Universidade Federal de Campina Grande
Centro de Ciências e Tecnologia
Departamento de Sistemas e Computação
Disciplina: Laboratório de Programação II
Professora: Lívia Sampaio

Relatório de Projeto

Hotel Urbano de Gotemburgo

Equipe:

Gabriel Valentino Botelho Alves
João Carlos Fernandes Bernardo
Melissa Diniz Gonçalves
Thaís Nicoly Araújo Toscano

Campina Grande
2016

Introdução

O Sistema desenvolvido visa auxiliar na administração do Hotel Urbano de Gotemburgo (HUG), através dele o gerente e colaboradores do hotel poderão ter acesso aos dados de transações do hotel, como checkin, checkout e os pedidos feitos no restaurante do mesmo; realizar operações como cadastro e remoção de hóspedes e suas estadias, também será possível atualizar informações já cadastradas, além de fornecer meios de acesso a gerência das refeições do restaurante e do cartão de fidelidade.

Pretendemos através deste relatório detalhar como o sistema foi desenvolvido. O mesmo será dividido em nove casos de uso, onde iremos abordar o padrão/design de desenvolvimento escolhido e os principais tópicos estudados nas disciplinas de Programação II e Laboratório de Programação II.

Sobre o design

Neste projeto, foi implementado dois padrões de design de orientação a objetos: Model Controller e Façade Design Patterns. Os primeiros são oriundos de um design ainda mais complexo, o Model View Controller (MVC), cujas ideias centrais são o reuso de código e a delegação de conceitos e/ou tarefas.
A utilização desse modelo permitiu solucionar o problema através da separação das tarefas de acesso aos dados da lógica de negócio, introduzindo um componente entre os dois: o controlador.
Com o padrão Façade podemos simplificar a utilização de um subsistema complexo apenas implementando uma classe que fornece uma interface única e mais razoável, porém se desejarmos acessar as funcionalidades de baixo nível do sistema isso é perfeitamente possível.
Vale ressaltar que o padrão Façade não "encapsula" as interfaces do sistema, ele apenas fornece uma interface simplificada para acessar as suas funcionalidades.
Com relação as Exceptions, criamos uma package chamada exceptions , nela possui todas as classes criadas para o encapsulamento de determinada situação de exceção. Por exemplo, CadastroHospedeInvalidoException encapsula todas as exceções relacionadas ao cadastro dos hóspedes no Hotel. Além disso, foi criada no mesmo pacote uma classe Verificá Exceção , ela possui métodos estáticos para verificação de parâmetros específicos, e ainda retorna uma mensagem específica (Erro no cadastro de Hóspede.) toda vez que essa exceção for lançada.

Caso 1

Inicialmente foi criado uma classe “Hotel” para o gerenciamento dos dados e transações do mesmo. Nele contém métodos para o cadastro, remoção e edição/atualização dos dados de um hóspede. Esses dados são armazenados em um HashMap, solução lógica encontrada já que o e-mail (chave do map) é utilizado durante grande parte da especificação do projeto para a busca dos hóspedes no sistema e a ordem em que eles foram adicionados é irrelevante. Além disso utilizamos também um hashmap para armazenar os quartos vazios pelo ID(chave).

Em seguida, foi criada a classe “Hóspede” que encapsula todos os atributos do mesmo, além disso, como um hóspede pode ter diversas estadias, há um linkedHashMap de estadias associado a ele, tal escolha de coleção foi devido ao fato de cada quarto está associado a apenas uma estadia (ou seja pode ser utilizado o ID do quarto como chave para encontrar a estadia associada a ele) e pela ordem de adição ser relevante, pois o mesmo mantém a ordem.

Uma importante aquisição desse caso é fazer com que os dados do hóspede fiquem mantidos no sistema do Hotel mesmo que ele não esteja mais hospedado, com isso, caso o hóspede volte a querer usufruir dos confortos deste Hotel seus dados ainda permanecem, e só precisará realizar o processo de atualização dos dados, caso necessário.

Caso 2

Durante a realização do check in do hóspede no Hotel, é associado um quarto (livre) à uma nova estadia (retira-se o quarto do mapa de quartos vazios) e integrando a mesma ao hóspede que está realizando tal operação.

