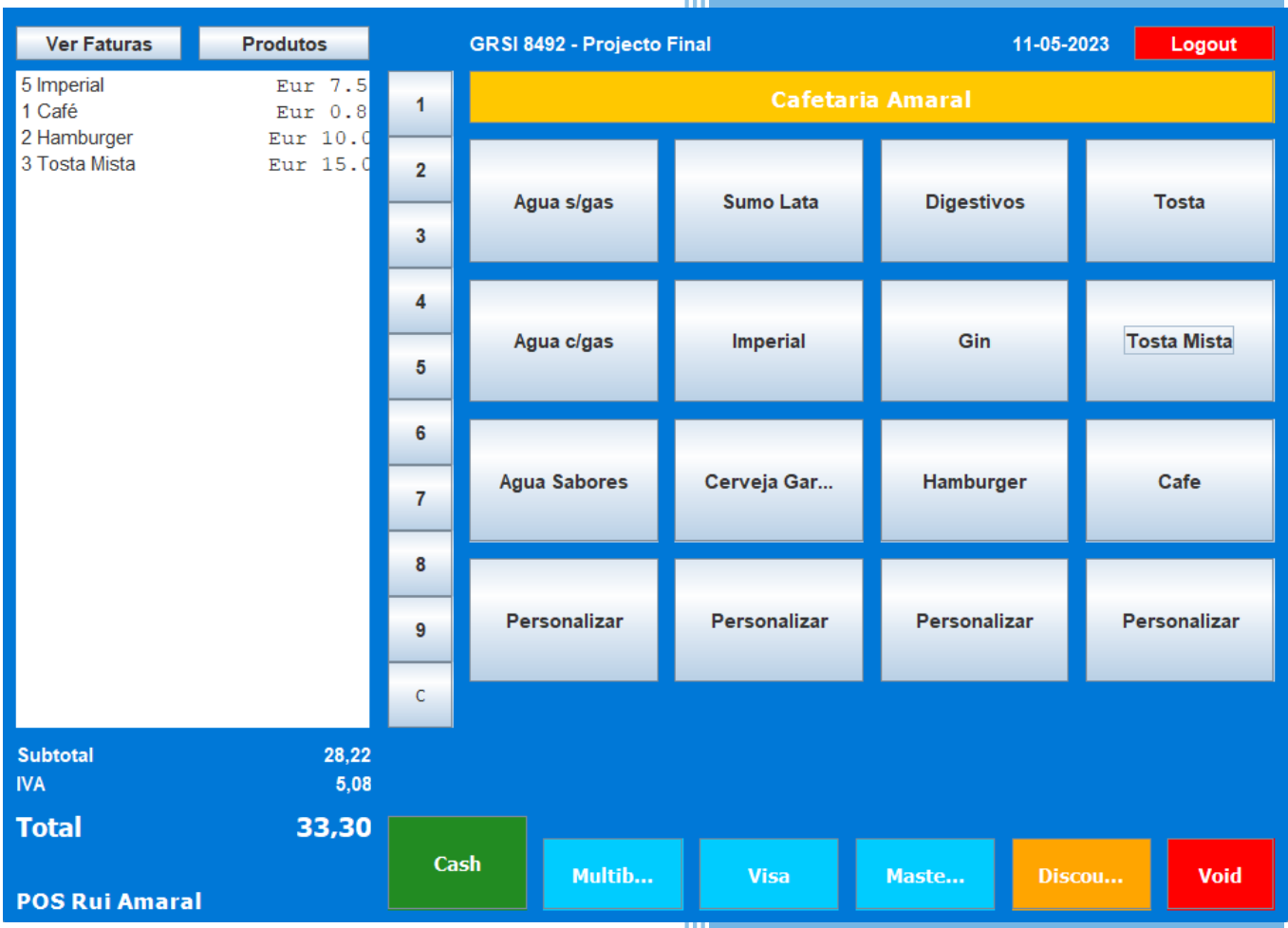
2023

# Projecto Final Programação JAVA

# POS System





Rui Amaral

GRSI 8492


11-05-2023

# Conteúdo

Introdução ..... 2
Apresentação em vídeo: ..... 2
Criação do Programa (em Eclipse IDE) ..... 3
Criação da Interface Gráfica ..... 7
Apresentação Visual do Programa ..... 8
Código Essencial para conexão à base de dados ..... 8
Organização estrutural ..... 9
Código comum a várias funções ..... 10
Botões de Artigos ..... 10
    Não Personalizáveis pelo utilizador: ..... 10
    Personalizáveis pelo utilizador: ..... 11
Multiplicadores ..... 12
    Limpar Botões ..... 13
Função Pesquisar Faturas / Produtos ..... 13
Botão Imprimir ..... 14
Botões de Pagamento ..... 15
Hora ..... 16
Botões de inserção e eliminação de artigos ..... 16
Dificuldades ..... 17
Conclusões ..... 18

# Introdução

Para o projeto final, pretendeu-se desenvolver um sistema de POS, pronto a ser usado num qualquer estabelecimento de vendas. O sistema desenvolvido é facilmente personalizável e permite o ajuste de artigos, valores e ivas
Neste projeto, em concreto, o tipo de estabelecimento escolhido para o funcionamento do programa foi uma cafetaria.
Pretendia-se então um software de vendas que permitisse o registo informático das vendas efetuadas, calculasse os valores a pagar pelo cliente, emitisse faturas com Ivas calculados, emitisse notas de crédito e ainda guardasse um histórico em base de dados dessas mesmas faturas.

O programa em si também permite a personalização por parte do cliente, de alguns botões de registo, tendo os mesmos acessos a outra base de dados onde através do próprio POS é possível a introdução ou extinção de um produto.

Assim, como objetivos propostos para esta tarefa definiu-se os seguintes pontos:

Interface de utilizador amigável: O sistema de POS deve ter uma interface intuitiva e fácil de usar para os funcionários da cafetaria. Eles devem ser capazes de registar as vendas de forma eficiente e rápida.

Registo de vendas: O sistema deve permitir que os funcionários registem as vendas realizadas, inserindo os itens comprados e suas quantidades. Isso pode ser feito através de um sistema de toque ou de um teclado.

Cálculo de valores: O software deve calcular automaticamente os valores a serem pagos pelos clientes com base nos itens comprados, seus preços individuais e quantidades. Ele também deve ser capaz de aplicar os impostos adequados, como o IVA (Imposto sobre Valor Acrescentado).

Emissão de faturas: O sistema deve ser capaz de gerar faturas para os clientes, exibindo os detalhes da compra, os valores totais, incluindo impostos, e outras informações relevantes, como o nome e endereço da cafetaria.

Notas de crédito: Em caso de devoluções ou cancelamentos de pedidos, o sistema deve ser capaz de gerar notas de crédito para os clientes.

Histórico de vendas e base de dados: O sistema deve armazenar um histórico das vendas realizadas numa base de dados.

Personalização: O software deve ser personalizável para atender às necessidades específicas da cafetaria. Isso pode incluir a capacidade de adicionar ou excluir itens do menu, ajustar preços e configurar diferentes taxas de IVA.

