



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Graduação em Sistemas de Informação

Bryan Diniz Rodrigues
Luiz Henrique Gomes Guimarães
Maria Luiza Moura Rocha
Thais Barcelos Lorentz

**RELATÓRIO DO LABORATÓRIO 2 DE PROGRAMAÇÃO ORIENTADA POR
OBJETOS**

Belo Horizonte
2019/2

Bryan Diniz Rodrigues
Luiz Henrique Gomes Guimarães
Maria Luiza Moura Rocha
Thais Barcelos Lorentz

TRABALHO FINAL INTERDISCIPLINAR DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Trabalho apresentado à disciplina Programação
Orientada a Objetos, do curso de Sistemas de
Informação da Pontifícia Universidade Católica de
Minas Gerais.

Orientador: Prof. Paulo Cesar do Amaral Pereira

Belo Horizonte
2019

Sumário

RESUMO	3
1 INTRODUÇÃO	4
1.3 UTILIZAÇÃO DO PROGRAMA	5
1.3.1 TELA INICIAL	5
1.3.2 DASHBOARD	5
1.3.3 ESTOQUE	6
1.3.4 VENDAS	7
1.3.5 FORNECEDORES	8
LABEL	9
LISTVIEW	9
GERADOR DE CÓDIGO DE BARRAS	10
EAN13	10
2.2.1.3CLASSE ARQUIVO FINANCEIRO	14
2.2.1.4 CLASSE ARQUIVO FORNECEDORES	15
2.2.1.5 CLASSE ARQUIVO VENDA	16
2.2.1.6 CLASSE GERAR ARQUIVOS	18
2.2.1.7 CLASSE RELATÓRIO VENDAS	19
2.2.2 PASTA PRODUTOS	20
2.2.2.1 INTERFACE PRODUTO	20
2.2.2.2 CLASSES HERDEIRAS DA INTERFACE PRODUTO	20

RESUMO

Este projeto se refere a um sistema para um mini comércio eletrônico que atende micro e pequenos empreendedores e seu objetivo é gerenciar produtos de estoque de um supermercado.

Para a realização deste projeto, utilizamos diversos recursos práticos e teóricos desenvolvidos em salas de aula e em laboratório de programação orientada a objeto, além de pesquisas bibliográficas, consultas à fóruns e diversos sites relacionados ao tema.

A proposta é desenvolver um sistema mais realista possível baseado nas pesquisas realizadas pelo grupo e técnicas de programação orientado a objeto que tem como objetivo a utilização dos conceitos aprendidos em sala de aula, como herança, polimorfismo, coesão, modularização, entre outros e aplicá-los na solução de um problema, neste caso, desenvolver um sistema de controle de veículos, motoristas e multas.

Após a finalização deste projeto, percebemos a importância e a melhoria que tivemos na criação de programas orientados a objeto, organizando melhor os códigos produzidos, facilitando manutenções e implementações futuras, além disso, no progresso que tivemos na administração e comunicação entre os membros do grupo.

Palavras-chave: Orientação a Objetos. Desenvolver um sistema. Aplicação prática da programação orientada a objeto.

1 INTRODUÇÃO

Este relatório trata-se do Trabalho Interdisciplinar(TFI) da disciplina de Programação Orientada por Objetos e tem como objetivo desenvolver o conhecimento obtido nas aulas de Laboratórios e Teóricas, do segundo período do curso de Sistemas de Informação.

Constam neste relatório o código desenvolvido para a criação do projeto proposto, além de uma breve explicação de como ele foi desenvolvido.

1.1 OBJETIVOS

Este trabalho tem como objetivos:

- Desenvolver o conhecimento obtido nas aulas de POO no segundo período do curso de Sistemas de Informação.
- Aprendizagem de novos termos muito utilizados no mercado de trabalho em nossa área.
- Aperfeiçoamento do uso de diferentes classes e objetos.
- Aprendizagem de Formulários, que são usados para o desenvolvimento de aplicativos usando a interface gráfica do Microsoft Windows.

1.2 ORGANIZAÇÃO DO TRABALHO

O principal objetivo do trabalho foi desenvolver um programa de Gestão de vendas, estoque e compra. Diante deste tema, foi proposto pelo professor Paulo Amaral a elaboração de um programa, onde um pequeno comércio quer automatizar sua gestão de vendas, estoque e compras de produtos. Produtos devem gerar compras de novas remessas quando o estoque atingir determinado nível. Produtos geram diferentes códigos de barras de acordo com sua categoria, no momento da venda. Dashboard, onde mostra as estatísticas das vendas e o saldo do estoque. Dentre outras coisas que incrementamos no projeto.

