# Projeto Sistema de Opinião

**disciplina:** Programação Orientada a Objetos em Java

**professor:** Jorge Luís

**Grupo:**

Matheus Ribeiro Scalzer Alves - 202108083887

Camila França Ramos – 202303300701

Davi Nascimento Mattos – 202303559305

---

```java
// Importações necessárias para trabalhar com imagens, interface gráfica, eventos, etc.
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

// Classe abstrata FeedbackButton que extende JButton e implementa ActionListener
abstract class FeedbackButton extends JButton implements ActionListener {
    protected FeedbackGUI feedbackGUI;  // Referência à interface gráfica principal
    protected JLabel countLabel;        // Label para exibir a contagem de feedback
    protected String label;             // Texto do botão
    protected String emoji;             // Emoji do botão
    protected Color color;              // Cor do botão

    // Construtor da classe FeedbackButton
    public FeedbackButton(FeedbackGUI feedbackGUI, JLabel countLabel, String label, String emoji, Color color) {
        this.feedbackGUI = feedbackGUI;
        this.countLabel = countLabel;
        this.label = label;
        this.emoji = emoji;
        this.color = color;
        setupButton();  // Configura o botão
    }

    // Método para configurar o botão
    private void setupButton() {
        setFont(new Font(name:"Segoe UI Emoji", Font.BOLD, size:30));
        setBackground(color);
        setForeground(Color.WHITE);
        setFocusPainted(b:false);
        setPreferredSize(new Dimension(width:400, height:300));
        setOpaque(isOpaque:true);
        setBorderPainted(b:false);

        setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();

        // Adicionando o emoji
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.weighty = 0.6;
        gbc.anchor = GridBagConstraints.CENTER;
        JLabel labelEmoji = new JLabel(emoji, SwingConstants.CENTER);
        labelEmoji.setFont(new Font(name:"Segoe UI Emoji", Font.PLAIN, size:80));
        labelEmoji.setForeground(Color.WHITE);
        add(labelEmoji, gbc);

        // Adicionando o texto
        gbc.gridy = 1;
        gbc.weighty = 0.6;
        JLabel labelText = new JLabel(label, SwingConstants.CENTER);
        labelText.setFont(new Font(name:"Segoe UI", Font.BOLD, size:40));
        labelText.setForeground(Color.WHITE);
        add(labelText, gbc);