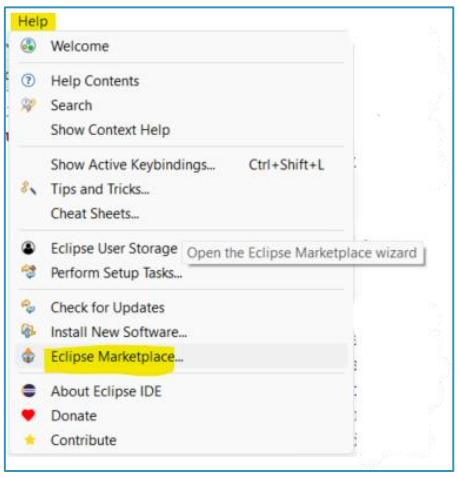
O programa deveria ser criado em “Eclipse IDE” com conexão a uma base de dados criada no Xampp e phpMyAdmin.

# Apresentação em vídeo:

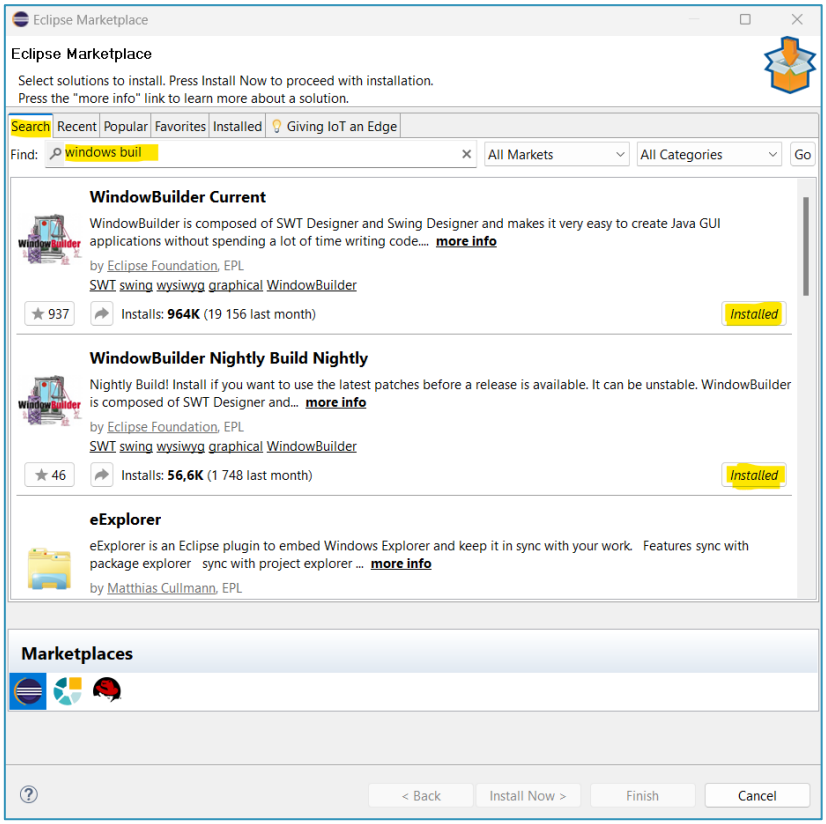
https://mega.nz/file/NR5BXYQb#53MNVb43ZVloXDhqiKkvmzaahIjQtsmM8CpkjFoK-_s