1.3 UTILIZAÇÃO DO PROGRAMA

1.3.1 TELA INICIAL



A tela inicial do programa não apresenta nenhuma função a não ser estética, e ao lado esquerdo está presente 4 botões para as principais funções do programa.

1.3.2 DASHBOARD



O Dashboard é onde será encontrado algumas informações sobre a situação do comércio, como o saldo real e o saldo em valor de estoque, assim como gráficos de vendas.

1.3.3 ESTOQUE

Dashboard

Estoque



Vendas

Fornecedores


TRABALHO FINAL INTERDISCIPLINAR POO

ESTOQUE

Código de barras	Categoria	Nome do produto	Preço Unidade/Kg	Quantidade
789123400001	Alimentos	Bolo	R\$ 10,00	97
789123400003	Alimentos	Pão de forma	R\$ 2,85	76
789123400004	Alimentos	Miojo	R\$ 1,00	67
789123400005	Alimentos	Feijão	R\$ 7,20	11
789123400006	Limpeza	Lava-louças Ypê 500ml	R\$ 1,70	19
789123400007	Higiene pessoal	Colgate Total 12	R\$ 8,28	33
789123400008	Outros	Kit Ferramentas Bosch	R\$ 55,00	2



COLGATE TOTAL 12 PROFESSIONAL WHITENING



Valor total em estoque: R\$ 273,57 Data de cadastro: 16/11/2019 08:55:36

Atualizar

Adicionar

Editar

Excluir

Bryan Diniz - Luiz Henrique - Maria Luiza - Thais Lorentz

O Estoque exibe uma lista em forma de tabela dos produtos cadastrados, quando um produto é selecionado é mostrado algumas informações sobre, como o nome, data de cadastro, o valor total que o produto representa no estoque, a imagem do produto, seu código de barras, e uma imagem que representa sua categoria.

Também é possível adicionar um novo produto, ou editar um já existente, e até excluir caso seja necessário.

Para adicionar um novo produto, basta clicar no botão adicionar que será exibido um nova tela, em que será necessário preencher os seguintes campos.

Nome do produto:

Código do produto:

Categoria do produto:

Quantidade do produto:

Preço da unidade/Kg:

Descrição do produto:

Adicionar produto:

Coca Cola 2 litros


789123400009

Alimentos

14

4,42

Refrigerante Coca Cola 2 litros



Carregar imagem

Cancelar

Adicionar

Para editar será exibido uma janela semelhante, mas só com os campos já preenchidos prontos para serem editados.

Ao clicar no botão excluir uma mensagem de aviso é exibida, requerendo a confirmação para a exclusão do produto do sistema.

1.3.4 VENDAS

TRABALHO FINAL INTERDISCIPLINAR POO

VENDA

Feijão

Impostos: 25,00%

Preço: R\$ 7,20

Nome do produto	Preço da unidade	Quantidade
Feijão	R\$ 7,20	1

789123400005 Adicionar Valor Total: R\$ 9,00

Limpar Confirmar

Bryan Diniz - Luiz Henrique - Maria Luiza - Thais Lorentz

Na aba de Vendas existe um campo de texto onde é inserido o código de barras e o produto será exibido semelhante a janela anterior, no entanto com imposto sendo aplicado de acordo com a categoria do produto. Quando um produto tem seu código lido, ele é adicionado na lista e o valor total da compra é somado. Antes de finalizar uma compra temos duas opções, limpar a lista de compras ou confirmar a compra. Ao confirmar é apresentada opção de gerar um cupom fiscal os os dados da compra no seguinte formato.

```
----- PUC MARKET -----  
Bolo - Quantidade: 2 - valor da unidade: R$ 10,00  
Miojo - Quantidade: 1 - valor da unidade: R$ 1,00  
Pão de forma - Quantidade: 2 - valor da unidade: R$ 2,85  
Colgate Total 12 - Quantidade: 1 - valor da unidade: R$ 8,29  
Feijão - Quantidade: 2 - valor da unidade: R$ 7,20  
Coca Cola 2 litros - Quantidade: 1 - valor da unidade: R$ 4,42  
  
valor total da compra: R$ 67,43  
-----
```


1.3.5 FORNECEDORES

TRABALHO FINAL INTERDISCIPLINAR POO

FORNECEDORES

Nome do fornecedor	CNPJ	Estado
Sadia S.a	20.730.099/0001-94	MG
Seara Alimentos Ltda	02.914.460/0001-50	MG
Perdigão S.a	86.547.619/0127-38	MG
Coca Cola Ltda	45.997.418/0001-53	AM

SADIA S.A

Sadia

☎ (00) 0 0000-0000
✉ sadia@gmail.com

Atualizar

Adicionar
Editar
Remover

Bryan Diniz - Luiz Henrique - Maria Luiza - Thais Lorentz

O Estoque funciona de forma semelhante ao estoque, onde também está presente uma tabela com algumas informações sobre o fornecedor, como nome, cnpj e o estado, ao lado mais alguns dados são mostrados, como o telefone e o e-mail da empresa, além de sua logo e sua respectiva categoria.

