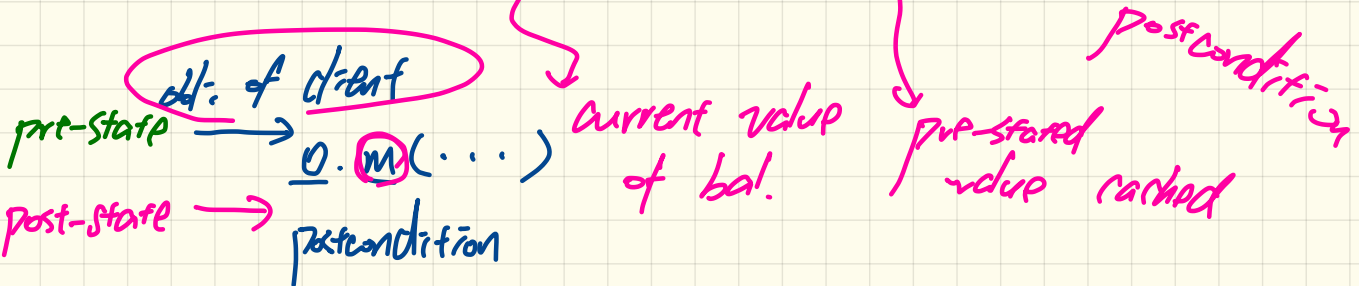


# LECTURE 3

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"; }
```



Compared  
with  
Version 4

# Bank Accounts in Java: Version 5 Critique

```
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

Supplier

```
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, WithdrawAmountTooLargeException  
    {  
        int oldBalance = this.balance;  
        if (amount < 0) { /* negated precondition */  
            throw new WithdrawAmountNegativeException(); }  
        else if (balance < amount) { /* negated precondition */  
            throw new WithdrawAmountTooLargeException(); }  
        else { this.balance = this.balance - amount; }  
        assert this.getBalance() > 0 : "Invariant: positive balance";  
        assert this.getBalance() == oldBalance - amount :  
            "Postcondition: balance deducted"; }  
}
```

Single  
Method  
Principle

make public.

**Supplier**

Contract's public  
preconditions  
postconditions  
class invariants

all the client wants to do  
overhead of client.

post-Cond. class inv.

