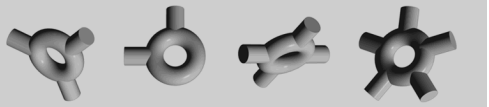
elements



Security Prices Package

TARMS Inc.

September 07, 2000

Typeset in LaTeX.

Contents

List of Figures

List of Tables

Package Description

Securities represent standardized interest-bearing contracts. Since the contracts are standardized, it is possible to both quote the prices for the securities — usually for some standardized amount — and receive a market feed of securities prices. The Security Prices package models the specification, quotation and derivation of these prices.

Since a security's price essentially reflects the price paid now for an amount of money in the future, securities prices and interest rates are closely linked. Yield curves may be used to derive implied securities prices. Conversely, security prices may be used to imply interest rates. The latter conversion is of interest in developing long-dated yield curves; bonds of 10 or more years duration are not uncommon, and can be used to extrapolate yield curves beyond simple quoted interest rates.

1 Use Cases

1.1 Price Per 100 Face Value

The *price per hundred face value* (or *PP100FV* for short) quotation method gives the amount paid, in the currency that the security is denominated in, for 100 currency units of that security.

As an example, for bonds with a face value of DEM 1,000,000 and a PP100FV of 101.23, the amount paid for the bonds is DEM 1,012,300.

1.2 Yield to Maturity

The yield to maturity represents the value of a bond at a constant rate of interest.[1][1]

Suppose a bond pays coupons and principal of $\{p_1, \ldots, p_n\}$ (with p_n usually being the return of principal) at dates $\{d_1, \ldots, d_n\}$. Suppose the yield to maturity is y, quoted as an annualized, compound rate. Then the dirty price paid for the bond is

$$\Sigma_{i=1}^{n} p_i \times (1+y)^{-t_i}$$

where t_i is the term in years between the trade date and d_i, calculated according to the date basis convention of the security.

As an example, suppose a bond with a face value of USD 1,000,000 matures on 26-Jan-2001 with a semi-annual coupon of 5%, paid on 26-Jan-2000 and 26-Jul-2000. If the trade date is 7-Jan-2000, the date basis is 30/360 and the quoted yield is 5.66%, then the dirty price is:

$$\begin{array}{rcl} 25000 \times 1.0566^{-19/360} + & & \\ 25000 \times 1.0566^{-199/360} + & = & 992863.93 \\ 1000000 \times 1.0566^{-379/360} & & \end{array}$$

[1] Most bond valuation methods value the bond against a yield curve, giving different interest rates and discount factors for each payment. A yield to maturity represents a single, composite interest rate corresponding to a quoted price.

1.3 Fraction Quotation

Price per hundred face value prices are sometimes quoted in fractional, rather than decimal form. Fractions are usually expressed in 1/32 of a unit, although 1/64 is sometimes used. The price 101 8/32, therefore, is equivalent to 101.25.

A short form drops the denominator and expresses the fraction as a whole part, a dash and the numerator of the fraction. In this form, 101 8/32 would be expressed as 101-08. If the fraction is quoted as n/64 then it is represented as a fraction of /32 +/1 1/64. So 101 3/64 would be represented as 101-01+ (ie. 101 + 1/32 + 1/64) and 101 1/64 would be 101-01- (ie. 101 + 1/32 - 1/64).

1.4 Yield

Rather than being quoted in direct terms, a bond price may be quoted as a *yield*. The yield is an interest rate roughly equivalent to the interest paid on the bond.

1.5 Simple Yield to Maturity

A simple calculation of yield that ignores the effects of the the time value of money.[1]

If a bond has a coupon rate of r, a face value (redemption amount) of p, a number of years to maturity of t and a simple yield to maturity of y then the clean price, c, of the bond is given by:

$$c = \frac{r + p/t}{y + 1/t}$$

For example, JPY 1,000,000 of a 7 year bond paying a coupon of 10% and with a simple yield to maturity of 11% has a clean price of

$$\frac{100000 + 1000000/7}{0.11 + 1/7} = 960451.98$$

Derivation:

$$y = \frac{r + \frac{p-c}{t}}{c/p}$$

1.6 Current Yield

A simple calculation of yield, ignoring the effects of the return of principal and the time value of money.[1]

If the coupon rate of a bond is r, the face value is p and the current yield is y then the clean price, c, for a bond is given by:

$$c = p \times \frac{r}{y}$$

For example, the clean price for bonds with DEM 1,000,000 face value, a coupon rate of 7% and a current yield of 8.56% is

$$1000000 \times \frac{0.07}{0.0856} = 817757.01$$

Derivation:

$$y = \frac{r}{c/p}$$

1.7 Clean/Dirty Price

Security prices may be either *clean* or *dirty*. When a security is purchased part-way through a coupon period, the seller of the security expects to be paid the accrued interest for the part of the coupon period during which the seller held the security.[1]

A price quoted including this accrual is called the dirty price. A price from which the accrued interest has been subtracted is the clean price.

For example, assume a bond paying a 6% semi-annual coupon. The last coupon date was 19-Aug-2002. The next coupon date will be 19-Feb-2003. The trade date is 16-Sep-2002, with accrued interest calculated on a 30/Actual date basis. The dirty price is 103.45 (PP100FV).

The accrued interest from 19-Aug-2002 to 16-Sep-2002 is $100 \times 0.06 \times \frac{27}{365} = 0.44$. The clean price is, therefore, $103.45 - 0.44 = 103.01$.

2 Interfaces

2.1 ImpliedSecurityInterestRateDerivationSpecifier

Just as a yield curve and a security can be used to derive a security's price (see ImpliedSecurityPriceConstructor §3.2), so can a security's price be used to derive an implied interest rate for the maturity date. Unlike the standard yield to maturity

calculations, this implied interest rate uses a yield curve to accurately discount the security's coupons, giving an accurate interest rate for the maturity date.

These implied interest rates can be used to perform *coupon strips* on a bond, allowing bond prices to be used while building yield curves.

2.1.1 Relationships

	Class	Description	Notes
⇑	ImpliedRateDerivationSpecifier		
↓	ImpliedSecurityInterestRateDerivationSpecifierModel §4.3		
↔	ImpliedSecurityInterestRateConstructorService §5.1	deriver	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.1.2 Operations

RateFunctionSpecifier yieldCurve() yieldCurve

Returns the yield curve used to discount the security cashflows. This rate specifier must have yieldCurve().specifier().type() = "Interest Rate" and no from- or to-date parameters set. The rate specifier may, or may not, include a derivation method.

RateFunctionSpecifier securityPrice() securityPrice

Returns the security price. This rate specifier must have securityPrice().specifier().type() = "Security Price" The rate specifier may, or may not, include a derivation method.

OrderedCollection<RateFunctionSpecifier> sources() sources

Returns the source rates. Returns an ordered collection built from the result of the yieldCurve() and securityPrice() operations.

RateConstructor constructor() constructor

Returns the rate constructor. Returns an instance of ImpliedSecurityInterestRateConstructor §3.1.

2.2 ImpliedSecurityPriceDerivationSpecifier

An implied security price is built from a yield curve. The yield curve is used to

discount the cashflows associated with the security price to provide a net present value (NPV). This NPV, normalized to 100 units, forms a price.

2.2.1 Relationships

	Class	Description	Notes
⇑	ImpliedRateDerivationSpecifier		
↓	ImpliedSecurityPriceDerivationSpecifierModel §4.4		

⇑:Inherits ↓:Realized by

2.2.2 Operations

RateConstructor constructor() constructor

The rate constructor. Returns an instance of ImpliedSecurityPriceConstructor §3.2.

RateFunctionSpecifier yieldCurve() yieldCurve

Returns the yield curve used to discount the security cashflows. This rate specifier must have yieldCurve().specifier().type() = “Interest Rate” and no from- or to-date parameters set. The rate specifier may, or may not, include a derivation method.

OrderedCollection<RateFunctionSpecifier> sources() sources

The source rates. Returns an ordered collection built from the result of the yieldCurve() operation.

2.3 RelativeSecurityPriceSpecifier

A relative security price is an implied security price built by taking the price for a different security and adding a margin to that security.

As an example, the price for a 8% US T-Bond, maturing on 21-Mar-2001 may be set equal to the price for an 8% US T-Bond, maturing on 21-Jan-2001, with a margin of -0-10. If the price of the base bond was 103-03, then the price for the relative bond would be 102-25.

A rate component is constructed by getting the base rate component and adding any margin specified for the rate component in the associated margin list. A “default” margin can be applied to components with no special margining requirements.

Chains of relative prices can be built by using a relative price as the base price.

2.3.1 Relationships

	Class	Description	Notes
⇑	ImpliedRateDerivationSpecifier		
↑	SecurityPrice §2.6		
↓	RelativeSecurityPriceSpecifier-Model §4.6		
↔	SecurityPrice §2.6	base price 1..1	→
↔	SecurityPriceQuote §2.9	margins 0..n	→
↔	RelativeSecurityPriceConstructorService §5.3	deriver 0..n	

⇑:Inherits ↑:Realizes ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.3.2 Operations

RateConstructor constructor() constructor

Returns an instance of RelativeSecurityPriceConstructor §3.3 with the sources and margins of this specifier.

RateFunctionSpecifier base() base

The base rate. Returns a rate specifier that gives the base security price to use when building this rate.

Collection<RateQuote> margins() margins

Returns a collection of components which give the margins to add to each component of the base rate. One margin may be distinguished by the "default" label.

OrderedCollection<RateFunctionSpecifier> sources() sources

The source rates. Returns a collection containing the result of the base() operation.

