

FINV: Biblioteca de Dados Financeiros de Bolsas de Valores

Rafael José Camargo Bekhauser¹

¹Graduando Bacharelado em Ciência da Computação - IFC Rio do Sul

rafaelcamargo.inf@gmail.com

Abstract. *The FInv library addresses the significant increase in B3 investors, adhering to SOLID principles and design patterns for efficient structuring. Data retrieval from Yahoo Finance, using the Template Method pattern, is simplified. The use of the Strategy pattern in statistical calculations and data export provides flexibility. The FInv class centralizes the application, offering a cohesive interface. Unit tests ensure robustness, covering stock retrieval, statistical calculations, and data export.*

Key-words: *Stock Exchange; SOLID; Design patterns.*

Resumo. *A biblioteca FInv atende ao aumento expressivo de investidores na B3, seguindo os princípios SOLID e design patterns para estruturação eficiente. A obtenção de dados simplificada via Yahoo Finance utiliza o padrão Template Method. O uso do padrão Strategy em cálculos estatísticos e exportação de dados proporciona flexibilidade. A classe FInv centraliza a aplicação, oferecendo uma interface coesa. Testes unitários garantem a robustez das operações, cobrindo a obtenção de ações, cálculos estatísticos e exportação de dados.*

Palavras-chave: *Bolsa de Valores; SOLID; Design Patterns.*

1. Introdução

Nos últimos anos, temos observado um expressivo aumento no número de investidores participando ativamente na Bolsa de Valores brasileira (B3). Este crescimento é notável, indicando um interesse crescente por parte dos investidores em explorar oportunidades no mercado financeiro.

Daniel Chinzarian (2022), analista de fundos de investimentos da corretora de valores XP, ao analisar dados da B3 referentes aos anos de 2019 a 2022, constatou um significativo aumento no número de investidores. Os percentuais de crescimento foram impressionantes, registrando aumentos de 310%, 182%, 132% e 127%, respectivamente, ao longo desses anos.

Com o notável aumento no número de investidores na Bolsa de Valores, torna-se imperativo o desenvolvimento de soluções de gerenciamento que possam efetivamente atender às crescentes demandas desse público diversificado.

Diante desse cenário em constante evolução, torna-se relevante considerar o desenvolvimento de uma biblioteca bem estruturada em conceitos de Orientação a Objetos voltada para o mercado de investidores na Bolsa de Valores. Esta aplicação, buscará proporcionar uma experiência eficiente, confiável e segura para os investidores, fornecendo dados para análises.

2. SOLID

Os princípios SOLID constituem um conjunto valioso de diretrizes de design para a programação orientada a objetos, proporcionando uma base sólida para o desenvolvimento de uma biblioteca financeira em Java. O acrônimo SOLID refere-se a Responsabilidade Única, Aberto/Fechado, Substituição de Liskov, Segregação de Interface e Inversão de Dependência.

Ao seguir esses princípios, os desenvolvedores têm a capacidade de criar uma aplicação que se destaca em termos de facilidade de teste, manutenção e extensibilidade. Essencialmente, a aplicação desses princípios assegura que cada classe possua uma responsabilidade única, evitando que alterações em uma classe afetem outras.

Além disso, a utilização de interfaces se torna uma prática crucial para a segregação de responsabilidades e para minimizar as dependências entre as classes. Ao adotar interfaces, é possível estabelecer contratos claros, facilitando a compreensão e a manutenção do código.

Assim, ao incorporar os princípios SOLID no desenvolvimento de uma solução, os desenvolvedores estarão aptos a construir uma solução robusta, adaptável e de fácil manutenção.

3. Design Patterns

Padrões de projeto ou Design Patterns, referem-se a soluções genéricas para desafios recorrentes que surgem durante o desenvolvimento de software. Eles representam abordagens consolidadas que os desenvolvedores podem adotar para resolver problemas comuns no design e desenvolvimento de software.

Esses padrões são como modelos que os desenvolvedores podem aplicar em diversas linguagens de programação, plataformas e arquiteturas, tornando-os ferramentas valiosas para equipes de desenvolvimento de software.

Os padrões de projeto desempenham um papel crucial no desenvolvimento de software, contribuindo para aprimorar a qualidade, confiabilidade e capacidade de manutenção dos sistemas.

Ao fornecer soluções testadas e comprovadas para desafios frequentes, esses padrões podem economizar tempo e esforço dos desenvolvedores, permitindo que eles se concentrem em aspectos mais complexos e desafiadores do desenvolvimento de software.

