



UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

LEONARDO FACHETTI JOVÊNCIO

**IMPLEMENTAÇÃO DE UM SISTEMA DE GERENCIAMENTO DE  
INVESTIMENTOS EM C++**

Rio de Janeiro

2023

## 1 INTRODUÇÃO

O presente trabalho tem como objetivo realizar a implementação de um sistema de gerenciamento de investimentos escrito na linguagem de programação c++. Para tal, o sistema oferece ao usuário diversas funções comuns características de um sistema de investimentos, como vender ações, comprar ações, exibir investimentos realizados, dentre outros.

As informações referentes as ações da bolsa de investimentos são recebidas por funções externas ao sistema implementado. Essas funções foram desenvolvidas na linguagem de programação python, dado que esta linguagem oferece diversas ferramentas que facilitam o trabalho com dados do mercado financeiro. Tais funções são responsáveis por buscar as informações mais atualizadas no momento e retornar esses dados para o sistema.

Nas seções seguintes será detalhado como esse processo é realizado, bem como foi implementado o sistema proposto.

## 2 IMPLEMENTAÇÃO DO SISTEMA

O sistema oferece ao usuário um menu principal interativo com 6 funcionalidades, sendo elas:

Opção	Descrição	Saída esperada do sistema
1	Exibir carteira de investimentos	Exibi as informações da carteira do usuário: patrimônio, valor investido, saldo da conta, rentabilidade, ativos adquiridos, quantidade, cotação atual e valor total por ação.
2	Comprar ações	Exibi uma relação de ações disponíveis e solicita ao usuário a ação e a quantidade desejada a ser comprada.
3	Vender ações	Exibi a relação de ações adquiridas e solicita ao usuário a ação e a quantidade desejada a ser vendida.
4	Exibir dados do IPCA	Exibi o valor do IPCA atual e no período acumulado do ano vigente.
5	Exibir dados do dólar	Exibi a cotação do dólar no momento atual.
6	Realizar análise fundamentalista	Exibi uma relação de empresas disponíveis, solicita ao usuário a desejada e exibi diversas métricas financeiras importantes sobre ela.

**Tabela 1 – Funcionalidades do menu do sistema proposto**

Para gerenciar a carteira de investimentos do usuário, foi implementado uma *classe* denominada **Carteira**. Essa *classe* é responsável por armazenar os dados da carteira do usuário, como nome do usuário, saldo atual, ações compradas, além de oferecer vários métodos públicos úteis para implementação do sistema. Além disso, para gerenciar as ações da bolsa, foi implementado uma *classe* denominada **Acao**. Essa classe é responsável por

armazenar as informações da ação, como ticker, valor de abertura, cotação atual, variação, dentre outros. Por fim, para gerenciar as informações das empresas, foi implementado uma *struct* denominada **Empresa**. Tanto as *classes* Carteira e Acao quanto a *struct* Empresa foram implementadas na linguagem de programação c++. Nas imagens abaixo é descrito suas implementações.

```
class Carteira
{
    friend ostream &operator<< (ostream &output, Carteira &minhaCarteira);

public:
    Carteira (string nomeUsuario="default", float saldoInicial=0);
    int iniciarSistema (vector<string> &listaAcoes, vector<Acao> &dadosAcoes);
    int atualizarCotacoes (vector<string> &listaAcoes, vector<Acao> &dadosAcoes, int modo);
    double getSaldo (void);
    Acao getAcao (int idAcao);
    int totalAcoesCarteira (void);
    int operator+ (Acao &novaAcao);
    void operator- (Acao &acao);
    void exibirIPCA (int dia, int mes, int ano);
    void exibirDolar (void);
    int analiseFundamentalista (string empresa);

private:
    string usuario;                // nome do usuário
    double saldo;                  // saldo atual da conta
    double investimentos;          // valor atual investido
    float rentabilidade;           // rentabilidade da carteira
    vector<Acao> acoesUsuario;     // ações compradas pelo usuário

    int CarregarDadosUsuario (vector<string> listaAcoes, vector<Acao> &dadosAcoes);
    int CarregarListaAcoes (vector<string> &listaAcoes);
    void computarInvestimento ();
    int dbCotacoes (string acao, double &abertura, double &cotacao);
    int dbEmpresa (string empresa, Empresa &dadosEmpresa);
    int dbDolar (double &atual);
    int dbIPCA (float &atual, float &acumulado, int dia, int mes, int ano);
};
```

