NUBANK – Ganho de Capital

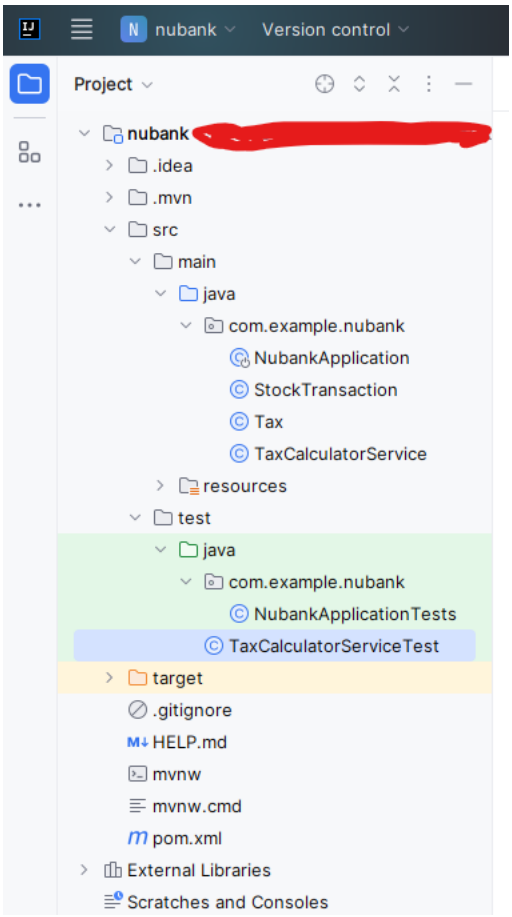
Sumário

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.

