
Accounts

Framework for Business Transaction Processing A Case Study

Framework for Investing

Ward Cunningham: framework for
securities trading

- • portfolio -- collection of instruments
- • instrument -- stock, bond, cash fund
- • transactions

Another Example

Accounting at a store

- • journal -- collection of accounts
- • account -- inventory, accounts payable (vender)
- • transactions

Similarities

Investment

- portfolio
- instrument
- transactions

Store

- journal
- account
- transactions

Transactions are similar, and instrument and account are similar.

Differences

Each journal defines a new kind of account, and all accounts in a journal are the same.

The same kind of instruments appear in every portfolio, and a single portfolio has many kinds of instruments.

Business Transaction Framework

Account

- List of transactions
- Attributes are functions of trans.

Transactions

- A record with a date.
- Paper forms

Accounts

List of transactions
Attributes are functions of trans.

Vender account
Employee account
Inventory account
Bank account

Transactions

A record with a date.
Paper forms

Invoice / purchase order
Paycheck / timecard
Invoice / sales transaction
Withdrawal / deposit / interest

An Inventory Account

An invoice for a purchase increases inventory and increases amount owed to vender.

A sales transaction decreases inventory and increases amount sold to date.

Inventory #384

		on hand	sold	purch
start		30	100	30
sales #6	(4) 26	104	30	
sales #8	(1) 25	105	30	
sales #13	(2) 23	107	30	
invoice #5	(50) 73	107	80	
sales #24	(1) 72	108	80	

Transactions

Records with dates, some of whose fields are multivalued

Invoice

Date: 11/19/93
Due Date: 1/19/94
Company Name: Paul
Invoice Number: 23
Amount Due: 0

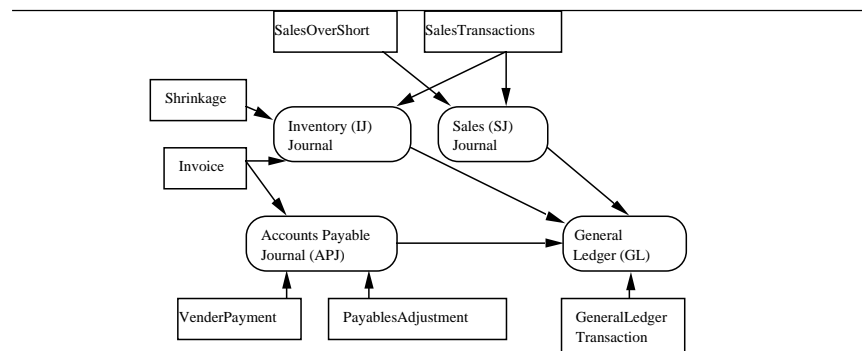
	item	quantity	price
1	lemons	10	15.00
2	oranges	23	12.95

12.95

Add Delete

Accept Cancel

Transactions and Accounts



Rectangles are kind of Transaction.
Ovals are journals, contain Accounts.

Attributes

Each account has a set of *attributes*.

An attribute can be:

- • total of a field taken from the transactions
- • month-to-date, year-to-date, etc
- • function of other attributes, e.g. inventory on hand is total purchased - total sold - shrinkage.

Attributes

Each attribute has a name
(SalesMonthToDate).

An attribute is time-dependent, i.e.
reading an attribute always requires
specifying a time.

First Version of Accounts

Accounts are responsible for

- • computing value of attributes
- • keeping track of transactions

Transactions are responsible for

- • knowing the accounts that they are posted to

First version of Accounts

Account

InventoryAccount

PayableAccount

...

Bond

Cash

...

Transaction

- Invoice

- SalesTransaction

- ...

- Purchase

- InterestPayment

- ...

How to use Accounts V1

White-box framework --- use by subclassing

Determine the kinds of accounts and transactions.

How to use Accounts V1

For each kind of account, make a subclass of Account.

Define a method for for computing each attribute. It must access transactions.

How to use Accounts V1

For each kind of transaction, make a subclass of Transaction.