**Figura 1 – Implementação da classe Carteira – arquivo carteira.h**

```

struct Empresa
{
    double margemLucro;
    double margemOperacional;
    double margemEBITDA;
    double margemBruta;
    double crescimentoReceita;
    double retornoSobreAtivos;
    double receitaPorAcao;
};

```

Figura 2 – Implementação da struct Empresa – arquivo empresa.h

```

class Acao
{
    friend ostream &operator<< (ostream &output, vector<Acao> &acoes);

public:
    Acao (string descricao = "default",
          double open = 0,
          double atual = 0,
          int qtd = 0,
          double compra = 0,
          float var = 0);

    void setTicker (string acao);
    void setValorAbertura (double abertura);
    void setValorAtual (double atual);
    void setQuantidade (int qtd);
    void setValorCompra (double compra);
    void setVariacao (float var);
    string getTicker (void);
    double getValorAbertura (void);
    double getValorCompra (void);
    double getValorAtual (void);
    int getQuantidade (void);
    float getVariacao (void);

private:
    string ticker;
    double valorAbertura;
    double valorAtual;
    int quantidade;
    double valorCompra;
    float variacao;
};

```

Figura 3 – Implementação da classe Acao – arquivo acao.h

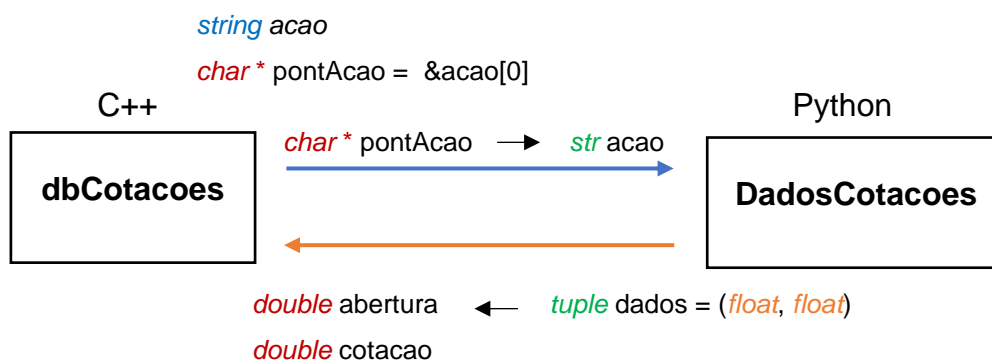
Os métodos privados **dbCotacoes**, **dbEmpresa**, **dbDolar** e **dbIPA** da *classe* *Carteira* são responsáveis por obterem as informações mais atuais disponíveis no momento sobre as cotações das ações, métricas financeiras das empresas, cotação do dólar e dados do IPCA, respectivamente. Esses métodos realizam uma interação com as funções externas ao sistema implementadas na linguagem de programação python. Essas funções acessam à internet, buscam as informações mais atualizadas no momento e retornam esses dados para os métodos privados citados.

As funções externas são compostas por um grupo de 4 funções, sendo elas:

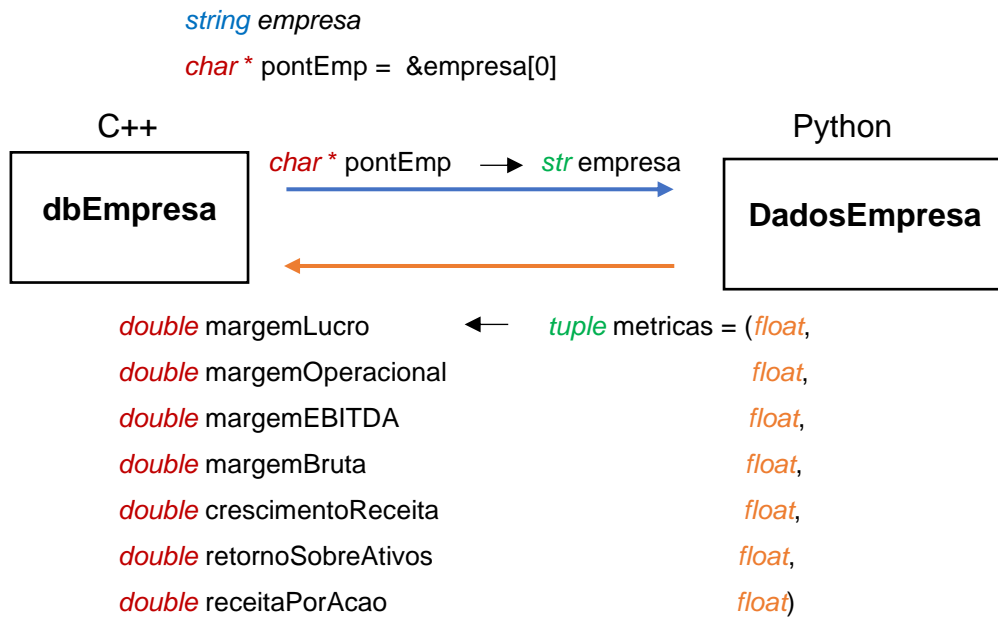
	Funções	Ação esperada
1	DadosCotacoes	Retornar as cotações mais atuais de uma ação
2	DadosEmpresa	Retornar as métricas financeiras mais atuais de uma empresa
3	DadosDolar	Retornar à cotação mais atual do dólar
4	DadosIPCA	Retornar os dados mais atuais do IPCA

**Tabela 2 – Funções externas e suas ações esperadas – arquivo funções\_externas.py**

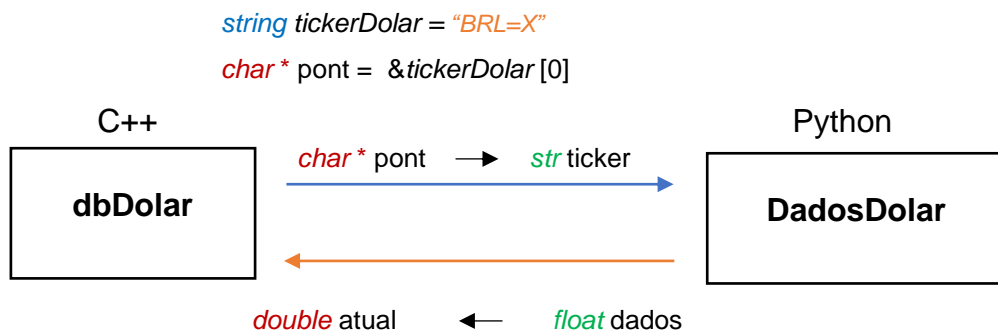
Para realizar essas ações esperadas, as funções externas fazem o uso da API [yfinance](#). Essa API oferece uma série de ferramentas úteis para trabalhar com o mercado financeiro, obtendo informações diretas da base de dados do [Yahoo Finance](#). Nas figuras abaixo é descrito um diagrama da interação entre os métodos da classe *Carteira* citados e essas funções externas em termos dos parâmetros recebidos e valores de retornos.



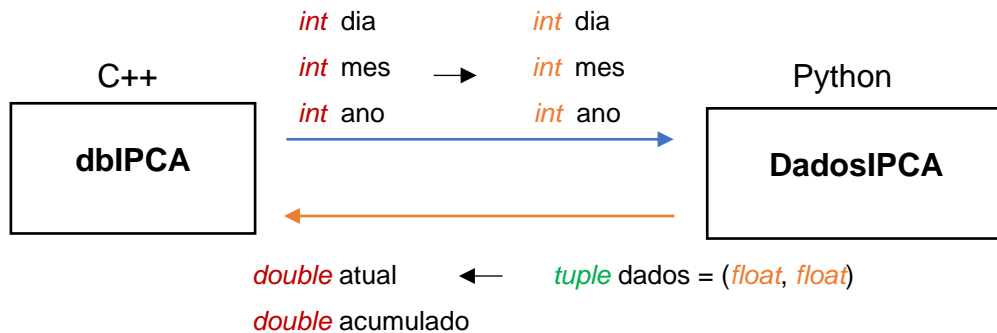
**Figura 4 – Interação entre o método privado dbCotacoes da classe Carteira e a função externa DadosCotacoes**