Também é possível adicionar, editar e remover um fornecedor, ao clicar no botão adicionar é exibido a seguinte janela, onde preenchemos os campos com os dados do respectivo fornecedor.

Editar fornecedores:

Nome do fornecedor: Sadia S.a Categoria ▼

CNPJ do fornecedor: 20,730,099/0001-94

Endereço do fornecedor: UF: AC CEP: 00000-000

Telefone do fornecedor: (00) 0 0000-0000

E-Mail do fornecedor: sadia@gmail.com

Carregar

Cancelar Editar

Para editarmos é da mesma forma do Estoque, onde os campos já vêm preenchidos para a edição.

2. DESENVOLVIMENTO

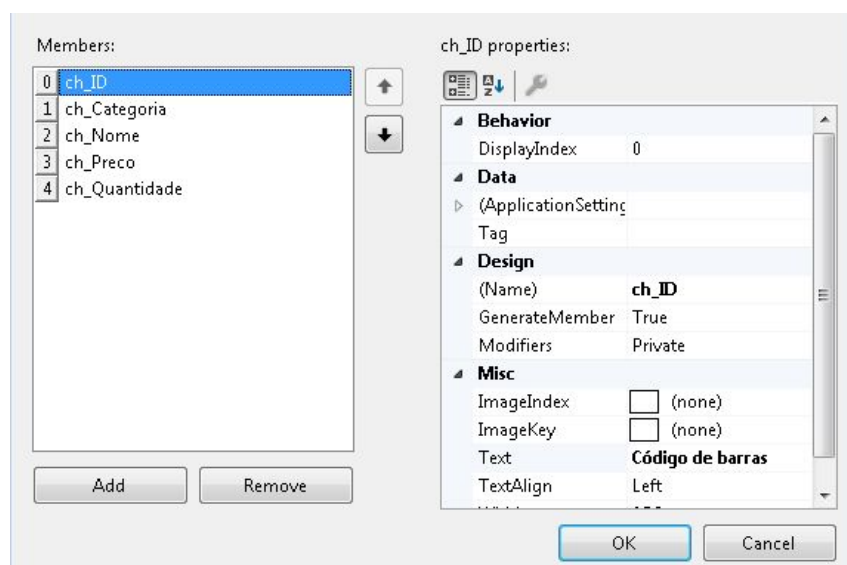
LABEL

Uma Label participa da ordem de tabulação de um formulário, mas não recebe o foco (o próximo controle na ordem de Tabulação recebe o foco). Por exemplo, se a propriedade UseMnemonic for definida como true e um caractere mnemônico-o primeiro caractere após um e comercial (&)-for especificado na propriedade Text do controle, quando um usuário pressionar ALT + a tecla mnemônico, o foco será movido para o próximo controle no ordem de tabulação. Esse recurso fornece navegação de teclado para um formulário. Além de exibir texto, o controle de Label também pode exibir uma imagem usando a propriedade Image ou uma combinação das propriedades ImageIndex e ImageList.

LISTVIEW

A ListView foi utilizada em diversas partes programa para listar desde o estoque a lista de compras de um cliente.

A implementação do ListView consiste em separar por colunas os atributos dos objetos que desejamos exibir na tabela, como podemos ver abaixo:



Já o código para adicionar um objeto na lista pode ser escrito da seguinte forma:

```

foreach (var produto in ArquivoEstoque.ListaProdutos)
{
    string id = produto.ID;
    string categoria = produto.Categoria;
    string nome = produto.Nome;
    string preco = "R$ " + produto.Preco.ToString("F2");
    string quantidade = produto.Quantidade.ToString();
    string dataDeCadastro = produto.DataCadastro.ToString();
    string descricao = produto.Descricao;

    string[] row = { id, categoria, nome, preco, quantidade };
    var listViewItem = new ListViewItem(row);
    listView_estoque.Items.Add(listViewItem);
}

```

GERADOR DE CÓDIGO DE BARRAS

O código de barras é gerado a partir de um dll desenvolvido pelo C# Ui Academy onde é possível gerar imagens bitmap que serão colocadas em PictureBox, é possível gerar diversos formatos de códigos incluindo o utilizado EAN13.

