✅ Pontos-chave do exemplo com GUI e Threads:

1. **A GUI (janela com botão e campo de texto) está na classe `Cronometro`.**
2. Ao clicar em **"Iniciar"**, uma nova **Thread** começa a executar, atualizando o campo de texto com a hora a cada segundo.
3. Ao clicar em **"Terminar"**, a thread é interrompida.
4. Para atualizar a GUI dentro da thread, é usada uma **classe interna anônima** que implementa a interface `Runnable`.
5. Isso permite **acessar os componentes da interface gráfica diretamente** (como a caixa de texto) **sem quebrar o encapsulamento**.

💡 O que é uma classe interna anônima?

É uma classe que **não tem nome** e que é **instanciada e implementada no mesmo lugar** onde é usada. Exemplo:

```
new Runnable() {
    public void run() {
        // Código da thread aqui
    }
}
```

🧪 Exemplo completo (versão simplificada do `Cronometro`)

```
import javax.swing.*;
import java.awt.event.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class CronometroGUI extends JFrame {
    private JTextField campoHora;
    private JButton botaoIniciar, botaoTerminar;
    private Thread threadRelogio;

    public CronometroGUI() {
        setTitle("Cronômetro");
        setSize(300, 100);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
        campoHora = new JTextField(20);
        campoHora.setEditable(false);

        botaoIniciar = new JButton("Iniciar");
        botaoTerminar = new JButton("Terminar");

        JPanel painel = new JPanel();
        painel.add(campoHora);
        painel.add(botaoIniciar);
        painel.add(botaoTerminar);

        add(painel);

        botaoIniciar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                threadRelogio = new Thread(new Runnable() {
                    public void run() {
                        while (!Thread.currentThread().isInterrupted()) {
                            String horaAtual = new SimpleDateFormat("HH:mm:ss").format(new
Date());
                            campoHora.setText(horaAtual);
                            try {
                                Thread.sleep(1000);
                            } catch (InterruptedException ex) {
                                Thread.currentThread().interrupt();
                            }
                        }
                    }
                });
                threadRelogio.start();
            }
        });

        botaoTerminar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (threadRelogio != null) {
                    threadRelogio.interrupt();
                }
            }
        });
    }

    public static void main(String[] args) {
        new CronometroGUI().setVisible(true);
    }
}
```

✅ Vantagens desse modelo:

- Permite manipular a GUI dentro da thread sem precisar criar variáveis públicas ou getters desnecessários.
- Mantém o encapsulamento e a organização da orientação a objetos.
- Código mais conciso usando classes anônimas.

✅ Resumo dos conceitos apresentados:

📌 1. Interface gráfica (GUI) em Java

- Criamos uma interface gráfica estendendo a classe `JFrame`.
- Os componentes visuais (como botão, texto, etc.) vêm de classes do Swing: `JLabel`, `JTextField`, `JButton`, `JComboBox`, `JPanel` etc.

📌 2. Painel e Layout

- O `JPanel` serve como área para organizar os elementos gráficos.
- Usar `setLayout(null)` permite posicionar manualmente os componentes com `setBounds(x, y, width, height)`.

📌 3. WindowBuilder

- É uma excelente ferramenta visual para criar GUIs no Eclipse (ajuda muito quem prefere arrastar e soltar componentes).

🧪 Exemplo prático (baseado nos códigos 53, 54 e 55)

```
import javax.swing.*;

public class Programa extends JFrame {

    // Componentes (Código 54)
    private JPanel painel;
    private JLabel rotulo;
```

```
    private JTextField campoTexto;
    private JComboBox<String> listaSuspensa;
    private JButton botao;

    // Construtor (Código 55)
    public Programa() {
        setTitle("Exemplo GUI");
        setBounds(100, 100, 400, 250); // x, y, width, height
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        painel = new JPanel();
        painel.setLayout(null); // posicionamento manual
        add(painel);

        rotulo = new JLabel("Nome:");
        rotulo.setBounds(10, 20, 100, 25);
        painel.add(rotulo);

        campoTexto = new JTextField();
        campoTexto.setBounds(120, 20, 200, 25);
        painel.add(campoTexto);

        listaSuspensa = new JComboBox<>(new String[]{"Opção 1", "Opção 2", "Opção 3"});
        listaSuspensa.setBounds(120, 60, 200, 25);
        painel.add(listaSuspensa);

        botao = new JButton("Enviar");
        botao.setBounds(120, 100, 100, 30);
        painel.add(botao);
    }

    // Método principal para executar
    public static void main(String[] args) {
        Programa janela = new Programa();
        janela.setVisible(true);
    }
}
```

✅ Resultado:

Ao rodar o código acima:

- Uma janela se abre.
- Ela exibe um rótulo “Nome:”, uma caixa de texto, uma lista suspensa com 3 opções e um botão "Enviar".

✅ Resumo e Explicação dos Conceitos

🧩 Eventos GUI

- O Java Swing é **orientado a eventos**: os componentes disparam eventos quando algo acontece (clique, tecla pressionada, etc.).
- Para reagir a esses eventos, usamos **interfaces chamadas "listeners"**, como:
    - `ActionListener` (para cliques de botão)
    - `KeyListener` (para teclas)
    - `MouseListener` (para ações com o mouse)
    - ... e muitos outros.

🧩 addXXXListener

- Os componentes Swing têm métodos como `addActionListener()`, `addKeyListener()`, etc.
- Você passa uma **implementação da interface de evento** como argumento.

🧩 Classe interna anônima

É uma forma rápida de criar o comportamento para o evento *dentro* do próprio método, sem declarar uma classe separada. Exemplo:

```
botao.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // comportamento ao clicar no botão
    }
});
```

🧪 Exemplo prático com `ActionListener` (código 56 adaptado)

Vamos adicionar o tratamento de evento ao botão "Enviar" que criamos anteriormente:

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

```
public class Programa extends JFrame {

    private JPanel painel;
    private JLabel rotulo;
    private JTextField campoTexto;
    private JComboBox<String> listaSuspensa;
    private JButton botao;

    public Programa() {
        setTitle("Exemplo GUI com Evento");
        setBounds(100, 100, 400, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        painel = new JPanel();
        painel.setLayout(null);
        add(painel);

        rotulo = new JLabel("Nome:");
        rotulo.setBounds(10, 20, 100, 25);
        painel.add(rotulo);

        campoTexto = new JTextField();
        campoTexto.setBounds(120, 20, 200, 25);
        painel.add(campoTexto);

        listaSuspensa = new JComboBox<>(new String[]{"Opção 1", "Opção 2", "Opção 3"});
        listaSuspensa.setBounds(120, 60, 200, 25);
        painel.add(listaSuspensa);

        botao = new JButton("Enviar");
        botao.setBounds(120, 100, 100, 30);
        painel.add(botao);

        // CÓDIGO 56: Evento de clique no botão
        botao.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String nome = campoTexto.getText();
                String opcao = (String) listaSuspensa.getSelectedItem();
                JOptionPane.showMessageDialog(null, "Nome: " + nome + "\nOpção: " + opcao);
            }
        });
    }

    public static void main(String[] args) {
        Programa janela = new Programa();
        janela.setVisible(true);
```

```
    }
}
```

✅ Resultado:

- Quando o usuário digita algo e clica em “Enviar”, aparece uma janela mostrando o conteúdo digitado e a opção selecionada.