Módulo 7

Interface Gráfica com o Usuário GUI - Introdução

Programação Orientada a Objetos I

Java

(Rone Ilídio)

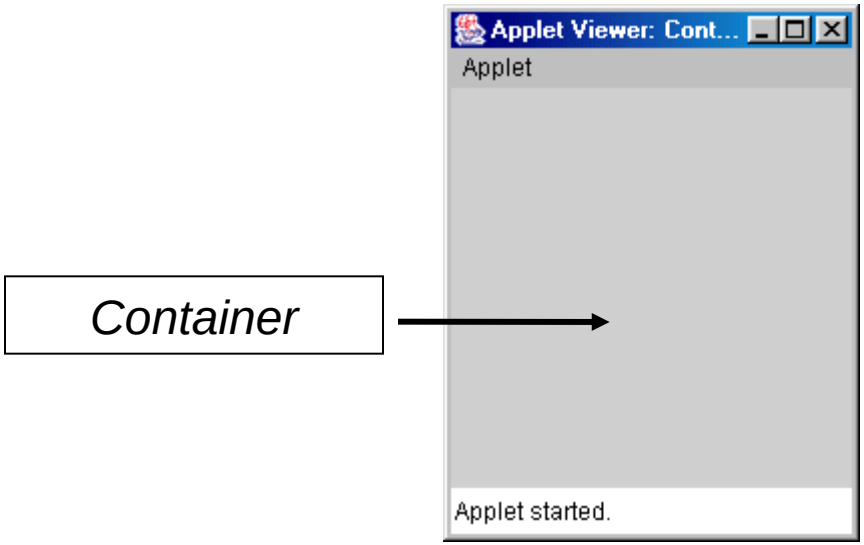
Interface Gráfica com o Usuário

- Os elementos gráficos utilizados estão no pacote javax.swing.*
- As interfaces serão construídas nos objetos *Containers* (painéis de conteúdo) contidos nos *JFrames* e nos *JApplets*
- Um objeto *Container* é capaz de receber e exibir na tela componentes da interface gráfica com o usuário
- Ele está no pacote *java.awt.*, mas não precisa ser importado, pois é utilizado indiretamente pelas classes JFrames* e *JApplets*

Interface Gráfica com o Usuário

```
import javax.swing.*;
public class ContainerApplet extends JApplet{
    public void init (){
    }
}
```

- Temos um *applet* vazio. Sua “área de trabalho” é um *Container*



Interface Gráfica com o Usuário

```
import javax.swing.*;
    public class Aplicativo extends JFrame{
    public Aplicativo(){
        getContentPane().setLayout(null);
        setBounds(10,10,300,300);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]){
        Aplicativo window = new Aplicativo();
    }
}
```

- Na execução aparecerá um janela padrão de aplicativo do Sistema Operacional

Interface Gráfica com o Usuário

- Neste exemplo, temos um aplicativo que recebe por herança as características de *JFrame*
- getContentPane().setLayout(null); → define o gerenciador de Layout; quando “null” todos os componentes devem ter o setBounds (inclusive a janela).
- setBounds(10,10,300,300)→ posiciona (10,10) e dimensiona a janela (300,300)
- setVisible(**true**); deixa a tela visível para o usuário
- setDefaultCloseOperation(JFrame.*EXIT_ON_CLOSE)* → *retira o programa da memória*

Interface Gráfica com o Usuário

- Elementos da GUI (*Graphical User Interface*) são objetos de classes já definidas dentro nos pacotes Java, os quais são adicionadas aos *Containers*.
- Estudaremos inicialmente 2 componentes:
  - JButton → botão
  - JTextField → caixa de texto

Interface Gráfica com o Usuário

Passos para criação de uma interface gráfica:

1) Criar a variável que representará o elemento gráfico
2) Configurar o gerenciador de layout
3) Criar o objeto do elemento gráfico
4) Configurar o objeto
5) Adicioná-lo ao Container
6) Para cada elemento gráfico esses passos devem ser executados, com exceção do passo 2
7) Configurar a tela (só aplicativos)

