UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

INF01120 - TÉCNICAS DE CONSTRUÇÃO DE PROGRAMAS

TRABALHO PRÁTICO 1 - MAINTAINER

IMPLEMENTAÇÃO DE FUNCIONALIDADES NO SISTEMA BANCÁRIO

Bruno Silva Trindade - *00240505*

Fábio Azevedo Gomes - *00287696*

Paulo Cesar Valcarenghi Junior - *00243695*

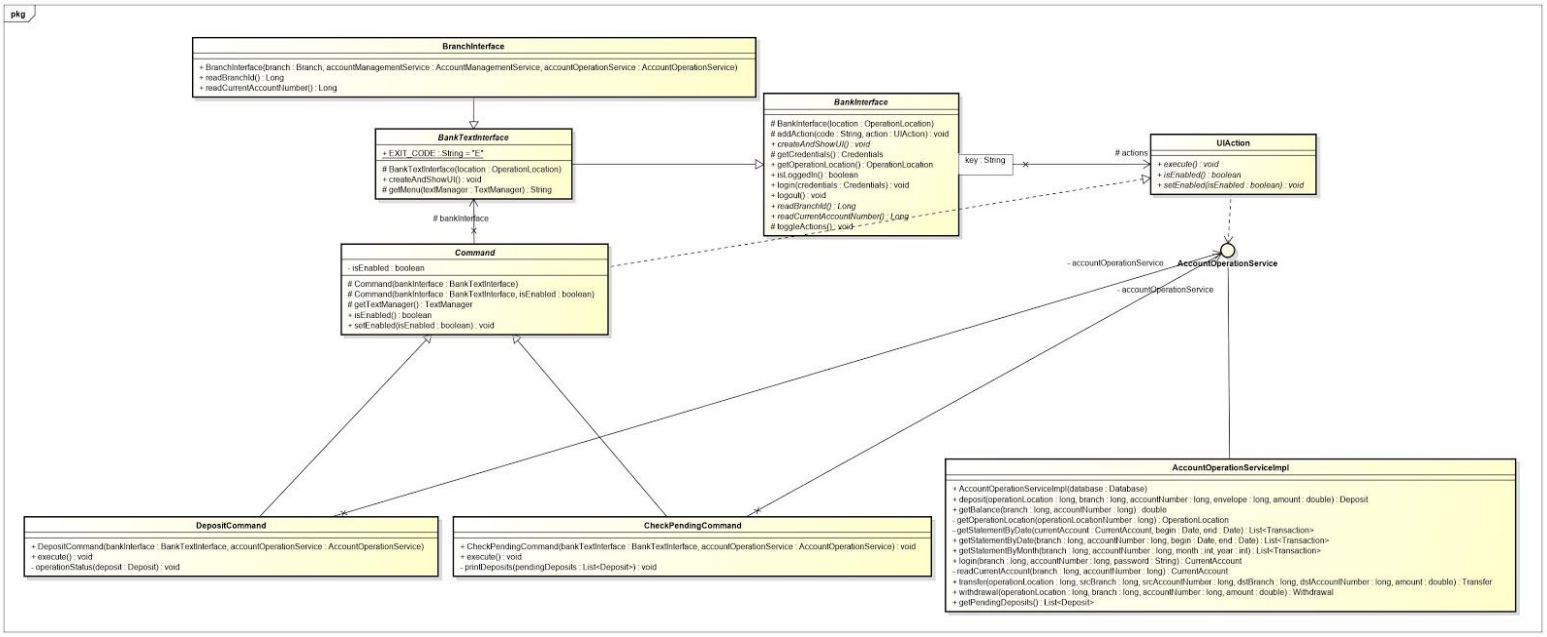
Rafael Baldasso Audibert - *00287695*

MAIO, 2019

I. DIAGRAMA DE CLASSES

Abaixo se encontram os diagramas de classes, que se referem somente às partes do Sistema Bancário as quais foram modificadas pelos integrantes do grupo. Ao final, possuímos imagens maiores, ocupando toda a folha em modo paisagem.

a) bank.UI.text



Existe uma imagem maior no final do arquivo

Tivemos várias alterações relacionadas à interface visual do programa (utilizamos a intercace de TEXTO. As alterações seguem definidas da forma *"classe"."método" ou "classe"."propriedade"*:

- *(AccountOperationService e AccountOperationServiceImpl) .getPendingDeposits() : List<Deposit>* -- <u>Método Novo - PÚBLICO</u>

  Essa função itera sobre todos os clientes do banco (*account = database.getAllCurrentAccounts()*), e para cada cliente itera sobre todos os depósitos já realizados (*deposit = account.getDeposits()*), retornando todos aqueles que estejam com *deposit.status == PENDING.*

- *BranchInterface() : BranchInterface* -- <u>Alteração Construtor</u>

  Foi adicionado uma nova ação *this.addAction("P", new CheckPendingCommand(this, accountOperationService))*, que é a responsável por chamar CheckPendingCommand, para mostrar todos os

depósitos que ainda estão pendentes para os funcionários da agência, para poder autorizar ou não um depósito.

