Lecture 3

Monday January 13

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"; }
```

"cache" the pre-state value of balance

pre-state

post-state

obj. of class

o.m(...)

postcondition

current value of bal.

pre-stored value cached

postcondition

Bank Accounts in Java: Version 5 Critique

Compared with Version 4

```
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"; }
```

Supplier

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"; }
```

error conditions (¬ preconditions)

Single Choice Principle

not public.

post-cond.

class inv.

Supplier

Contracts public

Preconditions
Postconditions
class invariants

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.");
```

all the client cases to do

overhead of client.

Design by Contract in Eiffel

Contract View

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

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

(tracing tag.) Bool expr.

single place.

Implementation View

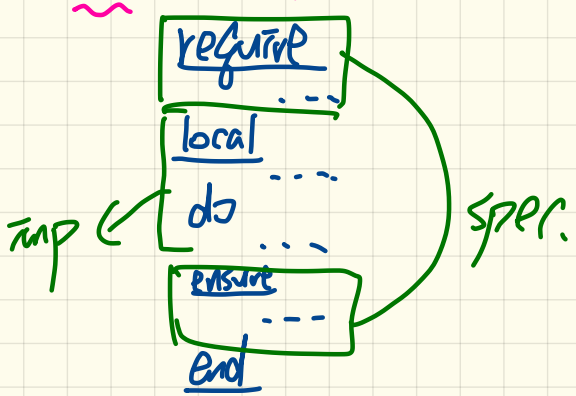
Features in Eiffel

Attributes

balance : INTEGER

Commands (≈ mutators)

set_balance (nb: INTEGER)



Queries (≈ accessors)

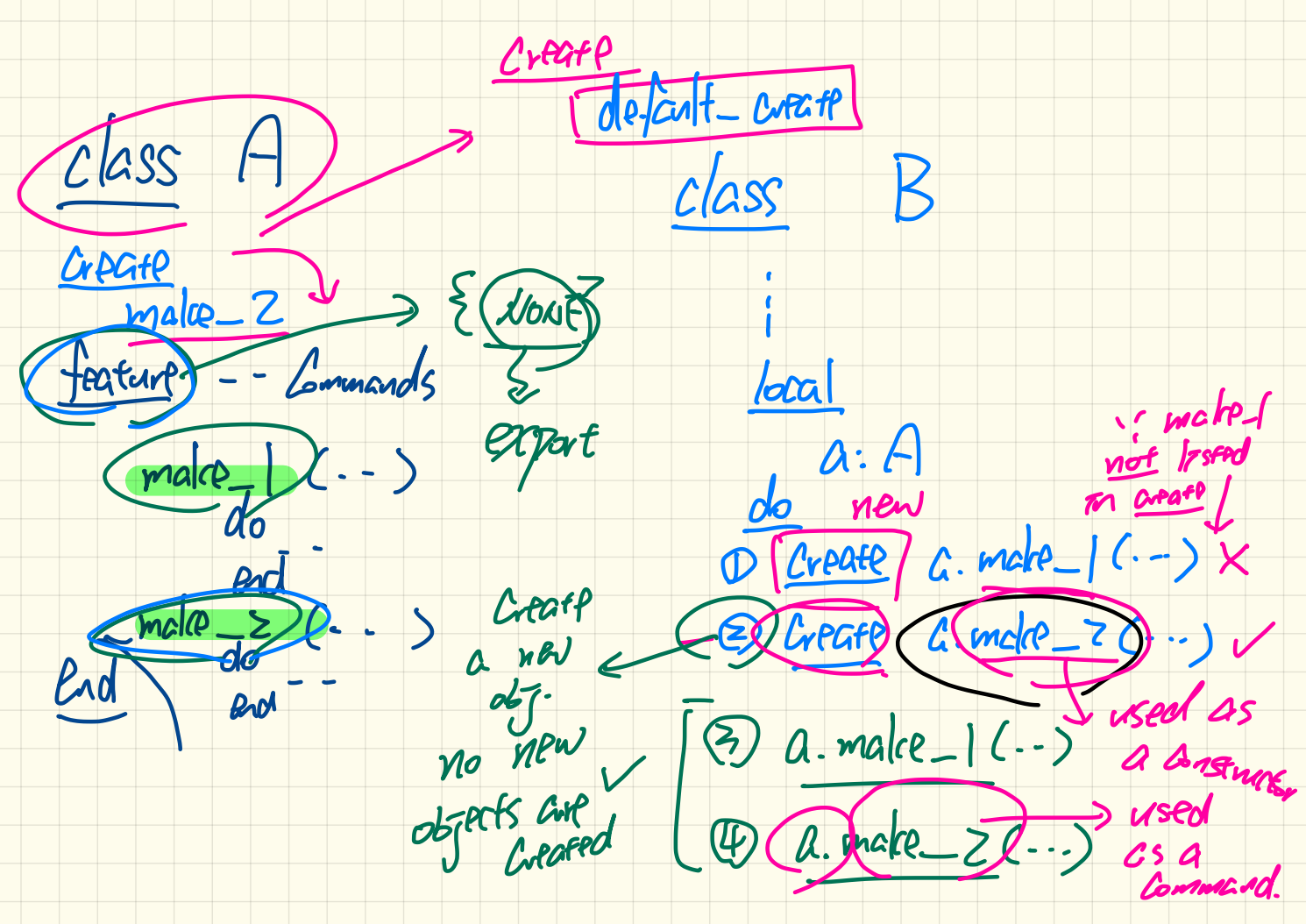
get_balance : INTEGER
require
local
do
ensure
end

Commands vs. Constructors

↳ Commands listed under create clause

