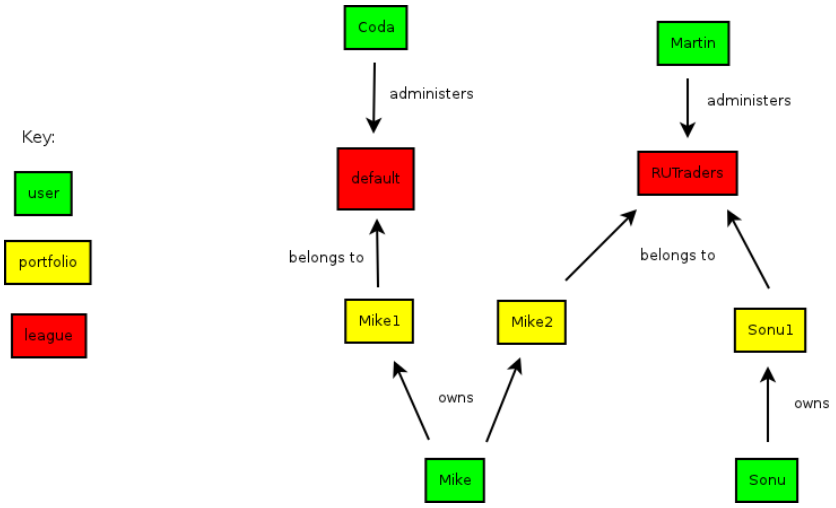
Users, Portfolios, and Leagues

Basic Definitions

- "User" -- A human player of PitFail. A user may manage more than one portfolio.
- "Portfolio", aka "Team" aka "Company" -- A made-up PitFail entity that *owns* and *trades*. Many times in this document it may be mentioned that a "portfolio" places an order. The reason for this phrasing is that the order is associated with a portfolio, not with a user. The primary traders in PitFail are portfolios. A portfolio may be owned by more than one user.
- "League" -- a collection of portfolios competing against each other. A league is managed by a User, but participated in by Portfolios. Hence a single user may have portfolios that belong to different leagues.

An example might help to illustrate what is going on here:



In this example, Mike and Sonu are users. Mike has two portfolios, named Mike1 and Mike2; Sonu has 1 portfolio, named Sonu1. Mike1 belongs to a league named "default"; Mike2 and Sonu1 belong to a league named "RUTraders".

Coda and Martin are users that administer the "default" and "RUTraders" leagues. Coda and Martin might have portfolios of their own, but this is not relevant to the business of administering leagues.

The reasons for the existence of each of these concepts is:

- "User" -- This provides a way for an actual human user to log in the the site, to have an experience that is tied to them.
- "Portfolio" -- These actually do the trading. A Portfolio is the one actually credited with owning assets and being responsible for the payment of liabilities, *not* the user.

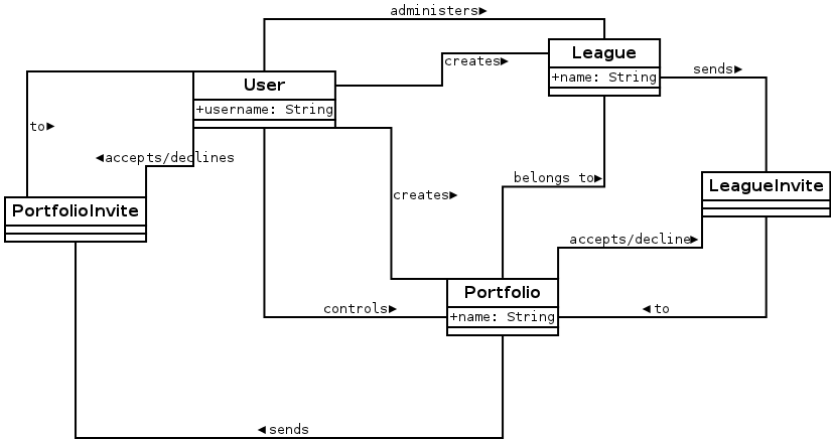
- "League" -- The purpose of a league is to represent "competition" between portfolios. Hence rankings are done within a league, and "rules" are set within a league. Trading, however, happens globally, among all leagues.

In the report we will often say that "a portfolio does this" and "a portfolio does that"; the action is being initiated by a human, but we model it as if the portfolio is the doer of an action: a portfolio buys a stock, a portfolio sells a stock. If we want to refer to a real human being we will use the word "player".

The User-Portfolio-League domain model

The basic concepts and relationships for the idle system are:

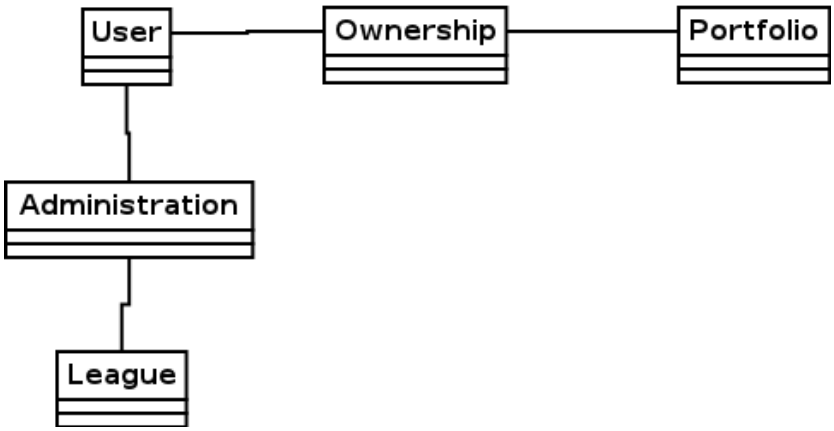
Adding some of the creation/joining operations, this becomes:



Note a few potentially surprising things about this model:

- PortfolioInvites are sent to Users, and LeagueInvites are sent to Portfolios. This is because it is a User who will control a portfolio, and a Portfolio that will join a league (users do not join leagues).
- Even though, in reality, a human user initiates the action of "sending" an invite, it is shown in the diagram as originating from a Portfolio or a League, because that is how we interpret it; invites come from the concepts that can be joined.