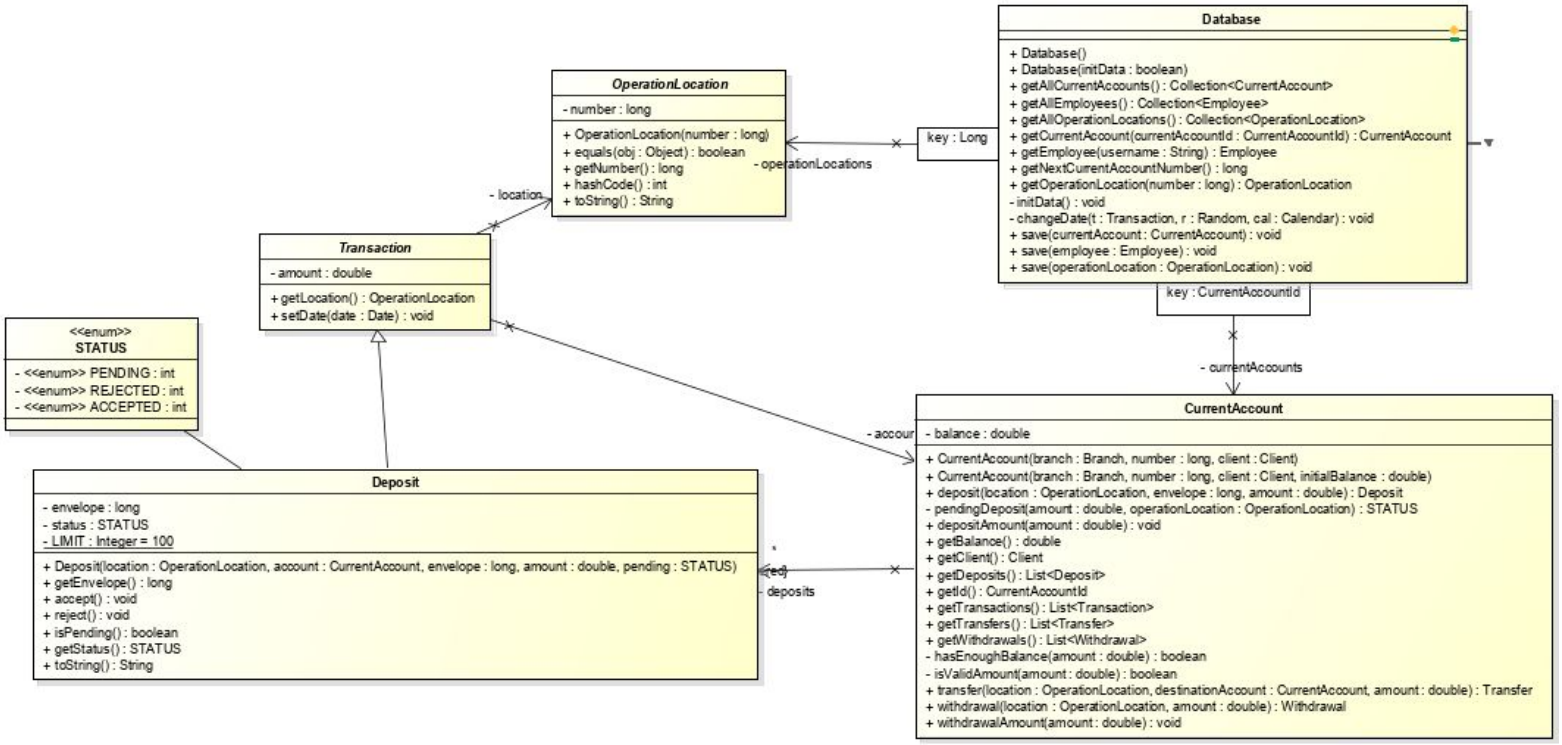
- `CheckPendingCommand() : CheckPendingCommand` -- ***NOVA CLASSE***

  Nova classe que herda de `Command`, responsável por verificar os comandos pendentes (acessível apenas da interface da `Branch`).

