'SimpleBank' Model in UML+OCL

Nuno Amálio

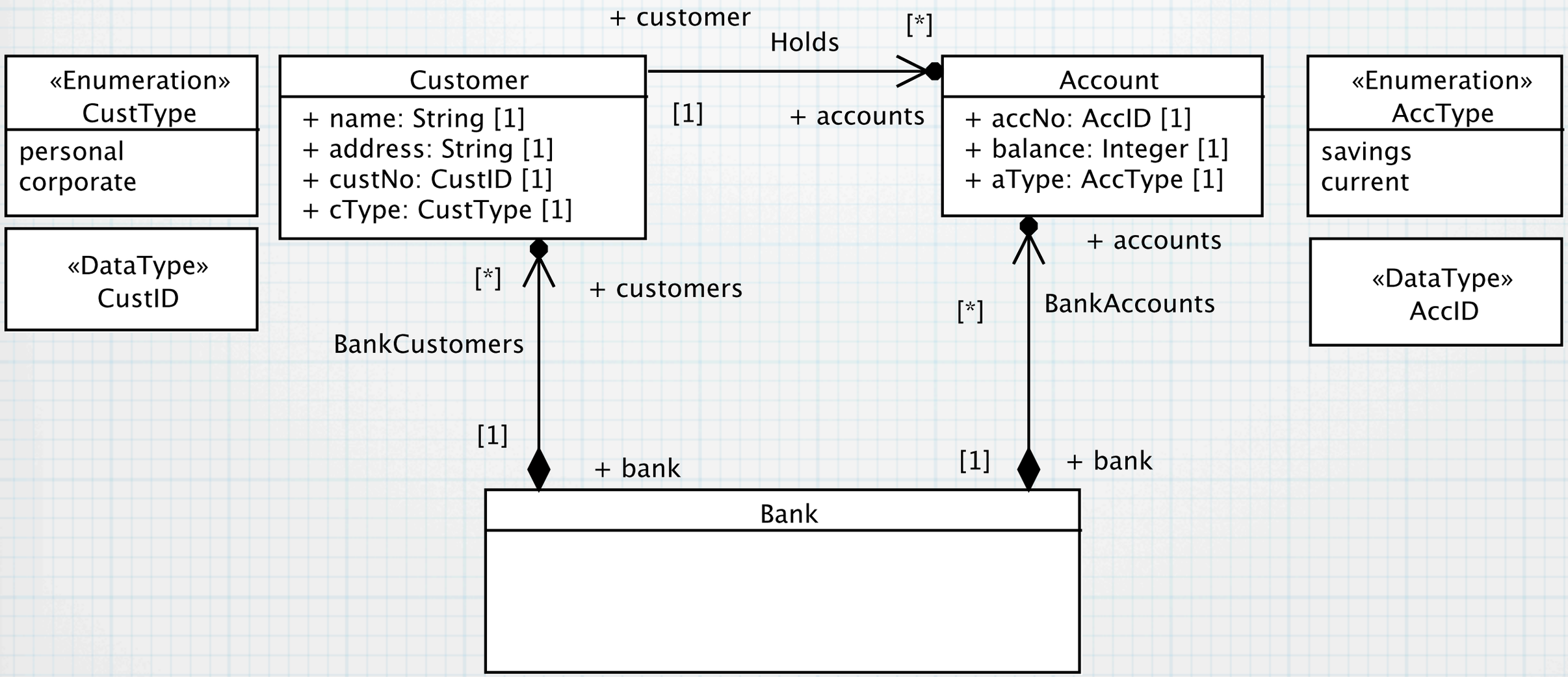
Simple Bank (I)

R1	System shall keep information of customers and their bank accounts. A customer may hold many accounts; an account is held by one customer.
R2	System must record name, address and type (corporate or personal) of customers. Each customer has its own unique customer number.
R3	A bank account shall have a balance and a type (current or savings). Each account has its own unique account number.
R4	Savings accounts cannot have negative balances.
R5	Customers of type corporate cannot hold savings accounts.
R6	To open a savings account, a customer must already hold a current account with the Bank.