In the actual code, some of the "many-to-many" relationships acquired an extra class (the association class). Such as:

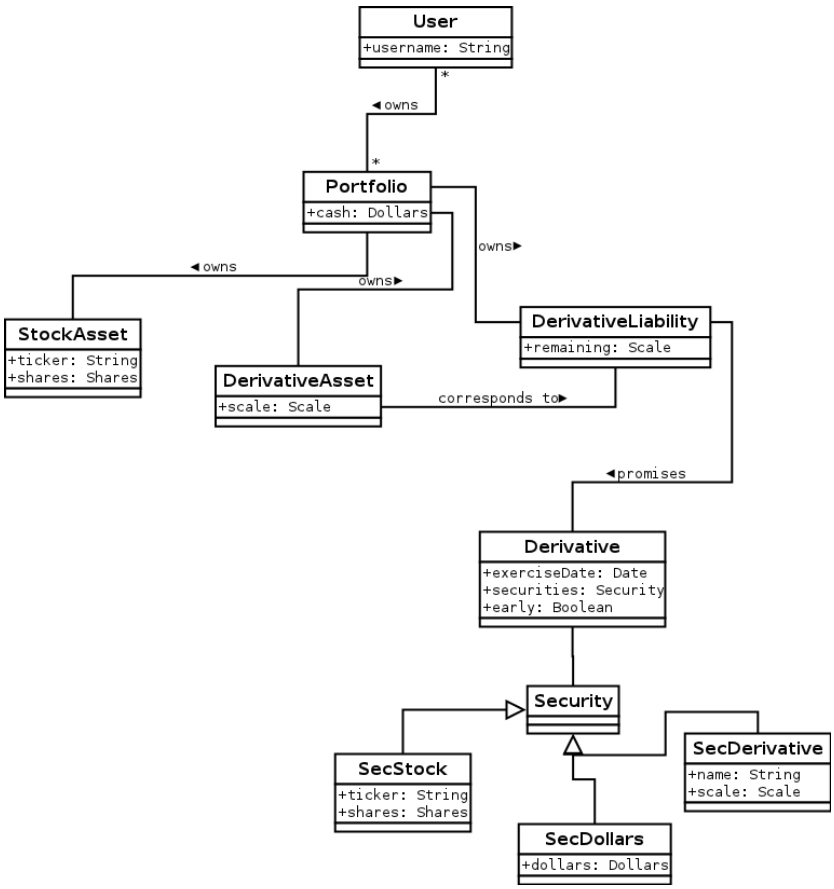


But this is a detail of the implementation; no meaningful attributes are stored with Ownership and Administration.

Assets and Liabilities

This part describes only the *ownership* aspect of assets and liabilities. The trading and exercising aspects will be described later.

The diagram below shows only the part of the domain model that relate to the ownership of assets and liabilities:



First, a user does not own assets. Their portfolio owns the assets. A user may control 0 or more portfolios, and each portfolio may be controlled by 0 or more users (a portfolio with 0 users must be allowed in the case that all its users wish to leave the game but they still have liabilities to other players).

There are two kinds of assets: StockAssets and DerivativeAssets, and one kind of liability: a DerivativeLiability.

How StockAssets work

A stock asset is simply a number of shares of a particular stock. So for example, 30 shares of MSFT is a stock asset.

How Derivative Assets/Liabilities work

A derivative, in PitFail, is a promise to exchange a list of assets on or before a specified date. There are 3 parts to this contract:

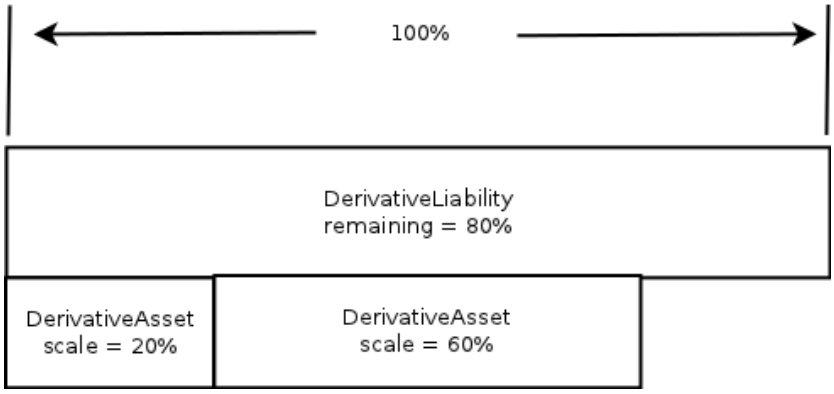
1. The *Derivative* is the statement of the contract; that is, it is the list of assets to be exchanged, the date on which it is to occur, and whether the

contract may be exercised early (See for example [American vs European Options]_). The exact nature of how the contract is specified is described in the section on Derivatives.

2. The *DerivativeLiability* is the statement by one portfolio that they will offer up the assets specified in the Derivative.

3. The *DerivativeAsset* is a promise to a portfolio that they will be able to collect the assets promised in the Derivative.

Each DerivativeAsset corresponds to exactly 1 DerivativeLiability, and each DerivativeLiability corresponds to 1 or more DerivativeAssets. Each DerivativeAsset has a property called `scale` which is the portion of the liability this asset has a claim on. A DerivativeLiability has an attribute `remaining` which is the fraction of the contract that has *not* been exercised:



Every time a DerivativeAsset is exercised, it is deleted, and the `remaining` of the corresponding DerivativeLiability is reduced by the `scale` of the DerivativeAsset. It is an invariant of the system that the sum of of the scales of all DerivativeAssets for a particular DerivativeLiability must equal the `remaining`.

Derivatives

The parts to a derivative contract are:

1. A list of securities to be traded.

2. A date on which this is to occur.

3. Whether it may be exercised early.

(2) and (3) are just a `DateTime` and a `Boolean` respectively; (1) is more complicated.

The list of securities is represented as a list, where each element may be one of:

1. A "stock" security, `SecStock`, which holds a ticker symbol and a number of shares.

2. A "dollars" security, `SecDollar`, which holds a dollar amount.

3. A "derivative" security, `SecDerivative`, which holds a named liability and a scale (see the section on Scaling Derivatives). (At the moment there is no way within the PitFail UI to create a `SecDerivative`. However, since the theoretical concepts behind it are complete, we describe it anyway).

If any of the quantities are negative (eg negative shares, negative dollars, negative scale), that means that the securities are supposed to move from the buyer to the seller.

Exercising Derivatives

When a derivative is exercised, the goal is to move the securities from their source (seller or buyer's portfolio) to their destination (buyer or seller's portfolio). When this is possible, the procedure is easy; the only complications that arise are when this is not possible.

Moving Dollars

Say $100 dollars needs to move from A to B. If A has $100, $100 is deducted from A's cash, and added to B's cash.

If A does not have $100, as much as possible is deducted and added to B's cash. this should begin a process of margin call and forced liquidation, but PitFail does not support this feature at this time.

Moving Stocks

Say 100 shares of MSFT need to be moved from A to B. If A has 100 shares of MSFT, they are deducted from A's portfolio and added to B's.

If A does not have 100 shares of MSFT, the following steps are taken:

1. First, A (under the control of the system, not the human player) attempts to buy 100 shares of MSFT at 15% above the last traded price. This is similar to a limit order in that the trade will execute at the ask price if the ask price is less than 1.15*(last trade price). This attempt to buy may be partially or completely executed (if there are shares available), or not at all.

2. If, after attempting to buy the remaining shares, A *still* does not thave 100 shares MSFT, pays the remaining debt to B in cash, at 1.15*(last trade price)*(shares unaccounted for).

3. If A does not have enough shares *or* enough cash, this should generate a margin call and A's assets should be liquidated, but PitFail does not support this feature.