- `CheckPendingCommand.execute() : void` -- ***NOVO MÉTODO***, *Público*

  Método chamado de `BankTextInterface.createAndShowUI` para executar a ação de "checar os depósitos pendentes". Os depósitos pendentes são acessados chamando `accountOperationService.getPendingDeposits()`. Enquanto temos depósitos pendentes ficamos mostrando todos os depósitos para o usuário (que precisa estar em uma `Branch`), pedindo para ele indicar o index de um dos depósitos. Caso ele digitar 0, ele retorna para `BankTextInterface`. Para cada depósito selecionado, ele pode aceitá-lo ou rejeitá-lo, chamando `Deposit.accept()` ou `Deposit.reject()`, respectivamente.

b) bank.business.domain



Dentro do pacote *bank.business.domain*, ocorreram alterações em várias classes, as quais seguem definidas da forma *"classe"."método" ou "classe"."propriedade"*:

- *enumerador STATUS* -- NOVO

  Enumerador que armazena os possíveis status que um Deposit pode estar. São eles: *PENDING*, *REJECTED* e *ACCEPTED*

- *Deposit()* -- Construtor Atualizado

  Agora configura o parametro status. Caso o depósito tenha sido criado em uma *Branch*, marca ele como *STATUS.ACCEPTED*, caso contrário, como *STATUS.PENDING*.

- *Deposit.LIMIT : Integer = 100* -- Propriedade Nova - PRIVADA, ESTÁTICA

  Constante que representa qual o valor máximo que pode ser debitado em uma conta sem a necessidade de verificação de um atendente.

- *Deposit.status : STATUS* -- Propriedade Nova - PRIVADA

  Estado atual do depósito, que pode ser qualquer um dos dispostos no enumerador *STATUS*.

- *Deposit.getStatus() : STATUS* -- Método Novo - PÚBLICO

Retorna o estado atual do depósito (definido em *status*)

- *Deposit.isPending() : Boolean* -- Método Novo - PÚBLICO

  Retorna *true* se estado atual do depósito (definido em *status*) for *PENDING*

- *Deposit.accept() : void* -- Método Novo - PÚBLICO

  Caso o depósito esteja pendente, autoriza o depósito, transformando o *status* para *ACCEPTED*, e creditando o valor do depósito na conta caso ele não tenha sido creditado durante a criação do Depósito (valor do depósito era superior à *LIMIT*)

- *Deposit.reject() : void* -- Método Novo - PÚBLICO

  Caso o depósito esteja pendente, rejeita o depósito, transformando o *status* para *REJECTED*, debitando o valor na conta caso esse valor tiver sido creditado durante a criação do Depósito (valor do depósito era inferior à *LIMIT*), podendo ficar com saldo negativo.

- *Deposit.toString() : String* -- Método Override - PÚBLICO

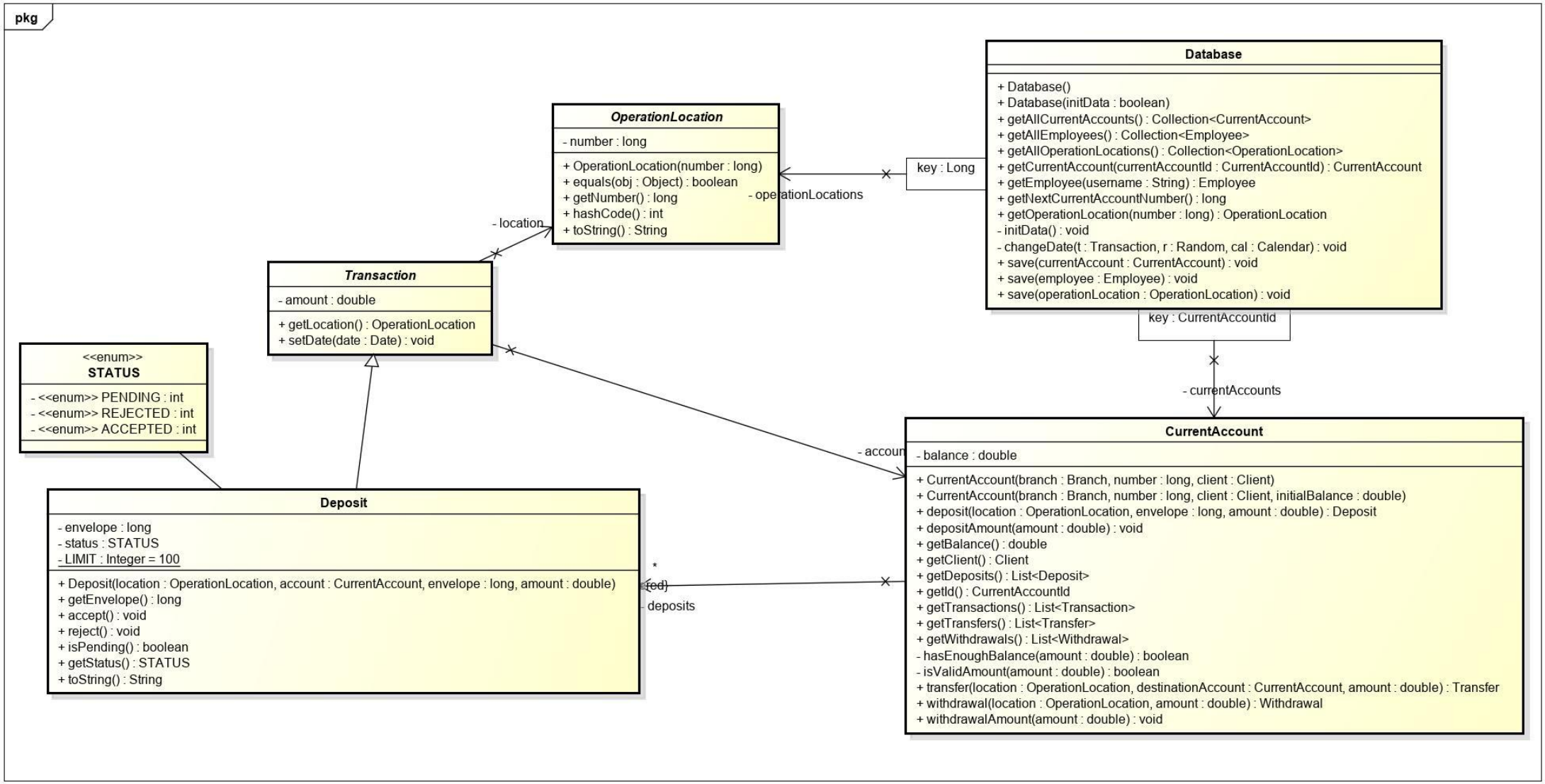
  Customização do *toString*() do depósito, retornando informações sobre o *envelope*, a *conta*, o *valor* e o *status* do dito depósito.