Obs: usualmente, os passos de 2 a 6 são executados no método construtor para aplicativos e no *init* para *applets*

```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  public class ContainerAplicativo extends JFrame{
4.      JButton botao;          //variavel para um botão
5.      JTextField caixatexto;  //variavel para uma caixa de texto
6.      public ContainerAplicativo(){
7.          getContentPane().setLayout(null); //configura o Container
8.          botao = new JButton();       // criação do objeto
9.          botao.setText("OK");         // configuração do objeto
10.         botao.setBounds(60,50,80,30);
11.         add(botao);                  //adicionando o objeto ao container
12.         caixatexto = new JTextField();     // criação do objeto
13.         caixatexto.setText("Texto de dentro"); // configuração do objeto
14.         caixatexto.setBounds(25,10,150,30);
15.         add(caixatexto); //adicionando o objeto ao container
16.         setBounds(20,20, 200,150);
17.         setVisible(true);
18.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.     }
20.     public static void main(String args[]){
21.         ContainerAplicativo janela = new ContainerAplicativo();
22.     }
23. }
```

Interface Gráfica com o Usuário

- 4 e 5 → criam as variáveis referentes aos elementos gráficos (Passso 1)
- 7 → gerenciador de layout = null (Passso 2)
- 8 → cria o objeto botão (Passso 3)
- 9-10 → configura o objeto botão (Passso 4)
- 11 → adiciona o botão à tela (Passso 5)
- 12 a 14 → idem 9 a 11, mas para a caixa de texto (Passo 6)
- 16 a 18 → configuram a tela (Passso 7)

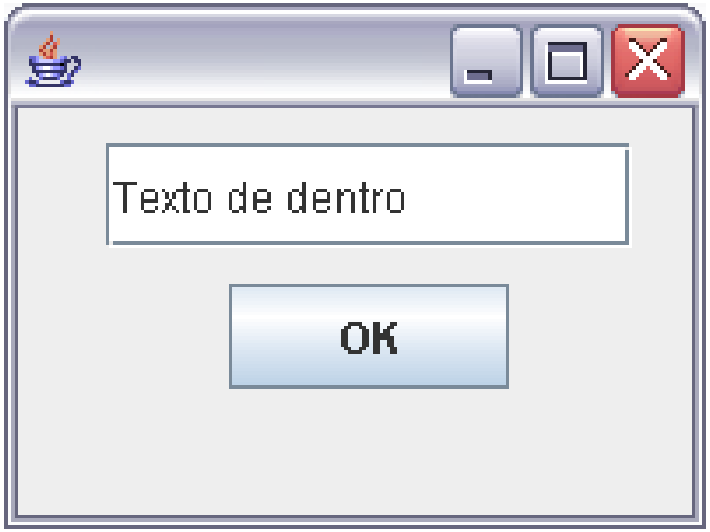
Interface Gráfica com o Usuário

Observações:

- Linha 7→ o método *getContentPane()* retorna uma referência para o *Container* do *JApplet* ou do *JFrame*
- Ele é implementado nessas classe e herdado pelas filhas
- Linha 8 → o métodos setLayout do *Container* determina a forma de inserção dos elementos gráficos.

Interface Gráfica com o Usuário

Resultado da execução:



Interface Gráfica com o Usuário

- A seguir, o mesmo programa em forma de *applet*
- As instruções do método construtor passam para o método *init*
- Não é necessário a configuração da tela.
- O tamanho da tela é configurado no arquivo .html
- O restante é igual!

```
import javax.swing.*;
import java.awt.*;
public class ContainerApplet extends JApplet{
    JButton botao; //variavel para um botão
    JTextField caixatexto; //variavel para uma caixa de texto
    public void init(){
        getContentPane().setLayout( null); //configura o Container

        botao = new JButton(); // criação do objeto
        botao.setText("OK"); // configuração do objeto
          botao.setBounds(60,50,80,30);
          add(botao); //adicionando o objeto ao container

        caixatexto = new JTextField(20); // criação do objeto
        caixatexto.setText("Texto de dentro"); // configuração do
    objeto
          caixatexto.setBounds(25,10,150,30);
         add(caixatexto); //adicionando o objeto ao container
          setBounds(20,20, 200,150);
          setVisible(true);
    }
}
```

