Instituto Federal Catarinense - Campus Rio do Sul
Faculdade de Ciência da Computação
Matéria de Estrutura de Dados I

Felipe Cauê Machado

Sistema de Restaurante em Interface de Linha de Comando (CLI), Focado na Utilização de Estrutura de Dados.

Rio do Sul
2024

Introdução

O sistema de restaurante tem como objetivo criar uma forma eficiente de gerenciar um restaurante. Neste relatório será explorado os aspectos do sistema. Desde o código, as funcionalidades e as regras de negócio. O trabalho tem um objetivo claro de mostrar a utilização de Estrutura de Dados dentro de um projeto de sistema para restaurantes. Fiz uma interface de Linha de Comando seguindo uma espécie de Modelo (MVC) onde o código se separa e Modelo, Controle e Visual. Os dados estão sendo salvos em uma classe com atributos estáticos para que seja possível utilizar como um banco de dados com acesso em um só lugar. Fiz uma lista genérica para fazer a persistência dos dados. Toda classe modelo tem um método equals onde sempre que for pedido para buscar por item os dados da lista por exemplo ele usa esse método para se localizar. As especificações mais claras serão definidas ao decorrer do trabalho. Todos os métodos getters e setters estão implícitos na documentação porem no código estão corretos **Fiz um jar para melhor visualização do sistema e para rodar é simples, somente colocar no terminal:**

```
java -jar Restaurante.jar
```

Link do GitHub: https://github.com/FelipeTMachado/restaurante.git

Código Fonte Documentado

Primeiro vou começar explicando o nodo e a lista genérica.

Nodo - O nodo é uma classe genérica que contém o atributo value do Tipo T genérico.

```
public class Nodo<T> {
    // ATRIBUTOS
    private T       value;
    private Nodo<T> next;
    private Nodo<T> back;

    // CONSTRUTOR
    public Nodo(T value) {
        this.value = value;
        next       = null;
        back       = null;
    }

    // GETTERS AND SETTERS IMPLICITOS
}
```

Lista - A lista tambem é uma classe genérica, utilizei genérico para que eu possa utilizar a mesma lista com objetos variados, obrigando aos objetos a implementação dos métodos `toString()` e `equals()`.

Método de inserção no inicio.

```
// METODOS FUNCIONAIS
public void insertBegin(T value) {

    Nodo<T> newNodo = new Nodo<T>(value);

    if (isEmpty()) {
        begin = newNodo;
        newNodo.setNext(null);
        newNodo.setBack(null);
    } else {
        newNodo.setNext(begin);
        begin.setBack(newNodo);
        begin = newNodo;
    }

    active = begin;
}
```

Método de inserção no fim.

```
public void insertEnd(T value) {
    Nodo<T> newNodo = new Nodo<T>(value);

    if (isEmpty()) {
        begin = newNodo;
        begin.setNext(null);
        begin.setBack(null);
    } else {
        Nodo<T> aux = begin;
        while (aux.getNext() != null) {
            aux = aux.getNext();
        }
        aux.setNext(newNodo);
        newNodo.setBack(aux);
    }
    active = begin;
}
```

Método de deletar por item. Esse método utiliza o equals da classe utilizada pra verificar a identidade do objeto.

```
public void deleteByItem(T item) {
    Nodo<T> aux = begin;
    while (aux != null) {
        if (aux.getValue().equals(item)) {
            if(aux == begin) {
                begin = begin.getNext();

                if (begin != null) {
                    begin.setBack(null);
                }
            } else {
                if (aux.getNext() != null) {
                    aux.getNext().setBack(aux.getBack());
                    aux.getBack().setNext(aux.getNext());

                } else {
                    aux.getBack().setNext(null);
                }
            }
        }
        aux = aux.getNext();
    }
    active = begin;
}
```

Para utilizar a iteração na lista eu fiz dois métodos, que servem para poder Iterar por fora da lista, como em um ArrayList podendo utilizar dentro de um while por exemplo. São os métodos `hasNext()` e `next()`.