2.4 SecurityBenchmarkSequence

Individual securities have a single, fixed maturity date. When using securities to calculate implied rates, it is often convenient to use a sequence of securities, with the closest security in the sequence to a particular date acting as the representative of the sequence. As the current date moves forward, the next security is chosen.

Using benchmark sequences aids in the maintenance of static data, since only the sequence needs to be updated over time, rather than the yield curves, etc. that use the sequence.

2.4.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
↓	SecurityBenchmarkSequence-Model §4.8		
↓	SecurityBechmarkReferenceData-Model §4.7		
↔	SecurityBechmarkReferenceData-Model §4.7	model 0..n	
↔	ImpliedSecurityInterestRateDeriva-tionSpecifierModel §4.3	benchmark 0..n	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.4.2 Operations

OrderedCollection<Security> sequence() sequence

The benchmark securities. Returns the sequence of securities, ordered by maturity date, that this benchmark sequence represents.

Security securityFor(Date date) securityFor

date: Date The date to benchmark against.

The closest security in the sequence to a given date. Returns the security from the sequence() operation that has a maturity date closest to the argument date. If two securities are equidistant, returns the later security.

Currency currency() currency

The currency of the sequence. Returns the common currency for the securities that make up the sequence.

Security securityFor(Period period) securityFor

period: Period The period to benchmark against.

The closest security in the sequence to a given date. Return the result of this.securityFor(date) where date is computed by adding the supplied period to the conversion date.

2.5 SecurityMarginQuotationMethod

Security price margins represent additions to or subtractions from a basic security price. The margins can be used to represent margins added to reflect extra risk, profit margins or internal trading arrangements.

2.5.1 Relationships

	Class	Description	Notes
⇑	QuotationMethod		
↓	PricePer100FaceValueMarginModel §4.5		

⇑:Inherits ↓:Realized by

2.5.2 Operations

Boolean isMargin() isMargin

Is this a marginal quotation method? Return true.

Boolean isCanonical() isCanonical

Is this quotation method in canonical form? The canonical form is a price per 100 face value margin.

String type() type

The type of instrument that this quotation method is for. Returns "Security Price".

Number addMargin(SecurityPriceQuotationMethod baseQuote, Number base, Number margin) addMargin

baseQuote: SecurityPriceQuotationMethod The quotation method used for the base price.
base: Number The base price.
margin: Number The margin to add.

Add a margin to a securities price. To add a margin, the form of the base price and the margin must match. The returned rate uses the quotation method of the base rate.

2.6 SecurityPrice

The basic model for a security price. Since price per hundred face value quotation methods give linear prices, these quotation methods are used to derive mid rates and calculate buy and sell amounts.

2.6.1 Relationships

	Class	Description	Notes
⇑	PointRate		
↓	BasicSecurityPriceModel §4.1		
↓	RelativeSecurityPriceSpecifier §2.3		
↔	RelativeSecurityPriceSpecifier §2.3	base price 0..n	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.6.2 Operations

RateQuote mid() mid

The mid component. Let b be the bid component and a the ask component. Let b' and a' be these components, expressed using the PricePer100FaceValueModel §4.11 and the same clean/dirty convention as the bid component.

Returns $\frac{b'+a'}{2}$ converted to the bid component quotation method.

Instrument buy(Instrument quantity) buy

quantity: Instrument The quantity to convert.

Buy one quantity of an instrument by paying another quantity of the instrument.

If this rate is mine, the quantity represents a security and the quantity is greater than zero, then use the bid rate. A change in any one of these characteristics flips from bid to ask. Another change flips back from ask to bid. Convert the component to a clean, PricePer100FaceValue, form: r.

If the instrument represents a quantity of a security, with face value p, then return an amount of the currency that the security is denominated in, with a date equal to the date on quantity and an amount equal to $\frac{p}{100} \times r$.

If the instrument represents a quantity, c, of the currency that the security of this rate is denominated in, then return an amount of the security with a date equal to the date on quantity and a face value equal to $\frac{c}{r} \times 100$.

Instrument sell(Instrument quantity) sell

quantity: Instrument The quantity to convert.

Sell one quantity of an instrument in exchange for another quantity of an instrument.

If this rate is mine, the quantity represents a security and the quantity is greater than zero, then use the ask rate. A change in any one of these characteristics flips from ask to bid. Another change flips back from bid to ask. Convert the component to a clean, PricePer100FaceValue, form: r.

If the instrument represents a quantity of a security, with face value p, then return an amount of the currency that the security is denominated in, with a date equal to the date on quantity and an amount equal to $\frac{p}{100} \times r$.

If the instrument represents a quantity, c, of the currency that the security of this rate is denominated in, then return an amount of the security with a date equal to the date on quantity and a face value equal to $\frac{c}{r} \times 100$.

2.7 SecurityPricePiece

A rate piece specialized towards the handling of security prices.

2.7.1 Relationships

	Class	Description	Notes
⇑	RatePiece		
↓	SecurityPricePieceModel §4.9		

⇑:Inherits ↓:Realized by

2.8 SecurityPriceQuotationMethod

Security prices can be quoted as either a price per hundred face value or as various forms of yield. In addition, security prices may be 'clean' or 'dirty'.

A security price is said to be quoted 'clean' if it excludes accrued interest. It is called 'dirty' if it includes accrued interest.

2.8.1 Relationships

	Class	Description	Notes
⇑	QuotationMethod		
↓	SecurityPriceQuotationMethod-Model §4.10		

⇑:Inherits ↓:Realized by

2.8.2 Operations

String type() type

The type of rate that this quotation method is for. Returns “Security Price”.

Boolean isCanonical(RateFunctionSpecifier specifier) isCanonical
specifier: RateFunctionSpecifier The specifier to test against.

Is this rate in canonical form? Returns true if this quotation method is a price per hundred face value in clean form.

Boolean isMargin() isMargin

Is this rate in margin form? Returns false.

CleanDirtyEnum cleanOrDirty() cleanOrDirty

Is this securities price a clean or dirty price? The price is ’clean’ if it excludes accrued interest. It is dirty otherwise.

ExOrCumEnum exOrCum() exOrCum

Is this price quoted ex or cum coupon? The price is cum-coupon if the purchaser of the security will receive the next coupon. It is ex-coupon if the purchaser does not receive the next coupon. Typically each security will have a standard ex-coupon period, where a purchaser will not receive the coupon if the security is traded within the ex-coupon period before the next coupon date.

«Static Method» QuotationMethod canonical() canonical

Canonical quotation method. Returns an instance of PricePer100FaceValueModel.

Number cleanPrice(Number r, SecurityPriceSpecifier specifier) cleanPrice
r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Convert a dirty price into a clean price.

If this quotation method is already in clean form, then return r. Otherwise, convert r into the clean equivalent price and return that price.

A RateQuotationException is returned if the conversion cannot be performed.

Number dirtyPrice(Number r, SecurityPriceSpecifier specifier) dirtyPrice
r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Convert a clean price into a dirty price. If this quotation method is already in dirty form, then return r. Otherwise, convert r into the dirty equivalent price and return that price.

A RateQuotationException is returned if the conversion cannot be performed.

2.9 SecurityPriceQuote

A rate quote specialized to handle security prices.

2.9.1 Relationships

	Class	Description	Notes
⇑	RateQuote		
↓	SecurityPriceQuoteModel §4.15		
↔	RelativeSecurityPriceSpecifier §2.3	margins 0..n	
↔	RelativeSecurityPriceSpecifier-Model §4.6	margins 0..n	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.10 SecurityPriceSpecifier

A security price specifies the value of a specific securities contract in terms of the currency that the security is denominated in. The contract supplies details as to the currency, date of maturity, etc.

2.10.1 Relationships

	Class	Description	Notes
⇑	RateFunctionSpecifier		
↓	SecurityPriceSpecifierModel §4.16		
↔	ImpliedSecurityPriceConstructorService §5.2	result	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.10.2 Operations

Collection<FormalRateParameter> formalParameters() formalParameters

The possible parameters for this rate specifier. Returns the following set of parameters:

Name	Type		Description
security	Security	discrete	The securities contract for this rate.

Security security() security

Returns the securities contract that this rate is for.

Collection<ActualRateParameter> actualParameters() actualParameters

The set parameters for this specifier. Returns an actual parameter corresponding to each formal parameter where the operation of the same name returns a non-nil result.

Date date() date

Returns the date at which the security price applies.

3 Service Interfaces

3.1 ImpliedSecurityInterestRateConstructor

A rate constructor that takes a yield curve and a securities price and constructs an equivalent interest rate for the security, discounting the security payments by the discount factors given by the yield curve.

A secondary use of this constructor is to allow yield curve constructors to use security prices as points on the yield curve. A partially constructed yield curve is supplied and the interest rate built by finding the interest rate that corresponds to the price.

3.1.1 Relationships

	Class	Description	Notes
⇑	RateConstructor		
↓	ImpliedSecurityInterestRateConstructorService §5.1		

⇑:Inherits ↓:Realized by

3.1.2 Operations

Rate construct(OrderedCollection<Rate> sources) construct
sources: OrderedCollection<Rate> The source rates.
Raises: RateConstructorException

Construct an implied interest rate. The supplied sources should be a yield curve and a security price. The currency of the security and the yield curve should be the same; otherwise, raise a RateConstructorException.

The basic approach to construction is to use the yield curve to remove coupon payments from the security price, leaving a single payment at the maturity date of the security.

Individual implementations may use different calculation techniques and allocate rate pieces in different ways.

RateFunctionSpecifier result() result

Returns the output rate specification. This is an InterestRateSpecifier with the currency of the yield curve and security.