# Criação do Programa (em Eclipse IDE)

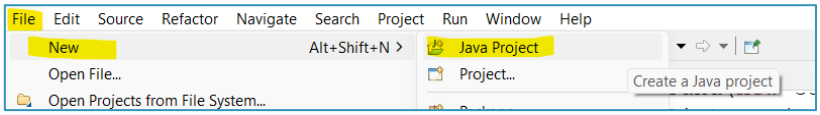
## Abrir o Eclipse

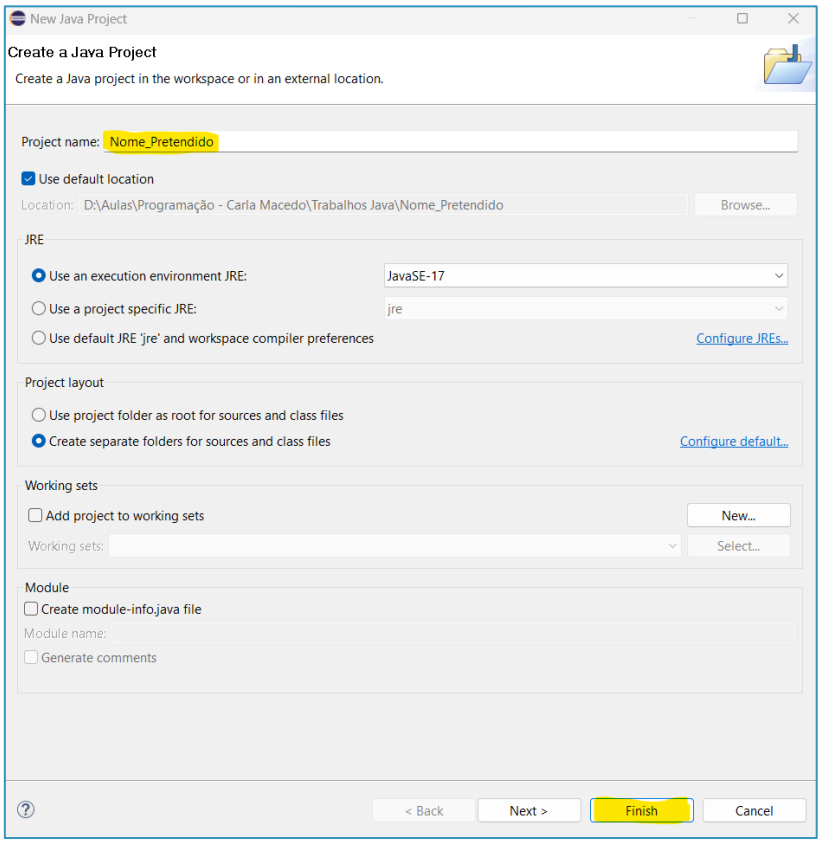


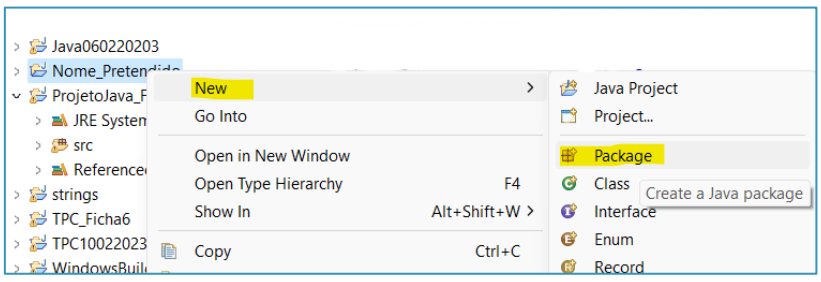

## Instalar as duas apps

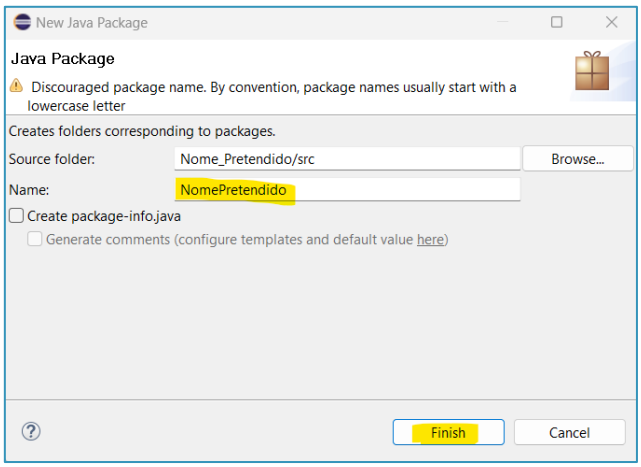


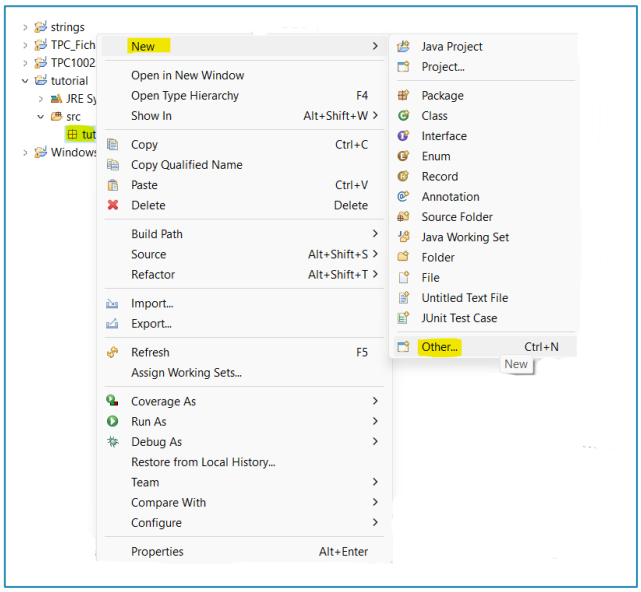
Criação do Main:

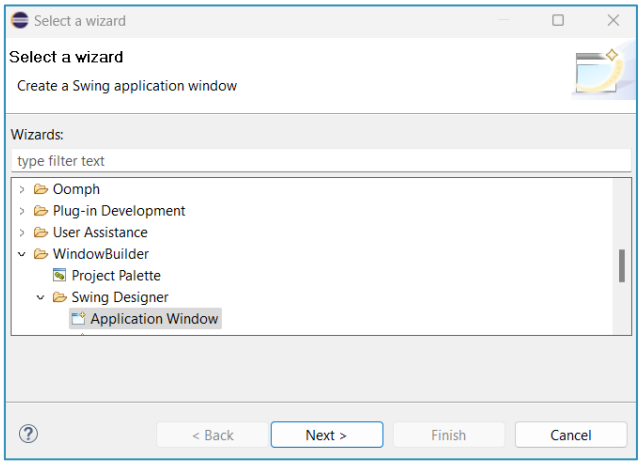


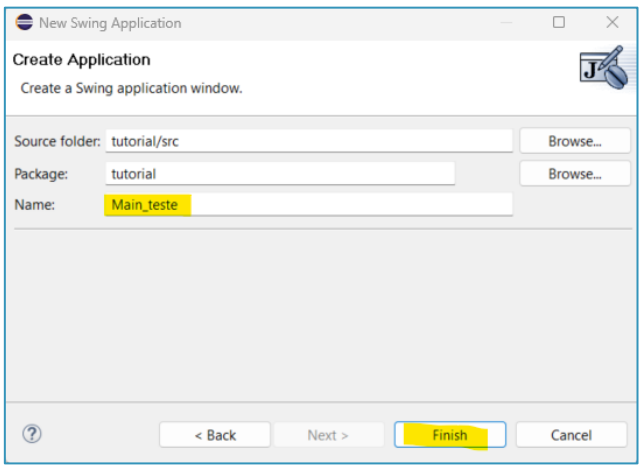



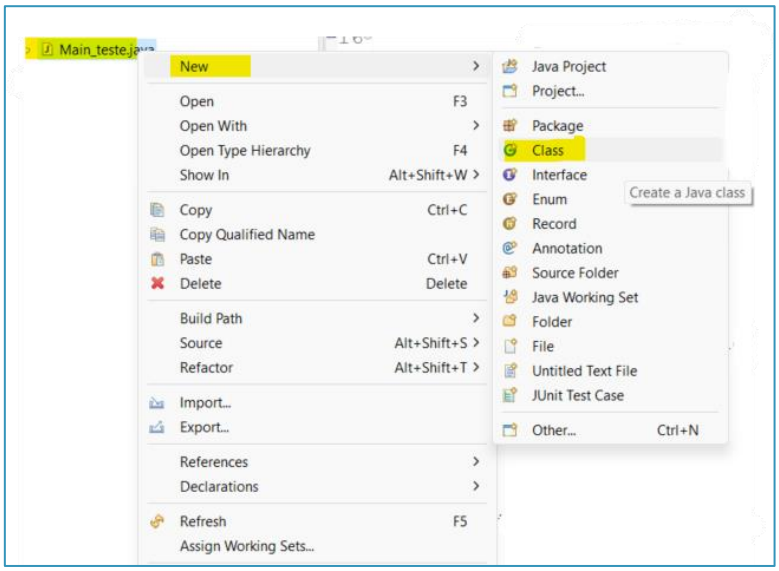



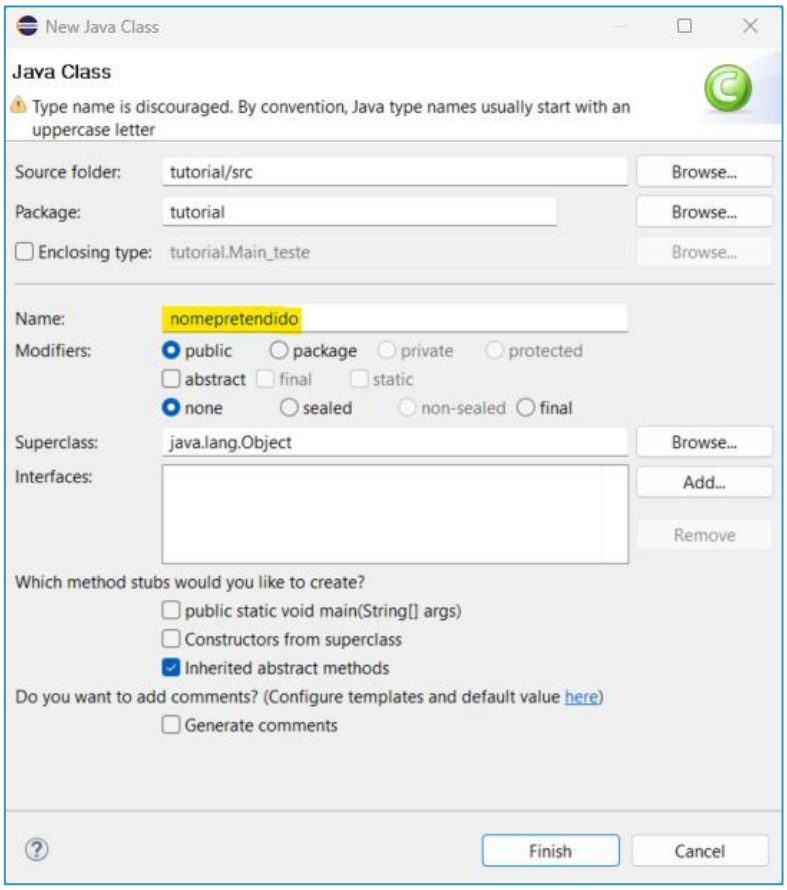

strings
TPC_Fich
TPC1002
tutorial
JRE Sy
src
tut
Windows
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Coverage As
Run As
Debug As
Restore from Local History...
Team
Compare With
Configure
Properties Alt+Enter
Java Project
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Other... Ctrl+N
New


