## **Alunos:**

Arthur Kassick Ferreira

João Davi Martins Nunes

Lucas Henrik Miranda Souza

Yago Carvalho

### Professora:

**Ingrid Nunes** 

Técnicas de Construção de Programas 2019/1

# Relatório do Trabalho Prático - Etapa 1

## INTRODUÇÃO

Trabalho Prático (Etapa 1) consiste na evolução do projeto do Sistema Bancário, tanto em sua implementação em Java, como em seus diagramas (de classes e de sequência), outrora apresentado em aula pela professora. Nesta etapa, o grupo atua como *MAINTAINER* do projeto, adicionando algumas funcionalidades que incrementam o projeto original.

## PONTOS DE MUDANÇA

Primeiro, mostramos como funciona o projeto originalmente.

- 1. O usuário, que possui uma conta no banco, deseja fazer um depósito.
- 2. Caso a solicitação seja feita de um caixa eletrônico, a agência e conta do cliente (usuário) são recuperadas da sessão, caso contrário (a solicitação seja feita de uma agência), o sistema solicita ao funcionário (usuário) agência e conta do cliente.
- 3. Sistema solicita a número do envelope, e o usuário informa.

- 4. Sistema solicita a quantidade a ser depositada, e o usuário informa.
- 5. Sistema credita o valor a ser depositado na conta.
- 6. Sistema registra a transação na conta.
- 7. Sistema exibe o status da operação.
- 8. Sistema retorna ao menu principal.

Agora, mostramos as mudanças, que consiste basicamente em um registro do estado do depósito, colocando-o em FINALIZADO ou PENDENTE, de acordo com os casos a seguir (mudanças nas etapas 5, 6 e 7):

- 5. Caso a solicitação seja feita em uma agência,
  - a. Sistema credita o valor a ser depositado na conta.
  - b. Sistema registra a transação com estado FINALIZADA.
- Caso a solicitação seja feita em um caixa eletrônico e a quantidade a ser depositada seja inferior ou igual à R\$100,00,
  - a. Sistema credita o valor a ser depositado na conta.
  - b. Sistema registra a transação com estado PENDENTE.
- 7. Caso a solicitação seja feita em um caixa eletrônico e a quantidade a ser transferida seja superior a R\$100,00,
  - a. Sistema registra a transação com estado PENDENTE na conta de origem.

O usuário também deverá obter mais um recurso: poder CONFIRMAR DEPÓSITO, na qual uma funcionalidade que habilitaria-o de visualizar os depósitos pendentes. Nesta nova funcionalidade, o usuário poderá confirmar ou recusar os depósitos via sistema, alterando o saldo da sua conta.

## DIAGRAMAS DE CLASSE

## • Diagrama de Classe Business/Domain

A interface *AccountOperationService* (e a sua implementação) teve a adição de outro método *public Deposit deposit*, utilizando-se a técnica de reusa de sobrecarga com diferentes assinaturas, na qual *String status* e *boolean creditEnable* foram adicionados. Tais parâmetros servem para controlar o status do depósito e o seu crédito disponível. Mais 2 métodos foram incrementados nas classes: *getDepositByStatus* e *updateDeposits*. O primeiro método retorna uma lista ordenada de acordo com os depósitos existentes; o segundo, atualiza o status de um certo depósito existente na conta.

Na classe *CurrentAccount*, o mesmo método *public Deposit deposit* foi feito (explicação acima). O método *updateDeposit* também foi implementado porém apenas com 3 parâmetros, pois já possui acesso à conta. Também alteramos a visibilidade do método *depositAmount* de privado para público.

## Diagrama de classe Domain

A classe *Deposit* foi adicionado um atributo estático *BigDecimal verification\_amount*, que é a quantia a ser verificada. Como consequência das modificações nas classes anteriores, o construtor foi alterado, adicionando *String status* como parâmetro.

Na classe abstrata *Transaction* é onde é implementada as modificações decorrentes do incremento do status do depósito. Um novo método construtor foi feito, adicionando o parâmetro do status, pois, se mexessemos no original, podíamos causar um erro no programa, em vista que poderiam haver outras classes e métodos que usassem o construtor original. Colocamos o *getter* e o *setter* do parâmetro *String status*. Um atributo estático *status* foi adicionado, que coloca em pendência os status dos depósitos do programa, além de mais 3 atributos públicos que irão certificar o status consequentes: *pending\_status*, *accepted\_status* e *refused\_status*.

Nas classes *Transfer e Withdrawal*, o construtor agora manda o status "FINALIZADO" para a classe da qual estende, não necessitando receber um status como parâmetro.

Na classe abstrata *OperationLocation* foi adicionado um método abstrato chamado de *initializeDeposit*, que, como o próprio nome diz: inicializa um depósito.

## • Diagrama Command

Foi criada uma nova classe *PendingDepositsCommand* que estende a classe *Command*, que possui os comandos para os depósitos pendentes. Foram inseridos os atributos privados:

- AccountOperationService accountOperationService
- *List*<*String*> *acceptedList*
- *List*<*String*> *rejected*

Os métodos dessa nova classe são:

- execute
- AmountToDebit
- AmountToCredit
- status\_map\_fill
- balance\_map\_fill

Mudamos a visibilidade do método *printStatement* da classe *StatementCommand* de privado para protegido.

