Construção de Sistemas de Software

SaleSys - Projecto 1

Grupo 018
António Rodrigues, 40853
João Rodrigues, 45582
Simão Neves, 45681

Índice

Introdução

De modo a implementar as funcionalidades de *fecho de venda*, *efectuar pagamento* e *consulta de conta corrente*, decidimos que teriam que ser efectuadas algumas modificações. São em seguida apresentadas as alterações efectuadas em cada uma das camadas, em cada uma das implementações. Contudo, visto existirem alterações comuns às três diferentes implementações serão de imediato apresentadas as alterações comuns seguidas das diferentes alterações em cada uma das implementações.

Alterações Comuns

Base de dados

Na base de dados decidimos adicionar uma tabela que representásse a conta corrente de um *customer*, e sendo que cada um pode apenas ter uma conta corrente não sentimos a necessidade de adicionar uma entidade (tabela) *conta_corrente*, visto isto, a tabela adicionada tem o nome de *sale_transaction*, e representa uma transação gerada tanto por ter sido feito um pagamento como por ter sido efectuado o pagamento de uma *sale*. Esta tabela conta com as seguintes colunas:

- Id - identificador unívoco de uma transação
- SaleId - chave estrangeira para relação com sale
- Value - valor da transação
- Type - tipo da transação
- CreatedAt - data de criação da transação

Contudo caso existisse a remota hipotese de um customer poder a ter várias contas correntes, a entidade (tabela) *conta_corrente*, seria criada nesta fase para evitar que futuras alterações se tornassem penosas caso viessem a ser necessárias.

Transaction Script

Sendo este padrão, um padrão focado em operações, o seu desenvolvimento foi mais rápido e imediato, contudo pudemos facilmente reparar que caso a aplicação viesse futuramente a oferecer funcionalidades actualmente não previstas, a sua inserção poderia tornar-se dispendiosa em termos de esforço/tempo. Neste modelo começámos a sentir a necessidade de atribuir responsabilidades a entidades, de modo a que estas se fossem encaradas como especialistas nessa/as operação/ões. São em seguida apresentadas as alterações efectuadas em cada uma das camadas.

Apresentação

Adicionadas:

- TransactionService.java
    - getTransactionDetails(int transactionId)
      Obtenção de uma Transaction pelo seu identificador unívoco.

Modificadas:

- SaleService.java
    - closeSale(saleId)
      Fecho de uma venda com base no seu identificador unívoco.

    - makePayment(int saleId, double amount)
      Registo de pagamento de uma determinada venda com base no seu identificador unívoco.

- CustomerService.java
    - getAccount(int vat)
      Obtenção de conta corrente de um customer com base no seu VAT.

Lógica de negócio

Adcionadas:

- SaleTransactionTransactionScripts.java
    - getTransactionDetails(int transactionId)
      Obtenção de uma transação com base no seu identificador unívoco.
      O tipo de objecto retornado depende do valor da coluna type da linha obtida. Pode ser retornada uma *CreditTransaction* ou uma *DebitTransaction*.

Modificadas:

- SaleTransactionScripts.java
    - closeSale(int saleId)
      Marca a sale como fechada e gera uma transação de débito com o valor da sale a fechar com base no seu identificador unívoco.

    - makePayment(int saleId, double amount)
      Marca a sale como paga e gera uma transação de crédito com o valor da sale com base no seu identificador unívoco.

- CustomerTransactionScripts.java
    - getAccountInfo(int customerId)
      Obtenção da informação da conta corrente de um customer, ou seja, a lista de transações associadas às suas sales, com base no seu identificador unívoco.

Acesso aos dados

Adicionadas:

- SaleTransactionRowDataGateway.java
    - insert()
      Persistência de objecto de acordo com o valor dos seus atributos.

    - getTransactionById(int transactionId)
      Obtenção de uma SaleTransactionRowDataGateway com base no seu indentificador unívoco.

