Nome dos integrantes do grupo: Adrielly Clara e Luis Antonio

Turma: INFO3

1. Introdução

Seu José vai abrir uma mercearia e, para isso, pretendia utilizar uma planilha do Excel para facilitar seu trabalho. No entanto, percebeu que, para "fechar" uma compra de algum cliente, era preciso um grande tempo e, por isso, deseja que um software seja criado para facilitar seu trabalho. Tal programa terá uma interface em que o usuário poderá escrever, inclusive, o código do produto da mercearia e, com isso, a compra será gerenciada de forma mais produtiva e rápida. Essa interface contará com quatro campos de texto: código, nome (nome do produto na planilha do Excel), quantidade, preço unt. (preço daquele produto informado pela planilha do Excel). O operador do caixa irá informar somente o código do produto e a quantidade desejada. O software pretende, também, criar uma tabela para estoque de produtos. Dessa forma, neste documento, será exposto a finalidade e as especificidades do código gerado, que almeja amparar o pequeno empresário.

2. Implementação

Descrição das classes:

- Classe "Produto":

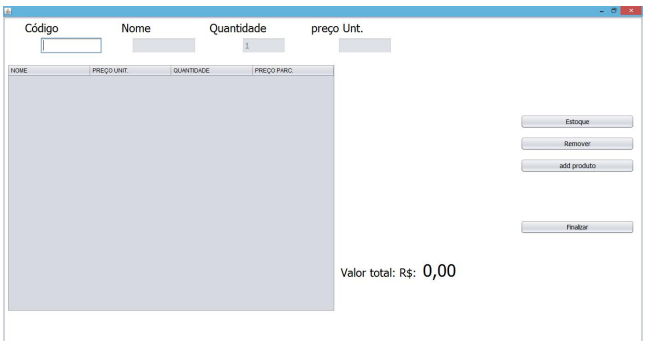
Nessa classe, é informado os atributos privados dos produtos da mercearia, o construtor, que servirá para informar a definição inicial das variáveis e os métodos getter e setter, que irão permitir obter e alterar os valores, respectivamente. Abaixo é possível observar a implementação das informações supracitadas.

```
public class Produto {

    private int codigo;
    private String nome;
    private double preco;
    private int quantidade;

    public Produto(int codigo, String nome, double preco, int quantidade) {
        this.codigo = codigo;
        this.nome = nome;
        this.preco = preco;
        this.quantidade = quantidade;
    }

    public int getCodigo() {
        return codigo;
    }

    public String getNome() {
        return nome;
    }

    public double getPreco() {
        return preco;
    }
```

```
    public void setPreco(double preco) {
        this.preco = preco;
    }

    public int getQuantidade() {
        return quantidade;
    }

    public void setQuantidade(int quantidade) {
        this.quantidade = quantidade;
    }

    @Override
    public String toString() {
        return this.codigo + ";" + this.nome + ";" + this.preco + ";" + this.quantidade;
    }
}
```

- Classe “CompraGUI”:

Nessa classe, temos a construção da interface inicial, citada na introdução deste documento. Mais especificamente, é possível notar mais 4 botões: estoque, remover, add produto e finalizar. Embora suas funções sejam bastante intuitivas, elas serão explicitadas nos métodos que serão comentados posteriormente.



Segue abaixo um print de uma parte do código:

```
import java.util.Vector;
import javax.swing.JOptionPane;
public class CompraGUI extends javax.swing.JPanel {

    //armazena temporariamente produto slecionado
    private Produto produtoSelecionado;
    private ModeloTabelaCompra carrinhoComprasTab;
    private double precoCompra;

    public CompraGUI() {
        initComponents();
        precoCompra = 0.0;
        meuInitComponents();
    }

    private void meuInitComponents(){
        this.carrinhoComprasTab = new ModeloTabelaCompra(this);

        //cria conexão entre tabela usuário
       tabelaCarrinho.setModel(carrinhoComprasTab);

    }
```

Inicialmente, temos a declaração das variáveis que irão armazenar o produto selecionado pelo usuário. Além disso, o construtor inicia com o preço de compra = 0, obviamente.

