day_of_week = 4
end
```

↳ ST : DATE .

⇒ Violation of R.T. expectation on last type DATE

```
check attached {DATE} b.get("SuYeon") as suyeon_bday then
  is_wednesday := suyeon_bday.get_day_of_week = 4
end
```

↳ downcast but DT STR can't fulfil

General Book violates Single Choice Principle

Storage

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

Retrievals

disadvantage

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

elseif attached {C101} ... then ...

repetition of check structure ⇒ violation of SCP.

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

elseif attached {C101} ... then ...

What if a new type C101 is introduced? X

What if type C100 becomes obselete?

Lecture 8

Part 2

Generic Book: Storage vs. Retrieval

Generic Book

declaration

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

ANY ANY DATE. usage. ANY DATE.

Supplier

```
birthday: DATE; phone_number: STRING
b: BOOK[DATE] ; is_wednesday: BOOLEAN
create {BOOK[DATE]} b.make
phone_number = "416-67-1010"
b.add ("SuYeon", phone_number)
create {DATE} birthday.make (1975, 4, 10)
b.add ("Yuna", birthday)
is_wednesday := b.get ("Yuna").get_day_of_week == 4
```

Client

more restrictive on storage.

retrieval does not require cast.

ST: DATE

Instantiating Generic Parameters

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

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

Clients use `DICTIONARY` with different degrees of instantiations:

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

C1 C2

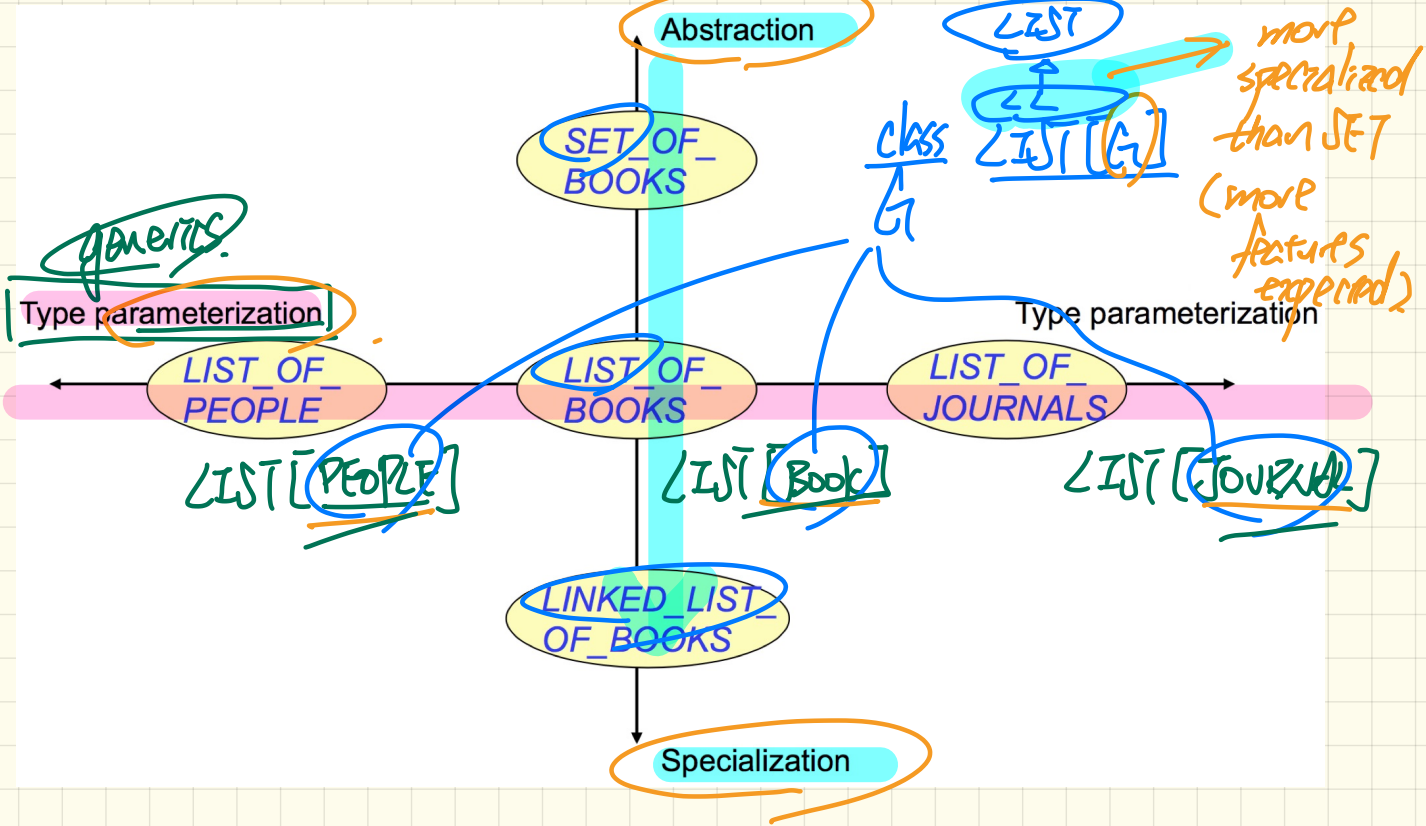
e.g., Declaring `DATABSE_TABLE[INTEGER, STRING]` instantiates `DICTIONARY[STRING, INTEGER]`.

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

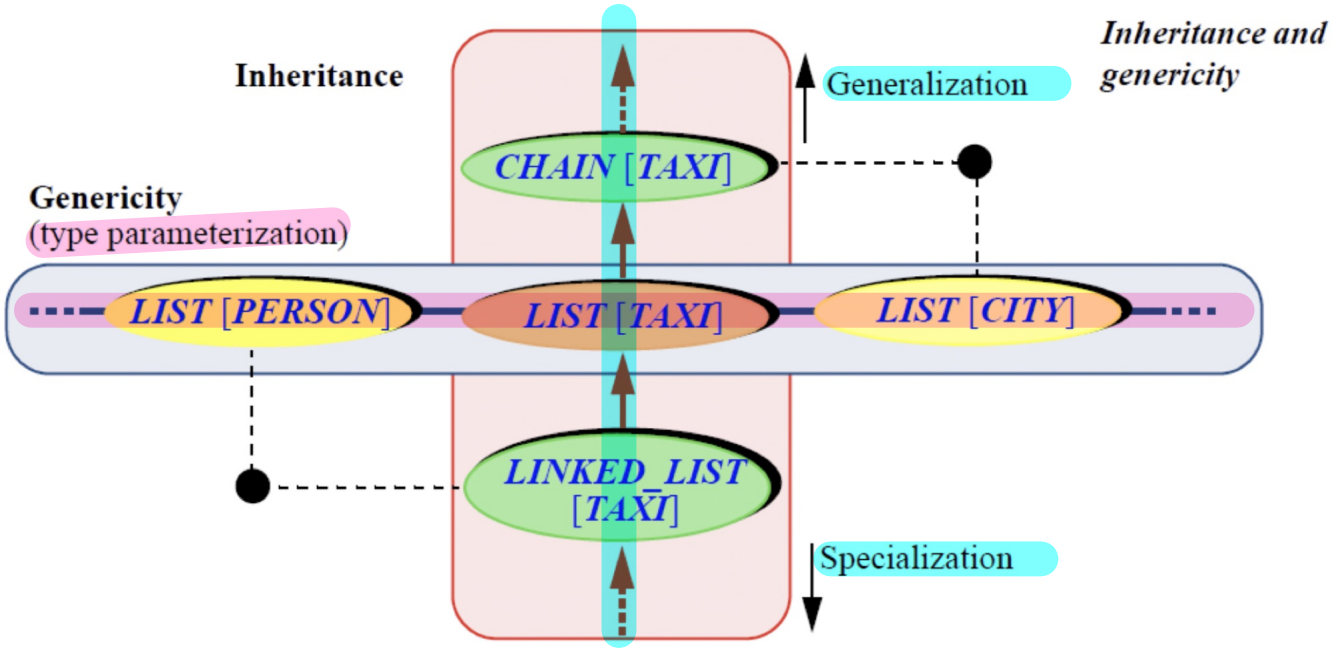
C3 C4

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

Generics vs. Inheritance (1)



Generics vs. Inheritance (2)



Lecture 8

Part 3

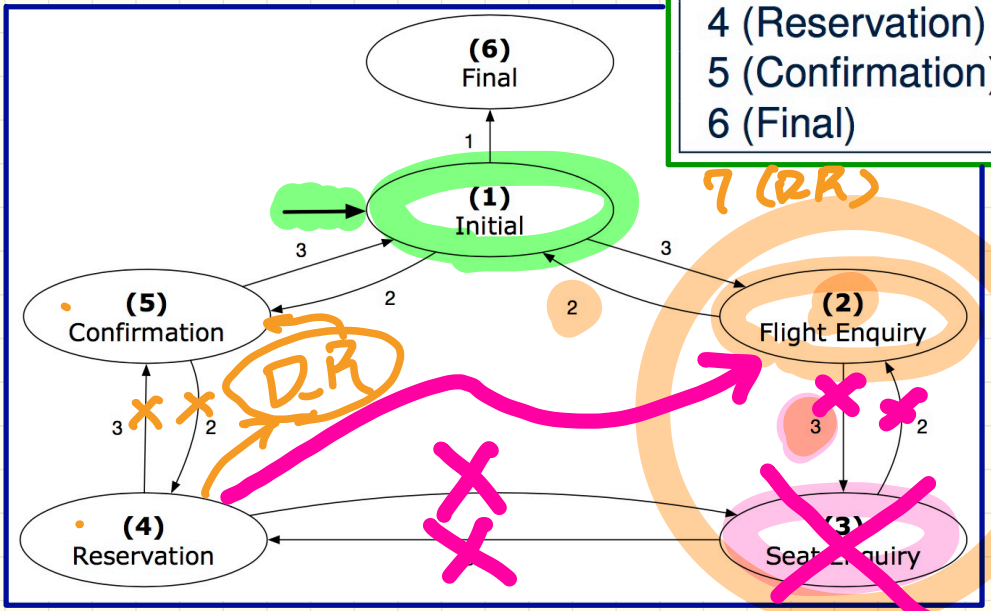
Motivating Problem: Interactive Systems

Finite State Machine (FSM)

State Transition Table

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

State Transition Diagram



Lecture 8

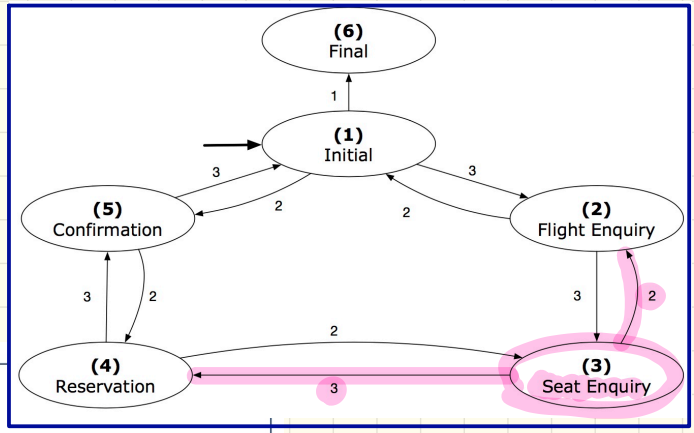
Part 4

First Design: Assembly Style

Design of a Reservation System: First Attempt

while (wrong choice ∨ wrong answer)
↳ stay condition
↳ as long as it's true
keep iterating.

exit condition:
as soon as it's true, exit



superman module (not modular)

```
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
```

or wrong timing

D.M.

≡ not w.a. and not w.c.

≡ correct a. ∧ correct c.

Lecture 8

Part 5

Second Design: Hierarchical Style

Design of a Reservation System: Second Attempt (1)

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

Examples: (target state)

transition(3, 2) → 2

transition(3, 3) → 4

State Transition Table

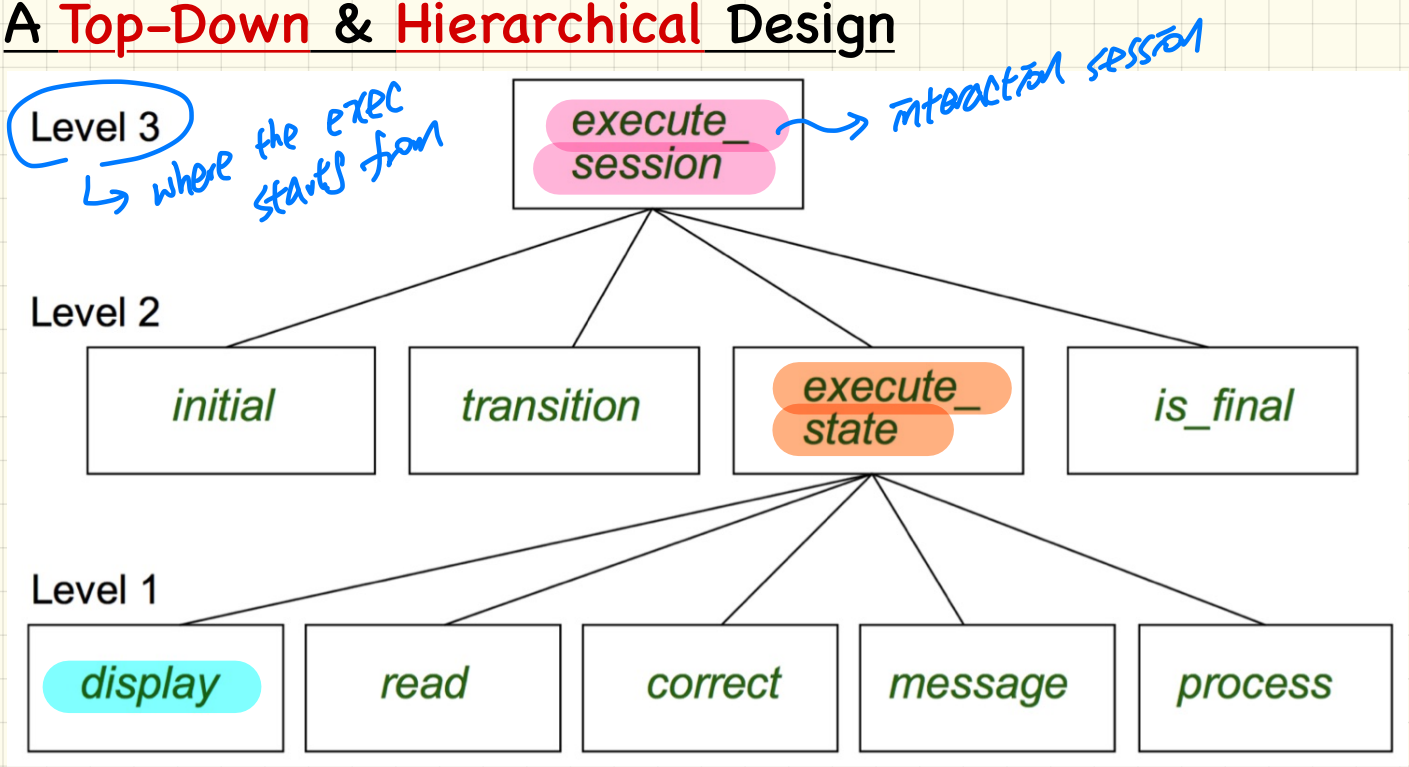
SRC STATE \ CHOICE	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

2D Array Implementation

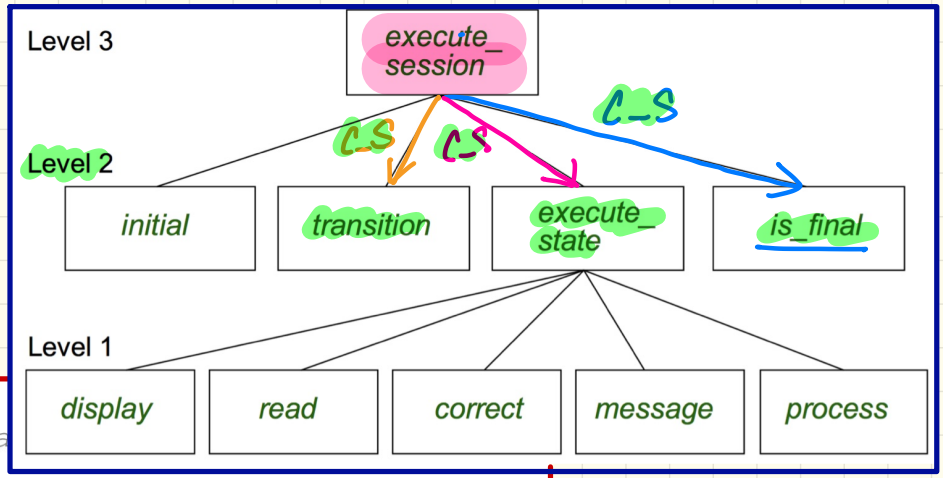
state \ choice	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

Design of a Reservation System: Second Attempt (2)

A Top-Down & Hierarchical Design



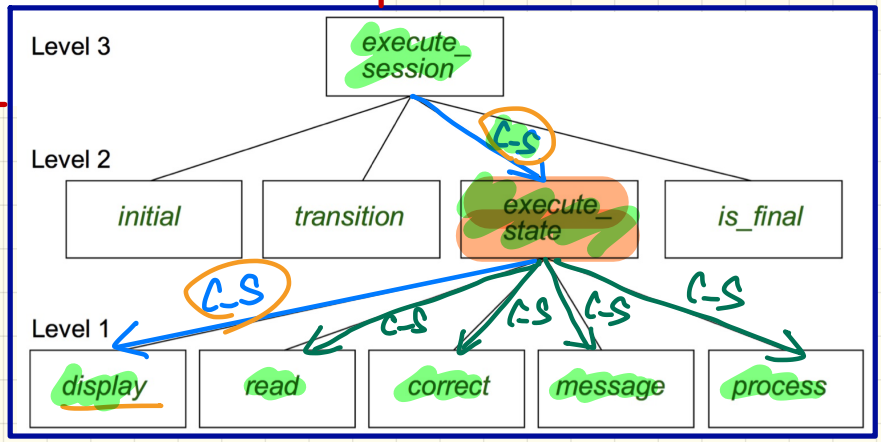
Design of a Reservation System: Second Attempt (3)




```
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
```

Design of a Reservation System: Second Attempt (4)

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

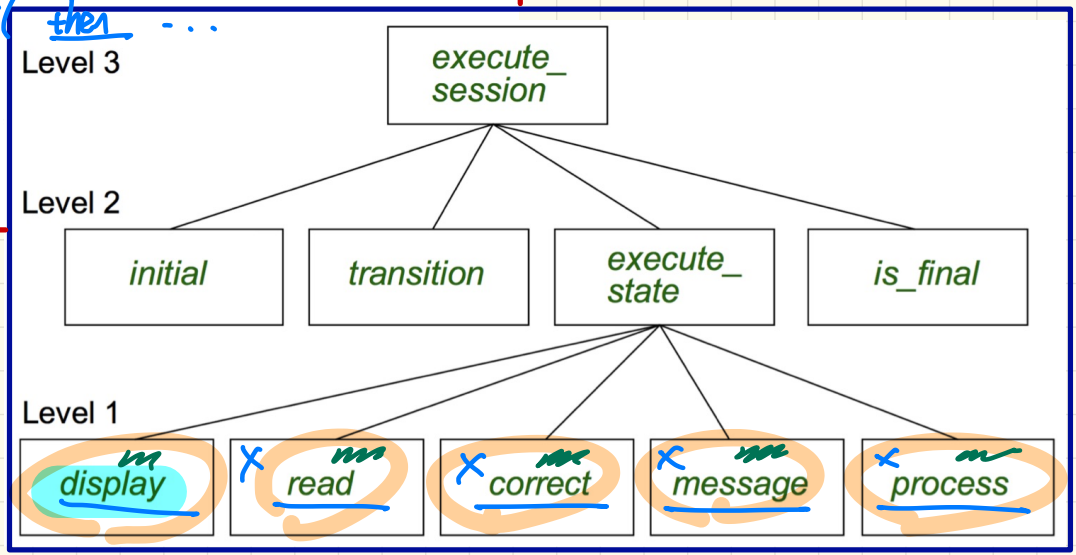


Design of a Reservation System: Second Attempt (5)

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

Add a new state? ?

Delete an existing state? ?



Lecture 8

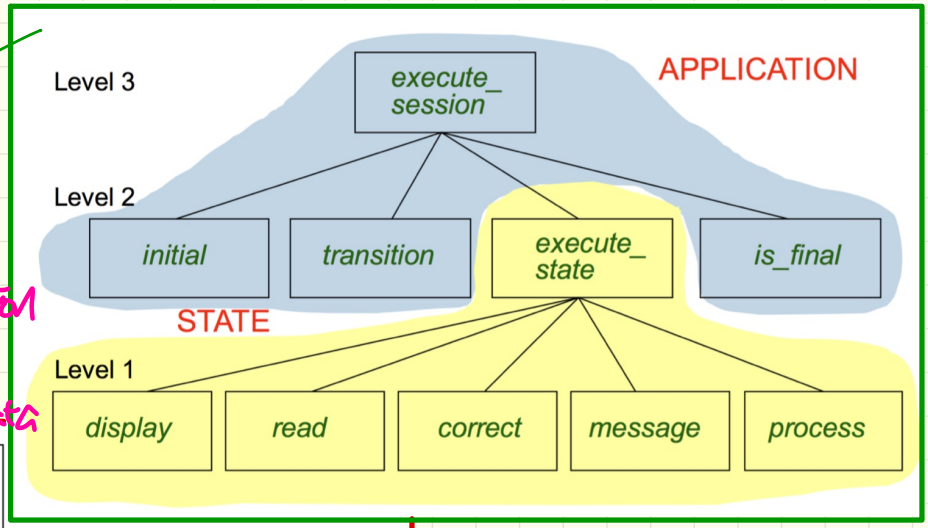
Part 6

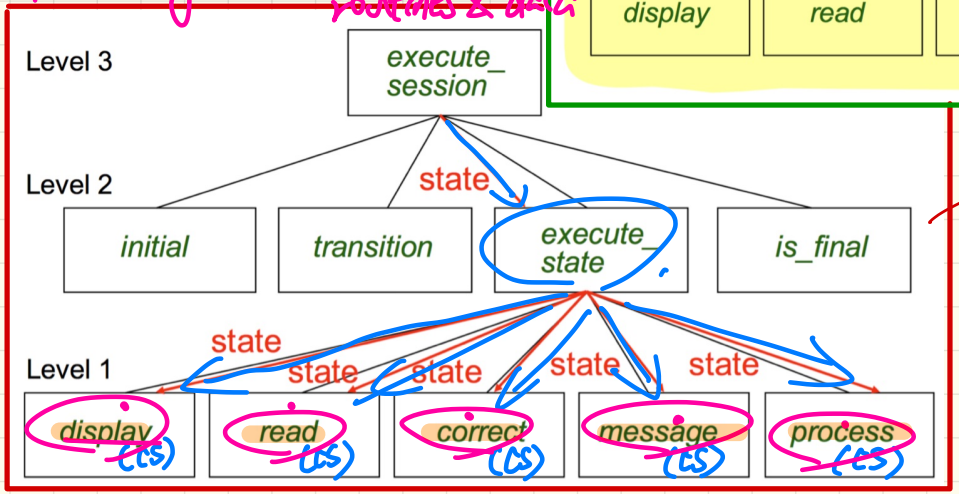
Template & State Patterns: Supplier

Moving from Top-Down Design to OO Design

Object-Oriented

current_state: STATE
current_state.execute

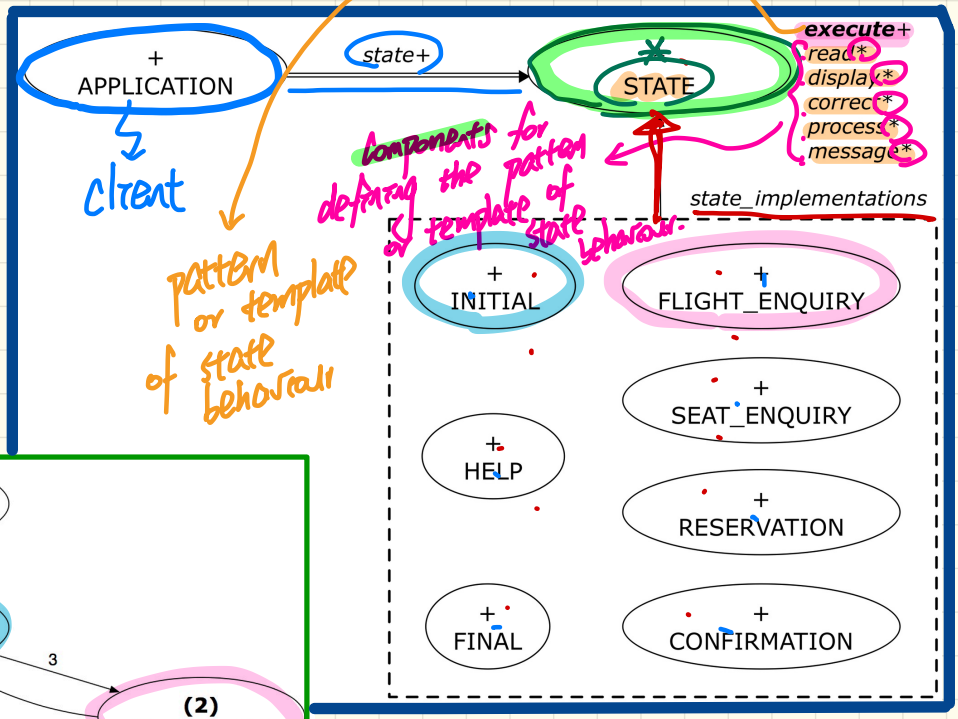


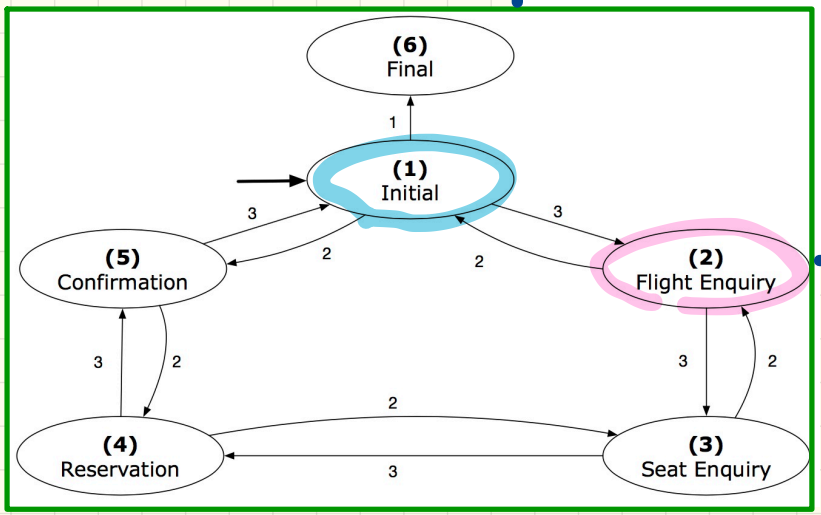


Top-Down

current_state: INTEGER
execute_state(current_stste)

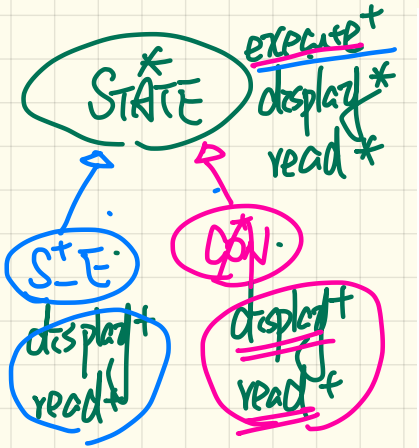
State Pattern: Architecture

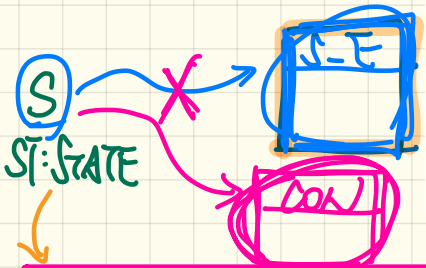