```

```
        addActionListener(this);  // Adiciona o ActionListener ao botão
    }

    // Método para atualizar a contagem de feedback
    protected void updateCount() {
        feedbackGUI.incrementCount(label);  // Incrementa a contagem na GUI principal
        feedbackGUI.updateBackgroundColor();  // Atualiza a cor de fundo com base nas contagens
    }

    // Método acionado quando o botão é pressionado
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            String feedbackComment = JOptionPane.showInputDialog(feedbackGUI, message:"Nos forneça um feedback aqui: ", label.toUpperCase());
            if (feedbackComment != null) {
                feedbackGUI.addFeedbackComment(feedbackComment);  // Adiciona o comentário de feedback
                updateCount();  // Atualiza a contagem
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

// Classes para os botões específicos (Bom, Médio, Ruim)
class BomButton extends FeedbackButton {
    public BomButton(FeedbackGUI feedbackGUI, JLabel countLabel) {
        super(feedbackGUI, countLabel, label:"Bom", emoji:"😀", Color.decode(nm:"#4CAF50"));
    }
}

class MedioButton extends FeedbackButton {
    public MedioButton(FeedbackGUI feedbackGUI, JLabel countLabel) {
        super(feedbackGUI, countLabel, label:"Médio", emoji:"😐", Color.decode(nm:"#FFC107"));
    }
}

class RuimButton extends FeedbackButton {
    public RuimButton(FeedbackGUI feedbackGUI, JLabel countLabel) {
        super(feedbackGUI, countLabel, label:"Ruim", emoji:"😞", Color.decode(nm:"#F44336"));
    }
}
```

```
// Classe para contagens de feedback
class FeedbackCounts implements Serializable {
    private static final long serialVersionUID = 1L;
    private int bomCount;
    private int medioCount;
    private int ruimCount;

    public FeedbackCounts() {
        this.bomCount = 0;
        this.medioCount = 0;
        this.ruimCount = 0;
    }

    public int getBomCount() {
        return bomCount;
    }

    public void setBomCount(int bomCount) {
        this.bomCount = bomCount;
    }

    public int getMedioCount() {
        return medioCount;
    }

    public void setMedioCount(int medioCount) {
        this.medioCount = medioCount;
    }

    public int getRuimCount() {
        return ruimCount;
    }

    public void setRuimCount(int ruimCount) {
        this.ruimCount = ruimCount;
    }
}

// Classe para comentários de feedback
class FeedbackComments implements Serializable {
    private static final long serialVersionUID = 1L;
    private List<String> comments;

    public FeedbackComments() {
        this.comments = new ArrayList<>();
    }

    public List<String> getComments() {
        return comments;
    }

    public void addComment(String comment) {
        comments.add(comment);
    }
}
```

```java
// Classe principal FeedbackGUI que estende JFrame
public class FeedbackGUI extends JFrame {
    private JLabel bomLabel;
    private JLabel medioLabel;
    private JLabel ruimLabel;
    private int bomCount;
    private int medioCount;
    private int ruimCount;
    private JPanel panel;
    private DefaultListModel<String> historyModel;
    private FeedbackCounts feedbackCounts;
    private FeedbackComments feedbackComments;

    // Construtor da classe FeedbackGUI
    public FeedbackGUI() {
        setTitle(title:"Sistema de Feedback");
        setSize(width:1200, height:600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(c:null);

        panel = new JPanel(new GridLayout(rows:2, cols:3, hgap:20, vgap:20));
        panel.setBorder(BorderFactory.createEmptyBorder(top:20, left:20, bottom:20, right:20));
        panel.setBackground(Color.WHITE);

        bomLabel = new JLabel("Bom: " + bomCount, SwingConstants.CENTER);
        medioLabel = new JLabel("Médio: " + medioCount, SwingConstants.CENTER);
        ruimLabel = new JLabel("Ruim: " + ruimCount, SwingConstants.CENTER);

        bomLabel.setFont(new Font(name:"Segoe UI", Font.BOLD, size:40));
        medioLabel.setFont(new Font(name:"Segoe UI", Font.BOLD, size:40));
        ruimLabel.setFont(new Font(name:"Segoe UI", Font.BOLD, size:40));

        BomButton bomButton = new BomButton(this, bomLabel);
        MedioButton medioButton = new MedioButton(this, medioLabel);
        RuimButton ruimButton = new RuimButton(this, ruimLabel);

        panel.add(bomButton);
        panel.add(medioButton);
        panel.add(ruimButton);
        panel.add(bomLabel);
        panel.add(medioLabel);
        panel.add(ruimLabel);

        add(panel, BorderLayout.CENTER);

        JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, hgap:20, vgap:10));
        JButton qrButton = new JButton(text:"Mostrar QR Code");
        qrButton.setFont(new Font(name:"Segoe UI", Font.BOLD, size:20));
        qrButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                showQRCode();
            }
        });
        buttonPanel.add(qrButton);
```

