Stock Portfolio Manager

Tool to analyze how your portfolio is doing

Georges Hawi

Scenario

When investing, taking analytics of how your investment journey is going and possible future outlooks is essential. Which is why I want to build a small tool where you can regularly update the price of securities you invested in and have your portfolio value be updated automatically, and also consider possible future outlooks.

What it does

- Allows me to check my portfolio value
- Calculates profits or losses from each stock
- Forecasts possible dividend returns
- Includes profits you must add to income for taxes
- Uses CSV input

What the user can do

The user has 2 accounts (I’m limiting it to 1 user, so there's no list of users as its not useful in my use cases). The user has access to 2 csv files.

1. Initial buy prices

The user inputs the ticker, price he bought the asset, date, and (if the case) dividend %

2. Current prices

The user modifies this file when he wants a new report. He simply inputs the new price and the date and (if the case) all of the last dividend payouts since the last report.

The program then generates a report based on the inputted information

Expected output (1)

1. Per-Security info: (for each security in the portfolio) → Sorted by performance descendingly

- Ticker
- Shares held
- Cost basis (Price per share when bought)
- Current price
- Profit (absolute)
- Profit (percentage)
- Annual Return

For dividend securities:

- Dividend payout history
- Total dividends Earned
- Projected dividends

Expected output (2)

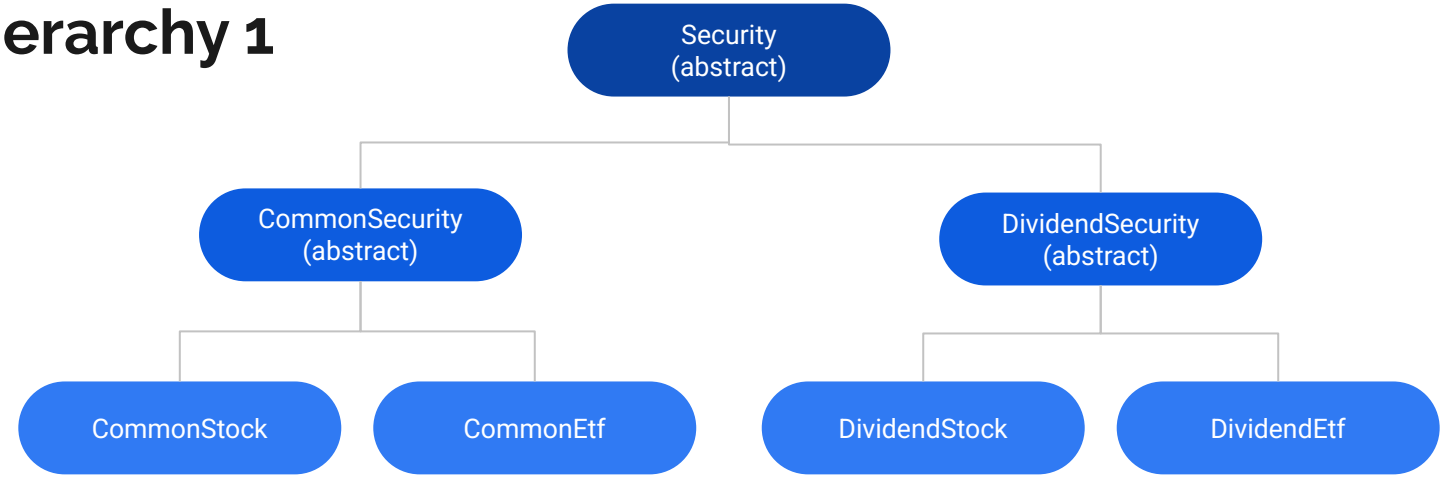
2. Account summary

- Total market value
- Total Profit (Absolute)
- Total Profit (Percentage)
- Annualized Return
- Total Dividends Earned
- Projected Dividends

For taxable account:

- Amount to add to taxable income (50% of total profits)

Hierarchy 1


Security
(abstract)
CommonSecurity
(abstract)
DividendSecurity
(abstract)
CommonStock
CommonEtf
DividendStock
DividendEtf

Methods for each class

Security superclass - abstract

- getMarketValue() → Returns current market value (shares * currentPrice)
- calculateProfit(double years, double return) → Projects profit over time using an inputted annual return