Segue abaixo a continuação do código:

```
private void codTxtKeyTyped(java.awt.event.KeyEvent evt) {
    //Apenas rodará se a tecla pressionada for enter
    if (evt.getKeyChar() == '\n') {

        //avaliando se existe informação no campo do código
        if (!codTxt.getText().isEmpty()) {
            try {
                //leitura da caixa de texto se texto for int
                int cod = Integer.parseInt(codTxt.getText());

                //efetua-se a busca com o código informado
                produtoSelecionado = FakeBancoDados.consultaProdutoCod(cod);

                //avaliamos a existência de um produto de mesmo código
                if (produtoSelecionado != null) {
                    //atualiza os campos do produto
                    //informações exibidas a caixa de txt = String
                    nomeTxt.setText(produtoSelecionado.getNome());
                    precoUnitTxt.setText(produtoSelecionado.getPreco() + "");
                    quantidadeTxt.setEnabled(true);

                } else {
                    //não cadastrado
                    JOptionPane.showMessageDialog(null, "Produto não cadastrado", "código", JOptionPane.INFORMATION_MESSAGE);
```

Neste momento, é importante salientar que, quando o usuário inicia o software, ao indicar qual é o código do produto, ele precisa dar um enter depois para funcionar. É possível observar isso nas primeiras linhas do código acima. Após isso, temos uma condição, em que será avaliado se há informação no campo de texto que o usuário digitou. Depois dessa verificação, ocorre a leitura do código (do tipo int) e, em seguida, o produto é buscado na classe FakeBancoDados. Em sequência, ocorre a procura desse produto, verificando se ele existe no arquivo ou não. Se a resposta para a condição for sim, o nome do produto, seu preço unitário e sua quantidade aparecerá na interface. Se a resposta for não, uma mensagem informando que o produto não está cadastrado será mostrada.

Em seguida, observe mais uma parte do código:

```
    private void quantidadeTxtKeyReleased(java.awt.event.KeyEvent evt) {
        //irformação de nova quantidade e calculo sobre ela

        //diferente de vazio ocorrerá alteração
        if (!quantidadeTxt.getText().isEmpty()) {
            try{

            //captura de texto quantidade e conversão para inteiro
            int quantidade = Integer.parseInt(quantidadeTxt.getText());

            //cálculo do montante
            double parc = produtoSelecionado.getPreco() * quantidade;
            //representação em duas casas decimais - "2f"
            precoUnitTxt.setText(String.format("%.2f", parc));
            }catch(NumberFormatException ex){
                precoUnitTxt.setText("N/A");
            }
        } else {
            precoUnitTxt.setText("N/A");
        }
    }
```

Nessa parte do código, é mostrado quando o usuário quer mudar a quantidade de produtos que quer levar. Se a caixa de texto de “quantidade” não estiver vazia, a condição acima será “rodada”. Nesse sentido, o número informado é capturado e armazenado e o cálculo do montante é realizado (produto * quantidade).

Segue abaixo mais uma parte do código:

```java
private void addProdutoBtnActionPerformed(java.awt.event.ActionEvent evt) {
    if(produtoSelecionado != null && !precoUnitTxt.getText().equals("N/A")){

        //produto apto a adição
        int quantidade = Integer.parseInt(quantidadeTxt.getText());
            if(quantidade <= produtoSelecionado.getQuantidade()){
                //quantidade do produto é suficiente

                Produto vendido = new Produto(produtoSelecionado.getCodigo(), produtoSelecionado.getNome(), produtoSelecionado.getPreco(), quantidade);

                //produtoSelecionado.setQuantidade(produtoSelecionado.getQuantidade() - quantidade);

                //calculo de total
                precoCompra += produtoSelecionado.getPreco() * quantidade;
                totalTxt.setText(String.format("%.2f", precoCompra));


                //limpando informações para formação de novo produto
                codTxt.setText("");
                quantidadeTxt.setText("1");
                precoUnitTxt.setText("");
                produtoSelecionado = null;
                quantidadeTxt.setEnabled(false);
                nomeTxt.setText("");

                //add produto vendido a tabela
                this.carrinhoComprasTab.addNovoProduto(vendido);
                tabelaCarrinho.updateUI();
            }
            else
            {
                JOptionPane.showMessageDialog(null, "Não há quantidade suficiente do produto", "quantidade insuf.", JOptionPane.INFORMATION_MESSAGE);
```

