**Documentação do Projeto "UDI-DECOLA"**

**Título do Projeto: UDI-DECOLA**

**Descrição**: O Udi-Decola é uma companhia que permite realizar buscas e compras de passagens aéreas e diárias em hotéis.

**Requisitos:**

**a) O Udi-Decola é uma companhia que permite realizar buscas e compras de passagens aéreas e diárias em hotéis**

R: Em desenvolvimento.

**b) Para o Udi-Decola é muito importante manter um cadastro dos trechos de voos disponíveis. Para cada trecho é importante saber o seu código identificador, que é formado por um conjunto de 3 letras e 5 nros. Também é importante armazenar o local de origem, o local de destino, a hora de saída e a hora de chegada. Pense na melhor forma de representar o local de origem e saída, pois uma cidade pode ter mais de um aeroporto. Também é preciso saber a companhia aérea do trecho voo.**

**R:** Foi criado a classe "Voo" que representa um voo de um aeroporto, entro da classe "Voo" foi definido um código identificador("codigo\_identificador") que passar por uma verificação para saber se o código possui 3 letras e 5 números. Foi armazenado em cada "Voo" seu local de origem("origem"), o local de destino("destino"), a hora de saída("saída") e a hora de chegada("entrada"). A forma para representar melhor a cidade de origem e destino foi a criação de uma lista de Voos dentro da classe “Companhia\_Aerea”, ou seja, a origem sempre vai ser a String que cada aeroporto possui "cidade" e o destino é sempre a String "cidade" que outro aeroporto possui mais seu código IATA.

**c) As companhias aéreas devem ser previamente cadastradas. Cada companhia aérea tem um CNPJ, nome oficial, nome de divulgação e data de criação.**

**R:** Foi criado um construtor para a classe "Companhia\_Aerea" e nele foi inserido "public Companhia\_Aerea(String cnpj, String nome\_oficial, String nome\_de\_divulgacao, String data\_de\_criacao)"

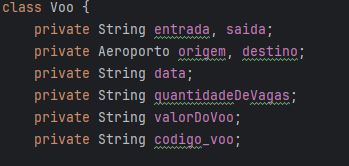
**d) Cada trecho de voo é realizado em vários dias do mês. Por exemplo. O trecho Uberlândia para Garulhos-SP no horário das 8h e chegada as 9h, aconteceu todos os dias desta semana, sendo cada dia um voo diferente. É importante registrar os voos de cada trecho de voo.**

**R:** Como a comunicação funciona de Aeroporto -> Aeroporto, quando se é pedido para realizar uma viagem mais distante ele realizará uma busca até encontrar o melhor caminho, levando em conta a origem, destino, horário entre cada voo e quantidade de vagas.

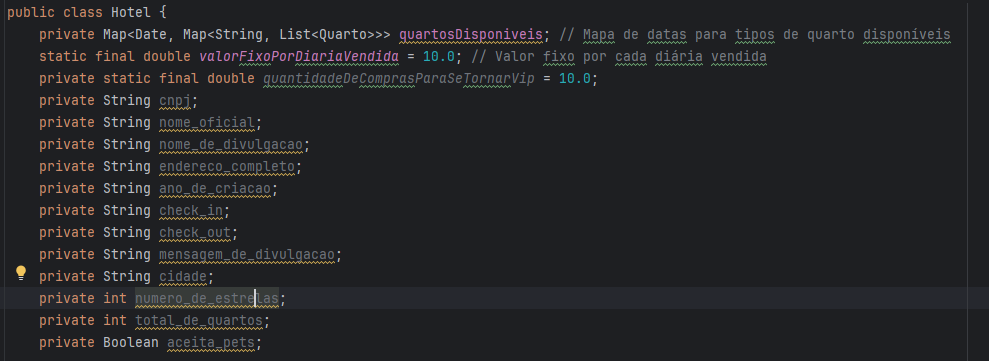
O método: “public Voo encontrarMelhorVoo(String origem, String destino, String data)” fornece o melhor voo tento como entra a origem, destino e a data.