Interface Gráfica com o Usuário

Resultado da execução com *Appletviewer*:

Modelo de Tratamento de Eventos

- Eventos são “acontecimentos” que ocorrem com programas, exemplos:
  - Usuário pressionou o botão direito do *mouse* sobre um componente
  - Usuário largou o botão
  - O programa iniciou
  - O programa terminou
  - O *mouse* moveu sobre um componente
  - O usuário digitou algo em um campo da tela
  - O usuário selecionou um item no menu

Modelo de Tratamento de Eventos

- Os eventos são tratados por objetos denominados ***listener***s
- Existe objetos ***listener***s para tratar eventos do mouse, outros para tratar eventos do teclado, etc
- O objeto ***listener*** que utilizaremos trata o clique do mouse sobre um determinado elemento
- As classes desses objetos estão no pacote java.awt.event.*;
- *Segue exemplo*

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;
public class ComEvento extends JFrame{
	JButton botao;
	JTextField caixatexto;
	public ComEvento(){
		getContentPane().setLayout(null);
		Handler objetolistener = new Handler();
		caixatexto = new JTextField();
		caixatexto.setText("Texto de dentro");
		caixatexto.setBounds(10,10,150,30);
		add(caixatexto);
		botao = new JButton();
		botao.setText("OK");
		botao.addActionListener(objetolistener);
		botao.setBounds(45,50,70,30);
		add(botao);
		setSize(200,130);
		setVisible(true);
	}
```

```
    public static void main(String args[]){
        ComEvento janela = new ComEvento();
    }
    public class Handler implements ActionListener{
    public void actionPerformed (ActionEvent event){
        if (event.getSource() == botao)
            JOptionPane.showMessageDialog(null,caixatexto.getText());
        }
    }
}
```

Modelo de Tratamento de Eventos

- O exemplo mostrado trata o evento de clicar o botão
- Quando ele ocorre um *JOptionPane* aparece na tela contendo o texto que está na caixa de texto
- Para a utilização de um objeto *listener* deve-se:
  - criar uma classe que implementa *ActionListener*
  - criar um objeto desta classe
  - e associá-lo a cada elemento que se deseja manipular os eventos

Inserindo Figuras

- Objetos da classe Icon são utilizados para inserir figuras em componentes da GUI
- Tais componentes podem ser botões, labels, etc
- As figuras podem ser no formato .gif, .jpg e .png
- O arquivo da figura deve está na pasta raiz do projeto
- Exemplo:

```
import javax.swing.*;
public class ComFigura extends JFrame {
	Icon fig;
	JLabel lbl;
	public ComFigura(){
	getContentPane().setLayout(null);
	fig = new ImageIcon("figura.png");
	lbl = new JLabel();
	lbl.setIcon(fig);
	lbl.setBounds(10,10,200,200);
	add(lbl);
	setSize(240,260);
	setVisible(true);
	setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
	public static void main (String args[]){
		ComFigura aplicacao = new ComFigura();
	}
}
```

Tratando eventos do Mouse

- Duas classes tratam os eventos do mouse. Elas possuem métodos associados a cada tipo de evento, são eles:
- MouseMotionListener
  - mouseDragged: depois do arraste
    - » Ocorre um mousePressed antes e um mouseRelease depois
  - mouseMoved: ponteiro do mouse movido quando o ponteiro dentro dos limites do objeto

Tratando eventos do Mouse

- MouseListener
  - mousePressed: botão do mouse é precionado
  - mouseClicked: botão do mouse é liberado sem movimento
  - mouseReleased: botão do mouse é liberado com movimento
  - mouseEntered: ponteiro do mouse entra nos limites físicos do componente
  - mouseExited: ponteiro do mouse sai dos limites físicos do componente

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class EventosMouse extends JFrame{
JLabel statusbar;

public EventosMouse(){

getContentPane().setLayout(null);
statusbar = new JLabel("Início");
statusbar.setBounds(10, 10, 300, 30);
add(statusbar);

Handler ol = new Handler();

addMouseListener(ol);
addMouseMotionListener(ol);

setBounds(10,10,300,300);
setSize(300,200);
setVisible(true);

    public static void main(String args[]){
        EventosMouse j = new EventosMouse();
        j.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