Select a wizard
Select a wizard
Create a Swing application window
Wizards:
type filter text
Oomph
Plug-in Development
User Assistance
WindowBuilder
Project Palette
Swing Designer
Application Window
< Back
Next >
Finish
Cancel


New Swing Application
Create Application
Create a Swing application window.
Source folder: tutorial/src
Browse...
Package: tutorial
Browse...
Name: Main_teste
< Back
Next >
Finish
Cancel

Criação de classes

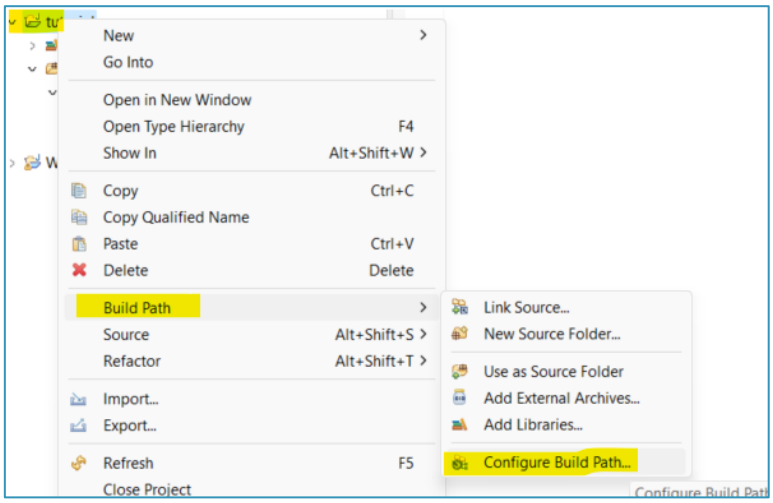


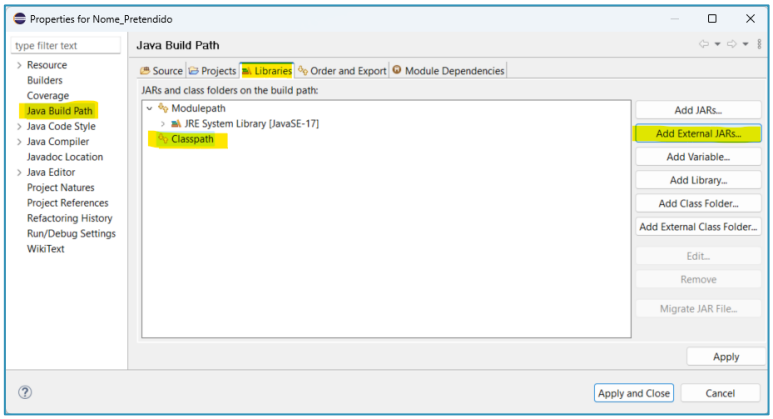




Instalação do Driver

Fazer download do driver para conectar à Base de Dados

http://www.java2s.com/Code/Jar/c/Downloadcommysqljdbc515jar.htm




Inserir o Driver descarregado anteriormente

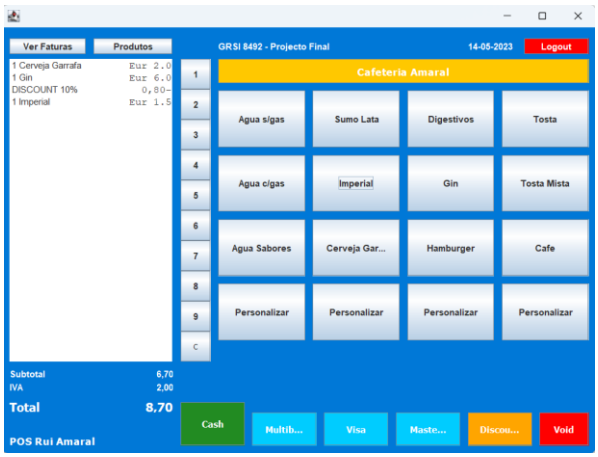
## Criação da Interface Gráfica

Foram usados 5 Jpanels cada um com uma interface diferente, no primeiro painel pretendia-se o menu inicial com botões para registar produtos, uma TextArea para mostrar os itens registados e respetivos valores e algumas Jlabels que atualizam o texto automaticamente consoante a soma dos valores dos artigos introduzidos.

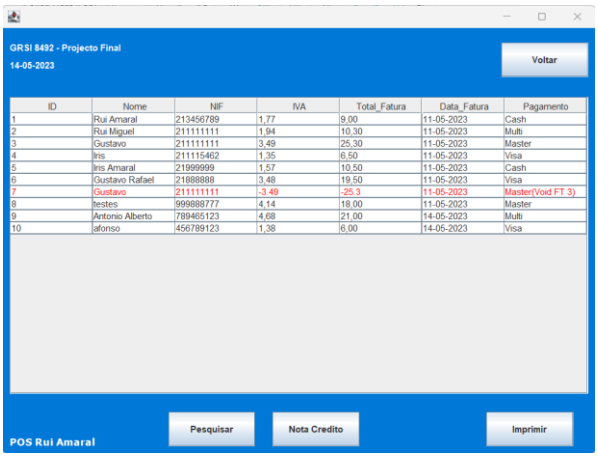
Foram também criados alguns botões para transitar entre painéis de forma a mostrar mais funcionalidades do programa, um botão em forma de título que ao pressionar 3x limpa as configurações dos botões e uma TextField que mostra a hora do sistema.

Nas restantes Jlabels introduziram-se funcionalidades de visualização, edição e eliminação de dados das bases de dados, consoante o pretendido.

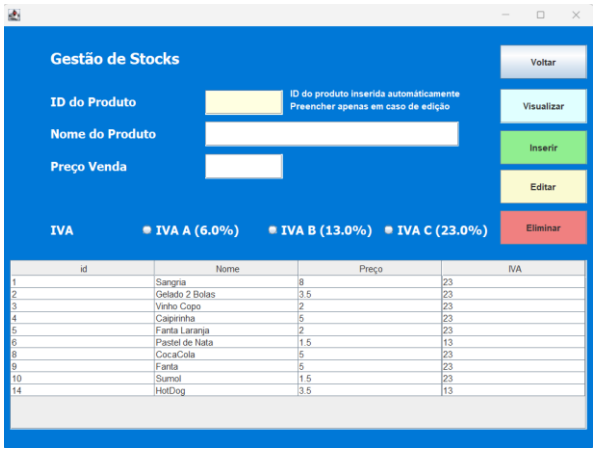
## Apresentação Visual do Programa



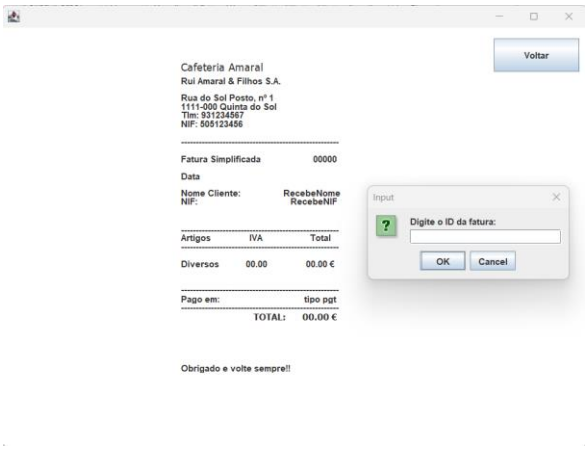

Menu Inicial




Visualizar Faturas




Gestão de Stock




Impressão Faturas

## Código Essencial para conexão à base de dados

Para o programa funcionar com ligação a uma base de dados, é necessário que além da instalação do driver SQL, ele seja chamado. Esse código deve ser criado fora do main numa classe (a qual vamos chamar de jdbccafe) e deve possuir métodos para conectar, verificar a conexão e fazer logout da Base de dados MySQL.