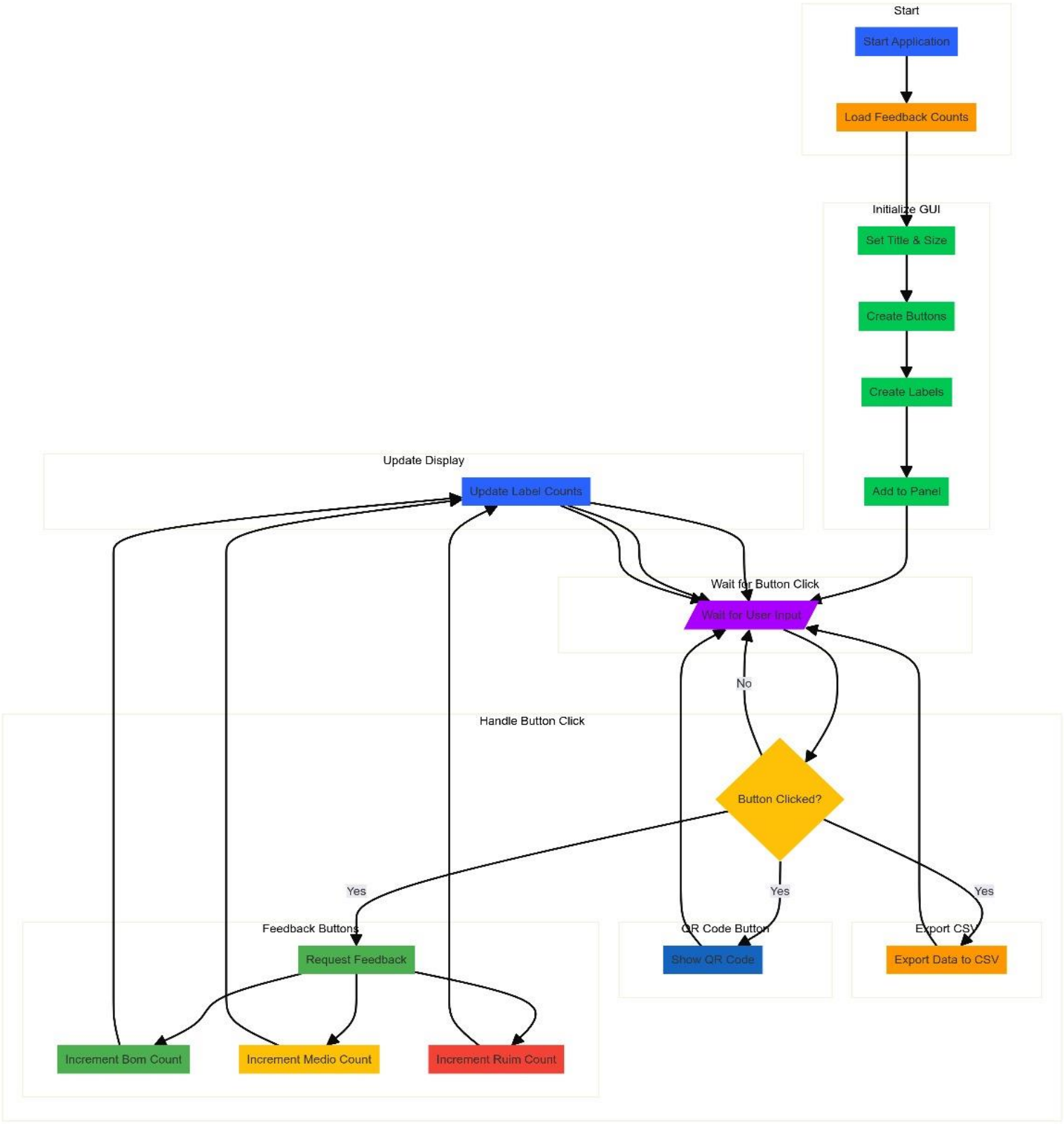
```java
        // Adicionando botão de exportação
        JButton exportButton = new JButton(text:"Exportar Feedbacks para CSV");
        exportButton.setFont(new Font(name:"Segoe UI", Font.BOLD, size:20));
        exportButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                exportFeedbacksToCSV();
            }
        });
        buttonPanel.add(exportButton);

        add(buttonPanel, BorderLayout.NORTH);

        // Adicionando o painel de histórico de feedbacks
        historyModel = new DefaultListModel<>();
        JList<String> historyList = new JList<>(historyModel);
        JScrollPane scrollPane = new JScrollPane(historyList);
        scrollPane.setBorder(BorderFactory.createTitledBorder(title:"Histórico de Feedbacks"));
        add(scrollPane, BorderLayout.EAST);

        // Carregar histórico de feedbacks
        feedbackComments = loadFeedbackComments();
        loadFeedbackHistory();

        // Carregar contagens de feedbacks
        feedbackCounts = loadFeedbackCounts();
        bomCount = feedbackCounts.getBomCount();
        medioCount = feedbackCounts.getMedioCount();
        ruimCount = feedbackCounts.getRuimCount();

        bomLabel.setText("Bom: " + bomCount);
        medioLabel.setText("Médio: " + medioCount);
        ruimLabel.setText("Ruim: " + ruimCount);

        updateBackgroundColor();
    }

    // Método para incrementar a contagem de feedback
    public void incrementCount(String feedbackType) {
        if (feedbackType.equals(anObject:"Bom")) {
            bomCount++;
            feedbackCounts.setBomCount(bomCount);
            bomLabel.setText("Bom: " + bomCount);
        } else if (feedbackType.equals(anObject:"Médio")) {
            medioCount++;
            feedbackCounts.setMedioCount(medioCount);
            medioLabel.setText("Médio: " + medioCount);
        } else if (feedbackType.equals(anObject:"Ruim")) {
            ruimCount++;
            feedbackCounts.setRuimCount(ruimCount);
            ruimLabel.setText("Ruim: " + ruimCount);
        }
        saveFeedbackCounts();  // Salva as contagens de feedback
    }
```

```java
    // Método para atualizar a cor de fundo com base nas contagens de feedback