Método retorna se existe um próximo elemento no nodo e se nao esta no fim da lista.

```
public boolean hasNext() {
    if (active == null) {
        active = begin;
        return false;
    } else {
        return true;
    }
}
```

Método move o elemento para o próximo e retorna o valor do nodo atual da lista.

```
public T next() {
    Nodo<T> aux = active;
    active = aux.getNext();
    return aux.getValue();
}
```

Para fazer a persistência dos dados e utilizar a Lista como uma base de dados, fiz uma classe que contem algumas listas para adicionar os dados do sistema, essas listas são setadas por uma classe **Singleton** ou seja so podem ser acessadas uma vez sem duplicatas no sistema, forcando a ter somente um lugar para as listas serem populadas como uma base de dados real. A primeira vez que é chamada a classe já cria todas as listas para serem populadas.

```
public class BaseDados {
    // ATRIBUTO
    private static BaseDados instance;
    private Lista<Pessoa>    basePessoa;
    private Lista<Pessoa>    baseFuncionario;
    private Lista<Pedido>    basePedido;
    private Lista<Mesa>      baseMesa;
    private Lista<Produto>   baseProduto;

    // CONSTRUTOR COMPOSTO SINGLETON
    private BaseDados() {
        basePessoa              = new Lista<Pessoa>();
        baseFuncionario = new Lista<Pessoa>();
        basePedido              = new Lista<Pedido>();
        baseMesa        = new Lista<Mesa>();
        baseProduto     = new Lista<Produto>();
    }

    public static BaseDados getInstance() {
        if (instance == null) {
            instance = new BaseDados();
        }
        return instance;
        }
        // GETTERs E SETTERs IMPLICITOS
}
```

Para popular as bases de dados o sistema tem alguns cadastros básicos e todos seguem o padrão model view controller (MVC), são eles: (Cliente e Funcionário) da classe Pessoa, Mesa e Produto. Vou fazer uma explicação sobre o cadastro de Cliente porém segue o mesmo padrão para as outras classes. Na Classe Pessoa que é o modelo do cliente, temos quatro atributos e dois construtores, os construtores são mais para fazer o cadastro de alguns clientes para ficar no sistema para teste. O método equals dessa classe funciona para os métodos `deleteItem()` e `findItem()` da lista. Todas as classes de modelo seguem esse padrão caso usem um dos dois métodos para comparar objeto com objeto na lista.

```
public class Pessoa {
        // ATRIBUTOS
        private String      nome;
        private String      cpf;
        private String      endereco;
        private ETipoPessoa tipo;

        // CONSTRUTOR SIMPLES
        public Pessoa(String n, String c, String e, ETipoPessoa t) {
                this.nome       = n;
                this.cpf        = c;
                this.endereco   = e;
                this.tipo       = t;
        }
        public Pessoa(String cpf) {
                this.cpf = cpf;
        }

        @Override
        public boolean equals(Object obj) {
                if(((Pessoa) obj).getCpf().equals(this.cpf)) {
                    return true;
                }
                return false;
        }
}
```

Depois do modelo temos o Controle (ControllePessoa) que faz a ponte entre o visual (VisualPessoa) e o modelo (Pessoa). A Classe controle pessoa é a classe que faz todos os métodos funcionais da pessoa.
O método principal dessa classe para o cliente é o menuCliente onde terá a regra de criação do menu de gerenciamento do cliente. Cada case desse entra nos menus específicos para manter o cliente.

```
public void menuCliente() {
    boolean ehSair = false;

    while (!ehSair) {
        visual.limparCampos();
        visual.menuCliente();
        Entrada e = Entrada.getInstance();
        String opcao = e.retornaDado("ESCOLHA SUA OPCAO: ");

        switch (opcao) {
        case "1": {
            novoCliente();
            break;
        }
        case "2": {
            buscarCliente();
            break;
        }
        case "3": {
            alterarCliente();
            break;
        }
        case "4": {
            excluirCliente();
            break;
        }
        case "9": {
            ehSair = true;
            break;
        }
        case "0": {
            System.exit(1);
        }
        default:
            Visual.getInstance().visualizarMensagemOpcaoInvalida();
        }
    }
}
```