**e) Para cada voo é preciso saber a sua data, a hora programada de partida e hora programada de chegada, que podem ser diferentes da hora do trecho do voo devido a alguma mudança da companhia. É importante também saber o nro de vagas disponíveis nesse voo para passageiros da Udi-Decola. O mesmo trecho de voo pode ter diferentes valores em diferentes voos, que dependem da época do ano e de eventos externos. Assim cada voo tem um preço da passagem.**

**R:** Foram definidas variáveis de entrada e saída de cada voo, sua data, o Aeroporto de origem e o de destino, o número de vagas para o voo e o preço.



**f) Além de voos, o Udi-decola também permite a busca por hotéis. Assim, é importante armazenar cada um dos hotéis parceiros. Para cada hotel é importante registrar o seu CNPJ, nome oficial, nome de divulgação, endereço completo, ano de criação, nro de estrelas, se aceita pets ou não, nro total de quartos, horário do check-in e horário do chek-out. Também é importante uma mensagem de divulgação que será exibida na busca. Ex: “Hotel 4 estrelas com buffet de café da manhã”. Uma breve descrição do hotel também deve ser exibida. Lembre-se que a cidade do hotel é importante para facilitar as buscas.**

**R:** para suprir a necessidade foi criado estas seguintes variáveis dentro da classe “hotel”:

**g) Para cada hotel é importante registrar para cada data do ano, o nro de quartos disponíveis para locação do tipo single, duplos e triplos, bem como o valor da diária de cada quarto sem desconto, o valor do desconto e valor após aplicar o desconto. Estes são considerados os quartos standard. Em cada data o hotel pode aceitar ou não cancelamento.**

**R:** Para suprir a necessidade foi implementado um sistema de hashmap no qual funciona da seguinte forma:

+------------------+  
| Hotel |  
+------------------+  
| - quartosDisponiveis: HashMap<Date, HashMap<String, List<Quarto>>>  
+------------------+  
  
+------------------------+  
| HashMap<Date, ...> |  
+------------------------+  
| | |  
| +---[15/01/2024]-----> HashMap<String, List<Quarto>>  
| | |  
| +---[16/01/2024]-----> HashMap<String, List<Quarto>>  
| | |  
| +---[17/01/2024]-----> HashMap<String, List<Quarto>>  
| | |  
| ... |  
+------------------------+  
  
+-------------------------------+  
| HashMap<String, List<Quarto>> |  
+-------------------------------+  
| | |  
| +---["Single"]-----------> Lista de quartos Single  
| | |  
| +---["Duplo"]------------> Lista de quartos Duplos  
| | |  
| +---["Triplo"]-----------> Lista de quartos Triplos  
| | |  
+-------------------------------+

Ou seja, foi criado um hashmap que possui um valor “date” e uma chave que é outro hashmap que contém ali um valor “tipo de quarto” que possue uma chave que é uma lista de quartos daquele tipo especificado.

**h) Para cada hotel é também importante registrar para cada data o nro de quartos de luxo, o valor da diária sem desconto, o valor do desconto e o valor após aplicar o desconto.**

**R:** Foi implementado uma variável “private boolean deLuxo;” que informa se o quarto é de luxo ou não e um método “public int contarQuartosLuxoDisponiveis(Date data)” que verifica se dentro dos quartos daquele hotel em específico naquela data o número de quartos de luxo. O valor da diária sem desconto, valor do desconto e o valor após aplicar o desconto pode ser concebidos pelas variáveis "private double diariaSemDesconto, private double desconto, private double valorAposDesconto." Dentro de cada quarto, pois um quarto pode ter preço variável.

**i) Os funcionários da empresa precisam ser cadastrados. Para isso é importante registrar o nome, CPF, endereço completo, nro da carteira de trabalho e data de nascimento. Cada funcionário tem um salário fixo. Lembre-se que um funcionário pode ser também um cliente da empresa.**

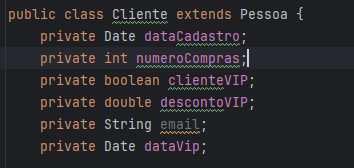
**R:** Foi criado uma classe “Pessoa” que possui nome, cpf, endereço completo e data de nascimento como variáveis e foi usando herança na classe funcionário e incrementado salario Fixo, endereco Completo, Carteira Trabalho.