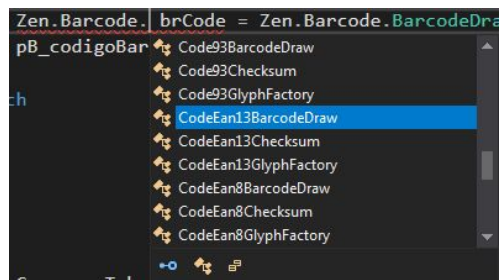
EAN13



É um formato internacional utilizada por diversos comércios, seu código é constituído por uma sequência de 12 números +1 verificador, sendo seus 3 primeiros números o código do país e de 4 a 6 o código da empresa e o restante o do produto.

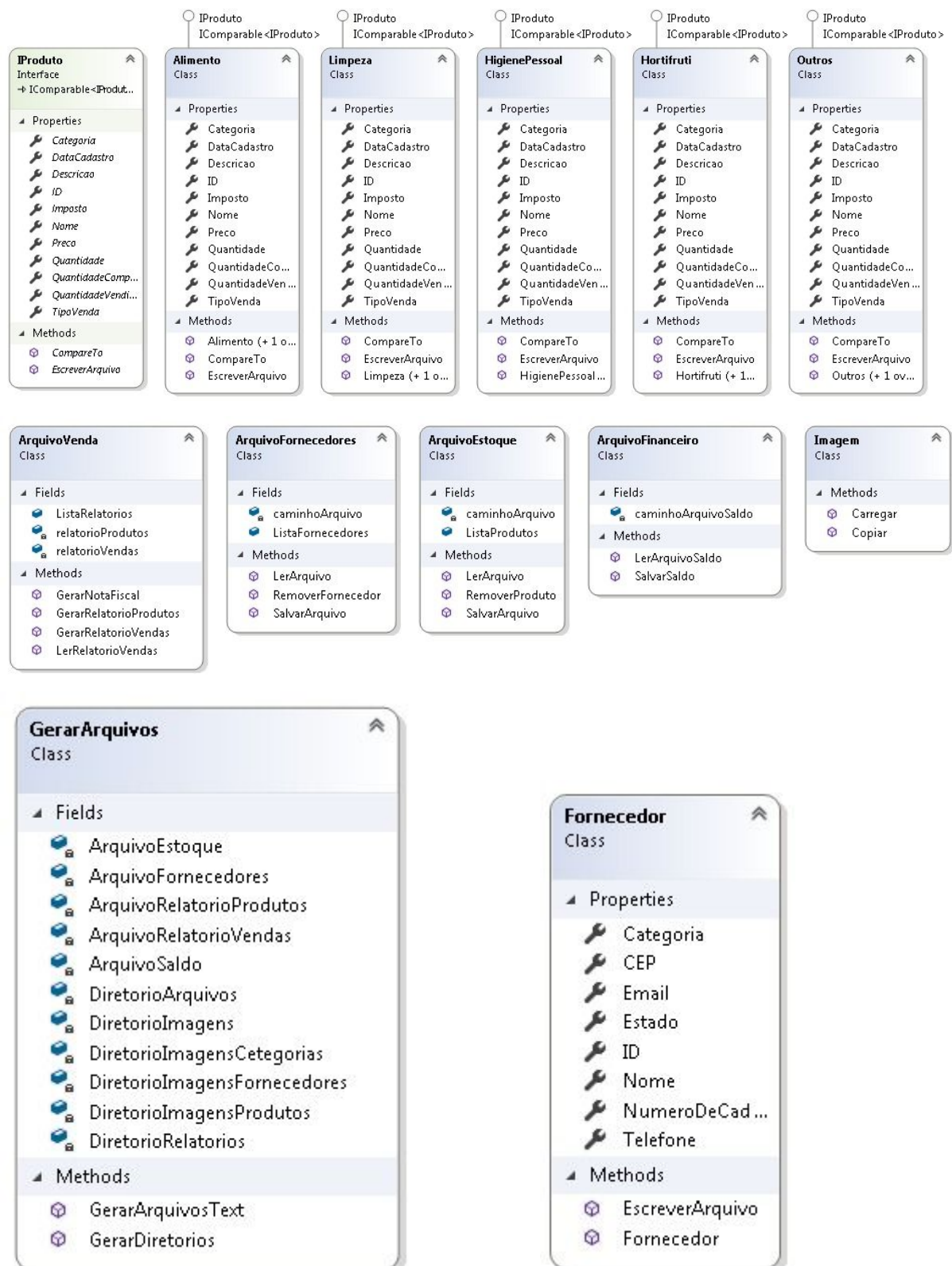
A utilização da biblioteca é simples, e é representado pelas linhas de código abaixo.

```
Zen.Barcode.CodeEan13BarcodeDraw brCode = Zen.Barcode.BarcodeDrawFactory.CodeEan13WithChecksum;  
pB_codigoBarras.Image = brCode.Draw(ArquivoEstoque.ListaProdutos[index].ID, 60, 20);
```



E como podemos ver aceita diversos outros formatos de códigos de barra.