**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
  → positive_balance: balance > 0
end
```

(tracing)  
tag.  
Bad expr.

single  
↑  
plate.

## Implementation View

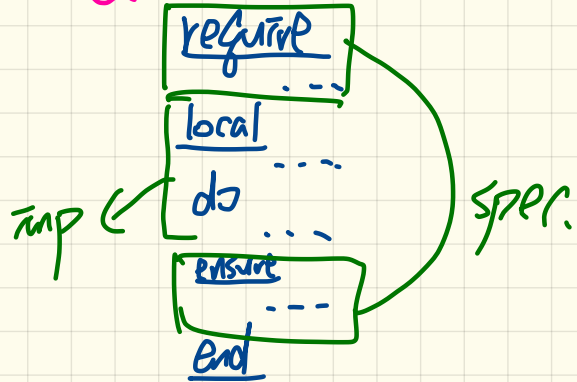
# Features in Eiffel

Commands (≈ mutators)

## Attributes

balance : INTEGER

set\_balance (nb: INTEGER)



Queries (≈ accessors)

`get_balance` : INTEGER

`require`

`local`

`do`

`ensure`

`end`

# Commands vs. Constructors

↳ Commands listed under CREATE clause

class A

class B

feature -- Commands

make\_1 (..)

do

end

make\_2 (..)

do

end

End

;  
local

do a: A

CREATE a.make\_1 (..) X

CREATE a.make\_2 (..) X

CREATE

default\_CREATE

class A

CREATE

make\_2

feature

-- Commands

{ None }

export

make\_1 (...) do

end

make\_2 (...) do

end

End

class B

local

do a: A new

① CREATE

a.make\_1 (...) X

② CREATE

a.make\_2 (...) ✓

∴ make\_1 not used for create ↓

used as a consumer

③ a.make\_1 (...) ✓

④ a.make\_2 (...) ✓