    public void updateBackgroundColor() {
        if (bomCount > 10 && bomCount > medioCount && bomCount > ruimCount) {
            panel.setBackground(Color.decode(nm:"#C8E6C9")); // Verde claro
        } else if (medioCount > 10 && medioCount > bomCount && medioCount > ruimCount) {
            panel.setBackground(Color.decode(nm:"#FFF9C4")); // Amarelo claro
        } else if (ruimCount > 10 && ruimCount > bomCount && ruimCount > medioCount) {
            panel.setBackground(Color.decode(nm:"#FFCDD2")); // Vermelho claro
        } else {
            panel.setBackground(Color.WHITE); // Branco
        }
    }

    // Método para adicionar um comentário de feedback
    public void addFeedbackComment(String comment) {
        feedbackComments.addComment(comment);
        saveFeedbackComments();  // Salva os comentários de feedback
        addFeedbackToHistory(comment);  // Adiciona o comentário ao histórico
    }

    // Método para carregar o histórico de feedbacks
    public void loadFeedbackHistory() {
        if (feedbackComments != null) {
            for (String comment : feedbackComments.getComments()) {
                historyModel.addElement(comment);
            }
        }
    }

    // Método para exportar os feedbacks para um arquivo CSV
    public void exportFeedbacksToCSV() {
        try (PrintWriter writer = new PrintWriter(new File(pathname:"feedbacks.csv"))) {
            StringBuilder sb = new StringBuilder();
            sb.append(str:"Tipo");
            sb.append(c:',');
            sb.append(str:"Quantidade");
            sb.append(c:'\n');

            sb.append(str:"Bom");
            sb.append(c:',');
            sb.append(bomCount);
            sb.append(c:'\n');

            sb.append(str:"Médio");
            sb.append(c:',');
            sb.append(medioCount);
            sb.append(c:'\n');

            sb.append(str:"Ruim");
            sb.append(c:',');
            sb.append(ruimCount);
            sb.append(c:'\n');

            writer.write(sb.toString());
            JOptionPane.showMessageDialog(this, message:"Feedbacks exportados com sucesso!");
        } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(this, message:"Erro ao exportar feedbacks!");
            e.printStackTrace();
        }
    }
```