```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute -> w.r.t.
create {CONFIRMATION} s.make
s.execute -> w.r.t.
```

State Pattern: State Module






```
deferred class STATE
  read
    -- Read user's inputs
    -- Set 'answer' and 'choice'
    deferred end
  answer: ANSWER
    -- Answer for current state
  choice: INTEGER
    -- Choice for next step
  display
    -- Display current state
    deferred end
  correct: BOOLEAN
    deferred end
  process
    require correct
    deferred end
  message
    require not correct
    deferred end
```

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

effective (implemented)

S.display
S.display
S.read
S.read

TEMPLATE (pattern)

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

{STATE}.execute
{STATE}.execute

{S_E}.display
{S_E}.read
{CONF}.display
{CONF}.read

Lecture 8

Part 6

Template & State Patterns: Client

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
```

call by value

put_state(@2, ...)

consequence of having a polymorphic argument is we have a polymorphic collection of states

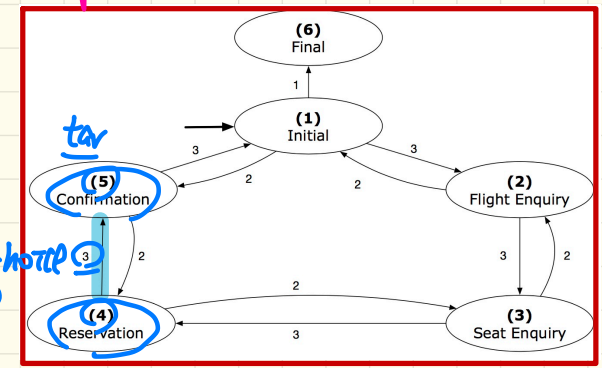
s := a

put_tran(5, 4, 3)

choice @3

tar

src



State Pattern: Test

polymorphic

```
test_application: BOOLEAN
  local
    app: APPLICATION ; current_state: STATE ; index: INTEGER
  do
    create app.make (6, 3)
    app.put_state (create {INITIAL}.make, 1)
    -- Similarly for other 5 states.
    app.choose_initial (1)
    -- Transit to FINAL given current state INITIAL and choice
    app.put_transition (6, 1, 1)
    -- Similarly for other 10 transitions.

    index := app.initial
    current_state := app.states [index]
    Result := attached {INITIAL} current_state
    check Result end
    -- Say user's choice is 3: transit from INITIAL to FLIGHT_STATUS
    index := app.transition.item (index, 3)
    current_state := app.states [index]
    Result := attached {FLIGHT_ENQUIRY} current_state
end
```

states

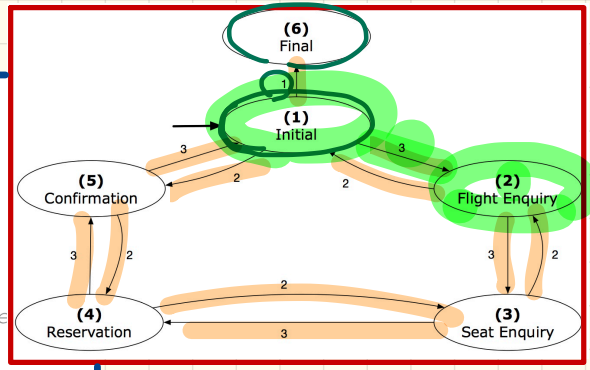
choices

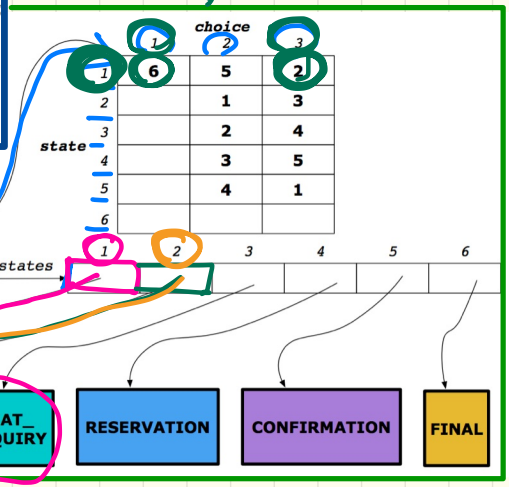
valid? ✓

set initial to 1

c_s.display

c_s.display





STATES[1] ST: STATE

[2] DT?

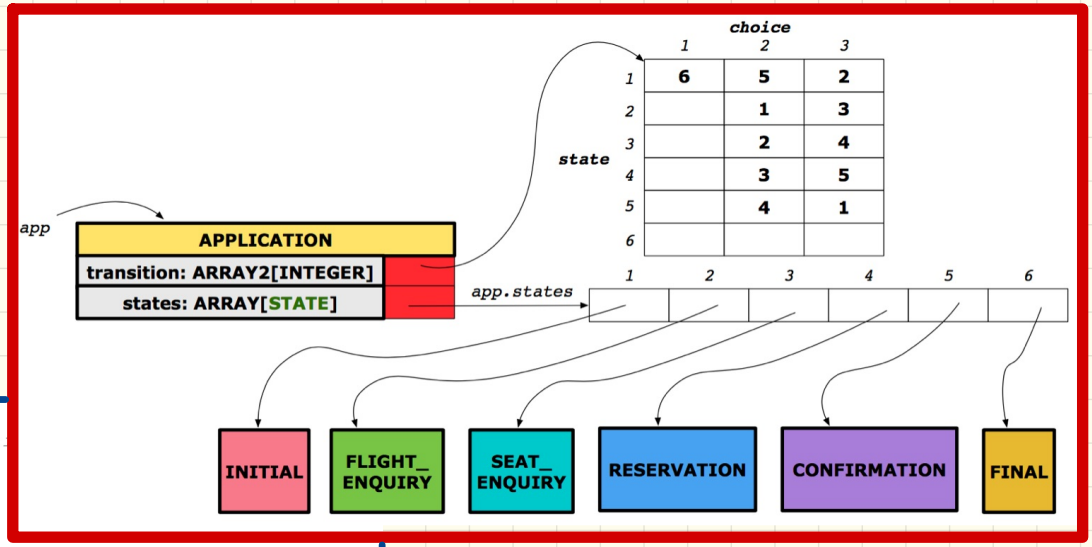
[3]

[6]

current_state

ST: STATE

State Pattern: Interactive Session

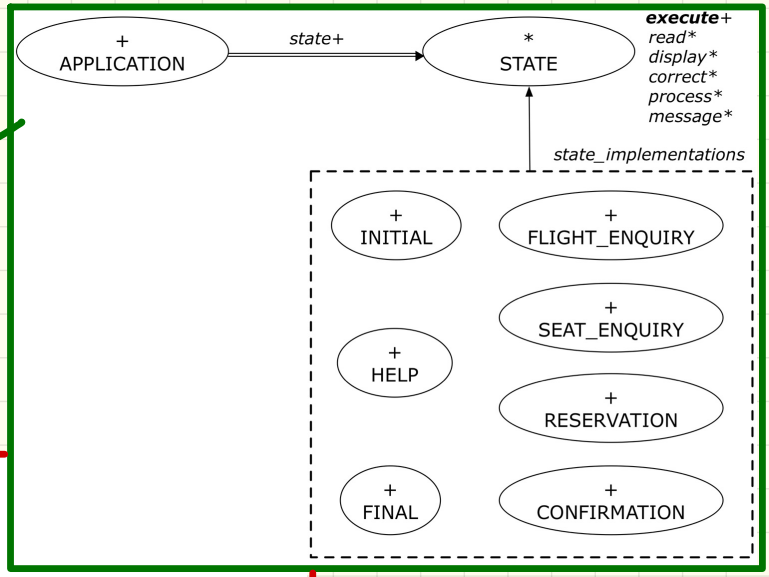


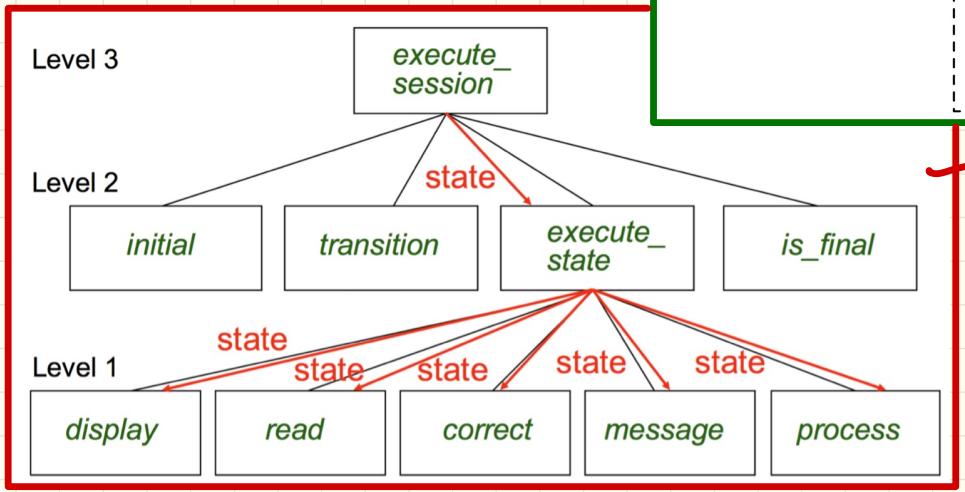

```
class APPLICATION
feature {NONE} -- Implementati
  transition: ARRAY2[INTEGER]
  states: ARRAY[STATE]
feature
  execute_session
    local
      current_state: STATE
      index: INTEGER
    do
      from
        index := initial
      until
        is_final (index)
      loop
        current_state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

Interactive System: Top-Down Design vs. OO Design

Object-Oriented

current_state: **STATE**
current_state.*execute*





Top-Down

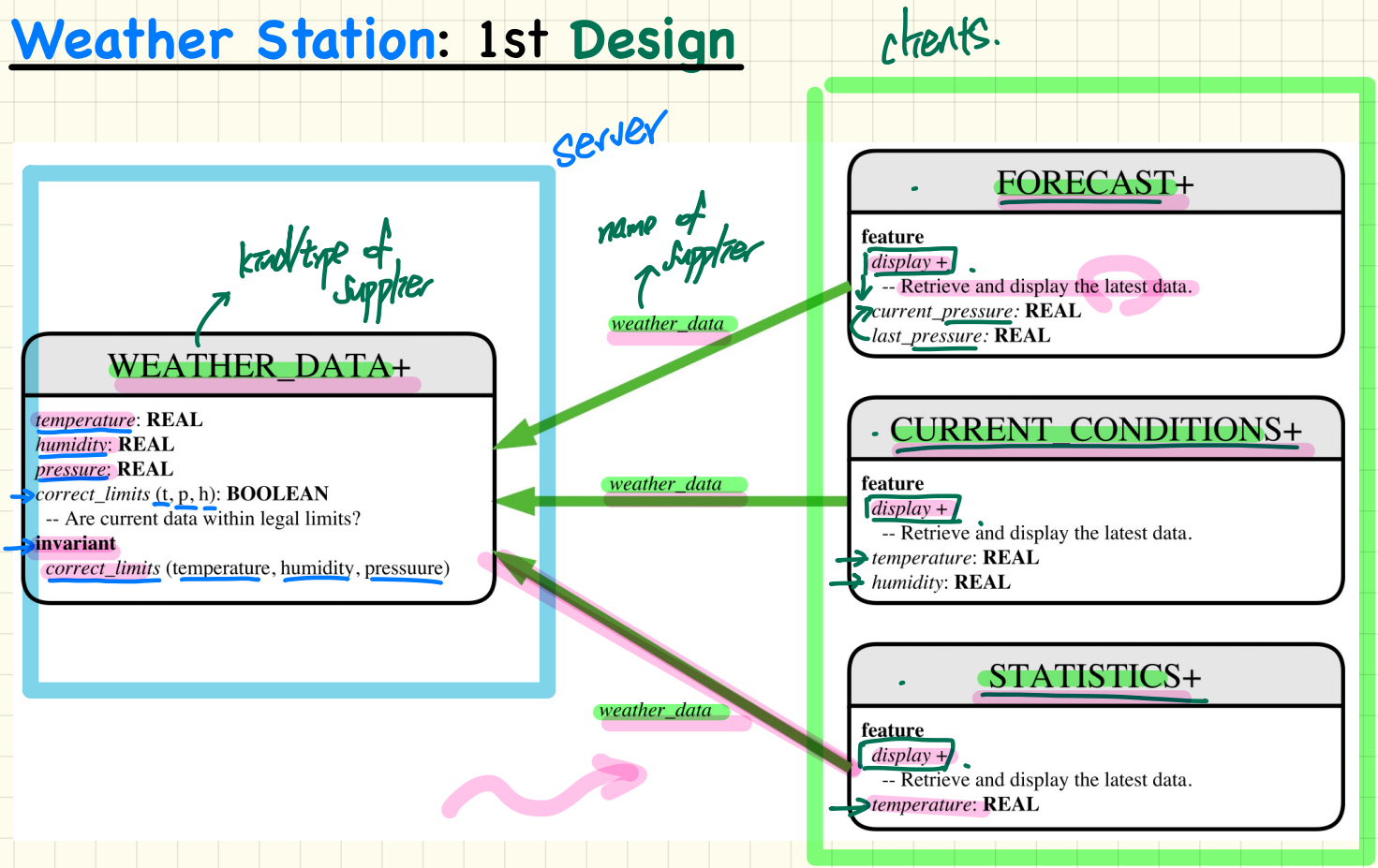
current_state: INTEGER
execute_state(current_stste)

Lecture 9

Part 1

Design 1 - Remote Procedure Calls

Weather Station: 1st Design



Weather Station:

1st Implementation

```
class WEATHER_DATA create make
feature -- Data
  temperature: REAL
  humidity: REAL
  pressure: REAL
feature -- Queries
  correct_limits(t,p,h: REAL): BOOLEAN
    ensure
      Result implies -36 <=t and t <= 60
      Result implies 50 <= p and p <= 110
      Result implies 0.8 <= h and h <= 100
feature -- Commands
  make (t, p, h: REAL)
    require
      correct_limits(temperature, pressure, humidity)
    ensure
      temperature = t and pressure = p and humidity = h
invariant
  correct_limits(temperature, pressure, humidity)
end
```

```
class FORECAST create make
feature -- Attributes
  current_pressure: REAL
  last_pressure: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do last_pressure := current_pressure
       current_pressure := weather_data.pressure
    end
  display
    do update
```

```
class CURRENT_CONDITIONS create make
feature -- Attributes
  temperature: REAL
  humidity: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = wd
  update
    do temperature := weather_data.temperature
       humidity := weather_data.humidity
    end
  display
    do update
```

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

Weather Station:

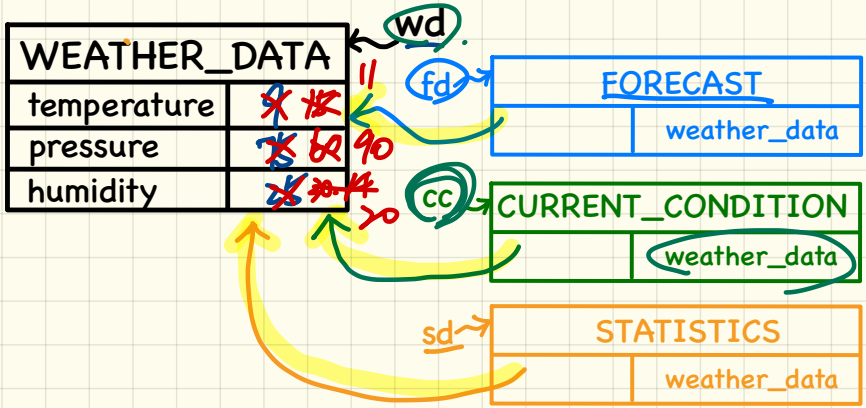
Testing 1st Design

```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd) ; create fd.make (wd) ; create sd.make (wd)

       wd.set_measurements (15, 60, 30.4)
       cc.display ; fd.display ; sd.display
       cc.display ; fd.display ; sd.display
       wd.set_measurements (11, 90, 20)
       cc.display ; fd.display ; sd.display
  end
end
```

```
class FORECAST create make
feature -- Attributes
  current_pressure: REAL
  last_pressure: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do last_pressure := current_pressure
       current_pressure := weather_data.pressure
    end
  display
    do update
```

```
class CURRENT_CONDITIONS create make
feature -- Attributes
  temperature: REAL
  humidity: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = wd
  update
    do temperature := weather_data.temperature
       humidity := weather_data.humidity
    end
  display
    do update
```

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



Lecture 9

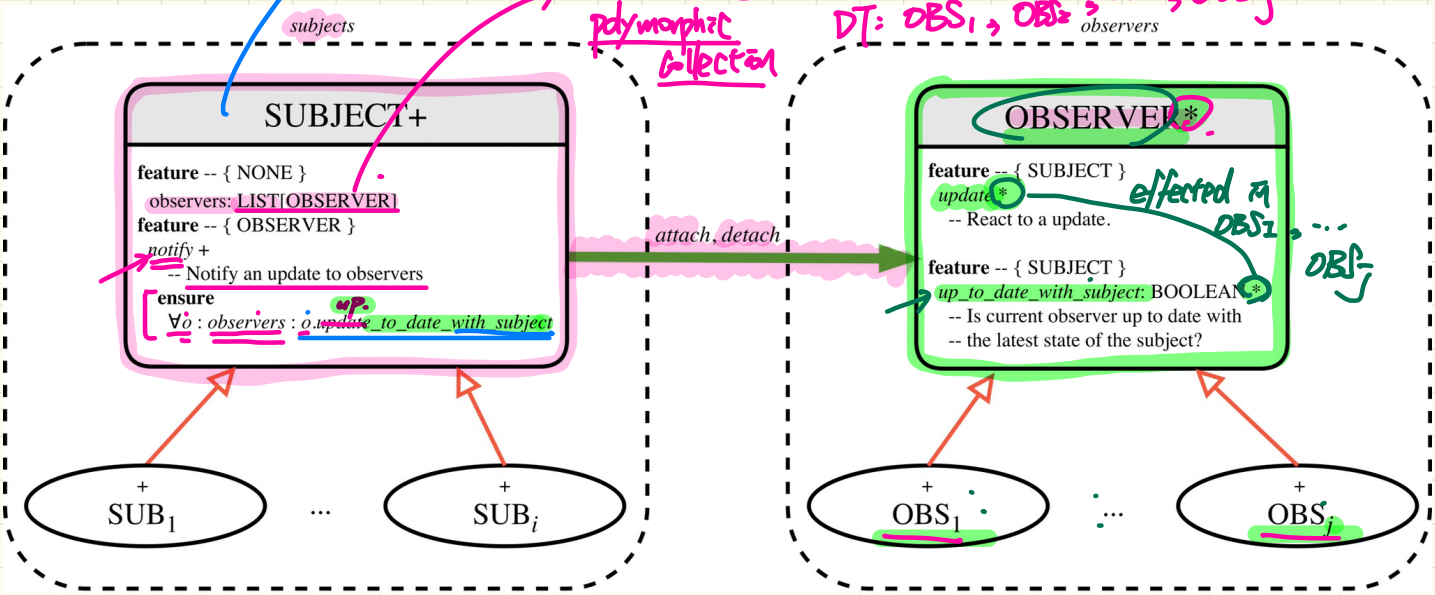
Part 2

Design 2 - Observer Design Pattern

The Observer Pattern

attach (obs: OBSERVER)
detach (obs: OBSERVER)

observers[i] has ST: OBSERVER
polymorphic collection
DT: OBS_1, OBS_2, ..., OBS_j

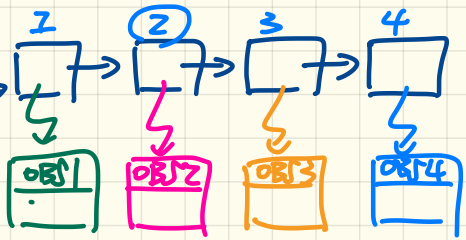


observers[i].up_to_date

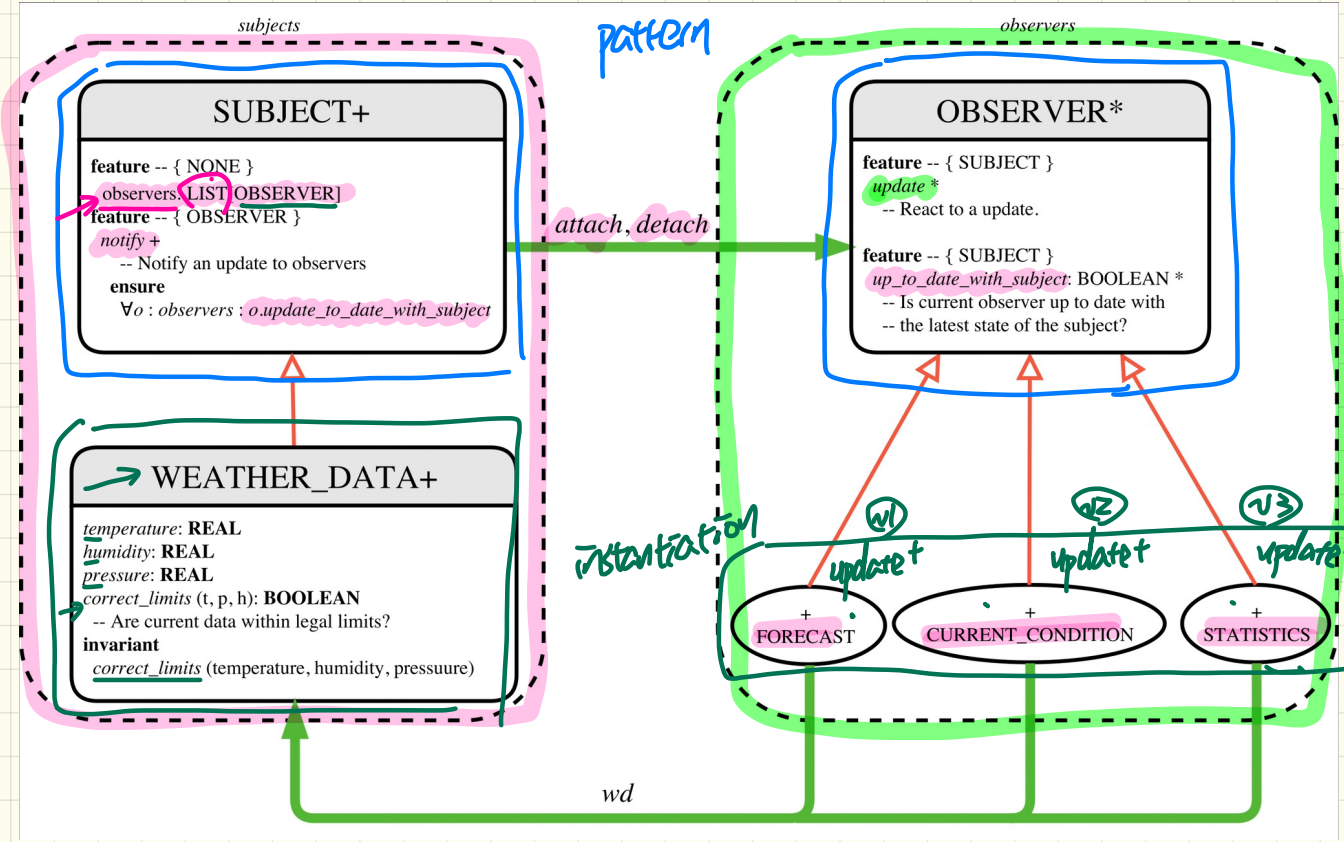
DT: OBS_i

dynamic binding ⇒ version in OBS_i

observers



Observer Pattern: Application to Weather Station



Weather Station: Subject