used as a command.

CREATE a new obj.  
no new objects are created ✓



class A

createp

make\_1, make\_2

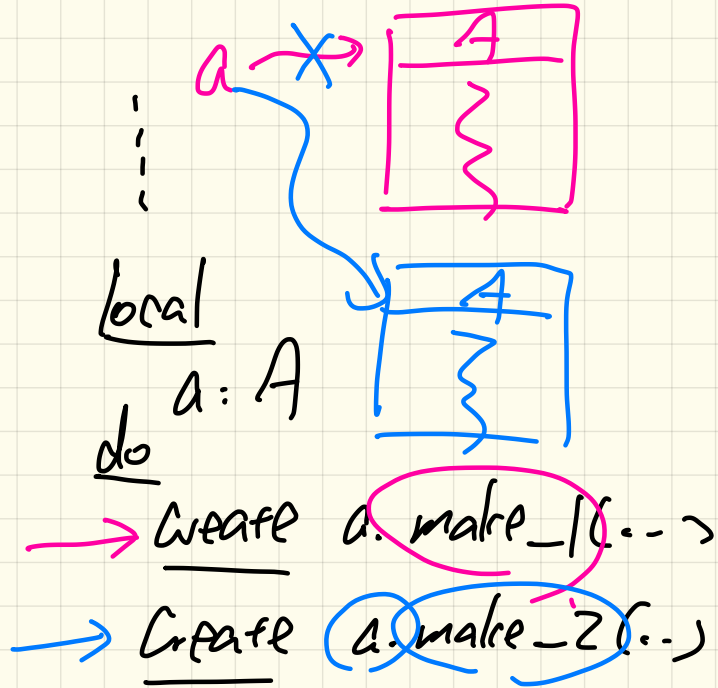
features

make\_1(..)

;

make\_2(..)

;



class A createp

{A} a.make\_2(...)

createp  
make\_1, make\_2

features

make\_1(...)  
;

make\_2(...)  
;

class, B

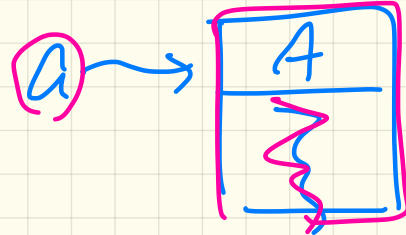
local

c : A

do

createp a.make\_2(...)

a.make\_1



Java.

A obj

= new

?(...);

may or may not be A.

Eiffel

Local

do obj: A

Create

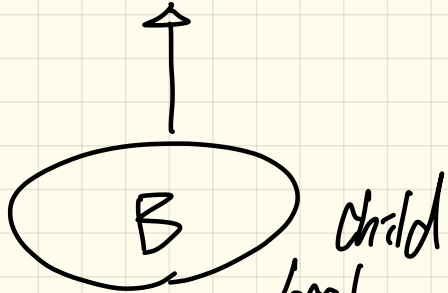
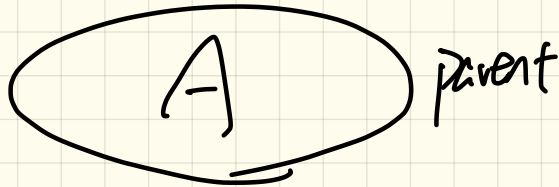
{ ? }

obj.

— ( — )

{ ? }

When the "?" same as the static type of obj, you can omit it.



local  
a1: A  
a2: A

create {A} a1. make (...)

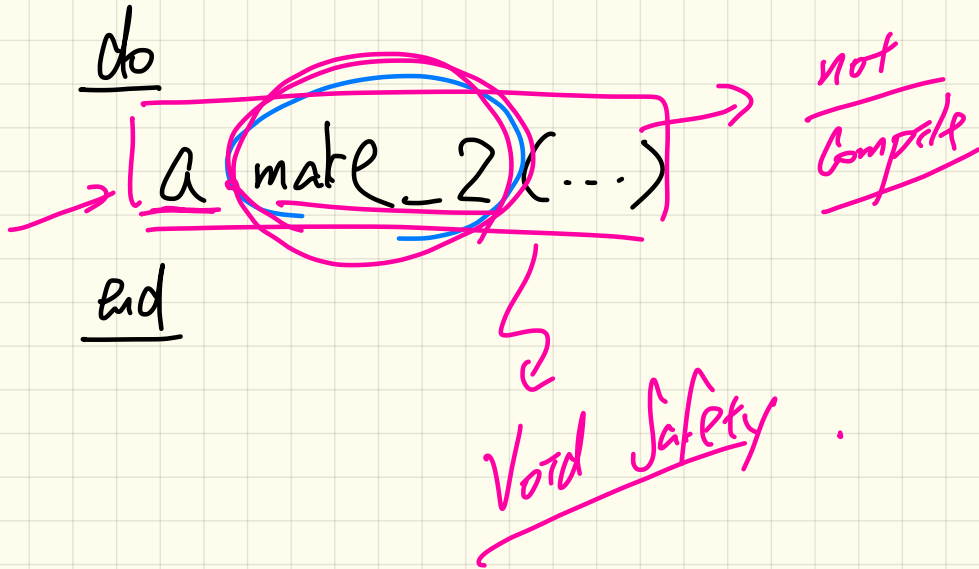
create {B} a2. make (...)

dynamic type .

create a1. make (...)

local  
a: A

Tippel lol



Cmd (---)

do

---

end



Cmd (...)

require

True

do

---

ensure

True

end

not appropriate  
∵ no constraint on supply.