No fundo, essa classe jdbccafe é usada para gerir a conexão com a Base de dados MySQL e executar consultas.

```
package ProjetoJava_Final_Rui;

import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```java
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class jdbccafe {
    static Connection connection = null;
    Statement statement = null; //gere as consultas
    ResultSet resultset = null; //armazena as informacoes das consultas


//**************** Conecta ao driver do XAMPP ************
public void conectar()
    {
    String JDBC_DRIVER = "com.mysql.jdbc.Driver";//caminho do driver que foi adicionado
    String DB_URL = "jdbc:mysql://localhost:3306/cafe";//porta padrao que o mysql usa
    String user = "root";
    String pass = "";
    try {
        Class.forName("com.mysql.jdbc.Driver");
        this.connection=  (Connection) DriverManager.getConnection(DB_URL, user, pass);
        this.statement=  (Statement) this.connection.createStatement();
        }catch (Exception e) //armazena informação sobre erros
           {System.out.println("erro: " + e.getMessage());} //retorna uma msg com o erro existente
    }

//*************** metodo que retorna um valor booleano ***************
//*************se conseguiu conectar então é diferente de null, dá resultado true ***************

public boolean estaconectado() {
    if(this.connection!=null) {
        return true;
    }else { //senao nao conseguiu connectar entao continua null, entra no false
        return false;}
}


//**********************logout com mensagem***************
public void logout() {
    try {
        this.connection.close();
        }catch(Exception e) {
         System.out.println("erro: " + e.getMessage());}
}
```

## Organização estrutural

Para facilitar a compreensão do programa e o seu funcionamento sem erros, no código “main” começou-se pelas declarações das diversas variáveis, seguidamente dos Jpanels, TextFields, Jlabels e todos os componentes que recebem informação seguidamente dos botões e respetivas instruções de código.

Foi criada uma classe à parte com 12 métodos distintos, para conexão à base de dados, e para chamadas ao sistema de forma a encurtar código em funções semelhantes.

O programa comporta 2 tabelas distintas, ligadas a 2 tabelas da mesma base de dados, com as funções de armazenarem “faturas” e de armazenar “artigos”.

Existem diversos botões no programa, alguns com funções fixas e outros com funções personalizáveis pelo utilizador.

## Código comum a várias funções

No programa existem partes de código que são comuns a várias funções e métodos criados, sendo eles:

```
jdbccafe cafe = new jdbccafe();
```

Cria uma instância da classe jdbccafe. Essa classe é responsável pela ligação à Base de dados.

```
cafe.conectar();
```

Chama o método conectar() na classe cafe para estabelecer a ligação com a Base de dados.

```
if (cafe.estaconectado()) {
```

Verifica se a conexão à Base de dados foi estabelecida com sucesso.

```
String sql = "SELECT * FROM <nome da tabela a consultar>";
```

Define uma string contendo uma consulta SQL, esta string será modificada e usada para executar ações na Base de dados

```
sql += " WHERE id = " + id;
```

Adiciona uma cláusula WHERE à consulta SQL para filtrar a consulta consoante o ID fornecido.

```
resultset = statement.executeQuery(sql);
```

Executa a consulta SQL na Base de dados e armazena o resultado num objeto ResultSet chamado resultset.

```
registo[0] = resultset.getInt("id");
```

Obtém o valor da coluna "id" da linha atual do resultset e armazena-o no primeiro elemento do array registo.

## Botões de Artigos

No Programa, conforme anteriormente descrito, temos 2 tipos de botões de artigos:

### Não Personalizáveis pelo utilizador:

No exemplo o botão “Agua s/gas” ao ser pressionado insere o texto “agua s/gas” juntamente com o valor da variável “Multiplicador”, numa TextArea. Multiplica o valor da variável “agua s/gas” pela variável “Multiplicador”, acumula esse valor numa variável “TOTAL”, calcula o valor do IVA do produto e coloca na variável “TAXA” e no fim mostra esses valores em Jlabels e TextAreas diferentes

```
//************* Botao não personalizavel Agua Sem Gas **************

JButton btnAguaSemGas = new JButton("Agua s/gas\r\n");      //nome do botão e texto a apresentar
btnAguaSemGas.addActionListener(new ActionListener() {      //"escuta" quando o botao é pressionado e executa o codigo seguinte
    public void actionPerformed(ActionEvent e) {
    textAreaValor.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT); //Informa que a TextAreaValor será alinhada á dir. (irá receber valores posterior
    textAreaProdutos.append(" "+Multiplicador +" Agua s/gas\n");  // Coloca o valor da variavel Multiplicador e o texto "Agua s...." na TextAreaProdutos
    textAreaValor.append("Eur " +AguaSemGas*Multiplicador +" \n");  // Multiplica 2 variaveis e coloca o valor do produto na TextAreaValor

    TOTAL=TOTAL+(AguaSemGas*Multiplicador);  //Calcula e acumula o valor do produto consoante o multiplicador escolhido
    TAXA = TAXA + (AguaSemGas*IVA_B/100*Multiplicador); //Calcula e acumula o valor do IVA tendo em conta o multiplicador escolhido

    lblSubtotal_1.setText(String.format("%.2f",TOTAL-TAXA)); // Calcula e coloca o valor do produto sem VA na label subtotal
    lblTotal_Iva.setText(String.format("%.2f",TAXA));// coloca o valor IVA na label subtotal
    lblTOTAL.setText(String.format("%.2f", TOTAL));  //Coloca o valor do produto com IVA na label TOTAL

    Multiplicador=1;  //Reeinicia o Multiplicador para começar a 1 no proximo produto
    }
});
btnAguaSemGas.setBounds(286, 84, 115, 75);
Artigos.add(btnAguaSemGas);
```

## Personalizáveis pelo utilizador:

No exemplo, quando o botão "Personalizar" é pressionado, ele verifica se já foi personalizado verificando o valor da variável "artigo1". Se o valor for igual a 0, exibe uma caixa de diálogo para confirmar se o utilizador pretende personalizá-lo. Se a resposta for sim, solicita o número do artigo que o botão deve representar e armazena o valor na string "inputid". Em seguida, essa string é convertida para a variável "Botao1Produto" e "Botao1Preco".

A seguir, uma instância da classe "jdbccafe" é criada e o método "conectar" é chamado para estabelecer uma conexão com o banco de dados. Se a conexão for bem-sucedida (verificado pelo método "estaconectado" da classe "jdbccafe"), o método "stock" é chamado, passando o valor de "Botao1Produto" como argumento. Esse método consulta a base de dados com base no valor do ID do produto e retorna o nome do produto correspondente na variável "nomeproduto".