```
    public class Handler implements MouseListener, MouseMotionListener{
        public void mousePressed(MouseEvent event){
            statusbar.setText("mousePressed : [" + event.getX() + "," + event.getY() + "]");
        }
        public void mouseClicked(MouseEvent event){
            statusbar.setText("mouseClicked : [" + event.getX() + "," + event.getY() + "]");
        }
        public void mouseReleased(MouseEvent event){
            statusbar.setText("mouseReleased : [" + event.getX() + "," + event.getY() + "]");
        }
        public void mouseEntered(MouseEvent event){
            JOptionPane.showMessageDialog(null,"mouseEntered");
        }
        public void mouseExited(MouseEvent event){
            JOptionPane.showMessageDialog(null,"mouseExited");
        }
        public void mouseDragged(MouseEvent event){
            statusbar.setText("mouseDragged : [" + event.getX() + "," + event.getY() + "]");
        }
        public void mouseMoved(MouseEvent event){
            statusbar.setText("mouseMoved : [" + event.getX() + "," + event.getY() + "]");
        }
    }
}
```

Classes Adaptadoras

- Para evitar a implementação de todos os métodos das interfaces *Listener* podemos utilizar as classes adaptadoras.
- Para cada interface existe uma classe correspondente:

  MouseListener : MouseAdapter

  MouseMouseListener : MouseMotionAdapter
- Para utilização de tal classes troque “implements mouseListener” por “extends mouseAdapter”

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;
public class ClassesAdaptadoras extends JFrame{
	int x=0, y=0;
	public ClassesAdaptadoras(){
		getContentPane().setLayout(null);
		Handler obj = new Handler();
		addMouseMotionListener(obj);
		setSize(200,200);
		setVisible(true);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		repaint();
	}
	public void paint(Graphics g){
		g.fillOval(x,y,4,4);
	}
	public static void main(String args[]){
		ClassesAdaptadoras janela = new ClassesAdaptadoras();
	}
	public class Handler extends MouseMotionAdapter{
		public void mouseDragged(MouseEvent event){
			x = event.getX();
			y = event.getY();
			repaint();
		}
	}
}
```

Posicionando componentes no container

- O Java possui gerenciadores de Layout, são eles:
  - **FlowLayout**: fluxo de componentes
  - **BorderLayout**: organiza em 5 regiões (NORTH, SOUTH, EAST, WEST, CENTER)
  - **GridLayout**: divide o container em células
  - GridBagLayout: idem GridLayout, mas os componentes podem ocupar mas de uma célula
  - CardLayout: organiza como um pilha de cartas, só o primeiro é visível (como abas)
  - BoxLayout