Esse método é acionado quando o botão “add produto” é clicado. Nele, haverá a verificação se o produto está apto para a adição (produto é diferente de null, por exemplo) e, logo após isso, há uma outra verificação, checando se a quantidade do produto em estoque é suficiente. Se for, o cálculo da compra total vai se feito e o preço aparecerá para o usuário. Finalmente, os campos de textos vão ser limpos para a entrada de um possível novo produto. Se a última verificação citada for falsa, aparecerá uma mensagem informando que não há quantidade de produtos suficientes. Se a primeira verificação for falsa, aparecerá uma mensagem informando que a entrada é inválida.

Veja abaixo mais uma parte do código dessa classe:

```java
private void removerBtnActionPerformed(java.awt.event.ActionEvent evt) {
    int linha = tabelaCarrinho.getSelectedRow();

    if(linha > -1){

        //a linha existe e pode remover produto
        int op = JOptionPane.showConfirmDialog(null, "Deseja remover o produto selecionado", "confirmação de exclusão", JOptionPane.YES_NO_OPTION);

    if(op == JOptionPane.YES_OPTION){
        //com a opção sim selecionada se remove o produto

        String senha = JOptionPane.showInputDialog(null, "informe a senha do gerente", "operação restrita", JOptionPane.INFORMATION_MESSAGE);

        if(senha != null && senha.equals("ifmg")){
        carrinhoComprasTab.removeProdutoCarrinho(linha);
        atualizaQuantidades();
        }
    }

    }else{
        //produto não foi selecionado para remoção
        JOptionPane.showMessageDialog(null, "É preciso selecionar um produto", "Selecionar!!!", JOptionPane.INFORMATION_MESSAGE);
    }


}
```

Esse é o código para o botão “remover”. Nele, o código, primeiramente, verifica se a linha existe. Após isso, pergunta se o usuário tem certeza se quer removê-lo. Se sim, o usuário receberá uma mensagem pedindo a senha e, caso for a correta, o produto será removido e o estoque será atualizado. Se o usuário clicar em uma linha inexistente, uma

mensagem aparecerá informando que uma linha deve ser selecionada para ser removida. Segue abaixo a última parte do código dessa classe que será mostrado:

```
    private void finalizarBtnActionPerformed(java.awt.event.ActionEvent evt) {
        //verificar a existência da quantidade dos produtos vendidos
        if(verificaQuantidades()){
            //atualizar banco de dados fake
        Vector<Produto> produtos = carrinhoComprasTab.produtosCarrinho();

            for(int i = 0; i < produtos.size(); i++){

                Produto estoque = FakeBancoDados.consultaProdutoCod(produtos.get(i).getCodigo());
                estoque.setQuantidade(estoque.getQuantidade() - produtos.get(i).getQuantidade());
            }

        //zerando valor total para prox compra
        precoCompra = 0.0;
        totalTxt.setText("0.0");

        //remover produtos do carrinho pós compra
        carrinhoComprasTab.limpaCarrinho();
        tabelaCarrinho.updateUI();
        }

    }
```

Esse método é acionado quando o botão “finalizar” é clicado. Nele, vai ser verificado, primeiramente, se a quantidade de produtos vendidos existe e, após isso, o banco de dados vai ser atualizado com a compra. Ademais, o estoque de produtos também será atualizado. No final dessa parte do código, o valor da compra é zerado e os produtos do carrinho são removidos.

- Classe “EstoqueGui”:

```
package revisao2;
public class EstoqueGUI extends javax.swing.JPanel {


    public EstoqueGUI() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    Generated Code


    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    // End of variables declaration
}
```

Essa classe não foi implementada.