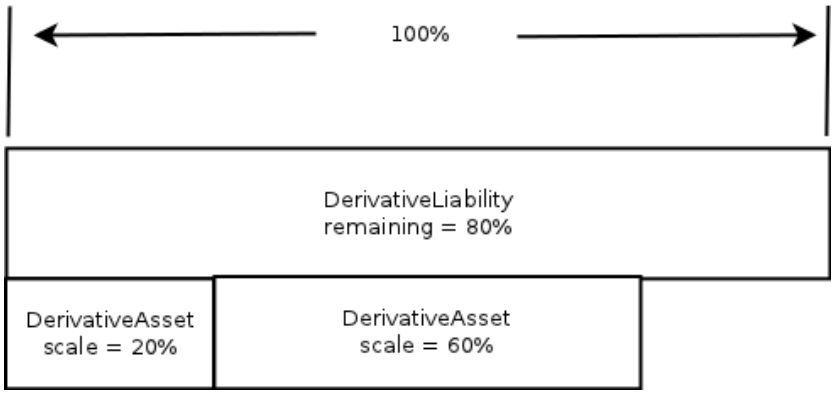
This procedure for moving stocks differs significantly from the old procedure (as of demo #1), because in the old version it was always possible to buy an unlimited amount of a stock. When this became no longer possible, it was necessary to design a system that would respect the limited volume available but still be largely automatic; since we do not expect PitFail players want to be bothered by an online game to resolve the issue. Hence the 15% premium -- high enough to give a user an incentive to actually own the stocks promised, but not so high as to make it a disaster if they do not.

Moving Derivatives

This feature was removed from the most recent version of PitFail because the UI still does not support creating a derivative that refers to another derivative (making the support in the backend moot). In the old version, the way this worked was that, if A owned the specified amount of the specified derivative, it would be moved. If not, a *new* derivative would be created with terms identical to the desired ones, for which A would hold the liability and B the asset.

Scaling Derivatives

Many aspects of PitFail require that derivatives be scaled. That is, given one derivative, create a new one with identical terms, but "smaller" or "larger":



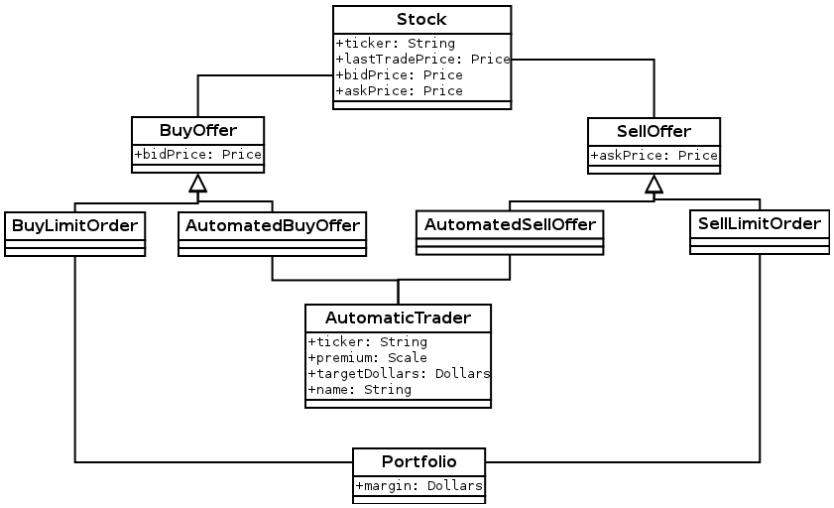
Scaling is done by scaling each security promised:

1. For SecDollar, scale the dollar amount
2. For SecStock, scale the share amount
3. For SecDerivative, scale the scale amount

and leaving the date and early exercise the same.

Trading Stocks

The diagram below represents the "idle state" of the system with respect to stock trading:



When the system is is idle, no trades are taking place; all that exist are orders that have yet to be fulfilled.

PitFail allows only two kinds of orders to sit idly. These are

1. Limit orders
2. Autamated (synthetic) trading orders.

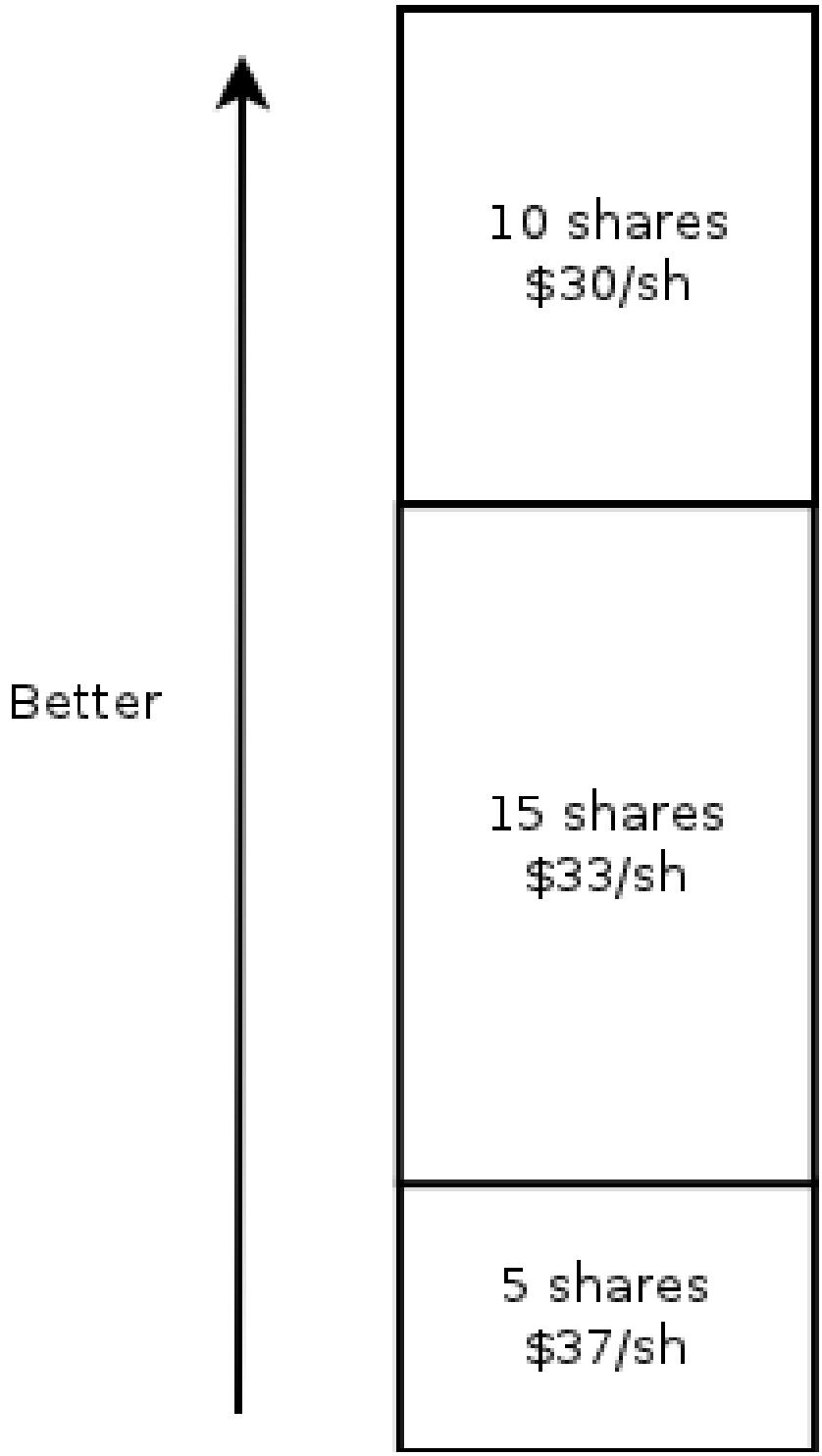
Market orders do not exist when the system is idle because market orders are executed at the offering price as soon as they are created. PitFail does not provide explicit support for stop orders, but it would be easy for a user to create one using the javascript automated trading API (and, when a Stop is triggered, it becomes a market order[Stop Order]_, and so will be executed immediately).