Além disso, os padrões de design promovem consistência e padronização em diferentes projetos de software, facilitando a colaboração e o compartilhamento de código entre os membros da equipe.

Existem três categorias principais de padrões de design: criacionais, estruturais e comportamentais. Os padrões criacionais abordam a criação de objetos e suas referências, enquanto os padrões estruturais tratam da composição de classes e objetos. Por sua vez, os padrões comportamentais concentram-se na interação e comunicação

entre objetos e classes. Cada tipo de padrão oferece diretrizes e melhores práticas específicas para resolver conjuntos específicos de desafios.

4. Desenvolvimento

A concepção da biblioteca FInv nasceu da necessidade premente de simplificar e potencializar o desenvolvimento de soluções voltadas para investimentos, oferecendo uma ampla gama de funcionalidades destinadas a tornar a análise financeira mais acessível e eficiente.

Essa iniciativa foi motivada pela ausência de opções gratuitas para obter dados de ações diretamente das bolsas de valores. A FInv surge, assim, como uma resposta a essa lacuna no fornecimento de soluções acessíveis e robustas para os desenvolvedores que buscam recursos específicos para análise e projeção de investimentos no mercado financeiro.

Conforme definido anteriormente, nosso foco concentra-se na obtenção de dados relacionados a ações, sendo assim, estabelecemos os seguintes parâmetros como sendo essenciais:

- Ticker: Identificador único que representa a ação no mercado financeiro;
- Nome da Empresa: Denominação completa da entidade comercial associada à ação;
- Moeda: A unidade monetária utilizada para a cotação da ação;
- Bolsa de Valores: Local onde a ação é listada e negociada;
- Tipo de Cotação: Indicação do tipo de cotação da ação (por exemplo, ações ordinárias, preferenciais, etc.);
- Cotação Atual: Incluindo preço atual, abertura, fechamento anterior e volume negociado;
- Cotações Históricas: Dados que representam o histórico de variação de preços ao longo do tempo;
- Eventos: Informações relevantes sobre eventos associados à ação que podem impactar seu desempenho no mercado (dividendos e divisões).

Esses parâmetros visam proporcionar uma visão abrangente e detalhada das ações, permitindo uma análise completa e precisa para os usuários da biblioteca. Para tal definimos um objeto principal denominado *Stock*, que tem por objetivo reunir as informações acima descritas, como podemos verificar na imagem abaixo.

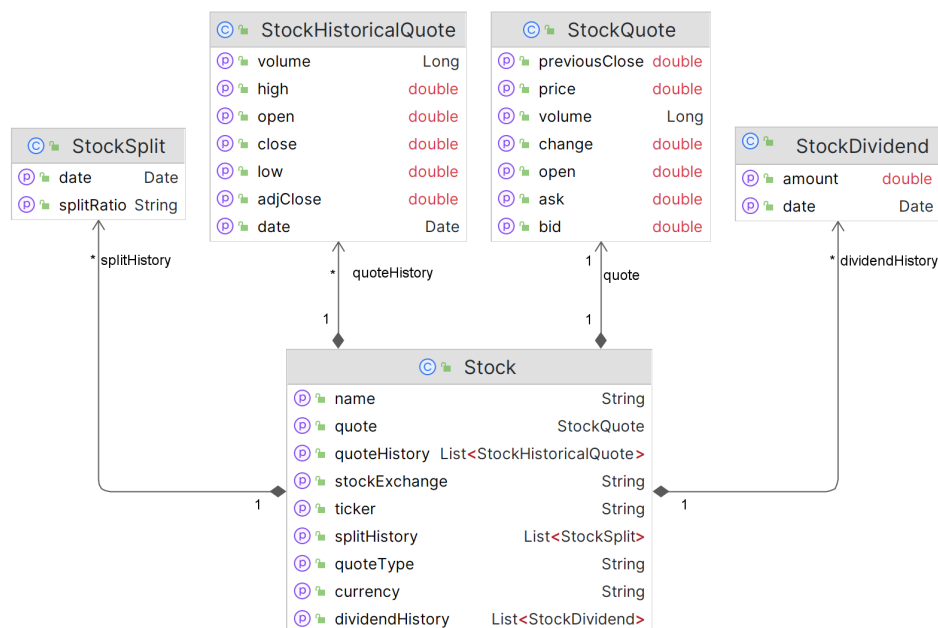


Imagem 1 – Diagrama de classe dos dados das ações.

