Sistema de Faturamento e Cobrança em Telecomunicações com Tarifários Dinâmicos

Realizei este exercício com o intuito de prosseguir no processo de recrutamento da Dellent e da Altice Labs, em que o exercício consistia numa operadora de telecomunicações que deseja melhorar o seu sistema para gerir pedidos de cobrança (Charging Request/Reply) e contabilização de uso (Billing Account e CDRs - Call Detail Record), com base em tarifários complexos.

Dado isto e mais algumas informações presentes no enunciado do exercício criei então 5 Classes

- Charging (Superclasse de ChargingRequest e ChargingReply)
- ChargingRequest
- ChargingReply
- BillingAccount
- CDR

Passando então a descrever cada uma das classes:

- Charging

  Criei esta classe com o intuito de ser uma superclasse para os Charging Requests e Charging Replys, pois ambos têm os seus ID's e assim consegui tirar proveito de uma das características da programação orientada aos objetos (POO), a herança.

- ChargingRequest

  Nesta classe comecei por definir as várias variáveis da mesma, como o TimeStamp do pedido, o Serviço pretendido, se está em Roaming, o MSISDN (número de telemóvel), o número de unidades pretendidas (RSU) e defini também uma flag, para sinalizar quando não é possível fazer uma transação (valores negativos de saldo).

  Depois de definir as funções padrão de uma classe, tais como Getters, Setters, Clone, Equals e ToString comecei por definir duas funções para verificar o dia em que o pedido é feito, uma para verificar se é durante o fim de semana e outra para verificar se é de noite. Em seguida defini duas funções para indicar o tipo de tarifário atribuído ao cliente, podendo variar entre Alpha 1, Alpha 2 ou Alpha 3 e Beta 1, Beta 2 ou Beta 3, cada um com os seus critérios específicos, tendo escrito as funções de forma separada, uma para os tarifários do serviço A e outra para os tarifários de serviço B, retornando 0 se nenhum dos tarifários for elegível para o cliente.

  Por fim, defini as funções que submetem o Request do cliente "Request" e "Request_Timed", em que uma deles submete no momento em que o cliente faz o pedido e outra que submete o pedido numa data que o cliente especifica, ambas funcionam da mesma maneira, é criado um ChargingRequest com o serviço pedido, estado de roaming, MSISDN do cliente e numero de SU pretendidas (caso o cliente use uma data específica utiliza a função "Request_Timed" e adiciona a data), em seguida é verificado qual o tarifário indicado, de acordo com o serviço pedido, sendo que se nenhum for atribuído é retornado um ChargingReply a avisar o cliente

do mesmo. Caso seja atribuído um tarifário é criado um ChargingReply onde é utilizado a função que dá a resposta ao pedido, bem como o seu pagamento "Reply", estando esta definida na classe ChargingReply.

- ChargingReply

Nesta classe apenas usei 2 variáveis, o Resultado do Reply, Result, e o GSU, SU concedidas do pedido. Mais uma vez defini os construtores, Getters, Setters, Clone, Equals e ToString e, em seguida, comecei por definir as funções de pagamento, em que primeiramente criei uma variável temporária *temp* para fazer todos os cálculos necessários, tendo em conta a especificidade de cada um dos tarifários e, no fim, alterar para o valor final do bucket correspondente, defini também a função "wichBucket" que determina qual bucket será usado no pagamento do tarifário Alpha1 e Beta1, tendo em conta as suas exigências na escolha do bucket para o pagamento.

Logo depois criei a função "wichPayment" que verifica o tarifário do Cliente e efetua o mesmo, sendo que, sempre que é efetuado um pagamento de serviço A, o CounterA é incrementado.

Por último, defini a função "Reply" onde é efetuado o pagamento, consoante o número de SU's pedidas, fazendo um pagamento de cada vez (dentro de um Ciclo "For") e verificando se o pagamento é possível, caso este seja possível incrementa o número de GSU's. Se o número de GSU's coincidir com o número de RSU's, então a resposta será "OK", senão, irá aparecer uma mensagem que diz "CreditLimitReached", informo o cliente de quantos SU's foram concedidos, incremento o "CounterB" caso o tarifário seja Beta1 e incremento o "CounterC" caso o cliente esteja em Roaming, em seguida altero o "CounterD" para a data do último pedido e adiciono o pedido à lista de CDR's do Cliente, criando o CDR através da função "List_CDR" definida na classe BillingAccount.

- BillingAccount

Defini as várias variáveis correspondentes à classe BillingAccount, tais como MSISDN, Buckets, Counters, Serviços e também a lista de CDR's do cliente e defini os construtores, Getters, Setters, Clone, Equals e ToString e, em seguida, defini as funções onde o cliente submete o seu pedido, podendo fazê-lo, como já referi acima, com uma data específica à sua escolha, ou então, no imediato. Nesta função apenas criei um "ChargingRequest" com os pedidos do Cliente e executei a função "Request" ou "Request_Timed", dependendo da escolha do cliente, da classe "ChargingRequest".

Prossegui criando uma função que cria um CDR através dos dados do Charging Request, do Cliente (Billing Account) e do Charging Reply, tendo criado antes duas funções que colocam os Buckets e os Counters num HashMap, para depois serem exibidos para o cliente e defini também uma função que coloca um CDR na lista de CDR's na Billing Account do Cliente.

- CDR

Comecei, mais uma vez por definir as variáveis da classe, tais como TimeStamp do Request, MSISDN, Serviço, ChargingRequest, ChargingReply, Buckets, Counters e ID do tarifário. Nesta classe acabei por apenas definir as funções padrão, Getters, Setters, Clone, Equals e ToString.

Para terminar vou explicar as decisões de design que tomei ao longo da resolução do desafio, bem como o fluxo de funcionamento da aplicação.

Começando pelo fluxo da aplicação, o cliente efetua um pedido, é criado um ChargingRequest com as informações do pedido do cliente e, em seguida é feito o Request no ChargingRequest, onde é determinado o tarifário, caso não seja possível avisa o cliente que não existe nenhum tarifário elegível, caso seja possível, avança para a criação de um ChargingReply onde será feita a Reply ao pedido, aí será feito o pagamento, mediante o número de SU's pretendidas no pedido, uma de cada vez, e verificando se o cliente não ultrapassa para valores negativos, caso ultrapasse avisa o cliente que não pode fazer mais transações e que esgotou o seu crédito e atualiza as informações na Billing Account do cliente e cria e adiciona o CDR à lista do cliente, caso seja bem sucedido, avisa o cliente com um "OK" e atualiza as informações, como foi dito anteriormente.

Passando para a decisões de design que tomei ao longo da resolução deste desafio, eu utilizei apenas HashMaps, pois achei que seriam mais fáceis de organizar os CDR's, Buckets, Counters, e mais fáceis de procurar o resultado pretendido, pois basta procurar por número de telemóvel, no caso do CDR ou pelo nome do Bucket/Counter. Quanto ao resto, na minha ótica foi a maneira mais fácil de implementar as funcionalidades pedidas da melhor maneira. E quanto ao uso "exaustivo" de if statments, eu poderia usar switch cases, porém, sinto-me mais confortável a usar if's neste momento, pois foi mais utilizado durante o meu período académico.

Realizado por: João Silva