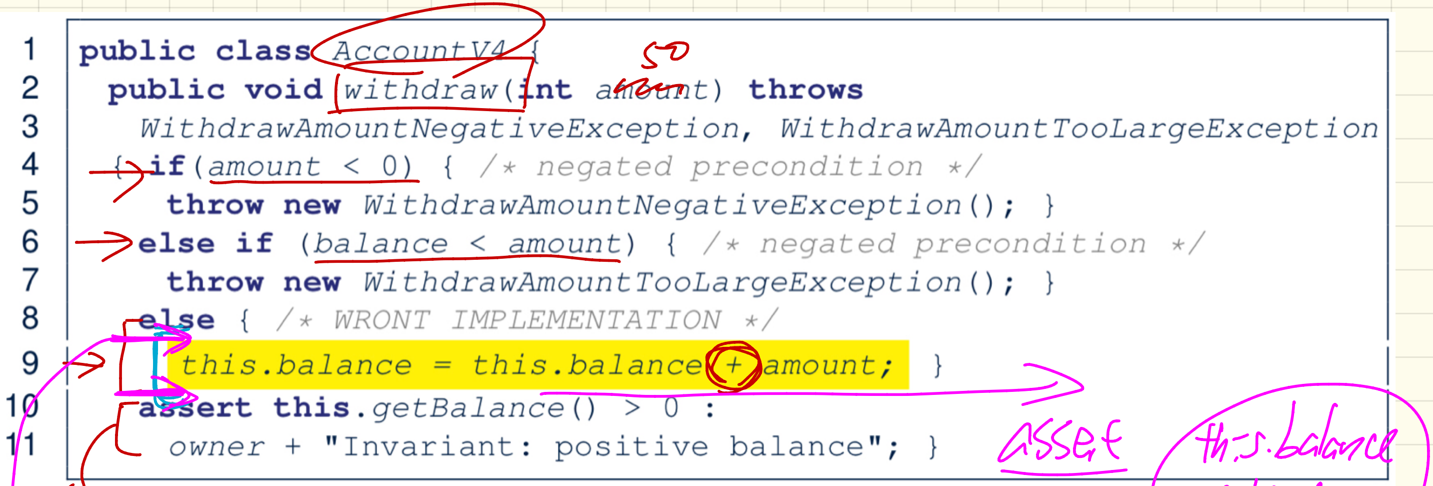
Monday January 14

Lecture 3

Bank Accounts in Java : Version 4

(with an evil supplier)