4.1. Obtenção de dados


A obtenção dos dados na biblioteca ocorre de maneira descomplicada por meio de um processo simplificado de requisição a uma URL específica no site Yahoo Finance. Utilizando técnicas eficientes de requisição, a biblioteca realiza requests nessa URL para extrair informações detalhadas sobre as ações de bolsas de valores listadas no Yahoo Finance. Essa abordagem proporciona aos desenvolvedores o acesso direto e eficiente a dados essenciais, incluindo preço, volume, variação e dividendos.

Ao realizar a requisição na URL do Yahoo Finance, a biblioteca oferece uma solução prática e eficaz para obter informações atualizadas e confiáveis sobre as ações desejadas.

Essa integração direta com a fonte de dados, aliada à capacidade de realizar requisições em tempo real, destaca a simplicidade e eficiência da biblioteca para a obtenção precisa de dados financeiros relevantes, tanto no contexto do mercado de ações brasileiro quanto no cenário global.

Para preservar a modularização e aderir ao Princípio da Responsabilidade Única, optamos por estruturar as requisições de forma separada. Nesse sentido, definimos uma interface comum que será compartilhada por todas as classes encarregadas de realizar as chamadas de requisição.

A imagem abaixo ilustra essa abordagem, proporcionando uma arquitetura coesa e orientada à responsabilidade única, onde cada classe se concentra em uma tarefa específica, promovendo assim a manutenção facilitada e a clareza na estrutura do código.



```

package finv.provider;

import finv.Stock;

public interface StockDataProvider {

    Stock fetchData();

}

```

Imagem 2 – Interface dos provedores de dados.

Para melhor desenvolver a obtenção dos dados, utilizaremos o padrão Template Method que define a estrutura de um algoritmo em uma operação, mas permite que algumas das subclasses redefinam etapas específicas do algoritmo sem alterar sua estrutura, como ilustrado na imagem abaixo.



```

package finv.provider;

import finv.Stock;
import finv.util.RequestHttp;
import finv.util.UrlBuilder;

import java.util.Map;

public abstract class AbstractStockDataProvider implements StockDataProvider {

    protected final Stock stock;

    public AbstractStockDataProvider(Stock stock) {
        this.stock = stock;
    }

    @Override
    public Stock fetchData() {
        try {
            String url = UrlBuilder.from(getApiUrl())
                .addParameter(getRequestParameters())
                .build();

            StringBuilder response = RequestHttp.fromURL(url)
                .method("GET")
                .contentType("application/json")
                .get();

            parseApiResponse(response.toString());
            return stock;
        } catch (Exception e) {
            return null;
        }
    }

    protected abstract String getApiUrl();

    protected abstract Map<String, String> getRequestParameters();

    protected abstract void parseApiResponse(String response);
}

```

Imagem 3 – Classe abstrata para obtenção de dados.

Neste caso, a classe *AbstractStockDataProvider* define um esquema para a obtenção de dados de ações por meio de um provedor de dados, mas permite que as

subclasses concretas implementem detalhes específicos, como a construção da URL da API, a obtenção de parâmetros de solicitação e a análise da resposta da API.

O método *fetchData()* em *AbstractStockDataProvider* utiliza o padrão Template Method ao chamar métodos abstratos (*getApiUrl()*, *getRequestParameters()*, *parseApiResponse()*) que são implementados pelas subclasses concretas (*StockData* neste caso). Esses métodos abstratos são os "ganchos" que permitem que as subclasses forneçam implementações específicas para as etapas individuais do algoritmo geral, como podemos analisar no exemplo de implementação concreta abaixo.

```
package finv.provider;

import finv.Finv;
import finv.Stock;
import finv.util.LogConfig;
import org.json.JSONObject;

import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.Collections;
import java.util.Map;

public class StockData extends AbstractStockDataProvider {

    @Override
    protected String getApiUrl() {
        try {
            return Finv.STOCKS_BASE_URL + URLEncoder.encode(stock.getTicker(), "UTF-8");
        } catch (UnsupportedEncodingException e) {
            logger.severe("Error building the API URL: " + e.getMessage());
            return null;
        }
    }

    @Override
    protected Map<String, String> getRequestParameters() {
        return Collections.emptyMap();
    }

    @Override
    protected void parseApiResponse(String response) {
        try {
            JSONObject jsonResponse = new JSONObject(response);
            JSONObject optionChainObject = jsonResponse.getJSONObject("optionChain");
            JSONObject resultObject = optionChainObject.getJSONArray("result").getJSONObject(0);
            JSONObject quoteObject = resultObject.getJSONObject("quote");

            stock.setCurrency(quoteObject.getString("currency"));
            stock.setName(quoteObject.getString("longName"));
            stock.setStockExchange(quoteObject.getString("fullExchangeName"));
            stock.setQuoteType(quoteObject.getString("quoteType"));

        } catch (Exception e) {
            logger.severe("Error parsing the API response: " + e.getMessage());
        }
    }
}
```