```
class SUBJECT create make
feature -- Attributes
  observers : LIST[OBSERVER]
feature -- Commands
  make
    do create {LINKED_LIST[OBSERVER]} observers.make
    ensure no_observers: observers.count = 0 end
feature -- Invoked by an OBSERVER
  attach (o: OBSERVER) -- Add 'o' to the observers
    require not_yet_attached: not observers.has (o)
    ensure is_attached: observers.has (o) end
  detach (o: OBSERVER) -- Add 'o' to the observers
    require currently_attached: observers.has (o)
    ensure is_attached: not observers.has (o) end
feature -- invoked by a SUBJECT
  notify -- Notify each attached observer about the update.
    do across observers as cursor loop cursor.item.update end
    ensure all_views_updated:
      across observers as o all o.item.up_to_date_with_subject end
    end
end
```

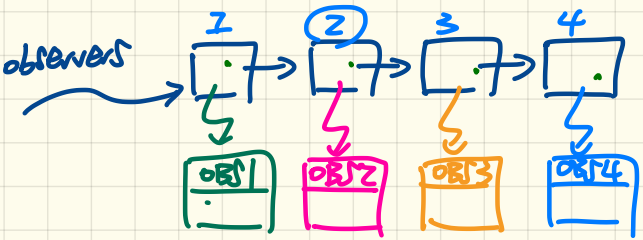
extend.

dynamic binding

ST: OBSERVER

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

rename make as make_subject end



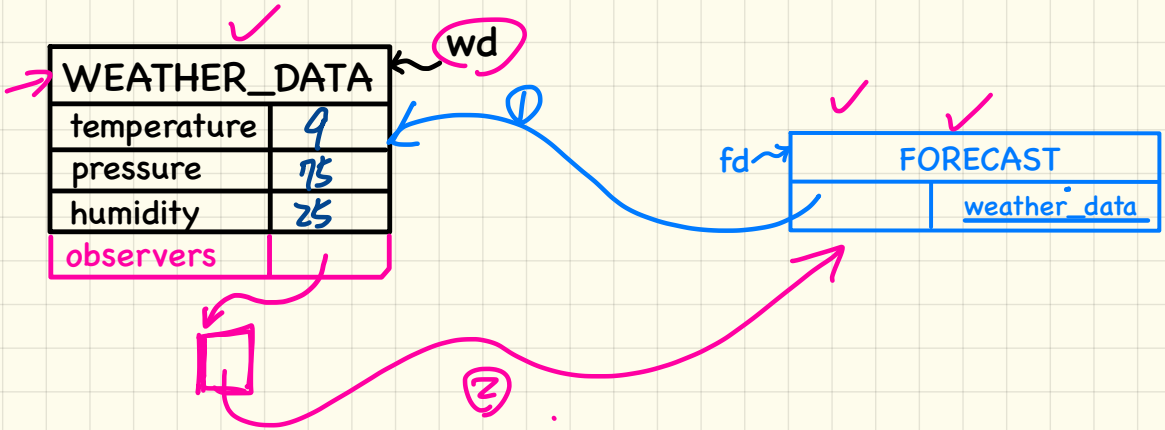
Initializing an Observer

```
class FORECAST
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
```

weather_data := a_weather_data

~~a_weather_data.observers.extend(Current)~~ ✗

a_weather_data.attach(Current)



Weather Station: Observers

```
deferred class
  OBSERVER
feature -- To be effected by a descendant
  up_to_date_with_subject: BOOLEAN
      -- Is this observer up to date with its subject?
    deferred
    end

  update
      -- Update the observer's view of 's'
    deferred
    ensure
      up_to_date_with_subject: up_to_date_with_subject
    end
end
```

```
class FORECAST
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
feature -- Queries
  up_to_date_with_subject: BOOLEAN
    ensure then
      Result = current_pressure = weather_data.pressure
  update
    do -- Same as 1st design; Called only on demand
    end
```

```
class CURRENT_CONDITIONS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
feature -- Queries
  up_to_date_with_subject: BOOLEAN
    ensure then Result = temperature = weather_data.temperature and
                         humidity = weather_data.humidity
  update
    do -- Same as 1st design; Called only on demand
    end
```

```
class STATISTICS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
feature -- Queries
  up_to_date_with_subject: BOOLEAN
    ensure then
      Result = current_temperature = weather_data.temperature
  update
    do -- Same as 1st design; Called only on demand
    end
```

Weather Station: Testing the Observer Pattern

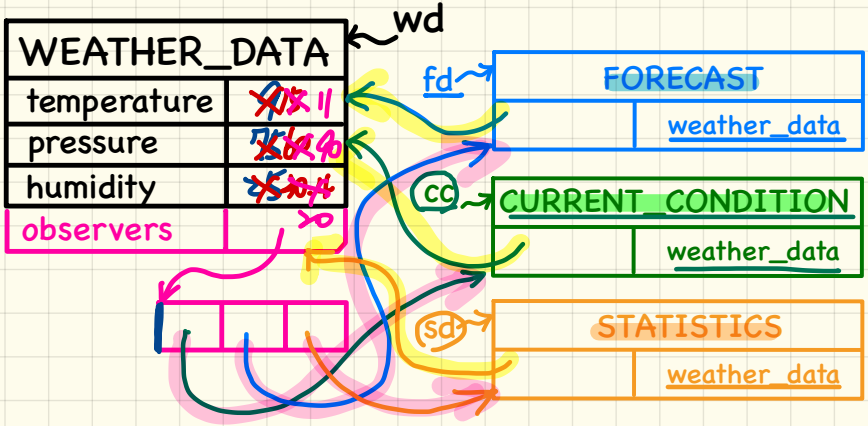
```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd) ; create fd.make (wd) ; create sd.make(wd)

       wd.set_measurements (15, 60, 30.4)
       wd.notify
       cc.display ; fd.display ; sd.display
       cc.display ; fd.display ; sd.display

       wd.set_measurements (11, 90, 20)
       wd.notify
       cc.display ; fd.display ; sd.display
  end
end
```

update not necessary anymore.

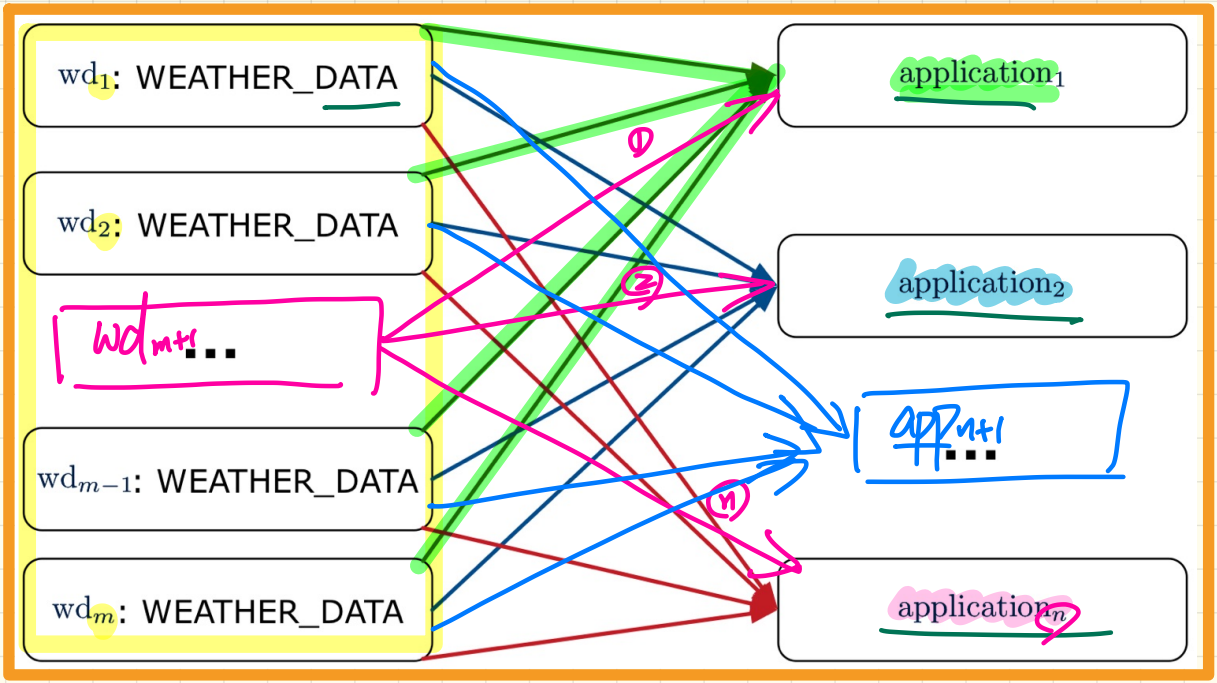
```
class FORECAST
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```

```
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
```

```
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
```



Multiple Subjects vs. Multiple Observers: Observer Pattern



Q1. Overall Complexity? O(m * n)

Q2. Complexity of adding a new subject? O(n)

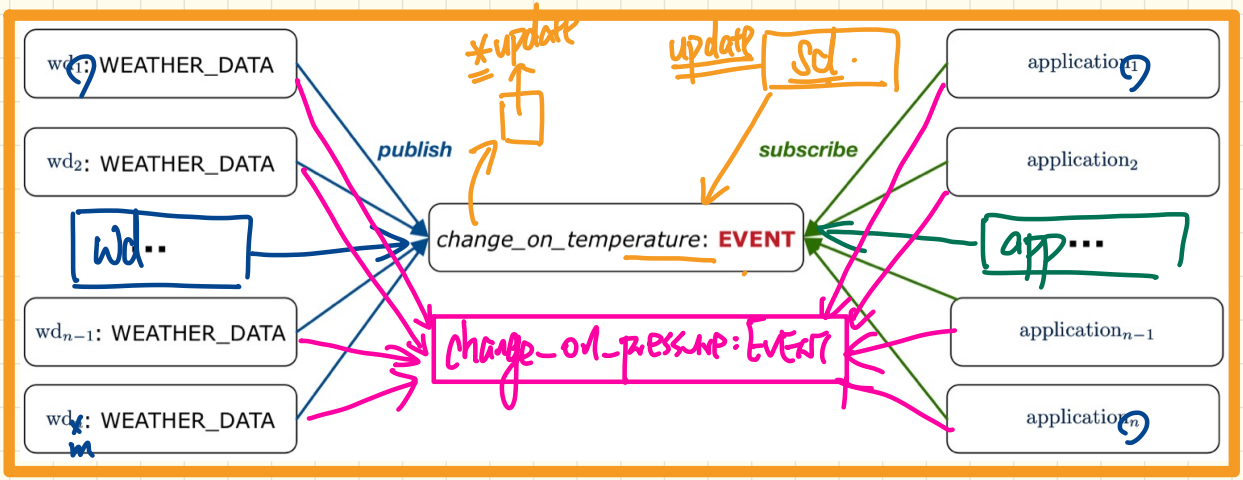
Q3. Complexity of adding a new observer? O(m)

Lecture 9

Part 3

Design 3 - Event-Driven Design

Multiple Subjects vs. Multiple Observers: Event-Driven Design



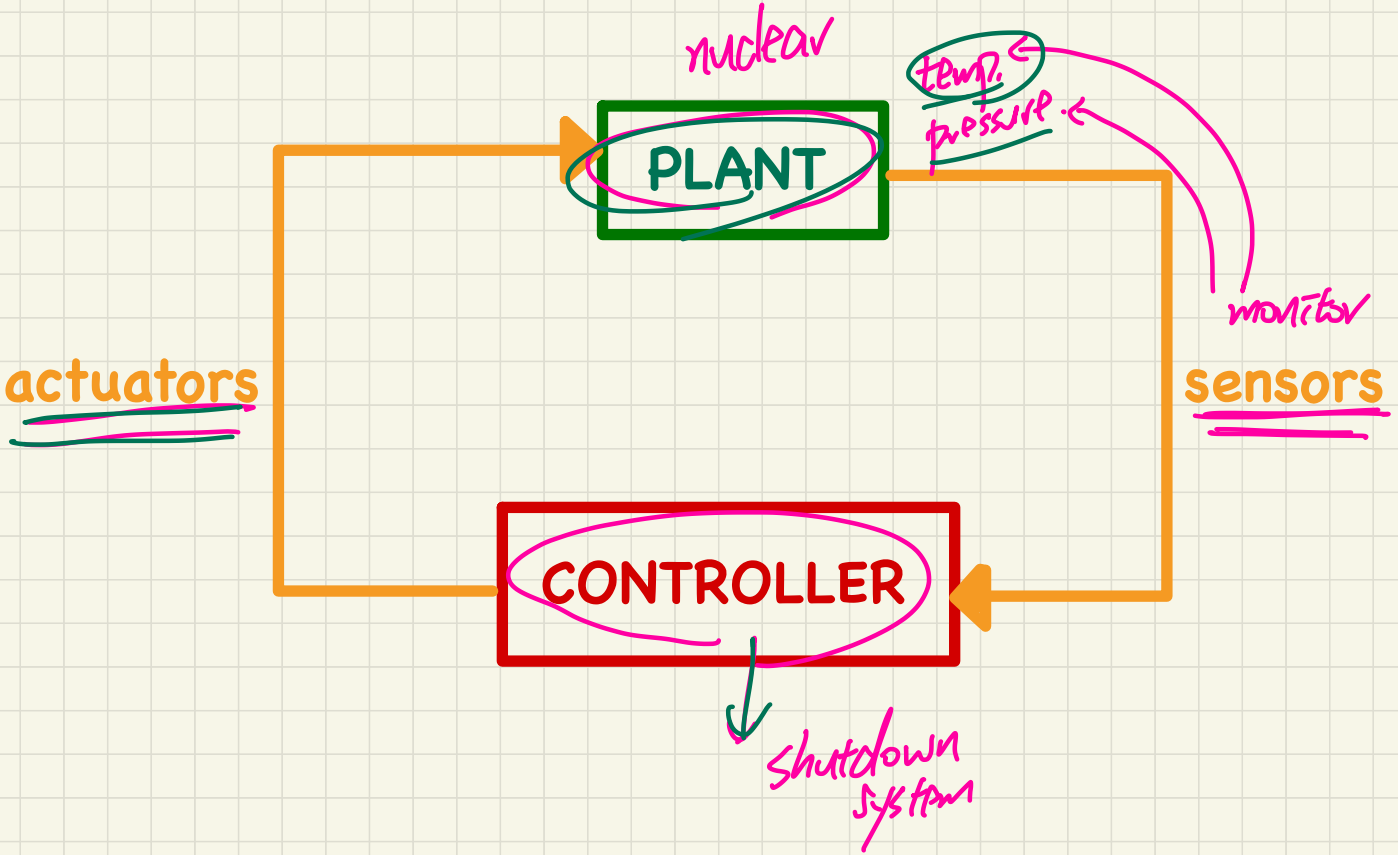
Q1. Overall **Complexity**? $O(m+n)$

Q2. **Complexity** of adding a new **observer**? $O(1)$

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

Q4. **Complexity** of adding a new **event type**? $O(m+n)$

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

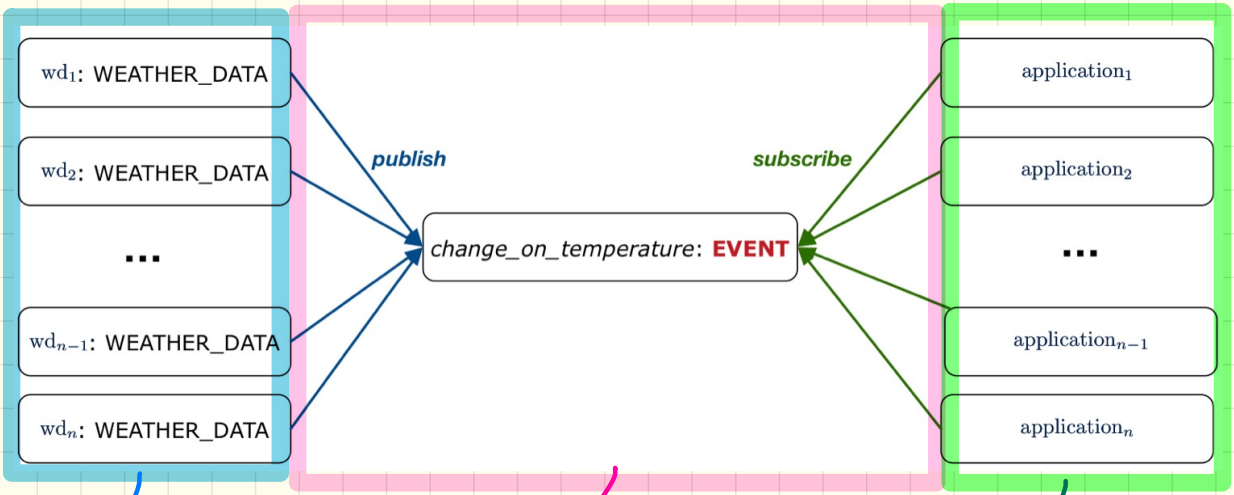


Lecture 9

Part 4

Event-Driven Design in Java

Implementing the Event-Driven Design



Module: Subject

Module: Event.

Module: Observer/ Listener.

Event-Driven Design in Java

```
public class WeatherStation {
  public static void main(String[] args) {
    WeatherData wd = new WeatherData(9, 75, 25);
    CurrentConditions cc = new CurrentConditions();
    System.out.println("=======");
    wd.setMeasurements(15, 60, 30.4);
    cc.display();
    System.out.println("=======");
    wd.setMeasurements(11, 90, 20);
    cc.display();
  } }
```

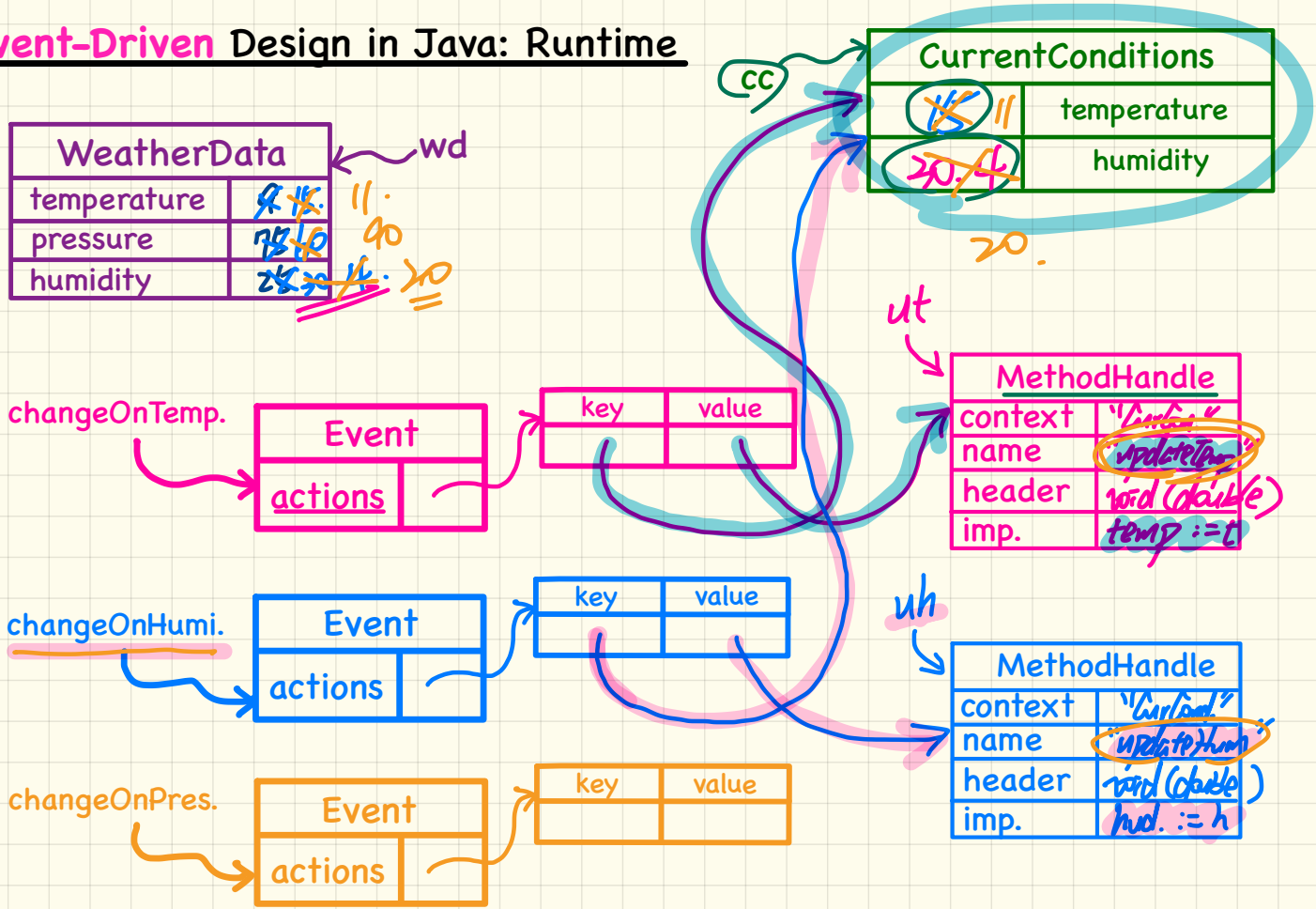
```
public class CurrentConditions {
  private double temperature; private double humidity;
  public void updateTemperature(double t) { temperature = t; }
  public void updateHumidity(double h) { humidity = h; }
  public CurrentConditions() {
    MethodHandles.Lookup lookup = MethodHandles.lookup();
    try {
      MethodHandle ut = lookup.findVirtual(
        this.getClass(), "updateTemperature",
        MethodType.methodType(void.class, double.class));
      WeatherData.changeOnTemperature.subscribe(this, ut);
      MethodHandle uh = lookup.findVirtual(
        this.getClass(), "updateHumidity",
        MethodType.methodType(void.class, double.class));
      WeatherData.changeOnHumidity.subscribe(this, uh);
    } catch (Exception e) { e.printStackTrace(); }
  }
  public void display() {
    System.out.println("Temperature: " + temperature);
    System.out.println("Humidity: " + humidity); } }
```

```
public class WeatherData {
  private double temperature;
  private double pressure;
  private double humidity;
  public WeatherData(double t, double p, double h) {
    setMeasurements(t, h, p);
  }
  public static Event changeOnTemperature = new Event();
  public static Event changeOnHumidity = new Event();
  public static Event changeOnPressure = new Event();
  public void setMeasurements(double t, double h, double p) {
    temperature = t;
    humidity = h;
    pressure = p;
    changeOnTemperature.publish(temperature);
    changeOnHumidity.publish(humidity);
    changeOnPressure.publish(pressure);
  }
}
```

```
public class Event {
  Hashtable<Object, MethodHandle> listenersActions;
  Event() { listenersActions = new Hashtable<>(); }
  void subscribe(Object listener, MethodHandle action) {
    listenersActions.put(listener, action);
  }
  void publish(Object arg) {
    for (Object listener : listenersActions.keySet()) {
      MethodHandle action = listenersActions.get(listener);
      try {
        action.invokeWithArguments(listener, arg);
      } catch (Throwable e) { }
    }
  }
}
```

ut or uh

Event-Driven Design in Java: Runtime

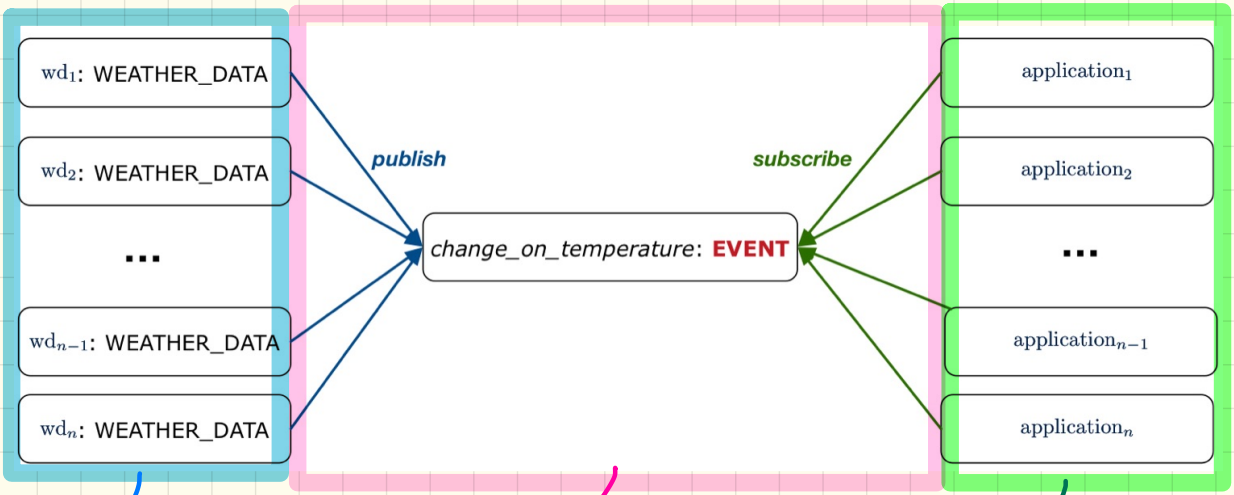


Lecture 9

Part 5

Event-Driven Design in Eiffel

Implementing the Event-Driven Design



Module: Subject

Module: Event.

Module: Observer/ Listener.