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

50

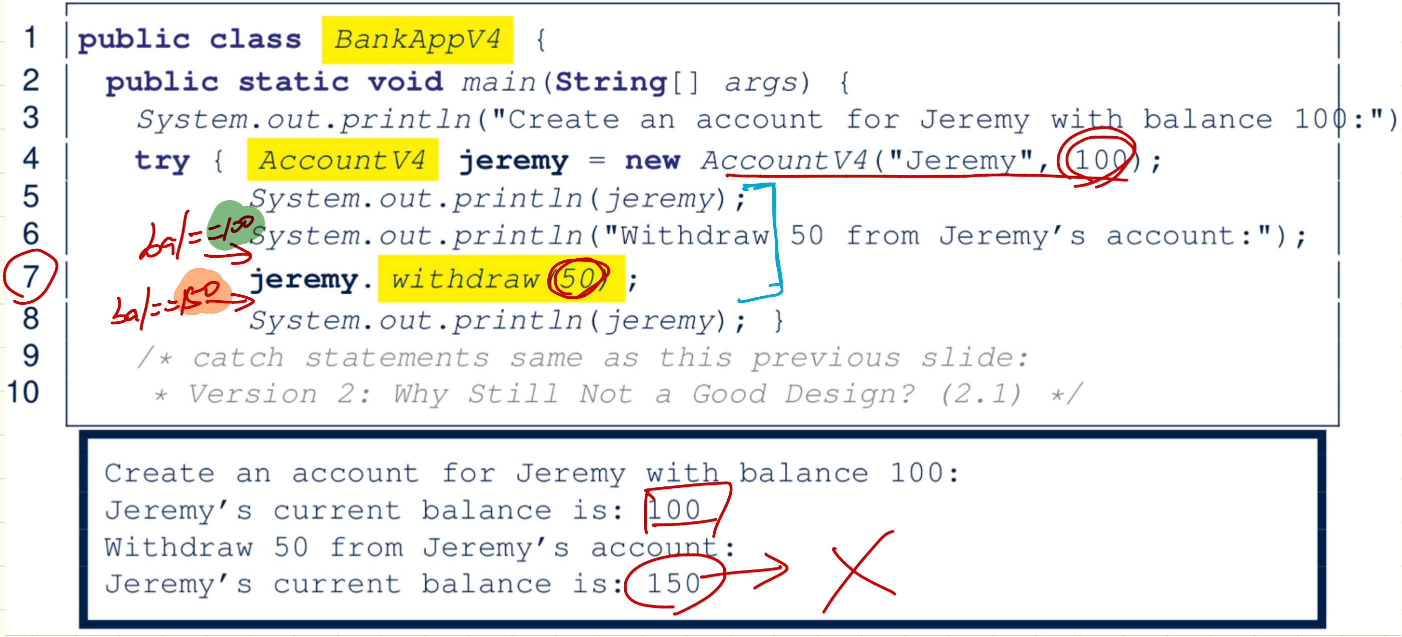
asset

this.balance
oldBalance

Inv-

int oldBalance = this.balance;

Bank Accounts in Java : Version 4 Critique



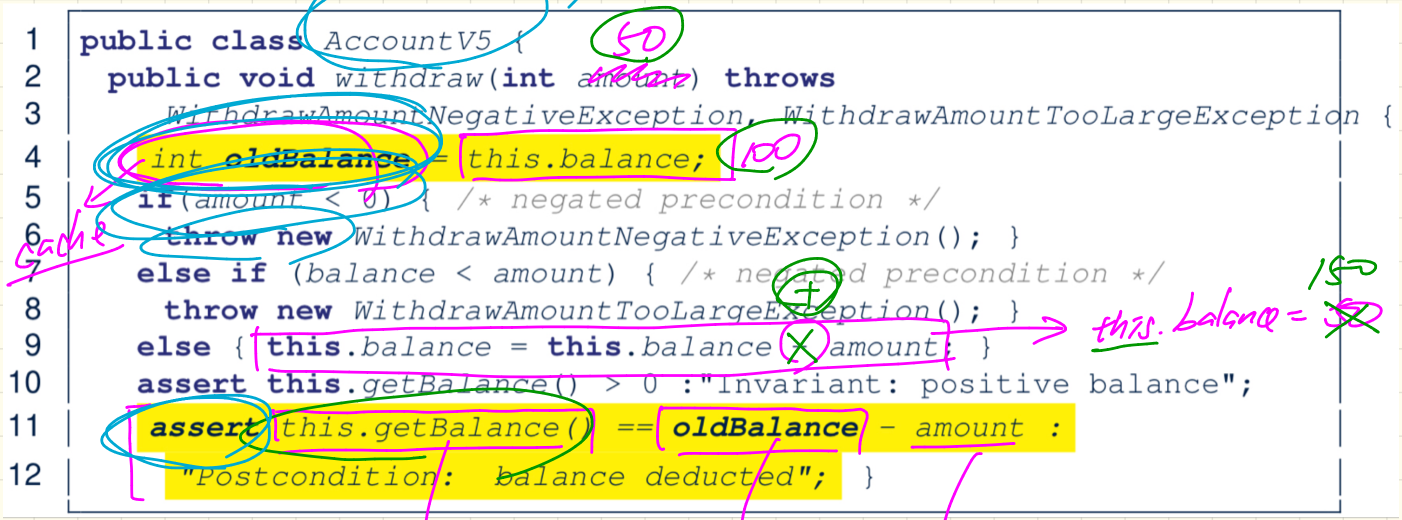
```java
public class BankAppV4 {
  public static void main(String[] args) {
    System.out.println("Create an account for Jeremy with balance 100:")
    try { AccountV4 jeremy = new AccountV4("Jeremy", 100);
          System.out.println(jeremy);
          System.out.println("Withdraw 50 from Jeremy's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

bal == 100

bal == 150

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

Bank Accounts in Java : Version 5