```

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
end
invariant -- class invariant
  positive_balance: balance > 0
end

```

cur. bal.  
100

0

# Runtime Monitoring of Contracts

```

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

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

*postcond\_withdraw:*  
 $acc.balance = \text{old } acc.balance - a \text{ and } acc.owner \sim \text{old } acc.owner$

STATE: balance owner

account\_inv:  $balance > 0$

call acc.withdraw(a)

*precond\_withdraw:*  
 $0 < a \text{ and } a < balance$

execute acc.withdraw(a)

safe state (inv. is maintained)

not (account\_inv)

not (precond\_withdraw)

not (postcond\_withdraw)

Class Invariant Violation

Precondition Violation  
 $a > 0$

Postcondition Violation

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

not (precond\_make)

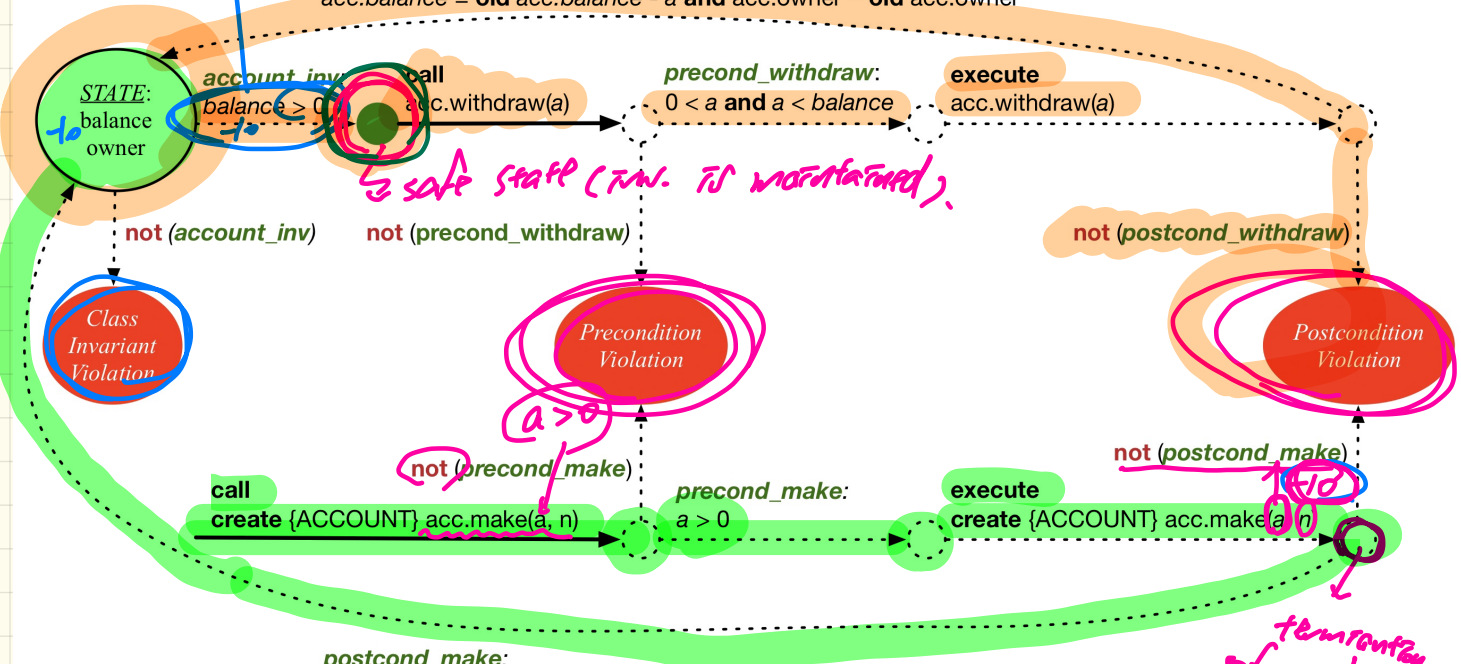
*precond\_make:*  
 $a > 0$

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

not (postcond\_make)

*postcond\_make:*  
 $acc.balance = a \text{ and } acc.owner = n$

termination of make.





# Precondition Violation: positive\_balance

Feature: bank ACCOUNT make

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

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

Call Stack: status = implicit exception pending

Supplier

Client

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

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

Tag

-10

# Precondition Violation:

## non\_negative\_amount

```
APPLICATION ACCOUNT withdraw  
Feature  
Flat view of feature 'withdraw' of class ACCOUNT  
withdraw (amount: INTEGER_32)  
  require  
    non_negative_amount: amount >= 0  
    affordable_amount: amount <= balance  
  do  
    balance := balance - amount  
  ensure  
    balance = old balance - amount  
end
```

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

## Supplier

## 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 "non_negative_amount"  
  mark.withdraw(-1000000)  
end  
end
```

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

# Precondition Violation:

affordable\_amount