**Figura 5 – Interação entre o método privado dbEmpresa da classe Carteira e a função externa DadosEmpresa**



**Figura 6 – Interação entre o método privado dbDolar da classe Carteira e a função externa DadosDolar**



**Figura 7 – Interação entre o método privado dbIPCA da classe Carteira e a função externa DadosIPCA**

Essa interação é realizada utilizando a técnica [Embedding](#) da linguagem, isto é, por meio da **incorporação** de uma outra linguagem de programação na aplicação desenvolvida. Dessa forma, uma aplicação desenvolvida em C++ pode executar instruções desenvolvidas em python, por exemplo, bastando apenas chamar um interpretador python no bloco de execução em questão.

Para gerenciar a incorporação da linguagem python, foi utilizado a [API](#) oficial disponibilizada pela linguagem, geralmente chamada de API Python/C. Para utilizarmos as funcionalidades disponibilizada por essa API, basta incluirmos o cabeçalho *Python.h* no nosso código. Como exemplo de implementação, tomemos o caso do método privado **dbCotacoes** da classe Carteira. Os demais métodos privados **dbEmpresa**, **dbDolar** e **dbIPA** da classe Carteira apresentam implementação semelhante.

Para implementação, foi utilizando algumas funções fundamentais da API Python/C e algumas funções responsáveis pela criação/conversão de objetos. É importante salientar que tanto funções quanto tipos de dados na linguagem python são representados por instâncias da classe ***PyObjects***. Na imagem a seguir é mostrado a inicialização do processo de incorporação da linguagem python no sistema desenvolvido.



```

405 int Carteira::dbCotacoes (string acao, double &abertura, double &cotacao)
406 {
407     PyObject *file, *function, *args, *resul;
408
409     // Inicializa o acesso ao interpretador python
410     Py_Initialize();
411
412     // Localiza o endereço atual de trabalho
413     PyObject *sys_path = PySys_GetObject("path");
414     PyList_Append(sys_path, PyUnicode_FromString(LOCAL_WORK));
415
416     // Importa o código escrito em python
417     file = PyImport_ImportModule("funcoes_externas");

```

Objetos da classe PyObject

Referência ao arquivo funções\_externas.py

**Figura 8 - Inicialização do interpretador e importação das funções externas python**

Caso a importação do arquivo contendo as funções externas seja realizada com sucesso, é feito o acesso à função externa python informada, seguido pela criação dos parâmetros a serem recebidos por ela e, por fim, a realização da chamada da função em questão.

```

419 if (file)
420 {
421     // Acessa a função externa em python descrita no segundo argumento
422     function = PyObject_GetAttrString(file, "DadosCotacoes");
423
424     // Avalia se a função python informada pode ser chamada
425     if (function && PyCallable_Check(function))
426     {
427         // Cria uma tupla para guardar os argumentos da função
428         args = PyTuple_New(1);
429
430         char *pontAcao;
431         pontAcao = &acao[0];
432         PyTuple_SetItem(args, 0, PyUnicode_FromString(pontAcao));
433
434         // Executa a função python informada e guarda seu retorno em resul
435         resul = PyObject_CallObject(function, args);
436         Py_DECREF(args);
437         Py_DECREF(function);
438         Py_DECREF(file);

```

Referência a função externa **DadosCotacoes**

Defini o elemento da posição 0 do objeto python tupla **args** como o objeto python obtido a partir da conversão do objeto C++ string, apontado pelo ponteiro **pontAcao**

É criado um objeto python *tupla* para armazenar o argumento a ser recebido pela função **DadosCotacoes** – 1 argumento do tipo python *str*

Realiza a chamada da função externa **DadosCotacoes**, passando o objeto python tupla **args** contendo o argumento da função e guarda seu retorno no objeto python **resul**