2.1 UML



2.2 CÓDIGOS

2.2.1 PASTA ARQUIVOS

2.2.1.1 CLASSE IMAGEM

```
7 public class Imagem
8 {
9     // carrega uma imagem para a memória
10    public static Bitmap Carregar(string caminho)
11    {
12        if (File.Exists(caminho))
13        {
14            byte[] imageByte = File.ReadAllBytes(caminho);
15            MemoryStream mStream = new MemoryStream();
16            mStream.Write(imageByte, 0, Convert.ToInt32(imageByte.Length));
17            Bitmap bm = new Bitmap(mStream, false);
18            mStream.Dispose();
19            return bm;
20        }
21        else
22        {
23            Bitmap bmp = new Bitmap(100, 100);
24            using (Graphics graph = Graphics.FromImage(bmp))
25            {
26                Rectangle ImageSize = new Rectangle(0, 0, 100, 100);
27                graph.FillRectangle(Brushes.White, ImageSize);
28            }
29            return bmp;
30        }
31    }
32
33    public static void Copiar(string entradaImg, string saidaImg)
34    {
35        FileInfo fi = new FileInfo(entradaImg);
36        Stream entrada = File.Open(entradaImg, FileMode.Open);
37        Stream saida = File.Open(saidaImg, FileMode.Create);
38
39        for (int i = 0; i < fi.Length; i++)
40        {
41            int b = entrada.ReadByte();
42        }
43    }
44 }
```

A função desta classe, basicamente é carregar uma foto selecionada em uma determinada pasta, para a memória do programa, acabando com a necessidade de ter a foto armazenada em um local físico, especificada por um certo caminho, e reduzindo o trabalho ao copiar a foto para exibi-la em várias partes do programa.

2.2.1.2 CLASSE ARQUIVOESTOQUE


```

public static void LerArquivo()
{
    ListaProdutos.Clear();

    try
    {
        if (File.Exists(caminhoArquivo))
        {
            using (StreamReader sr = new StreamReader(caminhoArquivo))
            {
                while (!sr.EndOfStream)
                {
                    string linha = sr.ReadLine();
                    //ID;Categoria;Nome;Preço Unidade;Quantidade;Quantidade vendida;Data de cadastro;Descrição

                    string[] aux = linha.Split(';');
                    string id = aux[0];
                    string categoria = aux[1];
                    string nome = aux[2];
                    double preco = double.Parse(aux[3]);
                    double quantidade = double.Parse(aux[4]);
                    double quantidadeVendida = double.Parse(aux[5]);
                    DateTime dataDeCadastro = DateTime.Parse(aux[6]);
                    string descricao = aux[7];

                    if (categoria == "Alimentos")
                    {
                        Alimento produto = new Alimento(id, nome, preco, (int)quantidade, (int)quantidadeVendida, dataDeCadastro, descricao);

```

O primeiro método encontrado na classe ArquivoEstoque, ela começa verificando se o arquivo existe, através da variável “caminhoarquivo”, que é uma string global que recebe como valor, o caminho do arquivo. Após isso, o mesmo método pega cada valor separado por “;” e atribui a variáveis separadas, facilitando o instanciamento exclusivo para cada tipo específico de produto e adicionando na lista:

```

        if (categoria == "Alimentos")
        {
            Alimento produto = new Alimento(id, nome, preco, (int)quantidade, (int)quantidadeVendida, dataDeCadastro, descricao);
            ListaProdutos.Add(produto);
        }
        else if (categoria == "Limpeza")
        {
            Limpeza produto = new Limpeza(id, nome, preco, (int)quantidade, (int)quantidadeVendida, dataDeCadastro, descricao);
            ListaProdutos.Add(produto);
        }
        else if (categoria == "Higiene pessoal")
        {
            HigienePessoal produto = new HigienePessoal(id, nome, preco, (int)quantidade, (int)quantidadeVendida, dataDeCadastro, descricao);
            ListaProdutos.Add(produto);
        }
        else if (categoria == "Hortifruti")
        {
            Hortifruti produto = new Hortifruti(id, nome, preco, quantidade, quantidadeVendida, dataDeCadastro, descricao);
            ListaProdutos.Add(produto);
        }
        else if (categoria == "Outros")
        {
            Outros produto = new Outros(id, nome, preco, (int)quantidade, (int)quantidadeVendida, dataDeCadastro, descricao);
            ListaProdutos.Add(produto);
        }
    }
}