Event-Driven Design in Eiffel

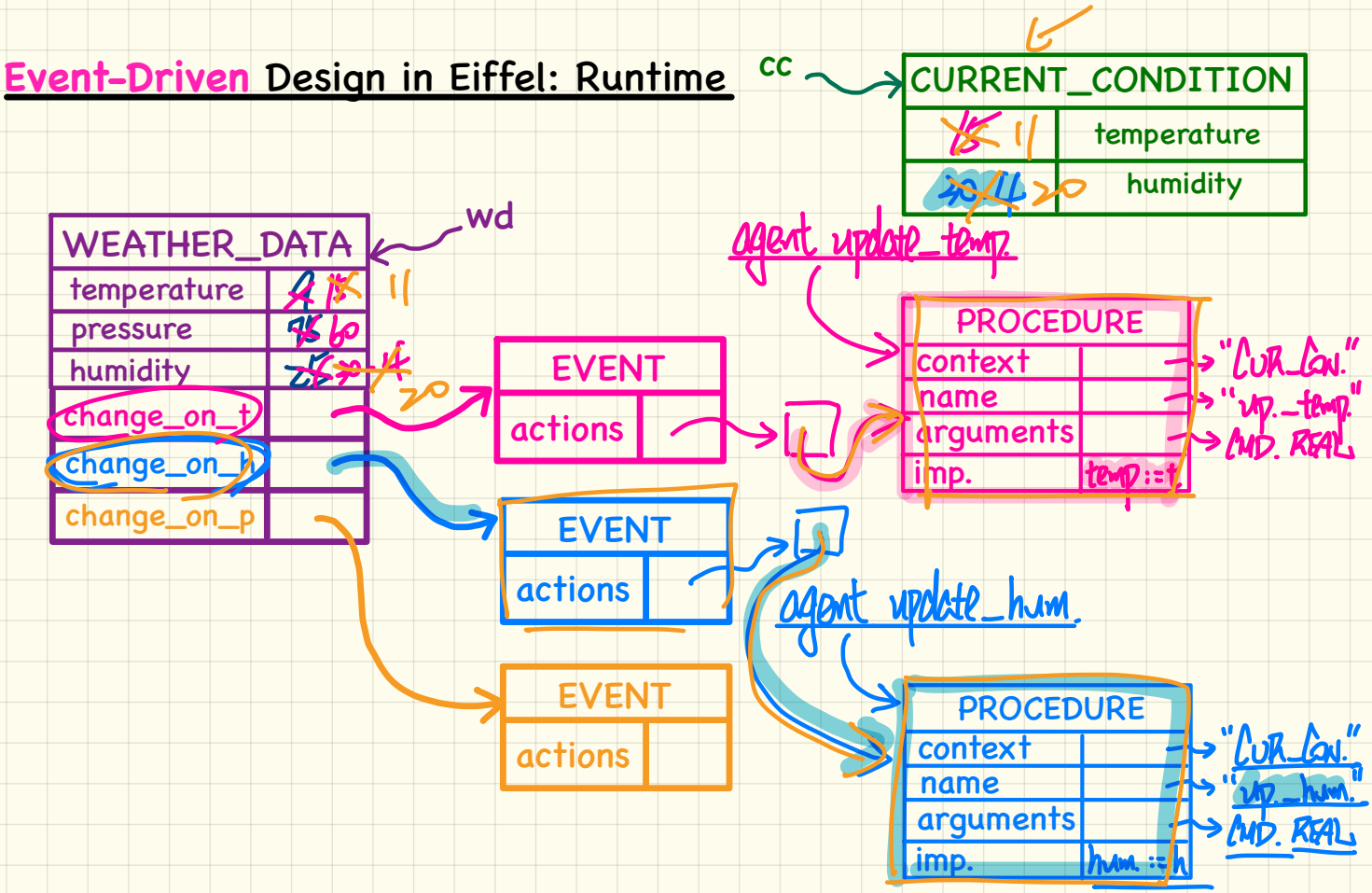
```
class WEATHER_STATION create make
feature
  cc: CURRENT_CONDITIONS
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd)
       wd.set_measurements (15, 60, 30.4)
       cc.display
       wd.set_measurements (11, 90, 20)
       cc.display
    end
end
```

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

```
class EVENT [ARGUMENTS -> TUPLE]
create make
feature -- Initialization
  actions: LINKED_LIST[PROCEDURE[ARGUMENTS]]
  make do create actions.make end
feature
  subscribe (an_action: PROCEDURE[ARGUMENTS])
    require action_not_already_subscribed: not actions.h
    do actions.extend (an_action)
    ensure action_subscribed: action.has(an_action) end
  publish (args: ARGUMENTS)
    do from actions.start until actions.after
       loop actions.item.call (args) ; actions.forth end
    end
end
```

```
class WEATHER_DATA
create make
feature -- Measurements
  temperature: REAL ; humidity: REAL ; pressure: REAL
  correct_limits(t,p,h: REAL): BOOLEAN do ... end
  make (t, p, h: REAL) do ... end
feature -- Event for data changes
  change_on_temperature : EVENT[TUPLE[REAL]]once create Result end
  change_on_humidity : EVENT[TUPLE[REAL]]once create Result end
  change_on_pressure : EVENT[TUPLE[REAL]]once create Result end
feature -- Command
  set_measurement (t, p, h: REAL)
    require correct_limits(t,p,h)
    do temperature := t ; pressure := p ; humidity := h
       change_on_temperature.publish ([t])
       change_on_humidity.publish ([p])
       change_on_pressure.publish ([h])
    end
invariant correct_limits(temperature, pressure, humidity) end
```

Event-Driven Design in Eiffel: Runtime



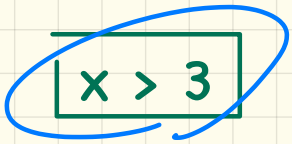
Lecture 10

Part 1

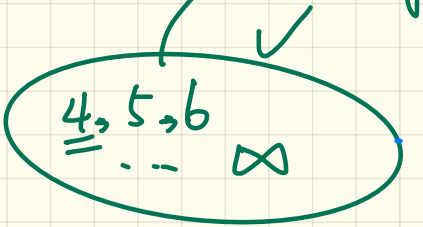
Contracts - Require Less vs. Ensure More

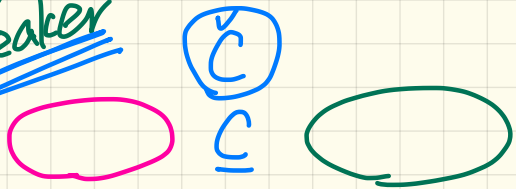
Assertions: Weak vs. Strong

Predicates → Boolean values
→ set of satisfying values



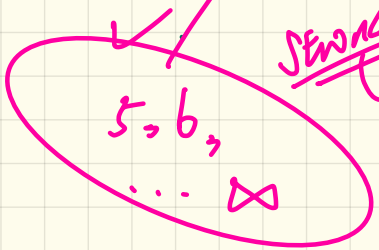
more accommodating, larger, weaker

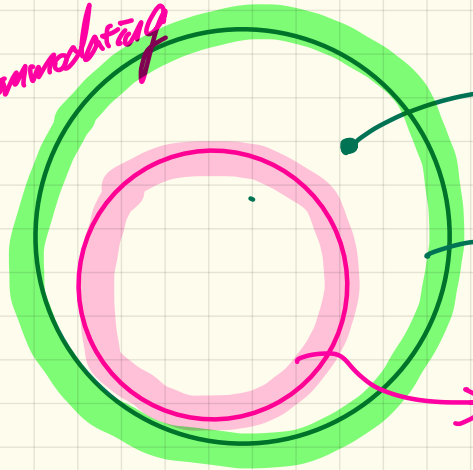




Weakest?
Strongest?

less accommodating, smaller, stronger

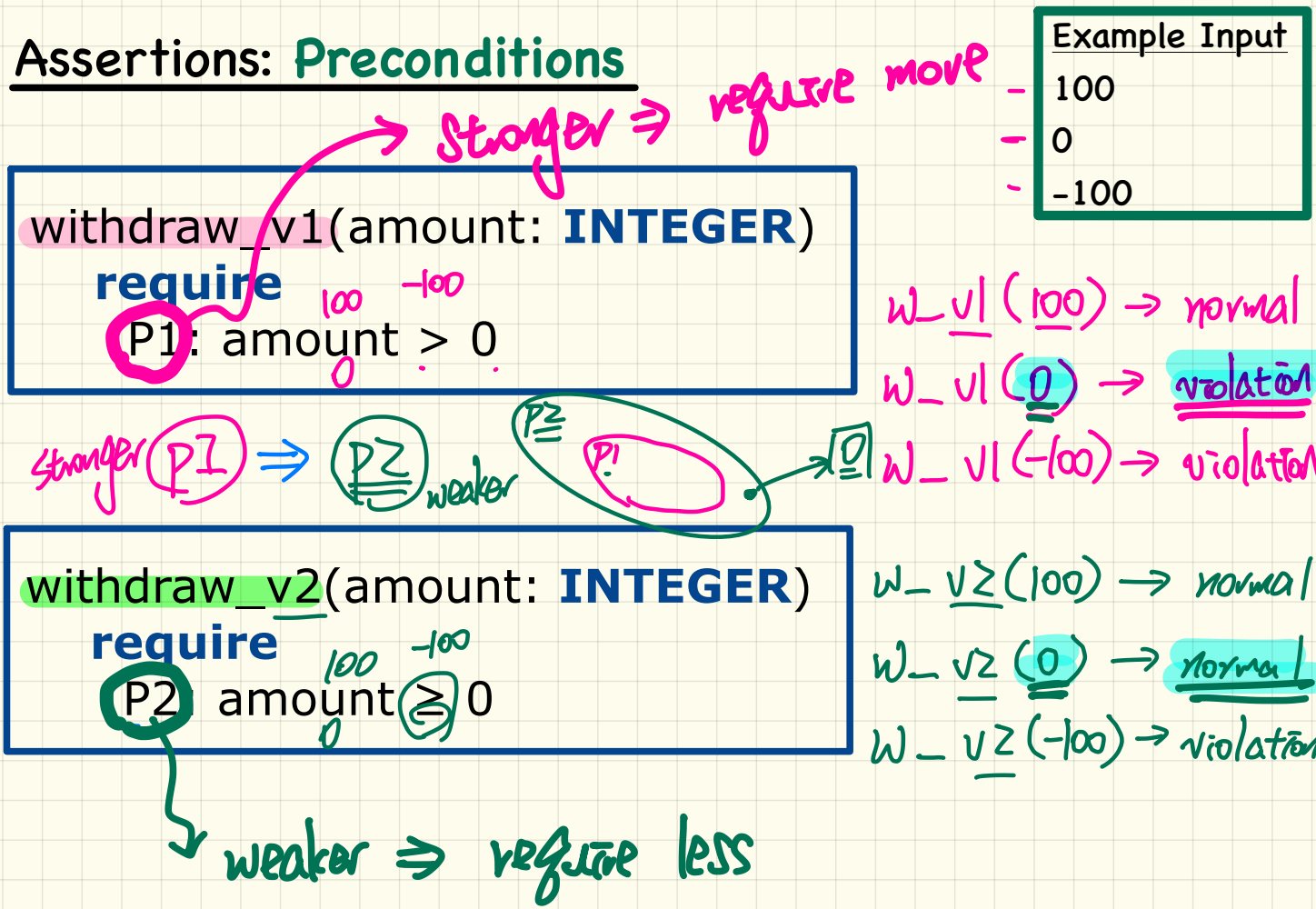


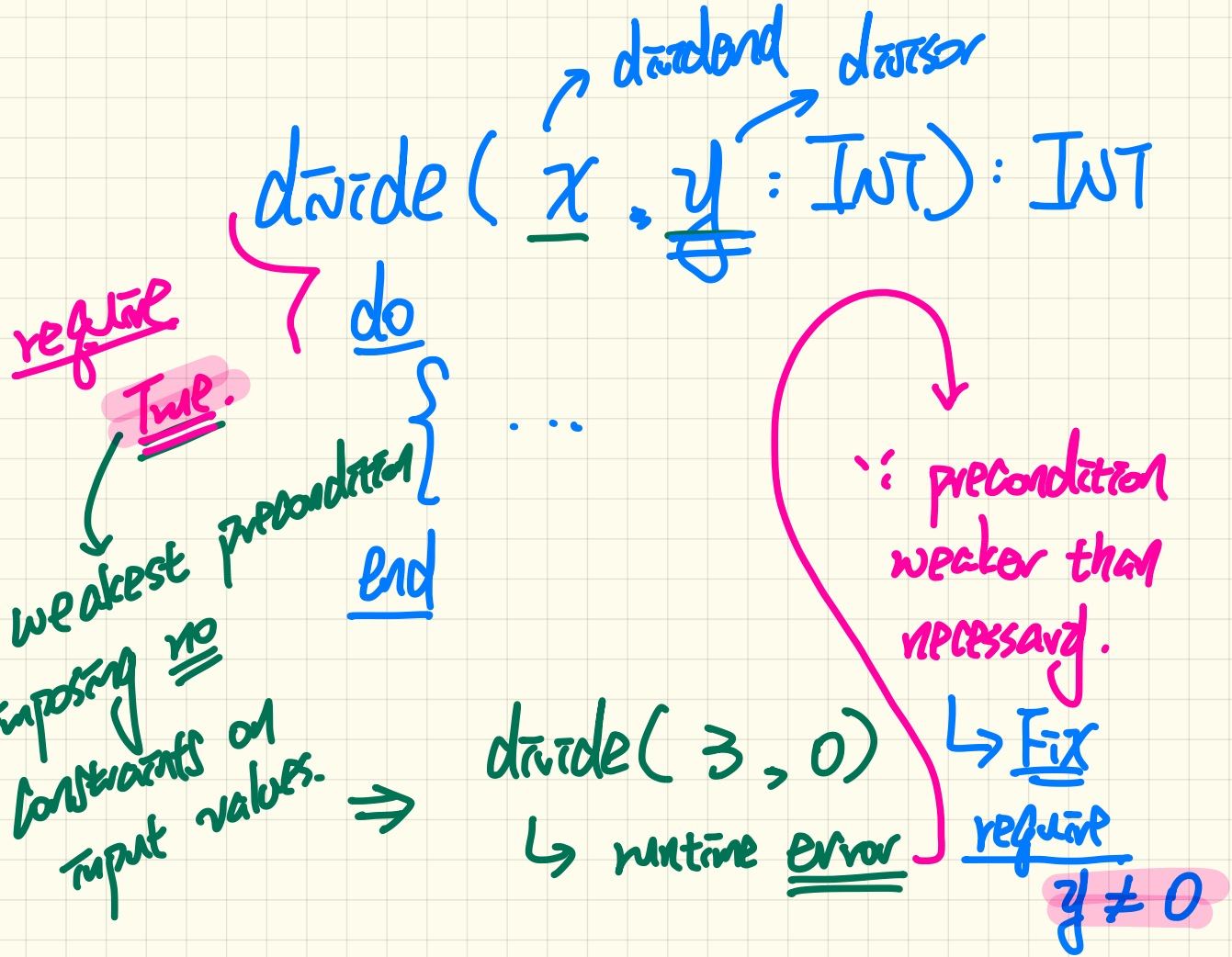


x > 3 ⇝ weaker.

x > 4 ⇝ stronger.

Assertions: Preconditions




dividend
divisor
divide (x , y : INT) : INT
require
True.
do
...
end
weakest precondition
imposing no
constraints on
input values.
⇒
divide (3 , 0)
↳ runtime error
: precondition
weaker than
necessary.
↳ Fix
require
y ≠ 0

Assertions: Postconditions

$x \wedge y \Rightarrow x \vee y$

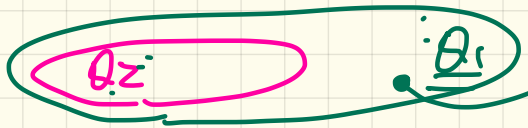
False → ϕ $\quad \phi \subseteq -$

```
f1(i: INTEGER): BOOLEAN
    ensure
      Q1: Result = (i > 0) or (i mod 2 = 0)
```

T T T

f1(79) → T
f1(-34) → T
f1(62) → T

Q2 ⇒ Q1



79
-34

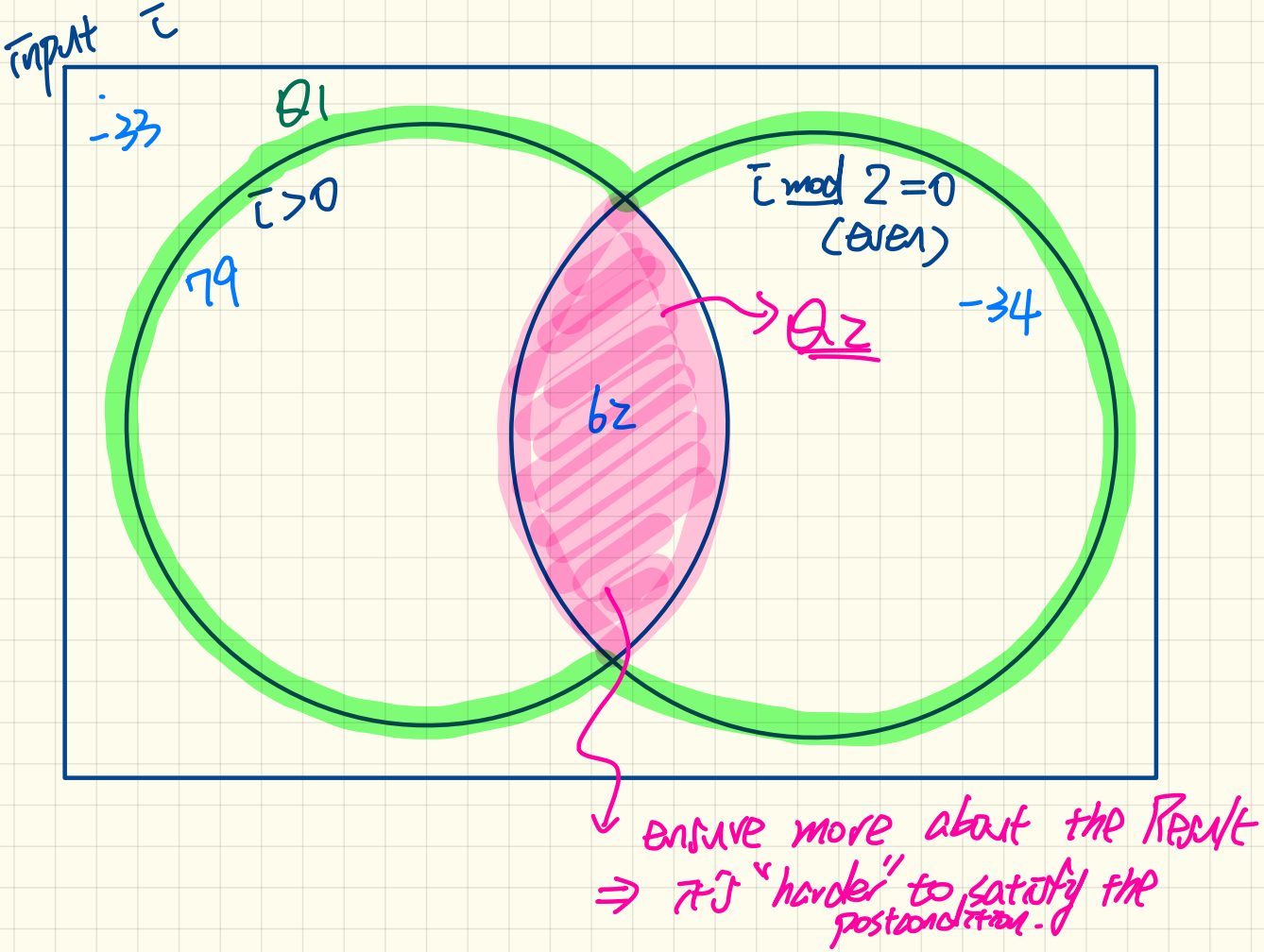
```
f2(i: INTEGER): BOOLEAN
    ensure
      Q2: Result = (i > 0) and (i mod 2 = 0)
```

F F

f2(79) → F
f2(-34) → F
f2(62) → T

What's the value of Result (output.) that will satisfy the postcondition.

Example Input
79
-34
62

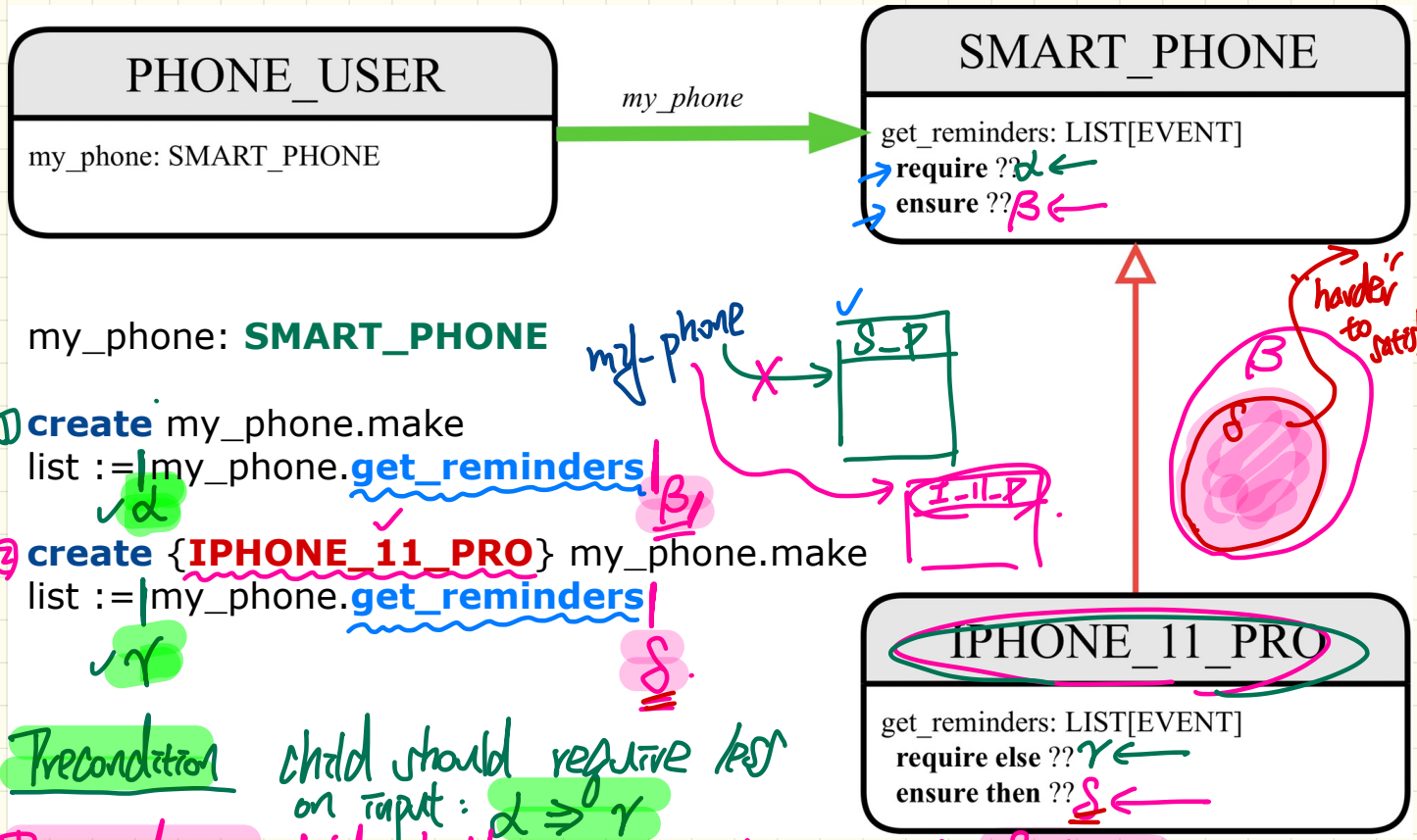

Input i
-33
Q1
i > 0
79
i mod 2 = 0
(even)
-34
Q2
62
ensure more about the Result
⇒ it's "harder" to satisfy the postcondition.

Lecture 10

Part 2

Inheritance & Contracts - Static Analysis

Subcontracting: Architectural View



Subcontracting: Example (1)

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

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

↳ poor design :(Stronger precondition

$\alpha \Rightarrow \beta$

$bl \geq 10\% \Rightarrow bl \geq 15\%$

α α γ

10, 11, 12, 13, 14, 15, 16 ... γ

C.E.

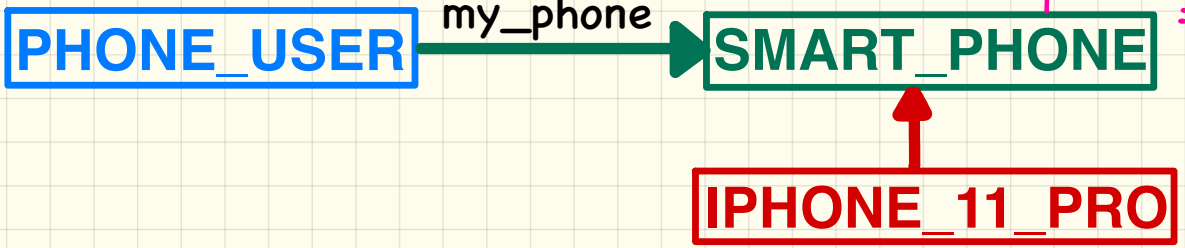
10, 11, 12, 13, 14 satisfy α but not γ

```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

create {IPHONE_11_PRO} mine.make
list := my_phone.get_reminders
```

α ✓ 12%.

γ 12%.

↳ precondition violation ⇒ shock to user.



Subcontracting: Example (2)

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

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

↳ poorly designed ⁉ ensures less

```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

create {IPHONE_11_PRO} mine.make
list := my_phone.get_reminders
```

Rule: $\delta \Rightarrow \beta$

δ: e happens today ∨ tmw

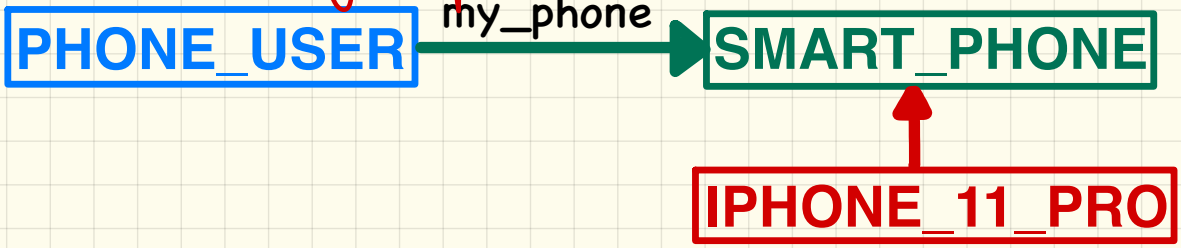
$\Rightarrow$?

