

Interfaces Gráficas em Java: Layouts, Eventos e Componentes

Construindo aplicações desktop com Java Swing

1. Introdução

Além de aplicações de linha de comando, o Java permite construir **interfaces gráficas de usuário** (GUIs) por meio da biblioteca **Swing** (pacote `javax.swing`), que oferece janelas, botões, campos de texto e diversos outros componentes visuais. Este texto aborda os três pilares de uma interface gráfica em Java: os **componentes** (o que é exibido), os **layouts** (como os componentes são organizados na tela) e os **eventos** (como a aplicação reage às ações do usuário).

2. Componentes Básicos

Toda interface Swing começa por uma janela principal, geralmente representada pela classe `JFrame`. Dentro dela, são adicionados diversos **componentes**, elementos visuais com os quais o usuário interage.

Componente	Descrição
<code>JFrame</code>	Janela principal da aplicação.
<code>JPanel</code>	Painel usado para agrupar e organizar outros componentes.
<code>JLabel</code>	Exibe um texto ou imagem, sem interação.
<code>JButton</code>	Botão clicável que dispara uma ação.
<code>TextField</code>	Campo de entrada de texto em uma linha.
<code>TextArea</code>	Campo de entrada de texto em múltiplas linhas.
<code>CheckBox</code>	Caixa de seleção (marcado/desmarcado).
<code>ComboBox</code>	Lista suspensa de opções.

```
import javax.swing.*;

public class JanelaSimples {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Minha Primeira Janela");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 150);

        JLabel label = new JLabel("Olá, Swing!");
        frame.add(label); // adiciona o componente à janela

        frame.setVisible(true); // torna a janela visível
    }
}
```

3. Gerenciadores de Layout

Um **gerenciador de layout** (*LayoutManager*) define como os componentes adicionados a um contêiner (como um `JFrame` ou `JPanel`) são posicionados e redimensionados na tela. Os mais usados são `FlowLayout`, `BorderLayout` e `GridLayout`.

Layout	Comportamento
<code>FlowLayout</code>	Posiciona os componentes em linha, um ao lado do outro, quebrando para a próxima linha quando necessário.
<code>BorderLayout</code>	Divide o contêiner em 5 regiões: NORTH, SOUTH, EAST, WEST e CENTER.
<code>GridLayout</code>	Organiza os componentes em uma grade de linhas e colunas de tamanho igual.
<code>BoxLayout</code>	Organiza os componentes em uma única linha ou coluna, respeitando seu tamanho preferido.

3.1. FlowLayout

```
import javax.swing.*;
import java.awt.FlowLayout;

JFrame frame = new JFrame("FlowLayout");
frame.setLayout(new FlowLayout());

frame.add(new JButton("Botão 1"));
frame.add(new JButton("Botão 2"));
frame.add(new JButton("Botão 3"));
// Os botões aparecem lado a lado, na ordem em que foram adicionados
```

3.2. BorderLayout

```
import javax.swing.*;
import java.awt.BorderLayout;

JFrame frame = new JFrame("BorderLayout");
frame.setLayout(new BorderLayout());

frame.add(new JButton("Topo"), BorderLayout.NORTH);
frame.add(new JButton("Base"), BorderLayout.SOUTH);
frame.add(new JButton("Esquerda"), BorderLayout.WEST);
frame.add(new JButton("Direita"), BorderLayout.EAST);
frame.add(new JButton("Centro"), BorderLayout.CENTER);
```

3.3. GridLayout

```
import javax.swing.*;
import java.awt.GridLayout;

JFrame frame = new JFrame("GridLayout");
frame.setLayout(new GridLayout(2, 3)); // 2 linhas, 3 colunas

for (int i = 1; i <= 6; i++) {
    frame.add(new JButton("Botão " + i));
}
// Os 6 botões são distribuídos em uma grade de tamanho igual
```