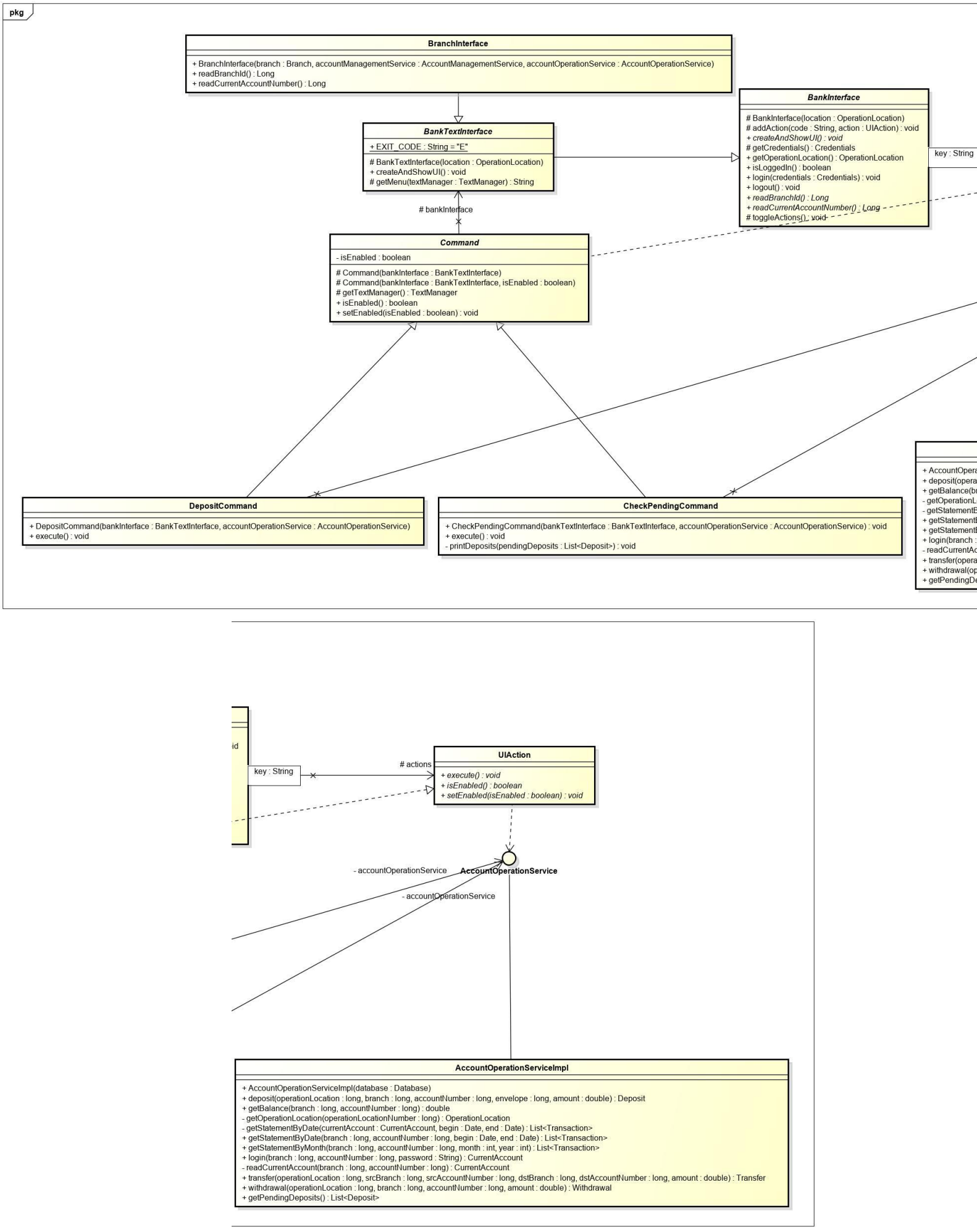
- *CurrentAccount.depositAmount(double amount) : void* -- Método Alterado - PÚBLICO