```
class A

feature -- Commands

    make_1 (...)
        do
        end

    make_2 (...)
        do
        end

end
```

```
class B

    ⋮

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


class A
create
make_2
feature -- Commands
make_1 (...)
do
end
make_2 (...)
do
end
end
{NONE}
export
create
default_create
class B
local
a: A
do
new
① create
② create
a new obj.
no new objects are created
a.make_1 (...) X
a.make_2 (...) ✓
used as a constructor
③ a.make_1 (...)
④ a.make_2 (...)
used as a command.
: make_1 not used in create

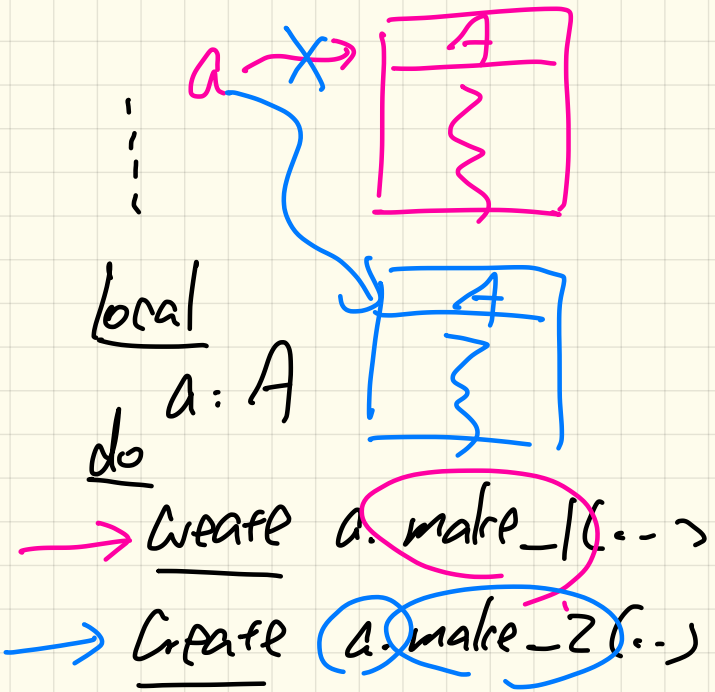
class A

create
make_1, make_2

features

make_1(..)
:

make_2(..)
:



class A

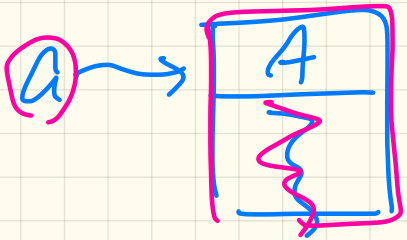
creatp

creatp
make_1, make_2

features
make_1(..)
;
make_2(..)
;

{A} a.make_2(..)

class B
;
local
c : A
do
creatp a.make_2(..)
a.make_1



Java.

A obj = new ?(...);

? may or may not be A.