**Figura 9 – Chamada da função externa DadosCotacoes utilizando a incorporação do python**

Por fim, é realizada a conversão do objeto python recebido como retorno da chamada da função externa em um tipo válido do C++.

```
440 // Avalia o valor de retorno
441 if (resul)
{
    // Converte o resultado recebido para um tipo c++ válido
    abertura = PyFloat_AsDouble (PyTuple_GetItem (resul, 0));
    cotacao = PyFloat_AsDouble (PyTuple_GetItem (resul, 1));
    return SUCESSO;
}
```

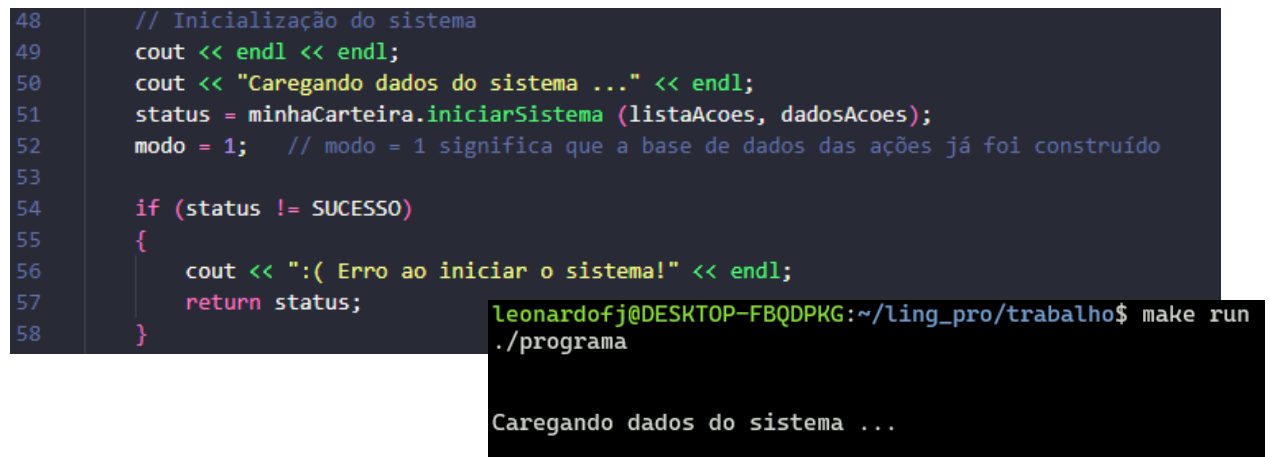
O objeto python **resul** guarda o objeto python *tupla* retornado pela função **DadosCotacoes**. Após acessarmos o tipo python *float*, é realizado a conversão para um tipo C++ válido, nesse caso, para *double*.

A função **PyTuple\_GetItem** acessa o elemento do objeto python *tupla* **resul** informado no segundo argumento da função. Nesse caso, esse elemento é do tipo python *float*.

**Figura 10 – Conversão do retorno da função python para um tipo C++ válido**

### 3 FUNCIONAMENTO DO SISTEMA

Ao iniciar o sistema, é realizado o carregamento dos dados do sistema. Esse carregamento é realizado pela chamada do método público da *classe* **Carteira** **iniciarSistema**.



```

48 // Inicialização do sistema
49 cout << endl << endl;
50 cout << "Carregando dados do sistema ..." << endl;
51 status = minhaCarteira.iniciarSistema (listaAcoes, dadosAcoes);
52 modo = 1; // modo = 1 significa que a base de dados das ações já foi construído
53
54 if (status != SUCESSO)
55 {
56     cout << ":( Erro ao iniciar o sistema!" << endl;
57     return status;
58 }

```

leonardofj@DESKTOP-FBQDPKG:~/ling\_pro/trabalho\$ make run  
./programa

Carregando dados do sistema ...

