1. INTRODUÇÃO

Este projeto consiste em um sistema para controle de uma operadora de telefonia celular ao manter e gerenciar seus números de celular e os dados de seus clientes, suas ligações e planos.

O diagrama UML do projeto em si apresenta apenas os atributos, tendo em vista que, devido à grande quantidade de métodos, o mesmo ficaria muito poluído. Os métodos serão identificados e melhor detalhados posteriormente.

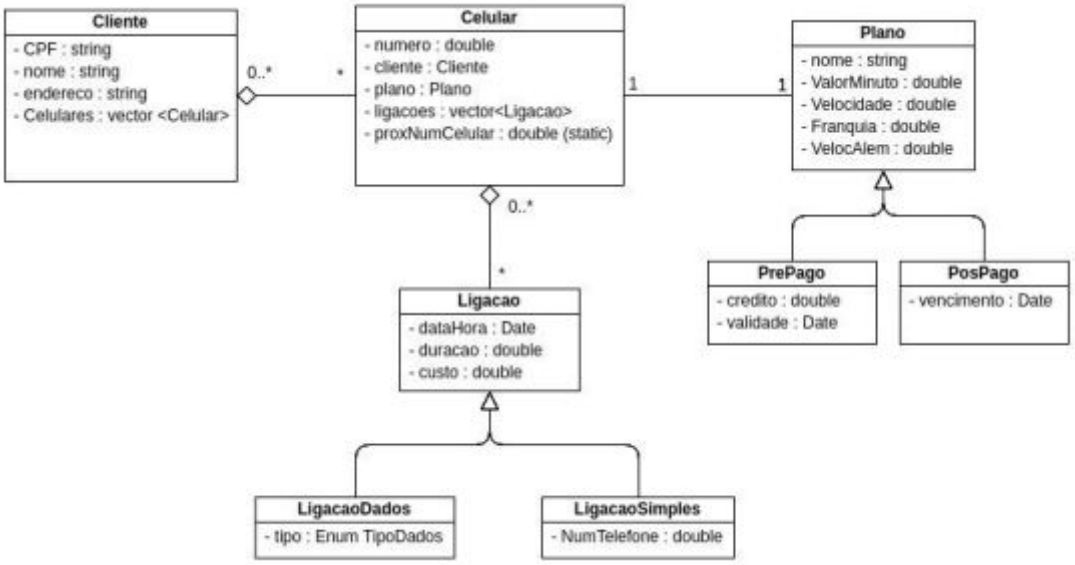

Figura 01 - Diagrama UML do sistema

A ideia é que os conceitos aprendidos durante todo o semestre em sala de aula, tais como, uso de ponteiros para trabalhar com a alocação dinâmica de memória para armazenar as estruturas criadas, sobrecarga de funções, funções constantes, funções virtuais, polimorfismo, classes abstratas, herança, exceções e outros conceitos.

2. IMPLEMENTAÇÃO DAS CLASSES

2.1. Classe Cliente

Descrição:

Os clientes são identificados por um CPF ou por um CNPJ e tem ainda atrelados a si, seu nome e seu endereço. Cada cliente pode ter quantos celulares (linhas telefônicas) forem necessários. Ao adquirir um novo celular, o cliente deve aderir a um dos planos existentes da operadora e recebe, automaticamente, um novo número.

```
class Cliente{

    string CPF;
    string nome;
    string endereco;
    vector<Celular*> celulares;

public:

    Cliente(string c_cpf, string c_nome, string c_endereco);
    void imprime_dados();
    void imprime_celulares();
    void AdicionaCelular(Celular* NovoCelular);
    ~Cliente();

};
```

Figura 02 - Classe Cliente

Métodos:

- ***Cliente****(string c_cpf, string c_nome, string c_endereco);*
  Construtor padrão da classe. Recebe os atributos da função: cpf, nome e endereço do cliente.

- ***void*** *imprime_dados();*
  Exibe na tela os atributos da classe.