Eiffel

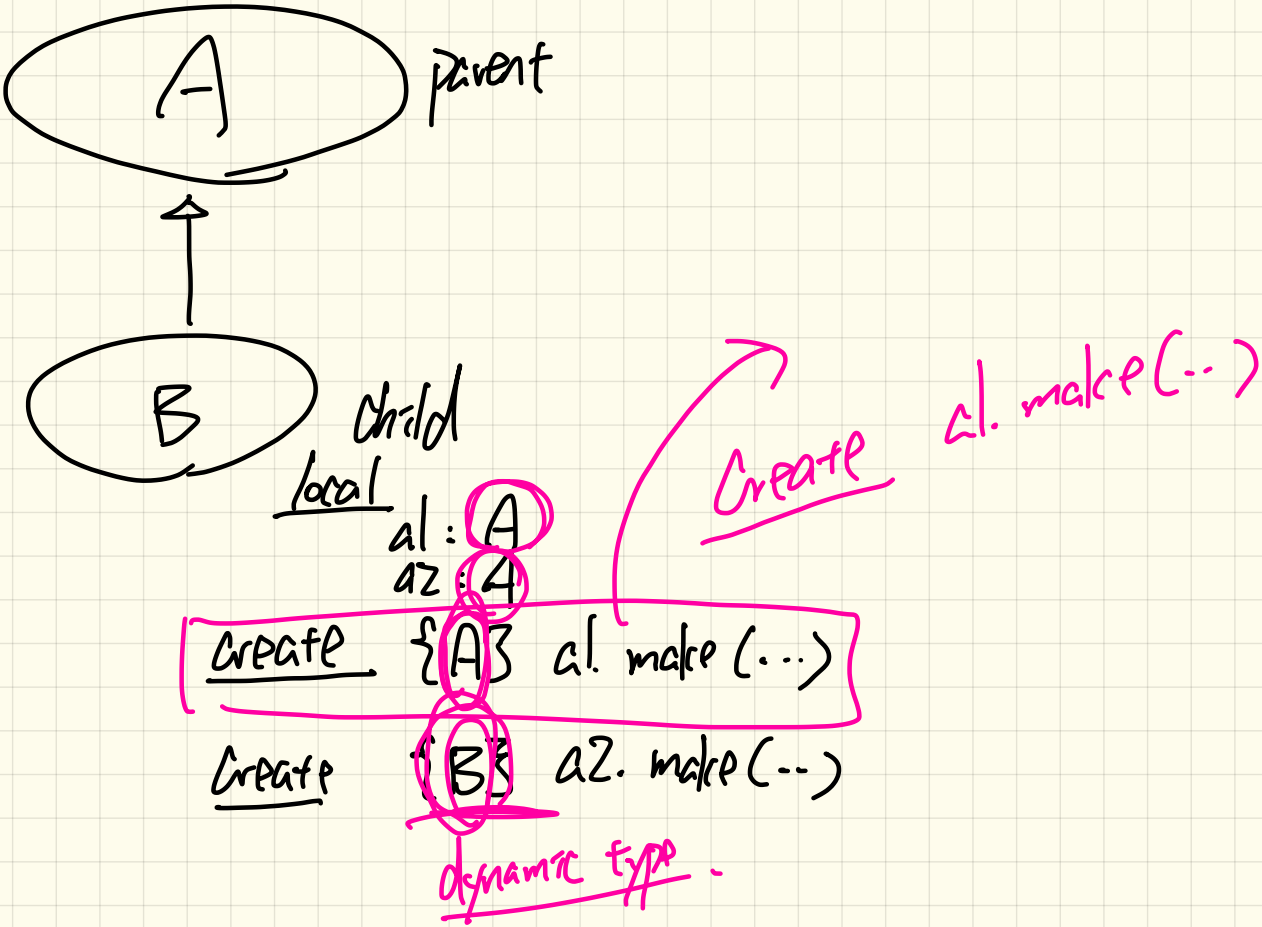
local

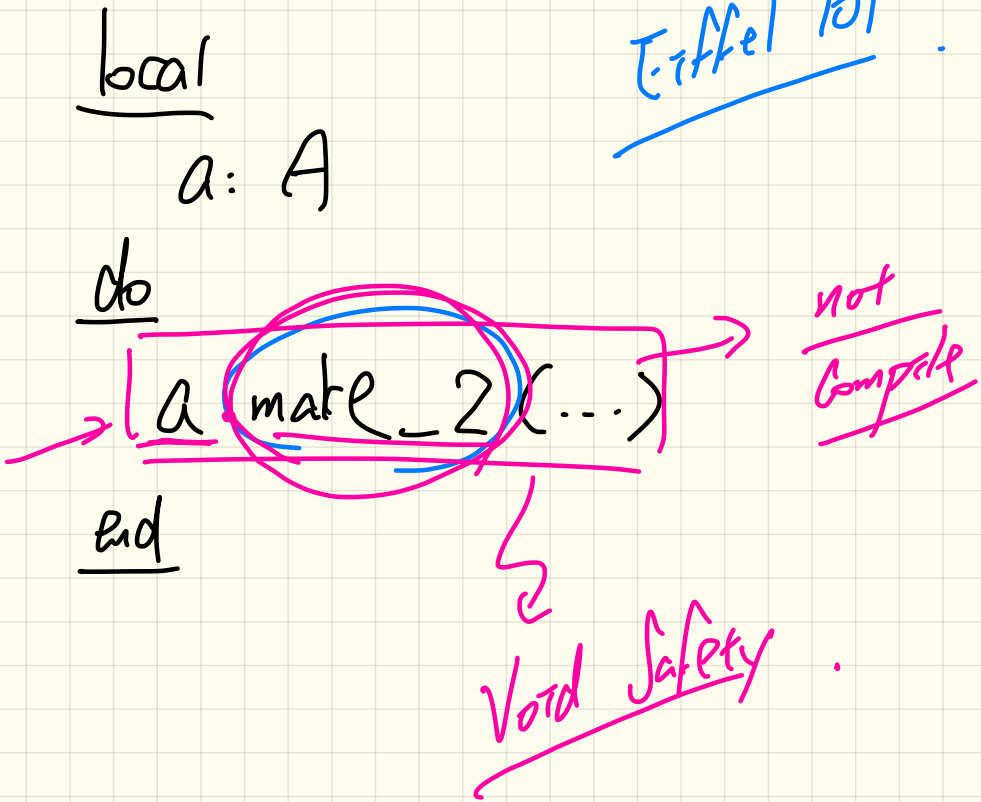
obj: A

do

Create { ? } obj.——(. . .).

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


A
parent
B
child
local
a1 : A
a2 : A
create {A} a1. make (...)
create {B} a2. make (...)
dynamic type.
Create
a1. make (..)