ImpliedSecurityInterestRateDerivationSpecifier deriver() deriver

The derivation method. Returns the derivation method for this constructor.

OrderedCollection<RateFunctionSpecifier> sources() sources

The source rate specifiers. Returns the result of sending sources() to the deriver.

InterestRate strip(YieldCurveConstructor yieldCurveConstructor, Collection<InterestRate> baseRates, SecurityPrice securityPrice) strip
yieldCurveConstructor: YieldCurveConstructor The constructor that is attempting to build the yield curve.
baseRates: Collection<InterestRate> The current collection of interest rate points.
securityPrice: SecurityPrice The security price to strip.

Raises: RateConstructorException

Strips a security price of coupons and returns an interest rate that reflects the security price. This operation finds an interest rate running from the conversion date to the maturity date of the security which, when added to the interest rates which already make up the yield curve and interpolated by the curve constructor correctly values the security.

Deriving the correct rate may involve complex, iterative, curve building.[2] Individual implementations may use different calculation techniques and allocate rate pieces in different ways.

Raises a RateConstructorException if the supplied rates are in error or if the rate cannot be constructed.

3.2 ImpliedSecurityPriceConstructor

A constructor that builds an implied security price from a yield curve and security definition.

3.2.1 Relationships

	Class	Description	Notes
⇑	RateConstructor		
↓	ImpliedSecurityPriceConstructorService §5.2		

⇑:Inherits ↓:Realized by

3.2.2 Operations

Rate construct(OrderedCollection<Rate> sources) construct

sources: OrderedCollection<Rate> The source rates.

Raises: RateConstructorException

Constructs an implied security price. The supplied sources should be a single yield curve, with the same currency as the currency of the security; otherwise, raises a RateConstructorException.

An outline of the basic approach taken here is that the cashflows of the security are discounted to build an implied price. Specific implementations will split the discounted value in different ways.

[2] The method the curve constructor uses may also have an effect. A linear curve constructor can pre-construct a curve for the known interest rates and concentrate on deriving the security-based in interest rate. A curve constructor using cubic-spline interpolation will need to rebuild the entire curve on each iteration.

RateFunctionSpecifier result() result

Returns the output rate specification. This must be a SecurityPriceSpecifier §2.10.

Security security() security

The security that this constructor derives a price for. Returns the security() operation on the deriver.

ImpliedSecurityPriceDerivationSpecifier deriver() deriver

The derivation method. Returns the derivation method for this constructor.

OrderedCollection<RateFunctionSpecifier> sources() sources

The source rate specifiers. Returns a collection consisting of the application of yieldCurve() to the deriver.

3.3 RelativeSecurityPriceConstructor

A rate constructor that takes a base security price and produces an offset security price for another security.

3.3.1 Relationships

	Class	Description	Notes
⇑	RateConstructor		
↓	RelativeSecurityPriceConstructorService §5.3		

⇑:Inherits ↓:Realized by

3.3.2 Operations

Rate construct(OrderedCollection<Rate> sources) construct
sources: OrderedCollection<Rate>
Raises: RateConstructorException

Builds a relative security price. The sources must consist of a single security price with a logical rate specifier that matches the base rate specifier; otherwise, raises a RateConstructorException. This price is the base rate.

A new security price, with a logical rate specifier given by the result() is built. A new component is built for each base rate component. Each component of the new security price is built in the following manner:

The associated margins are queried to get the margin component with the same identifier as component. If there is no margin with the identifier of the component, get the margin with an identifier of "default". If no margin exists, the new component is the base component.

Constructs a new rate component by adding each rate piece from the margin to each rate piece of the component. If there are two pieces with the same name, adds the two pieces together, otherwise simply includes the named piece.

As an example, suppose the base rate had a bid component of 100.34+0.05(risk)+0.10(profit) and there is a margin of +0.06(risk)+0.10(time) then the resulting component would be 100.34+0.11(risk)+0.10(profit)+0.10(time).

Returns the resulting security price.

ImpliedRateDerivationSpecifier deriver() deriver

Returns the deriver for this rate. The returned object gives the base rate and margins to apply.

RateFunctionSpecifier base() base

The base rate. Returns the result of the base() operation on the deriver.

Collection<RateQuote> margins() margins

Returns a collection of components which give the margins to add to each component of the base rate. Returns the result of the margins() operation on the deriver.

OrderedCollection<RateFunctionSpecifier> sources() sources

The source rates. Returns the result of the sources() operation on the deriver.

4 Classes

4.1 BasicSecurityPriceModel

A form of the basic point rate designed to hold security prices. Components of this interface are restricted to being BasicSecurityPriceComponent components.

4.1.1 Relationships

	Class	Description	Notes
⇑	BasicPointRateModel		
↑	SecurityPrice §2.6		

⇑:Inherits ↑:Realizes

4.2 ExOrCumEnum

An enumeration indicating whether a security price is quoted ex or cum coupon. A security is traded ex coupon if the purchaser is not entitled to receive the next coupon.

4.2.1 Operations

«Static Method» ExOrCumEnum ex() ex

Returns an instance with identifier 'ex'.

«Static Method» ExOrCumEnum cum() cum

Returns an instance with identifier 'cum'.

«Static Method» Collection<ExOrCumEnum> elements() elements

Returns a collection consisting of the results of the ex() and cum() operations.

4.3 ImpliedSecurityInterestRateDerivationSpecifierModel

A concrete implementation of the ImpliedSecurityInterestRateDerivationSpecifier §2.1 interface. The security's price may be either for a specific security (and derivation method) or a logical security chosen from a benchmark sequence.

4.3.1 Relationships

	Class	Description	Notes
↑	ImpliedSecurityInterestRateDerivationSpecifier §2.1		
↔	SecurityBenchmarkSequence §2.4	benchmark 0..1	→
↔	Period	period 0..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

4.3.2 Operations

RateFunctionSpecifier yieldCurve() yieldCurve

The yield curve used to discount the security cashflows. Return the associated curve.

RateFunctionSpecifier securityPrice() securityPrice

The security price. If there is an associated price, return the associated price. Otherwise, return a rate specifier consisting of a logical rate specifier containing the security chosen by the associated benchmark for the associated period and no derivation method.

4.4 ImpliedSecurityPriceDerivationSpecifierModel

A concrete implementation of the ImpliedSecurityPriceDerivationSpecifier interface.

4.4.1 Relationships

	Class	Description	Notes
↑	ImpliedSecurityPriceDerivationSpecifier §2.2		

↑:Realizes

4.4.2 Operations

RateFunctionSpecifier yieldCurve() yieldCurve

The yield curve used to discount the security cashflows. Returns the associated curve.

4.5 PricePer100FaceValueMarginModel

A margin which represents an additional amount added to a price per hundred face value amount.

This form of margin represents the only form of margin modeled at present. The non-linearity of yields makes margin conversion difficult.

4.5.1 Relationships

	Class	Description	Notes
↑	SecurityMarginQuotationMethod §2.5		

↑:Realizes

4.5.2 Attributes

printStyle: PriceFractionStyleEnum = PriceFractionStyleEnum.decimal() The fractional printing convention to use.

4.5.3 Operations

Number asCanonical(Number r, RateFunctionSpecifier specifier) asCanonical
r: Number The rate to convert.
specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in this quotation format to the canonical quotation format. Returns r.

Number fromCanonical(Number r, RateFunctionSpecifier specifier) fromCanonical
r: Number The rate to convert.
specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in the canonical quotation format to this quotation format. Returns r.

Number parse(InputStream stream, Boolean loose, RateFunctionSpecifier specifier) parse
stream: InputStream The stream to read from.
loose: Boolean Perform loose parsing.
specifier: RateFunctionSpecifier The specifier to use when parsing this rate.
Raises: ParseException

Read a text representation of a rate and turn it into an appropriately quoted rate. There are three possible text representations:

{+-}www.ff Decimal representation.

{+-}{www} nn/dd Explicit fractional representation.

{+-}www-nn Implicit fractional representation.

Where *www* represents a whole number, *ff* the decimal fraction of a number, *nn* a fractional numerator and *dd* a fractional denominator. Braces ({}) represent optional portions of the number.

If loose is true, then any input format is permitted. If loose is false, then the input format depends on the value of the printStyle attribute: decimal representation only in the case of decimal, implicit or explicit fractional representation in the case of fractional or half-fractional.

If the implicit fractional representation is used, the denominator of the fraction is 64 if printStyle is halfFraction, 32 otherwise.

printRate(OutputStream stream, Number rate, Boolean loose, RateFunctionSpecifier specifier) printRate

stream: OutputStream The stream to print onto.
rate: Number The rate to print.
loose: Boolean Print the rate in loose form?
specifier: RateFunctionSpecifier The specifier for the rate.

Prints a rate piece on the output stream.

If the printStyle is decimal, then if loose is true, print r to the natural level of accuracy of r. If loose is false, print r using a format of `[+-]#0.00`.

If the printStyle is fractional or halfFractional then: If loose is true, print r using the format `[+-]www-nn`, where www is the whole part of r and nn is the numerator of a fraction with a denominator of 32 for fractional and 64 for halfFractional print styles. If loose is false, print r using the format `[+-]{www} nn/dd`, where dd is equal to 32 for fractional and 64 for halfFractional print styles. The numerator is rounded off.

Brackets ([]) represent non-optional choices — margins are always signed. Braces ({}) represent optional components.

Number addMargin(SecurityPriceQuotationMethod baseQuote, Number base, Number margin) addMargin

baseQuote: SecurityPriceQuotationMethod The quotation method used for the base price.
base: Number The base price.
margin: Number The margin to add.

