# **NUBANK – Ganho de Capital**

### Sumário

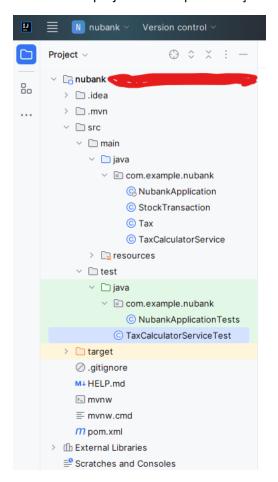
- Preparando o ambiente e estrutura de código (com imagens)
- Ferramentas e Frameworks utilizados
- Como utilizar o programa
- Testes Unitários Automatizados

# Preparando o ambiente e estrutura de código

Deve antes de tudo fazer o download das dependências do maven, comando 'mvn install' no console na pasta raiz do projeto. Existe um README dentro do projeto que detalha de forma mais descritiva, os frameworks e como executar o programa e os testes unitários. Este documento tem prints de tela que podem auxiliar na correta utilização do programa.

Não foi utilizado nenhuma arquitetura de código especifica ou design pattern por não existir complexidade suficiente que justifique como SOLID, Factory, Hexagonal, Observable etc... Apenas foi separado as regras de negocio, execução, testes e entidades que sempre é uma boa prática em na maioria das arquiteturas de software.

É um projeto com implementação mais simples e legível que pude imaginar.



### Ferramentas e Frameworks utilizados

Para esse programa foi utilizado framework do Spring Boot para simplificar o desenvolvimento e a execução da aplicação, abstraindo configuração de servidores para rodar o projeto e o Junit para os testes unitários automatizados. Também o JACKSON para facilitar a manipulação do JSON.

# Como utilizar o programa

Entrada e Saída esperadas no console. Exibe na tela instruções de como deve ser executado o programa. Exemplo de utilização do programa. Assim como consta no manual fornecido pela Nubank. É necessário pressionar ENTER após uma linha em branco para o programa entender que se trata de uma entrada de dados.

```
Console ⊙ Actuator ⊜ □ ⊚ → ⊘ :
                       [[ Lax .0.0], [ Lax .0.0], [ Lax .1000.0]]
                   [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
           =
                      {"operation": "buy", "unit-cost": 25.00, "quantity": 5000},
          =+
                     {"operation":"sell", "unit-cost":15.00, "quantity": 10000}]
          [{"tax":0.0},{"tax":0.0},{"tax":0.0}]
                      [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
                       {"operation":"buy", "unit-cost":25.00, "quantity": 5000},
                       {"operation":"sell", "unit-cost":15.00, "quantity": 10000},
                      {"operation":"sell", "unit-cost":25.00, "quantity": 5000}]
                      [{"tax":0.0},{"tax":0.0},{"tax":10000.0}]
                      [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
                      {"operation":"sell", "unit-cost":2.00, "quantity": 5000},
                      {"operation":"sell", "unit-cost":20.00, "quantity": 2000}, 
{"operation":"sell", "unit-cost":20.00, "quantity": 2000},
                      {"operation": "sell", "unit-cost": 25.00, "quantity": 1000}]
                      [{"tax":0.0},{"tax":0.0},{"tax":10000.0}]
r
                      [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
                       {"operation": "sell", "unit-cost": 2.00, "quantity": 5000},
                      {"operation":"sell", "unit-cost":20.00, "quantity": 2000},
                      {"operation": "sell", "unit-cost": 20.00, "quantity": 2000},
Ŀ
                      {"operation":"sell", "unit-cost":25.00, "quantity": 1000},
D
                      {"operation":"buy", "unit-cost":20.00, "quantity": 10000},
                      {"operation":"sell", "unit-cost":15.00, "quantity": 5000}, 
{"operation":"sell", "unit-cost":30.00, "quantity": 4350},
3
                      {"operation":"sell", "unit-cost":30.00, "quantity": 650}]
-
                      [\{"tax":0.0\}, \{"tax":0.0\}, \{"tax":4000.0\}, \{"tax":4000.0\}, \{"tax":3000.0\}, \{"tax":0.0\}, \{"tax":0.0\}, \{"tax":3700.0\}, \{"tax":3700.0\}, \{"tax":0.0\}, 
1
```