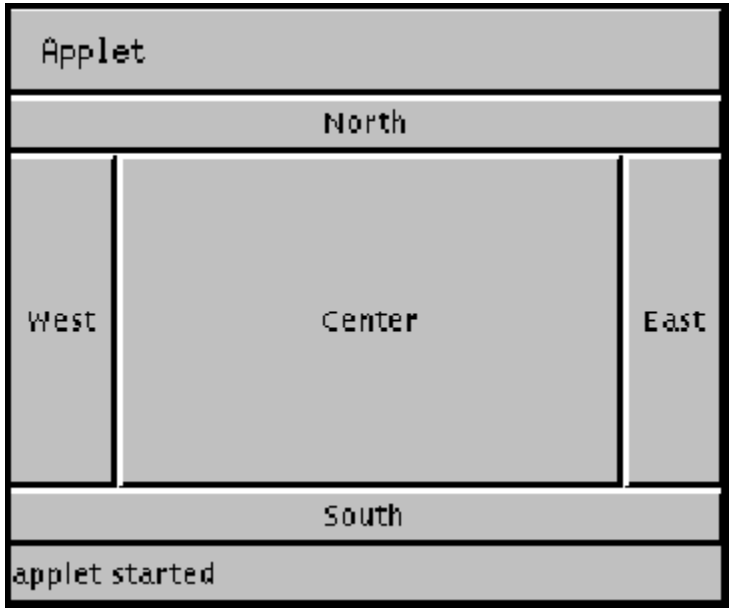
FlowLayout

- Um componente ao lado do outro
- Se não couber, o componente vai para a próxima linha
- Veja Exemplo

```
import java.awt.*;
import javax.swing.*;
public class TesteLayout extends JFrame{
    JTextField jtfNome,jtfTelefone;
    JButton jtfOk;
    public TesteLayout(){
        getContentPane().setLayout(new FlowLayout());
        jtfNome = new JTextField(10);
        add(jtfNome);
        jtfTelefone = new JTextField(15);
       add(jtfTelefone);
        jtfOk = new JButton("Botão OK");
       add(jtfOk);
       setVisible(true);
       setSize(300,300);
   }
   public static void main(String args[]){
       new TesteLayout();
   }
}
```

BorderLayout

- Possui posições predefinidas



```
import java.awt.*;
import javax.swing.*;
public class ComBorder extends JFrame{
      JTextField jtfNome,jtfTelefone;
      JButton jtfOk;
      public ComBorder(){
          getContentPane().setLayout(new BorderLayout());
            jtfNome = new JTextField(10);
            add(jtfNome,BorderLayout.SOUTH);
            jtfTelefone = new JTextField(15);
            add(jtfTelefone,BorderLayout.NORTH);
            jtfOk = new JButton("Botão OK");
            add(jtfOk,BorderLayout.LINE_END);
            setVisible(true);
            setSize(300,300);
      }
      public static void main(String args[]){
          new ComBorder();
      }
}
```

GridLayout

- Divide a tela em células como uma tabela
- O componente ocupa toda a célula
- Construturores
  - GridLayout(linha,coluna)
  - GridLayout(linha,coluna,hgap,vgap)

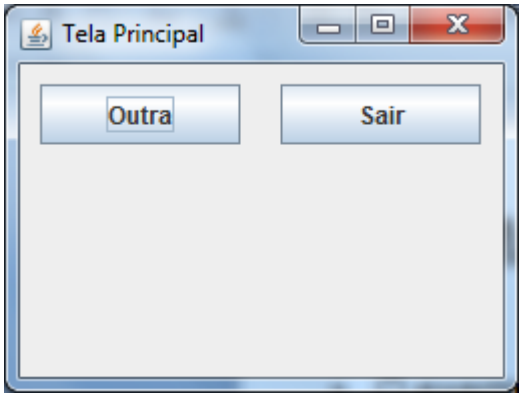
- Gap: espação entre linhas ou colunas
- Obs: 0 para linhas especifica quantas linhas forem necessárias

```
import java.awt.*;

import javax.swing.*;
public class ComGridLayout  extends JFrame{
      JTextField jtfNome,jtfTelefone;
      JButton jtfOk;
      public ComGridLayout (){
            getContentPane().setLayout(new GridLayout(0,2,30,30));
            jtfNome = new JTextField(10);
            add(jtfNome);
            jtfTelefone = new JTextField(15);
            add(jtfTelefone);
            jtfOk = new JButton("Botão OK");
            add(jtfOk);
            setVisible(true);
            setSize(300,300);
      }
      public static void main(String args[]){
          new ComGridLayout();
      }
}
```

Manipulando Várias Telas

- O exemplo a seguir possui
  - Tela principal (PrincipalCor)



  - Tela MudaCor: com dois botões onde é possível mudar a cor da tela PrincipalCor.