No inicio do menuCliente tem um método da classe VisualPessoa que faz o menu de gerenciamento na interface (CLI). Esse método utiliza um Framework que eu desenvolvi para facilitar a criação da interface de linha de comando (CLI). Basicamente esses métodos imprimem na tela os menus e fazem os gerenciametos dos campos digitados pelo usuário

```
public void menuCliente() {
    Visual v = Visual.getInstance();
    v.visualizarCabecalho("SISTEMA RESTAURANTE", "GERENCIAMENTO CLIENTE");
    v.visualizarTextoAlinhadoEsquerda("1 - NOVO");
    v.visualizarTextoAlinhadoEsquerda("2 - BUSCAR");
    v.visualizarTextoAlinhadoEsquerda("3 - ALTERAR");
    v.visualizarTextoAlinhadoEsquerda("4 - EXCLUIR");
    v.visualizarEspacoEmBranco();
    v.visualizarTextoAlinhadoEsquerda("9 - VOLTAR");
    v.visualizarTextoAlinhadoEsquerda("0 - SAIR");
    v.visualizarLinha();
}
```

Alguns menus tem menus dentro deles como o menu de Cliente que imprime um menu e tem a opção buscar que vai para outro menu. Sempre que o menu tem o **ehSair** setado como true então ele volta para um menu anterior ao escolhido.

Menu de Busca de Cliente

```
public void menuBuscarCliente() {
    Visual v = Visual.getInstance();
    v.visualizarCabecalho("SISTEMA RESTAURANTE", "BUSCA CLIENTE");
    v.visualizarTextoAlinhadoEsquerda("1 - BUSCAR POR CPF");
    v.visualizarTextoAlinhadoEsquerda("2 - BUSCAR TODOS");
    v.visualizarEspacoEmBranco();
    v.visualizarTextoAlinhadoEsquerda("9 - VOLTAR");
    v.visualizarTextoAlinhadoEsquerda("0 - SAIR");
    v.visualizarLinha();
}
```

Voltando ao gerenciamento do menu na classe de controle do cliente tem o método novoCliente. Esse método popula os campos do visual e mostra os campos populados para o usuário verificar, apos isso o usuário pode escolher salvar ou nao os dados na lista. Usando o método `insertBegin()` ele adiciona na base de dados que esta em um atributo que seta a base de dados `base`

```
private void novoCliente() {
    visual.limparCampos();
    visual.novoCliente();
    visual.verificarCampos();
    Entrada e = Entrada.getInstance();

    String ehSalvar = e.retornaDado("SALVAR DADOS DO CLIENTE [S / N]: ");

    if ((ehSalvar.equals("S")) || (ehSalvar.equals("s"))) {
        Pessoa cliente = new Pessoa(visual.getCampoNome(),
        visual.getCampoCpf(),
        visual.getCampoEndereco(),
        visual.getCampoTipo());

        if (base.isEmpty()) {
            base.insertBegin(cliente);
            visual.mensagemDadosSalvos();
        } else {
            boolean podeSalvar = true;

            while (base.hasNext()) {
                if (base.next().getCpf().equals(cliente.getCpf())) {
                    podeSalvar = false;
                }
            }

            if (podeSalvar) {
                base.insertBegin(cliente);
            } else {
                visual.mensagemClienteCadastradoSistema();
            }
        }
    } else {
    visual.mensagemDadosNaoSalvos();
    }
}
```

Os outros métodos de criação e exclusão das classes de controle e visual das outras classes modelo seguem o mesmo principio seguindo o menu e executando nas extremidades dos menus os métodos funcionais identificados pelo menu. Por isso vou começar a explicar o pedido. O menu principal do controle do pedido segue o mesmo padrão por isso vou adentrando dentro dos métodos.

