Tuesday Sep. 11

Lecture 2

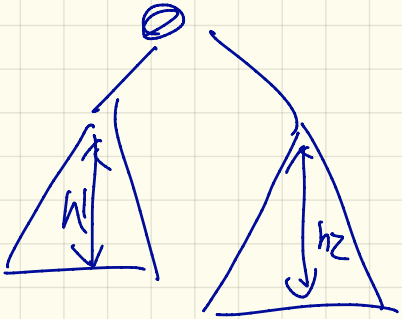
- Lab 1 tonight or tmw morning

- No office hours today

Requirements of Bank

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

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



$|h1 - h2| \leq 2$

global property for all Account objects.

Version 1: No Contracts

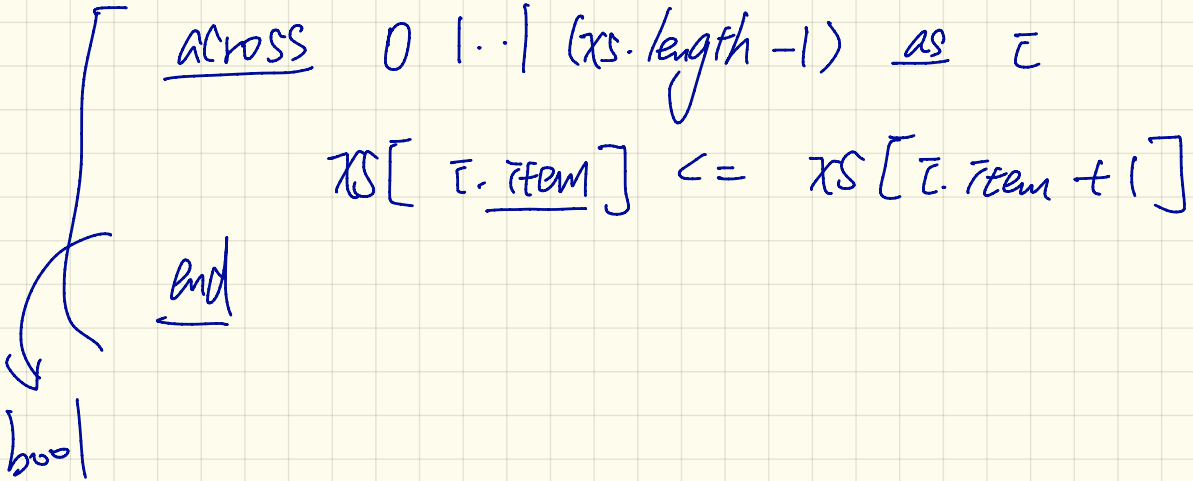
```
public class AccountV1 {
	private String owner;
	private int balance;
	public String getOwner() { return owner; }
	public int getBalance() { return balance; }
	public AccountV1(String owner, int balance) {
		this.owner = owner; this.balance = balance;
	}
	public void withdraw(int amount) {
		this.balance = this.balance - amount;
	}
	public String toString() {
		return owner + "'s current balance is: " + balance;
	}
}
```

Q: int[] xs sorted?

$\forall i : 0 \le i < xs.length \cdot xs[i] \le xs[i+1]$

$i+1 > i$

$xs[i] \le xs[i+1]$


across 0 |..| (xs.length - 1) as i
xs[i.item] <= xs[i.item + 1]
end
bool

Version 2: "Approximated" Preconditions

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

B

assert b > 0

assert b > 0

Invariant

b > 0

the global invariant (bal > 0) is not encoded.

try { B }

catch (Exception e) { ... }

fce instanceof BWE

else fc

class BalancedBST {

void insert(int c) {

insert

restore bal. if necessary

// invariant: tree is balanced

}

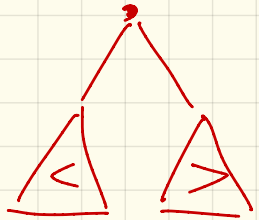
transfer(jim, tom) {

jim.withdraw(a)

tom.deposit(a)