Imagem 4 – Classe concreta de obtenção de dados de ações.

4.2. Dados estatísticos

A implementação das classes de dados estatísticos da biblioteca adota o padrão Strategy, um padrão de projeto comportamental que define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis.

Para ilustrar essa abordagem, abordaremos o desenvolvimento da classe responsável por calcular o Dividend Yield. A estratégia do *DividendYieldCalculator* é a lógica que determina como o valor de uma ação é calculado. Dentro desta estratégia, o algoritmo calcula o Dividend Yield utilizando os valores fornecidos.

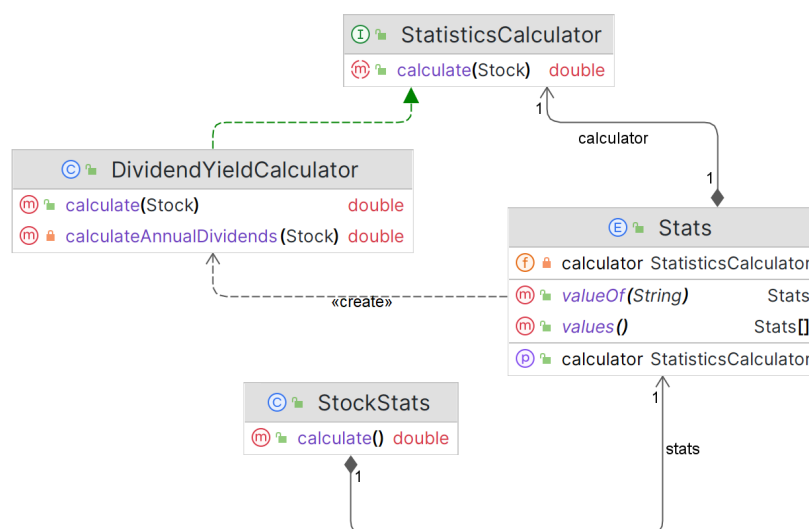


Imagem 5 – Diagrama de classe base dos cálculos estatísticos.

Ao criar o *calculador*, o código instancia a classe *StatisticsCalculator* com os valores necessários. *StatisticsCalculator* herda de *DividendYieldCalculator*, implementando a lógica de cálculo específica do Dividend Yield.

Ao criar a estratégia, o código passa uma instância da classe *StockStats* como parâmetro para o construtor. A classe *StockStats* contém as estatísticas da ação, como preço atual, preço do dia anterior, taxa de câmbio, entre outras. Essas informações são utilizadas pelo algoritmo para calcular o Dividend Yield.

O método *calculate()* da estratégia é implementado para executar a lógica de cálculo do Dividend Yield. Primeiramente, verifica-se se todos os valores necessários estão presentes na instância *StockStats*. Caso contrário, o método retorna um valor de erro.

Se todos os valores estão presentes, o método calcula o Dividend Yield utilizando a fórmula apropriada. Esta fórmula leva em consideração o valor total pago em dividendos ao longo do tempo, dividido pelo valor médio da ação durante o mesmo período.

A estratégia também inclui um método *calculateAnnualDividends()*, utilizado para calcular os dividendos anuais, obtendo essa informação da instância *StockStats*.

Em resumo, o *DividendYieldCalculator* utiliza o padrão Strategy para criar um algoritmo de cálculo flexível e reutilizável. A implementação segue as convenções do padrão Strategy, tornando-a fácil de compreender e utilizar.

4.3. Exportação dos dados

Com o intuito de viabilizar a utilização dos dados adquiridos por meio do FInv em diversas aplicações, implementamos a capacidade de exportação desses dados. Além disso, optamos por empregar o padrão Strategy também utilizado para as classes responsáveis pelos dados estatísticos, anteriormente apresentadas. Conforme podemos verificar no diagrama de classe abaixo.