- Define a method that returns the accounts to which the transaction should be posted.
- Define methods that attributes use to access transaction.

Inventory and Accounts Payable

First a purchase order is sent to a vender. The vender sends back the goods, supposedly with an invoice, though sometimes the invoice comes separately. The receiving department checks that all the goods arrived as stated on the invoice.

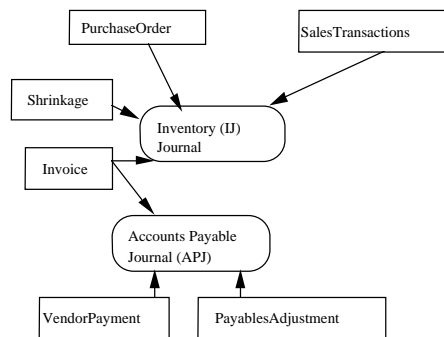
Payment is sent to the vender when it is due. The goods are put on the shelves, where they are sold. When the inventory is low again, another purchase order is written.

Create Accounts and Transactions

Transactions:

- Invoice, PurchaseOrder, SalesTransaction, VendorPayment, Shrinkage, PayablesAdjustment
- Accounts
 - InventoryAccount, VendorAccount

Define the Account to which a Transaction is posted



Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

632

Define fields on transactions

Invoice has:

- • date
- • vendor
- • due date
- • list of (inventory item, claimed quantity, actual quantity, price)
- • discount
- • total

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

633

Define attributes on accounts

Inventory has total received, total sold, total ordered, on-hand

- • define method for each attribute that sends message to transactions
- • for each Transaction posting to an Account, define method to respond to message sent by attribute

Inventory

Inventory has total received, total sold, total ordered, on-hand

- • define way attribute is computed (+, *, mtd, ytd)
 - "received" uses + and sends #received: to the Transactions

Inventory

- for each Transaction posting to an Account, define how the attribute gets a value
 - #received: to a PurchaseOrder = 0
 - #received: to a Shrinkage = 0
 - #received: to a SalesTrasaction = 0

Inventory

- #received: to an Invoice
- Look up inventory number in the Invoice, and return the "actual quantity".
 - (InventoryAccount is argument to #received:, and each InventoryAccount knows its inventory number.)

Eliminate Subclassing Account

Main difference between Accounts is their attributes.

First try:

Attribute is operation on Account.

Capture commonality by decomposing attributes and reusing pieces.

Second try:

Attribute is an object that is a component of an Account.

Build class hierarchy of "attribute components".

Specify each attribute as a composite object.

Attributes as a Strategy

A strategy is an algorithm represented as an object.

'Attribute' computes the value of an attribute of an account at a particular point in time.

Different subclasses of Attribute can compute the value differently.

Strategy Pattern

Making attribute an object means:

- • easy to change from total to MTD
- • can make a class hierarchy of attributes, so it is easy to make a new kind of attribute
- • eliminates tag checking, and makes Account smaller

Designing Attribute Class Hierarchy

Attribute

- TotalAttribute -- sum fields with a given name from all transactions
- YTDAttribute -- sum fields with a given name after start of year
- MTDAttribute -- sum fields with a given name after start of month

Designing Attribute Class Hierarchy

When objects do not come directly from the problem domain, it is often hard to design good abstract classes from first principles.

Usually best to design abstract class by generalizing from concrete subclasses.

More Attributes

Not all attributes are the sum of a field of a set of transactions.

- On-hand in an inventory is (purchased - sold - lost).
- Interest earned is (amount * interest rate).

Attributes

Idea: attribute a *function of time*

- "Sum of a field of transactions posted so far" is one kind of function of time.
- An arithmetic function of "functions of time" is a function of time

Interpreter

Need to define one attribute in terms of other attributes:

- $Average = Total / Count$
- $OnHand = Purchase - Sold - Shrinkage$

Interpreter

- 1) make a class hierarchy that represents nodes in abstract syntax tree (+, -, *, /, attribute)
- 2) define a method *valueAt: aDate* for each class
- 3) define +, -, /, etc functions in common superclass to return PlusNode, DifferenceNode, etc.

valueAt: aDate

An AttributeNode knows an Account and Attribute on that Account, and *valueAt: aDate* will read the Attribute on that Account for that date.

valueAt: aDate for a PlusNode will return the sum of the values for *aDate* for its two descendents.