Adds a margin to a securities price.

Converts base from the baseQuote quotation convention to a PricePer100FaceValueModel §4.11 convention, keeping the clean or dirty status.

Adds margin to the resulting base value. Converts the sum back into the baseQuote quotation convention and returns the result.

4.6 RelativeSecurityPriceSpecifierModel

A concrete implementation of the RelativeSecurityPriceSpecifier interface.

A base rate is specified by a rate specifier. This rate must have a logical specification of type “Security Price”. A derivation path is optional.

A collection of margins gives the margin for each rate component.

4.6.1 Relationships

	Class	Description	Notes
↑	RelativeSecurityPriceSpecifier §2.3		
↔	SecurityPriceQuote §2.9	margins 0..n	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.7 SecurityBechmarkReferenceDataModel

A reference data implementation of the SecurityBenchmarkSequence interface. Operations are delegated to the held benchmark sequence.

4.7.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	SecurityBenchmarkSequence §2.4		
↔	SecurityBenchmarkSequence §2.4	model 1..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.8 SecurityBenchmarkSequenceModel

A concrete implementation of the SecurityBenchmarkSequence interface. All associated securities must be denominated in the same currency.

4.8.1 Relationships

	Class	Description	Notes
↑	SecurityBenchmarkSequence §2.4		
↑	Validatable		
↔	Security	sequence 1..n	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

4.8.2 Attributes

identifier: String The unique identifier for the sequence.

4.8.3 Operations

OrderedCollection<Security> sequence() sequence

The benchmark securities. Returns the associated securities, sorted by maturity date.

Reportable validate() validate

Validate the benchmark sequence.

- Add an error if all securities in the sequence do not share a common currency.
- Add an error if two securities share a common maturity date.
- Add an error if all securities in the sequence have maturity dates before the current processing date.
- Add a warning if not all securities have a common issuer.

4.9 SecurityPricePieceModel

A concrete implementation of the SecurityPricePiece interface. Quotation methods used by this model must implement the SecurityPriceQuotationMethod §2.8 or SecurityMarginQuotationMethod §2.5 interfaces.

4.9.1 Relationships

	Class	Description	Notes
⇑	RatePieceModel		
↑	SecurityPricePiece §2.7		

⇑:Inherits ↑:Realizes

4.10 SecurityPriceQuotationMethodModel

An abstract superclass for the various security price quotation methods. All security prices may be quoted as clean or dirty.

4.10.1 Relationships

	Class	Description	Notes
↑	SecurityPriceQuotationMethod §2.8		
⇓	PricePer100FaceValueModel §4.11		
⇓	SecurityYieldModel §4.12		

⇓:Inherited by ↑:Realizes

4.10.2 Attributes

cleanOrDirty: CleanDirtyEnum = clean Whether or not this price is a clean or dirty quotation.

exOrCum: ExOrCumEnum = cum Is the bond traded ex or cum coupon?

4.11 PricePer100FaceValueModel

The price represents the price paid for 100 units of the face value of the security, using the currency in which the security is denominated.

4.11.1 Relationships

	Class	Description	Notes
⇑	SecurityPriceQuotationMethod-Model §4.10		

⇑:Inherits

4.11.2 Attributes

printStyle: PriceFractionStyleEnum = PriceFractionStyleEnum.decimal() The fractional printing convention to use.

4.11.3 Operations

Number asCanonical(Number r, RateFunctionSpecifier specifier) asCanonical

r: Number The rate to convert.
specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in this quotation format to the canonical quotation format. If this quotation method is in clean form, then returns r, otherwise converts r to the clean form, using the cleanPrice operation and returns the converted rate. Note that the specifier parameter will actually be a SecurityPriceSpecifier, which knows the appropriate date to perform the clean price calculations.

Number fromCanonical(Number r, RateFunctionSpecifier specifier) fromCanonical

r: Number The rate to convert.
specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in the canonical quotation format to this quotation format. If this quotation method is in clean form, then returns r, otherwise converts r to the dirty form, using the dirtyPrice operation and returns the converted rate. Note that the specifier parameter will actually be a SecurityPriceSpecifier, which knows the appropriate date to perform the clean price calculations.

Number parse(InputStream stream, Boolean loose, RateFunctionSpecifier specifier) parse

stream: InputStream The stream to read from.
loose: Boolean Perform loose parsing.
specifier: RateFunctionSpecifier The specifier to use when parsing this rate.
Raises: ParseException

Reads a text representation of a rate and turns it into an appropriately quoted rate. There are three possible text representations:

www.ff Decimal representation.

www nn/dd Explicit fractional representation.

www-nn Implicit fractional representation.

Where *www* represents a whole number, *ff* the decimal fraction of a number, *nn* a fractional numerator and *dd* a fractional denominator.

If loose is true, then any input format is permitted. If loose is false, then the input format depends on the value of the printStyle attribute: decimal representation only in the case of decimal, implicit or explicit fractional representation in the case of fractional or half-fractional.

If the implicit fractional representation is used, the denominator of the fraction is 64 if printStyle is halfFraction, 32 otherwise.

printRate(OutputStream stream, Number rate, Boolean loose, RateFunctionSpecifier specifier) printRate

stream: OutputStream The stream to print onto.
rate: Number The rate to print.
loose: Boolean Print the rate in loose form?
specifier: RateFunctionSpecifier The specifier for the rate.

Prints a rate piece on the output stream.

If the printStyle is decimal, then if loose is true, prints r to the natural level of accuracy of r. If loose is false, prints r using a format of "#0.00".

If the printStyle is fractional or halfFractional then: If loose is true, prints r using the format "www-nn", where www is the whole part of r and nn is the numerator of a fraction with a denominator of 32 for fractional and 64 for halfFractional print styles. If loose is false, prints r using the format "www nn/dd", where dd is equal to 32 for fractional and 64 for halfFractional print styles. The numerator is rounded off.

Number cleanPrice(Number r, SecurityPriceSpecifier specifier) cleanPrice

r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Converts a dirty price into a clean price. If this quotation method is clean, returns r.

If this quotation method is dirty, then uses the security returned by the specifier to calculate the accrued interest from the supplied trade date to the next coupon date of the security for 100 units, face value of this security. Subtracts the resulting amount from r and returns the result.

Number dirtyPrice(Number r, SecurityPriceSpecifier specifier) dirtyPrice

r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Converts a clean price into a dirty price. If this quotation method is dirty, returns r.

If this quotation method is clean, then uses the security returned by the specifier to calculate the accrued interest from the supplied trade date to the next coupon date of the security for 100 units, face value of this security. Adds the resulting amount to r and returns the result.

4.12 SecurityYieldModel

The price represents an interest-rate style yield for the security. The various forms of quotation convention that apply to interest rates apply to this yield, although an annualized yield is the most common convention.

This model represents a true yield to maturity representation. Subclasses implement the various simplifications.

4.12.1 Relationships

	Class	Description	Notes
⇑	SecurityPriceQuotationMethod-Model §4.10		
⇓	SecurityCurrentYieldModel §4.13		
⇓	SecuritySimpleYieldModel §4.14		

⇑:Inherits ⇓:Inherited by

4.12.2 Attributes

interestRateQuotation: InterestRateQuotationMethod The quotation convention for this interest rate. This quotation convention defaults to an annualized yield.

4.12.3 Operations

Number asCanonical(Number r, RateFunctionSpecifier specifier) asCanonical
r: Number The rate to convert.
specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in this quotation format to the canonical quotation format.

Let $\{p_1, \ldots, p_n\}$ be the payments of 100 units of face value for the security defined by the supplied specifier. Let $\{d_1, \ldots, d_n\}$ be the payment dates and $\{t_1, \ldots, t_n\}$ be the terms in years between the conversion date and each d_i, using the security's payment date basis. Let y be r converted from the interestRateQuotation interest rate quotation method to an annualized, compounding convention.

Calculate

$$r' = \Sigma_{i=1}^{n} p_i (1 + y)^{-t_i}$$

If this quotation method is a clean price, then returns r'. Otherwise, uses a dirty PricePer100FaceValueModel to convert r' to an equivalent clean price.

Number fromCanonical(Number r, RateFunctionSpecifier specifier) fromCanonical

r: Number The rate to convert.

specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in the canonical quotation format to this quotation format.

If this quotation method is a clean price, then returns $r' = r$. Otherwise, uses a dirty PricePer100FaceValueModel to convert r to an equivalent dirty price, r'.

Let $\{p_1, \ldots, p_n\}$ be the payments of 100 units of face value for the security defined by the supplied specifier. Let $\{d_1, \ldots, d_n\}$ be the payment dates and $\{t_1, \ldots, t_n\}$ be the terms in years between the conversion date and each d_i, using the security's payment date basis.

Calculate the solution to y so that

$$r' = \Sigma_{i=1}^{n} p_i (1 + y)^{-t_i}$$

y' is in the form of an annualized compounding yield. Returns the conversion of y' from an annualized compounding yield to the convention contained in interestRateQuotation.

Number parse(InputStream stream, Boolean loose, RateFunctionSpecifier specifier) parse

stream: InputStream The stream to read from.

loose: Boolean Perform loose parsing.

specifier: RateFunctionSpecifier The specifier to use when parsing this rate.

Raises: ParseException

Reads a text representation of a rate and turns it into an appropriately quoted rate. Delegates to the interestRateQuotation attribute.

printRate(OutputStream stream, Number rate, Boolean loose, RateFunctionSpecifier specifier) printRate
stream: OutputStream The stream to print onto.
rate: Number The rate to print.
loose: Boolean If true then print the rate in loose form
specifier: RateFunctionSpecifier The specifier for the rate.