local
a: A
do
a.make_2(...)
end
not Compile
Void Safety
Eiffel 101

Cmd (- - -)
do
- -
end

≈≈

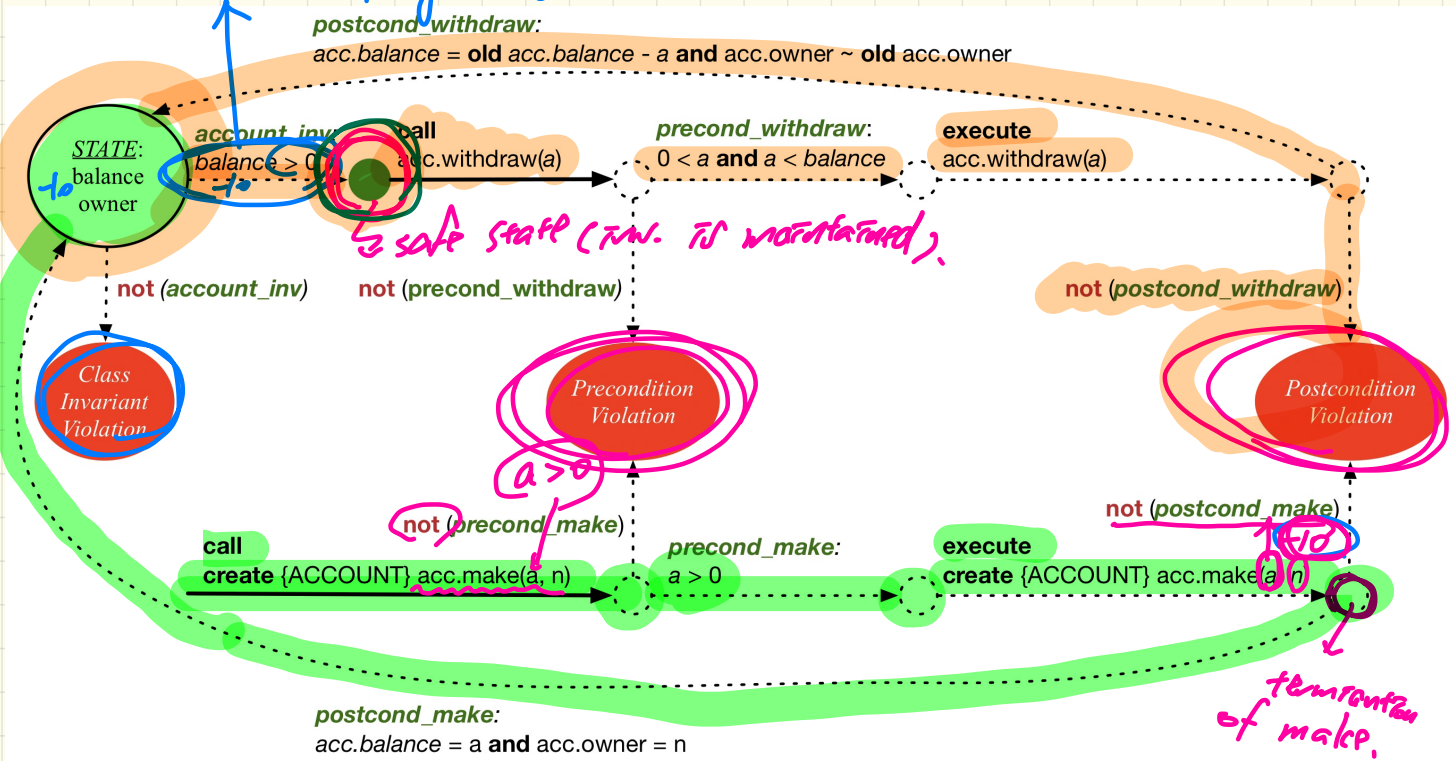
Cmd (. .)
require
True
do
- -
ensure
[True]
end

not appropriate
-: no constraint on suppl.

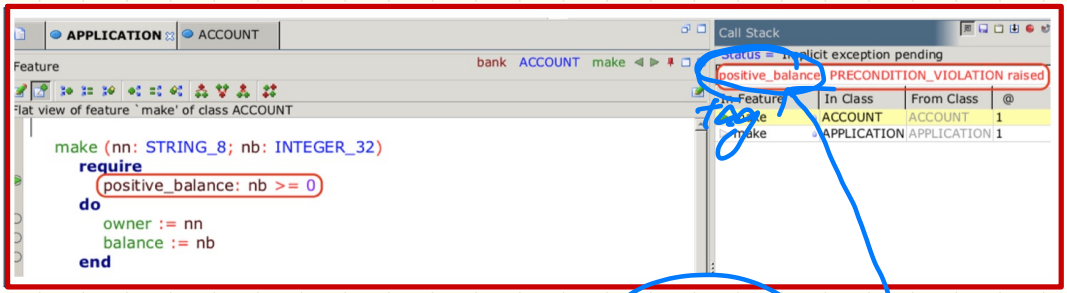
```
class ACCOUNT
create
      make
feature -- Attributes
      owner : STRING
      balance : INTEGER
feature -- Constructors
      make(nn: STRING; nb: INTEGER)
            require -- precondition
                  positive_balance: nb > 0
            end
feature -- Commands
      withdraw(amount: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

Runtime Monitoring of Contracts

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



Precondition Violation: positive_balance



Supplier

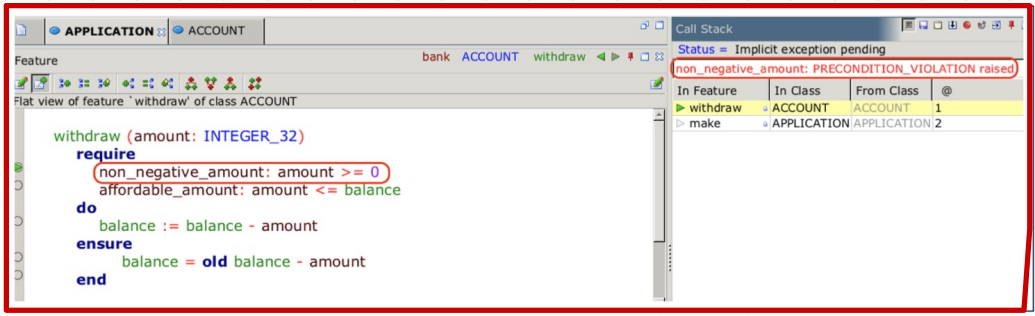
```
class ACCOUNT
create
        make
feature -- Attributes
        owner : STRING
        balance : INTEGER
feature -- Constructors
        make(nn: STRING; nb: INTEGER)
                require -- precondition
                        positive_balance: nb > 0
                end
feature -- Commands
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