Manipulando Várias Telas

- Classe PrincipalCor possui:
  - Um objeto que corresponde a segunda tela
    - MudarCor tela = **new** MudarCor(**this**);
  - Um método que controla qual tela será exibida

```
public void showPrincipal(){
 tela.setVisible(false);
}
```

  - Métodos para mudar a cor

```
public void azul(){
 getContentPane().setBackground(Color.BLUE);
}
public void amarelo(){
 getContentPane().setBackground(Color.YELLOW);
}
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class PrincipalCor extends JFrame {
	JButton jbtSair, jbtOutra;
	private String cor = "Default";
	MudarCor tela = new MudarCor(this);
	public PrincipalCor(){
		getContentPane().setLayout(null);
		setTitle("Tela Principal");
		Handler obj = new Handler();
		jbtOutra = new JButton("Outra");
		jbtOutra.setBounds(10,10,100,30);
		jbtOutra.addActionListener(obj);
		add(jbtOutra);
		jbtSair = new JButton("Sair");
		jbtSair.setBounds(130,10,100,30);
		jbtSair.addActionListener(obj);
		add(jbtSair);
		setBounds(10,10,300,300);
		setVisible(true);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
```

```
    public class Handler implements ActionListener{
        public void actionPerformed(ActionEvent e){
            if(e.getSource()==jbtSair)
                System.exit(0);
            if(e.getSource()==jbtOutra)
                tela.setVisible(true);
        }
    }
    public void mudaCor(String color){
        cor = color;
        if(color.equals("Azul"))
            getContentPane().setBackground(Color.BLUE);
        if(color.equals("Amarelo"))
            getContentPane().setBackground(Color.YELLOW);
      }
    public String getCor(){
       return cor;
    }
    public void showPrincipal(){
        tela.setVisible(false);
    }
    public static void main(String[] args) {
        new PrincipalCor();
    }
}
```

Manipulando Várias Telas

- Classe MudaCor:
  - Extends JDialog, poia possui o método showDialog :
    - true – só essa tela pode ser clicada
    - false – a primeira pode ser clicada
  - Recebe no construtor um objeto do tipo PrincipalCor e coloca esse objeto como global

```
PrincipalCor pc;
public MudarCor(PrincipalCor pc){
  this.pc = pc;
```

  - Chama os métodos do objeto pc:

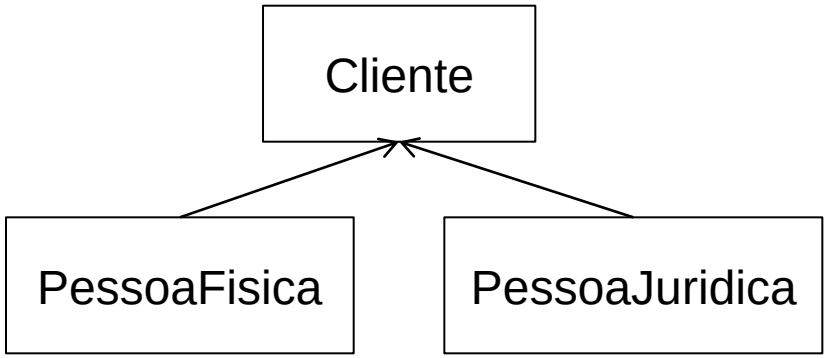
```
pc.mudaCor("Azul");
pc.showPrincipal();
setTitle("Cor atual: " + pc.getCor());
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MudarCor extends JDialog{
	JButton jbtAzul, jbtAmarelo, jbtFechar;
	PrincipalCor pc;
	public MudarCor(PrincipalCor pc){
		this.pc = pc;
		getContentPane().setLayout(null);
		setTitle("Cor atual: " + pc.getCor());
		Handler obj = new Handler();
		jbtAzul = new JButton("Azul");
		jbtAzul.setBounds(10,10,100,30);
		jbtAzul.addActionListener(obj);
		add(jbtAzul);
		jbtAmarelo = new JButton("Amarelo");
		jbtAmarelo.setBounds(130,10,100,30);
		jbtAmarelo.addActionListener(obj);
		add(jbtAmarelo);
		jbtFechar = new JButton("Fechar");
		jbtFechar.setBounds(70,60,100,30);
		jbtFechar.addActionListener(obj);
		add(jbtFechar);
		setBounds(150,200,255,150);
		setVisible(false);
		setModal(true);
	}
```

```
    public class Handler implements ActionListener{
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == jbtAzul){
                pc.mudaCor("Azul");
            }
            if(e.getSource() == jbtAmarelo){
                pc.mudaCor("Amarelo");
            }
            if(e.getSource() == jbtFechar){
                pc.showPrincipal();
                //setVisible(false);
            }
            setTitle("Cor atual: " + pc.getCor());
        }
    }
}
```