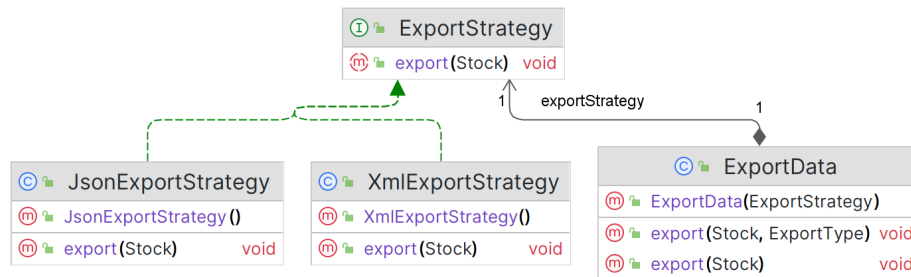


Imagem 6 – Diagrama de classe de exportação dos dados.

Para proporcionar essa flexibilidade, desenvolvemos um sistema de exportação que permite aos usuários escolher o formato desejado para os dados. Através da aplicação do padrão Strategy, encapsulamos as diferentes estratégias de exportação em classes específicas, possibilitando que tais estratégias sejam intercambiáveis.

Cada estratégia de exportação implementa a lógica necessária para gerar os dados no formato desejado, seja em JSON ou XML (ou qualquer outro formato codificado). Dessa forma, os usuários têm a liberdade de escolher a estratégia que melhor se adequa às necessidades de suas aplicações.

Essa implementação não apenas facilita a exportação dos dados para diferentes formatos, mas também proporciona uma extensibilidade notável, permitindo a inclusão de novas estratégias de exportação e cálculo estatístico sem modificar a estrutura principal do código. Assim, garantimos uma solução robusta e adaptável para as necessidades diversas dos usuários.

4.4. Centralização da aplicação

A centralização da biblioteca é evidente na classe *FInv*, que serve como ponto central para interações com diferentes partes do sistema, como podemos afirmar no diagrama de classe abaixo. Desde a recuperação de informações sobre ações até o cálculo de estatísticas e a exportação de dados, a classe oferece uma interface coesa para os usuários.

| © Finv | | |
|--------|---|--------------|
| Ⓜ | Finv() | |
| Ⓜ | export(Stock, ExportType) | void |
| Ⓜ | formatDate(LocalDate) | String |
| Ⓜ | get(String, List<Event>, String, String, Frequency) | Stock |
| Ⓜ | get(String, Event, String, String) | Stock |
| Ⓜ | get(String, Event, Frequency) | Stock |
| Ⓜ | get(String, List<Event>, String, String) | Stock |
| Ⓜ | get(String, Event) | Stock |
| Ⓜ | get(String, List<Event>, Frequency) | Stock |
| Ⓜ | get(String, List<Event>) | Stock |
| Ⓜ | get(String) | Stock |
| Ⓜ | getDividends(Stock, String, String, Frequency) | void |
| Ⓜ | getHistory(Stock, String, String, Frequency) | void |
| Ⓜ | getQuotes(Stock) | Stock |
| Ⓜ | getStock(String) | Stock |
| Ⓜ | getStockSplits(Stock, String, String, Frequency) | void |
| Ⓜ | stats(Stock, Stats) | double |
| Ⓜ | stats(Stock, List<Stats>) | List<Double> |

Imagem 7 – Diagrama de classe centralizadora.

Ao encapsular funcionalidades em métodos intuitivos, como *get()*, *stats()*, e *export()*, a biblioteca promove uma experiência de usuário clara e simplificada. A estrutura modular da classe *Finv* permite que os desenvolvedores utilizem e estendam facilmente suas funcionalidades, mantendo a coesão e a flexibilidade essenciais para um sistema financeiro robusto.

Um dos aspectos distintivos da classe *Finv* que contribui para a centralização eficiente da biblioteca é a aplicação cuidadosa da sobrecarga de métodos no método *get()*.

A sobrecarga de métodos no *get()* permite aos desenvolvedores escolher a quantidade de parâmetros que desejam fornecer, simplificando a utilização da biblioteca. Desde consultas básicas até consultas mais complexas, a sobrecarga de métodos capacita os usuários a personalizar suas solicitações conforme necessário.