All orders in the idle state have two important properties: the available number of shares, and the limit price. This will allow PitFail to form automatic matches, as described later.

An invariant of the system is that when the order system is Idle, there are no orders that can be matched with one another.

When a new order comes in

When a new order comes in, it has a desired number of shares, and it may or may not have a limit price. First, all existing orders for the same stock are collected, and sorted by desirability (ie, best price to worst price):



The incoming order is matched up against the best orders possible (that are below its limit price, if any). Those orders are then completely or partially executed:

In this example, 10 shares will be purchased at 30/sh, and 2 shares at 33/sh.

You will notice that the orders already *in* the pool pay a price in not being able to negotiate -- since the buyer is willing to pay 34/sh, they would, if they could, increase their limit to 34/sh to take advantage. However, by having orders in the pool that are *not* negotiated, there is a benefit in liquidity; hence traders who place orders unexecuted into the pool will change a liquidity premium in the trade (which is why there is a spread between the bid and ask price for a stock as offered by the same trader [Market Makers]_).

If the newly placed order is not fully executed, and the trader specified a limit, it will become part of the pool of unexecuted orders.

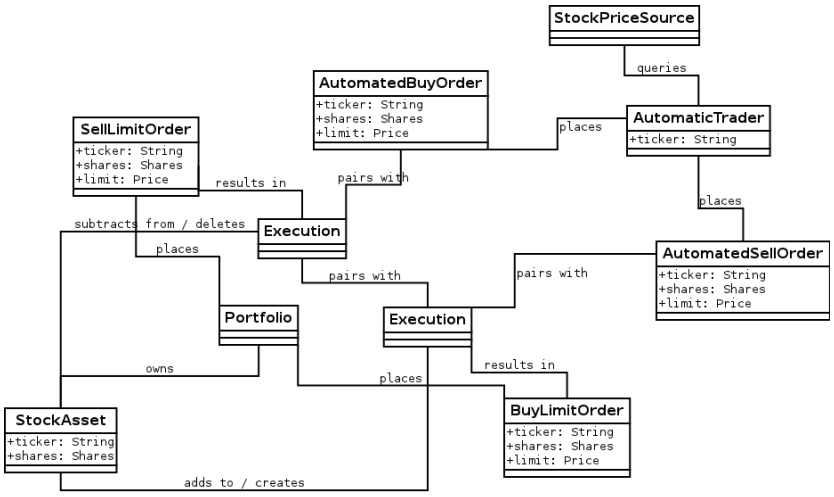
Margin

In order to ensure the smooth execution of orders, when a user places an order that is not executed immediately, they must set aside margin so that the order can be executed later. For a buy order the user sets aside cash that will be used to buy the shares when the order is executed, and for a sell order the user sets aside the shares that will be sold.

If the order is cancelled or not fully used the margin will be returned.

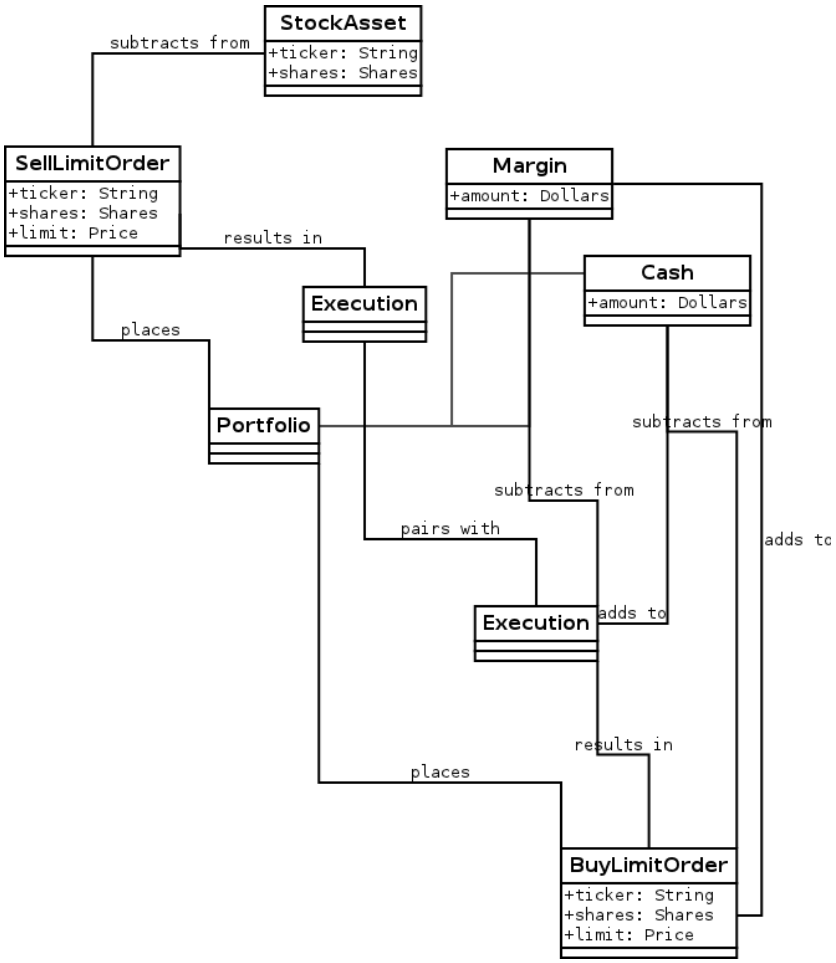
Domain model for trading

The model below does not correspond 1-1 to actual software classes because our architecture is not entirely object-oriented. For example, there is no class called Execution; execution of orders is procedural.



The association of AutomaticTrader with StockPriceSource is meant to convey that the automatic traders use real-world bid and ask prices to set their bid and ask prices.

