UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

CURSO DE CIÊNCIA DA COMPUTAÇÃO

Ariann Farias, Luan Rocha, Nilton Ginani, Yovany Cunha

Relatório referente ao projeto Hotel Urbano de Gotemburgo (HUG) da disciplina: LABORATÓRIO DE PROGRAMAÇÃO II.

Campina Grande, 2016.

Ariann Farias, Luan Rocha, Nilton Ginani, Yovany Cunha

Relatório referente ao projeto Hotel Urbano de Gotemburgo (HUG) da disciplina: LABORATÓRIO DE PROGRAMAÇÃO II.

Relatório apresentado como requisito para evidenciar os recursos da disciplina de Programação II, prática e teórica utilizados no projeto “Hotel Urbano de Gotemburgo (HUG)”.

Prof. Drº Matheus Gaudencio do Rêgo.

Campina Grande, 2016.

Resumo

Este relatório apresenta os recursos das disciplinas de Programação II e Laboratório de programação II utilizados no projeto Hotel de Gutemburgo (HUG), durante o percurso da disciplina muitos elementos foram vistos e trabalhos, tais como: Encapsulamento, Refatoramento, Ocultação da informação através dos métodos de acesso, Coleções, Tratamento de Erros através de exceptions checked e unchecked, Herança entre classes, Composição por meio das delegações de atividades, Interface, Polimorfismo, Design Patterns Strategy e por último a persistência de dados através do uso de arquivos. O projeto foi orientado a partir do Padrão GRASP, com suas regras de design, que orientaram a alocação de informação nas devidas classes.

DESENVOLVIMENTO DO PROJETO: ESTRUTURA E ELEMENTOS

1. ENCAPSULAMENTO, OCULTAÇÃO DA INFORMAÇÃO E REFATORAMENTO

O projeto Hotel de Gutemburgo (HUG) foi desenvolvido seguindo os recursos da disciplina, tais quais Encapsulamento em que é pensando a distribuição da informação em classes e suas responsabilidades, servindo para controle de métodos e acessos de atributos das diferentes classes do projeto. Por meio dos métodos de acesso, public e private, é gerada a ocultação da informação e segurança da mesma. A exemplo, na classe Hóspede, utilizada para instanciar um objeto do tipo Hóspede, temos ocultação da informação:

No exemplo abaixo, os atributos estão com visibilidade private para garantir ocultação da informação, evitando assim que outras classes modifiquem diretamente os atributos do projeto.

```
33      private String nome;
34      private String email;
35      private String dataNascimento;
```

Só é permitido o acesso e modificação por meio dos métodos public, criados para acesso das informações do objeto, como os Gets e Sets.

```
178     public String getNome() {
179         return nome;
180     }
181
182     public void setNome(String nome) throws StringInvalidaException {
183         this.nome = nome;
184     }
```

Sendo embasado no padrão GRASP, nos quesitos coesão e expert, pois cada classe possui distinta funcionalidades e ambos os quesitos possuem o intuito de adequar a informação e métodos.

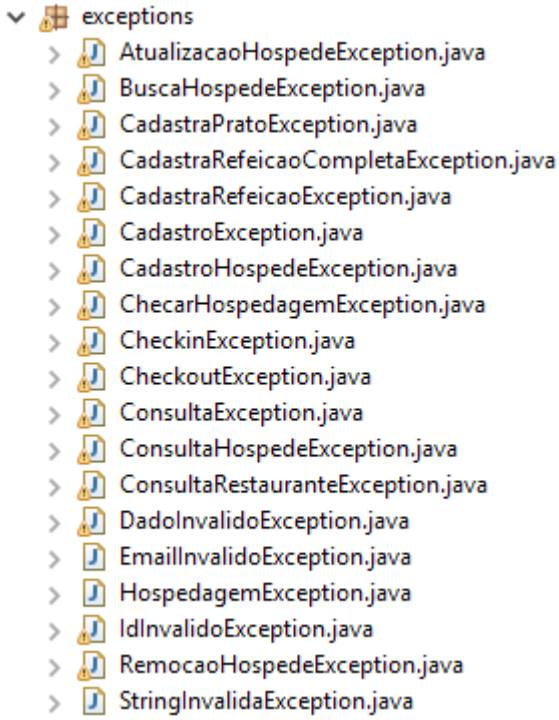
O refatoramento foi utilizado em todas as classes do projeto, na criação e manutenção do código durante o período de desenvolvimento, para melhor leitura e reúso do mesmo.

2. COLEÇÕES E ESTRUTURAS DE DADOS:

Ao projeto foram acrescentadas estruturas para armazenamento de informações nas diferentes classes, por exemplo, o HotelController, que possui um mapa (private Map<String, Hospede> hospedes;) de hospedes, que usamos com frequência para armazenamento de Hospedes, com todas os seus Membros. No Hospede possuimos mais três estruturas de armazenamento, que são 3 ArrayList, duas com objetos do tipo <Estadia> e outra com objetos do tipo <Transacao>, assim, através do Framework de Collections facilitamos, o armazenamento de dados do projeto, nas devidas classes.