  Foi alterado o método de visibilidade privada para pública (para poder lidar com a adição de valores em um *Deposit.accept* sem passar por *CurrentAccount.deposit,* que criaria um Depósito, de maneira incorreta)

- *CurrentAccount.withdrawalAmount(double amount, boolean checkBalance) : void* -- Método Alterado - PÚBLICO

  Foi alterado o método de visibilidade privada para pública (para poder lidar com a remoção de valores de um *Deposit.reject* sem passar por *CurrentAccount.withdrawal*, que criaria uma operação incorreta). Também foi adicionado um parametro booleano *checkBalance*, que caso seja falso, pode deixar o saldo fique negativo.

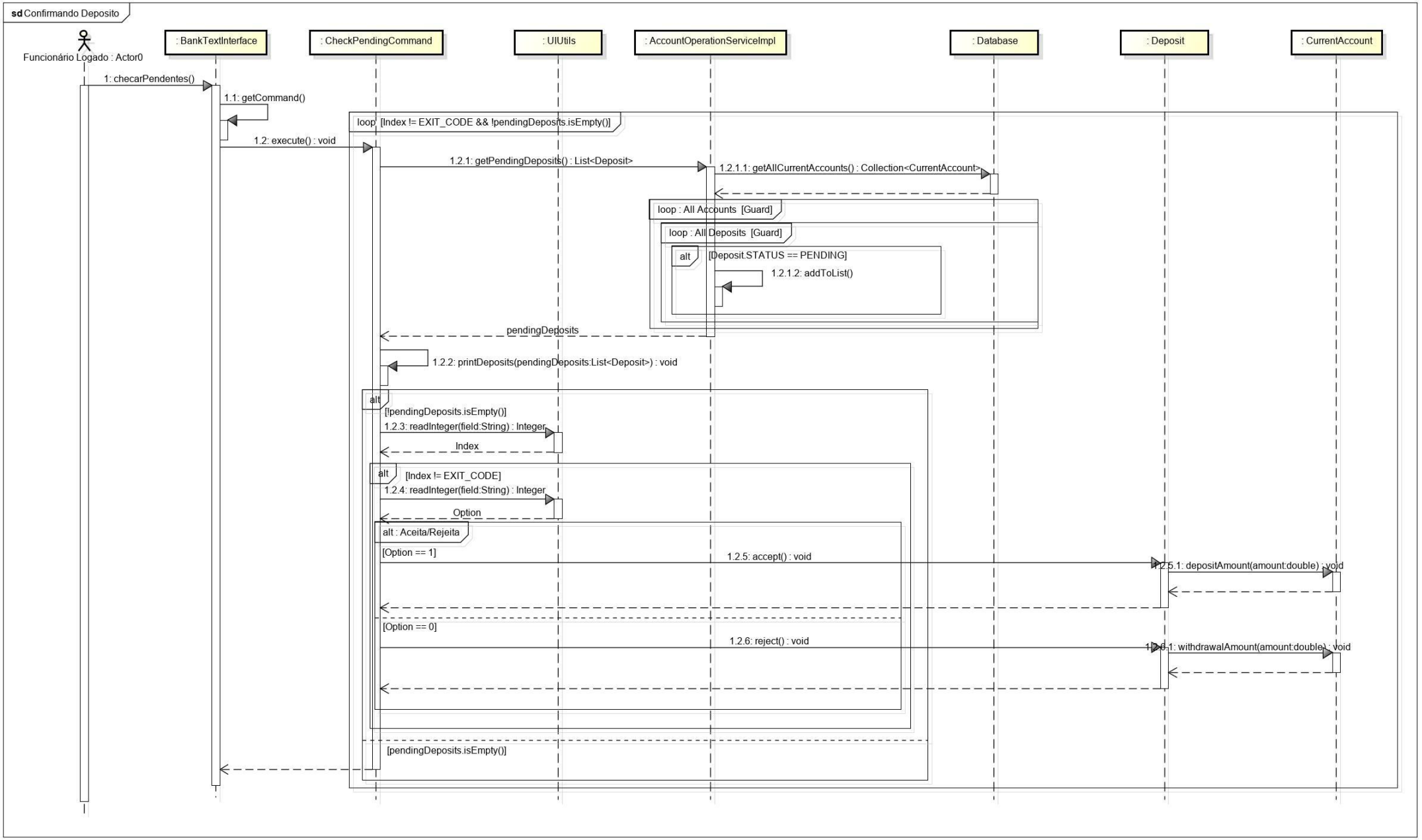

pkg
Database
+ Database()
+ Database(initData : boolean)
+ getAllCurrentAccounts() : Collection<CurrentAccount>
+ getAllEmployees() : Collection<Employee>
+ getAllOperationLocations() : Collection<OperationLocation>
+ getCurrentAccount(currentAccountId : CurrentAccountId) : CurrentAccount
+ getEmployee(username : String) : Employee
+ getNextCurrentAccountNumber() : long
+ getOperationLocation(number : long) : OperationLocation
- initData() : void
- changeDate(t : Transaction, r : Random, cal : Calendar) : void
+ save(currentAccount : CurrentAccount) : void
+ save(employee : Employee) : void
+ save(operationLocation : OperationLocation) : void
OperationLocation
- number : long
+ OperationLocation(number : long)
+ equals(obj : Object) : boolean
+ getNumber() : long
+ hashCode() : int
+ toString() : String
key : Long
- operationLocations
- location
Transaction
- amount : double
+ getLocation() : OperationLocation
+ setDate(date : Date) : void
key : CurrentAccountId
- currentAccounts
<<enum>>
STATUS
- <<enum>> PENDING : int
- <<enum>> REJECTED : int
- <<enum>> ACCEPTED : int
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) : 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
- accoun
Deposit
- envelope : long
- status : STATUS
- LIMIT : Integer = 100
+ Deposit(location : OperationLocation, account : CurrentAccount, envelope : long, amount : double)
+ getEnvelope() : long
+ accept() : void
+ reject() : void
+ isPending() : boolean
+ getStatus() : STATUS
+ toString() : String
*
{od}
- deposits