Prints a rate piece on the output stream. Delegates to the interestRateQuotation attribute.

Number cleanPrice(Number r, SecurityPriceSpecifier specifier) cleanPrice
r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Converts a dirty price into a clean price. If this quotation method is clean, returns r.

Otherwise, converts r into canonical form, and converts the canonical form into a quotation method with the same characteristics as this method, but with a clean price.

Number dirtyPrice(Number r, SecurityPriceSpecifier specifier) dirtyPrice
r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Converts a clean price into a dirty price. If this quotation method is dirty, returns r.

Otherwise, converts r into canonical form, and converts the canonical form into dirty form and then a quotation method with the same characteristics as this method, but with a dirty price.

4.13 SecurityCurrentYieldModel

A security price quoted using the current yield quotation convention.

Current yields only use clean prices, the cleanOrDirty attribute must, therefore, always be set to clean.

4.13.1 Relationships

	Class	Description	Notes
⇑	SecurityYieldModel §4.12		

⇑:Inherits

4.13.2 Operations

Number asCanonical(Number r, RateFunctionSpecifier specifier) asCanonical
r: Number The rate to convert.
specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in this quotation format to the canonical quotation format. Let i be the coupon rate for the security supplied by specifier. Returns

$$100 \times \frac{i}{r}$$

Number fromCanonical(Number r, RateFunctionSpecifier specifier) fromCanonical
r: Number The rate to convert.
specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in the canonical quotation format to this quotation format. Let i be the coupon rate of the security supplied by specifier. Returns

$$y = \frac{i}{r/100}$$

Number cleanPrice(Number r, SecurityPriceSpecifier specifier) cleanPrice
r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Converts a dirty price into a clean price. Returns r.

Number dirtyPrice(Number r, SecurityPriceSpecifier specifier) dirtyPrice
r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Converts a clean price into a dirty price. Current yield quotation methods do not allow dirty prices; raises a RateQuotationException.

4.14 SecuritySimpleYieldModel

A security price quoted using the simple yield to maturity quotation convention.

Simple yields only use clean prices, the cleanOrDirty attribute must, therefore, always be set to clean.

4.14.1 Relationships

	Class	Description	Notes
⇑	SecurityYieldModel §4.12		

⇑:Inherits

4.14.2 Operations

Number asCanonical(Number r, RateFunctionSpecifier specifier) asCanonical

r: Number The rate to convert.

specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in this quotation format to the canonical quotation format.

Let i be the coupon rate for the security supplied by specifier. Let t be the number of years from the conversion date to the maturity date of the security, using the date basis of the security.

Returns

$$\frac{i + 100/t}{r + 1/t}$$

Number fromCanonical(Number r, RateFunctionSpecifier specifier) fromCanonical

r: Number The rate to convert.

specifier: RateFunctionSpecifier The specifier to use when converting this rate.

Converts a rate in the canonical quotation format to this quotation format.

Let i be the coupon rate for the security supplied by specifier. Let t be the number of years from the conversion date to the maturity date of the security, using the date basis of the security.

Returns

$$\frac{i + \frac{100 - r}{t}}{r / 100}$$

Number cleanPrice(Number r, SecurityPriceSpecifier specifier) cleanPrice
r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Converts a dirty price into a clean price. Returns r.

Number dirtyPrice(Number r, SecurityPriceSpecifier specifier) dirtyPrice
r: Number The rate to convert.
specifier: SecurityPriceSpecifier The rate specifier describing this price.
Raises: RateQuotationException

Converts a clean price into a dirty price. Current yield quotation methods do not allow dirty prices; raises a RateQuotationException.

4.15 SecurityPriceQuoteModel

A concrete implementation of the SecurityPriceQuote interface. Instances of this class are restricted to holding SecurityPricePiece §2.7 rate pieces.

4.15.1 Relationships

	Class	Description	Notes
⇑	RateQuoteModel		
↑	SecurityPriceQuote §2.9		

⇑:Inherits ↑:Realizes

4.16 SecurityPriceSpecifierModel

A concrete implementation of the SecurityPriceSpecifier interface. The relevant parameters are modeled as associations.

4.16.1 Relationships

	Class	Description	Notes
↑	SecurityPriceSpecifier §2.10		
↔	Security	security 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.16.2 Attributes

date: Date

5 Services

5.1 ImpliedSecurityInterestRateConstructorService

An example interest rate constructor. Rate pieces are allocated to match the rate pieces in the source rates.

5.1.1 Relationships

	Class	Description	Notes
↑	ImpliedSecurityInterestRateConstructor §3.1		
↔	ImpliedSecurityInterestRateDerivationSpecifier §2.1	deriver	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

5.1.2 Operations

Rate construct(OrderedCollection<Rate> sources) construct
sources: OrderedCollection<Rate> The source rates.
Raises: RateConstructorException

Constructs an implied interest rate. The supplied sources should be a yield curve and a security price. The currency of the security and the yield curve should be the same; otherwise, raises a RateConstructorException.

Constructs an interest rate with the currency of the yield curve, a from date, d_0 of the conversion date and a to date of the security's maturity date. The interest rate has a party which is the issuer of the security and a location which is country of the security. The date basis of the interest rate is the date basis of the security.

The interest rate has components consisting of the intersection of the components of the yield curve and the security price. For each component, makes the following calculation:

Let $\{s_1, \ldots, s_m\}$ be the union of the piece names of both the yield curve and security price, with yield curve pieces coming before security price pieces. Let $\{p_1, \ldots, p_n\}$ be the remaining payments for 100 face value units of the security, with each payment p_i being made on the date d_i. Let $D_{d_1,d_2,j--k}$ be the discount factor between two dates, d_1 and d_2, using only pieces $j--k$ of the yield curve.

For each piece, l, Let v_l be the component of the security price, using only the pieces from 1–l, converted to a PricePer100FaceValueModel §4.11, dirty quotation method. Calculates a residual value of:

$$r = v - \Sigma_{i=1}^{n-1} p_i D_{d_0,d_i,1--l}$$

and an interest rate

$$y_l = \left(\frac{p_n}{r}\right)^{\frac{1}{t}} - 1$$

where t is the term in years between d_0 and d_n calculated using the date basis of the security.

Constructs a rate component consisting of $\{(s_i, y_i - y_{i-1})\}$ with each piece quoted as an annualized yield.

As an example, suppose we have a security with a coupon date of 7% at 14-Mar-2002 and a maturity date of 14-Mar-2003, with an Actual/Actual date basis The security price, quoted as a dirty PP100FV, is 102.23(base)-0.56(risk). The yield curve gives 6%(base) +150bp(country) Actual/Actual at 14-Mar-2002. The conversion date is 1-Jan-2002. The pieces are calculated as follows:

base $r = 102.23 - 7 \times 1.06^{-72/365} = 95.31$ and $y = (100/95.31)^{365/437} - 1 = 4.09\%$.

country $r = 102.23 - 7 \times 1.075^{-72/365} = 95.33$ and $y = (100/95.33)^{365/437} - 1 = 4.08\%$.

risk $r = 101.67 - 7 \times 1.075^{-72/365} = 94.77$ and $y = (100/95.33)^{365/437} - 1 = 4.59\%$.

The resulting rate is, therefore, 4.09%(base)-1bp(country)+51bp(risk).

ImpliedSecurityInterestRateDerivationSpecifier deriver() deriver

The derivation method. Return the associated deriver.

InterestRate strip(YieldCurveConstructor yieldCurveConstructor, Collection<InterestRate> baseRates, SecurityPrice securityPrice) strip

yieldCurveConstructor: YieldCurveConstructor The constructor that is attempting to build the yield curve.

baseRates: Collection<InterestRate> The current collection of interest rate points.

securityPrice: SecurityPrice The security price to strip.

Raises: RateConstructorException

Strips a security price of coupons and returns an interest rate that reflects the security price. The input interest rates should be interest rates with the same currency as the security and with to-dates before the maturity date of the security. Raises a RateConstructorException if these conditions are not met.

For each component of the intersection of the yield curve and security price components, builds an interest rate using the following method:

Let $\{s_1, \ldots, s_m\}$ be the rate pieces of the interest rates. For each piece, l, construct an interest rate, y_l, with the following properties:

The from-date of y_l is the conversion date, d_0. The to-date of y_l is the maturity date of the security. The currency of y_l is the currency of the security. The party of y_l is the issuer of the security. The location of y_l is the country of the security. The date basis of y_l is the date basis of the security.

Let v be the security price expressed as a PricePer100FaceValueModel §4.11, dirty quotation method, using only pieces $1 - -l$ of the security price or, in the case of $l = m$, all pieces of the security price.[3]

Uses the supplied curve constructor to build a new yield curve consisting of the supplied interest rates and y_l, using only pieces $1 - -l$ of the supplied interest rates. The curve must have the property that:

$$v = \Sigma_{i=1}^{n} p_i D_{d_0, d_i}$$

where d_i is the date of the i^{th} payment, p_i is the payment amount for 100 units face value of the security and D_{d_1, d_2} is the discount factor given by the yield curve between dates d_1 and d_2.

Constructs a component consisting of $\{(s_i, y_i - y_{i-1})\}$ for the resulting interest rate.