- ***void*** *imprime_celulares();*
  Exibe na tela os celulares do cliente, contendo o plano dos respectivos números.

- ***void*** *AdicionaCelular(Celular* NovoCelular);*
  Recebe como parâmetro um número de celular que é adicionado ao cliente. Dessa forma, a variável recebida é adicionada ao atributo do tipo vector “celulares”.

2.2. Classe Celular

Descrição:

Cada celular está associado a um único cliente, a um único número e a um único plano. Além disso, há um registro das ligações realizadas por cada um dos celulares dos seus clientes.

```
class Celular{


    Cliente* cliente;
    PosPago* planopos = NULL;
    PrePago* planopre = NULL;
    double numero;
    static double proxNumCelular;
    vector<LigacaoSimples*> ligacoesS;
    vector<LigacaoDados*> ligacoesD;
public:
    Celular (Cliente* usuario, PosPago* planpos);
    Celular (Cliente* usuario, PrePago* planpre);
    void imprime_cliente();
    void imprime_plano();
    void imprime_numero();
    void verifica_numero(double num_cel, double valor=0.0);
    void verifica_pacote_internet(double num_cel);
    void registra_ligacao(double num_cel, string data_hora, double duracao);
    void extrato_ligacao(double num_cel, string data);
    void registra_ligacao_Dados(double num_cel, string data_hora, double duracao, string tipo);
    void extrato_ligacao_dados(double num_cel, string data);
    void verifica_valor_conta(double num_cel);
    void alerta();
    ~Celular();
};
```

Figura 03 - Classe Celular

Métodos:

- ***Celular** (Cliente* usuario, PosPago* planpos);*
  ***Celular** (Cliente* usuario, PrePago* planpre);*

  Construtores padrão da classe. Recebe uma classe de cliente e uma classe do plano, podendo ser Pós Pago ou Pré Pago.

- ***void** imprime_cliente();*

  Chama o método da classe Cliente “imprime_dados”, que exibe na tela o nome, CPF e o endereço do cliente.

- ***void** imprime_plano();*

  Chama o método da classe Plano “imprime”, que exibe na tela o nome do plano, valor do minuto, velocidade, franquia e a velocidade após a franquia.

- ***void** imprime_numero();*

  Exibe na tela o atributo “double numero”, que contém o número do celular.

- ***void** verifica_numero(double num_cel, double valor=0.0);*

  Esse método tem como função ou verificar os créditos do número fornecido ou adicionar crédito, valor passado como parâmetro, ao número. O primeiro caso ocorre quando não é passado como parâmetro a variável “double_valor”, fazendo com que o método atribua a ela o valor default.

- ***void** verifica_pacote_internet(double num_cel);*

Verifica o pacote de internet para um determinado número de celular seja ele pós ou pré-pago retornando o quanto foi gasto e a velocidade ou se a franquia já foi excedida.

- ***void*** *registra_ligação (double num_cel, string data_hora, double duracao);*

Função para o registro de uma ligação simples em um determinado número de telefone. Está é uma função que trata de maneira diferente os planos pós e os pré-pagos.

Nos números pós-pagos, basta registrar a ligação em termos de sua data, hora, duração e valor. Já para os pré-pagos, o valor do crédito deve ser conferido bem como a validade deste. Feito isso, a ligação é realizada nos mesmos moldes do que ocorre no pós-pago e, ao final, o valor gasto é decrementado do valor de crédito.

- ***void*** *extrato_ligação (double num_cel, string data);*

Retorna as ligações efetuadas por um determinado numero de celular a partir de uma determinada data.

- ***void*** *registra_ligacao_Dados(double num_cel, string data_hora, double duracao, string tipo);*

Função para o registro de uma ligação de dados em um determinado número de telefone. Está é uma função que trata de maneira diferente os planos pós e os pré-pagos.

