Programação Orientada a Objetos
Trabalho Prático 2
Sistema de Operadora de Celular

André Lage e Augusto Mafra

[1]Universidade Federal de Minas Gerais

1. Introdução

Este trabalho implementa um sistema de operadora de telefonia celular utilizando conceitos de programação orientada a objetos na linguagem Java. O sistema desenvolvido suporta as operações requeridas para um sistema desse tipo, tais como cadastro e remoção de celulares, gerenciamento de celulares pré e pós pagos e registro de chamadas, por meio de uma interface simples de linha de comando.

A partir do diretório base do repositório TP2_POO, o código fonte do trabalho pode ser compilado no terminal *Unix* pelo comando `make`:

```
1 |     $ make
```

No mesmo diretório, o programa pode ser iniciado pela classe `Console` no interpretador Java, sendo passada opcionalmente a configuração para o modo teste. Nesse modo, o sistema já é iniciado com um conjunto de clientes, celulares e planos previamente carregados, para facilitar o teste de suas funcionalidades.

```
1 |     $ java Console
2 |     $ java Console -teste # para iniciar modo teste
```

2. Implementação

O sistema é organizado de acordo com a arquitetura representada no diagrama de classes UML da figura 1. Nessa seção, serão detalhados aspectos de implementação das classes propostas.

2.1. Console

A classe console é a responsável pelas interações com o usuário, envolvendo leitura de comandos e escrita no terminal por meio de um membro estático da classe Scanner.

Todos os métodos dessa classe são estáticos e privados, partindo da suposição de que o sistema requer apenas um Console. Seu funcionamento é baseado na execução da função `executarLinhaDeComando` em *loop*, que requisita informações do usuário pelas funções `prompt`, realiza a chamada de diferentes métodos da classe operadora e repassa as saídas obtidas para o terminal.

2.2. Cliente

Essa classe modela o Cliente no sistema de operadora de celular. Objetos desse tipo mantém *strings* para representar seu nome, endereço e CPF/CNPJ, além de uma lista de Celulares para armazenar os celulares possuídos pelo cliente. Todos os seus métodos são basicamente *getters* e *setters* para esses campos, ou funções *add* e *remove* para a lista de celulares.