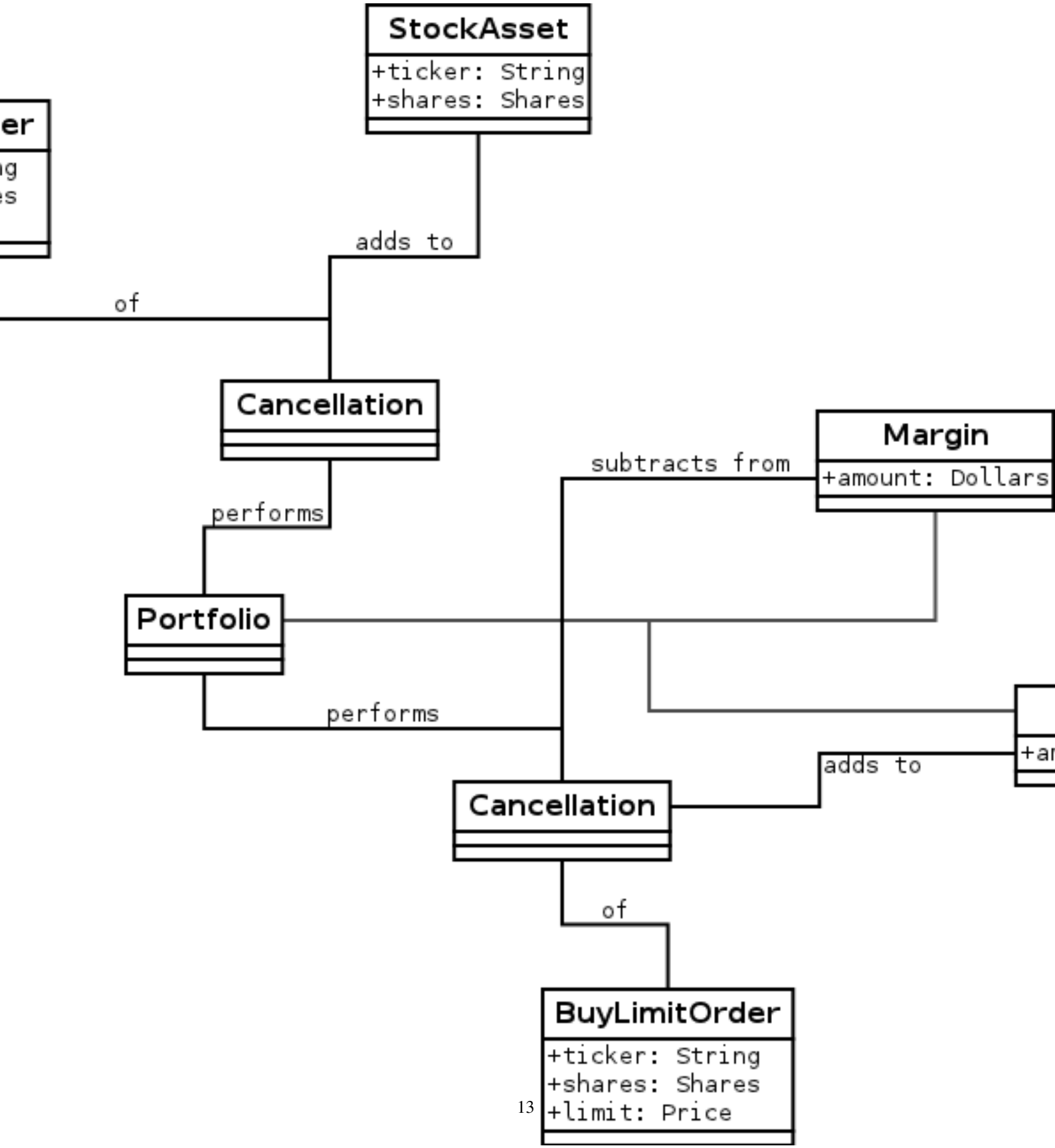
Because there is too much to fit on one diagram, here is the part of the domain model that deals with cash and margin:



(In the code, there is no object called Cash, rather it is an attribute of Portfloio; but it is helpful to show it as such for the domain model).

The reason that the execution of a BuyLimitOrder "adds to" Cash is that all the necessary cash has already been set aside in Margin; the cash that is being added is the leftover margin.

When an order is cancelled (by its owner), all that must happen is that the margin is restored:

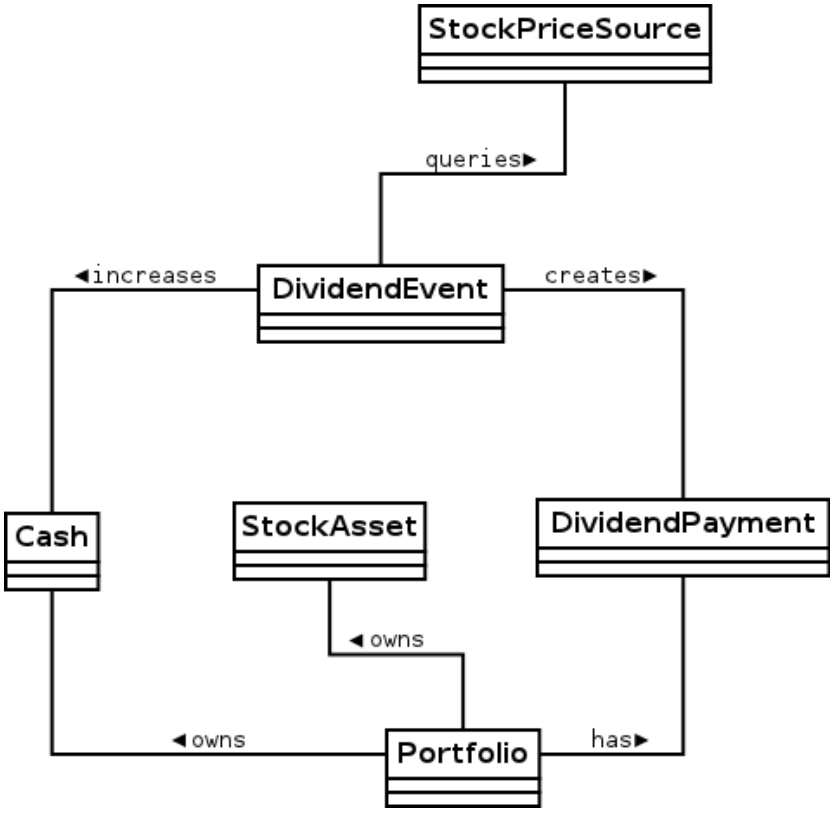

StockAsset
+ticker: String
+shares: Shares
adds to
of
Cancellation
Margin
subtracts from
+amount: Dollars
performs
Portfolio
performs
adds to
Cancellation
of
BuyLimitOrder
+ticker: String
+shares: Shares
+limit: Price

Dividends

It is very important for PitFail to keep track of dividends paid by stocks, for two reasons:

1. It would be unrealistic in a particularly unsettling way: stocks that will never pay dividends have no value; why are we trading them?
2. Because PitFail players will own stocks that pay dividends, and every time a dividend is paid the stock price drops abruptly, players would not appreciate having the price drop if they do not receive a dividend in return.

