SECOND JAVA COURSEWORK

CASINO ROYALE

(Stock trading simulation game)

Description

I made a stock trading simulation game where the player can travel to the past and buy and sell stocks. The game is based on historical stock prices scrapped from NASDAQ.

Can you be the next Warren Buffett? Or just a typical r/wallstreetbets member? Welcome to Casino Royale!

The coursework project utilizes web scrapping, database, multi-threading and Jframe GUI.

Caching system

In order to update stock prices as the game date progresses potentially requires the processing of a lot of data. As such making a request to the DB each time a stock price needs to be updated would be unreasonably slow. Saving all stock info to RAM also would not be a great solution.

As such I have devised a caching system as the best solution. Where stock prices for a certain period of time is retrieved from the DB and stored in a cache.

The data structure of the cache is such:

```
HashMap<LocalDate, HashMap<Stock, Double>> cache
```

And is utilized such:

```
cache.get(date).forEach((stock, newPrice) -> {
            playerPortfolioValue += (newPrice - stock.price) * stock.count;
            stock.setPrice(newPrice);
        });
```

The cache is updated in two scenarios:

- **Rolling updates:** once the current game date reaches near the max cached date, a new thread is initiated to update the cache.
- **Update on bought stock:** when a stock is bought its price history is scrapped and saved to the DB. However, the cache is left outdated as it is not updated with the newly bought stock prices. So, a separate procedure is initiated to inject new price info into the live cache.

Game clock

The progress of the game is coordinated and dictated by the App.gameClock object that runs on separate thread. The gameClock can be paused and unpaused, as well as run on different speeds (x1, x5, x10).

The gameClock stores the current game date and progresses it forward and calls for stock price updates.

```
@Override
    public void run() {
        synchronized (this) {
            while (true) {
                updateDate();
                updateStocks();
                sleep(100);

                if (isPaused) {
                    try {
                        wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }

    }
```

Database

The program utilizes a PostgreSQL database to save stock info. The program interfaces with the database via the App.db object.

The database is structured as such:

Table stocks
columns:

- symbol – stock identifier tag, primary key
- company_name
- industry
- description
- history_startdate – oldest available date in price history
- history_enddate - latest available date in price history
- count – how much of this stock does the player own

Table priceHisory
columns:

- date – date of the data point
- symbol – foreign key to stock
- price – the price of the stock on the specific date

Web scraping for stocks

Stock info is scrapped from the nasdq.com website.

Specifically, the info is retrieved from the json files used by the website to populate the information on the page. The json files themselves are retrieved from their API. From what I could tell, this API is specially designed for their website and is not meant for public use – for that they offer a different API service that you need to register for.

However, I find their website API to be of better use than their official one.

One, you don't have to register and obtain an API key. Two, the website API has unlimited requests, the official one has a limit for free accounts. And three, for whatever reason the official API has a data cut off date of 2018, their website, however, displays up to date price info.

The WebScrapper class is responsible for scrapping the NASDAQ website. WebsScapper class uses jsoup a json libraries.

Simple overview of buying and selling stocks