}

possible inv. violation



Version 3: Class Invariant

```
public class AccountV3 {
  public AccountV3(String owner, int balance) throws
     BalanceNegativeException
  {
    if(balance < 0) { /* negated precondition */
      throw new BalanceNegativeException(); }
    else { this.owner = owner; this.balance = balance; }
    assert this.getBalance() > 0 : "Invariant:  positive balance";
  }
  public void withdraw(int amount) throws
     WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
    if(amount < 0) { /* negated precondition */
      throw new WithdrawAmountNegativeException(); }
    else if (balance < amount) { /* negated precondition */
      throw new WithdrawAmountTooLargeException(); }
    else { this.balance = this.balance - amount; }
    assert this.getBalance() > 0 : "Invariant:  positive balance";
  }
```

so balance : 100

this.b = 49

this.b = a - 1

100 50

bool binSearch (x, xs)

$x \in xs \Rightarrow Result == true$

$x \notin xs \Rightarrow Result == false$

$Result \Longleftrightarrow x \in xs$

Version 4: Uncaught Faultly Implementation

```
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"; }
```

Version 5: Postconditions

update(3, "dog")

```
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"; }
```

update(3, 7)

2	3	4	5	6

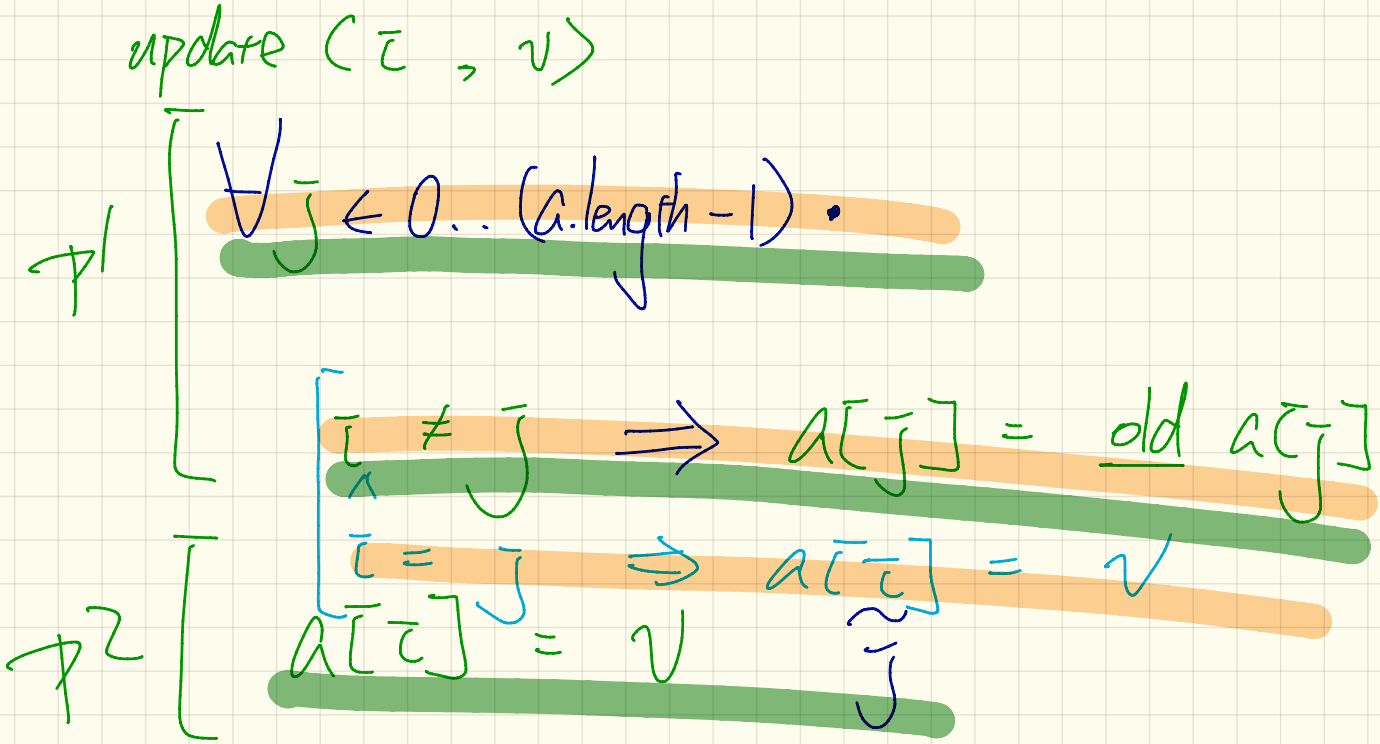
7

Q.

```
class Array {
  int[] a;
  void update(int i, int v)
    precond: 0 ≤ i < a.length
    postcond: a[i] = v
}
```

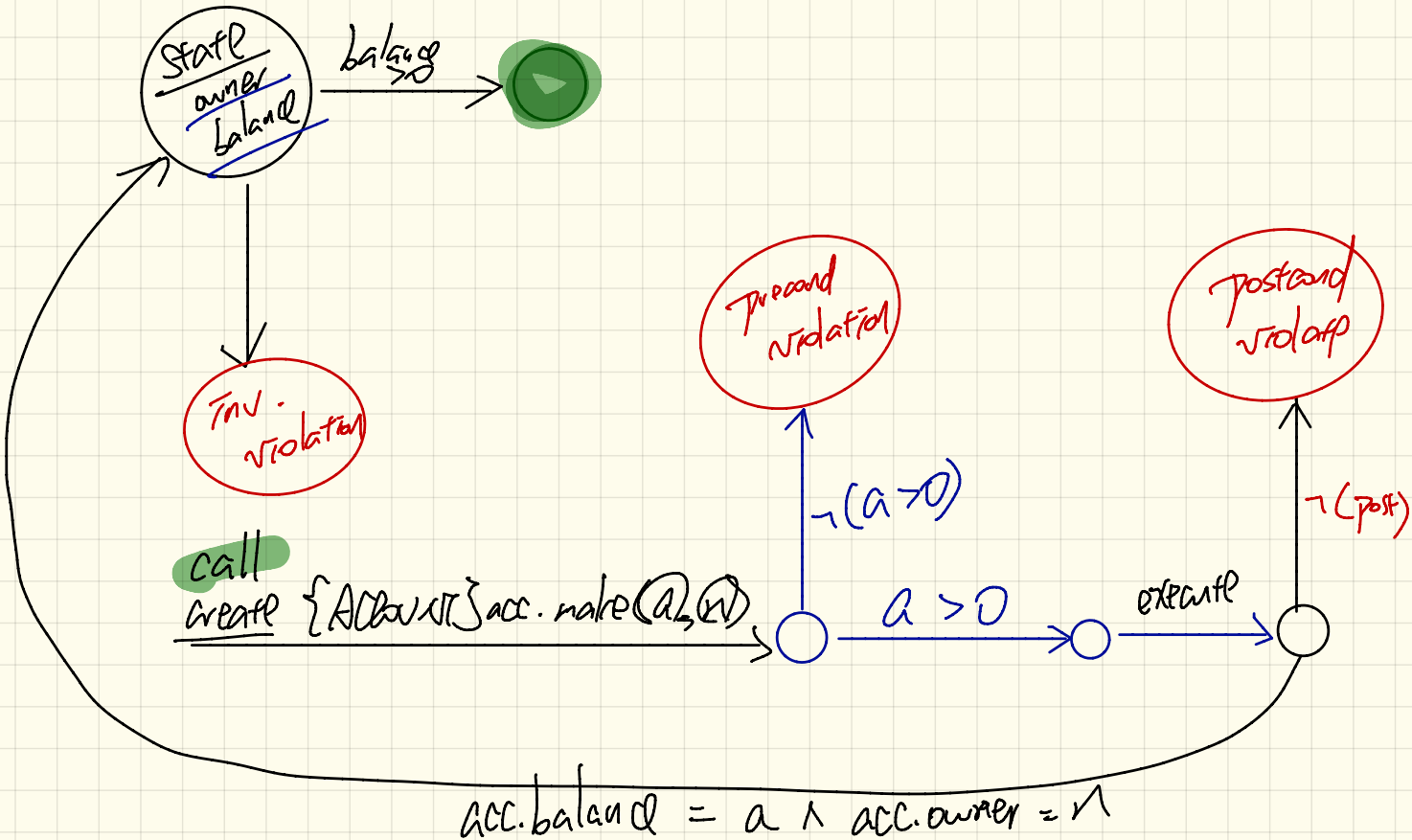
FI.