Em seguida, o texto do botão "Personalizar" é atualizado com o valor de "nomeproduto". O método "stock2" é chamado, passando o valor de "Botao1Preco" como argumento. Esse método consulta novamente a tabela da base de dados e retorna o preço do produto na variável "precoproduto".

Com as informações armazenadas nas variáveis "nomeproduto" e "precoproduto", o botão personaliza seu texto apresentando o nome do produto. Em seguida, ele executa o código semelhante ao botão anterior, adicionando o texto da variável "nomeproduto" juntamente com o valor da variável "Multiplicador" em uma TextArea. O valor de "precoproduto" é multiplicado pelo valor de "Multiplicador" e adicionado ao valor acumulado na variável "TOTAL". O valor do IVA do produto é calculado com base em "precoproduto" e adicionado ao valor acumulado na variável "TAXA". Os valores de "TOTAL", "TAXA" e subtotal são exibidos em diferentes JLabels e TextAreas.

Caso o botão já tenha sido personalizado anteriormente (valor diferente de 0 em "artigo2"), ele executa o código como um botão normal, sem solicitar novas informações ao utilizador.

```
//*************** Botao personalizar ****************************
JButton btnPersonalizar = new JButton("Personalizar");      //nome e descrição do botao
btnPersonalizar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {        //escuta se for pressionado
        if (artigo1==0) {                               //se não tiver personalização anterior (valor da variavel a zero) executa o codigo
            int saida = JOptionPane.showOptionDialog(null, "Pretende Personalizar o Botão?", "Confirmação",     //confirma se pretende a personalizaçao
                    JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, new Object[] {"Sim", "Não"}, "Não");
            if (saida == JOptionPane.NO_OPTION) {
                return;}
            else  {String inputid = JOptionPane.showInputDialog(null, "Digite o id do produto:");       //pede a introduçao do id do artigo a personalizar

            if (inputid == null) {                      // se o utilizador nao digitar id ou clicar em "cancelar" a execução do codigo é cancelada
                return;}
            int Botao1Produto = Integer.parseInt(inputid);      //converte a string "input" que contem "id do produto" em um inteiro "Botao1Produto "
            int Botao1Preco=Botao1Produto;                      //Atribui o valor de Botao1Produto à variável Botao1Preco
            jdbccafe cafe = new jdbccafe();
            cafe.conectar();                            //Chama o método conectar no objeto cafe para estabelecer uma conexão com o banco de dados
            if (cafe.estaconectado()) {
                nomeproduto = cafe.stock(Botao1Produto);    //Chama o método stock(Botao1Produto)para obter o nome do produto com base no ID do produto.
                                                            //O nome do produto é armazenado na variável nomeproduto.

                btnPersonalizar.setText(nomeproduto);       //Altera o texto do botão btnPersonalizar com o valor de nomeproduto
                double precoproduto = cafe.stock2(Botao1Preco);     //Chama o método stock2(Botao1Preco) para obter o preço do produto
                artigo1 = precoproduto;                     //atribui à variável artigo1 o valor de precoproduto
                textAreaProdutos.append(" "+Multiplicador + " " +nomeproduto +"\n");
                textAreaValor.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
                textAreaValor.append("Eur " +artigo1*Multiplicador +" \n");
                TOTAL=TOTAL+(artigo1*Multiplicador);
                TAXA = TAXA + (artigo1*IVA_B/100*Multiplicador);
                lblSubtotal_1.setText(String.format("%.2f",TOTAL-TAXA));
                lblTotal_Iva.setText(String.format("%.2f",TAXA));
                lblTOTAL.setText(String.format("%.2f", TOTAL));
                Multiplicador=1;
                }
            else {System.out.println("Não foi possivel conectar à BD");}
            }
            }
        else {              //caso o botao já tenha um valor definido na variavel artigo1, ele ja nao se personaliza e executa o codigo como um botao normal
            textAreaProdutos.append(" "+Multiplicador + " " +nomeproduto +"\n");
            textAreaValor.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
            textAreaValor.append("Eur " +artigo1*Multiplicador +" \n");
            TOTAL=TOTAL+(artigo1*Multiplicador);
            TAXA = TAXA + (artigo1*IVA_B/100*Multiplicador);
            lblSubtotal_1.setText(String.format("%.2f",TOTAL-TAXA));
            lblTotal_Iva.setText(String.format("%.2f",TAXA));
            lblTOTAL.setText(String.format("%.2f", TOTAL));
            Multiplicador=1;}
        }
});
btnPersonalizar.setBounds(661, 339, 115, 75);
Artigos.add(btnPersonalizar);
```

## Multiplicadores

Os botões multiplicadores, não são nada mais do que a adição de um valor a uma variável. Essa variável de nome “Multiplicador” é usada para que quando pressionado o botão 1, 2, 3, 4…. Ele multiplique esse número pelo valor do produto referente ao próximo botão de artigo a ser pressionado.
Assim, exemplificando, ao pressionar o botão “5” e depois o botão “tosta”, ele permite a multiplicação do valor de uma tosta por x5, evitando que seja necessário carregar 5 vezes no botão “tosta”. Caso o utilizador tenha pressionado o botão Multiplicador errado, existe um outro botão “C” que limpa o valor da variável Multiplicador e a recoloca a x1.

Exemplo do código do botão “1” que coloca a variável Multiplicador=1.

```
//************** Botao Multiplicador 1 ***************

JButton btn1 = new JButton("1");
btn1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
     Multiplicador=1;
     }
});
btn1.setBounds(236, 42, 40, 40);
Artigos.add(btn1);
```

## Limpar Botões

De forma a permitir a personalização infinita dos botões personalizaveis, foi necessário criar código que lhe fizessem um "reset", assim, ao clicar na barra de título "Cafeteria Amaral", a mesma deve limpar as personalizações dos botões configuráveis, para isso apresenta uma mensagem de confirmação e de seguida coloca as 4 variáveis (artigo1, artigo2, artigo3 e artigo4) a zero de forma a que os botões ao serem pressionados iniciem com as mesmas a zero.
O código também executa a ação de recolocar o nome dos botões como "Personalizar"

```
// ***** 3 clicks na barra e reeinicia os botoes personalizaveis **********

JButton btnBarrCafetaria = new JButton("Cafeteria Amaral");
btnBarrCafetaria.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        clickCount++;
        if (clickCount == 3) {
            int dialogResult = JOptionPane.showConfirmDialog(null, "Deseja executar a ação?", "Confirmação", JOptionPane.YES_NO_OPTION);
            if (dialogResult == JOptionPane.YES_OPTION) {
                nomeproduto="Personalizar";
                artigo1=0;artigo2=0;artigo3=0;artigo4=0;
                btnPersonalizar.setText(nomeproduto);
                btnPersonalizar_1.setText(nomeproduto);
                btnPersonalizar_2.setText(nomeproduto);
                btnPersonalizar_3.setText(nomeproduto);
                JOptionPane.showMessageDialog(null, "Botões formatados!");
                }
            clickCount = 0;
            }
        }
    });
btnBarrCafetaria.setFont(new Font("Tahoma", Font.BOLD, 14));
btnBarrCafetaria.setHorizontalTextPosition(SwingConstants.CENTER);
btnBarrCafetaria.setForeground(Color.WHITE);
btnBarrCafetaria.setBackground(Color.ORANGE);
btnBarrCafetaria.setBounds(286, 42, 490, 32);
Artigos.add(btnBarrCafetaria);
```

# Função Pesquisar Faturas / Produtos

Com a finalidade de visualizar os dados inseridos na base de dados, foram criados alguns botões para o efeito em que o seu funcionamento é semelhante.
Ao serem pressionados, os botões devem chamar um método numa classe separada, a qual fará uma query sql á nossa base de dados e retornará os valores dentro de uma tabela inserida num scrollpane.