Para os planos pós-pagos, observa-se se o plano possui liberação para internet ou não. Caso possua, compara-se o valor de dados consumido e o valor da franquia e caso o valor consumido seja maior, a velocidade é reduzida. Feito isso, a ligação ocorre e é registrada (em termos de data, hora, duração e custo). Ao término, a quantidade de dados consumida é somada ao valor consumido no mês.

Para os planos pré-pagos o procedimento é semelhante, entretanto, antes disso, verifica-se o valor de crédito e sua validade. Ao término, desconta-se o valor gasto do valor dos créditos daquele número.

- ***void*** *extrato_ligacao_dados(double num_cel, string data);*

Retorna o extrato de internet efetuadas por um determinado numero de celular a partir de uma determinada data.

- ***void*** *verifica_valor_conta (double num_cel);*

Retorna ao usuário o valor total da conta. Para isso, ele verifica se o número é pós-pago e, após isso, observa a data de vencimento, a data atual e com base nela qual é o mês anterior e com isso totaliza-se as ligações e seus valores.

- ***void*** *alerta();*

Esta é uma função que alerta o usuário que um determinado número de celular ultrapassou a franquia de dados.

2.3. Classe Plano

Descrição:

A operadora sendo criada possui duas opções de planos e eles podem ser: cartão/pré-pago ou assinatura/pós-pago

```
class Plano{

protected:

    string nome;
    double valorMinuto;
    double velocidade;
    double franquia;
    double velocAlem;
    double franquia_gasta=0;

public:

    Plano(string c_nome, double ValMin, double vel, double franq, double veloc);
    virtual void imprime();
    virtual double val_minuto();
    double valor_franquia();
    double valor_franquia_gasta();
    double Valor_velocidade();
    double Valor_velocAlem();
    void franquia_gast(double valor);
    void verifica_internet();
    ~Plano();
};
```

Figura 04 - Classe Plano

Métodos:

- ***Plano****(string c_nome, double ValMin, double vel, double franq, double veloc)*
  Construtor padrão da classe. Recebe os atributos da classe como seus parâmetros.

- ***virtual void*** *imprime();*
  Exibe na tela o nome do plano, valor do minuto, velocidade, franquia e a velocidade após a franquia.

- ***virtual double*** *val_minuto();*
  Retorna o valor do minuto de um determinado plano.

- ***double*** *valor_franquia();*
  Retorna a franquia de internet de um determinado plano.

- ***double*** *valor_franquia_gasta();*
  Retorna o valor da franquia gasta por um telefone.

- ***double*** *Valor_velocidade();*
  Retorna o valor da velocidade de uma determinado plano.

- ***double*** *Valor_velocAlem();*
  Retorna o valor da velocidade de um plano ao exceder a franquia.

- ***void** franquia_gast(double valor);*
  Adiciona o valor de uma ligação de dados ao valor já gasto por um número de celular.

- ***void** verifica_internet();*
  Retorna o extrato de internet para o usuário.

2.4. Classe PrePago

Descrição:

A classe PrePago é um tipo de Plano. Para celulares que possuem um plano pré-pago, a operadora deve controlar a disponibilidade dos créditos bem como suas validades.

```
class PrePago: public Plano{

    double credito=0.0;
    string data_validade;

public:

    PrePago(string c_nome, double ValMin, double vel, double franq,
            double veloc, double cred = 0.0, string dataval = "Vencido");
    void imprime();
    void adiciona_cred(double valor);
    void verifica_cred();
    double valor_cred();
    string data_val();
    void subtrai_cred(double valor);
    ~PrePago();
};
```

Figura 05 - Classe PrePago

Métodos:

- ***PrePago**(string c_nome, double ValMin, double vel, double franq, double veloc, double cred = 0.0, string dataval = "Vencido");*
  Construtor padrão da classe. Recebe o nome do plano, valor do minuto, a velocidade, franquia, velocidade após a franquia, o valor dos créditos e a validade dos créditos.

- ***void** imprime();*
  Exibe todas as variáveis recebidas pelo construtor.