Por exemplo, a chamada básica *get(String ticker)* permite recuperar informações essenciais sobre uma ação usando apenas o símbolo da ação. À medida que mais parâmetros são adicionados, como eventos específicos, frequência dos dados ou intervalo de datas, a mesma interface *get()* pode ser utilizada de maneira consistente.

Por outro lado, as chamadas *get(String ticker, Event event)* ou *get(String ticker, Event event, Frequency frequency)* adicionam camadas de complexidade de maneira gradual, permitindo que os usuários se aprofundem nas funcionalidades da biblioteca à medida que desenvolvem suas habilidades.

4.4. Testes unitários

Os testes unitários desempenham um papel crucial no desenvolvimento de aplicações, garantindo a robustez e a confiabilidade de suas funcionalidades. Nesta seção, abordaremos os principais casos de teste implementados para as diversas operações oferecidas pela biblioteca (tais quais podem ser conferidos no código fonte da biblioteca).

4.4.1. *testGetStock*

Este teste verifica se é possível obter informações básicas de uma ação (stock) utilizando o método `Finv.get`. Ele assegura que o objeto retornado não seja nulo e que o ticker da ação corresponda ao esperado.

4.4.2. *testGetStockWithEvents*

Neste caso de teste, é validado o funcionamento do método `Finv.get` ao fornecer eventos específicos para uma ação. Além de verificar se o objeto retornado e o ticker são consistentes, o teste garante que o histórico de cotações não seja nulo.

4.4.3. *testStatsAverageClosePrice* e *testStatsDividendYield*

Esses testes focam nas estatísticas individuais, avaliando se a biblioteca calcula corretamente o preço médio de fechamento e o rendimento de dividendos. Os resultados esperados são comparados com os valores calculados, com uma margem de erro de 0.01.

4.4.4. *testStatsList*

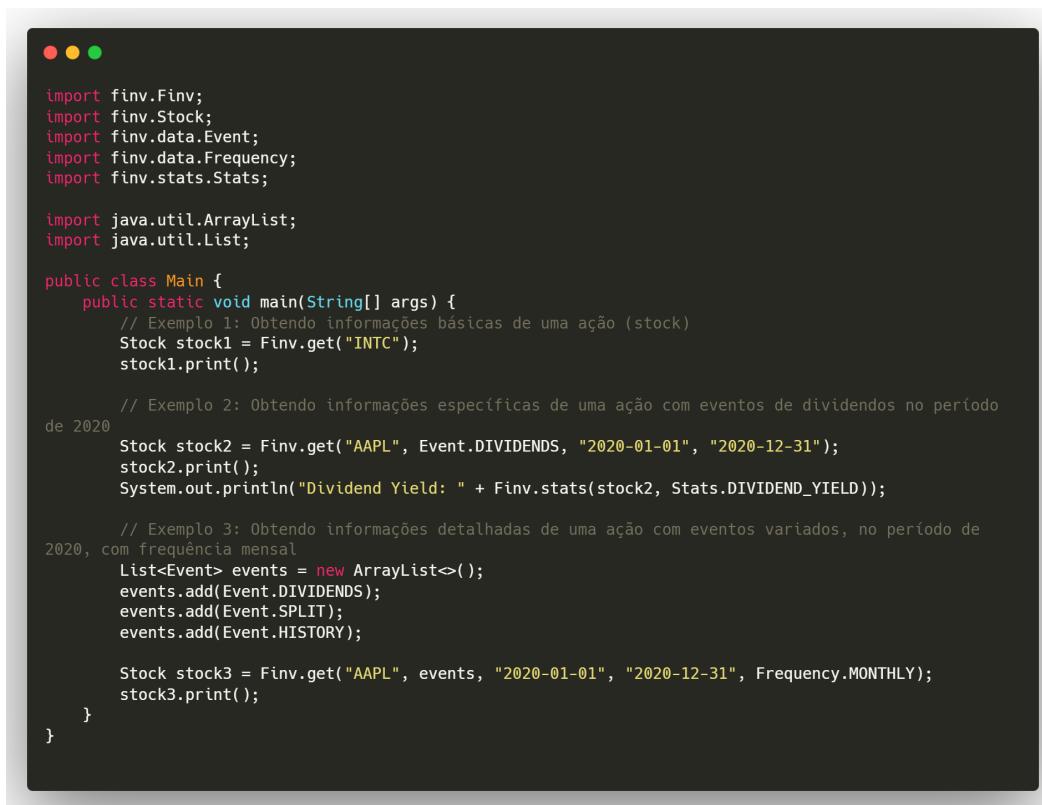
Este teste verifica se é possível obter uma lista de estatísticas para uma ação específica. Ele confirma a integridade da lista de resultados e permite adicionar asserções específicas para cada estatística incluída.