O método do exemplo a seguir "exibirFaturas", verifica se a string "input" está vazia ou se tem um valor inserido pelo utilizador. Após essa verificação permite fazer uma consulta SQL com base no valor fornecido pelo utilizador e mostra o resultado da consulta numa tabela com as colunas ID, Nome, NIF, IVA, etc…

Caso a consulta retorne algum valor negativo, essa linha da coluna deverá ficar com texto a vermelho. O objetivo é identificar visualmente Notas de Crédito.

```
public DefaultTableModel exibirFaturas(String input) {
    jdbccafe cafe = new jdbccafe();
    cafe.conectar();
    DefaultTableModel modelo = new DefaultTableModel();
    if (cafe.estaconectado()) {
        try {
            int id = -1;
            modelo.setRowCount(0);
            String sql = "SELECT * FROM fatura";
            if (input != null && !input.isEmpty()) {id = Integer.parseInt(input);  }
            if (id != -1) {sql += " WHERE id = " + id;}
            resultset = statement.executeQuery(sql);
            modelo.addColumn("ID");
            modelo.addColumn("Nome");
            modelo.addColumn("NIF");
            modelo.addColumn("IVA");
            modelo.addColumn("Total_Fatura");
            modelo.addColumn("Data_Fatura");
            modelo.addColumn("Pagamento");
            if (resultset != null) {
                while (resultset.next()) {
                    Object[] registo = new Object[7];
                    registo[0] = resultset.getInt("id");
                    registo[1] = resultset.getString("Nome");
                    registo[2] = resultset.getString("NIF");
                    registo[3] = resultset.getString("IVA");
                    registo[4] = resultset.getString("Total_Fatura");
                    registo[5] = resultset.getString("Data_Fatura");
                    registo[6] = resultset.getString("Pagamento");
                    modelo.addRow(registo);
                    if (registo[6].toString().contains("Void FT")) {
                        for (int i = 0; i < 7; i++) {
                            modelo.setValueAt("<html><font color='red'>" + registo[i] + "</font></html>", modelo.getRowCount() - 1, i);
                            }
                        }
                }

            }
        } catch (SQLException e) {e.printStackTrace(); }
        finally {logout();}
    }
    return modelo;
```

## Botão Imprimir

A função do botão imprimir é gerar uma imagem de uma fatura para cliente. A fatura deve ser capaz de, mediante o número introduzido pelo utilizador, executar uma query e resgatar todos os dados registados na base de dados, referentes a essa consulta, colocando-os em Jlabels alinhadas estrategicamente.

```
// ************ Botão Imprimir **************
        JButton btnImprimir = new JButton("Imprimir");
        btnImprimir.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                 Fatura.setVisible(false);
                    Papel.setVisible(true);
                String idFatura = JOptionPane.showInputDialog(null, "Digite o ID da fatura:");  // pede o nr da fatura a consultar

                try {
                    jdbccafe cafe = new jdbccafe();
                    cafe.conectar();

                    String sql = "SELECT * FROM fatura WHERE id = " + idFatura;  // cria a query com o nr escolhido pelo utilizador
                    ResultSet resultSet = cafe.statement.executeQuery(sql); // executa a query na base de dados

                    if (resultSet.next()) {             // se a consulta tiver resultados, armazena os valores em diferentes strings
                        String id = resultSet.getString("id");       // armazena o resultado da coluna "id" da base de dados na string id
                        String nome = resultSet.getString("Nome");  // armazena o resultado da coluna "Nome" da base de dados na string nome
                        String nif = resultSet.getString("NIF");  // armazena o resultado da coluna "NIF" da base de dados na string nif
                        String iva = resultSet.getString("IVA");
                        String totalFatura = resultSet.getString("Total_Fatura");
                        String dataFatura = resultSet.getString("Data_Fatura");
                        String pagamento = resultSet.getString("Pagamento");


                        double ivasemvirgula = Double.parseDouble(iva.replace(",", ".")); // Remove a vírgula e substitui por ponto
                        double totalFaturasemvirgula = Double.parseDouble(totalFatura.replace(",", ".")); // Remover a vírgula....

                        // Atualiza as JLabels com os valores obtidos
                            lblRecebeNrFatura.setText(id);
                            lblRecebeNome.setText(nome);
                            lblRecebeNIF.setText(nif);
                            lblRecebeIVA.setText(iva);
                            lblRecebeTotal.setText(totalFatura);
                            lblDataFatura.setText(dataFatura);
                            lblRecebeTipoPagamento.setText(pagamento);
```

```
                    // Realizar a operação de subtração para retornar um valor total sem iva
                    double totalSemIva = totalFaturasemvirgula - ivasemvirgula;

                    // Atualiza a JLabel com o valor calculado
                    String totalSemIva2 = String.format("%.2f", totalSemIva);
                    lblRecebeTotalSemIva.setText(String.valueOf(totalSemIva2));

                } else {
                    // Fatura não encontrada, exibir mensagem de erro ou tratar de outra forma
                }

                resultSet.close();
                cafe.logout(); // Fechar a conexão após o uso do ResultSet
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
    });

    btnImprimir.setBounds(641, 508, 115, 45);
    Fatura.add(btnImprimir);
```

Cafeteria Amaral
Rui Amaral & Filhos S.A.

Rua do Sol Posto, nº 1
1111-000 Quinta do Sol
Tlm: 931234567
NIF: 505123456

---

Fatura Simplificada 1

11-05-2023

Nome Cliente: Rui Amaral
NIF: 213456789

| Artigos | IVA | Total |
|---|---|---|
| Diversos | 1,77 | 7,23 |

Pago em: Cash

TOTAL: 9,00

Obrigado e volte sempre!!