- ***void** adiciona_cred(double valor);*
  Adiciona na variável “crédito” o valor passado como parâmetro. Além disso, define a data de validade dos créditos colocados.

- ***void** verifica_cred();*
  Exibe na tela a quantidade de créditos e a data de validade destes.

- ***double*** *valor_cred();*
  Retorna o valor de crédito disponível para um determinado número.

- ***string*** *data_val();*
  Retorna a data de validade dos créditos disponíveis para um determinado número.
- ***void*** *subtrai_cred(double valor);*
  Subtrai valor da ligação do total de créditos de um celular.

2.5. Classe PosPago

Descrição:

A classe PosPago é um tipo de Plano. Para celulares que possuem um plano pós pago, a operadora deve armazenar apenas o dia de vencimento da conta do cliente.

```
class PosPago: public Plano{

    string vencimento;

public:
    PosPago(string c_nome, double ValMin, double vel, double franq, double veloc, string venc);
    void imprime();
    string data_vencimento();
    ~PosPago();
};
```

Figura 06 - Classe PosPago

Métodos:

- ***PosPago****(string c_nome, double ValMin, double vel, double franq, double veloc, string venc);*
  Construtor padrão da classe. Recebe o nome do plano, valor do minuto, a velocidade, franquia, velocidade após a franquia e o dia do vencimento da fatura.

- ***void*** *imprime();*
  Exibe todas as variáveis recebidas pelo construtor.

- ***string*** *data_vencimento();*
  Retorna a data de vencimento das faturas de um determinado celular pós-pago.

2.6. Classe Ligacao

Descrição:

Para qualquer tipo de ligação, independentemente de qual tipo ela seja, a operadora registra a data, o horário, a duração em minutos e o custo desta.

```
class Ligacao{

protected:
    string dataHora;
    double duracao;
    double custo;

public:
    Ligacao(string data, double c_duracao, double valor_minuto);
    virtual void verifica_data(string data) = 0;
    ~Ligacao();
};
```

Figura 07 - Classe Ligação

Métodos:

- ***Ligacao****(string data, double c_duracao, double c_custo);*
  Construtor padrão da classe. Recebe a data, a duração e o custo da ligação.
- virtual void verifica_data(string data);
  É uma função que é sobre-escrita dentro das classes LigacaoDados e LigacaoSimples.

2.7. Classe LigacaoDados

Descrição:

As ligações de dados são um tipo de (classe) Ligação. Essas são as ligações feitas via consumo de dados de internet. Quando uma ligação desse tipo é realizada além da data, do horário e da duração devem ser registrados as informações referentes ao consumo de dados seja via upload ou via download.

```
class LigacaoDados: public Ligacao{
    string tipoDeDados; //upload ou download
    double dados_gasto;
public:
    LigacaoDados(string data, double c_duracao, double valor_minuto, string tipo_dados, double dados_gast);
    void verifica_data(string data);
    ~LigacaoDados();
};
```

Figura 08 - Classe LigaçãoDados

Métodos:

- **LigacaoDados**(string tipo_dados);
  Construtor padrão da classe. Recebe os atributos para a ligação (data, duração, valor do minuto, tipo de dado: upload ou download e a quantidade de dados gastos).
- **void** verifica_data(string data)
  Retorna extrato do consumo de dados de um celular ao usuário, exibindo a data da ligação, duração da ligação, os dados consumidos e tipo dos dados.

2.8. Classe LigacaoSimples

Descrição:

As ligações simples, assim como as ligações de dados também registram a data, o horário e a duração de cada chamada. Além disso no caso das ligações simples o número para o qual a chamada foi feita também deve ser registrado.

```
class LigacaoSimples: public Ligacao{
public:
    LigacaoSimples(string data, double c_duracao, double valor_minuto);
    void verifica_data(string data);
    double valor_ligacao(string mes_anterior, string mes_posterior);
    ~LigacaoSimples();
};
```

Figura 09 - Classe LigaçãoSimples

Métodos:

- **LigacaoSimples**(double Num_Celular);
  Construtor padrão da classe. Recebe o atributo da função, contendo o número do celular.
- **void** verifica_data (string data);
  Chama o construtor da classe Ligação com o a data, duração e o valor do minuto.