CommonSecurity extends Security- abstract

No new methods, inherits from Security

CommonStock extends CommonSecurity

No new methods, inherits from CommonSecurity

CommonEtf extends CommonSecurity

No new methods, inherits from CommonSecurity

DividendSecurity extends Security implements DividendPaying- abstract

- projectDividend(int years) → Forecast based on average historical payouts
- totalDividends(Map<LocalDate, Double> payouts) → Sums all payouts per share

DividendStock extends CommonSecurity

No new methods, inherits from CommonSecurity

DividendEtf extends CommonSecurity

No new methods, inherits from CommonSecurity

Hierarchy 2

Methods for each class

Account superclass - abstract

- getPortfolioMarketValue()
- getTotalProfit(initial value)
- getAnnualizedReturn(double years)

TaxableAccount extends Account implements TaxableIncomeCalculable

No new methods, inherits from Security

NonTaxableAccount extends Account

No new methods, inherits from CommonSecurity

Interfaces 1 -> DividendPaying

Provides a method to calculate dividends and dividends forecasts

Why it is needed:

Keeps dividend logic separate from non-dividend securities like CommonStock. Code would become wet if I needed to rewrite the methods in both dividend paying securities.

Interfaces 2 (AND TEXTIO) -> Reportable

Provides a method for exporting a report from the system to an output file. Is also the output TextIO part of the project.

Why it is needed:

Avoid wet code, since I would have to write the methods in both Account subclasses, and it would make modifying the output more annoying.

Interfaces 3 -> TaxableIncomeCalculable

Provides a method for calculating the taxable income based on profits, applicable only to taxable accounts.

Why it is needed:

In case I want to add more accounts that are taxable (say I have a wealth simple account, my regular investing account with the bank, an account on a crypto exchange, etc) then I won't have to add this method a bunch of times.

Runtime polymorphism (1)

1. writeReport(Account account, String outputFilePath)

Will have to override this method in the TaxableAccount class to add the taxable income part.

2. compareTo(T o1, T o2)

Defined in the Security class, it will be overridden in the DividendSecurity subclasses to have it compare based on dividend returned (%) AND the price increase.

TextIO

The output part was already explained in the interface reportable.

The Input part will be written in the main class, as it will be used as the input parameters for the methods in all the other classes.

I will have two input csv files:

1. Your CSV file containing a list of the Stock tickers, your original buy price, and the % dividend
2. A CSV file where you update the Prices for all the tickers and type in the dividends paid out. This file will be compared to the first one in order to generate the report on how your portfolio is doing.

