INSTITUIÇÃO DE ENSINO SUPERIOR
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

Relatório de Projeto da disciplina de Laboratório de Programação 2

Victor Eduardo Borges de Araújo
Wesley Gonçalves Aníbal
Dayvson Weslley Cantalice do Nascimento
Lucas Arcoverde do Nascimento

Prof. Matheus Gaudencio do Rêgo
Orientador

Jhonatan Batista
Monitor

Campina Grande, 12 de Outubro de 2016

Introdução

Neste relatório, explicaremos o design do nosso projeto organizando-o por caso de uso, explicando as características principais da solução para implementar cada caso, enfatizando os padrões de projeto usados, além de mostrar onde foram utilizados os principais tópicos visto na disciplina

- **Caso de Uso 1: CRUD de hóspede**

Neste primeiro caso, criamos uma classe representando um hóspede com as variáveis: Nome, Email e Data de nascimento, todas do tipo String. Mais à frente, adicionamos o cartão ao hóspede e uma lista de estadias, que serão explicados em casos de uso posteriores. Também foram criados os métodos de getters e setters, para mudança dos valores das variáveis e para retorno desses valores.

Também foi criado uma classe para Exception de hóspede inválido, na qual herda da classe SistemaInvalidoException que por sua vez herda de Exception para definir uma Exception para qualquer Exception que pode ocorrer devido ao sistema, e define a exception do tipo HospedeInvalidoException, que pode ser lançado/capturado quando há uma criação de um hospede invalido, que pode ser por data de nascimento em formato errado,email em formato errado ou nome,ou os com valor “null”.

- **Caso de Uso 2: Checkin de hóspedes**

O caso de uso 2, é marcado pela criação do quarto e de um enumeration representando os valores de cada quarto. Um quarto possui uma Id e um valor que é definido pelo enum, também criamos uma classe chamada estadia, a qual possui um quarto, uma quantidade de dias do tipo int, e os gastos do tipo double. O hóspede possui uma lista de estadias, na qual a cada check in é adicionado uma nova estadia com um valor. Também criamos a HotelController, responsável por controlar toda lógica do hotel, o cadastro de hóspedes, o checkin e checkout, entre outros, sendo ela a classe central do sistema. Por fim, criamos também a Façade, para o usuário não ter acesso ao sistema, e sim só a uma “porta”, ou como o próprio nome já diz, uma fachada.

Também foi criada a classe HotelInvalidoException,que herda de SistemaInvalidoException, que encapsula todo erro que pode ocorrer devido ao HotelController.

- **Caso de Uso 3: Checkout de hóspedes e histórico de lucros**

Neste caso de uso, primeiramente, nós buscamos nos dados do hóspede que está realizando o checkout, a estadia referente ao quarto que ele está desocupando. Com isso obtemos a informação necessária sobre o preço total da transação. Para guardar esses dados (id do quarto e preço total) em um histórico, nós criamos a classe Registrador que contém uma lista de gasto,na qual encapsulamos o gasto para aumentar a coesão e o expert,que contém o nome do hóspede,os gastos e as compras. Sendo assim, quando é realizado algum checkout, nós guardamos esses gastos em um objeto do tipo Registrador, que representa o histórico.

- **Caso de Uso 4: Restaurante do Hotel**

Neste caso de uso, criamos uma classe abstrata chamada Refeição, da qual herdam duas classes filhas, Prato e RefeiçãoCompleta, na qual RefeiçãoCompleta possui um ArrayList de Pratos. Refeição possui apenas um método abstrato, sendo ele o calculaPreço(). Esse método é implementado pelas duas classes filhas, sendo que no Prato ele retorna o preço real do prato, e na RefeiçãoCompleta, ele retorna o preço de todos os pratos do ArrayList de pratos somados, com um desconto de 10%. Além disso, criamos uma classe Cardápio, que irá possuir um List de Refeição, que serão todos os pratos e refeições completas cadastradas. Ao cadastrar uma RefeiçãoCompleta, fazemos uma verificação para checar se todos os pratos passados como parâmetro estão cadastrados no cardápio. Por fim, criamos uma classe RestauranteController, que possui um Cardápio e delega os métodos principais de Cardápio, na forma de uma Facade, uma vez que no estágio inicial, Restaurante e Hotel eram independentes.

Foi criada a classe RestauranteInvaliodoException, que herda de SistemaInvalidoException, que encapsula toda exception que pode ser ocorrida devido ao restaurante e ou sua criação, foram criadas também o PratoInvalidoException e RefeicaoInvalidoexception, que encapsula todas as exceptions que podem ocorrer no prato ou sua criação,e na refeição ou sua criação,respectivamente.

- **Caso de Uso 5: Organização do cardápio e pedidos**

Neste caso de uso, criamos uma classe Cardapio que possui uma lista de Refeições,. Para isso, utilizamos polimorfismo para guardar objetos do tipo Prato e RefeiçãoCompleta, que herdam da classe abstrata Refeição. Essa lista representa todas as Refeições que o restaurante possui.