Periodically, PitFail queries Yahoo Finance to see if stocks owned by the players have paid dividends. If they have, the system will pay dividends to the player, in what is represented here (though not in the code) as a `DividendEvent`:



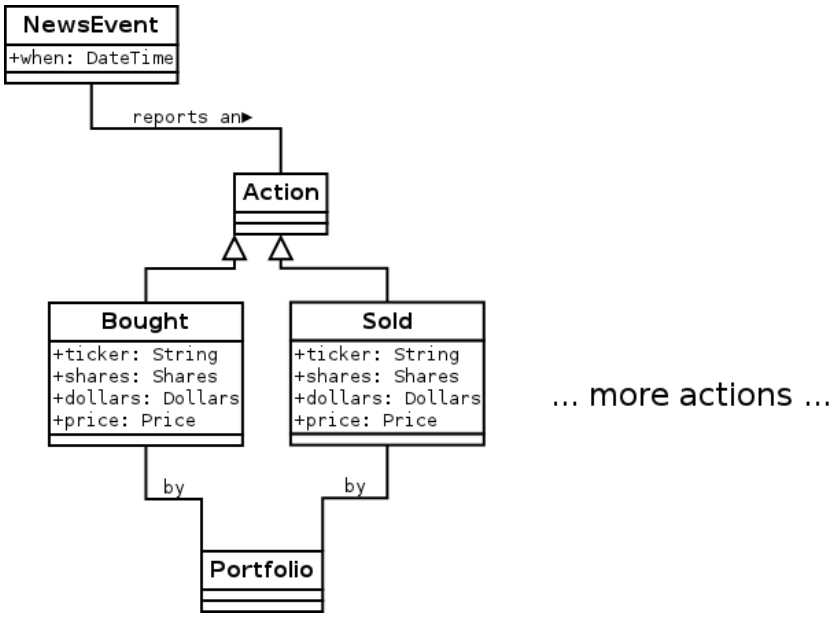
The `DividendPayment` object is created only to allow the user to view the history of their dividend payments.

News

The purpose of "news" is to show PitFail players to see what other PitFail players have been doing. Importantly, News is not part of actual trading; this is just for seeing what's going on.

This means that a single news event has associations with a lot of other concepts, but not in a way that affects the rest of the program: it's just point out, for example, which derivative was traded when reporting that a derivative was traded.

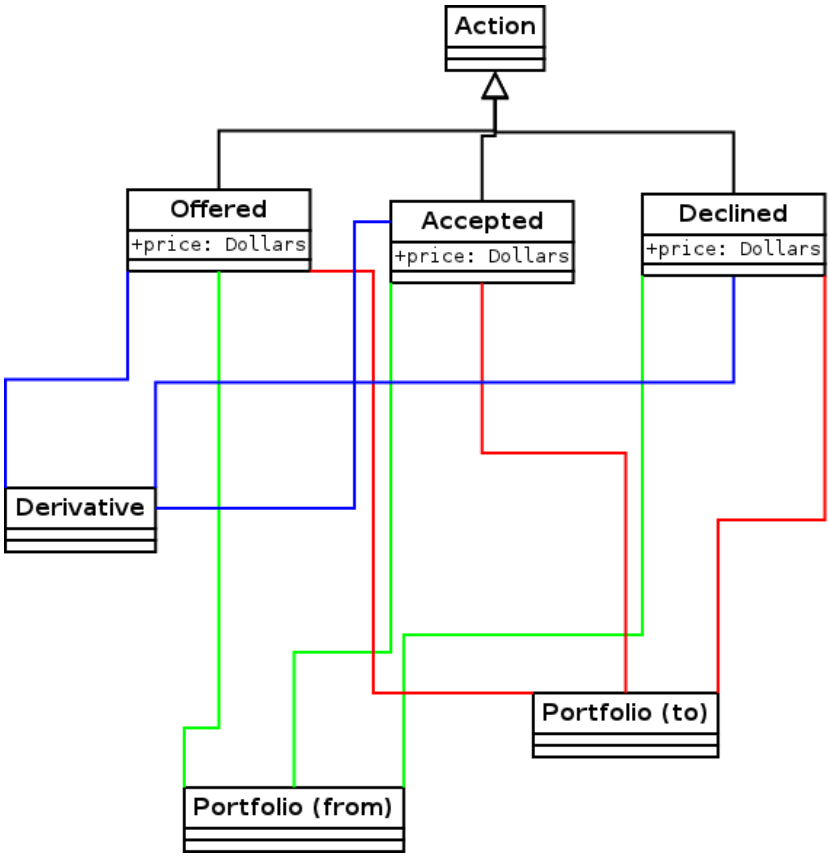
The basic model for News is:



only two actions are shown here; there are a lot so they are split up across multiple diagrams.

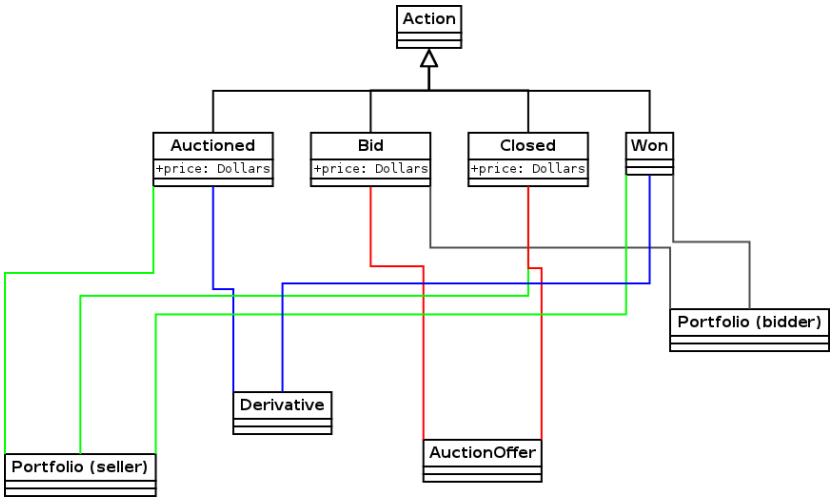
Buying and selling stocks, as shown above, refer to the Portfolio who "did" the action, and the information about what was bought or sold. This only applies to orders that are executed (either immediately or later). Orders that are delayed will generate another kind of an event.

Derivative Trading has the following kinds of events:



`from` and `to` are shown as separate concepts even though they are instances of the same class, because they play a different role in these events: one is the portfolio making the offer, the other is the portfolio receiving, and possible accepting, the offer.