O método `novoPedido()` é o método para criar um novo pedido, nele começamos digitando o CPF do garçom que é um funcionário e um cliente, logo apos uma mesa. Ai verifica se quer cancelar o pedido e se nao ele abre a tela de pedido cadastrado com o método `pedidoCadastrado()` já alterando o status do pedido para cadastrado que antes estava pendente.

```
private void novoPedido() {
    pedido = new Pedido();
    Pessoa garcom;
    Pessoa cliente;
    Mesa mesa;

    boolean status = true;
    boolean ehSair = false;

    if (base.isEmpty()) {
        pedido.setCodigo(1);
    } else {
        pedido.setCodigo(base.size() + 1);
    }

    while (!ehSair) {
        do {
            visual.inicioPedido(pedido.getCodigo());
                        Entrada e = Entrada.getInstance();
            String opcao = e.retornaDado("CANCELAR PEDIDO [S / n]: ");
            if (opcao.equals("S") || opcao.equals("s")) {
                ehSair = true;
            } else {
                garcom = new Pessoa(visual.getCampoCpfGarcom());
                cliente = new Pessoa(visual.getCampoCpfCliente());

                Integer numeroMesa;
```

```
                try {
                    numeroMesa = Integer
                            .parseInt(visual.getCampoNumeroMesa());
                } catch (Exception e) {
                    numeroMesa = 0;
                }

                mesa = (new ControleMesa())
                    .buscarMesaPorIndex(numeroMesa - 1);

                garcom = BaseDados.getInstance()
                    .getBaseFuncionario().findItem(garcom);

                cliente = BaseDados.getInstance()
                    .getBasePessoa().findItem(cliente);

                if (mesa == null) {
                    visual.mensagemMesaNaoEncontrada();
                    status = status && false;
                }

                if (cliente == null) {
                    visual.mensagemPessoaNaoEncontrada("CLIENTE");
                    status = status && false;
                }

                if (garcom == null) {
                    status = status && false;
                    visual.mensagemPessoaNaoEncontrada("GARCOM");
                }

                if (status) {
                    pedido.setCliente(cliente);
                    pedido.setGarcom(garcom);
                    pedido.setMesa(mesa);
                    pedido.setStatus(EStatusPedido.CADASTRANDO);
                    base.insertBegin(pedido);
                    ehSair = true;
                    pedidoCadastrado(pedido);
                }
            }
        } while (!ehSair);
    }
}
```

Ao abrir a tela de produto cadastrado ele abre de novo um método com o padrão de menu utilizando um case para selecionar entre as opções, o menu tem opções recorrentes por exemplo adicionar produto ao pedido e ver produtos que funcionam da mesa forma de cadastro setada anteriormente porem um um atributo do pedido que é uma lista de produtos. porem o método mais importante é o método `enviarProducao()` que é a opção quatro. primeiro ele verifica se o pedido contem produtos, se tiver ele envia para produção e fica com status `emproducao`. Ali ele abre outro menu que é quando o pedido esta em produção e pode ser visto os produtos, entregue ou cancelado. Segue o trecho que verifica se os produtos estão vazios, segue tambem o método de entregar o pedido `entregarPedido()` e o método de cancelar o pedido `cancelarPedido()`.

```
case "4": {
    if (pedido.getProdutos().isEmpty()) {
        visual.mensagemNaoHaProdutoNoPedido();
    } else {
        enviarProducao();
        ehSair = true;
    }
    break;
}
```

Método de cancelar pedido que so pode ser cancelado caso esteja em produção, se for antes disso ele pode ser cancelado ou excluído.

```
private void cancelarPedido() {
    visual.cancelarProduto();

    String opcao = Entrada.getInstance()
        .retornaDado("DESEJA CANCELAR O PEDIDO [S / n]: ");

    if (opcao.equals("S") || opcao.equals("s")) {
        base.deleteByItem(pedido);
        pedido = null;
        visual.mensagemPedidoCancelado();
    } else {
        visual.mensagemPedidoNaoCancelado();
    }
}
```

Método de entregar pedido. Apos esse método o pedido fica reconhecido no caixa do menu inicial do sistema, assim ele fica visível e fica contável nos menus específicos de valores que serão apresentados posteriormente.

```
private void entregarPedido() {
    Visual.getInstance()
        .visualizarCabecalho("SISTEMA RESTAURANTE", "ATENCAO");

    String opcao = Entrada.getInstance()
        .retornaDado("ENTREGAR PEDIDO [S / n]: ");

    if (opcao.equals("S") || opcao.equals("s")) {
        pedido.setStatus(EStatusPedido.ENTREGUE);
        visual.mensagemPedidoEntregue();
    } else {
        visual.mensagemPedidoNaoEntregue();
    }
}
```