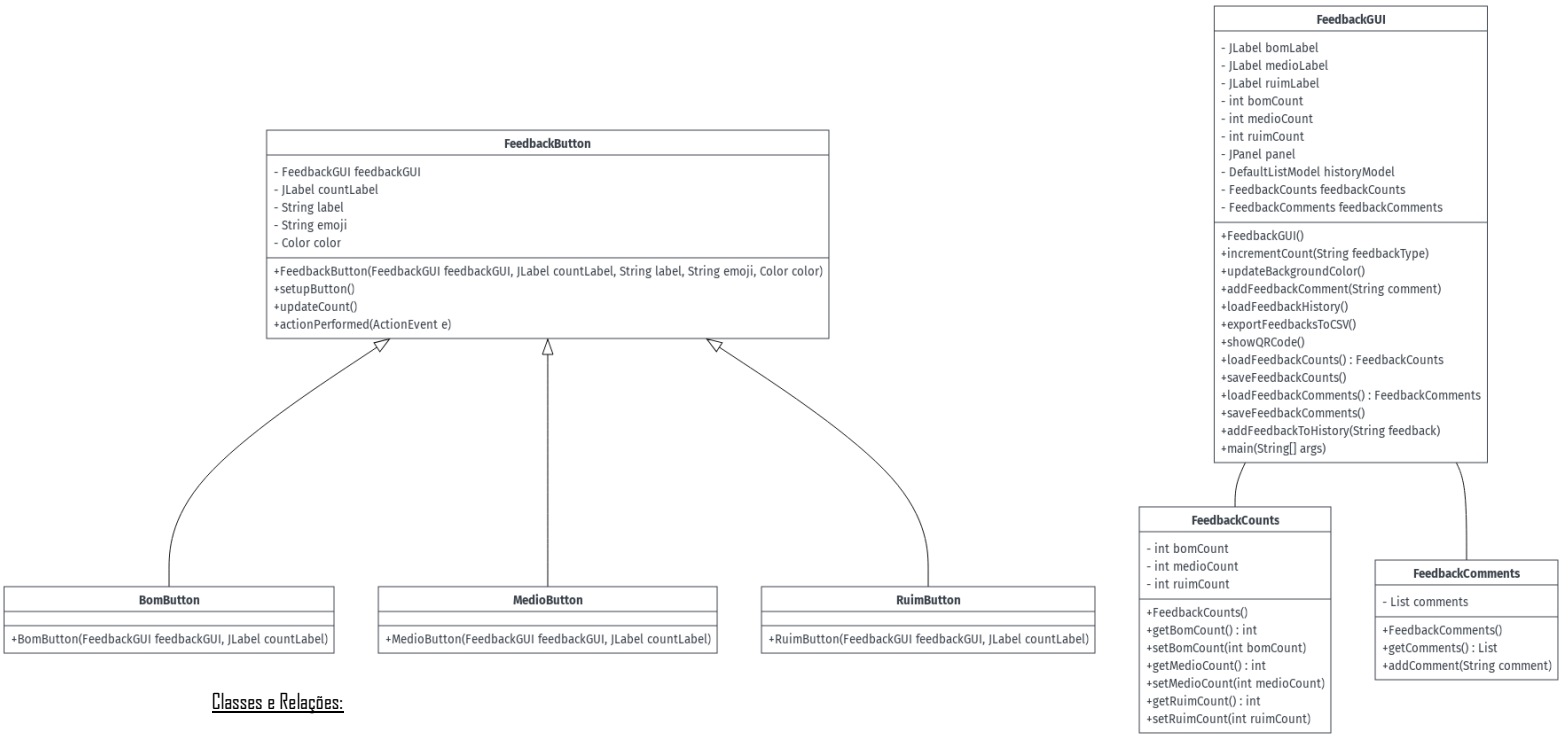
```java

    // Método para mostrar o QR Code
    public void showQRCode() {
        try {
            BufferedImage originalImage = ImageIO.read(getClass().getResource(name:"qrcode.png"));
            int maxWidth = 200;
            int maxHeight = 200;
            int newWidth = originalImage.getWidth();
            int newHeight = originalImage.getHeight();
            if (newWidth > maxWidth || newHeight > maxHeight) {
                double aspectRatio = (double) newWidth / newHeight;
                if (newWidth > maxWidth) {
                    newWidth = maxWidth;
                    newHeight = (int) (newWidth / aspectRatio);
                }
                if (newHeight > maxHeight) {
                    newHeight = maxHeight;
                    newWidth = (int) (newHeight * aspectRatio);
                }
            }
            Image scaledImage = originalImage.getScaledInstance(newWidth, newHeight, Image.SCALE_SMOOTH);
            ImageIcon icon = new ImageIcon(scaledImage);
            Font font = new Font(name:"Arial", Font.BOLD, size:18);
            UIManager.put(key:"OptionPane.messageFont", font);
            UIManager.put(key:"OptionPane.buttonFont", font);
            JOptionPane.showMessageDialog(this, message:"Nos forneça um feedback construtivo em nosso site", title:"QR CODE", JOptionPane.INFORMATION_MESSAGE, icon);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    // Método para carregar as contagens de feedbacks
    public FeedbackCounts loadFeedbackCounts() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(name:"feedback_counts.dat"))) {
            return (FeedbackCounts) ois.readObject();
        } catch (Exception e) {
            return new FeedbackCounts();
        }
    }

    // Método para salvar as contagens de feedbacks
    public void saveFeedbackCounts() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(name:"feedback_counts.dat"))) {
            oos.writeObject(feedbackCounts);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Método para carregar os comentários de feedbacks
    public FeedbackComments loadFeedbackComments() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(name:"feedback_comments.dat"))) {
            return (FeedbackComments) ois.readObject();
        } catch (Exception e) {
            return new FeedbackComments();
        }
    }