β: e happens today

Imp. output tmw. e. with δ

tomor. β today

δ. tmw events only.



Lecture 10

Part 3

Inheritance & Contracts - Runtime Checks

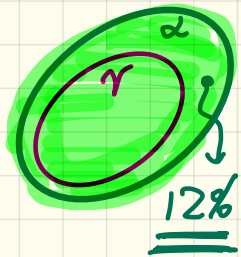
Subcontracting: Example (1)

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

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

Invalid

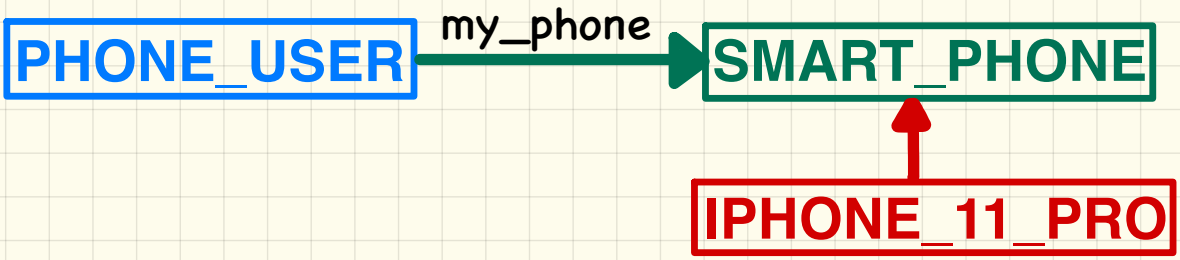
$\because \gamma \Rightarrow \alpha$



Runtime :

$\neg\alpha \vee \gamma \equiv \alpha$ (T)

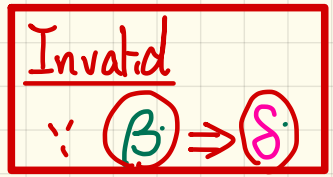
```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

create {IPHONE_11_PRO} mine.make
list := my_phone.get_reminders
```



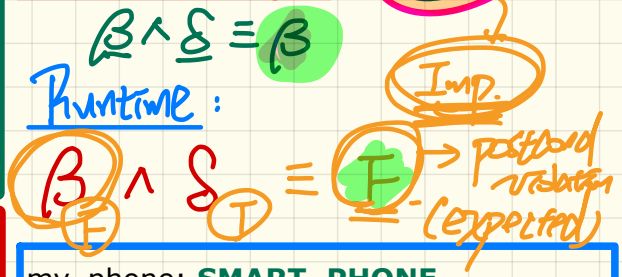
Subcontracting: Example (2)

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

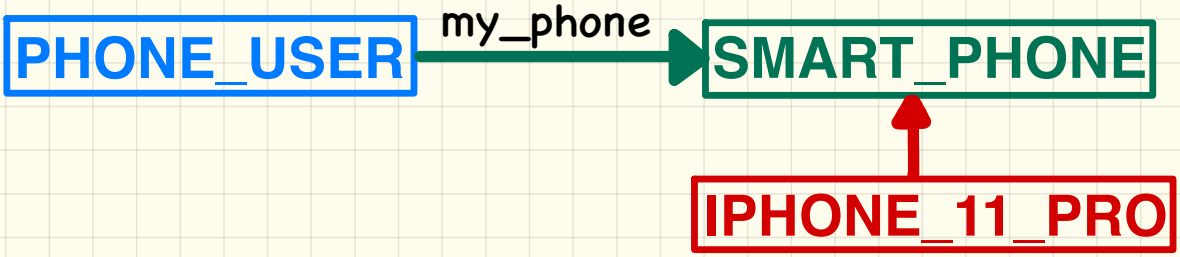
```
class IPHONE_11_PRO
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e : Result | e happens today or tomorrow
end
```



$\beta \wedge \delta \equiv \beta$

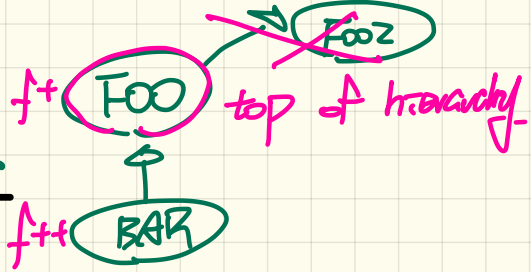



```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

create {IPHONE_11_PRO} mine.make
list := my_phone.get_reminders
```



Contract Re-Declaration:

Missing Pre-Condition in Ancestor




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

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

require True

True V new_pre

≡ True → at runtime, even if DT ⇒ BAR only check True to proceed

Contract Re-Declaration:

Missing Post-Condition in Ancestor

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

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

Stronger

ENSURE True.

Runtime.

True ∧ new_post.

≡ new_post.

When DT in BAR check new_post for postcond.

Contract Re-Declaration:

Missing Pre-Condition in Descendant

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

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

require else

~~or~~ False.

original_pre ∨ ?? False

≡ original_pre

Contract Re-Declaration:

Missing Post-Condition in Descendant

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

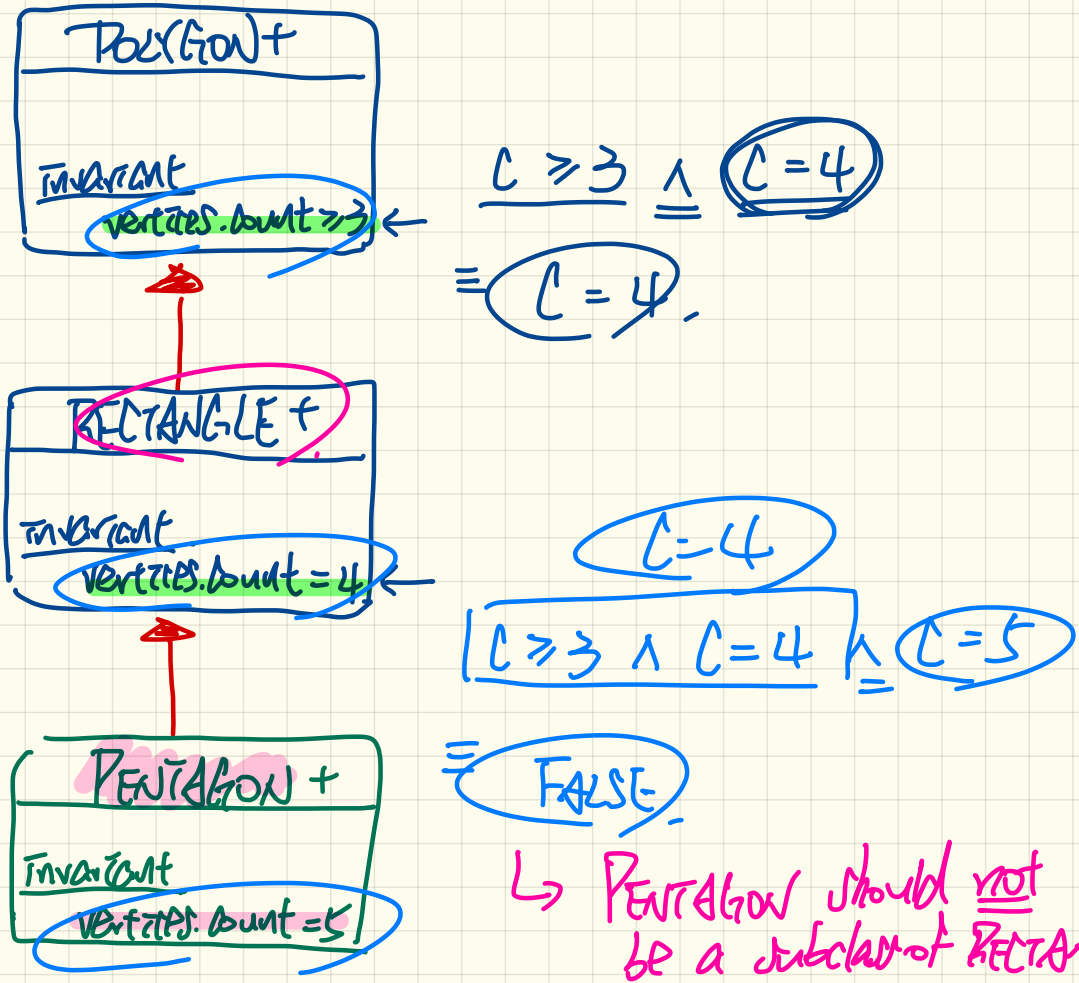
```
class BAR
inherit FOO redefine f end
  f
    do ...
  end
end
```

ensure then

~~??~~ True

original_post ∧ ?? True

≡ original_post

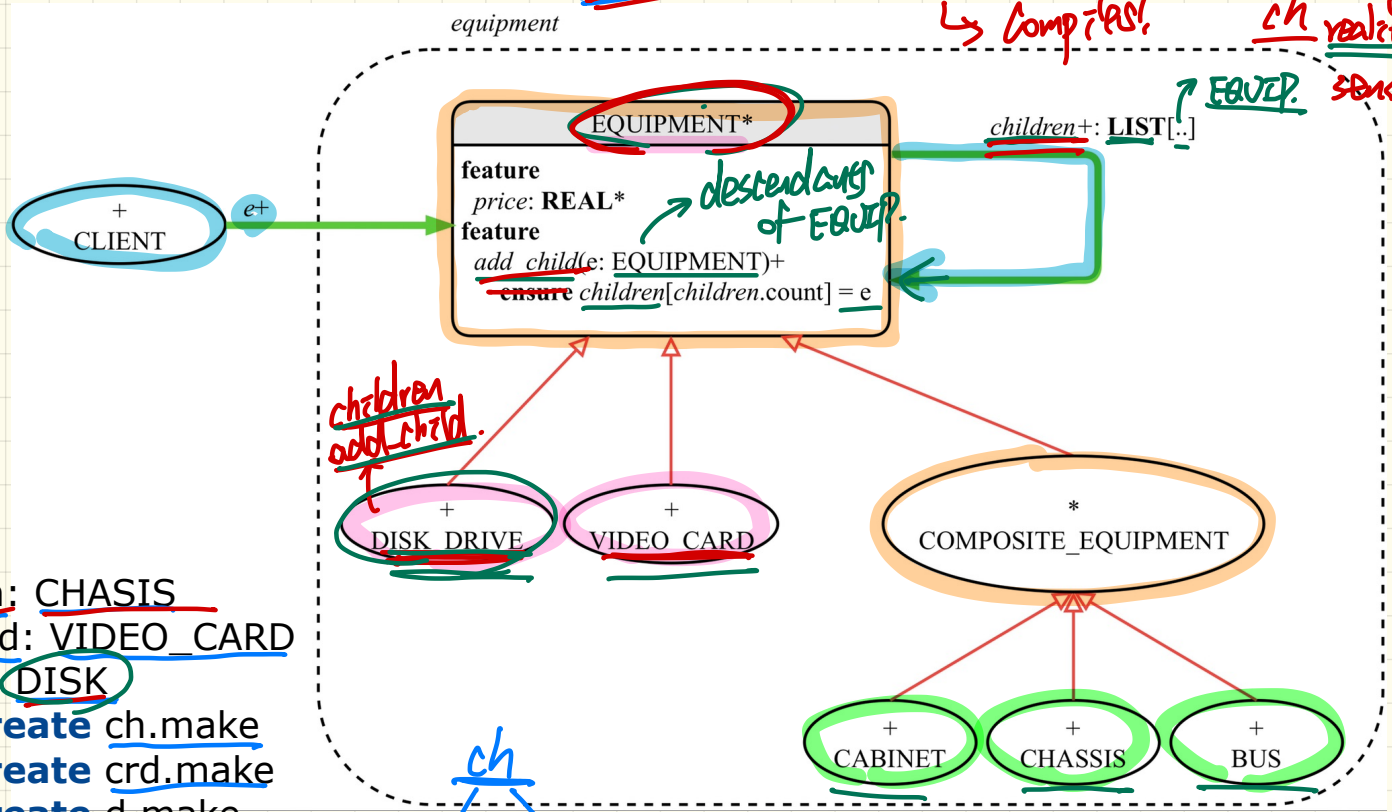

POLYGON+
INVARIANT
vertices.count >= 3
RECTANGLE+
INVARIANT
vertices.count = 4
PENTAGON+
INVARIANT
vertices.count = 5
C >= 3 ∧ C = 4
≡ C = 4
C = 4
C >= 3 ∧ C = 4 ∧ C = 5
≡ FALSE
↳ PENTAGON should not be a subclass of RECTAN.

Lecture 10

Part 4

Recursive Systems - Design Attempts

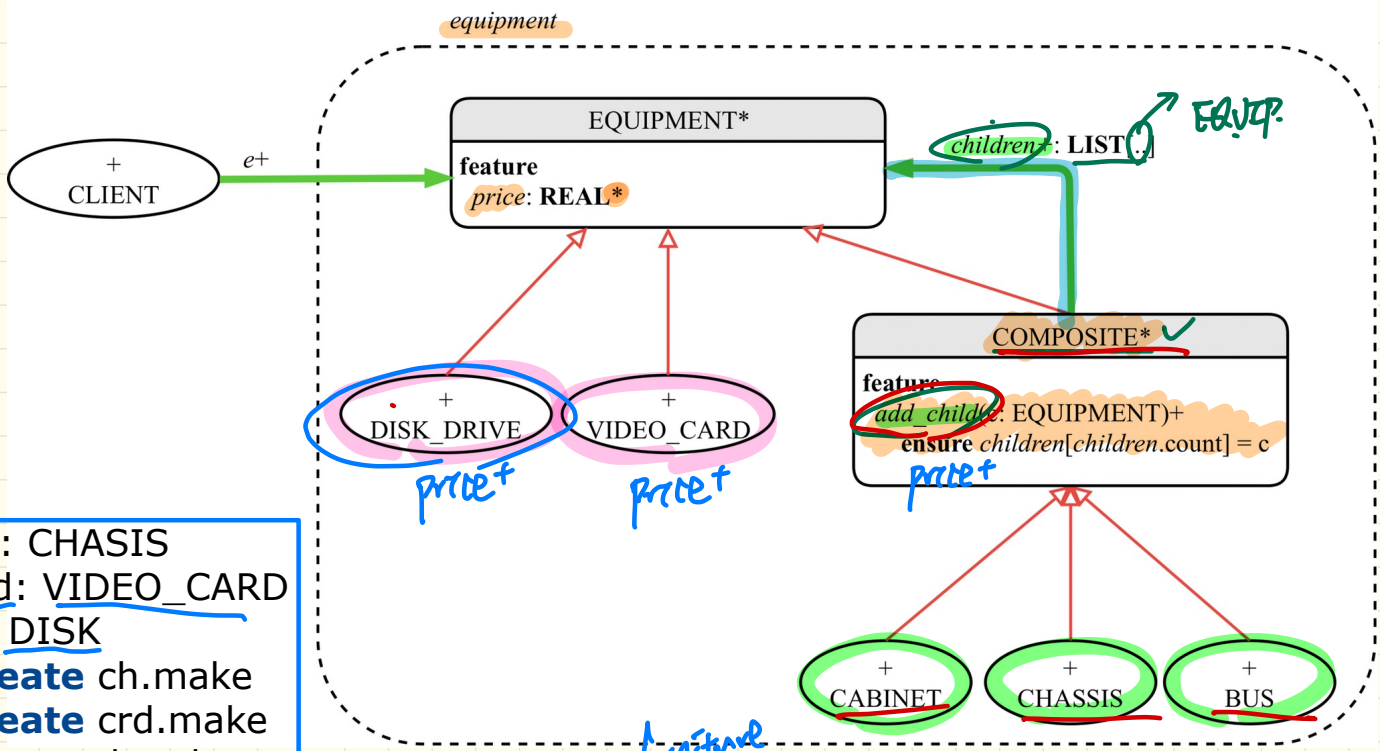
First Design Attempt




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

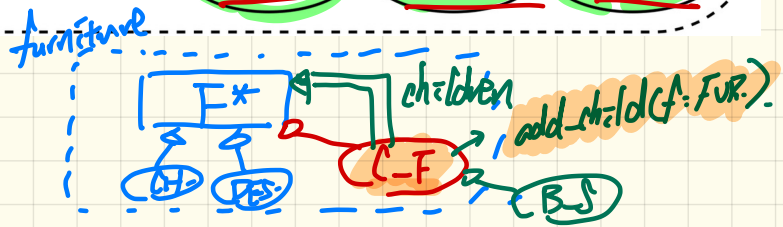
Second Design Attempt

SCP.




```
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)
```

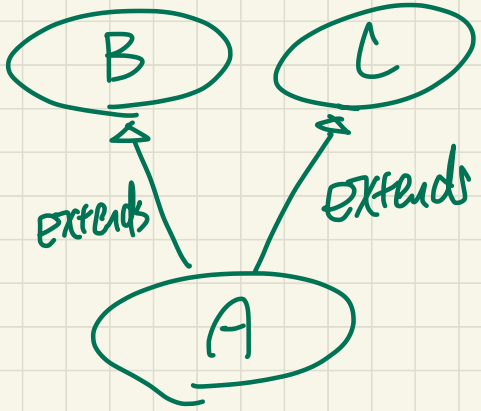
→ Compile? not.



Lecture 10

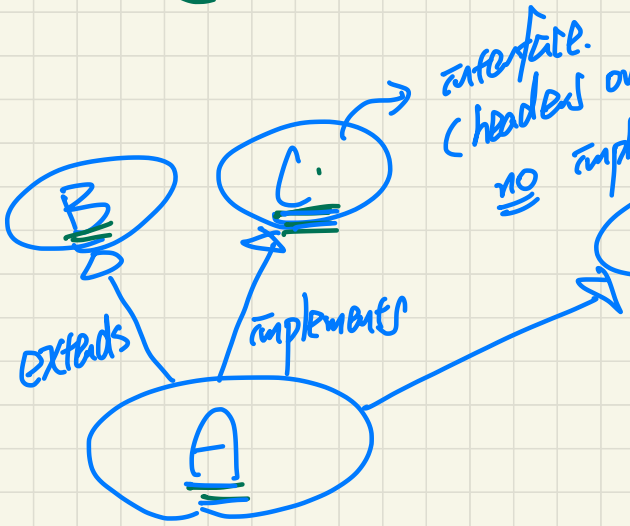
Part 5

Multiple Inheritance



class A extends B, C ✗

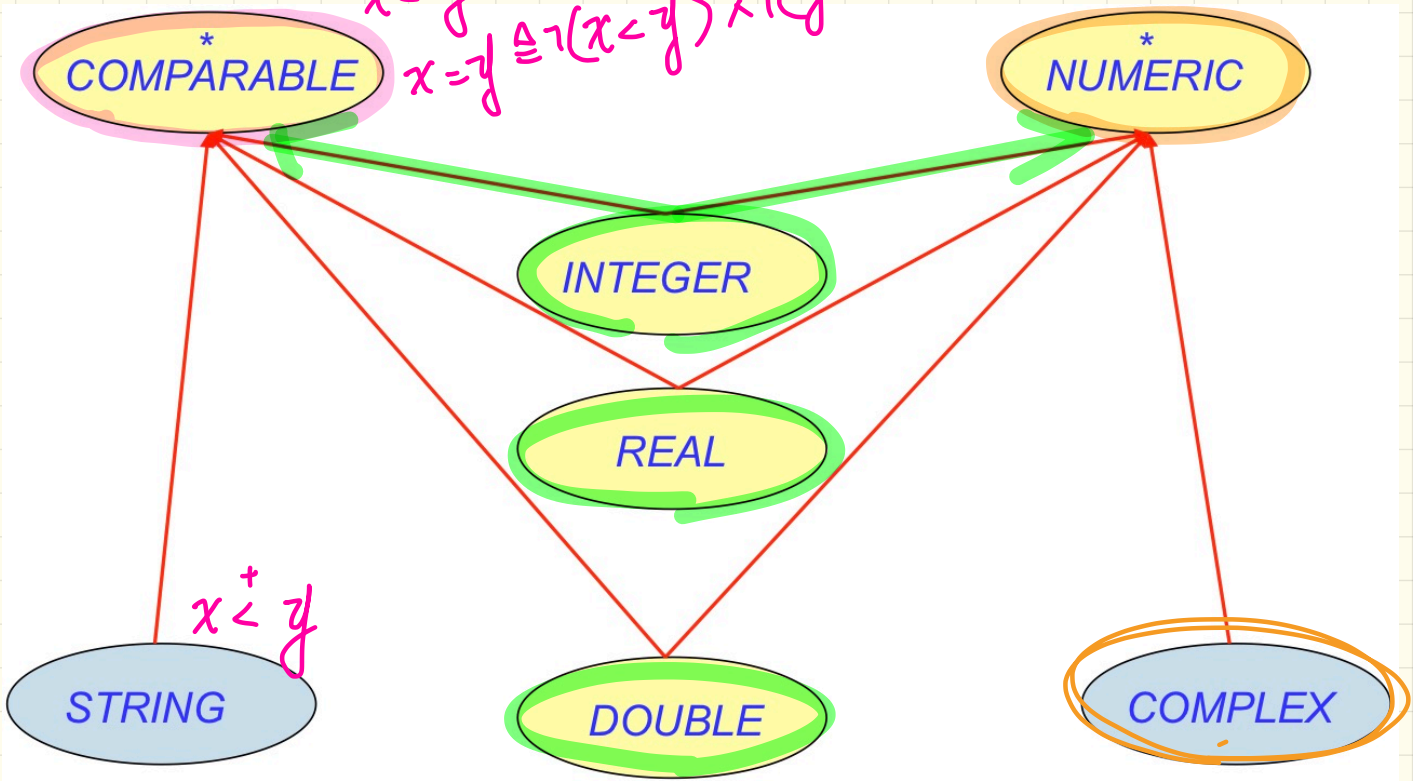
M.I. only supported in Java by a very limited way



class A extends B ✓.

implements C, D

Multiple Inheritance: Example



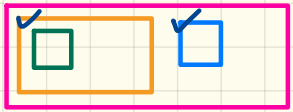
Multiple Inheritance: Exercise

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

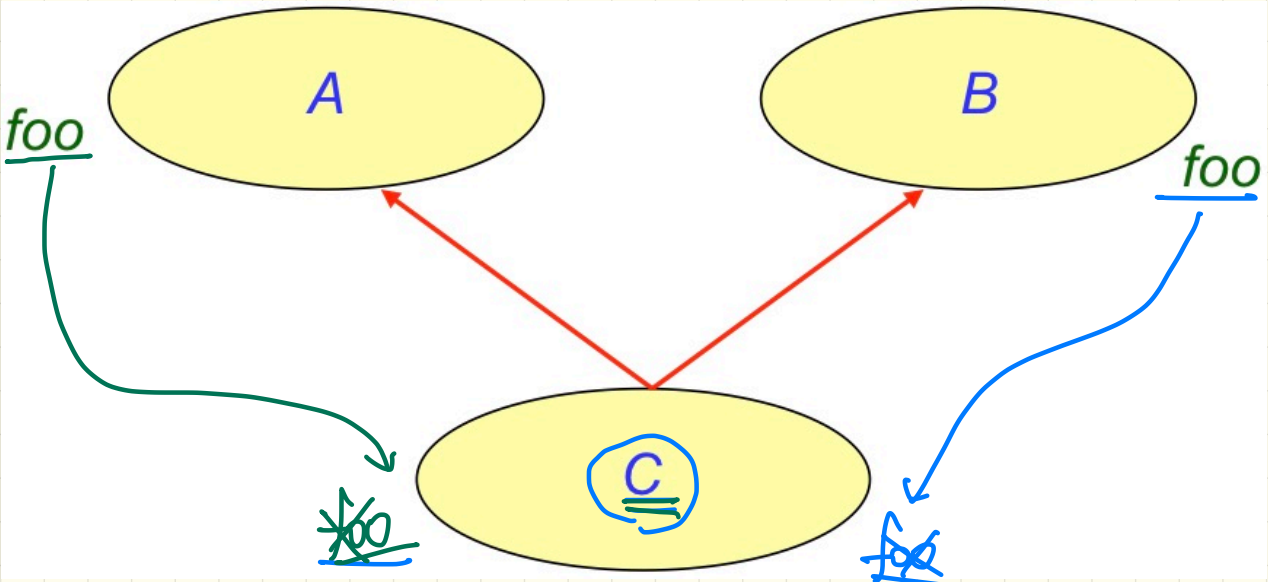
```
class TREE[G]
  feature -- Queries
    descendants: ITERABLE[G]
  feature -- Commands
    add (c: G)
        -- Add a child 'c'.
end
```

```
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
```

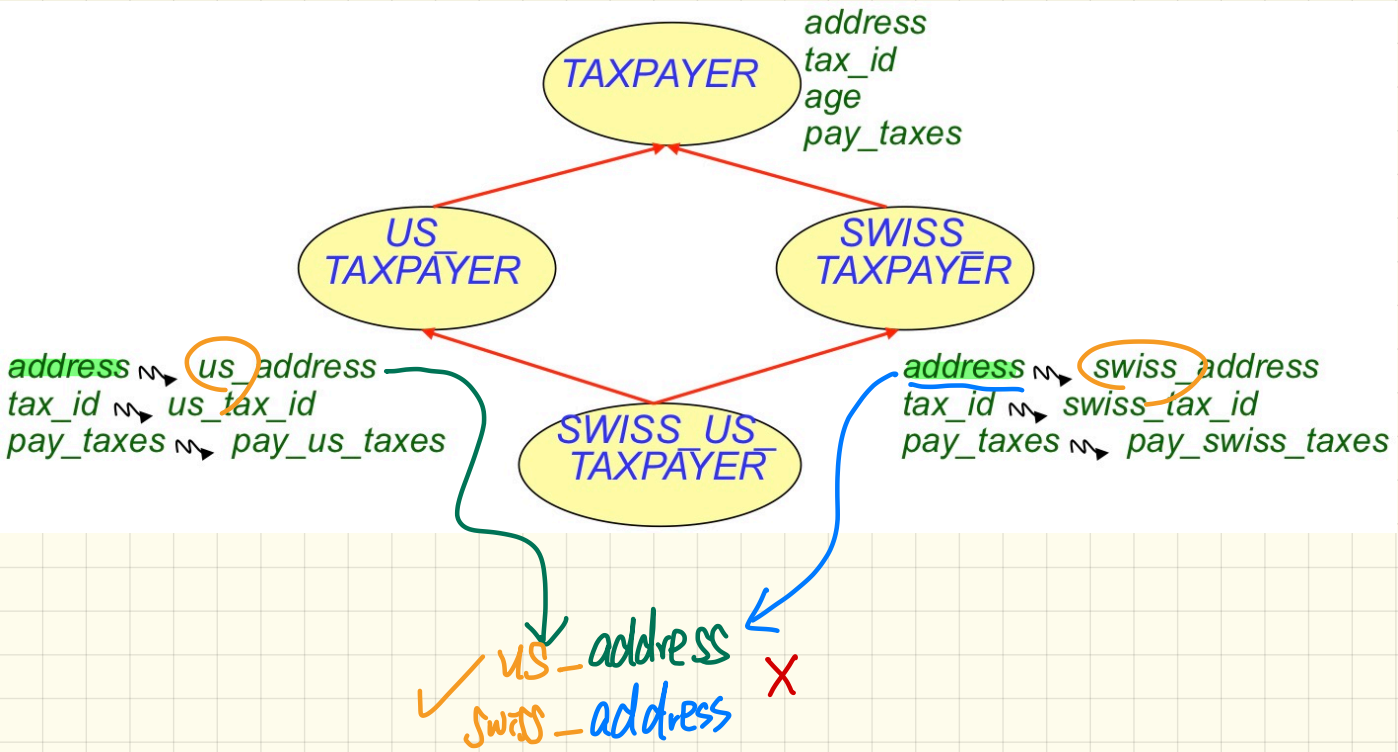


Multiple Inheritance: Name Clashes



x : C → exp: foo, Zoo.

