Sistema de Faturamento e Cobrança em Telecomunicações com Tarifários Dinâmicos -- Documentação

Por Joaquim Andrade 23/10/23

Contexto

Uma operadora de telecomunicações deseja aprimorar seu sistema para gerenciar pedidos de cobrança (Charging Request/Reply) e contabilização de uso (Billing Account e CDRs - Call Detail Record), com base em tarifários complexos.

Software Usado

Visual Studio Code: ide moderno ao qual estou habituado. Embora tenha usado eclipse para os projetos de java o vscode começa a ter features interessantes no desenvolvimento em java.

Java –version: java 21.0.1 2023-10-17 LTS nenhuma razão em particular.

Junit: livraria para testes;

Github e Onedrive: backup e controlo de versão.

Ficheiros incluídos

Source code: dentro de /src

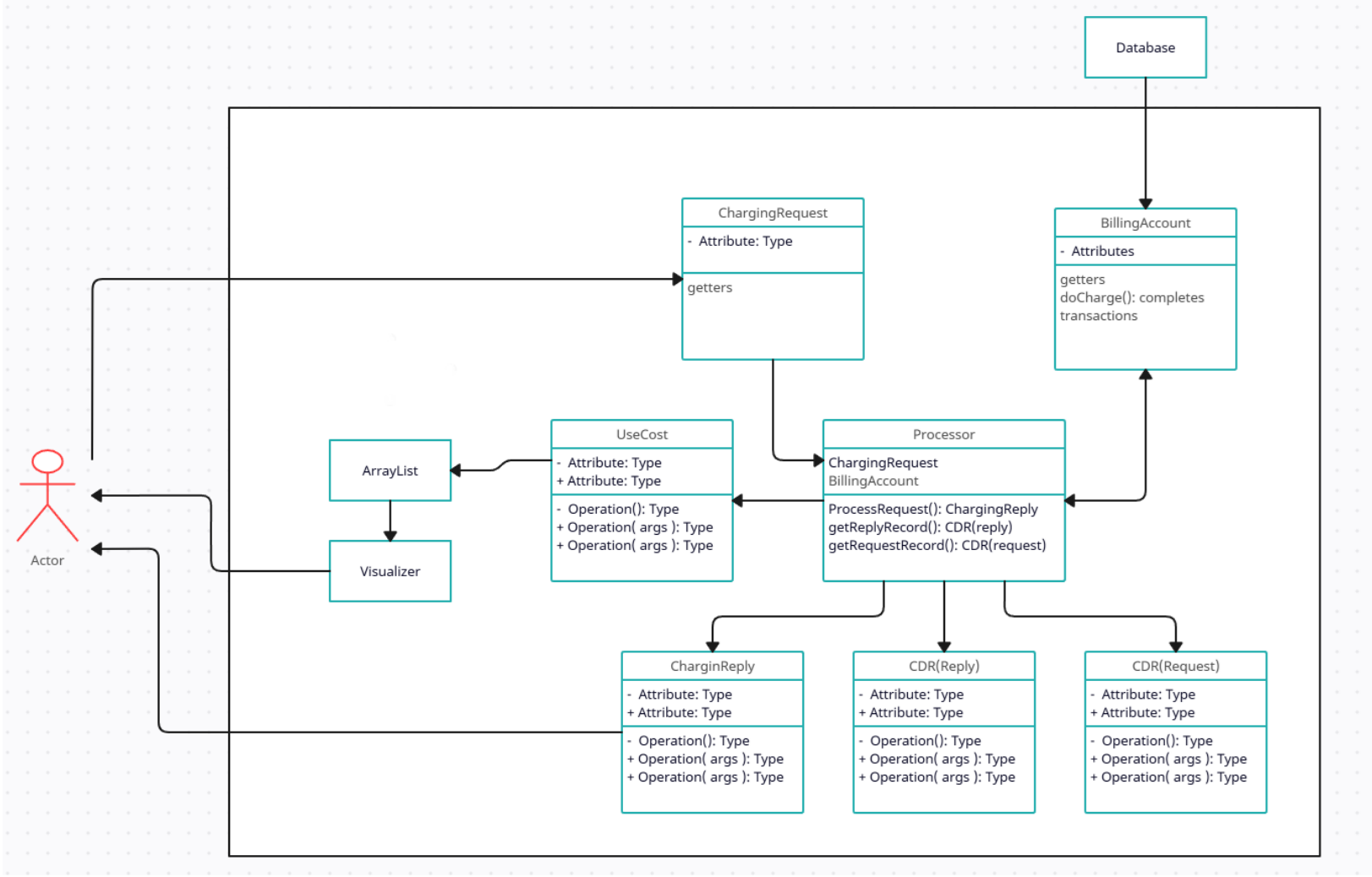
Test code: dentro de /test

Documentação: doc.pdf

Test examples: /populate*

*deve ser alterado o path dos ficheiros no main da App.java.

Resolução do problema



É recebido um ChargingRequest, este procura na lista de BillingAccounts por uma conta com o mesmo msisdn. Após encontrar é iniciado um Processor. Esta classe verifica a eligibilidade, calcula o rate, o preço final, cria os CDR(Client Records) e os UseCost. Cria também um ChargingReply que responderia ao Request.

ChargingRequest

```
private String requestId;
private Timestamp timeStamp;
private String service;
private boolean isRoaming;
private String msisdn;
private int rsu;
```

Esta classe permanece inalterada ao longo do programa. Apenas fornece a data referente ao request ao Processor.

BillingAccount

```
private String msisdn;
private int[] bucket;
private int[] counter;
private LocalDateTime counterD;
private String tariffServiceA;
private String tariffServiceB;
```

Esta classe é criada com dados do utilizador armazenados. Os seus vários parâmetros vão ser alterados assim que no processor for executada a função doCharge. Esta função altera o valor do bucket desejado e incrementa os contadores.