4.4.5. *testExportStock* e *testExportNullStock*

Os testes de exportação validam se a biblioteca pode exportar dados de ações para formatos específicos, como JSON. O primeiro teste examina a exportação de uma ação válida, enquanto o segundo lida com a exportação de um objeto nulo, garantindo que a biblioteca trate esse cenário de maneira adequada.

4.5. Exemplo de utilização

Conforme observado durante o desenvolvimento da biblioteca FInv, foram identificados diversos caminhos que os usuários podem seguir para utilizar suas funcionalidades. Para ilustrar um cenário básico de utilização, apresentamos a seguir um exemplo de aplicação por meio de um main.



```
import finv.Finv;
import finv.Stock;
import finv.data.Event;
import finv.data.Frequency;
import finv.stats.Stats;

import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        // Exemplo 1: Obtendo informações básicas de uma ação (stock)
        Stock stock1 = Finv.get("INTC");
        stock1.print();

        // Exemplo 2: Obtendo informações específicas de uma ação com eventos de dividendos no período
        // de 2020
        Stock stock2 = Finv.get("AAPL", Event.DIVIDENDS, "2020-01-01", "2020-12-31");
        stock2.print();
        System.out.println("Dividend Yield: " + Finv.stats(stock2, Stats.DIVIDEND_YIELD));

        // Exemplo 3: Obtendo informações detalhadas de uma ação com eventos variados, no período de
        // 2020, com frequência mensal
        List<Event> events = new ArrayList<>();
        events.add(Event.DIVIDENDS);
        events.add(Event.SPLIT);
        events.add(Event.HISTORY);

        Stock stock3 = Finv.get("AAPL", events, "2020-01-01", "2020-12-31", Frequency.MONTHLY);
        stock3.print();
    }
}
```

Imagem 8 – Exemplo de utilização do FInv.

Neste exemplo, o código demonstra três situações distintas de uso da biblioteca FInv, abrangendo desde a obtenção de informações básicas de uma ação até consultas mais específicas, considerando eventos como dividendos e operações de split, além da possibilidade de definir a frequência desejada para os dados. Essa abordagem proporciona uma visão abrangente das capacidades oferecidas pela biblioteca aos usuários finais.

5. Considerações finais

A biblioteca FInv representa uma resposta robusta à crescente demanda por soluções de gerenciamento no mercado financeiro, impulsionada pelo expressivo aumento no número de investidores na Bolsa de Valores brasileira. Desenvolvida com base nos sólidos princípios SOLID e na aplicação inteligente de Design Patterns, a FInv

oferece uma estrutura coesa e flexível para atender às diversas necessidades dos investidores.

O desenvolvimento da FInv abrange desde a obtenção eficiente de dados até a análise estatística e exportação, proporcionando uma experiência de usuário simplificada e flexível. A centralização na classe FInv, aliada à sobrecarga de métodos, oferece aos desenvolvedores uma interface intuitiva para interações variadas, desde consultas básicas até operações mais complexas.

Em suma, a FInv não apenas preenche uma lacuna no fornecimento de soluções acessíveis e robustas para análise financeira, mas também estabelece um padrão elevado para o desenvolvimento de bibliotecas financeiras, combinando elegância arquitetônica com funcionalidade prática. Essa iniciativa não apenas simplifica o desenvolvimento de aplicações financeiras, mas também contribui significativamente para a evolução do ecossistema financeiro digital.

Referências

BARRETO, José Victor Souza. **Fundos de investimento imobiliário no Brasil: as características que explicam o desempenho**. 2016. Tese de Doutorado.

CHINZARIAN, Daniel. **Análise do crescimento de investidores na Bolsa de Valores brasileira (B3)**. XP Investimentos 2022.

FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, 2002.

GAMMA, Erich et al. *Elements of Reusable Object-Oriented Software*. **Design Patterns**, 1995.

Martin, R. C. **Agile Software Development: Principles, Patterns, and Practices**. Prentice Hall, 2003.

YAHOO FINANCE. 2023. Disponível em: <<https://finance.yahoo.com/>>