## DIAGRAMA DE SEQUÊNCIA

Digite seu texto aqui Digite seu texto aqui.

## OBSERVAÇÕES

Diagramas no final do pdf seguem a sequência mostrada no relatório: Diagramas de Classe (Diagrama Business/Domain, Diagrama Domain e Diagrama Command) e o Diagrama de Sequência.

pkg

# AccountOperationService

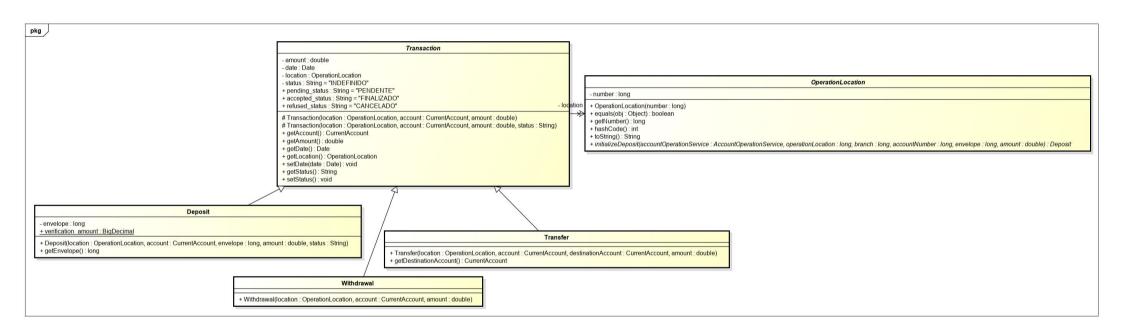
#### AccountOperationServiceImpl

- + AccountOperationServiceImpl(database : Database)
- + deposit(operationLocation: long, branch: long, accountNumber: long, envelope: long, amount: double, status: String): Deposit
- + deposit(operationLocation: long, branch: long, accountNumber: long, envelope: long, amount: double, status: String, creditEnable: boolean): Deposit
- + getBalance(branch : long, accountNumber : long) : double
- getOperationLocation(operationLocationNumber : long) : OperationLocation
- getStatementByDate(currentAccount : CurrentAccount, begin : Date, end : Date) : List<Transaction>
- + getStatementByDate(branch : long, accountNumber : long, begin : Date, end : Date) : List<Transaction>
- + getStatementByMonth(branch : long, accountNumber : long, month : int, year : int) : List<Transaction>
- + login(branch : long, accountNumber : long, password : String) : CurrentAccount
- + transfer(operationLocation: long, srcBranch: long, srcAccounttNumber: long, dstBranch: long, dstAccountNumber: long, amount: double): Transfer
- + withdrawal(operationLocation: long, branch: long, accountNumber: long, amount: double): Withdrawal
- + getDepositByStatus() : List<Transaction>
- + updateDeposits(branch : long, accountNumber : long, deposit : Deposit, newStatus : String, balanceChange : double) : Deposit

### CurrentAccount

#### - balance : double

- + CurrentAccount(branch : Branch, number : long, client : Client)
- + CurrentAccount(branch : Branch, number : long, client : Client, initialBalance : double)
- + deposit(location : OperationLocation, envelope : long, amount : double, status : String, creditEnable : boolean) : Deposit
- depositAmount(amount : double) : void
- + getBalance() : double
- + getClient(): Client
- + getDeposits() : List<Deposit>
- + getId(): CurrentAccountId
- + getTransactions() : List<Transaction>
- + getTransfers(): List<Transfer>
- + getWithdrawals(): List<Withdrawal>
- hasEnoughBalance(amount : double) : boolean
- isValidAmount(amount : double) : boolean
- + transfer(location : OperationLocation, destinationAccount : CurrentAccount, amount : double) : Transfer
- + withdrawal(location : OperationLocation, amount : double) : Withdrawal
- withdrawalAmount(amount : double) : void
- + updateDeposit(deposit : Deposit, newStatus : String, balanceChange : double) : Deposit



#### Command

- isEnabled : boolean

# Command(bankInterface : BankTextInterface)

# Command(bankInterface : BankTextInterface, isEnabled : boolean)

# getTextManager(): TextManager

+ isEnabled(): boolean

+ setEnabled(isEnabled : boolean) : void

## **PendingDepositsCommand**

- accountOperationService : AccountOperationService
- acceptList : List<String>
- rejectList : List<String>
- + execute(): void
- AmountToDebit(): Double
- AmountToCredit(amount : Double) : Double
- status\_map\_fill(): Map<String, String>
- balance\_map\_fill(): Map<String, Double>

### StatementCommand

## - NUMBER OF POSSIBLE MONTHS: int = 6

- + StatementCommand(bankInterface : BankTextInterface, accountOperationService : AccountOperationService)
- + execute(): void
- # printStatement(cald : CurrentAccountId, transactions : List<Transaction>) : void
- readStatementType(): StatementType
- showMonthlyStatement(branch : Long, accountNumber : Long) : void
- showStatementByPeriod(branch : Long, accountNumber : Long) : void