- **double** valor_ligacao (string mes_anterior, string mes_posterior)
  Verifica quais ligações estão dentro do prazo vigente da fatura e retorna qual é o custo.

3. INTERFACE

Para a operação do sistema, foram implementadas algumas opções. São permitidas as seguintes operações:

3.1. Cadastro

Permite a realização do cadastro de novos clientes. Para isso, o usuário deve passar os dados do cliente (nome, CPF/CNPJ e endereço) bem como qual é o plano associado.

```
///Usuario Adiciona Clientes

Cliente *A = new Cliente("123.456.789-10", "Satoshi", "Japan");
clientes.push_back(A);
Cliente *B = new Cliente("111.222.333-44", "Mitinick", "USA");
clientes.push_back(B);
```

Figura 10 - Processo de adição de novos clientes

Também é possível realizar o cadastro de novos planos. Para isso, o usuário deve inserir o nome do plano, o valor do minuto de ligação, a velocidade de download e de upload e a franquia do plano.

```
///Usuario Adiciona Planos

Plano *PreLigacao = new Plano("Pre-Pago Ligacao", 0.50, 1.0, 100, 0.1);
planos.push_back(PreLigacao);
Plano *PreInternet = new Plano("Pre-Pago Internet", 0.80, 5.0, 1000, 0.5);
planos.push_back(PreInternet);
Plano *PosControle = new Plano("Pos-Pago Controle", 0.30, 10.0, 5000, 1.0);
planos.push_back(PosControle);
Plano *PosInfinity = new Plano("Pos-Pago Infinite", 0.15, 50.0, 50000, 10.0);
planos.push_back(PosInfinity);
```

Figura 11 - Processo de adição de novos planos

3.2. Criação de novos celulares

A habilitação implica na geração automática do número por parte do sistema usando como base um atributo estático que tem a função de armazenar o próximo número a ser criado.

A criação de um novo número deve estar associado a um cliente já cadastrado e deve estar associado a um tipo dos tipos de plano. Caso seja um plano pós-pago, deve-se definir a data de vencimento da conta.

```
///Usuario habilita um novo celular em plano pós-pago

PosPago plano3("Pos-Pago Infinite", 0.15, 50.0, 50000, 10.0, "30");
Celular *clt1 = new Celular(clientes[0], &plano3);
celulares.push_back(clt1);
clientes[0]->AdicionaCelular(celulares[0]);

///Usuario habilita um novo celular em plano pré-pago

PrePago plano1("Pre-Pago Internet", 0.80, 5.0, 1000, 0.5);
Celular *clt2 = new Celular(clientes[1], &plano1);
celulares.push_back(clt2);
clientes[1]->AdicionaCelular(celulares[1]);
```

Figura 12 - Processo de habilitação/criação de novos números de celulares

3.3. Adicionar crédito

Permite o procedimento de recarga dos celulares que são pré-pagos. Para isso, o usuário fornece o número do celular a ser recarregado e o valor a ser adicionado. Com isso, o novo valor é somado ao valor já existente e a validade dos créditos é atualizada para 180 dias corridos após a data de recarga.

```
///Usuario Adicina creditos a um celular pre-pago

for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->verifica_numero(977031,35);
}
```

Figura 13 - Processo de adicionar créditos a um celular pré-pago

3.4. Registro de ligação

É o procedimento que permite a simulação da realização de uma ligação. É permitido o registro de ligações apenas após a data de sua realização.