Client

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

Precondition Violation:

non_negative_amount



Supplier

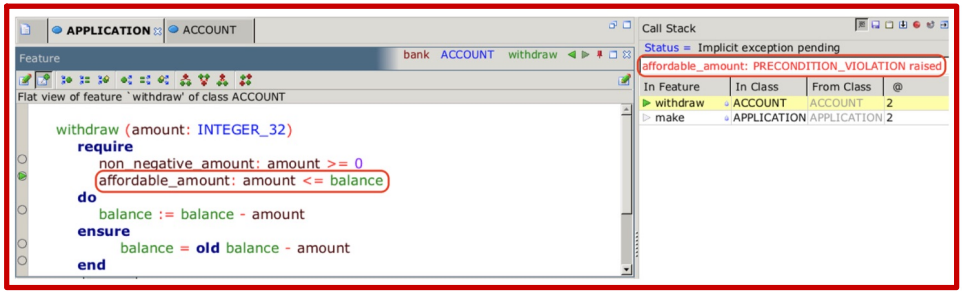
```
class ACCOUNT
create
	make
feature -- Attributes
	owner : STRING
	balance : INTEGER
feature -- Constructors
	make(nn: STRING; nb: INTEGER)
		require -- precondition
			positive_balance: nb > 0
		end
feature -- Commands
	withdraw(amount: INTEGER)
		require -- precondition
			non_negative_amount: amount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

Client

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

Precondition Violation:

affordable_amount



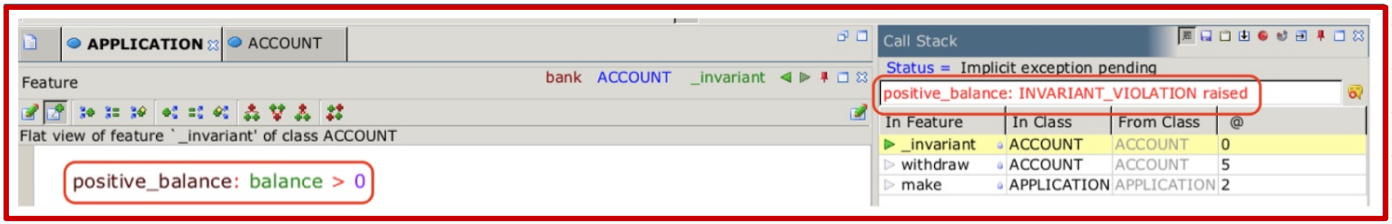
Supplier

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

Client

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