3. EXCEPTIONS E TRATAMENTO DE ERROS:

Para a necessidade do usuário tentar utilizar o programa com entradas que sejam consideradas casos Excepcionais, ou até mesmo para o caso de durante o processamento de informações ocorressem comportamentos que causam erros de lógica no programa, foram implementadas Exceptions, checáveis, através de uma hierarquia de Exceptions.



Algumas das Exceptions feitas possuem mensagem padrão, dependendo do erro que ocasione o seu lançamento ou captura, como a classe AtualizacaoHospedeException, que define um caso excepcional emitido, caso seja passado algum dado inválido durante a atualização de dados de um Hospede, juntamente com o lançamento da mensagem que informa o dado que por acaso foi passado de forma errônea, por exemplo, uma mensagem que venha da StringInvalidaException.

4. HERANÇA:

A Herança de classes foi utilizada no projeto entre as classes Refeicao (superclasse abstrata), RefeicaoCompleta e Prato (subclasses que herdam de Refeicao), ou seja, a classe Refeicao se subdivide em duas outras classes. A classe Refeicao é abstrata para que não ocorra instanciação, pois como superclasse, sua funcionalidade é ceder métodos para as classes que a herdam os reutilizarem, e aplicarem as funcionalidades desejadas à sua forma. As Exceções criadas também se utilizam de herança, pois para que seja feita a hierarquia de exceções, é necessário que para que a classe seja uma Exception e ela trate de algum caso excepcional, ela tem que herdar do tipo Exception superior. Para visualização no código segue abaixo:

```
8
9  /**
10  * Classe que representa uma abstracao de uma refeicao.
11  * @author Ariann Farias, Luan Rocha, Nilton Ginani, Yovany Cunha - Turma 03
12  *
13  */
14 public abstract class Refeicao implements Comparable<Refeicao>{
15
16     private String nome;
17     private String descricao;
18
19     public Refeicao(String nome, String descricao) throws StringInvalidaException {
20         verificaNome(nome);
21         verificaDescricao(descricao);
22         this.nome = nome;
23         this.descricao = descricao;
24     }
25
26     /**
27      * Este metodo calcula o preco de todos os pratos inclusos na refeicao.
28      * @return Retorna um double com o preco.
29      */
30     public abstract double calculaPreco();
31
32     public abstract String toString();
```

A exemplo, a classe possui os métodos calculaPreco() e o toString(), que são sobrescritos nas subclasses de Refeicao (Prato e RefeicaoCompleta).

5. COMPOSIÇÃO, INTERFACE E POLIMORFISMO:

Composição é uma técnica presente em todo o projeto, usada para delegar as devidas responsabilidades para os métodos. Na Facade e no Controller há a presença bem evidente de Composição, já que a Facade delega ao Controller depois de receber os dados cedidos pelo usuário. O Controller por sua vez aciona as classes de sua lógica para elas executem as suas funcionalidades.

O uso de polimorfismo fica mais evidente quando analisamos o cartão e seus tipos. Existe a presença de *Strategy* para que a mudança de estado do cartão, que pode ser Padrão, Vip ou Premium, seja definida de acordo com a quantidade de pontos de fidelidade.

A título de visualização segue uma imagem de dois métodos onde existe chamada

polimórfica, a depender do tipo de cartão:

```
38    public double aplicaDescontoGastos(double valorGasto){
39        int adicional = 0;
40        int desconto = 0;
41
42        if (valorGasto > 100) {
43            adicional = (int) ((valorGasto/100)* tipoCartao.adicionalDesconto());
44        }
45
46        return (valorGasto * tipoCartao.desconto() - adicional);
47    }
48
49    public double pagaDividasGastos(double pontos){
50        double valor = this.tipoCartao.pagaDividasGastos(pontos);
51        return valor;
52    }
53
```

O método aplicaDescontoGastos reduz o valor a ser pago pelo Hóspede dependendo do tipo cartão a ele associado, assim como o método pagaDividasGastos remunera o Hóspede com Pontos de Fidelidade obedecendo às regras definidas em cada tipo de Cartão.

6. ARQUIVOS E PERSISTÊNCIA DE DADOS:

Por fim para garantir a persistência do sistema foi utilizado o conhecimento de Arquivos em Java. As classes relevantes para que o sistema tenha seu estado garantido tem uma Interface chamada Serializable implementada, permitindo assim o reuso e a recuperação desse estado futuramente.

A seguir podem ser vistos dois métodos que são responsáveis por salvar e ler o

```
43    public void salvaHotelController(HotelController hotelController) throws IOException {
44        FileOutputStream fos = new FileOutputStream("arquivos_sistemas/hug.dat");
45        ObjectOutputStream oos = new ObjectOutputStream(fos);
46        oos.writeObject(hotelController);
47        oos.close();
48    }
49
50    public HotelController leHotelController() throws IOException, ClassNotFoundException {
51        FileInputStream fis = new FileInputStream("arquivos_sistemas/hug.dat");
52        ObjectInputStream ois = new ObjectInputStream(fis);
53        Object o = ois.readObject();
54        ois.close();
55        return (HotelController) o;
56    }
```

estado da classe HotelController:

Ao término do desenvolvimento desse projeto pode ser certificado que todos os assuntos abordados nas disciplinas de Programação II e Laboratório de Programação II foram analisados, debatidos e implementados em algum ponto.