```
public class AccountV5 {
  public void withdraw(int amount) throws
      WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
    int oldBalance = this.balance;
    if(amount < 0) { /* negated precondition */
      throw new WithdrawAmountNegativeException(); }
    else if (balance < amount) { /* negated precondition */
      throw new WithdrawAmountTooLargeException(); }
    else { this.balance = this.balance - amount; }
    assert this.getBalance() > 0 :"Invariant: positive balance";
    assert this.getBalance() == oldBalance - amount :
      "Postcondition: balance deducted"; }
```

50

100

cache

+

this.balance = ~~50~~ 150

~~50~~ 150

100

50

F

≠ ~~50~~ 150 == 100 - 50

T

int divide(int x, int y)

Result

<u>ensure</u>

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

boolean binSearch(int x, int[] xs)

<u>ensure</u>

such that

it is the case

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

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

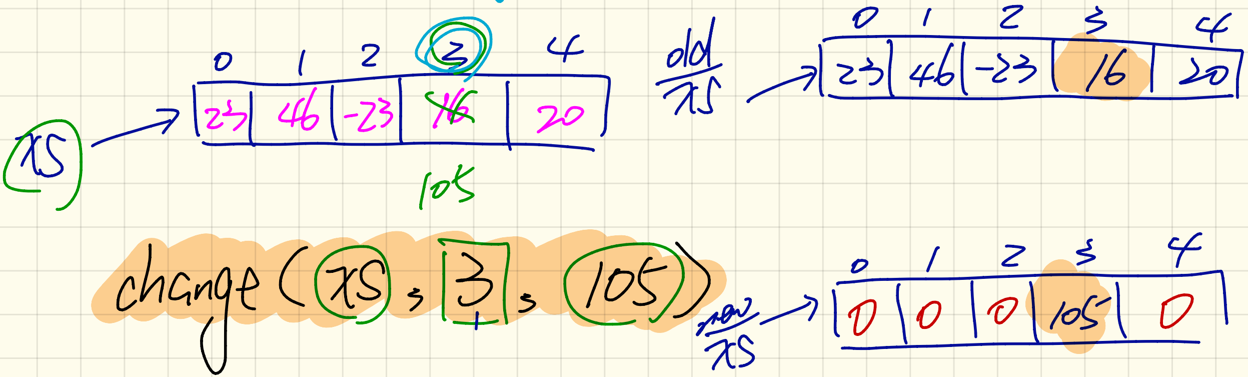
void change (int[] xs , int i , int x)

<u>require</u>

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