```
APPLICATION ACCOUNT  
Feature bank ACCOUNT withdraw  
Flat view of feature 'withdraw' of class ACCOUNT  
withdraw (amount: INTEGER_32)  
  require  
    non_negative_amount: amount >= 0  
    affordable_amount: amount <= balance  
  do  
    balance := balance - amount  
  ensure  
    balance = old balance - amount  
end
```

Call Stack  
Status = Implicit exception pending  
affordable\_amount: PRECONDITION\_VIOLATION raised  
In Feature | In Class | From Class | @  
withdraw | ACCOUNT | ACCOUNT | 2  
make | APPLICATION | APPLICATION | 2

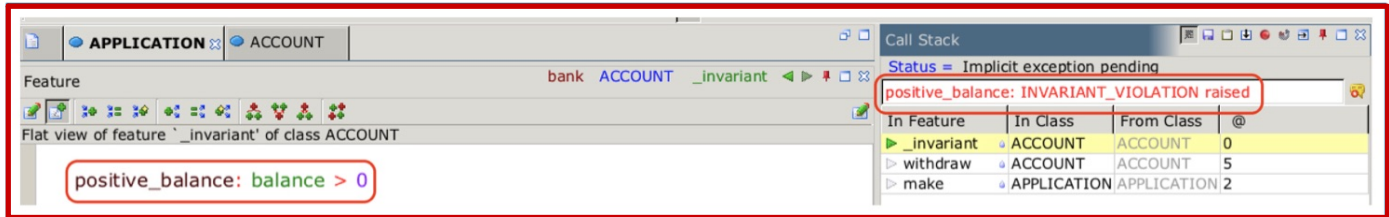
## Supplier

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

## Client

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

# Class Invariant Violation: **positive\_balance**



positive\_balance: balance > 0

Call Stack

Status = Implicit exception pending

positive\_balance: INVARIANT\_VIOLATION raised

In Feature	In Class	From Class	@
▶ <b>_invariant</b>	ACCOUNT	ACCOUNT	0
▶ withdraw	ACCOUNT	ACCOUNT	5
▶ make	APPLICATION	APPLICATION	2

## Supplier

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

## Client

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

# Postcondition Violation: **balance\_deducted**

The screenshot shows a development environment with two panes. The left pane displays the source code for the `ACCOUNT` class, specifically the `withdraw` feature. The code includes a precondition `affordable_amount: amount <= balance`, a `do` block where `balance := balance + amount`, and a postcondition `ensure balance_deducted: balance = old balance - amount`. The `ensure` line is circled in red. The right pane shows the Call Stack, with the status `Implicit exception pending` and a message `balance_deducted: POSTCONDITION_VIOLATION raised` circled in red. Below this, the call stack table is visible:

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

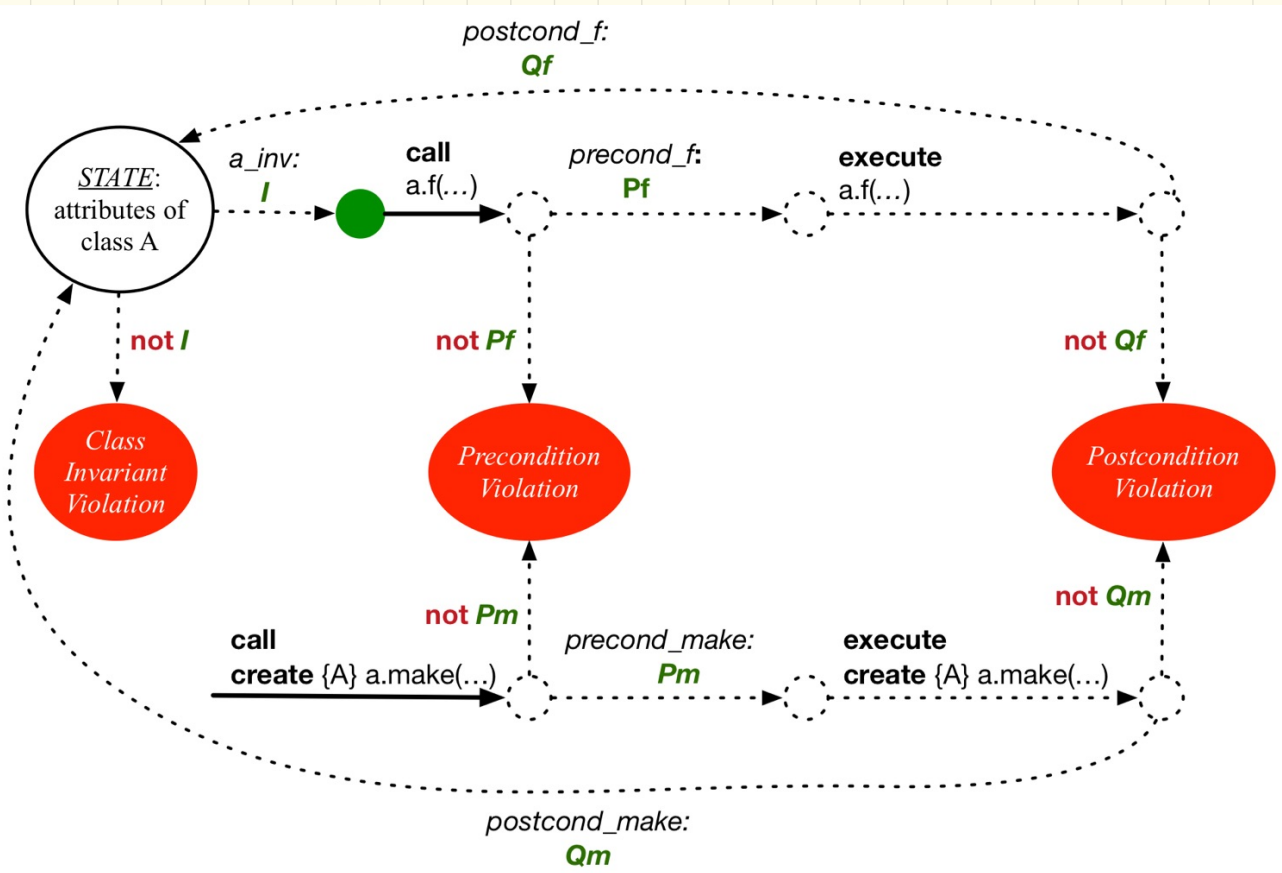
## Client

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

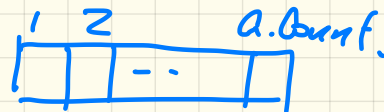
## Supplier

```
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
```

# 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:  $1 \leq i$  and  $i \leq a.length$

ensure

??

$a[i] \sim ns$