Figura 11 – Inicialização do sistema – função main

O método **iniciarSistema** realiza as seguintes ações:

1. Carrega uma lista de ações a serem disponibilizadas pelo sistema. Essa lista de ações está salva no arquivo *tickers.txt* disponível no diretório do sistema.
2. Realiza a atualização das cotações das ações com base nas informações mais atuais
3. Carrega os dados do usuário salvos em uma execução anterior do sistema
4. Computa os rendimentos da carteira com base nos valores atualizados das cotações

Ao chamar o método **iniciarSistema**, é passado como argumento por referência o *vector* de *string* **listaAcoes** e um *vector* de objetos da *classe* **Acao** **dadosAcoes**. Como retorno, é recebido os nomes das ações e os objetos da *classe* **Acao** compondo a base de dados das ações a serem disponibilizada pelo sistema, respectivamente. Além disso, a carteira do usuário é gerenciada pelo objeto instanciado da *classe* **Carteira** chamado **minhaCarteira**. Essa base é salva nas seguintes variáveis no programa **main**:

```
vector<string> listaAcoes    // Guarda o nome das ações e seus ids
vector<Acao> dadosAcoes    // Guarda todas informações da ação
Carteira minhaCarteira;    // Gerencia a carteira do usuário
```

Na tela inicial, é exibido o menu principal do sistema, conforme imagem a seguir.

```
-----
                        MENU
-----
[0] - Encerrar o programa
[1] - Exibir carteira de investimentos
[2] - Comprar ações
[3] - Vender ações
[4] - Exibir dados do IPCA
[5] - Exibir dados do dólar
[6] - Realizar análise fundamentalista
-----
Escolha uma opção do menu: █
```

Figura 12 – Menu principal do sistema

Ao selecionar a opção 1, será exibido as informações da carteira do usuário. Essas informações são exibidas utilizando o **operador** << sobrecarregado globalmente. Na imagem abaixo é mostrado o exemplo de saída do programa.

```
Meu patrimônio: R$ 10400.10
Meus investimentos: R$ 10200.10
Saldo da conta: R$ 200.00

Rentabilidade: 21.01 %

TOTAL AÇÕES: - 4 ações
```

ID	ACAO	QTD	COTAÇÃO (R\$)	TOTAL (R\$)
[00]	PETR4.SA	80	34.41	2752.80
[01]	VALE3.SA	50	72.75	3637.50
[02]	EMBR3.SA	40	23.11	924.40
[03]	ITUB4.SA	90	32.06	2885.40

```
else if (opcaoMenu == "1")
{
    cout << minhaCarteira;
}
```

Figura 13 – Saída do sistema na opção 1 do menu principal

Na opção 2 do sistema, é oferecido ao usuário a função de compra de ações. Em primeiro momento, é exibido uma relação de ações disponíveis e o usuário deve informar o id da ação desejada além da quantidade. Por fim, é realizada a compra da ação informada a partir do **operador** + sobrecarregado. O operador retorna -1 caso o saldo da conta seja insuficiente. Na imagem abaixo é mostrado o exemplo de saída do programa.

```
Escolha uma opção do menu: 2
```

Saldo da conta: R\$200.00

ID	ACAO	COTAÇÃO (R\$)	VARIACAO DIA (%)
[00]	PETR4.SA	34.41	2.08
[01]	VALE3.SA	72.75	-0.48
[02]	EMBR3.SA	23.11	-1.28
[03]	ITUB4.SA	32.06	1.68
[04]	BBDC4.SA	16.53	1.85
[05]	BBAS3.SA	54.43	0.98
[06]	ABEV3.SA	14.35	0.49
[07]	PETR3.SA	36.67	2.72
[08]	B3SA3.SA	13.59	2.26
[09]	ITSA4.SA	9.85	1.23
[10]	MGLU3.SA	2.18	-4.39
[11]	BHIA3.SA	0.51	-3.77
[12]	LREN3.SA	16.50	-1.26
[13]	PRI03.SA	44.76	4.09
[14]	VBBR3.SA	22.23	1.00
[15]	AMER3.SA	0.92	-2.13
[16]	MRFG3.SA	9.50	1.93
[17]	TIMS3.SA	17.93	1.13
[18]	WEGE3.SA	34.79	-1.61
[19]	ELET3.SA	41.00	-0.97

ID ação: █

Figura 14 – Exibição das ações disponíveis no sistema – saída da opção 2 do menu principal

```
ID ação: 9
Quantidade: 8
int statusCompra = minhaCarteira + dadosAcoes[id];

Compra realizada com sucesso!

RESUMO DA COMPRA:

AÇÃO: ITSA4.SA | QTD: 8 | COTAÇÃO: R$9.85 | VALOR INVESTIDO: R$ 78.80

Deseja continuar comprando?
[0] - sim
[1] - não

Opção: █
```