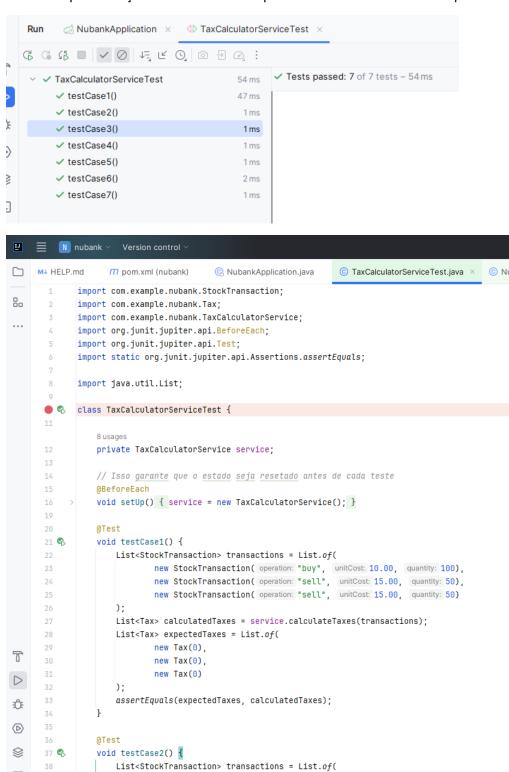
```
NubankApplication >
Run
τορειαιτοπ . σειτ , οπτι-σοσι .το.ου, quantity . σοιι
    [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
    {"operation": "sell", "unit-cost": 20.00, "quantity": 5000},
\downarrow
    {"operation": "sell", "unit-cost":5.00, "quantity": 5000}]
=
= \downarrow
    [{"operation":"buy", "unit-cost":10.00, "quantity": 100},
{"operation": "sell", "unit-cost": 15.00, "quantity": 50},
亩
    {"operation": "sell", "unit-cost":15.00, "quantity": 50}]
    [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
    {"operation": "sell", "unit-cost": 20.00, "quantity": 5000},
    {"operation": "sell", "unit-cost":5.00, "quantity": 5000}]
    [{"tax":0.0},{"tax":0.0},{"tax":0.0}]
    [{"operation": "buy", "unit-cost":10.00, "quantity": 100},
    {"operation": "sell", "unit-cost":15.00, "quantity": 50},
    {"operation": "sell", "unit-cost": 15.00, "quantity": 50}]
    [{"tax":0.0},{"tax":0.0},{"tax":0.0}]
    [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
    {"operation": "sell", "unit-cost": 20.00, "quantity": 5000},
    {"operation": "sell", "unit-cost":5.00, "quantity": 5000}]
    [{"tax":0.0},{"tax":0.0},{"tax":0.0}]
    [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
    {"operation": "sell", "unit-cost": 5.00, "quantity": 5000},
    {"operation": "sell", "unit-cost": 20.00, "quantity": 3000}]
    [{"tax":0.0},{"tax":0.0},{"tax":1000.0}]
    [{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
    {"operation": "sell", "unit-cost": 5.00, "quantity": 5000},
    {"operation": "sell", "unit-cost": 20.00, "quantity": 3000}]
```

#### **Testes Unitários Automatizados**

>\_

39

Existe a implementação de testes unitários para testas os casos de testes explícitos no documento:



new StockTransaction( operation: "buy", unitCost: 10.00, quantity: 10000),

```
N nubank ~
                        Version control ~
     M↓ HELP.md
                     m pom.xml (nubank)

    NubankApplication.java

                                                                         ○ TaxCalculatorServiceTest.java ×
       35
80
      36
                  @Test
      37 %
                  void testCase2() {
                      List<StockTransaction> transactions = List.of(
      38
                               new StockTransaction( operation: "buy", unitCost: 10.00, quantity: 10000),
      39
                               new StockTransaction( operation: "sell", unitCost: 20.00, quantity: 5000),
       40
                               new StockTransaction( operation: "sell", unitCost: 5.00, quantity: 5000)
       41
       42
                      );
                      List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
       43
       44
                      List<Tax> expectedTaxes = List.of(
       45
                               new Tax(0.0),
                               new Tax(10000.0),
                               new Tax(0.0)
       47
       48
                      );
                      assertEquals(expectedTaxes, calculatedTaxes);
      51
      52
                  @Test
      53 🚯
                  void testCase3() {
                      List<StockTransaction> transactions = List.of(
       54
                               new StockTransaction( operation: "buy", unitCost: 10.00, quantity: 10000),
                               new StockTransaction( operation: "sell", unitCost: 5.00, quantity: 5000),
                               new StockTransaction( operation: "sell", unitCost: 20.00, quantity: 3000)
       57
       58
                      );
                      List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
                      List<Tax> expectedTaxes = List.of(
      61
                               new Tax(0.0),
                               new Tax(0.0).
      62
                               new Tax(1000.0)
      64
                      );
                      assertEquals(expectedTaxes, calculatedTaxes);
      66
      67
      68
                  @Test
Ø
      69 B
                  void testCase4() {
      70
                      List<StockTransaction> transactions = List.of(
                               new StockTransaction( operation: "buy", unitCost: 10.00, quantity: 10000),
      71
>_
                               new StockTransaction( operation: "buy", unitCost: 25.00, quantity: 5000),
\bigcirc
         MuhankApplication V A TayCalculatorSorviceTest V
```

```
IJ
     ■ N nubank ∨ Version control
     M↓ HELP.md
                     m pom.xml (nubank)

    NubankApplication.java

                                                                         ○ TaxCalculatorServiceTest.java × ○ NubankApplicationTests.java
       68
80
       69 B
                  void testCase4() {
       70
                       List<StockTransaction> transactions = List.of(
                               new StockTransaction( operation: "buy", unitCost: 10.00, quantity: 10000),
                                new StockTransaction( operation: "buy", unitCost: 25.00, quantity: 5000),
       72
                               new StockTransaction( operation: "sell", unitCost: 15.00, quantity: 10000)
       73
       74
                       );
                       List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
       75
                       List<Tax> expectedTaxes = List.of(new Tax(0.0), new Tax(0.0), new Tax(0.0));
       76
       77
                       assertEquals(expectedTaxes, calculatedTaxes);
                  }
       78
       79
       80
                  @Test
       81 %
                  void testCase5() {
                      List<StockTransaction> transactions = List.of(
       82
                               new StockTransaction( operation: "buy", unitCost: 10.00, quantity: 10000),
       83
                               new StockTransaction( operation: "buy", unitCost: 25.00, quantity: 5000),
       84
                               new StockTransaction( operation: "sell", unitCost: 15.00, quantity: 10000),
       85
                               new StockTransaction( operation: "sell", unitCost: 25.00, quantity: 5000)
       86
       87
                       ):
       88
                       List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
                       List<Tax> expectedTaxes = List.of(\text{new Tax}(0.0), \text{new Tax}(0.0), \text{new Tax}(0.0), \text{new Tax}(0.0));
                       assertEquals(expectedTaxes, calculatedTaxes);
                  }
       92
       93
                  @Test
       94 %
                  void testCase6() {
       95
                       List<StockTransaction> transactions = List.of(
                               new StockTransaction( operation: "buy", unitCost: 10.00, quantity: 10000),
       96
                               new StockTransaction( operation: "sell", unitCost: 2.00, quantity: 5000),
       97
                               new StockTransaction( operation: "sell", unitCost: 20.00, quantity: 2000),
       98
                               new StockTransaction( operation: "sell", unitCost: 20.00, quantity: 2000),
       99
Ûŧ
                               new StockTransaction( operation: "sell", unitCost: 25.00, quantity: 1000)
                      ):
Ø
                      List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
\otimes
                      List<Tax> expectedTaxes = List.of(new Tax(0.0), new Tax(0.0), new Tax(0.0), new Tax(0.0), new Tax(3000.0));
      104
                       assertEquals(expectedTaxes, calculatedTaxes);
>_
```