Class Invariant Violation: positive_balance



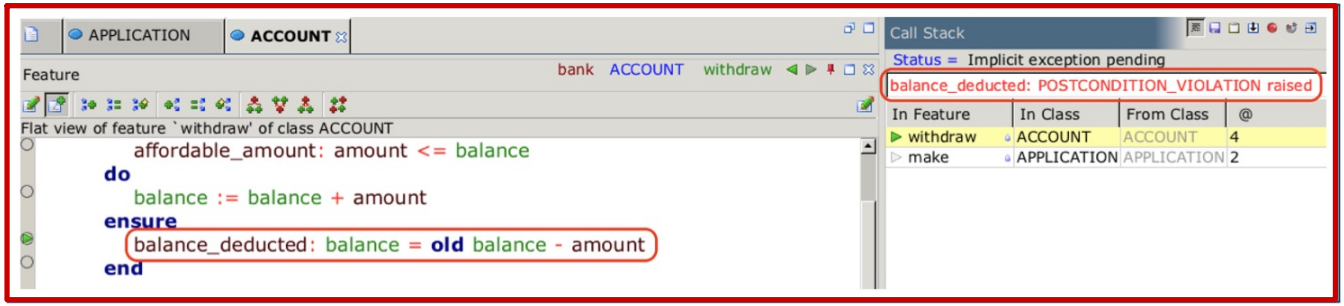
Supplier

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

Client

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

Postcondition Violation: balance_deducted



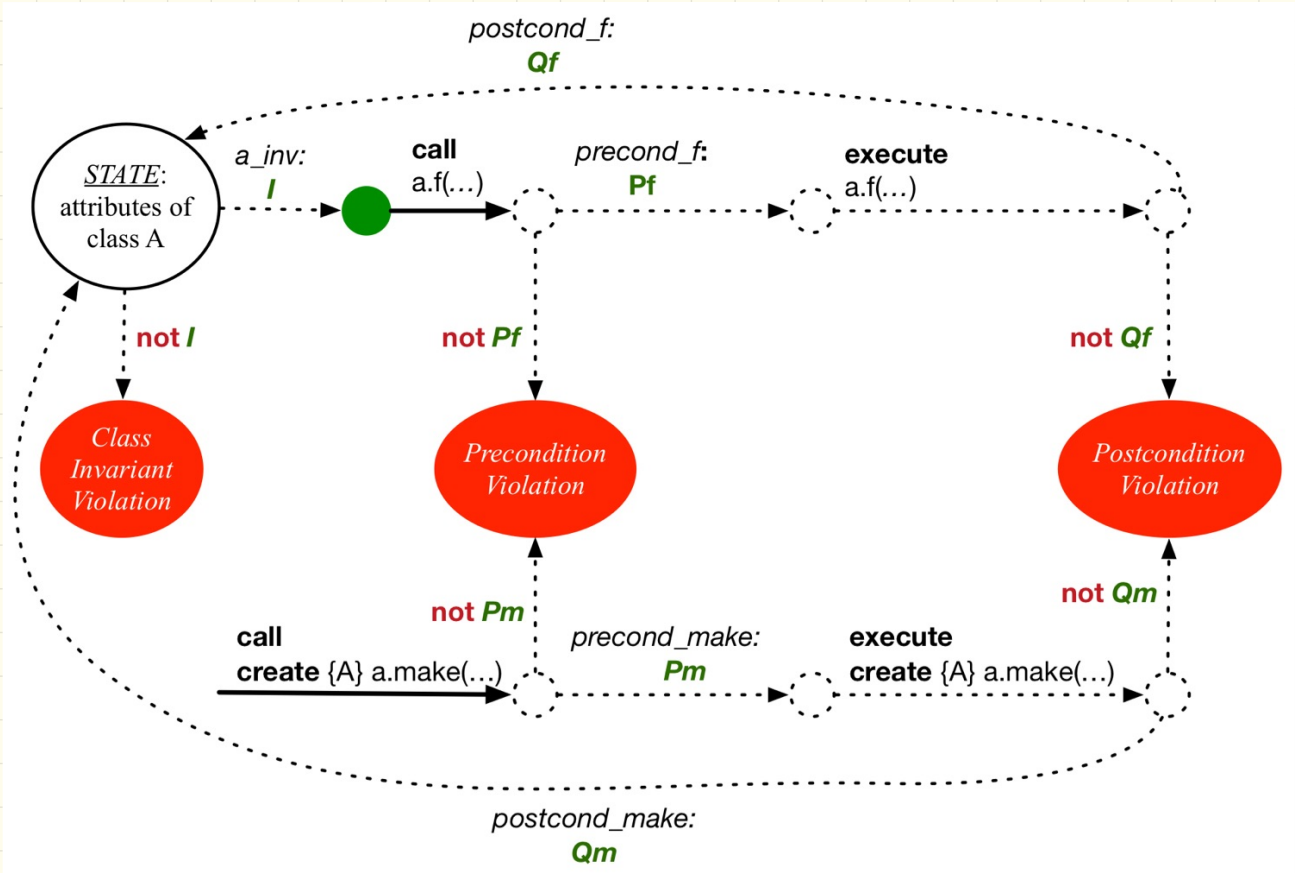
Supplier

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

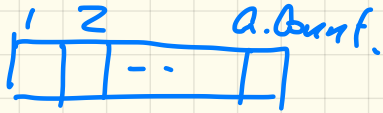
Client

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

Runtime Monitoring of Contracts



Precondition & Postcondition Exercise



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

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

ensure
??
a[i] ~ ns

change_at_i: a[i] ~ ns ✓

a

"Alan"

"Mark"

"Junk"

change_at(a, 5, "Mark")

Also. ∀ ∃

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