Hotel Urbano de Gotemburgo - HUG

Antonio Dartagnan, Laerson Saraiva
Lílian Honorio, Michele Santos

13 de outubro 2016

Introdução

O Hotel Urbano de Gutemburgo(HUG), é um sistema de um hotel, que será usado por administradores do hotel. A especificação do sistema foi construído pelos professores e monitores da disciplina de Laboratório de Programação junto com os alunos da disciplina de Didática. O objetivo do projeto é fornecer meios que possam ajudar o administrador do hotel com o controle básico de hóspedes, estadias, gerência das refeições do restaurante e do cartão de fidelidade. Tem como objetivo também fazer com que os estudantes apliquem os assuntos vistos na disciplina de Programação e Laboratório de Programação. Para começar a execução do projeto, os estudantes das turmas formaram grupos de 3 ou 4 integrantes, onde cada grupo desenvolveu a mesma especificação de projeto. O nosso projeto é um sistema de hotel, cuja organização deste foi dividida em 9 casos de uso. O projeto foi dividido em três milestones que acoplam os nove casos de uso. Os milestones são entregas de um conjunto específico de funcionalidade do software. No primeiro milestone deveria ser entregue a implementação dos casos de uso um, dois, três e quatro. No segundo milestone deveria ser entregue a implementação dos casos de uso cinco, seis e sete, juntamente com o diagrama de classes do projeto, uma versão inicial do design completo. No terceiro milestone, deveria ser entregue o diagrama de classes do projeto, a versão final, um relatório explicando o design e onde foram utilizados os principais tópicos vistos na disciplina e os casos de uso oito e nove.

1 Preparação e desenvolvimento do projeto

Inicialmente o grupo se reuniu para desenvovimento da UML(do inglês, UML - Unified Modeling Language, Linguagem de Modelagem Unificada). Foram feitos esboços de como o projeto seria desenvolvido e quais os padrões de projeto seriam utilizados. As atividades foram divididas em um caso de uso por integrante e conforme a disponibilidade dos mesmos, foi realizado o desenvolvimento de hieraquia de exeptions, javaDoc e testes unitários.

A mesma dinâmica se repetiu a cada milestone.

2 Padrões de Projeto

2.1 Model View Controller e Facade Design Patterns

Inicialmente, para o desenvolvimento do projeto, foram utilizados os seguintes padrões de projeto:

Model-View-Controller , em português modelo-visão-controlador, é um padrão de arquitetura de software (design pattern) que separa a representação da informação da interação do usuário com ele.

Utilizando este padrão de projeto foi possivel mascarar a lógica denegócio do software através de um controlador, definido como uma classe "Controller.java".

Façade Design Patterns: O Padrão de projeto Façade é um padrão de design de software usado comumente com programação orientada a objetos. Este nome é uma analogia para uma faixada arquitetural. Um Façade é um objeto que provê uma interface simplificada para um corpo de código maior, como por exemplo, uma biblioteca de classes.

Com o Padrão Facade podemos simplificar a utilização de um subsistema complexo apenas implementando uma classe que fornece uma interface única e mais razoável, porém se desejássemos acessar as funcionalidades de baixo nível do sistema isso seria perfeitamente possível. É importante ressaltar que o padrão Facade não "encapsula" as interfaces do sistema, o padrão Facade apenas fornece uma interface simplificada para acessar as suas funcionalidades. O Padrão Facade nos permite desconectar a implementação do cliente de qualquer subsistema. Assim, se quiséssemos acrescentar novas funcionalidades no subsistema seria necessário apenas alterar a Facade ao invés de alterar diversos pontos do sistema. Além disso, o padrão Facade simplifica uma interface tornando-a muito mais simples e unifica um conjunto de classes mais complexas que pertencem a um subsistema.

3 Lógica

3.1 Polimorfismo

Polimorfismo é uma interface comum entre tipos distintos, com o foco na capacidade de referenciar um objeto de diversas maneiras.

O polimorfismo é usado no nosso projeto para referenciar tipos distintos usando um mesmo tipo de referência. Exemplos desse uso são os diferentes tipos de cartão fidelidade, através do uso de interfaces, foi possivel atribuir comportamentos diferentes objetos que seguiam um mesmo "contrato". Assim foi possível, por exemplo, utilizar a mesma chamada de método para calcular descontos oferecidos de acordo com o topo de cartão fidelidade .