As an example, suppose the security is an 8% bond paying a coupon on 13-Jun-2005, maturing on 12-Jun-2006 with an Actual/Actual date basis. The conversion date is 2-Jan-2005. Supplied interest rates are 5%(base)+100bp(risk) from 2-Jan-2005 to 2-Mar-2005 and 6%+100bp(risk) from 2-Jan-2005 to 2-Jan-2006, using

[3] Unallocated security price pieces are bundled into the final piece of the interest rate.

an Actual/Actual date basis. The price for the bond is 101.00(base)-1.00(risk)-0.50(country). A linear yield curve constructor is being used. The interest rates obey the following formulae:

base $101.00 = 8 \times 1.0530^{-162/365} + 100 \times (1+y)^{-526/365}$ or $y = 5.02\%$.

risk $y = 99.50 - 8 \times 1.0630^{-162/365} + 100 \times (1+y)^{-526/365}$ or $y = 6.19\%$.

The resulting interest rate is, therefore, 5.02%(base)+115bp(risk).

The example above is a simple example, using linear interpolation and having a coupon that can be correctly discounted using the supplied interest rates. More complex construction methods, or coupons beyond the last interest rate will require more sophisticated root-finding techniques.

5.2 ImpliedSecurityPriceConstructorService

A concrete implementation of the ImpliedSecurityPriceConstructor service interface.

5.2.1 Relationships

	Class	Description	Notes
↑	ImpliedSecurityPriceConstructor §3.2		
↔	ImpliedRateDerivationSpecifier	deriver 1..1	→
↔	SecurityPriceSpecifier §2.10	result	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

5.2.2 Operations

Rate construct(OrderedCollection<Rate> sources) construct
sources: OrderedCollection<Rate> The source rates.
Raises: RateConstructorException

Constructs an implied security price. The supplied sources should be a single yield curve, with the same currency as the currency of the security; otherwise, raises a RateConstructorException.

Let $\{p_1, \ldots, p_n\}$ be the outstanding payments for 100 units of face value of the security, at dates $\{d_1, \ldots, d_n\}$. Let d_0 be the value date for the security, equal to SecurityPriceQuotationMethod.conversionDate().

For each component of the yield curve: Let $\{n_1, \ldots, n_m\}$ be the pieces of the yield curve component. Then

$$t_j = \Sigma_{i=0}^{n} p_i DF_{d_0,d_i,1--j}$$

where $DF_{f,t,l--k}$ is the discount factor derived from the yield curve between the f and t dates and only including pieces l to k.

The component of the security price is constructed from pieces $\{n_i, t_i - t_{i-1}\}$, each with a dirty PricePer100FaceValueModel §4.11 quotation method.

As an example, suppose that the security being valued is denominated in GBP, paying 6dates of 12-Jan-2004, 12-Jul-2004 and a maturity date of 12-Jan-2005. The valuation date is 12-Dec-2003. The supplied yield curve's bid component is 5%+100bp(risk) on 12-Jan-2004, and 5.5%+150bp(risk) on 12-Jul-2004 and 12-Jan-2005, with a 30/360 date basis.

$$t_1 = 3 \times 1.05^{-30/360} + 3 \times 1.055^{-210/360} + 100 \times 1.055^{-390/360} = 100.26$$

and

$$t_1 = 3 \times 1.06^{-30/360} + 3 \times 1.07^{-210/360} + 100 \times 1.07^{-390/360} = 98.80$$

The resulting bid price is, therefore, 100.26-1.46(risk).

RateFunctionSpecifier result() result

The output rate specification. Returns the associated specifier.

ImpliedSecurityPriceDerivationSpecifier deriver() deriver

The derivation method. Returns the associated deriver.

5.3 RelativeSecurityPriceConstructorService

A service implementation of the RelativeSecurityPriceConstructor service interface.

5.3.1 Relationships

	Class	Description	Notes
↑	RelativeSecurityPriceConstructor §3.3		
↔	RelativeSecurityPriceSpecifier §2.3	deriver 1..1	→
↔	RateFunctionSpecifier	result 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

5.3.2 Operations

ImpliedRateDerivationSpecifier deriver() deriver

The deriver for this rate. Returns the associated deriver.

RateFunctionSpecifier result() result

The output rate specification. Returns the associated specifier.

6 Enumerations

6.1 CleanDirtyEnum

An enumeration indicating whether a securities price is clean or dirty.

6.1.1 Relationships

	Class	Description	Notes
⇑	Enum		

⇑:Inherits

6.1.2 Operations

«Static Method» CleanDirtyEnum clean() clean

Returns an instance with an identifier of “clean”.

«Static Method» CleanDirtyEnum dirty() dirty

Returns an instance with an identifier of “dirty”.

«Static Method» Collection<CleanDirtyEnum> elements() elements

Returns a collection consisting of the results of the clean() and dirty() operations.

6.2 PriceFractionStyleEnum

An enumeration covering the various ways of quoting fractional pieces of a securities price.

6.2.1 Operations

«Static Method» PriceFractionStyleEnum decimal() decimal

Returns an instance of this class with an identifier of “decimal”.

«Static Method» PriceFractionStyleEnum fraction() fraction

Returns an instance of this class with an identifier of “fraction”.

«Static Method» PriceFractionStyleEnum halfFraction() halfFraction

Returns an instance of this class with an identifier of “half fraction”.

«Static Method» Collection<Enum> elements() elements

Returns a collection consisting of the results of the decimal(), fraction() and halfFraction() operations.

7 Associations

Table 1: Security Prices— Associations

Association Role	Class	Card.	Notes
security			
security	Security	1..1	→
rate specifier	SecurityPriceSpecifierModel §4.16	0..n	
base price			
base price	SecurityPrice §2.6	1..1	→
relative price	RelativeSecurityPriceSpecifier §2.3	0..n	
margins			
margin	SecurityPriceQuote §2.9	0..n	→
relative price	RelativeSecurityPriceSpecifier §2.3	0..n	
margins			
margin	SecurityPriceQuote §2.9	0..n	→
relative rate	RelativeSecurityPriceSpecifier-Model §4.6	0..n	
deriver			
deriver	RelativeSecurityPriceSpecifier §2.3	1..1	→
constructor	RelativeSecurityPriceConstructorService §5.3	0..n	

Table 1: . . . continued

Association Role	Class	Card.	Notes
result			
specifier	RateFunctionSpecifier	1..1	→
constructor	RelativeSecurityPriceConstructorService §5.3	0..n	
deriver			
deriver	ImpliedRateDerivationSpecifier	1..1	→
constructor	ImpliedSecurityPriceConstructorService §5.2	0..n	
result			
specifier	SecurityPriceSpecifier §2.10		→
constructor	ImpliedSecurityPriceConstructorService §5.2		
deriver			
deriver	ImpliedSecurityInterestRateDerivationSpecifier §2.1		→
constructor	ImpliedSecurityInterestRateConstructorService §5.1		
sequence			
securities	Security	1..n	→
benchmark sequence	SecurityBenchmarkSequenceModel §4.8	0..n	◊
model			
model	SecurityBenchmarkSequence §2.4	1..1	→
wrapper	SecurityBechmarkReferenceDataModel §4.7	0..n	
benchmark			
benchmark	SecurityBenchmarkSequence §2.4	0..1	→
implied specifier	ImpliedSecurityInterestRateDerivationSpecifierModel §4.3	0..n	
period			
period	Period	0..1	→
implied specifier	ImpliedSecurityInterestRateDerivationSpecifierModel §4.3	0..n	

→:Navigable ◊:Aggregate ⧫:Composite

7.1 security

Role: security *Navigable* Security, 1..1.
Role: rate specifier SecurityPriceSpecifierModel, 0..n.
The security contract for the security rate specifier.

7.2 base price

Role: base price *Navigable* SecurityPrice, 1..1.
Role: relative price RelativeSecurityPriceSpecifier, 0..n.
The base price from which a relative security price is calculated.

7.3 margins

Role: margin *Navigable* SecurityPriceQuote, 0..n.
Role: relative price RelativeSecurityPriceSpecifier, 0..n.
The set of margins over the base price for this security. The components that make up the margins should have no base piece (ie. be incomplete).

7.4 margins

Role: margin *Navigable* SecurityPriceQuote, 0..n.
Role: relative rate RelativeSecurityPriceSpecifierModel, 0..n.
The margins to add to the base rate.

7.5 deriver

Role: deriver *Navigable* RelativeSecurityPriceSpecifier, 1..1.
Role: constructor RelativeSecurityPriceConstructorService, 0..n.
The deriver used to construct the new security price.

7.6 result

Role: specifier *Navigable* RateFunctionSpecifier, 1..1.
Role: constructor RelativeSecurityPriceConstructorService, 0..n.
The rate specifier for the resulting rate.

7.7 deriver

Role: deriver *Navigable* ImpliedRateDerivationSpecifier, 1..1.
Role: constructor ImpliedSecurityPriceConstructorService, 0..n.
The derivation method for the implied security price constructor.

7.8 result

Role: specifier *Navigable* SecurityPriceSpecifier.
Role: constructor ImpliedSecurityPriceConstructorService.
The logical rate specifier for an implied security price constructor.

7.9 deriver

Role: deriver *Navigable* ImpliedSecurityInterestRateDerivationSpecifier.
Role: constructor ImpliedSecurityInterestRateConstructorService.
The derivation specification for the constructor.

7.10 sequence

Role: securities *Navigable* Security, 1..n.
Role: benchmark sequence *Aggregate* SecurityBenchmarkSequenceModel, 0..n.

7.11 model

Role: model *Navigable* SecurityBenchmarkSequence, 1..1.
Role: wrapper SecurityBechmarkReferenceDataModel, 0..n.
The actual model used by the reference data model.

7.12 benchmark

Role: benchmark *Navigable* SecurityBenchmarkSequence, 0..1.
Role: implied specifier ImpliedSecurityInterestRateDerivationSpecifierModel, 0..n.
The benchmark series that is used to derive this interest rate.