4. Tratamento de Eventos

Uma interface gráfica só é útil se reagir às ações do usuário — cliques, digitação, seleções. Em Swing, isso é feito com o padrão **Listener**: uma classe (ou expressão lambda) que implementa uma interface de evento, como `ActionListener`, e é registrada em um componente por meio de um método `addXxxListener`.

```
import javax.swing.*;
import java.awt.event.ActionListener;

JFrame frame = new JFrame("Evento de Clique");
frame.setLayout(new java.awt.FlowLayout());

JButton botao = new JButton("Clique aqui");
JLabel label = new JLabel("Aguardando clique...");

// Usando uma expressão lambda para tratar o evento
botao.addActionListener(e -> label.setText("Botão clicado!"));

frame.add(botao);
frame.add(label);
```

A interface `ActionListener` possui um único método abstrato, `actionPerformed(ActionEvent e)`, o que a torna uma **interface funcional** — por isso pode ser implementada de forma resumida com uma expressão lambda, como no exemplo acima.

4.1. Outros Tipos de Eventos

```
import javax.swing.*;
import javax.swing.event.*;

JTextField campo = new JTextField(15);

// Evento disparado a cada tecla digitada
campo.addKeyListener(new KeyAdapter() {
    @Override
    public void keyReleased(KeyEvent e) {
        System.out.println("Texto atual: " + campo.getText());
    }
});

JCheckBox checkbox = new JCheckBox("Aceito os termos");
checkbox.addItemListener(e -> {
    boolean marcado = checkbox.isSelected();
    System.out.println("Termos aceitos: " + marcado);
});
```

5. Exemplo Integrado: Formulário de Cadastro

O exemplo a seguir combina componentes, o layout `BorderLayout` (com um `JPanel` interno usando `GridLayout`) e tratamento de eventos em um pequeno formulário de cadastro de nomes.

```

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;

public class FormularioCadastro {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(FormularioCadastro::criarInterface);
    }

    private static void criarInterface() {
        List<String> nomes = new ArrayList<>();

        JFrame frame = new JFrame("Cadastro de Nomes");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout(10, 10));

        JPanel painelFormulario = new JPanel(new GridLayout(2, 2, 5, 5));
        JTextField campoNome = new JTextField();
        JButton botaoAdicionar = new JButton("Adicionar");

        painelFormulario.add(new JLabel("Nome:"));
        painelFormulario.add(campoNome);

        JTextArea areaLista = new JTextArea(8, 20);
        areaLista.setEditable(false);

        botaoAdicionar.addActionListener(e -> {
            String nome = campoNome.getText().trim();
            if (!nome.isEmpty()) {
                nomes.add(nome);
                areaLista.setText(String.join("\n", nomes));
                campoNome.setText("");
            }
        });

        frame.add(painelFormulario, BorderLayout.NORTH);
        frame.add(new JScrollPane(areaLista), BorderLayout.CENTER);
        frame.add(botaoAdicionar, BorderLayout.SOUTH);

        frame.pack();
        frame.setVisible(true);
    }
}

```

*Note o uso de `SwingUtilities.invokeLater()`: interfaces `Swing` devem ser criadas e manipuladas na **Event Dispatch Thread (EDT)**, a thread responsável por desenhar a interface e processar eventos, evitando problemas de concorrência.*

6. Boas Práticas

- Sempre inicie a interface dentro de `SwingUtilities.invokeLater()` para garantir que ela seja construída na Event Dispatch Thread.
- Prefira combinar múltiplos `JPanel` com layouts diferentes a tentar resolver todo o posicionamento com um único layout complexo.
- Use expressões lambda para tratar eventos simples, reservando classes separadas apenas para lógicas de evento mais elaboradas.

- Separe a lógica de negócio da lógica de interface, evitando colocar regras complexas diretamente dentro dos métodos de evento.

7. Conclusão

Construir uma interface gráfica em Java com Swing envolve combinar três elementos: os **componentes** que aparecem na tela, os **layouts** que organizam esses componentes de forma consistente, e os **eventos** que conectam as ações do usuário ao comportamento da aplicação. Como visto no exemplo do formulário de cadastro, mesmo aplicações simples já se beneficiam dessa separação, formando a base para o desenvolvimento de interfaces desktop mais complexas em Java.