```

O segundo e terceiro método, são responsáveis pela manipulação de dados no arquivo, incluindo adicionar o conteúdo da lista em um arquivo, através do Foreach, e

excluir um produto específico da lista, recebendo por parâmetro a posição do produto e excluindo através da função RemoveAt:

```
public static void SalvarArquivo()
{
    using (StreamWriter sr = new StreamWriter(caminhoArquivo))
    {
        foreach (var produto in ListaProdutos)
        {
            sr.WriteLine(produto.EscreverArquivo());
        }
    }
}

public static void RemoverProduto(int index)
{
    File.Delete(@"Arquivos\Imagens\Estoque\" + ListaProdutos[index].ID + ".png");
    ListaProdutos.RemoveAt(index);
}
```

2.2.1.3CLASSE ARQUIVO FINANCEIRO

A classe se inicia com o método que verifica a existência do arquivo, lendo como caminho o valor da string global caminhoArquivoSaldo. Após isso criando a variável auxiliar e o separador usando “;”, ele percorre o arquivo e salva os valores lidos na variável do tipo double = saldo. No final do método, ele retorna essa variável:


```

public static double LerArquivoSaldo()
{
    double saldo = 0;
    try
    {
        if (File.Exists(caminhoArquivoSaldo))
        {
            using (StreamReader sr = new StreamReader(caminhoArquivoSaldo))
            {
                string linha = sr.ReadLine();
                string[] aux = linha.Split(';');

                for (int i = 0; i < aux.Length; i++)
                {
                    saldo += double.Parse(aux[i]);
                }
            }
        }
        else
        {
            string message = "Arquivo do saldo não encontrado";
            string caption = "Erro";
            MessageBoxButtons buttons = MessageBoxButtons.OK;
            DialogResult result;

            result = MessageBox.Show(message, caption, buttons, MessageBoxIcon.Error);
        }
    }
    catch
    {
        string message = "Erro ao carregar saldo";
        string caption = "Erro";
        MessageBoxButtons buttons = MessageBoxButtons.OK;
    }
}

```

Na sequência, temos o método SalvarSaldo que recebe por parâmetro o valor do saldo e armazena no arquivo, adicionando “;” para manter o separador:

```

public static void SalvarSaldo(double saldo)
{
    using (StreamWriter sr = File.AppendText(caminhoArquivoSaldo))
    {
        sr.Write(";"+saldo);
    }
}

```

2.2.1.4 CLASSE ARQUIVO FORNECEDORES

Verifica a existência do arquivo, e logo em seguida, armazena as informações do arquivo nas variáveis dos tipos correspondentes, para posteriormente criar um objeto passando os devidos parâmetros do construtor.

```

public static void LerArquivo()
{
    ListaFornecedores.Clear();

    if (File.Exists(caminhoArquivo))
    {
        using (StreamReader sr = new StreamReader(caminhoArquivo))
        {
            while (!sr.EndOfStream)
            {
                string linha = sr.ReadLine();
                //Numero de cadastro;Nome do Fornecedor;CNPJ;Estado;CEP;Telefone;Email

                string[] aux = linha.Split(';');

                int numeroDeCadastro = int.Parse(aux[0]);
                string nome = aux[1];
                string categoria = aux[2];
                string id = aux[3];
                string estado = aux[4];
                string cep = aux[5];
                string telefone = aux[6];
                string email = aux[7];

                string descricao = aux[6];

                Fornecedor fornecedor = new Fornecedor(numeroDeCadastro, nome, categoria, id, estado, cep, telefone, email);
                ListaFornecedores.Add(fornecedor);
            }
        }
    }
    else
    {
        string message = "Erro ao carregar arquivo";
    }
}

```

Logo em seguida, a classe contém os métodos responsáveis por Salvar no arquivo e deletar através das funções da Lista.

```

public static void SalvarArquivo()
{
    using (StreamWriter sr = new StreamWriter(caminhoArquivo))
    {
        foreach (var fornecedor in ListaFornecedores)
        {
            sr.WriteLine(fornecedor.EscreverArquivo());
        }
    }
}

public static void RemoverFornecedor(int index)
{
    File.Delete(@"Arquivos\Imagens\Fornecedores\" + ListaFornecedores[index].NumeroDeCadastro + ".png");
    ListaFornecedores.RemoveAt(index);
}

```

2.2.1.5 CLASSE ARQUIVO VENDA

No começo, ele cria uma lista do tipo RelatórioVendas e duas strings com os caminhos utilizados ao decorrer do programa.