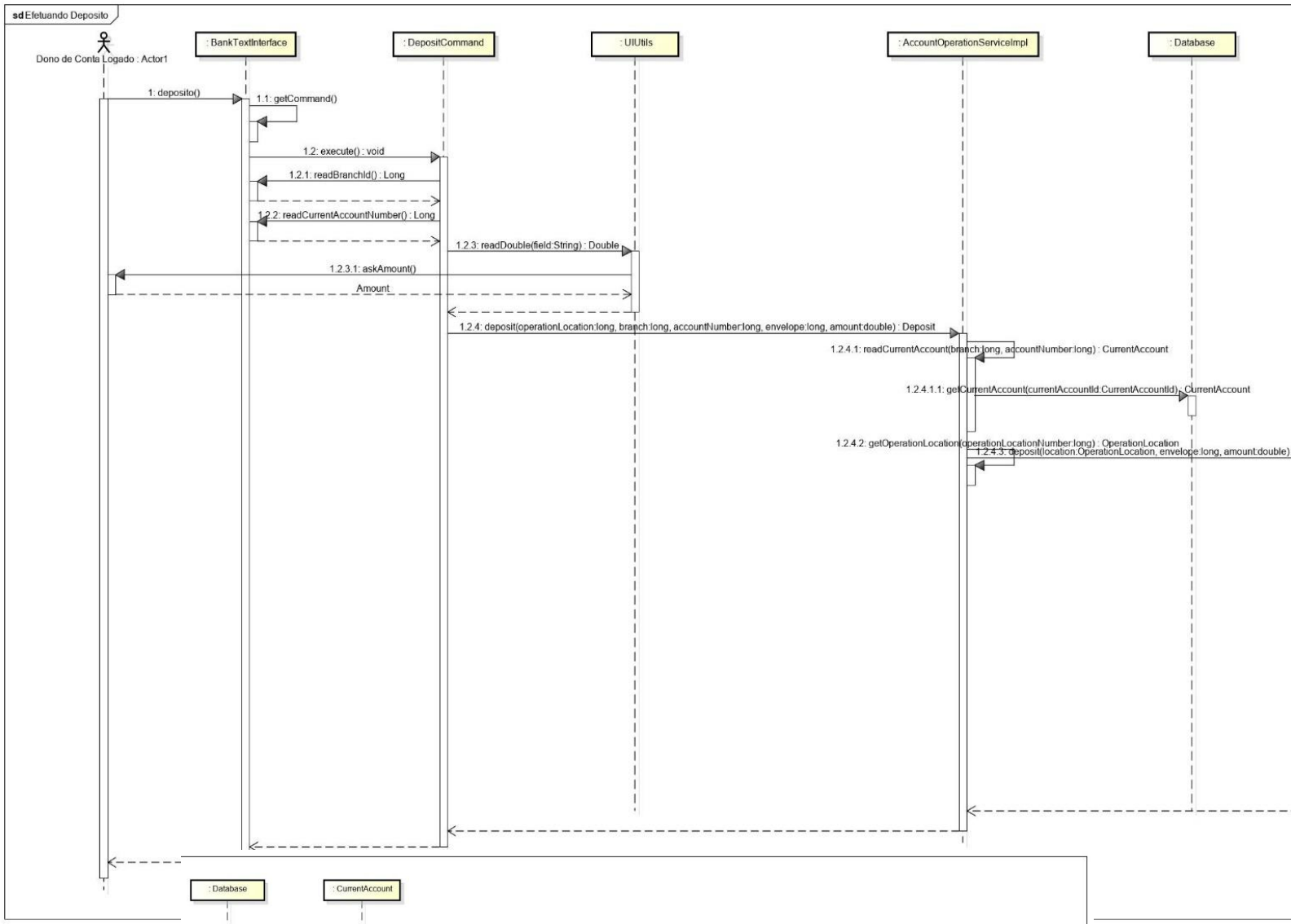

pkg
BranchInterface
+ BranchInterface(branch : Branch, accountManagementService : AccountManagementService, accountOperationService : AccountOperationService)
+ readBranchId() : Long
+ readCurrentAccountNumber() : Long
BankTextInterface
+ EXIT_CODE : String = "E"
# BankTextInterface(location : OperationLocation)
+ createAndShowUI() : void
# getMenu(textManager : TextManager) : String
# bankInterface
BankInterface
BankInterface(location : OperationLocation)
addAction(code : String, action : UIAction) : void
+ createAndShowUI() : void
# getCredentials() : Credentials
+ getOperationLocation() : OperationLocation
+ isLoggedIn() : boolean
+ login(credentials : Credentials) : void
+ logout() : void
+ readBranchId() : Long
+ readCurrentAccountNumber() : Long
# toggleActions() : void
key : String
Command
- isEnabled : boolean
Command(bankInterface : BankTextInterface)
Command(bankInterface : BankTextInterface, isEnabled : boolean)
getTextManager() : TextManager
+ isEnabled() : boolean
+ setEnabled(isEnabled : boolean) : void
DepositCommand