3.4.1. Ligação de Dados

```
///Usuario pos-pago Registra ligação de dados

for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->registra_ligacao_Dados(977021, "22.06.2019 - 07:30:00", 100, "dowload");
}

///Usuario pre-pago Registra ligação de dados

for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->registra_ligacao_Dados(977031, "22.06.2019 - 10:40:00", 200, "dowload");
}
```

Figura 14 - Processo de registro de ligações de dados

3.4.2. Ligação Simples

```
///Usuario pos-pago Registra ligação simples

for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->registra_ligacao(977021, "21.06.2019 - 18:30:00", 10);
    celulares[i]->registra_ligacao(977021, "21.06.2019 - 21:30:00", 40);
}

///Usuario pre-pago Registra ligação simples

for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->registra_ligacao(977031, "21.06.2019 - 20:30:00", 20);
}
```

Figura 15 - Processo de registro de ligações simples

3.5. Valor da Conta

Verifica o valor da conta de um determinado celular exibindo-o para o usuário.

```
///Usuario verifica valor da conta de celular pos-pago
cout << endl <<"Cliente (Sr. Satoshi) verifica valor da conta: " << endl;
for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->verifica_valor_conta(977021);
}
```

Figura 16 - Processo de exibição do valor da conta de determinado celular

3.6. Créditos disponíveis

```
///Usuario Verifica o valor e a validade dos creditos de um celular pre-pago

cout << "Cliente (Sr. Mitinick) verifica credito: " << endl;
for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->verifica_numero(977031);
}
```

Figura 17 - Processo de verificação de créditos

3.7. Extrato de ligações

3.7.1. Simples

```
///Extrato Ligacoes Simples usuario pos-pago

cout << endl << "Extrato Ligacoes (Sr. Satoshi): " << endl;
for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->extrato_ligacao(977021, "20.06.2019");
}

///Extrato Ligacoes Simples usuario pre-pago

cout << endl << "Extrato Ligacoes (Sr. Mitinick): " << endl;
for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->extrato_ligacao(977031, "20.06.2019");
}
```

Figura 18 - Processo de criação de extratos de ligações simples

3.7.2. Dados

```
///Extrato Ligacoes Dados usuario pos-pago

cout << endl << "Extrato Consumo de Dados (Sr. Satoshi): " << endl;
for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->extrato_ligacao_dados(977021, "20.06.2019");
}

///Extrato Ligacoes Dados usuario pre-pago

cout << endl << "Extrato Consumo de Dados (Sr. Mitinick): " << endl;
for(int i=0; i<celulares.size(); i++)
{
    celulares[i]->extrato_ligacao_dados(977031, "20.06.2019");
}
```

Figura 19 - Processo de criação de extratos de ligações de dados

3.8. Listagens

3.8.1. Clientes

Impressão do nome e os dados de todos os clientes cadastrados até o momento pela operadora.

```
///Listar todos clientes
cout << endl << "Lista de todos os clientes: " << endl;
for(int i=0; i<clientes.size(); i++)
{
    cout << endl;
    clientes[i]->imprime_dados();
    clientes[i]->imprime_celulares();
}
```

Figura 20 - Processo de listagem de todos os clientes da operadora

3.8.2. Planos

Impressão de todos os planos cadastrados até o momento.

```
///Listar todos Planos
cout << endl << "Lista de todos os planos: " << endl;
for(int i=0; i<planos.size(); i++)
{
    cout << endl;
    planos[i]->imprime();
}
```

Figura 21 - Processo de listagem de todos os planos existentes

3.8.3. Celulares

Impressão de todos os números de celulares já cadastrados até o momento juntamente com o nome do respectivo cliente e plano.

```
///Listar todos Celulares
cout << endl << "Lista de todos os celulares: " << endl;
for(int i=0; i<celulares.size(); i++)
{
    cout << endl;
    celulares[i]->imprime_numero();
    celulares[i]->imprime_cliente();
    celulares[i]->imprime_plano();
}
```