3.2 Composição

Composição é a combinação de Objetos para compor Objetos/comportamentos mais complexos.

Utilizou-se Composição em diferentes partes do projeto, pois esta possibilita encapsular e modularizar a lógica por trás de alguns comportamentos e a reutilização destes. Os principais exemplos de uso no projeto são: Entre a classe Facade e o Controller, e este, por sua vez, sendo composto pelos objetos BancoDeHospedes, BancoDeHospedagens, BancoDeRegistros, Restaurante, Validador e GerenteRelatorios

Há ainda a composição entre BancoDeHospedagens, Hospedes e Quartos. Essa composição foi organizada de forma que um Banco de Hospedagens encapsulasse as informações d de uma estadia que por sua vez possui um hospede associado e um quarto.

3.3 Factory

O padrão Factory é utilizado para encapsular a criação de Objetos. Utilizou-se este padrão para criação de quartos, pois para tal procedimento é necessário ser realizada validação do tipoDeQuarto. Utilizando a Factory não é necessário realizar validação sempre que for preciso criar um objeto do tipo Quarto, pois a validação está encapsulada na factory.

3.4 Strategy

O Strategy é usando para uma mudança dinâmica de tipo, para que se possa ter vários comportamento diferentes, que são definidos em tempo de execução. No projeto, o padrão de Strategy é usado no cartão de fidelidade do hospede, para que, a medida que sejam resgistradas despesas/gsatos cartão cumula pontos, de acordo com a pontuação atingida há uma troca dinâmica do tipo do cartão. E com o tipo de cartão alterado comportamentos diferentes são atribuidos ao objeto.

3.5 Hierarquia de Exceptions

Exception é um objeto que encapsula uma situação de erro que ocorre em tempo de execução ou compilação. Além disso, Exceptions podem ser organizadas segundo uma hierarquia através do uso de Herança.

Utilizamos hierarquia de Exceptions no projeto porque melhora a lógica de tratamento de erros e através da captura/lançamento de Exceptions e da concatenação das mensagens de erro é possível especificar onde o erro ocorreu e por qual motivo, sendo, portanto, de grande utilidade tanto para o programador, quanto para o usuário do sistema.

Exemplos de utilização deste conceito no projeto podem ser vistos principalmente nas classes Controller e Validador.

3.6 Arquivos

No nosso projeto foram usados fluxos e arquivos tanto para gerar históricos de cadastro do Restaurante, de hóspedes, de transações e para uma visão geral do sistema de hotel. Estes históricos foram gerados na forma arquivos de texto.

Para garantir a persistência dos registros dos hospedes, dados do restaurante e dados de trasações foi salvo em um arquivo, hig.dat, um objeto do tipo “controller”. Este objeto garantiu a persistência de dados, pois na sua classe estão encapsulados os dados de sistema atraves de outros “mini controllers”

3.7 Sobrescrita de método

Sobrescrita de método consiste na mudança do comportamento de um método herdado, preservando sua assinatura e seu tipo de retorno.

Este conceito é fundamental quando se pensa em chamadas polimórficas. Exemplos de aplicação podem ser vistos em todas as classes do projeto que implementam toString, equals e/ou hashCode, pois todas herdam da classe Object.

3.8 Sobrecarga de método

Sobrecarga de método consiste na criação de um mesmo método com possibilidades de parâmetros diferentes.

No nosso projeto, o método consultaTransacoes, localizado na classe BancoDeRegistos.java, é um exemplo disso. Há duas assinaturas diferentes para esse método. Uma para que sejam realizadas consultas apenas de acordo com uma String que mapeia o atribudo a ser consultado e outra para que seja possível consultar especificamente um item, através do seu indice e novamente mapeando o atribudo.

3.9 Testes de Unidade

Os testes de unidade são uma ferramenta mutio importante para garantir que cada componente do sistema atende à todos os quesitos que deveria. Eles foram feitos no nosso projeto através da biblioteca JUnit.

3.10 Collections

Foi feito o uso de collections para armazenar objetos que tinham tipo semelhante. Listas foram escolhidas por sua capacidade de indexação, iteração e possibilidade de fácil ordenação, como foi o caso de ordenar a lista de refeições do restaurante pelo seu preço ou pelo nome. Utilizamos Mapas para guardar estadias, de forma que, cada estadia é a chave, que está associada a um valor, que se trata de um objeto do tipo Hospede.