+ DepositCommand(bankInterface : BankTextInterface, accountOperationService : AccountOperationService)
+ execute() : void
CheckPendingCommand
+ CheckPendingCommand(bankTextInterface : BankTextInterface, accountOperationService : AccountOperationService) : void
+ execute() : void
- printDeposits(pendingDeposits : List<Deposit>) : void

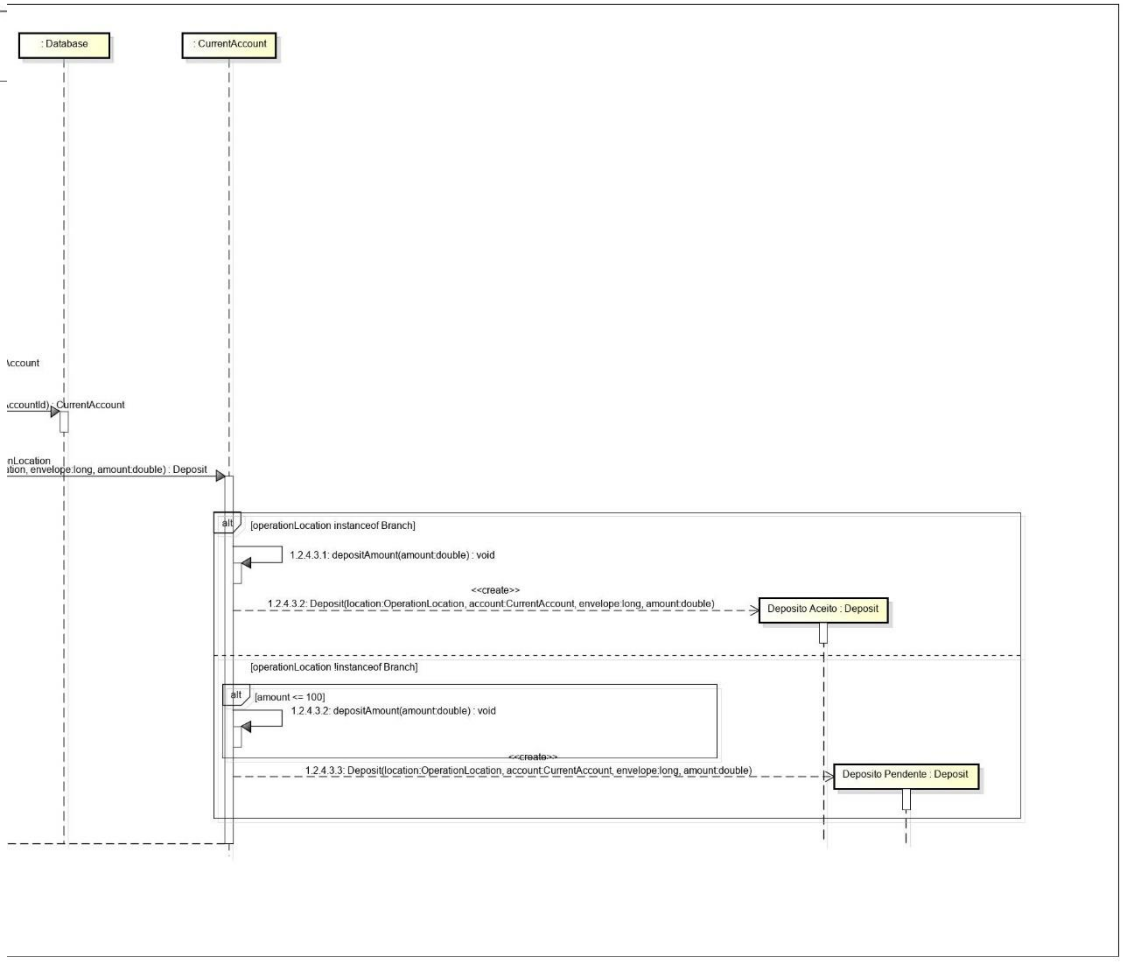
key : String
actions
UIAction
+ execute() : void
+ isEnabled() : boolean
+ setEnabled(isEnabled : boolean) : void
- accountOperationService
AccountOperationService
- accountOperationService
AccountOperationServiceImpl
+ AccountOperationServiceImpl(database : Database)
+ deposit(operationLocation : long, branch : long, accountNumber : long, envelope : long, amount : double) : 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
- readCurrentAccount(branch : long, accountNumber : long) : CurrentAccount
+ transfer(operationLocation : long, srcBranch : long, srcAccountNumber : long, dstBranch : long, dstAccountNumber : long, amount : double) : Transfer
+ withdrawal(operationLocation : long, branch : long, accountNumber : long, amount : double) : Withdrawal
+ getPendingDeposits() : List<Deposit>