<u>ensure</u>

changed : xs[i] = x



change(xs, 3, 105)

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

require

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

ensure

changed : $xs[i] = x$

$j == 0$

others unchanged :

~~change(xs , 3 , 105)~~

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

$xs[j] = old\ xs[j]$

3 4 4

0 1 2

old xs

0	1	2	3	4
23	46	-23	16	23

$23 == 0$

new xs

0	1	2	3	4
0	0	0	105	0

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

$$\boxed{\forall j} \mid \boxed{0 \le j < xs.length} \bullet$$

$$\boxed{j \neq i \Rightarrow xs[j] = \underline{old}\ xs[j]}$$

$\underline{across}$ 0 |..| (xs.length − 1) $\underline{as}$ $\boxed{j}$ → Integer cursor

$\underline{all}$

$\boxed{\cancel{j}}$ /= $\boxed{i}$ $\underline{implies}$ xs[$\cancel{j}$] = $\underline{old}$ xs[$\cancel{j}$]

j.item j.item j.item

$\underline{end}$

boolean allPositive (int[] xs)

ensure

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

all xs[~~i~~] > 0

i.item

end)

-1

$-1 - 0 + 1 = \boxed{0}$

[1, 10]

10 - 1 + 1

[x, y]

y - x + 1

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

→ allPositive (<< >>)

allPos (<< >>)

SomePos (<<-2, $\boxed{3}$, -4, -8>>)

SomePos (<< >>) F T

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

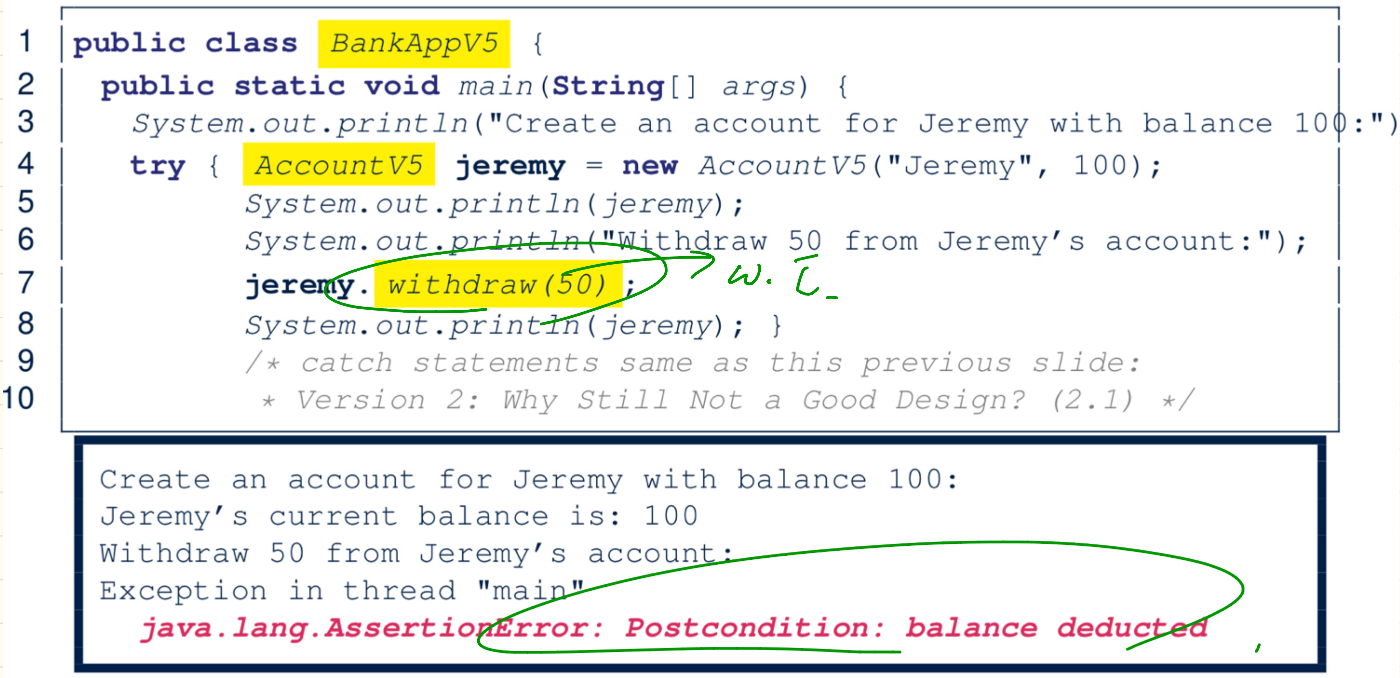
↳ "there is no such element $x \in \emptyset$ that can falsify $P(x)$

witness

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

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

Bank Accounts in Java: Version 5 Critique (Compared with Version 4)



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

w. C_

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

Design by Contract in Eiffel

Implementation View

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

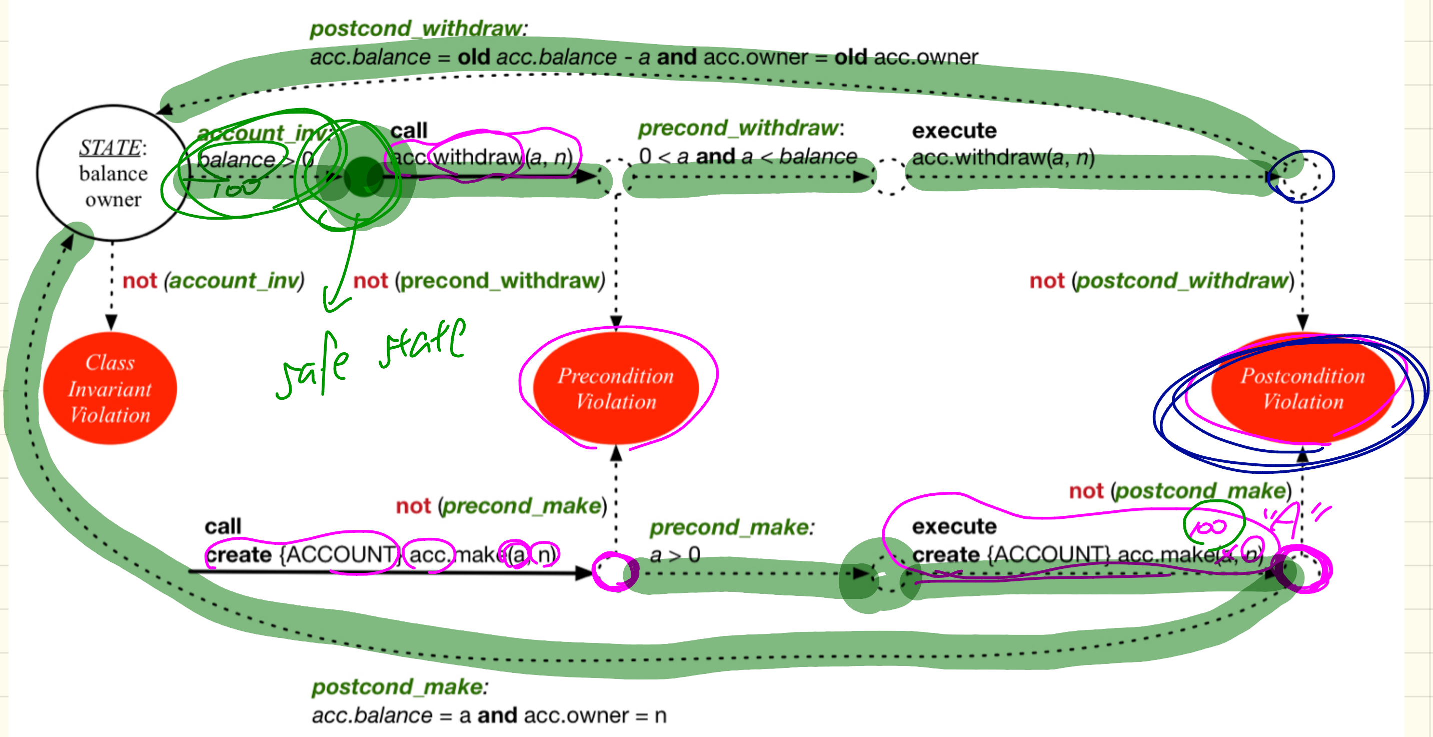
Specification

Implementation

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

Contract View

Runtime Monitoring of Contracts



postcond_withdraw:
acc.balance = **old** acc.balance - a **and** acc.owner = **old** acc.owner

STATE:
balance
owner

account_inv:
balance > 0

call
acc.withdraw(*a*, *n*)

precond_withdraw:
0 < *a* **and** *a* < *balance*

execute
acc.withdraw(*a*, *n*)