7.13 period

Role: period *Navigable* Period, 0..1.
Role: implied specifier ImpliedSecurityInterestRateDerivationSpecifierModel, 0..n.

The period at which to choose a benchmark security.

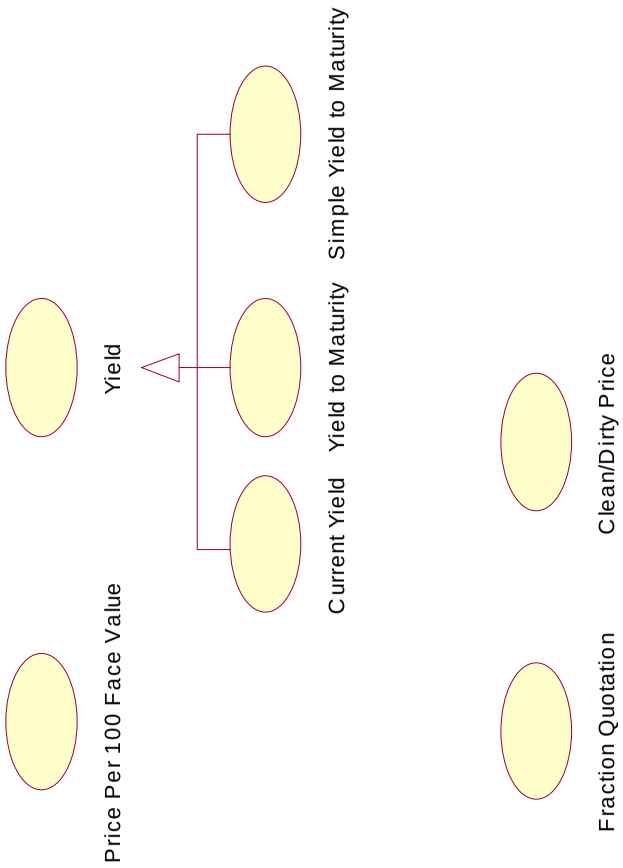


Figure 1: Class Diagram— Quotation Methods

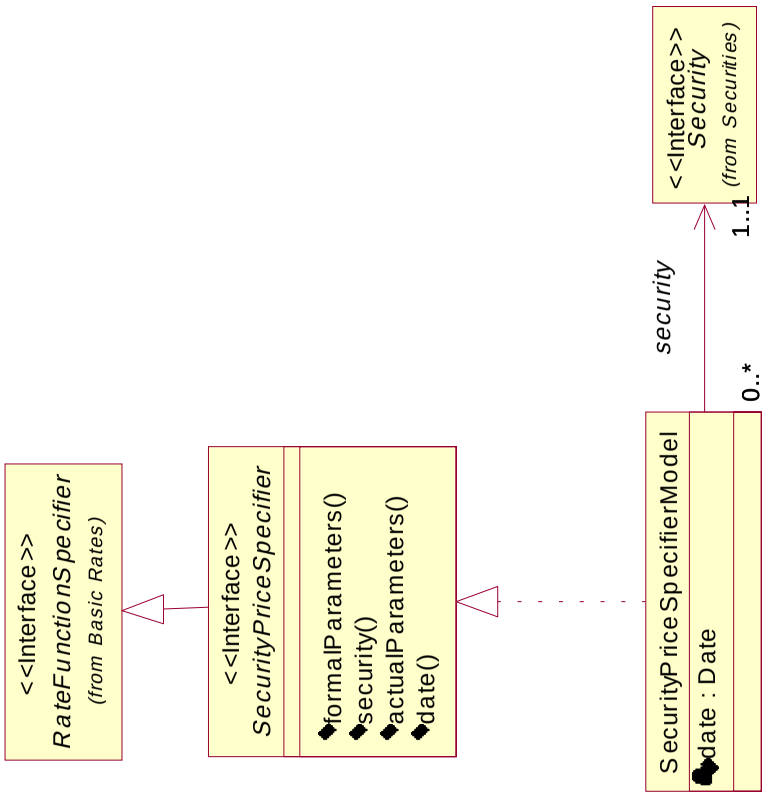


Figure 2: Class Diagram— Security Price Specification1

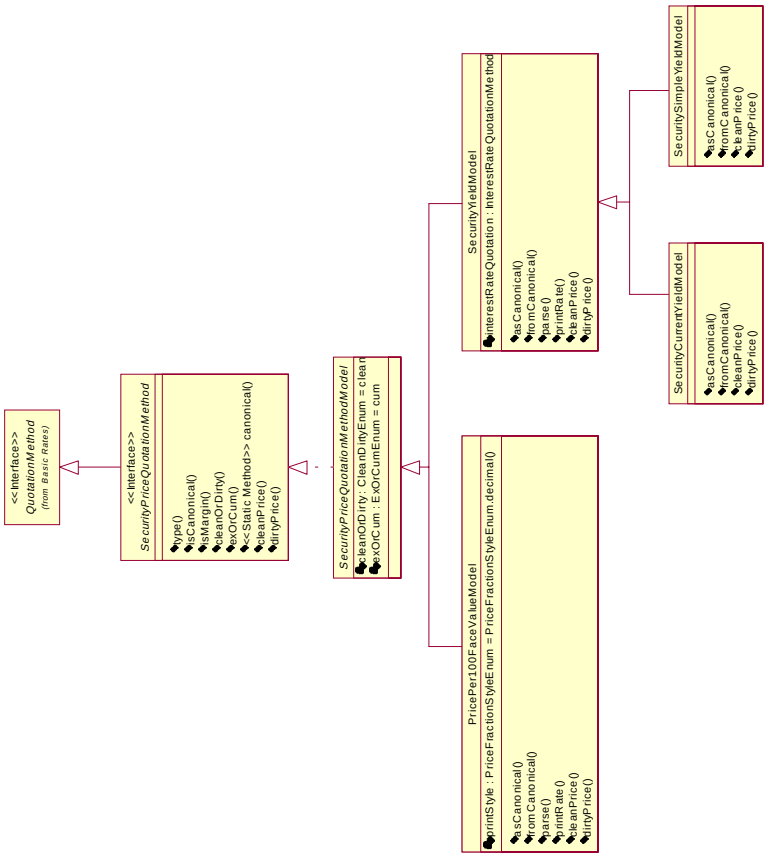


Figure 3: Class Diagram— Security Price Specification2

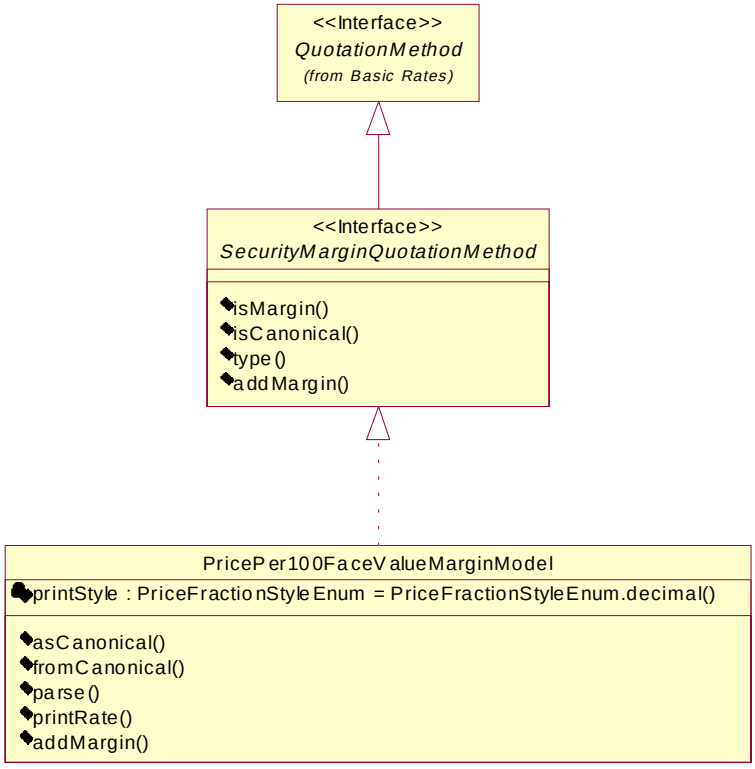


Figure 4: Class Diagram— Security Price Specification3

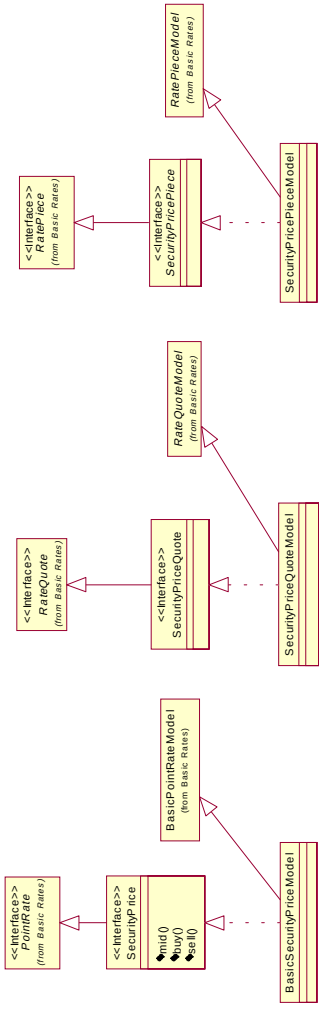