FunctionOfTime

AccountAttribute

- YTDAttribute
- MTDAttribute
- TotalAttribute

BinaryFunctionOfTime - has two FunctionOfTime components

- PlusNode
- DifferenceNode

ConstantFunctionOfTime

Interpreting Functions of Time

A `FunctionOfTime` responds to the `valueAt: message`.

An `AccountAttribute` computes `valueAt:` by summing a field of a set of transactions.

A `BinaryFunctionOfTime` computes `valueAt:` from the value of its components.

Advantages of Accounts V2?

Defining a new kind of `Account` is easy -- just define a set of attributes.

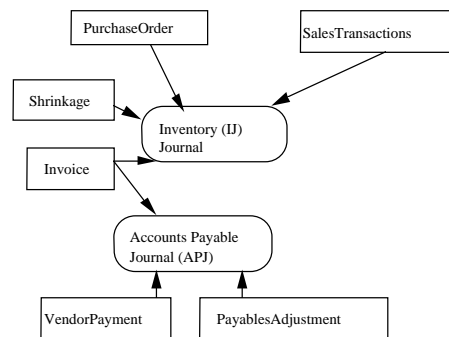
Defining an `Attribute` of an `Account` is easy -
- just define name, operation (almost always +), and operation to perform on `Transaction`.

Disadvantages of Accounts V2

Defining a Transaction is messy.

- • define how to find Accounts to post the Transaction
- • implement all the operations sent by all Attributes of all Accounts to which the Transaction is posted

Another Look at Accounts and Transactions



Multiple inheritance!

All Transactions posted to an Account handle the same operations. Thus, all Transactions in an Account can be thought of as having same superclass. Invoice must be subclass of both InventoryTransaction and PayableTransaction.

Simulating M.I. with Compound Objects

Multiple inheritance can always be simulated by dividing an object into pieces.
Sometimes this improves a design.

For Accounts

Instead of making Invoice be a subclass of InventoryTransaction and PayablesTransaction, make it be a composite transaction with subtransaction that are InventoryTransactions and PayableTransactions.

Subdividing a Transaction

Invoice now gets broken into a set of InventoryTransactions (one for each inventory item on the inventory) and a PayablesTransaction.

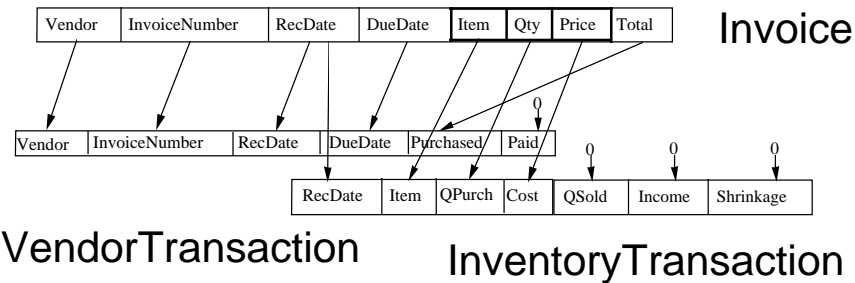
An InventoryAccount now only contains InventoryTransactions.

It is now trivial for the Attribute to get a value from Transaction; just read a field. Fields of Transaction posted to account now have only a single value. All Transactions are of same class and have same field. We might as well give the field the same name as the Attribute that reads it.

Conservation of Complexity

Problems don't go away, they just move. Hard part is now specifying how to break a complex transaction like Invoice into simpler transactions like InventoryTransaction.

Subdividing an Invoice



Item/Qty/Price in Invoice are multivalued fields. Create one VendorTransaction for each value.