No menu de caixa tem três métodos específicos, um que mostra valores com pedidos, outro mostra os valores por cliente e outro mostra o valor total recebido, porem todos mostram o valor total no final do relatório. Valor arrecadado com pedidos:

```
private void valorArrecadadoComPedidos() {
    visual.inicioValorArrecadado("VALOR ARRECADADO COM PEDIDOS");
    Double somaTotal = 0.0; Double soma = 0.0;
    while (pedidos.hasNext()) {
        Pedido pedido = pedidos.next(); soma = 0.0;
        if (pedido.getStatus() == EStatusPedido.ENTREGUE) {
            while (pedido.getProdutos().hasNext()) {
                Produto produto = pedido.getProdutos().next();
                soma += produto.getValor();
            }
            visual.mostrarPedidoPorColuna(pedido
                .getCodigo(), pedido.getCliente().getNome(), soma);
            somaTotal += soma;
        }
    }
    visual.finalValorArrecadadoComPedidos(somaTotal);
    Entrada.getInstance()
        .retornaDado("PRESSIONE ENTER PARA CONTINUAR ...");
}
```

Valor arrecadado com pedidos por cliente

```
private void valorArrecadadoPorCliente() {
    visual.inicioValorArrecadado("VALOR ARRECADADO POR CLIENTE");
    Double somaTotal = 0.0;
    Double soma = 0.0;
    Lista<Pessoa> clientes = new Lista<Pessoa>();
    while (pedidos.hasNext()) {
        Pedido pedido = pedidos.next();

        if (clientes.findItem(pedido.getCliente()) == null) {
            if (pedido.getStatus() == EStatusPedido.ENTREGUE) {
                clientes.insertBegin(pedido.getCliente());
            }
        }
    }

    while (clientes.hasNext()) {
        Pessoa cliente = clientes.next();
        soma = 0.0;
        while (pedidos.hasNext()) {
            Pedido pedido = pedidos.next();

            if (cliente.equals(pedido.getCliente())) {
                while(pedido.getProdutos().hasNext()) {
                    if (pedido.getStatus() == EStatusPedido.ENTREGUE) {
                        soma += pedido.getProdutos().next().getValor();
                    }
                }
            }
        }
        visual
            .mostrarClienteEValor(cliente.getNome(), cliente.getCpf(),
soma);

        somaTotal += soma;
    }

    visual.finalValorArrecadadoComCliente(somaTotal);
    Entrada.getInstance()
        .retornaDado("PRESSIONE ENTER PARA CONTINUAR ...");
}
```

Valor total recebido

```
private void valorTotalRecebido() {
    Double somaTotal = 0.0;
    Double soma = 0.0;
    while (pedidos.hasNext()) {
        Pedido pedido = pedidos.next();
        soma = 0.0;
        if (pedido.getStatus() == EStatusPedido.ENTREGUE) {
            while (pedido.getProdutos().hasNext()) {
                Produto produto = pedido.getProdutos().next();
                soma += produto.getValor();
            }

            somaTotal += soma;
        }
    }
    visual.valorTotalArrecadado(somaTotal);
    Entrada.getInstance()
        .retornaDado("PRESSIONE ENTER PARA CONTINUAR ...");
}
```