Figura 15 – Realização da compra de ação – saída da opção 2 do menu principal

Na opção 3 do sistema, é oferecido ao usuário a função de venda de ações. Essa funcionalidade é implementada a partir da sobrecarga do **operador -**. Nessa opção, é exibido uma relação de ações adquiridas pelo usuário e o usuário deve informar o id da ação deseja além da quantidade a ser vendida. Na imagem abaixo é mostrado o exemplo de saída do programa.

```
Escolha uma opção do menu: 3

-----
Meu patrimônio: R$ 10400.10
Meus investimentos: R$ 10200.10
Saldo da conta: R$ 200.00

Rentabilidade: 21.01 %

TOTAL AÇÕES: - 4 ações
-----


| ID   | ACAO     | QTD | COTAÇÃO (R\$) | TOTAL (R\$) |
|------|----------|-----|---------------|-------------|
| [00] | PETR4.SA | 80  | 34.41         | 2752.80     |
| [01] | VALE3.SA | 50  | 72.75         | 3637.50     |
| [02] | EMBR3.SA | 40  | 23.11         | 924.40      |
| [03] | ITUB4.SA | 90  | 32.06         | 2885.40     |


-----
Id ação: █
```

Figura 16 – Exibi as ações da carteira – saída da opção 3 do menu principal

```
Id ação: 2
Quantidade: 30
minhaCarteira - acaoVendida;

-----
Venda realizada com sucesso!

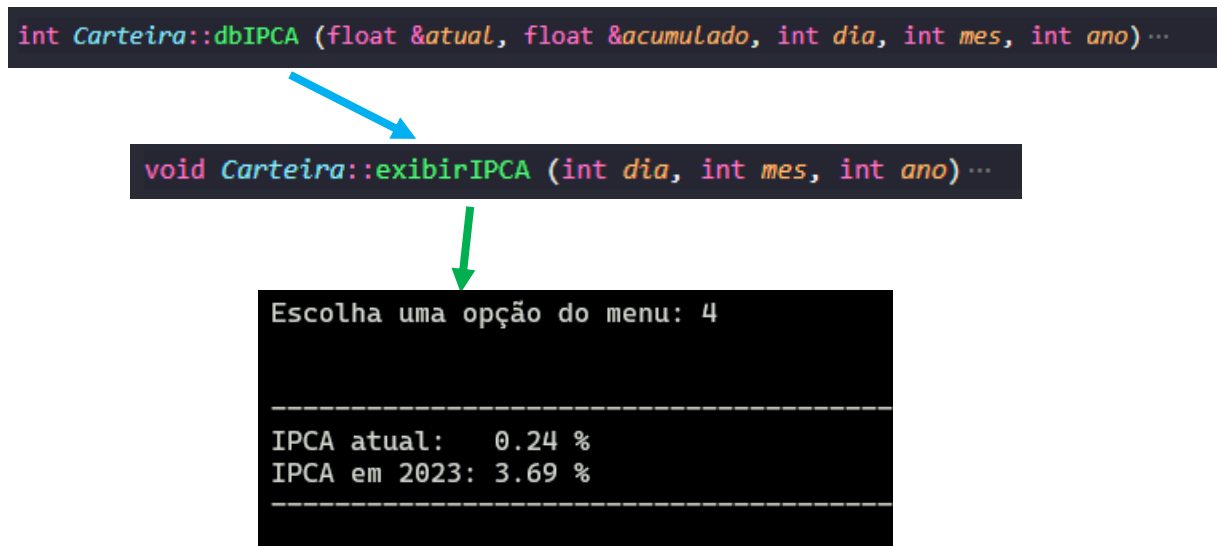
RESUMO DA VENDA:
-----
AÇÃO: EMBR3.SA | QTD: 30 | COTAÇÃO: R$23.11 | VALOR RECEBIDO: R$ 693.30
-----

Deseja continuar vendendo?
[0] - sim
[1] - não

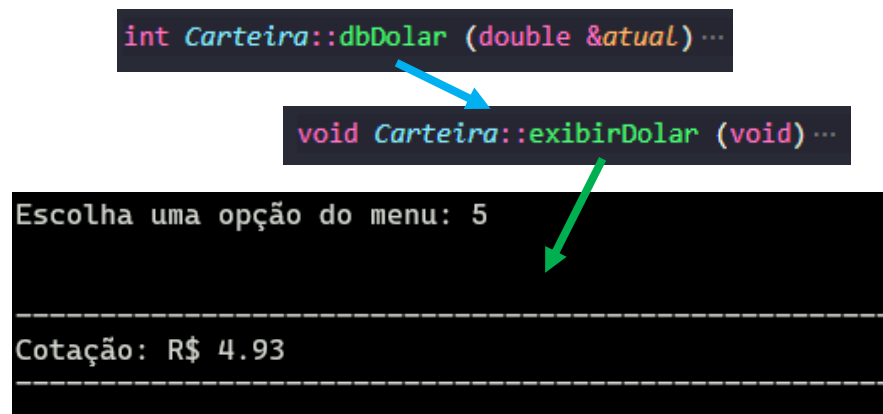
Opção: █
```