Na criação dos quartos é utilizado herança, onde os quartos “Luxo” e “Presidencial” herdam de “Quarto Simples” (sendo que essa classe não é abstrata, e sim outro tipo de quarto), sendo que a diferença entre eles, é unicamente, o preço de suas diárias.

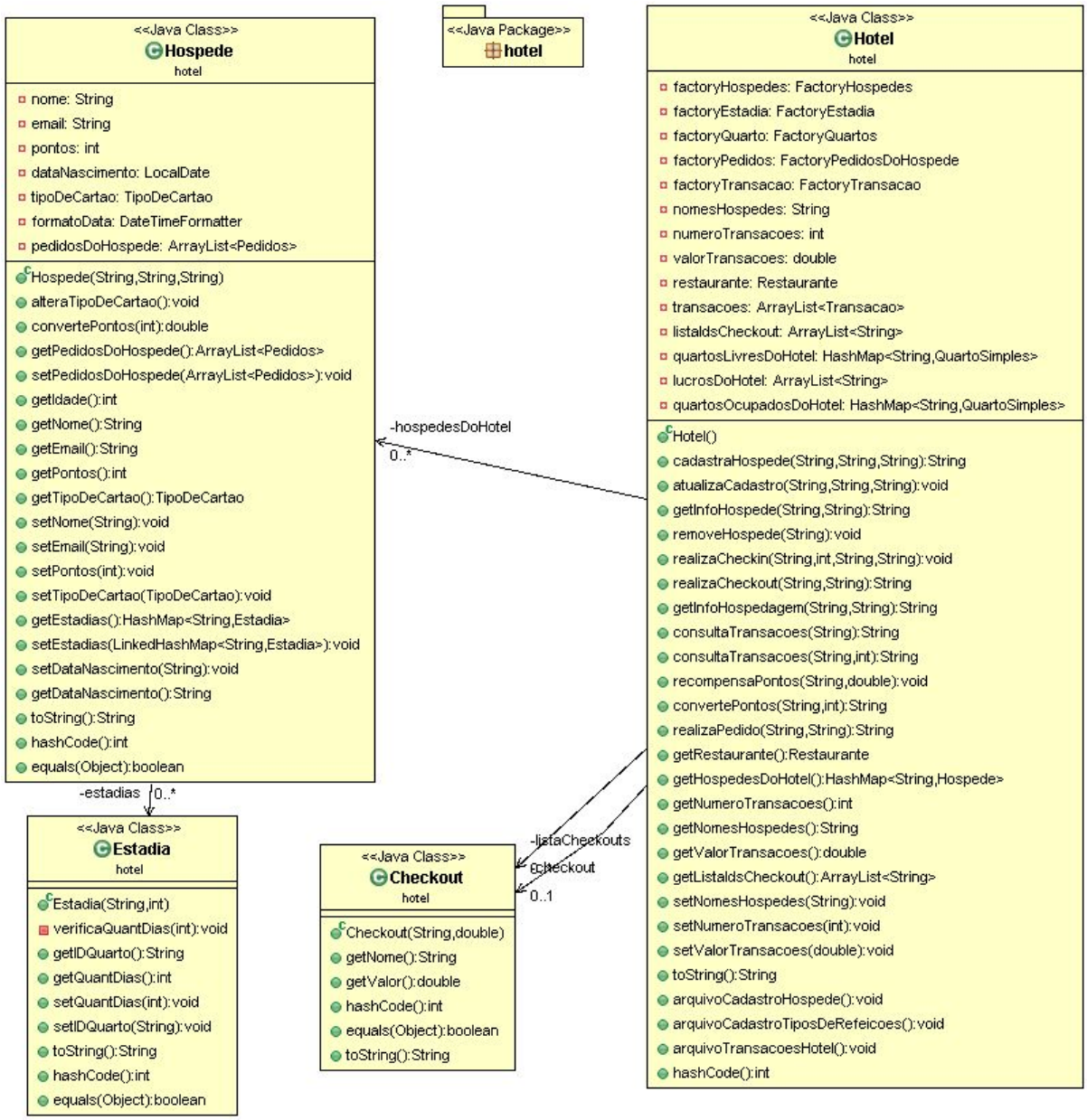
As estadias, que são criadas quando se obtém a criação do hóspede, encapsulam o total gasto pelo hóspede, que é calculado a partir da quantidade de dias e do tipo do quarto na qual ele se hospedará. Um mesmo hóspede poderá estar associado a várias estadias, ou seja, se uma família se hospedar apenas um responsável terá seus dados cadastrados no hotel, contudo, o processo de checkout é individual de cada estadia.