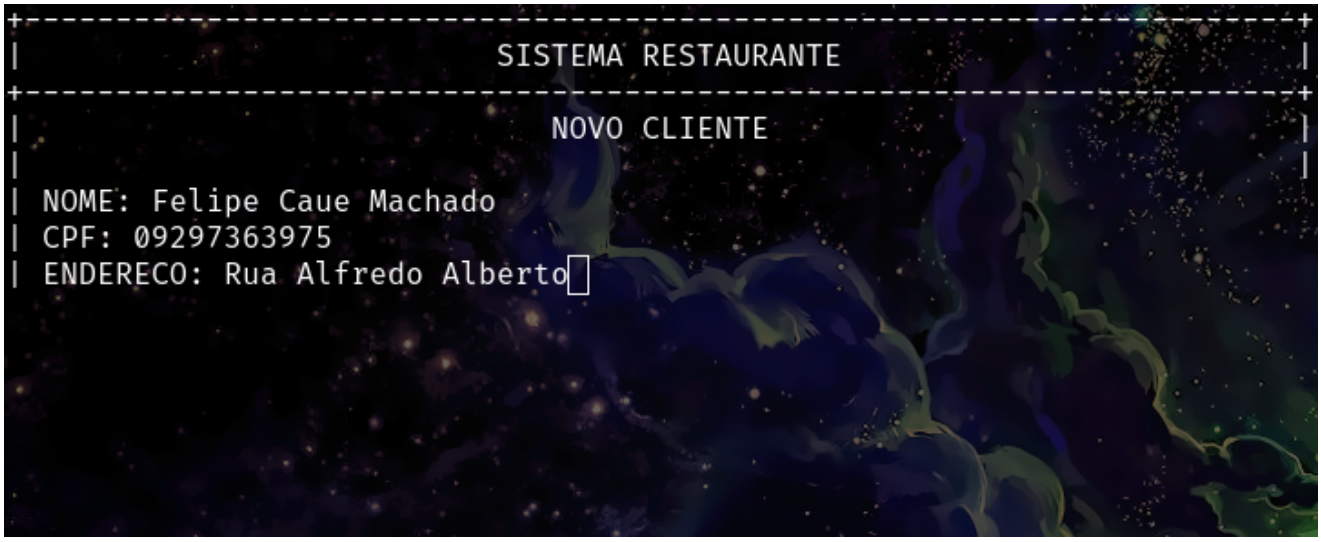
Testes

Menu Principal

Opcao 1 do menu principal, cadastros de cliente funcionario mesas e produto.

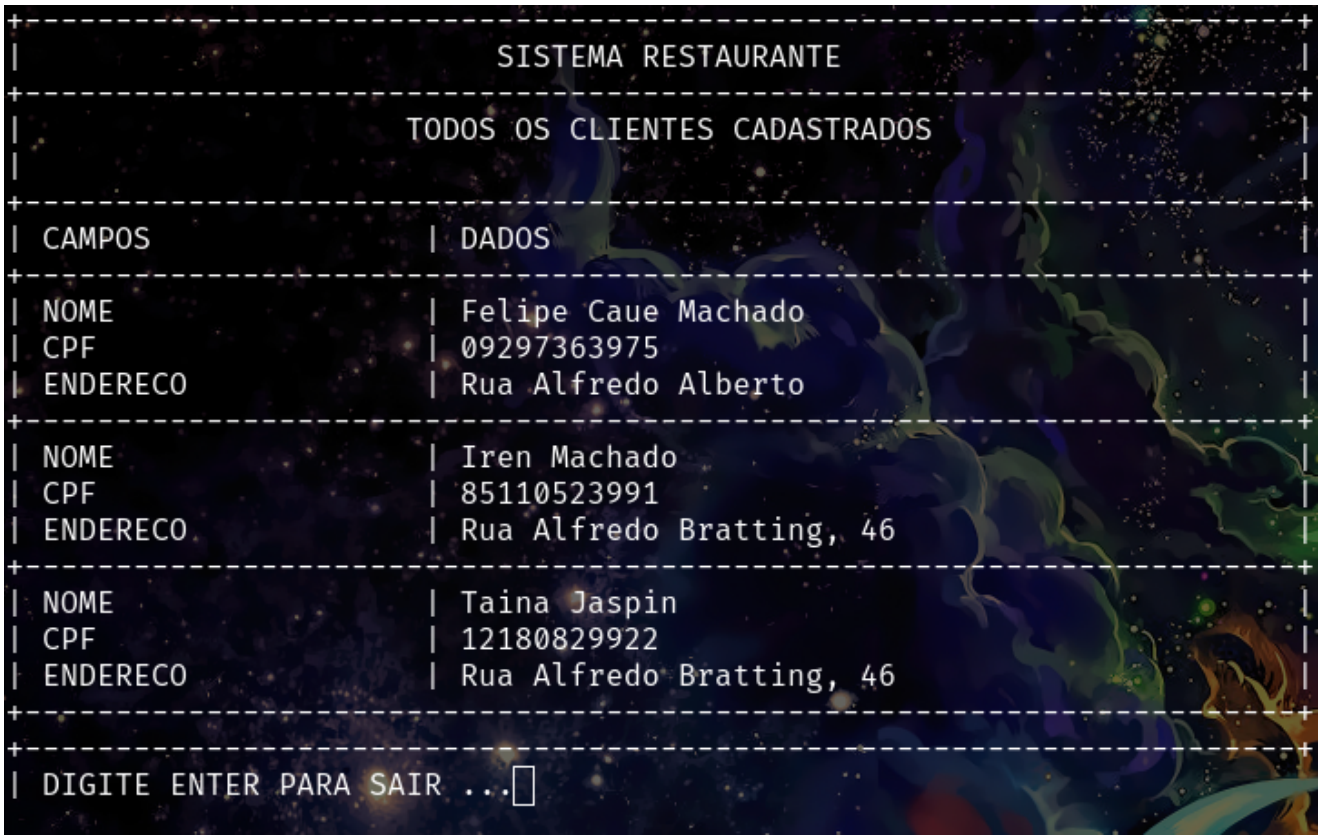
Opcao 1 do menu de cadastros faz o gerenciamento de clientes