    - getSaleTransactions(int saleId)
      Obtenação de todas as transações de uma sale com base no seu identificador unívoco.

Modificadas:

- SaleRowDataGateway.java
    - update()
      Actualizar a sale em base de dados com base no valor actual dos seus atributos.

    - getSalesByCustomerId(int customerId)
      Obtenção de todas as sales de um determinado customer com base no seu identificador unívoco.

Para além das já referidas alterações foram ainda adicionadas algumas classes ao igualmente adicionado *package domain*. Esta necessidade surge devido ao conteúdo exigido pelo cliente, e pela necessidade de garantir que esta não tem acesso à camada de persistência de dados.
Para tal foram criadas os classes *Account*, *Transaction*, *CreditTransaction*, *DebitTransaction*, *SaleProduct*. Estas classes têm acesso aos mesmo campos que *SaleTransactionRowDataGateway*(*Transaction*), *SaleProductRowDataGateway*(*SaleProduct*), contudo sem acesso à manipulação de dados persistidos.

Table Module

O padrão Table Module assenta na base de ter objectos únicos para cada tipo, que representam tabelas da base de dados na sua totalidade, e não objectos que representem rows de dados na base de dados, como estamos habituados. Ora este facto, aliado ao uso do padrão TableDataGateway que devolve dados tabulares (normalmente ResultSets, mas neste caso foi o TableData) dificultou a implementação das operações requisitadas no enunciado.
De seguida apresentam-se as alterações feitas ao projecto usando estes padrões.

Apresentação

Modificadas:

- SaleService.java
    - closeSale(int saleId)
      Fecho de uma venda com base no seu identificador unívoco.

    - makePayment(int saleId)
      Registo de pagamento de uma determinada venda com base no seu identificador unívoco.

- CustomerService.java
    - showCustomerCurrentAccount(int vat)
      Operação onde se obtém a conta corrente de um Customer com base no seu VAT e se pode listar os seus detalhes.

Lógica de negócio

Adcionadas:

- Transaction.java
  - newTransaction(int saleId, double value, TransactionType type)
    Cria uma nova Transacção para a Sale com id saleId na base de dados e devolve o novo id.
  - getAllTransactionsFromSale(int saleId)
    Devolve um TableData com Rows de todas as Transacções referentes à Sale com id saleId
  - print(TableData.Row row)
    Devolve uma String que representa textualmente uma Transacção.

- DebitTransaction.java (extends Transaction)
  - newTransaction(int saleId, double value)
    Cria uma nova Transação na base de dados, do tipo DEBIT

- PaymentTransaction.java (extends Transaction)
  - newTransaction(int saleId, double value)
    Cria uma nova Transação na base de dados, do tipo CREDIT

- TransactionType.java
  Enumerado com os diferentes tipos de Transacções, CREDIT ou DEBIT

- CustomerAccount.java
  Objecto que representa uma conta corrente, com uma lista de Rows, que são Transações
  - addTransactions(TableData ts)
    Adiciona todos Rows de ts a uma lista interna do objecto

  - print()
    Retorna uma String com todas as representações textuais de todas as Transações que tem na sua lista

  - computeTotal()
    Calcula, com base nas Transações que tem na sua lista, e retorna o total que o Customer desta CustomerAccount deve.

- printTransaction(int index)
  Retorna uma String que representa os detalhes da Transacção na posição index da lista interna, com base no seu tipo (chama printDebitTransactionDetails ou printCreditTransactionDetails).

- printDebitTransactionDetails(TableData.Row row)
  Retorna uma String que representa todos os SaleProducts da venda que está associada à Transacção representada por row.

- printCreditTransactionDetails(TableData.Row row)
  Retorna uma String que representa a Sale que está associada à Transacção representada por row.

Modificadas:

- SaleStatus.java
  Adicionou-se um novo Status, o PAYED, que indica que uma Sale foi paga

- Customer.java
  - getCustomerCurrentAccount(int vat)
    Valida o VAT do Customer e vai buscar à base de dados todas as Transacções e coloca-as num objecto CustomerAccount.

