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();
}
```

Os outros métodos da lista estão sendo escritos igualmente ao apresentado em sala de aula por isso deixei somente no código fonte.

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 Produto porém segue o mesmo padrão para as outras classes

Na Classe Produto que é o modelo, temos três atributos e dois construtores::

```
public class Produto {
    // ATRIBUTOS
    private Integer codigo;
    private String descricao;
    private Double valor;

    // CONSTRUTORES
    public Produto(String descricao, Double valor) {
        this.descricao = descricao;
        this.valor = valor;
    }

    public Produto(Integer codigo, String descricao, Double valor) {
        this.codigo = codigo;
        this.descricao = descricao;
        this.valor = valor;
    }

    public Produto() {

    }

    // GETTERS AND SETTERS
}
```

Depois do modelo temos o Controle (ControlleProduto) que faz a ponte entre o visual (VisualProduto) e o modelo (Produto). A Classe controle produto é a classe que faz todos os métodos funcionais do produto.

O método principal dessa classe é o menuProduto onde terá a regra de criação do menu de gerenciamento do produto.

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

    while(!ehSair) {
        visual.menuGerenciamento();
        String texto = "DIGITE SUA OPCAO: "
        String opcao = Entrada.getInstance().retornaDado(texto);

        switch (opcao) {
        case "1": {
            menuNovo();
            break;
        }
        case "2": {
            excluirProduto();
            break;
        }
        case "3": {
            buscarProdutos();
            break;
        }
        case "9": {
            ehSair = true;
            break;
        }
        case "0": {
            System.exit(1);
        }
        default:
            Visual.getInstance().visualizarMensagemOpcaoInvalida();
        }
    }
}
```