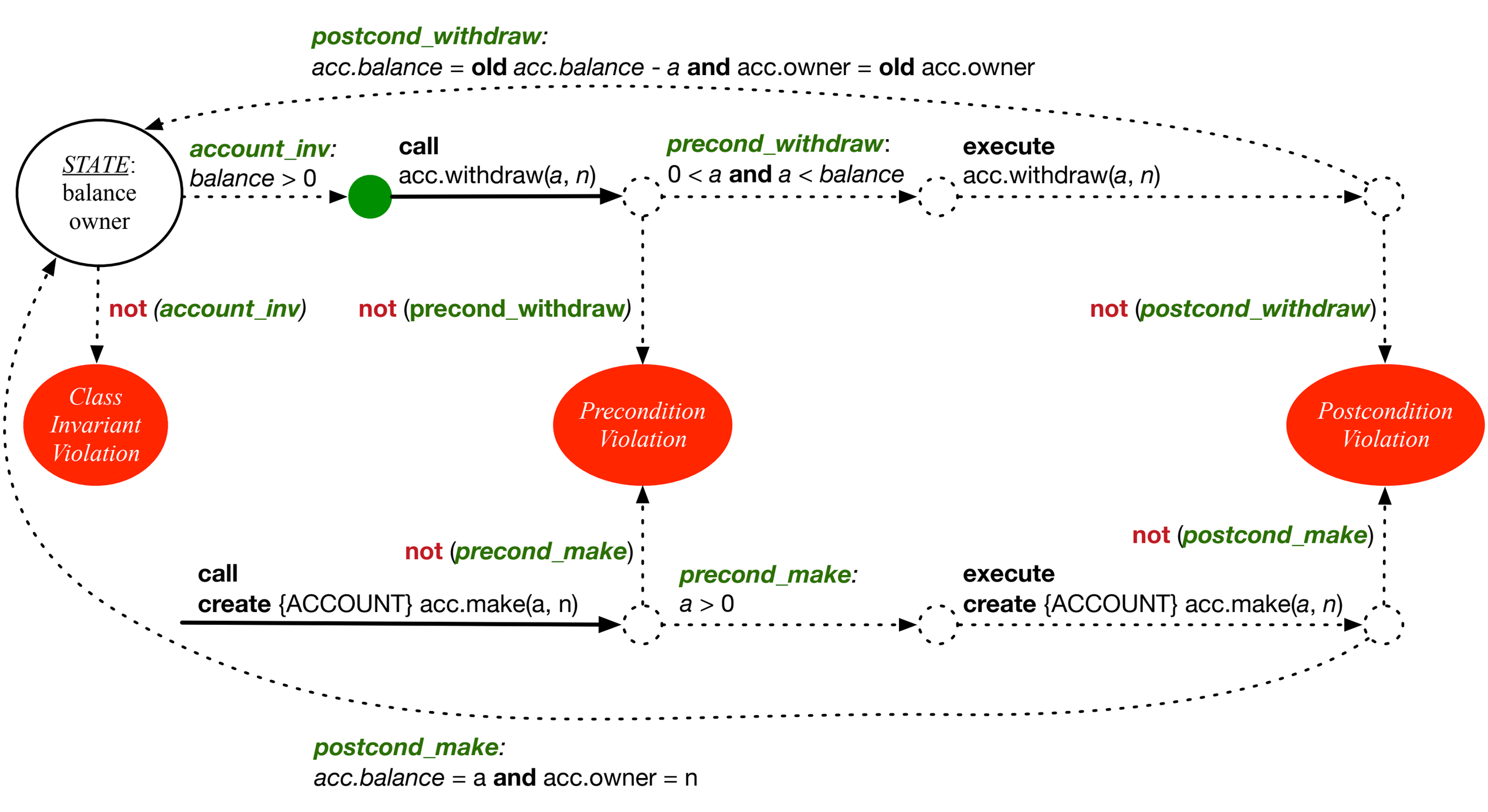
```
a[i] = v;
a[i-1] = v;
```


update (i , v)
p1
∀ j ∈ 0.. (a.length - 1) •
i ≠ j ⇒ a[j] = old a[j]
i = j ⇒ a[i] = v
p2
a[i] = v
i
j

Account Class: State, Contracts, Creation, Updates

```
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
            do
                  balance := balance - amount
            ensure -- postcondition
                  balance_deducted: balance = old balance - amount
            end
invariant -- class invariant
      positive_balance: balance > 0
end
```


State
owner
balance
balance >0
Inv. violation
call
create {Account}acc.make(a,N)
Precond violation
¬(a>0)
a > 0
execute
Postcond violation
¬(post)
acc.balance = a ∧ acc.owner = N


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 (postcond_make)
not (precond_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