Multiple Inheritance: Name Clashes

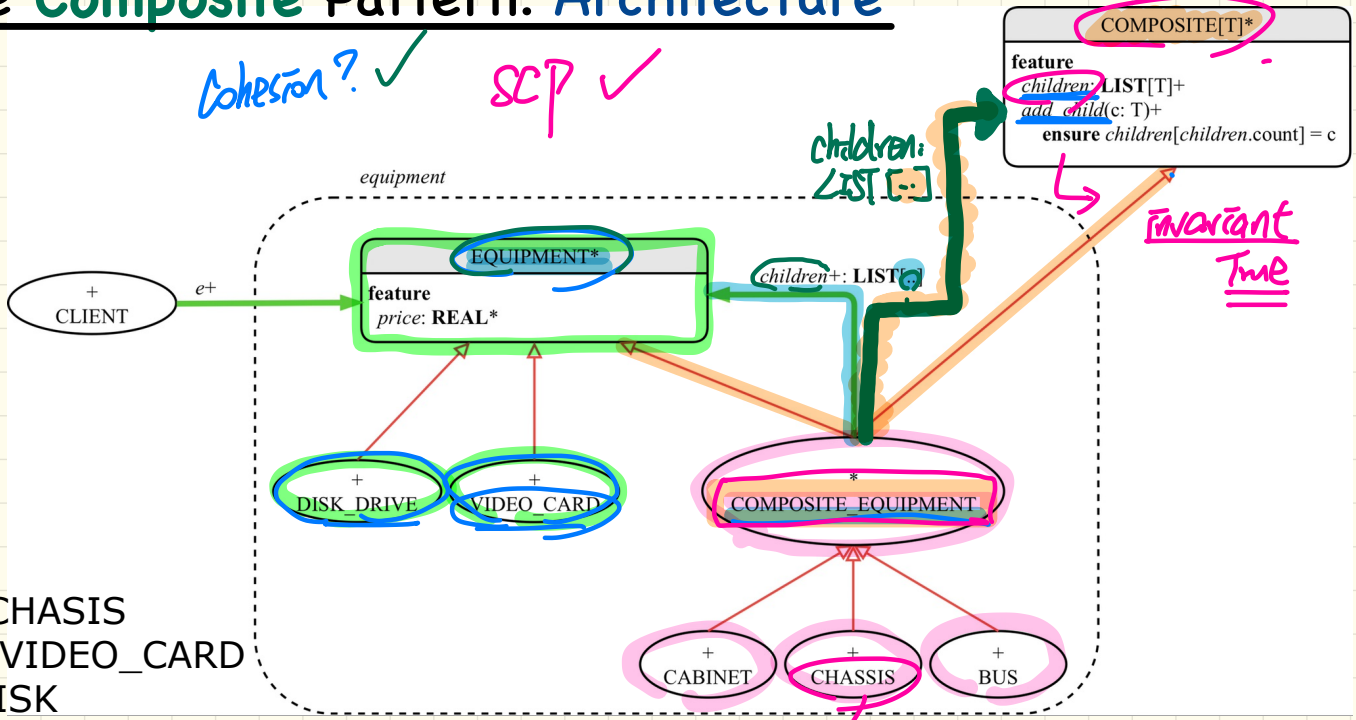


Lecture 10

Part 6

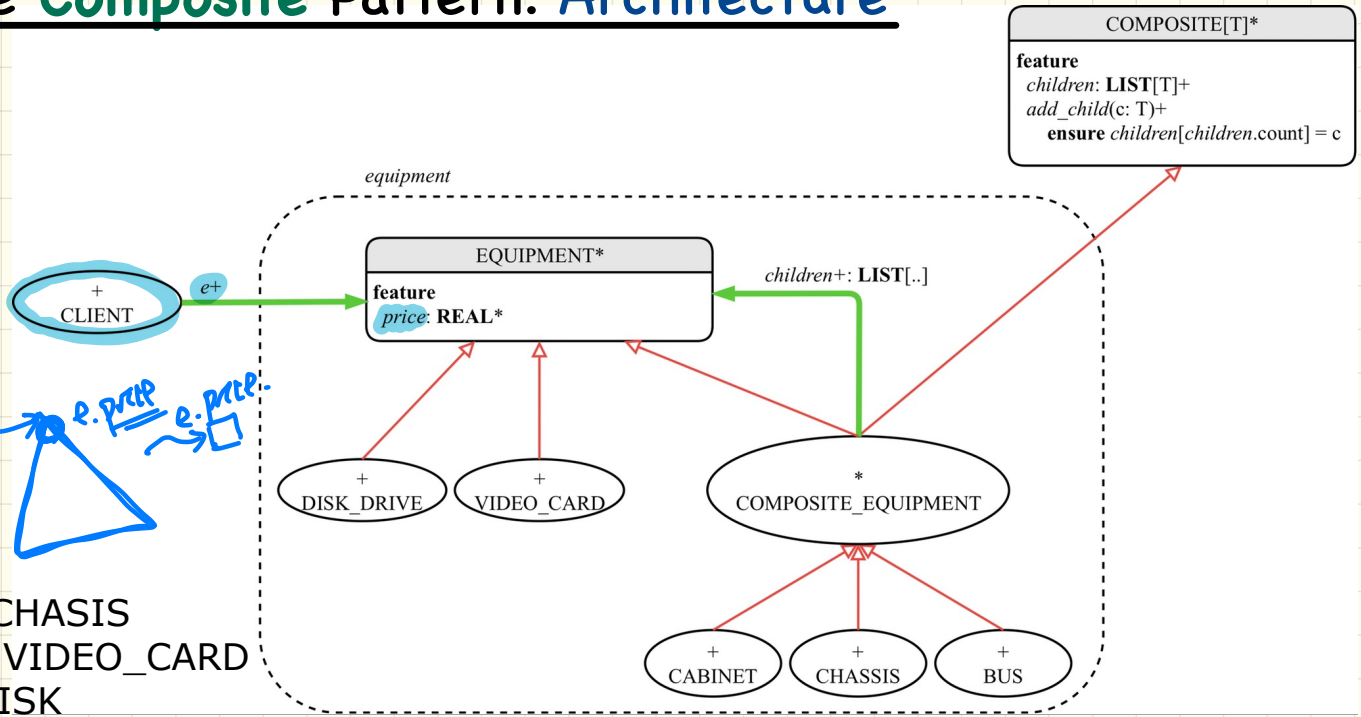
Composite Design Pattern

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) ✗

The Composite Pattern: Architecture



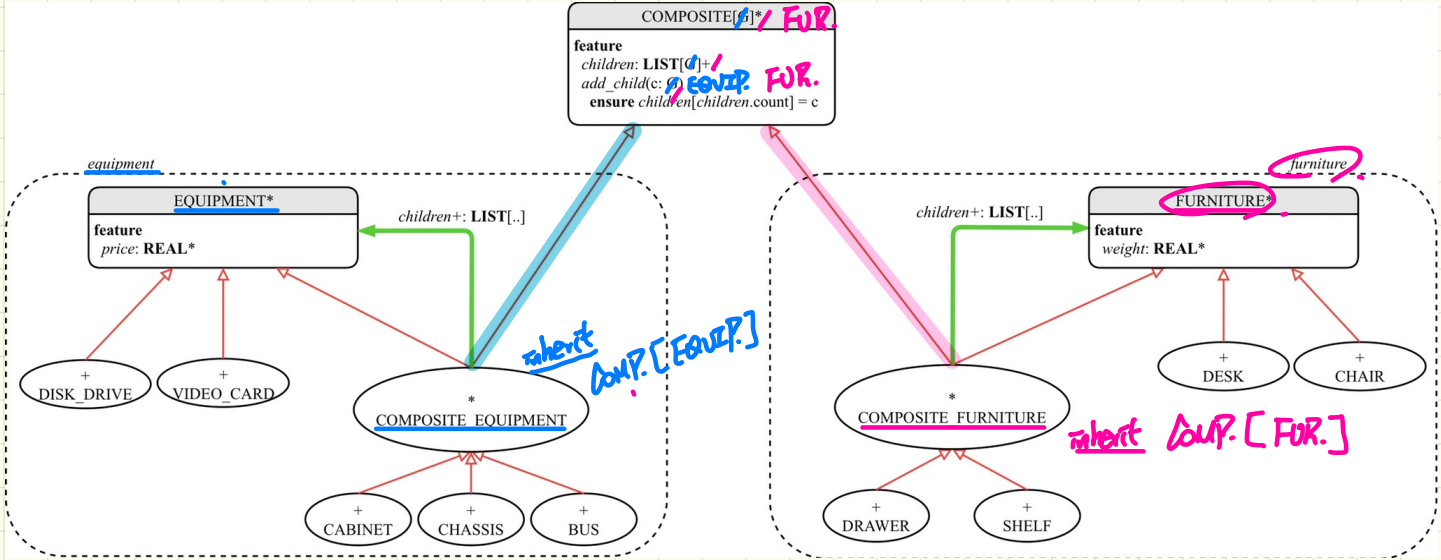
ch: CHASIS
crd: VIDEO_CARD
d: DISK
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)
crd.add_child(d)

Why is COMPOSITE a separate class?

↳ satisfies SCP.

The Composite Pattern: Architecture

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



The Composite Pattern: Implementation

```
deferred class
  EQUIPMENT
feature
  name: STRING
  price: REAL deferred end
end
```

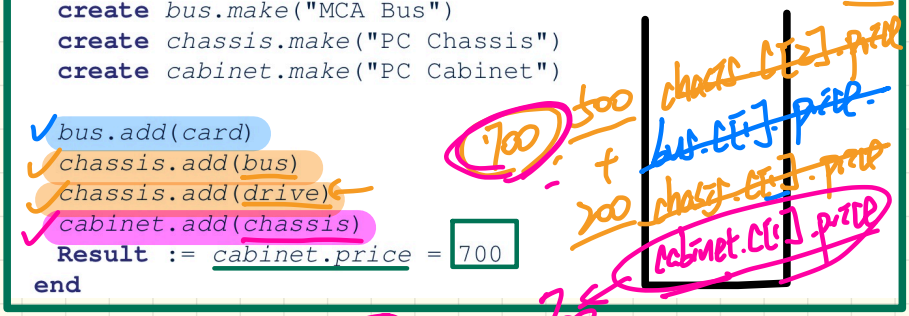
```
deferred class
  COMPOSITE[T]
feature
  children: LINKED_LIST[T]

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

```
class
  CARD
inherit
  EQUIPMENT
feature {NONE}
  unit_price: REAL
feature
  make (n: STRING; p: REAL)
    do name := n ; unit_price := p end
  price
    do Result := unit_price end
end
```

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

Testing the Composite Pattern

```
test_composite_equipment: BOOLEAN
  local
    card, drive: EQUIPMENT
    cabinet: CABINET -- holds a CHASSIS
    chassis: CHASSIS -- contains a BUS and a DISK_DRIVE
    bus: BUS -- holds a CARD
  do
    create {CARD} card.make("16Mbs Token Ring", 200)
    create {DISK_DRIVE} drive.make("500 GB harddrive", 500)
    create bus.make("MCA Bus")
    create chassis.make("PC Chassis")
    create cabinet.make("PC Cabinet")

    bus.add(card)
    chassis.add(bus)
    chassis.add(drive)
    cabinet.add(chassis)
    Result := cabinet.price = 700
  end
```




```
class
  CARD
inherit
  EQUIPMENT
feature {NONE}
  unit_price: REAL
feature
  make (n: STRING; p: REAL)
    do name := n ; unit_price := p end
  price
    do Result := unit_price end
end
```

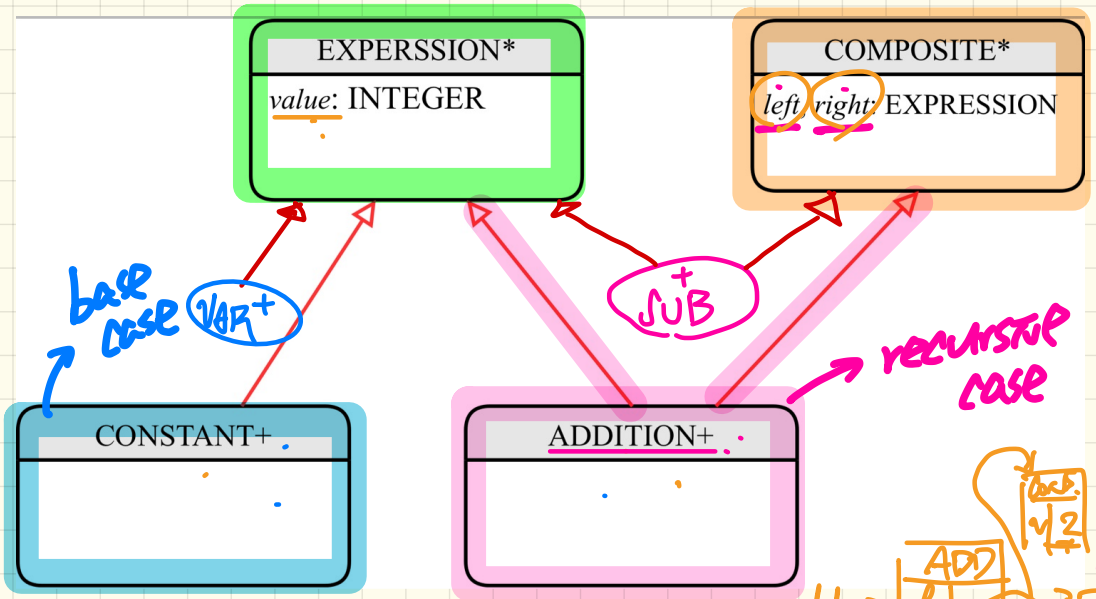
```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  price : REAL -- price is a quer
      -- Sum the net prices of all
    do
      across
        children is c
      loop
        Result := Result + c.price
      end
    end
end
```

Lecture 11

Part 1

Processing Recursive Systems

Design of Language Structure: Composite Pattern



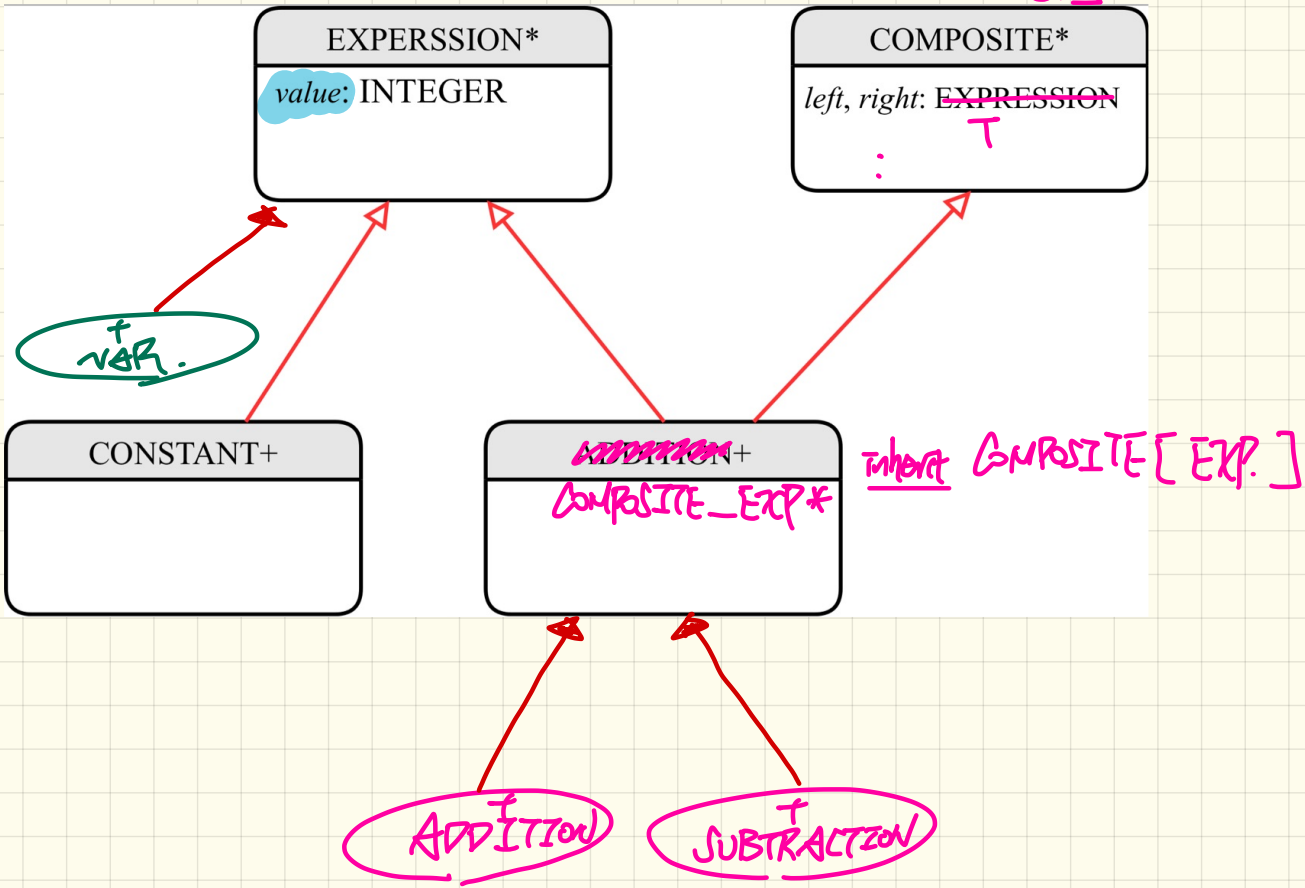
COMPOSITE*
children: LIST[EXP]
invariant
children = 2

c1, c2: CONSTANT
add: ADDITION
create c1.make(341)
create c2.make(2)
create add.make (c1, c2)
add.value → 343

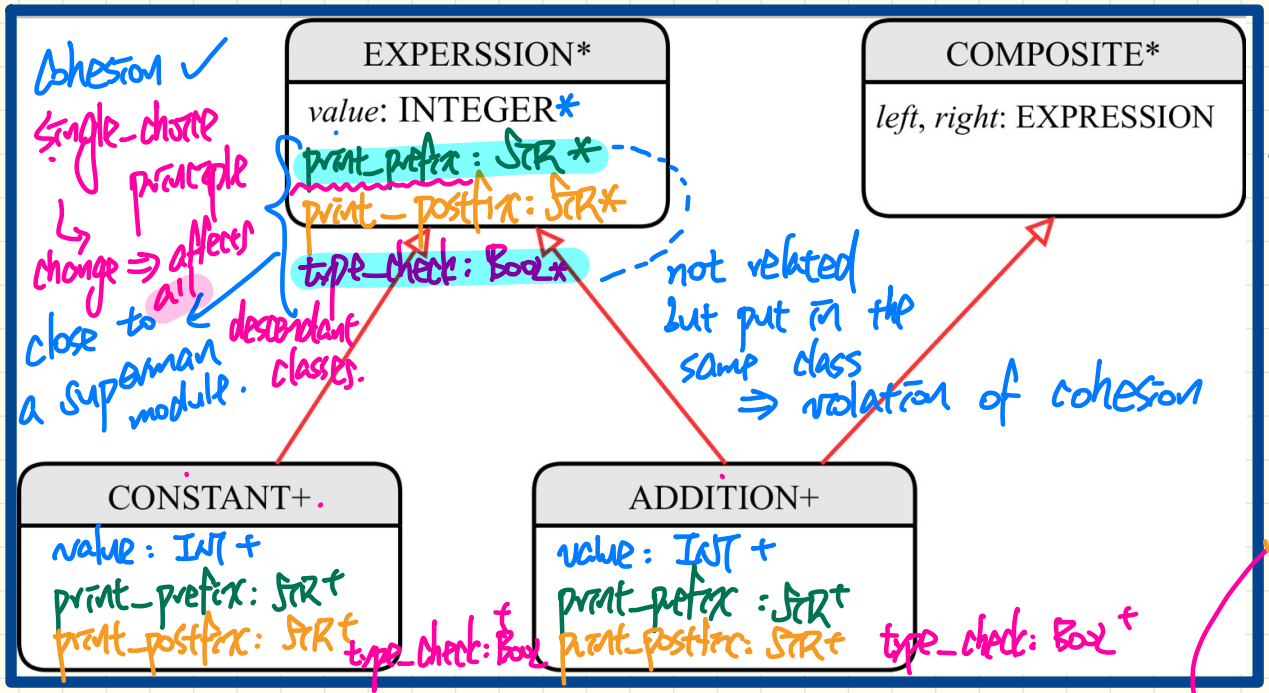
add → ADD (l. r.)

Q: How to construct a composite object representing "341 ⊕ 2"?

Q: How to extend the design to include variables and subtractions?


[T]
EXPERSSION*
value: INTEGER
COMPOSITE*
left, right: EXPRESSION
T
:
VAR.
CONSTANT+
COMPOSITE_EXP*
inherit COMPOSITE[EXP.]
ADDITION
SUBTRACTION

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



Lecture 11

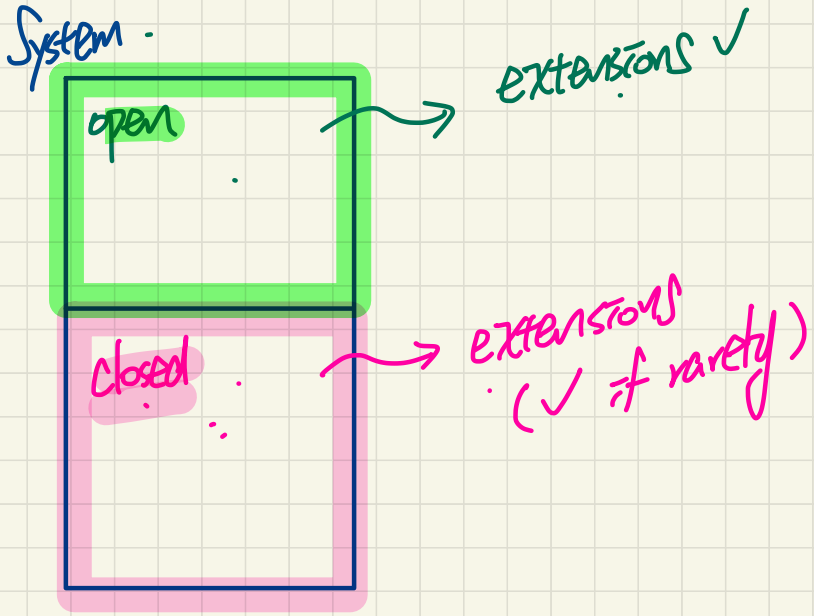
Part 2

Open-Closed Principle

Open-Closed Principle

How can the OCP be satisfied?

① There should be a clear separation/decomposition of the system into open vs. closed parts.



If there's a change:

② Mostly the change should touch the open part.

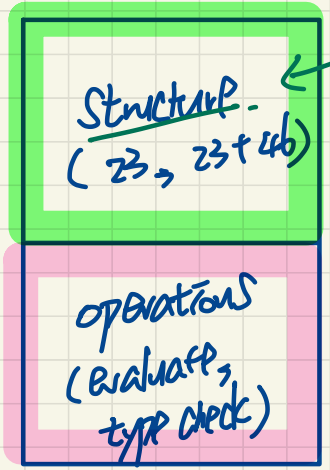
③ Rarely, if at all, a change may have to touch the closed part.

Applying the OCP to Exp. language design.

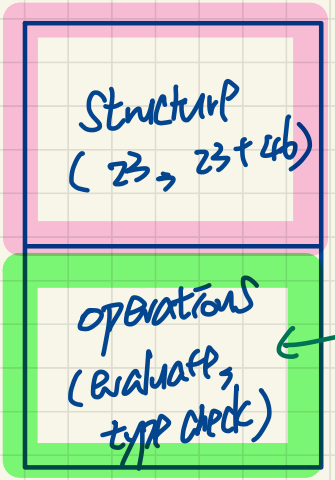
Both alternatives satisfy OCP.

Alt 1 ✓

Alt 2 → design context/assumption for Visitor Pattern.

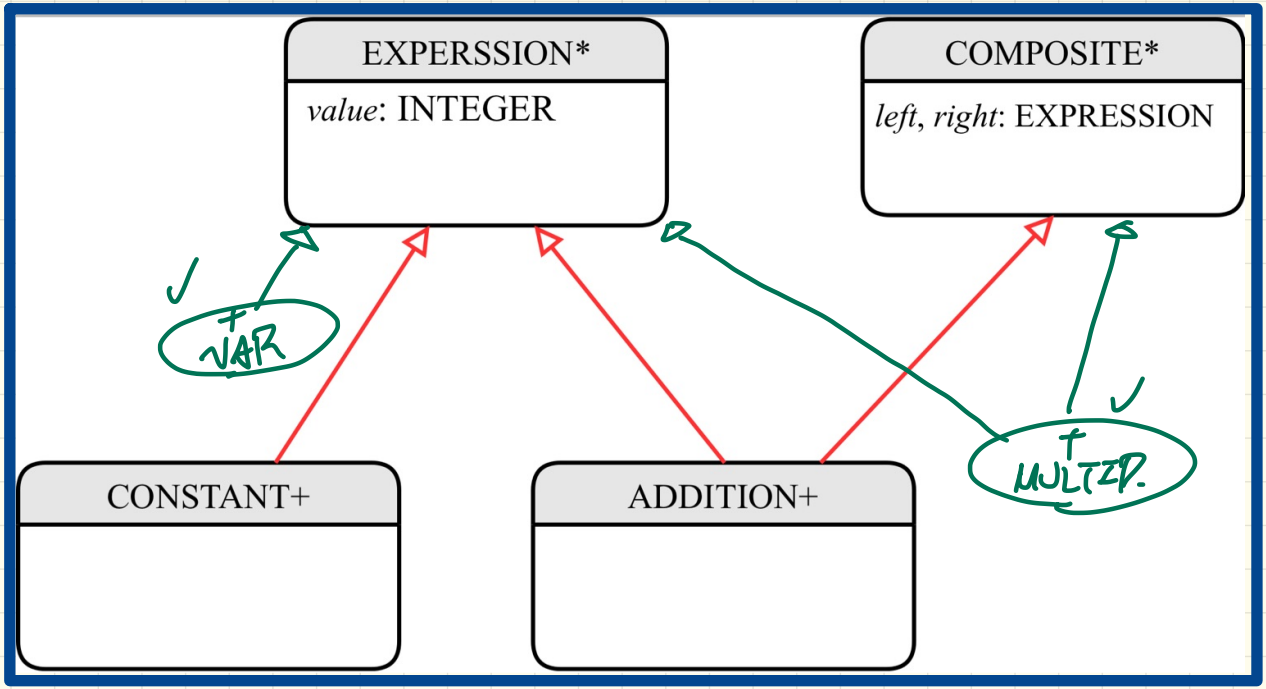


changes should happen here



changes should happen here.

Design of a Language Application: Open-Closed Principle



Structure

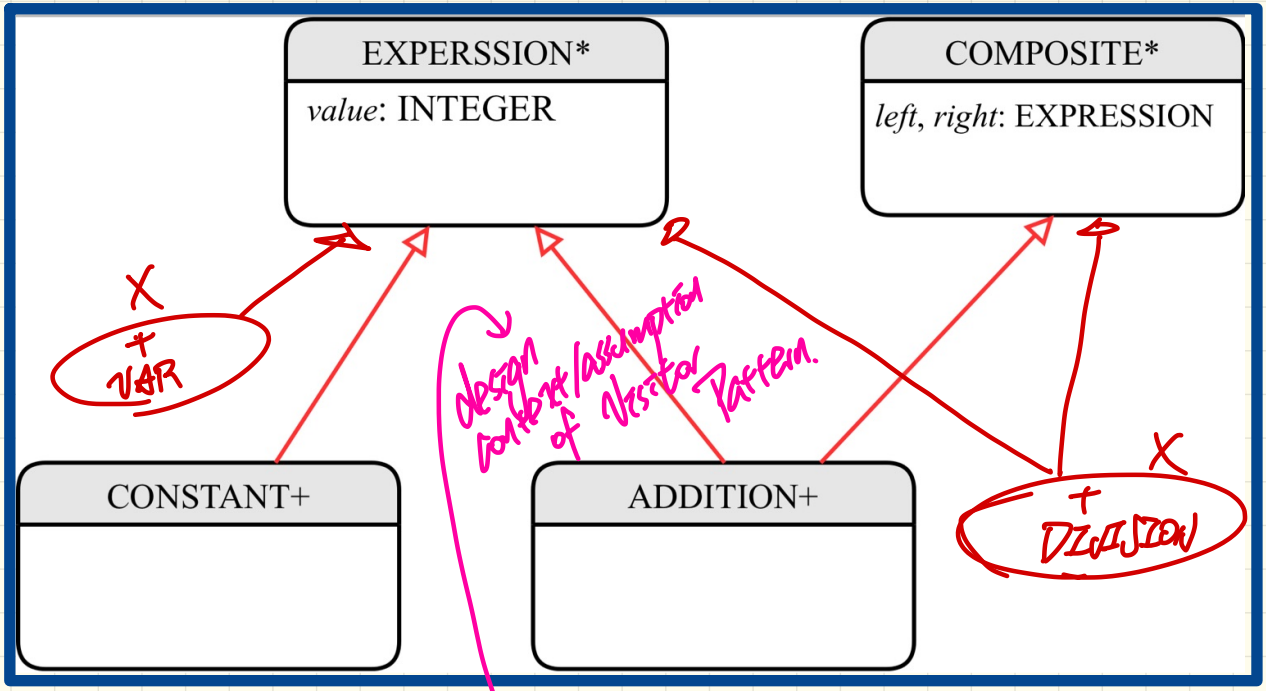
- evaluate
- print_prefix
- print_postfix
- type_check

Operations

optimize
code_gen.

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Design of a Language Application: Open-Closed Principle



evaluate
print_prefix
print_postfix
type_check

Operations

code_gen ✓
optimize ✓

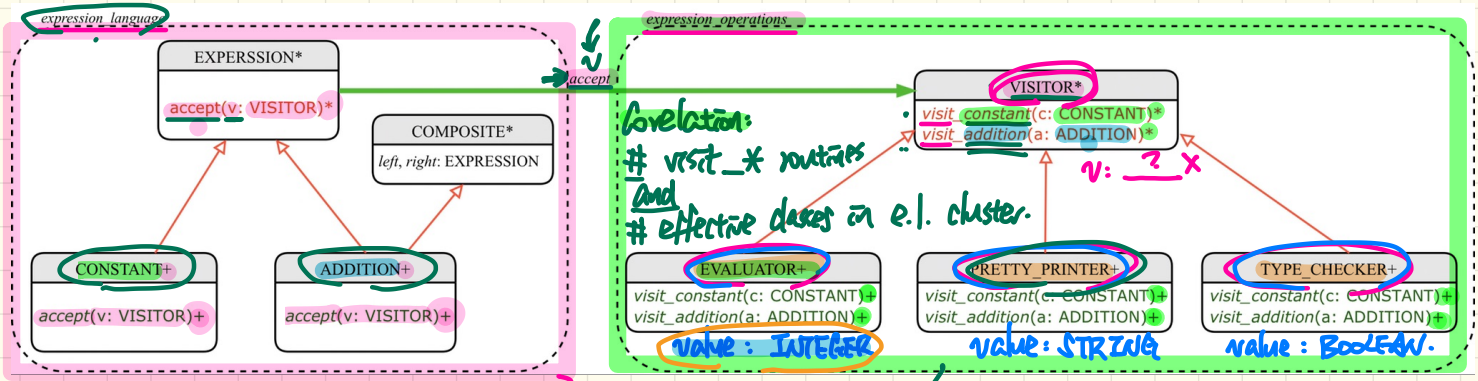
	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Lecture 11

Part 3

Visitor Design Pattern

Visitor Design Pattern: Architecture



How to Use Visitors

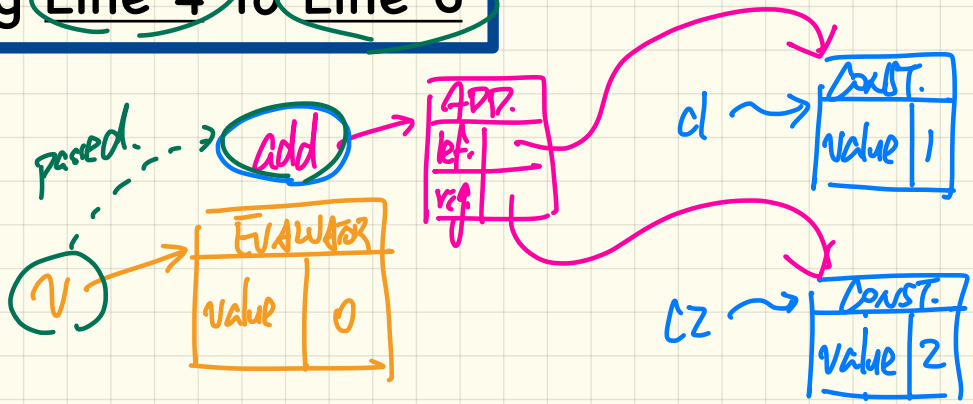
```
1  test_expression_evaluation: BOOLEAN
2    local add, c1, c2: EXPRESSION ; v: VISITOR
3    do
4     create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
5     create {ADDITION} add.make (c1, c2)
6     create {EVALUATOR} v.make
7     add.accept (v)
8     check attached {EVALUATOR} v as eval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