Simple Bank (II)

R7	System shall enable the creation of customers records.
R8	System shall allow new bank accounts to be opened.
R9	System shall allow money to be deposited and withdrawn from bank accounts
R10	System shall allow bank accounts to be deleted. An account may be deleted provided its balance is '0'
R11	System shall allow viewing the balance of accounts, all accounts that are in debt, and all accounts of some customer

UML Class Diagram after defining state



Invariant 'SavingsArePositive'

context: Account
inv:
self.accType = AccType::savings **implies self**.balance >= 0

Expressed as a formula akin to propositional logic

Invariant 'SavingsArePositive' (II)

context: Account
inv:
Account.allInstances->
select(aType = AccType::savings **and** balance < 0)->isEmpty()

Expressed as a set-theoretic formula

Invariant 'CorporateHaveNoSavings'

context: Customer
inv:
self.cType = CustType::corporate
implies self.accounts->forAll(a | a.aType <> AccType::savings)

Expressed as predicate-logic formula (quantifiers)

Invariant 'HasCurrentBefSavings'

context: Bank
inv:

```
self.customers->select( c| c.accounts->forAll(aType = AccType::current))
  ->includesAll(self.customers->select(c | c.accounts->forAll(aType = AccType::savings)))
```

Expressed as a set-theoretic formula

The set of customers with savings is a subset of the set of customers with current accounts!

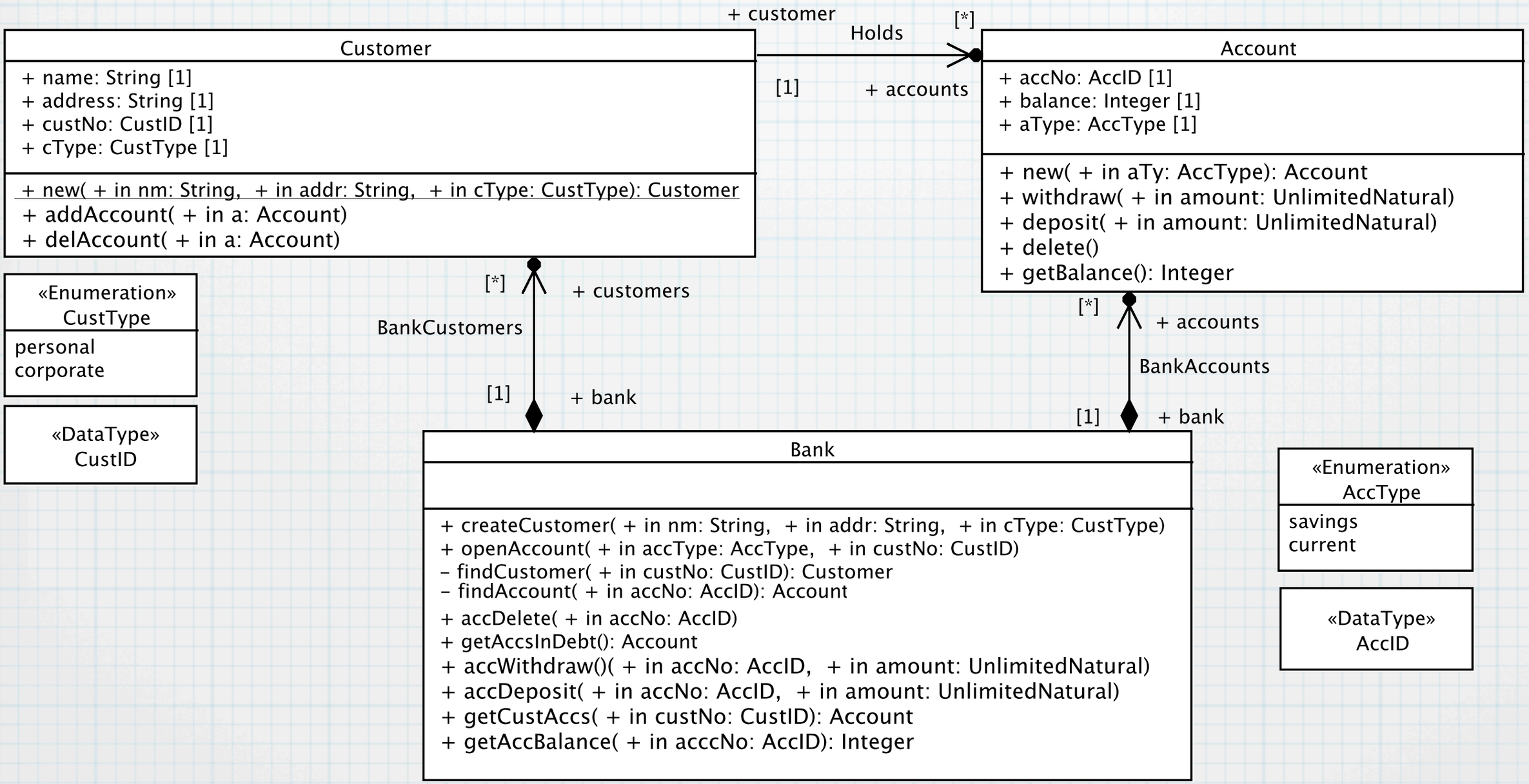
Invariant 'HasCurrentBefSavings' (II)