Informações Entre Várias Telas

- O exemplo a seguir possui um Vector na tela principal e utiliza duas outras telas para cadastra objetos nesse Vector
- Utiliza a seguinte estrutura de classes



- Verificar que novos objetos são criados no Vector a partir das telas de cadastro, utilizando-se para isso chamadas de métodos da tela principal

```
public abstract class Cliente {
	private long codigo;
	public long getCodigo() {
		return codigo;
	}
	public void setCodigo(long codigo) {
		this.codigo = codigo;
	}
	public abstract String todosDados();
}
public class PessoaFisica extends Cliente{
	private String cpf;
	public String getCpf() {
		return cpf;
	}
	public void setCpf(String cpf) {
		this.cpf = cpf;
	}
	public String todosDados(){
		return "Código:"+getCodigo() + "  CPF:" + getCpf();
	}
}
```

```
public class PessoaJuridica extends Cliente{
	private String cnpj;
	public String getCnpj() {
		return cnpj;
	}
	public void setCnpj(String cnpj) {
		this.cnpj = cnpj;
	}
	public String todosDados(){
		return "Código:"+getCodigo() + "
CNPJ" + getCnpj();
	}
}
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
public class PrincipalClientes extends JFrame {
    JButton jbtPF, jbtPJ;
    Vector <Cliente> meusclientes = new Vector<Cliente>();
    CadastraPF pf = new CadastraPF(this);
    CadastraPJ pj = new CadastraPJ(this);
    public PrincipalClientes(){
        getContentPane().setLayout(null);
        setTitle("Tela Principal");
        Handler obj = new Handler();
        jbtPF = new JButton("Pessoa Fisica");
        jbtPF.setBounds(10,10,150,30);
        jbtPF.addActionListener(obj);
        add(jbtPF);
        jbtPJ = new JButton("Pessoa Jurídica");
        jbtPJ.setBounds(10,60,150,30);
        jbtPJ.addActionListener(obj);
        add(jbtPJ);
        setBounds(10,10,300,300);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void cadastra(Cliente c){
        meusclientes.add(c);
    }
```

```
    public void showPrincipal(){
        pf.setVisible(false);
        pj.setVisible(false);
    }
    public void showCadastraPJ(){
        pf.setVisible(false);
        pj.setVisible(true);
    }
    public void showCadastraPF(){
        pf.setVisible(true);
        pj.setVisible(false);
    }
    public class Handler implements ActionListener{
        public void actionPerformed(ActionEvent e){
            if(e.getSource()==jbtPF){
                showCadastraPF();
            }
            if(e.getSource()==jbtPJ){
                showCadastraPF();
            }
        }
    }
    public static void main(String[] args) {
        new PrincipalClientes();
    }
}
```

```
public class CadastraPF extends JDialog{
	JButton jbtCadastrar, jbtFechar;
	JTextField jtfCodigo, jtfCpf;
	PrincipalClientes pc;
	public CadastraPF(PrincipalClientes pc){
		this.pc = pc;
		…
	}
	public class Handler implements ActionListener{
		public void actionPerformed(ActionEvent e){
			if(e.getSource() == jbtCadastrar){
				PessoaFisica aux = new PessoaFisica();
				aux.setCodigo(Long.parseLong(jtfCodigo.getText()));
				aux.setCpf(jtfCpf.getText());
				pc.cadastra(aux);
				jtfCodigo.setText("");
				jtfCpf.setText("");
			}
			if(e.getSource() == jbtFechar){
				pc.showPrincipal();
			}
		}
	}

}
```