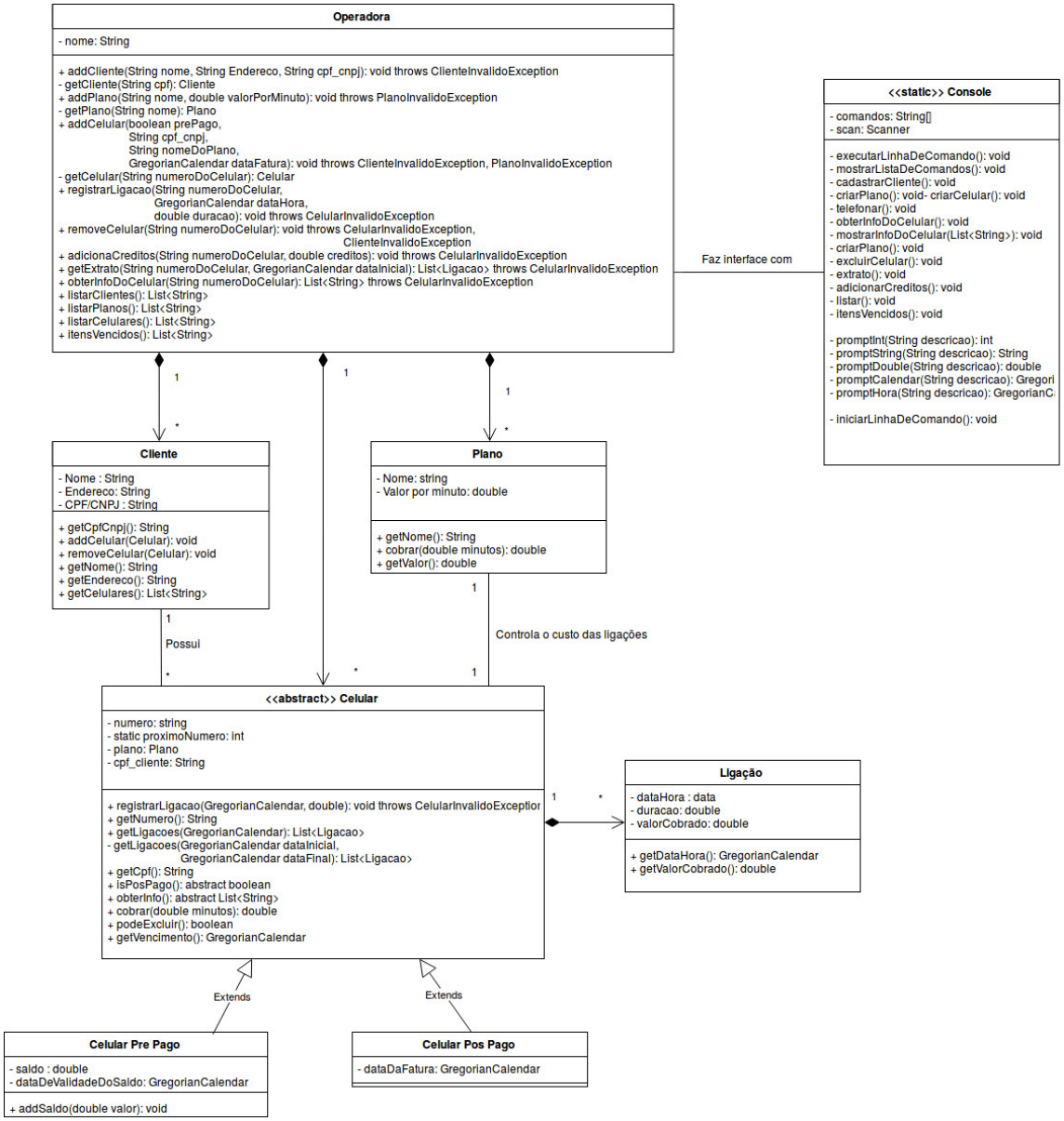

Figura 1. Diagrama de classes UML para o sistema proposto

2.3. Plano

Essa classe é utilizada para modelar os planos disponíveis para os celulares na operadora. Instâncias de Plano são armazenadas na Operadora e referenciadas por objetos Celular. Além de métodos de acesso para os campos ”nome”e ”valor por minuto”, essa classe inclui a função `cobrar`, que calcula o preço total em função de uma dada duração em minutos para uma ligação.

2.4. Celular

Utiliza-se essa classe abstrata para definir a estrutura básica do modelo para o celular na operadora. Essa classe possui como variáveis internas o número do celular, o cpf do cliente relacionado, uma lista com as ligações realizadas e um objeto plano, contendo as informações acerca do plano utilizado.

A classe é implementada nas suas duas subclasses : CelularPrePago e CelularPosPago, para o caso de celulares de cartão ou conta. Os celulares de cartão possuem como variáveis o saldo e sua validade, enquanto os pós pagos possuem apenas a data da fatura armazenada internamente. As subclasses implementam os métodos `registrarLigacao`, `isPosPago`, `obterInfo`, `podeExcluir` e `getVencimento` da classe base, cada uma inserindo nessas funções o seu comportamento específico.