Figura 17 – Realização da venda de ação – saída da opção 3 do menu principal

As opções 4 e 5 exibem para o usuário as informações a respeito do IPCA e da cotação atual do dólar, respectivamente. Seguem abaixo imagens da saída do sistema.



**Figura 18 – Exibição do valor do IPCA – saída da opção 4 do menu principal**



**Figura 19 – Exibição do valor do dólar – saída da opção 5 do menu principal**

Por fim, na opção 6 do menu principal, é oferecido ao usuário a possibilidade de realizar uma análise fundamentalista. Para tal, é exibido uma relação de empresas disponíveis na qual o usuário deve selecionar. Uma vez escolhido, o sistema exibi diversas métricas financeiras sobre ela. Nas imagens abaixo segue um exemplo de saída do sistema.

```
int Carteira::dbEmpresa (string empresa, Empresa &dadosEmpresa) ...
```

```
int Carteira::analiseFundamentalista (string empresa) ...
```

```
Escolha uma opção do menu: 6

-----
LISTA DE EMPRESAS DA BOLSA
-----
[00] - PETR4.SA
[01] - VALE3.SA
[02] - EMBR3.SA
[03] - ITUB4.SA
[04] - BBDC4.SA
[05] - BBAS3.SA
[06] - ABEV3.SA
[07] - PETR3.SA
[08] - B3SA3.SA
[09] - ITSA4.SA
[10] - MGLU3.SA
[11] - BHIA3.SA
[12] - LREN3.SA
[13] - PRI03.SA
[14] - VBBR3.SA
[15] - AMER3.SA
[16] - MRFG3.SA
[17] - TIMS3.SA
[18] - WEGE3.SA
[19] - ELET3.SA
-----

ID empresa: █
```

```
ID empresa: 13

-----
INFO
-----
Ação na bolsa: PRI03.SA
Margem de lucro: 0.46
Margem operacional: 0.55
Margem EBITDA: 0.68
Margem bruta: 0.61
Crescimento receita: 1.04
Retorno sobre ativos: 0.15
Receita por ação: 11.73
-----

Deseja avaliar outra empresa?
[0] - sim
[1] - não

Opção: █
```

Figura 20 – Na imagem à esquerda, exibição das opções de empresas, na imagem à direita, métricas da empresa escolhida – saída da opção 6 do menu principal



## REFERÊNCIAS

Baixe dados de mercado do Yahoo! API financeira. In: PYPI.ORG. Disponível em: <https://pypi.org/>. Acesso em: 04/12/2023.

Yahoo Finance. Disponível em: <https://finance.yahoo.com/>. Acesso em: 04/12/2023.

Incorporando python em outra aplicação. In: DOCS.PYTHON. Disponível em <https://docs.python.org/3/extending/embedding.html>. Acesso em: 04/12/2023.

Introdução. In: DOC.PYTHON. Disponível em <https://docs.python.org/3/c-api/intro.html#embedding-python>. Acesso em: 04/12/2023.