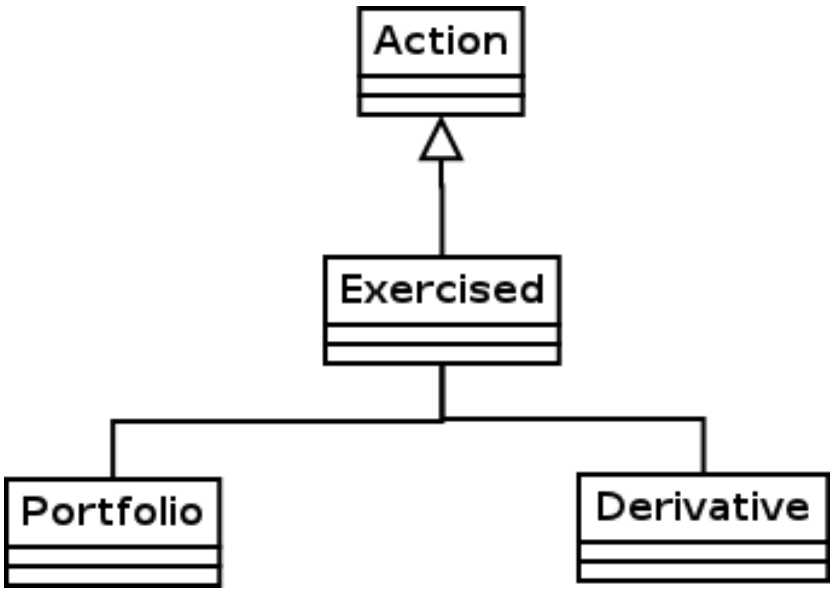
For Auctions we have:



There are other associations which are not shown, that relate to voting. These are

described in the section on voting.

Placing orders that get delayed are described by:



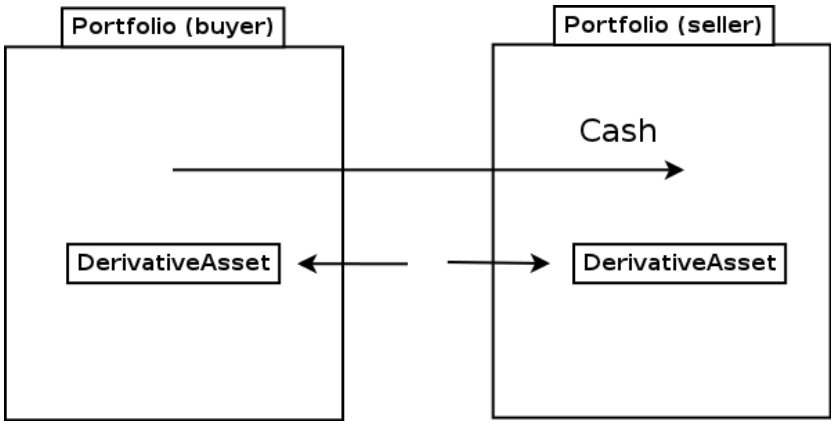
Where the associated portfolio is the one who performed the buy or sell.

There is one more event for exercising derivatives:

Where the associated portfolio is the one who did the exercising.

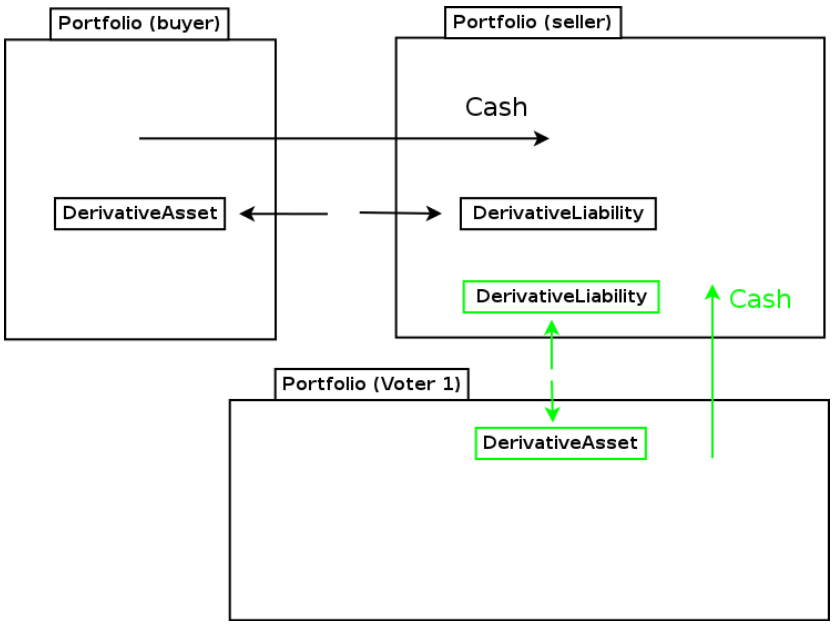
Voting

When players enter into a contract involving a derivative, the following assets are moved:



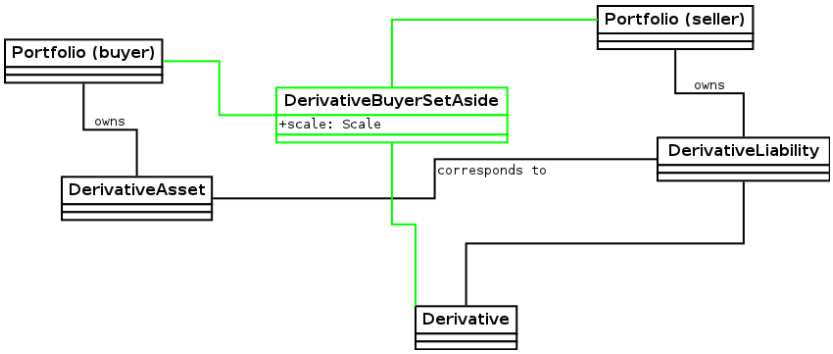
If owning the asset (being in the buyer side of the contract) pays off more than the cash payed, the buyer is happy. If owning the liability (being in the seller side of the contract) is not bad enough to negate the cash received, the seller got a good deal. These are not necessarily mutually exclusive.

Now, say a third player, the Voter, looks at his news feed and thinks that the buyer got a good deal (and maybe the seller too, but that is not relevant yet). The Voter would be happy with an arrangement like the following:



where the derivative in green resembles the derivative in black, and the cash in green resembles the cash in black. (As in, if it was a good deal for him, it's a good deal for me too. Not necessarily true, but it could be true sometimes).

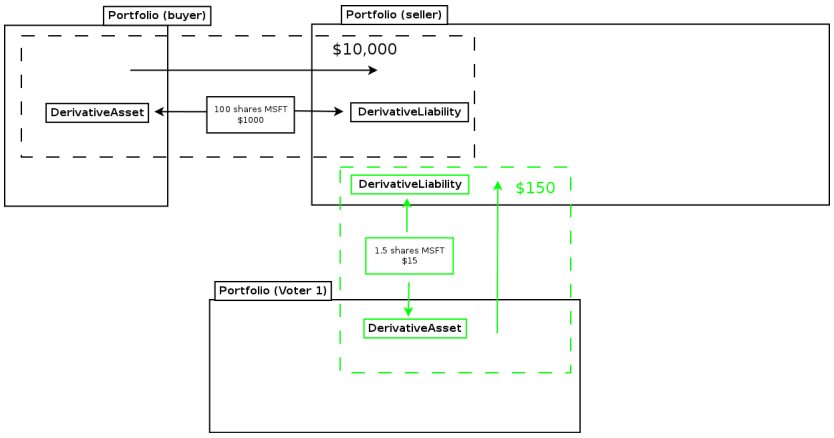
When two portfolios enter a derivative, an object is created called `DerivativeBuyerSetAside` (there is a nearly identical process for sellers, which is discussed next):