context: Bank
inv:
self.customers->forAll(c | c.accounts->exists(aType=AccType::savings) **implies**
c.accounts->exists(aType=AccType::current))

Expressed as a predicate-logic formula

For all customers, if the customer has a savings account then he/she must also have a current account!

UML Class Diagram with operations



Customer::new()

context: Customer::new():Customer

post:

```
self.name = nm
and self.cType = cType
and self.address = addr
and self.custNo = CustID.allInstances->any(true)
and result = self
```

Account::new()

context: Account::new():Account
post:

```
self.accNo = AccID.allInstances->any(true)
and self.balance = 0
and self.aType = aTy
and result = self
```

Account::withdraw()

context: Account::withdraw()
post:
self.balance = self.balance@pre() – amount

Account::deposit()

context: Account::deposit()
post:
self.balance = **self**.balance@**pre**() + amount

Account::delete()

context: Account::delete()
pre:
self.balance = 0

Account::getBalance()

context: Account::getBalance()
pre:
result = **self**.balance

Bank::createCustomer()

context: Bank::createCustomer()
post:

```
let c = Customer::New(cType, nm, addr)
in self.customers = customers@pre()->union(Set{c})
```

Bank::openAccount()

context: Bank::openAccount()

post:

```
let c = findCustomer(custNo),
      a = Account::New(accType)
in c.addAccount(a) and self.accounts = self.accounts@pre()->union(Set{a})
```

Bank::findCustomer()

context: Bank::findCustomer():Customer
post:
result = customers->any(c | c.custNo = custNo)

Bank::accWithdraw()

context: Bank::accWithdraw()
post:
let a = findAccount (accNo)
in a.withdraw(amount)

Bank::findAccount()

context: Bank::findAccount():Account
post:
result = **self**.accounts->any(a | a.accNo = accNo)

Bank::accDeposit()

context: Bank::accDeposit()
post:
let a = findAccount (accNo)
in a.deposit(amount)

Bank::accDelete()

context: Bank::accDelete()
post:
let a = findAccount (accNo)
in a.delete()

Bank::accGetBalance()

context: Bank::accGetBalance()
post:
let a = findAccount (accNo)
in result = a.getBalance()

Bank::getCustAccs()

context: Bank::getCustAccs()
post:
let c = findCustomer(custNo)
in result = c.accounts

Bank::getAccsInDebt()

context: Bank::getAccsInDebt()
post:
result = **self**.accounts->select(a | a.balance < 0)