No primeiro método, ele cria as variáveis necessárias para a manipulação do arquivo, logo em seguida, instância uma variável do tipo RelatorioVendas e adiciona como parâmetro para o construtor as variáveis contendo as informações devidas sobre a venda, e logo em seguida, adiciona na lista.

```
class ArquivoVenda
{
    public static List<RelatorioVendas> ListaRelatorios = new List<RelatorioVendas>();
    static string relatorioVendas = @"Arquivos\Relatorios\relatorioVendas.txt";
    static string relatorioProdutos = @"Arquivos\Relatorios\relatorioProdutos.txt";

    public static void LerRelatorioVendas()
    {
        if (File.Exists(relatorioVendas))
        {
            using (StreamReader sr = new StreamReader(relatorioVendas))
            {
                while (!sr.EndOfStream)
                {
                    string linha = sr.ReadLine();
                    string[] aux = linha.Split(';');
                    double valor = double.Parse(aux[0]);
                    DateTime data = DateTime.Parse(aux[1]);
                    RelatorioVendas relatorio = new RelatorioVendas(valor, data);
                    ListaRelatorios.Add(relatorio);
                }
            }
        }
    }
}
```

Em seguida, armazena os métodos para gerar o arquivo contendo as informações de produtos e vendas:

```

public static void GerarRelatorioProdutos(List<IProduto> lista)
{
    using (StreamWriter sw = new StreamWriter(relatorioProdutos))
        foreach (IProduto produto in lista)
        {
            if (produto.TipoVenda != "Quilo")
            {
                string codigo = produto.ID;
                string nome = produto.Nome;
                double quantidadeVendida = produto.QuantidadeComprada;
                string linha = codigo + ";" + nome + ";" + quantidadeVendida;
                sw.WriteLine();
            }
        }
}

public static void GerarRelatorioVendas(double valorTotal, DateTime dataDaVenda)
{
    using (StreamWriter sr = File.AppendText(relatorioVendas))
    {
        sr.WriteLine(valorTotal.ToString("F2") + ";" + dataDaVenda);
    }
}

```

Por fim, o método que gera o arquivo contendo a nota fiscal, que informa o nome do produto, a quantidade e o valor de cada unidade:

```

public static void GerarNotaFiscal(List<IProduto> lista, double valorTotal, string arquivoNota)
{
    using (StreamWriter sr = new StreamWriter(arquivoNota))
    {
        sr.WriteLine();
        sr.WriteLine("----- PUC MARKET -----");
        sr.WriteLine();
        foreach (IProduto produto in lista)
        {
            string linha = produto.Nome + " - Quantidade: " + produto.QuantidadeComprada + " - Valor da unidade: R$ " + produto.Preco.ToString("F2");
            sr.WriteLine(linha);
        }
        sr.WriteLine();
        sr.WriteLine("Valor total da compra: R$ " + valorTotal.ToString("F2"));
        sr.WriteLine();
        sr.WriteLine("-----");
    }
}

```

2.2.1.6 CLASSE GERAR ARQUIVOS

Contém os atributos com os valores que indicam o caminho de cada arquivo e verifica se o caminho existe, se o caminho não existe, ele cria o diretório através da função `CreateDirectory` da classe `Directory`.


```

class GerarArquivos
{
    static string ArquivoEstoque = @"Arquivos\estoque.txt";
    static string ArquivoSaldo = @"Arquivos\saldo.txt";
    static string ArquivoFornecedores = @"Arquivos\fornecedores.txt";
    static string ArquivoRelatorioVendas = @"Arquivos\Relatorios\relatorioVendas.txt";
    static string ArquivoRelatorioProdutos = @"Arquivos\Relatorios\relatorioProdutos.txt";

    static string DiretorioArquivos = @"Arquivos";
    static string DiretorioImagens = @"Arquivos\Imagens";
    static string DiretorioImagensProdutos = @"Arquivos\Imagens\Estoque";
    static string DiretorioImagensCategorias = @"Arquivos\Imagens\Categorias";
    static string DiretorioImagensFornecedores = @"Arquivos\Imagens\Fornecedores";
    static string DiretorioRelatorios = @"Arquivos\Relatorios";

    public static void GerarDiretorios()
    {
        if (!Directory.Exists(DiretorioArquivos))
        {
            Directory.CreateDirectory(DiretorioArquivos);
        }
        if (!Directory.Exists(DiretorioImagens))
        {
            Directory.CreateDirectory(DiretorioImagens);
        }
        if (!Directory.Exists(DiretorioImagensProdutos))
        {
            Directory.CreateDirectory(DiretorioImagensProdutos);
        }
        if (!Directory.Exists(DiretorioImagensCategorias))
        {
            Directory.CreateDirectory(DiretorioImagensCategorias);
        }
    }
}

```

2.2.1.7 CLASSE RELATÓRIO VENDAS

A classe RelatorioVendas, é composta apenas pelos seus atributos necessários para a criação de um método construtor, que será utilizado pela classe ArquivoVenda.

```

class RelatorioVendas
{
    public double Valor { get; private set; }
    public DateTime Data { get; set; }

    public RelatorioVendas(double valor, DateTime data)
    {
        Valor = valor;
        Data = data;
    }
}