Caso 3

Para o hóspede deixar o hotel, é preciso que ele realize antes o processo de checkout, para isso é necessário que o administrador do hotel busque suas informações no sistema. O checkout do sistema é um objeto a qual está encapsulado os dados da estadia, de cada hóspede, que foram encerrados.

O objetivo principal do processo de checkout é registrar o histórico de lucros do Hotel. Esse processo necessita conter a data que o sistema realizou o checkout (utilizamos o LocalDate.now()) , o nome do hóspede, o número do quarto que estava associado a sua estadia e o total pago pelo mesmo.

Vale ressaltar que o processo de checkout no nosso sistema não está ligado à atualização dos dados do hóspede, pois o mesmo representa a transação dos dados pertencentes àquele registro. Pelo fato de um mesmo hóspede está associado a várias estadias, o processo de checkout será feito um para cada estadia a qual o hóspede possui.



Caso 4

O Hotel Urbano de Gotemburgo possui um restaurante exclusivo para os hóspedes cadastrados. Este Restaurante contém pratos (que possui nome, preço e descrição como atributos) e refeições (que são um conjunto de 3 ou 4 pratos), para isto foi criada uma classe intitulada de “TiposDeRefeicoes”, sendo superclasse não abstrata dos tipos “Prato” e “Refeicao”. Essa herança é necessária para que pratos e refeições pudessem ser instanciadas com o mesmo tipo (upcast), possibilitando o restaurante ter apenas uma lista armazenando pratos e refeições.

O Restaurante inicialmente é responsável por cadastrar Pratos e Refeições, atualizar um prato ou uma refeição, assim como remover, e também imprimir um menu de todos os Pratos e Refeições disponíveis.

Caso 5

Para facilitar a exibição do cardápio, pelos administradores do sistema, aos hóspedes , é utilizada duas formas de ordenação: Ordem crescente alfabética pelo nome do prato/refeição completa e, ordem crescente de preço de cada prato/refeição completa.

Após isso, quando o hóspede realizar o pedido, o mesmo será registrado com isso, é necessário que o mesmo informe o nome do prato ou refeição completa. O valor gasto pelo hóspede será registrado diretamente no histórico de transações do sistema.