Ao selecionar a opção 1 para novo posso digitar os dados um por um como a seguir e logo depois de clicar enter no endereço abre um menu de verificação como no print abaixo ao cadastro e pergunta se quero ou nao salvar o cliente.



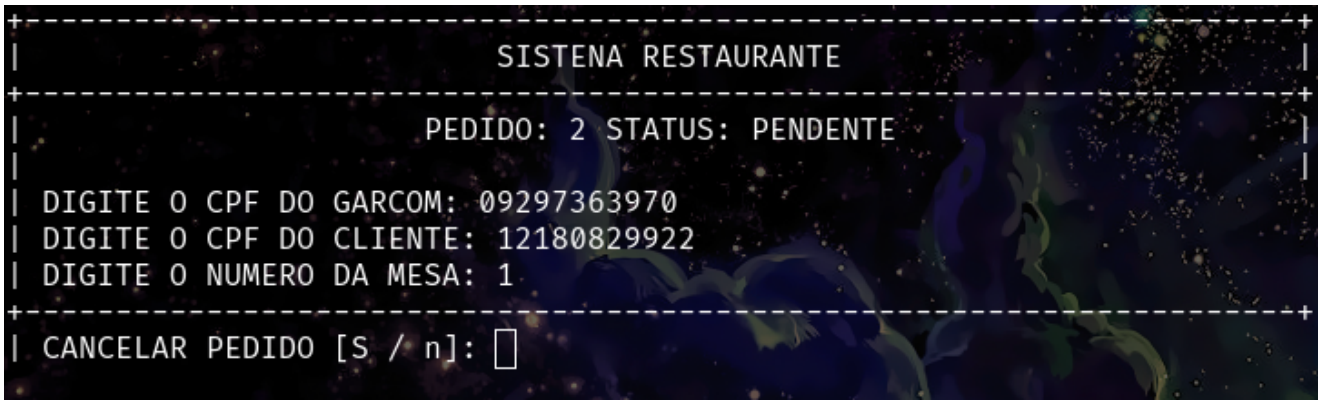
Na opcao de gerenciamento tambem posso buscar um cliente por codigo e todos.

Buscando por codigo aparece a mesma tela anterior de verificacao de cliente porem se clicar em todos aparece uma lista com todos os clientes e seus dados como a seguir:



Os outros cadastros funcionam da mesa forma por isso vou direto para o Pedido, porem caso queira pode rodar o jar que o aplicativo tem todos os cadastros funcionando

Ao clicar em novo pedido aparece uma tela para digitar como:

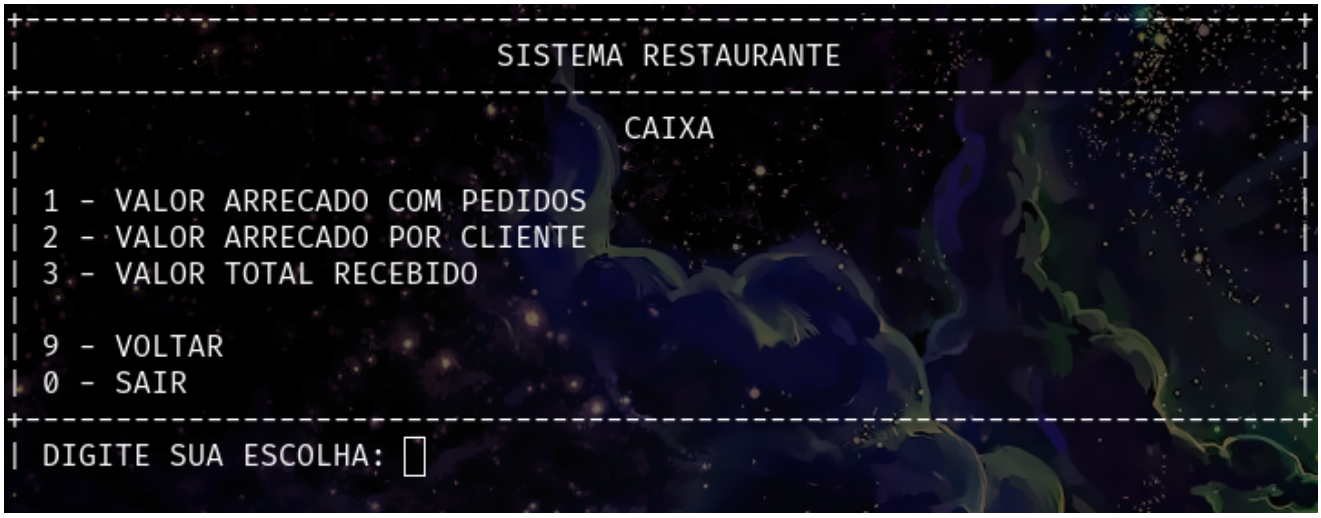


Se eu escolher nao então abre a tela de pedido cadastrado para adicionar produtos e enviar para produção:

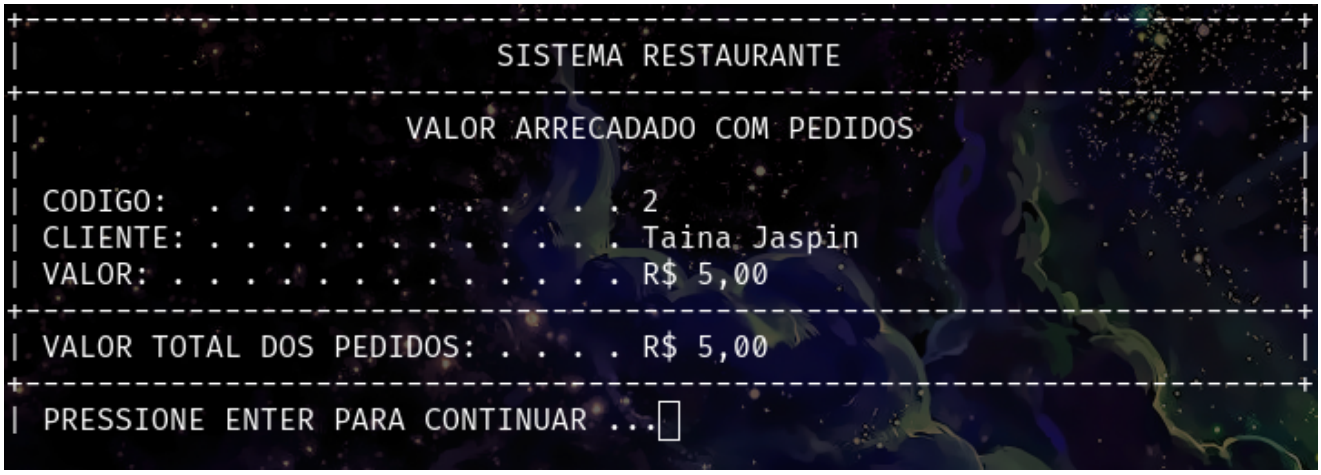
Ao enviar para produção com pedidos essa tela aparece:

Se entregar o pedido fica finalizado e passível de ver os produtos e tambem passível de ver os valores no men de caixa no menu principal:

O menu de caixa é assim



Valor arrecadado com pedidos mostra:



Valor arrecadado por cliente:

Valor total recebido