not (**account_inv**)

not (**precond_withdraw**)

not (**postcond_withdraw**)

Class Invariant Violation

Precondition Violation

Postcondition Violation

not (**precond_make**)

not (**postcond_make**)

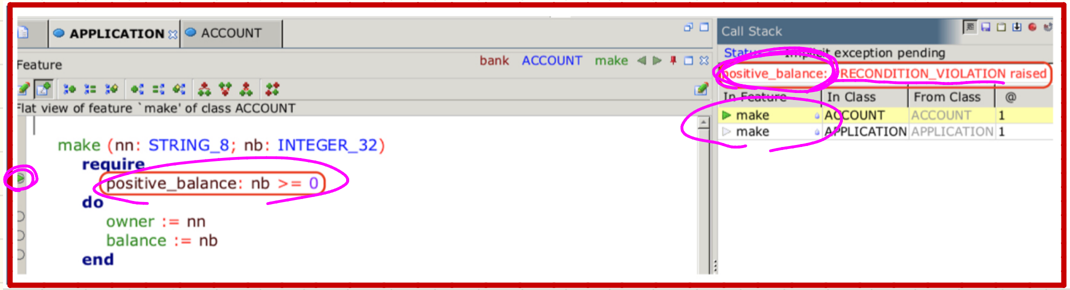
call
create {ACCOUNT} acc.make(a, n)

precond_make:
a > 0

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

postcond_make:
acc.balance = a **and** acc.owner = n

Precondition Violation (1)



Flat view of feature `make' of class ACCOUNT

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

Status = Implicit exception pending

positive_balance: PRECONDITION_VIOLATION raised

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

✓ Supplier

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

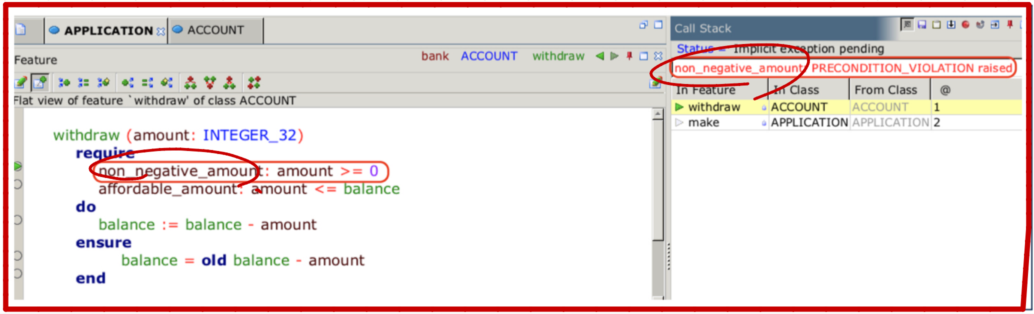
-10

F

✓ Client

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

Precondition Violation (2)



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

non_negative_amount PRECONDITION_VIOLATION raised

In Feature	In Class	From Class	@
withdraw	ACCOUNT	ACCOUNT	1
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
```