Figura 22 - Processo de listagem de todos os celulares

3.9. Informativo

3.9.1. Franquia de dados

O sistema informa ao usuário por meio da impressão de uma mensagem na tela do sistema, que o limite da franquia contratada foi atingida.

```
///Alerta o usuario que um determinado numero de celular ultrapassou a franquia

void Celular::alerta()
{
    cout << endl << "Cliente Excedeu limite de dados" << endl;
    cout << "Informacoes: " << endl;
    imprime_numero();
    imprime_cliente();
    imprime_plano();
}
```

Figura 23 - Emissão do alerta para o limite de dados

4. TESTES

Para a realização dos testes, foi executado no main.cpp cada possível função da Operadora desenvolvida. Dessa forma o usuário vai comandando as ações através da interface, como citado no tópico anterior.

Com isso, a saída do sistema para as ações chamadas no código, ou seja, a exibição dos comandos na tela, ficou dessa forma:

```
Cliente (Sr. Mitinick) verifica credito:
Creditos: R$ 35
Validade dos Creditos: 21.12.2019

Cliente Excedeu limite de dados
Informacoes:
Numero: 977031
Nome do cliente: Mitinick
CPF: 111.222.333-44
Endereco: USA
Nome do Plano: Pre-Pago Internet
Valor do Minuto: 0.8 cent
Velocidade: 5 Mbps
Franquia: 1000 MB
Velocidade Apos Franquia: 0.5 Mbps
Creditos: R$ 19
Validade dos Creditos: 21.12.2019

Cliente (Sr. Satoshi) verifica pacote de internet:
Franquia: 45000MB
Velocidade Atual: 50Mbps

Extrato Ligacoes (Sr. Satoshi):

Data do Consumo: 23.06.2019 - 18:30:00
Duracao da ligacao: 10min
Custo da ligacao: R$ 1.5

Data do Consumo: 23.06.2019 - 21:30:00
Duracao da ligacao: 40min
Custo da ligacao: R$ 6

Extrato Consumo de Dados (Sr. Satoshi):

Data da Ligacao: 23.06.2019 - 07:30:00
Duracao da ligacao: 100min
Dados comsumidos:5000MB
Tipo: dowload
```

Figura 24 - Tela de testes 1

```
Extrato Ligacoes (Sr. Mitinick):

Data do Consumo: 23.06.2019 - 20:30:00
Duracao da ligacao: 20min
Custo da ligacao: R$ 16

Extrato Consumo de Dados (Sr. Mitinick):

Data da Ligacao: 23.06.2019 - 10:40:00
Duracao da ligacao: 200min
Dados comsumidos:1000MB
Tipo: dowload

Data da Ligacao: 23.06.2019 - 13:20:00
Duracao da ligacao: 10min
Dados comsumidos:5MB
Tipo: upload

Cliente (Sr. Mitinick) verifica pacote de internet:
Franquia: Franquia excedida
Velocidade Atual 0.5Mbps

Cliente (Sr. Satoshi) verifica valor da conta:
Referente a 23.05.2019 - 00:00:00 a 23.06.2019 - 23:59:59
Valor Total: R$7.5

Lista de todos os clientes:

Nome do cliente: Satoshi
CPF: 123.456.789-10
Endereco: Japan
Celulares:
Numero: 977021
Nome do Plano: Pos-Pago Infinite
Valor do Minuto: 0.15 cent
Velocidade: 50 Mbps
Franquia: 50000 MB
Velocidade Apos Franquia: 10 Mbps
Dia de vecimento da fatura: 23
```

