Foliotracker - design

W12

Features

- create and delete multiple portfolio(s)
- view positions (stock name, number of shares held, price per share, value of the holding)
- add and remove stocks to your portfolio
- buy and sell a number of shares of each stock
- stock prices that refresh as changes are made
- view total portfolio value and your profits achieved
- save portfolios locally
- benefit estimates for a particular stock
- instant indication when a stock price fluctuates
- sorting positions by value, ticker symbol, etc

Design rationale

We maintain all our quotes – current stock values – stored at a single point: the `QuoteCache`. This cache of quotes gives back the name and the value for any quote that has already been fetched, knows how to fetch quotes by ticker, and knows how to update all the already fetched quotes.

This allows us to fetch every quote exactly once per global update. It also means that, to observe any changes in quote values, it is enough to observe this cache.

The set of all portfolios is maintained within a `PortfolioManager`. This global manager also maintains a reference to the cache of quotes, and it passes this reference to newly created portfolios and portfolios loaded from disk.

When a portfolio is saved to this, this reference to the cache of quotes is discarded. When a portfolio gets loaded, all the tickers of the stock managed by the portfolio is fetched into the quote cache and the manager hands the reference of it to the portfolio.

We do not have an intermediate concept for stocks. All the information about a stock in a portfolio can be obtained from the portfolio and the ticker.

Portfolios do not store the current state of stock. Instead, they store a list of transactions per each ticker, and information is then computed by iterating that list of transactions. This supposes a performance penalty but means that we do not have to worry about handling state and that concurrent code is less likely to present any issues.

We run a background thread which periodically updates all the already fetched quotes. At the same time, we may manually fetch the quote for a particular ticker, like when we are adding stock for the first time. To prevent any concurrency issues, we make both `fetch(String ticker)` and `updateAll()` methods synchronized.

Our first approach was to try and separate the parts of our software which were involved in any IO from those parts which weren't. Then, the idea was to make this parts communicate between them at the top-most level. Java and OOP don't favour this approach, so we stopped fighting against them and blended our IO doing methods with our purely functional methods. This resulted in a design which is far more intuitive from them standard Java-world viewpoint.

Backend API specifications

QuoteCache

```
/**
 * Fetches, stores and updates values and names of stock.
 */
public interface QuoteCache {

  /**
   * requires: ticker != null
   * effects: returns null if the given ticker has not been fetched before; or
   *          returns the name for the given ticker.
   */
  public String getName(String ticker);

  /**
   * requires: ticker != null
   * effects: returns null if the given ticker has not been fetched before; or
   *          returns the value for the given ticker.
   */
  public Double getValue(String ticker);

  /**
   * requires: ticker != null
   * modifies: this
   * effects: throws IOException if there was any IO error when fetching; or
   *          throws NoSuchTickerException if the given ticker is invalid; or
   *          stores the name and value for the given ticker and signals
   *          observers.
   */
  public void fetch(String ticker) throws IOException, NoSuchTickerException;

  /**
   * modifies: this
   * effects: throws IOException if there was any IO error when fetching; or
   *          fetches all the tickers fetched previously and signals
   *          observers.
   */
  public void updateAll() throws IOException;

  /**
   * requires: observer != null
   * modifies: this
   * effects: marks an observer as to be called whenever a bulk update
   *          happens.
   */
  public void addObserver(Observer observer);

  /**
   * requires: observer != null
   * modifies: this
   * effects: forgets about calling the given observer when a bulk update
   *          happens.
   */
  public void deleteObserver(Observer observer);
}
```

StockTransaction

```
/**
 * Represents an exchange of shares at a certain price in some moment in time.
 * Once constructed, transactions are immutable.
 */
public interface StockTransaction {

  /**
   * effects: returns the number of shares bought in the transaction;
   *          a positive integer represents a purchase,
   *          a negative integer represents a sell.
   */
  public int getShares();
  /**
   * effects: returns the price of each of the shares.
   */
  public double getPrice();

  /**
   * effects: returns the date the transaction was created.
   */
  public Date getTimestamp();

  /**
   * effects: returns the total value of the transaction.
   */
  public double getValue();
}
```