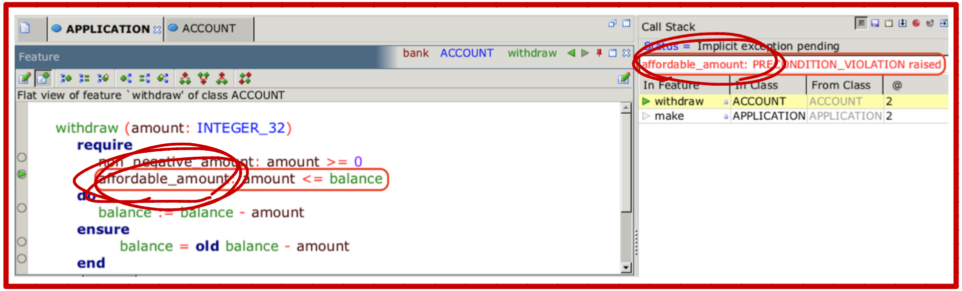
-1000

F

Client

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

Precondition Violation (3)



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
```

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
```

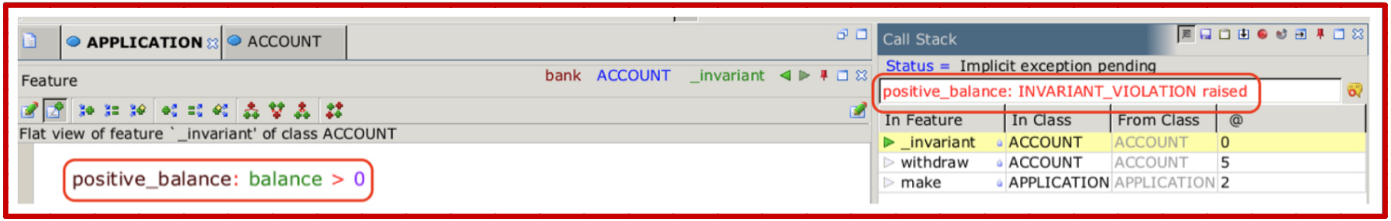
F

150 100

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



Flat view of feature `_invariant' of class ACCOUNT

positive_balance: balance > 0

Status = Implicit exception pending

positive_balance: INVARIANT_VIOLATION raised

In Feature	In Class	From Class	@
▶ _invariant	ACCOUNT	ACCOUNT	0
▷ withdraw	ACCOUNT	ACCOUNT	5
▷ make	APPLICATION	APPLICATION	2

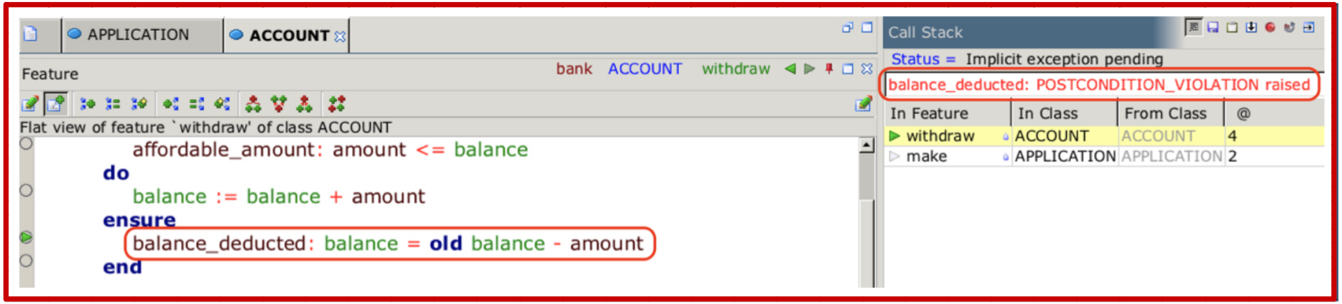
Supplier

```
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



Flat view of feature `withdraw' of class ACCOUNT