Figura 25 - Tela de testes 2

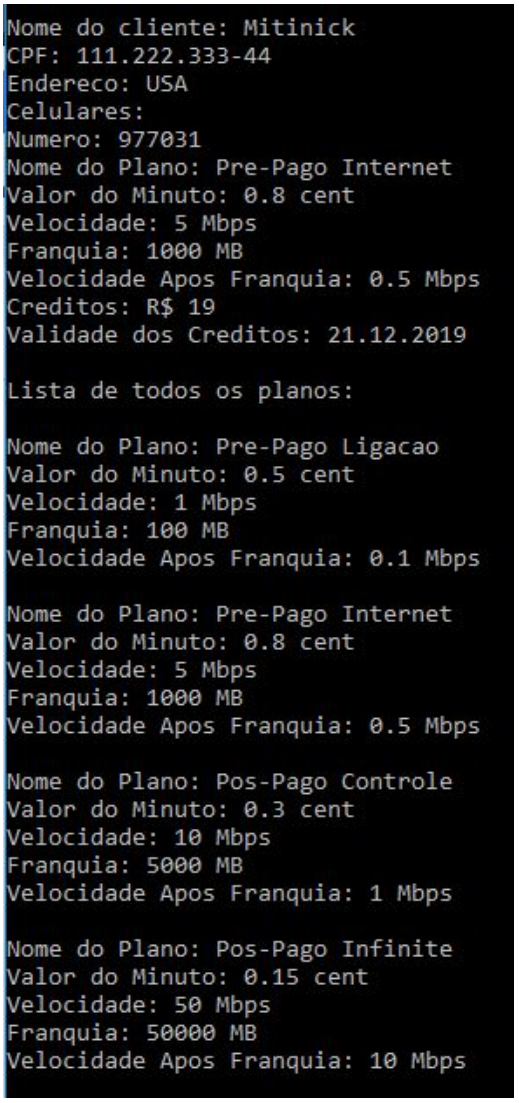

Figura 25 - Tela de testes 3

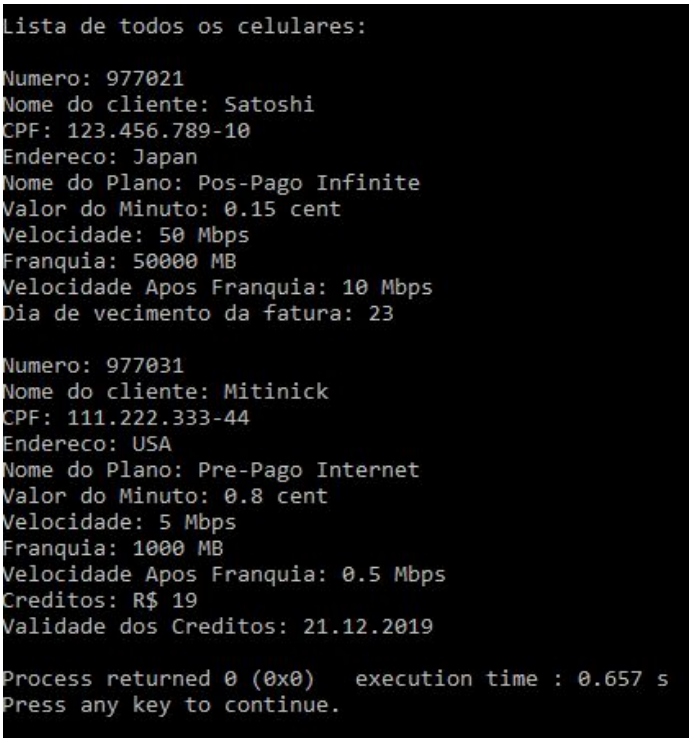

Figura 26 - Tela de testes 4

5. CONCLUSÃO

Com a execução deste trabalho foi possível a compreensão e a familiarização com muitos dos conceitos abordados durante as aulas teóricas e durante as aulas práticas no laboratório. Mais do que isso, foi possível ter um contato mais aprofundado com o paradigma da programação orientada à objetos, permitindo uma expansão das habilidades e ferramentas de desenvolvimento dos membros.

Acreditamos que, de maneira satisfatória, muito do que foi visto em sala de aula foi colocado em prática e muitas das discussões relativas às boas práticas para a programação na perspectiva da orientação à objetos passaram a fazer mais sentido e puderam também ser incorporadas quando eram cabíveis.