- Classe “FakeBancoDados”:

```java
private static void cargaArquivo() {
    //ajuste na criação do vetor vindo de inicialização e mudanças
    if (produtos == null) {
        produtos = new Vector<>();
    } else {
        produtos.clear();
    }

    // não se passa a extensão apenas ao manipular diretórios
    File arquivoCsv = new File("C:\\Users\\infin\\OneDrive\\Área de Trabalho\\progII\\Códigos\\produtos.csv");
    try {

        //trava de arquivo para impossibilidade de alteração externa ou apontamento de software.
        FileReader marcaLeitura = new FileReader(arquivoCsv);

        //otimiza a leitura ao acoplar vários bytes em um único envio ao contrário do unitário FileReader
        BufferedReader bufLeitura = new BufferedReader(marcaLeitura);

        // ---------------- leitura das linhas -----------------
        //primeira (cabeçalho) - descartável ->
        bufLeitura.readLine();

        //primeira linha útil (tendo os dados necessários) ->
        String linha = bufLeitura.readLine();
```

Nessas primeiras linhas de código, há a declaração do arquivo de leitura, ou seja, a planilha do Excel que informa os dados dos produtos. Podemos observar, ainda, o BufferedReader faz a leitura linha por linha. Ele, ainda, é otimizado pois acopla vários bytes em um único envio. É importante salientar que, nesse código, o arquivo fica travado para nenhum outro software modifica-lo, o que, depois, é destravado. Veja, em seguida, mais uma parte do código para ser analisado:

```java
            //toda última linha de arquivo é representada como null pela ausência de informação
            while (linha != null) {
                //outras linhas ->
                //corte das cedulas em um array
                String infos[] = linha.split(";");

                //refinamento dos dados vindos do array
                int cod = Integer.parseInt(infos[0]);
                String nome = infos[1];
                double preco = Double.parseDouble(infos[2]);
                int quant = Integer.parseInt(infos[3]);

                //produtos adcionados no vetor
                produtos.add(new Produto(cod, nome, preco, quant));

                //att da variável linha lendo a de baixo para continuidade do cilco até o término
                linha = bufLeitura.readLine();
            }
            //Liberando do arquivo para outros processos
            bufLeitura.close();
            //tratamento de possíveis erros
        } catch (FileNotFoundException ex) {
            System.out.println("Arquivo não existe");
        } catch (IOException e) {
            System.out.println("Arquivo corrompido");
        }
    }
```

Nessa parte do código, o programa está fazendo uma leitura linha a linha, realizando “quebras” sempre que encontra “;”. Essas “quebras” estão sendo armazenadas como atributos, código e nome, por exemplo. Esses atributos são adicionados para instanciar um produto. Note que o BufferedReader está sendo fechado. Veja, abaixo, mais uma parte do código:

```java
public static Produto consultaProdutoCod(int cod) {
    //Se o arquivo ainda não foi carregado, basta carregá-lo
    if (produtos == null) {
        cargaArquivo();
    }

    //busca pelo produto usando o código informado
    for (Produto prodI : produtos) {      //for (int i = 0; i<produtos.size; i++)
        if (prodI.getCodigo() == cod) //if(produtos.get(i).getCodigo() == cod)
        {
            return prodI;
        }
    }
    //não existe produto com o código informado
    return null;
}
```

Nessa parte, o arquivo, caso ainda não tiver sido carregado, vai ser carregado na primeira condição apresentada. Logo abaixo disso, temos o código da busca pelo produto. Em seguida, analise a última parte do código dessa classe:

```java
public static void atualizaArquivo(){
    File arquivo = new File("C:\\Users\\infin\\OneDrive\\Área de Trabalho\\progII\\Códigos\\produtos.csv");

    try {
        FileWriter escritor = new FileWriter(arquivo);

        BufferedWriter bufEscrita = new BufferedWriter(escritor);

        for(int i = 0; i < produtos.size();i++){
            //pela concatenação com uma string java entende a chamada auto do toString.
            bufEscrita.write(produtos.get(i)+"\n");
        }

        bufEscrita.flush();
        bufEscrita.close();

    } catch (IOException ex) {
        System.out.println("Dispositivo com falha");
    }
}
}
```

Nessa parte, é possível observar o código da atualização do arquivo depois de um produto ser comprado, ou seja, está atualizando o estoque.