2.5. Operadora

A classe operadora é a principal classe da aplicação, responsável por interligar todas as demais classes. Sua interface é representada a seguir, com a descrição dos seus principais métodos nas seções adiante:

```
1  public class Operadora {
2      private List<Cliente> clientes;
3      private List<Plano> planos;
4      private List<Celular> celulares;
5
6      private Cliente getCliente(String cpfCnpj);
7      private Plano getPlano(String nome);
8      private getCelular(String numero);
9
10     public void addCliente(String, String, String) throws
           ClienteInvalidoException;
11     public void addPlano(String, double) throws
           PlanoInvalidoException;
12     public String addCelular(boolean, String, String, Calendar)
           throws ClienteInvalidoException, CelularInvalidoException
           ;
13     public void registrarLigacao(String, Calendar, double) throws
            CelularInvalidoException;
14     public void removeCelular(String) throws
           CelularInvalidoException, ClienteInvalidoException;
15     public void adicionaCreditos(String, double) throws
           CelularInvalidoException;
16     public List<Ligacao> getExtrato(String, Calendar) throws
           CelularInvalidoException;
17     public List<String> obterInfoDoCelular(String) throws
           CelularInvalidoException;
18     public List<String> listarCelulares();
19     public List<String> listarPlanos();
20     public List<String> listarClientes();
21     public List<String> itensVencidos();
22 }
```

2.5.1. addCliente

Método responsável por adicionar um novo cliente ao objeto Operadora. Essa função verifica se o CPF/CNPJ do cliente já foi previamente cadastrado, e lança nesse caso uma exceção de `ClienteInvalidoException`.

2.5.2. addPlano

Ocorre a chamada do construtor da classe Plano, tendo como base os parâmetros passados pela classe console. O objeto criado é adicionado a lista de planos do operadora. Caso o nome passado para o plano já exista na operadora, a execução é interrompida com a exceção `PlanoInvalidoException`.

2.5.3. addCelular

Adiciona um celular à operadora. O método chama o construtor da classe Celular, e adiciona o objeto resultante na lista de celulares da operadora. Como parâmetros para o construtor são utilizados o cpf/cnpj, o plano e o tipo de celular. Todos esses parâmetros são fornecidos pelo console.

A validade dos argumentos ”cpfCnpj”e ”nomeDoPlano”passados pelo usuário é verificada com as listas de planos e clientes cadastrados. Caso algum não corresponda a um item válido, são lançadas exceções `ClienteInvalidoException` ou `PlanoInvalidoException`.

2.5.4. registrarLigacao

Registra uma ligação em um celular. Utiliza como parâmetros o número do celular, a data e a duração, em minutos. Lança exceção `CelularInvalidoException` caso o número de celular não exista, ou caso o celular seja pré-pago e tenha saldo insuficiente ou fora da validade para a ligação.

2.5.5. removeCelular

O método checa se um determinado objeto celular possui ligações a serem pagas ou créditos, e em caso negativo, o exclui. Caso contrário, a execução é interrompida com uma exceção `CelularInvalidoException`.

2.5.6. adicionaCreditos

Após requisitado pela operadora, esse método adiciona créditos em um objeto celular do tipo pré pago e adiciona mais 180 dias para o seu prazo de validade. Caso o celular requisitado pelo usuário não exista ou seja pós-pago, essa função lança uma exceção `CelularInvalidoException`.

2.5.7. getExtrato

Utilizando os parâmetros numero do celular e uma determinada data, fornecidos pelo console, o método obtém todas as ligações efetuadas pelo celular a partir da data. Essas ligações são repassadas para o console na forma de uma lista de objetos Ligação. Objetos

dessa classe, por sua vez, já tem sobrescrito o método toString, utilizado para a impressão de suas informações no console.

Essa função gera exceções `CelularInvalidoException` caso o número de celular requisitado não exista.

2.5.8. obterInfoDoCelular

Método responsável por repassar para o console diversas informações sobre o celular requisitado. A função recebe como parâmetro um número de celular e retorna uma string list, contendo o tipo de celular(pre ou pós pago), o plano, o saldo(para o caso do pré pago) e as datas de validade ou vencimento de fatura.

2.5.9. ListarCelulares, ListarPlanos, ListarClientes

Após um usuário requisitar o comando *listar*, esses métodos são os responsáveis por buscas as informações sobre usuários, planos e clientes

2.5.10. itensVencidos

Analisa as datas de vencimento de fatura e validade de créditos dos celulares, retornando para a aplicação console os números vencidos, além das informações dos clientes relacionados a estes números.