**j) Há um tipo de funcionário, que são os responsáveis por nova parcerias de hotel. Cada vez que um hotel é registrado é importante saber o funcionário responsável. Também é importante saber quantos hotéis um funcionário foi responsável pelo cadastro. O salário deste tipo de funcionário é o seu salário fixo acrescida de uma porcentagem fixa para cada novo hotel que ele adiciona.**

**R:** Foi criada uma classe “FuncionarioParcerias” que cria um funcionário parceiro sendo responsável por parceriais de hoteis que usa dos recursos de funcionário e um contador de hoteis que ele tem parceria a porcentagem de bonus por hotel, seu salário é acrescido do bônus.

**k) Para efetivar uma compra no Udi-Decola, os clientes devem ser previamente cadastrados. Para isso é preciso saber, CPF, nome, endereço completo, data de nascimento, data de cadastro e email. Quando um cliente efetua mais que um dado número de compras, ele se torna um cliente VIP e neste caso é importante armazenar o valor do desconto que o cliente terá em novas compras, bem como a data em que ele se tornou vip. O cálculo deste valor segue regras específicas da empresa, que não serão detalhadas aqui.**

**R:** A classe Cliente estende a classe Pessoa e adiciona os atributos dataCadastro, numeroCompras, clienteVIP, descontoVIP e dataVip. Esses atributos são específicos para os clientes do Udi-Decola e permitem rastrear informações importantes, como o status VIP do cliente e seu desconto associado, além de suas compras e data de cadastro.



**l) As pesquisas feitas por cliente devem ser registradas. Para cada pesquisa é importante registrar a origem, o destino, data de início e data de fim. Isso ajudará a verificar os destinos e datas mais procurados. Também é importante armazenar a data e hora que a pesquisa foi realizada.**

**R:** Dentro de cliente foi criado uma lista de pesquisas e posteriormente uma classe chamada pesquisa, sempre que o cliente faz uma pesquisa ele adiciona essa pesquisa à lista de pesquisas do cliente.

**m) Cada compra deve ser registrada. Uma compra é composta por informações sobre a passagem aérea, diárias em hotel ou ambas. Obviamente é importante armazenar a quantidade de cada um desses itens, já que uma compra pode envolver passagens para mais de uma pessoa em mais de um quarto de hotel.**

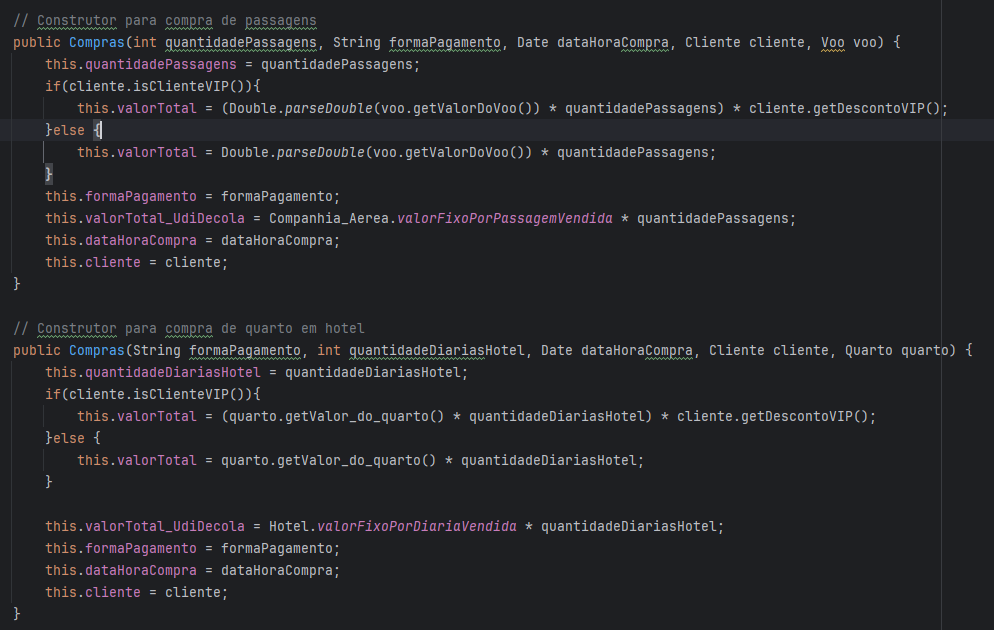
**R:** Assim como foi feito para “pesquisas” foi feito para “compras” onde cada cliente possui uma lista de compras.