```

2.2.2 PASTA PRODUTOS

2.2.2.1 INTERFACE PRODUTO

É uma classe que contém todos os atributos em comum utilizado em alimentos de várias categorias e o método EscreverArquivo, que será implementado nas classes que herdarão a interface.

```
interface IProduto : IComparable<IProduto>
{
    string ID { get; }
    string Categoria { get; }
    string Nome { get; }
    double Preco { get; }
    double Imposto { get; }
    string TipoVenda { get; }
    double Quantidade { get; set; }
    double QuantidadeComprada { get; set; }
    double QuantidadeVendida { get; set; }
    DateTime DataCadastro { get; }
    string Descricao { get; }

    string EscreverArquivo();

    int CompareTo(IProduto produto);
}
```

2.2.2.2 CLASSES HERDEIRAS DA INTERFACE PRODUTO

Iremos ilustrar e exemplificar as classes herdeiras, utilizando a Classe Alimento, que utilizará todos. As classes começam criando o get/set, e implementam a lógica dos métodos herdados, logo em seguida, criando um construtor.

```

class Alimento : IProduto, IComparable<IProduto>
{
    public double Imposto { get => 0.25; }
    public string ID { get; private set; }
    public string Categoria { get; private set; }
    public string Nome { get; private set; }
    public double Preco { get; private set; }
    public string TipoVenda { get; set; }
    public double Quantidade { get; set; }
    public double QuantidadeComprada { get; set; }
    public double QuantidadeVendida { get; set; }
    public DateTime DataCadastro { get; private set; }
    public string Descricao { get; private set; }

    public int CompareTo(IProduto produto)
    {
        // Se o número de vendas for igual então faz a ordenação de acordo com o maior preco
        if (this.QuantidadeVendida == produto.QuantidadeVendida)
        {
            return produto.Preco.CompareTo(this.Preco);
        }
        // Ordenação padrão : do maior número de vendas para o menor
        return produto.QuantidadeVendida.CompareTo(this.QuantidadeVendida);
    }

    public Alimento()
    {
    }

    public Alimento(string id, string nome, double preco, int quantidade, int quantidadeVendida, DateTime dataCadastro, string descricao)
    {
        ID = id;
        Categoria = "Alimentos";
    }
}

```

Começando pelo método CompareTo, da interface IComparable<>, que é uma interface pronta no ambiente do Visual Studio. Seu método foi utilizado para comparar objetos e facilitando na ordenação das listas:

```

public int CompareTo(IProduto produto)
{
    // Se o número de vendas for igual então faz a ordenação de acordo com o maior preco
    if (this.QuantidadeVendida == produto.QuantidadeVendida)
    {
        return produto.Preco.CompareTo(this.Preco);
    }
    // Ordenação padrão : do maior número de vendas para o menor
    return produto.QuantidadeVendida.CompareTo(this.QuantidadeVendida);
}

```

Dando sequência, o método construtor, e o método para adicionar as informações no arquivo:

```

public Alimento(string id, string nome, double preco, int quantidade, int quantidadeVendida, DateTime dataCadastro, string descricao)
{
    ID = id;
    Categoria = "Alimentos";
    Nome = nome;
    Preco = preco;
    TipoVenda = "Unidade";
    Quantidade = quantidade;
    QuantidadeVendida = quantidadeVendida;
    DataCadastro = dataCadastro;
    Descricao = descricao;
}

public string EscreverArquivo()
{
    return ID + ";" + Categoria + ";" + Nome + ";" + Preco.ToString("F2") + ";" + Quantidade + ";" + QuantidadeVendida + ";" + DataCadastro.ToString() + ";";
}

```

3. CONCLUSÃO

Considerando as medidas e análises realizadas neste trabalho, percebemos que para a criação dos programas, foram utilizados recursos práticos e teóricos desenvolvidos em salas de aula e em laboratório de programação orientada a objeto como como herança, polimorfismo, coesão, modularização, encapsulamento, UML, entre outros.

Neste trabalho, implementamos a UML (Unified Modeling Language) com o seu principal objetivo de especificar, construir, visualizar e documentar melhor os programas orientados a objetos.

Para a criação a UML, utilizamos uma extensão do Visual Studio. E para a criação dos programas foram utilizados o Visual Studio e a linguagem de programação usada é o C#.

Conclui-se que nós aperfeiçoamos nossos conhecimentos para a criação de programas orientados a objeto e percebemos que este modelo de análise e programação é um método mais eficiente e mais empregado para o desenvolvimento de softwares.