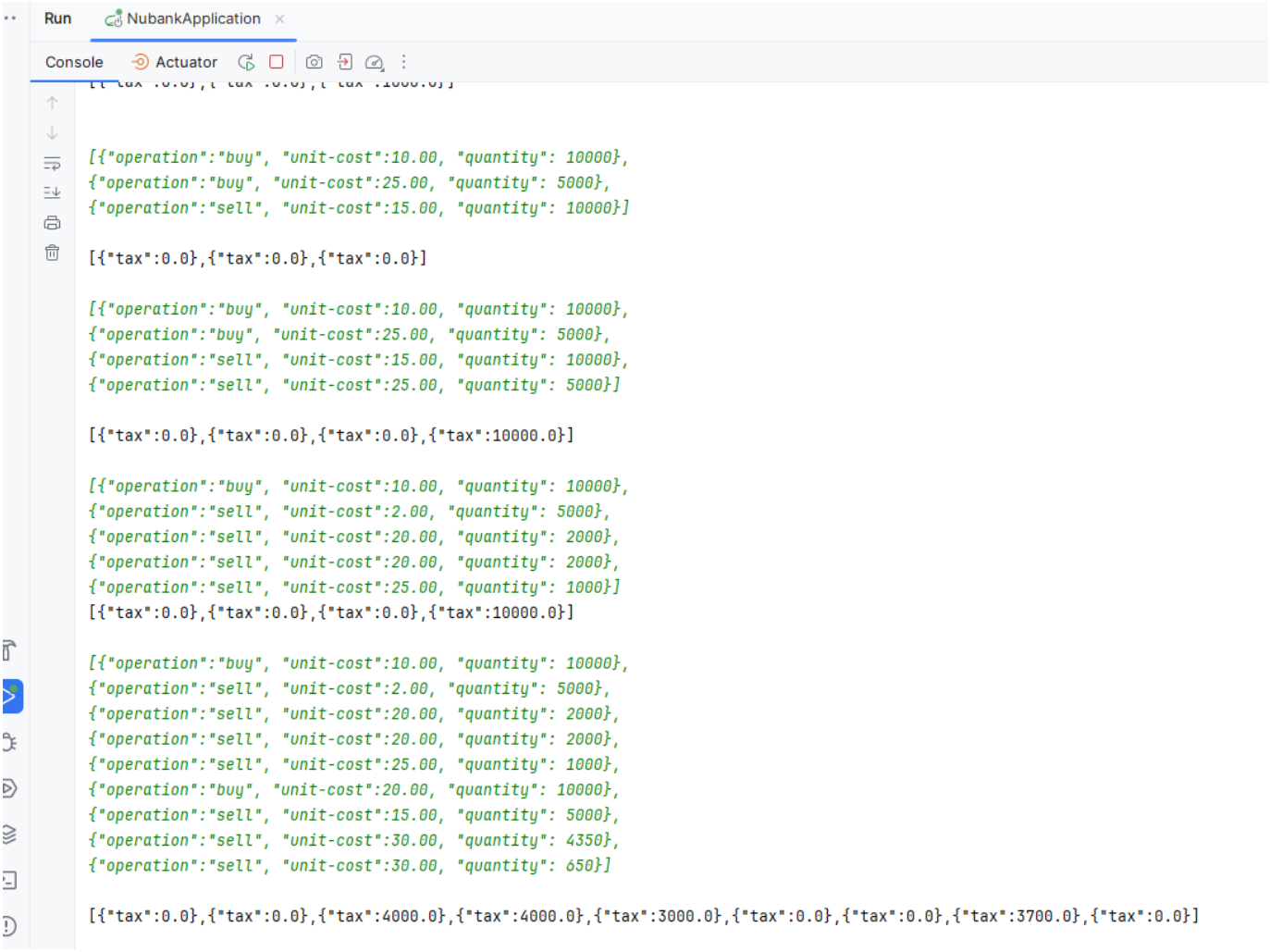


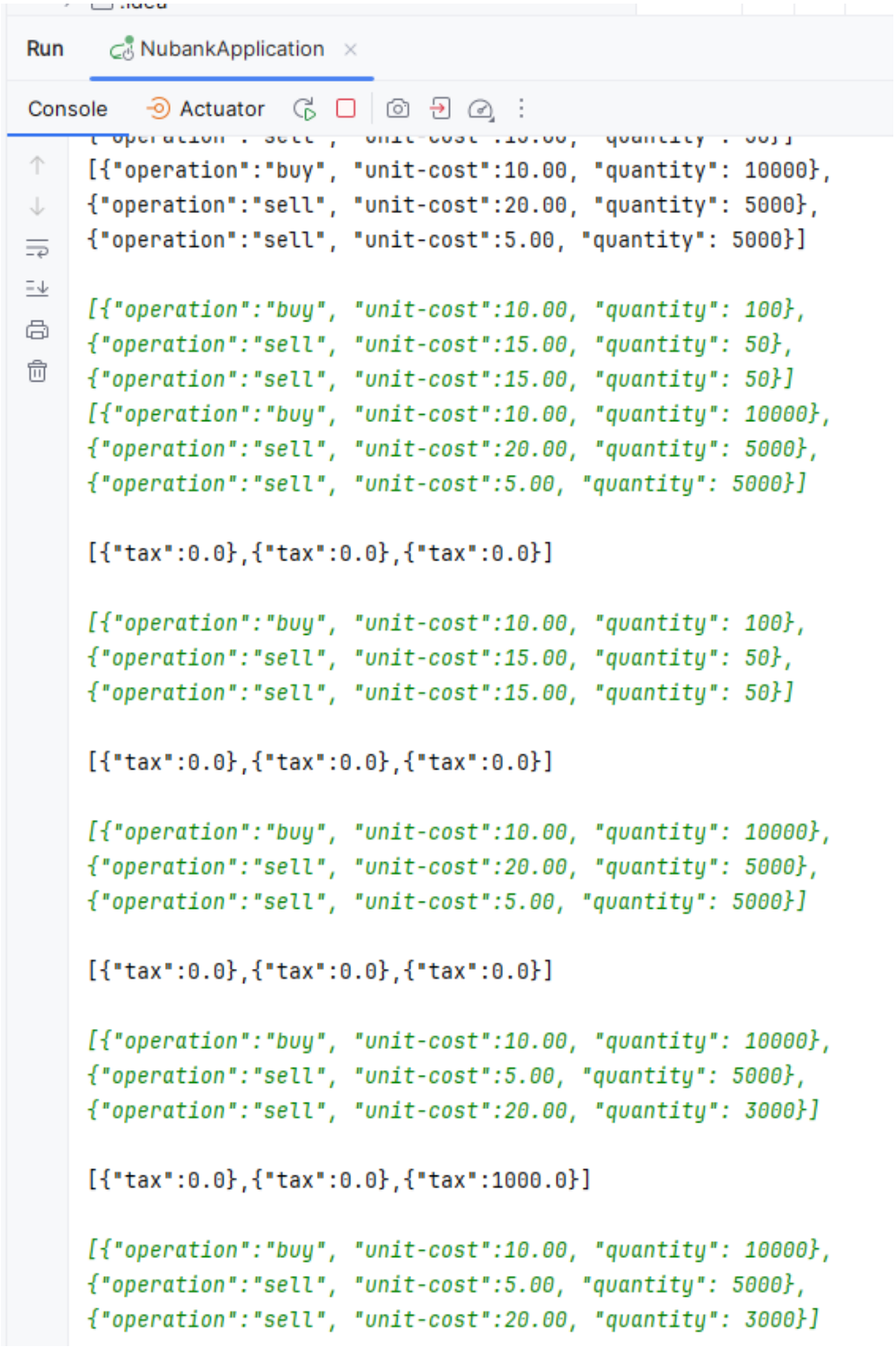
Ferramentas e Frameworks utilizados

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

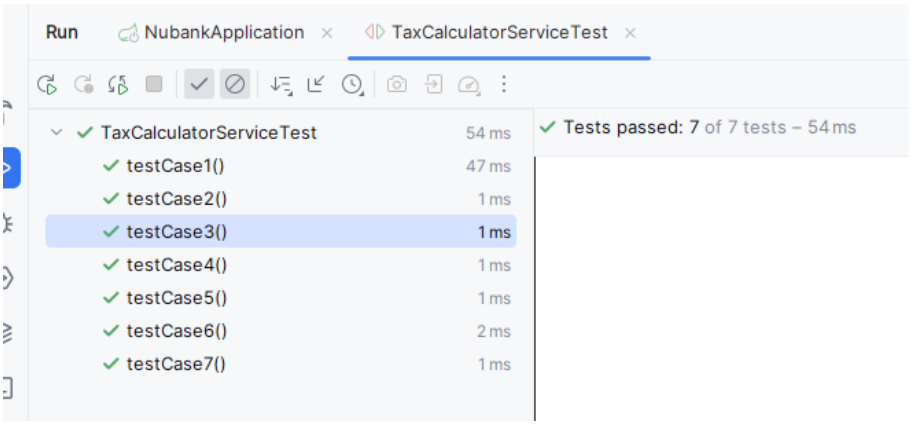
Entrada e Saída esperadas no console. Exibe na tela instruções de como deve ser executado o programa. Exemplo de utilização do programa. Assim como consta no manual fornecido pela Nubank. É necessário pressionar ENTER após uma linha em branco para o programa entender que se trata de uma entrada de dados.