Classe “ModeloTabelaCompra”:

```java
public class ModeloTabelaCompra extends AbstractTableModel {

    private Vector<Produto> carrinhoCompra;
    private CompraGUI painel;

    public ModeloTabelaCompra(CompraGUI painel){
        this.carrinhoCompra = new Vector<>();
        this.painel = painel;
    }

    @Override
    public int getRowCount() {
        return carrinhoCompra.size();
    }

    @Override
    public int getColumnCount() {
        //nome, preço, quantidade, total
        return 4;
    }
```

Nessa parte do código, há a declaração de variáveis, isto é, do carrinho de compras e do painel. Inclusive, há a presença do construtor, para as variáveis já iniciarem com valor predestinado. Além disso, os métodos getter também estão presentes. Veja, abaixo, mais uma parte do código:

```
@Override
public Object getValueAt(int linha, int coluna) {

    //objeto temporário devido as repetições de chamadas no código
    Produto temp = carrinhoCompra.get(linha);

    switch(coluna){
        case 0: return temp.getNome();
        case 1: return temp.getPreco();
        case 2: return temp.getQuantidade();
        case 3: return temp.getQuantidade() * temp.getPreco();
        default: return null;
    }
}

public void addNovoProduto (Produto vendido){
    this.carrinhoCompra.add(vendido);
}

public void removeProdutoCarrinho(int indice){
    this.carrinhoCompra.remove(indice);
}
```

Aqui, observa-se os valores dos atributos sendo chamados. Abaixo, temos 2 métodos, o de adicionar um novo produto no carrinho e, também, um método de remover produto. Esse switch serve para mostrar os atributos na tabela. Segue abaixo mais uma parte do código:

```
    @Override
    public String getColumnName(int coluna) {
        switch(coluna){
            case 0: return "NOME";
            case 1: return "PREÇO UNIT.";
            case 2: return "QUANTIDADE";
            case 3: return "PREÇO PARC.";
            default: return null;
        }

    }

    @Override
    public boolean isCellEditable(int linha, int coluna) {
        if(coluna == 2){
            return true;
        }else{
            return false;
        }
    }
```

No código acima, é possível ver que o switch possui a finalidade de mostrar os atributos na tabela.

- Classe “JanelaPrincipal”:

```java
public class JanelaPrincipal extends javax.swing.JFrame {

    //barra de rolagem
    private static JScrollPane barraRolagem;
    // auxilia transição dos paineis
    private static JPanel trocaInformacao;
    //gerencia o painel visivel
    private static CardLayout paineisLayout;

    public JanelaPrincipal() {
        initComponents();
        meuInitComponents();
    }

    private void meuInitComponents() {
        //ocupar toda área do frame com a barra de rolagem
        barraRolagem = new JScrollPane();

        //barra de rolagem ocupando todo o espaço
        this.setLayout(new BorderLayout());
        this.add(barraRolagem);

        //painel que receberá todos os demais para posterior refinamento
        //por ser um CardLayout que já se salva com o painel | nome?
        paineisLayout = new CardLayout();
        trocaInformacao = new JPanel(paineisLayout);

        //painel dentro das barras, ele que é o arranjo dos demais
        barraRolagem.setViewportView(trocaInformacao);

        //primeiro painel (antes dos demais serem instanciados):
        trocaInformacao.add(new CompraGUI(), "Compra");
        paineisLayout.show(trocaInformacao, "compra");
```

Nessa classe é perceptível que o gerenciamento de painéis está sendo feito. Nota-se partes do código auxiliando na transição de painéis, arrumando o frame para otimizar código, entre outros. Veja abaixo mais uma parte do código:

```java
//informações necessárias para o método show
public static void efetuaTransicao(JPanel novoPainel, String nome) {
    trocaInformacao.add(novoPainel, nome);
    paineisLayout.show(trocaInformacao, nome);

    //manter as dimensões como a do painel usado
    trocaInformacao.setPreferredSize(novoPainel.getPreferredSize());
}
```

Nessa parte, os painéis são trocadas efetivamente.

3. Conclusão

Por fim, faz-se necessário salientar que tivemos algumas dúvidas durante a implementação do código, porém, por meio da ajuda de professores e amigos, a conclusão foi certeira. Além disso, é importante dizer que, embora muito trabalhoso, o trabalho ajudou a reforçar os conceitos de programação, ou seja, serviu como uma revisão completa. Ademais, o projeto teve uma temática muito interessante e as aulas disponíveis ajudaram demasiadamente, o que facilitou na produção do código.

4. Bibliografia

https://www.devmedia.com.br/metodos-modulos-de-programa-em-java/26771

https://www.devmedia.com.br/vetores-em-java/21449