Foliotracker - Tests and assertions

W12

Java Assertions

Our `PortfolioImpl` implementation queries a `QuoteCache` interface to fetch values for tickers from external sources. In doing so, we make several assumptions:

- That after a successful `QuoteCache.fetch(ticker)`, the value obtained by `QuoteCache.getValue(ticker)` will not be `null`.
- That all the tickers listed on `PortfolioImpl.getTickers()` have their value available in the cache – `QuoteCache.getValue(ticker)` is not `null`.

The software should not be able to run into a state in which these two conditions do not hold. We use assertions to make sure of that and to explicitly express the state in which we rely on.

JUnit Testing rationale

Our backend has 5 implementation classes:

`PortfolioManagerImpl`

The purpose of this class is to manage our portfolios. This is via the following functionalities: adding, getting, removing, saving and loading of portfolios.

First we placed emphasis on testing our getters, ensuring that:

- getting a non existant folio returns null.
- getOrCreate returns a new folio when called with a non existing folio.
- ensuring getOrCreate does not overwrite an exisitng folio if called with an existing folio.
- ensuring getting all of our portfolios using getAll returns null initially (when none exist).
- ensuring getAll has the correct functionality when fetching a manager with several portfolios.

Then we tested our remove functionality, and that it would correctly remove a portfolio, given the name as a string. Along with testing the remove function in correspondance with all our getting methods tested above.

We did not unit test the `save` and `load` methods because they incur in disk IO, and we therefore understand they are better tested as part of integration tests or system tests.

`PortfolioImpl`

In this class we emphasised towards the coverage of the StockTransactionImpl.java and PortfolioImpl.java

There are tests regarding a variety of methods:

- The transaction functionality is being tested to check whether the back-end generates consistently the purchases and trades without making any invalid transaction (e.g sell more shares than he/she owns for a stock). What is more testing the ability to buy stock that doesn't exist or sell shares without buying any shares of a stock.
- The equality testing:
  - Symmetric
  - Transitive
  - Reflexive
  - Consistent
  - Null and Object
- The calculation check: This test comprises of 3 test (basic,specific and accumulative) which their main purpose is to check if the back-end carries out correctly the calculations for profit/loss that a stock generates.
- Tests regarding setting and getting Quotes which are the back-end information that are being fetched online through Yahoo(YahooQuoteCache class). But in the current testing strategy the system adopts a fictional RandomQuoteCache which generates random values for any ticker given.

`TransactionImpl`

This is a straighforward implementation with a constructor that sets the current date as a timestamp and three getters. It is just a triplet, and is used by `Portfolio`. We therefore decided not to create a test case for it.

`RandomQuoteCache`

This is an implementation whose only purpose is to provide the GUI with sample input when we are manually testing and the stock markets are closed. We therefore decided not to create a test case for it.

`YahooQuoteCache`

This implementation incurs in IO with external 3rd party services over the network for which we don't have any control over. Testing it as it is would have close to no value. Ideally, we would mock Java's `URL` API and test valid and invalid fictional responses. In practice, we decided to manually test it by running the GUI.