Composites

A Composite for T is a subclass of T that combines a group of instances of subclasses of T and lets them act like a single instance of T.

Invoice is a composite Transaction: it combines a group of Transactions and lets them act like a single instance of Transaction.

Composite Accounts

Journal, Portfolio are collections of Accounts. Can they be Accounts?

- what happens when transaction is posted to them?
- what does it mean to query their attributes?

CompositeAccount

What happens when Bob buys 100 shares of IBM stock?

- The transaction "Bob buys 100 shares of IBM stock" is posted to Bob's Portfolio, which in turn posts a transaction to an IBM stock account that is part of the portfolio.

CompositeAccount

What happens when a transaction is posted to a composite account?

- ask the transaction for the name of the component account, and then post the transaction to the component account

CompositeAccount

The value of Bob's portfolio is the sum of the values of the accounts in it.

What does it mean to query an attribute of a composite account?

- query that attribute of the component accounts and return the sum

Other CompositeAccounts?

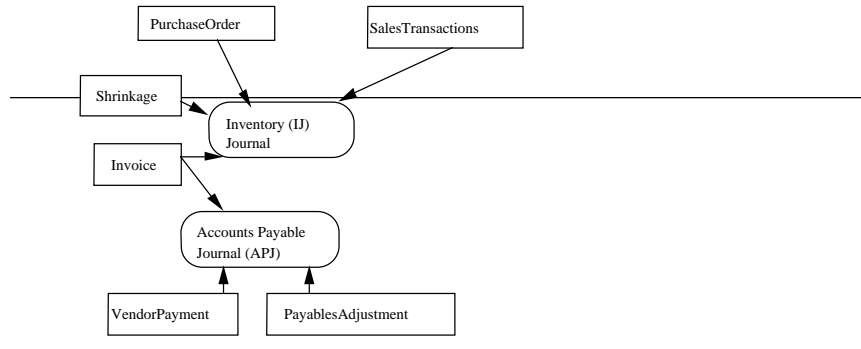
In general, a composite account is an account made up of other accounts.

- • post transaction by posting to components
- • compute attribute by computing attribute of components

Other CompositeAccounts

Can more than one component get a transaction?

Can attribute be computed other than by using all components?

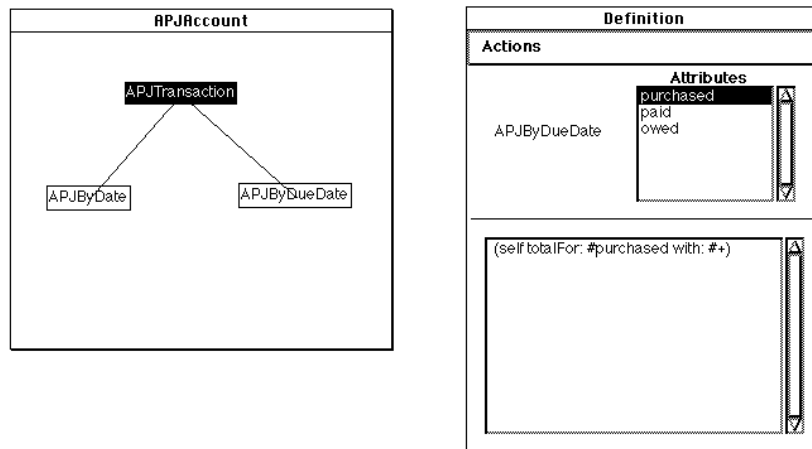


VendorAccount sorts transactions two ways; by received date and by due date.

Solution

VendorAccount is a composite account with two subaccounts, one that sorts by received date, and one by due date. Transactions posted to a VendorAccount are posted to both its components, attributes of a VendorAccount are computed using attributes of one of its components.

Heterogenous Composite Accounts



Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

670

Accounts are used as Prototypes

Prototype - object used to represent a class of objects

Instead of saying "make a new object of that class", say "make a copy of that object".