```
        affordable_amount: amount <= balance
    do
        balance := balance + amount
    ensure
        balance_deducted: balance = old balance - amount
    end
```

Status = Implicit exception pending

balance_deducted: POSTCONDITION_VIOLATION raised

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

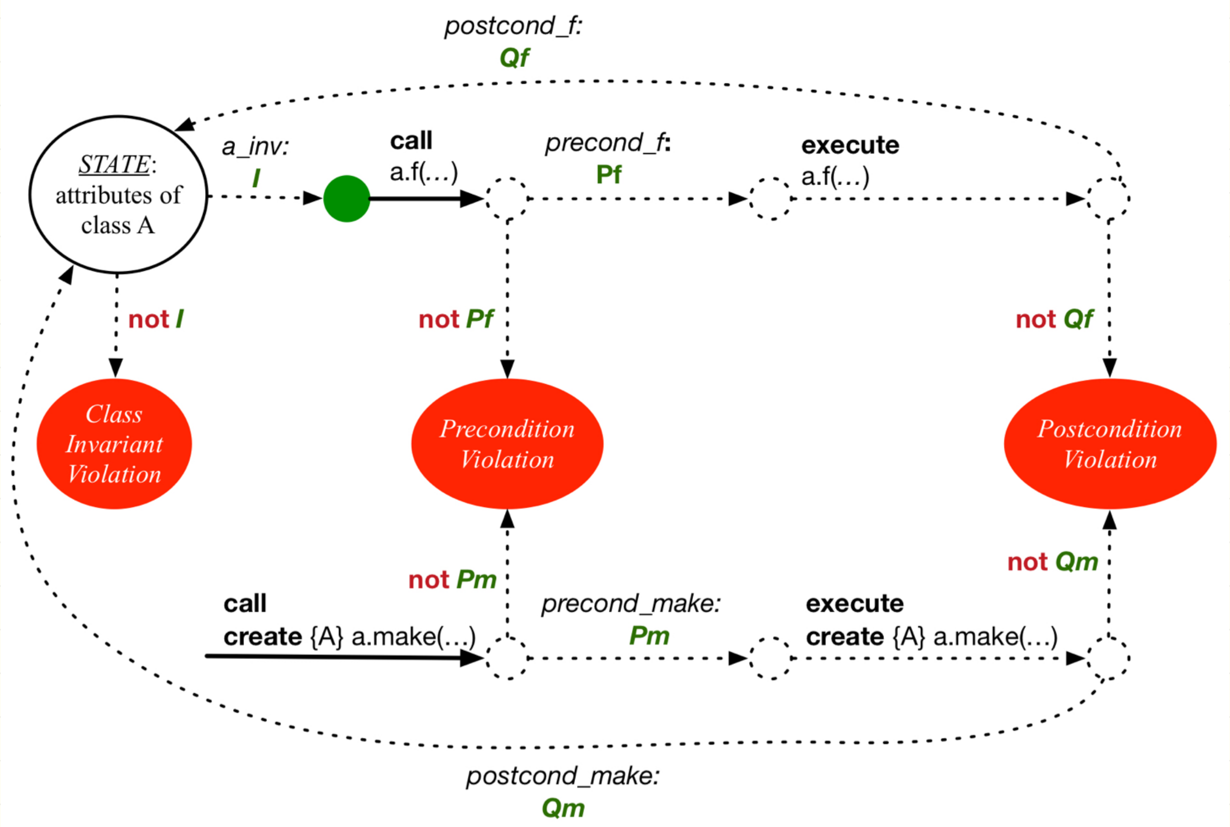
Supplier

```
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
    jeremy: ACCOUNT
  do
    -- Faulty implementation of withdraw in ACCOU
    -- balance := balance + amount
    create {ACCOUNT} jeremy.make ("Jeremy", 100)
    jeremy.withdraw(150)
    -- A postcondition violation with tag "balance_deducted"
  end
end
```

Runtime Monitoring of Contracts



Math_ = Eiffel

:= ×

f : INTEGER

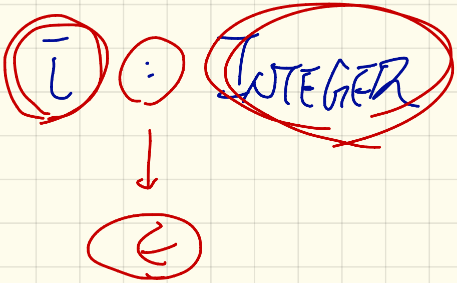
```
local
    . _
require
    - __
do
    - - .
ensure
end - - -
```

```
require ______
Imp: local
do
ensure ______
end
```

```
int i;

int i = 5;
```

```
local
    i : INTEGER
do
    i := 5
```



Logic .

$\neg p \vee$

$p \wedge q$

$p \vee q$

→

Java .

p && q .

p || q

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

require.

✓ $\underbrace{0 <= i}_{p}$ && $\underbrace{i < xs.length}_{q}$

&& xs[i] > 0

$\underbrace{0<=i}_{p}$ && xs[i]>0

&& $\underbrace{i < xs.length}_{q}$

p	q	p ∧ q
T	T	T
T	F	F
F	T	F
F	F	F