Visitor Design Pattern: Implementation

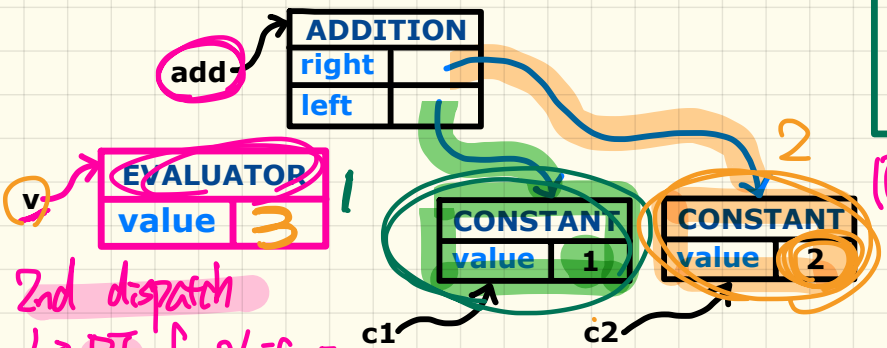
```
1  test_expression_evaluation: BOOLEAN
2    local add, c1, c2: EXPRESSION ; v: VISITOR
3    do
4      create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
5      create {ADDITION} add.make (c1, c2)
6      create {EVALUATOR} v.make
7      add.accept (v)
8      check attached {EVALUATOR} v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

→ building the composite/recursive object.

Visualizing Line 4 to Line 6



Executing Composite and Visitor Patterns at Runtime



2nd dispatch
↳ DT of v is EVALUATOR
↳ visit_addition in EVALUATOR called

Tracing add.accept(v)
Double Dispatch

add.accept(v)
↳ 1st (dynamic) dispatch
↳ DT of add is ADDITION
↳ accept in ADDITION 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
```