- Product.java
  - getFaceValue(int productId)
    Devolve um double que é o preço de um produto, vai buscar essa informação à base de dados com base no seu identificador único.

- Sale.java
  - isClosed(int saleId)
    Alterada lógica para ver se a Sale está mesmo CLOSED, porque foi adicionado mais um Status para a Sale, PAYED, se ela estiver PAYED então também está CLOSED

  - makePayment(int saleId)
    Operação de realizar um pagamento.
    Verifica que a Sale está fechada mas não paga, se isto for verdade coloca a Sale como PAYED e retorna o Id de uma nova uma Transação colocada na base de dados

    - isPayed(int saleId)
      Verifica se uma Sale está com Status PAYED

    - updateSale(int saleId, SaleStatus status, double total, double discount)
      Faz update de uma Sale na base de dados com os dados recebidos

    - closeSale(int saleId)
      Operação de fecho de uma Sale, calcula os valor total da Sale bem como o seu desconto, actualiza a Sale na base de dados com os novos valores e com o Status CLOSED. De seguida retorna o Id da nova Transacção que cria na base de dados

    - getAllSaleIdsFromCustomer(int customerId)
      Retorna uma lista de todos os ids de Sales de um Customer com base no seu id único

    - print(TableData.Row row)
      Retorna uma String que é a representação textual de uma Sale

- SaleProduct.java
    - getSaleProductsFromSale(int saleId)
      Retorna TableData com todos os SaleProducts de uma Sale com base no seu id único

    - print(TableData.Row row)
      Retorna uma String que representa textualmente um SaleProduct

Acesso aos dados

Adicionadas:

- TransactionTableDataGateway.java
    - newTransacion(int saleId, double value, TransationType type)
      Insere uma nova Transacção na base de dados com os dados recebidos e retorna o seu id

    - getAllTransactions(int saleId)

Devolve um TableData que contém todas as Transações de uma Sale com base no seu id

- readId(Row row)
  Lê e retorna o atributo Id de uma Row que represente uma Transação

- readSaleId(Row row)
  Lê e retorna o atributo saleId de uma Row que represente uma Transação

- readType(Row row)
  Lê e retorna o atributo Type de uma Row que represente uma Transação

- readCreatedAt(Row row)
  Lê e retorna o atributo createdAt de uma Row que represente uma Transação

- readValue(Row row)
  Lê e retorna o atributo Value de uma Row que represente uma Transação

Modificadas:

- SaleTableDataGateway.java
  Adicionou-se um novo Status para ser guardado na base de dados

  - updateSale(int saleId, SaleStatus status, double total, double discount)
    Converte o Status que recebe em String de 1 caractére para guardar na base de dados. Actualiza a Sale com base nos paramêtros recebidos.

  - getAllSalesFromCustomer(int customerId)
    Retorna TableData com todas as Sales que pertencem a um Customer, com base no seu id único

A decisão mais relevante talvez seja a criação de um classe CustomerAccount que representa a conta corrente de um Customer, que lá dentro tem uma lista de Rows de Transações.
Este objecto foi criado para se poder manipular e encapsular todos os detalhes de uma conta corrente. Visto que não existe uma tabela CustomerAccount na base de dados, este objecto não é um Table Module.

Domain Model

Sendo este um padrão totalmente orientado a objectos, aproveitámos para inserir a entidade Account. Deste modo cada Customer, tem a sua Account que por sua vez tem Sales e esta por sua vez tem Transactions. Tentámos minimizar a complexidade inerente das queries que são feitas, nomeadamente ao obter as transactions de um customer, visto que para as obtermos basta apenas perguntar ao customer pela sua account, e a esta pelas suas transactions. Em seguida apresentamos as alterações efectuadas para a implementação dos novos casos de uso requeridos.