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

671

Accounts are Used as Prototypes

Any account can be turned into a "template", and when you add an account you are asked for a template to be copied.

Three prototypes, SimpleAccount, HeterogenousCompositeAccount, and HomogenousCompositeAccount.

Accounts Framework, V3

Account

SimpleAccount -- contains a set of transactions posted to it

CompositeAccount

- HomogenousCompositeAccount -
- HeterogenousCompositeAccount

Accounts Framework, V3

Transactions

SimpleTransactions -- no multivalued
fields, posted to Simpleaccount

ComplexTransactions

Accounts Framework, V3

FunctionOfTime

- AccountAttribute (only of
SimpleAccounts and
HomogenousCAs)
- BinaryFunctionOfTime
- ...

How Accounts is a Framework

Abstract classes - Account, Transaction, Attribute

Way objects interact:

- Transaction -> Account - Transaction can post itself to set of Accounts
- Account -> Transaction - Account sorts its Transactions

(continued)

- Account -> Attribute - Account retrieves the value of an attribute at a particular time by delegating to the Attribute object
- Attribute -> Transaction - Attribute knows how to fetch value from Transaction

How to use Accounts

Define Transactions and Accounts.

- define the Accounts to which a Transaction is posted

Define fields on Transactions.

Define Attributes on Accounts.

Define how

HeterogenousCompositeAccount
divides Transactions posted to it

Define Attributes on Accounts.

- define way AccountAttribute is computed (+, *, mtd, ytd)
- define FunctionOfTime as attribute of HeterogenousCompositeAccount

Changing White-box to Black-box

Journal	Account
Portfolio	• SimpleAccount
Account	• (parameterized by Attributes)
InventoryAcc.	• CompositeAccount (parameterized by its components and their names)
VendorAcc.	
StockAccount	
BondAccount	

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

680

White-box to Black-box

Old: Make new Account by defining attributes as functions in new Account class.

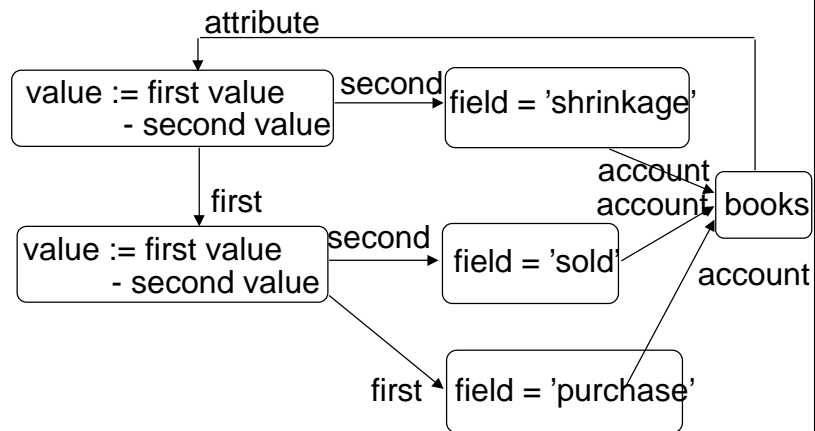
New: Make new Account by setting its list of Attribute objects.

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

681

Instance Diagram

OnHand = Purchase - Sold - Shrinkage



Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

682

Examples

Good examples are crucial:

- for finding framework
- for teaching framework

Examples define scope of the framework.

Concrete examples
framework



abstract

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

683

Finding Examples

Start with similar examples.

- should illustrate features that framework must support
- provides a base for generalization

Eventually examples should cover wide range.

- minimize number of examples

Finding Examples

Look for examples that will break the framework.

- • perhaps they will define limits of framework
- • perhaps they will lead to generalization
- • avoid lots of special cases

What Examples for Accounts?

We've worked most on:

- store
- investment
- payroll

We have thought about:

- personal finance
- budgeting

What Examples for Accounts?

We should consider:

- • banking
- • insurance
- • manufacturing inventory

Scale up!