Caso 6

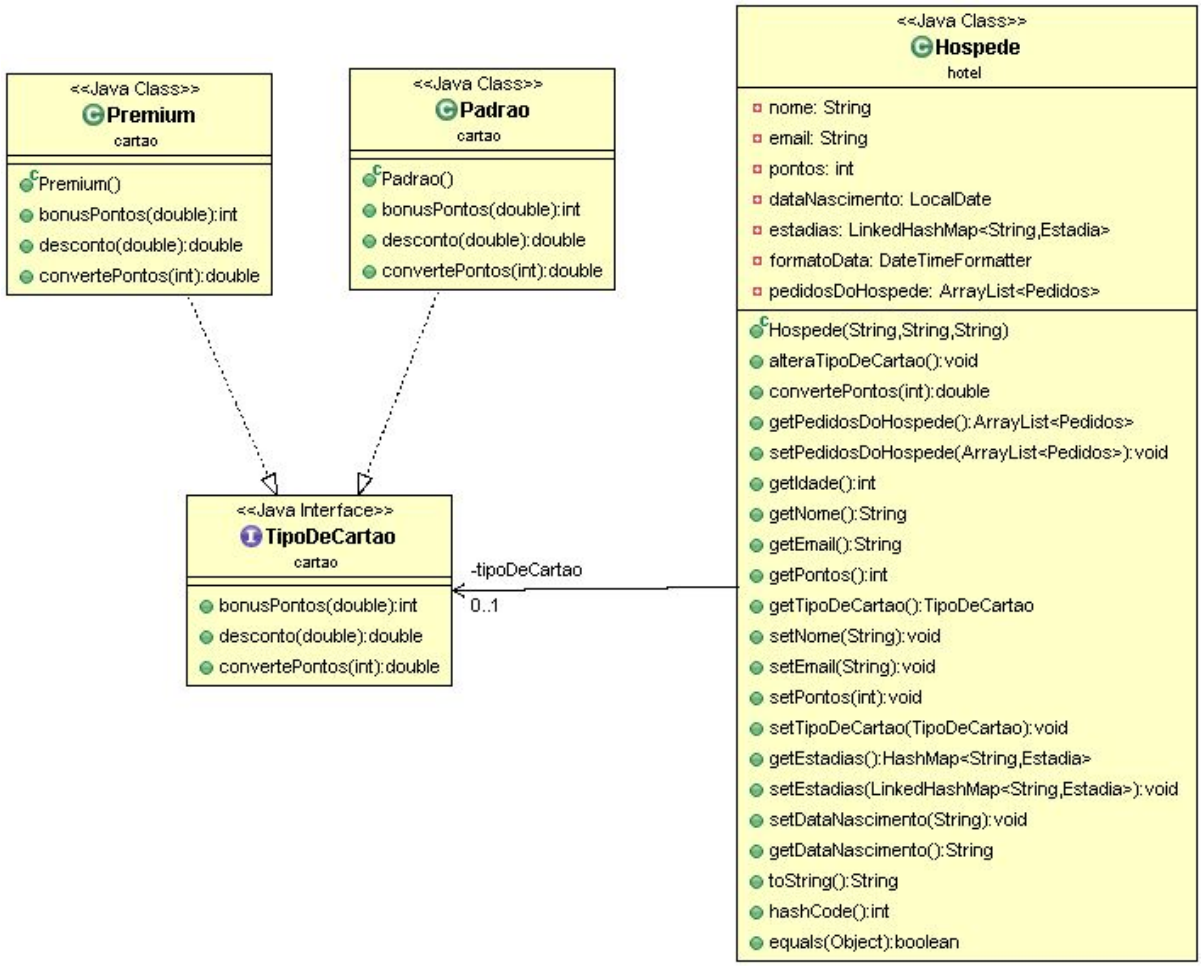
Cada hóspede agora tem um cartão associado a ele (por composição), esse cartão inicia sempre como padrão, e uma variável para armazenar os pontos adquiridos ao fazer qualquer tipo de transação, tal pontuação é utilizada para modificar o tipo de cartão quando suficiente.

Foi utilizado o padrão Strategy para obter o polimorfismo dos métodos do cartão (desconto e recompensa de pontos) e devido a necessidade de alterar o tipo do cartão em tempo de execução quando alcançado a pontuação necessária.

É válido ressaltar que os pontos são obtidos quando os hospedes fazem transações sendo essas pedidos no restaurante ou checkout.

Caso 7

Nesse caso é acrescentado a possibilidade do hóspede converter seus pontos do cartão em dinheiro, esse procedimento implica na perda dos pontos convertidos o que significa que o cartão pode regredir a um tipo de descontos mais baixo. Como a verificação da pontuação é feita antes de qualquer transação, tanto upgrade quanto o downgrade do cartão permanece garantido.



Caso 8

Nesse caso tínhamos como objetivo exportar os dados do hotel para arquivos, esses dados são as informações básicas dos hóspedes, dos dados do restaurante (menu), os detalhes das transações realizadas no hotel e por fim um resumo final do hotel; para isso foram criados cinco métodos, o iniciaArquivosSistema que basicamente cria os arquivos no diretório "arquivos_sistema/relatorios", que é criado automaticamente, e invoca os quatro métodos que percorrem os dados referentes a cada arquivo e escreve no mesmo usando em sua implementação o método Writer do FileWriter.

Caso 9

Chegando ao último caso o objetivo é armazenar e carregar todas as informações no sistema para assim garantir a persistência dos dados mesmo quando o sistema for finalizado. Temos o método fechaSistema que fecha e salva os dados do sistema, esse método salva as informações de forma serial no arquivo hug.dat. Usando ObjectOutputStream e writeObject conseguimos armazenar cada dado que implementa a interface Serializable. Visando uma maior independência entre os objetos escolhemos esse tipo de modularização.

Desse modo, o sistema é concluído e todas as informações/dados podem ser recuperados sempre que o sistema for iniciado e salvos quando o sistema for finalizado.