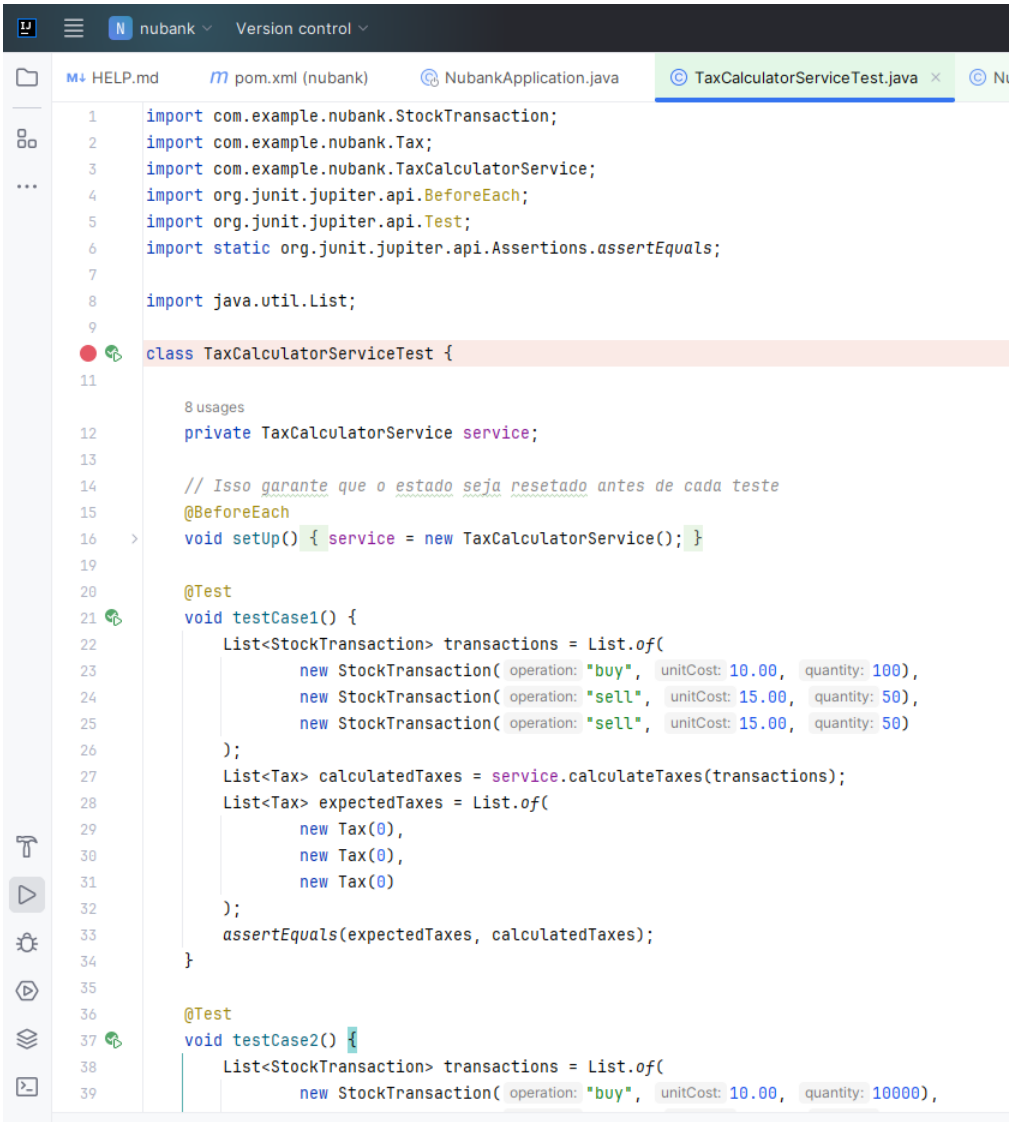

Run
NubankApplication
Console
Actuator
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"buy", "unit-cost":25.00, "quantity": 5000},
{"operation":"sell", "unit-cost":15.00, "quantity": 10000}]
[{"tax":0.0},{"tax":0.0},{"tax":0.0}]
[{"operation":"buy", "unit-cost":10.00, "quantity": 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":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":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},
{"operation":"buy", "unit-cost":20.00, "quantity": 10000},
{"operation":"sell", "unit-cost":15.00, "quantity": 5000},
{"operation":"sell", "unit-cost":30.00, "quantity": 4350},
{"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":0.0}]