Figure 5: Class Diagram— Point Rates

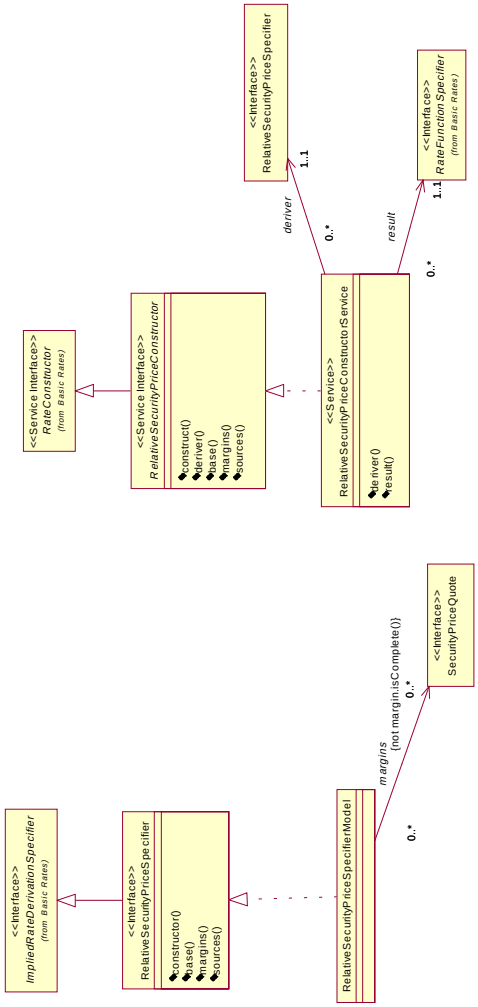


Figure 6: Class Diagram— Relative Security Prices

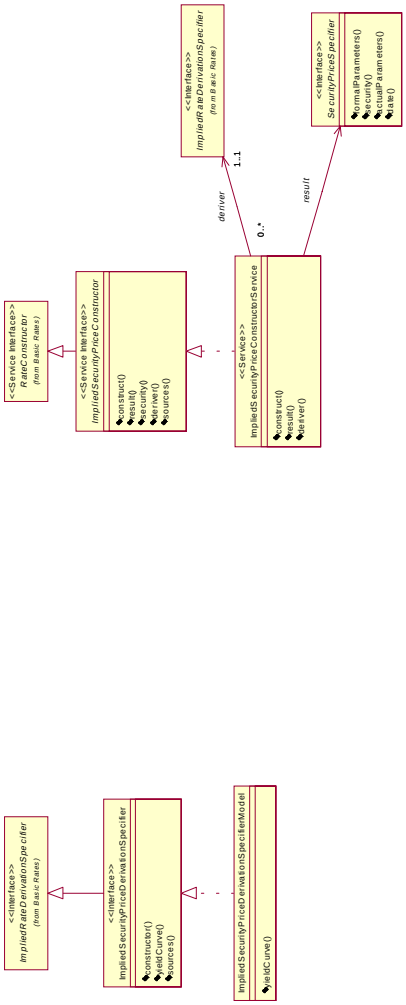


Figure 7: Class Diagram— Implied Security Prices

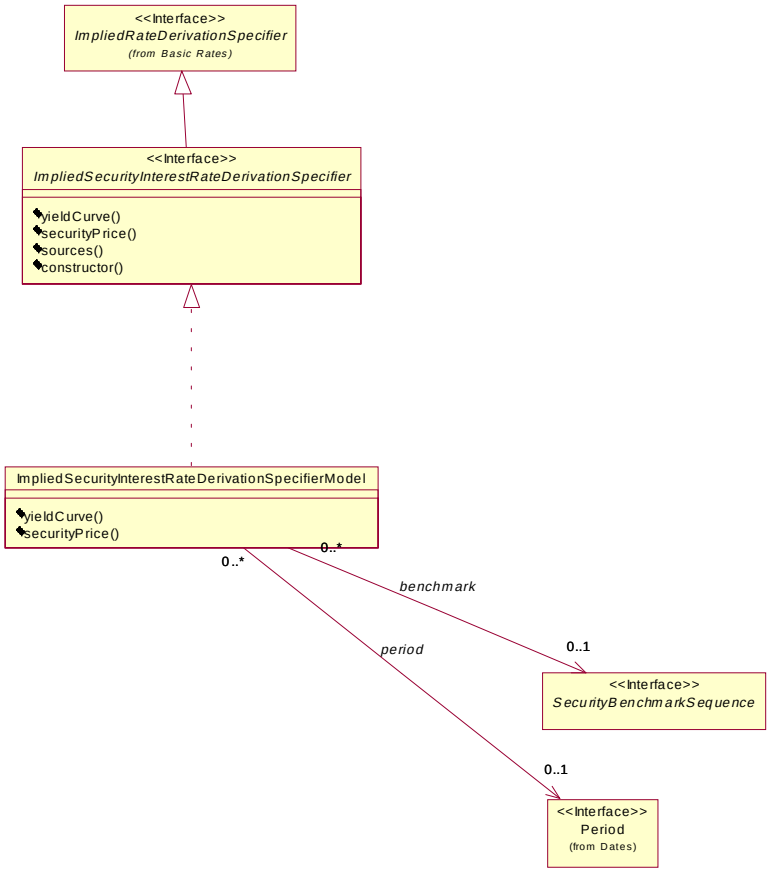


Figure 8: Class Diagram— Implied Interest Rates1

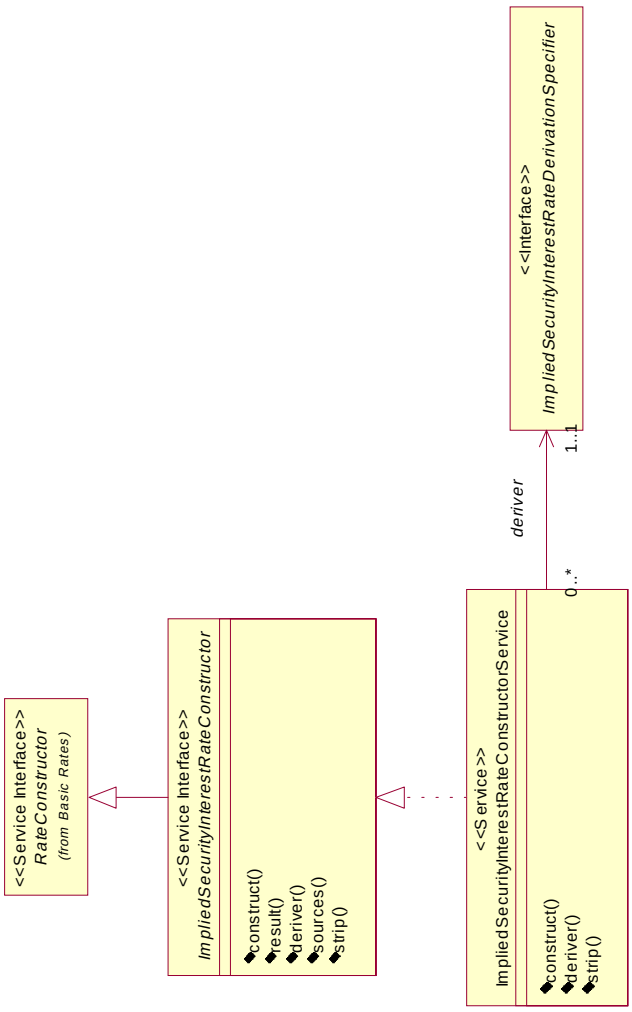


Figure 9: Class Diagram— Implied Interest Rates2

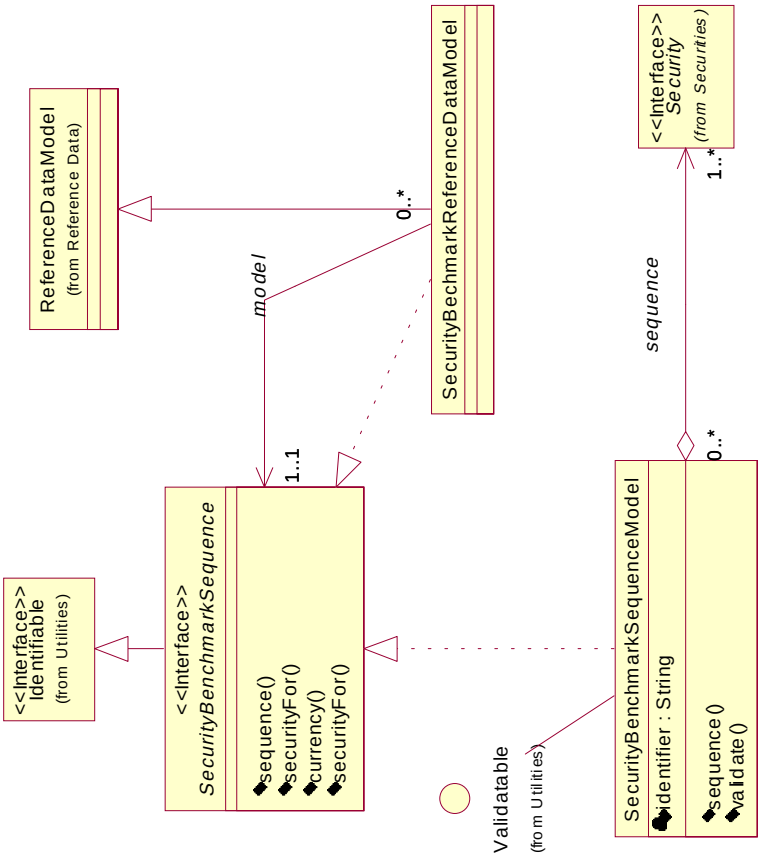


Figure 10: Class Diagram— Benchmark Sequences

References

[1] Robert Steiner. *Mastering Financial Calculations*. Pitman Publishing, 1998.