## Botões de Pagamento

Foram criados 4 botões de métodos de pagamento (Cash, Visa e Amex, Mastercard), estes botões têm como finalidade inserir o texto igual ao ao seu nome, dentro de uma string "pagamento" e trocar de painel para o painel de emissão de fatura.
O texto inserido nessa string será armazenado na base de dados, juntamente com os restantes valores da "despesa" do cliente. Pretendia-se assim que na emissão de fatura viesse sempre a informação do tipo de pagamento.

```
//************* Pagamento Cash **************

JButton btnCash = new JButton("Cash");
btnCash.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Pagamento = "Cash";
        Artigos.setVisible(false);
        Dados.setVisible(true);
        }
    });
btnCash.setHorizontalTextPosition(SwingConstants.CENTER);
btnCash.setFont(new Font("Tahoma", Font.BOLD, 12));
btnCash.setForeground(Color.WHITE);
btnCash.setBackground(new Color(34, 139, 34));
btnCash.setBounds(236, 496, 85, 57);
Artigos.add(btnCash);
```

## Hora

Como qualquer programa de faturação, pretendia-se que a hora atual de sistema estivesse presente tanto no menu de utilizador, como nos dados armazenados na Base de Dados, e assim como impresso na fatura.

```
//************* Labels Data Sistema Automática no JPane Artigos e JPane Fatura ***************

JLabel lblData = new JLabel("Data");
lblData.setForeground(Color.WHITE);
lblData.setBounds(616, 19, 65, 13);
Artigos.add(lblData);

// Cria um objeto DateTimeFormatter usando o método ofPattern e passamos como argumento
// a string "dd/MM/yyyy", para definir o padrão de formatação da data.
DateTimeFormatter FormatoDaData = DateTimeFormatter.ofPattern("dd-MM-yyyy");

// retorna a data atual do sistema na forma de um objeto LocalDate.
LocalDate dataAtual = LocalDate.now();

//formata a data atual (dataAtual)no formato especificado ("dd-MM-yyyy") e armazena o resultado
//numa string chamada dataFormatada.
String dataFormatada = FormatoDaData.format(dataAtual);

// Imprime a String na Label
lblData.setText(dataFormatada);

JLabel lblData_1 = new JLabel("Data1");
lblData_1.setForeground(Color.WHITE);
lblData_1.setBounds(10, 43, 65, 13);
Fatura.add(lblData_1);
lblData_1.setText(dataFormatada);
```

## Botões de inserção e eliminação de artigos

Existem botões para eliminação ou adição de artigos ao stock de vendas. Estes botões foram criados a pensar nos botões de personalizar. Assim caso o utilizador pretenda registar algum produto que não estava configurado de origem, pode adicionar ou remover o novo artigo á sua base de dados para posteriormente configurar um dos botões “personalizar”. Imaginando que decide vender o artigo “caracóis” e como este não vem pré configurado, ele pode adicioná-lo e de seguida alterar o botão do menu inicial para que ao ser pressionado registe o produto caracóis na venda ao cliente, se mais tarde decidir que já não quer vender caracóis, ele pode eliminar esse produto dos stocks.

O exemplo seguinte representa o código usado para eliminar um artigo:

```
//************* botao eliminar stocks*****************
// o botao mostra os artigos antes de pedir qual o artigo a eliminar, e atualiza a tabela de seguida *****

JButton btnEliminar = new JButton("Eliminar");
btnEliminar.setBackground(new Color(240, 128, 128));
btnEliminar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)  {
        try {
            jdbccafe cafe = new jdbccafe();
            cafe.conectar();
            modelo2.setRowCount(0);     // limpa os valores da tabela apresentada
            if (cafe.estaconectado()) {
                String sql = "SELECT * FROM stocks";    //executa uma query para mostrar uma tabela atualizada
                try (ResultSet resultSet = cafe.statement.executeQuery(sql)) {
                    while (resultSet.next()) {
                        Object[] registo1 = new Object[5];
                        registo1[0] = resultSet.getInt("id");
                        registo1[1] = resultSet.getString("nomeproduto");
                        registo1[2] = resultSet.getString("precoproduto");
                        registo1[3] = resultSet.getString("IVA");
                        modelo2.addRow(registo1);
                    }
                } catch (SQLException f) {f.printStackTrace();}
// os seguintes passos pedem o numero do artigo a ser eliminado
                try {
                    String id = JOptionPane.showInputDialog(null, "Digite o ID do artigo a eliminar:");
                    if (id == null || id.isEmpty()) {
                        JOptionPane.showMessageDialog(frame, "É necessário introduzir um id");
                        return;
                    }
```

```
                }
//com o numero do artigo a eliminar, chama o metodo "deleteFun" que elimina o produto da base de dados
                    cafe.deleteFun(Integer.parseInt(id));
                    JOptionPane.showMessageDialog(frame, "Artigo Removido");
                    modelo2.setRowCount(0); // limpa a tabela anteriormente apresentada

// atualiza novamente a tabela apresentada ao utilizador
                    try (ResultSet resultSet = cafe.statement.executeQuery(sql)) {
                        while (resultSet.next()) {
                            Object[] registo1 = new Object[5];
                            registo1[0] = resultSet.getInt("id");
                            registo1[1] = resultSet.getString("nomeproduto");
                            registo1[2] = resultSet.getString("precoproduto");
                            registo1[3] = resultSet.getString("IVA");
                            modelo2.addRow(registo1);
                        }
                    }
                } catch (NumberFormatException ex) { //identifica se o numero é inteiro
                    JOptionPane.showMessageDialog(frame, "O ID deve ser um número inteiro.");
                }
                cafe.logout();
            } else {
                System.out.println("Não foi possível conectar ao BD");
            }
        } catch (Exception f) {
            f.printStackTrace();
        }
    }
});
btnEliminar.setBounds(661, 245, 115, 45);
InsereStock.add(btnEliminar);
```

## Dificuldades

Considero que a elaboração do código foi relativamente simples, não é um código complexo nem demasiado comprido, e apesar de saber que ainda é possível reduzir mais, optei por deixar algumas partes duplicadas em vez de serem criadas ou usadas num método à parte, de forma a facilitar a minha compreensão e aprendizagem.

Foram sentidas algumas dificuldades que considero normais, no conhecimento de instruções a serem dadas, nomeadamente na escrita das mesmas por desconhecimento de como as usar, por exemplo a introdução de uma data automática, mas que com a ajuda de alguma pesquisa online e de chats AI foram possíveis de ultrapassar e compreender o seu funcionamento.

# Conclusões

Foi criado um programa demonstrativo, ainda que básico, funcional e de interface simples e intuitiva para o utilizador.
Foram cumpridos todos os requisitos a que me propus, mas gostaria de o aperfeiçoar acrescentando pelo menos 6 funcionalidades que considero importantes para o programa ser considerado pronto a uso real.

A primeira funcionalidade, é apenas uma melhoria do aspeto gráfico, pois acho que poderia inserir uma imagem ilustrativa do artigo a ser vendido, em cada botão, em vez de aparecer apenas “agua”. Mas não soube como o fazer.
Gostaria de ter conseguido implementar um ficheiro SAFT, o que não foi feito também por desconhecimento e falta de tempo para pesquisa.

Gostaria de ter melhorado a base de dados de stocks, com a hipótese de ter sempre o registo do total de produtos disponíveis em stock, não o fiz porque em parte considerei que essa função, num programa de registo vendas de uma cafeteria não faz sentido estar incluída, pois não é possível saber as quantidades reais de doses de bebidas vendidas ou engarrafadas, mas por outro lado, seria útil para caso o programa fosse adaptado a outro tipo de estabelecimento como um minimercado.

Gostaria também de com mais tempo melhorar a quantidade de dados armazenados aquando de uma venda, de forma que na emissão da fatura o programa fosse buscar e discriminasse todos os artigos refentes á venda, em vez de ir apenas retirar valores e data.
Essa funcionalidade não foi incluída por não saber de que forma poderia armazenar esses valores na base de dados para que mais tarde os fosse resgatar. Deveu-se apenas a uma não compreensão de como fazer o armazenamento, e não à forma de como o código os poderia resgatar.

Gostaria de ter tido tempo para implementar a função de mais do que um user.

Por fim, conseguir fazer a impressão para papel real de uma fatura. Não o fiz, novamente por não ter tido tempo para estudar e compreender o seu funcionamento.