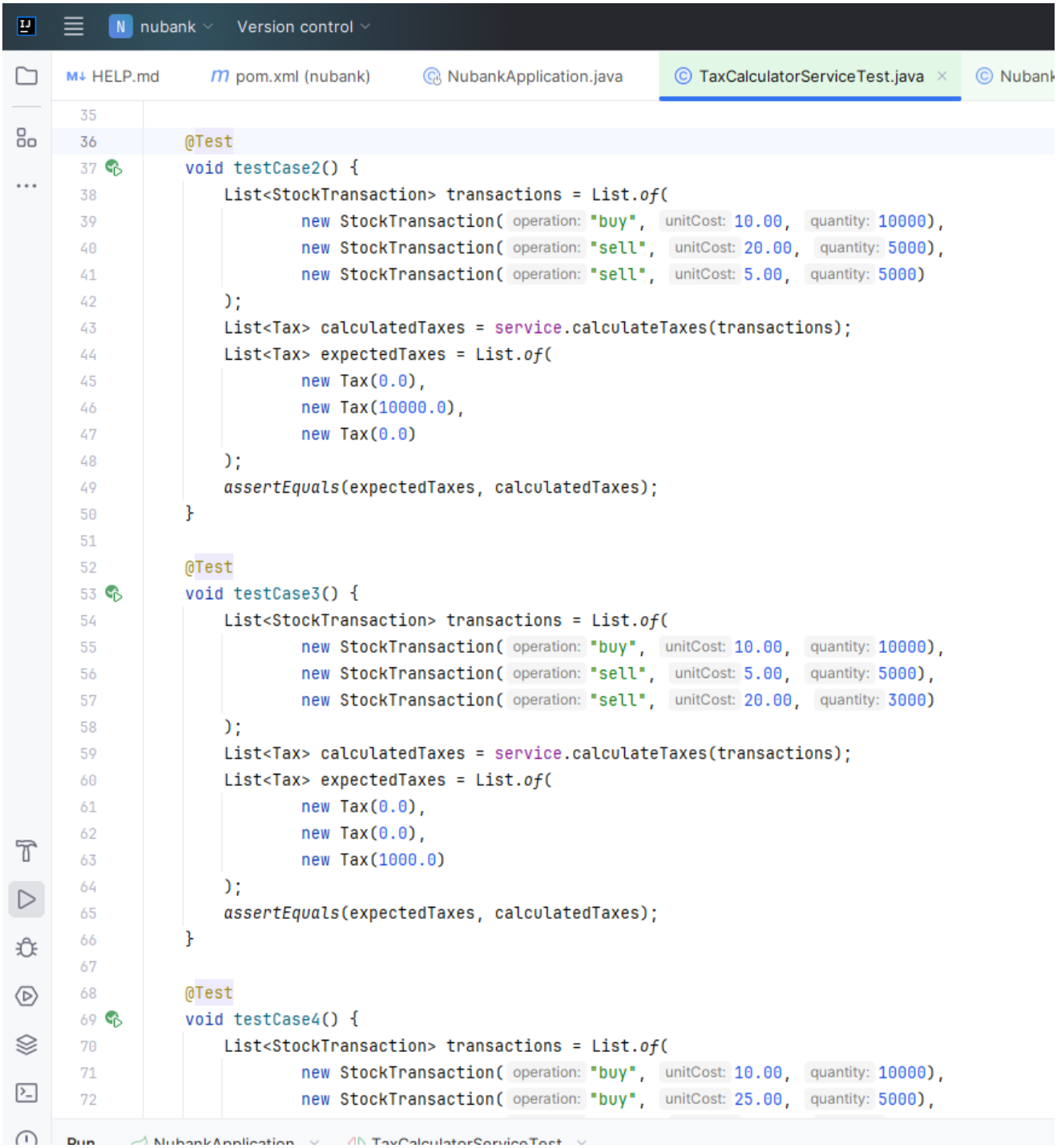

Run
NubankApplication
Console
Actuator
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"sell", "unit-cost":20.00, "quantity": 5000},
{"operation":"sell", "unit-cost":5.00, "quantity": 5000}]
[{"operation":"buy", "unit-cost":10.00, "quantity": 100},
{"operation":"sell", "unit-cost":15.00, "quantity": 50},
{"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}]