**n) Para cada compra é importante registrar também a data e hora em que foi realizada, bem como o cliente da compra. A compra também armazena o seu valor total, que é a soma dos itens comprados, bem como a forma de pagamento realizada.**

**R:** Assim foi adicionado valorTotal, formaPagamento, dataHoraCompra dentro da classe “Compras” que é usado como uma lista dentro de cada cliente.

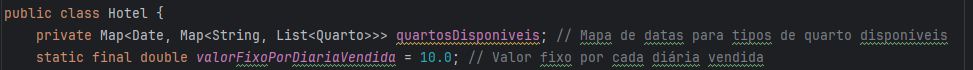
**o) Para cada compra é importante atualizar o número de compras do cliente, já que ele pode se tornar um cliente VIP após realizar uma compra e atingir o nro limite de comprar.**

**R:** O número de compras é a quantidade de compras presentes na lista “Compras”, como cada hotel tem uma quantidade para se tornar vip, sempre que o cliente realizar uma nova compra o programa vai comparar suas quantidades de compras e dizer se ele é vip com base na quantidade mínima para ser VIP de cada hotel. Ele sendo vip a classe “Compra” quando acionada vai verificar e retornar o valor total com desconto.



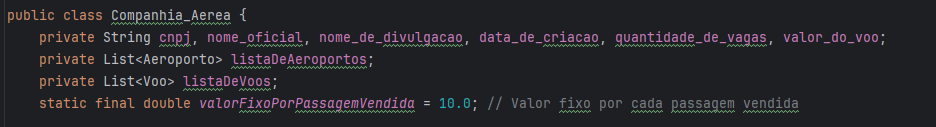
**p) Cada hotel paga um valor fixo para o Udi-Decola por cada diária de hotel vendida.**

**R:** Foi adicionado uma variável “valorFixoPorDiariaVendida” na criação de cada hotel que diz o valor fixo a ser pago para Udi-Decola por cada diária vendida.



**q) Cada companhia aérea paga um valor fixo para o Udi-Decola por cada passagem aérea vendida.**

**R:** Foi adicionado uma variável “valorFixoPorDiariaVendida” na criação de cada companhia aérea que diz o fixo a ser pago para Udi-Decola por cada passagem vendida.



**2- Implemente em Java, usando técnicas de encapsulamento todo o seu Sistema para a UdiDecola.**

**A seguir um exemplo:**

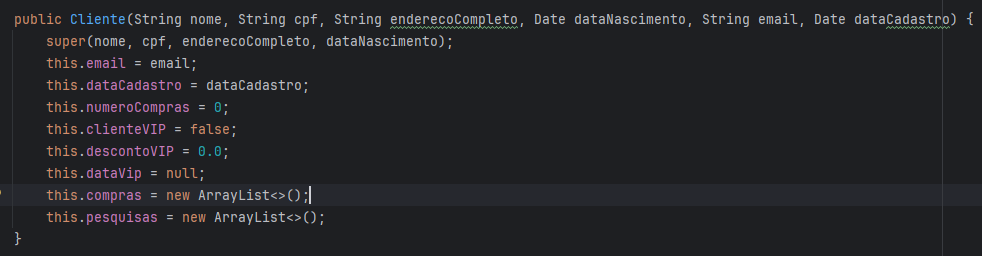
**public void setNome(String nome){ if (nome.length() > 0) { this.nome = nome; } }**

**a. Para validação de cpf, implemente o algoritmo real de validação de cpf (que é encontrado facilmente na Internet, implementado nas mais diversas linguagens).**