II. DIAGRAMA DE SEQUÊNCIAS

Abaixo se encontram os diagramas de sequência em modo paisagem, para mais fácil leitura. Eles representam, respectivamente, a função modificada de criar um depósito e a função nova de autorizar ou rejeitar os depósitos pendentes.


sd Confirmando Deposito
Funcionário Logado : Actor0
: BankTextInterface
: CheckPendingCommand
: UIUtils
: AccountOperationServiceImpl
: Database
: Deposit
: CurrentAccount
1: checarPendentes()
1.1: getCommand()
loop [Index != EXIT_CODE && !pendingDeposits.isEmpty()]
1.2: execute() : void
1.2.1: getPendingDeposits() : List<Deposit>
1.2.1.1: getAllCurrentAccounts() : Collection<CurrentAccount>
loop : All Accounts [Guard]
loop : All Deposits [Guard]
alt [Deposit.STATUS == PENDING]
1.2.1.2: addToList()
pendingDeposits
1.2.2: printDeposits(pendingDeposits:List<Deposit>) : void
alt [!pendingDeposits.isEmpty()]
1.2.3: readInteger(field:String) : Integer
Index
alt [Index != EXIT_CODE]
1.2.4: readInteger(field:String) : Integer
Option
alt : Aceita/Rejeita
[Option == 1]
1.2.5: accept() : void
1.2.5.1: depositAmount(amount:double) : void
[Option == 0]
1.2.6: reject() : void
1.2.6.1: withdrawalAmount(amount:double) : void
[pendingDeposits.isEmpty()]


sd Efetuando Deposito
Dono de Conta Logado : Actor1
: BankTextInterface
: DepositCommand
: UIUtils
: AccountOperationServiceImpl
: Database
1: deposito()
1.1: getCommand()
1.2: execute() : void
1.2.1: readBranchId() : Long
1.2.2: readCurrentAccountNumber() : Long
1.2.3: readDouble(field:String) : Double
1.2.3.1: askAmount()
Amount
1.2.4: deposit(operationLocation:long, branch:long, accountNumber:long, envelope:long, amount:double) : Deposit
1.2.4.1: readCurrentAccount(branch:long, accountNumber:long) : CurrentAccount
1.2.4.1.1: getCurrentAccount(currentAccountId:CurrentAccountId) : CurrentAccount
1.2.4.2: getOperationLocation(operationLocationNumber:long) : OperationLocation
1.2.4.3: deposit(location:OperationLocation, envelope:long, amount:double)


: Database
: CurrentAccount
alt
[operationLocation instanceof Branch]
1.2.4.3.1: depositAmount(amount:double) : void
<<create>>
1.2.4.3.2: Deposit(location:OperationLocation, account:CurrentAccount, envelope:long, amount:double)
Deposito Aceito : Deposit
[operationLocation !instanceof Branch]
alt
[amount <= 100]
1.2.4.3.2: depositAmount(amount:double) : void
<<create>>
1.2.4.3.3: Deposit(location:OperationLocation, account:CurrentAccount, envelope:long, amount:double)
Deposito Pendente : Deposit