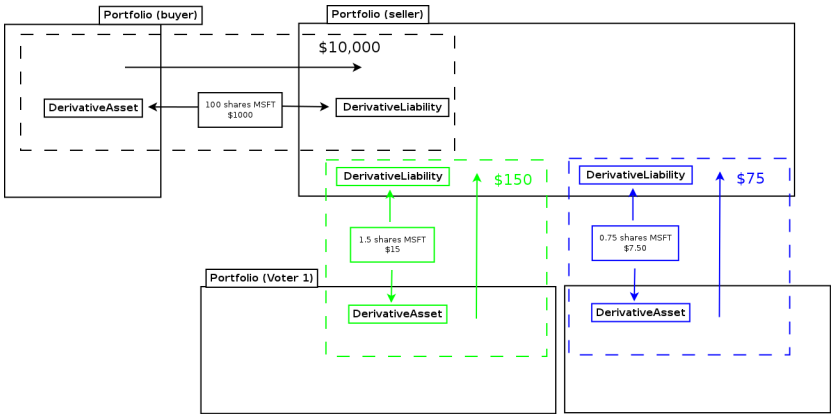
(remember, the Derivative holds the terms of the contract, and the DerivativeAsset and DerivativeLiability show who owns which end).

The DerivativeBuyerSetAside holds one attribute, which is the "amount" left to be voted on. For the precise meaning of this scale, see the section on Scaling Derivatives.

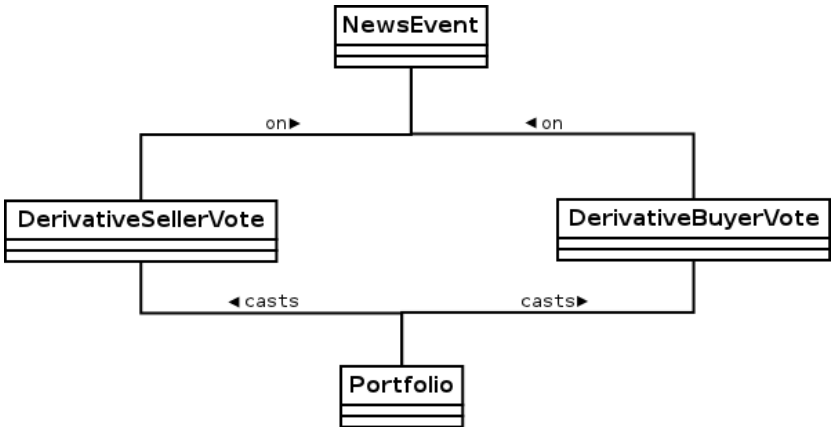
The `scale` remaining starts out at 3%. When the first voter votes in favor of the buyer, they enter into a contract with the seller that is identical to the original derivative, but scaled to 1.5% (= 3%/2). He also pays the seller 1.5% of what the original buyer paid:



The `scale` remaining is then cut by half to 1.5%.
Now if another player votes, they will realize 0.75% of the original trade:



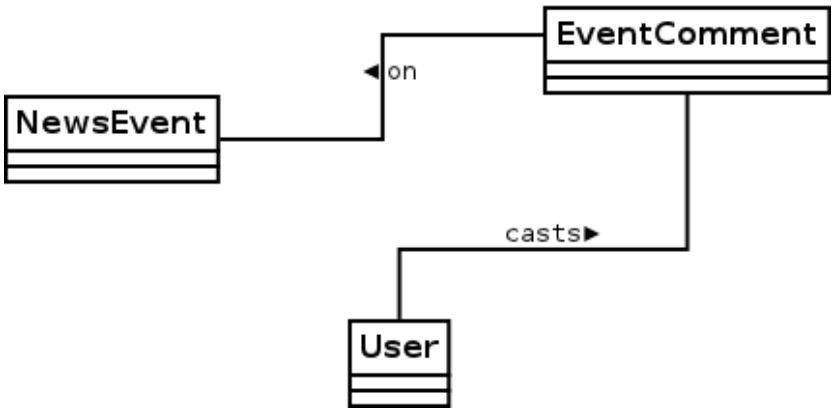
Votes are recorded and associated with the origanal NewsEvent, so that a score of buyer-votes and seller votes can be calculated:



Comments

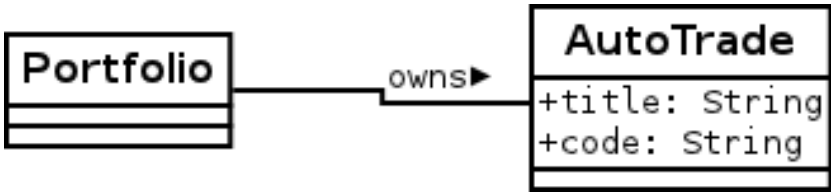
Compared to voting, comments are refreshingly simple.

Users, not portfolios, cast comments. A comment is associated with a news event:

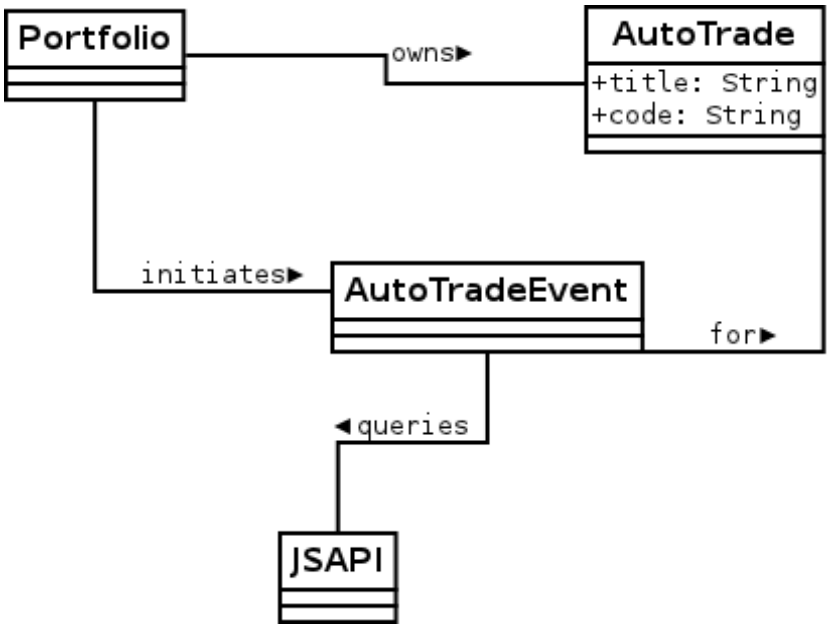


Auto Trades

While the system is idle, an auto-trade is represented as:



When a player runs an AutoTrade, we have what we conceptually (though not in the code) call an AutoTradeEvent:



The `JSAPI` is a set of JavaScript functions and corresponding server-side handlers that allow the Auto Trade to actually perform actions.