Apresentação

Adicionadas:

- CurrentAccountService.java
    - validateCustomer(vat)
      Validar o customer consoante o numero de Vat indicado
    - getAllTransations (vat)
      Objectivo é ir buscar todas as transactions que o customer já realizou
    - seeTransation (idTran)
      Dado uma transaction realiza o display de mais informação consoante o seu tipo.

Modificadas:

- ProcessSaleService
    - closeSale (vat)
    - paySale (vat, idSale)

Lógica de negócio

Adcionadas:

- CurrentAccountHandler.java - classe que dá suporte ao caso de uso consultar a conta corrente de um Customer. É a classe que contém a lógica associada às suas operações. É esta classe a responsável pela chamada da camadas de persistencia (catálogos) para que através destas possa obter Transactions. Os seus métodos são os seguintes:
    - getAllTransations(vat) - visualizar todas as transações
    - seeTransation(Transation) - visualizar uma transação
    - validateCustomer(vat) - validar um customer

- Account.java - classe que representa a conta corrente de um customer em que tem o seu balance e uma lista de transations com @OneToMany(cascade = ALL) @JoinColumn(name = "ACCOUNT_ID"). Isto permite associar à tabela de Transations uma coluna com o nome ACCOUNT_ID que tem o id da account. Foi introduzido uma @NamedQuery para procurar uma account pelo seu ID.

- Transation.java - Classe abstrata que representa uma transação. É utilizado uma factory para criar instâncias desta classe do tipo Credit ou Debit. Estas duas classes representam transactions do tipo Debit ou Credit respectivamente. A necessidade de criar duas classes que extendem um Transaction surge por diferenças evidentes em cada uma destas classes.
  Decidiu-se utilizar @Inheritance(strategy = SINGLE_TABLE) para gerar uma única tabela em que os três tipos de Objectos (Transaction, Debit, Credit). O JPA adiciona uma coluna extra à tabela com o tipo de objeto para seu mapeamento interno.
  Foram introduzidas duas @NamedQuery uma para procurar uma transaction pelo seu ID e outra para as encontrar a todas.
- Credit.java - transaçao do tipo credit
- Debit.java - transação do tipo de debito
- PaymentStatus.java - Representa o estado de um pagamento de uma Sale, Payed ou Not_Payed

Modificadas:

- Sale.java - Uma Sale passou a ter também uma lista de transations e foi anotada com (@OneToMany(cascade = ALL) @JoinColumn). Isto permite que a tabela

Transaction é que tenha o id da sale e nãoo cria uma tabela suplementar de Sale_Transation.
Tem também um PaymentSatus para verificar se a Sale já foi paga ou nao. Para representá-la na BD utilizou-se @Enumerated(EnumType.STRING) para presisti-la em String.

- Customer.java - um customer passou a ter uma account (@OneToOne)

Acesso aos dados

Adicionadas:

- AccountCatalog.java - classe que permite interagir com a base de dados para as accounts.
    - addAccount(balance) metodo para adicionar uma conta com determinado balance
    - addAccount() metodo para adicionar uma conta com balance a zero.
    - updateAccount(Account) metodo para realizar o update de uma account
    - getAccount(id) metodo para ir buscar uma dada account consoante o id passado

- TransationCatalog.java
    - addTransation(Transation) metodo para adicionar uma transaction
    - addTransation(String, Date, Sale, Account) metodo que cria uma transação e adiciona-a à base de dados
    - getTransation(id) metodo para ir buscar uma dada transação consoante o id passado
    - getTransations() metodo para ir buscar todas as transações
    - updateTransation(Transation) fazer o update da transação
    - addTransationToSale(Transation, Sale, Customer) consistem em adicionar uma transação a uma sale

Modificadas:

- SaleCatalog
    - updateSale(Sale) update de uma dada sale
    - getSale(id) metodo para ir buscar uma dada sale consoante o id fornecido