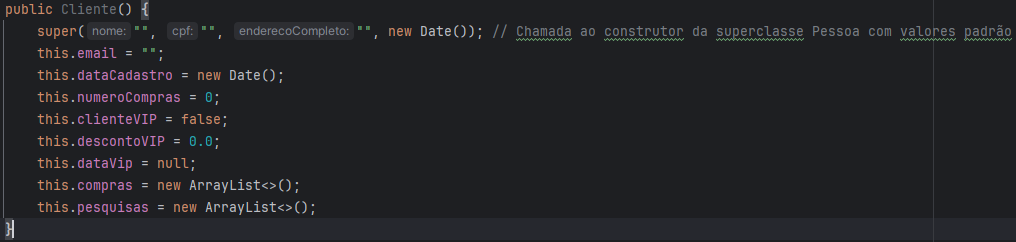
**R:** Foi implementado encapsulamento para todo o sistema da UdiDecola. Foi criado uma classe “Valida CPF” e foi aplicado para todas as pessoas.

**3- Considerando o uso de construtores:**

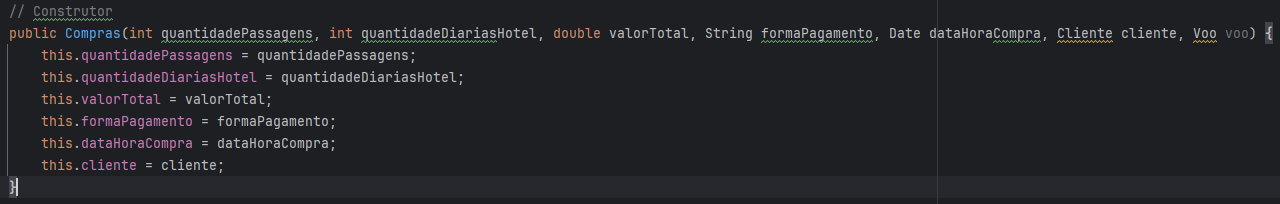
**a. Crie um construtor para a classe cliente que receba como parâmetro o seu nome e o seu CPF.**

**R: :**

**b. Crie um construtor default (sem parâmetros) explicitamente para a classe cliente.**

**R:** 

**c. Crie um construtor para a classe Compra de forma que este receba como parâmetro o cliente que fez a compra, o vôo da compra, bem como a data e hora. Desta forma, obriga-se a associação da compra com o cliente e vôo, que já foram previamente criados no sistema.**

**R:** Foi adicionado ao construtor a um tipo cliente e um tipo voo. 

**d. Crie construtores para cada uma das classes do problema de forma a atribuir valor aos atributos da classe com os parâmetros recebidos. Toda classe deve ter pelo menos um construtor criado por você.**

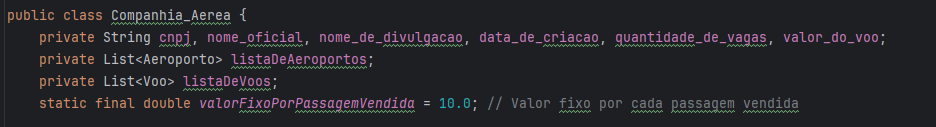
**R:** Foi feito um construtor para cada classe.

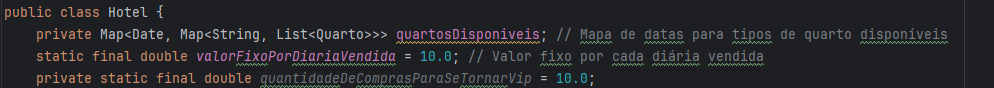
**4- Considerando os casos de herança, implemente todas as situações de herança descritas nos requisitos.**

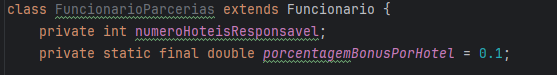
**R:** Foi usado herança para os casos de funcionario, funcionarioParceiro, cliente. Onde todos eles derivam de pessoa.