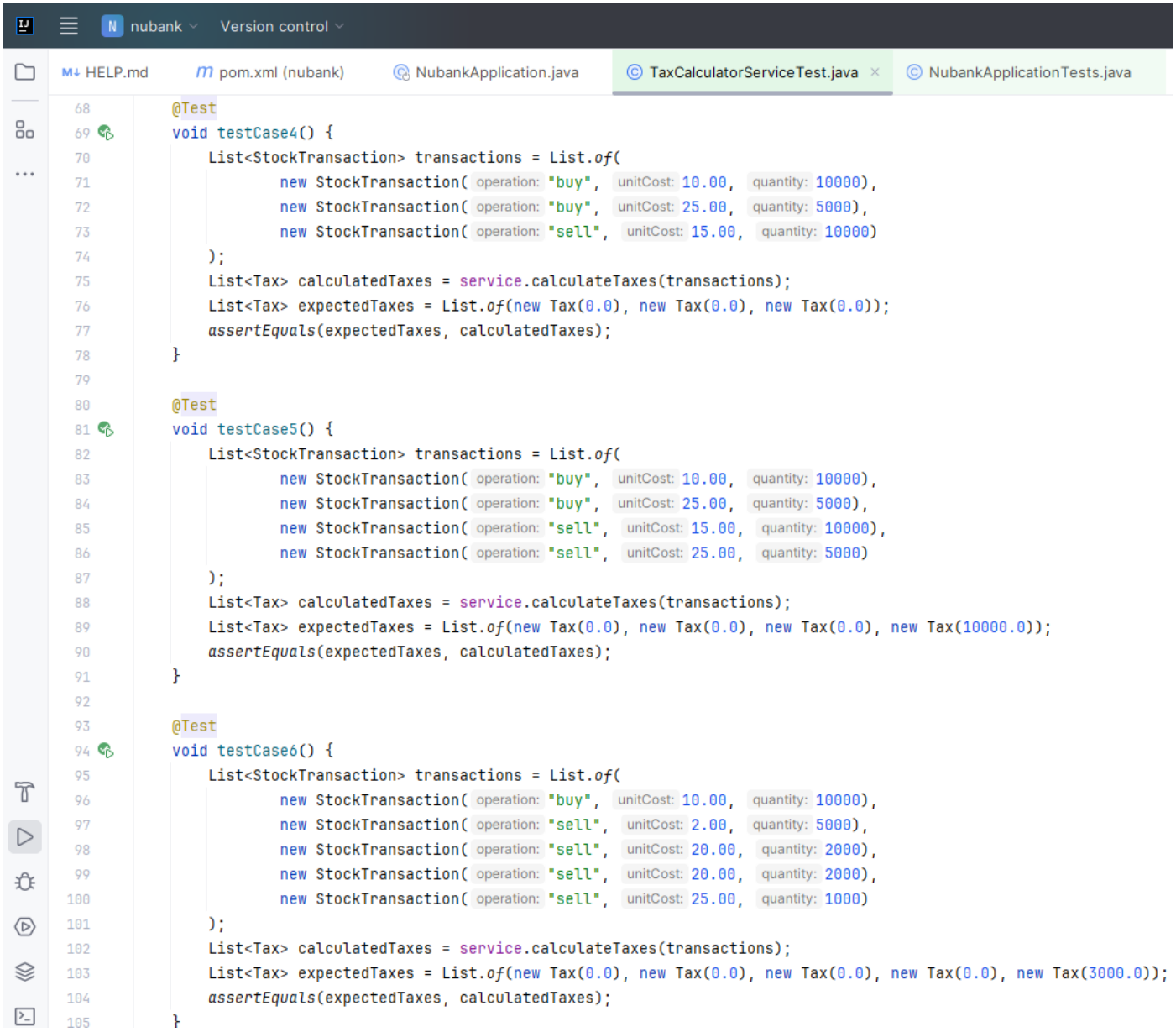
Testes Unitários Automatizados

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






nubank
Version control
HELP.md
pom.xml (nubank)
NubankApplication.java
TaxCalculatorServiceTest.java
Nubank
@Test
void testCase2() {
List<StockTransaction> transactions = List.of(
new StockTransaction(operation: "buy", unitCost: 10.00, quantity: 10000),
new StockTransaction(operation: "sell", unitCost: 20.00, quantity: 5000),
new StockTransaction(operation: "sell", unitCost: 5.00, quantity: 5000)
);
List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
List<Tax> expectedTaxes = List.of(
new Tax(0.0),
new Tax(10000.0),
new Tax(0.0)
);
assertEquals(expectedTaxes, calculatedTaxes);
}
@Test
void testCase3() {
List<StockTransaction> transactions = List.of(
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)
);
List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
List<Tax> expectedTaxes = List.of(
new Tax(0.0),
new Tax(0.0),
new Tax(1000.0)
);
assertEquals(expectedTaxes, calculatedTaxes);
}
@Test
void testCase4() {
List<StockTransaction> transactions = List.of(
new StockTransaction(operation: "buy", unitCost: 10.00, quantity: 10000),
new StockTransaction(operation: "buy", unitCost: 25.00, quantity: 5000),
Run


nubank
Version control
HELP.md
pom.xml (nubank)
NubankApplication.java
TaxCalculatorServiceTest.java
NubankApplicationTests.java
@Test
void testCase4() {
List<StockTransaction> transactions = List.of(
new StockTransaction(operation: "buy", unitCost: 10.00, quantity: 10000),