CONSTANT

```
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

double dispatch

double dispatch

evaluator

1 + 2 = 3

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

SI.

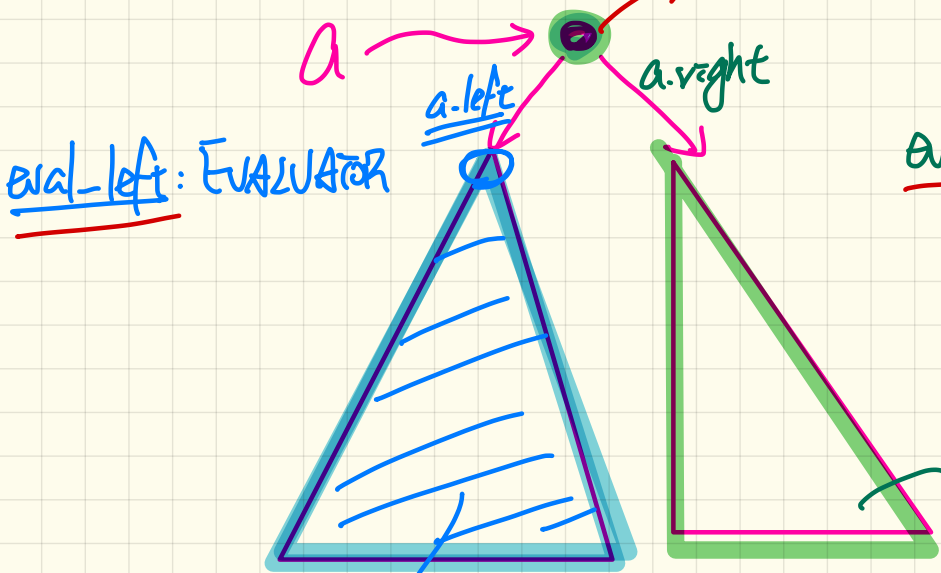
v

add

ADDITION

1+2

a : ADDITION



eval_left.value + eval_right.value

eval_right: EVALUATOR

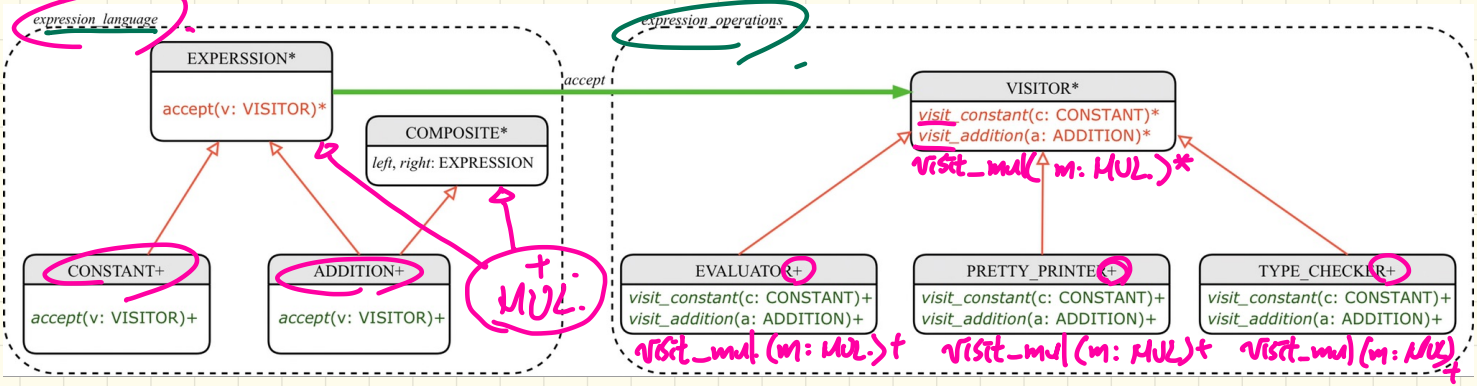
a.left.accept(eval_left)

eval_left.value

a.right.accept(eval_right)

eval_right.value

Visitor Pattern: Open-Closed and Single-Choice Principles



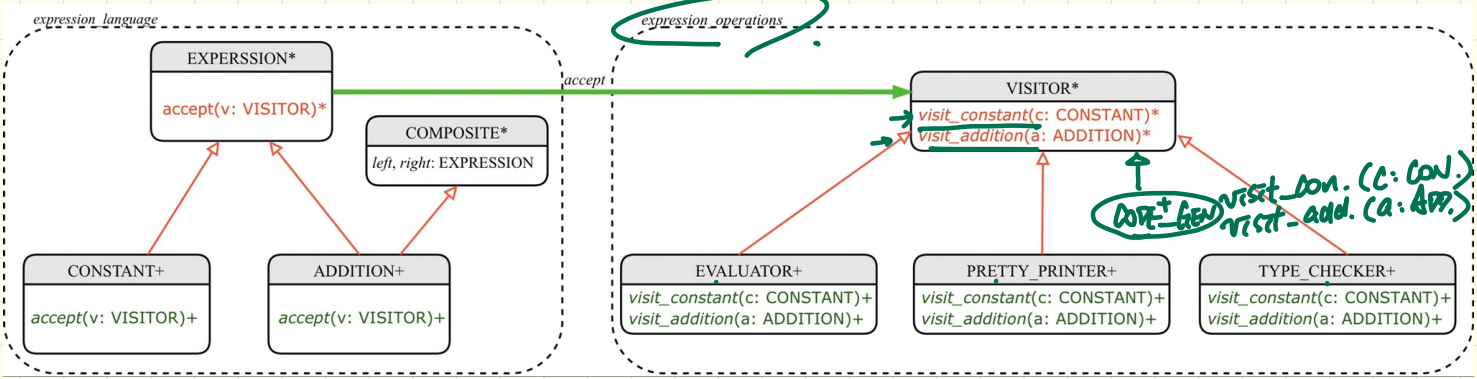
What if a new language construct is added?

① single class to be added to the structure operations] → violates SCP
② multiple places to modify in

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

structure

Visitor Pattern: Open-Closed and Single-Choice Principles



satisfies SCP.

What if a new language operation is added?

① single class added to operations
② all changes are restricted to this single class

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

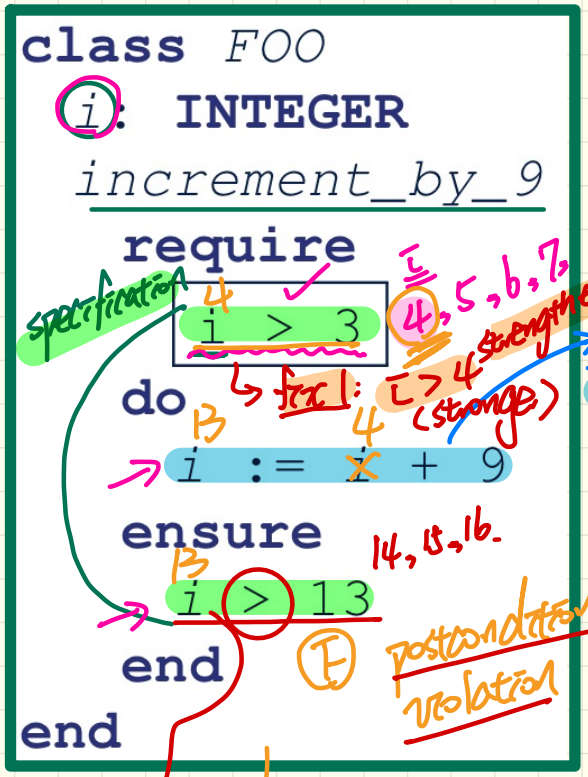
operations.

Lecture 12

Part 1

Correctness - Motivating Examples

Program Correctness: Example (1)



specification

precondition

implementation

postcondition

Correctness of Program: (relative)

implementation satisfies spec.

Given valid input (satisfying precond) executing the implementation will

(1) terminate

(2) upon termination, the postcondition is satisfied.

$i > 3$ — i: 4, 5, 6, 7, ...

fix 1: $i > 4$ (stronger) — strengthen precondition

$i := i + 9$

$i > 13$ — 14, 15, 16.

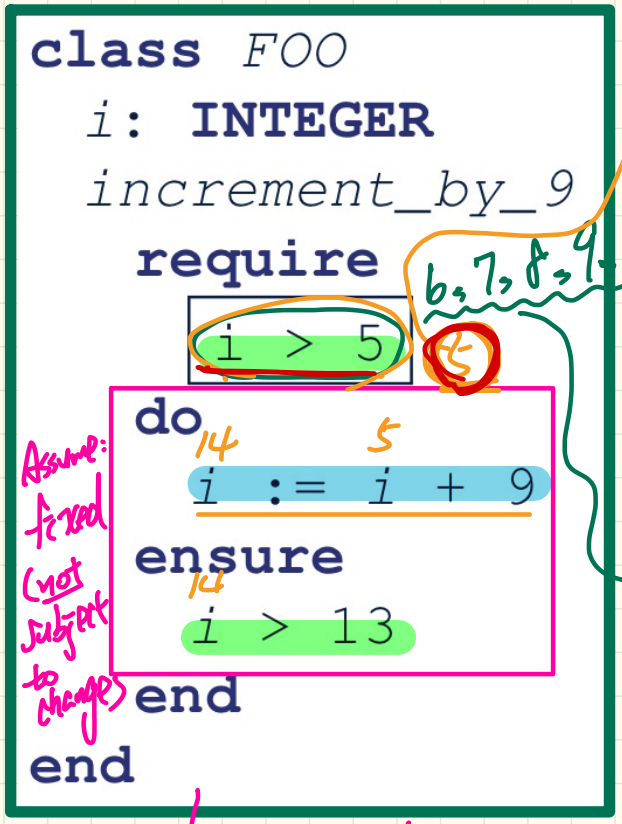
postcondition violation

precondition $i > 3$ is too weak ($i = 4$ is allowed)

this program is not correct.

Fix 2: weaken the postcondition: $i > 12$

Program Correctness: Example (2)



Guiding Principle

Is $i > 5$ too strong? $i \geq 5$ will not cause postcond. violat.

Precondition cannot be too weak (i.e. it does not allow any input value that can cause a postcondition violation).

6, 7, 8, 9 ...

whether an input should be included is up to the req.

all values satisfying this precondition will guarantee the satisfaction of postcondition (after executing the implementation).

Correct. (assuming that 5 should be excluded).

Lecture 12

Part 2

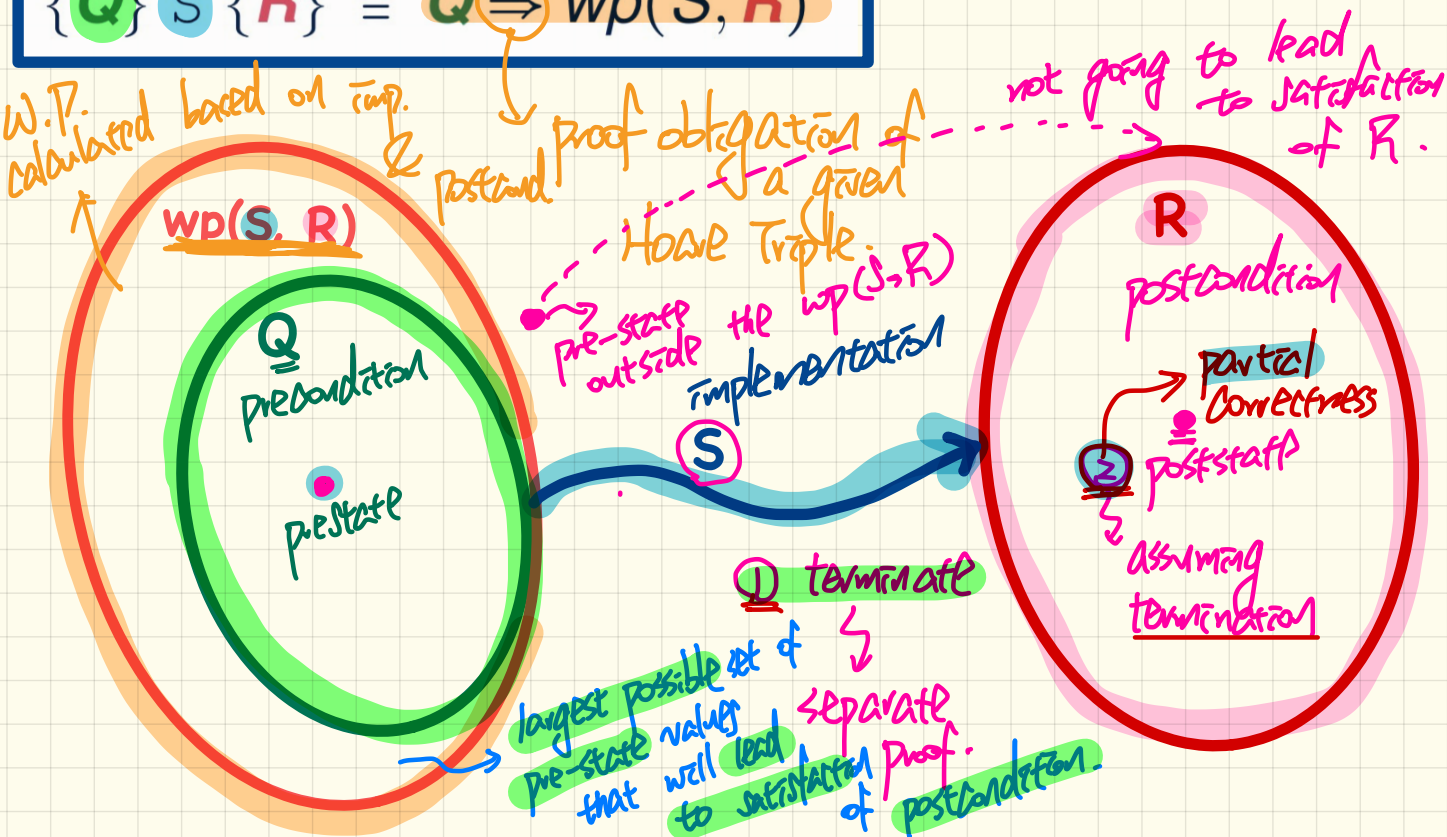
Hoare Triple and Weakest Precondition

Hoare Triple as a Predicate

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

② : partial correctness

① + ② : total correctness

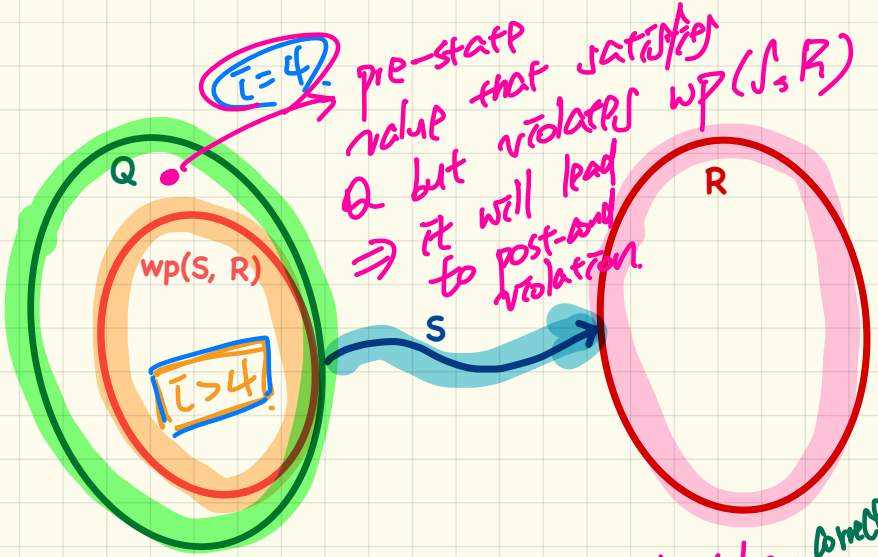


Program Correctness: Revisiting Example (1)

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

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

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



$\{i>3\}\ i := i+9\ \{i>13\}$ → Hoare Triple → predicate → tautology (correct.) / counter-example (incorrect.)

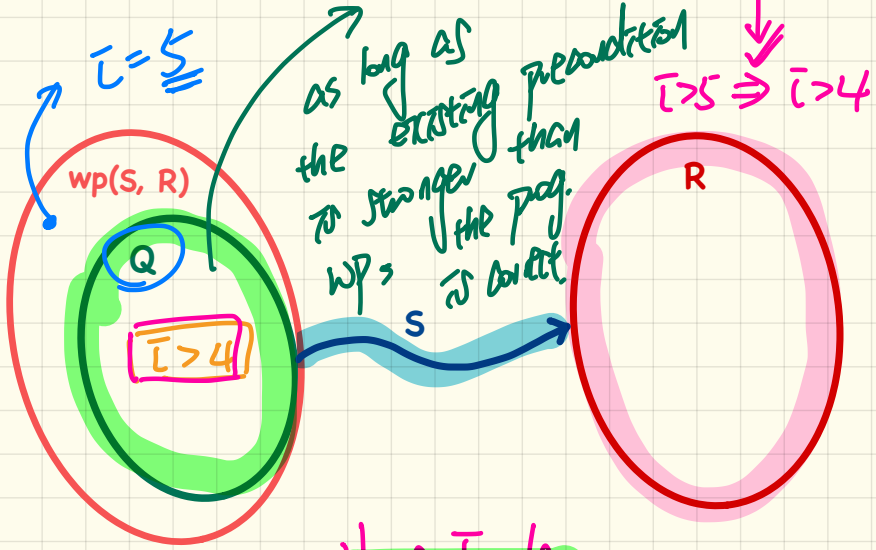
Program Correctness: Revisiting Example (2)

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

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

wp.

all allowed input values will satisfy



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

Hoare Triple
↳ can be proved as a tautology
↳ correct.

Lecture 12

Part 3

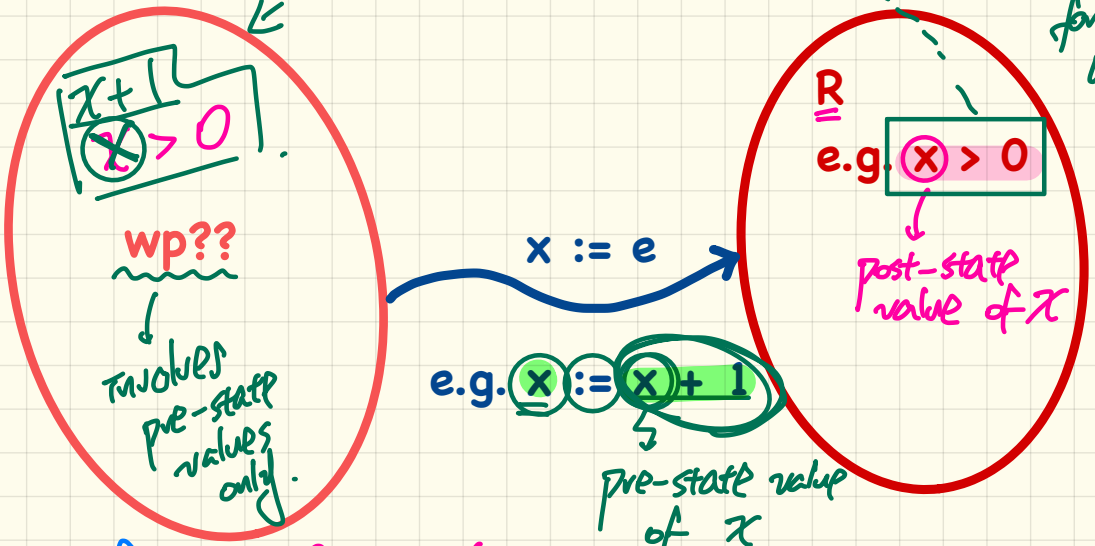
Rules of wp Calculus

Rules of Weakest Precondition: Assignment

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

replace every free occurrence of x by e.

massage this post-state expression into an equivalent one for the w.p. ↳ precondition



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

Correctness of Programs: Assignment (1)

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

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

post-state pre-state

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

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

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

pre-state value: $x_0 + 1$

$= \{$ replace each free occurrence of x by $x_0 + 1 \}$

$x_0 + 1 > x_0 = \{$ arithmetic $\} =$ True

In order for the design of ?? to be correct:

?? $\Rightarrow$ True

any precondition is ok.

w.p. for $x := x+1$ to establish $x > x_0$.

w.p. True

Correctness of Programs: Assignment (2)

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

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

Rules of Weakest Precondition: Conditionals

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

wp(S: PROGRAM; r: PRED)
- :=
- if-
- ;
- loop

Arbitrarily complicated

$B \overset{\wedge ?}{\Rightarrow} wp(S_1, R)$ ✓

$\wedge . \vee ?$

$\neg B \overset{\wedge ?}{\Rightarrow} wp(S_2, R)$ ✓

Rules of Weakest Precondition: Conditionals

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

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

vs.

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

??

$False[x := x+1] \equiv False$

Consider:

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

$y>0 \Rightarrow wp(x := x+1, x>=0)$ — T ⇒ F ≡ F

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

∧ : F

F → correct result by using ∧.

T → by using ∨, misleading result

$y=1, x=-4$

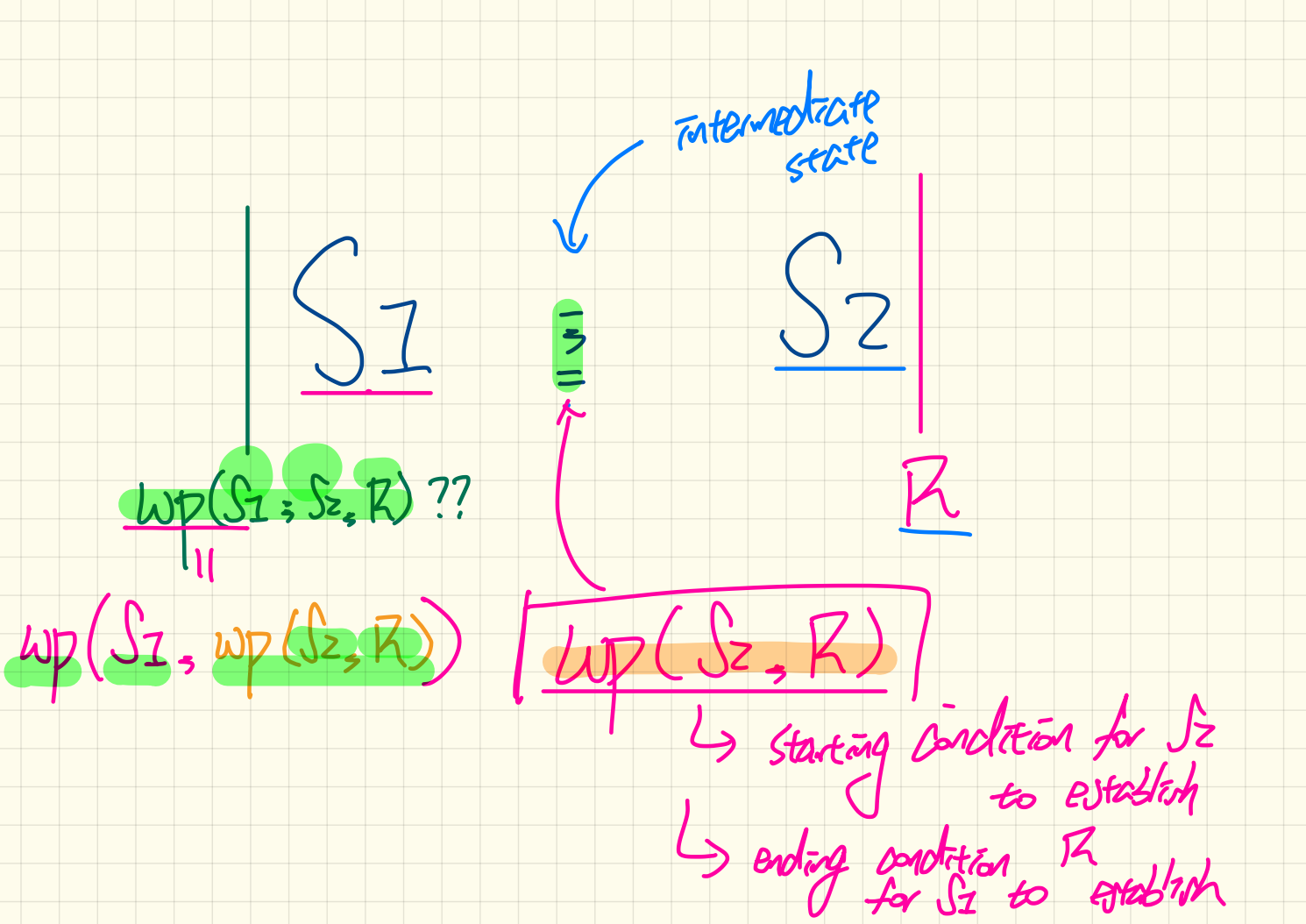
Correctness of Programs: Conditionals

Is this program correct?

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

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

$\{bigger \geq smaller\}$


intermediate state
S1
S2
R
wp(S1; S2, R) ??
=
wp(S1, wp(S2, R))
wp(S2, R)
↳ starting condition for S2 to establish
↳ ending condition for S1 to establish R

Correctness of Programs: Sequential Composition

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

Step 1: calculate wp

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

$= \{$ wp rule for seq. comp $\}$

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

$= \{$ wp rule for seq. comp. $\}$

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

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

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

Step 2

Prove or disprove:

$True \Rightarrow y > x$

$\equiv y > x$

not tautology. Counter-example.

x	4
y	3

prog not correct.

Rules of Weakest Precondition: Summary

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

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

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

Proof Rules using Weakest Precondition

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

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

$$\{Q\}\ \texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ end}\ \{R\}$$
$$\iff \left(\begin{array}{l} \{\ Q \wedge B\ \}\ S_1\ \{\ R\ \} \\ \wedge \\ \{\ Q \wedge \neg B\ \}\ S_2\ \{\ R\ \} \end{array} \right) \iff \left(\begin{array}{l} (Q \wedge B) \Rightarrow wp(S_1, R) \\ \wedge \\ (Q \wedge \neg B) \Rightarrow wp(S_2, R) \end{array} \right)$$

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

Lecture 12

Part 4

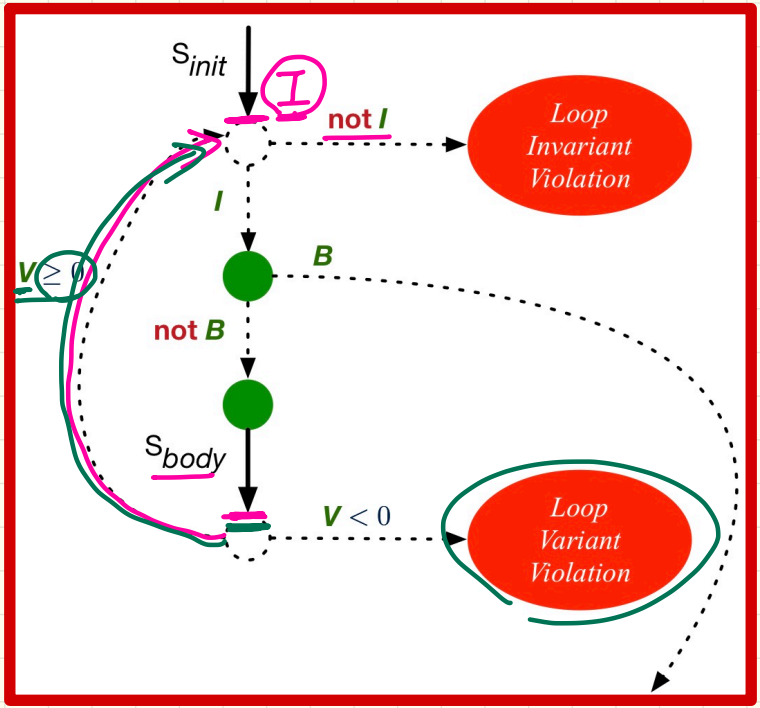
Contracts of Loops

Contracts of Loops

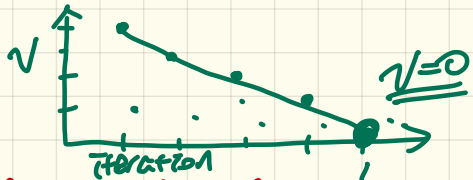
Syntax

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

Runtime Checks



Contracts of Loops: Example



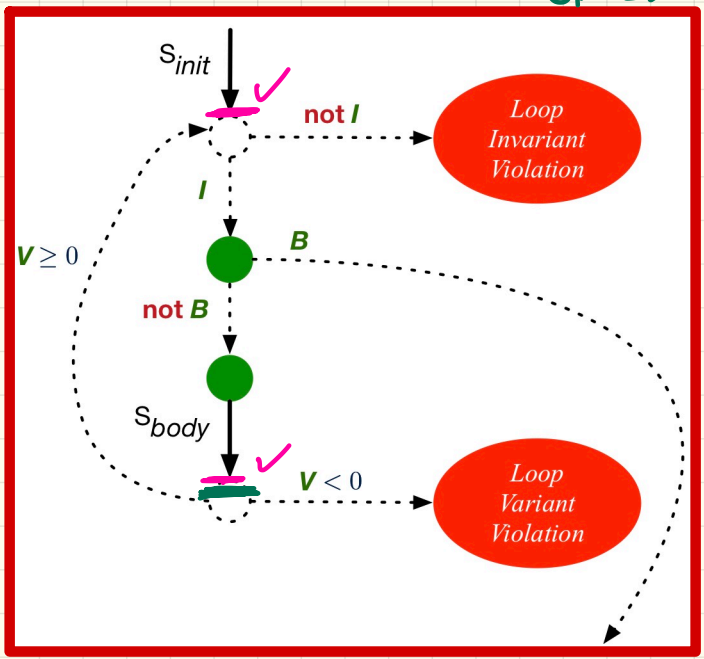
Syntax

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

6-2 4 ≥ 0
6-3 3 ≥ 0
6-6 0 ≥ 0
iteration 1
2
3
4
5

6 → not output.

Runtime Checks

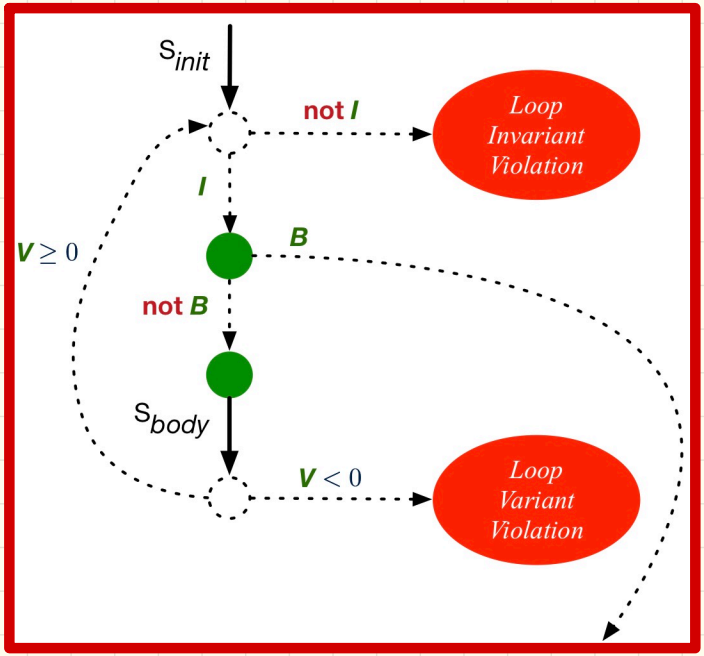


Contracts of Loops: Violations

Syntax

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

Runtime Checks



invariant: 1 <= i <= 5

↳ At the end of 5th iteration:

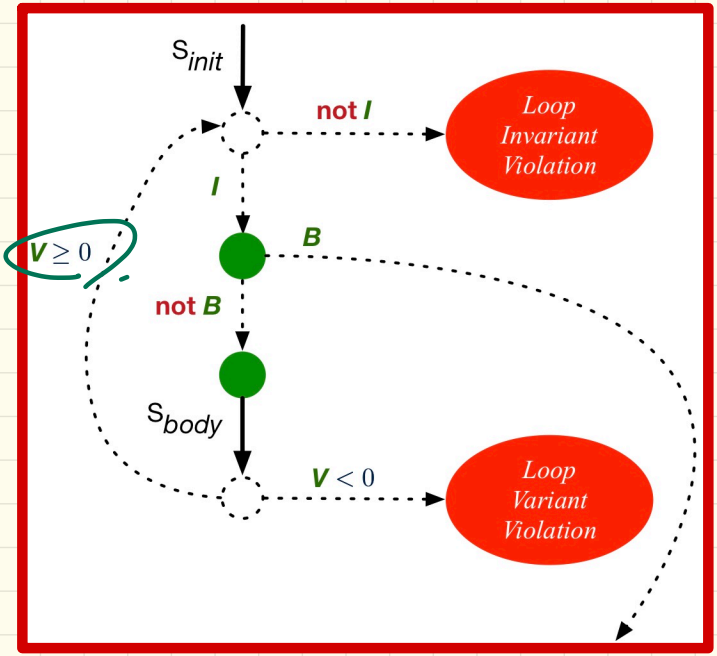
$1 \leq \overset{6}{i} \leq 5 \Rightarrow$ LI violation.

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
  5 X - i
    end
end
```

Runtime Checks

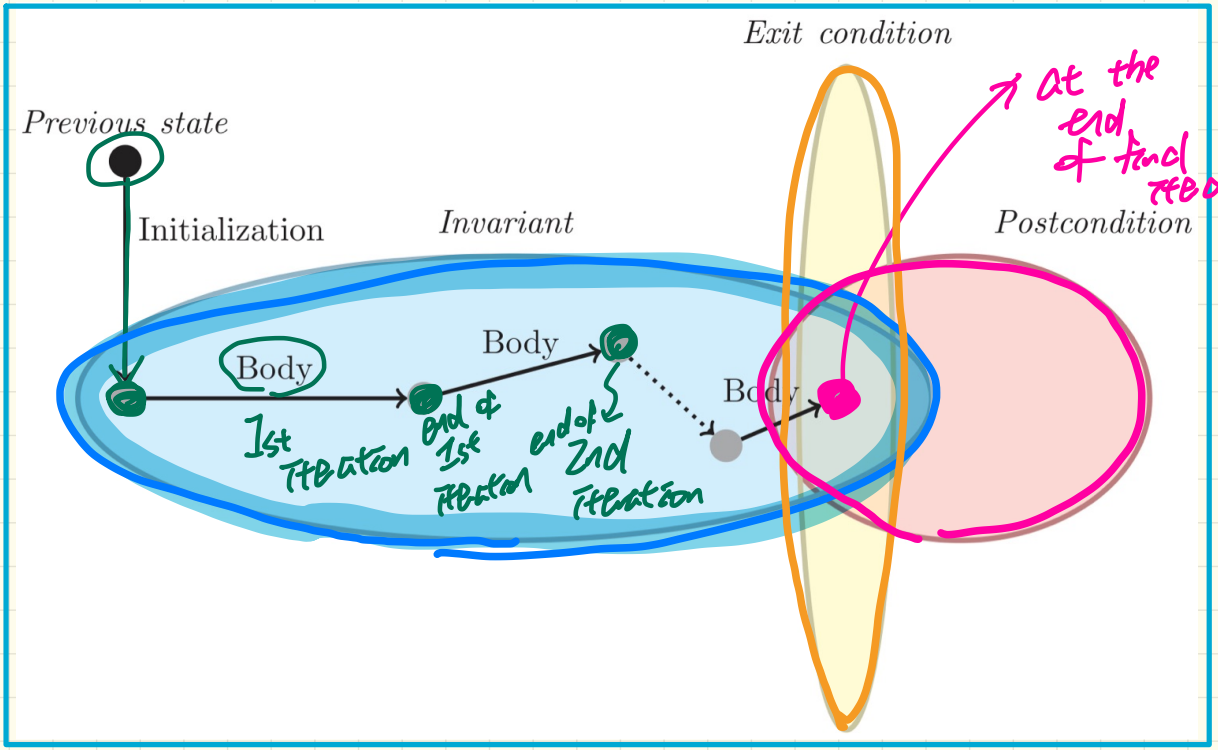


variant: 5 - i

↳ At the end of 5th iteration

LV: 5 - 6 = -1 ≥ 0 ≡ F ⇒ LV violation

Contracts of Loops: Visualization



Contracts of Loops: Loop Invariant

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: -- ∀j | a.lower ≤ j < i • Result ≥ a[j]
        across a.lower |..| (i - 1) as j all Result >= a [j.item] end
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper - i
    end
  ensure
    correct_result: -- ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
      across a.lower |..| a.upper as j all Result >= a [j.item]
  end
end
```

① After init before 1st iteration.

$\forall j : \boxed{1 \le j \le 0} \bullet \_\_\_$

F.

② At the end of 1st iteration: $i = 2$

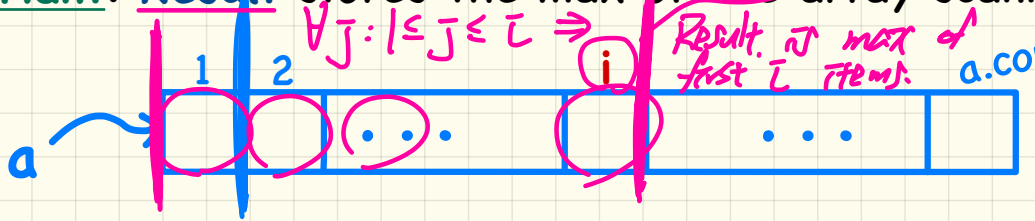
$\forall j : 1 \le j \le 1 \bullet \_\_\_$

T.

③ At the end of ith iteration: loop counter $i+1$

At the end of ith iteration, what's the value of i? $i+1$

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



Finding Max: Version 1

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: -- ∀j | a.lower ≤ j ≤ i • Result ≥ a[j]
        across a.lower |..| i as j all Result >= a [j.item] end
    until
      i > a.upper
    loop
    if a [i] > Result then Result := a [i] end
    i := i + 1
    variant
      loop_variant: a.upper - i + 1
    end
  ensure
    correct_result: -- ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
      across a.lower |..| a.upper as j all Result >= a [j.item]
  end
end
```

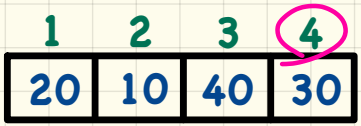
1	2	3	4
20	10	40	30

Iteration	Result	i	LI
Init	20	1	$1 \le j \le 1$
1	20	2	$1 \le j \le 2$
2	20	3	$1 \le j \le 3$

LI violation: Result 20 is not max of the first 3 items.

After Iteration	i	Result	LI	Exit ($i > a.upper$)?	LV
Initialization	●	●	●	●	●
1st	●	●	●	●	●
2nd	●	●	●	●	●

Finding Max: Version 2




```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: -- ∀j | a.lower ≤ j < i • Result ≥ a[j]
        across a.lower |..| (i - 1) as j all Result >= a [j.item] end
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
    i := i + 1
    variant
      loop_variant: a.upper - i
    end
  ensure
    correct_result: -- ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
      across a.lower |..| a.upper as j all Result >= a [j.item]
  end
end
```

4 - 5 = -1 ≥ 0 ≡ F

LV violation

After Iteration	i	Result	*LI*	Exit (*i* > *a.upper*)?	*LV*
Initialization	1	20	✓	×	–
1st	2	20	✓	×	2
2nd	3	20	✓	×	1
3rd	4	40	✓	×	0
→ 4th	5	●	●	●	-1

Lecture 12

Part 5

Correctness Proofs of Loops

Correct Loops: Proof Obligations

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

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

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

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

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

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

    $$I \wedge B \Rightarrow R$$

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

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

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

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

Correct Loops: Proof Obligations

```
find_max (a: ARRAY [INTEGER]): INTEGER
  local i: INTEGER
  do
    from
      i := a.lower ; Result := a[i]
    invariant
      loop_invariant: ∀j | a.lower ≤ j < i • Result ≥ a[j]
    until
      i > a.upper
    loop
      if a [i] > Result then Result := a [i] end
      i := i + 1
    variant
      loop_variant: a.upper − i + 1
    end
  ensure
    correct_result: ∀j | a.lower ≤ j ≤ a.upper • Result ≥ a[j]
  end
end
```

Initialization:

{ True } i := a.lower ;

Result := a[i]

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

Before Termination:

Upon Termination:

Non-Negative Variant:

Decreasing Variant: