

Interfaces gráficas

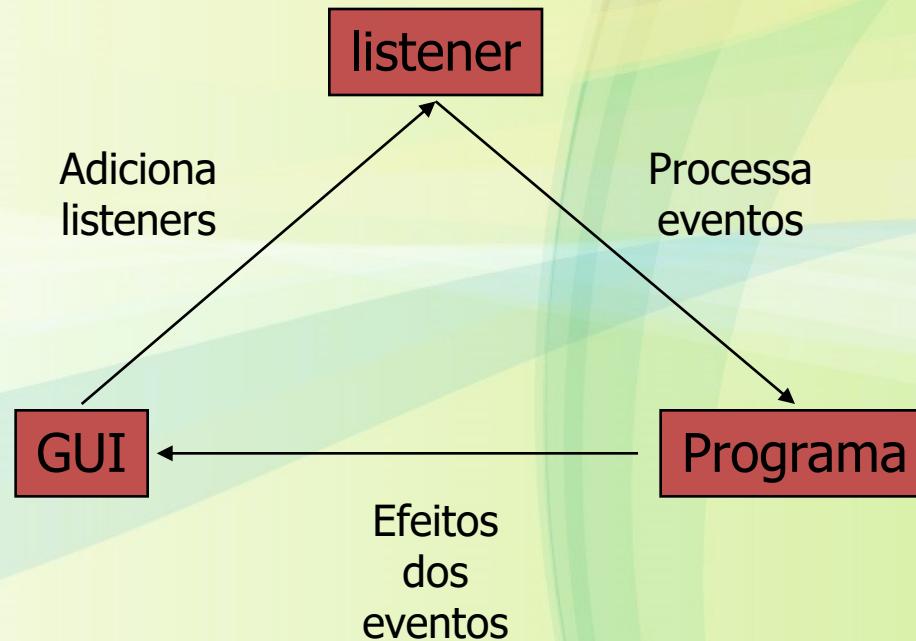
- Muitas aplicações disponibilizam aos seus utilizadores uma interface gráfica (**GUI - Graphical User Interface**)
- As packages **java.awt** e **javax.swing** suportam a criação deste tipo de interfaces
- Os elementos chave de uma interface gráfica em Java são:
 - Componentes
 - Gestores de posicionamento (layout managers)
 - Processadores de eventos
- **Componentes** são elementos como botões ou campos de texto que podem ser manipulados pelo utilizador com o rato ou o teclado

Interfaces gráficas

- **Gestores de posicionamento** determinam a forma como os componentes aparecem no ecrã
- **Processadores de eventos** respondem a ações do utilizadores (ex: pressão num botão do rato)
- Os programas com interfaces gráficas devem responder a eventos, gerados pelos componentes, que indicam que determinadas ações ocorreram
- Existe uma categoria de classes chamada **listeners** que está atenta à ocorrência de eventos
- Assim, um programa deste tipo é composto por:
 - Código que apresenta a interface ao utilizador
 - Os listeners que esperam que ocorram eventos
 - O código que é executado quando ocorre um evento

Interfaces gráficas

- O modelo de programação de interfaces gráficas pode ser representado por:



Interfaces gráficas

- Assim, no geral a programação de interfaces gráficas consiste em:
 - Colocar os componentes desejados
 - Esperar por uma acção do utilizador sobre um dos componentes
 - Quando a ação acontece é detectada pelo **event listener** que chama o(s) método(s) apropriado(s)
- Esta técnica é geralmente designada por programação dirigida por eventos (**event-driven programming**)

Interfaces gráficas

- Estas aplicações baseiam-se geralmente em janelas que contêm os diversos componentes
- A classe **JFrame** representa janelas em Java:

```
import javax.swing.*;  
public class Janela {  
    private final static int dimH = 300;  
    private final static int dimV = 300;  
    public static void main(String[] arg) {  
        JFrame jan = new JFrame();  
        jan.setSize(dimH,dimV);  
        jan.setTitle("Janela");  
        jan.setVisible(true);  
    }  
}
```

Interfaces gráficas

- Para além do `setSize()` e do `setTitle()`, a classe `JFrame` inclui outros métodos que permitem definir as características da janela. Por exemplo, `setCursor()` ou `setBackground()`
- A classe `JFrame` por si só é de pouca utilidade, uma vez que apenas cria a janela sem que esta tenha a capacidade de suportar desenhos ou texto
- Para isso é necessário utilizar um objecto da classe `JPanel` (ou um seu descendente)

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
public class Desenho extends JPanel {
    protected int cx, cy, comp, larg;
    protected Color cor;
    //Construtor, inicializa as variáveis de instância
    public Desenho(int cx, int cy, int comp, int larg, Color cor) {
        this.cx = cx; this.cy = cy; this.comp = comp;
        this.larg = larg; this.cor = cor;
    }
    //Cria o desenho. É chamado sempre que a janela fica visível
    public void paintComponent (Graphics g) {
        super.paintComponent(g);
        g.setColor(cor);
        g.fillRect(cx,cy,comp,larg);
    }
}
```

Interfaces gráficas

- O método **paintComponent()** desempenha um papel fundamental, uma vez que é chamado automaticamente sempre que é necessário desenhar a janela (por exemplo após ter sido removida uma outra janela que a tapava)
- O método **paintComponent()** recebe como parâmetro um objecto da classe **Graphics**, através do qual o intérprete lhe fornece o contexto gráfico em que o desenho vai ser efectuado

Interfaces gráficas

- Para que o desenho do retângulo possa aparecer na janela, é preciso uma ligação entre esta e o objecto da classe Desenho:

```
import java.awt.*;
import javax.swing.*;
public class Janela {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
        JFrame f = new JFrame();
        f.setSize(dimH,dimV);
        f.setTitle("Janela com rectângulo");
        Desenho d = new Desenho(100,50,100,80,Color.blue);
        Container c = f.getContentPane(); // superfície de desenho
        c.add(d);
        f.setVisible(true);
    }
}
```

Interfaces gráficas

- Suponhamos que se pretendem adicionar à janela três botões, um para tornar a cor do rectângulo mais clara, outro para a tornar mais escura e o terceiro para terminar a aplicação.
- Os botões são definidos pela classe **JButton**, incluída na biblioteca **javax.swing**.
- Para adicionar um botão a uma janela, segue-se uma aproximação semelhante à que foi utilizada para adicionar um objecto da classe Desenho, ou seja, utiliza-se o método **add()** da classe **JContainer**.

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
public class Janela {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
        JFrame f = new JFrame();
        f.setSize(dimH,dimV);
        f.setTitle("Janela com rectângulo");
        Desenho d = new Desenho(100,50,100,80,Color.blue);
        Container c = f.getContentPane();
        c.add(d);
        c.add(new JButton("Claro"));
        c.add(new JButton("Escuro"));
        c.add(new JButton("Sair"));
        f.setVisible(true);
    }
}
```

Interfaces gráficas

- Cada um dos componentes (JPanel e JButton) é adicionado à janela sem especificar a sua dimensão ou a sua posição.
- Nesta situação, o intérprete sobrepõe os diversos componentes pela ordem por que são criados, resultando em:



Interfaces gráficas

- Para evitar este comportamento indesejado, os vários componentes gráficos que se pretendem mostrar devem ser colocados dentro de um contentor.
- Este pode ser visto como uma caixa onde vão ser postos todos os elementos de comunicação utilizados na aplicação.
- Após a colocação dos componentes gráficos no contentor, este pode ser adicionado à janela.
- Os contentores são definidos pela classe **JPanel**, integrada também na biblioteca javax.swing.
- É, então, possível criar uma classe para representar um contentor que contenha os três botões da aplicação:

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
public class Botoes extends JPanel {
    public Botoes() {
        JButton claro = new JButton("Claro");
        this.add(claro);
        JButton escuro = new JButton("Escuro");
        this.add(escuro);
        JButton sair = new JButton("Sair");
        this.add(sair);
    }
}
```

Interfaces gráficas

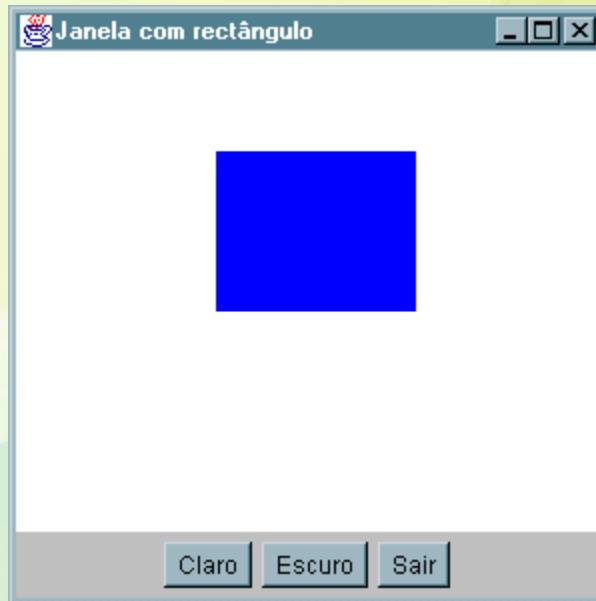
- Uma vez criado o contentor, é necessário adicioná-lo à janela.
- É possível controlar a localização do desenho e do contentor com os botões, de forma a evitar a sua sobreposição.
- Para isso, podem ser utilizados os termos “**North**”, “**South**”, “**West**”, “**East**” e “**Center**”.
- Estes termos definem o posicionamento relativo entre os vários elementos por analogia com os pontos cardeais e podem ser incluídos como parâmetro no método **add()**.

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
public class Janela {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
        JFrame f = new JFrame();
        f.setSize(dimH,dimV);
        f.setTitle("Janela com rectângulo");
        //Cria e adiciona a área de desenho no centro da janela
        Desenho d = new Desenho(100,50,100,80,Color.blue);
        Container c = f.getContentPane();
        c.add(d, "Center");
        //Cria e adiciona o contentor na parte de baixo da janela
        c.add (new Botoes(), "South");
        f.setVisible(true);
    }
}
```

Interfaces gráficas

- O resultado da execução deste programa é:



Interfaces gráficas

- Os três botões e o desenho aparecem agora corretamente posicionados.
- No entanto, o facto dos botões estarem visíveis não significa que funcionem como esperado.
- A classe **JButton** define apenas os atributos gráficos dos botões, mas não lhes confere qualquer funcionalidade.
- Para associar um botão a uma acção, é necessário fazer a gestão dos eventos correspondentes

Interfaces gráficas

- A linguagem Java faz a gestão dos eventos provocados pelo utilizador através de objetos destinados à sua receção e processamento.
- Simplificando, pode-se afirmar que sempre que o utilizador seleciona um componente, este gera um evento que é enviado para um processador de eventos que lhe deve estar associado.
- O processador deve responder à ação do utilizador através de um método denominado **actionPerformed()**.
- No caso da aplicação que tem vindo a ser apresentada, é necessário associar um processador de eventos a cada um dos três botões, o que deve ser feito no construtor da classe Botoes

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Botoes extends JPanel {
    public Botoes(Desenho d) {
        JButton claro = new JButton("Claro");
        claro.addActionListener(new GereEventos(1, d));
        this.add(claro);
        JButton escuro = new JButton("Escuro");
        escuro.addActionListener(new GereEventos(2, d));
        this.add(escuro);
        JButton sair = new JButton("Sair");
        sair.addActionListener(new GereEventos(3, d));
        this.add(sair);
    }
}
```

Interfaces gráficas

- É importante realçar algumas diferenças entre esta versão da classe **Botoes** e a anterior:
 - A inclusão de **import java.awt.event.*;**; que indica ao compilador a biblioteca onde estão as classes necessárias para o processamento dos eventos
 - O parâmetro da classe **Desenho** no construtor da classe **Botoes**. Este é necessário, porque os botões vão ter que fornecer o endereço da área de desenho ao processador de eventos, para que este possa provocar as correspondentes alterações no desenho
 - A chamada do método **addActionListener()**, definido na classe **JButton**, para associar o processador de eventos a cada botão. Este método recebe um objeto da classe **GereEventos** (o processador de eventos)
 - O construtor da classe **GereEventos** recebe dois parâmetros, um int que identifica qual dos botões foi acionado e o endereço da área de desenho

Interfaces gráficas

- O código da classe GereEventos é apresentado em seguida:

```
import java.awt.event.*;
public class GereEventos implements ActionListener {
    private int bot;
    private Desenho des;
    public GereEventos(int n, Desenho d) {
        bot = n;
        des = d;
    }
    public void actionPerformed(ActionEvent e) {
        switch (bot) {
            case 1:des.claro(); break;
            case 2:des.escuro(); break;
            case 3:System.exit(0); break;
        }
    }
}
```

Interfaces gráficas

- De referir que a classe **GereEventos** implementa uma interface **ActionListener**
- A interface **ActionListener** define apenas um método, **actionPerformed()**, pelo que só este terá que ser implementado em **GereEventos**

Interfaces gráficas

- O método `actionPerformed()` é executado sempre que o utilizador selecionar com o rato um dos botões.
- O primeiro dos parâmetros que recebe permite-lhe identificar qual dos botões foi ativado e reagir em conformidade
- O segundo parâmetro é o endereço da área de desenho que o utilizador poderá modificar através da utilização dos botões. Este endereço é obtido no método `main()`, e deve ser passado ao contentor que contém os botões através do respetivo construtor, que poderá depois passá-lo ao gestor de eventos (da classe `GereEventos`)
- Obtém-se, assim, uma nova versão para a classe Janela:

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Janela {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
        JFrame f = new JFrame();
        f.setSize(dimH,dimV);
        f.setTitle("Janela com retângulo");
        Desenho d = new Desenho(100,50,100,80,Color.blue);
        Container c = f.getContentPane();
        c.add(d, "Center");
        c.add(new Botoes(d), "South");
        f.setVisible(true);
    }
}
```

Interfaces gráficas

- A classe Desenho teve também que sofrer alterações, de modo a incluir os métodos **claro()** e **escuro()**:

```
public void claro() {  
    cor = cor.brighter();  
    repaint();  
}  
  
public void escuro() {  
    cor = cor.darker();  
    repaint();  
}
```

Interfaces gráficas

- Exemplo: Applet que replica o texto introduzido num textfield



Interfaces gráficas

```
import javax.swing.*;  
  
// Classe Mimic demonstra um componente simples e um evento  
public class Mimic extends JApplet {  
    MimicGUI gui = new MimicGUI (this);  
    public void init() {  
        gui.init();  
    }  
    // Repete o texto introduzido no text field  
    public void update_label() {  
        gui.update_label ();  
    }  
}
```

Interfaces gráficas

```
// Classe MimicGUI representa a GUI para a applet Mimic
import javax.swing.*;
import java.awt.*;
class MimicGUI {

    private JTextField quote = new JTextField(20);
    private JLabel label = new JLabel ("Só sei que nada sei");
    private Mimic applet;
    private Mimic_Action_Listener listener;

    // construtor MimicGUI
    public MimicGUI (Mimic mimic_applet) {
        applet = mimic_applet;
        listener = new Mimic_Action_Listener (applet);
    }
}
```

Interfaces gráficas

```
// Cria a GUI e o listener para o text field
public void init() {
    Container c = applet.getContentPane();
    c.add (quote, "North");
    c.add (label, "Center");
    applet.resize (250,100);
    quote.addActionListener (listener);
}

// Actualiza o texto da label.
public void update_label () {
    label.setText (quote.getText());
}
```

Interfaces gráficas

```
// Classe Mimic_Action_Listener trata dos eventos para a Mimic
import java.awt.event.*;
class Mimic_Action_Listener implements ActionListener {
    private Mimic applet;

    // Prepara o listener guardando uma referência para a applet
    // constructor Mimic_Action_Listener
    public Mimic_Action_Listener (Mimic listening_applet) {
        applet = listening_applet;
    }

    // Actualiza a label quando ocorre um evento
    public void actionPerformed (ActionEvent event) {
        applet.update_label();
    }
}
```

Interfaces gráficas

- Há uma **interface listener** definida para cada tipo de evento, contendo cada uma delas os métodos necessários para responder a esse tipo de evento
- Uma classe **listener** implementa uma dada **interface listener**, pelo que deve implementar todos os métodos nela definidos
- Os listeners são adicionados aos componentes
- Quando um componente gera um evento, o método correspondente a esse evento é executado no seu listener
- Um componente pode ter vários listeners (para vários eventos)
- Uma classe listener pode implementar vários interfaces listener, ou seja pode detetar vários tipos de eventos

Interfaces gráficas

- Existem diversos tipos de eventos e cada um deles tem o seu tipo de **listener** que terá que ser implementado
- Por exemplo:

Acção do utilizador	Evento gerado	Event listener
---------------------	---------------	----------------

Seleccionar botão	ActionEvent	ActionListener
Mover o rato	MouseEvent	MouseListener
Rato entra no componente	FocusEvent	FocusListener
Escrever no teclado	KeyEvent	KeyListener
Escrever num TextField	ActionEvent	ActionListener

Interfaces gráficas

```
import java.awt.*;
import java.applet.*;
// Classe Events demonstra a ocorrência de eventos. Espera vários
// eventos
// e escreve mensagens à medida que ocorrem
public class Events extends Applet {
    private TextArea log = new TextArea (15, 100);
    private Label count_label = new Label("", Label.CENTER);
    private Universal_Listener listener = new
        Universal_Listener (log, count_label);
```

Interfaces gráficas

```
public void init() {  
    add ("North", new Label ("Event Logging", Label.CENTER));  
    add ("Center", log);  
    add ("South", count_label);  
    addComponentListener (listener);  
    log.addFocusListener (listener);  
    log.addKeyListener (listener);  
    log.addMouseListener (listener);  
    log.addMouseMotionListener (listener);  
    setSize (800, 400);  
} } // class Events
```

Interfaces gráficas

```
// Classe Universal_Listener implementa muitos event interfaces,  
import java.awt.*;  
import java.awt.event.*;  
class Universal_Listener implements ComponentListener,  
    MouseMotionListener, MouseListener, KeyListener, FocusListener {  
    private TextArea log;  
    private Label count_label;  
    private String count_text = "Number of Events: ";  
    private int count = 0;  
  
    public Universal_Listener (TextArea log, Label count_label) {  
        this.log = log;  
        this.count_label = count_label;  
    } // constructor Universal_Listener
```

Interfaces gráficas

```
private void log_event (AWTEvent event) {  
    count++;  
    count_label.setText (count_text + count);  
    log.append (event.toString() + "\n");  
}  
public void componentMoved (ComponentEvent event) {  
    log_event (event);  
}  
public void componentHidden (ComponentEvent event) {  
    log_event (event);  
}  
public void componentResized (ComponentEvent event) {  
    log_event (event);  
}  
public void componentShown (ComponentEvent event) {  
    log_event (event);  
}
```

Interfaces gráficas

```
public void mouseDragged (MouseEvent event) {  
    log_event (event);  
}  
public void mouseMoved (MouseEvent event) {  
    log_event (event);  
}  
public void mousePressed (MouseEvent event) {  
    log_event (event);  
}  
public void mouseReleased (MouseEvent event) {  
    log_event (event);  
}  
public void mouseEntered (MouseEvent event) {  
    log_event (event);  
}  
public void mouseExited (MouseEvent event) {  
    log_event (event);  
}  
public void mouseClicked (MouseEvent event) {  
    log_event (event);  
}
```

Interfaces gráficas

```
public void keyPressed (KeyEvent event) {  
    log_event (event);  
}  
public void keyReleased (KeyEvent event) {  
    log_event (event);  
}  
public void keyTyped (KeyEvent event) {  
    log_event (event);  
}  
public void focusGained (FocusEvent event) {  
    log_event (event);  
}  
public void focusLost (FocusEvent event) {  
    log_event (event);  
}  
} // class Universal_Listener
```

Interfaces gráficas

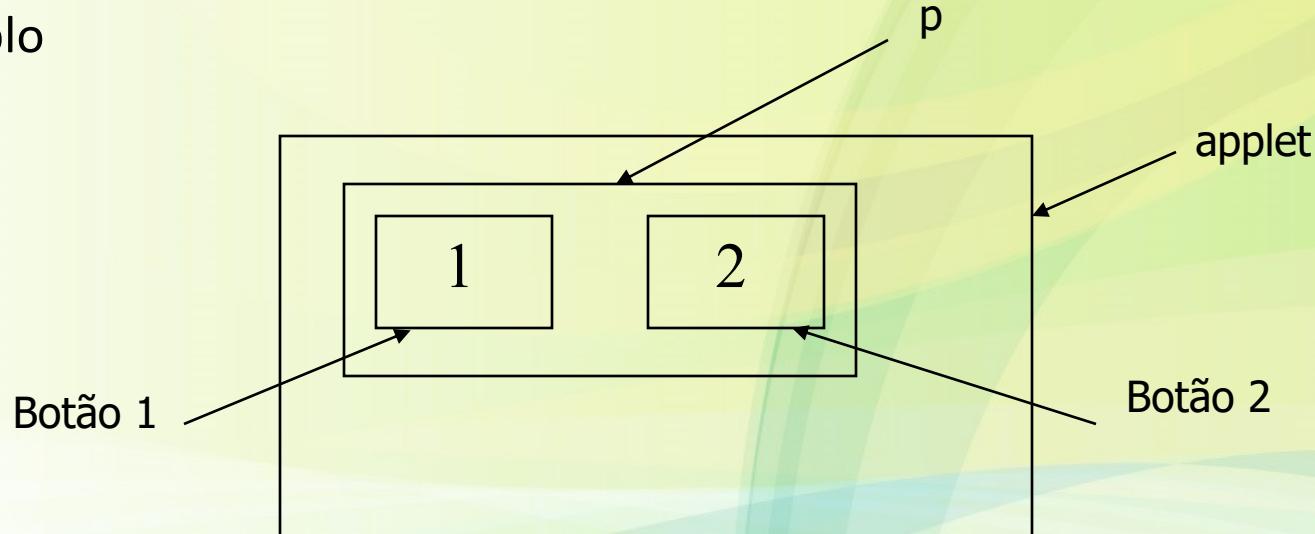
- Um contentor (**container**) é um tipo especial de componente que serve para agrupar outros componentes (eventualmente outro contentor)
- Por exemplo, uma applet é um contentor e, por isso, podem-lhe ser adicionados outros componentes
- A cada contentor é adicionado um gestor de posicionamento (layout manager) que controla a forma como os seus componentes são mostrados
- Cada contentor tem dois métodos:
 - add (...) - Adiciona um componente ao contentor
 - setLayout (...) - Indica qual o gestor de posicionamento a usar

Interfaces gráficas

- Alguns contentores (**JPanel** e **JApplet**) têm que ser ligados a uma superfície gráfica
- Uma applet é ligada a um browser ou a um appletviewer
- Outros contentores (**JWindow**, **JFrame**, **JDialog**) podem ser movidos independentemente
- A utilização de contentores implica que os componentes sejam primeiro adicionados ao contentor, sendo este depois adicionado à applet
- Assim, uma interface gráfica é conseguida pela junção de contentores e outros componentes
- O desenho da interface gráfica é uma parte fundamental do desenvolvimento de aplicações
- Não esquecer que para um utilizador comum **a interface é o sistema**

Interfaces gráficas

- Exemplo



// Na applet

```
 JButton botao1 = new JButton ("1");
 JButton botao2 = new JButton ("2");
 JPanel p = new JPanel();
 p.add (botao1);
 p.add (botao2);
 getContentPane().add (p);
```

Interfaces gráficas

- Há diversos tipos de componentes que permitem interacção com o utilizador:
 - Etiquetas - Classe **JLabel**
 - Campos de texto - Classe **JTextField**
 - Áreas de texto - Classe **JTextArea**
 - Listas - Class **JList**
 - Botões - Classes **JButton**, **JCheckbox**
 - Scrollbars - Classe **JScrollbar**
 - Etc.

Interfaces gráficas

- Um gestor de posicionamento é um objecto que decide como distribuir os componentes num contentor
- Cada contentor tem um gestor de posicionamento por defeito
- Para alterar pode ser utilizado o método `setLayout (...)`
- Há vários gestores de posicionamento predefinidos:
 - `FlowLayout`
 - `BorderLayout`
 - `CardLayout`
 - `GridLayout`
 - `GridBagLayout`

Interfaces gráficas

- **FlowLayout**
 - Default para JPanel
 - Os componentes são colocados da esquerda para a direita, passando para a linha seguinte quando já não couber na linha corrente
 - Os componentes são centrados
- **BorderLayout**
 - Default para JApplet
 - Tem cinco locais para colocar componentes: North, South, West, East e Center
 - O programador especifica em qual das áreas quer cada componente
 - O tamanho relativo das áreas é definido pelo tamanho dos componentes que lá são colocados

Interfaces gráficas

- BorderLayout (cont.)



Interfaces gráficas

- **GridLayout**
 - Os componentes são colocados numa grelha com um determinado número de linhas e colunas
 - Cada componente é colocado numa célula
 - Todas as células têm a mesma dimensão
- **CardLayout**
 - Os componentes são colocados uns em cima dos outros
 - Apenas um componente está visível em cada momento
 - Os métodos podem controlar qual dos componentes é colocado visível
- **GridBagLayout**
 - Uma grelha bidimensional de linhas e colunas
 - Nem todas as células são do mesmo tamanho
 - Os componentes podem ocupar várias linhas ou colunas
 - Cada componente é associado com um conjunto de restrições

Interfaces gráficas

- Em resumo, para construir a interface gráfica:
 - Declarar os componentes e os seus listeners
 - Instanciar e inicializar componentes e listeners
 - Adicionar os componentes aos seus contentores (se houver)
 - Adicionar os componentes (e/ou contentores) à applet
 - Registar os listeners nos respectivos componentes
- Para tratar eventos:
 - Escrever métodos na applet para cada tipo de ação que o utilizador possa efectuar
 - Escrever uma classe (ou classes) para cada tipo de listener, incluindo os métodos respectivos

Interfaces gráficas

- Converter applets em aplicações:
 - Mudar o nome do método **init** para o nome da classe, tornando-o assim no seu construtor (eliminar void do cabeçalho)
 - Alterar o cabeçalho da classe (herda **JFrame** em vez de **JApplet**)
 - Criar um método **main** como:

```
public static void main (String[] args){  
    JFrame f = new NomeClasse ();  
    f.resize(300,300);  
    f.setVisible(true);  
}
```
 - Eliminar o import da classe Applet
 - Adicionar um método **handleEvent** para tratar o fim da aplicação:

```
public boolean handleEvent (Event e) {  
    if (e.id == Event.WINDOW_DESTROY)  
        System.exit(0);  
    return super.handleEvent(e);  
}
```

Interfaces gráficas

- Converter applets em aplicações (cont.)
 - Adicionar a invocação de um método para definir o gestor de posicionamento a usar. Tipicamente no construtor:
`setLayout(new FlowLayout());`
- Numa aplicação podem ser usados menus (ao contrário do que acontece nas applets)

Menus, imagens e sons

```
import java.awt.*;
import javax.swing.*;
import java.applet.*;
import java.net.*;

public class Janela extends JFrame{
    private final static int dimH = 400; // Dimensões da Janela
    private final static int dimV = 300;
    private JMenuItem m11, m12, m13; // Subopções do menu
    private JMenuItem opcao;
    private ImageIcon cao; // Imagens
    private ImageIcon ovelha;
    private ImageIcon vaca;
    private AudioClip som_cao; // Sons
    private AudioClip som_ovelha;
    private AudioClip som_vaca;
    private JButton botao;
```