Exercícios

- Crie a tela para cadastro de PessoaJuridica
- Crie uma tela que exibe todos os dados de todos os objetos cadastrados no Vector

```
package trataclientes;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
public class PrincipalClientes extends JFrame {
     JButton jbtPF, jbtPJ, jbtExibe;
     Vector <Cliente> meusclientes = new Vector<Cliente>();
     CadastraPF pf = new CadastraPF(this);
     CadastraPJ pj = new CadastraPJ(this);
     public PrincipalClientes(){
          getContentPane().setLayout(null);
          setTitle("Tela Principal");
          Handler obj = new Handler();
          jbtPF = new JButton("Pessoa Fisica");
          jbtPF.setBounds(10,10,150,30);
          jbtPF.addActionListener(obj);
          add(jbtPF);
          jbtPJ = new JButton("Pessoa Jurídica");
          jbtPJ.setBounds(10,60,150,30);
          jbtPJ.addActionListener(obj);
          add(jbtPJ);
          jbtExibe = new JButton("Exibe");
          jbtExibe.setBounds(10,110,150,30);
          jbtExibe.addActionListener(obj);
          add(jbtExibe);
          setBounds(110,10,300,300);

          setVisible(true);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     }
     public void cadastra(Cliente c){
          meusclientes.add(c);
     }
```

```
    public void showPrincipal(){
        pf.setVisible(false);
        pj.setVisible(false);
    }
    public void showCadastraPJ(){
        pf.setVisible(false);
        pj.setVisible(true);
    }
    public void showCadastraPF(){
        pf.setVisible(true);
        pj.setVisible(false);
    }
    public class Handler implements ActionListener{
        public void actionPerformed(ActionEvent e){
            if(e.getSource()==jbtPF){
                showCadastraPF();
            }
            if(e.getSource()==jbtPJ){
                showCadastraPF();
            }
            if(e.getSource()==jbtExibe){
                new ExibeDados(meusclientes);
            }
        }
    }
    public static void main(String[] args) {
        new PrincipalClientes();
    }
}
```

```
package trataclientes;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
public class ExibeDados extends JDialog {
	JTextArea jtaDados;
	JButton jbtFechar;
	Vector<Cliente> clientes;
	public ExibeDados(Vector<Cliente>c){
		clientes = c;
		getContentPane().setLayout(null);
		Handler obj = new Handler();
		jbtFechar = new JButton("Fechar");
		jbtFechar.setBounds(80,270,100,30);
		jbtFechar.addActionListener(obj);
		add(jbtFechar);

		jtaDados = new JTextArea();
		jtaDados.setBounds(10,10,250,250);
		jtaDados.setText(todosDados());
		add(jtaDados);

		setBounds(100,100,300,300);
		setVisible(true);
		setModal(true);
	}
```

```
    public String todosDados(){
        String s = "";
        for(int i=0;i< clientes.size();i++){
            s = s + "\n" + clientes.get(i).todosDados();
        }
        return s;
    }

    public class Handler implements ActionListener{
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == jbtFechar){
                setVisible(false);
            }
        }

    }

}
```