Processor

```
private ChargingRequest request;
private BillingAccount account;
private ChargingReply reply;
private CDR requestRecord;
private CDR replyRecord;
private String  message = "OK";
private double  cost;
private int     bucket;
```

```
private int     gsu;
private String  tariffService;
private double  finalPrice;
```

Recebe uma BillingAccount e um ChargingRequest.

Com a função processRequest() cria um registo(CDR) do ChargingRequest e do ChargingReply, verifica a eligibilidade do request (verifyEligibility()), encontra o rate em que o serviço se encontra(checkRating()) e a quantidade de fundos do bucket ao qual deseja retirar o custo do serviço. Devolve um ChargingReply após concluir a transação.

Caso esta não se conclua devolve um reply com a mensagem de erro.

ChargingReply

```
private String requestId;
private String result;
private int gsu;
```

Resposta ao request.

UseCost

```
private Timestamp timeStamp;
private int gsu;
private double  cost;
private String msisdn;
private double  finalPrice;
```

Classe criada para facilitar o fornecimento dos dados de transações pelo visualizer.

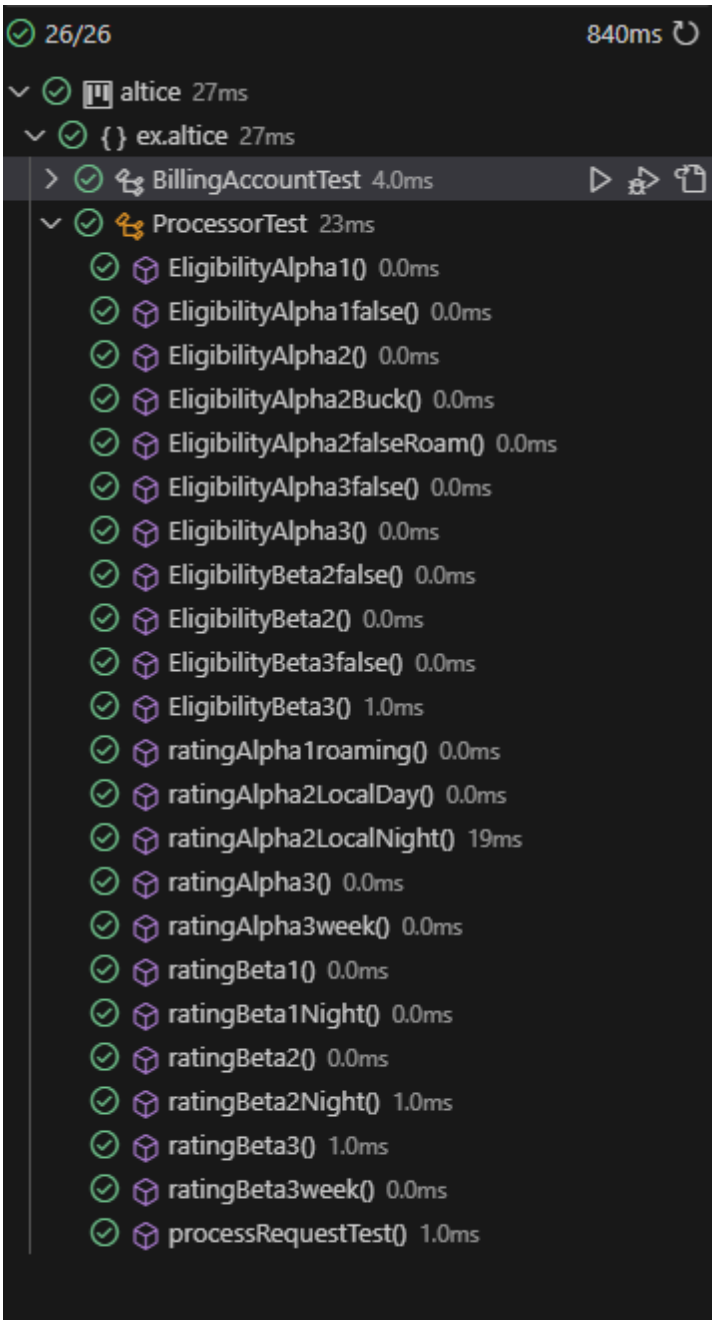
Vizualizer

Permite ao user aceder a informações sobre as transações através de uma interface gráfica.

Testes

Com o junit foram testados alguns métodos internos da classe Processor e da classe BillingAccount. Foram criados diversos testes para cada tarifário, no entanto não foram testadas todas as possibilidades. Também foi testado o programa como um todo criando ArrayLists de Contas e de Requests, populados por um ficheiro exterior.

Todos os testes do junit foram passados com sucesso. Os erros encontrados com o program inteiro foram corrigidos, e, embora certamente haja mais, não foram encontrados por mim antes da deadline.



Comentários

O diagrama acima explica as interações do problema. Penso que o mesmo tenha sido implementado de forma bastante simples.

Gostaria de ter parametrizado os tarifários de forma a ser mais simples a sua alteração e análise. Talvez criar uma classe Serviço que iria dar "extend" ao Tarifário, no entanto quando reparei já era tarde.

Quanto á robustez penso que poderia ter sido um bocado melhor e ter criado mecanismos de gestão de falhas.

Haviam algumas variáveis e questões bastante ambíguas às quais conscientemente decidi conforme o meu senso comum.

Foi um exercício engraçado e uma boa forma de passar o fim de semana e gastar uma embalagem de café.

Peço que independentemente do que acontecer, me enviem feedback do exercício para joaquimandrade@ua.pt. Obrigado.