Menus, imagens e sons

```
public Janela () {  
    JMenuBar m = new JMenuBar(); // Cria menu  
    JMenu m1 = new JMenu("Animais"); // Cria opção do menu  
    m.add (m1); // e junta-a ao menu  
    m11 = new JMenuItem("Cão"); // Cria subopções  
    m12 = new JMenuItem("Ovelha");  
    m13 = new JMenuItem("Vaca");  
    m1.add (m11); // e junta-as à opção do menu  
    m1.add (m12);  
    m1.add (m13);  
    setJMenuBar (m); // Associa menu à janela  
                      // Junta gestores de eventos às subopções do menu  
    m11.addActionListener(new GereEventos (this));  
    m12.addActionListener(new GereEventos (this));  
    m13.addActionListener(new GereEventos (this));  
  
    Container c = this.getContentPane(); // Obtém Content Pane da Janela  
    botao = new JButton(""); // Cria botão  
    botao.addActionListener(new GereEventos(this)); //Junta gestor  
    botao.setVisible(false); // Botão invísivel  
    c.add(botao); // Junta botão à Janela
```

Menus, imagens e sons

```
cao=new ImageIcon("dog.jpg");           // Carrega imagens
ovelha=new ImageIcon("sheep.jpg");
vaca=new ImageIcon("cow.jpg");

som_cao=Applet.newAudioClip(getURL("dog.au")); // Carrega sons
som_ovelha=Applet.newAudioClip(getURL("sheep.au"));
som_vaca=Applet.newAudioClip(getURL("cow.au"));

c.setLayout (new FlowLayout());           // Define Layout
setSize(dimH, dimV);                   // Define dimensões da Janela
setTitle("Quinta");                  // Define título da Janela
setVisible(true);                      // Mostra Janela
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Fecha-a quando termina aplicação
}

public static void main(String[] args){
    Janela frame = new Janela(); // Cria Janela
}
```

Menus, imagens e sons

```
public JMenuItem getcao() {  
    return m11; // Devolve opção 1 do menu  
}  
  
public JMenuItem getOvelha() {  
    return m12; // Devolve opção 2 do menu  
}  
  
public JButton getBotao() {  
    return botao; // Devolve botão  
}  
  
public void Botao(){  
    if (opcao==m11) som_cao.play(); // Toca som do animal selecionado  
    else if (opcao==m12) som_ovelha.play();  
    else som_vaca.play();  
}
```

Menus, imagens e sons

```
public void Cao() {  
    botao.setIcon(cao); // Define imagem e texto do botão  
    botao.setText("Sou um cão...");  
    botao.setVisible(true); // Mostra botão  
    opcao=m11; // animal selecionado  
}  
  
public void Ovelha() {  
    botao.setIcon(ovelha); // Define imagem e texto do botão  
    botao.setText("Sou uma ovelha...");  
    botao.setVisible(true); // Mostra botão  
    opcao=m12;  
}  
  
public void Vaca() {  
    botao.setIcon(vaca); // Define imagem e texto do botão  
    botao.setText("Sou uma vaca...");  
    botao.setVisible(true); // Mostra botão  
    opcao=m13;  
}
```

Menus, imagens e sons

```
public URL getURL (String nome) {  
    URL u = null;  
    try { // Obtém URL do ficheiro nome  
        u=new URL("file:"+System.getProperty("user.dir")+"\\"+ nome);  
    }  
    catch (MalformedURLException e) {  
    }  
    return u;  
}  
}
```

Menus, imagens e sons

```
import java.awt.event.*;
public class GereEventos implements ActionListener {

    private Janela j;

    public GereEventos (Janela j){
        this.j = j;
    }

    public void actionPerformed (ActionEvent e){
        Object o = e.getSource();
        if (o.equals(j.getBotao())) j.Botao();
        else if (o.equals(j.getCao())) j.Cao();
        else if (o.equals(j.getOvelha())) j.Ovelha();
        else j.Vaca();
    }
}
```