**5- Implemente todos os atributos estáticos para o funcionamento do sistema, tais como o valor fixo que os hotéis e companhias aéreas pagam para o Udi-Decola, o nro fixo de compras que um cliente deve fazer para se tornar um cliente VIP e o valor fixo pago para funcionário que registram novos hotéis.**

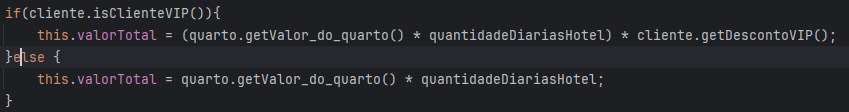
**R:** Foi implementado “private static final (tipo) (nome)” para os valores fixos que hostéis e companhias aéreas pagam para a Udi-Decola, para o número fixo de compras que um cliente deve fazer e para os funcionários que registram novos hotéis.

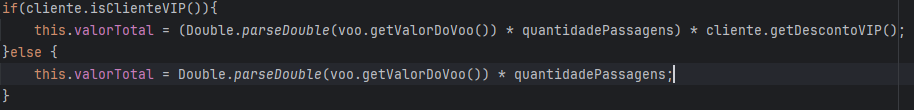






**6- Implemente a realização de uma compra. Lembre-se que a realização de uma compra tem impacto na conversão de um cliente para VIP. Lembre-se de calcular para cada compra, o valor total a ser pago pelo cliente e o valor total a ser pago para o Udi-Decola.**

**R:** Foi implemento a realização de uma compra de passagem aerea e uma compra de quarto de hotel na main. Para realizar o desconto do cliente vip foi feito da seguinte forma dentro do construtor de compra de quarto e passagem:  
  




O valora ser pago pelo cliente foi dito no programa principal junto com o valor a ser pago para a Udi-Decola. (MAIN).

**7- Classes Abstratas**

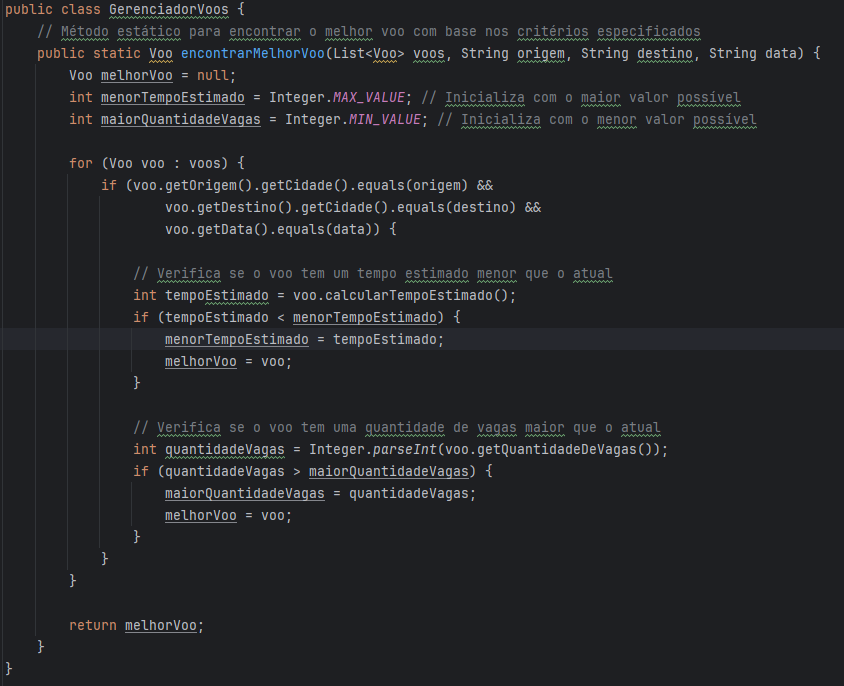
**a. Veja se alguma classe do problema deveria ser abstrata.**

**R:** A classe Pessoa é uma classe abstrata.

**8- Requisito extra**

**a. Invente um requisito no problema que justifique a criação de uma nova classe ou de um novo método (podem ou não ser abstratos) ou de uma nova subclasse. Não esqueça de descrever o requisito**

**R:** Um requisito extra para o projeto foi a criação da classe “GerenciadorVoos” que me retorna o melhor voo atendendo as especificações de origem, destino e data.



**Integrantes:**

* **Felipe Santos Silva (12221BCC042)**