Comparable

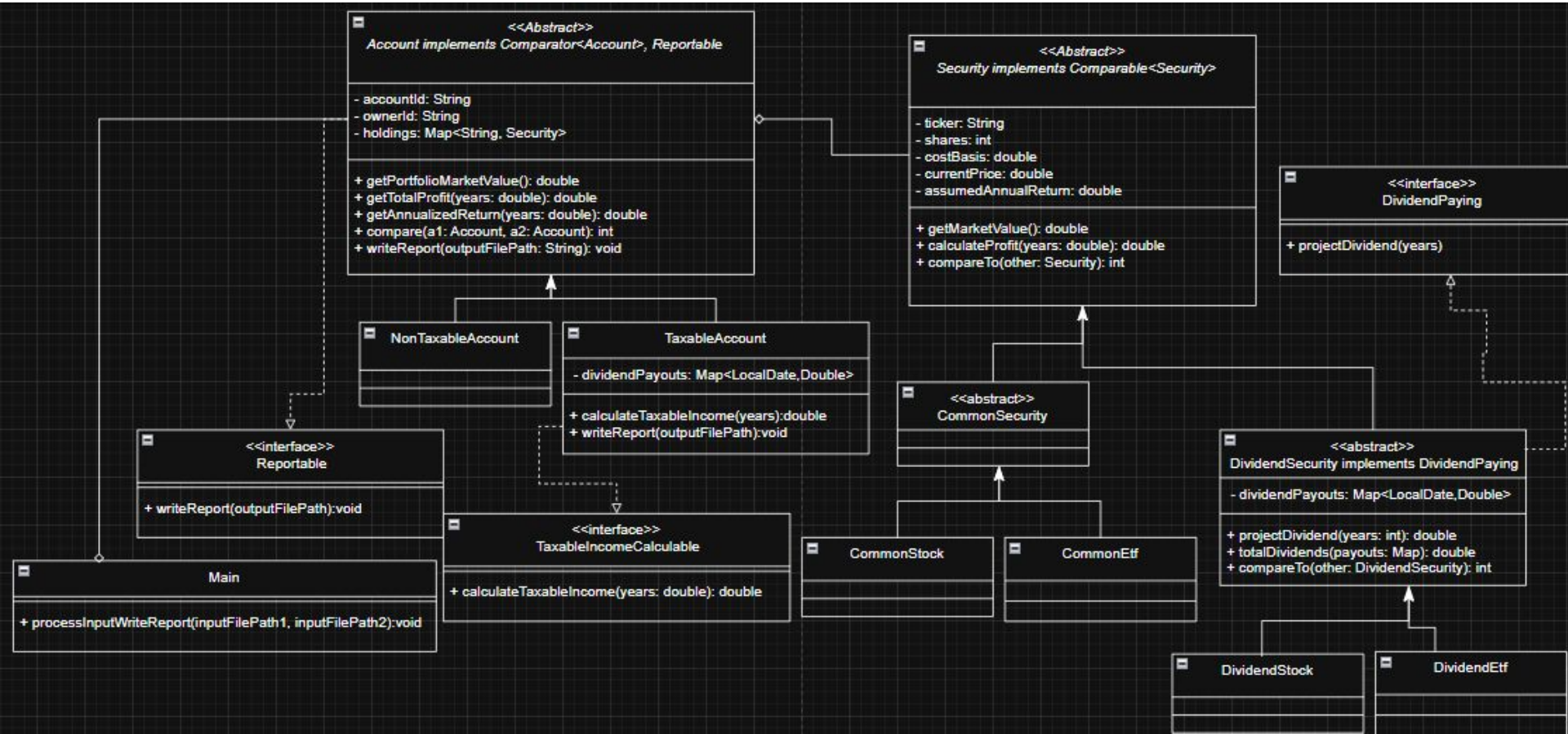
Will be used to compare the stock performances and allow for the sorting in the report. It will be implemented in the superclass Security and will sort descendingly based on % return.

The method will be overridden in the DividendSecurity subclasses to have it compare based on dividend returned (%) AND the price increase.

Comparator - static

Will be implemented in the Account superclass and will allow the two accounts be compared based on absolute return. Sorted descendingly.

Diagram


<<Abstract>>
Account implements Comparator<Account>, Reportable
- accountId: String
- ownerId: String
- holdings: Map<String, Security>
+ getPortfolioMarketValue(): double
+ getTotalProfit(years: double): double
+ getAnnualizedReturn(years: double): double
+ compare(a1: Account, a2: Account): int
+ writeReport(outputFilePath: String): void
<<Abstract>>
Security implements Comparable<Security>
- ticker: String
- shares: int
- costBasis: double
- currentPrice: double
- assumedAnnualReturn: double
+ getMarketValue(): double
+ calculateProfit(years: double): double
+ compareTo(other: Security): int
<<interface>>
DividendPaying
+ projectDividend(years)
NonTaxableAccount
TaxableAccount
- dividendPayouts: Map<LocalDate,Double>
+ calculateTaxableIncome(years):double
+ writeReport(outputFilePath):void
<<abstract>>
CommonSecurity
<<interface>>
Reportable
+ writeReport(outputFilePath):void
<<abstract>>
DividendSecurity implements DividendPaying
- dividendPayouts: Map<LocalDate,Double>
+ projectDividend(years: int): double
+ totalDividends(payouts: Map): double
+ compareTo(other: DividendSecurity): int
<<interface>>
TaxableIncomeCalculable
+ calculateTaxableIncome(years: double): double
CommonStock
CommonEtf
Main
+ processInputWriteReport(inputFilePath1, inputFilePath2):void
DividendStock
DividendEtf

Goals for deliverable 2

- Write all methods and class headers (leave body empty)
- Write all interface and method headers (leave body empty)
- Write all the fields