3.11 Enumerations

São conjuntos de constantes organizados em ordem de declaração que tem como principal função agrupar valores com o mesmo sentido dentro de uma única estrutura.

Foram usadas ao longo o projeto, para representar os tipos de quarto.

4 Milestones

4.1 Milestone 1

O milestone 1 foi entregue no dia 20 de setembro, contendo os casos de uso do um ao quatro. Primeiro caso de uso, que foi o crud de hóspede, tem como responsabilidade cadastrar e remover hospedes do hotel, registrando suas estadias e gastos. Fizemos o desbloqueio do sistema, a controller e a facade do mesmo. Criamos as classes Facade, Controller, Hospede e Banco de Hospede. Neste caso de uso, cadastramos os hóspedes, onde cada hospede possuia um nome, um e-mail e um ano de nascimento. Criando com essas informações um banco de hospede que guarda seus dados para em uma próxima visita, já ter o seu cadastro. Que permite a atualização, busca e remoção de hospedes do sistema.

Segundo caso de uso, que foi o checkin de hóspedes, tem como responsabilidade permitir o checkin dos hóspedes cadastrados no sistema . Criamos as classes Estadia,

Quarto, Tipo de Quarto e Banco de hospedagem. Após fazer o checkin, o hóspede será associado a uma estadia, porém ele pode estar associado a mais de uma. A estadia é composta por um quarto e uma quantidade de dias em que o hospede vai ficar no hotel. O quarto por sua vez possui um numero e um valor de diária, que muda de acordo com o tipo de quarto. No fim a estadia informa o total que o hospede vai pagar de acordo com a quantidade de dias e o tipo de quarto.

Terceiro caso de uso, que foi o checkout de hóspedes e histórico de lucros, tem como responsabilidade realizar o checkout dos hóspedes para registrar os lucros de estadias obtidos pelo Hotel. Ao realizar o checkout o sistema deve inserir a data do dia, o nome do hóspede, o numero do quarto que estava hospedado, e o total pago por ele. Um detalhe importante é que caso o hospede esteja associado a mais de uma estádia, ele deve fazer um checkout para cada estadia.

Quarto caso de uso, foi implementada a classe Restaurante.java, que, 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.

4.2 Milestone 2

Quinto caso de uso: Foi implementado na interface Refeicao.Java a classe Comparable, para que os pratos e refeicoesCompletas possam ser comparavéis por nome e assim ordenados. Foi desenvolvida também uma classe especifica para outro tipo de ordenação: RefeicaoComparatorByPrice.Java. Nela também é implementada a inteface Comparable e e criado métódo para ordenação de acordo com preço da Refeicao.

Sexto caso de uso: Foi implementada a classe CartaoFidelidade.java. Agora cada objeto do tipo hóspede possui um atribudo cartaoFidelidade. O cartaoFidelidade possui um tipo que muda dinamicamente de acordo com a quantidade de pontuação acumulada.

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. Os pontos são obtidos quando os hospedes fazem transações sendo essas pedidos no restaurante ou checkout.

Sétimo caso de uso: foi acrescentado no cartão fidelidade o método convertePontos. Trata-se de receber um valor em pontos para ser repassado como crédito ao hospede e subtrair este valor da pontuação total.

4.3 Milestone 3

Oitavo caso de uso: 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; Foi criada a classe GerenteRelatorios com métodos que, de acordo com o tipo de relatório solicitado, realiza a criação de arquivos txt com as informações registradas.

Nono caso de uso: Para garantir a persistência dos registros dos hospedes, dados do restaurante e dados de trasações, foi criada a classe Armazenamento.java. Que salva dados do sistema e carrega caso eles existam. Para isso foi salvo em um arquivo, hug.dat, um objeto do tipo “controller”. Este objeto garantiu a persistência de dados, pois na sua classe estão encapsulados os dados de sistema atraves de outros “mini controllers”. Dentro

da Facade o metodo iniciaSistema foi implementado de for que, caso o arquivo exista e contenha dados, eles sejam lidos. Caso contrário é criado um novo objeto.

Já o método fechaSistema é resposável por salvar os dados, no caso escrever em hug.dat um objeto do tipo controller, ou sobreescrevê-los caso seja necessário;