Portfolio

```
/**
 * Tracks shares of stock values over time.
 * You can:
 *  - Inspect the contents of the portfolio.
 *  - Buy and sell stock.
 *  - Compute the benefit.
 *  - Get a historical of the transactions made.
 */
public interface Portfolio extends java.io.Serializable {

  /**
   * effects: returns the name of the portfolio.
   */
  public String getName();

  /**
   * effects: returns the tickers of stock that has been bought into the
   *          portfolio, even if it has been sold later.
   */
  public Set<String> getTickers();

  /**
   * effects: returns the reference to the cache of quotes.
   */
  public QuoteCache getQuoteCache();
```

```
/**
 * requires: cache != null
 * modifies: this
 * effects: throws IOException if there is any error populating the cache
 *          with the tickers in the portfolio; or
 *          populates the cache with the tickers in the portfolio and uses
 *          the cache for any future operations on quotes.
 */
public void setQuoteCache(QuoteCache cache) throws IOException;

/**
 * requires: ticker != null
 * effects: returns an empty list if the given ticker has had no
 *          transactions; or the list of transactions for the given quote
 *          ordered in time.
 */
public List<StockTransaction> getTransactions(String ticker);

/**
 * requires: ticker != null
 * effects: returns the total number of shares for the given ticker.
 */
public int getShares(String ticker);

/**
 * requires: ticker != null
 * modifies: this
 * effects: does nothing if shares == 0; or
 *          throws IOException if there was an error getting the current
 *          price of the ticker; or
 *          throws NoSuchTickerException if there exists no stock with the
 *          given ticker; or
 *          records a purchase with the current price of the given ticker
 *          and with the given shares.
 */
public void buy(String ticker, int shares)
    throws IOException, NoSuchTickerException;

/**
 * requires: ticker != null
 * modifies: this
 * effects: does nothing if shares == 0; or
 *          throws IOException if there was an error getting the current
 *          price of the ticker; or
 *          throws NoSuchTickerException if there exists no stock with the
 *          given ticker; or
 *          throws InsufficientSharesException if shares >
 *          getShares(ticker); or
 *          records a sell with the current price of the given ticker and
 *          with the given shares.
 */
public void sell(String ticker, int shares)
    throws IOException, NoSuchTickerException, InsufficientSharesException;

/**
 * requires: ticker != null
 * modifies: this
 * effects: does nothing if getShares(ticker)  == 0; or
 *          throws IOException if there was an error getting the current
 *          price of the ticker; or
 *          throws NoSuchTickerException if there exists no stock with the
 *          given ticker; or
 *          records a sell transaction with the current price for the
 *          given ticker and with all the shares of stock currently in the
 *          portfolio.
 */
public void sellAll(String ticker)
    throws IOException, NoSuchTickerException;

/**
 * requires: ticker != null
 * effects: returns the total value of all the shares of the given ticker.
 */
public double getValue(String ticker);

/**
 * effects: returns the total value of the portfolio.
 */
public double getValue();

/**
 * requires: ticker != null
 * effects: returns the total price paid for all the shares of the given
 *          ticker.
 */
public double getCost(String ticker);

/**
 * effects: returns the total price paid for all the stock in the
 *          portfolio.
 */
public double getCost();
```

```
  /**
   * requires: ticker != null
   * effects: returns the absolute profit made with the given quote; where
   *          a negative value represents loss.
   */
  public double getBenefit(String ticker);

  /**
   * effects: returns the absolute profit made with the portfolio.
   */
  public double getBenefit();
}
```

PortfolioManager

```
/**
 * Manages portfolios of stock.
 * You can add, get, remove, save and load portfolios.
 * Maintains a cache of fetched quotes.
 */
public interface PortfolioManager {
  /**
   * requires: name != null
   * effects: returns the portfolio with the given name; or
   *          null if such a portfolio has not been created.
   */
  public Portfolio get(String name);

  /**
   * requires: name != null
   * modifies: this
   * effects: returns the portfolio with the given name if it exists; or
   *          creates and returns a new portfolio with that name.
   */
  public Portfolio getOrCreate(String name);

  /**
   * effects: returns all the portfolios in the quotes.
   */
  public Set<Portfolio> getAll();

  /**
   * requires: name != null
   * modifies: this
   * effects: removes the portfolio with the given name if it exists; and
   *          returns whether a portfolio with such a name existed.
   */
  public boolean remove(String name);

  /**
   * effects: returns the reference to the cache of quotes.
   */
  public QuoteCache getQuoteCache();

  /**
   * requires: portfolio != null && path != null
   * effects: throws IOException if there is any problem writting the file; or
   *          writes the representation of the portfolio to the given path.
   */
  public void save(Portfolio portfolio, Path path) throws IOException;

  /**
   * requires: path != null
   * effects: throws IOException if there is any problem reading the file; or
   *          throws PortfolioAlreadyExistsException if a portfolio with the
   *          name of the loaded portfolio already exists; or
   *          loads the portfolio into the manager and returns it.
   */
  public Portfolio load(Path path) throws IOException, PortfolioAlreadyExistsException;

}
```