new StockTransaction(operation: "buy", unitCost: 25.00, quantity: 5000),
new StockTransaction(operation: "sell", unitCost: 15.00, quantity: 10000)
);
List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
List<Tax> expectedTaxes = List.of(new Tax(0.0), new Tax(0.0), new Tax(0.0));
assertEquals(expectedTaxes, calculatedTaxes);
}
@Test
void testCase5() {
List<StockTransaction> transactions = List.of(
new StockTransaction(operation: "buy", unitCost: 10.00, quantity: 10000),
new StockTransaction(operation: "buy", unitCost: 25.00, quantity: 5000),
new StockTransaction(operation: "sell", unitCost: 15.00, quantity: 10000),
new StockTransaction(operation: "sell", unitCost: 25.00, quantity: 5000)
);
List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
List<Tax> expectedTaxes = List.of(new Tax(0.0), new Tax(0.0), new Tax(0.0), new Tax(10000.0));
assertEquals(expectedTaxes, calculatedTaxes);
}
@Test
void testCase6() {
List<StockTransaction> transactions = List.of(
new StockTransaction(operation: "buy", unitCost: 10.00, quantity: 10000),
new StockTransaction(operation: "sell", unitCost: 2.00, quantity: 5000),
new StockTransaction(operation: "sell", unitCost: 20.00, quantity: 2000),
new StockTransaction(operation: "sell", unitCost: 20.00, quantity: 2000),
new StockTransaction(operation: "sell", unitCost: 25.00, quantity: 1000)
);
List<Tax> calculatedTaxes = service.calculateTaxes(transactions);
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));
assertEquals(expectedTaxes, calculatedTaxes);
}