Para que os hóspedes possam realizar pedidos, nós criamos o método consultaMenu que retorna todos os nomes das Refeições presentes no cardápio separados por ";".

Para facilitar o uso do cardápio pelos administradores, nós criamos duas classes que implementam a interface Comparator<T>, uma para comparar as Refeições por nome (em ordem crescente alfabética) e outra pelo preço (em ordem crescente de preço). A partir daí, criamos um método para ordenar o cardápio.

Para registrar os pedidos feito pelos hóspedes em um histórico foi necessário adicionarmos uma lista, na classe Registrador, que guarda o gasto que contém o nome da refeição (também guardará o Id dos quartos) referente a transação. Sendo assim, utilizamos o mesmo objeto do Tipo registrador que guardava o histórico de Checkout para armazenar também os pedidos e o preço da Refeição, basicamente da mesma forma que ocorre no realizaCheckout.

- **Caso de Uso 6: Cartão e Pontos de Fidelidade**

No caso de uso 6, nos utilizamos do Padrão Strategy. Criamos uma interface TipoDeCartaoIF e três classes que implementam-a, sendo elas Padrão, Premium e Vip. Cada uma delas implementará os métodos da interface, que são o addPontos() e o aplicaDesconto(), ao seu modo, possuindo ou não vantagens. Além disso, criamos uma classe CartãoFidelidade, que possui a quantidade de pontos do cartão e possui um objeto do tipo TipoDeCartaoIF, esse que poderá mudar o algoritmo aplicado em cada método em tempo de execução, sendo esse o benefício do uso do Padrão Strategy como vimos em sala de aula. Usamos o padrão pois o cliente do Hotel muda seu tipo de fidelidade de acordo com a quantidade de pontos, ou seja, um cliente que vira Vip, não será Vip para sempre, dependendo da quantidade de pontos que ele tem, ele poderá voltar a ser Premium ou até mesmo Padrão.

Foi criada a classe CartaoInvalidoException,que herda de SistemaInvalidoException, que encapsula o erro que pode ser gerado pelo cartão, como exemplo no método convertePontos,que se for passado uma quantidade maior de pontos para converter do que o cartão possui é lançado a exception do tipo CartaoInvalidoException.

- **Caso de Uso 7: Descontos e benefícios da fidelidade**

Para o caso de uso 7, utilizamos de toda estrutura já montada no Caso de Uso 6, assim, bastou adicionar um método na interface TipoDeCartaoIF, sendo esse o credito() (que será delegado pelo método convertePontos() na classe CartãoFidelidade) e implementamos o método em cada uma das classes Padrão, Premium e Vip, cada uma retornando o valor de acordo com as especificações do projeto. Além disso, no método convertePontos() na classe CartãoFidelidade (que delega o método converte() de acordo com o estado atual do tipo de fidelidade do hóspede) fazemos uma verificação do estado do tipo de fidelidade do hóspede, uma vez que ao converter pontos em dinheiro (ao fazer uma operação de crédito) ele perde os pontos que foram convertidos. Sendo assim, se um hóspede era Vip, e converteu um número de pontos alto que o fez ficar com a quantidade de pontos necessária para ser apenas Premium, ele assumirá o tipo de fidelidade Premium.

- **Caso de Uso 8: Geração de relatórios do sistema**

Criamos um fluxo para escrita de caracteres com “bufferização”, utilizamos quatro métodos : gravaArquivoHospede(), gravaArquivoPratosRefeicoes(), gravaArquivoRegistros() e gravaArquivoResumo(), para gerar quatro relatórios, sendo o primeiro referente aos hóspedes cadastrados no sistema, contendo seu email, nome e data de nascimento; o segundo referente aos pratos e refeições completas cadastradas no sistema, contendo o nome, preço, descrição do prato/refeição completa e no caso da refeição, os pratos que compõem a refeição; o terceiro referente ao histórico de transações do sistema, contendo o nome do hóspede, o valor gasto e detalhes sobre estes gasto; e finalmente o quarto relatório que é um resumo geral do estado do hotel.

- **Caso de Uso 9: Persistência do sistema**

Neste passo, foi feito a conclusão do projeto, na qual criamos a “persistência” do sistema, criando uma classe que representa um banco de dados, que salva e lê o arquivo hug.dat, salvamos neste arquivo o objeto HotelController, que é a base de todo o sistema, ao iniciar o sistema, com o método iniciaSistema(), ele procura o arquivo hug.dat, caso não exista ele cria, caso exista, ele lê e retorna a instância que estava no arquivo para o atributo do hotel, deixando o sistema no estado que estava antes, e ao fechar o sistema, com o método fechaSistema(), todo estado atual do sistema é salvo no arquivo hug.dat, para ser lido no próximo inicia sistema.