```

```
    // Método para salvar os comentários de feedbacks
    public void saveFeedbackComments() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(name:"feedback_comments.dat"))) {
            oos.writeObject(feedbackComments);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Método para adicionar feedback ao histórico
    public void addFeedbackToHistory(String feedback) {
        historyModel.addElement(feedback + " - " + java.time.LocalDateTime.now());
    }

    // Método principal para iniciar a aplicação
    Run main | Debug main
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                try {
                    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
                new FeedbackGUI().setVisible(b:true);
            }
        });
    }
}

```

UML de Uso:


Start
Start Application
Load Feedback Counts
Initialize GUI
Set Title & Size
Create Buttons
Create Labels
Add to Panel
Update Display
Update Label Counts
Wait for Button Click
Wait for User Input
No
Handle Button Click
Button Clicked?
Yes
Yes
Yes
Feedback Buttons
Request Feedback
QR Code Button
Show QR Code
Export CSV
Export Data to CSV
Increment Bom Count
Increment Medio Count
Increment Ruim Count

# UML de classe:




Classes e Relações:

**1. Classe FeedbackButton**

**Atributos:**

FeedbackGUI feedbackGUI: Referência à interface gráfica principal.

JLabel countLabel: Label para exibir a contagem de feedback.

String label: Texto do botão.

String emoji: Emoji do botão.

Color color: Cor do botão.

**Métodos:**

FeedbackButton(FeedbackGUI feedbackGUI, JLabel countLabel, String label, String emoji, Color color): Construtor da classe.

setupButton(): Método para configurar o botão.

updateCount(): Método para atualizar a contagem de feedback.

actionPerformed(ActionEvent e): Método acionado quando o botão é pressionado.

**2. Classe BomButton**

**Métodos:**

BomButton(FeedbackGUI feedbackGUI, JLabel countLabel): Construtor específico para o botão "Bom".

**Herança:**

BomButton herda de FeedbackButton.

**3. Classe MedioButton**

**Métodos:**

MedioButton(FeedbackGUI feedbackGUI, JLabel countLabel): Construtor específico para o botão "Médio".

**Herança:**

MedioButton herda de FeedbackButton.

**4. Classe RuimButton**

**Métodos:**

RuimButton(FeedbackGUI feedbackGUI, JLabel countLabel): Construtor específico para o botão "Ruim".

**Herança:**

RuimButton herda de FeedbackButton.

**5. Classe FeedbackCounts**

**Atributos:**

int bomCount: Contagem de feedback "Bom".

int medioCount: Contagem de feedback "Médio".

int ruimCount: Contagem de feedback "Ruim".

**Métodos:**

FeedbackCounts(): Construtor da classe.

getBomCount(): Retorna a contagem de feedback "Bom".

setBomCount(int bomCount): Define a contagem de feedback "Bom".

getMedioCount(): Retorna a contagem de feedback "Médio".

setMedioCount(int medioCount): Define a contagem de feedback "Médio".

getRuimCount(): Retorna a contagem de feedback "Ruim".

setRuimCount(int ruimCount): Define a contagem de feedback "Ruim".

**6. Classe FeedbackComments**

**Atributos:**

List<String> comments: Lista de comentários.

**Métodos:**

FeedbackComments(): Construtor da classe.

getComments(): Retorna a lista de comentários.

addComment(String comment): Adiciona um comentário à lista.

**7. Classe FeedbackGUI**

**Atributos:**

JLabel bomLabel: Label para a contagem de feedback "Bom".

JLabel medioLabel: Label para a contagem de feedback "Médio".

JLabel ruimLabel: Label para a contagem de feedback "Ruim".

int bomCount: Contagem de feedback "Bom".

int medioCount: Contagem de feedback "Médio".

int ruimCount: Contagem de feedback "Ruim".

JPanel panel: Painel principal.

DefaultListModel<String> historyModel: Modelo para a lista de histórico de feedbacks.

FeedbackCounts feedbackCounts: Objeto para gerenciar contagens de feedback.

FeedbackComments feedbackComments: Objeto para gerenciar comentários de feedback.

**Métodos:**

FeedbackGUI(): Construtor da classe.

incrementCount(String feedbackType): Incrementa a contagem de feedback.

updateBackgroundColor(): Atualiza a cor de fundo com base nas contagens de feedback.

addFeedbackComment(String comment): Adiciona um comentário de feedback.

loadFeedbackHistory(): Carrega o histórico de feedbacks.

exportFeedbacksToCSV(): Exporta os feedbacks para um arquivo CSV.

showQRCode(): Mostra um QR Code.

loadFeedbackCounts(): Carrega as contagens de feedbacks.

saveFeedbackCounts(): Salva as contagens de feedbacks.

loadFeedbackComments(): Carrega os comentários de feedbacks.

saveFeedbackComments(): Salva os comentários de feedbacks.

addFeedbackToHistory(String feedback): Adiciona feedback ao histórico.

main(String[] args): Método principal para iniciar a aplicação.

**Relações:**

FeedbackButton é a classe pai de BomButton, MedioButton e RuimButton.

FeedbackGUI possui associações com FeedbackCounts e FeedbackComments.

Em resumo nosso projeto para este trabalho se trata de uma interface de sistema de opinião.

---

Link do Github para o arquivo completo:

Github: https://github.com/scalzr/Sistema-de-opiniao-2.0